Analysis on GPT and BERT based Transformer models

David Hwang
hwan0259@umn.edu
University of Minnesota
CSCI 8523: AI for Earth
Github Link: https://github.umn.edu/hwan0259/CSCI8363_Transformers

Abstract

GPT and Bert based transformers are some of the most popular models to use in the current NLP landscape. However, it can be hard to distinguish which model to use in each downstream task as there are not many resources that pinpoint the differences between the two and explore the limitations and possibilities of the models. In this paper I conduct a deep dive into the architecture of the 4 transformer based models, 2 that are GPT based: GPT-2 and GPT-Neo and two that are Bert based: Bert and RoBERTa, all models that are used and relevant in modern times. I also run and compare the transformer models on benchmark tasks that they are both able to perform such as sentiment analysis, natural language inference, and question answering. From these results we are able to see on the relevant test sets BERT performs the best on natural language inference with an accuracy of 0.839, RoBERTa outperforms all other models in question answering with a F1 of 91.82, and GPT-Neo does the best with sentiment analysis with accuracy 0.94.

1 Introduction

Major developments have been made in the field of Natural language processing (NLP) where the goal is to create AI that learns how to process and understand human language. Transformers is one of the most revolutionary architectures that has come out in recent times that solves NLP problems and has overtaken previous common architectures like RNN and LSTM in this space.

With the release of transformers it paved a new method to solve NLP tasks. Transformers uses a sequence-to-sequence architecture that uses an encoder and decoder. The encoder converts the input into a latent space and then decodes the space into an output. Encoder and decoders have been seen before with RNN and LSTM but the addition that transformers makes is with the self-attention mechanism. The self-attention mechanism helps provide deep context on words in the sequence [3]. This mechanism is improved with the multi-headed attention layer which is a stack of parallel attention layers that helps with parsing the sequences and keeping long term dependency between words [3].

In the wave of transformers there have been various new architectures that were built on top of the transformer architecture. Architectures like BERT and GPT-2 that were created to be pre-trained NLP models that can be used on a variety of language understanding tasks like natural language inference, question answering, semantic similarity, text classification. The BERT architecture only adopts the encoder blocks of the transformers while the GPT architecture adopts the decoder blocks of the transformer which allows for the architectures to solve different NLP tasks.

With so many advancements in the previous years it can be hard to parse through all the models that have been based on the different architectures that have been created and find the disadvantages and advantages that each model provides. In this paper I seek to experiment with BERT and GPT based models — BERT, RoBERTa, GPT-2, and GPT-Neo and fine-tune these models on different benchmark datasets to solve various NLP tasks.

2 History

Natural language processing (NLP) tasks can be divided into sentence level tasks such as natural language inference and paraphrasing or token level tasks such as entity recognition and question answering. The current existing strategies for solving NLP tasks is to use a pre-trained language model which are then used on downstream tasks [4]. Downstream tasks utilizes a dataset which a model is then fine-tuned on to solve the relevant task. Fine-tuning retrains weights from the pre-trained language models in order to get better results on the downstream task. The architecture that can be used for downstream tasks are feature-based architecture and fine-tuning architecture [4].

Feature-based architecture consists of models like ELMo which has pre-trained representations and then feeds the representations into another model that is then fine-tuned on the downstream task. Fine-tuning architecture like GPT and BERT do not need a downstream model and can fine-tune itself on NLP tasks [4, 1]. By using pre-trained models this allows us to bypass the need to train a model from scratch and reuse our pre-trained model on other tasks. This allows us to take on complex NLP problems [1].

Before transformers, some of the most influential work was done with word-embeddings which transform words into vectors to build semantic relationships beteween text [5]. Word2Vec and Glove are common ways to implement word-embeddings which use a shallow neural network to learn word embeddings by pre-training on a large corpus. ELMo improved upon these previous methods and added contextualized word-embeddings which captured the context of a word in its embedding by using bidirectional language models [5]. Created in 2018 this method achieved state-of-the-art performance in NLP tasks such as question answering, sentiment analysis, and named-entity extraction and was a huge step towards understanding pre-training in NLP tasks [5].

Also released in 2018, Bidirectional Encoder Representations from Transformers (BERT) changed the way in which token embeddings captured the context of a sequence. BERT advanced the state of the art (SOTA) for NLP tasks and this was shortly followed by GPT-2 the next year which also achieved SOTA on multiple datasets.

RoBERTa, based on BERT architecture, was released in 2019 and made changes to the existing architecture by removing the next sentence prediction pre-training objective and increasing the size of the corpus in pre-training which improved results over BERT [13].

As the NLP landscape evolved it was found that the best performing models scale with the number of parameters it has [11]. Ultimately, this led to scaling transformer models to a much larger degree resulting in so-called large language models (LLM) having two orders of magnitude greater than GPT-2. GPT-3, released in 2020, has 175 billion parameters 100 times more parameters than GPT-2. LLMs are protected intellectual property of large organizations and the implementation of such models are behind commercial API or are not available to the public. GPT-Neo adopts similar architecture used in GPT-3 was created on the idea to make LLM available to the public [10].

3 Background on Models

3.1 Transformer

The architecture for transformers was released in 2017 with the paper "Attention is all you need". Removing the need for recurrence or convolutions it uses an encoder decoder network with an attention mechanism and achieved state of the art on translation tasks [3].

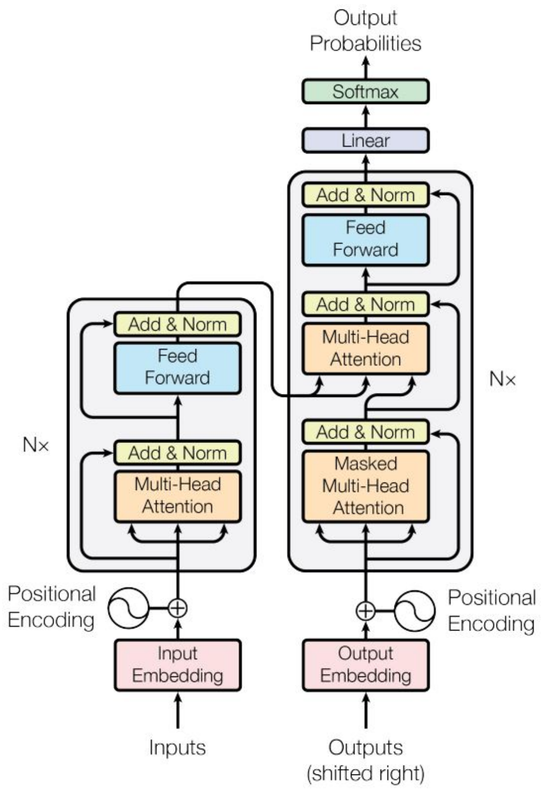

Figure 1: Transformer model architecture

3.1.1 Encoder

The encoder is composed of the multi-head self attention layer and the feed forward neural network. This composition makes up an encoder block which in the paper is stacked 6 times. Layer normalization happens between layers which takes in a residual connection.

For each token in the input sequence an embedding is generated. This happens at the start of the encoder where the input is fed through an embedding algorithm which only happens once through the encoder decoder architecture. A positional encoding vector is then added to the initial embedding in order to help locate where the word is in the sequence which creates the final embedding. The final embedding is a vector which has a size of 512 characters which is then fed to the self-attention layer.

3.1.2 Attention Mechanism

The idea with attention is to identify words in the sequence that are most relevant to the target word. From the encoder's embedding three vectors are generated: a Query, Key, and Value vector. These vectors are created by multiplying the embedding by three separate weight matrices (W_q, W_k, W_v). These weight matrices are set and updated during the training process. For each embedding an attention score is calculated which utilizes these three vectors.

In order to perform this process all at once, transformers put all the embeddings into a matrix X and then multiply the matrix X separately by the weight matrix for W_q, W_k, W_v to get the resulting Query, Key, and Value matrices. The Query matrix is then multiplied by the transpose of the Key matrix and subsequently divided by the square root of the dimension of the key vectors. Then softmax is performed on the resulting matrix to normalize the scores and is multiplied by the value matrix to get the matrix Z which is the self-attention for each of the embeddings. Visualization of this process can be seen below.

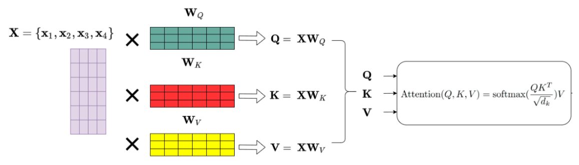
$\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4\}$
$\mathbf{W}_Q$
$\mathbf{W}_K$
$\mathbf{W}_V$
$\mathbf{Q} = \mathbf{X}\mathbf{W}_Q$
$\mathbf{K} = \mathbf{X}\mathbf{W}_K$
$\mathbf{V} = \mathbf{X}\mathbf{W}_V$
Q
K
V
$\text{Attention}(Q,K,V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$

Figure 2: Attention calculation with key, query, value

Multi-headed attention uses layers of this attention mechanism. In the paper "attention is all you need" multi-headed attention utilizes 8 heads which is equivalent to 8 layers of attention. This allows for a more robust method to embed the token in the input. Using 8 heads would create 8 different self attention matrices which would need to be merged together to one self attention matrix. This can be done by concatenating all the attention heads together and multiplying them by a weight matrix to get the final attention matrix which is sent to the feed forward neural network.

3.1.3 Decoder

The decoder is composed of masked multi-head attention layer, a multi-head attention layer, and a feed forward neural network. These layers create a decoder block which is stacked 6 times in the Transformer architecture.

The decoder is an auto regressive model so it predicts the next output from its previous inputs. Multiple passes will be made to generate the sequence from the input given. On the first pass the output from the encoder, which is some contextualized output, is fed to each of the decoders through the encoder-decoder attention layer. The decoder will predict the first word in the sequence and then on the next pass this output will be fed into the decoder blocks as input to predict the next output. With masked multi-head attention only previous tokens in the sequence are visible and future tokens are masked.

3.2 GPT Architecture

3.2.1 Language Models

During pre-training for GPT architecture it is standard to use a standard language modeling objective. Given an unsupservised corpus of tokens $U = \{u_1, .., u_n\}$ an objective can be formed as seen below [1].

$$L_1(U) = \sum_i logP(u_i|u_{i-k}, ..., u_{i-1}; \theta)$$

In the formula k is the size of the context window, P is the conditional probability that is modeled using a neural network where θ are the parameters. Training is done with stochastic gradient descent. In general the objective for language models is to predict the next word in the sequence. This objective has been used in numerous architectures such as RNN, LSTM, transformers [1].

3.2.2 GPT-1

Generative Pre-Trained Transformer (GPT) was published in 2018 and bases itself on the transformer architecture. The key difference is that it only uses the decoder blocks in the architecture (composed of 12 decoder blocks) [1]. When models are trained on large sets of data the common practice is to use data that is labeled. However, the amount of unlabeled data far outstrips the amount of labeled data thus the main advantage of GPT is that it uses generative pre-training to utilize this unlabeled data through unsupervised pre-training [1]. After pre-training is done then discriminative fine-tuning is done on the downstream task. The goal with this architecture is to learn a universal representation which can be utilized on a variety of downstream NLP tasks [1]. This pre-training is a two step process where first a language model objective is learned on the unlabeled text data and then supervised fine-tuning adjusts the parameters of the model on NLP tasks using labeled text data.

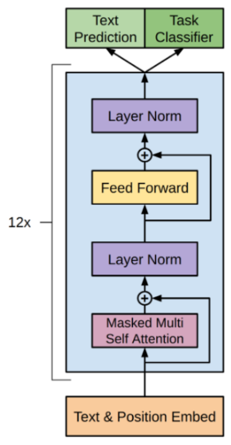

Figure 3: GPT-1 model architecture

4 Models

In this section I will go into an in-depth overview of the models that I use in my project.

4.1 BERT

BERT also known as Bidirectional Encoder Representations from Transformers is a language representation model which was inspired from models like ELMo and was one of the first models to apply the bidirectional training of Transformer as a pre-training objective [4].

It builds from the transformer architecture using just the encoder layers and discarding the decoder layers. It uses 12 encoder blocks. There are two types of BERT that the authors created — Bert-base architecture which stacks 12 encoder layers and Bert-Large which stacks 24 encoder layers. The purpose of stacking the encoder layers is to build upon patterns found in previous layers and extract different features in the input [4].

As BERT was created to handle a variety of NLP tasks the input to the model is a sequence of words. Embedding of these words is done through WordPiece embeddings which contain a 30,000 token vocabulary. When sequences are tokenized a classification token [CLS] is added to the start and sentences are separated by the separator token [SEP]. A positional embedding is added to the WordPiece embedding to indicate the location of the token in the sequence. Pre-training is done on BooksCorpus (800M) and English Wikipedia (2,500M). From wikipedia only text passages are extracted [4].

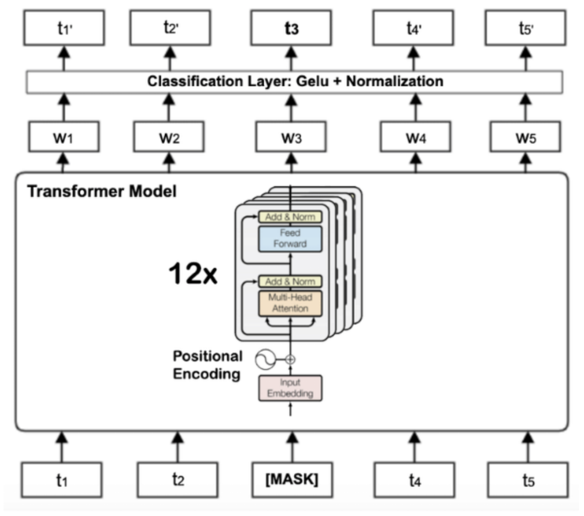

Figure 4: BERT model architecture

BERT was created to address the unidirectionality of language models where tokens can only be input from left-to-right or right-to-left in language models. Even models like ELMo, which uses a bidirectional LSTM, builds its contextual representation from the summation of left-to-right and right-to-left language models which fails to make the representations it generates deeply bidirectional and fail to completely incorporate the context of a sequence [4].

This restriction is resolved by using the transformer architecture which takes in all inputs at once in which we can used a masked language model pre-training objective. This method randomly masks 15% of the tokens, replacing the token with [MASK], where the objective is to predict the correct id of the masked token from the context in the sequence. This allows the representation to use context on the right and left side of the masked word for prediction which allows for a true bidirectional transformer [4].

During the training process BERT is trained on predicting the masked word. However, there is a problem that arises when fine-tuning is done as the [MASK] token does not exist unless specifically doing masked language modeling (LM). So the authors introduce different possibilities for the masked token to mitigate this problem. From the 15% randomly chosen tokens to be masked, 80% are replaced with the [MASK] token, 10% are replaced by a random token, and 10% remain unchanged. The reasoning for including random tokens is to have the LM lean more on the context when predicting the masked word and the reason for keeping tokens the same is to encourage the LM to predict the correct id.

BERT also uses another pre-training objective — binarized next sentence prediction task given two sentences A and B, predict if sentence B follows A or does not. The authors argue that next sentence prediction helps with important downstream tasks such as question answering and natural language inference.

4.2 RoBERTa

Robustly optimized BERT approach (RoBERTa) modifies the pre-training procedure to improve accuracy on NLP Tasks. BERT during pre-training uses masked language modeling and next sentence prediction objective. RoBERTa makes alterations on the masked language modeling (MLM) strategy to include dynamic masking in which the masking pattern is changed every time a sequence is fed to the model when pre-training occurs. This results in a different mask every training instance [13].

Next Sentence Prediction (NSP) is also contested in RoBERTa. It was found that removing NSP during pretraining resulted in better performance and so NSP was dropped in RoBERTa. Another adapation that was made was with he batch size. Batch size was increased from 256 to 8k as it was observed that training with large batches improves the perplexity for the MLM objective [13].

RoBERTa tokenizes words with Byte Pair Encoding (BPE) which works fairly similarly to wordpiece. BPE is a character and word level representation that works on subunit words and uses bytes to learn the vocabulary to encode input text. Vocabulary size was additionally increased from 30k to 50k.

All of these changes were combined and resulted in RoBERTa. RoBERTa was also pre-trained over more data — BOOKCORPUS, CC-NEWS, OPENWEBTEXT, STORIES that totals 160 gb versus the 60 gb that was originally used for BERT.

4.3 GPT-2

GPT-2 builds upon GPT-1 and goes on the path of training with unlabeled datasets. It also introduces a way to perform tasks in what is called a zero-shot setting: where down-stream tasks can be done without needing to change the parameters or architecture of the model. In essence, the model can solve an NLP task without needing to be fine-tuned on a labeled dataset. Since BERT and RoBERTa are not intended to be used in a zero-shot setting this is not tested in the project.

There are some modifications made from GPT-1, layer normalization was moved to the input of the sub block and another layer normalization is done after the last self-attention block. Vocabulary was expanded, the context size was doubled, and batch size of 512 was used. However the biggest differences between the two models is that GPT2 uses 48 decoder blocks instead of 12 blocks, and the amount and quality of the dataset used to pre-train the model [2].

GPT-2 has 1.5B parameters versus GPT-1 117M parameters and uses a more dense and diverse corpus. The dataset used to pre-train GPT-2 was created by scraping all the out links (45 million links) from reddit and then extracting the text from the links. This resulted in over 8 million documents with 40GBs of text [2]. In order to not require pre-processing or tokenization at every space, byte pair encoding (BPE) is done to compress text which brings balance between character and word level representations.

4.4 GPT-Neo

In this project I use GPT-Neo from EleutherAI. This implementation is a transformer model which is replicated from the GPT-3 architecture but the number of parameters that it can hold is much lower than GPT-3 [10]. GPT-Neo is a class of models which various parameter sizes can be chosen. I have chosen GPT-Neo 1.3B which is slightly less than the number of parameters than GPT-2 has (1.5B) which allows for a better comparison to GPT-2 and also requires less time that is needed to fine-tune the model.

GPT-Neo was pre-trained on Pile which contains over 825GB data from 22 different sources of academic writing, internet resources, prose, dialogue, and other miscellaneous information. Architecture mimics GPT-3 which has 96 decoder layers [11]. Compared to GPT-2 it has some minor differences with different weight intialization, pre-normailzation but the major difference between these two models is that GPT-Neo alternates between dense and sparse attention patterns. Dense attention is the same as normal attention however sparse attention only uses a subset of tokens in the sequence. More information on the implementation of this can be found in the Sparse Transformer paper [11, 7].

5 Experiment

5.1 Objective

In the previous sections I introduced the models that I will use to fine-tune on various benchmark tasks. The benchmark tasks I have chosen is sentiment analysis (SST dataset), natural language inference (SNLI dataset), and question answering (SQuAD2.0). More details regarding these tasks will be given later.

The purpose of fine tuning BERT, RoBERTa, GPT-2, GPT-Neo is to identify how each of the models perform on the natural language processing tasks and also compare the models training time. I want to identify what models perform the best on each task. One naive hypothesis that could be made is that more parameters in a given model would result in a better performance across all NLP tasks since parameters allows for a more robust model during training [11].

For my models I use the hugging face implementations for BERT Base which contains 110 million(m) parameters, RoBERTa Base which contains 123m parameters, EleutherAI/GPT-Neo-1.3B which contains 1.3billion(b) parameters, and GPT-2 which contains 1.5b parameters. Since GPT-2 has the most parameters I hypothesize that it performs the best across all datasets followed by GPT-Neo, RoBERTa, and finally BERT which has the lowest amount of parameters. I would also assume training time would increase with the number of parameters and so I would also predict that GPT-2 will take the longest time to run then GPT-Neo, RoBERTa, and BERT in that order will take less time to run.

5.2 Datasets

5.2.1 Stanford Sentiment Treebank (SST)

Stanford Sentiment Treebank dataset gathers data from Rotten Tomatoes, a movie review site. The data that it collects consists of movie reviews which are then labeled from zero through four on sentiment which are negative, somewhat negative, neutral, somewhat postive, and positive. This dataset consists of 222k phrases and labels that range from 0-4 corresponding to the sentiment. [8]

5.2.2 Stanford Natural Language Inference (SNLI)

The stanford natural language inference consists of 570k english sentence pairs that are hand-written and labeled entailment, contradiction, and neutral. Each datapoint consists of a premise, hypothesis, and judgement. From the premise a hypothesis is given and then the judgement is given which is labeled contradiction if the hypothesis is not sound, entailment if hypothesis can be made from premise, and neutral if neither no inference can be made. [9]

5.2.3 Stanford Question Answering Dataset (SQuAD2.0)

The Stanford Question Answering Dataset (SQuAD) is based on various wikipedia articles in which the data consists of a question, the context, and the answer to the question with the starting location in the context. SQuAD2.0 is an evolution of SQuad1.1 and combines the 100,000 questions with over 50,000 questions that are unanswerable. [6]

5.3 Preprocessing

For each dataset the method in which they are preprocessed are very different due to the different nature of the tasks they represent. Some require much more preprocessing than others like question answer, and others are much more straightforward to preprocess like sentiment analysis. Due to limited time constraints and processing constraints GPT-2 and GPT-Neo are sometimes preprocessed differently than BERT, and RoBERTa. This is due to GPT having significantly more parameters to fine tune which results in a necessity of more computing power and RAM compared to the other models. More details regarding this will be given below.

5.3.1 Stanford Sentiment Treebank (Sentiment Analysis)

In SST we drop any rows that contain NA values and make sure that there are no phrases that are empty. I make sure to set the max length of all the phrases to 283 characters as when training phrase length must be the same. There is a tradeoff in the max length that is specified to the batch size that we can create due to RAM limitations.

We can then tokenize each of the phrases by the model tokenizer. Each model has a different way in which tokenization or data is encoded into numbers. Tokenization allows for efficient preprocessing of the data and results in a faster training process [12].

Since the max length of text is 283 characters long if the phrase is less than the max length then we put the phrase into a sequence and fill the rest of the space with a special token [PAD] to 283 chars long. When this sequence is tokenized we then gather the input ids, attention mask, and sentiment label. The input ids represent each of the tokens in the sequence, the attention mask represents if the token is in the sequence or is just padding, and the sentiment label is the same from the dataset. This is done for every phrase in the training, validation, and test set.

5.3.2 Stanford Natural Language Inference (Natural Language Inference)

For SNLI the data consists of the premise, hypothesis, and the judgement. The hypothesis and judgement are each cut to 64 chars long, so together 128 chars long. In addition to this we cut any observation that contains a hypothesis and judgement that are longer than 64 characters which allows for training to be done properly which effectively halves the training set. The judgement consists of text which are entailment, neutral, and contradiction which are encoded into numbers. In the data there are some labels which consist of just a dash character, which is an unexpected, so all these observation that contain this label are removed. We also remove all observations that contain NA values. As we have both the premise and hypothesis in our observation we feed these to the tokenizer together and gather the input ids, attention mask, and label (judgement). Due to time constraints and computing power, GPT-2's training set was cut to 300k from originally 550k and for GPT-Neo the training set was cut to 75k.

5.3.3 Stanford Question Answering Dataset 2.0 (Question Answering)

This dataset was the hardest to preprocess as the data that we need to train our task is different than sentiment analysis and natural language inference. SQUAD2.0 consists of question, context, and answer. For each observation question and context are tokenized together with a max length of 384 characters. This task requires a special tokenizer called the fast tokenizer which allows a way to map a word to a token space and also is generally faster in training then normal tokenizers [12]. When the data is tokenized we gather the input ids, attention mask, start point and end point. The start point and end point is found from the starting index and ending index of the answer within the context. Due to running out of compute units, which is explained more in the limitations section, I had to drastically cut GPT-Neo training set to 10k out of 150k observations. GPT-2's training set was not cut.

5.4 Fine-tuning details

Fine tuning was performed similarly across all models. The optimizer used was Adam with a learning rate of 5e-5. Batch sizes vary across models as I tried to use the largest batch size possible that fit with the RAM capacity that I was able to use. I only ran the models with one epoch due to time and computing power constraints.

6 Results

Model	SST	SNLI	SQuAD2.0 (F1)
BERT	0.6077	0.8396	86.34
RoBERTa	0.5826	0.3459	91.82
GPT-2	0.917	0.3449*	17.45
GPT-Neo	0.940	0.335*	4.06*

Table 1: Scores from the transformer models on various datasets. Asterisk exist on scores where models were fine-tuned on a training set of different size

Model	SST	SNLI	SQuAD2.0)
BERT	23m	8m	15m
RoBERTa	26m	24m	15m
GPT-2	34m	22m	28m
GPT-Neo	2h 44m	45 m	31m

Table 2: Time it took to run each model on the respective dataset. m - minutes, h - hours

I fine-tuned BERT, RoBERTa, GPT-2, and GPT-Neo on SST, SNLI, and SQuAD2.0 datasets getting the accuracy for SST, SNLI and the F1 score for SQuAD2.0. I have asterisked some values in the table as some of the models were only able to run a subset of the training data due to time and computing restraints. For GPT-2 in the SNLI dataset it only used utilized 300k / 550k observations from the training set. For GPT-Neo in the SNLI dataset it only utilized 75k / 550k from the training set and from SQuAD2.0 dataset it only used 10k / 150k from the training set.

6.1 Sentiment Analysis

In sentiment analysis I used the Stanford Sentiment Treebank (SST) dataset. This dataset contains 222k training points gathered from movie reviews where each observation consists of a phrase and a sentiment. The objective of sentiment analysis is to predict the sentiment from a given phrase [8].

In this task GPT-Neo actually performed the best having a classification accuracy of 0.94 on the test set followed by GPT-2 with a score of 0.917 followed by BERT with a score of 0.607 and RoBERTa with a score of 0.5826. It seems that the GPT architecture performs better than the BERT architectures on this NLP task. This could possibly be explained by the difference in parameters that the models contain in which GPT-2 and GPT-Neo has 10x more parameters than BERT and RoBERTa. One particular point I find interesting is that GPT-Neo actually outperforms GPT-2. GPT-Neo contains less parameters than GPT-2 but the architecture is slightly different which could account for the difference in performance.

6.2 Natural Language Inference

For natural language inference I used the Stanford Natural Language Inference (SNLI) dataset. The dataset contains of 550k training points from wikipedia articles where each observation consists of a premise, hypothesis, and judgement. The aim of natural language inference is to train a model on premise, and hypothesis to then predict the judgement [9].

I found surprisingly BERT actually performed the best over all other models getting an accuracy of 0.8396 on the test set. There was then a steep drop in performance where RoBERTa had the second best accuracy at 0.3459 followed by GPT-2 and GPT Neo. Since I trained GPT-2 and GPT-Neo on less data the results on these models are not conclusive. If trained on more data I could foresee GPT-2 and GPT-Neo outperforming RoBERTa as the scores are very close with GPT-2 having an accuracy score of 0.3449 and GPT-Neo having an accuracy score of 0.335. However, BERT outperforms all other models by a wide margin so it appears to have a clear advantage in this task.

Since the only major difference between BERT and RoBERTa is BERT having next sentence prediction pre-training objective it is a possibility that BERT's pre-training helps it perform better on natural language inference. GPT-2 and GPT-Neo perform the worst which might be due to their architectural differences from BERT and RoBERTa but the GPT based models actually performs similarly to RoBERTa so it is hard to make any such conclusion on why they perform worse in this task.

6.3 Question Answering

In question answering the dataset that I use to perform this task is the Stanford Question Answering (SQuAD2.0) dataset. SQuAD2.0 consists of over 150k training observations that consist of question, context, and answer. The aim of question answering is given a question procure the answer from the given context [12].

In this task I found that RoBERTa performed the best in this task with a F1 score of 91.82 followed by BERT with a score of 86.34 and then GPT-2 with a score of 17.24 and lastly GPT-Neo with a score of 4.06. BERT architecture seems to have a clear advantage over the GPT architecture in this task however some of the performance difference could be due to huggingface having a specific question answering model for both BERT and RoBERTa that I used to fine-tune on SQuAD2.0. I did base my implementations for GPT-2 and GPT-Neo on the same question answering implementation however since the F1 scores are so low this raises a question if my implementation is lacking in some aspect or if the architecture is not fit for question answering keeping in mind that these models are only trained on one epoch.

6.4 Training time

In regards to training time it took a long time to run all the models on the datasets. I have recorded the time it takes for the models to run on the datasets for one epoch. I have omitted the time it took to run the models on the test set.

I find that GPT-Neo took longer to train than the other models. On SST it ran for 2 hours and 44 minutes which is longer than all the other models combined. I reduced the amount of training data fed to GPT-Neo on SNLI and SQuAD2.0 but even with the smaller training subset training times were still high. GPT-2 took the second longest to train on SST and SQuAD2.0 but RoBERTa took longer to run on the SNLI dataset by two minutes. BERT had the shortest training times only taking 8 minutes on the SNLI dataset which seems to be strange when compared to RoBERTa which took 24 minutes to run on the same dataset.

It is interesting to see different models taking a variable amount of training time. It seems that the number of parameters is not the sole factor in training time since GPT-Neo has less parameters than GPT-2 however it takes a lot longer to train.

7 Conclusion

From my project the results that I gathered was unexpected and certainly did not follow my predictions that I made in my objective statement. From fine-tuning BERT, RoBERTa, GPT-2, and GPT-Neo on different NLP tasks— sentiment analysis (SST), natural language inference (SNLI), and question answering(SQuAD2.0) dataset there was not one model that outperformed the others in all the tasks. In fact it seems that each model had a task in which they performed the best with the exception of GPT-2. BERT did the best on SNLI with an accuracy score of 0.8396 far better than the other models. RoBERTa performed the best on SQuAD2.0 with a F1 score of 91.82 followed closely by BERT. On the SST dataset GPT-Neo outperformed all the other models with an accuracy score of 0.94 followed closely by GPT-2.

From these results it can be said that the BERT based models is more suited towards natural language inference (with the exception of RoBERTa) and question answering. The GPT based architecture seems to have the edge in sentiment analysis. It could be said that my hypothesis of more parameters leading to better performance over all NLP tasks is not entirely accurate in the case of my project and that different architectures seems to specialize in specific NLP tasks which would be interesting to explore in the future.

I also find that training times do not exactly line up with more parameters taking more time to train. I find that GPT-Neo takes significantly longer than all other models to fine-tune. GPT-2 takes the second longest to train followed by RoBERTa, and then BERT. GPT-Neo has slightly less parameters than GPT-2 so it is interesting to see that GPT-Neo takes longer than GPT-2. BERT took the least amount of time to fine-tune which was expected. Thus, it seems that there is some correlation to parameters in training time however the architecture of the models could account for significant differences.

There are some major caveats with these results in my project. I only fine-tuned the models on one epoch so training the models for a much longer period of time could yield different results. Also I cut the training set in GPT-2 and GPT-Neo for some datasets due to limitations I will explain in the next section. This affected GPT-2 and GPT-Neo's performance when scored against the test set.

Overall I find that there is more that can be explored with the project especially with using different architectures and comparing to see how they perform against the models that I use here. It will be interesting to see how models and architectures evolve over time and how performance can be improved in the NLP space.

8 Limitations/Reflection

In this project I had a lot of difficulties in the conceptualization of this project. Initially I had considered using the ELMo architecture along with RoBERTa and GPT-2 however I quickly found that the libraries for ELMo are outdated and implementation was not feasible as some of the code was incompatible with current libraries for deep learning. So I had to scrap the work that was done on ELMo and I pivoted to using BERT, RoBERTa, GPT-2, and GPT-Neo. RoBERTa is based off of BERT and GPT-Neo is based off of GPT so there are parallels there that I could compare and contrast the models with.

This increased my workload as I decided to implement the different models on three benchmark NLP tasks — sentiment analysis, natural language inference, and question answering. As I implemented the models I found myself constantly debugging code which was a result of me using one workflow for one model on a task and then for the other models I found different ways to implement the NLP task. This created more problems for me and after some time I decided to finally base the implementation of one model on a task to the other models which simplified the debugging process.

I also found that implementing question answering was difficult for GPT-2 and GPT-Neo. There are not a lot of resources on how to fine-tune these models with SQuAD2.0 and so I had to look at the github code for how BERT and RoBERTa implemented question answering and use that as a baseline to create a workflow.

When running these models there was a lot of limitations that I encountered. Out of all the limitations easily the biggest issue is with hardware since I do not have a powerful computer. Running deep learning models is not an easy process at all and especially with these models containing a lot of parameters and large datasets that I have to batch and train with I frequently found myself running out of RAM or my computing power was insufficient which resulted in getting a CUDA error or the model needing to take days to run. Thus, I ultimately decided to use one epoch to fine-tune these models on all the datasets.

This did not solve completely solve the issue with hardware. Initially, I used google colab to build and run these models. Using the free version I found that even with one epoch the computing power and RAM space that can be provided was limited and not sufficient to run my models.

So one resource that I used when running these models is the Minnesota Supercomupting Institute (MSI) which provides an interactive Jupyter workspace which I utilized. Unfortunately, I experienced problems with using this resource as I was only able to get access to MSI on December 16th and after December 21st my access was subsequently removed which I surmise to be a result of my access being contingent with one of my classes that ended. So I had to quickly switch to another resource to run my models.

I ended up using google colab pro in which I had to purchase computing units to run my models. Google colab pro provides more computing power and RAM space than the free version however computing units are consumed quickly and once consumed more units need to be purchased. This is a costly process and so I had to make some decisions on how much data to run with especially in regards to my GPT-2 and GPT-Neo models.

Before this class I had never explored the architecture of transformers and there were definitely a lot of questions in my mind regarding the implementation of these transformer based models. After struggling to get my models to work and after finally running them on benchmark datasets I learned a lot through this process and how they work. I also learned a lot about different resources out there like hugging face that provides a way to implement deep learning models. There was a lot that was gained from this project and so even with all of the setbacks that I went through I consider this as a worthwhile experience.

References

[1] Ilya Sutskever Alec Radford, Karthik Narasimhanm Tim Salimans. Improving language understanding by generative pre-training. 2018.

[2] Rewon Child David Luan Dario Amodei Ilya Sutskever Alec Radford, Jeffrey Wu. Language models are unsupervised multitask learners. 2018.

[3] Niki Parmar Jakob Uszkoreit Llion Jones Aidan N. Gomez Lukasz Kaiser Illia Polosukhin Ashish Vaswani, Noam Shazeer. Attention is all you need. 2017.

[4] Kenton Lee Kristina Toutanova Jacob Devlin, Ming-Wei Chang. Bert: Pre-training of deep bidirectional transformers for language understanding. 2019.

[5] Mohit Iyyer Matt Gardner Christopher Clark Kenton Lee Luke Zettlemoyer Matthew E. Peters, Mark Neumann. Deep contextualized word representations. 2018.

[6] Percy Liang Pranav Rajpurkar, Robin Jia. Know what you don't know: Unanswerable questions for squad. 2018.

[7] Alec Radford Ilya Sutskever Rewon Child, Scott Gray. Generating long sequences with sparse transformers. 2019.

[8] Jean Wu Jason Chuang Christopher D. Manning Andrew Ng Christopher Potts Richard Socher, Alex Perelygin. Sentiment analysis on movie reviews. 2014.

[9] Christopher Potts Samuel R. Bowman, Gabor Angeli and Christopher D. Manning. The stanford natural language inference (snli) corpus. 2015.

[10] e.g. Sid Black, Stella Biderman. Gpt-neox-20b: An open-source autoregressive language model. 2022.

[11] Nick Ryder Melanie Subbiah Jared Kaplan e.g. Tom B. Brown, Benjamin Mann. Language models are few-shot learners. 2020.

[12] Yang Song Dave Dopson Denny Zhou Xinying Song, Alex Salcianu. Fast wordpiece tokenization. 2021.

[13] Naman Goyal Jingfei Du Mandar Joshi Danqi Chen Omer Levy Mike Lewis Luke Zettlemoyer Veselin Stoyanov Yinhan Liu, Myle Ott. Roberta: A robustly optimized bert pretraining approach. 2019.