Introduction

The paper I read was “Efficient Passage Retrieval with Hashing for Open-domain Question Answering” (Yamada et al., 2021). In this paper they explored representing passages as binary representations instead of vector embeddings, this method is better known as Binary Passage Retrieval (BPR). By representing the passages as binary representations we require significantly less space to store passages, also it is computationally cheaper, thus allowing us to produce answers much faster. This architecture was built on top of the traditional Dense Passage Retriever structure (DPR). By utilizing a “retriever - reader” style architecture the authors of this paper were able to create a Q&A system that was more light weight and maintained the same accuracy as previous DPR models.

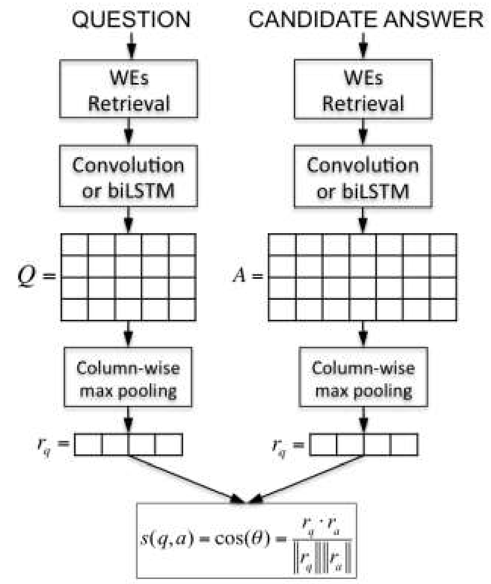
This idea of representing passages and answers as binary representation was first developed by XU et al., (2020). In their “Hashing Based Answer Selection” paper they stated that since pooling techniques have shown to be less accurate than the interaction mechanisms, such as DPR, they would disregard the use of binary representations in pooling techniques. However, they only looked at Attentive Pooling (AP) (Santos et al., 2016), which relies solely on glove word2vec embedding and simple CNN or Bi-LSTM structure. For this assignment I intend to modify the structure such that it utilizes a BERT tokenizer and a pre-trained BERT embedding. Also the soft logit layer of the sentence embedding will be based on a Bi-Directional GRU or LSTM model.

Why bother increasing the accuracy of pooling techniques? Pooling techniques are better at reducing the possible answer space thereby they require less space than DPR. If we are able to improve the accuracy of pooling QA approaches then we can further decrease the space needed to store passages by applying BPR techniques to pooling approaches.

Method

Motivation

The main objective of my model is to increase the accuracy of current pooling techniques for open domain questions and answering. Utilizing temporal pooling allows us to capture the most relevant temporal features in a sequence of words, this allows us to better understand context and relationship between words. Currently pooling techniques perform worse than interaction techniques because they operate on fixed-size windows and might not capture contextual information effectively. For instance, Figure 1, shows a traditional pooling technique. It is evident that this model limits the interaction between the question and answer, thereby generating results that are less than ideal. To overcome this issue I intend to utilize a Distilled BERT tokenizer and a fine tuned Distill Bert embedding, that embed the passage and question simultaneously.



<u>Figure 1:</u> *Q&A model that utilizes pooling technique.*

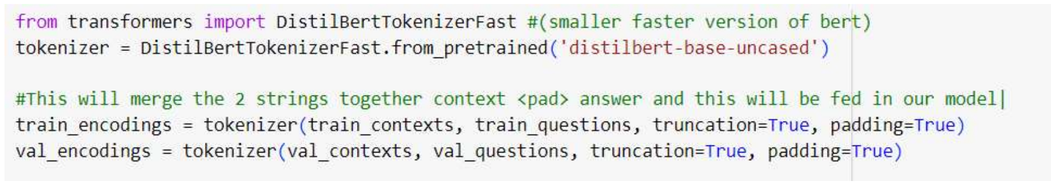
In my proposed model I will take advantage of the distilled BERT fast tokenizer and a fine tuned Q&A Bert tokenizer. This will provide me with a better vector embedding than the one described in the attentive pooling paper (Santos et al., 2016). Also, I will train this model independently of the other models such that I prevent overfitting. Next I will train a bi-directional GRU/LSTM using a fine tuned TriviA QA distilled BERT model. However, before passing the output of the BERT model to the GRU I will take the softlogit layer of the BERT model and apply a max pooling. This will reduce the answer space, thus saving space and increasing the speed of the QA model. This will increase the accuracy of the pooling approach while also requiring less memory than the traditional DPR approach.

Implementation

Step 1) Tokenization

The first crucial part of the model is tokenizing the questions and answers, to do this I used a pre-trained *DistilBertTokenizerFast*. This differs from the previous approach's use of a Glove

Word2Vec embedding. Using a BERT tokenizer allows us to better generalize the questions and answers. It also provides us with more features, for instance it provides us the start and end index of the answer passage within the text. Lastly, by tokenizing the question and answer together we increase the impact the question has on the model's prediction of the answer.



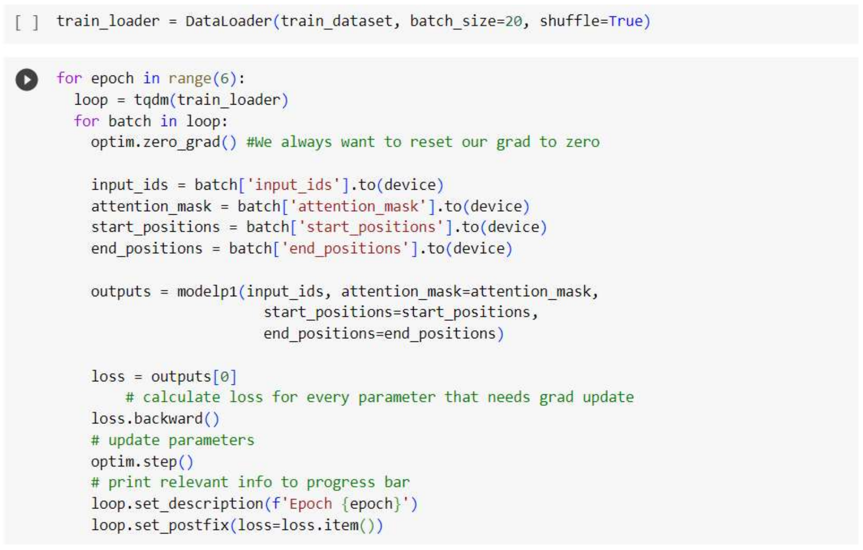
```
from transformers import DistilBertTokenizerFast #(smaller faster version of bert)
tokenizer = DistilBertTokenizerFast.from_pretrained('distilbert-base-uncased')

#This will merge the 2 strings together context <pad> answer and this will be fed in our model|
train_encodings = tokenizer(train_contexts, train_questions, truncation=True, padding=True)
val_encodings = tokenizer(val_contexts, val_questions, truncation=True, padding=True)
```

Figure 2: *Toeknization technique*

Step 2) Sentence Embedding

For this particular model we will be training and testing the data on the Trivia QA dataset. I went with a fine tuned Distilled BERT vector embedding, this is because using a BERT embedding provides a positional token embedding and segment embedding. BERT has also been proven to be one of the better embedding techniques and is more efficient than the one used in the Santos et al., (2016) papper.

Figure 3, shows how we will be training this model, first we will pass in the *input_ids'*, *attention_mask, start* and *end* position of the relevant spans of text. In this case the *input_ids'* represents the question and the *attention_mask* represents the answer. This model will then output a sentence embeddings that we can then use to predict the loss and update the model parameters as needed. We are training this portion of the model independently such that it can correctly identify question and answer pairs independently. Also, this code was configured in a way that the batch size of the *DataLoader* indicated how many passages you want to return, top k-passages.



```
[ ] train_loader = DataLoader(train_dataset, batch_size=20, shuffle=True)
```

```
for epoch in range(6):
  loop = tqdm(train_loader)
  for batch in loop:
    optim.zero_grad() #We always want to reset our grad to zero

    input_ids = batch['input_ids'].to(device)
    attention_mask = batch['attention_mask'].to(device)
    start_positions = batch['start_positions'].to(device)
    end_positions = batch['end_positions'].to(device)

    outputs = modelp1(input_ids, attention_mask=attention_mask,
                      start_positions=start_positions,
                      end_positions=end_positions)

    loss = outputs[0]
        # calculate loss for every parameter that needs grad update
    loss.backward()
    # update parameters
    optim.step()
    # print relevant info to progress bar
    loop.set_description(f'Epoch {epoch}')
    loop.set_postfix(loss=loss.item())
```

Figure 3: *Fine Tuned Bert*

Step 3) Pooling and Bi-GRU

Inorder to overcome the limitations of using a single distilled BERT model we will be using a temporal pooling technique on the soft logits of the previous model. The results of this pooling will then be passed to a bi-diection GRU or LSTM model composed of 50 RNN layers. Soft logits refer to the layer right before the sentence embedding, *Figure 4.* Applying max pooling over time, or temporal pooling, allows us to capture the most relevant features in a sequence of words, thereby increasing the overall effectiveness of the sentence embedding. Which will increase the overall accuracy of polling approaches in open domain QA.

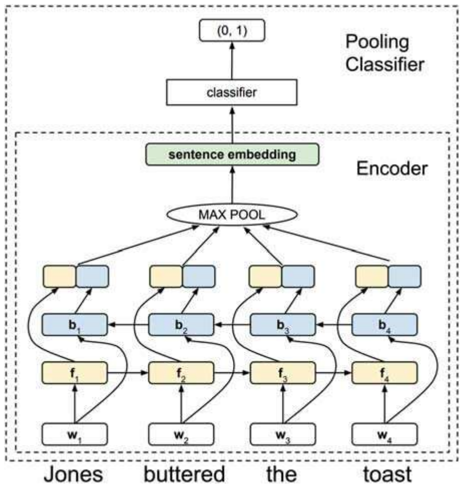


Figure 4: BERT Encoder with Pooling

Once we applied the pooling to the soft-ogits we will then pass this to the GRU model to create the new sentence embedding. By utilizing the GRU short and long term memory recall parameters we are able to generate more accurate sentence embeddings.

Evaluation:

Retriever	Top-20 (Trivia QA) Accuracy
Binary Passage Retrieval (BPR)	79.4
Dense Passage Retrieval (DPR)	79.4
Attentive Pooling (AP)	68.8
My model (Bert-GRU pooling)	50.2
My model (BertLSTM pooling)	50.8

Table 1.0: *Accuracy results for triviaQA dataset*

For all these models we choose a k value of 20, meaning that we returned the top 20 passages retrieved by the corresponding model. From analyzing the results in *Table 1.0* it is evident that DPR and BPR perform better than the attentive pooling techniques and the pooling technique that I created. Also, the attentive pooling approach by santos et al., (2016) out performed my proposed pooling approach. Despite the drop in accuracy this model does require significantly less computational resources than that of the pre-existing models.

However after analyzing these results it is evident that the original hypothesis of coupling BERT token and sentence embeddings to increase the performance of pooling is not valid. Despite that it is still possible to improve pooling techniques with BERT if we encode the passage and answer separately and apply a similar attentive pooling method (santos et al., 2016), although this has not been done as of now, it is definitely something to consider moving forward.

Lastly, a contributing factor to this model's failure is the lack of computational resources provided. Due to limited computational resources I was unable to train the data on the whole dataset thereby increasing the probability of overfitting the model. To circumvent this issue, I initially applied an early stopping approach however this resulted in a similar accuracy. Therefore, I experimentally determined that the ideal amount of epochs needed was 8.

Discussion:

Due to limited computational resources I was unable to utilize more powerful models such as BERT or a more complex pooling technique. Thus the speed at which the model was able to process questions was less than ideal. As seen in the previous section this model did not outperform either the BPR or the Attentive pooling approach.

However, this model does have advantages over the other models, primarily it is significantly more lightweight than the other models. Since it only utilizes a Distilled BERT tokenizer, fine tuned distilled BERT QA and a bi-directional GRU or LSTM, it requires significantly less computational resources than the other two models. Also, unlike the BPR model which requires a 2 step information retrieval process this model only requires one making it faster at retrieving relevant passages. Another advantage of this model is that it encodes passages and questions together so when we apply pooling the question directly impacts the answer. Unlike previous pooling approaches which performed pooling techniques on questions and answers independently this model applies them simultaneously. This interaction would ideally allow for more relevant spans of text to be identified within a passage.

Despite the model's advantages we see that for a K = 20 this model produces poor accuracy results when compared to the other models. This is most likely due to the fact that other models encode the passage and context separately thereby allowing a greater number of relevant passages to be retrieved. Also, since BPR uses a 2 phase IR approach they are able to further refine their retrieved passages which allows them to generate more meaningful and semantically similar results. For this reason I can conclude that their model outperformed mine.

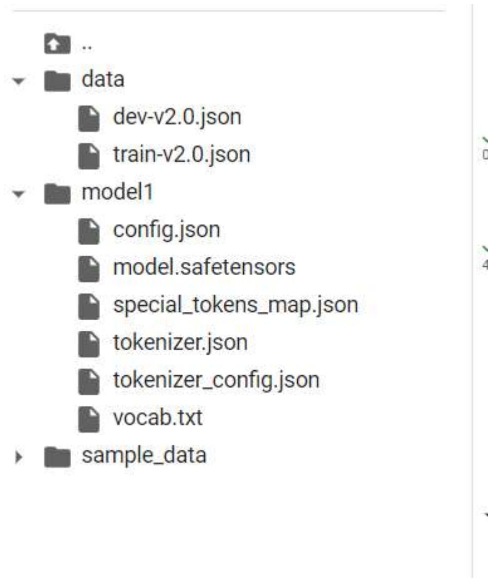
Conclusion

By utilizing a distilled BERT word embedding, coupled with a max pooling approach I hoped to increase the accuracy of the passages returned. However, when we ran the model and asked it to return 20 relevant spans of text and compared the returned results with the golden answer we were able to see that this model does not outperform the pre-existing BPR model. However, by utilizing more advanced pooling techniques and decoupling the question and answer embedding this model may indeed outperform the attentive pooling approach to open domain QA solutions.

Work cited

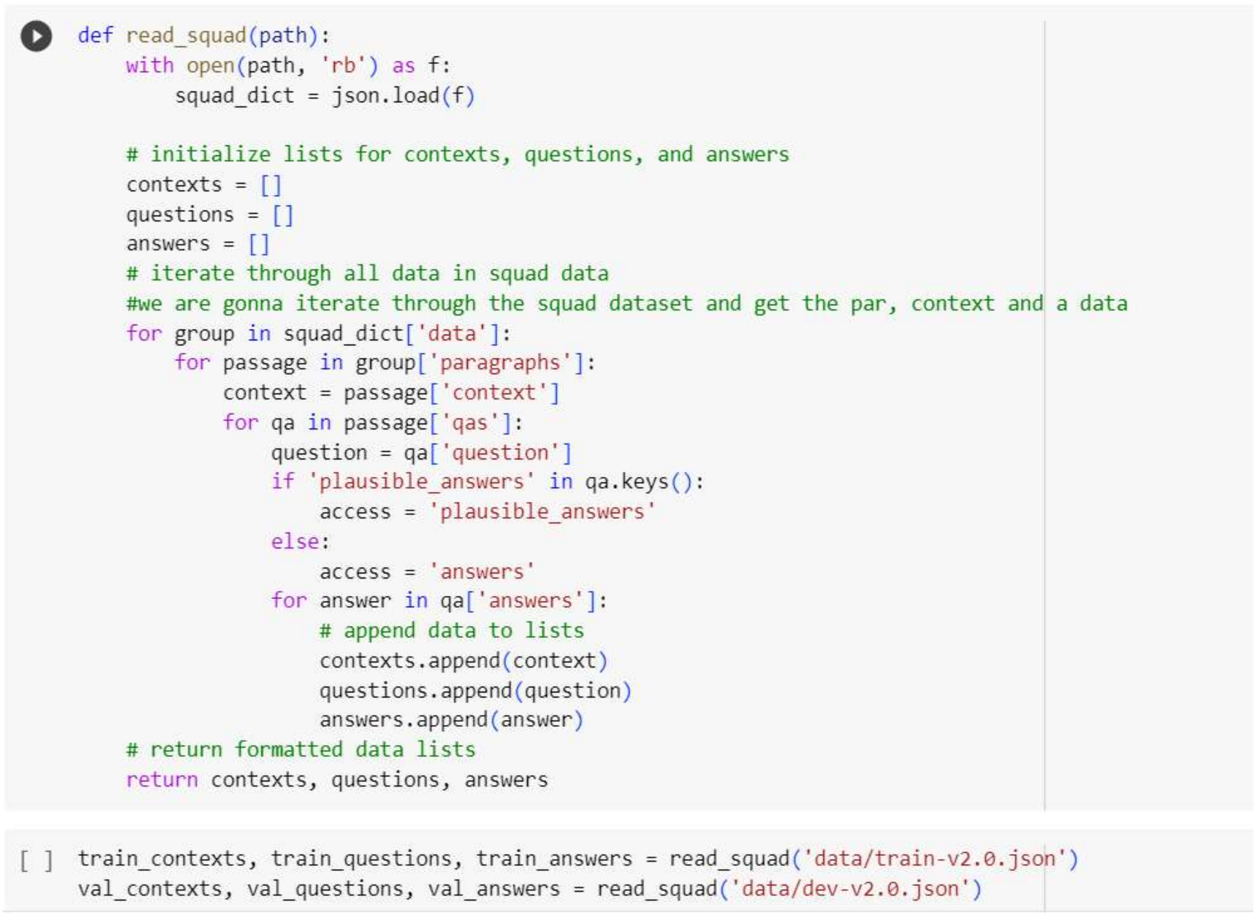
1. Danqi Chen, Adam Fisch, Jason Weston, and Antoine Bordes. 2017. Reading Wikipedia to Answer OpenDomain Questions. In Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 1870– 1879.
2. Xu, D., & Li, W. J. (2020, April). Hashing based answer selection. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 34, No. 05, pp. 9330-9337).
3. dos Santos, C., Tan, M., Xiang, B., & Zhou, B. Attentive Pooling Networks.
4. Yamada, I., Asai, A., & Hajishirzi, H. (2021, August). Efficient Passage Retrieval with Hashing for Open-domain Question Answering. In Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing.

Appendix:



```
..
data
    dev-v2.0.json
    train-v2.0.json
model1
    config.json
    model.safetensors
    special_tokens_map.json
    tokenizer.json
    tokenizer_config.json
    vocab.txt
sample_data
```

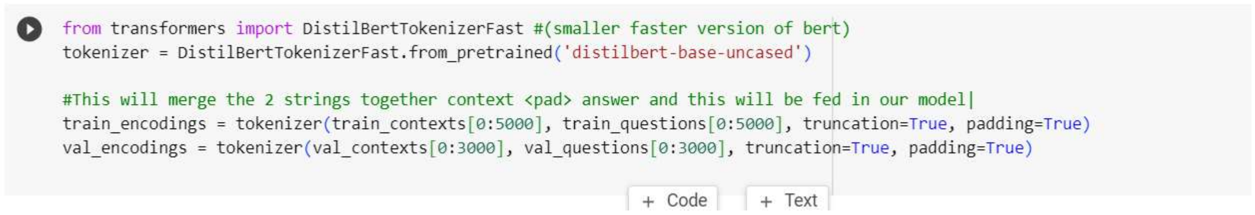
Figure 5: Data and fine tuned BERT model path



```
def read_squad(path):
    with open(path, 'rb') as f:
        squad_dict = json.load(f)

    # initialize lists for contexts, questions, and answers
    contexts = []
    questions = []
    answers = []
    # iterate through all data in squad data
    #we are gonna iterate through the squad dataset and get the par, context and a data
    for group in squad_dict['data']:
        for passage in group['paragraphs']:
            context = passage['context']
            for qa in passage['qas']:
                question = qa['question']
                if 'plausible_answers' in qa.keys():
                    access = 'plausible_answers'
                else:
                    access = 'answers'
                for answer in qa['answers']:
                    # append data to lists
                    contexts.append(context)
                    questions.append(question)
                    answers.append(answer)
    # return formatted data lists
    return contexts, questions, answers
```

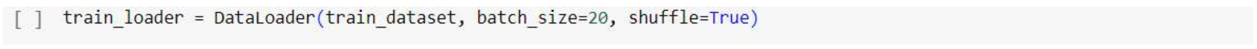
```
[ ] train_contexts, train_questions, train_answers = read_squad('data/train-v2.0.json')
    val_contexts, val_questions, val_answers = read_squad('data/dev-v2.0.json')
```

Figure 6: Read in the data set and split it to train and validate

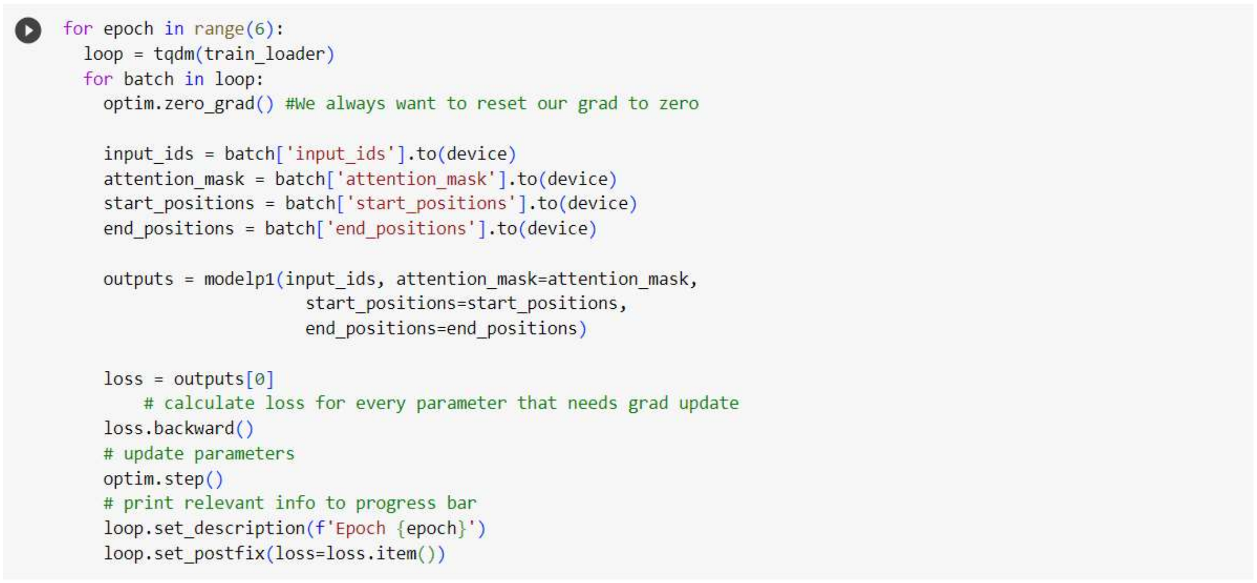


```
from transformers import DistilBertTokenizerFast #(smaller faster version of bert)
tokenizer = DistilBertTokenizerFast.from_pretrained('distilbert-base-uncased')

#This will merge the 2 strings together context <pad> answer and this will be fed in our model|
train_encodings = tokenizer(train_contexts[0:5000], train_questions[0:5000], truncation=True, padding=True)
val_encodings = tokenizer(val_contexts[0:3000], val_questions[0:3000], truncation=True, padding=True)
```

Figure 7: using distilled BERT to tokenize the question and context

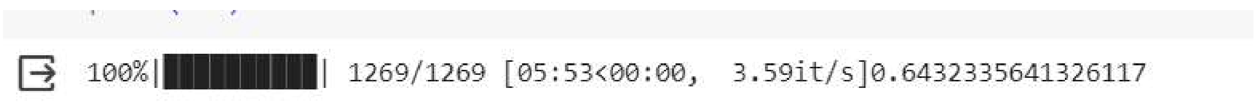


```
[ ] train_loader = DataLoader(train_dataset, batch_size=20, shuffle=True)
```



```
for epoch in range(6):
  loop = tqdm(train_loader)
  for batch in loop:
    optim.zero_grad() #We always want to reset our grad to zero

    input_ids = batch['input_ids'].to(device)
    attention_mask = batch['attention_mask'].to(device)
    start_positions = batch['start_positions'].to(device)
    end_positions = batch['end_positions'].to(device)

    outputs = modelp1(input_ids, attention_mask=attention_mask,
                    start_positions=start_positions,
                    end_positions=end_positions)

    loss = outputs[0]
        # calculate loss for every parameter that needs grad update
    loss.backward()
    # update parameters
    optim.step()
    # print relevant info to progress bar
    loop.set_description(f'Epoch {epoch}')
    loop.set_postfix(loss=loss.item())
```

Figure 8**:** Fine tuning BERT to the triviaQA dataset (NOTE: batch size equal numbers k-relevant passages)



```
100%|██████████| 1269/1269 [05:53<00:00,  3.59it/s]0.6432335641326117
```

Figure 8**:** Accuracy of model using SQUAD dataset and bi-LSTM