2022-2023 Interação em Linguagem Natural

Grupo 4 | 54859 Rita Maria Oliveira Rodrigues | 39074 Ana Cristina Antunes da Silva

For the resolution of the project, we contributed equally, investing a total of 15 hours each.

Key aspects of the structure and functionality of the implemented software

Library Installation and Dependency Import

The project began by installing the necessary libraries, such as the datasets library, as well as the accelerate library for efficient training. These libraries provide the required tools and resources for natural language processing tasks. Additionally, the project imported other essential dependencies, including transformers and PyTorch.

Dataset Loading and Exploration

The next step loads the datasets SuperGLUE and BoolQ from the Hugging Face library. Then, an exploratory data analysis was conducted to gain insights of its structure and content. It was discovered that the test dataset did not contain any labels, making it not viable to evaluate the model's performance on the test set. To address this issue, a decision was made to split a portion (10%) of the training dataset to create a new test dataset for future evaluation.

Dataset Tokenization

To prepare the dataset for model training, tokenization was performed using the RoBERTa tokenizer. The tokenizer from the RoBERTa model was employed to tokenize the question and passage pairs from the dataset. This process involved truncating the input sequences and padding them to a maximum length. The tokenization step ensures that the textual data is appropriately formatted and ready for further processing.

Model Definition and Fine-tuning

In this project, the RoBERTa model was chosen for sequence classification tasks. The model was instantiated using the RobertaForSequenceClassification class from the Transformers library. This class allows the pre-trained RoBERTa model to be fine-tuned for the specific task at hand. The number of labels was set to 2, indicating binary classification.

For the fine-tuning, it was configured the training arguments, such as the number of training epochs (set to 2), the batch size for training and evaluation, warm-up steps, weight decay, and logging parameters. The Trainer class was employed to manage the training process. The trainer object was initialized with the defined model, training arguments, and the training and evaluation datasets. The compute_metrics function was also specified to evaluate the model's performance.

Model Saving and Evaluation

Once the model finished training, it was saved for future use. The next step involved evaluating its performance on the separate test dataset that was created earlier. We also implemented a function called "test(passage, question)" as part of the program. This function takes a context and a question as inputs and returns the answer, which is the output of the fine-tuned model for BoolQ, based on the provided context and question.

Values of Hyperparameters

```
training_args = TrainingArguments(
    output_dir='./results',
    num_train_epochs=2, # Treino durante 2 épocas.
    per_device_train_batch_size=12, # Lote de tamanho 12.
    per_device_eval_batch_size=12, # Lote de tamanho 12.
    warmup_steps=100,
    weight_decay=0.01,
    logging_dir='./logs',
    logging_steps=100,
    evaluation_strategy='steps',
    eval_steps=100,
    save_strategy='steps',
    save_steps=200
)
```

- **output_dir='./results':** It is the directory where model checkpoints and training results will be saved.
- **num_train_epochs=2:** It means it goes through the entire dataset twice.
- **per_device_train_batch_size=12:** 12 samples are processed simultaneously during training.
- **per_device_eval_batch_size=12:** Similar to the previous, this hyperparameter specifies the batch size for evaluation, which is the number of evaluation samples processed together during model evaluation.
- **warmup_steps=100:** Warmup steps refer to the number of initial steps during which the learning rate is gradually increased.
- **weight_decay=0.01:** Weight decay is a regularization technique that reduces the model's overfitting tendency. A weight decay of 0.01 indicates a moderate regularization strength.
- **logging_dir='./logs' and logging_steps=100:** It indicates the directory and frequency at which logs of the training progress will be stored.
- **evaluation_strategy='steps' and eval_steps=100:** These settings define the strategy and frequency of evaluation during training. The evaluation is performed at every 100 steps (iterations) in this case, allowing for periodic assessment of the model's performance.
- **save_strategy='steps' and save_steps=200:** It stipulates the saving strategy and frequency for model checkpoints. In this example, the model is saved every 200 steps, ensuring that intermediate-trained models are stored for future use or to resume training.

Results and Discussion

#1 iteration

Step	Training Loss	Validation Loss	Accuracy
100	0.684500	0.665605	0.621713
200	0.667500	0.676395	0.621713
300	0.656100	0.666250	0.621713
400	0.682300	0.665523	0.621713
500	0.667700	0.666432	0.621713
600	0.667800	0.663476	0.621713
700	0.672200	0.663715	0.621713
800	0.672100	0.663226	0.621713
900	0.659100	0.666343	0.621713
1000	0.664300	0.664518	0.621713
1100	0.659600	0.663652	0.621713
1200	0.664500	0.664898	0.621713
1300	0.663900	0.663337	0.621713
1400	0.677900	0.663303	0.621713

#2 iteration

Step	Training Loss	Validation Loss	Accuracy
100	0.663600	0.707232	0.621713
200	0.669000	0.665933	0.621713
300	0.665400	0.671561	0.621713
400	0.666500	0.683387	0.621713
500	0.673500	0.663276	0.621713
600	0.666700	0.665156	0.621713
700	0.667100	0.665425	0.621713
800	0.676500	0.665833	0.621713
900	0.662100	0.669987	0.621713
1000	0.674200	0.664207	0.621713
1100	0.669100	0.663238	0.621713
1200	0.656600	0.663473	0.621713
1300	0.656300	0.665023	0.621713
1400	0.653500	0.664350	0.621713

A few iterations were performed to analyse the behaviour of the results. Overall they do not change significantly, as was expected.

The **training loss** represents the average loss calculated during the training process. It measures how well the model is learning to make accurate predictions on the training dataset. In the results, the training loss decreases over the iterations, indicating that the model is improving in terms of fitting the training data.

The **validation loss** measures the average loss on a separate validation dataset, which is not used for training. It serves as an indicator of how well the model generalizes to unseen data. In the given results, the validation loss fluctuates but remains relatively stable over the iterations. A stable validation loss suggests that the model is not overfitting to the training data and is generalizing reasonably well.

The **accuracy** metric represents the proportion of correctly predicted samples compared to the total number of samples in the dataset. In the provided results, the accuracy remains constant at 0.621713 throughout the training process. This indicates that the model is consistently predicting correctly on approximately 62% of the samples.

It seems that the model is not improving significantly during the training process. The training loss decreases, but the validation loss and accuracy remain relatively stable. This could indicate that the model might be underfitting the data, as it is not able to capture the underlying patterns effectively. To improve the model's performance, further analysis and experimentation, such as tuning hyperparameters, adjusting the model architecture, or increasing the training data, may be necessary.

The fine-tuned model achieved an evaluation accuracy of approximately 64% on the **test dataset**. This means that out of the total samples in the test dataset, the model correctly predicted the correct answer for approximately 64% of them. While an accuracy of 64% may indicate that the model is making correct predictions on a significant portion of the test data, it also suggests that there is room for improvement.

The implemented **test function** predicted the response for the two examples provided by the professor. In the case of the first example, the prediction of 1 aligns with the expected response. However, in the second example, the model incorrectly predicted a response of 1 instead of the correct response of 0.