

Indian Institute of Technology Gandhinagar



NLP Assignment-3 Team - Cloud 8

CS613 Assignment-3 Tasks and Results
18th November 2023

Authors

Aditi Agarwal 20110006
Harshvardhan Vala 20110075
Inderjeet Singh 20110080
Joy Makwana 20110086
Kalash Kankaria 20110088
Kanishk Singhal 20110091
Medhansh Singh 20110111
Nokzendi Aier 20110173

Under the guidance of
Professor. Mayank Singh

Tasks

Git Repo : [CS613-NLP/assignment-3](https://github.com/CS613-NLP/assignment-3)

1-3. Pre-Train model

Dataset Description:

- [wikitext-2-raw-v1](#) - train split
- 36.7k instances (rows)
- Each row is a string

text string · lengths

= Valkyria Chronicles III =
Senjō no Valkyria 3 : Unrecorded Chronicles (Japanese : 戦場のヴァルキュリア3 , lit . Valkyria of the Battlefield 3) , commonly referred to as Valkyria Chronicles III outside Japan , is a tactical role @-@ playing video game developed by Sega and Media.Vision for the PlayStation Portable . Released in January 2011 in Japan , it is the..
The game began development in 2010 , carrying over a large portion of the work done on Valkyria Chronicles II . While it retained the standard features of the series , it also underwent multiple adjustments , such as making the game more forgiving for series newcomers . Character designer Raita Honjou and composer Hitoshi Sakimoto both..
It met with positive sales in Japan , and was praised by both Japanese and western critics . After release , it received downloadable content , along with an expanded edition in November of that year . It was also adapted into manga and an original video animation series . Due to low sales of Valkyria Chronicles II , Valkyria..
= = Gameplay = =
As with previous Valkyria Chronicles games , Valkyria Chronicles III is a tactical role @-@ playing game where players take control of a military unit and take part in missions against enemy forces . Stories are told through comic book @-@ like panels with animated character portraits , with characters speaking partially through voiced..
The game 's battle system , the Blitz system , is carried over directly from Valkyria Chronicles . During missions , players select each unit using a top @-@ down perspective of the battlefield map : once a character is selected , the player moves the character around the battlefield in third @-@ person . A character can only act..
Troops are divided into five classes : Scouts , Shocktroopers , Engineers , Lancers and Armored Soldier . Troopers can switch classes by changing their assigned weapon . Changing class does not greatly affect the stats gained while in a previous class . With victory in battle , experience points are awarded to the squad , which are..

Example rows

Data Preprocessing:

- All strings converted to lowercase
- Removed empty rows within the dataset

Pre-Training:

- Model used: bert-base-uncased
- Hyperparameters:
 - Tokenizing block size: 128
 - Mlm_probability (k): 0.15

- No. of Epochs: 5 (as required)
- Batch size: 32

Calculating Model Parameters:

```
from transformers import BertForPreTraining

pre_trained_model = BertForPreTraining.from_pretrained("Skratch99/bert-pretrained")
params = sum(p.numel() for p in pre_trained_model.parameters())

print("No. of parameters: ", params)
```

➡ No. of parameters: 110106428

The [paper](#) reports the following number of parameters:

BERT_{BASE} (L=12, H=768, A=12, Total Parameters=110M) and **BERT_{LARGE}** (L=24, H=1024,

Therefore the number of parameters in our model and that reported in the paper are approximately the same, that is 110 million params.

4. Perplexity Score

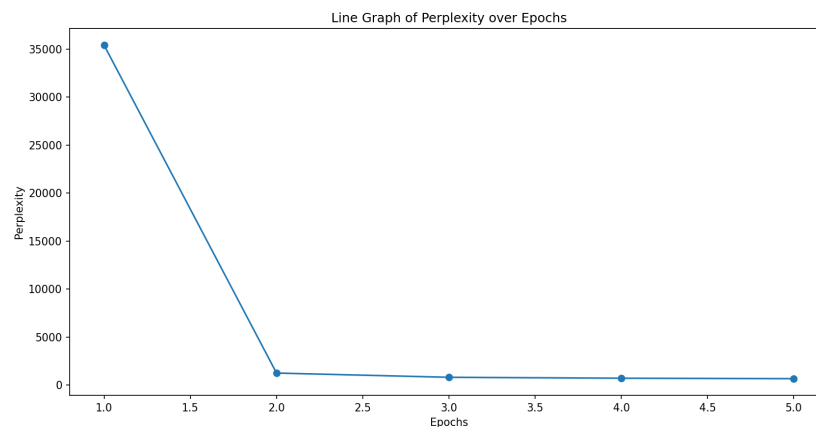


Fig. Perplexity over Epoch

From the above plots, it is evident that the perplexity scores on the test split of ‘wikitext-2-raw-v1’ consistently decrease after each epoch. The decreasing perplexity scores indicate that the model is improving in predicting the correct output in the sequence. Lower perplexity scores generally correspond to better language model performance. This improvement can be attributed to the optimization process during training, where the model adjusts its parameters to better capture the patterns and dependencies in the training data.

Epochs	Perplexity
Epoch: 1	35415.32800860076
Epoch: 2	1263.3219515
Epoch: 3	822.706899047
Epoch: 4	726.690016938
Epoch: 5	679.951110888

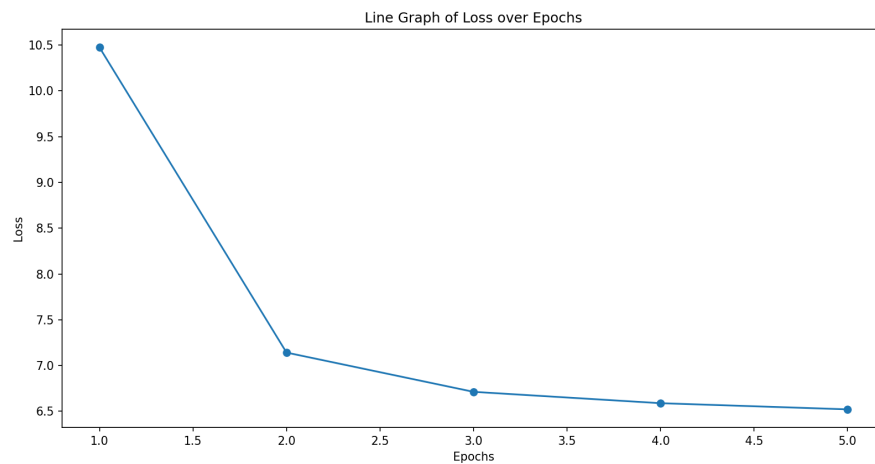


Fig. Log of Perplexity over Epoch

It can be observed that the loss which is basically the log of perplexity also decreases after each epoch as the model is becoming better by getting trained on more data. After the pre-trained model is loaded again and evaluated, the **reported perplexity is around 680**.

5. Pushing the Pre-Trained Model

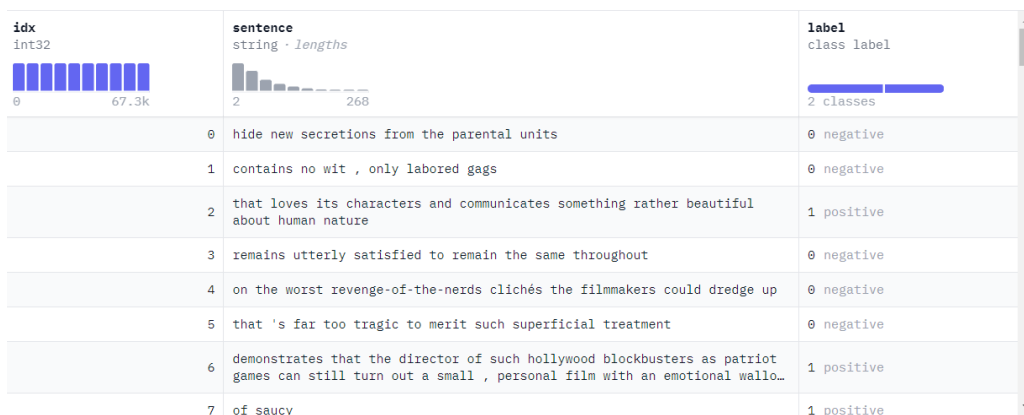
The pretrained model was then pushed to <https://huggingface.co/> 🤖.

Link to the Model: <https://huggingface.co/Skratch99/bert-pretrained>

6. Fine-Tuning model

a) Classification

The pre-trained model was pretrained on the “SST-2” Dataset. A preview of the dataset is shown below.



idx int32	sentence string · lengths	label class label
0	hide new secretions from the parental units	0 negative
1	contains no wit , only labored gags	0 negative
2	that loves its characters and communicates something rather beautiful about human nature	1 positive
3	remains utterly satisfied to remain the same throughout	0 negative
4	on the worst revenge-of-the-nerds clichés the filmmakers could dredge up	0 negative
5	that 's far too tragic to merit such superficial treatment	0 negative
6	demonstrates that the director of such hollywood blockbusters as patriot games can still turn out a small , personal film with an emotional wallo...	1 positive
7	of saucy	1 positive

We load the dataset ‘sst2’ and split it into an 80:20 split of train-test using stratify as mentioned in the assignment. After splitting the dataset, we did the following:

- Get the labels from the dataset
- Padding and truncating the input sentences. This is done to make the length of the input sentences consistent and make the calculations easier. Truncating the sentences makes it easier to evaluate and it computationally less intensive in memory and time.
- Set the format for PyTorch




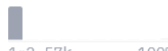
We load the model from ‘BertForSequenceClassification’ and the tokenizer from ‘AutoTokenizer’ from [Skratch99/bert-pretrained · Hugging Face](https://huggingface.co/Skratch99/bert-pretrained).

The model was trained on 5 epochs on the training dataset. [Here](#) is the link of the pushed model on hugging face.

b) Question-Answering

We were asked to fine-tune the pre-trained model on the “squad_v2” dataset.

This is what the dataset looks like:

id string · lengths	title string · lengths	context string · lengths	question string · lengths	answers sequence
 24 100%	 3-9 30.8%	 507-863 59.6%	 1-2.57k 100%	
56be85543aeaaa14008c9065	Beyoncé	Beyoncé Giselle Knowles-Carter (/bi:ˈjɒnsə/ bee-YON-say) (born September 4, 1981) is an American singer, songwriter, record producer and actress. Born and raised in Houston, Texas, she performed in various singing and dancing competitions as a child, and rose to fame in the late 1990s as lead singer	What areas did Beyonce compete in when she was growing up?	{ "text": ["singing and dancing"], "answer_start": [207] }

What we want importantly from this dataset is- “questions” , “answers” and “context” columns.

We load the dataset from “squad_v2” and then we perform the train_test split as was given in the assignment.

After this we preprocess the data and did the following:

- Filtered out the empty rows from the dataset.
- Added an attribute of answer_end in the answer column of the dataset.

We then loaded the pre trained model on ‘TFAutoModelForQuestionAnswering’ and the tokenizer on ‘AutoTokenizer’ from [Skratch99/bert-pretrained · Hugging Face](#).

We then split the dataset using the methods given in the assignment and then trained the model on the training dataset for 3 epochs as well as 10 epochs.

We then pushed the model on Hugging Face:

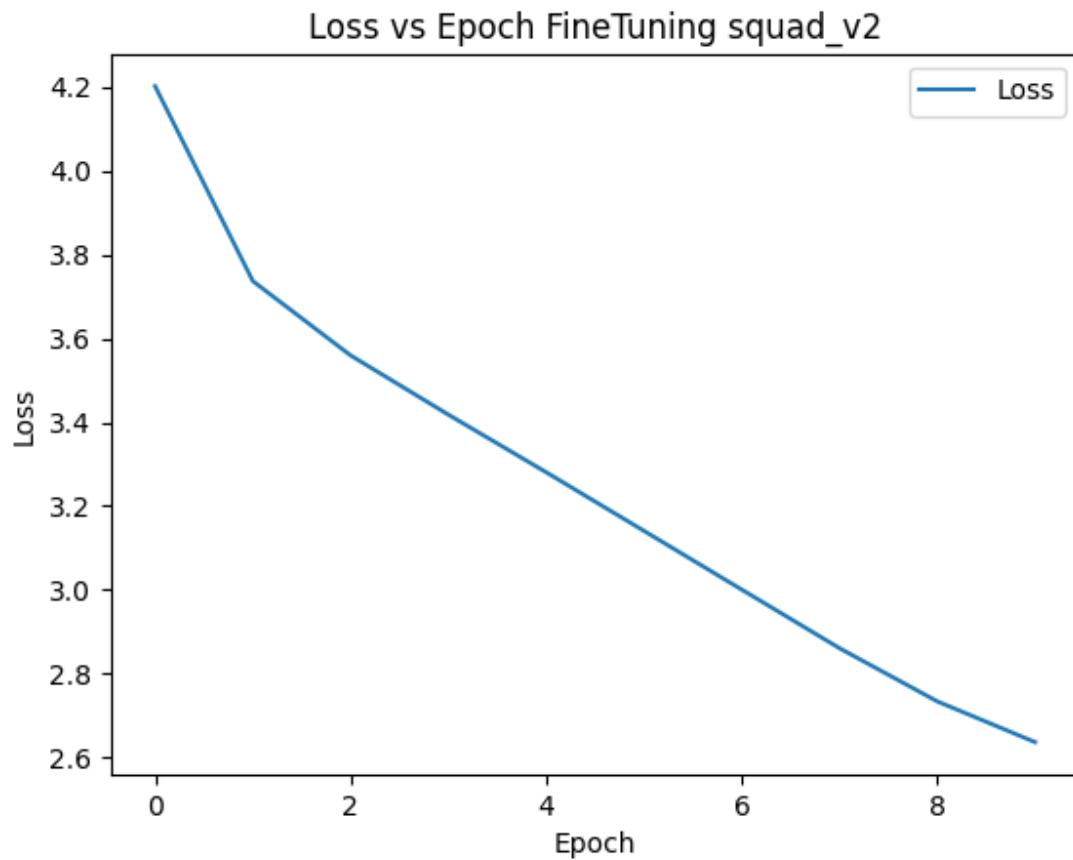
Trained on 3 Epochs:

[Skratch99/bert-finetuned-squadv2 · Hugging Face](#)

Trained on 10 Epochs:

[Skratch99/finetuned-bert-squadv2 · Hugging Face](#)

The loss while training for 10 Epochs can be seen in this plot:



7. Scores on Test Split

a. Classification

Metric	Value
Accuracy	0.5578322197475872
F1 score	0.716164696911933
Precision score	0.5578322197475872
Recall	1

b. QnA

For 3 Epochs:

Metric	Value
squad_v2	Results: {'exact': 9.3023, 'f1':17.9266389409888, 'total': 5928, 'HasAns_exact': 9.3023255819745682, 'HasAns_f1': 17.859266389409888, 'HasAns_total': 43, 'best_exact': 9.302325581974568, 'best_exact_thresh': 0.0, 'best_f1': 17.9266389409888, 'best_f1_thresh': 0.0}
F1	17.9266389409888
METEOR	29.7214587295
BLEU	{'bleu': 0.06316337326201324, 'precisions': [0.13942440099146242, 0.0737007582763085, 0.04661308840413318, 0.0332309410779112],

	'brevity_penalty': 1.0, 'length_ratio': 1.1939169749280722}
ROUGE	<pre>{'rouge1': AggregateScore(low=Score(precision= 0.3659287092181297, recall=0.45010879908258855, fmeasure=0.39420725650241517), mid=Score(precision=0.370462414717 9629, recall=0.45615310215275123, fmeasure=0.39849092233210226), high=Score(precision=0.374545205435 7944, recall=0.4613699628585486, fmeasure=0.4026998877301764)), 'rouge2': AggregateScore(low=Score(precision= 0.11608306959245825, recall=0.1460156434079125, fmeasure=0.12568868029506816), mid=Score(precision=0.119075039659 55783, recall=0.14957466879324813, fmeasure=0.12892569840384763), high=Score(precision=0.122204557582 61263, recall=0.1537522710858374, fmeasure=0.1324781377328654)), 'rougeL': AggregateScore(low=Score(precision= 0.260440283291041, recall=0.32031875018573314, fmeasure=0.28045675634386463), mid=Score(precision=0.263626035492 81477, recall=0.32402205753483726, fmeasure=0.2834183704951454),</pre>

	high=Score(precision=0.266801631098 1992, recall=0.32782000957434587, fmeasure=0.286454196739078)), 'rougeLsum': AggregateScore(low=Score(precision= 0.260656500152031, recall=0.32019107909196537, fmeasure=0.28044674518718316), mid=Score(precision=0.263709308552 1349, recall=0.3242258944877382, fmeasure=0.28347237052978547), high=Score(precision=0.266733095491 1444, recall=0.328069295890713, fmeasure=0.2863317161588235)))}
exact-match	9.3023

For 10 Epochs:

Metric	Value
squad_v2	{'exact': 10.878706199460916, 'f1': 18.90159908670889, 'total': 18550, 'HasAns_exact': 10.878706199460916, 'HasAns_f1': 18.90159908670889, 'HasAns_total': 18550, 'best_exact': 10.878706199460916, 'best_exact_thresh': 0.0, 'best_f1': 18.90159908670889, 'best_f1_thresh': 0.0}
F1	18.90159908670889
METEOR	34.42349882500762

BLEU	<pre>{'bleu': 0.07474411444487124, 'precisions': [0.16116218190673323, 0.08790346524154896, 0.055196789710064045, 0.03991398129744876], 'brevity_penalty': 1.0, 'length_ratio': 1.0560288019544184, 'translation_length': 65704, 'reference_length': 62218}</pre>
ROUGE	<pre>{'rouge1': AggregateScore(low=Score(precision= 0.3880581739816478, recall=0.4550323675149823, fmeasure=0.40900635426960424), mid=Score(precision=0.389353302496 93045, recall=0.4567322690085111, fmeasure=0.4103497252640948), high=Score(precision=0.390676030151 9545, recall=0.45862590291124583, fmeasure=0.4116893438014193)), 'rouge2': AggregateScore(low=Score(precision= 0.12528948929063083, recall=0.150201841445336, fmeasure=0.13317084526081496), mid=Score(precision=0.126323247483 28383, recall=0.1514989027008819, fmeasure=0.13426731369454203), high=Score(precision=0.127552530633 95253, recall=0.15298924635620328, fmeasure=0.13557166179535804)), 'rougeL':</pre>

	AggregateScore(low=Score(precision=0.27524800173164926, recall=0.32233724286555226, fmeasure=0.2899624542456411), mid=Score(precision=0.27616365718365277, recall=0.3235875644282537, fmeasure=0.2908665929400386), high=Score(precision=0.2771266845676933, recall=0.324837436794631, fmeasure=0.29187489300822833)), 'rougeLsum': AggregateScore(low=Score(precision=0.2752319220963203, recall=0.322357382059127, fmeasure=0.289915588313673), mid=Score(precision=0.27619126243883463, recall=0.32358788119911713, fmeasure=0.2908832134862902), high=Score(precision=0.2772057732795161, recall=0.3247918159208708, fmeasure=0.29187799639799783)))}
exact-match	10.878706199460916

8-9. Model Parameters before and after Tuning

The number of model parameters for the pre-trained model were 110106428. The number of parameters for the fine-tuned models was as follows:

- Question-answering model: 108893186
- Classification model: 109483778

We can see that the parameters do not remain the same, and they have instead decreased from the pre-trained model. The reason for this is mentioned in point 10 b).

The Hugging Face Links for the pushed Fine Tuned models:

Fine Tune Classification: [Nokzendi/bert_sst2_finetuned · Hugging Face](#)

Fine Tune Question-Answering: [Skratch99/finetuned-bert-squadv2 · Hugging Face](#)

10. Rationale behind

a. Performance

Classification Task

The above scores are calculated on the test split of the SST-2 dataset. After fine-tuning, the model reports a high accuracy of 0.55. The model's accuracy is not very good as it is trained for very few epochs. Training for higher epochs will show promising results as the finetuned model is performing better than the pre-trained model. So increasing the number of epochs will improve the results as the dataset is very large, and such a small number of epochs is not enough. But training on more epochs on currently available configurations(GPU on colab) is not plausible.

Question Answering Task

The scores have been calculated on the squad_v2 dataset for the Question Answering task. Fine tuning has improved the model as seen from the comparison between the metrics on the three epochs model vs. the ten epoch model. While this improvement hasn't been much thus finetuning the model for more epochs might not be the best approach to increase these metrics. The F1 Score of around 18.90% showcases the model's ability to provide relevant information in its answers. While the F1 for unigrams as seen from Rouge is 40.90% which is a moderate outcome. The BLEU score is relatively low at 7.47%, suggesting that the model's outputs may not align closely with the reference answers. Overall, the model demonstrates a moderate level of performance, with room for improvement. While it provides some correct answers (as indicated by the EM and F1 scores), there is potential for enhancement in terms of

generating more precise and contextually accurate responses, as reflected in the Rouge, Meteor, and BLEU scores.

The performance of our fine-tuned models is not as good as expected and doesn't give expected answers, which can also be justified by the poor metric scores for respective tasks. This might be due to the following reasons:

- More number of fine-tuning epochs are required. (The computational overhead binds us since the Google Colab resources limit us to not run for more epochs than we have)
- The dataset we have might have an imbalance in the train and test splits.
- The pre-training is not done on a varied enough dataset or more epochs are required.
- Additionally, initial data preprocessing techniques could also have used techniques like data augmentation to increase the size of the dataset for both pre-training and fine-tuning for better outcomes.
- We could also have experimented with different hyperparameter configurations to find what best works for the model.

b. Number of parameters between Pre-Training and Fine Tuning

We infer that the number of parameters differs between the pre-trained and both the fine-tuned models in the sense that the pre-trained model has more parameters than the fine-tuned models.

The number of parameters in

- Pretrained - $p_1 = 110106428$
- Finetuned_q&a - $p_2 = 108893186$
- Finetuned_classification - $p_3 = 109483778$

This can be justified by inspecting the architectures of the models:

Pre-trained Model:

```
print(pre_trained_model)

BertForPreTraining(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0-11): 12 x BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (cls): BertPreTrainingHeads(
    (predictions): BertLMPredictionHead(
      (transform): BertPredictionHeadTransform(
        (dense): Linear(in_features=768, out_features=768, bias=True)
        (transform_act_fn): GELUActivation()
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      )
      (decoder): Linear(in_features=768, out_features=30522, bias=True)
    )
    (seq_relationship): Linear(in_features=768, out_features=2, bias=True)
  )
)
```

Question-Answering Finetuned Model:

```
print(q_a_finetuned)

BertForQuestionAnswering(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0-11): 12 x BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
  )
  (qa_outputs): Linear(in_features=768, out_features=2, bias=True)
)
```

We can see that the finetuned Question Answering architecture differs from the pretrained architecture with respect to the following:

Pretrained has BertPooler and BertPreTrainingHeads modules while finetuned_q&a has qa_outputs layer.

BertForPreTraining is a Bert Model with two heads on top as done during the pretraining: a masked language modeling head and a next sentence prediction (classification) head.

The seq_relationship layer helps in Next Sentence Prediction while the BertLMPredictionHead does so in MLM which are not there in the finetuned_q&a since it is not required for this

specific task.

On the other hand, BertForQuestionAnswering is a Bert Model with a span classification head on top for extractive question-answering tasks like SQuAD (linear layers on top of the hidden-states) and therefore the extra qa_outputs layer.

If calculated, it can be seen that the pooler and cls modules correspond to more parameters than the qa_outputs module, and hence,

$p_1 > p_2$

Classification Finetuned Model:

```
print(classification_finetuned)

BertForSequenceClassification(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0-11): 12 x BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (classifier): Linear(in_features=768, out_features=2, bias=True)
)
```

We can see that the finetuned Question Answering architecture differs from the pre-trained architecture with respect to the following modules:

Pretrained has the BertPreTrainingHeads module extra while the classification_finnetuned has the classifier layer extra.

This again can be allocated to the task-specific functionality of the classification_finnetuned model.

Again, If calculated, it can be seen that the additional modules and layers in pre trained correspond to more number of parameters than the classifier layer in classification_finnetuned and hence,

p1>p3

12. Hugging Face Links and Metrics :

Pre-Trained Model: [Skratch99/bert-pretrained · Hugging Face](#)

Fine Tune Classification: [Nokzendi/bert_sst2_finnetuned · Hugging Face](#)

Fine Tune Question-Answering: [Skratch99/finnetuned-bert-squadv2 · Hugging Face](#)

The computation of metrics of all these public models are done in the “eval” files in the git repo.

12. Contributions

Aditi Agarwal: Worked on metrics calculations and inference for the test splits for classification and question-answering models. Worked on optimization like parallel processing and partitioning to speed up the evaluation of a large dataset.

Harshvardhan Vala: Worked on perplexity score measurement on the test split and parameter calculations for the model.

Inderjeet Singh: Worked on fine-tuning of the pre-trained model for the question-answering task on the squad_v2 dataset. Helped in optimizing the code at various places to speed up the process. Helped fix loads of bugs in the assignment.

Joy Makwana: Worked on the perplexity score measurement of the test splits and measured the trends across epochs.

Kalash Kankaria: Worked on pre-training of the Bert model on the Wikitext dataset and calculation of parameters of the pre-trained model. Contributed to the inferences for the 10th point (performance evaluation and parameter changes understanding)

Kanishk Singhal: Worked on evaluation of the classification fine-tuned model on the SST-2 dataset.

Medhansh Singh: Worked on metrics calculations for the test splits for classification and question-answering models.

Nokzendi Aier: Worked on fine-tuning of the pre-trained model for the classification task on the SST-2 dataset. Helped in debugging and evaluation.