Modul 1

Purwadhika
Digital Technology School

Artificial Intelligence

Introduction to Neural Networks

Job Connector Program

Outline

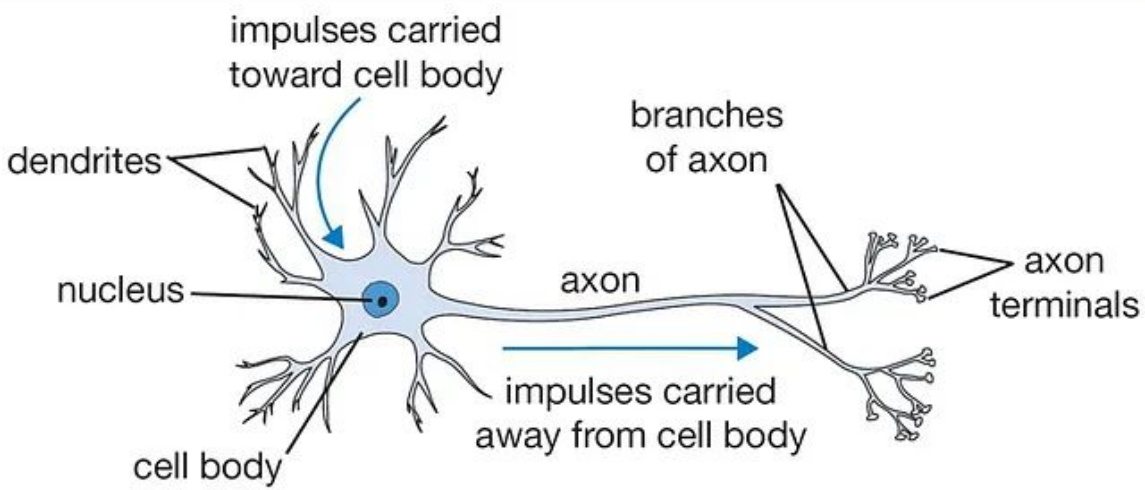
- Neural Network Definition, Purpose, and Applications
- Exploring the Structure, Concept, and How It Works
- Hands On Neural Networks with Python

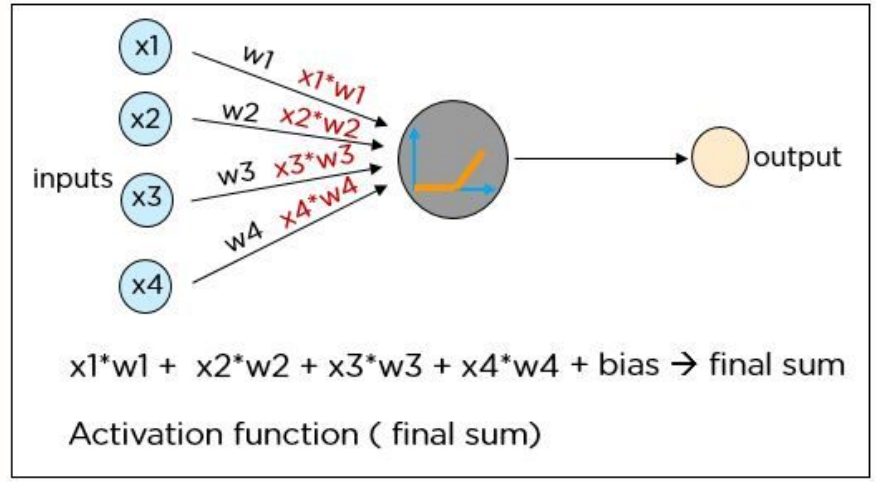
Purwadhika
Digital Technology School

Neural Networks

Definition, Basic Concept, and Applications

Biological Inspiration of Neural Networks





biological neuron

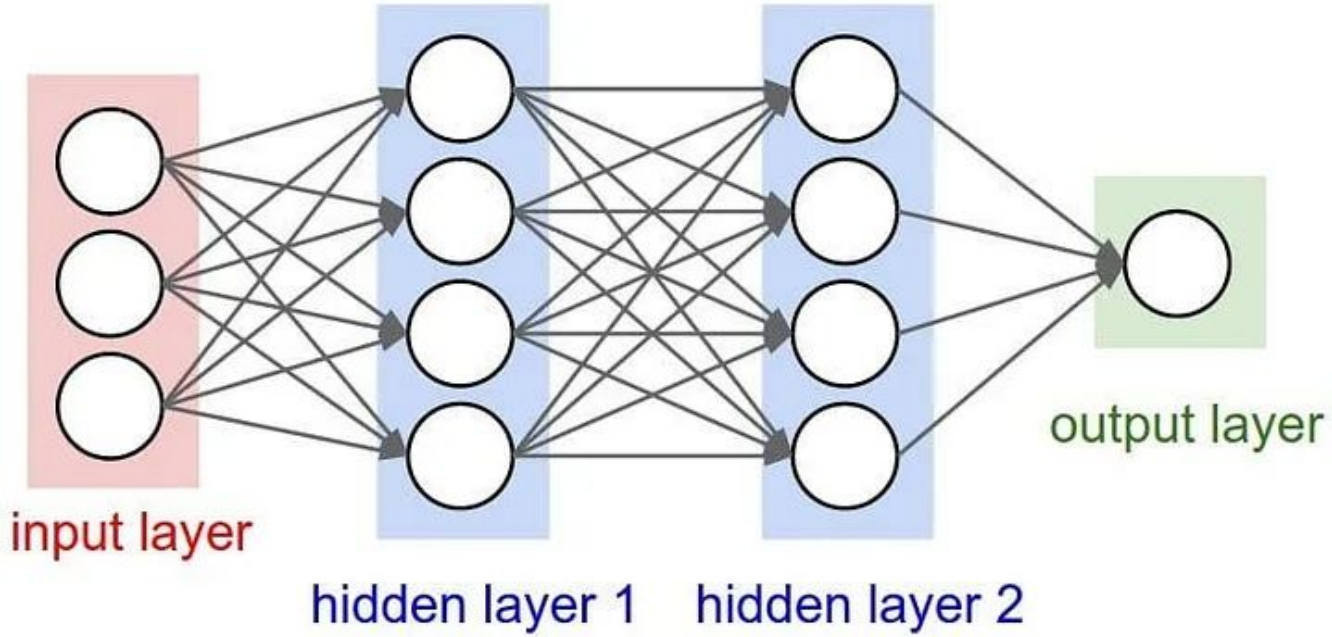
mathematical model

About Neural Networks

Purwadhika Digital Technology School

- A Neural Network is a computational model inspired by the human brain, consisting of layers of interconnected nodes (or neurons) that process data by adjusting the strength of connections (weights) through learning.
- It is designed to recognize patterns and relationships in data, making it useful for tasks such as classification, regression, and image or speech recognition.
- Neural networks learn from examples, improving their performance over time without being explicitly programmed with rules.

Basic Structure of Neural Networks

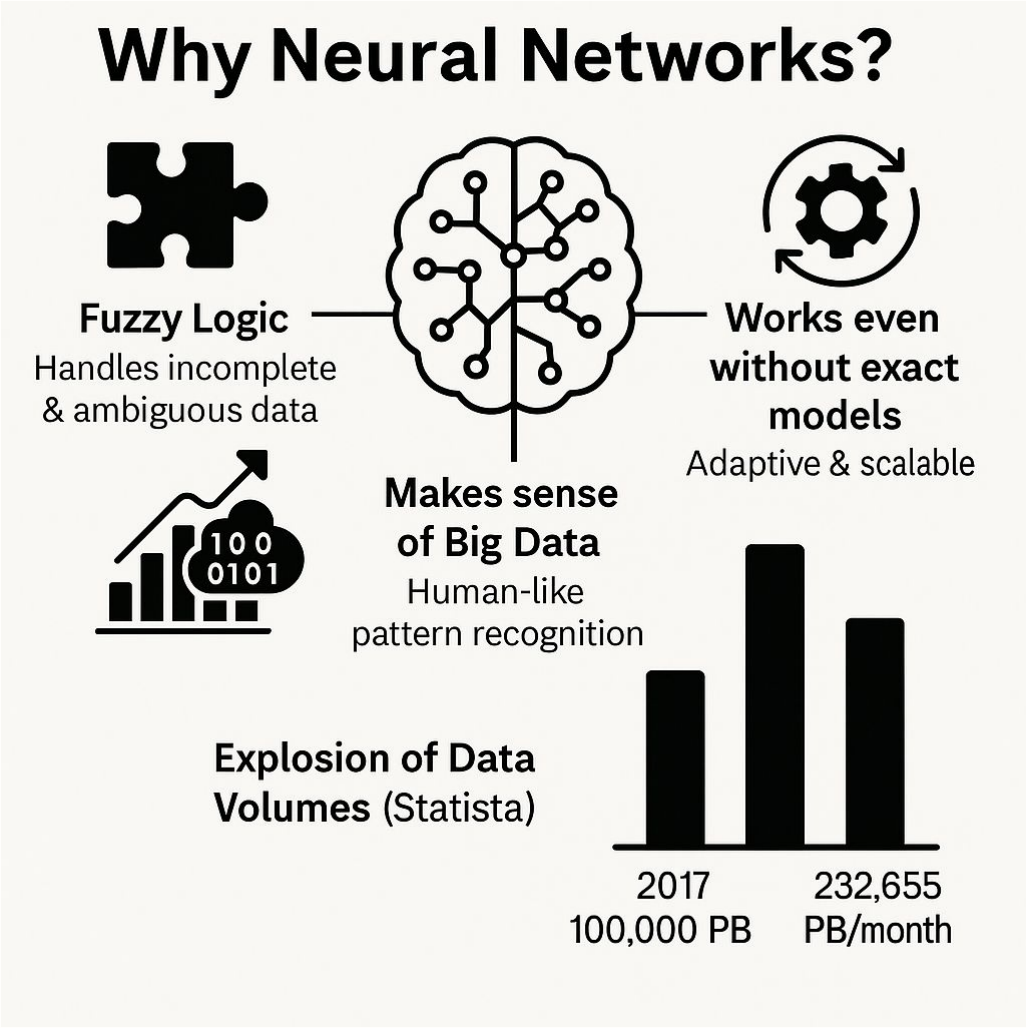


Input Layer: Takes in the raw data and passes it to the next layers.

Hidden Layers: Process the data to find useful patterns.

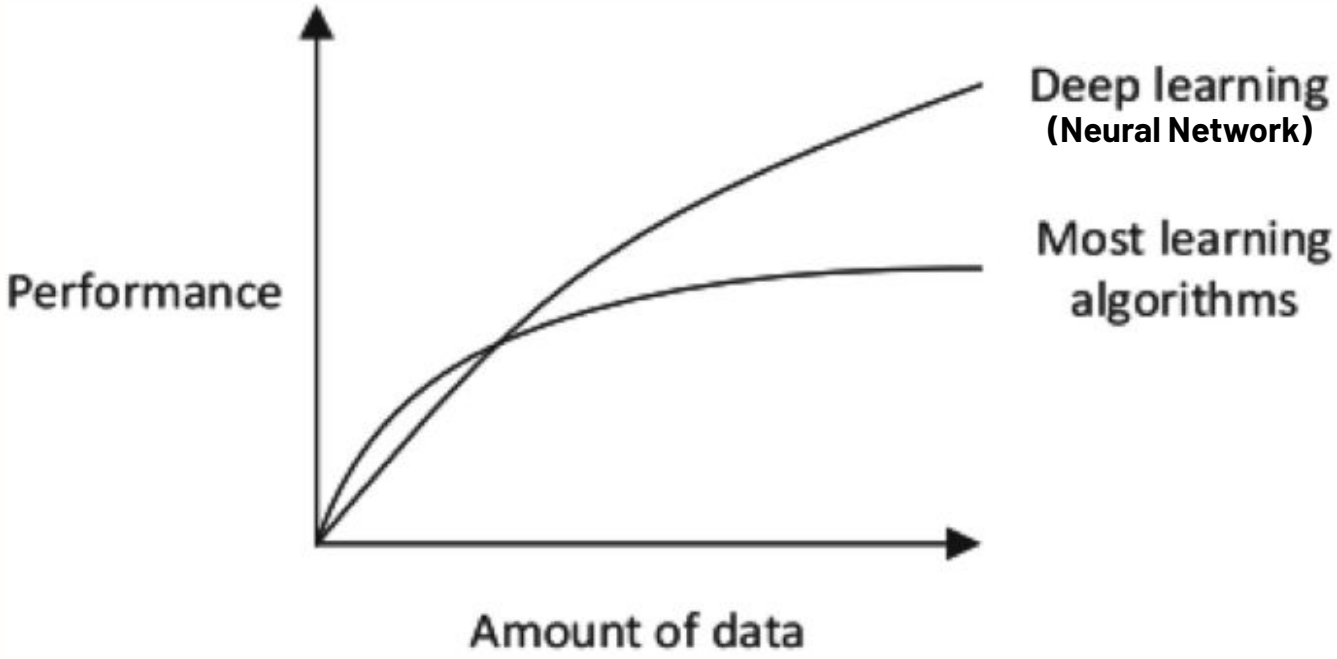
Output Layer: Gives the final prediction or result.

Why using Neural Networks?



- Mimics the way the human brain works
- Capable of handling ambiguous & incomplete data
- Works even without a definite model
- Effectively processes big data
- Global data continues to increase drastically

Why using Neural Networks? (cont'd)



Neural Networks Applications

Purwadhika Digital Technology School

1 Finance & Banking
- stock forecast
- fraud detection
- credit scoring

2 E-Commerce
- recommendation system
- forecasting product sales
- product classification

3 Healthcare
- medical diagnosis
- drug discovery

4 Social Media
- sentiment analysis
- social network analysis
- topic modeling

5 Transportation
- self-driving cars
- traffic optimization
- predictive maintenance for vehicles.

6 Entertainment
- content recommendation
- image recognition in video games

4 Manufacturing
- quality control (visual inspection)
- predictive maintenance
- supply chain optimization

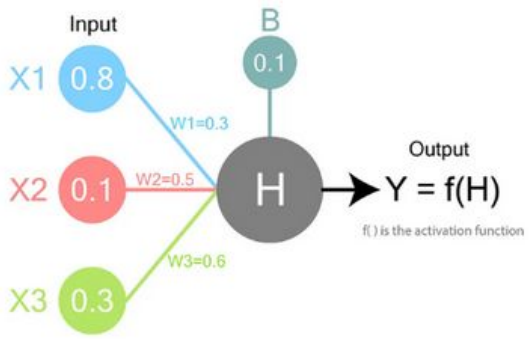
5 Retail
- inventory management
- customer segmentation
- personalized recommendations

6 Education
- personalized learning
- automated grading and assesment

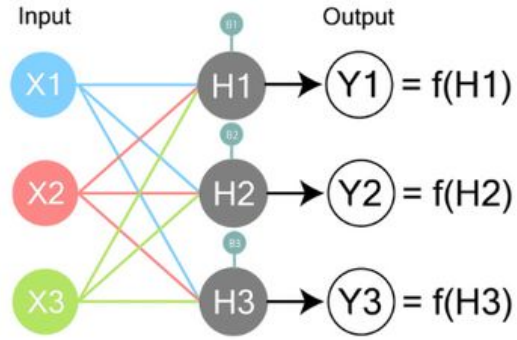
Neural Network to Deep Learning

Purwadhika Digital Technology School

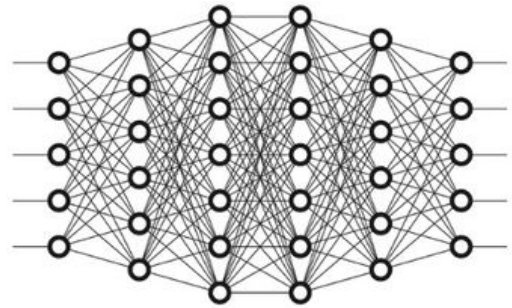
single perceptron



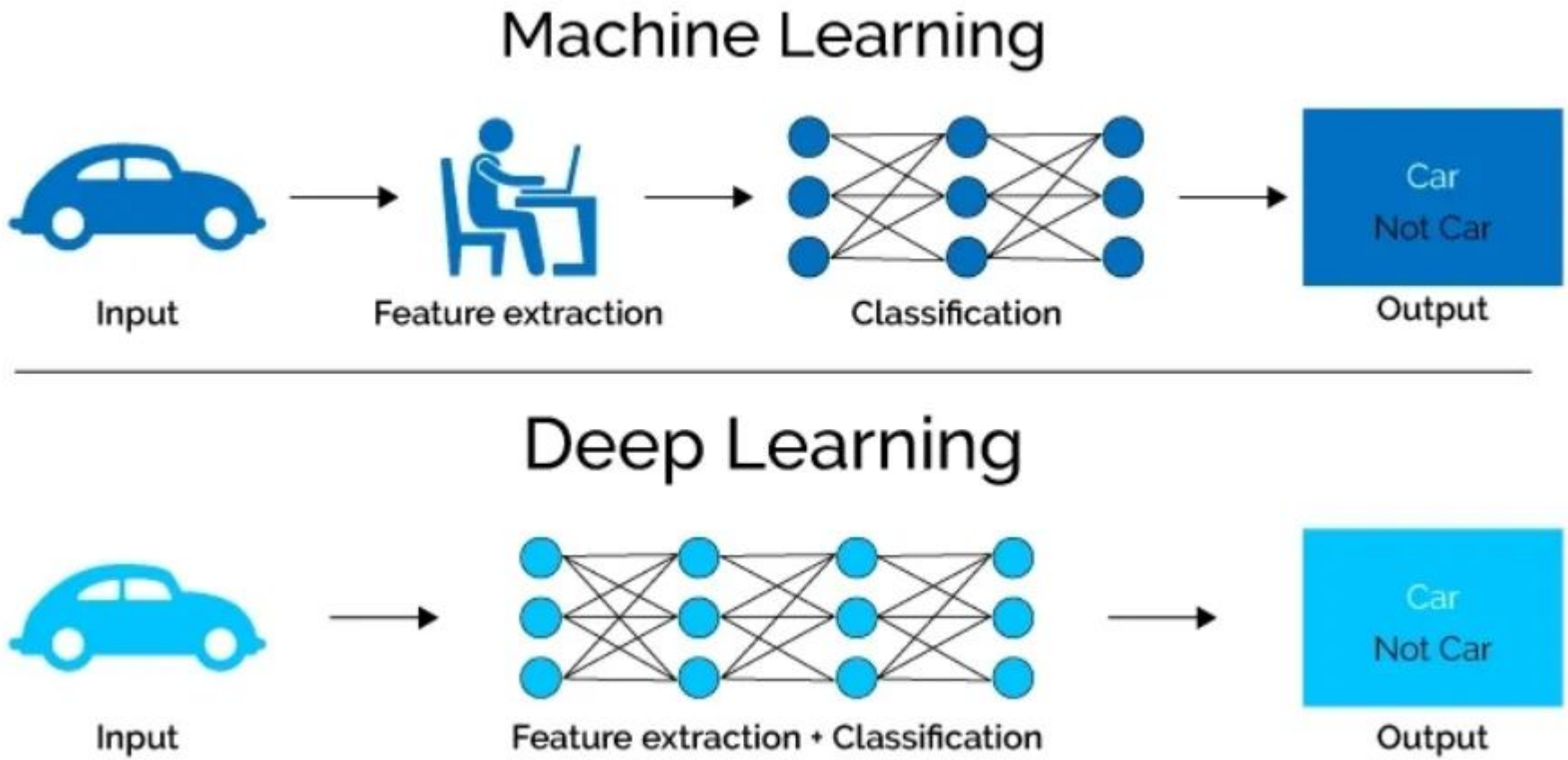
multi layer perceptron



deep learning

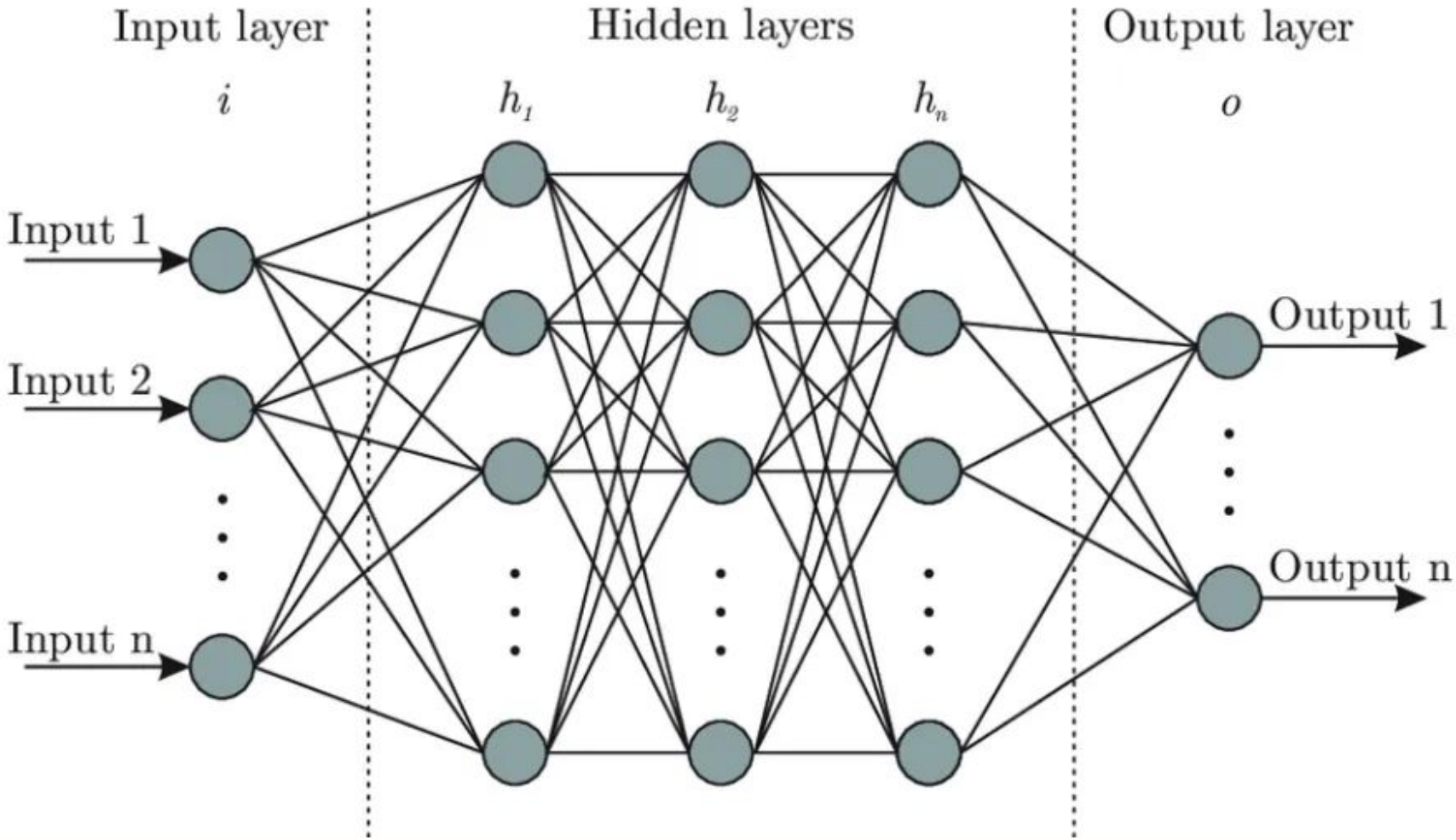


What's the Difference between ML and DL?



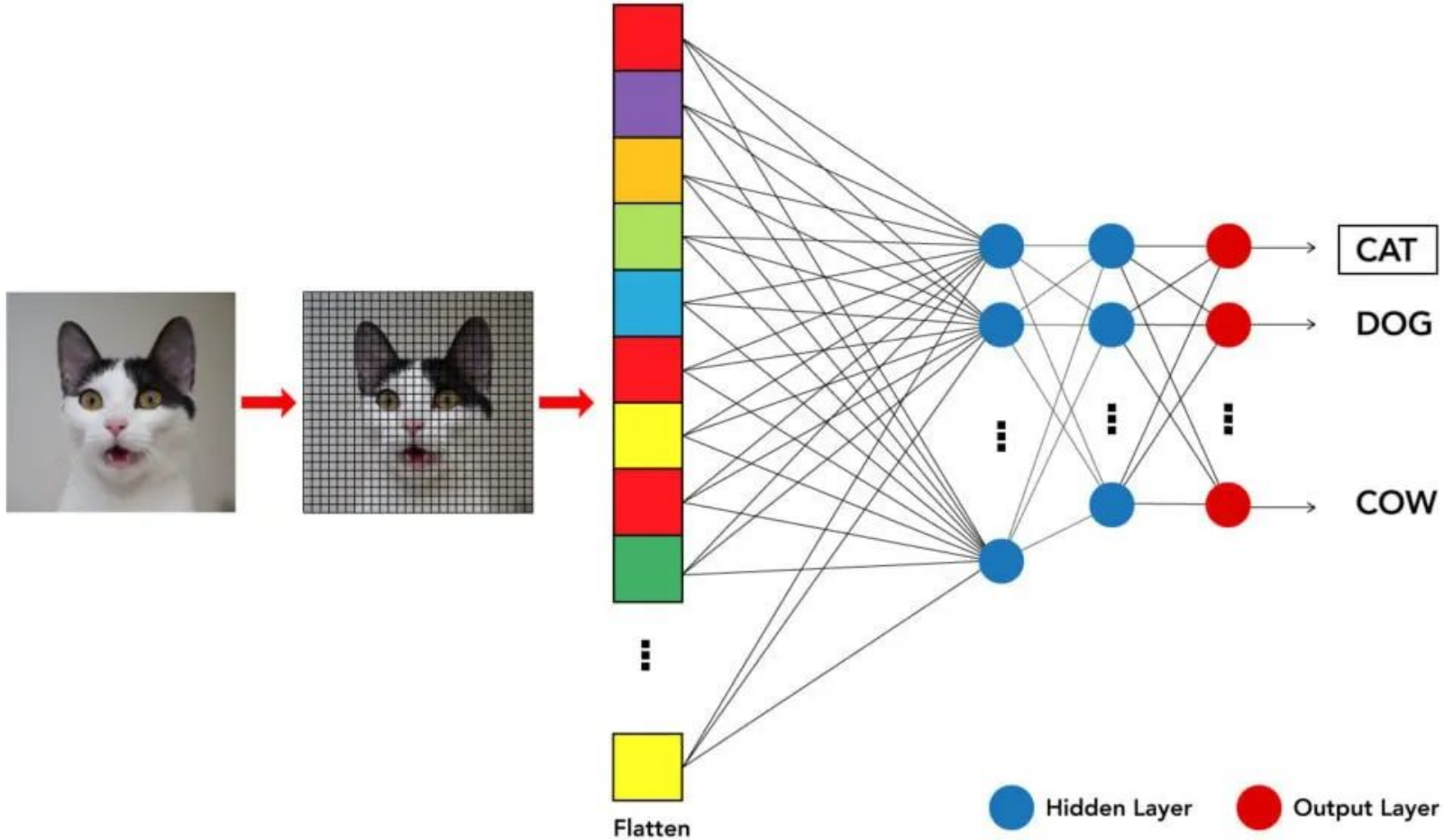
How Neural Networks Learns Numerical Data

Purwadhika Digital Technology School



How Neural Networks Learns Image Data

Purwadhika Digital Technology School



Purwadhika
Digital Technology School

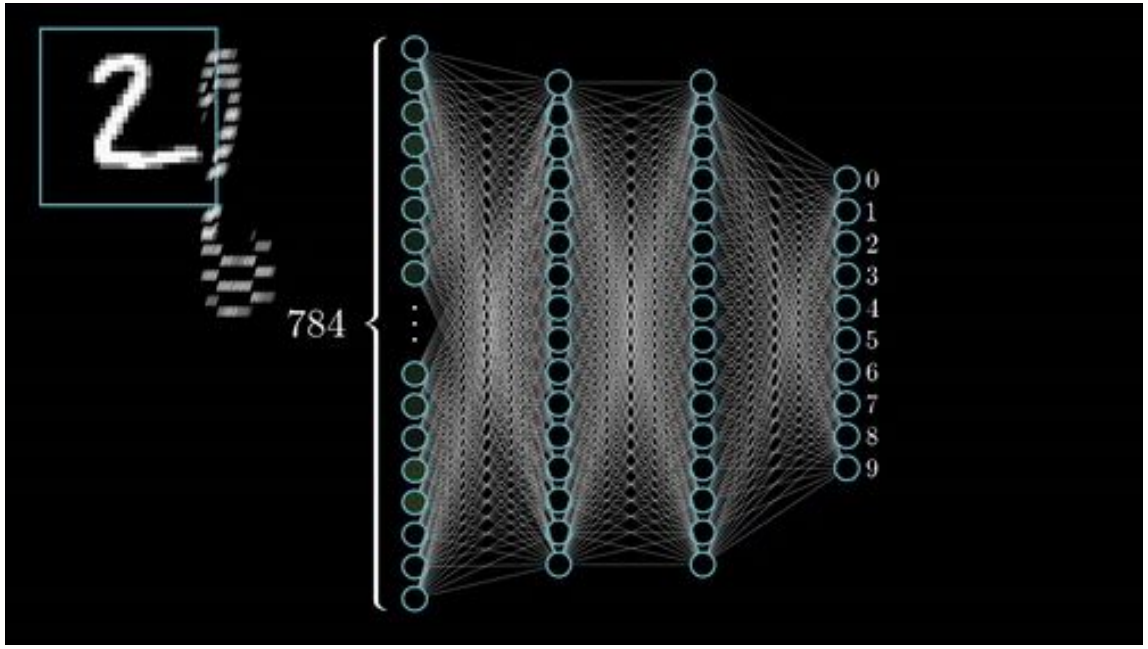
Neural Networks

Exploring the Structure, Concept, and How It Works

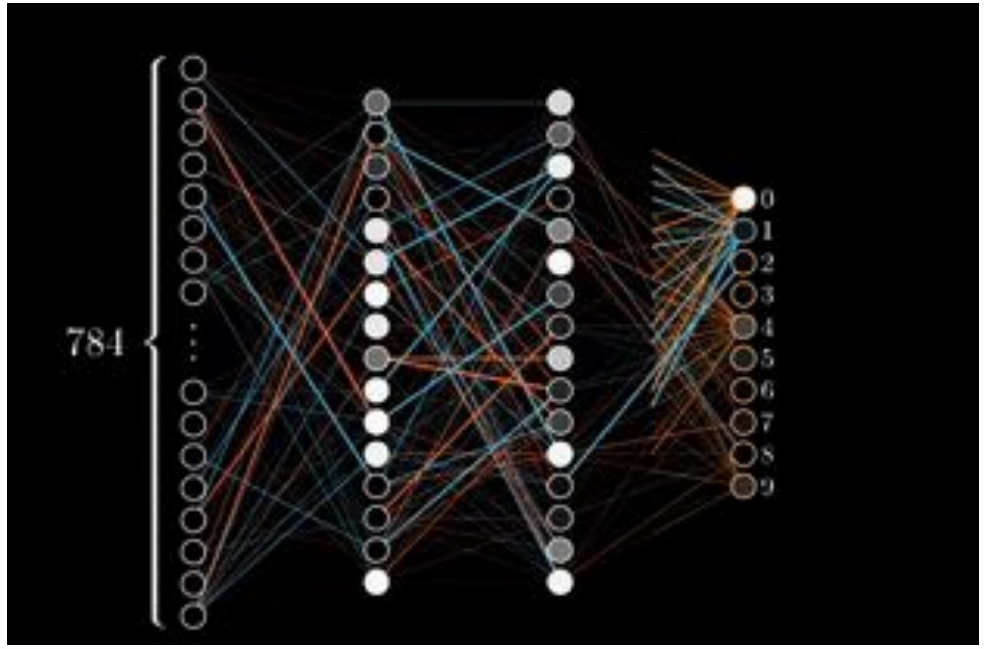
Feedforward and Backpropagation

Purwadhika Digital Technology School

Feedforward



Backpropagation

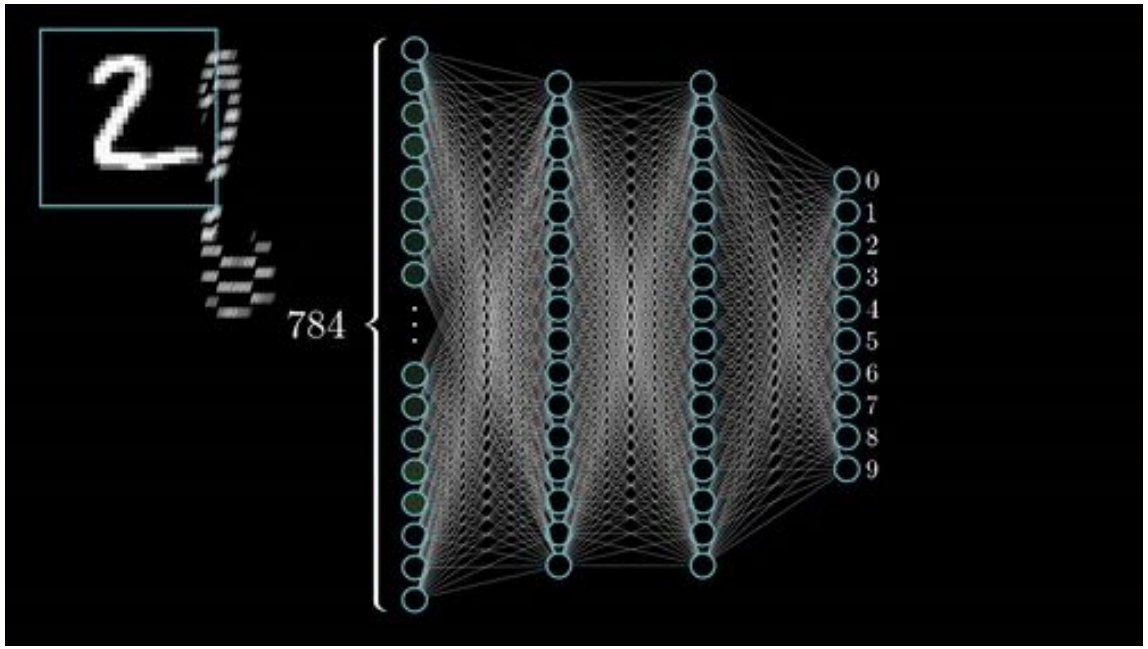


Purwadhika
Digital Technology School

Neural Networks

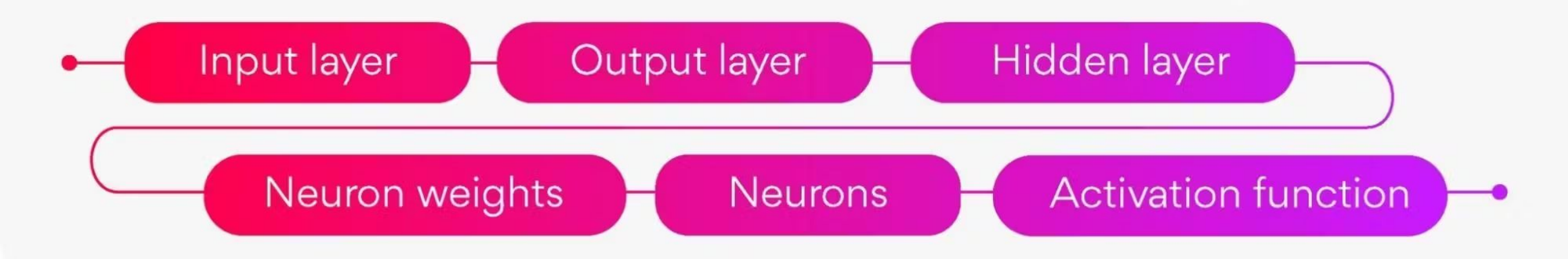
Layers of Feed Forward Neural Network

Feedforward Neural Network

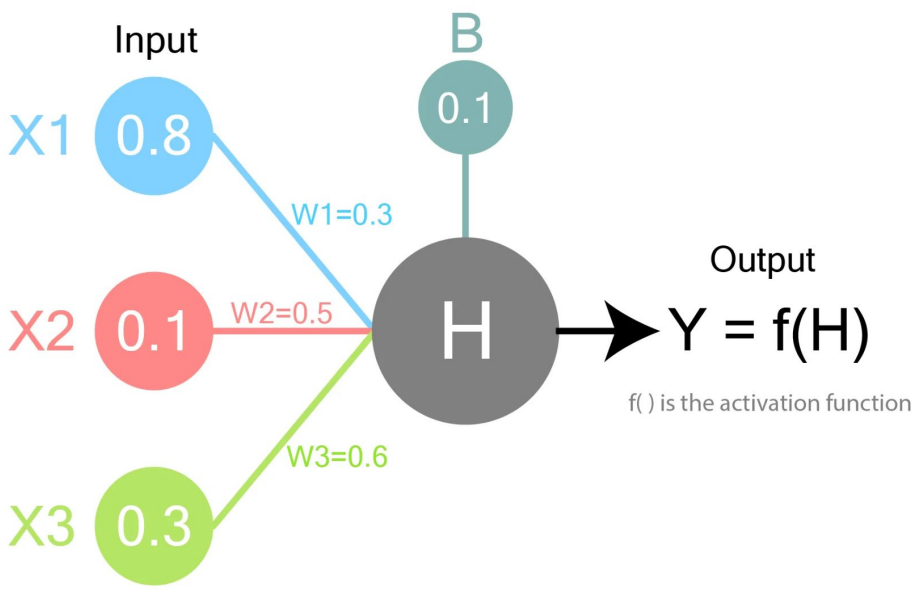


Feedforward neural network is an artificial neural network where connections moving the information in only one direction, forward, from the input nodes, through the hidden nodes (if any) and to the output nodes.

Feedforward Neural Network Layers



Main Components from Perceptron



$$H = X1*W1 + X2*W2 + X3*W2 + B$$

$$Y = f(H)$$

- **Y** is the final value of the node.
- **W** represents the weights between the nodes in the previous layer and the output node.
- **X** represents the values of the nodes of the previous layer.
- **B** represents bias, which is an additional value present for each neuron. Bias is essentially a weight without an input term. It's useful for having an extra bit of adjustability which is not dependant on previous layer.
- **H** is the intermediate node value. This is not the final value of the node.
- **f( )** is called an Activation Function and it is something we can choose.

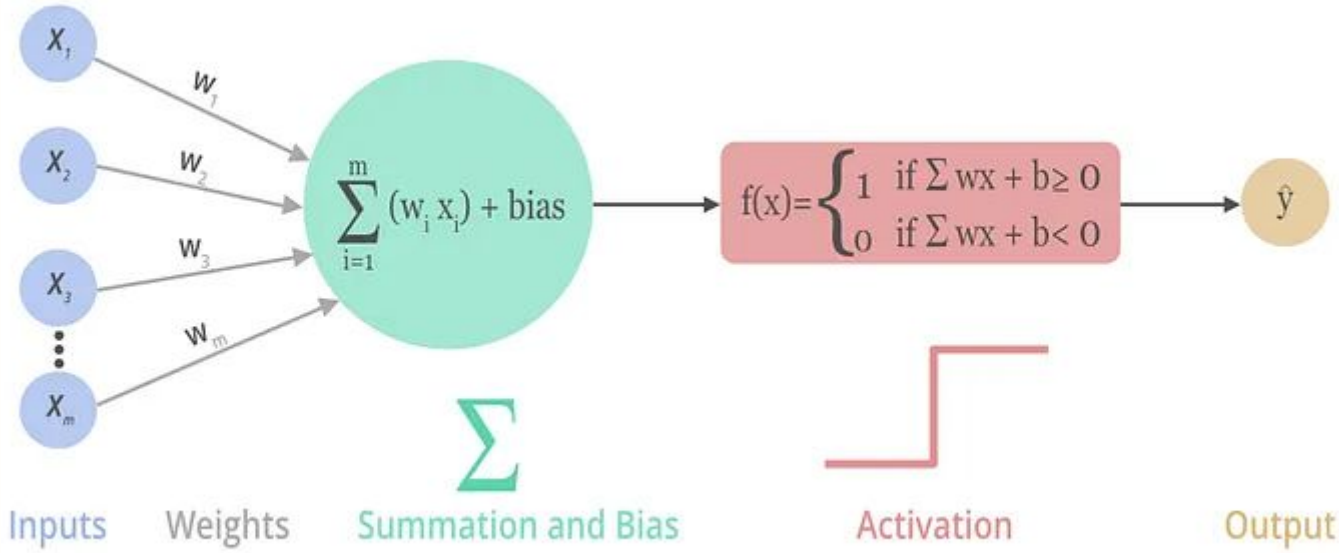
Main Components from Feedforward Process

Purwadhika Digital Technology School

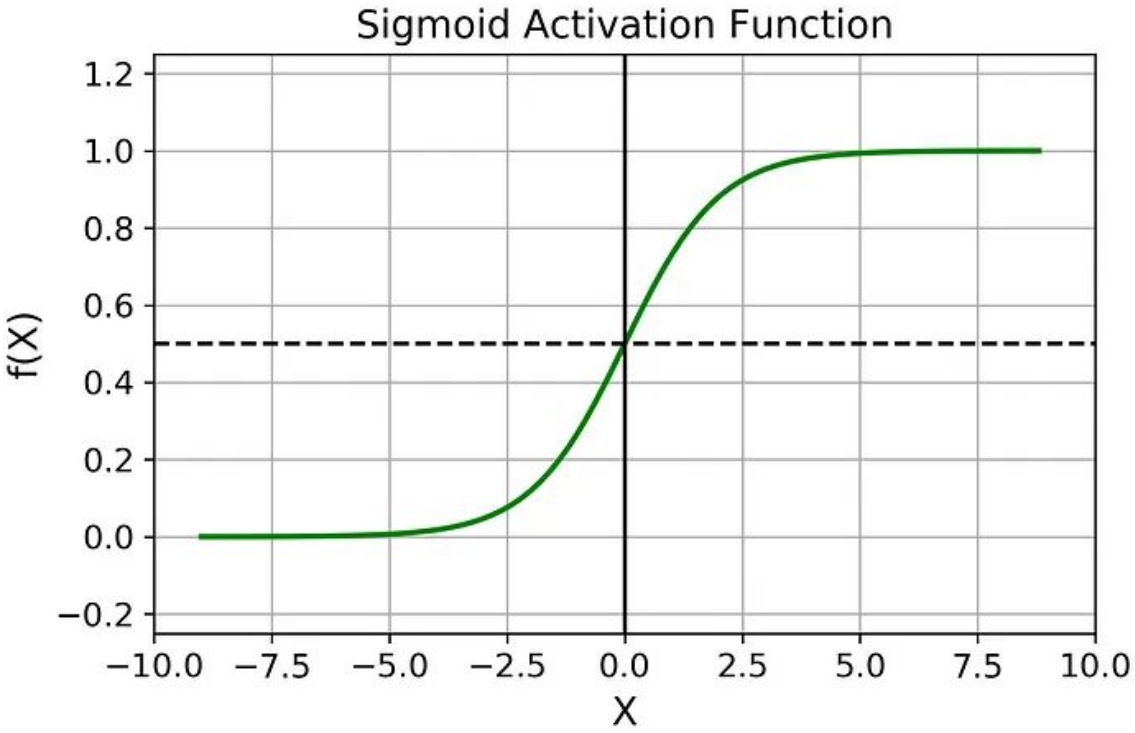
- **Input Layer**: Receives raw data and passes it to the next layers. The number of neurons matches the number of input features.
- **Output Layer**: Produces the final prediction based on the model's objective (e.g., classification or regression).
- **Hidden Layer**: Sits between input and output layers, often stacked in multiple layers to process data through transformations and learn complex patterns.
- **Neuron Weights**: Weights determine the strength of connections between neurons, similar to coefficients in linear regression. They typically range between 0 and 1 and are adjusted during training.
- **Neurons**: Modeled after biological neurons, they compute weighted input sums and apply activation functions (linear or nonlinear) to process data. Neurons and their weights are key to learning in neural networks.

Activation Function

An activation function is a function that is added to an artificial neural network in order to help the network learn complex patterns in the data.

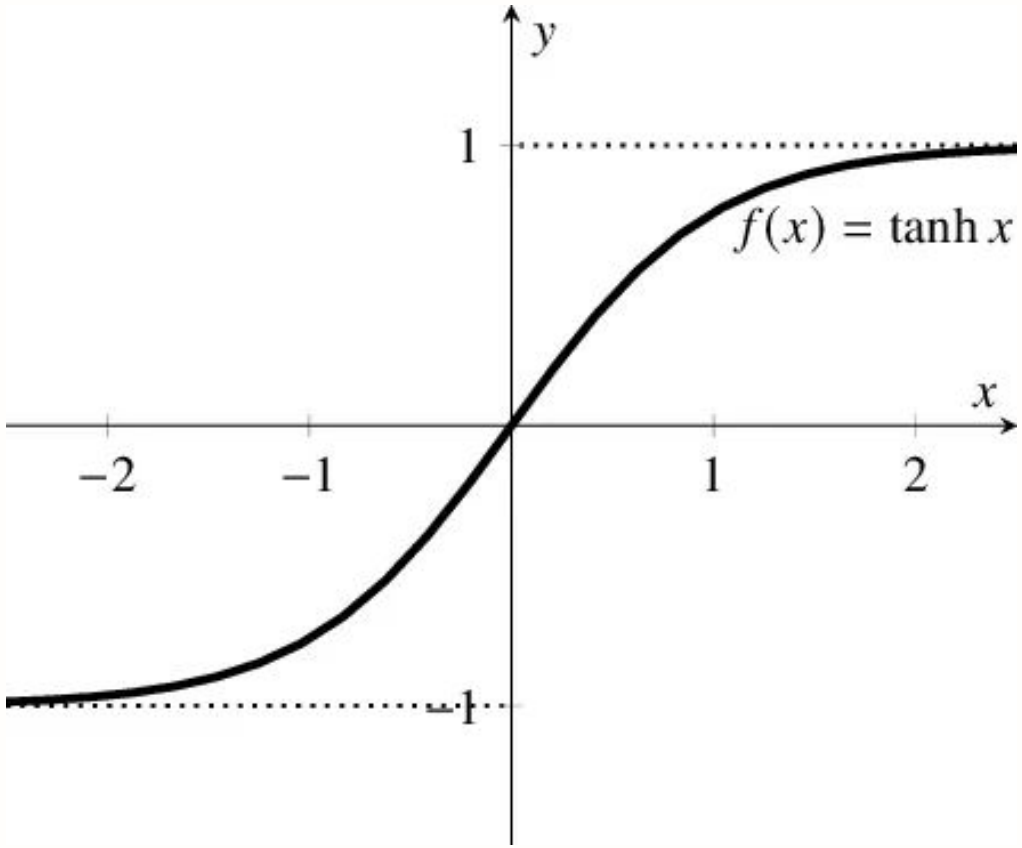


Sigmoid Activation Function



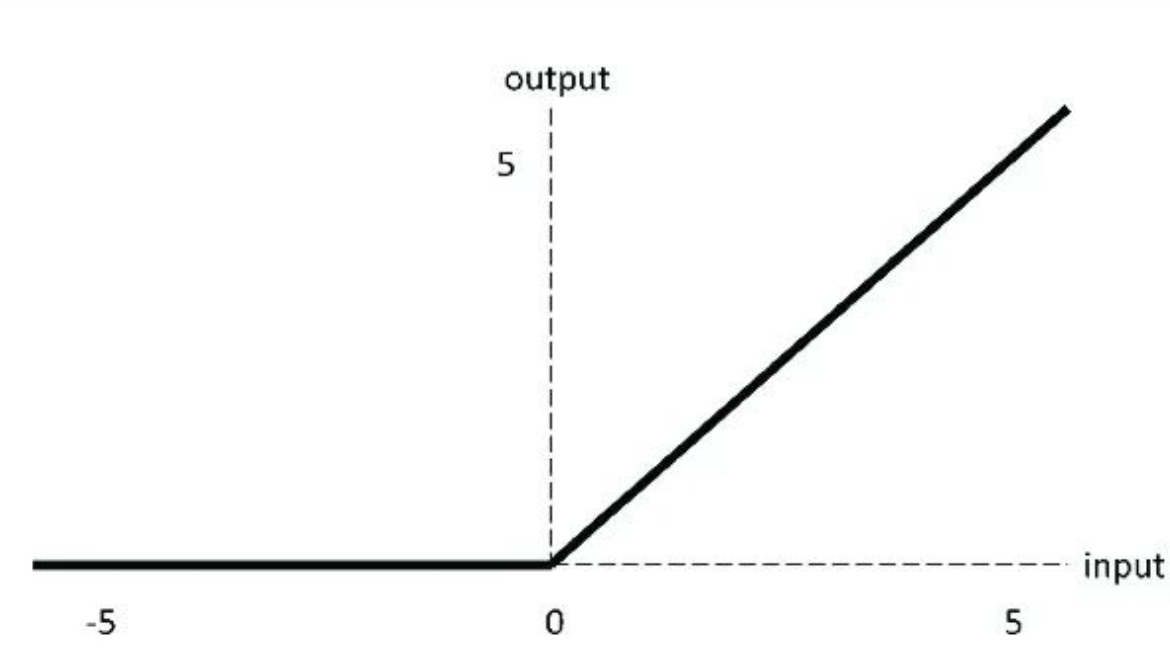
- **Formula** : f(x) = 1 / (1 + e^(-x))
- **When to Use**: Commonly used in binary classification problems, especially in the output layer to produce probabilities between 0 and 1. However, it can suffer from vanishing gradient issues, making it less suitable for deep networks.

Tanh or Hyperbolic Tangent Activation Function



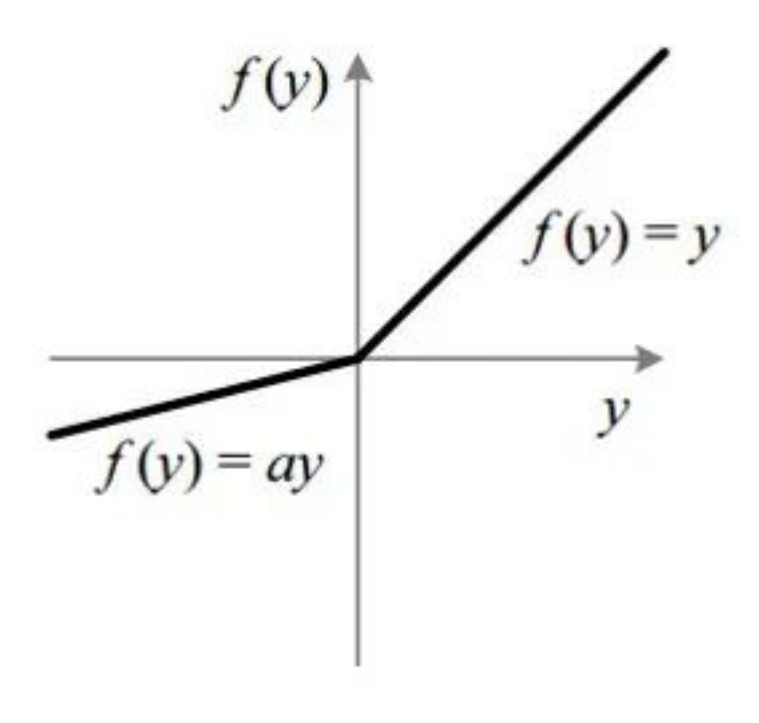
- **Formula** : f(x) = (e^x - e^(-x)) / (e^x + e^(-x))
- **When to Use**: Preferred over sigmoid in hidden layers as it outputs values between -1 and 1, centering the data and often leading to faster convergence. Still, it can also face vanishing gradient problems in deep networks.

ReLU (Rectified Linear Unit) Activation Function



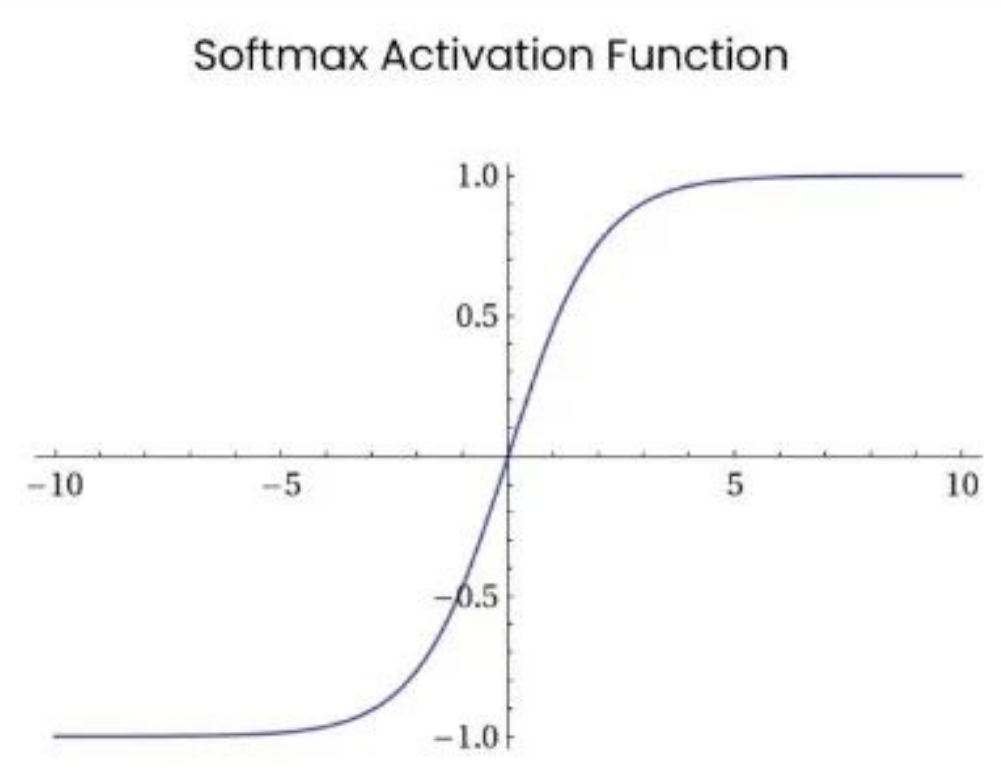
- **Formula** : f(x) = max(0, x)
- **When to Use**: Widely used in hidden layers of deep neural networks due to its simplicity and efficiency. It helps mitigate the vanishing gradient problem. However, neurons can "die" during training if they output zero consistently.

Leaky ReLU Activation Function



- **Formula** : f(x) = x if x > 0; αx if x ≤ 0 (where α is a small constant, e.g., 0.01)
- **When to Use**: Addresses the "dying ReLU" problem by allowing a small, non-zero gradient when the unit is not active. Useful in deep networks to ensure neurons continue learning.

Softmax



- **Formula** : f(x_i) = e^(x_i) / Σ e^(x_j) for j in all classes
- **When to Use**: Used in the output layer for multi-class classification problems to produce a probability distribution over classes.

Function in Feed Forward Neural Network

Purwadhika Digital Technology School

- **Loss Function**

The loss function is a mathematical expression that evaluates how well the neural network's prediction matches the true output for a single training example. It guides the learning process by providing feedback on prediction errors, which are minimized during training through optimization techniques like gradient descent.

cross-entropy loss for binary classification

$$L_i = -[y_i \log(\hat{p}_i) + (1 - y_i) \log(1 - \hat{p}_i)]$$

cross-entropy loss for multiclass classification

$$L(\Theta) = -\sum_{i=1}^{k} y_i \log(\hat{y}_i)$$

Function in Feed Forward Neural Network (cont'd)

Purwadhika Digital Technology School

- **Cost Function**

The cost function is a mathematical formula that evaluates how well a neural network's predictions match the actual target values. It calculates the total error across all training examples, guiding the learning process by indicating how the model's parameters should be updated to improve accuracy.

cross-entropy loss for binary classification

$$J(\theta) = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{p}_i) + (1 - y_i)\log(1 - \hat{p}_i)]$$

- J(θ) = cost function
- N = amount of total samples
- yi = Target (0 or 1)
- pi = Predict probability model for class 1

Purwadhika
Digital Technology School

Neural Networks

Backpropagation Process

What is Backpropagation?

Purwadhika Digital Technology School

Backpropagation is an algorithm that makes an ANN learn based on the prediction error it makes. The algorithm can be summarized into three steps:

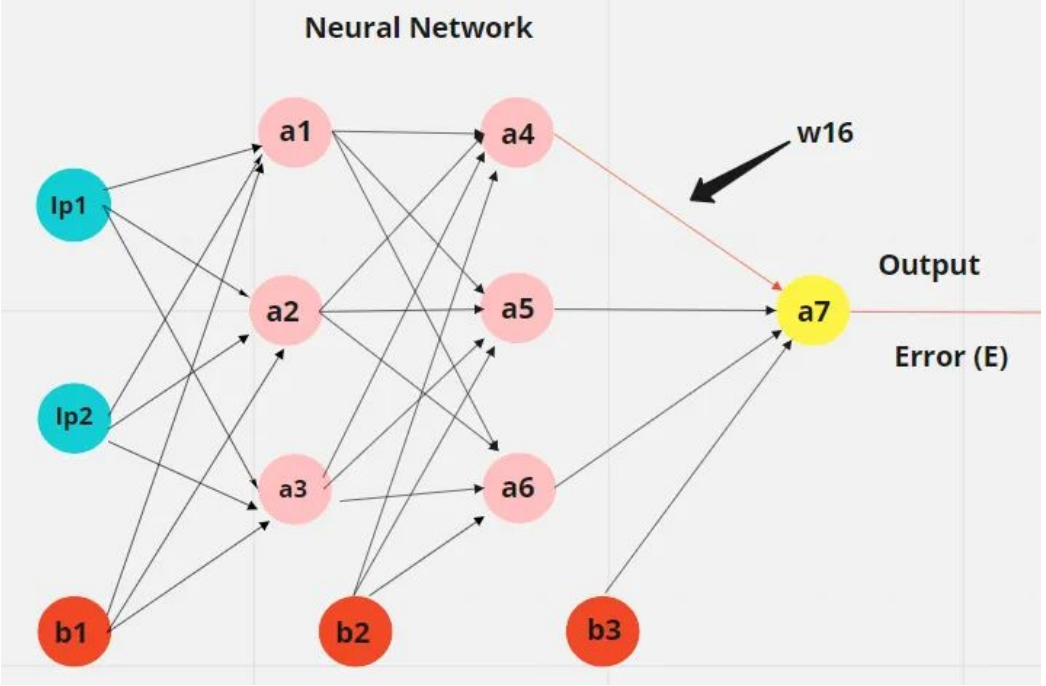
- Input data is passed forward through the network and the ANN makes a **prediction** based on its initial weights.
- The output of the network is then compared with the desired results (also called target values). The difference between the two is considered as an **error** of the network.
- The error is then passed backwards through the network and the weights of the neuron connections are **adjusted** depending on their influence on the error.

This way the backpropagation algorithm guarantees an approximation to the desired output with each learning iteration

How's Backpropagation Works?

Purwadhika Digital Technology School

1. Assume we have this network



2. Calculate the gradient of w16

$$\frac{\partial E}{\partial w16}$$

Derivative of loss

w.r.t w16

3. partial derivative of error by w16 with chain rule

$$\frac{\partial E}{\partial w16} = \frac{\partial E}{\partial y} * \frac{\partial y}{\partial net} * \frac{\partial net}{\partial w16}$$

The derivation of output w.r.t the affine

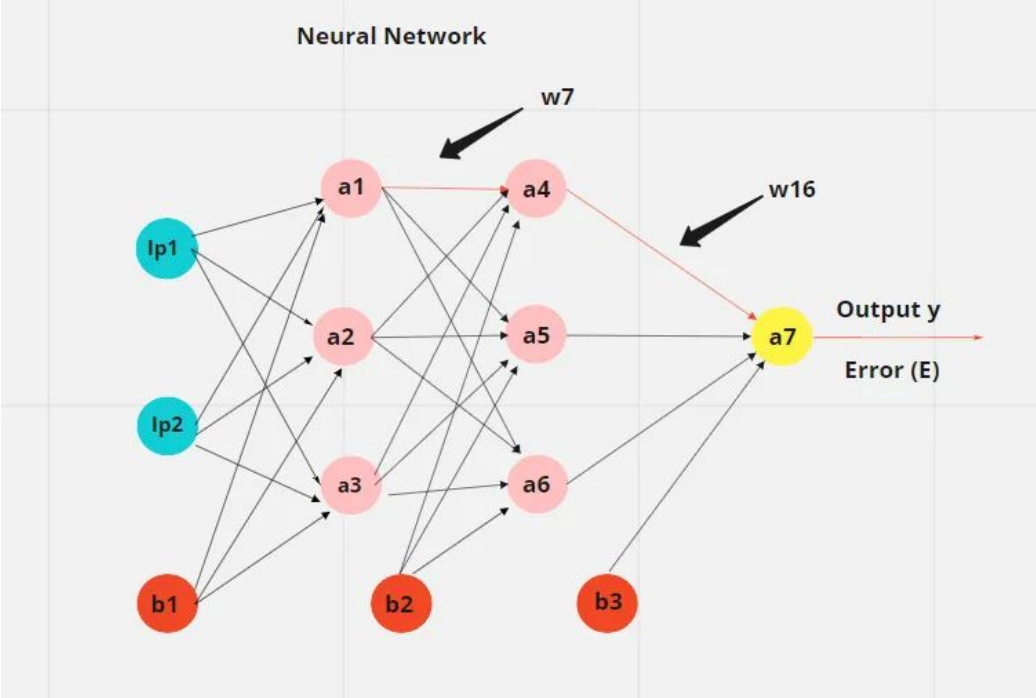
The derivative of error w.r.t output

derivative of affine w.r.t the weight

How's Backpropagation Works? (cont'd)

Purwadhika Digital Technology School

4. We repeat the same process for all the weights in the last layer. Now we focus on w7



5. When we compute the derivative of w7, we must keep in mind that it affects the value of w16 as well, so we have to account for it when we expand using the chain rule. Using the same formula as above, we get:

$$\frac{\partial E}{\partial w7} = \frac{\partial E}{\partial a4} * \frac{\partial a4}{\partial net7} * \frac{\partial net7}{\partial w7}$$

6.derivative of the error with respect to a4

The derivation of output w.r.t the affine

$$\frac{\partial E}{\partial a4} = \frac{\partial E}{\partial y} * \frac{\partial y}{\partial net} * \frac{\partial net}{\partial a4}$$

The derivative of error w.r.t output

derivative of affine w.r.t the activation 4

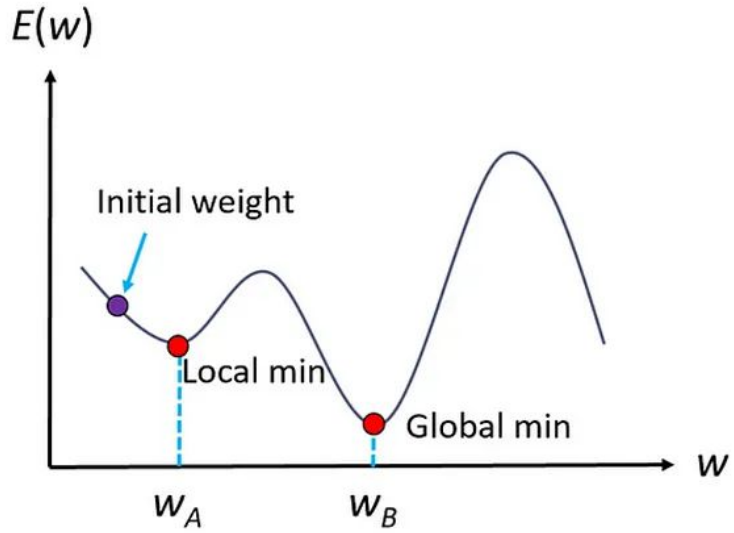
Putting it all together we get formula:

$$\frac{\partial E}{\partial y} * \frac{\partial y}{\partial net} * \frac{\partial net}{\partial a4} * \frac{\partial a4}{\partial net7} * \frac{\partial net7}{\partial w7}$$

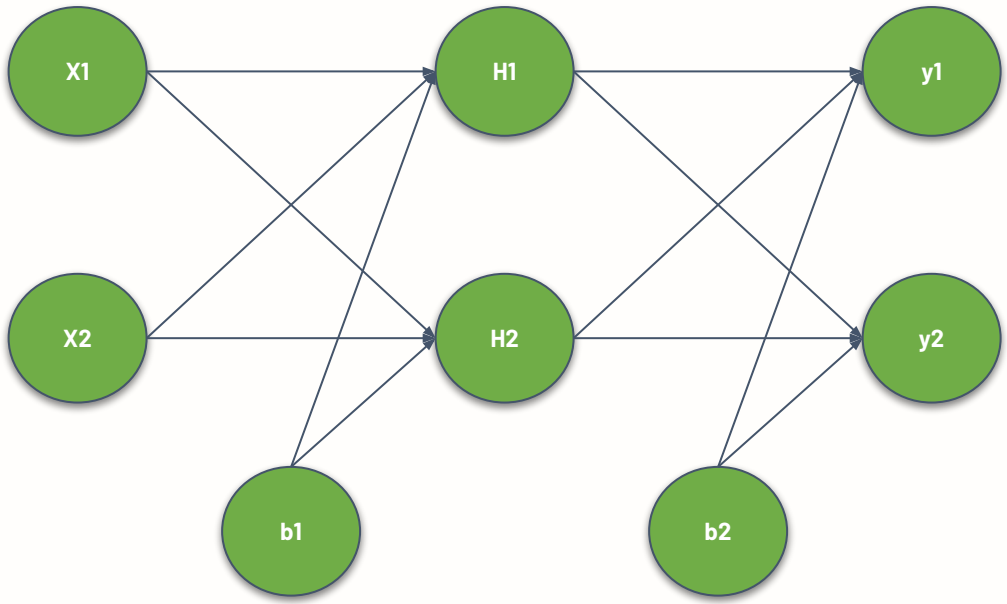
How's Backpropagation Works? (cont'd)

Purwadhika Digital Technology School

- We repeat the same process for all weights in a Neural Network and update the weights.
- All weight updates are done in one iteration, and we perform a number of n-iterations as we define the number of epochs or steps.
- By updating the weights, we hope to obtain output values that are close to the ground truth and minimize the error (obtain the global min).



Example



Input:
X1 = 0.05
X2 = 0.10

Target Values:
$\hat{y}1 = 0.01$
$\hat{y}2 = 0.99$

Biases:
b1 = 0.35
b2 = 0.60

Activation Func:
Sigmoid

Initial weights:

W1 = 0.15	W5 = 0.40
W2 = 0.20	W6 = 0.45
W3 = 0.25	W7 = 0.50
W4 = 0.30	W8 = 0.55

Example (cont'd)

Purwadhika Digital Technology School

Feed Forward Process

In H1 = (x1*w1) + (x2*w2) + b1
= (0.05 * 0.15) + (0.10 *0.20) + 0.35
= 0.3775

In H2 = (x1*w3) + (x2*w4) + b1
= (0.05 * 0.25) + (0.10 *0.30) + 0.35
= 0.3925

Out H1 = $\frac{1}{1+e^{-in\,H1}}$
= $\frac{1}{1+e^{-0.33775}}$
= 0.593269992

Out H2 = $\frac{1}{1+e^{-in\,H2}}$
= $\frac{1}{1+e^{-0.3925}}$
= 0.5968843783

Example (cont'd)

Purwadhika Digital Technology School

Feed Forward Process

In $\hat{y}1$

= (out H1*w5)+(out H2*w6)+b2

= (0.593269992*0.4)+(0.5968843783*0.45)+0.6

= 1.105905967

Out $\hat{y}1$ $= \frac{1}{1+e^{-in\,\hat{y}1}}$

$= \frac{1}{1+e^{-1.105905967}}$

= 0.75136507

In $\hat{y}2$

= (out H1*w7)+(out H2*w8)+b2

= (0.593269992*0.5)+(0.5968843783*0.55)+0.6

= 1.2249214041

Out $\hat{y}2$ $= \frac{1}{1+e^{-in\,\hat{y}2}}$

$= \frac{1}{1+e^{-1.2249214041}}$

= 0.772928465

Example (cont'd)

Purwadhika Digital Technology School

Feed Forward Process

Out $\hat{y}1$ = 0.75136507

Out $\hat{y}2$ = 0.772928465

E total $= 1/2 \sum (target - output)^2$

$= 0.5(0.01 - 0.75136507)^2 + 0.5(0.99 - 0.772928465)^2$

$= 0.274811083 + 0.023560026$

$= 0.298371109$

Example (cont'd)

Purwadhika Digital Technology School

Back Propagation Process

Error at w5 = $\frac{\partial Etotal}{\partial w5}$

E total = ½(target $\hat{y}1$ – out $\hat{y}1)^2$ + ½(target $\hat{y}2$ – out $\hat{y}2)^2$

Out $\hat{y}1 = \frac{1}{1+e^{-in\,\hat{y}1}}$

In $\hat{y}1$ = (out H1*w5)+(out H2*w6)+b2

$\frac{\partial Etotal}{\partial w5} = \frac{\partial Etotal}{\partial out\,\hat{y}1} * \frac{\partial out\,\hat{y}1}{\partial in\,\hat{y}1} * \frac{\partial in\,\hat{y}1}{\partial w5}$

a) $\frac{\partial Etotal}{\partial out\,\hat{y}1} = 2 * \frac{1}{2}(\text{target } \hat{y}1 - \text{out } \hat{y}1)^{2-1} * -1 + 0$

= -(target $\hat{y}1$ – out $\hat{y}1$)

= -(0.01 - 0.75136507)

= 0.74136507

b) $\frac{\partial out\,\hat{y}1}{\partial in\,\hat{y}1} = \frac{1}{2(\cosh(in\,\hat{y}1)+1)} = \frac{1}{2*(1.6764359889828+1)} = 0.186815602$

c) $\frac{\partial in\,\hat{y}1}{\partial w5}$ = (out H1*w5)+(out H2*w6)+b2

= 1* out H1 * $w5^{1-1}$ + 0 + 0

= 0.593269992

$\frac{\partial Etotal}{\partial w5} = \frac{\partial Etotal}{\partial out\,\hat{y}1} * \frac{\partial out\,\hat{y}1}{\partial in\,\hat{y}1} * \frac{\partial in\,\hat{y}1}{\partial w5}$

= 0.74136507 * 0.186815602 * 0.593269992

= 0.082167041

Example (cont'd)

Purwadhika Digital Technology School

Back Propagation Process

w5 = w5 - μ* $\frac{\partial Etotal}{\partial w5}$, Learning Rate $(\mu) = 0.5$

w5 = w5 – 0.5 * $\frac{\partial Etotal}{\partial w5}$

= 0.4 – 0.5 * 0.082167041

= 0.35891648

w6 = 0.408666186

w7 = 0.511301270

w8 = 0.561370121

Example (cont'd)

Purwadhika Digital Technology School

Back Propagation Process

Error at w1 = $\frac{\partial Etotal}{\partial w1}$

$$\frac{\partial Etotal}{\partial w1} = \frac{\partial Etotal}{\partial out\ H1} * \frac{\partial out\ H1}{\partial in\ H1} * \frac{\partial in\ H1}{\partial w1}$$

A B C

A) $\frac{\partial Etotal}{\partial out\ H1} = \frac{\partial E1}{\partial out\ H1} + \frac{\partial E2}{\partial out\ H2}$

$= (\frac{\partial E1}{\partial in\ \hat{y}1} * \frac{\partial in\ \hat{y}1}{\partial out\ H1}) + \frac{\partial E2}{\partial out\ H2}$

$= ((\frac{\partial E1}{\partial out\ \hat{y}1} * \frac{\partial out\ \hat{y}1}{\partial in\ \hat{y}1}) * \frac{\partial in\ \hat{y}1}{\partial out\ H1}) + \frac{\partial E2}{\partial out\ H2}$

= 0.74136607 * 0.186815602 * 0.4 + (-0.019049119)

= 0.0363503805

B) $\frac{\partial out\ H1}{\partial in\ H1} = \frac{1}{2(\cosh(in\ H1)+1)}$ = 0.241800709

C) $\frac{\partial in\ H1}{\partial w1}$ = 1 *x1 + 0 + 0 = 0.05

$\frac{\partial Etotal}{\partial w1}$ = 0.0363503805 * 0.241800709 * 0.05

= 0.000438568

Example (cont'd)

Purwadhika Digital Technology School

Back Propagation Process

$w1 = w1 - \mu * \frac{\partial Etotal}{\partial w1}$, Learning Rate $(\mu) = 0.5$

$w1 = w1 - 0.5 * \frac{\partial Etotal}{\partial w1}$

$= 0.15 - 0.5 * 0.000438568$

$= 0.149780716$

$w2 = 0.19956143$

$w3 = 0.24975114$

$w4 = 0.29950229$

Conclusion

Purwadhika Digital Technology School

- The concept of neural networks is inspired by biological neural networks.
- Neural networks can process data better than traditional machine learning.
- The complexity of neural network calculations is very high so that the computational process required is greater.
- The neural network architecture is more customizable and the depth of the layer can be adjusted.
- The activation function will help the neural network calculation process to obtain more accurate results.
- The results of backpropagation are expected to obtain new weights that can reduce the prediction error values.

Purwadhika
Digital Technology School

Neural Networks

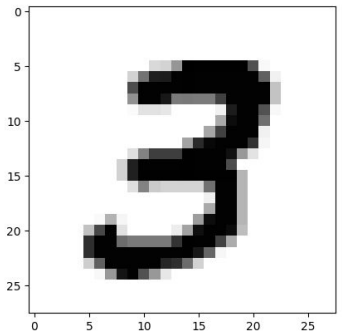
Hands On Neural Networks with Python

Required Library

Prepare the Data

- Load data
- Perform all necessary data processing
- All features in numeric vector
- Label/target transformed to one hot vector

data



numeric vector

```
print(f"x_train_1d shape: {x_train_1d.shape}"
print(f"x_test_1d shape: {x_test_1d.shape}")
```

```
x_train_1d shape: (60000, 784)
x_test_1d shape: (10000, 784)
```

Define Neural Network Model

Purwadhika Digital Technology School

```
# design ANN architecture
model = Sequential()
model.add(layers.Input(shape=(x_train_1d.shape[1],)))
model.add(Dense(64, activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(10, activation='softmax'))
```

Model architecture

- Define every layers of Neural Network
- Define activation function
- For multi-class classification task, add softmax in last layer
- Set value of last Dense equal to number of unique label/target

Compile mode

- Set loss = “sparse_categorical_crossentropy” for multi-class classification
- Choose optimizer, in this example “adam”.
- Set metrics based on use case.

```
# complie model
model.compile(loss='sparse_categorical_crossentropy',
              optimizer='adam', metrics=['accuracy'])
```

Train Your Neural Network

```
history = model.fit(x_train_1d, y_train, epochs=10, batch_size=128, validation_split=0.2)
```

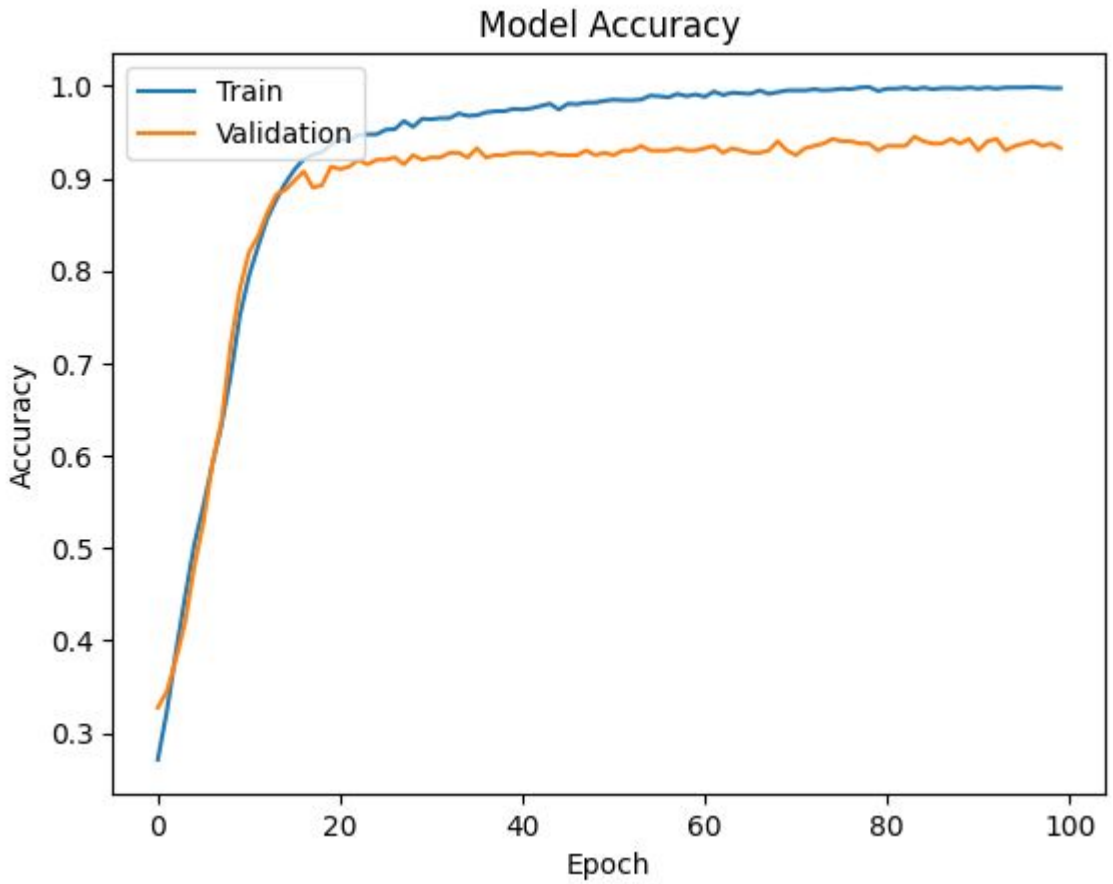
```
Epoch 1/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.7847 - loss: 0.7724 - val_accuracy: 0.9352 - val_loss: 0.2264
Epoch 2/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9382 - loss: 0.2136 - val_accuracy: 0.9521 - val_loss: 0.1695
Epoch 3/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9565 - loss: 0.1540 - val_accuracy: 0.9592 - val_loss: 0.1420
Epoch 4/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9636 - loss: 0.1216 - val_accuracy: 0.9617 - val_loss: 0.1290
Epoch 5/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9692 - loss: 0.1038 - val_accuracy: 0.9632 - val_loss: 0.1225
Epoch 6/10
375/375 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9748 - loss: 0.0862 - val_accuracy: 0.9671 - val_loss: 0.1110
```

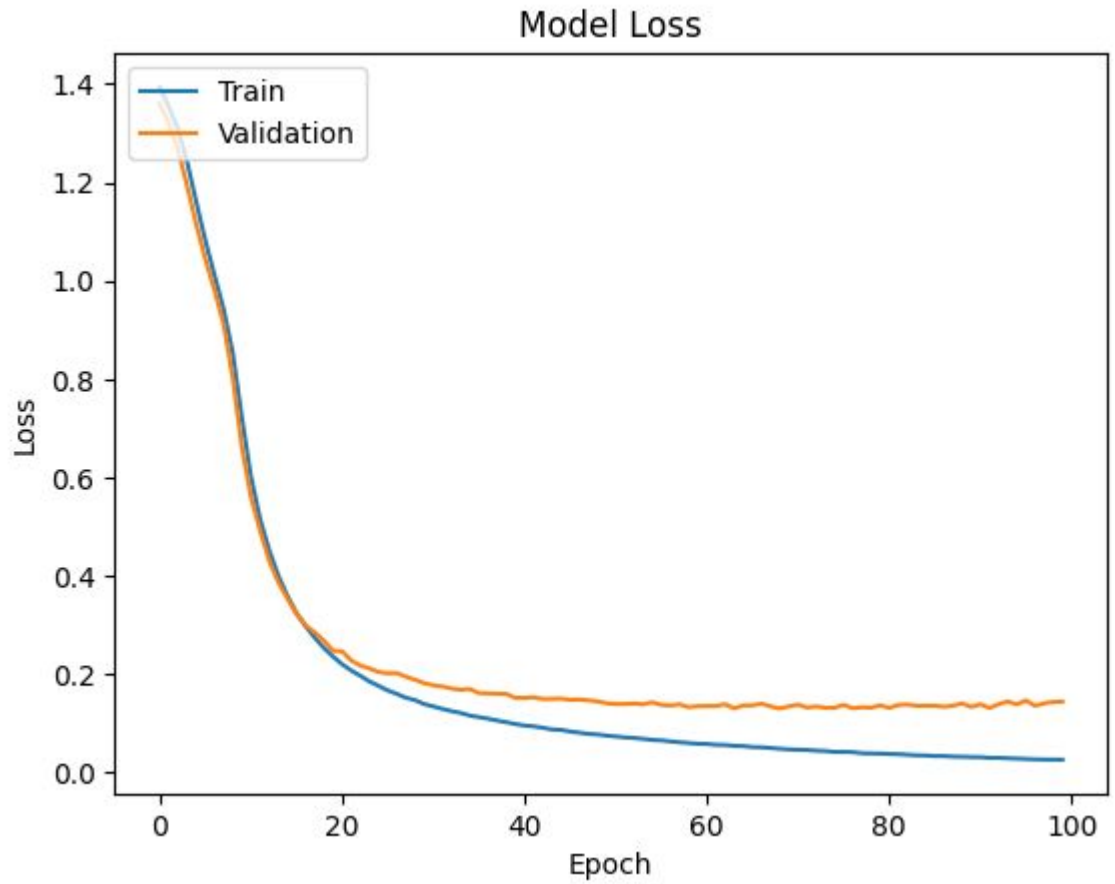
epoch number

loss & accuracy on data train

loss & accuracy on data test

Learning Curve Neural Network





Model Evaluation

```
# calculate metric performance
y_pred = model.predict(X_test)
y_pred_classes = np.argmax(y_pred, axis=1)

print(classification_report(y_test, y_pred_classes))
```

```
13/13 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
              precision    recall  f1-score   support

           0       0.95      0.91      0.93       105
           1       0.90      0.95      0.92        91
           2       0.92      0.93      0.93        92
           3       0.95      0.94      0.95       112

    accuracy                           0.93       400
   macro avg       0.93      0.93      0.93       400
weighted avg       0.93      0.93      0.93       400
```

THANK YOU