Text-to-SQL Model, A Deep Learning Approach

February 29, 2024

0.1 By: Owen Petter

0.1.1 Contents:

0.1.2 Introduction:

The goal of this project is to utilize a transformer sequence-to-sequence with positional embedding to try and pick up the relationship between plain English and SQL queries, and generate SQL queriers from plain english text. The nature of this problem varies from traditional language-to-language translation-based deep learning problems as there is a substantial difference in the number of words (tokens) in the English language when compared to the number of SQL tokens. Additionally, there is a vast amount of ways one could describe a SELECT or WHERE function, this may be a challenge for the model to learn the nuance and complex combinations one might use to enquire about a certain SQL function.

0.1.3 Business Implications:

Within our organization, access to our relational database is currently restricted to individuals with proficiency in Structured Query Language (SQL), creating a significant imbalance in data access across the organization. This imbalance leads to several adverse effects, including an increased workload for our technical data team, who are required to extract data for non-technical members, diverting their time from value-added projects. Furthermore, this situation results in delays for non-technical individuals who need data, as they must wait for the data team to process their requests, which can significantly impede their workflow.

In light of this issue, the primary objective of my deep learning solution is to enable non-technical members to obtain direct access to data more quickly by providing them with SQL code generated from plain English queries. This approach will allow users to execute these queries directly in a Database Management System (DBMS) or use them as a foundation for developing their queries. However, there is a potential risk that the generated output may not meet the user's exact needs, leading to errors during data extraction or the retrieval of incorrect data. Such errors

could adversely affect decision-making processes or reports and reduce productivity if time is spent correcting these errors.

The implementation of this model will involve several key steps. Initially, a user interface (UI) will be developed to allow users to input their queries and receive SQL outputs. This interface could be a desktop application accessible to all organization members. The UI will connect to an API that translates the user's natural language queries into SQL queries, which are then returned to the UI for use in the DBMS. A beta version of this system will be released to the technical data team first, featuring a rating system for evaluating the quality of the generated SQL code. Feedback from this beta phase will be used to refine the model's SQL generation capabilities. Following approval from the data team, comprehensive documentation on using the application and navigating the organization's DBMS will be created. This documentation is essential for ensuring that users can effectively use the queries generated by the application. Training sessions will also be conducted to familiarize all users with the UI, the model's outputs and limitations, and the DBMS navigation process. User feedback on the SQL outputs will be collected through the UI, contributing to the model's ongoing improvement and accuracy.

The execution of this plan will require a combination of technical, human, and financial resources. Technically, continuous access to computing power is necessary for training and updating the model, which could be sourced from on-premises infrastructure or cloud platforms like GCP. From a human resources perspective, software developers, database administrators, and project managers will play critical roles in developing the UI and API, documenting the DBMS process, and overseeing the project's testing and implementation phases. Financially, the project costs will primarily cover computational resources and personnel involved in the project.

0.1.4 Data:

The data used for this project, WikiSQL, is a large crowd-sourced dataset of 80654 hand annotated examples of questions and SQL queries distributed across 24241 tables from Wikipedia. It is being pulled from Hugging Face and is imported via the Hugging Face dataset library. It is split into 56,355 training examples, 8,421 validation examples, and 15,878 testing examples.

Initial Data Analysis and Structuring:

[1]:
```
from datasets import load_dataset

dataset = load_dataset("wikisql")
```

[2]:
```
dataset
```

[2]:
```
DatasetDict({
    test: Dataset({
        features: ['phase', 'question', 'table', 'sql'],
        num_rows: 15878
    })
    validation: Dataset({
        features: ['phase', 'question', 'table', 'sql'],
        num_rows: 8421
    })
    train: Dataset({
```

```
        features: ['phase', 'question', 'table', 'sql'],
        num_rows: 56355
    })
})
```

[3]:
```
#Data is already split from source, making each set a variable

train_data = dataset['train']
validation_data = dataset['validation']
test_data = dataset['test']
```

[4]:
```
#Inspecting data

train_data[0]
```

[4]:
```
{'phase': 1,
 'question': 'Tell me what the notes are for South Australia ',
 'table': {'header': ['State/territory',
   'Text/background colour',
   'Format',
   'Current slogan',
   'Current series',
   'Notes'],
  'page_title': '',
  'page_id': '',
  'types': ['text', 'text', 'text', 'text', 'text', 'text'],
  'id': '1-1000181-1',
  'section_title': '',
  'caption': '',
  'rows': [['Australian Capital Territory',
    'blue/white',
    'Yaa·nna',
    'ACT · CELEBRATION OF A CENTURY 2013',
    'YIL·00A',
    'Slogan screenprinted on plate'],
   ['New South Wales',
    'black/yellow',
    'aa·nn·aa',
    'NEW SOUTH WALES',
    'BX·99·HI',
    'No slogan on current series'],
   ['New South Wales',
    'black/white',
    'aaa·nna',
    'NSW',
    'CPX·12A',
    'Optional white slimline series'],
```

```
    ['Northern Territory',
     'ochre/white',
     'Ca·nn·aa',
     'NT · OUTBACK AUSTRALIA',
     'CB·06·ZZ',
     'New series began in June 2011'],
    ['Queensland',
     'maroon/white',
     'nnn·aaa',
     'QUEENSLAND · SUNSHINE STATE',
     '999·TLG',
     'Slogan embossed on plate'],
    ['South Australia',
     'black/white',
     'Snnn·aaa',
     'SOUTH AUSTRALIA',
     'SOOO·AZD',
     'No slogan on current series'],
    ['Victoria',
     'blue/white',
     'aaa·nnn',
     'VICTORIA - THE PLACE TO BE',
     'ZZZ·562',
     'Current series will be exhausted this year']],
  'name': 'table_1000181_1'},
 'sql': {'human_readable': 'SELECT Notes FROM table WHERE Current slogan = SOUTH
AUSTRALIA',
  'sel': 5,
  'agg': 0,
  'conds': {'column_index': [3],
   'operator_index': [0],
   'condition': ['SOUTH AUSTRALIA']}}}
```

[5]:
```
#changing structure of dataset, making it simplier to work with (idea from␣
↪https://colab.research.google.com/github/mrm8488/shared_colab_notebooks/blob/
↪master/T5_wikiSQL_with_HF_transformers.ipynb#scrollTo=mt-c0E0GEPI7)

def structure_dataset(raw_data):
 return {'input': raw_data['question'], 'target':␣
↪raw_data['sql']['human_readable']}
```

[6]:
```
struc_train_data = train_data.map(structure_dataset, remove_columns=train_data.
↪column_names)
```

[7]:
```
struc_train_data[0]
```

[7]:
```
{'input': 'Tell me what the notes are for South Australia ',
 'target': 'SELECT Notes FROM table WHERE Current slogan = SOUTH AUSTRALIA'}
```

[8]:
```
#adding [START] AND [END] at beinging and end of target SQL code to assist with␣
 ↪translation later
input_list_train = [example['input'] for example in struc_train_data]
target_list_train = [example['target'] for example in struc_train_data]


formatted_train_data = []

for i in range(len(target_list_train)):
    target_list_train[i] = "[start] " + target_list_train[i] + " [end]"
    formatted_train_data.append((input_list_train[i], target_list_train[i]))

formatted_train_data[0]
```

[8]:
```
('Tell me what the notes are for South Australia ',
 '[start] SELECT Notes FROM table WHERE Current slogan = SOUTH AUSTRALIA [end]')
```

[9]:
```
struc_test_data = test_data.map(structure_dataset, remove_columns=test_data.
 ↪column_names)
```

[10]:
```
#doing the same for the test data set
input_list_test = [example['input'] for example in struc_test_data]
target_list_test = [example['target'] for example in struc_test_data]


formatted_test_data = []

for i in range(len(target_list_test)):
    target_list_test[i] = "[start] " + target_list_test[i] + " [end]"
    formatted_test_data.append((input_list_test[i], target_list_test[i]))

formatted_test_data[0]
```

[10]:
```
("What is terrence ross' nationality",
 '[start] SELECT Nationality FROM table WHERE Player = Terrence Ross [end]')
```

[11]:
```
struc_val_data = validation_data.map(structure_dataset,␣
 ↪remove_columns=validation_data.column_names)
```

[12]:
```
#doing the same for the validation data set
input_list_val = [example['input'] for example in struc_val_data]
target_list_val = [example['target'] for example in struc_val_data]
```

```
formatted_val_data = []

for i in range(len(target_list_val)):
    target_list_val[i] = "[start] " + target_list_val[i] + " [end]"
    formatted_val_data.append((input_list_val[i], target_list_val[i]))

formatted_val_data[0]
```

[12]:
```
('What position does the player who played for butler cc (ks) play?',
 '[start] SELECT Position FROM table WHERE School/Club Team = Butler CC (KS)
[end]')
```

[13]:
```
#counting input and target lengths so that I can make sure im setting␣
 ↪approriate token size, using split to count words opposed to characters
input_word_lengths = [len(example[0].split()) for example in␣
 ↪formatted_train_data]
target_word_lengths = [len(example[1].split()) for example in␣
 ↪formatted_train_data]
```

[14]:
```
print("Input Lengths:")
print("   Min:", min(input_word_lengths))
print("   Max:", max(input_word_lengths))
print("   Mean:", sum(input_word_lengths) / len(input_word_lengths))

print("\nTarget Lengths:")
print("   Min:", min(target_word_lengths))
print("   Max:", max(target_word_lengths))
print("   Mean:", sum(target_word_lengths) / len(target_word_lengths))
```

```
Input Lengths:
   Min: 3
   Max: 44
   Mean: 11.637654156685299

Target Lengths:
   Min: 6
   Max: 84
   Mean: 13.653233963268566
```

[15]:
```
from collections import Counter
import re

#Need to find how many unique tokens are in training set to ensure I set an␣
 ↪appropriate vocab size when training my models

def tokenize(text):
    text = text.lower()
```

```
    tokens = text.split()
    return tokens


word_count = Counter()

for input_text, target_text in formatted_train_data:
    word_count.update(tokenize(input_text))
    word_count.update(tokenize(target_text))


total_unique_words = len(word_count)

print(total_unique_words)
```

```
66232
```

[16]:
```
import numpy as np

#Further inspecting data by looking at what my max input text looks like and␣
 ↪inspecting what my max SQL output query looks like

max_input_index = np.argmax(input_word_lengths)

max_target_index = np.argmax(target_word_lengths)

print("Index of Max Input Length:", max_input_index)
print("Max Input Length:", input_word_lengths[max_input_index])

print("\nIndex of Max Target Length:", max_target_index)
print("Max Target Length:", target_word_lengths[max_target_index])
```

```
Index of Max Input Length: 25185
Max Input Length: 44

Index of Max Target Length: 8435
Max Target Length: 84
```

[17]:
```
formatted_train_data[25185]
```

[17]:
```
('What is the sum of Natural Change, when Natural Change (Per 1000) is greater
than 5.4, when Crude Death Rate (Per 1000) is less than 14.3, when Live Births
is less than 6,950, and when Crude Birth Rate (Per 1000) is less than 28.2?',
 '[start] SELECT SUM Natural change FROM table WHERE Natural change (per 1000) >
5.4 AND Crude death rate (per 1000) < 14.3 AND Live births < 6,950 AND Crude
birth rate (per 1000) < 28.2 [end]')
```

[18]:
```
formatted_train_data[8435]
```

[18]:
```
('List all the locations where net capacity is 950?',
 '[start] SELECT Location Chernobyl 1 Chernobyl 2 Chernobyl 3 Chernobyl 4
Chernobyl 5 Ignalina 1 Ignalina 2 Ignalina 3 Kursk 1 Kursk 2 Kursk 3 Kursk 4
Kursk 5 Kursk 6 Leningrad 1 Leningrad 2 Leningrad 3 Leningrad 4 Smolensk 1
Smolensk 2 Smolensk 3 Smolensk 4 Directorate for Construction of Kostoma NPP
(For Kostroma 1 and 2) Table 31. Technology and Soviet Energy Availability -
November 1981 - NTIS order #PB82-133455 (For Ignalina 4) FROM table WHERE Net
Capacity (MW) = 950 [end]')
```

[19]:
```
import tensorflow as tf
import tensorflow.keras.layers as layers
import string

#Vectorizing the input english and output SQL strings

strip_chars = string.punctuation
strip_chars = strip_chars.replace("[", "")
strip_chars = strip_chars.replace("]", "")

def custom_standardization(input_string):
    lowercase = tf.strings.lower(input_string)
    return tf.strings.regex_replace(
        lowercase, f"[{re.escape(strip_chars)}]", "")

vocab_size = 65000
sequence_length = 40
#These were chosen from the above data exploration


source_vectorization = layers.TextVectorization(
    max_tokens=vocab_size,
    output_mode="int",
    output_sequence_length=sequence_length,
)
target_vectorization = layers.TextVectorization(
    max_tokens=vocab_size,
    output_mode="int",
    output_sequence_length=sequence_length + 1,
    standardize=custom_standardization,
)
train_input_text = [example[0] for example in formatted_train_data]
train_output_sql = [example[1] for example in formatted_train_data]
source_vectorization.adapt(train_input_text)
target_vectorization.adapt(train_output_sql)
```

```
2024-02-26 15:02:24.584867: I external/local_tsl/tsl/cuda/cudart_stub.cc:31]
Could not find cuda drivers on your machine, GPU will not be used.
2024-02-26 15:02:25.378438: E
```

```
external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9261] Unable to register
cuDNN factory: Attempting to register factory for plugin cuDNN when one has
already been registered
2024-02-26 15:02:25.379385: E
external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:607] Unable to register
cuFFT factory: Attempting to register factory for plugin cuFFT when one has
already been registered
2024-02-26 15:02:25.549011: E
external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1515] Unable to
register cuBLAS factory: Attempting to register factory for plugin cuBLAS when
one has already been registered
2024-02-26 15:02:25.901590: I external/local_tsl/tsl/cuda/cudart_stub.cc:31]
Could not find cuda drivers on your machine, GPU will not be used.
2024-02-26 15:02:25.905620: I tensorflow/core/platform/cpu_feature_guard.cc:182]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild
TensorFlow with the appropriate compiler flags.
```

[20]:
```
batch_size = 128
#My data set is fairly large, so a batch size of 128 helps speed up the␣
 ↪training process opposed to a smaller batch size. Since the set is so large␣
 ↪not worried about loosing out on possible learned features from using␣
 ↪smaller batch size

#Formating the vectorized data to be sent through my model for training and␣
 ↪translation

def format_dataset(text, sql):
    text = source_vectorization(text)
    sql = target_vectorization(sql)
    return ({
        "text input": text,
        "sql output": sql[:, :-1],
    }, sql[:, 1:])

def make_dataset(data):
    input_text, output_sql = zip(*data)
    input_text = list(input_text)
    output_sql = list(output_sql)
    dataset = tf.data.Dataset.from_tensor_slices((input_text, output_sql))
    dataset = dataset.batch(batch_size)
    dataset = dataset.map(format_dataset, num_parallel_calls=4)
    return dataset.shuffle(2048).prefetch(16).cache()

train_ds = make_dataset(formatted_train_data)
val_ds = make_dataset(formatted_val_data)
```

[21]:
```
train_ds
```

[21]:
```
<CacheDataset element_spec=({'text input': TensorSpec(shape=(None, 40),
dtype=tf.int64, name=None), 'sql output': TensorSpec(shape=(None, 40),
dtype=tf.int64, name=None)}, TensorSpec(shape=(None, 40), dtype=tf.int64,
name=None))>
```

[22]:
```
#ensuring that the input and output shapes align

for inputs, targets in train_ds.take(1):
    print(f"inputs['text input'].shape: {inputs['text input'].shape}")
    print(f"inputs['sql output'].shape: {inputs['sql output'].shape}")
    print(f"targets.shape: {targets.shape}")
```

```
inputs['text input'].shape: (128, 40)
inputs['sql output'].shape: (128, 40)
targets.shape: (128, 40)
```

```
2024-02-26 15:02:37.154529: W
tensorflow/core/kernels/data/cache_dataset_ops.cc:858] The calling iterator did
not fully read the dataset being cached. In order to avoid unexpected truncation
of the dataset, the partially cached contents of the dataset  will be discarded.
This can happen if you have an input pipeline similar to
`dataset.cache().take(k).repeat()`. You should use
`dataset.take(k).cache().repeat()` instead.
```

0.1.5 Basic Sequence-to-Sequence with RNN Model (Base model)

Architecture based on chapter 11 of Deep Learning with Python, Second Edition, by Francois Chollet

[23]:
```
#encoder set up (based on chapter 11 DLwP)
from tensorflow import keras

embed_dim = 256
latent_dim = 1024

source = keras.Input(shape=(None,), dtype="int64", name="text input")
x = layers.Embedding(vocab_size, embed_dim, mask_zero=True)(source)
encoded_source = layers.Bidirectional(
    layers.GRU(latent_dim), merge_mode="sum")(x)
```

Research Reference and Reasoning Behind Dropout Layer:
In the academic paper "Dropout: A Simple Way to Prevent Neural Networks from Overfitting" the researchers from the University of Toronto study the effectiveness of a dropout layer as a regularization technique in neural networks. This process involves dropping out a subset of nodes in the network during its training, in hopes of preventing nodes from co-adapting too closely to the specific training data, which could lead to a model that would not generalize well to new data. The study found that incorporating a dropout layer showed substantial empirical evidence of mitigating overfitting and enhancing generalization over a variety of tasks.

Although my base reference model is rather simple, adding a dropout layer will be a strong way to improve its generalization and robustness against overfitting. I will look to deploy the technique to both my base model and advanced model as given the complexity of translating plain English to SQL code, my models would be particularly susceptible to overfitting.

Source: Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research, 15, 1929-1958.

[24]:
```
#decoder set up (based on chapter 11 DLwp)

#embedding layer
past_target = keras.Input(shape=(None,), dtype="int64", name="sql output")
x = layers.Embedding(vocab_size, embed_dim, mask_zero=True)(past_target)
#GRU unit to process embeded sql outputs
decoder_gru = layers.GRU(latent_dim, return_sequences=True)
x = decoder_gru(x, initial_state=encoded_source)
#dropout layer to assist with regulization
x = layers.Dropout(0.5)(x)
target_next_step = layers.Dense(vocab_size, activation="softmax")(x)
base_seq2seq_rnn = keras.Model([source, past_target], target_next_step)
```

[25]:
```
#training my base rnn (with gru) model
from tensorflow.keras.callbacks import ModelCheckpoint

base_seq2seq_rnn.compile(
    optimizer="rmsprop",
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"])

base_seq2seq_rnn.summary()
```

```
Model: "model"
__________________________________________________________________________________________________
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================================
 text input (InputLayer)     [(None, None)]            0         []

 sql output (InputLayer)     [(None, None)]            0         []

 embedding (Embedding)       (None, None, 256)         1664000   ['text
input[0][0]']
                                                       0

 embedding_1 (Embedding)     (None, None, 256)         1664000   ['sql
output[0][0]']
                                                       0
```

```
 bidirectional (Bidirection  (None, 1024)                7876608
['embedding[0][0]']
 al)

 gru_1 (GRU)                 (None, None, 1024)          3938304
['embedding_1[0][0]',
'bidirectional[0][0]']

 dropout (Dropout)           (None, None, 1024)          0
['gru_1[0][0]']

 dense (Dense)               (None, None, 65000)         6662500
['dropout[0][0]']
                                                         0

==========================================================================
==================
Total params: 111719912 (426.18 MB)
Trainable params: 111719912 (426.18 MB)
Non-trainable params: 0 (0.00 Byte)
--------------------------------------------------------------------------
------------------
```

[26]:
```python
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

# EarlyStopping callback to stop training if no improvement in val_loss
early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=3,  # after 3 epochs with no val loss improvment it will stop and␣
 ↪save the model
    verbose=1,
    restore_best_weights=True
)

model_checkpoint = ModelCheckpoint(
    filepath='base_seq_to_seq_model.keras',
    monitor='val_loss',
    save_best_only=True,
    verbose=1
)

callbacks = [early_stopping, model_checkpoint]
```

[ ]:
```python
base_seq2seq_rnn.fit(train_ds, epochs=15, validation_data=val_ds,␣
 ↪callbacks=callbacks)
```

```
Epoch 1/15
```

```
441/441 [==============================] - ETA: 0s - loss: 4.3288 - accuracy:
0.4708
Epoch 1: val_loss improved from inf to 3.72040, saving model to
base_seq_to_seq_model.keras
441/441 [==============================] - 2316s 5s/step - loss: 4.3288 -
accuracy: 0.4708 - val_loss: 3.7204 - val_accuracy: 0.5152
Epoch 2/15
441/441 [==============================] - ETA: 0s - loss: 3.6853 - accuracy:
0.5255
Epoch 2: val_loss improved from 3.72040 to 3.57576, saving model to
base_seq_to_seq_model.keras
441/441 [==============================] - 2289s 5s/step - loss: 3.6853 -
accuracy: 0.5255 - val_loss: 3.5758 - val_accuracy: 0.5139
Epoch 3/15
441/441 [==============================] - ETA: 0s - loss: 3.4920 - accuracy:
0.5355
Epoch 3: val_loss improved from 3.57576 to 3.40280, saving model to
base_seq_to_seq_model.keras
441/441 [==============================] - 2293s 5s/step - loss: 3.4920 -
accuracy: 0.5355 - val_loss: 3.4028 - val_accuracy: 0.5450
Epoch 4/15
339/441 [======================>...] - ETA: 8:20 - loss: 3.0511 - accuracy:
0.5635
```

[26]:
```
#appears to be glicth in epoch print out but model did complete training after␣
↪about 9 hours
```

[27]:
```
base_loaded_rnn = tf.keras.models.load_model('base_seq_to_seq_model.keras')
```

[28]:
```
base_loaded_rnn.summary()
```

```
Model: "model"
__________________________________________________________________________________
__________________
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================
==================
 text input (InputLayer)     [(None, None)]            0         []

 sql output (InputLayer)     [(None, None)]            0         []

 embedding (Embedding)       (None, None, 256)         1664000   ['text
input[0][0]']
                                                       0

 embedding_1 (Embedding)     (None, None, 256)         1664000   ['sql
output[0][0]']
                                                       0
```

```
 bidirectional (Bidirection  (None, 1024)                7876608
['embedding[0][0]']
 al)

 gru_1 (GRU)                 (None, None, 1024)          3938304
['embedding_1[0][0]',
'bidirectional[0][0]']

 dropout (Dropout)           (None, None, 1024)          0
['gru_1[0][0]']

 dense (Dense)               (None, None, 65000)         6662500
['dropout[0][0]']
                                                         0

==========================================================================
==================
Total params: 111719912 (426.18 MB)
Trainable params: 111719912 (426.18 MB)
Non-trainable params: 0 (0.00 Byte)
--------------------------------------------------------------------------
------------------
```

[39]:

```python
#Translating some test data to asses initial perfomance

import numpy as np
import random

sql_target = target_vectorization.get_vocabulary()
sql_index_lookup = dict(zip(range(len(sql_target)), sql_target))
max_decoded_sentence_length = 15

def decode_sequence(input_sentence):
    tokenized_input_sentence = source_vectorization([input_sentence])
    decoded_sentence = "[start]"
    for i in range(max_decoded_sentence_length):
        tokenized_target_sentence = target_vectorization([decoded_sentence])
        next_token_predictions = base_loaded_rnn.predict(
            [tokenized_input_sentence, tokenized_target_sentence], verbose = 0)
        sampled_token_index = np.argmax(next_token_predictions[0, i, :])
        sampled_token = sql_index_lookup[sampled_token_index]
        decoded_sentence += " " + sampled_token
        if sampled_token == "[end]":
            break
    return decoded_sentence
```

```
test_sql_inputs = [example[0] for example in formatted_test_data]
for _ in range(5):
    input_sentence = random.choice(test_sql_inputs)
    print("-")
    print(input_sentence)
    print(decode_sequence(input_sentence))
```

```
-
Who had the high points at the United Center 20,389?
[start] select high points from table where location san francisco united [end]
-
When did the consort who had talal i as a spouse die?
[start] select date of birth from table where name of the church of the the the
-
What was the result in a year before 2013 that the nomination category was
Presenter Talent Show?
[start] select result from table where year ceremony 1998 and venue of the final
february in
-
Which Number of households has a Median household income of $32,806, and a
Population larger than 11,291?
[start] select sum number of households from table where area km² per km 2 and
area
-
How many poles does the European F3 Open series have?
[start] select poles from table where season premiere april 28 [end]
```

This base model appears to do a decent job at understanding the general idea of a SQL query in relation to plain english. There seems to be some difficulty in generating meaningful output between SQL functions such as tables and column names, but since the model was not trained with table and column inputs this would be a difficult task. It does seem to do an okay job at bootstrapping table and column names from input info, these are not always meaningful though.

0.1.6 Model Assesment

Going to asses how well the generated SQL creates the proper components required for the query, using SELECT, FROM, WHERE, JOIN, MIN, MAX

Assessing generated text accuracy can be rather complicated as it involves a multitude of complex criteria such as subjectivity, context, and comparability, meaning there can be many correct answers for the same generated output. To try and eliminate this complexity and measure the outcomes based on the main purpose of my project, to assist non-technical individuals in getting the proper SQL queries/functions, I believe that tracking the model's ability to produce the proper primary SQL functions is the most important. This does ignore measuring proper order and context but does measure that the model is equipping a possible user with the right tools to push them closer to generating a SQL query. It is noted that a more accurate way of testing the model would be to capture the whole output context, order, and quality against the ground truth SQL query.

This assessment works by checking and counting the presence of key SQL functions: "select", "from", "where", "count", "join", "min", and "max" in the generated output, and checking if these match what is in the ground truth query from the testing set, for example, if the ground truth

query has “select”, “where” and “count” and the generated query only has “select” and “where”, then it would score 2/3.

[40]:
```
def check_keyword_presence(sql_query):
    keywords = ["select", "from", "where", "count", "join", "min", "max"]
    presence = {keyword: keyword in sql_query.lower() for keyword in keywords}
    return presence
```

[41]:
```
def compare_keyword_presence(generated_presence, truth_presence):
    correct_count = sum(generated_presence[key] == truth_presence[key] for key␣
 ↪in generated_presence)
    return correct_count
```

[42]:
```
import random


sample_size = 500
sampled_test_data = random.sample(formatted_test_data, sample_size)

total_correct_counts = 0
total_keywords = 7 * len(sampled_test_data)

for input_text, ground_truth_sql in sampled_test_data:
    generated_sql = decode_sequence(input_text)

    generated_presence = check_keyword_presence(generated_sql)
    truth_presence = check_keyword_presence(ground_truth_sql)

    correct_count = compare_keyword_presence(generated_presence, truth_presence)
    total_correct_counts += correct_count


average_accuracy = total_correct_counts / total_keywords
print(f"Average Keyword Matching Accuracy: {average_accuracy:.4f}")
```

```
Average Keyword Matching Accuracy: 0.9689
```

0.1.7 Transformer Sequence-to-Sequence Model with Positional Embedding

Research Reference and Reasoning Behind Choice of Transformer Model:
In the paper, “Attention is all you need” written by a team of Google’s highly accomplished researchers, the transformer model was introduced. They stated by exclusively utilizing attention mechanisms a transformer achieves superior performance in translation tasks, including increased efficiency and a reduction in training time (something which would be very beneficial as my previous base model required around 45 minutes per epoch). The self-attention mechanism of the model which acted to weigh the importance of different parts of input data proved to allow for a more flexible way to adjust the model’s focus. A big innovation within the attention mechanism was their findings on the Multi-Head attention mechanism. Opposed to performing a single attention

operation, the multi-head goes through the attention mechanism serval times in parallel with different learned linear projections, this in turn drastically enhances the model's ability to focus on different parts of a sequence and understand complex dependencies.
This is highly applicable to my goal of a Text-to-SQL model as being able to capture the complex dependencies of the text input and SQL output without considering their positions directly in sequence makes the essential component of capturing the relationships between different parts of the text and queries easier. The transformer will be able to dynamically focus on the relevant parts of the text and query which is a complicated task as there are many underlying ways plain English can elude to the same SQL query.
Source: Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention Is All You Need. In Advances in Neural Information Processing Systems (pp. 5998–6008). 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA

Architecture based on chapter 11 of Deep Learning with Python, Second Edition, by Francois Chollet

[64]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

class TransformerEncoder(layers.Layer):
    def __init__(self, embed_dim, dense_dim, num_heads, **kwargs):
        super().__init__(**kwargs)
        self.embed_dim = embed_dim
        self.dense_dim = dense_dim
        self.num_heads = num_heads
        self.attention = layers.MultiHeadAttention(
            num_heads=num_heads, key_dim=embed_dim)
        self.dense_proj = keras.Sequential(
            [layers.Dense(dense_dim, activation="relu"),
             layers.Dense(embed_dim),]
        )
        self.layernorm_1 = layers.LayerNormalization()
        self.layernorm_2 = layers.LayerNormalization()

    def call(self, inputs, mask=None):
        if mask is not None:
            mask = mask[:, tf.newaxis, :]
        attention_output = self.attention(
            inputs, inputs, attention_mask=mask)
        proj_input = self.layernorm_1(inputs + attention_output)
        proj_output = self.dense_proj(proj_input)
        return self.layernorm_2(proj_input + proj_output)

    def get_config(self):
        config = super().get_config()
        config.update({
```

```
            "embed_dim": self.embed_dim,
            "num_heads": self.num_heads,
            "dense_dim": self.dense_dim,
        })
        return config
```

[65]:
```
class TransformerDecoder(layers.Layer):
    def __init__(self, embed_dim, dense_dim, num_heads, **kwargs):
        super().__init__(**kwargs)
        self.embed_dim = embed_dim
        self.dense_dim = dense_dim
        self.num_heads = num_heads
        self.attention_1 = layers.MultiHeadAttention(
            num_heads=num_heads, key_dim=embed_dim)
        self.attention_2 = layers.MultiHeadAttention(
            num_heads=num_heads, key_dim=embed_dim)
        self.dense_proj = keras.Sequential(
            [layers.Dense(dense_dim, activation="relu"),
             layers.Dense(embed_dim),]
        )
        self.layernorm_1 = layers.LayerNormalization()
        self.layernorm_2 = layers.LayerNormalization()
        self.layernorm_3 = layers.LayerNormalization()
        self.supports_masking = True

    def get_config(self):
        config = super().get_config()
        config.update({
            "embed_dim": self.embed_dim,
            "num_heads": self.num_heads,
            "dense_dim": self.dense_dim,
        })
        return config

    def get_causal_attention_mask(self, inputs):
        input_shape = tf.shape(inputs)
        batch_size, sequence_length = input_shape[0], input_shape[1]
        i = tf.range(sequence_length)[:, tf.newaxis]
        j = tf.range(sequence_length)
        mask = tf.cast(i >= j, dtype="int32")
        mask = tf.reshape(mask, (1, input_shape[1], input_shape[1]))
        mult = tf.concat(
            [tf.expand_dims(batch_size, -1),
             tf.constant([1, 1], dtype=tf.int32)], axis=0)
        return tf.tile(mask, mult)

    def call(self, inputs, encoder_outputs, mask=None):
```

```
        causal_mask = self.get_causal_attention_mask(inputs)
        if mask is not None:
            padding_mask = tf.cast(
                mask[:, tf.newaxis, :], dtype="int32")
            padding_mask = tf.minimum(padding_mask, causal_mask)
        attention_output_1 = self.attention_1(
            query=inputs,
            value=inputs,
            key=inputs,
            attention_mask=causal_mask)
        attention_output_1 = self.layernorm_1(inputs + attention_output_1)
        attention_output_2 = self.attention_2(
            query=attention_output_1,
            value=encoder_outputs,
            key=encoder_outputs,
            attention_mask=padding_mask,
        )
        attention_output_2 = self.layernorm_2(
            attention_output_1 + attention_output_2)
        proj_output = self.dense_proj(attention_output_2)
        return self.layernorm_3(attention_output_2 + proj_output)
```

[66]:
```
class PositionalEmbedding(layers.Layer):
    def __init__(self, sequence_length, input_dim, output_dim, **kwargs):
        super().__init__(**kwargs)
        self.token_embeddings = layers.Embedding(
            input_dim=input_dim, output_dim=output_dim)
        self.position_embeddings = layers.Embedding(
            input_dim=sequence_length, output_dim=output_dim)
        self.sequence_length = sequence_length
        self.input_dim = input_dim
        self.output_dim = output_dim

    def call(self, inputs):
        length = tf.shape(inputs)[-1]
        positions = tf.range(start=0, limit=length, delta=1)
        embedded_tokens = self.token_embeddings(inputs)
        embedded_positions = self.position_embeddings(positions)
        return embedded_tokens + embedded_positions

    def compute_mask(self, inputs, mask=None):
        return tf.math.not_equal(inputs, 0)

    def get_config(self):
        config = super(PositionalEmbedding, self).get_config()
        config.update({
            "output_dim": self.output_dim,
```

```
            "sequence_length": self.sequence_length,
            "input_dim": self.input_dim,
        })
        return config
```

[67]:
```
embed_dim = 256
dense_dim = 2048
num_heads = 8

encoder_inputs = keras.Input(shape=(None,), dtype="int64", name="text input")
x = PositionalEmbedding(sequence_length, vocab_size, embed_dim)(encoder_inputs)
encoder_outputs = TransformerEncoder(embed_dim, dense_dim, num_heads)(x)

decoder_inputs = keras.Input(shape=(None,), dtype="int64", name="sql output")
x = PositionalEmbedding(sequence_length, vocab_size, embed_dim)(decoder_inputs)
x = TransformerDecoder(embed_dim, dense_dim, num_heads)(x, encoder_outputs)
x = layers.Dropout(0.5)(x)
decoder_outputs = layers.Dense(vocab_size, activation="softmax")(x)
transformer = keras.Model([encoder_inputs, decoder_inputs], decoder_outputs)


transformer.compile(
    optimizer="adam",
    loss="sparse_categorical_crossentropy",
    metrics=["accuracy"])
transformer.summary()
```

```
Model: "model_2"
__________________________________________________________________________________
__________________
 Layer (type)                Output Shape              Param #   Connected to
==================================================================================
==================
 text input (InputLayer)     [(None, None)]            0         []

 sql output (InputLayer)     [(None, None)]            0         []

 positional_embedding_2 (Po  (None, None, 256)         1665024   ['text
input[0][0]']
 sitionalEmbedding)                                    0

 positional_embedding_3 (Po  (None, None, 256)         1665024   ['sql
output[0][0]']
 sitionalEmbedding)                                    0

 transformer_encoder_1 (Tra  (None, None, 256)         3155456
['positional_embedding_2[0][0]
 nsformerEncoder)                                                 ']
```

```
 transformer_decoder_1 (Tra  (None, None, 256)            5259520
['positional_embedding_3[0][0]
 nsformerDecoder)                                                  ',
'transformer_encoder_1[0][0]'
                                                                   ]

 dropout_2 (Dropout)         (None, None, 256)            0
['transformer_decoder_1[0][0]'
                                                                   ]

 dense_14 (Dense)            (None, None, 65000)          1670500
['dropout_2[0][0]']
                                                          0

==================================================================================
==================
Total params: 58420456 (222.86 MB)
Trainable params: 58420456 (222.86 MB)
Non-trainable params: 0 (0.00 Byte)
----------------------------------------------------------------------------------
------------------
```

[68]:
```
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

#EarlyStopping callback to stop training if no improvement in val_loss
early_stopping = EarlyStopping(
    monitor='val_loss',
    patience=7,
    verbose=1,
    restore_best_weights=True
)

model_checkpoint = ModelCheckpoint(
    filepath='best_transformer_modelV5.keras',
    monitor='val_loss',
    save_best_only=True,
    verbose=1
)

callbacks = [early_stopping, model_checkpoint]
```

[ ]:
```
transformer.fit(train_ds, validation_data=val_ds, epochs=30,␣
 ↪callbacks=callbacks)
```

```
Epoch 1/30
441/441 [==============================] - ETA: 0s - loss: 3.7875 - accuracy:
0.5444
```

```
Epoch 1: val_loss improved from inf to 2.90097, saving model to
best_transformer_modelV5.keras
441/441 [==============================] - 1387s 3s/step - loss: 3.7875 -
accuracy: 0.5444 - val_loss: 2.9010 - val_accuracy: 0.6095
Epoch 2/30
441/441 [==============================] - ETA: 0s - loss: 2.5629 - accuracy:
0.6480
Epoch 2: val_loss improved from 2.90097 to 2.38395, saving model to
best_transformer_modelV5.keras
441/441 [==============================] - 1386s 3s/step - loss: 2.5629 -
accuracy: 0.6480 - val_loss: 2.3840 - val_accuracy: 0.6844
441/441 [==============================] - ETA: 0s - loss: 0.6027 - accuracy:
0.8682
Epoch 8: val_loss did not improve from 2.02048
441/441 [==============================] - 1384s 3s/step - loss: 0.6027 -
accuracy: 0.8682 - val_loss: 2.1173 - val_accuracy: 0.7690
Epoch 9/30
441/441 [==============================] - ETA: 0s - loss: 0.5013 - accuracy:
0.8868
Epoch 9: val_loss did not improve from 2.02048
441/441 [==============================] - 1369s 3s/step - loss: 0.5013 -
accuracy: 0.8868 - val_loss: 2.1570 - val_accuracy: 0.7727
Epoch 10/30
441/441 [==============================] - ETA: 0s - loss: 0.4289 - accuracy:
0.9003
Epoch 10: val_loss did not improve from 2.02048
441/441 [==============================] - 1389s 3s/step - loss: 0.4289 -
accuracy: 0.9003 - val_loss: 2.1812 - val_accuracy: 0.7715
Epoch 11/30
441/441 [==============================] - ETA: 0s - loss: 0.3755 - accuracy:
0.9112
Epoch 11: val_loss did not improve from 2.02048
441/441 [==============================] - 1396s 3s/step - loss: 0.3755 -
accuracy: 0.9112 - val_loss: 2.2332 - val_accuracy: 0.7713
Epoch 12/30
441/441 [==============================] - ETA: 0s - loss: 0.3320 - accuracy:
0.9200Restoring model weights from the end of the best epoch: 5.

Epoch 12: val_loss did not improve from 2.02048
441/441 [==============================] - 1400s 3s/step - loss: 0.3320 -
accuracy: 0.9200 - val_loss: 2.2746 - val_accuracy: 0.7752
Epoch 12: early stopping
```

[ ]:
```
<keras.src.callbacks.History at 0x7f0466c67970>
```

[70]:
```
from tensorflow.keras.models import load_model
```

```
# since I used custome objects in my model, need to identify them before␣
 ↪loading in the model to make sure the whole model is correctly/properly␣
 ↪loaded for testing
custom_objects = {
    'TransformerEncoder': TransformerEncoder,
    'TransformerDecoder': TransformerDecoder,
    'PositionalEmbedding': PositionalEmbedding,
}


loaded_model = load_model('best_transformer_modelV5.keras',␣
 ↪custom_objects=custom_objects)
```

[79]:
```
#translatting some test data into SQL outputs to see how the model performs at␣
 ↪surface level

import random

sql_target = target_vectorization.get_vocabulary()
sql_index_lookup = dict(zip(range(len(sql_target)), sql_target))
max_decoded_sentence_length = 20

def decode_sequence_2(input_sentence):
    tokenized_input_sentence = source_vectorization([input_sentence])
    decoded_sentence = "[start]"
    for i in range(max_decoded_sentence_length):
        tokenized_target_sentence = target_vectorization(
            [decoded_sentence])[:, :-1]
        predictions = loaded_model(
            [tokenized_input_sentence, tokenized_target_sentence])
        sampled_token_index = np.argmax(predictions[0, i, :])
        sampled_token = sql_index_lookup[sampled_token_index]
        decoded_sentence += " " + sampled_token
        if sampled_token == "[end]":
            break
    return decoded_sentence

test_eng_texts = [pair[0] for pair in formatted_test_data]
for _ in range(10):
    input_sentence = random.choice(test_eng_texts)
    print("-")
    print(input_sentence)
    print(f"generated SQL: ",decode_sequence_2(input_sentence))
```

```
-
At the match in windy hill, how much did the home team score?
generated SQL:  [start] select score from table where venue windy hill [end]
-
```

```
When the venue was punt road oval, what was the Home teams score?
generated SQL:  [start] select venue from table where venue punt road oval [end]
-
Which VIETNAMESE has a CHINESE of    / bùrúmìduō?
generated SQL:  [start] select hanyu pinyin from table where chinese simplified
   [end]
-
What team had 4 podiums?
generated SQL:  [start] select team from table where podiums 4 [end]
-
What was the film title used in nomination for the film directed by Gheorghe
Vitanidis?
generated SQL:  [start] select film title used in nomination table where
director bent christensen [end]
-
What is City, when IATA is "amm"?
generated SQL:  [start] select city from table where iata cxb [end]
-
How many wins did Prema Powerteam won with 240 points?
generated SQL:  [start] select wins from table where points 206 and driver
giancarlo fisichella [end]
-
What is Points For, when Losing Bonus is "2", when Lost is "8", and when Club is
"Rhyl And District RFC"?
generated SQL:  [start] select points from table where losing bonus 2 and club
iserlohn roosters and wins 8 [end]
-
What is the players name in the guard position?
generated SQL:  [start] select player from table where position guard [end]
-
What's the record in the game in which Brad Miller (7) did the high rebounds?
generated SQL:  [start] select record from table where high rebounds brad miller
7 [end]
```

The transformer model, though more advanced appears to do a comparable job at generating meaningful SQL functions, and struggles to produce meaningful outputs between main functions.

Assesing Transformer Model:

[80]:
```python
def check_keyword_presence(sql_query):
    keywords = ["select", "from", "where", "count", "join", "min", "max"]
    presence = {keyword: keyword in sql_query.lower() for keyword in keywords}
    return presence
```

[81]:
```python
def compare_keyword_presence(generated_presence, truth_presence):
    correct_count = sum(generated_presence[key] == truth_presence[key] for key␣
 ↪in generated_presence)
    return correct_count
```

[82]:
```
import random


sample_size = 500
sampled_test_data = random.sample(formatted_test_data, sample_size)

total_correct_counts = 0
total_keywords = 7 * len(sampled_test_data)

for input_text, ground_truth_sql in sampled_test_data:
    generated_sql = decode_sequence_2(input_text)

    generated_presence = check_keyword_presence(generated_sql)
    truth_presence = check_keyword_presence(ground_truth_sql)

    correct_count = compare_keyword_presence(generated_presence, truth_presence)
    total_correct_counts += correct_count


average_accuracy = total_correct_counts / total_keywords
print(f"Average Keyword Matching Accuracy: {average_accuracy:.4f}")
```

```
Average Keyword Matching Accuracy: 0.9751
```

[78]:
```
print(decode_sequence_2("What is the population of canada"))
```

```
[start] select population from table where district canada [end]
```

There is very marginal improvment on the accurate generation of primary SQL fucntions, the Transformer model did provide some gains.

0.1.8 End Notes:

Overall the Transformer model was able to accurately produce key SQL functions from plain English inputs. It does struggle with complex context and producing meaningful statements between key functions, such as proper table, and column names. This issues could possibly be fixed if the model was trained purely on queries from specific databases, allowing it to properly learn what tables, columns and data are being referenced.
There was not a major considerable improvement with the advanced transformer model as opposed to the base RNN that utilized bidirectional and GRU units. There can be several factors at play as to why this may have happened, the primary factor may be that the short nature of these inputs and outputs does not allow the transformer to fully realize its advantages of understanding long-range dependencies. The structure of an RNN might be equally fitted to achieving this task as SQL queries tend be highly structured allowing it to learn the needed relationships.

This model would lay a foundation that can be built upon to start bridging the gap between an organizations technical and non technical teams, and empowering everyone with greater access to organizational data.