Lesson: Artificial Intelligence II

Student's name: Christina Christodoulou

I.D Number: LT1200027

Field of Studies: Language Technology

Homework 4

REPORT

In this assignment, the pretrained **BERT**-base (**B**idirectional **E**ncoder **R**epresentations from **T**ransformers) model[1] is fine-tuned for two NLP tasks using Python, namely text classification and question answering. The solution was implemented using the Pytorch library in Google Colab. The available device used for training was the GPU. The assignment involves 3 tasks. Each task is provided in a separate ipynb file.

More specifically, in the first task, a multiclass vaccine sentiment classifier is developed by categorizing texts into 3 classes (neutral, anti-vax, pro-vax). The training and validation datasets were provided in csv form containing tweets and their labels, 0 for neutral, 1 for anti-vax and 2 for pro-vax. The validation dataset was copied, renamed as *vaccine_test _set.csv* and used as the test dataset as well. I experimented with the dropout probability, the learning rate, the batch size and the number of epochs. The model is saved after training and can be loaded for evaluation with any prepared test dataset by changing the name of the input csv file. The precision, recall and F1 score for each class as well as the accuracy of the model is calculated. I also plotted the loss vs epochs and accuracy vs epochs as well as the confusion matrix to check the model's performance.

In the second task, the BERT model is finetuned for question-answering in the SQuAD 2.0 dataset. The finetuned model is saved after training and can be loaded for testing in a few question-answering examples. The Exact match of the model is calculated. The exact match and the F1 score are calculated in several examples. I also plotted the loss vs epochs.

In the third task, BERT is finetuned for question-answering in the SQuAD 2.0 dataset and is evaluated on both the SQuAD 2.0 and the TriviaQA dataset and some results are reproduced. Th Exact Match (EM) and the F1 score were calculated on single examples. In the following paragraphs, I will explain in detail the steps followed in order to develop the models as well as my observations.

The pretrained BERT model

There are many reasons for selecting to fine-tune the BERT model instead of developing a model from scratch. First of all, the BERT model was already trained on a huge amount of unlabeled data extracted from BooksCorpus (800 million words) and Wikipedia (2,500 million words), thus it takes much less time to fine-tune a pretrained model. By adding a simple one-hidden-layer neural network classifier on top of BERT and fine-tuning BERT, state-of-the-art performance can be achieved. What is more, BERT utilizes the bidirectional nature of the encoder stacks, since it consists of several Transformer encoders stacked together. Each Transformer encoder encapsulates two sub-layers, a self-attention layer and a feed-forward layer. This means that BERT learns information from a sequence of words from both directions. Moreover, due to its pre-trained weights, the model can be fine-tuned on a much smaller dataset compared to one that would be required in a model that is built from scratch. Finally, the fine-tuning procedure can offer impressive results on a small dataset for a few training epochs for various NLP tasks.

The BERT model takes as input a sequence of tokens. There are 2 special tokens for Bert:

- [CLS]: The first token of every sequence, which stands for classification token.

[1] Available on Hugging Face: https://huggingface.co/models

- [SEP]: The token that makes BERT know which token belongs to which sequence. This is important for a next sentence prediction task or a question-answering task. In case of one sequence, this token is appended to the end of the sequence.

An input sentence is easily transformed into a sequence of tokens appropriate for BERT by using its tokenizer (BertTokenizer). The maximum size of tokens for BERT is 512. In case the tokens included in a sequence are less than 512, padding is used to fill the empty token slots with the [PAD] token. In case the tokens included in a sequence are longer than 512, truncation is used.

The BERT model outputs an embedding vector of size 768 in each of the tokens. These vectors can be used as an input for different kinds of NLP tasks, like text classification and question-answering that are developed in this assignment.

Text classification: Finetuning BERT on a vaccine sentiment dataset – Task 1

For the text classification task, the focus is on the embedding vector output from the special [CLS] token. In other words, the embedding vector of size 768 from [CLS] token is used as an input for the vaccine classifier, which will output the number of the classes. The *bert-base-uncased* model is used because it is computationally efficient. It has 12 layers, 768 hidden states, 12 heads, 110M parameters in total. The BERT tokenizer is utilized. The *do_lower_case* is applied to the tokenizer in order to convert everything to lower-case.

A class named *BertClassifier* constructs the Bert-base model with two additional linear layers on top of it, a dropout layer and the softmax activation function. The first additional layer uses the embedding vector of size 768 from the [CLS] token and the maximum size of tokens that can be fed into BERT model, which is 512. Then, the ReLU activation function is applied. It returns 0 if it receives a negative input, it returns that value back for a positive input. The ReLU output, which ranges from 0 to infinity, passes it through a dropout layer. After that, it generates as output the number of the classes through a linear layer. At the end of the linear layer, the output is a vector of size 3 passed from the softmax activation function, which corresponds to the three labels (neutral, anti-vax, pro-vax). The softmax activation function makes sure the three probabilities sum up to 1.

A. Code description - Steps

First of all, the necessary libraries like torch, tqdm, nltk, sklearn, numpy, pandas and matplotlib were imported as well as the necessary packages like the transformers library by huggingface were downloaded. Then, the English set of stopwords was set to use. To run this notebook, the following files were uploaded: *vaccine_train_set.csv, vaccine_validation_set.csv* and the *vaccine_test _set.csv*. The script contains many functions from the previous assignments that were used for data preparation and metrics' calculation as well as new functions and classes prepared for this assignment.

Steps and functions for data preparation and preprocessing:

- *read_explore_dataframe(csv_file)* function: takes as parameter a file in csv form. It opens and reads the file as a dataframe using the pandas library and utf-8 encoding, gets the values and number of its values. It also checks whether there are empty values and duplicates, removes the missing values, the index and the duplicates.
- *text_preprocessing(text)* function: takes as parameter a text and applies some preprocessing steps like removing unusual characters, urls, punctuation, emoticons, numbers, stopwords and applying lowercasing. Apart from this, it tokenizes the text using the *word_tokenize* function and gets the lemmas of the tokens using the *WordNetLemmatizer* offered by nltk and returns the re-created sentences with their lemmas.
- *get_columns(dataframe, feature, label)* function: takes as parameters a dataframe, the name of the feature's column and the name of the label's column. More specifically, it selects the columns of interest and puts them in a new dataframe using pandas. It checks the distribution of each class and then

separates the values of the label's column into 3 classes, 0 for neutral, 1 for antivax and 2 for provax. Here, I apply resampling of the classes due to the imbalance data problem (mainly in class 1) I highlighted in the previous assignments, using the *resample* function offered by sklearn. Instead, I created a new dataframe with the equal number of samples so as to achieve data balance and fair predictions of the model for each class. What is more, the previous function is applied to the text (feature) column for pre-processing. After pre-processing, the feature column is assigned as the x input value and the label column as the y target value. The x and y values are concatenated in a new cleaned dataframe, which is returned by the function.

- The new cleaned datasets are saved in csv form under a new name that denotes their purpose (*train*, *validation*, *test*). Next, the train and validation datasets are opened using the pandas library and their tweet (text) values are concatenated and encoded by BERT's tokenizer with the addition of special tokens in order to get the maximum length of the sequences. The maximum sequence length of the train and validation datasets was found to be 186.
- *Data_Preparation(Dataset)* class: uses the *Dataset* from torch.utils.data and takes as input the texts, their labels, a tokenizer and a maximum length. It transforms the dataset into the appropriate format for BERT. I used the tweet.values and the label.values from the train and validation dataframes, the defined BERT tokenizer and the maximum sequence length of the train and validation datasets. This class uses the *tokenizer.encode_plus* function to add the special BERT tokens, pad the sentences according to the maximum length, truncate sentences, return the attention mask and pytorch tensors, but do not return the token type ids. It returns the input_id for each token, its attention_mask and its true label.
- The encoded train and validation datasets are then passed through the *DataLoader*. Random batches from the train dataset are taken using the RandomSampler, while sequential batches from the validation dataset. The same procedure is followed for the preparation of the test dataset further on the Colab notebook after loading the saved model.

Function for model training and evaluation (training and validation datasets):

- *train_evaluate(model, train_dataloader, valid_dataloader, loss_function, optimizer, scheduler, batch_size, epochs)* function: takes as parameters a defined model, the train and validation dataloaders, a defined loss function, an optimizer, a scheduler and the number of epochs. It performs the training and then the validation phase using the train and validation datasets using a Pytorch training and evaluation loop. It saves and returns the training and validation losses and accuracies as well as the training and evaluation amount of time for each epoch into a list.
- This list is then opened as a dataframe to inspect the results for each epoch. After that, the learning curves of loss and accuracy are plotted to view the model performance.

Function for model evaluation (test dataset):

- *test(model, test_dataloader)* function: It evaluates the model using a test dataloader through a Pytorch evaluation loop. It calculates the steps, the predictions, the losses and accuracies for each epoch. It turns the predictions into one dimensional tensors and saves the predicted labels and the true labels into two lists. It passes them to the GPU for faster computation and calculates the metrics using the *calculate_metrics(y_true, preds)* function. It returns the saved true and predicted labels.

Functions used for metric calculation:

- *accuracy(preds, y_true)* function: takes as parameters the predictions of labels of the model and the true labels. It changes the predictions into one dimensional tensors and calculates the accuracy. This function is specifically included and used in the *train_evaluate* function.
- *calculate_metrics(y_true, preds)* function: takes as parameters the y true values and the predicted values. It calculates all the metrics asked for this assignment, namely the precision, F1 score and

recall. It also returns the estimated the accuracy of the models and prints the classification matrix as a general summary of all the metrics. This function is specifically included and used in the *test* function.

- *show_confusion_matrix(confusion_matrix)* function: takes as parameter a confusion matrix of true labels and predicted labels and plots the confusion matrix with labels in blue colors. This function is specifically included and used in the *calculate_metrics* function.

Other functions used:

- *calculate_time(elapsed_time)* function: calculates the time in seconds during training and evaluation for each epoch and returns a string in the form hh:mm:ss.
- *set_seed(seed_value)* function: sets the seed for reproducibility of the same results.

B. EXPERIMENTS

According to Devlin, Chang, Lee, Toutanova (2019), a batch size of 32, 3 epochs and the learning rates 5e-5, 4e-5, 3e-5, and 2e-5 are mentioned for fine-tuning BERT. For this reason, during the data preparation and the training process, I experimented with the aforementioned parameters. More particularly, I experimented with:

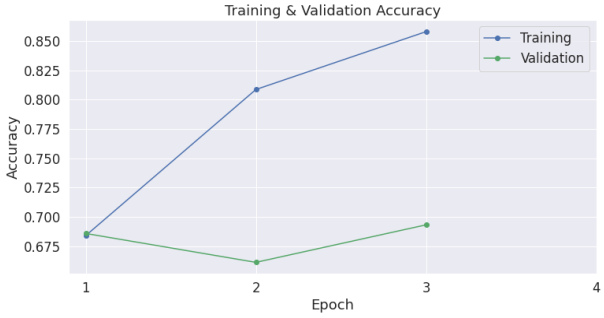
- the dropout probability: I experimented with 0.1, 0.3, 0.4, 0.5.
- the learning rate: I experimented with 5e-5, 4e-5, 3e-5, and 2e-5.
- the batch size: I experimented with 16 and 32. I also tried 64, but I encountered out of memory limit issue, thus I decided to reduce the batch size.
- the number of epochs: I experimented with 3 and 4 epochs. The training with 4 epochs lasted way much longer, so I decided to reduce the number of epochs.

My final choice of hyperparameters can be seen from the table below:

HYPERPARAMETER	VALUE
Max Seq Length	186
Dropout	0.5
Learning rate	5e-5
Batch size	32
Epochs	3

C. RESULTS – OBSERVATIONS

The whole training and evaluation process took approximately 3 hours to complete. Each epoch took about 1 hour to train and about 3 minutes to evaluate. I selected to inspect the model training progress by printing every 214 steps. 214 occurred by dividing the 1498 training instances with 7, so that the progress can appear 6 times in Colab. Every step appeared approximately every 8 minutes. The testing process also took approximately 3 minutes to complete. Overall, the training and evaluation process lasted much longer than expected, perhaps due to the large maximum sequence length chosen, which affects the training and evaluation speed.



As can be seen from the plot of loss vs epochs, the model is overfit, because the training loss continues to decrease with experience at the end of the plot, while the validation loss increases and then starts gradually decreasing. There is also a huge gap between the loss curves which denotes high variance. This indicates that the model was capable of further learning and that the training process was halted prematurely.

Precision, Recall and F1 score for each class + Accuracy of models

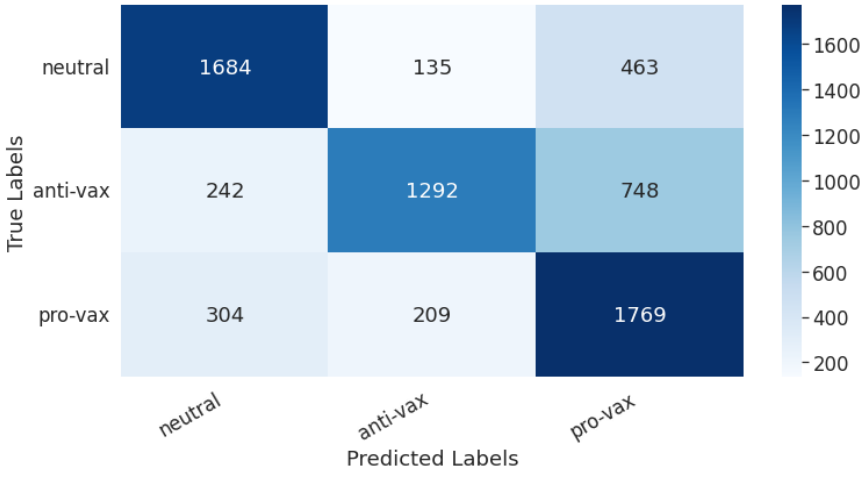
MODELS	PRECISION (CLASS 0 | 1 | 2)	RECALL (CLASS 0 | 1 | 2)	F1 SCORE (CLASS 0 | 1 | 2)	ACCURACY
SOFTMAX REGRESSION MODEL	0.93 | 0.94 | 0.91	0.89 | 0.98 | 0.91	0.91 | 0.96 | 0.91	0.92
FEED-FORWARD NEURAL NETWORK MODEL	0.58 | 0.59 | 0.58	0.73 | 0.69 | 0.32	0.64 | 0.64 | 0.41	0.58
BI RNN WITH LSTM	0.75 | 0.64 | 0.56	0.71 | 0.73 | 0.51	0.73 | 0.68 | 0.53	0.65
BERT BASE UNCASED	**0.76 | 0.79 | 0.59**	**0.74 | 0.57 | 0.78**	**0.75 | 0.66 | 0.67**	**0.69**

From the table above, it is evident that the model achieved satisfying results by fine-tuning BERT, it could have performed much better in my opinion though. Although I experimented a lot with the learning rate and the batch size, the results remained about the same. For example, in one experiment, the F1 score was 75 for class 0, 69 for class 1 and 65 for class 2, while the accuracy was 69.

The model achieved higher precision score in the anti-vax class (1) and the lowest in the pro-vax class (2). On the contrary, the recall score in the anti-vax class is the highest, while in the pro-vax class the lowest. The model achieved higher F1 score in the neutral class (0), while the lowest score in the anti-vax class. Finally, the model performed with 69% accuracy.

By comparing with the other models from the previous assignments, it can be noted that the BERT classifier comes second in model performance, while the Softmax Regression Model remains by far the best model since it achieved the highest F1 scores in the three classes and the highest accuracy amongst the models. The Feed-Forward Neural Network achieved the lowest F1 scores, in class 2 even below 0.50, as well as the lowest accuracy.

From the confusion matrix below, it seems that the model was able to classify 1684 neutral labels, 1292 anti-vax and 1769 pro-vax labels out of the 2282 for each class correctly. An ideal classifier would result in a confusion matrix where there are values only on the diagonal though. Thus, the classifier could perform better since there are a lot of misclassified values. All things considered, the multi-class vaccine classifier by fine-tuning BERT offered moderately good results. However, no matter the tuning, the classifier could not achieve higher results.



Question-Answering: Finetuning BERT on SQuAD 2.0 – Task 2

In this task, the pretrained BERT-base-uncased model is fine-tuned on the SQuAD 2.0 dataset using the Pytorch library for question answering. The aim of this task is for BERT to be able to find the span of text in a provided passage that corresponds to the correct answer in a given question. It has to find the token that marks the start of the answer (answer start) and the token that marks the end (answer end). BERT takes as input the question and the passage containing the answer which are separated by the special [SEP] token as well as the segment embeddings in order to differentiate the question from the passage (reference text). These are two embeddings (for segments 'A' - Question and 'B'- reference text) that are added to the token embeddings before feeding them into the input layer.

One of the most popular datasets for Question-Answering is the Stanford Question Answering Dataset, or SQuAD, which comes in two versions: SQuAD 1.1 and SQuAD 2.0. These reading comprehension datasets consist of questions posed on a set of Wikipedia articles, where the answer to every question is a segment (or span) of the corresponding passage (context). In SQuAD 1.1, all questions have an answer in the corresponding passage. SQuAD 2.0 steps up the difficulty by including 50,000 unanswerable questions written by crowd workers. The SQuAD dataset comes in json format and it includes several layers like 'data', 'paragraphs', 'qas' that lead to 3 main parts:

- Question: strings containing the question that are asked to the model to find the answer.
- Context: larger segments of text that contain the answers to the questions.
- Answers: shorter strings which are extracted from the given contexts and provide the answers to the questions.

A. Code description - Steps

The necessary packages and libraries were installed and imported. The SQuAD 2.0 dataset was downloaded with the function *download_squad(version = 1)*, which uses the *wget* method to download and save the training and development JSON files of any SQuAD version into a newly created folder. The BertForQuestionAnswering and the Autotokenizer of bert-base-uncased were initialized.

- **Steps and functions for data preparation and preprocessing:**

- *get_store_SQuAD_data(file_path)* function: gets as parameter the path of a SQuAD file, iterates through the file content, extracts the questions, texts and answers and saves them into separate lists. The length of the lists is printed and the lists are returned by the function. In this way, the training and development content from the JSON files was extracted and stored into lists. The contexts and the questions lists are in str form and the answer for each question can be found within the context, while the answers lists is a dictionary including the answer and the answer start.
- *add_end_position(answers, contexts)* function: takes as parameters the lists of the answers and the contexts and aims to find and add the answer end. To find the answer end, it adds the start answer position and the length of the answer. However, there are cases where the answer can be false by one or two characters. For this reason, the context is cut by one or two tokens in order to be the same as the given answer and thus the create appropriate input for BERT.
- *add_token_positions(encodings, answers)* function: takes as parameters the encoded passages and questions as well as the answers. It finds the start and end position of the answer and puts them into separate lists using the *char_to_token* method. It also examines certain cases where there is no start or end position, then the answer has been truncated, and cases where there is no end position. After the tokenizer, it seemed that the end position character was None in 964 answers of the 86821-training data and in 359 answers of the 20302-development data. Thus, the answer is moved one position to the left (added -1). In case of one moving position to the left and still None, the maximum sequence length was provided.

- *Prepare_Dataset(torch.utils.data.Dataset)* class: uses the Dataset from torch.utils.data and takes as input the encoded contexts and their questions. It transforms the data into the correct format for training BERT with the PyTorch library.
- The encoded train and validation datasets are then passed through the DataLoader. Random batches from the train dataset are taken using the RandomSampler, while sequential batches from the validation dataset.
- *def normalize_text(text)* function: takes as parameter a text and applies preprocessing steps like lowercasing, removing punctuation and articles. It is used for the context and its predicted answer in other functions when testing the model with actual questions given a context.

Function for model training and evaluation (training and validation datasets):

- *train_evaluate(model, train_dataloader, valid_dataloader, loss_function, optimizer, scheduler, batch_size, epochs)* function: takes as parameters a defined model, the train and validation dataloaders, a defined loss function, an optimizer, a scheduler and the number of epochs. It performs the training and then the validation phase using the train and validation datasets using a Pytorch training and evaluation loop. It saves and returns the trained model and the training and validation losses, the exact match in the validation dataset as well as the training and evaluation amount of time for each epoch into a list.
- This list is then opened as a dataframe to inspect the results for each epoch. After that, the learning curve of loss is plotted to view the model performance.

Function for model evaluation:

- *def question_answering(question, text)* function: takes as parameters a question and a text in quotation marks as strings and provides the predicted answer using the finetuned BERT-base model. It was used on several random examples to check the model's performance.

Functions used for metric calculation:

- def *compute_exact_match(predicted_answer, true_answer)* function: takes as parameters a predicted answer by the model and its true answer which is given in a text. It computes the exact match of the predicted answer and the true answer, meaning whether the model got or not the right start or end token.
- def *compute_metrics(predicted_answer, true_answer)* function: takes as parameters a predicted answer by the model and its true answer which is given in a text. It computes the precision, recall and F1 score based on a predicted and a true answer given a question and a text.
- The functions *normalize_text(text), compute_exact_match(predicted_answer, true_answer)* and *compute_metrics(predicted_answer, true_answer)* were mimicked from the *squad_evaluate.py*, the official evaluation script for v1.1 of the SQuAD dataset. They can be found here: https://github.com/huminghao16/SpanABSA/blob/master/squad/squad_evaluate.py

Other functions used:

The functions *calculate_time(elapsed_time)* and *set_seed(seed_value)* from the first task are also used in this ipynb notebook.

B. EXPERIMENTS

According to Devlin et al. (2019) in Table 4 of the article, certain hyperparameters are mentioned for finetuning BERT for question-answering. Thus, I decided to experiment with these hyperparameters and a few others. At this point, I would like to note that I tried the exact hyperparameters referred in the article, but could not use them all at once to finetune the model as the memory limit exceeded. More particularly, I experimented with:

- the learning rate: I experimented with 5e-5 and 3e-5, but I kept 3e-5 as mentioned by the authors and as there was not big difference in the results, both showed overfitting.
- the batch size: I experimented with 16, 20 and 24 but with 20 and 24 I encountered out of memory limit issue in Google Colab, thus I decided to reduce the batch size and keep it to 16.
- the number of epochs: I experimented with 2 and 3 epochs. The training with 3 epochs lasted way much longer and could not overcome the overfitting issue, so I decided to reduce the number of epochs.
- The maximum sequence length: I tried 384 with a smaller batch size than the recommended (16) due to memory limit and 256 with batch size 24. The model with the reduced maximum sequence length took longer time to train.
- I also tried to train the model with much less training data (2000) and max sequence length 384 but the model was still overfit and could not produce any correct answer in the question-answering examples.

I selected the model with the less maximum sequence length due to the smaller training amount of time. In fact, the results between the model with the 256 and with the 384 max sequence length do not vary. My final choice of hyperparameters can be seen from the table below:

HYPERPARAMETER	VALUE
Max Seq Length	256
Doc stride	128
Learning rate	3e-5
Batch size	24
Epochs	2

D. RESULTS – OBSERVATIONS

The whole training and evaluation process took approximately 2 and a half hours to complete. Each epoch took about 1 hour and ten minutes to train and about 5 minutes to evaluate. I selected to inspect the model training progress by printing every 603 steps. 603 occurred by dividing the 3618 training instances by 6, so that the progress can appear 5 times in Colab. Every step appeared approximately every 11 minutes. Overall, the training and evaluation process lasted much longer than expected, perhaps due to the large maximum sequence length chosen, which affects the training and evaluation speed. On the other hand, with maximum sequence 384, batch size 16, learning rate 3e-5 and 2 epochs, the whole training and evaluation process took approximately 4 hours to complete. Each epoch took about 1 hour and 50 minutes to train and about 9 minutes to evaluate. Every step appeared approximately every 12 minutes.

As can be seen from the plot of loss vs epochs and the table below, the model is overfit, because the training loss is very high in the first epoch and drops instantly in the second, while the validation loss remains stable upon adding training examples. It seems that the model is able to get the exact span of an answer correctly (EM) by 69%. Unfortunately, it is not a very impressive result because question answering is a demanding task and needs well-performing models to be up to the task. The plot of loss is exactly the

same for the model with the 384 maximum sequence length. The slight differences can be seen from the table below.

Epoch	Training Loss	Validation Loss	Accuracy (Exact Match)
1 (256 max seq len)	1.39	1.08	0.68
2 (256 max seq len)	0.76	1.08	0.69
1 (384 max seq len)	1.31	1.1	0.69
2 (384 max seq len)	0.72	1.1	0.70

The finetuned model can be uploaded using the torch library and the CPU to test in a few question-answering examples. Two article abstracts are provided along with a question. The answer is contained in the abstract. The first abstract was taken by Devlin, et al. (2019) in “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding” and the second from Sen et al. (2020) “What do Models Learn from Question Answering Datasets?”. A user’s input is also provided so that anyone can test the model with his/her examples by providing a question and a context in the white box without quotation marks.

We introduce a new language representation model called BERT, which stands for Bidirectional Encoder Representations from Transformers. Unlike recent language representation models (Peters et al., 2018a; Radford et al., 2018), BERT is designed to pretrain deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT model can be finetuned with just one additional output layer to create state- of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial taskspecific architecture modifications. BERT is conceptually simple and empirically powerful. It obtains new state-of-the-art results on eleven natural language processing tasks, including pushing the GLUE score to 80.5% (7.7% point absolute improvement), MultiNLI accuracy to 86.7% (4.6% absolute improvement), SQuAD v1.1 question answering Test F1 to 93.2 (1.5 point absolute improvement) and SQuAD v2.0 Test F1 to 83.1 (5.1 point absolute improvement).

Question: *What does BERT stand for?*

Answer: "bidirectional encoder representations from transformers ."

Exact Match 0

F1 score 0.06666666666666667

Extractive reading comprehension systems can often locate the correct answer to a question in a context document, but they also tend to make unreliable guesses on questions for which the correct answer is not stated in the context. Existing datasets either focus exclusively on answerable questions, or use automatically generated unanswerable questions that are easy to identify. To address these weaknesses, we present SQuAD 2.0, the latest version of the Stanford Question Answering Dataset (SQuAD). SQuAD 2.0 combines existing SQuAD data with over 50,000 unanswerable questions written adversarially by crowdworkers to look similar to answerable ones. To do well on SQuAD 2.0, systems must not only answer questions when possible, but also determine when no answer is supported by the paragraph and abstain from answering. SQuAD 2.0 is a challenging natural language understanding task for existing models: a strong neural system that gets 86% F1 on SQuAD 1.1 achieves only 66% F1 on SQuAD 2.0.

Question: *What is SQuAD 2.0?*

Answer: "existing datasets either focus exclusively on answerable questions , or use automatically generated unanswerable questions that are easy to identify . to address these weaknesses , we present squad 2 . 0 , the latest version of the stanford question answering dataset"

Exact Match 0

F1 score 0.3666666666666667

To sum up, it seems that when the questions include words or phrases from the paragraph the model answers correctly, but it is not fully able to find the correct span of the answer. Taking into consideration the fact that the model had overfitting issues, it has moderate performance.

Question-Answering: Scores on SQuAD 2.0 and TriviaQA – Task 3

In this task, the pretrained BERT-base-uncased model for Question Answering was asked to be employed in order to reproduce the F1 score of various datasets according to Table 3 from the article by Devlin, et al. (2019). I tried to reproduce the results on the SQuAD and the TriviaQA datasets. Unfortunately, due to limited GPU RAM memory and space disk, I was not able to run the evaluation on the whole TriviaQA dataset. I used the dataset library where I downloaded the 5% of the dataset. I did not manage to produce the F1 score for the whole SQuAD 2.0 and the TriviaQA datasets, only at single examples, but I managed to get the EM. I used the CPU and loaded my finetuned model named *SQuAD_2.0_BERT_model_384.pt* with the parameters below. Unfortunately, I could not use bigger batch size number because the Cuda ran out of memory, but it resembles the model the researchers finetuned in the article.

HYPERPARAMETER	VALUE
Max Seq Length	384
Doc stride	128
Learning rate	3e-5
Batch size	16
Epochs	2

RESULTS – OBSERVATIONS

The exact match of the model in the SQuAD 2.0 dataset is about 70%. I tested the model on some examples from article abstracts as in the previous task. It seems that the model cannot always answer the question correctly, especially when the question is complicated or something particular is asked, for example “What is the Master's name?”, “christina christodoulou” or “Is she Greek?”, "christina christodoulou. i am from greece and i live in athens." or “What does the 'B' in BERT stand for?” and it answers "bidirectional encoder representations from transformers". The model can be tested by using any question and context by entering in the user’s input a text and a question without quotation marks.

I kept only specific contents of the dataset by using the *format_dataset(example)* function and I created a new dictionary with new contents, the context, a unique correct answer and other possible answers. I also filtered out the empty examples that were in the context using the filter() function. The function *evaluate_result(example)* gets the dataset, which is in dictionary form, encodes the question and the context in a form appropriate for BERT with Truncation, maximum length 384, document stride 128, makes predictions and computes the exact match of the sentences. The evaluation lasted about 20 minutes as the dataset was quite small. The wrong and the correct results are also produced along with the correct answer to inspect the model’s performance. It seems that the model produces a very low exact match (EM) score, 11.38, as it only predicts the correct answer for only 66 examples on the TriviaQA dataset. Moreover, I created a dataframe to view the created content and then turned the categories into separate lists, the contexts, the questions, the unique answers and the possible answers. I used the function *question_answering_trivia(question, context)* to get the predicted answer, the exact match and F1 score for

single examples and I tested it in random examples. It seems that the model is not able to find the correct span of text as answer, thus its result is wrong or it includes unnecessary information. For example, in the question “Who became the host of the BBC Radio 1 weekday breakfast show in September 2012?” the model answers “"tony blackburn," whereas the correct answer is Nick Grimshaw, or in the question “In which year was the Juilliard School founded in New York?” the model answers “"1905. it is informally referred to as juilliard. the school trains about 850 undergraduate and graduate students in dance, drama, and music. it is widely regarded as… “while the correct answer is just “1905”. This can be mainly due to the fact that the model was trained on the SQuAD 2.0 dataset. The performance on its training dataset is not ideal either though.

All things considered, it is evident that the question answering task is a very demanding and time-consuming procedure and it requires some necessary tools to be fully completed without issues, like a great GPU and very large disk space. It also requires a great deal of reading and research in previous studies.

Bibliography

Devlin, J., Chang, M., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *ArXiv*, abs/1810.04805.

Rajpurkar, P., Jia, R., & Liang, P. (2018). Know What You Don’t Know: Unanswerable Questions for SQuAD. *ACL*. 10.18653/v1/P18-2124

Sen, Priyanka & Saffari, Amir. (2020). What do Models Learn from Question Answering Datasets?. 2429-2438. 10.18653/v1/2020.emnlp-main.190.

https://www.mygreatlearning.com/blog/relu-activation-function/