Eeshaan Ravi Tivari (2019465)
Manvi Goel (2019472)

REPORT
Assignment 2

CSE641- Deep Learning

PART I: Convolution Neural Network (CNN)

Note -

1. *We have reduced the size of training data to half, to reduce the computation time as with full data the model is taking a longer duration to complete even with GPU usage. Also, since with even half of the training data we are achieving 70%+ accuracy so it is worth reducing the size of train data.*

2. *We've used 2 fully connected layers to show the working of dropout when used between fully connected layers. Since we have to use dropout between fully connected layers as mentioned in Q4) (ii), at least 2 layers are required which is why we've used 2 fully connected layers in our network.*

3. ***Pre-Processing***
   - *We have resized all the images to 64 X 64 size for uniformity.*
   - *All the images are converted to PyTorch tensors.*
   - *The data is standardized before using for training using the formula*

     X = [(X - mean) / std]

     where mean is mean and std is standard deviation

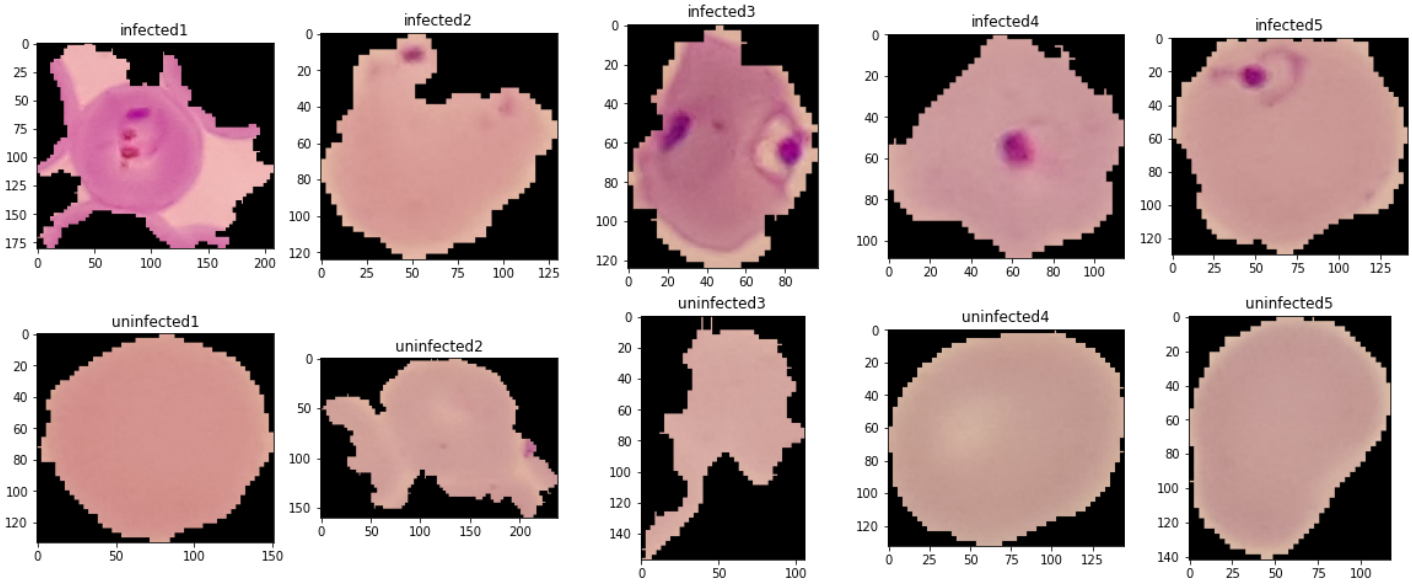
4. *Batch size used = 512, training epochs used = 15*

5. *Our network architecture -*

```
CNN(
  (conv1): Conv2d(3, 5, kernel_size=(9, 9), stride=(1, 1))
  (conv2): Conv2d(5, 7, kernel_size=(6, 6), stride=(1, 1))
  (conv3): Conv2d(7, 8, kernel_size=(3, 3), stride=(1, 1))
  (dropout1): Dropout(p=0.5, inplace=False)
  (fc1): Linear(in_features=3528, out_features=50, bias=True)
  (fc2): Linear(in_features=50, out_features=1, bias=True)
)
```

1. Visualize 5 samples from each class

For this, first, we created lists of all images of both infected and uninfected classes. Then we generated a list of 5 unique indexes from the range 0 to 100 randomly. Using these indexes, 5 random images from both the classes are read and displayed.



2. Implement a CNN architecture

For this, we have defined a class CNN which initializes the convolution neural network object with all the required parameters.

CNN has -

3 convolution layers, and 2 fully connected layers, Sigmoid activation at the output layer
Dropout layer is also present which is activated using a parameter in the __init__() method.

Function in CNN class -

- __init__(self,initialization,name,dropout_control=0,regularization=None,alpha=0):
  Method to initialize the class object responsible for CNN network. All the layers of the CNN are defined within this method along with other data members such as regularization, model_name, and dropout_control. initialization is a string parameter that tells the network what type of weight initialization is to be done for the network. name is a string parameter representing the model's name used on saving in its path. Dropout_control and regularization as the name suggests are used to control the where and which type of regularization to use with alpha referring to the regularization parameter.
- forward(self, x):
  Method to do forward pass using input x and return the output of the model.
- Initialize(self, method):

Helper function to do the weight initialization of the model using the criteria specified by the string parameter method. ‘method’ can take 3 values - “zero”, “random”, “He” for the zero, random and He initialization.

3. Initializing weights with Zero or Random or He initialization

For this we have made a class method named Initialize(self, method) which takes in the criteria to be used for weight initialization or network.

If ‘method’ is “zero”-

Fill all the layer weights with 0 using a tensor. fill_(0) from PyTorch.

If ‘method’ is “random”-

Fill all the layer weights with random values drawn from a normal distribution with a mean of 0 and a standard deviation of 0.01 using the torch.nn.init.normal_(tensor, mean=0.0, std=0.01).

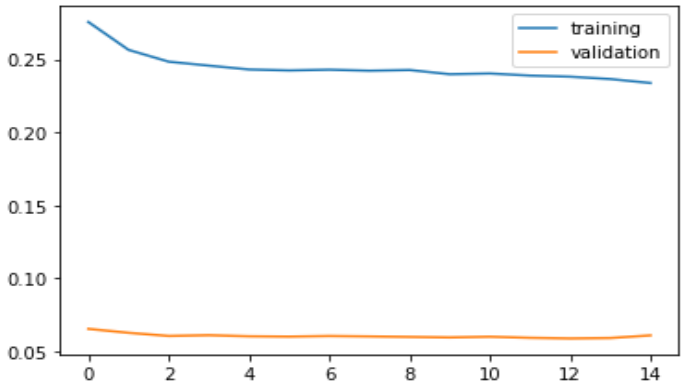
If ‘method’ is “He”-

Fill all the layer weights with He initialization techniques using the torch.nn.init.kaiming_uniform_(tensor).

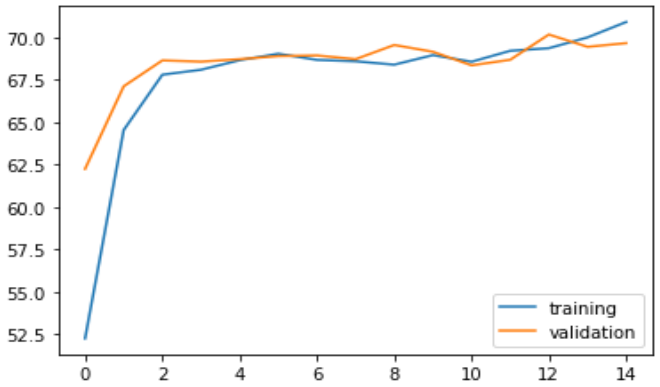
Results -

1. Zero initialization -
   Test accuracy - 69.219%

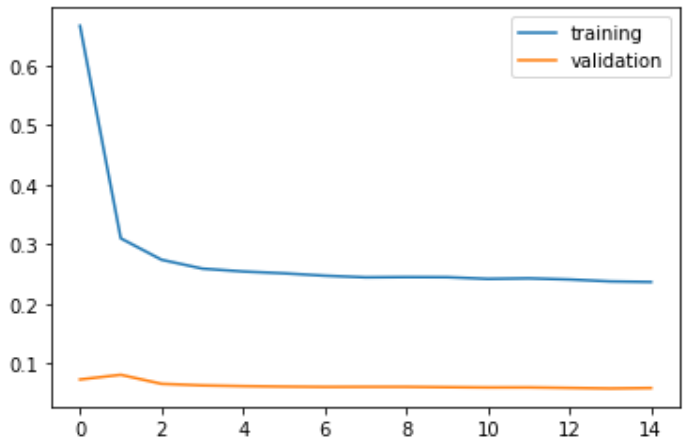
Loss plots

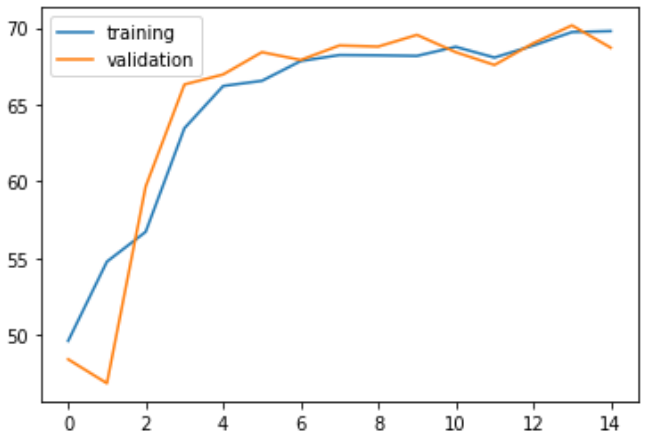


Accuracy plots

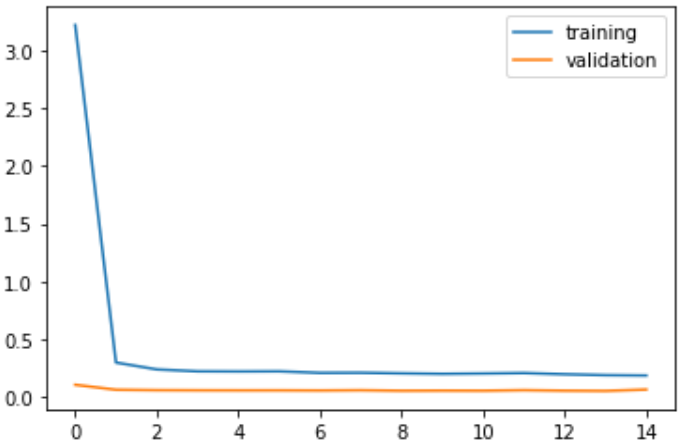


2. Random initialization
   Test accuracy - 69.364%





3. He initialization
   Test accuracy - 67.695%



Loss plots

Accuracy plots

Analysis-
He initialization worked the best because compared to both zero and random initialization model, He initialized model is less biased as the difference between the training and validation loss is minimum in its case after complete training. Also, it clocked the highest accuracy for both train and validation sets i.e. Train Accuracy: 78.483% and Validation Accuracy: 75.971% after epoch 14's completion after which the model shows some overfitting trends and validation accuracy took a hit. But this proves He initialization works the best.

4. Use Dropout for i) After convolutional layers, ii) Between fully connected Layers
To use dropout in our network we have made a dropout layer in our CNN architecture with keep probability = 0.5, named self.dropout1. To use this dropout layer in the network, the dropout_control parameter in the init() method of our model needs to be set to either 1 or 2.

If dropout_control is 1-

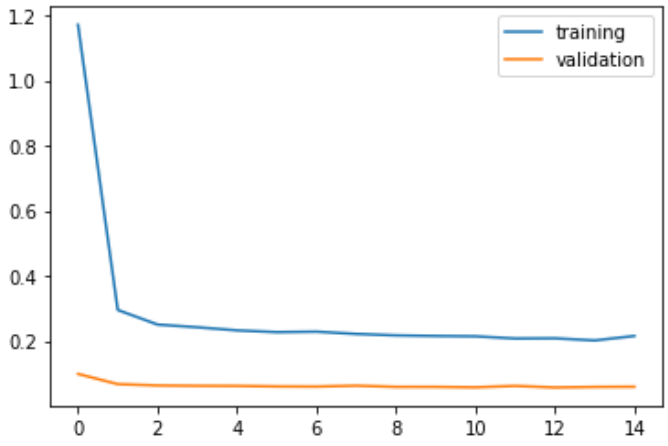
Use dropout layer after convolution layers i.e., before passing as input to the fully connected layers, dropout will be applied to it.

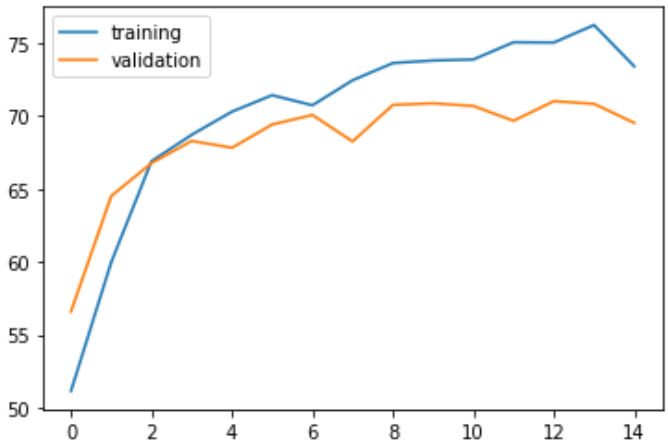
If dropout_control is 2-

Use the dropout layer between fully connected layers i.e., between fully connected layers 1 and 2 (self.fc1 and self.fc2).

Results -

1. Dropout after convolution layers
   Test accuracy - 70.526%

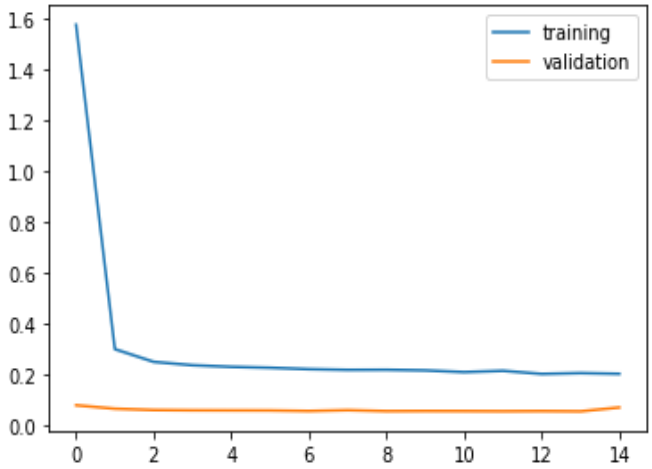




Loss plot

Accuracy Plot

2. Dropout between fully connected layer
   Test accuracy - 61.161%



Analysis-

Both the dropout model have comparable performance however, after epoch 14, the fully connected layer model shows some overfitting trends, and its validation and test accuracy took a sharp dip. Till epoch 14, dropout between fully connected layer model worked slightly better than the after convolution dropout model as it is slightly less biased and almost similar or slightly

better performance. From both the plots we can infer that dropout in both the cases work almost similarly as shapes of both the loss plots and the accuracy plots are also the same except in the accuracy plot after epoch 14.

5. L1 and L2 regularization

To apply regularization in our model, the regularization parameter in the init() method needs to be set to either "L1" or "L2" along with the value for regularization parameter alpha in the function parameter 'alpha' of init() method. If no regularization is passed alpha will be 0.

```
# based on the regularization criteria
# update the loss value accordingly
if model.regularization=="L2":
  l2_val = sum(p.pow(2.0).sum()
               for p in model.parameters())

  loss = loss + model.alpha * l2_val

elif model.regularization=="L1":
  l1_val = sum(p.abs().sum()
               for p in model.parameters())

  loss = loss + model.alpha  * l1_val

loss.backward()
```

If regularization is "L1"-

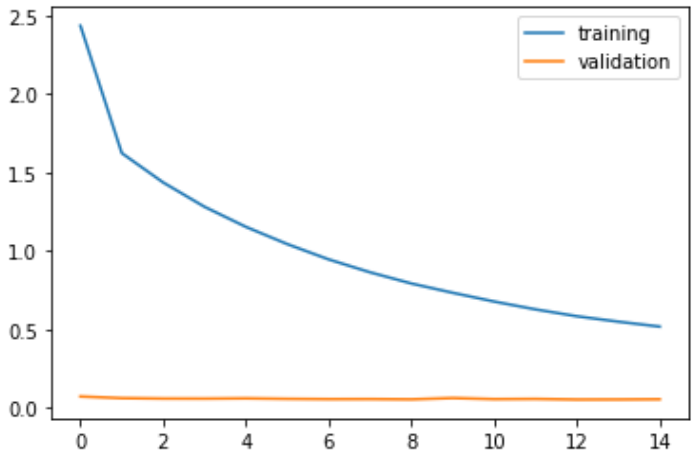
Use L1 regularization i.e. add the product of alpha and the sum of the absolute value of weights of the network in the loss calculated during training.

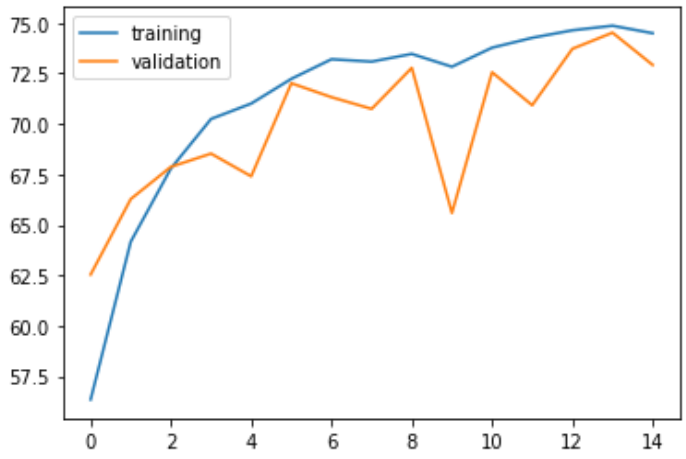
If dropout_control is 2-

Use L2 regularization i.e. add the product of alpha and the sum of squares of weights of the network in the loss calculated during training.

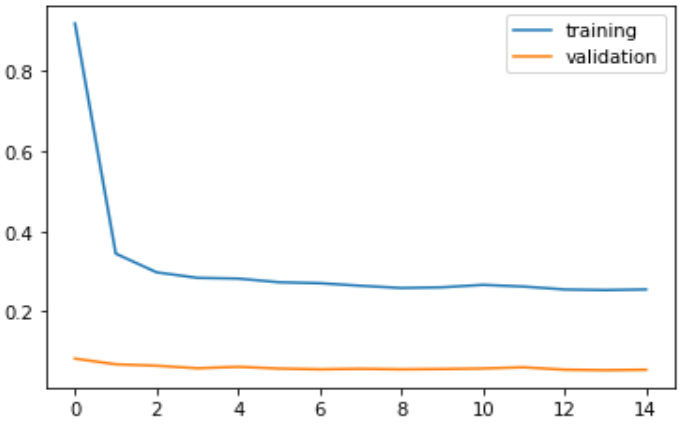
Results-

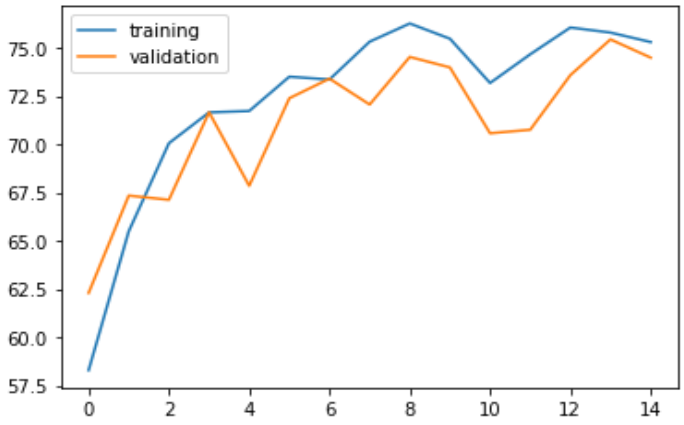
1. L1 regularization
   Test accuracy - 73.502%





2. L2 regularization
   Test Accuracy - 73.103%





Analysis-

The L2 regularized model works better than the L1 counterpart. Both are giving almost the same accuracy on the test set but on the train and validation set, the L2 model shows slightly better results in terms of both loss and accuracy. Also, the L1 model is highly biased when compared to the L2 model as the difference between training and validation loss for the L1 model is much higher than the L2 model (more than twice). Also, we can see that convergence is much slower in the L1 regularized model as it started training with higher loss values which then slowly decreased whereas, in the L2 model, losses converged more quickly to even lower values within the same no. of epochs.

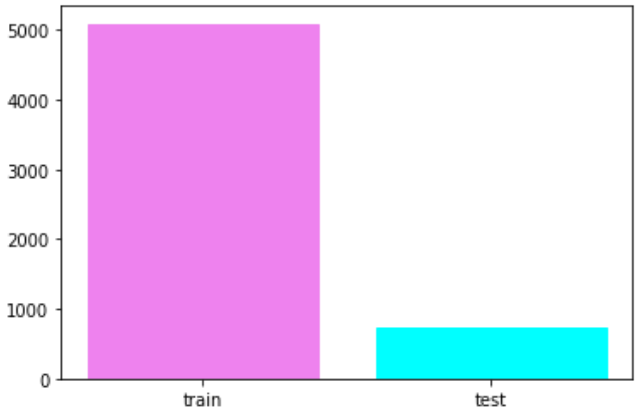
PART II: Long Short Term Memory

Approach

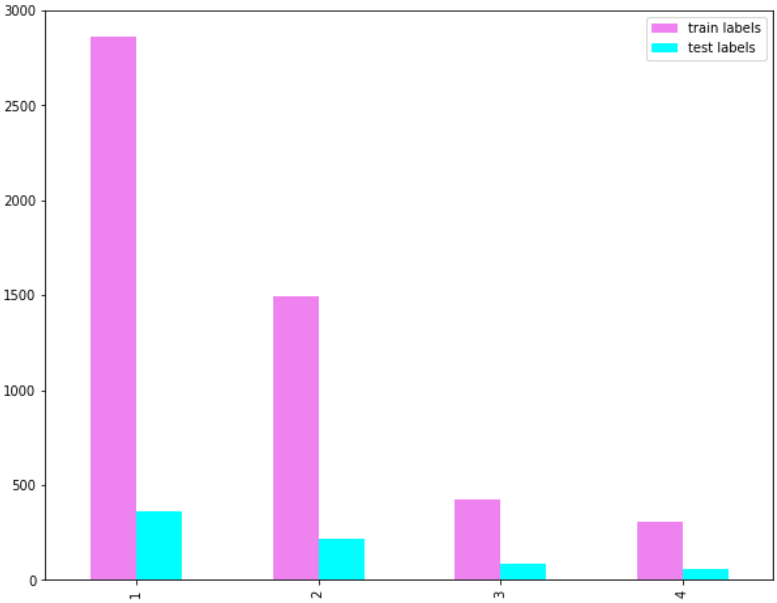
1) Visualizing Dialogue Corpus
   Corpus was visualized using *matplotlib* and *pandas.*

   Legend
   Pink = Training Data, *Cyan* = Testing Data

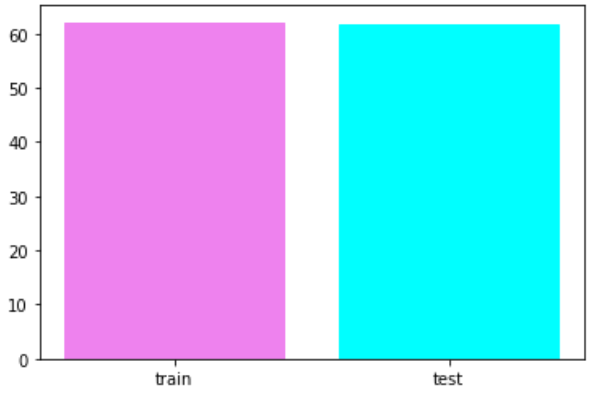
   Size of Train and Test Data



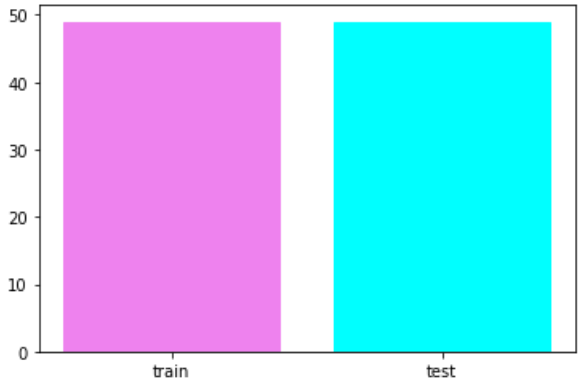
   Distribution of Class Labels



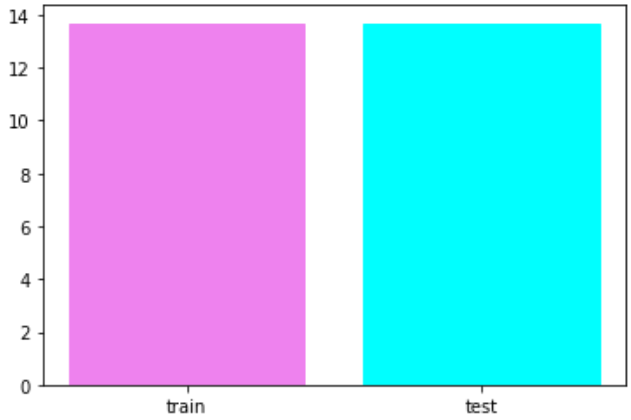
Average Length of Utterance



Median Length of Utterance



Average Number of Words



2) LSTM for Intent Prediction

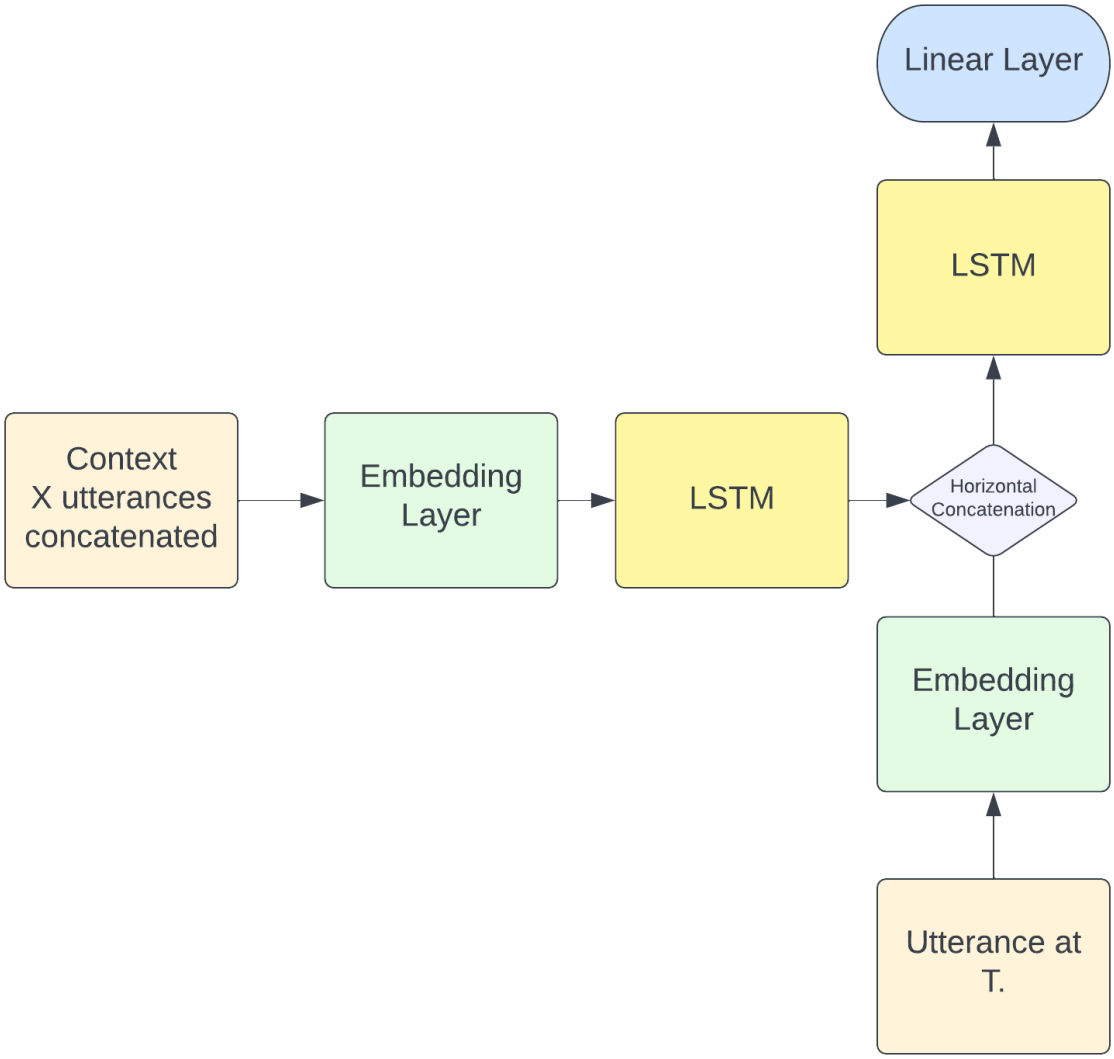
Final Architecture Selected.

2 Embedding Layers (Output = 50)
2 LSTM Layers (Output = 50)
1 Linear Layer

Learning Rate = 0.01
Batch Size = 512

Note: The code uses GPU to speed up the process.

Model



The final model Architecture

Test over Different Architectures. (When Context is 1)

(Change in)	Model Architectures	Validation Accuracy
(Number of Layers)	2 Embedding Layers 2 LSTM Layer 2 Linear Layers 1 Linear Layer Learning Rate = 0.01	50.3
(Number of Layers	2 Embedding Layers 1 LSTM Layer 1 Linear Layers Learning Rate = 0.01	50.3
(Learning Rate)	2 Embedding Layers 2 LSTM Layer 1 Linear Layers Learning Rate = 0.001	Similar Accuracies but longer Convergence
(Learning Rate)	2 Embedding Layers 2 LSTM Layer 1 Linear Layers Learning Rate = 0.02	50.3
(Dimensions of Layers) From 50 all to 30 all	2 Embedding Layers 2 LSTM Layer 1 Linear Layers Learning Rate = 0.01	50.3
(Dimensions of Layers) From 50 all to (50, 30, 30, 10)	2 Embedding Layers 2 LSTM Layer 1 Linear Layers Learning Rate = 0.01	50.3

Code Explanation

Initializing Model.

In the IntentClassfication Class, initializing the neural net using the layer dimension and architecture. The current architecture is given in the code.

Training Model.

Defining a forward function according to the specified model.

Training the model using categorical cross-entropy.

Calculating the required metrics for each epoch for both the training and testing data.

3) Plotting graphs

   Saving the required evaluations metrics as dictionaries and returns to plot the required graphs.

4) Multiple tests were run with different architectures to ensure the result remained the same, regardless of the architecture.

Preprocessing

Removing the punctuations and numbers.

Turning the text into lowercase.

A helper function is used.

Output

Sample Output

The model architecture and the training loss, validation loss, and validation accuracy for every 5th epoch.

```
intentPredictionNN(
  (embeddings1): Embedding(4802, 50, padding_idx=0)
  (lstm1): LSTM(50, 50, batch_first=True)
  (linear): Linear(in_features=50, out_features=5, bias=True)
)
Epoch 1.0: Train loss 1.067, Val loss 1.175, Val accuracy 0.503, and Val rmse 1.190
Epoch 6.0: Train loss 1.065, Val loss 1.189, Val accuracy 0.503, and Val rmse 1.190
```

Does the performance of the model increase with the increase in X?

Comparing the performance of the model over multiple evaluation metrics gives us various insights into the use of context to predict the intent of the utterance. We see that the accuracy of the model does not improve with the addition of context, and the accuracy remains similar overall for all models (the maximum accuracy achieved stays around 74%).

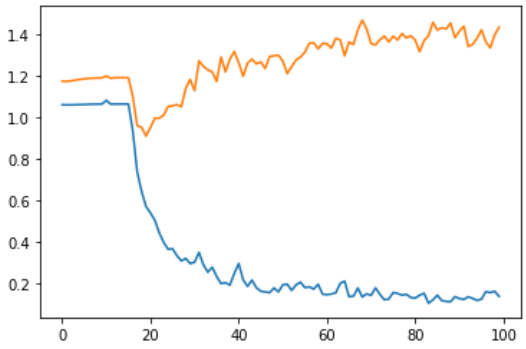
But we can see from the plots, that the epochs taken for convergence changes. The model (X = 1) takes the largest number of epochs to bypass the local minima to converge and the model (X = 4) takes the least (even less than the model (X = 0)). We are not considering the model (X = 0) in the context of epochs taken due to stark differences in the model architecture. We see that with similar architecture, the presence of context vectors can help the model quickly modulate weights to reach convergence.

Thus, we can conclude that the advantage of context depends on the criteria for judging the performance of the model.

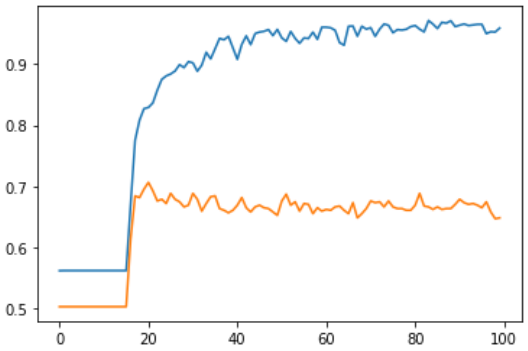
Plots for each Context
Legend Blue: Training and *Orange*: Validation

I. When the context is 0

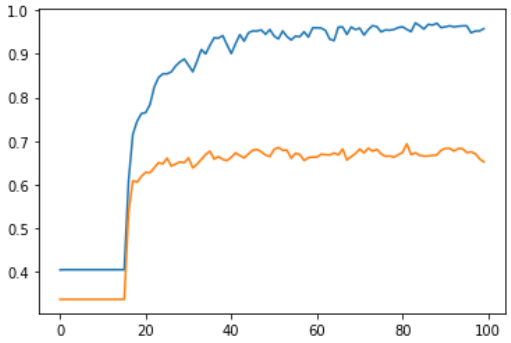
Cross entropy loss



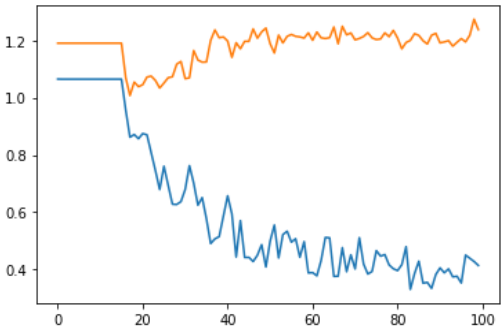
Accuracy



F1 score (weighted)

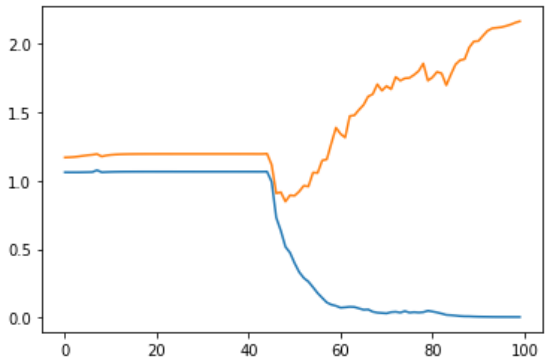


RMSE

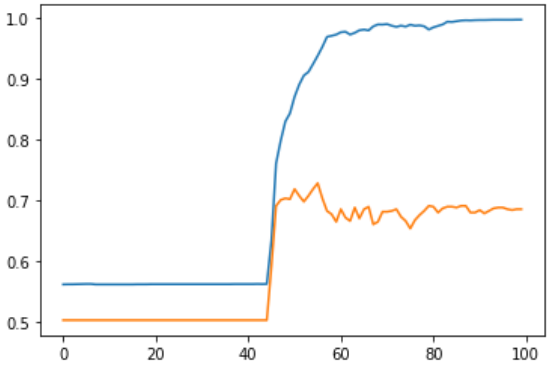


II. When the context is 1

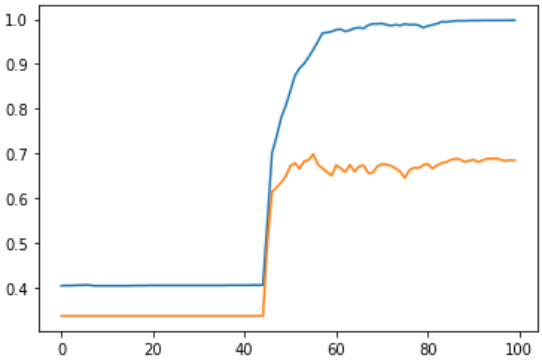
Cross-Entropy Loss



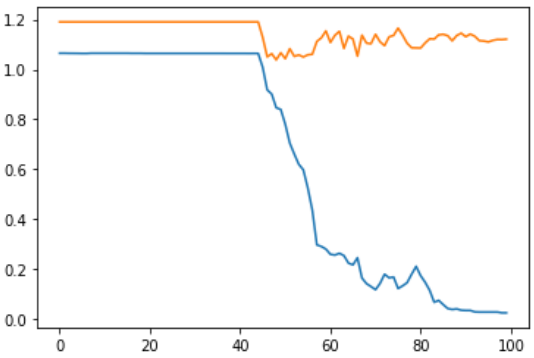
Accuracy



F1 Score

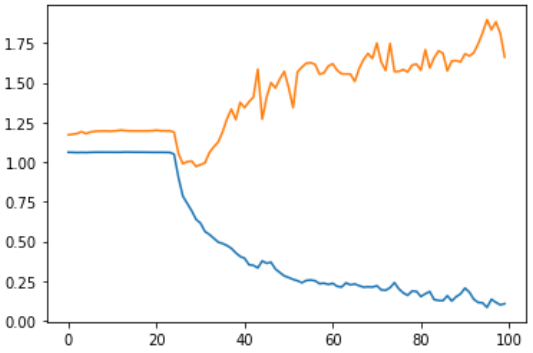


RMSE

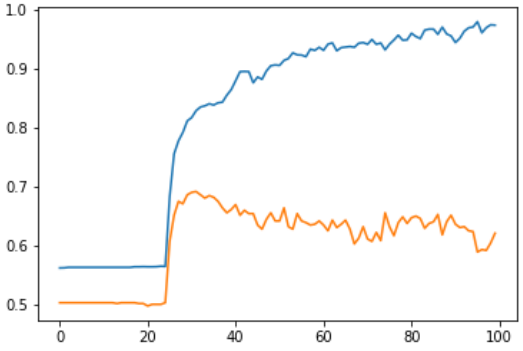


III. When the context is 2

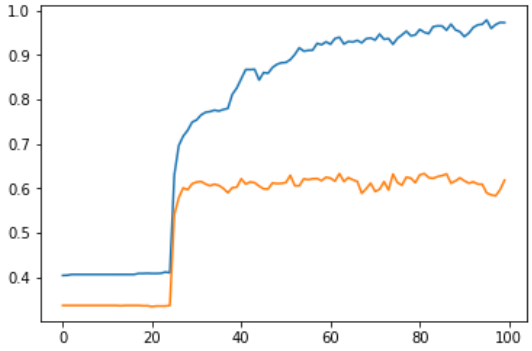
Cross-Entropy Loss



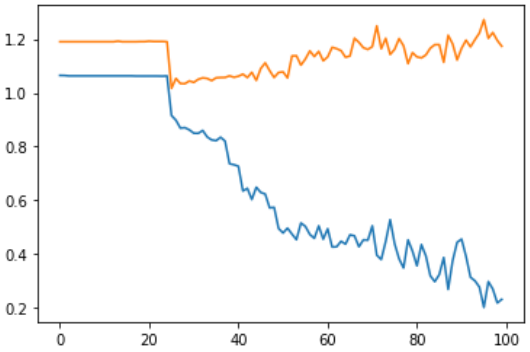
Accuracy



F1 Score

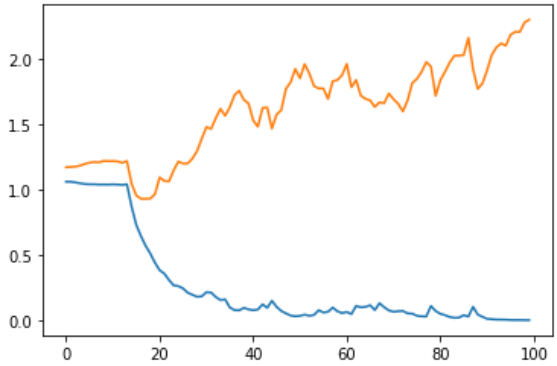


RMSE

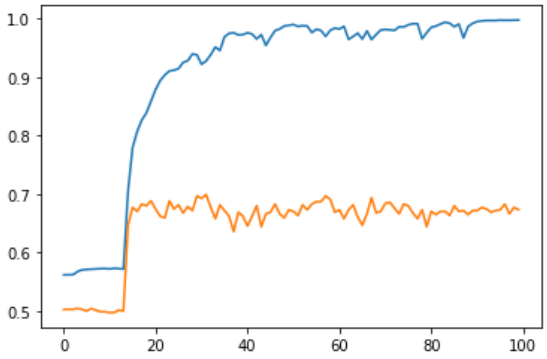


IV. When the context is 3

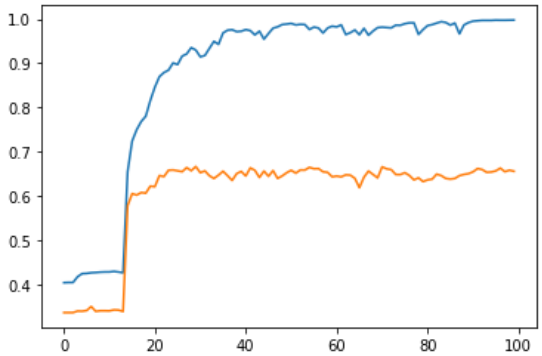
Loss



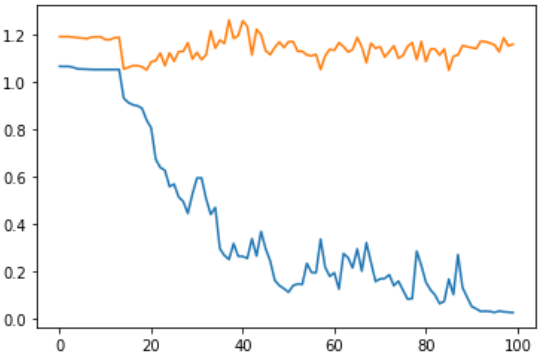
Accuracy



F1 Score

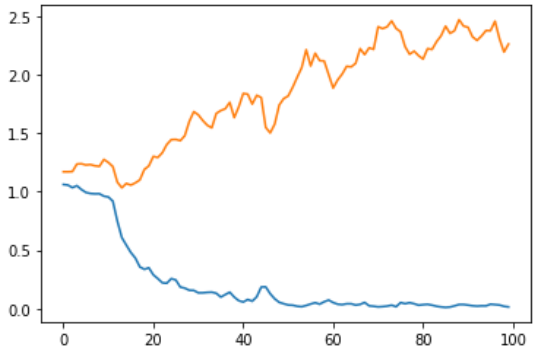


RMSE

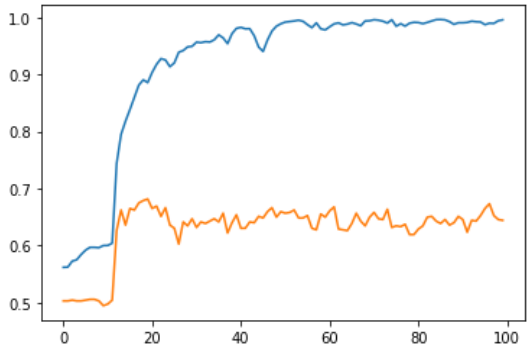


V. When the context is 4

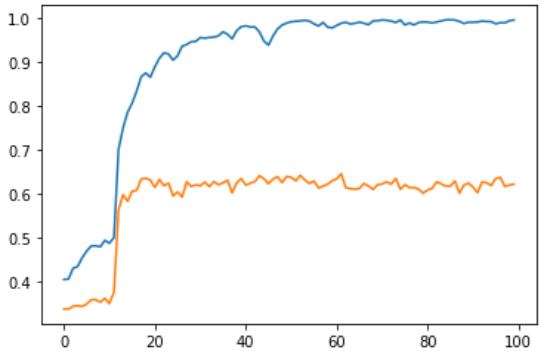
Loss



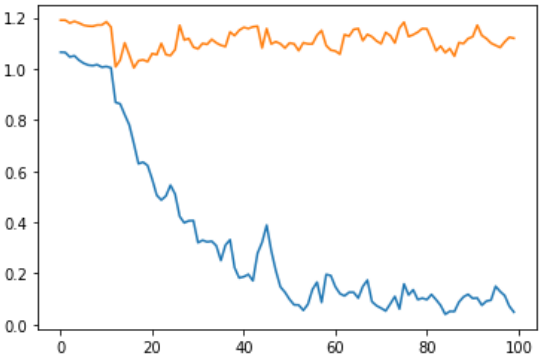
Accuracy



F1 Score



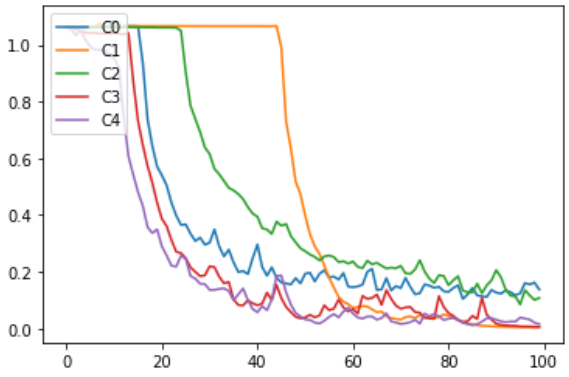
RMSE



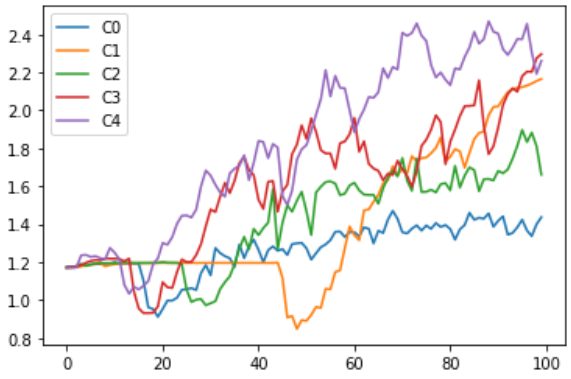
Plots for the values of the metrics across all X

Training Loss

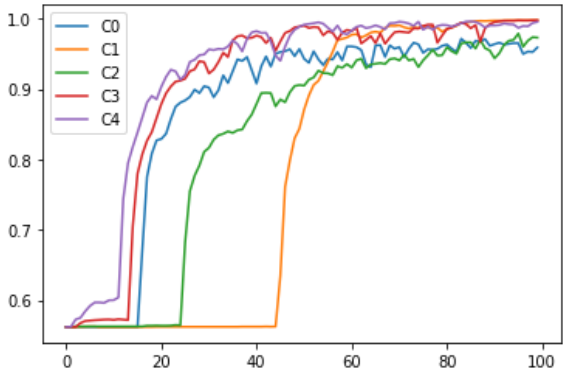
Validation Loss



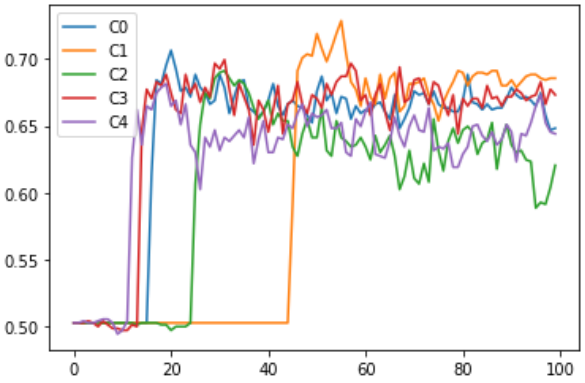
Training Accuracy



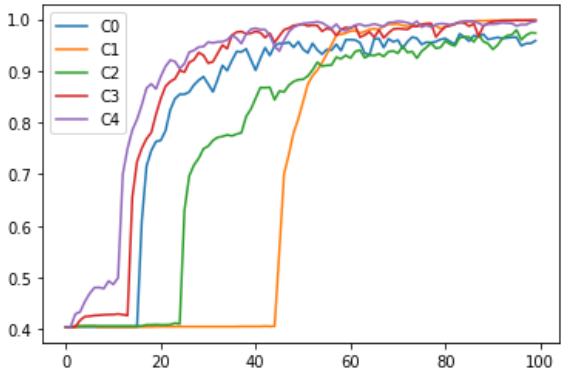
Validation Accuracy



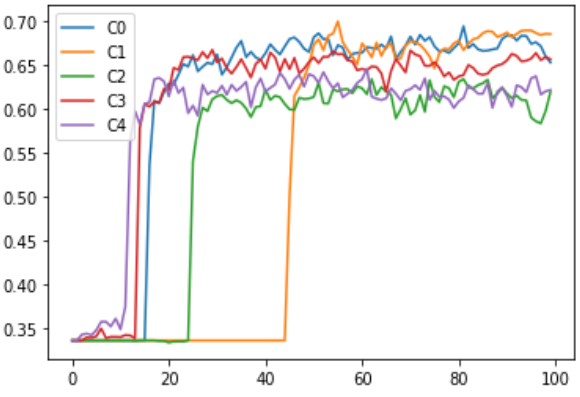
Training F1 Score



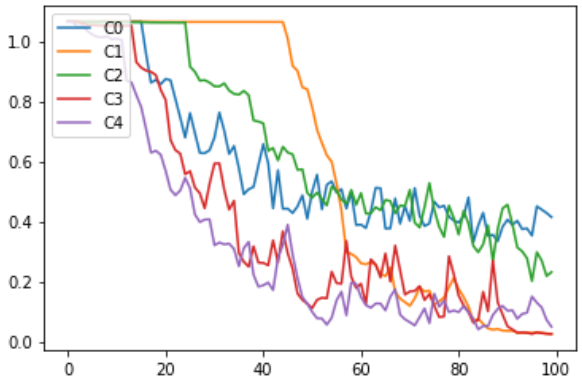
Validation F1 Score

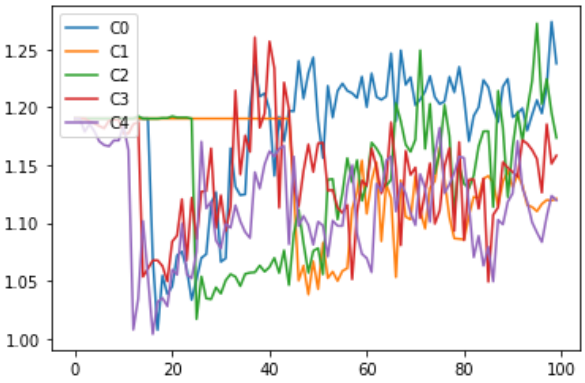


Training RMSE



Validation RMSE


C0
C1
C2
C3
C4
1.0
0.8
0.6
0.4
0.2
0.0
0
20
40
60
80
100


C0
C1
C2
C3
C4
1.25
1.20
1.15
1.10
1.05
1.00
0
20
40
60
80
100

Thank You.