# Attention is All you need

April 28, 2022

| | |
|---|---|
| Registration number: | 2100927 |
| Project: | Seq2seq translation |
| Link to GitHub: | http: |

| | |
|---|---|
| Executive summary (max. 250 words) | 225 |
| Introduction (max. 600 words) | 525 |
| Data (max. 300 words/dataset) | 298 |
| Methodology (max. 600 words) | 600 |
| Results and Discussion (max. 1000 words combined) | 947 |
| Conclusions (max. 500 words) | 421 |
| Total word count | 3016 |

## Contents

**Abstract**

This project aims to create a Seq2Seq model for transforming natural language questions to SQL queries. The purpose of the project is to optimize the classifier for a particular question type. Many companies use local database to store all their information and it is hard for people without SQL expertise to come to come to grips of the nature of data. It is worse in research settings because it is common to work with multiple databases. One solution for this is to allow users to pose their queries in natural language format. In this work I describe a Transformer model using attention mechanism to accomplish this task. I have used Embedding and Positional Encoding to process the data before feeding them to the model. The data set consists of a subsample of 1000 questions and equivalent query from spyder dataset. I have used 70 – 15 – 15 Train, Validation and Test Split. By training the model for 750 epochs I was able to get a training accuracy of 82 %. However, validation accuracy of the full query seemed to be a disappointing . But when considering accuracy of tables and columns predicted correctly, I think that it is a good model and can be improved on lot by transformer parameters, which I had to keep to a bear minimum because of performance issues with the system used.

## 1 Introduction

Natural Language Querying allows a question to be posed without knowledge of the database specific languages like SQL [1]. In principle this eases data access to non-expert users. Most multinational companies today use relational databases to store their data and the use of a Seq2Seq model for translating natural language to SQL has never been higher. Natural Language interface for database is on the rise allowing users without database expertise to communicate with databases and retrieve the data they were looking for. Recent surveys [2, 3] segmented them into 5 approaches: keyword-based, pattern-based, syntax based, grammar based, and connectionist based.

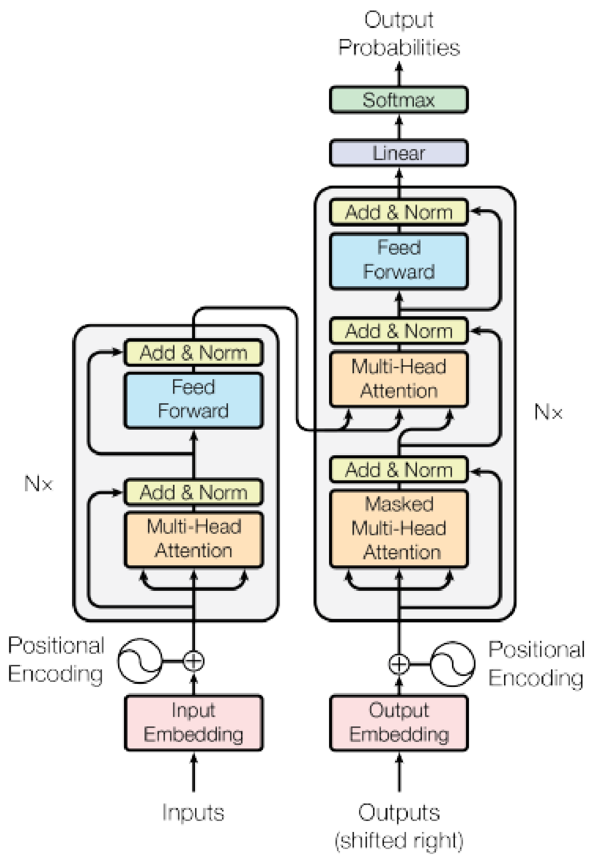


Figure 1: Transformer - Model Architecture [8]

Here we have used a connectionist-based technique also known as the computational intelligence-based approach, tries to learn statistical patterns in data and/or distributed representations of it, allowing for richer variability. These methods use either traditional Machine learning or deep learning which uses natural language processing techniques. Earlier Seq2Seq models consisted basically an encoder and decoder RNNs with encoder summarizing all information the information we want from the input sentence [4]. Each word from the input sentence is vectorized and every new state is calculated using the previous and the next words. The final state of encoder feeds information to the decoder. The decoder uses previous hidden state and output word to compute new hidden state and word. Later people realized this model is too strict as it only allows the hidden state from the last unit. In this model words get added one after the other in the encoder and because of this, we lose information from beginning of the input sentence. Information can get lost during the encoding phase. The inherent sequential nature precludes parallelization within training examples, which becomes important as sequence length gets longer [5].

With attention we add a new input to the cells in decoder RNN called the context vector. The context vector will provide some information about which encoder is output is more important to a decoder depending on a weighted average. Using attention means we use not only the hidden state and the previous output of the encoder, but also make use of all the outputs of the encoder, which can clearly produce a better model.

The transformer (figure-1) model allows more parallelization and can reach new highs in terms of translation quality. After Google released NL2NL model using attention converting English to French, realizing attention added to RNN was all that was required for powerful model, attention model has become very impactful in sequence modelling and transduction models [6,7]. In this project I attempt to replicate the attention model for NL2SQL translation.

## 2 Data

The data is taken from Spider dataset, a large-scale complex and cross-domain semantic parsing and text-to-SQL dataset annotated by 11 Yale students. The dataset contained 7000 question-query data, 6791 after removing duplicates. I have a taken a small subset of the (1000 instances) Spider dataset [8] after removing the duplicates.

I have created a function which gives instances where there is only 1 JOIN clause and a WHERE condition in the query. The dataset is made up of 1000 such data. Here I have tried to optimize the problem for 1 specific type of problem.

### 2.1 Split

I have split the dataset into training(75%)-validation(15%)-testing(15%).

### 2.2 Cleaning

For cleaning the dataset, I have removed all punctuations and replaced short notations like "What's" to "What is". I have restrained from removing punctuations and T1, T2 used to denote tables as suggested in Assignment 1 because I realized these play a important role in predicting the output later.

### 2.3 Tokenizing

Tokenizing transforms each word into one corresponding number. I have used Subword Text Encoding to do so by passing in the cleaned questions and queries. I have set the maximum vocabulary size to 40. I have added 2 to both size of question and queries to mark beginning of sentence and ending of sentence tokens. I have then set the inputs and outputs using these tokens. I have set the Maximum length of Questions and Queries to 40 removing all the instances which goes over this limit. Having high maximum limit needs high processing power. Tokenizing is done only on the training set and after it 601 data remained.

### 2.4 Padding

Size of each instance is set to maximum length using padding. We use keras library to pad 0's at end of each sentence as it is not used by tokenizer. A vector dataset is then created using tensor_slice.

## 3 Methodology

With the dataset now created and ready for training, I have set the buffer size to 20,000 and batch size to 64. These are used for shuffling of the dataset. I have used cache and prefetch with experimental Autotune which increases speed of training.

### 3.1 Positional Encoding

Order of words make an impact in CNN's and RNN's. Words close to each other appear in same filter. But with attention mechanism we have a global way of finding correlation between words and each word is given the same role. A positional encoder helps by telling us about the position of tokens in a sequence. I have used the formula-1,2 to do this using sine and cosine functions.

$$PE_{(pos,2i)} = sin(pos/10000^{(2i/d_model)}) \tag{1}$$

$$PE_{(pos,2i+1)} = cos(pos/10000^{(2i/d_model)}) \tag{2}$$

### 3.2 Attention

The main idea behind Attention is to see how each element in sequence A (Questions in English) is related to each element in B (corresponding SQL Query) (Formula-3). Then we want to rearrange the information in A depending on how each of its element is related to B. Before attention we have a sequence A and a Context B. After attention we will have a new sequence where element I is a mix of elements from A that were related to Bi.

$$Attention(Q, K, V) = Softmax((QK^T)/\sqrt{d_k})V \tag{3}$$

### 3.3 Scaled Dot Product

I have added an encoding and a decoding layer, each starting with an embedded layer. Embedding is done using the TensorFlow function. Then each of the element becomes a vector and DotProduct is used to find the relationship between two vectors. For simplicity, I have taken DotProduct between 1 and -1. If the product is close to 1, the two items are similar. 0 represents no correlation and -1 means 2 words are opposite in meaning. I achieved this using Matrix Multiplication according to Equation 1, where Q (query), K (key), and V (value) are matrix representing sequence/sentences. I have then scaled by dividing with dk, which is each element of each word of keys. Then I have applied softmax to get weights on elements. Finally, I have multiplied with V to recompose elements from our sentence. (figure-2)

### 3.4 Multi-headed Attention

I have linearly projected queries, keys, and values h times with different learned linear projections to dk, dk and dv dimension. On each of these, I have then performed attention function in parallel, getting dv-dimensional output values. Then I have concatenated this and projected again giving the final values as in figure-3. This allows model to jointly attend to information from different representation sub-spaces at different positions. number of Projections=8.

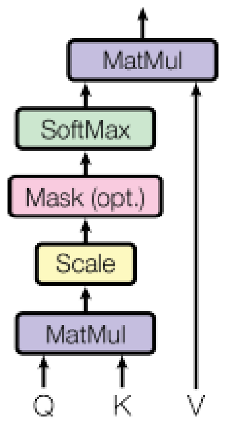


Figure 2: Scaled Dot Product [8]

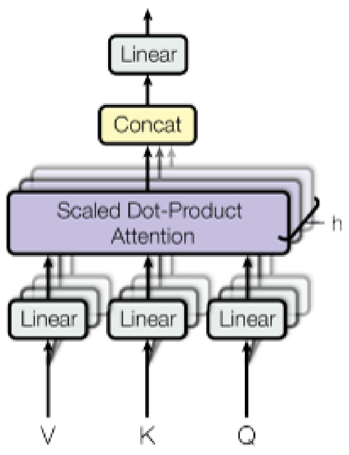


Figure 3: Multi Headed Attention [8]

## 3.5 Encoder-Decoder

Both encoder and decoder are comprised of a stack of N=6 identical layers. Each layer in the encoder has 2 sub layers, first a multi-head self-attention mechanism and then a simple position-wise fully connected Feed-Forward-Network. Each sublayer has a residual connection with dropout rate set to 0.01 [9] followed by normalization. In [8] the dmodel is set to 512 but here due to system performance I had to go with 128. Each Sublayer Output = LayerNorm(x + Sublayer(x)), where Sublayer(x) is function implemented by sublayer itself. In addition to sublayers in encoder, decoder adds a 3rd sublayer that performs multihead attention on output encoder stack.

## 3.6 Feed Forward Layers

After attention layers, each of the layers in our encoder-decoder contains 2 fully connected FFNs (formula-4). This contains 2 linear transformations with Relu activation in between [8]. Dimensionality of input and output is dmodel=128 and inner layer has dimensionality dff=512.

$$FFN(x) = max(0, xW1 + b1)W2 + b2 \tag{4}$$

## 3.7 Self Attention

At beginning of each of Encoding or Decoding Layers we recompose sequence to identify relationship of each element others, by using global information of the sequence. Q=K=V.

### 3.8 Look Ahead Mask

While training we feed whole output, whereas prediction word, we should not see words after it. So, we multiply with matrix represented in figure-4.

$$QK^T * \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Figure 4: Matrix Multiplication for Look Ahead Mask [8]

| **Hyper Parameter** | **Used** | **Original from [8]** |
|---|---|---|
| Model Dimension | 128 | 512 |
| Number of Layers | 6 | 6 |
| FFN Units | 512 | 2048 |
| Number of Projections | 8 | 8 |
| Dropout Rate | 0.01 | 0.1 |
| Warmup Steps | 4000 | 4000 |

Table 1: Hyper Parameters

## 4 Results

During training we are to put some probability for each word in our vocabulary to be the next word appearing in our output sequence. We achieve this by using Sparce Categorical Cross entropy. I have masked the padding tokens when computing loss so that only real words get predictions and not the 0s we added at the end to reach max length. I have used the mean of losses during training to get the training loss. I have used Sparce Categorical Accuracy to get the training Accuracy. I use a custom scheduler for learning rates using warmup steps =4000 as in [8] using formula 5. I have used an Adam Optimizer with the initial learning rate=0.01 and setting the beta 1 and beta2 parameters to 0.9 and 0.98 respectively. Epsilon is set to a large negative number. I have trained the model for 1500 epochs and found the training accuracy to reach its threshold at **0.8265**, with a training loss of **0.0035**. The model is saved at the end of each epoch, however I only keep the last 5 checkpoints, so if there are more than 5, the older ones get deleted.

$$Irate = d_{(model)}^{(-0.5)} \cdot min(step\_num^{(-0.5)}, step\_num \cdot warmup\_steps^{(-1.5)}) \tag{5}$$

| | **Validation Set** | **Training Set** |
|---|---|---|
| Correct Predictions (%) Correct Predictions / Total Predictions (150) | 25.33 | 24.66 |
| Has 1 Where and only 1 Join | 150 | 150 |
| Table Predictions (Correct Predictions / Total number of tables (300) | 42 | 45 |
| Mean Cosine Similarity | 0.676 | 0.69 |
| Mean Score for finding correct columns (Mean of Correctly Predicted Columns / Total Number of Columns in original query) | 0.38 | 0.387 |

Table 2: Results from Validation and Test Sets

During the validation and the test phase we need to first clean the sentence removing punctuations and short forms just like we did for the training set. We also need to encode the question sentence using tokeizer and add starting and ending tokens. I used the evaluator to predict the query for questions in the validation dataset and found that the translator was able to predict the query **25.33%** of the time by comparing predicted query to the original. Even though this is not a high score I found the model to be good when considering other facts. The first test was to check if all the queries generated has 1 **Where** condition and only 1 **Join** Clause, since the model was trained to that specific type of data and all **150** predicted queries seem to follow this. I then created a function to check the accuracy of the model in correctly predicting the table names, comparing it to the original query. Since there are only 2 tables involved, the function returns 2 if both tables are correctly predicted, 1 if one of the tables is correctly predicted and 0 if none of the tables are correctly predicted. It was found that the model was able to predict tables correctly **42%** of the times. I have then used Cosine Similarity (formula-6) to show the similarity between the original query and the predicted one. The mean of which was a striking **0.676**. I then tried to get the score of predicting columns correctly by dividing the number of columns predicted correctly by total number of columns in the original query, the mean of which was found to be **0.38**.

$$Cos(\Theta) = \frac{A \cdot B}{\| A \| \cdot \| B \|} \tag{6}$$

I have then used the model to predict the query for the testing dataset. The translator was able to predict the exact query **24.66%** of the times. It was again found that all the **150** predicted queries have a **Where** condition and only 1 **Join** clause. It was found the model was able to predict the correct tables **45%** of the times. The model had a mean cosine similarity score of **0.69** on the testing set. The mean score for finding the correct column was found to be **0.387**. I then used the cosine similarity score to categorize predicted queries according to their effectiveness to Very Poor for scores between (0-0.2). Poor for scores between (0.2-0.4), Good for scores between (0.4-0.6), Very Good for between (0.4-0.8), and Excellent for scores between (0.8-1). It was found that there was no query in the Very Bad Category and only 1 query in the Poor Category. I then plot the Effectiveness Vs Similarity as represented by figure-5

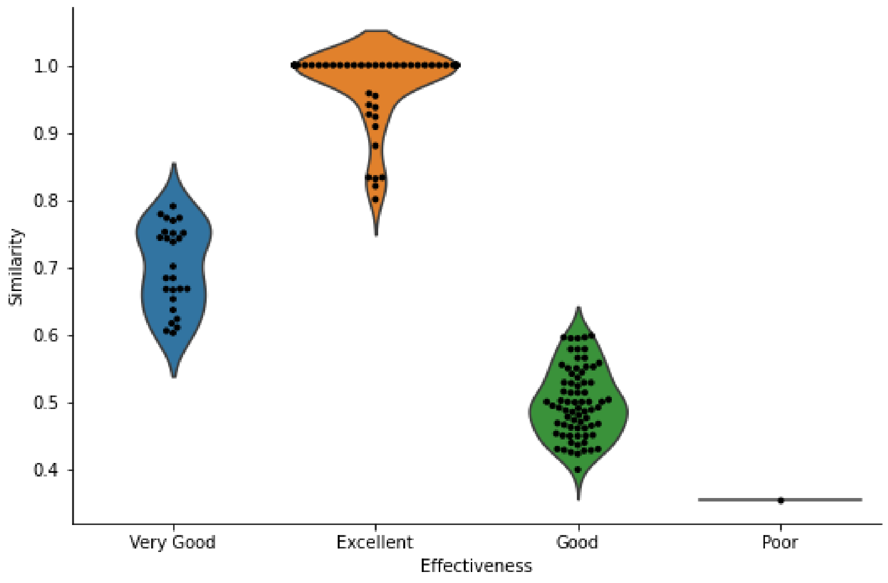
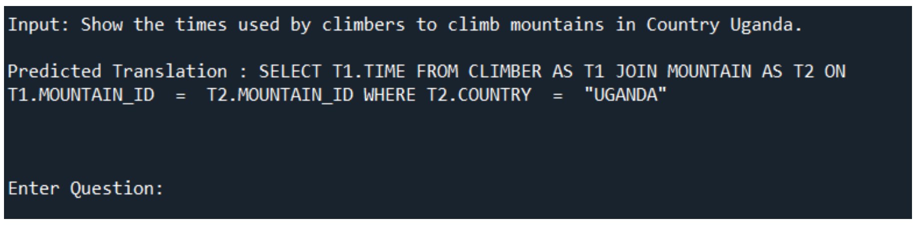


Figure 5: Similarity Vs Effectiveness Plot



```
Input: Show the times used by climbers to climb mountains in Country Uganda.

Predicted Translation : SELECT T1.TIME FROM CLIMBER AS T1 JOIN MOUNTAIN AS T2 ON
T1.MOUNTAIN_ID  =  T2.MOUNTAIN_ID WHERE T2.COUNTRY  =  "UGANDA"


Enter Question:
```

Figure 6: Output Screen

| Question | Original Query | Predicted Query | Correc Predicted Tables | Colum Score | Cosine Similarity | Effectiveness |
|---|---|---|---|---|---|---|
| WHAT ARE THE LAST NAMES OF THE TEACHERS WHO TEACH THE STUDENT CALLED GELL TAMI | SELECT T2.LASTNAME FROM LIST AS T1 JOIN TEACHERS AS T2 ON T1.CLASSROOM = T2.CLASSROOM WHERE T1.FIRSTNAME = "GELL" AND T1.LASTNAME = "TAMI" | SELECT T2.LASTNAME FROM LIST AS T1 JOIN TEACHERS AS T2 ON T1.CLASSROOM = T2.CLASSROOM WHERE T1.FIRSTNAME = "GELL" AND T1.LASTNAME = "TAMI" | 2 | 1 | 1 | Excellent |
| FIND THE AVERAGE PRICE OF WINES THAT ARE NOT PRODUCED FROM SONOMA COUNTY | SELECT AVG(PRICE) FROM WINE WHERE APPELATION NOT IN (SELECT T1.APPELATION FROM APPELLATIONS AS T1 JOIN WINE AS T2 ON T1.APPELATION = T2.APPELATION WHERE T1.COUNTY = 'SONOMA') | SELECT AVG(T2.PRICE) FROM APPELLATIONS AS T1 JOIN WINE AS T2 ON T1.APPELATION = T2.APPELATION WHERE T1.COUNTY = "SONOMA" | 2 | 0.77 | 0.75 | Very Good |
| SHOW THE TOTAL NUMBER OF ROOMS OF ALL APARTMENTS WITH FACILITY CODE GYM | SELECT SUM(T2.ROOM_CO FROM APARTMENT_FACILITIES AS T1 JOIN APARTMENTS AS T2 ON T1.APT_ID = T2.APT_ID WHERE T1.FACILITY_CODI = "GYM" | SELECT T1.BUILDING_ADD FROM APARTMENT_BUILDINGS AS T1 JOIN APARTMENTS AS T2 ON T1.BUILDING_ID = T2.BUILDING_ID WHERE T2.BATHROOM_CC 2 | 1 | 0.529 | 0 | Good |

Table 3: Practice used for results

## 5 Discussion

Even though the results in validation and testing gave low scores compared to the training accuracy, The Cosine Similarity seems to show that the model is good and has great potential to be improved. There is almost no predicted query with similarity score lower than 0.4. Because this is a translation problem, we cannot say the model is bad just depending on the accuracy score. Queries with same meaning can be achieved using different wordings. Due to the vastness of the dataset used predicting the table names alone is huge and the model seems to get it right over 40% of the times. Another important thing to note is that all the predicted queries are semantically correct and follow the SQL syntax. I also think that the model can be improved if using the model dimension and FFN units used in [8] which I had to restrain from because performance issues.

An accuracy of over 60% was attained by [10] over the entire spyder dataset in only 90 epochs. Both [10] and [11] describes queries with multiple tables as complex, and even though it took a larger number of epochs I was able to get a training accuracy of 82%. [11] discusses various benchmarks and provides performance evaluation of various NL2SQL translators. Our test accuracy of 25% seem a relevant number of methods available like NaLIR and Templar.

The performance of the model can also be improved using a larger dataset for training. We could use beam search strategy for decoding the test dataset instead of greedy approach (argmax). One of the drawbacks of the attention model is that it is time consuming. This is the case especially when the data are long sequences and since I chose to us complex query with two tables I had to cut down the size of training date removing all sequences where length was greater than 40 in either question or query.

## 6 Conclusions

In this work I have presented a transformer model with attention and use a sub sample of the Syder dataset involving complex queries with a Where condition and only 1 Join Clause. This is built on top of the basic encoder decoder model (figure-7), by adding in a new global variable called the context vector which provides weighted information about each state of the encoder during decoder. The decoder now not only has access to the previous state and the hidden state but also all the states of the encoder (figure-8). This help in predicting the output sequence better.

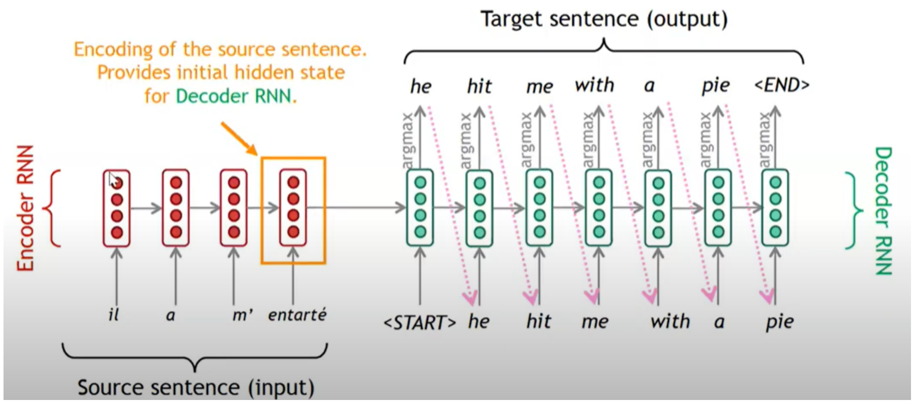

Target sentence (output)
Encoding of the source sentence. Provides initial hidden state for Decoder RNN.
he hit me with a pie <END>
argmax
Encoder RNN
Decoder RNN
il a m' entarté
<START> he hit me with a pie
Source sentence (input)

Figure 7: Basic Encoder-Decoder Decoder

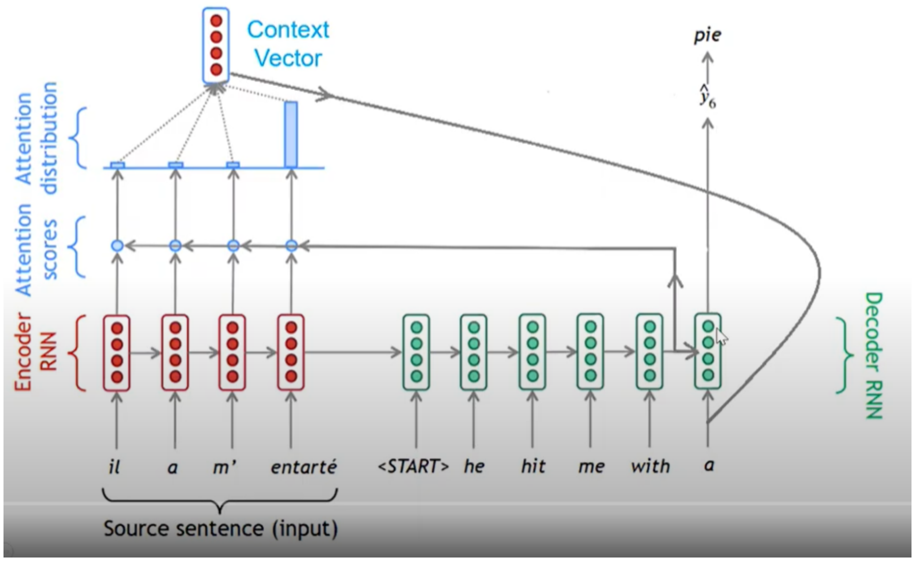


Figure 8: Attention Model

During the training phase I was not able to use the same hyper parameters used in [8] due to performance issues and because the attention model is time consuming. I was still able to get a high accuracy score of 82%. Even though the accuracy on the validation and test dataset was only 25%, I believe this to be a good model considering other factors such as accuracy of predicting the correct tables and correctly predicted columns over total number of original columns. When using Cosine Similarity score comparing the original and the predicted queries, it was seen that most of the queries had a score over 0.4. All the predicted queries followed the SQL syntax and had no syntactical error. Even though the accuracy in predicting the exact query is not very good a closer look at the other factors involved suggests that the model is good.

I feel that there is certainly room for a lot of improvement by increasing the size of dataset. While vectorizing I had to decide up on a maximum length for the questions and queries to follow and since these complex queries had large sentence length, it was not in my advantage as I had to decide on a smaller max length for training to be less time consuming. I believe using a system with higher processing power could help with this. I also believe not having to compromise on the hyper parameters could have helped improve the model.

For future work I would certainly like to add to the complexity of the queries. I would also suggest working with 2 or more different types of queries with a larger dataset and see how the attention model performs to it. Contrastingly I believe lowering the complexity of the query would improve the testing accuracy score. I also think that using tables from only 1 database would prove to have a higher accuracy score while testing.

## References

[1] G. Micklem, “Translating synthetic natural language to database queries with a polyglot deep learning framework,” *Scientific Reports*, vol. 11, pp. 2045–2322, 2021. [Online]. Available: https://doi.org/10.1038/s41598-021-98019-3

[2] K. Affolter, K. Stockinger, and A. Bernstein, “A comparative survey of recent natural language interfaces for databases,” *The VLDB Journal*, vol. 28, pp. 793–819, 10 2019. [Online]. Available: https://doi.org/10.1007/s00778-019-00567-8

[3] S. A. C. Bukhari, H. Dar, M. I. Lali, F. Keshtkar, K. Malik, and S. Kadry, “Frameworks for querying databases using natural language: A literature review – nlp-to-db querying frameworks,” *International Journal of Data Warehousing and Mining*, vol. 17, pp. 21–38, 04 2021.

[4] Y. Mellah, E. H. Ettifouri, A. Rhouati, W. Dahhane, B. Toumi, and M. Belkasmi, “Sql generation from natural language using supervised learning and recurrent neural networks,” pp. 175–183, 01 2021.

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin, “Attention is all you need,” vol. 30, pp. 1–9, 2017. [Online]. Available: https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[6] A. M. Rush, S. Chopra, and J. Weston, “A neural attention model for abstractive sentence summarization,” *CoRR*, vol. abs/1509.00685, 2015. [Online]. Available: http://arxiv.org/abs/1509.00685

[7] A. P. Parikh, O. Täckström, D. Das, and J. Uszkoreit, “A decomposable attention model for natural language inference,” *CoRR*, vol. abs/1606.01933, 2016. [Online]. Available: http://arxiv.org/abs/1606.01933

[8] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, Z. Zhang, and D. R. Radev”, “Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task,” *Empirical Methods in Natural Language Processing*, pp. 11–10, 2018.

[9] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” pp. 770–778, 2016.

[10] U. Brunner and K. Stockinger, “Valuenet: A natural language-to-sql system that learns from database information,” in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, 2021, pp. 2177–2182.

[11] H. Kim, B.-H. So, W.-S. Han, and H. Lee, “Natural language to sql: Where are we today?” *Proc. VLDB Endow.*, vol. 13, pp. 1737–1750, 2020.

[1] [2] [3] [4] [5] [6] [7] [8] [9] [10] [11]