**COSINE SIMILARITY IN QUESTION ANSWERING**

Cosine similarity is a measure that calculates the cosine angle between two given non-zero vectors. This metric is proven to be very relevant when it comes to comparing words, documents, paragraphs or sentences. Given two documents (represented as a non-zero vector) we can directly calculate the angle between these two vectors which will enable us to determine whether the documents or sentences are similar or not. We can do that using the equation of the dot product which is as follows:

Where **a** is the first vector, **b** is the second vector, |**a**| is the magnitude of vector **a**, |**b**| is the magnitude of vector b and is the angle between vector **a** and vector **b** and ranging from 0 to 180 degrees. If two documents are related, they will have cosine similarity value closer to one; cosine similarity value 1 depicts that the two documents have the same meaning. On the other hand, when the cosine angle is 0, the documents are not related. With that said, we can find the similarity between a user asked question and the questions in our database.

**Normalization of data**

Before converting the sentences or documents to the vectors, we have normalized it by removing byte-like characters, and stop words. Since stop words are commonly used and do not affect the result in anyway, we have removed them.

**Vectorization**

However, we cannot pass the word sentences to the model instead we have to convert them to fixed vectors. We used “tfid” vectorizer to transform all the questions in our training into vectors. We also turned all the test questions into vectors of the same size as the training vectors. We then compute the cosine angle (cosine similarity between) them. The question in our dataset that gives the highest cosine similarity with the test question is probably the question asked. The model then responses to the user with the answer of the most likely similar question.

**MINIMUM EDIT DISTANCE FOR QUESTION ANSWERING**

Minimum Edit Distance is the measure of how dissimilar two words are to one another by counting the number of inserts/deletes/substitutes operations needed to transform one to another.  
Usually minimum edit distance is computed by counting the characters that are inserted, deleted or substituted to transform one string to another. However, this algorithm can be extrapolated to compute the minimum edit distance between sentences, by considering word tokens rather than characters. This the basis for using minimum edit distance fore question answering. For instance, given two sentences: s1= “how are you” and s2=” you are coming”, then the minimum edit distance=4.

**Training Model for Question Answering Using Minimum Edit Distance**

To train your model for question answering, a dataset consisting of (question, answers) pairs are provided. Given a user’s query, the minimum edit distance between each question in the dataset and the user’s query is computed, and the question in the dataset having the smallest minimum edit distance with the user’s query is returned as the most probably question similar to the user’s question. Using the most probable question, its corresponding answer can then be returned to the user as the appropriate response to the user.

This model may viewed as a basic implementation for question answering, but running it with test data reveals a high level of accuracy.

**NEURAL QUESTION ANSWERING**

Neural question answering involve the use of neural networks to solve the task of question answering. We considered two ways to go about it: a sequence to sequence model, and an encoder followed by a cosine similarity step like the one described previously.

**Seq2seq model**

Concerning the sequence to sequence model, it consists of two parts. First, an autoencoder that is RNN which extracts the semantic vector of the sentence. The second part is another RNN that takes the semantic vector and then outputs a sequence of numbers based on that. We built a vocabulary object that had all these individual words and a unique index for each of them. These indexes are used as data to train the embedding layer of the encoder. To avoid information loss, the hidden states from the encoder is passed as input to the decoder using an attention mechanism. The sequence returned is now turned back to a sentence using indexing from the vocabulary object.

**encoder**

Another approach (which was not fully implemented) is to have just an encoder. For that approach, we can use a RNN as encoder and have it generate a representation that is the closest possible to the question by using the negative of the log of the cosine similarity as cost function. Alternatively, this can also be done using a vectorizer and a simple feed forward network. This method would be better because of the training cost of big RNNs.

Unfortunately, the seq2seq model could not be fully trained because of the lack of resources (GPUs). It resulted in poor results. However, the main code is available in a Jupyter notebook.

**CHOICE**

We decided to go for the cosine similarity as our main implementation because of its reliability. In terms of consistency of results, cosine similarity seems to perform better than other