

Artificial Intelligence II - Project 4

Violetta Gkika

February 2022

Contents

1	Exercise 1	2
1.1	Tokenizer	2
1.2	Model Architecture	2
1.3	Learning rate	4
1.4	Epochs	4
1.5	Batch size	5
1.6	References	6
2	Exercise 2	8
2.1	General assumptions:	8
2.2	F1 calculation	8
2.3	Tokenizer	8
2.4	Learning Rate	8
2.5	Epochs	9
2.6	Batch size	9
2.7	Model Architecture	9
2.8	Experimenting	10
2.9	References	14
3	Exercise 3	15

1 Exercise 1

General information about the implementation:

For this exercise I used bert-base-uncased pretrained model from hugging face. The data preproces is the same as in all previous projects I chose a really simple architecture for my neural network consisting of only a dense layer (plus bert model and a dropout layer). The tweets are tokenized using the Bert tokenizer from transformers. The training loop follows the same steps as in the previous projects with the only difference that now, as I get the embeddings through the bert tokenizer, I use the returned input_ids and attention_mask as input on my model. I used Adam as optimizer since from the experimenting on the previous projects I had concluded that Adam worked always better in our dataset.

1.1 Tokenizer

As mentioned above I used bert tokenizer to tokenize the tweets using the following parameters:

padding='max_length': with the purpose to perform padding in the end of each tweet in order to get tweets with the same length.

max_length = 64: (maximum sequence length) There are two reasons I chose the length of the tweets to be 64. Firstly, as the memory space in colab (and kaggle) is limited for users using big sized tweets I could not increase the number of batches. For example with max_length = 512, the bigger number of batches I could run was 6 for colab (8 for kaggle). Also, after the preproces of the data I ended up with small sentenced tweets, that consists of 15-18 words so 64 tokens are more than enough.

truncation=True: To truncate the tweet in case there is a tweet greater than max_length = 64, which in our case is useless since all the tweets has less than 64 words but I kept it in case the test data has tweets longer than 64 words.

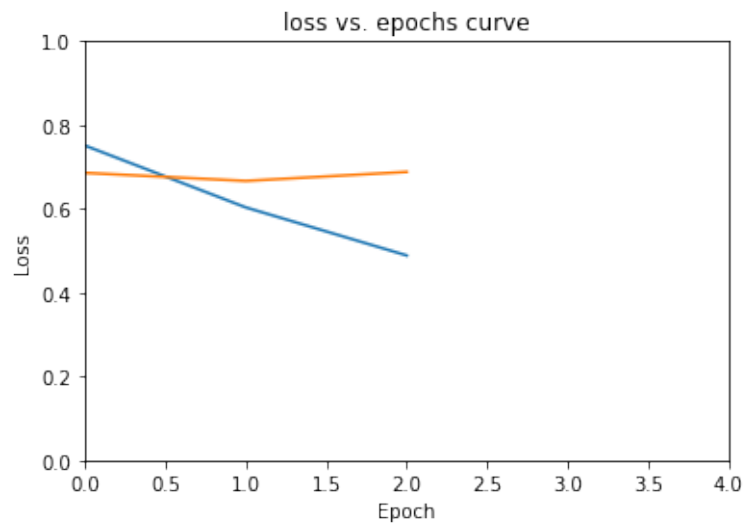
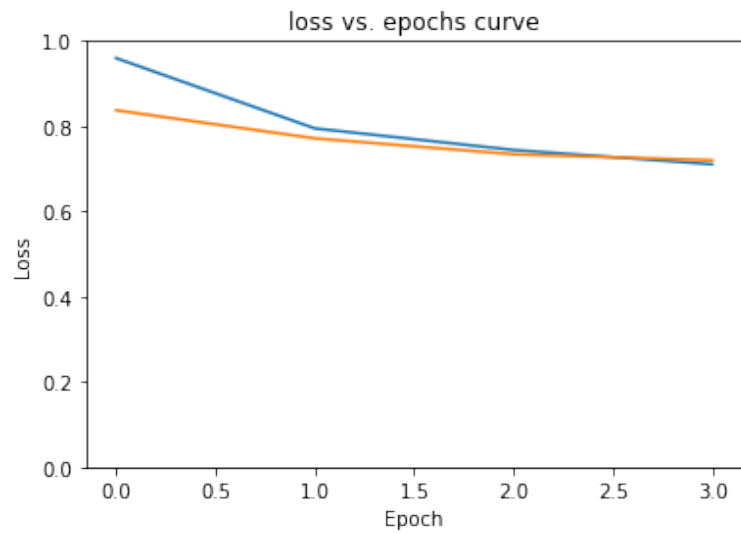
1.2 Model Architecture

As mentioned on the beginning I used a very simple architecture for the model consisting of just one additional output layer to create state-of-the-art models. The first "layer" is the pretrained BertModel which returns a vector of 768 dimensions, then for the classification process I use the embedding of the first token, which is the [CLS] token, (the output of [CLS] is inferred by all other words in the tweet, so [CLS] contains all information for the tweet) as input on the feed forward layer(Linear).

(I refer to the Bert as a layer as it is a full model consisting of other layers but

actually we can see it as a way of getting the embedding we need to feed to our layers).

I tried to use a dropout layer of 0.2 and 0.5.

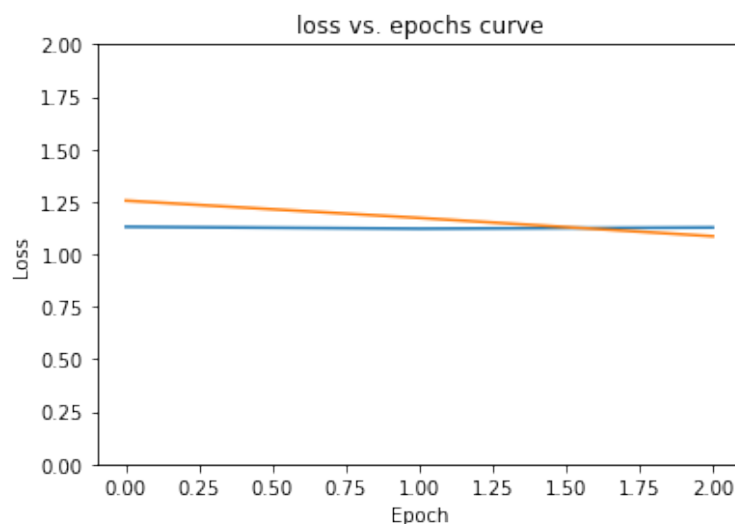


As you can see the dropout layer "helped": a lot the model in order to not overfi. I ended up using a dropout layer of 0.5 as you can see on the notebook.

1.3 Learning rate

In this project learning rate was the key hyperparameter. A fairly small learning rate worked better.

"Catastrophic forgetting is usually a common problem in transfer learning, which means the pre-trained knowledge is erased during learning of new knowledge. (McCloskey and Cohen,1989)". Based on this I experimenting with small learning rate, and as shown from the images below a large learning rate did not give good results.



LR=0.005 - Catastrophic forgetting
F1 SCORE: 0.2327162028106311

Small learning rates gave in general better results. I tried different values from $1e-5$ to $5e-6$ and since the difference between them was not so important I chose a learning rate = $1e-5$, based on the Stanford paper <https://arxiv.org/pdf/1905.05583.pdf>.

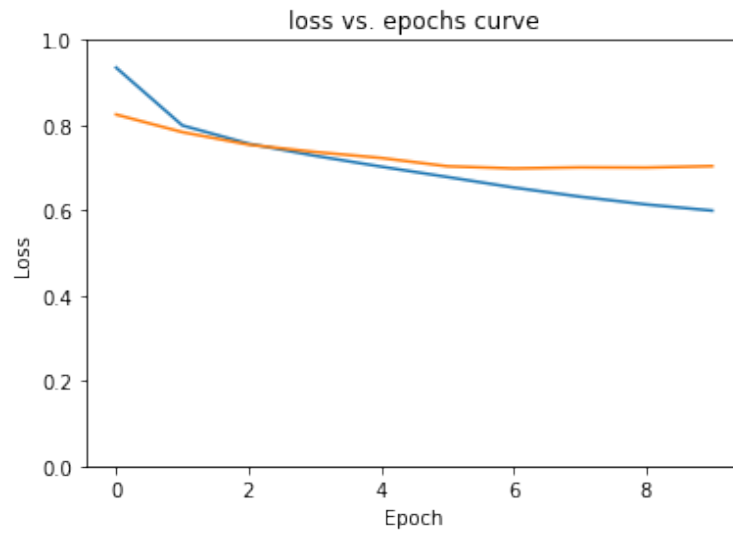
1.4 Epochs

:

What is interesting in this project about epochs is that we do not need as much epochs to reach the greatest result. I ended up using 3 epochs but 2 epochs were more than enough to reach good accuracy and low loss value. This is an advantage of using pretrained models, but I also think that the small data played a role on this.

Below is an example showing how a large number of epochs make the model to

overfit.



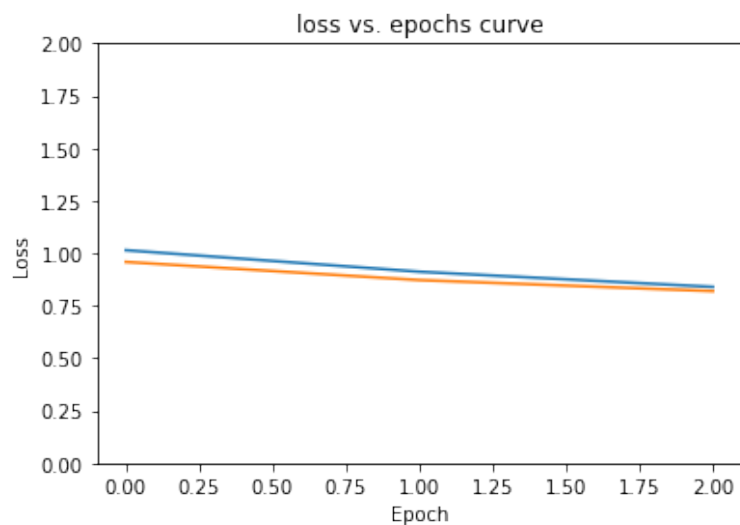
MANY EPOCHS - overfitting

1.5 Batch size

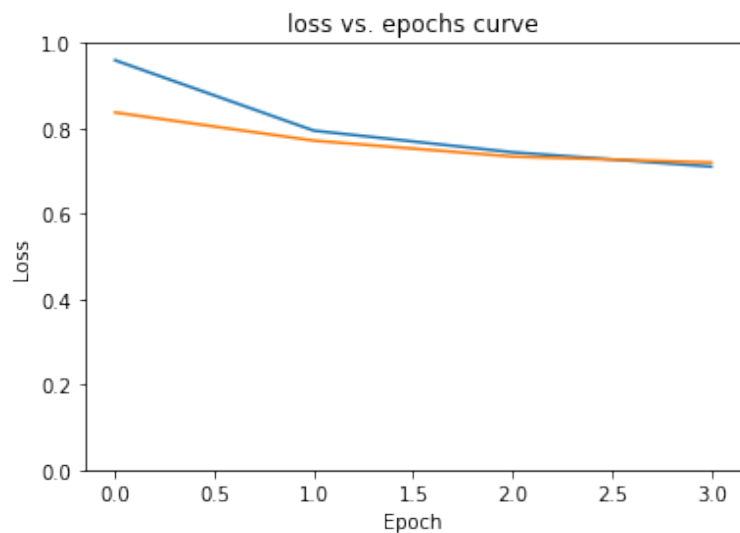
:

As already mentioned the number of batches depends on the maximum sequence length. Also, depends on the learning rate. In general small batches worked better.

So, I ended up using a batch size = 64 which gave a greater F1 score.



Batch size = 128
F1 SCORE: 0.6125479845085552



Batch size = 16
F1 SCORE: 0.6482058933016291

Notice: In the first picture the Y axis is between $[0,2]$ and in the second $[0,1]$.

1.6 References

<https://towardsdatascience.com/text-classification-with-bert-in-pytorch-887965e5820f>
<https://skimai.com/fine-tuning-bert-for-sentiment-analysis/>

<https://datascience.stackexchange.com/questions/64583/what-are-the-good-parameter-ranges-for>
Documentation from hugging face
<https://huggingface.co/docs/transformers/training>
https://huggingface.co/docs/transformers/main_classes/tokenizer

2 Exercise 2

2.1 General assumptions:

For this question the examples shown are not so many because the training process was time consuming and I did not manage to run all the desirable tests before the due date but I had run a number of experiments when I was still experimenting for my self trying to figure out things and I have wrote and explained my observations below.

There are some parts of code from evaluation script of the squad site:

<https://rajpurkar.github.io/SQuAD-explorer/>

<https://worksheets.codalab.org/rest/bundles/0x6b567e1cf2e041ec80d7098f031c5c9e/contents/blob/>

Also, the implementation of the squad data download and data extraction process is based on the code from different online tutorials like for example

<https://towardsdatascience.com/how-to-fine-tune-a-q-a-transformer-86f91ec92997>,

as this are basic and procedural steps I chose to follow these tutorials and focus more on the fine tuning and the experimentation part of the exercise. I am hoping this is acceptable.

2.2 F1 calculation

For F1 scores calculation I ended up following the implementation of the Stanford from the provided script on the site (<https://worksheets.codalab.org/rest/bundles/0x6b567e1cf2e041ec80d7098f031c5c9e/contents/blob/>)

I calculated the F1 score for each answer using their function and than calculated the average of all the answers as the final F1 score (for each epoch).

2.3 Tokenizer

The tokenizer was one of the most "difficult" decision due to the limited GPU quota of collab(and kaggle). I had a lot of trouble using colab since the amount of data was too big and it kept banning me so I used kaggle for the experimentation process. Also the max_length parameter (maximum sequence length) was one of the key parameters that determined the accuracy of the model and also the different type of tokenizers I tried had very different results. Will be analyzed later on the Experimenting section.

2.4 Learning Rate

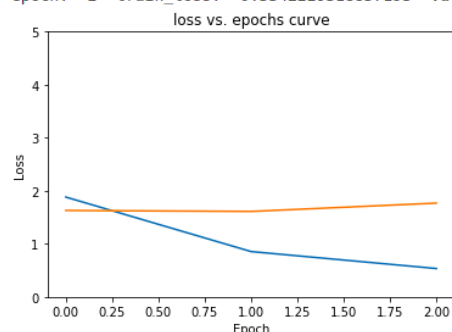
Since we use a pretrained model and for the same reasons as explained above on Exercise 1 (see Exercise 1 section Learning rate), I experimented with small learning rates.

2.5 Epochs

The training for the Squad data was really time consuming so I did not manage to try a lot of epochs, but in general it was not necessary to do so since I reached good results with a fairly small number of epochs. Also as explained in Exercise 1 there is no need for a lot of epochs since we use a pretrained model. And from what I observed the model tend to overfit on the third epoch and the accuracy do not increase fro one epoch to another.

A representative example:

```
0.4460699741745105
epoch: 0 train_loss: 1.8814460473537444 val_loss: 1.628993395979448
0.4518645524618116
epoch: 1 train_loss: 0.8539763971209526 val_loss: 1.6106734782411922
0.4587066804986334
epoch: 2 train_loss: 0.5342229318857193 val_loss: 1.7675900833363214
```



The number printed before each epoch is the corresponding F1 score for each epoch. As you can see as the epochs runs the model overfits and the F1 score increase with only 0.01 which is just 1%.

2.6 Batch size

Since we have a limited GPU quota we have to use small batches in general in order to use larger sequence length which affects more the accuracy of the model. So the batch size had to be 64 always depending on the maximum sequence length value and the available GPU.

2.7 Model Architecture

I used a very simple architecture consisting of BertForQuestionAnswering model. I wanted to try adding different layers, because as shown on this¹ paper more complex architectures give greater results, but due to time limitations I did not manage to try any.

2.8 Experimenting

Since, I "learned" from the first exercise that we need a small learning rate and a fairly small number of epochs I set those parameters as:

Learning rate = $3e-5$ and Number of Epochs = 2(sometimes 3)

and experimented more with the sequence length, batch size and the amount of data. Also, I tried two different models "bert-base-uncased" and "distilBert-base-uncased".

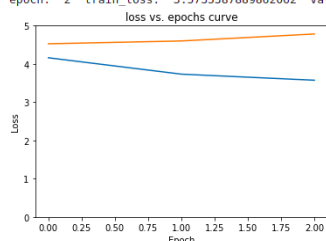
Since the training process is very time consuming most of the examples shown below run only 20000 data in order to show the affects of the hyperparameters.

distilBert-base-uncased

tokenizer: DistilBertTokenizerFast

DistilBert according to google "is a small, fast, cheap and light. It has 40% less parameters than bert-base-uncased, runs 60% faster while preserving over 95% of BERT's performances as measured on the GLUE language understanding benchmark." What I observed is that even though distilBert runs a lot faster than Bert, the accuracy of the results has a large gap between them.

```
epoch: 0 train_loss: 4.1600982168197636 val_loss: 4.526271595169859 F1_score: 0.10139653058453808
epoch: 1 train_loss: 3.731996886253357 val_loss: 4.59971209691792 F1_score: 0.10325580939790718
epoch: 2 train_loss: 3.5735387889862062 val_loss: 4.782160718702689 F1_score: 0.10620288130731654
```



distilBert

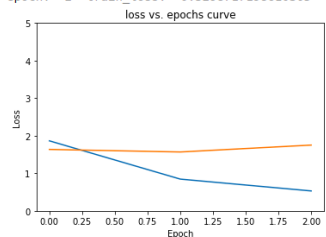
11 min/epoch

Total = 44min Data amount = 20000

max sequence length = 256

Batch size = 16

```
epoch: 0 train_loss: 1.8638742532253265 val_loss: 1.6349636087330377 F1 score: 0.4459225240488845
epoch: 1 train_loss: 0.844880679488182 val_loss: 1.5673324295705775 F1 score: 0.45526117243972875
epoch: 2 train_loss: 0.5298717198610305 val_loss: 1.7510511168346898 F1 score: 0.4571656664274544
```



Bert

22 min/epoch

Total = 1h 12min Data amount = 20000

max sequence length = 256
Batch size = 16

"Total" refers to the total time including the training the tokenization and all the time other cells need to run.

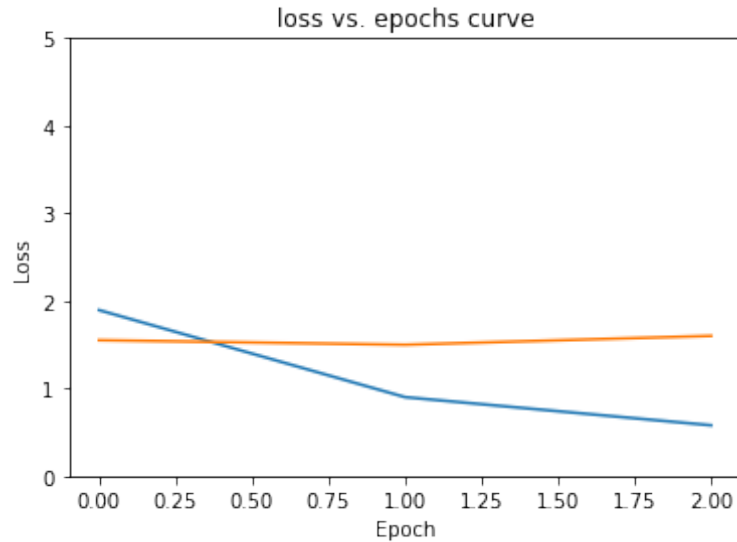
As we can see distilBert is much faster but the F1 scores are significantly worse. If we increase the amount of data the accuracy increase but still Bert has better F1 scores overall.

Bert-based-uncased

tokenizer: AutoTokenizer

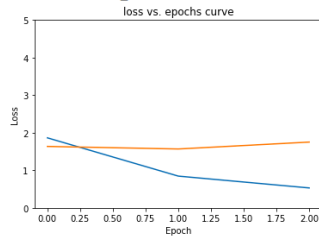
I used this tokenizer since "Auto" automatically retrieve the relevant model so its the same as running BertTokenizer.

All three experiments below runs only 20000 data.



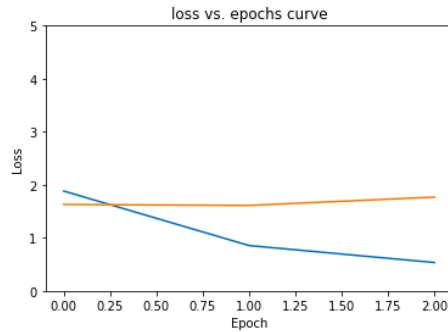
max sequence length = 128
Batch size = 32
F1 SCORE: 38%

```
epoch: 0 train_loss: 1.8638742532253265 val_loss: 1.6349636087330377 F1 score: 0.4459225240488845
epoch: 1 train_loss: 0.844880679488182 val_loss: 1.5673324295705775 F1 score: 0.45526117243972875
epoch: 2 train_loss: 0.5298717198618365 val_loss: 1.7518511168346898 F1 score: 0.4571656664274544
```



Data amount = 20000
max sequence length = 256
Batch size = 16
F1 SCORE: 45%

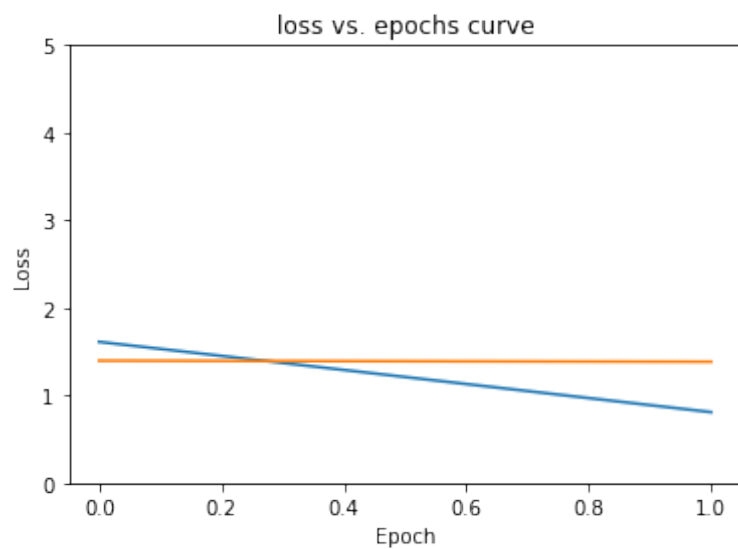
```
0.4460699741745105
epoch: 0 train_loss: 1.8814460473537444 val_loss: 1.628993395979448
0.4518645524618116
epoch: 1 train_loss: 0.8539763971209526 val_loss: 1.6106734782411922
0.4587066804986334
epoch: 2 train_loss: 0.5342229318857193 val_loss: 1.7675900833363214
```



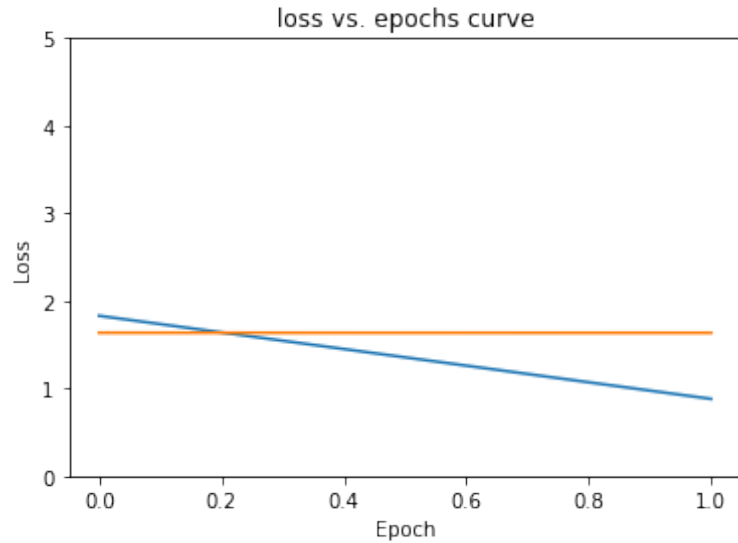
max sequence length = 512
Batch size = 16
F1 SCORE: 45

As you can see as the maximum sequence length value increase the F1 score also increase, but for maximum sequence length 256 and 512 there is no significant difference.

Below are two examples to compare the amount of data. As we increase the data amount the F1 score also increase. So it is better to use all the dataset.



Data amount = 50% of the dataset
max sequence length = 256
Batch size = 16
F1 SCORE: 48%



Data amount = 20000
max sequence length = 256
Batch size = 16
F1 SCORE: 44%

As for the best model based on the paper¹, and on my experiments the best

parameters are as below.

model: bert-base-uncased Learning rate: 5e05 (paper3e05)

Batch size: 24 (same as paper)

Epochs: 2 (same) 1 epoch would be also enough.

Sequence length: 384 (same)

(I could not run this model on colab due to memory limits, the notebook is downloaded from kaggle).

2.9 References

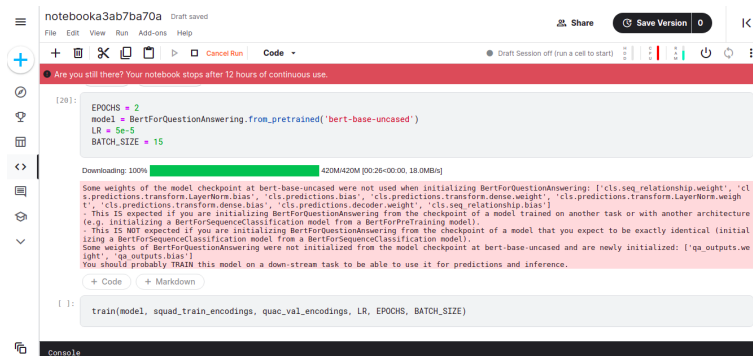
¹<https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1194/reports/default/15848021.pdf>
<http://https://towardsdatascience.com/how-to-fine-tune-a-q-a-transformer-86f91ec92997>
https://qa.fastforwardlabs.com/no%20answer/null%20threshold/bert/distilbert/exact%20match/f1/robust%20predictions/2020/06/09/Evaluating_BERT_on_SQuAD.html?fbclid=IwAR3CkT_hoMUBvTPbu7Ft-Nnxq0ITaoe6lQ07m5zu0Ep5BHp-zDHTYoVq36A#Load-the-dev-set-using-HF-data-processors
<https://kierszbaumsamuel.medium.com/f1-score-in-nlp-span-based-qa-task-5b115a5e7d41>
<https://arxiv.org/pdf/1806.03822.pdf>
Bert documentation:
https://huggingface.co/transformers/v2.10.0/model_doc/bert.html https://huggingface.co/docs/transformers/model_doc/bert https://huggingface.co/docs/transformers/model_doc/distilbert

3 Exercise 3

For exercise 3 there is a folder named **Ex.3** which contains the files for three of the datasets and their combinations (TriviaQA, SQuAD, QuAC). Files have names of format **dataset1_dataset2** where dataset1 corresponds to the dataset used for training and dataset2 corresponds to the dataset used for the evaluation. For example **QuAC_TriviaQA.ipynb** uses QuAC dataset to train the model and TriviaQA dataset to evaluate the model.

I tried to run the dataset based on the parameters that worked best on Exercise 2 but the available space in kaggle was not enough so I run only half of the dataset.

After hours of training kaggle banned me for time limitations and this happened with all the files I tried to run.



The screenshot shows a Jupyter Notebook interface with the title 'notebook3ab7ba70a'. The top bar includes 'File', 'Edit', 'View', 'Run', 'Add-ons', and 'Help' menus, along with 'Share', 'Save Version', and a 'Draft Session off' indicator. A red warning banner at the top states: 'Are you still there? Your notebook stops after 12 hours of continuous use.' The code cell contains the following Python code:

```
[20]: EPOCHS = 2
model = BertForQuestionAnswering.from_pretrained('bert-base-uncased')
LR = 5e-5
BATCH_SIZE = 15
```

Below the code, a green progress bar indicates 'Downloading: 100%' with a size of '420M/420M [30.26<00:00, 18.0MB/s]'. A red error message is displayed below the progress bar:

```
Some weights of the model checkpoint at bert-base-uncased were not used when initializing BertForQuestionAnswering: ['cls.seq_relationship.weight', 'cls.predictions.transform.LayerNorm.bias', 'cls.predictions.bias', 'cls.predictions.transform.dense.weight', 'cls.predictions.transform.LayerNorm.weight', 'cls.predictions.transform.dense.bias', 'cls.predictions.decoder.weight', 'cls.seq_relationship.bias'].
This is expected if you are initializing BertForQuestionAnswering from the checkpoint of a model trained on another task or with another architecture (e.g. initializing a BertForSequenceClassification model from a BertForPretraining model).
This is NOT expected if you are initializing BertForQuestionAnswering from the checkpoint of a model that you expect to be exactly identical (initializing a BertForSequenceClassification model from a BertForSequenceClassification model).
Some weights of BertForQuestionAnswering were not initialized from the model checkpoint at bert-base-uncased and are newly initialized: ['qa_outputs.weight', 'qa_outputs.bias'].
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

At the bottom of the code cell, there is a line of code:

```
[ ]: train(model, squad_train_encodings, quac_val_encodings, LR, EPOCHS, BATCH_SIZE)
```