Assignment 3 Ungraded Sections - Part 2: T5 SQuAD Model

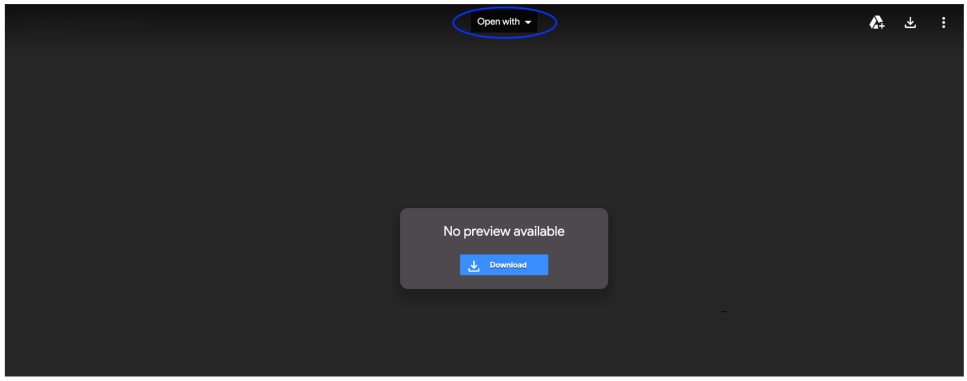
Welcome to the part 2 of testing the models for this week's assignment. This time we will perform decoding using the T5 SQuAD model. In this notebook we'll perform Question Answering by providing a "Question", its "Context" and see how well we get the "Target" answer.

Colab

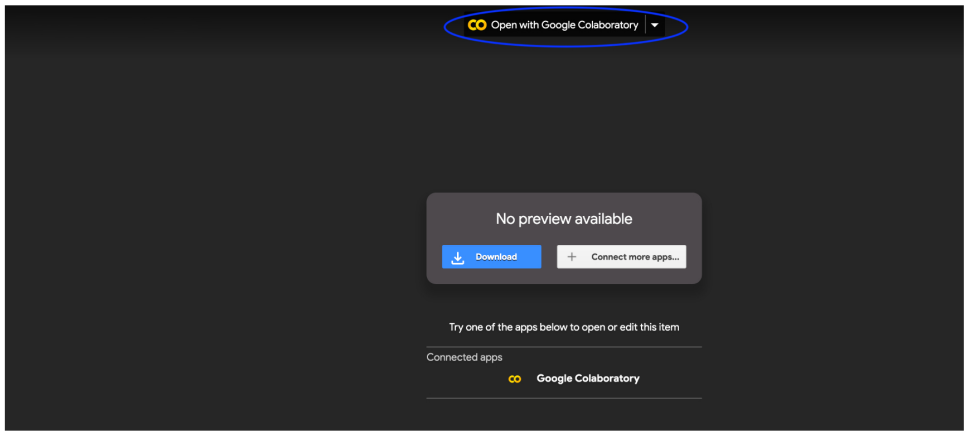
Since this ungraded lab takes a lot of time to run on coursera, as an alternative we have a colab prepared for you.

T5 SQuAD Model Colab

- If you run into a page that looks similar to the one below, with the option `Open with`, this would mean you need to download the `Colaboratory` app. You can do so by `Open with -> Connect more apps -> in the search bar write "Colaboratory" -> install`



- After installation it should look like this. Click on `Open with Google Colaboratory`



Outline

Overview

In this notebook you will:

- Implement the Bidirectional Encoder Representation from Transformer (BERT) loss.
- Use a pretrained version of the model you created in the assignment for inference.

Part 1: Getting ready

Run the code cells below to import the necessary libraries and to define some functions which will be useful for decoding. The code and the functions are the same as the ones you previsouly ran on the graded assignment.

In [ ]:
```
import string
import t5
import numpy as np
import trax
from trax.supervised import decoding
import textwrap

wrapper = textwrap.TextWrapper(width=70)
```

In [ ]:
```
PAD, EOS, UNK = 0, 1, 2


def detokenize(np_array):
    return trax.data.detokenize(
        np_array,
        vocab_type='sentencepiece',
        vocab_file='sentencepiece.model',
        vocab_dir='.')


def tokenize(s):
    return next(trax.data.tokenize(
        iter([s]),
        vocab_type='sentencepiece',
        vocab_file='sentencepiece.model',
        vocab_dir='.'))


vocab_size = trax.data.vocab_size(
    vocab_type='sentencepiece',
    vocab_file='sentencepiece.model',
    vocab_dir='.')


def get_sentinels(vocab_size, display=False):
    sentinels = {}
    for i, char in enumerate(reversed(string.ascii_letters), 1):
        decoded_text = detokenize([vocab_size - i])
        # Sentinels, ex: <Z> - <a>
        sentinels[decoded_text] = f'<{char}>'
        if display:
            print(f'The sentinel is <{char}> and the decoded token is:', decoded_text)
    return sentinels


sentinels = get_sentinels(vocab_size, display=False)


def pretty_decode(encoded_str_list, sentinels=sentinels):
    # If already a string, just do the replacements.
    if isinstance(encoded_str_list, (str, bytes)):
        for token, char in sentinels.items():
            encoded_str_list = encoded_str_list.replace(token, char)
        return encoded_str_list

    # We need to decode and then prettyfy it.
    return pretty_decode(detokenize(encoded_str_list))
```

Part 2: Fine-tuning on SQuAD

Now let's try to fine tune on SQuAD and see what becomes of the model.For this, we need to write a function that will create and process the SQuAD `tf.data.Dataset` . Below is how T5 pre-processes SQuAD dataset as a text2text example. Before we jump in, we will have to first load in the data.

2.1 Loading in the data and preprocessing

You first start by loading in the dataset. The text2text example for a SQuAD example looks like:

```
{
  'inputs': 'question: <question> context: <article>',
  'targets': '<answer_0>',
}
```

The squad pre-processing function takes in the dataset and processes it using the sentencePiece vocabulary you have seen above. It generates the features from the vocab and encodes the string features. It takes on question, context, and answer, and returns "question: Q context: C" as input and "A" as target.

In [ ]:
```
# Retrieve Question, C, A and return "question: Q context: C" as input and "A" as target.
def squad_preprocess_fn(dataset, mode='train'):
    return t5.data.preprocessors.squad(dataset)
```

In [ ]:
```
# train generator, this takes about 1 minute
train_generator_fn, eval_generator_fn = trax.data.tf_inputs.data_streams(
  'squad/plain_text:1.0.0',
  data_dir='data/',
  bare_preprocess_fn=squad_preprocess_fn,
  input_name='inputs',
  target_name='targets'
)

train_generator = train_generator_fn()
next(train_generator)
```

In [ ]:
```
#print example from train_generator
(inp, out) = next(train_generator)
print(inp.decode('utf8').split('context:')[0])
print()
print('context:', inp.decode('utf8').split('context:')[1])
print()
print('target:', out.decode('utf8'))
```

2.2 Decoding from a fine-tuned model

You will now use an existing model that we trained for you. You will initialize, then load in your model, and then try with your own input.

In [ ]:
```
# Initialize the model
model = trax.models.Transformer(
    d_ff = 4096,
    d_model = 1024,
    max_len = 2048,
    n_heads = 16,
    dropout = 0.1,
    input_vocab_size = 32000,
    n_encoder_layers = 24,
    n_decoder_layers = 24,
    mode='predict')  # Change to 'eval' for slow decoding.
```

In [ ]:
```
# Load in the model
# this will take a minute
shape11 = trax.shapes.ShapeDtype((1, 1), dtype=np.int32)
model.init_from_file('model_squad.pkl.gz',
                     weights_only=True, input_signature=(shape11, shape11))
```

In [ ]:
```
# create inputs
# a simple example
# inputs = 'question: She asked him where is john? context: John was at the game'

# an extensive example
inputs = 'question: What are some of the colours of a rose? context: A rose is a woody perennial flowering plant of the genus
```

In [ ]:
```
# tokenizing the input so we could feed it for decoding
print(tokenize(inputs))
test_inputs = tokenize(inputs)
```

Run the cell below to decode.

Note: This will take some time to run

In [ ]:
```
# Temperature is a parameter for sampling.
#   # * 0.0: same as argmax, always pick the most probable token
#   # * 1.0: sampling from the distribution (can sometimes say random things)
#   # * values inbetween can trade off diversity and quality, try it out!
output = decoding.autoregressive_sample(model, inputs=np.array(test_inputs)[None, :],
                                        temperature=0.0, max_length=5) # originally max_length=10
print(wrapper.fill(pretty_decode(output[0])))
```

You should also be aware that the quality of the decoding is not very good because max_length was downsized from 10 to 5 so that this runs faster within this environment. The colab version uses the original max_length so check that one for the actual decoding.