INTER IIT TECH MEET
Final Evaluation
DEVREV
Team ID - 57

Abstract:

The importance of retrieving information is paramount and has become a critical need in today's modern world. Thus, an accurate question-answering system is very important. This report highlights our various attempts toward the problem statement and how well the model performs on the tasks given based on the metrics used for the evaluation. Pre-trained Language Models are predominantly used for complex Natural Language Processing tasks (like Summarization, Question Answering, Image Captioning, and Sentence Similarity) and have been proven effective.

Section (I)

Methods Evaluated

Sub-Task 1

I. Attempted but Failed Methods

1) Our initial model consisted of using a *BERT_en_uncased* model to generate the embeddings for our questions and contexts and passing them through a simple feed-forward neural network to get a prediction on the Yes/No question. The drawback of this method was that it needed to provide us with the top-k relevant contexts.
2) For returning the top-k passages, we tried training the *Deep Passage Retriever* model by Facebook AI Research on our data. This approach could not be completed as the Google Colab free tier crashed before we could finish testing.
3) We also tried to reduce the overhead of the above method by using other encoders (*Bert_en_uncased* but with 512-word limit and *Facebook QA encoder*), but that too resulted in failure since we could not test our model due to the Google Colab free tier crashing.

II. Speculated Method

A method that we speculated is as follows: We would use the model that we are using for the second subpart and predict answers from the given corpus for all the questions on the given dataset, irrespective of whether they can be answered or not. In case they are non-answerable, we get a random text output. Finally, we could have trained a deep neural network on the similarities of these questions and predicted answers and the target they corresponded to (0/1). This way, we could have estimated a threshold of similarity for the questions and found answers to declare that the context is enough to answer the same.

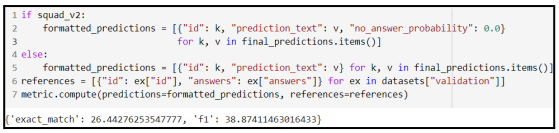
One of the drawbacks of this method is that it has a huge overhead, and thus, it would have led us to overshoot our resource constraints. Also, its implementation is a bit tricky as we could not converge upon the measure of similarity to use.

Sub-Task 2

I. Attempted but Failed Methods

1) Our starting approach was to embed and vectorise the questions and the context with the help of *Mass Language Models*, which resulted in poor performance metrics because of the sheer size of the data and low scalability.
2) Our second approach aimed at implementing *Knowledge Graph Question Answering (KGQA)* models because it was unable to retain the context of the text in lower layers.
3) Long-term dependencies are not well captured by LSTM. The transformers in BERT perform better than any bi-directional LSTM, which captures both forward and backward context. In BIDAF, we only use attention because that is how much attention the other words in the sequence require. We use self-attention-based transformers in BERT. While encoding a particular word, the self-attention layer aids the encoder in looking at other words in the input sentence.
4) Once we were stuck with transformer models, especially BERT, we tried implementing an ensemble of $BERT_{Base}$ & $BERT_{Large}$ models. As the primary objective of the problem is to optimise the extraction of the possible answer, we concluded that an ensemble model would not be suited for this approach. Moreover, being an aggregation of large models, we noticed an explosion of spatial and temporal complexities.

5) To reduce these complexities of the model, we tried implementing the most shrunk-down versions of BERT, like $BERT_{Tiny}$, and $BERT_{Mini}$, but their baseline metrics are significantly subpar compared to their $BERT_{Base}$ counterpart.



```
if squad_v2:
    formatted_predictions = [{"id": k, "prediction_text": v, "no_answer_probability": 0.0}
                             for k, v in final_predictions.items()]
else:
    formatted_predictions = [{"id": k, "prediction_text": v} for k, v in final_predictions.items()]
references = [{"id": ex["id"], "answers": ex["answers"]} for ex in datasets["validation"]]
metric.compute(predictions=formatted_predictions, references=references)
```

```
{'exact_match': 26.442762535477777, 'f1': 38.87411463016433}
```

Fig 1 - *Low accuracy of $BERT_{Tiny}$ Model*

Section (II)

Implementation

Paragraph Retrieval Task

I. Synthetic Data Generation

We are taking 40,000 points from the original *SQUAD V2* dataset and all available points from the data given and capping it to 10,000 data points by augmenting and generating synthetic data.

II. Training Methodology

We divide this task into two subtasks. The first task is finding the top-k paragraphs that can answer the given question from the given set of passages. The second set is predicting the paragraphs that can answer our question among this top-k and passing it on to the next subpart of the project.

For the first subpart, we are using the methodology used in the *Deep Passage Retrieval* methods, where we generate Dense Vectors of the given context and the question and find the k most probable ones using the dot-product metric.

In the second subpart, we pass these paragraphs with the question into an *SBert Encoder,* which generates the embeddings corresponding to these two. Finally, these embeddings are passed into a theme fine-tuned feed-forward neural network which works to detect patterns among them corresponding to yes/no.

We have trained the model on the 40000 random SQUAD V2 data points along with 10000 theme-based data (original + synthetic) for each theme.

III. Fine Tuning:

Since this model is not as complex as BERT, finetuning does not make viable sense. So, we introduced training data that is biased towards the themes by adding 10,000 data points (both synthetic and given) on top of 40,000 generic random data points from the original SQUAD V2 dataset.

IV. Final Implemented Pipeline

Final Pipeline is as follows:

Data (Question+Context) → Top-k Passage Predictor (using Facebook QA encoders and the cross-product metric) → k passages and the question → DL Model using SBert Embeddings → Final corpus that probably contains the answer ready to be passed to the next stage for answer prediction.

Question-Answering Task

I. Synthetic Data Generation:

This step was necessary for fine-tuning as the number of data points was less. Going by existing literature and conforming experimentally, we reached the conclusion that synthetic data generation is necessary for good results.

Our implementation involves two stages of data generation. They are as follows:

Stage 1 - Albumentation by Shuffling of Sentences. Assuming the answer text spans from positions *[i…j]* in the context - let *i'* be the index of the sentence delimiter just before *i* and *j'* be the index of the sentence delimiter just after *j* - so now the context can be divided into 3 segments → *[0..i'-1], [i'...j']* and *[j'+1..n]*.

For the first and the last segments, the sentences are randomly shuffled among them and within them, keeping *[i..j]* fixed.

Stage 2 - Using NLP Aug. The technique provided by NLP Aug, which suited the best for our problem, is represented by the array augmenter in the implementation, which includes the following:

1) Substitute word by word2vec similarity;
2) Substitute word by WordNet's synonym;
3) Substitute word by antonym;
4) Deletion of random words; and
5) Substitute words by TF-IDF similarity.

A subset of operations is chosen randomly for each piece of text given so as to maintain the uniformity of the sample but not allow one operation to dominate over others. **Back Translation Augmenter* and *Contextual Word Embeddings Augmenter* could have provided

better results, but the provided constraints of the problem statement prevented us from doing so.

II. Training Methodology

Since the number of trainable parameters in the BERT Base model is very high, we used transfer learning by adjusting the weights of the last few layers of the model. HuggingFace provides a direct functionality to execute the training process.

III. Fine Tuning Methodology

The down streaming process comprises three stages which are as follows:

1) **Fine Tuning the Distilbert Uncased Pre-Trained Model on Squad dataset.** This step provides a baseline for the model as the BERT architecture specifies its implementation in that particular way.
2) **Clustering on the Basis of Themes.** The squad dataset comprises approximately 650 themes, which span various disciplines. So clustering seemed an important optimisation metric to correlate the common themes and reduce the dimensionality of the data to 21 clusters, which was verified by the elbow method.

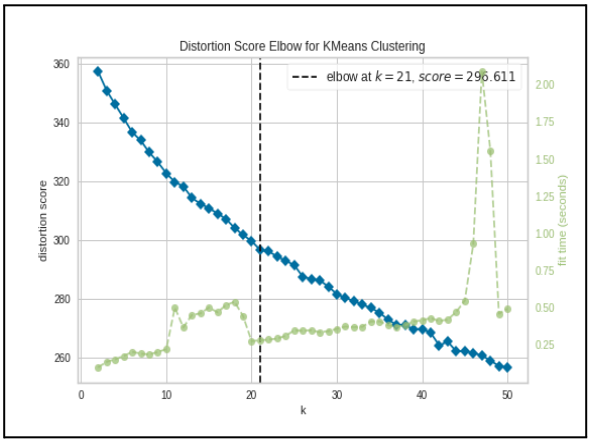


Fig 2 - Graph on K Means clustering

The cluster formation was done by applying KMeans++ algorithms on the embeddings of the themes given by Sentence BERT (SBert).

3) **Fine Tuning Based on a Single Theme Augmented to a Cap.** For a particular theme, we find the nearest cluster (by centroid distance), and the chosen cluster is further fine-tuned on the given data aided with the synthetic data to a cap as we go down the pipeline.

Theme	DataPoints	BEST - clustered : Augmented Synthetic - 300 points 1 epoch (Everything included) - Original model		Base model(only on squad_v1) - 0 epochs	
		Exact	F1	Exact	F1
The_Times	43	81.39534884	89.92248062	72.09302326	83.56637296
Iranian_languages	13	69.23076923	82.56410256	61.53846154	89.67032967
Buddhism	122	81.96721311	88.02248107	78.68852459	87.22354389
San_Diego	47	80.85106383	90.81266251	63.82978723	79.25680309
ASCII	22	63.63636364	84.50781103	54.54545455	81.30268424
Chinese_characters	34	67.64705882	83.30125757	67.64705882	78.24929972
Mandolin	42	78.57142857	87.25672878	57.14285714	82.72055441
Canadian_football	19	78.94736842	78.94736842	47.36842105	60.39872408

Fig 3 - *Comparison between the baseline model and the best-fine-tuned model.*

V. Using Already Answered Questions

We implemented hashmaps to map the context questions with indices of the unique questions. This reduces the processing time significantly by utilizing the inherent feature of hashmap to fetch the answer directly had it been previously stored.

VI. Final Implemented Pipeline

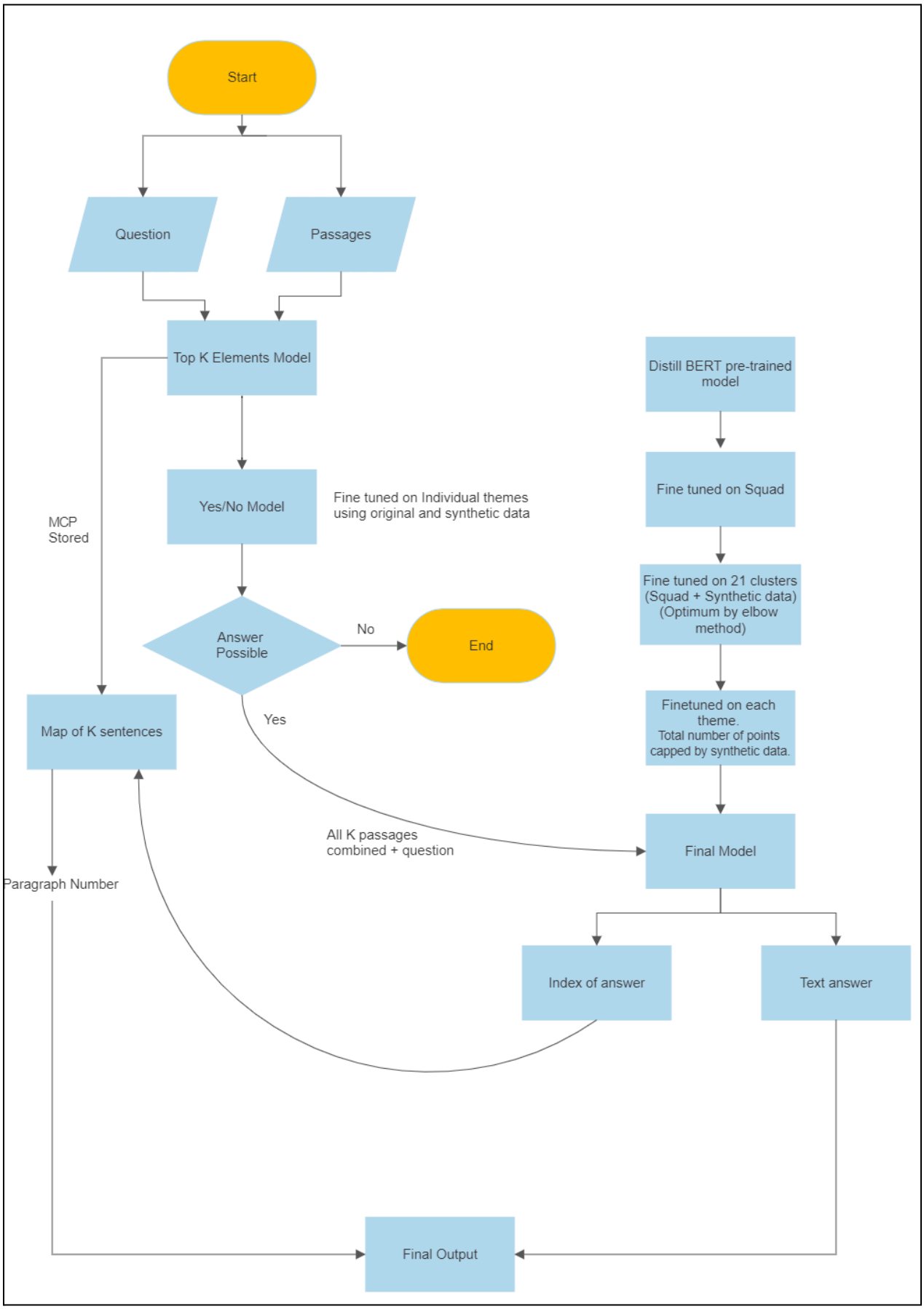


Fig 4 - *Final Implemented Pipeline.*

Implementation Steps:

1) **Requirements:**

Huggingface:

- Retrieving the pre-trained distill BERT model and SQUAD dataset.'
- Storing of the fine-tuned models on Hugging face Hub

2) Fine Tuning of the pre-trained distill BERT model on SQUAD

3) Creation of subsidiary files
   - For clustering of 650 themes present in SQUAD dataset. This notebook is used: https://colab.research.google.com/drive/1s05akuHsL6O1vINrT2wRiKV-nwl3J9la
     This notebook gave the file cluster.json, which stores the details of clusters and their centroids.
   - For each cluster, finetuning the pre-trained models and saving on the Huggingface hub. Refer to the following notebook: https://colab.research.google.com/drive/1T1EGfjACCokKaWV_MQ5iZDy5tIDKo7t5

   - Preprocessing the data given on the evaluation day to the standard squad format. Please refer to this notebook: https://colab.research.google.com/drive/1v7ZFLfV0n5lGO8ktUOuFEbJQ9L7_zVIK

     Final Train data: https://drive.google.com/file/d/1TX35oLZEx_VgSa6I-Oe1-EhnuDQ5JB3G/view?usp=sharing

   - Fine Tuning based on themes, the boolean model, which answers whether the question is answerable, and manually storing checkpoints. Refer to this notebook. https://colab.research.google.com/drive/1hHdg2p53LbRmjtoGPsh3_yR4ZRBwSt0u#scrollTo=H5T3jufe0cWx

     Zip of yes/no model checkpoint trained only on squad_v2: https://drive.google.com/file/d/1eaNR2UxOC6NfJRV85hsJqrRSJYWK1Wqi/view?usp=share_link

     Zip of yes/no model checkpoints trained on the squad and fine-tuned theme-wise: https://drive.google.com/file/d/1LefY2GNWX9IhDKCbFkV7iH03OY379KX3/view?usp=share_link

   - Fine Tuning the already cluster-wise fine-tuned model on the basis of themes given on training data. Refer to this notebook: https://colab.research.google.com/drive/14dN-A1pTIQYOueSY1BewQKROqT2nf6Zs

4) Final testing
   Refer to this model: https://colab.research.google.com/drive/1K_qlHDMZUiRYyQnHznGmQz-kae_qsv1k#scrollTo=nEacwezAAOCF

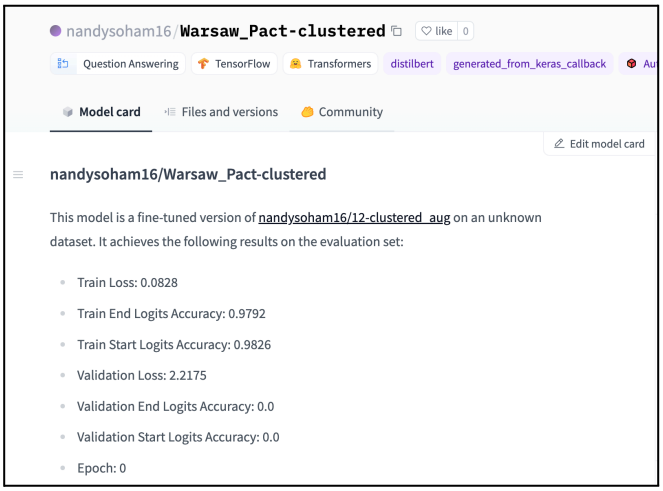


Fig 4.1 : Snap of a random theme wise fine-tuned model on Huggingface which has been derived from its nearest finetuned cluster - in this case, cluster 12

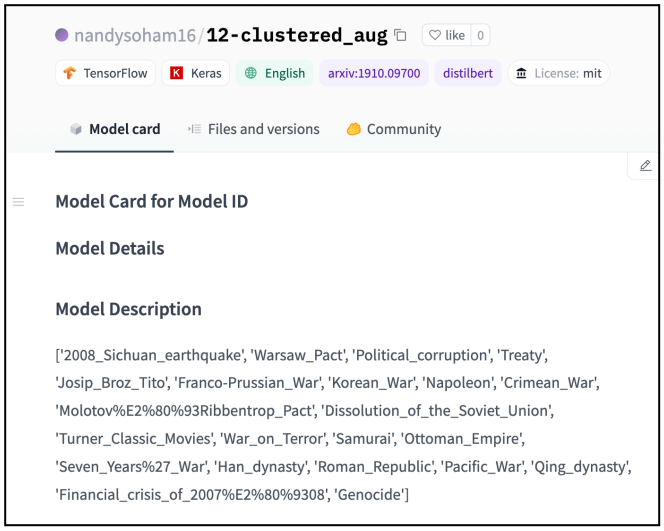


Fig 4.2: Snap of cluster 12 - The card shows themes similar to Warsaw Pact - namely Wars, Treaties and dynasties being clubbed under 1 cluster

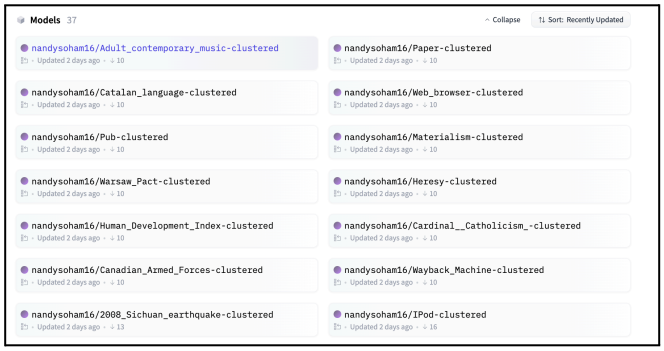


Fig 4.3: Snap of the 14 fine-tuned models based on the themes given during the evaluation

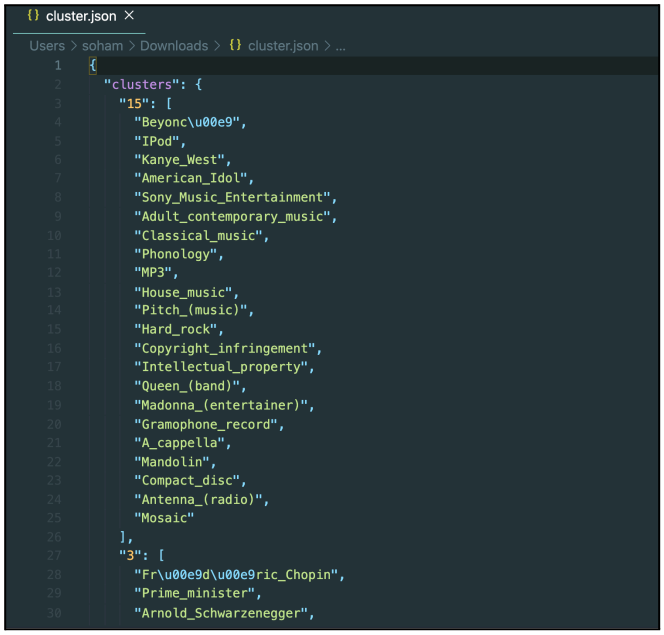


Fig 4.4 : Snap from Cluster.json showing a random cluster

Runtime Analysis:

Testing:
Fine Tuning the yes/No model - **~6 hrs**

Fine Tuning the nearest cluster to the theme by capping the data points to 300 (both original + synthetic) -(for 14 themes given in the question set) **~3 hrs**

Testing:
Final testing time - 9423.687 s => **~ 2.62 hrs**

Section *(III)*
Literature Review

The Search for feasible approaches began with Knowledge Graph Question Answering (KHQA) systems.[1][2][3] The approach of KGQA worked pretty well for standard question answering, but at its core, it undermined the potential of the current NLP approaches, which include first pretraining a model and then fine-tuning it. This brought us to Masked Language Models [4] like BERT, ALBERT, RoBERTa, XLNet, ConvBERT, BART, etc. [5]

RNN-Based Encoder-Decoder Models:
Introduced for the first time in 2014 by Google, a sequence-to-sequence model aims to map a fixed-length input with a fixed-length output where the length of the input and output may differ.

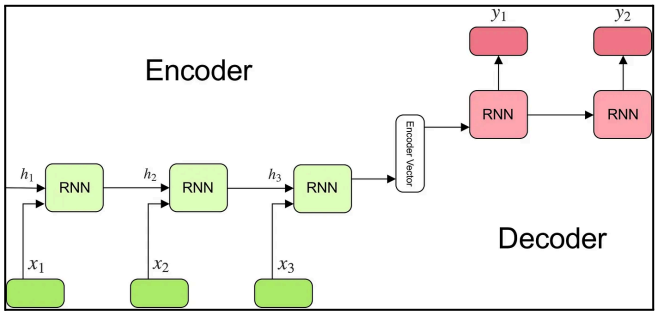


Fig 5 - RNN based Encoder-Decoder Models

Encoder
A collection of multiple recurrent units (LSTM or GRU cells for better performance), each of which accepts a single input sequence element and propagates information for that element forward.
In a problem involving question-answering, the input sequence consists of the real question's words. Each word is denoted by the symbol x_i, where i denotes the word's order.

The following formula is used to calculate the hidden states h_i:

$$h_t = f(W^{(hh)}h_{t-1} + W^{(hx)}x_t)$$

Decoder

A collection of multiple recurrent units, each of which forecasts an output at time step t, y_t.

Each recurrent unit receives a hidden state from the prior unit and generates both an output and a hidden state of its own.

The output sequence in the question-answering problem is a compilation of each word in the response. Each word is denoted by the symbol y_i, where i denotes the word's order.

Any hidden state h_i is computed using the formula:

$$h_t = f(W^{(hh)}h_{t-1})$$

The output y_t at time step t is computed using the formula:

$$y_t = softmax(W^S h_t)$$

The hidden state at the current time step and the appropriate weight W(S) is used to calculate the outputs. Softmax is used to create a probability vector that will enable us to predict the outcome (for example, the word in the problem requiring question-answering).

The power of this model lies in the fact that it can map sequences of different lengths to each other. As we can see, the inputs and outputs do not always have the same length. As a result, an entirely new set of issues can now be resolved using such an architecture.

Deep Attention in Sequence Models:

Attention is a mechanism that was developed to improve the performance of the Encoder-Decoder RNN, mostly for on-machine translation but works for various context-based problem scenarios.

Attention is proposed as a solution to the limitation of the Encoder-Decoder model's encoding of the input sequence to one fixed-length vector from which to decode each output time step. This issue is believed to be more of a problem when decoding long sequences. Instead of encoding the input sequence into a single fixed context vector, the attention model develops a context vector that is filtered specifically for each output time step.

As with the Encoder-Decoder paper, the technique is applied to a machine translation problem and uses GRU units rather than LSTM memory cells. In this case, a bidirectional input is used where the input sequences are provided both forward and backwards, which are then concatenated before being passed on to the decoder.

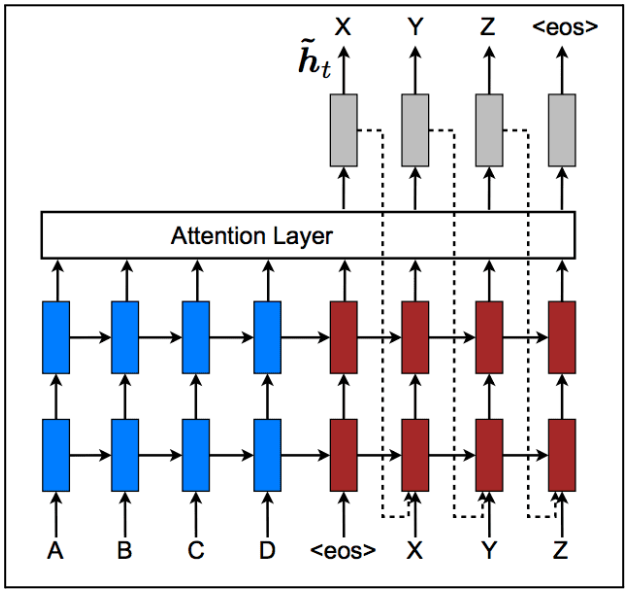


Fig 6 - *Attention layer in sequence model*

Transformers:

Like most neural networks, transformer models are basically large encoder/decoder blocks that process data. In addition, the math that transformers use lends itself to parallel processing, so these models can run fast.

Transformers use positional encoders to tag data elements coming in and out of the network. Attention units follow these tags, calculating a kind of algebraic map of how each element relates to the others. Attention queries are typically executed in parallel by calculating a matrix of equations in what's called "multi-headed attention."

Transformers using Attention:

Transformers are our best current solution to contextualisation. The type of attention used in them is called self-attention. This mechanism relates different positions of a single sequence to compute a representation of the same sequence. It is instrumental in machine reading, abstractive summarisation, and even image description generation.

Scaled dot-product attention, the main component, is very elegant in that it accomplishes a lot with only a

few linear algebra operations. There are three matrices that make up the system: Q, K, and V, also known as query, key, and value matrices. Each of these matrices has a dimension of d_k. Given that the output will only contain the values of v where alpha is 1, the vector alpha is in this instance, the de facto query.

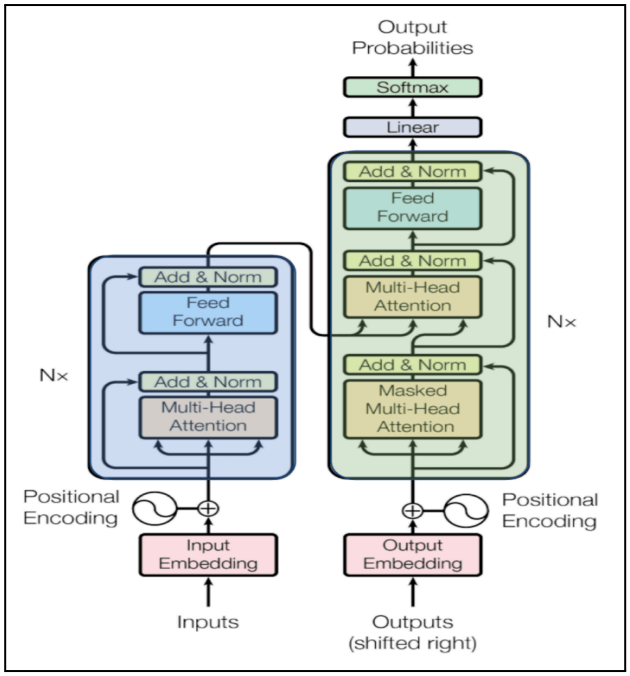
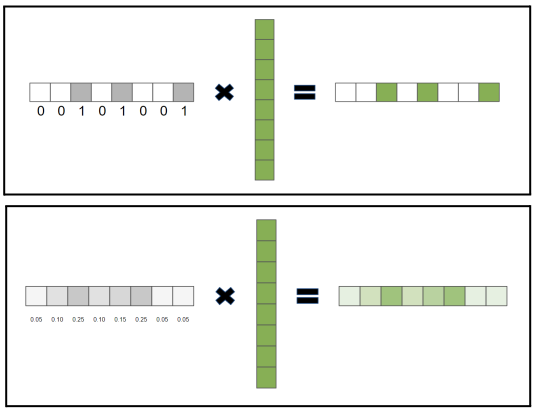


Fig 7 - Flow chart of the transformer using attention



$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

$$a_i = \frac{exp(S_i)}{\sum_j \exp(S_j)}$$

$$attention\ value = \sum_i a_i V_i$$

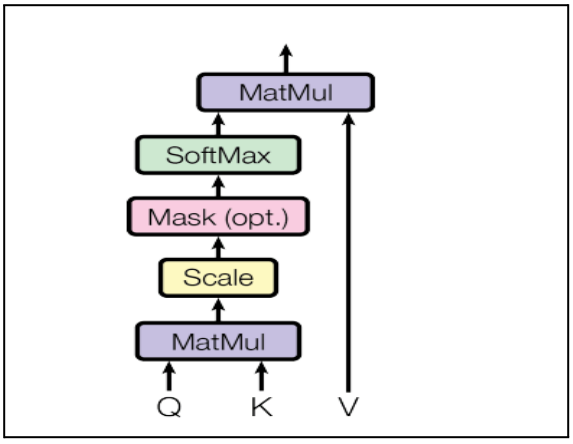


Fig 8 - Scalar dot product attention

It is possible to have several different key, query, and value matrix sets. As a result, each attention module can concentrate on determining particular contextualized embeddings and different kinds of relationships between the inputs. The output of the so-called Multi-Headed Attention Module can then be created by concatenating these embeddings and passing them through a standard linear neural network layer, as shown in the diagram above. It turns out that this method not only enhances model performance but also training stability.

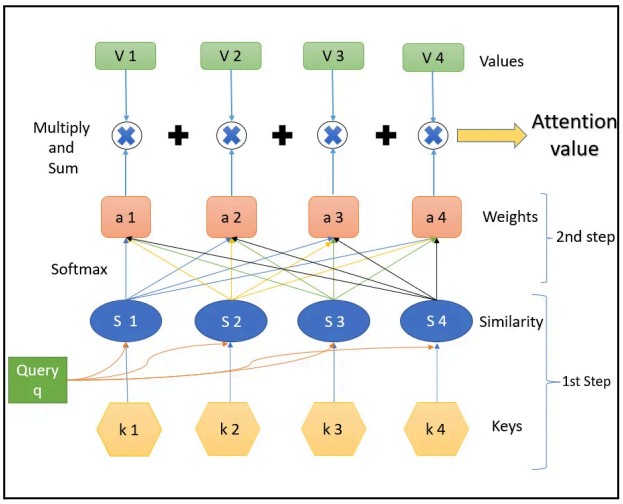


Fig 9 - Multi headed attention architecture

Similarity Calculation

$$
\begin{aligned}
S_i = f(q,k_i) &= q^T k_i \ldots\ldots\ldots\ldots \textit{dot product} \\
&\quad (T \textit{ is transpose}) \\
&= q^T k_i/\sqrt{d} \ldots\ldots\ldots \textit{scaled dot product} \\
&\quad (d \textit{ is dimensionality of each key}) \\
&= q^T W k_i \ldots\ldots\ldots \textit{general dot product} \\
&\quad (\textit{query projected into a new space using } W, \\
&\quad\quad W \textit{ is weight matrix}) \\
&= \text{Kernel Methods (Mapping the two vectors q and} \\
&\quad\quad \text{k in a new space using a non} - \text{linear function})
\end{aligned}
$$

$$
\begin{aligned}
&\text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) \cdot V \\
&= \begin{bmatrix} \text{softmax}\left(\frac{e_{11}}{\sqrt{d_k}} & \frac{e_{12}}{\sqrt{d_k}} & \ldots & \frac{e_{1n}}{\sqrt{d_k}}\right) \\ \text{softmax}\left(\frac{e_{21}}{\sqrt{d_k}} & \frac{e_{22}}{\sqrt{d_k}} & \ldots & \frac{e_{2n}}{\sqrt{d_k}}\right) \\ \vdots & \vdots & \ddots & \vdots \\ \text{softmax}\left(\frac{e_{m1}}{\sqrt{d_k}} & \frac{e_{m2}}{\sqrt{d_k}} & \ldots & \frac{e_{mn}}{\sqrt{d_k}}\right) \end{bmatrix} \cdot \begin{bmatrix} v_{11} & v_{12} & \ldots & v_{1d_v} \\ v_{21} & v_{22} & \ldots & v_{2d_v} \\ \vdots & \vdots & \ddots & \vdots \\ v_{n1} & v_{n2} & \ldots & v_{nd_v} \end{bmatrix}
\end{aligned}
$$

The attention value, which is a weighted sum/linear combination of the Values V, is obtained with the aid of the query q and the keys k. The weights are derived from some similarity between the query and the keys.

BiDirectional Attention Flow (BiDAF):

This is an extractive, closed-domain Q&A model that can only respond to factoids. These traits suggest that BiDAF needs Context to respond to a Query. BiDAF always returns an Answer that is a substring of the supplied Context. The BiDAF architecture is as follows:

1. Embedding Layers: BiDAF has 3 embedding layers whose function is to change the representation of words in the Query and the Context from strings into vectors of numbers.
2. Attention and Modeling Layers: These Query and Context representations then enter the attention and modelling layers. These layers use several matrix operations to fuse the information contained in the Query and the Context. The output of these steps is another representation of the Context that contains information from the Query. The paper refers to this output as the “Query-aware Context representation.”

3. Output Layer: The Query-aware Context representation is then passed into the output layer, which will transform it into a bunch of probability values. These probability values will be used to determine where the Answer starts and ends.

BERT Model (Bidirectional Encoder Representations from Transformers):

The main technological development of BERT is the translation of the popular attention model Transformer's bidirectional training to language modeling. However, earlier studies either examined text sequences from a left-to-right training perspective or from a combined left-to-right and right-to-left training perspective. The results of the study show that bidirectionally trained language models are better able to understand context and language flow than single-direction language models. In their paper, the authors present a novel technique known as Masked LM (MLM), which enables bidirectional training in models where it was previously not feasible.

How BERT works

BERT makes use of a Transformer, an attention mechanism that learns contextual relations between words (or sub-words) in a text. Transformer's basic design consists of two independent mechanisms: an encoder that reads the text input and a decoder that generates a task prediction.

The Transformer encoder reads the entire sequence of words at once, in contrast to directional models, which read the text input sequentially (from right to left or left to right). Although it would be more accurate to describe it as non-directional, it is therefore thought of as bidirectional. Due to this feature, the model can determine the context of a word based on both its left and right surroundings.

Masked LM (MLM)

Word sequences are changed with a [MASK] token for 15% of the words in each sequence before being fed into the BERT. Based on the context offered by the other non-masked words in the sequence, the model then makes an attempt to predict the original value of the masked words. Technically speaking, the output words' prediction calls for:

Adding a classification layer on top of the encoder output. Multiplying the output vectors by the embedding matrix, we transform them into the vocabulary dimension. Calculating the probability of each word in the vocabulary with softmax.

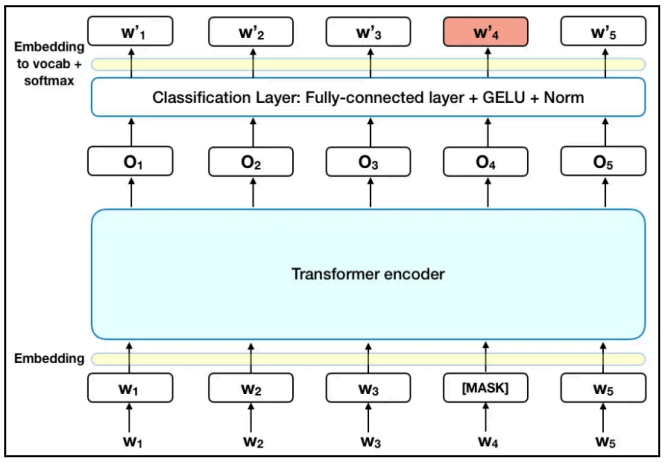


Fig 10 - Flow chart of Masked LM

Next Sentence Prediction (NSP):

In the BERT training process, the model learns to predict whether the second sentence in a pair will come after another in the original document by receiving pairs of sentences as input. During training, 50% of the inputs are paired in which the second sentence is the next one in the original document, and in the remaining 50%, the second sentence is a randomly selected sentence from the corpus. The underlying presumption is that the second sentence will not be connected to the first. Before entering the model, the input is processed as follows to aid the model in differentiating between the two sentences during training: The first sentence has a [CLS] token at the start, and each subsequent sentence has a [SEP] token at the end.

Each token has a sentence embedding that designates Sentence A or Sentence B. Token embeddings with a vocabulary of 2 and sentence embeddings share a similar concept. Each token receives a positional embedding to indicate its place in the sequence. The Transformer paper presents the theory and practice of positional embedding.

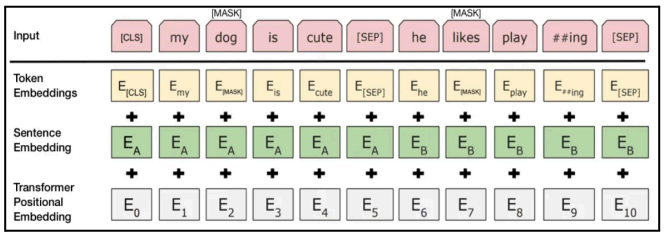


Fig 11 - Next sentence embedding (NSP)

To predict if the second sentence is indeed connected to the first, the following steps are performed: The entire input sequence goes through the Transformer model.

The output of the [CLS] token is transformed into a 2×1 shaped vector, using a simple classification layer (learned matrices of weights and biases). Utilizing softmax to determine the IsNextSequence probability.When training the BERT model, Masked LM and Next Sentence Prediction are trained together to minimize the combined loss function of the two strategies. BERT only adds a thin layer to the basic model but can be applied to a wide range of language tasks: Similar to Next Sentence classification, sentiment analysis is classified by adding a classification layer on top of the Transformer output for the [CLS] token.

In Question Answering tasks (like SQuAD v1.1), the program must mark the correct response in the text sequence after receiving a question about it. A Q&A model can be trained using BERT by learning two additional vectors that indicate the start and end of the response.

BERT model experimental inferences:

Even at very large scales, model size matters. The largest model of its kind is called BERT_large, and it has 345 million parameters. It clearly outperforms BERT_base, which uses the same architecture but "only" has 110 million parameters on small-scale tasks. More training steps equate to greater accuracy when there is enough training data. For instance, on the MNLI task, training on 1M steps (128,000 words batch size) results in a 1.0% increase in BERT_base accuracy compared to 500K steps with the same batch size.Because only 15% of words are predicted in each batch, BERT's bidirectional approach (MLM) converges more slowly than left-to-right approaches, but after a few pre-training steps, bidirectional training still outperforms left-to-right training.

References:

1) *Hamilton, Bajaj, Zitnik, Jurafsky, Leskovec, Embedding Logical Queries on Knowledge Graphs*
2) *Ren, Hu, Leskovec , QUERY2BOX: REASONING OVER KNOWLEDGE GRAPHS IN VECTOR SPACE USING BOX EMBEDDINGS*

3) *Diomedi, Hogan Question Answering over Knowledge Graphs with Neural Machine Translation and Entity Linking*
4) *Koustuv Sinha, Robin Jia, Dieuwke Hupkes, Joelle Pineau, Adina Williams, Douwe Kiela Masked Language Modeling and the Distributional Hypothesis: Order Word Matters Pre-training for Little*
5) Kate Pearce , Tiffany Zhan , Aneesh Komanduri , Justin Zhan *A Comparative Study of Transformer-Based Language Models on Extractive Question Answering*
6) Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*
7) *Open Sourcing BERT: State-of-the-Art Pre-training for Natural Language Processing* https://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html
8) *Karpukhin, V., Oğuz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D. and Yih, W.T., 2020. Dense passage retrieval for open-domain question answering. arXiv preprint arXiv:2004.04906.* https://arxiv.org/pdf/2004.04906.pdf
9) *Freeman, T. and Zuo, A., Answerability Verification in SQuAD.* https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1204/reports/default/report12.pdf
10) *Ji, Y., Chen, L., Dou, C., Ma, B. and Li, X., 2022. To answer or not to answer? Improving machine reading comprehension model with span-based contrastive learning. arXiv preprint arXiv:2208.01299.* https://arxiv.org/pdf/2208.01299.pdf
11) *Deng, Andrew, and Wenli Looi. "Compressed SQuAD 2.0 Model With BERT."* https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1204/reports/default/report17.pdf
12) *Distilling BERT Using an Unlabeled question-Answering Dataset* Blog *by Towardsdatascience*
13) *Nikka Mofid , Emanuel Pinilla , Elijah Freeman Tackling SQuAD 2.0 with Ensemble Methods and Data Augmentation* https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1204/reports/default/report13.pdf
14) *Data Augmentation in NLP: Best Practices From a Kaggle Master* Blog *by neptune.ai*
15) *Risch, J., Möller, T., Gutsch, J. and Pietsch, M., 2021. Semantic answer similarity for evaluating question-answering models. arXiv preprint arXiv:2108.06130.* https://arxiv.org/pdf/2108.06130.pdf
16) *Semantic Answer Similarity: The Smarter Metric to Score Question Answering Predictions* Blog *by .deepset.ai*
17) *BERT For Measuring Text Similarity* Blog *by Towardsdatascience*
18) *Aithal, S.G., Rao, A.B. and Singh, S., 2021. Automatic question-answer pairs generation and question similarity mechanism in the question-answering system. Applied Intelligence, pp.1-14.* https://link.springer.com/article/10.1007/s10489-021-02348-9
19) *Seo, M., Kembhavi, A., Farhadi, A. and Hajishirzi, H., 2016. Bidirectional attention flow for machine comprehension. arXiv preprint arXiv:1611.01603.* https://arxiv.org/pdf/1611.01603.pdf.
20) *BERT Explained: State of the art language model for NLP* , Blog *by Towardsdatascience*
21) *Question Answering with a fine-tuned BERT… using Hugging Face Transformers and PyTorch on CoQA dataset by Stanford,* Blog *by Towardsdatascience*
22) *Which flavor of BERT should you use for your QA task?* Blog *by KDnuggets*
23) *Transformers,* Blog *by Medium*
24) *Official Docs of HuggingFace*, https://huggingface.co/docs/transformers/index
25) *Google BERT GitHub Repository*, https://github.com/google-research/bert
26) *Huang, K., Tang, Y., Huang, J., He, X. and Zhou, B., 2019, November. Relation module for non-answerable predictions on reading comprehension. In Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL) (pp. 747-756).* https://aclanthology.org/K19-1070.pdf
27) *An Illustrated Guide to Bi-Directional Attention Flow (BiDAF)* Blog