PART A - TensorFlow and the Low Level API [50 Marks]

Question1_1: Application of Basic SoftMax layer model to Fashion MNIST dataset

Description of the three functions in our TensorFlow program

1) **forward_pass:**
    - This function takes in the training feature matrix which contains all the training examples, weight matrix which contains the weights incoming to the SoftMax layer and the bias matrix that contain the bias for each neuron of the SoftMax layer.
    - This function returns the matrix of probabilities for each of the training instances.
    - First we calculate initial output matrix where each column represents the pre-activation outputs of each neuron in the SoftMax layer for each of the training instance facilitated by the matrix multiplication of train feature matrix and the weights matrix followed by the addition of the bias matrix to the product.
    - Then we converted our pre-activation output matrix such that each element now becomes e to the power of element producing a new exponential matrix.
    - Further we calculated the column wise sum so that we can convert each element in every column of exponential matrix to the corresponding probabilities. This produced the column sums equal to the number of training instances.
    - Then we divided each element of a particular column by its respective column sum so that its value gets converted between 0 and 1. This way we get the probability matrix where each column represents the probability of each of the 10 classes of a single training instance in our feature training data.
2) **cross_entropy:**
    - This function takes in the one hot encoded true class labels matrix of the training set and the probability matrix obtained from the forward pass function.
    - This function returns the mean loss value that exists across each of the true class labels and the predicted class labels.
    - The formula to calculate the loss for a single training instance is given below

$$L(p^i, y^i) = -\sum_{j=1}^{c} y_j^i \log(p_j^i)$$

- We then calculated the natural log of each element in the predicted probabilities matrix producing the log matrix.
- After then we multiplied each element of particular column in log matrix with the corresponding element of the column in actual class labels matrix producing the product matrix.
- Then we performed the negation of each element in the product matrix producing the negated product matrix.
- To get the loss for each of the training instances, we performed the column wise sum on the negated product matrix which satisfies the formula above for the loss calculation.
- The formula for the cost function of the SoftMax model is below

$$C = \frac{1}{m}\sum_{i=1}^{m} L(p^i, y^i)$$

- In order to calculate the mean loss value we calculated the mean of the individual losses of all the training instances satisfying the above formula.

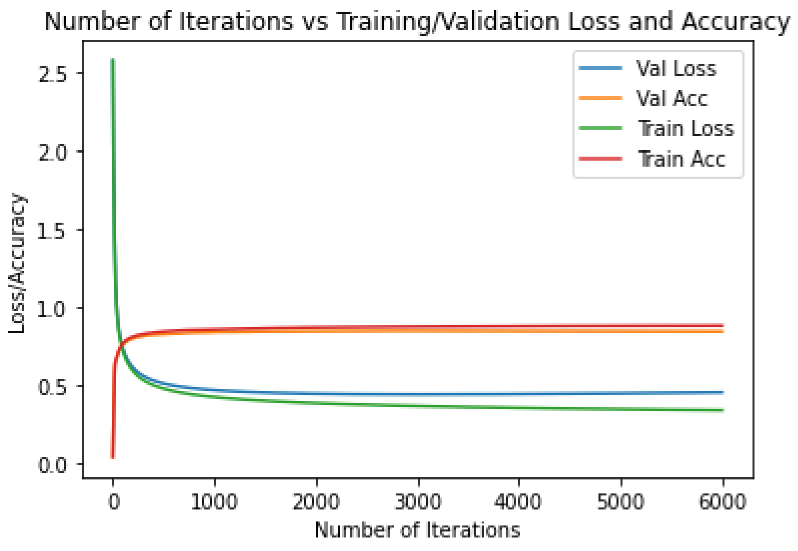
3) calculate_accuracy:

- This function takes in the feature matrix, label matrix, weight matrix and the bias matrix as the arguments.
- This function returns the accuracy value (percentage of correctly predicted instances) with each iteration applicable for training and the test feature data.
- First we calculate the predicted probabilities matrix with the help of forward pass function for the feature matrix.
- Then we compute the predicted and the actual labels of each of training instance by using the argmax function of TensorFlow on predicted matrix and actual true label matrix.
- Then we compared these 2 tensors of predicted and the actual labels which returned us a Boolean tensor where 1 represents that the predicted value is correct and 0 represents that it is incorrect.
- Then to obtain the accuracy we computed the mean of this Boolean tensor.

Evaluation graph of Basic SoftMax Model

We trained our model for 6500 iterations on the **Fashion MNIST dataset** with only one SoftMax layer containing the 10 neurons. The final training/validation loss and accuracy are in the table below which are recorded after 6000 iterations.

Training Loss	**Training Accuracy**	**Validation Loss**	**Validation Accuracy**
0.33955416	0.8804333	0.4536483	0.8433



```
The final test accuracy of the Fashion MNIST dataset is 0.843500018119812
```

Observation of the above graph

- Here we have considered test set only as our validation set, so below wherever we use the term validation, we mean that computation is carried out on test set only.
- Clearly our basic SoftMax model is performing well on the test set achieving the test set accuracy of 84.35 %.
- Training accuracy curve shows up a very small increase with the increase in the number of iterations and rises up very slowly from 81% at 200 iterations to 88% at iterations 6000. On the other hand, the validation accuracy flattens out from 1100 iterations averaging around 84%.
- The wider the gap between the train and the validation loss, the greater the degree of the overfitting on the training data.
- We can clearly see that the network is overfitting on the training data but not too much aggressively and begins to overfit from iteration 500 because of the very small and the continuous increase in the divergence/gap between the training loss and the validation loss.
- So, we can estimate our right/optimal number of iterations to be around 500 with the basic SoftMax model on this Fashion MNIST dataset.

Question1_2_1: Application of Network Architecture A to Fashion MNIST dataset

- This is a 2 layer architecture in which the hidden layer is ReLU with 300 neurons and the output layer is SoftMax with 10 neurons.
- We defined the **forward_pass** function according to the below steps

$$a^{[1]} = w^{[1]}x + b^{[1]}$$

$$h^{[1]} = act(a^{[1]})$$

$$a^{[2]} = w^{[2]} h^{[1]} + b^{[2]}$$

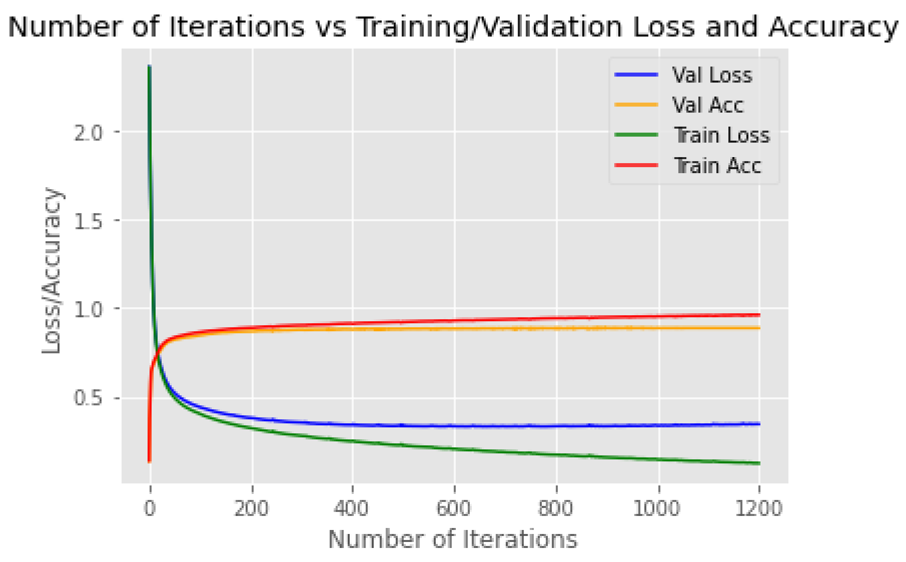
$$h^{[2]} = act(a^{[2]})$$

- This function takes in the feature matrix along with the weight matrix 1 and 2 with the bias matrix 1 and 2.
- This function returns the matrix of probabilities for each of the training instances.
- We then calculated matrix A1 containing the 300 outputs from the 300 neurons of the hidden layer for each of the training instance produced by the dot product of weight matrix W1 and the training feature data X followed by the addition of bias matrix b1 to this product.
- Then we passed the each element of the A1 matrix to the ReLU activation function such that elements less than 0 gets converted to 0 and the elements greater than or equal to 0 retains the same value. We get this matrix H1 as the output.
- Then we calculated matrix A2 produced by the dot product of H1 (outputs from the hidden layer) and W2 (weights of each of neurons incoming to the SoftMax layer) followed by the addition of bias matrix b2 to this product. Each column of this matrix represents the pre-activation outputs of each neuron in the SoftMax layer for each of the training instance.
- Then we converted our pre-activation output matrix A2 such that each element now becomes e to the power of element producing a new exponential matrix.
- Further we calculated the column wise sum so that we can convert each element in every column of exponential matrix to the corresponding probabilities. This produced the column sums equal to the number of training instances.
- Then we divided each element of a particular column by its respective column sum so that its value gets converted between 0 and 1. This way we get the probability matrix H2 where each column represents the probability of each of the 10 classes of a single training instance in our feature training data.

Evaluation graph of Network Architecture A

We trained our model for 1200 iterations on the **Fashion MNIST dataset** with two layers; first one is ReLU activation layer with 300 neurons and another one is SoftMax layer containing the 10 neurons. The final training/validation loss and accuracy are in the table below which are recorded after 1200 iterations.

Training Loss	**Training Accuracy**	**Validation Loss**	**Validation Accuracy**
0.120933935	0.9623167	0.3430179	0.8877



```
The final test accuracy of the Fashion MNIST dataset is 0.8876000046730042
```

Observation of the above graph

- Here we have considered test set only as our validation set, so below wherever we use the term validation, we mean that computation is carried out on test set only.
- Clearly our Network Architecture A is performing well on the test set achieving the test set accuracy of 88.76 %.
- Training accuracy curve shows up a very small increase with the increase in the number of iterations and rises up very slowly from 93% at 600 iterations to 96% at iterations 1200. On the other hand, the validation accuracy flattens out from 340 iterations averaging around 88%.
- We can clearly see that the network began overfitting on the training data from 200 iterations because of the continuous increasing gap between the training loss and the validation loss.
- So, we can estimate our right/optimal number of iterations to be around 200 with the Network Architecture A on this Fashion MNIST dataset.

Question1_2_2: Application of Network Architecture B to Fashion MNIST dataset

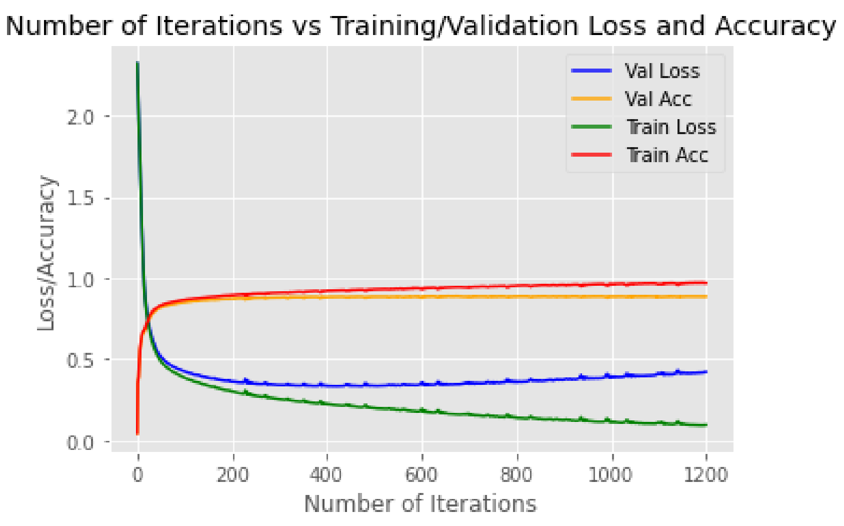
- This is a 3 layer architecture in which the hidden layer 1 is ReLU with 300 neurons, hidden layer 2 is also ReLU with 100 neurons and the output layer is SoftMax with 10 neurons.
- We defined the **forward_pass** function according to the below steps which takes in the feature matrix, weight matrices W1, W2 and W3 along with the bias matrices b1, b2 and b3 and returns the matrix of probabilities for each of the training instances.
- We performed each step in the same way carried out in the network architecture A.

$$A^{[1]} = W^{[1]} . X + b^{[1]}$$
$$H^{[1]} = act(A^{[1]})$$
$$A^{[2]} = W^{[2]} . H^{[1]} + b^{[2]}$$
$$H^{[2]} = act(A^{[2]})$$
$$A^{[3]} = W^{[3]} . H^{[2]} + b^{[3]}$$
$$H^{[3]} = act(A^{[3]})$$

Evaluation graph of Network Architecture B

We trained our model for 1200 iterations on the **Fashion MNIST dataset** with three layers; first one is ReLU activation layer with 300 neurons, second is also ReLU with 100 neurons and last one is SoftMax layer containing the 10 neurons. The final training/validation loss and accuracy are in the table below which are recorded after 1200 iterations.

Training Loss	**Training Accuracy**	**Validation Loss**	**Validation Accuracy**
0.09595539	0.9691667	0.4222676	0.8858



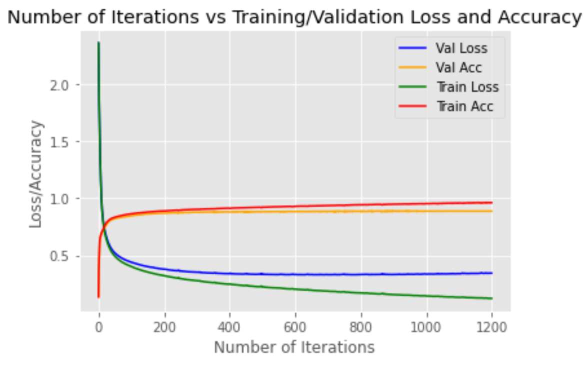
```
The final test accuracy of the Fashion MNIST dataset is 0.8844000101089478
```

Observation of the above graph

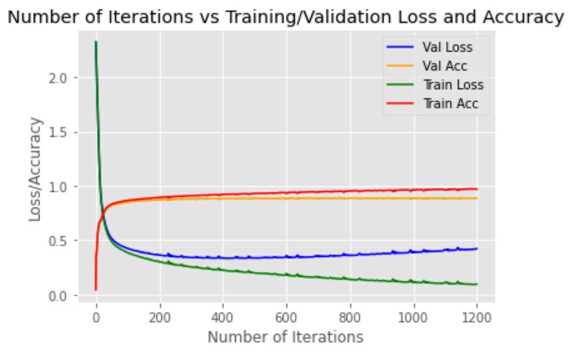
- Here we have considered test set only as our validation set, so below wherever we use the term validation, we mean that computation is carried out on test set only.
- Clearly our Network Architecture B is performing well on the test set achieving the test set accuracy of 88.44 %.
- Training accuracy curve shows up a very small increase with the increase in the number of iterations and rises up very slowly from 94% at 600 iterations to 97% at iterations 1200. On the other hand, the validation accuracy flattens out from 320 iterations averaging around 88%.
- We can clearly see that the network began overfitting on the training data from 200 iterations because of the continuous increasing gap between the training loss and the validation loss.
- So, we can estimate our right/optimal number of iterations to be around 200 with the Network Architecture B on this Fashion MNIST dataset.

Sensitivity to Number of Neurons in Network Architecture A and B

- In the network architecture A, there are 300 neurons in the hidden layer and in network architecture B, there are 400 neurons combined in both the hidden layers.
- We then compare both the architectures A and B having less and more number of neurons overall respectively in terms of performance and overfitting levels.
- Graph on the left is the evaluation graph of the network architecture A and one on the right is the graph of network architecture B.



```
The final test accuracy of the Fashion MNIST dataset is 0.8876000046730042
```



```
The final test accuracy of the Fashion MNIST dataset is 0.8844000101089478
```

Conclusion Analysis of the above 2 graphs

- Both the architectures A and B have similar level of test set accuracy around 88%.
- Final training accuracy of both the architectures A and B also have similar values around 96%.
- Both the architectures began overfitting on the training data from 200 iterations, but the architecture A which has less number of neurons in the hidden layer have less overfitting level than the architecture B which has more number of neurons. This is

because the validation loss in the architecture A tends to become constant, but in architecture B, it tends to increase, creating a greater amount of gap/divergence between the training and the validation loss.

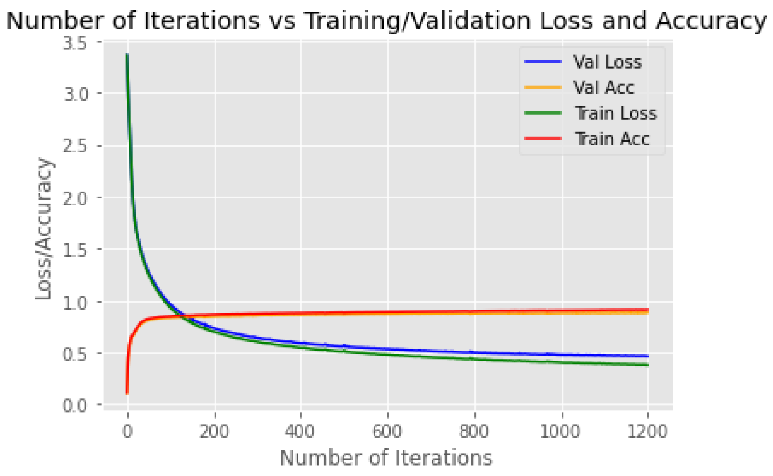
- So to **conclude**, we can say that as evident from both of these architectures increasing the number of neurons in the network architecture does not make any significant improvement in the validation (test) set and the training set accuracy. But the one with the less number of neurons has slightly less level of overfitting than the one which has more number of neurons, so our best choice in this scenario is to consider network architecture A over network architecture B.

Question1_3: Application of L1 and L2 regularisation for the network architecture B

- The main idea behind regularisation is to reduce the magnitude of the weights so that it prevents overfitting that may be caused because of the increase in the magnitude of the weights when we leave our model to train for large number of iterations.
- The main idea is to add an additional component to our loss function such that gradient descent will not only try to reduce the difference between the actual and the predicted values, but also the magnitude of the weights.
- In the formulas below, the sigma is the constant called regularisation rate and controls the amount of shrinkage in the weights. The larger this constant is, the greater is the amount of shrinkage in the weights.

Application of L1 regularisation to Network Architecture B

- We have our cost calculated through the **cross_entropy** function.
- We then calculated the additional component by multiplying the regularisation rate of 0.0001 to the sum of all the elements of the weight matrices W1, W2 and W3.
- We then added this additional component to the cost value to get the final loss.
- Below is the **evaluation graph** of the L1 regularisation applied on the network architecture B trained for 1200 iterations.



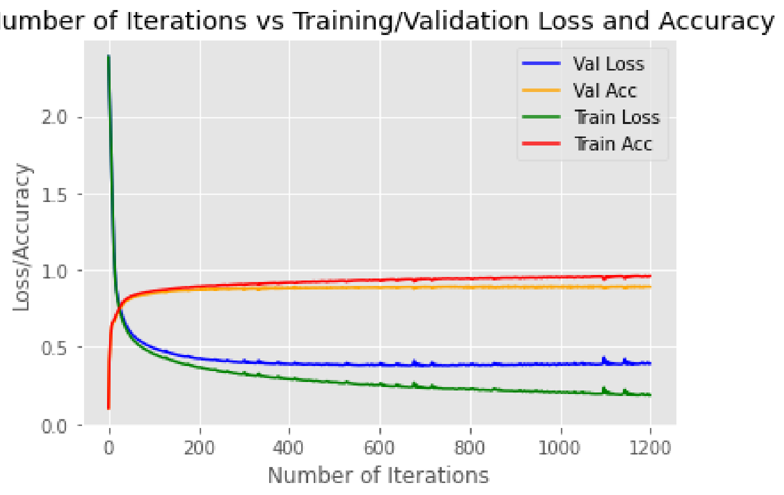
```
The final test accuracy of the Fashion MNIST dataset is 0.8805999755859375
```

Observation of the above graph

- We observed that L1 regularisation produced the test set accuracy of about 88%.
- Training accuracy cure shows up a very small increase and rises up very slowly from 88% from 412 iterations to 91% at 1200 iterations. On the other hand the validation accuracy flattens out from 600 iterations averaging around 88%.
- We can clearly see that the network began overfitting on the training data on a very small scale from epoch 650 as the gap between the training and the validation loss is increasing but at a much much smaller rate.
- So we can say that the optimal number of iterations to train our model with L1 regularisation could be around 650 or 700.

Application of L2 regularisation to Network Architecture B

- We have our cost calculated through the **cross_entropy** function.
- We then calculated the additional component by multiplying the regularisation rate of 0.0001 to the sum of all the squared elements of the weight matrices W1, W2 and W3.
- We then added this additional component to the cost value to get the final loss.
- Below is the **evaluation graph** of the L2 regularisation applied on the network architecture B trained for 1200 iterations.



```
The final test accuracy of the Fashion MNIST dataset is 0.8932999968528748
```

Observation of the above graph

- We observed that L2 regularisation produced the test set accuracy of about 89%.
- Training accuracy cure shows up a very small increase and rises up very slowly from 93% from 530 iterations to 96% at 1200 iterations. On the other hand the validation accuracy flattens out from 400 iterations averaging around 89%.

- We can clearly see that the network began overfitting on the training data from 200 iterations because of the continuous increasing gap between the training loss and the validation loss.
- So we can say that the optimal number of iterations to train our model with L2 regularisation could be around 200 or 300.

Comparative analysis of L1 architecture, L2 architecture with Initial Network Architecture B.

- Both the L1 and L2 regularisation produced same level of test set accuracy around 88% with 1200 iterations.
- Final training set accuracy of L1 is 91% and that of L2 comes out to be 96%. In both the techniques, the validation set accuracy flattens from around 500 iterations at 89%.
- It is clearly visible that when applied to network architecture B, L1 regularisation reduced the overfitting on the training data at much greater level than the L2 regularisation.
- By this we can conclude that, for the network architecture B, L1 is the effective technique to reduce overfitting levels when compared with L2 regularisation.

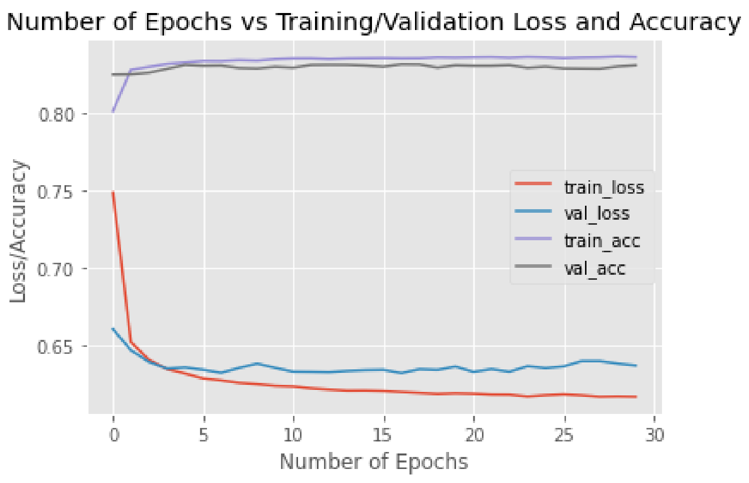
PART B - Keras – High Level API [30 Marks]

Part B – Question 1 – Development and Analysis of 4 different SoftMax Architectures.

Architecture 1- Evaluation of SoftMax Model

We trained our model for 30 epochs on the **letter dataset** with only one SoftMax layer containing the 10 neurons. The final training/validation loss and accuracy are in the table below which are recorded after 30 epochs.

Training Loss	Training Accuracy	Validation Loss	Validation Accuracy
0.6165	0.8367	0.6367	0.8314



```
532/532 [==============================] - 1s 2ms/step - loss: 0.5489 - accuracy: 0.8552
The test set loss and accuracy is
 [0.5488766431808472, 0.8551764488220215]
```

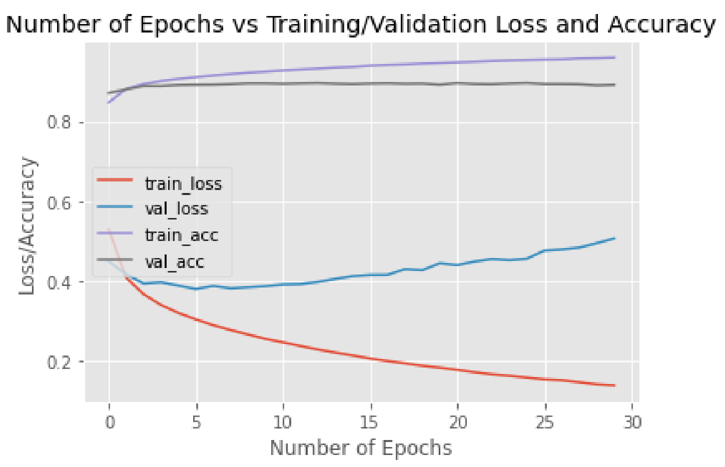
Observation of the above graph

- Clearly our basic SoftMax model is performing well on the test set achieving the test set accuracy of 85.5 %.
- Training accuracy begins to flatten out at approx. 83% from the epochs 3 and there is also very small increase in the validation accuracy from epoch 5 averaging around 83%.
- We can clearly see that the network is overfitting on the training data but not too much aggressively and begins to overfit as early as epoch 5 or 6 because of the little and continuous increase in the divergence/gap between the training loss and the validation loss.
- So, we can estimate our right/optimal number of epochs to be around 5 with the basic SoftMax model on this letter dataset.

Architecture 2- Evaluation of 2 layer SoftMax Model

We trained our model for 30 epochs on the **letter dataset** with 1st layer as ReLU activation layer containing 200 neurons and 2nd layer as SoftMax layer containing the 10 neurons. The final training/validation loss and accuracy are in the table below which are recorded after 30 epochs.

Training Loss	Training Accuracy	Validation Loss	Validation Accuracy
0.1401	0.9590	0.5070	0.8907



```
532/532 [==============================] - 1s 2ms/step - loss: 0.4071 - accuracy: 0.9120
The test set loss and accuracy is
 [0.40713396668434143, 0.9120000004768372]
```

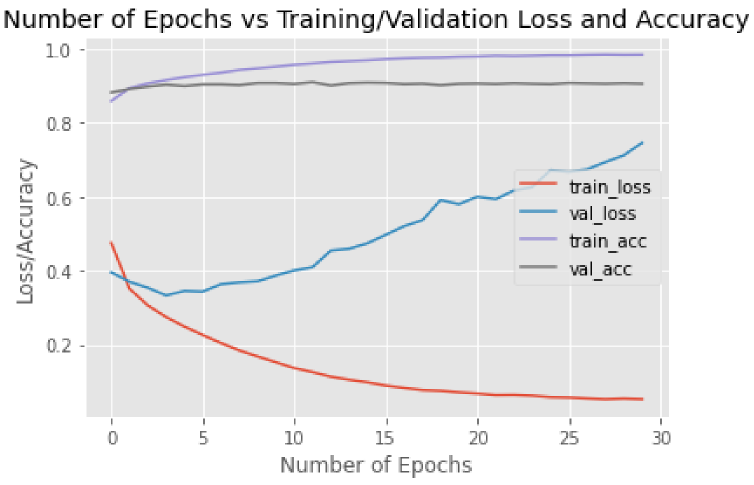
Observation of the above graph and comparison with basic SoftMax model

- Clearly in terms of test set accuracy, this 2 layer model performed better than the basic SoftMax model achieving an accuracy of 91.2% when compared with 85.5%.
- Training curve shows up a little increase in the accuracy levels from 90% at epoch 5 to 95% at epoch 30. On the other hand, validation accuracy tends to flatten out from epoch 5 averaging around 89%.
- We can clearly see that the network begins to overfit aggressively on the training data from the epoch 4 or 5 because of the increasing divergence/gap between the training loss and the validation loss at a much greater rate leading to a wider and wider gap between the two curves.
- Overall if we compare this 2 layer architecture with the basic SoftMax one, then this one has a good level of performance but this also at the same time overfits on the training data more aggressively when compared with the basic architecture.
- So for this architecture also we can estimate our right/ optimal number of epochs to around 5.

Architecture 3- Evaluation of 3 layer SoftMax Model

We trained our model for 30 epochs on the **letter dataset** with 1st layer as ReLU activation layer containing 400 neurons, 2nd ReLU layer with 200 neurons and 3rd layer as SoftMax layer containing the 10 neurons. The final training/validation loss and accuracy are in the table below which are recorded after 30 epochs.

Training Loss	Training Accuracy	Validation Loss	Validation Accuracy
0.0522	0.9838	0.7456	0.9050



```
532/532 [==============================] - 1s 2ms/step - loss: 0.6061 - accuracy: 0.9211
The test set loss and accuracy is
 [0.6061061024665833, 0.9211176633834839]
```

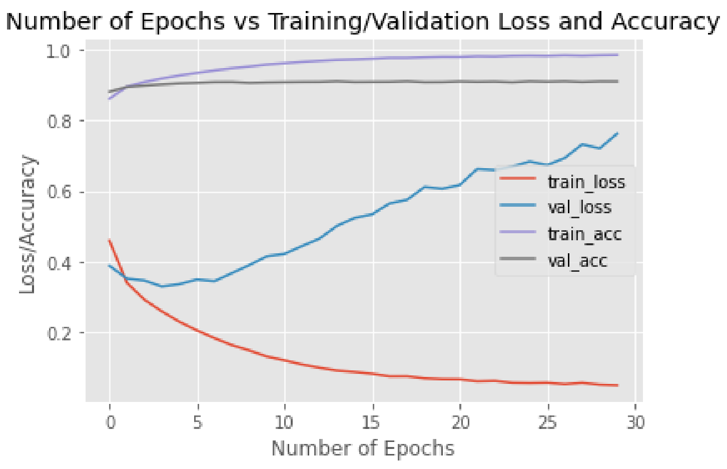
Observation of the above graph

- The model achieves a good test set accuracy of 92.11%.
- The training curve shows a gradual increase in the accuracy starting at 93% from epoch 7 and achieving 98% up till epoch 30. On the other hand, validation accuracy tends to flatten out from epoch 6 averaging around 90%.
- We can clearly see that the network begins to overfit aggressively on the training data from the epoch 3 or 4 because of the increasing divergence/gap between the training loss and the validation loss at a much greater rate leading to a wider and wider gap between the two curves.
- Therefore for this architecture, we can determine that the optimal/right number of epochs comes out to be around 4.

Architecture 4- Evaluation of 4 layer SoftMax Model

We trained our model for 30 epochs on the **letter dataset** with 1st layer as ReLU activation layer containing 600 neurons, 2nd ReLU layer with 400 neurons, 3rd ReLU layer with 200 neurons and 4th layer as SoftMax layer containing the 10 neurons. The final training/validation loss and accuracy are in the table below which are recorded after 30 epochs.

Training Loss	**Training Accuracy**	**Validation Loss**	**Validation Accuracy**
0.0501	0.9843	0.7621	0.9097



```
532/532 [==============================] - 1s 2ms/step - loss: 0.6006 - accuracy: 0.9278
The test set loss and accuracy is
 [0.6006399989128113, 0.9277647137641907]
```

Observation of the above graph

- The model achieves a good test set accuracy of 92.77%.
- The training curve shows a gradual increase in the accuracy starting at 92% from epoch 5 and achieving 98% up till epoch 30. On the other hand, validation accuracy tends to flatten out from epoch 4 averaging around 91%.
- We can clearly see that the network begins to overfit aggressively on the training data from the epoch around 4 because of the increasing divergence/gap between the training loss and the validation loss at a much greater rate leading to a wider and wider gap between the two curves.
- Therefore for this architecture, we can determine that the optimal/right number of epochs comes out to be around 4.

Comparative Analysis of Architecture 2, 3, 4

Now we will be performing the comparative analysis of the SoftMax architecture having 2, 3 and 4 layers as mentioned in the specification. This will lead to the examination of the deeper neural network to our letter classification problem.

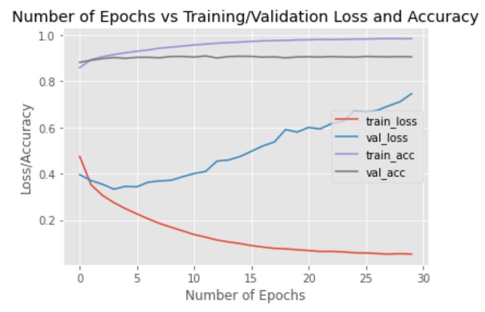
- When we compare the architectures having 2, 3 and 4 layers in the terms of test set accuracy, then all the models have the same level of accuracies around 92% showing that there is no significance difference on the performance by increasing the number of layers.
- Training accuracy in all the three models reaches around 92% at the point where the overfitting begins to occur on all these three architectures. On the other hand the validation accuracies show the constant behaviour achieving around 90% accuracy.
- It is clearly displayed that as we increase the number of layers in our architecture, the more and more aggressively overfitting occurs on the training data. In other words this means that increasing the number of layers make the gap between the training and the validation loss even more and more wider with the increase in the number of epochs.
- We can now generalize and make a fair conclusion for kind of selecting the best model for this letter classification problem.
- We can consider the architecture with 2 layers as increasing in the number of layers is not making the significant impact on the unseen test set performance. Moreover from these three model architectures, we can also observe that around epoch 5, the overfitting begins to occur so it is better to consider this **number=5** as an optimal training parameter.

Part B – Question 2 – Application and Analysis of Dropout Regularization to Architecture 3 and Architecture 4.

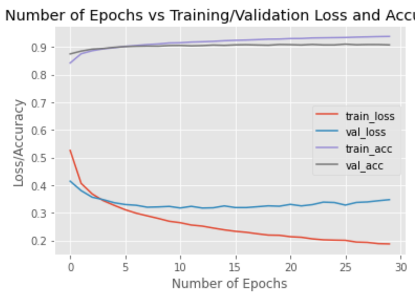
- We would be choosing architecture 3 and 4 for the application of dropout regularisation techniques as these architectures were more aggressively overfitting than the other ones.
- For each layer we are having the outputs from each neuron for each of our training examples which are passed through a Boolean filter.
- Given the dropout probability with each layer of Neural Network, this technique reduces some of the node outputs to 0 for that particular training instance.
- This is simply disabling the outputs from the first layer that will feed into the next layer of neurons.

Comparison of Normal Architecture 3 and Dropout Architecture 3

We applied the dropout with the probability of 0.2 after the layer 1 with 400 neurons and also after the layer 2 having 200 neurons. Below is the analysis of the evaluation graphs before and after the application of dropout layers to this particular architecture.



```
532/532 [==============================] - 1s 2ms/step - loss: 0.6061 - accuracy: 0.9211
The test set loss and accuracy is
 [0.6061061024665833, 0.9211176633834839]
```



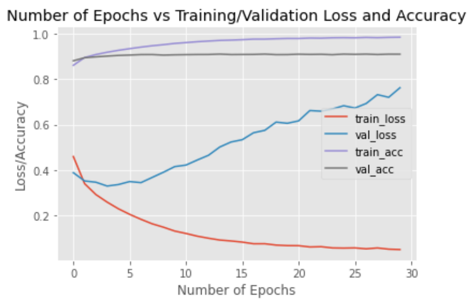
```
532/532 [==============================] - 1s 2ms/step - loss: 0.2724 - accuracy: 0.9272
The test set loss and accuracy is
 [0.2724141478538513, 0.9272353053092957]
```

Conclusion Analysis of the above 2 graphs

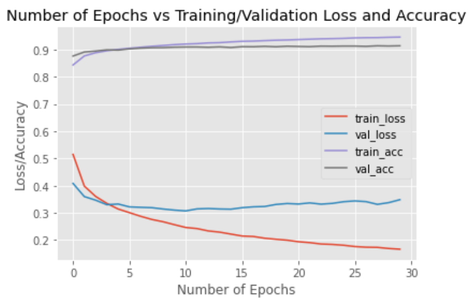
- There is not much improvement in the test set accuracy after the application of dropout regularisation to this architecture. Earlier without the dropout layers, the accuracy was 92.11% and then after these dropout layers it was reported only 92.72% (a minute increase).
- After the application of dropout, there is a decline in the final training accuracy from 98% to 93% till 30 epochs. On the other hand in both the situations (before and after dropout), the validation accuracy flattens out from epoch 6 averaging around 90%.
- It is clearly evident that the application of dropout to this architecture reduced the aggressiveness of the overfitting levels leading to the fact that the gap between train and the validation loss is now much much reduced than previous approach. The network even after adding the dropout layers starts overfitting on the training data from epoch 5 or 6 but at the same time, the validation loss curve is not increasing at a much greater rate than before.
- So to conclude we can say that the earlier network began overfitting much earlier from epoch 3 or 4 and at a much higher rate because of the rapid increase in the validation loss curve than the dropout approach where the network began overfitting from epoch 5 or 6 but not at a much higher rate because of the very slow increase in the validation loss curve.

Comparison of Normal Architecture 4 and Dropout Architecture 4

We applied the dropout with the probability of 0.2 after the layer 1 with 600 neurons, after the layer 2 having 400 neurons and also after the layer 3 having 200 neurons. Below is the analysis of the evaluation graphs before and after the application of dropout layers to this particular architecture.



```
532/532 [==============================] - 1s 2ms/step - loss: 0.6006 - accuracy: 0.9278
The test set loss and accuracy is
 [0.6006399989128113, 0.9277647137641907]
```



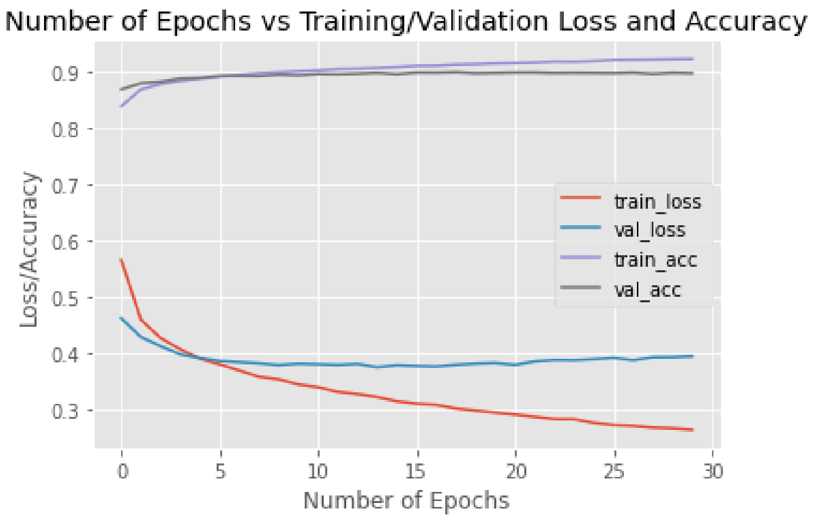
```
532/532 [==============================] - 1s 2ms/step - loss: 0.2704 - accuracy: 0.9315
The test set loss and accuracy is
 [0.2704298496246338, 0.9314705729484558]
```

Conclusion Analysis of the above 2 graphs

- There is not much improvement in the test set accuracy after the application of dropout regularisation to this architecture. Earlier without the dropout layers, the accuracy was 92.77% and then after these dropout layers it was reported only 93.14% (a minute increase).
- After the application of dropout, there is a decline in the final training accuracy from 98% to 94% till 30 epochs. On the other hand in both the situations (before and after dropout), the validation accuracy flattens out from epoch 4 and 16 respectively averaging around 91%.
- It is clearly evident that the application of dropout to this architecture reduced the aggressiveness of the overfitting levels leading to the fact that the gap between train and the validation loss is now much much reduced than previous approach. The network even after adding the dropout layers starts overfitting on the training data from epoch 5 or 6 but at the same time, the validation loss curve is not increasing at a much greater rate than before.
- So to conclude we can say that the earlier network began overfitting much earlier around epoch 4 and at a much higher rate because of the rapid increase in the validation loss curve than the dropout approach where the network began overfitting from epoch 5 or 6 but not at a much higher rate because of the very slow increase in the validation loss curve.

Best architecture for the letter classification problem

- We have applied the dropout regularisation techniques to the architecture 3 and 4. We found that applying this decreases the level of overfitting in each of these architectures and but at the same time, there is no significant improvement in the test set accuracy after applying the dropout.
- Till now we have not applied the dropout to the architecture 2 that have only two layers, therefore we applied the same and found that the overfitting level drops down but as the previous two models, the test set accuracy remained same.
- So we can conclude application of the dropout to the architectures 2, 3 and 4 definitely decreased the level of overfitting but we have already seen in our previous analysis that increasing the number of layers is not making the significant increase in the levels of test set accuracy which remains around 92% in all the three models with dropout.
- Therefore it is best to choose the architecture 2 with the dropout layer of 0.2 as unnecessarily with the increase in the number of layers we are increasing the complexity of our model as well as the training time.
- Below is the evaluation graph of the architecture 2 with the dropout layer.



```
532/532 [==============================] - 1s 2ms/step - loss: 0.3081 - accuracy: 0.9188
The test set loss and accuracy is
 [0.3081086277961731, 0.9187647104263306]
```

Part C: Research [20 Marks]

What is Batch Normalization?

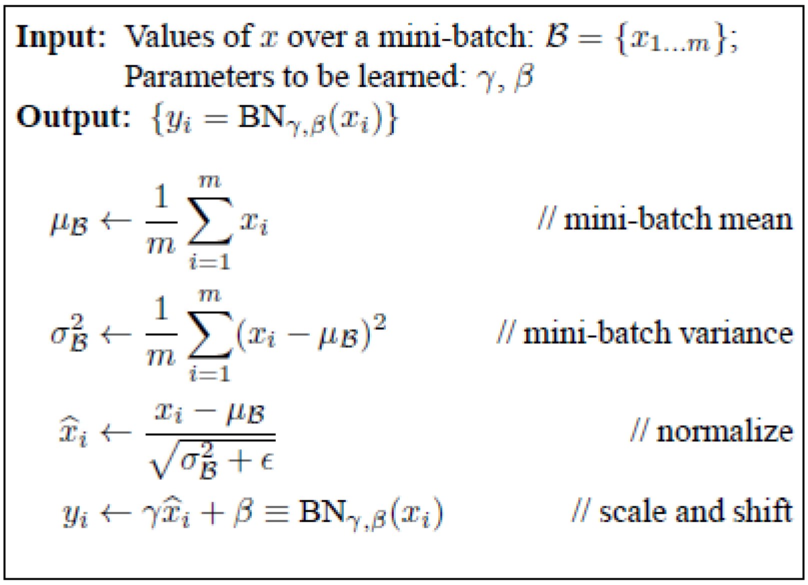
- Batch Normalisation is the algorithmic technique that normalises the input data for each mini batch that is incoming to the very deep hidden layers to improve the stability, performance and the speed of the training process in deep neural networks.
- **Why we need to normalise our input data in the deep layers** – Let us understand this by some example. Let's say we train our network with the colored images of cats meaning that the network has learnt the distribution of the pixel intensities in these training images. Now if we test our model on the black images of cats, then it will for sure not perform well in the scenario where we have not included batch normalisation to our model architecture in the training process. So it is very essential to include batch norm to the deep layers in the network so that the network can learn right normalised distribution in training process and perform well on the unseen data.

Description of the Problem that Batch Norm Addresses

- Data Normalisation is the technique as the part of pre-processing the data before pushing the training examples in batches for the training process. This is simply mean centring and variance scaling that we have seen in our machine learning module where goal is to bring the feature to a common scale.
- Let's say we normalise our batch data before pushing it to our deep neural network, then this data is viewed as the right distribution only to the neurons in the initial layers, but for the deeper neurons this will be a totally different distribution leading to the incorrect training of the network. This distribution change is happening because of the complex operations in the deep neural network leading to a fact that a small change in the beginning can cause a significant change in the deep layers of the network.
- This change in the distribution of the batch training examples acting as the inputs to the deeper layers in the network is called ***internal covariate shift*** (change in the original distribution of covariates i.e. predictors or input variables).
- This problem of internal covariate shift within the deep neural network can cause the network to reach the convergence in much greater amount of time thus increasing the training time which we definitely don't want.
- Batch normalisation tend to reduce these covariate shifts as the network training is most efficient when the distribution of input data is to a similar level in each of the layers in the network, thus reducing the training time and producing the reliable models.

Operation/Working of Batch Normalisation

- To overcome this problem of internal covariate shift, we introduce a batch norm layer which normalizes the mini batch before getting into the deeper layers and now every batch that will be passed through this layer would be normalized.
- In simple words, we are basically standardizing the activation outputs for each input variable from the previous layer so that this normalized input would go into the next layer.
- Below is the snapshot of the actual research paper titled "*Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*"



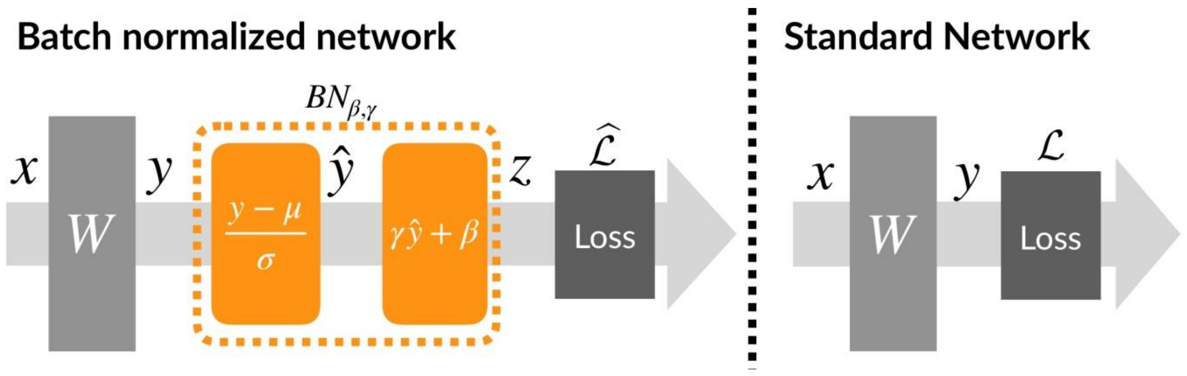
```
**Input:** Values of $x$ over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
           Parameters to be learned: $\gamma, \beta$
**Output:** $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

    $\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i$                          // mini-batch mean
    $\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\mathcal{B})^2$    // mini-batch variance
    $\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}}$                     // normalize
    $y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i)$        // scale and shift
```

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

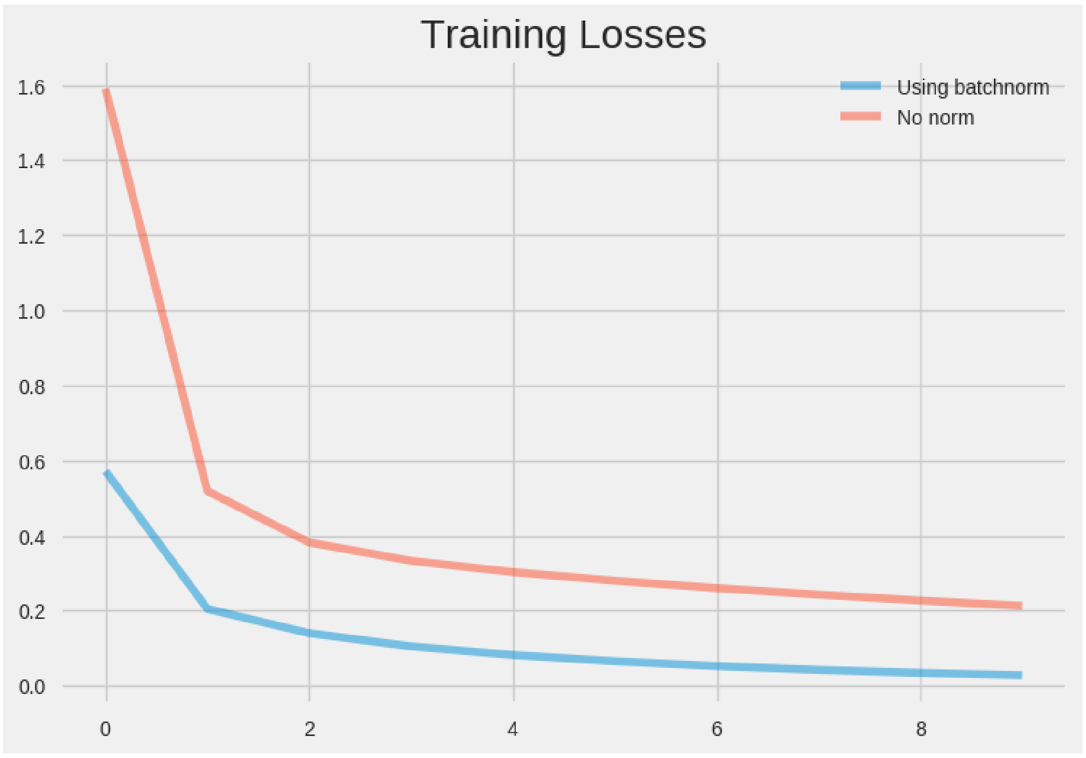
1) Here the x denotes the activation data obtained from the previous layer and the size of the mini batch is m which needs to be standardized.
2) We first take the mini batch mean by dividing the sum of all values in batch with its size.
3) The next step is to compute the variance of the mini batch B containing m values.
4) We then transform/normalize each value in the mini batch B such that the new mean of the batch comes out to be 0 and standard deviation of 1 when the constant epsilon is not taken into account which is responsible for the numerical stability by adding to mini batch variance.
5) Finally we apply the batch normalizing transform by multiplying each value by gamma and adding the result to beta.
6) This way we get the new values in our mini batch B through scaling and shifting.

- We can see that the final normalized value in the current mini batch B is controlled by two parameter gamma and beta. These are learned in the same way as the weights are learnt through backpropagation algorithm.
- It is clearly visible that all the steps in the process of Batch Normalisation are differentiable; therefore we can have it as a layer with parameter learned through back-propagation for which the only requirement is the calculation of gradients that we can certainly do with all the differentiable functions.



Potential Advantages of Batch Normalisation

1. **Enable higher learning rate:** In neural network, if we use the large learning rates then it will amplify the gradients during back-propagation and lead to the model explosion and the network may stuck in local minima. The inclusion of batch normalisation of the activations throughout the network allows the usage of higher learning rate by preventing this vanishing or exploding gradients problem.
2. **Provides weak regularisation to the network:** Batch normalisation also seems to add some regularization effect to the network which in turn improves the generalisation capability of the model because of the introduction of some noise to the deep layer activations and mitigate overfitting. Therefore in some cases we can totally remove or reduce the dropout regularisation from our network and preventing the loss of any information.
3. **Provides faster network training:** Reduction in the covariate shift weakens the coupling between the parameters of early layer and that of deep layers, thus allowing each layer of the network to learn by itself. Thus the network tends to converge faster and the training time is significantly reduced.
4. **Less Sensitive to weight initialization schemes:** Generally the deep networks are sensitive to the weight initialization schemes, but the inclusion of the batch normalisation to the network, introduces the stability to the network and makes it more robust with the selection of weight initialisation choices.
5. **Simplifies the creation of deeper networks:** Due to the above points, it is much easier to build and train deep neural models and produce better results.



Observation from the above graph: It is clearly evident that the training loss converges around 9 epochs when batch normalisation was used and when batch norm was not used we can see that the network still not converges at epoch 9. This means that the use of batch norm tends to reduce the training time.

References

[1] https://en.wikipedia.org/wiki/Batch_normalization

[2] Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift

[3]https://machinelearningmastery.com/batch-normalization-for-training-of-deep-neural-networks/