Deep Learning I: Neural Networks

Swati Mishra

Applications of Machine Learning (4AL3)

Fall 2024

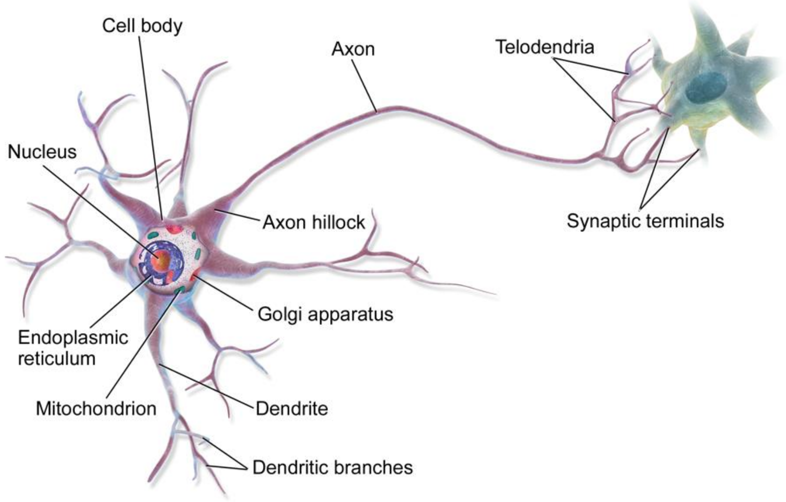
McMaster University
ENGINEERING

Review

- The concept of Uncertainty.
- Measure uncertainty in Machine Learning
- Active Learning
- Sampling Strategies in Active Learning

Neural Networks: Fundamentals

- Neural Networks are inspired from the brain structure seen in many organisms including humans.
- They form the fundamental building block of Deep Learning. They are also called feedforward networks, or multilayer perceptron (MLPs).



Picture Source: Wikipedia

Neural Networks: Fundamentals

- Neural Networks are inspired from the brain structure seen in many organisms including humans.
- They form the fundamental building block of Deep Learning. They are also called feedforward networks, or multilayer perceptron (MLPs).
- They take an input vector of p variables and can transform it to a target variable using a non-linear function. They improve upon the linear models.

$$f(x) = f_3(f_2(f_1(x))) \quad \textit{where, } x = (x_1, x_2, \dots, x_p)$$

Neural Networks: Fundamentals

- Neural Networks are inspired from the brain structure seen in many organisms including humans.
- They form the fundamental building block of Deep Learning. They are also called feedforward networks, or multilayer perceptron (MLPs).
  - When they have feedback, they are called Recurrent Neural Networks.
- They take an input vector of p variables and can transform it to a target variable using a non-linear function. They improve upon the linear models.

$$f(x) = f_3(f_2(f_1(x))) \quad \textit{where, } x = (x_1, x_2, \ldots, x_p)$$

McMaster University

Neural Networks: Fundamentals

$$f(x) = f_3(f_2(f_1(x)))$$

Input Layer

- **Input layer**: Contains the input weights

Neural Networks: Fundamentals

$$f(x) = f_3(f_2(f_1(x)))$$

Hidden Layers Input Layer

- **Input layer**: Contains the input weights
- **Hidden layer(s)**: The behavior of these layers is determined by the learning algorithm to determine close approximation of $f(x)$.

Neural Networks: Fundamentals

$$f(x) = f_3(f_2(f_1(x)))$$

Output Layer Hidden Layers Input Layer

- **Input layer**: Contains the input weights
- **Hidden layer(s)**: The behavior of these layers is determined by the learning algorithm to determine close approximation of $f(x)$.
- **Output layer**: Where the final output is available for use.

Neural Networks: Fundamentals

$$f(x) = f_3(f_2(f_1(x)))$$

Output Layer Hidden Layers Input Layer

- **Input layer**: Contains the input weights
- **Hidden layer(s)**: The behavior of these layers is determined by the learning algorithm to determine close approximation of $f(x)$.
- **Output layer**: Where the final output is available for use.
- f_2 is the **activation function** that computes the output of the hidden layer.

Neural Networks: Fundamentals

- Let us consider a neural network model written in linear form as:

$$f(x) = \beta_0 + \sum_{k=1}^{K} \beta_k h_k(X)$$

- Then the functions of the hidden layer can be written as:

$$h_k(X) = \mathrm{g}(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j)$$

where $\mathrm{g}(z)$ $=$ non-linear activation function

K $=$ number of activations

$h_k(X)$ instead of X means some transformation of X as opposed to X

Neural Networks: Fundamentals

- Let us consider a neural network model written in linear form as :

$$f(x) = \beta_0 + \sum_{k=1}^{K} \beta_k h_k(X)$$

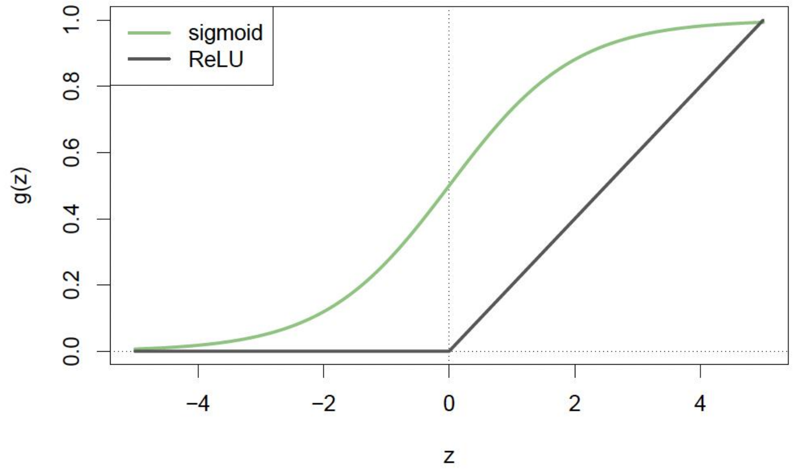
- Then the functions of the hidden layer can be written as:

$$h_k(X) = g(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j)$$

where $g(z)$ = non-linear activation funct

K = number of activations

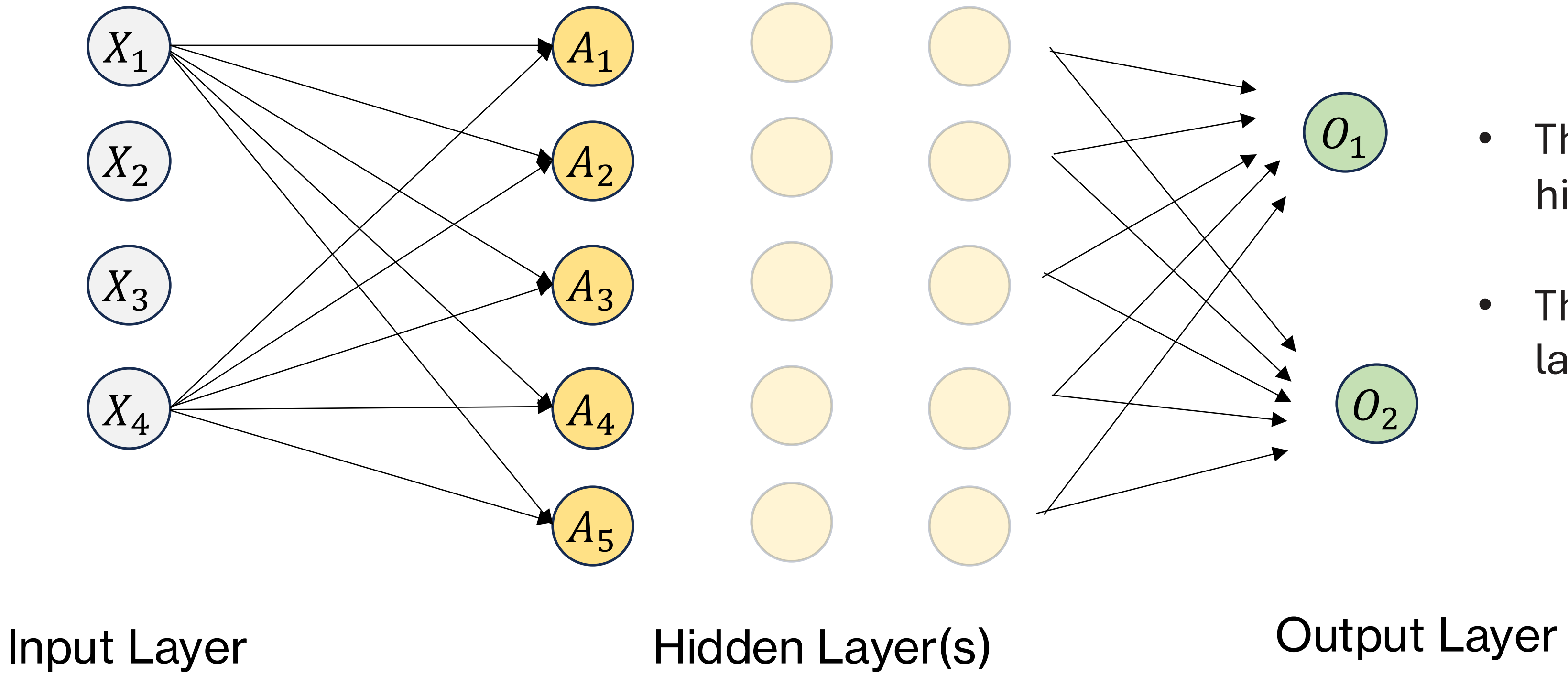
$$g(z) = (z)_+ = \begin{cases} 0 & \text{if } z < 0 \\ z & \text{otherwise.} \end{cases}$$



- Most popular activation function nowadays is Rectified Linear Units, few years ago it was Sigmoid.

Neural Networks: Architecture

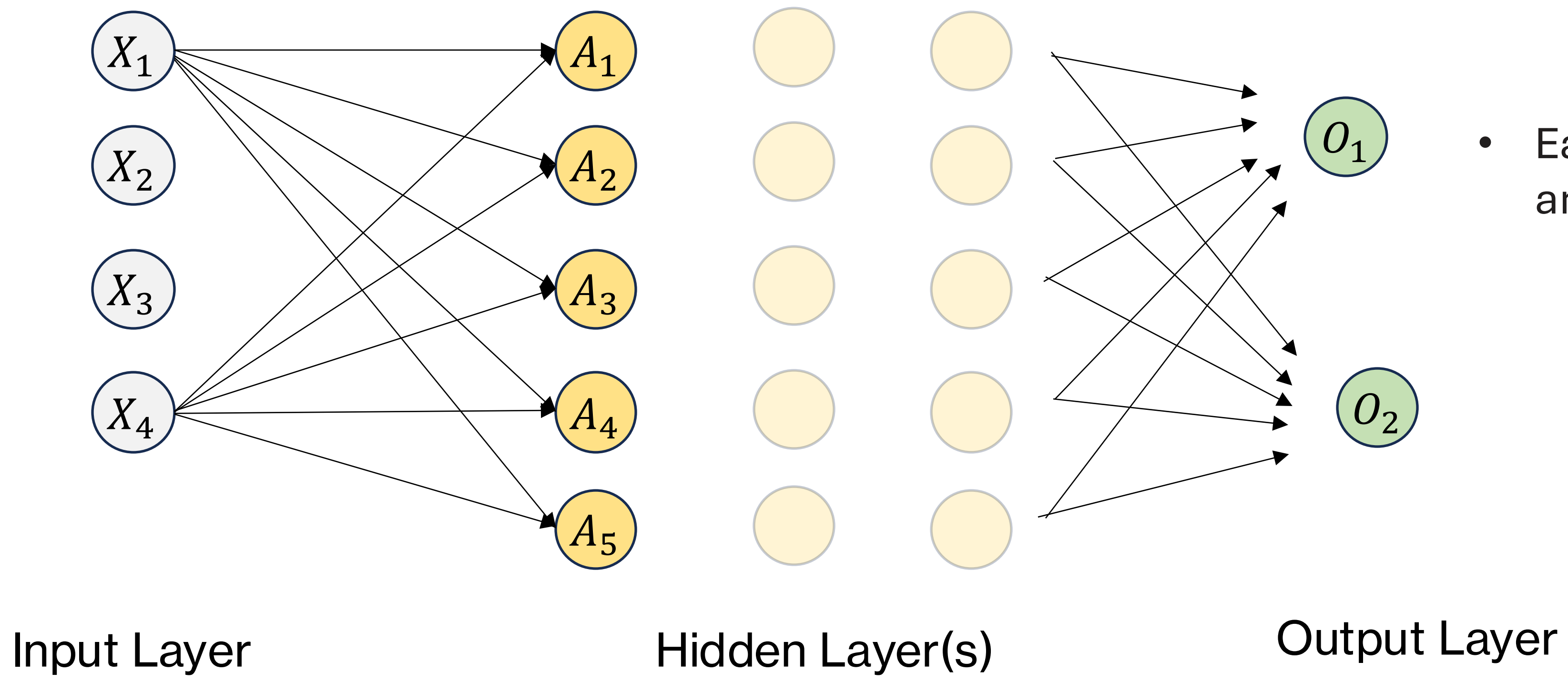
- Let us consider a neural network model in visual form as:



- The dimensionality of the hidden layers is called **width**.
- The number of the hidden layers is called the **depth**.

Neural Networks: Architecture

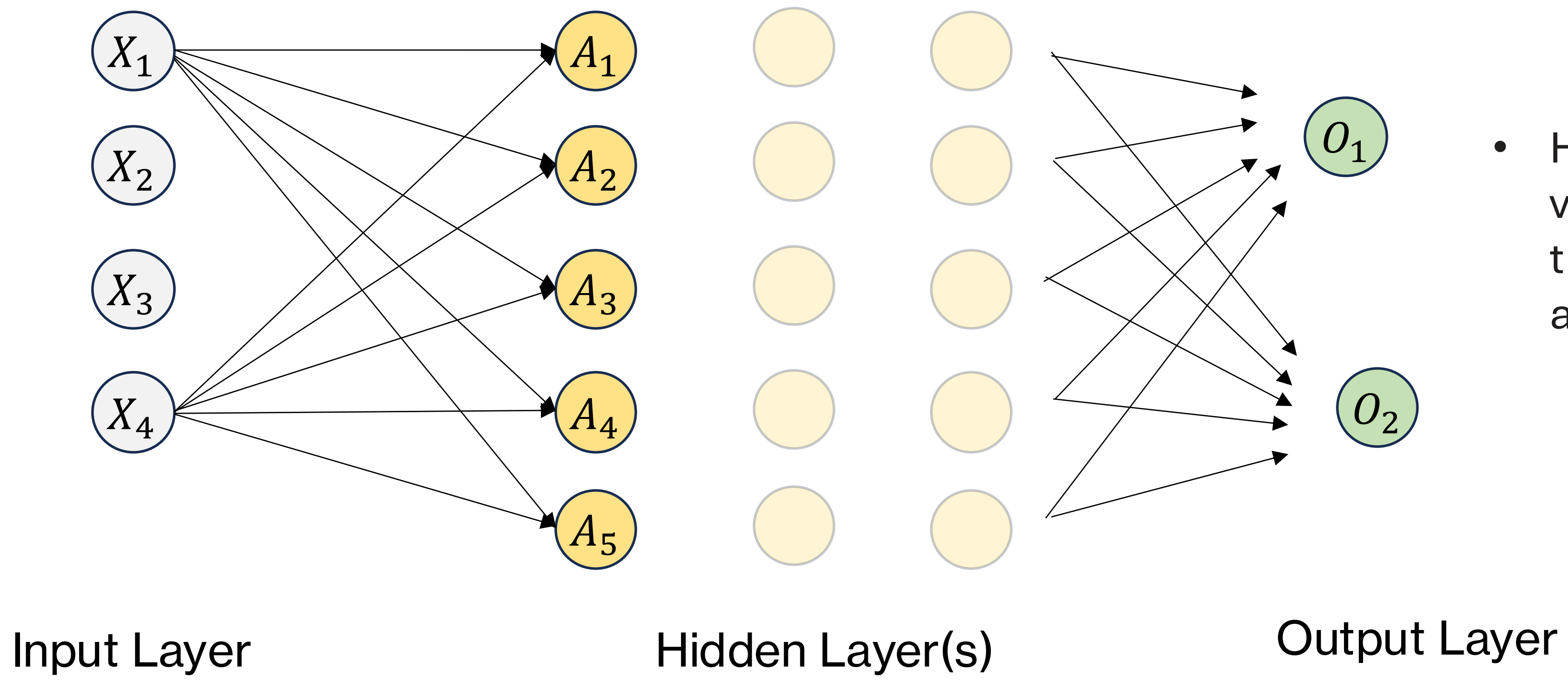
- Let us consider a neural network model in visual form as:



- Each of these activations are **units**

Neural Networks: Architecture

- Let us consider a neural network model in visual form as:



- Hidden layers are vector values and essentially a transformation function applied to input.

Neural Networks: Fundamentals

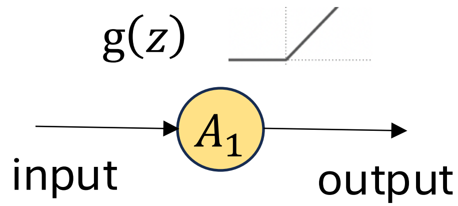
- When there are multiple hidden layers, the first hidden layer:

$$A_k^{(1)} = h_k^{(1)}(X) = g(w_{k0}^{(1)} + \sum_{j=1}^{p} w_{kj}^{(1)} X_j)$$

- The second hidden layer is described as:

$$A_l^{(2)} = h_l^{(2)}(X) = g(w_{l0}^{(2)} + \sum_{k=1}^{K_1} w_{lk}^{(2)} A_k^{(1)})$$

where $g(z)$ = non-linear activation function



$$g(z) = (z)_+ = \begin{cases} 0 & \text{if } z < 0 \\ z & \text{otherwise.} \end{cases}$$

K_1 = number of units in first hidden layer

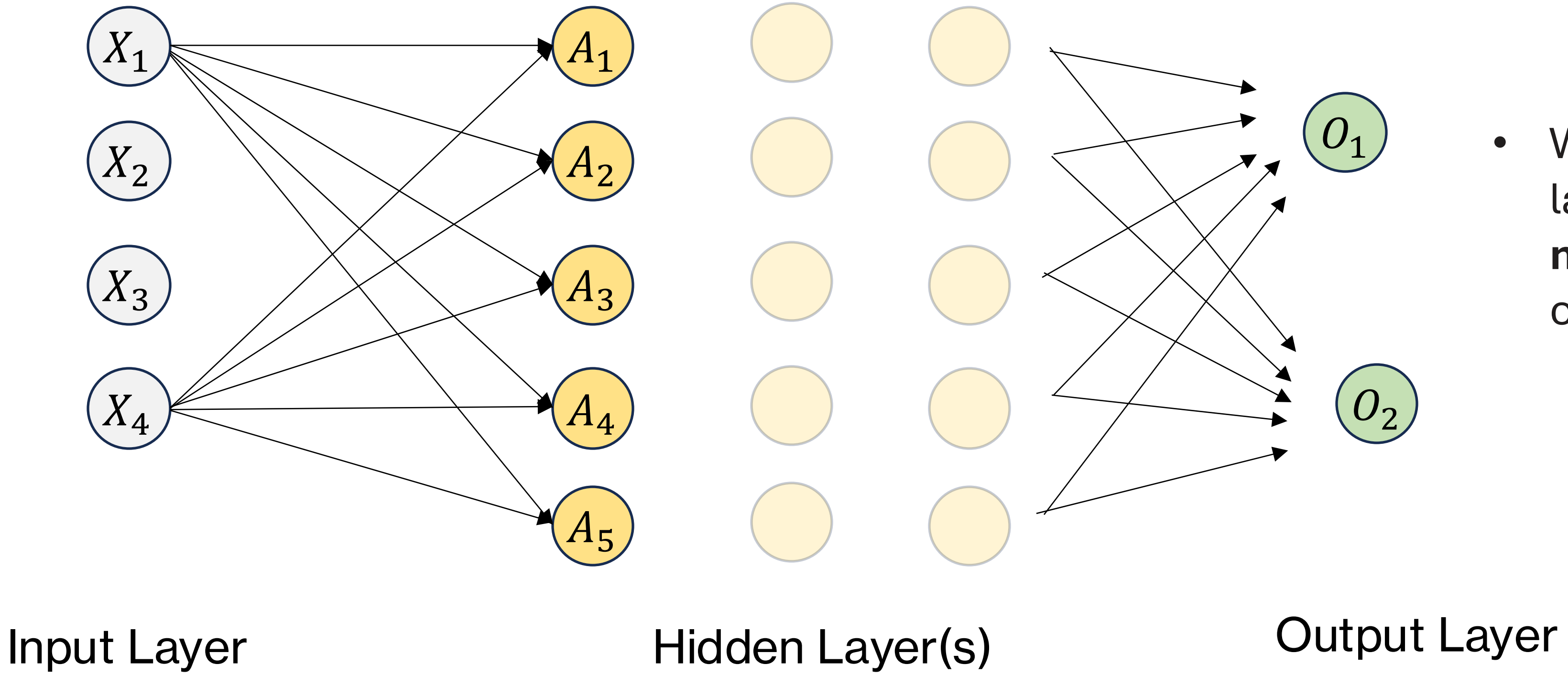
K_2 = number of units in second hidden layer

$h_k(X)$ means some transformation of X.

$A_k^{(1)}$ = activations of first hidden layer

Neural Networks: Architecture

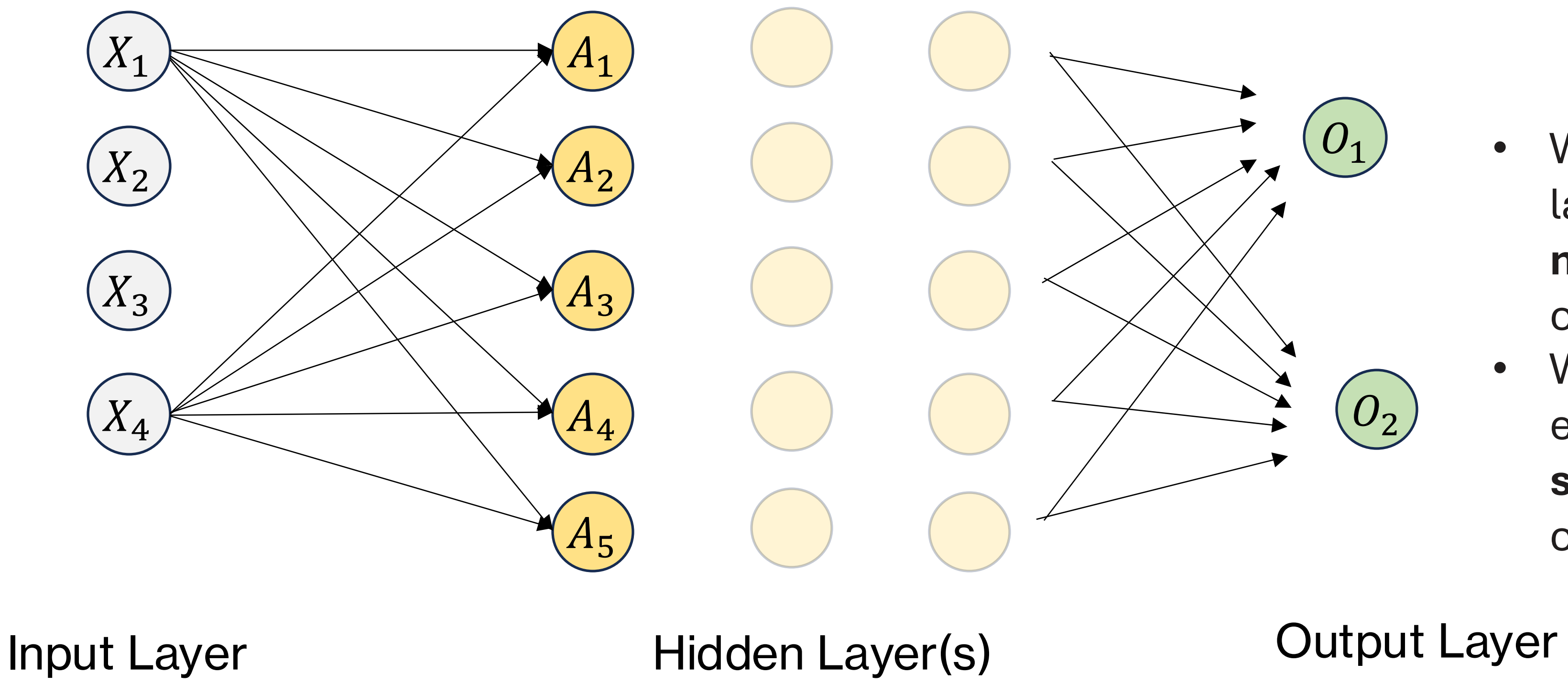
- Let us consider a neural network model in visual form as:



- We can envision these layers as consisting of **many vector to scalar** operations.

Neural Networks: Architecture

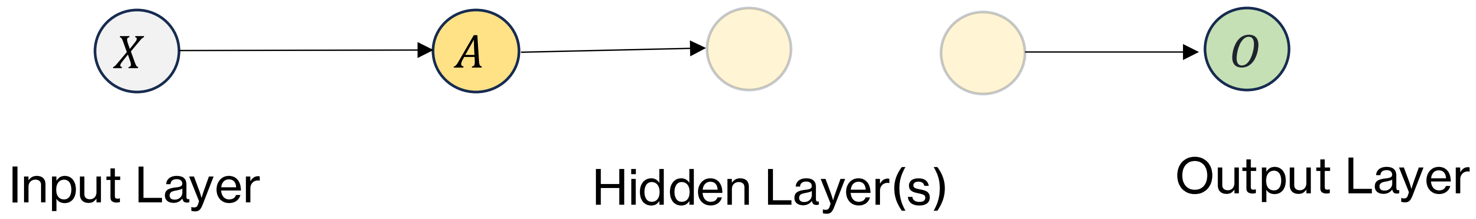
- Let us consider a neural network model in visual form as:



- We can envision these layers as consisting of **many vector to scalar** operations
- We can also envision each layer performing a **single vector-to-vector** operation.

Neural Networks: Architecture

- We can draw the neural architecture in this form as well:

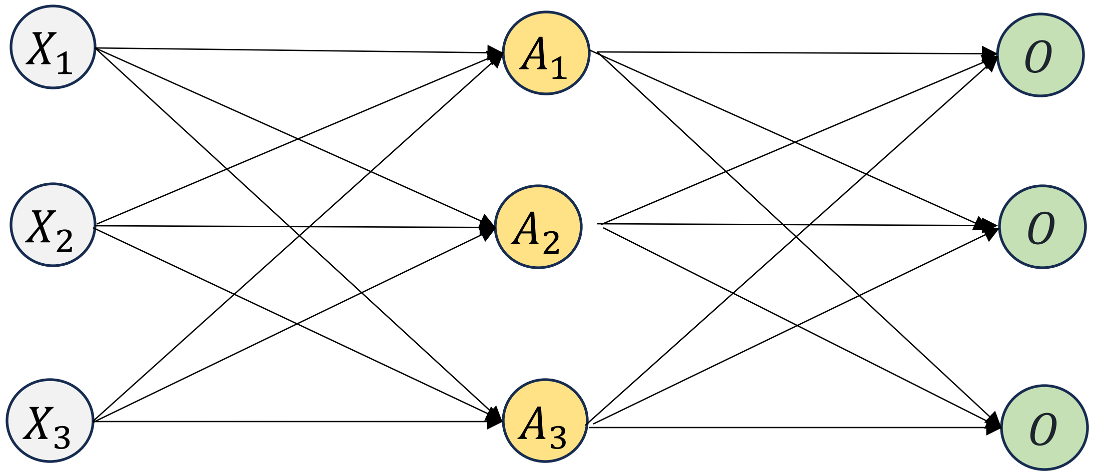


- We can also envision each layer performing a **single vector-to-vector** operation.

- Large architectures are drawn in this form.

Neural Networks: Architecture

- Let's consider an example of neural network architecture.

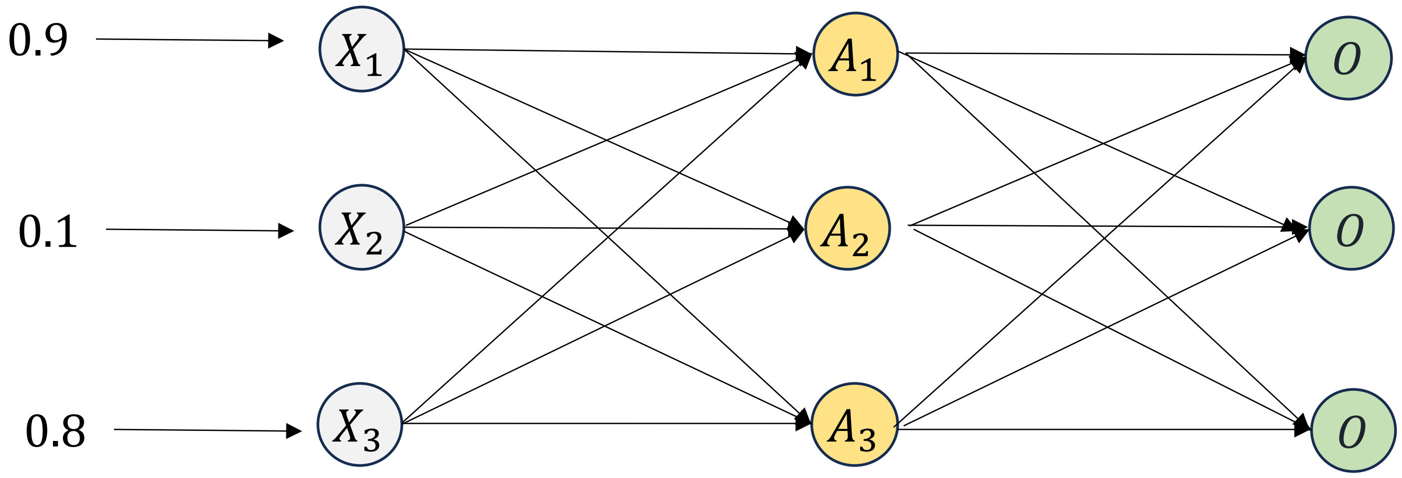


Input vector

$$X = \begin{vmatrix} 0.9 \\ 0.1 \\ 0.8 \end{vmatrix}$$

Neural Networks: Architecture

- Let's consider an example of neural network architecture.

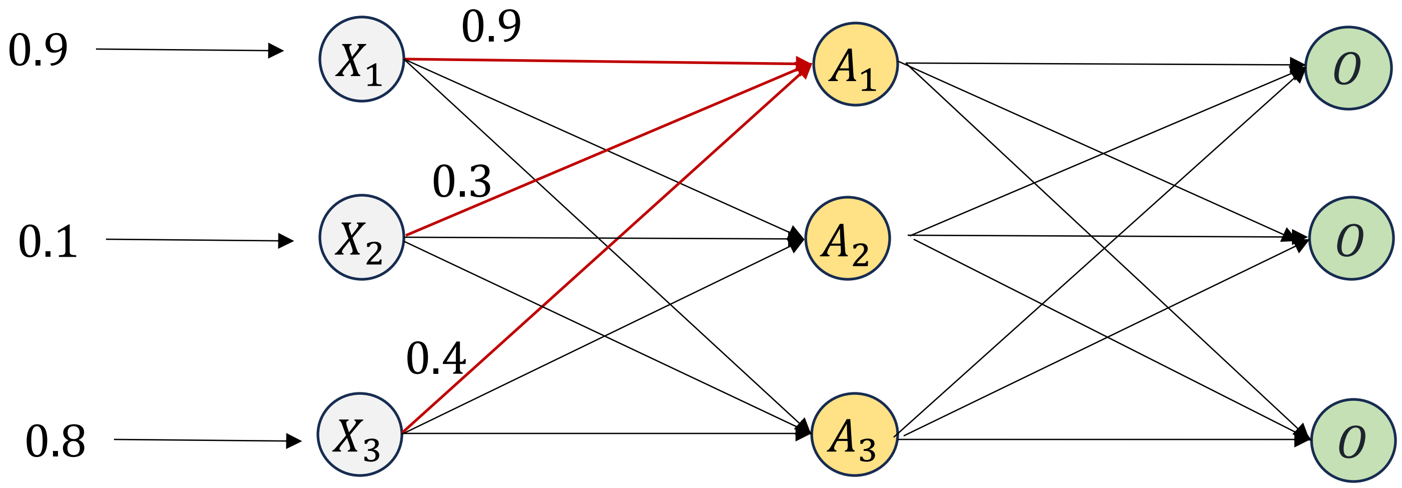


$$W = \begin{vmatrix} 0.9 & 0.3 & 0.4 \\ 0.2 & 0.8 & 0.2 \\ 0.1 & 0.5 & 0.6 \end{vmatrix}$$

$$X = \begin{vmatrix} 0.9 \\ 0.1 \\ 0.8 \end{vmatrix}$$

Neural Networks: Architecture

- Let's consider an example of neural network architecture.

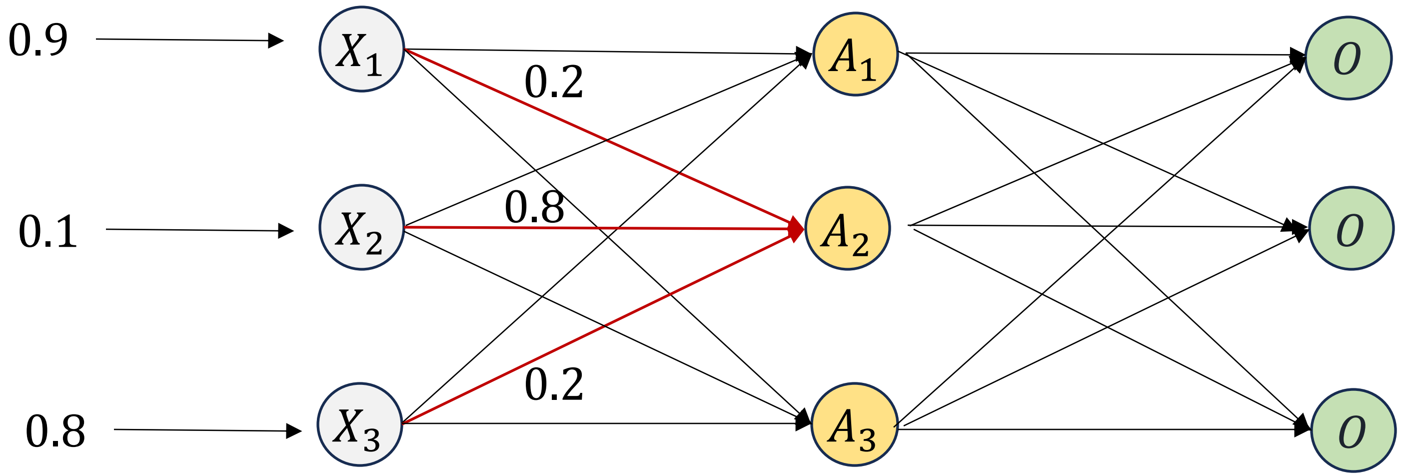


$$W_{XA} = \begin{vmatrix} 0.9 & 0.3 & 0.4 \\ 0.2 & 0.8 & 0.2 \\ 0.1 & 0.5 & 0.6 \end{vmatrix}$$

$$X = \begin{vmatrix} 0.9 \\ 0.1 \\ 0.8 \end{vmatrix}$$

Neural Networks: Architecture

- Let’s consider an example of neural network architecture.

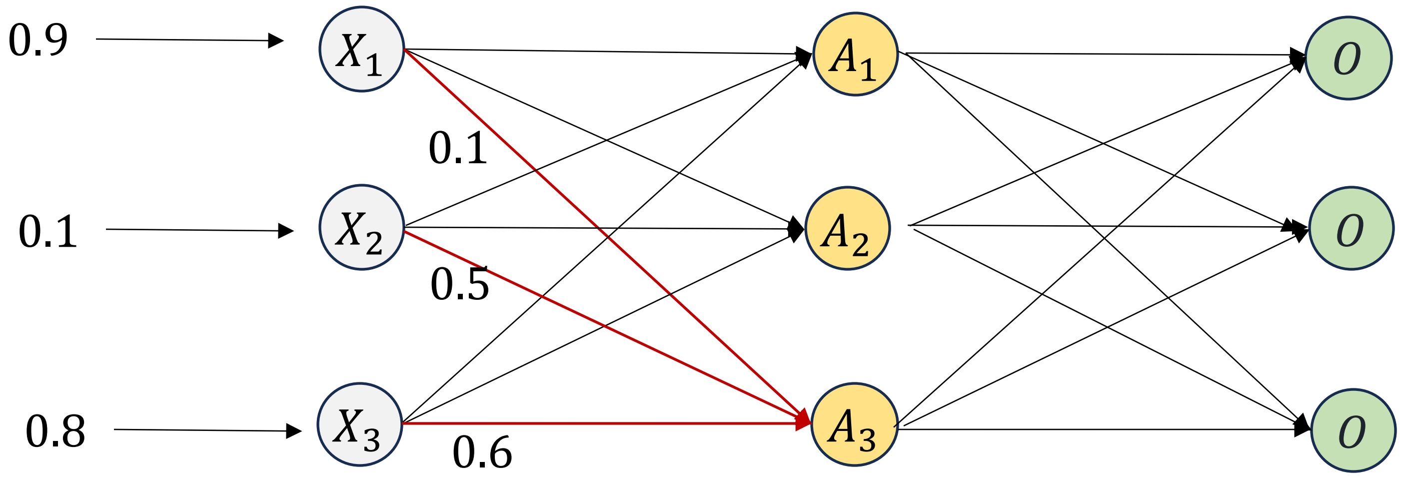


$$X = \begin{vmatrix} 0.9 \\ 0.1 \\ 0.8 \end{vmatrix}$$

$$W_{XA} = \begin{vmatrix} 0.9 & 0.3 & 0.4 \\ \color{red}{0.2} & \color{red}{0.8} & \color{red}{0.2} \\ 0.1 & 0.5 & 0.6 \end{vmatrix}$$

Neural Networks: Architecture

- Let's consider an example of neural network architecture.

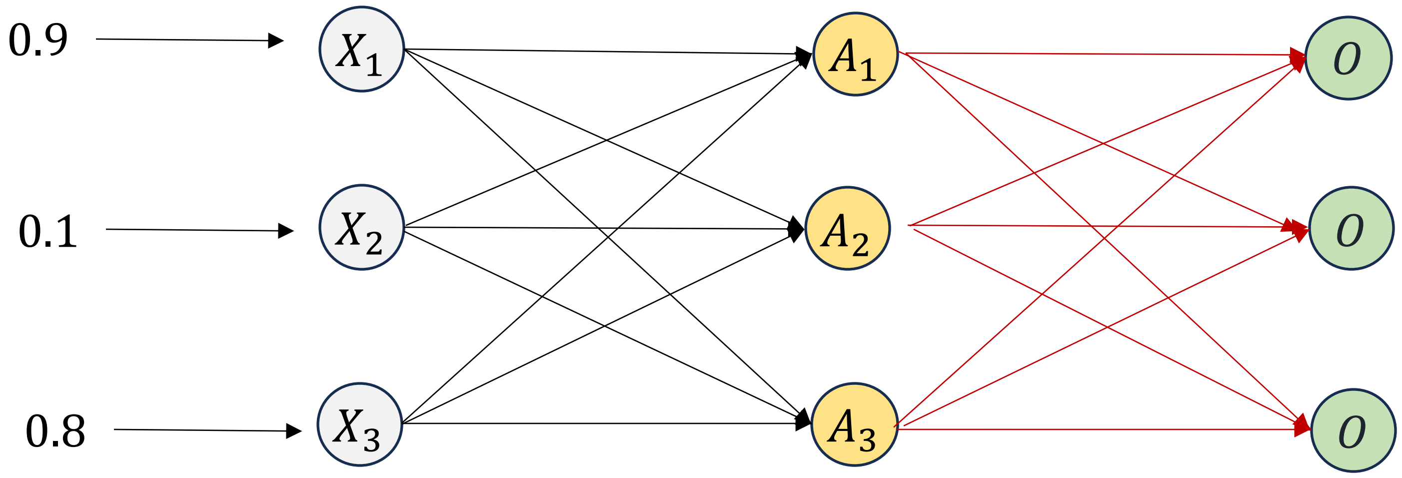


$$X = \begin{vmatrix} 0.9 \\ 0.1 \\ 0.8 \end{vmatrix}$$

$$W_{XA} = \begin{vmatrix} 0.9 & 0.3 & 0.4 \\ 0.2 & 0.8 & 0.2 \\ \color{red}{0.1} & \color{red}{0.5} & \color{red}{0.6} \end{vmatrix}$$

Neural Networks: Architecture

- Let's consider an example of neural network architecture.

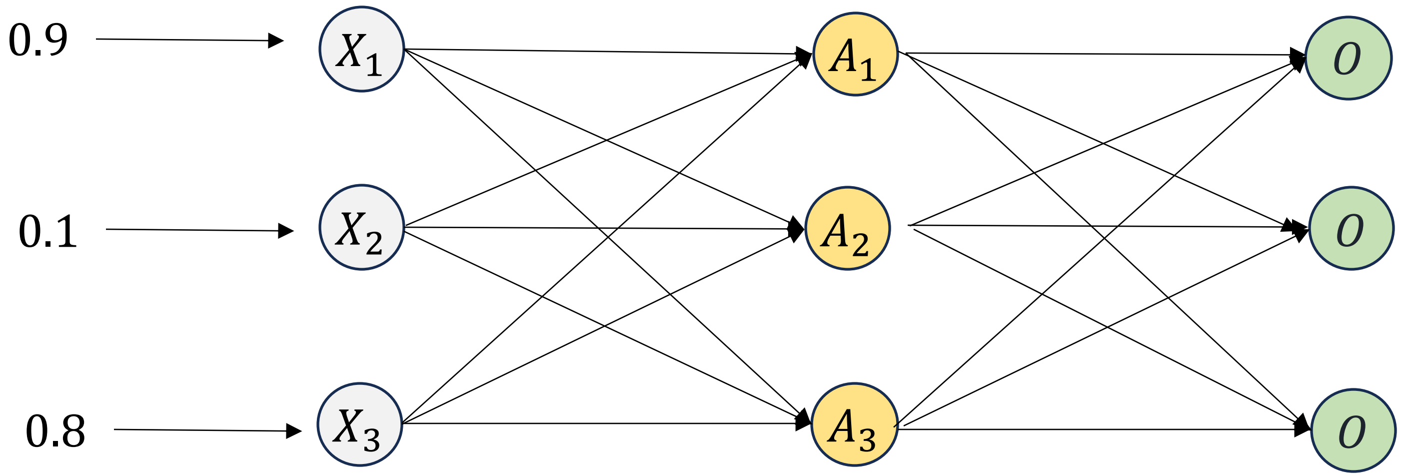


$$W_{AO} = \begin{vmatrix} 0.3 & 0.7 & 0.5 \\ 0.6 & 0.5 & 0.2 \\ 0.8 & 0.1 & 0.9 \end{vmatrix}$$

$$X = \begin{vmatrix} 0.9 \\ 0.1 \\ 0.8 \end{vmatrix} \quad W_{XA} = \begin{vmatrix} 0.9 & 0.3 & 0.4 \\ 0.2 & 0.8 & 0.2 \\ 0.1 & 0.5 & 0.6 \end{vmatrix}$$

Neural Networks: Architecture

- Let's consider an example of neural network architecture.



$$X_{hidden} = W_{XA}.\mathrm{X}$$

```python
import numpy as np
w = np.array([[0.9,0.3,0.4],
              [0.2,0.8,0.2],
              [0.1,0.5,0.6]])
x = np.array([[0.9],[0.1],[0.8]])
w.dot(x)
```

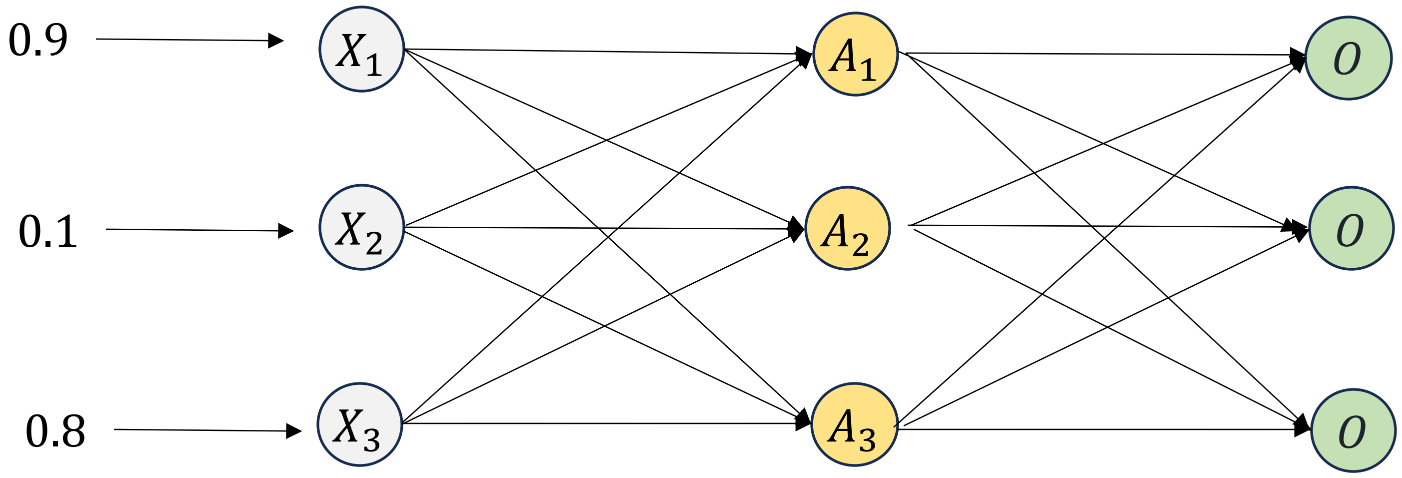
0.0s

```
array([[1.16],
       [0.42],
       [0.62]])
```

$$X = \begin{vmatrix} 0.9 \\ 0.1 \\ 0.8 \end{vmatrix} \quad W_{XA} = \begin{vmatrix} 0.9 & 0.3 & 0.4 \\ 0.2 & 0.8 & 0.2 \\ 0.1 & 0.5 & 0.6 \end{vmatrix} \quad W_{AO} = \begin{vmatrix} 0.3 & 0.7 & 0.5 \\ 0.6 & 0.5 & 0.2 \\ 0.8 & 0.1 & 0.9 \end{vmatrix}$$

Neural Networks: Architecture

- Let's consider an example of neural network architecture.



$$X = \begin{vmatrix} 0.9 \\ 0.1 \\ 0.8 \end{vmatrix} \quad W_{XA} = \begin{vmatrix} 0.9 & 0.3 & 0.4 \\ 0.2 & 0.8 & 0.2 \\ 0.1 & 0.5 & 0.6 \end{vmatrix} \quad W_{AO} = \begin{vmatrix} 0.3 & 0.7 & 0.5 \\ 0.6 & 0.5 & 0.2 \\ 0.8 & 0.1 & 0.9 \end{vmatrix}$$

$$X_{hidden} = W_{XA}.\mathrm{X}$$

```python
import numpy as np
w = np.array([[0.9,0.3,0.4],
              [0.2,0.8,0.2],
              [0.1,0.5,0.6]])
x = np.array([[0.9],[0.1],[0.8]])
w.dot(x)
```

✓ 0.0s

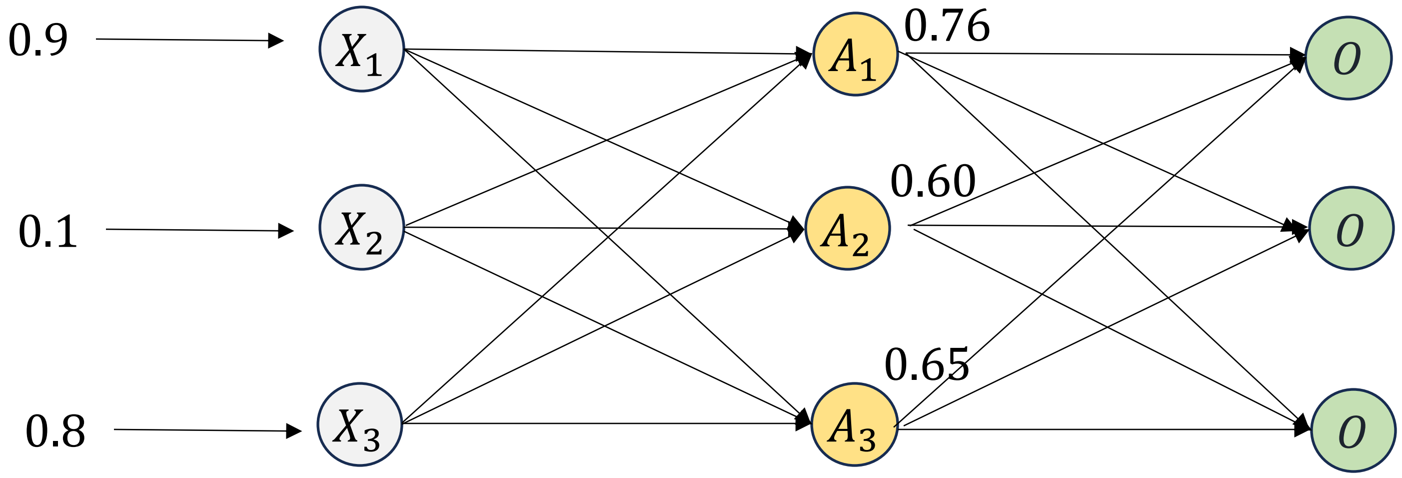
```
array([[1.16],
       [0.42],
       [0.62]])
```

Sigmoid function

$$g(z) = \frac{e^z}{1+e^z} = \frac{1}{1+e^{-z}},$$

Neural Networks: Architecture

- Let's consider an example of neural network architecture.



$$X = \begin{vmatrix} 0.9 \\ 0.1 \\ 0.8 \end{vmatrix} \quad W_{XA} = \begin{vmatrix} 0.9 & 0.3 & 0.4 \\ 0.2 & 0.8 & 0.2 \\ 0.1 & 0.5 & 0.6 \end{vmatrix} \quad W_{AO} = \begin{vmatrix} 0.3 & 0.7 & 0.5 \\ 0.6 & 0.5 & 0.2 \\ 0.8 & 0.1 & 0.9 \end{vmatrix}$$

$$X_{hidden} = W_{XA}.\mathrm{X}$$

```python
import numpy as np
w = np.array([[0.9,0.3,0.4],
              [0.2,0.8,0.2],
              [0.1,0.5,0.6]])
x = np.array([[0.9],[0.1],[0.8]])
w.dot(x)
```
✓ 0.0s

```python
def sigmoid(z):
    return 1/(1 + np.exp(-z))
```
[6] ✓ 0.0s

```python
a = w.dot(x)
sigmoid(a)
```
[7] ✓ 0.0s

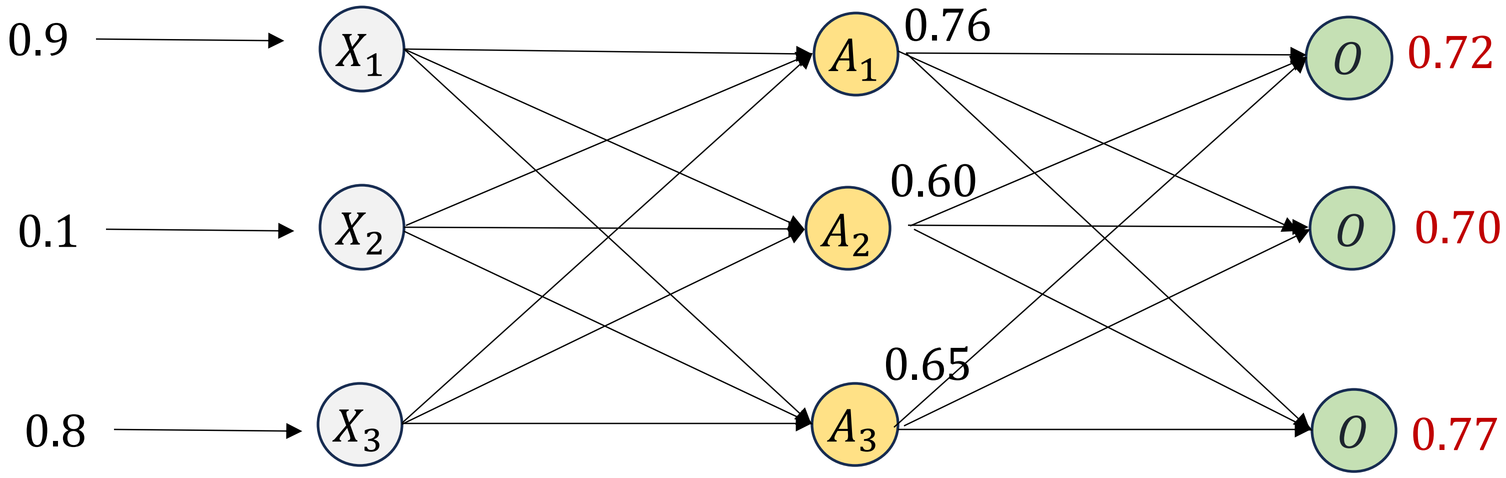
```
array([[0.76133271],
       [0.60348325],
       [0.65021855]])
```

McMaster University

Neural Networks: Architecture

- Let's consider an example of neural network architecture.

$$X_{output} = W_{AO} . X_{hidden}$$




```
b = sigmoid(a)
w2=np.array([[0.3,0.7,0.5],
             [0.6,0.5,0.2],
             [0.8,0.1,0.9]])
sigmoid(w2.dot(b))
```

[8] ✓ 0.0s

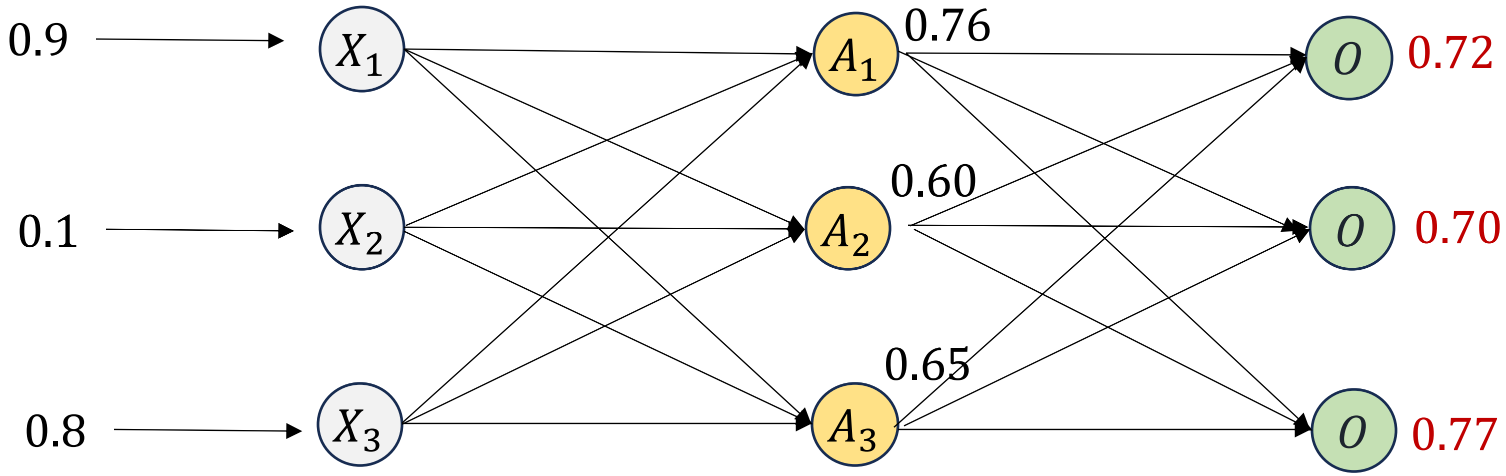
```
array([[0.72630335],
       [0.70859807],
       [0.77809706]])
```

$$X = \begin{vmatrix} 0.9 \\ 0.1 \\ 0.8 \end{vmatrix} \quad W_{XA} = \begin{vmatrix} 0.9 & 0.3 & 0.4 \\ 0.2 & 0.8 & 0.2 \\ 0.1 & 0.5 & 0.6 \end{vmatrix} \quad W_{AO} = \begin{vmatrix} 0.3 & 0.7 & 0.5 \\ 0.6 & 0.5 & 0.2 \\ 0.8 & 0.1 & 0.9 \end{vmatrix}$$

Neural Networks: Architecture

- Let's consider an example of neural network architecture.

$$X_{output} = W_{AO}.X_{hidden}$$




```
b = sigmoid(a)
w2=np.array([[0.3,0.7,0.5],
             [0.6,0.5,0.2],
             [0.8,0.1,0.9]])
sigmoid(w2.dot(b))
```

[8] ✓ 0.0s

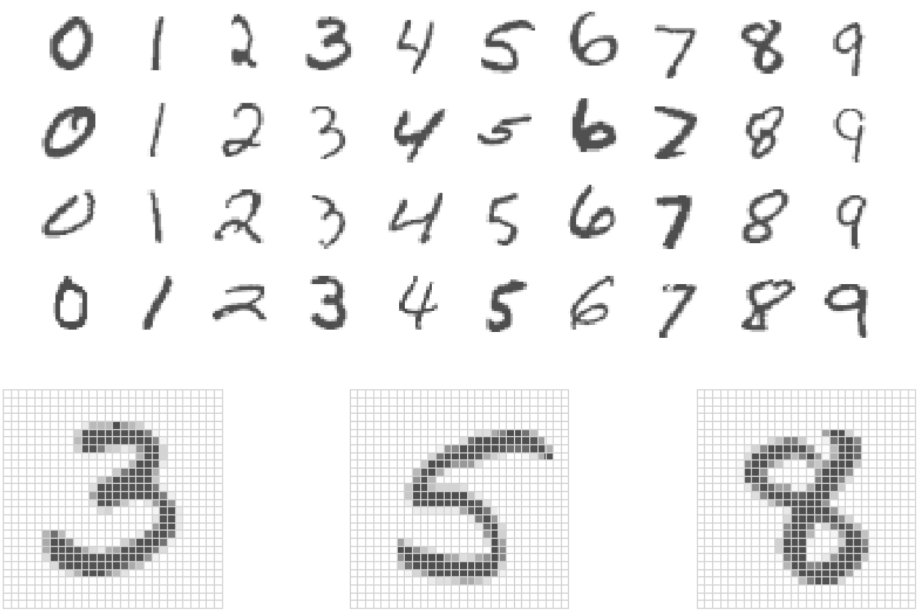
```
array([[0.72630335],
       [0.70859807],
       [0.77809706]])
```

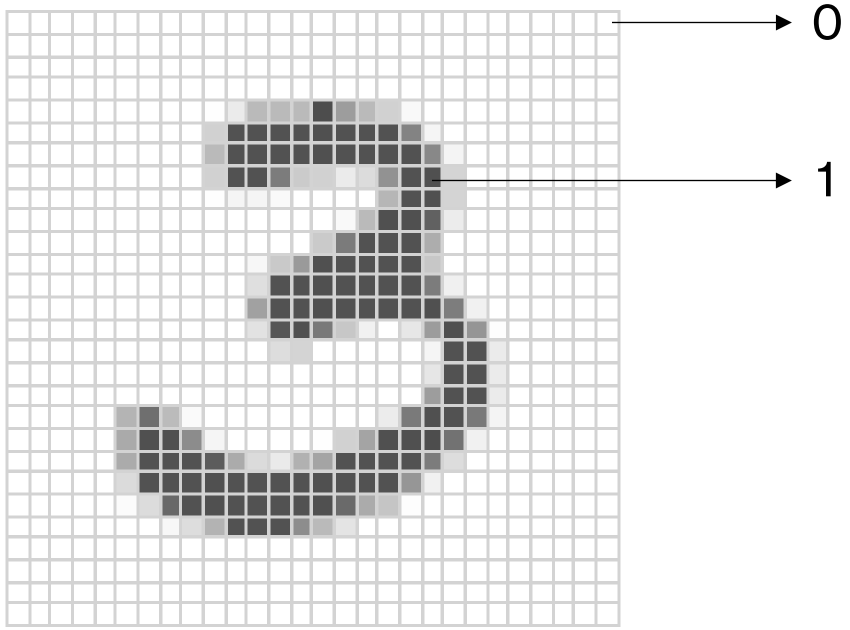
$$X = \begin{vmatrix} 0.9 \\ 0.1 \\ 0.8 \end{vmatrix} \quad W_{XA} = \begin{vmatrix} 0.9 & 0.3 & 0.4 \\ 0.2 & 0.8 & 0.2 \\ 0.1 & 0.5 & 0.6 \end{vmatrix} \quad W_{AO} = \begin{vmatrix} 0.3 & 0.7 & 0.5 \\ 0.6 & 0.5 & 0.2 \\ 0.8 & 0.1 & 0.9 \end{vmatrix}$$

Is there a problem with this example?

Neural Networks: Architecture

- Let’s consider an example of neural network architecture… at scale..

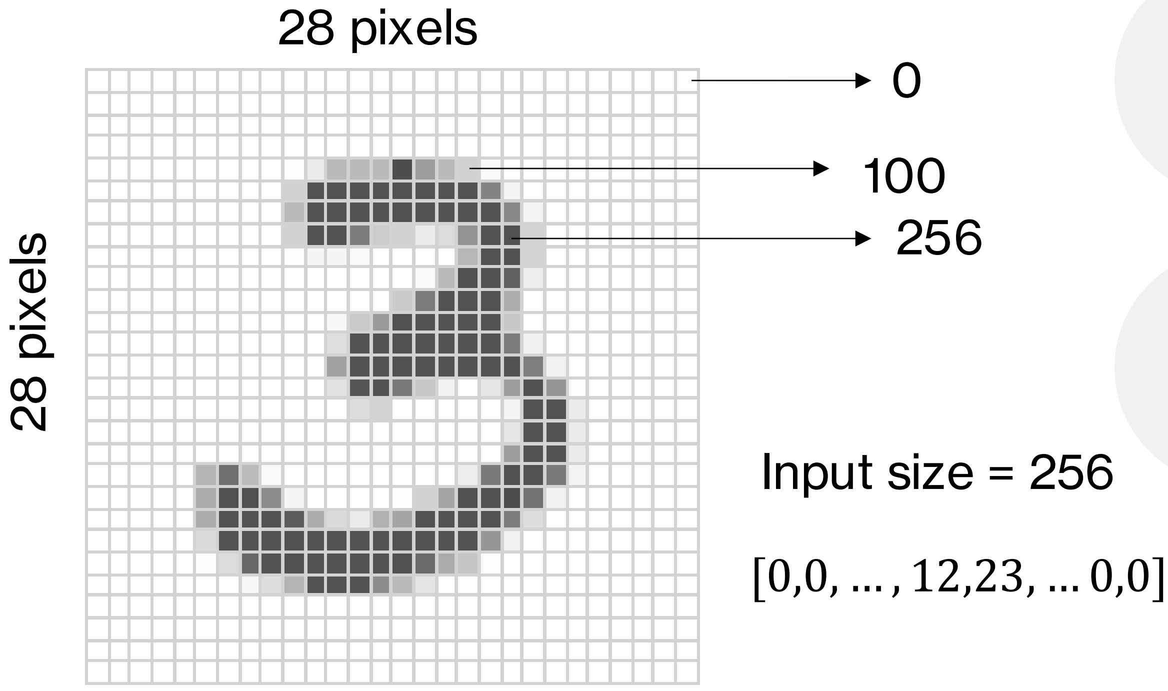




Transformation of simple image to vector

Neural Networks: Architecture

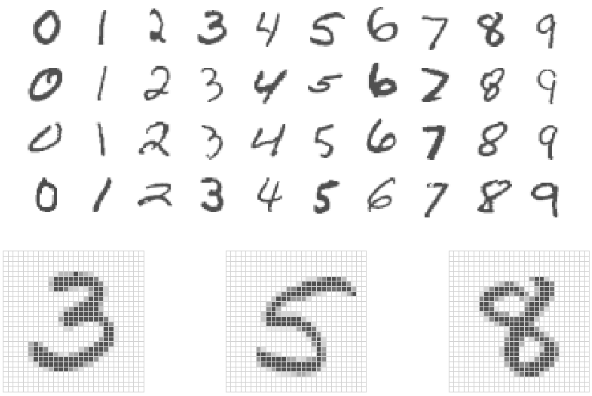
- Let's consider an example of neural network architecture… at scale..

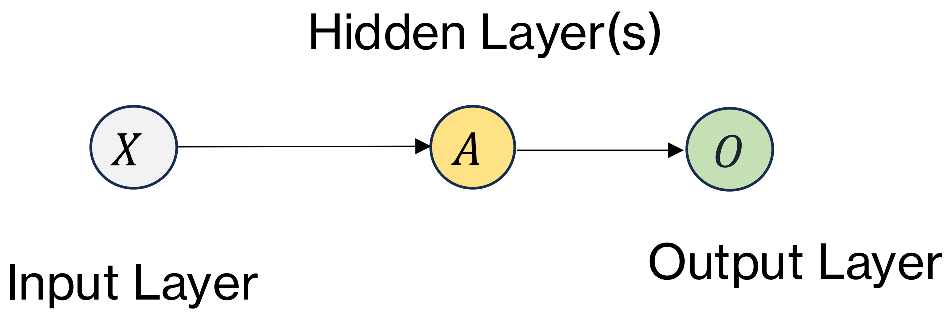


Transformation of simple image to vector

Neural Networks: Architecture

- To design architecture
  - We need a input layer.
    - Size: ??



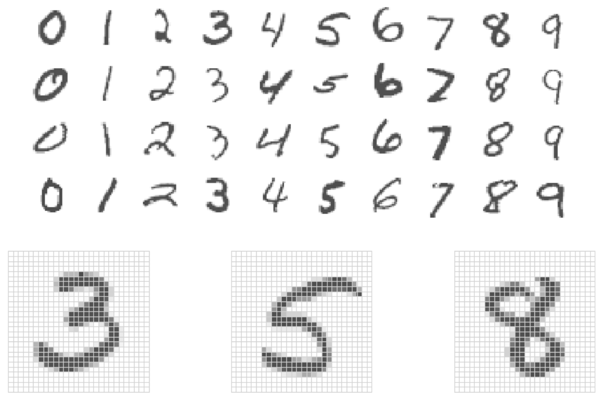


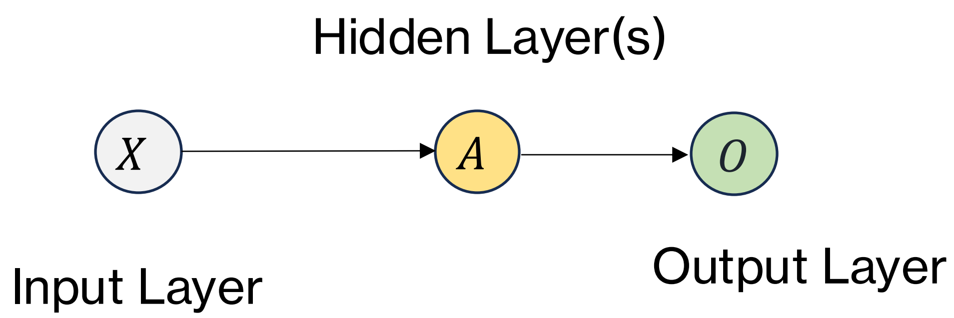
Neural Networks: Architecture

- Design Considerations:
  - Input Layer:
    - The size of the input layer depends on the type of neural network (e.g. 256x1 or 28x28)

Neural Networks: Architecture

- To design architecture
  - We need a input layer.
    - Size: 256 units
  - We need output layers.
    - Size: ??



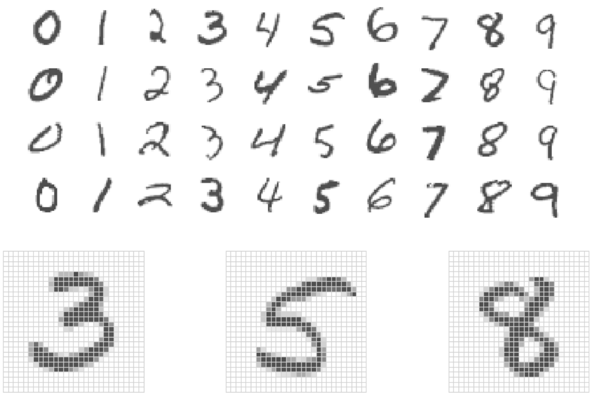


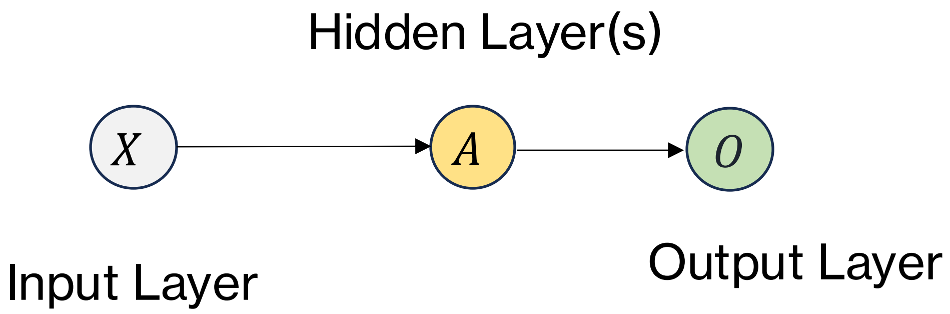
Neural Networks: Architecture

- Design Considerations:
  - Input Layer:
    - The size of the input layer depends on the type of neural network (e.g. 256x1 or 28x28)
  - Output Layer:
    - The choice of cost function depends upon cost function. For instance, if we use cross entropy loss, the output must be qualitative.
    - **Linear** unit: With no nonlinearity, they can be of the form $y' = W^T h + b$ .They are easy to work with.
    - **Sigmoid** unit: For predicting the value of a binary variable e.g., classification with 2 classes.
    - **SoftMax** unit: For representing a probability distribution over a discrete variable ($n > 2$ classes)

Neural Networks: Architecture

- To design architecture
  - We need a input layer.
    - Size: 256 units
  - We need output layers.
    - Size: 10 (0-9) units
  - We need hidden layers.
    - Size: ??
    - Activation Function:??





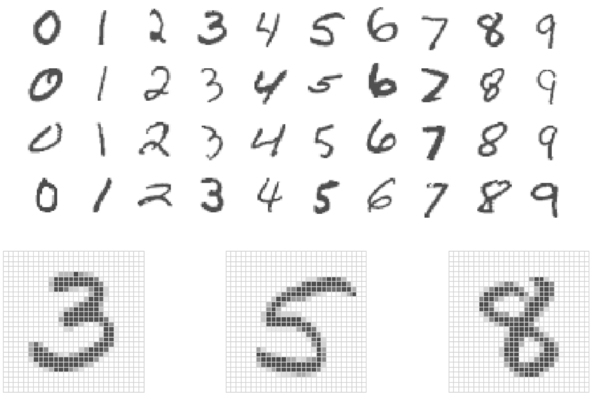
Neural Networks: Architecture

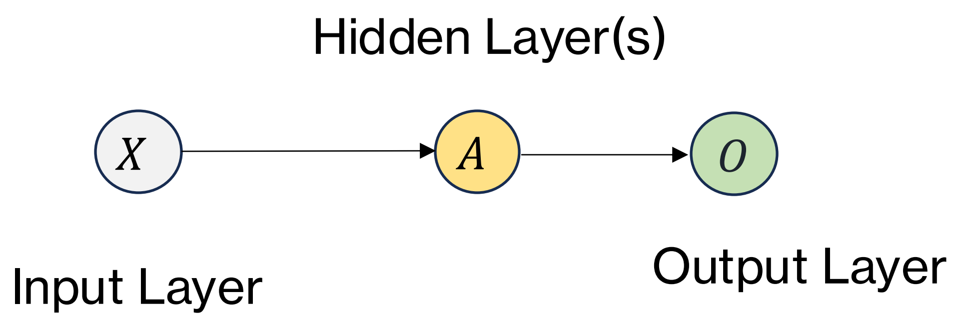
- To design architecture
  - We need a input layer.
    - Size: 256
  - We need output layers.
    - Size: 10 (0-9)
  - We need hidden layers.
    - Size: L1 = 256 , L2 = 128
    - Activation Function: ReLu

$$g(z) = (z)_+ = \begin{cases} 0 & \text{if } z < 0 \\ z & \text{otherwise.} \end{cases}$$

General Relu can be written as:

$$h_i = g(\boldsymbol{z}, \boldsymbol{\alpha})_i = \max(0, z_i) + \alpha_i \min(0, z_i)$$





Neural Networks: Architecture

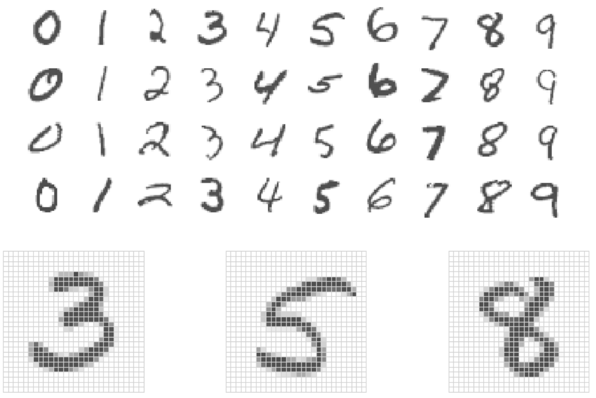
- To design architecture
  - We need a input layer.
    - Size: 256
  - We need output layers.
    - Size: 10 (0-9)
  - We need hidden layers.
    - Size: L1 = 256 , L2 = 128
    - Activation Function: ReLu

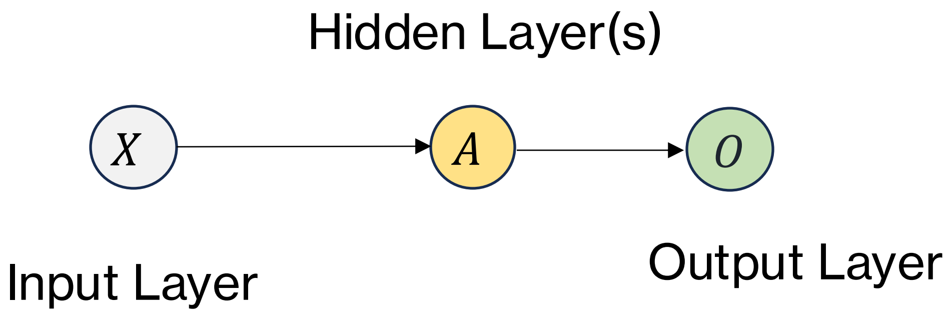
$$g(z) = (z)_+ = \begin{cases} 0 & \text{if } z < 0 \\ z & \text{otherwise.} \end{cases}$$

General Relu can be written as:

$$h_i = g(\boldsymbol{z}, \boldsymbol{\alpha})_i = \max(0, z_i) + \alpha_i \min(0, z_i)$$

Absolute value rectification: $\alpha_i = -1$





Neural Networks: Architecture

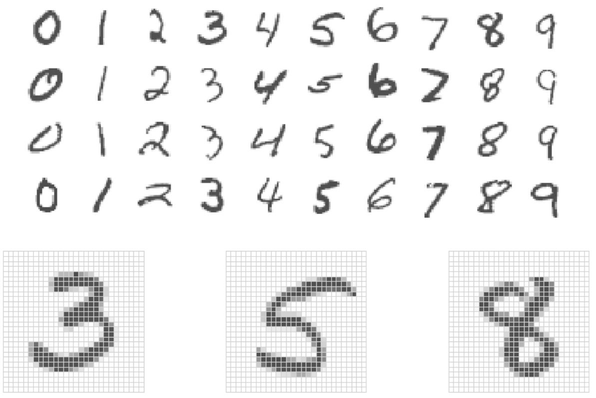
- To design architecture
  - We need a input layer.
    - Size: 256
  - We need output layers.
    - Size: 10 (0-9)
  - We need hidden layers.
    - Size: L1 = 256 , L2 = 128
    - Activation Function: ReLu

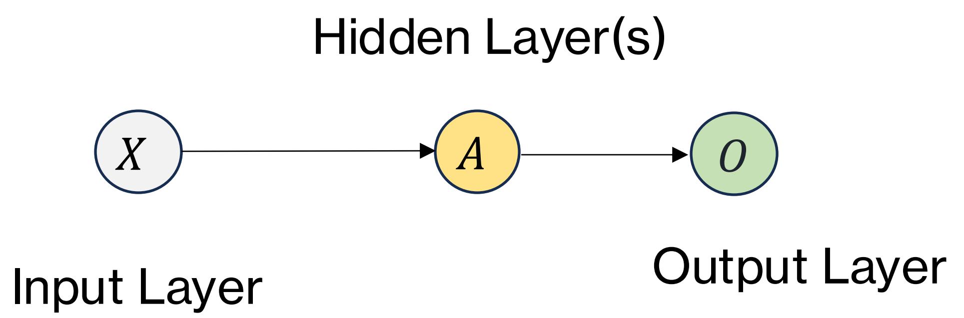
$$g(z) = (z)_+ = \begin{cases} 0 & \text{if } z < 0 \\ z & \text{otherwise.} \end{cases}$$

General Relu can be written as:

$$h_i = g(\boldsymbol{z}, \boldsymbol{\alpha})_i = \max(0, z_i) + \alpha_i \min(0, z_i)$$

Leaky Relu: $\alpha_i = 0.01$





Neural Networks: Architecture

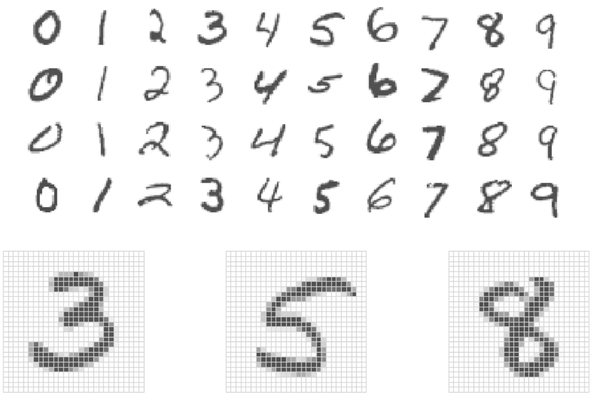
- To design architecture
  - We need a input layer.
    - Size: 256
  - We need output layers.
    - Size: 10 (0-9)
  - We need hidden layers.
    - Size: L1 = 256 , L2 = 128
    - Activation Function: ReLu

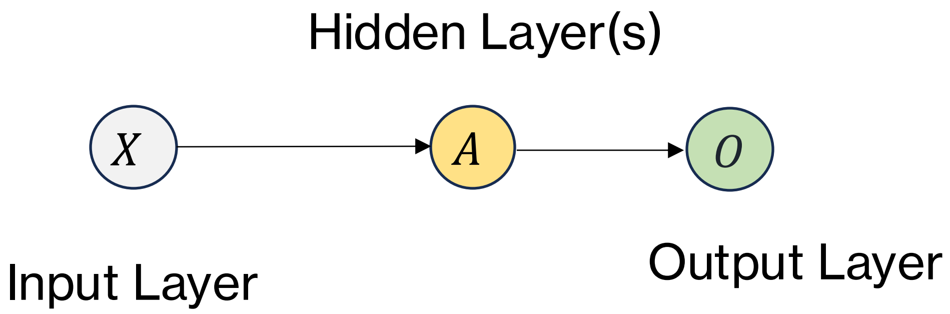
$$g(z) = (z)_+ = \begin{cases} 0 & \text{if } z < 0 \\ z & \text{otherwise.} \end{cases}$$

General Relu can be written as:

$$h_i = g(\boldsymbol{z}, \boldsymbol{\alpha})_i = \max(0, z_i) + \alpha_i \min(0, z_i)$$

Parametric Relu: α_i is learned





Neural Networks: Architecture

- Design Considerations:
  - Input Layer:
    - The size of the input layer depends on the type of neural network (e.g. 256x1 or 28x28)
  - Output Layer:
    - The choice of cost function depends upon cost function. For instance, if we use cross entropy loss, the output must be qualitative.
    - Linear unit: With no nonlinearity, they can be of the form $y' = W^T h + b$ .They are easy to work with.
    - Sigmoid unit: For predicting the value of a binary variable e.g., classification with 2 classes.
    - SoftMax unit: For representing a probability distribution over a discrete variable ($n > 2$ classes)
  - Hidden Layer:
    - The design of hidden units is does not have well defined guiding theoretical principles.
    - Too many hidden layers for a simple problem (or vice versa) is a bad design choice.
    - Experimenting with hidden layers is very common.
    - Using variation of Rectified Linear Units requires visualization of outputs.

Readings

Required Readings:

Introduction to Statistical Learning

- Chapter 10 – Section 10.1 and 20.2 page 400 - 406

***Supplemental Readings*:**

Deep Learning

- Chapter 6 – page 168 - 224

Thank You

McMaster University