An Introduction to Machine Learning

Yao-Liang Yu
yaoliang.yu@uwaterloo.ca
School of Computer Science
University of Waterloo

December 30, 2021

Abstract

This is the lecture note for CS480/680.

Contents

-1 Optimization Basics

Goal

Convex sets and functions, gradient and Hessian, Fenchel conjugate, Lagrangian dual and gradient descent.

Alert -1.1: Convention

Gray boxes are not required hence can be omitted for unenthusiastic readers.

This note is likely to be updated again soon.

Definition -1.2: Vector space

Throughout the course, our universe is typically a (linear) vector space V over the real scalar field $\mathbb{R}$. On V we have two operators: addition $+ : \mathsf{V} \times \mathsf{V} \to \mathsf{V}$ and (scalar) multiplication $\cdot : \mathbb{R} \times \mathsf{V} \to \mathsf{V}$. Together they satisfy the following axioms:

- Addition commutativity: $\forall \mathbf{u}, \mathbf{v} \in \mathsf{V},\ \mathbf{u} + \mathbf{v} = \mathbf{v} + \mathbf{u}$;
- Addition associativity: $\forall \mathbf{u}, \mathbf{v}, \mathbf{w} \in \mathsf{V}, (\mathbf{u} + \mathbf{v}) + \mathbf{w} = \mathbf{u} + (\mathbf{v} + \mathbf{w})$;
- Identity of addition: $\exists \mathbf{0} \in \mathsf{V}$ such that $\forall \mathbf{v} \in \mathsf{V}, \mathbf{0} + \mathbf{v} = \mathbf{v}$;
- Inverse of addition: $\forall \mathbf{v} \in \mathsf{V}, \exists$ unique $\mathbf{u} \in \mathsf{V}$ such that $\mathbf{v} + \mathbf{u} = \mathbf{0}$, in which case we denote $\mathbf{u} = -\mathbf{v}$;
- Identity of multiplication: $\forall \mathbf{v} \in \mathsf{V}, 1 \cdot \mathbf{v} = \mathbf{v}$, where $1 \in \mathbb{R}$ is the usual constant 1;
- Compatibility: $\forall a, b \in \mathbb{R}, \mathbf{v} \in \mathsf{V}, a(b\mathbf{v}) = (ab)\mathbf{v}$;
- Distributivity over vector addition: $\forall a \in \mathbb{R}, \mathbf{u}, \mathbf{v} \in \mathsf{V}, a(\mathbf{u} + \mathbf{v}) = a\mathbf{u} + a\mathbf{v}$;
- Distributivity over scalar addition: $\forall a, b \in \mathbb{R}, \mathbf{v} \in \mathsf{V}, (a + b)\mathbf{v} = a\mathbf{v} + b\mathbf{v}$.

These axioms are so natural that you may question why do we bother to formalize them? Well, take two images/documents/graphs/speeches/DNAs/etc., how do you add them? multiply with scalars? inverse? Not so obvious... Representing objects as vectors in a vector space so that we can exploit linear algebra tools is arguably one of the most important lessons in ML/AI.

Example -1.3: Euclidean space

The most common vector space we are going to need is the d-dimensional Euclidean space $\mathbb{R}^d$. Each vector $\mathbf{v} \in \mathbb{R}^d$ can be identified with a d-tuple: $\mathbf{v} = (v_1, v_2, \ldots, v_d)$. The addition and multiplication operators are defined element-wise, and we can easily verify the axioms:

- Let $\mathbf{u} = (u_1, u_2, \ldots, u_d)$, then $\mathbf{u} + \mathbf{v} = (u_1 + v_1, u_2 + v_2, \ldots, u_d + v_d)$;
- Verify associativity yourself;
- $\mathbf{0} = (0, 0, \ldots, 0)$;
- $-\mathbf{v} = (-v_1, -v_2, , \ldots, -v_d)$;
- Obvious;
- $a\mathbf{v} = (av_1, av_2, \ldots, av_d)$;
- Verify distributivity yourself;

- Verify distributivity yourself.

In the perceptron lecture, we encode an email as a binary vector $\mathbf{x} \in \{0,1\}^d \subseteq \mathbb{R}^d$. This crude bag-of-words representation allowed us to use all linear algebra tools.

Definition -1.4: Convex set

A point set $C \subseteq \mathsf{V}$ is called convex if

$$\forall \mathbf{w}, \mathbf{z} \in C, \ [\mathbf{w}, \mathbf{z}] := \{\lambda \mathbf{w} + (1-\lambda)\mathbf{z} : \lambda \in [0,1]\} \subseteq C.$$

Elements in the "interval" $[\mathbf{w}, \mathbf{z}]$ are called convex combinations of $\mathbf{w}$ and $\mathbf{z}$.

Theorem -1.5: Intersection and union of convex sets

Arbitrary intersection and increasing union of convex sets are convex. $\square$

Thus, $\liminf_\alpha C_\alpha := \cup_\alpha \cap_{\beta \geq \alpha} C_\beta$ is convex. However, arbitrary union hence $\limsup_\alpha C_\alpha := \cap_\alpha \cup_{\beta \geq \alpha} C_\beta$ may not be convex.

Exercise -1.6: Hyperplane and halfspace

Verify the convexity of the hyperplane and halfspace:

$$\partial \mathsf{H}_{\mathbf{w},b} := \{\mathbf{x} \in \mathbb{R}^d : \langle \mathbf{x}, \mathbf{w} \rangle + b = 0\}$$
$$\mathsf{H}_{\mathbf{w},b} := \{\mathbf{x} \in \mathbb{R}^d : \langle \mathbf{x}, \mathbf{w} \rangle + b \leq 0\}$$

(The partial notation ∂ in front of a set means boundary.)

Exercise -1.7: Polyhedron

A polyhedron is some finite intersection of halfspaces:

$$P := \bigcap_{i=1,\ldots,n} \mathsf{H}_{\mathbf{w}_i, b_i} = \{\mathbf{x} \in \mathbb{R}^d : W\mathbf{x} + \mathbf{b} \leq \mathbf{0}\}.$$

Prove any polyhedron is convex.

If a polyhedron is bounded, then we call it a polytope. Prove the following:

- the unit ball of ℓ_1 norm $\{\mathbf{x} : \|\mathbf{x}\|_1 \leq 1\}$ is a polytope;
- the unit ball of ℓ_∞ norm $\{\mathbf{x} : \|\mathbf{x}\|_\infty \leq 1\}$is a polytope;
- the unit ball of ℓ_2 norm $\{\mathbf{x} : \|\mathbf{x}\|_2 \leq 1\}$ is not a polytope.

Theorem -1.8: Convex sets as intersection of halfspaces

Any closed convex set is intersection of halfspaces:

$$C = \bigcap_{i \in I} \mathsf{H}_{\mathbf{w}_i, b_i}$$

$\square$

The subtlety is that for non-polyhedra, the index set I is infinite. For instance:

$$\{\mathbf{w} : \|\mathbf{w}\|_2 \leq 1\} = \bigcap_{\mathbf{w}:\|\mathbf{w}\|_2=1} \mathsf{H}_{\mathbf{w},-1}.$$

Definition -1.9: Convex function (Jensen 1905)

The extended real-valued function $f : \mathsf{V} \to (-\infty, \infty]$ is called convex if Jensen's inequality holds:

$$\forall \mathbf{w}, \mathbf{z} \in \mathsf{V}, \forall \lambda \in (0,1),\ f(\lambda\mathbf{w} + (1-\lambda)\mathbf{z}) \leq \lambda f(\mathbf{w}) + (1-\lambda)f(\mathbf{z}). \tag{-1.1}$$

It is necessary that the (effective) domain of f, i.e. $\operatorname{dom} f := \{\mathbf{w} \in \mathsf{V} : f(\mathbf{w}) < \infty\}$, is a convex set.

We call f *strictly* convex iff the equality in (-1.1) holds only when $\mathbf{w} = \mathbf{z}$.

A function f is (strictly) concave iff $-f$ is (strictly) convex.

According to wikipedia, Jensen (Danish) never held any academic position and proved his mathematical results in his spare time.

Jensen, Johan Ludwig William Valdemar (1905). "Om konvekse Funktioner og Uligheder mellem Middelværdier". *Nyt Tidsskrift for Matematik B*, vol. 16, pp. 49–68.

Exercise -1.10: Affine = convex and concave

Prove that a function is both convex and concave iff it is affine (see Definition 1.13).

Remark -1.11: Convexity is remarkably important!

In the above definition of convexity, we have used the fact that V is a vector space, so that we can add vectors and multiply them with (real) scalars. It is quite remarkable that such a simple definition leads to a huge body of interesting results, a tiny part of which we shall be able to present below.

Definition -1.12: Epigraph and sub-level sets

The epigraph of a function f is defined as the set of points lying on or above its graph:

$$\operatorname{epi} f := \{(\mathbf{w}, t) \in \mathsf{V} \times \mathbb{R} : f(\mathbf{w}) \leq t\}.$$

It is clear that the epigraph of a function completely determines it:

$$f(\mathbf{w}) = \min\{t : (\mathbf{w}, t) \in \operatorname{epi} f\}.$$

So two functions are equal iff their epigraphs (sets) are the same. Again, we see that functions and sets are somewhat equivalent.

The sub-level sets of f are defined as:

$$\forall t \in \mathbb{R},\ L_t := [\![f \leq t]\!] := \{\mathbf{w} \in \mathsf{V} : f(\mathbf{w}) \leq t\}.$$

Sub-level sets also completely determine the function:

$$f(\mathbf{w}) = \min\{t : \mathbf{w} \in L_t\}.$$

Each sub-level set is clearly a section of the epigraph.

Exercise -1.13: Calculus for convexity

Prove the following:

- Any norm is convex;
- If f and g are convex, then for any $\alpha, \beta \geq 0$, $\alpha f + \beta g$ is also convex; (what about $-f$?)
- If $f : \mathbb{R}^d \to \mathbb{R}$ is convex, then so is $\mathbf{w} \mapsto f(A\mathbf{w} + \mathbf{b})$;
- If f_t is convex for all $t \in T$, then $f := \sup_{t \in T} f_t$ is convex;
- If $f(\mathbf{w}, t)$ is jointly convex in $\mathbf{w}$ and t, then $\mathbf{w} \mapsto \inf_{t \in T} f(\mathbf{w}, t)$ is convex;
- f is a convex function iff its epigraph $\operatorname{epi} f$ is a convex set;
- If $f : C \to \mathbb{R}$ is convex, then the perspective function $g(\mathbf{w}, t) := t f(\mathbf{w}/t)$ is convex on $C \times \mathbb{R}_{++}$;
- If $f(\mathbf{w}, \mathbf{z})$ is convex, then for any $\mathbf{w}$, $f_{\mathbf{w}} := f(\mathbf{w}, \cdot)$ is convex and similarly for $f^{\mathbf{z}} := f(\cdot, \mathbf{z})$. (what about the converse?)
- All sub-level sets of a convex function are convex, but a function with all sub-level sets being convex need not be convex (these are called quasi-convex functions).

Definition -1.14: Fenchel conjugate function

For any extended real-valued function $f : \mathsf{V} \to (-\infty, \infty]$ we define its Fenchel conjugate function as:

$$f^*(\mathbf{w}^*) := \sup_{\mathbf{w}} \langle \mathbf{w}, \mathbf{w}^* \rangle - f(\mathbf{w}).$$

According to one of the rules in Exercise -1.13, f^* is always a convex function (of $\mathbf{w}^*$).

If $\operatorname{dom} f$ is nonempty and closed, and f is continuous, then

$$f^{**} := (f^*)^* = f.$$

This remarkable property of convex functions will be used later in the course.

Theorem -1.15: Verifying convexity

Let $f : \mathbb{R}^d \to \mathbb{R}$ be twice differentiable. Then f is convex iff its Hessian $\nabla^2 f$ is always positive semidefinite. If the Hessian is always positive definite, then f is strictly convex. □

The function $f(x) = x^4$ is strictly convex but its Hessian vanishes at its minimum $x = 0$.

Fix any $\mathbf{w}$ and take any direction $\mathbf{d}$. Consider the univariate function $h(t) := f(\mathbf{w} + t\mathbf{d})$. We verify that $h''(t) = \langle \mathbf{d}, \nabla^2 f(\mathbf{w} + t\mathbf{d})\mathbf{d} \rangle \geq 0$. In other words, a convex function has increasing derivative along any direction $\mathbf{d}$ (starting from any point $\mathbf{w}$).

Exercise -1.16: Example convex functions

Prove the following functions are convex (Exercise -1.13 and Theorem -1.15 may be handy):

- affine functions: $f(\mathbf{w}) = \mathbf{w}^\top \mathbf{w} + b$;
- exponential function: $f(x) = \exp(x)$;
- entropy: $f(x) = x \log x$ with $x \geq 0$ and $0 \log 0 := 0$;
- log-sum-exp: $f(\mathbf{w}) = \log \sum_{j=1}^d \exp(x_j)$; (its gradient is the so-called `softmax`, to be discussed later)

(You may appreciate the epigraph more after the last exercise.)

Definition -1.17: Optimization problem

Consider a function $f : \mathbb{R}^d \to \mathbb{R}$, we are interested in the minimization problem:

$$\mathfrak{p}_* := \min_{\mathbf{w} \in C} \ f(\mathbf{w}), \tag{-1.2}$$

where $C \subseteq \mathbb{R}^d$ represents the constraints that $\mathbf{w}$ *must* satisfy.

Historically, the minimization problem (-1.2) was motivated by the need of solving nonlinear equations $h(\mathbf{w}) = 0$ (Cauchy 1847; Curry 1944), which can be reformulated as a least squares minimization problem $\min_{\mathbf{w}} h^2(\mathbf{w})$. As pointed out by Polyak (1963), the minimization problem itself has become ubiquitous, and often does not have to correspond to solving any nonlinear equation.

We remind that the minimum value $\mathfrak{p}_*$ is an extended real number in $[-\infty, \infty]$ (where $\mathfrak{p}_* = \infty$ iff $C = \emptyset$). When $\mathfrak{p}_*$ is finite, any feasible $\mathbf{w} \in C$ such that $f(\mathbf{w}) = \mathfrak{p}_*$ is called a minimizer. Minimum value always exists while minimizers may not!

Cauchy, M. Augustin-Louis (1847). "Méthode générale pour la résolution des systémes d'équations simultanées". *Comptes rendus hebdomadaires des séances de l'Académie des sciences*, vol. 25, no. 2, pp. 536–538.
Curry, Haskell B. (1944). "The Method of Steepest Descent for Non-linear Minimization Problems". *Quarterly of Applied Mathematics*, vol. 2, no. 3, pp. 258–261.
Polyak, Boris Teodorovich (1963). "Gradient methods for the minimization of functionals". *USSR Computational Mathematics and Mathematical Physics*, vol. 3, no. 4, pp. 643–653.

Definition -1.18: Level-bounded

A function $f : \mathbb{R}^d \to (-\infty, \infty]$ is said to be level bounded iff for all $t \in \mathbb{R}$, the sublevel set $[\![f \le t]\!]$ is bounded. Equivalently, f is level bounded iff $\|\mathbf{w}\| \to \infty \implies f(\mathbf{w}) \to \infty$.

Theorem -1.19: Existence of minimizer

A continuous and level-bounded function $f : \mathbb{R}^d \to (-\infty, \infty]$ *has a minimizer.* □

To appreciate this innocent theorem, let $f(x) = \exp(x)$:

- Is f continuous?
- Is f level-bounded?
- What is the minimum value of f? Does f has a minimizer on $\mathbb{R}$?

Definition -1.20: Extrema of an unconstrained function

Recall that $\mathbf{w}$ is a local minimizer of f if there exists an (open) neighborhood $\mathcal{N}$ of $\mathbf{w}$ so that for all $\mathbf{z} \in \mathcal{N}$ we have $f(\mathbf{w}) \le f(\mathbf{z})$. In case when $\mathcal{N}$ can be chosen to be the entire space $\mathbb{R}^d$, we say $\mathbf{w}$ is a global minimizer of f with the notation $\mathbf{w} \in \operatorname{argmin} f$.

By definition, a global minimizer is always a local minimizer while the converse is true only for a special class of functions (knowns as invex functions). The global minimizer may not be unique (take a constant function) but the global minimum value is.

The definition of a local (global) maximizer is analogous.

Exercise -1.21: Properties of extrema

Prove the following:

- $\mathbf{w}$ is a local (global) minimizer of f iff $\mathbf{w}$ is a local (global) maximizer of $-f$.
- $\mathbf{w}$ is a local (global) minimizer of f iff $\mathbf{w}$ is a local (global) minimizer of $\lambda f + c$ for any $\lambda > 0$ and $c \in \mathbb{R}$.
- If $\mathbf{w}$ is a local (global) minimizer (maximizer) of f, then it is a local (global) minimizer (maximizer) of $g(f)$ for any increasing function $g : \mathbb{R} \to \mathbb{R}$. What if g is *strictly* increasing?
- $\mathbf{w}$ is a local (global) minimizer (maximizer) of a positive function f iff it is a local (global) maximizer (minimizer) of $1/f$.
- $\mathbf{w}$ is a local (global) minimizer (maximizer) of a positive function f iff it is a local (global) minimizer (maximizer) of $\log f$.

Alert -1.22: The epigraph trick

Often, we rewrite the optimization problem

$$\min_{\mathbf{w} \in C} \; f(\mathbf{w})$$

as the equivalent one:

$$\min_{(\mathbf{w},t) \in \operatorname{epi} f \cap C \times \mathbb{R}} \; t, \tag{-1.3}$$

where the newly introduced variable t is jointly optimized with $\mathbf{w}$. The advantages of (-1.3) include:

- the objective in (-1.3) is a simple, canonical linear function $\langle (\mathbf{0}, 1), (\mathbf{w}, t) \rangle$;
- the constraints in (-1.3) may reveal more (optimization) insights, as we will see in SVMs.

Theorem -1.23: Local is global under convexity

Any local minimizer of a convex function (over some convex constraint set) is global.

Proof. Let $\mathbf{w}$ be a local minimizer of a convex function f. Suppose there exists $\mathbf{z}$ such that $f(\mathbf{z}) < f(\mathbf{w})$. Take convex combination and appeal to the definition of convexity in Definition -1.9:

$$\forall \lambda \in (0,1), \; f(\lambda \mathbf{w} + (1-\lambda)\mathbf{z}) \le \lambda f(\mathbf{w}) + (1-\lambda) f(\mathbf{z}) < f(\mathbf{w}),$$

contradicting to the local minimality of $\mathbf{w}$. □

In fact, in Theorem 2.22 we will see that any stationary point of a convex function is its global minimizer.

Theorem -1.24: Fermat's necessary condition for extrema

A necessary condition for $\mathbf{w}$ *to be a local minimizer of a differentiable function* $f : \mathbb{R}^d \to \mathbb{R}$ *is*

$$\nabla f(\mathbf{w}) = \mathbf{0}.$$

(Such points are called stationary, a.k.a. critical.) If f is convex, then the necessary condition is also

sufficient.

Proof. Suppose t is a local minimizer of a univariate function $g : \mathbb{R} \to \mathbb{R}$, then apply the definition of derivative we may easily deduce that $g'(t) \le 0$ and $g'(t) \ge 0$, i.e. $g'(t) = 0$.

Now if $\mathbf{w}$ is a local minimizer of $f : \mathbb{R}^d \to \mathbb{R}$, then $t = 0$ is a local minimizer of the univariate function $g(t) := f(\mathbf{w} + t\mathbf{w})$ for any $\mathbf{w}$. Apply the previous result for univariate functions and the chain rule:

$$g'(0) = \mathbf{w}^\top \nabla f(\mathbf{w}) = 0.$$

Since $\mathbf{w}$ was arbitrary, we must have $\nabla f(\mathbf{w}) = \mathbf{0}$.

If f is convex, then we have

$$\langle \nabla f(\mathbf{w}), \mathbf{z} - \mathbf{w} \rangle = \lim_{t \to 0^+} \frac{f(\mathbf{w} + t(\mathbf{z} - \mathbf{w})) - f(\mathbf{w})}{t} \le \frac{tf(\mathbf{z}) + (1-t)f(\mathbf{w})) - f(\mathbf{w})}{t} = f(\mathbf{z}) - f(\mathbf{w}).$$

In other words, we obtain the first order characterization of convexity:

$$\forall \mathbf{w}, \mathbf{z}, \quad f(\mathbf{z}) \ge f(\mathbf{w}) + \langle \nabla f(\mathbf{w}), \mathbf{z} - \mathbf{w} \rangle.$$

Plugging in $\nabla f(\mathbf{w}) = \mathbf{0}$ we see that $\mathbf{w}$ is in fact a global minimizer. □

Take $f(x) = x^3$ and $x = 0$ we see that this necessary condition is not sufficient for nonconvex functions. For local maximizers, we simply negate the function and apply the theorem to $-f$ instead.

Example -1.25: The difficulty of satisfying a constraint

Consider the trivial problem:

$$\min_{x \ge 1} \; x^2,$$

which admits a unique minimizer $x_\star = 1$. However, if we ignore the constraint and set the derivative to zero we would obtain $x = 0$, which does not satisfy the constraint!

We can apply Theorem 2.22 only when there is no constraint. We will introduce the Lagrangian to "remove" constraints below.

A common trick is to ignore any constraint and apply Theorem 2.22 anyway to derive a "bogus" minimizer $\mathbf{w}_\star$. Then, we verify if $\mathbf{w}_\star$ satisfies the constraint: If it does, then we actually find a minimizer for the *constrained* problem! For instance, hand the constraint above been $x \ge -1$ we would be fine by ignoring the constraint. Needless to say, this trick only works occasionally.

Remark -1.26: Iterative algorithm

The prevailing algorithms in machine learning are iterative in nature, i.e., we will construct a sequence $\mathbf{w}_0, \mathbf{w}_1, \ldots$, which will hopefully "converge" to something we contend with .

Definition -1.27: Projection to a closed set

Let $C \subseteq \mathbb{R}^d$ be a closed set. We define the (Euclidean) projection of a point $\mathbf{w} \in \mathbb{R}^d$ to C as:

$$\mathsf{P}_C(\mathbf{w}) := \operatorname*{argmin}_{\mathbf{z} \in C} \; \|\mathbf{z} - \mathbf{w}\|_2,$$

i.e., points in C that are closest to the given point $\mathbf{w}$. Needless to say, $\mathsf{P}_C(\mathbf{w}) = \mathbf{w}$ iff $\mathbf{w}$ lies in C.

The projection is always unique iff C is convex (Bunt 1934; Motzkin 1935). The only if part remains a long open problem when the space is infinite dimensional.

Bunt, L. N. H. (1934). "Bijdrage tot de theorie de convexe puntverzamelingen". PhD thesis. University of Groningen.

Motzkin, Theodore Samuel (1935). "Sur quelques propriétés caractéristiques des ensembles convexes". *Atti della Reale Accademia Nazionale dei Lincei*, vol. 21, no. 6, pp. 562–567.

Exercise -1.28: Projecting to nonnegative orthant

Let $C = \mathbb{R}^d_+$ be the nonnegative orthant. Find the formula for $\mathsf{P}_C(\mathbf{w})$. (Exercise -1.21 may be handy.)

Algorithm -1.29: Algorithm of feasible direction

We remind that line 4 below is possible because our universe is a vector space! This testifies again the (obvious?) importance of making machine learning amenable to linear algebra.

```
Algorithm: Algorithm of feasible direction
   Input: $\mathbf{w}_0 \in \operatorname{dom} f \cap C$
1  for $t = 0, 1, \ldots$ do
2      choose direction $\mathbf{d}_t$                // e.g. $\limsup_{t\to\infty} \langle \mathbf{d}_t, \nabla f(\mathbf{w}_t)\rangle > 0$
3      choose step size $\eta_t > 0$
4      $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta_t \mathbf{d}_t$                // update
5      $\mathbf{w}_{t+1} = \mathsf{P}_C(\mathbf{w}_{t+1})$                // optional projection step
```

Intuitively, Algorithm -1.29 first finds a direction $\mathbf{d}_t$, and then moves the iterate $\mathbf{w}_t$ along the direction. How far we move away from the current iterate $\mathbf{w}_t$ is determined by the step size (assuming $\mathbf{d}_t$ is normalized). To motivate the selection of the direction, apply Taylor's expansion:

$$f(\mathbf{w}_{t+1}) = f(\mathbf{w}_t - \eta_t \mathbf{d}_t) = f(\mathbf{w}_t) - \eta_t \langle \mathbf{d}_t, \nabla f(\mathbf{w}_t)\rangle + o(\eta_t),$$

where $o(\eta_t)$ is the small order term. Clearly, if $\langle \mathbf{d}_t, \nabla f(\mathbf{w}_t)\rangle > 0$ and η_t is small, then $f(\mathbf{w}_{t+1}) < f(\mathbf{w}_t)$, i.e. the algorithm is descending hence making progress. Typical choices for the direction include:

- Gradient Descent (GD): $\mathbf{d}_t = \nabla f(\mathbf{w}_t)$;
- Newton: $\mathbf{d}_t = [\nabla^2 f(\mathbf{w}_t)]^{-1} \nabla f(\mathbf{w}_t)$;
- Stochastic Gradient Descent (SGD): $\mathbf{d}_t = \xi_t$, $\mathsf{E}(\xi_t) = \nabla f(\mathbf{w}_t)$.

Remark -1.30: Randomness helps?

Note that if we start from a stationary point $\mathbf{w}_0$, i.e. $\nabla f(\mathbf{w}_0) = \mathbf{0}$, then gradient descent will not move, no matter what step size we choose! This is why such points are called stationary in the first place. On the other hand, stochastic gradient descent may still move because of the random noise added to it.

Remark -1.31: More on SGD

In machine learning we typically minimize some average loss over a training set $\mathcal{D} = \wr(\mathbf{w}_i, y_i)\wr$:

$$\min_{\mathbf{w}} \frac{1}{n}\sum_{i=1}^n \ell(\mathbf{w}; \mathbf{w}_i, y_i) \qquad =: \qquad \min_{\mathbf{w}} \hat{\mathsf{E}}\ell(\mathbf{w}; \mathbf{w}, y), \tag{-1.4}$$

where $(\mathbf{w}, y)$ is randomly chosen from the training set $\mathcal{D}$ and the empirical expectation is taken w.r.t. $\mathcal{D}$. Computing the gradient obviously costs $O(n)$ since we need to go through each sample in the training set

$\mathcal{D}$. On the other hand, if we take a random sample $(\mathbf{w}, y)$ from the training set, and compute

$$\xi = \nabla \ell(\mathbf{w}; \mathbf{w}, y).$$

Obviously, $\hat{\mathsf{E}}\xi$ equals the gradient but computing ξ is clearly much cheaper. In practice, one usually sample a mini-batch, and compute the (average) gradient over the mini-batch.

Exercise -1.32: Perceptron as SGD

Recall the perceptron update: if $y(\langle \mathbf{w}, \mathbf{w} \rangle + b) \leq 0$, then

$$\mathbf{w} \leftarrow \mathbf{w} + y\mathbf{w}, \; b \leftarrow b + y.$$

Construct a loss function $\ell(y(\langle \mathbf{w}, \mathbf{w} \rangle + b))$ in (-1.4) so that perceptron reduces to SGD (with step size say $\eta \equiv 1$).

Example -1.33: Descending alone does NOT guarantee convergence to stationary point

Consider the function

$$f = \begin{cases} \frac{3}{4}(1-x)^2 - 2(1-x), & x > 1 \\ \frac{3}{4}(1+x)^2 - 2(1+x), & x < -1 \\ x^2 - 1, & -1 \leq x \leq 1 \end{cases}, \text{ with gradient } f' = \begin{cases} \frac{3}{2}x + \frac{1}{2}, & x > 1 \\ \frac{3}{2}x - \frac{1}{2}, & x < -1 \\ 2x, & -1 \leq x \leq 1 \end{cases}.$$

Clearly, f is convex and has a unique minimizer $x^\star = 0$ (with $f^\star = -1$). It is easy to verify that

$$f(x) < f(y) \iff |x| < |y|.$$

Start with $x_0 > 1$, set the step size $\eta \equiv 1$, and choose $d = f'(x)$, then $x_1 = x_0 - f'(x_0) = -\frac{x_0+1}{2}$. By induction it is easy to show that $x_{t+1} = -\frac{1}{2}(x_t - (-1)^t)$. Thus, $x_t > 1$ if t is odd and $x_t < -1$ if t is even. Moreover, $|x_{t+1}| < |x_t|$, implying $f(x_{t+1}) < f(x_t)$. It is easy to show that $|x_t| \to 1$ and $f(x_t) \to 0$, hence the algorithm is not converging to a stationary point.

Remark -1.34: Step sizes

Let us mention a few ways to choose the step size η_t:

- Cauchy's rule (Cauchy 1847), where the existence of the minimizer is assumed:

$$\eta_t \in \operatorname*{argmin}_{\eta \geq 0} f(\mathbf{w}_t - \eta \mathbf{d}_t).$$

- Curry's rule (Curry 1944), where the finiteness of η_t is assumed:

$$\eta_t = \inf\{\eta \geq 0 : f'(\mathbf{w}_t - \eta \mathbf{d}_t) = 0\}.$$

- Constant rule: $\eta_t \equiv \eta > 0$.
- Summable rule: $\sum_t \eta_t = \infty, \sum_t \eta_t^2 < \infty$, e.g. $\eta_t = O(1/t)$;
- Diminishing rule: $\sum_t \eta_t = \infty, \lim_t \eta_t = 0$, e.g. $\eta_t = O(1/\sqrt{t})$.

The latter three are most common, especially for SGD.

Note that under the condition $\langle \mathbf{d}_t, \nabla f(\mathbf{w}_t) \rangle > 0$, the function $h(\eta) := f(\mathbf{w} - \eta \mathbf{d}_t)$ is decreasing for small positive η. Therefore, Curry's rule essentially selects the smallest local minimizer of h while Cauchy's rule selects the global minimizer of h. Needless to say, Cauchy's rule leads to larger per-step decrease of the function value but is also computationally more demanding. Under both Cauchy's and Curry's rule, we have the orthogonality property:

$$\langle \mathbf{d}_t, \nabla f(\mathbf{w}_{t+1})\rangle = 0,$$

i.e., the gradient at the next iterate $\mathbf{w}_{t+1}$ is orthogonal to the current direction vector $\mathbf{d}_t$. This explains the zigzag behaviour in gradient algorithms.

Cauchy, M. Augustin-Louis (1847). "Méthode générale pour la résolution des systémes d'équations simultanées". *Comptes rendus hebdomadaires des séances de l'Académie des sciences*, vol. 25, no. 2, pp. 536–538.

Curry, Haskell B. (1944). "The Method of Steepest Descent for Non-linear Minimization Problems". *Quarterly of Applied Mathematics*, vol. 2, no. 3, pp. 258–261.

Definition -1.35: L-smoothness

For twice differentiable functions f, define its smoothness parameter (a.k.a. the Lipschitz constant of the gradient):

$$L = \max_{\mathbf{w}} \|\nabla^2 f(\mathbf{w})\|_{\mathrm{sp}}$$

where the norm $\|\cdot\|_{\mathrm{sp}}$ is the spectral norm (i.e., the largest singular value, see Definition 2.13). For such functions, choosing $\eta \in (0, \frac{1}{L})$ will guarantee convergence of gradient descent. For convex functions, the step size can be enlarged to $\eta \in (0, \frac{2}{L})$.

With $\eta = 2/L$, gradient descent may not converge (although the function value still converges to some value): simply revisit Example -1.33.

Example -1.36: Iterates of gradient descent may NOT converge

While the function values $f(\mathbf{w}_t)$ of an iterative algorithm (such as gradient descent) usually converge, the iterate $\mathbf{w}_t$ itself may not:

- when there is no minimizer at all: $f(x) = \exp(x)$;
- when there is a unique minimizer but the function is non-differentiable: consider the convex function

$$f(x) = \begin{cases} e^{-x}, & x \le 0 \\ x+1, & x > 0 \end{cases};$$

- even when the function is smooth but there are many minimizers: see the intriguing "smoothed Mexican hat" function in (Absil et al. 2005, Fig 2.1).

Absil, P., R. Mahony, and B. Andrews (2005). "Convergence of the Iterates of Descent Methods for Analytic Cost Functions". *SIAM Journal on Optimization*, vol. 16, no. 2, pp. 531–547.

Example -1.37: Gradient descent may converge to saddle point

Consider the function $f(x, y) = \frac{1}{2}x^2 + \frac{1}{4}y^4 - \frac{1}{2}y^2$, whose gradient is $\nabla f(x, y) = \binom{x}{y^3-y}$ and Hessian is $\begin{bmatrix} 1 & 0 \\ 0 & 3y^2-1 \end{bmatrix}$. Clearly, there are three stationary points $(0, 0), (0, 1), (0, -1)$, where the the last two are global minimizers whereas the first is a saddle point. Take any $\mathbf{w}_0 = (t, 0)$, then $\mathbf{w}_t = (x_t, 0)$ hence it can only converge to $(0, 0)$, which is a saddle point.

Definition -1.38: Lagrangian Dual

Consider the canonical optimization problem:

$$\min_{\mathbf{w} \in C \subseteq \mathbb{R}^d} \ f(\mathbf{w}) \tag{-1.5}$$

$$\text{s.t. } \mathbf{g}(\mathbf{w}) \leq \mathbf{0}, \tag{-1.6}$$

$$\mathbf{h}(\mathbf{w}) = \mathbf{0}, \tag{-1.7}$$

where $f : \mathbb{R}^d \to \mathbb{R}$, $\mathbf{g} : \mathbb{R}^d \to \mathbb{R}^n$, and $\mathbf{h} : \mathbb{R}^d \to \mathbb{R}^m$. The set C is retained here to represent "simple" constraints that is more convenient to deal with directly than put into either (-1.6) or (-1.7). (Alternatively, one may always put $C = \mathbb{R}^d$.)

The (nonlinear) constraints (-1.6) are difficult to deal with. Fortunately, we can introduce the Lagrangian multipliers (a.k.a. dual variables) $\boldsymbol{\mu} \in \mathbb{R}^n_+$, $\boldsymbol{\nu} \in \mathbb{R}^m$ to move constraints into the Lagrangian:

$$L(\mathbf{w}; \boldsymbol{\mu}, \boldsymbol{\nu}) := f(\mathbf{w}) + \boldsymbol{\mu}^\top \mathbf{g}(\mathbf{w}) + \boldsymbol{\nu}^\top \mathbf{h}(\mathbf{w}).$$

We can now rewrite the original problem (-1.5) as the following fancy min-max problem:

$$\mathfrak{p}_\star := \min_{\mathbf{w} \in C} \max_{\boldsymbol{\mu} \geq \mathbf{0}, \boldsymbol{\nu}} \ L(\mathbf{w}; \boldsymbol{\mu}, \boldsymbol{\nu}). \tag{-1.8}$$

(Here $\mathfrak{p}$ stands for primal, as opposed to the dual below.) Indeed, if we choose some $\mathbf{w} \in C$ that does not satisfy either of the two constraints in (-1.6)-(-1.7), then there exist $\boldsymbol{\mu} \in \mathbb{R}^n_+$ and $\nu \in \mathbb{R}^m$ such that $L(\mathbf{w}; \boldsymbol{\mu}, \boldsymbol{\nu}) \to \infty$. Thus, in order to minimize w.r.t. $\mathbf{w}$, we are forced to choose $\mathbf{w} \in C$ to satisfy both constraints in (-1.6)-(-1.7), in which case $\max_{\boldsymbol{\mu} \geq \mathbf{0}, \boldsymbol{\nu}} L(\mathbf{w}; \boldsymbol{\mu}, \boldsymbol{\nu})$ simply reduces to $f(\mathbf{w})$ (by setting for instance $\boldsymbol{\mu} = \mathbf{0}, \boldsymbol{\nu} = \mathbf{0}$). In other words, the explicit constraints in (-1.5) have become implicit in (-1.8)!

The Lagrangian dual simply swaps the order of min and max:

$$\mathfrak{d}^\star := \max_{\boldsymbol{\mu} \geq \mathbf{0}, \boldsymbol{\nu}} \min_{\mathbf{w} \in C} \ L(\mathbf{w}; \boldsymbol{\mu}, \boldsymbol{\nu}). \tag{-1.9}$$

(Here $\mathfrak{d}$ stands for dual.) We emphasize the dimension of the dual variable $\boldsymbol{\mu}$ is the number of inequality constraints while that of $\boldsymbol{\nu}$ is the number of equality constraints; they are different from the dimension of the (primal) variable $\mathbf{w}$. Note also that there is no constraint on $\mathbf{w}$ in the dual problem (-1.9) (except the simple one $\mathbf{w} \in C$ which we do not count ☺), implicit or explicit!

In some sense, the Lagrangian dual is "the trick" that allows us to remove complicated constraints and apply Theorem 2.22.

Theorem -1.39: Weak duality

For any function $f : \mathsf{X} \times \mathsf{Y} \to \mathbb{R}$, *we have*

$$\min_{\mathbf{w}} \max_{\mathbf{y}} \ f(\mathbf{w}, \mathbf{y}) \quad \geq \quad \max_{\mathbf{y}} \min_{\mathbf{w}} \ f(\mathbf{w}, \mathbf{y})$$

Proof. Left as exercise. □

We immediately deduce that the Lagrangian dual (-1.9) is a lower bound of the original problem (-1.5), i.e. $\mathfrak{p}_\star \geq \mathfrak{d}^\star$. When equality is achieved, we say strong duality holds. For instance, if f, g_i for all $i = 1, \ldots, n$, and C are convex, $\mathbf{h}$ is affine, and some mild regularity condition holds (e.g. Slater's condition), then we have strong duality for the Lagrangian in Definition -1.38.

Exercise -1.40: Does the direction matter?

Derive the Lagrangian dual of the following problem:

$$\begin{aligned} \max_{\mathbf{w}\in C} \ & f(\mathbf{w}) \\ \text{s.t. } & \mathbf{g}(\mathbf{w}) \geq \mathbf{0} \\ & \mathbf{h}(\mathbf{w}) = \mathbf{0}. \end{aligned}$$

(Start with reducing to (-1.5) and then see if you can derive directly.)

Definition -1.41: KKT conditions (Karush 1939; Kuhn and Tucker 1951)

The inner minimization in the Lagrangian dual (-1.9) can usually be solved in closed-form:

$$\mathfrak{X}(\boldsymbol{\mu}, \boldsymbol{\nu}) := \operatorname*{argmin}_{\mathbf{w}\in C} \ L(\mathbf{w}; \boldsymbol{\mu}, \boldsymbol{\nu}). \tag{-1.10}$$

When $C = \mathbb{R}^d$, applying Theorem 2.22 we obtain the stationary condition (-1.12) below. Plugging any minimizer $\mathfrak{X}(\boldsymbol{\mu}, \boldsymbol{\nu})$ back into (-1.9) we obtain the dual problem:

$$\max_{\boldsymbol{\mu}\in\mathbb{R}^n_+, \boldsymbol{\nu}\in\mathbb{R}^m} \ L(\mathfrak{X}(\boldsymbol{\mu}, \boldsymbol{\nu}); \boldsymbol{\mu}, \boldsymbol{\nu}) \qquad \equiv \qquad -\min_{\boldsymbol{\mu}\in\mathbb{R}^n_+, \boldsymbol{\nu}\in\mathbb{R}^m} \ -L(\mathfrak{X}(\boldsymbol{\mu}, \boldsymbol{\nu}); \boldsymbol{\mu}, \boldsymbol{\nu}). \tag{-1.11}$$

Compared to the original problem (-1.5) that comes with complicated constraints (-1.6)-(-1.7), the dual problem (-1.11) has only very simple nonnegative constraint on $\boldsymbol{\mu}$. Thus, we may try to solve the Lagrangian dual (-1.11) instead! In fact, we have the following necessary conditions for $\mathbf{w}$ to minimize the original problem (-1.5) and for $\boldsymbol{\mu}, \boldsymbol{\nu}$ to maximize the dual problem (-1.11):

- primal feasibility:

$$\mathbf{g}(\mathbf{w}) \leq \mathbf{0}, \ h(\mathbf{w}) = \mathbf{0}, \ \mathbf{w} \in C;$$

- dual feasibility:

$$\boldsymbol{\mu} \geq \mathbf{0};$$

- stationarity: $\mathbf{w} = \mathfrak{X}(\boldsymbol{\mu}, \boldsymbol{\nu})$; for $C = \mathbb{R}^d$, we simply have

$$\nabla f(\mathbf{w}) + \sum_{i=1}^n \mu_i \nabla g_i(\mathbf{w}) + \sum_{j=1}^m \nu_j \nabla h_j(\mathbf{w}) = \mathbf{0}; \tag{-1.12}$$

- complementary slackness:

$$\langle \boldsymbol{\mu}, \mathbf{g}(\mathbf{w}) \rangle = 0.$$

Note that from primal and dual feasibility we always have

$$\forall i = 1, \ldots, n, \quad \mu_i g_i(\mathbf{w}) \leq 0.$$

Thus, complementary slackness actually implies equality in all n inequalities above.

When strong duality mentioned in Theorem -1.39 holds, the above KKT conditions are also sufficient!

Karush, W. (1939). "Minima of Functions of Several Variables with Inequalities as Side Constraints". MA thesis. University of Chicago.

Kuhn, H. W. and A. W. Tucker (1951). "Nonlinear programming". In: *Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability*, pp. 481–492.

Algorithm -1.42: Dual gradient descent (Uzawa 1958)

We can apply Algorithm -1.29 to the dual problem (-1.11), equipped with the projection onto nonnegative orthants in Exercise -1.28. After solving the dual, we may attempt to recover the primal through (-1.10). In both steps we sidestep any complicated constraints! However, one needs to always keep the following Alert -1.43 in mind.

Uzawa, Hirofumi (1958). "Iterative methods for concave programming". In: *Studies in linear and non-linear programming*. Ed. by Kenneth J. Arrow, Leonid Hurwicz, and Hirofumi Uzawa. Standford University Press, pp. 154–165.

Alert -1.43: Instability

A popular way to solve the primal problem (-1.5) is to solve the dual (-1.11) first. Then, with the optimal dual variable $(\boldsymbol{\mu}^\star, \boldsymbol{\nu}^\star)$ at hand, we "recover" the primal solution by

$$\mathfrak{X}(\boldsymbol{\mu}^\star, \boldsymbol{\nu}^\star) = \operatorname*{argmin}_{\mathbf{w}\in C} \; L(\mathbf{w}; \boldsymbol{\mu}^\star, \boldsymbol{\nu}^\star).$$

The minimizing set $\mathfrak{X}$ always contains all minimizers of the primal problem (-1.5). Thus, if $\mathfrak{X}$ is a singleton, everything is fine. Otherwise $\mathfrak{X}$ may actually contain points that are not minimizers of the primal problem! Fortunately, we need only verify primal feasibility in order to identify the true minimizers of the primal (-1.5) (when strong duality holds). The problem is, in practice, our algorithm only returns one "solution" from $\mathfrak{X}$, and if it fails primal feasibility, we may not be able to fetch a different "solution" from $\mathfrak{X}$.

Example -1.44: Instability

Let us consider the trivial problem:

$$\min_x 0, \;\; \text{s.t. } x \geq 0,$$

whose minimizers are $\mathbb{R}_+$. We derive the Lagrangian dual:

$$\max_{\mu\geq 0} \; \min_x -\mu x \quad = \quad \max_{\mu \geq 0} \begin{cases} 0, & \text{if } \mu = 0 \\ -\infty, & \text{o.w.} \end{cases}.$$

Clearly, we have $\mu^\star = 0$. Fixing $\mu^\star = 0$ and solving

$$\min_x -\mu^\star x$$

gives us $\mathfrak{X} = \mathbb{R}$ which strictly contains the primal solutions $\mathbb{R}_+$! Verifying primal feasibility $x \geq 0$ then identifies true minimizers.

Example -1.45: (Kernel) ridge regression

Recall ridge regression:

$$\min_{\mathbf{w},\mathbf{z}} \; \tfrac{1}{2}\|\mathbf{z}\|_2^2 + \tfrac{\lambda}{2}\|\mathbf{w}\|_2^2$$
$$\text{s.t. } X\mathbf{w} - \mathbf{y} = \mathbf{z},$$

where we *introduced* an "artificial" constraint (and variable). Derive the Lagrangian dual:

$$\max_{\boldsymbol{\alpha}} \min_{\mathbf{w},\mathbf{z}} \; \tfrac{1}{2}\|\mathbf{z}\|_2^2 + \tfrac{\lambda}{2}\|\mathbf{w}\|_2^2 + \boldsymbol{\alpha}^\top(X\mathbf{w} - \mathbf{y} - \mathbf{z}),$$

Applying Theorem 2.22 to the inner minimization problem:

$$\mathbf{w}_\star = -X^\top \boldsymbol{\alpha}/\lambda, \quad \mathbf{z}_\star = \boldsymbol{\alpha} \tag{-1.13}$$

Plugging it back in (and simplify) we obtain the dual:

$$\max_{\boldsymbol{\alpha}} -\tfrac{1}{2\lambda}\|X^\top \boldsymbol{\alpha}\|_2^2 - \boldsymbol{\alpha}^\top \mathbf{y} - \tfrac{1}{2}\|\boldsymbol{\alpha}\|_2^2$$

Applying Theorem 2.22 again we obtain:

$$\boldsymbol{\alpha}^\star = -(XX^\top/\lambda + I)^{-1}\mathbf{y}$$

and hence from (-1.13) we have

$$\mathbf{w}_\star = X^\top (XX^\top + \lambda I)^{-1}\mathbf{y}.$$

From Section 2 we know the solution of ridge regression is

$$\mathbf{w}_\star = (X^\top X + \lambda I)^{-1} X^\top \mathbf{y}$$

Verify the two solutions of $\mathbf{w}_\star$ are the same. (In fact, we have accidentally proved the Sherman-Morrison formula.) The purpose of this "linear algebra exercise" will become evident when we discuss reproducing kernels.

Algorithm -1.46: Gradient descent ascent (GDA)

When we cannot solve the inner minimization problem in the Lagrangian dual (-1.9), an alternative is to iteratively perform one gradient descent step on the inner minimization and then perform another gradient ascent step on the outer maximization. This idea can be traced back to (at least) (Brown and Neumann 1950; Arrow and Hurwicz 1958).

More generally, let us consider the min-max optimization problem:

$$\min_{\mathbf{w}\in \mathsf{X}} \max_{\mathbf{y}\in \mathsf{Y}} f(\mathbf{w}, \mathbf{y}).$$

```
Algorithm: Gradient descent ascent for min-max
Input: $(\mathbf{w}_0, \mathbf{y}_0) \in \operatorname{dom} f \cap \mathsf{X}\times\mathsf{Y}$
1 $s_{-1} = 0, (\bar{\mathbf{w}}_{-1}, \bar{\mathbf{y}}_{-1}) = (\mathbf{0}, \mathbf{0})$                    // optional
2 for $t = 0, 1, \ldots$ do
3     choose step size $\eta_t > 0$
4     $\mathbf{w}_{t+1} = \mathsf{P}_\mathsf{X}[\mathbf{w}_t - \eta_t \nabla_\mathbf{w} f(\mathbf{w}_t, \mathbf{y}_t)]$          // GD on minimization
5     $\mathbf{y}_{t+1} = \mathsf{P}_\mathsf{Y}[\mathbf{y}_t + \eta_t \nabla_\mathbf{y} f(\mathbf{w}_t, \mathbf{y}_t)]$          // GA on maximization
6     $s_t = s_{t-1} + \eta_t$
7     $(\bar{\mathbf{w}}_t, \bar{\mathbf{y}}_t) = \frac{s_{t-1}(\bar{\mathbf{w}}_{t-1}, \bar{\mathbf{y}}_{t-1}) + \eta_t(\mathbf{w}_t, \mathbf{y}_t)}{s_t}$    // averaging: $(\bar{\mathbf{w}}_t, \bar{\mathbf{y}}_t) = \sum_{k=1}^t \eta_k(\mathbf{w}_k, \mathbf{y}_k)/\sum_k \eta_k$
```

Variations of Algorithm -1.46 include (but are not limited to):

- use different step sizes on $\mathbf{w}$ and $\mathbf{y}$;
- use $\mathbf{w}_{t+1}$ in the update on $\mathbf{y}$ (or vice versa);
- use stochastic gradients in both steps;
- after every update in $\mathbf{w}$, perform k updates in $\mathbf{y}$ (or vice versa);

Brown, G. W. and J. von Neumann (1950). "Solutions of Games by Differential Equations". In: *Contributions to the Theory of Games I*. Ed. by H. W. Kuhn and A. W. Tucker. Princeton University Press, pp. 73–79.

Arrow, Kenneth J. and Leonid Hurwicz (1958). "Gradient method for concave programming I: Local results". In: *Studies in linear and non-linear programming*. Ed. by Kenneth J. Arrow, Leonid Hurwicz, and Hirofumi Uzawa. Standford University Press, pp. 117–126.

Example -1.47: Vanilla GDA may never converge for any step size

Let us consider the following simple problem:

$$\min_{x\in[-1,1]} \max_{y\in[-1,1]} xy \qquad \equiv \qquad \max_{y\in[-1,1]} \min_{x\in[-1,1]} xy.$$

The left-hand side is equivalent as $\min_{x\in[-1,1]} |x|$ hence with unique minimizer $x^* = 0$. Similarly, the right-hand side is equivalent as $\max_{y\in[-1,1]} -|y|$ hence with unique maximizer $y^* = 0$. It follows that the optimal saddle-point (equilibrium) is $x^* = y^* = 0$.

If we run vanilla (projected) GDA with step size $\eta_t \geq 0$, then

$$\begin{aligned} x_{t+1} &= [x_t - \eta_t y_t]_{-1}^{1} \\ y_{t+1} &= [y_t + \eta_t x_t]_{-1}^{1}, \end{aligned}$$

where $[z]_{-1}^{1} := (z \wedge 1) \vee (-1)$ is the projection of z onto the interval $[-1, 1]$. Thus, we have

$$x_{t+1}^2 + y_{t+1}^2 \geq 1 \wedge [(x_t - \eta_t y_t)^2 + (y_t + \eta_t x_t)^2] = 1 \wedge [(1+\eta_t^2)(x_t^2 + y_t^2)] \geq 1 \wedge (x_t^2 + y_t^2).$$

Therefore, if we do *not* initialize at the equilibrium $x^* = y^* = 0$, then the norm of (x_t, y_t) will always be lower bounded by $1 \wedge \|\binom{x_0}{y_0}\| > 0 = \|\binom{x^*}{y^*}\|$. In other words, (x_t, y_t) will not converge to (x^*, y^*).

In fact, we can generalize the above failure to any bilinear matrix game:

$$\min_{\mathbf{w}\in C} \max_{\mathbf{y}\in D} \mathbf{w}^\top A\mathbf{y},$$

where C and D are closed sets and $A \in \mathbb{R}^{d\times d}$ is nonsingular. Suppose there exists $\epsilon > 0$ so that any equilibrium $(\mathbf{w}^*, \mathbf{y}^*)$ is ϵ away from the boundary: $\mathrm{dist}(\mathbf{w}^*, \partial C) \wedge \mathrm{dist}(\mathbf{y}^*, \partial D) > \epsilon$, then we know $A\mathbf{y}^* = \mathbf{0}$ hence $\mathbf{y}^* = \mathbf{0}$ and similarly $\mathbf{w}^* = \mathbf{0}$. It follows that vanilla projected gradient

$$\begin{aligned} \mathbf{w}_{t+1} &= \mathsf{P}_C(\mathbf{w}_t - \eta_t A\mathbf{y}_t) \\ \mathbf{y}_{t+1} &= \mathsf{P}_D(\mathbf{y}_t + \eta_t A^\top \mathbf{w}_t) \end{aligned}$$

will not converge to any equilibrium. Indeed, for vanilla gradient to converge, the projection will eventually be vacuous (otherwise we are ϵ away), but then for the equilibrium $(\mathbf{w}^* = \mathbf{0}, \mathbf{y}^* = \mathbf{0})$:

$$\|\mathbf{w}_{t+1}\|_2^2 + \|\mathbf{y}_{t+1}\|_2^2 = \|\mathbf{w}_t\|_2^2 + \|\mathbf{y}_t\|^2 + \eta_t^2 \left\| \begin{bmatrix} \mathbf{0} & A^\top \\ A & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{y}_t \\ \mathbf{w}_t \end{bmatrix} \right\|_2^2 = (1 + \eta_t^2 \sigma_{\min}^2(A))(\|\mathbf{w}_t\|_2^2 + \|\mathbf{y}_t\|^2),$$

which is strictly lower bounded by $\|\mathbf{w}_0\|_2^2 + \|\mathbf{y}_0\|_2^2$ if the starting point $(\mathbf{w}_0, \mathbf{y}_0)$ does not coincide with the equilibrium.

Algorithm -1.48: Alternating

The following alternating algorithm is often applied in practice for solving the joint minimization problem:

$$\min_{\mathbf{w}\in\mathsf{X}} \min_{\mathbf{y}\in\mathsf{Y}} f(\mathbf{w}, \mathbf{y}).$$

Algorithm: Alternating minimization for min-min

```
   Input: $(\mathbf{w}_0, \mathbf{y}_0) \in \operatorname{dom} f \cap \mathsf{X} \times \mathsf{Y}$
1  for $t = 0, 1, \ldots$ do
2      $\mathbf{w}_{t+1} = \operatorname{argmin}_{\mathbf{w} \in \mathsf{X}} f(\mathbf{w}, \mathbf{y}_t)$        // exact minimization
3      $\mathbf{y}_{t+1} = \operatorname{argmin}_{\mathbf{y} \in \mathsf{Y}} f(\mathbf{w}_{t+1}, \mathbf{y})$    // exact maximization
```

It is tempting to adapt alternating to solve min-max problems. The resulting algorithm, when compared to dual gradient (see Algorithm -1.42), is more aggressive in optimizing $\mathbf{y}$: dual gradient only takes a gradient ascent step while the latter finds the exact maximizer. Surprisingly, being more aggressive (and spending more effort) here actually hurts (sometimes).

Example -1.49: When to expect alternating to work?

Let us consider the simple problem

$$\min_{x \in [-1,1]} \min_{y \in [-1,1]} xy,$$

where two minimizers $\{(1, -1), (-1, 1)\}$ exist. Let us apply the alternating Algorithm -1.48. Choose any $x > 0$, we obtain

$$y \leftarrow -1, x \leftarrow 1, y \leftarrow -1,$$

which converges to an optimal solution after 1 iteration!

To compare, let us consider the "twin" problem:

$$\min_{x \in [-1,1]} \max_{y \in [-1,1]} xy,$$

where a unique equilibrium $(x^*, y^*) = (0, 0)$ exists. Choose any $x > 0$ (the analysis for $x < 0$ is similar). Let us perform the inner maximization exactly to obtain $y = 1$. With $y = 1$ in mind, we perform the outer inner minimization exactly to obtain $x = -1$. Keep iterating, we obtain the cycle

$$y \leftarrow -1, x \leftarrow 1, y \leftarrow 1, x \leftarrow -1, y \leftarrow -1,$$

which is bounded away from the equilibrium! Of course, if we perform averaging along the trajectory we again obtain convergence. Dual gradient essentially avoids the oscillating behaviour of alternating by (implicitly) averaging.

0 Statistical Learning Basics

Goal

Maximum Likelihood, Prior, Posterior, MAP, Bayesian LR

Alert 0.1: Convention

Gray boxes are not required hence can be omitted for unenthusiastic readers.

This note is likely to be updated again soon.

Definition 0.2: Distribution and density

Recall that the cumulative distribution function (cdf) of a random vector $\mathbf{X} \in \mathbb{R}^d$ is defined as:

$$F(\mathbf{x}) := \Pr(\mathbf{X} \leq \mathbf{x}),$$

and its probability density function (pdf) is

$$p(\mathbf{x}) := \frac{\partial^d F}{\partial x_1 \cdots \partial x_d}(\mathbf{x}), \text{ or equivalently } F(\mathbf{x}) = \int_{-\infty}^{x_1} \cdots \int_{-\infty}^{x_d} p(\mathbf{x}) \, \mathrm{d}\mathbf{x}.$$

Clearly, each cdf $F : \mathbb{R}^d \to [0, 1]$ is

- monotonically increasing in each of its inputs;
- right continuous in each of its inputs;
- $\lim_{\mathbf{x} \to \infty} F(\mathbf{x}) = 1$ and $\lim_{\mathbf{x} \to -\infty} F(\mathbf{x}) = 0$.

On the other hand, each pdf $p : \mathbb{R}^d \to \mathbb{R}_+$

- integrates to 1, i.e. $\int_{-\infty}^{\infty} p(\mathbf{x}) \, \mathrm{d}\mathbf{x} = 1$.

(The cdf and pdf of a discrete random variable can be defined similarly and is omitted.)

Remark 0.3: Change-of-variable

Let $\mathsf{T} : \mathbb{R}^d \to \mathbb{R}^d$ be a diffeomorphism (differentiable bijection with differentiable inverse). Let $\mathbf{X} = \mathsf{T}(\mathbf{Z})$, then we have the change-of-variable formula for the pdfs:

$$p(\mathbf{x}) \, \mathrm{d}\mathbf{x} \approx q(\mathbf{z}) \, \mathrm{d}\mathbf{z}, \ \textit{i.e.} \ p(\mathbf{x}) = q(\mathsf{T}^{-1}(\mathbf{x})) \left| \det \frac{\mathrm{d}\mathsf{T}^{-1}}{\mathrm{d}\mathbf{x}}(\mathbf{x}) \right|$$

$$q(\mathbf{z}) = p(\mathsf{T}(\mathbf{z})) \left| \det \frac{\mathrm{d}\mathsf{T}}{\mathrm{d}\mathbf{z}}(\mathbf{z}) \right|,$$

where det denotes the determinant.

Definition 0.4: Marginal, conditional, and independence

Let $\mathbf{X} = (\mathbf{X}_1, \mathbf{X}_2)$ be a random vector with pdf $p(\mathbf{x}) = p(\mathbf{x}_1, \mathbf{x}_2)$. We say $\mathbf{X}_1$ is a marginal of $\mathbf{X}$ with pdf

$$p_1(\mathbf{x}_1) = \int_{-\infty}^{\infty} p(\mathbf{x}_1, \mathbf{x}_2) \, \mathrm{d}\mathbf{x}_2,$$

where we marginalize over $\mathbf{X}_2$ by integrating it out. Similarly $\mathbf{X}_2$ is a marginal of $\mathbf{X}$ with pdf

$$p_2(\mathbf{x}_2) = \int_{-\infty}^{\infty} p(\mathbf{x}_1, \mathbf{x}_2)\, \mathrm{d}\mathbf{x}_1.$$

We then define the conditional $\mathbf{X}_1|\mathbf{X}_2$ with density:

$$p_{1|2}(\mathbf{x}_1|\mathbf{x}_2) = p(\mathbf{x}_1, \mathbf{x}_2)/p_2(\mathbf{x}_2),$$

where the value of $p_{1|2}$ is arbitrary if $p_2(\mathbf{x}_2) = 0$ (usually immaterial). Similarly we may define the conditional $\mathbf{X}_2|\mathbf{X}_1$. It is obvious from our definition that

$$p(\mathbf{x}_1, \mathbf{x}_2) = p_1(\mathbf{x}_1)p_{2|1}(\mathbf{x}_2|\mathbf{x}_1) = p_2(\mathbf{x}_2)p_{1|2}(\mathbf{x}_1|\mathbf{x}_2),$$

namely the joint density p can be factorized into the product of marginal p_1 and conditional $p_{2|1}$. Usually, we omit all subscripts in p when referring to the marginal or conditional whenever the meaning is obvious from context.

Iterating the above construction, we obtain the famous chain rule:

$$p(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_d) = \prod_{j=1}^{d} p(\mathbf{x}_j|\mathbf{x}_1, \ldots, \mathbf{x}_{j-1}),$$

with obviously $p(\mathbf{x}_1|\mathbf{x}_1, \ldots, \mathbf{x}_0) := p(\mathbf{x}_1)$. We say that the random vectors $\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_d$ are independent if

$$p(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_d) = \prod_{j=1}^{d} p(\mathbf{x}_j).$$

All of our constructions above can be done with cdfs as well (with serious complication for the conditional though). In particular, we have the Bayes rule:

$$\Pr(A|B) = \frac{\Pr(A, B)}{\Pr(B)} = \frac{\Pr(B|A)\Pr(A)}{\Pr(B, A) + \Pr(B, \neg A)}.$$

Definition 0.5: Mean, variance and covariance

Let $\mathbf{X} = (X_1, \ldots, X_d)$ be a random (column) vector. We define its mean (vector) as

$$\boldsymbol{\mu} = \mathsf{E}\mathbf{X}, \quad \text{where} \quad \mu_j = \int x_j \cdot p(x_j)\, \mathrm{d}x_j$$

and its covariance (matrix) as

$$\Sigma = \mathsf{E}(\mathbf{X} - \boldsymbol{\mu})(\mathbf{X} - \boldsymbol{\mu})^\top, \quad \text{where} \quad \Sigma_{ij} = \int (x_i - \mu_i)(x_j - \mu_j) \cdot p(x_i, x_j)\, \mathrm{d}x_i\, \mathrm{d}x_j.$$

By definition Σ is symmetric $\Sigma_{ij} = \Sigma_{ji}$ and positive semidefinite (all eigenvalues are nonnegative). The j-th diagonal entry of the covariance $\sigma_j^2 := \Sigma_{jj}$ is called the variance of X_j.

Exercise 0.6: Covariance

Prove the following equivalent formula for the covariance:

- $\Sigma = \mathsf{E}\mathbf{X}\mathbf{X}^\top - \boldsymbol{\mu}\boldsymbol{\mu}^\top$;

- $\Sigma = \frac{1}{2}\mathsf{E}(\mathbf{X}-\mathbf{X}')(\mathbf{X}-\mathbf{X}')^\top$, where $\mathbf{X}'$ is iid (independent and identically distributed) with $\mathbf{X}$.

Suppose $\mathbf{X}$ has mean $\boldsymbol{\mu}$ and covariance Σ. Find the mean and covariance of $A\mathbf{X}+\mathbf{b}$, where $A, \mathbf{b}$ are deterministic.

Example 0.7: Multivariate Gaussian

The pdf of the multivariate Gaussian distribution (a.k.a. normal distribution) is:

$$p(\mathbf{x}) = (2\pi)^{-d/2}[\det(\Sigma)]^{-1/2}\exp\left(-\tfrac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})\right),$$

where d is the dimension and det denotes the determinant of a matrix. We typically use the notation $\mathbf{X}\sim\mathcal{N}(\boldsymbol{\mu},\Sigma)$, where $\boldsymbol{\mu} = \mathsf{E}\mathbf{X}$ is its mean and $\Sigma = \mathsf{E}(\mathbf{X}-\boldsymbol{\mu})(\mathbf{X}-\boldsymbol{\mu})^\top$ is its covariance.

An important property of the multivariate Gaussian distribution is its equivariance under affine transformations:

$$\mathbf{X}\sim\mathcal{N}(\boldsymbol{\mu},\Sigma) \implies A\mathbf{X}+\mathbf{b}\sim\mathcal{N}(A\boldsymbol{\mu}+\mathbf{b}, A\Sigma A^\top).$$

(This property actually characterizes the multivariate Gaussian distribution.)

Exercise 0.8: Marginal and conditional of multivariate Gaussian

Let $\begin{bmatrix}\mathbf{X}_1\\ \mathbf{X}_2\end{bmatrix}\sim\mathcal{N}\left(\begin{bmatrix}\boldsymbol{\mu}_1\\ \boldsymbol{\mu}_2\end{bmatrix},\begin{bmatrix}\Sigma_{11} & \Sigma_{12}\\ \Sigma_{21} & \Sigma_{22}\end{bmatrix}\right)$. Prove the following results:

$$\mathbf{X}_1\sim\mathcal{N}(\boldsymbol{\mu}_1,\Sigma_{11}),\quad \mathbf{X}_2|\mathbf{X}_1\sim\mathcal{N}(\boldsymbol{\mu}_2+\Sigma_{21}\Sigma_{11}^{-1}(\mathbf{X}_1-\boldsymbol{\mu}_1),\Sigma_{22}-\Sigma_{21}\Sigma_{11}^{-1}\Sigma_{12});$$
$$\mathbf{X}_2\sim\mathcal{N}(\boldsymbol{\mu}_2,\Sigma_{22}),\quad \mathbf{X}_1|\mathbf{X}_2\sim\mathcal{N}(\boldsymbol{\mu}_1+\Sigma_{12}\Sigma_{22}^{-1}(\mathbf{X}_2-\boldsymbol{\mu}_2),\Sigma_{11}-\Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}).$$

Remark 0.9: Bias-variance trade-off

Suppose we are interested in predicting a random (scalar) quantity Y based on some feature vector (a.k.a. covariate) $\mathbf{X}$, using the function $\hat f$. Here the hat notation suggests $\hat f$ may depend on other random quantities, such as samples from a training set. Often, we use the squared loss to evaluate our prediction:

$$\begin{aligned}\mathsf{E}(\hat f(\mathbf{X})-Y)^2 &= \mathsf{E}\Big(\hat f(\mathbf{X})-\mathsf{E}\hat f(\mathbf{X})+\mathsf{E}\hat f(\mathbf{X})-\mathsf{E}(Y|\mathbf{X})+\mathsf{E}(Y|\mathbf{X})-Y\Big)^2\\ &= \underbrace{\mathsf{E}\big(\hat f(\mathbf{X})-\mathsf{E}\hat f(\mathbf{X})\big)^2}_{\text{variance}}+\underbrace{\mathsf{E}\big(\mathsf{E}\hat f(\mathbf{X})-\mathsf{E}(Y|\mathbf{X})\big)^2}_{\text{bias}^2}+\underbrace{\mathsf{E}\big(\mathsf{E}(Y|\mathbf{X})-Y\big)^2}_{\text{difficulty}},\end{aligned}$$

where recall that $\mathsf{E}(Y|\mathbf{X})$ is the so-called regression function. The last term indicates the difficulty of our problem and cannot be reduced by our choice of $\hat f$. The first two terms reveals an inherent trade-off in designing $\hat f$:

- the variance term reflects the fluctuation incurred by training on some random training set. Typically, a less flexible $\hat f$ will incur a smaller variance (e.g. constant functions have 0 variance);
- the (squared) bias term reflects the mismatch of our choice of $\hat f$ and the optimal regression function. Typically, a very flexible $\hat f$ will incur a smaller bias (e.g. when $\hat f$ can model *any* function).

The major goal of much of ML is to strike an appropriate balance between the first two terms.

Definition 0.10: Maximum likelihood estimation(MLE)

Suppose we have a dataset $\mathcal{D} = \lbag \mathbf{x}_1, \ldots, \mathbf{x}_n \rbag$, where each sample $\mathbf{x}_i$ (is assumed to) follow some pdf $p(\mathbf{x}|\theta)$ with *unknown* parameter θ. We define the likelihood of a parameter θ given the dataset $\mathcal{D}$ as:

$$L(\theta) = L(\theta; \mathcal{D}) := p(\mathcal{D}|\theta) = \prod_{i=1}^{n} p(\mathbf{x}_i|\theta),$$

where in the last equality we assume our data is iid. A popular way to find an estimate of the parameter θ is to maximize the likelihood over some parameter space Θ:

$$\theta_{\mathsf{MLE}} := \mathrm{argmax}_{\theta \in \Theta} \ L(\theta).$$

Equivalently, by taking the log and negating, we minimize the negative log-likelihood (NLL):

$$\theta_{\mathsf{MLE}} := \mathrm{argmin}_{\theta \in \Theta} \sum_{i=1}^{n} - \log \ p(\mathbf{x}_i|\theta).$$

We remark that MLE is applicable only when we can evaluate the likelihood function efficiently, which turns out to be not the case in many settings and we will study alternative algorithms.

Example 0.11: Sample mean and covariance as MLE

Let $\mathbf{x}_1, \ldots, \mathbf{x}_n$ be iid samples from the multivariate Gaussian distribution $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$ where the parameters $\boldsymbol{\mu}$ and Σ are to be found. We apply maximum likelihood:

$$\hat{\boldsymbol{\mu}}_{\mathsf{MLE}} := \operatorname*{argmin}_{\boldsymbol{\mu}} \ \frac{1}{2} \sum_{i=1}^{n} (\mathbf{x}_i - \boldsymbol{\mu})^\top \Sigma^{-1} (\mathbf{x}_i - \boldsymbol{\mu}).$$

Applying Theorem 2.22 we obtain the sample mean:

$$\hat{\boldsymbol{\mu}}_{\mathsf{MLE}} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i =: \hat{\mathsf{E}}\mathbf{x},$$

where the hat expectation $\hat{\mathsf{E}}$ is w.r.t. the given data.

Similarly we can show

$$\hat{\Sigma}_{\mathsf{MLE}} := \operatorname*{argmin}_{\Sigma} \ \log\det\Sigma + \sum_{i=1}^{n} (\mathbf{x}_i - \boldsymbol{\mu})^\top \Sigma^{-1} (\mathbf{x}_i - \boldsymbol{\mu}).$$

Or equivalently

$$\hat{\Sigma}_{\mathsf{MLE}}^{-1} := \operatorname*{argmin}_{S} \ -\log\det S + \sum_{i=1}^{n} (\mathbf{x}_i - \boldsymbol{\mu})^\top S (\mathbf{x}_i - \boldsymbol{\mu}).$$

Applying Theorem 2.22 (with the fact that the gradient of $\log\det S$ is S^{-1}), we obtain:

$$\hat{\Sigma}_{\mathsf{MLE}} = \frac{1}{n} \sum_{i=1}^{n} (\mathbf{x}_i - \boldsymbol{\mu})(\mathbf{x}_i - \boldsymbol{\mu})^\top = \hat{\mathsf{E}}\mathbf{x}\mathbf{x}^\top - (\hat{\mathsf{E}}\mathbf{x})(\hat{\mathsf{E}}\mathbf{x})^\top,$$

where we plug in the ML estimate $\hat{\boldsymbol{\mu}}_{\mathsf{MLE}}$ of $\boldsymbol{\mu}$ if it is not known.

Exercise 0.12: Bias and variance of sample mean and covariance

Calculate the following bias and variance:

$$\begin{aligned} \mathsf{E}[\boldsymbol{\mu} - \hat{\boldsymbol{\mu}}_{\mathsf{MLE}}] &= \\ \mathsf{E}[\boldsymbol{\mu} - \hat{\boldsymbol{\mu}}_{\mathsf{MLE}}][\boldsymbol{\mu} - \hat{\boldsymbol{\mu}}_{\mathsf{MLE}}]^\top &= \\ \mathsf{E}[\Sigma - \hat{\Sigma}_{\mathsf{MLE}}] &= \end{aligned}$$

Definition 0.13: f-divergence (Csiszár 1963; Ali and Silvey 1966)

Let $f : \mathbb{R}_+ \to \mathbb{R}$ be a strictly convex function (see Definition -1.9) with $f(1) = 0$. We define the following f-divergence to measure the closeness of two pdfs p and q:

$$\mathsf{D}_f(p\|q) := \int f\big(p(\mathbf{x})/q(\mathbf{x})\big) \cdot q(\mathbf{x})\,\mathrm{d}\mathbf{x},$$

where we assume $q(\mathbf{x}) = 0 \implies p(\mathbf{x}) = 0$ (otherwise we put the divergence to ∞).

Csiszár, Imre (1963). "Eine informationstheoretische Ungleichung und ihre Anwendung auf den Beweis der Ergodizität von Markoffschen Ketten". *A Magyar Tudományos Akadémia Matematikai Kutató Intézetének közleményei*, vol. 8, pp. 85–108.

Ali, S. M. and S. D. Silvey (1966). "A General Class of Coefficients of Divergence of One Distribution from Another". *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 28, no. 1, pp. 131–142.

Exercise 0.14: Properties of f-divergence

Prove the following:

- $\mathsf{D}_f(p\|q) \geq 0$, with 0 attained iff $p = q$;
- $\mathsf{D}_{f+g} = \mathsf{D}_f + \mathsf{D}_g$ and $\mathsf{D}_{sf} = s\mathsf{D}_f$ for $s > 0$;
- Let $g(t) = f(t) + s(t-1)$ for any s. Then, $\mathsf{D}_g = \mathsf{D}_f$;
- If $p(\mathbf{x} = 0) \iff q(\mathbf{x}) = 0$, then $\mathsf{D}_f(p\|q) = \mathsf{D}_{f^\diamond}(q\|p)$, where $f^\diamond(t) := t \cdot f(1/t)$;
- $f^\diamond$ is (strictly) convex, $f^\diamond(1) = 0$ and $(f^\diamond)^\diamond = f$;

The second last result indicates that f-divergences are not usually symmetric. However, we can always symmetrize them by the transformation: $f \leftarrow f + f^\diamond$.

Example 0.15: KL and LK

Let $f(t) = t \log t$, then we obtain the Kullback-Leibler (KL) divergence:

$$\mathsf{KL}(p\|q) = \int p(\mathbf{x}) \log(p(\mathbf{x})/q(\mathbf{x}))\,\mathrm{d}\mathbf{x}.$$

Reverse the inputs we obtain the reverse KL divergence:

$$\mathsf{LK}(p\|q) := \mathsf{KL}(q\|p).$$

Verify by yourself that the underlying function $f = -\log$ for reverse KL.

Definition 0.16: Entropy, conditional entropy, cross-entropy, and mutual information

We define the entropy of a random vector $\mathbf{X}$ with pdf p as:

$$\mathsf{H}(\mathbf{X}) := \mathsf{E} - \log p(\mathbf{X}) = -\int p(\mathbf{x}) \log p(\mathbf{x}) \, \mathrm{d}\mathbf{x},$$

the conditional entropy between $\mathbf{X}$ and $\mathbf{Z}$ (with pdf q) as:

$$\mathsf{H}(\mathbf{X}|\mathbf{Z}) := \mathsf{E} - \log p(\mathbf{X}|\mathbf{Z}) = -\int p(\mathbf{x}, \mathbf{z}) \log p(\mathbf{x}|\mathbf{z}) \, \mathrm{d}\mathbf{x} \, \mathrm{d}\mathbf{z},$$

and the cross-entropy between $\mathbf{X}$ and $\mathbf{Z}$ as:

$$\dagger(\mathbf{X}, \mathbf{Z}) := \mathsf{E} - \log q(\mathbf{X}) = -\int p(\mathbf{x}) \log q(\mathbf{x}) \, \mathrm{d}\mathbf{x}.$$

Finally, we define the mutual information between $\mathbf{X}$ and $\mathbf{Z}$ as:

$$\mathsf{I}(\mathbf{X}, \mathbf{Z}) := \mathsf{KL}(p(\mathbf{x}, \mathbf{z}) \| p(\mathbf{x})q(\mathbf{z})) = \int p(\mathbf{x}, \mathbf{z}) \log \frac{p(\mathbf{x}, \mathbf{z})}{p(\mathbf{x})q(\mathbf{z})} \, \mathrm{d}\mathbf{x} \, \mathrm{d}\mathbf{z}$$

Exercise 0.17: Information theory

Verify the following:

$$\begin{aligned}
\mathsf{H}(\mathbf{X}, \mathbf{Z}) &= \mathsf{H}(\mathbf{Z}) + \mathsf{H}(\mathbf{X}|\mathbf{Z}) \\
\dagger(\mathbf{X}, \mathbf{Z}) &= \mathsf{H}(\mathbf{X}) + \mathsf{KL}(\mathbf{X} \| \mathbf{Z}) = \mathsf{H}(\mathbf{X}) + \mathsf{LK}(\mathbf{Z} \| \mathbf{X}) \\
\mathsf{I}(\mathbf{X}, \mathbf{Z}) &= \mathsf{H}(\mathbf{X}) - \mathsf{H}(\mathbf{X}|\mathbf{Z}) \\
\mathsf{I}(\mathbf{X}, \mathbf{Z}) &\geq 0, \text{ with equality iff } \mathbf{X} \text{ independent of } \mathbf{Z} \\
\mathsf{KL}(p(\mathbf{x}, \mathbf{z}) \| q(\mathbf{x}, \mathbf{z})) &= \mathsf{KL}(p(\mathbf{z}) \| q(\mathbf{z})) + \mathsf{E}[\mathsf{KL}(p(\mathbf{x}|\mathbf{z}) \| q(\mathbf{x}|\mathbf{z}))].
\end{aligned}$$

All of the above can obviously be iterated to yield formula for more than two random vectors.

Exercise 0.18: Multivariate Gaussian

Compute

- the entropy of the multivariate Gaussian $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$;
- the KL divergence between two multivariate Gaussians $\mathcal{N}(\boldsymbol{\mu}_1, \Sigma_1)$ and $\mathcal{N}(\boldsymbol{\mu}_2, \Sigma_2)$.

Example 0.19: More divergences, more fun

Derive the formula for the following f-divergences:

- χ^2-divergence: $f(t) = (t-1)^2$;
- Hellinger divergence: $f(t) = (\sqrt{t} - 1)^2$;
- total variation: $f(t) = |t - 1|$;
- Jensen-Shannon divergence: $f(t) = t \log t - (t+1) \log(t+1) + \log 4$;
- Rényi divergence (Rényi 1961): $f(t) = \frac{t^\alpha - 1}{\alpha - 1}$ for some $\alpha > 0$ (for $\alpha = 1$ we take limit and obtain ?).

Which of the above are symmetric?

Rényi, Alfréd (1961). "On Measures of Entropy and Information". In: *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability*, pp. 547–561.

Remark 0.20: MLE = KL minimization

Let us define the empirical "pdf" based on a dataset $\mathcal{D} = \wr \mathbf{x}_1, \ldots, \mathbf{x}_n \int$:

$$\hat{p}(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} \delta_{\mathbf{x}_i},$$

where $\delta_{\mathbf{x}}$ is the "illegal" delta mass concentrated at $\mathbf{x}$. Then, we claim that

$$\theta_{\mathsf{MLE}} = \operatorname*{argmin}_{\theta\in\Theta} \ \mathsf{KL}\big(\hat{p}\|p(\mathbf{x}|\theta)\big).$$

Indeed, we have

$$\mathsf{KL}(\hat{p}\|p(\mathbf{x}|\theta)) = \int [\log(\hat{p}(\mathbf{x})) - \log p(\mathbf{x}|\theta)]\hat{p}(\mathbf{x})\,\mathrm{d}\mathbf{x} = C + \frac{1}{n}\sum_{i=1}^{n} -\log p(\mathbf{x}_i|\theta),$$

where C is a constant that does not depend on θ.

Exercise 0.21: Is the flood gate open?

Now obviously you are thinking to replace the KL divergence with any f-divergence, hoping to obtain some generalization of MLE. Try and explain any difficulty you may run into. (We will revisit this in the GAN lecture.)

Exercise 0.22: Why KL is so special

To appreciate the uniqueness of the KL divergence, prove the following:

log is the only continuous function satisfying $f(st) = f(s) + f(t)$.

Remark 0.23: Information theory for ML

A beautiful they that connects information theory, Bayes risk, convexity and proper loss is available in (Grünwald and Dawid 2004; Reid and Williamson 2011) and the references therein.

Grünwald, Peter D. and Alexander Philip Dawid (2004). "Game theory, maximum entropy, minimum discrepancy and robust Bayesian decision theory". *Annals of Statistics*, vol. 32, no. 4, pp. 1367–1433.

Reid, Mark D. and Robert C. Williamson (2011). "Information, Divergence and Risk for Binary Experiments". *Journal of Machine Learning Research*, vol. 12, no. 22, pp. 731–817.

Example 0.24: Linear regression as MLE

Let us now give linear regression a probabilistic interpretation, by making the following assumption:

$$Y = \mathbf{x}^\top \mathbf{w} + \epsilon,$$

where $\epsilon \sim \mathcal{N}(0, \sigma^2)$. Namely, the response is a linear function of the feature vector $\mathbf{x}$, corrupted by some standard Gaussian noise, or in fancy notation: $Y \sim \mathcal{N}(\mathbf{x}^\top \mathbf{w}, \sigma^2)$. Given a dataset $\mathcal{D} = \wr(\mathbf{x}_1, y_1) \ldots, (\mathbf{x}_n, y_n)\int$ (where we assume the feature vectors $\mathbf{x}_i$ are fixed and deterministic, unlike the responses y_i which are random), the likelihood function of the parameter $\mathbf{w}$ is:

$$L(\mathbf{w}; \mathcal{D}) = p(\mathcal{D}|\mathbf{w}) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - \mathbf{x}_i^\top \mathbf{w})^2}{2\sigma^2}\right)$$

$$\hat{\mathbf{w}}_{\mathsf{MLE}} = \operatorname*{argmin}_{\mathbf{w}} \; \frac{n}{2}\log\sigma^2 + \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \mathbf{x}_i^\top \mathbf{w})^2,$$

which is exactly the ordinary linear regression.

Moreover, we can now also obtain an MLE of the noise variance σ^2 by solving:

$$\begin{aligned}\hat{\sigma}^2_{\mathsf{MLE}} &= \operatorname*{argmin}_{\sigma^2} \; \frac{n}{2}\log\sigma^2 + \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \mathbf{x}_i^\top \mathbf{w})^2 \\ &= \frac{1}{n}\sum_{i=1}^{n}(y_i - \mathbf{x}_i^\top \hat{\mathbf{w}}_{\mathsf{MLE}})^2,\end{aligned}$$

which is nothing but the average training error.

Definition 0.25: Prior

In a full Bayesian approach, we also assume the parameter θ is random and follows a prior pdf $p(\theta)$. Ideally, we choose the prior $p(\theta)$ to encode our *a priori* knowledge of the problem at hand. (Regrettably, in practice computational convenience often dominates the choice of the prior.)

Definition 0.26: Posterior

Suppose we have chosen a prior pdf $p(\theta)$ for our parameter of interest θ. After observing some data $\mathcal{D}$, our belief on the probable values of θ will have changed, so we obtain the posterior:

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})} = \frac{p(\mathcal{D}|\theta)p(\theta)}{\int p(\mathcal{D}|\theta)p(\theta)\,\mathrm{d}\theta},$$

where recall that $p(\mathcal{D}|\theta)$ is exactly the likelihood of θ given the data $\mathcal{D}$. Note that computing the denominator may be difficult since it involves an integral that may not be tractable.

Example 0.27: Bayesian linear regression

Let us consider linear regression (with vector-valued response $\mathbf{y} \in \mathbb{R}^m$, matrix-valued covariate $X \in \mathbb{R}^{m\times d}$):

$$\mathbf{Y} = X\mathbf{w} + \boldsymbol{\epsilon},$$

where the noise $\boldsymbol{\epsilon} \sim \mathcal{N}_m(\boldsymbol{\mu}, S)$ and we impose a Gaussian prior on the weights $\mathbf{w} \sim \mathcal{N}_d(\boldsymbol{\mu}_0, S_0)$. As usual we assume $\boldsymbol{\epsilon}$ is independent of $\mathbf{w}$. Given a dataset $\mathcal{D} = \wr(X_1, \mathbf{y}_1), \ldots, (X_n, \mathbf{y}_n)\int$, we compute the posterior:

$$\begin{aligned}p(\mathbf{w}|\mathcal{D}) &\propto p(\mathbf{w})p(\mathcal{D}|\mathbf{w}) \\ &\propto \exp\left(-\frac{(\mathbf{w}-\boldsymbol{\mu}_0)^\top S_0^{-1}(\mathbf{w}-\boldsymbol{\mu}_0)}{2}\right) \cdot \prod_{i=1}^{n} \exp\left(-\frac{(\mathbf{y}_i - X_i\mathbf{w} - \boldsymbol{\mu})^\top S^{-1}(\mathbf{y}_i - X_i\mathbf{w} - \boldsymbol{\mu})}{2}\right) \\ &= \mathcal{N}(\boldsymbol{\mu}_n, S_n),\end{aligned}$$

where (by completing the square) we have

$$S_n^{-1} = S_0^{-1} + \sum_{i=1}^n X_i^\top S^{-1} X_i$$
$$\boldsymbol{\mu}_n = S_n \left(S_0^{-1} \boldsymbol{\mu}_0 + \sum_{i=1}^n X_i^\top S^{-1} (\mathbf{y}_i - \boldsymbol{\mu}) \right).$$

The posterior covariance S_n contains both the prior covariance S_0 and the data X_i. As $n \to \infty$, data dominates the prior. Similar remark applies to the posterior mean $\boldsymbol{\mu}_n$.

We can also derive the predictive distribution on a new input X:

$$\begin{aligned} p(\mathbf{y}|X, \mathcal{D}) &= \int p(\mathbf{y}|X, \mathbf{w}) p(\mathbf{w}|\mathcal{D}) \,\mathrm{d}\mathbf{w} \\ &= \mathcal{N}(X\boldsymbol{\mu}_n + \boldsymbol{\mu}, X S_n X^\top + S) \end{aligned}$$

The covariance $X S_n X^\top + S$ reflects our uncertainty on the prediction at X.

Theorem 0.28: Bayes classifier

Consider the classification problem with random variables $\mathbf{X} \in \mathbb{R}^d$ and $Y \in [\mathsf{c}] := \{1, \ldots, \mathsf{c}\}$. The optimal (Bayes) classification rule, defined as

$$\operatorname*{argmin}_{h:\mathbb{R}^d \to [\mathsf{c}]} \ \Pr(Y \neq h(\mathbf{X})),$$

admits the closed-form formula:

$$\begin{aligned} h^\star(\mathbf{x}) &= \operatorname*{argmax}_{k \in [\mathsf{c}]} \ \Pr(Y = k | \mathbf{X} = \mathbf{x}) \qquad (0.1) \\ &= \operatorname*{argmax}_{k \in [\mathsf{c}]} \ \underbrace{p(\mathbf{X} = \mathbf{x}|Y = k)}_{\textit{likelihood}} \cdot \underbrace{\Pr(Y = k)}_{\textit{prior}}, \end{aligned}$$

where ties can be broken arbitrarily.

Proof. Let $h(\mathbf{x})$ be any classification rule. Its classification error is:

$$\Pr(h(\mathbf{X}) \neq Y) = 1 - \Pr(h(\mathbf{X}) = Y) = 1 - \mathsf{E}[\Pr(h(\mathbf{X}) = Y|\mathbf{X})].$$

Thus, conditioned on $\mathbf{X}$, to minimize the error we should maximize $\Pr(h(\mathbf{X}) = Y|\mathbf{X})$, leading to $h(\mathbf{x}) = h^\star(\mathbf{x})$.

To understand the second formula, we resort to the definition of conditional expectation:

$$\begin{aligned} \int_A \Pr(Y = k|\mathbf{X} = \mathbf{x}) p(\mathbf{x}) \,\mathrm{d}\mathbf{x} &= \Pr(\mathbf{X} \in A, Y = k) \\ &= \Pr(\mathbf{X} \in A|Y = k) \Pr(Y = k) \\ &= \int_A p(\mathbf{X} = \mathbf{x}|Y = k) \Pr(Y = k) \,\mathrm{d}\mathbf{x}. \end{aligned}$$

Since the set A is arbitrary, we must have

$$\Pr(Y = k|\mathbf{X} = \mathbf{x}) = \frac{p(\mathbf{X} = \mathbf{x}|Y = k) \Pr(Y = k)}{p(\mathbf{X} = \mathbf{x})}.$$

(We assume the marginal density $p(\mathbf{x})$ and class-specific densities $p(\mathbf{x}|Y = k)$ exist.) □

In practice, we do not know the distribution of $(\mathbf{X}, Y)$, hence we cannot compute the optimal Bayes classification rule. One natural idea is to estimate the pdf of $(\mathbf{X}, Y)$ and then plug into (0.1). This approach however does not scale to high dimensions and we will see direct methods that avoid estimating the pdf.

It is clear that the Bayes error (achieved by the Bayes classification rule) is:

$$\mathsf{E}\big[1 - \max_{k \in [\mathsf{c}]} \Pr(Y = k | \mathbf{X})\big].$$

In particular, for $\mathsf{c} = 2$, we have

$$\text{Bayes error} = \mathsf{E}\big[\min\{\Pr(Y = 1|\mathbf{X}), \Pr(Y = -1|\mathbf{X})\}\big].$$

Exercise 0.29: Cost-sensitive classification (Elkan 2001)

Cost-sensitive classification refers to the setting where making certain mistakes is more expensive than making some other ones. Formally, we suffer cost c_{ij} when we predict class i while the true class is j. We may of course assume $c_{ii} \equiv 0$. Derive the optimal Bayes rule.

Elkan, Charles (2001). "The Foundations of Cost-Sensitive Learning". In: *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 973–978.

Exercise 0.30: Bayes estimator

Let $\ell : \hat{\mathcal{Y}} \times \mathcal{Y} \to \mathbb{R}_+$ be a loss function that compares our prediction $\hat{\mathbf{y}}$ with the groundtruth $\mathbf{y}$. We define the Bayes estimator as:

$$\min_{f:\mathcal{X}\to\hat{\mathcal{Y}}} \mathsf{E}\ell(f(\mathbf{X}), \mathbf{Y}).$$

Can you derive the formula for the Bayes estimator (using conditional expectation)?

Definition 0.31: Maximum a posteriori (MAP)

Another popular parameter estimation algorithm is the MAP that simply maximizes the posterior:

$$\begin{aligned}\theta_{\mathsf{MAP}} &:= \operatorname*{argmax}_{\theta \in \Theta} \ p(\theta|\mathcal{D}) \\ &= \operatorname*{argmin}_{\theta \in \Theta} \ \underbrace{-\log p(\mathcal{D}|\theta)}_{\text{negative log-likelihood}} + \underbrace{-\log p(\theta)}_{\text{prior as regularization}}\end{aligned}$$

A strong (i.e. sharply concentrated, i.e. small variance) prior helps reducing the variance of our estimator, with potential damage to increasing our bias (see Definition 0.10) if our *a priori* belief is mis-specified, such as stereotypes ☺.

MAP is *not* a Bayes estimator, since we cannot find an underlying loss ℓ for it.

Example 0.32: Ridge regression as MAP

Continuing Example 0.24 let us now choose a standard Gaussian prior $\mathbf{w} \sim \mathcal{N}(\mathbf{0}, \frac{1}{\lambda}\mathbb{I})$. Then,

$$\hat{\mathbf{w}}_{\mathsf{MAP}} = \operatorname*{argmin}_{\mathbf{w}} \ \frac{n}{2}\log\sigma^2 + \frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i - \mathbf{x}_i^\top \mathbf{w})^2 + \frac{\lambda}{2}\|\mathbf{w}\|_2^2 - \frac{d}{2}\log\lambda,$$

which is exactly equivalent to ridge regression. Note that the larger the regularization constant λ is, the smaller the variance of the prior is. In other words, larger regularization means more determined prior information.

Needless to say, if we choose a different prior on the weights, MAP would yield a different regularized linear regression formulation. For instance, with the Laplacian prior (which is more peaked than the Gaussian around the mode), we obtain the celebrated Lasso (Tibshirani 1996):

$$\min_{\mathbf{w}} \ \frac{1}{2\sigma^2}\|X\mathbf{w}-\mathbf{y}\|_2^2 + \lambda\|\mathbf{w}\|_1.$$

Tibshirani, Robert (1996). "Regression Shrinkage and Selection via the Lasso". *Journal of the Royal Statistical Society: Series B*, vol. 58, no. 1, pp. 267–288.

Theorem 0.33: Bayes rule arose from optimization (e.g. Zellner 1988)

Let $p(\theta)$ be a prior pdf of our parameter θ, $p(\mathcal{D}|\theta)$ the pdf of data $\mathcal{D}$ given θ, and $p(\mathcal{D}) = \int p(\theta)p(\mathcal{D}|\theta)\,\mathrm{d}\theta$ the data pdf. Then,

$$p(\theta|\mathcal{D}) = \operatorname*{argmin}_{q(\theta)} \ \mathsf{KL}\big(p(\mathcal{D})q(\theta) \ \| \ p(\theta)p(\mathcal{D}|\theta)\big), \tag{0.2}$$

where the minimization is over all pdf $q(\theta)$.

Proof. KL is nonnegative while the posterior $p(\theta|\mathcal{D})$ already achieves 0. In fact, only the posterior can achieve 0, see Exercise 0.14. ☐

This result may seem trivial at first sight. However, it motives a number of important extensions:

- If we restrict the minimization to a subclass $\mathcal{P}$ of pdfs, then we obtain some KL projection of the posterior $p(\theta|\mathcal{D})$ to the class $\mathcal{P}$. This is essentially the so-called variational inference.
- If we replace the KL divergence with any other f-divergence, the same result still holds. This opens a whole range of possibilities when we can only optimize over a subclass $\mathcal{P}$ of pdfs.
- The celebrated expectation-maximization (EM) algorithm also follows from (0.2)!

We will revisit each of the above extensions later in the course.

Zellner, Arnold (1988). "Optimal Information Processing and Bayes's Theorem". *The American Statistician*, vol. 42, no. 4, pp. 278–280.

1 Perceptron

Goal

Understand the celebrated perceptron algorithm for online binary classification.

Alert 1.1: Convention

Gray boxes are not required hence can be omitted for unenthusiastic readers.

This note is likely to be updated again soon.

Definition 1.2: Binary classification

Given a set of n known example pairs $\llbracket(\mathbf{x}_i, y_i) : i = 1, 2, \ldots, n\rrbracket$, where $\mathbf{x}_i \in \mathbb{R}^d$ and $y_i \in \{\pm 1\}$, we want to learn a (binary) "classification rule" $h : \mathbb{R}^d \to \{\pm 1\}$, so that

$$h(\mathbf{x}) = y$$

on *most* unseen (future) examples $(\mathbf{x}, y)$. Throughout we will call $\mathbf{x}_i$ the feature vector of the i-th example, and y_i the (binary) label of the i-th example. Together, the known example pairs $\llbracket(\mathbf{x}_i, y_i) : i = 1, 2, \ldots, n\rrbracket$ are called the training set, with n being its size and d being its dimension. The unseen future example $(\mathbf{x}, y)$ will be called the test example. If we have a set of test examples, together they will be called a test set.

Alert 1.3: Notations

We use boldface letters, e.g. $\mathbf{x}$, for a vector of appropriate size. Subscripts are used for two purposes: (the bold) $\mathbf{x}_i$ denotes a vector that may have nothing to do with $\mathbf{x}$, while (the non-bold) x_i denotes the i-th coordinate of $\mathbf{x}$. The j-th coordinate of $\mathbf{x}_i$ will be denoted as x_{ji}. We use $\mathbf{1}$ and $\mathbf{0}$ to denote a vector of all 1s and 0s of appropriate size (which should be clear from context), respectively.

By default, all vectors are column vectors and we use $\mathbf{x}^\top$ to denote the transpose (i.e. a row vector) of a column vector $\mathbf{x}$.

Definition 1.4: Functions and sets are equivalent

A binary classification rule $h : \mathbb{R}^d \to \{\pm 1\}$ can be identified with a set $P \subseteq \mathbb{R}^d$ and its complement $N = \mathbb{R}^d \setminus P$, where $h(\mathbf{x}) = 1 \iff \mathbf{x} \in P$.

Exercise 1.5: Multiclass rules

Let $h : \mathbb{R}^d \to \{1, 2, \ldots, c\}$, where $c \geq 2$ is the number of classes. How do we identify the function h with sets?

Remark 1.6: Memorization does NOT work... Or does it?

The challenge of binary classification lies in two aspects:

- on a test example $(\mathbf{x}, y)$, we actually only have access to $\mathbf{x}$ but not the label y. It is our job to predict y, hopefully correctly most of the time.
- the test example $\mathbf{x}$ can be (very) different from any of the training examples $\{\mathbf{x}_i : i = 1, \ldots, n\}$. So we can not expect *naive* memorization to work.

Essentially, we need a (principled?) way to interpolate from the training set (where labels are known) and hopefully generalize to the test set (where labels need to be predicted). For this to be possible, we need

- the training set to be "indicative" of what the test set look like, and/or
- a proper baseline (competitor) to compare against.

Definition 1.7: Statistical learning

We assume the training examples $(\mathbf{x}_i, y_i)$ and the test example $(\mathbf{x}, y)$ are drawn independently and identically (i.i.d.) from an unknown distribution $\mathbb{P}$:

$$(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n), (\mathbf{x}, y) \overset{i.i.d.}{\sim} \mathbb{P},$$

in which case we usually use capital letters $(\mathbf{X}_i, Y_i)$ and $(\mathbf{X}, Y)$ to emphasize the random nature of these quantities. Our goal is then to find a classification rule $h : \mathbb{R}^d \to \{\pm 1\}$ so that the classification error

$$\mathbb{P}(h(\mathbf{X}) \neq Y)$$

is as small as possible. Put in optimization terms, we are interested in solving the following (abstract) optimization problem:

$$\min_{h:\mathbb{R}^d \to \{\pm 1\}} \ \mathbb{P}(h(\mathbf{X}) \neq Y). \tag{1.1}$$

We will shortly see that if $\mathbb{P}$ is known, then the classification problem (1.1) admits a closed-form solution known as the Bayes classifier. In the more realistic case where $\mathbb{P}$ is not known, our hope is that the training set $\{(\mathbf{X}_i, Y_i) : i = 1, \ldots, n\}$ may provide enough information about $\mathbb{P}$, as least when n is sufficiently large, which is basically the familiar law of large numbers in (serious) disguise.

Remark 1.8: i.i.d., seriously?

Immediate objections to the i.i.d. assumption in statistical learning include (but not limit to):

- the training examples are hardly i.i.d..
- the test example may follow a different distribution than the training set, known as domain shift.

Reasons to support the i.i.d. assumption include (but not limit to):

- it is a simple, clean mathematical abstraction that allows us to take a first step in understanding and solving the binary classification problem.
- for many real problems, the i.i.d. assumption is not terribly off. In fact, it is a reasonably successful approximation.
- there exist more complicated ways to alleviate the i.i.d. assumption, usually obtained by refining results under the i.i.d. assumption.

We will take a more pragmatic viewpoint: we have to start from somewhere and the i.i.d. assumption seems to be a good balance between what we can analyze and what we want to achieve.

Definition 1.9: Online learning

A different strategy, as opposed to statistical learning, is not to put any assumption whatsoever on the data, but on what we want to compare against: Given a collection of existing classification rules $\mathcal{G} = \{g_k : k \in K\}$, we want to construct a classification rule h that is competitive against the "best" $g^* \in \mathcal{G}$, in terms of the number of mistakes:

$$\mathfrak{M}(h) := \sum_{i=1}^{n} [\![h(\mathbf{x}_i) \neq y_i]\!].$$

The "subtlety" is that even the best $g^* \in \mathcal{G}$ may not perform well on the data $\mathcal{D} = \wr(\mathbf{x}_i, y_i) : i = 1, \ldots, n \int$, so being competitive against the best g^* in $\mathcal{G}$ may or may not be as significant as you would have liked.

When the examples $(\mathbf{x}_i, y_i)$ come one at a time, i.e. in the online (streaming) fashion, we can give ourselves even more flexibility: we construct a sequence of classification rules $\{h_i : i = 1, 2, \ldots\}$, and the evaluation proceeds as follows. Start with $i = 1$ and choose h_1:

(I). receive $\mathbf{x}_i$ and predict $\hat{y}_i = h_i(\mathbf{x}_i)$

(II). receive true label y_i and possibly suffer a mistake if $\hat{y}_i \neq y_i$

(III). adjust h_i to h_{i+1} and increment i by 1.

(We could also allow h_i to depend on $\mathbf{x}_i$, i.e. delay the adjustment of h_i until receiving $\mathbf{x}_i$.) Note that while we are allowed to adaptively adjust our classification rules $\{h_i\}$, the competitor is more restricted: it has to stick to some fixed rule $g_k \in \mathcal{G}$ chosen well before seeing any example.

Definition 1.10: Proper vs. improper learning

When we require $h \in \mathcal{G}$, we are in the proper learning regime, where we (who chooses h) must act in the same space as the competitor (who is constrained to choose from $\mathcal{G}$). However, sometimes learning is easier if we abandon $h \in \mathcal{G}$ and operate in the improper learning regime. Life is never fair anyways ☺.

Of course, this distinction is relative to the class $\mathcal{G}$. In particular, if $\mathcal{G}$ consists of all possible functions, then any learning algorithm is proper but we are competing against a very strong competitor.

Alert 1.11: Notation

The Iverson notation $[\![A]\!]$ or sometimes also $\mathbf{1}(A)$ (or even $\mathbf{1}_A$) denotes the indicator function of the event $A \subseteq \mathbb{R}^d$, i.e., $[\![A]\!]$ is 1 if the event A holds and 0 otherwise.

We use $|A|$ to denote the size (i.e. the number of elements) of a set A.

Definition 1.12: Thresholding

Often it is more convenient to learn a real-valued function $f : \mathbb{R}^d \to \mathbb{R}$ and then use thresholding to get a binary-valued classification rule: $h = \operatorname{sign}(f)$, where say we define $\operatorname{sign}(0) = -1$ (or $\operatorname{sign}(0) = 1$, the actual choice is usually immaterial).

Definition 1.13: Linear and affine functions

Perhaps the simplest multivariate function is the class of linear/affine functions. Recall that a function $f : \mathbb{R}^d \to \mathbb{R}$ is linear if for all $\mathbf{w}, \mathbf{z} \in \mathbb{R}^d$ and $\alpha, \beta \in \mathbb{R}$:

$$f(\alpha\mathbf{w} + \beta\mathbf{z}) = \alpha f(\mathbf{w}) + \beta f(\mathbf{z}).$$

From the definition it follows that $f(\mathbf{0}) = 0$ for any linear function f.

Similarly, a function $f : \mathbb{R}^d \to \mathbb{R}$ is called affine if for all $\mathbf{w}, \mathbf{z} \in \mathbb{R}^d$ and $\alpha \in \mathbb{R}$:

$$f(\alpha \mathbf{w} + (1-\alpha)\mathbf{z}) = \alpha f(\mathbf{w}) + (1-\alpha) f(\mathbf{z}).$$

Compared to the definition of linear functions, the restriction $\alpha + \beta = 1$ is enforced.

Exercise 1.14: Representation of linear and affine functions

Prove:

- If $f : \mathbb{R}^d \to \mathbb{R}$ is linear, then for any $n \in \mathbb{N}$, any $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$, any $\alpha_1, \ldots, \alpha_n \in \mathbb{R}$, we have

$$f\left(\sum_{i=1}^n \alpha_i \mathbf{x}_i\right) = \sum_{i=1}^n \alpha_i f(\mathbf{x}_i).$$

  What is the counterpart for affine functions?

- A function $f : \mathbb{R}^d \to \mathbb{R}$ is linear iff there exists some $\mathbf{w} \in \mathbb{R}^d$ so that $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x}$.
- A function $f : \mathbb{R}^d \to \mathbb{R}$ is affine iff there exists some $\mathbf{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$ so that $f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$, i.e. an affine function is a linear function translated by some constant.

Definition 1.15: Inner product

Recall the inner product (i.e. dot product) between two vectors $\mathbf{w}, \mathbf{x} \in \mathbb{R}^d$ is defined as

$$\mathbf{w}^\top \mathbf{x} = \sum_{j=1}^d w_j x_j = \mathbf{x}^\top \mathbf{w}.$$

The notation $\langle \mathbf{w}, \mathbf{x} \rangle$ is also used (especially when we want to abstract away coordinates/basis).

Algorithm 1.16: Perceptron

Combining thresholding (see Definition 1.12) and affine functions (see Definition 1.13), the celebrated perceptron algorithm of Rosenblatt (1958) tries to learn a classification rule

$$h(\mathbf{x}) = \operatorname{sign}(f(\mathbf{x})), \qquad f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b,$$

parameterized by the weight vector $\mathbf{w} \in \mathbb{R}^d$ and bias (threshold) $b \in \mathbb{R}$ so that

$$\forall i, \quad y_i = h(\mathbf{x}_i) \iff y_i \hat{y}_i > 0, \quad \hat{y}_i = f(\mathbf{x}_i) = \mathbf{w}^\top \mathbf{x}_i + b, \tag{1.2}$$

where we have used the fact that $y_i \in \{\pm 1\}$ (and ignored the possibility of $\hat{y}_i = 0$ for the direction $\Rightarrow$).

In the following perceptron algorithm, the training examples come in the online fashion (see Definition 1.9), and the algorithm updates only when it makes a "mistake" (line 3).

```
Algorithm: Perceptron (Rosenblatt 1958)
Input: Dataset $\mathcal{D} = \wr(\mathbf{x}_i, y_i) \in \mathbb{R}^d \times \{\pm 1\} : i = 1, \ldots, n\int$, initialization $\mathbf{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$,
       threshold $\delta \geq 0$
Output: approximate solution $\mathbf{w}$ and $b$
1 for $t = 1, 2, \ldots$ do
2     receive training example index $I_t \in \{1, \ldots, n\}$          // the index $I_t$ can be random
3     if $y_{I_t}(\mathbf{w}^\top \mathbf{x}_{I_t} + b) \leq \delta$ then
4         $\mathbf{w} \leftarrow \mathbf{w} + y_{I_t}\mathbf{x}_{I_t}$          // update only after making a ‘‘mistake’’
5         $b \leftarrow b + y_{I_t}$
```

We can break the for-loop if a maximum number of iterations has been reached, or if all training examples are correctly classified in a full cycle (in which case the algorithm will no longer update itself).

Rosenblatt, F. (1958). “The perceptron: A probabilistic model for information storage and organization in the brain”. *Psychological Review*, vol. 65, no. 6, pp. 386–408.

Remark 1.17: Padding

If we define $\mathbf{a}_i = y_i \begin{bmatrix} \mathbf{x}_i \\ 1 \end{bmatrix}$, $\mathsf{A} = [\mathbf{a}_1, \ldots, \mathbf{a}_n] \in \mathbb{R}^{p \times n}$ (where $p = d + 1$ stands for the number of predictors), and $\mathbf{w} = \begin{bmatrix} \mathbf{w} \\ b \end{bmatrix}$, then clearly

$$\mathbf{a}_i^\top \mathbf{w} = y_i(\mathbf{w}^\top \mathbf{x}_i + b).$$

Thus, the perceptron problem (1.2) can be concisely reduced to the following (slightly more general) system of linear inequalities:

$$\text{Given } \mathsf{A} \in \mathbb{R}^{p \times n} \quad \text{find } \mathbf{w} \in \mathbb{R}^p \text{ so that } \mathsf{A}^\top \mathbf{w} > \mathbf{0}, \tag{1.3}$$

where the (strict) inequality is meant elementwise, i.e. $\mathbf{x} > \mathbf{w} \iff \forall j,\ x_j > w_j$. In the sequel we will identify the perceptron problem (1.2) with the above system of linear equalities in (1.3).

The trick to pad the constant 1 to $\mathbf{x}_i$ and the bias b to $\mathbf{w}$ so that we can deal with the pair $(\mathbf{w}, b)$ more concisely is used ubiquitously in machine learning. The trick to multiply the *binary* label y_i to $\mathbf{x}_i$ is also often used in binary classification problems.

Alert 1.18: Notation

We use $\mathbf{x}$ and $\mathbf{w}$ for the original vectors and $\mathbf{x}$ and $\mathbf{w}$ for the padded versions (with constant 1 and bias b respectively). Similar, we use X and W for the original matrices and X and $\mathbf{w}$ for the padded versions.

We use $\mathrm{\hat{y}} \in \mathbb{R}$ for a real-valued prediction and $\hat{y} \in \{\pm 1\}$ for a binary prediction, keeping in mind that usually $\hat{y} = \mathrm{sign}(\mathrm{\hat{y}})$.

Remark 1.19: “If it ain’t broke, don’t fix it”

The perceptron algorithm is a perfect illustration of the good old wisdom: “If it ain’t broke, don’t fix it.” Indeed, it maintains the same weight vector $(\mathbf{w}, b)$ until when a “mistake” happens, i.e. line 3 in Algorithm 1.16. This principle is often used in designing machine learning algorithms, or in life ☺.

On the other hand, had we always performed the updates in line 4 and 5 (even when we predicted correctly), then it is easy to construct an infinite sequence $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots$, that is strictly linearly separable (see Definition 1.24 below), and yet the modified (aggressive) perceptron will make infinitely many mistakes.

Indeed, let $\delta = 0$ and $y_i \equiv 1$. This dataset (with any $\mathbf{x}_i$) is clearly linearly separable (simply take $\mathbf{w} = \mathbf{0}, b = 1$). The aggressive perceptron maintains the weight (assuming w.l.o.g that $\mathbf{w}_0 = \mathbf{0}$)

$$\mathbf{w}_t = \sum_{i=1}^{t} \begin{pmatrix} \mathbf{x}_i \\ 1 \end{pmatrix}.$$

Thus, to fool it we need

$$\langle \mathbf{w}_t, \mathbf{a}_{t+1} \rangle = \left\langle \sum_{i=1}^{t} \mathbf{x}_i, \mathbf{x}_{t+1} \right\rangle + t < 0, \quad i.e., \quad \langle \bar{\mathbf{x}}_t, \mathbf{x}_{t+1} \rangle < -1,$$

where $\bar{\mathbf{x}}_t := \frac{1}{t} \sum_{i=1}^{t} \mathbf{x}_i$. Now, let

$$\mathbf{x}_t = \begin{cases} -2\mathbf{x}, & \text{if } \log_2 t \in \mathbb{N} \\ \mathbf{x}, & \text{otherwise} \end{cases},$$

for any fixed $\mathbf{x}$ with unit length, i.e. $\|\mathbf{x}\|_2 = 1$. Then, we verify $\bar{\mathbf{x}}_t \to \mathbf{x}$ while for sufficiently large t such that $\log_2(t+1) \in \mathbb{N}$, we have $\langle \bar{\mathbf{x}}_t, \mathbf{x}_{t+1} \rangle \to \langle \mathbf{x}, -2\mathbf{x} \rangle = -2 < -1$. Obviously, such t's form an infinite subsequence, on which the aggressive perceptron errs.

Exercise 1.20: Mistakes all the time

Construct a linearly separable sequence $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots$, so that the aggressive perceptron makes mistakes on every example! [Hint: you may let $\mathbf{x}_i$ go unbounded. Can you make it bounded?]

History 1.21: "What doesn't kill you makes you stronger" ?

Historically, perceptron was the first algorithm that kicked off the entire field of artificial intelligence. Its design, analysis, and application have had lasting impact on the machine learning field even to this day. Ironically, the failure of perceptron on nonlinear problems (to be discussed in later lectures) almost killed the entire artificial intelligence field as well...

Exercise 1.22: Perceptron for solving (homogeneous) linear inequalities

Modify Algorithm 1.16 to solve the system of (homogeneous) linear inequalities (1.3).

Alert 1.23: Existence and uniqueness of solution

For any problem you are interested in solving, the first question you should ask is:

$$\boxed{\text{Does there exist a solution?}}$$

- If the answer is "no," then the second question you should ask is:

$$\boxed{\text{If there is no solution at all, can we still ``solve'' the problem in certain meaningful ways?}}$$

- If the answer is "yes," then the second question you should ask is:

$$\boxed{\text{If there is at least one solution, then is the solution unique?}}$$

  - If the answer is "no," then the third question you should ask is:

Is there any reason to prefer a certain solution?

Too often in ML, we hasten to design algorithms and run experiments, without fully understanding or articulating what we are trying to solve, or deciding if the problem is even solvable in any well-defined sense. There is certain value in this philosophy but also great danger.

Definition 1.24: (Strictly) linear separable

We say that the perceptron problem (1.3) is (strictly) linearly separable if for some hence all $s>0$, there exists some $\mathbf{w}$ such that $\forall i, \mathbf{a}_i^\top \mathbf{w} \geq s > 0$ (or in matrix notation $\mathsf{A}^\top \mathbf{w} \geq s\mathbf{1}$). Otherwise, we say the perceptron problem is linearly inseparable.

This is the reason why the threshold parameter δ in Algorithm 1.16 is immaterial, at least in terms of convergence when the problem is indeed linearly separable.

Definition 1.25: Norms and Cauchy-Schwarz inequality

For any vector $\mathbf{w} \in \mathbb{R}^d$, its Euclidean ($\ell_2$) norm (i.e., length) is defined as:

$$\|\mathbf{w}\|_2 := \sqrt{\mathbf{w}^\top \mathbf{w}} = \sqrt{\sum_{j=1}^d |w_j|^2}.$$

More generally, for any $p \geq 1$, we define the ℓ_p norm

$$\|\mathbf{w}\|_p := \Big(\sum_{j=1}^d |w_j|^p\Big)^{1/p}$$

while for $p = \infty$ we define the max norm

$$\|\mathbf{w}\|_\infty := \max_{j=1,\ldots,d} |w_j|.$$

Even more generally, a norm is any function $\|\cdot\| : \mathbb{R}^d \to \mathbb{R}_+$ that satisfies:

- (definite) $\|\mathbf{w}\| = 0 \iff \mathbf{w} = \mathbf{0}$
- (homogeneous) for all $\lambda \in \mathbb{R}$ and $\mathbf{w} \in \mathbb{R}^d$, $\|\lambda\mathbf{w}\| = |\lambda| \cdot \|\mathbf{w}\|$
- (triangle inequality) for all $\mathbf{w}$ and $\mathbf{z} \in \mathbb{R}^d$:

$$\|\mathbf{w} + \mathbf{z}\| \leq \|\mathbf{w}\| + \|\mathbf{z}\|.$$

The norm function is a convenient way to convert a vector quantity to a real number, for instance, to facilitate numerical comparison. Part of the business in machine learning is to understand the effect of different norms on certain learning problems, even though all norms are “formally equivalent:” for any two norms $\|\cdot\|$ and $|\!|\!|\cdot|\!|\!|$, there exist constants $c_d, C_d \in \mathbb{R}$ so that $\forall \mathbf{w} \in \mathbb{R}^d$,

$$c_d\|\mathbf{w}\| \leq |\!|\!|\mathbf{w}|\!|\!| \leq C_d\|\mathbf{w}\|.$$

The subtlety lies on the dependence of the constants c_d, C_d on the dimension d: could be exponential and could affect a learning algorithm a lot.

The dual (norm) $\|\cdot\|_\circ$ of the norm $\|\cdot\|$ is defined as:

$$\|\mathbf{z}\|_\circ := \max_{\|\mathbf{w}\|=1} \mathbf{w}^\top \mathbf{z} = \max_{\mathbf{w}\neq 0} \frac{\mathbf{w}^\top \mathbf{z}}{\|\mathbf{w}\|} = \max_{\|\mathbf{w}\|=1} |\mathbf{w}^\top \mathbf{z}| = \max_{\mathbf{w}\neq 0} \frac{|\mathbf{w}^\top \mathbf{z}|}{\|\mathbf{w}\|}.$$

From the definition it follows the important inequality:

$$\mathbf{w}^\top \mathbf{z} \leq |\mathbf{w}^\top \mathbf{z}| \leq \|\mathbf{w}\| \cdot \|\mathbf{z}\|_\circ.$$

The Cauchy-Schwarz inequality, which will be repeatedly used throughout the course, is essentially a self-duality property of the ℓ_2 norm:

$$\mathbf{w}^\top \mathbf{z} \leq |\mathbf{w}^\top \mathbf{z}| \leq \|\mathbf{w}\|_2 \cdot \|\mathbf{z}\|_2,$$

i.e., the dual norm of the ℓ_2 norm is itself. The dual norm of the ℓ_p norm is the ℓ_q norm, where

$$\infty \geq p, q \geq 1 \text{ and } \frac{1}{p} + \frac{1}{q} = 1.$$

Exercise 1.26: Norms

Prove the following:

- for any $\mathbf{w} \in \mathbb{R}^d$: $\|\mathbf{w}\|_p \to \|\mathbf{w}\|_\infty$ as $p \to \infty$
- for any $\infty \geq p \geq 1$, the ℓ_p norm is indeed a norm. What about $p < 1$?
- the dual of any norm $\|\cdot\|$ is indeed again a norm
- for any $\infty \geq p \geq q \geq 1$, $\|\mathbf{w}\|_p \leq \|\mathbf{w}\|_q \leq d^{\frac{1}{q}-\frac{1}{p}}\|\mathbf{w}\|_p$
- for any $\mathbf{w}, \mathbf{z} \in \mathbb{R}^d$: $\|\mathbf{w}+\mathbf{z}\|_2^2 + \|\mathbf{w}-\mathbf{z}\|_2^2 = 2(\|\mathbf{w}\|_2^2 + \|\mathbf{z}\|_2^2)$.

Remark 1.27: Key insight

The key insight for the success of the perceptron Algorithm 1.16 is the following simple inequality:

$$\langle \mathsf{a}, \mathsf{w}_{t+1} \rangle = \langle \mathsf{a}, \mathsf{w}_t + \mathsf{a} \rangle = \langle \mathsf{a}, \mathsf{w}_t \rangle + \|\mathsf{a}\|_2^2 > \langle \mathsf{a}, \mathsf{w}_t \rangle.$$

(Why can we assume w.l.o.g. that $\|\mathsf{a}\|_2^2 > 0$?) Therefore, if the condition $\langle \mathsf{a}, \mathsf{w}_t \rangle > \delta$ is violated, then we perform an update which brings us strictly closer to satisfy that constraint. [The magic is that by doing so we do not ruin the possibility of satisfying all other constraints, as we shall see.]

This particular update rule of perceptron can be justified as performing stochastic gradient descent on an appropriate objective function, as we are going to see in a later lecture.

Theorem 1.28: Perceptron convergence theorem (Block 1962; Novikoff 1962)

Assuming the data A (see Remark 1.17) is (strictly) linearly separable and denoting w_t the iterate after the t-th update in the perceptron algorithm. Then, $\mathsf{w}_t \to$ some w^* in finite time. If each column of A is selected infinitely often, then $\mathsf{A}^\top \mathsf{w}^* > \delta \mathbf{1}$.

Proof. Under the linearly separable assumption there exists some solution $\mathsf{w}^\star$, i.e., $\mathsf{A}^\top \mathsf{w}^\star \geq s\mathbf{1}$ for some $s > 0$. Then, upon making an update from w_t to w_{t+1} (using the data example denoted as a):

$$\langle \mathsf{w}_{t+1}, \mathsf{w}^\star \rangle = \langle \mathsf{w}_t, \mathsf{w}^\star \rangle + \langle \mathsf{a}, \mathsf{w}^\star \rangle \geq \langle \mathsf{w}_t, \mathsf{w}^\star \rangle + s.$$

Hence, by telescoping we have $\langle \mathsf{w}_t, \mathsf{w}^\star \rangle \geq \langle \mathsf{w}_0, \mathsf{w}^\star \rangle + ts$, which then, using the Cauchy-Schwarz inequality, implies $\|\mathbf{w}_t\|_2 \geq \frac{\langle \mathbf{w}_0, \mathbf{w}^\star\rangle + ts}{\|\mathbf{w}^\star\|_2}$. [Are we certain that $\|\mathbf{w}^\star\|_2 \neq 0$?]

On the other hand, using the fact that we make an update only when $\langle \mathbf{a}, \mathbf{w}_t\rangle \leq \delta$:

$$\|\mathbf{w}_{t+1}\|_2^2 = \|\mathbf{w}_t + \mathbf{a}\|_2^2 = \|\mathbf{w}_t\|_2^2 + 2\langle \mathbf{w}_t, \mathbf{a}\rangle + \|\mathbf{a}\|_2^2 \leq \|\mathbf{w}_t\|_2^2 + 2\delta + \|\mathbf{a}\|_2^2.$$

Hence, telescoping again we have $\|\mathbf{w}_t\|_2^2 \leq \|\mathbf{w}_0\|_2^2 + (2\delta + \|\mathsf{A}\|_{2,\infty}^2)t$, where we use the notation $\|\mathsf{A}\|_{2,\infty} := \max_i \|\mathbf{a}_i\|_2$.

Combine the above two (blue) inequalities: $\frac{\langle \mathbf{w}_0, \mathbf{w}^\star\rangle + ts}{\|\mathbf{w}^\star\|_2} \leq \sqrt{\|\mathbf{w}_0\|_2^2 + (2\delta + \|\mathsf{A}\|_{2,\infty}^2)t}$, solving which gives:

$$t \leq \frac{(2\delta + \|\mathsf{A}\|_{2,\infty}^2)\|\mathbf{w}^\star\|_2^2 + 2s\|\mathbf{w}^\star\|_2\|\mathbf{w}_0\|_2}{s^2}. \tag{1.4}$$

Thus, the perceptron algorithm performs at most a finite number of updates, meaning that $\mathbf{w}_t$ remains unchanged thereafter. □

Typically, we start with $\mathbf{w}_0 = \mathbf{0}$ and we choose $\delta = 0$, then the perceptron algorithm converges after at most $\frac{\|\mathsf{A}\|_{2,\infty}^2 \|\mathbf{w}^\star\|_2^2}{s^2}$ updates.

Block, H. D. (1962). "The perceptron: A model for brain functioning". *Reviews of Modern Physics*, vol. 34, no. 1, pp. 123–135.

Novikoff, A. (1962). "On Convergence proofs for perceptrons". In: *Symposium on Mathematical Theory of Automata*, pp. 615–622.

Exercise 1.29: Data normalization

Suppose the data $\mathcal{D} = \lbag(\mathbf{x}_i, y_i) \in \mathbb{R}^d \times \{\pm 1\} : i = 1, \ldots, n\rbag$ is linearly separable, i.e., there exists some $s > 0$ and $\mathbf{w} = (\mathbf{w}; b)$ such that $\mathsf{A}^\top \mathbf{w} \geq s\mathbf{1}$, where recall that $\mathbf{a}_i = y_i \binom{\mathbf{x}_i}{1}$.

- If we scale each instance $\mathbf{x}_i$ to $\lambda \mathbf{x}_i$ for some $\lambda > 0$, is the resulting data still linearly separable? Does perceptron converge faster or slower after scaling? How does the bound (1.4) change?
- If we translate each instance $\mathbf{x}_i$ to $\mathbf{x}_i + \bar{\mathbf{x}}$ for some $\bar{\mathbf{x}} \in \mathbb{R}^d$, is the resulting data still linearly separable? Does perceptron converge faster or slower after translation? How does the bound (1.4) change?

[Hint: you could consider initializing $\mathbf{w}_0$ differently after the scaling.]

Remark 1.30: Optimizing the bound

As we mentioned above, (for linearly separable data) the perceptron algorithm converges after at most $\frac{\|\mathsf{A}\|_{2,\infty}^2 \|\mathbf{w}^\star\|_2^2}{s^2}$ steps, if we start with $\mathbf{w}_0 = \mathbf{0}$ (and choose $\delta = 0$). Note, however, that the "solution" $\mathbf{w}^\star$ is introduced merely for the analysis of the perceptron algorithm; the algorithm in fact does not "see" it at all. In other words, $\mathbf{w}^\star$ is "fictional," hence we can tune it to optimize our bound as follows:

$$\min_{(\mathbf{w}^\star, s): \mathsf{A}^\top \mathbf{w}^\star \geq s\mathbf{1}} \frac{\|\mathbf{w}^\star\|_2^2}{s^2} = \min_{(\mathbf{w}, s): \|\mathbf{w}\|_2 \leq 1,\ \mathsf{A}^\top \mathbf{w} \geq s\mathbf{1}} \frac{1}{s^2} = \left[\frac{1}{\max_{(\mathbf{w}, s): \|\mathbf{w}\|_2 \leq 1,\ \mathsf{A}^\top \mathbf{w} \geq s\mathbf{1}} s}\right]^2 = \Bigg[\frac{1}{\underbrace{\max_{\mathbf{w}: \|\mathbf{w}\|_2 \leq 1} \min_i \langle \mathbf{a}_i, \mathbf{w}\rangle}_{\text{the margin } \gamma_2}}\Bigg]^2,$$

where we implicitly assumed the denominator is positive (i.e. A is linearly separable). Therefore, the perceptron algorithm (with $\mathbf{w}_0 = \mathbf{0}, \delta = 0$) converges after at most

$$T = T(\mathsf{A}) := \frac{\max_i \|\mathbf{a}_i\|_2^2}{\left(\max_{\|\mathbf{w}\|_2 \leq 1} \min_i \langle \mathbf{a}_i, \mathbf{w}\rangle\right)^2}$$

steps. If we scale the data so that $\|\mathsf{A}\|_{2,\infty} := \max_i \|\mathbf{a}_i\|_2 = 1$, then we have the appealing bound:

$$T = T(\mathsf{A}) = \frac{1}{\gamma_2^2}, \quad \text{where} \quad \gamma_2 = \gamma_2(\mathsf{A}) = \max_{\|\mathbf{w}\|_2 \le 1} \min_i \langle \mathbf{a}_i, \mathbf{w} \rangle \le \min_i \max_{\|\mathbf{w}\|_2 \le 1} \langle \mathbf{a}_i, \mathbf{w} \rangle = \min_i \|\mathbf{a}_i\|_2 \le \|\mathsf{A}\|_{2,\infty} = 1. \tag{1.5}$$

Intuitively, the margin parameter γ_2 characterizes how "linearly separable" a dataset A is, and the perceptron algorithm converges faster if the data is "more" linearly separable!

Remark 1.31: Uniqueness

The perceptron algorithm outputs a solution $\mathbf{w}$ such that $\mathsf{A}\mathbf{w} > \delta\mathbf{1}$, but it does not seem to care which solution to output if there are multiple ones. The iteration bound in (1.5) actually suggests a different algorithm, famously known as the support vector machines (SVM). The idea is simply to find the weight vector $\mathbf{w}$ that attains the margin in (1.5):

$$\max_{\|\mathbf{w}\|_2 \le 1} \min_i \langle \mathbf{a}_i, \mathbf{w} \rangle \iff \min_{\mathbf{w}: \mathsf{A}\mathbf{w} \ge \mathbf{1}} \|\mathbf{w}\|_2^2,$$

where the right-hand side is the usual formula for hard-margin support vector machines (SVM), to be discussed in a later lecture! (Strictly speaking, we need to replace $\|\mathbf{w}\|_2^2$ with $\|\mathbf{w}\|_2^2$, i.e., unregularizing the bias b in SVM.)

Alert 1.32: Notation

For two real numbers $u, v \in \mathbb{R}$, the following standard notations will be used throughout the course:

- $u \vee v := \max\{u, v\}$: maximum of the two
- $u \wedge v := \min\{u, v\}$: minimum of the two
- $u^+ = u \vee 0 = \max\{u, 0\}$: positive part
- $u^- = \max\{-u, 0\}$: negative part

These operations extend straightforwardly in the elementwise manner to two vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^d$.

Exercise 1.33: Decomposition

Prove the following claims (note that the negative part u^- is a positive number by definition):

- $u^+ = (-u)^-$
- $u = u^+ - u^-$
- $|u| = u^+ + u^-$

Theorem 1.34: Optimality of perceptron

Let $n = 1/\gamma^2 \wedge d$. For any deterministic algorithm $\mathcal{A}$, there exists a dataset $\wr(\mathbf{e}_i, y_i)\int_{i=1}^n$ with margin at least γ such that $\mathcal{A}$ makes at least n mistakes on it.

Proof. For any deterministic algorithm $\mathcal{A}$, set $y_i = -\mathcal{A}(\mathbf{e}_1, y_1, \ldots, \mathbf{e}_{i-1}, y_{i-1}, \mathbf{e}_i)$. Clearly, $\mathcal{A}$ makes n mistakes on the dataset $\wr(\mathbf{e}_i, y_i)\int_{i=1}^n$ (due to the hostility of nature in constructing y_i).

We need only verify the margin claim. Let $w_i^\star = y_i\gamma$ (and $b = 0$), then $y_i \langle \mathbf{e}_i, \mathbf{w}^\star \rangle = \gamma$. Thus, the dataset $\{(\mathbf{e}_i, y_i)\}_{i=1}^n$ has margin at least γ. □

Therefore, for high dimensional problems (i.e. large d), the perceptron algorithm achieves the optimal worst-case mistake bound.

Theorem 1.35: Perceptron boundedness theorem (Amaldi and Hauser 2005)

Let $\mathsf{A} \in \mathbb{R}^{p\times n}$ *be a matrix with nonzero columns,* $\mathbf{w}_0 \in \mathbb{R}^p$ *arbitrary,* $\eta_t \in [0, \bar\eta]$*, and define*

$$\mathbf{w}_{t+1} = \mathbf{w}_t + \eta_t \mathbf{a}_{I_t},$$

where $\mathbf{a}_{I_t}$ *is some column of* A *chosen such that* $\langle \mathbf{w}_t, \mathbf{a}_{I_t} \rangle \le 0$. *Then, for all* t,

$$\|\mathbf{w}_t\|_2 \le 2\max\left[\|\mathbf{w}_0\|_2,\quad \bar\eta \max_i \|\mathbf{a}_i\|_2 \times \left((1 \wedge \min_i \|\mathbf{a}_i\|_2)^{\mathrm{rank}(\mathsf{A})} \times \kappa(\mathsf{A}) 2^{3p/2} + 1\right)\right], \tag{1.6}$$

where the condition number

$$\kappa^{-2}(\mathsf{A}) := \min\{\det(B^\top B) : B = [\mathbf{a}_{i_1}, \mathbf{a}_{i_2}, \ldots, \mathbf{a}_{i_{\mathrm{rank}(\mathsf{A})}}] \text{ is a submatrix of } \mathsf{A} \text{ with full column rank}\}.$$

Proof. We omit the somewhat lengthy proof. □

The preceptron algorithm corresponds to $\eta_t \equiv 1$, in which case the boundedness claim (without the quantitative bound (1.6)) was first established in Minsky and Papert (1988, originally published in 1969) and Block and Levin (1970).

Amaldi, Edoardo and Raphael Hauser (2005). "Boundedness Theorems for the Relaxation Method". *Mathematics of Operations Research*, vol. 30, no. 4, pp. 939–955.

Minsky, Marvin L. and Seymour A. Papert (1988, originally published in 1969). *Perceptron*. second expanded. MIT press.

Block, H. D. and S. A. Levin (1970). "On the boundedness of an iterative procedure for solving a system of linear inequalities". *Proceedings of the American Mathematical Society*, vol. 26, pp. 229–235.

Remark 1.36: Reducing multiclass to binary

We can easily adapt the perceptron algorithm to datasets with $c > 2$ classes, using either of the following general reduction schemes:

- one-vs-all: For each class k, use its examples as positive and examples from all other classes as negative, allowing us to train a perceptron with weight $\mathbf{w}_k = [\mathbf{w}_k; b_k]$. Upon receiving a new example $\mathbf{x}$, we predict according to the "winner" of the c perceptrons (break ties arbitrarily):

$$\hat y = \operatorname*{argmax}_{k=1,\ldots,c} \quad \mathbf{w}_k^\top \mathbf{x} + b_k.$$

  The downside of this scheme is that when we train the k-th perceptron, the dataset is imbalanced, i.e. we have much more negatives than positives. The upside is that we only need to train c (or $c-1$ if we set one class as the default) perceptrons.

- one-vs-one: For each pair (k, k') of classes, we train a perceptron $\mathbf{w}_{k,k'}$ where we use examples from class k as positive and examples from class k' as negative. In total we train $\binom{c}{2}$ perceptrons. Upon receiving a new example $\mathbf{x}$, we count how many times each class k is the (binary) prediction:

$$|\{k : \mathbf{x}^\top \mathbf{w}_{k,k'} + b_{k,k'} > 0 \text{ or } \mathbf{x}^\top \mathbf{w}_{k',k} + b_{k',k} \le 0\}|.$$

  Of course, we take again the "winner" as our predicted class. The downside here is we have to train $O(c^2)$ perceptrons while the upside is that each time the training set is more balanced.

Algorithm 1.37: General linear inequalities (Agmon 1954; Motzkin and Schoenberg 1954)

More generally, to solve any (non-homogeneous) linear inequality system

$$\mathsf{A}^\top \mathbf{w} \le \mathbf{c}, \quad \textit{i.e.}, \quad \mathbf{a}_i^\top \mathbf{w} \le c_i, \ i = 1, \ldots, n,$$

we can extend the idea of perceptron to the following projection algorithm:

```
Algorithm: Projection Algorithm for Linear Inequalities
  Input: $\mathsf{A} \in \mathbb{R}^{p\times n}, \mathbf{c} \in \mathbb{R}^n$, initialization $\mathbf{w} \in \mathbb{R}^p$, relaxation parameter $\eta \in (0, 2]$
  Output: approximate solution $\mathbf{w}$
1 for $t = 1, 2, \ldots$ do
2     select an index $I_t \in \{1, \ldots, n\}$                // the index $I_t$ can be random
3     $\mathbf{w} \leftarrow (1-\eta)\mathbf{w} + \eta \left[\mathbf{w} - \frac{(\mathbf{a}_{I_t}^\top \mathbf{w} - c_{I_t})^+}{\langle \mathbf{a}_{I_t}, \mathbf{a}_{I_t}\rangle} \mathbf{a}_{I_t}\right]$
```

The term within the square bracket is exactly the projection of $\mathbf{w}$ onto the halfspace $\mathbf{a}_{I_t}^\top \mathbf{w} \le c_{I_t}$. If we choose $\eta \equiv 1$ then we just repeatedly project $\mathbf{w}$ onto each of the halfspaces. With $\eta \equiv 2$ we actually perform reflections, which, as argued by Motzkin and Schoenberg (1954), can accelerate convergence a lot in certain settings.

Agmon, Shmuel (1954). "The Relaxation Method for Linear Inequalities". *Canadian Journal of Mathematics*, vol. 6, pp. 382–392.

Motzkin, T. S. and I. J. Schoenberg (1954). "The Relaxation Method for Linear Inequalities". *Canadian Journal of Mathematics*, vol. 6, pp. 393–404.

Remark 1.38: Choosing the index

There are a couple of ways to choose the index I_t, i.e., which example we are going to deal with at iteration t:

- cyclic: $I_t = (I_{t-1} + 1) \mod n$.
- chaotic: $\exists \tau \ge n$ so that for any $t \in \mathbb{N}$, $\{1, 2, \ldots, n\} \subseteq \{I_t, I_{t+1}, \ldots, I_{t+\tau-1}\}$.
- randomized: $I_t = i$ with probability p_i. A typical choice is $p_i = \|\mathbf{a}_i\|_2^2 / \sum_\imath \|\mathbf{a}_\imath\|_2^2$.
- permuted: in each epoch randomly permute $\{1, 2, \ldots, n\}$ and then follow cyclic.
- maximal distance: $\frac{(\mathbf{a}_{I_t}^\top \mathbf{w} - c_{I_t})^+}{\|\mathbf{a}_{I_t}\|_2} = \max_{i=1,\ldots,n} \frac{(\mathbf{a}_i^\top \mathbf{w} - c_i)^+}{\|\mathbf{a}_i\|_2}$ (break ties arbitrarily).
- maximal residual: $(\mathbf{a}_{I_t}^\top \mathbf{w} - c_{I_t})^+ = \max_{i=1,\ldots,n} (\mathbf{a}_i^\top \mathbf{w} - c_i)^+$ (break ties arbitrarily).

Remark 1.39: Understanding perceptron mathematically

Let us define a polyhedral cone $\mathrm{cone}(\mathsf{A}) := \{\mathsf{A}\boldsymbol{\lambda} : \boldsymbol{\lambda} \ge \mathbf{0}\}$ whose dual is $[\mathrm{cone}(\mathsf{A})]^* = \{\mathbf{w} : \mathsf{A}^\top \mathbf{w} \ge \mathbf{0}\}$. The linear separability assumption in Definition 1.24 can be written concisely as $\mathrm{int}([\mathrm{cone}(\mathsf{A})]^*) \ne \emptyset$, but it is known in convex analysis that the dual cone $[\mathrm{cone}(\mathsf{A})]^*$ has nonempty interior iff $\mathrm{int}([\mathrm{cone}(\mathsf{A})]^*) \cap \mathrm{cone}(\mathsf{A}) \ne \emptyset$, i.e., iff there exists some $\boldsymbol{\lambda} \ge \mathbf{0}$ so that $\mathbf{w} = \mathsf{A}\boldsymbol{\lambda}$ satisfies $\mathsf{A}^\top \mathbf{w} > \mathbf{0}$. Slightly perturb $\boldsymbol{\lambda}$ we may assume w.l.o.g. $\boldsymbol{\lambda}$ is rational. Perform scaling if necessary we may even assume $\boldsymbol{\lambda}$ is integral. The perceptron algorithm gives a constructive way to find such an *integral* $\boldsymbol{\lambda}$ (hence also $\mathbf{w}$).

Remark 1.40: Variants

We mention some interesting variants of the perceptron algorithm: (Cesa-Bianchi et al. 2005; Dekel et al. 2008; Soheili and Peña 2012; Soheili and Peña 2013).

Cesa-Bianchi, Nicolò, Alex Conconi, and Claudio Gentile (2005). "A Second-Order Perceptron Algorithm". *SIAM Journal on Computing*, vol. 34, no. 3, pp. 640–668.
Dekel, Ofer, Shai Shalev-Shwartz, and Yoram Singer (2008). "The Forgetron: A Kernel-based Perceptron on A Budget". *SIAM Journal on Computing*, vol. 37, no. 5, pp. 1342–1372.
Soheili, Negar and Javier Peña (2012). "A Smooth Perceptron Algorithm". *SIAM Journal on Optimization*, vol. 22, no. 2, pp. 728–737.
— (2013). "A Primal–Dual Smooth Perceptron–von Neumann Algorithm". In: *Discrete Geometry and Optimization*, pp. 303–320.

Remark 1.41: More refined results

A primal-dual version is given in (Spingarn 1985; Spingarn 1987), which solves the general system of linear inequalities in finite time (provided a solution exists in the interior).

For a more refined analysis of the perceptron algorithm and related, see (Goffin 1980; Goffin 1982; Ramdas and Peña 2016; Peña et al. 2021).

Spingarn, Jonathan E. (1985). "A primal-dual projection method for solving systems of linear inequalities". *Linear Algebra and its Applications*, vol. 65, pp. 45–62.
— (1987). "A projection method for least-squares solutions to overdetermined systems of linear inequalities". *Linear Algebra and its Applications*, vol. 86, pp. 211–236.
Goffin, J. L. (1980). "The relaxation method for solving systems of linear inequalities". *Mathematis of Operations Research*, vol. 5, no. 3, pp. 388–414.
— (1982). "On the non-polynomiality of the relaxation method for systems of linear inequalities". *Mathematical Programming*, vol. 22, pp. 93–103.
Ramdas, Aaditya and Javier Peña (2016). "Towards a deeper geometric, analytic and algorithmic understanding of margins". *Optimization Methods and Software*, vol. 31, no. 2, pp. 377–391.
Peña, Javier F., Juan C. Vera, and Luis F. Zuluaga (2021). "New characterizations of Hoffman constants for systems of linear constraints". *Mathematical Programming*, vol. 187, pp. 79–109.

Remark 1.42: Solving conic linear system

The perceptron algorithm can be used to solve linear programs (whose KKT conditions form a system of linear inequalities) and more generally conic linear programs, see (Dunagan and Vempala 2008; Belloni et al. 2009; Peña and Soheili 2016; Peña and Soheili 2017).

Dunagan, John and Santosh Vempala (2008). "A simple polynomial-time rescaling algorithm for solving linear programs". *Mathematical Programming*, vol. 114, no. 1, pp. 101–114.
Belloni, Alexandre, Robert M. Freund, and Santosh Vempala (2009). "An Efficient Rescaled Perceptron Algorithm for Conic Systems". *Mathematics of Operations Research*, vol. 34, no. 3, pp. 621–641.
Peña, Javier and Negar Soheili (2016). "A deterministic rescaled perceptron algorithm". *Mathematical Programming*, vol. 155, pp. 497–510.
— (2017). "Solving Conic Systems via Projection and Rescaling". *Mathematical Programming*, vol. 166, no. 1-2, pp. 87–111.

Remark 1.43: Herding

Some interesting applications of the perceptron algorithm and its boundedness can be found in (Gelfand et al. 2010; Harvey and Samadi 2014), (Briol et al. 2015; Briol et al. 2019; Chen et al. 2016), and (Phillips and Tai 2020; Dwivedi and Mackey 2021; Turner et al. 2021).

Gelfand, Andrew, Yutian Chen, Laurens Maaten, and Max Welling (2010). "On Herding and the Perceptron Cycling Theorem". In: *Advances in Neural Information Processing Systems*.

Harvey, Nick and Samira Samadi (2014). “Near-Optimal Herding”. In: *Proceedings of The 27th Conference on Learning Theory*, pp. 1165–1182.

Briol, François-Xavier, Chris J. Oates, Mark Girolami, Michael A. Osborne, and Dino Sejdinovic (2015). “Frank-Wolfe Bayesian Quadrature: Probabilistic Integration with Theoretical Guarantees”. In: *Advances in Neural Information Processing Systems*.

— (2019). “Probabilistic Integration: A Role in Statistical Computation?” *Statistical Science*, vol. 34, no. 1, pp. 1–22.

Chen, Yutian, Luke Bornn, Nando de Freitas, Mareija Eskelin, Jing Fang, and Max Welling (2016). “Herded Gibbs Sampling”. *Journal of Machine Learning Research*, vol. 17, no. 10, pp. 1–29.

Phillips, Jeff M. and Wai Ming Tai (2020). “Near-Optimal Coresets of Kernel Density Estimates”. *Discrete & Computational Geometry*, vol. 63, pp. 867–887.

Dwivedi, Raaz and Lester Mackey (2021). “Kernel Thinning”. In: *Proceedings of Thirty Fourth Conference on Learning Theory*, pp. 1753–1753.

Turner, Paxton, Jingbo Liu, and Philippe Rigollet (2021). “A Statistical Perspective on Coreset Density Estimation”. In: *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, pp. 2512–2520.

2 Linear Regression

Goal

Understand linear regression for predicting a real response. Regularization and cross-validation.

Alert 2.1: Convention

Gray boxes are not required hence can be omitted for unenthusiastic readers.

This note is likely to be updated again soon.

Some notations and conventions in this note are different from those in the slides.

Definition 2.2: Interpolation

Given a sequence of pairs $\mathcal{D} = \llbracket (\mathbf{x}_i, \mathbf{y}_i) \in \mathbb{R}^d \times \mathbb{R}^r : i = 1, \ldots, n \rrbracket$, we want to find a function $f : \mathbb{R}^d \to \mathbb{R}^r$ so that for all i:

$$f(\mathbf{x}_i) \approx \mathbf{y}_i.$$

Most often, $r = 1$, i.e., each $\mathbf{y}_i$ is real-valued. However, we will indulge ourselves for treating any r (since it brings very minimal complication).

The variable r here stands for the number of responses (tasks), i.e., how many values we are interested in predicting (simultaneously).

Theorem 2.3: Exact interpolation

For any finite number of pairs $\mathcal{D} = \llbracket (\mathbf{x}_i, \mathbf{y}_i) \in \mathbb{R}^d \times \mathbb{R}^r : i = 1, \ldots, n \rrbracket$ *that satisfy* $\mathbf{x}_i = \mathbf{x}_j \implies \mathbf{y}_i = \mathbf{y}_j$, *there exist infinitely many functions* $f : \mathbb{R}^d \to \mathbb{R}^r$ *so that for all* i:

$$f(\mathbf{x}_i) = \mathbf{y}_i.$$

Proof. W.l.o.g. we may assume all $\mathbf{x}_i$'s are distinct. Lagrange polynomials give immediately such a claimed function. More generally, one may put a bump function within a small neighborhood of each $\mathbf{x}_i$ and then glue them together. In details, set $\mathcal{N}_i := \{\mathbf{z} : \|\mathbf{z} - \mathbf{x}_i\|_\infty < \delta\} \subseteq \mathbb{R}^d$. Clearly, $\mathbf{x}_i \in \mathcal{N}_i$ and for δ sufficiently small, $\mathcal{N}_i \cap \mathcal{N}_j = \emptyset$. Define

$$f_i(\mathbf{z}) = \begin{cases} \mathbf{y}_i e^{d/\delta^2} \prod_{j=1}^d \exp\left(-\frac{1}{\delta^2 - (z_j - x_{ji})^2}\right), & \text{if } \mathbf{z} \in \mathcal{N}_i \\ 0, & \text{otherwise} \end{cases}.$$

The function $f = \sum_i f_i$ again exactly interpolates our data $\mathcal{D}$. □

The condition $\mathbf{x}_i = \mathbf{x}_j \implies \mathbf{y}_i = \mathbf{y}_j$ is clearly necessary, for otherwise there cannot be any function so that $\mathbf{y}_i = f(\mathbf{x}_i) = f(\mathbf{x}_j) = \mathbf{y}_j$ and $\mathbf{y}_i \neq \mathbf{y}_j$. Of course, when all $\mathbf{x}_i$'s are distinct, this condition is trivially satisfied.

Exercise 2.4: From 1 to ∞

Complete the proof of Theorem 2.3 with regard to the "infinite" part.

Remark 2.5: Infinitely many choices...

Theorem 2.3 has the following important implication: Given a finite training set $\mathcal{D}$, no matter how large its size might be, there exist infinitely many *smooth* (infinitely differentiable) functions f that maps each $\mathbf{x}_i$ in $\mathcal{D}$ *exactly* to $\mathbf{y}_i$, i.e. they all achieve zero training "error." However, on a new test instance $\mathbf{x} \not\in \mathcal{D}$, the predictions $\hat{\mathbf{y}} = f(\mathbf{x})$ of different choices of f can be wildly different (in fact, we can make $f(\mathbf{x}) = \mathbf{y}$ for any $\mathbf{y} \in \mathbb{R}^r$).

Which function should we choose then?

Definition 2.6: Least squares regression

To resolve the difficulty in Remark 2.5, we need some statistical assumption on how our data is generated. In particular, we assume $(\mathbf{X}_i, \mathbf{Y}_i)$ are independently and identically distributed (i.i.d.) random samples from an unknown distribution $\mathbb{P}$. The test sample $(\mathbf{X}, \mathbf{Y})$ is also drawn independently and identically from the same distribution $\mathbb{P}$. We are interested in solving the least squares regression problem:

$$\min_{f:\mathbb{R}^d \to \mathbb{R}^r} \quad \mathsf{E}\|\mathbf{Y} - f(\mathbf{X})\|_2^2, \tag{2.1}$$

i.e., finding a function $f : \mathbb{R}^d \to \mathbb{R}^r$ so that $f(\mathbf{X})$ approximates $\mathbf{Y}$ well in expectation. (Strictly speaking we need the technical assumption that f is measurable so that the above expectation is even defined.)

In reality, we do not know the distribution $\mathbb{P}$ of $(\mathbf{X}, \mathbf{Y})$ hence will not be able to compute the expectation, let alone minimizing it. Instead, we use the training set $\mathcal{D} = \{(\mathbf{X}_i, \mathbf{Y}_i) : i = 1, \ldots, n\}$ to *approximate* the expectation:

$$\min_{f:\mathbb{R}^d \to \mathbb{R}^r} \quad \hat{\mathsf{E}}\|\mathbf{Y} - f(\mathbf{X})\|_2^2 := \frac{1}{n}\sum_{i=1}^{n} \|\mathbf{Y}_i - f(\mathbf{X}_i)\|_2^2. \tag{2.2}$$

By law of large numbers, for any *fixed* function f, we indeed have

$$\frac{1}{n}\sum_{i=1}^{n} \|\mathbf{Y}_i - f(\mathbf{X}_i)\|_2^2 \xrightarrow{\text{as } n\to\infty} \mathsf{E}\|\mathbf{Y} - f(\mathbf{X})\|_2^2.$$

Remark 2.7: Properties of (conditional) expectation

- The expectation $\mathsf{E}[\mathbf{Y}]$ intuitively is the (elementwise) average value of the random variable $\mathbf{Y}$.
- The conditional expectation $\mathsf{E}[\mathbf{Y}|\mathbf{X}]$ is an (equivalent class of) real-valued function(s) of $\mathbf{X}$.
- $\mathsf{E}[f(\mathbf{X})|\mathbf{X}] = f(\mathbf{X})$ (almost everywhere). Intuitively, conditioned on $\mathbf{X}$, $f(\mathbf{X})$ is not random hence the (conditional) expectation is vacuous.
- Law of total expectation: $\mathsf{E}[\mathbf{Y}] = \mathsf{E}[\mathsf{E}(\mathbf{Y}|\mathbf{X})]$ for any random variable $\mathbf{Y}$ and $\mathbf{X}$. Intuitively, the left hand side is the average value of $\mathbf{Y}$, while the right hand side is the average of averages: $\mathsf{E}(\mathbf{Y}|\mathbf{X})$ is the average of $\mathbf{Y}$ for a given realization of $\mathbf{X}$. For instance, take $\mathbf{Y}$ to be the height of a human being and $\mathbf{X}$ to be the gender. Then, the average height of a human (left hand side) is equal to the average of the average heights of man and woman (right hand side).

Definition 2.8: Regression function

For a moment let us assume the distribution $\mathbb{P}$ is known to us, so we can at least in theory solve the least squares regression problem (2.1). It turns out there exists an optimal solution whose closed-form expression can be derived as follows:

$$\begin{aligned}
\mathsf{E}\|\mathbf{Y} - f(\mathbf{X})\|_2^2 &= \mathsf{E}\|\mathbf{Y} - \mathsf{E}(\mathbf{Y}|\mathbf{X}) + \mathsf{E}(\mathbf{Y}|\mathbf{X}) - f(\mathbf{X})\|_2^2 \\
&= \mathsf{E}\|\mathbf{Y} - \mathsf{E}(\mathbf{Y}|\mathbf{X})\|_2^2 + \mathsf{E}\|\mathsf{E}(\mathbf{Y}|\mathbf{X}) - f(\mathbf{X})\|_2^2 + 2\mathsf{E}[\langle \mathbf{Y} - \mathsf{E}(\mathbf{Y}|\mathbf{X}), \mathsf{E}(\mathbf{Y}|\mathbf{X}) - f(\mathbf{X})\rangle] \\
&= \mathsf{E}\|\mathbf{Y} - \mathsf{E}(\mathbf{Y}|\mathbf{X})\|_2^2 + \mathsf{E}\|\mathsf{E}(\mathbf{Y}|\mathbf{X}) - f(\mathbf{X})\|_2^2 + 2\mathsf{E}\Big[\mathsf{E}[\langle \mathbf{Y} - \mathsf{E}(\mathbf{Y}|\mathbf{X}), \mathsf{E}(\mathbf{Y}|\mathbf{X}) - f(\mathbf{X})\rangle \,|\mathbf{X}]\Big] \\
&= \mathsf{E}\|\mathbf{Y} - \mathsf{E}(\mathbf{Y}|\mathbf{X})\|_2^2 + \mathsf{E}\|\mathsf{E}(\mathbf{Y}|\mathbf{X}) - f(\mathbf{X})\|_2^2 + 2\mathsf{E}\Big[\,\langle \mathsf{E}[\mathbf{Y} - \mathsf{E}(\mathbf{Y}|\mathbf{X})|\mathbf{X}], \mathsf{E}(\mathbf{Y}|\mathbf{X}) - f(\mathbf{X})\rangle\,\Big] \\
&= \mathsf{E}\|\mathbf{Y} - \mathsf{E}(\mathbf{Y}|\mathbf{X})\|_2^2 + \mathsf{E}\|\mathsf{E}(\mathbf{Y}|\mathbf{X}) - f(\mathbf{X})\|_2^2,
\end{aligned}$$

whence the regression function

$$f^\star(\mathbf{X}) := \mathsf{E}(\mathbf{Y}|\mathbf{X}) \tag{2.3}$$

eliminates the second nonnegative term while the first nonnegative term is not affected by any f at all.

With hindsight, it is not surprising the regression function $f^\star$ is an optimal solution for the least squares regression problem: it basically says given $(\mathbf{X}, \mathbf{Y})$, we set $f^\star(\mathbf{X}) = \mathbf{Y}$ if there is a unique value of $\mathbf{Y}$ asscoiated with $\mathbf{X}$ (which of course is optimal), while if there are multiple values of $\mathbf{Y}$ associated to the given $\mathbf{X}$, then we simply average them.

The constant term $\mathsf{E}\|\mathbf{Y} - \mathsf{E}(\mathbf{Y}|\mathbf{X})\|_2^2$ describes the difficulty of our regression problem: no function f can reduce it.

Definition 2.9: Overfitting, underfitting, and regularization

All regression methods are in one way or another trying to approximate the regression function (2.3). In reality, we only have access to an i.i.d. training set, but if we solve (2.2) naively we will run into again the difficulty in Remark 2.5: we achieve very small (or even zero) error on the training set but the performance on a test sample can be very bad. This phenomenon is known as overfitting, i.e. we are taking our training set "too seriously." Remember in machine learning we are not interested in doing well on the training set at all (even though training data is all we got, oh life!); training set is used only as a means to get good performance on (future) test set.

Overfitting arises here because we are considering all functions $f : \mathbb{R}^d \to \mathbb{R}^r$, which is a *huge* class, while we only have limited (finite) training examples. In other words, we do not have enough training data to support our ambition. To address overfitting, we will restrict ourselves to a subclass $\mathcal{F}_n$ of functions $\mathbb{R}^d \to \mathbb{R}^r$ and solve:

$$\min_{f\in\mathcal{F}_n} \ \frac{1}{n}\sum_{i=1}^{n} \|\mathbf{Y}_i - f(\mathbf{X}_i)\|^2. \tag{2.4}$$

In other words, we regularize our choice of candidate functions f to avoid fitting too well on the training set. Typically, $\mathcal{F}_n$ grows as n increases, i.e. with more training data we could allow us to consider more candidate functions. On the other hand, if there are too few candidate functions in $\mathcal{F}_n$ (for instance when $\mathcal{F}_n$ consists of all constant functions), then we may not be able to do well even on the training set. This phenomemon is known as underfitting. Generally speaking, doing too well or too badly on the training set are both indications of poor design (either more data needs to be collected or a larger/smaller function class needs to be considered).

Remark 2.10: Approximation vs. estimation and bias vs. variance

It is possible that the regression function $f^\star$ (see (2.3)), the object we are trying to find, is not in the chosen function class $\mathcal{F}_n$ at all. This results in the so-called approximation error (which solely depends on the function class $\mathcal{F}_n$ but not training data), or bias in statistical terms. Clearly, the "larger" $\mathcal{F}_n$ is, the smaller the approximation error. On the other hand, finding the "optimal" $f \in \mathcal{F}_n$ based on (2.4) is more challenging for a larger $\mathcal{F}_n$ (when n is fixed). This is called estimation error, e.g. the error due to using a finite training set (random sampling). Typically, the "larger" $\mathcal{F}_n$ is, the larger the estimation error is. Often, with a "larger" $\mathcal{F}_n$, the performance on the test set has more variation (had we repeated with different training data) .

Much of the work on regression is about how to balance between the approximation error and estimation error, a.k.a. the bias and variance trade-off.

Definition 2.11: Linear least squares regression

The simplest choice for the function class $\mathcal{F}_n \equiv \mathcal{F}$ is perhaps the class of linear/affine functions (recall Definition 1.13). Adopting this choice leads to the linear least squares regression problem:

$$\min_{W \in \mathbb{R}^{r\times d}, \mathbf{b} \in \mathbb{R}^r} \frac{1}{n} \sum_{i=1}^n \|\mathbf{Y}_i - W\mathbf{X}_i - \mathbf{b}\|_2^2, \tag{2.5}$$

where recall that $\mathbf{X}_i \in \mathbb{R}^d$ and $\mathbf{Y}_i \in \mathbb{R}^r$.

Exercise 2.12: Linear functions may not exactly interpolate

Show that Theorem 2.3 fails if we are only allowed to use linear/affine functions.

Definition 2.13: Matrix norms

For a matrix $W \in \mathbb{R}^{r\times d}$, we define its Frobenius norm

$$\|W\|_{\mathsf{F}} = \sqrt{\sum_{ij} w_{ij}^2},$$

which is essentially the matrix analogue of the vector Euclidean norm $\|\mathbf{w}\|_2$.

Another widely used matrix norm is the spectral norm

$$\|W\|_{\mathrm{sp}} = \max_{\|\mathbf{x}\|_2=1} \|W\mathbf{x}\|_2,$$

which coincides with the largest singular value of W.

It is known that

- $\|W\|_{\mathrm{sp}} \le \|W\|_{\mathsf{F}} \le \sqrt{\mathrm{rank}(W)}\|W\|_{\mathrm{sp}}$,
- $\|W\mathbf{x}\|_2 \le \|W\|_{\mathrm{sp}}\|\mathbf{x}\|_2$.

Remark 2.14: Padding again

We apply the same padding trick as in Remark 1.17. Define $\mathsf{x}_i = \binom{\mathbf{X}_i}{1}$ and $\mathsf{X} = [\mathsf{x}_1, \ldots, \mathsf{x}_n] \in \mathbb{R}^{p\times n}$, $Y = [\mathbf{Y}_1, \ldots, \mathbf{Y}_n] \in \mathbb{R}^{r\times n}$, and $\mathsf{W} = [W, \mathbf{b}] \in \mathbb{R}^{r\times p}$, where of course $p = d + 1$. We can then rewrite the linear least squares problem in the following (equivalent but prettier) matrix format:

$$\min_{\mathsf{W}\in\mathbb{R}^{r\times p}} \quad \tfrac{1}{n}\|Y-\mathsf{W}\mathsf{X}\|_{\mathsf{F}}^2. \tag{2.6}$$

Remark 2.15: Pseudo-inverse

Recall that the Moore-Penrose pseudo-inverse $A^\dagger$ of any matrix $A\in\mathbb{R}^{p\times n}$ is the unique matrix $G\in\mathbb{R}^{n\times p}$ so that

$$AGA=A,\quad GAG=G,\quad (AG)^\top=AG,\quad (GA)^\top=GA.$$

In particular, if $A=USV^\top$ is the thin SVD (singular value decomposition) of A, then $A^\dagger=VS^{-1}U^\top$. If A is in fact invertible, then $A^\dagger=A^{-1}$.

We can use pseudo-inverse to solve the linear least squares problem (2.6). Indeed, it is known that $W^\star := A^\dagger CB^\dagger$ is a solution for the minimization problem:

$$\min_W \quad \|AWB-C\|_{\mathsf{F}}^2.$$

In fact, $W^\star$ is the unique solution that enjoys the minimum Frobenius norm. See this lecture note for proofs and more related results.

Equipped with the above result, we know that $\mathsf{W}^\star=Y\mathsf{X}^\dagger$ is a closed-form solution for the linear least squares problem (2.6).

Definition 2.16: Gradient and Hessian (for the brave hearts)

Recall that the gradient of a smooth function $f:\mathbb{R}^d\to\mathbb{R}^r$ at $\mathbf{w}$ is defined as the linear mapping $\nabla f(\mathbf{w}):\mathbb{R}^d\to\mathbb{R}^r$ so that:

$$\lim_{\mathbf{0}\neq\Delta\mathbf{w}\to\mathbf{0}} \frac{\|f(\mathbf{w}+\Delta\mathbf{w})-f(\mathbf{w})-[\nabla f(\mathbf{w})](\Delta\mathbf{w})\|}{\|\Delta\mathbf{w}\|}=0, \tag{2.7}$$

where say the norm $\|\cdot\|$ is the Euclidean norm. Or equivalently in big-o notation:

$$\|f(\mathbf{w}+\Delta\mathbf{w})-f(\mathbf{w})-[\nabla f(\mathbf{w})](\Delta\mathbf{w})\|=o(\|\Delta\mathbf{w}\|).$$

As $\mathbf{w}$ varies, we may think of the gradient as the (nonlinear) mapping:

$$\nabla f:\mathbb{R}^d\to\mathcal{L}(\mathbb{R}^d,\mathbb{R}^r),\quad \mathbf{w}\mapsto\{\nabla f(\mathbf{w}):\mathbb{R}^d\to\mathbb{R}^r\}$$

where $\mathcal{L}(\mathbb{R}^d,\mathbb{R}^r)$ denotes the class of linear mappings from $\mathbb{R}^d$ to $\mathbb{R}^r$, or equivalently the class of matrices $\mathbb{R}^{r\times d}$.

We can iterate the above definition. In particular, replacing f with ∇f we define the Hessian $\nabla^2 f:\mathbb{R}^d\to\mathcal{L}(\mathbb{R}^d,\mathcal{L}(\mathbb{R}^d,\mathbb{R}^r))\simeq\mathcal{B}(\mathbb{R}^d\times\mathbb{R}^d,\mathbb{R}^r)$ of f as the gradient of the gradient ∇f, where $\mathcal{B}(\mathbb{R}^d\times\mathbb{R}^d,\mathbb{R}^r)$ denotes the class of bilinear mappings from $\mathbb{R}^d\times\mathbb{R}^d$ to $\mathbb{R}^r$.

Definition 2.17: Gradient and Hessian through partial derivatives

Let $f:\mathbb{R}^d\to\mathbb{R}$ be a real-valued smooth function. We can define its gradient through partial derivatives:

$$[\nabla f(\mathbf{w})]_j=\frac{\partial f}{\partial w_j}(\mathbf{w}).$$

Note that the gradient $\nabla f(\mathbf{w})\in\mathbb{R}^d$ has the same size as the input $\mathbf{w}$.

Similarly, we can define the Hessian through partial derivatives:

$$[\nabla^2 f(\mathbf{w})]_{ij} = \frac{\partial^2 f}{\partial w_i \partial w_j}(\mathbf{w}) = \frac{\partial^2 f}{\partial w_j \partial w_i}(\mathbf{w}) = [\nabla^2 f(\mathbf{w})]_{ji},$$

where the second equality holds as long as f is twice-differentiable. Note that the Hessian is a symmetric matrix $\nabla^2 f(\mathbf{w}) \in \mathbb{R}^{d\times d}$ with the same number of rows/columns as the size of the input $\mathbf{w}$.

Remark 2.18: Matrix input

If $f : \mathbb{R}^{r\times p} \to \mathbb{R}$ takes a matrix as input, then its gradient is computed similarly as in Definition 2.17:

$$[\nabla f(\mathsf{W})]_{ij} = \frac{\partial f}{\partial w_{ij}}(\mathsf{W}).$$

Again, the gradient $\nabla f(\mathsf{W}) \in \mathbb{R}^{r\times p}$ has the same size as the input W.

In principle, computing the Hessian is completely similar, although we will need 4 indices and the notation can get quite messy quickly. Fortunately, we will rarely find ourselves facing this challenge.

Example 2.19: Quadratic function

Consider the following quadratic function

$$f : \mathbb{R}^d \to \mathbb{R}, \quad \mathbf{w} \mapsto \mathbf{w}^\top Q\mathbf{w} + \mathbf{p}^\top \mathbf{w} + \alpha, \tag{2.8}$$

where $Q \in \mathbb{R}^{d\times d}$, $\mathbf{p} \in \mathbb{R}^d$, and $\alpha \in \mathbb{R}$. We can write explicitly:

$$f(\mathbf{w}) = \alpha + \sum_{k=1}^d w_k \left[p_k + \sum_{l=1}^d q_{kl} w_l\right],$$

whence follows

$$\begin{aligned}[\nabla f(\mathbf{w})]_j = \frac{\partial f}{\partial w_j}(\mathbf{w}) &= \sum_{k=1}^d \left[\frac{\partial w_k}{\partial w_j}\left(p_k + \sum_{l=1}^d q_{kl} w_l\right) + w_k \frac{\partial \left(p_k + \sum_{l=1}^d q_{kl} w_l\right)}{\partial w_j}\right] \\ &= \left(p_j + \sum_{l=1}^d q_{jl} w_l\right) + \sum_{k=1}^d w_k q_{kj} \\ &= p_j + [(Q + Q^\top)\mathbf{w}]_j.\end{aligned}$$

That is, collectively

$$\nabla f(\mathbf{w}) = \mathbf{p} + (Q + Q^\top)\mathbf{w}.$$

Similarly,

$$[\nabla^2 f(\mathbf{w})]_{ij} = \frac{\partial^2 f}{\partial w_i \partial w_j}(\mathbf{x}) = \frac{\partial \left(p_j + \sum_{l=1}^d q_{jl} w_l + \sum_{k=1}^d w_k q_{kj}\right)}{\partial w_i} = q_{ji} + q_{ij}.$$

That is, collectively

$$\nabla^2 f(\mathbf{w}) \equiv Q + Q^\top.$$

The formula further simplifies if we assume Q is symmetric, i.e. $Q = Q^\top$ (which is usually the case).

As demonstrated above, using partial derivatives to derive the gradient and Hessian is straightforward but tedious. Fortunately, we need only do this once: derive and memorize a few, and then resort to the chain rule.

Example 2.20: Quadratic function (for the brave hearts)

Another way to derive the gradient and Hessian is to guess and then verify the definition (2.7). Using the quadratic function (2.8) again as an example. We need to verify:

$$\begin{aligned} o(\|\Delta\mathbf{w}\|) &= \|f(\mathbf{w}+\Delta\mathbf{w}) - f(\mathbf{w}) - [\nabla f(\mathbf{w})](\Delta\mathbf{w})\| \\ &= \|(\mathbf{w}+\Delta\mathbf{w})^\top Q(\mathbf{w}+\Delta\mathbf{w}) + \mathbf{p}^\top(\mathbf{w}+\Delta\mathbf{w}) - \mathbf{w}^\top Q\mathbf{w} - \mathbf{p}^\top\mathbf{w} - [\nabla f(\mathbf{w})](\Delta\mathbf{w})\| \\ &= \|\mathbf{w}^\top Q\Delta\mathbf{w} + (\Delta\mathbf{w})^\top Q\mathbf{w} + \mathbf{p}^\top\Delta\mathbf{w} - [\nabla f(\mathbf{w})](\Delta\mathbf{w}) + (\Delta\mathbf{w})^\top Q(\Delta\mathbf{w})\|, \end{aligned}$$

whence we guess

$$\nabla f(\mathbf{w}) = \mathbf{p} + (Q + Q^\top)\mathbf{w}$$

so that we can cancel out the first four terms (that are all linear in $\Delta\mathbf{w}$). It is then easy to verify that the remaining term indeed satisfies

$$\|(\Delta\mathbf{w})^\top Q(\Delta\mathbf{w})\| = o(\|\Delta\mathbf{w}\|).$$

Similarly, we can guess and verify the Hessian:

$$\begin{aligned} o(\|\Delta\mathbf{w}\|) &= \|\nabla f(\mathbf{w}+\Delta\mathbf{w}) - \nabla f(\mathbf{w}) - [\nabla^2 f(\mathbf{w})](\Delta\mathbf{w})\| \\ &= \|(Q+Q^\top)\Delta\mathbf{w} - [\nabla^2 f(\mathbf{w})](\Delta\mathbf{w})\|, \end{aligned}$$

from which it is clear that $\nabla^2 f(\mathbf{w}) = Q + Q^\top$ would do.

Alert 2.21: Symmetric gradient

Let us consider the function of a symmetric matrix:

$$f(X) := \mathrm{tr}(AX),$$

where both A and X are symmetric. What is the gradient?

$$X \qquad vs. \qquad 2X - \mathrm{diag}(X).$$

(representation of the derivative depends on the underlying inner product. if we identify the symmetric matrix $[a, b; b, c]$ with the vector $[a, b, c]$, we are equipping the latter with the dot product $aa' + cc' + 2bb'$.)

Theorem 2.22: Fermat's necessary condition for extrema (recalled)

A necessary condition for $\mathbf{w}$ to be a local minimizer of a differentiable function $f : \mathbb{R}^d \to \mathbb{R}$ is

$$\nabla f(\mathbf{w}) = \mathbf{0}.$$

(Such points are called stationary, a.k.a. critical.) For convex f the necessary condition is also sufficient.

Proof. See the lecture note. □

Take $f(w) = w^3$ and $w = 0$ we see that this necessary condition is not sufficient for nonconvex functions. For local maximizers, we simply negate the function and apply the theorem to $-f$ instead.

Alert 2.23: Existence of a minimizer

Use the geometric mean as an example for the inapplicability of Fermat's condition when the minimizer does not exist.

Definition 2.24: Normal equation

We may now solve the linear least squares problem (2.6). Simply compute its gradient (by following Example 2.19) and set it to $\mathbf{0}$, as suggested by Theorem 2.22. Upon simplifying, we obtain the so-called normal equation (for the unknown variable $\mathsf{W} \in \mathbb{R}^{r\times p}$):

$$\mathsf{W}\mathsf{X}\mathsf{X}^\top = Y\mathsf{X}^\top, \text{ or after transposing } \mathsf{X}\mathsf{X}^\top\mathsf{W}^\top = \mathsf{X}Y^\top. \tag{2.9}$$

This is a system of linear equations, which we can solve using standard numerical linear algebra toolboxes (Cholesky decomposition in this case). The time complexity is $O(p^3 + p^2n + pnr)$. For large n or p, we may use iterative algorithms (e.g. conjugate gradient) to directly but approximately solve (2.6).

Exercise 2.25: Linear least squares is linear

Based on the original linear least squares formula (2.6), or the normal equation (2.9) (or the more direct solution $Y\mathsf{X}^\dagger$ using pseudo-inverse), prove the following equivariance property of linear least squares:

- If we apply a nonsingular transformation $T \in \mathbb{R}^{p\times p}$ to X (or X), what would happen to the linear least squares solution W?
- If we apply a nonsingular transformation $T \in \mathbb{R}^{r\times r}$ to Y, what would happen to the linear least squares solution W?

Definition 2.26: Prediction

Once we have the linear least squares solution $\hat{\mathsf{W}} = (\hat{W}, \hat{\mathbf{b}})$, we perform prediction on a (future) test sample $\mathbf{X}$ naturally by:

$$\hat{\mathbf{Y}} := \hat{W}\mathbf{X} + \hat{\mathbf{b}}.$$

We measure the "goodness" of our prediction $\hat{\mathbf{Y}}$ by:

$$\|\hat{\mathbf{Y}} - \mathbf{Y}\|_2^2,$$

which is usually averaged over a test set.

Alert 2.27: Calibration

Note that we used the squared loss $(y, \hat{y}) \mapsto (y - \hat{y})^2$ in training linear least squares regression, see (2.5). Thus, naturally, when evaluating the linear least squares solution $\hat{\mathsf{W}}$ on a test set, we should use the same squared loss. If we use a different loss, such as the absolute error $|\hat{y} - y|$, then our training procedure may be suboptimal. Be consistent in terms of the training objective and the test measure!

Nevertheless, sometimes it might be necessary to use one loss $\ell_1(\hat{y}, y)$ for training and another one $\ell_2(\hat{y}, y)$ for testing. For instance, ℓ_1 may be easier to handle computationally. The theory of calibration studies when a minimizer under the loss ℓ_1 remains optimal under a different loss ℓ_2. See the (heavy) paper of Steinwart (2007) and some particular refinement in Long and Servedio (2013) for more details.

Steinwart, Ingo (2007). "How to compare different loss functions and their risks". *Constructive Approximation*, vol. 26, no. 2, pp. 225–287.

Long, Philip M. and Rocco A. Servedio (2013). "Consistency versus Realizable H-Consistency for Multiclass Classification". In: *Proceedings of the 31st International Conference on Machine Learning (ICML)*.

Definition 2.28: Ridge regression with Tikhonov regularization (Hoerl and Kennard 1970)

The class of linear functions may still be too large, leading linear least squares to overfit or instable. We can then put some extra restriction, such as the Tikhonov regularization in ridge regression:

$$\min_{\mathsf{W}\in\mathbb{R}^{r\times p}} \ \tfrac{1}{n}\|Y - \mathsf{WX}\|_{\mathsf{F}}^2 + \lambda\|\mathsf{W}\|_{\mathsf{F}}^2, \tag{2.10}$$

where $\lambda \geq 0$ is the regularization constant (hyperparameter) that balances the two terms.

To understand ridge regression, consider

- when λ is small, thus we are neglecting the second regularization term, and the solution resembles that of the ordinary linear least squares solution;
- when λ is large, thus we are neglecting the first data term, and the solution degenerates to $\mathbf{0}$.

In the literature, the following variant that chooses *not* to regularize the bias term $\mathbf{b}$ is also commonly used:

$$\min_{\mathsf{W}=[W,\mathbf{b}]} \ \tfrac{1}{n}\|Y - \mathsf{WX}\|_{\mathsf{F}}^2 + \lambda\|W\|_{\mathsf{F}}^2. \tag{2.11}$$

Hoerl, Arthur E. and Robert W. Kennard (1970). "Ridge regression: biased estimation for nonorthogonal problems". *Technometrics*, vol. 12, no. 1, pp. 55–67.

Exercise 2.29: Solution for ridge regression

Prove that the unique solution for (2.10) is given by the following normal equation:

$$(\mathsf{XX}^\top + n\lambda I)\mathsf{W}^\top = \mathsf{X}Y^\top,$$

where I is the identity matrix of appropriate size.

- Follow the derivation in Definition 2.24 by computing the gradient and setting it to $\mathbf{0}$.
- Use the trick you learned in elementary school, completing the square, to reduce ridge regression to the ordinary linear least squares regression.
- Data augmentation: ridge regression is equivalent to the ordinary linear least squares regression (2.6) with

$$\mathsf{X} \leftarrow [\mathsf{X}, \sqrt{n\lambda}I_{p\times p}], \quad Y \leftarrow [Y, \mathbf{0}_{r\times p}].$$

How about the solution for the variant (2.11)?

Remark 2.30: Equivalence between regularization and constraint

The regularized problem

$$\min_{\mathsf{W}} \ \ell(\mathsf{W}) + \lambda \cdot r(\mathsf{W}) \tag{2.12}$$

is "equivalent" to the following constrained problem:

$$\min_{r(\mathsf{W})\leq c} \ \ell(\mathsf{W}) \tag{2.13}$$

in the sense that

- for any $\lambda \geq 0$ and any solution W of (2.12) there exists some c so that W remains to be a solution for (2.13);
- under mild conditions, for any c there exists some $\lambda \geq 0$ so that any solution W of (2.13) remains to be a solution for (2.12).

The regularized problem (2.12) is computationally more appealing as it is *unconstrained* while the constrained problem (2.13) is more intuitive and easier to analyze theoretically. Using this equivalence we see that ridge regression essentially performs linear least squares regression over the restricted class of linear functions whose weights are no larger than some constant c.

Exercise 2.31: Is ridge regression linear?

Redo Exercise 2.25 with ridge regression.

Remark 2.32: Choosing the regularization constant

The regularization constant $\lambda \geq 0$ balances the data term and the regularization term in ridge regression (2.10). A typical way to select λ is through a validation set $\mathcal{V}$ that is different from the training set $\mathcal{D}$ and the test set $\mathcal{T}$. For each λ in a candidate set $\Lambda \subseteq \mathbb{R}_+$ (chosen by experience or trial and error), we train our algorithm on $\mathcal{D}$ and get $\hat{\mathsf{W}}_\lambda$, evaluate the performance of $\hat{\mathsf{W}}_\lambda$ on $\mathcal{V}$ (according to some metric perf), choose the best $\hat{\mathsf{W}}_\lambda$ (w.r.t. perf) and finally apply it on the test set $\mathcal{T}$. Importantly,

- One should never "see" the test set during training and parameter tuning; otherwise we are cheating and bound to overfit.
- The validation set $\mathcal{V}$ should be different from both the training set and the test set.
- We can only use the validation set once in parameter tuning. Burn After Reading! (However, see the recent work of Dwork et al. (2017) and the references therein for data re-usage.)
- As mentioned in Alert 2.27, the performance measure perf should ideally align with the training objective.

Dwork, Cynthia, Vitaly Feldman, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Aaron Roth (2017). "Guilt-free Data Reuse". *Communications of the ACM*, vol. 60, no. 4, pp. 86–93.

Algorithm 2.33: Cross-validation

When data is limited, we do not have the luxury to afford a separate validation set, in which case we may use the cross-validation procedure for hyperparameter selection. Essentially we split the training set and hold a (small) part out as validation set. However, to avoid random fluctuations (especially when the validation set is small), we repeat the split k times and average the results.

Algorithm: Cross-Validation

```
  Input: candidate regularization constants $\Lambda \subseteq \mathbb{R}$, performance metric perf, $k \in \mathbb{N}$, training data $\mathcal{D}$
  Output: best regularization constant $\lambda^\star$
1 randomly partition $\mathcal{D}$ into similarly sized $k$ subsets $\mathcal{D}_1, \ldots, \mathcal{D}_k$     // e.g. $|\mathcal{D}_l| = \lfloor |\mathcal{D}|/k \rfloor, \forall l < k$
2 for $\lambda \in \Lambda$ do
3     $p(\lambda) \leftarrow 0$
4     for $l = 1, \ldots, k$ do
5         train on $\mathcal{D}_{\neg l} := \cup_{j \neq l} \mathcal{D}_j = \mathcal{D} \setminus \mathcal{D}_l$ and get $\hat{\mathsf{W}}_{\lambda, \neg l}$
6         $p(\lambda) \leftarrow p(\lambda) + \mathsf{perf}(\hat{\mathsf{W}}_{\lambda,\neg l}, \mathcal{D}_l)$     // evaluate $\hat{\mathsf{W}}_{\lambda,\neg l}$ on the holdout set $\mathcal{D}_l$
7 $\lambda^\star \leftarrow \mathrm{argmin}_{\lambda \in \Lambda} p(\lambda)$     // assuming the smaller perf the better
```

With the "optimal" $\lambda^\star$ at hand, we re-train on the entire training set $\mathcal{D}$ to get $\hat{\mathsf{W}}_{\lambda^\star}$ and then evaluate it on the test set using the same performance measure $\mathsf{perf}(\hat{\mathsf{W}}_{\lambda^\star}, \mathcal{T})$.

In the extreme case where $k = |\mathcal{D}|$, each time we train on the entire training set except one data instance. This is called leave-one out cross-validation. Typically we set $k = 10$ or $k = 5$. Note that the larger k is, the more expensive (computationally) cross-validation is.

As an example, for ridge regression, we can set the performance measure as:

$$\mathsf{perf}(\mathsf{W}, \mathcal{D}) := \frac{1}{|\mathcal{D}|} \sum_{i=1}^{|\mathcal{D}|} \|\mathbf{Y}_i - W\mathbf{X}_i - \mathbf{b}\|_2^2 = \frac{1}{|\mathcal{D}|} \|Y - \mathsf{W}\mathsf{X}\|_{\mathsf{F}}^2,$$

where recall that

$$\mathsf{X} = \begin{bmatrix} \mathbf{X}_1 & \cdots & \mathbf{X}_{|\mathcal{D}|} \\ 1 & \cdots & 1 \end{bmatrix}, \quad Y = [\mathbf{Y}_1, \ldots, \mathbf{Y}_{|\mathcal{D}|}], \quad \text{and } \mathsf{W} = [W, \mathbf{b}].$$

Exercise 2.34: Can we use training objective as performance measure?

Explain if we can use the regularized training objective $\frac{1}{|\mathcal{D}|}\|Y - \mathsf{W}\mathsf{X}\|_{\mathsf{F}}^2 + \lambda\|\mathsf{W}\|_{\mathsf{F}}^2$ as our performance measure perf?

3 Logistic Regression

Goal

Understand logistic regression. Comparison with linear regression.

Alert 3.1: Convention

Gray boxes are not required hence can be omitted for unenthusiastic readers.

This note is likely to be updated again soon.

Unlike the slides, in this note we encode the label $y \in \{\pm 1\}$ and we arrange $\mathbf{x}_i$ in X columnwise.

We use $\mathbf{x}$ and $\mathbf{w}$ for the original vectors and x and w for the padded versions (with constant 1 and bias b respectively). Similar, we use X and W for the original matrices and X and W for the padded versions.

Remark 3.2: Confidence of prediction

In perceptron we make predictions directly through a linear threshold function:

$$\hat{y} = \operatorname{sign}(\mathbf{w}^\top \mathbf{x} + b).$$

Often, we would also like to know how confident we are about this prediction $\hat{y}$. For example, we can use the magnitude $|\mathbf{w}^\top \mathbf{x} + b|$ as the indication of our "confidence." This choice, however, can be difficult to interpret at times, after all the magnitude could be any positive real number.

In the literature there are many attempts to turn the real output of a classifier into probability estimates, see for instance Vovk and Petej (2012) and Vovk et al. (2015).

Vovk, Vladimir and Ivan Petej (2012). "Venn-Abers Predictors". In: *UAI*.

Vovk, Vladimir, Ivan Petej, and Valentina Fedorova (2015). "Large-scale probabilistic predictors with and without guarantees of validity". In: *NIPS*.

Remark 3.3: Reduce classification to regression?

Recall that the optimal Bayes classifier is

$$h^\star(\mathbf{x}) = \operatorname{sign}(2\eta(\mathbf{x}) - 1), \quad \text{where} \quad \eta(\mathbf{x}) = \Pr(Y = 1 | \mathbf{X} = \mathbf{x}).$$

The posterior probability $\eta(\mathbf{x})$ is a perfect measure of our confidence in predicting $\hat{y} = h^\star(\mathbf{x})$. Therefore, one may attempt to estimate the posterior probability

$$\eta(\mathbf{x}) = \Pr(Y = 1 | \mathbf{X} = \mathbf{x}) = \mathsf{E}(\mathbf{1}_{Y=1} | \mathbf{X} = \mathbf{x}).$$

If we define $\mathfrak{Y} = \mathbf{1}_{Y=1} \in \{0, 1\}$, then $\eta(\mathbf{X})$ is exactly the regression function of $(\mathbf{X}, \mathfrak{Y})$, see Definition 2.8. So, in principle, we could try to estimate the regression function based on some i.i.d. samples $\{(\mathbf{X}_i, \mathfrak{Y}_i) : i = 1, \ldots, n\}$.

The issue with the above approach is that we are in fact reducing an easier problem (classification) to a more general hence harder problem (regression). Note that the posterior probability $\eta(\mathbf{x})$ always lies in $[0, 1]$, and we would like to exploit this *a priori* knowledge. However, a generic approach to estimate the regression function would not be able to take this structure into account. In fact, an estimate of the regression function (e.g. through linear regression) may not always take values in $[0, 1]$ at all.

As a practical rule of thumb: Never try to solve a more general problem than necessary. (Theoreticians violate this rule all the time but nothing is meant to be practical in theory anyways.)

Definition 3.4: Bernoulli model

Let us consider the binary classification problem with labels $y \in \{\pm 1\}$. With the parameterization:

$$\Pr(Y = 1|\mathbf{X} = \mathbf{x}) =: p(\mathbf{x}; \mathbf{w}), \tag{3.1}$$

where p is a function that maps $\mathbf{x}$ and $\mathbf{w}$ into $[0, 1]$, we then have the Bernoulli model for generating the label $y \in \{\pm 1\}$:

$$\Pr(Y = y|\mathbf{X} = \mathbf{x}) = p(\mathbf{x}; \mathbf{w})^{(1+y)/2}[1 - p(\mathbf{x}; \mathbf{w})]^{(1-y)/2}.$$

Let $\mathcal{D} = \{(\mathbf{x}_i, y_i) : i = 1, \ldots, n\}$ be an i.i.d. sample from the same distribution as $(\mathbf{X}, Y)$. The conditional likelihood factorizes under the i.i.d. assumption:

$$\begin{aligned}\Pr(Y_1 = y_1, \ldots, Y_n = y_n|\mathbf{X}_1 = \mathbf{x}_1, \ldots, \mathbf{X}_n = \mathbf{x}_n) &= \prod_{i=1}^{n} \Pr(Y_i = y_i|\mathbf{X}_i = \mathbf{x}_i) \\ &= \prod_{i=1}^{n} p(\mathbf{x}_i; \mathbf{w})^{(1+y_i)/2}[1 - p(\mathbf{x}_i; \mathbf{w})]^{(1-y_i)/2}. \end{aligned} \tag{3.2}$$

A standard algorithm in statistics and machine learning for parameter estimation is to maximize the (conditional) likelihood. In this case, we can maximize (3.2) w.r.t. $\mathbf{w}$. Once we figure out $\mathbf{w}$, we can then make probability estimates on any new test sample $\mathbf{x}$, by simply plugging $\mathbf{w}$ and $\mathbf{x}$ into (3.1).

Example 3.5: What is that function $p(\mathbf{x}; \mathbf{w})$?

Let us consider two extreme cases:

- $p(\mathbf{x}; \mathbf{w}) = p(\mathbf{x})$, i.e., the function p can take any value on any $\mathbf{x}$. This is the extreme case where anything we learn from one data point $\mathbf{x}_i$ may have nothing to do with what $p(\mathbf{x}_j)$ can take. Denote $p_i = p(\mathbf{x}_i)$, take logarithm on (3.2), and negate:

$$\min_{p_1,\ldots,p_n} \ -\frac{1}{2}\sum_{i=1}^{n}(1 + y_i)\log p_i + (1 - y_i)\log(1 - p_i).$$

  Since the p_i's are not related, we can solve them separately. Recall the definition of the KL divergence in Definition 15.3, we know

$$-\tfrac{1+y_i}{2}\log p_i - \tfrac{1-y_i}{2}\log(1 - p_i) = \mathsf{KL}\left(\begin{pmatrix}\frac{1+y_i}{2} \\ \frac{1-y_i}{2}\end{pmatrix} \middle\| \begin{pmatrix} p_i \\ 1 - p_i\end{pmatrix}\right) - \tfrac{1+y_i}{2}\log\tfrac{1+y_i}{2} - \tfrac{1-y_i}{2}\log\tfrac{1-y_i}{2}.$$

  Since the KL divergence is nonnegative, to maximize the conditional likelihood we should set

$$p_i = \tfrac{1+y_i}{2}.$$

  This result does make sense, since for $y_i = 1$ we set $p_i = 1$ while for $y_i = -1$ we set $p_i = 0$ (so that $1 - p_i = 1$, recall that p_i is the probability for y_i being 1).

- $p(\mathbf{x}; \mathbf{w}) = p(\mathbf{w}) = p$, i.e., the function p is independent of $\mathbf{x}$ hence is a constant. The is the extreme case where anything we learn from one data point immediately applies in the same way to any other data point. Similar as above, we find p by solving

$$\min_{p} \ -\frac{1}{2}\sum_{i=1}^{n}(1 + y_i)\log p + (1 - y_i)\log(1 - p).$$

  Let $\bar{p} = \frac{1}{2n}\sum_{i=1}^{n}(1 + y_i)$, which is exactly the fraction of positive examples in our training set $\mathcal{D}$. Obviously then $1 - \bar{p} = \frac{1}{2n}\sum_{i=1}^{n}(1 - y_i)$. Prove by yourself that $\bar{p}$ is indeed the optimal choice. This again makes sense: if we have to pick one and only one probability estimate (for every data point), intuitively we should just use the fraction of positive examples in our training set.

The above two extremes are not satisfactory: it is either too flexible by allowing each data point to have its own probability estimate (which may have nothing to do with each other hence learning is impossible) or it is too inflexible by restricting every data point to use the same probability estimate. Logistic regression, which we define next, is an interpolation between the two extremes.

Definition 3.6: Logistic Regression (Cox 1958)

Motivated by the two extreme cases in Example 3.5, we want to parameterize $p(\mathbf{x}; \mathbf{w})$ in a not-too-flexible and not-too-inflexible way. One natural choice is to set p as an affine function (how surprising): $p(\mathbf{x}; \mathbf{w}) = \mathbf{w}^\top \mathbf{x} + b$. However, this choice has the disadvantage in the sense that the left-hand side takes value in $[0, 1]$ while the right-hand side takes value in $\mathbb{R}$. To avoid this issue, we first take a logit transformation of p and then equate it to an affine function:

$$\log \frac{p(\mathbf{x}; \mathbf{w})}{1 - p(\mathbf{x}; \mathbf{w})} = \mathbf{w}^\top \mathbf{x} + b.$$

The ratio on the left-hand side is known as odds ratio (probability of 1 divide by probability of -1). Or equivalently,

$$p(\mathbf{x}; \mathbf{w}) = \frac{1}{1 + \exp(-\mathbf{w}^\top \mathbf{x} - b)} = \mathtt{sgm}(\mathbf{w}^\top \mathbf{x} + b), \quad \text{where} \quad \mathtt{sgm}(t) = \frac{1}{1 + \exp(-t)} = \frac{\exp(t)}{1 + \exp(t)} \tag{3.3}$$

is the so-called sigmoid function. Note that our definition of p involves $\mathbf{x}$ but not the label y. This is crucial as later on we will use $p(\mathbf{x}; \mathbf{w})$ to predict the label y.

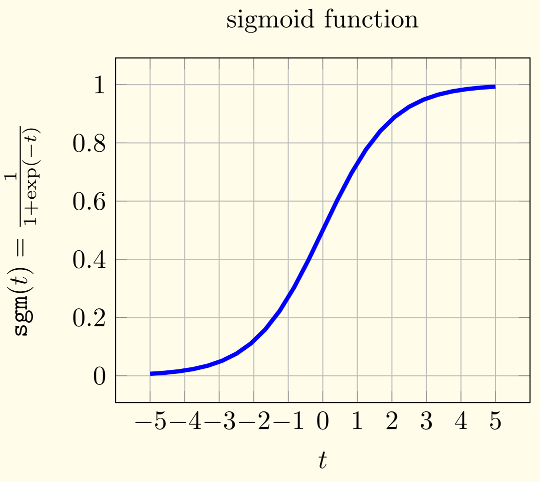
Plugging (3.3) into the conditional likelihood (3.2) and maximizing $\mathbf{w} = (\mathbf{w}, b)$ we get the formulation of logistic regression, or in equivalent form:

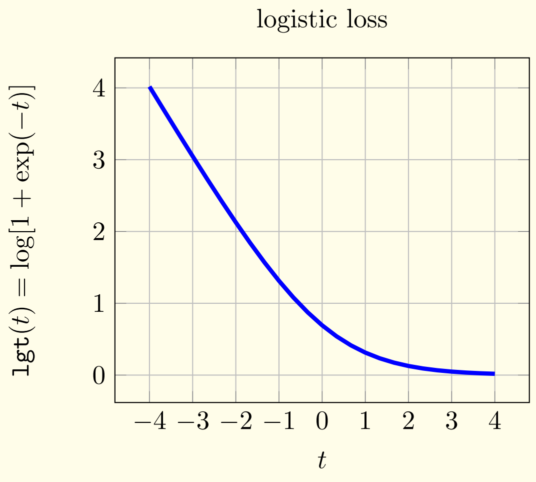
$$\min_{\mathbf{w}} \ \frac{1}{2} \sum_{i=1}^{n} (1 + y_i) \log(1 + \exp(-\mathbf{x}_i^\top \mathbf{w})) + (1 - y_i) \log(1 + \exp(-\mathbf{x}_i^\top \mathbf{w})) + (1 - y_i)\mathbf{x}_i^\top \mathbf{w},$$

which is usually written in the more succinct form:

$$\min_{\mathbf{w}} \ \sum_{i=1}^{n} \mathtt{lgt}(y_i \hat{\mathsf{y}}_i), \quad \text{where} \quad \hat{\mathsf{y}}_i = \mathbf{w}^\top \mathbf{x}_i, \quad \text{and } \mathtt{lgt}(t) := \log(1 + \exp(-t)) \tag{3.4}$$

is the so-called logistic loss in the literature. (Clearly, the base of log is immaterial and we use the natural log.) In the above we have applied padding, see Remark 1.17, to ease notation.





Cox, D. R. (1958). "The Regression Analysis of Binary Sequences". *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 20, no. 2, pp. 215–242.

Alert 3.7: Binary labels: $\{\pm 1\}$ or $\{0, 1\}$?

In this note we choose to encode binary labels as $\{\pm 1\}$. In the literature the alternative choice $\{0, 1\}$ is also common. In essence there is really no difference if we choose one convention or the other: the eventual conclusions would be the same. However, some formulas do look a bit different on the surface! For example, the neat formula (3.4) becomes

$$\min_{\mathbf{w}} \sum_{i=1}^{n} \log[\exp\left((1 - y_i)\mathbf{w}^\top \mathbf{x}_i\right) + \exp(-y_i \mathbf{w}^\top \mathbf{x}_i)],$$

had we chosen the convention $y_i \in \{0, 1\}$. Always check the convention before you subscribe to any formula!

Remark 3.8: Prediction with confidence

Once we solve $\mathbf{w}$ as in (3.4) and given a new test sample $\mathbf{x}$, we can compute $p(\mathbf{x}; \mathbf{w}) = \frac{1}{1+\exp(-\mathbf{w}^\top \mathbf{x})}$ and predict

$$\hat{y}(\mathbf{x}) = \begin{cases} 1, & \text{if } p(\mathbf{x}; \mathbf{w}) \geq 1/2 \iff \mathbf{w}^\top \mathbf{x} \geq 0 \\ -1, & \text{otherwise} \end{cases}.$$

In other words, for predicting the label we are back to the familiar rule $\hat{y} = \operatorname{sign}(\mathbf{w}^\top \mathbf{x})$. However, now we are also equipped with the probability confidence $p(\mathbf{x}; \mathbf{w})$.

It is clear that logistic regression is a linear classification algorithm, whose decision boundary is given by the hyperplane

$$H = \{\mathbf{x} : \mathbf{w}^\top \mathbf{x} = 0\}.$$

Alert 3.9: Something is better than nothing?

It is tempting to prefer logistic regression over other classification algorithms since the former spits out not only label predictions but also probability confidences. However, one should keep in mind that in logistic regression, we make the assumption (see Equation (3.3))

$$\Pr(Y = 1 | \mathbf{X} = \mathbf{x}) = \mathtt{sgm}(\mathbf{w}^\top \mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w}^\top \mathbf{x})},$$

which may or may not hold on your particular dataset. So the probability estimates we get from logistic regression can be totally off. Is it really better to have a probability estimate that is potentially very wrong than not to have anything at all? To exaggerate in another extreme, for any classification algorithm we can "make up" a 100% confidence on each of its predictions. Does this "completely fake" probability confidence bring any comfort? But, how is this any different from the numbers you get from logistic regression?

Alert 3.10: Do not do extra work

Logistic regression does more than classification, since it also tries to estimate the posterior probabilities. However, if prediction (of the label) is our sole goal, then we do not have to, and perhaps should not, estimate the posterior probabilities. Put it more precisely, all we need to know is whether or not $\eta(\mathbf{x}) = \Pr(Y = 1 | \mathbf{X} = \mathbf{x})$ is larger than $1/2$. The precise value of $\eta(\mathbf{x})$ is not important; only its comparison with $1/2$ is. As we shall see, support vector machines, in contrast, only tries to estimate the decision boundary (i.e. the relative comparison between $\eta(\mathbf{x})$ and $1/2$), hence can be more efficient.

Remark 3.11: More than logistic regression

The main idea behind logistic regression is to equate the posterior probability $p(\mathbf{x}; \mathbf{w})$ with some transformation F of the affine function $\mathbf{w}^\top \mathbf{x}$. Here the transformation F turns a real number into some value in $[0, 1]$ (where the posterior probability belongs to). Obviously, we can choose F to be any cumulative distribution function (cdf) on the real line. Indeed, plug the formula

$$p(\mathbf{x}; \mathbf{w}) = F(\mathbf{w}^\top \mathbf{x}),$$

into the conditional likelihood (3.2) gives us many variants of logistic regression. If we choose F to be the cdf of the logistic distribution (hence the name)

$$F(x; \mu, s) = \frac{1}{1 + \exp\left(-\frac{x-\mu}{s}\right)},$$

where μ is the mean and s is some shape parameter (with variance $s^2\pi^2/3$), then we recover logistic regression (provided that $\mu = 0$ and $s = 1$).

If we choose F to be the cdf of the standard normal distribution, then we get the so-called probit regression.

Algorithm 3.12: Gradient descent for logistic regression

Unlike linear regression, logistic regression no longer admits a closed-form solution. Instead, we can apply gradient descent to iteratively converge to a solution. All we need is to apply the chain rule to compute the gradient of each summand of the objective function in (3.4):

$$\begin{aligned}\nabla \mathtt{lgt}(y_i \mathbf{x}_i^\top \mathbf{w}) &= -\frac{\exp(-t)}{1 + \exp(-t)}\Big|_{t = y_i \mathbf{w}^\top \mathbf{x}_i} \cdot y_i \mathbf{x}_i = -\mathtt{sgm}(-y_i \mathbf{w}^\top \mathbf{x}_i) \cdot y_i \mathbf{x}_i \\ &= -y_i \mathbf{x}_i + \mathtt{sgm}(y_i \mathbf{w}^\top \mathbf{x}_i) y_i \mathbf{x}_i \\ &= \begin{cases} (p(\mathbf{x}_i; \mathbf{w}) - 1)\mathbf{x}_i, & \text{if } y_i = 1 \\ (p(\mathbf{x}_i; \mathbf{w}) - 0)\mathbf{x}_i, & \text{if } y_i = -1 \end{cases} \\ &= \left(p(\mathbf{x}_i; \mathbf{w}) - \tfrac{y_i + 1}{2}\right) \mathbf{x}_i.\end{aligned}$$

In the following algorithm, we need to choose a step size η. A safe choice is

$$\eta = \frac{4}{\|\mathsf{X}\|_{\mathrm{sp}}^2},$$

namely, the inverse of the largest singular value of the Hessian (see below). An alternative is to start with some small η and decrease it whenever we are not making progress (known as step size annealing).

Algorithm: Gradient descent for binary logistic regression.

```
Input: $X \in \mathbb{R}^{d \times n}$, $\mathbf{y} \in \{-1, 1\}^n$ (training set), initialization $\mathbf{w} \in \mathbb{R}^{\mathsf{p}}$
Output: $\mathbf{w} \in \mathbb{R}^{\mathsf{p}}$
1 for $t = 1, 2, \ldots, \mathtt{maxiter}$ do
2     sample a minibatch $I = \{i_1, \ldots, i_{\mathsf{m}}\} \subseteq \{1, \ldots, n\}$
3     $\mathbf{g} \leftarrow \mathbf{0}$
4     for $i \in I$ do                                  // use for-loop only in parallel implementation
5         $p_i \leftarrow \frac{1}{1+\exp(-\mathbf{w}^\top \mathbf{x}_i)}$          // in serial, replace with $\mathbf{p} \leftarrow \frac{1}{1+\exp(-\mathsf{X}_{:,I}^\top \mathbf{w})}$
6         $\mathbf{g} \leftarrow \mathbf{g} + (p_i - \frac{1+y_i}{2})\mathbf{x}_i$    // in serial, replace with $\mathbf{g} \leftarrow \mathsf{X}_{:,I}(\mathbf{p} - \frac{1+\mathbf{y}_I}{2})$
7     choose step size $\eta > 0$
8     $\mathbf{w} \leftarrow \mathbf{w} - \eta \mathbf{g}$
9     check stopping criterion                          // e.g. $\|\eta \mathbf{g}\| \leq \mathtt{tol}$
```

For small problems ($n \leq 10^4$ say), we can set $I = \{1, \ldots, n\}$, i.e., use the entire dataset in every iteration.

Algorithm 3.13: Newton iteration for logistic regression

We can also apply Newton's algorithm for solving logistic regression. In addition to computing the gradient, we now also need to compute the Hessian:

$$H_i = \nabla^2_{\mathbf{w}} \mathtt{lgt}(y_i \mathbf{w}^\top \mathbf{x}_i) = \mathbf{x}_i [\nabla_{\mathbf{w}} p(\mathbf{x}_i; \mathbf{w})]^\top = p(\mathbf{x}_i; \mathbf{w})[1 - p(\mathbf{x}_i; \mathbf{w})] \mathbf{x}_i \mathbf{x}_i^\top.$$

Algorithm: Newton iteration for binary logistic regression.

```
Input: $X \in \mathbb{R}^{d \times n}$, $\mathbf{y} \in \{-1, 1\}^n$ (training set), initialization $\mathbf{w} \in \mathbb{R}^p$
Output: $\mathbf{w} \in \mathbb{R}^p$
1  for $t = 1, 2, \ldots, \mathtt{maxiter}$ do
2      sample a minibatch $I = \{i_1, \ldots, i_m\} \subseteq \{1, \ldots, n\}$
3      $\mathbf{g} \leftarrow \mathbf{0}$, $H \leftarrow \mathbf{0}$
4      for $i \in I$ do                          // use for-loop only in parallel implementation
5          $p_i \leftarrow \frac{1}{1+\exp(-\mathbf{w}^\top \mathbf{x}_i)}$      // in serial, replace with $\mathbf{p} \leftarrow \frac{1}{1+\exp(-X_{:,I}^\top \mathbf{w})}$
6          $\mathbf{g} \leftarrow \mathbf{g} + (p_i - \frac{1+y_i}{2})\mathbf{x}_i$      // in serial, replace with $\mathbf{g} \leftarrow X_{:,I}(\mathbf{p} - \frac{1+\mathbf{y}_I}{2})$
7          $H \leftarrow H + p_i(1 - p_i)\mathbf{x}_i \mathbf{x}_i^\top$      // in serial, replace with $H \leftarrow X_{:,I} \operatorname{diag}(\mathbf{p} \odot (1 - \mathbf{p})) X_{:,I}^\top$
8      choose step size $\eta > 0$
9      $\mathbf{w} \leftarrow \mathbf{w} - \eta H^{-1} \mathbf{g}$                // solve $H^{-1}\mathbf{g}$ as linear system
10     check stopping criterion                   // e.g. $\|\eta \mathbf{g}\| \leq \mathtt{tol}$
```

Typically, we need to tune η at the initial phase but quickly we can just set $\eta \equiv 1$. Newton's algorithm is generally much faster than gradient descent. The downside, however, is that computing and storing the Hessian can be expensive. For example, Algorithm 3.12 has per-step time complexity $O(\mathsf{m}d)$ and space complexity $O(d)$ (or $O(nd)$ if X is stored explicitly in memory) while Algorithm 3.13 has per-step time complexity $O(\mathsf{m}d^2 + d^3)$ and space complexity $O(d^2)$ (or $O(nd + d^2)$ if X is stored explicitly in memory).

Alert 3.14: Overflow and underflow

Numerically computing $\exp(\mathbf{a})$ can be tricky when the vector $\mathbf{a}$ has very large or small entries. The usual trick is to shift the origin as follows. Let $t = \max_i a_i - \min_i a_i$ be the range of the elements in $\mathbf{a}$. Then, after shifting 0 to $t/2$:

$$\exp(\mathbf{a}) = \exp(\mathbf{a} - t/2) \exp(t/2).$$

Computing $\exp(\mathbf{a} - t/2)$ may be numerically better than computing $\exp(\mathbf{a})$ directly. The scaling factor $\exp(t/2)$ usually will cancel out in later computations so we do not need to compute it. (Even when we have to, it may be better to return $t/2$ than $\exp(t/2)$.)

Remark 3.15: Logistic regression as iterative re-weighted linear regression

Let us define the diagonal matrix $\hat{S} = \operatorname{diag}\big(\hat{\mathbf{p}} \odot (1 - \hat{\mathbf{p}})\big)$. If we set $\eta \equiv 1$, then we can interpret Newton's iteration in Algorithm 3.13 as iterative re-weighted linear regression (IRLS):

$$\begin{aligned}
\mathbf{w}_+ &= \mathbf{w} - (\mathsf{X}\hat{S}\mathsf{X}^\top)^{-1} \mathsf{X}(\hat{\mathbf{p}} - \tfrac{1+\mathbf{y}}{2}) \\
&= (\mathsf{X}\hat{S}\mathsf{X}^\top)^{-1}[(\mathsf{X}\hat{S}\mathsf{X}^\top)\mathbf{w} - \mathsf{X}(\hat{\mathbf{p}} - \tfrac{1+\mathbf{y}}{2})] \\
&= (\mathsf{X}\hat{S}\mathsf{X}^\top)^{-1} \mathsf{X}\hat{S}\mathfrak{y}, \quad \mathfrak{y} := \mathsf{X}^\top \mathbf{w} - \hat{S}^{-1}(\hat{\mathbf{p}} - \tfrac{1+\mathbf{y}}{2}) \\
&= \operatorname*{argmin}_{\mathbf{w}} \sum_{i=1}^n \hat{s}_i (\mathbf{w}^\top \mathbf{x}_i - \mathfrak{y}_i)^2, \quad \hat{s}_i := \hat{p}_i (1 - \hat{p}_i),
\end{aligned} \tag{3.5}$$

where the last equality can be seen by setting the derivative w.r.t. $\mathbf{w}$ to $\mathbf{0}$.

So, Newton's algorithm basically consists of two steps:

- given the current $\mathbf{w}$, compute the weights $\hat{s}_i$ and update the targets $\mathfrak{y}_i$. Importantly, if the current $\mathbf{w}$ yields very confident prediction $\hat{p}_i$ for the i-th training example (i.e., when $\hat{p}_i$ is close to 0 or 1), then the corresponding weight $\hat{s}_i$ is close to 0, i.e., we are down-weighting this training example whose label we are already fairly certain about. On the other hand, if $\hat{p}_i$ is close to $1/2$, meaning we are very unsure about the i-th training example, then the corresponding weight $\hat{s}_i$ will be close to the maximum value $1/4$, i.e. we pay more attention to it in the next iteration.
- solve the re-weighted least squares problem (3.5).

We have to iterate the above two steps because $\hat{\mathbf{p}}$ hence $\hat{\mathbf{s}}$ are both functions of $\mathbf{w}$ themselves. It would be too difficult to solve $\mathbf{w}$ in one step. This iterative way of solving complicated problems is very typical in machine learning.

Remark 3.16: Linear regression vs. logistic regression

In the following comparison, $\hat{S} = \mathrm{diag}\big(\hat{\mathbf{p}} \odot (1 - \hat{\mathbf{p}})\big)$. We note that as $\hat{y}_i$ deviates from y_i, the least squares loss varies from 0 to ∞. Similarly, as $\hat{p}_i$ deviates from y_i, the cross-entropy loss varies from 0 to ∞ as well.

- least-squares: $\sum_{i=1}^n (y_i - \hat{y}_i)^2$
- prediction: $\hat{y}_i = \mathbf{w}^\top \mathbf{x}_i$
- objective: $\|\mathbf{y} - \hat{\mathbf{y}}\|_2^2$
- grad: $\mathbf{w} \leftarrow \mathbf{w} - \eta X(\hat{\mathbf{y}} - \mathbf{y})$
- Newton: $\mathbf{w} \leftarrow \mathbf{w} - \eta(XX^\top)^{-1}X(\hat{\mathbf{y}} - \mathbf{y})$

- cross-entropy: $\sum_{i=1}^n -\frac{1+y_i}{2}\log \hat{p}_i - \frac{1-y_i}{2}\log(1-\hat{p}_i)$
- prediction: $\hat{y}_i = \mathrm{sign}(\mathbf{w}^\top \mathbf{x}_i)$, $\hat{p}_i = \mathtt{sgm}(\mathbf{w}^\top \mathbf{x}_i)$
- objective: $\mathsf{KL}(\frac{1+\mathbf{y}}{2} \| \hat{\mathbf{p}})$
- grad: $\mathbf{w} \leftarrow \mathbf{w} - \eta X(\hat{\mathbf{p}} - \frac{1+\mathbf{y}}{2})$
- Newton: $\mathbf{w} \leftarrow \mathbf{w} - \eta(X\hat{S}X^\top)^{-1}X(\hat{\mathbf{p}} - \frac{1+\mathbf{y}}{2})$

Exercise 3.17: Linearly separable

If the training data $\mathcal{D} = \wr(\mathbf{x}_i, y_i) : i = 1, \ldots, n\int$ is linearly separable (see Definition 1.24), does logistic regression have a solution $\mathbf{w}$? What happens if we run gradient descent (Algorithm 3.12) or Newton's iteration (Algorithm 3.13)?

Exercise 3.18: Regularization

Derive the formulation and an algorithm (gradient or Newton) for ℓ_2-regularized logistic regression, where we add $\lambda\|\mathbf{w}\|_2^2$.

Remark 3.19: More than 2 classes

We can easily extend logistic regression to $\mathsf{c} > 2$ classes. As before, we make the assumption

$$\Pr(Y = k | \mathbf{X} = \mathbf{x}) = f_k(W^\top \mathbf{x}), \quad k = 1, \ldots, \mathsf{c},$$

where $W = [\mathbf{w}_1, \ldots, \mathbf{w}_\mathsf{c}] \in \mathbb{R}^{\mathsf{p}\times\mathsf{c}}$ and the vector-valued function $\mathbf{f} = [f_1, \ldots, f_\mathsf{c}] : \mathbb{R}^\mathsf{c} \to \Delta_{\mathsf{c}-1}$ maps a vector of size $\mathsf{c} \times 1$ to a probability vector in the simplex $\Delta_{\mathsf{c}-1}$. Given an i.i.d. training dataset $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i) : i = 1, \ldots, n\}$, where each $\mathbf{y}_i \in \{0,1\}^\mathsf{c}$ is a one-hot vector, i.e. $\mathbf{1}^\top \mathbf{y}_i = 1$, then the (negated) conditional log-likelihood is:

$$-\log \Pr(\mathbf{Y}_1 = \mathbf{y}_1, \ldots, \mathbf{Y}_n = \mathbf{y}_n | \mathbf{X}_1 = \mathbf{x}_1, \ldots, \mathbf{X}_n = \mathbf{x}_n) = -\log \prod_{i=1}^n \prod_{k=1}^\mathsf{c} [f_k(W^\top \mathbf{x}_i)]^{y_{ki}}$$

$$= \sum_{i=1}^{n}\sum_{k=1}^{\mathsf{c}} -y_{ki} \log f_k(\mathsf{W}^\top \mathbf{x}_i).$$

To minimize the negated log-likelihood, we can apply gradient descent or Newton's iteration as before:

$$\nabla \ell_i(\mathsf{W}) = \sum_{k=1}^{\mathsf{c}} -y_{ki} \frac{1}{f_k(\mathsf{W}^\top \mathbf{x}_i)} \mathbf{x}_i [\nabla f_k \big|_{\mathsf{W}^\top \mathbf{x}_i}]^\top, \tag{3.6}$$

$$\forall \mathsf{G} \in \mathbb{R}^{\mathsf{p}\times\mathsf{c}}, \quad [\nabla^2 \ell_i(\mathsf{W})](\mathsf{G}) = \sum_{k=1}^{\mathsf{c}} -y_{ki} \frac{1}{f_k^2(\mathsf{W}^\top \mathbf{x}_i)} (\mathbf{x}_i \mathbf{x}_i^\top) \mathsf{G} [\nabla f_k \nabla f_k^\top - f_k \nabla^2 f_k] \big|_{\mathsf{W}^\top \mathbf{x}_i}, \tag{3.7}$$

where recall that $\nabla \ell_i(\mathsf{W}) \in \mathbb{R}^{\mathsf{p}\times\mathsf{c}}$ and $\nabla^2 \ell_i(\mathsf{W}) : \mathbb{R}^{\mathsf{p}\times\mathsf{c}} \to \mathbb{R}^{\mathsf{p}\times\mathsf{c}}$. Note that due to our one-hot encoding, the above summation has actually one term.

Definition 3.20: Multiclass logistic regression, a.k.a. Multinomial logit or softmax regression

The multinomial logit model corresponds to choosing the softmax function:

$$\mathbf{f}(\mathsf{W}^\top \mathbf{x}) = \mathtt{softmax}(\mathsf{W}^\top \mathbf{x}), \quad \text{where} \quad \mathtt{softmax} : \mathbb{R}^{\mathsf{c}} \to \Delta_{\mathsf{c}-1}, \ \mathfrak{y} \mapsto \frac{\exp(\mathfrak{y})}{\mathbf{1}^\top \exp(\mathfrak{y})}.$$

Let $\hat{\mathbf{p}}_i = \mathbf{f}(\mathsf{W}^\top \mathbf{x}_i)$ and specialize (3.6) and (3.7) to the softmax function we obtain its gradient and Hessian:

$$\nabla \ell_i(\mathsf{W}) = \mathbf{x}_i (\hat{\mathbf{p}}_i - \mathbf{y}_i)^\top,$$
$$\forall \mathsf{G} \in \mathbb{R}^{\mathsf{p}\times\mathsf{c}}, \quad [\nabla^2 \ell_i(\mathsf{W})](\mathsf{G}) = (\mathbf{x}_i \mathbf{x}_i^\top) \mathsf{G} \big(\operatorname{diag}(\hat{\mathbf{p}}_i) - \hat{\mathbf{p}}_i \hat{\mathbf{p}}_i^\top \big).$$

In the multiclass setting, solving the Newton step could quickly become infeasible ($O(\mathsf{d}^3\mathsf{c}^3)$). As Böhning (1992) pointed out, we can instead use the upper bound:

$$\mathbf{0} \preceq \operatorname{diag}(\hat{\mathbf{p}}_i) - \hat{\mathbf{p}}_i \hat{\mathbf{p}}_i^\top \preceq \frac{1}{2} (\mathbb{I}_k - \tfrac{1}{k+1} \mathbf{1}\mathbf{1}^\top),$$

which would reduce the computation to inverting only the data matrix $\mathsf{X}\mathsf{X}^\top = \sum_i \mathbf{x}_i \mathbf{x}_i^\top$.

Böhning, Dankmar (1992). "Multinomial logistic regression algorithm". *Annals of the Institute of Statistical Mathematics*, vol. 44, no. 1, pp. 197–200.

Remark 3.21: Mean and Covariance

We point out the following "miracle:" Let $\mathbf{Y}$ be a random vector taking values on standard bases $\{\mathbf{e}_k \in \{0,1\}^{\mathsf{c}} : k = 1, \ldots, \mathsf{c}, \mathbf{1}^\top \mathbf{e}_k = 1\}$ and following the multinomial distribution:

$$\Pr(\mathbf{Y} = \mathbf{e}_k) = p_k, \quad k = 1, \ldots, \mathsf{c}.$$

Then, straightforward calculation verifies:

$$\mathsf{E}(\mathbf{Y}) = \mathbf{p},$$
$$\operatorname{Cov}(\mathbf{Y}) = \operatorname{diag}(\mathbf{p}) - \mathbf{p}\mathbf{p}^\top.$$

Remark 3.22: Removing translation invariance in `softmax`

In the above multiclass logistic regression formulation, we used a matrix W with c columns to represent c

classes. Note however that the `softmax` function is translation invariant:

$$\forall \mathbf{w}, \quad \mathtt{softmax}((\mathsf{W} + \mathbf{w}\mathbf{1}^\top)^\top \mathbf{x}) = \mathtt{softmax}(\mathsf{W}^\top \mathbf{x}).$$

Therefore, for identifiability purposes, we may assume w.l.o.g. $\mathbf{w}_\mathsf{c} = \mathbf{0}$ and we need only optimize the first $\mathsf{c} - 1$ columns. If we denote $L(\mathbf{w}_1, \ldots, \mathbf{w}_{\mathsf{c}-1}, \mathbf{w}_\mathsf{c})$ as the original negated log-likelihood in Definition 3.20, then fixing $\mathbf{w}_\mathsf{c} = \mathbf{0}$ changes our objective to $L(\mathbf{w}_1, \ldots, \mathbf{w}_{c-1}, \mathbf{0})$. Clearly, the gradient and Hessian formula in Definition 3.20 still works after deleting the entries corresponding to $\mathbf{w}_\mathsf{c}$.

Setting $\mathsf{c} = 2$ we recover binary logistic regression, with the alternative encoding $y \in \{0, 1\}$ though.

Exercise 3.23: Alternative constraint to remove translation invariance

An alternative fix to the translation-invariance issue of `softmax` is to add the following constraint:

$$\mathsf{W}\mathbf{1} = \mathbf{0}. \tag{3.8}$$

In this case our objective changes to $L(\mathbf{w}_1, \ldots, \mathbf{w}_{c-1}, -\sum_{k=1}^{\mathsf{c}-1} \mathbf{w}_k)$. How should we modify the gradient and Hessian?

Interestingly, after we add ℓ_2 regularization to the unconstrained multiclass logistic regression:

$$\min_{\mathsf{W}} \; L(\mathbf{w}_1, \ldots, \mathbf{w}_\mathsf{c}) + \lambda \|\mathsf{W}\|_\mathsf{F}^2,$$

the solution automatically satisfies the constraint (3.8). Why? What if we added ℓ_1 regularization?

Definition 3.24: Generalized linear models (GLMs)

The similarity between linear regression and logistic regression is not coincidental: they both belong to generalized linear models (i.e. exponential family noise distributions), see Nelder and Wedderburn (1972).

Nelder, J. A. and R. W. M. Wedderburn (1972). "Generalized Linear Models". *Journal of the Royal Statistical Society. Series A (General)*, vol. 135, no. 3, pp. 370–384.

4 Support Vector Machines (SVM)

Goal

Define and understand the classical hard-margin SVM for binary classification. Dual view.

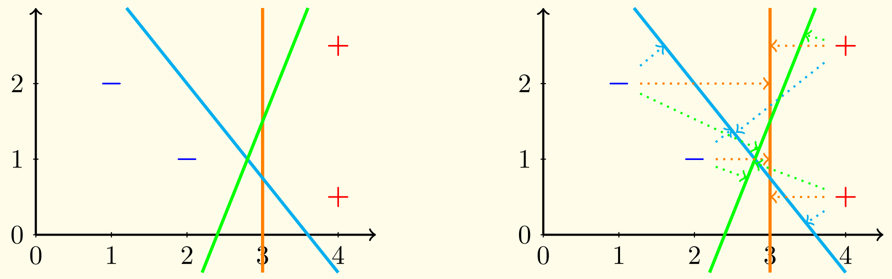
Alert 4.1: Convention

Gray boxes are not required hence can be omitted for unenthusiastic readers.

For less mathematical readers, think of the norm $\|\cdot\|$ and its dual norm $\|\cdot\|_\circ$ as the Euclidean ℓ_2 norm $\|\cdot\|_2$. Treat all distances as the Euclidean distance. All of our pictures are for this special case.

This note is likely to be updated again soon.

Definition 4.2: SVM as maximizing minimum distance



Given a (strictly) linearly separable dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i) \subseteq \mathbb{R}^d \times \{\pm 1\} : i = 1, \ldots, n\}$, there exists a separating hyperplane $H_\mathbf{w} = \{\mathbf{x} \in \mathbb{R}^d : \mathbf{w}^\top \mathbf{x} + b = 0\}$, namely that

$$\forall i, \ y_i(\mathbf{w}^\top \mathbf{x}_i + b) > 0.$$

In fact, there exist infinitely many separating hyperplanes: if we perturb $(\mathbf{w}, b)$ *slightly*, the resulting hyperplane would still be separating, thanks to continuity. Is there a particular separating hyperplane that stands out, and be “optimal”?

The answer is yes! Let $H_\mathbf{w}$ be any separating hyperplane (w.r.t. the given dataset $\mathcal{D}$). We can compute the distance from each training sample $\mathbf{x}_i$ to the hyperplane $H_\mathbf{w}$:

$$\begin{aligned}
\operatorname{dist}(\mathbf{x}_i, H_\mathbf{w}) &:= \min_{\mathbf{x} \in H_\mathbf{w}} \|\mathbf{x} - \mathbf{x}_i\|_\circ && \text{(e.g., the typical choice } \|\cdot\|_\circ = \|\cdot\| = \|\cdot\|_2) \\
&\geq \left| \frac{\mathbf{w}^\top(\mathbf{x} - \mathbf{x}_i) + b - b}{\|\mathbf{w}\|} \right| && \text{(Cauchy-Schwarz, see Definition 1.25)} \\
&= \frac{|\mathbf{w}^\top \mathbf{x}_i + b|}{\|\mathbf{w}\|} && \text{(equality at } \mathbf{x} = \mathbf{x}_i - \tfrac{\mathbf{z}}{\|\mathbf{w}\|^2}(b + \mathbf{w}^\top \mathbf{x}_i), \ \underbrace{\mathbf{z}^\top \mathbf{w} = \|\mathbf{w}\|^2, \|\mathbf{z}\|_\circ = \|\mathbf{w}\|}_{\mathbf{z} \in \partial\left[\frac{1}{2}\|\mathbf{w}\|^2\right]}) \\
&= \frac{y_i(\mathbf{w}^\top \mathbf{x}_i + b)}{\|\mathbf{w}\|} && (y_i \in \{\pm 1\} \text{ and } H_\mathbf{w} \text{ is separating}). \qquad (4.1)
\end{aligned}$$

Here and in the following, we always assume w.l.o.g. that the dataset $\mathcal{D}$ contains at least 1 positive example and 1 negative example, so that $\mathbf{w} = \mathbf{0}$ with any b cannot be a separating hyperplane.

Among all separating hyperplanes, support vector machines (SVM) tries to find one that maximizes the minimum distance (with the typical choice $\|\cdot\| = \|\cdot\|_2$ in mind):

$$\max_{\mathbf{w}: \forall i, y_i \hat{y}_i > 0} \ \min_{i=1,\ldots,n} \frac{y_i \hat{y}_i}{\|\mathbf{w}\|}, \quad \text{where} \quad \hat{y}_i = \mathbf{w}^\top \mathbf{x}_i + b. \qquad (4.2)$$

We remark that the above formulation is scaling-invariant: If $\mathsf{w} = (\mathbf{w}, b)$ is optimal, then so is $\gamma\mathsf{w}$ for any $\gamma > 0$ (the fraction is unchanged and the constraint on w is not affected). This is not at all surprising, as w and $\gamma\mathsf{w}$ really represent the same hyperplane: $H_{\mathsf{w}} = H_{\gamma\mathsf{w}}$. Note also that the separating condition $\forall i, y_i\hat{\mathrm{y}}_i > 0$ can be omitted since it is automatically satisfied if the dataset $\mathcal{D}$ is indeed (strictly) linearly separable.

Alert 4.3: Margin as minimum distance

We repeat the formula in Definition 4.2:

$$\operatorname{dist}(\mathbf{x}, H_{\mathsf{w}}) := \Big[\min_{\mathbf{z}\in H_{\mathsf{w}}} \|\mathbf{z}-\mathbf{x}\|_\circ\Big] = \frac{|\mathbf{w}^\top\mathbf{x}+b|}{\|\mathbf{w}\|} = \frac{y(\mathbf{w}^\top\mathbf{x}+b)}{\|\mathbf{w}\|} = \frac{y\hat{\mathrm{y}}}{\|\mathbf{w}\|},$$

where the third equality holds if $y\hat{\mathrm{y}} \geq 0$ and $y \in \{\pm 1\}$. Given any hyperplane H_{w}, we define its margin w.r.t. a data point $(\mathbf{x}, y)$ as:

$$\gamma((\mathbf{x}, y); H_{\mathsf{w}}) := \frac{y\hat{\mathrm{y}}}{\|\mathbf{w}\|}, \quad \hat{\mathrm{y}} = \mathbf{w}^\top\mathbf{x}+b,$$

Geometrically, when the hyperplane H_{w} classifies the data point $(\mathbf{x}, y)$ correctly (i.e. $y\hat{\mathrm{y}} > 0$), this margin is exactly the distance from $\mathbf{x}$ to the hyperplane H_{w}, and the negation of the distance otherwise.

Fixing any hyperplane H_{w}, we can extend the notion of its margin to a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i) : i = 1, \ldots, n\}$ by taking the (worst-case) minimum:

$$\gamma(\mathcal{D}; H_{\mathsf{w}}) := \Big[\min_{i=1,\ldots,n} \gamma((\mathbf{x}_i, y_i); H_{\mathsf{w}})\Big] = \min_i \frac{y_i\hat{\mathrm{y}}_i}{\|\mathbf{w}\|}, \quad \hat{\mathrm{y}}_i := \mathbf{w}^\top\mathbf{x}_i + b.$$

Again, when the hyperplane H_{w} (strictly) separates the dataset $\mathcal{D}$, the margin $\gamma(\mathcal{D}; H_{\mathsf{w}}) > 0$ coincides with the minimum distance, as we saw in Definition 4.2. However, when $\mathcal{D}$ is not (strictly) separated by H_{w}, the margin $\gamma(\mathcal{D}; H_{\mathsf{w}}) \leq 0$ is the negation of the maximum distance among all wrongly classified data points.

We can finally define the margin of a dataset $\mathcal{D}$ as the (best-case) maximum among all hyperplanes:

$$\gamma(\mathcal{D}) := \Big[\max_{\mathsf{w}} \gamma(\mathcal{D}; H_{\mathsf{w}})\Big] = \max_{\mathsf{w}} \min_{i=1,\ldots,n} \frac{y_i\hat{\mathrm{y}}_i}{\|\mathbf{w}\|}. \tag{4.3}$$

Again, when the dataset $\mathcal{D}$ is (strictly) linearly separable, the margin $\gamma(\mathcal{D}) > 0$ reduces to the minimum distance to the SVM hyperplane, in which case the margin definition here coincides with what we saw in Remark 1.30 (with the choice $\|\cdot\|_\circ = \|\cdot\| = \|\cdot\|_2$) and characterizes "how linearly separable" our dataset $\mathcal{D}$ is. On the other hand, when $\mathcal{D}$ is not (strictly) linearly separable, the margin $\gamma(\mathcal{D}) \leq 0$.

To summarize, hard-margin SVM, as defined in Definition 4.2, maximizes the margin among all hyperplanes on a (strictly) linearly separable dataset. Interestingly, with this interpretation, the hard-margin SVM formulation (4.3) continues to make sense even on a linearly inseparable dataset.

In the literature, sometimes people often call the unnormalized quantity $y\hat{\mathrm{y}}$ margin, which is fine as long as the scale $\|\mathbf{w}\|$ is kept constant.

Definition 4.4: Alternative definition of margin

We give a slightly different definition of margin here: γ^+. As the notation suggests, γ^+ coincides with the definition in Alert 4.3 on a (strictly) linearly separable dataset, and reduces to 0 otherwise.

- Given any hyperplane H_{w}, we define its margin w.r.t. a data point $(\mathbf{x}, y)$ as:

$$\gamma^+((\mathbf{x}, y); H_{\mathsf{w}}) := \frac{(y\hat{\mathrm{y}})^+}{\|\mathbf{w}\|}, \quad \hat{\mathrm{y}} = \mathbf{w}^\top\mathbf{x}+b,$$

where recall $(t)^+ = \max\{t, 0\}$ is the positive part. Geometrically, when the hyperplane H_{w} classifies the data point $(\mathbf{x}, y)$ correctly (i.e. $y\hat{y} \geq 0$), this margin is exactly the distance from $\mathbf{x}$ to the hyperplane $H_{\mathbf{w}}$, and 0 otherwise.

- Fixing any hyperplane $H_{\mathbf{w}}$, we can extend the notion of its margin to a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i) : i = 1, \ldots, n\}$ by taking the (worst-case) minimum:

$$\gamma^+(\mathcal{D}; H_{\mathbf{w}}) := \left[\min_{i=1,\ldots,n} \gamma^+((\mathbf{x}_i, y_i); H_{\mathbf{w}})\right] = \min_i \frac{(y_i\hat{y}_i)^+}{\|\mathbf{w}\|}, \quad \hat{y}_i := \mathbf{w}^\top \mathbf{x}_i + b.$$

Again, when the hyperplane $H_{\mathbf{w}}$ (strictly) separates the dataset $\mathcal{D}$, the margin $\gamma^+(\mathcal{D}; H_{\mathbf{w}}) > 0$ coincides with the minimum distance, as we saw in Definition 4.2. However, when $\mathcal{D}$ is not (strictly) separated by $H_{\mathbf{w}}$, the margin $\gamma^+(\mathcal{D}; H_{\mathbf{w}}) = 0$.

- We can finally define the margin of a dataset $\mathcal{D}$ as the (best-case) maximum among all hyperplanes:

$$\gamma^+(\mathcal{D}) := \left[\max_{\mathbf{w}} \gamma^+(\mathcal{D}; H_{\mathbf{w}})\right] = \max_{\mathbf{w}} \min_{i=1,\ldots,n} \frac{[y_i\hat{y}_i]^+}{\|\mathbf{w}\|}.$$

Again, when the dataset $\mathcal{D}$ is (strictly) linearly separable, the margin $\gamma^+(\mathcal{D})$ reduces to the minimum distance to the SVM hyperplane. In contrast, when $\mathcal{D}$ is not (strictly) linearly separable, the margin $\gamma^+(\mathcal{D}) = 0$.

Remark 4.5: Important standardization trick

A simple *standardization* trick in optimization is to introduce an extra variable so that we can reduce an arbitrary objective function to the canonical linear function. For instance, if we are interested in solving

$$\min_{\mathbf{w}} \quad f(\mathbf{w}),$$

where f can be any complicated nonlinear function. Upon introducing an extra variable t, we can reformulate our minimization problem equivalently as:

$$\min_{(\mathbf{w},t): f(\mathbf{w}) \leq t} \quad t,$$

where the new objective $(\mathbf{0}; 1)^\top(\mathbf{w}; t)$ is a simple linear function of $(\mathbf{w}; t)$. The expense, of course, is that we have to deal with the extra constraint $f(\mathbf{w}) \leq t$ now.

Remark 4.6: Removing homogeneity by normalizing direction

To remove the scaling-invariance mentioned in Definition 4.2, we can restrict the direction vector $\mathbf{w}$ to have unit norm, which happened to yield the same formulation as that in Rosen (1965) (see Remark 4.23 below for more details):

$$\max_{\mathbf{w}:\|\mathbf{w}\|=1} \min_{i=1,\ldots,n} y_i\hat{y}_i. \tag{4.4}$$

Applying the trick in Remark 4.5 (and noting we are maximizing here) yields the reformulation:

$$\max_{(\mathbf{w},\delta):\|\mathbf{w}\|=1} \quad \delta, \quad \text{s.t.} \quad \min_{i=1,\ldots,n} y_i\hat{y}_i \geq \delta \iff y_i\hat{y}_i \geq \delta, \ \forall i = 1, \ldots, n,$$

which is completely equivalent to (4.3) (except by excluding out the trivial solution $\mathbf{w} = 0$).

Observe that on any linearly separable dataset, at optimality we can always achieve $\delta \geq 0$. Thus, we may relax the unit norm constraint on $\mathbf{w}$ slightly:

$$\max_{\mathbf{w},\delta} \quad \delta \tag{4.5}$$

$$\text{s.t.} \quad \|\mathbf{w}\| \leq 1$$
$$y_i \hat{\mathtt{y}}_i \geq \delta, \ \forall i = 1, \ldots, n.$$

It is clear if the dataset $\mathcal{D}$ is indeed linearly separable, at maximum we may choose $\|\mathbf{w}\| = 1$, hence the "relaxation" is in fact equivalent (on any linearly separable dataset that consists of at least 1 positive and 1 negative).

Note that (4.5) is exactly the bound we got for the perceptron algorithm, see Remark 1.30. Thus, SVM could have been derived by optimizing the convergence bound of perceptron. So, theory does inspire new algorithms, although this was not what really happened in history: Vapnik and Chervonenkis (1964) did not seem to be aware of Theorem 1.28 at the time; in fact they wrongly claimed that the perceptron algorithm may not find a solution even when one exists....

Rosen, J.B (1965). "Pattern separation by convex programming". *Journal of Mathematical Analysis and Applications*, vol. 10, no. 1, pp. 123–134.

Vapnik, Vladimir N. and A. Ya. Chervonenkis (1964). "On a class of perceptrons". *Automation and Remote Control*, vol. 25, no. 1, pp. 112–120.

Exercise 4.7: Detecting linear separability

Prove an additional advantage of the "relaxation" (4.5): Its maximum value is always greater than 0, which is attained iff the dataset is not (strictly) linearly separable.

In contrast, prove that the original formulation (4.4) with *exact unit norm constraint*

- is equivalent to (4.5) with strictly positive maximum value, iff the dataset is (strictly) linearly separable;
- is different from (4.5) with strictly negative maximum value, iff the dataset is not (strictly) linearly separable and the intersection of positive and negative convex hulls has nonempty (relative) interior;
- is similar to (4.5) with exactly 0 maximum value, iff the dataset is not (strictly) linearly separable and the intersection of positive and negative convex hulls has empty (relative) interior.

Remark 4.8: History of SVM

In this box we summarize the first SVM paper due to Vapnik and Lerner (1963). Our terminology and notation are different from the somewhat obscure original.

Let our universe be $\mathcal{X}$ and $\mathcal{D} \subseteq \mathcal{X}$ a training set. Vapnik and Lerner (1963) considered essentially the unsupervised setting, where labels are not provided even at training. Let $\varphi : \mathcal{X} \to \mathcal{S} \subseteq \mathcal{H}$ be a mapping that turns the original input $\mathbf{x} \in \mathcal{X}$ into a point $\mathbf{z} := \varphi(\mathbf{x})$ in the unit sphere $\mathcal{S} := \{\mathbf{z} \in \mathcal{H} : \|\mathbf{z}\|_2 = 1\}$ of a Hilbert space $\mathcal{H}$ (with induced norm $\|\cdot\|_2$). Our goal is to divide the data into c (disjoint) categories $\mathcal{C}_1, \ldots, \mathcal{C}_{\mathsf{c}}$, each of which is represented by a center $\mathbf{c}_k \in \mathcal{S}$ so that

$$\max_{i \in \mathcal{C}_k} \|\mathbf{z}_i - \mathbf{c}_k\|_2^2 < \min_{j \notin \mathcal{C}_k} \|\mathbf{z}_j - \mathbf{c}_k\|_2^2.$$

In other words, if we circumscribe the training examples in each category $\mathcal{C}_k$ by the smallest ball with center in the unit sphere $\mathcal{S}$, then these balls only contain training examples in the same category (in fact this could be how we define categories). (However, these balls may still intersect.) Since both $\mathbf{z}$ and $\mathbf{c}$ have unit norm, equivalently we may require

$$\min_{i \in \mathcal{C}_k} \langle \mathbf{z}_i, \mathbf{c}_k \rangle > \max_{j \notin \mathcal{C}_k} \langle \mathbf{z}_j, \mathbf{c}_k \rangle . \tag{4.6}$$

In other words, there exists a hyperplane with direction $\mathbf{c}_k$ that (strictly) separates the category $\mathcal{C}_k$ from the rest categories $\mathcal{C}_{\neg k} := \bigcup_{l \neq k} \mathcal{C}_l$. Let us define

$$r_k = \max_{i \in \mathcal{C}_k} \|\mathbf{z}_i - \mathbf{c}_k\|_2, \quad k = 1, \ldots, \mathsf{c},$$

so that we may declare any point $\mathbf{z} \in \mathsf{B}(\mathbf{c}_k, r_k) := \{\mathbf{z} \in \mathcal{H} : \|\mathbf{z} - \mathbf{c}_k\|_2 \le r_k\}$, or equivalently any $\mathbf{z} \in H_k^+ := \{\mathbf{z} : \langle \mathbf{w}_k, \mathbf{z}\rangle \ge 1\}$ where $\mathbf{w}_k := \frac{\mathbf{c}_k}{1-\frac{1}{2}r_k^2}$, is in category k.

Vapnik and Lerner (1963) considered two scenarios:

- transductive learning (distinction): each (test) example is known to belong to one and only one ball $\mathsf{B}_k := \mathsf{B}(\mathbf{c}_k, r_k)$. In this case we may identify the category for any $\mathbf{x} \in \mathcal{X}$:

$$\mathbf{x} \in \mathcal{C}_k \iff \mathbf{z} := \varphi(\mathbf{x}) \in \mathsf{B}_k \setminus \mathsf{B}_{\neg k} := \bigcup_{l \neq k} \mathsf{B}_l, \quad \text{or simply} \quad \mathbf{z} \in \mathsf{B}_k.$$

  In other words, $\|\mathbf{z} - \mathbf{c}_k\|_2 \le r_k$ and $\|\mathbf{z} - \mathbf{c}_l\|_2 > r_l$ for all $l \neq k$, or equivalently

$$\forall l \neq k, \quad \langle \mathbf{z}, \mathbf{w}_k\rangle \ge 1 > \langle \mathbf{z}, \mathbf{w}_l\rangle, \quad \text{where} \quad \mathbf{w}_k := \frac{\mathbf{c}_k}{1 - \frac{1}{2}r_k^2}.$$

  Therefore, we may use the simple rule to predict the category $c(\mathbf{x})$ of $\mathbf{x}$ (under transformation φ):

$$c(\mathbf{x}) = \operatorname*{argmax}_{k=1,\ldots,\mathsf{c}} \langle \varphi(\mathbf{x}), \mathbf{w}_k\rangle. \tag{4.7}$$

- inductive learning (recognition): each (test) example may be in several balls or may not be in any ball at all. We may still use the same prediction rule (4.7), but declare "failure"
  - if $\max_{k=1,\ldots,\mathsf{c}} \langle \varphi(\mathbf{x}), \mathbf{w}_k\rangle < 1$: $\mathbf{x}$ does not belong to any existing category;
  - on the other hand, if $|c(\mathbf{x})| > 1$: $\mathbf{x}$ belongs to at least two existing categories. Ambiguous.

Vapnik and Lerner (1963) ended with an announcement of the main result in Remark 4.11, namely how to find the centers $\mathbf{c}_k$ (or equivalently the weights $\mathbf{w}_k$) using (4.6).

Vapnik, Vladimir N. and A. Ya. Lerner (1963). "Pattern Recognition using Generalized Portraits". *Automation and Remote Control*, vol. 24, no. 6, pp. 709–715.

Remark 4.9: More on (Vapnik and Lerner 1963)

Vapnik and Lerner (1963) defined a dataset as indefinite, if there exists some test example $\mathbf{x}$ so that

$$\forall k, \ \mathbf{z} \in \mathsf{B}(\mathbf{c}_k, r_{\neg k}) \setminus \mathsf{B}(\mathbf{c}_k, r_k), \quad \text{where} \quad r_{\neg k} := \max_{l \neq k} r_l,$$

i.e., $\mathbf{z}$ is not in category k but would be if we increase its radius r_k to that of the best alternative $r_{\neg k}$.

Vapnik and Lerner (1963) proposed to use the (positive) quantity

$$I = I(\mathcal{D}) = 1 - \max_{k \neq l} \langle \mathbf{c}_k, \mathbf{c}_l\rangle$$

to measure the distinguishability of our dataset: the bigger I is, the more spread (orthogonal) the centers are hence the easier to distinguish the categories. Using I as the evaluation metric one can sequentially refine the feature transformation φ so that the resulting distinguishability is steadily increased.

Vapnik and Lerner (1963) also noted the product space trick: Let $\{\varphi_j : \mathcal{X} \to \mathcal{H}_j, j = 1, \ldots, m\}$ be a set of feature transformations. Then, w.l.o.g., we can assemble them into a single transformation $\varphi : \mathcal{X} \to \mathcal{H} := \mathcal{H}_1 \times \cdots \times \mathcal{H}_m$.

Vapnik, Vladimir N. and A. Ya. Lerner (1963). "Pattern Recognition using Generalized Portraits". *Automation and Remote Control*, vol. 24, no. 6, pp. 709–715.

Remark 4.10: Linear separability, revisited

Recall our definition of (strict) linear separability of a dataset $\mathcal{D} = \{(\mathbf{x}_i, y_i) \in \mathbb{R}^d \times \{\pm 1\} : i = 1, \ldots, n\}$:

$$\exists \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}, s > 0, \text{ such that } y_i \hat{y}_i \geq s, \ \forall i = 1, \ldots, n, \quad \text{where} \quad \hat{y}_i := \mathbf{w}^\top \mathbf{x}_i + b.$$

Let us now break the above condition for any positive example $y_i = 1$ and any negative example $y_j = -1$:

$$\begin{aligned} \mathbf{w}^\top \mathbf{x}_i + b \geq s \geq -s \geq \mathbf{w}^\top \mathbf{x}_j + b &\iff \mathbf{w}^\top \mathbf{x}_i \geq s - b \geq -s - b \geq \mathbf{w}^\top \mathbf{x}_j \\ &\iff \min_{i: y_i = 1} \mathbf{w}^\top \mathbf{x}_i > \max_{j: y_j = -1} \mathbf{w}^\top \mathbf{x}_j. \end{aligned}$$

It is clear now that the linear separability condition has nothing to do with the offset term b but the normal vector $\mathbf{w}$.

Remark 4.11: History of SVM, continued

In this box we summarize the main result in Vapnik and Chervonenkis (1964).

Inspired by (4.6), Vapnik and Chervonenkis (1964) essentially applied the one-vs-all reduction (see Remark 1.36) and arrived at what they called the *optimal approximation*:

$$\begin{aligned} \max_{\|\mathbf{w}\| = 1} \quad & \min_{i: y_i = 1} \mathbf{w}^\top \mathbf{x}_i \\ \text{s.t.} \quad & \min_{i: y_i = 1} \mathbf{w}^\top \mathbf{x}_i \geq \max_{j: y_j = -1} \mathbf{w}^\top \mathbf{x}_j. \end{aligned} \tag{4.8}$$

According to Remark 4.10, the last condition is equivalent to requiring the dataset to be linearly separable (strictly or not). Reintroducing the offset b and applying the standardization trick in Remark 4.5:

$$\begin{aligned} \max_{\mathbf{w}, b, \delta} \quad & \delta - b \\ \text{s.t.} \quad & \|\mathbf{w}\| = 1 \xrightarrow{\text{assuming obj} > 0} \|\mathbf{w}\| \leq 1 \\ & y_i \hat{y}_i \geq \delta \geq 0, \quad \hat{y}_i := \mathbf{w}^\top \mathbf{x}_i + b, \ i = 1, \ldots, n. \end{aligned} \tag{4.9}$$

The above problem obviously admits a solution iff the dataset is linearly separable. Moreover, if we assume the maximum objective is strictly positive (which is different from strict linear separability: take say $\mathcal{D} = \{(-1, +), (-2, -)\}$), then we can relax the unit norm constraint, in which case the optimal $\mathbf{w}$ is unique.

This formulation (4.9) of Vapnik and Chervonenkis differs from our previous one (4.5) mainly in the objective function: $\delta - b$ vs. δ, i.e. Vapnik and Chervonenkis always subtract the offset from the minimum margin. This difference can be significant though, see Example 4.12 below for an illustration.

The main result in (Vapnik and Chervonenkis 1964), aside from the formulation in (4.8), is the derivation of its Lagrangian dual, which is quite a routine derivation nowadays. Nevertheless, we reproduce the original argument of Vapnik and Chervonenkis for historical interests.

Define $C(\mathbf{w}) = \min_{i: y_i = 1} \mathbf{w}^\top \mathbf{x}_i$, and define the set of support vectors $S(\mathbf{w}) := \{y_i \mathbf{x}_i : \mathbf{w}^\top \mathbf{x}_i = C(\mathbf{w})\}$. By definition there is always a positive support vector while there may not be any negative support vector (consider say $\mathcal{D} = \{(1, -), (2, +)\}$). Recall that Vapnik and Chervonenkis assumed the optimal objective $C^\star = C(\mathbf{w}^\star)$ in (4.6) is strictly positive, hence the uniqueness of the optimal solution $\mathbf{w}^\star$. By restricting the norm $\|\cdot\| = \|\cdot\|_2$, Vapnik and Chervonenkis made the following observations:

- $\mathbf{w}^\star$ is a conic combination of support vectors $S = S(\mathbf{w}^\star)$. Suppose not, let P denote the ℓ_2 projector onto the conic hull of support vectors. Let $\mathbf{w}_\eta = (I - P)\mathbf{w}^\star + \eta P \mathbf{w}^\star$. For any support vector $y\mathbf{x} \in S$:

$$y[\langle \mathbf{w}_\eta, \mathbf{x} \rangle - \eta \langle \mathbf{w}^\star, \mathbf{x} \rangle] = (1 - \eta) \langle \mathbf{w}^\star - P\mathbf{w}^\star, y\mathbf{x} \rangle = (1 - \eta) \langle \mathbf{w}^\star - P\mathbf{w}^\star, (y\mathbf{x} + P\mathbf{w}^\star) - P\mathbf{w}^\star \rangle \geq 0,$$

  since P is the projector onto the conic hull and $\eta \geq 1$. Since $C(\mathbf{w}^\star) > 0$ by assumption, slightly increase η from 1 to $1 + \epsilon$ will maintain all constraints but increase the objective in (4.6), contradiction to the optimality of $\mathbf{w}^\star$.

- If we normalize the weight vector $\mathbf{w}^* = \mathbf{w}^\star / C(\mathbf{w}^\star)$, the constraint in (4.6) is not affected but the objective $C(\mathbf{w}^*)$ now becomes unit. Of course, $\mathbf{w}^*$ remains to be a conic combination of support vectors:

$$\mathbf{w}^* = \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i, \quad \alpha_i \geq 0, \quad \alpha_i(\mathbf{x}_i^\top \mathbf{w}^* - 1) = 0, \quad y_i(\mathbf{x}_i^\top \mathbf{w}^* - 1) \geq 0, \quad i = 1, \ldots, n, \tag{4.10}$$

  where the conditions follow from our definition of support vectors and is known as the KKT condition.

- Vapnik and Chervonenkis mentioned the following differential system:

$$\frac{\mathrm{d}\boldsymbol{\alpha}}{\mathrm{d}t} = -\epsilon\boldsymbol{\alpha} + [\mathbf{y} - (K \odot \mathbf{y}\mathbf{y}^\top)\boldsymbol{\alpha}]^+, \quad K_{ij} := \langle \mathbf{x}_i, \mathbf{x}_j \rangle,$$

  whose equilibrium will approach the KKT condition hence the solution in (4.10) as $\epsilon \to 0$. Vapnik and Chervonenkis concluded that to compute $\boldsymbol{\alpha}$, we need only the dot product K. Moreover, to reconstruct $\mathbf{w}$, only the support vectors from the training set are needed.

- To perform testing, we compare $\mathbf{x}^\top \mathbf{w}^*$ with threshold 1 (see the last condition in (4.10)). Or equivalently, using uniqueness we can recover $\mathbf{w}^\star = \mathbf{w}^*/\|\mathbf{w}^*\|_2$, where

$$\|\mathbf{w}^*\|_2^2 = \langle \mathbf{w}^*, \mathbf{w}^* \rangle = \left\langle \mathbf{w}^*, \sum_{i=1}^n \alpha_i y_i \mathbf{x}_i \right\rangle = \sum_{i=1}^n \alpha_i y_i \langle \mathbf{w}^*, \mathbf{x}_i \rangle = \boldsymbol{\alpha}^\top \mathbf{y}.$$

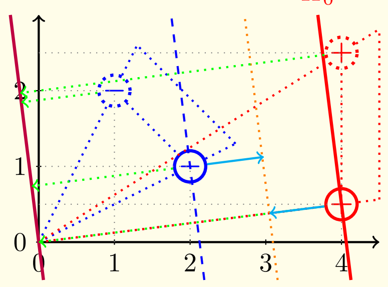
  Thus, we may also compare $\mathbf{x}^\top \mathbf{w}^\star$ with the threshold $\frac{1}{\sqrt{\boldsymbol{\alpha}^\top \mathbf{y}}}$. Usually we prefer to use $\mathbf{w}^*$ because its threshold is normalized: in case there are multiple classes, we can then use the argmax rule: $\hat{y} = \operatorname{argmax}_k \mathbf{w}_k^\top \mathbf{x}$. However, note that this decision boundary corresponds to the hyperplane that first passes through a positive example!

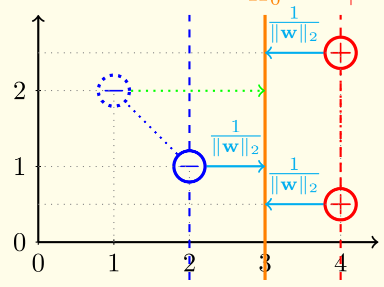
- Vapnik and Chervonenkis mentioned the possibility to dynamically update the dual variables $\boldsymbol{\alpha}$: upon making a mistake on an example $(\mathbf{x}, y)$, we augment the support vectors S with $(\mathbf{x}, y)$ and recompute the dual variables on the "effective training set" S. Again, only dot products are needed.

Vapnik, Vladimir N. and A. Ya. Chervonenkis (1964). "On a class of perceptrons". *Automation and Remote Control*, vol. 25, no. 1, pp. 112–120.

Example 4.12: SVM, old and new

As illustrated in the following figure, the optimal approximation in Remark 4.11 can behave very differently from the familiar SVM formulation (4.11) in Remark 4.13. To verify that the left purple solid line in the left plot is indeed optimal, simply note that the minimum distance from positive examples to any hyperplane passing through the origin (e.g. the objective in (4.8)) is at most the minimum distance from positive examples to the origin, which the left purple line achieves.




H_-
H_0
H_+
$\frac{1}{\|\mathbf{w}\|_2}$
$\frac{1}{\|\mathbf{w}\|_2}$
$\frac{1}{\|\mathbf{w}\|_2}$
0
1
2
0
1
2
3
4

Remark 4.13: Removing homogeneity by normalizing offset

A different way to remove the scaling-invariance mentioned in Definition 4.2 is to perform normalization on the offset so that

$$\min_{i=1,\ldots,n} y_i \hat{\mathsf{y}}_i = \delta,$$

where $\delta > 0$ is any fixed constant. When the dataset $\mathcal{D}$ is indeed (strictly) linearly separable, this normalization can always be achieved (simply by scaling $\mathbf{w}$). After normalizing this way, we can simplify (4.2) as:

$$\max_{\mathbf{w}} \ \frac{\delta}{\|\mathbf{w}\|}, \quad \text{s.t.} \quad \min_{i=1,\ldots,n} y_i \mathsf{y}_i = \delta.$$

We remind again that δ here is any fixed positive constant and we are *not* optimizing it (in contrast to what we did in Remark 4.6). Applying some elementary transformations (that do not change the minimizer) we arrive at the usual formulation of SVM (due to Boser et al. (1992)):

$$\begin{aligned} \min_{\mathbf{w}} \quad & \tfrac{1}{2}\|\mathbf{w}\|^2 \\ \text{s.t.} \quad & y_i \hat{\mathsf{y}}_i \geq \delta, \ \forall i = 1, \ldots, n. \end{aligned} \tag{4.11}$$

It is clear that the actual value of the positive constant δ is immaterial. Most often, we simply set $\delta = 1$, which is our default choice in the rest of this note.

The formulation (4.11) only makes sense on (strictly) linearly separable datasets, unlike our original formulation (4.3).

Boser, Bernhard E., Isabelle M. Guyon, and Vladimir N. Vapnik (1992). "A Training Algorithm for Optimal Margin Classiers". In: *COLT*, pp. 144–152.

Alert 4.14: Any positive number but not zero

Note that in the familiar SVM formulation (4.11), we can choose δ to be any (strictly) positive number (which amounts to a simple change of scale). However, we cannot set $\delta = 0$, for otherwise the solution could be trivially $\mathbf{w} = \mathbf{0}, b = 0$.

Remark 4.15: Perceptron vs. SVM

We can formulate perceptron as the following feasibility problem:

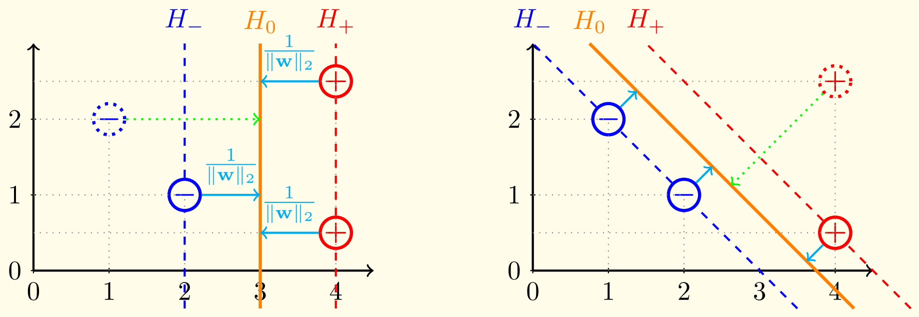
$$\begin{aligned} \min_{\mathbf{w}} \quad & 0 \\ \text{s.t.} \quad & y_i \hat{\mathsf{y}}_i \geq \delta, \ \forall i = 1, \ldots, n, \end{aligned}$$

where as before $\delta > 0$ is any fixed constant.

Unlike SVM, the objective function of perceptron is the trivial constant 0 function, i.e., we are not trying to optimize anything (such as distance/margin) other than satisfying a bunch of constraints (separating the positives from the negatives). Computationally, perceptron belongs to linear programming (LP), i.e., when the objective function and all constraints are linear functions. In contrast, SVM belongs to the slightly more complicated quadratic programming (QP): the objective function is a quadratic function while all constraints are still linear. Needless to say, LP $\subsetneq$ QP.

Remark 4.16: Three parallel hyperplanes

Geometrically, we have the following intuitive picture. As an example, the dataset $\mathcal{D}$ consists of 2 positive and 2 negative examples. The left figure shows the SVM solution, and for comparison the right figure depicts a suboptimal solution. We will see momentarily why the left solution is optimal.


H_-
H_0
H_+
$\frac{1}{\|\mathbf{w}\|_2}$
$\frac{1}{\|\mathbf{w}\|_2}$
$\frac{1}{\|\mathbf{w}\|_2}$
H_-
H_0
H_+

To understand the above figure, let us take a closer look at the SVM formulation (4.11), where w.l.o.g. we choose $\delta = 1$. Recall that the dataset $\mathcal{D}$ contains at least 1 positive example and 1 negative example (so that $\mathbf{w} = \mathbf{0}$ is ruled out). Let us breakdown the constraints in (4.11):

$$\left.\begin{aligned} \mathbf{w}^\top \mathbf{x}_i + b \geq 1, \quad & y_i = 1 \\ \mathbf{w}^\top \mathbf{x}_i + b \leq -1, \quad & y_i = -1 \end{aligned}\right\} \iff 1 - \min_{i:y_i=1} \mathbf{w}^\top \mathbf{x}_i \leq b \leq -1 - \max_{i:y_i=-1} \mathbf{w}^\top \mathbf{x}_i.$$

If one of the inequalities is strict, say the left one, then we can decrease b slightly so that both inequalities are strict. But then we can scale down $\mathbf{w}$ and b without violating any constraint while decreasing the objective $\frac{1}{2}\|\mathbf{w}\|^2$ further. Therefore, at minimum, we must have

$$1 - \min_{i:y_i=1} \mathbf{w}^\top \mathbf{x}_i = b = -1 - \max_{i:y_i=-1} \mathbf{w}^\top \mathbf{x}_i, \ \textit{i.e.}, \ y_i \hat{y}_i = 1 \text{ for at least one } y_i = 1 \text{ and one } y_i = -1.$$

Given the SVM solution $(\mathbf{w}, b)$, we can now define three parallel hyperplanes:

$$\begin{aligned} H_0 &:= \{\mathbf{x} : \mathbf{w}^\top \mathbf{x} + b = 0\} \\ H_+ &:= \{\mathbf{x} : \mathbf{w}^\top \mathbf{x} + b = 1\} && \text{(we choose } \delta = 1) \\ H_- &:= \{\mathbf{x} : \mathbf{w}^\top \mathbf{x} + b = -1\}. \end{aligned}$$

The hyperplane H_0 is the decision boundary of SVM: any point above or below it is classified as positive or negative, respectively, i.e. $y = \mathrm{sign}(\mathbf{w}^\top \mathbf{x} + b)$. The hyperplane H_+ is the translate of H_0 on which for the first time we pass through some positive examples, and similarly for H_-. Note that there are no training examples between H_- and H_+ (a dead zone), with H_0 at the middle between H_- and H_+. More precisely, we can compute the distance between H_0 and H_+:

$$\begin{aligned} \mathrm{dist}(H_+, H_0) &:= \min_{\mathbf{p} \in H_+} \min_{\mathbf{q} \in H_0} \|\mathbf{p} - \mathbf{q}\|_\circ \\ &= \min_{i:y_i=1} \mathrm{dist}(\mathbf{x}_i, H_0) && \text{(since } H_+ \text{ first passes through positive examples)} \\ &= \frac{1}{\|\mathbf{w}\|} && \text{(see (4.1))} \\ &= \min_{i:y_i=-1} \mathrm{dist}(\mathbf{x}_i, H_0) && \text{(since } H_- \text{ first passes through negative examples)} \\ &= \mathrm{dist}(H_-, H_0). \end{aligned}$$

Exercise 4.17: Uniqueness of w

For the ℓ_2 norm, prove the parallelogram equality

$$\|\mathbf{w}_1 + \mathbf{w}_2\|_2^2 + \|\mathbf{w}_1 - \mathbf{w}_2\|_2^2 = 2(\|\mathbf{w}_1\|_2^2 + \|\mathbf{w}_2\|_2^2).$$

(The parallelogram law, in fact, characterizes norms that are induced by an inner product). With this choice $\|\cdot\| = \|\cdot\|_2$, prove

- that the SVM weight vector $\mathbf{w}$ is unique;
- that the SVM offset b is also unique.

Definition 4.18: Convex set

A set $C \subseteq \mathbb{R}^d$ is called convex iff for all $\mathbf{x}, \mathbf{z} \in C$ and for all $\alpha \in [0, 1]$ we have

$$(1-\alpha)\mathbf{x} + \alpha\mathbf{z} \in C,$$

i.e., the line segment connecting any two points in C remains in C.

By convention the empty set is convex. Obviously, the universe $\mathbb{R}^d$, being a vector space, is convex.

Exercise 4.19: Basic properties of convex sets

Prove the following:

- The intersection $\bigcap_{\gamma\in\Gamma} C_\gamma$ of a collection of convex sets $\{C_\gamma\}_{\gamma\in\Gamma}$ is convex.
- A set in $\mathbb{R}$ (the real line) is convex iff it is an interval (not necessarily bounded or closed).
- The union of two convex sets need not be convex.
- The complement of a convex set need not be convex.
- Hyperplanes $H_0 := \{\mathbf{x} \in \mathbb{R}^d : \mathbf{w}^\top\mathbf{x} + b = 0\}$ are convex.
- Halfspaces $H_\leq := \{\mathbf{x} \in \mathbb{R}^d : \mathbf{w}^\top\mathbf{x} + b \leq 0\}$ are convex.

(In fact, a celebrated result in convex analysis shows that any closed convex set is an intersection of halfspaces.)

Definition 4.20: Convex hull

The convex hull $\mathrm{conv}(A)$ of an arbitrary set A is the intersection of all convex supersets of A, i.e.,

$$\mathrm{conv}(A) := \bigcap_{\text{convex } C \supseteq A} C.$$

In other words, the convex hull is the "smallest" convex superset.

Exercise 4.21: Convex hull as convex combination

We define the convex combination of a finite set of points $\mathbf{x}_1, \ldots, \mathbf{x}_n$ as any point $\mathbf{x} = \sum_{i=1}^n \alpha_i \mathbf{x}_i$ with coefficients $\boldsymbol{\alpha} \geq 0, \mathbf{1}^\top \boldsymbol{\alpha} = 1$, i.e. $\boldsymbol{\alpha} \in \Delta_{n-1}$. Prove that for any $A \subseteq \mathbb{R}^d$:

$$\operatorname{conv}(A) = \left\{ \mathbf{x} = \sum_{i=1}^{n} \alpha_i \mathbf{x}_i : n \in \mathbb{N}, \boldsymbol{\alpha} \in \Delta_{n-1}, \mathbf{x}_i \in A \right\},$$

i.e., the convex hull is simply the set of all convex combinations of points in A.

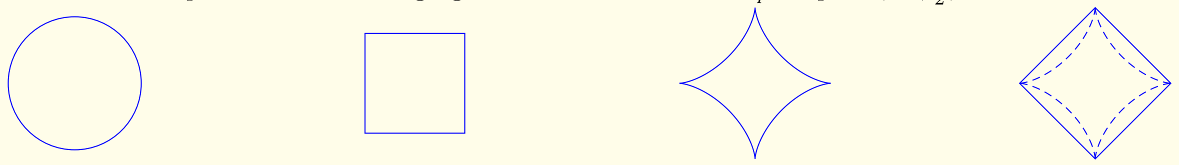
(The celebrated Carathéodory theorem allows us to restrict $n \leq d+1$, and $n \leq d$ if A is connected.)

Exercise 4.22: Unit balls of norms are convex

Recall that the unit ball of the ℓ_p "norm" is defined as:

$$\mathsf{B}_p := \{\mathbf{x} : \ \|\mathbf{x}\|_p \leq 1\},$$

which is convex iff $p \geq 1$. The following figure shows the unit ball B_p for $p = 2, \infty, \frac{1}{2}, 1$.



As shown above:

$$\operatorname{conv}(\mathsf{B}_{\frac{1}{2}}) = \mathsf{B}_1.$$

- For what values of p and q do we have $\operatorname{conv}(\mathsf{B}_p) = \mathsf{B}_q$?
- For what value of p is the sphere $\mathsf{S}_p := \{\mathbf{x} : \ \|\mathbf{x}\|_p = 1\} = \partial \mathsf{B}_p$ convex?

Remark 4.23: The first dual view of SVM (Rosen 1965)

Rosen (1965) was among the first few people who recognized that a dataset $\mathcal{D}$ is (strictly) linearly separable (see Definition 1.24) iff

$$\operatorname{conv}(\mathcal{D}^+) \cap \operatorname{conv}(\mathcal{D}^-) = \emptyset, \quad \text{where} \quad \mathcal{D}^\pm := \{\mathbf{x}_i \in \mathcal{D} : y_i = \pm 1\}.$$

(Prove the only if part by yourself; to see the if part, note that the convex hull of a compact set (e.g. finite set) is compact, and disjoint compact sets can be strictly separated by a hyperplane, due to the celebrated Hahn-Banach Theorem.)

To test if a given dataset $\mathcal{D}$ is (strictly) linearly separable, Rosen's idea was to compute the minimum (Euclidean) distance between the (convex hulls of the) two classes. In his Eq (2.5), after applying the standardization trick, see Remark 4.5, Rosen proposed exactly (up to a constant $\frac{1}{2}$) the hard-margin SVM formulation (4.4). Then, to get an equivalent *convex* formulation, Rosen (1965) did some simple algebraic manipulations to arrive at the familiar hard-margin SVM in (4.11) (his Eq (2.6), again, up to a constant $\frac{1}{2}$). Rosen (1965) proved the uniqueness of the hard-margin SVM solution, and he further proved that the number of support vectors can be bounded by $d+1$. Rosen (1965) also discussed how to separate more than two classes, using basically the one-vs-one and the one-vs-all reductions (see Remark 1.36). It would seem appropriate to attribute our hard-margin SVM formulations in (4.4) and (4.11) to Rosen (1965).

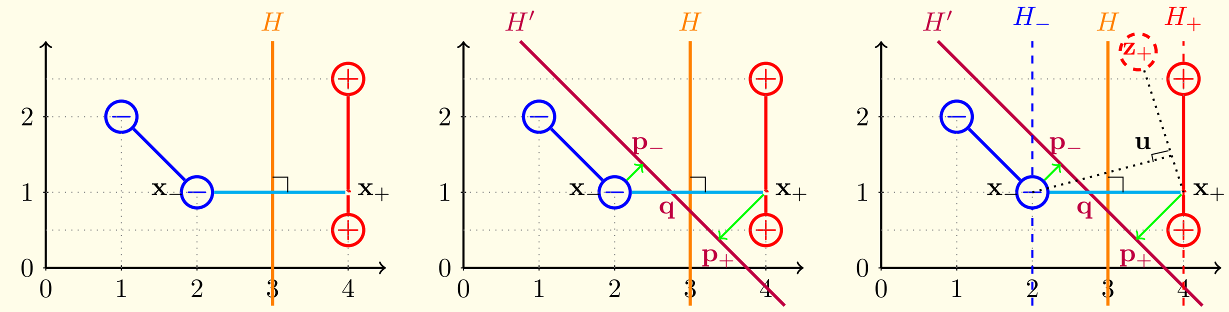
Rosen, J.B (1965). "Pattern separation by convex programming". *Journal of Mathematical Analysis and Applications*, vol. 10, no. 1, pp. 123–134.

Remark 4.24: More on linear separability detection (Mangasarian 1965)

omitted.

Mangasarian, O. L. (1965). "Linear and Nonlinear Separation of Patterns by Linear Programming". *Operations Research*, vol. 13, no. 3, pp. 444–452.

Remark 4.25: Dual view of SVM, as bisector of minimum distance pair



In Definition 4.2 we defined SVM as maximizing the minimum distance of training examples to the decision boundary H_0. We now provide a dual view which geometrically is very appealing.

- We first make a simple observation about a (strict) separating hyperplane H:

$$\left.\begin{array}{ll}\langle \mathbf{w}, \mathbf{x}_i\rangle + b > 0, & \text{if } \mathbf{x}_i \in \mathcal{D}^+ := \{\mathbf{x}_j : y_j = 1\} \\ \langle \mathbf{w}, \mathbf{x}_i\rangle + b < 0, & \text{if } \mathbf{x}_i \in \mathcal{D}^- := \{\mathbf{x}_j : y_j = -1\}\end{array}\right\} \implies \begin{cases}\langle \mathbf{w}, \mathbf{x}\rangle + b > 0, & \text{if } \mathbf{x} \in \operatorname{conv}(\mathcal{D}^+) \\ \langle \mathbf{w}, \mathbf{x}\rangle + b < 0, & \text{if } \mathbf{x} \in \operatorname{conv}(\mathcal{D}^-)\end{cases},$$

  i.e., H also (strictly) separates the convex hulls of positive examples and negative ones.

- The second observation we make is about the minimum distance of all positive (negative) examples to a separating hyperplane:

$$\min_{\mathbf{x}\in\mathcal{D}^\pm} \operatorname{dist}(\mathbf{x}, H) = \min_{\mathbf{x}\in\mathcal{D}^\pm} \frac{\pm(\mathbf{w}^\top \mathbf{x} + b)}{\|\mathbf{w}\|} = \min_{\mathbf{x}\in\operatorname{conv}(\mathcal{D}^\pm)} \frac{\pm(\mathbf{w}^\top \mathbf{x} + b)}{\|\mathbf{w}\|} = \min_{\mathbf{x}\in\operatorname{conv}(\mathcal{D}^\pm)} \operatorname{dist}(\mathbf{x}, H),$$

  where the first equality follows from (4.1), the second from linearity, and the third from our observation above. In other words, we could replace the datasets $\mathcal{D}^\pm$ with their convex hulls.

- Based on the second observation, we now find the pair of $\mathbf{x}_+ \in \operatorname{conv}(\mathcal{D}_+)$ and $\mathbf{x}_- \in \operatorname{conv}(\mathcal{D}_-)$ so that $\operatorname{dist}(\mathbf{x}_+, \mathbf{x}_-)$ achieves the minimum distance among all pairs from the two convex hulls. We connect the segment from $\mathbf{x}_+$ to $\mathbf{x}_-$ and find its bisector, a separating hyperplane H that passes the middle point $\frac{1}{2}(\mathbf{x}_+ + \mathbf{x}_-)$ with normal vector proportional to $\partial \left[\frac{1}{2}\|\mathbf{x}_+ - \mathbf{x}_-\|^2\right]$. We claim that

$$\min_{\mathbf{x}\in\mathcal{D}^\pm} \operatorname{dist}(\mathbf{x}, H) = \min_{\mathbf{x}\in\operatorname{conv}(\mathcal{D}^\pm)} \operatorname{dist}(\mathbf{x}, H) = \tfrac{1}{2}\operatorname{dist}(\mathbf{x}_+, \mathbf{x}_-) = \tfrac{1}{2}\operatorname{dist}(\operatorname{conv}(\mathcal{D}^+), \operatorname{conv}(\mathcal{D}^-)).$$

  To see the second equality, we translate H in parallel until it passes $\mathbf{x}_+$ and $\mathbf{x}_-$, and obtain hyperplanes H_+ and H_-, respectively. Since H is a bisector of the line segment $\mathbf{x}_+\mathbf{x}_-$,

$$\operatorname{dist}(H_+, H) = \operatorname{dist}(H_-, H) = \tfrac{1}{2}\operatorname{dist}(\mathbf{x}_+, \mathbf{x}_-).$$

  We are left to prove there is no point in $\operatorname{conv}(\mathcal{D}^\pm)$ that lies between H_- and H_+. Suppose, for the sake of contradiction, there is some $\mathbf{z}_+ \in \operatorname{conv}(\mathcal{D}^+)$ that lies between H_- and H_+. The remaining proof for the Euclidean case where $\|\cdot\| = \|\cdot\|_2$ is depicted above: We know the angle $\angle \mathbf{x}_-\mathbf{x}_+\mathbf{z}_+ < 90°$. If we move a point $\mathbf{u}$ on the segment $\mathbf{z}_+\mathbf{x}_+$ from $\mathbf{z}_+$ to $\mathbf{x}_+$, because the angle $\angle \mathbf{u}\mathbf{x}_-\mathbf{x}_+ \to 0°$, so eventually we will have $\angle \mathbf{x}_- \mathbf{u} \mathbf{x}_+ \geq 90^\circ$, in which case we would have $\mathrm{dist}(\mathbf{u}, \mathbf{x}_-) < \mathrm{dist}(\mathbf{x}_+, \mathbf{x}_-)$. Since $\mathbf{u} \in \mathrm{conv}(\mathcal{D}^+)$, we have a contradiction:

$$\mathrm{dist}(\mathbf{u}, \mathbf{x}_-) \geq \mathrm{dist}(\mathrm{conv}(\mathcal{D}^+), \mathrm{conv}(\mathcal{D}^-)) = \mathrm{dist}(\mathbf{x}_+, \mathbf{x}_-) > \mathrm{dist}(\mathbf{u}, \mathbf{x}_-).$$

The proof for any norm is as follows: Since the line segment $\mathbf{z}_+\mathbf{x}_+ \in \mathrm{conv}(\mathcal{D}^+)$ and by definition $\mathrm{dist}(\mathbf{x}_+, \mathbf{x}_-) = \mathrm{dist}(\mathrm{conv}(\mathcal{D}^+), \mathrm{conv}(\mathcal{D}^-))$, we know for any $\mathbf{u}_\lambda = \lambda \mathbf{z}_+ + (1-\lambda)\mathbf{x}_+$ on the line segment, $f(\lambda) := \mathrm{dist}(\mathbf{u}_\lambda, \mathbf{x}_-) \geq \mathrm{dist}(\mathbf{x}_+, \mathbf{x}_-) = f(0)$, i.e. the minimum of $f(\lambda)$ over the interval $\lambda \in [0,1]$ is achieved at $\lambda = 0$. Since $f(\lambda)$ is convex its right derivative at $\lambda = 0$, namely $\langle \mathbf{w}, \mathbf{z}_+ - \mathbf{x}_+ \rangle$, where $\mathbf{w} \in \partial \|\mathbf{x}_+ - \mathbf{x}_-\|$, must be positive. But we know the hyperplane $H_+ = \{\mathbf{x} : \mathbf{w}^\top(\mathbf{x} - \mathbf{x}_+) = 0\}$ and the middle point $\frac{1}{2}(\mathbf{x}_+ + \mathbf{x}_-)$ is on the left side of H_+, hence $\mathbf{z}_+$ is on the right side of H_+, contradiction.

- We can finally claim that H is the SVM solution, i.e., H maximizes the minimum distance to every training examples in $\mathcal{D}$. Indeed, let H' be any other separating hyperplane. According to our first observation above, H' intersects with the line segment $\mathbf{x}_+\mathbf{x}_-$ at some point $\mathbf{q}$ (due to separability). Define $\mathbf{p}_\pm$ as the projection of $\mathbf{x}_\pm$ onto the hyperplane H', and since $\mathbf{q} \in H'$,

$$\mathrm{dist}(\mathbf{x}_\pm, \mathbf{p}_\pm) = \mathrm{dist}(\mathbf{x}_\pm, H') \leq \mathrm{dist}(\mathbf{x}_\pm, \mathbf{q}).$$

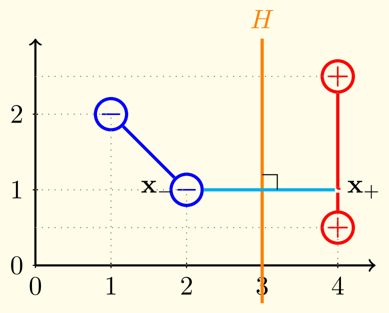
Therefore, using our second and third observations above:

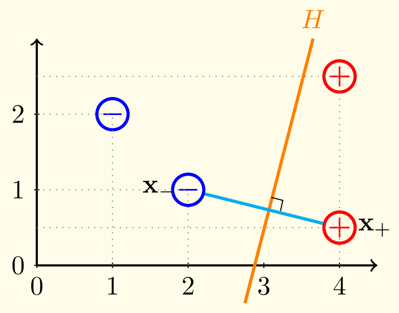
$$\begin{aligned}
\min_{\mathbf{x} \in \mathcal{D}^\pm} \mathrm{dist}(\mathbf{x}, H') = \min_{\mathbf{x} \in \mathrm{conv}(\mathcal{D}^\pm)} \mathrm{dist}(\mathbf{x}, H') &\leq \mathrm{dist}(\mathbf{x}_+, \mathbf{p}_+) \wedge \mathrm{dist}(\mathbf{x}_-, \mathbf{p}_-) \\
&\leq \tfrac{1}{2}[\mathrm{dist}(\mathbf{x}_+, \mathbf{p}_+) + \mathrm{dist}(\mathbf{x}_-, \mathbf{p}_-)] \\
&\leq \tfrac{1}{2}[\mathrm{dist}(\mathbf{x}_+, \mathbf{q}) + \mathrm{dist}(\mathbf{x}_-, \mathbf{q})] \\
&= \tfrac{1}{2}\mathrm{dist}(\mathbf{x}_+, \mathbf{x}_-) \\
&= \min_{\mathbf{x} \in \mathrm{conv}(\mathcal{D}^\pm)} \mathrm{dist}(\mathbf{x}, H) = \min_{\mathbf{x} \in \mathcal{D}^\pm} \mathrm{dist}(\mathbf{x}, H).
\end{aligned}$$

Exercise 4.26: Necessity of convex hull

In Remark 4.25, we picked the pair $\mathbf{x}_+$ and $\mathbf{x}_-$ from the two convex hulls $\mathcal{D}^\pm$ of the positive and negative examples, respectively. Prove the following:

- One of $\mathbf{x}_+$ and $\mathbf{x}_-$ can be chosen from the original datasets $\mathcal{D}^\pm$.
- Not both of $\mathbf{x}_+$ and $\mathbf{x}_-$ may be chosen from the original datasets $\mathcal{D}^\pm$.
- What observation(s) in Remark 4.25 might fail if we insist in picking both $\mathbf{x}_+$ and $\mathbf{x}_-$ from the original datasets $\mathcal{D}^\pm$?





Remark 4.27: SVM dual, from geometry to algebra

We complement the geometric dual view of SVM in Remark 4.25 with a "simpler" algebraic view. Applying scaling we may assume the weight vector $\mathbf{w}$ of a separating hyperplane $H_{\mathbf{w}}$ is normalized. Then, we maximize the minimum distance as follows:

$$\begin{aligned}
\max_{\|\mathbf{w}\|=1,b} \operatorname{dist}(\mathcal{D}^+, H_{\mathbf{w}}) \wedge \operatorname{dist}(\mathcal{D}^-, H_{\mathbf{w}}) &= \max_{\|\mathbf{w}\|=1,b} \left[\min_{\mathbf{x}_+\in\mathcal{D}^+} (\mathbf{w}^\top \mathbf{x}_+ + b) \wedge \min_{\mathbf{x}_-\in\mathcal{D}^-} -(\mathbf{w}^\top \mathbf{x}_- + b) \right] \\
&= \max_{\|\mathbf{w}\|=1,b} \left[\min_{\mathbf{x}_\pm\in\mathcal{D}^\pm, t\in[0,1]} t(\mathbf{w}^\top \mathbf{x}_+ + b) + (1-t)(-\mathbf{w}^\top \mathbf{x}_- - b) \right] \\
&= \max_{\|\mathbf{w}\|\le 1,b} \left[\min_{\mathbf{x}_+\in t\operatorname{conv}(\mathcal{D}^+), \mathbf{x}_-\in(1-t)\operatorname{conv}(\mathcal{D}_-), t\in[0,1]} \mathbf{w}^\top(\mathbf{x}_+ - \mathbf{x}_-) + b(2t-1) \right] \\
&= \min_{\mathbf{x}_+\in t\operatorname{conv}(\mathcal{D}^+), \mathbf{x}_-\in(1-t)\operatorname{conv}(\mathcal{D}_-), t\in[0,1]} \max_{\|\mathbf{w}\|\le 1,b} \left[\mathbf{w}^\top(\mathbf{x}_+ - \mathbf{x}_-) + b(2t-1) \right] \\
&= \min_{\mathbf{x}_+\in\frac{1}{2}\operatorname{conv}(\mathcal{D}^+), \mathbf{x}_-\in\frac{1}{2}\operatorname{conv}(\mathcal{D}_-)} \max_{\|\mathbf{w}\|\le 1} \mathbf{w}^\top(\mathbf{x}_+ - \mathbf{x}_-) \\
&= \min_{\mathbf{x}_+\in\frac{1}{2}\operatorname{conv}(\mathcal{D}^+), \mathbf{x}_-\in\frac{1}{2}\operatorname{conv}(\mathcal{D}_-)} \|\mathbf{x}_+ - \mathbf{x}_-\|_\circ \\
&= \tfrac{1}{2}\operatorname{dist}(\operatorname{conv}(\mathcal{D}^+), \operatorname{conv}(\mathcal{D}^-)),
\end{aligned}$$

where in the third equality we used linearity to replace with convex hulls, which then allowed us to apply the minimax theorem to swap max with min. The sixth equality follows from Cauchy-Schwarz and is attained when $\mathbf{w} \propto \mathbf{x}_+ - \mathbf{x}_-$, i.e. when $H_{\mathbf{w}}$ is a bisector.

5 Soft-margin Support Vector Machines

Goal

Extend hard-margin SVM to handle linearly inseparable data.

Alert 5.1: Convention

Gray boxes are not required hence can be omitted for unenthusiastic readers.

This note is likely to be updated again soon.

6 Reproducing Kernels

Goal

Understand the kernel trick for training nonlinear classifiers with linear techniques. Reproducing kernels.

Alert 6.1: Convention

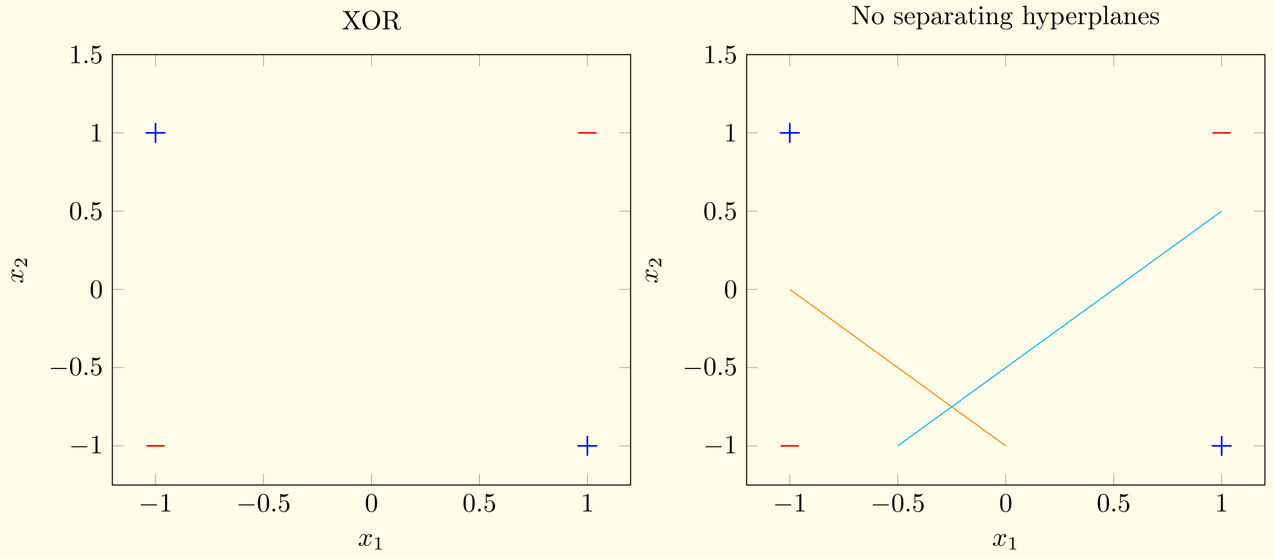
Gray boxes are not required hence can be omitted for unenthusiastic readers.

This note is likely to be updated again soon.

[Remes et al., 2017] Sami Remes, Markus Heinonen, and Samuel Kaski. Non-stationary spectral kernels. In Advances in Neural Information Processing Systems 30 (NIPS), pages 4642–4651, 2017.

[Samo and Roberts, 2015] Yves-Laurent Kom Samo and Stephen Roberts. Generalized spectral kernels. arXiv preprint arXiv:1506.02236, 2015.

Example 6.2: XOR problem



The famous XOR problem is a simple binary classification problem with dataset

$$\begin{aligned}
\mathcal{D} = \{\mathbf{x}_1 &= (1;1), & y_1 &= -1,\\
\mathbf{x}_2 &= (-1;-1), & y_2 &= -1,\\
\mathbf{x}_3 &= (1;-1), & y_3 &= 1,\\
\mathbf{x}_4 &= (-1;1), & y_4 &= 1\},
\end{aligned}$$

which is perfectly classified by the (nonlinear) XOR rule:

$$y = x_1 \ \mathsf{xor}\ x_2 := -x_1 x_2.$$

However, as illustrated on the right plot, no separating hyperplane can achieve 0 error on $\mathcal{D}$. Indeed, a hyperplane parameterized by $(\mathbf{w}, b)$ (strictly) linearly separates $\mathcal{D}$ iff for all i, $y_i \hat{\mathsf{y}}_i > 0$, where as usual $\mathsf{y} = \mathbf{w}^\top \mathbf{x} + b$. On the XOR dataset $\mathcal{D}$, we would obtain

$$\begin{aligned}
y_1\hat{\mathsf{y}}_1 &= -(w_1 + w_2 + b) > 0\\
y_2\hat{\mathsf{y}}_2 &= -(-w_1 - w_2 + b) > 0\\
y_3\hat{\mathsf{y}}_3 &= (w_1 - w_2 + b) > 0
\end{aligned}$$

$$y_4\hat{y}_4 = (-w_1 + w_2 + b) > 0.$$

Adding the 4 inequalities we obtain $0 > 0$, which is absurd, hence there cannot exist a (strictly) separating hyperplane for $\mathcal{D}$.

Example 6.3: Dimension of Gaussian kernel feature space

Since the Gaussian density is a kernel, we know there exists a feature transformation $\varphi : \mathbb{R}^d \to \mathcal{H}$ so that

$$\langle \varphi(\mathbf{x}), \varphi(\mathbf{x}')\rangle = \exp(-\|\mathbf{x} - \mathbf{x}'\|_2^2/\sigma),$$

where $\sigma > 0$ is any fixed positive number. We prove that $\dim(\mathcal{H})$ cannot be finite below.

For the sake of contradiction, suppose $\dim(\mathcal{H}) = h < \infty$. For any n points $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$ we define the matrix

$$\Phi = [\varphi(\mathbf{x}_1), \ldots, \varphi(\mathbf{x}_n)] \in \mathbb{R}^{h\times n}.$$

It is immediate that $\text{rank}(\Phi) \le n \wedge h$.

Next, we choose a set of distinct points $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$ and define the matrix

$$K_{ij} = \exp(-\|\mathbf{x}_i - \mathbf{x}_j\|_2^2/\sigma).$$

We claim that $\text{rank}(K) = n$. But we also know $K = \Phi^\top\Phi$ hence $\text{rank}(K) \le \text{rank}(\Phi) \le n \wedge h$. Now if we set $n > h$ we arrive at a contradiction.

We are left to prove the claim that $\text{rank}(K) = n$. We use a tensorization argument. Recall that the tensor product $\mathcal{A} = \mathbf{u} \otimes \mathbf{v} \otimes \cdots \otimes \mathbf{w}$ is a multi-dimensional array such that $\mathcal{A}_{i,j,\cdots,l} = u_i v_j \cdots w_l$. We use the short-hand notation $\mathbf{x}^{\otimes k} := \underbrace{\mathbf{x} \otimes \cdots \otimes \mathbf{x}}_{k \text{ times}}$. For example, $\mathbf{x}^{\otimes 1} = \mathbf{x}$ and $\mathbf{x}^{\otimes 2} \simeq \mathbf{x}\mathbf{x}^\top$. The following claims are easily proved:

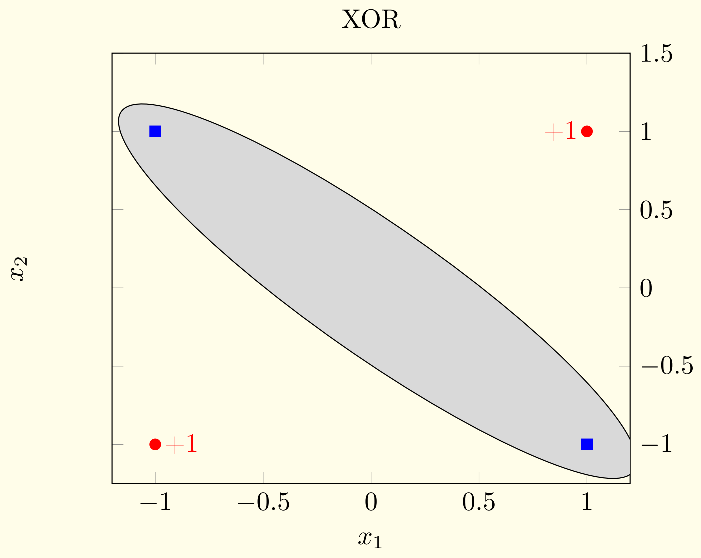
- Let $\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n \in \mathbb{R}^p$ be linearly independent vectors. Then, for any $k \in \mathbb{N}$, $\mathbf{x}^{\otimes k}$, $\mathbf{y}^{\otimes k}$, ..., $\mathbf{z}^{\otimes k}$ are linearly independent vectors in $(\mathbb{R}^p)^{\otimes k}$.
- Let $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^p$ be linearly independent, and $\mathbf{x}_{n+1} \notin \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$. Then there exists some $k \in \{1, \ldots, n+1\}$ such that $\mathbf{x}_1^{\otimes k}, \ldots, \mathbf{x}_n^{\otimes k}, \mathbf{x}_{n+1}^{\otimes k}$ are linearly independent.
  - Suppose not, then we know

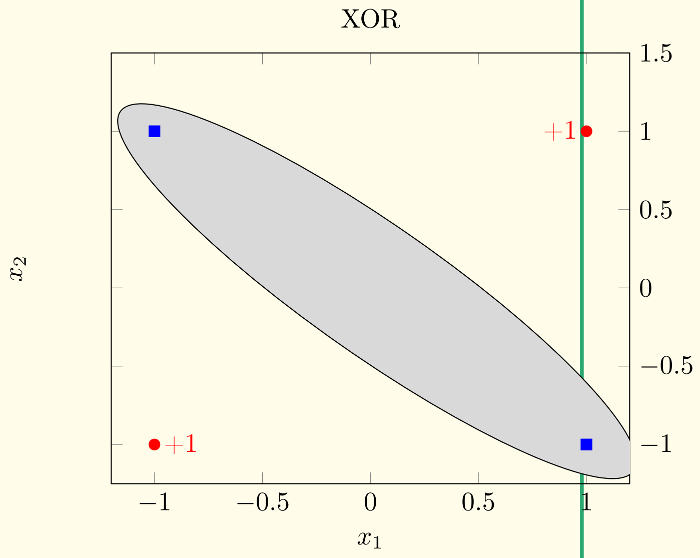
$$\forall k = 1, \ldots, n+1, \quad \mathbf{x}_{n+1}^{\otimes k} = \sum_{i=1}^n \alpha_i^k \mathbf{x}_i^{\otimes k} = \left(\sum_{i=1}^n \alpha_i^1 \mathbf{x}_i\right)^{\otimes k} = \sum_{i_1=1}^n \cdots \sum_{i_k=1}^n \alpha_{i_1}^1 \cdots \alpha_{i_k}^1 \mathbf{x}_{i_1} \otimes \cdots \otimes \mathbf{x}_{i_k}.$$

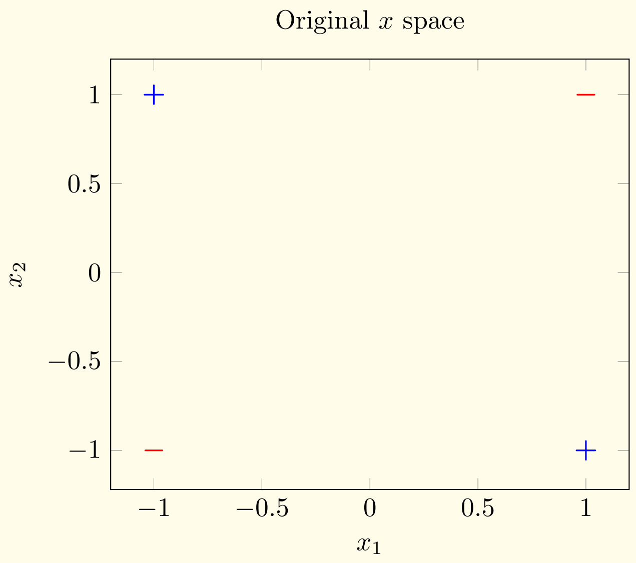
  Since $\{\mathbf{x}_i\}$ are linearly independent, so are $\{\mathbf{x}_{i_1} \otimes \cdots \otimes \mathbf{x}_{i_k}\}$. Thus,

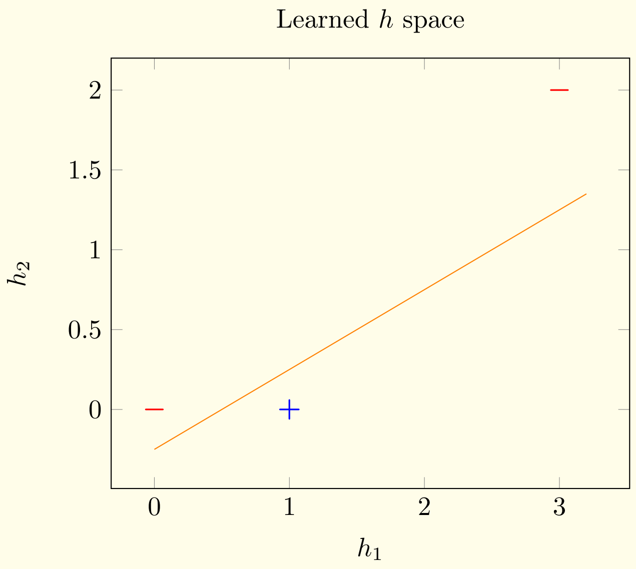
$$|\{i_1, \ldots, i_k\}| > 1 \implies \prod_{j=1}^k \alpha_{i_j}^1 = 0.$$

Example 6.4: XOR revisited









7 Automatic Differentiation (AutoDiff)

Goal

Forward and reverse mode auto-differentiation.

Alert 7.1: Convention

Gray boxes are not required hence can be omitted for unenthusiastic readers.

This note is likely to be updated again soon.

Definition 7.2: Function Superposition and Computational Graph (Bauer 1974)

Let $\mathcal{BF}$ be a class of basic functions. A (vector-valued) function $g : \mathsf{X} \subseteq \mathbb{R}^d \to \mathbb{R}^m$ is a superposition of the basic class $\mathcal{BF}$ if the following is satisfied:

- There exist some DAG $\mathscr{G} = (\mathscr{V}, \mathscr{E})$ where using topological sorting we arrange the nodes as follows:

$$\underbrace{v_1, \ldots, v_d}_{\text{input}}, \quad \underbrace{v_{d+1}, \ldots, v_{d+k}}_{\text{intermediate variables}}, \quad \underbrace{v_{d+k+1}, \ldots, v_{d+k+m}}_{\text{output}}, \qquad \text{and } (v_i, v_j) \in \mathscr{E} \implies i < j.$$

  Here we implicitly assume the outputs of the function g do not depend on each other. If they do, we need only specify the indices of the output nodes accordingly (i.e. they may not all appear in the end).

- For each node v_i, let $\mathscr{I}_i := \{u \in \mathscr{V} : (u, v_i) \in \mathscr{E}\}$ and $\mathscr{O}_i := \{u \in \mathscr{V} : (v_i, u) \in \mathscr{E}\}$ denote the (immediate) predecessors and successors of v_i, respectively. Clearly, $\mathscr{I}_i = \emptyset$ if $i \le d$ (i.e. input nodes) and $\mathscr{O}_i = \emptyset$ if $i > d + k$ (i.e. output nodes).

- The nodes are computed as follows: sequentially for $i = 1, \ldots, d + k + m$,

$$v_i = \begin{cases} x_i, & i \le d \\ f_i(\mathscr{I}_i), & i > d \end{cases}, \quad \text{where} \quad f_i \in \mathcal{BF}. \tag{7.1}$$

Our definition of superposition closely resembles the computational graph of Bauer (1974), who attributed the idea to Kantorovich (1957).

Bauer, F. L. (1974). "Computational Graphs and Rounding Error". *SIAM Journal on Numerical Analysis*, vol. 11, no. 1, pp. 87–96.

Kantorovich, L. V. (1957). "On a system of mathematical symbols, convenient for electronic computer operations". *Soviet Mathematics Doklady*, vol. 113, no. 4, pp. 738–741.

Exercise 7.3: Neural Networks as Function Superposition

Let $\mathcal{BF} = \{+, \times, \sigma, \text{constant}\}$. Prove that any multi-layer NN is a superposition of the basic class $\mathcal{BF}$.

Is exp a superposition of the basic class above?

Theorem 7.4: Automatic Differentiation (e.g. Kim et al. 1984)

Let $\mathcal{BF}$ be a basic class of differentiable functions that includes $+, \times$, and all constants. Denote $T(f)$ as the complexity of computing the function f and $T(f, \nabla f)$ the complexity with additional computation of the gradient. Let $\mathscr{I}_f$ and $\mathscr{O}_f$ be the input and output arguments and assume there exists some constant

$C = C(\mathcal{BF}) > 0$ so that

$$\forall f \in \mathcal{BF}, \quad T(f, \nabla f) + |\mathscr{I}_f||\mathscr{O}_f|[T(+) + T(\times) + T(\text{constant})] \le C \cdot T(f).$$

Then, for any superposition $g : \mathbb{R}^d \to \mathbb{R}^m$ of the basic class $\mathcal{BF}$, we have

$$T(g, \nabla g) \le C\gamma(m \wedge d) \cdot T(g),$$

where γ is the maximal output dimension of basic functions used to superpose g.

Proof. Applying the chain rule to the recursive formula (7.1) it is clear that any superposition g is differentiable too. We split the proof into two parts: a forward mode and a backward mode.

Forward mode: Let us define the block matrix $V = [V_1, \ldots, V_d, V_{d+1}, \ldots, V_{d+k}, V_{d+k+1}, \ldots, V_{d+k+m}] \in \mathbb{R}^{d \times \sum_i d_i}$, where each column block V_i corresponds to the gradient $\nabla v_i = \frac{\partial v_i}{\partial \mathbf{x}} \in \mathbb{R}^{d \times d_i}$, where d_i is the output dimension of node v_i (typically 1). By definition of the input nodes we have

$$V_i = \mathbf{e}_i, \quad i = 1, \ldots, d,$$

where $\mathbf{e}_i$ is the standard basis vector in $\mathbb{R}^d$. Using the recursive formula (7.1) and chain rule we have

$$V_i = \sum_{j \in \mathscr{I}_i} V_j \cdot \nabla_j f_i, \quad \text{where} \quad \nabla_j f_i = \frac{\partial f_i}{\partial v_j} \in \mathbb{R}^{d_j \times d_i}.$$

In essence, by differentiating at each node, we obtained a square and sparse system of linear equations, where $\nabla_j f_i$ are known coefficients and V_i are unknown variables. Solving the linear system yields $V_{d+k+1}, \ldots, V_{d+k+m}$, the desired gradient of g. Thanks to the topological ordering, we can simply solve V_i one by one. Let $\gamma = \max_i d_i$ be the maximum output dimension of any node. We bound the complexity of the forward mode as follows:

$$\begin{aligned} T(g, \nabla g) &\le \sum_{i \in \mathscr{V}} T(f_i, \nabla f_i) + \sum_{j \in \mathscr{I}_i} d d_i d_j [T(+) + T(\times) + T(\text{constant})] \\ &\le d\gamma \sum_{i \in \mathscr{V}} T(f_i, \nabla f_i) + |\mathscr{I}_{f_i}||\mathscr{O}_{f_i}|[T(+) + T(\times) + T(\text{constant})] \le d\gamma C T(g). \end{aligned}$$

Reverse mode: Let us rename the outputs $y_i = v_{d+k+i}$ for $i = 1, \ldots, m$. Similarly we define the block matrix $V = [V_1; \ldots; V_d; V_{d+1}; \ldots; V_{d+k}; V_{d+k+1}; \ldots; V_{d+k+m}] \in \mathbb{R}^{\sum_i d_i \times m}$, where each row block V_i corresponds to the transpose of the gradient $\nabla v_i = \frac{\partial \mathbf{y}}{\partial v_i} \in \mathbb{R}^{m \times d_i}$, where d_i is the output dimension of node v_i (typically 1). By definition of the output nodes we have

$$V_{d+k+i} = \mathbf{e}_i, \quad i = 1, \ldots, m, \quad \mathbf{e}_i \in \mathbb{R}^{1 \times m}.$$

Using the recursive formula (7.1) and chain rule we have

$$V_i = \sum_{j \in \mathscr{O}_i} \nabla_i f_j \cdot V_j, \quad \text{where} \quad \nabla_i f_j = \frac{\partial f_j}{\partial v_i} \in \mathbb{R}^{d_i \times d_j}.$$

Again, by differentiating at each node we obtained a square and sparse system of linear equations, where $\nabla_i f_j$ are known coefficients and V_i are unknown variables. Solving the linear system yields $V_1, \ldots, V_d$, the desired gradient of g. Thanks to the topological ordering, we can simply solve V_i one by one backwards, after a forward pass to get the function values at each node. Similar as the forward mode, we can bound the complexity as $m\gamma CT(g)$. □

Thus, surprisingly, for real-valued superpositions ($m = \gamma = 1$), computing the gradient, which is a $d \times 1$ vector, costs at most constant times that of the function value (which is a scalar), if we operate in the reverse mode! The common misconception is that the gradient has size $d \times 1$ hence if we compute one component at a time we end up d times slower. This is wrong, because we can recycle computations. Note also that even reading the input already costs $O(d)$. However, this time complexity gain, as compared to that of the forward mode, is achieved through a space complexity tradeoff: in reverse mode we need a forward pass first to collect and store all function values at each node, whereas in the forward mode these function values can be computed on the fly.

Kim, K. V., Yuri E. Nesterov, and B. V. Cherkasskii (1984). "An estimate of the effort in computing the gradient". *Soviet Mathematics Doklady*, vol. 29, no. 2, pp. 384–387.

Algorithm 7.5: Automatic Differentiation (AD) Pesudocode

We summarize the forward and reverse algorithms below. Note that to compute the gradient-vector multiplication $\nabla g \cdot \mathbf{w}$ for some compatible vector $\mathbf{w}$, we can use the forward mode and initialize V_i with $\mathbf{w}$. Similarly, to compute $\mathbf{w} \cdot \nabla g$, we can use the reverse mode with proper initialization to V_{d+k+i}.

```
Algorithm: Forward Automatic Differentiation for Superposition.
  Input: $\mathbf{x} \in \mathbb{R}^d$, basic function class $\mathcal{BF}$, computational graph $\mathscr{G}$
  Output: gradient $[V_{d+k+1}, \ldots, V_{d+k+m}] \in \mathbb{R}^{d\times m}$
1 for $i = 1, \ldots, d$ do                // forward: initialize function values and derivatives
2     $v_i \leftarrow x_i$
3     $V_i \leftarrow \mathbf{e}_i \in \mathbb{R}^{d\times 1}$
4 for $i = d+1, \ldots, d+k+m$ do        // forward: accumulate function values and derivatives
5     compute $v_i \leftarrow f_i(\mathscr{I}_i)$
6     for $j \in \mathscr{I}_i$ do
7         compute partial derivatives $\nabla_j f_i(\mathscr{I}_i)$
8     $V_i \leftarrow \sum_{j\in\mathscr{I}_i} V_j \cdot \nabla_j f_i$
```

```
Algorithm: Reverse Automatic Differentiation for Superposition.
  Input: $\mathbf{x} \in \mathbb{R}^d$, basic function class $\mathcal{BF}$, computational graph $\mathscr{G}$
  Output: gradient $[V_1; \ldots; V_d] \in \mathbb{R}^{d\times m}$
1 for $i = 1, \ldots, d$ do                // backward: initialize function values and derivatives
2     $v_i \leftarrow x_i$
3     $V_{d+k+i} \leftarrow \mathbf{e}_i \in \mathbb{R}^{1\times m}$
4 for $i = d+1, \ldots, d+k+m$ do                       // forward: accumulate function values
5     compute $v_i \leftarrow f_i(\mathscr{I}_i)$
6 for $i = d+k, \ldots, 1$ do             // backward: accumulate function values and derivatives
7     $V_i \leftarrow \sum_{j\in\mathscr{O}_i} \nabla_i f_j \cdot V_j$
```

We remark that, as suggested by Wolfe (1982), one effective way to test AD (or manually programmed derivatives) and locate potential errors is through the classic finite difference approximation.

Wolfe, Philip (1982). "Checking the Calculation of Gradients". *ACM Transactions on Mathematical Software*, vol. 8, no. 4, pp. 337–343.

Exercise 7.6: Matrix multiplication

To understand the difference between forward-mode and backward-mode differentiation, let us consider the simple matrix multiplication problem: Let $A_\ell \in \mathbb{R}^{d_\ell \times d_{\ell+1}}, \ell = 1, \ldots, L$, where $d_1 = d$ and $d_{L+1} = m$. We are interested in computing

$$A = \prod_{\ell=1}^{L} A_\ell.$$

- What is the complexity if we multiply from left to right (i.e. $\ell = 1, 2, \ldots, L$)?
- What is the complexity if we multiply from right to left (i.e. $\ell = L, L-1, \ldots, 1$)?
- What is the optimal way to compute the product?

Remark 7.7: Further insights on AD

If we associate an edge weight $w_{ij} = \frac{\partial v_j}{\partial v_i}$ to $(i, j) \in \mathscr{E}$, then the desired gradient

$$\frac{\partial g_i}{\partial x_j} = \sum_{\text{path } P: v_j \to v_i} \prod_{e \in P} w_e. \tag{7.2}$$

However, we cannot compute the above naively, as the number of paths in a DAG can grow exponentially quickly with the depth. The forward and reverse modes in the proof of Theorem 7.4 correspond to two dynamic programming solutions. (Incidentally, this is exactly how one computes the graph kernel too.)

Naumann (2008) showed that finding the optimal way to compute (7.2) is NP-hard.

Naumann, Uwe (2008). "Optimal Jacobian accumulation is NP-complete". *Mathematical Programming*, vol. 112, no. 2, pp. 427–441.

Remark 7.8: Tightness of dimension dependence in AD (e.g. Griewank 2012)

The dimensional dependence $m \wedge d$ cannot be reduce in general. Indeed, consider the simple function $\mathbf{f}(\mathbf{x}) = \sin(\mathbf{w}^\top \mathbf{x})\mathbf{b}$, where $\mathbf{x} \in \mathbb{R}^d$ and $\mathbf{b} \in \mathbb{R}^m$. Computing $\mathbf{f}$ clearly costs $O(d + m)$ (assuming sin can be evaluated in $O(1)$) while even outputting the gradient costs $O(dm)$.

Griewank, Andreas (2012). "Who Invented the Reverse Mode of Differentiation?" *Documenta Mathematica*, vol. Extra Volume ISMP, pp. 389–400.

Exercise 7.9: Backpropogation (e.g. Rumelhart et al. 1986)

Apply Theorem 7.4 to multi-layer NNs and recover the celebrated backpropogation algorithm. Distinguish two cases:

- Fix the network weights $W_1, \ldots, W_L$ and compute the derivative w.r.t. the input $\mathbf{x}$ of the network. This is useful for constructing adversarial examples.
- Fix the input $\mathbf{x}$ of the network and compute the derivative w.r.t. the network weights $W_1, \ldots, W_L$. This is useful for training the network.

Suppose we know how to compute the derivatives of $f(x, y)$. Explain how to compute the derivative of $f(x, x)$?

- Generalize from above to derive the backpropogation rule for convolutional neural nets (CNN).
- Generalize from above to derive the backpropogation rule for recurrent neural nets (RNN).

Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams (1986). "Learning representations by back-propagating errors". *Nature*, vol. 323, pp. 533–536.

Remark 7.10: Fast computation of other derivatives (Kim et al. 1984)

Kim et al. (1984) pointed out an important observation, namely that the proof of Theorem 7.4 only uses the chain-rule property of differentiation:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial y} \cdot \frac{\partial x}{\partial y}.$$

In other words, we could replace differentiation with any other operation that respects the chain rule and obtain the same efficient procedure for computation. For instance, the relative differential in numerical analysis or the directional derivative can both be efficiently computed in the same way. Similarly, one can compute the Hessian-vector multiplication efficiently as it also respects the chain rule (Møoller 1993; Pearlmutter 1994).

Kim, K. V., Yuri E. Nesterov, V. A. Skokov, and B. V. Cherkasskii (1984). "An efficient algorithm for computing derivatives and extremal problems". *Ekonomika i matematicheskie metody*, vol. 20, no. 2, pp. 309–318.

Møoller, M. (1993). *Exact Calculation of the Product of the Hessian Matrix of Feed-Forward Network Error Functions and a Vector in $O(N)$ Time*. Tech. rep. DAIMI Report Series, 22(432).

Pearlmutter, Barak A. (1994). "Fast Exact Multiplication by the Hessian". *Neural Computation*, vol. 6, no. 1, pp. 147–160.

8 Deep Neural Networks

Goal

Define and understand the notion of fairness in machine learning algorithms.

Alert 8.1: Convention

Gray boxes are not required hence can be omitted for unenthusiastic readers.

This note is likely to be updated again soon.

9 Convolutional Neural Networks

Goal

Introducing the basics of CNN and the popular architectures.

Alert 9.1: Convention

Gray boxes are not required hence can be omitted for unenthusiastic readers.

This note is likely to be updated again soon.

10 Recurrent Neural Networks

Goal

Introducing the basics of RNN. LSTM. GRU. PixelRNN and related.

Alert 10.1: Convention

Gray boxes are not required hence can be omitted for unenthusiastic readers.

This note is likely to be updated again soon.

11 Graph Neural Networks

Goal

Introducing the basics of GNN and the popular variants.

Alert 11.1: Convention

Gray boxes are not required hence can be omitted for unenthusiastic readers.

This note is likely to be updated again soon.

Nice surveys on this topic include Bronstein et al. (2017) and Wu et al. (2020).

Bronstein, M. M., J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst (2017). "Geometric Deep Learning: Going beyond Euclidean data". *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42.

Wu, Z., S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu (2020). "A Comprehensive Survey on Graph Neural Networks". *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–21.

Definition 11.2: Graph learning

Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathfrak{l})$ with nodes $\mathcal{V}$, edges $\mathcal{E}$, node feature/attribute/label $\mathfrak{l}_v \in \mathbb{R}^d$ for each node $v \in \mathcal{V}$, and edge feature/attribute/label $\mathfrak{l}_e \in \mathbb{R}^p$ for each edge $e \in \mathcal{E}$. The graph may be directed or undirected, where the direction of the edge can be easily encoded in the edge feature. We use $\mathcal{N}_v = \mathcal{N}(v) \subseteq \mathcal{V}$ to denote the neighboring nodes of v and $\mathcal{M}_v = \mathcal{M}(v) \subseteq \mathcal{E}$ for the edges that have node v as a vertex. For positional graphs, we also have an injective function $\mathfrak{p}_v : \mathcal{N}_v \to \{1, 2, \ldots, |\mathcal{V}|\}$ that encodes the relative position of each neighbor of a node v. For instance, on a 2-D image, $\{1, 2, 3, 4\}$ may represent the west, north, east, and south neighbor, respectively.

Alert 11.3: All for one, and one for all

Let $(\mathcal{G}_i, \mathbf{y}_i), i = 1, \ldots, n$ be a given supervised set of graphs and labels. Our goal is to learn a predictive function $\hat{\mathbf{y}}$ that maps a new test graph $\mathcal{G}$ to its corresponding label: $\hat{\mathbf{y}}(\mathcal{G}) \approx \mathbf{y}$. The labels could be at the node, edge or graph level. Do not confuse the label $\mathbf{y}$ with the feature $\mathfrak{l}$, since some authors also refer to the latter as "labeling."

Interestingly, we can piece all graphs into one large, disconnected graph, greatly simplifying our notation and without compromising generality. Note that this is more than just a reduction trick: in some cases it is actually the natural thing to do, such as in web-scale applications where the entire internet is just one giant graph. We follow this trick throughout.

Example 11.4: Some applications of graph learning

We mention some example applications of graph learning:

- Each node may represent an atom in some chemical compound while the edges model the (strength of) chemical bonds linking the atoms. We may be interested in predicting how a certain disease reacts to the chemical compound.
- All image analyses fall into graph learning with each pixel playing a node of the underlying (regular) grid and the pixel value being the node feature.
- Social network, where we may be interested in classifying the nodes or imputing missing links. For instance, each webpage is a node and hyperlinks act as edges.

Definition 11.5: Graph neural network (GNN) (Scarselli et al. 2009)

GNNs, as defined here, can be regarded as a natural extension of recurrent networks, from a chain graph to a general graph. Indeed, we define the following recursion: for all $v \in \mathcal{V}$,

$$\mathbf{h}_v \leftarrow \mathbf{f}(\mathbf{h}_v, \mathbf{h}_{\mathcal{N}_v}, \mathfrak{l}_v, \mathfrak{l}_{\mathcal{N}_v}, \mathfrak{l}_{\mathcal{M}_v}; \mathbf{w})$$
$$\mathbf{o}_v = \mathbf{g}(\mathbf{h}_v, \mathfrak{l}_v; \mathbf{w}),$$

where $\mathbf{h}_v$ is the hidden state at node v and $\mathbf{o}_v$ is its output. The two (local) update functions $\mathbf{f}, \mathbf{g}$ are parameterized by $\mathbf{w}$, which is shared among all nodes. We remark that in general it is up to us to define the neighborhoods $\mathcal{N}$ and $\mathcal{M}$, and $\mathbf{f}, \mathbf{g}$ may have slightly different forms (such as involving other inputs).

Collect all local updates into one abstract formula:

$$\mathbf{x} := \begin{bmatrix} \mathbf{h} \\ \mathbf{o} \end{bmatrix} \leftarrow \mathsf{F}(\mathbf{x}, \mathfrak{l}; \mathbf{w}). \tag{11.1}$$

Note that the input node/edge features $\mathfrak{l}$ are fixed. Thus, for a fixed weight $\mathbf{w}$, the above update defines the (enhanced) state $\mathbf{x}$ as a fixed point of the map $\mathsf{F}_{\mathfrak{l},\mathbf{w}} : \mathbf{x} \mapsto \mathsf{F}(\mathbf{x}, \mathfrak{l}; \mathbf{w})$.

To compute the state $\mathbf{x}$ with a fixed weight $\mathbf{w}$, we perform the (obvious) iteration:

$$\mathbf{x}_{t+1} = \mathsf{F}(\mathbf{x}_t, \mathfrak{l}; \mathbf{w}), \quad \mathbf{x}_0 \text{ initialized.} \tag{11.2}$$

According to Banach's fixed point theorem, (for any initialization $\mathbf{x}_0$) the above iteration converges geometrically to the unique fixed point of $\mathsf{F}_{\mathfrak{l},\mathbf{w}}$, provided that the latter is a contraction (or more generally a firm nonexpansion). For later reference, we abstract (the unique) solution of the nonlinear equation (11.1) as:

$$\mathbf{o} = \hat{\mathbf{y}}(\mathfrak{l}; \mathbf{w}),$$

where we have discarded the state $\mathbf{h}$ and only retained the output $\mathbf{o}$.

Scarselli, F., M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini (2009). "The Graph Neural Network Model". *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80.

Alert 11.6: Recursive neural network

When the underlying graph is a DAG (and the update function of a node only depends on its descendants), we may arrange the computation in (11.2) according to some topological ordering so that it stops after one (sequential) pass of all nodes. When the graph is a chain, we recover the familiar recurrent neural network.

Example 11.7: Local update function

We mention two examples of local update function:

- For positional graphs, we arrange the neighbors in $\mathbf{h}_{\mathcal{N}_v}, \mathfrak{l}_{\mathcal{N}_v}, \mathfrak{l}_{\mathcal{M}_v}$ according to their relative positions decided by $\mathfrak{p}_v$ (say in increasing order). For non-existent neighbors, we may simply pad with null values.
- For non-positional graphs, the following permutation-invariant local update is convenient:

$$\mathbf{h}_v \leftarrow \frac{1}{|\mathcal{N}_v|} \sum_{u \in \mathcal{N}_v} \mathbf{f}(\mathbf{h}_v, \mathbf{h}_u, \mathfrak{l}_v, \mathfrak{l}_u, \mathfrak{l}_{(v,u)}).$$

  More generally, we may replace the above average with any permutation-invariant function (e.g. averaged ℓ_p norm), see Xu et al. (2019) for some discussion on possible limitations of this choice.

Xu, Keyulu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka (2019). "How Powerful are Graph Neural Networks?" In: *International Conference on Learning Representations*.

Algorithm 11.8: Learning GNN

To learn the weights $\mathbf{w}$ of a GNN, we choose a loss function ℓ, and apply (stochastic) gradient descent to solve

$$\min_{\mathbf{w}} \quad \ell(\hat{\mathbf{y}}(\mathfrak{l};\mathbf{w}),\mathbf{y}),$$

where recall that $\hat{\mathbf{y}}(\mathfrak{l};\mathbf{w})$ is (the unique) solution of the nonlinear equation (11.1) and is practically computed by the iteration (11.2) (similar to unrolling in RNN). If $\mathsf{F}(\mathbf{x},\mathfrak{l};\mathbf{w})$ is differentiable in $\mathbf{w}$ and contracting in $\mathbf{x}$, then a simple application of the implicit function theorem reveals that the solution $\hat{\mathbf{y}}(\mathfrak{l};\mathbf{w})$ is also differentiable in $\mathbf{w}$. Thus, we may apply the recurrent back-propagation algorithm. If memory is not an issue, we can also apply back-propagation through time (BPTT) by replacing $\hat{\mathbf{y}}$ with $\mathbf{o}_t$ after a fixed number of unrolling steps in (11.1).

Example 11.9: Parameterizing local update function

- Affine: Let $\mathsf{F}(\mathbf{x},\mathfrak{l};\mathbf{w}) = A(\mathfrak{l};\mathbf{w})\mathbf{x} + \mathbf{b}(\mathfrak{l};\mathbf{w})$, where the matrix A and bias vector $\mathbf{b}$ are outputs of some neural net with input $\mathfrak{l}$ and weights $\mathbf{w}$. By properly scaling A, it is easy to make F a contraction.
- More generally, we may parameterize F by a (highly) nonlinear deep network. However, care must be taken (e.g. through regularization) so that F is (close to) a contraction at the learned weights.
- We remark that in theory any parameterization of F can be used; it does not have to be a neural network.

Example 11.10: PageRank belongs to GNN

Define the normalized adjacency matrix

$$\bar{A}_{uv} = \begin{cases} \frac{1}{|\mathcal{N}_u|}, & \text{if } (u,v)\in\mathcal{E} \\ 0, & \text{otherwise} \end{cases},$$

which represents the probability of visiting a neighboring node v once we are at node u. Consider the GNN with linear state update function:

$$\mathbf{x} \leftarrow \alpha\mathbf{x}_0 + (1-\alpha)\bar{A}^\top\mathbf{x},$$

where the parameter $\alpha \in [0,1)$ models the probability of "telescoping" and $\mathbf{x}_0 \in \Delta$. In other words, the state of node v is an aggregation of the states of its neighbors:

$$x_v = \alpha x_{v,0} + (1-\alpha)\sum_{(u,v)\in\mathcal{E}} \tfrac{1}{|\mathcal{N}_u|} x_u.$$

For any $\alpha \in (0,1)$, the above iterate converges to a unique fixed point known as the PageRank.

Definition 11.11: Spatial convolutional networks on graphs (Bruna et al. 2014)

Given a (weighted) graph $\mathcal{G}^0 = (\mathcal{V}^0, A^0)$, where A^0 is the adjacency matrix, we define a sequence of coarsenings $\mathcal{G}^l = (\mathcal{V}^l, A^l), l = 1,\ldots,L$, where recursively each node $V \in \mathcal{V}^{l+1}$ is a subset (e.g. neighborhood) of nodes in $\mathcal{V}^l$, i.e.

$$\mathcal{V}^{l+1} \subseteq 2^{\mathcal{V}^l}, \quad \text{and for all } U, V \in \mathcal{V}^{l+1},\ A^{l+1}_{UV} = \sum_{u \in U \subseteq \mathcal{V}^l} \sum_{v \in V \subseteq \mathcal{V}^l} A^l_{uv}.$$

Typically, the nodes in $\mathcal{V}^{l+1}$ form a partition of the nodes in $\mathcal{V}^l$, using say some graph partitioning algorithm. For instance we may cluster a node u with all "nearby" and available nodes v with $A_{uv} \le \epsilon$, hence forming an ϵ-cover.

Let $\mathbf{x}^l = [\mathbf{x}^l_1; \ldots; \mathbf{x}^l_{d_l}] \in \mathbb{R}^{|\mathcal{V}^l|d_l}$ be a d_l-channel signal on the nodes of graph $\mathcal{G}^l$. We define a layer of spatial convolution as follows:

$$\mathbf{x}^{l+1}_r = \mathscr{P}\big(\sigma(W^l_r \mathbf{x}^l)\big), \quad W^l_r \in \mathbb{R}^{|\mathcal{V}^l| \times |\mathcal{V}^l| d_l}, \quad r = 1, \ldots, d_{l+1},$$

where each W^l_r is a *spatially compact* filter (with nonzero entries only when A^l_{uv} larger than some threshold), $\sigma : \mathbb{R} \to \mathbb{R}$ some (nonlinear) component-wise activation function, and $\mathscr{P}$ a pooling operator that pools the values in each neighborhood (corresponding to nodes in $\mathcal{V}^{l+1}$). The total number of parameters in the filter $\mathcal{W}^l$ is $O(|\mathcal{E}^l| d_l d_{l+1})$. Since nodes in a general graph (as opposed to regular ones such as grids) may have different neighborhoods, it is not possible to share the filter weights at different nodes (i.e. the rows in W^l_r have to be different).

Bruna, Joan, Wojciech Zaremba, Arthur Szlam, and Yann LeCun (2014). "Spectral Networks and Locally Connected Networks on Graphs". In: *International Conference on Learning Representations*.

Example 11.12: Spatial CNN (Niepert et al. 2016)

The main difficulty in extending spatial convolution to general graphs is the lack of correspondence of the nodes. Niepert et al. (2016) proposed to first label the nodes so that they are somewhat in correspondence. Consider $\mathfrak{l} : \mathcal{V} \to \mathsf{L}$ that sends a node $v \in \mathcal{V}$ to a color $\mathfrak{l}_v$ in some totally ordered set L. For instance, $\mathfrak{l}$ could simply be the node degree or computed by the WL Algorithm 11.23 below. We proceed similarly as in CNN:

- The color $\mathfrak{l}$ induces an ordering of the nodes, allowing us to select a fixed number n of nodes, starting from the "smallest" and incrementing with stride s. We pad (disconnected) trivial nodes if run out of choices.
- For each chosen node v above, we incrementally select its neighbors $\mathcal{N}_v := \bigcup_d \{u : \text{dist}(u, v) \le d\}$ using breadth first search (BFS), until exceeding the receptive field size or running out of choice.
- We recompute colors on $\mathcal{N}_v$ with the constraint $\text{dist}(u, v) < \text{dist}(w, v) \implies \mathfrak{l}_u < \mathfrak{l}_w$. Depending on the size of $\mathcal{N}_v$, we either select a fixed number m of (top) neighbors and recompute their colors, or pad (disconnected) trivial nodes to make the fixed number m. Lastly, we perform canoniocalization using NAUTY (McKay and Piperno 2014) while respecting the node colors.
- Finally, we collect the results into tensors with size $n \times m \times d$ for d-dim node features and $n \times m \times m \times p$ for p-dim edge features, which can be reshaped to $nm \times d$ and $nm^2 \times p$. We apply 1-d convolution with stride and receptive field size m to the first and m^2 to the second tensor.

For grid graphs, if we use the WL Algorithm 11.23 to color the nodes, then it is easy to see that the above procedure recovers the usual CNN.

Niepert, Mathias, Mohamed Ahmed, and Konstantin Kutzkov (2016). "Learning Convolutional Neural Networks for Graphs". In: *Proceedings of The 33rd International Conference on Machine Learning*, pp. 2014–2023.

McKay, Brendan D. and Adolfo Piperno (2014). "Practical graph isomorphism, II". *Journal of Symbolic Computation*, vol. 60, pp. 94–112.

Definition 11.13: Graph Laplacian

Let A be the usual adjacency matrix of an (undirected) graph and D the diagonal matrix of degrees:

$$A_{uv} = \begin{cases} 1, & \text{if } (u,v) \in \mathcal{E} \\ 0, & \text{otherwise} \end{cases}, \quad D_{uv} = \begin{cases} \sum_v A_{uv}, & \text{if } u = v \\ 0, & \text{otherwise} \end{cases}.$$

More generally, we may consider a weighted graph with (nonnegative, real-valued and symmetric) weights $A_{uv} = w_{uv}$. We define the graph Laplacian and its normalized version:

$$L = D - A, \quad \bar{L} = I - D^{-1/2}AD^{-1/2} = D^{-1/2}LD^{-1/2}.$$

Among many other nice properties, the graph Laplacian is useful because of its connection to quadratic potentials. Too see this, let $\mathbf{x}_v \in \mathbb{R}^d$ be a feature vector at each node v and we verify that

$$\frac{1}{2}\sum_{u,v} A_{uv}\|\mathbf{x}_u - \mathbf{x}_v\|_2^2 = \frac{1}{2}\sum_{u,v} A_{uv}[\|\mathbf{x}_u\|_2^2 + \|\mathbf{x}_v\|_2^2 - 2\langle \mathbf{x}_u, \mathbf{x}_v\rangle] = \sum_u d_u\|\mathbf{x}_u\|_2^2 - \sum_{u,v} A_{uv}\langle \mathbf{x}_u, \mathbf{x}_v\rangle$$

$$= \operatorname{tr}(X(D-W)X^\top) = \operatorname{tr}(XLX^\top) = \sum_{j=1}^d X_{j:}LX_{j:}^\top, \quad X = [\ldots, \mathbf{x}_v, \ldots] \in \mathbb{R}^{d\times|\mathcal{V}|}.$$

Taking $d = 1$ we see that the Laplacian L is symmetric and positive semidefinite. Similarly,

$$\operatorname{tr}(X\bar{L}X^\top) = \operatorname{tr}((XD^{-1/2})L(D^{-1/2}X^\top)) = \frac{1}{2}\sum_{u,v} A_{uv}\|\tfrac{\mathbf{x}_u}{\sqrt{d_u}} - \tfrac{\mathbf{x}_v}{\sqrt{d_v}}\|_2^2.$$

Of course, the normalized graph Laplacian is also symmetric and positive semidefinite.

Exercise 11.14: Laplacian and Connectedness

Prove that the dimension of the null space of the Laplacian is exactly the number of connected components in the (weighted) graph.

Moreover, $L\mathbf{1} = 0$, so the Laplacian always has 0 as an eigenvalue and $\mathbf{1}$ as the corresponding eigenvector.

Remark 11.15: Graph Laplacian is everywhere

The graph Laplacian played significant roles in the early days of segmentation, dimensionality reduction and semi-supervised learning, see Shi and Malik (e.g. 2000), Dhillon et al. (2007), Zhu et al. (2003), Zhou et al. (2004), Coifman et al. (2005), Belkin et al. (2006), Belkin and Niyogi (2008), Hammond et al. (2011), and Shuman et al. (2013). It allows us to propagate information from one node to another through traversing the edges and to enforce global consistency through local ones. Typical ways to construct graph from sampled data include thresholding pairwise distances or comparing node features.

Shi, Jianbo and J. Malik (2000). "Normalized cuts and image segmentation". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, no. 8, pp. 888–905.

Dhillon, I. S., Y. Guan, and B. Kulis (2007). "Weighted Graph Cuts without Eigenvectors A Multilevel Approach". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 29, no. 11, pp. 1944–1957.

Zhu, Xiaojin, Zoubin Ghahramani, and John Lafferty (2003). "Semi-Supervised Learning Using Gaussian Fields and Harmonic Functions". In: *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, pp. 912–919.

Zhou, Dengyong, Olivier Bousquet, Thomas N. Lal, Jason Weston, and Bernhard Schölkopf (2004). "Learning with Local and Global Consistency". In: *Advances in Neural Information Processing Systems 16*, pp. 321–328.

Coifman, R. R., S. Lafon, A. B. Lee, M. Maggioni, B. Nadler, F. Warner, and S. W. Zucker (2005). "Geometric diffusions as a tool for harmonic analysis and structure definition of data: Diffusion maps". *Proceedings of the National Academy of Sciences*, vol. 102, no. 21, pp. 7426–7431.

Belkin, Mikhail, Partha Niyogi, and Vikas Sindhwani (2006). "Manifold Regularization: A Geometric Framework for Learning from Labeled and Unlabeled Examples". *Journal of Machine Learning Research*, vol. 7, pp. 2399–2434.
Belkin, Mikhail and Partha Niyogi (2008). "Towards a theoretical foundation for Laplacian-based manifold methods". *Journal of Computer and System Sciences*, vol. 74, no. 8, pp. 1289–1308.
Hammond, David K., Pierre Vandergheynst, and Rémi Gribonval (2011). "Wavelets on graphs via spectral graph theory". *Applied and Computational Harmonic Analysis*, vol. 30, no. 2, pp. 129–150.
Shuman, D. I., S. K. Narang, P. Frossard, A. Ortega, and P. Vandergheynst (2013). "The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains". *IEEE Signal Processing Magazine*, vol. 30, no. 3, pp. 83–98.

Definition 11.16: Spectral convolutional networks on graphs (Bruna et al. 2014)

Bruna et al. (2014) also defined the spectral graph convolution of two graph signals $\mathbf{x} \in \mathbb{R}^{|\mathcal{V}|}$ and $\mathbf{g} \in \mathbb{R}^{|\mathcal{V}|}$ as:

$$\mathbf{x} * \mathbf{g} := U[(U^\top \mathbf{x}) \odot (U^\top \mathbf{g})], \quad \text{where} \quad L = U\Lambda U^\top$$

is the spectral decomposition of the graph Laplacian L and $\odot$ denotes component-wise multiplication. Let $\mathbf{g}$, or equivalently $\mathbf{w} := U^\top \mathbf{g}$, represent a filter. We then define a layer of spectral graph convolution as:

$$\mathbf{x}_r^{l+1} = \sigma\big(U[W_r^l \odot (U^\top X^l)]\mathbf{1}\big), \ r = 1, \ldots, d_{l+1}, \ X^l = [\mathbf{x}_1^l, \ldots, \mathbf{x}_{d_l}^l], \tag{11.3}$$

where d_l is the number of channels for layer l and $\sigma : \mathbb{R} \to \mathbb{R}$ is some component-wise (nonlinear) activation function. The formula (11.3) continues to make sense if we only take say bottom s_l eigenvectors in U (corresponding to the smallest eigenvalues). Thus, the number of filter parameters in $\mathcal{W}^l$ is $O(s_l d_l d_{l+1})$, which we may reduce through interpolating a few "landmarks": $W_r^l = B\boldsymbol{\alpha}_r^l$, where B is a fixed interpolation kernel and the few knots $\boldsymbol{\alpha}_r^l$ are tunable.

When a sequence of coarsenings $\mathcal{G}^l$ is available (like the spatial convolution in Definition 11.11), we can then perform pooling on the signal X^l by pooling the values in each neighborhood (corresponding to nodes in $\mathcal{V}^{l+1}$).

Henaff et al. (2015) also considered learning the graph topology and spectral convolution alternately.

Bruna, Joan, Wojciech Zaremba, Arthur Szlam, and Yann LeCun (2014). "Spectral Networks and Locally Connected Networks on Graphs". In: *International Conference on Learning Representations.*
Henaff, Mikael, Joan Bruna, and Yann LeCun (2015). "Deep Convolutional Networks on Graph-Structured Data".

Definition 11.17: Chebyshev polynomial

Let $\mathfrak{p}_0 \equiv 1$ and $\mathfrak{p}_1(x) = x$. For $k \geq 2$ we define the k-th Chebyshev polynomial recursively:

$$\mathfrak{p}_k(x) = 2x \cdot \mathfrak{p}_{k-1}(x) - \mathfrak{p}_{k-2}(x).$$

It is known that Chebyshev polynomials form an orthogonal basis for $L^2([-1,1], \mathrm{d}x/\sqrt{1-x^2})$.

Example 11.18: Chebyshev Net (Defferrard et al. 2016)

The spectral graph convolution in Definition 11.16 is expensive as we need to eigen-decompose the Laplacian L. However, note that

$$\mathbf{x} * \mathbf{g} := U[(U^\top \mathbf{g}) \odot (U^\top \mathbf{x})] = U[\operatorname{diag}(f(\boldsymbol{\lambda}; \mathbf{w}))(U^\top \mathbf{x})] = [U \operatorname{diag}(f(\boldsymbol{\lambda}; \mathbf{w}))U^\top]\mathbf{x},$$

where we assume $U^\top \mathbf{g} = f(\boldsymbol{\lambda}; \mathbf{w})$ and recall the eigen-decomposition $L = U \operatorname{diag}(\boldsymbol{\lambda})U^\top$. The univariate function $f : \mathbb{R} \to \mathbb{R}$ is parameterized by $\mathbf{w}$ and is applied component-wise to a vector (and component-wise to the eigenvalues of a symmetric matrix). Then, it follows

$$\mathbf{x} * \mathbf{g} = f(L; \mathbf{w})\mathbf{x},$$

and with a polynomial function $f(\lambda; \mathbf{w}) = \sum_{j=0}^{k-1} w_j \lambda^j$ we have $\mathbf{x} * \mathbf{g} = \sum_{j=0}^{k-1} w_j L^j \mathbf{x}$, where the polynomial L^j only depends on nodes within j edges hence localized. Using the Chebyshev polynomial we may then parameterize spectral convolution:

$$\mathbf{x} * \mathbf{g} = \sum_{j=0}^{k-1} w_j \mathfrak{p}_j(\tilde{L})\mathbf{x}, \qquad \text{where} \quad \tilde{L} := 2L/\|L\| - I,$$

whose spectrum lies in $[-1, 1]$. If we define $\mathbf{x}^j = \mathfrak{p}_j(\tilde{L})\mathbf{x}$, then recursively

$$\mathbf{x}^j = 2\tilde{L}\mathbf{x}^{j-1} - \mathbf{x}^{j-2}, \quad \text{with} \quad \mathbf{x}^0 = \mathbf{x}, \ \mathbf{x}^1 = \tilde{L}\mathbf{x}.$$

The above recursion indicates that Chebyshev net is similar to a k-step unrolling of GNN with linear update functions.

Thus, computing the graph convolution $\mathbf{x} * \mathbf{g}$ costs only $O(k|\mathcal{E}|)$. We easily extend to multi-channel signals $X = [\mathbf{x}_1, \ldots, \mathbf{x}_s] \in \mathbb{R}^{|\mathcal{V}| \times s}$ with filters $W_r = [\mathbf{w}_0^r, \ldots, \mathbf{w}_{k-1}^r] \in \mathbb{R}^{s \times k}$:

$$\mathbf{x} * \mathbf{g}_r = \sum_{i=1}^{s} \sum_{j=0}^{k-1} w_{ij}^r \mathfrak{p}_j(\tilde{L})\mathbf{x}_i = \left[\mathfrak{p}_0(\tilde{L}), \cdots, \mathfrak{p}_{k-1}(\tilde{L})\right] \mathtt{vec}(XW_r), \quad r = 1, \ldots, t,$$

where s and t are the number of input and output channels, respectively. Component-wise nonlinear activation is applied afterwards, and pooling can be similarly performed as before if a sequence of coarsenings is available.

Defferrard, Michaël, Xavier Bresson, and Pierre Vandergheynst (2016). "Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering". In: *Advances in Neural Information Processing Systems 29*, pp. 3844–3852.

Definition 11.19: Graph convolutional network (GCN) (Kipf and Welling 2017)

Given a weighted graph $\mathcal{G} = (\mathcal{V}, A)$, a layer of GCN is defined concisely as:

$$X^{l+1} = \sigma\big(\mathring{D}^{-1/2}\mathring{A}\mathring{D}^{-1/2}X^l W^l\big), \quad X^l = [\mathbf{x}_1^l, \ldots, \mathbf{x}_s^l] \in \mathbb{R}^{|\mathcal{V}| \times s}, \quad W^l \in \mathbb{R}^{s \times t}, \tag{11.4}$$

where $\mathring{A} = A + I$ (i.e. adding self-cycle), $\mathring{D}$ is the usual diagonal degree matrix of $\mathring{A}$, and s and t are the number of input and output channels, respectively.

GCN can be motivated by setting $k = 1$ and with weight-sharing $w_{i,0}^r = -w_{i,1}^r = w_i^r$ in Chebyshev net (see Example 11.18):

$$\mathbf{x} * \mathbf{g}_r = \sum_{i=1}^{s} (w_{i,0}^r I + w_{i,1}^r \tilde{L})\mathbf{x}_i = \sum_{i=1}^{s} w_i^r (I - 2L/\|L\| + I)\mathbf{x}_i.$$

If we use the normalized Laplacian and assume $\|\bar{L}\| = 2$, then

$$\mathbf{x} * \mathbf{g}_r = \sum_{i=1}^{s} w_i^r (I + D^{-1/2}AD^{-1/2})\mathbf{x}_i = (\underbrace{I}_{\text{self-loop}} + \underbrace{D^{-1/2}AD^{-1/2}}_{\text{1-hop neighbors}})X\mathbf{w}_r.$$

Comparing to (11.4), we see that GCN first adds the self-loop to the adjacency matrix to get $\mathring{A}$ and then renormalizes to get the 1-hop neighbor term $\mathring{D}^{-1/2}\mathring{A}\mathring{D}^{-1/2}$.

Comparing to Chebyshev net, 1 layer of GCN only takes 1-hop neighbors into account while Chebyshev net takes all k-hop neighbors into account. However, this can be compensated by stacking k layers in GCN. Kipf and Welling (2017) applied GCN to semi-supervised node classification where cross-entropy on labeled nodes is minimized while the unlabeled nodes affect the Laplacian hence also learning of the weights W.

Kipf, Thomas N. and Max Welling (2017). "Semi-Supervised Classification with Graph Convolutional Networks". In: *International Conference on Learning Representations*.

Example 11.20: Simple graph convolution (SGC) (Wu et al. 2019)

As mentioned above, GCN replaces a layer of Chebyshev net with k compositions of a simple layer defined in (11.4):

$$X \to \sigma(\mathring{L}XW^1) \to \cdots \to \sigma(\mathring{L}XW^k), \quad \mathring{L} := \mathring{D}^{-1/2}\mathring{A}\mathring{D}^{-1/2}.$$

Surprisingly, Wu et al. (2019) showed that collapsing the above leads to similar performance, effectively bringing us back to Chebyshev net with a different polynomial parameterization:

$$X \to \sigma(\mathring{L}^k XW).$$

Wu et al. (2019) proved that the self-loop in $\mathring{A}$ effectively shrinks the spectrum.
Wu, Felix, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger (2019). "Simplifying Graph Convolutional Networks". In: *Proceedings of the 36th International Conference on Machine Learning*, pp. 6861–6871.

Exercise 11.21: Multiplication is indeed composition

Prove that the mapping $\mathbf{x} \mapsto L^k\mathbf{x}$ depends only on k-hop neighbors.

Alert 11.22: The deeper, the worse? (Oono and Suzuki 2020)

Both GCN and SGC seem to suggest that we do not need to build very deep graph networks. This is possibly due to the small-world phenomenon in many real-word graphs, namely that each node can be reached from any other node through very few hops. See Oono and Suzuki (2020) for an interesting result along this direction.
Oono, Kenta and Taiji Suzuki (2020). "Graph Neural Networks Exponentially Lose Expressive Power for Node Classification". In: *International Conference on Learning Representations*.

Algorithm 11.23: Iterative color refinement (Weisfeiler and Lehman 1968)

```
Algorithm: Weisfeiler-Lehman iterative color refinement (Weisfeiler and Lehman 1968)
  Input: Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathfrak{l}^0)$
  Output: $\mathfrak{l}^{|\mathcal{V}|-1}$
1 for $t = 0, 1, \ldots, |\mathcal{V}| - 1$ do
2     $\mathfrak{l}^{t+1} \leftarrow \mathtt{hash}\Big([\mathfrak{l}^t_v, \mathfrak{l}^t_{u\in\mathcal{N}_v}] : v \in \mathcal{V}\Big)$        // $[\cdot]$ is a multiset, allowing repetitions
```

```
Algorithm: Assuming node features $\mathfrak{l}$ from a totally ordered space $\mathsf{L}$
1 Function hash$\Big([\mathfrak{l}_v, \mathfrak{l}_{u\in\mathcal{N}_v}] : v \in \mathcal{V}\Big)$:
2     for $v \in \mathcal{V}$ do
3         sort$\Big(\mathfrak{l}_{u\in\mathcal{N}_v}\Big)$                                   // sort the neighbors
4         add $\mathfrak{l}_v$ as prefix to the sorted list $[\mathfrak{l}_v, \mathfrak{l}_{u\in\mathcal{N}_v}]$     // $\mathfrak{l}_v$ does not participate in sorting!
5         $\mathfrak{l}^+_v \leftarrow f([\mathfrak{l}_v, \mathfrak{l}_{u\in\mathcal{N}_v}])$    // $f : \mathsf{L}^* \to \mathsf{L}$ strictly increasing w.r.t. lexicographic order
```

We follow Shervashidze et al. (2011) to explain the Weisfeiler-Lehman (WL) iterative color refinement algorithm. Consider a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathfrak{l})$ with node feature $\mathfrak{l}_v$ in some totally ordered space L for each node $v \in \mathcal{V}$. For instance, we may simply set $\mathfrak{l}_v \equiv 1$ and $\mathsf{L} = \{1, 2, \ldots, |\mathcal{V}|\}$ (a.k.a. colors) if no better information is available. Then, for each node (in parallel) we repeatedly aggregate information from its neighbors and reassign its node feature using a hash function (which may change from iteration to iteration).

A typical choice for the hash function is illustrated above, based on sorting the neighbors and using a strictly increasing function $f : \mathsf{L}^* \to \mathsf{L}$ that maps the smallest neighborhood $[\mathfrak{l}_v, \mathfrak{l}_{u \in \mathcal{N}_v}]$ to the smallest element in L, and so on and so forth. (Note that in this convention f may change in different iterations in WL). By construction, the node feature $\mathfrak{l}_v$ for any node will never decrease (thanks to the monotonicity of f). W.l.o.g. we may identify $\mathsf{L} = \{1, 2, \ldots, |\mathcal{V}|\}$, from which we see that the algorithm need only repeat for at most $|\mathcal{V}|$ iterations: $|\mathcal{V}|^2 \geq \sum_v \mathfrak{l}_v \geq |\mathcal{V}|$ and each non-vacuous update increases the sum by at least 1. If we maintain a histogram on the alphabet L, then we may early stop the algorithm when the histogram stops changing. WL can be implemented in almost linear time (e.g. Berkholz et al. 2017).

As mentioned in this historic comment, WL was motivated by applications in computational chemistry, where a precursor already appeared in Morgan (1965). An interesting story about Andrey Lehman is available here while an unsettling story about the disappearance of Boris Weisfeiler is available here.

Weisfeiler, Boris and Andrey Lehman (1968). "The reduction of a graph to canonical form and the algebra which appears therein". *Nauchno-Technicheskaya Informatsia*, vol. 2, no. 9, pp. 12–16.

Shervashidze, Nino, Pascal Schweitzer, Erik Jan van Leeuwen, Kurt Mehlhorn, and Karsten M. Borgwardt (2011). "Weisfeiler-Lehman Graph Kernels". *Journal of Machine Learning Research*, vol. 12, no. 77, pp. 2539–2561.

Berkholz, C., P. Bonsma, and M. Grohe (2017). "Tight Lower and Upper Bounds for the Complexity of Canonical Colour Refinement". *Theory of Computing Systems*, vol. 60, pp. 581–614.

Morgan, H. L. (1965). "The Generation of a Unique Machine Description for Chemical Structures-A Technique Developed at Chemical Abstracts Service". *Journal of Chemical Documentation*, vol. 5, no. 2, pp. 107–113.

Algorithm 11.24: Graph isomorphism test

Testing whether two graphs are isomorphic is one of the few surprising problems in NP that we do not know if it is in NPC or P. The WL Algorithm 11.23 immediately leads to an early test for graph isomorphism: we simply "glue" the two input graphs as disjoint components into one graph and start with trivial labeling $\mathfrak{l}_v \equiv 1$. Run WL Algorithm 11.23. If at some iteration the histograms on the two components/graphs differ, then we claim "non-isomorphic." Otherwise we classify as "possibly isomorphic."

The above test was mistakenly believed to be a solution to graph isomorphism (Weisfeiler and Lehman 1968) but soon counterexamples were found. Nevertheless, Babai and Kucera (1979) and Babai et al. (1980) proved that for almost all graphs, the WL test is valid. The exact power of the WL test has been characterized in Arvind et al. (2015) and Kiefer et al. (2015).

Weisfeiler, Boris and Andrey Lehman (1968). "The reduction of a graph to canonical form and the algebra which appears therein". *Nauchno-Technicheskaya Informatsia*, vol. 2, no. 9, pp. 12–16.

Babai, L. and L. Kucera (1979). "Canonical labelling of graphs in linear average time". In: *20th Annual Symposium on Foundations of Computer Science*, pp. 39–46.

Babai, László, Paul Erdös, and Stanley M. Selkow (1980). "Random Graph Isomorphism". *SIAM Journal on Computing*, vol. 9, no. 3, pp. 628–635.

Arvind, V., Johannes Köbler, Gaurav Rattan, and Oleg Verbitsky (2015). "On the Power of Color Refinement". In: *Fundamentals of Computation Theory*, pp. 339–350.

Kiefer, Sandra, Pascal Schweitzer, and Erkal Selman (2015). "Graphs Identified by Logics with Counting". In: *Mathematical Foundations of Computer Science*, pp. 319–330.

Algorithm 11.25: High dimensional WL (e.g. Grohe 2017; Weisfeiler 1976, §O)

For any $k \geq 2$, we may *lift* the WL algorithm by considering k-tuples of nodes v in $\mathcal{V}^k$. Variations on the neighborhood $\mathcal{N}_\mathsf{v}$ include:

- $\mathrm{WL_k}$: $\mathcal{N}_\mathsf{v} := [\mathcal{N}_{\mathsf{v},1}, \ldots, \mathcal{N}_{\mathsf{v},k}]$, where $\mathcal{N}_{\mathsf{v},j} = [\mathsf{u} \in \mathcal{V}^k : \mathsf{u}_{\setminus j} = \mathsf{v}_{\setminus j}]$.
- $\mathrm{fWL_k}$: $\mathcal{N}_\mathsf{v} := [\mathcal{N}_{\mathsf{v},u} : u \in \mathcal{V}]$, where $\mathcal{N}_{\mathsf{v},u} = [(u, \mathsf{v}_2, \ldots, \mathsf{v}_k), (\mathsf{v}_1, u, \ldots, \mathsf{v}_k), \ldots, (\mathsf{v}_1, \mathsf{v}_2, \ldots, u)]$.

- $\mathrm{sWL_k}$ (Morris et al. 2019): $\mathcal{N}_{\mathsf{v}} := [\mathsf{u} \in \mathcal{V}^k : |\mathsf{u} \cap \mathsf{v}| = k - 1]$.

We initialize k-tuples u and v with the same node feature (color) if the (ordered) subgraph they induce are isomorhpic (and with the same node features inherited from the original graph). The WL Algorithm 11.23 will be denoted as $\mathrm{WL_1}$; see (Grohe 2017, p. 84) on how to unify the description.

It is known that $\mathrm{WL_{k+1}}$ is as powerful as $\mathrm{fWL_k}$ (Grohe and Otto 2015). For $k \geq 2$, $\mathrm{WL_{k+1}}$ is strictly more powerful than $\mathrm{WL_k}$ (Cai et al. 1992; Grohe and Otto 2015, Observation 5.13 and Theorem 5.17), while $\mathrm{WL_1}$ is equivalent to $\mathrm{WL_2}$ (Cai et al. 1992; Grohe and Otto 2015). Moreover, $\mathrm{sWL_k}$ is strictly weaker than $\mathrm{WL_k}$ (Sato 2020, page 15).

Grohe, Martin (2017). *Descriptive Complexity, Canonisation, and Definable Graph Structure Theory*. Cambridge University Press.

Weisfeiler, Boris (1976). *On Construction and Identification of Graphs*. Springer.

Morris, Christopher, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe (2019). "Weisfeiler and Leman Go Neural Higher-Order Graph Neural Networks". In: *Proceedings of the AAAI Conference on Artificial Intelligence*.

Grohe, Martin and Martin Otto (2015). "Pebble Games and Linear Equations". *The Journal of Symbolic Logic*, vol. 80, no. 3, pp. 797–844.

Cai, J., M. Fürer, and N. Immerman (1992). "An optimal lower bound on the number of variables for graph identification". *Combinatorica*, vol. 12, pp. 389–410.

Sato, Ryoma (2020). "A Survey on The Expressive Power of Graph Neural Networks".

Remark 11.26: The connection between WL and GCN

The similarity between WL Algorithm 11.23 and GCN is recognized in (Kipf and Welling 2017). Indeed, consider the following specialization of the `hash` function in Algorithm 11.23:

$$\mathfrak{l}_v^{l+1} = \sigma\left(\left[\tfrac{1}{d_v+1}\mathfrak{l}_v + \sum_{u\in\mathcal{N}_v} \tfrac{a_{vu}}{\sqrt{(d_v+1)(d_u+1)}}\mathfrak{l}_u^l\right] W^l\right),$$

which is exactly the GCN update in (11.4) (with the identification $X_{v:} = \mathfrak{l}_v$). From this observation we see that even with random weights W, GCN may still be able to extract useful node features, as confirmed through an example in (Kipf and Welling 2017, Appendix A.1).

More refined and exciting findings along this connection have appeared in Xu et al. (e.g. 2019), Maron et al. (2019), Morris et al. (2019), and Sato (2020) lately. See also Kersting et al. (2009) and Kersting et al. (2014) for applications to graphical models.

Kipf, Thomas N. and Max Welling (2017). "Semi-Supervised Classification with Graph Convolutional Networks". In: *International Conference on Learning Representations*.

Xu, Keyulu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka (2019). "How Powerful are Graph Neural Networks?" In: *International Conference on Learning Representations*.

Maron, Haggai, Heli Ben-Hamu, Hadar Serviansky, and Yaron Lipman (2019). "Provably Powerful Graph Networks". In: *Advances in Neural Information Processing Systems 32*, pp. 2156–2167.

Morris, Christopher, Martin Ritzert, Matthias Fey, William L. Hamilton, Jan Eric Lenssen, Gaurav Rattan, and Martin Grohe (2019). "Weisfeiler and Leman Go Neural Higher-Order Graph Neural Networks". In: *Proceedings of the AAAI Conference on Artificial Intelligence*.

Sato, Ryoma (2020). "A Survey on The Expressive Power of Graph Neural Networks".

Kersting, Kristian, Babak Ahmadi, and Sriraam Natarajan (2009). "Counting Belief Propagation". In: *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 277–284.

Kersting, Kristian, Martin Mladenov, Roman Garnett, and Martin Grohe (2014). "Power Iterated Color Refinement". In: *The Twenty-Eighth AAAI Conference on Artificial Intelligence (AAAI)*.

12 k-Nearest Neighbors (kNN)

Goal

Understand k-nearest neighbors for classification and regression. Relation to Bayes error.

Alert 12.1: Convention

Gray boxes are not required hence can be omitted for unenthusiastic readers.

This note is likely to be updated again soon.

Definition 12.2: Distance

Given a domain $\mathsf{X} \subseteq \mathbb{R}^d$, we define a distance metric $\mathrm{dist} : \mathsf{X} \times \mathsf{X} \to \mathbb{R}_+$ as any function that satisfies the following axioms:

- nonnegative: $\mathrm{dist}(\mathbf{x}, \mathbf{z}) \geq 0$;
- identity: $\mathrm{dist}(\mathbf{x}, \mathbf{z}) = 0$ iff $\mathbf{x} = \mathbf{z}$;
- symmetric: $\mathrm{dist}(\mathbf{x}, \mathbf{z}) = \mathrm{dist}(\mathbf{z}, \mathbf{x})$;
- triangle inequality: $\mathrm{dist}(\mathbf{x}, \mathbf{z}) \leq \mathrm{dist}(\mathbf{x}, \mathbf{y}) + \mathrm{dist}(\mathbf{y}, \mathbf{z})$.

We call the space X equipped with a distance metric dist a metric space, with notation $(\mathsf{X}, \mathrm{dist})$.

If we relax the "iff" part in identity to "if" then we obtain pseudo-metric; if we drop symmetry we obtain quasi-metric; and finally if we drop the triangle inequality we get semi-metric.

Exercise 12.3: Example distances

Given any norm $\|\cdot\|$ on a vector space V, it immediately induces a distance metric:

$$\mathrm{dist}_{\|\cdot\|}(\mathbf{x}, \mathbf{z}) = \|\mathbf{x} - \mathbf{z}\|.$$

Verify by yourself $\mathrm{dist}_{\|\cdot\|}$ is indeed a distance metric.

In particular, for the ℓ_p norm defined in Definition 1.25, we obtain the ℓ_p distance.

Another often used "distance" is the cosine similarity:

$$\angle(\mathbf{x}, \mathbf{z}) = \frac{\mathbf{x}^\top \mathbf{z}}{\|\mathbf{x}\|_2 \cdot \|\mathbf{z}\|_2}.$$

Is it a distance metric?

Remark 12.4: kNN in a nutshell

Given a metric space $(\mathsf{X}, \mathrm{dist})$ and a dataset $\mathcal{D} = \llbracket(\mathbf{x}_1, y_1) \ldots, (\mathbf{x}_n, y_n)\rrbracket$, where $\mathbf{x}_i \in \mathsf{X}$, upon receiving a new instance $\mathbf{x} \in \mathsf{X}$, it is natural to find near neighbors (e.g. "friends") in our dataset $\mathcal{D}$ according to the metric dist and predict $\hat{y}(\mathbf{x})$ according to the y-values of the neighbors. The underlying assumption is

neighboring feature vectors tend to have similar or same y-values.

The subtlety of course lies on what do we mean by neighboring, i.e., how do we choose the metric dist.

Remark 12.5: The power of an appropriate metric

Suppose we have $(\mathbf{X}, Y)$ following some distribution on $\mathsf{X} \times \mathsf{Y}$, where the target space Y is equipped with some metric dist_y (acting as a measure of our prediction error). Then, we may define a (pseudo)metric on X as:

$$\mathrm{dist}_x(\mathbf{x}, \mathbf{x}') := \mathsf{E}[\mathrm{dist}_y(Y, Y') | \mathbf{X} = \mathbf{x}, \mathbf{X}' = \mathbf{x}'],$$

where $(\mathbf{X}', Y')$ is an independent copy of $(\mathbf{X}, Y)$. (Note that $\mathrm{dist}_x(\mathbf{x}, \mathbf{x}) = 0$ may not hold.) Given a test instance $\mathbf{X} = \mathbf{x}$, if we can find a near neighbor $\mathbf{X}' = \mathbf{x}'$ so that $\mathrm{dist}_x(\mathbf{x}, \mathbf{x}') \leq \epsilon$, then predicting $Y(\mathbf{x})$ according to $Y(\mathbf{x}')$ gives us at most ϵ error:

$$\mathsf{E}[\mathrm{dist}_y(Y(\mathbf{X}), Y(\mathbf{X}'))] = \mathsf{E}[\mathrm{dist}_x(\mathbf{X}, \mathbf{X}')] \leq \epsilon.$$

Of course, we would not be able to construct the distance metric dist_x in practice, as it depends on the unknown distribution of our data.

Algorithm 12.6: kNN

Given a dataset $\mathcal{D} = ⟅(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_n, \mathbf{y}_n)⟆$, where $\mathbf{x}_i \in (\mathsf{X}, \mathrm{dist})$ and $\mathbf{y}_i \in \mathsf{Y}$, and a test instance $\mathbf{x}$, we predict according to the knn algorithm:

```
Algorithm: kNN
   Input: Dataset $\mathcal{D} = ⟅(\mathbf{x}_i, \mathbf{y}_i) \in \mathsf{X} \times \mathsf{Y} : i = 1, \ldots, \mathsf{n}⟆$, new instance $\mathbf{x} \in \mathsf{X}$, hyperparameter $k$
   Output: $\mathbf{y} = \mathbf{y}(\mathbf{x})$
1  for $i = 1, 2, \ldots, \mathsf{n}$ do
2      $d_i \leftarrow \mathrm{dist}(\mathbf{x}, \mathbf{x}_i)$                    // avoid for-loop if possible
3  find indices $i_1, \ldots, i_k$ of the $k$ smallest entries in $\mathbf{d}$
4  $\mathbf{y} \leftarrow$ aggregate($\mathbf{y}_{i_1}, \ldots, \mathbf{y}_{i_k}$)
```

For different target space Y, we may use different aggregations:

- multi-class classification $\mathsf{Y} = \{1, \ldots, \mathsf{c}\}$: we can perform majority voting

$$\mathbf{y} \leftarrow \operatorname*{argmax}_{j=1,\ldots,\mathsf{c}} \#\{\mathbf{y}_{i_l} = j : l = 1, \ldots, k\}, \tag{12.1}$$

  where ties can be broken arbitrarily.

- regression: $\mathsf{Y} = \mathbb{R}^m$: we can perform averaging

$$\mathbf{y} \leftarrow \frac{1}{k} \sum_{l=1}^{k} \mathbf{y}_{i_l}. \tag{12.2}$$

Strictly speaking, there is no training time in kNN as we need only store the dataset $\mathcal{D}$. For testing, it costs $O(nd)$ as we have to go through the entire dataset to compute all distances to the test instance. There is a large literature that aims to bring down this complexity in test time by pre-processing our dataset and often by contending with near (but not necessarily nearest) neighbors (see e.g. Andoni and Indyk (2008)).

Andoni, Alexandr and Piotr Indyk (2008). "Near-optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions". *Communications of the ACM*, vol. 51, no. 1, pp. 117–122.

Exercise 12.7: The power of weights

More generally, suppose we also have a distance metric dist_y on Y, we may set

$$\pi \leftarrow \operatorname*{argmin}_{\pi} \sum_{i=1}^{n} w_i^{\downarrow} \cdot \mathrm{dist}_x(\mathbf{x}, \mathbf{x}_{\pi(i)}) \tag{12.3}$$

$$\mathbf{y} \leftarrow \operatorname*{argmin}_{\mathbf{y}\in\mathsf{Y}} \sum_{i=1}^{n} v_i^{\downarrow} \cdot \mathrm{dist}_y^{2}(\mathbf{y}, \mathbf{y}_{\pi(i)}), \tag{12.4}$$

where $\pi : [n] \to [n]$ is a permutation, and $w_1 \geq w_2 \geq \cdots \geq w_n \geq 0$, $v_1 \geq v_2 \geq \cdots \geq v_n \geq 0$ are weights (e.g. how much each training instance should contribute to the final result). We may also use dist_y in (12.4) (without squaring). A popular choice is to set $v_i \propto 1/d_{\pi(i)}$ so that nearer neighbors will contribute more to predicting $\mathbf{y}$.

Prove that with the following choices we recover (12.1) and (12.2) from (12.3)-(12.4), respectively:

- Let $\mathsf{Y} = \{1, \ldots, \mathsf{c}\}$ and $\mathrm{dist}_y(\mathbf{y}, \mathbf{y}') = \begin{cases} 0, & \text{if } \mathbf{y} = \mathbf{y}' \\ 1, & \text{o.w.} \end{cases}$ be the discrete distance. Use the kNN weights $\mathbf{w} = \mathbf{v} = (\underbrace{1, \ldots, 1}_{k}, 0, \ldots, 0)$.
- Let $\mathsf{Y} = \mathbb{R}^m$ and $\mathrm{dist}_y(\mathbf{y}, \mathbf{y}') = \|\mathbf{y} - \mathbf{y}'\|_2$ be the ℓ_2 distance.

Remark 12.8: Effect of k

Intuitively, using a larger k would give us more stable predictions (if we vary the training dataset), as we are averaging over more neighbors, corresponding to smaller variance but potentially larger bias (see **??**):

- If we use $k = n$, then we always predict the same target irrespective of the input $\mathbf{x}$, which is clearly not varied at all but may incur a large bias.
- Indeed, if we have a dataset where different classes are *well* separated, then using a large k can bring significant bias while 1NN achieves near 0 error.

In practice we may select k using cross-validation (see Algorithm 2.33). For a moderately large dataset, typically $k = 3$ or 5 suffices. A rule of thumb is we use larger k for larger and more difficult datasets.

Theorem 12.9: kNN generalization error (Biau and Devroye 2015)

Let k be odd and fixed. Then, for all distributions of $(\mathbf{X}, Y)$, as $n \to \infty$,

$$\mathbb{L}_{kNN} := \Pr[h_n(\mathbf{X}) \neq Y] \to \mathsf{E}\left[\sum_{l=0}^{k} \binom{k}{l} \mathsf{r}^l(\mathbf{X})(1 - \mathsf{r}(\mathbf{X}))^{k-l}\Big(\mathsf{r}(\mathbf{X})[\![l < \tfrac{k}{2}]\!] + (1 - \mathsf{r}(\mathbf{X}))[\![l \geq \tfrac{k}{2}]\!]\Big)\right],$$

where the knn classifier h_n is defined in (12.5) and $\mathsf{r}(\mathbf{x}) := \Pr[Y = 1|\mathbf{X} = \mathbf{x}]$ is the regression function.

Proof. Let $\mathbf{X}_1, \ldots, \mathbf{X}_n \overset{i.i.d.}{\sim} \mathbf{X}$ and let $Y_i = [\![U_i \leq \mathsf{r}(\mathbf{X}_i)]\!]$, where $U_i \overset{i.i.d.}{\sim} \mathrm{Uniform}([0, 1])$. Clearly, $(\mathbf{X}_i, Y_i, U_i)$ form an i.i.d. sequence where $(\mathbf{X}_i, Y_i) \sim (\mathbf{X}, Y)$. Let $\mathcal{D}_n = \{(\mathbf{X}_i, Y_i, U_i), i = 1, \ldots, n\}$. Fixing $\mathbf{x}$, define $\tilde{Y}_i(\mathbf{x}) = [\![U_i \leq \mathsf{r}(\mathbf{x})]\!]$. Order $\mathbf{X}_{(i)}(\mathbf{x})$, $Y_{(i)}(\mathbf{x})$, $\tilde{Y}_{(i)}(\mathbf{x})$ and $U_{(i)}(\mathbf{x})$ according to the distance $\mathrm{dist}(\mathbf{X}_i, \mathbf{x})$. Consider the classifiers:

$$h_n(\mathbf{x}) = \begin{cases} 1, & \text{if } \sum_{l=1}^{k} Y_{(l)}(\mathbf{x}) > k/2 \\ 0, & \text{o.w.} \end{cases}, \qquad \tilde{h}_n(\mathbf{x}) = \begin{cases} 1, & \text{if } \sum_{l=1}^{k} \tilde{Y}_{(l)}(\mathbf{x}) > k/2 \\ 0, & \text{o.w.} \end{cases}. \tag{12.5}$$

Then, we have

$$\begin{aligned}\Pr[h_n(\mathbf{X}) \neq \tilde{h}_n(\mathbf{X})] &\leq \Pr\left[\sum_{l=1}^k Y_{(l)}(\mathbf{X}) \neq \sum_{l=1}^k \tilde{Y}_{(l)}(\mathbf{X})\right] \\ &\leq \Pr\left[\left(Y_{(1)}(\mathbf{X}), \ldots, Y_{(k)}(\mathbf{X})\right) \neq \left(\tilde{Y}_{(1)}(\mathbf{X}), \ldots, \tilde{Y}_{(k)}(\mathbf{X})\right)\right] \\ &\leq \Pr\left[\bigcup_{l=1}^k [\![\mathsf{r}(\mathbf{X}_{(l)}(\mathbf{X})) \wedge \mathsf{r}(\mathbf{X}) < U_{(l)}(\mathbf{X}) \leq \mathsf{r}(\mathbf{X}_{(l)}(\mathbf{X})) \vee \mathsf{r}(\mathbf{X})]\!]\right] \\ &\leq \sum_{l=1}^k \mathsf{E}\left|\mathsf{r}(\mathbf{X}_{(l)}(\mathbf{X})) - \mathsf{r}(\mathbf{X})\right| \overset{n\to\infty}{\longrightarrow} 0, \text{ see Stone's Lemma 12.13 below.}\end{aligned}$$

Recall that $\mathfrak{L}(h_n) := \Pr(h_n(\mathbf{X}) \neq Y|\mathcal{D})$ and similarly for $\mathfrak{L}(\tilde{h}_n)$. Thus,

$$\mathsf{E}\left|\mathfrak{L}(h_n) - \mathfrak{L}(\tilde{h}_n)\right| \leq \Pr[h_n(\mathbf{X}) \neq \tilde{h}_n(\mathbf{X})] = o(1),$$

whereas noting that given $\mathbf{x}$, $\tilde{Y}_l(\mathbf{x}) \overset{i.i.d.}{\sim}$ Bernoulli($\mathsf{r}(\mathbf{x})$), hence

$$\begin{aligned}\mathsf{E}\mathfrak{L}(\tilde{h}_n) &= \Pr\left[\text{Binomial}(k, \mathsf{r}(\mathbf{X})) > \tfrac{k}{2}, Y = 0\right] + \Pr\left[\text{Binomial}(k, \mathsf{r}(\mathbf{X})) \leq \tfrac{k}{2}, Y = 1\right] \\ &= \mathsf{E}\left[(1 - \mathsf{r}(\mathbf{X}))[\![\text{Binomial}(k, \mathsf{r}(\mathbf{X})) > \tfrac{k}{2}]\!] + \mathsf{r}(\mathbf{X})[\![\text{Binomial}(k, \mathsf{r}(\mathbf{X})) \leq \tfrac{k}{2}]\!]\right].\end{aligned}$$

Combining the above completes the proof. □

The proof above exploits the beautiful decoupling idea: $Y_{(i)}$'s, which the kNN classifier g_n depends on, are coupled through the ordering induced by the $\mathbf{X}_i$'s. On the other hand, $\tilde{Y}_{(i)}$'s are independent (conditioned on $\mathbf{X} = \mathbf{x}$) hence allow us to analyze the closely related classifier $\tilde{g}_n$ with ease. Stone's Lemma 12.13 adds the final piece that establishes the asymptotic equivalence of the two classifiers.

Biau, Gérard and Luc Devroye (2015). *Lectures on the Nearest Neighbor Method*. Springer.

Corollary 12.10: 1NN $\leq 2\times$Bayes (Cover and Hart 1967)

For $n \to \infty$, we have

$$\mathbb{L}_{Bayes} \leq \mathbb{L}_{1NN} \leq 2\mathbb{L}_{Bayes}(1 - \mathbb{L}_{Bayes}) \leq 2\mathbb{L}_{Bayes},$$

and $\mathbb{L}_{3NN} = \mathsf{E}[\mathsf{r}(\mathbf{X})(1 - \mathsf{r}(\mathbf{X}))] + 4\mathsf{E}[\mathsf{r}^2(\mathbf{X})(1 - \mathsf{r}(\mathbf{X}))^2]$.

Proof. For $k = 1$, it follows from Theorem 12.9 that

$$\mathbb{L}_{\text{1NN}} = 2\mathsf{E}[\mathsf{r}(\mathbf{X})(1 - \mathsf{r}(\mathbf{X}))]$$

whereas the Bayes error is

$$\mathbb{L}_{\text{Bayes}} = \mathsf{E}[\mathsf{r}(\mathbf{X}) \wedge (1 - \mathsf{r}(\mathbf{X}))].$$

Therefore, letting $\mathsf{s}(\mathbf{x}) = \mathsf{r}(\mathbf{x}) \wedge (1 - \mathsf{r}(\mathbf{x}))$, we have

$$\mathbb{L}_{\text{1NN}} = 2\mathsf{E}[\mathsf{s}(\mathbf{X})(1 - \mathsf{s}(\mathbf{X}))] = 2\mathsf{E}\mathsf{s}(\mathbf{X}) \cdot \mathsf{E}(1 - \mathsf{s}(\mathbf{X})) - 2 \cdot \text{Variance}(\mathsf{s}(\mathbf{X})) \leq 2\mathbb{L}_{\text{Bayes}}(1 - \mathbb{L}_{\text{Bayes}}).$$

The formula for $\mathbb{L}_{\text{3NN}}$ follows immediately from Theorem 12.9. □

We note that for trivial problems where $\mathbb{L}_{\text{Bayes}} = 0$ or $\mathbb{L}_{\text{Bayes}} = \frac{1}{2}$, $\mathbb{L}_{\text{1NN}} = \mathbb{L}_{\text{Bayes}}$. On the other hand, when the Bayes error is small, $\mathbb{L}_{\text{1NN}} \sim 2\mathbb{L}_{\text{Bayes}}$ while $\mathbb{L}_{\text{3NN}} \sim \mathbb{L}_{\text{Bayes}}$.

Cover, T. M. and P. E. Hart (1967). "Nearest Neighbor Pattern Classification". *IEEE Transactions on Information Theory*, vol. 13, no. 1, pp. 21–27.

Proposition 12.11: Continuity

Let $f : \mathbb{R}^d \to \mathbb{R}$ be (Lebesgue) integrable. If $k/n \to 0$, then

$$\frac{1}{k}\sum_{l=1}^{k} \mathsf{E}\left|f\big(\mathbf{X}_l(\mathbf{X})\big) - f(\mathbf{X})\right| \to 0,$$

where $\mathbf{X}_{(i)}(\mathbf{X})$ is ordered by the distance $\|\mathbf{X}_i - \mathbf{X}\|_2$ and $\mathbf{X}_i \sim \mathbf{X}$ for $i = 1, \ldots, n$.

Proof. Since $\mathcal{C}_c$ is dense in $\mathcal{L}_1$, we may approximate f by a (uniformly) continuous function f_ϵ with compact support. In particular, for $\epsilon > 0$ there exists $\delta > 0$ such that $\mathrm{dist}(\mathbf{x}, \mathbf{z}) \le \delta \implies |f_\epsilon(\mathbf{x}) - f_\epsilon(\mathbf{z})| \le \epsilon$. Thus,

$$\begin{aligned}
\frac{1}{k}\sum_{l=1}^{k} \mathsf{E}\left|f\big(\mathbf{X}_l(\mathbf{X})\big) - f(\mathbf{X})\right| &\le \frac{1}{k}\sum_{l=1}^{k} \mathsf{E}\left|f\big(\mathbf{X}_l(\mathbf{X})\big) - f_\epsilon(\mathbf{X}_l(\mathbf{X}))\right| + \mathsf{E}\left|f_\epsilon\big(\mathbf{X}_l(\mathbf{X})\big) - f_\epsilon(\mathbf{X})\right| + \mathsf{E}\left|f_\epsilon(\mathbf{X}) - f(\mathbf{X})\right| \\
\text{(Stone's Lemma 12.13)} \quad &\le (\gamma_d + 2)\mathsf{E}\left|f(\mathbf{X}) - f_\epsilon(\mathbf{X})\right| + 2\|f_\epsilon\|_\infty \cdot \Pr[\mathrm{dist}(\mathbf{X}_{(k)}, \mathbf{X}) > \delta] + \epsilon \\
&\le (\gamma_d + 2)\epsilon + 2\|f_\epsilon\|_\infty \cdot \Pr[\mathrm{dist}(\mathbf{X}_{(k)}, \mathbf{X}) > \delta] \\
&\le (\gamma_d + 3)\epsilon, \quad \text{thanks to Theorem 12.12 when } n \text{ is large.}
\end{aligned}$$

The proof is complete by noting that ϵ is arbitrary. □

Theorem 12.12: projection through kNN

Fix $\mathbf{x}$ and define $\rho = \mathrm{dist}(\mathbf{x}, \mathrm{supp}\mu)$ where $\mathrm{supp}\mu$ is the support of some measure μ. If $k/n \to 0$, then almost surely

$$\mathrm{dist}(\mathbf{X}_{(k)}(\mathbf{x}), \mathbf{x}) \to \rho,$$

where $\mathbf{X}_i \overset{i.i.d.}{\sim} \mu$ and $\mathbf{X}_{(i)}$ is ordered by $\mathrm{dist}(\mathbf{X}_i, \mathbf{x})$, $i = 1, \ldots, n$.

Proof. Fix any $\epsilon > 0$ and let $p = \Pr(\mathrm{dist}(\mathbf{X}, \mathbf{x}) \le \epsilon + \rho) > 0$. Then, for large n,

$$\begin{aligned}
\Pr(\mathrm{dist}(\mathbf{X}_{(k)}, \mathbf{x}) - \rho > \epsilon) &= \Pr\left(\sum_{i=1}^{n} B_i < k\right), \quad \text{where} \quad B_i \overset{i.i.d.}{\sim} \mathrm{Bernoulli}(p) \\
&= \Pr\left(\frac{1}{n}\sum_{i=1}^{n}(B_i - p) < k/n - p\right) \\
&\le \exp\left(-2n(p - k/n)^2\right).
\end{aligned}$$

Since $p > 0$ and $k/n \to 0$, the theorem follows. □

Let $\mathbf{X} \sim \mu$ be another independent copy, then with $k/n \to 0$:

$$\mathrm{dist}(\mathbf{X}_{(k)}, \mathbf{X}) \xrightarrow{a.s.} 0.$$

Indeed, for μ-almost all $\mathbf{x}$ and large n, we have

$$\Pr\left[\sup_{m \ge n} \mathrm{dist}(\mathbf{X}_{(k,m)}(\mathbf{x}), \mathbf{x}) \ge \epsilon\right] \le \sum_{m \ge n} \exp(-mp^2) \xrightarrow{n \to \infty} 0.$$

Lemma 12.13: Stone's Lemma (Stone 1977)

Let $(w_1^{(n)}, \ldots, w_n^{(n)})$ be a probability vector with $w_1^{(n)} \geq \cdots \geq w_n^{(n)}$ for all n. Then, for any integrable function $f : \mathbb{R}^d \to \mathbb{R}$,

$$\mathsf{E}\left[\sum_{i=1}^{n} w_i^{(n)} \left|f(\mathbf{X}_{(i)}(\mathbf{X})\right|\right] \leq (1+\gamma_d)\mathsf{E}|f(\mathbf{X})|,$$

where $\mathbf{X}_i$'s are i.i.d. copies of $\mathbf{X}$, $\mathbf{X}_{(i)}$'s are ordered by $\|\mathbf{X}_i - \mathbf{X}\|_2$, and $\gamma_d < \infty$ only depends on d.

Proof. Define

$$W_i^{(n)}(\mathbf{x}) := W_i^{(n)}(\mathbf{x}; \mathbf{x}_1, \ldots, \mathbf{x}_n) := w_k^{(n)}$$

if $\mathbf{x}_i$ is the k-th nearest neighbor of $\mathbf{x}$ (ties broken by index). We first prove

$$\sum_{i=1}^{n} W_i^{(n)}(\mathbf{x}_i; \mathbf{x}_1, \ldots, \mathbf{x}_{i-1}, \mathbf{x}, \mathbf{x}_{i+1}, \ldots, \mathbf{x}_n) \leq (1+\gamma_d). \tag{12.6}$$

Cover $\mathbb{R}^d$ with γ_d angular cones $\mathsf{K}_t, t = 1, \ldots, \gamma_d$, each with angle $\pi/12$. Let $A = \{i : \mathbf{x}_i = \mathbf{x}\}$ and $B_t = \{i : \mathbf{x}_i \in (\mathsf{K}_t + \mathbf{x}) \setminus \{\mathbf{x}\}\}$. Choose any $a, b \in B_t$ such that $0 < \|\mathbf{x}_a - \mathbf{x}\| \leq \|\mathbf{x}_b - \mathbf{x}\|$, then

$$\|\mathbf{x}_a - \mathbf{x}_b\|^2 \leq \|\mathbf{x}_a - \mathbf{x}\|^2 + \|\mathbf{x}_b - \mathbf{x}\|^2 - 2\|\mathbf{x}_a - \mathbf{x}\|\|\mathbf{x}_b - \mathbf{x}\|\cos(\pi/6) < \|\mathbf{x}_b - \mathbf{x}\|^2. \tag{12.7}$$

Therefore, if $\mathbf{x}_b$ is the k-th closest to $\mathbf{x}$ among $\mathbf{x}_{B_t}$, then $\mathbf{x}$ is at best the k-th closest to $\mathbf{x}_b$ among $\mathbf{x}, \mathbf{x}_{B_t \setminus \{b\}}$. Since the weights $w_i^{(n)}$ are ordered, we have

$$\sum_{i \in B_t} W_i^{(n)}(\mathbf{x}_i; \mathbf{x}_1, \ldots, \mathbf{x}_{i-1}, \mathbf{x}, \mathbf{x}_{i+1}, \ldots, \mathbf{x}_n) \leq \sum_{i=1}^{n-|A|} w_i^{(n)} \leq 1$$
$$\sum_{i \in A} W_i^{(n)}(\mathbf{x}_i; \mathbf{x}_1, \ldots, \mathbf{x}_{i-1}, \mathbf{x}, \mathbf{x}_{i+1}, \ldots, \mathbf{x}_n) = \sum_{i=1}^{|A|} w_i^{(n)} \leq 1.$$

Taking unions over the γ_d angular cones proves (12.6).

Therefore,

$$\begin{aligned}
\mathsf{E}\left[\sum_{i=1}^{n} w_i^{(n)} \left|f(\mathbf{X}_{(i)}(\mathbf{X})\right|\right] &= \mathsf{E}\left[\sum_{i=1}^{n} W_i^{(n)}(\mathbf{X}) \left|f(\mathbf{X}_i)\right|\right] \\
\text{(symmetrization)} \quad &= \mathsf{E}\left[|f(\mathbf{X})| \sum_{i=1}^{n} W_i^{(n)}(\mathbf{X}_i; \mathbf{X}_1, \ldots, \mathbf{X}_{i-1}, \mathbf{X}, \mathbf{X}_{i+1}, \ldots, \mathbf{X}_n)\right] \\
&\leq (1+\gamma_d)\mathsf{E}|f(\mathbf{X})|.
\end{aligned}$$

□

Here γ_d is the covering number of $\mathbb{R}^d$ by angular cones:

$$\mathsf{K}(\mathbf{z}, \theta) := \{\mathbf{x} \in \mathbb{R}^d : \angle(\mathbf{x}, \mathbf{z}) \leq \theta\}.$$

The proof above relies on the ℓ_2 distance only in (12.7).

Stone, Charles J. (1977). "Consistent Nonparametric Regression". *The Annals of Statistics*, vol. 5, no. 4, pp. 595–620.

Theorem 12.14: No free lunch (Shalev-Shwartz and Ben-David 2014)

Let h be any classifier learned from a training set $\mathcal{D}_n$ with size $n \leq |\mathsf{X}|/2$. Then, there exists a distribution $(\mathbf{X}, Y)$ over $\mathsf{X} \times \{0, 1\}$ such that the Bayes error is zero while

$$\Pr\left[h(\mathbf{X}; \mathcal{D}_n) \neq Y\right] \geq \tfrac{1}{4}.$$

In particular, with probability at least $\frac{1}{7}$ over the training set $\mathcal{D}_n$ we have $\Pr\left[h(\mathbf{X}; \mathcal{D}_n) \neq Y | \mathcal{D}_n\right] \geq \frac{1}{8}$.

Proof. We may assume w.l.o.g. that $|\mathsf{X}| = 2n$. Enumerate all $T = 2^{2n}$ functions $h_t : \mathsf{X} \to \{0, 1\}$, each of which induces a distribution where $\mathbf{X} \in \mathsf{X}$ is uniformly random while $Y = h_t(\mathbf{X})$. For each labeling function h_t, we have $S = (2n)^n$ possible training sets $\mathcal{D}_n(s, t)$. Thus,

$$\begin{aligned}
\max_{t \in [T]} \frac{1}{S} \sum_{s=1}^{S} \Pr[h(\mathbf{X}; \mathcal{D}_n(s,t)) \neq h_t(\mathbf{X})] &\geq \frac{1}{T} \sum_{t=1}^{T} \frac{1}{S} \sum_{s=1}^{S} \Pr[h(\mathbf{X}; \mathcal{D}_n(s,t)) \neq h_t(\mathbf{X})] \\
&\geq \min_{s \in [S]} \frac{1}{T} \sum_{t=1}^{T} \Pr[h(\mathbf{X}; \mathcal{D}_n(s,t)) \neq h_t(\mathbf{X})] \\
&\geq \min_{s \in [S]} \frac{1}{T} \sum_{t=1}^{T} \frac{1}{2|\mathsf{X} \setminus \mathcal{D}_n(s,t)|} \sum_{\mathbf{x}_i \in \mathsf{X} \setminus \mathcal{D}_n(s,t)} [\![h(\mathbf{x}_i; \mathcal{D}_n(s,t)) \neq h_t(\mathbf{x}_i)]\!] \\
&= \min_{s \in [S]} \frac{1}{2|\mathsf{X} \setminus \mathcal{D}_n(s,t)|} \frac{1}{T} \sum_{t=1}^{T} \sum_{\mathbf{x}_i \in \mathsf{X} \setminus \mathcal{D}_n(s,t)} [\![h(\mathbf{x}_i; \mathcal{D}_n(s,t)) \neq h_t(\mathbf{x}_i)]\!] \\
&\geq \tfrac{1}{4},
\end{aligned}$$

since we apparently have

$$[\![h(\mathbf{x}_i; \mathcal{D}_n(s,t)) \neq h_t(\mathbf{x}_i)]\!] + [\![h(\mathbf{x}_i; \mathcal{D}_n(s,\tau)) \neq h_\tau(\mathbf{x}_i)]\!] = 1,$$

for two labeling functions h_t and h_τ which agree on $\mathbf{x}$ iff $\mathbf{x} \in \mathcal{D}_n$. □

Let $c > 1$ be arbitrary. Consider the uniform grid X in the cube $[0, 1]^d$ with $1/c$ distance between neighbors. Clearly, there are $(c + 1)^d$ points in X. If our training set is smaller than $(c + 1)^d/2$, then kNN suffers at least $1/4$ error while the Bayes error is 0! Thus, the condition $n \to \infty$ in Theorem 12.9 can be very unrealistic in high dimensions!

Shalev-Shwartz, Shai and Shai Ben-David (2014). *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press.

13 Decision Trees

Goal

Define and understand the classic decision trees. Bagging and random forest.

Alert 13.1: Convention

Gray boxes are not required hence can be omitted for unenthusiastic readers.

This note is likely to be updated again soon.

14 Boosting

Goal

Understand the ensemble method for combining classifiers. Bagging, Random Forest, and the celebrated Adaboost.

Alert 14.1: Convention

Gray boxes are not required hence can be omitted for unenthusiastic readers.

This note is likely to be updated again soon.

Remark 14.2: Together and stronger?

Often it is possible to train a variety of different classifiers for a particular problem at hand, and a lot of time, energy and discussion are spent on debating and choosing the most appropriate classifier. This makes sense when the classifiers are "expensive" to obtain (be it computationally or financially or resourcefully).

Putting operational costs aside, however, is it possible to combine a bunch of classifiers and get better performance, for instance when compared against the best classifier in the gang? Of course, one usually does not know which classifier in the gang is "best" (unless when we try all of them out).

Remark 14.3: The power of aggregation

To motivate our development, let us consider an ideal scenario where we have a collection of classifiers $\{h_t : \mathcal{X} \to \{0,1\}, t = 1, 2, \ldots, 2T+1\}$ for a binary classification problem (where we encode the labels as $\{0,1\}$). Conditional on a given test sample (X, Y), we assume the classifiers h_t ~~independently~~ achieve accuracy

$$p := \Pr(h_t(X) = Y | X, Y) > \tfrac{1}{2}.$$

(For instance, if classifier h_t is trained on an independent dataset $\mathcal{D}_t$.) We predict according to the majority:

$$h(X) = \begin{cases} 1, & \text{if } \#\{t : h_t(X) = 1\} \geq T+1 \\ 0, & \text{if } \#\{t : h_t(X) = 0\} \geq T+1 \end{cases}.$$

What is the accuracy of this "meta-classifier" h? Simple calculation reveals:

$$\begin{aligned}\Pr(h(X) = Y | X, Y) &= \Pr\left(\sum_{t=1}^{2T+1} [\![h_t(X) = Y]\!] \geq T+1 \middle| X, Y\right) = \sum_{k=T+1}^{2T+1} \binom{2T+1}{k} p^k (1-p)^{2T+1-k} \\ &= \Pr\left(\frac{1}{\sqrt{(2T+1)p(1-p)}} \sum_{t=1}^{2T+1} [[\![h_t(X) = Y]\!] - p] \geq \frac{T+1-p(2T+1)}{\sqrt{(2T+1)p(1-p)}} \middle| X, Y\right) \\ &\xrightarrow{T\to\infty} 1 - \Phi\left(\frac{T+1-p(2T+1)}{\sqrt{(2T+1)p(1-p)}}\right) \xrightarrow[2p>1]{T\to\infty} 1,\end{aligned}$$

where $\Phi : \mathbb{R} \to [0,1]$ is the cdf of a standard normal distribution, and the convergence follows from the central limit theorem.

Therefore, provided that we can combine a large number of conditionally independent classifiers, each of which is slightly better than random guessing, we can approach perfect accuracy! The caveat of course is the difficulty (or even impossibility) in obtaining *decent* independent classifiers in the first place.

Exercise 14.4: Odd vs. even

Perform a similar analysis as in Remark 14.3 for $2T$ classifiers. Is there any advantage in using the odd $2T+1$ over the even $2T$?

Algorithm 14.5: Bootstrap Aggregating (Bagging, Breiman 1996)

One way to achieve (approximately) independent classifiers is to simply train them on independent datasets, which in most (if not all) applications is simply a luxury. However, we can use the same bootstrapping idea as in cross-validation (cf. Algorithm 2.33):

```
Algorithm: Bagging predictors.
   Input: training set $\mathcal{D}$, number of reptitions $T$
   Output: meta-predictor $h$
1  for $t = 1, 2, \ldots, T$ do                                  // in parallel
2      sample a new training set $\mathcal{D}_t \subseteq \mathcal{D}$        // with or without replacement
3      train predictor $h_t$ on $\mathcal{D}_t$
4  $h \leftarrow$ aggregate($h_1, \ldots, h_T$)   // majority vote for classification; average for regression
```

There are two ways to sample a new training set:

- Sample with replacement: copy a (uniformly) random element from $\mathcal{D}$ to $\mathcal{D}_t$ and repeat $|\mathcal{D}|$ times. Usually there will be repetitions of the same element in $\mathcal{D}_t$ (which has no effect on most machine learning algorithms). *This is the common choice.*

- Sample without replacement: "cut" a (uniformly) random element from (a copy of) $\mathcal{D}$ to $\mathcal{D}_t$ and repeat say $70\% \times |\mathcal{D}|$ times. There will be no repetitions in each $\mathcal{D}_t$. Note that had we repeated $|\mathcal{D}|$ times (just as in sample with replacement) we would have $\mathcal{D}_t \equiv \mathcal{D}$, which is not very useful.

Of course the training sets $\{\mathcal{D}_t : t = 1, \ldots, T\}$ are not really independent of each other, but aggregating predictors trained on them usually (but not always) improves performance.

Breiman, Leo (1996). "Bagging Predictors". *Machine Learning*, vol. 24, no. 2, pp. 123–140.

Algorithm 14.6: Randomizing output (e.g. Breiman 2000)

Bagging perturbs the training set by taking a random subset. We can also perturb the training set by simply adding noise:

- for regression tasks: replace line 2 in the bagging algorithm by "adding small Gaussian noise to each response y_i"

- for classification tasks: replace line 2 in the bagging algorithm by "randomly flip a small portion of the labels y_i"

We will come back to this perturbation later. Intuitively, adding noise can prevent overfitting, and aggregating reduces variance.

Breiman, Leo (2000). "Randomizing outputs to increase prediction accuracy". *Machine Learning*, vol. 40, no. 3, pp. 229–242.

Algorithm 14.7: Random forest (Breiman 2001)

One of the most popular machine learning algorithms is random forest, where we combine bagging and decision trees. Instead of training a usual decision tree, we introduce yet another layer of randomization:

- during training, at each internal node where we need to split the training samples, we select a random subset of features and perform the splitting. A different subset of features is used at a different internal node.

Then we apply bagging and aggregate a bunch of the above "randomized" decision trees.

Breiman, Leo (2001). "Random Forest". *Machine Learning*, vol. 45, no. 1, pp. 5–32.

Algorithm 14.8: Hedging (Freund and Schapire 1997)

```
Algorithm: Hedging.
  Input: initial weight vector $\mathbf{w}_1 \in \mathbb{R}^n_{++}$, discount factor $\beta \in [0, 1]$
  Output: last weight vector $\mathbf{w}_{T+1}$
1 for $t = 1, 2, \ldots, T$ do
2     learner chooses probability vector $\mathbf{p}_t = \frac{\mathbf{w}_t}{\mathbf{1}^\top \mathbf{w}_t}$        // normalization
3     environment chooses loss vector $\boldsymbol{\ell}_t \in [0, 1]^n$        // $\boldsymbol{\ell}_t$ may depend on $\mathbf{p}_t$!
4     learner suffers (expected) loss $\langle \mathbf{p}_t, \boldsymbol{\ell}_t \rangle$
5     learner updates weights $\mathbf{w}_{t+1} = \mathbf{w}_t \odot \beta^{\boldsymbol{\ell}_t}$        // element-wise product $\odot$ and power
6     optional scaling: $\mathbf{w}_{t+1} \leftarrow c_{t+1}\mathbf{w}_{t+1}$        // $c_{t+1} > 0$ can be arbitrary
```

Imagine the following horse racing game[a]: There are n horses in a race, which we repeat for T rounds. On each round we bet a fixed amount of money, with p_{it} being the proportion we spent on horse i at the t-th round. At the end of round t we receive a loss $\ell_{it} \in [0, 1]$ that we suffer on horse i. Note that the losses can be completely arbitrary (e.g. no i.i.d. assumption), although for simplicity we assume they are bounded. How should we place our bets (i.e. p_{it}) to hedge our risk? Needless to say, we must decide the proportions $\mathbf{p}_t$ *before* we see the losses $\boldsymbol{\ell}_t$. On the other hand, the losses $\boldsymbol{\ell}_t$ can be completely adversarial (i.e. depend on $\mathbf{p}_t$).

The Hedging algorithm gives a reasonable (in fact in some sense optimal) strategy: basically if a horse i is doing well (badly) on round t, then we spend a larger (smaller) proportion $p_{i,t+1}$ on it in the next round $t + 1$. On a high level, this is similar to the idea behind perceptron (see Section 1).

Note that line 5 continues decreasing each entry in the weight vector (because both $\beta \in [0, 1]$ and $\ell \in [0, 1]$). To prevent the weights from underflowing, we can re-scale them by a positive number $c > 0$, as shown in the optional line 6. Clearly, this scaling does not change $\mathbf{p}$ hence the algorithm (in any essential way).

Freund, Yoav and Robert E. Schapire (1997). "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting". *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139.

[a]Viewer discretion is advised; please do not try at home!

Alert 14.9: Multiplicative Updates

The Hedging algorithm belongs to the family of multiplicative updates, where we repeatedly multiplying, instead of adding (cf. Perceptron in Section 1), our weights by some correction terms. For multiplicative updates, it is important to start with strictly positive initializations, for a zero weight will remain as zero in multiplicative updates.

Theorem 14.10: Hedging Guarantee

For any nonempty $S \subseteq \{1, \ldots, n\}$, we have

$$L \le \frac{-\ln(\sum_{i\in S} p_{i1}) - (\ln \beta) \max_{i\in S} L_i}{1 - \beta},$$

where $L := \sum_{t=1}^{T} \langle \mathbf{p}_t, \boldsymbol{\ell}_t \rangle$ is our total (expected) loss and $L_i := \sum_{t=1}^{T} \ell_{it}$ is the total loss on the i-th horse.

Proof. We first lower bound the sum of weights at the end:

$$\begin{aligned}
\mathbf{1}^\top \mathbf{w}_{T+1} = \sum_{i=1}^{\mathsf{n}} w_{i,T+1} \geq \sum_{i \in S} w_{i,T+1} & \qquad \text{// weights remain nonnegative} \\
= \sum_{i \in S} w_{i,1} \beta^{L_i} & \qquad \text{// line 5 of Algorithm 14.8} \qquad (14.1) \\
\geq \beta^{\max_{i \in S} L_i} \sum_{i \in S} w_{i,1} & \qquad \text{// } \beta \in [0,1]. \qquad (14.2)
\end{aligned}$$

Then, we upper bound the sum of weights:

$$\begin{aligned}
\mathbf{1}^\top \mathbf{w}_{t+1} &= \sum_{i=1}^{\mathsf{n}} w_{i,t+1} = \sum_{i=1}^{\mathsf{n}} w_{i,t} \beta^{\ell_{i,t}} && \text{// line 5 of Algorithm 14.8} \\
&\leq \sum_{i=1}^{\mathsf{n}} w_{i,t}(1 - (1-\beta)\ell_{i,t}) && \text{// } x^\ell \leq 1^\ell + (x-1)\ell\text{: } x^\ell \text{ concave when } \ell \in [0,1] \text{ and } x \geq 0 \quad (14.3) \\
&= (\mathbf{1}^\top \mathbf{w}_t)[1 - (1-\beta)\mathbf{p}_t^\top \boldsymbol{\ell}_t] && \text{// line 2 of Algorithm 14.8.}
\end{aligned}$$

Thus, by telescoping:

$$\frac{\mathbf{1}^\top \mathbf{w}_{T+1}}{\mathbf{1}^\top \mathbf{w}_1} \leq \prod_{t=1}^{T} [1 - (1-\beta)\mathbf{p}_t^\top \boldsymbol{\ell}_t] \leq \exp\left[-(1-\beta)\sum_{t=1}^{T} \mathbf{p}_t^\top \boldsymbol{\ell}_t\right] = \exp[-(1-\beta)L], \qquad (14.4)$$

where we have used the elementary inequality $1 - x \leq \exp(-x)$ for any x.

Finally, combining the inequalities (14.2) and (14.4) completes the proof. □

By inspecting the proof closely, we realize that the same conclusion still holds if we change the update to

$$\mathbf{w}_{t+1} = \mathbf{w}_t \odot U_\beta(\boldsymbol{\ell}_t)$$

as long as the function U_β satisfies

$$\beta^\ell \leq U_\beta(\ell) \leq 1 - (1-\beta)\ell$$

(so that inequalities (14.1) and (14.3) still hold).

Exercise 14.11: Optional re-scaling has no effect

Show that the same conclusion in Theorem 14.10 still holds even if we include the optional line 6 in Algorithm 14.8.

Corollary 14.12: Comparable to best "horse" in hindsight

If we choose $\mathbf{w}_1 = \mathbf{1}$ *and* $|S| = 1$*, then*

$$\sum_{t=1}^{T} \langle \mathbf{p}_t, \boldsymbol{\ell}_t \rangle \leq \frac{\min_i L_i \ln \frac{1}{\beta} + \ln \mathsf{n}}{1-\beta}. \qquad (14.5)$$

In addition, if we choose $\beta = \frac{1}{1+\sqrt{(2m)/U}}$, *where* $U \geq L_{\min} := \min_i L_i$ *and* $m \geq \ln \mathsf{n}$, *then*

$$\frac{1}{T}\sum_{t=1}^{T}\langle \mathbf{p}_t, \boldsymbol{\ell}_t\rangle \leq \frac{L_{\min}}{T} + \frac{\sqrt{2mU}}{T} + \frac{\ln \mathsf{n}}{T}. \tag{14.6}$$

If we also choose $m = \ln \mathsf{n}$ *and* $U = T$ *(in our choice of* β*), then*

$$\frac{1}{T}\sum_{t=1}^{T}\langle \mathbf{p}_t, \boldsymbol{\ell}_t\rangle \leq \frac{L_{\min}}{T} + \sqrt{\frac{2\ln \mathsf{n}}{T}} + \frac{\ln \mathsf{n}}{T}. \tag{14.7}$$

Proof. Indeed, for $\beta \in [0,1]$ we have

$$\begin{aligned} 2 \leq 2/\beta &\implies 2\beta \geq 2\ln\beta + 2 && // \ 2\beta - 2\ln\beta - 2 \text{ is decreasing and nonnegative at } \beta = 1 \\ &\implies \beta^2 \leq 1 + 2\beta\ln\beta && // \ \beta^2 - 2\beta\ln\beta - 1 \text{ is increasing and nonpositive at } \beta = 1 \\ &\iff \ln\tfrac{1}{\beta} \leq \tfrac{1-\beta^2}{2\beta}. \end{aligned}$$

Plugging our choice of β into (14.5) and apply the above inequality we obtain (14.6). The bound (14.7) holds because we can choose $U = T$ (recall that we assume $\ell_{i,t} \in [0,1]$ for all t and i). □

Based on a result due to Vovk (1998), Freund and Schapire (1997) proved that the coefficients in (14.5) (i.e. $\frac{-\ln\beta}{1-\beta}$ and $\frac{\ln \mathsf{n}}{1-\beta}$) cannot be simultaneously improved for any β, using any algorithm (not necessarily Hedging).

In our last setting, $\beta = \frac{1}{1+\sqrt{(2\ln \mathsf{n})/T}}$, indicating for a longer game (larger T) we should use a larger β (to discount less aggressively) while for more horses (larger n) we should do the opposite (although this effect is much smaller due to the log).

Vovk, Valadimir (1998). "A Game of Prediction with Expert Advice". *Journal of Computer and System Sciences*, vol. 56, no. 2, pp. 153–173.

Freund, Yoav and Robert E. Schapire (1997). "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting". *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139.

Remark 14.13: Appreciating the significance of Hedging

Corollary 14.12 implies that the *average* loss of Hedging on the left-hand side of (14.7) is no larger than the average loss of the "best horse", plus a constant term proportional to $\sqrt{\frac{2\ln \mathsf{n}}{T}}$ (where we omit the higher order term $\frac{\ln \mathsf{n}}{T}$). As the number of rounds T goes to infinity, Hedging can compete against the best horse in hindsight! However, we emphasize three important points:

- It does not mean Hedging will achieve small loss, simply because the best horse may itself achieve a *large* average loss, in which case being competitive against the best horse does not really mean much.
- The loss vectors $\boldsymbol{\ell}_t$ can be adversarial against the Hedging algorithm! However, if the environment always tries to "screw up" the Hedging algorithm, then the guarantee in Corollary 14.12 implies that the environment inevitably also "screws up" the best horse.
- If the best horse can achieve very small loss, i.e. $L_{\min} \approx 0$, then by setting $U \approx 0$ we know from (14.6) that Hedging is off by at most a term proportional to $\frac{\ln \mathsf{n}}{T}$, which is much smaller than the dominating term $\sqrt{\frac{2\ln \mathsf{n}}{T}}$ in (14.7).

Remark 14.14: History of Boosting

Schapire (1990) first formally proved the possibility to combine a few mediocre classifiers into a very accurate one, which was subsequently improved in (Freund 1995) and eventually in (Freund and Schapire 1997), which has since become the main source of the celebrated Adaboost algorithm. Freund and Shapire received the Gödel prize in 2003 for this seminal contribution.

Schapire, Robert E. (1990). "The strength of weak learnability". *Machine Learning*, vol. 5, no. 2, pp. 197–227.

Freund, Y. (1995). "Boosting a Weak Learning Algorithm by Majority". *Information and Computation*, vol. 121, no. 2, pp. 256–285.

Freund, Yoav and Robert E. Schapire (1997). "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting". *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139.

Algorithm 14.15: Adaboost (Freund and Schapire 1997)

```
Algorithm: Adaptive Boosting.
  Input: initial weight vector $\mathbf{w}_1 \in \mathbb{R}^{\mathsf{n}}_{++}$, training set $\mathcal{D}_{\mathsf{n}} = \wr(\mathbf{x}_i, y_i)\wr_{i=1}^{\mathsf{n}} \subseteq \mathbb{R}^d \times \{0,1\}$
  Output: meta-classifier $\bar{h}: \mathbb{R}^d \to \{0,1\},\ \mathbf{x} \mapsto [\![\sum_{t=1}^T (\ln \frac{1}{\beta_t})(h_t(\mathbf{x}) - \frac{1}{2}) \geq 0]\!]$
1 for $t = 1, 2, \ldots, T$ do
2     $\mathbf{p}_t = \mathbf{w}_t / \mathbf{1}^\top \mathbf{w}_t$                    // normalization
3     $h_t \leftarrow \mathsf{WeakLearn}(\mathcal{D}_{\mathsf{n}}, \mathbf{p}_t)$          // $t$-th weak classifier $h_t: \mathbb{R}^d \to [0,1]$
4     $\forall i,\ \ell_{it} = 1 - |h_t(\mathbf{x}_i) - y_i|$     // loss is higher if prediction is more accurate!
5     $\epsilon_t = 1 - \langle \mathbf{p}_t, \boldsymbol{\ell}_t \rangle = \sum_{i=1}^{\mathsf{n}} p_{it} |h_t(\mathbf{x}_i) - y_i|$     // weighted (expected) error $\epsilon_t \in [0,1]$ of $h_t$
6     $\beta_t = \epsilon_t / (1 - \epsilon_t)$          // adaptive discounting parameter $\beta_t \leq 1 \iff \epsilon_t \leq \frac{1}{2}$
7     $\mathbf{w}_{t+1} = \mathbf{w}_t \odot \beta_t^{\boldsymbol{\ell}_t}$              // element-wise product $\odot$ and power
8     optional scaling: $\mathbf{w}_{t+1} \leftarrow c_{t+1} \mathbf{w}_{t+1}$         // $c_{t+1} > 0$ can be arbitrary
```

Provided that $\epsilon_t \leq \frac{1}{2}$ hence $\beta_t \in [0,1]$ (so the classifier h_t is better than random guessing), if h_t predicts *correctly* on a training example $\mathbf{x}_i$, then we suffer a *larger* loss ℓ_{ti} so that in the next iteration we assign *less* weight to $\mathbf{x}_i$. In other words, each classifier is focused on hard examples that are *misclassified* by the previous classifier. On the other hand, if $\epsilon_t > \frac{1}{2}$ then $\beta_t > 1$ and we do the opposite.

For the meta-classifier $\bar{h}$, we first perform a weighted aggregation of the confidences of individual (weak) classifiers and then threshold. Alternatively, we could threshold each weak classifier first and then perform (weighted) majority voting. The former approach, which we adopt here, is found to work better in practice. Note that, a classifier h_t with lower (training) error ϵ_t will be assigned a higher weight $\ln \frac{1}{\beta_t} = \ln(\frac{1}{\epsilon_t} - 1)$ in the final aggregation, making intuitive sense.

Freund, Yoav and Robert E. Schapire (1997). "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting". *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139.

Exercise 14.16: Implicitly fixing "bad" classifiers

If $\epsilon_t \geq \frac{1}{2}$ (hence $\beta_t \geq 1$), the t-th (weak) classifier is in fact worse than random guessing! In our binary setting here, a natural idea is to discard h_t but use instead $\tilde{h}_t := 1 - h_t$. Prove the following (all tilde quantities are w.r.t. the flipped classifier $\tilde{h}_t$):

- $\tilde{\boldsymbol{\ell}}_t = 1 - \boldsymbol{\ell}_t$
- $\tilde{\epsilon}_t = 1 - \epsilon_t$
- $\tilde{\beta}_t = 1/\beta_t$
- The Adaboost algorithm is not changed in any essential way even if we flip all "bad" classifiers. In particular, we arrive at the same meta-classifier.

In other words, Adaboost implicitly fixes all "bad" classifiers!

Theorem 14.17: Adaboost Guarantee

The following bound on the training error holds for the Adaboost Algorithm 14.15:

$$\sum_{i=1}^{n} p_{i1}[\![\bar{h}(\mathbf{x}_i) \neq y_i]\!] \leq \prod_{t=1}^{T} \sqrt{4\epsilon_t(1-\epsilon_t)}.$$

Proof. The proof closely follows what we have seen in the proof of Theorem 14.10. As before, we upper bound the sum of weights in the same manner:

$$\frac{\|\mathbf{w}_{T+1}\|_1}{\|\mathbf{w}_1\|_1} \leq \prod_{t=1}^{T}[1-(1-\beta_t)\mathbf{p}_t\boldsymbol{\ell}_t] = \prod_{t=1}^{T}[1-(1-\beta_t)(1-\epsilon_t)].$$

To lower bound the sum of weights, let $S = \{i : \bar{h}(\mathbf{x}_i) \neq y_i\}$, where recall that $\bar{h}$ is the meta-classifier constructed in Algorithm 14.15. Proceed as before:

$$\frac{\|\mathbf{w}_{T+1}\|_1}{\|\mathbf{w}_1\|_1} \geq \sum_{i\in S} \frac{w_{i,T+1}}{\|\mathbf{w}_1\|_1} = \sum_{i\in S} p_{i,1} \prod_{t=1}^{T} \beta_t^{\ell_{i,t}} \geq \sum_{i\in S} p_{i,1} \left[\prod_{t=1}^{T} \beta_t\right]^{1/2},$$

where the last inequality follows from the definition of S and the meta-classifier $\bar{h}$:

$$i \in S \implies 0 \geq \operatorname{sign}(2y_i - 1)\sum_t (\ln \tfrac{1}{\beta_t})(h_t(\mathbf{x}_i) - \tfrac{1}{2}) = \sum_t (\ln \tfrac{1}{\beta_t})(\tfrac{1}{2} - |h_t(\mathbf{x}_i) - y_i|) = \sum_t (\ln \tfrac{1}{\beta_t})(\ell_{i,t} - \tfrac{1}{2}).$$

Combine the upper bound and the lower bound:

$$\sum_{i\in S} p_{i,1} \prod_{t=1}^{T} \sqrt{\beta_t} \leq \prod_{t=1}^{T}[1-(1-\beta_t)(1-\epsilon_t)], \quad i.e., \quad \sum_{i\in S} p_{i,1} \leq \prod_{t=1}^{T} \frac{1-(1-\beta_t)(1-\epsilon_t)}{\sqrt{\beta_t}}.$$

Optimizing w.r.t. β_t we obtain $\beta_t = \epsilon_t/(1-\epsilon_t)$. Plugging it back in we complete the proof. □

Importantly, we observe that the training error of the meta-classifier $\bar{h}$ is upper bounded by the errors of all classifiers h_t: improving any individual classifier leads to a better bound. Moreover, the symmetry on the right-hand side confirms again that in the binary setting a very "inaccurate" classifier (i.e. large ϵ_t) is as good as a very accurate classifier (i.e. small ϵ_t).

Corollary 14.18: Exponential decay of training error

Assume $|\epsilon_t - \frac{1}{2}| > \gamma_t$, *then*

$$\sum_{i=1}^{n} p_{i1}[\![\bar{h}(\mathbf{x}_i) \neq y_i]\!] \leq \prod_{t=1}^{T} \sqrt{1-4\gamma_t^2} \leq \exp\left(-2\sum_{t=1}^{T}\gamma_t^2\right).$$

In particular, if $\gamma_t \geq \gamma$ *for all* t, *then*

$$\sum_{i=1}^{n} p_{i1}[\![\bar{h}(\mathbf{x}_i) \neq y_i]\!] \leq \exp(-2T\gamma^2).$$

□

Thus, to achieve ϵ (weighted) training error, we need to combine at most

$$T = \lceil \frac{1}{2\gamma^2} \ln \frac{1}{\epsilon} \rceil$$

weak classifiers, each of which is slightly better than random guessing (by a margin of γ).

Remark 14.19: Generalization Error of Adaboost

It is possible to bound the generalization error of Adaboost as well. For instance, Freund and Schapire (1997) bounded the VC dimension of the (family of) meta-classifiers constructed by Adaboost. A standard application of the VC theory then relates the generalization error with the training error. More refined analysis can be found in Schapire et al. (1998), Koltchinskii and Panchenko (2002), Koltchinskii et al. (2003), Koltchinskii and Panchenko (2005), Freund et al. (2004), and Rudin et al. (2007) (just to give a few pointers).

Freund, Yoav and Robert E. Schapire (1997). "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting". *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–139.

Schapire, Robert E., Yoav Freund, Peter Bartlett, and Wee Sun Lee (1998). "Boosting the margin: a new explanation for the effectiveness of voting methods". *The Annals of Statistics*, vol. 26, no. 5, pp. 1651–1686.

Koltchinskii, V. and D. Panchenko (2002). "Empirical Margin Distributions and Bounding the Generalization Error of Combined Classifiers". *The Annals of Statistics*, vol. 30, no. 1, pp. 1–50.

Koltchinskii, Vladimir, Dmitriy Panchenko, and Fernando Lozano (2003). "Bounding the generalization error of convex combinations of classifiers: balancing the dimensionality and the margins". *The Annals of Applied Probability*, vol. 13, no. 1, pp. 213–252.

Koltchinskii, Vladimir and Dmitry Panchenko (2005). "Complexities of convex combinations and bounding the generalization error in classification". *The Annals of Statistics*, vol. 33, no. 4, pp. 1455–1496.

Freund, Yoav, Yishay Mansour, and Robert E. Schapire (2004). "Generalization bounds for averaged classifiers". *The Annals of Statistics*, vol. 32, no. 4, pp. 1698–1722.

Rudin, Cynthia, Robert E. Schapire, and Ingrid Daubechies (2007). "Analysis of boosting algorithms using the smooth margin function". *The Annals of Statistics*, vol. 35, no. 6, pp. 2723–2768.

Alert 14.20: Does Ababoost Overfit? (Breiman 1999; Grove and Schuurmans 1998)

It has long been observed that Adaboost, even after decreasing the training error to zero, continues to improve test error. In other words, Adaboost does not seem to overfit even when we combine many many (weak) classifiers.

A popular explanation, due to Schapire et al. (1998), attributes Adaboost's resistance against overfitting to margin maximization: after decreasing the training error to 0, Adaboost continues to improve the margin (i.e. $yh(\mathbf{x})$), which leads to better generalization. However, Breiman (1999) and Grove and Schuurmans (1998) later designed the LPboost that explicitly maximizes the margin but observed inferior generalization.

Breiman, Leo (1999). "Prediction Games and Arcing Algorithms". *Neural Computation*, vol. 11, no. 7, pp. 1493–1517.

Grove, Adam J. and Dale Schuurmans (1998). "Boosting in the Limit: Maximizing the Margin of Learned Ensembles". In: *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, pp. 692–699.

Schapire, Robert E., Yoav Freund, Peter Bartlett, and Wee Sun Lee (1998). "Boosting the margin: a new explanation for the effectiveness of voting methods". *The Annals of Statistics*, vol. 26, no. 5, pp. 1651–1686.

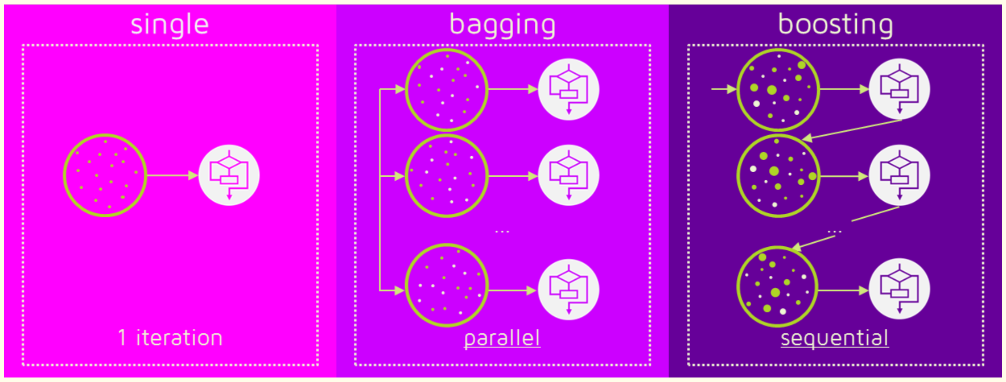
Algorithm 14.21: Face Detection (Viola and Jones 2004)

Viola and Jones (2004) applied the Adaboost algorithm to real-time face detection and are among the first few people who demonstrated the power of Adaboost in real challenging applications.

Viola, Paul and Michael J. Jones (2004). "Robust Real-Time Face Detection". *International Journal of Computer Vision*, vol. 57, no. 2, pp. 137–154.

Remark 14.22: Comparison

The following figure illustrates the similarity and difference between bagging and boosting:



To summarize:

- Both bagging and boosting train an ensemble of (weak) classifiers, which are combined in the end to produce a "meta-classifier";
- Bagging is amenable to parallelization while boosting is strictly sequential;
- Bagging resamples training examples while boosting reweighs training examples;
- As suggested in Breiman (2004), we can think of bagging as averaging *independent* classifiers (in order to reduce variance), while boosting averages dependent classifiers which may be analyzed through dynamic system and ergodic theory;
- We can of course combine bagging with boosting. For instance, each classifier in boosting can be obtained through bagging (called bag-boosting by Bühlmann and Yu, see the discussion of Friedman et al. (2000)).

Breiman, Leo (2004). "Population theory for boosting ensembles". *The Annals of Statistics*, vol. 32, no. 1, pp. 1–11.
Friedman, Jerome, Trevor Hastie, and Robert Tibshirani (2000). "Additive logistic regression: a statistical view of boosting (With discussion and a rejoinder by the authors)". *The Annals of Statistics*, vol. 28, no. 2, pp. 337–407.

Exercise 14.23: Diversity

Recall that $\epsilon_t(h) := \sum_{i=1}^{n} p_{it}|h(\mathbf{x}_i) - y_i|$ is the weighted error of classifier h at iteration t. Assume the (weak) classifiers are always binary-valued (and $\beta_t \in (0,1)$ for all t). Prove:

$$\epsilon_{t+1}(h_t) \equiv \tfrac{1}{2}.$$

In other words, Adaboost would never choose the same weak classifier twice in a row.

15 Expectation-Maximization (EM) and Mixture Models

Goal

Mixture models for density estimation and the celebrated expectation-maximization algorithm.

Alert 15.1: Convention

Gray boxes are not required hence can be omitted for unenthusiastic readers.

This note is likely to be updated again soon.

Definition 15.2: Density estimation

The central problem of this note is to estimate a density function (or more generally a probability measure), through a finite training sample. Formally, we are interested in estimating a probability measure χ from a (non)parametric family $\{\chi_\theta\}_{\theta\in\Theta}$. A typical approach is to minimize some statistical divergence (distance) between a noisy version $\hat\chi$ and χ_θ:

$$\inf_{\theta\in\Theta} \mathsf{D}(\hat\chi \| \chi_\theta).$$

However, the minimization problem above may not always be easy to solve, and alternative (indirect) strategies have been developed. As we show below, choosing the KL divergence corresponds to the maximum likelihood estimation procedure.

Definition 15.3: KL and LK

Recall that the Kullback-Leibler (KL) divergence between two density functiosn p and q is defined as:

$$\mathsf{KL}(p\|q) = \int p(\mathbf{x}) \log \frac{p(\mathbf{x})}{q(\mathbf{x})} \,\mathrm{d}\mathbf{x}.$$

Reverse the inputs we obtain the reverse KL divergence:

$$\mathsf{LK}(p\|q) := \mathsf{KL}(q\|p).$$

(In Section 18 we will generalize the above two divergences to a family of f-divergence.)

Definition 15.4: Entropy, conditional entropy, cross-entropy, and mutual information

We define the entropy of a random vector $\mathbf{X}$ with pdf p as:

$$\mathsf{H}(\mathbf{X}) := -\mathsf{E}\log p(\mathbf{X}) = -\int p(\mathbf{x}) \log p(\mathbf{x}) \,\mathrm{d}\mathbf{x},$$

the conditional entropy between $\mathbf{X}$ and $\mathbf{Z}$ (with pdf q) as:

$$\mathsf{H}(\mathbf{X}|\mathbf{Z}) := -\mathsf{E}\log p(\mathbf{X}|\mathbf{Z}) = -\int p(\mathbf{x},\mathbf{z}) \log p(\mathbf{x}|\mathbf{z}) \,\mathrm{d}\mathbf{x}\,\mathrm{d}\mathbf{z},$$

and the cross-entropy between $\mathbf{X}$ and $\mathbf{Z}$ as:

$$\dagger(\mathbf{X},\mathbf{Z}) := -\mathsf{E}\log q(\mathbf{X}) = -\int p(\mathbf{x}) \log q(\mathbf{x}) \,\mathrm{d}\mathbf{x}.$$

Finally, we define the mutual information between $\mathbf{X}$ and $\mathbf{Z}$ as:

$$\mathsf{I}(\mathbf{X},\mathbf{Z}) := \mathsf{KL}(p(\mathbf{x},\mathbf{z})\|p(\mathbf{x})q(\mathbf{z})) = \int p(\mathbf{x},\mathbf{z}) \log \frac{p(\mathbf{x},\mathbf{z})}{p(\mathbf{x})q(\mathbf{z})} \,\mathrm{d}\mathbf{x}\,\mathrm{d}\mathbf{z}$$

Exercise 15.5: Information theory

Verify the following:

$$\begin{aligned}
\mathsf{H}(\mathbf{X},\mathbf{Z}) &= \mathsf{H}(\mathbf{Z}) + \mathsf{H}(\mathbf{X}|\mathbf{Z}) \\
\dagger(\mathbf{X},\mathbf{Z}) &= \mathsf{H}(\mathbf{X}) + \mathsf{KL}(\mathbf{X}\|\mathbf{Z}) = \mathsf{H}(\mathbf{X}) + \mathsf{LK}(\mathbf{Z}\|\mathbf{X}) \\
\mathsf{I}(\mathbf{X},\mathbf{Z}) &= \mathsf{H}(\mathbf{X}) - \mathsf{H}(\mathbf{X}|\mathbf{Z}) \\
\mathsf{I}(\mathbf{X},\mathbf{Z}) &\geq 0, \text{ with equality iff } \mathbf{X} \text{ independent of } \mathbf{Z} \\
\mathsf{KL}(p(\mathbf{x},\mathbf{z})\|q(\mathbf{x},\mathbf{z})) &= \mathsf{KL}(p(\mathbf{z})\|q(\mathbf{z})) + \mathsf{E}[\mathsf{KL}(p(\mathbf{x}|\mathbf{z})\|q(\mathbf{x}|\mathbf{z}))].
\end{aligned}$$

All of the above can obviously be iterated to yield formula for more than two random vectors.

Exercise 15.6: Multivariate Gaussian

Compute

- the entropy of the multivariate Gaussian $\mathcal{N}(\boldsymbol{\mu},\Sigma)$;
- the KL divergence between two multivariate Gaussians $\mathcal{N}(\boldsymbol{\mu}_1,\Sigma_1)$ and $\mathcal{N}(\boldsymbol{\mu}_2,\Sigma_2)$.

Remark 15.7: MLE = KL minimization

Let us define the empirical "pdf" based on a dataset $\mathcal{D} = \wr \mathbf{x}_1,\ldots,\mathbf{x}_n \int$:

$$\hat{p}(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n} \delta_{\mathbf{x}_i},$$

where $\delta_{\mathbf{x}}$ is the "illegal" delta mass concentrated at $\mathbf{x}$. Then, we claim that

$$\theta_{\mathsf{MLE}} = \operatorname*{argmin}_{\theta\in\Theta} \ \mathsf{KL}\big(\hat{p}\|p(\mathbf{x}|\theta)\big).$$

Indeed, we have

$$\mathsf{KL}(\hat{p}\|p(\mathbf{x}|\theta)) = \int [\log(\hat{p}(\mathbf{x})) - \log p(\mathbf{x}|\theta)]\hat{p}(\mathbf{x})\,\mathrm{d}\mathbf{x} = C + \frac{1}{n}\sum_{i=1}^{n} -\log p(\mathbf{x}_i|\theta),$$

where C is a constant that does not depend on θ.

Exercise 15.8: Why KL is so special

To appreciate the uniqueness of the KL divergence, prove the following:

log is the only continuous function satisfying $f(st) = f(s) + f(t)$.

Algorithm 15.9: Expectation-Maximization (EM) (Dempster et al. 1977)

We formulate EM under the density estimation formulation in Definition 15.2, except that we carry out the procedure in a lifted space $\mathsf{X} \times \mathsf{Z}$ where Z is the space that some latent random variable $\mathbf{Z}$ lives in. Importantly, we do not observe the latent variable $\mathbf{Z}$: it is "artificially" constructed to aid our job. We fit our model with a prescribed family of joint distributions

$$\mu_\theta(\mathrm{d}\mathbf{x}, \mathrm{d}\mathbf{z}) = \zeta_\theta(\mathrm{d}\mathbf{z}) \cdot \mathfrak{D}_\theta(\mathrm{d}\mathbf{x}|\mathbf{z}) = \chi_\theta(\mathrm{d}\mathbf{x}) \cdot \mathfrak{E}_\theta(\mathrm{d}\mathbf{z}|\mathbf{x}), \quad \theta \in \Theta.$$

In EM, we typically specify the joint distribution μ_θ explicitly, and in a way that the posterior distribution $\mathfrak{E}_\theta(\mathrm{d}\mathbf{z}|\mathbf{x})$ can be easily computed. Similarly, we "lift" $\chi(\mathrm{d}\mathbf{x})$ (our target of estimation) to the joint distribution

$$\hat{\nu}(\mathrm{d}\mathbf{x}, \mathrm{d}\mathbf{z}) = \hat{\chi}(\mathrm{d}\mathbf{x}) \cdot \mathcal{E}(\mathrm{d}\mathbf{z}|\mathbf{x}).$$

(We use the hat notation to remind that we do not really have access to the true distribution χ but a sample from it, represented by the empirical distribution $\hat{\chi}$.) Then, we minimize the discrepancy between the joint distributions $\hat{\nu}$ and μ_θ, which is an *upper bound* of the discrepancy of the marginals $\mathsf{KL}(\hat{\chi}\|\chi_\theta)$ (Exercise 15.5):

$$\inf_{\theta\in\Theta} \inf_{\mathcal{E}(\mathrm{d}\mathbf{z}|\mathbf{x})} \mathsf{KL}(\hat{\nu}(\mathrm{d}\mathbf{x}, \mathrm{d}\mathbf{z})\|\mu_\theta(\mathrm{d}\mathbf{x}, \mathrm{d}\mathbf{z})).$$

Note that there is no restriction on $\mathcal{E}$ (and do not confuse it with $\mathfrak{E}_\theta$, which is "prescribed").

The EM algorithm proceeds with alternating minimization:

- (E-step) Fix θ_t, we solve $\mathcal{E}_{t+1}$ by (recall Exercise 15.5)

$$\inf_{\mathcal{E}} \mathsf{KL}(\hat{\nu}(\mathrm{d}\mathbf{x}, \mathrm{d}\mathbf{z})\|\mu_{\theta_t}(\mathrm{d}\mathbf{x}, \mathrm{d}\mathbf{z})) = \mathsf{KL}(\hat{\chi}\|\chi_{\theta_t}) + \mathsf{E}_{\hat{\chi}}\mathsf{KL}(\mathcal{E}\|\mathfrak{E}_{\theta_t}),$$

  which leads to the "closed-form" solution:

$$\mathcal{E}_{t+1} = \mathfrak{E}_{\theta_t}.$$

- (M-step) Fix $\mathcal{E}_{t+1}$, we solve θ_{t+1} by

$$\inf_{\theta\in\Theta} \mathsf{KL}(\hat{\nu}_{t+1}(\mathrm{d}\mathbf{x}, \mathrm{d}\mathbf{z})\|\mu_\theta(\mathrm{d}\mathbf{x}, \mathrm{d}\mathbf{z})) = \underbrace{\mathsf{KL}(\hat{\chi}\|\chi_\theta)}_{\text{likelihood}} + \underbrace{\mathsf{E}_{\hat{\chi}}\mathsf{KL}(\mathcal{E}_{t+1}\|\mathfrak{E}_\theta)}_{\text{regularizer}}.$$

  For the generalized EM algorithm, we need only decrease the above (joint) KL divergence if finding a (local) minima is expensive. It may be counter-intuitive that minimizing the sum of two terms above can be easier than minimizing the first likelihood term only!

Obviously, the EM algorithm monotonically decreases our (joint) KL divergence $\mathsf{KL}(\hat{\nu}, \mu_\theta)$. Moreover, thanks to construction, the EM algorithm also ascends the likelihood:

$$\mathsf{KL}(\hat{\chi}\|\chi_{\theta_{t+1}}) \le \mathsf{KL}(\hat{\nu}_{t+1}\|\mu_{\theta_{t+1}}) \le \mathsf{KL}(\hat{\nu}_{t+1}\|\mu_{\theta_t}) = \mathsf{KL}(\hat{\chi}\|\chi_{\theta_t}).$$

Dempster, A. P., N. M. Laird, and D. B. Rubin (1977). "Maximum Likelihood from Incomplete Data via the EM Algorithm". *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 39, no. 1, pp. 1–38.

Definition 15.10: Exponential family distribution

The exponential family distributions have the following density form:

$$\mathsf{p}(\mathbf{x}) = h(\mathbf{x}) \exp\big(\langle \boldsymbol{\eta}, T(\mathbf{x})\rangle - A(\boldsymbol{\eta})\big),$$

where $T(\mathbf{x})$ is the sufficient statistics, $\boldsymbol{\eta}$ is the natural parameter, A is the log-partition function, and h represents the base measure. Since p integrates to 1, we have

$$A(\boldsymbol{\eta}) = \log \int \exp\left(\langle \boldsymbol{\eta}, T(\mathbf{x})\rangle\right) \cdot h(\mathbf{x})\,\mathrm{d}\mathbf{x}$$

We can verify that A is a convex function.

Example 15.11: Gaussian distribution in exponential family

Recall that the multivariate Gaussian density is:

$$\begin{aligned}\mathsf{p}(\mathbf{x}) &= (2\pi)^{-d/2}[\det(\Sigma)]^{-1/2}\exp\left(-\tfrac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})\right)\\ &= \exp\left(\left\langle \begin{bmatrix}\mathbf{x}\\ -\frac{1}{2}\mathbf{x}\mathbf{x}^\top\end{bmatrix}, \begin{bmatrix}\Sigma^{-1}\boldsymbol{\mu}\\ \Sigma^{-1}\end{bmatrix}\right\rangle - \tfrac{1}{2}(\boldsymbol{\mu}^\top\Sigma^{-1}\boldsymbol{\mu} + d\log(2\pi) + \log\det\Sigma)\right).\end{aligned}$$

Thus, we identify

$$\begin{aligned}T(\mathbf{x}) &= (\mathbf{x}, -\tfrac{1}{2}\mathbf{x}\mathbf{x}^\top)\\ \boldsymbol{\eta} &= (\Sigma^{-1}\boldsymbol{\mu}, \Sigma^{-1}) =: (\boldsymbol{\xi}, S)\\ A(\boldsymbol{\mu}, \Sigma) &= \tfrac{1}{2}(\boldsymbol{\mu}^\top\Sigma^{-1}\boldsymbol{\mu} + d\log(2\pi) + \log\det\Sigma)\\ A(\boldsymbol{\eta}) &= A(\boldsymbol{\xi}, S) = \tfrac{1}{2}(\boldsymbol{\xi}^\top S^{-1}\boldsymbol{\xi} + d\log(2\pi) - \log\det S).\end{aligned}$$

Example 15.12: Bernoulli and Multinoulli in exponential family

The Bernoulli distribution is given as:

$$\begin{aligned}\mathsf{p}(z) &= \pi^z(1-\pi)^{1-z} = \exp\left(z\log\pi + (1-z)\log(1-\pi)\right)\\ &= \exp(z\log\tfrac{\pi}{1-\pi} + \log(1-\pi)).\end{aligned}$$

Thus, we may identify

$$\begin{aligned}T(z) &= z\\ \eta &= \log\tfrac{\pi}{1-\pi}\\ A(\eta) &= \log(1+\exp(\eta)).\end{aligned}$$

We can also consider the multinoulli distribution:

$$\begin{aligned}\mathsf{p}(\mathbf{z}) &= \prod_{k=1}^{c}\pi_k^{z_k} = \exp\left(\langle \mathbf{z}, \log\boldsymbol{\pi}\rangle\right)\\ &= \exp\left(\tilde{\mathbf{z}}^\top\log\tfrac{\tilde{\boldsymbol{\pi}}}{1-\langle\mathbf{1},\tilde{\boldsymbol{\pi}}\rangle} + \log(1-\langle\mathbf{1},\tilde{\boldsymbol{\pi}}\rangle)\right),\end{aligned}$$

where recall that $\mathbf{z} \in \{0,1\}^c$ is one-hot (i.e. $\mathbf{1}^\top\mathbf{z} = 1$), and we use the tilde notation to denote the subvector with the last entry removed. Thus, we may identify

$$\begin{aligned}T(\tilde{\mathbf{z}}) &= \tilde{\mathbf{z}}\\ \tilde{\boldsymbol{\eta}} &= \log\tfrac{\tilde{\boldsymbol{\pi}}}{1-\langle\mathbf{1},\tilde{\boldsymbol{\pi}}\rangle}\\ A(\tilde{\boldsymbol{\eta}}) &= \log(1+\langle\mathbf{1},\exp(\tilde{\boldsymbol{\eta}})\rangle).\end{aligned}$$

(Here, we use the tilde quantities to remove one redundancy since $\mathbf{1}^\top\mathbf{z} = \mathbf{1}^\top\boldsymbol{\pi} = 1$).

Exercise 15.13: Mean parameter and moments

Prove that for the exponential family distribution,

$$\nabla A(\boldsymbol{\eta}) = \mathsf{E}[T(\mathbf{X})]$$
$$\nabla^2 A(\boldsymbol{\eta}) = \mathsf{E}[T(\mathbf{X}) \cdot T(\mathbf{X})^\top] - \mathsf{E}[T(\mathbf{X})] \cdot \mathsf{E}[T(\mathbf{X})]^\top = \mathtt{Cov}(T(\mathbf{X})),$$

where the last equality confirms again that the log-partition function A is convex (since the covariance matrix is positive semidefinite).

Exercise 15.14: Marginal, conditional and product of exponential family

Let $\mathsf{p}(\mathbf{x}, \mathbf{z})$ be a joint distribution from the exponential family. Prove the following:

- The marginal $\mathsf{p}(\mathbf{x})$ need not be from the exponential family.
- The conditional $\mathsf{p}(\mathbf{z}|\mathbf{x})$ is again from the exponential family.
- The product of two exponential family distributions is again in exponential family.

Exercise 15.15: Exponential family approximation under KL

Let $\mathsf{p}(\mathbf{x})$ be an arbitrary distribution and $\mathsf{q}_{\boldsymbol{\eta}}(\mathbf{x})$ from the exponential family with sufficient statistics T, natural parameter $\boldsymbol{\eta}$ and log-partition function A. Then,

$$\boldsymbol{\eta}^* := \operatorname*{argmin}_{\boldsymbol{\eta}} \ \mathsf{KL}(\mathsf{p} \| \mathsf{q}_{\boldsymbol{\eta}})$$

is given by moment-matching:

$$\mathsf{E}_{\mathsf{p}} T(\mathbf{X}) = \mathsf{E}_{\mathsf{q}_{\boldsymbol{\eta}}} T(\mathbf{X}) = \nabla A(\boldsymbol{\eta}), \quad i.e., \quad \boldsymbol{\eta} = \nabla A^{-1}(\mathsf{E}_{\mathsf{p}} T(\mathbf{X})).$$

Exercise 15.16: EM for exponential family

Prove that the M-step of EM simplifies to the following, if we assume the joint distribution $\mu_{\boldsymbol{\eta}}$ is from the exponential family with natural parameter $\boldsymbol{\eta}$, sufficient statistics T and log-partition function A:

$$\boldsymbol{\eta}_{t+1} = \nabla A^{-1}(\mathsf{E}_{\hat{\nu}_{t+1}}(T(\mathbf{X}))).$$

Definition 15.17: Mixture Distribution

We define the joint distribution over a discrete latent random variable $Z \in \{1, \ldots, c\}$ and an observed random variable $\mathbf{X}$:

$$\mathsf{p}(\mathbf{x}, \mathbf{z}) = \prod_{k=1}^{c} [\pi_k \cdot \mathsf{p}_k(\mathbf{x}; \theta_k)]^{z_k},$$

where we represent $\mathbf{z}$ using one-hot encoding. We easily obtain the marginal and conditional:

$$\mathsf{p}(\mathbf{x}) = \sum_{k=1}^{c} \pi_k \cdot \mathsf{p}_k(\mathbf{x}; \theta_k) \qquad (15.1)$$

$$\mathsf{p}(\mathbf{z} = \mathbf{e}_k) = \pi_k$$
$$\mathsf{p}(\mathbf{x}|\mathbf{z} = \mathbf{e}_k) = \mathsf{p}_k(\mathbf{x}; \theta_k)$$
$$\mathsf{p}(\mathbf{z} = \mathbf{e}_k|\mathbf{x}) = \frac{\pi_k \cdot \mathsf{p}_k(\mathbf{x}; \theta_k)}{\sum_{j=1}^{c} \pi_j \cdot \mathsf{p}_j(\mathbf{x}; \theta_j)}.$$

The marginal distribution $\mathsf{p}(\mathbf{x})$ can be interpreted as follows: There are c component densities p_k. We choose a component p_k with probability π_k and then we sample $\mathbf{x}$ from the resulting component density. However, in reality we do not know which component density an observation $\mathbf{x}$ is sampled from, i.e., the discrete random variable $\mathbf{Z}$ is not observed (missing).

Let $\mathsf{p}_k(\mathbf{x}; \theta_k)$ be multivariate Gaussian (with θ_k denoting its mean and covariance) we get the popular Gaussian mixture model (GMM).

Algorithm 15.18: Mixture density estimation – MLE

Replacing the parameterization χ_θ with the mixture model in (15.1) we get a direct method for estimating the density function χ based on a sample:

$$\min_{\boldsymbol{\pi}\in\Delta, \boldsymbol{\theta}\in\boldsymbol{\Theta}} \mathsf{KL}(\hat{\chi}\|\mathsf{p}), \quad \mathsf{p}(\mathbf{x}) = \sum_{k=1}^{c} \pi_k \cdot \mathsf{p}_k(\mathbf{x}; \theta_k)$$

where Δ denotes the simplex constraint (i.e., $\boldsymbol{\pi} \geq \mathbf{0}$ and $\mathbf{1}^\top\boldsymbol{\pi} = 1$). The number of components c is a hyperparameter that needs to be determined *a priori*. We may apply (projected) gradient descent to solve $\boldsymbol{\pi}$ and $\boldsymbol{\theta}$. However, it is easy to verify that the objective function is nonconvex hence convergence to a reasonable solution may not be guaranteed.

We record the gradient here for later comparison. We use $\mathsf{p}_W(\mathbf{x}, \mathbf{z})$ for the joint density whose marginalization over the latent $\mathbf{z}$ gives $\mathsf{p}(\mathbf{x})$. For mixtures, the parameter W includes both $\boldsymbol{\pi}$ and $\boldsymbol{\theta}$.

$$\frac{\partial}{\partial W} = -\mathsf{E}_{\hat{\mathsf{p}}_W(\mathbf{z},\mathbf{x})} \frac{\partial \log \mathsf{p}_W(\mathbf{x}, \mathbf{z})}{\partial W}, \quad \text{where} \quad \hat{\mathsf{p}}_W(\mathbf{z}, \mathbf{x}) := \hat{\chi}(\mathrm{d}\mathbf{x}) \cdot \mathsf{p}_W(\mathbf{z}|\mathbf{x}).$$

Algorithm 15.19: Mixture density estimation – EM

Let us now apply the EM Algorithm 15.9 to the mixture density estimation problem. As mentioned before, we minimize the upper bound:

$$\min_{\boldsymbol{\pi}\in\Delta, \boldsymbol{\theta}\in\boldsymbol{\Theta}} \min_{\mathcal{E}} \mathsf{KL}(\hat{\nu}(\mathbf{x}, \mathbf{z})\|\mathsf{p}(\mathbf{x}, \mathbf{z})), \quad \hat{\nu}(\mathbf{x}, \mathbf{z}) = \hat{\chi}(\mathbf{x})\mathcal{E}(\mathbf{z}|\mathbf{x}), \quad \mathsf{p}(\mathbf{x}, \mathbf{z}) = \prod_{k=1}^{c} [\pi_k \cdot \mathsf{p}_k(\mathbf{x}; \theta_k)]^{z_k},$$

with the following two steps alternated until convergence:

- E-step: Fix $\boldsymbol{\pi}^{(t)}$ and $\boldsymbol{\theta}^{(t)}$, we solve

$$\mathcal{E}_{t+1} = \mathsf{p}^{(t)}(\mathbf{z} = \mathbf{e}_k|\mathbf{x}) = \frac{\pi_k^{(t)} \cdot \mathsf{p}_k(\mathbf{x}; \theta_k^{(t)})}{\sum_{j=1}^{c} \pi_j^{(t)} \cdot \mathsf{p}_j(\mathbf{x}; \theta_j^{(t)})} =: r_k^{(t+1)}(\mathbf{x}). \tag{15.2}$$

- M-step: Fix $\mathcal{E}_{t+1}$ hence $\hat{\nu}_{t+1}$, we solve

$$\begin{aligned}\min_{\boldsymbol{\pi}\in\Delta} \min_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} \mathsf{KL}(\hat{\nu}_{t+1}(\mathbf{x}, \mathbf{z})\|\mathsf{p}(\mathbf{x}, \mathbf{z})) \quad &\equiv \quad \max_{\boldsymbol{\pi}\in\Delta} \max_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} \mathsf{E}_{\hat{\chi}}\mathsf{E}_{\mathcal{E}_{t+1}} [\langle \mathbf{z}, \log \boldsymbol{\pi}\rangle + \langle \mathbf{z}, \log \mathbf{p}(\mathbf{X}; \boldsymbol{\theta})\rangle] \\ &= \quad \max_{\boldsymbol{\pi}\in\Delta} \max_{\boldsymbol{\theta}\in\boldsymbol{\Theta}} \mathsf{E}_{\hat{\chi}} \left[\left\langle \mathbf{r}^{(t+1)}(\mathbf{X}), \log \boldsymbol{\pi}\right\rangle + \left\langle \mathbf{r}^{(t+1)}(\mathbf{X}), \log \mathbf{p}(\mathbf{X}; \boldsymbol{\theta})\right\rangle\right].\end{aligned}$$

It is clear that the optimal

$$\boldsymbol{\pi}^{(t+1)} = \mathsf{E}_{\hat{\chi}}\mathbf{r}^{(t+1)}(\mathbf{X}) = \sum_{i=1}^{n} \hat{\chi}(\mathbf{x}_i) \cdot \mathbf{r}^{(t+1)}(\mathbf{x}_i). \tag{15.3}$$

(For us the empirical training distribution $\hat{\chi} \equiv \frac{1}{n}$, although we prefer to keep everything abstract and general.)

The θ_k's can be solved independently:

$$\max_{\theta_k \in \Theta_k} \mathsf{E}_{\hat{\chi}} \left[r_k^{(t+1)}(\mathbf{X}) \cdot \log \mathsf{p}_k(\mathbf{X}; \theta_k) \right] \quad \equiv \quad \min_{\theta_k \in \Theta_k} \mathsf{KL}(\hat{\chi}_k^{(t+1)} \| \mathsf{p}_k(\cdot; \theta_k)), \tag{15.4}$$

where we define $\hat{\chi}_k^{(t+1)} \propto \hat{\chi} \cdot r_k^{(t+1)}$. (This is similar to Adaboost, where we reweigh the training examples!)

If we choose the component density $\mathsf{p}_k(\mathbf{x}; \theta_k)$ from the exponential family with sufficient statistics T_k and log-partition function A_k, then from Exercise 15.15 we know (15.4) can be solved in closed-form:

$$\theta_k^{(t+1)} = \nabla A_k^{-1} \left[\mathsf{E}_{\hat{\chi}_k^{(t+1)}} T_k(\mathbf{X}) \right] \tag{15.5}$$

Alert 15.20: implicit EM vs. explicit ML

We now make an important connection between EM and ML. We follow the notation in Algorithm 15.18. For the joint density $\mathsf{p}_W(\mathbf{x}, \mathbf{z})$, EM solves

$$W_{t+1} = \operatorname*{argmin}_{W} \ \mathsf{KL}(\hat{\mathsf{p}}_{t+1} \| \mathsf{p}_W), \qquad \text{where} \quad \hat{\mathsf{p}}_{t+1}(\mathbf{x}, \mathbf{z}) := \hat{\mathsf{p}}_{W_t}(\mathbf{x}, \mathbf{z}) = \hat{\chi}(\mathrm{d}\mathbf{x}) \cdot \mathsf{p}_{W_t}(\mathbf{z}|\mathbf{x}).$$

In particular, at a minimizer W_{t+1} the gradient vanishes:

$$-\mathsf{E}_{\hat{\mathsf{p}}_{t+1}} \frac{\partial \log \mathsf{p}_W(\mathbf{x}, \mathbf{z})}{\partial W} = \mathbf{0}.$$

In other words, EM solves the above nonlinear equation (in W) to get W_{t+1} while ML with gradient descent simply performs one fixed-point iteration.

Example 15.21: Gaussian Mixture Model (GMM) – EM

Using the results in multivariate Gaussian Example 15.11 we derive:

$$\nabla A(\boldsymbol{\eta}) = \begin{bmatrix} S^{-1}\boldsymbol{\xi} \\ -\frac{1}{2}(S^{-1}\boldsymbol{\xi}\boldsymbol{\xi}^\top S^{-1} + S^{-1}) \end{bmatrix} = \begin{bmatrix} \boldsymbol{\mu} \\ -\frac{1}{2}(\boldsymbol{\mu}\boldsymbol{\mu}^\top + \Sigma) \end{bmatrix},$$

$$\mathsf{E}_{\hat{\chi}^{(t+1)}} T(\mathbf{X}) = \begin{bmatrix} \mathsf{E}_{\hat{\chi}^{(t+1)}} \mathbf{X} \\ -\frac{1}{2}\mathsf{E}_{\hat{\chi}^{(t+1)}} \mathbf{X}\mathbf{X}^\top \end{bmatrix} \propto \sum_{i=1}^{n} (\hat{\chi} \cdot r^{(t+1)})(\mathbf{x}_i) \begin{bmatrix} \mathbf{x}_i \\ -\frac{1}{2}\mathbf{x}_i\mathbf{x}_i^\top \end{bmatrix},$$

where we have omitted the component subscript k. Thus, from (15.5) we obtain:

$$\boldsymbol{\mu}_k^{(t+1)} = \mathsf{E}_{\hat{\chi}_k^{(t+1)}} \mathbf{X} = \sum_{i=1}^{n} \frac{(\hat{\chi} \cdot r_k^{(t+1)})(\mathbf{x}_i)}{\sum_{\iota=1}^{n} (\hat{\chi} \cdot r_k^{(t+1)})(\mathbf{x}_\iota)} \cdot \mathbf{x}_i \tag{15.6}$$

$$\Sigma_k^{(t+1)} = \mathsf{E}_{\hat{\chi}_k^{(t+1)}} \mathbf{X}\mathbf{X}^\top - \boldsymbol{\mu}_k^{(t+1)} \boldsymbol{\mu}_k^{(t+1)\top} = \sum_{i=1}^{n} \frac{(\hat{\chi} \cdot r_k^{(t+1)})(\mathbf{x}_i)}{\sum_{\iota=1}^{n} (\hat{\chi} \cdot r_k^{(t+1)})(\mathbf{x}_\iota)} \cdot (\mathbf{x}_i - \boldsymbol{\mu}_k^{(t+1)})(\mathbf{x}_i - \boldsymbol{\mu}_k^{(t+1)})^\top, \tag{15.7}$$

where we remind that the empirical training distribution $\hat{\chi} \equiv \frac{1}{n}$.

The updates on "responsibility" $\mathbf{r}$ in (15.2), mixing distribution $\boldsymbol{\pi}$ in (15.3), on the means $\boldsymbol{\mu}_k$ in (15.6), and on the covariance matrices S_k in (15.7), consist of the main steps for estimating a GMM using EM.

16 Restricted Boltzmann Machine (RBM)

Goal

Gibbs sampling, Boltzmann Machine and Restricted Boltzmann Machine.

Alert 16.1: Convention

Gray boxes are not required hence can be omitted for unenthusiastic readers.

This note is likely to be updated again soon.

Algorithm 16.2: The Metropolis-Hastings algorithm (Metropolis et al. 1953; Hastings 1970)

Suppose we want to take a sample from a density p, where direct sampling is costly. Instead, we resort to an iterative algorithm:

```
Algorithm: The Metropolis-Hastings Algorithm
   Input: proposal (conditional) density $\mathsf{q}(\mathbf{y}|\mathbf{x})$, symmetric function $s(\mathbf{x},\mathbf{y})$, target density $\mathsf{p}(\mathbf{x})$
   Output: approximate sample $\mathbf{X} \sim \mathsf{p}$
1  choose $\mathbf{X}$
2  repeat
3      sample $\mathbf{Y} \sim \mathsf{q}(\cdot|\mathbf{X})$
4      $\alpha(\mathbf{X},\mathbf{Y}) \leftarrow \frac{s(\mathbf{X},\mathbf{Y})}{1+\frac{\mathsf{p}(\mathbf{X})\mathsf{q}(\mathbf{Y}|\mathbf{X})}{\mathsf{p}(\mathbf{Y})\mathsf{q}(\mathbf{X}|\mathbf{Y})}} = s(\mathbf{X},\mathbf{Y})\frac{\mathsf{p}(\mathbf{Y})\mathsf{q}(\mathbf{X}|\mathbf{Y})}{\mathsf{p}(\mathbf{Y})\mathsf{q}(\mathbf{X}|\mathbf{Y})+\mathsf{p}(\mathbf{X})\mathsf{q}(\mathbf{Y}|\mathbf{X})}$
5      with probability $\alpha(\mathbf{X},\mathbf{Y})$: $\mathbf{X} \leftarrow \mathbf{Y}$
6  until until convergence
```

Obviously, the symmetric function s must be chosen so that $\alpha \in [0,1]$. Popular choices include:

- Metropolis-Hastings (Hastings 1970):

$$s(\mathbf{x},\mathbf{y}) = \frac{\mathsf{p}(\mathbf{x})\mathsf{q}(\mathbf{y}|\mathbf{x}) + \mathsf{p}(\mathbf{y})\mathsf{q}(\mathbf{x}|\mathbf{y})}{\mathsf{p}(\mathbf{x})\mathsf{q}(\mathbf{y}|\mathbf{x}) \vee \mathsf{p}(\mathbf{y})\mathsf{q}(\mathbf{x}|\mathbf{y})} \implies \alpha(\mathbf{x},\mathbf{y}) = 1 \wedge \frac{\mathsf{p}(\mathbf{y})\mathsf{q}(\mathbf{x}|\mathbf{y})}{\mathsf{p}(\mathbf{x})\mathsf{q}(\mathbf{y}|\mathbf{x})} \tag{16.1}$$

- Barker (Barker 1965):

$$s(\mathbf{x},\mathbf{y}) = 1 \implies \alpha(\mathbf{x},\mathbf{y}) = \frac{\mathsf{p}(\mathbf{y})\mathsf{q}(\mathbf{x}|\mathbf{y})}{\mathsf{p}(\mathbf{y})\mathsf{q}(\mathbf{x}|\mathbf{y}) + \mathsf{p}(\mathbf{x})\mathsf{q}(\mathbf{y}|\mathbf{x})}$$

The algorithm simplifies considerably if the proposal q is symmetric, i.e. $\mathsf{q}(\mathbf{x}|\mathbf{y}) = \mathsf{q}(\mathbf{y}|\mathbf{x})$, which is the original setting in (Metropolis et al. 1953):

```
Algorithm: The Symmetric Metropolis-Hastings Algorithm
   Input: symmetric proposal density $\mathsf{q}(\mathbf{y}|\mathbf{x})$, symmetric function $s(\mathbf{x},\mathbf{y})$, target density $\mathsf{p}(\mathbf{x})$
   Output: approximate sample $\mathbf{X} \sim \mathsf{p}$
1  choose $\mathbf{X}$
2  repeat
3      sample $\mathbf{Y} \sim \mathsf{q}(\cdot|\mathbf{X})$
4      $\alpha(\mathbf{X},\mathbf{Y}) \leftarrow s(\mathbf{X},\mathbf{Y})\frac{\mathsf{p}(\mathbf{Y})}{\mathsf{p}(\mathbf{Y})+\mathsf{p}(\mathbf{X})}$
5      with probability $\alpha(\mathbf{X},\mathbf{Y})$: $\mathbf{X} \leftarrow \mathbf{Y}$
6  until until convergence
```

For MH's rule (16.1), we now have

$$s(\mathbf{x},\mathbf{y}) = \frac{\mathsf{p}(\mathbf{x})}{\mathsf{p}(\mathbf{y})} \wedge \frac{\mathsf{p}(\mathbf{y})}{\mathsf{p}(\mathbf{x})} \implies \alpha(\mathbf{x},\mathbf{y}) = 1 \wedge \frac{\mathsf{p}(\mathbf{y})}{\mathsf{p}(\mathbf{x})}$$

while Barker's rule (6) reduces to

$$s(\mathbf{x},\mathbf{y}) = 1 \implies \alpha(\mathbf{x},\mathbf{y}) = \frac{\mathsf{p}(\mathbf{y})}{\mathsf{p}(\mathbf{y}) + \mathsf{p}(\mathbf{x})}.$$

In particular, if $\mathsf{p}(\mathbf{Y}) \geq \mathsf{p}(\mathbf{X})$, then MH always moves to the new position $\mathbf{Y}$ while Barker's rule may still reject and repeat over.

Metropolis, Nicholas, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller (1953). "Equation of state calculations by fast computing machines". *Journal of Chemical Physics*, vol. 21, pp. 1087–1092.

Hastings, W. Keith (1970). "Monte Carlo sampling methods using Markov chains and their applications". *Biometrika*, vol. 57, pp. 97–109.

Barker, A. A. (1965). "Monte Carlo calculations of the radial distribution functions for a proton-electron plasma". *Australian Journal of Physics*, vol. 18, no. 2, pp. 119–134.

Alert 16.3: Significance of MH

To appreciate the significance of MH, let us point out that:

- There is immense flexibility in choosing the proposal q!
- We need only know the target density p up to a constant!

Both are crucial for our application to (restricted) Boltzmann machines, as we will see.

Algorithm 16.4: Gibbs sampling (Hastings 1970; Geman and Geman 1984)

If we choose the proposal density q so that $\mathsf{q}(\mathbf{y}|\mathbf{x}) \neq 0$ only if the new position $\mathbf{y}$ and the original position $\mathbf{x}$ do not differ much (e.g. agree on all but 1 coordinate), then we obtain the so-called Gibbs sampler. Variations include:

- randomized: randomly choose a (block of) coordinate(s) j in $\mathbf{x}$ and change it (them) according to q_j.
- cyclic: loop over each (block of) coordinate(s) j in $\mathbf{x}$ and change it (them) according to q_j.

If we choose $\mathsf{q}(\mathbf{y}|\mathbf{x}) = \mathsf{p}(\mathbf{y}|\mathbf{x})$, then for MH's rule $\alpha \equiv 1$ while for Barker's rule $\alpha \equiv \frac{1}{2}$.

Hastings, W. Keith (1970). "Monte Carlo sampling methods using Markov chains and their applications". *Biometrika*, vol. 57, pp. 97–109.

Geman, Stuart and Donald Geman (1984). "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 6, no. 6, pp. 721–741.

Remark 16.5: Optimality of MH

Peskun (1973) showed that the MH rule is optimal in terms of asymptotic variance.

Peskun, P. H. (1973). "Optimum Monte-Carlo Sampling Using Markov Chains". *Biometrika*, vol. 60, no. 3, pp. 607–612.

Definition 16.6: Boltzmann distribution (e.g. Hopfield 1982)

We say a (discrete) random variable $\mathbf{S} \in \{\pm 1\}^m$ follows a Boltzmann distribution p iff there exists a symmetric matrix $W \in \mathbb{S}^{m+1}$ such that

$$\forall \mathbf{s} \in \{\pm 1\}^m, \quad \mathsf{p}_W(\mathbf{S} = \mathbf{s}) = \exp(\mathbf{s}^\top W \mathbf{s} - A(W)), \quad \text{where} \quad A(W) = \log \sum_{\mathbf{s} \in \{\pm 1\}^m} \exp(\mathbf{s}^\top W \mathbf{s}) \tag{16.2}$$

is the log-partition function. It is clear that Boltzmann distributions belong to the exponential family Definition 15.10, with sufficient statistics

$$T(\mathbf{s}) = \mathbf{s}\mathbf{s}^\top.$$

We remind that we have appended the constant 1 in $\mathbf{s}$ so that W contains the bias term too.

Hopfield, John J. (1982). "Neural networks and physical systems with emergent collective computational abilities". *Proceedings of the National Academy of Sciences*, vol. 79, no. 8, pp. 2554–2558.

Alert 16.7: Coding convention

We used the encoding $\{\pm 1\}$ to represent a binary value above. As a consequence, the diagonal entries in W only contribute a constant (independent of the realization $\mathbf{s}$) in (16.2). Thus, w.l.o.g. we may absorb $\text{diag}(W)$ into $A(W)$ so that $\text{diag}(W) = \mathbf{0}$.

On the other hand, if we use the encoding $\{0, 1\}$, while conceptually being equivalent, we will no longer need to perform padding, since the bias term can now be stored in the diagonal of W.

Alert 16.8: Intractability of Boltzmann distributions

Despite the innocent form (16.2), Boltzmann distributions are in general intractable (for large m), since the log-partition function involves summation over 2^m terms (Long and Servedio 2010). This is common in Bayesian analysis where we know a distribution only up to an intractable normalization constant.

Long, Philip M. and Rocco A. Servedio (2010). "Restricted Boltzmann Machines Are Hard to Approximately Evaluate or Simulate". In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*, pp. 703–710.

Example 16.9: m=1 reduces to (binary) logistic

For $m = 1$ we have

$$\mathsf{p}_W(S = 1) = \frac{\exp(2w_{12})}{\exp(2w_{12}) + \exp(-2w_{12})} = \mathtt{sgm}(w), \quad \text{where} \quad w := 4w_{12}$$

and recall the sigmoid function $\mathtt{sgm}(t) = \frac{1}{1+\exp(-t)}$.

This example confirms that even if we can choose any W, the resulting set of Boltzmann distributions forms a strict subset of all discrete distributions over the cube $\{\pm 1\}^m$.

Definition 16.10: Boltzmann machine (BM) (Ackley et al. 1985; Hinton and Sejnowski 1986)

Now let us partition the Boltzmann random variable $\mathbf{S}$ into the concatenation of an observed random variable $\mathbf{X} \in \{\pm 1\}^d$ and a latent random variable $\mathbf{Z} \in \{\pm 1\}^t$. We call the marginal distribution over $\mathbf{X}$ a Boltzmann machine. Note that $\mathbf{X}$ no longer belongs to the exponential family!

Given a sample $\mathbf{X}_1, \ldots, \mathbf{X}_n$, we are interested in learning the Boltzmann machine, namely the marginal density $\mathsf{p}_W(\mathbf{x})$. We achieve this goal by learning the symmetric matrix W that defines the joint Boltzmann distribution $\mathsf{p}_W(\mathbf{s}) = \mathsf{p}_W(\mathbf{x}, \mathbf{z})$ in (16.2).

Ackley, David H., Geoffrey E. Hinton, and Terrence J. Sejnowski (1985). "A learning algorithm for boltzmann machines". *Cognitive Science*, vol. 9, no. 1, pp. 147–169.

Hinton, Geoffrey E. and T. J. Sejnowski (1986). "Learning and Relearning in Boltzmann Machines". In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations*. Ed. by David E. Rumelhart, James L. McClelland, and the PDP Research Group. The MIT Press, pp. 282–317.

Definition 16.11: Restricted Boltzmann Machine (RBM) (Smolensky 1986)

Let us consider the partition of the symmetric matrix

$$W = \begin{bmatrix} W_{xx} & W_{xz} \\ W_{xz}^\top & W_{zz} \end{bmatrix}.$$

If we require $W_{xx} = \mathbf{0}$ and $W_{zz} = \mathbf{0}$, then we obtain the restricted Boltzmann machine:

$$\mathsf{p}_W(\mathbf{x}, \mathbf{z}) \propto \exp\left(\mathbf{x}^\top W_{xz} \mathbf{z}\right), \tag{16.3}$$

i.e., only cross products are allowed.

Similarly, we will consider learning RBM through estimating the (rectangular) matrix $W_{xz} \in \mathbb{R}^{(d+1)\times(t+1)}$.

Smolensky, Paul (1986). "Information Processing in Dynamical Systems: Foundations of Harmony Theory". In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition. Volume 1: Foundations*. The MIT Press, pp. 194–281.

Example 16.12: $m = 1, t = 1$

Let $m = 1$ and $t = 1$. We have for (R)BM:

$$\mathsf{p}(X = x, Z = z) \propto \exp\left(2xzw_{12} + 2xw_{13} + 2zw_{23}\right)$$
$$\mathsf{p}(X = 1) \propto \exp\left(2w_{12} + 2w_{13} + 2w_{23}\right) + \exp\left(-2w_{12} + 2w_{13} - 2w_{23}\right),$$

In general, RBM is a strict subset of BM.

Remark 16.13: Representation power of (R)BM—the power of latent variables

Freund and Haussler (1992) and Neal (1992) are among the first to prove that RBM and BM can approximate any discrete distribution on $\{\pm 1\}^d$ arbitrarily well if the number t of latent variables is large (approaching 2^d). More refined results appeared later in (Le Roux and Bengio 2008; Le Roux and Bengio 2010; Montúfar and Ay 2011; Montúfar 2014).

In essence, when we marginalize out the latent variables in a (restricted) Boltzmann distribution, we create a mixture of many components on the remaining variables, hence the ability to approximate any discrete distribution.

Freund, Yoav and David Haussler (1992). "Unsupervised learning of distributions on binary vectors using two layer networks". In: *Advances in Neural Information Processing Systems 4*. Ed. by J. E. Moody, S. J. Hanson, and R. P. Lippmann, pp. 912–919.

Neal, Radford M. (1992). "Connectionist learning of belief networks". *Artificial Intelligence*, vol. 56, no. 1, pp. 71–113.

Le Roux, Nicolas and Yoshua Bengio (2008). "Representational Power of Restricted Boltzmann Machines and Deep Belief Networks". *Neural Computation*, vol. 20, no. 6, pp. 1631–1649.

— (2010). "Deep Belief Networks Are Compact Universal Approximators". *Neural Computation*, vol. 22, no. 8, pp. 2192–2207.

Montúfar, Guido and Nihat Ay (2011). "Refinements of Universal Approximation Results for Deep Belief Networks and Restricted Boltzmann Machines". *Neural Computation*, vol. 23, no. 5, pp. 1306–1319.

Montúfar, Guido F. (2014). "Universal Approximation Depth and Errors of Narrow Belief Networks with Discrete Units". *Neural Computation*, vol. 26, no. 7, pp. 1386–1407.

Remark 16.14: Computing the gradient

We now apply the same maximum likelihood idea in Algorithm 15.18 for learning an (R)BM:

$$\min_W \ \mathsf{KL}(\hat{\chi}(\mathbf{x})\|\mathsf{p}_W(\mathbf{x})) \qquad \equiv \qquad \min_W \ -\mathsf{E}_{\hat{\chi}} \log \sum_{\mathbf{z}\in\{\pm1\}^t} \mathsf{p}_W(\mathbf{X},\mathbf{z}).$$

Taking derivative w.r.t. W we obtain:

$$\begin{aligned}\frac{\partial}{\partial W} &= -\mathsf{E}_{\hat{\chi}} \frac{\sum_{\mathbf{z}} \frac{\partial \mathsf{p}_W(\mathbf{X},\mathbf{z})}{\partial W}}{\sum_{\mathbf{z}} \mathsf{p}_W(X,\mathbf{z})} \\ &= -\mathsf{E}_{\hat{\chi}} \sum_{\mathbf{z}} \mathsf{p}_W(\mathbf{z}|\mathbf{X}) \frac{\partial \log \mathsf{p}_W(\mathbf{X},\mathbf{z})}{\partial W} \\ &= -\mathsf{E}_{\hat{\chi}} \sum_{\mathbf{z}} \mathsf{p}_W(\mathbf{z}|\mathbf{X}) \frac{\partial(\mathsf{s}^\top W\mathsf{s} - A(W))}{\partial W} \\ &= -\mathsf{E}_{\hat{\chi}} \sum_{\mathbf{z}} \mathsf{p}_W(\mathbf{z}|\mathbf{X})\mathsf{s}\mathsf{s}^\top + \nabla A(W).\end{aligned}$$

Denoting $\hat{\mathsf{p}}_W(\mathbf{x},\mathbf{z}) = \hat{\chi}(\mathrm{d}\mathbf{x})\mathsf{p}_W(\mathbf{z}|\mathbf{x})$ and applying Exercise 15.13 we obtain the beautiful formula:

$$\frac{\partial}{\partial W} = -\mathsf{E}_{\hat{\mathsf{p}}_W}\mathsf{s}\mathsf{s}^\top + \mathsf{E}_{\mathsf{p}_W}\mathsf{s}\mathsf{s}^\top, \quad \text{where} \quad \mathsf{s} = (\mathbf{x};\mathbf{z};1).$$

The same result, with the restriction that $W_{xx} = \mathbf{0}$ and $W_{xz} = \mathbf{0}$, holds for RBM.

Therefore, we may apply (stochastic) gradient descent to find W, provided that we can evaluate the two expectations above. This is where we need the Gibbs sampling algorithm in Algorithm 16.4.

Alert 16.15: Failure of EM

Let us attempt to apply the EM Algorithm 15.9 for estimating W:

- E-step: $\mathcal{E}_{t+1}(\mathbf{z}|\mathbf{x}) = \mathsf{p}_{W_t}(\mathbf{z}|\mathbf{x})$.
- M-step: $W_{t+1} = \operatorname{argmin}_W \ \mathsf{KL}(\hat{\mathsf{p}}_{t+1}\|\mathsf{p}_W)$, where recall that $\hat{\mathsf{p}}_{t+1}(\mathbf{x},\mathbf{z}) := \hat{\chi}(\mathrm{d}\mathbf{x}) \cdot \mathsf{p}_{W_t}(\mathbf{z}|\mathbf{x})$.

It follows then from Exercise 15.15 (or more directly from Exercise 15.16) that

$$W_{t+1} = \nabla A^{-1}(\mathsf{E}_{\hat{\mathsf{p}}_{t+1}}T(\mathbf{X})), \quad \textit{i.e.} \quad \mathsf{E}_{\mathsf{p}_{t+1}}T(\mathbf{X}) = \mathsf{E}_{\hat{\mathsf{p}}_{t+1}}T(\mathbf{X}), \quad \text{where} \quad \mathsf{p}_{t+1} := \mathsf{p}_{W_{t+1}}. \tag{16.4}$$

However, we cannot implement (16.4) since the log-partition function A hence also its gradient ∇A is not tractable!

In fact, the gradient algorithm in Remark 16.14 is some explicit form that bypasses the difficulty in EM. Namely, to solve a nonlinear equation

$$\mathbf{f}(W) = 0, \qquad [\text{ for our case, } \mathbf{f}(W) = \mathsf{E}_{\mathsf{p}_W}T(\mathbf{X}) - \mathsf{E}_{\hat{\mathsf{p}}_{t+1}}T(\mathbf{X})]$$

we apply the fixed-point iteration:

$$W \leftarrow W - \eta \cdot \mathbf{f}(W),$$

which converges (if at all) iff $\mathbf{f}(W) = 0$.

Algorithm 16.16: Learning BM

To estimate $\mathsf{E}_{\mathsf{p}_W} \mathbf{s}\mathbf{s}^\top$, we need to be able to draw a sample $\mathbf{S} \sim \mathsf{p}_W$. This can be achieved by the Gibbs sampling Algorithm 16.4. Indeed, we know (recall that conditional of exponential family is again in exponential family, Exercise 15.14)

$$\mathsf{p}_W(S_j = s_j | \mathbf{S}_{\setminus j} = \mathbf{s}_{\setminus j}) \propto \exp\left(2s_j \left\langle W_{\setminus j,j}, \mathbf{s}_{\setminus j}\right\rangle\right), \quad i.e. \quad \mathsf{p}_W(S_j = s_j | \mathbf{S}_{\setminus j} = \mathbf{s}_{\setminus j}) = \mathtt{sgm}(4s_j \left\langle W_{\setminus j,j}, \mathbf{s}_{\setminus j}\right\rangle), \tag{16.5}$$

where the subscript $_{\setminus j}$ indicates removal of the j-th entry or row.

Algorithm: Gibbs sampling from Boltzmann distribution p_W

```
    Input: symmetric matrix $W \in \mathbb{S}^{m+1}$
    Output: approximate sample $\mathbf{s} \sim \mathsf{p}_W$
1   initialize $\mathbf{s} \in \{\pm 1\}^m$
2   repeat
3       for $j = 1, \ldots, m$ do
4           $p_j \leftarrow \mathtt{sgm}(4 \left\langle W_{\setminus j,j}, \mathbf{s}_{\setminus j}\right\rangle)$
5           with probability $p_j$ set $s_j = 1$, otherwise set $s_j = -1$
6   until until convergence
```

Similarly, to estimate $\mathsf{E}_{\hat{\mathsf{p}}_W} \mathbf{s}\mathbf{s}^\top$ we first draw a training sample $\mathbf{x} \sim \hat{\chi}$, and then draw $\mathbf{z} \sim \mathsf{p}_W(\cdot|\mathbf{x})$. For the latter, we fix $\mathbf{x}$ and apply the Gibbs sampling algorithm:

Algorithm: Gibbs sampling from conditional Boltzmann distribution $\mathsf{p}_W(\cdot|\mathbf{x})$

```
    Input: symmetric matrix $W \in \mathbb{S}^{m+1}$, training sample $\mathbf{x}$
    Output: approximate sample $\mathbf{z} \sim \mathsf{p}_W(\cdot|\mathbf{x})$
1   initialize $\mathbf{z} \in \{\pm 1\}^t$ and set $\mathbf{s} = (\mathbf{x}; \mathbf{z}; 1)$
2   repeat
3       for $j = d+1, \ldots, m$ do
4           $p_j \leftarrow \mathtt{sgm}(4 \left\langle W_{\setminus j,j}, \mathbf{s}_{\setminus j}\right\rangle)$
5           with probability $p_j$ set $s_j = 1$, otherwise set $s_j = -1$
6   until until convergence
```

The above algorithms are inherently sequential hence extremely slow.

Algorithm 16.17: Learning RBM

We can now appreciate a big advantage in RBM:

$$\mathsf{p}_W(Z_j = z_j | \mathbf{Z}_{\setminus j} = \mathbf{z}_{\setminus j}, \mathbf{X} = \mathbf{x}) \propto \exp\left(z_j \left\langle W_{x,j}, \mathbf{x}\right\rangle\right), \quad i.e. \quad \mathsf{p}_W(Z_j = z_j | \mathbf{Z}_{\setminus j} = \mathbf{z}_{\setminus j}, \mathbf{X} = \mathbf{x}) = \mathtt{sgm}(2z_j \left\langle W_{x,j}, \mathbf{x}\right\rangle),$$

and similarly

$$\mathsf{p}_W(X_j = x_j | \mathbf{X}_{\setminus j} = \mathbf{x}_{\setminus j}, \mathbf{Z} = \mathbf{z}) = \mathtt{sgm}(2x_j \left\langle W_{j,z}, \mathbf{z}\right\rangle).$$

This is possible since in RBM (16.3), $\mathbf{X}$ only interacts with $\mathbf{Z}$ but there is no interaction within either $\mathbf{X}$ or $\mathbf{Z}$. Thus, we may apply block Gibbs sampling:

Algorithm: Block Gibbs sampling from RBM p_W

```
Input: rectangular matrix $W \in \mathbb{R}^{(d+1)\times(t+1)}$
Output: approximate sample $\mathbf{s} \sim \mathsf{p}_W$
1 initialize $\mathbf{s} = (\mathbf{x}, \mathbf{z}) \in \{\pm 1\}^{d+t}$
2 repeat
3     $\mathbf{p} \leftarrow \mathtt{sgm}(2W\mathbf{z})$
4     for $j = 1, \ldots, d$, in parallel do
5         with probability $p_j$ set $x_j = 1$, otherwise set $x_j = -1$
6     $\mathbf{q} \leftarrow \mathtt{sgm}(2W^\top \mathbf{x})$
7     for $j = 1, \ldots, t$, in parallel do
8         with probability $q_j$ set $z_j = 1$, otherwise set $z_j = -1$
9 until until convergence
```

Similarly, to estimate $\mathsf{E}_{\hat{\mathsf{p}}_W} \mathbf{s}\mathbf{s}^\top$ we first draw a training sample $\mathbf{x} \sim \hat{\chi}$, and then draw $\mathbf{z} \sim \mathsf{p}_W(\cdot|\mathbf{x})$. For the latter, we fix $\mathbf{x}$ and apply the Block Gibbs sampling algorithm:

Algorithm: Block Gibbs sampling from conditional RBM $\mathsf{p}_W(\cdot|\mathbf{x})$

```
Input: rectangular matrix $W \in \mathbb{R}^{(d+1)\times(t+1)}$
Output: approximate sample $\mathbf{z} \sim \mathsf{p}_W(\cdot|\mathbf{x})$
1 initialize $\mathbf{z} \in \{\pm 1\}^{t}$
2 repeat
3     $\mathbf{q} \leftarrow \mathtt{sgm}(2W^\top \mathbf{x})$
4     for $j = 1, \ldots, t$, in parallel do
5         with probability $q_j$ set $z_j = 1$, otherwise set $z_j = -1$
6 until until convergence
```

Remark 16.18: Sampling and marginalization

After we have learned the parameter matrix W, we can draw a new sample $(\mathbf{X}, \mathbf{Z})$ from $\mathsf{p}_W(\mathbf{x}, \mathbf{z})$ using the same unconditioned (block) Gibbs sampling algorithm. Simply dropping $\mathbf{Z}$ we obtain a sample $\mathbf{X}$ from the marginal distribution $\mathsf{p}_W(\mathbf{x})$.

For RBM, we can actually derive the marginal density (up to a constant):

$$\begin{aligned}\mathsf{p}_W(\mathbf{X} = \mathbf{x}) &\propto \sum_{\mathbf{z}\in\{\pm1\}^t} \exp\left(\mathbf{x}^\top W \mathbf{z}\right) \qquad (16.6)\\ &= \sum_{\mathbf{z}\in\{\pm1\}^t} \prod_{j=1}^{t+1} \exp(\mathbf{x}^\top W_{:j} z_j)\\ &= \exp(\mathbf{x}^\top W_{:,t+1}) \prod_{j=1}^{t} \left[\exp(\langle \mathbf{x}, W_{:j}\rangle) + \exp(-\langle \mathbf{x}, W_{:j}\rangle)\right].\end{aligned}$$

A similar formula for $\mathsf{p}_W(\mathbf{Z} = \mathbf{z})$ obviously holds as well. Thus, for RBM, if we need only draw a sample $\mathbf{X}$, we can and perhaps should *directly* apply Gibbs sampling to the marginal density (16.6).

Exercise 16.19: Conditional independence in RBM

Prove or disprove the following for BM and RBM:

$$\mathsf{p}_W(\mathbf{z}|\mathbf{x}) = \prod_{j=1}^{t} \mathsf{p}_W(z_j|\mathbf{x}), \quad \mathsf{p}_W(\mathbf{x}|\mathbf{z}) = \prod_{j=1}^{d} \mathsf{p}_W(x_j|\mathbf{z}).$$

Remark 16.20: RBM as stochastic/feed-forward neural network

RBM is often referred to as a two-layer stochastic neural network, where $\mathbf{X}$ is the input layer while $\mathbf{Z}$ is the output layer. It also defines an underlying nonlinear, deterministic, feed-forward network. Indeed, let

$$y_j = \mathsf{p}_W(Z_j = 1|\mathbf{X} = \mathbf{x}) = \mathtt{sgm}(2\langle W_{:j}, \mathbf{x}\rangle)$$

we obtain a nonlinear feedforward network that takes an input $\mathbf{x} \in \{\pm 1\}^d$ and maps it non-linearly to an output $\mathbf{y} \in [0,1]^t$, through:

$$\mathbf{h} = 2W^\top \mathbf{x}$$
$$\mathbf{y} = \mathtt{sgm}(\mathbf{h}).$$

Of course, we can stack RBMs on top of each other and go "deep" (Hinton and Salakhutdinov 2006; Salakhutdinov and Hinton 2012). Applications can be found in (Mohamed et al. 2012; Sarikaya et al. 2014).

Hinton, G. E. and R. R. Salakhutdinov (2006). "Reducing the Dimensionality of Data with Neural Networks". *Science*, vol. 313, pp. 504–507.

Salakhutdinov, Ruslan and Geoffrey Hinton (2012). "An Efficient Learning Procedure for Deep Boltzmann Machines". *Neural Computation*, vol. 24, no. 8, pp. 1967–2006.

Mohamed, A., G. E. Dahl, and G. Hinton (2012). "Acoustic Modeling Using Deep Belief Networks". *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 1, pp. 14–22.

Sarikaya, R., G. E. Hinton, and A. Deoras (2014). "Application of Deep Belief Networks for Natural Language Understanding". *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 22, no. 4, pp. 778–784.

Remark 16.21: Not necessarily binary data

It is possible to extend (R)BM to handle other types of data, see for instance (Welling et al. 2005).

In fact, let $\mathsf{p}_W(\mathbf{x}, \mathbf{z}) = \exp(\langle T(\mathbf{x}, \mathbf{z}), W\rangle - A(W))$ be any joint density in the exponential family. Then, we can estimate the parameter matrix W as before:

$$\min_W \ \mathsf{KL}(\hat{\chi}(\mathbf{x}) \| \mathsf{p}_W(\mathbf{x})) \qquad \equiv \qquad \min_W \ -\mathsf{E}_{\hat{\chi}} \log \int_{\mathbf{z}} \mathsf{p}_W(\mathbf{X}, \mathbf{z}) \, \mathrm{d}\mathbf{z}.$$

Denoting $\hat{\mathsf{p}}_W(\mathbf{x}, \mathbf{z}) = \hat{\chi}(\mathrm{d}\mathbf{x}) \mathsf{p}_W(\mathbf{z}|\mathbf{x})$ and following the same steps as in Remark 16.14 we obtain:

$$\frac{\partial}{\partial W} = -\mathsf{E}_{\hat{\mathsf{p}}_W} T(\mathbf{X}, \mathbf{Z}) + \mathsf{E}_{\mathsf{p}_W} T(\mathbf{X}, \mathbf{Z}).$$

A *restricted* counterpart corresponds to

$$\langle T(\mathbf{x}, \mathbf{z}), W\rangle = T_1(\mathbf{x})^\top W_{xz} T_2(\mathbf{z}).$$

Similar Gibbs sampling algorithms can be derived to approximate the expectations.

Welling, Max, Michal Rosen-zvi, and Geoffrey E. Hinton (2005). "Exponential Family Harmoniums with an Application to Information Retrieval". In: *Advances in Neural Information Processing Systems 17*, pp. 1481–1488.

17 Deep Belief Networks (DBN)

Goal

Belief network, sigmoid belief network, deep belief network, maximum likelihood, Gibbs sampling.

Alert 17.1: Convention

Gray boxes are not required hence can be omitted for unenthusiastic readers.
This note is likely to be updated again soon.

Definition 17.2: DAG

A directed acyclic graph (DAG) is a directed graph that contains no directed cycles. Note that the underlying undirected graph could still contain cycles.

Definition 17.3: Notions in DAGs

A useful property of a DAG is that we can always define a partial ordering on its vertices so that $u \leq v$ iff there is a directed path from u to v. For each vertex v in the directed graph $\mathcal{G} = (V, E)$, we use $\mathrm{pa}(v) := \{u : \overrightarrow{uv} \in E\}$, $\mathrm{ch}(v) := \{u : \overrightarrow{vu} \in E\}$, $\mathrm{an}(v) := \{u : \exists w_1, \ldots, w_k \in V, \overrightarrow{uw_1}, \overrightarrow{w_1w_2}, \ldots, \overrightarrow{w_kv} \in E\}$, $\mathrm{de}(v) := \{u : \exists w_1, \ldots, w_k \in V, \overrightarrow{vw_1}, \overrightarrow{w_1w_2}, \ldots, \overrightarrow{w_ku} \in E\}$, and $\mathrm{nd}(v) := V \setminus (\{v\} \cup \mathrm{de}(v))$ to denote the parents, children, ancestors, descendants, and non-descendants of v, respectively. Similarly we define such sets for a set of nodes by taking unions.

Definition 17.4: Belief/Bayes networks (BNs) (Pearl 1986)

For each vertex v of a DAG $\mathcal{G} = (V = \{1, \ldots, m\}, E)$, we associate a random variable S_v with it. Interchangeably we refer to the vertex either by v or S_v. Together $\mathbf{S} = (S_1, \ldots, S_m)$ defines a Belief network w.r.t. $\mathcal{G}$ iff the joint density p factorizes as follows:

$$\mathsf{p}(s_1, \ldots, s_m) = \prod_{v \in V} \mathsf{p}(s_v \mid \mathrm{pa}(s_v)). \tag{17.1}$$

Pearl, Judea (1986). "Fusion, propagation, and structuring in belief networks". *Artificial Intelligence*, vol. 29, no. 3, pp. 241–288.

Theorem 17.5: Factorization of BNs

Fix a DAG $\mathcal{G}$. For any set of normalized probability densities $\{\mathsf{p}_v(s_v \mid \mathrm{pa}(s_v))\}_{v \in V}$,

$$\mathsf{p}(s_1, \ldots, s_m) = \prod_{v \in V} \mathsf{p}_v(s_v \mid \mathrm{pa}(s_v)) \tag{17.2}$$

defines a BN over $\mathcal{G}$, whose conditionals are precisely given by $\{\mathsf{p}_v\}$.

Proof. W.l.o.g. we may assume the nodes are so arranged that $u \in \mathrm{de}(v) \implies u \geq v$. This is possible since the graph is a DAG. We claim that for all j,

$$\mathsf{p}(s_1, \ldots, s_j) = \prod_{1 \le v \le j} \mathsf{p}_v(s_v \mid \mathrm{pa}(s_v)).$$

The idea is to perform marginalization bottom-up. Indeed,

$$\begin{aligned}
\mathsf{p}(s_1, \ldots, s_j) &:= \int \mathsf{p}(s_1, \ldots, s_m)\,\mathrm{d}s_{j+1} \cdots \mathrm{d}s_m \\
&= \prod_{1 \le v \le j} \mathsf{p}_v(\underbrace{s_v \mid \mathrm{pa}(s_v)}_{\text{no } s_{j+1}, \ldots, s_m}) \int \prod_{j+1 \le v \le m} \mathsf{p}_v(s_v \mid \mathrm{pa}(s_v))\,\mathrm{d}s_{j+1} \cdots \mathrm{d}s_m \\
&= \prod_{1 \le v \le j} \mathsf{p}_v(s_v \mid \mathrm{pa}(s_v)) \int \prod_{j+1 \le v \le m-1} \mathsf{p}_v(\underbrace{s_v \mid \mathrm{pa}(s_v)}_{\text{no } s_m})\,\mathrm{d}s_{j+1} \cdots \mathrm{d}s_{m-1} \Big(\int \mathsf{p}_m(s_m \mid \mathrm{pa}(s_m))\,\mathrm{d}s_m \Big) \\
&= \prod_{1 \le v \le j} \mathsf{p}_v(s_v \mid \mathrm{pa}(s_v)) \int \prod_{j+1 \le v \le m-1} \mathsf{p}_v(s_v \mid \mathrm{pa}(s_v))\,\mathrm{d}s_{j+1} \cdots \mathrm{d}s_{m-1} \\
&= \cdots = \prod_{1 \le v \le j} \mathsf{p}_v(s_v \mid \mathrm{pa}(s_v)).
\end{aligned}$$

This also verifies that p, as defined in (17.2), is indeed a density. Moreover, for all j,

$$\mathsf{p}(s_j \mid s_1, \ldots, s_{j-1}) := \frac{\mathsf{p}(s_1, \ldots, s_j)}{\mathsf{p}(s_1, \ldots, s_{j-1})} = \mathsf{p}_j(s_j \mid \mathrm{pa}(s_j)).$$

Therefore,

$$\begin{aligned}
\mathsf{p}(s_j, \mathrm{pa}(s_j)) &= \int \mathsf{p}(s_1, \ldots, s_j) \prod_{v \in \{1, \ldots, j-1\} \setminus \mathrm{pa}(s_j)} \mathrm{d}s_v \\
&= \mathsf{p}_j(s_j \mid \mathrm{pa}(s_j)) \int \mathsf{p}(s_1, \ldots, s_{j-1}) \prod_{v \in \{1, \ldots, j-1\} \setminus \mathrm{pa}(s_j)} \mathrm{d}s_v \\
&= \mathsf{p}_j(s_j \mid \mathrm{pa}(s_j)) \cdot \mathsf{p}(\mathrm{pa}(s_j)),
\end{aligned}$$

implying the coincidence of the conditionals

$$\mathsf{p}(s_j \mid s_1, \ldots, s_{j-1}) = \mathsf{p}_j(s_j \mid \mathrm{pa}(s_j)) = \mathsf{p}(s_j \mid \mathrm{pa}(s_j)),$$

hence completing our proof. □

The main significance of this theorem is that by specifying local conditional probability densities $\mathsf{p}_v(s_v \mid \mathrm{pa}(s_v))$ for each node v, we automatically obtain a *bona fide* joint probability density p over the DAG.

Example 17.6: Necessity of the DAG condition

Theorem 17.5 can fail if the underlying graph is not a DAG. Consider the simple graph with two nodes X_1, X_2 taking values in $\{0, 1\}$. Define the conditionals as follows:

$$\mathsf{p}(X_1 = 1 \mid X_2 = 1) = 0.5, \mathsf{p}(X_1 = 1 \mid X_2 = 0) = 1, \mathsf{p}(X_2 = 1 \mid X_1 = 1) = 1, \mathsf{p}(X_2 = 1 \mid X_1 = 0) = 0.5.$$

Then if the graph is cyclic and the factorization (**??**) holds, then we have

$$\begin{aligned}
\mathsf{p}(X_1 = 1, X_2 = 1) &= \mathsf{p}(X_1 = 1 \mid X_2 = 1) \cdot \mathsf{p}(X_2 = 1 \mid X_1 = 1) = 0.5 \\
\mathsf{p}(X_1 = 1, X_2 = 0) &= \mathsf{p}(X_1 = 1 \mid X_2 = 0) \cdot \mathsf{p}(X_2 = 0 \mid X_1 = 1) = 0 \\
\mathsf{p}(X_1 = 0, X_2 = 1) &= \mathsf{p}(X_1 = 0 \mid X_2 = 1) \cdot \mathsf{p}(X_2 = 1 \mid X_1 = 0) = 0.25
\end{aligned}$$

$$\mathsf{p}(X_1 = 0, X_2 = 0) = \mathsf{p}(X_1 = 0 \mid X_2 = 0) \cdot \mathsf{p}(X_2 = 0 \mid X_1 = 0) = 0,$$

not a joint distribution. Note that even if we re-normalize the joint distribution, we won't be able to recover the conditionals: $\mathsf{p}(X_1 = 1 \mid X_2 = 1) = \frac{0.5}{0.75} \neq 0.5$.

Remark 17.7: Economic parameterization

For a binary valued random vector $\mathbf{X} \in \{\pm 1\}^d$, in general we need $2^d - 1$ positive numbers to specify its joint density. However, if $\mathbf{X}$ is a BN over a DAG whose maximum in-degree is k, then we need only (at most) $d(2^{k+1} - 1)$ positive numbers to specify the joint density (by specifying each conditional in (17.1)).

For $k = 0$ we obtain the independent setting while for $k = 1$ we include the so-called naive Bayes model.

Definition 17.8: Sigmoid belief network (SBN) (Neal 1992)

Instead of parameterizing each conditional densities $\mathsf{p}(s_v|\mathrm{pa}(s_v))$ directly, we now consider more efficient ways, which is also necessary if S_v is continuous valued.

Following Neal (1992), we restrict our discussion to binary $\mathbf{S} \in \{\pm 1\}^m$ and define

$$\forall j = 1, \ldots, m, \quad \mathsf{p}_W(S_j = s_j | \mathbf{S}_{<j} = \mathsf{s}_{<j}) = \mathtt{sgm}\left(s_j(W_{j,m+1} + \sum_{k=1}^{j-1} s_k W_{jk})\right) = \mathtt{sgm}(s_j W_{j:}\mathsf{s}),$$

where $W \in \mathbb{R}^{m \times (m+1)}$ is strictly lower-triangular (i.e. $W_{jk} = 0$ if $m \geq k \geq j$). Note that we have added a bias term in the last column of W and appended the constant 1 to get $\mathsf{s} = (\mathbf{s}; 1)$. Note the similarity with Boltzmann machines (16.5).

The ordering of the nodes S_j here are chosen rather arbitrarily. A different ordering may lead to consequences in later analysis, although we shall assume in the following a fixed ordering is given without much further discussion.

Applying Theorem 17.5 we obtain a joint density $\mathsf{p}_W(\mathbf{s}) := \mathsf{p}_W(\mathbf{x}, \mathbf{z})$, parameterized by the (strictly lower-triangular) weight matrix W. We will see how to learn W based on a sample $\mathbf{X}_1, \ldots, \mathbf{X}_n \sim \mathsf{p}_W(\mathbf{x})$.

Neal, Radford M. (1992). "Connectionist learning of belief networks". *Artificial Intelligence*, vol. 56, no. 1, pp. 71–113.

Example 17.9: SBN with $m = 1$ and $m = 2$

Let $m = 1$ in SBN. We have

$$\mathsf{p}_b(S = s) = \mathtt{sgm}(sb),$$

which is exactly the Bernoulli distribution whose mean parameter p is parameterized as a sigmoid transformation of b. Compare with Example 16.9.

For $m = 2$, we have instead

$$\mathsf{p}_{w,b,c}(S_1 = s_1, S_2 = s_2) = \mathtt{sgm}(s_1 c) \cdot \mathtt{sgm}(s_2(ws_1 + b)).$$

These examples confirm that without introducing latent variables, SBNs may not approximate all distributions over the cube well.

Definition 17.10: Noisy-OR belief network (Neal 1992)

The sigmoid function is chosen in Definition 17.8 mostly to mimic Boltzmann machines, but it is clear that any univariate CDF works equally well. Indeed, we can even define:

$$\mathsf{p}(S_j = 1|\mathbf{S}_{<j} = \mathbf{s}_{<j}) = 1 - \exp\left(-W\tfrac{1+\mathbf{s}}{2}\right),$$

where as before $W \in \mathbb{R}^{m\times(m+1)}$ is strictly lower-triangular. The awkward term $\frac{\mathbf{s}+1}{2}$ is to reduce to the $\{0,1\}$-valued case, the setting where noisy-or is traditionally defined in. Note that we need the constraint $W(1+\mathbf{s}) \geq 0$ (for instance $W \geq \mathbf{0}$ suffices) to ensure the resulting density is nonnegative.

See (Arora et al. 2017) for some recent results on learning noisy-or networks.

Neal, Radford M. (1992). "Connectionist learning of belief networks". *Artificial Intelligence*, vol. 56, no. 1, pp. 71–113.
Arora, Sanjeev, Rong Ge, Tengyu Ma, and Andrej Risteski (2017). "Provable Learning of Noisy-OR Networks". In: *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pp. 1057–1066.

Remark 17.11: Universality

As shown in (Neal 1992), Boltzmann distributions, SBNs and Noisy-OR BNs in general represent *different* strict subsets of all discrete distributions over the cube $\{\pm1\}^m$. If we introduce (a large number of) t latent variables, then the marginal distribution on $\mathbf{X}$ can approximate any discrete distribution arbitrarily well, for all 3 networks. More refined universality results can be found in (Sutskever and Hinton 2008).
Neal, Radford M. (1992). "Connectionist learning of belief networks". *Artificial Intelligence*, vol. 56, no. 1, pp. 71–113.
Sutskever, Ilya and Geoffrey E. Hinton (2008). "Deep, Narrow Sigmoid Belief Networks Are Universal Approximators". *Neural Computation*, vol. 20, no. 11, pp. 2629–2636.

Example 17.12: SBN with $d = 1$ and $t = 1$

Let $m = 2$ in SBN. We partition $\mathbf{S} = (X, Z)$, hence

$$\mathsf{p}_{w,b,c}(X = x, Z = z) = \mathtt{sgm}(xc) \cdot \mathtt{sgm}(z(wx + b)).$$

Marginalizing over $z = \{\pm1\}$:

$$\mathsf{p}_{w,b,c}(X = x) = \mathtt{sgm}(xc)[\mathtt{sgm}(wx + b) + \mathtt{sgm}(-wx - b)] = \mathtt{sgm}(xc).$$

On the other hand, if we swap the position of Z and X:

$$\mathsf{p}_{w,b,c}(Z = z, X = x) = \mathtt{sgm}(zc) \cdot \mathtt{sgm}(x(wz + b)).$$

Marginalizing over $z = \{\pm1\}$:

$$\mathsf{p}_{w,b,c}(X = x) = [\mathtt{sgm}(c)\mathtt{sgm}(x(w + b)) + \mathtt{sgm}(-c)\mathtt{sgm}(x(-w + b))],$$

which is a mixture.

Algorithm 17.13: SBN–Maximum Likelihood

Given a sample $\mathbf{X}_1, \ldots, \mathbf{X}_n \in \{\pm1\}^d$, we apply ML to estimate W:

$$\min_{W\in\mathbb{R}^{m\times(m+1)}} \mathsf{KL}(\hat{\chi}(\mathbf{x})\|\mathsf{p}_W(\mathbf{x})),$$

where the SBN p_W is defined in Definition 17.8 and we remind again that W is always constrained to be strictly lower-triangular (modulus the last bias column) in this section (although we shall not signal this anymore).

To apply (stochastic) gradient descent, we compute the gradient:

$$\frac{\partial}{\partial W_{j:}} = -\mathsf{E}_{\hat{\mathsf{p}}_W} \frac{\partial \log \mathsf{p}_W(\mathbf{s})}{\partial W_{j:}}, \quad \text{where} \quad \hat{\mathsf{p}}_W(\mathbf{s}) := \mathsf{p}_W(\mathbf{z}, \mathbf{x}) := \hat{\chi}(\mathrm{d}\mathbf{x}) \cdot \mathsf{p}_W(\mathbf{z}|\mathbf{x})$$

$$= -\mathsf{E}_{\hat{\mathsf{p}}_W} \frac{\partial \log \prod_{j=1}^m \mathsf{p}_W(s_j|\mathbf{s}_{<j})}{\partial W_{j:}} = -\mathsf{E}_{\hat{\mathsf{p}}_W} \frac{\partial \log \mathsf{p}_W(s_j|\mathbf{s}_{<j})}{\partial W_{j:}} = -\mathsf{E}_{\hat{\mathsf{p}}_W} \frac{\mathtt{sgm}'(s_j W_{j:}\mathbf{s})}{\mathtt{sgm}(s_j W_{j:}\mathbf{s})} s_j \tilde{\mathbf{s}}_{<j},$$

where we use the tilde notation $\tilde{\mathbf{s}}_{<j}$ to denote the full size vector where coordinates $k \in [j, m]$ are zeroed out (since W is strictly lower-triangular). Using the fact that $\mathtt{sgm}'(t) = \mathtt{sgm}(t)\mathtt{sgm}(-t)$ we obtain:

$$\forall j = 1, \ldots, d, \ \forall k \notin [j, m], \quad \frac{\partial}{\partial W_{jk}} = -\mathsf{E}_{\hat{\mathsf{p}}_W} \mathtt{sgm}(-s_j W_{j:}\mathbf{s}) s_j s_k.$$

Note that the gradient only involves 1 expectation, while there were 2 in BMs (cf. Remark 16.14). This is perhaps expected, since in BM we have that intractable log-partition function to normalize the density while in BN the joint density is automatically normalized since each conditional is so.

Algorithm 17.14: Conditional Gibbs sampling for SBN

We now give the conditional Gibbs sampling algorithm for $\hat{p}_W(\mathbf{x}, \mathbf{z}) = \hat{\chi}(\mathrm{d}\mathbf{x}) \cdot \mathsf{p}_W(\mathbf{z}|\mathbf{x})$. For that we derive the conditional density (Pearl 1987):

$$\begin{aligned}
\mathsf{p}(S_j = t|\mathbf{S}_{\setminus j} = \mathbf{s}_{\setminus j}) &\propto \mathsf{p}(\mathbf{S}_{\setminus j} = \mathbf{s}_{\setminus j}, S_j = t) \\
&\propto \mathsf{p}(S_j = t|\mathbf{S}_{<j} = \mathbf{s}_{<j}) \cdot \prod_{k>j} \mathsf{p}(S_k = s_k|\mathbf{S}_{<k,\setminus j} = \mathbf{s}_{<k,\setminus j}, S_j = t) \\
&= \mathtt{sgm}(tW_{j:}\mathbf{s}) \prod_{k>j} \mathtt{sgm}[s_k(W_{k:}\mathbf{s} + W_{kj}(t - s_j))]
\end{aligned}$$

We omit the pseudo-code but point out that the matrix vector product $W\mathbf{s}$ should be dynamically updated to save computation.

Pearl, Judea (1987). "Evidential reasoning using stochastic simulation of causal models". *Artificial Intelligence*, vol. 32, no. 2, pp. 245–257.

Alert 17.15: Importance of ordering: training vs. testing

If we choose to put the observed variables $\mathbf{X}$ before the latent variables $\mathbf{Z}$ so that $\mathbf{S} = (\mathbf{X}, \mathbf{Z})$. Then, drawing a sample $\mathbf{Z}$ necessitates the above (iterative) Gibbs sampling procedure. Thus, training is expensive. However, after training drawing a new (exact) sample for $\mathbf{X}$ only takes one-shot. In contrast, if we arrange $\mathbf{S} = (\mathbf{Z}, \mathbf{X})$. Then, drawing a sample $\mathbf{Z}$ during training no longer requires the Gibbs sampling algorithm hence training is fast. However, the price to pay is that at test time when we want to draw a new sample $\mathbf{X}$, we would have to run the iterative Gibbs sampling algorithm.

Exercise 17.16: Failure of EM

I know you must wondering if we can apply EM to SBN. The answer is no and the details are left as exercise.

Definition 17.17: Deep belief network (DBN) (Bengio and Bengio 1999)

We now extend SBN to handle any type of data and go deep. We achieve this goal through 3 key steps:

- First, we realize that SBN amounts to specifying d univariate, parameterized densities. Indeed, let

$$\mathsf{p}(s_j|s_{<j}) = \mathsf{p}_j(s_j; \theta_j(\mathbf{s}_{<j})),$$

where p_j is a prescribed univariate density on s_j with parameter θ_j. For instance, p_j may be the exponential family with natural parameter θ_j (e.g. Bernoulli where $\theta_j = p_j$ or Gaussian where $\theta_j = (\mu_j, \sigma_j^2)$).

- Second, the parameter θ_j can be computed as a function from the previous observations $\mathsf{s}_{<j}$. For instance, SBN specifies p_j to be Bernoulli whose mean parameter p_j is computed as the output of a two-layer neural network with sigmoid activation function:

$$\mathbf{h} = W\mathbf{s}$$
$$\mathbf{p} = \mathtt{sgm}(\mathbf{h}).$$

- It is then clear that we could use a deep neural net to compute the parameter θ_j:

$$\begin{aligned} \mathbf{h}_0 &= \mathbf{s} \\ \forall \ell = 1, \ldots, L-1, \quad \mathbf{h}_\ell &= \sigma(W_\ell \mathsf{h}_{\ell-1}) \\ \boldsymbol{\theta} &= \sigma(W_L \mathsf{h}_{L-1}). \end{aligned} \tag{17.3}$$

Note that we need to make sure θ_j depends only on the first $j-1$ inputs $\mathbf{s}_{<j}$, which can be achieved by wiring the network appropriately. For instance, if W_1 is strictly lower-triangular while $W_{\geq 2}$ is lower-triangular (Bengio and Bengio 1999), then we obviously satisfy this constraint.

Bengio, Yoshua and Samy Bengio (1999). "Modeling High-Dimensional Discrete Data with Multi-Layer Neural Networks". In: *NeurIPS*.

Alert 17.18: Weight sharing

We compare the network structure (17.3) with the following straightforward alternative:

$$\begin{aligned} \mathbf{h}_0^{(j)} &= \mathbf{s}_{<j} \\ \forall \ell = 1, \ldots, L-1, \quad \mathbf{h}_\ell^{(j)} &= \sigma(W_\ell^{(j)} \mathsf{h}_{\ell-1}^{(j)}) \\ \theta_j &= \sigma(W_L^{(j)} \mathsf{h}_{L-1}^{(j)}), \end{aligned} \tag{17.4}$$

where we use separate networks for each parameter θ_j. The weights $W_\ell^{(j)}$ above can be arbitrary. We observe that the parameterization in (17.3) is much more efficient, since the weights used to compute θ_j are shared with all subsequent computations for $\theta_{\geq j}$. Needless to say, the parameterization in (17.4) are more flexible.

18 Generative Adversarial Networks (GAN)

Goal

Push-forward, Generative Adversarial Networks, min-max optimization, duality.

Alert 18.1: Convention

Gray boxes are not required hence can be omitted for unenthusiastic readers.
This note is likely to be updated again soon.

Example 18.2: Simulating distributions

Suppose we want to sample from a Gaussian distribution with mean $\mathbf{u}$ and covariance S. The typical approach is to first sample from the standard Gaussian distribution (with zero mean and identity covariance) and then perform the transformation:

$$\text{If } \mathbf{Z} \sim \mathcal{N}(\mathbf{0}, \mathbb{I}), \quad \text{then } X = \mathsf{T}(\mathbf{Z}) := \mathbf{u} + S^{1/2}\mathbf{Z} \sim \mathcal{N}(\mathbf{u}, S).$$

Similarly, we can sample from a χ^2 distribution with zero mean and degree d by the transformation:

$$\text{If } \mathbf{Z} \sim \mathcal{N}(\mathbf{0}, \mathbb{I}_d), \quad \text{then } X = \mathsf{T}(\mathbf{Z}) := \sum_{j=1}^{d} Z_j^2 \sim \chi^2(d).$$

In fact, we can sample from any distribution F on $\mathbb{R}$ by the following transformation:

$$\text{If } Z \sim \mathcal{N}(0, 1), \quad \text{then } X = \mathsf{T}(Z) := \mathsf{F}^{-}(\Phi(Z)) \sim \mathsf{F}, \quad \text{where} \quad \mathsf{F}^{-}(z) = \min\{x : F(x) \geq z\},$$

and Φ is the cumulative distribution function of standard normal.

Theorem 18.3: Transforming to any probability measure

Let μ be a diffuse (Borel) probability measure on a polish space Z and similarly ν be any (Borel) probability measure on another polish space X. Then, there exist (measurable) maps $\mathsf{T} : \mathsf{Z} \to \mathsf{X}$ such that

$$\textit{If } Z \sim \mu, \quad \textit{then } X := \mathsf{T}(Z) \sim \nu.$$

□

Recall that a (Borel) probability measure is diffuse iff any single point has measure 0. For less mathematical readers, think of $\mathsf{Z} = \mathbb{R}^p$, $\mathsf{X} = \mathbb{R}^d$, μ and ν as probability densities on the respective Euclidean spaces.

Definition 18.4: Push-forward generative modeling

Given an i.i.d. sample $\mathbf{X}_1, \ldots, \mathbf{X}_n \sim \chi$, we can now estimate the target density χ by the following push-forward approach:

$$\inf_{\boldsymbol{\theta}} \ \mathsf{D}(\mathbf{X}, \mathsf{T}_{\boldsymbol{\theta}}(\mathbf{Z})),$$

where say $\mathbf{Z} \sim \mathcal{N}(\mathbf{0}, \mathbb{I}_p)$, $\mathsf{T}_{\boldsymbol{\theta}} : \mathbb{R}^p \to \mathbb{R}^d$, and $\mathbf{X} \sim \chi$ (the true underlying data generating distribution). The function D is a "distance" that measures the closeness of our (true) data distribution (represented by $\mathbf{X}$) and model distribution (represented by $\mathsf{T}_{\boldsymbol{\theta}}(\mathbf{Z})$). By minimizing D we bring our model $\mathsf{T}_{\boldsymbol{\theta}}(\mathbf{Z})$ close to our data $\mathbf{X}$.

Remark 18.5: The good, the bad, and the beautiful

One big advantage of the push-forward approach in Definition 18.4 is that after training (e.g. finding a reasonable $\boldsymbol{\theta}$) we can *effortlessly* generate new data: we sample $\mathbf{Z} \in \mathcal{N}(\mathbf{0}, \mathbb{I}_d)$ and then set $\mathbf{X} = \mathsf{T}_{\boldsymbol{\theta}}(\mathbf{Z})$. On the flip side, we no longer have any explicit form for the model density (namely, that of $\mathsf{T}_{\boldsymbol{\theta}}(\mathbf{Z})$ when $p < d$). This renders direct maximum likelihood estimation of $\boldsymbol{\theta}$ impossible.

This is where we need the beautiful idea called duality. Basically, we need to distinguish two distributions: the data distribution represented by a sample $\mathbf{X}$ and the model distribution represented by a sample $\mathsf{T}_{\boldsymbol{\theta}}(\mathbf{Z})$. We distinguish them by running many tests, represented by functions f:

$$\sup_{f \in \mathcal{F}} \left|\mathsf{E} f(\mathbf{X}) - \mathsf{E} f\big(\mathsf{T}_{\boldsymbol{\theta}}(\mathbf{Z})\big)\right|.$$

If the class of tests $\mathcal{F}$ we run is dense enough, then we would be able to tell the difference between the two distributions and provide feedback for the model $\boldsymbol{\theta}$ to improve, until we no longer can tell the difference.

Definition 18.6: f-divergence (Csiszár 1963; Ali and Silvey 1966)

Let $f : \mathbb{R}_+ \to \mathbb{R}$ be a strictly convex function (see the background lecture on optimization) with $f(1) = 0$. We define the following f-divergence to measure the closeness of two pdfs p and q:

$$\mathsf{D}_f(\mathsf{p}\|\mathsf{q}) := \int f\big(\mathsf{p}(\mathbf{x})/\mathsf{q}(\mathbf{x})\big) \cdot \mathsf{q}(\mathbf{x})\, \mathrm{d}\mathbf{x}, \tag{18.1}$$

where we assume $\mathsf{q}(\mathbf{x}) = 0 \implies \mathsf{p}(\mathbf{x}) = 0$ (otherwise we put the divergence to ∞).

For two random variables $Z \sim \mathsf{q}$ and $X \sim \mathsf{p}$, we sometimes abuse the notation to mean

$$\mathsf{D}_f(X\|Z) := \mathsf{D}_f(\mathsf{p}\|\mathsf{q}).$$

Csiszár, Imre (1963). "Eine informationstheoretische Ungleichung und ihre Anwendung auf den Beweis der Ergodizität von Markoffschen Ketten". *A Magyar Tudományos Akadémia Matematikai Kutató Intézetének közleményei*, vol. 8, pp. 85–108.

Ali, S. M. and S. D. Silvey (1966). "A General Class of Coefficients of Divergence of One Distribution from Another". *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 28, no. 1, pp. 131–142.

Exercise 18.7: Properties of f-divergence

Prove the following:

- $\mathsf{D}_f(\mathsf{p}\|\mathsf{q}) \geq 0$, with 0 attained iff $\mathsf{p} = \mathsf{q}$;
- $\mathsf{D}_{f+g} = \mathsf{D}_f + \mathsf{D}_g$ and $\mathsf{D}_{sf} = s\mathsf{D}_f$ for $s > 0$;
- Let $g(t) = f(t) + s(t-1)$ for any s. Then, $\mathsf{D}_g = \mathsf{D}_f$;
- If $\mathsf{p}(\mathbf{x}) = 0 \iff \mathsf{q}(\mathbf{x}) = 0$, then $\mathsf{D}_f(\mathsf{p}\|\mathsf{q}) = \mathsf{D}_{f^\diamond}(\mathsf{q}\|\mathsf{p})$, where $f^\diamond(t) := t \cdot f(1/t)$;
- $f^\diamond$ is (strictly) convex, $f^\diamond(1) = 0$ and $(f^\diamond)^\diamond = f$;

The second last result indicates that f-divergences are not usually symmetric. However, we can always symmetrize them by the transformation: $f \leftarrow f + f^\diamond$.

Example 18.8: KL and LK

Let $f(t) = t \log t$, then we obtain the Kullback-Leibler (KL) divergence:

$$\mathsf{KL}(\mathsf{p}\|\mathsf{q}) = \int \mathsf{p}(\mathbf{x}) \log(\mathsf{p}(\mathbf{x})/\mathsf{q}(\mathbf{x}))\, \mathrm{d}\mathbf{x}.$$

Reverse the inputs we obtain the reverse KL divergence:

$$\mathsf{LK}(\mathsf{p}\|\mathsf{q}) := \mathsf{KL}(\mathsf{q}\|\mathsf{p}).$$

Verify by yourself that the underlying function $f = -\log$ for reverse KL.

Example 18.9: More divergences, more fun

Derive the formula for the following f-divergences:

- χ^2-divergence: $f(t) = (t-1)^2$;
- Hellinger divergence: $f(t) = (\sqrt{t}-1)^2$;
- total variation: $f(t) = |t-1|$;
- Jensen-Shannon divergence: $f(t) = t\log t - (t+1)\log(t+1) + \log 4$;
- Rényi divergence (Rényi 1961): $f(t) = \frac{t^\alpha - 1}{\alpha - 1}$ for some $\alpha > 0$ (for $\alpha = 1$ we take limit and obtain ?).

Which of the above are symmetric?

Rényi, Alfréd (1961). "On Measures of Entropy and Information". In: *Proceedings of the Fourth Berkeley Symposium on Mathematical Statistics and Probability*, pp. 547–561.

Definition 18.10: Fenchel conjugate function

For any extended real-valued function $f : \mathsf{V} \to (-\infty, \infty]$ we define its Fenchel conjugate function as:

$$f^*(\mathbf{x}^*) := \sup_{\mathbf{x}} \langle \mathbf{x}, \mathbf{x}^* \rangle - f(\mathbf{x}).$$

We remark that f^* is always a convex function (of $\mathbf{x}^*$).

If dom f is nonempty and closed, and f is continuous, then

$$f^{**} := (f^*)^* = f.$$

This remarkable property of convex functions will now be used!

Example 18.11: Fenchel conjugate of JS

Consider the convex function that defines the Jensen-Shannon divergence:

$$f(t) = t\log t - (t+1)\log(t+1) + \log 4. \tag{18.2}$$

We derive its Fenchel conjugate:

$$f^*(s) = \sup_t st - f(t) = \sup_t st - t\log t + (t+1)\log(t+1) - \log 4.$$

Taking derivative w.r.t. t we obtain

$$s - \log t - 1 + \log(t+1) + 1 = 0 \iff t = \frac{1}{\exp(-s) - 1},$$

and plugging it back we get

$$\begin{aligned}
f^*(s) &= \frac{s}{\exp(-s)-1} - \frac{1}{\exp(-s)-1}\log\frac{1}{\exp(-s)-1} + \frac{\exp(-s)}{\exp(-s)-1}\log\frac{\exp(-s)}{\exp(-s)-1} - \log 4 \\
&= \frac{s}{\exp(-s)-1} - \frac{1}{\exp(-s)-1}\log\frac{1}{\exp(-s)-1} + \frac{\exp(-s)}{\exp(-s)-1}\log\frac{1}{\exp(-s)-1} - \frac{s\exp(-s)}{\exp(-s)-1} - \log 4 \\
&= -s - \log(\exp(-s)-1) - \log 4 \\
&= -\log(1-\exp(s)) - \log 4. \qquad (18.3)
\end{aligned}$$

Using conjugation again, we obtain the important formula:

$$f(t) = \sup_s st - f^*(s) = \sup_s st + \log(1-\exp(s)) + \log 4.$$

Exercise 18.12: More conjugates

Derive the Fenchel conjugate of the other convex functions in Example 18.8 and Example 18.9.

Definition 18.13: Generative adversarial networks (GAN) (Goodfellow et al. 2014)

We are now ready to define the original GAN, which amounts to using the Jensen-Shannon divergence in Definition 18.4:

$$\inf_{\boldsymbol{\theta}} \ \mathsf{JS}(\mathbf{X}\|\mathsf{T}_{\boldsymbol{\theta}}(\mathbf{Z})), \quad \text{where} \quad \mathsf{JS}(\mathsf{p}\|\mathsf{q}) = \mathsf{D}_f(\mathsf{p}\|\mathsf{q}) = \mathsf{KL}(\mathsf{p}\|\tfrac{\mathsf{p}+\mathsf{q}}{2}) + \mathsf{KL}(\mathsf{p}\|\tfrac{\mathsf{p}+\mathsf{q}}{2}),$$

and the convex function f is defined in (18.2), along with its Fenchel conjugate f^* given in (18.3).

To see how we can circumvent the lack of an explicit form of the density $\mathsf{q}(\mathbf{x})$ of $\mathsf{T}_{\boldsymbol{\theta}}(\mathbf{Z})$, we expand using duality:

$$\begin{aligned}
\mathsf{JS}(\mathbf{X}\|\mathsf{T}_{\boldsymbol{\theta}}(\mathbf{Z})) &= \int_{\mathbf{x}} f\big(\mathsf{p}(\mathbf{x})/\mathsf{q}(\mathbf{x})\big)\mathsf{q}(\mathbf{x})\,\mathrm{d}\mathbf{x} \\
&= \int_{\mathbf{x}} [\sup_s s\mathsf{p}(\mathbf{x})/\mathsf{q}(\mathbf{x}) - f^*(s)]\mathsf{q}(\mathbf{x})\,\mathrm{d}\mathbf{x} \\
&= \int_{\mathbf{x}} [\sup_s s\mathsf{p}(\mathbf{x}) - f^*(s)\mathsf{q}(\mathbf{x})]\,\mathrm{d}\mathbf{x} \\
&= \sup_{\mathsf{S}:\mathbb{R}^d\to\mathbb{R}} \int_{\mathbf{x}} \mathsf{S}(\mathbf{x})\mathsf{p}(\mathbf{x})\,\mathrm{d}\mathbf{x} - \int_{\mathbf{x}} f^*(\mathsf{S}(\mathbf{x}))\mathsf{q}(\mathbf{x})\,\mathrm{d}\mathbf{x} \\
&= \sup_{\mathsf{S}:\mathbb{R}^d\to\mathbb{R}} \mathsf{E}\mathsf{S}(\mathbf{X}) - \mathsf{E}f^*(\mathsf{S}(\mathsf{T}_{\boldsymbol{\theta}}(\mathbf{Z}))).
\end{aligned}$$

Therefore, if we parameterize the test function S by $\boldsymbol{\phi}$ (say a deep net), then we obtain a lower bound of the Jensen-Shannon divergence for minimizing:

$$\inf_{\boldsymbol{\theta}} \sup_{\boldsymbol{\phi}} \ \mathsf{E}\mathsf{S}_{\boldsymbol{\phi}}(\mathbf{X}) - \mathsf{E}f^*(\mathsf{S}_{\boldsymbol{\phi}}(\mathsf{T}_{\boldsymbol{\theta}}(\mathbf{Z}))).$$

Of course, we cannot compute either of the two expectations, so we use sample average to approximate them:

$$\inf_{\boldsymbol{\theta}} \sup_{\boldsymbol{\phi}} \ \hat{\mathsf{E}}\mathsf{S}_{\boldsymbol{\phi}}(\mathbf{X}) - \hat{\mathsf{E}}f^*(\mathsf{S}_{\boldsymbol{\phi}}(\mathsf{T}_{\boldsymbol{\theta}}(\mathbf{Z}))), \qquad (18.4)$$

where the first sample expectation $\hat{\mathsf{E}}$ is simply the average of the given training data while the second sample expectation is the average over samples generated by the model $\mathsf{T}_{\boldsymbol{\theta}}(\mathbf{Z})$ (recall Remark 18.5).

In practice, both $\mathsf{T}_{\boldsymbol{\theta}}$ and $\mathsf{S}_{\boldsymbol{\phi}}$ are represented by deep nets, and the former is called the generator while the latter is called the discriminator. Our final objective (18.4) represents a two-player game between the generator and the discriminator. At equilibrium (if any) the generator is forced to mimic the (true) data distribution (otherwise the discriminator would be able to tell the difference and incur a loss for the generator).

See the background lecture on optimization for a simple algorithm (gradient-descent-ascent) for solving (18.4).

Goodfellow, Ian, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio (2014). "Generative Adversarial Nets". In: *NIPS*.

Remark 18.14: Approximation

We made a number of approximations in Definition 18.13. Thus, technically speaking, the final GAN objective (18.4) no longer minimizes the Jensen-Shannon divergence. Nock et al. (2017) and Liu et al. (2017) formally studied this approximation trade-off.

Nock, Richard, Zac Cranko, Aditya K. Menon, Lizhen Qu, and Robert C. Williamson (2017). "f-GANs in an Information Geometric Nutshell". In: *NIPS*.

Liu, Shuang, Léon Bottou, and Kamalika Chaudhuri (2017). "Approximation and convergence properties of generative adversarial learning". In: *NIPS*.

Exercise 18.15: Catch me if you can

Let us consider the game between the generator $\mathsf{q}(\mathbf{x})$ (the implicit density of $\mathsf{T}_{\boldsymbol{\theta}}(\mathbf{Z})$) and the discriminator $\mathsf{S}(\mathbf{x})$:

$$\inf_{\mathsf{q}} \sup_{\mathsf{S}} \int_{\mathbf{x}} \mathsf{S}(\mathbf{x})\mathsf{p}(\mathbf{x})\,\mathrm{d}\mathbf{x} + \int_{\mathbf{x}} \log\big(1-\exp(\mathsf{S}(\mathbf{x}))\big)\mathsf{q}(\mathbf{x})\,\mathrm{d}\mathbf{x} + \log 4.$$

- Fixing the generator q, what is the optimal discriminator S?
- Plugging the optimal discriminator S back in, what is the optimal generator?
- Fixing the discriminator S, what is the optimal generator q?
- Plugging the optimal generator q back in, what is the optimal discriminator?

Exercise 18.16: KL vs. LK

Recall that the f-divergence $\mathsf{D}_f(\mathsf{p}\|\mathsf{q})$ is infinite iff for some $\mathbf{x}$, $\mathsf{p}(\mathbf{x}) \neq 0$ while $\mathsf{q}(\mathbf{x}) = 0$. Consider the following twin problems:

$$\mathsf{q}_{\mathsf{KL}} := \operatorname*{argmin}_{\mathsf{q}\in\mathcal{Q}} \ \mathsf{KL}(\mathsf{p}\|\mathsf{q})$$
$$\mathsf{q}_{\mathsf{LK}} := \operatorname*{argmin}_{\mathsf{q}\in\mathcal{Q}} \ \mathsf{LK}(\mathsf{p}\|\mathsf{q}).$$

Recall that $\operatorname{supp}(\mathsf{p}) := \operatorname{cl}\{\mathbf{x} : \mathsf{p}(\mathbf{x}) \neq \mathbf{0}\}$. What can we say about $\operatorname{supp}(\mathsf{p})$, $\operatorname{supp}(\mathsf{q}_{\mathsf{KL}})$ and $\operatorname{supp}(\mathsf{q}_{\mathsf{LK}})$?

What about JS?

Definition 18.17: f-GAN (Nowozin et al. 2016)

Following Nowozin et al. (2016), we summarize the main idea of f-GAN as follows:

- Generator: Let μ be a fixed reference probability measure on space Z (usually the standard normal distribution) and $Z \sim \mu$. Let ν be any target probability measure on space X and $X \sim \nu$. Let

$\mathcal{T} \subseteq \{\mathsf{T} : \mathsf{Z} \to \mathsf{X}\}$ be a class of transformations. According to Theorem 18.3 we know there exist transformations T (which *may or may not* be in our class $\mathcal{T}$) so that $\mathsf{T}(Z) \sim X \sim \nu$. Our goal is to approximate such transformations T using our class $\mathcal{T}$.

- Loss: We use the f-divergence to measure the closeness between the target X and the transformed reference $\mathsf{T}(Z)$:

$$\inf_{\mathsf{T} \in \mathcal{T}} \ \mathsf{D}_f\big(X \| \mathsf{T}(Z)\big).$$

In fact, any loss function that allows us to distinguish two probability measures can be used. However, we face an additional difficulty here: the densities of X and $\mathsf{T}(Z)$ (w.r.t. a third probability measure λ) are not known to us (especially the former) so we cannot naively evaluate the f-divergence in (18.1).

- Discriminator: A simple variational reformulation will resolve the above difficulty! Indeed,

$$\begin{aligned}
\mathsf{D}_f(X\|\mathsf{T}(Z)) &= \int f\left(\frac{\mathrm{d}\nu}{\mathrm{d}\tau}(\mathbf{x})\right)\mathrm{d}\tau(\mathbf{x}) && (\mathsf{T}(Z) \sim \tau)\\
&= \int \sup_{s \in \mathrm{dom}(f^*)} \left[s\frac{\mathrm{d}\nu}{\mathrm{d}\tau}(\mathbf{x}) - f^*(s)\right]\mathrm{d}\tau(\mathbf{x}) && (f^{**} = f)\\
&\geq \sup_{\mathsf{S} \in \mathcal{S}} \int \left[S(\mathbf{x})\frac{\mathrm{d}\nu}{\mathrm{d}\tau}(\mathbf{x}) - f^*(S(\mathbf{x}))\right]\mathrm{d}\tau(\mathbf{x}) && (\mathcal{S} \subseteq \{\mathsf{S} : \mathsf{X} \to \mathrm{dom}(f^*)\})\\
&= \sup_{\mathsf{S} \in \mathcal{S}} \ \mathbf{E}[\mathsf{S}(X)] - \mathbf{E}[f^*\big(\mathsf{S}(\mathsf{T}(Z))\big)] && \left(\text{equality if } f'\left(\frac{\mathrm{d}\nu}{\mathrm{d}\tau}\right) \in \mathcal{S}\right),
\end{aligned}$$

so our estimation problem reduces to the following minimax zero-sum game:

$$\inf_{\mathsf{T} \in \mathcal{T}} \ \sup_{\mathsf{S} \in \mathcal{S}} \ \mathbf{E}[\mathsf{S}(X)] - \mathbf{E}[f^*\big(\mathsf{S}(\mathsf{T}(Z))\big)].$$

By replacing the expectations with empirical averages we can (approximately) solve the above problem with classic stochastic algorithms.

- Reparameterization: The class of functions $\mathcal{S}$ we use to test the difference between two probability measures in the f-divergence must have their range contained in the domain of f^*. One convenient way to enforce this constraint is to set

$$\mathcal{S} = \sigma \circ \mathcal{U} := \{\sigma \circ \mathsf{U} : \mathsf{U} \in \mathcal{U}\}, \quad \sigma : \mathbb{R} \to \mathrm{dom}(f^*), \quad \mathcal{U} \subseteq \{\mathsf{U} : \mathsf{X} \to \mathbb{R}\},$$

where the functions U are unconstrained and the domain constraint is enforced through a *fixed* "activation function" σ. With this choice, the final f-GAN problem we need to solve is:

$$\inf_{\mathsf{T} \in \mathcal{T}} \ \sup_{\mathsf{U} \in \mathcal{U}} \ \mathbf{E}[\sigma \circ \mathsf{U}(X)] - \mathbf{E}[(f^* \circ \sigma)\big(\mathsf{U}(\mathsf{T}(Z))\big)].$$

Typically we choose an increasing σ so that the composition $f^* \circ \sigma$ is "nice." Note that the monotonicity of σ implies the same monotonicity of the composition $f^* \circ \sigma$ (since f^* is always increasing as f is defined only on $\mathbb{R}_+$). In this case, we prefer to pick a test function U so that $\mathsf{U}(X)$ is large while $\mathsf{U}(\mathsf{T}(Z))$ is small. This choice aligns with the goal to "maximize target and minimize transformed reference," although the opposite choice would work equally well (merely a sign change).

Nowozin, Sebastian, Botond Cseke, and Ryota Tomioka (2016). "f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization". In: *NIPS*.

Remark 18.18: f-GAN recap

To specify an f-GAN, we need:

- A reference probability measure μ: should be easy to sample and typically we use standard normal;
- A class of transformations (generators): $\mathcal{T} \subseteq \{\mathsf{T} : \mathsf{Z} \to \mathsf{X}\}$;
- An increasing convex function $f^* : \mathrm{dom}(f^*) \to \mathbb{R}$ with $f^*(0) = 0$ and $f^*(s) \geq s$ (or equivalently an f-divergence);
- An increasing activation function $\sigma : \mathbb{R} \to \mathrm{dom}(f^*)$ so that $f^* \circ \sigma$ is "nice";
- A class of *unconstrained* test functions (discriminators): $\mathcal{U} \subseteq \{\mathsf{U} : \mathsf{X} \to \mathbb{R}\}$ so that $\mathcal{S} = \sigma \circ \mathcal{U}$.

Definition 18.19: Wasserstein GAN (WGAN) (Arjovsky et al. 2017)

If we let the test functions range over the set of all 1-Lipschitz continuous functions $\mathcal{L}$, we then obtain WGAN:

$$\inf_{\boldsymbol{\theta}} \sup_{\mathsf{S} \in \mathcal{L}} \mathsf{ES}(\mathbf{X}) - \mathsf{ES}\big(\mathsf{T}_{\boldsymbol{\theta}}(\mathbf{Z})\big),$$

which corresponds to the dual of the 1-Wasserstein distance.

Arjovsky, Martin, Soumith Chintala, and Léon Bottou (2017). "Wasserstein Generative Adversarial Networks". In: *ICML*.

Definition 18.20: Maximum Mean Discrepancy GAN (MMD-GAN)

If, instead, we choose the test functions from a reproducing kernel Hilbert space (RKHS), then we obtain the so-called MMD-GAN (Dziugaite et al. 2015; Li et al. 2015; Li et al. 2017; Bellemare et al. 2017; Bińkowski et al. 2018):

$$\inf_{\boldsymbol{\theta}} \sup_{\mathsf{S} \in \mathcal{H}_\kappa} \mathsf{ES}(\mathbf{X}) - \mathsf{ES}\big(\mathsf{T}_{\boldsymbol{\theta}}(\mathbf{Z})\big),$$

where $\mathcal{H}_\kappa$ is the unit ball of the RKHS induced by the kernel κ.

Dziugaite, Gintare Karolina, Daniel M. Roy, and Zoubin Ghahramani (2015). "Training generative neural networks via maximum mean discrepancy optimization". In: *UAI*.

Li, Yujia, Kevin Swersky, and Rich Zemel (2015). "Generative Moment Matching Networks". In: *ICML*.

Li, Chun-Liang, Wei-Cheng Chang, Yu Cheng, Yiming Yang, and Barnabas Poczos (2017). "MMD GAN: Towards Deeper Understanding of Moment Matching Network". In: *NIPS*.

Bellemare, Marc G., Ivo Danihelka, Will Dabney, Shakir Mohamed, Balaji Lakshminarayanan, Stephan Hoyer, and Remi Munos (2017). "The Cramer Distance as a Solution to Biased Wasserstein Gradients". arXiv:1705.10743.

Bińkowski, Mikolaj, Dougal J. Sutherland, Michael Arbel, and Arthur Gretton (2018). "Demystifying MMD GANs". In: *ICLR*.

19 Attention

Goal

Introduction to transformer, attention, BERT, GPTs, and the exploding related.
A nice code tutorial is available: `https://nlp.seas.harvard.edu/2018/04/03/attention.html`.

Alert 19.1: Convention

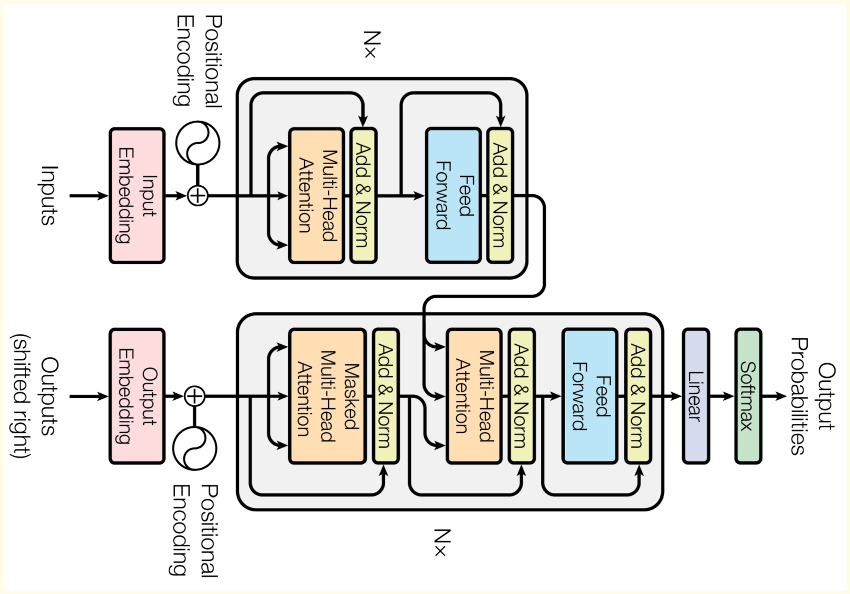
Gray boxes are not required hence can be omitted for unenthusiastic readers. Tokens are arranged row-wise.
This note is likely to be updated again soon.

Remark 19.2: The input sequential price of RNNs

A lot of natural language processing (NLP) techniques rely on recurrent neural networks, which are unfortunately sequential in nature (w.r.t. input tokens). Our main goal in this lecture is to use a hierarchical, parallelizable attention mechanism to trade the input sequential part in RNNs with that in network depth.

Definition 19.3: Transformer (Vaswani et al. 2017)

In a nutshell, a transformer (in machine learning!) is composed of multiple blocks of components that we explain in details below. It takes an input sequence and outputs another sequence, much like an RNN:



Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin (2017). "Attention is All you Need". In: *Advances in Neural Information Processing Systems 30*, pp. 5998–6008.

Definition 19.4: Input and output embeddings

Typically, the tokens in an input sequence are one-hot encoded over a dictionary with size say p. We add and learn a distributed representation $W_e \in \mathbb{R}^{p\times d}$ of the tokens. The same W_e is also used to decode the output tokens, and its transpose $W_e^\top$ is used to compute the `softmax` output probability (over tokens).

Definition 19.5: Positional encoding

The order of tokens in an input sequence matters. To encode this information, we may simply add a positional vector $\mathfrak{p}_t \in \mathbb{R}^d$ for each fixed position t:

$$\mathfrak{p}_{t,2i} = \sin\left(t/10000^{2i/d}\right), \quad \mathfrak{p}_{t,2i+1} = \cos\left(t/10000^{2i/d}\right), \quad i = 0, \ldots, \tfrac{d}{2} - 1.$$

It is clear that $\mathfrak{p}_{t+k}$ is an orthogonal transformation of $\mathfrak{p}_t$, since

$$\begin{aligned}\begin{bmatrix}\mathfrak{p}_{t+k,2i}\\ \mathfrak{p}_{t+k,2i+1}\end{bmatrix} &= \begin{bmatrix}\cos(k/10000^{2i/d}) & \sin(k/10000^{2i/d})\\ -\sin(k/10000^{2i/d}) & \cos(k/10000^{2i/d})\end{bmatrix}\begin{bmatrix}\sin\left(t/10000^{2i/d}\right)\\ \cos\left(t/10000^{2i/d}\right)\end{bmatrix}\\ &= \begin{bmatrix}\cos(k/10000^{2i/d}) & \sin(k/10000^{2i/d})\\ -\sin(k/10000^{2i/d}) & \cos(k/10000^{2i/d})\end{bmatrix}\begin{bmatrix}\mathfrak{p}_{t,2i}\\ \mathfrak{p}_{t,2i+1}\end{bmatrix}.\end{aligned}$$

The periodic functions chosen here also allows us to handle test sequences that are longer than the training sequences. Of course, one may instead try to directly *learn* the positional encoding $\mathfrak{p}_t$.

Definition 19.6: Residual connection, layer-normalization and dropout

In each layer we add residual connection (He et al. 2016) and layer-wise normalization (Ba et al. 2016) to ease training. We may also add dropout layers (Srivastava et al. 2014).

He, K., X. Zhang, S. Ren, and J. Sun (2016). "Deep Residual Learning for Image Recognition". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778.

Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E. Hinton (2016). "Layer Normalization".

Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov (2014). "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". *Journal of Machine Learning Research*, vol. 15, no. 56, pp. 1929–1958.

Definition 19.7: Attention (e.g. Bahdanau et al. 2015)

Given a query vector $\mathbf{q} \in \mathbb{R}^{d_k}$ and a database consisting of m key-value pairs $(K, V) : K = [\mathbf{k}_1, \ldots, \mathbf{k}_m]^\top \in \mathbb{R}^{m\times d_k}, V = [\mathbf{v}_1, \ldots, \mathbf{v}_m]^\top \in \mathbb{R}^{m\times d_v}$, a natural way to guess/retrieve the value of the query is through comparison to the known key-value pairs:

$$\mathtt{att}(\mathbf{q}; K, V) = \sum_{i=1}^m \pi_i \cdot \mathbf{v}_i = \boldsymbol{\pi}^\top V, \tag{19.1}$$

namely the value of the query is some convex combination of the values in the database.

The coefficient vector $\boldsymbol{\pi}$ can be determined as follows:

$$\operatorname*{argmin}_{\boldsymbol{\pi}\in\Delta_{m-1}} \sum_{i=1}^m \pi_i \cdot \mathrm{dist}_k(\mathbf{q}, \mathbf{k}_i) + \lambda \cdot \pi_i \log \pi_i, \tag{19.2}$$

where $\mathrm{dist}_k(\mathbf{q}, \mathbf{k}_i)$ measures the dissimilarity between the query $\mathbf{q}$ and the key $\mathbf{k}_i$, and $\pi_i \log \pi_i$ is the so-called entropic regularizer. As you can verify in Exercise 19.8:

$$\pi_i = \frac{\exp(-\mathrm{dist}_k(\mathbf{q}, \mathbf{k}_i)/\lambda)}{\sum_{j=1}^m \exp(-\mathrm{dist}_k(\mathbf{q}, \mathbf{k}_j)/\lambda)}, \quad \textit{i.e.}, \quad \boldsymbol{\pi} = \mathtt{softmax}(-\mathrm{dist}_k(\mathbf{q}, K)/\lambda). \tag{19.3}$$

A popular choice for the dissimilarity function is $\mathrm{dist}_k(\mathbf{q}, \mathbf{k}) = -\mathbf{q}^\top\mathbf{k}$.

Now, with the above $\boldsymbol{\pi}$, we solve a similar weighted regression problem to retrieve the value of the query:

$$\operatorname*{argmin}_{\mathbf{a}} \sum_{i=1}^m \pi_i \cdot \mathrm{dist}_v(\mathbf{a}, \mathbf{v}_i).$$

With $\mathrm{dist}_v(\mathbf{a}, \mathbf{v}_i) := \|\mathbf{a} - \mathbf{v}_i\|_2^2$, we easily verify the convex combination in (19.1).

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2015). "Neural machine translation by jointly learning to align and translate". In: *International Conference on Learning Representations.*

Exercise 19.8: KL leads to softmax

Prove that the optimal solution of (19.2) is given by the softmax in (19.3).

Example 19.9: Popular attention mechanisms

Popular choices for the dissimilarity function dist_k include:

- (negated) dot-product: we have already mentioned this choice $\mathrm{dist}_k(\mathbf{q}, \mathbf{k}) = -\mathbf{q}^\top \mathbf{k}$. We further point out that when normalized, i.e. $\|\mathbf{q}\|_2 = \|\mathbf{k}\|_2 = 1$, then the (negated) dot-product essentially measures the angular distance between the query and the key. This choice is particularly efficient, as we can easily implement the matrix-product $QK^\top$ (for a set of queries and keys) in parallel. One typically sets $\lambda = \sqrt{d_k}$ so that if $\mathbf{q} \perp \mathbf{k} \sim \mathcal{N}(0, I_{d_k})$, then $\mathrm{Var}(\mathbf{q}^\top \mathbf{k}/\sqrt{d_k}) = 1$.

- additive: more generally we may parameterize the dissimilarity function as a feed-forward network $\mathrm{dist}_k(\mathbf{q}, \mathbf{k}; \mathbf{w})$ whose weight vector $\mathbf{w}$ is learned from data.

Remark 19.10: Sparse attention (Sukhbaatar et al. 2019; Correia et al. 2019)

TBD

Sukhbaatar, Sainbayar, Edouard Grave, Piotr Bojanowski, and Armand Joulin (2019). "Adaptive Attention Span in Transformers". In: *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 331–335.

Correia, Gonçalo M., Vlad Niculae, and André F. T. Martins (2019). "Adaptively Sparse Transformers". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 2174–2184.

Exercise 19.11: Self-attention in the limit

Let $V \in \mathbb{R}^{m\times d}$ be arbitrary and consider applying self-attention repeatedly to it:

$$V \leftarrow \mathsf{A}_\lambda(V) := \mathtt{softmax}(VV^\top/\lambda)V,$$

where of course the softmax operator is applied row-wise. What is the limiting behaviour of $\mathsf{A}_\lambda^\infty(V)$?

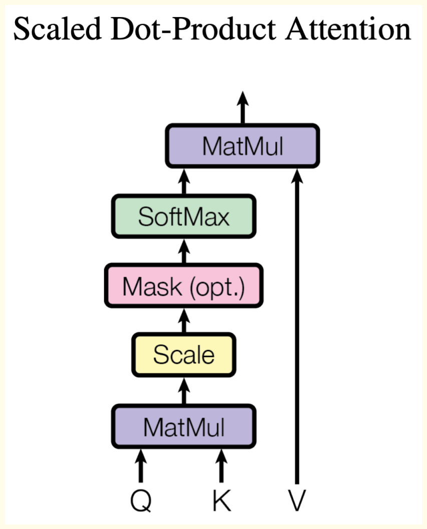
Definition 19.12: Multi-head attention

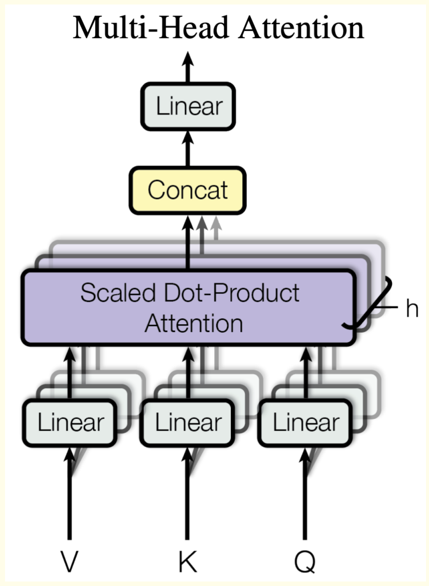
Recall that in CNNs, we employ a number of filters to extract different feature maps. Similarly, we use multi-head attention so that our model can learn to attend to different parts of the input simultaneously:

$$H = [A_1, \ldots, A_h]W, \text{ with } A_i = \mathtt{att}(QW_i^Q; KW_i^K, VW_i^V), \quad i = 1, \ldots, h,$$

where $W \in \mathbb{R}^{(hd_v)\times d}, W_i^Q \in \mathbb{R}^{d\times d_k}, W_i^K \in \mathbb{R}^{d\times d_k}, W_i^V \in \mathbb{R}^{d\times d_v}$ are weight matrices to be learned, and att is an attention mechanism in Definition 19.7 (applied to each row of QW_i^Q; see also Example 19.9).

If we set $d_k = d_v = d/h$, then the number of parameters in h-head attention is on par with single-head attention. The choice of dimensions here also facilitates the implementation of residual connections above.





Remark 19.13: implicit distance learning?

Another way to interpret the linear projections W_i^Q and W_i^K is through distance learning. Indeed, the distance between the query and key, after linear projection, is

$$-(QW_i^Q)(KW_i^K)^\top = -Q\big(W_i^Q(W_i^K)^\top\big)K^\top =: -QM_iK^\top,$$

where the low-rank matrix $M_i \in \mathbb{R}^{d\times d}$ "distorts" the dot-product: $\langle \mathbf{q}, \mathbf{k}\rangle_{M_i} := \mathbf{q}^\top M_i \mathbf{k}$. This explanation suggests tying together W_i^Q, W_i^K, and possibly also W_i^V.

Definition 19.14: Self- and context- attention

The transformer uses multi-head attention along with an encoder-decoder structure, and employs self-attention (e.g. Cheng et al. 2016) as a computationally efficient way to relate different positions in an input sequence:

- encoder self-attention: In this case $Q = K = V$ all come from the input (of the current encoder layer). Each output can attend to all positions in the input.
- context attention: In this case Q comes from the input of the current decoder layer while (K, V) come from the output of (the final layer of) the encoder, namely the context. Again, each output can attend to all positions in the input.
- decoder self-attention: In this case Q comes from the input (of the current decoder layer), $K = Q \odot M$ and $V = Q \odot M$ are masked versions of Q so that each output position can only attend to positions up to and including the current position. In practical implementation we can simply reset *illegal* dissimilarities:

$$\mathrm{dist}_k(\mathbf{q}_i, \mathbf{q}_j) \leftarrow \infty \quad \text{if} \quad i < j.$$

The input to the transformer is a sequence $X = [\mathbf{x}_1, \ldots, \mathbf{x}_m]^\top \in \mathbb{R}^{m\times p}$ and it generates an output sequence $Y = [\mathbf{y}_1, \ldots, \mathbf{y}_l]^\top \in \mathbb{R}^{l\times p}$. Note that we shift the output sequence 1 position to the right, so that combined with the decoder self-attention above, each output symbol only depends on symbols before it (not including itself).

Cheng, Jianpeng, Li Dong, and Mirella Lapata (2016). "Long Short-Term Memory-Networks for Machine Reading". In: *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 551–561.

Definition 19.15: Position-wise feed-forward network

In each encoder and decoder layer, we also apply a feed-forward network at each position. For instance, let $\mathbf{h}_t \in \mathbb{R}^d$ be the (row) representation for position t (at some layer), then we compute

$$\mathtt{FFN}(\mathbf{h}_t) = \sigma(\mathbf{h}_t^\top \mathsf{W}_1)\mathsf{W}_2.$$

The weights $\mathsf{W}_1 \in \mathbb{R}^{d\times 4d}$ and $\mathsf{W}_2 \in \mathbb{R}^{4d\times d}$ are shared among different positions but change from layer to layer.

Remark 19.16: Comparison between attention, RNN and CNN

Layer type	per-layer complexity	sequential operations	max path length
Self-attention	$O(m^2d)$	$O(1)$	$O(1)$
Recurrent	$O(md^2)$	$O(m)$	$O(m)$
Convolution	$O(kmd^2)$	$O(1)$	$O(\log_k m)$
Self-attention (restricted)	$O(rmd)$	$O(1)$	$O(m/r)$

Let m be the length of an input sequence and d the internal representation dimension (as above). In self-attention, the dot-product $QQ^\top$ costs $O(m^2d)$ (recall that $Q \in \mathbb{R}^{m\times d}$). However, this matrix-matrix multiplication can be trivial parallelized using GPUs. We define the maximum path length to be the maximum number of sequential operations for any output position to attend to any input position. Clearly, for the transformer, each output position can attend to each input position hence its maximum path length is $O(1)$.

In contrast, for RNNs, computation is sequential in terms of the tokens in the input sequence. Each recurrence, e.g. $\mathbf{h} \leftarrow W_2\sigma(W_1(\mathbf{h}, \mathbf{x}))$, costs $O(d^2)$, totaling $O(md^2)$. For CNNs (e.g. Gehring et al. 2017) with filter size k, each convolution costs $O(kd)$ and for a single output filter we need to repeat m times while we have d output filters, hence the total cost $O(kmd^2)$. Convolutions can be trivially parallelized on GPUs, and if we employ dilated convolutions (Kalchbrenner et al. 2016) the maximum path length is $O(\log_k m)$ (and $O(m/k)$ for usual convolutions with stride k). Separable convolutions (Chollet 2017) can reduce the per-layer complexity to $O(kmd + md^2)$.

Finally, if we restrict attention to the r neighbors (instead of all m tokens in the input sequence), we may reduce the per-payer complexity to $O(rmd)$, at the cost of increasing the maximum path length to $O(m/r)$.

From the comparison we see that transformer (with restricted attention, if quadratic time/space complexity is a concern) is very suitable for modeling long-range dependencies.

Gehring, Jonas, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin (2017). "Convolutional Sequence to Sequence Learning". In: *Proceedings of the 34th International Conference on Machine Learning*, pp. 1243–1252.

Kalchbrenner, Nal, Lasse Espeholt, Karen Simonyan, Aaron van den Oord, Alex Graves, and Koray Kavukcuoglu (2016). "Neural Machine Translation in Linear Time". In:

Chollet, F. (2017). "Xception: Deep Learning with Depthwise Separable Convolutions". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1800–1807.

Remark 19.17: Attention as interpretation?

One advantage of the attention mechanism is visualization: we can inspect the attention distribution over layers or positions and try to interpret the model; see (Jain and Wallace 2019; Wiegreffe and Pinter 2019) for some discussions.

Jain, Sarthak and Byron C. Wallace (2019). "Attention is not Explanation". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 3543–3556.

Wiegreffe, Sarah and Yuval Pinter (2019). "Attention is not not Explanation". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 11–20.

Definition 19.18: Training using machine translation

Vaswani et al. (2017) trained the transformer using the supervised machine translation problem on the WMT 2014 dataset that consists of pairs (X, Y) of sentences, one (X) from the source language (say English) and the other (Y) from the target language (say French or German). The usual (multi-class) cross-entropy loss is used as the objective function:

$$\min \quad -\hat{\mathsf{E}}\left[\left\langle Y, \log \hat{Y} \right\rangle\right], \quad \hat{Y} = [\hat{\mathbf{y}}_1, \ldots, \hat{\mathbf{y}}_\ell]^\top,$$

where $\hat{\mathbf{y}}_j$ depends on the input sentence $X = [\mathbf{x}_1, \ldots, \mathbf{x}_m]^\top$ and all previous target tokens $Y_{<j} := [\mathbf{y}_1, \ldots, \mathbf{y}_{j-1}]^\top$. Note that the sequence lengths m and l vary from one input to another. In practice, we "bundle" sentence pairs with similar lengths to streamline the parallel computation.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin (2017). "Attention is All you Need". In: *Advances in Neural Information Processing Systems 30*, pp. 5998–6008.

Remark 19.19: Some practicalities about Transformer

We mention some training choices in the original work:

- optimizer: Adam (Kingma and Ba 2015) was used with $\beta_1 = 0.9, \beta_2 = 0.98$ and $\epsilon = 10^{-9}$.
- learning rate:

$$\eta_{\mathtt{iter}} = \tfrac{1}{\sqrt{d}} \cdot \min\{\tfrac{1}{\sqrt{\mathtt{iter}}}, \mathtt{iter} \cdot \tfrac{1}{\tau\sqrt{\tau}}\},$$

  where $\tau = 4000$ controls the warm-up stage where the learning rate increases linearly. After that, the learning rate decreases inverse proportionally to the square root of the iteration number.
- regularization: (a) dropout (with rate 0.1) was added after the positional encoding layer and after each attention layer; (b) residual connection and layer normalization is performed after each attention layer and feed-forward layer; (c) label smoothing (Szegedy et al. 2016):

$$\mathbf{y} \leftarrow (1-\alpha)\mathbf{y} + \alpha\tfrac{1}{C},$$

  where $\mathbf{y}$ is the (original, typically one-hot encoded) label vector, C is the number of classes, and $\alpha = 0.1$ controls the amount of smoothing.

Kingma, D. P. and J. Ba (2015). "Adam: A method for stochastic optimization". In: *International Conference on Learning Representations.*

Szegedy, C., V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna (2016). "Rethinking the Inception Architecture for Computer Vision". In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2818–2826.

Definition 19.20: Beam search during inference time

After we have trained the transformer, during test time we are given an input sentence and asked to decode its translation. We use beam search:

- We keep $b = 4$ currently best candidate translations;
- We augment each candidate translation Y_k with a next word $\mathbf{y}$:

$$\mathtt{score}_k \leftarrow \mathtt{score}_k - \log \hat{P}(\mathbf{y}|X, Y_k).$$

  We may prune the next word by considering only those whose score contribution lies close to the best one (up to some threshold, say b).
- We keep only the best b (in terms of score) augmented translations.
- If some candidate translation ends (either by outputting the stop word or exceeding maximum allowed length, say input length plus 50), we compute its normalized score by dividing its length to the power of $\alpha = 0.6$ (to reduce bias towards shorter translations).
- We prune any candidate translation that lies some threshold (say b) below the best normalized score.

Note that *beam search is highly sequential and subject to possible improvements.*

Example 19.21: Image transformer (Parmar et al. 2018)

Parmar, Niki, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran (2018). "Image Transformer". In: *Proceedings of the 35th International Conference on Machine Learning*, pp. 4055–4064.

Example 19.22: Sparse transformer (Child et al. 2019)

Child, Rewon, Scott Gray, Alec Radford, and Ilya Sutskever (2019). "Generating Long Sequences with Sparse Transformers".

Definition 19.23: Embedding from Language Models (ELMo) (Peters et al. 2018)

Unlike conventional word embeddings that assign each word a *fixed* representation, ELMo is contextualized (Melamud et al. 2016; McCann et al. 2017), where the representation for each word depends on the entire sentence (context) it lies in. ELMo is applied for down-stream tasks in the following way:

- **Two-Stage** (**TS**, not to be confused with few-shot learning in Example 19.34): We fix the parameters in ELMo and use it as an (additional) feature extractor, on top of which we tune a task-specific architecture. For the latter, Peters et al. (2018) also found that stacking ELMo with original token representation $\mathbf{x}$ in its input layer and with its output before passing to a `softmax` layer appears to improve performance.

ELMo trains a bidirectional LM:

$$\min_{\Theta,\Phi} \ \hat{\mathsf{E}} - \log \overrightarrow{\mathsf{p}}(X|\Theta) - \log \overleftarrow{\mathsf{p}}(X|\Phi), \quad \text{where}$$

$$\overrightarrow{\mathsf{p}}(X|\Theta) = \prod_{j=1}^{m} \mathsf{p}(\mathbf{x}_j|\mathbf{x}_1, \ldots, \mathbf{x}_{j-1}; \Theta), \quad \overleftarrow{\mathsf{p}}(X|\Phi) = \prod_{j=1}^{m} \mathsf{p}(\mathbf{x}_j|\mathbf{x}_{j+1}, \ldots, \mathbf{x}_m; \Phi),$$

and the two probabilities are modeled by two LSTMs with shared embedding and `softmax` layers but different hidden parameters.

Given an input sequence, the ELMo representation of any token $\mathbf{x}$ is computed as:

$$\mathtt{ELMo}(\mathbf{x}; \mathsf{A}) = \mathsf{A}(\mathbf{h}_0, \{\overrightarrow{\mathbf{h}}_l\}_{l=1}^L, \{\overleftarrow{\mathbf{h}}_l\}_{l=1}^L) =: \mathsf{A}(\mathbf{h}_0, \{\mathbf{h}_l\}_{l=1}^L), \quad \mathbf{h}_l := [\overrightarrow{\mathbf{h}}_l; \overleftarrow{\mathbf{h}}_l],$$

where recall that the embedding $\mathbf{h}_0$ (e.g. $W_e\mathbf{x}$, or extracted from any context insensitive approach) is shared between the two L-layer LSTMs, whose hidden states are $\overrightarrow{\mathbf{h}}$ and $\overleftarrow{\mathbf{h}}$, respectively. Here A is an aggregation function. Typical choices include:

- Top: $\mathsf{A}(\mathbf{h}_0, \{\overrightarrow{\mathbf{h}}_l\}_{l=1}^L, \{\overleftarrow{\mathbf{h}}_l\}_{l=1}^L) = \mathbf{h}_L$, where only the top layers are retained.
- ELMo as in (Peters et al. 2018):

$$\mathtt{ELMo}(\mathbf{x}; \mathbf{s}, \gamma) = \gamma \sum_{l=0}^{L} s_l \mathbf{h}_l,$$

  where the layer-wise scaling parameters $\mathbf{s}$ and global scaling parameter γ are task-dependent.

ELMo employs 3 layers of representation: context-insensitive embedding through character-level CNN, followed by the forward and backward LSTMs. With $L = 2$, Peters et al. (2018) found that the lower layers tend to learn syntactic information while semantic information is captured in higher layers (through e.g. testing on tasks that require syntactic/semantic information), justifying the aggregation scheme in ELMo.

Peters, Matthew, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, and Luke Zettlemoyer (2018). "Deep Contextualized Word Representations". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 2227–2237.

Melamud, Oren, Jacob Goldberger, and Ido Dagan (2016). "context2vec: Learning Generic Context Embedding with Bidirectional LSTM". In: *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pp. 51–61.

McCann, Bryan, James Bradbury, Caiming Xiong, and Richard Socher (2017). "Learned in Translation: Contextualized Word Vectors". In: *Advances in Neural Information Processing Systems 30*, pp. 6294–6305.

Definition 19.24: Bidirectional Encoder Representation from Transformers (Devlin et al. 2019)

BERT followed up on the pre-training path in GPT, and added some interesting twists:

- The input to BERT is a concatenation of two sequences, starting with a special symbol `[CLS]`, followed by sequence A, the special separator `[SEP]`, sequence B, and ending again with `[SEP]`. (It obviously still works in the absence of a sequence B.) The final representation of `[CLS]` is a sentence-level abstraction and can be used for various downstream sentence classification tasks, while the two-sequence input format can be extremely useful in question answering tasks (e.g. question for A and passage for B).
- Apart from token positional embedding, we also add a sequence positional embedding, where tokens in the first sequence share an embedding vector while those in the second sequence share another one.
- **Masked language model (MLM)**: As in ELMo (Definition 19.23), Devlin et al. (2019) argue that in certain tasks it is important to exploit information from both directions, instead of the left-to-right order in usual language models (LMs, e.g. GPT). Thus, BERT aims at training an MLM: On each input, it randomly replaces 15% tokens with the special symbol `[Mask]`. The modified input then goes through the Transformer encoder where each position can attend to any other position (hence bidirectional). The final hidden representations are passed to a `softmax` layer where we predict only the masked tokens with the usual cross-entropy loss. Note that our predictions on the masked tokens are in parallel, unlike in usual LMs where tokens are predicted sequentially and may affect each other (at test time).
- **Hack**: At test time, the input never includes the special symbol `[Mask]` hence creating a mismatch between training and testing in BERT. To remedy this issue, of the 15% tokens chosen during training, only 80% of them will actually be replaced with `[Mask]`, while 10% of them will be replaced with a random token and the remaining 10% will remain unchanged.

- Next sequence prediction (NSP): Given the first sequence, we choose the second sequence as its next sequence 50% of the time (labeled as `true`) and as a random sequence the remaining time (labeled as `false`). Then, BERT pre-training includes a binary classification loss besides MLM, based on the (binary) `softmax` applied on the final hidden representation of `[CLS]`.
- The training loss of BERT is the sum of averaged MLM likelihood and NSP likelihood.
- **Fine-tuning (FT)**: during fine-tuning, depending on the task we may proceed differently. For sequence classification problems, we add a `softmax` layer to the final hidden representation of `[CLS]`, while for token-level predictions we add `softmax` layers to the final hidden representations of all relevant input tokens. For instance, we add two (independent) `softmax` layers (corresponding to `start` and `end`) in span prediction where during test time we use the approximation

$$\log \Pr(\texttt{start} = i, \texttt{end} = j) \approx \log \Pr(\texttt{start} = i) + \log \Pr(\texttt{end} = j),$$

considering of course only $i \leq j$. Note that all parameters are adjusted in FT, in sharp contrast to the TS approach in ELMo. Of course, BERT can also be applied in the TS setting, where the performance may be slightly worse (Devlin et al. 2019).
- BERT showed that scaling to extreme model sizes leads to surprisingly large improvements on very small scale tasks, provided that the model has been sufficiently pre-trained.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova (2019). "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 4171–4186.

Definition 19.25: XLNet (Yang et al. 2019)

Yang, Zhilin, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le (2019). "XLNet: Generalized Autoregressive Pretraining for Language Understanding". In: *Advances in Neural Information Processing Systems 32*, pp. 5753–5763.

Example 19.26: RoBERTa (Liu et al. 2020)

Liu, Yinhan et al. (2020). "RoBERTa: A Robustly Optimized BERT Pretraining Approach".

Example 19.27: A lite BERT (ALBERT) (Lan et al. 2020)

Lan, Zhenzhong, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut (2020). "ALBERT: A Lite BERT for Self-supervised Learning of Language Representations". In: *International Conference on Learning Representations*.

Remark 19.28: More on BERT

(Joshi et al. 2020; Lewis et al. 2020; Saunshi et al. 2019; Song et al. 2019)

Joshi, Mandar, Danqi Chen, Yinhan Liu, Daniel S. Weld, Luke Zettlemoyer, and Omer Levy (2020). "SpanBERT: Improving Pre-training by Representing and Predicting Spans". *Transactions of the Association for Computational Linguistics*, vol. 8, pp. 64–77.

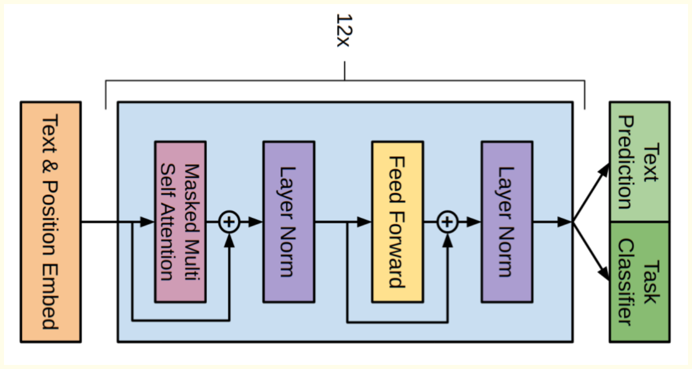
Lewis, Mike, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer (2020). "BART: Denoising Sequence-to-Sequence Pre-training for Natural Language

Generation, Translation, and Comprehension". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 7871–7880.

Saunshi, Nikunj, Orestis Plevrakis, Sanjeev Arora, Mikhail Khodak, and Hrishikesh Khandeparkar (2019). "A Theoretical Analysis of Contrastive Unsupervised Representation Learning". In: *Proceedings of the 36th International Conference on Machine Learning*, pp. 5628–5637.

Song, Kaitao, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu (2019). "MASS: Masked Sequence to Sequence Pre-training for Language Generation". In: *Proceedings of the 36th International Conference on Machine Learning*, pp. 5926–5936.

Definition 19.29: Generative Pre-Training (GPT) (Radford et al. 2018)



GPT works in two stages:

- unsupervised pre-training through learning a language model:

$$\min_{\Theta} \quad -\hat{\mathsf{E}} \log \mathsf{p}(X|\Theta), \quad \text{where} \quad \mathsf{p}(X|\Theta) = \prod_{j=1}^{m} \mathsf{p}(\mathbf{x}_j|\mathbf{x}_1, \ldots, \mathbf{x}_{j-1}; \Theta).$$

  Namely, given the context consisting of previous tokens $\mathbf{x}_1, \ldots, \mathbf{x}_{j-1}$, we aim to predict the current token $\mathbf{x}_j$. The conditional probability is computed through a multi-layer transformer decoder (Liu et al. 2018):

$$\begin{aligned} H^{(0)} &= XW_e + W_p \\ H^{(\ell)} &= \mathtt{transformer\_decoder\_block}(H^{(\ell-1)}), \quad \ell = 1, \ldots, L \\ \mathsf{p}(\mathbf{x}_j|\mathbf{x}_1, \ldots, \mathbf{x}_{j-1}; \Theta) &= \mathtt{softmax}(\mathbf{h}_j^{(L)} W_e^\top), \end{aligned}$$

  where $X = [\mathbf{x}_1, \ldots, \mathbf{x}_m]^\top$ is the input sequence consisting of m tokens (m may vary from input to input), L is the number of transformer blocks, W_e is the token embedding matrix and W_p is the position embedding matrix.

- supervised fine-tuning with task-aware input transformations:

$$\min_{W_y} \min_{\Theta} \quad -\hat{\mathsf{E}} \log \mathsf{p}(\mathbf{y}|X, \Theta) - \lambda \cdot \hat{\mathsf{E}} \log \mathsf{p}(X|\Theta), \quad \text{where} \quad \mathsf{p}(\mathbf{y}|X, \Theta) = \left\langle \mathbf{y}, \mathtt{softmax}(\mathbf{h}_m^{(L)} W_y) \right\rangle,$$

  and we include the unsupervised pre-training loss to help improving generalization and accelerating convergence. Note that for different tasks, we only add an extra `softmax` layer to do the classification. Unlike ELMo, GPT avoids task-specific architectures and aims to learn a *universal* representation (through language model pre-training) for *all* tasks.

Sequence pre-training has been explored earlier in (Dai and Le 2015; Howard and Ruder 2018) using LSTMs.

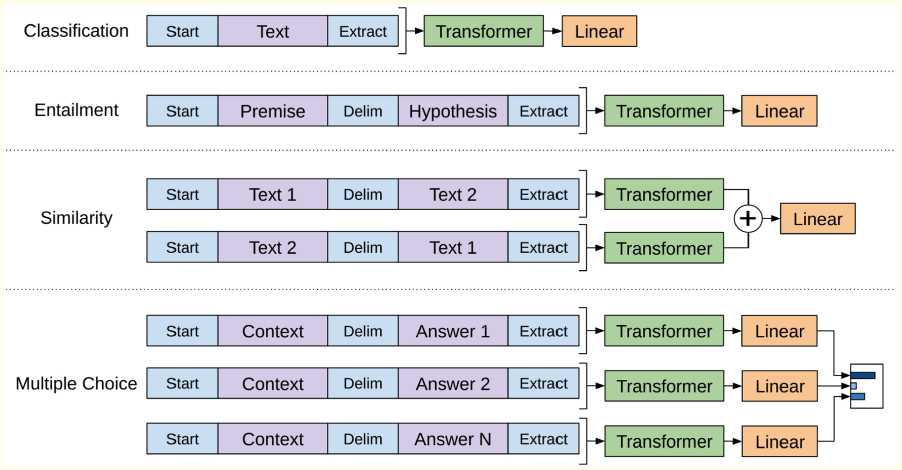
Radford, Alec, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever (2018). "Improving Language Understanding by generative pre-training".

Liu, Peter J., Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer (2018). "Generating Wikipedia by Summarizing Long Sequences". In: *International Conference on Learning Representations*.

Dai, Andrew M and Quoc V Le (2015). "Semi-supervised Sequence Learning". In: *Advances in Neural Information Processing Systems 28*, pp. 3079–3087.

Howard, Jeremy and Sebastian Ruder (2018). "Universal Language Model Fine-tuning for Text Classification". In: *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pp. 328–339.

Remark 19.30: Input transformations



GPT employs task-specific input transformations so that its fine-tuning strategy in Definition 19.29 is applicable:

- **Textual entailment**: Given `premise`, decide if `hypothesis` holds. We simply concatenate `premise` with `hypothesis`, delimited by a special token, as input to GPT and reduce to 3-class classification: `entailment, neutral, contradiction\verb`.

- **Similarity**: Similarly, we concatenate the two input sentences in both orders and add the resulting representation.

- **Question answering and reasoning**: We concatenate the context, question and each possible answer to obtain multiple input sequences and reduce to multi-class classification.

Remark 19.31: Some practicalities about GPT

We mention the following implementation choices in GPT:

- Byte pair encoding (BPE): Current character-level language models (LMs) (e.g. Gillick et al. 2016) are not as competitive as word-level LMs (e.g. Al-Rfou et al. 2019). BPE (e.g. Sennrich et al. 2016) provides a practical middle ground, where we start with UTF-8 *bytes* (256 base vocabs instead of the overly large 130k codes) and repeatedly merge pairs with the highest frequency in our corpus. We prevent merging between different (UTF-8 code) categories, with an exception on spaces (to reduce similar vocabs such as `dog` and `dog!` but allow `dog cat`). BPE (or any other character-level encoding) allows us to compute probabilities even over words and sentences that are not seen at training.

- Gaussian Error Linear Unit (GELU) (Hendrycks and Gimpel 2016):

$$\sigma(x) = x\Phi(x) = \mathsf{E}(x \cdot m|x), \quad \text{where} \quad m \sim \text{Bernoulli}(\Phi(x)),$$

  i.e., on the input x, with probability $\Phi(x)$ we drop it. Unlike Relu where we drop any negative input, GELU drops the input more probably as the latter decreases.

- fine-tuning uses a smaller batch size 32, and converges after 3 epochs in most cases. $\lambda = \frac{1}{2}$; warm-up $\tau = 2000$ during pre-training and 0.2% during fine-tuning.

Gillick, Dan, Cliff Brunk, Oriol Vinyals, and Amarnag Subramanya (2016). "Multilingual Language Processing From Bytes". In: *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 1296–1306.

Al-Rfou, Rami, Dokook Choe, Noah Constant, Mandy Guo, and Llion Jones (2019). "Character-Level Language Modeling with Deeper Self-Attention". In: *AAAI*.

Sennrich, Rico, Barry Haddow, and Alexandra Birch (2016). "Neural Machine Translation of Rare Words with Subword Units". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, pp. 1715–1725.

Hendrycks, Dan and Kevin Gimpel (2016). "Gaussian Error Linear Units (GELUs)".

Alert 19.32: "What I cannot create, I do not understand." — Richard Feynman

A popular hypothesis to support pre-training (through say language modeling) is that the underlying generative model learns to perform many of the downstream (supervised) tasks in order to improve its language modeling capability, and the long-range dependencies allowed by transformers assists in transfer compared to LSTMs (or other RNNs).

Example 19.33: GPT-2 (Radford et al. 2019)

Most current ML systems are narrow in the sense that they specialize exclusively on a single task. This approach, however successful, suffers from generalizing to other domains/tasks that are not encountered during training. While unsupervised pre-training combined with supervised fine-tuning (such as in GPT) proves effective, GPT-2 moves on to the completely unsupervised zero-shot setting.

McCann et al. (2018) showed that many NLP tasks (input and output) can be specified purely by language, and hence can be solved through training a *single* model. Indeed, the target output to a natural language task is just one of the many possible *next sentences*. Thus, training a sufficiently large *unsupervised* language model may allow us to perform well on a range of tasks (not explicitly trained with supervision), albeit in a much less data-efficient way perhaps.

GPT-2 also studied the effect of test set contamination, where the large training set accidentally includes near-duplicates of part of the test data. For example CIFAR-10 has 3.3% overlap between train and test images (Barz and Denzler 2020).

Apart from some minor adjustments, the main upgrade from GPT to GPT-2 is a (sharp) increase in model capacity, a larger training set, and the exclusive focus on zero-shot learning. Conceptually, GPT-2 demonstrated the (surprising?) benefit of training (excessively?) large models, with near-human performance on some NLP tasks (such as text generation).

Radford, Alec, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever (2019). "Language models are unsupervised multitask learners".

McCann, Bryan, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher (2018). "The Natural Language Decathlon: Multitask Learning as Question Answering".

Barz, Björn and Joachim Denzler (2020). "Do We Train on Test Data? Purging CIFAR of Near-Duplicates". *Journal of Imaging*, vol. 6, no. 41, pp. 1–8.

Example 19.34: GPT-3 (Brown et al. 2020)

The main contribution of GPT-3 is perhaps its crystallization of different evaluation schemes:

- Fine-Tuning (FT): this was explored in GPT-1 and involves task-dedicated datasets and gradient updates on both the LM and the task-dependent classifier.

- Few-Shot (FS): a natural language description of the task (e.g. `Translate English to French`) and a few example demonstrations (e.g. English-French sentence pairs), followed by a final context (e.g. English sentence) that will be completed by GPT-3 (e.g. a French translation).
- One-Shot (1S): same as above with the number of demonstrations restricted to 1.
- Zero-Shot (0S): just the task description (in natural language form) is provided.

GPT-3 focuses exclusively on the last 3 scenarios and never performs gradient updates on the LM. The Common Crawl dataset that GPT-3 was built on is so large that no sequence was ever updated twice during training.

Brown, Tom B. et al. (2020). "Language Models are Few-Shot Learners".

Example 19.35: Pixel GPT (Chen et al. 2020)

Chen, Mark, Alec Radford, Rewon Child, Jeffrey K Wu, Heewoo Jun, David Luan, and Ilya Sutskever (2020). "Generative Pretraining From Pixels". In: *Proceedings of the 37th International Conference on Machine Learning*.

Alert 19.36: The grand comparison

model	data	en	de	input	embedding	heads	params	batch	steps	GPUs	time
transformer	WMT14	6	6	-	512	8	213M	50k tokens	100k	8×P100	12h
sparseTran											
imageTran											
GPT-i											
ELMo	1B Word Bench	-	2×2	2048c	512	-	?	?	10e	?	?
GPT-1	BooksCorp	0	12	512×40k	768	12	100M	64	100e	8×P600	1m
BERT$_{\text{base}}$	BooksCorp+Wiki	12	0	512×30k	768	12	110M	256	1M	4×cTPU	4d
BERT$_{\text{large}}$		24			1024	16	340M			16×cTPU	
GPT-2	WebText	0	12	1024× 50k	768	12	117M	512	?	?	?
			24		1024		345M				
			36		1280		762M				
			48		1600		1542M				
RoBERTa											
XLNet											
ALBERT											
GPT-3-S	CommonCrawl	0	12	2048× 50k	768	12	125M	0.5M	?	?×V100	?
GPT-3-M			24		1024	16	350M	0.5M			
GPT-3-L			24		1536	16	760M	0.5M			
GPT-3-XL			24		2048	24	1.3B	1M			
GPT-3-SS			32		2560	32	2.7B	1M			
GPT-3-MM			32		4096	32	6.7B	2M			
GPT-3-LL			40		5140	40	13B	2M			
GPT-3			96		12288	96	175B	3.2M			

20 Learning to Learn

Goal

A general introduction to the recent works on meta-learning.

Alert 20.1: Convention

Gray boxes are not required hence can be omitted for unenthusiastic readers.

This note is likely to be updated again soon.

Earlier developments can be found in the monograph (Thrun and Pratt 1998) while recent ones can be found in the survey article (Hospedales et al. 2020).

Thrun, Sebastian and Lorien Pratt (1998). "Learning to Learn: Introduction and Overview". In: *Learning to Learn*. Springer.

Hospedales, Timothy, Antreas Antoniou, Paul Micaelli, and Amos Storkey (2020). "Meta-Learning in Neural Networks: A Survey".

Definition 20.2: c-class and s-shot (cs)

In few-shot learning, we are given cs training examples, with s shots (examples) per each of the c classes, and we are asked to classify new test examples into the c classes. Typically, both c and s are small (e.g. $c = 5$ or $c = 10$, $s = 5$ or $s = 1$ or even $s = 0$). Training a deep model based on the cs training examples can easily lead to *severe overfitting*.

Algorithm 20.3: kNN in few-shot learning

A common algorithm in the above c-class and s-shot setting is the k nearest neighbor algorithm discussed in Section 5. In addition, we may train a feature extractor (such as BERT or GPT) based on unlabeled data (e.g. images or documents depending on the application) collected from the internet, and apply kNN (with the labeled cs training examples) in the feature space.

Recall that there is no learning but memorization (of the training set) in kNN.

Definition 20.4: Matching network (Vinyals et al. 2016)

Matching network (Vinyals et al. 2016) (along with many precursors) is a simple extension of the above kNN algorithm. It makes the prediction $\hat{\mathbf{y}}$ on a new test sample $\mathfrak{x}$ after seeing a few shots $\mathcal{S} = \wr(\mathbf{x}_i, \mathbf{y}_i), i = 1, \ldots, m\int$ as follows:

$$\hat{\mathbf{y}}(\mathfrak{x}; \mathbf{w}|\mathcal{S}) = \sum_{i=1}^{m} \mathfrak{a}_{\mathbf{w}}(\mathfrak{x}, \mathbf{x}_i|\mathcal{S}) \cdot \mathbf{y}_i, \qquad (20.1)$$

where the attention mechanism $\mathfrak{a} : \mathsf{X} \times \mathsf{X} \to \mathbb{R}_+$ measures the similarity between its inputs and is learned parametrically. (For classification problems we interpret $\hat{\mathbf{y}}$ as the confidence, as usual.) The idea to *parameterize a non-parametric model is prevalent in ML* (e.g. neural nets or kernel learning).

Vinyals, Oriol, Charles Blundell, Timothy Lillicrap, Koray Kavukcuoglu, and Daan Wierstra (2016). "Matching Networks for One Shot Learning". In: *Advances in Neural Information Processing Systems 29*, pp. 3630–3638.

Exercise 20.5: kNN as attention

With what fixed choice of the attention mechanism can we reduce (20.1) to kNN?

Alert 20.6: Like father, like son

An ML algorithm is good at tasks that we train them to be good at[a]. Thus, in the few-shot setting it is natural to consider meta-training (Vinyals et al. 2016), to reflect the challenges that will be encountered in the test phase. Specifically, we define a task (a.k.a. episode) $\mathcal{T}$ to consist of $m = cs$ training examples $\mathcal{S} := \{(\mathbf{x}_i, \mathbf{y}_i), i = 1, \ldots, m\}$ and r test examples $\mathcal{R} = \{(\mathfrak{x}_\iota, \mathfrak{y}_\iota), \iota = 1, \ldots, r\}$ from c sampled classes:

$$\min_{\mathbf{w}} \ \mathsf{E}_{\mathcal{T}=(\mathcal{S},\mathcal{R})} \ \underbrace{\mathsf{E}_{(\mathfrak{x},\mathfrak{y})\sim\mathcal{R}} \ \ell\big(\mathfrak{y}, \overbrace{\hat{\mathbf{y}}(\mathfrak{x}; \mathbf{w}|\mathcal{S})}^{\text{prediction based on training data}}\big)}_{\text{mimick the test loss}},$$

where ℓ is some loss function (e.g. cross-entropy for classification). Note that the c classes may be different for different tasks $\mathcal{T}$. In fact, in more challenged evaluations, it is typical to use disjoint classes in the test phase than those appeared in the above training phase.

After training, we have two choices:

- directly apply the model, *without any further fine-tuning*, to new few-shot settings that may involve novel classes that were never seen before. Of course, we expect the performance to degrade as the novel classes become increasingly different from the ones appeared in training;
- fine-tune the model using the given $m = cs$ training examples before applying to test examples, at the risk of potential severe overfitting.

Vinyals, Oriol, Charles Blundell, Timothy Lillicrap, Koray Kavukcuoglu, and Daan Wierstra (2016). "Matching Networks for One Shot Learning". In: *Advances in Neural Information Processing Systems 29*, pp. 3630–3638.

[a]There is of course some mild generalization beyond memorization, but one should not take it too far: we simply cannot expect an algorithm trained on dogs to recognize cats, unless we incorporate *additional mechanism* to make it even plausible.

Example 20.7: Instantiation of Matching Network

The simplest parameterization of the attention mechanism in (20.1) is through learning an embedding (representation): Let $f(\mathbf{x}; \mathbf{w}) = g(\mathbf{x}; \mathbf{w})$ be a feature extractor (e.g. deep network with weights $\mathbf{w}$), and set

$$\mathfrak{a}_{\mathbf{w}}(\mathfrak{x}, \mathbf{x}_i) := \mathtt{softmax}\Big(-\operatorname{dist}\big(f(\mathfrak{x}; \mathbf{w}), f(X; \mathbf{w})\big)\Big), \quad X = [\mathbf{x}_1, \ldots, \mathbf{x}_m], \ f(X; \mathbf{w}) = [f(\mathbf{x}_1; \mathbf{w}), \ldots, f(\mathbf{x}_m; \mathbf{w})].$$

Vinyals et al. (2016) also considered the more complicated parameterization:

$$\mathfrak{a}_{\mathbf{w}}(\mathfrak{x}, \mathbf{x}_i) := \mathtt{softmax}\Big(-\operatorname{dist}\big(f(\mathfrak{x}; X, \boldsymbol{\theta}), g(X; \boldsymbol{\phi})\big)\Big), \quad \mathbf{w} = [\boldsymbol{\theta}, \boldsymbol{\phi}],$$

where dist is the cosine distance and f and g are two embedding networks with tunable parameters $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$, respectively. We choose a bidirectional LSTM for g where the examples X are treated as the input sequence (so that the embedding of $\mathbf{x}_i$ depends on the entire set X). For f, we parameterize it as a memory-enhanced LSTM:

$$[\mathbf{h}_k, \mathbf{c}_k] = \mathtt{LSTM}(f_0(\mathfrak{x}), [\mathbf{h}_{k-1}, \mathbf{r}_{k-1}], \mathbf{c}_{k-1})$$

$$\mathbf{r}_k = \underbrace{g(X)}_{\text{memory}} \cdot \underbrace{\mathtt{softmax}\Big(g(X)^\top \big(\overbrace{\mathbf{h}_k + f_0(\mathfrak{x})}^{\text{residual}}\big)\Big)}_{\text{attention weights}}, k = 1, \ldots, K,$$

where $f_0(\mathfrak{x})$ is the input embedding of $\mathfrak{x}$, $\mathbf{h}_k$ is the LSTM state after k recurrence, $\mathbf{c}_k$ is the cell state of LSTM, and $g(X) = g(X; \boldsymbol{\phi}) = [g(\mathbf{x}_1; \boldsymbol{\phi}), \ldots, g(\mathbf{x}_m; \boldsymbol{\phi})]$ acts as memory while $\mathbf{r}_k$ is the read from memory (using again an attention mechanism). We set $f(\mathfrak{x}; X, \boldsymbol{\theta}) = \mathbf{h}_K$.

Vinyals, Oriol, Charles Blundell, Timothy Lillicrap, Koray Kavukcuoglu, and Daan Wierstra (2016). "Matching Networks for One Shot Learning". In: *Advances in Neural Information Processing Systems 29*, pp. 3630–3638.

Definition 20.8: Prototypical Network (Snell et al. 2017)

Prototypical network is a drastic simplification of the matching network, by first grouping training examples (in the feature space) from the same class before passing to `softmax`:

$$\min_{\mathbf{w}} \ \mathsf{E}_{\mathcal{T}=(\mathcal{S},\mathcal{R})} \underbrace{\mathsf{E}_{(\mathfrak{x},\mathfrak{y})\sim\mathcal{R}}\, \ell\big(\mathfrak{y}, \overbrace{\hat{\mathbf{y}}(\mathfrak{x};\mathbf{w}|\mathcal{S})}^{\text{prediction based on training data}}\big)}_{\text{mimick the test loss}}, \quad \text{where} \quad \hat{\mathbf{y}}(\mathfrak{x};\mathbf{w}|\mathcal{S}) = \sum_{k=1}^{c} \mathfrak{a}\big(f(\mathfrak{x};\mathbf{w}), \bar{f}(\mathcal{S}_k;\mathbf{w})\big)\cdot \mathbf{e}_k,$$

$\mathcal{S}_k := \{\mathbf{x}_i : \mathbf{y}_i = \mathbf{e}_k\}$ is the set of training examples from the k-th class. As usual,

$$\mathfrak{a}\big(f(\mathfrak{x};\mathbf{w}), \bar{f}(\mathcal{S}_k;\mathbf{w})\big) \propto \exp\Big(-\operatorname{dist}\big(f(\mathfrak{x};\mathbf{w}), \bar{f}(\mathcal{S}_k;\mathbf{w})\big)\Big),$$

while we use the center (i.e. mean) of each class as its prototype:

$$\bar{f}(\mathcal{S}_k;\mathbf{w}) = \frac{1}{|\mathcal{S}_k|}\sum_{\mathbf{x}\in\mathcal{S}_k} f(\mathbf{x};\mathbf{w}).$$

Snell et al. (2017) showed that the (squared) Euclidean distance $\operatorname{dist}(\mathfrak{x},\mathbf{x}_i) := \|\mathfrak{x}-\mathbf{x}_i\|_2^2$ performs better than the cosine distance in matching networks. It is empirically observed that using more classes during training while maintaining the same number of shots tends to improve test performance.

Mensink et al. (2013) and Rippel et al. (2016) also considered using multiple prototypes (e.g. mixture model) for each class, providing an interpolation between the prototypical and the matching network.

Snell, Jake, Kevin Swersky, and Richard Zemel (2017). "Prototypical Networks for Few-shot Learning". In: *Advances in Neural Information Processing Systems 30*, pp. 4077–4087.

Mensink, T., J. Verbeek, F. Perronnin, and G. Csurka (2013). "Distance-Based Image Classification: Generalizing to New Classes at Near-Zero Cost". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 11, pp. 2624–2637.

Rippel, Oren, Manohar Paluri, Piotr Dollar, and Lubomir Bourdev (2016). "Metric Learning with Adaptive Density Discrimination". In: *ICLR*.

Definition 20.9: Archetypes vs. prototypes (Cutler and Breiman 1994)

Cutler and Breiman (1994) proposed the following archetype analysis: Given training data $X = [\mathbf{x}_1,\ldots,\mathbf{x}_n]$, we solve

$$\min_{U\in\mathbb{R}_+^{n\times r}, V\in\mathbb{R}_+^{r\times n}} \ \ell(X, XUV), \qquad \text{s.t.} \quad U^\top\mathbf{1}=\mathbf{1}, V^\top\mathbf{1}=\mathbf{1}, \tag{20.2}$$

where $Z = [\mathbf{z}_1,\ldots,\mathbf{z}_r] := XU$ are called archetypes. The objective in (20.2) aims at recovering each training point $\mathbf{x}_i$ through convex combinations (with weights $\mathbf{v}_i$) of the archetypes, which are themselves convex combinations of the original data points. In fact, at optimality we may choose archetypes to lie on the boundary of the convex hull of training points, and many of them tend to be extremal (especially for large r). We can solve (20.2) by alternating minimization, although Cutler and Breiman (1994) pointed out that local minima increases with the number r of archetypes. *Archetypes also tend to be non-robust under the least squares objective.*

Archetypes have the advantage of being quite interprettable. We may display them and analyze (the distribution of) the reconstruction weights $\mathbf{v}_i$ for each training point $\mathbf{x}_i$.

Cutler, Adele and Leo Breiman (1994). "Archetypal Analysis". *Technometrics*, vol. 36, no. 4, pp. 338–347.

Definition 20.10: Model-agnostic meta-learning (MAML) (Finn et al. 2017)

Both matching and prototypical networks are some variations of (soft) kNN, which does not require any training at the classifier level (although we do learn a versatile representation). Recall our meta-training objective:

$$\min_{\mathbf{w}} \ \mathsf{E}_{\mathcal{T}=(\mathcal{S},\mathcal{R})} \underbrace{\mathsf{E}_{(\mathfrak{x},\mathfrak{y})\sim\mathcal{R}} \ \ell\big(\mathfrak{y}, \overbrace{\hat{\mathbf{y}}(\mathfrak{x};\mathbf{w}|\mathcal{S})}^{\text{prediction after fine-tuning}}\big)}_{\text{mimick the test loss}},$$

where $\hat{\mathbf{y}}(\mathfrak{x};\mathbf{w}|\mathcal{S})$ now can be any prediction rule based on training data $\mathcal{S}$ and model initializer $\mathbf{w}$. In particular, MAML (Finn et al. 2017) proposed the following instantiation:

$$\hat{\mathbf{y}}(\mathfrak{x};\mathbf{w}|\mathcal{S}) = \hat{\mathbf{y}}(\mathfrak{x};\tilde{\mathbf{w}}), \quad \tilde{\mathbf{w}} = \mathsf{T}_{\eta,\mathcal{S}}(\mathbf{w}) := \mathbf{w} - \eta\nabla L(\mathbf{w};\mathcal{S}), \quad L(\mathbf{w};\mathcal{S}) := \hat{\mathsf{E}}_{(\mathbf{x},\mathbf{y})\sim\mathcal{S}} \ \ell(\mathbf{y}, \hat{\mathbf{y}}(\mathbf{x};\mathbf{w})) \tag{20.3}$$

where $\hat{\mathbf{y}}(\mathbf{x};\mathbf{w})$ is a chosen prediction rule (e.g. `softmax` for classification or linear for regression). In other words, based on the training data $\mathcal{S}$ we update the model parameter $\mathbf{w}$ to $\tilde{\mathbf{w}}$, which is then applied on the test set $\mathcal{R}$ to compute a loss for $\mathbf{w}$, mimicking completely what one would do during fine-tuning. Of course, we may change the one-step gradient in (20.3) to any update rule, such as k-step gradient. Plugging everything in we obtain:

$$\min_{\mathbf{w}} \ \mathsf{E}_{\mathcal{T}=(\mathcal{S},\mathcal{R})} \underbrace{\mathsf{E}_{(\mathfrak{x},\mathfrak{y})\sim\mathcal{R}} \ \ell\Big(\mathfrak{y}, \overbrace{\hat{\mathbf{y}}\Big(\mathfrak{x}; \overbrace{\mathbf{w} - \eta\hat{\mathsf{E}}_{(\mathbf{x},\mathbf{y})\sim\mathcal{S}} \ \nabla\ell\big(\mathbf{y}, \hat{\mathbf{y}}(\mathbf{x};\mathbf{w})\big)}^{\text{mimick fine-tuning using training data}}\Big)}^{\text{mimick prediction on test set after fine-tuning}}\Big)}_{\text{mimick the loss on test set}},$$

MAML performs (stochastic) gradient update on the above objective, which requires computing the Hessian $\nabla^2\ell$ (vector product). Empirically, omitting the Hessian part seems to not impair the performance noticeably.

Importantly, MAML imposes no restriction on the prediction rule $\hat{\mathbf{y}}(\mathbf{x};\mathbf{w})$ (as long as we can backpropagate gradient), and it aims at learning a representation (or initialization), which will be further fine-tuned at test time (by running the update $\mathsf{T}_{\eta,\mathfrak{S}}(\mathbf{w})$ various steps on new shots $\mathfrak{S}$ in test set). In contrast, fine-tuning is optional in both matching and prototypical networks (thanks to their kNN nature).

Finn, Chelsea, Pieter Abbeel, and Sergey Levine (2017). "Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks". In: *Proceedings of the 34th International Conference on Machine Learning*, pp. 1126–1135.

Definition 20.11: Learning to optimize few-shot learning (Ravi and Larochelle 2017)

Ravi and Larochelle (2017) noted that the (stochastic) gradient update we typically use resembles the update for the cell state in an LSTM:

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{c}}_t,$$

where $\mathbf{f}_t \equiv 1$, $\mathbf{c}_{t-1} = \mathbf{w}_{t-1}$, $\mathbf{i}_t = \alpha_t$, and $\tilde{\mathbf{c}}_t = -\nabla L_t$. Inspired by (Andrychowicz et al. 2016), we can thus employ an LSTM to parameterize our optimization algorithm:

$$\begin{aligned}
\mathbf{i}_t &= \sigma(U[\nabla L_t, L_t, \mathbf{w}_{t-1}, \mathbf{i}_{t-1}]) \\
\mathbf{f}_t &= \sigma(V[\nabla L_t, L_t, \mathbf{w}_{t-1}, \mathbf{f}_{t-1}]) \\
\mathbf{w}_t &= \mathbf{c}_t.
\end{aligned}$$

The parameters U, V of the LSTM are learned using the meta-training objective.

Ravi, Sachin and Hugo Larochelle (2017). "Optimization as a Model for Few-Shot Learning". In: *ICLR*.

Andrychowicz, Marcin, Misha Denil, Sergio Gómez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando de Freitas (2016). "Learning to learn by gradient descent by gradient descent". In: *Advances in Neural Information Processing Systems 29*, pp. 3981–3989.