STATS M231A Homework 3

Laxman Dahal

Abstract

This homework aims at giving a hands-on experience with state-of-the-art language modeling and prediction models such as BERT and GPT2. Pre-trained models were used by utilizing the huggingface python package. Three different sentences were changed for both the BERT and GPT2 model. The results indicate that both the models give very reasonable answers.

1 Problem 1- BERT

BERT stands for Bidirectional Encoder Representations from Transformers and is the current state-of-the-art model in natural language processing. It is a language representation model that is designed to pre-train bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers.

1.1 Pre-training

The BERT is pre-trained with masked language modeling (MLM) and next sentence prediction (NSP). For every word, we get token embedding from the pre-trained word piece embeddings and add the position and segment embeddings to account for the ordering of the inputs. These are then passed into BERT which under the hood is just a stack of transformer encoders. The BERT outputs a bunch of word vectors for MLM and a binary value for NSP. It is important to note that the word vectors generated by BERT have same size and they are generated simultaneously. We need to take the each word vector, pass it into a fully connected layered output (for eg: softmax layer with 30K neurons- 30k neurons represent the number of tokens in the vocabulary). The softmax essentially gives a probability distribution of the word vector which is then used to train by computing cross entropy loss of only the masked/predicted word. Once the training is complete, BERT is capable of understanding language and context.

1.2 How many parts does embedding contain, what are they?

In BERT, there are three types of embedding: 1) word embedding, 2) position embedding, and 3) segment embedding to account for the ordering of the inputs. Figure 1 give a graphical representation of how three different embeddings are combined to get the input embeddings.

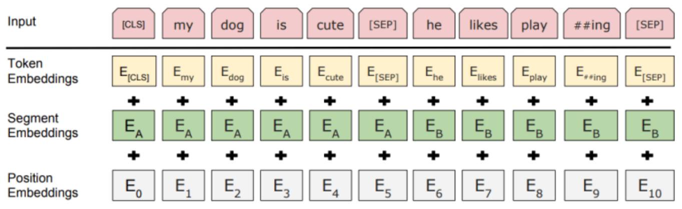

Input
[CLS]
my
dog
is
cute
[SEP]
he
likes
play
##ing
[SEP]
Token Embeddings
$E_{[CLS]}$ E_{my} E_{dog} E_{is} E_{cute} $E_{[SEP]}$ E_{he} E_{likes} E_{play} E_{**ing} $E_{[SEP]}$
Segment Embeddings
E_A E_A E_A E_A E_A E_A E_B E_B E_B E_B E_B
Position Embeddings
E_0 E_1 E_2 E_3 E_4 E_5 E_6 E_7 E_8 E_9 E_{10}

Figure 1: Three types of embedding used in BERT. The input embeddings are the sum of the token, segmentation, and position embeddings

1.3 Transformer: what is self-attention? How is the linear layer like?

Attention mechanism is based on the idea of directing focus and paying greater attention to certain factors when processing the data. Broadly speaking, it answers the question: what part of the input should we focus on? In practice, attention is computed on a set of queries that are represented in a matrix-form, Q. The keys and values are also packed together into matrices K and V respectively. The attention is computed as:

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

Self attention quantifies how relevant the i^{th} word in an English sentence is with respect to the other words in the same English sentence. This is computed in the i^{th} attention vector as show in Figure 2. In other words, the self attention vectors captures contextual relationships between words in the sentence.

The linear layers is a feed-forward layer that is used after the decoder block. It is used to expand dimensions to the number of words in the French language. In other words, in linear layer, number of neurons is equal to the number of words in French. After the linear layer, softmax is used to transform it into a probability distribution. And the final word predicted is the word corresponding to highest probability. Overall the decoder predicts the next sentence and we execute the decoder over multiple time steps until the end of sentence token is generated.

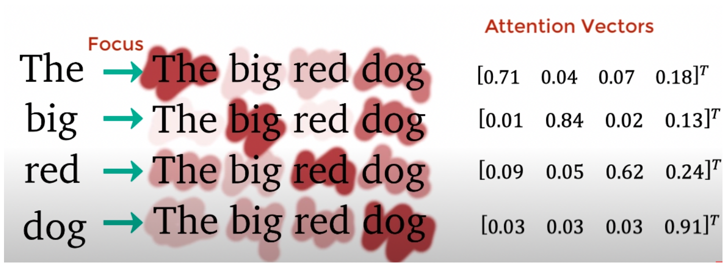


Figure 2: Graphical illustration of attention vector for an example sentence.

1.4 Multihead attention block

It is likely that when the word is first encoded, the attention vector might give much higher attention to itself, which is why a set of eight attention vectors are used. The idea behind using eight attention vectors and computing weighted average is to capture the context and interactions between the words. Since eight attention-vectors are used, it is called multi-head attention block.

1.5 Output: How does the output from transformer become word?

Based on the description so far, we know that there are three different ways transformers use multi-headed attention. Attention in the encoder is a self-attention where it attends to the keys, values, and queries from the same place. The self attention in the decoder is masked because we don't want the model to see the word that it is trying to predict. However, the most important is probably the "encoder-decoder attention" layer where the queries come from the previous decoder layer, and the memory keys and values come from the output of the encoder. This is then passed into feedforward network followed by fully connected linear layer with 30,000 tokens. Following, linear layer, softmax function is used to computed the probability distribution of the predicted word.

1.6 Component in the source code

1. word embedding based on the vocabulary is on line 171 under "BertEmbeddings(nn.Module)" class
2. position embedding is on line 172 under "BertEmbeddings(nn.Module)" class
3. segment embedding is on line 173 under "BertEmbeddings(nn.Module)" class

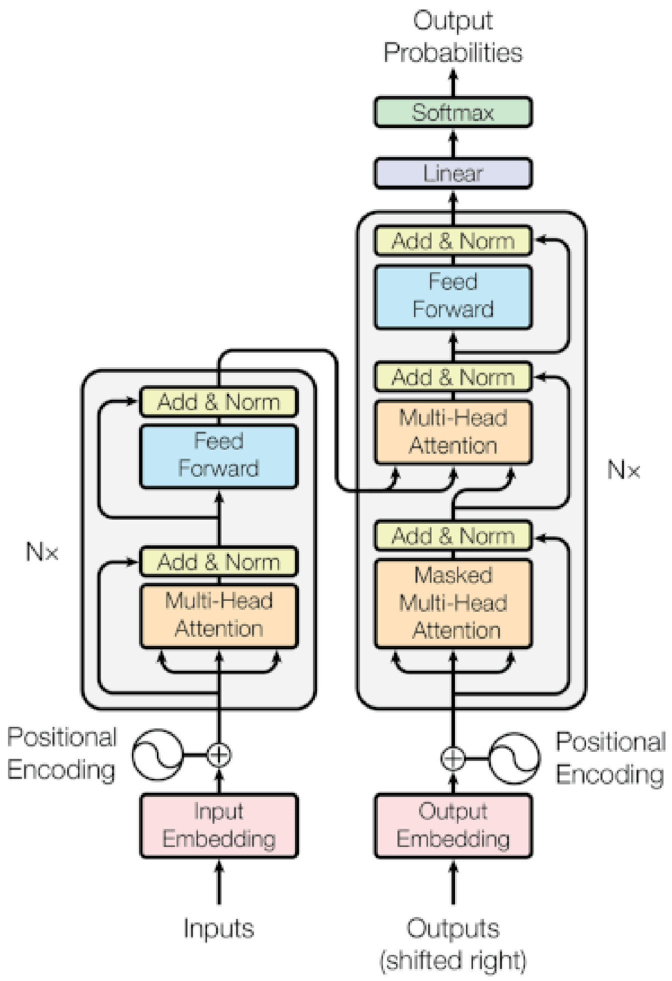


Figure 3: A Transformer's network structure.

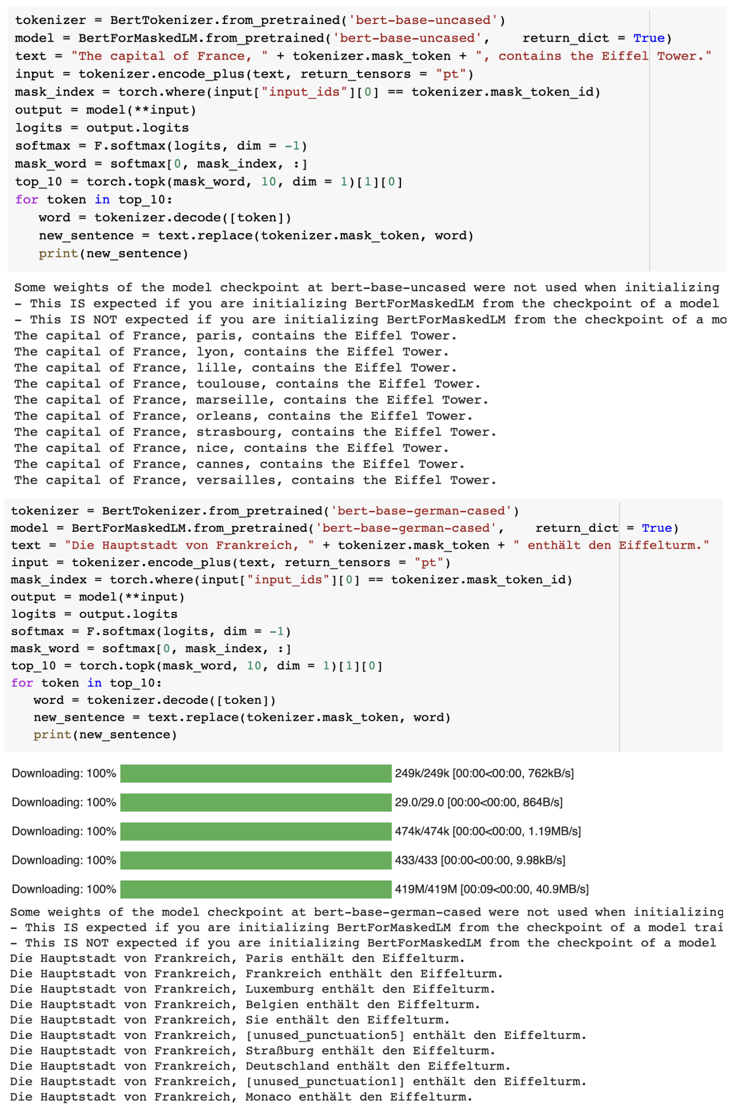
4. "BertSelfAttention(nn.Module)" class on line 226 defines the multi-headed self attention.
   (a) query (Q) is on like 239
   (b) key (K) is on like 240
   (c) value (V) is on line 241

1.7 Experiments

As a part of the experiment, three different sentences were changed using BERT-BASE-UNCASED pre-trained model. Another part was to use a different model, for which BERT-BASE-GERMAN-CASED was used. Figure 4 shows the same sentence unmasked or predicted using the two different models. It can be seen that both the models gave good answers.

2 GPT2

GPT2 stands for Generative Pre-trained Transformers 2. It is a general-purpose learner that was not specifically trained to do any specific tasks but it has been used to translate text, answer questions, summarize passages, and generate texts outputs. Figure 5 shows the difference between GPT2 and BERT. In BERT, encoders are stacked, but in GPT decoders are stacked. The attention mechanism or the idea of transformers used in both the methods are similar. One key difference between the two is that GPT2, like traditional language models, outputs one token at a time.



```
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertForMaskedLM.from_pretrained('bert-base-uncased',    return_dict = True)
text = "The capital of France, " + tokenizer.mask_token + ", contains the Eiffel Tower."
input = tokenizer.encode_plus(text, return_tensors = "pt")
mask_index = torch.where(input["input_ids"][0] == tokenizer.mask_token_id)
output = model(**input)
logits = output.logits
softmax = F.softmax(logits, dim = -1)
mask_word = softmax[0, mask_index, :]
top_10 = torch.topk(mask_word, 10, dim = 1)[1][0]
for token in top_10:
   word = tokenizer.decode([token])
   new_sentence = text.replace(tokenizer.mask_token, word)
   print(new_sentence)
```

```
Some weights of the model checkpoint at bert-base-uncased were not used when initializing
- This IS expected if you are initializing BertForMaskedLM from the checkpoint of a model
- This IS NOT expected if you are initializing BertForMaskedLM from the checkpoint of a mo
The capital of France, paris, contains the Eiffel Tower.
The capital of France, lyon, contains the Eiffel Tower.
The capital of France, lille, contains the Eiffel Tower.
The capital of France, toulouse, contains the Eiffel Tower.
The capital of France, marseille, contains the Eiffel Tower.
The capital of France, orleans, contains the Eiffel Tower.
The capital of France, strasbourg, contains the Eiffel Tower.
The capital of France, nice, contains the Eiffel Tower.
The capital of France, cannes, contains the Eiffel Tower.
The capital of France, versailles, contains the Eiffel Tower.
```

```
tokenizer = BertTokenizer.from_pretrained('bert-base-german-cased')
model = BertForMaskedLM.from_pretrained('bert-base-german-cased',    return_dict = True)
text = "Die Hauptstadt von Frankreich, " + tokenizer.mask_token + " enthält den Eiffelturm."
input = tokenizer.encode_plus(text, return_tensors = "pt")
mask_index = torch.where(input["input_ids"][0] == tokenizer.mask_token_id)
output = model(**input)
logits = output.logits
softmax = F.softmax(logits, dim = -1)
mask_word = softmax[0, mask_index, :]
top_10 = torch.topk(mask_word, 10, dim = 1)[1][0]
for token in top_10:
   word = tokenizer.decode([token])
   new_sentence = text.replace(tokenizer.mask_token, word)
   print(new_sentence)
```

```
Downloading: 100%  249k/249k [00:00<00:00, 762kB/s]
Downloading: 100%  29.0/29.0 [00:00<00:00, 864B/s]
Downloading: 100%  474k/474k [00:00<00:00, 1.19MB/s]
Downloading: 100%  433/433 [00:00<00:00, 9.98kB/s]
Downloading: 100%  419M/419M [00:09<00:00, 40.9MB/s]
Some weights of the model checkpoint at bert-base-german-cased were not used when initializing
- This IS expected if you are initializing BertForMaskedLM from the checkpoint of a model trai
- This IS NOT expected if you are initializing BertForMaskedLM from the checkpoint of a model
Die Hauptstadt von Frankreich, Paris enthält den Eiffelturm.
Die Hauptstadt von Frankreich, Frankreich enthält den Eiffelturm.
Die Hauptstadt von Frankreich, Luxemburg enthält den Eiffelturm.
Die Hauptstadt von Frankreich, Belgien enthält den Eiffelturm.
Die Hauptstadt von Frankreich, Sie enthält den Eiffelturm.
Die Hauptstadt von Frankreich, [unused_punctuation5] enthält den Eiffelturm.
Die Hauptstadt von Frankreich, Straßburg enthält den Eiffelturm.
Die Hauptstadt von Frankreich, Deutschland enthält den Eiffelturm.
Die Hauptstadt von Frankreich, [unused_punctuation1] enthält den Eiffelturm.
Die Hauptstadt von Frankreich, Monaco enthält den Eiffelturm.
```

Figure 4: An example implementation of two different BERT models

2.1 The decoder block

Figure 8 shows a typical decoder block where a layer is allowed to attend to specific segments from the encoder. One key difference in the self-attention layer in the decoder block is that it masks future tokens - not my changing the work to [mask] like BERT, but by interfering in the self-attention calculation blocking information from tokens that are to the right of the position being calculated.

2.2 How many parts does embedding contain, what are they?

GPT2 contains one token embedding matrix and one positional embedding matrix.

2.3 Fully connected layers: How word is predicted

GPT-2 consists of two fully connected neural network where the block processes its input token after self attention has included the appropriate context in its representation. Figure 7 shows that the first fully connected layer in GPT small is four times its model size (768). This ensures that transformer models have enough representational capacity to be able to perform sentence prediction.

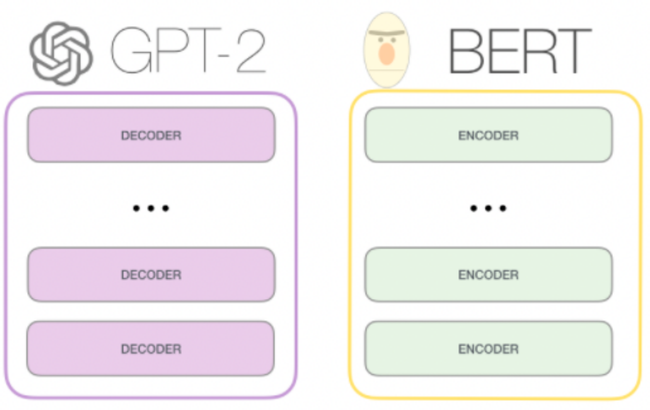


Figure 5: Comparison between GPT2 and BERT.

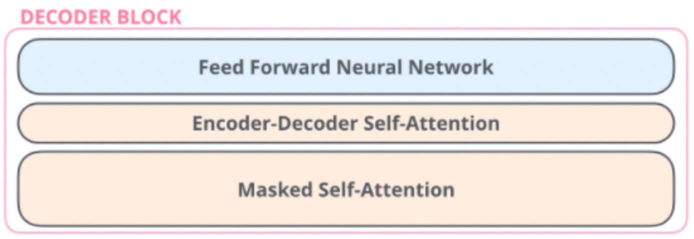


Figure 6: Illustration of a decoder block.

2.4 Component in the source code

1. word embedding based on the vocabulary is on line 673 under "GPT2Model()" class
2. position embedding is on line 374 under "GPT2Model()" class
3. "GPT2Attention(nn.Module)" class is defined on lines 135-346 but it is first initialized on line 373 inside the "GPT2Block(nn.Module)" class

2.5 Experiments

As a part of the experiment, three different sentences were changed using "TFGPT2LMHeadModel" Model and "GPT2Tokenizer" was used where gpt2-large was selected as an option. Please refer to the attached notebook for the word predicted and sentences generated. **??** shows one of the examples implemented and the predicted words. The input was *I think it will rain tomorrow. Maybe I should.*

How to run There are two jupyter notebook attached with this written report. The first notebook titled "hw3_BERT.ipynb" is the notebook for BERT which also include the example sentences. The second notebook titled "hw3_GPT2.ipynb" is used for the GPT2 model and it also contains the example. Both the notebooks were executed in the Google Colab Pro using their GPU capability.

2.6 Conclusion

In this homework state-of-the-art language modeling and prediction models were used to complete the sentences. It was observed that BERT is really good at masked language modeling and next sentence prediction. On the other hand, GPT2 is intended to be and best at text generation.

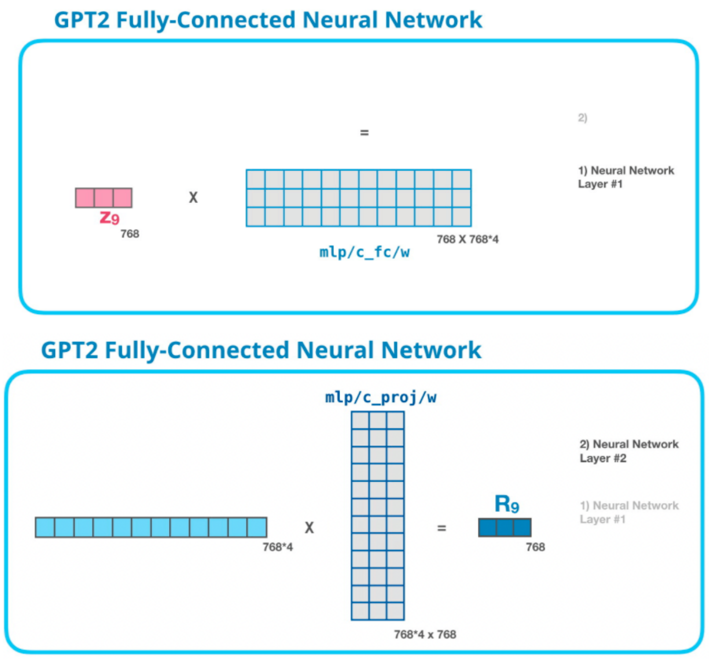
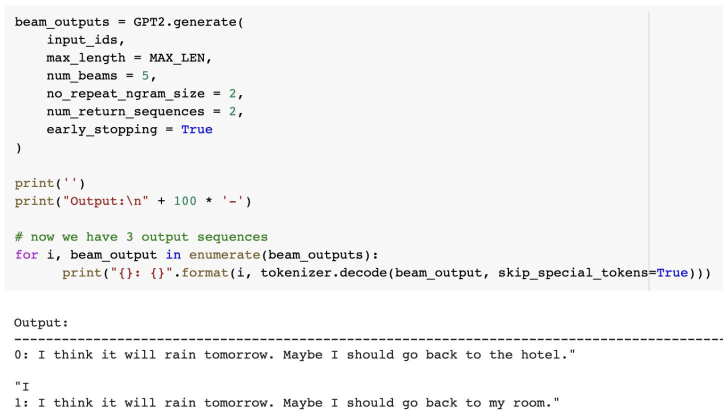

GPT2 Fully-Connected Neural Network
=
2)
1) Neural Network Layer #1
z9
768
X
768 X 768*4
mlp/c_fc/w
GPT2 Fully-Connected Neural Network
mlp/c_proj/w
2) Neural Network Layer #2
R9
1) Neural Network Layer #1
768*4
X
=
768
768*4 x 768

Figure 7: Two fully connected layers of GPT2 layers



```
beam_outputs = GPT2.generate(
    input_ids,
    max_length = MAX_LEN,
    num_beams = 5,
    no_repeat_ngram_size = 2,
    num_return_sequences = 2,
    early_stopping = True
)

print('')
print("Output:\n" + 100 * '-')

# now we have 3 output sequences
for i, beam_output in enumerate(beam_outputs):
      print("{}: {}".format(i, tokenizer.decode(beam_output, skip_special_tokens=True)))
```

```
Output:
----------------------------------------------------------------------------------------------------
0: I think it will rain tomorrow. Maybe I should go back to the hotel."

"I
1: I think it will rain tomorrow. Maybe I should go back to my room."
```

Figure 8: An example implementation of the GPT2. The input sequence was: *I think it will rain tomorrow. Maybe I should*