CAN THO UNIVERSITY
COLLEGE OF INFORMATION AND COMMUNICATION TECHNOLOGY

GRADUATION THESIS
BACHELOR OF ENGINEERING IN
INFORMATION TECHNOLOGY
(HIGH-QUALITY PROGRAM)

Transformer Sentiment analysis

Student: Huỳnh Phi Hồng
Student ID: B2005839
Class: DI20V7F1 (Cohort K46)
Advisor: Ph.D Lam Nhut Khang

Can Tho, 11/2023

CAN THO UNIVERSITY
COLLEGE OF INFORMATION AND COMMUNICATION TECHNOLOGY
DEPARTMENT OF INFORMATION TECHNOLOGY

GRADUATION THESIS
BACHELOR OF ENGINEERING IN
INFORMATION TECHNOLOGY
(HIGH-QUALITY PROGRAM)

Transformer Sentiment analysis

Student: Huỳnh Phi Hồng
Student ID: B2005839
Class: DI20V7F1 (Cohort K46)
Advisor: Ph.D Lam Nhut Khang

Can Tho, 11/2023

Abstract

Sentiment analysis (or opinion mining) is a natural language processing (NLP) task involving analyzing and determining the expressed sentiment in a sequence of text in order to determine whether the text is positive, negative or neutral. It has crucial role in various applications like feedback analysis, market research and media monitoring, which helps organizations gather insights to improve their businesses.

In the recent years, the rapid development of deep learning models, and Transformers models in particular, have shown noteworthy performance in various NLP task including sentiment analysis. Leveraging self-attention mechanisms, Transformers model could capture contextual meanings in a long sequence. This makes them well suited for the task in this study.

The study will explore the application of Transformers model in this task by training a BERT, distilBERT, RoBERTa models to perform sentiment analysis on customers' feedbacks. The models are trained on IMDB dataset to distinguish whether a review is positive or negative. Then we conduct validating on data using metrics like accuracy and f1 score to evaluate our models and performing comparison. From the comparison, we find that RoBERTa is the most accuracy models, attaining 95,4% accuracy on the test set.

Chapter 1: Introduction to Transformer model

Transformer[1] is a deep learning model which was introduced by Google researchers in 2017. It does not compute the input and output representations by using Recurrent Neural Network (RNN) or Concurrent Neural Network (CNN) mechanic but instead using encoder-decoder structure and self-attention mechanism which weights differently for each part of the input data.

The model is mainly used in field of Natural Language Processing (NLP) like for text classification, summarization, translation, text generation, with many models such as Bert[2] ,GPT-3[3] , RoBERTa[4] achieve state-of-the-art results. Another common field for Transformer model is Computer Vision (CV) for tasks like object detection, image classification, video classification.

1.1. General Architecture

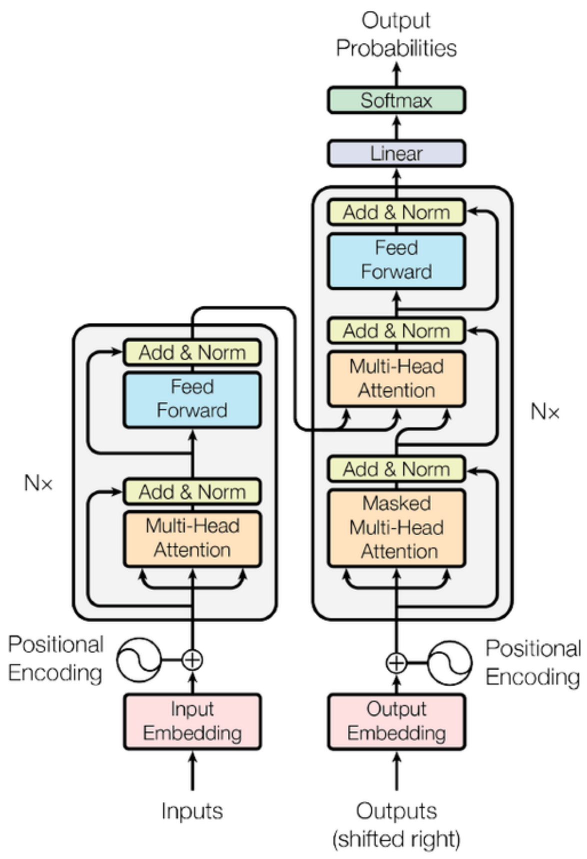


Figure 1 Transformer architecture[5]

The model architecture is generally composed of two components: Encoder and Decoder

- Encoder: It accepts input that represents text and then process them into numerical representation (features) that contains information about which part of the inputs are relevant to each other. The output is then passed into decoder.
- Decoder: It uses the Encoder's representation which incorporated contextual information alongside with its usual sequence input to generate a sequence.

1.2. Input Embedding layer

A word embedding layer can be thought of as a lookup table to grab a learned vector representation of each word. Neural networks learn through numbers so each word maps to a vector with continuous values to represent that word. However, Transformer processes words in parallel, so with just word embedding the model can't tell where the words are. Therefore, we need a mechanism to inject information about where the words are in the input vector.

1.3. Positional Encoding

The creators of the transformer came up with a solution using sin and cosine functions to obtain information about positions into the input. At even dimension indices the sine formula is applied and at odd dimension indices the cosine formula is applied

$$PE_{(pos,2i)} = sin\,(pos/n^{2i/d_{model}})$$
$$PE_{(pos,2i+1)} = cos\,(pos/n^{2i/d_{model}})$$

Where:

pos: Position of an object in the input sequence

d_{model}: Dimension of the output embedding space

$PE_{(pos,j)}$: Position function for mapping a position *pos* in the input sequence to index (k,j) of the positional matrix.

n: User-defined scalar, set to 10,000 by the authors.

i: Used for mapping to column indices $0 \leq i < d/2$, with a single value of *i* maps to both sine and cosine functions

1.4. Encoder

Encoder can consist of many similar encoder layers. Each encoder layer consists of two main components: multi head attention and feedforward network. There are also residual connections and layer normalization

- **Multi-Headed Attention**

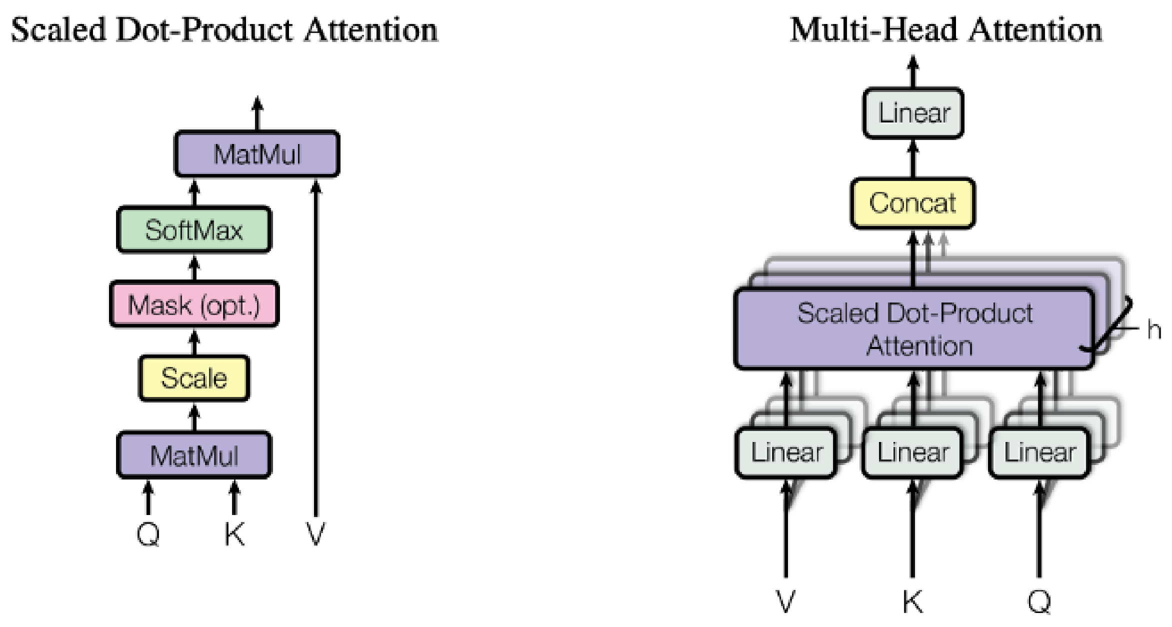


Figure 2 Multi-headed Attention[6]

Multi-headed attention in the encoder applies a specific attention mechanism called self-attention. Self-attention allows the models to associate each word in the input, to other words. To achieve self-attention, we feed each word of the input sequence into 3 distinct fully connected layers to create the query, key, and value vectors.

- Query vector: vector used to contain the information of the searched word. It likes the query of google search.
- Key vector: a vector used to represent the information of the words which are compared to the searched word. For example, the website that google will compare with the Keyword that you search.
- Value vector: The vector represents the content, the meaning of the words. It's like the web page content is displayed to the user after the search.

After feeding the query, key, and value vector through a linear layer, the queries and keys go through matrix multiplication to produce a matrix. the scores of the matrix then get scaled down by getting divided by the square root of the dimension of query and key. It then uses the softmax function to normalize the scores within matrix in range 0-1. Finally, the matrix is multiplied by the value matrix generated beforehand to obtain the attention based matrix.

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

We want the model to detect various patterns from each word of the input. From each self-attention we learned one pattern; Therefore, to learn more pattern, we want to apply more self attention. To do that, we need to split query, key, value into N vectors before applying self attention. Each split is then applied self-attention individually.

- **Residuals Connection and Normalization Layer**

In the architecture of transformer models, residuals connection and normalization layer are used everywhere. those techniques that help the model train faster and prevent the loss of information during the process.

1.5. Decoder

Decoder receive from encoder 2 vectors key and value to decode source sequence to generate another sequence. The architecture of the decoder is very similar to the encoder, except that there is an additional multi head attention in the middle used to learn the relationship between the word being generated and the words in the source sequence.

The process of the masked multi-head attention is similar to that of the regular multi-head attention. However, we need to mask the words from the future that have not been generated by the model. To do this, we simply multiply a vector that contains 0 and 1 values after the scaling of multiplication of matrices Q and K. As a result, the model will only pay attention to all the words before the position of the next generating word.

Chapter 2: Methodology

The chapter describes the methodology used in this study for sentiment analysis.

2.1. Choosing dataset

IMDB dataset having 50K movie reviews for natural language processing or Text analytics. This is a dataset for binary sentiment classification containing substantially more data than previous benchmark datasets. The dataset provide a set of 25,000 highly polar movie reviews for training and 25,000 for testing which are classified into positive and negative sentiment labels. The training set is split into 22500 reviews for training, 2500 for validation during training. For evaluation after training is finished we randomly pick 5000 reviews in testing set.

2.2. Models experiment

Experiments were carried out using Google Colab’s GPU hardware accelerator platform. The extracted features for the training set were fed to the input of the tuned models. The batch size and learning rate were set to 16 and $2*10^{-5}$, respectively. The models were optimized using the AdamW optimizer. The models were trained for five epochs

The BERT-base-uncased model consisting of twelve layered transformer blocks with each block containing twelve head self-attention layers and 768 hidden layers resulting in a total of ≈ 110 million parameters was used. A single sentence was fed into the model at a time. The input sentences were split into tokens and mapped to their indexes using the BERT tokenizer library, indicated as input ids. The [CLS] (classification token) and [SEP] (separate segment token) were appended at the beginning and end of every sentence, respectively. An input attention mask of fixed length with 0 indicating padded tokens, and 1, indicating unpadded tokens was applied. Each of the transformers indicated received a list of token embeddings and produced a feature vector of the same length at the output. The output of [CLS] for the 12th transformer layer containing vector transformations of prediction probabilities was used as aggregated sequence representation from which classifications were made.

The RoBERTa-base model made up of twelve transformer layers with 768-hidden layers and twelve attention heads, and having a total of 125 million parameters was used in the experiment. The RoBERTa tokenizer was used to encode the input texts into tokens and designated as the input ids. These ids were padded to a fixed length to avoid variations per row. Features were then extracted from these tokens from which sentence pair classification was made.

The DistilBERT-uncased model was used in this experiment. It contained six transformer layers, 768-hidden layers, and twelve attention heads. After tokenizing the input texts and converting the tokens into input ids, they were padded and fed into the DistilBERT model for the multiclassification task.

2.3. Evaluation

For this task, we use accuracy and f1 score metrics for our evaluation of the model.

$$F1\ Score = \frac{2TP}{2TP + FP + FN}$$

$$Classification\ accuracy = \frac{TP + TN}{TP + TN + FN + FP}$$

Where:

- TP (True positive): predicted positive sentiment and actual label of text sequence is positive.
- TF (True negative): predicted negative sentiment and actual label of text sequence is negative.

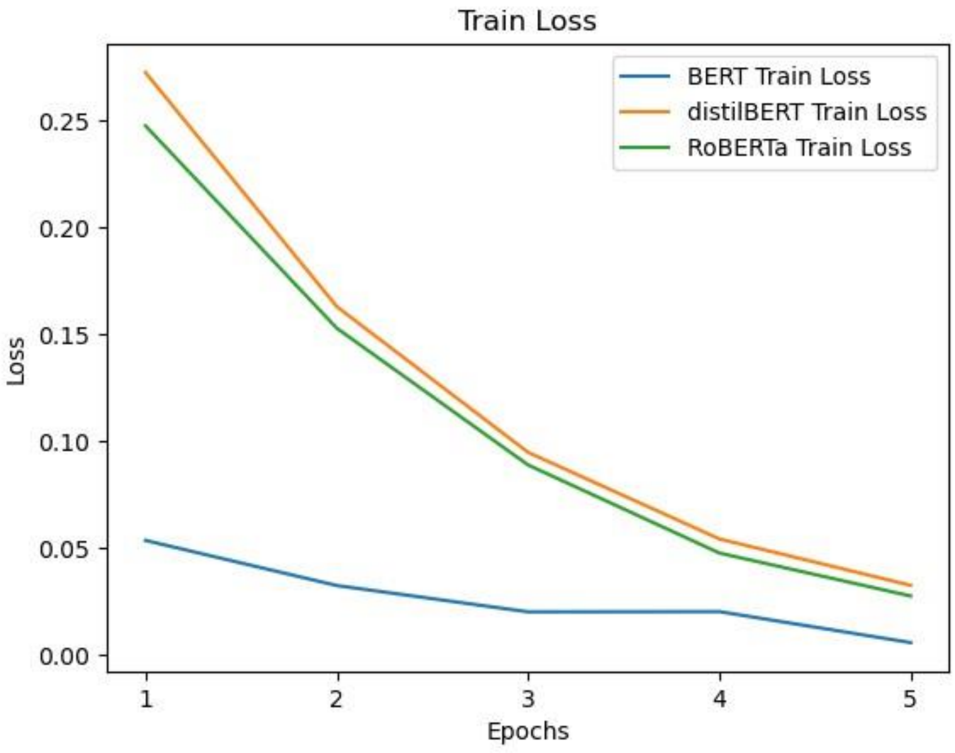
- FP (False positive): predicted positive sentiment but actual label of text sequence is negative.
- FN (False negative): predicted negative sentiment but actual label of text sequence is positive.

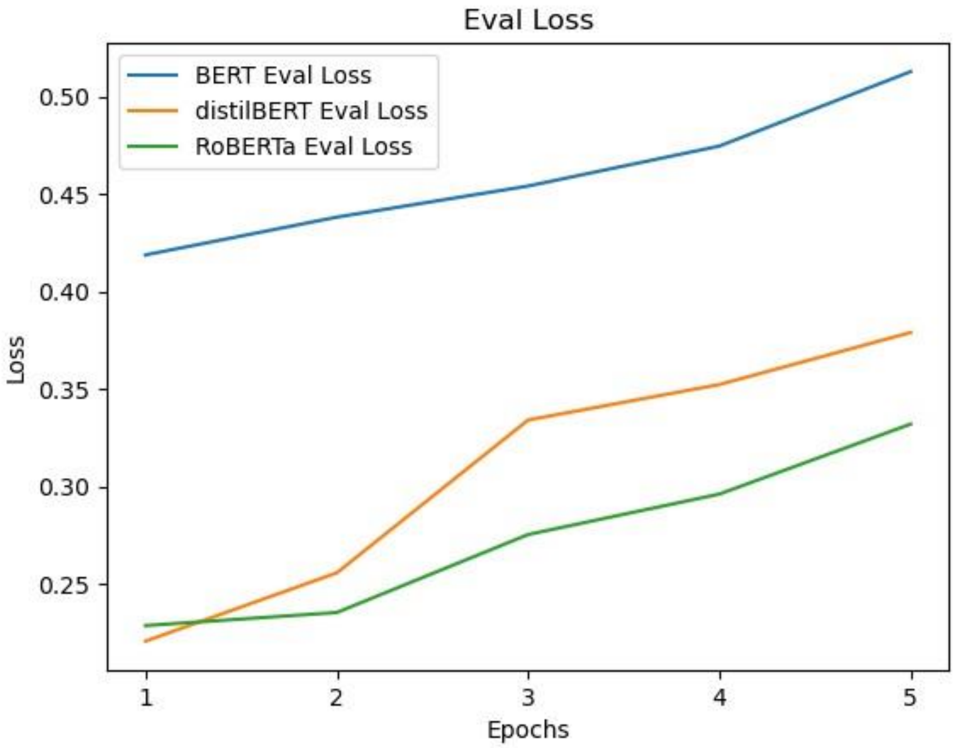
Chapter 3: Experimental result

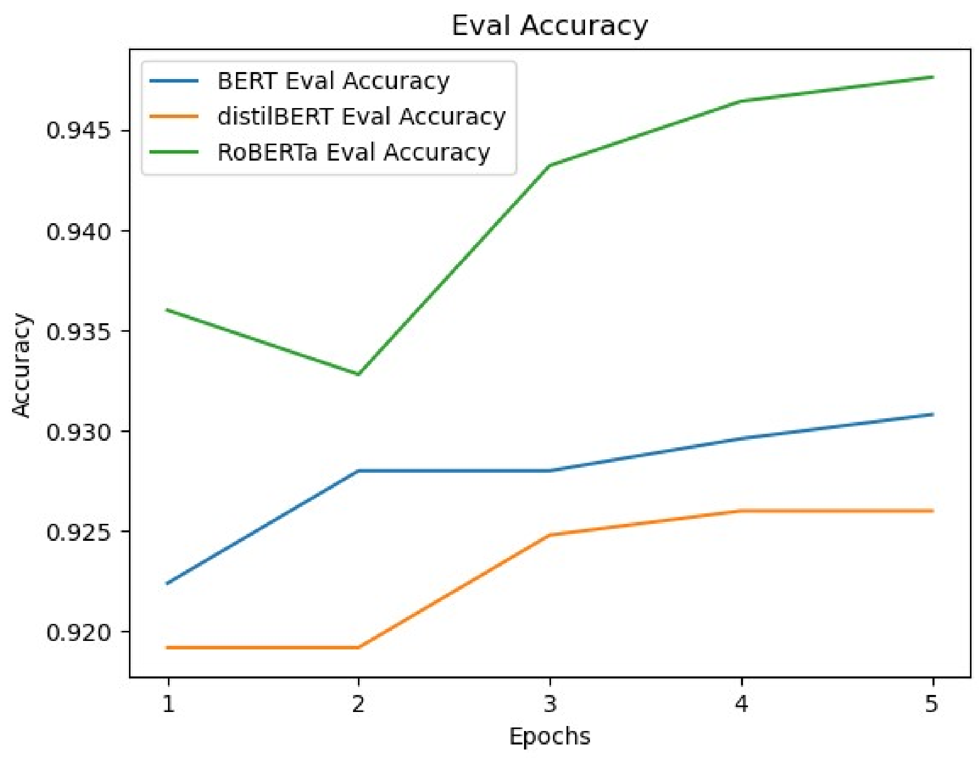
	BERT	RoBERTa	DistilBERT
Model size	417.7 MB	475.6 MB	255.4 MB
Training time	3:00:59	3:16:00	1:27:16
Testing loss	0.4734	0.2918	0.3670
Testing accuracy	0.9382	0.9544	0.9294
Testing F1 score	0.9550	0.9386	0.9300
Sample per second	28.497	30.675	55.762

Table 1 Training results of models

From the results shown above, we could see that RoBERTa models achieves the best accuracy with 95.44%, following by BERT and DistilBERT with 93.82% and 92.94%. However, DistilBERT as a distilled version of BERT achieves very impressive accuracy despite almost less than twice the size of BERT. It is also notable for very high prediction speed with 55 samples per second and training speed of 1 hour and 27 minutes which is only about half the training time of other models.


Train Loss
BERT Train Loss
distilBERT Train Loss
RoBERTa Train Loss
Loss
0.00
0.05
0.10
0.15
0.20
0.25
1
2
3
4
5
Epochs


Eval Loss
BERT Eval Loss
distilBERT Eval Loss
RoBERTa Eval Loss
Loss
0.25
0.30
0.35
0.40
0.45
0.50
1
2
3
4
5
Epochs



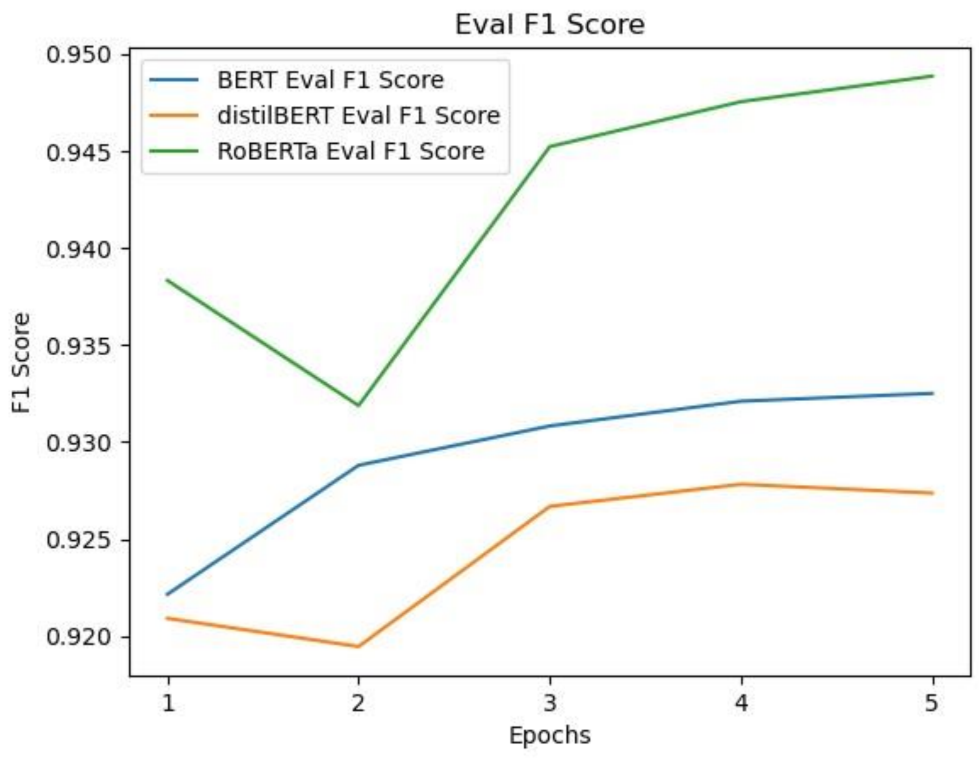


Figure 1 Comparison of various metrics

Overall, the models accuracy are only improved very minimally during the course of training, with RoBERTa increased the most from 93.6% to 94.7% for validation set, which is only approximately 1% increase. This may due to the fact that the pre-trained model is already performative for the task and the number of epochs are not high enough.

Chapter 4: Conclusion

In conclusion, we've implemented various Transformer models to perform sentiment analysis on IMDB dataset. Based on the results we observed, all models are proved efficient in detecting emotion from the texts, with RoBERTa achieves the highest accuracy while distilBERT is the fastest in sentiment analyzing

References:

[1],[5],[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. *Attention Is All You Need*. arXiv:1706.03762v5

[2] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.* arXiv:1810.04805

[3] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. *Language Models are Few-Shot Learners.* arXiv:2005.14165v4

[4] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. *RoBERTa: A Robustly Optimized BERT Pretraining Approach.* arXiv:1907.11692v1