**Training a Question-Answering Machine learning model- A case study**

1. **Introduction:**

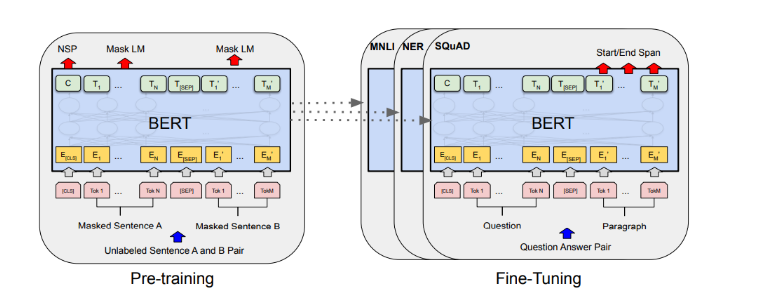
Question-Answering Models are machine or deep learning models that can answer questions given some context, and sometimes without any context (e.g. open-domain QA). They can extract answer phrases from paragraphs, paraphrase the answer generatively, or choose one option out of a list of given options, and so on. It all depends on the dataset it was trained on or the problem it was trained for, or to some extent the neural network architecture. So, for example, if you feed this paragraph (context) to your model trained to extract answer phrases from context, and ask a question like "What is a question-answering model?", it should output the first line of this paragraph. Such models need to understand the structure of the language, have a semantic understanding of the context and the questions, have an ability to locate the position of an answer phrase, and much more. So, without any doubt, it is difficult to train models that perform these tasks. Fortunately, the concept of attention in neural networks has been a lifesaver for such difficult tasks. Since [its introduction](https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html#a-family-of-attention-mechanisms) for sequence modelling tasks, lots of RNN networks with sophisticated attention mechanisms like [R-NET](https://www.microsoft.com/en-us/research/wp-content/uploads/2017/05/r-net.pdf), [FusionNet](https://arxiv.org/pdf/1711.07341.pdf), etc. have shown great improvement in QA tasks. However, a completely new neural network architecture based on attention, specifically self-attention, called [Transformer](https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html), has been the real game-changer in NLP. Here I will discuss one such variant of the Transformer architecture called BERT, with a brief overview of its architecture, how it performs a question answering task, and then write our code to train such a model to answer COVID-19 related questions from research papers. A language model is a probabilistic model that learns the probability of the occurrence of a sentence, or sequence of tokens, based on the examples of text it has seen during training. For example: We have certain different probability density functions for the prediction of multi label classification related problems in the situation of Sentence classification.

**P(That which does not kill us make us stronger) = P(That) P(which|That) P(does|That,which) P(not|That,which,does).**

**P(That which does not kill us make us stronger) = 0.65**

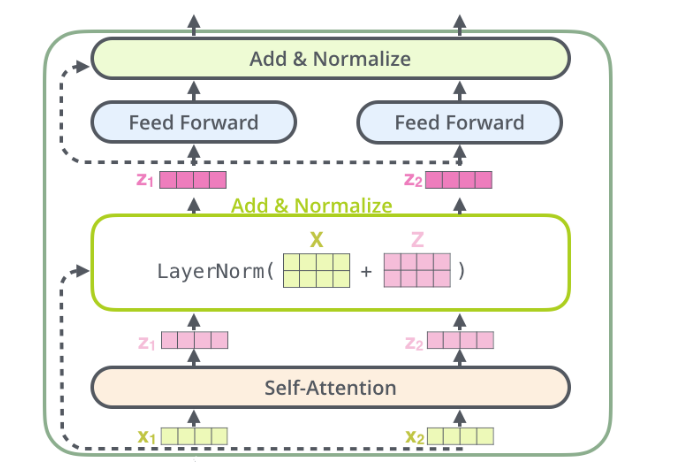
These language models, if big enough and trained on a sufficiently large dataset, can start understanding any language and its intricacies really well. Traditionally RNNs were used to train such models due to the sequential structure of language, but they are slow to train (due to sequential processing of each token) and sometimes difficult to converge (due to vanishing/exploding gradients).

However, different variants of Transformers, with their ability to process tokens in parallel and impressive performance due to self-attention mechanism and different pre-training objectives, have made training large models (and sometimes [really really large models](https://arxiv.org/abs/2005.14165)), which understand natural language really well, possible. Different Transformer-based language models, with small changes in their architecture and pre-training objective, perform differently on different types of tasks. [BERT](https://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html) (Bidirectional Encoder Representations from Transformers) is one such model. BERT has been trained using the Transformer Encoder architecture, with Masked Language Modelling (MLM) and the Next Sentence Prediction (NSP) pre-training objective.



**Figure 1: Architecture of BERT**

Now that we know what BERT is, let us go through its architecture and pre-training objectives briefly. BERT uses Transformer Encoder from the [original Transformer paper](https://arxiv.org/abs/1706.03762). An Encoder has a stack of encoder blocks (where the output of one block is fed as the input to the next block), and each encoder block is composed of two neural network layers. First there is a self-attention layer (which is the magic operation that makes transformers so powerful) and then a simple feed-forward layer. After each layer, there is a residual connection and a layer normalization operation as shown in the figure below.



**Figure 2: Structure of One-Encoder Block**

So, for each encoder layer, the number (with a maximum limit of 512) of input vectors and output vectors is always the same. And before the first encoder layer, the input vector for each token is obtained by adding token embedding, positional embedding, and segment embedding. These vectors are processed in parallel inside each encoder layer using matrix multiplications, and the obtained output vectors are fed to the next encoder block. After being processed sequentially through *N* such blocks, the obtained output vectors start understanding natural language very well.

1. **Pre-Training Objective**

A pre-training objective is a task on which a model is trained before being fine-tuned for the end task. GPT models are trained on a Generative Pre-Training task (hence the name GPT) i.e. generating the next token given previous tokens, before being fine-tuned on, say, SST-2 (sentence classification data) to classify sentences.

Similarly, BERT uses MLM and NSP as its pre-training objectives. It uses a few special tokens like CLS, SEP, and MASK to complete these objectives. We will see the use of these tokens as we go through the pre-training objectives. But before proceeding, we should know that each tokenized sample fed to BERT is appended with a CLS token in the beginning and the output vector of CLS from BERT is used for different classification tasks. Now let's start with MLM.

In the MLM objective, a percentage of tokens are masked i.e. replaced with special token MASK, and the model is asked to predict the correct token in place of MASK. To accomplish this a masked language model head is added over the final encoder block, which calculates a probability distribution over the vocabulary only for the output vectors (output from the final encoder block) of MASK tokens. And in NSP, the two sentences tokenized and the SEP token appended at their end are concatenated and fed to BERT. The output vector of the CLS token is then used to calculate the probability of whether the second sentence in the pair is the subsequent sentence in the original document. For both the objectives, standard cross-entropy loss with AdamW optimizer is used to train the weights.

The above pre-training objectives are really powerful in capturing the semantics of the natural language in comparison to other pre-training objectives, e.g. the generative pre-training objective. Hence, many models with similar or slightly tweaked pre-training objectives, with more or less the same architecture as BERT, have been trained to achieve SOTA results on many NLP tasks. [RoBERTA](https://arxiv.org/abs/1907.11692), [SpanBERT](https://www.cs.princeton.edu/~danqic/papers/tacl2020.pdf), [DistilBERT](https://arxiv.org/abs/1910.01108), [ALBERT](https://arxiv.org/abs/1909.11942) etc. are a few of them.

1. **Question Answering** 
   1. **Dataset**

As mentioned before, the QA task can be framed in different ways. Here I will be focusing on context-based question answering, where questions are asked from a given paragraph. SQuAD is a popular dataset for this task which contains many paragraphs of text, different questions related to the paragraphs, their answers, and the start index of answers in the paragraph. There are two versions of SQuAD, SQuAD1.1 and SQuAD2.0, with the main difference being that SQuAD2.0 contains over 50,000 unanswerable questions that look similar to the answerable ones. So to do well on SQuAD2.0, systems must not only answer questions when possible, but also determine when no answer is supported by the paragraph and abstain from answering

* 1. **BERT SQuAD Architecture**

To perform the QA task we add a new question-answering head on top of BERT, just the way we added a masked language model head for performing the MLM task. The purpose of this question-answering head is to find the start token and end token of an answer for any given paragraph. Everything that comes in between, including the start and end token, is considered an answer.

For instance

Consider the Paragraph for the processing: “BERT-large is really big………………… it has 24 layers and an embedding size of 1024 for a total of 340M parameters! Altogether it is 1.34GB, so expect it to take a couple of minutes download.  
**Question**: How many parameters does BERT-large have?  
**Answer**: 340 M Parameters

Start Token: 340

End Token: Parameters

Inside the question answering head are two sets of weights, one for the start token and another for the end token, which have the same dimensions as the output embeddings. The output embeddings of all the tokens are fed to this head, and a dot product is calculated between them and the set of weights for the start and end token, separately. In other words, the dot product between the start token weight and output embeddings is taken, and the dot product between the end token weight and output embeddings is also taken. Then a softmax activation is applied to produce a probability distribution over all the tokens for the start and end token set (each set also separately). The tokens with the maximum probability are chosen as the start and end token, respectively. In this process, it may so happen that the end token could appear before the start token.

* 1. **Training a QnA Model**

We will be using Hugging Face's [Transformers](https://github.com/huggingface/transformers/) library for training our QA model. We will also be using BioBERT, which is a language model based on BERT, with the only difference being that it has been finetuned with MLM and NSP objectives on different combinations of general & biomedical domain corpora. Different domains have specific jargons and terms which occur very rarely in standard English, and if they occur it could mean different things, or imply different contexts. Hence, models like BioBERT, LegalBERT, etc. have been trained to learn such nuances of the domain-specific text so that domain-specific NLP tasks could be performed with better accuracy.

Here we aim to use the QA model to extract relevant information from COVID-19 research literature. Hence, we will be finetuning BioBERT using Hugging Face's Transformers library on SQuADv2 data.

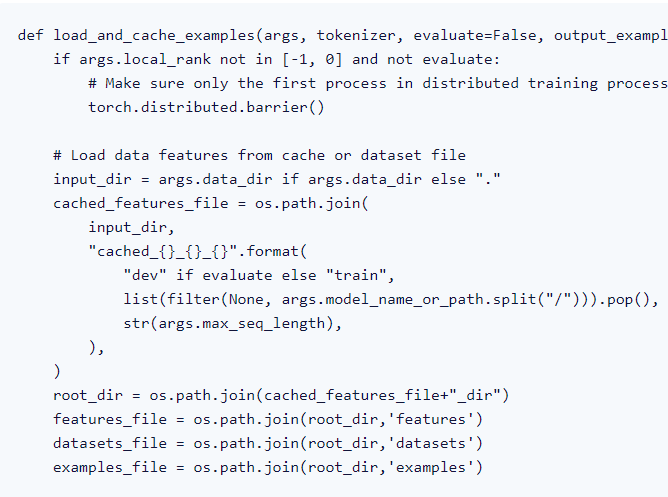
In the examples section of the Transformers repository, Hugging Face has already provided a script, run\_squad.py, to train the QA model on SQuAD data. This script can be run easily using the below command.



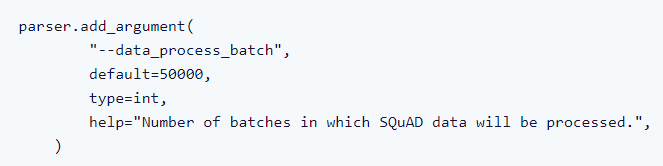
Figure 3: Python Script

One can understand most of the parameters from their names. For more details on the parameters and an exhaustive list of parameters that can be adjusted, one can refer to the run\_squad.py [script](https://github.com/huggingface/transformers/blob/master/examples/question-answering/run_squad.py).

Using this script, the model can be easily finetuned to perform the QA task. However, running this script is RAM heavy, because squad\_convert\_examples\_to\_features tries to process the complete SQuAD data at once and requires more than 12GB of RAM. So, I have modified load\_and\_cache\_examples and added a new function named read\_saved\_data which can process SQuAD data in batches. You can check out these methods below.



Basically, the added modifications run the same method squad\_convert\_examples\_to\_features on mini-batches of data and save the created features in a folder. One can define the minibatch size by adding the below line at line 660 in run\_squad.py, and providing an argument data\_process\_batch in the command I mentioned above.



1. Conclusion:

Question answering is a critical NLP problem and a long-standing artificial intelligence milestone. QA systems allow a user to express a question in natural language and get an immediate and brief response. QA systems are now found in search engines and phone conversational interfaces, and they’re fairly good at answering simple snippets of information. On more hard questions, however, these normally only go as far as returning a list of snippets that we, the users, must then browse through to find the answer to our question.