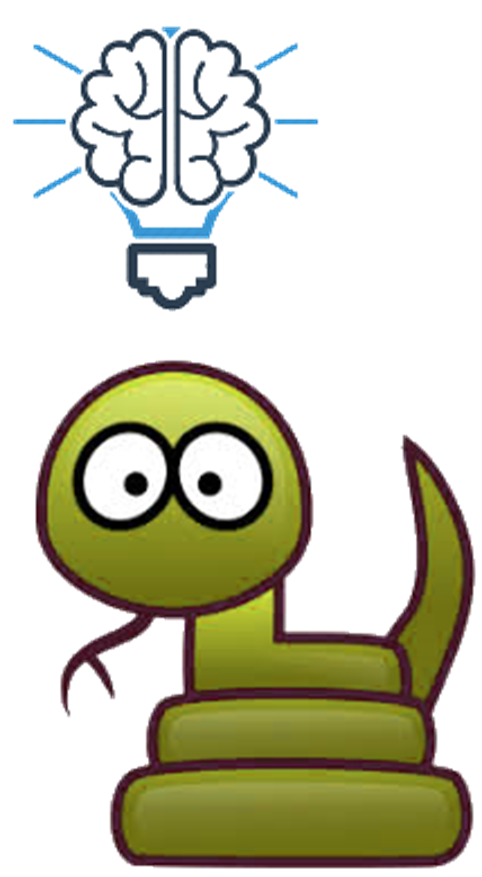
Snake [AI]



Ankit Suman,

Vivek Raj,

(Software Trainee, NathCorp)

**Contents**

|  |  |  |
| --- | --- | --- |
| S.No. | Topics | Page No. |
| 1 | Introduction | 2 |
| 2 | Abstract | 3 |
| 3 | Requirements | 4 |
| 4 | Game Rules | 5 |
| 5 | Game Logic | 6-9 |
| 6 | Tracking Data for Model | 10-11 |
| 7 | Dataset | 12 |
| 8 | Artificial Neural Network | 13 |
| 9 | Artificial Neuron and Biological neuron. | 14 |
| 10 | Building an ANN Model | 18 |

# **Introduction**

Machine learning is revolutionizing our modern world in ways human would never have thought off. There are so many applications of machine learning around us with which we interact on a daily basic. For example, Google Assistant, Facebook’s auto face recognition to tag photos automatically, Uber Share and a lot more are out there. The machines are capable of learning things very fast. And they are learning even more each day. So in order to need to cope up with this changing technologies, we need to learn things like Machine Learning, AI, Deep Learning as well.

The moto of this project was to learn the concepts of Machine Learning (Deep Learning to be more specific), how it works, how the computer system (that knows only 0’s and 1’s) learns to do something that it not hard coded onto them, to understand the complications which arises during implementing the Machine Learning models and what is the scope of Machine Learning.

# **Abstract**

In this project, our goal was to learn python programming, some common packages like NumPy, Pandas, PyGames, Keras, TensorFlow etc. as well as to get a good concept of Machine Learning. Understanding a problem statement is very important and then thinking of a solution according to the tools that you have in your arsenal is equally important as well. Finding a pattern in the data that we have or tracking the right set of data for future processing plays a very vital role in the success of a system. Choosing the correct features and a correct Machine Learning model is the key to make a good Machine Learning Model.

In this project we had to make a Snake game which we used to play back in 1990’s in our Nokia mobile phones. And then we had to play the game and generate data from it while playing and record the directions in which we are directing the snake to. Now the third step was to use this generated dataset to train a model so that it can play the game on its own.

**Requirements**

**Software Requirements:**

* Python 3.5.2
* PyCharm

**Packages Required:**

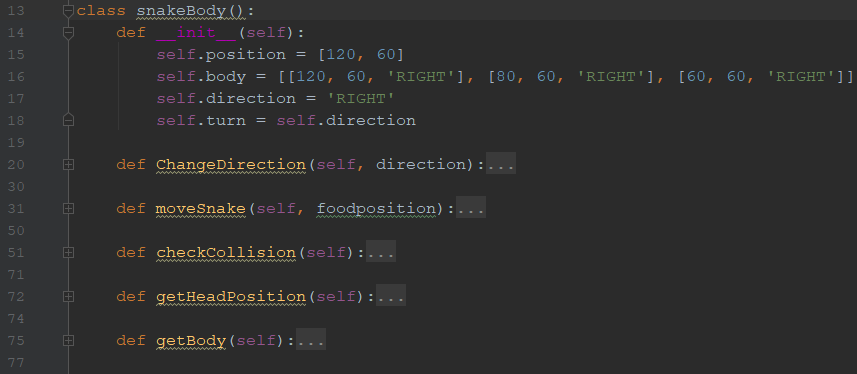
* NumPy
* Pandas
* Random
* Time
* Sys
* Keras
* TensorFlow
* PyGames

**Game Rules**

* Can move forward only.
* Can turn left or right.
* If the head touches the boundaries or its body itself the snake dies.
* Grows after eating food.
* Gets points for every food it eats.
* Eat as much food as it needs.

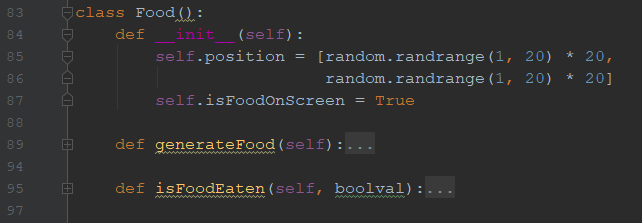
**Game Logic**

* **Snake Class:**

****

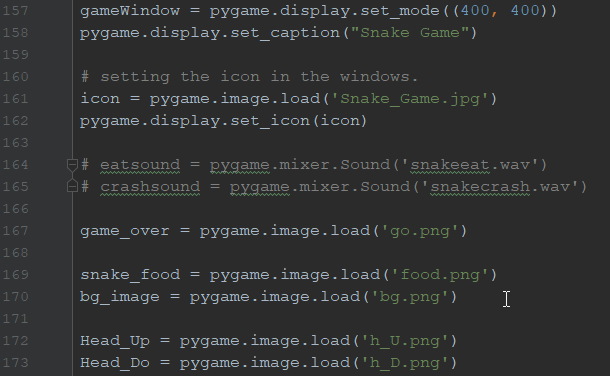
In this class we are using the variables in the **\_\_init\_\_()** function to initialize the starting position of the snake, its body and the direction it is moving to. The **ChangeDirection()** method is used to change the direction of the snake as required by the user. **moveSnake()** method is used to move the body of snake ahead in the direction the user wants to. **checkCollision()** method is called in every frame to check whether the snake has collided with the wall or its body. If it does the game ends and if not, it continues. **getHeadPosition()** and **getBody()** is just a function to get the position of the snake’s head or body when required.

* **Food Class:**

****

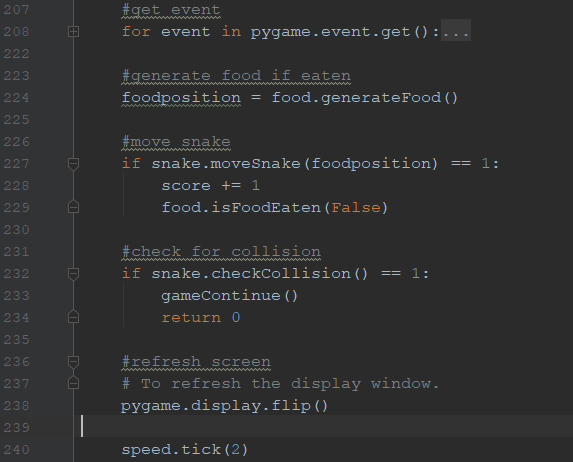
**\_\_init\_\_()** method is used to initialize the initial position of the food and. **generateFood()** is used to generate a new food for the snake after food is eaten. **isFoodEaten()** method is used to check whether the food has been eaten by snake or not.

* **Initializing the PyGames window:**

****

Now we need to initialize the PyGame window which we have taken to be 400, 400 pixels. And load all the images that are required for the game.

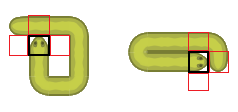
* **Working in every frame:**

****

For every frame we will check if the food is on the screen or not. If not, then we need to generate a new food using the random function. Then we will check the event and move the snake accordingly, and we will also check if the food is eaten by the snake or not. If yes, then we pass False boolean value to the **isFoodEaten()** method. Then we check if the snake has collided or not. If yes, then we end the game. If No, then we refresh the screen and continue.

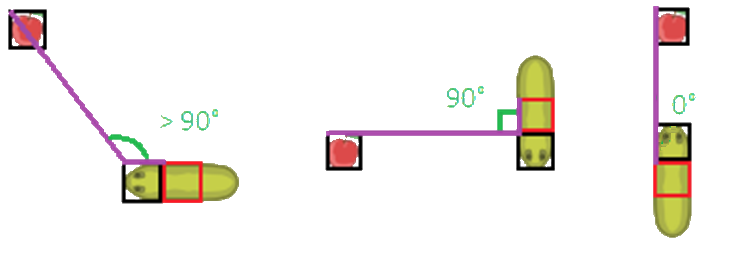
**Tracking Data for Model**

* **Check Obstacles:**

****

We need to keep track of the directions the snake cannot move to. That is, whether the left, right or the front of the snake’s head is blocked or not. If the direction is blocked, then we set the variable to 1 if not then we set the variable to 0.

* **Check Angle:**

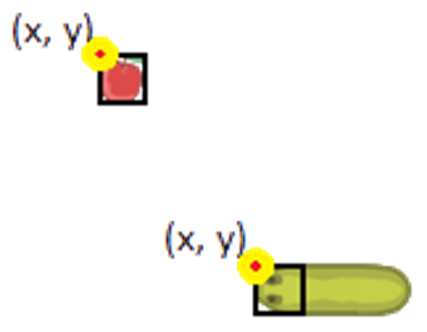


In this we need to keep track of the angle between the food and the snake. We are tracking this because there is a pattern which follows. If we take a look at the data set we find that our goal is to move in a direction in which the angle will decrease or increase up to 90° and then the snake turns towards the food and the angle becomes 0° and the snake moves forward until it reaches the food.

* **Status:**

Status is just another variable which keeps track of whether the snake is moving towards the food, away from the food or has collided. It tracks this using the same angle concept.

* **Co-ordinates:**



Here we are storing the x and y co-ordinates of the food and snake’s head.

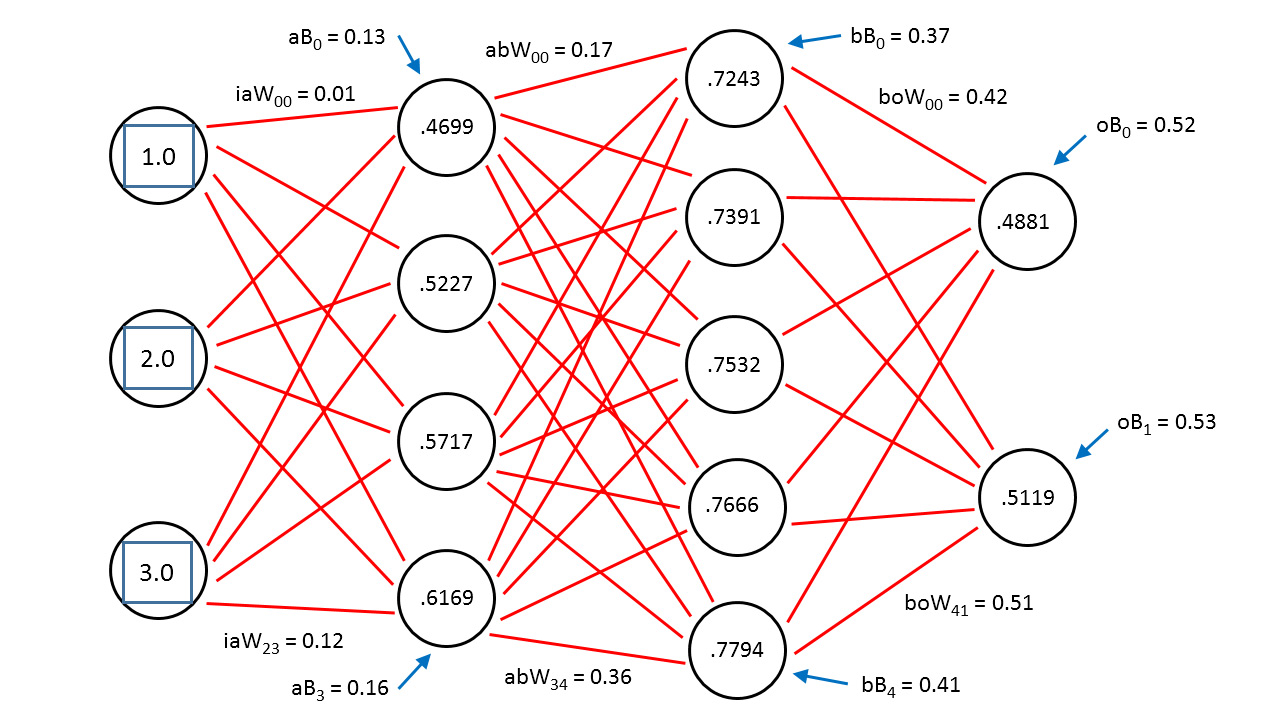
**Dataset**

So, the dataset that we have, consists of the following attributes.

* Left\_B
* Right\_B
* Front\_B
* Cosine\_Angle
* Status
* Food\_x
* Food\_y
* Snake\_x
* Snake\_y
* Suggested\_Direction

# **ARTIFICIAL NEURAL NETWORK**

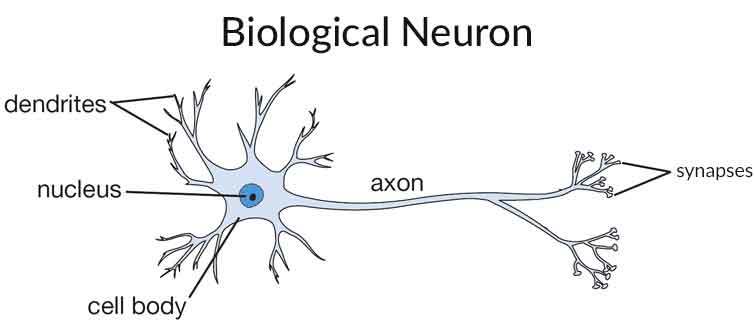
* **Neural network** is a series of algorithms that seek to identify relationships in a data set via a process that mimics how the human brain works.
* An artificial neural network is an interconnected group of nodes, akin to the vast network of neurons in a brain. Here, each circular node represents an artificial neuron and an arrow represents a connection from the output of one artificial neuron to the input of another.

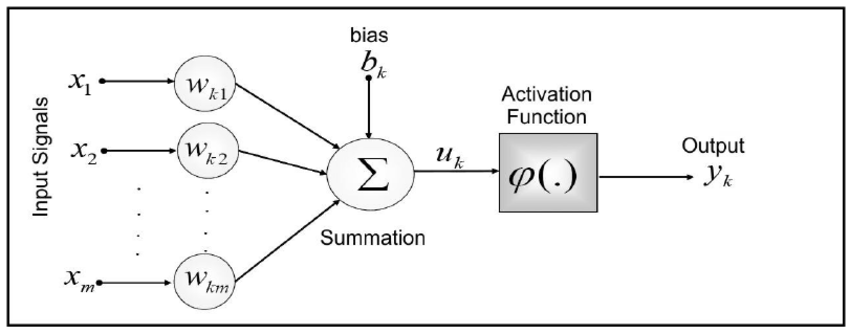


* An Artificial **Neural Network** (ANN) is an information processing paradigm that is inspired by the way biological nervous systems, such as the brain, process information.
* The key element of this paradigm is the novel structure of the information processing system.

**ARTIFICIAL NEURON & BIOLOGICAL NEURON**

There is lots of similarities between biological neuron and artificial neuron. Given below is the picture that describes the relationship well.





Here,

* Dendrites is act as input signal \* weight.
* Nucleus is act as additive junction i.e. summation.
* We have axon as Activation function which used to map input to output and
* Lastly the synapses or nerve ending act as output.

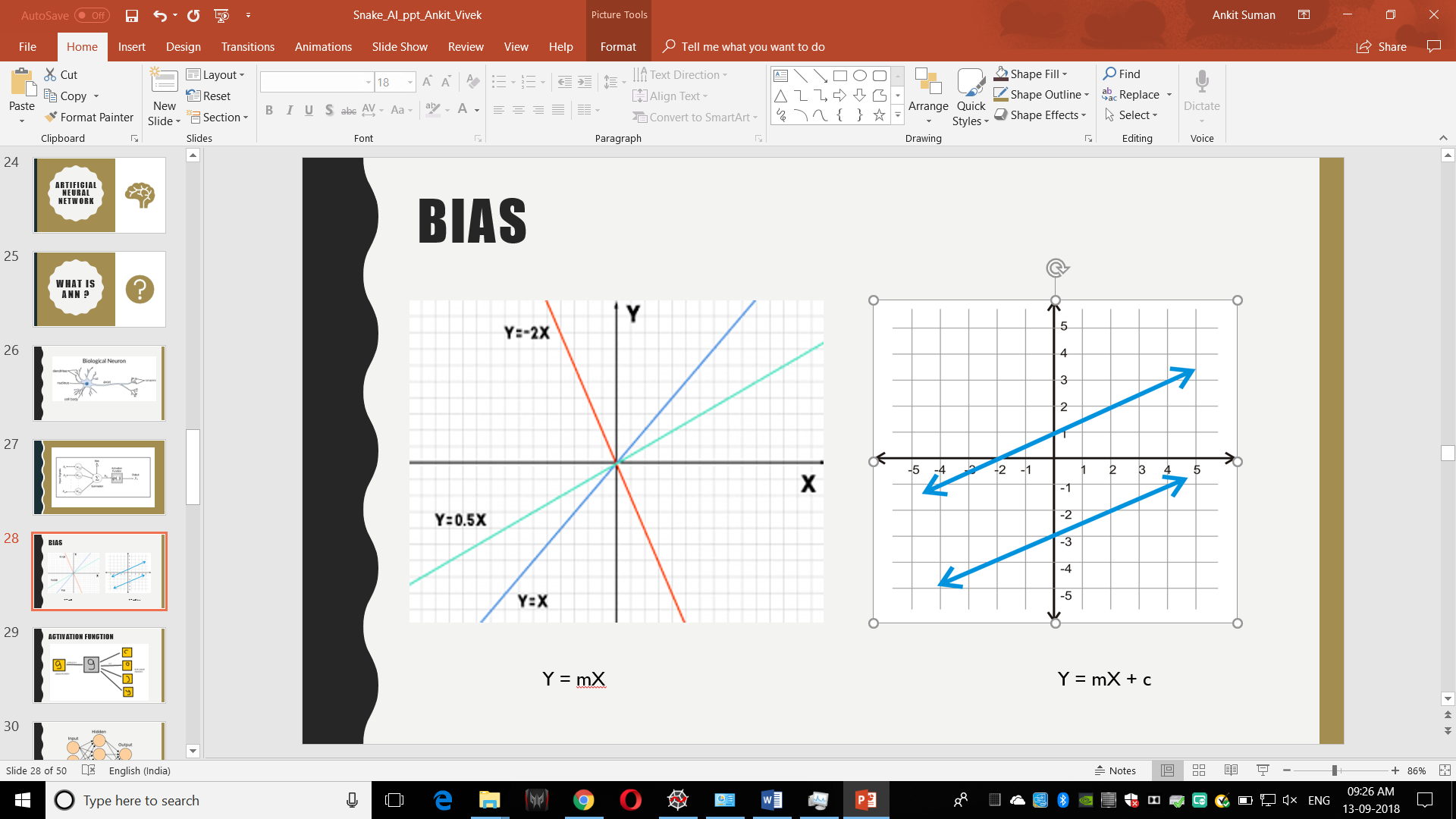
1. **WEIGHT:**

**weight** refers to the strength or amplitude of a connection between two nodes. A set of weighted inputs allows each artificial neuron or node in the system to produce related outputs. Weight is generally get multiplied with input and get used by the additive junction.

1. **BIAS:**
   * A bias unit is an "extra" neuron added to each pre-output layer that stores the value of 1. Bias units give freedom to best fit the given data. **A bias value allows you to shift the activation function to the left or right,** which may be critical for successful learning.
   * A simpler way to understand what the bias is: it is somehow similar to the constant *b* of a linear function

*Y = mX + c*

* + It allows you to move the line up and down to fit the prediction with the data better. Without *c* the line always goes through the origin (0, 0) and you may get a poorer fit.



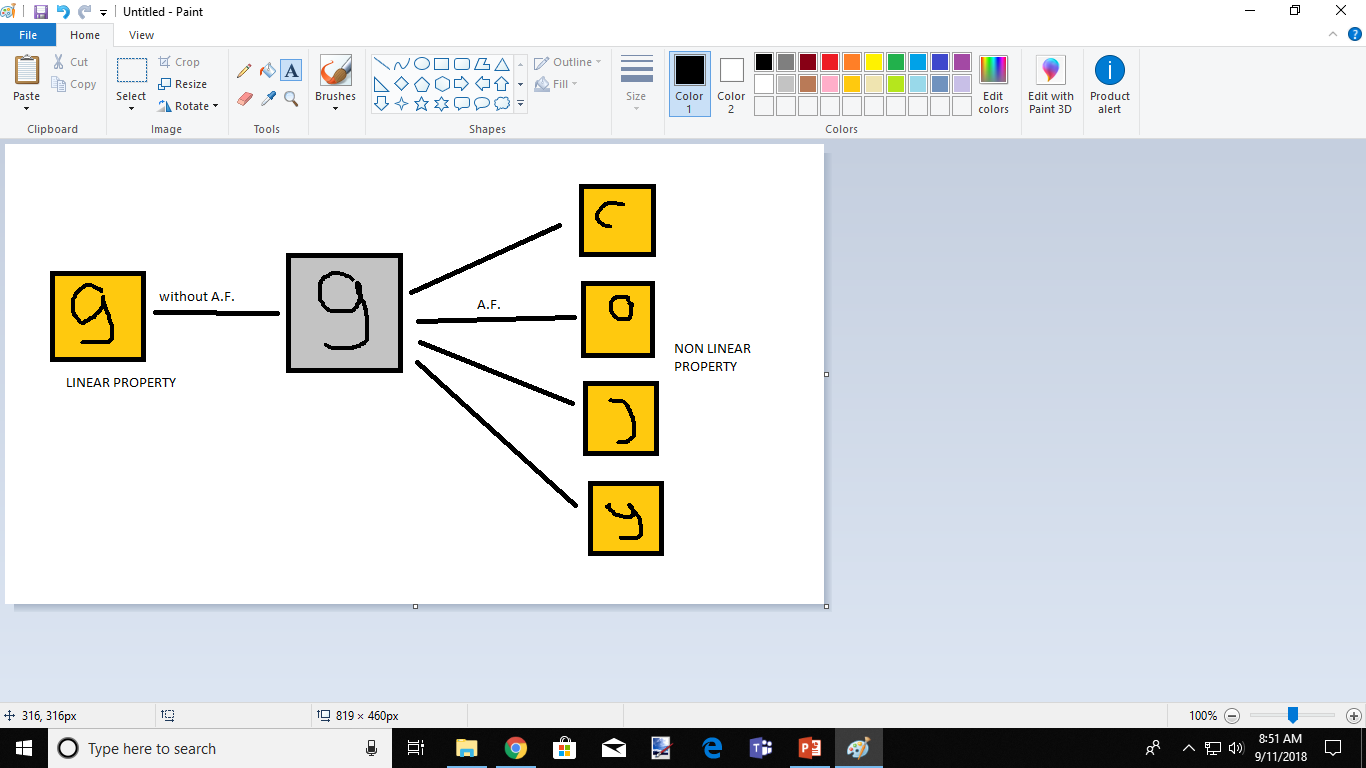
* So, here we can see that we have a line equation as Y = mx. so, in this equation whenever we try to put the value of x and y, it always going to plot the line that is going to pass through the center i.e. (0,0). hence, we can assume that we cannot fit the all line with this equation.
* On the other hand, we have an equation y = mx + c. so, whenever we try to plot the line through this equation by passing the value of x and y, we are going to fit all the line equation.
* So, we can see the difference in both of the equation that we have **‘c’** as constant in the equation 2 here which is known as constant in neural network term we referred it as ‘**BIAS**’. Bias give us freedom to best fit the given data.

1. **ACTIVATION FUNCTION:**

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

### **The question arises that why can’t we do it without activating the input signal?**

* If we do not apply an Activation function, then the output signal would simply be a simple **linear function**.
* A linear function is just a polynomial of **one degree.**Now, a linear equation is easy to solve but they are limited in their complexity and have less power to learn complex functional mappings from data.
* A Neural Network without Activation function would simply be a **Linear regression Model,**which has limited power and does not performs good most of the times. We want our Neural Network to not just learn and compute a linear function but something more complicated than that. Also without activation function our Neural network would not be able to learn and model other complicated kinds of data such as images, videos, audio, speech etc
* That is why we use Artificial Neural network techniques such asDeep learning to make sense of something complicated, high dimensional, non-linear -big datasets, where the model has lots and lots of hidden layers in between and has a very complicated architecture which helps us to make sense and extract knowledge form such complicated big datasets.



# Let us consider an example of predicting the letter ‘9’.

# So, if we don’t apply the activation function to then, it is only going to compare the given image with the stored image for the prediction, there is no extra property to compare with.

# Next, we apply the activation function to then, it is going to extract some property which is going to be complex and non-linear property. This non-linear property allows to train the model efficiently do to do better prediction.

# Non-Linear property increases the accuracy of the model.

# **BUILDING AN ANN MODEL**

**1) Importing the dataset:**

dataset = pd.read\_csv('SNAKE\_A.csv')  
dataset = dataset.dropna()  
  
X = dataset.iloc[:, 0:9].values  
y = dataset.iloc[:, 9].values

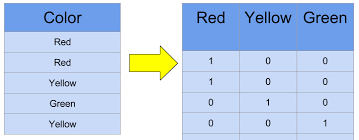
* The first step in building the ANN model is that we require a dataset on which we are going to build our out model.
* For that we are using **pandas** as **pd** and **dataset = pd.csv(‘SNAKE\_A.csv’)** to load our data file into the variable dataset.
* Next, we have used ‘**dropna,**’ the use of ‘**dropna**’ is to drop the rows that are having the ‘**null**’ or ‘**na**’ values.
* Since we have more than 100k data in our dataset so, we have used it to remove some null value around 20 to 30. Suppose, if we have data around 300 in our dataset and we have missing value around 20-30 then, in that case we cannot remove the null value because it makes our dataset small and inefficient to make prediction with such small data. So, in this case use the **IMPUTER** class, which replace the missing value with the **MEAN**, **MEDIAN**, **MODE**.
* Next, we have spited the independent and dependent column into X and y respectively.
* We have used ‘**iloc**’ for the splitting which is used to select the column based on their index value. We have ‘**loc**’, which can be used to select column based on their column name. and we have ‘**ix**’, which is the hybrid version of both which can be used for both index and column name selection.
* At the end we have used **‘values’**, this is because we are only taking values from the dataset not the header and index (column names).

**2) Making prediction column as categorical:**

ohe = OneHotEncoder(sparse=False)

y = y.reshape(-1, 1)  
y = ohe.fit\_transform(y)

* One hot encoder generally encodes the data in given column into the number of column and assign binary value to it. For example



* Here, we have used a parameter **sparse = False,** which is going to return an array and then we reshape it. If we take **sparse = Ture**, which means that we want the sparse matrix in return (i.e. most of the element of the matrix will be zero).
* Next, we have used reshape function to reshape the matrix in as many rows possible with only one column. **‘-1’** indicates number of the row i.e. all the rows and **‘1’** indicates the number of columns.
* Next, we have used the ‘**fit\_transform’** to transform and fit the data into y as shown the pic above.

**3) Splitting of the data into train and test:**

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.2, , stratify=True)

* For training and testing our model we require the train and test data for that we split our data X and y into train data and test data.
* We can see that here, ‘**test\_size = 0.2’,** which means that we are going to split our data into 2:8 i.e. 80% of data are going to be our train data and 20% of our data are going to be test data.
* Train data are used make the prediction model and test data are used to do testing against the trained model.
* Next, we have ‘**stratify = true’**, which means we are going to split the data into equal ratio.

**4) Initializing Neural Network:**

model = Sequential ()

* Next, we initialize the model with the sequential class.
* Sequential class is nothing just a linear stack of layers. Since, we are going to add different layers in the model for that we require a base where we can define our different layers.
* We can create a sequential model by passing a list of layer instances to the constructor:

**5) Defining First Hidden Layer in Neural Network:**

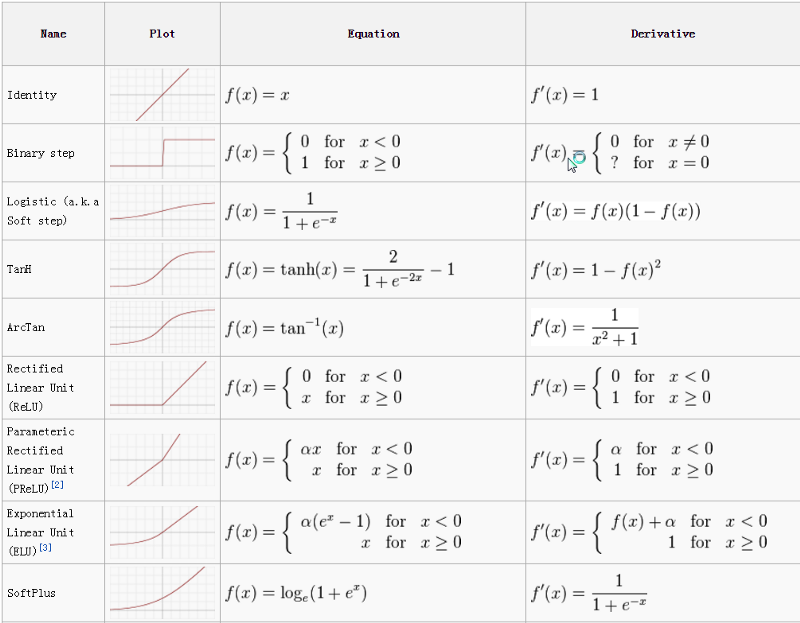
model.add(Dense(output\_dim = 80, init = /

'uniform', activation = 'relu', input\_dim = 9))

* Here, we have used **‘add’** to add this layer to the model and we have **‘Dense’ layer**, dense means fully connected (each and every neuron of one layer is connected with each and every neuron to its adjacent layer). We have **‘Dropout’** **layer**, setting a fraction of inputs to zero in an effort to reduce overfitting. We have **‘Merge’ layer,** combine the inputs from multiple models into a single model.
* Here, we are defining our first hidden layer, this layer is consisting of input value (here we are providing 9 input) and output value (output is given by this layer is 80).
* Next, we are defining the initial weight as **init = ‘uniform’,** we initialize the network weights to a small random number generated from a **uniform distribution.** In uniform, weight ranges from 0 to 0.05.
* similarly, we can take **init = ‘normal’**, weights are assigned to small gaussian random value range from 0 (mean) to 0.05 (standard deviation). **Gaussian distribution** is used for weight assignment
* And we have another as **init = ’zero’,** all weights are set to zero values.
* Next, we are defining the activation function **‘Relu’ {rectified linear unit}.** In **‘Relu’** any value less than ‘0’ (i.e. negative value) will be converted to ’ 0’ and for any value greater equal to ‘0’, there will be no change.
* 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 its derivative **both are** **monotonic**.

****

**CHEATSHEET FOR ACTIVATION FUNCTION.**



**6) Defining Second Hidden Layer in Neural Network:**

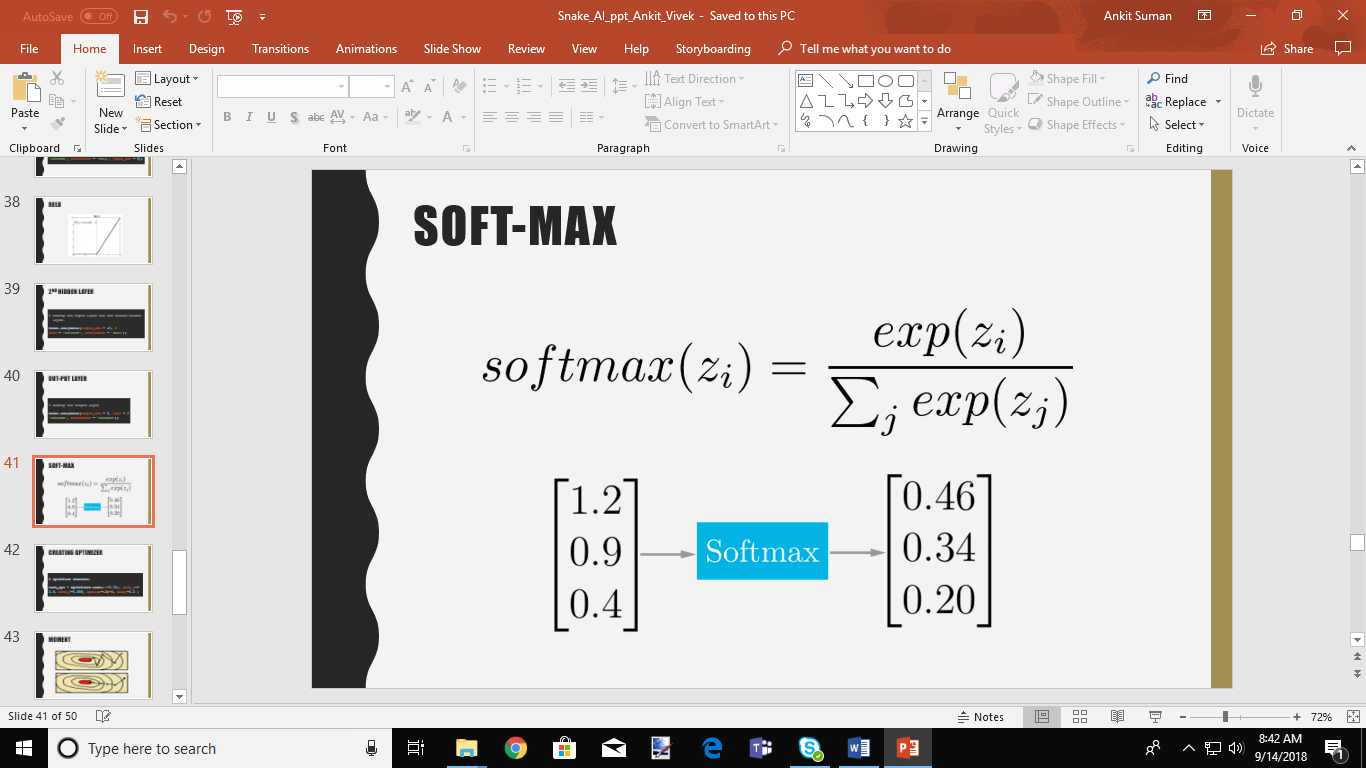
model.add(Dense(output\_dim = 50,init = 'uniform', activation = 'relu’))

**7) Defining Output Layer in Neural Network:**

# Adding the output layer

model.add(Dense(output\_dim = 3, init = 'uniform’, activation = 'softmax’))

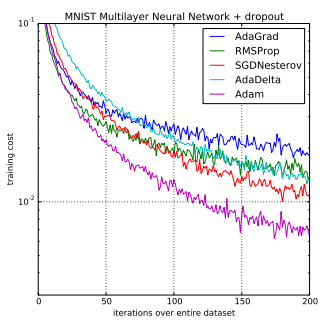
* Here, we have used activation function as **‘softmax’** because RELU can only be used in the hidden layer not in the output layer.
* **Softmax** function is used to find the probability distribution.
* In the output layer **‘softmax’** generates probability for each classes and class having the highest probability will get selected.



**8) Creating optimizer for our network:**

Adam\_opt = optimizers.Adam(lr = 0.001, beta\_1 = 0.9, beta\_2=0.999, epsilon = 10E-8, decay = 0.0 )

* Here, we are creating an optimizer for our neural network. Optimization algorithms helps us to **minimize (or maximize)** an **Objective** function (another name for **Error** function) **E(x)**which is simply a mathematical function dependent on the Model’s internal **learnable parameters** which are used in computing the target values(**Y**) from the set of predictors(**X**) used in the model.
* For example — we call the **Weights(W)** and the **Bias(b)** values of the neural network as its internal learnable parameters which are used in computing the output values and are learned and updated in the direction of optimal solution i.e. minimizing the **Loss**by the network’s training process and also play a major role in the **training** process of the Neural Network Model.
* There are different optimization algorithms, but we have used the ADAM optimization because it gives best result in deep learning.



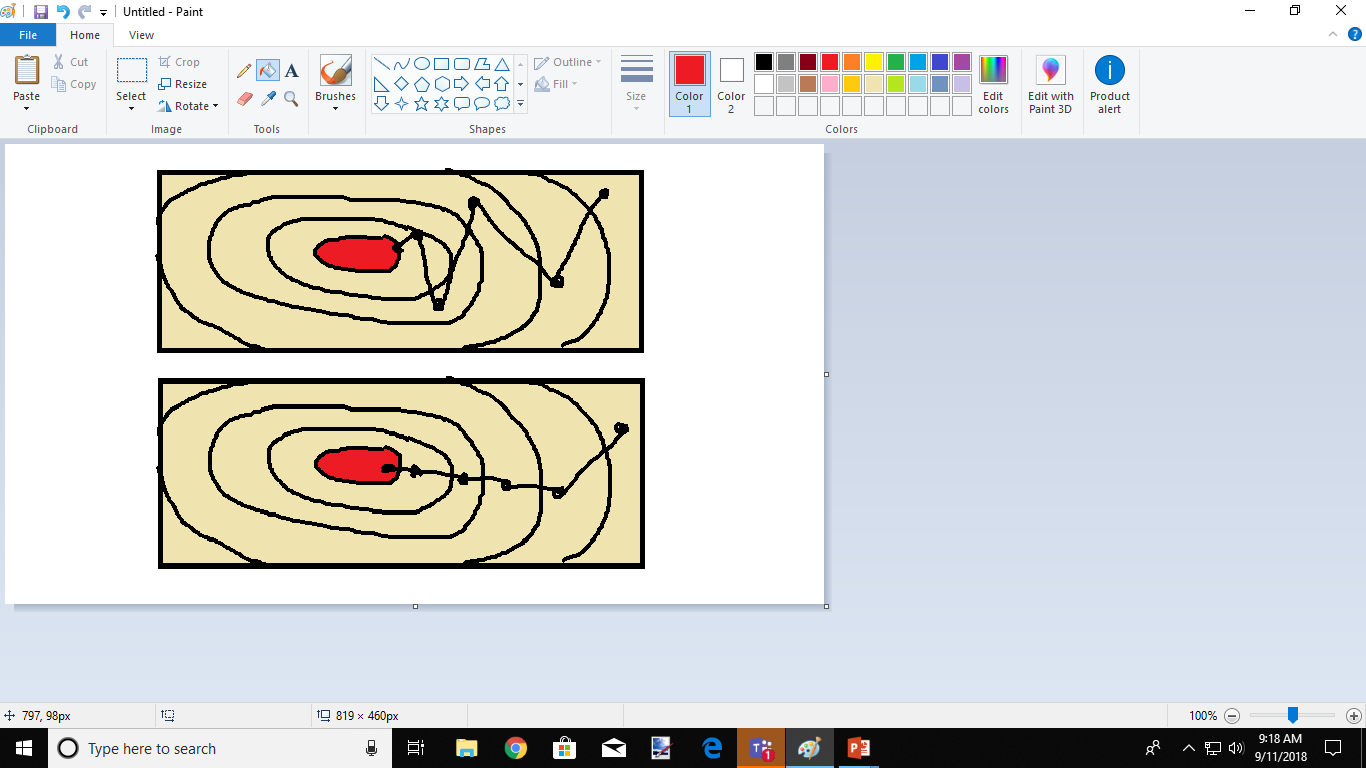
* The parameters of ADAM optimizer are:
  + **ALPHA (Learning Rate):**

Also referred to as the learning rate or step size. The proportion that weights are updated (e.g. 0.001). Larger values (e.g. 0.3) results in faster initial learning before the rate is updated. Smaller values (e.g. 1.0E-5) slow learning right down during training

* + **BETA1 (Rate of decay for moment 1):** The exponential decay rate for the first moment estimates (e.g. 0.9).
  + **BETA2 (Rate of decay for moment 2):** The exponential decay rate for the second-moment estimates (e.g. 0.999). This value should be set close to 1.0 on problems with a sparse gradient (e.g. NLP and computer vision problems).
  + **DECAY:** Decay rate for learning.
  + **EPSILON:**  Is a very small number to prevent any division by zero in the implementation (e.g. 10E-8).

**Good default settings for the tested machine learning problems are alpha=0.001, beta1=0.9, beta2=0.999 and epsilon=10−8**

**MOMENT: Moment generally speed up the iteration process as show in the given figure, first one is without moment and second one is with moment. (total iteration is 6 in both).**

****

**9) Compiling our model.**

model.compile(optimizer = Adam\_opt,loss = 'categorical\_crossentropy’, metrics = ['accuracy’])

* For compiling our model, we have used optimizer as **ADAM** that we have created, and loss function is taken as **'categorical\_crossentropy**’ because we are dealing with the classification problem and we have matrices as **accuracy** since, we are trying to figure out the accuracy of the model.

**10) Saving and loading our model.**

* The main idea behind the saving and loading the model is to do prediction efficiently. Since the compilation of the model took time. Suppose we have created our model and want to do prediction, in that case we cannot rerun the whole model to do prediction so, instead of that we simply save the model and load the model whenever required for doing prediction. We can simply pass the input parameters and get the output.
* There are several methods for saving the model.
  + **Joblib**
  + **Pickle**
  + **H5py (we have used), etc.**