CS485/685
Lecture 7: Jan 24, 2012

Perceptrons, Neural Networks

[B]: Sections 4.1.7, 5.1

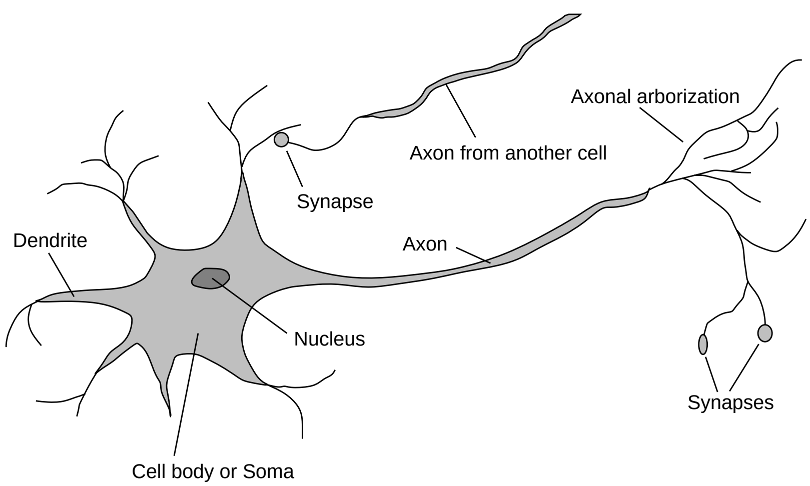
Outline

- Neural networks
  - Perceptron
  - Supervised learning algorithms for neural networks

Brain

- Seat of human intelligence
- Where memory/knowledge resides
- Responsible for thoughts and decisions
- Can learn
- Consists of nerve cells called **neurons**

Neuron



Comparison

- Brain
  - Network of neurons
  - Nerve signals propagate in a neural network
  - Parallel computation
  - **Robust (neurons die everyday without any impact)**
- Computer
  - Bunch of gates
  - Electrical signals directed by gates
  - Sequential and parallel computation
  - **Fragile (if a gate stops working, computer crashes)**

Artificial Neural Networks

- Idea: **mimic the brain to do computation**
- Artificial neural network:
  - Nodes (a.k.a units) correspond to neurons
  - Links correspond to synapses
- Computation:
  - Numerical signal transmitted between nodes corresponds to chemical signals between neurons
  - Nodes modifying numerical signal corresponds to neurons firing rate

ANN Unit

- For each unit i:
- **Weights: $\boldsymbol{W}$**
  - Strength of the link from unit j to unit i
  - Input signals x_j weighted by W_{ji} and linearly combined: $a_i = \sum_j W_{ji} x_j + w_0 = \boldsymbol{W_i^T \bar{x}}$
- **Activation function: $\boldsymbol{h}$**
  - Numerical signal produced: $y_i = h(a_i)$

ANN Unit

- Picture

Activation Function

- Should be nonlinear
  - Otherwise network is just a linear function
- Often chosen to mimic firing in neurons
  - Unit should be "active" (output near 1) when fed with the "right" inputs
  - Unit should be "inactive" (output near 0) when fed with the "wrong" inputs

Common Activation Functions

Threshold

Sigmoid

Logic Gates

- McCulloch and Pitts (1943)
  - Design ANNs to represent Boolean functions
- What should be the weights of the following units to code AND, OR, NOT ?

AND OR NOT

Network Structures

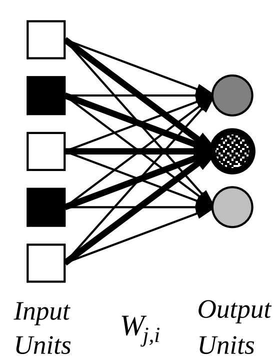
- **Feed-forward network**
  - Directed **acyclic** graph
  - No internal state
  - Simply computes outputs from inputs
- **Recurrent network**
  - Directed **cyclic** graph
  - Dynamical system with internal states
  - Can memorize information

Feed-forward network

- Simple network with two inputs, one hidden layer of two units, one output unit

Perceptron

- Single layer feed-forward network



Supervised Learning

- Given list of $(\boldsymbol{x}, \boldsymbol{y})$ pairs
- Train feed-forward ANN
  - To compute proper outputs $\boldsymbol{y}$ when fed with inputs $\boldsymbol{x}$
  - Consists of adjusting weights W_{ji}
- **Simple learning algorithm for threshold perceptrons**

Threshold Perceptron Learning

- Learning is done separately for each unit i
  - Since units do not share weights
- Perceptron learning for unit i:
  - For each $(\boldsymbol{x}, y)$ pair do:
    - Case 1: correct output produced
      $\forall_j \; W_{ji} \leftarrow W_{ji}$
    - Case 2: output produced is 0 instead of 1
      $\forall_j \; W_{ji} \leftarrow W_{ji} + x_j$
    - Case 3: output produced is 1 instead of 0
      $\forall_j \; W_{ji} \leftarrow W_{ji} - x_j$
  - Until correct output for all training instances

Threshold Perceptron Learning

- Dot products: $\bar{\boldsymbol{x}}^{\boldsymbol{T}}\bar{\boldsymbol{x}} \geq 0$ and $-\bar{\boldsymbol{x}}^{\boldsymbol{T}}\bar{\boldsymbol{x}} \leq 0$
- Perceptron computes
  1 when $\boldsymbol{w}^{\boldsymbol{T}}\bar{\boldsymbol{x}} = \sum_j x_j w_j + w_0 > 0$
  0 when $\boldsymbol{w}^{\boldsymbol{T}}\bar{\boldsymbol{x}} = \sum_j x_j w_j + w_0 < 0$
- If output should be 1 instead of 0 then
  $\boldsymbol{w} \leftarrow \boldsymbol{w} + \bar{\boldsymbol{x}}$ since $(\boldsymbol{w} + \bar{\boldsymbol{x}})^T\bar{\boldsymbol{x}} \geq \boldsymbol{w}^{\boldsymbol{T}}\bar{\boldsymbol{x}}$
- If output should be 0 instead of 1 then
  $\boldsymbol{w} \leftarrow \boldsymbol{w} - \bar{\boldsymbol{x}}$ since $(\boldsymbol{w} - \bar{\boldsymbol{x}})^T\bar{\boldsymbol{x}} \leq \boldsymbol{w}^{\boldsymbol{T}}\bar{\boldsymbol{x}}$

Alternative Approach

- Let $y \in \{-1,1\}\ \forall y$
- Let $M = \{\boldsymbol{x}_n, y_n\}$ be the set of misclassified examples
  - i.e., $y_n \boldsymbol{w}^{\boldsymbol{T}}\bar{\boldsymbol{x}}_{\boldsymbol{n}} < 0$
- Find $\boldsymbol{w}$ that minimizes misclassification

$$E(\boldsymbol{w}) = -\sum_{(x_n, y_n) \in M} y_n \boldsymbol{w}^{\boldsymbol{T}}\bar{\boldsymbol{x}}_{\boldsymbol{n}}$$

- Algorithm: gradient descent

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \boldsymbol{\nabla E}$$

learning rate
or step length

Sequential Gradient Descent

- Gradient: $\boldsymbol{\nabla E} = -\sum_{(x_n,y_n)\in M} y_n \overline{\boldsymbol{x}}_{\boldsymbol{n}}$
- Sequential gradient descent:
  - Adjust $\boldsymbol{w}$ based on one example $(\boldsymbol{x}, y)$ at a time

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta y \overline{\boldsymbol{x}}$$

- When $\eta = 1$, we recover the threshold perceptron learning algorithm

Threshold Perceptron Hypothesis Space

- Hypothesis space $h_{\boldsymbol{w}}$:
  - All binary classifications with parameters $\boldsymbol{w}$ s.t.

$$\boldsymbol{w}^{\boldsymbol{T}}\overline{\boldsymbol{x}} > 0 \rightarrow +1$$
$$\boldsymbol{w}^{\boldsymbol{T}}\overline{\boldsymbol{x}} < 0 \rightarrow -1$$

- Since $\boldsymbol{w}^{\boldsymbol{T}}\overline{\boldsymbol{x}}$ is linear in $\boldsymbol{w}$, perceptron is called a **linear separator**
- **Theorem:** Threshold perceptron learning converges iff the data is linearly separable

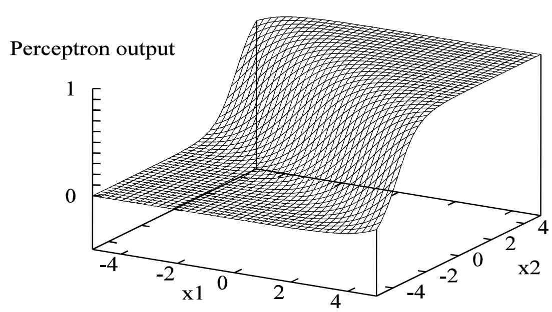
Linear Separability

- Examples:

Linearly separable Non-linearly separable

Sigmoid Perceptron

- Represent “soft” linear separators
- **Same hypothesis space as logistic regression**



Sigmoid Perceptron Learning

- Possible objectives
  - **Minimum squared error**

    $$E(\boldsymbol{w}) = \frac{1}{2}\sum_n E_n(\boldsymbol{w})^2 = \frac{1}{2}\sum_n \left(y_n - \sigma\left(\boldsymbol{w}^{\boldsymbol{T}}\bar{\boldsymbol{x}}_{\boldsymbol{n}}\right)\right)^2$$

  - **Maximum likelihood**
    - Same algorithm as for logistic regression
  - Maximum a posteriori hypothesis
  - Bayesian Learning

Gradient

- Gradient:

  $$\frac{\partial E}{\partial w_j} = \sum_n E_n(w)\,\frac{\partial E_n}{\partial w_j}$$

  $$= \sum_n E_n(w)\sigma'(w^T\bar{x}_n)x_j$$

  Recall that $\sigma' = \sigma(1-\sigma)$

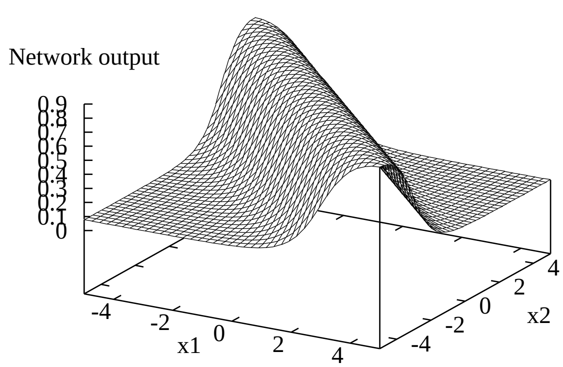
  $$= \sum_n E_n(w)\sigma(w^T\bar{x}_n)\left(1-\sigma(w^T\bar{x}_n)\right)x_j$$

Sequential Gradient Descent

- Perceptron-Learning(examples,network)
  - Repeat
    - For each $(\boldsymbol{x_n}, y_n)$ in examples do
      $E_n \leftarrow y_n - \sigma(\boldsymbol{w^T}\boldsymbol{\bar{x}_n})$
      $\boldsymbol{w} \leftarrow \boldsymbol{w} + \eta\, E_n\, \sigma(\boldsymbol{w^T}\boldsymbol{\bar{x}_n})\left(1 - \sigma(\boldsymbol{w^T}\boldsymbol{\bar{x}_n})\right)\, \boldsymbol{\bar{x}_n}$
  - Until some stopping criteria satisfied
  - Return learnt network
- N.B. η is a learning rate corresponding to the step size in gradient descent

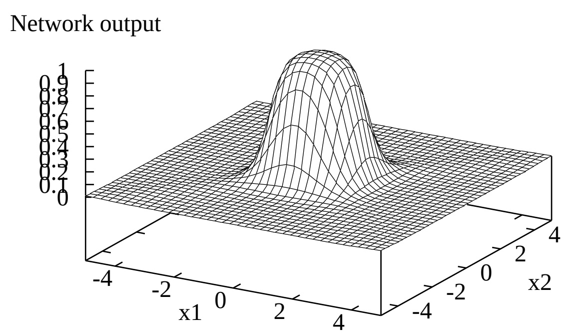
Multilayer Networks

- Adding two sigmoid units with parallel but opposite "cliffs" produces a ridge



Multilayer Networks

- Adding two intersecting ridges (and thresholding) produces a bump



Multilayer Networks

- By tiling bumps of various heights together, we can approximate any function

- Training algorithm:
  - **Back-propagation**
  - Essentially sequential gradient descent performed by propagating errors backward into the network
  - Derivation next class

Neural Net Applications

- Neural nets can approximate any function, hence millions of applications
  - NETtalk for pronouncing English text
  - Character recognition
  - Paint-quality inspection
  - Vision-based autonomous driving
  - Etc.