CS 4824/ECE 4424: Perceptron

Acknowledgement:
Many of these slides are derived from Tom Mitchell, Pascal Poupart, Pieter Abbeel, Eric Eaton, Carlos Guestrin, William Cohen, and Andrew Moore.
Human Intelligence

- Brian is responsible for human intelligence by performing
  - Learning
  - Memorization
  - Cognition and recognition
  - Decision making

- Brian consists of nerve cells called neurons
  - Neurons can propagate nervous signal
  - Neurons form giant network of signal propagation
Comparison

- **Brain**
  - Network of neurons
  - Nerve signals propagate via 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

- **Key idea:** emulate biological neurons for computation

- Artificial neural network (ANN)
  - Units are called "nodes" and correspond to neurons
  - Connections between nodes correspond to synapses

- Correspondence between ANN and biological neural network
  - Numerical signal transmitted between nodes corresponds to chemical signals between neurons
  - Nodes modifying numerical signal correspond to neurons firing rate
ANN: Node

- Schematic

\[ a_j = \sum_{i} w_{ji} x_i + w_j \theta \]

Activation function: \( h(a_j) \)
ANN

- **Node**: $i$

- **Weights**: $W$
  - Strength of the connection from node $i$ to node $j$
  - Input signals $x_i$ weighted by $W_{ji}$ and linearly combined:
    \[
    a_j = \sum_{i} W_{ji} x_i + w_0 = \mathbf{W}_{ji} \mathbf{x}
    \]

- **Activation function**: $h$
  - Numerical signal produced: $y_j = h(a_j)$

- **Nodes are interconnected to form a network**
ANN: Activation Function

- Generally non-linear
  - Else, the network is just a linear function

- Mimics the firing in neurons
  - Nodes should be “active” (output close to 1) when fed with the “right” inputs
  - Nodes should be “inactive” (output close to 0) when fed with the “wrong” inputs
Common Activation Functions

Identity

\[ h(a) = a \]

Threshold

\[
h(a) = \begin{cases} 
1 & \text{if } a \geq 0 \\
0 & \text{if } a < 0 
\end{cases}
\]

Sigmoid

\[ h(a) = \sigma(a) = \frac{1}{1 + e^{-a}} \]
Representing Boolean Functions

- Design ANN for logic gates

- What should be the weights of the following units to represent AND, OR, NOT gates?

<table>
<thead>
<tr>
<th>AND</th>
<th>OR</th>
<th>NOT</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_1 )</td>
<td>( x_2 )</td>
<td>( x_1 )</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

\( w_0 = -2 \) \( w_1 = 1 \) \( w_2 = 1 \) \( w_0 = -0.5 \) \( w_1 = 1 \) \( w_0 = 1 \)
Representing Boolean Functions

- ANNs can be used to design various logic gates
- So ANNs can be used to approximate any boolean functions
Network Architecture

- **Feed-forward Network**
  - Directed **acyclic** graph
  - No internal state

- **Recurrent Network**
  - Directed **cyclic** graph
  - Dynamical system with internal states
  - Can memorize information
Perceptron

- Single layer feed-forward network
Threshold Perceptron

- Given list of $\{(x, y)\}$ pairs
- Train feed-forward ANN
  - Compute correct outputs $y$ when fed with inputs $x$
  - Accordingly adjust weights $W_{ji}$
- Leads to a simple algorithm for threshold perceptrons
Threshold Perceptron Learning

- Learning is done separately for each output node $j$
  - Since nodes do not share weights

- Perceptron learning for node $j$
  - For each $(x, y)$ pairs do:
    - Case 1: correct output produced
      \[
      \forall_i W_{ji} \leftarrow W_{ji}
      \]
    - Case 2: output produced 0 instead of 1
      \[
      \forall_i W_{ji} \leftarrow W_{ji} + x_i
      \]
    - Case 3: output produced 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 $x^Tx \geq 0$ and $-x^Tx \leq 0$

- Perceptron computes
  - 1 when $w^Tx = \sum_i x_i w_i + w_0 > 0$
  - 0 when $w^Tx = \sum_i x_i w_i + w_0 < 0$

- If output should be 1 instead of 0
  - $w \leftarrow w + x$ since $(w + x)^Tx \geq w^Tx$

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

- Let $y \in \{-1, 1\}$ $\forall y$
- Let $M = \{\{x_n, y_n\} \forall n\}$ be set of misclassified examples
  - i.e. $y_n w^T x < 0$
- Find $w$ that minimizes misclassification error:
  - $E(w) = - \sum_{(x_n, y_n) \in M} y_n w^T x$
- Apply gradient descent algorithm
  - $w \leftarrow w - \eta \nabla E$

*learning rate or step size*
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 algorithm
Threshold Perceptron Algorithm

- Let $y \in \{-1, 1\} \forall y$

- Start with randomly initialized weights: $w$

- For $t = 1..T$ (T passes over data)
  - For $l = 1..n$: (each training example)
    - Classify with current weights
      - $\hat{y} = \text{sign}(w^T x)$ where $\text{sign}(x) = +1$ if $x > 0$ else $-1$
    - If correct (i.e., $\hat{y} = y^l$), no change!)
  - If wrong: update:
    - $w \leftarrow w + y^l x^l$
Properties of Threshold Perceptron

- Hypothesis space $h_w$
- Binary classifications with parameters $w$
- Since $w^T x$ is linear in $w$, perceptron is a **linear separator**
- Converges *iff* the data is linearly separable
Perceptron Linear Separability

- Examples

Linearly Separable

Linearly Nonseparable
Sigmoid Perceptron

- “Soft” linear separator

Can we use sigmoid perceptron for linearly nonseparable data points?
Sigmoid Perceptron Learning

- Maximum likelihood estimation
  - Equivalent to logistic regression

- Objective function can be:
  - Minimimim squared error

\[ E(w) = \frac{1}{2} \sum_n E_n(w)^2 = \frac{1}{2} \sum_n (y_n - \sigma(w^T x_n))^2 \]
Gradient

- Derivation

\[
\frac{\partial E}{\partial w_i} = \sum_n E_n \frac{\partial E_n}{\partial w_i}
\]

\[
= \sum_n E_n(w) \sigma'(w^T x_n) x_i
\]

\[
= \sum_n E_n(w) \sigma(w^T x_n)(1 - \sigma(w^T x_n)) x_i
\]

Recall,

\[
\sigma(x) = \frac{1}{1 + e^{-x}}
\]

\[
\sigma'(x) = \frac{d}{dx} \sigma(x) = \sigma(x)(1 - \sigma(x))
\]

No closed form solution!
Gradient Descent

- Perceptron-Learning(examples, network)
  - Repeat
  - For each $(x_n, y_n)$ in examples, do:
    - $E_n \leftarrow y_n - \sigma (w^T x)$
    - $w \leftarrow w + \eta E_n \sigma (w^T x) (1 - \sigma (w^T x)) x_n$
  - Until some stopping criteria satisfied
  - Return learnt network
Demo Time 😊

https://playground.tensorflow.org/