ITCS 6156/8156 Fall 2023
Machine Learning

Neural Networks

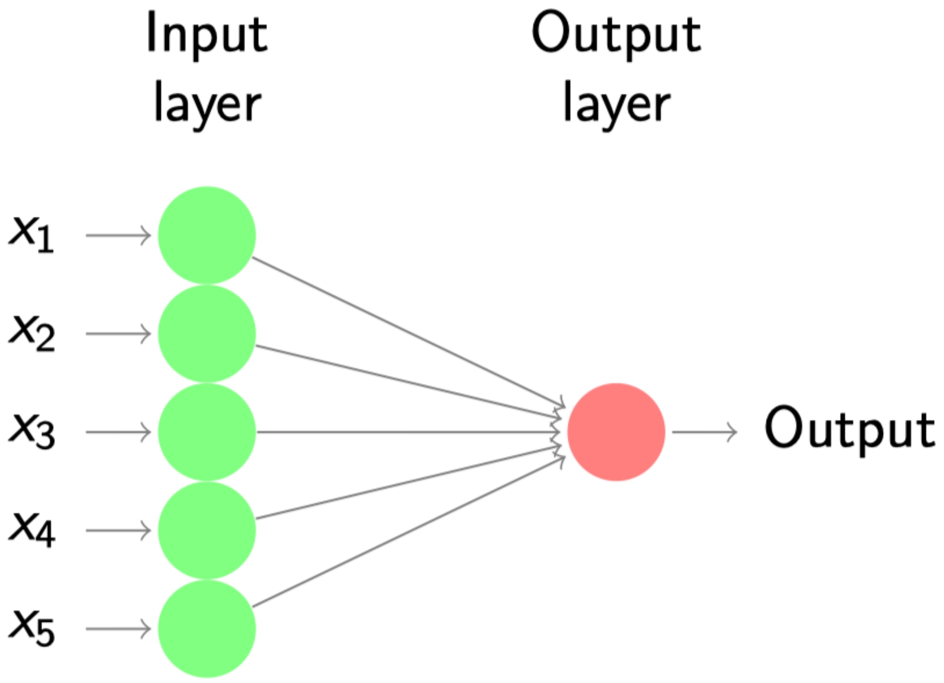
Instructor: Hongfei Xue
Email: hongfei.xue@charlotte.edu
Class Meeting: Mon & Wed, 4:00 PM – 5:15 PM, CHHS 376

UNIVERSITY OF NORTH CAROLINA CHARLOTTE

Some content in the slides is based on Dr. Varun's lecture

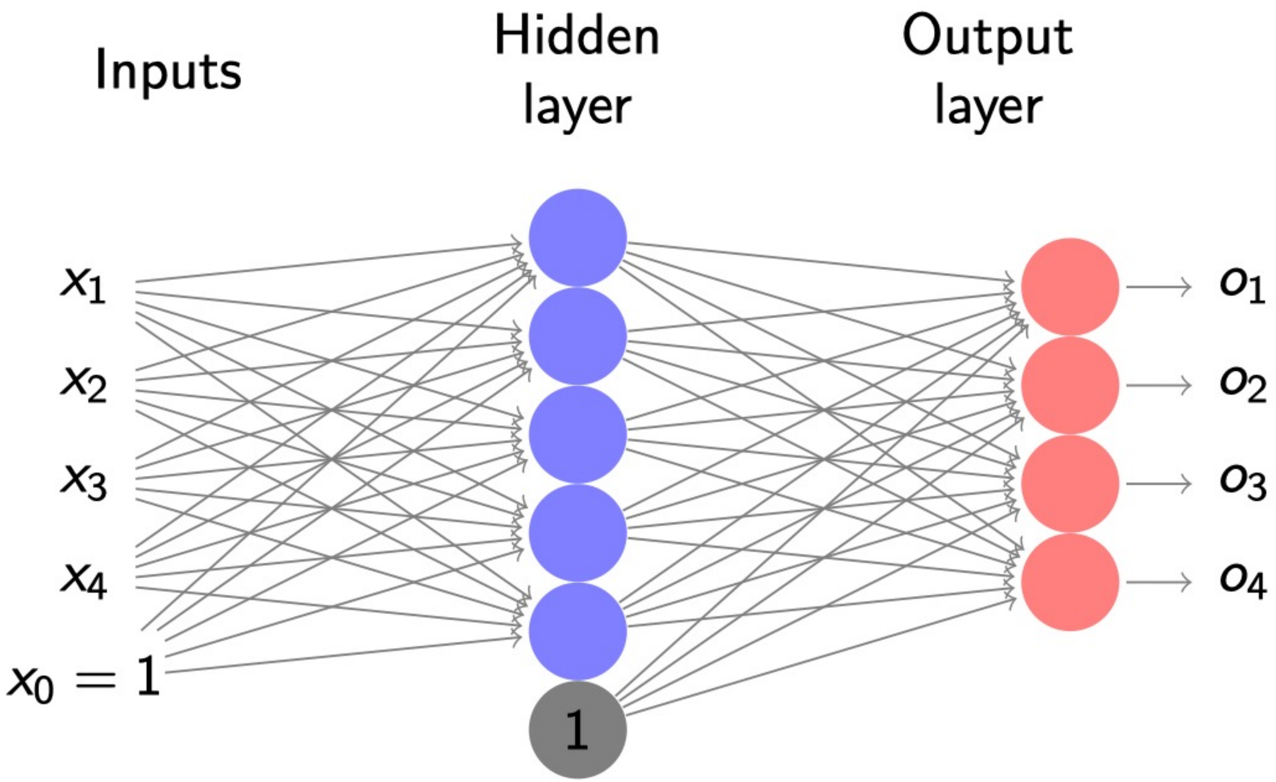
Perceptron

- Questions?
  - Why not work with thresholded perceptron?
    - Not differentiable
  - How to learn non-linear surfaces?
  - How to generalize to multiple outputs, numeric output?



Multiple Labels

- Distinguishing between multiple categories
- *Solution:* Add another layer - **Multi Layer Neural Networks**

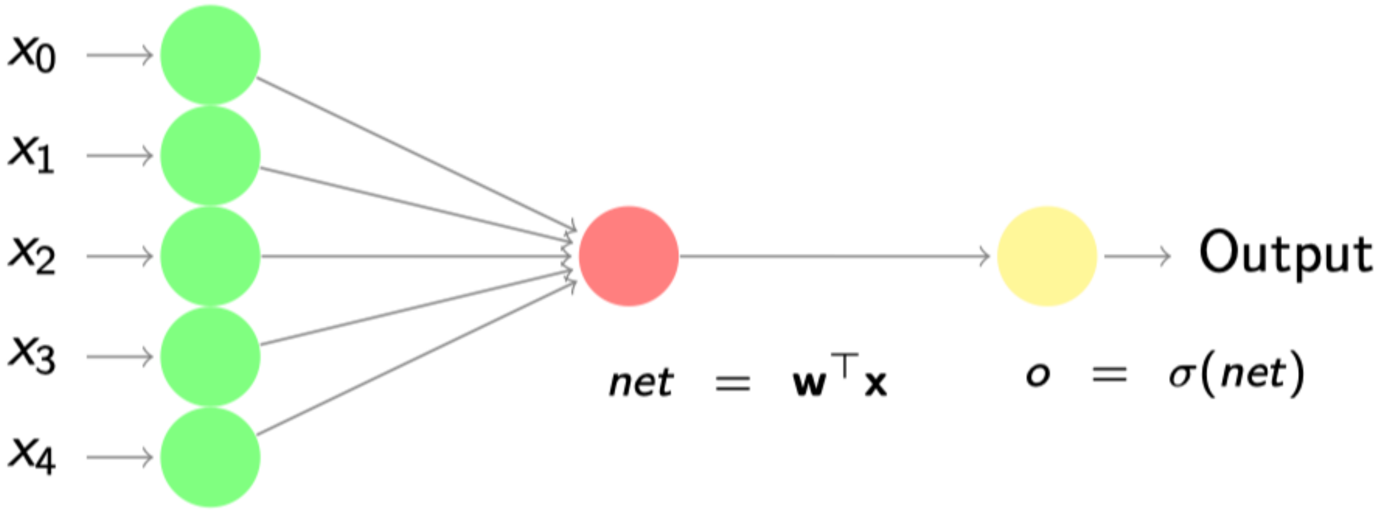


Threshold Unit (Activation Function)

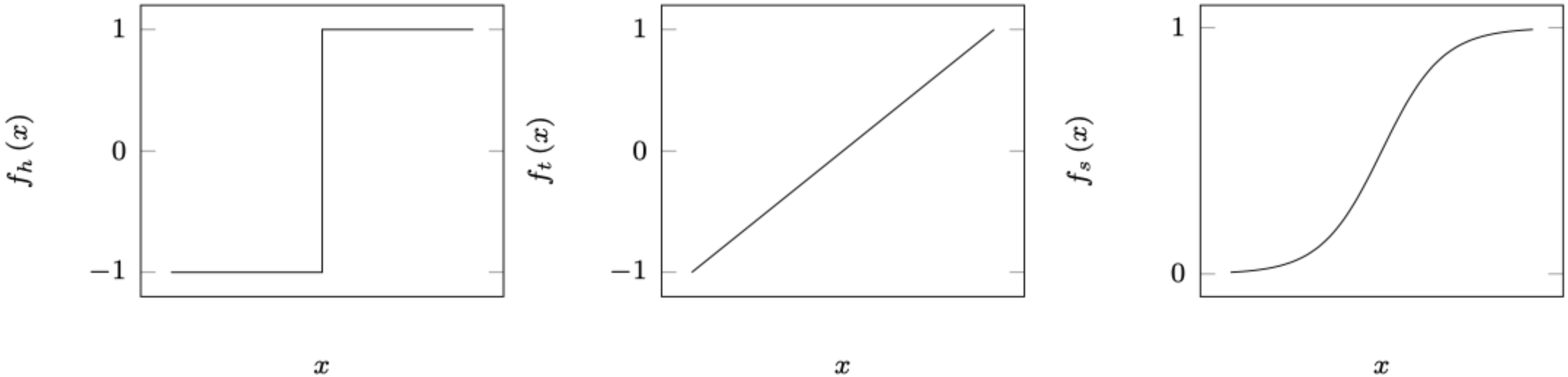
- ~~Linear Unit~~
- ~~Perceptron Unit~~
- Sigmoid Unit
  - Smooth, differentiable threshold function

$$\sigma(net) = \frac{1}{1 + e^{-net}}$$

  - Non-linear output



Properties of Sigmoid Function

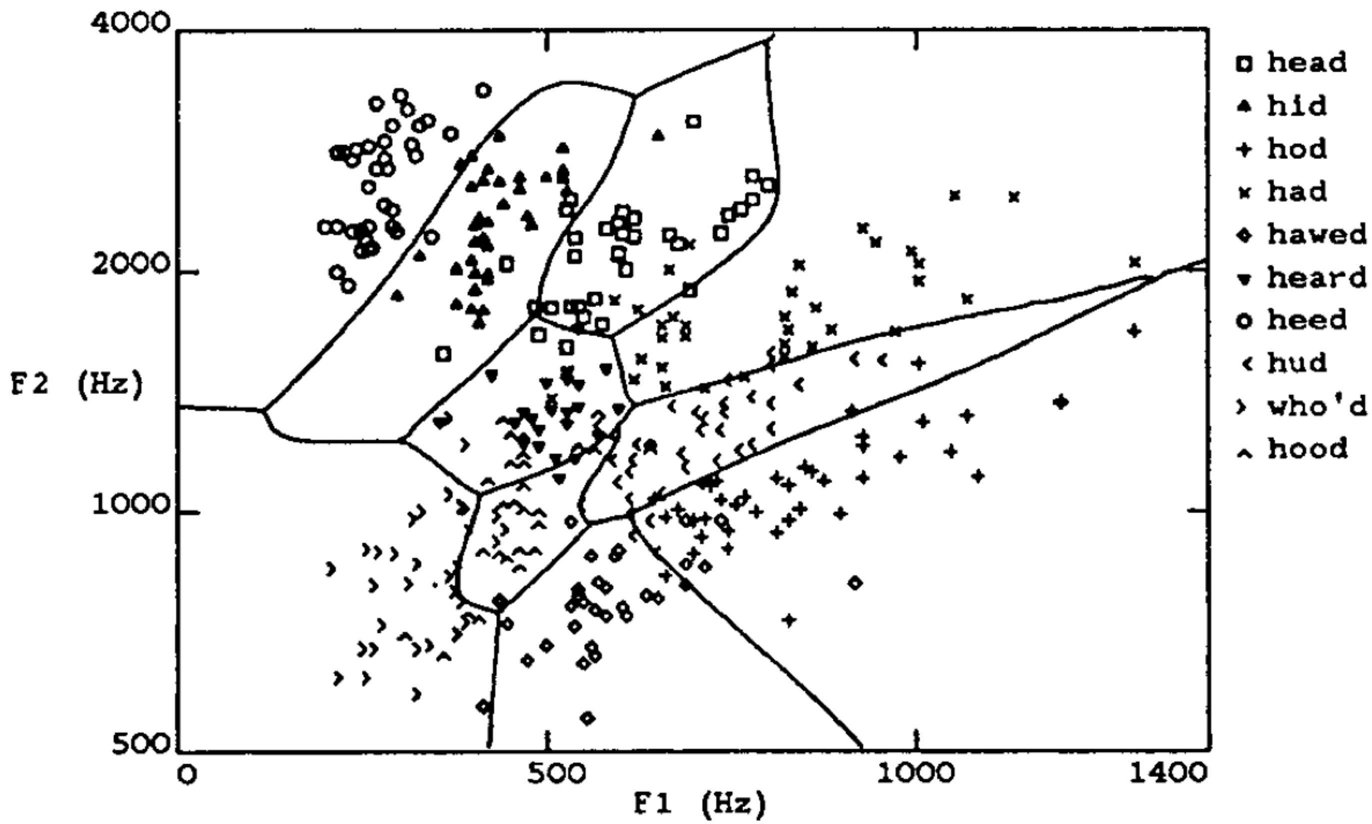


- The threshold output in the case of the sigmoid unit is continuous and smooth, as opposed to a perceptron unit or a linear unit. A useful property of sigmoid is that its derivative can be easily expressed as:

$$\frac{D\sigma(y)}{Dy} = \sigma(y)(1 - \sigma(y))$$

- One can also use e^{-ky} instead of e^{-y} , where k controls the "steepness" of the threshold curve.

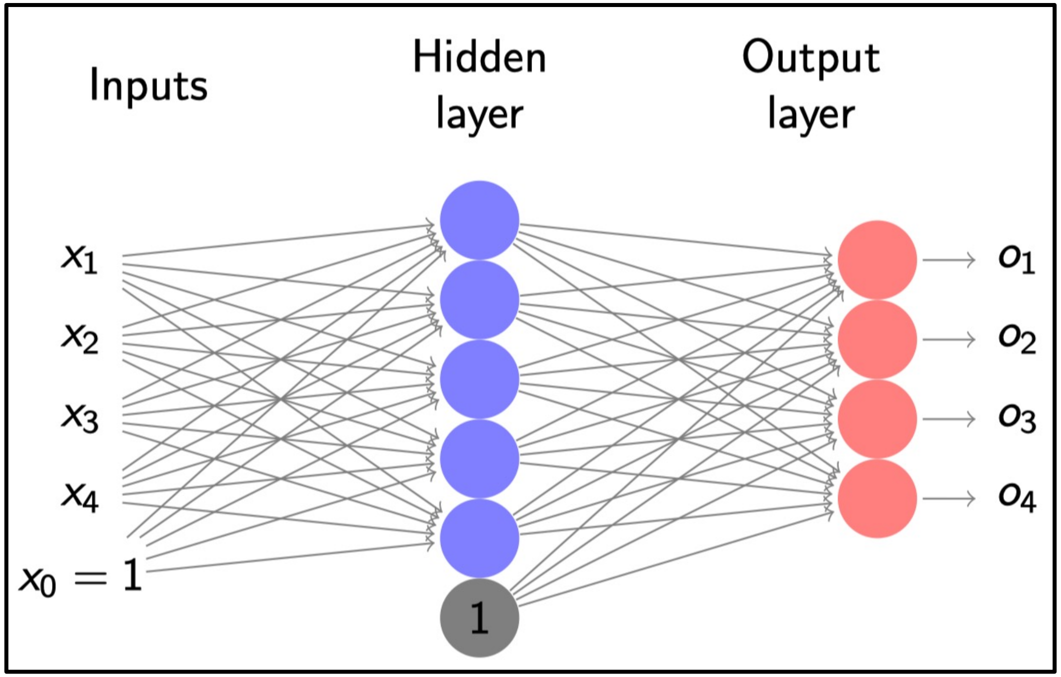
A Real-world Problem



- The learning problem is to recognize 10 different vowel sounds from the audio input. The raw sound signal is compressed into two features using spectral analysis.

Feed Forward Neural Networks

- D input nodes (excluding bias)
- M hidden nodes (excluding bias)
- K output nodes
- At hidden nodes: $\mathbf{w_j}, 1 \leq j \leq M$, $\mathbf{w_j} \in \mathbb{R}^{D+1}$
- At output nodes: $\mathbf{w_l}, 1 \leq l \leq K$, $\mathbf{w_l} \in \mathbb{R}^{M+1}$



- The multi-layer neural network shown above is used in a feed forward mode, i.e., information only flows in one direction (forward).
- Each hidden node "collects" the inputs from all input nodes and computes a weighted sum of the inputs and then applies the sigmoid function to the weighted sum. The output of each hidden node is forwarded to every output node.
- The output node "collects" the inputs (from hidden layer nodes) and computes a weighted sum of its inputs and then applies the sigmoid function to obtain the final output.
- The class corresponding to the output node with the largest output value is assigned as the predicted class for the input.

Backpropagation

- Assume that the network structure is predetermined (number of hidden nodes and interconnections)
- Objective function for N training examples:

$$J = \sum_{i=1}^{N} J_i = \frac{1}{2} \sum_{i=1}^{N} \sum_{l=1}^{K} (y_{il} - o_{il})^2$$

- y_{il} - Target value associated with l^{th} class for input ($\mathbf{x}_i$)
- $y_{il} = 1$ when k is true class for $\mathbf{x}_i$, and 0 otherwise
- o_{il} - Predicted output value at l^{th} output node for $\mathbf{x}_i$

What are we learning?

Weight vectors for all output and hidden nodes that minimize J

Backpropagation

1. Initialize all weights to *small values*
2. For each training example, $\langle \mathbf{x}, \mathbf{y} \rangle$:
   2.1 **Propagate input forward** through the network
   2.2 **Propagate errors backward** through the network

Backpropagation

Gradient Descent

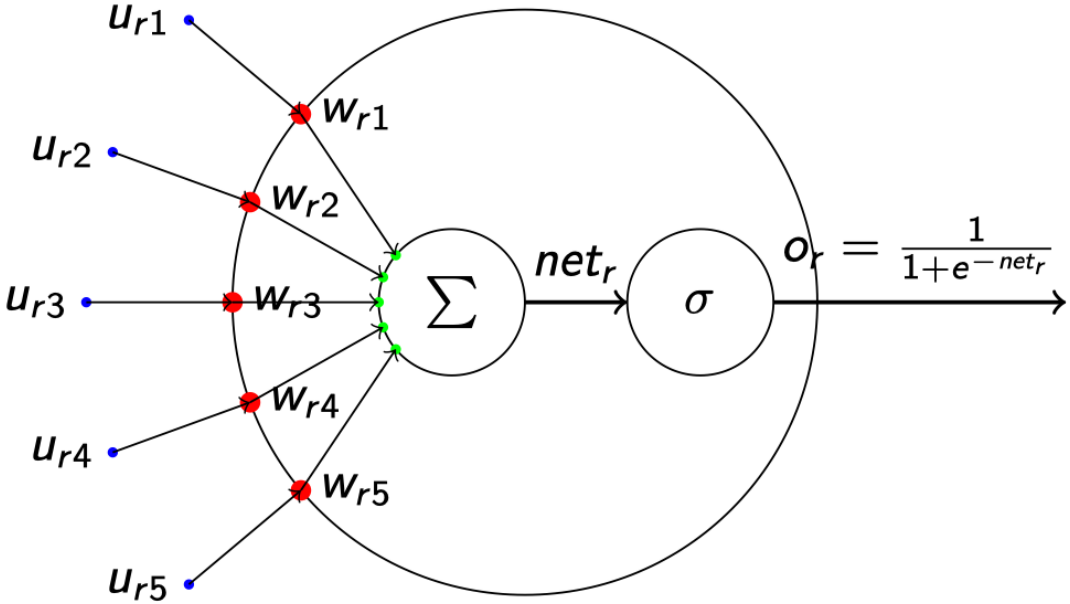
- Move in the opposite direction of the **gradient** of the objective function
- $-\eta \nabla J$

$$\nabla J = \sum_{i=1}^{N} \nabla J_i$$

- What is the gradient computed with respect to?
  - Weights - m at hidden nodes and k at output nodes
  - $\mathbf{w}_j$ $(j = 1 \ldots m)$
  - $\mathbf{w}_l$ $(l = 1 \ldots k)$
- $\mathbf{w}_j \leftarrow \mathbf{w}_j - \eta \frac{\partial J}{\partial \mathbf{w}_j} = \mathbf{w}_j - \eta \sum_{i=1}^{N} \frac{\partial J_i}{\partial \mathbf{w}_j}$
- $\mathbf{w}_l \leftarrow \mathbf{w}_l - \eta \frac{\partial J}{\partial \mathbf{w}_l} = \mathbf{w}_l - \eta \sum_{i=1}^{N} \frac{\partial J}{\partial \mathbf{w}_l}$

$$\nabla J_i = \begin{bmatrix} \frac{\partial J_i}{\partial \mathbf{w}_1} \\ \frac{\partial J_i}{\partial \mathbf{w}_2} \\ \vdots \\ \frac{\partial J_i}{\partial \mathbf{w}_{m+k}} \end{bmatrix}$$

Anatomy of a Sigmoid Unit (r)



$$\frac{\partial J_i}{\partial \mathbf{w}_r} = \begin{bmatrix} \frac{\partial J_i}{\partial w_{r1}} \\ \frac{\partial J_i}{\partial w_{r2}} \\ \vdots \end{bmatrix}$$

- Need to compute $\frac{\partial J_i}{\partial w_{rq}}$
- Update rule for the q^{th} entry in the r^{th} weight vector:

$$w_{rq} \leftarrow w_{rq} - \eta \frac{\partial J}{\partial w_{rq}} = w_{rq} - \eta \sum_{i=1}^{N} \frac{\partial J_i}{\partial w_{rq}}$$

Derivation of the Backpropagation

Assume that we only one training example, i.e., $i = 1$, $J = J_i$. Dropping the subscript i from here onwards.

- Consider any weight w_{rq}
- Let u_{rq} be the q^{th} element of the input vector coming in to the r^{th} unit.

Observation 1

Weight w_{rq} is connected to J through $net_r = \sum_i w_{r_i} u_{r_i}$.

$$\frac{\partial J}{\partial w_{rq}} = \frac{\partial J}{\partial net_r} \frac{\partial net_r}{\partial w_{rq}} = \frac{\partial J}{\partial net_r} u_{rq}$$

Derivation of the Backpropagation

Observation 2

net_l for an **output node** is connected to J only through the output value of the node (or o_l)

$$\frac{\partial J}{\partial net_l} = \frac{\partial J}{\partial o_l} \frac{\partial o_l}{\partial net_l}$$

The first term above can be computed as:

$$\frac{\partial J}{\partial o_l} = \frac{\partial}{\partial o_l} \frac{1}{2} \sum_{l=1}^{K} (y_l - o_l)^2$$

The entries in the summation in the right hand side will be non zero only for l. This results in:

$$\begin{aligned} \frac{\partial J}{\partial o_l} &= \frac{\partial}{\partial o_l} \frac{1}{2} (y_l - o_l)^2 \\ &= -(y_l - o_l) \end{aligned}$$

Moreover, the second term in the chain rule above can be computed as:

$$\begin{aligned} \frac{\partial o_l}{\partial net_l} &= \frac{\partial \sigma(net_l)}{\partial net_l} \\ &= o_l(1 - o_l) \end{aligned}$$

Derivation of the Backpropagation

Update Rule for Output Units

$$w_{lj} \leftarrow w_{lj} + \eta \delta_l u_{lj}$$

where $\delta_l = (y_l - o_l)o_l(1 - o_l)$.

- *Question:* What is u_{lj} for the l^{th} output node?

Derivation of the Backpropagation

Observation 3

net_j for a **hidden node** is connected to J through all output nodes

$$\frac{\partial J}{\partial net_j} = \sum_{l=1}^{K} \frac{\partial J}{\partial net_l} \frac{\partial net_l}{\partial net_j}$$

Remember that we have already computed the first term on the right hand side for output nodes:

$$\frac{\partial J}{\partial net_l} = -\delta_l$$

where $\delta_l = (y_l - o_l)o_l(1 - o_l)$. This result gives us:

$$\begin{aligned}
\frac{\partial J}{\partial net_j} &= \sum_{l=1}^{K} -\delta_l \frac{\partial net_l}{\partial net_j} = \sum_{l=1}^{K} -\delta_l \frac{\partial net_l}{\partial z_j} \frac{\partial z_j}{\partial net_j} \\
&= \sum_{l=1}^{K} -\delta_l w_{lj} \frac{\partial z_j}{\partial net_j} = \sum_{l=1}^{K} -\delta_l w_{lj} z_j (1 - z_j) \\
&= -z_j(1 - z_j) \sum_{l=1}^{K} \delta_l w_{lj}
\end{aligned}$$

Derivation of the Backpropagation

Update Rule for Hidden Units

$$w_{jp} \leftarrow w_{jp} + \eta\delta_j u_{jp}$$

$$\delta_j = o_j(1 - o_j)\sum_{l=1}^{K}\delta_l w_{lj}$$

$$\delta_l = (y_l - o_l)o_l(1 - o_l)$$

- *Question:* What is u_{jp} for the j^{th} hidden node?

Final Algorithm

- While not converged:
  - *Move forward* to compute outputs at hidden and output nodes
  - *Move backward* to propagate errors back
    - Compute δ errors at output nodes (δ_l)
    - Compute δ errors at hidden nodes (δ_j)
  - Update all weights according to weight update equations

Conclusion about NN

- Error function contains many local minima
- No guarantee of convergence
  - Not a “big” issue in practical deployments
- Improving backpropagation
  - Adding momentum
  - Using stochastic gradient descent
  - Train multiple times using different initializations

Bias Variance Tradeoff

- Neural networks are *universal function approximators*
  - By making the model more complex (increasing number of hidden layers or m) one can lower the error
- Is the model with least training error the best model?
  - The simple answer is **no**!
  - Risk of overfitting (chasing the data)
  - Overfitting $\Leftarrow$ **High generalization error**

High Variance - Low Bias

- "Chases the data"
- Very low training error
- Poor performance on unseen data

Low Variance - High Bias

- Less sensitive to training data
- Higher training error
- Better performance on unseen data

Bias Variance Tradeoff

- General rule of thumb – If two models are giving similar training error, choose the **simpler** model
- What is simple for a neural network?
- Low weights in the weight matrices?
  - Why?

Introducing Bias

- Penalize solutions in which the weights are high
- Can be done by introducing a penalty term in the objective function
  - **Regularization**

Regularization for Backpropagation

$$\widetilde{J} = J + \frac{\lambda}{2n}\left(\sum_{j=1}^{M}\sum_{i=1}^{D+1}(w_{ji}^{(1)})^2 + \sum_{l=1}^{K}\sum_{j=1}^{M+1}(w_{lj}^{(2)})^2\right)$$

Questions?

UNIVERSITY OF NORTH CAROLINA
CHARLOTTE