CS 4824/ECE 4424: Neural Networks I

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

\( \circ \) Let \( y \in \{-1, 1\} \ \forall y \)

\( \circ \) Start with randomly initialized weights: \( w \)

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

- Hypothesis space $h_w$
- Binary classifications with parameters $w$
- Since $w^Tx$ is linear in $w$, perceptron is a linear separator
- Converges iff the data is linearly separable
Sigmoid Perceptron

- “Soft” linear separator

Sigmoid perceptron cannot be used for linearly non-separable data points
Multilayer Networks

- Adding two sigmoid nodes with parallel but opposite “cliffs” produces a ridge

- Schematic
Multilayer Networks

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

- Schematic
Multilayer Networks

- A bump can classify linearly non-separable data points

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

https://playground.tensorflow.org/
Multilayer Neural Networks are Expressive!

- Multilayer Neural Networks can approximate any function, hence millions of applications
  - Machine translation
  - Computer vision
  - Speech recognition
  - Word embedding
  - …
Network Architecture

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

- **Recurrent Network**
  - Directed *cyclic* graph
  - Dynamical system with internal states
  - Can memorize information
Two-layer Feed-forward Network

- Architecture

- Hidden nodes: \( z_j = h_1(\mathbf{w}_j^{(1)\top} \mathbf{x}) \)
- Output nodes: \( y_k = h_2(\mathbf{w}_k^{(2)\top} z) \)
- Overall: \( y_k = h_2 \left( \sum_j \mathbf{w}_{kj}^{(2)} h_1 \left( \sum_i \mathbf{w}_{ji}^{(1)} x_i \right) \right) \)
Two-layer Feed-forward Network

- Regression
- Classification
Common Activation Functions $h$

- **Identity** $h(a) = a$
- **Threshold** $h(a) = \begin{cases} 1 & \text{if } a \geq 0 \\ -1 & \text{if } a < 0 \end{cases}$
- **Sigmoid** $h(a) = \sigma(a) = \frac{1}{1 + e^{-a}}$
- **Gaussian** $h(a) = e^{-\frac{1}{2}(\frac{a-\mu}{\sigma})^2}$
- **Tanh** $h(a) = \tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}$
Optimizing the Weights

- Parameters: $<W^{(1)}, W^{(2)}, ...>$
- Objective:
  - Error minimization
  - Backpropagation (aka backprop)