Community Question Answering Ranking: Methodology Survey

Ahmed Zaazaa*[1], Mohsen Rashwan*[2], Ossama Emam*[3]

* *Electronics and Communication Department, Faculty of Engineering, Cairo University Giza, 12613, Egypt*

[1]azaazaa@eg.ibm.com

[2] mrashwan@rdi-eg.ai

[3]ossama.emam1@ibm.com

Abstract: ***This paper surveys the evolution of word embeddings along with the methodologies used in Community Question Answering (cQA), and how these methodologies use word embeddings to achieve higher performance metrics. The paper first discusses vector modelling and how it affected Natural Language Processing (NLP) as a whole, then it details some of the approaches used like the one-hot-encoding, word2vec and others. The paper then discusses contextualized embeddings and how they improve on the previous techniques. The paper then sheds some light on language modelling along with new attention-based architectures (Transformers), discussing briefly how they work and how they affected not only cQA but NLP in general. Then the paper discusses in brief the shift in the field from model-based AI where most of the focus is on producing a model with high performance metrics to Data Centric AI where the focus is on trying to have a systemic way of labelling the data to ease the generation of a high-performance model.***

Key words: ***Machine Learning (ML), Deep Learning (DL), Natural Language Processing (NLP), Community Question Answering (cQA), Ranking***

1 INTRODUCTION

Many of the systems that handle language focus on the downstream task performance metrics while turning a blind eye to other aspects a Machine Learning based system should include, such as training time, inference time and ease of operationalization. So, the paper explores how state of the art methods can improve NLP tasks in general and cQA specifically.
An appropriate pipeline design handles both functional requirements reflected in getting the agreed-upon score in a specific performance metric, such as the metric used in ranking: Mean Average Precision (MAP), while maintaining a design that is easily operationalized. However, most of the state-of-the-art approaches in cQA only focus on getting high MAP disregarding the other aspects of a balanced system such as maintainability and ease of operationalization, rendering the model incapable of being easily deployed on production environments.
The problem in the state-of-the-art approaches stems from the increasing number and/or complexity of the models used to classify relevant documents/answers or rank them, as well as from the increasing number and complexity of features being generated to be fed to said models. In a production environment, a system of such complexity will be very exhausting to run efficiently and maintain.
These approaches were inevitable when it comes to cQA for Arabic language, because of the scarcity of Arabic data and specifically the scarcity of data that could be used in cQA. However, what is proposed in this paper is: instead of focusing on extracting multiple features (typically sentences or embeddings) and feeding them to the potentially – multiple - models, we focus on properly representing the sentences in cQA and feed these representations (embeddings) to a single high-performance model.

The motivation behind choosing cQA as a downstream task is due to two main reasons: the first is that it is very important to have a system that can answer questions in our daily lives, and that could be seen by how virtual assistants developed by google, apple and amazon became successful on their launch. The systems developed by these three companies (and several others) have been used extensively in our everyday chores, however, systems that are good in specialized tasks, such as law, health and history are lacking and, in some fields, non-existent such as religious casuistry (الافتاء). The second reason for choosing cQA is simply because it is very interesting. Artificial Intelligence (AI) – especially deep neural models - has exceeded expectations in pretty much every field it was introduced to, now with researchers taking interest in "Explainability" it would be enlightening to know why deep learning-based models answers the way they do. The end goal

is to take a step towards having a pretrained model fit in a variety of domains with little to no effort, keeping in mind performance measures like storage and speed.
Choosing Community Question Answering task over a Question Answering task is due its practical potential – especially in the middle east and Arabian community – where communities are a lot scarcer and much less maintained than that in English; Not to say that that Question Answering has no potential, but people nowadays are more reliant to find answers to their questions on communities rather than searching through books or references.

2 Literature Review

Since a typical pipeline makes use of several technologies, the paper will review multiple fields. This review will be talking about Vector Space Modelling, a brief history of Word Embeddings, why NLP became stale as well as brief introductions to Contextualized Embeddings, Transfer Learning, Transformers and Data Centric AI. Where in each section, only the techniques with the highest impact are discussed.

A. Vector Space Modelling

Machines by default cannot understand text natively, they only understand numbers. That is why many approaches have been developed to turn words into Vectors that machines can understand and process. In this section, we will begin with the simplest approach "One-Hot encoding" to get the intuition of how words are represented to the computer, then we will jump to word2vec to get the intuition of embeddings and how they are generated.

1) One-Hot Encoding:

The first approach that comes to mind, is to have a Vocabulary vector $\underline{\mathbf{V}} \in R^{v}$ where 'v' is the size of the vocabulary, and have each word in the text (Sentence, document, etc..) be represented in a certain index (or dimension) within that vector by a number (0 or 1) as shown in Figure 1. This approach is called "One Hot Encoding", and even though it is simple to implement, it has the following main caveats: -

- The average vocabulary size used by current systems averages around 35,000, meaning the vector size would also be of the same size, which is huge and will cause unnecessary processing overhead.
- More importantly, each word in the vocabulary is orthogonal to each other word, hence it does not capture relationships between words. (i.e., dog is similar to fox as it is similar to lazy)

word\dimension	1	2	3	4	5	6	7	8	9
...					...				
the	1	0	0	0	0	0	0	0	0
quick	0	1	0	0	0	0	0	0	0
brown	0	0	1	0	0	0	0	0	0
fox	0	0	0	1	0	0	0	0	0
jumped	0	0	0	0	1	0	0	0	0
over	0	0	0	0	0	1	0	0	0
lazy	0	0	0	0	0	0	1	0	0
dog	0	0	0	0	0	0	0	1	0
...					...				
Zebra	0	0	0	0	0	0	0	0	1

Figure 1: Illustration of one-hot-encoding scheme

2) Word2Vec:

Developed at Google in 2013 [1], Word2Vec is a neural approach to create dense vectors (embeddings) for word representations, and it is based on the distributional assumption that words having similar meanings are found in similar contexts. Word2Vec uses one of two architectures: Continuous Bag of Words (CBoW) or Skip grams.

CBoW is achieved by moving a sliding window of arbitrary size (N) through the training text, the training data is generated by having inputs be the words surrounding our target output within the sliding window. For example, given the sentence

"A fool thinks himself to be wise, but a wise man knows himself to be a fool" and N=3, the dataset should be as shown in TABLE I

TABLE I

EXAMPLE OF THE CBOW APPROACH

Input1	**Input2**	**Output**
A	Thinks	Fool
Fool	Himself	Thinks
Thinks	To	Himself
Himself	be	To
To	wise	Be

Each of the words are turned into embeddings and then passed to a shallow feed forward neural network, which in turn generates a probability distribution on what the next word might be. In short, the goal of the CBoW architecture is to predict a word by using its surrounding context.

The second architecture is the skipgram, where the goal is to predict the words surrounding the input word (or context), for simplicity, it could be thought of as the inverse of what the CBoW is doing.
Given the same sentence ***"A fool thinks himself to be wise, but a wise man knows himself to be a fool"*** the dataset will look as shown in TABLE II

TABLE II

EXAMPLE OF THE SKIPGRAM APPROACH

Input1	**Input2**	**Are Neighbors?**
Fool	A	1
Fool	Thinks	1
Thinks	Fool	1
Thinks	Himself	1
Himself	Thinks	1
Himself	To	1
To	Himself	1
To	Be	1
Be	To	1
Be	wise	1

The problem now could be seen as a classification problem, where the model predicts whether two words occur simultaneously in the same context – defined by the window of size N- or not. But to do that, an extra step must be added which is negative sampling to avoid a model that always predicts 1. The authors did that by adding extra pairs of words that do not occur in the same context as shown in TABLE III. (Again, defined by the window of size N).

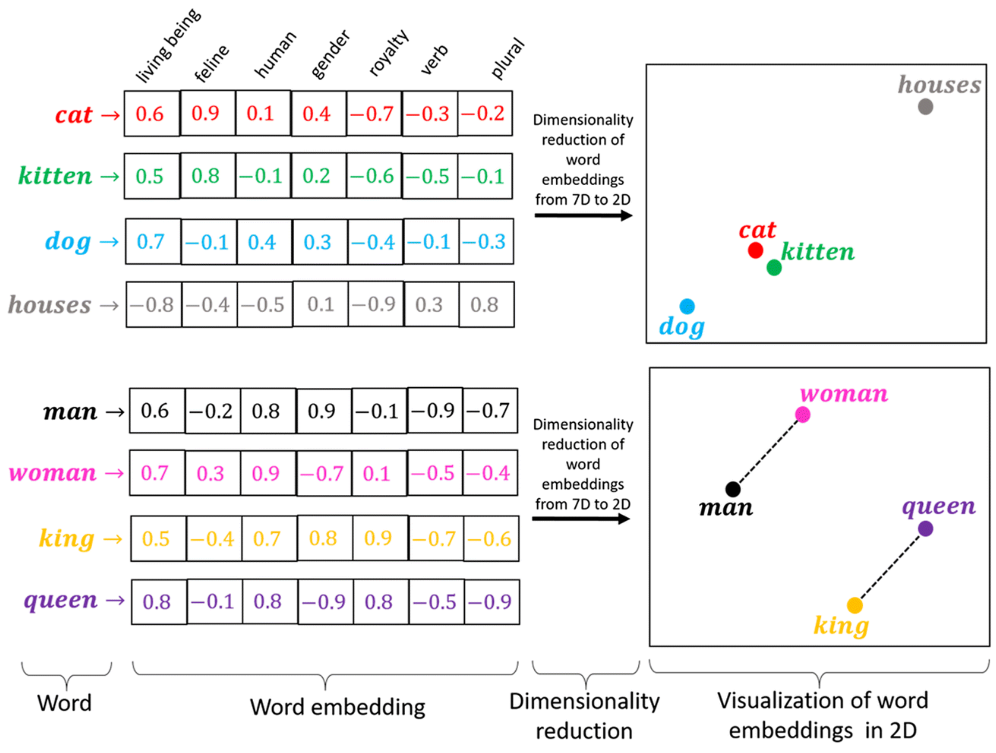
Both CBoW and skipgram performed quite well. However, according to the authors' notes, between the two architectures, CBoW was faster to train but the skipgram had more accurate results.

TABLE III

EXAMPLE OF NEGATIVE SAMPLING IN SKIPGRAM

Input1	**Input2**	**Are Neighbors?**
Fool	A	1
Fool	Thinks	1
Fool	Apple	0
Thinks	Fool	1
Thinks	Himself	1
Thinks	Cars	0
Himself	Thinks	1
Himself	To	1
Himself	Carpet	0
To	Himself	1
To	Be	1
Be	To	1
Be	wise	1
Be	Orange	0

How embeddings solve the downsides of one-hot encodings is by having a dense vector that differentiates words from each other eliminating the processing overhead, moreover, words now have some relations to each other rather than being orthogonal to each other. Figure 2 illustrates this by displaying multiple words, some of them are related, while others are not. The similarity in the vector space for cat and kitten can be seen, as well as the (lesser) similarity of animals in relation to houses. The relationship of pronouns can also be seen between man, woman, king, and queen, where man to woman has the same relationship (graphically) as king to queen.



living being
feline
human
gender
royalty
verb
plural
cat → 0.6 0.9 0.1 0.4 −0.7 −0.3 −0.2
kitten → 0.5 0.8 −0.1 0.2 −0.6 −0.5 −0.1
dog → 0.7 −0.1 0.4 0.3 −0.4 −0.1 −0.3
houses → −0.8 −0.4 −0.5 0.1 −0.9 0.3 0.8
man → 0.6 −0.2 0.8 0.9 −0.1 −0.9 −0.7
woman → 0.7 0.3 0.9 −0.7 0.1 −0.5 −0.4
king → 0.5 −0.4 0.7 0.8 0.9 −0.7 −0.6
queen → 0.8 −0.1 0.8 −0.9 0.8 −0.5 −0.9
Dimensionality reduction of word embeddings from 7D to 2D
Dimensionality reduction of word embeddings from 7D to 2D
houses
cat
kitten
dog
woman
man
queen
king
Word
Word embedding
Dimensionality reduction
Visualization of word embeddings in 2D

Figure 2: A 2D visualization of a 7D word2vec embeddings of sample words [2]

3) Contextualized Embeddings:

The effectiveness of word embeddings such as word2vec was mainly because the context was taken into consideration. However, these embeddings were ***static*** for each word in the vocabulary and could not handle homographs. In the two sentences ***"I will deposit my money in the bank"*** and ***"All the animals lined up along the riverbank"***, even though the word **"bank"** is semantically different, it would have the same embedding generated by word2vec for both sentences. A solution to this is using deep contextualized embeddings [3], which are word embeddings given to words depending on the context they are found in, hence capturing the meaning of the words. This is done by means of a two-layered bi-LSTM that takes the left and right context of the target word into consideration before assigning an embedding to it.

Embeddings from Language Models (ELMo embeddings) are made by taking the input sentence (e.g., The quick brown fox jumped over the lazy dog) then outputting the vector for each word in the sentence, as seen in Figure 3. ELMo consists of 2 layers of bi-directional LSTM, each changing the state according to the input word as well as its surrounding words (context) Figure 4.

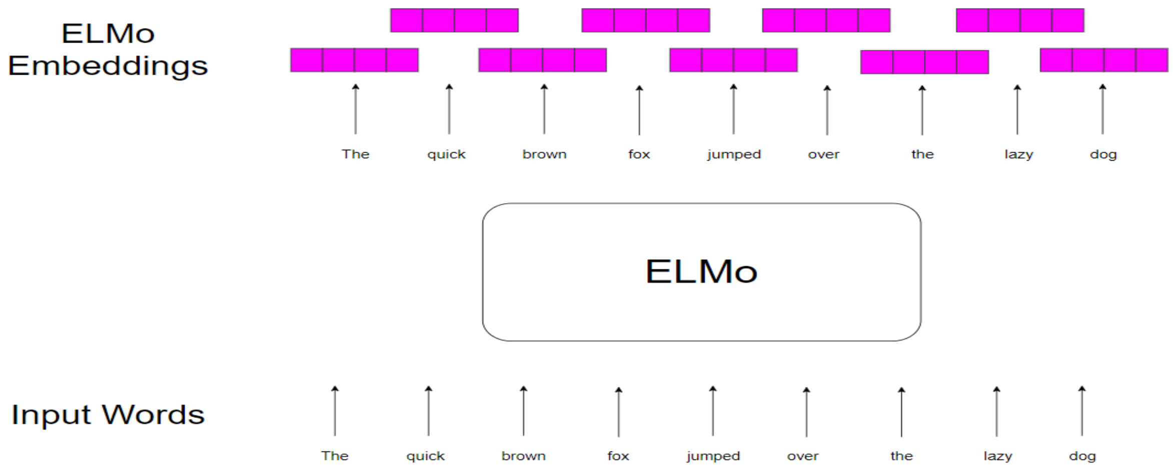


Figure 3: ELMo embedding from the highest level

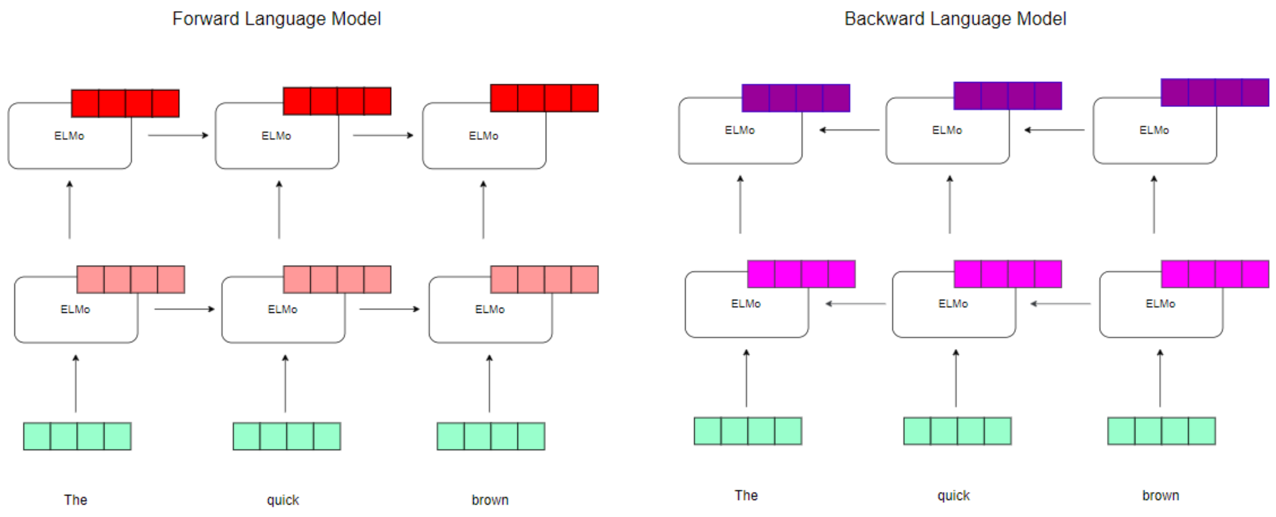


Figure 4: The hidden layers of ELMo from both directions

To produce the ELMo embeddings, the hidden layers of the word and the context are taken, concatenated, multiplied by a weight (weights differ according to the task) then summed, producing the ELMo embedding of the word, shown in Figure 5. Since this vector is a function of the word, the context and the order of the words, an embedding of a word will always change whenever the context/sentence change, solving the issue of static embeddings such as those from word2vec.

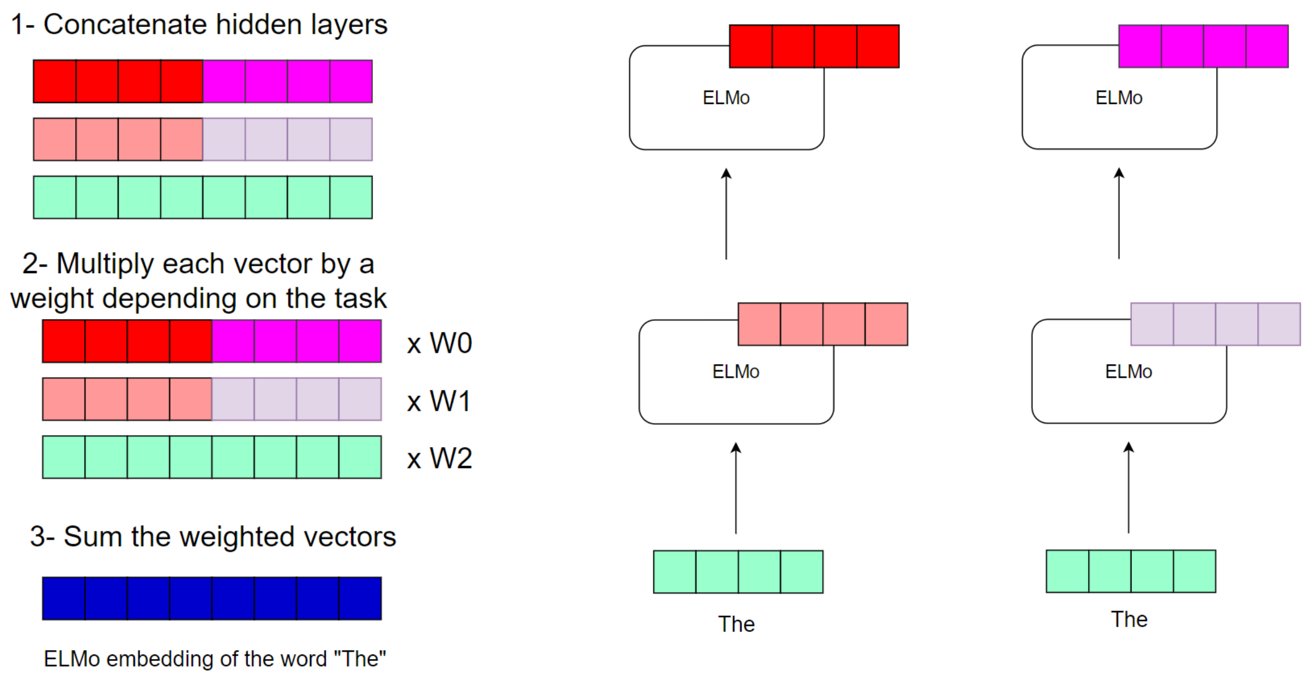

1- Concatenate hidden layers
2- Multiply each vector by a weight depending on the task
x W0
x W1
x W2
3- Sum the weighted vectors
ELMo embedding of the word "The"
ELMo
ELMo
The
ELMo
ELMo
The

Figure 5: How ELMo embedding is done under the hood

B. Information Retrieval Methodologies

Now that we are done with the aspect of representation, we go to other aspects of cQA which are Retrieval and Ranking. Since cQA systems have the goal of providing users with specific "queries" information that meets those queries, cQA can be considered information retrieval with an added ranking module.
From a scope perspective, IR systems were used to be viewed from 2 complementary views:

- Machine Centered Perspective: which deals with creating efficient and effective indexes, retrieving the information the users need quickly and a ranking "scheme" that enhances the retrieved results
- User Centered Perspective: which deals with User Experience (UX) and embedding our information about the user within the models/algorithms to personalize the retrieved results.

A sample high-level architecture of a simple IR system (Figure 6) mainly consists of the following components:

- A document/data collection stored on a filesystem/database.
- An indexer which indexes the contents of the collection generating a representation of it.
- A retrieval system that takes the "user query" which encompasses the user's need as an input.
- A ranking module to rank the results based on what the system thinks is "relevant" to the user

The retrieval system parses the user input, expands it (with synonyms and spelling variants) generating what is called the "system query" which is then queried against the collection (or rather its index) retrieving a subset of the collection.
This collection subset is then passed through a ranking module to rank the results based on what the system thinks is "relevant" to the user; Hence, from a user perspective, the ranking module was (and sometimes still is) considered the most critical component of the IR system. It is worth noting that deciding what a relevant document is, is subjective since a user could think that a specific document is relevant to a certain query, while another user thinks it is not, hence the importance of User Centered perspective mentioned earlier.
Finally, to retrieve the documents (or questions/answer pairs in the context of cQA) from the collection and decide whether they are relevant or not, a certain representation – as discussed in the previous section - should be created for them for the system to work properly. There are many methods by which this could be achieved all of which aim at making working with documents easier. This representation and any modification to the content (such as stop words removal like "the, a, an", depending on the situation) is done on both the documents in the repository and on the system query so that the comparison is fair and relevant.

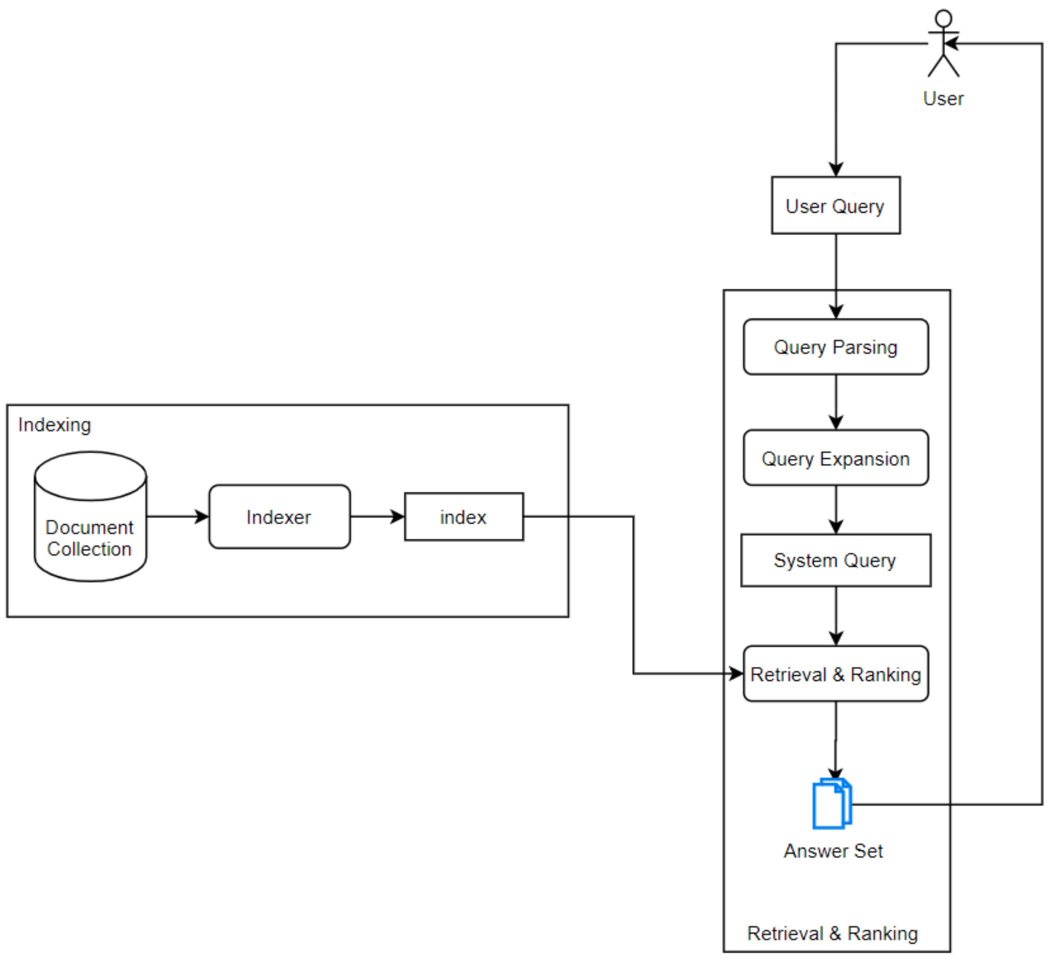
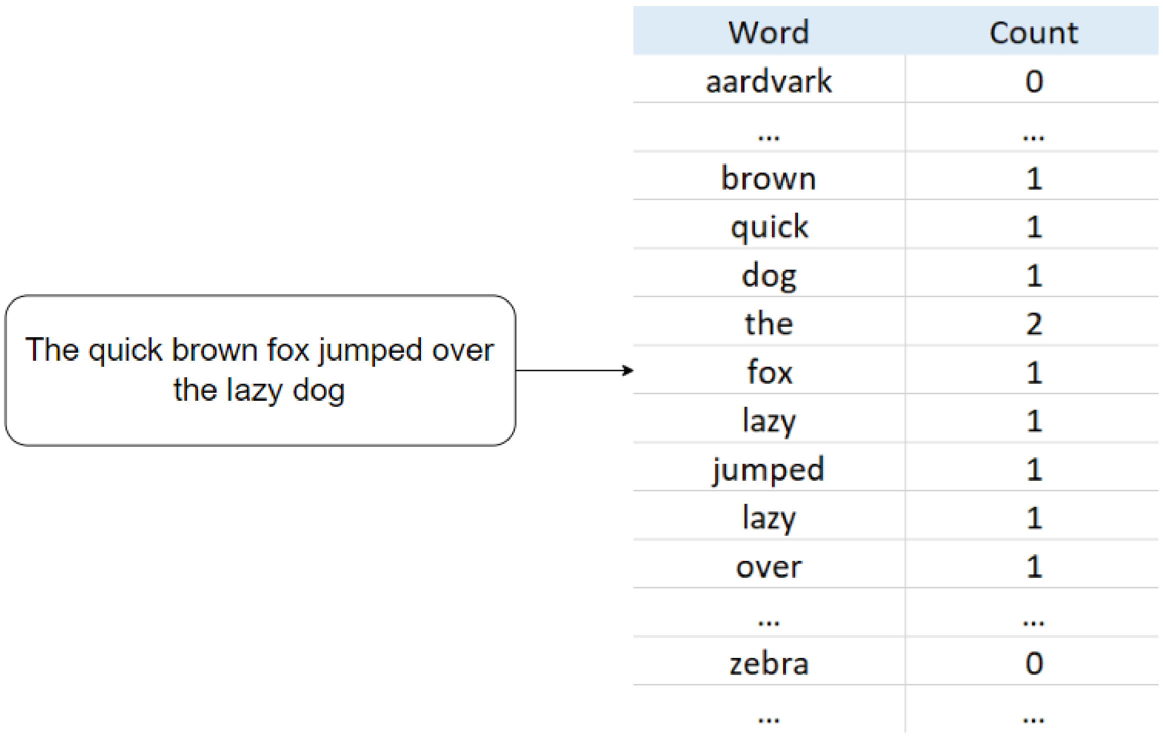


Figure 6: A High-Level Architecture of a Simple Information Retrieval System

1) Bag of Words & TF-IDF

This model was first referenced by Zellig Harris in his article "Distributional Structure" [4]. The basic idea of the Bag of Words model (BoW) is to have a vector of length ***v*** and each index of the vector corresponding to a word in the vector. So far, BoW is similar to the One Hot Encoding approach, the difference however is instead of checking whether a word exists in a text or not, the number of occurrences of the words are captured as well via a counter, capturing the frequencies of words (Term Frequency or TF) in a span of text or document as a feature. (Figure 7)
This method faced a problem of having texts - which could be similar for us humans – be orthogonal to each other. Another problem this method faced is having stop words dominate the vector representation of the text, which is mitigated by simply removing them. However, in domain specific applications, some terms are used more than others, and simply pruning them is not a solution. For that reason, a measure of how rare a term is across different spans of texts or documents is needed, which is called the Inverse Document Frequency (IDF). Coupling both TF and IDF (TF-IDF) gave a measure of how important some terms are relative to others, which made a huge improvement in Information Retrieval (IR). However, much like the One Hot encoding approach, the Bag of Words model disregards word order.
It is worth noting that both bag of words and TF-IDF could be used for document representation (subject to the downsides discussed earlier), as well as for retrieval where ranking is done by means of similarity measures.



Word	Count
aardvark	0
...	...
brown	1
quick	1
dog	1
the	2
fox	1
lazy	1
jumped	1
lazy	1
over	1
...	...
zebra	0
...	...

Figure 7: Illustration of the Bag of Words Model

2) Language Modelling Based Information Retrieval

One of the earliest general approaches of Question retrieval in IR was to estimate the probability that given a certain user query, a retrieved document would be relevant. At the time, IR systems had 2 models:

- Indexing model: that relates (or labels) a document with certain tags to be used when searching.
- A retrieval model that handles matching the user query with the documents through their tags

This 2-model structure had 2 problems:

- A parametric assumption: which states that there is an indexing model out there that fits the data set, in such a way that the tags fit the data perfectly.
- Documents are members of a an already defined class: which assumes that documents fall into mainly 2 categories "Elite" and "Non-Elite", where an 'elite' document for a given query term is a document that satisfies that query with that single term.

To bypass these 2 problems, the Language Modelling (LM) [5] approach was used, where the concept behind it is to generate a language model for each and every document in the data set, then the ranking criteria would then be on the probability that a certain query is generated given the language models of the documents, as shown in Figure 8.

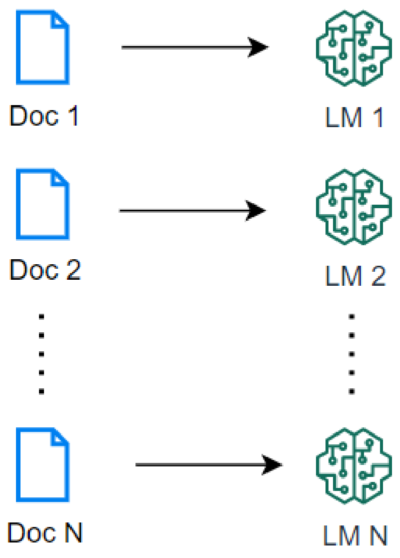


Figure 8: High Level overview of the Language Modelling Approach in Question Retrieval

The goal of the LM approach is to estimate the likelihood of a query being generated by a certain language model Md $p(Q|M_d)$, which could be formulated as:

$$p_{ml}(t|M_d) = \frac{tf(t,d)}{dl_d} \qquad (1)$$

where:

$tf(t,d)$**:** is the raw term frequency of term t in document **d**
dl_d **:** is the total number of tokens in document **d**

It is assumed that the terms within a query occur independently given a language model, which results in the ranking formula of a document 'd' to be as shown in equation 2:

$$\prod_{t \in Q} p_{ml}(t,d) \qquad (2)$$

This formula would be calculated for each document, getting the product of the generation of each term 't' in the user query 'Q'. There are several problems with this estimator; The most obvious practical problem is that we do not wish to assign a probability of zero to a document that is missing one or more of the query terms. In addition to this practical consideration, from a probabilistic perspective, it is a somewhat radical to infer that $p_{ml}(t|M_d) = 0$. I.e., the fact that we have not seen a certain term before does not infer that it is impossible to see it in the future. Instead, an assumption is made that a "not so occurring term" is possible to occur, but with a probability no bigger than what would be expected by chance in the collection (i.e. $\frac{c_{ft}}{cs}$) where c_{ft} is the raw count of term t in the collection and cs is the raw collection size or the total number of tokens in the collection. The value of this assumption is that it provides us with a more reasonable distribution and circumvents the two practical problems mentioned.

3) Translation Based Information Retrieval

When a person tries to search for a question (or a document) in a retrieval system like google or Quora, it could be thought that what he/she is really doing is imagining the "perfect" document in his/her head then try to distill this document into a term or more formulating the query being input to the system. The translation-based approach is based on an idea which treats this "perfect" document being imagined by the user (i.e. the document with the highest relevance/matching score to the user) and the input query as parallel texts, and as such, the query could be treated as a translation from that perfect document being imagined, shown in Figure 9.

At the time, the authors [6] tried to explore this approach with word-based translation models, not because it is better than the traditional approaches such as BoW based or LM-based, but because it proved to be practically convenient and that this approach has much potential to be explored.

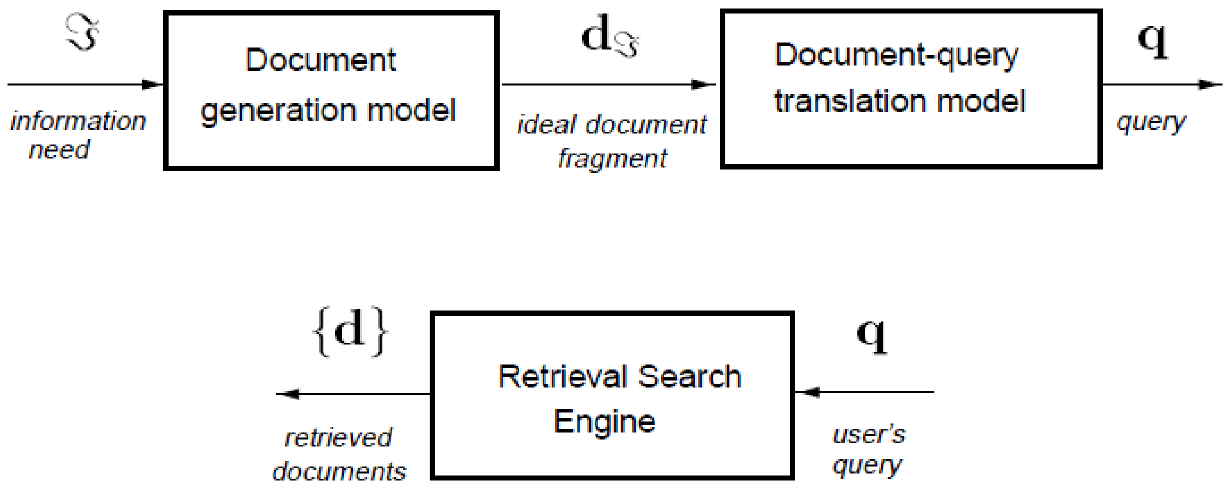


Figure 9: Model of query generation and retrieval [6]

The idea behind the translation-based approach is summarized in Figure 9, which summarizes the model of query generation to the following points: -

- The user needing an information $\boldsymbol{\zeta}$
- From this need, a user generates a perfect document part $\boldsymbol{d_\zeta}$
- The user then chooses a set of words from the generated document parts, building the query $\boldsymbol{q}$ from the chosen words.

Now, the task of the retrieval system is to find the a posteriori probability that a document $\boldsymbol{d}$ satisfies a query $\boldsymbol{q}$, given the query and the user's general preferences U, given by $p(d|\boldsymbol{q}, U)$ which could be decomposed into:

$$p(d|\boldsymbol{q}, U) = \frac{p(\boldsymbol{q}|\boldsymbol{d}, U)\, p(\boldsymbol{d}|U)}{p(\boldsymbol{q}|U)} \quad (3)$$

The term $p(\boldsymbol{q}|U)$ in the denominator could be ignored, since it is fixed for a given a query and a given user, so it will not affect the ranking. Hence a relevance metric is defined as

$$\rho_{\boldsymbol{q}}(\boldsymbol{d}) = p(\boldsymbol{q}|\boldsymbol{d}, U)\, p(\boldsymbol{d}|U) \quad (4)$$

Equation (4) highlights the decomposition of relevance into two terms: a query-dependent term $p(\boldsymbol{q}|\boldsymbol{d}, U)$ measuring the proximity of d to q, and second, a query-independent or "prior" term $p(\boldsymbol{d}|U)$, measuring the quality of the document according to the user's general preferences and needs. Though in this work [6] the prior term is taken to be uniform over all documents, in real-world retrieval systems, the prior term will be crucial for performance, and for adapting/fitting to the user's needs and interests.

C. Attention Mechanism

Attention is a mechanism that allows a system to focus its "attention" on a part of the input that is important to accomplishing a specific task or subtask. Attention was implemented as a method to enhance machine translation [7].
A very good graph showing this is Figure 10 where a French sentence is translated to English. In the figure, a matrix of attention weights is formed between words, where when trying to output a certain English word, higher weights (attention) is given to certain French words denoted by the white squares, while the other relatively unimportant parts are given very low weights.

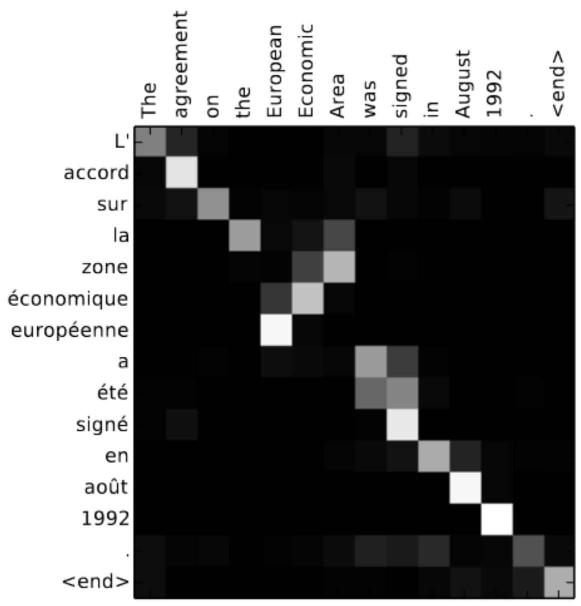


Figure 10: A sample alignment example from French to English **[7]**

Machine Translation used to be done using a pipeline of statistical components, then a neural approach proposed by [8], [9] and [10] where a single neural network is trained and used to output the translation from a source language to the target language. This approach solved the classical problem of several sub-components, because only one model is trained and fine-tuned, whereas in the classical approach, each sub-component had to be trained and fine-tuned separately.
In [7], the authors propose a novel architecture to the conventional neural approach proposed by [8], [9] and [10], where the architecture was made up of an encoder RNN and a decoder RNN components. The encoder takes the input words and, in the end, encodes the entire sentence in a vector that is then passed to the decoder to decode, and output the translation, shown in Figure 11.

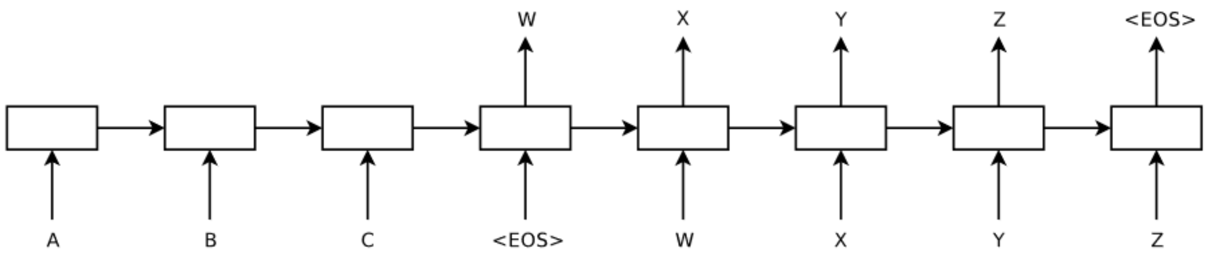


Figure 11: Machine Translation using the "Encoder-Decoder" Architecture

The problem with this approach is that if the input sentence is long - longer than the longest sentence in the training dataset- performance deteriorates drastically, and the authors of [7] argue that this is due to having the vector passed to decoder components of fixed length, which cannot capture long term dependencies very well.
What is proposed in [7] is a method for the model to align and translate jointly, by means of soft searching the words within the input sentence where the most relevant data lies. The model then predicts the words in the output sentence based on the context vectors of the soft-searched words along with any previously generated (translated) words.
Based on this suggestion and the fact that a fixed length context vector is replaced with a vector for each word, the encoder-decoder is freed from compressing all the information lying within the input sentence from a single fixed length vector and enabled to generate the target words in a dynamic fashion depending on each word translated.

Formally, in the traditional Encoder-Decoder architecture, the encoder gets the input $X = (x_0, x_1 \dots x_T)$ and compresses that into vector c where:

$$c = q(\{h_1, h_2 \dots h_{T_x}\}) \quad (5)$$

where h_t is the hidden state at time t, and it is calculated as:

$$h_t = f(x_t, h_{t-1}) \qquad (6)$$

and f and q are non-linear functions.
The decoder then takes the context vector c and any translated target words $\{y_0, y_1, y_2 \dots y_{t-1}\}$ as input and outputs the most probable word y_t where:

$$p(y) = \prod_{t=1}^{\mathrm{T}} p(y_t|\{y_0, y_1, y_2 \dots y_{t-1}\}, c) \quad (7)$$

where each conditional probability at time t in the RNN could be derived to:

$$p(y_t|\{y_0, y_1, y_2 \dots y_{t-1}\}, c) = g(y_{t-1}, s_t, c) \quad (8)$$

Where g is a non-linear function that outputs the probability of y_t at time t, and s_t is the hidden state of the decoder at time t.

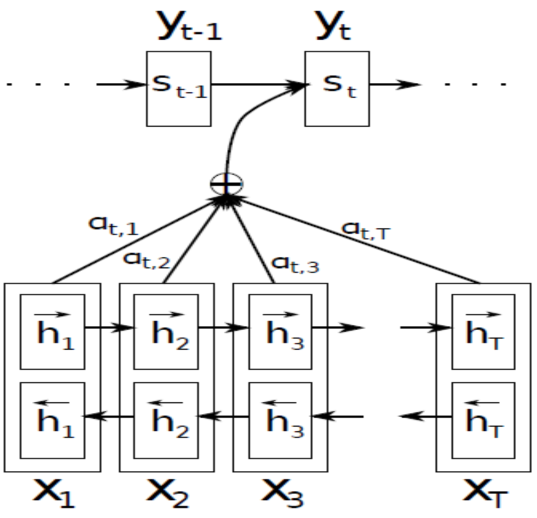


Figure 12: Proposed Architecture to Support Attention Mechanism [7]

In the new architecture (Figure 12), the authors model the conditional probability on the decoder side as follows: -

$$p(y_i|y_1, y_2 \dots y_{i-1}, x) = g(y_{i-1}, s_i, c_i) \quad (9)$$

where s_i – like the traditional architecture – is the hidden state of the RNN at time i, and is computed by the formula

$$s_i = f(s_{i-1}, y_{i-1}, c_i) \qquad (10)$$

where c_i is a distinct context vector used to predict the target word y_i. This context vector is dependent on the annotations $(h_0, h_1 \dots h_T)$ which the encoder maps the words within the input sentence to. Each vector h_i captures information about the whole sentence, with special focus (or "attention") to one or more words. Each of the context vectors hence are computed by

$$c_i = \sum_{j=0}^{\mathrm{T_x}} \alpha_{ij} h_j \qquad (11)$$

where α, the weight given to each annotation, and is calculated by: -

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{\mathrm{k=1}}^{\mathrm{T_x}} \exp(e_{ik})} \qquad (12)$$

where:

$$e_{ij} = a\left(s_{i-1}, h_j\right) \qquad (13)$$

is the alignment model which scored how relevant input j is to the target output i and the model a is a feed forward network that is trained jointly with the RNN. [7] shows these equations in action at the Decoder side.

It is also noted that unlike the traditional approach of having a uni directional RNN at the decoder side and/or encoder side, this approach uses a Bi-directional RNN in order to capture not just the information within the preceding words, but also the following ones. The resulting annotation vectors are then concatenated together – both from the forward and backward RNN – and used to calculate e_{ij} which in turn used to calculate α_{ij} with the trained alignment model, which is used to calculate the context vectors c_i which is used to calculate s_i which is passed to the non-linear function calculating the probability distribution of the word y_t.

In summary, the attention mechanism is an approach in which the model gives weights to each lexical input – be it words or sentences – and focuses (or pays attention) to the inputs with the highest weights when generating the target output(s). How Attention is beneficial in cQA is that Transformers [11] (discussed in section E) is based on this mechanism, which is widely used to produce context-based embeddings in a fast manner.

D. Transformers

"Attention Is All You Need" [11] is a paper published in December 2017 by Google Research and Google Brain, and it was one of the most revolutionizing papers to date. Up till the time before this paper was published, the NLP field was dependent on Recurrent and Convolutional models. Of course, the state of art results at the time was not due to the vanilla versions of these recurrent or convolutional architectures, but more sophisticated variants of them, mainly Long Short-Term Memory Networks (LSTMs) [12] and Gated Recurrent Units (GRUs) [10].

These variants mainly address the issue of long-term dependencies since vanilla RNNs cannot "remember" contexts the larger the input text, so what these variants add are "Gates" that enable the network to have the ability to keep some contexts and forget others at will through training. Also, the addition of attention mechanisms further enhanced the ability of these networks. The attention mechanism enables the network to give weights to the input, focusing its "attention" to relevant part of the inputs as seen in the previous section.

Although LSTMs and GRUs did a good job, RNNs in general suffered from being unable to be trained in parallel, because the layers are recurrent in time which made the model sequential by nature, rendering all technological advancements in Graphical Processing Units (GPUs) and parallel training useless. To bypass this dilemma, the authors of [11] threw away recurrence all together in favor of using "Self-Attention" which enabled parallel training as well as enhancing performance. Although the proposed approach does not use recurrent neural networks, the high-level architecture of having encoders and decoders remain, as can be seen in Figure 13.

1) The Encoder Stack

The encoder stack consists of $N_x = 6$ encoders (Could have been more, but this was the choice the authors went with at the time of publishing), each layer consists of two sublayers, the first is a multi-headed self-attention sublayer and the second is a point-wise feed forward network, as could also be seen from Figure 13, a residual connection has been put after each sublayer, followed by a layer normalization step. So, the output becomes $LayerNorm(x + Sublayer(x))$ where $Sublayer(x)$ is the functionality being implemented by the sublayer (i.e., Multi-headed attention or the point-wise feed forward connection). Also, to facilitate these residual connections, the authors fixed the size of output of all layers (embeddings included) to $d_{model} = 512$.

2) The Decoder Stack

Like the encoders, $N_x = 6$ identical decoders were employed, each decoder has the same multi-headed self-attention and point-wise feed forward network with the residual connections and the fixed output size d_{model}. So far, everything is identical, except for a third sub-layer that performs multi-headed attention over the output of the encoder. Also, the self-attention layer in the decoders is modified so that positions do not attend to subsequent positions (peek into future tokens, in colloquial terms). Due to this modification and the fact that outputs are offset by one from the inputs ensures that outputs are solely dependent on past outputs and nothing more.

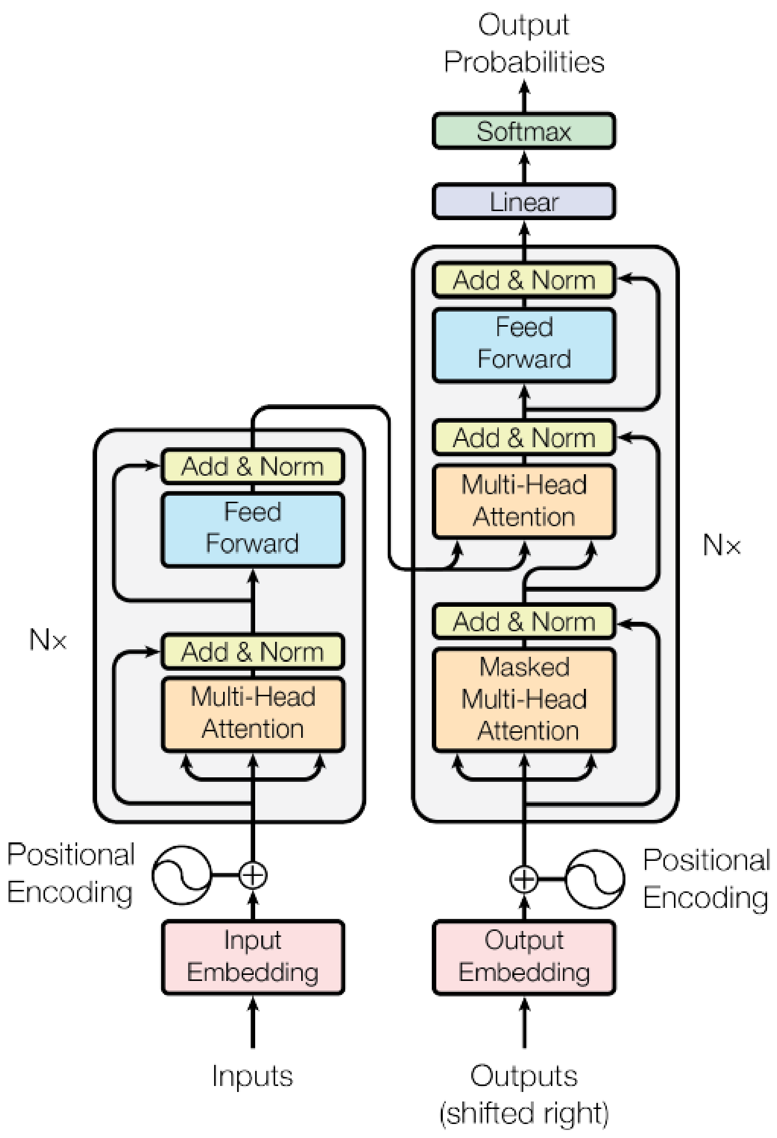


Figure 13: The Transformer Model Architecture [11]

Before proceeding with equations, it is worth explaining "attention" in the context of [11]. The authors explain attention in a very intriguing manner which is "A function that maps a query and a set of key-value pairs to an output". One can intuitively think of the query as the word in question in terms of "How much attention should be given to this word?", and in similar fashion, one could think of the key-value pairs as the representations – not to be confused with embeddings – of the words within the input sentence, similar to query but in another space.
The output "How much attention should be given" is a weighted sum of the "value" vectors, where the weights assigned are calculated by a "compatibility function" using the query and key vectors. The formula (shown in Figure 14) for the attention is given by

$$Attention(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \qquad (14)$$

where Q is the query matrix, K is the Key matrix and V is the value matrix, each of these matrices are multiplied by the embedding of each of input of the encoder to produce the resulting query, key and value vectors specific to that input word. The Q, K and V matrices are trained jointly with the model.

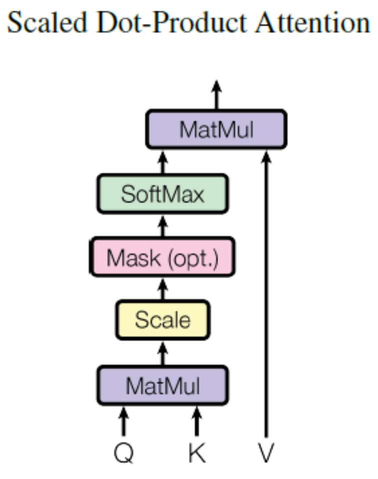


Figure 14: Scaled Dot-Product Attention [11]

The authors also mentioned that there are different attention functions, most commonly are the “Dot-Product Attention” which was used in this paper, and the “Additive Attention”. Unlike the additive attention used in [7] which calculates the compatibility function through a single-layered feed-forward neural network, the Dot-Product attention function uses dot product between matrices to achieve this purpose (Figure 14) The only difference between the “traditional” dot-product approach and that of [11] is the addition of the scalar $\frac{1}{\sqrt{d_k}}$ (hence the Scaled part in the name). The reason for this is that the authors suspect that for large values of d_k the dot product becomes too large, it pushes the SoftMax function into regions with very small gradients. However, at small values of d_k both Dot-Product Attention and Additive Attention perform similarly.

The authors also mentioned that they found it useful to calculate the attention $h = 8$ times, each projecting the query, key and value vectors with different learned linear projections W_i. The output in this case is dependent on the concatenated sets of h vectors which is then projected yet again resulting in the final output. Concretely the formula is: -

$$MultiHead(Q, K, V) = Concat(head_1, head_2 \dots head_h)W^O \quad (15)$$
$$where\ head_i = Attention\left(QW_i^Q, KW_i^K, VW_V^V\right) \quad (16)$$

This Multi-headed approach allows the model to capture different representations subspaces at each input position, that could not have been achieved using a single-headed attention.

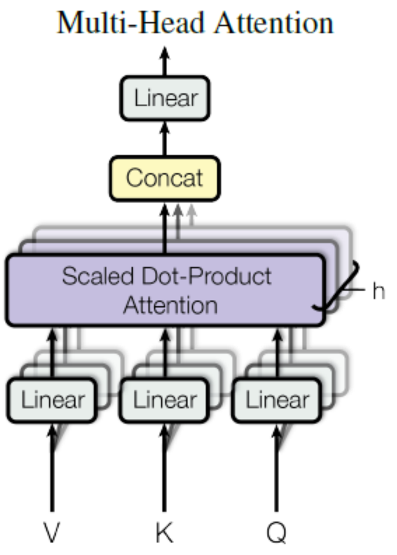


Figure 15: multi-Headed attention consisting of h attention layers running in parallel [11]

So, in summary the Attention mechanism is used in the Transformer architecture in 3 ways: -

- In the self-attention component within the encoder, allowing the encoder to attend to all words in the input. In this case all query, key and value vectors come from the previous encoder layer.
- In the "Encoder-Decoder" attention in the decoders, allowing the decoders to attend to the input from the encoders. In this case, the query vectors come from the previous decoder layer whereas the key and value vectors come from the encoder stack.
- In the self-attention component within the decoder, allowing the decoder sublayers to attend to the output of the previous decoder layers. The authors note that any inputs in positions following the current position being calculated should be blocked, in order to preserve the auto-regressive property of the decoder. This blocking mechanism is implemented by setting all inputs to the scaled dot-product function to $-\infty$ coming from illegal connections, effectively setting the softmax output of these inputs to 0.

There is one last thing to note in and that is when adopting this new approach, rather than the traditional "recurrent" one, the sequential properties of the inputs have been stripped. So how would the model know the position of each word in the sentence? The answer to that is "Positional Encoding", which uses sine and cosine function with different frequencies:

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right) \quad (17)$$

$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_{model}}}}\right) \quad (18)$$

Where, pos is the position of the word and i is the dimension. So, each dimension of the positional encoding vector corresponds to a sinusoid. Note also that the positional encoding has a dimension length similar to that of the word embeddings layer on the encoder side, which means that they can be summed.

E. Bidirectional Encoder Representations from Transformers (BERT)

BERT [13] is a transformer-based architecture, where only the Encoder part of the transformer is used (some architectures use the decoder only like [14]). What the authors of [13] are trying to accomplish is to enhance current schemes of pre-training [15] a model that with little modification could be used in other tasks and achieve high level of performance.

The authors mentioned the several approaches for transfer learning, one of which is feature based approach and fine-tuning based approach. Feature based Approach is similar to that of ELMo [14] where the model is pretrained on a task generating representations of the words, then developing task specific architectures which uses these representations, which is obviously not the best due to the need to modify the architecture. Fine-tuning based approach however is similar to that of GPT [14], where the model is originally trained on a task then have the entire model fine-tuned on the downstream task. It is worth noting that both these approaches share the same pre-training objective, which is using unlabeled text to learn language representations.

Although the authors of [13] will be adopting the fine-tuning-based approach [15], they argue that existing techniques limit the potential of pre-trained representations, due the fact that the current architectures use a left-to-right approach in training, which the authors think is suboptimal in sentence level tasks and could be dangerous in token level tasks like Question Answering, where it is imperative to incorporate contexts from both directions.

To address this, the authors improve fine-tuning approaches by introducing the encoder only architecture which is free from the unidirectional constraints of previous architectures by using "Masked Language Model" (MLM) as a pre-training objective. MLM is a task where the model is given a sentence with some of the tokens masked. The model must figure out what these masked tokens (in the form of embeddings) are. One other task BERT is trained on is "Next Sentence Prediction" where the model is given two sentences, and the model has to predict whether these sentences come after each other or not.

These two pre-training objectives combined enables the model to capture information from both left and right contexts, as well as capture sentence level insights. This approach is different than that of GPT [14] since it takes both left and right contexts and different from ELMo which trains two bi-directional LSTM networks independently from each other, then shallowly concatenate the output representations. Due to these objectives, BERT achieves state of the art results in eleven - token level and sentence level - NLP tasks.

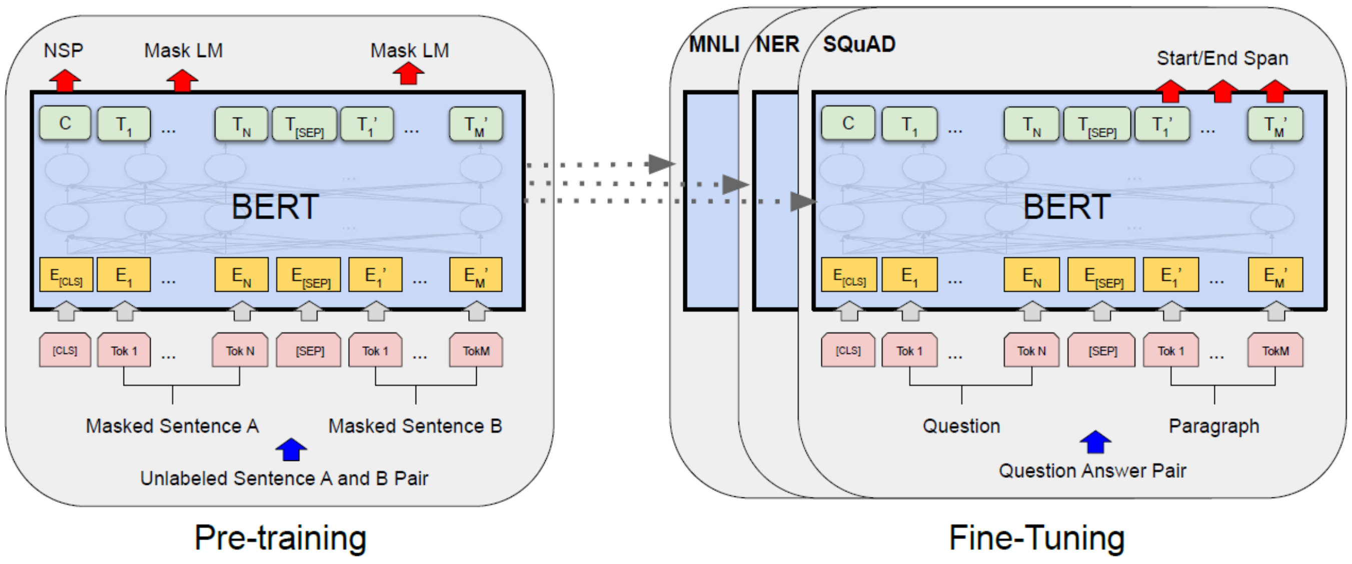


Figure 16: The pre-training and fine-tuning stages of BERT [13]

The overall pre-training and fine-tuning steps of BERT are illustrated in Figure 16. The pretraining step consists of two phases: -

- Pretraining: where the model is pretrained on several tasks
- Fine-tuning: where the model is initialized with the pre-trained parameters then fine-tuned with the labeled data of the downstream task.

Two things to notice in Figure 16, first is that each downstream task has a separate fine-tuned model and second is that BERT has the same architecture across the different tasks.
The model architecture the authors employed is very similar to the one by [11], except that BERT only makes use of the encoder part. The number of layers is denoted as L, the hidden size as H and the number of self-attention heads as A. The paper reports the results of two model sizes.

$$BERT_{Base}(L = 12, H = 768, A = 12, Total\ Parameters = 110M)$$
$$BERT_{Large}(L = 24, H = 1024, A = 16, Total\ Parameters = 340M)$$

Also, to have BERT be able to handle a variety of downstream tasks, the authors designed BERT to be able to handle both a single sentence and a pair of sentences as input. In the paper, they refer to "sentence" as a span of text that could be a single linguistic sentence or more. Moreover, they refer to "sequence" as one or more "sentences" packed together. BERT also uses Word Piece embeddings rather than the regular word embeddings, with a 30,000 token vocabulary, where the first token in every sequence is the special token [CLS]. It is worth noting that the final hidden state representing this special token is used as the overall "sequence" representation for classification tasks.
When feeding "sequences" to the BERT as input, the methods by which the model knows the separation of "sentences" is by: -

- Using another special token [SEP] between each sentence.
- Adding learned embeddings to each token in the input indicating whether said token belongs to sentence A or sentence B.

As shown in Figure 16, the authors denote the input embeddings as E, the final hidden vector representing the [CLS] token as C and final hidden vector for the i^{th} input as T_i. Also, as shown in Figure 17, each token representation is constructed by summing: -

- The corresponding token embedding
- The corresponding segment embedding
- The corresponding position embedding

Now let's talk in detail about the pre-training phase. The first task BERT is trained on is Masked Language Model. This task was the task of choice because it trains the model on both left and right contexts and by that method it learns a deep bi-directional representation of words. (Again, unlike the relatively shallow representation employed by [3]).

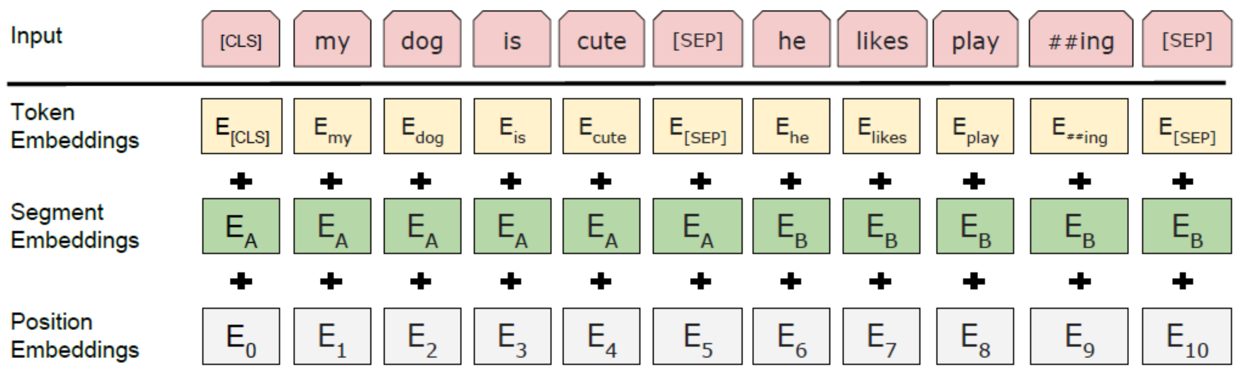

Input
[CLS]
my
dog
is
cute
[SEP]
he
likes
play
##ing
[SEP]
Token Embeddings
E[CLS] E my E dog E is E cute E[SEP] E he E likes E play E##ing E[SEP]
Segment Embeddings
E A E A E A E A E A E A E B E B E B E B E B
Position Embeddings
E 0 E 1 E 2 E 3 E 4 E 5 E 6 E 7 E 8 E 9 E 10

Figure 17 BERT input representation [13]

The MLM task goes in this sequence: -

- An input sequence is fed to BERT
- Any token has a 15% chance of getting masked
- In order to avoid a mismatch between pre-training and fine-tuning by overfitting the model on the literal token $[MASK]$, only 80% of the to-be-masked tokens are replaced by the token $[MASK]$, 10% remains unchanged and 10% are replaced by another random token.

Next, the model has to predict the masked words from the context, not the entire input sequence. Then, the second pre-training task is Next Sentence Prediction (NSP) task, where two sentences are fed to BERT, and the model has to predict whether the second sentence comes after the first or not. This task is important because many tasks - such as QA and Natural Language Inference (NLI) - are dependent on the model understanding the relationships between sentences, which normally could not be captured with traditional LM tasks.
Finally, the training data used in the pretraining tasks, are: -

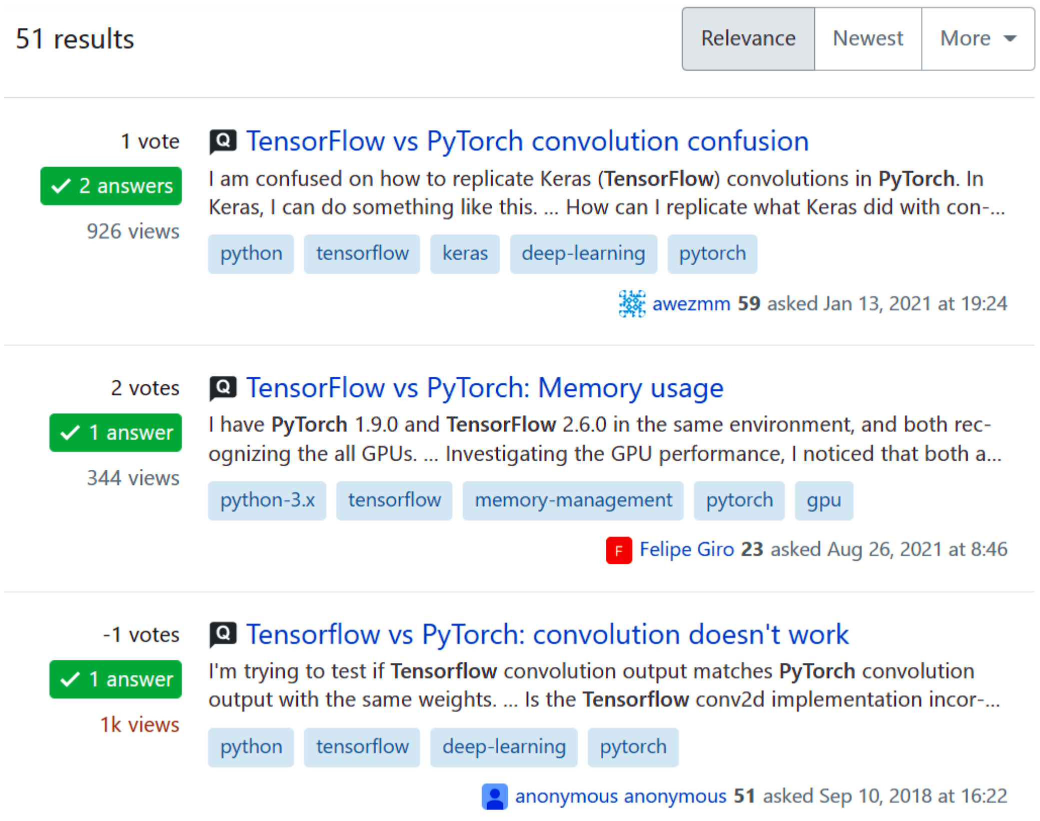
- Books Corpus: A dataset of books comprising of 800M words
- English Wikipedia: 2,500M words.

BERT has achieved state of the art metrics in several downstream NLP tasks, using relatively less data than [16]. However, it is not without its flaws, because even though BERT (or Transformers in general) has the potential of learning long-term dependencies within the text (unlike RNNs), it is still limited by the fact that only a fixed amount of input can feed at a time. This limitation creates disruptions in learning dependencies between sentences fed at different times, which is solved later by using auto-regressive architectures like [17] and [18].
Even though limited context is a huge downside, in practice, the length of words being input at a time (input batch) usually do not reach that limit, coupled with how has many implementations have been made available through the hugging face library [19] which is widely adoption by the AI community, the upsides greatly exceed the downsides of BERT. Moreover, there variants of BERT which underwent knowledge distillation ([20], [21] and [22]) such as distill BERT [22] which is a smaller, faster BERT with an accuracy that is not so far off the original BERT.

F. Data Centric AI (DCAI)

As the industries learn the value of AI, use cases are starting to get more complex and the requirements of the AI-based systems are becoming more strenuous. The usual trend was to either invent a new complex model to handle the use case or put together an ensemble of relatively simpler models to do the job. This approach while showed results in previous years, is now showing its limits, since ensembles of complex models are not only difficult to train, but hard to maintain and operationalize. As such, the new trend of Data Centric AI [24] and [25] came to be, where the approach is not to develop complex models to solve the cases, but to properly label and preprocess the data to be of higher quality, enabling relatively simple models to do a better job. The goal of DCAI is to have a systematic approach in engineering the data in order to facilitate the learning to the models.

What DCAI aims at is to unify and agree on the definition of categories/classes and what defines each class within a category. In the context of cQA, there should be an agreement and a methodology that marks 2 or more questions as relevant or not. An example to this is in [Figure 18] where I wanted to check what the differences were between tensor flow [26] and pytorch [27], so *"tensorflow vs pytorch"* is queried hoping to find a question that asks about a comparison between the 2 libraries. To approach this problem with this example from a systemic approach annotator need to agree on what defines a relevant question to my query, as in "Is any comparison considered relevant?", and if not, then what is the degree of granularity should be considered relevant. For example, the second question "TensorFlow vs PyTorch: Memory Usage" talks about the comparison between TensorFlow and pytorch but from a memory standpoint only, hence, annotators will have to agree on whether such aspects of comparison are enough to make the retrieved results relevant or not.



1 vote
2 answers
926 views
TensorFlow vs PyTorch convolution confusion
I am confused on how to replicate Keras (**TensorFlow**) convolutions in **PyTorch**. In Keras, I can do something like this. ... How can I replicate what Keras did with con-...
python tensorflow keras deep-learning pytorch
awezmm **59** asked Jan 13, 2021 at 19:24

2 votes
1 answer
344 views
TensorFlow vs PyTorch: Memory usage
I have **PyTorch** 1.9.0 and **TensorFlow** 2.6.0 in the same environment, and both recognizing the all GPUs. ... Investigating the GPU performance, I noticed that both a...
python-3.x tensorflow memory-management pytorch gpu
Felipe Giro **23** asked Aug 26, 2021 at 8:46

-1 votes
1 answer
1k views
Tensorflow vs PyTorch: convolution doesn't work
I'm trying to test if **Tensorflow** convolution output matches **PyTorch** convolution output with the same weights. ... Is the **Tensorflow** conv2d implementation incor-...
python tensorflow deep-learning pytorch
anonymous anonymous **51** asked Sep 10, 2018 at 16:22

Figure 18 A sample query to stack overflow

3 CONCLUSIONS

In Summary, Community Question Answering has a vast array of techniques (both mentioned in this paper and not) that tries to increase relevancy metrics (such as MAP), where the retrieved documents are relevant to the query input to the system, However, many of the older techniques lacked precision in the Ranking part where the documents are sorted/ranked from most to least relevant. To solve this, better representations of the query and documents are required, where a similarity function could be easily applied, and the similarity score can be used in ranking. One such representation scheme is to use Embeddings to map the meaning/semantics of the sentence or document to a vector space. This mapping eases the ranking task by enabling the use of similarity functions, where the vectors of retrieved documents that are closer to the vector of the query are more relevant.

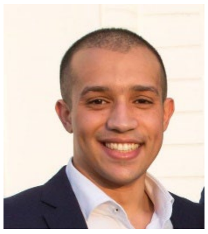
However, embeddings alone are not enough, because in order to make use of good embeddings, data should be cleaned and annotated properly to convey the correct classification and meaning, otherwise the embeddings would not be of much help. This is where DCAI comes to play, where methodological, systematic way of engineering the data to be coherent to enhance the training of the model and have the target categories - hence, the relevancy metrics - consistent.

REFERENCES

[1] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," arXiv preprint arXiv:1301.3781v3, 2013
[2] Swati Meena (2020). Training Word2vec using genism, Available from https://swatimeena989.medium.com/training-word2vec-using-gensim-14433890e8e4

[3] M. E. Peters *et al.*, "Deep contextualized word representations," arXiv preprint arXiv:1802.05365v2, 2018.
[4] Z. S. Harris, "Distributional Structure," WORD, 10:2-3, 146-162, DOI: 10.1080/00437956.1954.11659520
[5] J. M. Ponte and W. B. Croft, "A Language Modeling Approach to Information Retrieval," in proc. of the 21st annual international ACM SIGIR conference on Research and development in information retrieval, pp. 275–281, August 1998.
[6] A. Berger and J. Lafferty, "Information Retrieval as Statistical Translation," in proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval, pp.222–229, August 1999.
[7] D. Bahdanau, K. Cho, and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate," arXiv preprint arXiv:1409.0473v7, 2014.
[8] N. Kalchbrenner and P. Blunsom, "Recurrent Continuous Translation Models," In Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing, pp.1700–1709, Seattle, Washington, USA, 2013. Association for Computational Linguistics.
[9] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to Sequence Learning with Neural Networks," arXiv preprint arXiv:1409.3215v3, 2014
[10] K. Cho *et al.*, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," arXiv preprint arXiv:1406.1078v3, 2014.
[11] A. Vaswani *et al.*, "Attention is All you Need," In Advances in Neural Information Processing Systems, pp.6000–6010, 2017.
[12] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," Journal of Neural Computation, vol.9, no.8, pp.1735-1780, 1997.
[13] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," arXiv preprint arXiv:1810.04805, 2018.
[14] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving Language Understanding by Generative Pre-Training," 2018. [Online]. Available: https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf
[15] J. Howard and S. Ruder, "Universal Language Model Fine-tuning for Text Classification," arXiv preprint arXiv:1801.06146v5, 2018.
[16] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language Models are Unsupervised Multitask Learners," 2019. [Online]. Available: https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf
[17] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov, "Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context," arXiv preprint arXiv:1901.02860, 2019.
[18] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, "XLNet: Generalized Autoregressive Pretraining for Language Understanding," arXiv preprint arXiv:1906.08237, 2019.
[19] T. Wolf *et al.*, "Hugging Face's Transformers: State-of-the-art Natural Language Processing," arXiv preprint arXiv:1910.03771, 2019.
[20] G. Hinton, O. Vinyals, and J. Dean, "Distilling the Knowledge in a Neural Network (2015)," arXiv preprint arXiv:1503.02531v1, 2015.
[21] X. Liu, X. Wang, and S. Matwin, "Improving the Interpretability of Deep Neural Networks with Knowledge Distillation," arXiv preprint arXiv:1812.10924, 2018.
[22] J. Gou, B. Yu, S. Maybank, and D. Tao, "Knowledge Distillation: A Survey," arXiv preprint arXiv:2006.05525, 2020.
[23] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distil BERT, a distilled version of BERT: smaller, faster, cheaper and lighter," arXiv preprint arXiv:1910.01108, 2019.
[24] B. Koch, E. Denton, A. Hanna, and J. G. Foster, "Reduced, Reused and Recycled: The Life of a Dataset in Machine Learning Research," arXiv preprint arXiv:2112.01716, 2021.
[25] Data Centric AI Web Site: https://datacentricai.org/ (accessed 26 August 2021)
[26] M. Abadi, *et al.*, "TensorFlow: A system for large-scale machine learning," arXiv preprint arXiv:1605.08695, 2016.
[27] A. Paszke *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," arXiv preprint arXiv:1912.01703, 2019.
[28] Glossary of Scientific AI Terms, Available from https://aiinarabic.com/glossary/

BIOGRAPHY



Ahmed Zaazaa
Ahmed received the B.Sc. in Communications and Computer from the Faculty of Engineering, Cairo University, Cairo, Egypt. Ahmed has been working in IBM for 7 years, 6 of which in the field Artificial Intelligence and Cognitive Solutions.

Prof. Mohsen A. A. Rashwan
Professor Mohsen Rashwan received the B.Sc. and M.Sc. degrees in Electronics and Electrical Communications from the Faculty of Engineering, Cairo University, Cairo, Egypt, another M.Sc. degree in systems and computer engineering from Carleton University, Ottawa, ON, Canada, and the Ph.D. degree in electronics and electrical communications from Queen's University, Kingston, ON, Canada.

Dr. Ossama Emam
Dr. Ossama Emam received B.Sc., M.Sc. and Ph.D. degrees in Systems and Computer Engineering from the Faculty of Engineering, Cairo University, Cairo, Egypt. Dr. Ossama has more than 36 years of experience in NLP and ML technologies. Dr. Ossama has been leading the IBM Cairo Human Language Technologies since 1986. And in 2008, he became the Chairperson of the MEA invention Development Team.

Arabic Abstract

الاجابة على اسئلة المجتمعات: دراسة استقصائية عن منهجيتها

أحمد زعزع*[1]، محسن رشوان*[2]، أسامة إمام*[3]

قسم هندسة الاتصالات والالكترونيات، كلية الهندسة، جامعة القاهرة، جبزة، مصر*

[1]azaazaa@eg.ibm.com

[2] mrashwan@rdi-eg.ai

[3]ossama.emam1@ibm.com

ملخص

تستعرض هذه الورقة تطور تضمين الكلمات جنبًا إلى جنب مع المنهجيات المستخدمة في الإجابة على أسئلة المجتمع، وكيف تستخدم هذه المنهجيات تضمين الكلمات لتحقيق مقاييس أداء أعلى. تناقش الورقة أولاً نمذجة المتجهات وكيف أثرت على معالجة اللغة الطبيعية ككل، ثم تشرح بالتفصيل بعض الأساليب المستخدمة مثل ترميز ون هوت، وطريقة نموذج الكلمة إلى المتجه وغيرها. ثم تناقش الورقة التضمينات السياقية للكلمات وكيفية تحسينها على التقنيات السابقة. تسلط الورقة بعد ذلك بعض الضوء على نمذجة اللغة جنبًا إلى جنب مع البنى الجديدة القائمة على آلية الاهتمام (المحولات)، وتناقش بإيجاز كيف تعمل وكيف أثرت ليس فقط على إجابة على أسئلة المجتمع لكن على البرمجة اللغوية العصبية بشكل عام. ثم تناقش الورقة بإيجاز التحول في المجال من الذكاء الاصطناعي القائم على النموذج، حيث ينصب معظم التركيز على إنتاج نموذج بمقاييس عالية الأداء إلى ذكاء اصطناعي مرتكز البيانات حيث ينصب التركيز على محاولة الحصول على طريقة منهجية لتصنيف البيانات حتى يسهل توليد نموذج عالي الأداء.

الكلمات المفتاحية

تعلم الآلة، التعلم العميق، معالجة اللغة الطبيعية، الاجابة على اسئلة المجتمعات، الترتيب