2021 Fall STATSM231A/CSM276A: Pattern Recognition and Machine Learning

Yaxuan Zhu
Department of Statistics
University of California, Los Angeles
9401 Bolter Hall., Los Angeles, California 90095
yaxuanzhu@g.ucla.edu

Abstract

In this course, you will learn multiple algorithms in machine learning. In general, for each homework, you are required to write the corresponding codes. We will provide a structure of code and a detailed tutorial online. Your job is (1) to deploy and understand the code. (2) Run the code and output reasonable results. (3) Make ablation study and parameter comparison. (4) Write the report with algorithms, experiment results and conclusions. For detailed instructions, please check the specification of each homework.

Homework submission forms

You have to submit both code and reports. For the code, zip everything into a single zip file. For the report, use this latex template. When submitting, no need to submit the tex file. Submit one single pdf file only.

1 Coding part: BERT and GPT

1.1 Problems

In the first part of the homework, we are tasked with using BERT and a variation of BERT to predict a masked word within a list of words given as input. As the model performs classification from the words within its vocabulary to guess the masked word, this problem is a classification problem.

In the second part of the homework, we are asked to use the pre-trained GPT-2 model to perform next word prediction for a set amount of times. First, we use the model to continue an input sequence for a maximum of 30 words. Then, we extend the word limit to create paragraphs. We then compare the results with a more specialised model. As the generation of each word is performed using classification, the problem can be thought of as a classification problem.

1.2 Introduction: BERT

BERT which is an acronym for Bidirectional Encoder Representations from Transformers is a pre-trained language model that can be used as a base model to fine-tune for specific natural language processing tasks. It was introduced by Google in the paper BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding in 2018.

BERT uses the transformer architecture to generate embeddings for words given to it as input. Unlike Glove or Word2Vec embeddings, the embeddings generated by BERT have the advantage of being context aware. Furthermore, due to the usage of Transformers, BERT is capable of looking at the entire input as a whole instead of looking at either left or right of the input for contextual information.

BERT was trained on the entire Wikipedia and Book Corpus in a semi-supervised way. During training, a portion of the words input to BERT was masked and the model was asked to determine what these words could be. Moreover, BERT was trained to perform next sentence classification. For this task, BERT was given a sentence and another following sentence. The model was asked to predict if the following sentence was a continuation of the original sentence or just a randomly picked sentence.

The architecture of BERT can be evaluated in three different sections: Embedding, Transformer and Output.

1.2.1 BERT Key Equations and Structure

Embedding

Before entering the model, the input words to BERT must be converted to numerical values so that the model can perform the necessary operations on them. This process is referred to as embedding the input words. There are three separate embedding operations performed for BERT to get the final word embeddings.

1. Token Embedding

The goal of token embedding is to transform each input word to a vector of fixed dimension. In BERT, the dimension of these vectors is 768. To perform the token embedding, the input sentence is first tokenized using a method called WordPiece tokenization. Furthermore, there are special tokens added to this result: [CLS] at the beginning of the list of tokens and [SEP] at the end. The advantage of using WordPiece tokenization is that BERT does not have to learn a large amount of words. After tokenization, each word turns into a 768 length vector.

2. Segment Embedding

As mentioned before, BERT is trained to be able to predict if a sentence given as input is related to the sentence given before. In order to differentiate between the input sentences, segment embedding is utilised.

Implementation of segment embedding is fairly straightforward. Here, all the tokens belonging to the first input sentence is given 0 as the segment embedding and all the tokens of the following sentence is given 1 as the segment embedding.

3. Positional Embedding

Transformers by nature are indifferent to the positions of the vectors given as input. This implies that without an additional embedding, BERT would be unable to use any information regarding the position of a specific word in a sentence. To overcome this problem, positional embeddings are utilised.

The implementation of the positional embedding is also straightforward. In order to give BERT the ability to use positional information, a lookup table that transforms the index of a word to a vector of length 768 is used. As BERT is designed to be able to process input sequences of up to length 512, the size of the lookup table is 512x768. Each row of the lookup table corresponds to the positional embedding of any word located at the index of the row in the given input.

To produce the final word embeddings, all three types of word embeddings are calculated and summed in an element-wise fashion.

The embeddings are implemented in the BertEmbeddings class in the provided source code. They are defined in lines 171-173 and implemented in between the lines 213 and 223.

Transformer

After the embedding process, the vectors are given as inputs to the BERT model. The architecture of BERT is made up of a number of stacked encoders as follows:

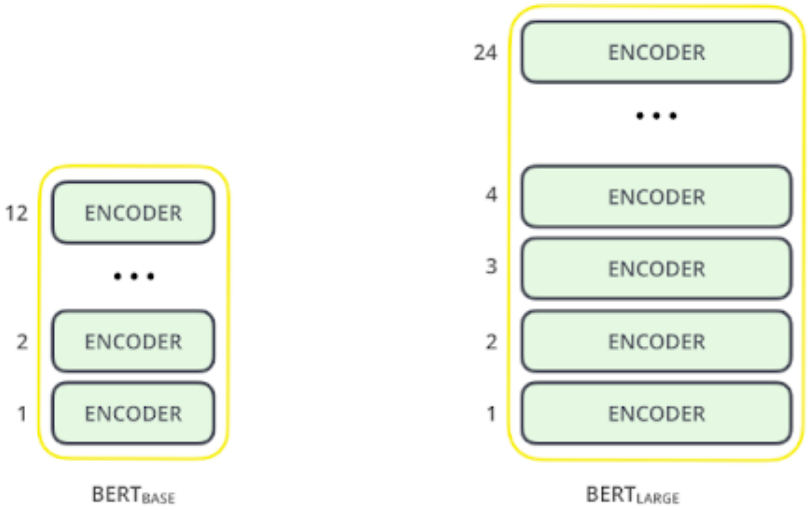

Figure 1: The architecture of different variations of BERT

The base BERT uses 12 successive encoding layers and the large BERT uses 24 such layers. Here, each encoder is composed of the encoder block of a transformer, which uses self attention. Let us now examine the encoding part of the transformer architecture in more detail.

Transformer Encoder

Each transformer encoder block in BERT is made up of the following:

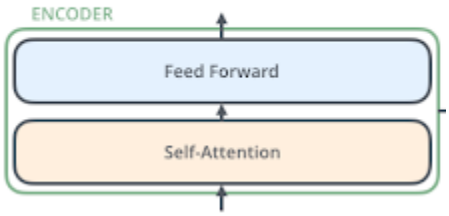

Figure 2: The components within each transformer block

Let us dive deeper to each component.

Self Attention

The concept of self attention in Transformers allows BERT to learn the relationship between a single word in a sentence and all the other words present in the sentence.

To calculate self attention, the first step is to calculate the query, key and value vectors for each input vector to the transformer. These vectors are calculated using three different matrices.The process can be illustrated using the following image:

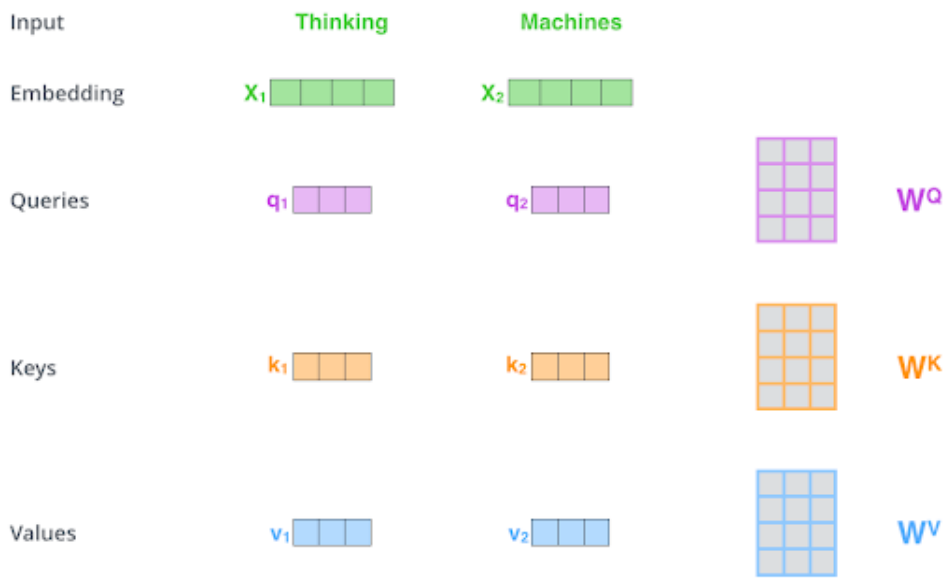

Figure 3: Calculation of query, key and value vectors

Here, the embedding of each word is multiplied with the W_Q matrix to generate the query vectors, W_K matrix to generate the key vectors and W_V to get the value vectors. The values of these matrices are calculated during the training process.

Now, we are ready to calculate the attention weights.

To calculate the weights that the ith word will use to pay attention to the surrounding words, we perform dot product between the ith query vector and all of the key vectors. The results are then divided with the square root of the dimension of the key vectors. This value is 64 for the original paper. We then calculate the softmax values using the obtained results to get the attention weights.In summary, what we do can be expressed as follows:

$$Attention_{ij} = \frac{e^{\frac{\langle q_i, k_j \rangle}{\sqrt{d_k}}}}{\sum_d e^{\frac{\langle q_i, k_d \rangle}{\sqrt{d_k}}}}$$

where $Attention_{ij}$ is the attention weight to calculate the attention the ith word will pay to the jth word, q_i is the query vector of the ith word, k_j is the key vector of the jth word and d_k is the dimension of the key vector. The sum is performed over all the words in the input.

To get the output of the self-attention for the ith word, we multiply the calculated attention weights with the value vectors of each word in the input sequence. In other words, we get:

$$z_i = \sum_j Attention_{ij} * v_j$$

where z_i is the output for the ith word, $Attention_{ij}$ is the attention payed to the jth word by the ith word and v_j is the value vector of the jth word. The sum is performed over all of the words in the input.

This entire process gives the output for the first word. Therefore, the same process is repeated for each word input to the encoder.

The detailed process is only for when there is only 1 attention head. In the base version of BERT, there are actually 12 attention heads for each encoder, which means performing the explained process 12 times for each encoder with different key, query and value matrices for each head. To combine the outputs of each head, we do the following operation:

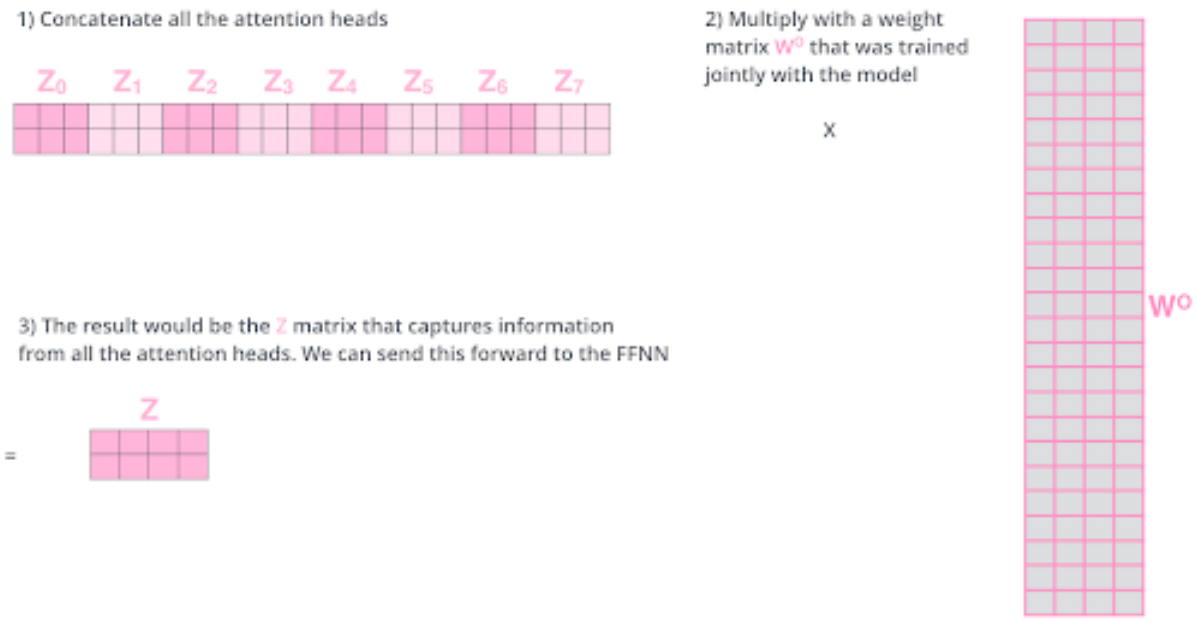

Figure 4: Illustration of combining the attention heads

The Z here is given as input to the feed-forward linear layer, which is used to transform the output of self-attention into a form acceptable by the next encoder.

The self attention is implemented using the class BertSelfAttention between lines 226 and 348 of the provided source code for BERT. The linear part of the transformer is implemented using class BertSelfOutput between lines 351 and 362. Finally, these two components are combined to get the final transformer, which is implemented using the class BertAttention between lines 365 to 411.

Output

After performing the transformer operation 12 times for BertBase and 24 times for BertLarge, we get embeddings for each input word. To use these embeddings for word prediction to determine what a masked word is, we simply use a fully connected layer to transform the embedding at the masked index to a logit vector for all of the words the model is expected to know. Then, we use the softmax operation to get the word with the highest probability.

The output is implemented using the BertForMaskedLM class between lines 1278 and 1379. The calculation of word scores to generate the output is defined with the class BertOnlyMLMHead between lines 677 to 684.

As additional information, if the task is the classification of the entire sequence to determine relevance with a previous sequence, we use the embedding of the start token and add layers to that perform operations on this value.

This concludes the operation of the BERT architecture.

1.3 Introduction: GPT

GPT (Generative Pre-trained Transformer) is a language model developed by OpenAI. The first version was published in 2018 while the second version GPT-2 was published in 2019. As the homework makes use of GPT-2, this will be the architecture we will focus on.

GPT-2 was trained on 40 GB of internet data with the goal of next word prediction. However, experiments show that GPT-2 is capable of doing much more than that. In fact, GPT-2 has been utilised in tasks ranging from paragraph generation to text summarization. Furthermore, it has been

shown that GPT-2 is capable of performing better than task-specific language models on the tasks these models were trained on, without being trained on any task specific data. This implies that the key to having better language models may be to build models with more parameters instead of relying on specialised training.

Just like BERT, GPT-2 uses the transformer architecture. However, while BERT uses the encoder block of transformers, GPT-2 uses the decoder block. Now, let us dive deeper to the components of GPT-2.

1.3.1 GPT Key Equations and Structure

Embedding

As with BERT, GPT-2 needs a form of embedding in order to be able to operate on the words given as inputs. The embedding in GPT-2 is composed of two parts. The first part is token embedding, where each word is converted to a vector with a fixed dimension (768 for small GPT-2). This embedding is performed using byte pair encodings, where the given list of words is compressed into a set vocabulary size by recursively keying the most common word components to unique values.

Simply using the token embeddings is not enough. Much like BERT, GPT needs information regarding the position of each word since the transformer architecture is indifferent to the positions of the input. This is why we also make use of positional embedding. This embedding has the same logic as the positional embedding of BERT, where each index is assigned a specific vector of a specific length.

In the end, the positional and token embeddings are summed element-wise to get the final embeddings.

The token and position embeddings are implemented in the GPT2Model class in the provided source code. On lines 673 and 674, the embeddings are defined and they are summed up on line 831.

Transformer Decoder Block

Each decoder block in GPT-2 has the following architecture:

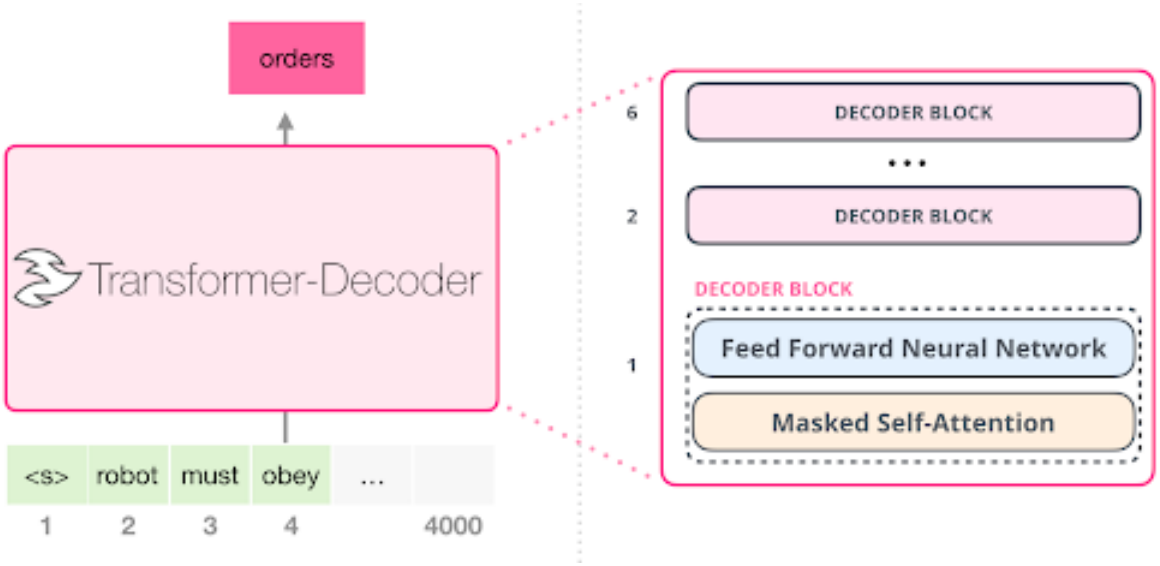

Figure 5: The illustration of each decoder in GPT

As can be seen, the structure of the decoder is quite similar to the structure of the encoder in BERT. The main difference is the fact that the decoder of GPT uses masked self attention. This simply means that when calculating the attention weights using the query for the ith word, we only examine the words located from the beginning to the ith index. We do not look at the words that come after the ith word. Since the rest of the masked self attention calculation is similar to the attention calculation used in the encoder of BERT, the process will not be covered again.

The feed forward neural network in this architecture is composed of two parts. The first part maps the output to a vector that is four times the size of the output (Since the output size of GPT-2 small is 768, the resulting vector would have a dimension of 768*4 = 3072). The four times multiplication is simply borrowed from the original transformer paper. This seems to give transformer models enough representational capacity to handle different tasks.

The second layer projects the result from the first layer back into the output dimension of the embeddings (768 for the small GPT2). The result of this multiplication is the result of the transformer block for this token.

The attention mechanism within the decoder is defined in the GPT2Attention class between lines 135 and 346. The linear fully connected network is defined with the GPT2MLP class between the lines 343 and 363. These mechanisms are combined in the GPT2Block class between the lines 366 and 444. These blocks are then placed one after the other to form the final GPT-2 architecture on line 677 within the GPT2Model class.

Output

Now that we have a better understanding of the model architecture, let us examine how to predict the next word when given a set of words. First, each input word is embedded using token and positional embeddings. These embeddings are summed up to get the word embedding vectors. These vectors are then given to the decoder. Here, for each word, the masked-self attention generates a new vector based on all the words before the said word. Then, the feed forward neural network projects the output of self-attention to a dimension suitable for the next decoder. This process repeats itself until the output of the final decoder. The last element of this output is multiplied with the token embedding matrix. This gives us a score for each word in the vocabulary of the model. To get the actual next word, softmax probabilities for these scores are calculated. Then, a new word is sampled from the vocabulary based on the calculated probabilities. The sample word becomes the predicted next word. The model keeps repeating this process until the token limit of GPT-2 is reached or the end of sequence token is generated.

As can be understood, GPT-2 produces next words on a one-by-one basis, which differs from BERT since BERT is capable of giving out the embeddings for all words at once.

The code for GPT-2 output is located within the GPT2LMHeadModel class between lines 942 and 1099. Here, a linear layer is added to the output of the GPT-2 stacked decoders to turn the decoder outputs to scores for the word the model is responsible for.

This concludes the discussion of the architectures of the different models. Now, let us move on to the different experiments.

1.4 Experiments

1.4.1 BERT: Mask token prediction experiments

How to run To run the code for this section, use the bert.ipynb notebook. Here, the first section loads the transformer library. The second section loads the uncased BERT model and uses the model to generate masked word prediction for the sentence in the tutorial. The sections 3 to 5 are used to test the BERT uncased model on original inputs. The 6th section loads the German BERT.The sections from 7 to 9 are used to perform masked word generation with the German BERT.

For this experiment, we are tasked with using BERT and a variation of BERT to perform masked word prediction. To do so, first we load the pre-trained models. Then, we provide our own inputs to see if the models are capable of accurately guessing the masked words.

In the example, the phrase “I am a [MASKED] model” is given as input and the goal is to guess what the masked word can be. Here, BERT gives the following words for what the masked token can be:

Table 1: The masked word predictions for the input in the tutorial

Input	BERT Predictions for MASK
MASKI am a I am a I am a pt< -I am a pt> del[0] m o	Fashion
	Role
	New
	Super
	Fine

As can be seen, replacing the masked token with the words guessed by BERT allows generating meaningful sentences. Therefore, we can see that BERT is capable of predicting masked words. Now, let us experiment with BERT by providing our own input sentences.

Table 2: The masked word predictions for the first input sentence

Input	BERT Predictions for MASK
MASKI went to the I went to the I went to the pt< -I went to the pt> t see a movie[0] o	theater
	cinema
	bathroom
	movies
	movie

As can be seen, two of the 5 predictions actually make sense. Since BERT uses the transformer architecture, it is capable of paying attention to the last word of the sentence in order to accurately guess what the masked word may be.

Table 3: The masked word predictions for the second input sentence

Input	BERT Predictions for MASK
MASKI drank some I drank some I drank some pt< -I drank some pt> t wake up[0] o	water
	coffee
	wine
	blood
	beer

Table 4: The masked word predictions for the third input sentence

Input	BERT Predictions for MASK
MASKI did not I did not I did not pt< -I did not pt> st night.[0]	sleep
	dream
	remember
	drink
	mean

By examining the results, it can be said that using BERT allows accurate and meaningful predictions for masked words in a sentence.

Now, let us switch things up. Instead of using the base BERT, we are going to load the German BERT and attempt to guess the masked token of a German sentence.

To do so, we first load the German BERT. Then, we give it some input sentences.

Table 5: The German BERT masked word predictions

Input	BERT Predictions for MASK
Ich mag meine [MASK] fahren. (I like to ride my [MASK])	Autos (cars)
	Ski (ski)
	Bahn (train)
	Pferde(horse)
Ich mag [MASK] essen am Frühstück. (I like eat [MASK] at breakfast).	viel (more)
	was (what)
	nicht (not)
	nichts (not)
	nur (only)
Ich gehe gerne ins [MASK]. (I like to go to [MASK])	Kino (cinema)
	Theater (theater)
	Ausland (foreign countries)
	Fernsehen (watch TV)
	Restaurant (restaurant)

As can be seen from the examples above, BERT is not a language model architecture that is specific to English. It can easily be modified to accommodate different languages as well.

1.4.2 GPT: Sentence prediction

How to run To run the code for this section, use the gpt.ipynb notebook. Here, the first section loads the transformer library. The second section loads the GPT-2 model. The sections 3 generates input continuation for the example in the tutorial. Sections 4 to 6 are used to provide our own input to GPT-2. Section 5 attempts to use GPT-2 for paragraph generation. Section 6 loads the specialised story generating GPT-2. Section 7 uses the specialised GPT-2 for paragraph generation.

In this part of the homework, we are asked to generate new words based on an input sequence of words. If we follow the tutorial, we get the following results:

Input: Hello, I'm a language model,

Outputs

Hello, I'm a language model, I'm writing a new language for you. But first, I'd like to tell you about the language itself

Hello, I'm a language model, and I'm trying to be as expressive as possible. In order to be expressive, it is necessary to know

Hello, I'm a language model, so I don't get much of a license anymore, but I'm probably more familiar with other languages on that

Hello, I'm a language model, a functional model... It's not me, it's me! I won't bore you with how

Hello, I'm a language model, not an object model. In a nutshell, I need to give language model a set of properties that

As can be seen, the output generated by GPT is quite impressive. There is some cohesion between the input sentence and the words generated. Furthermore, there also appears to be some logical order. However, there can still be some illogical statements.

Now, let us give some of our own inputs.

Input: The meaning of life is

Outputs

The meaning of life is not simple. Your ability to understand the world, to get from it to its fulfillment, to act on it makes you free

The meaning of life is always the same: the world is always you." He continued, as if trying to escape from it all

The meaning of life is that we will become human. We will die as if we lived, and will live and die. We would have made ourselves

The meaning of life is the ultimate question; our ultimate purpose is a simple one: to love one another. Love is selfless; no one should

The meaning of life is a human right, if it becomes difficult to live independently." – David Lynch Our human right is freedom of speech,

Input: The key to happiness

Outputs

The key to happiness is always staying motivated. In the long term, this goes hand in hand with our ability to be positive.If you

The key to happiness isn't "what your life is like," but the things you are capable of doing and the things that are in your plan.

The key to happiness is hard work, a feeling of success, determination and resilience.For this to work, people want to improve their work

The key to happiness is getting ahead without getting carried away."There are many ways to help." What has your life been like

The key to happiness comes from realizing that something really great is possible, that there are things you can really do to overcome that hurdle and that something truly

Input: I need to work but

Outputs

I need to work but I'm going to spend hours here. I'm like a teenager and I had to learn, and I tried all these things

"I need to work but I can't do this". After he had gone to court, Mr Trump accused Ms Farrow of being

"I need to work but the guys have their jobs. So maybe we'll not get an opportunity like we had last year," she said later.

I need to work but I just can't afford to stay as busy as I do. But if I can move on, I will move on.

'I need to work but the rest of my friends and I have a problem.'So it is not in your interest to leave the town'

As can be seen, GPT-2 is capable of generating new words that have some cohesion with the original input. Although there are some logical discrepancies that may arise, it is still capable of retaining some common themes and making the generated words related to the input.

Paragraph generation with GPT-2

Now, let us allow the GPT-2 model to generate more words to see if it can produce a meaningful paragraph. To do so, the maximum length of the output was set to 300 instead of the original 30. Now, we are ready to generate more sophisticated outputs.

We wanted to have the GPT-2 model write a story for us. Therefore the following was given as input:

Input: The man was sitting alone in a cafe

Let us examine the story GPT-2 generated for us.

Output:

The man was sitting alone in a cafe, making $100 a day at a time. It was a nice break.His wife had come down to visit but there was no way back. He thought he had just broken up with his wife but her response was no help. "Oh, he's fine...""It's just his wife."His wife pulled a bag off the man's back that had fallen off his dresser. This made him dizzy. She held up his laptop and handed it to him. "Hi there," she said. "It's your laptop. It's his. You look terrible." She pointed in his direction. "He's just looking really bad." "Um, what?" "No..." "I have no idea anything is wrong. Let's move on." The man's wife looked over at her, trying to give a smile. She took a little longer than planned before asking him to move on and he walked over to her laptop.He made his way over to what he considered to be a small computer booth. It was his computer. The man didn't seem to make any new purchases but did have a lot of personal items left and he didnt seem to care about the laptop."Hey, there it is," he began. "It's my laptop."The white screen flashed'.

As can be seen, GPT-2 is capable of generating sophisticated paragraphs. However, that does not necessarily mean that there is a consistent flow of events or logic between the sentences. Nevertheless, there is a general theme and certain consistencies such as the man, the computer, the wife, etc. Clearly, the model is far from achieving human level language skills but even the current state of the model is quite impressive.

Paragraph generation with story GPT

Let us see if we can improve the story a little bit. To do so, the gpt2-genre-story-generator model was downloaded from the model zoo of huggingface. This model is a GPT-2 model that is fine tuned to generate genre-based stories.

Let us give the same input: "The man was sitting alone in a cafe" and set the genre to drama to see if the fine tuned model can create a more fleshed out story. The output of this operation is as follows:

Output

The man was sitting alone in a cafe with a cigar. He had an unpleasant dream where he was walking on the balcony of another room. As if he had not seen the whole scene before, he had a vision of himself walking on the balcony of another room. As he was walking on the balcony, he recalled his childhood memories of the life of his father, his younger brother, his mother, his aunt, his father's cousin, and of his mother's death during a struggle, which made his father very angry with his mother. Then he remembered his uncle and told him that once he gets married to the duchess of the country, then he will marry his cousin's brother. His uncle thought that he promised her for that but the daughter refused, while the daughter refused her brother's proposal as he wanted her to be like his other daughter. Then he began to get an invitation from the local girl, and he read her name in the paper and decided to travel to the country of the duchess. He then started to pick flowers from flowers, making a deal with her to buy. When he saw her again, she suggested that he would spend the night in the palace and sleep that night there. Later that evening, he heard the duchess singing and decided to enter the palace. At the rendezvous, the daughter begged him to leave, to which he refused. Then she jumped out of the bed and beat on the ground, which scared.

As can be seen, there is a slightly more coherent flow of events. The result also shares some similarities with the previous story in the sense that the main character (man) and some of the characters remain constant. However, even the fine tuned model still creates some logical issues in the generated story.

Although it is far from a story a human can write, the fact that such a story can even be created by a machine marks a significant progress in the world of Natural Language Processing.

1.5 Conclusion

In conclusion, we learned about various language models that utilise transformers and utilised pre-trained architectures based on BERT and GPT-2 to perform hidden word prediction, next word generation and paragraph generation.

Running of the code was fairly straightforward and there were no major issues. The longest part of the code was installing the transformers library and downloading the various models. However, the actual longest part of the homework was learning more about the components of each model and how the operations were performed.

Reference

"What Is Bert: Bert for Text Classification." Analytics Vidhya, 14 June 2020, https://www.analyticsvidhya.com/blog/2019/09/demystifying-bert-groundbreaking-nlp-framework/.

"Why Bert Has 3 Embedding Layers and Their Implementation Details." Medium, Medium, 3 Jan. 2021, https://medium.com/@init/why-bert-has-3-embedding-layers-and-their-implementation-details-9c261108e28a.

Alammar, Jay. "The Illustrated Transformer." The Illustrated Transformer – Jay Alammar – Visualizing Machine Learning One Concept at a Time., https://jalammar.github.io/illustrated-transformer/.

"Understanding Bert - NLP." GeeksforGeeks, 11 May 2020, https://www.geeksforgeeks.org/understanding-bert-nlp/.

Radford, Alec. "Better Language Models and Their Implications." OpenAI, OpenAI, 21 June 2021, https://openai.com/blog/better-language-models/.

Alammar, Jay. "The Illustrated GPT-2 (Visualizing Transformer Language Models)." The Illustrated GPT-2 (Visualizing Transformer Language Models) – Jay Alammar – Visualizing Machine Learning One Concept at a Time., https://jalammar.github.io/illustrated-gpt2/.

Ankit, Utkarsh. "Transformer Neural Network: Step-by-Step Breakdown of the Beast." Medium, Towards Data Science, 15 Sept. 2021, https://towardsdatascience.com/transformer-neural-network-step-by-step-breakdown-of-the-beast-b3e096dc857f.