CS 4824/ECE 4424: Neural Networks

Acknowledgement:
Many of these slides are derived from Tom Mitchell, Pascal Poupart, Pieter Abbeel, Eric Eaton, Carlos Guestrin, William Cohen, and Andrew Moore.
Two-layer Feed-forward Network

- Architecture

  - Hidden nodes: \( z_j = h_1 (\mathbf{w}_j^{(1)} \mathbf{x}) \)
  - Output nodes: \( y_k = h_2 (\mathbf{w}_k^{(2)} \mathbf{z}) \)
  - Overall: \( y_k = h_2(\sum_j \mathbf{w}_{kj}^{(2)} h_1(\sum_i \mathbf{w}_{ji}^{(1)} x_i)) \)
Common Activation Functions \( h \)

- Identity \( h(a) = a \)
- Threshold \( h(a) = \begin{cases} 1 & \text{if } a \geq 0 \\ 0 & \text{if } a < 0 \end{cases} \)
- Sigmoid \( h(a) = \sigma(a) = \frac{1}{1 + e^{-a}} \)
- Gaussian \( h(a) = e^{-\frac{1}{2}(\frac{a-\mu}{\sigma})^2} \)
- Tanh \( h(a) = tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}} \)
Two-layer Feed-forward Network

- Regression
- Classification
Combining Activation Functions

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

- Schematic
Combining Activation Functions

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

- Schematic
Combining Activation Functions

- A bump can classify linearly non-separable data points

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

- Combining activation functions in a neural network enables us to approximate any function, hence millions of applications
  - Machine translation
  - Computer vision
  - Speech recognition
  - Word embedding
  - ...
Optimizing the Weights

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

- Two phases:
  - Forward phase: compute output $z_j$ for each node $j$
  - Backward phase: compute output $\delta_j$ for each node $j$
Forward Phase

- Propagate inputs forward through the network to compute the output of each node
- Output $z_j$ at node $j$
  - $z_j = h(a_j)$ where $a_j = \sum_i w_{ji} z_i$
Backward Phase

- Use chain rule to recursively compute gradient
  - For each weight $w_{ji}$
    $\delta_{ji} = \frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}}$
  
- Let $\delta_j = \frac{\partial E_n}{\partial a_j}$
  
- Then
  $\delta_j = \begin{cases} h'(a_j)(z_j - y_j) & \text{base case: } j \in \text{output nodes} \\ h'(a_j) \sum_k w_{kj} \delta_k & \text{recursion: } j \in \text{hidden nodes} \end{cases}$

- Since
  $a_j = \sum_k w_{ji} z_i$ then $\frac{\partial a_j}{\partial w_{ji}} = z_i$

- Therefore, $\delta_{ji} = \frac{\partial E_n}{\partial w_{ji}} = \frac{\partial E_n}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}} = \delta_j z_i$
An Example

- A simple two-layer network:
  - Hidden nodes (Tanh): \( h(a) = \tanh(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}} \)
  - Note: \( \tanh'(a) = (1 - (\tanh'(a))^2 \)

- Output nodes (Identity): \( h(a) = a \)

- Objective function: squared error
The derivation

- **Forward phase**
  - Hidden nodes: \( a_j = \)
  - Output nodes: \( a_k = \)

- **Backward phase**
  - Output nodes: \( \delta_k = \)
  - Hidden nodes: \( \delta_j = \)

- **Gradients**
  - Hidden layer: \( \frac{\partial E_n}{\partial w_{ji}} = \)
  - Output layer: \( \frac{\partial E_n}{\partial w_{kj}} = \)
The derivation

- **Forward phase**
  - Hidden nodes: \( a_j = \sum_i w_{ji} x_i \) \( z_j = \tanh(a_j) \)
  - Output nodes: \( a_k = \sum_j w_{kj} z_j \) \( z_k = a_k \)

- **Backward phase**
  - Output nodes: \( \delta_k = z_k - y_k \)
  - Hidden nodes: \( \delta_j = [1 - (z_j)^2] \sum_k w_{kj} \delta_k \)

- **Gradients**
  - Hidden layer: \( \frac{\partial E_n}{\partial w_{ji}} = \delta_j x_i = [1 - (z_j)^2] \sum_k w_{kj} \delta_k x_i \)
  - Output layer: \( \frac{\partial E_n}{\partial w_{kj}} = \delta_k z_j = (z_k - y_k) z_j \)
Non-linear regression examples

- Two layer network:
  - 3 tanh hidden units and 1 identity output unit

\[
y = x^2
\]

\[
y = \sin x
\]

\[
y = |x|
\]

\[
y = \int_{-\infty}^{x} \delta(t) dt
\]
Demo Time 😊

https://playground.tensorflow.org/