SUMMARY FOR TENSORFLOW:

1.To train tensorflow model we have to bring down the values between 0 & 1.

2. To bring it down divide the X_train, X_test value by 255(the higher value)

3. Build the model

    a. Sequenial Model

    b. Flatten Layer : It basically transform the data into 1-D so that it can be fed into next layer.

    c. Dense Layer :

optimizer

loss function

activation function

metrics

model.evaluate(X_test, y_test)

model.predict = Set of array : continuous values

model.predict_classes : predict the particular class

To check the max values in a array of list then : np.argmax(pred[0])

Neural Networks :

Neural-Networks are considered Universal Function Approximators. It means that they can compute and learn any function at all. Almost any process we can think of can be represented as a functional computation in Neural Networks.

ANN : Artificial Neural Networks : System Built of a large number of simple elements, called neurons or perceptrons. Each neuron can make simple decisions and feeds those decisions to other neurons, organized in interconected laters.

Shallow feedforward neural networks : Single hidden layers along with output and input layer.

Deep Neural Networks : Multiple hidden layers along with input and output layer.

    a. Each input neuron gets multiplied with the particular weights and then it passes through the activation function and then the output is produced.

If activation function not applied then the output signal would be a simple linear function. A linear function is just a polynomial of one degree.

Also another important feature of a Activation function is that it should be differentiable. We need it to be this way so as to perform backpropogation optimization strategy while propogating backwards in the network to compute gradients of Error(loss) with respect to Weights and then accordingly optimize weights using Gradient descend or any other Optimization technique to reduce Error.

Activation function :

a. **sigmoid** : For two output classes:

It is a activation function of form $f(x) = 1 / 1 + \exp(-x)$. Its Range is between 0 and 1. It is a S — shaped curve. It is easy to understand and apply but it has major reasons which have made it fall out of popularity -

Vanishing gradient problem

Secondly , its output isn't zero centered. It makes the gradient updates go too far in different directions. $0 < output < 1$, and it makes optimization harder.

Sigmoids saturate and kill gradients.

Sigmoids have slow convergence.

b. **tanh(Hyperbolic tangent)** :

It's mathamatical formula is $f(x) = 1 - \exp(-2x) / 1 + \exp(-2x)$. Now it's output is zero centered because its range in between -1 to 1 i.e $-1 < output < 1$. Hence optimization is easier in this method hence in practice it is always preferred over Sigmoid function . But still it suffers from Vanishing gradient problem.

c. **relu(Rectified linear units) :**

It was recently proved that it had 6 times improvement in convergence from Tanh function. It's just $R(x) = \max(0,x)$ i.e if $x < 0$, $R(x) = 0$ and if $x >= 0$, $R(x) = x$. Hence as seeing the mathamatical form of this function we can see that it is very simple and efficinent . A lot of times in Machine learning and computer science we notice that most simple and consistent techniques and methods are only preferred and are best. Hence it avoids and rectifies vanishing gradient problem . Almost all deep learning Models use ReLu nowadays

It is capable of outputting a true zero value allowing the activation of hidden layers in neural networks to contain one or more true zero values called Representational Sparsity

The downside for being zero for all negative values called dying ReLU. So if once neuron gets negative it is unlikely for it to recover. This is called “dying ReLU” problem

ReLU generally not used in RNN because they can have very large outputs so they might be expected to be far more likely to explode than units that have bounded values

But its limitation is that it should only be used within Hidden layers of a Neural Network Model

Another problem with ReLu is that some gradients can be fragile during training and can die. It can cause a weight update which will makes it never activate on any data point again. Simply saying that ReLu could result in Dead Neurons.

To fix this problem another modification was introduced called Leaky ReLu to fix the problem of dying neurons. It introduces a small slope to keep the updates alive.

We then have another variant made form both ReLu and Leaky ReLu called Maxout function .

d. **leakyrelu** :

Leaky ReLUs attempt to fix the “dying ReLU” problem. Instead of the function being zero when $x < 0$, a leaky ReLU gives a small negative slope

e. **softmax** :

f. **Maxout :**

The Maxout activation is a generalization of the ReLU and the leaky ReLU functions. It is a learnable activation function

It is a piecewise linear function that returns the maximum of the inputs, designed to be used in conjunction with the dropout regularization technique.

Both ReLU and leaky ReLU are special cases of Maxout. The Maxout neuron, therefore, enjoys all the benefits of a ReLU unit (linear regime of operation, no saturation) and does not have its drawbacks (dying ReLU).

However, it doubles the total number of parameters for each neuron and hence, a higher total number of parameters need to be trained

BEST PRACTICES :

Use ReLU in hidden layer activation, but be careful with the learning rate and monitor the fraction of dead units.

If ReLU is giving problems. Try Leaky ReLU, PReLU, Maxout. Do not use sigmoid

Normalize the data in order to achieve higher validation accuracy, and standardize if you need the results faster

The sigmoid and hyperbolic tangent activation functions cannot be used in networks with many layers due to the vanishing gradient problem.

VANISHING AND EXPLODING GRADIENTS:

During training of a deep network when the gradients are being propagated back in time all the way to the initial layer. The gradients coming from the deeper layers have to go through continuous matrix multiplications because of the the chain rule, and as they approach the earlier layers, if they have small values (<1), they shrink exponentially until they vanish and make it impossible for the model to learn , this is the vanishing gradient problem. While on the other hand if they have large values (>1) they get larger and eventually blow up and crash the model, this is the exploding gradient problem.

Neural networks Building :

NN includes two important types of artificial neuron(perceptron and the sigmoid neuron), and the stochastic gradient descent.

NLP:

Create Pipeline

```
clf = Pipeline([('tfidf',TfIdfVectorizer()), ('clf',RandomForestClassifier(n_estimators=100, n_jobs=-1))])
```