<u>CS 613: NLP</u>

Solutions to Assignment 3: Pretraining and fine-tuning an LLM

Group : Vec2R

GitHub repository: https://github.com/PechimuthuMithil/CS613_Assignment3

---

2) Calculating the number of parameters.

We can use the summary function from the torchinfo library. This function gives us a total summary of the layers, their shapes along with the trainable and non-trainable parameters related to each of the layers. This function was to get the following summary (code is present in the repository).

[redacted]ncoder

```
================================================================
Layer (type:depth-idx)                        Param #
================================================================
BertLayer                                     --
├─BertAttention: 1-1                          --
│    └─BertSelfAttention: 2-1                 --
│    │    └─Linear: 3-1                       590,592 = W^Q
│    │    └─Linear: 3-2                       590,592 = W^K
│    │    └─Linear: 3-3                       590,592 = W^V
│    │    └─Dropout: 3-4                      --
│    └─BertSelfOutput: 2-2                    --
│    │    └─Linear: 3-5                       590,592 = W^O = h x d_q x d_model + d_model = d_model^2 + d_model
│    │    └─LayerNorm: 3-6                    1,536 = 2 x d_model
│    │    └─Dropout: 3-7                      --
├─BertIntermediate: 1-2                       --
│    └─Linear: 2-3                            2,362,368 = W_1 + b_1 = d_model x d_ff + d_ff
│    └─GELUActivation: 2-4                    --
├─BertOutput: 1-3                             --
│    └─Linear: 2-5                            2,360,064 = W_2 + b_2 = d_ff x d_model + d_model
│    └─LayerNorm: 2-6                         1,536
│    └─Dropout: 2-7                           --
================================================================
Total params: 7,087,872
Trainable params: 7,087,872
Non-trainable params: 0
================================================================
```

Handwritten notes beside the BertLayer summary:

- W^Q, W^K, W^V: each is $h \times d_{model} \times d_q + d_{model}$; also, $h \times d_q = d_{model}$ $\Rightarrow$ total $= 3 \times d_{model}^2 + 3 \times d_{model}$
- part of SA layers
- part of FF layers
- LN layers

```
========================================================================
Layer (type:depth-idx)                               Param #
========================================================================
BertModel                                            --
├─BertEmbeddings: 1-1                                --
│    └─Embedding: 2-1                                23,440,896 = d_vocab x d_model = word embedding matrix
│    └─Embedding: 2-2                                393,216 = d_max_positional_embeddings x d_model = positional embeddings
│    └─Embedding: 2-3                                1,536
│    └─LayerNorm: 2-4                                1,536
│    └─Dropout: 2-5                                  --
├─BertEncoder: 1-2                                   --
│    └─ModuleList: 2-6                               --
│    │    └─BertLayer: 3-1                           7,087,872
│    │    └─BertLayer: 3-2                           7,087,872
│    │    └─BertLayer: 3-3                           7,087,872
│    │    └─BertLayer: 3-4                           7,087,872
│    │    └─BertLayer: 3-5                           7,087,872
│    │    └─BertLayer: 3-6                           7,087,872
│    │    └─BertLayer: 3-7                           7,087,872
│    │    └─BertLayer: 3-8                           7,087,872
│    │    └─BertLayer: 3-9                           7,087,872
│    │    └─BertLayer: 3-10                          7,087,872
│    │    └─BertLayer: 3-11                          7,087,872
│    │    └─BertLayer: 3-12                          7,087,872
├─BertPooler: 1-3                                    --
│    └─Linear: 2-7                                   590,592
│    └─Tanh: 2-8                                     --
========================================================================
Total params: 109,482,240
Trainable params: 109,482,240
Non-trainable params: 0
========================================================================
```

Handwritten note beside the BertLayer 3-1 to 3-12 rows: 12 - Encoders

Figure 1: Summary of all the layer shapes in bert-base. Note how Each of the query, key and values parameters also include an additional bias vector of size d_model (which was mentioned during the class but not considered).

The notations used below and in Figure 1 are the same as used in class here.

The paper states that the model has the following parameters:
h = 12
d_model = 768
d_ff = 3072
L = 12
d_vocab = 30000 (but the 🤗model has a vocabulary size of 30522)
d_max_positional_encodings = 512

We primarily report results on two model sizes: **BERT**$_{\mathbf{BASE}}$ (L=12, H=768, A=12, Total Parameters=110M) and **BERT**$_{\mathbf{LARGE}}$ (L=24, H=1024, A=16, Total Parameters=340M).

Figure 2: Snip from paper [1] that states the total number of parameters in $\text{BERT}_{\text{BASE}}$.

Moreover other than the word embeddings and positional embeddings, BERT also has Token embeddings of shape [2, 768] = 1536 total trainable parameters. There is also a Layer normalization after all the embedding is done. This adds another 1536 total trainable parameters.

The paper states that bert-base has around 110 million parameters. From Figure 1 we can conclude that the model has a total of 109,482,240 trainable parameters. The values does not match as the total parameters that are stated in the paper is an approximation and also the vocabulary size used by 🤗model is different from the vocabulary size used by the paper.

---

3) Time taken to run the 5 epochs of pre training: 1 hour, 46 minutes and 38 seconds
Pre Training is done through the program here.

---

4)

Epoch Number	1	2	3	4	5
Log (base 10) Perplexity score	124	116	111	105	99

Table 1: Variation of perplexity with epochs

Explanation of the trend seen in Table 1: As can be seen from Table 1 that the perplexity scores start to fall drastically in the first few epochs. This trend can be attributed to the ability of the Encoder architecture to capture the patterns in the dataset. A steeper fall would have been seen had we pre-trained it over a larger dataset. This trend will be sharper (a steeper fall in perplexity scores) during fine tuning, this because the model can exploit its already pre-trained weights, to reach low perplexity quickly [8].

NOTE: The pretrained model was first uploaded to 🤗 for fine tuning task. It was then tested for perplexity all again and the curated data is what will be seen above. We have performed a total of 5 epochs of training. Please find the google collab link here.

---

5) The pre trained model was pushed to 🤗 here.

---

6 & 7) We loaded the dataset using the Hugging Face Datasets library with load_dataset('glue', 'sst2'). We used it as it had the train validation and test set automatically defined.

a) Classification (Code + eval)
   Accuracy: **0.5092**
   Precision: **0.5092**
   Recall: (Measures the ratio of correctly predicted positive observations to all actual positives. It's about the coverage of actual positives.) **1.0**
   F1: (The harmonic-mean of precision and recall. It gives a balanced measure between precision and recall). **0.6748**

   The Accuracy and Precision scores are the same as the model is making few false positives and errors.

b) Question-Answering (Code)(eval) For the training, we let the model run for just one epoch.
   F1 score: **0.001326**
   Average Meteor Score: **0.05448**
   Average BLEU Score: **6.803596311961173e-233**
   Rouge Score: **754.0**
   Exact Match Score: **0.13263%**

   The scores are not very promising and reflect the poor performance of the model.

---

8) Total number of parameters of fine tuned parameters:

```
<<<<<<<<<<<<<<<<<<<Printing Layer Shapes>>>>>>>>>>>>>>>>>>>
<<<<<<<<<<<<<<<<<<<Bert-base Shapes>>>>>>>>>>>>>>>>>>>
================================================================================
Layer (type:depth-idx)                                          Param #
================================================================================
BertForSequenceClassification                                   --
├─BertModel: 1-1                                                --
│    └─BertEmbeddings: 2-1                                      --
│    │    └─Embedding: 3-1                                      23,440,896
│    │    └─Embedding: 3-2                                      393,216
│    │    └─Embedding: 3-3                                      1,536
│    │    └─LayerNorm: 3-4                                      1,536
│    │    └─Dropout: 3-5                                        --
│    └─BertEncoder: 2-2                                         --
│    │    └─ModuleList: 3-6                                     85,054,464
│    └─BertPooler: 2-3                                          --
│    │    └─Linear: 3-7                                         590,592
│    │    └─Tanh: 3-8                                           --
├─Dropout: 1-2                                                  --
├─Linear: 1-3                                                   1,538
================================================================================
Total params: 109,483,778
Trainable params: 109,483,778
Non-trainable params: 0
================================================================================
```

Figure 3: Summary of the bert-base-uncased model, fine tuned for classification.

Figure 3 shows that to fine tune the model for classification we had to add a linear layer. This linear layer added an additional 1,538 trainable parameters. So the number of parameters doesn't remain the same as for the pretrained model, as we have to add an extra later to analyze the [CLS] token in a way that is specific for that particular task.

Similar values were observed for the fine tuned model for question answering. It also had a total of 109,483,778 parameters that are 1,538 more than the pre-trained model. The reasoning for this is very similar to that for the fine tuned model for classification.

---

9) Pushed the fine-tuned model to 🤗(Link)

---

10)

a) Performance : The observed performance is very low. Table 2 shows the results on fine tuning during classification. Also the answers to question 7 contains more performance scores of the fine tuned model.

Epoch	Training Loss	Validation Loss	Accuracy

1	0.695900	0.694548	0.490826
2	0.694500	0.721894	0.509174
3	0.696300	0.702839	0.509174
4	0.699000	0.693834	0.509174
5	0.693800	0.704753	0.50917

Table 2: Training results during fine-tuning (classification) for 5 epochs

This poor performance is majorly attributed to the low number of epochs during pre training and fine tuning, which is supplemented by the small dataset on which the model was pre trained. Even the performance for the QnA is also poor.

b) Rationale behind the increased number of parameters in the fine tuned model: To understand the increase in number of parameters in the fine tuned model, we should know why language models are pretrained. The major objective of pretraining is to learn relationships between sentences and predict if one sentence follows another. Once this is done, the pre-trained model is ready to be used in a specific use case after it is fine tuned for that task. This is beneficial, as the time taken, the number of epochs required to reach low perplexity scores are much lower during finetuning, since we need to tweak a lower number of parameters that were added for fine tuning. Had this not been the case, then we would have to train the full model (that is the total number of parameters equals the parameters in the fine tuned model) then training the model would require extensive resources and time. We are saving time and compute by pre-training a model apriori.

Contribution

Name	Roll Number	Contribution (Questions)
Mithil Pechimuthu	21110129	Q2) Q8) Q10) + Final Documentation
Kaushal Kothiya	21110107	Q3) Q4) Q5)
Dhruv Gupta	21110070	Q3) Q6) Q7) Q8)
Rachit Verma	21110171	Q6) Q7) Q8)
Sachin Jalan	21110183	Q6) Q7) Q8)
Anish Karnik	21110098	Q3) Q4) Q5)
Ayush Modi	21110039	Q3) Q4) Q5)
Sahil Das	21110184	Q6) Q7) Q8)

More Rutwik	21110133	Q3) Q6) + Final Documentation

References

1. [torchinfo] Obtaining the summary of a model using summary function of *torchinfo* library
2. [Calculating total parameters] Sir's handout for calculating total number of parameters for BERT
3. [HF's Notebooks] Huggingface's notebooks
4. [Stackoverflow] Number of parameters
5. [perplexity on SO] Perplexity
6. [transformers-util] If low on resources.
7. [SO] Adding layers on top of bert.
8. [Medium article] Understanding the process of fine tuning the BERT-model
9. [Hugging Face perplexity] For calculating perplexity