Assignment 3 Ungraded Sections - Part 1: BERT Loss Model

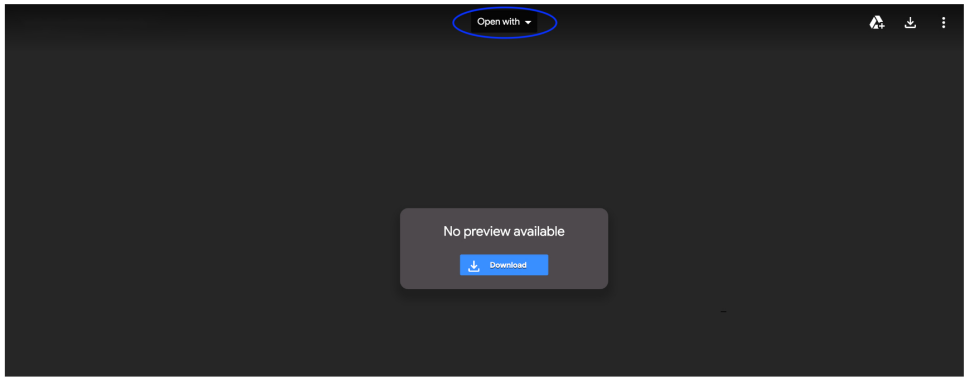
Welcome to the part 1 of testing the models for this week's assignment. We will perform decoding using the BERT Loss model. In this notebook we'll use an input, mask (hide) random word(s) in it and see how well we get the "Target" answer(s).

Colab

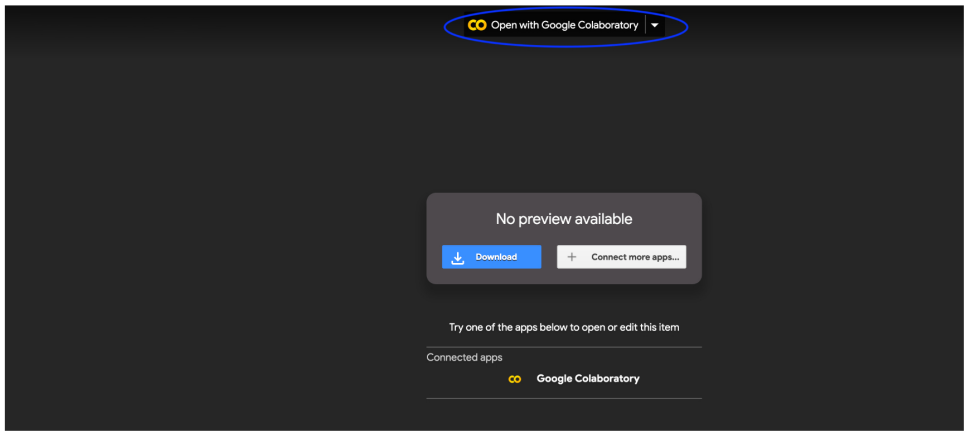
Since this ungraded lab takes a lot of time to run on coursera, as an alternative we have a colab prepared for you.

BERT Loss 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 pickle
import string
import ast
import numpy as np
import trax
from trax.supervised import decoding
import textwrap

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

In [ ]:
```
example_jsons = list(map(ast.literal_eval, open('data.txt')))
```

```
natural_language_texts = [example_json['text'] for example_json in example_jsons]

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))


inputs_targets_pairs = []

# here you are reading already computed input/target pairs from a file
with open ('inputs_targets_pairs_file.txt', 'rb') as fp:
    inputs_targets_pairs = pickle.load(fp)


def display_input_target_pairs(inputs_targets_pairs):
    for i, inp_tgt_pair in enumerate(inputs_targets_pairs, 1):
        inps, tgts = inp_tgt_pair
        inps, tgts = pretty_decode(inps), pretty_decode(tgts)
        print(f'[{i}]\n'
              f'inputs:\n{wrapper.fill(text=inps)}\n\n'
              f'targets:\n{wrapper.fill(text=tgts)}\n\n\n\n')

display_input_target_pairs(inputs_targets_pairs)
```

Part 2: BERT Loss

Now that you created the encoder, we will not make you train it. Training it could easily cost you a few days depending on which GPUs/TPUs you are using. Very few people train the full transformer from scratch. Instead, what the majority of people do, they load in a pretrained model, and they fine tune it on a specific task. That is exactly what you are about to do. Let's start by initializing and then loading in the model.

Initialize the model from the saved checkpoint.

In [ ]:
```
# Initializing 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')
```

In [ ]:
```
# Now load in the model
# this takes about 1 minute
shape11 = trax.shapes.ShapeDtype((1, 1), dtype=np.int32)  # Needed in predict mode.
model.init_from_file('model.pkl.gz',
                     weights_only=True, input_signature=(shape11, shape11))
```

2.1 Decoding

Now you will use one of the `inputs_targets_pairs` for input and as target. Next you will use the `pretty_decode` to output the input and target. The code to perform all of this has been provided below.

In [ ]:
```
# using the 3rd example
c4_input = inputs_targets_pairs[2][0]
c4_target = inputs_targets_pairs[2][1]

print('pretty_decoded input: \n\n', pretty_decode(c4_input))
print('\npretty_decoded target: \n\n', pretty_decode(c4_target))
print('\nc4_input:\n\n', c4_input)
print('\nc4_target:\n\n', c4_target)
print(len(c4_target))
print(len(pretty_decode(c4_target)))
```

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(c4_input)[None, :],
                                        temperature=0.0, max_length=5) # originally max_length = 50
print(wrapper.fill(pretty_decode(output[0])))
```

At this point the RAM is almost full, this happens because the model and the decoding is memory heavy. You can run decoding just once. Running it the second time with another example might give you an answer that makes no sense, or repetitive words. If that happens restart the runtime (see how to at the start of the notebook) and run all the cells again.

You should also be aware that the quality of the decoding is not very good because max_length was downsized from 50 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.