Bank Note Authentication Using Neural Networks

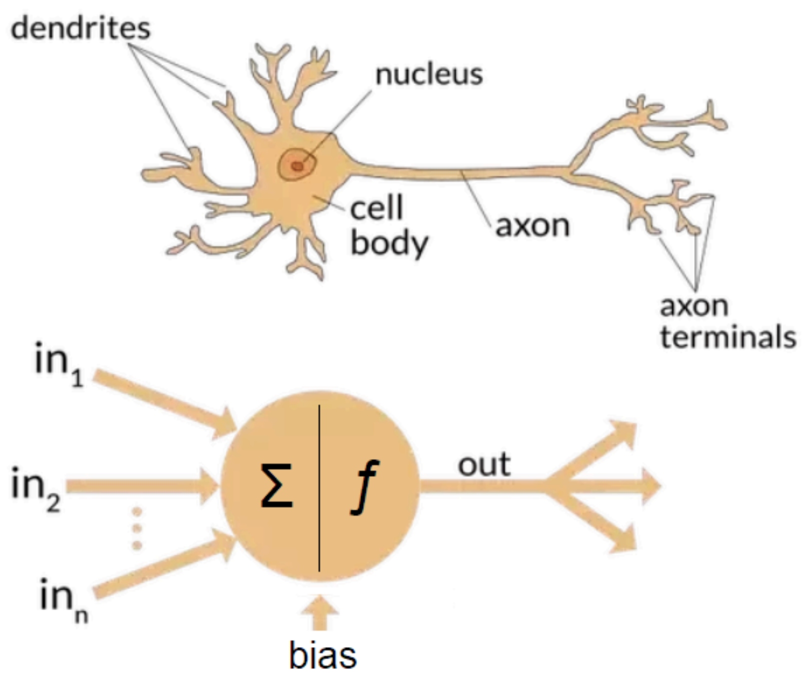
- SHIKHA SEN
-shikhasenthakur@gmail.com
+919958272728

There are already existing conventional algorithmic techniques and strategies which are very much capable of problem solving but with the given specific instructions and unless the specific learning and instructive steps are known to computers, they can not solve a problem. So we can say that human intervention at major steps is a must to feed instructions to the computers. But what if a human is not clear of what instructions should be given to the computer and here, the conventional algorithmic approach could be a failure. Hence, Neural Networks comes into play.

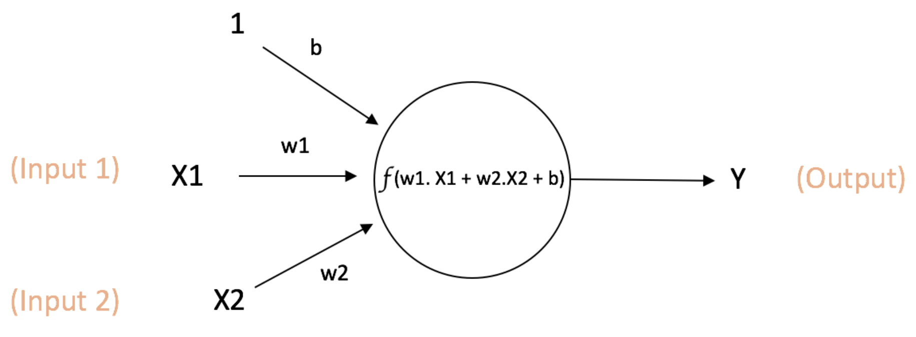
Neural networks can deal with data very much like a human cerebrum does. These networks can learn from past and examples, you can just program them like conventional algorithms to process and therefore they don't need all and every instruction to perform a task. It's not a new term, it was first instituted noticeable all around in 1992 but due the lack of enough data source and computational resources, it could not work then but now we are blessed.

Working of Neural Network

How The human brain exactly works is still kind of mysterious. Because f we understand the brain, we would understand its power and limits. The neural network somewhat resembles human brain. The building block of neural network is the neuron. A biological neuron receives some input, generates a non linear response and gives the response as the output.



So we can say that Artificial Neural Networks are computing frameworks motivated by the organic neural networks that constitute to creature(animals) brain. Such framework learns (and logically improves performance)to tackle tasks by considering examples, usually without task-explicit programming.



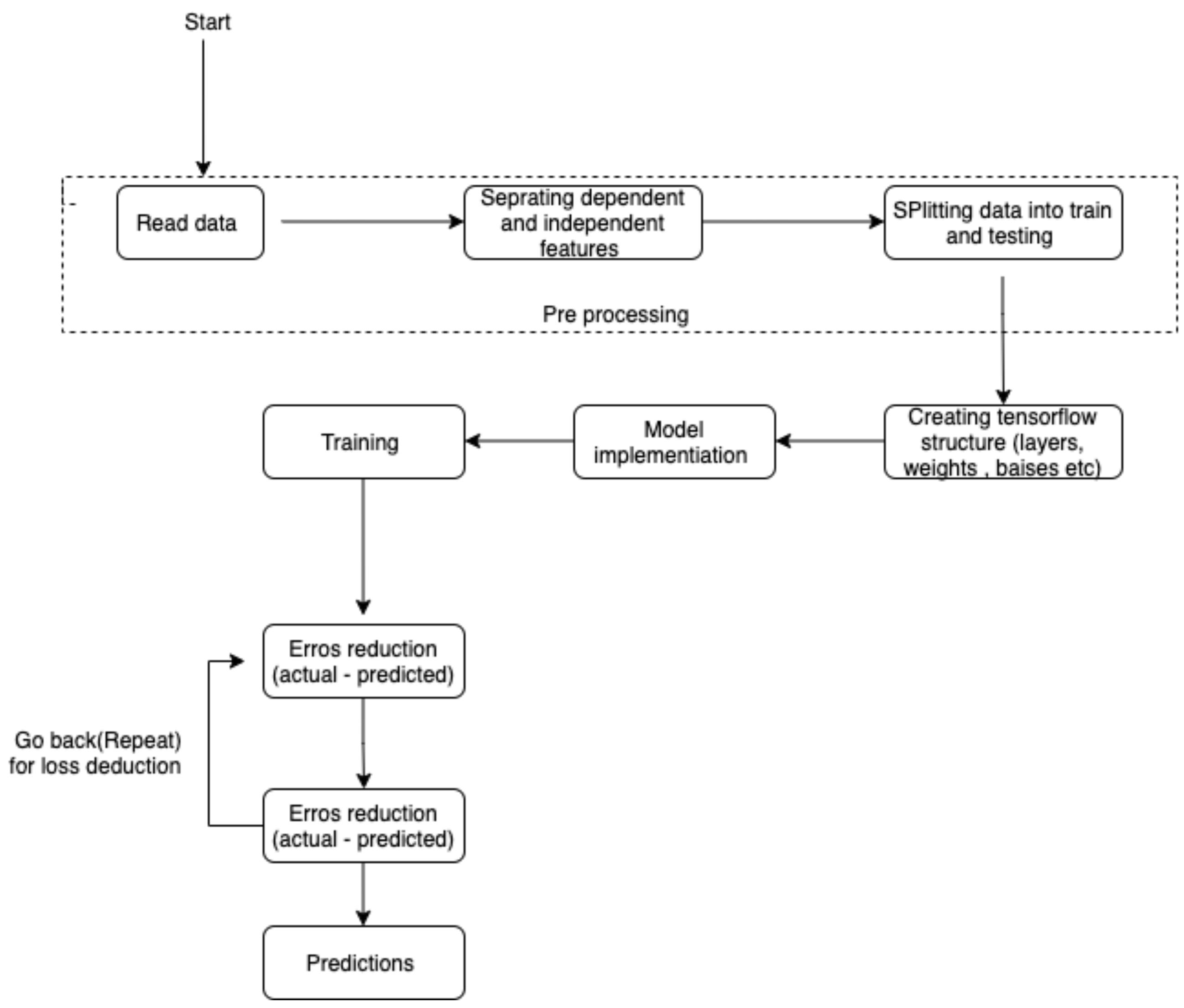
Output of neuron = Y= f(w1. X1 + w2.X2 + b)

Here's the working of artificial neurone called perceptron is explained.
A network may have three types of layers: input layers that take raw input from the domain, **hidden layers** that take input from another layer and pass output to another layer, and **output layers** that make a prediction.
We have various inputs x1, x2 … xn and their corresponding weights and biases as well, then we calculated the weighted sum of these inputs + biases and then we pass it to the 'Activation function', this activation function provided a threshold value, so above that value the neurone will fire else it would not.
A proper complete neural network involves a lot of neurones with their specific activation functions to give out the correct output.

Activation Function:
An activation function in a neural network characterizes how the weighted amount of the input is changed into an output from layer or layers of the network.
There are many activation functions like step, sigmoid (Equation : $A = 1/(1 + e^{-x})$), Softmax Function is also a type of sigmoid function but is handy when we are trying to handle classification problems. etc.

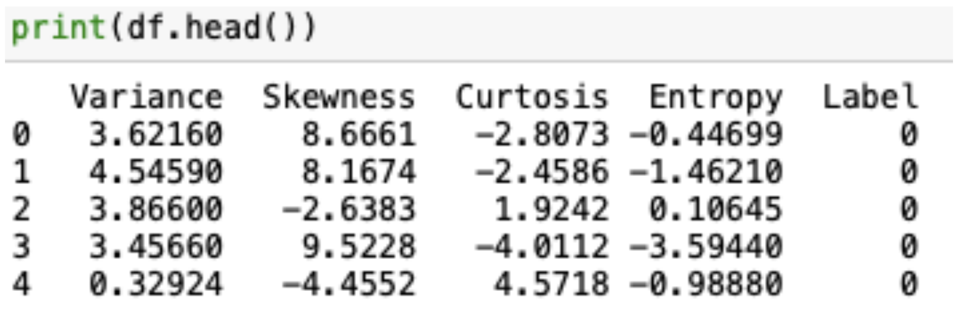
USE CASE

Problem statement - Using ANN, we need to identify if the bank note is real(1) or fake(0)


Start
Read data
Seprating dependent and independent features
SPlitting data into train and testing
Pre processing
Creating tensorflow structure (layers, weights , baises etc)
Model implementiation
Training
Erros reduction (actual - predicted)
Go back(Repeat) for loss deduction
Erros reduction (actual - predicted)
Predictions

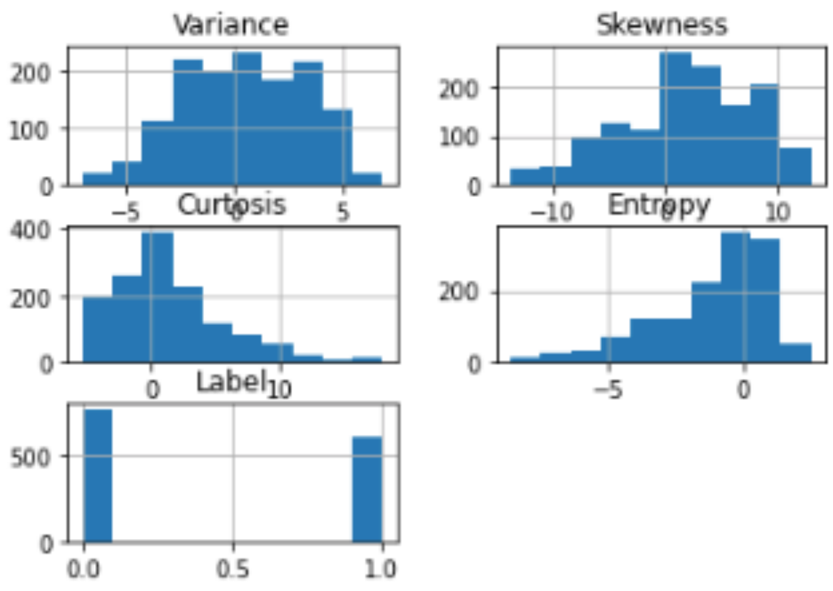
The data set looks like this, taken from the url = ‘https://raw.githubusercontent.com/jbrownlee/Datasets/master/banknote_authentication.csv'

Data were taken from pictures that were taken from veritable and fake banknote-like examples. The final data images have 400×400 pixels. Because of the object lens and distance to the explored object gray-scale, pictures with a resolution of around 660 dpi were acquired. Wavelet Transform apparatus were utilized to extracts features from pictures.



```
print(df.head())
```

```
   Variance  Skewness  Curtosis  Entropy  Label
0   3.62160    8.6661   -2.8073 -0.44699      0
1   4.54590    8.1674   -2.4586 -1.46210      0
2   3.86600   -2.6383    1.9242  0.10645      0
3   3.45660    9.5228   -4.0112 -3.59440      0
4   0.32924   -4.4552    4.5718 -0.98880      0
```

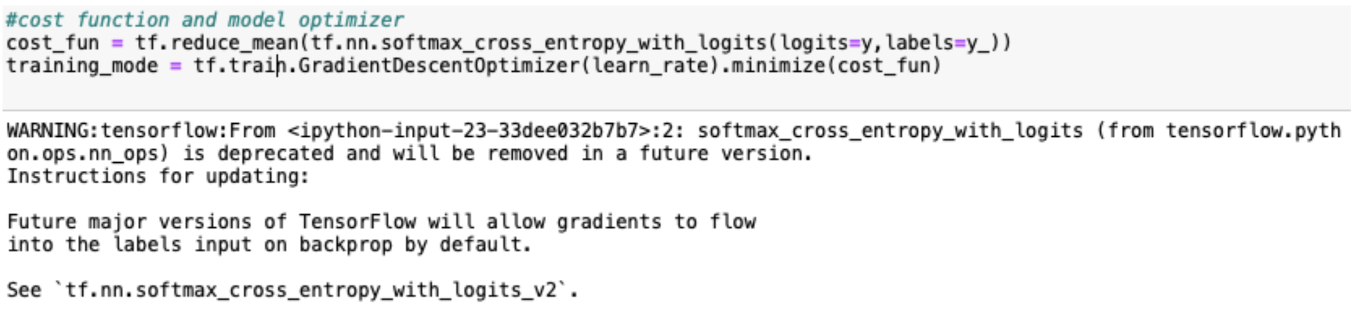


Importing Libraries
Import pandas as pd
Import numpy as np
Import Matplotlib.pylot as plt
import sklearn
Import tensor flow as tf
from sklearn.preprocessing import LabelEncoder
from sklearn.utils import shuffle
from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
The data has been split into 80:20 ratio .
Learning rate : 0.3
In particular, the learning rate is a configurable hyperparameter utilized in the preparation of neural networks that has a little positive worth, frequently in the reach somewhere in the range of 0.0 and 1.0. The learning rate controls how rapidly the model is adjusted to the issue and adapts tot he problem

Hidden layer with 10 neurones
x, and y_ holds the placeholder value for the model, w and b are weights and biases, y_ holds the actual value for the class(labels)

After variable initialisation for tensor flow,
We call the model for training :
And most importantly the cost func (cost function to calculate the error between the actual and model output) and training_mode is where the training begins



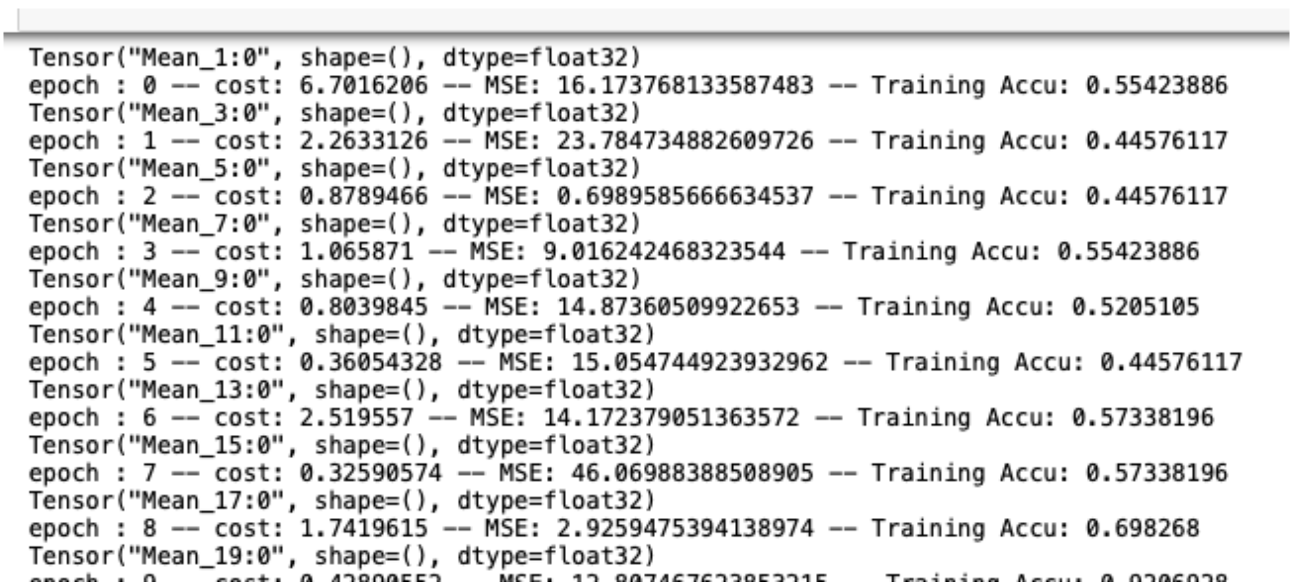
```
#cost function and model optimizer
cost_fun = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=y,labels=y_))
training_mode = tf.train.GradientDescentOptimizer(learn_rate).minimize(cost_fun)
```

```
WARNING:tensorflow:From <ipython-input-23-33dee032b7b7>:2: softmax_cross_entropy_with_logits (from tensorflow.pyth
on.ops.nn_ops) is deprecated and will be removed in a future version.
Instructions for updating:

Future major versions of TensorFlow will allow gradients to flow
into the labels input on backprop by default.

See `tf.nn.softmax_cross_entropy_with_logits_v2`.
```

A critical part of most man-made brainpower and AI/ML is looping, for example the framework improving over numerous cycles/iterations of training
In training_mode , Gradient Descent optimiser is a learning algorithm that attempts to minimise some error which is defined in the cost_function.



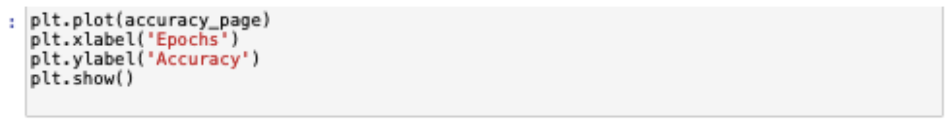
```
Tensor("Mean_1:0", shape=(), dtype=float32)
epoch : 0 -- cost: 6.7016206 -- MSE: 16.173768133587483 -- Training Accu: 0.55423886
Tensor("Mean_3:0", shape=(), dtype=float32)
epoch : 1 -- cost: 2.2633126 -- MSE: 23.784734882609726 -- Training Accu: 0.44576117
Tensor("Mean_5:0", shape=(), dtype=float32)
epoch : 2 -- cost: 0.8789466 -- MSE: 0.6989585666634537 -- Training Accu: 0.44576117
Tensor("Mean_7:0", shape=(), dtype=float32)
epoch : 3 -- cost: 1.065871 -- MSE: 9.016242468323544 -- Training Accu: 0.55423886
Tensor("Mean_9:0", shape=(), dtype=float32)
epoch : 4 -- cost: 0.8039845 -- MSE: 14.87360509922653 -- Training Accu: 0.5205105
Tensor("Mean_11:0", shape=(), dtype=float32)
epoch : 5 -- cost: 0.36054328 -- MSE: 15.054744923932962 -- Training Accu: 0.44576117
Tensor("Mean_13:0", shape=(), dtype=float32)
epoch : 6 -- cost: 2.519557 -- MSE: 14.172379051363572 -- Training Accu: 0.57338196
Tensor("Mean_15:0", shape=(), dtype=float32)
epoch : 7 -- cost: 0.32590574 -- MSE: 46.06988388508905 -- Training Accu: 0.57338196
Tensor("Mean_17:0", shape=(), dtype=float32)
epoch : 8 -- cost: 1.7419615 -- MSE: 2.9259475394138974 -- Training Accu: 0.698268
Tensor("Mean_19:0", shape=(), dtype=float32)
```

After that we run the epochs which was taken as 100.

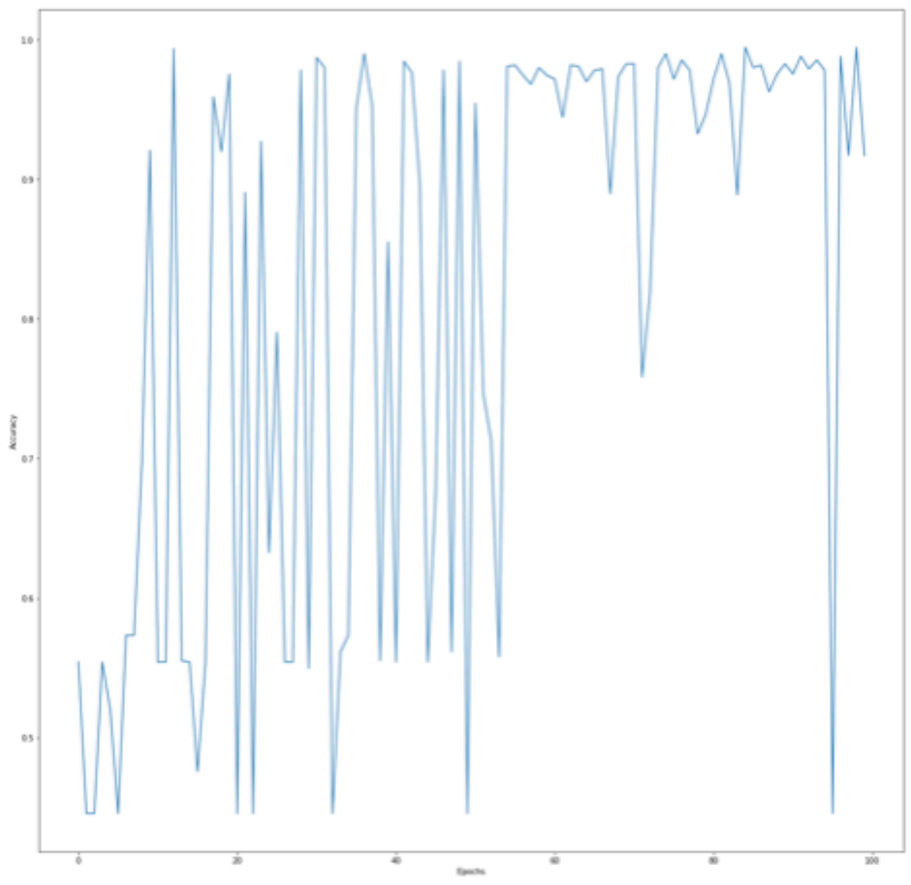
For every epoch we calculate the change in error and also accuracy on the training data.

The plot on training data looks somewhat like :

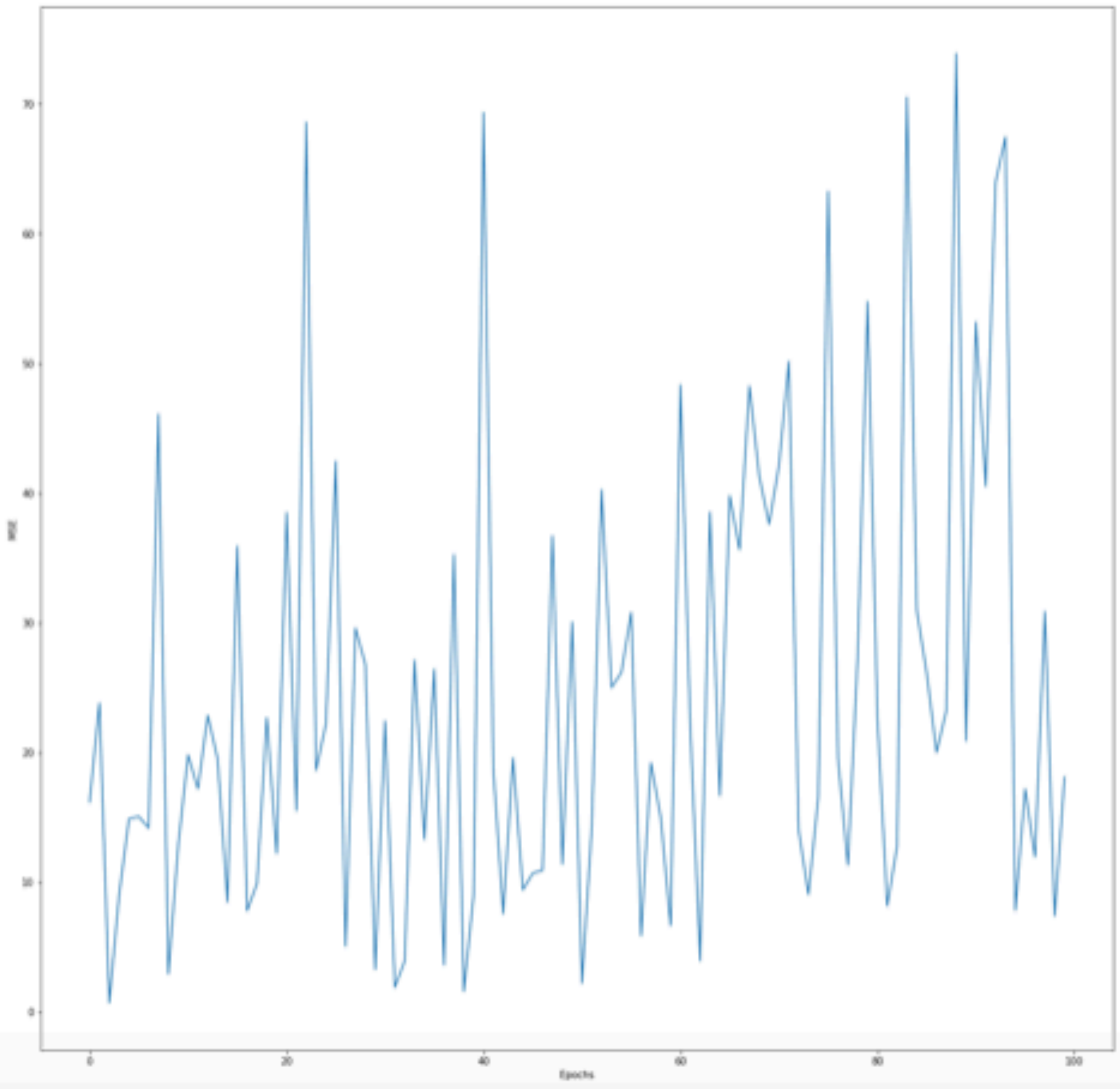
(Using 1 hidden layer)



```
plt.plot(accuracy_page)
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.show()
```



```
plt.plot(mse_page)
plt.xlabel('Epochs')
plt.ylabel('MSE')
plt.show()
```



The plot on epochs vs MSE

```
Test accuracy 0.9927273
Mse: 32.184365667394246
```