ಭಾರತೀಯ ಮಾಹಿತಿ ತಂತ್ರಜ್ಞಾನ ಸಂಸ್ಥೆ ರಾಯಚೂರು
भारतीय सूचना प्रौद्योगिकी संस्थान रायचूर
Indian Institute of Information Technology Raichur

Surveying NLP Techniques for Question Answering on the SQuAD Dataset

Yugal Garg | Mini Project | 2nd May 2023
Supervisor – Dr. Neha Agarwal

Contents

Abstract

This paper presents survey/analysis of state-of-the-art techniques for question answering on the SQuAD dataset, which is a new reading comprehension dataset consisting of questions posed by crowd workers on a set of Wikipedia articles. The paper starts with an introduction to the SQuAD dataset, the importance of word representation techniques in natural language processing (NLP), and their role in question answering systems. The paper then provides an overview of different word representation techniques, including classical models, such as WordNet, one-hot encoding, and TF-IDF, and distributed representations, such as word2vec. The techniques of BiDAF, BERT were studied in detail, and their key features, strengths, and weaknesses were discussed. This paper presents the methodology used to implement and fine-tune each technique on the SQuAD dataset, along with the evaluation metrics used to compare their performance. The results of the analysis are presented and the performance of each technique is discussed in detail. The paper concludes with a summary of the main findings and their implications for the field of question answering. At the end of the paper, the relevant literature is cited to support the arguments and provide a context for the analysis. The paper also discusses the limitations of the SQuAD dataset and the need for new datasets that adequately prepare question answering models for real-world scenarios.

Introduction

The Stanford Question Answering Dataset (SQuAD) is a special type of dataset used to teach computers how to read and understand written language, much like humans would. It was created by a team at Stanford University and contains a collection of articles and questions related to these articles. The articles were obtained from Wikipedia and covered a wide range of topics. For example, an article might be about the history of ancient Rome or the life of a famous scientist. The questions were designed to test the computer's ability to comprehend and extract information from articles. Some of the questions are simple and require a computer to retrieve a fact from the article, while others are more complex and require a computer to reason and infer based on the information in the article.

The answers to the questions were either short-text snippets taken directly from the article or a special "no answer" token if the computer could not find an answer. This allows the computer to learn how to recognize and extract relevant information from articles. SQuAD has become a popular dataset in the field of natural language processing, and many researchers and engineers have used it to develop and evaluate machine-learning models. These models can be used to power virtual assistants, chatbots, and other applications that require computers to read and understand the human language. In summary, SQuAD is a powerful tool for teaching computers how to read and understand a written language. It contains a diverse set of articles and questions that allow computers to learn how to extract relevant information and answer questions like a human would. The dataset has been instrumental in advancing the field of natural language processing and will continue to play a critical role in developing advanced language models in the future.

Word representation is fundamental and plays a crucial role in the domain of natural language processing (NLP) and question answering (QA) systems. These techniques help computers/machines to understand and process the language, or we can say the textual data by converting the words in the text into vector/numerical representations that can be easily analyzed or processed by NLP algorithms. Accurate word representations are essential for capturing the semantic relationships between words and phrases, which are vital for tasks such as question understanding and answer generation.

Several types of word representation techniques are used in natural language processing (NLP), including classical models and distributed representations. Classical models such as WordNet, one-hot encoding, and TF-IDF, as well as

distributed representations such as word2vec and Global Vectors for Word Representation (GloVe), have been widely used in natural language processing (NLP) tasks, including question answering systems. These techniques help extract relevant entities and concepts from the text, which is necessary for identifying the type of question being asked and generating appropriate answers. The quality of word representations directly affects the performance of question-answer systems. Inaccurate or incomplete representations can lead to poor performance because the system may not be able to correctly identify the type of question being asked or generate appropriate answers. In contrast, high-quality word representation can enable the system to identify relevant entities and concepts from the text, which is necessary for generating correct answers or providing high accuracy. Advancements in word representation techniques, particularly in distributed representations, have led to significant improvements in computers' ability to understand and process text-based content. These techniques enable computers to capture subtle semantic relationships between words and can be trained on large amounts of data, making them more effective for tasks such as answering questions.

Consequently, word representation techniques have become an essential component of modern NLP systems, particularly for tasks such as question answering, machine translation, sentiment analysis, and natural language generation. Ongoing research in this area continues to improve the quality and effectiveness of these techniques, thereby further enhancing the capabilities of NLP systems.

Word Representation Techniques

Word representation techniques are essential in natural language processing (NLP) to convert words into numerical/vector representations (such that similar words have similar word representations) that can be easily processed by NLP algorithms. The following is an overview of the various word representation techniques.

Classical Models

Classical word-representation techniques are based on the idea of creating a sparse representation for each word in a vocabulary. These techniques are simpler and more transparent than distributed representations, and are often used as a baseline in NLP tasks.

WordNet

WordNet is a large lexical database of English that plays a significant role in the domain of natural language processing (NLP) tasks. It organizes words in a grammatical English format, such as English nouns, verbs, adjectives, and adverbs, into sets of cognitive synonyms called synsets, each expressing a distinct concept. These synsets are interlinked by conceptual-semantic and lexical relations, forming a network of meaningfully related words and concepts. WordNet connects words into various semantic relations, including synonyms, hyponyms, and meronyms. The synonyms were grouped into synsets with short definitions and usage examples, making WordNet a combination and extension of a dictionary and thesaurus. It is designed for use under program control and is considered an essential resource for researchers in computational linguistics, text analysis, and related fields. The structure of WordNet makes it a useful tool for computational linguistics and natural language processing tasks, providing ontological and lexical knowledge that can be leveraged in various applications. Therefore, we can say that WordNet is a large database that groups words together based on their meanings. It also defines relationships between words, such as how they are related or how they are different. For example, it groups words like "car," "vehicle," and "automobile" together because they have a similar meaning. WordNet also shows how these words are related to broader categories, such as "animal" or "mammal." This can be helpful in understanding the meanings of words and how they are related to each other. However, WordNet works only for the English language and is not perfect because it is created by humans.

One-hot Encoding

One-hot encoding is a classical word-representation technique used in the domain of natural language processing (NLP) to convert words into numerical/vector representations. In one-hot encoding, each word in the vocabulary is assigned a unique index and represented as a binary vector with a single '1' in the position corresponding to the word's index and '0's elsewhere. For example, in a vocabulary containing the words "cat," "dog," and "fish," the word "cat" would be represented as [1, 0, 0], the word "dog" as [0, 1, 0], and the word "fish" as [0, 0, 1], respectively, for the vector [cat, dog, fish].
One-hot encoding is simple and fast to create, making it suitable for small vocabularies and tasks in which semantic relationships between words are not crucial. However, this study had several limitations.

1. It does not capture the semantic relatedness among words because the vectors are orthogonal to each other.
2. This results in high-dimensional sparse representations that can be inefficient for large vocabularies.
3. It is inflexible when dealing with new words, as assigning new indices for new words changes the dimensions of the representation, potentially leading to compatibility issues.

Despite these disadvantages and limitations, one-hot encoding is still used to represent words in certain NLP tasks such as document classification and sentiment analysis, and serves as a foundation for more advanced word representation techniques.

TF-IDF

Term Frequency-Inverse Document Frequency (TF-IDF) is a widely used statistical method in the natural language processing (NLP) and information retrieval domains. It measures the importance of a term within a document relative to a collection of documents (i.e., corpus). The TF-IDF score is the product of two statistics: term frequency (TF) and inverse document frequency (IDF).

TF (term the frequency) measures how which a word appears in a document. The more often a word appears, the more important it is to a document. However, some words are common and appear in many documents; therefore, they may not be as important to the document as fewer common words. This is where IDF (inverse document frequency) occurs. This represents the local importance of a term within a document. The IDF measures the frequency with which a word appears in all documents. Words that appear in many documents (such as "the" or "and") have a low IDF, while words that appear in few documents have a high IDF.

By combining the TF and IDF, we can assign weights to words that are both frequent in a document and important to that document. The ((IDF) is calculated as the logarithm of the total number of documents in the corpus divided by the number of documents containing the term. It measures the global importance of a term, diminishing the weight of terms that occur frequently in the corpus, and increasing the weight of terms that occur rarely. The TF-IDF score is calculated as the product of TF and IDF: tf-idf (t, d) = tf (t, d) * log (N / (df + 1)), where t is the term, d is the document, N is the total number of documents, and df is the number of documents containing the term. A high TF-IDF score indicates that a term is important within a specific document, but less common across the entire corpus, making it a useful measure for tasks such as text analysis and natural language processing algorithms.

For example, if the word "cat" appears 10 times in a document, and there are 1,000 documents in the corpus, but the word "cat" only appears in 100 of those documents, then the TF-IDF weight of "cat" in that document would be higher than a more common word like "the," which appears in every document. TF-IDF is useful for tasks such as document classification, search engines, and information retrieval, as it helps identify important words in a document and can improve the accuracy of NLP algorithms.

Distributed Representation Models

Distributed models, also known as word embeddings, are a type of word representation technique that learns a dense vector representation for each word in a corpus by analyzing its context.

Word2Vec

Word2Vec is a prominent model for natural language processing (NLP) tasks. It is a technique for natural language processing (NLP) published in 2013 that uses a neural network to learn the context of words from a large text corpus. It efficiently creates word embeddings (dense vector representations of words), which are continuous representations of words that use semantic and syntactic relationships between them. The semantic meaning given by word2vec for each word in vector representations has served as a useful task in machine-learning text classification. For example, the vector representation of "king" might be closer to "queen" than to "car," reflecting the fact that "king" and "queen" are more semantically similar than "king" and "car." Word2Vec has various applications such as text similarity, recommendation systems, and other NLP tasks.

There are two main architectures in Word2Vec:

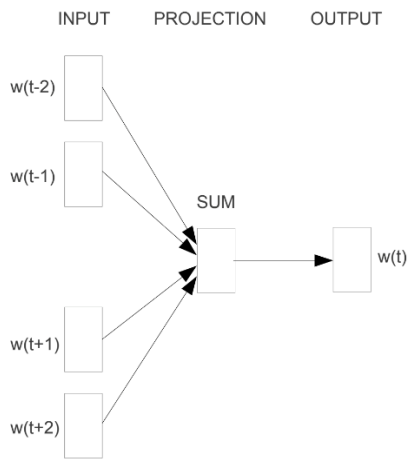
- Continuous Bag of Words (CBOW)
- Continuous Skip Gram

Both architectures use shallow, two-layer neural networks that are trained to reconstruct the linguistic contexts of words.

Upon training the word2vec model, it can detect synonyms or suggest additional words for a partial sentence. The major advantage of Word2Vec is its ability to group the word embeddings of similar types of words, making strong estimates of a word's meaning based on their occurrences in the text. The most famous example is the formula: "king" - "man" + "woman" = "queen."

Continuous Bag of words (CBOW)

It is one of the two main models of Word2Vec that represents words as a vector representation while keeping the context in mind. In this technique, the main goal is to predict a target word (the center word) based on source context words (surrounding words).



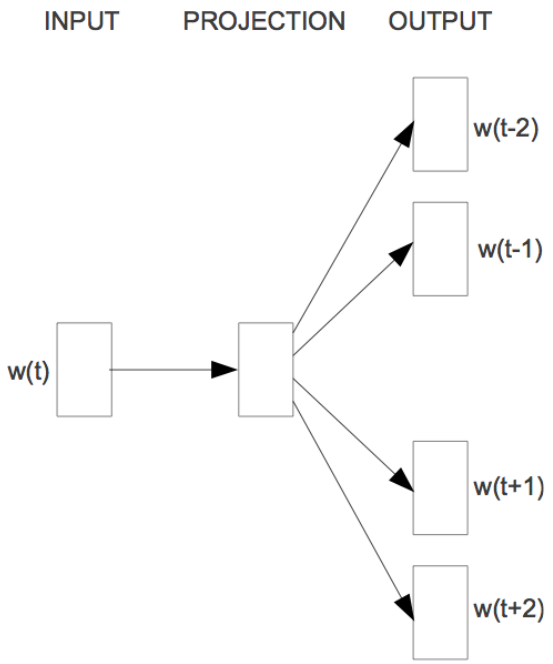
CBOW

To do this, a Continuous Bag of Words (CBOW) takes a group of words that surround the target word as input and uses a mathematical formula to come up with a number that represents that group of words. Now, this output number can be used to represent the target word by changing it to a format that the computer can understand. For example, consider the sentence "the quick brown fox jumps over the lazy dog." In CBOW, the model would try to predict the word "fox" based on the context words "quick" and "brown" (assuming a context window of size 1). The model is trained on a large dataset of text using an optimization algorithm, and the learned word embeddings capture the semantic and syntactic relationships between words.

CBOW is useful because it is very good at understanding the meaning of a sentence as a whole rather than just focusing on individual words. This makes it helpful for tasks such as determining the sentiment of a sentence (whether it is positive or negative) or classifying text into different categories.

Continuous Skip Gram

The Continuous Skip-Gram model is one of the two main architectures in Word2Vec, along with the continuous bag-of-words (CBOW) model. In contrast to CBOW, which predicts a target word based on its surrounding context words, the skip-gram model predicts context words given a target word. The goal of this technique is to predict words that are likely to appear near a target word in a sentence.



Skip-Gram Model

To do this, the continuous skip-gram model takes a target word and window of context words (the words that come before and after the target word in a sentence) as input and tries to predict each context word. The above process is iterated through a sentence for each word in the sentence, building up a set of predictions that can be used to generate word embeddings. For example, consider the sentence "the quick brown fox jumps over the lazy dog." In the Skip-Gram model, given the word "fox," the model would try to predict the context words "quick" and "brown" (assuming a context window of size 1).

Skip-gram tends to perform better with small datasets and can better represent less frequent words compared to CBOW. However, it is computationally more expensive than CBOW, because it predicts multiple context words for each input word. The choice between CBOW and Skip-Gram depends on the specific requirements of the NLP task and the available resources.

GloVe

GloVe, short for Global Vectors for Word Representation, is a word embedding technique used in natural language processing (NLP) that learns vector representations of words by leveraging global co-occurrence statistics from a corpus. Unlike Word2Vec, which relies on local context windows, GloVe captures both global and local information by constructing a large co-occurrence matrix of words in the corpus, and then factorizing it to obtain word embeddings.

GloVe aims to improve upon all of the other techniques by capturing the context of the word in the embedding through explicitly capturing these co-occurrence probabilities. The essence of GloVe is to build a matrix of these probabilities and subsequently learn a vector representation of each word, taking a moving window of a set size as the context across the corpus.

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

[Ref: GloVe]

The GloVe model first creates a matrix that counts how often each word appears in the same context as every other word in the corpus. This matrix was then used to calculate a set of weights that reflected the relative importance of each word in the corpus. These weights were used to adjust the weights of the neural network during the training. The word embeddings generated by GloVe are high-dimensional vectors that capture semantic relationships between words in a dense and continuous space. This means that words that have similar meanings or appear in similar contexts are located close to each other in vector space.

GloVe combines the advantages of both global matrix factorization methods (such as Latent Semantic Analysis) and local context window methods (such as Word2Vec), resulting in embeddings that are effective for various NLP tasks.

Impact of word representation techniques on question-answering systems.

We have seen how word representation techniques help in the conversion of words that are in human language to numerical representation/vector representations that can be interpreted by computers, or we can say that different other algorithms and state-of-the-art models take them as their input.

Word representation techniques play a crucial role in question-answering systems, as they enable computers to understand and process textual data. Different word representation techniques can be used in question-answering (QA) systems to improve performance by capturing semantic and syntactic relationships between words. These techniques include one-hot encoding, TF-IDF, Word2Vec, and GloVe. In QA systems, word embeddings are used to represent both questions and passages, enabling the model to measure the similarities between them and identify relevant information.

Classical models such as WordNet, one-hot encoding, and TF-IDF can help in identifying the type of question being asked and generating appropriate answers. For example, WordNet provides a hierarchical structure of words and their meanings, which can help identify synonyms and hypernyms of words in a given question. One-hot encoding can be used to represent words as binary vectors that indicate their presence or absence in a sentence, which can help identify the key entities and concepts in a given question. TF-IDF, on the other hand, can help identify the relevance of words in a given question by measuring how frequently they appear in the corpus of text.

Distributed representations like word2vec and GloVe, on the other hand, can capture the semantic relationships between words and phrases in a more accurate way. These representations can be used to train machine-learning models that can answer questions by predicting the most appropriate answer given a question and a set of candidate answers. For example, a question like "Who won the Nobel Prize in Physics in 2020?" can be answered by identifying the most relevant entities and concepts in the question, such as "Nobel Prize," "Physics," and "2020," and then using these to retrieve the most appropriate answer from a knowledge base or text corpus.

The choice of word representation technique can affect the performance of QA models. For instance, Word2Vec and GloVe embeddings capture semantic

relationships more effectively than one-hot encoding or TF-IDF, leading to better performance in tasks requiring semantic understanding.

In summary, the choice of the word representation technique plays a crucial role in the performance of QA models, with more advanced techniques generally leading to better results in capturing semantic and syntactic relationships between words and improving the overall performance of the QA system.

Question-answering models are significantly influenced by the use of word representation methods. The precision of word representations is necessary to grasp the semantic correlations between phrases and words, which are essential for tasks related to question comprehension and answer construction. There are various ways in which different word representation methods affect the performance of question-answer models.

- **Improved accuracy:** By utilizing methodologies such as word2vec and GloVe for word representation, question answering models have been able to achieve greater accuracy. These strategies capture the semantic connections between words and phrases, giving the models a more profound understanding of question meanings and assisting in the identification of the most helpful entities and concepts needed to produce correct answers.
- **Better generalization:** Including word representation methods in question answering models can significantly enhance their ability to adapt to various domains and languages, particularly through pre-trained word embeddings, such as GloVe and word2vec. Using these techniques, models can tackle new challenges beyond their initial training data and achieve greater generalization capabilities.
- **Faster training:** Building real-time question answering systems that offer swift and precise responses can benefit from word representation techniques, such as one-hot encoding and TF-IDF. Unlike distributed representations, these methods are computationally efficient and require less training.
- **Improved interpretability:** In legal and medical circles, interpretable models such as WordNet and TF-IDF are advantageous because they help comprehend how models arrive at solutions. Classical models are easily interpretable and crucial.

In conclusion, the way words are represented can affect how well question-answer models perform. Some techniques work better for certain situations than others. Factors such as task complexity, data size, and available resources determine the technique to use.

Techniques for Question Answering

Question Answering (QA) is a sub-domain of natural language processing (NLP) that deals with the answer to the question in a similar way that humans do in unseen passage solving. It includes the creation of algorithms/techniques and different models that can understand a given text and provide relevant and accurate answers.

Question Answering is an important and challenging task in the domain of natural language processing, which not only includes understanding the syntax and structure of the given text but also includes an understanding of the semantics and the context in which the given question is asked. In recent years, there have been many advances in the domain of natural language processing (NLP), owing to which there has been significant development in the field of question answering (QA) through the emergence of several deep learning techniques such as recurrent neural networks (RNNs), convolutional neural networks (CNNs), and transformers.

The training of the model for the question answering (QA) system can be performed on an open domain system or a closed domain, similar to some datasets. One of the most popular datasets for training and evaluating these techniques is the Stanford Question Answering Dataset (SQuAD). It is a large dataset consisting of articles from Wikipedia consisting of over 100,000 question-answer pairs and has become a benchmark for evaluating the performance of QA Models.

There are several different approaches for building a Question Answering (QA) systems, including rule-based systems, template-based systems, and machine learning-based systems.

Rule-based systems, also known as handcrafted systems, are based on a set of predefined rules and patterns to extract important features from a corpus of text, or can be used to extract answers from text. These sets of predefined rules are defined by experts in their domains and are based on linguistic, semantic, and syntactic knowledge. While rule-based systems can achieve good accuracy in some domains, they have limitations, such as the ability to handle complex questions and the requirement of significant expertise in a certain domain to develop the rules for the model.

Template-based systems, are similar to rule-based systems, which are based on a predefined template to answer a set of questions. It extracts text based on a

predefined template. These templates are designed to capture common patterns in questions and answers, and can be filled in with information from the text. For example, a template-based system might have a template for questions about the birthplace of a person, with placeholders for the name of the person and answer. The system then searches for the name of the person in the text and fills the answer. These systems are more flexible than rule-based systems, but are still limited owing to their pre-defined template sets.

Machine learning-based systems, are different from the above models as they do not use any predefined rules or templates to predict the answer to the questions. Instead, they use statistical models and algorithms to find patterns and relationships in the data and make predictions about the answer for a given question. These systems are trained on a large corpus of text to learn different patterns and obtain accurate answers to the given questions. They used these data to learn how to identify relevant information in the text and generate accurate answers. There are several models/types of machine-learning algorithms in the domain of Question Answering (QA) systems, such as classification, regression, clustering, and deep learning.

Of these several types of methods, one popular machine learning-based technique for question answering is the use of neural networks, which are models that are inspired or mimic the human brain. Neural Networks are a type of model that contain several interconnected layers that process the input to generate the output. In question answering systems, neural networks can be used to encode the question and text and generate an answer based on the learned patterns and relationships in the data.

There are several state-of-the-art techniques based on neural networks in the domain of Question Answering (QA), such as BiDAF, BERT, and GPT etc. The choice of technique for Question Answering (QA) in natural language processing (NLP) depends on the purpose, specific tasks, and available resources. While rule-based and template-based systems can be useful for simple question-answering tasks, machine learning-based techniques offer higher accuracy and greater flexibility for handling complex questions and large amounts of data.

Bidirectional Attention Flow – BiDAF

Bidirectional attention **flow** (BiDAF) is a closed-domain, extractive Q&A model that can only answer factoid questions.

The structure of BiDAF can be described in three hierarchical layers:

- **Embedding Layers**

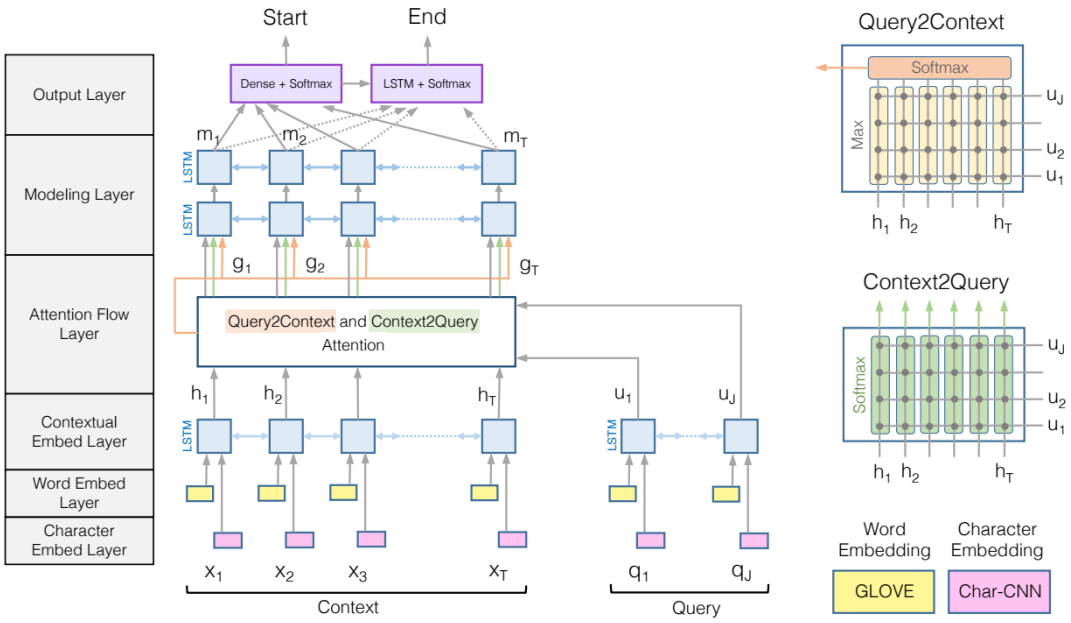
There are 3-word embeddings layers are present in the BiDAF model, whose work is to change the Context and Query representation from text to numerical/vector representation (from strings into vectors of numbers).

- **Attention and Modeling Layers**

The output of the previous layers, which is the vector representation of Context and Query, is then sent to these attention and modelling layers. These layers perform several mathematical matrix operations to fuse the information contained in the context and the query representation. The output of these layers is a vector representation of the context that contains information from the query. The output to these layers is referred to as "Query-aware Context representation." In the paper named *Bidirectional Attention Flow for Machine Comprehension*.

- **Output Layer**

Now the output of the previous layers that is "Query-aware Context representation" is passed to the final output layer of the BiDAF model, which will transform it to group of probability values. These values refer or say depicts to where the answer to the given query starts and ends in the context.



[Ref: *Bidirectional Attention Flow for Machine Comprehension paper*]

EMBEDDING LAYERS

Now, let us consider the operation of the BiDAF model:

Tokenization

The input to the model was a passage called Context and the question Query. The first step is to divide the Context and Query into words, called tokenization. In the BiDAF paper, the symbols T and J denote the number of words in Context and Query, respectively.

Word level embedding

Now, all words are converted into vector representations such that the vector representations of these words capture the syntax and semantics of the words.

In the BiDAF model, word embeddings are performed on three levels of granularity:

- Character Level

- Word Level
- Contextual Level

Let us begin with the word embedding layer:

The technique used for word embedding in the BiDAF model was GloVe. The BiDAF model uses pretrained GloVe embeddings to convert words in the Context and the Query to its vector/numerical representation.

The output of this layer consists of two matrices, one for the Context and the other for the Query.

Character Level Embedding

GloVe can provide the word embedding for most of the words. However, there are some words that are not in the pre-trained model of the GloVe, which we call out-of-vocabulary words. GloVe simply assigns any random vector to them while changing the words into their vector representations. If this random vector is not removed, it can confuse the BiDAF model. Therefore, we used another level of word embedding, called character-level embedding. Character level embedding uses a one-dimensional convolutional neural network (1D-CNN) to find numeric representations of words by examining their character-level compositions. The output for this layer is similar to the output of the word-embedding layer, which consists of two matrices, one for the Context and the other for the Query.

Highway Network

Now, we have two sets of word representations: one is given by word-level embedding (GloVe) and the other is given by character-level embedding (1D-CNN). The next step was to concatenate these representations.

The output for this process is a set of two matrices: one for the Context and other for the Query. These matrices are then passed to a highway network that is similar to a feedforward network. The role of the highway network is to adjust the relative contributions from the word-embedding and character-embedding steps. In simple terms, we prefer the word representation of GloVe if the word is in the vocabulary of GloVe and if the word is out of vocabulary, the 1D-CNN, that is, character-level word representation, is preferred for that word.

The outputs of the highway network are again two matrices: one for the context and one for the query. They represent the adjusted vector representations of words in the Query and the Context from the word- and character-embedding steps.

Contextual Embedding

Now we have word embedding for the context as well as the query, but the problem is that these embeddings do not take into account the words' contextual meaning, which is the meaning derived from the words' surroundings. Therefore, we processed word embedding through a contextual embedding layer. The contextual embedding layer consists of long short-term memory (LSTM) sequences. The BiDAF employs a bidirectional LSTM (bi-LSTM), which is composed of both forward and backward LSTM sequences.

The output of this layer is again two matrices, one for the context and the other for the query, but now the word embedding also has a contextual meaning in its representation.

ATTENTION FLOW LAYER

Formation of Similarity Matrix

In this step, it takes the output of the previous step, which is two matrices: one for the context and the other for the context. Then, using the concept of the seq2seq model, a similarity matrix was formed from the above two matrices. This entails applying a comparison function to each column in the context matrix (H) and each column in the query matrix(U).

Just because a Context word and query word are identical does not necessarily imply that their vector representations are highly similar. On the other hand, we can see that the similarity value of two distinct words with close semantic and grammatical meaning, such as "situated" and "located, is relatively high. This is due to our word and character embedding layers, which can generate vector representations that accurately reflect a word's meaning.

A comparison function is used to form this similarity matrix. This step is the output of this step.

Context-to-Query Attention (C2Q) Attention

The similarity matrix acts as an input for this step in the Attention Flow layer of the BiDAF model. This is the same as the second and third steps of the seq2seq model.

Some features of this step are as follows.

- Semantic similarity strongly contributes to the relevance.
- Context words "understand" the information requested by the query words.

Outside of this step is a matrix representation of Context, as previously described. However, this time it contains different types of information. As the previous representation encapsulates the semantic, syntactic, and contextual meanings of each Context Word, this representation encapsulates the information about the relevance of each query word to each context word.

Query-to-Context (Q2C) Attention

This step was similar to the previous C2Q step.

In this step, our goal is to find which context word is most similar to either of the query words; hence, it is critical to answer the query.

The output of this step is another representation of the context that encapsulates information about the most important words in the context with respect to the query.

Mega – Merge

The goal of this step is to combine all three matrices output by the previous layers, that is,

- The original context matrix encapsulates the semantic, syntactic, and contextual meanings of context words.
- Context matrix that encapsulates the relevance of each query word to each context word.
- Context matrix that encapsulates information about the most important words in the context with respect to the query.

We merged all these matrices via a mathematical function used in the Attention Layer in the BiDAF model.

The output of this step is a large matrix. That is, you can think of each column vector in the output matrix as a vector representation of a context word that is "aware" of the existence of the query and has incorporated relevant information from it.

MODELING LAYER

The two input modelling layers are the output of mega-merge, which is a giant matrix. The modelling layer was a collection of two bi-LSTM layers.

These two layers use the giant matrix as the input and output matrices. The difference between these representations and the previous representations of context words is that these representations have embedded in them information about the entire context paragraph as well as the query.

OUTPUT LAYER

Now, we have the representation of each word in the form of two numeric vectors that encode the word's relevance to query. The last step is to obtain two probability values from these two numeric representations of the context words so that we can compare the query relevance of all context words.

We obtain two probabilities as the start and end indices, as depicted in the images below.

$\mathbf{p1} = \text{softmax}(\mathbf{w}^{T}_{(p1)} [\mathbf{G} ; \mathbf{M1}])$, where...

- **p1** is a vector of probabilities for each Context word of it being the first word in the Answer. **p1** has the same number of elements as the number of words in the Context (**T**)
- $\mathbf{w}^{T}_{(p1)}$ is a 1-by-10d trainable weight vector
- **G** is an 8d-by-T matrix from the "megamerge" step
- **M1** is a 2d-by-T matrix obtained in step 9 by passing **G** through a bi-LSTM layer
- [;] is vector concatenation across row

[Ref: Wiki]

$p2 = \text{softmax}(w^{T}_{(p2)} [G ; M2])$, where...

- **p2** is a vector of probabilities for each Context word of it being the last word in the Answer; it has the same number of elements as the number of words in the Context (**T**)
- $w^{T}_{(p2)}$ is a 1-by-10d trainable weight vector
- **G** is an 8d-by-T matrix from the "megamerge" step
- **M2** is a 2d-by-T matrix obtained in step 9 by passing **G** through a bi-LSTM layer
- [;] is vector concatenation across row

[Ref: Wiki]

Then, p1 and p2 were used to find the best answer span. The best answer span was simply a substring of the context with the highest span score.

Hence, we obtain the final output as the starting index and end index from the context, which is our answer to the given query.

The key features of BiDAF include its ability to pinpoint the location of the answer within a context and its use of bidirectional attention to capture the interaction between context and query. This approach has shown strong performance in the QA domain, particularly on the SQuAD dataset.

STRENGTHS

- BiDAF has achieved state-of-the-art performance on the SQuAD benchmark, which is considered to be one of the most challenging question answering datasets.
- BiDAF can model bidirectional interactions between questions and passages, thus capturing the semantic relationships between words in both questions and passages.
- BiDAF uses the character-level embedding of words. This allows the model to handle out-of-vocabulary words and to improve the performance of rare words.
- BiDAF uses highway meshes to prevent overfitting, which is a common problem in deep-learning-based models.

WEAKNESSES

- BiDAF is computationally intensive and requires numerous resources for training.
- However, BiDAF may not perform well on datasets that differ significantly from the SQuAD benchmark.

- BiDAF may address issues that require justification beyond passage.

Overall, BiDAF is a powerful question-answering technique that achieves a state-of-the-art performance on the SQuAD benchmark. It is particularly effective at modeling interactions between questions and sentences, and can capture semantic relationships between words in both directions. However, it may not be the best choice for all question-answering tasks, and may require significant computational resources for training.

Bidirectional Encoder Representation from Transformer – BERT

This is a popular state-of-the-art technique in the domain of natural language processing (NLP) that can be used for several NLP tasks such as classification, text prediction, and question answering.

Transformers were used in the BERT machine-learning framework. Every output component is connected to every input component in the transformer, and weights are assigned to establish the appropriate relationships. We call this issue attention. Unsupervised language modelling was used by BERT to pretrain the model on a larger dataset. The model can understand the context of the input text through pretraining on a sizable dataset. BERT can produce promising outcomes by fine-tuning the model using task-specific supervised data.

At this point, two approaches can be used: feature-based and fine-tuning. The feature-based method is used by ELMo (Embeddings from Language Models), in which the model architecture is task specific. This implies that various models and previously taught language representations were employed in each job. This implies that various models and previously taught language representations were employed in each job.

The BERT model uses bidirectional transformer encoders and fine-tuning to understand language, and hence, its name. It is important to remember that BERT can comprehend a word's whole context. BERT examines words that come before and after a term to assess their association.

One of BERT's important characteristics of BERT is its bidirectionality, which enables it to consider both the words that come before and after a particular word when processing it. Traditionally, NLP models have been unidirectional, considering only one direction of context at a time. BERT also uses a multilayered transformer architecture, which enables it to recognize intricate word associations in the context.

MODEL ARCHITECTURE

The architecture of the BERT model is a multilayered model that uses several layers of bidirectional transformers.

The architecture of a transformer consists of an encoder that is used to embed the input, and a decoder that is used to decode the embedded output back into text (a string). This process can be observed in the encoding-decoding algorithms.

BERT is a type of neural network that employs numerous transformers layered on top of one another, similar to the transformer architecture. The context and meaning of each word are captured, as these transformers parse the input text. However, BERT can have multiple layers stacked on top of each other, unlike conventional transformers, which only have one layer depending on the particular task it needs to complete. In addition, the input embeddings—the numerical representations of the input text—are adjusted to the task at hand and then run through a unique classifier created for that particular task. To create the final output, this classifier further processed the updated embeddings.

Simply put, BERT represents words and parts of words in a sentence using tiny objects referred to as "tokens". Each token is assigned a special code that represents what it means for it to appear in the sentence. To comprehend the overall meaning of a text, BERT employs these codes to comprehend how words are related to one another in a phrase. This is distinct from simply examining words on their own without considering their relationship to one another in a phrase. These token representations enable BERT to accurately comprehend and analyze natural language content.

Here, is a list of tokens used by the BERT model.

- [CLS]: Represents the beginning of a sentence and is used to understand the meaning of the entire sentence in classification tasks.
- [SEP]: Separate two sentences or a question and answer in question-answering tasks.
- [MASK]: Used to mask a word during pre-training and fine-tuned to predict the masked word.
- [UNK]: - represents an unknown word that is not present in the vocabulary.
- [PAD]: Used for padding to create all input sequences of the same length.

These tokens aid the model's comprehension of the text and improve its accuracy when performing tasks, such as categorization and question answering.

Tokenizer

The tokenizer, which is the first component of BERT, receives raw text as the input and separates it into tokens. The fundamental textual components used in NLP are tokens. To create a series of tokens, the tokenizer analyzed the text using a particular vocabulary and dividing rules.

Input Embeddings

The input embeddings of BERT map each token to a high-dimensional vector representation after the text has been tokenized. Based on each token's context within the sentence, these vectors capture the meaning of each word. The encoder, the next major component of BERT, is subsequently fed with the input embeddings.

Encoder

The input embeddings are processed by a multi-layer bidirectional transformer encoder in BERT. The encoder is made up of numerous levels of stacked transformers, each of which handles the input embeddings differently. The associations between the tokens and the context in which they appear in the input text are captured by BERT thanks to these transformer layers.

Masked LM Head

A task-specific layer that is trained to anticipate masked tokens in the input sequence is the Masked Language Model (MLM) head. BERT randomly masks some of the input tokens during pre-training and trains the model to predict the original values of those tokens based on the context of the tokens around them. As a result, BERT may learn to fill in the blanks when a phrase has a missing word.

Next Sentence Prediction Head

The Next Sentence Prediction (NSP) head is another task-specific layer that is trained to predict whether two input sentences are consecutive or not. This helps BERT capture contextual relationships between sentences. For example, in a

question answering task, BERT can use NSP to understand if a given answer is logically related to the question.

Pooler

A fixed-length vector representation of the input sequence is then produced by BERT's pooler using the output of the last transformer layer. This vector can be used as an input for tasks that come after it, such regression or classification. The pooler is helpful when the model's output must have a fixed length regardless of the length of the input sequence.

To summaries, the tokenizer, input embeddings, encoder, masked LM head, next sentence prediction head, and pooler are the essential components of BERT. Together, these components make BERT an efficient architecture for a variety of NLP applications by enabling it to capture the meaning and context of words and phrases in text.

WORKING MECHANISM

The following outline how BERT operates:

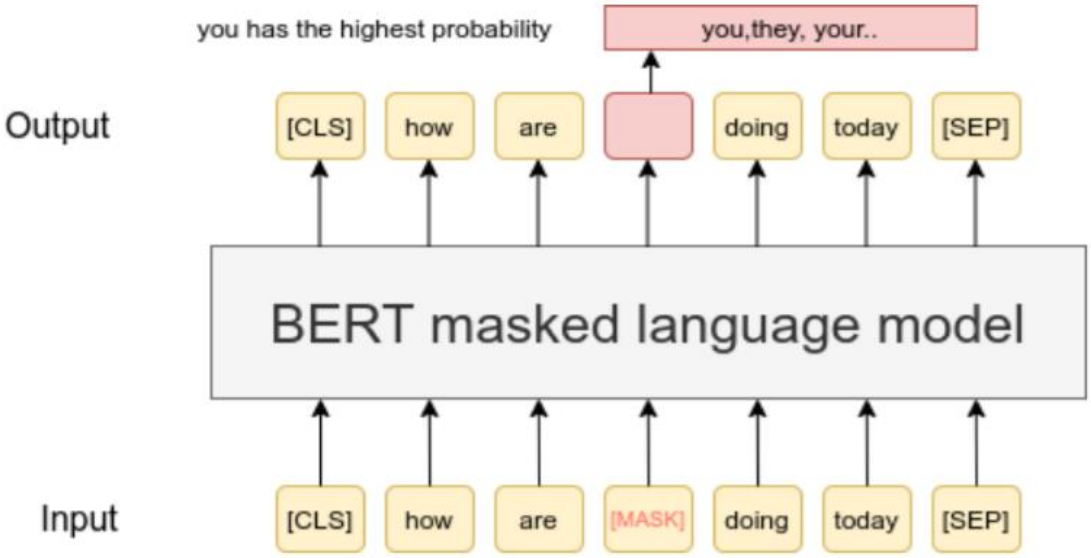
Training data in large amounts

Simply put, BERT is built to comprehend a wide variety of words and languages, allowing it to use massive amounts of data to develop a thorough understanding of English and other languages. But it can take a while to train BERT to learn all of this information. To help with this, BERT uses a special type of architecture called a transformer, which makes the training process more efficient. Tensor Processing Units (TPUs) can also be used to accelerate training even more.

Masked Language Model

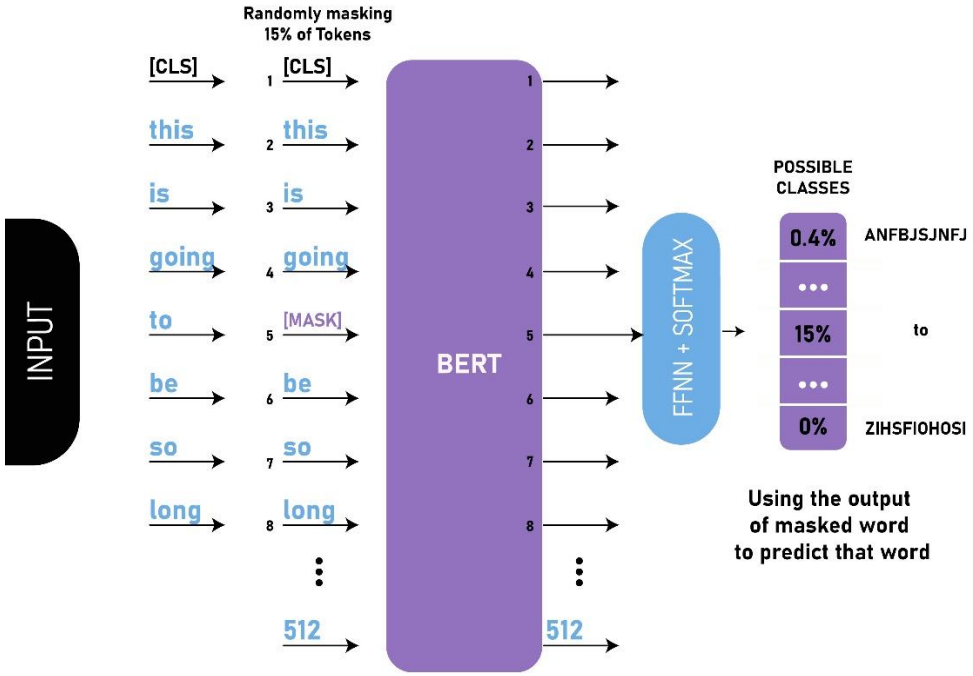
The Masked Language Model (MLM) method is used by the Bidirectional Encoder Representations from Transformers (BERT) to learn from text in both directions. In MLM, a word is concealed within a sentence, and the hidden word is then predicted using BERT using the words on each side of the hidden word. By taking into account the words that come before and after a word, BERT is able to

comprehend its context in this way. In layman's terms, it's similar to asking BERT to fill in a blank in a sentence using the words around it. By utilizing MLM, BERT is able to generate more precise predictions while processing language since it has a better understanding of how words relate to one another.



[Ref: https://quantpedia.com/bert-model-bidirectional-encoder-representations-fromtransformers]

BERT will attempt to infer the missing word from the terms that come before and after it, for instance, in the sentence "I am going to the ____." In order to comprehend the context of the sentence, it does this by examining the words on both sides of the missing word. Through practice, BERT can improve its ability to predict missing words with accuracy. During training, 15% of a text's words are randomly obscured or "masked," and BERT's task is to ascertain the meaning of the missing words from the ones around them.



[Ref: https://quantpedia.com/bert-model-bidirectional-encoder-representations-fromtransformers]

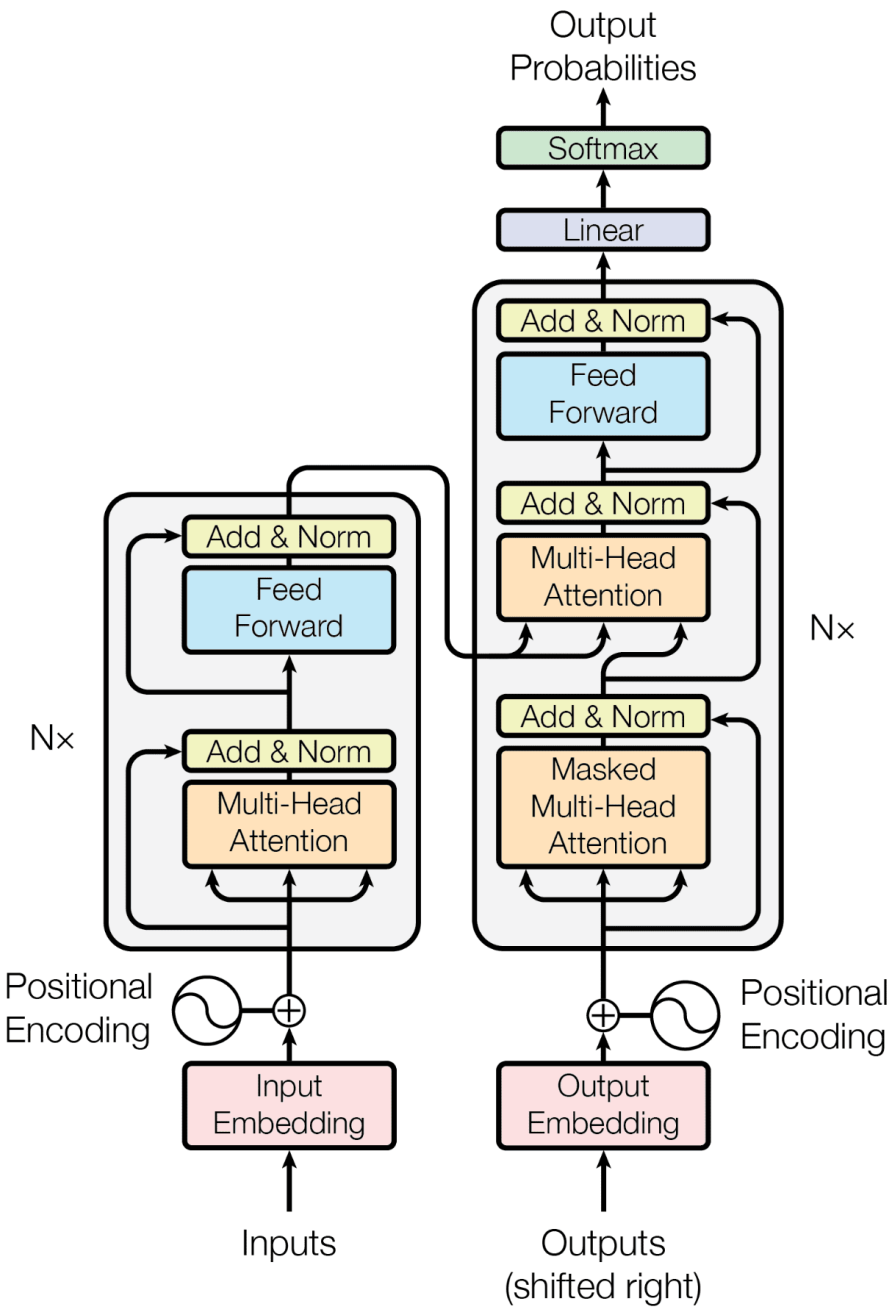
Prediction of Next Sentence

BERT employs the Next Sentence Prediction (NSP) method to determine how various sentences are related to one another. As an illustration, if the first sentence begins, "The cat sat on the mat," the second sentence can start, "It began to purr." BERT would use NSP to recognize the relationship between the first and second sentences and that the cat must be content. In order to learn about the relationships between sentences, BERT observes pairs of sentences that are correctly related as well as pairs that are randomly paired.

Transformers

When training models like BERT to process natural language, the transformer architecture is a potent tool. It employs a method called attention, which concentrates on a sentence's crucial components while avoiding time-wasting details. Imagine it as underlining the important passages in a text. Encoders and decoders are the two basic components of transformers. The decoder foresees the output after the encoder has processed the input text. Transformers can effectively

handle massive amounts of data and learn from it by layering multiple encoders and decoders on top of one another. Because of this, they are especially beneficial for jobs like language processing where a lot of data is available.



[Ref: https://quantpedia.com/bert-model-bidirectional-encoder-representations-fromtransformers]

STRENGTHS

- **Contextualized word embeddings:** BERT creates contextualized word embeddings, enabling it to discern a word's meaning from its surroundings.
- **Pre-training and fine-tuning:** A number of NLP tasks are significantly improved by BERT thanks to its large-scale pre-training on unstructured data and fine-tuning on particular tasks with smaller labelled datasets.
- **State-of-the-art performance:** BERT is a preferred option for researchers and practitioners because of its outstanding performance on several benchmarks.

WEAKNESSES

- **Computational expense:** BERT is a computationally intensive algorithm that needs a lot of resources and training data.
- **Inference time:** BERT can be computationally expensive at the inference stage, making its usage in scaled manufacturing difficult.
- **Limited understanding:** As a data-driven model, BERT may have trouble with some linguistic features like common sense and pragmatic inference.

To use BERT for particular NLP tasks and applications, it is critical to comprehend their strengths and weaknesses.

Methodology

It is important to know that how we can use different state-of-the-art techniques by knowing its accuracy on different data set. So, we have given a set of steps you can follow to apply a technique to a particular dataset.

BIDAF APPLICATION ON SQUAD

- **Prepare the SQuAD dataset:** A series of queries and responses pertinent to a specific context are included in the SQuAD dataset. Preparing the data by dividing it into training and validation sets is the first stage in applying BiDAF to the SQuAD dataset. In order to train the model on the training set and assess its effectiveness on the validation set, this is done.
- **Pre-process the data:** To feed the SQuAD dataset into the BiDAF model, pre-processing is required. Tokenizing, lowercase conversion, stop word elimination, and text conversion to numerical form are all steps in the pre-processing process.
- **Build the BiDAF model:** Multiple layers of attention-based convolutional and recurrent neural networks (RNNs) make up the BiDAF model. The context and the question are the model's two inputs, and it determines the most pertinent excerpt from the context that responds to the question to provide the response.
- **Train the BiDAF model:** The BiDAF model must then be trained using the SQuAD dataset. The weights and biases of the model are optimised during training in order to reduce the loss function. The discrepancy between the predicted and actual results is measured by the loss function.
- **Fine-tune the BiDAF model:** The BiDAF model needs to be adjusted after it has been trained on the SQuAD dataset in order to perform better on particular tasks. The model's hyperparameters, such as learning rate and batch size, are changed during fine-tuning, and it is trained on a dataset tailored to the job at hand.
- **Evaluate the BiDAF model:** Evaluation of the BiDAF model's performance using the SQuAD dataset is the last step. By contrasting the model's projected results with the actual results in the SQuAD dataset, one may determine how

accurate the model is. Common measures for assessing the success of the model include the F1 score and the Exact Match score.

In conclusion, establishing the BiDAF model, training the model, fine-tuning the model, and assessing its performance are required to execute and optimise the BiDAF technique on the SQuAD dataset.

BERT APPLICATION ON SQUAD

- **Data Pre-processing:** To make the SQuAD dataset usable by BERT, pre-processing is performed on it. This entails utilising BERT's tokenizer to tokenize the input text into sub words and mapping the tokens to the associated token IDs.
- **Model Architecture:** BERT is a pre-trained model, meaning it has previously undergone extensive training on a variety of data. As a result, we may fine-tune the pre-trained BERT model using the SQuAD dataset. The input embedding layer, the multi-layer bidirectional transformer encoder, and the pooler layer are some of the layers that make up the BERT model. We additionally build a task-specific output layer on top of the BERT model for the SQuAD dataset.
- **Fine-tuning:** By training it on the SQuAD training set, the pre-trained BERT model is adjusted for the SQuAD dataset. The pre-trained BERT model's weights are modified during fine-tuning to better fit the SQuAD dataset. The SQuAD dataset is used to train the model to reduce the cross-entropy loss between the anticipated start and finish positions of the answer and the actual answer positions.
- **Evaluation:** The performance of the improved BERT model is assessed on the SQuAD development set. The F1 score and exact match score are calculated after the model's predictions on the development set are contrasted with the correct answers.
- **Inference:** The model can be used to forecast responses to new queries after being adjusted and assessed. The BERT tokenizer is used to first tokenize the input text and turn it into token IDs. The start and end positions of the response in the input text are then predicted using the refined BERT model. By choosing the subtext between the projected start and end places and turning it back to text, the predicted response can be found.

Evaluation Metrics

To evaluate the performance of BiDAF and BERT techniques on the SQuAD dataset, two primary metrics are used: Exact Match (EM) and F1 score.

EXAT MATCH (EM) SCORE

The proportion of anticipated answers that exactly match the ground truth answers is known as the EM score. A better model performance is indicated by a higher EM score.

F1 SCORE

The F1 score is the harmonic mean of accuracy and recall, where precision is the percentage of properly anticipated responses and recall is the percentage of correctly predicted answers from the ground truth. A higher F1 score suggests that the model is doing better.

Formula

$$F_1 = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}} = \frac{\mathrm{TP}}{\mathrm{TP} + \frac{1}{2}(\mathrm{FP} + \mathrm{FN})}$$

TP = number of true positives

FP = number of false positives

FN = number of false negatives

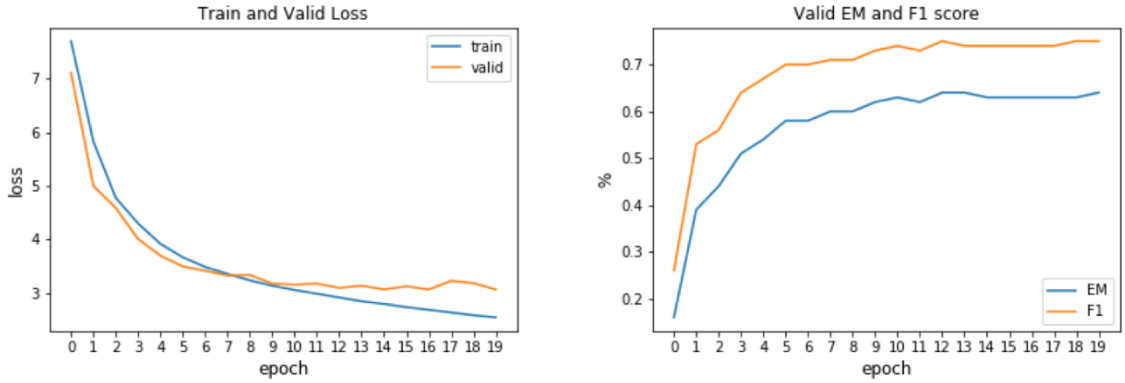
[Ref: Wikipedia]

The F1 score is the harmonic mean of precision and recall, which evaluates the overlap between anticipated and ground truth responses. The EM metric counts the proportion of predictions that exactly match the ground truth answers. By accounting for both the precision and recall of the model's predictions, these metrics offer a quantitative assessment of the accuracy of the model's conclusions.

The EM and F1 score metrics are relevant since they evaluate the model's capacity to offer the right answer, and the SQuAD dataset is made for answering questions. These metrics offer a uniform way to assess the effectiveness of various models and enable researchers to track the development of question-answering systems.

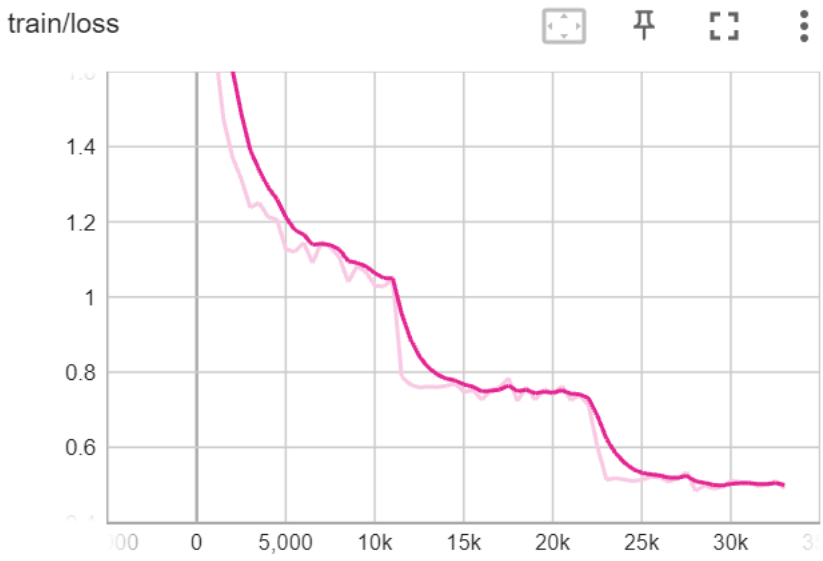
Results

BIDAF



BiDAF evaluation metrics on SQuAD Dataset
[Ref: *https://github.com/ElizaLo/Question-Answering-based-on-SQuAD*]

BERT



BERT training loss metric for SQuAD Dataset
[Ref: *https://huggingface.co/huggingface-course/bert-finetunedsquad*]

	Exact Match Score	F1 Score
BiDAF	64%*	75%*
BERT	80.9%*	88.2%*

[Ref: *https://github.com/ElizaLo/Question-Answering-based-on-SQuAD*]
[Ref: https://huggingface.co/csarron/bert-base-uncased-squad-v1]

**The accuracy scores can vary depending on the parameters and hyperparameters while training of the models.*

References

[1] *How Does BERT Answer Questions? A Layer-Wise Analysis of Transformer Representations*

[2] Usman Naseem, Imran Razzak, Shah Khalid Khan, Mukesh Prasad, *A Comprehensive Survey on Word Representation Models: From Classical to State-Of-The-Art Word Representation Language Models*

[3] Zhiyuan Liu, Yankai Lin & Maosong Sun, *Word Representation*

[4] Minjoon Seo, Aniruddha Kembhavi, Ali Farhadi, Hannaneh Hajishirzi, *Bidirectional Attention Flow for Machine Comprehension*

[5] Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova, *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*

[6] Devlin, J., Chang, M. W., Lee, K., & Toutanova, K. (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.*

[7] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). *Attention is all you need. Advances in neural information processing systems*.

[8] Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... & Amodei, D. (2020). *Language models are few-shot learners. Advances in neural information processing systems.*

[9] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., ... & Brew, J. (2019). *HuggingFace's transformers: State-of-the-art natural language processing.*

[10] Han, X., Zhu, H., Ye, M., & Guo, X. (2020). *An overview of the BERT architecture and its application in NLP.*