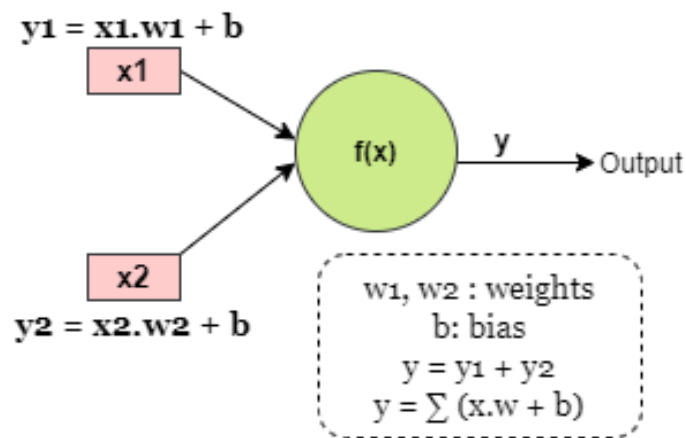


Neural Networks: A Quick Overview

Neural Networks are indispensable for Deep Learning. The basic idea behind Neural Networks is the Perceptron Model which is similar to biological neurons in terms of both features and functionality. Neural networks are the network of artificial neurons or nodes that learn features and patterns from input data by analyzing and performing computational tasks to give the desired output after training on the dataset.



A Perceptron Model

Weights – Weights are associated with every input neuron. They are the information used by neurons and represent the strength of the connection between neurons in a Neural Network. The more the weight of the neuron, the more impact it will have. Dimensions: $w[l] \rightarrow (n[l], n[l-1])$

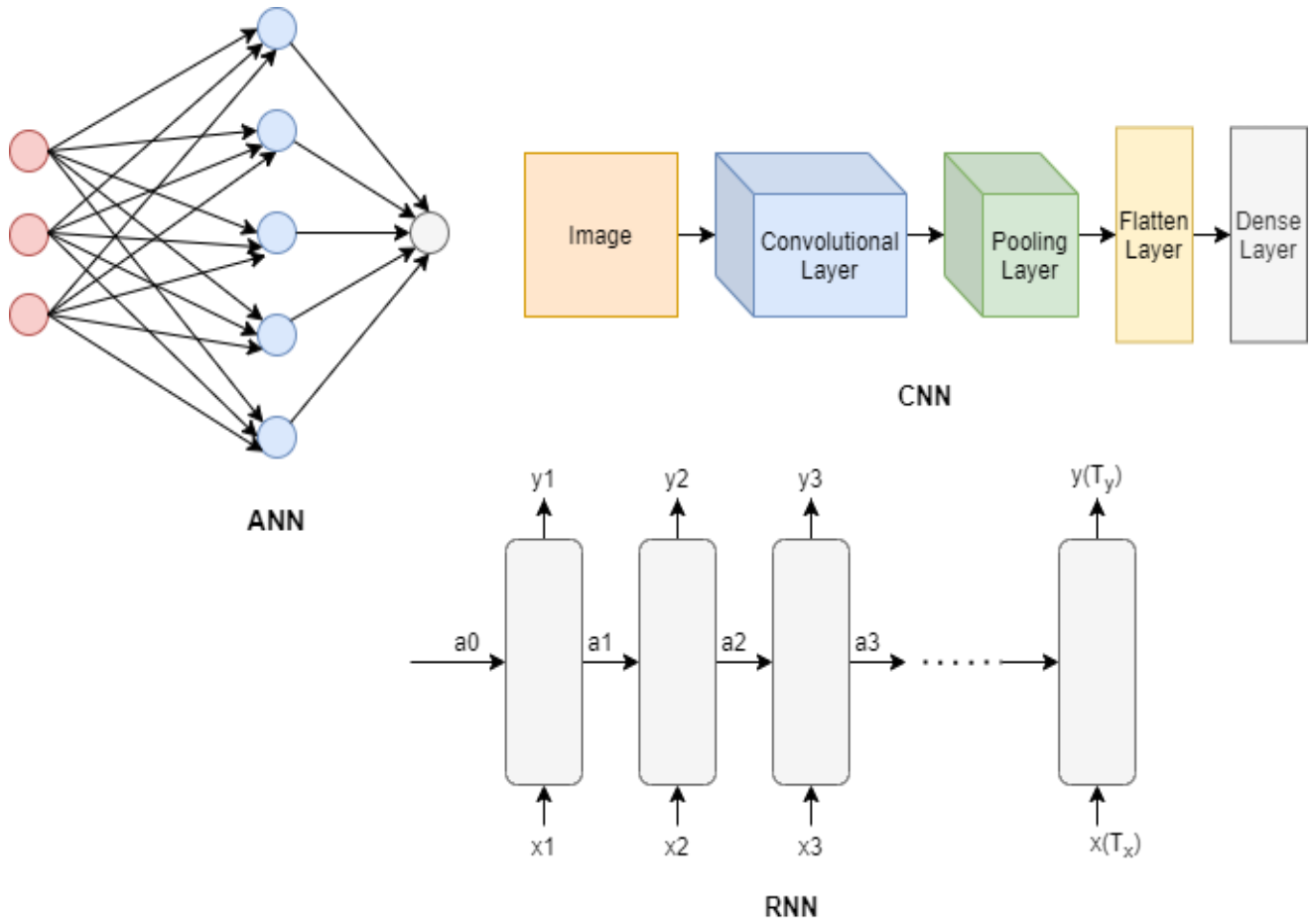
Bias – Bias allows the activation function to shift by a constant value. It's used to adjust the output of the weighted sum of input. It's analogous to an intercept of a linear equation. Dimensions: $b[l] \rightarrow (n[l], 1)$

Note: $n[l]$ - no of nodes/neurons in the l th layer. $n[l-1]$ – no of nodes/neurons in the previous layer i.e. $(l-1)$ th layer.

Three Major Types of Neural Networks:

1. **Artificial Neural Network (ANN)** – Voice recognition,
2. **Convolutional Neural Network (CNN)** – Image Recognition, image classification, face recognition, object detection, pattern recognition, etc.

3. **Recurrent Neural Network (RNN)** – Speech Recognition, Machine Translation, Music generation, Sentiment Classification, Name entity recognition, etc.



Let's see a brief overview of ANNs in this section.

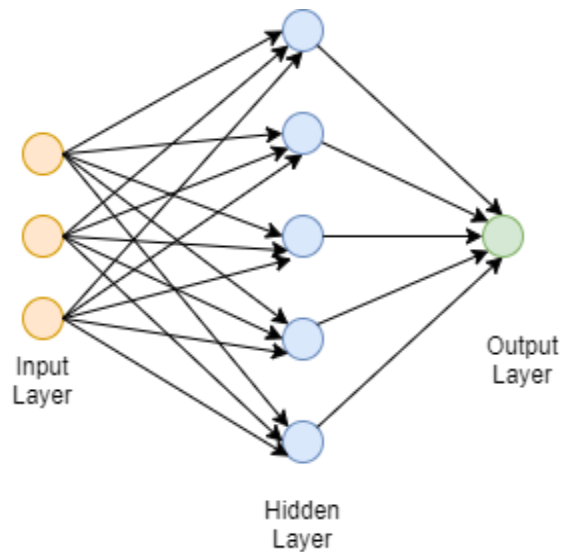
In a complete Neural Network i.e. a multi-layer perceptron model, every neuron is connected to every neuron in the next layer. Also called Fully Connected Dense Layer. Artificial Neural Nets can be visualized as directed weighted graphs where each neuron is a node and an edge represents weights between input and output neurons.

3 Types of Layers:

1. Input Layer – Input Feature Matrix (x_1, x_2, \dots, x_m) of size (n_x, n_m)
2. Hidden Layers – Processing the input and performing all the computations to find hidden patterns and features.
3. Output Layer – Outputs the learning (y_1, y_2, \dots, y_m) of size $(1, m)$

Note: m is the no. of training data/examples.

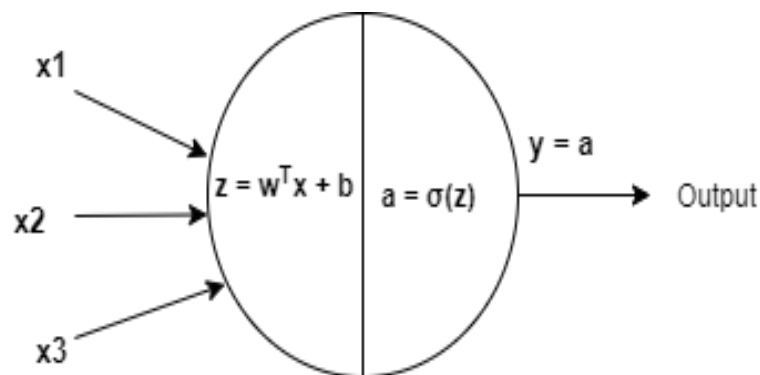
There can be multiple hidden layers depending on the problem's complexity and functionality.



In an Artificial Neural Network (ANN), there are 2 elementary steps of computation for each neuron as shown below:

Step 1: $z = \sum (w^T x + b)$

Step 2 : $a = \sigma(z)$. Here, σ is an activation function.



Note: Here, w and b are parameters.

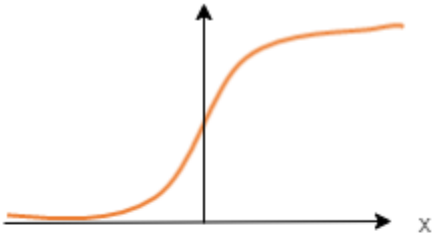
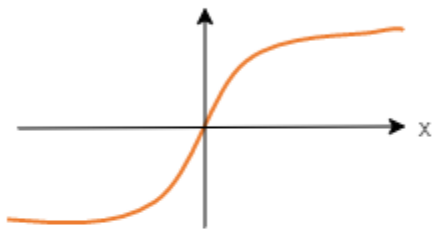
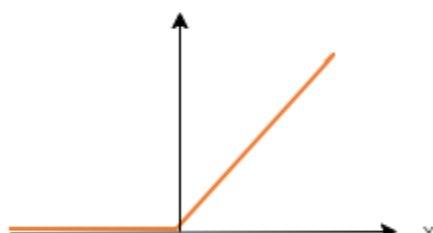
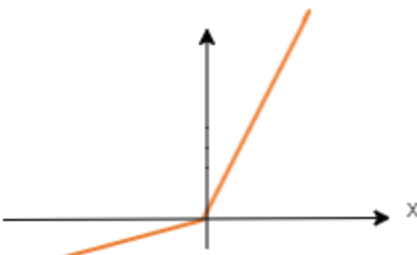
The idea is that each layer learns weights during the training of the data and the learned data is then fed to an activation function to get the output. The no. of layers and no. of neurons or nodes (hyperparameters) in each hidden layer depends on the complexity of the problem and should be tuned to get the desired result. There is a bi-directional propagation i.e. Forward Propagation and Backward Propagation.

Note: $a_i^{[l]}$ - activation of i^{th} node/neuron of the l^{th} layer.

Few terminologies:

1. Activation Function – It's a function that defines the output of a neuron and determines whether a neuron should be activated or not.

Types of Activation Functions:

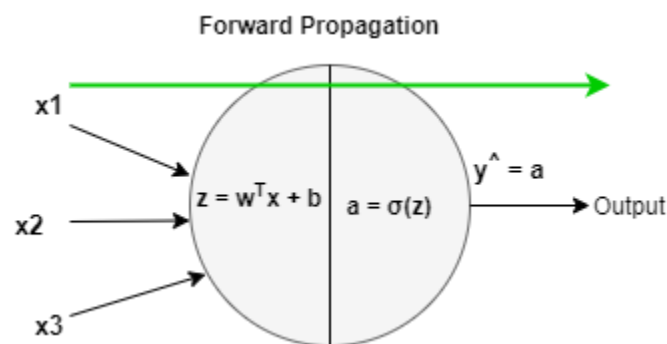
Activation Functions	
sigmoid $\sigma(x) = \frac{1}{(1 + e^{-x})}$	
tanh $\tanh(x) = \frac{e^x + e^{-x}}{e^x - e^{-x}}$	
ReLU $\max(0, x)$	
Leaky ReLU $\max(0.1x, x)$	

Note: ReLU → Rectified Linear Unit

1. Loss Function – The Loss function is used to evaluate the performance of the model. It defines the measure of how good output \hat{y} performs when the true label is y .
2. Cost Function – It measures how well parameters w and b are doing on the training set.
Cost Function : $J(w, b) = 1/m * (\sum L(\hat{y}, y))$

Note: Initialization of w and b should never be zero. It should be initialized to random numbers. Poor initialization leads to vanishing or exploding gradients which decelerates the optimization algorithms.

In forward Propagation, weights are multiplied with inputs and biases are added which is then fed to an activation function (as shown in the diagram). The model learns by adjusting weights and biases (parameters) using algorithms.



In backpropagation, gradient of the loss function is computed. Algorithms iterate backward by computing the partial derivative of cost function w.r.t all parameters.

Note: Gradient Descent is performed to minimize the cost or loss function. Gradient Descent is the most common optimization Algorithm used in Deep Learning. Other Optimization Algorithms are Adam, Gradient Descent with Momentum, mini-batch Gradient Descent, and Root Mean Square prop (RMSprop).

General Method to build a Neural Network:

1. Define the Neural Network Structure according to the problem under consideration – no. of input neurons, hidden nodes, no. of layers, learning rate α , etc.
2. Initialize parameters - weights and biases.
3. Iterate – No of iterations depends on batch size and no. of batches (hyperparameters)
 1. Perform Forward Prop – In forward pass, model generates its initial predictions

$$z^{[l]} = w^{[l]} a^{[l-1]} + b^{[l]}$$

$$a^{[l]} = g^{[l]}(z^{[l]})$$
 2. Compute Loss – Loss function is calculated to check how far the actual and predicted value are.
 3. Implement Backward Propagation

```

dz[l] = a[l] - y
dw[l] = 1/m * (dz[l]* a[l-1].T)
db[l] = 1/m * np.sum(dz[l], keepdims = true, axis =1) //Note: keepdims helps to
prevent rank 1 arrays

.....
dz[l-1] = w[l].T * dz[l] * g'[l-1](z[l-1])
4. Update parameters
W[l] = w[l] - α*dw[l]
b[l] = b[l] - α*db[l]

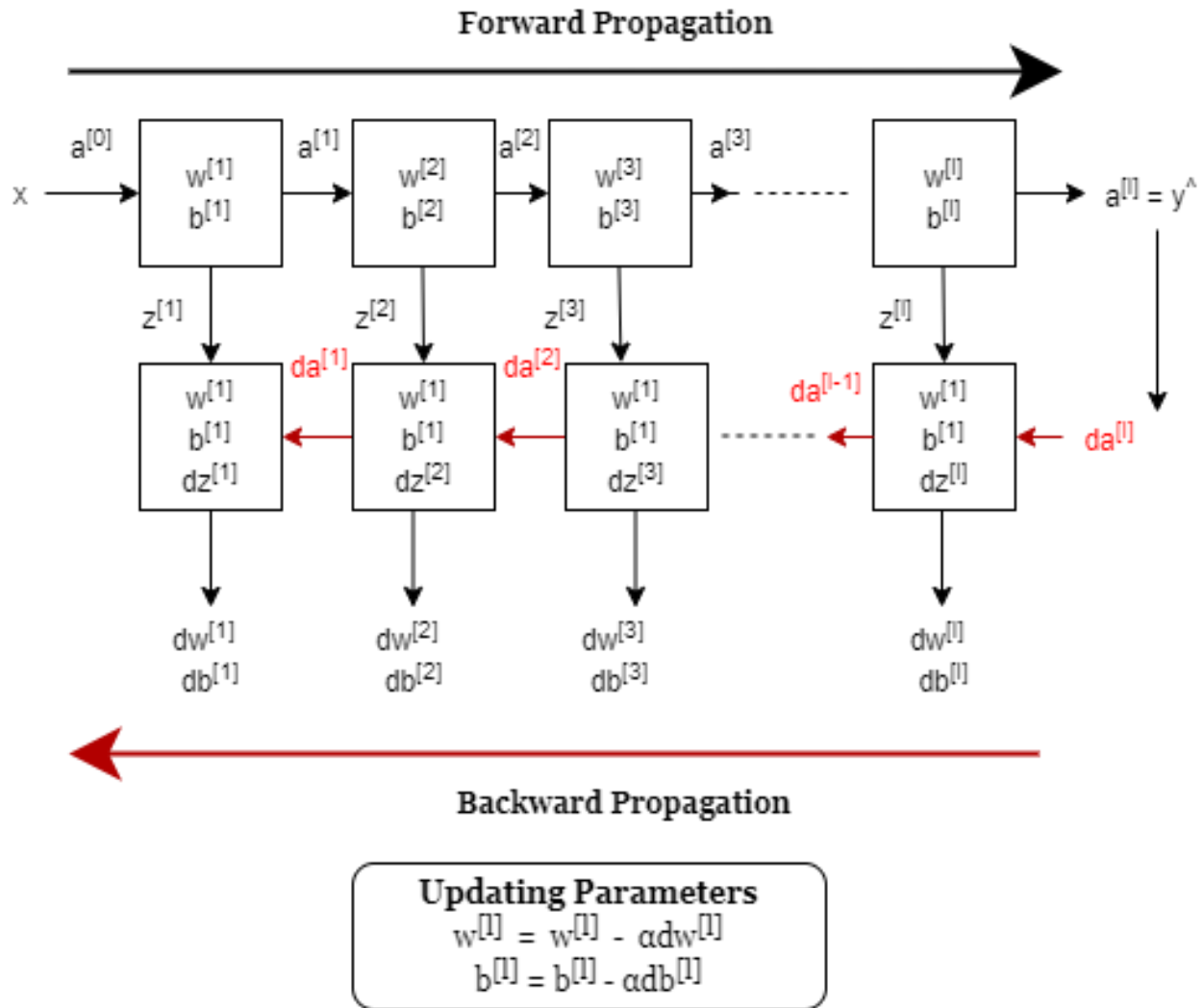
```

Note: 1 epoch is one iteration of the entire training set. Batch size is the total no of training examples in a single batch. Iterations is no of batches of the batch size needed to complete 1 epoch.

Let's take an example:

m = 10,000 training examples.

Say, we have a batch size of 1000. So, to complete 1 epoch i.e. to cover all 10,000 examples we would require 10 no of batches or iterations.



Notations:

Layer l : $w^{[l]}, b^{[l]}$

Forward Prop: Input – $a^{[l-1]}$ **and** Output – $a^{[l]}$

$$z^{[l]} = w^{[l]}a^{[l-1]} + b^{[l]}$$

$$a^{[l]} = g^{[l]}(z^{[l]})$$

Note: $da^{[l]}$ is calculated after applying loss function

Backward Prop: Input- $da^{[l]}, z^{[l]}$ **and** output – $da^{[l-1]}, dw^{[l]}, db^{[l]}$

Hyperparameters:

1. Learning Rate – α
2. No of iterations of gradient descent/ optimization algorithm
3. No of hidden layers

4. No of neurons in each layer
5. Activation function choice

Applications and Scope of Neural Networks: Neural Networks finds application from industry to day-to-day tasks like image recognition, speech to text, machine translation, signature verification, object detection, and many more.