# Project Report on Question Answering

Submitted by :-

**Team 16**
**Rahul Padhy** (2022201003)
**Arun Das** (2022201021)

## Introduction :-

Question and Answering (henceforth referred to as QnA) is a computer science discipline within the fields of information retrieval and natural language processing, which focuses on building systems that automatically answer questions posed by humans in a natural language. QnA systems are now found in search engines and phone conversational interfaces, and they're fairly good at answering simple snippets of information. On harder questions, however, these normally only go as far as returning a list of snippets that the users must then browse through to find the answer to our question. QnA models are often used to automate the response to frequently asked questions by using a knowledge base (e.g. documents) as context.

QnA systems differ in the way answers are created :-

- Extractive QnA - The model extracts the answer from a context and provides it directly to the user. It is usually solved with BERT-like models.

- Generative QnA - The model generates free text directly based on the context. It leverages text generation models.

QnA systems also differ in where answers are taken from :-

- Open QnA - The answer is taken from a context.
- Closed QnA - No context is provided and the answer is completely generated by a model.

This project aims to build an **Extractive, Open QnA model**. For achieving this task, the following 2 datasets have been explored - SQuAD 1.0 and SQuAD 2.0.

## Datasets :-

The **SQuAD** is a popular benchmark for evaluating the performance of question answering systems. It consists of over 100,000 question-answer pairs, based on over 500 Wikipedia articles, which are used for training and evaluation of machine learning models. The answer to each question is a segment of text or span from the corresponding reading passage. Sometimes the questions might be unanswerable. SQuAD has 2 versions. **SQuAD 1.1**, contains 100,000+ question-answer pairs on 500+ articles. **SQuAD2.0** combines the 100,000 questions in **SQuAD1.1** with over 50,000 unanswerable questions written adversarially by crowdworkers to look similar to answerable ones.

Training on **SQuAD2.0** is tougher as we not only have to answer the questions, but also abstain from answering when the question is unanswerable. So extra effort is needed to determine if the question is unanswerable.

## Evaluation Metrics for QnA :-

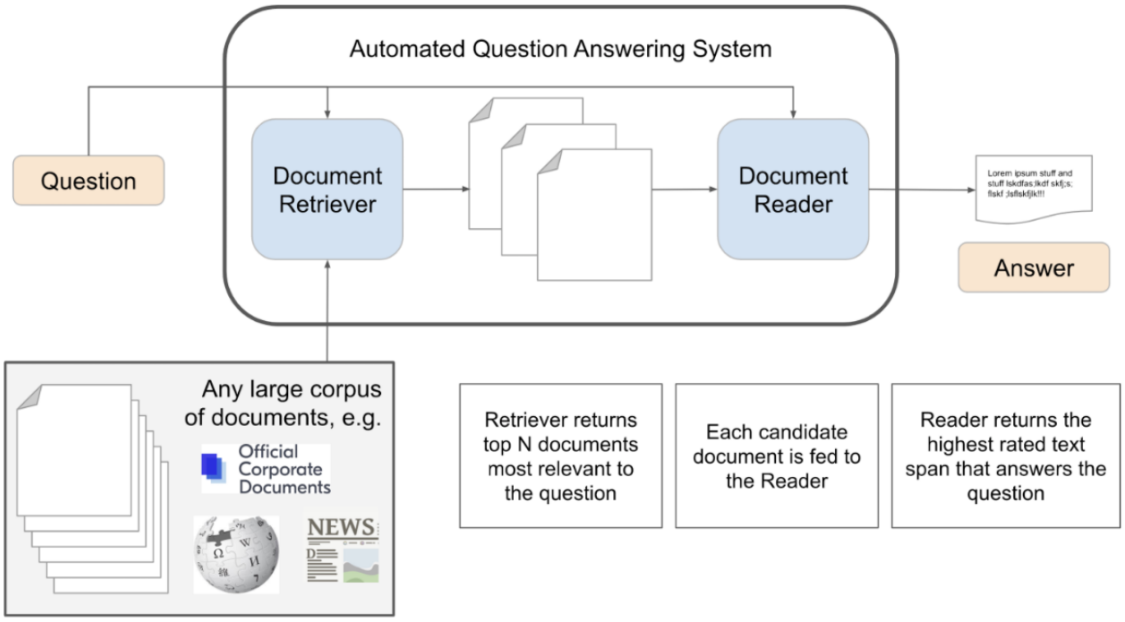
- The **Exact Match (EM) score** metric measures the percentage of questions for which the model produces an exact match with the correct answer. A predicted answer is considered correct if it exactly matches the gold standard answer in the dataset. The EM score ranges from 0 to 100, with higher scores indicating better performance.
- The **F1 score** metric measures the average overlap between the predicted answer and the correct answer, based on the harmonic mean of precision and recall. Precision is the ratio of the number of correct answers to the total number of predicted answers, and recall is the ratio of the number of correct answers to the total number of gold standard answers. The F1 score ranges from 0 to 100, with higher scores indicating better performance.
- **HuggingFace Pipeline Score** is used to measure the relevance of an answer extracted from a certain context, for the BERT-based models.

Both EM and F1 scores are commonly used to evaluate the performance of machine learning models on the SQuAD datasets. In general, higher scores on both metrics indicate better performance, and a model with high scores on both metrics is considered to be a strong performer. However, the choice of metric may depend on the specific task and application, and other metrics such as precision, recall, and accuracy may also be used to evaluate performance in different contexts.

## Approaches Undertaken :-

- Document Retrieval
- Transformer-based Models' Answer Extraction

## Document Retrieval :-




*Generic IR QA system*

**Fig.**https://qa.fastforwardlabs.com/methods/background/2020/04/28/Intro-to-QA.html

- The data is converted into an appropriate format (Pandas dataframe) from JSON.
- Documents are encoded and questions are transformed into vectors using TF-IDF or Word2Vec.
- Search for a question is performed by comparing with the document vectors.
- The most similar document (NearestNeighbor) is returned.
- The actual **context_id** is checked against the returned context - if it's present, then the process is marked as a success, otherwise a failure.

## Accuracy Scores for Document Retrieval :-

Here, for both TF-IDF and Word2Vec, the vector-size was fixed as 300 and the corresponding accuracy scores on the datasets are documented below :-

| | Accuracy for Document Retrieval using TF-IDF | Accuracy for Document Retrieval using Word2Vec |
|---|---|---|
| SQuAD 1.0 Train | 0.0141 | 0.2151 |
| SQuAD 1.0 Validation | 0.065 | 0.1376 |
| SQuAD 2.0 Train | 0.0123 | 0.2018 |
| SQuAD 2.0 Validation | 0.0123 | 0.1072 |

QnA is a difficult problem, since we have multiple documents and the answer can be in 1 or more documents. So, the retriever should return 'k' documents (where, k > 1), as it's complete / fair to return only 1 document.

Reasons as to why both TF-IDF and Word2Vec underperformed :-

- They are able to compute similarity between questions and documents that present the same words, so they are unable to capture synonyms properly.
- They cannot understand the question's context or the meaning of the words.

The accuracy is observed to have been increased when >= 3 documents are returned, instead of the 1 taken above. But still, Document Retrieval isn’t considered to be an efficient technique for QnA. Many QA systems adopt the approach of retrieving large

sets of documents using bag-of-words retrieval with question keywords, then exhaustively post-processing to check the higher-level constraints that indicate a likely answer. 1 criticism of this approach is that it is difficult to know a-priori how many documents to retrieve; even a very large set of results may not contain an answer if the key terms are very common. A second criticism is that it can be slow; though they aim for a ‘large enough’ set of documents, systems may retrieve and process more text than is really required to answer a question. Interactive systems are particularly vulnerable, as they must balance the amount of processing against the number of results processed while the user waits.

## Approach to QnA using Transformer-based Models (BERT, specifically) :-

Initially, Recurrent Neural Networks (RNNs) and Long Short Term Memory Networks (LSTMs) were used widely for question-answering tasks. An RNN is a generalization of a feedforward neural network that has an internal memory. RNNs can use their internal state (memory) to process sequences of inputs. LSTM networks are a modified version of RNNs, which makes it easier to remember past data in memory. The vanishing gradient problem of RNN is resolved here. An LSTM is well-suited to classify, process and predict time series given time lags of unknown duration. It trains the model by using back-propagation.

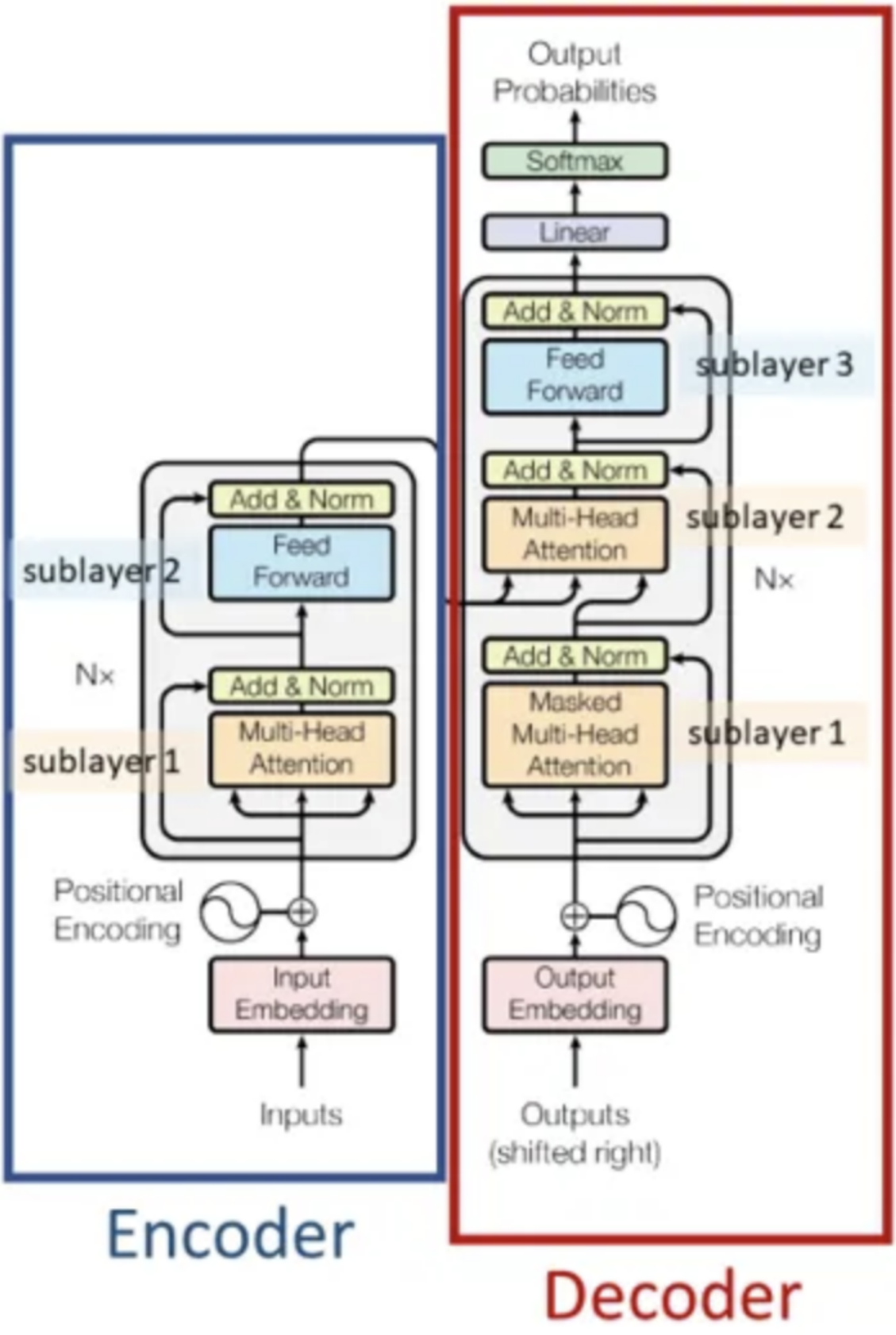
Both RNNs and LSTMs have the following 2 core properties :-

- **Sequential processing :** sentences must be processed word by word.
- **Past information retained through past hidden states :** Sequence-to-Sequence models follow the Markov property,

each state is assumed to be dependent only on the previously seen state.

The first property is the reason why RNNs and LSTMs can't be trained in parallel. In order to encode the second word in a sentence, we need the previously computed hidden states of the first word, therefore we need to compute that first. Information in RNNs and LSTMs are retained thanks to previously computed hidden states. The point is that the encoding of a specific word is retained only for the next time step, which means that the encoding of a word strongly affects only the representation of the next word, its influence is quickly lost after a few time steps. LSTMs (and also GRUs) can boost the dependency range they can learn thanks to a deeper processing of the hidden states through specific units (which comes with an increased number of parameters to train) but nevertheless the problem is inherently related to recursion. Another way in which people mitigated this problem is to use Bi-directional models, which encode the same sentence from two direction, from the start to end and from the end to the start, allowing this way words at the end of a sentence to have stronger influence in the creation of the hidden representation, but this is just a workaround rather than a real solution for very long dependencies.

The paper **Attention is all you need** describes transformers and what is called a sequence-to-sequence (Seq2Seq) architecture. Sequence-to-sequence is a neural net that transforms a given sequence to another sequence for a specific task. The most famous application of Seq2Seq models is translation, where the sequence of words from one language is transformed into a sequence of words in another language. A popular choice for this type of model is the LSTM based model. However, for a long sequence LSTM has slow training time and missing information (long-range dependency problem). So the Transformers model was born to solve these problems of LSTMs. The attention mechanism

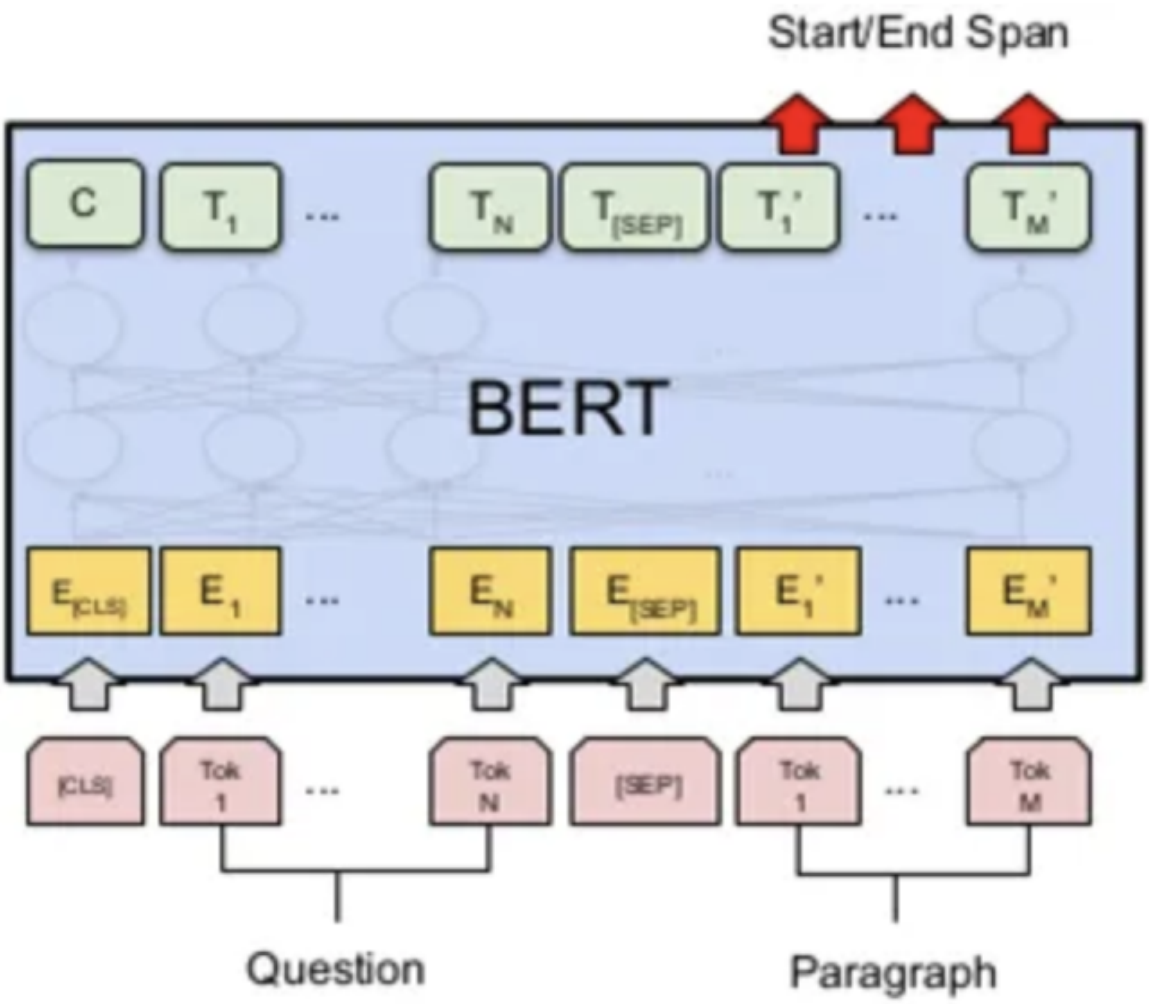

Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
sublayer 3
Add & Norm
Multi-Head Attention
sublayer 2
Nx
Add & Norm
Masked Multi-Head Attention
sublayer 1
Positional Encoding
Output Embedding
Outputs (shifted right)
Add & Norm
Feed Forward
sublayer 2
Add & Norm
Multi-Head Attention
sublayer 1
Nx
Positional Encoding
Input Embedding
Inputs
Encoder
Decoder

replaced the recurrent mechanism.

Transformers were introduced in the context of machine translation with the purpose to avoid recursion in order to allow parallel computation (to reduce training time) and also to reduce drop in performances due to long dependencies. The attention-mechanism looks at an input sequence and decides at each step which other parts of the sequence are important. It sounds abstract, but let's clarify with an easy example: When reading a piece of text, we always focus on the word we read but at the same time our mind still holds the important keywords of the text in memory in order to provide context.

BERT, which stands for Bidirectional Encoder Representations from Transformers developed by researchers at Google in 2018, is based on Transformers, a deep learning model in which every output element is connected to every input element, and the weightings between them are dynamically calculated based upon their connection. It is designed to pre-train deep bidirectional representations from an unlabeled text by jointly conditioning on both the left and right contexts. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of NLP tasks. BERT stacks multiple transformer encoders on top of each other. The transformer is based on the famous multi-head attention module which has shown substantial success in both vision and language tasks.

BERT helps the search engine understand the significance of transformer words like ‘to’ and ‘for’ in the keywords used. For the Question Answering System, BERT takes two parameters, the input question, and passage as a single packed sequence. Then we fine-tune the output to display the answer that exists in the passage.




## Our original Fine-Tuning implementation of BERT base model (uncased) :-

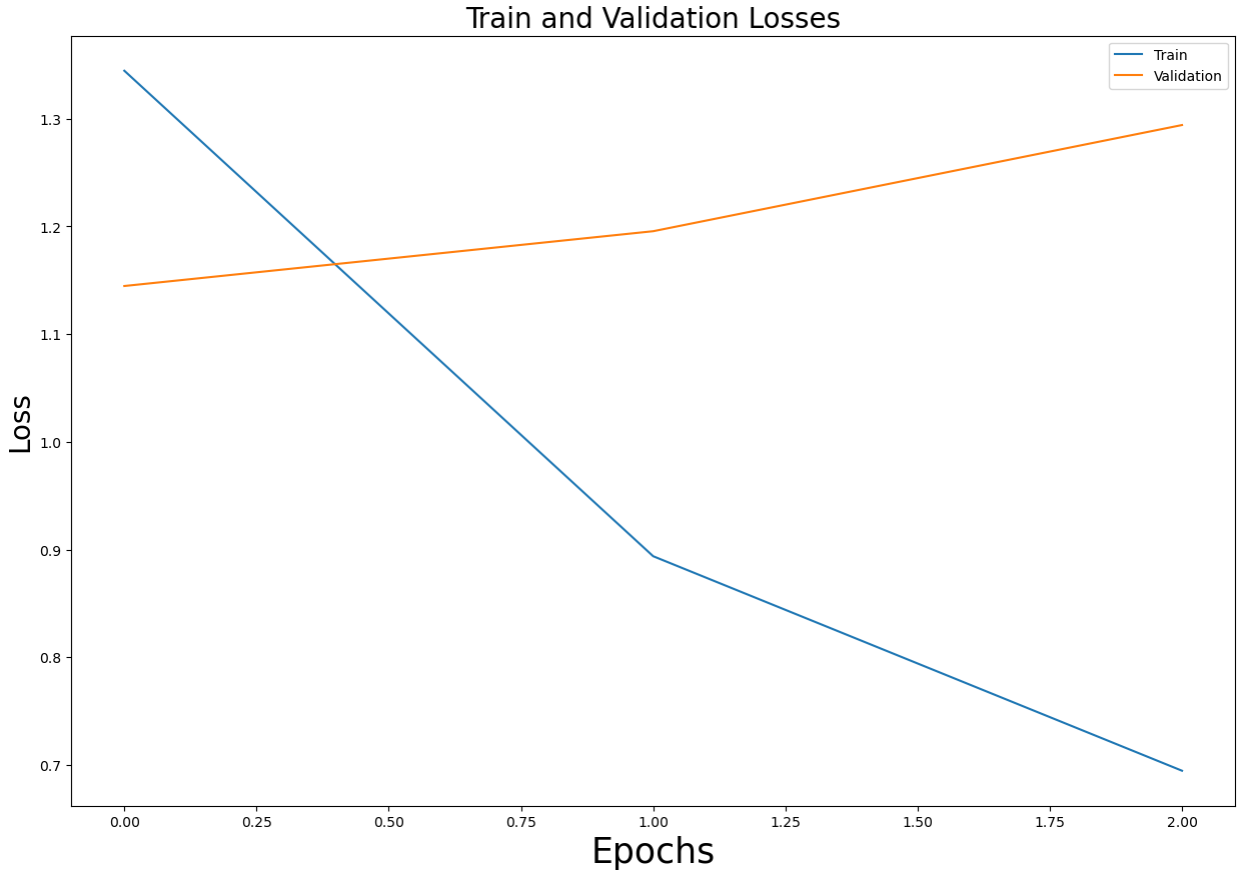
- Data is stored into separate lists (contexts, questions, answers) from the train and validation datasets of SQuAD 1.0 or SQuAD 2.0.
- Since BERT needs both start and end position characters of the answer, those positions were found and stored for later use. Sometimes, it was also noticed that SQuAD answers *eat up* 1 or 2 characters from the real answer in the passage. So

in these cases we handled this problem by *cutting* the passage by 1 or 2 characters to be the same as the given answer. This strategy works because BERT works with **tokens** of a specific format so we needed to process the SQuAD dataset to keep up with the input that BERT expects.

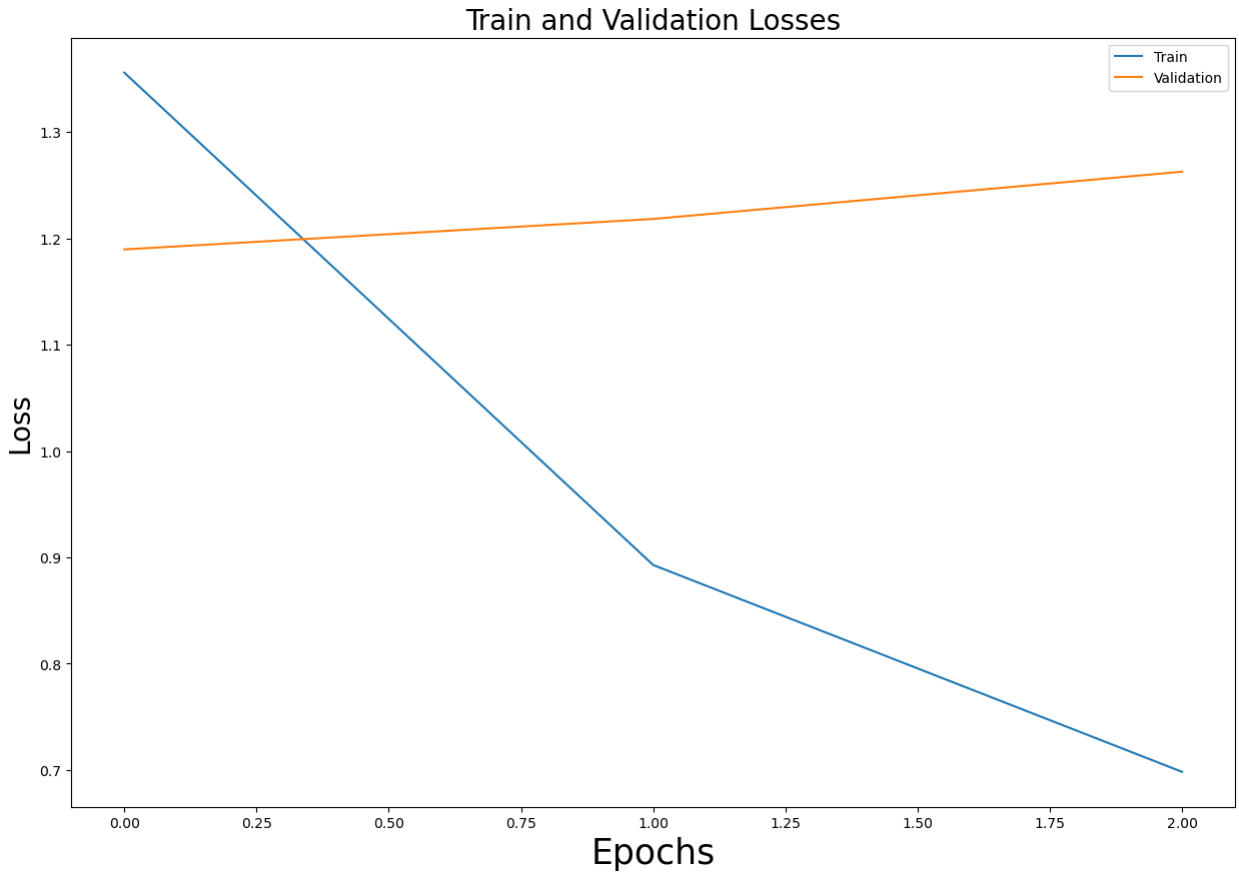
- Tokenization of passages and queries is done using pre-trained tokenizers from **bert-base-uncased**.
- Creation of a SQuAD dataset class (that inherits from torch.utils.data.Dataset), that helps in training and validating previous data more easily and converts encodings to datasets.
- **BertForQuestionAnswering** was selected from the **transformers** library, as it suits the current task the most. When a model is instantiated in PyTorch using *from_pretrained()*, the model configuration and pre-trained weights of the specified model are used to initialize the model.
- For training purposes, the PyTorch optimizer AdamW was used, as it implements gradient bias correction as well as weight decay. 3 epochs were chosen and the learning rate was selected to be 5*10^(-5). A batch size of 8 was chosen.
- In this fine-tuning process, the entire model was trained with the error that was back-propagated through the entire architecture and the pre-trained weights of the model were updated based on the new dataset.

## Observations :-

- The whole training process took 8.51 hours for SQuAD 1.0 and 8.4 hours for SQuAD 2.0.
- Training vs Validation Losses per epoch for SQuAD 2.0 ➖




- Training vs Validation Losses per Epoch for SQuAD 1.0




- The training process was limited to 3 epochs, since it was noticed that a lot of time was getting consumed and there are obvious limitations due to Google Colab and even Kaggle limit of 30 hours of GPU usage per week. Also, it was noticed that overfitting happened from the very first epoch.
- If a bigger batch size was used for training, then the model used to crash on Kaggle as it used to run out of memory.
- Even an addition of a bigger dropout probability didn't change things that much, in fact, the results observed were worse.

## **Testing of fine-tuned BERT model performance on custom data :-**

- The pre-trained, fine-tuned BERT model obtained above was loaded and the tokenizer used was **bert-base-uncased**.
- For simpler custom examples, it was noted that the BERT model trained on SQuAD 1.0 performs better than the one trained on SQuAD 2.0, wrt Exact Match (henceforth referred to as EM) and F1 scores.
- When an answer is required that requires more than 1 entity as the answer, separated by commas, it was noted that both BERT 1.0 (the model trained on SQuAD 1.0) and BERT 2.0 (the model trained on SQuAD 2.0) gave equally good answers.
- But when asked about *the type of music band* in 1 of the questions, BERT 2.0 interestingly gave the answer **british rock** (even when the origin of the band was nowhere asked in the question), whereas the answer given by BERT 1.0 was not that proper.
- When questions were asked that required arithmetic answers that had the same words as the context, BERT 2.0 performed very well, whereas BERT 1.0 gave weird answers.
- For questions that didn't have any answers from the corresponding contexts, both BERT 1.0 and BERT 2.0 gave weird replies, whereas the expectation was that blank replies should have been given in that case.
- In some cases, even when the answer to a question isn't explicitly mentioned in the context, BERT 1.0 gives the correct answer, whereas BERT 2.0 fails to do so.

## Observations wrt Validation Data of SQuAD 1.0 and SQuAD 2.0 and comparison of our fine-tuned BERT with more robust models :-

- For seeing how well our fine-tuned BERT models performed on the unseen validation datasets of SQuAD 1.0 and SQuAD 2.0, we used 2 robust, well-trained models - **bert-large-uncased-whole-word-masking-finetuned-squad** and **distilbert-base-uncased-distilled-squad**.
- The model **bert-large-uncased-whole-word-masking-finetuned-squad** was trained on wikipedia and bookcorpus and contains an estimated 336M parameters.
- Interestingly, the model **distilbert-base-uncased-distilled-squad** was trained on SQuAD itself.
- The following table shows the relative performances of the models on the SQuAD 1.0 validation dataset ➖

| | **Average F1 Score** | **Average HuggingFace pipeline score** |
|---|---|---|
| **BERT 1** | 0.2886 | 0.2444 |
| **bert-large-uncased-whole-word-masking-finetuned-squad** | 0.79 | 0.6948 |
| **distilbert-base-uncased-distilled-squad** | 0.7302 | 0.6607 |

- The following table shows the relative performances of the models on the SQuAD 2.0 validation dataset ➖

| | **Average F1 Score** | **Average HuggingFace Pipeline Score** |
|---|---|---|
| **BERT 2** | 0.3391 | 0.3013 |
| **bert-large-uncased-whole-word-masking-finetuned-squad** | 0.7704 | 0.6715 |
| **distilbert-base-uncased-distilled-squad** | 0.7153 | 0.6423 |

- It can be clearly seen that BERT 1 and BERT 2 still lag behind the larger, well-trained models. One reason is that we don't have enough computational resources to run multiple experiments in parallel. Also, BERT 1 and BERT 2 have been fine-tuned on only SQuAD 1.0 or SQuAD 2.0 respectively.

## Advantages and Disadvantages of using BERT or any transformer-based models :-

- The advantage is that the answers that exist in the context are always captured. For the case of *no answer*, which means when the answer doesn't exist in the context, the system will predict *no answer* or predict an answer that has a low

probability so that it can be handled appropriately and eliminated.

- The disadvantage of this system is that, in reality, not all documents and answers are similar to a question in such a way, so domain-specific lexical knowledge is still also necessary for many questions, and the SQuAD dataset also doesn’t cover all the cases.