Deep Learning
Minor-2 Report
(M22MA003)

Roll Number - ABC	003
DD/MM/YYYY	21/06/1997
Dataset	TinyImageNet
Weight Initialisation	Xavier Normal
Pooling Operation	Average Pool

Question 1 :

A. Teacher Network.

Architecture Specifications:-

Layers	12 Conv Layers + 1 Pool Layer + 1 FC (512 Nodes)
Loss Function Used	CrossEntropy

Steps Followed:

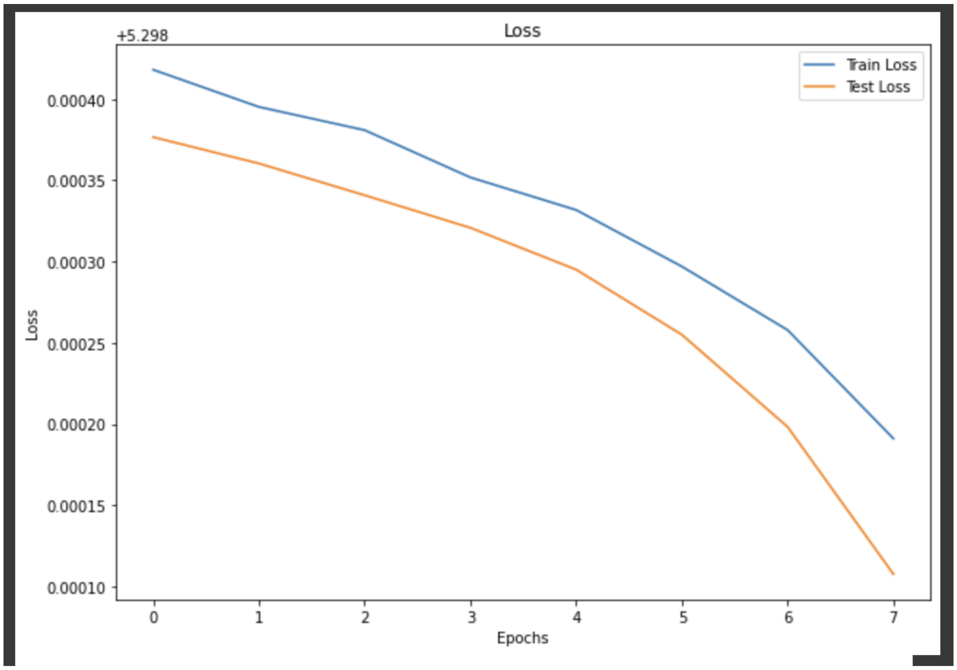
1. Imported the required libraries.
2. Defined Parameters - number of epochs, batch size, learning rate.
3. Using the TinyImageNet zip file to import the images and labels.
4. Load data using ImageFolder and DataLoader. Without normalization:-

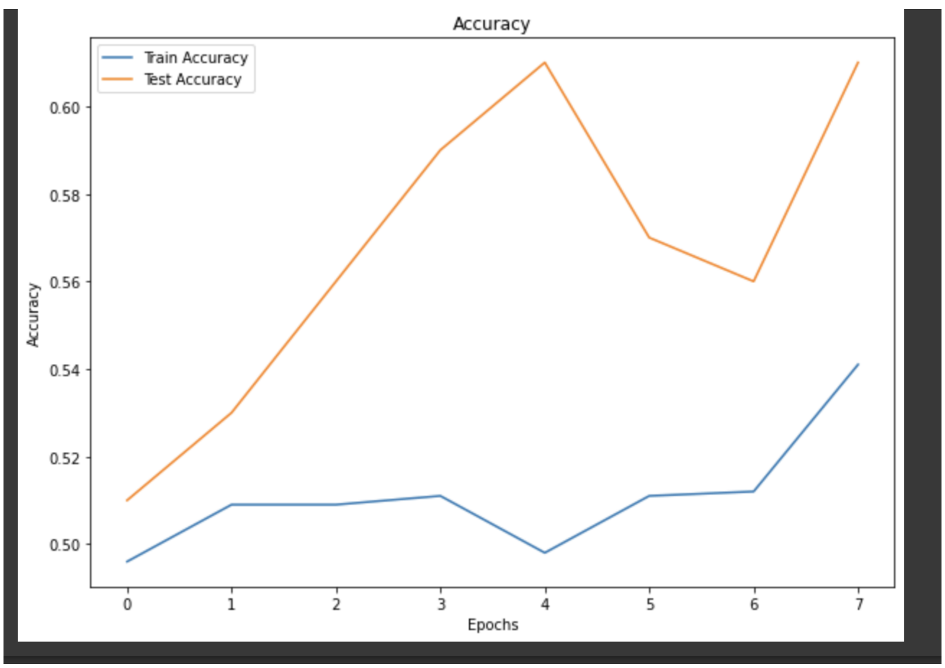
After Normalisaton:-

5. Defined a CNN with 12 filters in the first layer.
6. Added a total of 12 Feature Extraction Layers and one pooling layer.
7. Initialized the respective weights of the six layers using Xavier init.
8. Defined two linear layers out of which preceding is Fully Connected Layer and later is output layer.
9. Performed training on the CNN model with 8 epochs. Below are the results.

```
Epoch [4/8], Step [2000/2000], Train Accuracy: 0.511 , Train Loss: 5.2973
Epoch [4/8], Test Accuracy of the network: 0.59 %, Test Loss: 5.2983
------------------------------------------------------------------------------------------------------------
0.511 tensor(5.2984, device='cuda:0', grad_fn=<DivBackward0>) 0.59 tensor(5.2983, device='cuda:0')
Epoch [5/8], Step [500/2000], Train Accuracy: 0.420 , Train Loss: 5.2987
Epoch [5/8], Step [1000/2000], Train Accuracy: 0.492 , Train Loss: 5.3006
Epoch [5/8], Step [1500/2000], Train Accuracy: 0.499 , Train Loss: 5.2970
Epoch [5/8], Step [2000/2000], Train Accuracy: 0.498 , Train Loss: 5.2968
Epoch [5/8], Test Accuracy of the network: 0.61 %, Test Loss: 5.2983
------------------------------------------------------------------------------------------------------------
0.498 tensor(5.2983, device='cuda:0', grad_fn=<DivBackward0>) 0.61 tensor(5.2983, device='cuda:0')
Epoch [6/8], Step [500/2000], Train Accuracy: 0.516 , Train Loss: 5.2993
Epoch [6/8], Step [1000/2000], Train Accuracy: 0.530 , Train Loss: 5.2997
Epoch [6/8], Step [1500/2000], Train Accuracy: 0.516 , Train Loss: 5.2989
Epoch [6/8], Step [2000/2000], Train Accuracy: 0.511 , Train Loss: 5.2984
Epoch [6/8], Test Accuracy of the network: 0.57 %, Test Loss: 5.2983
------------------------------------------------------------------------------------------------------------
0.511 tensor(5.2983, device='cuda:0', grad_fn=<DivBackward0>) 0.57 tensor(5.2983, device='cuda:0')
Epoch [7/8], Step [500/2000], Train Accuracy: 0.548 , Train Loss: 5.2986
Epoch [7/8], Step [1000/2000], Train Accuracy: 0.550 , Train Loss: 5.2958
Epoch [7/8], Step [1500/2000], Train Accuracy: 0.525 , Train Loss: 5.2983
Epoch [7/8], Step [2000/2000], Train Accuracy: 0.512 , Train Loss: 5.2987
Epoch [7/8], Test Accuracy of the network: 0.56 %, Test Loss: 5.2982
------------------------------------------------------------------------------------------------------------
0.512 tensor(5.2983, device='cuda:0', grad_fn=<DivBackward0>) 0.56 tensor(5.2982, device='cuda:0')
Epoch [8/8], Step [500/2000], Train Accuracy: 0.524 , Train Loss: 5.2972
Epoch [8/8], Step [1000/2000], Train Accuracy: 0.560 , Train Loss: 5.2971
Epoch [8/8], Step [1500/2000], Train Accuracy: 0.560 , Train Loss: 5.2987
Epoch [8/8], Step [2000/2000], Train Accuracy: 0.541 , Train Loss: 5.2993
Epoch [8/8], Test Accuracy of the network: 0.61 %, Test Loss: 5.2981
------------------------------------------------------------------------------------------------------------
0.541 tensor(5.2982, device='cuda:0', grad_fn=<DivBackward0>) 0.61 tensor(5.2981, device='cuda:0')
Finished Training Teacher Network
```

10. Testing results on the Teacher network





[Colab Link Question 1]

B. Student Network

Layers	4 Conv Layers + 1 Pool Layer + 1 FC (512 Nodes)
Loss Function Used	CrossEntropy + Knowledge Distillation Loss

1. Defined a CNN with 12 filters in the first layer.
2. Added a total of 4 Feature Extraction Layers and one pooling layer.
3. Initialized the respective weights of the six layers using Xavier_normal initialisation.
4. Defined two linear layers out of which preceding is Fully Connected Layer and later is output layer.
5. Performed training on the CNN model with 8 epochs. Below are the training results.

```
Epoch [5/8], Step [2000/2000], Train Accuracy: 1.307 , Train Loss: 3.6054
Epoch [5/8], Test Accuracy of the network: 1.58 %, Test Loss: 5.2350
------------------------------------------------------------------------------------------------------------
1.307 tensor(3.6681, device='cuda:0', grad_fn=<DivBackward0>) 1.58 tensor(5.2350, device='cuda:0')
Epoch [6/8], Step [500/2000], Train Accuracy: 1.756 , Train Loss: 3.7292
Epoch [6/8], Step [1000/2000], Train Accuracy: 1.780 , Train Loss: 3.6724
Epoch [6/8], Step [1500/2000], Train Accuracy: 1.819 , Train Loss: 3.6233
Epoch [6/8], Step [2000/2000], Train Accuracy: 1.871 , Train Loss: 3.7011
Epoch [6/8], Test Accuracy of the network: 2.09 %, Test Loss: 5.2187
------------------------------------------------------------------------------------------------------------
1.871 tensor(3.6552, device='cuda:0', grad_fn=<DivBackward0>) 2.09 tensor(5.2187, device='cuda:0')
Epoch [7/8], Step [500/2000], Train Accuracy: 2.180 , Train Loss: 3.6292
Epoch [7/8], Step [1000/2000], Train Accuracy: 2.266 , Train Loss: 3.6287
Epoch [7/8], Step [1500/2000], Train Accuracy: 2.360 , Train Loss: 3.5951
Epoch [7/8], Step [2000/2000], Train Accuracy: 2.411 , Train Loss: 3.6367
Epoch [7/8], Test Accuracy of the network: 2.4 %, Test Loss: 5.1966
------------------------------------------------------------------------------------------------------------
2.411 tensor(3.6394, device='cuda:0', grad_fn=<DivBackward0>) 2.4 tensor(5.1966, device='cuda:0')
Epoch [8/8], Step [500/2000], Train Accuracy: 2.852 , Train Loss: 3.5799
Epoch [8/8], Step [1000/2000], Train Accuracy: 2.956 , Train Loss: 3.6541
Epoch [8/8], Step [1500/2000], Train Accuracy: 3.020 , Train Loss: 3.6360
Epoch [8/8], Step [2000/2000], Train Accuracy: 3.068 , Train Loss: 3.7464
Epoch [8/8], Test Accuracy of the network: 2.95 %, Test Loss: 5.1625
------------------------------------------------------------------------------------------------------------
3.068 tensor(3.6153, device='cuda:0', grad_fn=<DivBackward0>) 2.95 tensor(5.1625, device='cuda:0')
Finished Training Student Network
```

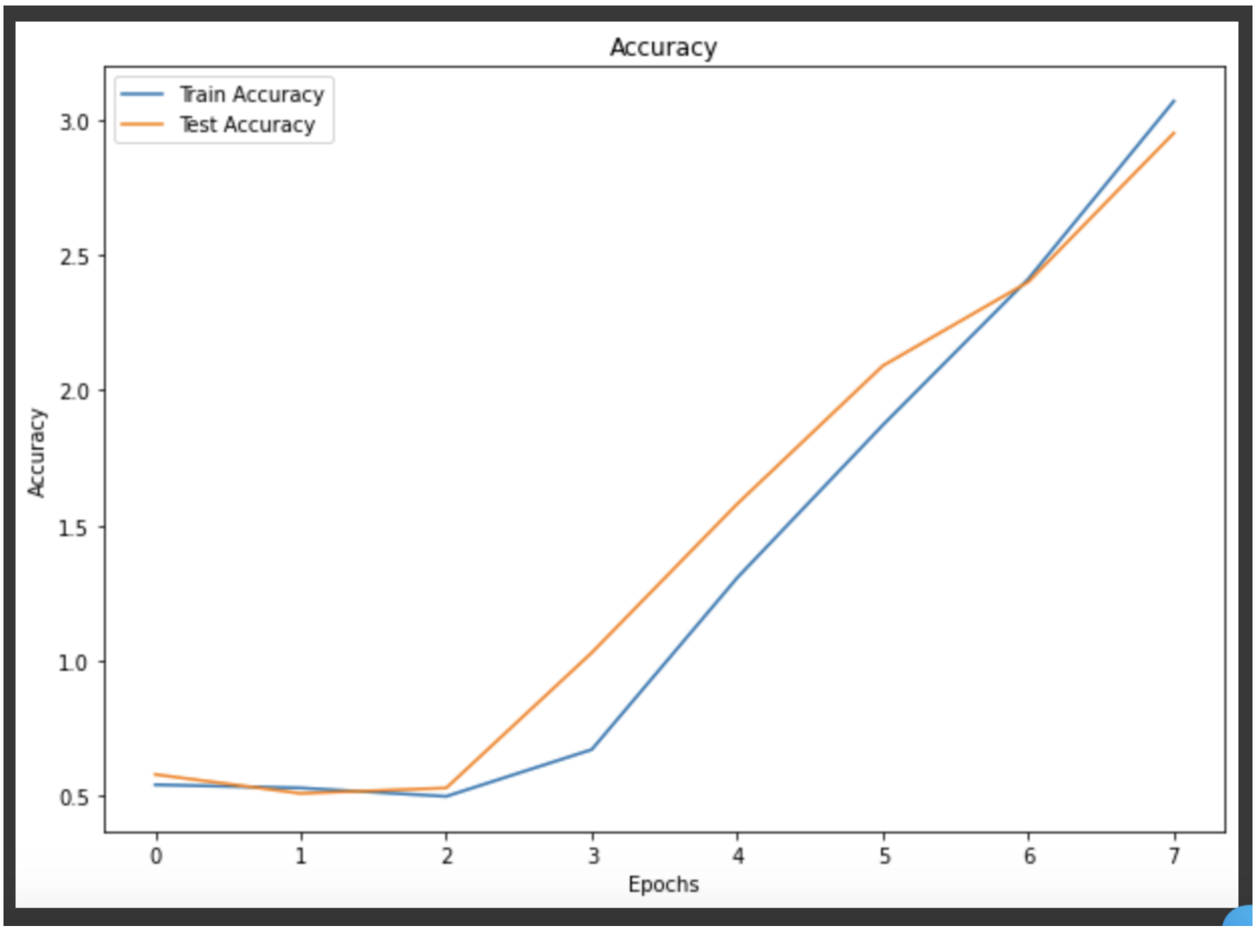
Loss Function:-

```
temperature = 3
def t_n_s_loss_fn(
    t_out,
    s_out,
    labels,
):
    cce_l = nn.CrossEntropyLoss()(s_out, labels.view(-1))
    s_out = nn.LogSoftmax()(s_out/temperature)
```

```
t_out = torch.softmax(t_out/temperature, dim=-1)
kde_loss = nn.KLDivLoss()(s_out, t_out)

return kde_loss*(0.3*temperature*temperature) + cce_l*0.7
```

6. Testing results on the Student network.



Observation:-

The accuracy is better in the Student network as compared to the Teacher network.

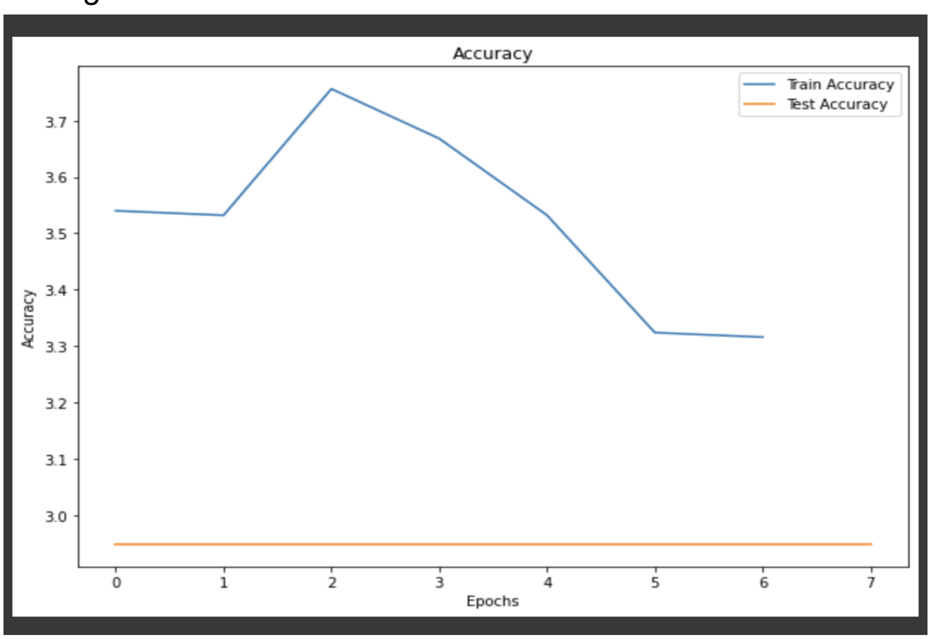
[Colab Link Question 1]

Network with Exponential Mean Average:-

1. By default, torch.optim.swa_utils.AveragedModel computes a running equal average of the parameters provided.
2. One can also use custom averaging functions with the avg_fn parameter. Here, ema_model computes an exponential moving average.
3. Performed training on the CNN model with 8 epochs. Below are the training results.

```
Epoch [4/8], Step [2000/2000], Train Accuracy: 3.439 , Train Loss: 3.7085
Epoch [4/8], Test Accuracy of the network: 2.95 %, Test Loss: 5.1625
------------------------------------------------------------------------------------------------------------------------------------
3.439 tensor(3.7088, device='cuda:0', grad_fn=<DivBackward0>) 2.95 tensor(5.1625, device='cuda:0')
Epoch [5/8], Step [500/2000], Train Accuracy: 3.532 , Train Loss: 3.7088
Epoch [5/8], Step [1000/2000], Train Accuracy: 3.458 , Train Loss: 3.7126
Epoch [5/8], Step [1500/2000], Train Accuracy: 3.428 , Train Loss: 3.7095
Epoch [5/8], Step [2000/2000], Train Accuracy: 3.439 , Train Loss: 3.7100
Epoch [5/8], Test Accuracy of the network: 2.95 %, Test Loss: 5.1625
------------------------------------------------------------------------------------------------------------------------------------
3.439 tensor(3.7088, device='cuda:0', grad_fn=<DivBackward0>) 2.95 tensor(5.1625, device='cuda:0')
Epoch [6/8], Step [500/2000], Train Accuracy: 3.436 , Train Loss: 3.7067
Epoch [6/8], Step [1000/2000], Train Accuracy: 3.544 , Train Loss: 3.7116
Epoch [6/8], Step [1500/2000], Train Accuracy: 3.499 , Train Loss: 3.7088
Epoch [6/8], Step [2000/2000], Train Accuracy: 3.439 , Train Loss: 3.7133
Epoch [6/8], Test Accuracy of the network: 2.95 %, Test Loss: 5.1625
------------------------------------------------------------------------------------------------------------------------------------
3.439 tensor(3.7088, device='cuda:0', grad_fn=<DivBackward0>) 2.95 tensor(5.1625, device='cuda:0')
Epoch [7/8], Step [500/2000], Train Accuracy: 3.324 , Train Loss: 3.7083
Epoch [7/8], Step [1000/2000], Train Accuracy: 3.352 , Train Loss: 3.7040
Epoch [7/8], Step [1500/2000], Train Accuracy: 3.391 , Train Loss: 3.7090
Epoch [7/8], Step [2000/2000], Train Accuracy: 3.439 , Train Loss: 3.7115
Epoch [7/8], Test Accuracy of the network: 2.95 %, Test Loss: 5.1625
------------------------------------------------------------------------------------------------------------------------------------
3.439 tensor(3.7088, device='cuda:0', grad_fn=<DivBackward0>) 2.95 tensor(5.1625, device='cuda:0')
Epoch [8/8], Step [500/2000], Train Accuracy: 3.316 , Train Loss: 3.7047
Epoch [8/8], Step [1000/2000], Train Accuracy: 3.396 , Train Loss: 3.7086
Epoch [8/8], Step [1500/2000], Train Accuracy: 3.459 , Train Loss: 3.7114
Epoch [8/8], Step [2000/2000], Train Accuracy: 3.439 , Train Loss: 3.7084
Epoch [8/8], Test Accuracy of the network: 2.95 %, Test Loss: 5.1625
------------------------------------------------------------------------------------------------------------------------------------
3.439 tensor(3.7088, device='cuda:0', grad_fn=<DivBackward0>) 2.95 tensor(5.1625, device='cuda:0')
```

4. Testing results on the Student network with EMA.



Question 2:

Problem (i) :-

Suppose you have a neural network with one input layer, one hidden layer with 10 neurons, and one output layer with one neuron. The input layer has 50 features, and the output layer produces a continuous value between 0 and 1. The weights of the network are initialized randomly using a normal distribution with mean 0 and standard deviation 0.1.

You have a dataset with 1000 samples, and you want to train the network using stochastic gradient descent with a batch size of 10, a learning rate of 0.01, and the mean squared error loss function. After running the training algorithm for 100 epochs, the mean squared error on the validation set is 0.05.

(a) What is the total number of weights in the network?
(b) How many batches are there in each epoch?
(c) What is the size of the weight update for a given batch?
(d) What is the expected value of the weight update for the entire dataset?
(e) What is the expected value of the mean squared error on the training set after 100 epochs?
(f) How would you modify the learning rate and batch size to improve the convergence of the algorithm?

Solution from ChatGPT :-

(a) The total number of weights in the network is calculated as follows:

Number of weights between the input layer and the hidden layer: (50 * 10) = 500
Number of bias weights for the hidden layer: 10
Number of weights between the hidden layer and the output layer: (10 * 1) = 10
Number of bias weights for the output layer: 1
Therefore, the total number of weights in the network is 521.

(b) The number of batches in each epoch is calculated as follows:

Total number of samples in the dataset: 1000
Batch size: 10
Therefore, the number of batches in each epoch is 100.

(c) The size of the weight update for a given batch is calculated using the following formula:

Weight update = learning rate * (gradient of the loss function with respect to the weights)
The gradient of the loss function with respect to the weights is computed by backpropagating the errors from the output layer to the hidden layer and then to the input layer. The size of the weight update depends on the values of the gradient and the learning rate.

(d) The expected value of the weight update for the entire dataset is difficult to calculate since it depends on the values of the gradients and the learning rate at each iteration. However, the weight update for each batch is calculated as above, and the expected value of the weight update for the entire dataset can be approximated by taking the average of the weight updates across all the batches.

(e) The expected value of the mean squared error on the training set after 100 epochs is not given in the question, and it is difficult to calculate analytically. However, we know that the mean squared error on the validation set after 100 epochs is 0.05, which indicates that the network has achieved good generalization.

(f) To improve the convergence of the algorithm, we could try the following modifications:

Increase the learning rate: A larger learning rate can help the network converge faster, but it may also cause the algorithm to overshoot the optimal weights and converge to a suboptimal solution.
Decrease the batch size: A smaller batch size can help the network converge faster by providing more frequent weight updates, but it may also increase the noise in the gradient estimates and make the training process less stable.
Use a different optimization algorithm: Stochastic gradient descent is a simple optimization algorithm that works well in many cases, but there are other optimization algorithms such as Adam, RMSprop, and Adagrad that can be more effective for certain types of networks and datasets.

The procedure and values calculated by ChatGPT are correct.

Problem 2: Can we use MSE loss for binary class classification?

Solution from ChatGPT

Yes, you can use Mean Squared Error (MSE) loss function for binary classification problems, but it is not the most common loss function used for this type of problem.

MSE loss function tries to minimize the difference between predicted and actual values by squaring the difference between them. In binary classification problems, the target variable is binary, meaning it can take only two values (0 or 1). Therefore, the output of the neural network should also be a single value between 0 and 1 that represents the predicted probability of the positive class.

In such cases, the sigmoid activation function is used in the output layer to squash the output of the neural network between 0 and 1. Then, the MSE loss function can be used to calculate the difference between predicted and actual probabilities.

However, the more common loss function used for binary classification problems is Binary Cross-Entropy (BCE) loss function. BCE loss function is defined as:

BCE = -[y*log(y_hat) + (1-y)*log(1-y_hat)]

where y is the actual label (0 or 1) and y_hat is the predicted probability of the positive class.

BCE loss function is preferred over MSE loss function for binary classification problems because it penalizes the model more strongly for making confident incorrect predictions.

Real World Observation :-
Mean Squared Error (MSE) is a poor option for binary classification issues for two reasons:

By using MSE, we presume that the underlying data is in the form of normal distribution (a bell-shaped curve). This implies that we take a Gaussian prior, in Bayesian words. A dataset that can be divided into two categories (i.e., binary) is not, however, actually from a normal distribution.
If we employ maximum likelihood estimation (MLE), making the incorrect assumption that the data comes from a normal distribution, we obtain the MSE as a cost function for refining our model.

Second, for binary classification, the MSE function is non-convex. Simply said, if a binary classification model is trained with the MSE Cost function, the Cost function will not necessarily be minimized. This is because binary classification models employ the sigmoid/logistic function to output probabilities in the range (0,1), whereas MSE function needs real-valued inputs in the ange(-∞, ∞)

Problem 3 : Can we use a hybrid activation function?

Solution from ChatGPT

Yes, it is possible to hybridize the activation functions in a neural network, and this is a common practice in many advanced neural network architectures.

Activation functions are used to introduce non-linearity into a neural network, which is important for modeling complex relationships between input and output data. There are several activation functions available, such as ReLU, sigmoid, tanh, and softmax, each with their own strengths and weaknesses.

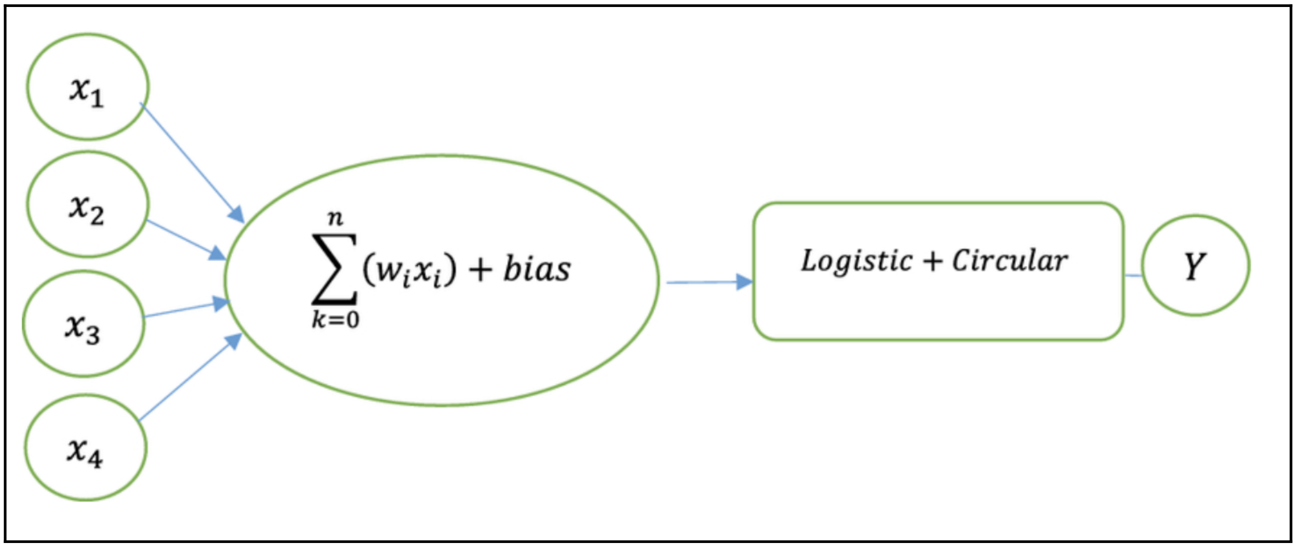
Hybrid activation functions combine two or more activation functions to take advantage of their complementary properties. For example, the Rectified Linear Unit (ReLU) activation function is good at handling the vanishing gradient problem, but it can cause neurons to die if the output is consistently negative. In contrast, the Exponential Linear Unit (ELU) activation function is a smooth function that avoids the dying neuron problem, but it is more computationally expensive than ReLU.

To combine the advantages of ReLU and ELU, a hybrid activation function called the Scaled Exponential Linear Unit (SELU) was introduced. SELU uses the ELU activation function for negative values and a scaled version of the ReLU activation function for positive values. This hybrid function has been shown to improve the performance of neural networks, especially for deep architectures.

Another example of hybrid activation function is the Gated Linear Unit (GLU), which combines the strengths of sigmoid and ReLU activation functions. GLU uses the sigmoid activation function to control the gating mechanism and the ReLU activation function to introduce non-linearity.

Overall, hybrid activation functions can be a powerful tool for improving the performance of neural networks. However, it is important to carefully design and test these functions to ensure that they do not introduce unintended side-effects or computational overhead.

Real world answer:-



The adaptive hybrid activation function (AHAF) is proposed that com-bines the properties of the rectifier units and the squashing functions. The proposed function can be used as a drop-in replacement for ReLU, SiL and Swish activations for deep neural networks and can evolve to one of such functions during the training.
The hybrid acti-vation functions were introduced that combine the properties of both the rectifiers and sigmoid functions. The list of hybrid functions includes Swish, S-shaped and other similar functions.
Performance of the adaptive hybrid activation function was evaluated on the image classification task on two different datasets with two base neural network architectures in a similar way. The base architectures were modified to use AHAF activations instead of “classic” activations like ReLU and SiL. The performance of the modified networks was compared to the reference implementations.

References:-

https://pytorch.org/docs/stable/optim.html?highlight=exponentially+weighted+average
http://journal.iasa.kpi.ua/article/view/259203/255848
https://towardsdatascience.com/why-using-mean-squared-error-mse-cost-function-for-binary-classification-is-a-bad-idea-933089e90df7
https://medium.com/analytics-vidhya/knowledge-distillation-in-a-deep-neural-network-c9dd59aff89b
https://towardsdatascience.com/pytorch-ignite-classifying-tiny-imagenet-with-efficientnet-e5b1768e5e8f
https://www.researchgate.net/figure/Diagram-of-neural-network-with-hybrid-activation-function_fig5_358162575