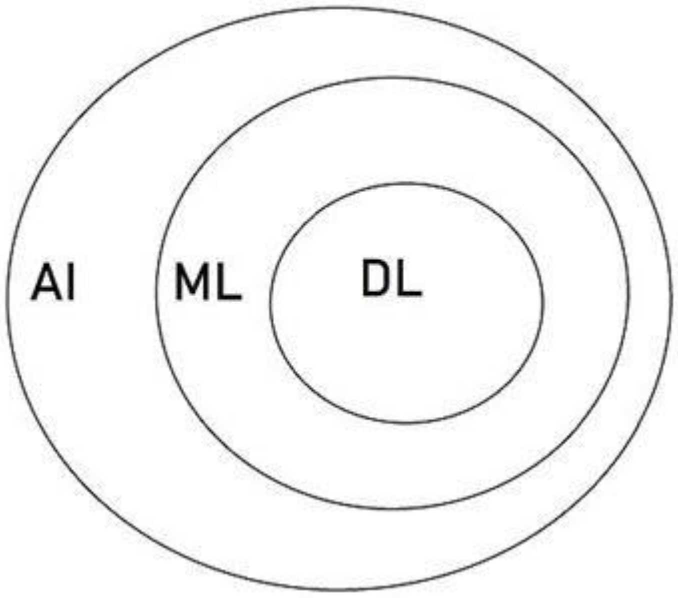
What is Deep Learning?

Deep Learning is a machine learning subfield concerned with algorithms inspired by the brain's structure and a function called artificial neural networks.

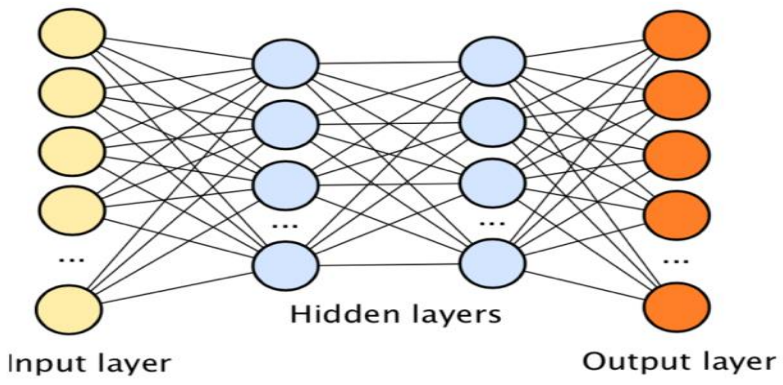


Deep learning is a key technology behind driverless cars, enabling them to be familiar with a stop sign or distinguish a pedestrian from a lamppost.

A computer model learns to perform classification tasks directly from images, text, or sound in deep learning. Deep learning models can achieve state-of-the-art accuracy, sometimes exceeding performance at the human level.

Models are trained using a large set of labelled data and neural network architectures that contain many layers.

How Deep Learning works?



The Input Layer: At the information layer, the neural system retains all the unclassified information that it is given. This implies separating the data into numbers and transforming them into bits of yes-or-no information, or “neurons”. If you needed to show a neural system to perceive words, at that point the information layer would be numerically characterizing the state of each letter, separating it into advanced language so the system can begin working. The info layer can be really basic or fantastically unpredictable, contingent upon the fact that it is so natural to speak to something numerically.

The Hidden Layer: At the focal point of the neural system are concealed layers — somewhere in the range of one to many. These layers are made of their advanced neurons, which are intended to enact or not initiate dependent on the layer of neurons that goes before them. A solitary neuron is an essential “on the off chance that this, at that point that” show, yet layers are made of long chains of neurons, and a wide range of layers can impact one another, making complex outcomes. The objective is to enable the neural system to perceive a wide range of highlights and consolidate them into a solitary acknowledgment, similar to a kid figuring out how to perceive each letter and afterward framing them together to perceive a full word, regardless of whether that word is composed somewhat messy.

The concealed layers are additionally where a great deal of profound getting the hang of preparing goes on. For instance, if the calculation neglected to precisely perceive a word, software engineers send back, “Sorry, that is not right,” and the calculation would modify how it gauged information until it found the correct answers. Rehashing this procedure (software engineers may likewise alter loads physically) enables the neural system to develop strong concealed layers that are adroit at searching out the correct answers through a great deal of experimentation also, some outside guidance — once more, much like how the human cerebrum functions. As the above picture appears, concealed layers can turn out to be extremely mind-boggling!

The output layer: The yield layer has generally a couple of “neurons” since it’s the place ultimate choices are made. Here the neural system applies the last examination, settles on definitions for the information, and reaches the customized inferences dependent on those definitions. For instance, “Enough of the information lines up to the state that this word is a lake, not path.” Ultimately all information that goes through the system is limited to explicit neurons in the yield layer. Since this is the place the objectives are understood, it’s regularly one of the initial segments of the system made.

Examples of Deep Learning

Deep learning applications are used in industries from automated driving to medical devices.

Automated Driving: Automotive researchers are using deep learning to automatically detect objects such as stop signs and traffic lights. Besides, deep learning is used to detect pedestrians, which helps decrease accidents.

Aerospace and Defence: Deep learning is used to identify objects from satellites that locate areas of interest, and identify safe or unsafe zones for troops.

Medical Research: Cancer researchers are using deep learning to automatically detect cancer cells. Teams at UCLA built an advanced microscope that yields a high-dimensional data set used to train a deep learning application to accurately identify cancer cells.

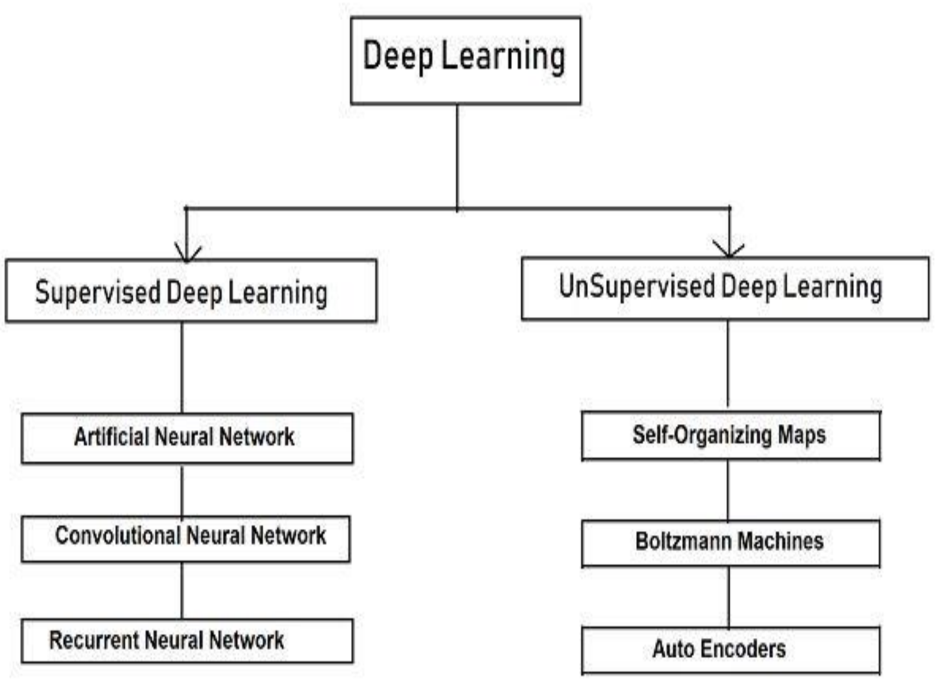
Industrial Automation: Deep learning is helping to improve worker safety around heavy machinery by automatically detecting when people or objects are within an unsafe distance of machines.

Electronics: Deep learning is being used in automated hearing and speech translation. For example, home assistance devices that respond to your voice and know your preferences are powered by deep learning application.

Types of Deep Learning Algorithms

There some variations of how to define the types of Deep Learning Algorithms but commonly they can be divided into categories according to their purpose and the main categories are the following:

- **Supervised learning**
- **Unsupervised Learning**



Supervised Learning

I like to think of supervised learning with the concept of function approximation, where we train an algorithm and at the end of the process, we pick the function that best describes the input data, the one that for a given X makes the best estimation of y (X -> y).

Most of the time we are not able to figure out the true function that always makes the correct predictions and another reason is that the algorithm relies upon an assumption made by humans about how the computer should learn, and these assumptions introduce a bias,

Here the human experts act as the teacher where we feed the computer with training data containing the input/predictors and we show it the correct answers (output) and from the data, the computer should be able to learn the patterns.

- Supervised learning algorithms try to *model relationships and dependencies between the target prediction output and the input features* such that we can predict the output values for new data based on those relationships which it learned from the previous data sets.

List of Common Algorithms

- Artificial Neural Network
- Convolutional Neural Network
- ·Recurrent Neural Network

Unsupervised Learning

- The computer is trained with unlabelled data.
- Here there's no teacher at all the computer might be able to teach you new things after it learns patterns in data, these algorithms a particularly useful in cases where the human expert doesn't know what to look for in the data.
- Its the family of Deep learning algorithms which are mainly used in *pattern detection* and *descriptive modelling*. However, *there are no output categories or labels* here based on which the algorithm can try to model relationships.

- These algorithms try to use techniques on the input data to *mine for rules*, *detect patterns*, and *summarize and group the data points which* help in deriving meaningful insights and describe the data better to the users.

Artificial Neural Network (ANN)

1. What is an Artificial Neural Network?
2. Types of Artificial Neural Network.
3. Activation Functions and Their types?
4. How Do Backpropagation work?

5. Practical Implementation of ANN in Keras and TensorFlow.

What is an Artificial Neural Network?

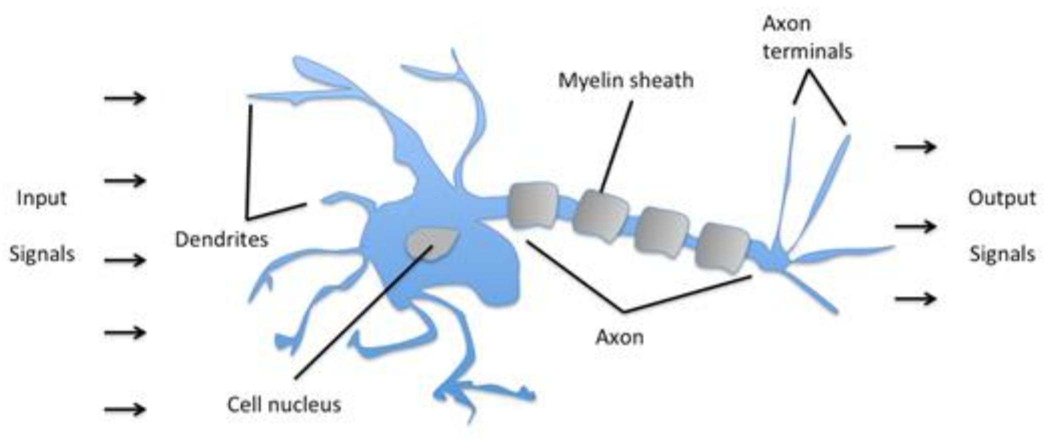
- Artificial neural networks are one of the main tools used in machine learning. As the “neural” part of their name suggests, they are brain-inspired systems that are intended to replicate the way that we humans learn.
- Neural networks consist of input and output layers, as well as (in most cases) a hidden layer consisting of units that transform the input into something that the output layer can use.
- They are excellent tools for finding patterns that are far too complex or numerous for a human programmer to extract and teach the machine to recognize.

Basic Structure of ANNs

The idea of ANNs is based on the belief that working of the human brain by making the right connections can be imitated using silicon and wires as living **neurons** and **dendrites**.

- The human brain is composed of 86 billion nerve cells called **neurons.** They are connected to other thousand cells by **Axons.**
- Stimuli from the external environment or inputs from sensory organs are accepted by dendrites. These inputs create electric impulses, which quickly travel through the neural network.

- A neuron can then send the message to another neuron to handle the issue or does not send it forward.



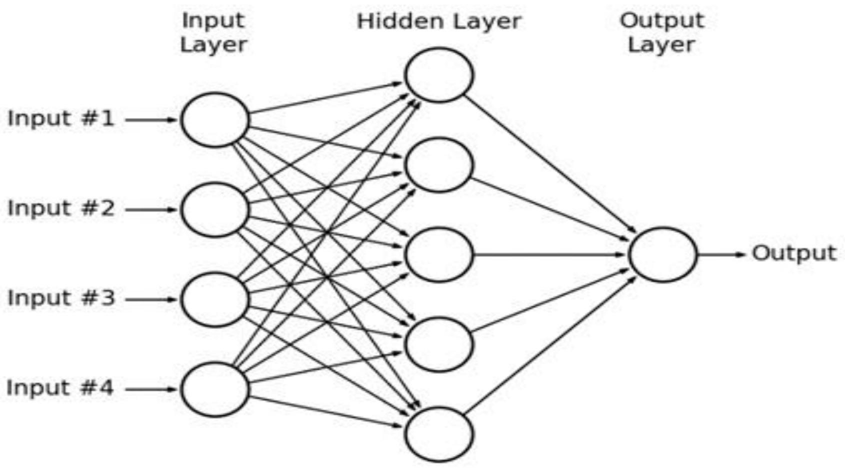
Schematic of a biological neuron.

- ANNs are composed of multiple **nodes**, which imitate biological **neurons** of the human brain.

- The neurons are connected by links and they interact with each other. The nodes can take input data and perform simple operations on the data.

- The result of these operations is passed to other neurons. The output at each node is called its **activation** or **node value.**

- Each link is associated with **weight.** ANNs are capable of learning, which takes place by altering weight values.

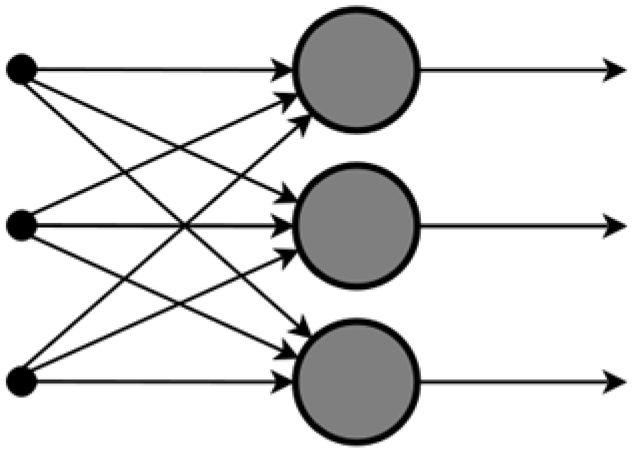


Types of Artificial Neural Networks

There are two Artificial Neural Network topologies – **Feedforward** and **Feedback.**

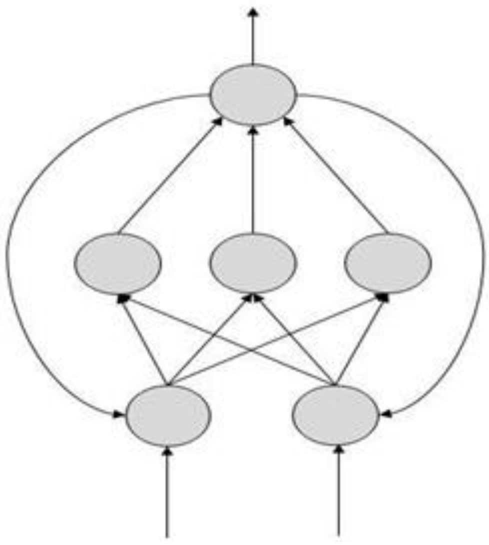
Feedforward ANN

- In this ANN, the information flow is unidirectional. A unit sends information to another unit from which it does not receive any information. There are no feedback loops.
- ***They are used in pattern generation/recognition/classification.***
- They have fixed inputs and outputs.



Feedback ANN

Here, feedback loops are allowed. They are used in content-addressable memories.



Activation Functions and Their Types?

What is Activation Function?

It’s just a thing function that you use to get the output of the node. It is also known as **Transfer Function**.

- *Activation functions* are important for an Artificial Neural Network to learn and make sense of something reallocated and Non-linear complex functional mappings between the inputs and response variable.
- *They introduce non-linear properties to our Network. Their main purpose is to convert an input signal of a node in an ANN to an output signal.*
- That output signal now is used as an input in the next layer in the stack.

- Specifically, in A-NN we do the sum of products of inputs(**X**) and their corresponding Weights (**W**) and apply an Activation function **f(x)** to it to get the output of that layer and feed it as an input to the next layer.

Types of activation Functions?

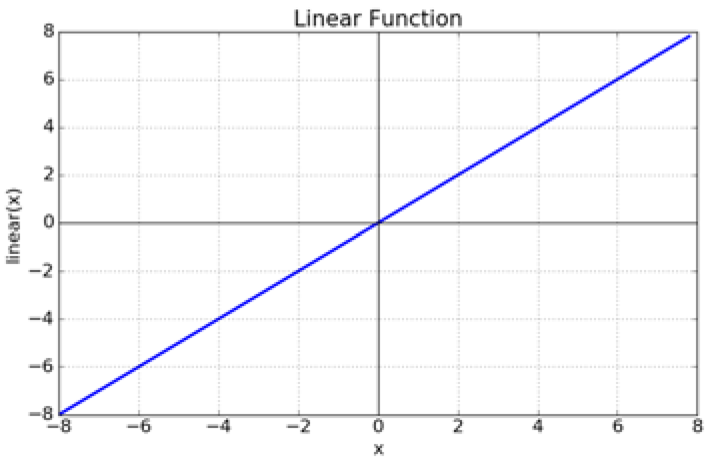
It is used to determine the output of neural network like yes or no. It maps the resulting values in between 0 to 1 or -1 to 1 etc. (depending upon the function).

The Activation Functions can be based on 2 types-

1. Linear Activation Function
2. Non-linear Activation Functions

Linear Activation Function

As you can see the function is a line or linear. Therefore, the output of the functions will not be confined between any range.



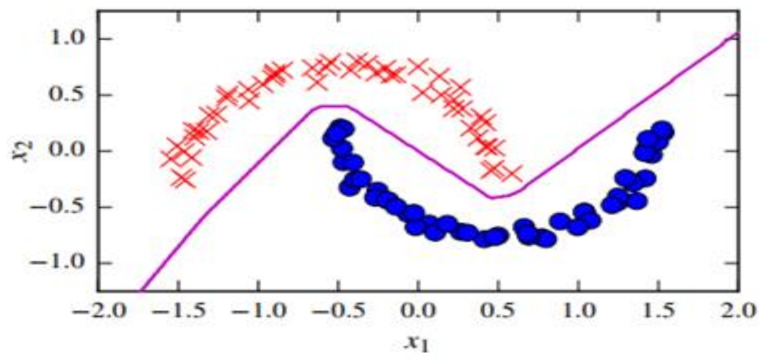
Equation: f(x) = x

Range: (-infinity to infinity)

It doesn’t help with the complexity of various parameters of usual data that is fed to the neural networks.

Non-linear Activation Function

The Nonlinear Activation Functions are the most used activation functions. Nonlinearity helps to makes the graph look something like this



It makes it easy for the model to generalize or adapt to a variety of data and to differentiate between the outputs.

The main terminologies needed to understand for nonlinear functions are:

Derivative or Differential: Change in y-axis w.r.t. change in the x-axis. It is also known as a slope.

Monotonic function: A function that is either entirely non-increasing or non-decreasing.

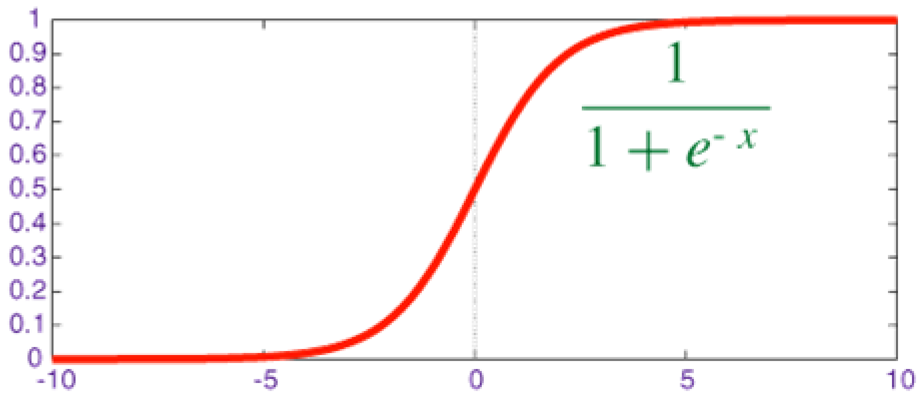
Meaning of monotonic: (*of a function or quantity) varying in such a way that it either never decreases or never increases.*)

The Nonlinear Activation Functions are mainly divided on based on **range or curves**

Different Types of Activation function in non-Linear

1. Sigmoid Activation Function

The Sigmoid Function curve looks like an S-shape.



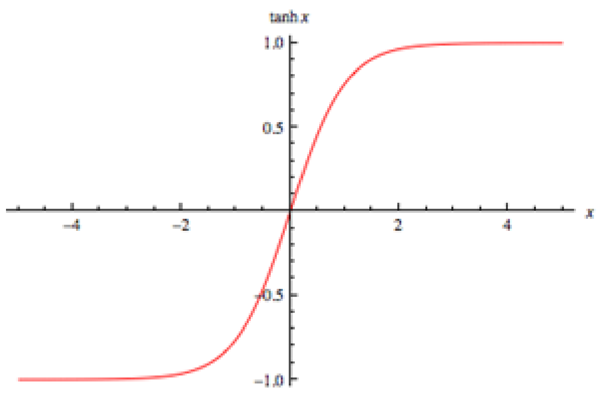
- The main reason why we use the sigmoid function is that it exists between **(0 to 1).**
- It is especially used for models where we have to **predict the probability** as an output.
- Since the probability of anything exists only between the range of **0 and 1,** sigmoid is the right choice.
- The function is **differentiable**. That means, we can find the slope of the sigmoid curve at any two points.

- The logistic sigmoid function can cause a neural network to get stuck at the training time.

The **SoftMax function** is a more generalized logistic activation function that is used for multiclass classification.

Tanh Activation Function

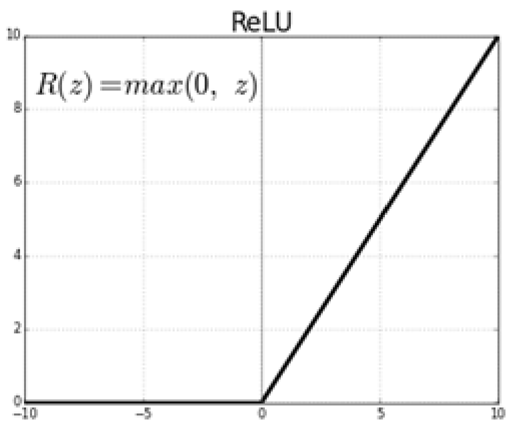
tanh is also like logistic sigmoid but better. The range of the tanh function is from (-1 to 1). tanh is also sigmoidal (s-shaped).



- The advantage is that the negative inputs will be mapped strongly negative and the zero inputs will be mapped near zero in the tanh graph. The function is **differentiable**.
- The function is **monotonic** while its **derivative is not monotonic**.
- The tanh function is mainly used classification between two classes.
- Both tanh and logistic sigmoid activation functions are used in feed-forward nets.

ReLU (Rectified Linear Unit) Activation Function

- The ReLU is the most used activation function in the world right now. Since it is used in almost all the convolutional neural networks or deep learning.



- As you can see, the ReLU is half rectified (from bottom).
- f(z) is zero when z is less than zero and f(z) is equal to z when z is above or equal to zero.

Range: [0 to infinity]

- The function and it is derivative **both are monotonic**.
- But the issue is that all the negative values become zero immediately which decreases the ability of the model to fit or train from the data properly.
- That means any negative input given to the ReLU activation function turns the value into zero immediately in the graph, which in turn affects the resulting graph by not mapping the negative values appropriately.

How Backpropagation works?

Let's dive into the mathematics part and see the backpropagation working with feedforward and feed backward Neural Network in depth.

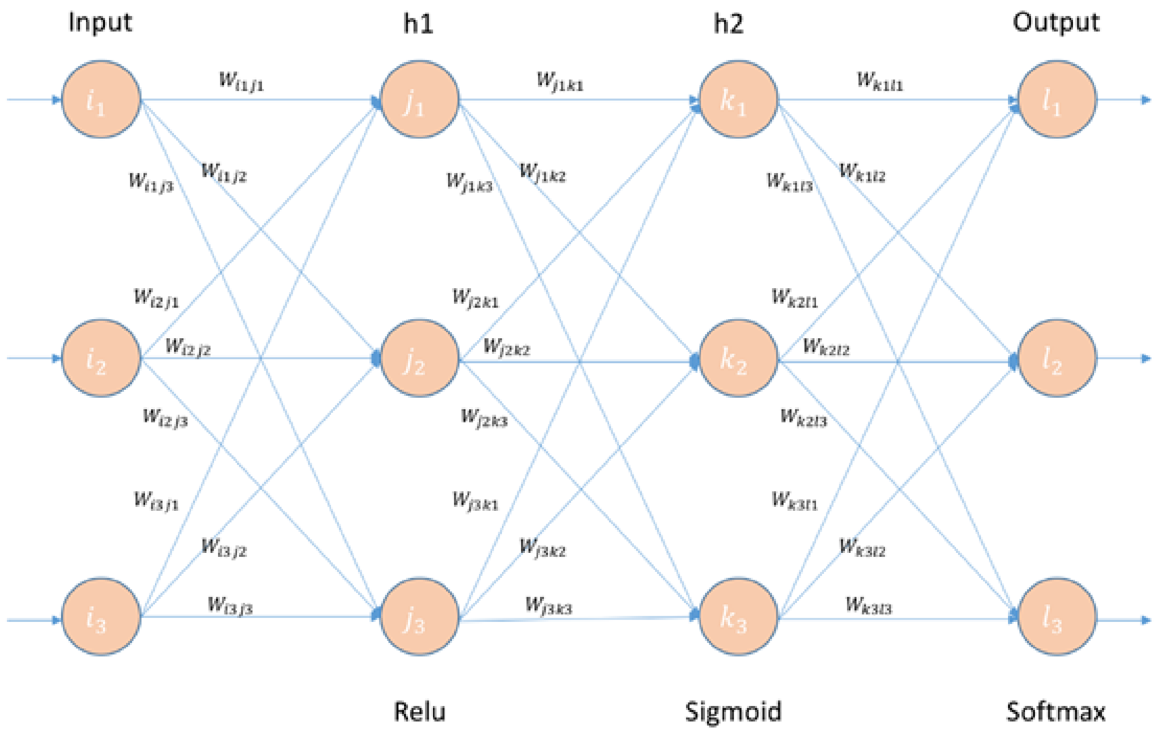
Approach

Step 1: Randomly initialize the Weights to a small number close to 0 (but not 0).
Step 2: Input the first observation of your dataset in the input layer, each feature in one input node.
Step 3: Forward-Propagation: from left to right, the neurons are activated in a way that the impact of each neuron's activation is limited by the weights. Propagates the activations until getting the predicted result y.
Step 4: Compare the predicted result to the actual result. Measure the generated error.
Step 5: Back-Propagation: from left to right, the error is back-Propagated. Update the weights according to how much they are responsible for the error. The Learning rate decides how much we update the weights.

Step 6: Repeat step 1 to step 5 and updates the weights after each observation.
Step 7: When the whole training set passed through the ANN that makes an epoch. Redo more epoch.

Architecture

- Build a Feed-Forward neural network with 2 hidden layers. All the layers will have 3 Neurons each.
- 1st and 2nd hidden layers will have RELU and sigmoid respectively as activation functions. The final layer will have a SoftMax activation function.
- Error is calculated using cross-entropy.



How Does Backpropagation work?

Inputs

$$\text{Inputs} = [i_1 \quad i_2 \quad i_3]$$

$$= [0.1 \quad 0.3 \quad 0.5]$$

Actual Outputs

$$\text{Outputs} = [y_1 \quad y_2 \quad y_3]$$

$$= [0.0 \quad 1.0 \quad 0.0]$$

Randomly initialize the Weights to small number close to 0 (but not 0)

Weights for Layer 1

$$W_{ij} = \begin{bmatrix} W_{i1j1} & W_{i1j2} & W_{i1j3} \\ W_{i2j1} & W_{i2j2} & W_{i2j3} \\ W_{i3j1} & W_{i3j2} & W_{i3j3} \end{bmatrix}$$

$$W_{ij} = \begin{bmatrix} 0.2 & 0.4 & 0.5 \\ 0.3 & 0.7 & 0.1 \\ 0.2 & 0.6 & 0.9 \end{bmatrix}$$

Weights for Layer 2

$$W_{jk} = \begin{bmatrix} W_{j1k1} & W_{j1k2} & W_{j1k3} \\ W_{j2k1} & W_{j2k2} & W_{j2k3} \\ W_{j3k1} & W_{j3k2} & W_{j3k3} \end{bmatrix}$$

$$W_{jk} = \begin{bmatrix} 0.4 & 0.6 & 0.1 \\ 0.2 & 0.7 & 0.3 \\ 0.8 & 0.1 & 0.5 \end{bmatrix}$$

Weights for Layer 3

$$W_{kl} = \begin{bmatrix} W_{k1l1} & W_{k1l2} & W_{k1l3} \\ W_{k2l1} & W_{k2l2} & W_{k2l3} \\ W_{k3l1} & W_{k3l2} & W_{k3l3} \end{bmatrix}$$

$$W_{kl} = \begin{bmatrix} 0.2 & 0.5 & 0.3 \\ 0.4 & 0.6 & 0.8 \\ 0.1 & 0.9 & 0.7 \end{bmatrix}$$

Feedfarwad Neural Network: Forward-Propagation: from left to right, the neurons are activated in a way that the impact of each neuron's activation is limited by the weights. Propagates the activations until getting the predicted result y.

Layer 1 Matrix Operation

$$= [i_1 \quad i_2 \quad i_3] \times \begin{bmatrix} W_{i1j1} & W_{i1j2} & W_{i1j3} \\ W_{i2j1} & W_{i2j2} & W_{i2j3} \\ W_{i3j1} & W_{i3j2} & W_{i3j3} \end{bmatrix} + [b_{j1} \quad b_{j2} \quad b_{j3}]$$

$$= [0.1 \quad 0.3 \quad 0.5] \times \begin{bmatrix} 0.2 & 0.4 & 0.5 \\ 0.3 & 0.7 & 0.1 \\ 0.2 & 0.6 & 0.9 \end{bmatrix} + [1.0 \quad 1.0 \quad 1.0]$$

$$[h1_{in1} \quad h1_{in2} \quad h1_{in3}] = [1.2 \quad 1.6 \quad 1.5]$$

RELU operation on layer 1

$$\text{RELU} = \max(0, x)$$

$$[h1_{out1} \quad h1_{out2} \quad h1_{out3}] = [\max(0, h1_{in1}) \quad \max(0, h1_{in2}) \quad \max(0, h1_{in3})]$$

$$[h1_{out1} \quad h1_{out2} \quad h1_{out3}] = [1.2 \quad 1.6 \quad 1.5]$$

Layer 2 Matrix Operation

$$= [h1_{out1} \quad h1_{out2} \quad h1_{out3}] \times \begin{bmatrix} W_{j1k1} & W_{j1k2} & W_{j1k3} \\ W_{j2k1} & W_{j2k2} & W_{j2k3} \\ W_{j3k1} & W_{j3k2} & W_{j3k3} \end{bmatrix} + [b_{k1} \quad b_{k2} \quad b_{k3}]$$

$$= [1.2 \quad 1.6 \quad 1.5] \times \begin{bmatrix} 0.4 & 0.6 & 0.1 \\ 0.2 & 0.7 & 0.3 \\ 0.8 & 0.1 & 0.5 \end{bmatrix} + [1.0 \quad 1.0 \quad 1.0]$$

$$[h2_{in1} \quad h2_{in2} \quad h2_{in3}] = [3.0 \quad 3.0 \quad 2.4]$$

Sigmoid Operation on layer 2

$$\text{Sigmoid} = 1 \,/\, (1 + e^{-x})$$

$$[h2_{out1} \quad h2_{out2} \quad h2_{out3}] = [1/(1+e^{-h2_{in1}}) \quad 1/(1+e^{-h2_{in2}}) \quad 1/(1+e^{-h2_{in3}})]$$

$$[h2_{out1} \quad h2_{out2} \quad h2_{out3}] = [1/(1+e^{-3.0}) \quad 1/(1+e^{-3.0}) \quad 1/(1+e^{-2.4})]$$

$$[h2_{out1} \quad h2_{out2} \quad h2_{out3}] = [0.952 \quad 0.952 \quad 0.916]$$

Layer 3 Matrix Operation

$$= [h2_{out1} \quad h2_{out2} \quad h2_{out3}] \times \begin{bmatrix} W_{k1l1} & W_{k1l2} & W_{k1l3} \\ W_{k2l1} & W_{k2l2} & W_{k2l3} \\ W_{k3l1} & W_{k3l2} & W_{k3l3} \end{bmatrix} + [b_{l1} \quad b_{l2} \quad b_{l3}]$$

$$= [0.952 \quad 0.952 \quad 0.916] \times \begin{bmatrix} 0.2 & 0.5 & 0.3 \\ 0.4 & 0.6 & 0.8 \\ 0.1 & 0.9 & 0.7 \end{bmatrix} + [1.0 \quad 1.0 \quad 1.0]$$

$$[o_{in1} \quad o_{in2} \quad o_{in3}] = [1.7 \quad 2.9 \quad 2.7]$$

Softmax operation on layer 3

$$\text{Softmax} = e^{o_{ina}} / (\textstyle\sum_{a=1}^{3} e^{o_{ina}})$$

$$[o_{out1} \quad o_{out2} \quad o_{out3}] = \left[e^{o_{ina}} / (\sum_{a=1}^{3} e^{o_{ina}}) \quad e^{o_{ina}} / (\sum_{a=1}^{3} e^{o_{ina}}) \quad e^{o_{ina}} / (\sum_{a=1}^{3} e^{o_{ina}}) \right]$$

$$[o_{out1} \quad o_{out2} \quad o_{out3}] = [0.1420 \quad 0.4717 \quad 0.3862]$$

The actual output is [0.0 1.0 0.0] but we got [0.1420 0.4717 0.3862]

We have done the forward pass

Compare the predicted result to the actual result. Measure the generated error.

Logistic Loss

Log Loss = $-y_i * \log(o_{outi}) - (1 - y_i) * \log(1 - o_{outi})$

$$\begin{aligned} E_1 &= -y_1 * \log(o_{out1}) - (1 - y_1) * \log(1 - o_{out1}) \\ &= -0 * \log(0.1420) - (1 - 0) * \log(1 - 0.1420) \\ &= -0 * \log(0.1420) - 1 * \log(1 - 0.1420) \\ &= -\log(0.858) \\ &= +\, 0.0665 \end{aligned}$$

$$\begin{aligned} E_2 &= -y_2 * \log(o_{out2}) - (1 - y_2) * \log(1 - o_{out2}) \\ &= -1 * \log(0.4717) - (1 - 1) * \log(1 - 0.4717) \\ &= -1 * \log(0.4717) - 0 \\ &= -\log(0.4717) \\ &= +0.3263 \end{aligned}$$

$$\begin{aligned} E_3 &= -y_3 * \log(o_{out3}) - (1 - y_3) * \log(1 - o_{out3}) \\ &= -0 * \log(0.3862) - (1 - 0) * \log(1 - 0.3862) \\ &= -0 * \log(0.3862) - 1 * \log(1 - 0.3862) \\ &= -\log(0.6138) \\ &= +0.2119 \end{aligned}$$

Cost Function

$$= -\frac{1}{N}\left[\sum_{i=1}^{N} y_i * \log(o_{outi}) + (1-y_i) * \log(1-o_{outi})\right]$$

$$= -\frac{1}{3}\left[\sum_{i=1}^{3} y_i * \log(o_{outi}) + (1-y_i) * \log(1-o_{outi})\right]$$

$$= \frac{0.0665+0.3263+0.2119}{3}$$

$$=0.2015$$

Feed Backward Neural Network: Back-Propagation: from left to right, the error is back-Propagated. Update the weights according to how much they are responsible for the error. The Learning rate is decides how much we updates the weights.

Back Propagating the error — (Hidden Layer 2 — Output Layer) Weights

Derivative of cross entropy

$$\frac{\partial E_i}{\partial o_{outi}} = \frac{-y_i * \log(o_{outi}) - (1-y_i) * \log(1-o_{outi})}{\partial o_{outi}}$$

$$\frac{\partial E_i}{\partial o_{outi}} = -y_i * (1/o_{outi}) - (1-y_i) * (1/(1-o_{outi}))$$

$$\begin{bmatrix} \frac{\partial E_1}{\partial o_{out1}} \\ \frac{\partial E_2}{\partial o_{out2}} \\ \frac{\partial E_3}{\partial o_{out3}} \end{bmatrix} = \begin{bmatrix} -y_1 * (1/o_{out1}) - (1-y_1) * (1/(1-o_{out1})) \\ -y_2 * (1/o_{out2}) - (1-y_2) * (1/(1-o_{out2})) \\ -y_3 * (1/o_{out3}) - (1-y_3) * (1/(1-o_{out3})) \end{bmatrix}$$

$$\begin{bmatrix} \frac{\partial E_1}{\partial o_{out1}} \\ \frac{\partial E_2}{\partial o_{out2}} \\ \frac{\partial E_3}{\partial o_{out3}} \end{bmatrix} = \begin{bmatrix} -1.1655 \\ -2.1199 \\ -1.6291 \end{bmatrix}$$

Derivative of Softmax layer

$$\frac{\partial o_{out1}}{\partial o_{in1}} = \frac{\partial (e^{o_{in1}}/(e^{o_{in1}} + e^{o_{in2}} + e^{o_{in3}}))}{\partial o_{in1}}$$

$$\begin{bmatrix} \frac{\partial o_{out1}}{\partial o_{in1}} \\ \frac{\partial o_{out2}}{\partial o_{in2}} \\ \frac{\partial o_{out3}}{\partial o_{in3}} \end{bmatrix} = \begin{bmatrix} (e^{o_{in1}} * (e^{o_{in2}} + e^{o_{in3}}))/(e^{o_{in1}} + e^{o_{in2}} + e^{o_{in3}})^2 \\ (e^{o_{in2}} * (e^{o_{in1}} + e^{o_{in3}}))/(e^{o_{in1}} + e^{o_{in2}} + e^{o_{in3}})^2 \\ (e^{o_{in3}} * (e^{o_{in1}} + e^{o_{in2}}))/(e^{o_{in1}} + e^{o_{in2}} + e^{o_{in3}})^2 \end{bmatrix}$$

$$\begin{bmatrix} \frac{\partial o_{out1}}{\partial o_{in1}} \\ \frac{\partial o_{out2}}{\partial o_{in2}} \\ \frac{\partial o_{out3}}{\partial o_{in3}} \end{bmatrix} = \begin{bmatrix} 0.1218 \\ 0.2492 \\ 0.2370 \end{bmatrix}$$

$$\frac{\partial o_{in1}}{\partial W_{k1l1}} = \frac{\partial ((h2_{out1} * W_{k1l1}) + (h2_{out2} * W_{k2l1}) + (h2_{out3} * W_{k3l1}) + b_{l1})}{\partial W_{k1l1}}$$

$$\frac{\partial o_{in1}}{\partial W_{k1l1}} = h2_{out1}$$

$$\frac{\partial o_{in1}}{\partial W_{k2l1}} = h2_{out2}$$

$$\frac{\partial o_{in1}}{\partial W_{k3l1}} = \frac{\partial((h2_{out1} * W_{k1l1}) + (h2_{out2} * W_{k2l1}) + (h2_{out3} * W_{k3l1}) + b_{l1})}{\partial W_{k3l1}}$$

$$\frac{\partial o_{in1}}{\partial W_{k3l1}} = h2_{out3}$$

$$\frac{\partial o_{in2}}{\partial W_{k1l2}} = \frac{\partial((h2_{out1} * W_{k1l2}) + (h2_{out2} * W_{k2l2}) + (h2_{out3} * W_{k3l2}) + b_{l2})}{\partial W_{k1l2}}$$

$$\frac{\partial o_{in2}}{\partial W_{k1l2}} = h2_{out1}$$

$$\frac{\partial o_{in2}}{\partial W_{k2l2}} = \frac{\partial((h2_{out1} * W_{k1l2}) + (h2_{out2} * W_{k2l2}) + (h2_{out3} * W_{k3l2}) + b_{l2})}{\partial W_{k2l2}}$$

$$\frac{\partial o_{in2}}{\partial W_{k2l2}} = h2_{out2}$$

$$\frac{\partial o_{in2}}{\partial W_{k3l2}} = \frac{\partial((h2_{out1} * W_{k1l2}) + (h2_{out2} * W_{k2l2}) + (h2_{out3} * W_{k3l2}) + b_{l2})}{\partial W_{k3l2}}$$

$$\frac{\partial o_{in2}}{\partial W_{k3l2}} = h2_{out3}$$

$$\frac{\partial o_{in2}}{\partial W_{k3l2}} = \frac{\partial((h2_{out1} * W_{k1l2}) + (h2_{out2} * W_{k2l2}) + (h2_{out3} * W_{k3l2}) + b_{l2})}{\partial W_{k3l2}}$$

$$\frac{\partial o_{in2}}{\partial W_{k3l2}} = h2_{out3}$$

$$\frac{\partial o_{in3}}{\partial W_{k1l3}} = \frac{\partial((h2_{out1} * W_{k1l3}) + (h2_{out2} * W_{k2l3}) + (h2_{out3} * W_{k3l3}) + b_{l3})}{\partial W_{k1l3}}$$

$$\frac{\partial o_{in3}}{\partial W_{k1l3}} = h2_{out1}$$

$$\frac{\partial o_{in3}}{\partial W_{k2l3}} = \frac{\partial((h2_{out1} * W_{k1l3}) + (h2_{out2} * W_{k2l3}) + (h2_{out3} * W_{k3l3}) + b_{l3})}{\partial W_{k2l3}}$$

$$\frac{\partial o_{in3}}{\partial W_{k2l3}} = h2_{out2}$$

$$\frac{\partial o_{in3}}{\partial W_{k3l3}} = \frac{\partial((h2_{out1} * W_{k1l3}) + (h2_{out2} * W_{k2l3}) + (h2_{out3} * W_{k3l3}) + b_{l3})}{\partial W_{k3l3}}$$

$$\frac{\partial o_{in3}}{\partial W_{k3l3}} = h2_{out3}$$

$$\frac{\partial E_1}{\partial W_{k1l1}} = \frac{\partial E_1}{\partial o_{out1}} * \frac{\partial o_{out1}}{\partial o_{in1}} * \frac{\partial o_{in1}}{\partial W_{k1l1}}$$

$$\frac{\partial E_1}{\partial W_{k2l1}} = \frac{\partial E_1}{\partial o_{out1}} * \frac{\partial o_{out1}}{\partial o_{in1}} * \frac{\partial o_{in1}}{\partial W_{k2l1}}$$

$$\frac{\partial E_1}{\partial W_{k3l1}} = \frac{\partial E_1}{\partial o_{out1}} * \frac{\partial o_{out1}}{\partial o_{in1}} * \frac{\partial o_{in1}}{\partial W_{k3l1}}$$

$$\frac{\partial E_2}{\partial W_{k1l2}} = \frac{\partial E_2}{\partial o_{out2}} * \frac{\partial o_{out2}}{\partial o_{in2}} * \frac{\partial o_{in2}}{\partial W_{k1l2}}$$

$$\frac{\partial E_2}{\partial W_{k2l2}} = \frac{\partial E_2}{\partial o_{out2}} * \frac{\partial o_{out2}}{\partial o_{in2}} * \frac{\partial o_{in2}}{\partial W_{k2l2}}$$

$$\frac{\partial E_2}{\partial W_{k3l2}} = \frac{\partial E_2}{\partial o_{out2}} * \frac{\partial o_{out2}}{\partial o_{in2}} * \frac{\partial o_{in2}}{\partial W_{k3l2}}$$

$$\frac{\partial E_3}{\partial W_{k1l3}} = \frac{\partial E_3}{\partial o_{out3}} * \frac{\partial o_{out3}}{\partial o_{in3}} * \frac{\partial o_{in3}}{\partial W_{k1l3}}$$

$$\frac{\partial E_3}{\partial W_{k2l3}} = \frac{\partial E_3}{\partial o_{out3}} * \frac{\partial o_{out3}}{\partial o_{in3}} * \frac{\partial o_{in3}}{\partial W_{k2l3}}$$

$$\frac{\partial E_3}{\partial W_{k3l3}} = \frac{\partial E_3}{\partial o_{out3}} * \frac{\partial o_{out3}}{\partial o_{in3}} * \frac{\partial o_{in3}}{\partial W_{k3l3}}$$

$$\partial W_{kl} = \begin{bmatrix} \frac{\partial E_1}{\partial W_{k1l1}} & \frac{\partial E_2}{\partial W_{k1l2}} & \frac{\partial E_3}{\partial W_{k1l3}} \\ \frac{\partial E_1}{\partial W_{k2l1}} & \frac{\partial E_2}{\partial W_{k2l2}} & \frac{\partial E_3}{\partial W_{k2l3}} \\ \frac{\partial E_1}{\partial W_{k3l1}} & \frac{\partial E_2}{\partial W_{k3l2}} & \frac{\partial E_3}{\partial W_{k3l3}} \end{bmatrix}$$

$$\partial W_{kl} =$$
$$\begin{bmatrix} \frac{\partial E_1}{\partial o_{out1}} * \frac{\partial o_{out1}}{\partial o_{in1}} * \frac{\partial o_{in1}}{\partial W_{k1l1}} & \frac{\partial E_2}{\partial o_{out2}} * \frac{\partial o_{out2}}{\partial o_{in2}} * \frac{\partial o_{in2}}{\partial W_{k1l2}} & \frac{\partial E_3}{\partial o_{out3}} * \frac{\partial o_{out3}}{\partial o_{in3}} * \frac{\partial o_{in3}}{\partial W_{k1l3}} \\ \frac{\partial E_1}{\partial o_{out1}} * \frac{\partial o_{out1}}{\partial o_{in1}} * \frac{\partial o_{in1}}{\partial W_{k2l1}} & \frac{\partial E_2}{\partial o_{out2}} * \frac{\partial o_{out2}}{\partial o_{in2}} * \frac{\partial o_{in2}}{\partial W_{k2l2}} & \frac{\partial E_3}{\partial o_{out3}} * \frac{\partial o_{out3}}{\partial o_{in3}} * \frac{\partial o_{in3}}{\partial W_{k2l3}} \\ \frac{\partial E_1}{\partial o_{out1}} * \frac{\partial o_{out1}}{\partial o_{in1}} * \frac{\partial o_{in1}}{\partial W_{k3l1}} & \frac{\partial E_2}{\partial o_{out2}} * \frac{\partial o_{out2}}{\partial o_{in2}} * \frac{\partial o_{in2}}{\partial W_{k3l2}} & \frac{\partial E_3}{\partial o_{out3}} * \frac{\partial o_{out3}}{\partial o_{in3}} * \frac{\partial o_{in3}}{\partial W_{k3l3}} \end{bmatrix}$$

$$\partial W_{kl} =$$
$$\begin{bmatrix} -1.1655 * 0.1218 * 0.952 & -2.1199 * 0.2492 * 0.952 & -1.6291 * 0.2370 * 0.952 \\ -1.1655 * 0.1218 * 0.952 & -2.1199 * 0.2492 * 0.952 & -1.6291 * 0.2370 * 0.952 \\ -1.1655 * 0.1218 * 0.916 & -2.1199 * 0.2492 * 0.916 & -1.6291 * 0.2370 * 0.916 \end{bmatrix}$$

$$\partial W_{kl} = \begin{bmatrix} \partial W_{k1l1} & \partial W_{k1l2} & \partial W_{k1l3} \\ \partial W_{k2l1} & \partial W_{k2l2} & \partial W_{k2l3} \\ \partial W_{k3l1} & \partial W_{k3l2} & \partial W_{k3l3} \end{bmatrix}$$

$$\partial W_{kl} = \begin{bmatrix} -0.1351 & -0.5029 & -0.3675 \\ -0.1351 & -0.5029 & -0.3675 \\ -0.1300 & -0.4839 & -0.3536 \end{bmatrix}$$

$$W'_{kl} = \begin{bmatrix} W_{k1l1} - (lr * \partial W_{k1l1}) & W_{k1l2} - (lr * \partial W_{k1l2}) & W_{k1l3} - (lr * \partial W_{k1l3}) \\ W_{k2l1} - (lr * \partial W_{k2l1}) & W_{k2l2} - (lr * \partial W_{k2l2}) & W_{k2l3} - (lr * \partial W_{k2l3}) \\ W_{k3l1} - (lr * \partial W_{k3l1}) & W_{k3l2} - (lr * \partial W_{k3l2}) & W_{k3l3} - (lr * \partial W_{k3l3} \end{bmatrix}$$

$$= \begin{bmatrix} 0.2 - (0.01 * -0.1351) & 0.5 - (0.01 * -0.5029) & 0.3 - (0.01 * -0.3675) \\ 0.4 - (0.01 * -0.1351) & 0.6 - (0.01 * -0.5029) & 0.8 - (0.01 * -0.3675) \\ 0.1 - (0.01 * -0.1300) & 0.9 - (0.01 * -0.4839) & 0.7 - (0.01 * -0.3536) \end{bmatrix}$$

$$= \begin{bmatrix} -0.2013 & -0.5050 & -0.3036 \\ -0.4013 & -0.6050 & -0.8036 \\ -0.1013 & -0.9048 & -0.7035 \end{bmatrix}$$

Back Propagating the error — (Hidden Layer 1 — Hidden Layer 2) Weights

Derivative of Sigmoid Layer

$$\frac{\partial h2_{out1}}{\partial h2_{in1}} = \frac{\partial Sigmoid(h2_{in1})}{\partial h2_{in1}}$$

$$\frac{\partial h2_{out1}}{\partial h2_{in1}} = Sigmoid(h2_{in1}) * (1 - Sigmoid(h2_{in1}))$$

$$\begin{bmatrix} \frac{\partial h2_{out1}}{\partial h2_{in1}} \\ \frac{\partial h2_{out2}}{\partial h2_{in2}} \\ \frac{\partial h2_{out3}}{\partial h2_{in3}} \end{bmatrix} = \begin{bmatrix} Sigmoid(h2_{in1}) * (1 - Sigmoid(h2_{in1})) \\ Sigmoid(h2_{in2}) * (1 - Sigmoid(h2_{in2})) \\ Sigmoid(h2_{in3}) * (1 - Sigmoid(h2_{in3})) \end{bmatrix}$$

$$\begin{bmatrix} \frac{\partial h2_{out1}}{\partial h2_{in1}} \\ \frac{\partial h2_{out2}}{\partial h2_{in2}} \\ \frac{\partial h2_{out3}}{\partial h2_{in3}} \end{bmatrix} = \begin{bmatrix} 0.0451 \\ 0.0451 \\ 0.0762 \end{bmatrix}$$

$$\frac{\partial h2_{in1}}{\partial W_{j1k1}} = \frac{\partial((h1_{out1} * W_{j1k1}) + (h1_{out2} * W_{j2k1}) + (h1_{out3} * W_{j3k1}) + b_{k1})}{\partial W_{j1k1}}$$

$$\frac{\partial h2_{in1}}{\partial W_{j1k1}} = h1_{out1}$$

$$\frac{\partial h2_{in1}}{\partial W_{j2k1}} = \frac{\partial((h1_{out1} * W_{j1k1}) + (h1_{out2} * W_{j2k1}) + (h1_{out3} * W_{j3k1}) + b_{k1})}{\partial W_{j2k1}}$$

$$\frac{\partial h2_{in1}}{\partial W_{j2k1}} = h1_{out2}$$

$$\frac{\partial h2_{in1}}{\partial W_{j3k1}} = \frac{\partial((h1_{out1} * W_{j1k1}) + (h1_{out2} * W_{j2k1}) + (h1_{out3} * W_{j3k1}) + b_{k1})}{\partial W_{j1k1}}$$

$$\frac{\partial h2_{in1}}{\partial W_{j3k1}} = h1_{out3}$$

$$\frac{\partial h2_{in2}}{\partial W_{j1k2}} = \frac{\partial((h1_{out1} * W_{j1k2}) + (h1_{out2} * W_{j2k2}) + (h1_{out3} * W_{j3k2}) + b_{k2})}{\partial W_{j1k2}}$$

$$\frac{\partial h2_{in2}}{\partial W_{j1k2}} = h1_{out1}$$

$$\frac{\partial h2_{in2}}{\partial W_{j2k2}} = \frac{\partial((h1_{out1} * W_{j1k2}) + (h1_{out2} * W_{j2k2}) + (h1_{out3} * W_{j3k2}) + b_{k2})}{\partial W_{j2k2}}$$

$$\frac{\partial h2_{in2}}{\partial W_{j2k2}} = h1_{out2}$$

$$\frac{\partial h2_{in2}}{\partial W_{j3k2}} = \frac{\partial((h1_{out1} * W_{j1k2}) + (h1_{out2} * W_{j2k2}) + (h1_{out3} * W_{j3k2}) + b_{k2})}{\partial W_{j3k2}}$$

$$\frac{\partial h2_{in2}}{\partial W_{j3k2}} = h1_{out3}$$

$$\frac{\partial h2_{in3}}{\partial W_{j1k3}} = \frac{\partial((h1_{out1} * W_{j1k3}) + (h1_{out2} * W_{j2k3}) + (h1_{out3} * W_{j3k3}) + b_{k3})}{\partial W_{j1k3}}$$

$$\frac{\partial h2_{in3}}{\partial W_{j1k3}} = h1_{out1}$$

$$\frac{\partial h2_{in3}}{\partial W_{j2k3}} = \frac{\partial((h1_{out1} * W_{j1k3}) + (h1_{out2} * W_{j2k3}) + (h1_{out3} * W_{j3k3}) + b_{k3})}{\partial W_{j2k3}}$$

$$\frac{\partial h2_{in3}}{\partial W_{j2k3}} = h1_{out2}$$

$$\frac{\partial h2_{in3}}{\partial W_{j3k3}} = \frac{\partial((h1_{out1} * W_{j1k3}) + (h1_{out2} * W_{j2k3}) + (h1_{out3} * W_{j3k3}}{\partial W_{j3k3}}$$

$$\frac{\partial h2_{in3}}{\partial W_{j3k3}} = h1_{out3}$$

$$\frac{\partial E_{total}}{\partial W_{j1k1}} = \frac{\partial E_{total}}{\partial h2_{out1}} * \frac{\partial h2_{out1}}{\partial h2_{in1}} * \frac{\partial h2_{in1}}{\partial W_{j1k1}}$$

$$\frac{\partial E_{total}}{\partial W_{j2k1}} = \frac{\partial E_{total}}{\partial h2_{out1}} * \frac{\partial h2_{out1}}{\partial h2_{in1}} * \frac{\partial h2_{in1}}{\partial W_{j2k1}}$$

$$\frac{\partial E_{total}}{\partial W_{j3k1}} = \frac{\partial E_{total}}{\partial h2_{out1}} * \frac{\partial h2_{out1}}{\partial h2_{in1}} * \frac{\partial h2_{in1}}{\partial W_{j3k1}}$$

$$\frac{\partial E_{total}}{\partial W_{j1k2}} = \frac{\partial E_{total}}{\partial h2_{out2}} * \frac{\partial h2_{out2}}{\partial h2_{in2}} * \frac{\partial h2_{in2}}{\partial W_{j1k2}}$$

$$\frac{\partial E_{total}}{\partial W_{j2k2}} = \frac{\partial E_{total}}{\partial h2_{out2}} * \frac{\partial h2_{out2}}{\partial h2_{in2}} * \frac{\partial h2_{in2}}{\partial W_{j2k2}}$$

$$\frac{\partial E_{total}}{\partial W_{j2k2}} = \frac{\partial E_{total}}{\partial h2_{out2}} * \frac{\partial h2_{out2}}{\partial h2_{in2}} * \frac{\partial h2_{in2}}{\partial W_{j2k2}}$$

$$\frac{\partial E_{total}}{\partial W_{j3k2}} = \frac{\partial E_{total}}{\partial h2_{out2}} * \frac{\partial h2_{out2}}{\partial h2_{in2}} * \frac{\partial h2_{in2}}{\partial W_{j3k2}}$$

$$\frac{\partial E_{total}}{\partial W_{j1k3}} = \frac{\partial E_{total}}{\partial h2_{out3}} * \frac{\partial h2_{out3}}{\partial h2_{in3}} * \frac{\partial h2_{in3}}{\partial W_{j1k3}}$$

$$\frac{\partial E_{total}}{\partial W_{j2k3}} = \frac{\partial E_{total}}{\partial h2_{out3}} * \frac{\partial h2_{out3}}{\partial h2_{in3}} * \frac{\partial h2_{in3}}{\partial W_{j2k3}}$$

$$\frac{\partial E_{total}}{\partial W_{j3k3}} = \frac{\partial E_{total}}{\partial h2_{out3}} * \frac{\partial h2_{out3}}{\partial h2_{in3}} * \frac{\partial h2_{in3}}{\partial W_{j3k3}}$$

$$\partial W_{jk} = \begin{bmatrix} \frac{\partial E_{total}}{\partial W_{j1k1}} & \frac{\partial E_{total}}{\partial W_{j1k2}} & \frac{\partial E_{total}}{\partial W_{j1k3}} \\ \frac{\partial E_{total}}{\partial W_{j2k1}} & \frac{\partial E_{total}}{\partial W_{j2k2}} & \frac{\partial E_{total}}{\partial W_{j2k3}} \\ \frac{\partial E_{total}}{\partial W_{j3k1}} & \frac{\partial E_{total}}{\partial W_{j3k2}} & \frac{\partial E_{total}}{\partial W_{j3k3}} \end{bmatrix}$$

$$\partial W_{jk} = \begin{bmatrix} \frac{\partial E_{total}}{\partial h2_{out1}} * \frac{\partial h2_{out1}}{\partial h2_{in1}} * \frac{\partial h2_{in1}}{\partial W_{j1k1}} & \frac{\partial E_{total}}{\partial h2_{out2}} * \frac{\partial h2_{out2}}{\partial h2_{in2}} * \frac{\partial h2_{in2}}{\partial W_{j1k2}} & \frac{\partial E_{total}}{\partial h2_{out3}} * \frac{\partial h2_{out3}}{\partial h2_{in3}} * \frac{\partial h2_{in3}}{\partial W_{j1k3}} \\ \frac{\partial E_{total}}{\partial h2_{out1}} * \frac{\partial h2_{out1}}{\partial h2_{in1}} * \frac{\partial h2_{in1}}{\partial W_{j2k1}} & \frac{\partial E_{total}}{\partial h2_{out2}} * \frac{\partial h2_{out2}}{\partial h2_{in2}} * \frac{\partial h2_{in2}}{\partial W_{j2k2}} & \frac{\partial E_{total}}{\partial h2_{out3}} * \frac{\partial h2_{out3}}{\partial h2_{in3}} * \frac{\partial h2_{in3}}{\partial W_{j2k3}} \\ \frac{\partial E_{total}}{\partial h2_{out1}} * \frac{\partial h2_{out1}}{\partial h2_{in1}} * \frac{\partial h2_{in1}}{\partial W_{j3k1}} & \frac{\partial E_{total}}{\partial h2_{out2}} * \frac{\partial h2_{out2}}{\partial h2_{in2}} * \frac{\partial h2_{in2}}{\partial W_{j3k2}} & \frac{\partial E_{total}}{\partial h2_{out3}} * \frac{\partial h2_{out3}}{\partial h2_{in3}} * \frac{\partial h2_{in3}}{\partial W_{j3k3}} \end{bmatrix}$$

$$\frac{\partial E_{total}}{\partial h2_{out1}} = \frac{\partial E_1}{\partial h2_{out1}} * \frac{\partial E_2}{\partial h2_{out1}} * \frac{\partial E_3}{\partial h2_{out1}}$$

$$\frac{\partial E_1}{\partial h2_{out1}} = \frac{\partial E_1}{\partial o_{out1}} * \frac{\partial o_{out1}}{\partial o_{in1}} * \frac{\partial o_{in1}}{\partial h2_{out1}}$$

$$\frac{\partial E_2}{\partial h2_{out1}} = \frac{\partial E_2}{\partial o_{out2}} * \frac{\partial o_{out2}}{\partial o_{in2}} * \frac{\partial o_{in2}}{\partial h2_{out1}}$$

$$\frac{\partial E_3}{\partial h2_{out1}} = \frac{\partial E_3}{\partial o_{out3}} * \frac{\partial o_{out3}}{\partial o_{in3}} * \frac{\partial o_{in3}}{\partial h2_{out1}}$$

$$\frac{\partial E_{total}}{\partial h2_{out2}} = \frac{\partial E_1}{\partial h2_{out2}} * \frac{\partial E_2}{\partial h2_{out2}} * \frac{\partial E_3}{\partial h2_{out2}}$$

$$\frac{\partial E_1}{\partial h2_{out1}} = \frac{\partial E_1}{\partial o_{out1}} * \frac{\partial o_{out1}}{\partial o_{in1}} * \frac{\partial o_{in1}}{\partial h2_{out1}}$$

$$\frac{\partial E_2}{\partial h2_{out1}} = \frac{\partial E_2}{\partial o_{out2}} * \frac{\partial o_{out2}}{\partial o_{in2}} * \frac{\partial o_{in2}}{\partial h2_{out1}}$$

$$\frac{\partial E_3}{\partial h2_{out1}} = \frac{\partial E_3}{\partial o_{out3}} * \frac{\partial o_{out3}}{\partial o_{in3}} * \frac{\partial o_{in3}}{\partial h2_{out1}}$$

$$\frac{\partial E_{total}}{\partial h2_{out2}} = \frac{\partial E_1}{\partial h2_{out2}} * \frac{\partial E_2}{\partial h2_{out2}} * \frac{\partial E_3}{\partial h2_{out2}}$$

$$\frac{\partial E_1}{\partial h2_{out2}} = \frac{\partial E_1}{\partial o_{out1}} * \frac{\partial o_{out1}}{\partial o_{in1}} * \frac{\partial o_{in1}}{\partial h2_{out2}}$$

$$\frac{\partial E_2}{\partial h2_{out2}} = \frac{\partial E_2}{\partial o_{out2}} * \frac{\partial o_{out2}}{\partial o_{in2}} * \frac{\partial o_{in2}}{\partial h2_{out2}}$$

$$\frac{\partial E_3}{\partial h2_{out2}} = \frac{\partial E_3}{\partial o_{out3}} * \frac{\partial o_{out3}}{\partial o_{in3}} * \frac{\partial o_{in3}}{\partial h2_{out2}}$$

$$\frac{\partial E_{total}}{\partial h2_{out3}} = \frac{\partial E_1}{\partial h2_{out3}} * \frac{\partial E_2}{\partial h2_{out3}} * \frac{\partial E_3}{\partial h2_{out3}}$$

$$\frac{\partial E_1}{\partial h2_{out3}} = \frac{\partial E_1}{\partial o_{out1}} * \frac{\partial o_{out1}}{\partial o_{in1}} * \frac{\partial o_{in1}}{\partial h2_{out3}}$$

$$\frac{\partial E_2}{\partial h2_{out3}} = \frac{\partial E_2}{\partial o_{out2}} * \frac{\partial o_{out2}}{\partial o_{in2}} * \frac{\partial o_{in2}}{\partial h2_{out3}}$$

$$\frac{\partial E_3}{\partial h2_{out3}} = \frac{\partial E_3}{\partial o_{out3}} * \frac{\partial o_{out3}}{\partial o_{in3}} * \frac{\partial o_{in3}}{\partial h2_{out3}}$$

$$\begin{bmatrix} \frac{\partial E_{total}}{\partial h2_{out1}} \\ \frac{\partial E_{total}}{\partial h2_{out2}} \\ \frac{\partial E_{total}}{\partial h2_{out3}} \end{bmatrix} = \begin{bmatrix} \left(\frac{\partial E_1}{\partial o_{out1}} * \frac{\partial o_{out1}}{\partial o_{in1}} * \frac{\partial o_{in1}}{\partial h2_{out1}}\right) + \left(\frac{\partial E_2}{\partial o_{out2}} * \frac{\partial o_{out2}}{\partial o_{in2}} * \frac{\partial o_{in2}}{\partial h2_{out1}}\right) + (\frac{\partial E_3}{\partial o_{out3}} * \frac{\partial o_{out3}}{\partial o_{in3}} * \frac{\partial o_{in3}}{\partial h2_{out1}}) \\ \left(\frac{\partial E_1}{\partial o_{out1}} * \frac{\partial o_{out1}}{\partial o_{in1}} * \frac{\partial o_{in1}}{\partial h2_{out2}}\right) + \left(\frac{\partial E_2}{\partial o_{out2}} * \frac{\partial o_{out2}}{\partial o_{in2}} * \frac{\partial o_{in2}}{\partial h2_{out2}}\right) + (\frac{\partial E_3}{\partial o_{out3}} * \frac{\partial o_{out3}}{\partial o_{in3}} * \frac{\partial o_{in3}}{\partial h2_{out2}}) \\ \left(\frac{\partial E_1}{\partial o_{out1}} * \frac{\partial o_{out1}}{\partial o_{in1}} * \frac{\partial o_{in1}}{\partial h2_{out3}}\right) + (\frac{\partial E_2}{\partial o_{out2}} * \frac{\partial o_{out2}}{\partial o_{in2}} * \frac{\partial o_{in2}}{\partial h2_{out3}}) + (\frac{\partial E_3}{\partial o_{out3}} * \frac{\partial o_{out3}}{\partial o_{in3}} * \frac{\partial o_{in3}}{\partial h2_{out3}}) \end{bmatrix}$$

$$\begin{bmatrix} \frac{\partial o_{in1}}{\partial h2_{out1}} & \frac{\partial o_{in2}}{\partial h2_{out1}} & \frac{\partial o_{in3}}{\partial h2_{out1}} \\ \frac{\partial o_{in1}}{\partial h2_{out2}} & \frac{\partial o_{in2}}{\partial h2_{out2}} & \frac{\partial o_{in3}}{\partial h2_{out2}} \\ \frac{\partial o_{in1}}{\partial h2_{out3}} & \frac{\partial o_{in2}}{\partial h2_{out3}} & \frac{\partial o_{in3}}{\partial h2_{out3}} \end{bmatrix} = \begin{bmatrix} W_{k1l1} & W_{k1l2} & W_{k1l3} \\ W_{k2l1} & W_{k2l2} & W_{k2l3} \\ W_{k3l1} & W_{k3l2} & W_{k3l3} \end{bmatrix}$$

$$\begin{bmatrix} \frac{\partial E_{total}}{\partial h2_{out1}} \\ \frac{\partial E_{total}}{\partial h2_{out2}} \\ \frac{\partial E_{total}}{\partial h2_{out3}} \end{bmatrix} = \begin{bmatrix} \left(\frac{\partial E_1}{\partial o_{out1}} * \frac{\partial o_{out1}}{\partial o_{in1}} * W_{k1l1}\right) + \left(\frac{\partial E_2}{\partial o_{out2}} * \frac{\partial o_{out2}}{\partial o_{in2}} * W_{k1l2}\right) + (\frac{\partial E_3}{\partial o_{out3}} * \frac{\partial o_{out3}}{\partial o_{in3}} * W_{k1l3}) \\ \left(\frac{\partial E_1}{\partial o_{out1}} * \frac{\partial o_{out1}}{\partial o_{in1}} * W_{k2l1}\right) + \left(\frac{\partial E_2}{\partial o_{out2}} * \frac{\partial o_{out2}}{\partial o_{in2}} * W_{k2l2}\right) + (\frac{\partial E_3}{\partial o_{out3}} * \frac{\partial o_{out3}}{\partial o_{in3}} * W_{k2l3}) \\ \left(\frac{\partial E_1}{\partial o_{out1}} * \frac{\partial o_{out1}}{\partial o_{in1}} * W_{k3l1}\right) + (\frac{\partial E_2}{\partial o_{out2}} * \frac{\partial o_{out2}}{\partial o_{in2}} * W_{k3l2}) + (\frac{\partial E_3}{\partial o_{out3}} * \frac{\partial o_{out3}}{\partial o_{in3}} * W_{k3l3}) \end{bmatrix}$$

$$= \begin{bmatrix} (-1.1655 * 0.1218 * 0.2) + (-2.1199 * 0.2492 * 0.5) + (-1.6291 * 0.2370 * 0.3) \\ (-1.1655 * 0.1218 * 0.4) + (-2.1199 * 0.2492 * 0.6) + (-1.6291 * 0.2370 * 0.8) \\ (-1.1655 * 0.1218 * 0.1) + (-2.1199 * 0.2492 * 0.9) + (-1.6291 * 0.2370 * 0.7) \end{bmatrix}$$

$$= \begin{bmatrix} (-0.0283) + (-0.2641) + (-0.1158) \\ (-0.0567) + (-0.3169) + (-0.3088) \\ (-0.0141) + (-0.4754) + (-0.2702) \end{bmatrix}$$

$$= \begin{bmatrix} -0.4082 \\ -0.6824 \\ -0.7597 \end{bmatrix}$$

$$\partial W_{jk} = \begin{bmatrix} -0.4082 * 0.0451 * 1.2 & -0.6824 * 0.0451 * 1.2 & -0.7597 * 0.0762 * 1.2 \\ -0.4082 * 0.0451 * 1.6 & -0.6824 * 0.0451 * 1.6 & -0.7597 * 0.0762 * 1.6 \\ -0.4082 * 0.0451 * 1.5 & -0.6824 * 0.0451 * 1.5 & -0.7597 * 0.0762 * 1.5 \end{bmatrix}$$

$$\partial W_{jk} = \begin{bmatrix} -0.0220 & -0.0369 & -0.0694 \\ -0.0294 & -0.0492 & -0.0926 \\ -0.0276 & -0.0461 & -0.0868 \end{bmatrix}$$

$$W'_{jk} = \begin{bmatrix} W_{j1k1} - (lr * \partial W_{j1k1}) & W_{j1k2} - (lr * \partial W_{j1k2}) & W_{j1k3} - (lr * \partial W_{j1k3}) \\ W_{j2k1} - (lr * \partial W_{j2k1}) & W_{j2k2} - (lr * \partial W_{j2k2}) & W_{j2k3} - (lr * \partial W_{j2k3}) \\ W_{j3k1} - (lr * \partial W_{j3k1}) & W_{j3k2} - (lr * \partial W_{j3k2}) & W_{j3k3} - (lr * \partial W_{j3k3} \end{bmatrix}$$

$$W'_{jk} = \begin{bmatrix} W_{j1k1} - (lr * \partial W_{j1k1}) & W_{j1k2} - (lr * \partial W_{j1k2}) & W_{j1k3} - (lr * \partial W_{j1k3}) \\ W_{j2k1} - (lr * \partial W_{j2k1}) & W_{j2k2} - (lr * \partial W_{j2k2}) & W_{j2k3} - (lr * \partial W_{j2k3}) \\ W_{j3k1} - (lr * \partial W_{j3k1}) & W_{j3k2} - (lr * \partial W_{j3k2}) & W_{j3k3} - (lr * \partial W_{j3k3} \end{bmatrix}$$

$$W'_{jk} = \begin{bmatrix} 0.4 - (0.01 * -0.0220) & 0.6 - (0.01 * -0.0369) & 0.1 - (0.01 * -0.0694) \\ 0.2 - (0.01 * -0.0294) & 0.7 - (0.01 * -0.0492) & 0.3 - (0.01 * -0.0926) \\ 0.8 - (0.01 * -0.0276) & 0.1 - (0.01 * -0.0461) & 0.5 - (0.01 * -0.0868 \end{bmatrix}$$

$$W'_{jk} = \begin{bmatrix} -0.4002 & -0.6003 & -0.1006 \\ -0.2002 & -0.7004 & -0.3009 \\ -0.8002 & -0.1004 & -0.5008 \end{bmatrix}$$

Back Propagating the error — (Input Layer — Hidden Layer 1)

Derivation of Relu Layer

$$\frac{\partial h1_{out1}}{\partial h1_{in1}} = \frac{\partial Relu(h1_{in1})}{\partial h1_{in1}}$$

$$\begin{bmatrix} \frac{\partial h1_{out1}}{\partial h1_{in1}} \\ \frac{\partial h1_{out2}}{\partial h1_{in2}} \\ \frac{\partial h1_{out3}}{\partial h1_{in3}} \end{bmatrix} = \begin{bmatrix} 1.00 \\ 1.00 \\ 1.00 \end{bmatrix}$$

$$\frac{\partial h1_{in1}}{\partial W_{i1j1}} = \frac{\partial((I_{out1} * W_{i1j1}) + (I_{out2} * W_{i2j1}) + (I_{out3} * W_{i3j1}) + b_{j1})}{\partial W_{i1j1}}$$

$$\frac{\partial I_{in1}}{\partial W_{i1j1}} = I_{out1}$$

$$\frac{\partial h1_{in1}}{\partial W_{i2j1}} = \frac{\partial((I_{out1} * W_{i1j1}) + (I_{out2} * W_{i2j1}) + (I_{out3} * W_{i3j1}) + b_{j1})}{\partial W_{i2j1}}$$

$$\frac{\partial I_{in1}}{\partial W_{i2j1}} = I_{out2}$$

$$\frac{\partial h1_{in1}}{\partial W_{i3j1}} = \frac{\partial((I_{out1} * W_{i1j1}) + (I_{out2} * W_{i2j1}) + (I_{out3} * W_{i3j1}) + b_{j1})}{\partial W_{i3j1}}$$

$$\frac{\partial I_{in1}}{\partial W_{i3j1}} = I_{out3}$$

$$\frac{\partial h1_{in1}}{\partial W_{i1j2}} = \frac{\partial((I_{out1} * W_{i1j2}) + (I_{out2} * W_{i2j2}) + (I_{out3} * W_{i3j2}) + b_{j2})}{\partial W_{i1j2}}$$

$$\frac{\partial I_{in1}}{\partial W_{i1j2}} = I_{out1}$$

$$\frac{\partial h1_{in1}}{\partial W_{i2j2}} = \frac{\partial((I_{out1} * W_{i1j2}) + (I_{out2} * W_{i2j2}) + (I_{out3} * W_{i3j2}) + b_{j2})}{\partial W_{i2j2}}$$

$$\frac{\partial I_{in1}}{\partial W_{i2j2}} = I_{out2}$$

$$\frac{\partial h1_{in1}}{\partial W_{i3j2}} = \frac{\partial((I_{out1} * W_{i1j2}) + (I_{out2} * W_{i2j2}) + (I_{out3} * W_{i3j2}) + b_{j2})}{\partial W_{i3j2}}$$

$$\frac{\partial I_{in1}}{\partial W_{i3j2}} = I_{out3}$$

$$\frac{\partial h1_{in1}}{\partial W_{i1j3}} = \frac{\partial((I_{out1} * W_{i1j3}) + (I_{out2} * W_{i2j3}) + (I_{out3} * W_{i3j3}) + b_{j3})}{\partial W_{i1j3}}$$

$$\frac{\partial I_{in1}}{\partial W_{i1j3}} = I_{out1}$$

$$\frac{\partial h1_{in1}}{\partial W_{i2j3}} = \frac{\partial((I_{out1} * W_{i1j3}) + (I_{out2} * W_{i2j3}) + (I_{out3} * W_{i3j3}) + b_{j3})}{\partial W_{i2j3}}$$

$$\frac{\partial I_{in1}}{\partial W_{i2j3}} = I_{out2}$$

$$\frac{\partial h1_{in1}}{\partial W_{i3j3}} = \frac{\partial((I_{out1} * W_{i1j3}) + (I_{out2} * W_{i2j3}) + (I_{out3} * W_{i3j3}) + b_{j3})}{\partial W_{i3j3}}$$

$$\frac{\partial I_{in1}}{\partial W_{i3j3}} = I_{out3}$$

$$\partial W_{ij} = \begin{bmatrix} \frac{\partial E_{total}}{\partial W_{i1j1}} & \frac{\partial E_{total}}{\partial W_{i1j2}} & \frac{\partial E_{total}}{\partial W_{i1j3}} \\ \frac{\partial E_{total}}{\partial W_{i2j1}} & \frac{\partial E_{total}}{\partial W_{i2j2}} & \frac{\partial E_{total}}{\partial W_{i2j3}} \\ \frac{\partial E_{total}}{\partial W_{i3j1}} & \frac{\partial E_{total}}{\partial W_{i3j2}} & \frac{\partial E_{total}}{\partial W_{i3j3}} \end{bmatrix}$$

$$\partial W_{ij} = \begin{bmatrix} \frac{\partial E_{total}}{\partial h1_{out1}} * \frac{\partial h1_{out1}}{\partial h1_{in1}} * \frac{\partial h1_{in1}}{\partial W_{i1j1}} & \frac{\partial E_{total}}{\partial h1_{out2}} * \frac{\partial h1_{out2}}{\partial h1_{in2}} * \frac{\partial h1_{in2}}{\partial W_{i1j2}} & \frac{\partial E_{total}}{\partial h1_{out3}} * \frac{\partial h1_{out3}}{\partial h1_{in3}} * \frac{\partial h1_{in3}}{\partial W_{i1j3}} \\ \frac{\partial E_{total}}{\partial h1_{out1}} * \frac{\partial h1_{out1}}{\partial h1_{in1}} * \frac{\partial h1_{in1}}{\partial W_{i2j1}} & \frac{\partial E_{total}}{\partial h1_{out2}} * \frac{\partial h1_{out2}}{\partial h1_{in2}} * \frac{\partial h1_{in2}}{\partial W_{i2j2}} & \frac{\partial E_{total}}{\partial h1_{out3}} * \frac{\partial h1_{out3}}{\partial h1_{in3}} * \frac{\partial h1_{in3}}{\partial W_{i2j3}} \\ \frac{\partial E_{total}}{\partial h1_{out1}} * \frac{\partial h1_{out1}}{\partial h1_{in1}} * \frac{\partial h1_{in1}}{\partial W_{i3j1}} & \frac{\partial E_{total}}{\partial h1_{out2}} * \frac{\partial h1_{out2}}{\partial h1_{in2}} * \frac{\partial h1_{in2}}{\partial W_{i3j2}} & \frac{\partial E_{total}}{\partial h1_{out3}} * \frac{\partial h1_{out3}}{\partial h1_{in3}} * \frac{\partial h1_{in3}}{\partial W_{i3j3}} \end{bmatrix}$$

$$\begin{bmatrix} \frac{\partial E_{total}}{\partial h1_{out1}} \\ \frac{\partial E_{total}}{\partial h1_{out2}} \\ \frac{\partial E_{total}}{\partial h1_{out3}} \end{bmatrix} = \begin{bmatrix} \frac{\partial E_{total}}{\partial h2_{out1}} * \frac{\partial h2_{out1}}{\partial h2_{in1}} * \frac{\partial h2_{in1}}{\partial h1_{out1}} \\ \frac{\partial E_{total}}{\partial h2_{out2}} * \frac{\partial h2_{out2}}{\partial h2_{in2}} * \frac{\partial h2_{in2}}{\partial h1_{out2}} \\ \frac{\partial E_{total}}{\partial h2_{out3}} * \frac{\partial h2_{out3}}{\partial h2_{in3}} * \frac{\partial h2_{in3}}{\partial h1_{out3}} \end{bmatrix}$$

$$\begin{bmatrix} \frac{\partial E_{total}}{\partial h1_{out1}} \\ \frac{\partial E_{total}}{\partial h1_{out2}} \\ \frac{\partial E_{total}}{\partial h1_{out3}} \end{bmatrix} = \begin{bmatrix} \frac{\partial E_{total}}{\partial h2_{out1}} * \frac{\partial h2_{out1}}{\partial h2_{in1}} * W_{j1k1} \\ \frac{\partial E_{total}}{\partial h2_{out2}} * \frac{\partial h2_{out2}}{\partial h2_{in2}} * W_{j2k2} \\ \frac{\partial E_{total}}{\partial h2_{out3}} * \frac{\partial h2_{out3}}{\partial h2_{in3}} * W_{j3k3} \end{bmatrix}$$

$$= \begin{bmatrix} -0.4082 * 0.0451 * 0.4 \\ -0.6824 * 0.0451 * 0.7 \\ -0.7597 * 0.0762 * 0.5 \end{bmatrix}$$

$$= \begin{bmatrix} -0.007363 \\ -0.021543 \\ -0.028944 \end{bmatrix}$$

$$W_{ij} = \begin{bmatrix} \frac{\partial E_{total}}{\partial h1_{out1}} * \frac{\partial h1_{out1}}{\partial h1_{in1}} * \frac{\partial h1_{in1}}{\partial W_{i1j1}} & \frac{\partial E_{total}}{\partial h1_{out2}} * \frac{\partial h1_{out2}}{\partial h1_{in2}} * \frac{\partial h1_{in2}}{\partial W_{i1j2}} & \frac{\partial E_{total}}{\partial h1_{out3}} * \frac{\partial h1_{out3}}{\partial h1_{in3}} * \frac{\partial h1_{in3}}{\partial W_{i1j3}} \\ \frac{\partial E_{total}}{\partial h1_{out1}} * \frac{\partial h1_{out1}}{\partial h1_{in1}} * \frac{\partial h1_{in1}}{\partial W_{i2j1}} & \frac{\partial E_{total}}{\partial h1_{out2}} * \frac{\partial h1_{out2}}{\partial h1_{in2}} * \frac{\partial h1_{in2}}{\partial W_{i2j2}} & \frac{\partial E_{total}}{\partial h1_{out3}} * \frac{\partial h1_{out3}}{\partial h1_{in3}} * \frac{\partial h1_{in3}}{\partial W_{i2j3}} \\ \frac{\partial E_{total}}{\partial h1_{out1}} * \frac{\partial h1_{out1}}{\partial h1_{in1}} * \frac{\partial h1_{in1}}{\partial W_{i3j1}} & \frac{\partial E_{total}}{\partial h1_{out2}} * \frac{\partial h1_{out2}}{\partial h1_{in2}} * \frac{\partial h1_{in2}}{\partial W_{i3j2}} & \frac{\partial E_{total}}{\partial h1_{out3}} * \frac{\partial h1_{out3}}{\partial h1_{in3}} * \frac{\partial h1_{in3}}{\partial W_{i3j3}} \end{bmatrix}$$

$$W_{ij} = \begin{bmatrix} -0.007363 * 1 * 0.1 & -0.021543 * 1 * 0.1 & -0.028944 * 1 * 0.1 \\ -0.007363 * 1 * 0.3 & -0.021543 * 1 * 0.3 & -0.028944 * 1 * 0.3 \\ -0.007363 * 1 * 0.5 & -0.021543 * 1 * 0.5 & -0.028944 * 1 * 0.5 \end{bmatrix}$$

$$W_{ij} = \begin{bmatrix} -0.0007363 & -0.0021543 & -0.0028944 \\ -0.0022089 & -0.0064629 & -0.0086832 \\ -0.0036815 & -0.0107715 & -0.014472 \end{bmatrix}$$

$$W'_{ij} = \begin{bmatrix} W_{i1j1} - (lr * \partial W_{i1j1}) & W_{i1j2} - (lr * \partial W_{i1j2}) & W_{i1j3} - (lr * \partial W_{i1j3}) \\ W_{i2j1} - (lr * \partial W_{i2j1}) & W_{i2j2} - (lr * \partial W_{i2j2}) & W_{i2j3} - (lr * \partial W_{i2j3}) \\ W_{i3j1} - (lr * \partial W_{i3j1}) & W_{i3j2} - (lr * \partial W_{i3j2}) & W_{i3j3} - (lr * \partial W_{i3j3}) \end{bmatrix}$$

$$W'_{ij} =$$
$$\begin{bmatrix} 0.2 - (0.01 * -0.0007363) & 0.4 - (0.01 * -0.0021543) & 0.5 - (0.01 * -0.0028944) \\ 0.3 - (0.01 * -0.0022089) & 0.7 - (0.01 * -0.0064629) & 0.1 - (0.01 * -0.0086832) \\ 0.2 - (0.01 * -0.0036815) & 0.6 - (0.01 * -0.0107715) & 0.9 - (0.01 * -0.014472) \end{bmatrix}$$

$$W'_{ij} = \begin{bmatrix} 0.20007 & 0.4003 & 0.5007 \\ 0.30016 & 0.7001 & 0.1007 \\ 0.20078 & 0.6008 & 0.9009 \end{bmatrix}$$

So Weights are updated in each layer. Again, forward pass then recalculates the errors with actual output and then backward pass repeat the step until error is reduced.

So in these ways backpropagation algorithm work.

Practical Implementation of Artificial Neural Network?

Please refer my GitHub id for Source code and dataset: https://github.com/SatwikPM

Churn Modelling Problem

you will be solving a data analytics challenge for a bank. You will be given a dataset with a large sample of the bank's customers.

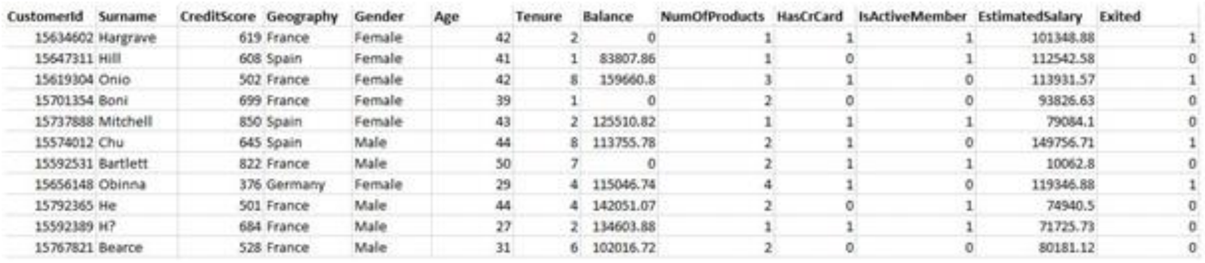
- To make this dataset, the bank gathered information such as customer id, credit score, gender, age, tenure, balance, if the customer is active, has a credit card, etc.
- During 6 months, the bank observed if these customers left or stayed in the bank.

Your goal is to make an Artificial Neural Network that can predict, based on geo-demographical and transactional information given above, if any individual customer will leave the bank or stay (customer churn).

Besides, you are asked to rank all the customers of the bank, based on their probability of leaving. To do that, you will need to use the right Deep Learning model, one that is based on a probabilistic approach.

If you succeed in this project, you will create significant added value to the bank. By applying your Deep Learning model, the bank may significantly reduce customer churn.

Dataset sample:

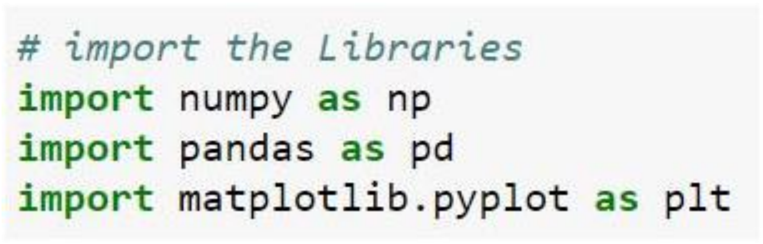


CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	EstimatedSalary	Exited
15634602	Hargrave	619	France	Female	42	2	0	1	1	1	101348.88	1
15647311	Hill	608	Spain	Female	41	1	83807.86	1	0	1	112542.58	0
15619304	Onio	502	France	Female	42	8	159660.8	3	1	0	113931.57	1
15701354	Boni	699	France	Female	39	1	0	2	0	0	93826.63	0
15737888	Mitchell	850	Spain	Female	43	2	125510.82	1	1	1	79084.1	0
15574012	Chu	645	Spain	Male	44	8	113755.78	2	1	0	149756.71	1
15592531	Bartlett	822	France	Male	50	7	0	2	1	1	10062.8	0
15656148	Obinna	376	Germany	Female	29	4	115046.74	4	1	0	119346.88	1
15792365	He	501	France	Male	44	4	142051.07	2	0	1	74940.5	0
15592389	H?	684	France	Male	27	2	134603.88	1	1	1	71725.73	0
15767821	Bearce	528	France	Male	31	6	102016.72	2	0	0	80181.12	0

Part 1: Data Pre-processing

1.1 Import the Libraries

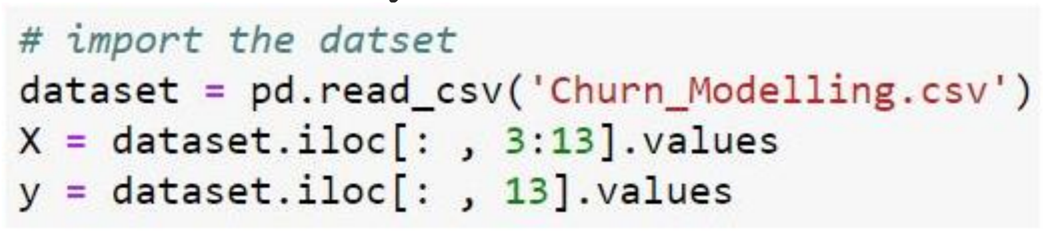
In this step, we import three Libraries in Data Pre-processing part. A library is a tool that you can use to make a specific job. First of all, we import the **numpy** library used for multidimensional array then import the **pandas** library used to import the dataset and in last we import **matplotlib** library used for plotting the graph.



```
# import the Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

1.2 Import the dataset

In this step, we import the dataset to do that we use the **pandas** library. After import our dataset we define our dependent and independent variable. Our independent variables are 1 to 12 attributes as you can see in the sample dataset which we call '**X**' and dependent is our last attribute which we call '**y**' here.



```
# import the datset
dataset = pd.read_csv('Churn_Modelling.csv')
X = dataset.iloc[: , 3:13].values
y = dataset.iloc[: , 13].values
```

1.3 Encoding the Categorical data

In this step, we Encode our categorical data. If we see our dataset then Geography & Gender attribute is in Text and we Encode these two attributes in this part use the LabelEncoder and OneHOTEncoder from the Sklearn.processing library.

```python
# Encoding the categorical data
from sklearn.preprocessing import LabelEncoder , OneHotEncoder
labelencoder_X1 = LabelEncoder()
X[:, 1] = labelencoder_X1.fit_transform(X[ : , 1])
labelencoder_X2 = LabelEncoder()
X[:, 2] = labelencoder_X2.fit_transform(X[: , 2])
onehotencoder = OneHotEncoder(categorical_features = [1])
X = onehotencoder.fit_transform(X).toarray()
X = X[:, 1:]
```

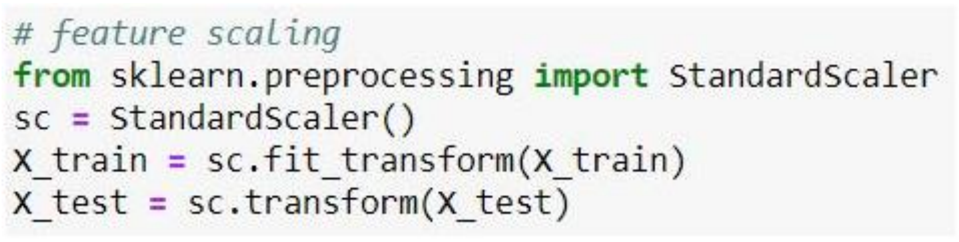
1.4 Split the dataset for test and train

In this step, we split our dataset into a test set and train set and an 80% dataset split for training and the remaining 20% for tests. Our dataset contains 10000 instances so 8000 data we train and 2000 for the test.

```python
# split the dataset into test set and train set
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =train_test_split(X,y, test_size= 0.2, random_state=0)
```

Standardisation	Normalisation
$x_{\text{stand}} = \dfrac{x - \text{mean}(x)}{\text{standard deviation }(x)}$	$x_{\text{norm}} = \dfrac{x - \min(x)}{\max(x) - \min(x)}$

Here we use standard Scaler import from Sklearn Library.



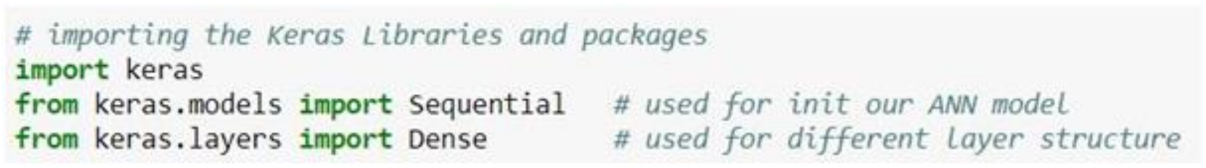
```
# feature scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Part 2: Building our Model

In this part, we model our Artificial Neural Network model.

2.1 Import the Libraries

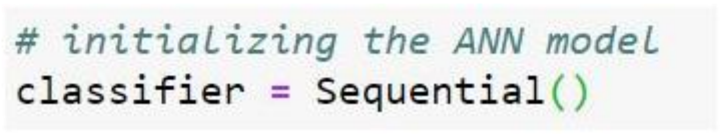
In this step, we import the Library which will build our ANN model. We import **Keras** Library which will build a deep neural network based on TensorFlow because we use TensorFlow backhand. Here we import the two modules from Keras. The first one is **Sequential** used for initializing our ANN model and the second is **Dense** used for adding different layers of ANN.



```
# importing the Keras Libraries and packages
import keras
from keras.models import Sequential    # used for init our ANN model
from keras.layers import Dense         # used for different layer structure
```

2.2 Initialize our ANN model

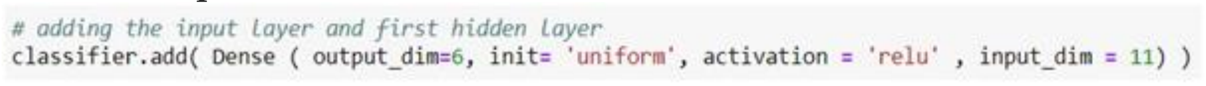
In this step, we initialize our Artificial Neural Network model to do that we use sequential modules.



```
# initializing the ANN model
classifier = Sequential()
```

2.3 Adding the input layer and first hidden layer

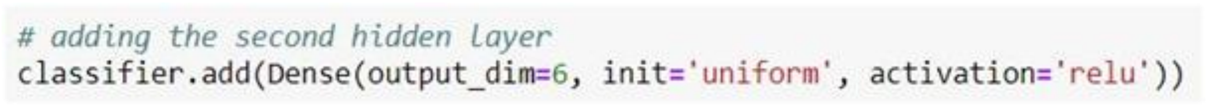
In this step, we use the Dense model to add a different layer. The parameter which we pass here first is output_dim=6 which defines hidden layer=6, the second parameter is init= uniform basically this is a uniform function that randomly initializes the weights which are close to 0 but not 0. The third parameter is activation= relu here in the first hidden layer we use relu activation. And the last parameter which we pass in dense function is input_dim= 11 which means the input node of our Neural Network is 11 because our dataset has 11 attributes that's why we choose 11 input nodes.



```
# adding the input layer and first hidden layer
classifier.add( Dense ( output_dim=6, init= 'uniform', activation = 'relu' , input_dim = 11) )
```

2.4 Adding the Second Hidden layer

In this step, we add another hidden layer



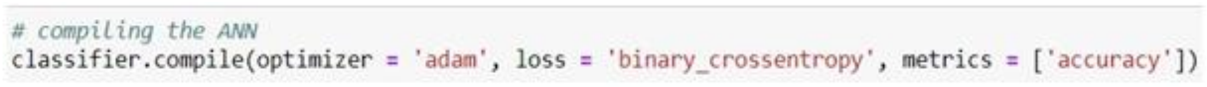
```
# adding the second hidden layer
classifier.add(Dense(output_dim=6, init='uniform', activation='relu'))
```

2.5 Adding the output layer

In this step, we add an output layer in our ANN structure output_dim= 1 which means one output node here we use the sigmoid function because our target attribute has a binary class which is one or zero that's why we use sigmoid activation function.

2.6 Compiling the ANN

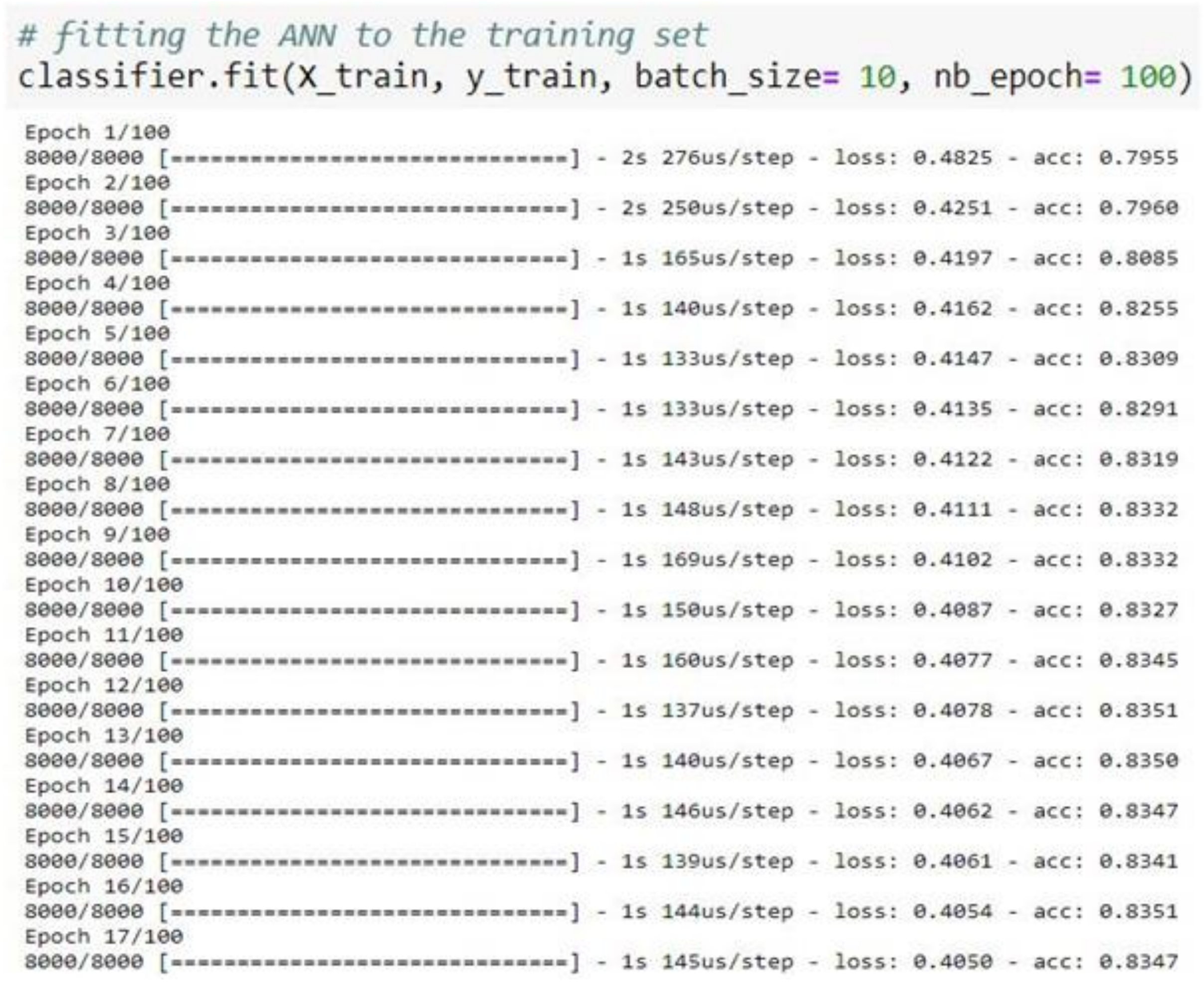
In this step, we compile the ANN to do that we use the compile method and add several parameters the first parameter is **optimizer = Adam** here use the optimal number of weights. So for choosing the optimal number of weights, there are various algorithms of Stochastic Gradient Descent but very efficient one which is Adam so that's why we use Adam optimizer here. The second parameter is loss this corresponds to loss function here we use **binary_crossentropy** because if we see target attribute our dataset which contains the binary value that's why we choose the binary cross-entropy. The final parameter is **metrics** basically It's a list of metrics to be evaluated by the model and here we choose the accuracy metrics.



```
# compiling the ANN
classifier.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

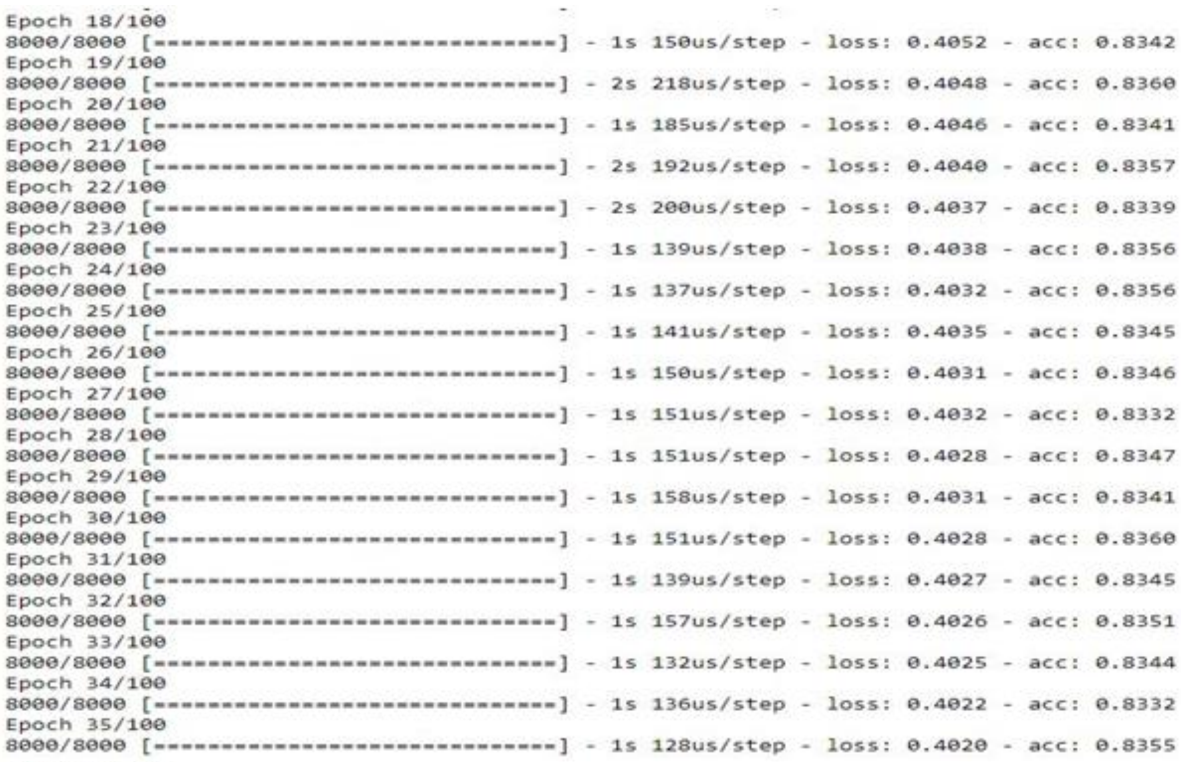
2.7 Fitting the ANN

In this step we fit the training data our model X_train, y_train is our training data. Here a **batch size** is basically a number of observations after which you want to update the weights here we take batch size 10. And the final parameter is **epoch** is basically when whole the training set passed through the ANN here we choose the 100 number of the epoch.

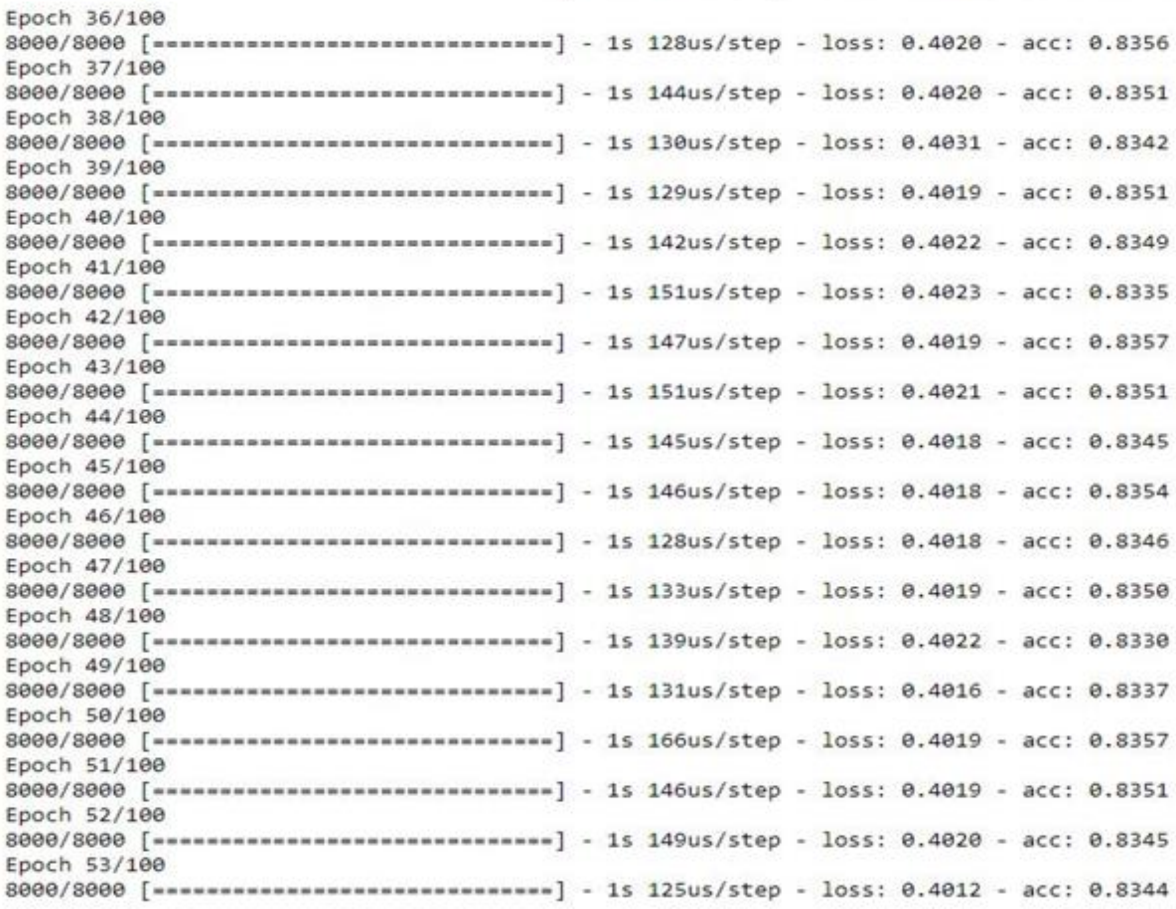


```
# fitting the ANN to the training set
classifier.fit(X_train, y_train, batch_size= 10, nb_epoch= 100)
```

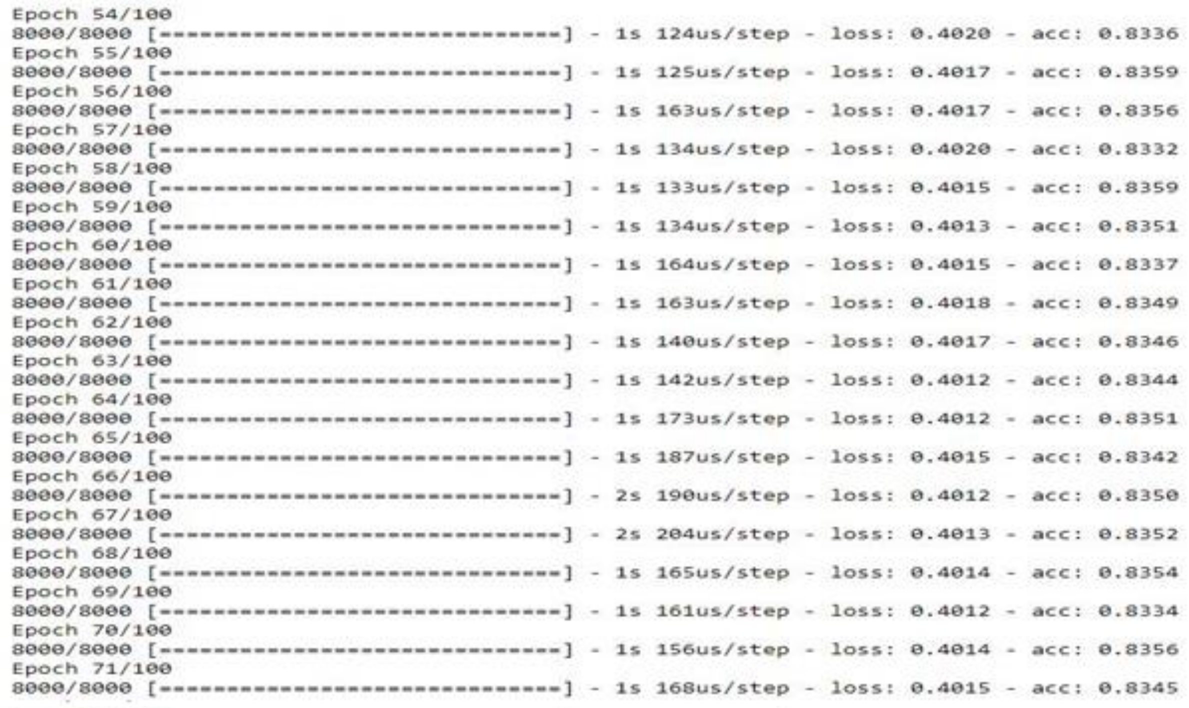
```
Epoch 1/100
8000/8000 [==============================] - 2s 276us/step - loss: 0.4825 - acc: 0.7955
Epoch 2/100
8000/8000 [==============================] - 2s 250us/step - loss: 0.4251 - acc: 0.7960
Epoch 3/100
8000/8000 [==============================] - 1s 165us/step - loss: 0.4197 - acc: 0.8085
Epoch 4/100
8000/8000 [==============================] - 1s 140us/step - loss: 0.4162 - acc: 0.8255
Epoch 5/100
8000/8000 [==============================] - 1s 133us/step - loss: 0.4147 - acc: 0.8309
Epoch 6/100
8000/8000 [==============================] - 1s 133us/step - loss: 0.4135 - acc: 0.8291
Epoch 7/100
8000/8000 [==============================] - 1s 143us/step - loss: 0.4122 - acc: 0.8319
Epoch 8/100
8000/8000 [==============================] - 1s 148us/step - loss: 0.4111 - acc: 0.8332
Epoch 9/100
8000/8000 [==============================] - 1s 169us/step - loss: 0.4102 - acc: 0.8332
Epoch 10/100
8000/8000 [==============================] - 1s 150us/step - loss: 0.4087 - acc: 0.8327
Epoch 11/100
8000/8000 [==============================] - 1s 160us/step - loss: 0.4077 - acc: 0.8345
Epoch 12/100
8000/8000 [==============================] - 1s 137us/step - loss: 0.4078 - acc: 0.8351
Epoch 13/100
8000/8000 [==============================] - 1s 140us/step - loss: 0.4067 - acc: 0.8350
Epoch 14/100
8000/8000 [==============================] - 1s 146us/step - loss: 0.4062 - acc: 0.8347
Epoch 15/100
8000/8000 [==============================] - 1s 139us/step - loss: 0.4061 - acc: 0.8341
Epoch 16/100
8000/8000 [==============================] - 1s 144us/step - loss: 0.4054 - acc: 0.8351
Epoch 17/100
8000/8000 [==============================] - 1s 145us/step - loss: 0.4050 - acc: 0.8347
```



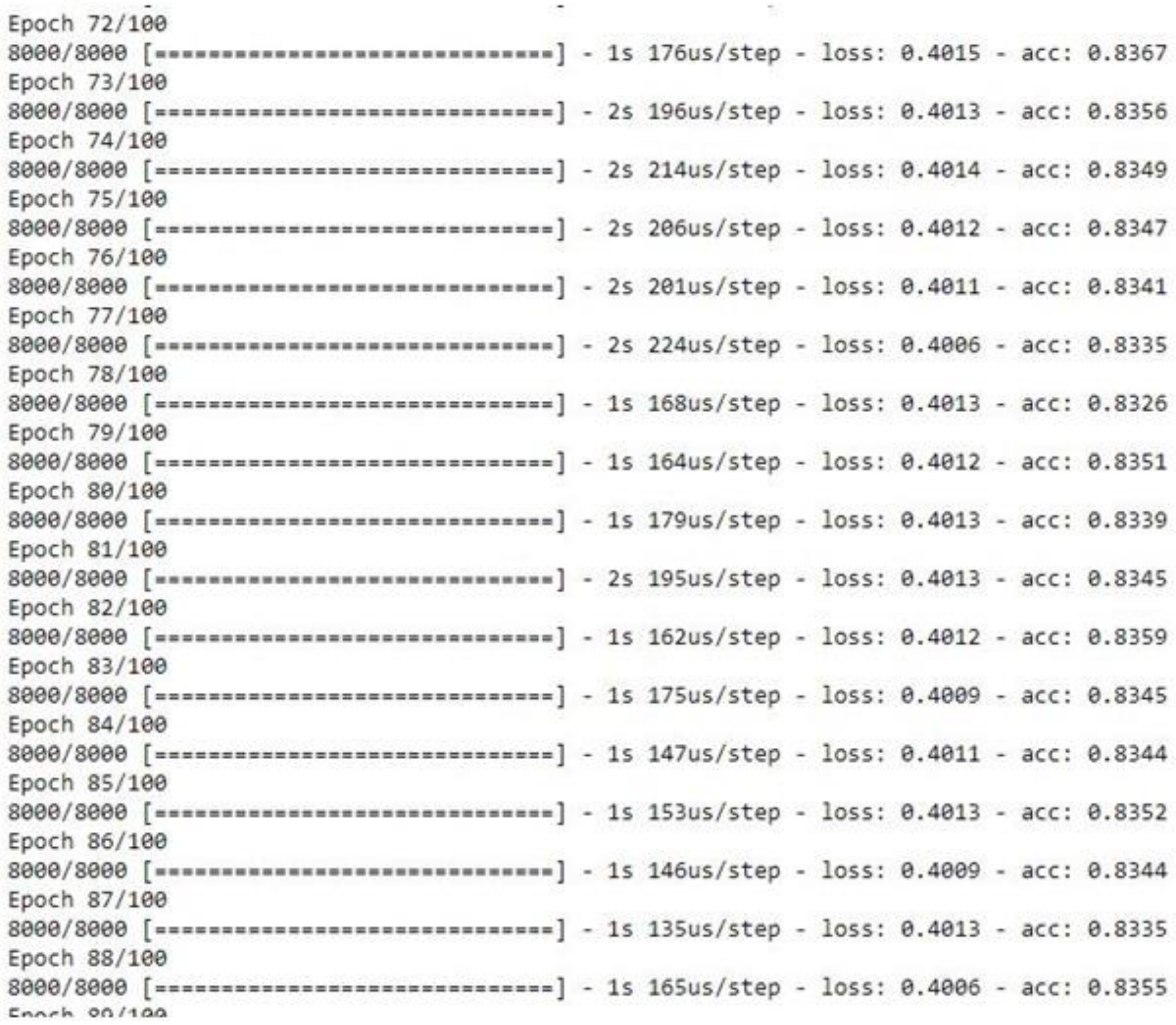
```
Epoch 18/100
8000/8000 [==============================] - 1s 150us/step - loss: 0.4052 - acc: 0.8342
Epoch 19/100
8000/8000 [==============================] - 2s 218us/step - loss: 0.4048 - acc: 0.8360
Epoch 20/100
8000/8000 [==============================] - 1s 185us/step - loss: 0.4046 - acc: 0.8341
Epoch 21/100
8000/8000 [==============================] - 2s 192us/step - loss: 0.4040 - acc: 0.8357
Epoch 22/100
8000/8000 [==============================] - 2s 200us/step - loss: 0.4037 - acc: 0.8339
Epoch 23/100
8000/8000 [==============================] - 1s 139us/step - loss: 0.4038 - acc: 0.8356
Epoch 24/100
8000/8000 [==============================] - 1s 137us/step - loss: 0.4032 - acc: 0.8356
Epoch 25/100
8000/8000 [==============================] - 1s 141us/step - loss: 0.4035 - acc: 0.8345
Epoch 26/100
8000/8000 [==============================] - 1s 150us/step - loss: 0.4031 - acc: 0.8346
Epoch 27/100
8000/8000 [==============================] - 1s 151us/step - loss: 0.4032 - acc: 0.8332
Epoch 28/100
8000/8000 [==============================] - 1s 151us/step - loss: 0.4028 - acc: 0.8347
Epoch 29/100
8000/8000 [==============================] - 1s 158us/step - loss: 0.4031 - acc: 0.8341
Epoch 30/100
8000/8000 [==============================] - 1s 151us/step - loss: 0.4028 - acc: 0.8360
Epoch 31/100
8000/8000 [==============================] - 1s 139us/step - loss: 0.4027 - acc: 0.8345
Epoch 32/100
8000/8000 [==============================] - 1s 157us/step - loss: 0.4026 - acc: 0.8351
Epoch 33/100
8000/8000 [==============================] - 1s 132us/step - loss: 0.4025 - acc: 0.8344
Epoch 34/100
8000/8000 [==============================] - 1s 136us/step - loss: 0.4022 - acc: 0.8332
Epoch 35/100
8000/8000 [==============================] - 1s 128us/step - loss: 0.4020 - acc: 0.8355
```



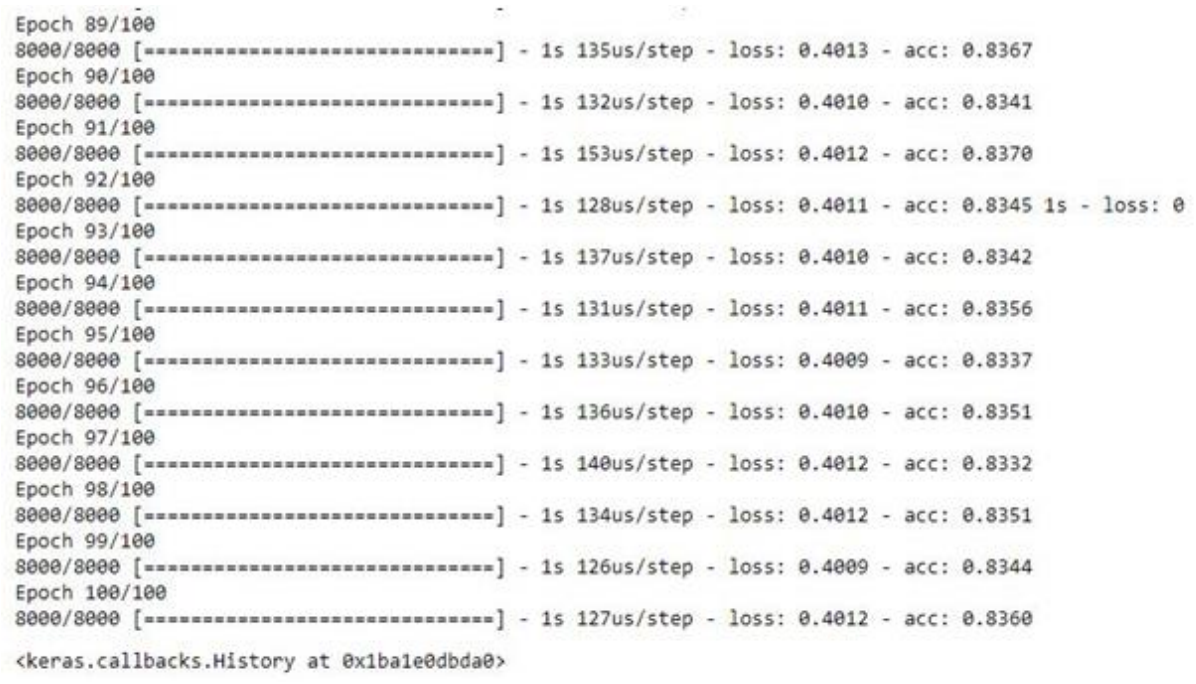
```
Epoch 36/100
8000/8000 [==============================] - 1s 128us/step - loss: 0.4020 - acc: 0.8356
Epoch 37/100
8000/8000 [==============================] - 1s 144us/step - loss: 0.4020 - acc: 0.8351
Epoch 38/100
8000/8000 [==============================] - 1s 130us/step - loss: 0.4031 - acc: 0.8342
Epoch 39/100
8000/8000 [==============================] - 1s 129us/step - loss: 0.4019 - acc: 0.8351
Epoch 40/100
8000/8000 [==============================] - 1s 142us/step - loss: 0.4022 - acc: 0.8349
Epoch 41/100
8000/8000 [==============================] - 1s 151us/step - loss: 0.4023 - acc: 0.8335
Epoch 42/100
8000/8000 [==============================] - 1s 147us/step - loss: 0.4019 - acc: 0.8357
Epoch 43/100
8000/8000 [==============================] - 1s 151us/step - loss: 0.4021 - acc: 0.8351
Epoch 44/100
8000/8000 [==============================] - 1s 145us/step - loss: 0.4018 - acc: 0.8345
Epoch 45/100
8000/8000 [==============================] - 1s 146us/step - loss: 0.4018 - acc: 0.8354
Epoch 46/100
8000/8000 [==============================] - 1s 128us/step - loss: 0.4018 - acc: 0.8346
Epoch 47/100
8000/8000 [==============================] - 1s 133us/step - loss: 0.4019 - acc: 0.8350
Epoch 48/100
8000/8000 [==============================] - 1s 139us/step - loss: 0.4022 - acc: 0.8330
Epoch 49/100
8000/8000 [==============================] - 1s 131us/step - loss: 0.4016 - acc: 0.8337
Epoch 50/100
8000/8000 [==============================] - 1s 166us/step - loss: 0.4019 - acc: 0.8357
Epoch 51/100
8000/8000 [==============================] - 1s 146us/step - loss: 0.4019 - acc: 0.8351
Epoch 52/100
8000/8000 [==============================] - 1s 149us/step - loss: 0.4020 - acc: 0.8345
Epoch 53/100
8000/8000 [==============================] - 1s 125us/step - loss: 0.4012 - acc: 0.8344
```



```
Epoch 54/100
8000/8000 [==============================] - 1s 124us/step - loss: 0.4020 - acc: 0.8336
Epoch 55/100
8000/8000 [==============================] - 1s 125us/step - loss: 0.4017 - acc: 0.8359
Epoch 56/100
8000/8000 [==============================] - 1s 163us/step - loss: 0.4017 - acc: 0.8356
Epoch 57/100
8000/8000 [==============================] - 1s 134us/step - loss: 0.4020 - acc: 0.8332
Epoch 58/100
8000/8000 [==============================] - 1s 133us/step - loss: 0.4015 - acc: 0.8359
Epoch 59/100
8000/8000 [==============================] - 1s 134us/step - loss: 0.4013 - acc: 0.8351
Epoch 60/100
8000/8000 [==============================] - 1s 164us/step - loss: 0.4015 - acc: 0.8337
Epoch 61/100
8000/8000 [==============================] - 1s 163us/step - loss: 0.4018 - acc: 0.8349
Epoch 62/100
8000/8000 [==============================] - 1s 140us/step - loss: 0.4017 - acc: 0.8346
Epoch 63/100
8000/8000 [==============================] - 1s 142us/step - loss: 0.4012 - acc: 0.8344
Epoch 64/100
8000/8000 [==============================] - 1s 173us/step - loss: 0.4012 - acc: 0.8351
Epoch 65/100
8000/8000 [==============================] - 1s 187us/step - loss: 0.4015 - acc: 0.8342
Epoch 66/100
8000/8000 [==============================] - 2s 190us/step - loss: 0.4012 - acc: 0.8350
Epoch 67/100
8000/8000 [==============================] - 2s 204us/step - loss: 0.4013 - acc: 0.8352
Epoch 68/100
8000/8000 [==============================] - 1s 165us/step - loss: 0.4014 - acc: 0.8354
Epoch 69/100
8000/8000 [==============================] - 1s 161us/step - loss: 0.4012 - acc: 0.8334
Epoch 70/100
8000/8000 [==============================] - 1s 156us/step - loss: 0.4014 - acc: 0.8356
Epoch 71/100
8000/8000 [==============================] - 1s 168us/step - loss: 0.4015 - acc: 0.8345
```



```
Epoch 72/100
8000/8000 [==============================] - 1s 176us/step - loss: 0.4015 - acc: 0.8367
Epoch 73/100
8000/8000 [==============================] - 2s 196us/step - loss: 0.4013 - acc: 0.8356
Epoch 74/100
8000/8000 [==============================] - 2s 214us/step - loss: 0.4014 - acc: 0.8349
Epoch 75/100
8000/8000 [==============================] - 2s 206us/step - loss: 0.4012 - acc: 0.8347
Epoch 76/100
8000/8000 [==============================] - 2s 201us/step - loss: 0.4011 - acc: 0.8341
Epoch 77/100
8000/8000 [==============================] - 2s 224us/step - loss: 0.4006 - acc: 0.8335
Epoch 78/100
8000/8000 [==============================] - 1s 168us/step - loss: 0.4013 - acc: 0.8326
Epoch 79/100
8000/8000 [==============================] - 1s 164us/step - loss: 0.4012 - acc: 0.8351
Epoch 80/100
8000/8000 [==============================] - 1s 179us/step - loss: 0.4013 - acc: 0.8339
Epoch 81/100
8000/8000 [==============================] - 2s 195us/step - loss: 0.4013 - acc: 0.8345
Epoch 82/100
8000/8000 [==============================] - 1s 162us/step - loss: 0.4012 - acc: 0.8359
Epoch 83/100
8000/8000 [==============================] - 1s 175us/step - loss: 0.4009 - acc: 0.8345
Epoch 84/100
8000/8000 [==============================] - 1s 147us/step - loss: 0.4011 - acc: 0.8344
Epoch 85/100
8000/8000 [==============================] - 1s 153us/step - loss: 0.4013 - acc: 0.8352
Epoch 86/100
8000/8000 [==============================] - 1s 146us/step - loss: 0.4009 - acc: 0.8344
Epoch 87/100
8000/8000 [==============================] - 1s 135us/step - loss: 0.4013 - acc: 0.8335
Epoch 88/100
8000/8000 [==============================] - 1s 165us/step - loss: 0.4006 - acc: 0.8355
Epoch 89/100
```

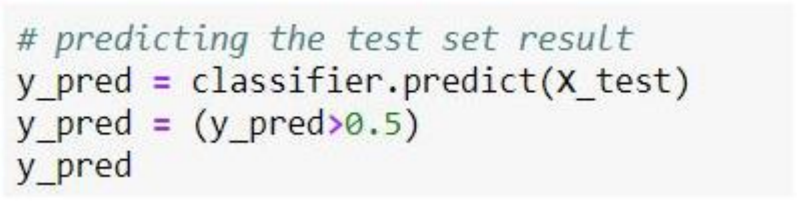


```
Epoch 89/100
8000/8000 [==============================] - 1s 135us/step - loss: 0.4013 - acc: 0.8367
Epoch 90/100
8000/8000 [==============================] - 1s 132us/step - loss: 0.4010 - acc: 0.8341
Epoch 91/100
8000/8000 [==============================] - 1s 153us/step - loss: 0.4012 - acc: 0.8370
Epoch 92/100
8000/8000 [==============================] - 1s 128us/step - loss: 0.4011 - acc: 0.8345 1s - loss: 0
Epoch 93/100
8000/8000 [==============================] - 1s 137us/step - loss: 0.4010 - acc: 0.8342
Epoch 94/100
8000/8000 [==============================] - 1s 131us/step - loss: 0.4011 - acc: 0.8356
Epoch 95/100
8000/8000 [==============================] - 1s 133us/step - loss: 0.4009 - acc: 0.8337
Epoch 96/100
8000/8000 [==============================] - 1s 136us/step - loss: 0.4010 - acc: 0.8351
Epoch 97/100
8000/8000 [==============================] - 1s 140us/step - loss: 0.4012 - acc: 0.8332
Epoch 98/100
8000/8000 [==============================] - 1s 134us/step - loss: 0.4012 - acc: 0.8351
Epoch 99/100
8000/8000 [==============================] - 1s 126us/step - loss: 0.4009 - acc: 0.8344
Epoch 100/100
8000/8000 [==============================] - 1s 127us/step - loss: 0.4012 - acc: 0.8360
<keras.callbacks.History at 0x1ba1e0dbda0>
```

Part 3: Making the Prediction and Accuracy Result.

3.1 Predict the test set Result

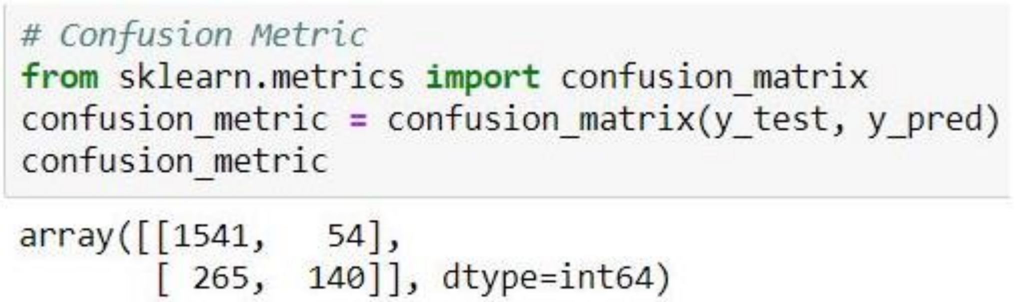
In this step, we predict our test set result here our prediction results in probability so we choose 1(customer leave the bank) if the probability is greater than one 0.5 otherwise 0(customer don't leave the bank).



```python
# predicting the test set result
y_pred = classifier.predict(X_test)
y_pred = (y_pred>0.5)
y_pred
```

3.2 Confusion metrics

In this step we make a confusion metric of our test set result to do that we import confusion matrix from sklearn.metrics then in confusion matrix, we pass two parameters first is y_test which is the actual test set result and second is y_pred which predicted the result.

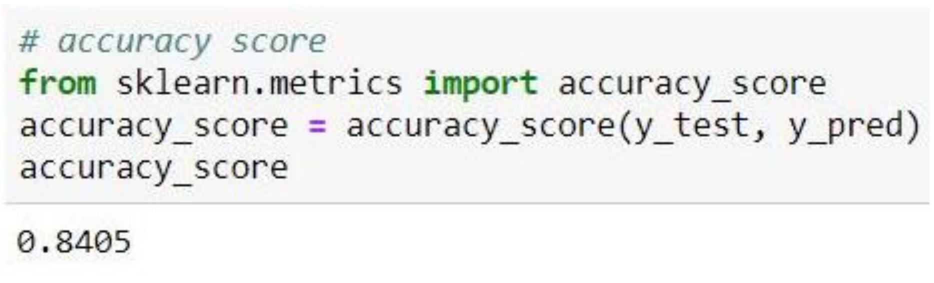


```python
# Confusion Metric
from sklearn.metrics import confusion_matrix
confusion_metric = confusion_matrix(y_test, y_pred)
confusion_metric
```

```
array([[1541,   54],
       [ 265,  140]], dtype=int64)
```

3.3 Accuracy Score

In this step, we calculate the accuracy score based on the actual test result and predict test results.



```
# accuracy score
from sklearn.metrics import accuracy_score
accuracy_score = accuracy_score(y_test, y_pred)
accuracy_score
```

```
0.8405
```

So here we go we get 84.05% of our ANN model.