Attention Is All You Need

February 24, 2022

Registration number: 2100927
Project: Seq2seq translation
Link to GitHub: https://github.com/AJ6546/CE888/tree/main/Assignment_1

Executive summary	131
Introduction	510
Data	526
Methodology	598
Conclusions	461
Total word count	2158

Contents

Abstract

The project is about creating a Seq2Seq model for converting natural language questions to sql queries. The goal of the project is to optimize classifiers for one specific type of question. A lot of businesses have all their internal information within local databases and answering even minor questions requires extensive human expertise. So, building Seq2Seq model can be crucial to reduce human error. I am going to use a sub-sample containing 1000 questions its equivalent SQL queries of the spider Dataset. I am going to use a Transformer model to accomplish this task. Then we are going to use attention mechanism to improve the model. We will use Embedding and Positional Encoding to process the data before feeding them to the model. We are going to use a 70-15-15 Train-Validation-Test split.

1 Introduction

A seq2Seq model for translating natural language to SQL is of high importance especially because, most of the multinational companies use SQL for their relational database. Given a question in natural language, translating it to SQL statement to answer the question could be very useful when applied to commercial relational database. [1] Natural Language interface for database is a growing field and it allows users without SQL expertise to interact with the database. These interfaces solve the problem of synthesizing SQL queries based on requirement. Early works in the field cover rule-based semantic-parsing. This method proved to be effective but failed to cover linguistic variation and sophistication of end-user.

Earlier models on NL2SQL consisted of basically an encoder and a decoder. The encoder is what is supposed to summarize all the information that we want to use from the input sentence and the decoder will use as inputs the output of the encoder to create the right output. Each word is converted to vectors and each new state of the encoder is computed from the previous one and the next word. The final state of the encoder conveys information to start decoding. The decoder uses the previous hidden state and output word to compute new hidden state and word. One of the drawbacks of the model is that we add words one after the other in encoder and so we will certainly lose information from the beginning of the input sentence. So, for long sentences the information will not be complete and when we want to predict the next word in the decoder, we could miss information that has been lost through encoding phase. But What Attention does is during the decoding phase it adds a new input to the cells in RNN called the context vector, that is supposed to convey a global information about the whole input sequence and how it is related to the current status in decoding phase.

There has been lots of research going in the field and Recurrent Neural Networks, Long Short-Term Memory and gated Recurrent Neural Networks have been established as the go-to methods for sequence-to-sequence translation. There are only few that use the attention mechanism to do the translation. All you need is attention is the title of the article release by google after they realized that attention, they added to RNN was all that was required to create a powerful model. However, they used the model for NL2NL conversion of English to French. I would like to try this model for converting English language questions to corresponding SQL queries. I think that such an idea is worth exploring because at the end of the day, SQL is still a language and if trained well using the Attention model, we might get some good results.

The transformer allows significantly more parallelization and can reach new highs in terms of translation quality. As per [2] Attention mechanisms has become very impactful in sequence modelling and transduction models. It allows modelling of dependencies without regard to their distance in the input or output sequences.

2 Data

The data is taken from Spider dataset, which is a large-scale complex and cross-domain semantic parsing and text-to-SQL dataset annotated by 11 Yale students. I have a taken a small subset of the (1000 instances) Spider dataset [3] after removing the duplicates.

2.1 Dataset 1

I have first set some words used in SQL statements and classified them into different lists. Then I have searched the entire Spider dataset 7000 instances for each word in the different lists and

added them to different data frames. Then from these data frames I have selected 1000 instances to form the final dataset. I have taken 6% out of all the values from the different datasets because with this, I was getting values near to 1000 after deleting the duplicates.

2.2 Cleaning Datasets

For cleaning the data set I have written a method which clears up the text of some of the punctuations, short notations like What's that could appear is replaced with "What Is". I have also removed T1, T2 etc used to denote table names in the query as I think this might be confusing and it is better if we can add them later after the decoding phase, checking if the query has JOIN clause.

2.3 Tokenizing

Tokenizing is the process of transforming each word into one corresponding number. tensorfow_datasets library provides a SubwordTextEncoder to which we can pass out list clean questions and queries. It also allows to set a Maximum vocabulary length. Now we get the vocabulary size for both the questions and the queries. We need to add 2 to both because we are going to use tokens, 1 to mark the beginning of the sentence and another to mark the ending of the sentence. I have then set the inputs and outputs using these tokens. I have set the Maximum length of Questions and Queries to 40 and removed all the instances which exceeded this limit. We are doing this because in the next step we are going to pad the sentences and make them all the same length, so setting the maximum length too high and keeping all the instances is going to take a lot of processing power. 868 instances remained.

2.4 Padding

Padding is done to set the size of each instance to max length. For padding we use keras library. We are going to pad at the end of each sentence, and we use 0 for padding as it is not used by the tokenizer. Then using tensor_slices I have created the dataset containing vectors inputs and outputs and further split this into train, validation and test sets.

2.5 Dataset 2

I have created a function that gives out instances where there is 1 'JOIN' clause and a 'WHERE' condition in the Query. With this dataset, I am trying to optimise the problem to one specific type of question.

2.6 Cleaning Datasets

For cleaning the data set I have performed the same steps done for cleaning Dataset-1.

2.7 Tokenizing

For this Dataset also I have set the maximum Length to 40 and removed all the instances which exceeded this limit.After Tokenizing 854 instances remained.

2.8 Padding

Here again I have padded this datset with 0. Then used tensor_slice to create datasets

3 Methodology

We do a Train-Validation-Test Split of 70-15-15%. Then we fit the model figure 1 to training-set. Validate using SparseCategoricalCrossEntropy on validation-set and then use it to predict the test-set.

The main idea of Attention is to see how each element in sequence A (Questions in English) is related to each element in B (the corresponding SQL Query). Then we want to rearrange the information in A depending on how each of its element is related to B.

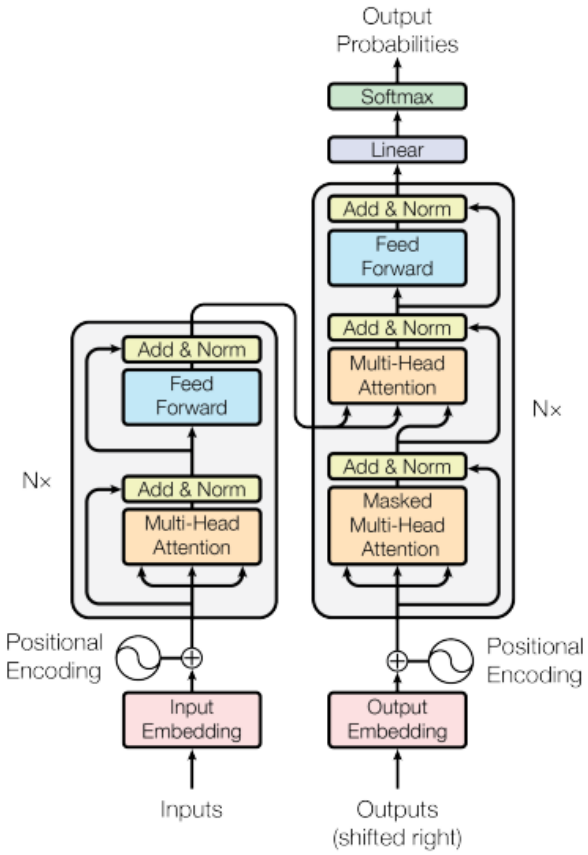

Figure 1: Figure 1: Transformer - Model Architecture

We have an encoding and a decoding layer each starting with an Embedding layer. The Embedding will be done using TensorFlow function. After this each element will be a vector and we can use Dot-Product to find relation between 2 vectors. For simplicity we take the Dot-Product to be between -1 and 1. If the dot product is close to 1 the 2 elements are similar. There is no correlation if dot product is 0 and the elements are opposite to each-other if it is close to -1. This is achieved using Matrix-Multiplication as per formula 1 where Q (Query), K (Keys) and V (Values) are the matrix representing the Sequences/Sentences Then a scaling is done to improve the model by dividing each by root of dimension of k. A Softmax function is used to get the weights on the values.

$$Attention(Q, K, V) = Softmax((QK^T)/\sqrt{d_k})V \tag{1}$$

3.1 Self-Attention

In the beginning of the Encoding or Decoding Layers we recompose sequence to know how each element is related to the others, grabbing global information about a sentence or sequence. Q=K=V.

3.2 Encoder-Decoder Attention

We have an internal sequence Q in Decoder and a context in Encoder (K=V). Our new sequences are a combination of information from our context, guided by the relation decoder-encoder output: K=V

3.3 Look-Ahead Mask

During training we feed the whole output, but while prediction word n, we must not look at words after n. This is achieved by multiplying with a matrix as represented in figure 4.

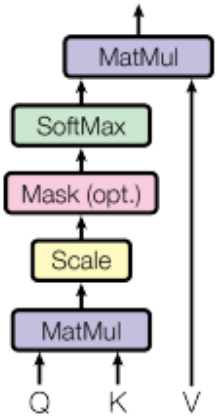

Figure 2: Scaled Dot Product

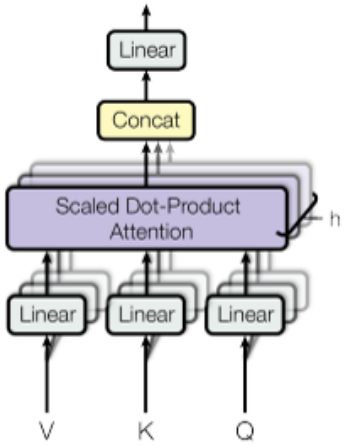

Figure 3: Multi Headed Attention

3.4 Multi-Head Attention

As per [2] a multi-head attention allows the model to jointly attend to information from different representation sub-spaces at different positions. With a single attention head, averaging inhibits this. Figure-3

3.5 Positional Encoding

With CNN's or RNN's the order of the words had an impact on the process, so the words that are close to each other appear in the same filter. But with the transformer its not the case anymore as attention mechanism has a global way of finding corelation between words and we give the same role to each word. Because of this we need to add something that tells us about the position of tokens in the sequence. There are several methods to do positional encoding but here we use sine and cosine functions as per formula 2 3 which is used in [2]

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

$$PE_{(pos,2i+1)} = cos(pos/10000^{(2i/d_model)}) \tag{3}$$

3.6 Feed-forward layers

We are going to use a feed forward layer at the end of encoder and decoder. These are made of 2 dense layers which is used to learn more about the data by itself. So as in formula 3 x is the

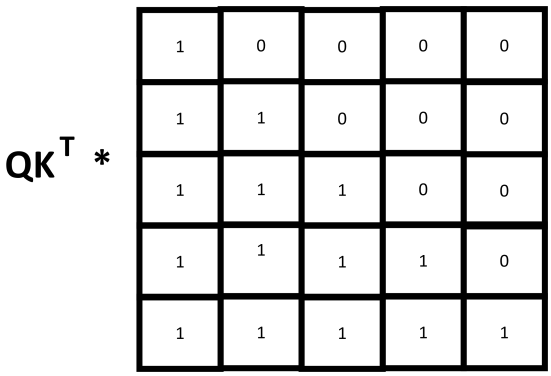

Figure 4: Matrix Multiplication for Look Ahead Mask

input of the feed forward, we multiply it with a weight matrix and add a bias. Then we add a relu-activation function, and this removes all negative values of layer. Then we multiply this with a 2nd weight matrix and add a bias.

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

3.7 Add Norm

The output of each sublayer is added to the input of themselves, and we do a Standard-Normalizaion. This is so that we don't forget about information we had in each position also helps learning during back-propagation.

3.8 Linear-Layer

output of decoder goes through a dense layer with vocab_size units and a Softmax, to get probabilities of each word.

4 Conclusions

In this work I will be trying to present a transformer model based on attention and use 2 different sub sample datasets from the Spyder dataset. The first dataset is taken by dividing the original dataset into different groups based on occurrence of some often-used semantics and operators in sql. Then I select an instance from these different groups, delete any duplicates and form a dataset of 1000 instances. I think this would tell how a transformer will do when it has a variety of different queries, and the questions cover a broader area. The 2nd dataset focuses more on a specific type of query. So, for this I have selected only instances where the queries have a where clause and 1 join in them.

According to [2] training a transformer can be relatively faster for translation than with recurrent or convolutional layers. Earlier when RNN's were used we had an encoder and a decoder which consist of different cells as in figure-5. For the encoder each of these cells took as input the outputs of previous cell and the next word from input sentence and outputs a hidden state. All the cells were identical, and we had to use as many cells as length of input sentence. Final state of encoder conveys information to start decoding. One of the disadvantages of this model is we could lose information during encoding phase as by the time we get to the decoder the input from beginning of the sentence is faded. The sequential model is not global enough and loses information for long sentences. But with attention as in figure-6 we add a new global context vector that uses all the input sequence and the last state from decoder. It says how the current state of the decoder is related to the global input sequence and hence improves decoding phase.

After cleaning the Datasets, the vocab size for questions and queries were found to be 2158 and 2484 for Dataset 1 and 2026 and 2162 with Dataset 2. I have used a 70-15-15 split for train validation and test sets. One of the disadvantages of the model that I could find was we need to set up a maximum length for the instances and it would take lot processing power if this were set too

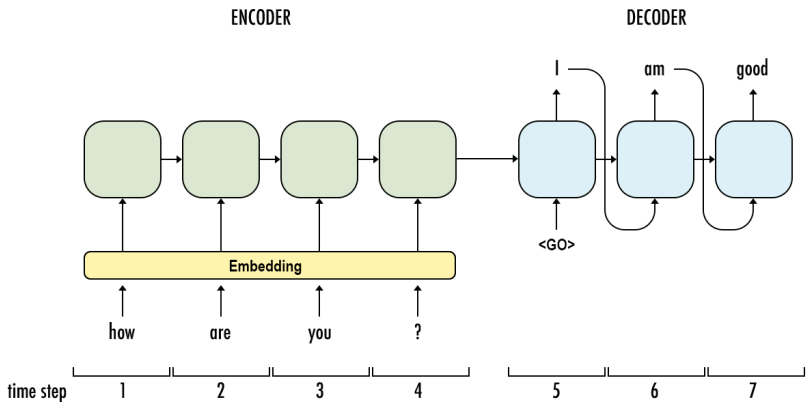

Figure 5: Basic Encoder-Decoder Model

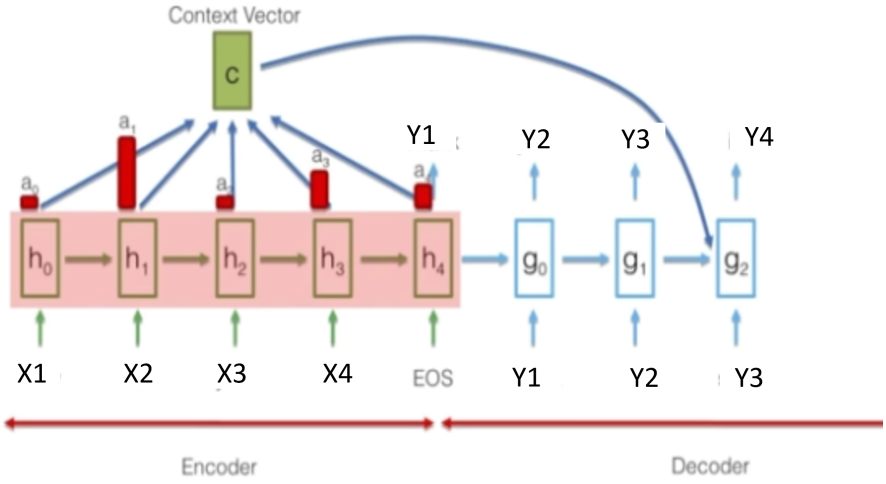

Figure 6: Attention Model

high. I have set this to 40 which I thought was a good number and would go easy on the processor during training phase. So, I had to remove some of instances from the initial dataset we still have over 800 instances left for both the Datasets. Also, all the inputs must be of the same length, so we need to pad the shorter sentences to match the size of maximum length we have set up.

4.1 How to write Mathematics

Let $X_1, X_2, \ldots, X_n$ be a sequence of independent and identically distributed random variables with $\mathrm{E}[X_i] = \mu$ and $\mathrm{Var}[X_i] = \sigma^2 < \infty$, and let

$$S_n = \frac{X_1 + X_2 + \cdots + X_n}{n} = \frac{1}{n}\sum_{i}^{n} X_i$$

denote their mean. Then as n approaches infinity, the random variables $\sqrt{n}(S_n - \mu)$ converge in distribution to a normal $\mathcal{N}(0, \sigma^2)$.

[1]. [2] [3]

References

[1] K. Dong, K. Lu, X. Xia, D. Ciesla, and N. V. Chawl. Tan optimized nl2sql system for enterprise data mart. *Lecture Notes in Computer Science*, 12979.(1):1–3, 2021.

[2] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. 30:1–9, 2017.

[3] 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*, pages 11–10, 2018.