CS 615 – Deep Learning

Artificial Neural Networks

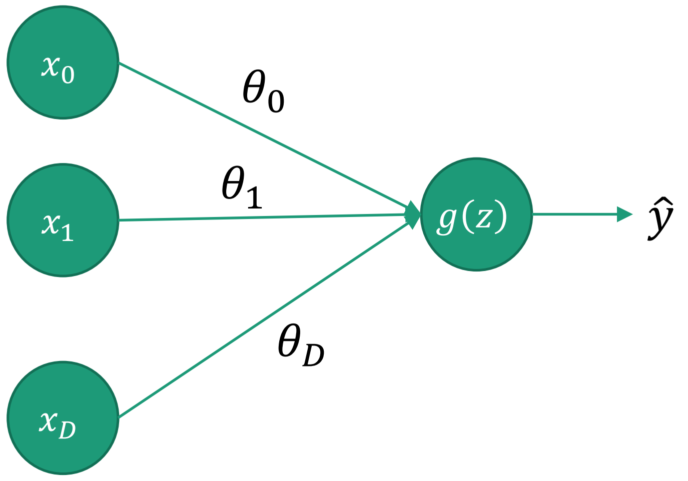
Slides adapted from material created by E. Alpaydin
Prof. Mordohai, Prof. Greenstadt, Pattern Classification (2nd Ed.),
Pattern Recognition and Machine Learning

Objectives

- Artificial Neurons as Networks/Graphs
- Shallow Artificial Neural Networks (ANNs)
- Forward Propagation
- Backwards Propagation

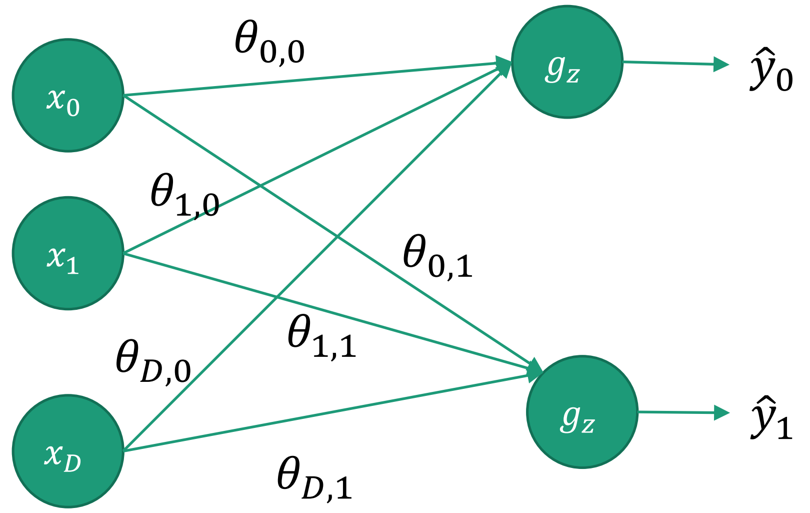
Graphical Model

- Another way we can think of linear regression and perceptrons is as a graph/network.
- In a graphical model, we have both input nodes (for the features) and an output node.
- The weights can be thought of as edges between the input nodes and the output nodes.



Multi-Output Graph

- We can extend this to have K output nodes (the multi-class scenario).
- The weight $\theta_{i,k}$ will now represent the edge going from input node i to output node k

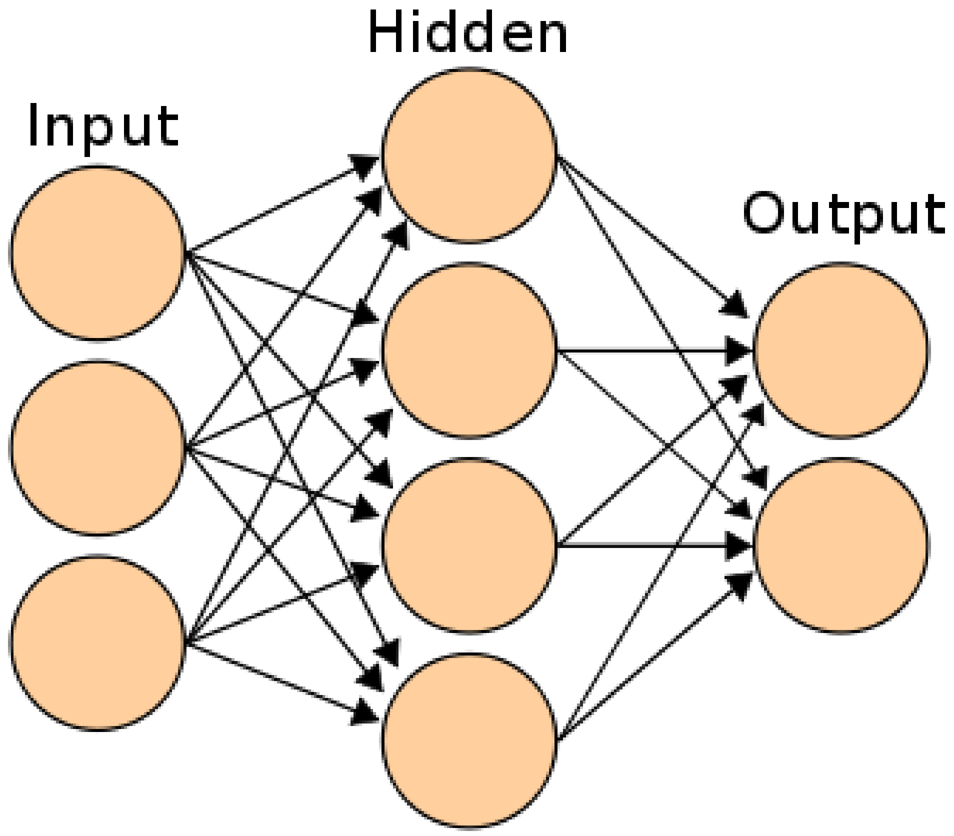


Graph Layers

- We often refer to the set of input nodes as the *input layer* and the set of output nodes as the *output layer*.
- Networks that only have an input and output layer (like linear regression and perceptron models) are somewhat limited.
    - They can only learn linear concepts.
- If we add additional layers between the input and output layers, then we can learn more complicated concepts.
- These sort of graphs are called *artificial neural networks (ANNs)*

Artificial Neural Network (ANN)

1. Input layer (features)
2. Hidden layer(s)
3. Output layer (predicted values)



ANN Design Choices

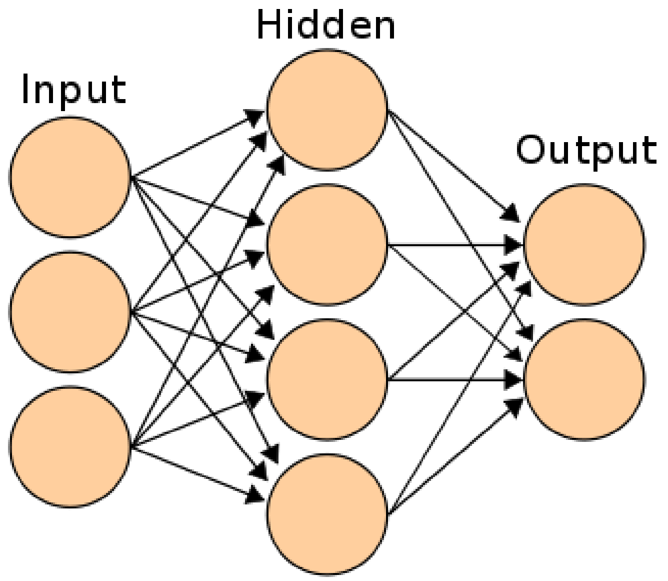
- ANNs require some design choices!
  - Number of nodes in the hidden layer
  - Activation function at hidden layer and output layer.
- As usual we need to decide on the objective function and the activation function at the output.
- But now we also need to decide on:
  - Number of nodes in the hidden layer
  - The activation function in the hidden layer.

ANN Design Choices

- Although there are many potential objective functions, here's what we've learned thus far:
  - Squared Error
  - Log Likelihood
  - Cross Entropy
- And as far as the activation functions:
  - Linear: $g(z) = z$
  - Logistic: $g(z) = \frac{1}{1+e^{-z}}$
  - Softmax: $g(z) = \frac{e^{z}}{\sum_{k=1}^{K} e^{z_k}}$
- And some additional common activations functions include:
  - Rectified Linear: $g(z) = \max\{0, z\}$
  - Hyperbolic Tangent $g(z) = \tanh(z)$

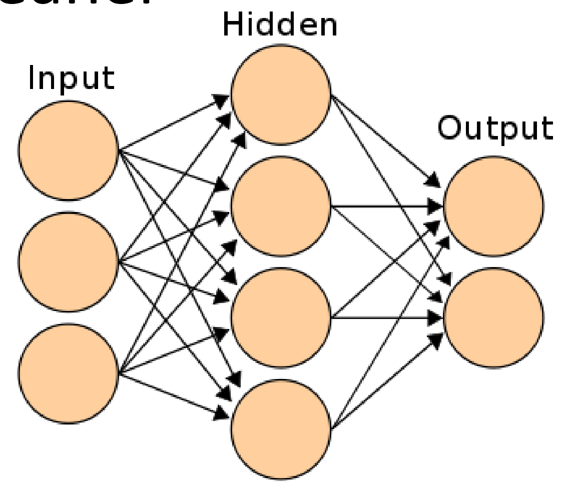
Training an ANN

- Now we have lots of weights to learn
  - Each hidden layer node weights the output of the input layer.
  - Each output layer node weights the output of the hidden layer node.
- Again we're going to learn these weights via iterative updating of our parameters using gradients.
- In the context of ANNs, the gradient update process is often referred to as *forward-backward propagation.*



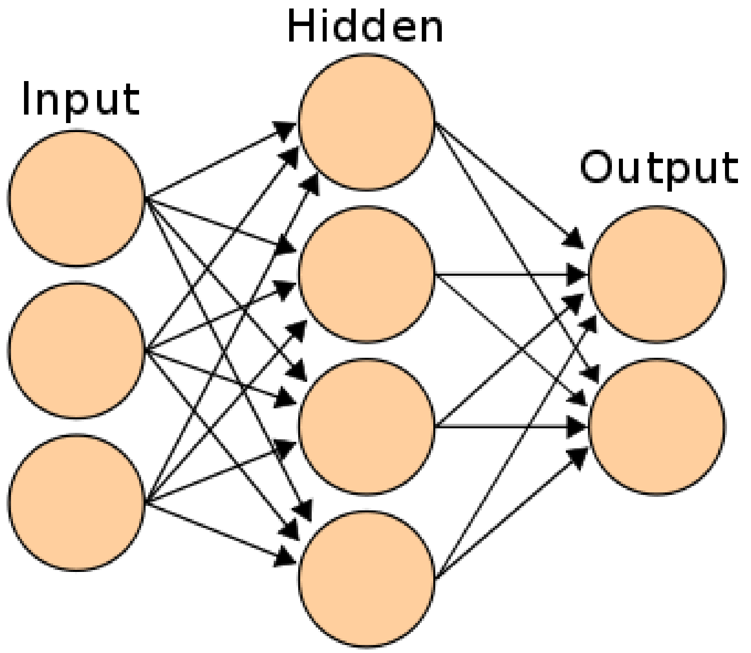
ANN Notation

- Let's add in some more notation to make things cleaner
- Our artificial neural network will have
  - D input nodes
  - M hidden nodes
  - K output nodes
- We will call the input to node j, net_j
- The output from node j after being processed by activation function g with then be $g(net_j)$
  - For the hidden layer, this output will be denoted h_j
  - For the output layer this will be denoted $\hat{y}_k$

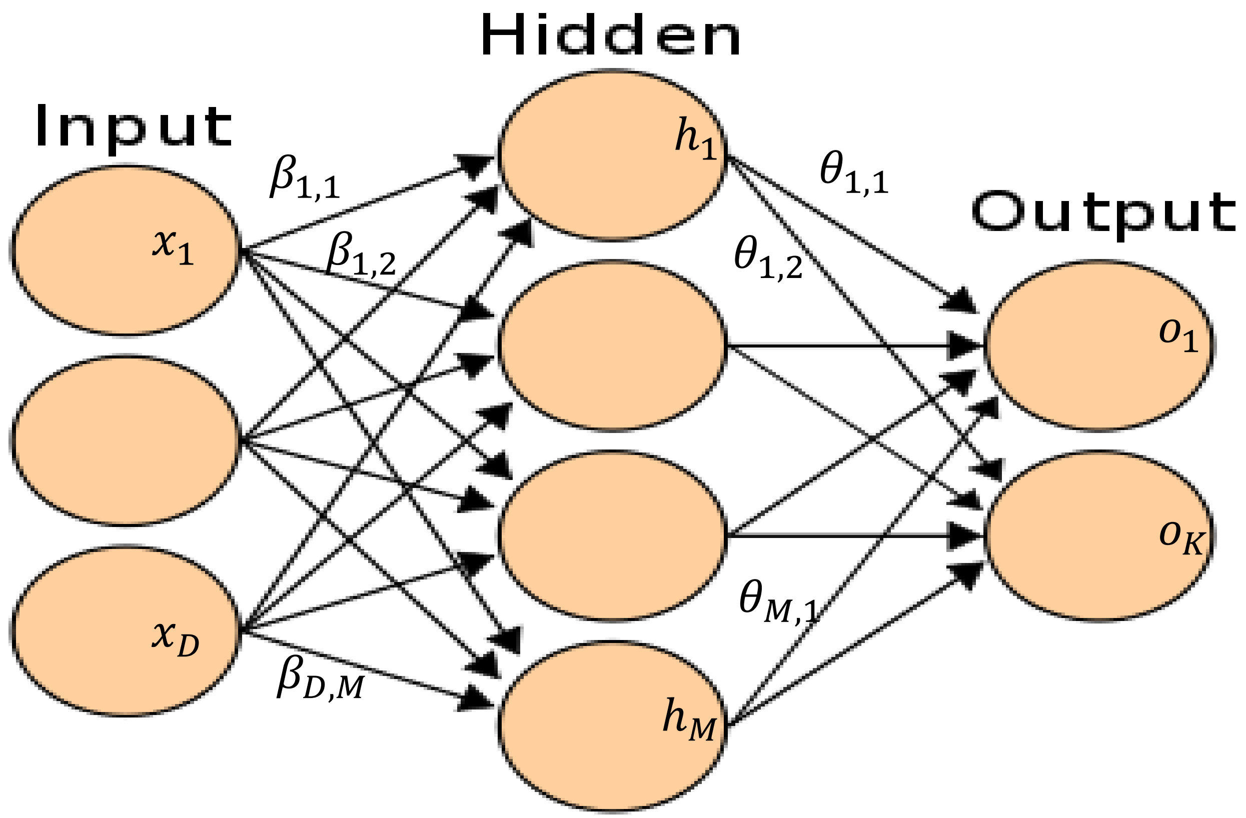


ANN Notation

- A matrix (or vector of vectors) of weights connecting nodes from one layer to the next.
- The weights connecting to the input layer to the hidden layer will be denoted β, such that the connection from input layer node i to hidden layer node j is β_{ij}
- The weights connecting the hidden layer to output layer will be θ, such that the connection from hidden layer node j to output layer node k is θ_{jk}



ANN Notation



Forward Propagation

- Forward propagation allows us to determine the values going in and out of each node, starting at the input layer (i.e the features) and ending with the output of the output layer.
- The values will be used to update the weights/parameters.
- Once trained, this same process is used to produce our predicted output values.

Forward Propagation

- Let’s forward propagate!
- Given input values x and an activation function $g(z)$, we can compute the values going into the hidden layers as $net_h = x\beta$
- The values coming out of the hidden, layer, as processed by its activation function, are then:

$$h = g(net_h)$$

- Next we take those values, and compute the values going into the output layers as $net_y = h\theta$
- The values coming out of the output layer is then:

$$\hat{y} = g\left(net_y\right)$$

- Lets look at an example of this, where we have linear activation functions at both the hidden and output layers.

Forward Propagation Example

- Input to hidden layer:

$$net_h = x\beta = \begin{bmatrix}0.35 & 0.9\end{bmatrix}\begin{bmatrix}0.1 & 0.4\\ 0.8 & 0.6\end{bmatrix} = \begin{bmatrix}0.755 & 0.68\end{bmatrix}$$

- Output of hidden layer:

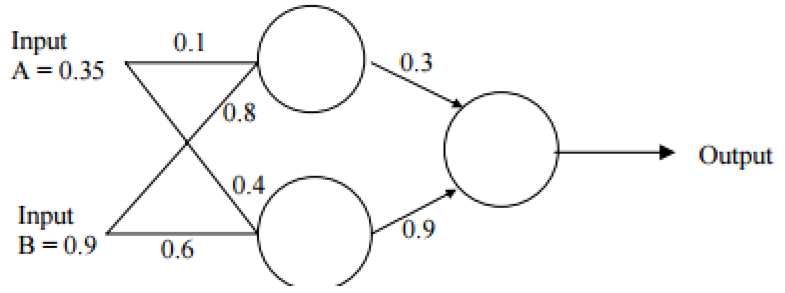
$$h = g(net_h) = \begin{bmatrix}0.755 & 0.68\end{bmatrix}$$

- Input to output layer

$$net_o = h\theta = \begin{bmatrix}0.755 & 0.68\end{bmatrix}\begin{bmatrix}0.3\\ 0.9\end{bmatrix} = \begin{bmatrix}0.8385\end{bmatrix}$$

- Output of output layer

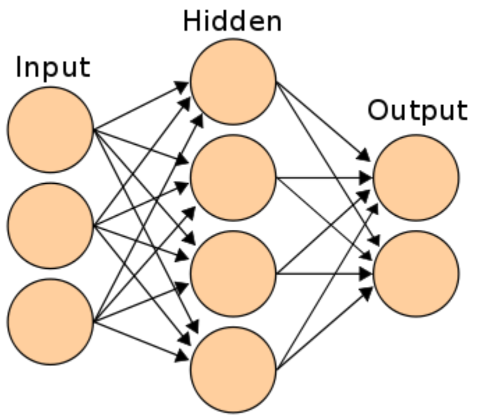
$$\hat{y} = g(net_o) = \begin{bmatrix}0.8385\end{bmatrix}$$



Back Propagation

- Imagine that we know that our desired output is 1.0.
- Now we want to update our parameter θ and β to get us closer to that result.
- To update those parameters we'll need the gradient of the objective function with regards to those parameters:

$$\frac{\partial J}{\partial \theta_{j,k}} \quad \frac{\partial J}{\partial \beta_{i,j}}$$

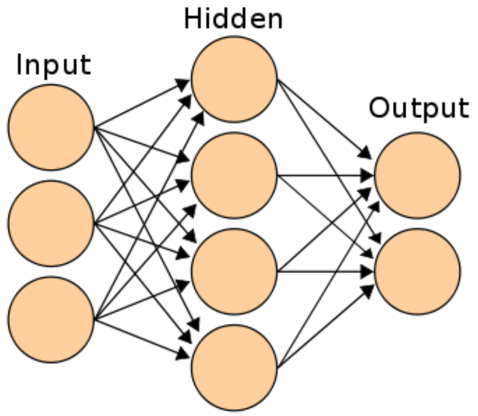


Back Propagation

- For illustrative purposes lets choose the **squared error** as our objective function.
- For simplicity, let’s assume a single output.
- Therefore, for a single observation:

$$J = (y - \hat{y})^2 = \left(y - g(net_o)\right)^2$$

- Lets try to find the gradient rules!
- And now we’ll see how leveraging the chain rule can help us!



Back Propagation Derivation

$$J = \left(y - g(net_o)\right)^2$$

- It's best to work from the output backwards (hence, *back propagation),* so lets find $\frac{\partial J}{\partial \theta}$ *first.*
  - *Note: Since we only have a single output for now,* θ *will only have one subscript (the hidden layer node number)*
- Continuing what we did with the chain rule:

$$\frac{\partial J}{\partial \theta_j} = \frac{\partial J}{\partial g(net_o)} \cdot \frac{\partial g(net_o)}{\partial net_o} \cdot \frac{\partial net_o}{\partial \theta_j}$$

- We'll find that it's often easiest to start at the right and move left.

Back Propagation Derivation

$$\frac{\partial J}{\partial \theta_j} = \frac{\partial J}{\partial g(net_o)} \cdot \frac{\partial g(net_o)}{\partial net_o} \cdot \frac{\partial net_o}{\partial \theta_j}$$

- So what is $\frac{\partial net_o}{\partial \theta_j}$?
- Recall that $net_o = h\theta$

$$\frac{\partial net_o}{\partial \theta_j} = h_j$$

Back Propagation Derivation

$$\frac{\partial J}{\partial \theta_j} = \frac{\partial J}{\partial g(net_o)} \cdot \frac{\partial g(net_o)}{\partial net_o} \cdot \boldsymbol{h_j}$$

- How about $\frac{\partial g(net_o)}{\partial net_o}$?
- Since we're using the linear activation function, $g(net_o) = net_o$

$$\frac{\partial g(net_o)}{\partial net_o} = 1$$

Back Propagation Derivation

$$\frac{\partial J}{\partial \theta_j} = \frac{\partial J}{\partial g(net_o)} \cdot \mathbf{1} \cdot \boldsymbol{h}_j$$

- How about $\frac{\partial J}{\partial g(net_o)}$?
- Recall that using the squared error objective function we have

$$J = \left(y - g(net_o)\right)^2$$

- Therefore:

$$\frac{\partial J}{\partial g(net_o)} = -2\left(y - g(net_o)\right) = 2(\hat{y} - y)$$

- Putting these together we have:

$$\frac{\partial J}{\partial \theta_j} = 2(\hat{y} - y)h_j$$

Back Propagation Derivation

$$\frac{\partial J}{\partial \theta_j} = 2(\hat{y} - y)h_j$$

- Vectorizing the parameters:

$$\frac{\partial J}{\partial \theta} = 2h^T(\hat{y} - y)$$

- For batch mode:

$$\frac{\partial J}{\partial \theta} = \frac{2}{N}H^T\left(\hat{Y} - Y\right)$$

Back Propagation Derivation

$$J = \left(y - g(net_o)\right)^2$$

- That wasn't so bad!
- How about $\frac{\partial J}{\partial \beta}$?
- Ah…. this is trickier
  - Since these come from before the hidden layer.
- But we can handle it if we use the chain rule!

$$\frac{\partial J}{\partial \beta_{i,j}} = \frac{\partial J}{\partial g(net_o)} \cdot \frac{\partial g(net_o)}{\partial net_o} \cdot \frac{\partial net_o}{\partial g\left(net_{h\,j}\right)} \cdot \frac{\partial g\left(net_{h\,j}\right)}{\partial net_{h_j}} \cdot \frac{\partial net_{h_j}}{\partial \beta_{ij}}$$

Back Propagation Derivation

$$\frac{\partial J}{\partial \beta_{i,j}} = \frac{\partial J}{\partial g(net_o)} \cdot \frac{\partial g(net_o)}{\partial net_o} \cdot \frac{\partial net_o}{\partial g\left(net_{h_j}\right)} \cdot \frac{\partial g\left(net_{h_j}\right)}{\partial net_{h_j}} \cdot \frac{\partial net_{h_j}}{\partial \beta_{ij}}$$

- What is $\frac{\partial net_{h_j}}{\partial \beta_{ij}}$?

$$\frac{\partial net_{h_j}}{\partial \beta_{ij}} = x_i$$

Back Propagation Derivation

$$\frac{\partial J}{\partial \beta_{i,j}} = \frac{\partial J}{\partial g(net_o)} \cdot \frac{\partial g(net_o)}{\partial net_o} \cdot \frac{\partial net_o}{\partial g\left(net_{h_j}\right)} \cdot \frac{\partial g\left(net_{h_j}\right)}{\partial net_{h_j}} \cdot \boldsymbol{x_i}$$

- Given linear activation function at the hidden layer what is $\frac{\partial g\left(net_{h_j}\right)}{\partial net_{h_j}}$?

$$\frac{\partial g\left(net_{h_j}\right)}{\partial net_{h_j}} = 1$$

Back Propagation Derivation

$$\frac{\partial J}{\partial \beta_{i,j}} = \frac{\partial J}{\partial g(net_o)} \cdot \frac{\partial g(net_o)}{\partial net_o} \cdot \frac{\partial net_o}{\partial g\left(net_{h\,j}\right)} \cdot \mathbf{1} \cdot \boldsymbol{x_i}$$

- What is $\frac{\partial net_o}{\partial g\left(net_{h_j}\right)}$?

$$\frac{\partial net_o}{\partial g\left(net_{h_j}\right)} = \frac{\partial net_o}{\partial h_j} = \theta_j$$

Back Propagation Derivation

$$\frac{\partial J}{\partial \beta_{i,j}} = \frac{\partial J}{\partial g(net_o)} \cdot \frac{\partial g(net_o)}{\partial net_o} \cdot \boldsymbol{\theta_j} \cdot \mathbf{1} \cdot \boldsymbol{x_i}$$

- What is $\frac{\partial g(neo)}{\partial net_o}$ if we have a linear output activation function?

$$\frac{\partial g(net_o)}{\partial net_o} = 1$$

Back Propagation Derivation

$$\frac{\partial J}{\partial \beta_{i,j}} = \frac{\partial J}{\partial g(net_o)} \cdot \mathbf{1} \cdot \boldsymbol{\theta_j} \cdot \mathbf{1} \cdot \boldsymbol{x_i}$$

- And finally, what is $\frac{\partial J}{\partial g(net_o)}$?
- Recall for squared error $J = (y - \hat{y})^2$
- Therefore, $\frac{\partial g(net_o)}{\partial net_o} = 2(\hat{y} - y)$
- Putting it all together…

$$\frac{\partial J}{\partial \beta_{i,j}} = 2(\hat{y} - y) \cdot 1 \cdot \theta_j \cdot 1 \cdot x_i = 2x_i(\hat{y} - y)\theta_j$$

Back Propagation Derivation

$$\frac{\partial J}{\partial \beta_{i,j}} = 2(\hat{y} - y) \cdot 1 \cdot \theta_j \cdot 1 \cdot x_i = 2x_i(\hat{y} - y)\theta_j$$

- Vectorizing to update all parameters of β at once:

$$\frac{\partial J}{\partial \beta} = 2x^T(\hat{y} - y)\theta$$

- For batch mode:

$$\frac{\partial J}{\partial \beta} = \frac{2}{N} X^T\left(\hat{Y} - Y\right)\theta^T$$

Multi-Output Back Propagation

- How would things change if we had more than one output?
- If we have multiple outputs, our squared error objective function then becomes:

$$J = \sum_{k=1}^{K} J_k(x, y) = \sum_{k=1}^{K} (y_k - \widehat{y_k})^2 = \sum_{k=1}^{K} \left(y_k - g\left(net_{o_k}\right)\right)^2$$

- Now we need to find θ_{jk} (as opposed to just θ_j).
- The chain rule for this will be:

$$\frac{\partial J}{\partial \theta_{j,k}} = \frac{\partial J}{\partial g\left(net_{o_k}\right)} \cdot \frac{\partial g\left(net_{o_k}\right)}{\partial net_{o_k}} \cdot \frac{\partial net_{o_k}}{\partial \theta_{jk}}$$

Multi-Output Back Propagation

$$\frac{\partial J}{\partial \theta_{j,k}} = \frac{\partial J}{\partial g\left(net_{o_k}\right)} \cdot \frac{\partial g\left(net_{o_k}\right)}{\partial net_{o_k}} \cdot \frac{\partial net_{o_k}}{\partial \theta_{jk}}$$

- So what is $\frac{\partial net_{o_k}}{\partial \theta_{j,k}}$?
- Recall that $net_{o_k} = h\theta_{:k}$

$$\frac{\partial net_{o_k}}{\partial \theta_{j,k}} = h_j$$

Multi-Output Back Propagation

$$\frac{\partial J}{\partial \theta_{j,k}} = \frac{\partial J}{\partial g\left(net_{o_k}\right)} \cdot \frac{\partial g\left(net_{o_k}\right)}{\partial net_{o_k}} \cdot \boldsymbol{h_j}$$

- How about $\frac{\partial g\left(net_{o_k}\right)}{\partial net_{o_k}}$?
- Since we're using the linear activation function, $g\left(net_{o_k}\right) = net_{o_k}$

$$\frac{\partial g\left(net_{o_k}\right)}{\partial net_{o_k}} = 1$$

Multi-Output Back Propagation

$$\frac{\partial J}{\partial \theta_{j,k}} = \frac{\partial J(x,y)}{\partial g\left(net_{o_k}\right)} \cdot 1 \cdot \boldsymbol{h_j}$$

- How about $\frac{\partial J}{\partial g\left(net_{o_k}\right)}$?
- Recall that using the squared error objective function we have

$$J = \sum_{k=1}^{K} J_k(x,y) = \sum_{k=1}^{K} \left(y_k - g\left(net_{o_k}\right)\right)^2$$

- But the only term in the summation that will be non-zero is when $k = k$
- Therefore:

$$\frac{\partial J}{\partial g\left(net_{o_k}\right)} = \frac{\partial J_k}{\partial g\left(net_{o_k}\right)} - 2\left(y - g\left(net_{o_k}\right)\right) = 2(\widehat{y_k} - y_k)$$

- Putting these together we have:

$$\frac{\partial J}{\partial \theta_{j,k}} = 2(\widehat{y_k} - y_k)h_j$$

Multi-Output Back Propagation

$$\frac{\partial J}{\partial \theta_{j,k}} = 2(\widehat{y_k} - y_k)h_j$$

- Vectorizing this:

$$\frac{\partial J}{\partial \theta} = 2h^T(\hat{y} - y)$$

- For batch mode:

$$\frac{\partial J}{\partial \theta} = \frac{2}{N}H^T(\hat{Y} - Y)$$

Multi-Output Back Propagation

$$J = \sum_{k=1}^{K} J_k = \sum_{k=1}^{K} \left(y_k - g\left(net_{o_k}\right)\right)^2$$

- Next up, finding the gradient with respect to β
- Since we're trying to find β_{ij} and this does not have a specific output k, we need to take the derivative of **each** term of the summation:

$$\frac{\partial J}{\partial \beta_{ij}} = \sum_{k=1}^{K} \frac{\partial J_k}{\partial \beta_{ij}}$$

- $\frac{\partial J}{\partial \beta_{i,j}} = \sum_{k=1}^{K} \left(\frac{\partial J_k}{\partial g\left(net_{o_k}\right)} \cdot \frac{\partial g\left(net_{o_k}\right)}{\partial net_{o_k}} \cdot \frac{\partial net_{o_k}}{\partial g\left(net_{h_j}\right)} \cdot \frac{\partial g\left(net_{h_j}\right)}{\partial net_{h_j}} \cdot \frac{\partial net_{h_j}}{\partial \beta_{ij}} \right)$

Multi-Output Back Propagation

$$\frac{\partial J}{\partial \beta_{i,j}} = \sum_{k=1}^{K} \left(\frac{\partial J_k}{\partial g(net_{o_k})} \cdot \frac{\partial g(net_{o_k})}{\partial net_{o_k}} \cdot \frac{\partial net_{o_k}}{\partial g\left(net_{h\,j}\right)} \cdot \frac{\partial g\left(net_{h\,j}\right)}{\partial net_{h_j}} \cdot \frac{\partial net_{h_j}}{\partial \beta_{ij}} \right)$$

- The good news is that we already did the last two terms:

$$\frac{\partial net_{h_j}}{\partial \beta_{ij}} = x_i$$

$$\frac{\partial g\left(net_{h\,j}\right)}{\partial net_{h_j}} = 1$$

Multi-Output Back Propagation

$$\frac{\partial J}{\partial \beta_{i,j}} = \sum_{k=1}^{K} \left(\frac{\partial J_k}{\partial g(net_{o_k})} \cdot \frac{\partial g(net_{o_k})}{\partial net_{o_k}} \cdot \frac{\partial net_{o_k}}{\partial g\left(net_{h_j}\right)} \cdot \mathbf{1} \cdot x_i \right)$$

- And the remaining terms shouldn't be too bad…

$$\frac{\partial net_{o_k}}{\partial g\left(net_{h_j}\right)} = \frac{\partial net_{o_k}}{\partial h_j} = \theta_{jk}$$

$$\frac{\partial g\left(net_{o_k}\right)}{\partial net_{o_k}} = 1$$

$$\frac{\partial g\left(net_{o_k}\right)}{\partial net_{o_k}} = 2(\hat{y}_k - y_k)$$

Multi-Output Back Propagation

- Putting it all together…

$$\frac{\partial J}{\partial \beta_{i,j}} = \sum_{k=1}^{K} \left(2(\hat{y}_k - y_k) \cdot 1 \cdot \theta_{jk} \cdot 1 \cdot x_i\right) = 2x_i \sum_{k=1}^{K} (\hat{y}_k - y_k)\theta_{jk}$$

- Vectorizing to update all values of β at once:

$$\frac{\partial J(x,y)}{\partial \beta} = 2x^T(\hat{y} - y)\theta$$

- For batch mode:

$$\frac{\partial J}{\partial \beta} = \frac{2}{N} X^T\left(\hat{Y} - Y\right)\theta^T$$

Other Objective and Activations Functions

- We'll want to be able to easily modify this to work for different objective functions and gradient functions.
- Returning to our generalized chain rule gradients:

$$\frac{\partial J}{\partial \theta_{j,k}} = \frac{\partial J}{\partial g\left(net_{o_k}\right)} \cdot \frac{\partial g\left(net_{o_k}\right)}{\partial net_{o_k}} \cdot \frac{\partial net_{o\,k}}{\partial \theta_{jk}}$$

$$\frac{\partial J}{\partial \beta_{i,j}} = \sum_{k=1}^{K} \left(\frac{\partial J_k}{\partial g\left(net_{o_k}\right)} \cdot \frac{\partial g\left(net_{o_k}\right)}{\partial net_{o_k}} \cdot \frac{\partial net_{o_k}}{\partial g\left(net_{h\,j}\right)} \cdot \frac{\partial g\left(net_{h\,j}\right)}{\partial net_{h_j}} \cdot \frac{\partial net_{h_j}}{\partial \beta_{ij}} \right)$$

Other Objective and Activations Functions

$$\frac{\partial J}{\partial \theta_{j,k}} = \frac{\partial J}{\partial g(net_{o_k})} \cdot \frac{\partial g(net_{o_k})}{\partial net_{o_k}} \cdot \frac{\partial net_{o\,k}}{\partial \theta_{jk}}$$

$$\frac{\partial J}{\partial \beta_{i,j}} = \sum_{k=1}^{K} \left(\frac{\partial J_k}{\partial g(net_{o_k})} \cdot \frac{\partial g(net_{o_k})}{\partial net_{o_k}} \cdot \frac{\partial net_{o_k}}{\partial g\left(net_{h\,j}\right)} \cdot \frac{\partial g\left(net_{h\,j}\right)}{\partial net_{h_j}} \cdot \frac{\partial net_{h_j}}{\partial \beta_{ij}} \right)$$

- A few of these terms are independent of the objective and activation functions:

$$\frac{\partial J}{\partial \theta_{j,k}} = \frac{\partial J}{\partial g(net_{o_k})} \cdot \frac{\partial g(net_{o_k})}{\partial net_{o_k}} \cdot \boldsymbol{h_j}$$

$$\frac{\partial J}{\partial \beta_{i,j}} = \boldsymbol{x_i} \sum_{k=1}^{K} \left(\frac{\partial J_k}{\partial g(net_{o_k})} \cdot \frac{\partial g(net_{o_k})}{\partial net_{o_k}} \cdot \boldsymbol{\theta_{jk}} \right) \frac{\partial g\left(net_{h\,j}\right)}{\partial net_{h_j}}$$

BP for Logistic Activation and LLE

- As practice, let's derive the back propagation rules when we use logistic activation functions and log likelihood estimate objective function.
- Recall that for the logistic activation function $g(z) = \frac{1}{1+e^{-z}}$, its partial derivative is:

$$\frac{\partial g(z)}{\partial z} = g(z)\big(1 - g(z)\big)$$

- So

$$\frac{\partial J}{\partial \theta_{j,k}} = \frac{\partial J}{\partial g(net_{o_k})} \cdot \frac{\partial g(net_{o_k})}{\partial net_{o_k}} \cdot h_j = \frac{\partial J}{\partial g(net_{o_k})} \cdot \boldsymbol{\widehat{y_k}(1 - \widehat{y_k})} \cdot h_j$$

$$\frac{\partial J}{\partial \beta_{i,j}} = x_i \sum_{k=1}^{K} \left(\frac{\partial J_k}{\partial g(net_{o_k})} \cdot \boldsymbol{\widehat{y_k}(1 - \widehat{y_k})} \cdot \theta_{jk} \right) \boldsymbol{h_j(1 - h_j)}$$

Back Propagation Derivation

$$\frac{\partial J}{\partial \theta_{j,k}} = \frac{\partial J}{\partial g\left(net_{o_k}\right)} \cdot \widehat{y_k}\left(1 - \widehat{y_k}\right) \cdot h_j$$

$$\frac{\partial J}{\partial \beta_{i,j}} = x_i \sum_{k=1}^{K} \left(\frac{\partial J_k}{\partial g\left(net_{o_k}\right)} \cdot \widehat{y_k}\left(1 - \widehat{y_k}\right) \cdot \theta_{jk} \right) h_j\left(1 - h_j\right)$$

- And for our log likelihood function:

$$J_k = y_k \, ln\left(g\left(net_{o_k}\right)\right) + \left(1 - y_k\right) ln\left(1 - g\left(net_{o_k}\right)\right)$$

- And the partial of this with respect to $g\left(net_{o_k}\right)$ is

$$\frac{\partial J_k}{\partial g\left(net_{o_k}\right)} = \frac{y_k}{g\left(net_{o_k}\right)} - \frac{\left(1 - y_k\right)}{1 - g\left(net_{o_k}\right)} = \frac{y_k}{\hat{y}_k} - \frac{\left(1 - y_k\right)}{1 - \hat{y}_k} = \frac{\left(y_k - \hat{y}_k\right)}{\hat{y}_k\left(1 - \hat{y}_k\right)}$$

Back Propagation Derivation

$$\frac{\partial J}{\partial \theta_{j,k}} = \frac{(\boldsymbol{y_k} - \boldsymbol{\hat{y}_k})}{\boldsymbol{\hat{y}_k(1 - \hat{y}_k)}} \cdot \widehat{y_k}(1 - \widehat{y_k}) \cdot h_j$$

$$\frac{\partial J}{\partial \beta_{i,j}} = x_i \sum_{k=1}^{K} \left(\frac{(\boldsymbol{y_k} - \boldsymbol{\hat{y}_k})}{\boldsymbol{\hat{y}_k(1 - \hat{y}_k)}} \cdot \widehat{y_k}(1 - \widehat{y_k}) \cdot \theta_{jk} \right) h_j(1 - h_j)$$

- Simplifying…

$$\frac{\partial J}{\partial \theta_{j,k}} = (y_k - \hat{y}_k) h_j$$

$$\frac{\partial J}{\partial \beta_{i,j}} = x_i \sum_{k=1}^{K} \left((y_k - \hat{y}_k) \cdot \theta_{jk} \right) h_j(1 - h_j)$$

Back Propagation Derivation

$$\frac{\partial J}{\partial \theta_{j,k}} = (y_k - \hat{y}_k)h_j$$

$$\frac{\partial J}{\partial \beta_{i,j}} = x_i \sum_{k=1}^{K} \left((y_k - \hat{y}_k) \cdot \theta_{jk}\right) h_j(1 - h_j)$$

- To vectorize these we need to introduce a new operator, the **Hadamard** product.
  - This is *per-element* multiplication
  - And we'll denote it as ∘

$$\frac{\partial J}{\partial \theta} = h^T(y - \hat{y})$$

$$\frac{\partial J}{\partial \beta} = x^T\left(\left((y - \hat{y})\theta^T\right) \circ \left(h \circ (1 - h)\right)\right)$$

Back Propagation Derivation

$$\frac{\partial J}{\partial \theta} = h^T(y - \hat{y})$$

$$\frac{\partial J}{\partial \beta} = x^T\left(\left((y - \hat{y})\theta^T\right) \circ \left(h \circ (1 - h)\right)\right)$$

- For batches…

$$\frac{\partial J}{\partial \theta} = H^T\left(Y - \hat{Y}\right)$$

$$\frac{\partial J}{\partial \beta} = X^T\left(\left(\left(\left(Y - \hat{Y}\right)\right)\theta^T\right) \circ \left(H \circ (1 - H)\right)\right)$$

Performance Note

- Training can take thousands of iterations
    - Slow! ☹
- But using network after training is very fast! ☺