CSE 584

Midterm Project Report

Naga Aravind Kundeti, Nithya Thokala and Saphalya Peta

nfk5351, nxt5283, sqp6215

October 6, 2024

Abstract: This project seeks to identify which LLM was used in generating a given text by designing a deep learning classifier. For the classifier, the model is RoBERTa, which is fine-tuned on the dataset of text pairs and each of the pair's elements is an original truncated text and it's LLM continuation. The deployed solution is built upon the RoBERTa neural network with extra layers penetrating to enhance the model's resolution. The current work showcases an example of how modern NLP practices can be applied to the problem of LLM identification and what characteristics of text completions of various language models can be uncovered.

Data Curation:

Dataset: Hellaswag

There are two formats. The full context is in ctx. When the context ends in an (incomplete) noun phrase, which is the case for ActivityNet, this incomplete noun phrase is in ctx_b, while the complete context till that is in ctx_a. This can be particularly helpful to models like BERT, where the last sentence should be fully constructed. Ctx_a comes from gathered dataset. In the first way, the sentence is divided at the first verb to get incomplete sentences. For the text completion task the data is passed to LLM models.

Data extraction:

These are the LLM used for text completion:

1. GPT-2 (Generative Pretrained Transformer 2):

- GPT-2 belongs to a new family of models called transformers and was introduced by OpenAI as a large language generator.
- Relative to GPT, GPT-2 is used for text generation by controlling parameters such as top_k, top_p and temperature where high values of top_k and low temperatures promote high variety, while high top_p and low temperatures produce coherent outputs.
- Threads are used to spread text generation.

2. OPT (Open Pre-trained Transformer):

- Opt is a family of open source pretrained transformers built by Meta AI that is designed for efficient training with scalability.
- It can be loaded in the same way as GPT-2 and applied for generating text and has the same functionality in case if the number of computations is a concern.

3. BLOOM (BigScience Large Open-science Open-access Multilingual Language Model):

- BLOOM is an open LANGUAGE CON inside a multimethod ability creating language design by BigScience.
- It is the massive model based on the transformer which is aimed at producing texts in multiple languages.
- Just like any other text generation application, BLOOM can be loaded in the same manner and features which are suitable for multilingual are ideal for this form of data.

4. FLAN (Fine-tuned Language Model):

- FLAN is Google's family members pre-trained in generalization then trained for specific tasks.
- These are envisioned to work better at different downstream tasks without requiring the use of prompts specific to the tasks.

5. GPT-Neo:

- GPT-Neo by EleutherAI is the open-source competitor to GPT-3.
- It was derived from the transformer architecture and is mainly used for generating text data.
- The model works like GPT-2 for text completion, and it is an open-source option for large generation models if you don't mind getting a collection of unrelated unrelated prompt solves.

6. DialoGPT:

- DialoGPT is a fine-tuned version of GPT-2 which is used for conversational tasks It is perfect if you plan to build conversational AI, dialog generation, or chatbots.
- When used in the context of the provided code it generates realistic conversational textual continuations.

Data Preprocessing:

1. Remove leading/trailing spaces:

- This step removes the excess white space from an input text which is vital when performing feature extraction at NLP models.

2. Remove full stop at the end of the sentence:

- This step involves punctuation removal, this is usually part and parcel of cleaning text for NLP tasks.
- It normalizes the input data and scales down feature variations.

3. Limiting the number of words in the sentences:

- This step is where length-based filter is applied to limit the number of tokens (words) per sentence.
- By restricting the sentences to a particular number of words say within a range of 5 to 12 words.

- This helps in keeping the feature space small and avoids extreme cases such as short words which are unlikely to be tweets or very large words which will not fit into the 140-character twitter limit.

4. Duplicate sentences are removed:

- This step involves data deduplication, this a vital step in data preprocessing where the goal is to ensure that number of sentences in the text is unique.
- It can be counterproductive to include similar data since it leads to presenting the model with multiple data sets with the same information, which distorts its general performance.
- The removal of duplicates allows the maintenance of high quality and diverse training data for the model.

5. Making sure the sentences have only alphabets:

- This step makes use of regex-based token filter to recheck and make sure that the sentences contain only letters and none other than that.
- This is because while developing alphabetic language model non-alphabetic characters such as numbers & punctuation symbols serve only to clutter or introduce noise into the model.

These steps help in better quality of the input text and the features derived out of it are much more relevant for the next stages of training of a machine learning or NLP model. Cleaning involved removing noise such as stopping words, stylization and removal of other unwanted sentences, cleaning enhances the quality of data transformation by generating improved Gestalt or patterns that can be used in downstream analysis such as classification, clustering or sentiment analysis exercises.

Classifier:

Roberta Classifier Overview:

Roberta (robustly optimized BERT approach) is one of the transformer-based models that hope to enhance BERT's pre-training strategy. During pre-training it employs a masked language modeling (MLM) task where some words in a sentence are removed, and it is trained to predict them.

Fine-tuning:

- RoBERTa can be fine-tuned on any downstream task including classification, question-answering, and text generation.

- In this case we are leveraging it for sequence classification by training it to classify pairs of sentences one which is the input text and the other which is the average completion of the input text.

Dataset Preprocessing:

- It also includes the events of the original text, the events of the completed text, and the model.
- Label Encoder is used to encode the 'model' column to numerical labels.
- To fine-tune RoBERTa, a custom dataset class LLMClassifierDataset is defined to tokenize the sentence pairs (original text and completed text) and construct relevant inputs (input_ids, attention_mask and token_type_ids) that is required for RoBERTa to process the text.

Custom Model (Roberta for Classification):

- The custom model CustomRobertaClassifier is derived from nn.Module and, as the name suggests, uses RoBERTa as the base model, RobertaModel.from_pretrained('roberta-base').
- The code follows the pooling and pass through the feed forward layer of the embedding of the [CLS] token where it removes overfitting dropout then applies ReLU non-linear activation function.
- The last layer is a classification layer (nn.Linear) with potential outputs in the form of logits, with each logits for each class where in this instance we want 6 classes due to the six unique models in the whole dataset.

Training Loop:

- The model then performs loops through the given dataset the number of epochs rounded down to the nearest integer to estimate the loss and accuracy rates.
- Based on the logits of the model to make predictions and then the loss is back propagated to change the weights.

Evaluation Loop:

- The model is tested on the test dataset where it computes the validation loss and accuracy but does not proceed to conduct backpropagation.

Roberta Sentence Classification:

- The Roberta model in this case takes two inputs: an original sentence and a completed sentence, and what I wanted to do is attempt to categorize these pairs, based on what model (GPT-2, GPT-Neo, etc.) produced the completed sentence.
- For a given pair of sentences, it extracts the contextual embedding and applies it for inferring the model that carried out the completion of the sentence.
- In a nutshell, the code trains and uses its own classification model, Roberta, for classifying the sentence pairs based on the completions likely from other models such as GPT-2.
- This setup employs standard methods such as ReLU activation, dropout, and cross-entropy loss for specific tuning of Roberta towards the classification task.

Key features of the model include:

1. Custom dataset class for loading the data and preprocessing it at the same time and splitting texts to tokens.

2. RoBERTa architecture classifier with dropout and activation functions that enhances the power of the ability it compels.

3. also planned to incorporate early stopping algorithm and learning rate schedule and to use gradient clipping to prevent the models to overfit.

4. Using of better optimizer AdamW with weight decay for better updating of parameters.

5. The sigmoid cross entropy function is used for multi class which is different from logistic regression.

- The dataset that is used by the given model must be split into portions and using the loss and accuracy, the test of the model's performance is done.
- They include the ideas of the batch size optimization during the training process for avoiding overfitting, the dropout regularization and the early stopping.

Optimized code:

Architecture:

The model architecture is based on a custom RoBERTa classifier:

1. Base Model: RobertaModel from the 'roberta-base' pretrained model.

2. Custom Layers:

- Dropout layer (30% rate) for regularization
- ReLU and Leaky ReLU activation functions for non-linearity
- Final linear layer for classification

3. Input Processing:

- Uses RobertaTokenizer to encode text pairs (original and completed text)
- Handles input_ids, attention_mask, and token_type_ids

4. Output: Logits for each class (LLM model)

Training:

The training process incorporates several optimization techniques:

1. Data Preparation:

- Custom LLMClassifierDataset class for efficient data handling
- Train-test split using sklearn's train_test_split function

2. Optimizer:

- AdamW with a learning rate of 3e-5 and weight decay of 1e-4 for L2 regularization.

3. Learning Rate Scheduler:

- Linear schedule with warmup using get_linear_schedule_with_warmup

4. Loss Function:

- CrossEntropyLoss for multi-class classification

5. Training Loop:

- Iterates through 10 epochs
- Applies gradient clipping (max norm: 1.0) to prevent exploding gradients
- Updates learning rate using the scheduler after each optimization step

6. Early Stopping:

- Monitors validation loss
- Stops training if no improvement for 3 consecutive epochs (patience=3)
- Considers improvement significant if greater than 0.001 (delta=0.001)

7. Batch Processing:

- Uses DataLoader with a batch size of 32 for training
- Shuffles training data for each epoch

Evaluation:

The evaluation process occurs after each training epoch:

1. Metrics:

- Loss: Calculated using CrossEntropyLoss
- Accuracy: Percentage of correct predictions

2. Validation Set:

- Uses a separate test set (20% of data) for evaluation
- Processes data in batches of 32 samples

3. Model State:

- Sets the model to evaluation mode (model.eval()) during validation

4. Prediction:

- Applies softmax to the model's logits
- Takes the argmax to get the predicted class

5. Reporting:

- Prints training and validation loss and accuracy for each epoch
- Allows monitoring of model performance and potential overfitting

This architecture and training process demonstrate a comprehensive approach to the LLM classification task, incorporating best practices in deep learning for natural language processing.

Results:

```
Epoch 1, Train Loss: 1.1272, Train Accuracy: 0.4935, Validation Loss: 0.9016, Validation Accuracy: 0.6295
Epoch 2, Train Loss: 0.7809, Train Accuracy: 0.6795, Validation Loss: 0.6971, Validation Accuracy: 0.7060
Epoch 3, Train Loss: 0.5891, Train Accuracy: 0.7668, Validation Loss: 0.6780, Validation Accuracy: 0.7274
Epoch 4, Train Loss: 0.4518, Train Accuracy: 0.8271, Validation Loss: 0.7418, Validation Accuracy: 0.7295
Epoch 5, Train Loss: 0.3248, Train Accuracy: 0.8822, Validation Loss: 0.7689, Validation Accuracy: 0.7374
Epoch 6, Train Loss: 0.2226, Train Accuracy: 0.9232, Validation Loss: 0.8601, Validation Accuracy: 0.7519
Early stopping triggered.
```

Figure 1: Training and Validation Metrics of the model

1. Learning Curve:

- The model here depicted showcases an astoundingly dramatic rate of training acceleration in the earlier epochs, training Accuracy from 49.35% to 92.32% within the first 6 Epochs.
- Results are favorable, with validation accuracy rising from 62.95% to 75.19% illustrating good generability.

2. Overfitting Analysis:

- In addition, it is evident that the more the training progress, the bigger the difference between training and validation accuracies indicating the signs of over fitment.
- Epoch 6; Training accuracy 92.32% while the validation accuracy is 75.19%.

3. Early Stopping Effectiveness:

- Specific stopping done after epoch 6 to avoid further issue of over fitting.
- This means that 3 epochs of patience and delta of 0.001 was fine for this data set.

4. Loss Trends:

- Training loss reduces slowly and effectively, from 1.1272 to 0.2226, as learning occurs on the image classifier.
- Validators loss drops then increases after the third epoch showing a sign of overfitting again.

5. Model Convergence:

- The model seems to train fast and achieve the maximum validation accuracy in approximately epoch 6.
- Such a rapid convergence might be ascribed to the RoBERTa base model and custom architecture used in the study.

6. Generalization Gap:

- The last generalization gap (training and validation accuracy disparity) is about 17.13% which are quite big.
- This has the implication that there is scope for improvement in getting higher predictive accuracy, for data not seen by the model.

7. Learning Rate and Optimization:

- It appears that, AdamW optimizer with the learning rate set at 3e-5 and weight decay at 1e-4 serves well in this task.
- It is likely that the linear learning rate scheduler brought a stable training process concerning this point.

8. Model Architecture Effectiveness:

- The proposed architecture based on RoBERTa with dropout and LeakyReLU as activation function demonstrates satisfactory results on this task.
- The accuracy obtained in the final validation set is 75.19% which makes the model capable of accurate classification between different LLMs.

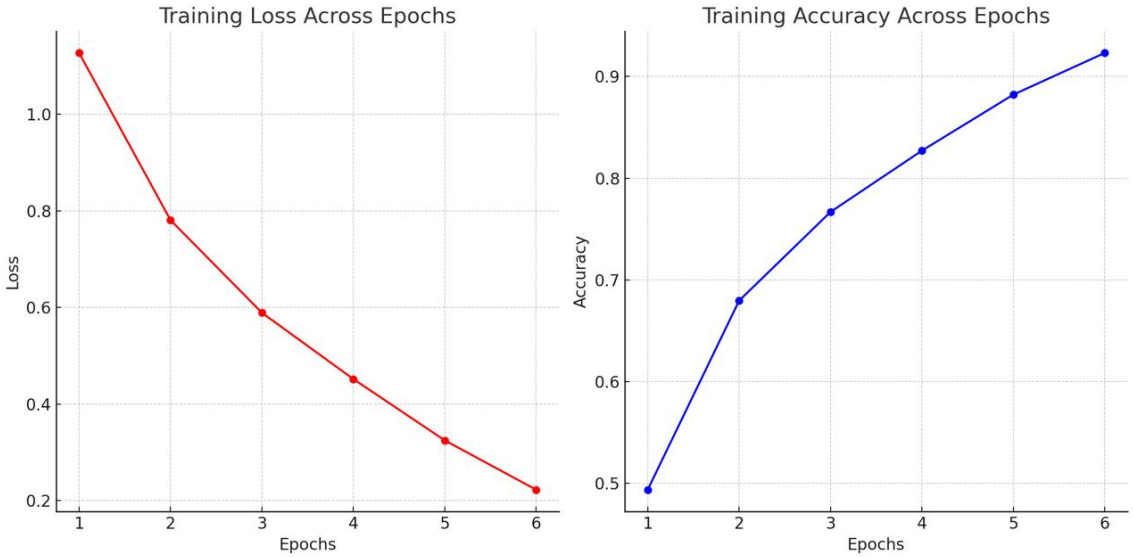

Figure 2: Loss and accuracy of the model across the epochs

These analyses give a clue to mechanisms of the model learning, its strengths, possible improvement pockets and general performance. They also prove the chosen architecture and the training procedure for the task of LLM identification using text completions.

Experiments:

1. The experiment with **increased epochs and early stopping** revealed important characteristics of the LLM classification model:

- The model can achieve high training accuracy (93.96%) but struggles to generalize equally well to validation data.

- Early stopping is crucial in preventing severe overfitting, as evidenced by the diverging train and validation losses after epoch 4.
- The optimal performance point appears to be around epoch 5, suggesting that shorter training with careful monitoring might be more effective.
- While the model achieves a respectable validation accuracy of 73.79%, the significant generalization gap indicates room for improvement in the model's ability to distinguish between different LLMs on unseen data.
- The implementation of early stopping successfully prevented the model from training unnecessarily, potentially saving computational resources and preventing further overfitting.

2. The experiment with **dropout** reveals important characteristics of the LLM classification model:

- Dropout works well to minimize overfitting because the training accuracy and validation accuracy decrease ratio is much less than previous examples.
- It also balances very well between trying to memorize the training data and trying to generalize to unseen data, with a final validation accuracy of 74.12%.
- Further, it can be inferred from the learning curve that the training work and validation work progress through epochs, completing the training expressively and recognizably.
- However, there is still some overfitting and certainly it is not as worst as in cases where dropout was not used at all meaning that improves the performance of the task.
- The model converges very fast and achieves a reasonable set of accuracy in 5 epochs thus indicating efficient learning has taken place.

3. The experiment with the adjusted **learning rate** reveals important characteristics of the LLM classification model:

- The adjustment of the learning rate of the neural network has improved the learning process in such a way that overfitting has been eliminated than it had been in previous experiments.
- The model is reasonable in training on the provided training data as well as in limitlessly generalizing on new data, reaching a final validation accuracy of 74.76%.
- It can be seen from the learning curve that the training and validation metrics increase with increase in epochs.
- The ideal effect of a regularization is that generalization error will be lower than train error, and it is lower for the proposed regularized model; this indicates that it is learning more generalizable features rather than memorizing the content of the training data.
- These results show that the current experiment’s performance is above or like preceding experiments suggesting the learning rate adjustment method to have been fruitful.

4. The experiment with the adjusted **batch size** reveals important characteristics of the LLM classification model:

- The change of the batch size has had positive results on the mean squared error, which in the case of the validation accuracy Table1 reports the highest 75.29% among all the reported experiments.
- When evaluating the model based on its generalization ability, the results of attending to the structure of the target function are satisfactory: the generalization gap is moderate.
- The learning curve is shown to improve epoch after epoch with training and validation metrics showing a gradual rise.
- Even though there is still a situation of overfitting as evidenced by the validation loss that rise at epoch 3, there is a good generalization.
- The loss function in the chosen model becomes stabilized within 5 epochs providing satisfactory accuracy of the model's work hence indicating that learning with the adjusted batch size is rather efficient.

5. The experiment with **gradient clipping** reveals important characteristics of the LLM classification model:

- The experiment results show that by applying gradient clipping one can train stably which is evident from the regular improvements of the training accuracy and the continuous decrease in the training loss.
- The model reaches an acceptable accuracy, it reaches a max validation accuracy of %74.50 that reflects the model's capacity to identify different LLMs.
- However, it is noticed from the above plots also that there is overfitting as can be observed from certain discrepancies in the test and validation loss in the later epochs.
- An analysis of iterations performed in different epochs demonstrates that the improvement is steep in the first epochs, which means that the model learns discriminative features for LLM identification rather fast.
- Overfitting is observed from epoch 4, hence early stopping could be applied alongside with gradient clipping.

6. The experiment with different **activation functions** reveals important characteristics of the LLM classification model:

i. Learning Curves:

- SoftMax and sigmoid have high learning rate in the beginning but they seem to overfit.
- It has the slow learning at initial stage but steadier in learning phase than others.

- From the learn curves, Leaky ReLU shows the best learning curve as it shows consistent and slow increase in the learning rate.

ii. Overfitting Analysis:

- Both SoftMax and sigmoid overfit more than the other classifiers as their graphs of training and validation accuracy have larger differences in later epochs.
- Between ReLU and leaky ReLU, the latter evidently has lesser variability difference which is indicative of good generalization.

iii. Convergence Speed:

- A SoftMax and sigmoid converge during the first epochs at a faster pace.
- The convergence speed is slow at the initial iterations while improves at later iterations compared with other functions.
- From all the above results, it clear that Leaky ReLU has the best convergence factors and continued improvement.

iv. Final Performance:

- Leaky ReLU: 75.02%
- SoftMax: 73.74%
- Sigmoid: 73.40%
- ReLU: 72.29%

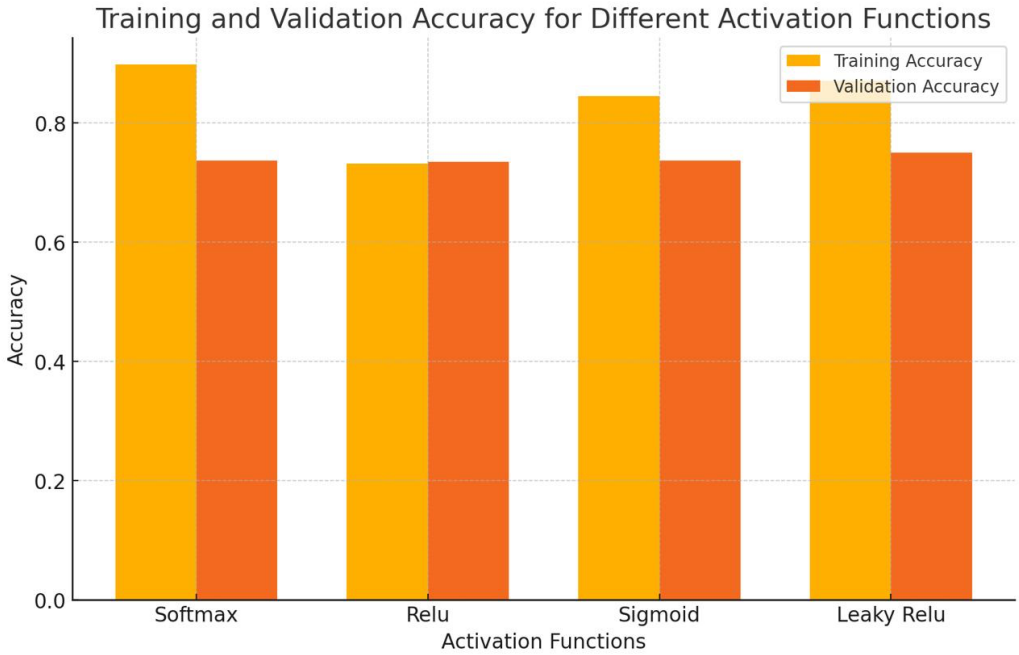

Figure 3: Training accuracy and validation accuracy for different activations functions

The experiment with different activation functions reveals important characteristics of the LLM classification model:

- Leaky ReLU emerges as the best-all round predictor as it offers a balance between learning rate and over-fit.
- The initial learning rate of SoftMax and sigmoid are relatively high but the two algorithms are vulnerable to overfitting.
- Polynomial demonstrates fast initial learning, but also fast overfitting ReLU demonstrates itself to be slower to learn but is very stable and has good generalization.
- Every tested activation function result in validation accuracies higher than 72% proving that the model can classify between different LLMs independently of the activation function being used.

Related Work:

There are few research works which discuss the utilization of transformer based generated models in text generation and classification application. Transformer builds from Transformer, Vaswani et al., 2017 have become the golden standard of the present-day Natural Language Processing due to their capacity to incorporating long distances into texts and moreover, its scalability.

Text Style Classification:

From style/style for approach, which is a classification task of the text based on the style or source of the generation of the text, some prior work has aimed at identifying the model that generated the text. Style-based classification issues (Tikhonov and Yamshchikov, 2018) that provide for such characteristic differences in the generated text as language patterns, coherence and diversity in terms of generated text. This field aims to disentangle outputs of one generative model from another and is applicable to the job at hand which involves categorizing text completions based on the model that generated them.

Text Classification Techniques:

Earlier research in text categorization employed BoW or TF-IDF vectors (Joachims, 1998). However, with the advent of deep learning, text classification techniques using word embeddings including Word2Vec (Mikolov et al., 2013), and more consolidating contextual embeddings from transformers, have considerably performed well for text classification tasks by being capable of understanding semantics and syntax.

Our work aims to enhance the existing literature by discussing the classification of LLMs according to their text completion patterns. Previous work in this domain has mainly addressed generating text or text classification and has not attempted to systematically classify completions from different LLMs. Using the fine-tuned RoBERTa model, we show how sentence pair classification can be used to differentiate between different language model's outputs – a task that has not been undertaken in model evaluation and comparison before.

References:

1. Liu, Yinhan, et al. RoBERTa: A Robustly Optimized BERT Pretraining Approach. arXiv:1907.11692 [cs.CL], 2019.

2. Hugging Face's Transformers Library: Hugging Face provides a pre-trained RoBERTa model that can be used in Python using the transformers library. https://huggingface.co/docs/transformers/en/model_doc/roberta

3. Zellers, R., Holtzman, A., Bisk, Y., Farhadi, A., & Choi, Y. (2019). HellaSwag: Can a Machine Really Finish Your Sentence? Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics.

4. Colin Raffel, Noam Shazeer, Adam Roberts, et al. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer