Multilayer Perceptron

EXAMPLE

Updating Weights in Multilayer NN

- We still do not have talked about how the weights would be updated in case of multi-layer NN!
- The weights are updated using an algorithm called **“Backpropagation”**

Updating Weights in Multilayer NN

Gradient Descent:

- Optimization algorithm that finds the set of input variables for a target function that results in a minimum value of the target function, called the minima of the function!
  - **Derivative:** Slope of curvature of a target function with respect to specific input values to the function
  - **Gradient:** Vector of partial derivates of a target function with respect to input variables
  - **Step Size:** Learning rate or alpha or eta, used to control how much to change each input variable with respect to the gradient
  - **Loss Function:** Target function to be minimized
  - **Model Parameters:** Input parameters to the loss function that are being optimized

Backpropagation (or backprop):

- Algorithm for calculating the gradient of a loss function with respect to variables of a model
- Backpropagation is an automatic differentiation algorithm that can be used to calculate the gradients for the parameters in neural network
- Gradient Descent relies on Backpropagation

Updating Weights in Multilayer NN

- **Stochastic Gradient Descent with Backpropagation:**
  - Backpropagation refers only to the method for computing the gradient, while another algorithm, such as Stochastic Gradient Descent, is used to perform learning using this gradient.

- Neural Network learns using **Stochastic Gradient Descent optimizer** and **backpropagation is used to calculate gradients as part of the optimization procedure**.
  - A **different optimizer can be used** instead of Stochastic Gradient Descent (SGD)
    - **“adam”, “nadam”, “RMSProp”, “adadelta”, …**
    - When in doubt, use “adam” or “sgd”

- A different algorithm can be used to optimize the parameters of a neural network, such as genetic algorithm (that does not require gradients)

Backprop and Gradient Descent

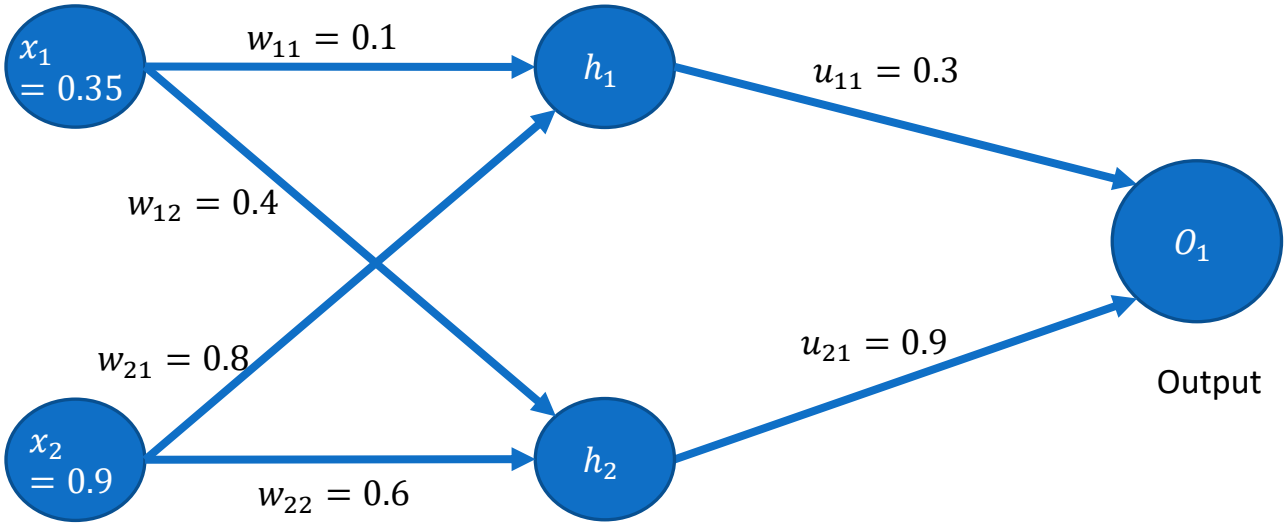
Backpropagation is the process of calculating the derivatives and gradient descent is the process of descending through the gradient, i.e. adjusting the parameters of the model to go down through the loss function.

- **Further Study:**
  - https://datascience.stackexchange.com/questions/44703/how-does-gradient-descent-and-backpropagation-work-together
  - https://machinelearningmastery.com/difference-between-backpropagation-and-stochastic-gradient-descent/

Backpropagation: Example

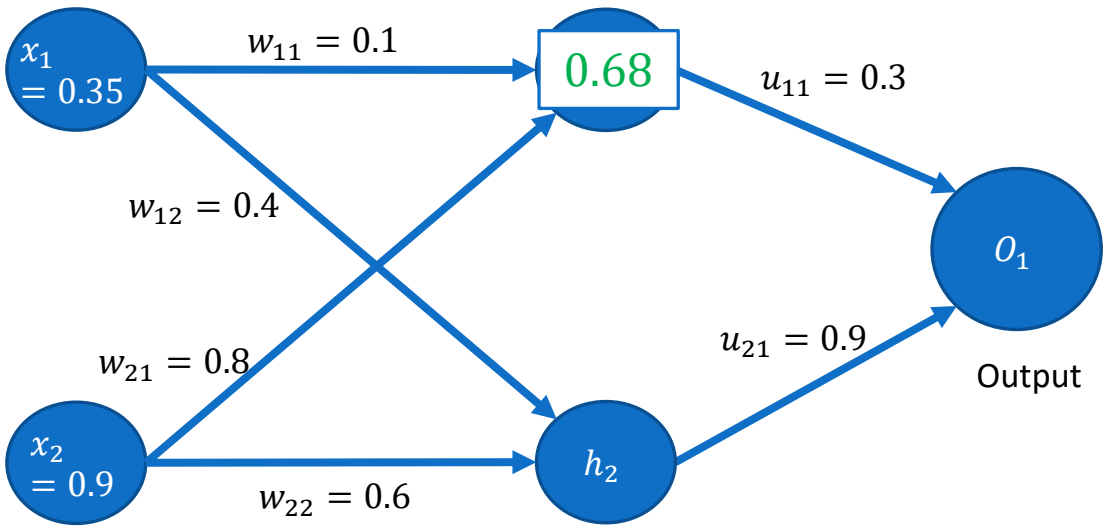
Activation Function: sigmoid **Learning Rate: 1** **Actual output: 0.5**

To Do:
- Perform Forward Pass
- Perform Backward Pass
- Perform Forward Pass



Backpropagation: Example – Forward Pass

Activation Function: sigmoid	Learning Rate: 1	Actual output: 0.5



1 – Apply Summation term:

$$z_j = \sum_j (w_{ij} \times x_i)$$

2 – Apply sigmoid activation:

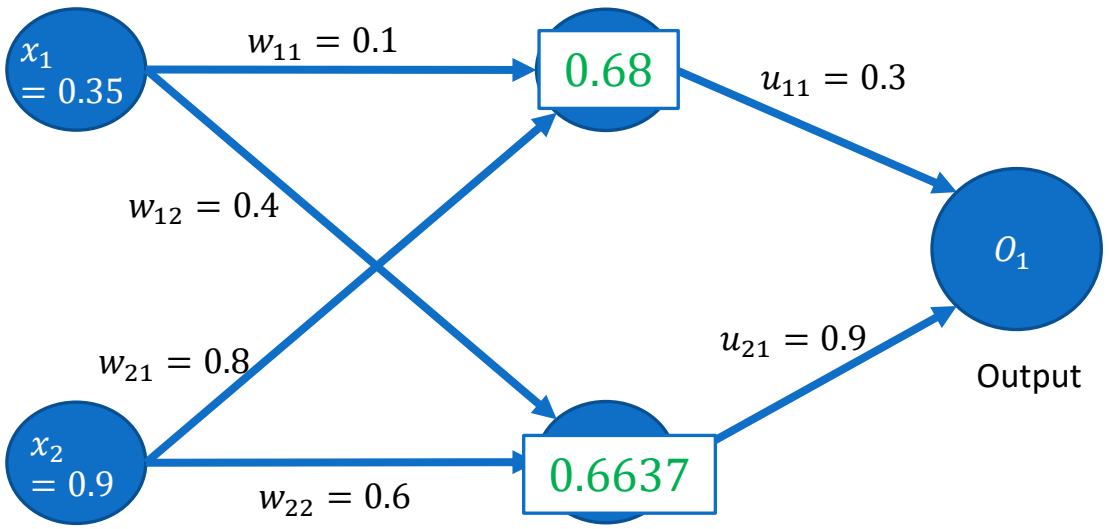
$$\sigma(z_j) = \frac{1}{1+e^{-z_j}}$$

$$z_1 = (0.1 \times 0.35) + (0.8 \times 0.9) = 0.755$$

$$h_1 = \frac{1}{(1+e^{-0.755})} = 0.68$$

Backpropagation: Example – Forward Pass

Activation Function: sigmoid	Learning Rate: 1	Actual output: 0.5



1 – Apply Summation term:

$$z_j = \sum_j (w_{ij} \times x_i)$$

2 – Apply sigmoid activation:

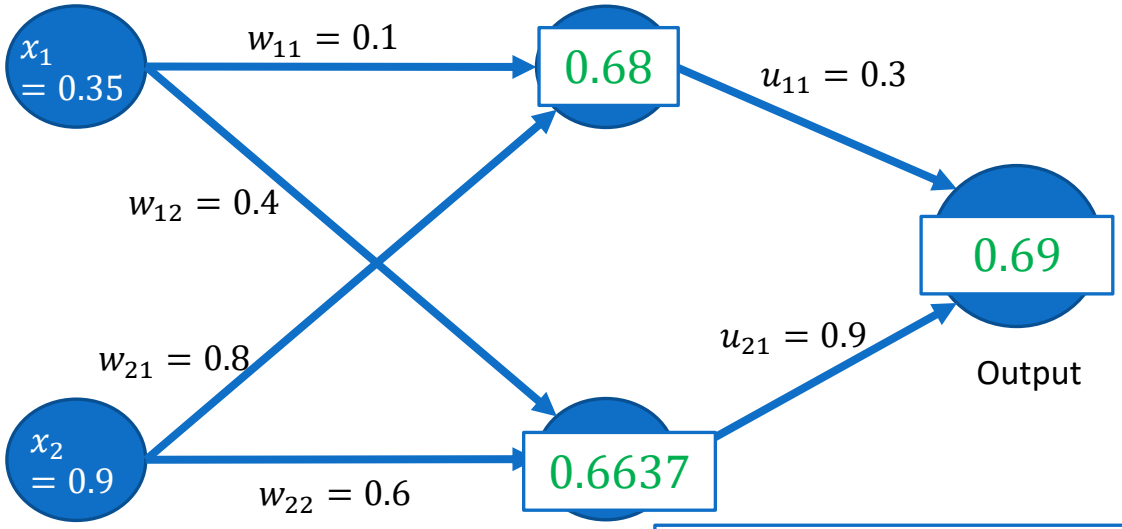
$$\sigma(z_j) = \frac{1}{1+e^{-z_j}}$$

$$z_2 = (0.4 \times 0.35) + (0.6 \times 0.9) = 0.68$$

$$h_2 = \frac{1}{(1 + e^{-0.68})} = 0.6637$$

Backpropagation: Example – Forward Pass

Activation Function: sigmoid	Learning Rate: 1	Actual output: 0.5



1 – Apply Summation term:

$$z_j = \sum_j (w_{ij} \times x_i)$$

2 – Apply sigmoid activation:

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

$$z_1 = (0.3 \times 0.68) + (0.9 \times 0.6637) = 0.801$$

$$O_1 = \frac{1}{(1+e^{-0.801})} = 0.69$$

$$Error = y - \hat{y} = 0.5 - 0.69 = -0.19$$

Backpropagation: Example – Backward Pass

- We **cannot use perceptron training rule** to update the weights if **we applied an activation function** and transformed the summation term to nonlinear value!

- **How to change the weights when the activation function is sigmoid?**

To calculate ΔW, we need to know the derivative of activation function

As we are using sigmoid activation, the derivative of sigmoid is:

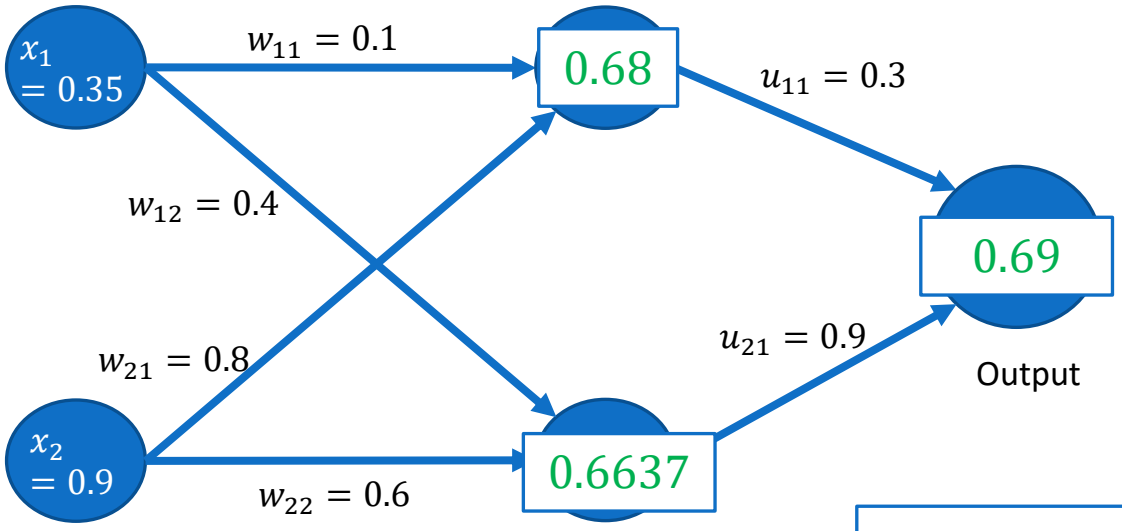
$$\sigma'(x) = \sigma(x) \times \big(1 - \sigma(x)\big)$$

Where σ is sigmoid activation function.

Details: https://hausetutorials.netlify.app/posts/2019-12-01-neural-networks-deriving-the-sigmoid-derivative/

Backpropagation: Example – Backward Pass

Activation Function: sigmoid | **Learning Rate: 1** | **Actual output: 0.5**



$$\sigma'(x) = \sigma(x) \times \big(1 - \sigma(x)\big)$$

Where σ is sigmoid activation function.

Recall: $\Delta w_i = \eta(y - \hat{y})x_i$

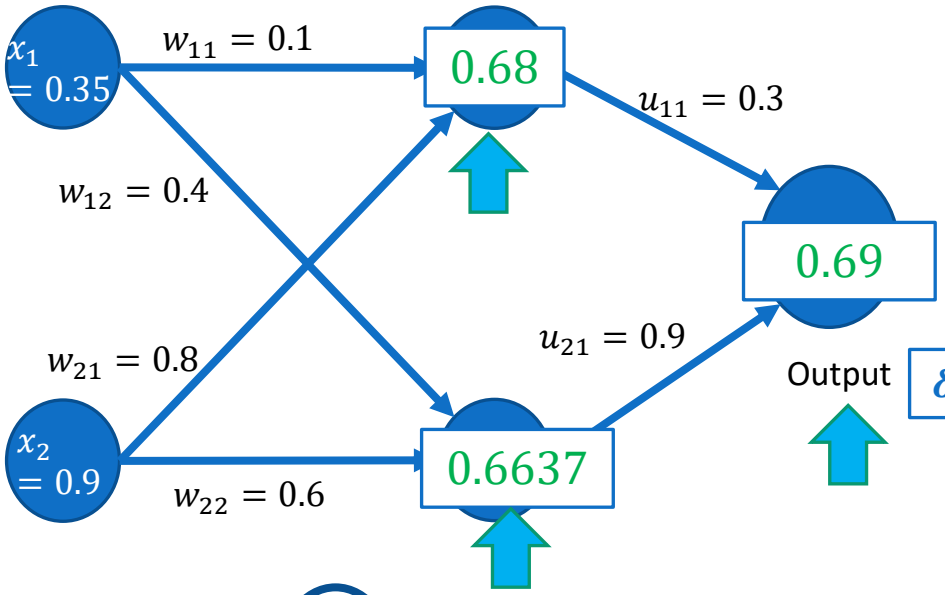
$$\Delta w_{ji} = \eta \times \delta_j \times x_i$$

Where: $\delta_j = \sigma(x) \times \big(1 - \sigma(x)\big) \times (y - \hat{y})$

So basically, now error is multiplied with derivative of activation function (sigmoid), which we call δ_j

Backpropagation: Example – Backward Pass

Activation Function: sigmoid	Learning Rate: 1	Actual output: 0.5



$$\delta O_1 = \sigma(x) \times (1 - \sigma(x)) \times (y - \hat{y})$$

$$\delta O_1 = 0.69 \times (1 - 0.69) \times (0.5 - 0.69)$$

$$\delta O_1 = -0.0406$$

$$\delta h_1 = 0.68 \times (1 - 0.68) \times (0.3 \times (-0.0406))$$

$$\delta h_1 = -0.00265$$

$$\delta h_2 = 0.6637 \times (1 - 0.6637) \times (0.9 \times (-0.0406))$$

$$\delta h_2 = -0.0082$$

$$\Delta w_{ji} = \eta \times \delta_j \times x_i$$

So with respect to h_1 or h_2, current weight is y and δ of next neuron is $\hat{y}$ but both terms are now multiplied, not subtracted.

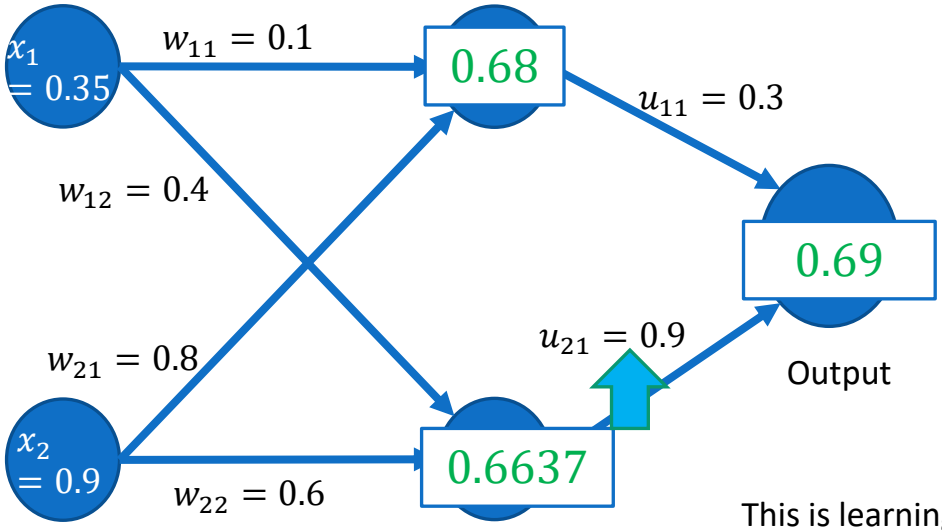
Backpropagation: Example – Backward Pass

Activation Function: sigmoid | **Learning Rate: 1** | **Actual output: 0.5**

$\delta O_1 = -0.0406$

$\delta h_1 = -0.00265$

$\delta h_2 = -0.0082$



$$\Delta w_{ji} = \eta \times \delta_j \times x_i$$

This is learning rate which dictates how much the weights should change

$$\Delta u_{21} = \eta \times \delta_j \times x_i$$

Recall: $\Delta w_i = \eta(y - \hat{y})x_i$

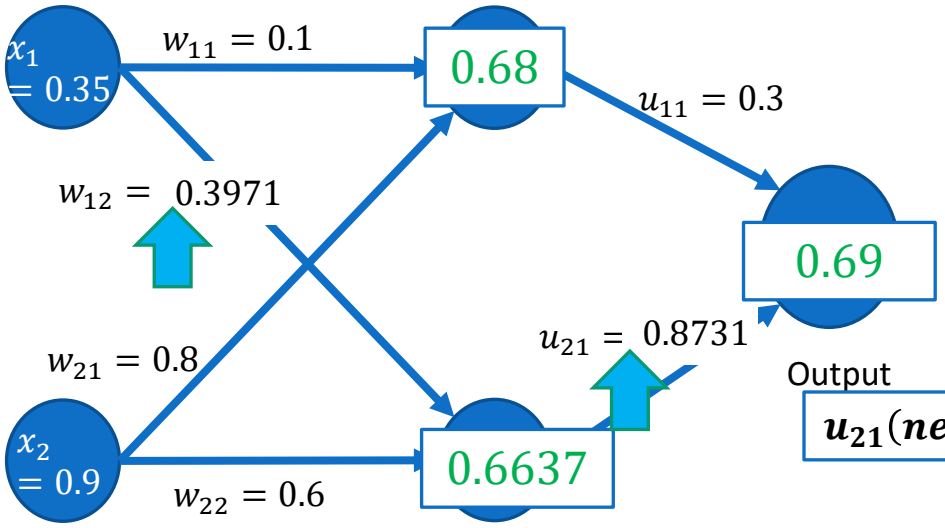
Error

Input

$$\delta_j = \sigma(x) \times \big(1 - \sigma(x)\big) \times (y - \hat{y})$$

Backpropagation: Example – Backward Pass

Activation Function: sigmoid **Learning Rate: 1** **Actual output: 0.5**



$$\delta O_1 = -0.0406$$

$$\delta h_1 = -0.00265$$

$$\delta h_2 = -0.0082$$

$$\Delta u_{21} = \eta \times \delta_j \times o_i$$

$$\Delta u_{21} = 1 \times -0.0406 \times 0.6637 = -0.0269$$

$$u_{21}(new) = \Delta u_{21} + u_{21}(old) = -0.0269 + 0.9 = 0.8731$$

$$\Delta w_{12} = \eta \times \delta_j \times x_i$$

$$\Delta w_{12} = 1 \times -0.0082 \times 0.35 = -0.00287$$

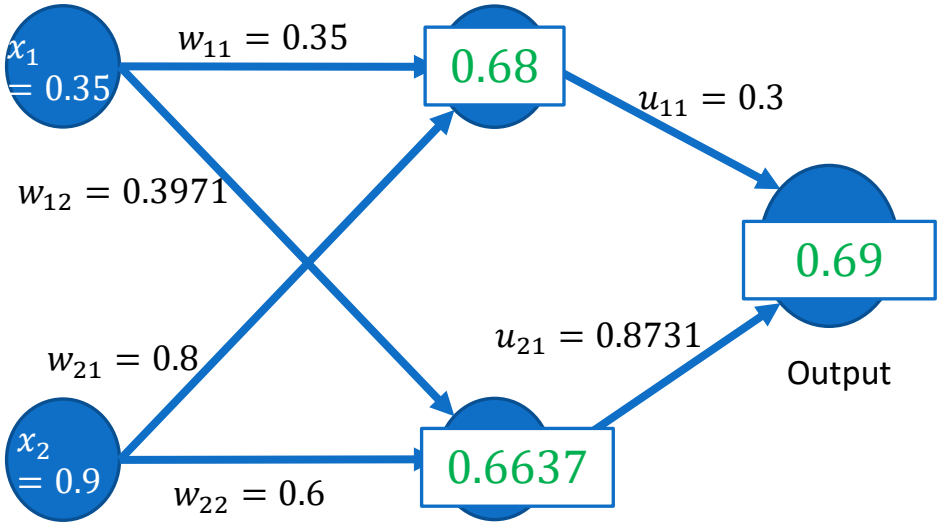
$$w_{12}(new) = \Delta w_{12} + w_{12}(old) = -0.00287 + 0.4 = 0.3971$$

$$\Delta w_{ji} = \eta \times \delta_j \times x_i$$

Homework (non-graded): Calculate other new weights…

Backpropagation: Example – All updated Weights

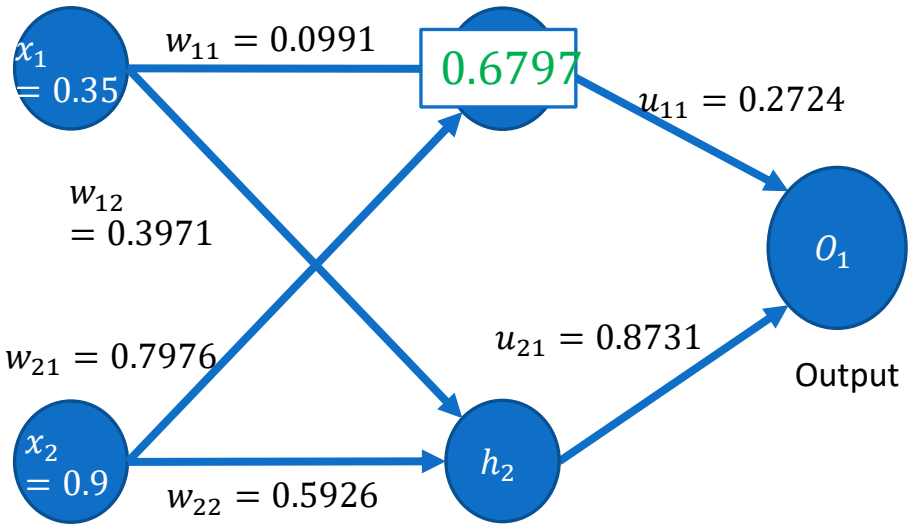
Activation Function: sigmoid	Learning Rate: 1	Actual output: 0.5



	Old weight	δ_j	x_i	Updated Weight
w_{11}	0.1	-0.00265	0.35	0.0991
w_{21}	0.8	-0.00265	0.9	0.7976
w_{12}	0.4	-0.0082	0.35	0.3971
w_{22}	0.6	-0.0082	0.9	0.5926
u_{11}	0.3	-0.0406	0.68	0.2724
u_{21}	0.9	-0.0406	0.6637	0.8731

Backpropagation: Example – Another Forward Pass

Activation Function: sigmoid	Learning Rate: 1	Actual output: 0.5



1 – Apply Summation term:

$$z_j = \sum_j (w_{ij} \times x_i)$$

2 – Apply sigmoid activation:

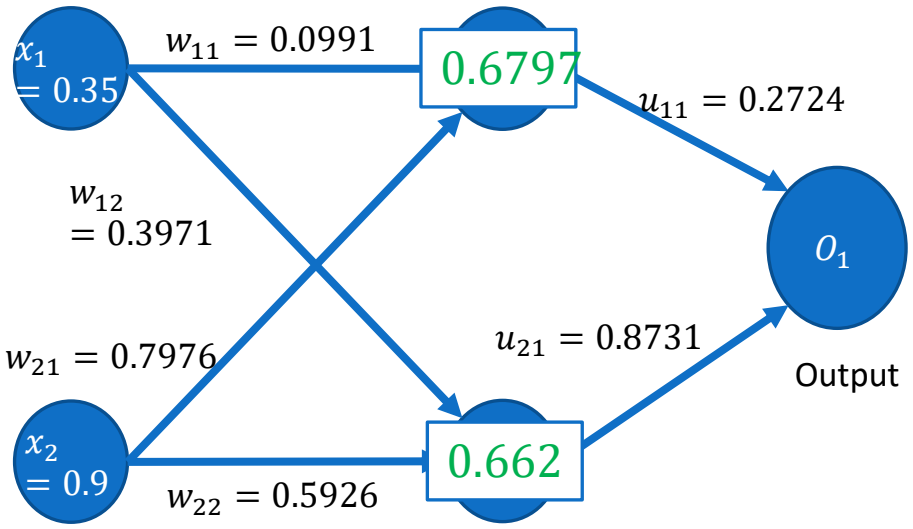
$$\sigma(z_j) = \frac{1}{1+e^{-z_j}}$$

$$z_1 = (0.0991 \times 0.35) + (0.7976 \times 0.9) = 0.7525$$

$$h_1 = \frac{1}{(1 + e^{-0.7525})} = 0.6797$$

Backpropagation: Example – Another Forward Pass

Activation Function: sigmoid	Learning Rate: 1	Actual output: 0.5



1 – Apply Summation term:

$$z_j = \sum_j (w_{ij} \times x_i)$$

2 – Apply sigmoid activation:

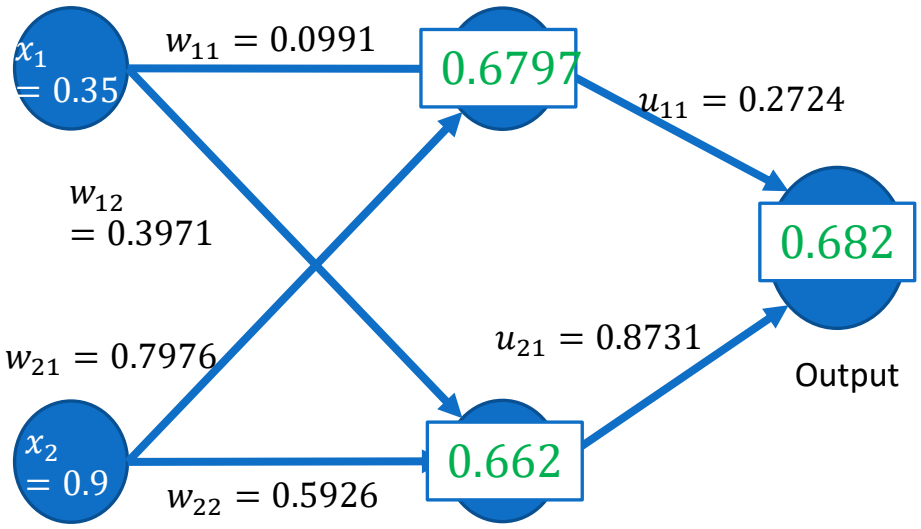
$$\sigma(z_j) = \frac{1}{1+e^{-z_j}}$$

$$z_2 = (0.3971 \times 0.35) + (0.5926 \times 0.9) = 0.6723$$

$$h_2 = \frac{1}{(1 + e^{-0.6723})} = 0.662$$

Backpropagation: Example – Another Forward Pass

Activation Function: sigmoid	Learning Rate: 1	Actual output: 0.5



1 – Apply Summation term:

$$z_j = \sum_j (w_{ij} \times x_i)$$

2 – Apply sigmoid activation:

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

$$z_1 = (0.2724 \times 0.6797) + (0.8731 \times 0.662) = 0.7631$$

$$O_1 = \frac{1}{(1 + e^{-0.7631})} = 0.682$$

$Previous\ Error = -0.19$

$Error = 0.5 - 0.682 = -0.182$

Repeat till error no longer reduces….

Summary of Equations

- We want to compute new weights:

$$\Delta w_{ji} = \eta \times \delta \times x_i$$

  - Where j is the next neuron
  - i is current neuron

- If i is the output unit:

$$\delta = \sigma(x) \times \left(1 - \sigma(x)\right) \times (y - \hat{y})$$

- If i is hidden unit

$$\delta = \sigma(x) \times \left(1 - \sigma(x)\right) \times (w_{ji} \times \delta_j)$$

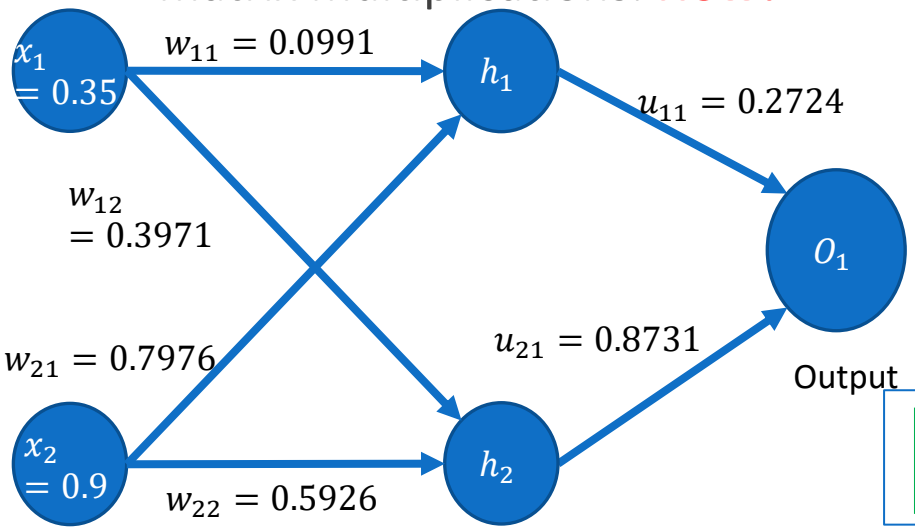
Summary

- Backpropagation algorithm is able to find optimal weights
- Weights are changed with respect to error
- It also works for multiple layers (theoretically infinite layers)

From now on, we will talk in terms of layers and architectures, without worrying about how in this architecture the weights will change!!

In class activity…

- The output of the above model in forward pass is nothing but two matrix multiplications. **How?**



$$W = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix}, X = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$h(X) = W^T X$$

$$\begin{bmatrix} w_{11}, w_{21} \\ w_{12}, w_{22} \end{bmatrix} \times \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = [\,]$$

$$\begin{bmatrix} 0.0991, 0.7976 \\ 0.3971, 0.5926 \end{bmatrix} \times \begin{bmatrix} 0.35 \\ 0.9 \end{bmatrix} = \begin{bmatrix} 0.7525 \\ 0.6723 \end{bmatrix}$$

$$\begin{bmatrix} u_{11} \\ u_{21} \end{bmatrix} \times \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} = [\,]$$

$$h(X) = U^T X$$

$$[u_{11}, u_{21}] \times \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} = [\,]$$

Note: Apply activation function after matrix multiplication to get h_1 and h_2

$$[0.2724, 0.8731] \times \begin{bmatrix} 0.6797 \\ 0.662 \end{bmatrix} = [0.7631]$$

Book Reading

- Murphy – Chapter 8
- Jurafsky – Chapter 5, Chapter 4, Chapter 7
- Tom Mitchel – Chapter 4