Indian Institute of Technology Madras

CS6910 Deep Learning

Assignment 4

Vimal Suresh Mollyn ED17B055
Dyava Naveen Reddy ME17B140
Ramakrishnan NA18B030

May 16, 2022

Vimal Suresh Mollyn, ED17B055
Dyava Naveen Reddy, ME17B140
Ramakrishnan, NA18B030

Contents

1 Task 1: Machine Translation Using Transformers

In this task, the objective is to build a neural machine translation model that can translate between english to tamil. For this, we use the publically available *Samantar* dataset which contains about 5 million corresponding pairs of tamil and english sentences. We follow the same architecture for the transformer as proposed by Vaswani et al in *Attention is all you need*, except for the modification that we use 3 transformer layers instead of 8, in both the encoder and the decoder.

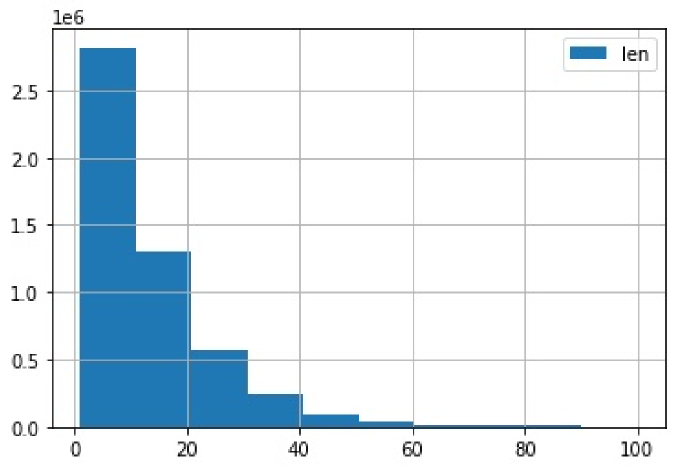

Figure 1: distribution of the number of words per sentence

1.1 Transformers

Transformers are a sequence to sequence model that makes use of the *attention* mechanism for sequence modelling. In this assignment, we use the encoder-decoder transformer architecture with multi-head self attention for the neural machine translation task.

1.2 Data Preprocessing

The Samantar dataset has around 5 million sentence pairs between english and tamil. However, the distribution of sentence lengths is quite skewed, and as a results, we made a design choice to omit sentences that had more than 10 words in them. The choice of 10 words as the maximum is derived from another popular MT dataset know as the *Multi30k* dataset, containing about 30k sentence pairs between german and english.

1.2.1 Byte-Pair encoding

When we were building the vocabulary for this massive dataset, we realised that it was quite large. Furthermore, due to the source of the Samantar dataset, there exist a huge number of proper nouns in each sentence, thereby increasing the vocabulary size. One popular method to mitigate both these issues is to break down each word into subwords using a technique known as Byte-Pair encoding (BPE). In BPE, the most commonly occuring subwords are added to the vocab and this is repeated for a number of iterations - thereby drastically reducing the vocabulary size.
We made use of the *subword-nmt* library in order to train a BPE tokenizer. We set the max number of iterations and vocabulary size to both be 32000. We trained 2 separate BPE tokenizers - one for tamil and another for english.
After training BPE tokenizers, we used this to tokenize the sentences.

1.2.2 Normalization

The unicode characters for the indian scripts such as tamil contain many artifacts and as a result need to be normalized. We make use of the *indic-nlp-lib* library in order to normalize these artifacts.

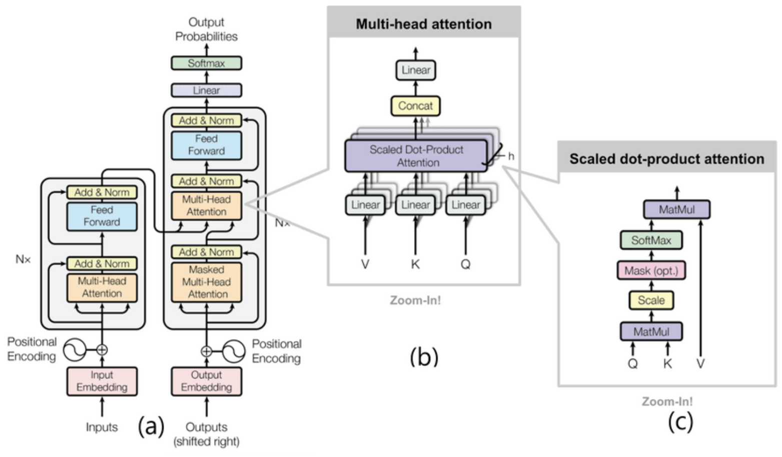

Figure 2: Architecture of the transformer model (Note that we use 3 layered encoders and decoders in our implementation

1.2.3 Further preprocessing

To aid with training, we also added special [CLS] (start), [SEP] (stop), <unk> and <pad> tokens to the vocabulary and the start and end of each sentence.

1.3 Model architecture

As stated earlier, we use the original transformer architecture proposed in the paper, *Attention is all you need* (Figure 2). We modify the number of encoder and decoder blocks and also tune the embedding dimension and the fully connected layer dimensions.

1.4 Training

We implemented the network using the pytorch framework. In particular, we utilized the official tutorial on nn.Transformer and torchtext (`https://pytorch.org/tutorials/beginner/translation_transformer.html`) for our code implementation. The embedding size and the fully connected layer size were hyperparameters that we played with.
For training we followed the same protocol as the original paper - using the Adam optimizer with a learning rate of 1e-4.
The training progress graphs are provided in figure 3.

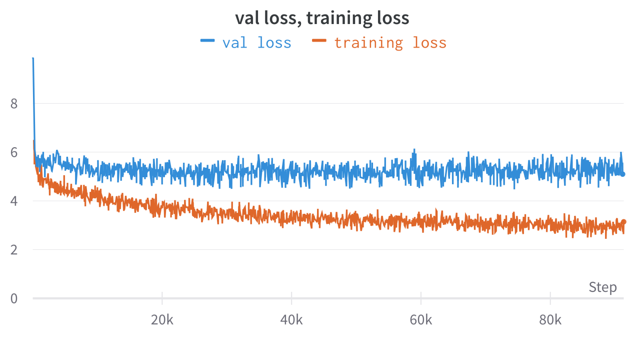

Figure 3: Training Progress for the machine translation task

1.5 Results

The results for the various configurations of the model are shown below in tabel 1.

For the metrics, we use the BLEU score, calculated using the *sacrebleu* library.

Hidden Dim Size	**Embedding Size**	**BLEU Score (val)**	**BLEU Score (test)**
512	512	7.0 (22.3/10.5/6.3/2.0)	7.0 (23.0/10.9/6.4/2.0)
2048	1024	6.1 (30.2/16.8/9.7/1.9)	6.4 (29.5/16.5/9.5/2.1)

Table 1: BLEU Scores for the Machine translation task for different hyperparameters. The verbose score is provided in parentheses (1-4 ngram order).

2 Task 2: Sentiment Analysis

Fine tuning the BERT model for sentiment analysis task

2.1 BERT

Bidirectional Encoder Representations from Transformers (BERT) is a transformer-based machine learning technique for natural language processing (NLP)
BERT was pretrained on two tasks: language modelling (15 percent of tokens were masked and BERT was trained to predict them from context) and next sentence prediction (BERT was trained to predict if a chosen next sentence was probable or not given the first sentence). As a result of the training process, BERT learns contextual embeddings for words. After pretraining, which is computationally expensive, BERT can be finetuned with less resources on smaller datasets to optimize its performance on specific tasks.

2.2 Description

- This section describes the implementation of fine tuning of BERT for sentiment classification task of text data into one of the five classes, reviews namely (1,2,3,4,5)
- We have used 5000, 1000, 1000 samples of training, validation and test data respectively
- The early stopping is set such that when the training loss stops decreasing we stop the training of model
- On anlaysing the length of word sin each review, it is observed that 90 percent of sentences are less than 160 words in length, thus we have chose the common sentence length to be 160
- We have used BERT tokenizer to tokenize the sentence into sequence of input ids and attention masks which is used as input to BERT model
- The training was done using a batch size of 16 (which we empirically found to be the max size that would run on our machine)
- We have used the BERT base uncased version of BERT model

- The BERT fine tuning architecture is as follows
  - Input of dimension (1, 160) of input ids and attention masks is passed to the pretrained BERT model
  - The pooled output from the BERT model is passed to a droput function with probability of dropout p = hp1
  - The output from dropout layer is passed to a linear layer with 5 nodes
- Loss function used : Cross-entropy loss
- Optimizer used : AdamW optimizer with learning rate (lr)
- Mode of training : Mini batch gradient descent
- Hyperparameters :
  - hp1 : dropout probability (p)
  - hp2 : learning rate (lr)

2.3 Observations

- We explored the following set of hyperparameters to find the best fit based on validation loss
- Dropout probability p : [0.3, 0.5]
- Learning rate : [2e-5, 2e-4, 2e-3]
- The following plots are the evolution of training loss and validation loss with epochs for different combination of hyperparameters

2.4 Plot of training loss and validation loss with epochs

Results for 0.3 dropout probability and 2e-5 learning rate

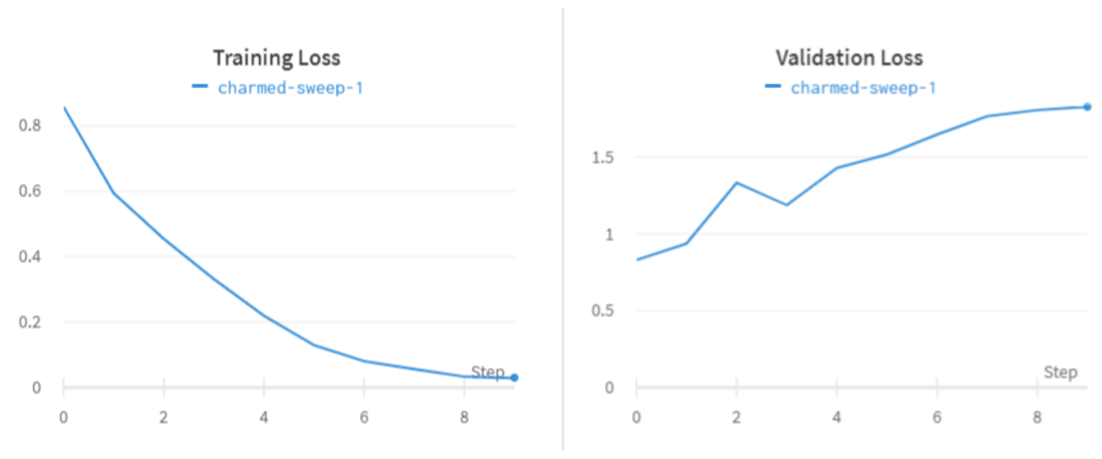

Figure 4: Average loss vs. epoch

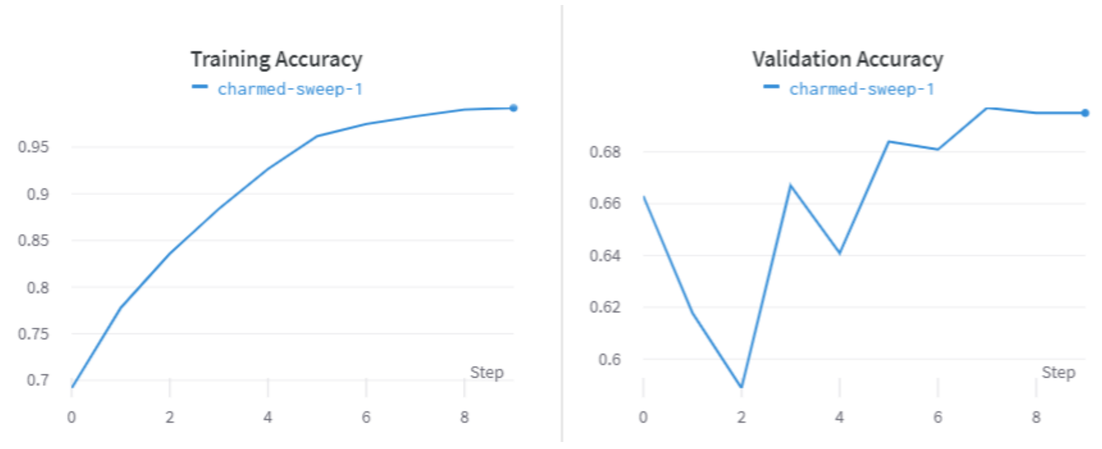

Figure 5: Accuracy vs. epoch

Results for 0.3 dropout probability and 2e-4 learning rate

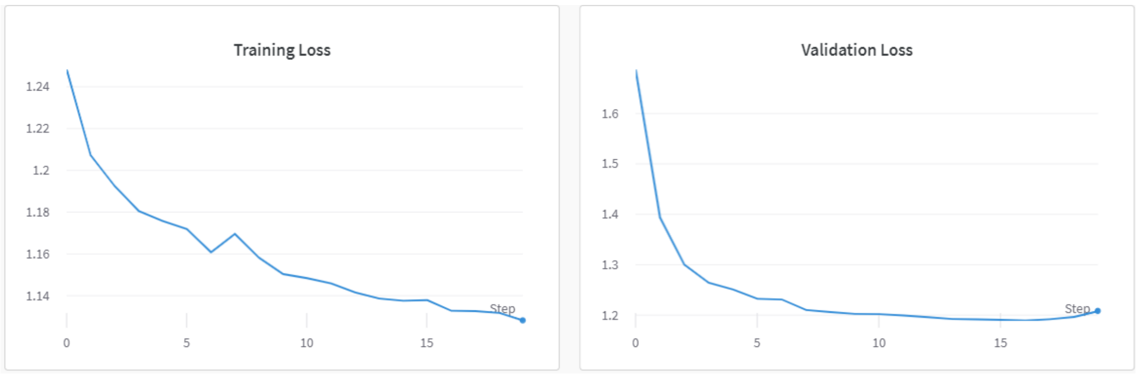

Figure 6: Average loss vs. epoch

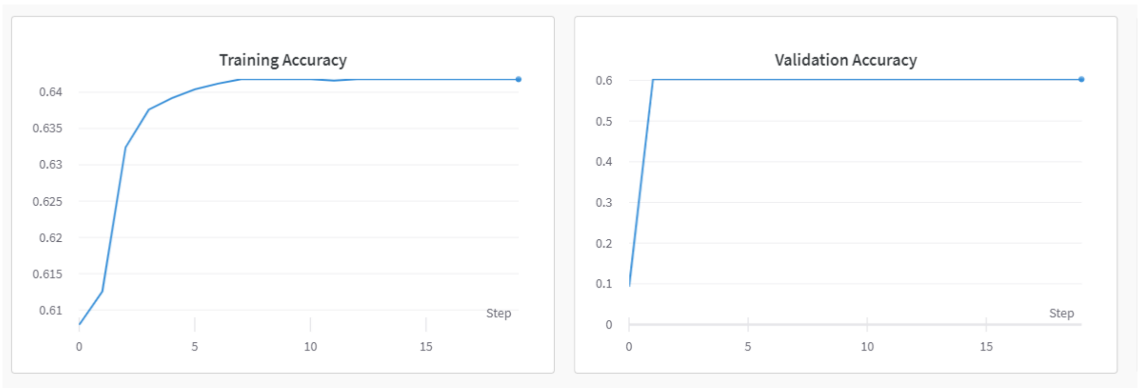

Figure 7: Accuracy vs. epoch

Results for 0.3 dropout probability and 2e-3 learning rate

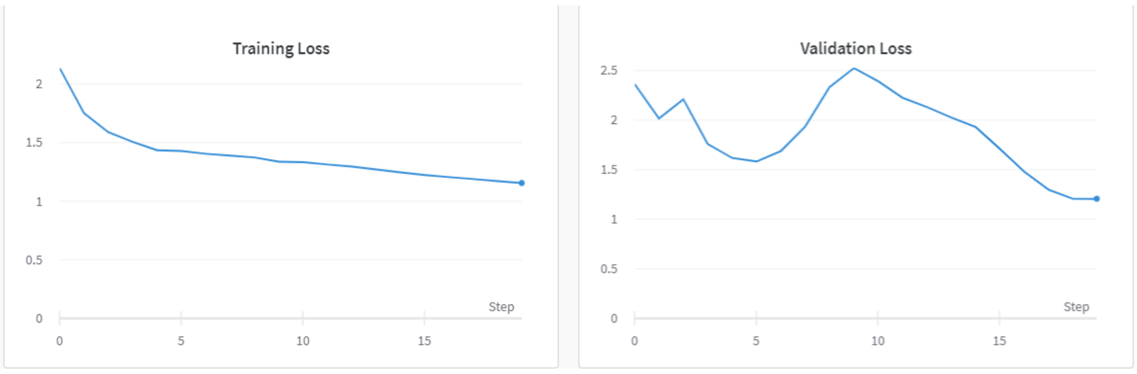

Figure 8: Average loss vs. epoch

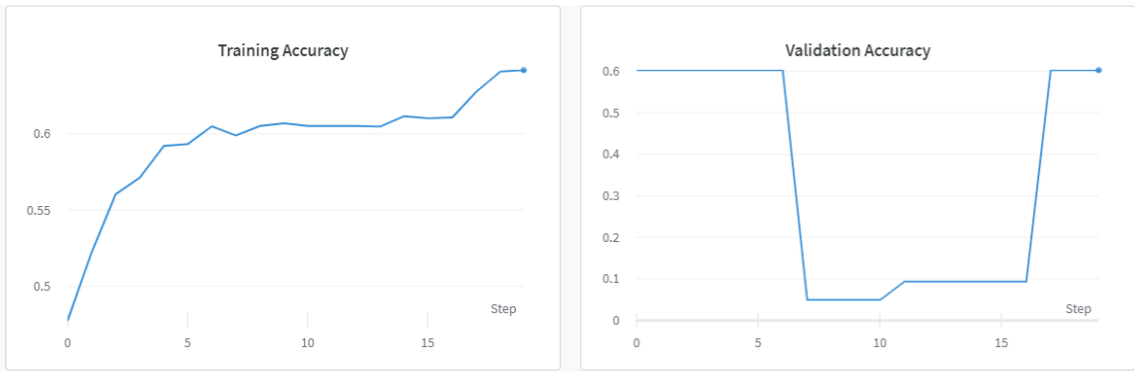

Figure 9: Accuracy vs. epoch

Results for 0.7 dropout probability and 2e-4 learning rate

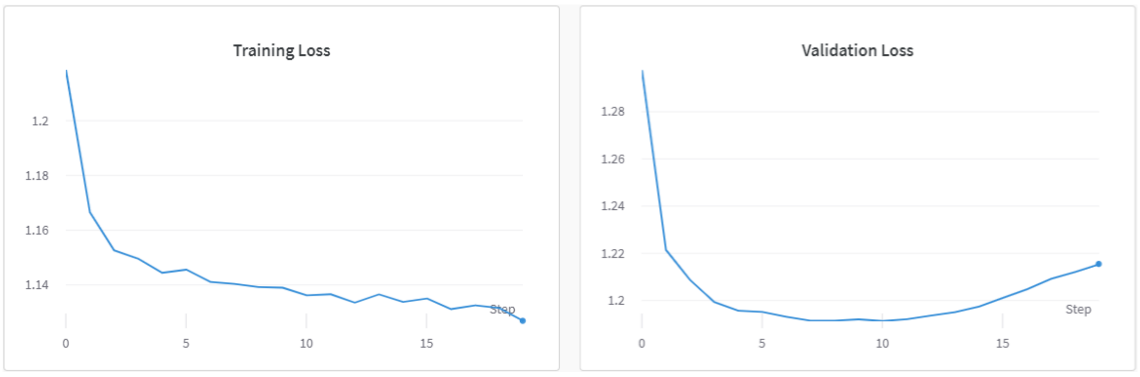

Figure 10: Average loss vs. epoch

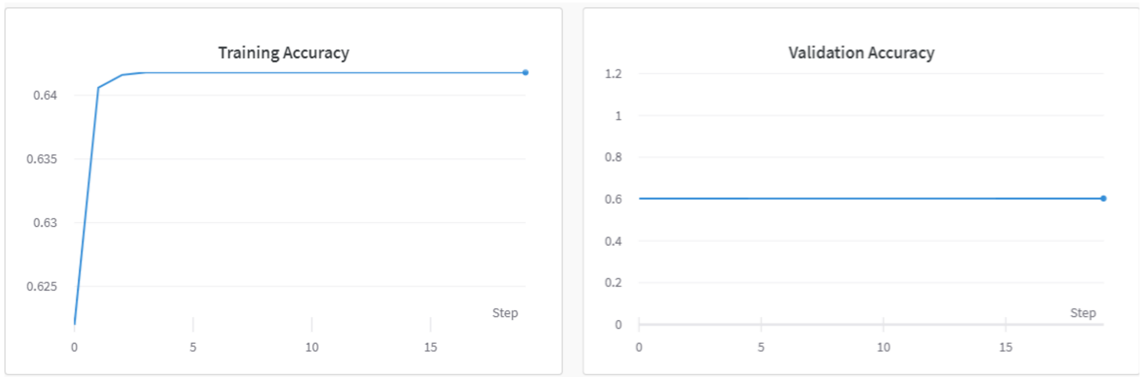

Figure 11: Accuracy vs. epoch

2.5 Summary

dropout p	lr	Train error	Train accuracy	Val error	Val accuracy	Epochs
0.3	2e-5	0.15	98	1.6	70	7
0.3	2e-4	1.15	65	1.2	60	10
0.3	2e-3	1.2	62	1.2	60	16
0.7	2e-4	1.12	64	1.1	60	10

2.6 Conclusion

Based on the observations we conclude that the following set of implementation gives the best result

- Dropout probability : 0.3
- Learning rate : 2e-5

Results:

- Training error : 0.15
- Training accuracy : 98
- Validation error : 1.6
- Validation accuracy : 70
- Test accuracy : 70

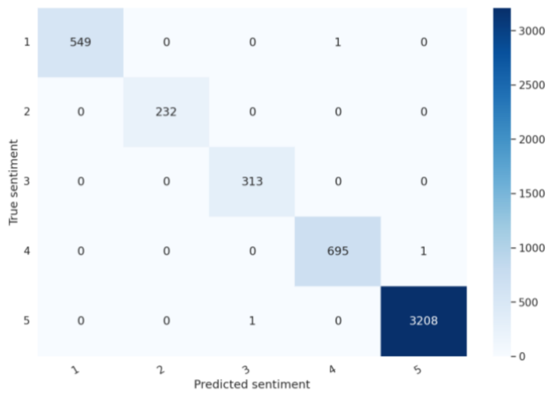

Figure 12: Confusion matrix on training data

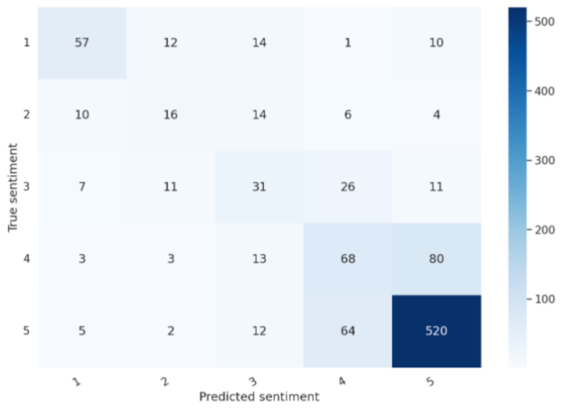

Figure 13: Confusion matrix on validation data

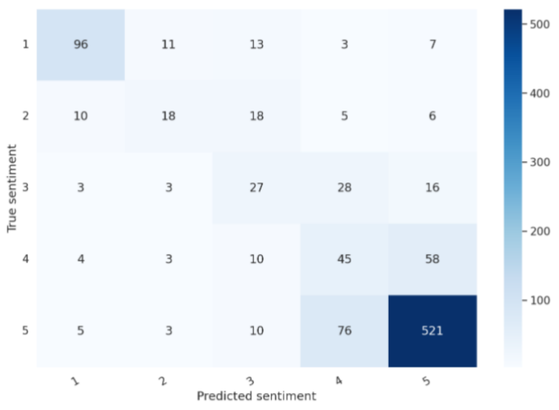

Figure 14: Confusion matrix on validation data

3 Task 3: Question Answering Task

Objective: In this task the objective is to fine-tune a pre-trained BERT Model for question answering. Given a paragraph and a question, the objective is to train a model which will return the position of the answer within the given paragraph. The paragraph is also called context. We will solve this by predicting the positions of start and end tokens corresponding to the answer inside the context.
Input: Question & Context.
Output: Positions of start & end tokens of the answer.

3.1 Details about the given dataset

Features: The given dataset contains context, question, answers, answer position, id and title.

- In the given dataset each context has multiple question answer pairs. We observed that all the questions for a given context had same answer. So we randomly selected only one question-answer pair for each context.
- All the features including the multiple question answer pairs per context are given as a single string per context. So we processed the given raw data using python regular expressions and made a final data while and saved in in .csv format.
- The final .csv file of the data contains Context, Question, Answer, Answer Position, id & title as features.
- One observations we made is that the answer position are not zero-indexed, they are indexed with 1 as starting position. According to the method we implemented, we have to use zero-indexing and we have converted the given start positions into zero-indexed fashion.

The table above shows the minimum, mean & the maximum number of characters in the Context, Question & the Answer of the given data.

Statistic	Context	Question	Answer
Min. Size	151	1	1
Mean. Size	683.5	60.0	17.7
Max. Size	3134	256	239

- Finally Context, Question, Answer & Answer Position are the features used for training and validation.
- The total number of data samples is 9749
- The data is split into train and validation in the ratio of 85:15 respectively
- The number of samples in the training data is 8286
- The number of samples in the validation data is 1463
- The training and test data are randomly sampled with a random seed = 21

3.2 Data Pre-Processing

We are going to use a pre-trained **bert-base-uncased** model provided by hugging face transformers. We can't pass the text directly into the model, we have to tokenize the text and convert it to corresponding format which will be taken as an input by the the bert-base-cased model. The bert-base-cased models has its corresponding tokenizer, which is as pre-trained tokenizer. This pre-trained tokenizer already has token representations of [CLS], [PAD] and [SEP] tokens already built into them. We pass the text into the tokenizer as two separate strings. In between they get combined to form a strings as shown below.

$$[CLS]\ question\ [SEP]\ context\ [SEP]$$

The string gets further tokenized and gets converted into token representations corresponding to the data on which the tokenizer was trained on. The tokenizer will output 3 main types of tensors, they are input ids, token type ids & attention masks. The input ids are the corresponding main input to the bert model. The toke type ids are the labels are segment embeddings of the the combined question and answer string. The attention masks has labels which represents weather the model should process the tokens at those given positions or not. Whenever we batch the inputs, we are going to use [PAD] tokens to represent the padding. The attention mask contains value "0" corresponding to the padding tokens.
Along with inputs, we also have to pass the start position and end positions, because the are necessary for calculating losses and validation metrics.
The bert-base-cased model in the hugging face framework has a maximum input size (question + context) of 384. But the question + answer length can exceed the size

of 384. In such cases if the answers contain in the truncated part we will not be able to train properly. In order to mitigate this problem we have to construct the dataset in such a way that the overflowing tokens can be used to construct new examples of the data. But we can't directly break the data, we have to use certain stride value and allow overlapping tokens between these examples because the answer might be present in the context segments of two broken contexts. The stride value allows the overlap and ensures the answer is presents in at least one of these examples. We have used stride value of 128. This decision was made after analyzing come statistics about the lengths of questions, answers and context lengths. These statistics can be seen in the previous table.
Since we created dataset by allowing overflowing tokens to form new examples, we also have to maintain offset mappings to map the corresponding token position in a given example to the position in the original input string. Since a given long input is getting broken up into pieces the start and end positions also change correspondingly. Whenever an answer is completely present in the given example we modify the start and end position values correspondingly. But whenever the answers is not completely present then we assign both the start and end position to the position of [CLS] token, i.e., 0 & 0. Note that we only truncate and allow overflowing tokens on context part of the input, we don't do it on the question.
Below is an example of how a long input string is split up into pieces using overflowing tokens method.
For example below is an input whose size exceeds the max size of the model:
Original Input:
'[CLS] To whom did the Virgin Mary allegedly appear in 1858 in Lourdes France? [SEP] Architecturally, ' 'the school has a Catholic character. Atop the Main Buildingś gold dome is a golden statue of the Virgin ' 'Mary. Immediately in front of the Main Building and facing it, is a copper statue of Christ with arms ' 'upraised with the legend " Venite Ad Me Omnes ". Next to the Main Building is the Basilica of the Sacred ' 'Heart. Immediately behind the basilica is the Grotto, a Marian place of prayer and reflection. It is a ' 'replica of the grotto at Lourdes, France where the Virgin Mary reputedly appeared to Saint Bernadette ' 'Soubirous in 1858. At the end of the main drive (and in a direct line that connects through 3 statues ' 'and the Gold Dome), is a simple, modern stone statue of Mary. [SEP]'

By using overflowing tokens method the following inputs are created:

Final Inputs:
1. [CLS] To whom did the Virgin Mary allegedly appear in 1858 in Lourdes France? [SEP] Architecturally, the school has a Catholic character. Atop the Main Buildingś gold dome is a golden statue of the Virgin Mary. Immediately in front of the Main Building and facing it, is a copper statue of Christ with arms upraised with the legend " Venite Ad Me Omnes ". Next to the Main Building is the Basilica of the Sacred Heart. Immediately behind the basi [SEP]
2. [CLS] To whom did the Virgin Mary allegedly appear in 1858 in Lourdes France? [SEP] the Main Building and facing it, is a copper statue of Christ with arms upraised with the legend " Venite Ad Me Omnes ". Next to the Main Building is the Basilica of the Sacred Heart. Immediately behind the basilica is the Grotto, a Marian place of prayer and reflection. It is a replica of the grotto at Lourdes, France where the Virgin [SEP]
3. [CLS] To whom did the Virgin Mary allegedly appear in 1858 in Lourdes France? [SEP] Next to the Main Building is the Basilica of the Sacred Heart. Immediately behind the basilica is the Grotto, a Marian place of prayer and reflection. It is a replica of the grotto at Lourdes, France where the Virgin Mary reputedly appeared to Saint Bernadette Soubirous in 1858. At the end of the main drive (and in a direct line that connects through 3 [SEP]
4. [CLS] To whom did the Virgin Mary allegedly appear in 1858 in Lourdes France? [SEP]. It is a replica of the grotto at Lourdes, France where the Virgin Mary reputedly appeared to Saint Bernadette Soubirous in 1858. At the end of the main drive (and in a direct line that connects through 3 statues and the Gold Dome), is a simple, modern stone statue of Mary. [SEP]

3.3 Model for Question Answering

The **bert-based-uncased** model is used. It has 12 transformer layers, hidden dimension as 768 and 12 heads and approximately 110 million parameters. The hidden outputs of the final layer of the bert morel are passed to two linear layers output layers, one to calculate the start position and the second one to calculate the end position. The outputs of the linear layers are logits for respective start and end positions. These logits are passed into a softmax layer to calculate the start and end positions.

Loss Calculation: Cross-entropy loss is used for calculating the losses incurred in predicting start and end token positions. We calculate cross entropy loss for both start and end tokens separately and we average the both of them. The averaged value of losses is. treated as the final loss of the model. The model is then trained through back-propagation to update the weights of the model.

Experiments: We have done mainly two experiments. In the first experiment the weights of the BERT Model are frozen and only the final linear layers are trained. In the second experiments the weights of the bert model are not frozen and the model is trained end-to-end. The results of both the experiments will be presented in the results section.

3.4 Metrics:

EM-Score: For each question+answer pair, if the characters of the model's prediction exactly match the characters of (one of) the True Answer(s), EM = 1, otherwise EM = 0. This is a strict all-or-nothing metric; being off by a single character results in a score of 0. When assessing against a negative example, if the model predicts any text at all, it automatically receives a 0 for that example.

F1 Score: F1 score is the harmonic mean of precision and recall. It is appropriate when we care equally about precision and recall. In this case, it's computed over the individual words in the prediction against those in the True Answer. The number of shared words between the prediction and the truth is the basis of the F1 score: precision is the ratio of the number of shared words to the total number of words in the prediction, and recall is the ratio of the number of shared words to the total number of words in the ground truth.

Loss: In this task Cross-Entropy loss as a loss. The average of loss of predictions of start index and the end index is used as a loss value.

3.5 Model Training:

- No. of encoder layers in the BERT model = 12
- Hidden size of transformer = 768
- Self-attention heads = 12
- No. of Linear layers at output = 2 {BERT -> Start token layer & BERT -> End token layer}
- No. of trainable parameters = 110 Million
- Batch size used = 16
- early stopping -> min delta = 0.1 & patience = 10
- max epochs = 30
- Optimizer = Adam { epsilon = 1e-08}
- Default Initialization of parameters in PyTorch

3.6 Results

The metrics like EM Score & F1 Score are calculated on the validation data.

Metric	BERT Frozen Model	End-to-End trained Model
Exact Match Score	1.84	88.37
F1 Score	10.08	94.41
Validation Loss	5.06	0.3267
Training Loss	5.039	0.09564
No. of epochs	30	13

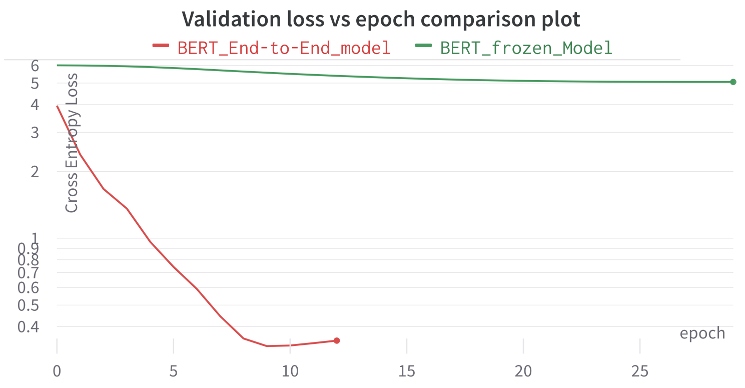

Figure 15

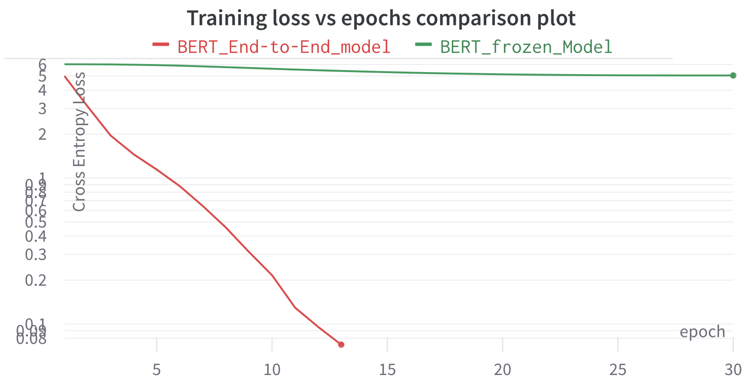

Figure 16

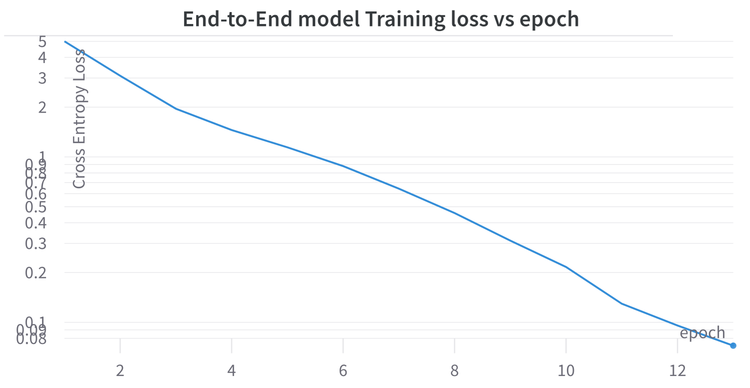

Figure 17

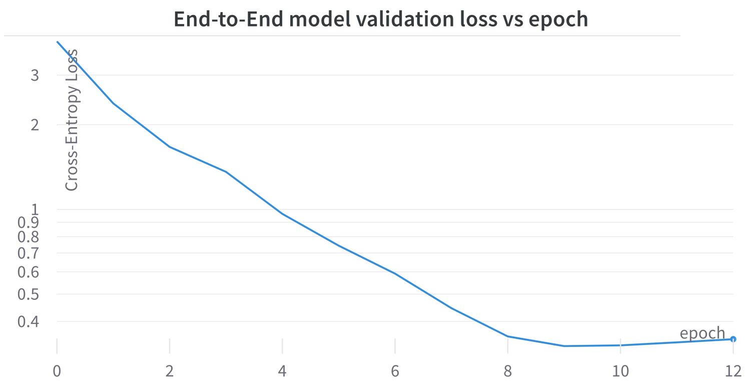

Figure 18

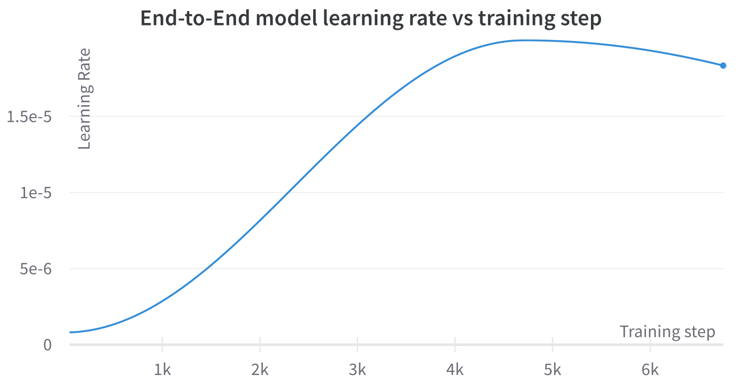

Figure 19

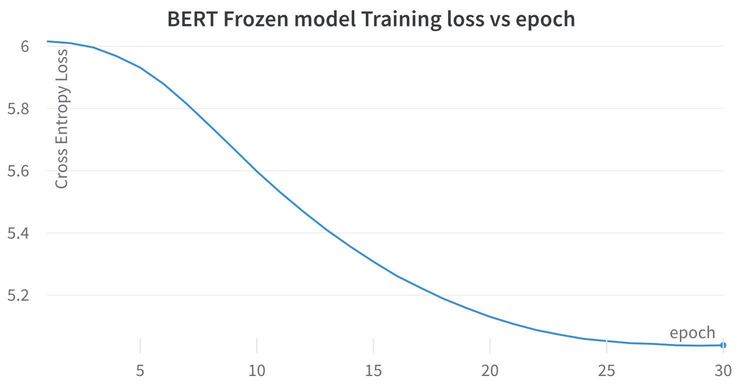

Figure 20

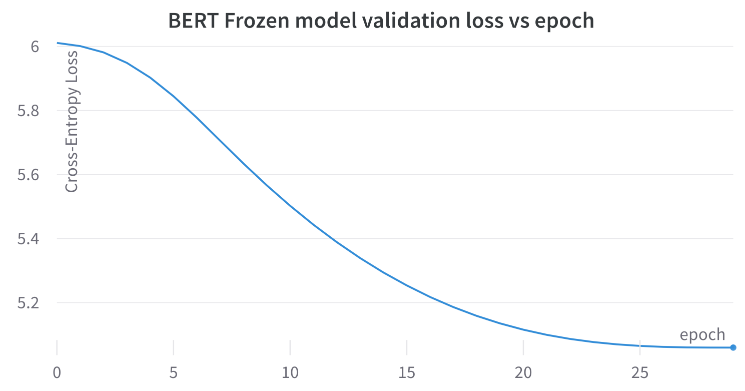

Figure 21

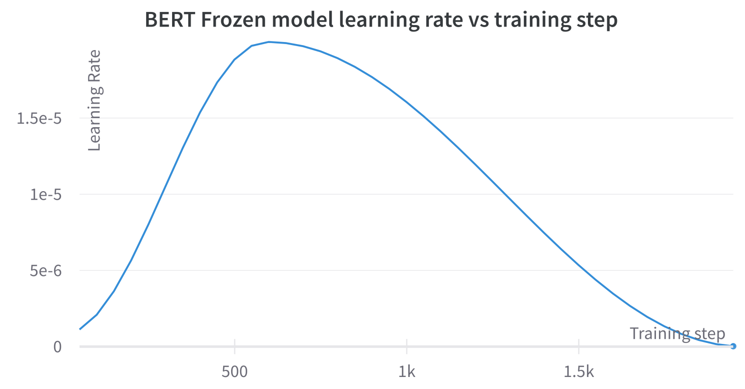

Figure 22

3.7 Conclusion

From the results presented in the previous section it can be observed that the BERT model which is End-to-End trained on the dataset is having superior performance compared to the model in which only the final linear layers are trained. We have a final Exact Match score of 88.37 & F1 score of 0.9441. This is an indication of very good perfomance of the model.