Homework 4

Βησσαρίων Μουτάφης sdi1800119@di.uoa.gr
1115201800119

1. Sentiment Analysis with Bert

In the *sentiment_analysis.ipynb* notebook we will examine BERT's performance on a multi-sentiment classification task.

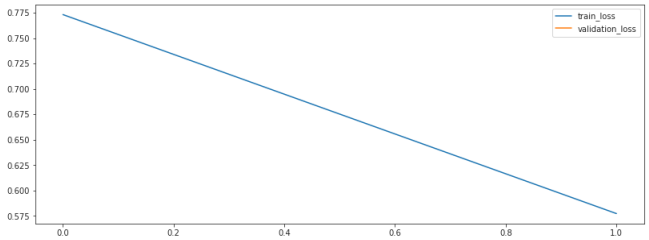
The tweets' preprocessing function will just strip the dataset from trailing whitespaces. We will provide a ClassifierDataset class as in previous assignments so that the user can load the dataset and apply batching and/or shuffling, using pytorch library datasets' methods. Finaly, we used the bert base tokenizer to create the word tokens and after some experiments, we concluded that we could give a max length of 50 with truncation and padding available, in order to get the best model in respect of training duration and generalization score.

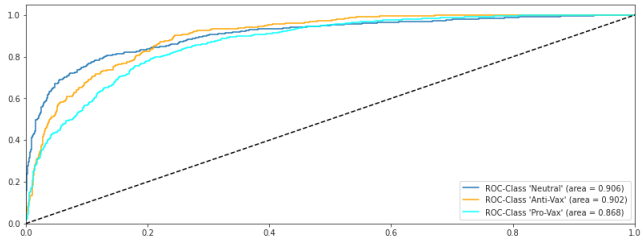
We will train the model on google colabs gpu and provide a simple training function that will print the train loss throughout the training. The training function is in the same spirit as in the previous assignments and is tuned based on sentiment analysis tutorial by hugging face library[1]. As we will see in the following plot the training error was decreasing constantly in 3 epochs (could train with more epochs to learn the dataset better, but this is not the point). The hyper parameters of the model will be presented in table 1 1

Epochs	3
Learning Rate	$3e-5$
Batch Size	24

Table 1: Hyperparameter Tuning

During the evaluation we monitor the precision, recall and macro f1 scores, indicating that our transformer-based model is the based sentiment classifier of all. This can also be seen by the ROC-Curve since we can watch the sensitivity between classification per class exploding, even for the Anti-Vax Class where we repeatedly caused us problems, due to the lack of training samples.





2. FineTuning BERT on QA Datasets

To tune Bert on the different datasets, we downloaded and formated all the datasets based on this repository on github that converted the different datasets to the squad format and implemented the actual scripts from the given paper. After we get datasets, we used a self made training loop and finally provided a notebook that evaluates the results of our final models.

Additionaly, we provide separate notebooks to fine tune the models and use the huggingface's hub to save our models in order to use them all together in the final notebook and save time so that we do not train them each time.

Note that we also fine tuned distilBert, to compare with the default bert model, but due to the performance loss we proceeded with the original Bert model. If you want to check the fine tune performance of distilBERT on SQuAD, check the respective notebook.

2.1 Kaggle vs Colab

In our attempts to train the models we used Kaggle as a colab alternative, since colab would throw a time out error in the middle of most of the fine tuning runs. Also, Kaggle used more state-of-the-art GPUs so the models run a bit more fast than in Colab. Kaggle notebooks run on Kaggle kernels and we must provide our own datasets. The relative converted datasets were privately uploaded in my personal kaggle account and will be released shortly after the assignment's end.

2.2 Dataset Preprocessing

To preprocess the dataset we had to use the dataset mapping function of huggingface's datasets. In order to get a grasp to the workings of hugging face's tokenizer and datasets we read the following guides 2, 3, 4, as well as hugging face bert tokenizer docs and question answering guides provided in the home page.

The tokenizer hyper-parameters (max length, doc stride, truncation option, padding option) are the same as the given paper's. Since the contex of most examples is way to long we will apply overflowing token generation, so that we can get more than one instances of the same example, some of them containing the answers while others don't. After the tokenization of a single instance, for all produced instances, we will define the answer range, defining start and end positions in the tokenized context. To do that we will use the offset mapping since one word might be mapped to one or more tokens, meaning that the start to end positions in the original context might not be the same since the indices change, after its tokenization.

Very important adjustment in the datasets was the **exclusion of questions** that overflowed the doc stride, on which the tokenizer would throw an error, explained here.

After preprocessing, we will exclude all the questions that have an answer, but **no part of it** is in the assigned context. This is an attempt to decrease the immense size of the datasets.

2.3 Data Loaders and batching

To load the data of any dataset we used a loading script similar to the one of *squad_v2* 5, where we only changed the way that the script acquires the data, from downloading to local storage searching, given data file paths, as described in the load_dataset() method documentation. The script was loaded in a private dataset in kaggle (will be released with the datasets as well in kaggle account)

After the loading and preprocessing of the data we will place them in pytorch data loaders that will shuffle and batch them creating an iterable that we can use to fine tune our models.

2.4 Training

The fine tuning is implemented via a training loop very similar to the one of the sentiment analysis notebook. We will also provide an exact match calculation through out the batch iterations for every epoch on order to get a glimpse on the fine-tuning scaling. The exact match metric is implemented as in 4 since its a simple measurement. The training was done in kaggle's GPUs since the Colab timeouts during most of the time in large fine tuning tasks. Note that we provide a special notebook for every fine tune task with no evaluation at the end of the training with the exception of fine tuning on squad, where we should conduct a full model report.

Important Note: Due to time and space difficulties we fine tuned on all of the datasets **but the Natural Questions one, on which we fine tuned on a subset of the latter**.

At the end of every fine tune task we will upload the resulting model in hugging face hub for later use.

As a validation set, for validation scores monitoring we used squad_v2 dev split in all of the fine tuning tasks in order to measure performance on a fixed dataset for each model.

2.5 Evaluation

The evaluation notebook, we will download all models and all datasets and evaluate all produced models on all of the *dev* splits of the 5 datasets. Evaluation metrics used are Exact Match and f1 score, as we got them from the respective hugging face metric, used by simple calling load metric routine for squad_v2 dataset (check the hugging face docs for extra insight) and formatting the data as such.

2.6 Conclusions

As we can see in Table 2, the results are quite similar to the papers with the only difference the scores we acquire in the NewsQA dataset. Not given the exact preprocessing method we can only tune the models with the proposed hyper parameters and tune the BERT transformers, in order to solve the QA tasks, given the training datasets. We can see that in most cases the scores might be close or even higher than the ones in the paper and that, only, in NewsQA fine-tuned model we fail to approach the paper's scores, which can only mean that our text preprocessing methods might differ a bit.

Nevertheless we can see that SQuAD and TriviaQA models are the best models and hit high f1 scores in comparison to the ones in the paper. Also we notice that our preprocessing

	SQuAD	TriviaQA	NQ	QuAC	NewsQA
SQuAD	**70.54**	64.83	38.95	36.03	58.97
TriviaQA	39.30	**72.28**	38.47	55.74	59.08
NQ	44.45	56.44	**61.26**	59.96	59.33
QuAC	28.41	56.98	16.60	**53.98**	49.16
NewsQA	31.61	7.97	39.21	4.15	**22.06**

Table 2: Each row refers to the F1-Scores of BERT fine-tuned on the indicated dataset, tested on each dataset (column reference).

will result to achieving better scores on the NewsQA dev dataset on every model and in higher scores on all dev datasets tested on the QuAC-fine-tuned BERT. Nevertheless, we see a decline in NQ dev dataset's scores, which is supposed to yield greater scores than TriviaQA and NewsQA. At the same time, we assume that given the length of NQ's questions' context and our size decrease at all datasets, train and dev, the models could not be able to find the short answer in the long context and for that they scored lower in that dataset-specific task.

At the end, given that we used the paper convertion scripts and default evaluation metrics for all squad-formated datasets, we can conclude that the QA pre-processing method we used will enhance the performance in couple of datasets but will also decrease the performance on others (i.e. NewsQA-fine-tuned model), indicating the importance of a similar preprocessing method, to approach the problem.

3. References

1. https://huggingface.co/blog/sentiment-analysis-python
2. https://towardsdatascience.com/question-answering-with-a-fine-tuned-bert-bc4dafd45626
3. https://towardsdatascience.com/how-to-fine-tune-a-q-a-transformer-86f91ec92997
4. https://github.com/huggingface/notebooks/blob/master/examples/question_answering.ipynb
5. https://github.com/huggingface/datasets/blob/master/datasets/squad/squad.py