Perceptrons, Neural Networks

[D] Chapt. 4, [HTF] Chapt. 11, [B] Sec. 4.1.7, 5.1, [M] Sec. 8.5.4, [RN] Sec. 18.7
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

- Axonal arborization
- Axon from another cell
- Synapse
- Dendrite
- Nucleus
- Cell body or Soma
- Synapses
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: \( W \)**
  – Strength of the link from unit \( i \) to unit \( j \)
  – Input signals \( x_i \) weighted by \( W_{ji} \) and linearly combined:
    \[
    a_j = \sum_i W_{ji} x_i + w_0 = W_j \bar{x}
    \]

• **Activation function: \( h \)**
  – Numerical signal produced: \( y_j = h(a_j) \)
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?
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 \((x, y)\) pairs
- Train feed-forward ANN
  - To compute proper outputs \(y\) when fed with inputs \(x\)
  - Consists of adjusting weights \(W_{ji}\)
- Simple learning algorithm for threshold perceptrons
Threshold Perceptron Learning

• Learning is done separately for each unit $j$
  – Since units do not share weights

• Perceptron learning for unit $j$:
  – For each $(x, y)$ pair do:
    • Case 1: correct output produced
      $\forall i W_{ji} \leftarrow W_{ji}$
    • Case 2: output produced is 0 instead of 1
      $\forall i W_{ji} \leftarrow W_{ji} + x_i$
    • Case 3: output produced is 1 instead of 0
      $\forall i W_{ji} \leftarrow W_{ji} - x_i$
  – Until correct output for all training instances
Threshold Perceptron Learning

• Dot products: $x^T x \geq 0$ and $-x^T x \leq 0$

• Perceptron computes
  
  1 when $w^T x = \sum_i x_i w_i + w_0 > 0$
  0 when $w^T x = \sum_i x_i w_i + w_0 < 0$

• If output should be 1 instead of 0 then
  
  $w \leftarrow w + x$ since $(w + x)^T x \geq w^T x$

• If output should be 0 instead of 1 then
  
  $w \leftarrow w - x$ since $(w - x)^T x \leq w^T x$
Alternative Approach

• Let \( y \in \{-1,1\} \quad \forall y \)
• Let \( M = \{(x_n, y_n) \}_{\forall n} \) be set of misclassified examples
  – i.e., \( y_n w^T \bar{x}_n < 0 \)

• Find \( w \) that minimizes misclassification

\[
E(w) = - \sum_{(x_n,y_n) \in M} y_n w^T \bar{x}_n
\]

• Algorithm: gradient descent

\[
w \leftarrow w - \eta \nabla E
\]

learning rate or step length
Sequential Gradient Descent

- Gradient: $\nabla E = - \sum_{(x_n, y_n) \in M} y_n x_n$

- Sequential gradient descent:
  - Adjust $w$ based on one example $(x, y)$ at a time
    $$ w \leftarrow w + \eta y x $$

- When $\eta = 1$, we recover the threshold perceptron learning algorithm
Threshold Perceptron
Hypothesis Space

• Hypothesis space $h_w$:
  
  – All binary classifications with parameters $w$ s.t.
    
    $w^T \bar{x} > 0 \rightarrow +1$
    
    $w^T \bar{x} < 0 \rightarrow -1$

• Since $w^T \bar{x}$ is linear in $w$, 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
    \[ E(w) = \frac{1}{2} \sum_n E_n(w)^2 = \frac{1}{2} \sum_n (y_n - \sigma(w^T \bar{x}_n))^2 \]
  – Maximum likelihood
    • Same algorithm as for logistic regression
  – Maximum a posteriori hypothesis
  – Bayesian Learning
Gradient

- Gradient:

\[
\frac{\partial E}{\partial w_i} = \sum_n E_n(w) \frac{\partial E_n}{\partial w_i} = - \sum_n E_n(w) \sigma'(w^T \bar{x}_n)x_i
\]

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

\[
= - \sum_n E_n(w) \sigma(w^T \bar{x}_n)(1 - \sigma(w^T \bar{x}_n))x_i
\]
Sequential Gradient Descent

• Perceptron-Learning(examples, network)
  – Repeat
    • For each \((x_n, y_n)\) in examples do
      \[ E_n \leftarrow y_n - \sigma(w^T \bar{x}_n) \]
      \[ w \leftarrow w + \eta E_n \sigma(w^T \bar{x}_n) \left( 1 - \sigma(w^T \bar{x}_n) \right) \bar{x}_n \]
  – Until some stopping criterion satisfied
  – Return learnt network

• N.B. \(\eta\) 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

![Network output graph]
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
  – Speech recognition
  – Vision-based object recognition
  – Word embeddings
  – Vision-based autonomous driving
  – Etc.