HW 1 Report

For running the code I've used google collab. I am providing the .ipnb files. For this you need to upload the irish.test, irish.dev, irish.train for every .ipnb notebook. Further I've added the .py file for each module as well. Each module is placed in a separate file.

1.a [1 credit] Model Read/Write:

For this part I've used inbuilt torch.save() function. The save function takes 2 arguments

1) model_state which contains the state and optimizer

2) Path: where to save the file.

To load the saved model, I've used torch.load() function which takes in 1 argument, path, to retrieve saved model.

The model is stored in .pth file. Here I've saved my model in model_save.pth.

References: https://stackoverflow.com/questions/42703500/best-way-to-save-a-trained-model-in-pytorch

1.b [1 credit] Unknown Words:

For this I've changed the prepare_sequence function to handle the words which are not in the training data and replace them 'UNK' which are not in training data. By training the model on the Irish(Testing and Training files). For 10 epochs, I've got an **accuracy of** `79.5 %.`

The model is saved in model_save.pth file.

The second part where we are supposed to be substitute the training_data with "UNK" I have created 2 functions

1) Substitute_with_UNK() // Parameters: data and n. The data is training data and the value of n is used to weed out the words which are less than n.
2) Rare_words_to_UNK() // replaces the less frequent words with 'UNK' in the training data.

By training the model to replace the less frequent words with 'UNK' on Irish(Testing and Training files), For 10 epochs, I've got an **accuracy of** `81.5%.`

The model is saved in `model_save_UNK.pth.`

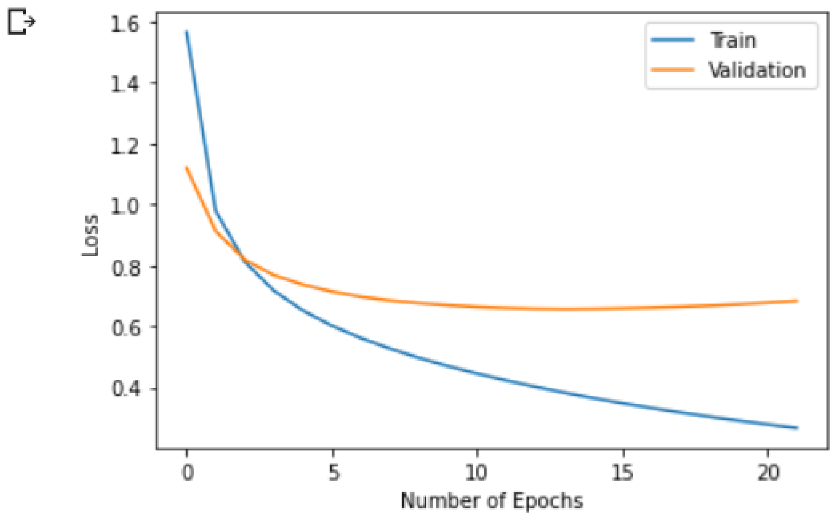
References: https://stackoverflow.com/questions/36656870/replace-rare-word-tokens-python

1.c [1 credit] Early Stopping: I have applied Early stopping criteria on Irish (Training, Testing and development) data set. Used Patience as 8 (I,e.) watch the model till 8 epochs if the validation loss does not improve for 8 consecutive epochs. Restore the previous model.

For **Irish dataset without replacing with UNK.** I've gotten an **accuracy of of 80.02 %.** The model has reached an **Early stopping at Epoch 21 (**Restoring the model which is saved at Epoch 13).

Model is saved at model_save_Early_stop.

Graph for the same is.

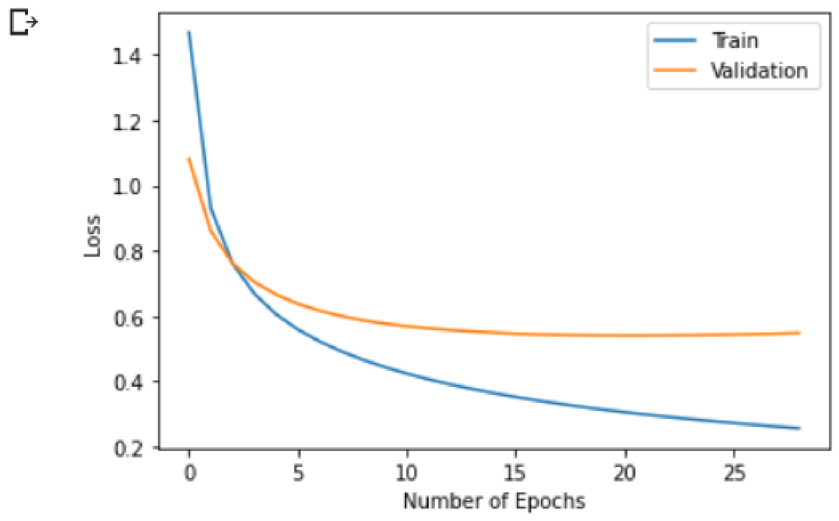


Early stopping with UNK: After replacing the less frequent words as UNK in the training set and applying Early stopping criteria on Irish(Training, Testing and development) data set.

For Irish dataset with replacing with UNK. I've gotten an **accuracy of 83.36%.** The model has reached an **Early stopping at Epoch 28 (**Restoring the model which is saved at Epoch 20).

The model is saved at model_save_Early_stop_UNK.pth.

Graph for the same is attached below.



1.d [1 credit] Batching: For batching I've used pad_sequence(from torch.nn.utils.rnn) and DataLoader(from torch.utils.data) to pad and batch the padded data after sending the training data in the process sequence. For 'PAD' I've used the padding value as Zero.

For Irish data set (training and test) **by using SGD optimizer, with a batch size of 32.** I've gotten an **accuracy of 43%.** One of the reasons for dropping accuracy is that the training data is getting padded. As the neural network does not understand that the pads are insignificant.

But with **Adam optimizer and a batch size of 32**, I've gotten an **accuracy of 85.11%**.

Yes, the training speed does increase significantly when used batching.

The model is saved at model_save_Batch.pth.

References: https://discuss.pytorch.org/t/facing-an-issue-while-using-batching-in-lstm/95263

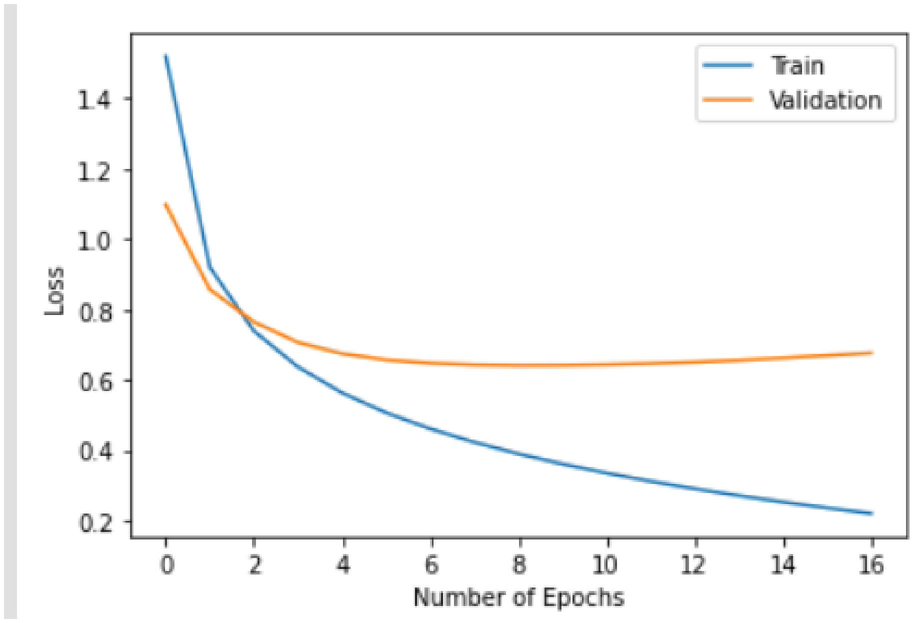
2.a [1 credit] Incorporate context from both sides:

What performance do you get on the Irish development set by incorporating the right-hand-side context into your models?

After incorporating the bidirectional LSTM, on the Irish dataset (Training, Dev and Testing) the accuracy of the model significantly improved when compared with batching (from 43%) to 81.35% on SGD.

The time taken to train the model significantly increased. That is the tradeoff to performance and accuracy.

The graph for the same is:



2.b [2 credits] Optimizing training and hyper-parameters:

For this I have used the Bi-LSTM and added number of layers as 2

The best accuracy, the model has achieved is for the following hyper parameters:

number of layers = 2	dropout = 0.2	hidden dimension = 32
optimizer = Adam	Accuracy = 84.44	Irish test dataset
early stopped, restored epoch 17		

Further, I've tested for other combinations of parameters as well.

number of layers = 2	dropout = 0.5	hidden dimension = 32
optimizer = SGD	Accuracy = 80.67	50 epocs
Irish test dataset		

number of layers = 2	dropout = 0.2	hidden dimension = 16
optimizer = Adam	Accuracy = 83.4	Irish test dataset
Early stopped, restored epoch 10		

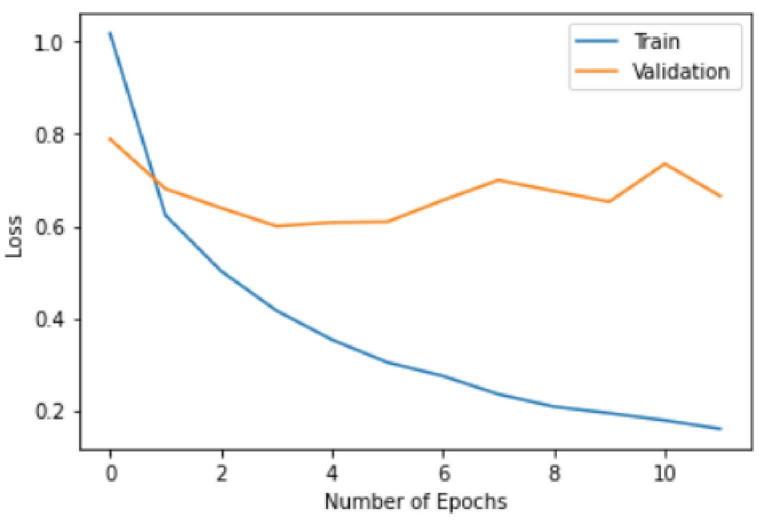
number of layers = 2	dropout = 0.2	hidden dimension = 64
Embedding dimension = 16	optimizer = Adam	
Accuracy = 82.9	Early stopped, restored epoch 3.	

What is your takeaway of the best hyper-parameter setting?

My takeaway is that the model converges faster in Adam optimizer when compared to SGD. If early stopping criterion is not applied there is a heavy chance that we might over fit the model in Adam optimizer.

As the number of layers increased the training time also increased. The early stopping criterion is being reached way faster.

Curves of SGD on the data are much smoother after higher epochs than Adam optimizer because Adam has a larger update value when they reach the steady state.



Graph of Adam optimizer.