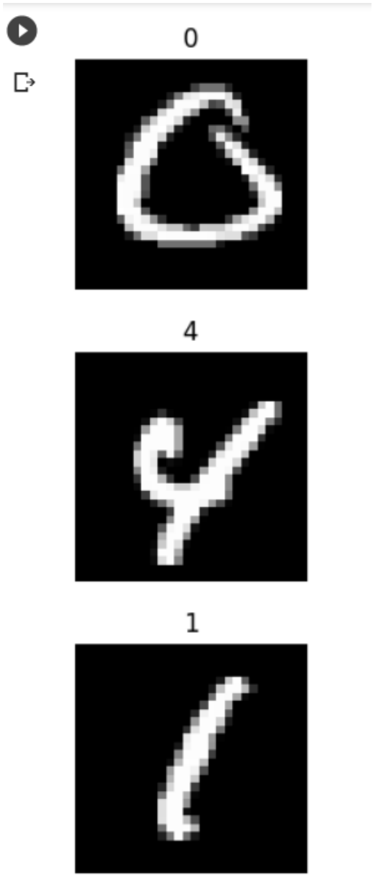
Report

1. Task1 (Classification)

Setup1:

1. Visualization of 3 Random sample images for SEQ Dataset:



2. We created a Simple-RNN based architecture in which the first part is Simple RNN with 256 units and input shape 28 by 28 array consisting of pixel intensities. Then its output acts as input to part 2 which is Multilayer perceptron(MLP) and it has 2 hidden layers consisting of 64, 32 neurons respectively with activation function relu followed by a classification head consisting of 10 neurons with softmax for the given problem of 10 classes(digit zero to nine) classification problem.

Model architecture: (SimpleRNN + MLP)

```
model_seq = Sequential(name="RNN_plus_MLP")

model_seq.add(SimpleRNN(units=units, input_shape=input_shape))
model_seq.add(Dense(64, activation='relu'))
model_seq.add(Dense(32, activation='relu'))
model_seq.add(Dense(10, activation='softmax'))

model_seq.summary()
```

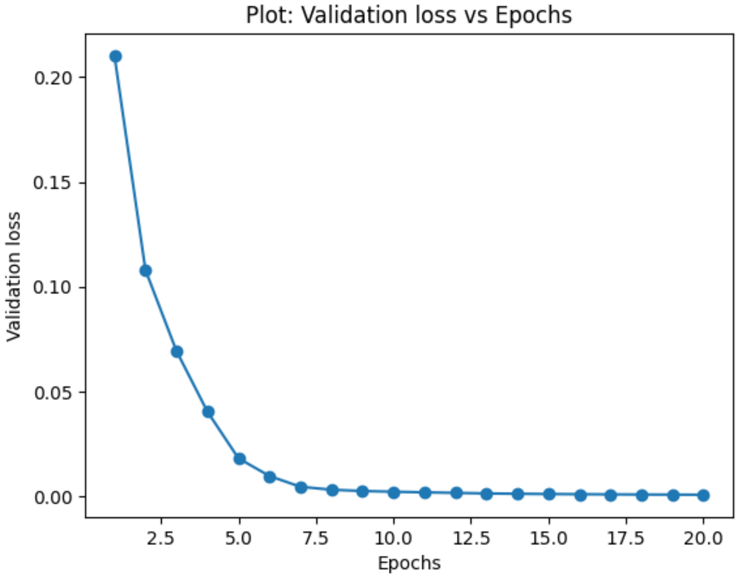
Model Parameters/Hyperparameters:

Input shape = (28, 28)
Units = 256

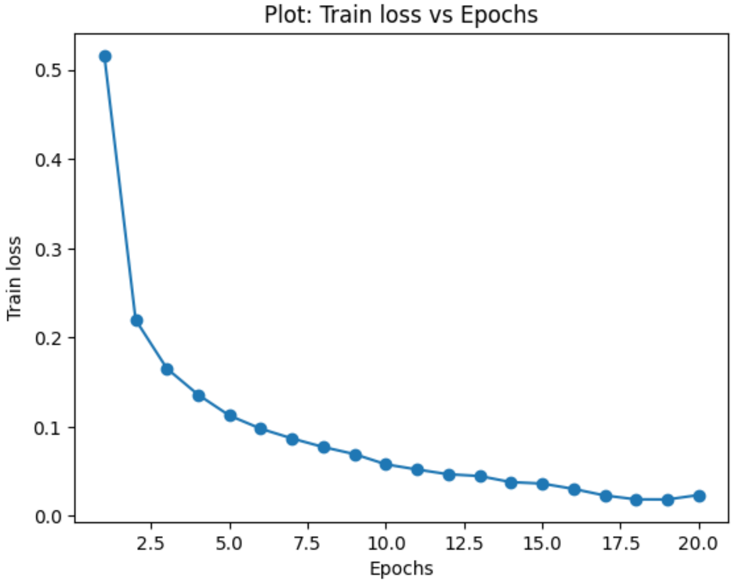
Loss = sparse categorical cross entropy
Optimizer = sgd
Metrics = accuracy

3. Plots:

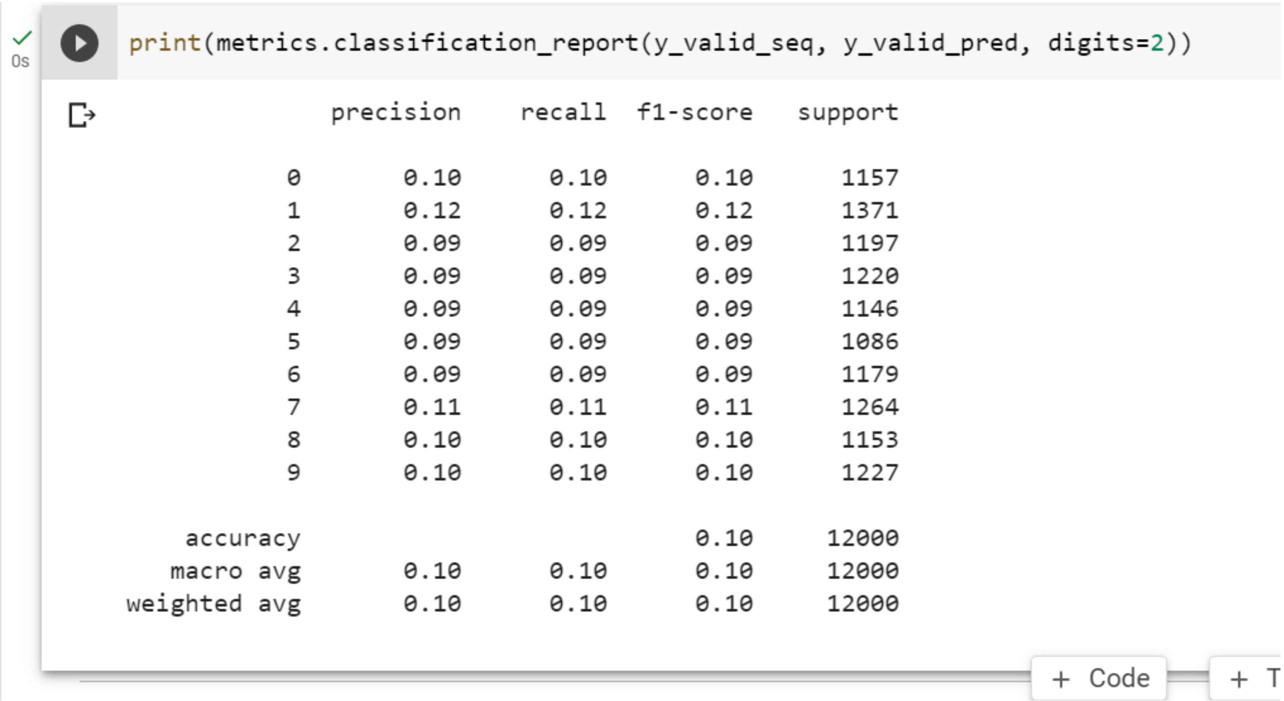
Validation loss vs no. of epochs:



Training loss vs no. of epochs:



4. Class-wise F1 Score:

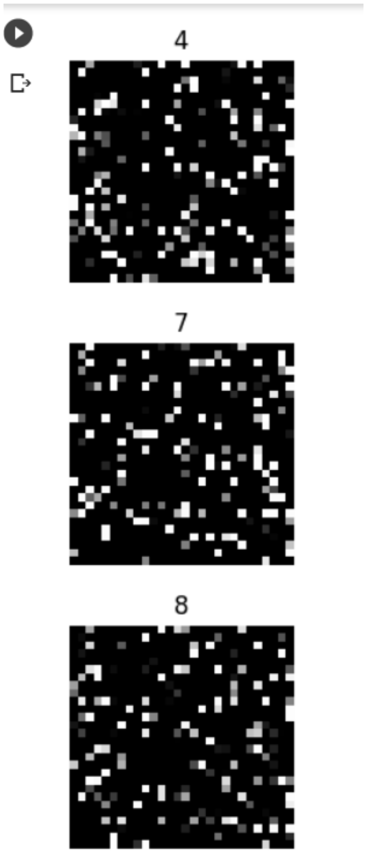


```
print(metrics.classification_report(y_valid_seq, y_valid_pred, digits=2))
```

```
              precision    recall  f1-score   support

           0       0.10      0.10      0.10      1157
           1       0.12      0.12      0.12      1371
           2       0.09      0.09      0.09      1197
           3       0.09      0.09      0.09      1220
           4       0.09      0.09      0.09      1146
           5       0.09      0.09      0.09      1086
           6       0.09      0.09      0.09      1179
           7       0.11      0.11      0.11      1264
           8       0.10      0.10      0.10      1153
           9       0.10      0.10      0.10      1227

    accuracy                           0.10     12000
   macro avg       0.10      0.10      0.10     12000
weighted avg       0.10      0.10      0.10     12000
```

Setup2:

1. Visualization of 3 Random sample images for PERMUTATED Dataset:



2. We created a Simple-RNN based architecture in which the first part is Simple RNN with 256 units and input shape 28 by 28 array consisting of pixel intensities. Then its output acts as input to part 2 which is Multilayer perceptron(MLP) and it has 2 hidden layers consisting of 64, 32 neurons respectively with activation function relu followed by a classification head consisting of 10 neurons with softmax for the given problem of 10 classes(digit zero to nine) classification problem.

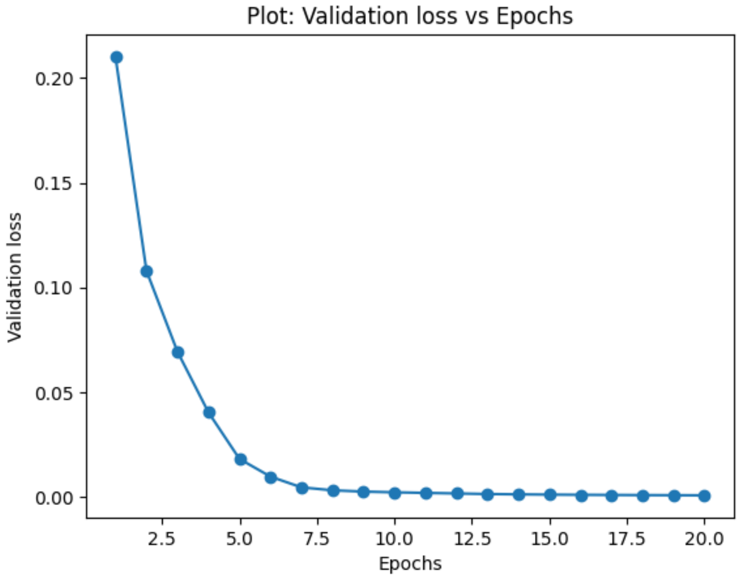
Model architecture: (SimpleRNN + MLP)

```
model_seq = Sequential(name="RNN_plus_MLP")

model_seq.add(SimpleRNN(units=units, input_shape=input_shape))
model_seq.add(Dense(64, activation='relu'))
model_seq.add(Dense(32, activation='relu'))
model_seq.add(Dense(10, activation='softmax'))

model_seq.summary()
```

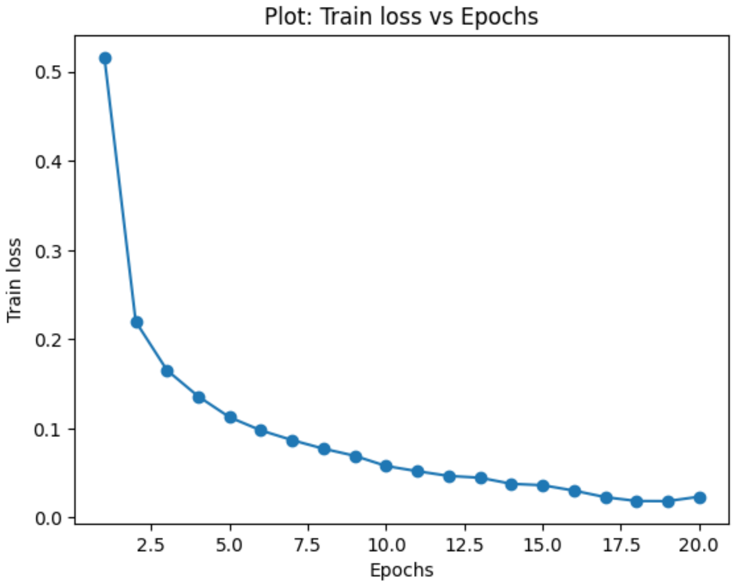
Model Parameters/Hyperparameters:
Input shape = (28, 28)
Units = 256

Loss = sparse categorical cross entropy
Optimizer = sgd
Metrics = accuracy

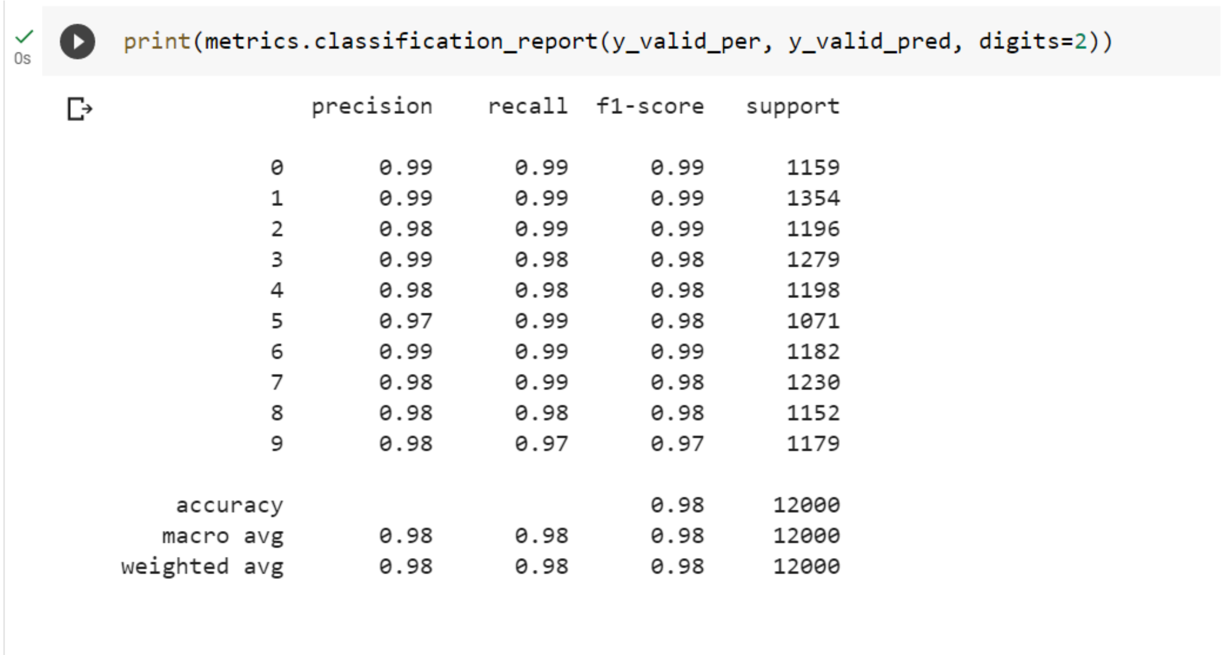
3. Plots:
Validation loss vs no. of epochs:



Training loss vs no. of epochs:



4. Class-wise F1 Score:



```
print(metrics.classification_report(y_valid_per, y_valid_pred, digits=2))
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      1159
           1       0.99      0.99      0.99      1354
           2       0.98      0.99      0.99      1196
           3       0.99      0.98      0.98      1279
           4       0.98      0.98      0.98      1198
           5       0.97      0.99      0.98      1071
           6       0.99      0.99      0.99      1182
           7       0.98      0.99      0.98      1230
           8       0.98      0.98      0.98      1152
           9       0.98      0.97      0.97      1179

    accuracy                           0.98     12000
   macro avg       0.98      0.98      0.98     12000
weighted avg       0.98      0.98      0.98     12000
```

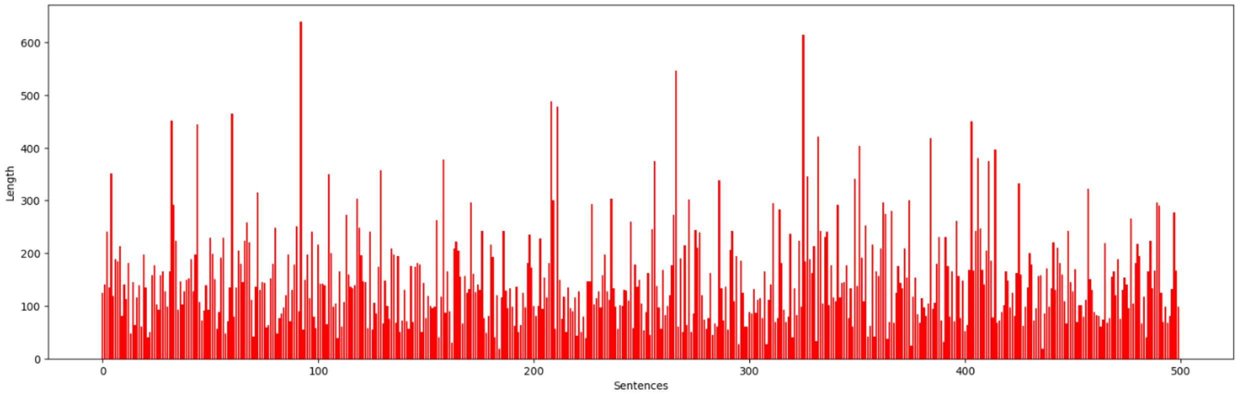
2. Task2 (Translation)

The train data is preprocessed (removal of punctuations, symbols, whitespaces, digits, etc), and cut down to 0.01 of its size for training purposes.

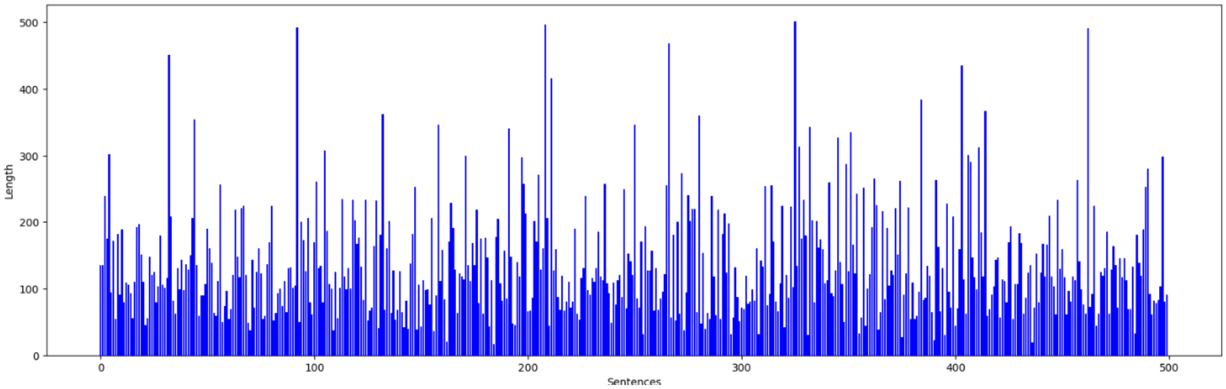
Visualization:

a. Sequence Length

German: Highest recorded was over 600 in the first 500 samples.

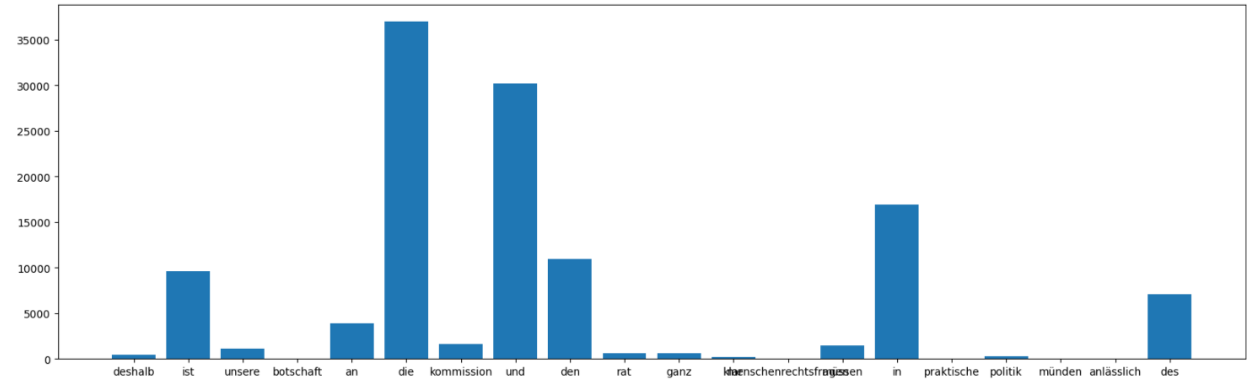


English: Highest recorded was around 500 for the first 500 samples.

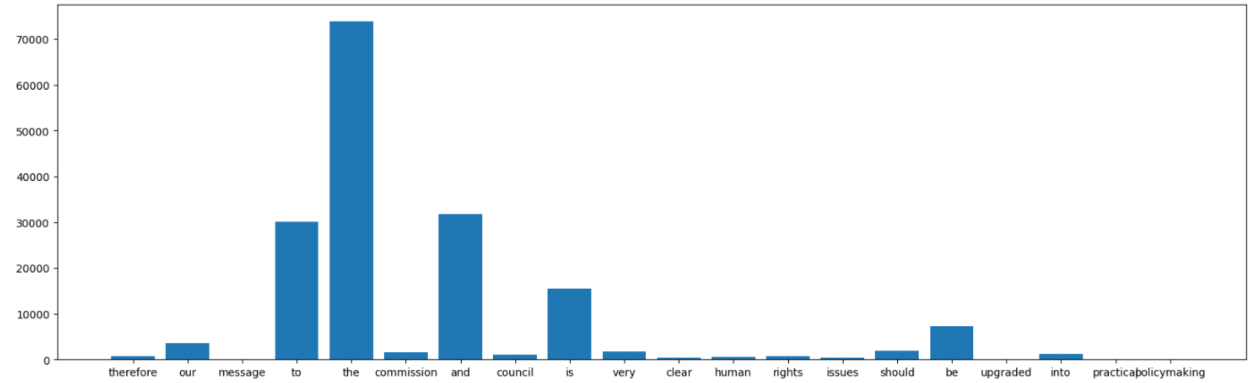


b. Frequency of Words (Analyzing 20 words)

German: Highest frequency around 35,000 for the word ‘de’.

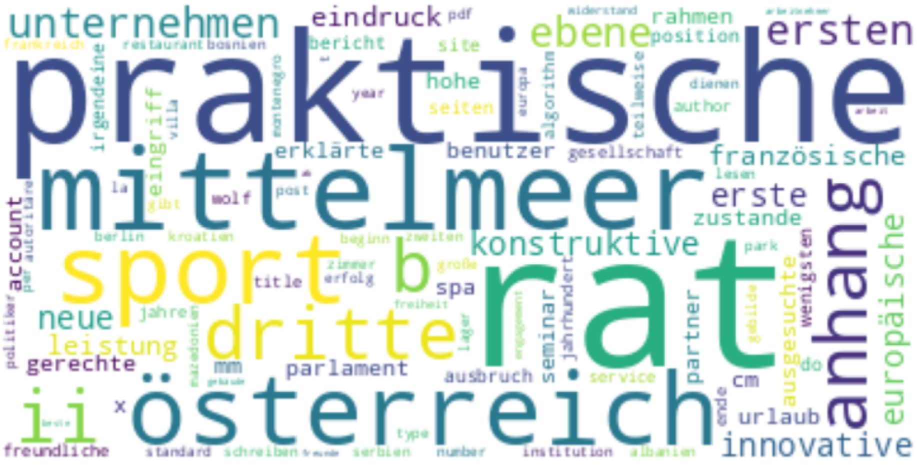


English: Highest frequency around 70,000 for the word ‘the’.

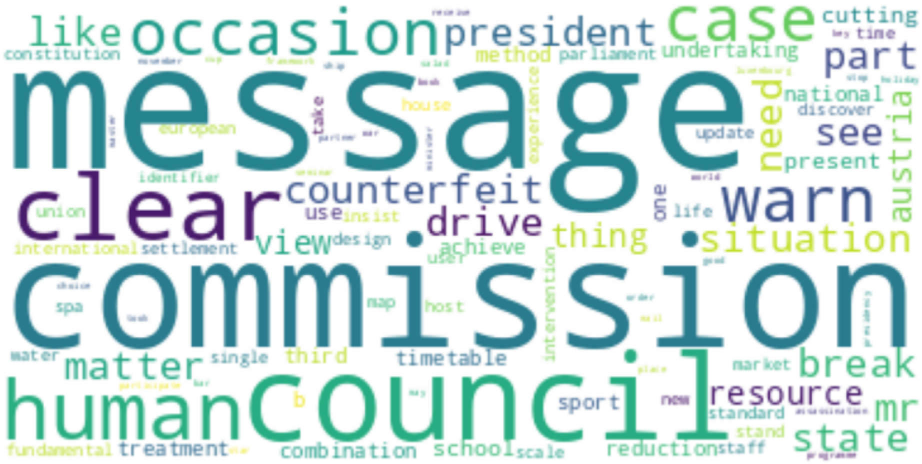


c. Word Cloud

German:

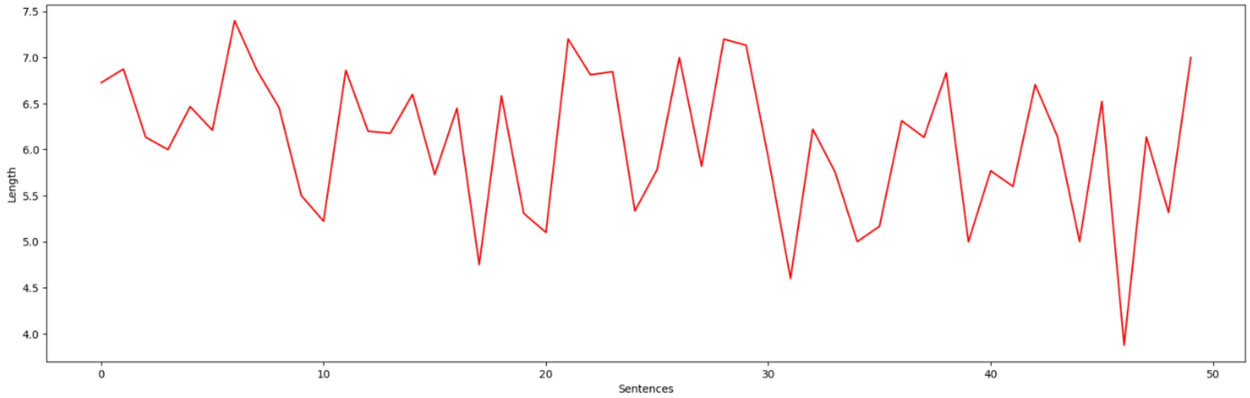


English:

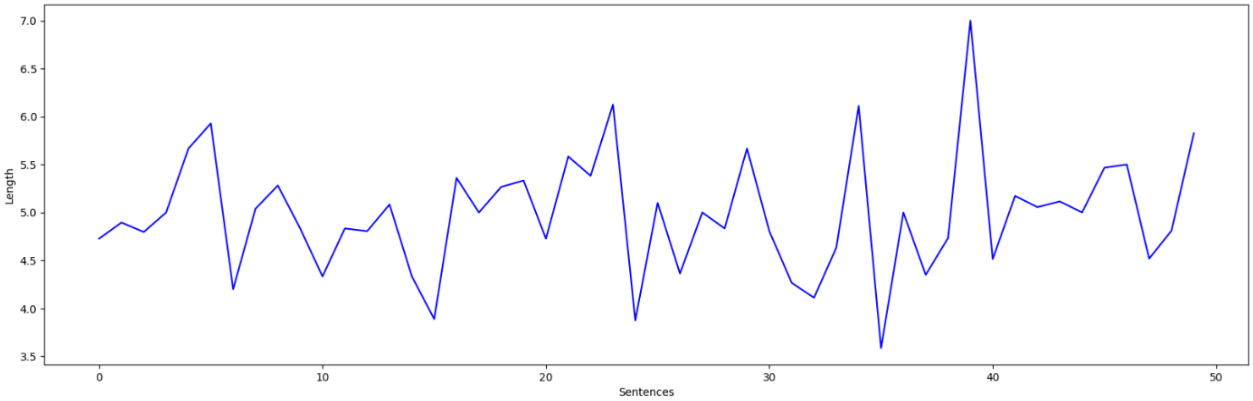


d. Mean Token Length

German:



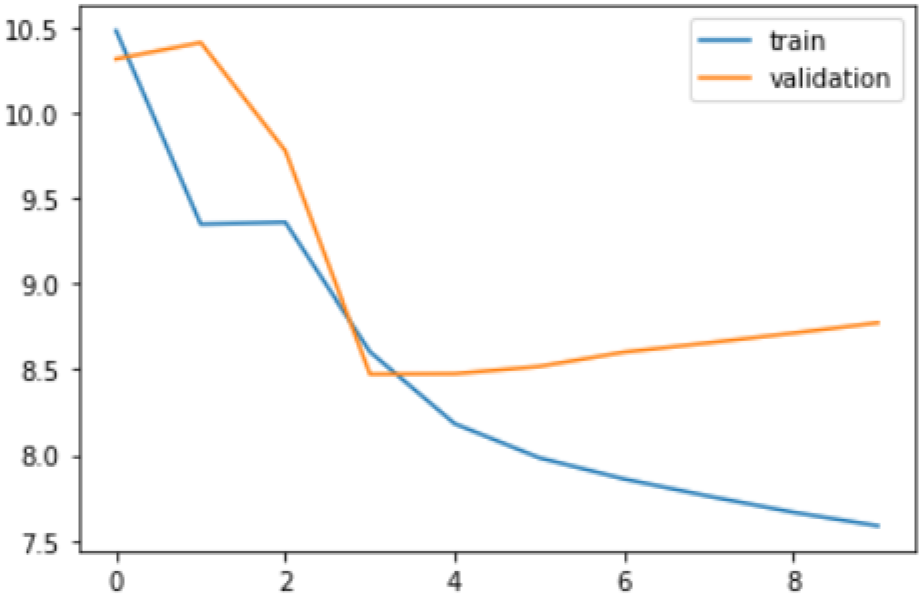
English:



Each of the train, val and test sets have their data in text, which are then encoded (Embeddings + Paddings) into vectors and then put into the network for processing.

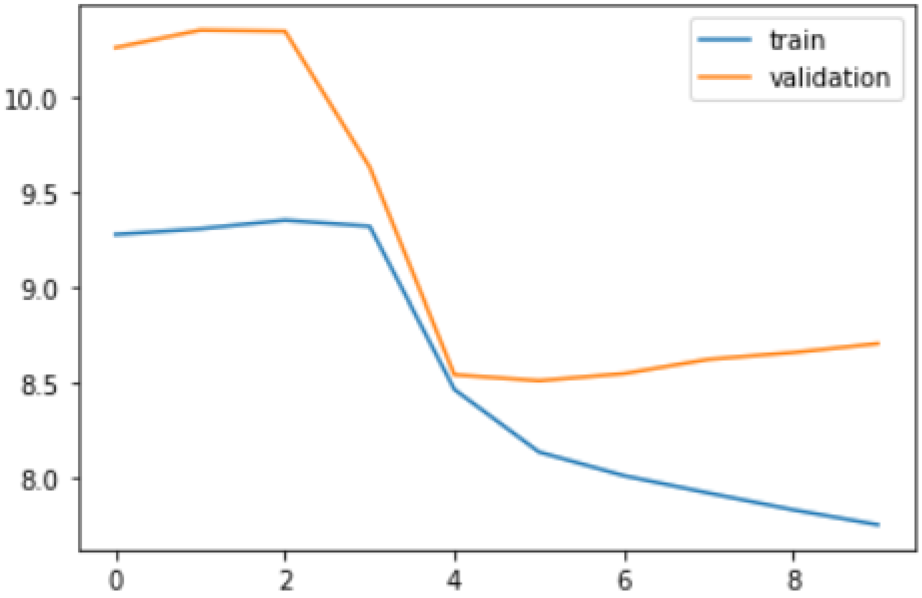
Each of the models have two LSTM layers, Adam as their optimizers, Sparse categorical cross entropy as their loss functions and run for 10 epochs over a batch size of 1024.

Model A: LSTM



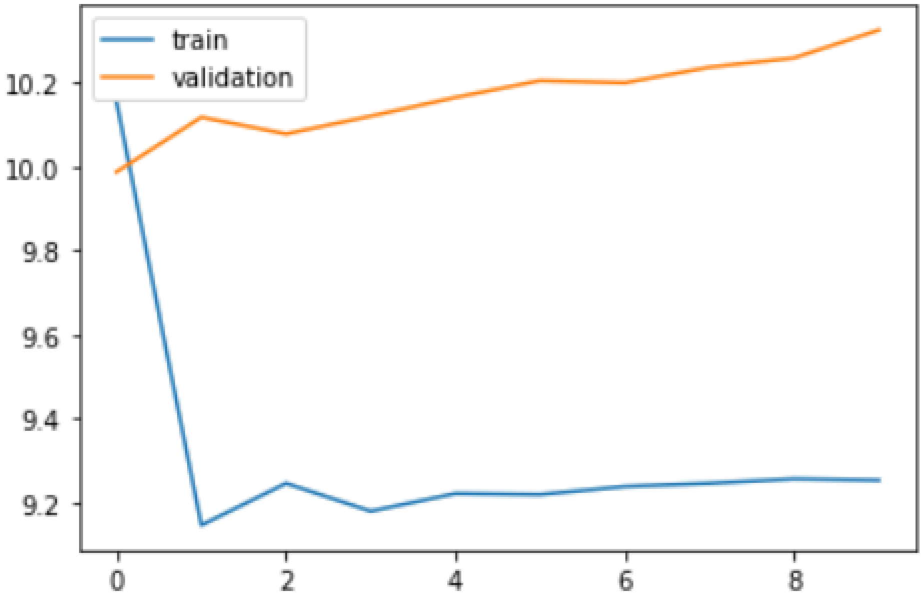
The training loss drops but the validation loss increases again after dropping, which suggests that the training needs to be more (or to have larger sample size).

Model B: LSTM + Global Attention



The results are almost similar to the basic LSTM model but it gives slightly better results on the validation set, and the loss reduces to ~8.7 after 10 epochs, which was ~8.77 in the case of the simple LSTM model.

Model C: LSTM + Local Attention



The results in the case of local attention are not better than Global Attention model, but are better than the simple LSTM model. The stabilizing train loss and the increasing val loss depicts that the training done in this case is not enough. The val loss at the end of 10 epochs is ~10.32.

Conclusion:

The LSTM + Global Attention model performs better than the LSTM + Local Attention model, followed by the simple LSTM model. This is largely due to the fact that Global Attention takes into account all the words in a particular sequence for attention, where the Local Attention might miss out on some of the important words in a sequence.