Deep Learning for Natural Language Processing

Homework 4

Eirini Kolimatsi

Student ID: 7115112200015
Email: eirini[dot]kolimatsi[at]di[dot]uoa[dot]gr

MSc Computer Science
University of Athens
Academic Year: 2022-2023

Contents

1 Task 1: BERT Sentiment Analysis 1

1.1 Data Preparation . . . 1

1.2 Models . . . 2

1.3 Training . . . 2

1.4 Results & Model Comparison . . . 3

2 Task 2: Wikidata Question Answering Models 5

2.1 Data Preparation . . . 5

2.2 Pytorch Preparation . . . 5

2.3 Models . . . 6

2.4 Training Setup . . . 6

2.5 Results . . . 7

3 Task 3: Wikidata Question Answering Interface 7

4 References 7

1 Task 1: BERT Sentiment Analysis

1.1 Data Preparation

Libraries Firstly, all required libraries should be loaded. For the needs of this assignments modules from pytorch (torch, torchtext), scikit-learn (sklearn), pandas, numpy, re, matplotlib, contractions, google.colab and other utility libraries.

Data Loading In order to access the data provided for the assignment, the csv file was uploaded in Google Drive. Then the notebook got ”connected” to Google Drive, to allow reading the file from a path. The path that leads to the file is `/content/drive/path/to/file`. When testing the model against other data, the Google Drive file path should be replaced in the last section of the relevant notebook.

Pytorch Device If GPUs are available, then the notebook runs on GPU using CUDA to reduce the required time to execute cells through parallelising tasks. In Google Colab, GPU runtime is provided, but in cases it is not available simple CPUs will be used. However, given the required run times to train the model, using CPUs is not recommended.

Data Cleaning The provided data have 3 columns: rating, URL and review. The URL column is not used. Reviews are expressed in a scale from 1 to 10. As a result, all negative reviews (with a score below or equal to 4) are marked with 0 and all positive reviews (with a score above or equal to 7) are marked with 1.
The data cleaning steps remain the same as in the 2nd assignment:

- HTML tags removal
- contractions removal
- length reduction of repeated letters
- removal of multiple spaces

Given that this is the same dataset as in the three previous assignment, data exploration has already taken place and will not be repeated in this report. However, insights from the previous reports/analysis were used throughout this assignment.

Splits The dataset was splitted in 3 sets: train, validation and test sets, using the relevant utility from `sklearn`. First, the set was split in train (90%) and test (10%) sets. Then, the train set was split in train (8/9) and validation (1/9) sets. The split was done with stratification for y, to maintain the class balance. The final sizes of the datasets are:

- Train: 80%
- Validation: 10%
- Test: 10%

BERT Configuration The fine tuning was done using the uncased pretrained BERT. Given that some reviews are really long, truncation was performed to reduce the length of those reviews. The max length was set to 235, which is close to the average size of the reviews in dataset. Obviously, some reviews are shorter than 235 tokens, so in these cases padding is required.

Pytorch Tensors After data cleaning and tokenization/encoding, the reviews and the ratings are transformed into tensors. In detail, a Pytorch dataset is created for each of the train, validation and test datasets containing the encoded tokens, the attention masks and the true labels of whether the review is positive or negative. The dataloaders are created as well, using a batch size of 10. This is a rather small batch size, but when trying to use larger I faced problem with memory that I couldn't resolve unless reducing the batch size.

1.2 Models

A main model has been created, with the ability to produce different architecture through parameters:

- Pretrained BERT Model
- Linear Layers
- Dropout Layers

The activation function used in ReLU. From the experimentation with activation functions in the second assignment (Tanh and Leaky ReLU), and the very strong performance of ReLU compared to the other two, it was selected as the activation function and no further experiments were performed towards that.
Regarding the rest of the training and design options of the model, due to the required time to complete the training for each epoch experimentation was limited and often terminated before completion. Thus, in the notebook not many experiments are included due to this factor. However, experiments with different learning rates, hidden size and dropout probabilities were performed.

1.3 Training

The training parameters were set as follows:

Loss Function The loss function used in training is BCEWithLogitsLoss, which combines a Sigmoid layer and BCELoss. It is chosen over BCELoss because of the increased numerical stability that provides compared to applying a sigmoid layer in the model and then selecting the BCELoss criterion. Finally, it is chosen over the simple CrossEntropyLoss as it directly take cares of the needs of binary classification that we aim to achieve.

Optimizer The optimizer used is AdamW. The reason it is selected over Adam that was used in the previous assignments is its better generalization ability. The default learning rate and epsilon value were selected.

Learning Rate Scheduler Even though a scheduler was not used in the previous assignments, in this one it is very important to use a scheduler given the time BERT takes to fine tune. A scheduler can help with adjusting the learning rate with a predefined scheduled, which is crucial in the time needed for the model to converge.

Gradient Clipping As seen in the 3rd assignment, gradient clipping impacts the performance quite a lot, so given these results gradient clipping is done, with size equal to 1.

Epochs 2 epochs due to limited resources and given that results are satisfying.

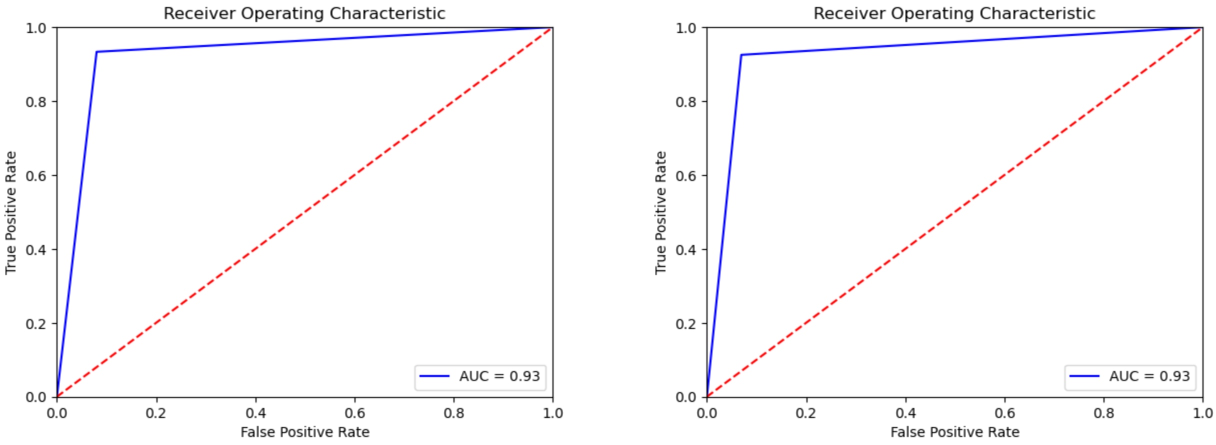
The training included evaluation in the validation set for each epoch. At the end of the training, each model was evaluated on the test set for the required metrics (F1-score, Precision, Recall, Accuracy).

1.4 Results & Model Comparison

From the experiments performed, below are presented two of them, one with a dropout layer and one without.

Metric	**No Dropout**	**Dropout 0.5**
F1-score	92.590	92.805
Recall	93.321	92.498
Precision	91.870	93.114
Accuracy	92.646	92.779

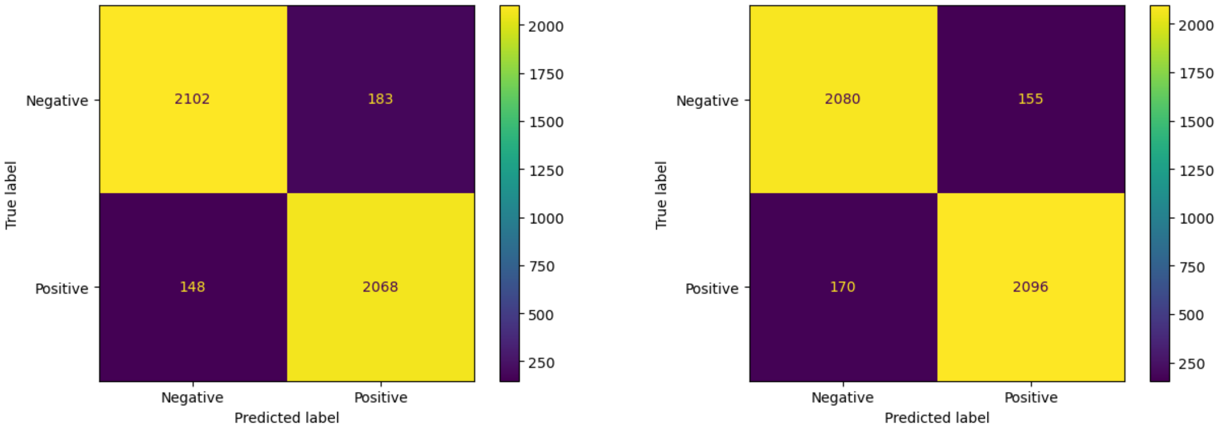
As we can see, the results are pretty close. However, the model that has a dropout layer performs slightly better, as we would expect. In terms of the ROC curve, both models have the same performance.



(a) Without Dropout

(b) With Dropout

Figure 1: ROC Curve



(a) Without Dropout

(b) With Dropout

Figure 2: Confusion Matrix

The training and validation loss and training and validation accuracy learning plots were done as well and can be found in the notebook. However, due to the limited epochs of training, they do not provide useful insights for the performance of the models, thus they're omitted from this report.

In conclusion, the best model that is selected as the most appropriate for the task on top of BERT has a linear layer (hidden size=50), ReLU as an activation function, a dropout layer with probability 0.5 and gradient clipping size 1.

Comparing the BERT fine-tuned model with the previous assignments, we can see a considerable improvements compared to all 3 other models. This is for sure an expected behavior given the capabilities of BERT. Even though the results is quite good, I would expect maybe to be slightly better (eg around 95%). Trying a different BERT model or more complex model architecture might have fulfilled that expectation, but the complexity of the model and the required runtime was an important factor in the experimentation process.

Metric	BERT	BiLSTM	FFNN	LR
F1-score	92.805	84.941	83.333	88.332
Recall	92.498	85.829	82.852	88.892
Precision	93.114	84.073	83.820	87.531
Accuracy	92.779	84.781	83.426	88.203

To run the model on hidden test data, relevant instructions are given at the end of the notebook.

2 Task 2: Wikidata Question Answering Models

2.1 Data Preparation

Firstly, the data were directly loaded from Github and a list of all the unique identifiers were extracted to minimize the number of required Wikidata API calls to get the data labels. Then a loop was performed to return all the labels. Throughout the process some API calls didn't return a valid label for the id. For those labels, a separate loop was performed to check whether it was a glitch or the label was not available. None of those labels could be retrieved, so data containing data them were removed from the dataset. Potentially these could have been kept in the dataset, but for consistency reasons and since the rows affected were not many, they were removed.

After extracting the mapping as explained above, the entity match spans were calculated. To do that, an additional column was added in the dataframe containing a list if zeros and ones where ones are indicating that this token is part of the entity label, while zeros that is not. The span matching process resulted to certain rows not having any match in the spans, so the results being a list of zeros. After investigating this, punctuation and accent was affecting the procedure, so proper cleaning was performed. Even after that some rows still didn't have a valid entity span. This was due to different spelling of the entity between the label and the question or due to the use of a different name to refer to the same entity. These problems couldn't be fixed in an automated way, so they were removed from the dataset.

Lastly, the two supporting datasets are created that are relation vocabulary, that contains a unique numeric identifier to be used in the model, along with the relation id. Also, an entity to label dataset, containing all the entity ids and their labels.

2.2 Pytorch Preparation

In order to fine tune a pretrained BERT model, proper tokenization should take place and data should be loaded in Pytorch dataloaders. The tokenization (and training) was done using the uncased BERT model. Questions were truncated to the maximum length present in the dataset, given that this length is rather short. As a results, padding was added were necessary, to reach the maximum length.

The Pytorch dataset included the BERT encoded tokens, the attention masks produced in tokenization/encoding, the unique identifiers of the relation ids (labels), the entity span match start point and entity span match end point. The selected batch size is 16.

2.3 Models

Initially, I attempted to combine both tasks in one model, but due to difficulties in debugging, I decided to split the tasks in two different models. Thus, below the two different models will be presented, even though they're quite similar in their architecture.

Relation Prediction Classifier The model consists of a two main elements:

- Pretrained BERT Model
- Linear Layers

The model get the input ids and the attention masks as input and calculates the probabilities over each relation id, to get the one that is most likely to be true for each question. Practically, this model is similar to the one that was presented in the IMDB Sentiment Analysis task, but here instead of binary classification there are multiple labels.

Entity Span Prediction The model consists of the same elements as the Relation Prediction model that was described above, with the difference that the model has two heads so it can predict both the start and the end position for each span, instead of a probability for each relation.

2.4 Training Setup

The training parameters were set as follows:

Loss Function The loss function used in training is CrossEntropyLoss.

Optimizer The optimizer used is AdamW. The reason it is selected over Adam that was used in the previous assignments is its better generalization ability. The default learning rate and epsilon value were selected. A learning rate scheduler is used as well.

Gradient Clipping Gradient Clipping with size equal to 1 is used.

Epochs 3 for the relation task, 4 for the span task.

The training included evaluation in the validation set for each epoch. At the end of the training, each model was evaluated on the test set for the required metrics (F1-score, Precision, Recall, Accuracy).
It is important to further explain the way metrics are calculated in the Entity Span Detection task, because it differs from the usual way of measuring accuracy etc. The calculations are averaged for train, valid and test sets. The final results are the metrics averaged in the test set.

Accuracy Span accuracy is measured in a way to match the whole span, which happens when both the predicted start position matches the true start position and the predicted end position matches the actual end position.

Precision The number of common elements both in predicted and true spans divided by the number of tokens in the predicted span.

Recall The number of common elements both in predicted and true spans divided by the number of tokens in the true span.

F-score Calculated as usual, `(2*precision*recall)/(precision+recall)`

2.5 Results

The results seem quite good as it can be seen from the scores below. It was also attempted to use the models with random questions similar to the ones in the dataset and it looks that most of the times it predicts correctly both the entity span and the relation.

	Accuracy	**F1**
BERT	93.70	98.53

(a) Entity span prediction

	Accuracy
BERT	96.53

(b) Relation Prediction

Table 1: Model Performance

Given that the model score well, some further tests with other parameters were done, but since they didn't result to something better, are not presented here.

3 Task 3: Wikidata Question Answering Interface

For this question a set of functions has been developed at the end of the notebook. Briefly, when the question is posed, it is first cleaned, then after being tokenized it is fed in both the relation model and the entity span model. The model's prediction are mapped and then the SPARQL query is generated. After the API call, the first results is returned, using the `qa_engine` function.

4 References

Websites

- Pytorch documentation
- https://learn.microsoft.com/en-us/windows/ai/windows-ml/tutorials/pytorch-analysis-train-model
- https://aclanthology.org/2022.lrec-1.150.pdf
- https://medium.com/@martinpella/how-to-use-pre-trained-word-embeddings-in-pytorch-71ca59249f76