Optimization for Data Science

Lec 00: Introduction

Yaoliang Yu

UNIVERSITY OF WATERLOO | FACULTY OF MATHEMATICS
DAVID R. CHERITON SCHOOL OF COMPUTER SCIENCE

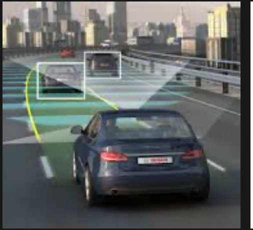
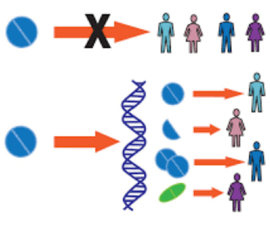
Course Information

- Instructor: Yao-Liang Yu (yaoliang.yu@uwaterloo.ca)
- Website: cs.uwaterloo.ca/~y328yu/teaching/794
- Prerequisites: Basic linear algebra, calculus, probability, algorithm
- Textbooks: No required textbook
  - if interested, see course website for some further readings
- Notes and slides are posted on the course website

Machine Learning is Everywhere

- Everyone uses ML everyday

- Lots of cool applications

- Excellent for job-hunting

And More

Ill. Niklas Elmehed © Nobel Prize Outreach

John J. Hopfield

Prize share: 1/2

Ill. Niklas Elmehed © Nobel Prize Outreach

Geoffrey Hinton

Prize share: 1/2

The Nobel Prize in Physics 2024 was awarded jointly to John J. Hopfield and Geoffrey E. Hinton "for foundational discoveries and inventions that enable machine learning with artificial neural networks"

Ill. Niklas Elmehed © Nobel Prize Outreach

David Baker

Prize share: 1/2

Ill. Niklas Elmehed © Nobel Prize Outreach

Demis Hassabis

Prize share: 1/4

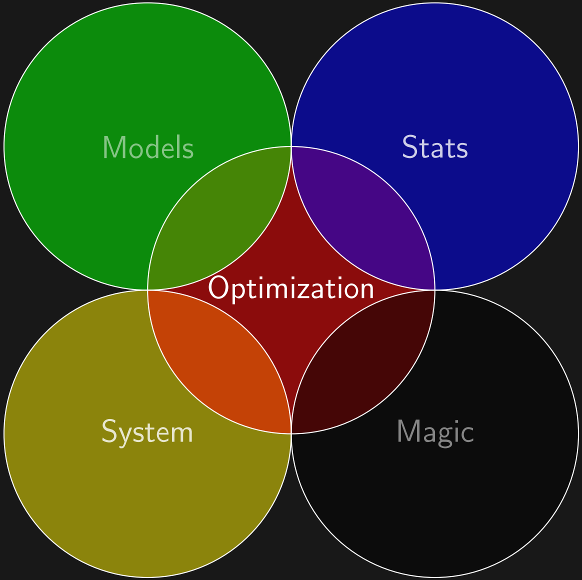
Ill. Niklas Elmehed © Nobel Prize Outreach

John Jumper

Prize share: 1/4

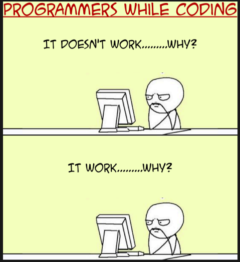
The Nobel Prize in Chemistry 2024 was divided, one half awarded to David Baker "for computational protein design", the other half jointly to Demis Hassabis and John Jumper "for protein structure prediction"

At the Core is Optimization



What You Will Learn

- Learn the basic theory and algorithms
- Gain some implementation experience
- Know when to use which algorithm with what guarantees
- Start to formulate problems with algorithms in mind



Let the Journey Begin

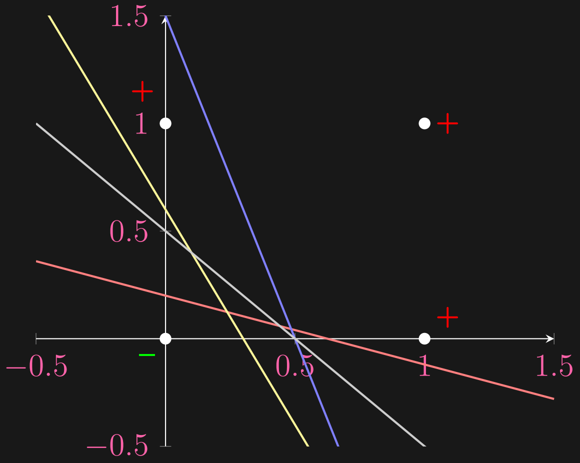
What a Dataset Looks Like

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$	$\cdots$	$\mathbf{x}_n$	$\mathbf{x}$	$\mathbf{x}'$
$\mathbb{R}^d \ni$	0	1	0	1	$\cdots$	1	1	0.9
	0	0	1	1	$\cdots$	0	1	1.1
	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$	$\vdots$
	1	0	1	0	$\cdots$	1	1	-0.1
y	+	+	−	+	$\cdots$	−	?	?!

- each column is a data point: n in total; each has d features
- bottom y is the label vector; binary in this case
- $\mathbf{x}$ and $\mathbf{x}'$ are test samples whose labels need to be predicted

OR Dataset

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$
	0	1	0	1
	0	0	1	1
y	−	+	+	+



The Early Hype in AI...

NEW NAVY DEVICE LEARNS BY DOING

Psychologist Shows Embryo of Computer Designed to Read and Grow Wiser

WASHINGTON, July 7 (UPI) —The Navy revealed the embryo of an electronic computer today that it expects will be able to walk, talk, see, write, reproduce itself and be conscious of its existence.

The embryo—the Weather Bureau's $2,000,000 "704" computer—learned to differentiate between right and left after fifty attempts in the Navy's demonstration for newsmen.

The service said it would use this principle to build the first of its Perceptron thinking machines that will be able to read and write. It is expected to be finished in about a year at a cost of $100,000.

Dr. Frank Rosenblatt, designer of the Perceptron, conducted the demonstration. He said the machine would be the first device to think as the human brain. As do human beings, Perceptron will make mistakes at first, but will grow wiser as it gains experience, he said.

Dr. Rosenblatt, a research psychologist at the Cornell Aeronautical Laboratory, Buffalo, said Perceptrons might be fired to the planets as mechanical space explorers.

Without Human Controls

The Navy said the perceptron would be the first non-living mechanism "capable of receiving, recognizing and identifying its surroundings without any human training or control."

The "brain" is designed to remember images and information it has perceived itself. Ordinary computers remember only what is fed into them on punch cards or magnetic tape.

Later Perceptrons will be able to recognize people and call out their names and instantly translate speech in one language to speech or writing in another language, it was predicted.

Mr. Rosenblatt said in principle it would be possible to build brains that could reproduce themselves on an assembly line and which would be conscious of their existence.

In today's demonstration, the "704" was fed two cards, one with squares marked on the left side and the other with squares on the right side.

Learns by Doing

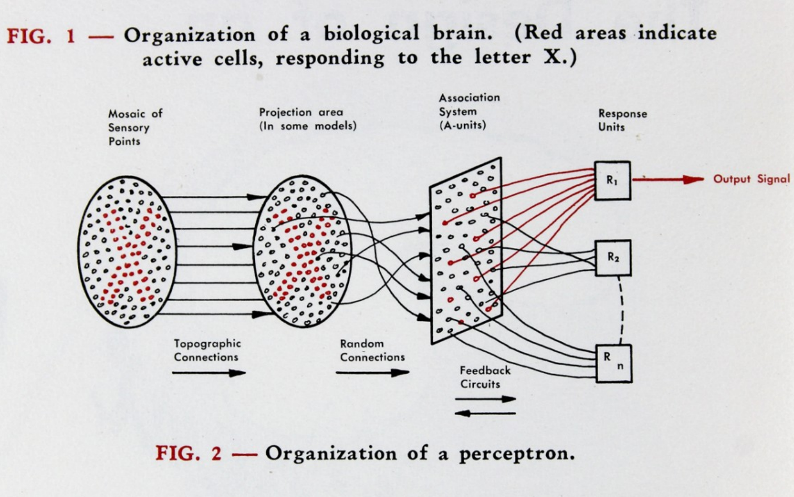
In the first fifty trials, the machine made no distinction between them. It then started registering a "Q" for the left squares and "O" for the right squares.

Dr. Rosenblatt said he could explain why the machine learned only in highly technical terms. But he said the computer had undergone a "self-induced change in the wiring diagram."

The first Perceptron will have about 1,000 electronic "association cells" receiving electrical impulses from an eye-like scanning device with 400 photo-cells. The human brain has 10,000,000,000 responsive cells, including 100,000,000 connections with the eyes.

New York Times, 1958

...due to Perceptron



Frank Rosenblatt
(1928 – 1971)

Perceptron as an Optimization Problem

- Affine function: $f(\mathbf{x}) = \langle \mathbf{x}, \mathbf{w} \rangle + b$, where $\langle \mathbf{x}, \mathbf{w} \rangle := \sum_j x_j w_j$

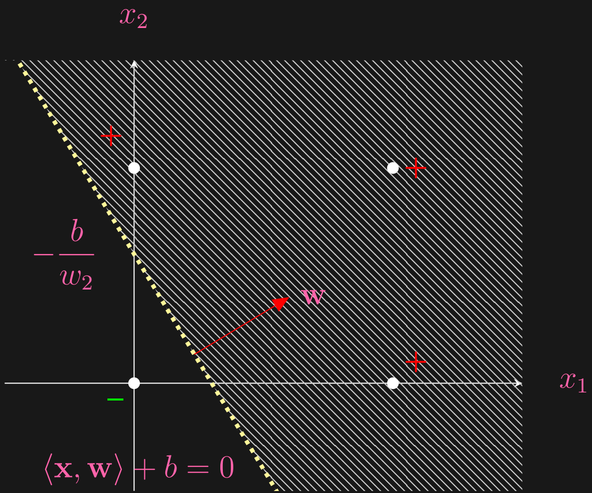
$$\text{find } \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R} \text{ such that } \forall i, \ \mathsf{y}_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + b) > 0.$$

- Perceptron solves the above optimization problem!
  - it is iterative: going through the data one by one
  - it converges faster if the problem is easier
  - it behaves benignly even if no solution exists
- Abstract a bit more:

$$\text{find } \mathbf{w} \in \mathcal{S} \subseteq \mathbb{R}^d.$$

  - we often can only describe $\mathcal{S}$ partially

Geometrically



Algorithm 1: Perceptron

Input: Dataset $\mathcal{D} = \lbag (\mathbf{x}_i, \mathsf{y}_i) \in \mathbb{R}^d \times \{\pm 1\} : i = 1, \ldots, n \rbag$, initialization $\mathbf{w} \in \mathbb{R}^d$ and $b \in \mathbb{R}$, threshold $\delta \geq 0$

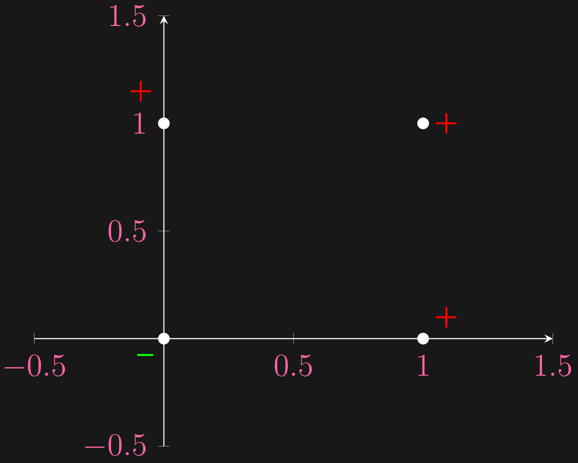
Output: approximate solution $\mathbf{w}$ and b

```
for t = 1, 2, ... do
    receive index I_t ∈ {1, ..., n}                    // I_t can be random
    if y_{I_t}(⟨x_{I_t}, w⟩ + b) ≤ δ then
        w ← w + y_{I_t} x_{I_t}                        // update after a "mistake"
        b ← b + y_{I_t}
```

- Typically $\delta = 0$ and $\mathbf{w}_0 = \mathbf{0}$, $b = 0$
  - $\mathsf{y}\hat{y} > 0$ vs. $\mathsf{y}\hat{y} < 0$ vs. $\mathsf{y}\hat{y} = 0$, where $\hat{y} = \langle \mathbf{x}, \mathbf{w} \rangle + b$
- Lazy update: "if it ain't broke, don't fix it"

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

Does it work? ☞ code

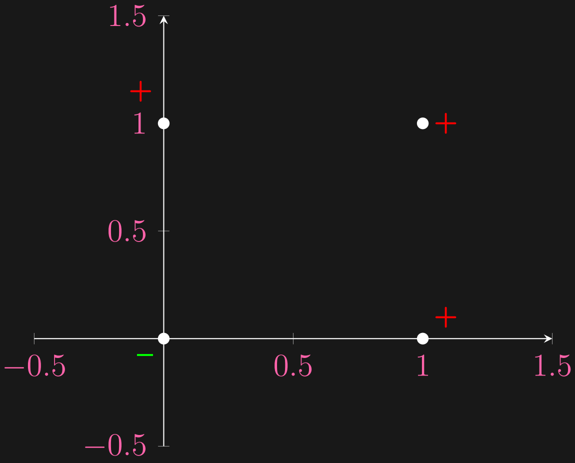


$$\hat{y} = \mathrm{sign}(\langle \mathbf{x}, \mathbf{w} \rangle + b),$$

where $\mathrm{sign}(0)$ is undefined (e.g., always counted as a mistake).

Does it work?

☞ code

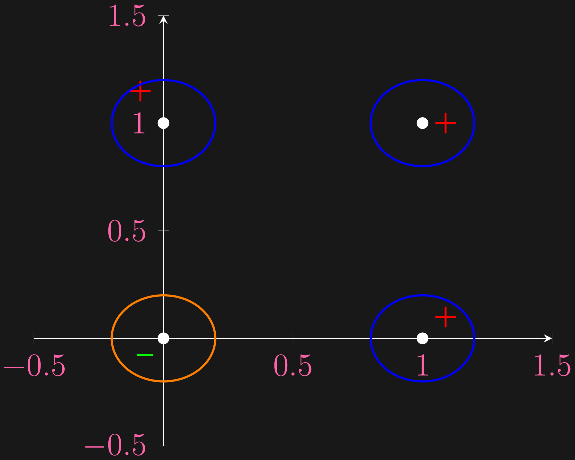


$$\mathbf{w} = [0, 0], \; b = 0, \quad \hat{y} = \text{sign}(\langle \mathbf{x}, \mathbf{w} \rangle + b),$$

where $\text{sign}(0)$ is undefined (e.g., always counted as a mistake).

Does it work?

☞ code

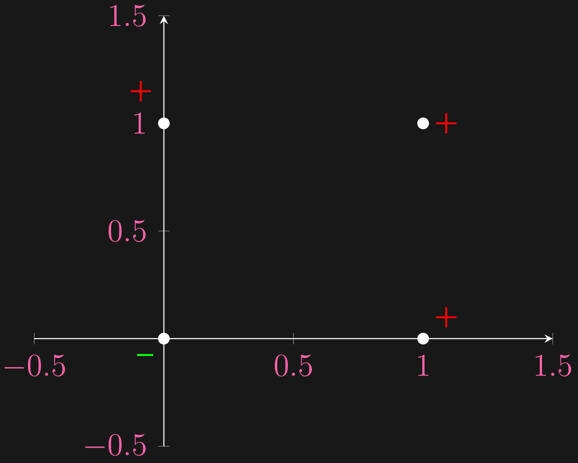


$$\mathbf{w} = [0,0],\ b = 0,\quad \hat{y} = \mathrm{sign}(\langle \mathbf{x}, \mathbf{w}\rangle + b),$$

where $\mathrm{sign}(0)$ is undefined (e.g., always counted as a mistake).

Does it work?

☞ code

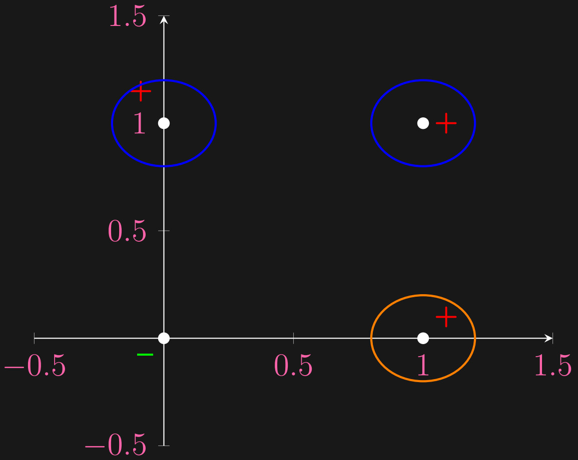


$$\mathbf{w} = [0, 0], \; b = -1, \quad \hat{y} = \text{sign}(\langle \mathbf{x}, \mathbf{w} \rangle + b),$$

where $\text{sign}(0)$ is undefined (e.g., always counted as a mistake).

Does it work?

☞ code

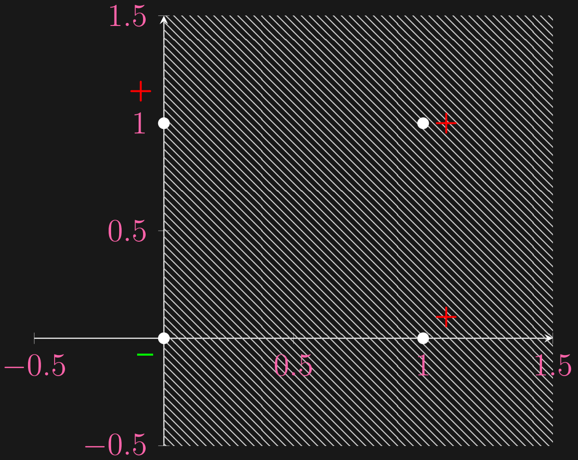


$$\mathbf{w} = [0, 0],\ \ b = -1,\ \ \ \hat{y} = \text{sign}(\langle \mathbf{x}, \mathbf{w} \rangle + b),$$

where $\text{sign}(0)$ is undefined (e.g., always counted as a mistake).

Does it work?

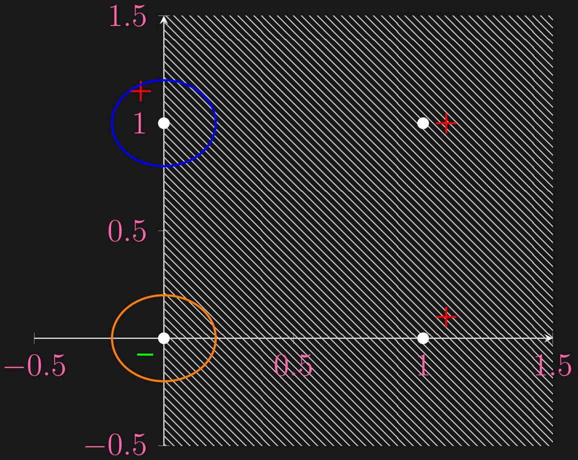
☞ code



$$\mathbf{w} = [1, 0],\ b = 0,\ \ \hat{y} = \text{sign}(\langle \mathbf{x}, \mathbf{w} \rangle + b),$$

where $\text{sign}(0)$ is undefined (e.g., always counted as a mistake).

Does it work? ☞ code

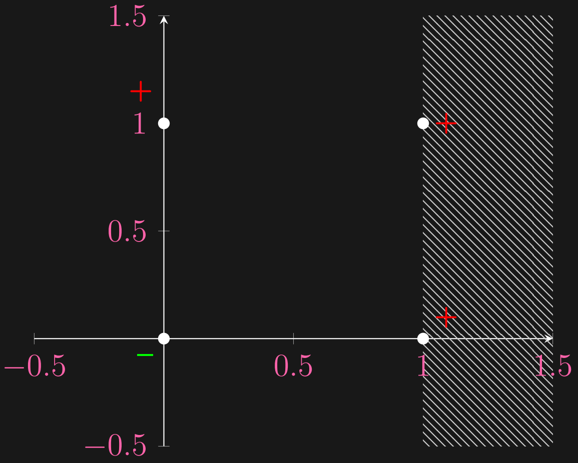


$$\mathbf{w} = [1, 0], \; b = 0, \quad \hat{y} = \text{sign}(\langle \mathbf{x}, \mathbf{w} \rangle + b),$$

where $\text{sign}(0)$ is undefined (e.g., always counted as a mistake).

Does it work?

☞ code

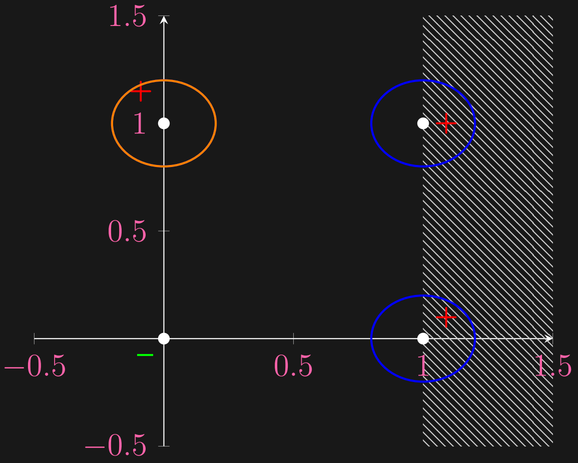


$$\mathbf{w} = [1, 0],\ b = -1,\ \hat{y} = \mathrm{sign}(\langle \mathbf{x}, \mathbf{w} \rangle + b),$$

where $\mathrm{sign}(0)$ is undefined (e.g., always counted as a mistake).

Does it work?

☞ code

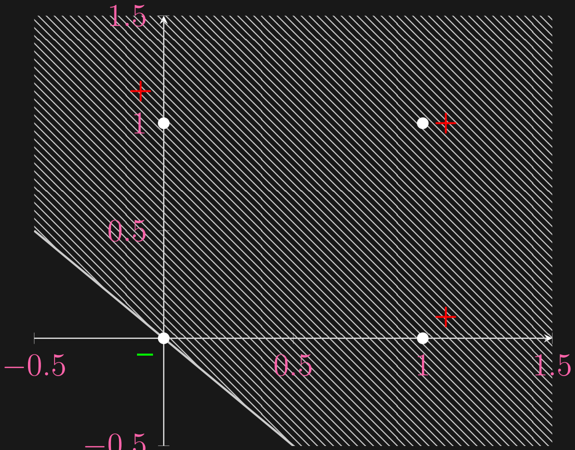


$$\mathbf{w} = [1, 0], \ b = -1, \quad \hat{\mathrm{y}} = \mathrm{sign}(\langle \mathbf{x}, \mathbf{w} \rangle + b),$$

where $\mathrm{sign}(0)$ is undefined (e.g., always counted as a mistake).

Does it work?

☞ code

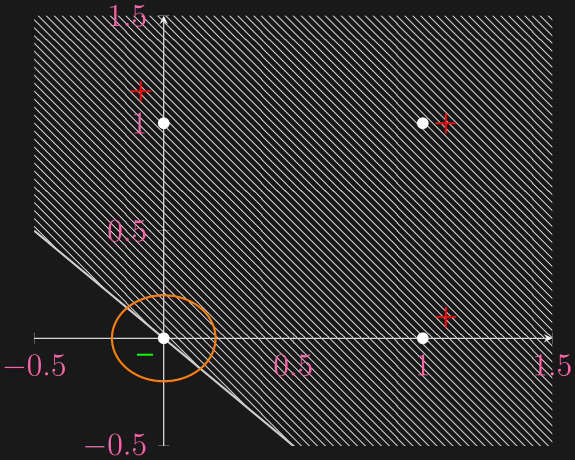


$$\mathbf{w} = [1, 1], \; b = 0, \quad \hat{y} = \mathrm{sign}(\langle \mathbf{x}, \mathbf{w} \rangle + b),$$

where $\mathrm{sign}(0)$ is undefined (e.g., always counted as a mistake).

Does it work?

☞ code

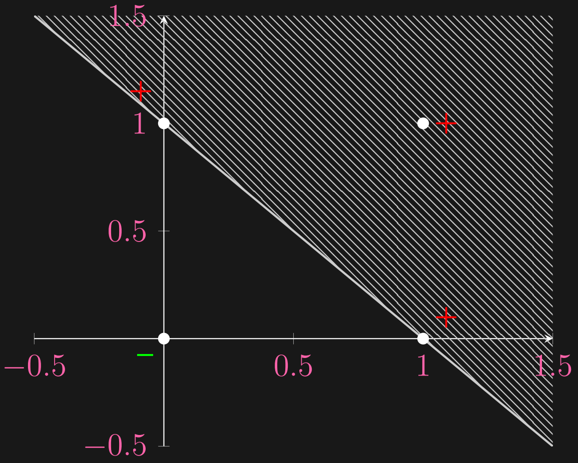


$$\mathbf{w} = [1, 1],\ b = 0,\ \ \hat{y} = \mathrm{sign}(\langle \mathbf{x}, \mathbf{w} \rangle + b),$$

where $\mathrm{sign}(0)$ is undefined (e.g., always counted as a mistake).

Does it work?

☞ code

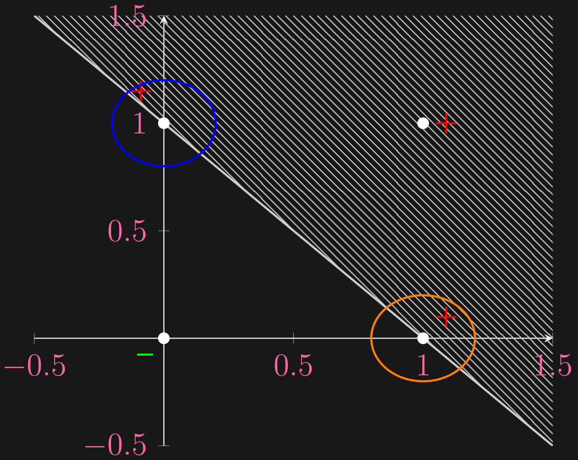


$$\mathbf{w} = [1, 1],\ b = -1,\ \ \hat{y} = \text{sign}(\langle \mathbf{x}, \mathbf{w} \rangle + b),$$

where $\text{sign}(0)$ is undefined (e.g., always counted as a mistake).

Does it work?

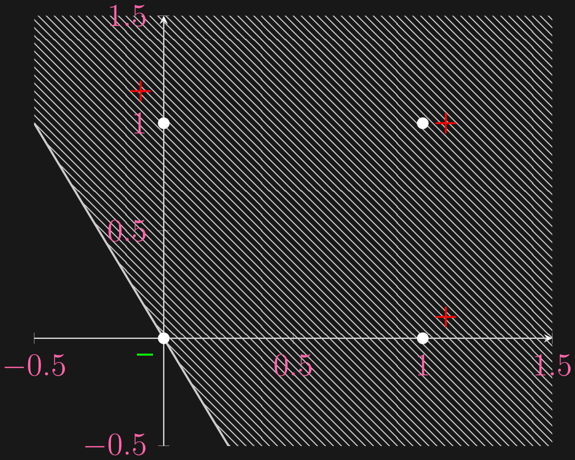
☞ code



$$\mathbf{w} = [1, 1], \; b = -1, \quad \hat{y} = \text{sign}(\langle \mathbf{x}, \mathbf{w} \rangle + b),$$

where $\text{sign}(0)$ is undefined (e.g., always counted as a mistake).

Does it work? ☞ code

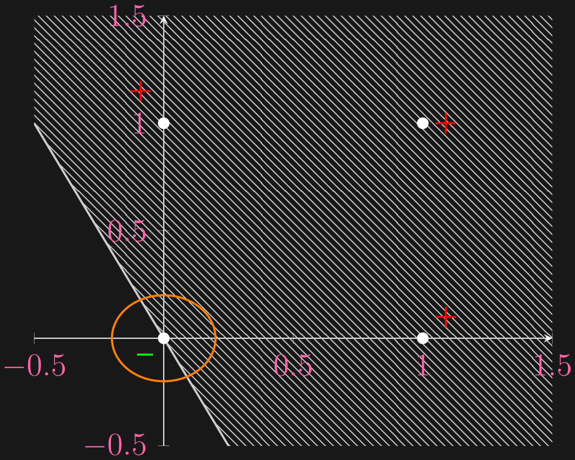


$$\mathbf{w} = [2, 1],\ \ b = 0,\quad \hat{\mathrm{y}} = \mathrm{sign}(\langle \mathbf{x}, \mathbf{w} \rangle + b),$$

where $\mathrm{sign}(0)$ is undefined (e.g., always counted as a mistake).

Does it work?

☞ code

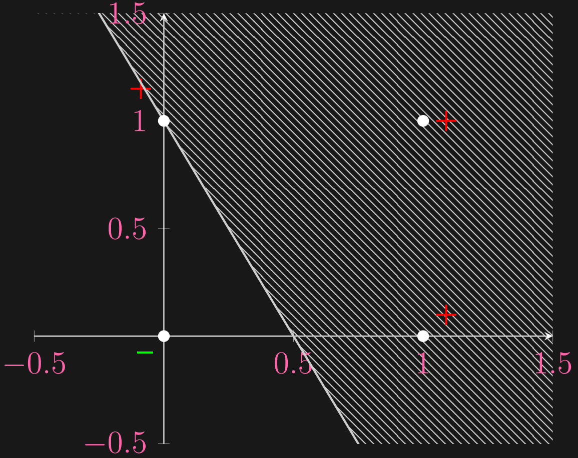


$$\mathbf{w} = [2, 1],\ b = 0,\quad \hat{y} = \mathrm{sign}(\langle \mathbf{x}, \mathbf{w} \rangle + b),$$

where $\mathrm{sign}(0)$ is undefined (e.g., always counted as a mistake).

Does it work?

☞ code

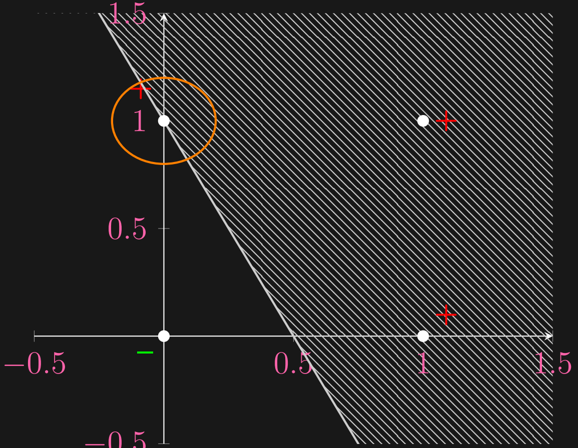


$$\mathbf{w} = [2, 1], \; b = -1, \quad \hat{\mathrm{y}} = \mathrm{sign}(\langle \mathbf{x}, \mathbf{w} \rangle + b),$$

where $\mathrm{sign}(0)$ is undefined (e.g., always counted as a mistake).

Does it work?

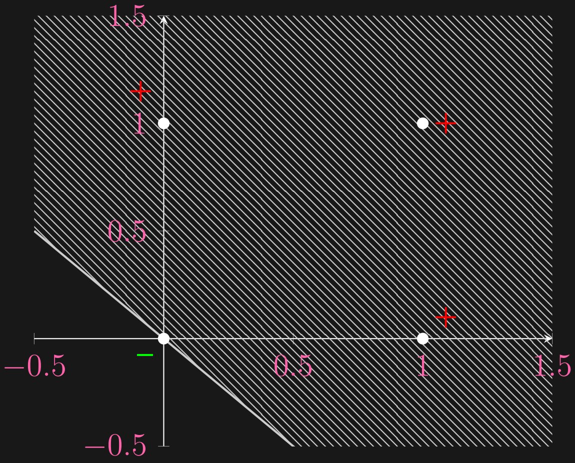
☞ code



$$\mathbf{w} = [2, 1], \; b = -1, \quad \hat{y} = \text{sign}(\langle \mathbf{x}, \mathbf{w} \rangle + b),$$

where $\text{sign}(0)$ is undefined (e.g., always counted as a mistake).

Does it work? ☞ code

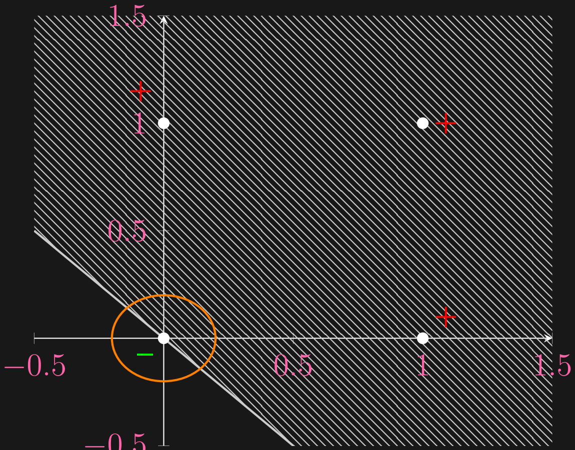


$$\mathbf{w} = [2, 2],\ \ b = 0,\ \ \ \hat{y} = \operatorname{sign}(\langle \mathbf{x}, \mathbf{w} \rangle + b),$$

where $\operatorname{sign}(0)$ is undefined (e.g., always counted as a mistake).

Does it work?

☞ code

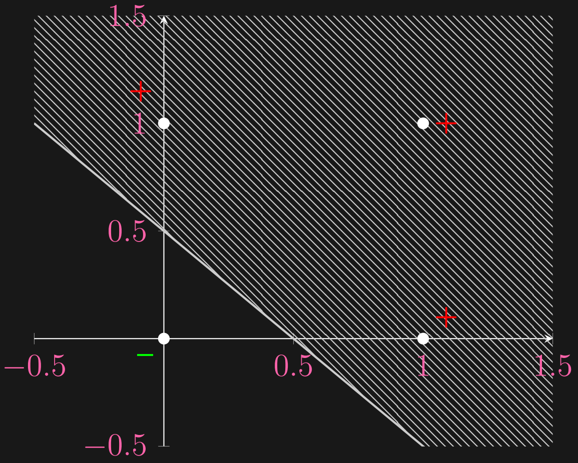


$$\mathbf{w} = [2, 2],\ b = 0,\ \ \hat{y} = \mathrm{sign}(\langle \mathbf{x}, \mathbf{w} \rangle + b),$$

where $\mathrm{sign}(0)$ is undefined (e.g., always counted as a mistake).

Does it work?

☞ code

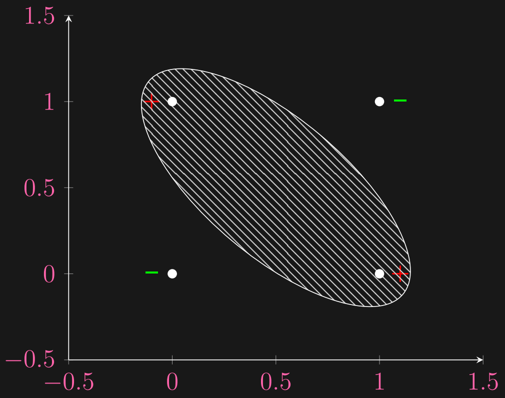


$$\mathbf{w} = [2, 2],\ \ b = -1,\ \ \ \hat{y} = \mathrm{sign}(\langle \mathbf{x}, \mathbf{w} \rangle + b),$$

where $\mathrm{sign}(0)$ is undefined (e.g., always counted as a mistake).

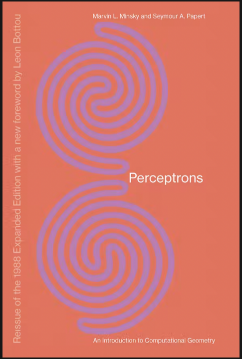
XOR Dataset

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$
	0	1	0	1
	0	0	1	1
$\mathbf{y}$	$-$	$+$	$+$	$-$

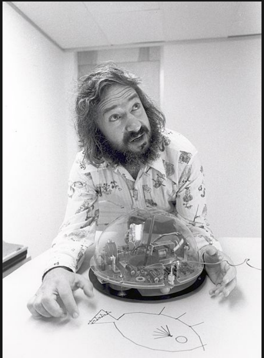


- Prove that no line can separate $+$ from $-$
- What happens if we run Perceptron regardless?

Perceptron and the 1st AI Winter



Marvin Minsky
(1927 – 2016)



Seymour Papert
(1928 – 2016)

M. L. Minsky and S. A. Papert. "Perceptron". MIT press, 1969.

Projection Algorithms

$$\text{find } \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R} \quad \text{such that} \quad \forall i,\ \mathsf{y}_i(\langle \mathbf{x}_i, \mathbf{w} \rangle + b) > 0$$

$$\text{find } \mathbf{w} = [\mathbf{w}; b] \in \mathbb{R}^{d+1} \quad \text{such that} \quad \forall i,\ \langle \mathbf{a}_i, \mathbf{w} \rangle \le c_i,\ \ \mathbf{a}_i = -\mathsf{y}_i[\mathbf{x}_i; 1]$$

$$\text{find } \mathbf{w} \in \mathbb{R}^p \quad \text{such that} \quad \mathbf{A}^\top \mathbf{w} \le \mathbf{c}$$

Algorithm 2: Projection Algorithm for Linear Inequalities

Input: $\mathbf{A} \in \mathbb{R}^{p \times n}, \mathbf{c} \in \mathbb{R}^n$, initialization $\mathbf{w} \in \mathbb{R}^p$, relaxation parameter $\eta \in (0, 2]$

```
for t = 1, 2, ... do
    select index I_t ∈ {1, ..., n}                          // index I_t can be random
    w ← (1 − η)w + η [ w − ((⟨a_{I_t}, w⟩ − c_{I_t})^+ / ||a_{I_t}||_2) · (a_{I_t} / ||a_{I_t}||_2) ]
```

$$\mathbf{w} \leftarrow (1-\eta)\mathbf{w} + \eta \left[\mathbf{w} - \frac{(\langle \mathbf{a}_{I_t}, \mathbf{w} \rangle - c_{I_t})^+}{\|\mathbf{a}_{I_t}\|_2} \cdot \frac{\mathbf{a}_{I_t}}{\|\mathbf{a}_{I_t}\|_2} \right]$$

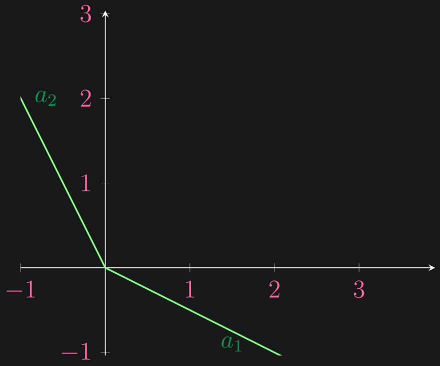
T. S. Motzkin and I. J. Schoenberg. "The Relaxation Method for Linear Inequalities". *Canadian Journal of Mathematics*, vol. 6 (1954), pp. 393–404, S. Agmon. "The Relaxation Method for Linear Inequalities". *Canadian Journal of Mathematics*, vol. 6 (1954), pp. 382–392.

Interpreting Perceptron

Theorem:

$$\operatorname{int}\operatorname{cone}^* A \neq \emptyset \iff \operatorname{int}\operatorname{cone}^* A \cap \operatorname{cone} A \neq \emptyset.$$

$$\operatorname{cone} A := \{A\boldsymbol{\lambda} : \boldsymbol{\lambda} \geq \mathbf{0}\}$$
$$\operatorname{cone}^* A := \{\mathbf{w} : A^\top \mathbf{w} \geq \mathbf{0}\}$$
$$\operatorname{int}\operatorname{cone}^* A := \{\mathbf{w} : A^\top \mathbf{w} > \mathbf{0}\}$$



Interpreting Perceptron

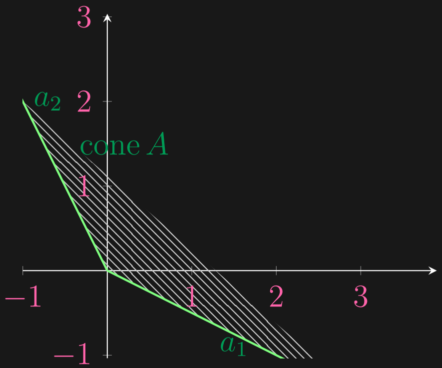
Theorem:

$$\operatorname{int}\operatorname{cone}^* A \neq \emptyset \iff \operatorname{int}\operatorname{cone}^* A \cap \operatorname{cone} A \neq \emptyset.$$

$$\operatorname{cone} A := \{A\boldsymbol{\lambda} : \boldsymbol{\lambda} \geq \mathbf{0}\}$$

$$\operatorname{cone}^* A := \{\mathbf{w} : A^\top \mathbf{w} \geq \mathbf{0}\}$$

$$\operatorname{int}\operatorname{cone}^* A := \{\mathbf{w} : A^\top \mathbf{w} > \mathbf{0}\}$$

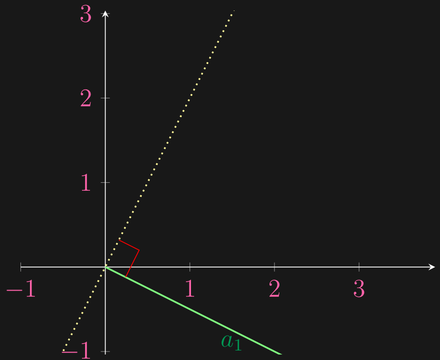


Interpreting Perceptron

Theorem:

$\operatorname{int}\operatorname{cone}^* A \neq \emptyset \iff \operatorname{int}\operatorname{cone}^* A \cap \operatorname{cone} A \neq \emptyset$.

$$\operatorname{cone} A := \{A\boldsymbol{\lambda} : \boldsymbol{\lambda} \geq \mathbf{0}\}$$
$$\operatorname{cone}^* A := \{\mathbf{w} : A^\top \mathbf{w} \geq \mathbf{0}\}$$
$$\operatorname{int}\operatorname{cone}^* A := \{\mathbf{w} : A^\top \mathbf{w} > \mathbf{0}\}$$

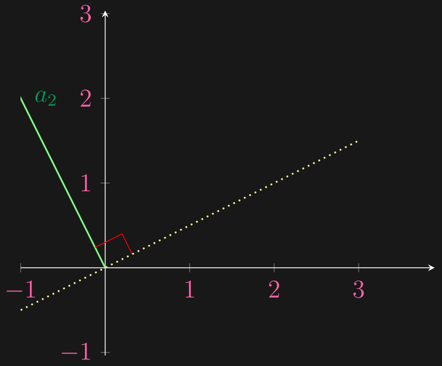


Interpreting Perceptron

Theorem:

$\operatorname{int}\operatorname{cone}^* A \neq \emptyset \iff \operatorname{int}\operatorname{cone}^* A \cap \operatorname{cone} A \neq \emptyset$.

$$\operatorname{cone} A := \{A\boldsymbol{\lambda} : \boldsymbol{\lambda} \geq \mathbf{0}\}$$
$$\operatorname{cone}^* A := \{\mathbf{w} : A^\top \mathbf{w} \geq \mathbf{0}\}$$
$$\operatorname{int}\operatorname{cone}^* A := \{\mathbf{w} : A^\top \mathbf{w} > \mathbf{0}\}$$



Interpreting Perceptron

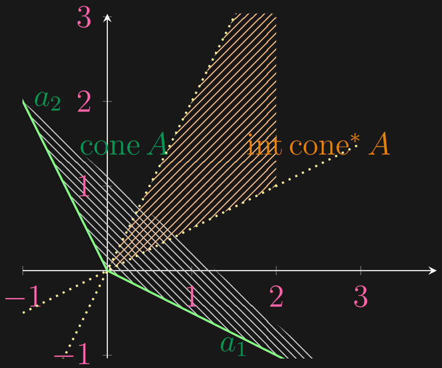
Theorem:

$$\operatorname{int}\operatorname{cone}^* A \neq \emptyset \iff \operatorname{int}\operatorname{cone}^* A \cap \operatorname{cone} A \neq \emptyset.$$

$$\operatorname{cone} A := \{A\boldsymbol{\lambda} : \boldsymbol{\lambda} \geq \mathbf{0}\}$$

$$\operatorname{cone}^* A := \{\mathbf{w} : A^\top \mathbf{w} \geq \mathbf{0}\}$$

$$\operatorname{int}\operatorname{cone}^* A := \{\mathbf{w} : A^\top \mathbf{w} > \mathbf{0}\}$$

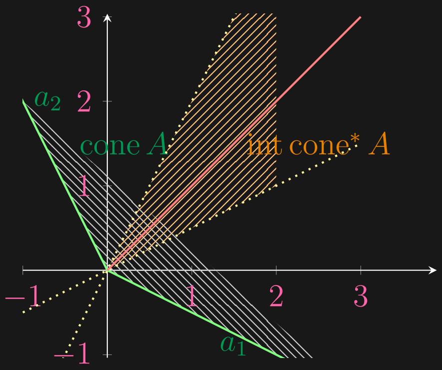


Interpreting Perceptron

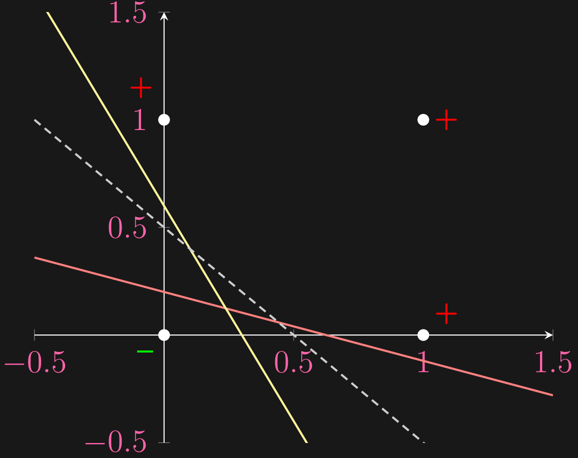
Theorem:

$\operatorname{int}\operatorname{cone}^* A \neq \emptyset \iff \operatorname{int}\operatorname{cone}^* A \cap \operatorname{cone} A \neq \emptyset.$

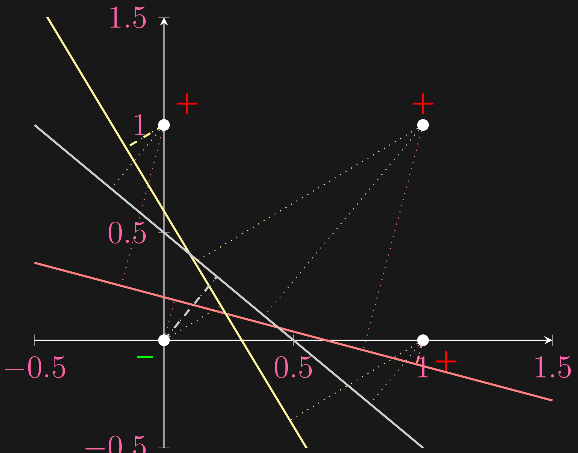
$$\operatorname{cone} A := \{A\boldsymbol{\lambda} : \boldsymbol{\lambda} \geq \mathbf{0}\}$$
$$\operatorname{cone}^* A := \{\mathbf{w} : A^\top \mathbf{w} \geq \mathbf{0}\}$$
$$\operatorname{int}\operatorname{cone}^* A := \{\mathbf{w} : A^\top \mathbf{w} > \mathbf{0}\}$$



Is Perceptron Unique?

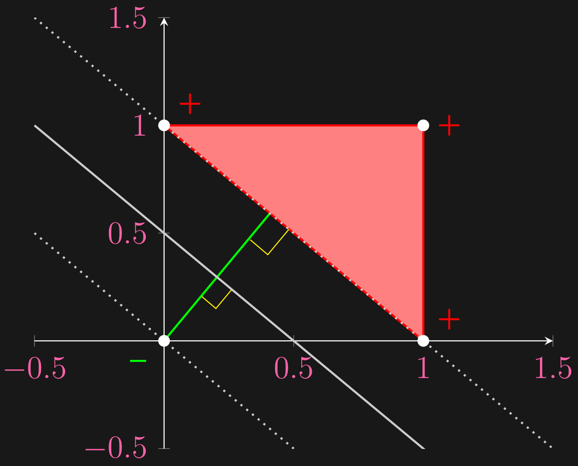


Support Vector Machines: Primal



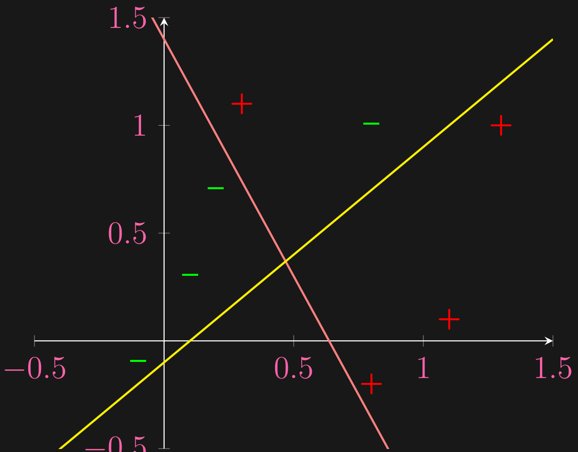
$$\max_{\mathbf{w}:\forall i, \hat{y}_i \mathsf{y}_i > 0} \min_{i=1,\ldots,n} \frac{\hat{y}_i \mathsf{y}_i}{\|\mathbf{w}\|}, \quad \text{where} \quad \hat{y}_i := \langle \mathbf{x}_i, \mathbf{w} \rangle + b$$

Support Vector Machines: Dual



$$\min_{\boldsymbol{\mu} \in \Delta_+} \min_{\boldsymbol{\nu} \in \Delta_-} \Big\| \sum_{i:\mathsf{y}_i = +} \mu_i \mathbf{x}_i - \sum_{j:\mathsf{y}_j = -} \nu_j \mathbf{x}_j \Big\|$$

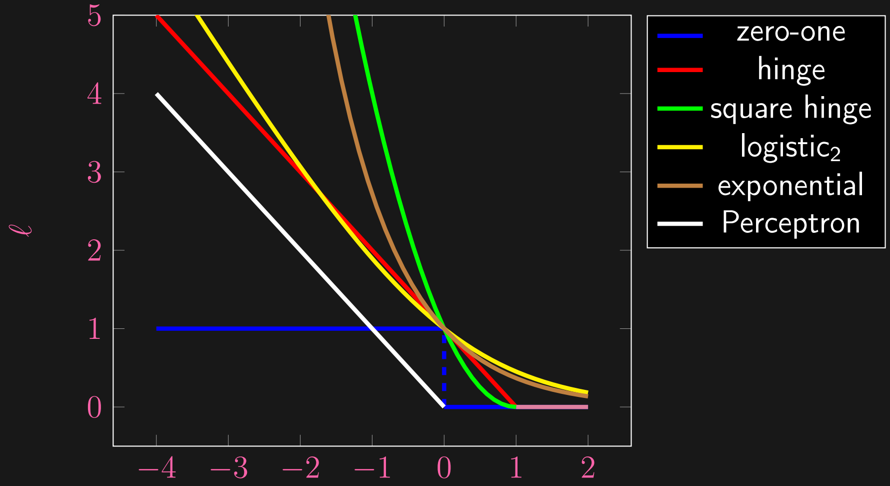
Beyond Separability



$$\min_{\mathbf{w}} \ \hat{\mathbb{E}}\ell(\mathsf{y}\hat{y}) + \mathrm{reg}(\mathbf{w}), \quad \text{s.t.} \quad \hat{y} := \langle \mathbf{x}, \mathbf{w} \rangle + b$$

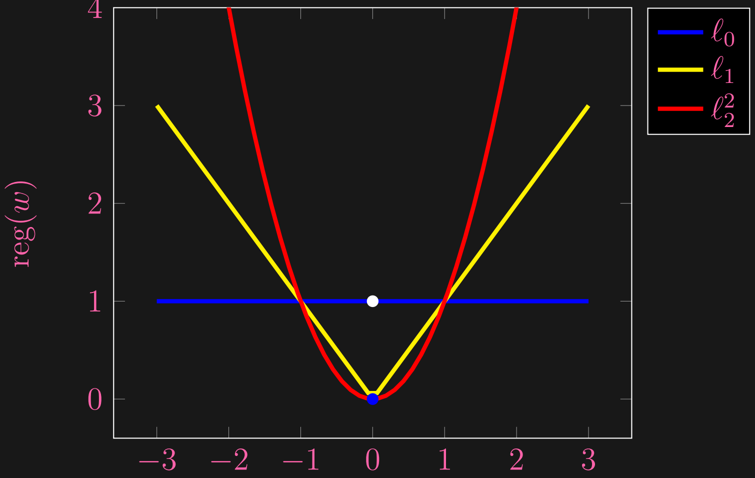
Empirical Risk Minimization

$$\min_{\mathbf{w}} \ \hat{\mathbb{E}}\ell(\mathsf{y}\hat{y}) \qquad \text{s.t.} \quad \hat{y} := \langle \mathbf{x}, \mathbf{w} \rangle + b$$



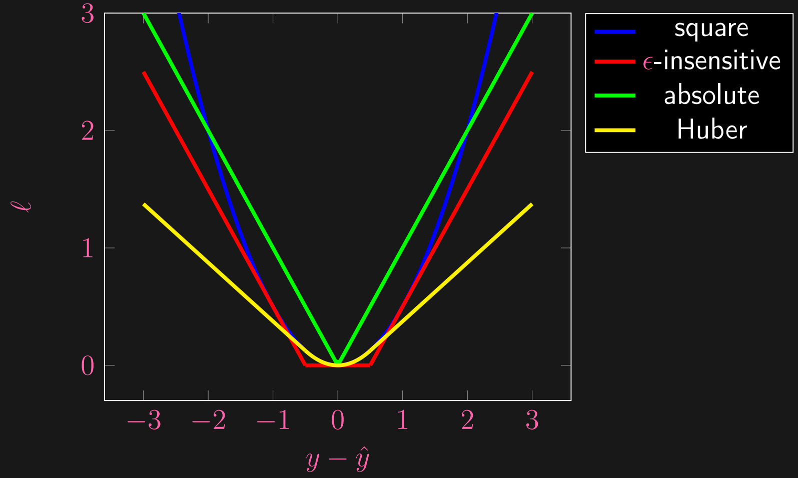
Regularization

$$\min_{\mathbf{w}} \quad \text{reg}(\mathbf{w}), \quad \text{s.t.} \quad \hat{y} := \langle \mathbf{x}, \mathbf{w} \rangle + b$$



Regression

$$\min_{\mathbf{w}} \ \hat{\mathbb{E}}\ell(y-\hat{y}) + \mathrm{reg}(\mathbf{w}), \quad \text{s.t.} \quad \hat{y} := \langle \mathbf{x}, \mathbf{w} \rangle + b$$



Day I: Basic

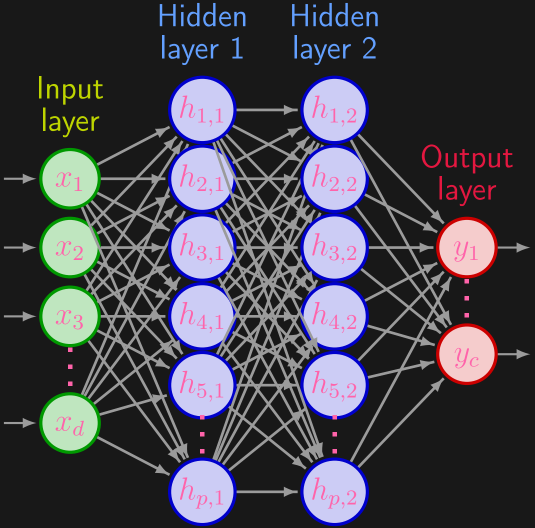
- Lec01: Gradient Descent: smooth ℓ
- Lec02: Proximal Gradient: smooth ℓ + nonsmooth reg
- Lec03: Conditional Gradient: smooth ℓ + nonsmooth reg

Denoising

$$\min_{\mathbf{z}} \underbrace{\frac{1}{2}\|\mathbf{x}-\mathbf{z}\|_2^2}_{\text{fidelity}} + \underbrace{\lambda \cdot \|\mathbf{z}\|_{\text{tv}}}_{\text{regularization}}$$

- λ controls the trade-off
- regularization encodes prior knowledge
- crucial to not over-smooth

Adversarial Examples



Shetland 😀

Collie 😟

Robustness as Optimization

- Empirical risk minimization recalled:

$$\min_{\mathbf{w}} \hat{\mathbb{E}}\ell(\mathbf{w}; \mathbf{x}, y)$$

- Adversarial attack perturbs $(\mathbf{x}, y)$ while fixing $\mathbf{w}$:

$$\max_{\text{size}(\boldsymbol{\delta})\leq\epsilon} \ell(\mathbf{w}; \mathbf{x}+\boldsymbol{\delta}, y)$$

- Robustness by anticipating the worst-case:

$$\min_{\mathbf{w}} \hat{\mathbb{E}} \max_{\text{size}(\boldsymbol{\delta})\leq\epsilon} \ell(\mathbf{w}; \mathbf{x}+\boldsymbol{\delta}, y)$$

- The game continues by anticipating the anticipation:

$$\max_{\text{size}(\boldsymbol{\delta})\leq\epsilon} \ell(\mathbf{w}; \mathbf{x}+\boldsymbol{\delta}, y) \qquad \text{leader}$$

$$\min_{\mathbf{w}} \hat{\mathbb{E}}\ell(\mathbf{w}; \mathbf{x}+\boldsymbol{\delta}, y) \qquad \text{follower}$$

Day II: Slightly Advanced

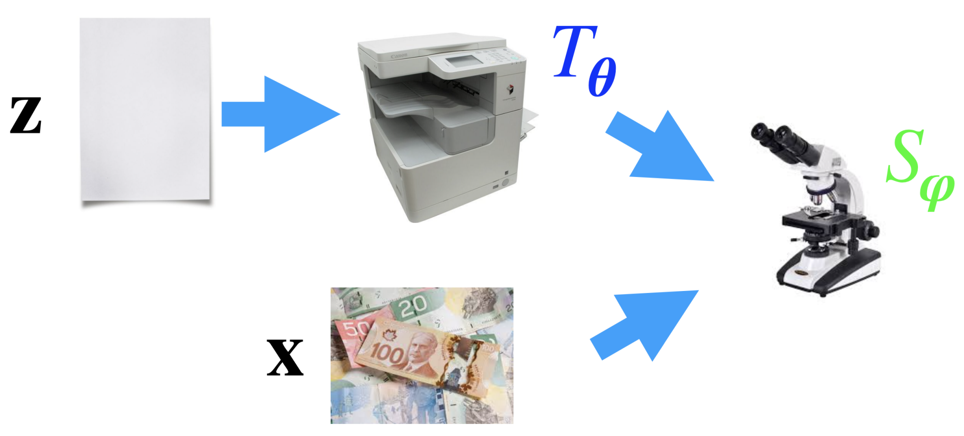
- Lec04: Subgradient: nonsmooth ℓ + nonsmooth reg
- Lec05: Acceleration: optimal algorithm under smoothness
- Lec06: Mirror Descent: smooth ℓ + nonsmooth reg
- Lec07: Metric Gradient: smooth ℓ + different norm

Day III: Game-theoretic

- Lec08: Minimax: understanding duality
- Lec09: Alternating: divide and conquer
- Lec10: Projection algorithms
- Lec11: Splitting: exploiting structure
- Fictitious Play: playing against oneself

Generative Adversarial Networks

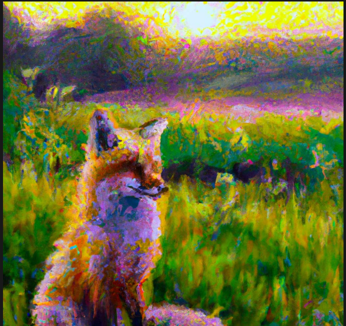
$$\min_{\theta} \max_{\varphi} \hat{\mathbb{E}} \log S_\varphi(\mathbf{x}) + \hat{\mathbb{E}} \log(1 - S_\varphi \circ T_\theta(\mathbf{z}))$$



I. Goodfellow et al. "Generative Adversarial Nets". In: *Advances in Neural Information Processing Systems*. 2014.


Parti


TheJasper
Whisperer.
com



Day IV: Stochastic

- Lec12: Stochastic Gradient: large dataset
- Lec13: Variance Reduction
- Lec14: Randomized Smoothing: simulating gradient
- Lec15: Sampling

Day V: Advanced

- Lec16: Newton: even faster under smoothness
- Lec17: Riemannian Gradient
- Lec18: Adaptation
- Lec19: Performance Estimation

History Goes A Long Way Back

"Nothing in the world takes place without optimization, and there is no doubt that all aspects of the world that have a rational basis can be explained by optimization methods."

— *Leonhard Euler, 1744*

"Every year I meet Ph.D. students of different specializations who ask me for advice on reasonable numerical schemes for their optimization models. And very often they seem to have come too late. In my experience, if an optimization model is created without taking into account the abilities of numerical schemes, the chances that it will be possible to find an acceptable numerical solution are close to zero. In any field of human activity, if we create something, we know in advance why we are doing so and what we are going to do with the result."

— *Yurii Nesterov*

No Free Lunch

- On average, no algorithm is better than any other[1]
- In general, optimization problems are unsolvable[2]
- Implications:
  - don't try to solve all problems; one (class) at a time!
  - "efficient optimization methods can be developed only by intelligently employing the structure of particular instances of problems"
  - know your algorithms and their limits
  - be open to the impossible

"There are no inferior algorithms, only inferior engineers."

[1] D. H. Wolpert and W. G. Macready. "No free lunch theorems for optimization". *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1 (1997), pp. 67–82.

[2] K. G. Murty and S. N. Kabadi. "Some NP-complete problems in quadratic and nonlinear programming". *Mathematical Programming*, vol. 39, no. 2 (1987), pp. 117–129.