**REFERENCES:**

[1] G. Cybenko, “Approximation by superposition of a sigmoidal function,” Math. ControlSignals Systems, 1989.

[2] F. Chollet. *Deep Learning with Python*. Manning, Shelter Island, NY, USA, 1st edition, 2017

[3] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning (Adaptive Computation and Machine Learning Series*. MIT Press, Cambridge, MA, USA, 2016.

[4] <https://www.ibm.com/cloud/learn/gradient-descent>

[5] Sebastian Ruder. *An overview of gradient descent optimization algorithms*

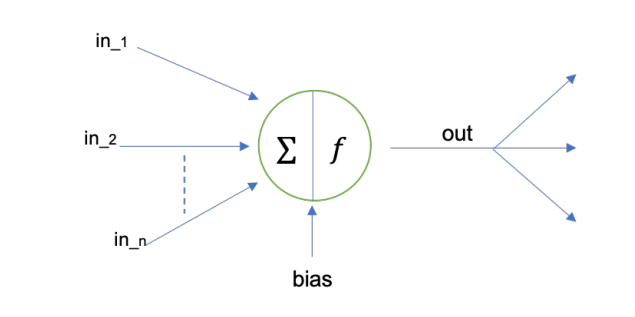
**Background Knowledge**

1. **Artificial Neural Networks (ANN)**

The concept of Artificial Neural Networks was built on the inspire of human brain biology characteristics, Cybenko [1] proved that in a neural network a single hidden layer containing a finite number of neurons is capable of approximating any continuous function to any desired precision. it included a group of machine learning models that are able to “learn” to execute or perform some specific tasks based on supplied examples. In general, these networks were created by a group of units or nodes connected each other like a simulating version of neurons in human brain, which called artificial neurons.

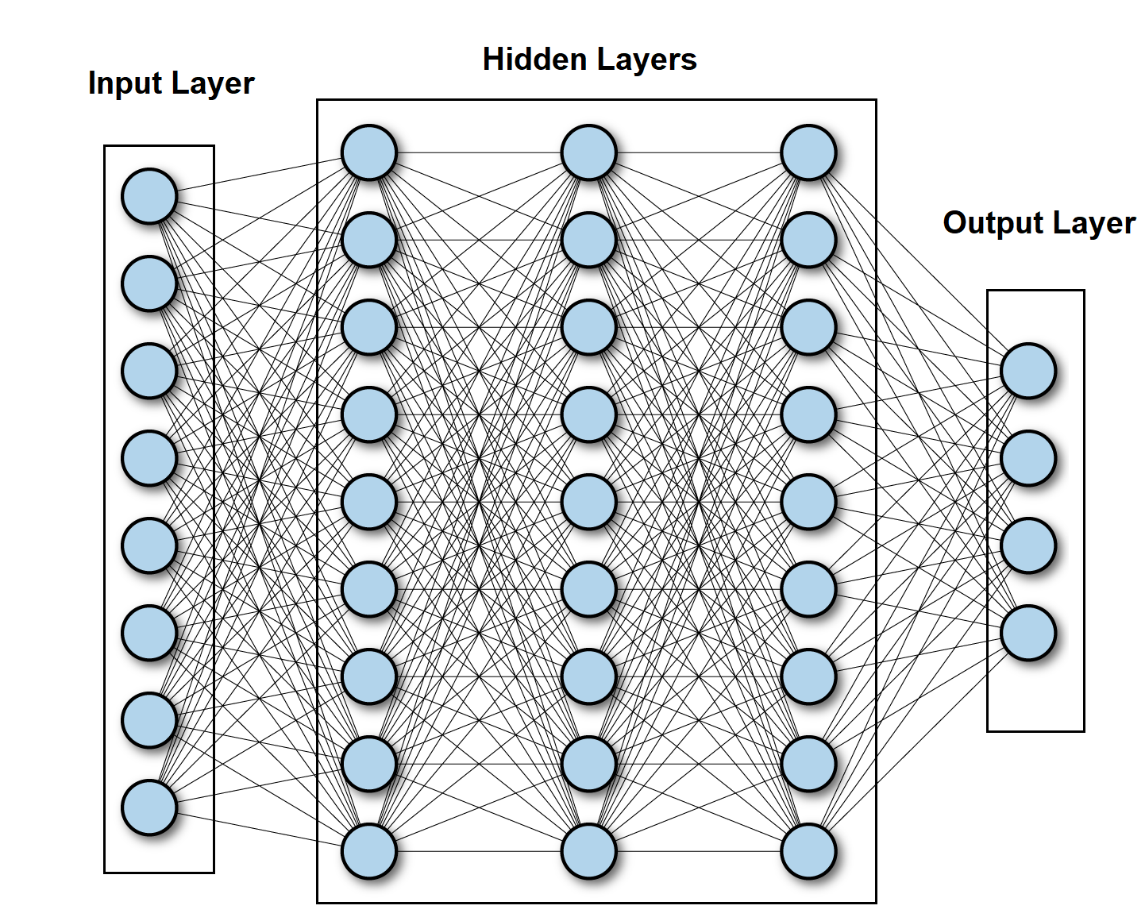
* 1. **Neuron**

Analog electrical signals flow inside biological neurons, while digital values present as the signals for artificial neurons. Each artificial neuron has an output being computed by using a function of the sum of the input signal values with generally added bias, the connection between the neurons as known as edges, in each connection contains an adjustable model parameter named weight that can be “learned” through a process called “model training”. The strength of the synaptic will determines the interaction’s level between the neurons, this function combines the neurons, bias and weights in the connections called “activation function”, and there are several activation functions that mean there is not only one fixed activation function for ANN, and we can choose the activation function for the network’s output. These functions will be mentioned later in this thesis.



* 1. **Layer**

In typical, all neurons are linked directly in network, on the other hand, they are grouped into subsets of neurons known as layers, which are generally connected in a sequential way based on the activation function in every node to compute nodes’ outputs from node’s inputs. In detail, the neurons in a layer are connected to a set of neurons in previous layer and to another one in next layer. Because of the modeling of ANNs as a connection of a neurons’ structure in layer, neurons’ outputs at a certain layer become neurons’ inputs at the following layer, which networks are understood as feedforward neural networks. There are no loops in present layer, therefore the inputs are always sent forward. As a result, the neurons into a layer cannot be connected with each other. When all the neurons in a layer are connected to all other ones in previous layer and following layer, it is called “Fully Connected Layers”, this state of the connection is kept in whole network as a series of fully connected layers that will be called “Fully Connected Neural Network”.



There is an existing of 3 kinds of layer inside a network, they are input layer, hidden layers and output layer.

Input layer represent for the input of the network, the number of nodes as known as neurons are size of the input that call “input dimension”. The hidden layers lay between input lay and output layer, there are so many hidden layers in a network, however there is only one input layer and output layer, and the output layer represent for the output of the network. In addition, the bias contains in input and output layer, not in output layer. Although simple decisions are made at the first layer based on the input, sophisticated judgments are already made in the second layer depending on decisions made in the first layer. As one moves deeper into the network, more difficult and abstract judgments emerge. The phrase deep neural networks refer to a network with numerous hidden layers.

* 1. **Forward and Backward (back propagation) steps**

As the mentioning before, the signals travel from input layer to output layer traversing hidden layers, which operations performed over the values received from the nodes in previous layers (in all layers but in the input layer) and whole activation function results are sent forwards to the next layer till the output layer. However, it is not the end of process. After getting the result of output layer, there is a backwards named back-propagation applying in here as known as training step for ANN. The algorithm is used to effectively train a neural network through a technique called chain rule. In simple terms, after each forward pass through a network, backpropagation performs a backward pass while adjusting the model’s parameters (weights and biases), which helps minimize the error for network’s result, it is called “loss function” or “cost function” which is the evaluation between predicted output and expected output at the final point of forward step. Basically, this function calculates the gradient of the loss function to each parameter in network respectively that means it work from the top layer (output layer) to the bottom layer (input layer) to calculate the contribution of each parameter in that value [2], which update the weights until reaching the minimum values. Furthermore, the key point of calculating the minimum value for loss function is gradient descent, which is a process of determining the modifications needed to minimize the loss and optimize the model to a specific evaluation measure by calculating the derivative of the loss function [3].

* 1. **Gradient Descent:**

Gradient descent is an optimization algorithm which is commonly-used to train machine learning models and neural networks [4]. It is built on a convex function and iteratively changes its parameters to minimize a given function to its local minimum. On the other words, Gradient descent begins by defining the initialized parameter values, and then iteratively adjusts the values to minimize the given cost-function. Deeply, a gradient is a function’s derivative with several input variables. In terms of mathematics, a gradient is known as the function’s slope, and it simply measures the change in all weights in relation to the change in error. The steeper the slope and the faster a model can learn, the higher the gradient. However, if the slope is zero, the model will stop learning. A gradient is a partial derivative with regard to its inputs.

Imaging that a blindfolded man wants to hike down to the bottom of a valley from the top of a hill with as few steps as possible. He might begin going down the hill by taking large steps in the sharpest direction, which he can do as long as he is not close to the bottom. However, as he gets closer to the bottom, his steps will become smaller and smaller in order to prevent overshooting it. The gradient can be used to mathematically characterize this process. The learning rate η determines the size of the steps we take to reach a (local) minimum. In other words, we follow the direction of the slope of the surface created by the objective function downhill until we reach a valley [5].

To reach the local minimum for gradient descent, we should choose a relevant value for the learning rate, which is not both too low and too high. This is very significant because if the steps are too large, it could fail to reach the local minimum since it bounces between the convex function of gradient descent. Besides, gradient descent will eventually reach the local minimum if we set the learning rate to a very low value, but it may take a lot of time. So, the key point is choosing an optimal learning rate that helps model converge to the minimum value without spending much time.



Gradient descent has three forms that differ in the amount of data used to compute the gradient of the objective function. We choose between the accuracy of the parameter update and the time it takes to complete an update based on the amount of data.

* + 1. **Batch Gradient Descent**

Batch gradient descent calculates the sum of the error for each point in a training set and updates the model only after all training instances have been evaluated. This procedure is known as a training epoch. Though this batching supplies a computation efficiency, it can still take a long time to execute big training datasets because it must still keep all of the data in memory. Batch gradient descent also frequently yields a stable error gradient and convergence, although that convergence point isn't always the best, locating the local minimum rather than the global minimum.

* + 1. **Stochastic Gradient Descent**

Stochastic gradient descent (SGD) performs a training epoch for each example in the dataset and updates the parameters of each training example at a time. They are easier to remember because you only need to hold one training example. In spite of the fact that these frequent updates provide more detail and speed, they can lead in computational efficiency losses when it be compared to batch gradient descent. Despite of ability of noisy gradients production, these frequent updates can also aid in escape the local minimum and locating the global one.

* + 1. **Mini-batch Gradient Descent**

Mini-batch gradient descent combines batch gradient descent with stochastic gradient descent ideas. It divides the training dataset into small batches and updates each of those batches. This method strikes a balance between batch gradient descent's computing efficiency and stochastic gradient descent's speed.

Besides, gradient descent has its own set of challenges such as: Local minima and saddle points, vanishing and exploding gradients.

* Local minima resemble global minima in shape with the slope of the cost function increasing on each side of the current location.
* Saddle points happen when the negative gradient exists just on one side of the point, reaching a local maximum on one side and a local minimum on the other. Its name was inspired by horse's saddle;
* Vanishing gradients happens when the gradient is too small. The gradient continues to shrink as we move backwards during backpropagation, leading earlier levels of the network to learn more slowly than later layers. When this occurs, the weight parameters are updated until they become inconsequential, i.e., 0, resulting in an algorithm that is no longer learning.
* Exploding gradients occurs when the gradient becomes too big, resulting in an unstable model. In this instance, the model weights will get excessively huge and will be represented as “NaN” finally. One approach to this problem is to use a dimensionality reduction technique, which can help to reduce model complexity.

1. **Convolutional Neural Network**