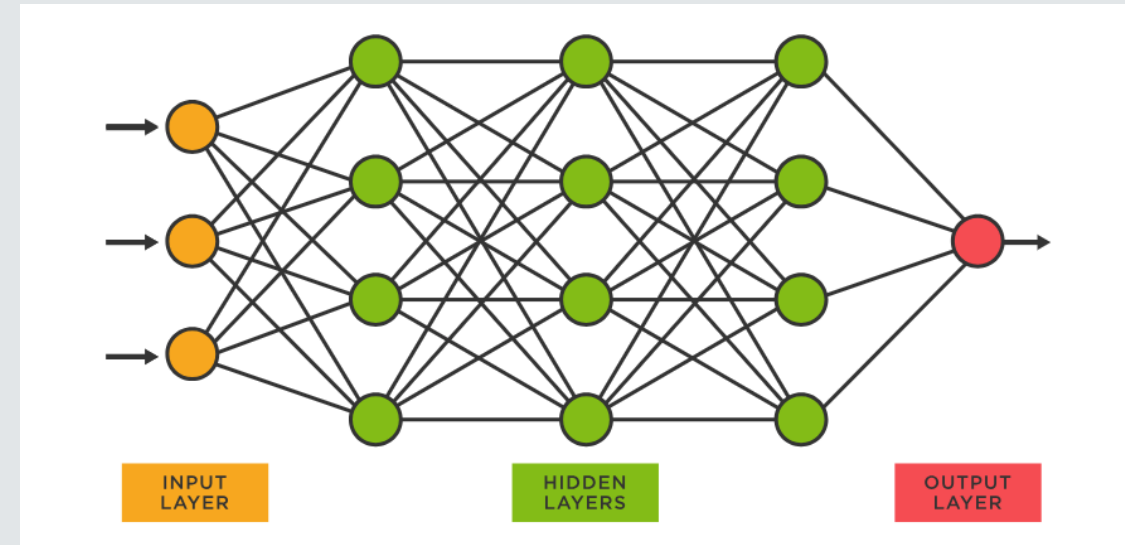


Lecture 9 (FNN)

Feedforward Neural Network

- Neural networks learn a mapping function $f: X \rightarrow Y$ from input features $X \in \mathbb{R}^n$ to output labels Y , where:
 - Classification: Y is discrete (e.g., $Y \in \{1, 2, \dots, k\}$).
 - Regression: Y is continuous (e.g., $Y \in \mathbb{R}$).
- Key components of a neural network:
 - Layers (Depth): define the number of layers. This includes the input, hidden, and output layer.
 - Width (Number of neurons per layer): define the number of neurons per layer.
 - Activation function: This will add a layer of non-linearity to allow to model complex relationships
 - Loss function: Measures the difference between the predicted output and the actual target. (MSE [Regression], Cross-entropy [Classification])



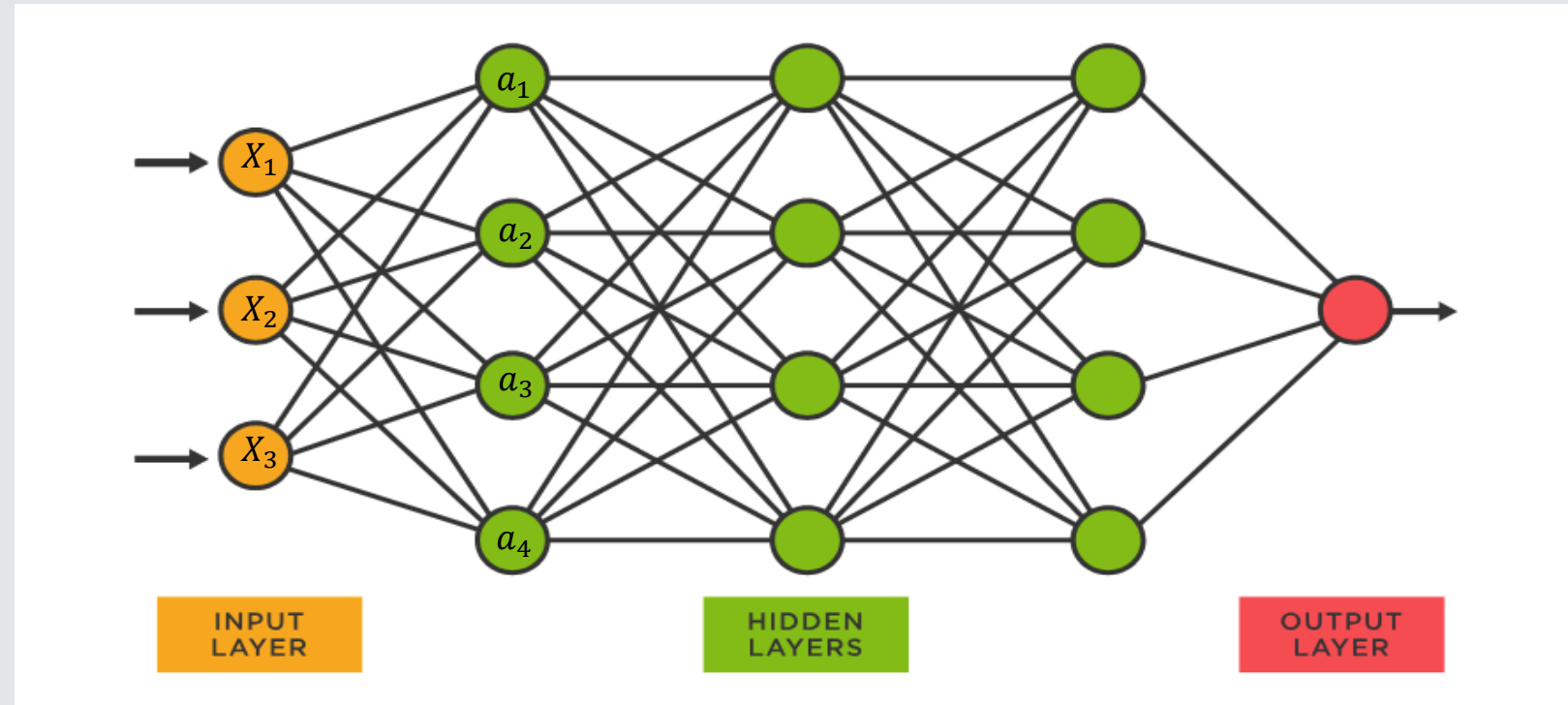
<https://www.spotfire.com/glossary/what-is-a-neural-network>

Feedforward Neural Network

- $a_1 = f(W_1X + b_1)$

where $W_1 \in \mathbb{R}^{1 \times 3}$, $X = \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix}$, $b_1 \in \mathbb{R}$

f is the activation function which could be a linear or a non-linear activation function. The non-linearity will give us the flexibility of modelling non-linear problems.



<https://www.spotfire.com/glossary/what-is-a-neural-network>

Feedforward Neural Network – Forward Propagation (Classification)

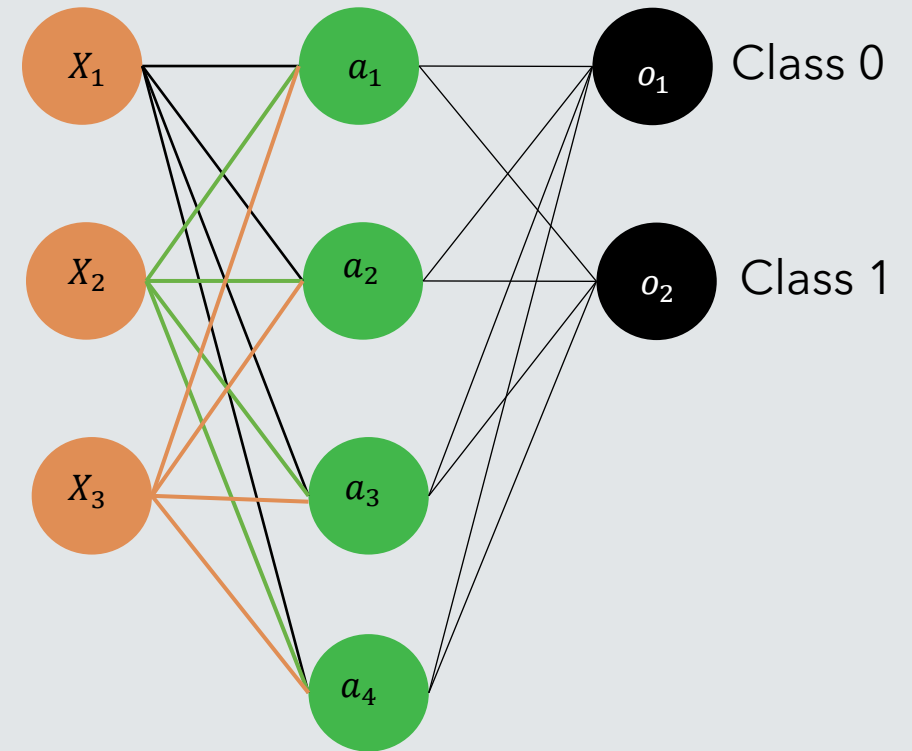
- $a_1 = f(W_1^{(1)}X + b_1) = (W_1^{(1)}X + b_1^{(1)})$; f is a linear activation function.
where

$$W_1^{(1)} = [W_{11} \quad W_{21} \quad W_{31}] \in \mathbb{R}^{1 \times 3}$$

$W_{ij}^{(k)}$ (i is the starting node and j is the ending node, and k is the layer)

$$X = \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} \in \mathbb{R}^{3 \times 1}; b_1^{(1)} \in \mathbb{R}$$

- $o_1 = f(W_1^{(2)}a + b_1^{(2)}) = f(\hat{o}_1) = \frac{e^{\hat{o}_1}}{e^{\hat{o}_1} + e^{\hat{o}_2}}$; f is a softmax function
- $o_2 = f(W_2^{(2)}a + b_2^{(2)}) = f(\hat{o}_2) = \frac{e^{\hat{o}_2}}{e^{\hat{o}_1} + e^{\hat{o}_2}}$; f is a softmax function
- $l = -(y \log(o_2) + (1 - y) \log(o_1))$; where y is the true label and \hat{y} is the predicated label



Feedforward Neural Network – Forward Propagation (Classification)

- $a_1 = f(W_1^{(1)}X + b_1) = (W_1^{(1)}X + b_1^{(1)})$; f is a linear activation function.

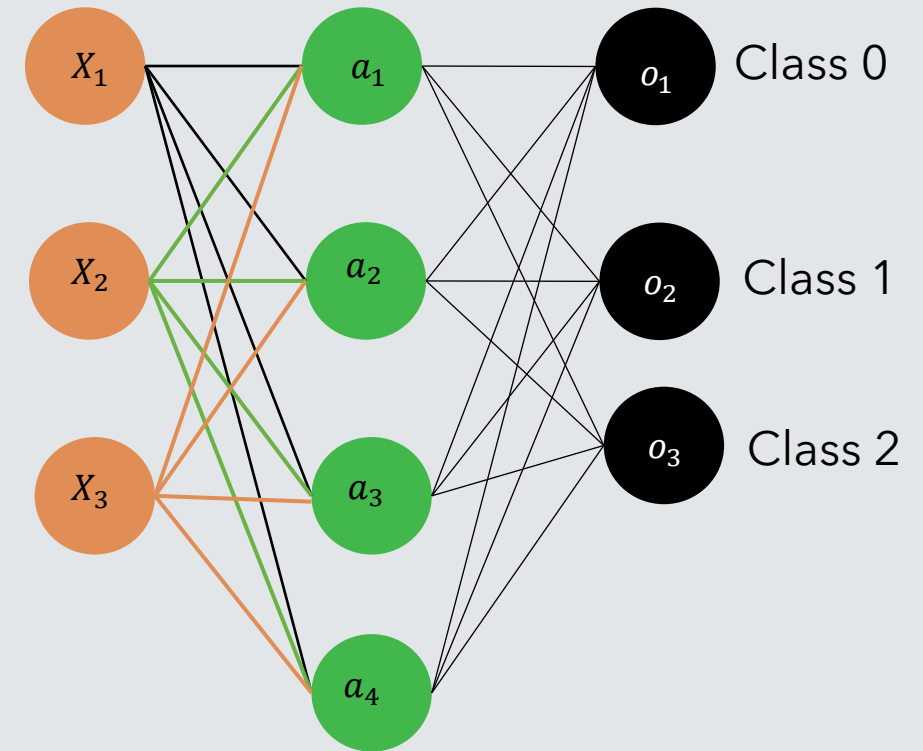
where

$$W_1^{(1)} = [W_{11} \quad W_{21} \quad W_{31}] \in \mathbb{R}^{1 \times 3}$$

$W_{ij}^{(k)}$ (i is the starting node and j is the ending node, and k is the layer)

$$X = \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} \in \mathbb{R}^{3 \times 1}; b_1^{(1)} \in \mathbb{R}$$

- $o_1 = f(W_1^{(2)}a + b_1^{(2)}) = f(\hat{o}_1) = \frac{e^{\hat{o}_1}}{e^{\hat{o}_1} + e^{\hat{o}_2} + e^{\hat{o}_3}}$; f is a softmax function
- $o_2 = f(W_2^{(2)}a + b_2^{(2)}) = f(\hat{o}_2) = \frac{e^{\hat{o}_2}}{e^{\hat{o}_1} + e^{\hat{o}_2} + e^{\hat{o}_3}}$; f is a softmax function
- $o_3 = f(W_3^{(2)}a + b_3^{(2)}) = f(\hat{o}_3) = \frac{e^{\hat{o}_3}}{e^{\hat{o}_1} + e^{\hat{o}_2} + e^{\hat{o}_3}}$; f is a softmax function
- $l = -\sum_{i=1}^3 p_i \log(o_i)$; where p_i is the true distribution and o_i is the predicated distribution



Feedforward Neural Network – Forward Propagation (Regression)

- $a_1 = f(W_1^{(1)}X + b_1) = (W_1^{(1)}X + b_1^{(1)})$; f is a linear activation function.

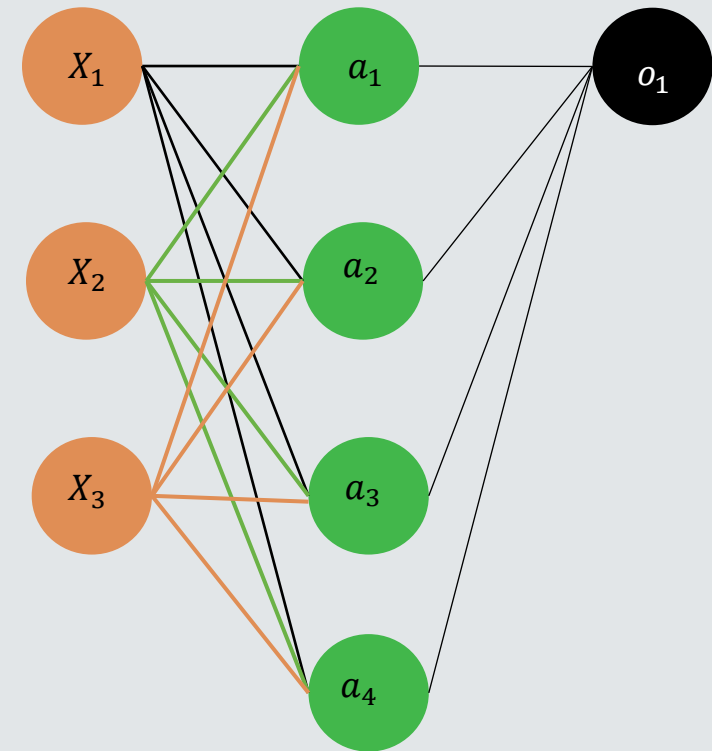
where

$$W_1^{(1)} = [W_{11} \quad W_{21} \quad W_{31}] \in \mathbb{R}^{1 \times 3}$$

$W_{ij}^{(k)}$ (i is the starting node and j is the ending node, and k is the layer)

$$X = \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} \in \mathbb{R}^{3 \times 1}; b_1^{(1)} \in \mathbb{R}$$

- $o_1 = f(W_1^{(2)}a + b_1^{(2)}) = W_1^{(2)}a + b_1^{(2)}$; f is a linear function
- $l = (y - o_1)^2$ (SE); where y is the true value and o_1 is the predicated value



Feedforward Neural Network – Forward Propagation (Regression)

- $a_1 = f(W_1^{(1)}X + b_1) = (W_1^{(1)}X + b_1^{(1)})$; f is a linear activation function.

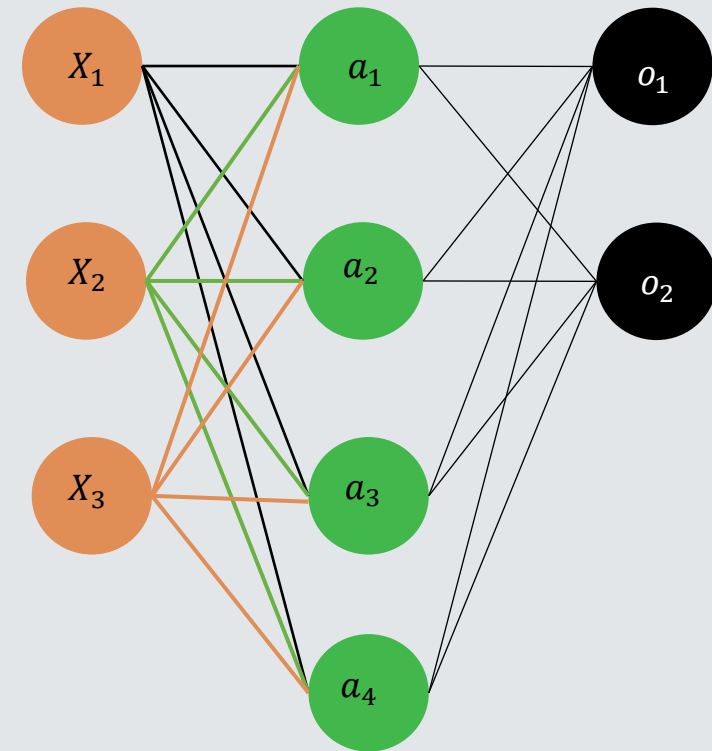
where

$$W_1^{(1)} = [W_{11} \quad W_{21} \quad W_{31}] \in \mathbb{R}^{1 \times 3}$$

$W_{ij}^{(k)}$ (i is the starting node and j is the ending node, and k is the layer)

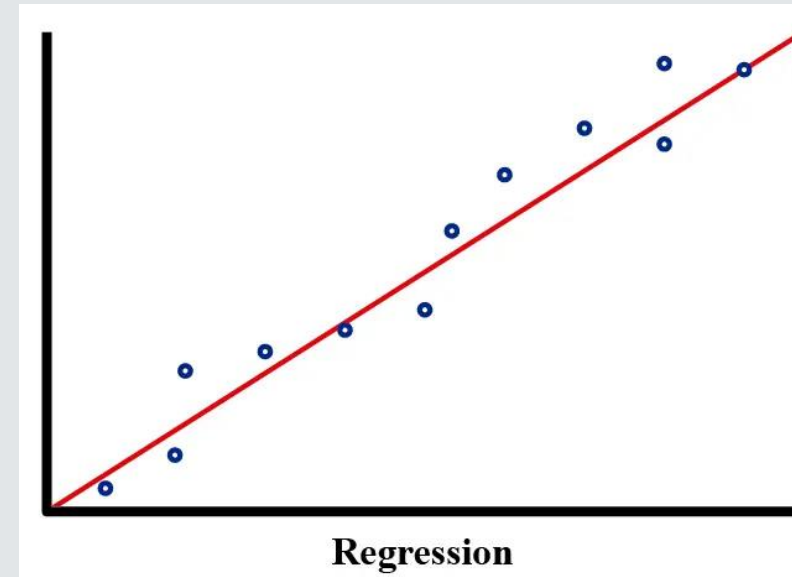
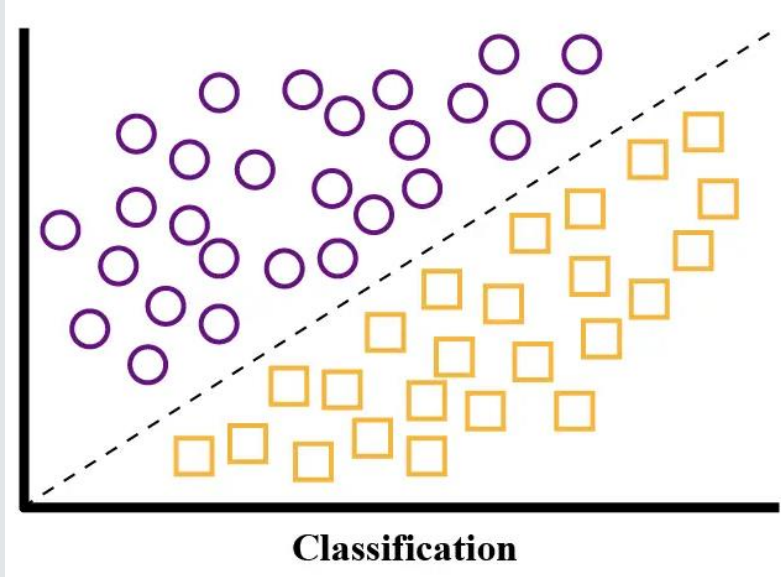
$$X = \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} \in \mathbb{R}^{3 \times 1}; b_1^{(1)} \in \mathbb{R}$$

- $o_1 = f(W_1^{(2)}a + b_1^{(2)}) = W_1^{(2)}a + b_1^{(2)}$; f is a linear function
- $o_2 = f(W_2^{(2)}a + b_2^{(2)}) = W_2^{(2)}a + b_2^{(2)}$; f is a linear function
- $l = \frac{1}{2}[(y_1 - o_1)^2 + (y_2 - o_2)^2]$ (MSE); where y_1, y_2 is the true value and o_1, o_2 is the predicted value

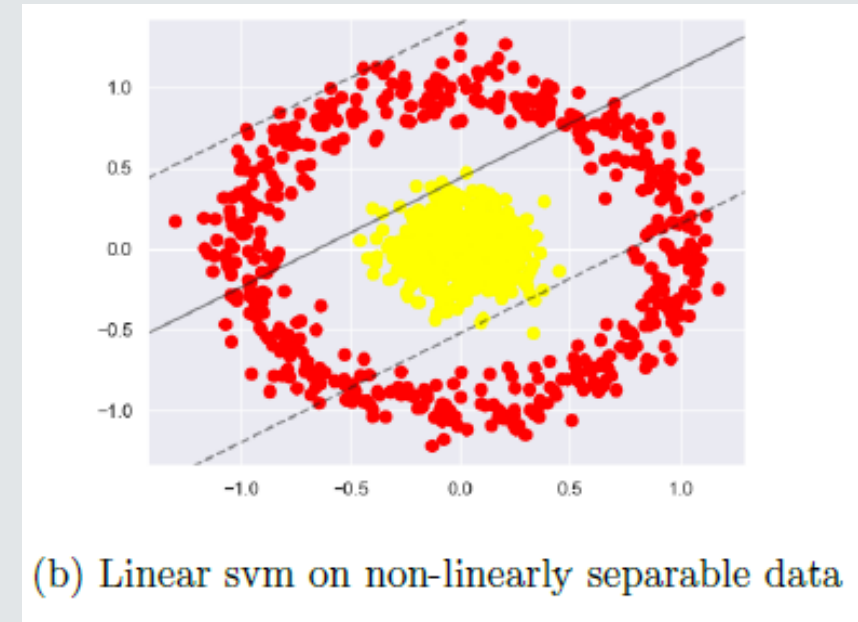
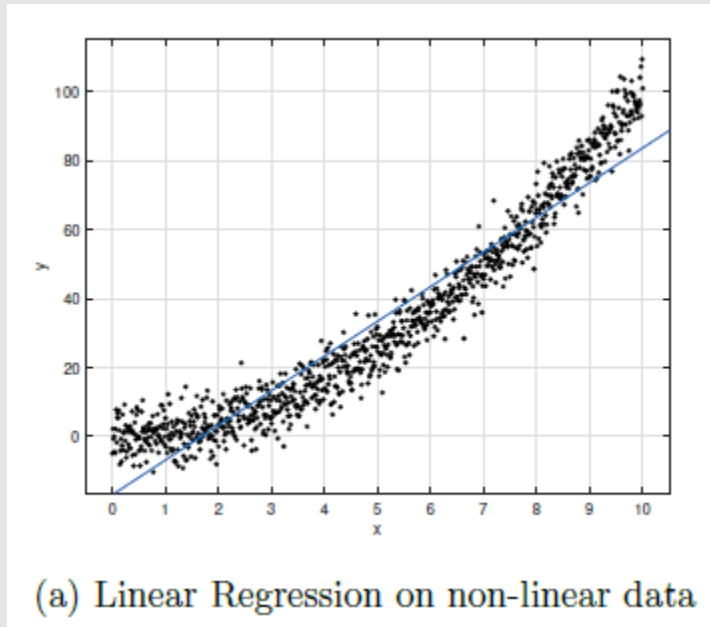


Feedforward Neural Network

- These linear activation functions will allow us to model and capture linear relationships as shown below but it fails in the case of non-linear relationships.



Feedforward Neural Network



Therefore, we need non-linear activation functions

Feedforward Neural Network – Activation Functions

- Activation functions introduce non-linearity into neural networks, enabling them to approximate complex functions. Each function has unique characteristics that impact the network's performance, convergence, and generalization. Below are common activation functions used in neural networks.

Feedforward Neural Network – Activation Functions

- Sigmoid function

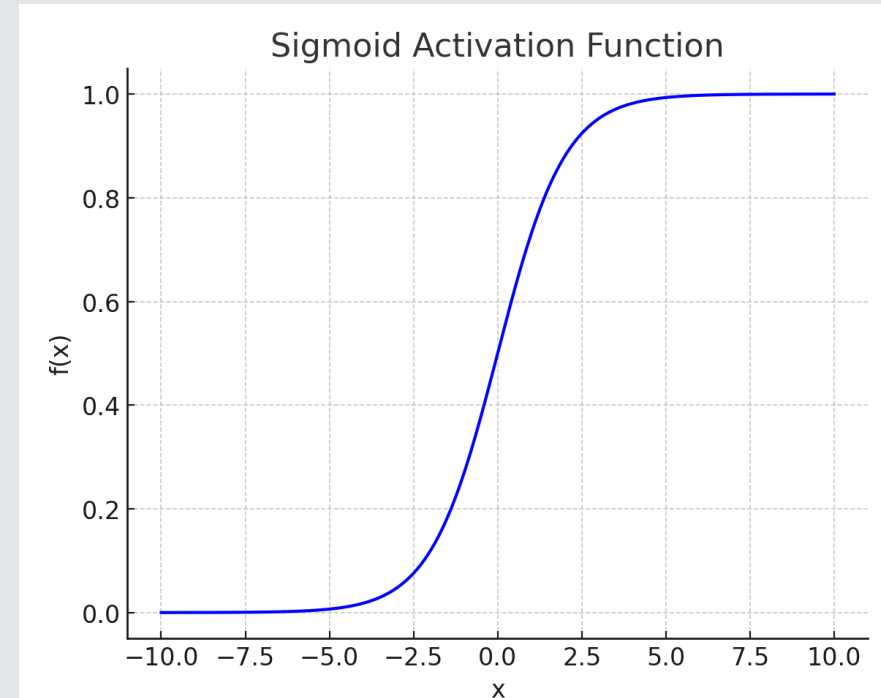
$$f(x) = \frac{1}{1 + e^{-x}}$$

- **Advantages:**

- *Smooth gradient, useful for binary classification.*
- *Output values between 0 and 1, which can represent probabilities.*

- **Disadvantages:**

- *Vanishing gradient problem for large or small inputs, slowing down training.*



Feedforward Neural Network – Activation Functions

- Tanh function

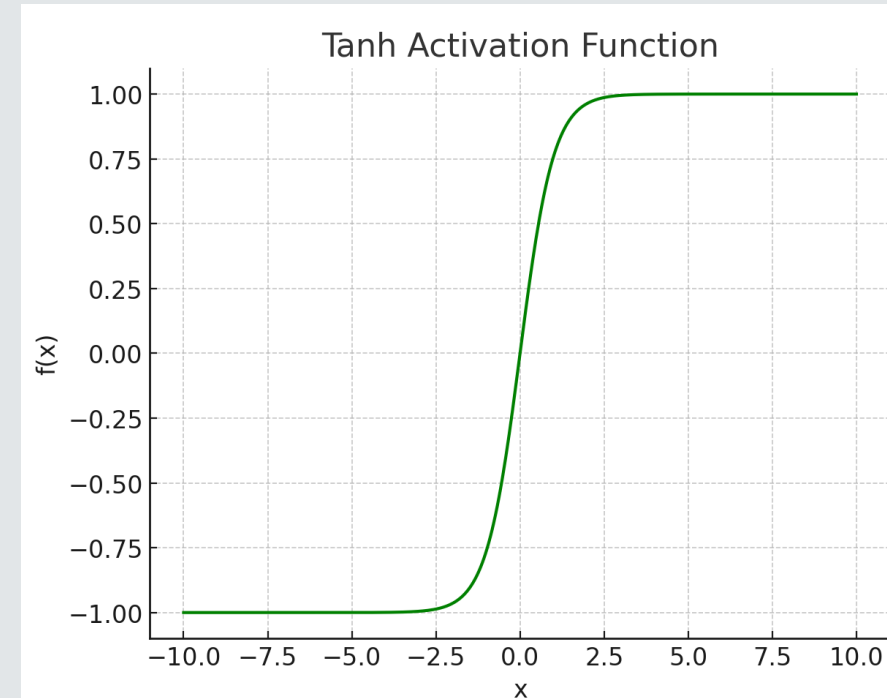
$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

- **Advantages:**

- *Zero-centered output, which helps faster convergence*
- *Larger gradients compared to sigmoid, reducing vanishing gradient*

- **Disadvantages:**

- *Still suffers from the vanishing gradient problem, especially in deep networks.*



Feedforward Neural Network – Activation Functions

- Relu function

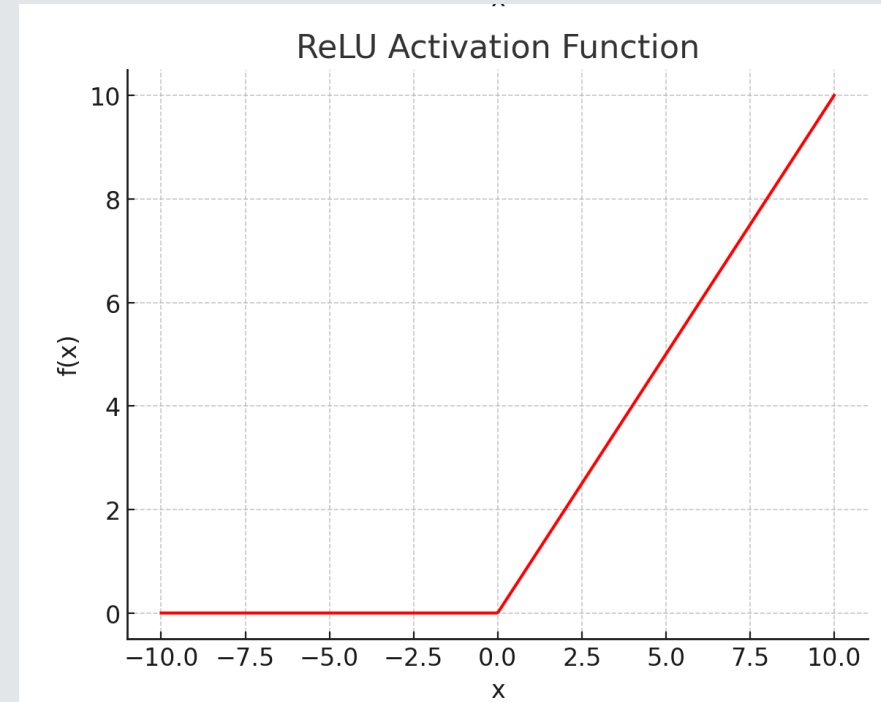
$$f(x) = \max(0, x)$$

- **Advantages:**

- Solves vanishing gradient problem for positive values

- **Disadvantages:**

- *Dead neurons issue, where certain neurons always output zero and stop learning.*
- *Can cause gradient explosion in certain cases.*



Feedforward Neural Network – Activation Functions

- Add something about 0/1 function and the difficulty of optimization

Feedforward Neural Network – Forward Propagation (Classification)

$$\bullet \ a_1 = f(W_1^{(1)}X + b_1) = \begin{cases} \max(0, W_1^{(1)}X + b_1); f \text{ is a relu function} \\ \frac{1}{1+e^{-W_1^{(1)}X+b_1}}; f \text{ is a sigmoid function} \\ \frac{e^{W_1^{(1)}X+b_1}-e^{-W_1^{(1)}X+b_1}}{e^{W_1^{(1)}X+b_1}+e^{-W_1^{(1)}X+b_1}}; f \text{ is a tanh function} \end{cases}$$

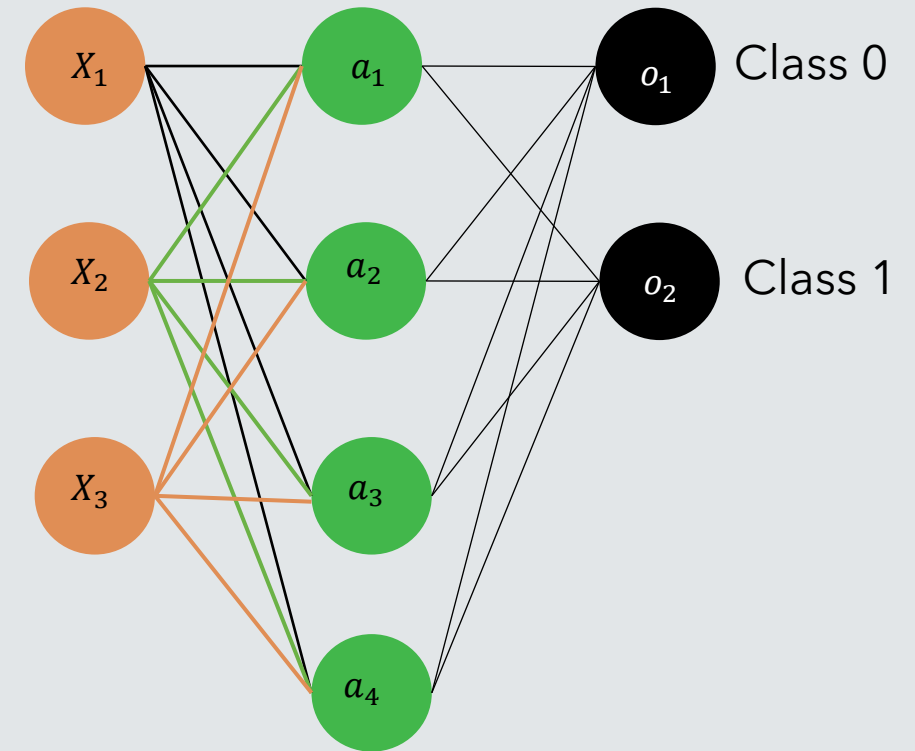
where

$$W_1^{(1)} = [W_{11} \ W_{21} \ W_{31}] \in \mathbb{R}^{1 \times 3}$$

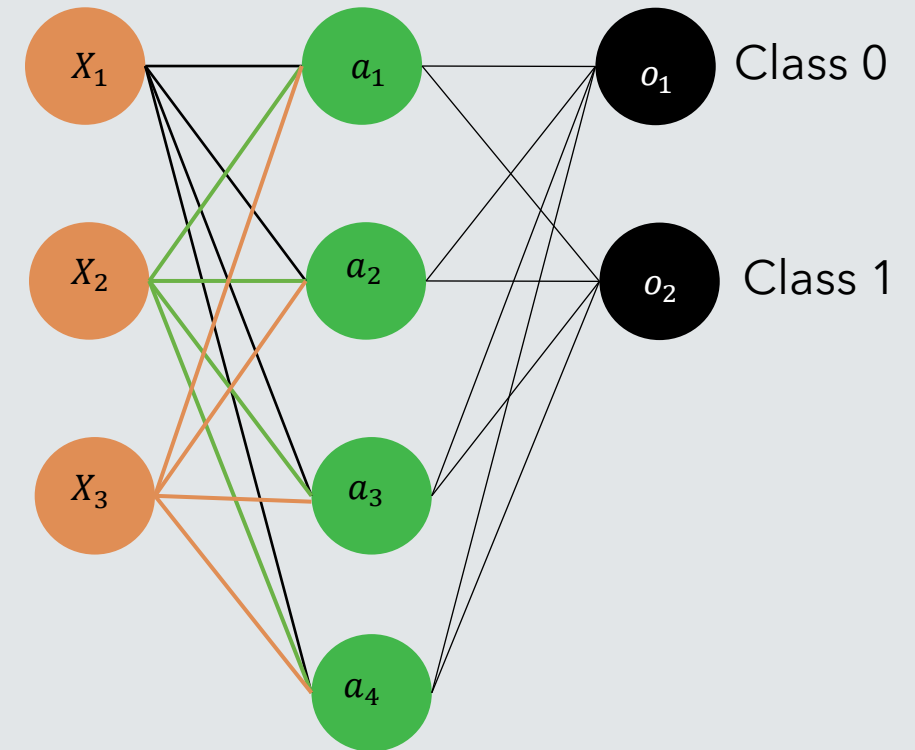
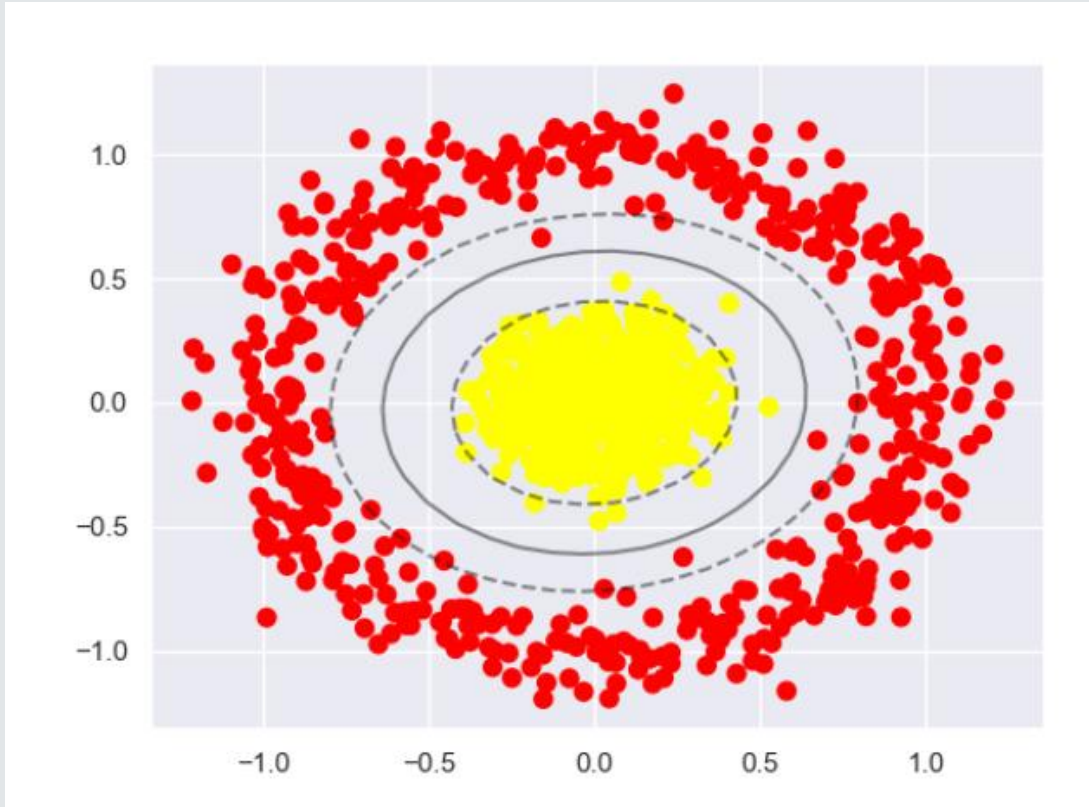
$W_{ij}^{(k)}$ (i is the starting node and j is the ending node, and k is the layer)

$$X = \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} \in \mathbb{R}^{3 \times 1}; b_1^{(1)} \in \mathbb{R}$$

- $o_1 = f(W_1^{(2)}a + b_1^{(2)}) = f(\hat{o}_1) = \frac{e^{\hat{o}_1}}{e^{\hat{o}_1} + e^{\hat{o}_2}}; f \text{ is a softmax function}$
- $o_2 = f(W_2^{(2)}a + b_2^{(2)}) = f(\hat{o}_2) = \frac{e^{\hat{o}_2}}{e^{\hat{o}_1} + e^{\hat{o}_2}}; f \text{ is a softmax function}$
- $l = -(y \log(o_2) + (1 - y) \log(o_1));$ where y is the true label and \hat{y} is the predicated label



Feedforward Neural Network – Forward Propagation (Classification)



Feedforward Neural Network

Wait!

How do we calculate all of the weights and biases of the neural network?

Feedforward Neural Network – Backpropagation (Regression)

- $a_1 = (W_1^{(1)}X + b_1^{(1)})$

$$W_1^{(1)} = \begin{bmatrix} W_{11}^{(1)} & W_{21}^{(1)} & W_{31}^{(1)} \end{bmatrix} \in \mathbb{R}^{1 \times 3}$$

$$X = \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} \in \mathbb{R}^{3 \times 1}; b_1^{(1)} \in \mathbb{R}$$

- $o_1 = W_1^{(2)}a + b_1^{(2)}; o_2 = W_2^{(2)}a + b_2^{(2)}$

- $l = \frac{1}{2}[(y_1 - o_1)^2 + (y_2 - o_2)^2]$

- How to update the weights and biases?

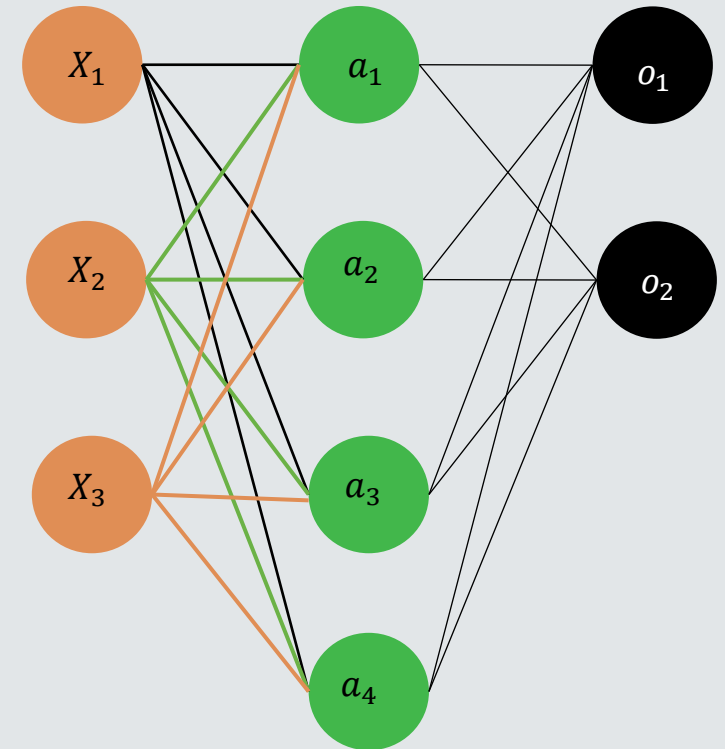
$$\min_{W_{11}^{(1)}, W_{21}^{(1)}, W_{31}^{(1)}, W_{12}^{(1)}, W_{22}^{(1)}, W_{32}^{(1)}, W_{13}^{(1)}, W_{23}^{(1)}, W_{33}^{(1)}, W_{11}^{(2)}, W_{21}^{(2)}, \dots} l$$

- There are several optimizer that will be used:

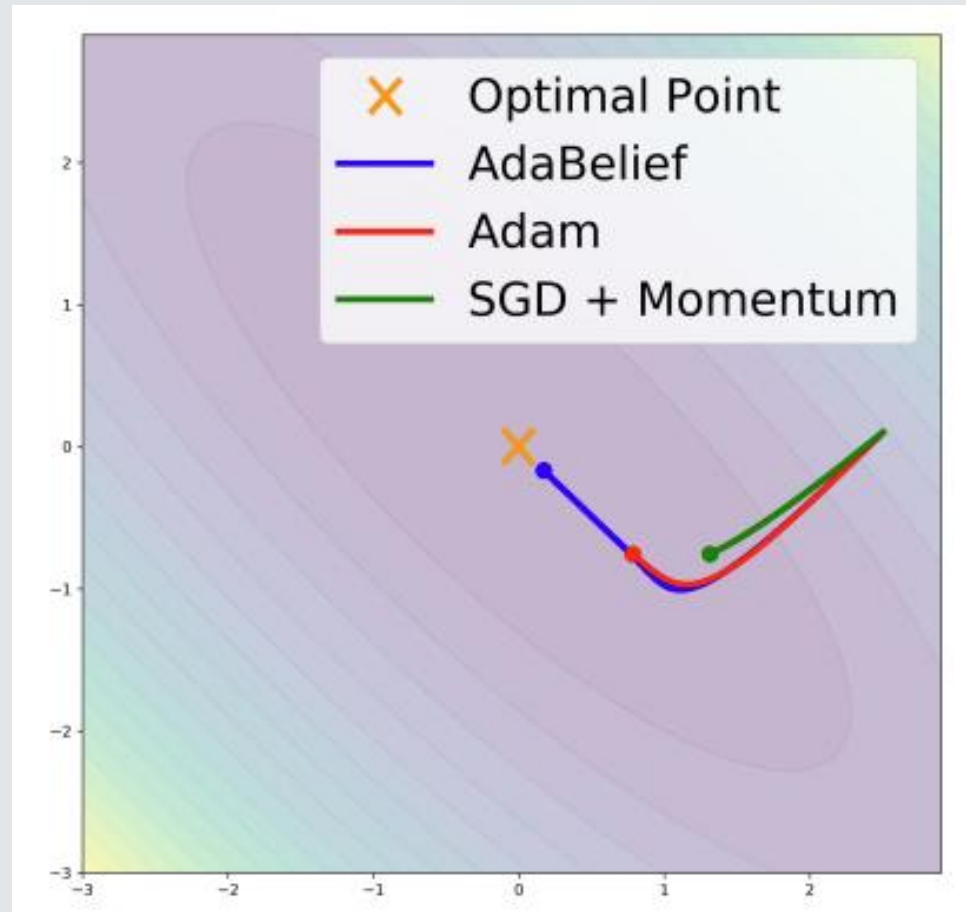
- Stochastic Gradient Descent

- SGD with momentum

- Adam



Feedforward Neural Network – Backpropagation (Regression)



<https://arxiv.org/pdf/2010.07468>

Feedforward Neural Network – Backpropagation (Regression)

- $a_1 = (W_1^{(1)}X + b_1^{(1)})$

$$W_1^{(1)} = \begin{bmatrix} W_{11}^{(1)} & W_{21}^{(1)} & W_{31}^{(1)} \end{bmatrix} \in \mathbb{R}^{1 \times 3}$$

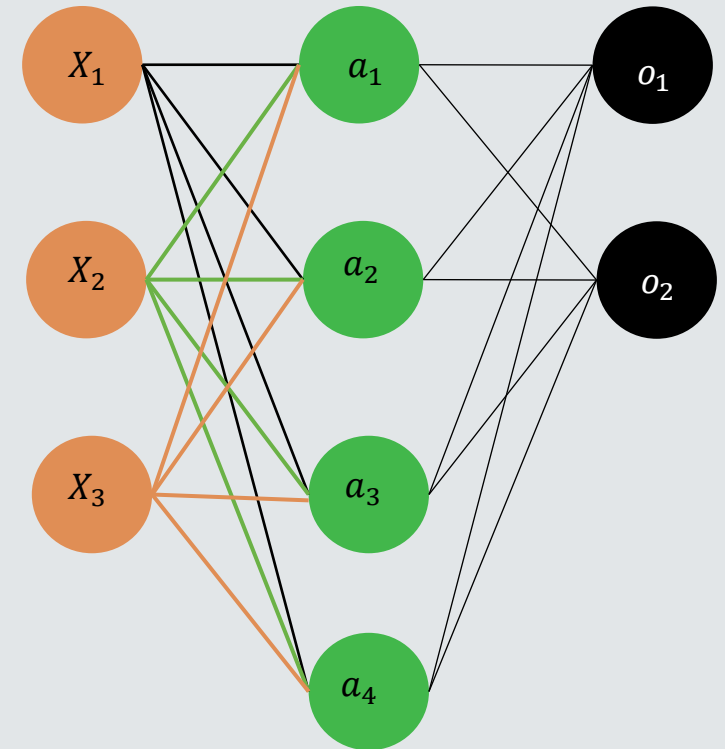
$$X = \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} \in \mathbb{R}^{3 \times 1}; b_1^{(1)} \in \mathbb{R}$$

- $o_1 = W_1^{(2)}a + b_1^{(2)}; o_2 = W_2^{(2)}a + b_2^{(2)}$

- $l = \frac{1}{2}[(y_1 - o_1)^2 + (y_2 - o_2)^2]$

- $$\begin{aligned} W_{11}^{(2)} &= W_{11}^{(2)} - \eta \frac{\partial l}{\partial W_{11}^{(2)}} = W_{11}^{(2)} - \eta \frac{\partial o_1}{\partial W_{11}^{(2)}} \frac{\partial l}{\partial o_1} \\ &= W_{11}^{(2)} - \eta \frac{\partial o_1}{\partial W_{11}^{(2)}} \frac{\partial}{\partial o_1} \left(\frac{1}{2} [(y_1 - o_1)^2 + (y_2 - o_2)^2] \right) \\ &= W_{11}^{(2)} - \eta \frac{\partial}{\partial W_{11}^{(2)}} \left[W_{11}^{(2)}a_1 + W_{21}^{(2)}a_2 + W_{31}^{(2)}a_3 + W_{41}^{(2)}a_4 \right] (-(y_1 - o_1)) = W_{11}^{(2)} - \eta a_1 \end{aligned}$$

The same procedure will be done for all of the weights and biases in the neural network.



Feedforward Neural Network – Forward Propagation (Regression)

$$\bullet \quad a_1 = f(W_1^{(1)}X + b_1) = \begin{cases} \max(0, W_1^{(1)}X + b_1); f \text{ is a relu function} \\ \frac{1}{1+e^{-W_1^{(1)}X + b_1}}; f \text{ is a sigmoid function} \\ \frac{e^{W_1^{(1)}X + b_1} - e^{-W_1^{(1)}X + b_1}}{e^{W_1^{(1)}X + b_1} + e^{-W_1^{(1)}X + b_1}}; f \text{ is a tanh function} \end{cases}$$

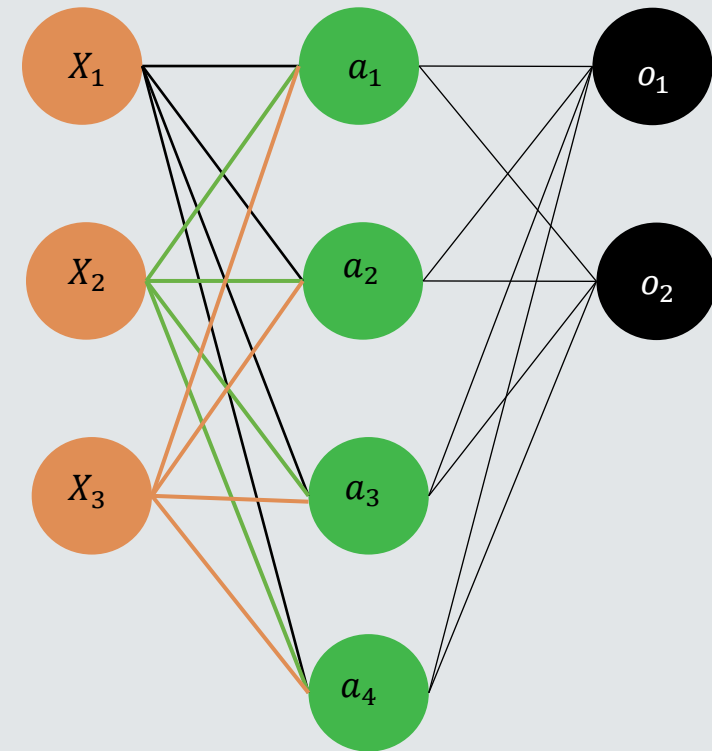
where

$$W_1^{(1)} = [W_{11} \quad W_{21} \quad W_{31}] \in \mathbb{R}^{1 \times 3}$$

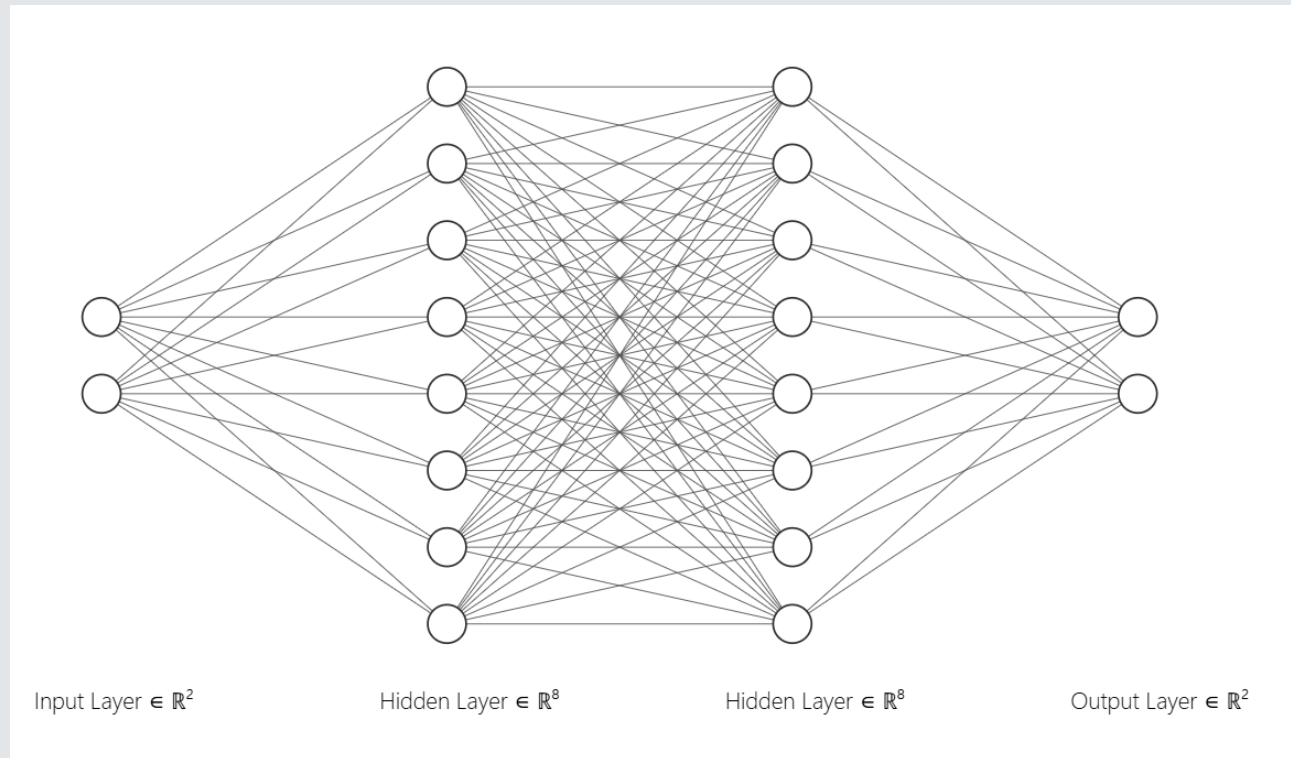
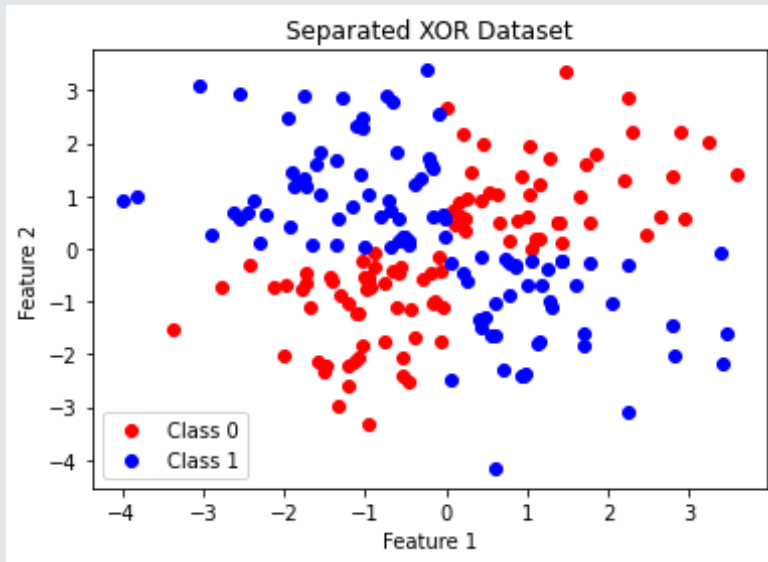
$W_{ij}^{(k)}$ (i is the starting node and j is the ending node, and k is the layer)

$$X = \begin{bmatrix} X_1 \\ X_2 \\ X_3 \end{bmatrix} \in \mathbb{R}^{3 \times 1}; b_1^{(1)} \in \mathbb{R}$$

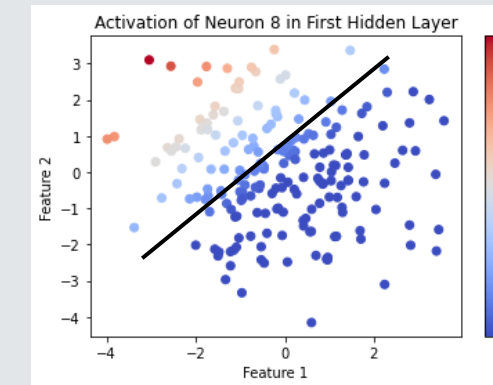
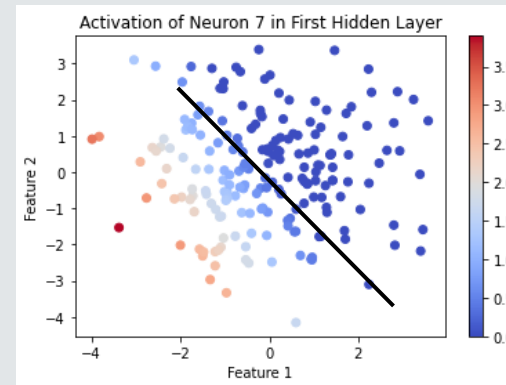
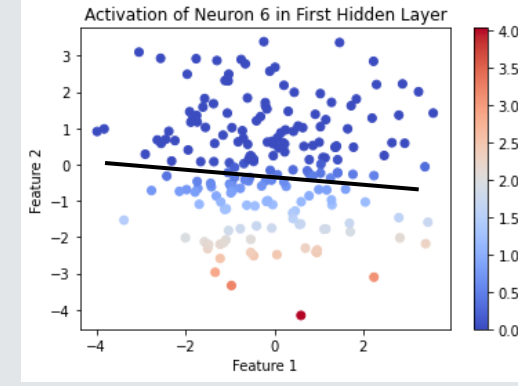
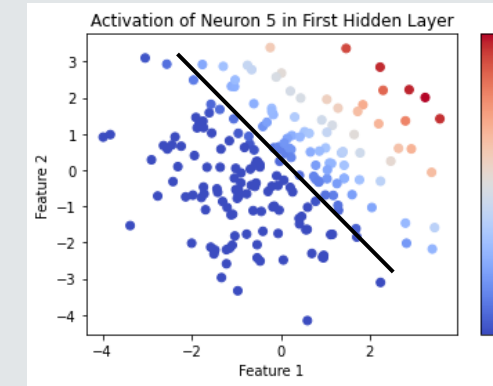
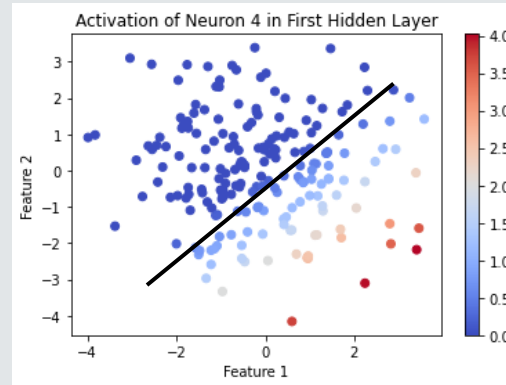
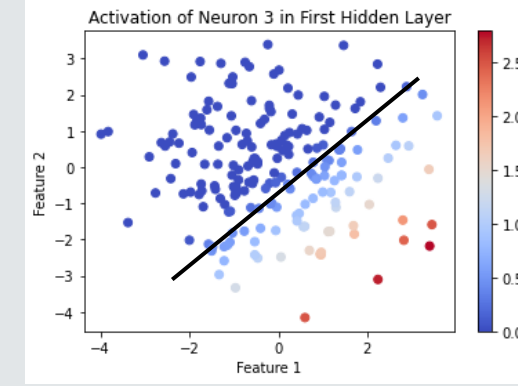
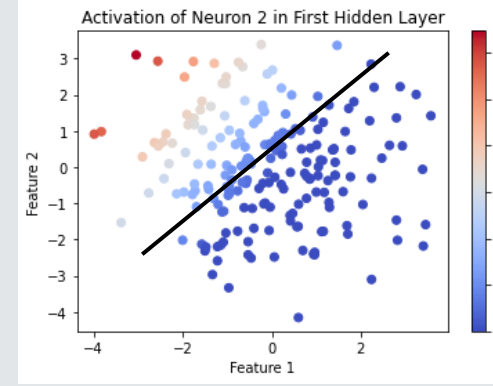
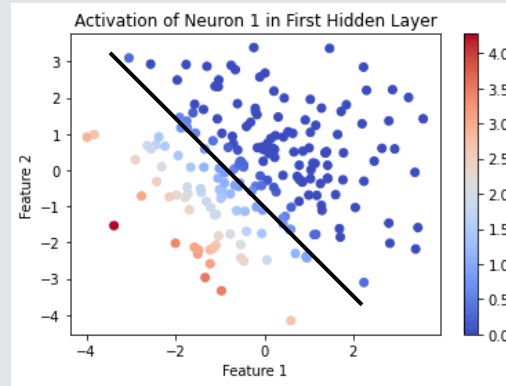
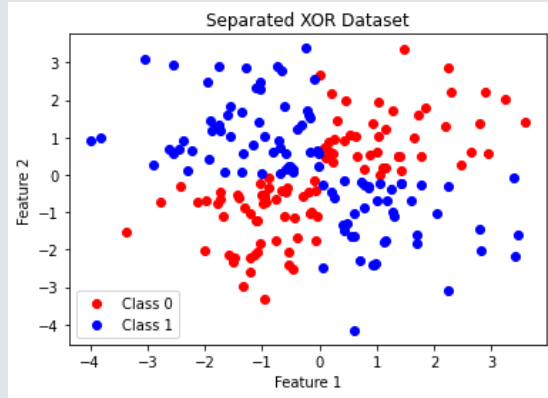
- $\bullet \quad o_1 = f(W_1^{(2)}a + b_1^{(2)}) = W_1^{(2)}a + b_1^{(2)}; f \text{ is a linear function}$
- $\bullet \quad o_2 = f(W_2^{(2)}a + b_2^{(2)}) = W_2^{(2)}a + b_2^{(2)}; f \text{ is a linear function}$
- $\bullet \quad l = \frac{1}{2}[(y_1 - o_1)^2 + (y_2 - o_2)^2]$ (MSE); where y_1, y_2 is the true value and o_1, o_2 is the predicted value

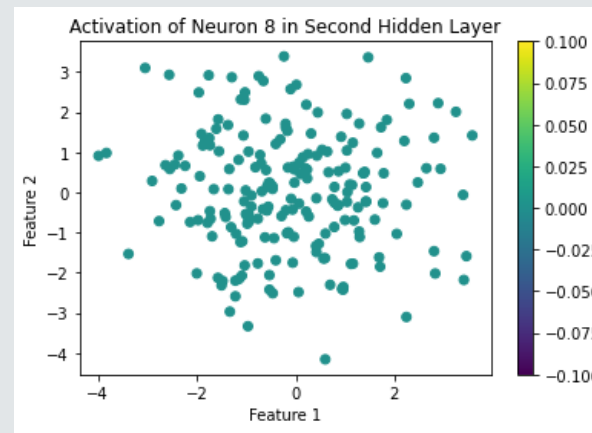
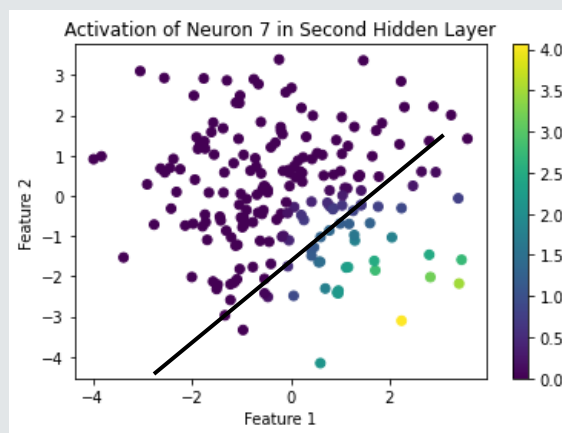
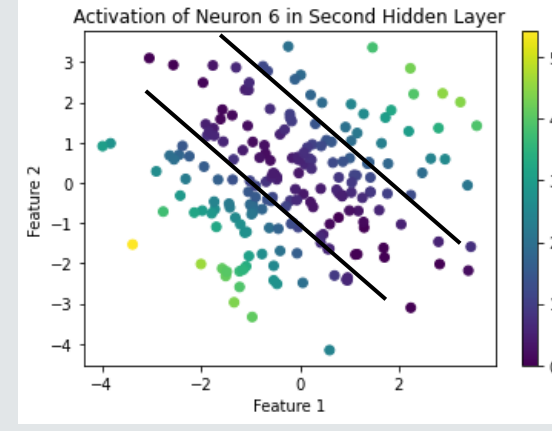
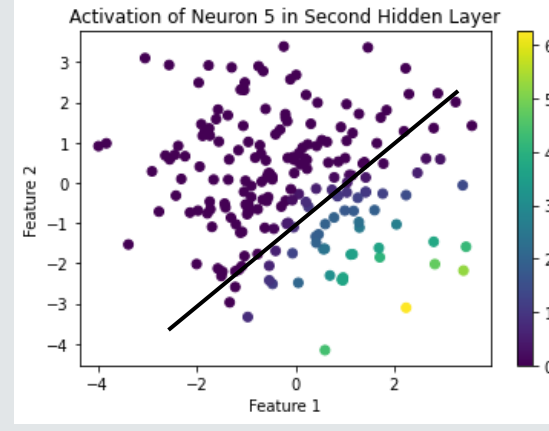
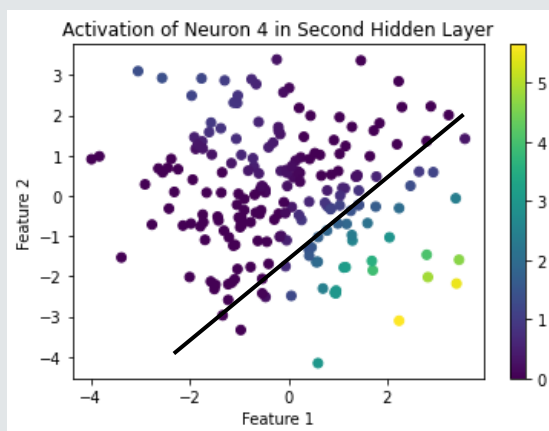
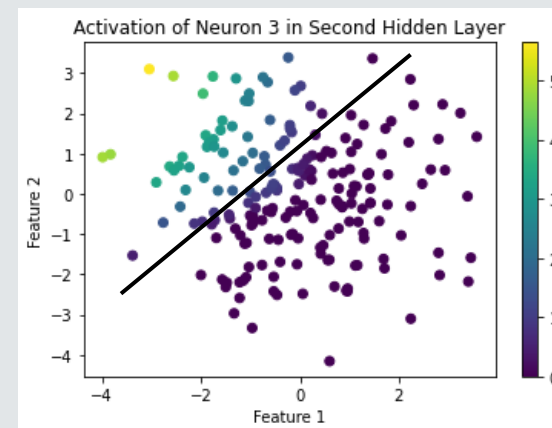
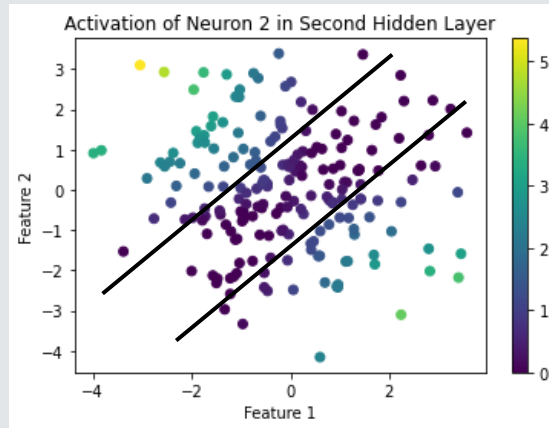
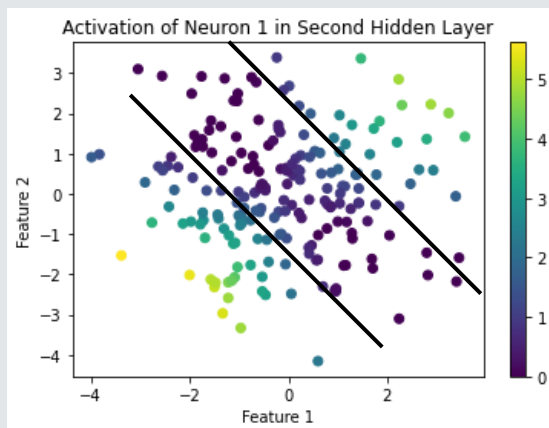


Feedforward Neural Network

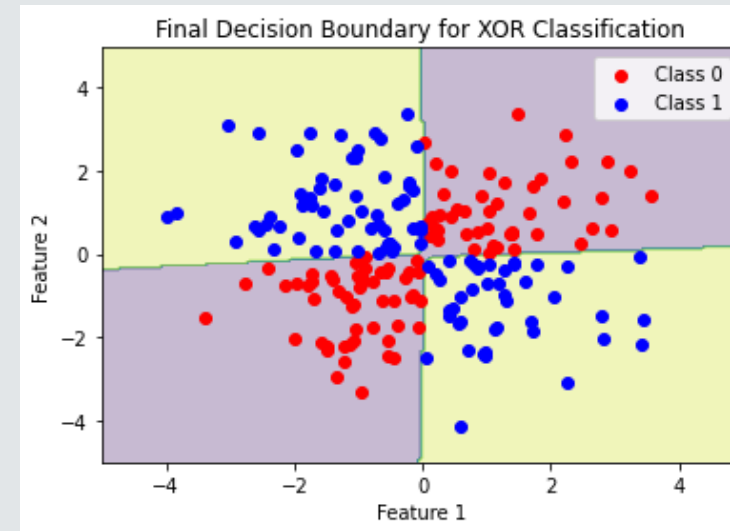
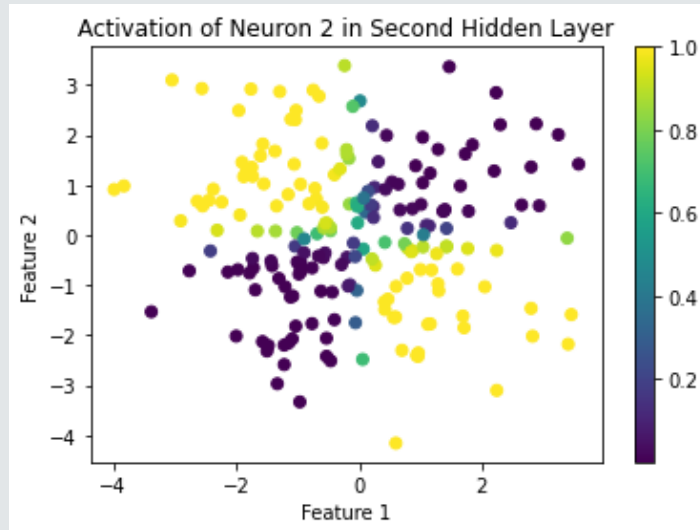
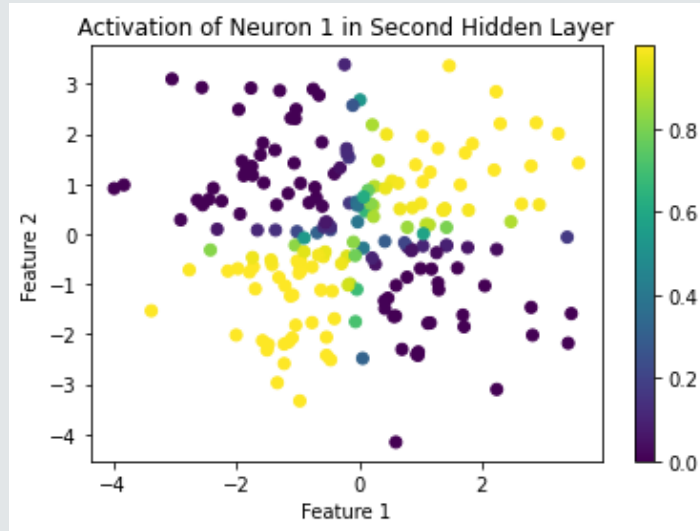


Feedforward Neural Network





Feedforward Neural Network



Feedforward Neural Network – Weight Initialization

- Random Initialization: weights are random initialization, typically from a uniform or normal distribution. The issue with this initialization is that it can lead to large or small gradients that can cause slow convergence or divergence.

Advantages: Allows neurons to learn different features

Disadvantages: Without further scaling, it can lead to large or small gradients, causing slow convergence or divergence.

- He Initialization: specifically designed for ReLU activation

$$W \sim \text{Normal}\left(0, \sqrt{\frac{2}{n_{in}}}\right)$$

Ensures that the variance of the weights does not shrink or explode.

- Xavier Initialization: for layer l , weights are initialized with a uniform or normal distribution scaled by

$$W \sim \text{Uniform}\left(-\frac{\sqrt{6}}{\sqrt{n_{in} + n_{out}}}, \frac{\sqrt{6}}{\sqrt{n_{in} + n_{out}}}\right)$$

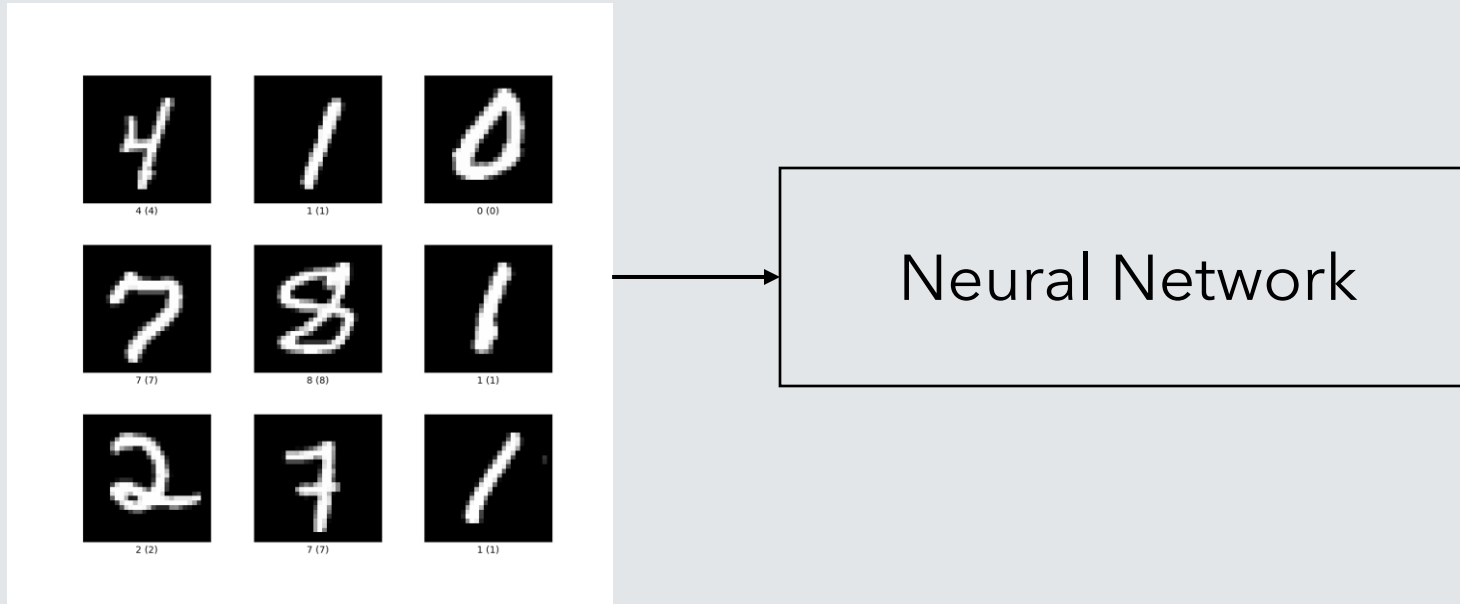
where n_{in} and n_{out} are the number of input and output units in the layer.

Balances the variance across layers, making it suitable for sigmoid and tanh activations.

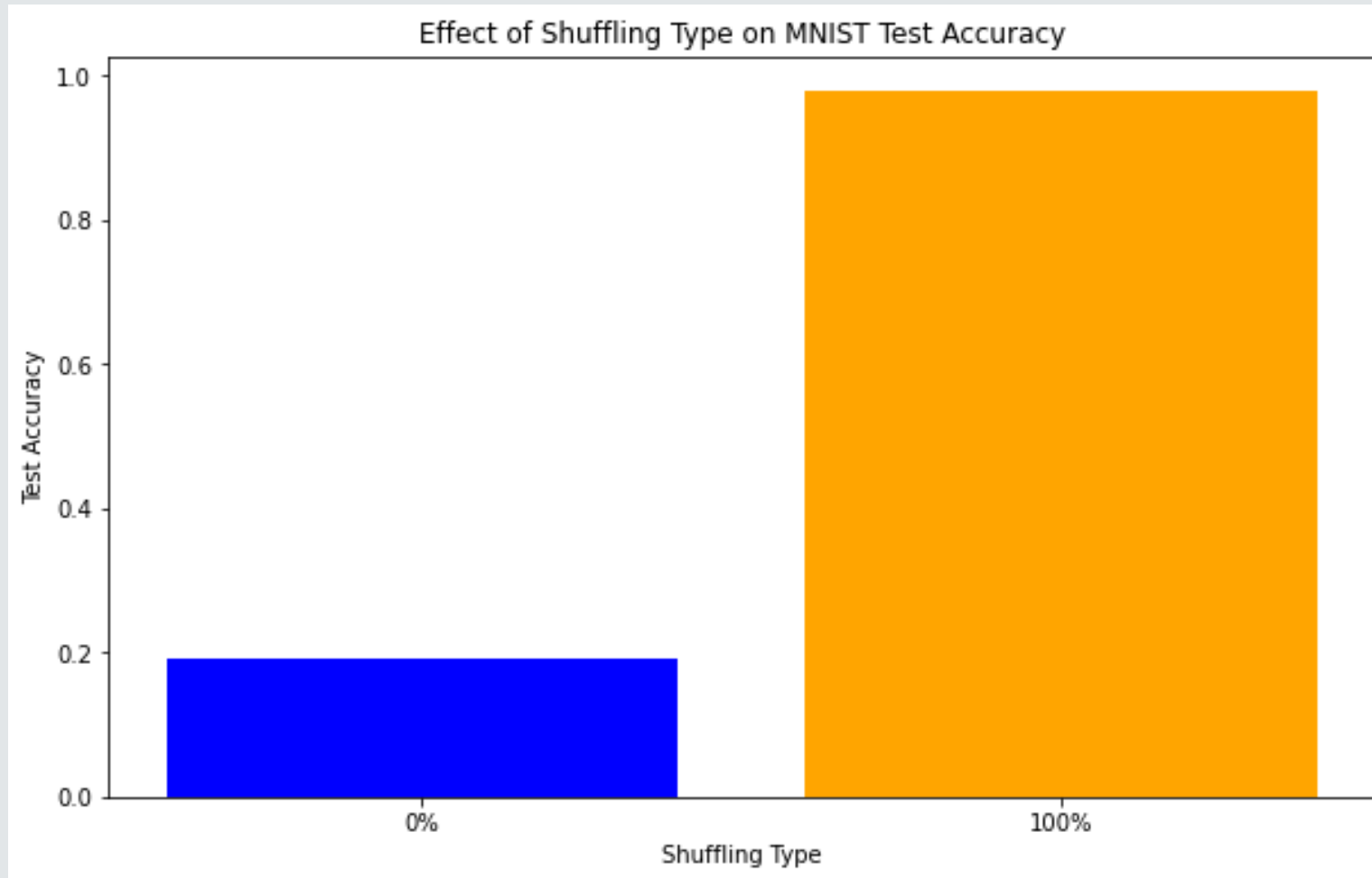
Feedforward Neural Network – Wider vs Deeper

- **Are there functions that can be expressed by wide and shallow neural networks, that cannot be approximated by any narrow neural network, unless its depth is very large?**
- (Correct) On the other hand, if the answer is negative, then depth generally plays a more significant role than width for the expressive power of neural networks. [Proven in <https://proceedings.mlr.press/v178/vardi22a/vardi22a.pdf>]
- (Wrong) If the answer is positive, then width and depth, in principle, play an incomparable role in the expressive power of neural networks, as sometimes depth can be more significant, and sometimes width.

Feedforward Neural Network – MNIST



Feedforward Neural Network – MNIST



Feedforward Neural Network – MNIST

