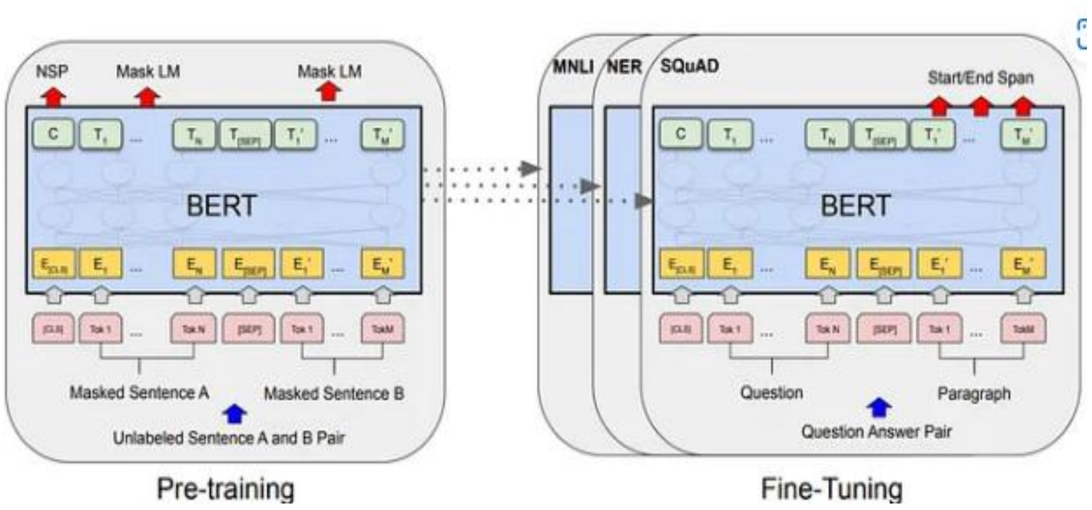
Experiment No 7

Aim: Fine tuning BERT model to perform Natural Language Processing task.

Theory:

BERT

BERT stands for **B**idirectional **E**ncoder **R**epresentations from **T**ransformers and is a language representation model by Google. It uses two steps, pre-training and fine-tuning, to create state-of-the-art models for a wide range of tasks. Its distinctive feature is the unified architecture across different downstream tasks — what these are, we will discuss soon. That means that the same pre-trained model can be fine-tuned for a variety of final tasks that might not be similar to the task model was trained on and give close to state-of-the-art results.



BERT Architecture

BERT has to differ Architecture BERT Base and BERT Large
BERT Base: L=12, H=768, A=12.
Total Parameters=110M!
BERT Large: L=24, H=1024, A=16.
Total Parameters=340M!!
L = Number of layers (i.e., #Transformer encoder blocks in the stack).
H = Hidden size (i.e. the size of *q, k* and *v* vectors).
A = Number of attention heads.

Pre-training BERT

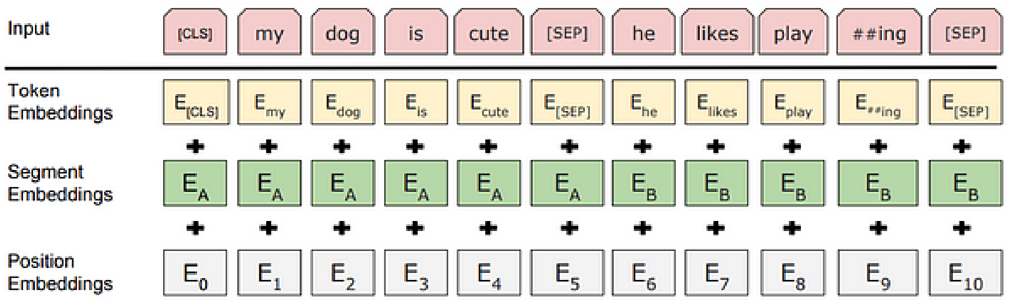
The BERT model is trained on the following two unsupervised tasks.
1. Masked Language Model (MLM)
This task enables the deep bidirectional learning aspect of the model. In this task, some percentage of the input tokens are masked (Replaced with [*MASK*] token) at random and the model tries to predict these masked tokens — not the entire input sequence. The predicted tokens from the model are then fed into an output softmax over the vocabulary to get the final output words.

This, however creates a mismatch between the pre-training and fine-tuning tasks because the latter does not involve predicting masked words in most of the downstream tasks. This is mitigated by a subtle twist in how we mask the input tokens.
Approximately 15% of the words are masked while training, but all of the masked words are not replaced by the [*MASK*] token.
80% of the time with [*MASK*] tokens.
10% of the time with a random tokens.
10% of the time with the unchanged input tokens that were being masked.

2. Next Sentence Prediction (NSP)

The LM doesn't directly capture the relationship between two sentences which is relevant in many downstream tasks such as Question Answering (QA) and Natural Language Inference (NLI). The model is taught sentence relationships by training on binarized NSP task.
In this task, two sentences — A and B — are chosen for pre-training.
50% of the time B is the actual next sentence that follows A.
50% of the time B is a random sentence from the corpus.
Training — Inputs and Outputs.

The model is trained on both above mentioned tasks simultaneously. This is made possible by clever usage of inputs and outputs.
Inputs



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}

The input representation for BERT

The model needs to take input for both a single sentence or two sentences packed together unambiguously in one token sequence. Authors note that a "sentence" can be an arbitrary span of contiguous text, rather than an actual linguistic sentence. A [SEP] token is used to separate two sentences as well as a using a learnt segment embedding indicating a token as a part of segment A or B.

Problem #1*:* All the inputs are fed in one step — as opposed to RNNs in which inputs are fed sequentially, the model is ***not able to preserve the ordering*** of the input tokens. The order of words in every language is significant, both semantically and syntactically.

Problem #2*:* In order to perform Next Sentence Prediction task properly we need to be able to ***distinguish between sentences A and B***. Fixing the lengths of sentences can be too restrictive and a potential bottleneck for various downstream tasks.

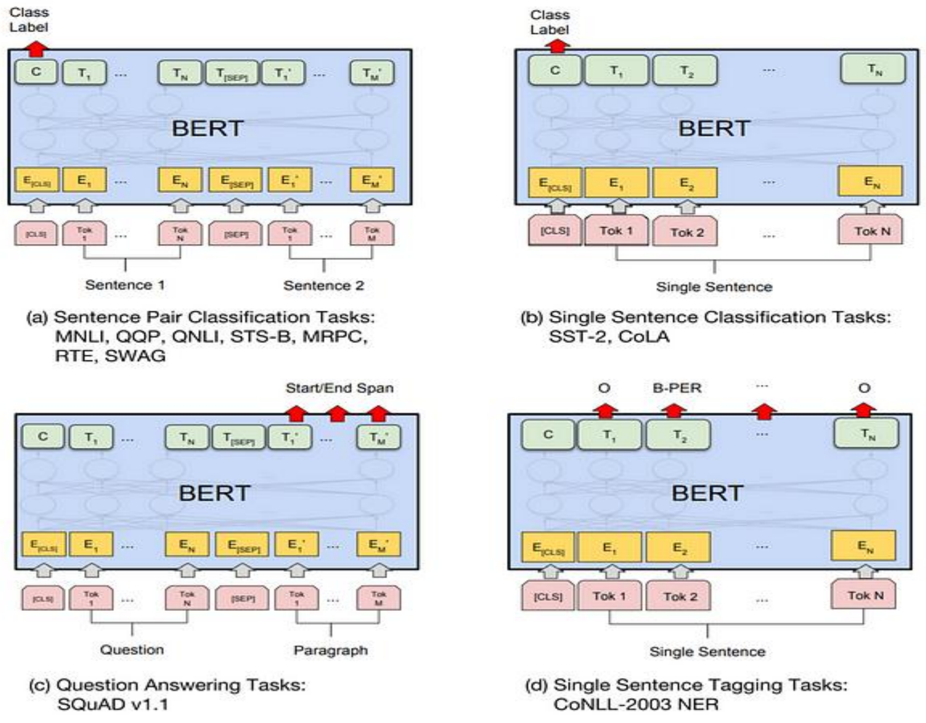
Both of these problems are solved by adding embeddings containing the required information to our original tokens and using the result as the input to our BERT model. The following embeddings are added to token embeddings:

Segment Embedding: They provide information about the sentence a particular token is a part of.

Position Embedding: They provide information about the order of words in the input.
Outputs

Fine-tuning BERT

Fine-tuning on various downstream tasks is done by swapping out the appropriate inputs or outputs. In the general run of things, to train task-specific models, we add an extra output layer to existing BERT and fine-tune the resultant model — all parameters, end to end. A positive consequence of adding layers — input/output and not changing the BERT model is that only a minimal number of parameters need to be learned from scratch making the procedure fast, cost and resource efficient.
Just to give you an idea of how fast and efficient it is, the authors claim that all the results in the paper can be replicated in *at most 1 hour* on a *single Cloud TPU*, or *a few hours on a GPU*, starting from the exact same pre-trained model.


Class Label
BERT
Sentence 1
Sentence 2
(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG
Class Label
BERT
Single Sentence
(b) Single Sentence Classification Tasks: SST-2, CoLA
Start/End Span
BERT
Question
Paragraph
(c) Question Answering Tasks: SQuAD v1.1
O
B-PER
O
BERT
Single Sentence
(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

Fine-tuning BERT on various downstream tasks.

In Sentence Pair Classification and Single Sentence Classification, the final state corresponding to [*CLS*] token is used as input for the additional layers that makes the prediction.
In QA tasks, a start (S) and an end (E) vector are introduced during fine tuning. The question is fed as sentence A and the answer as sentence B. The probability of word *i* being the start of the answer span is computed as a dot product between T*i* (final state corresponding to *i*th input

token) and S (start vector) followed by a softmax over all of the words in the paragraph. A similar method is used for end span.

Advantages of Fine-Tuning

Quicker Development

First, the pre-trained BERT model weights already encode a lot of information about our language. As a result, it takes much less time to train our fine-tuned model - it is as if we have already trained the bottom layers of our network extensively and only need to gently tune them while using their output as features for our classification task. In fact, the authors recommend only 2-4 epochs of training for fine-tuning BERT on a specific NLP task (compared to the hundreds of GPU hours needed to train the original BERT model or a LSTM from scratch!).

Less Data

In addition, and perhaps just as important, because of the pre-trained weights this method allows us to fine-tune our task on a much smaller dataset than would be required in a model that is built from scratch. A major drawback of NLP models built from scratch is that we often need a prohibitively large dataset in order to train our network to reasonable accuracy, meaning a lot of time and energy had to be put into dataset creation. By fine-tuning BERT, we are now able to get away with training a model to good performance on a much smaller amount of training data.

Better Results

Finally, this simple fine-tuning procedure (typically adding one fully-connected layer on top of BERT and training for a few epochs) was shown to achieve state of the art results with minimal task-specific adjustments for a wide variety of tasks: classification, language inference, semantic similarity, question answering, etc. Rather than implementing custom and sometimes-obscure architectures shown to work well on a specific task, simply fine-tuning BERT is shown to be a better (or at least equal) alternative.

Steps to Fine Tune BERT Model to perform Multi Class Text Classification

1. **Load dataset**
2. **Pre-process data**
3. **Define model**
4. **Train the model**
5. **Evaluate**

Lab Exercise to be Performed in this Session:

Perform Text Classification by Fine tuning BERT model.