Project #4: Transformers with Tensorflow and Keras

Amir Sadovnik
COSC 424/525: Deep Learning (Spring 2023)

1 Overview

In this project you will be building a token based Transformer-based neural network to write Beatles songs. We will frame the problem as a many-to-many task, where you are trying to predict a series of words.

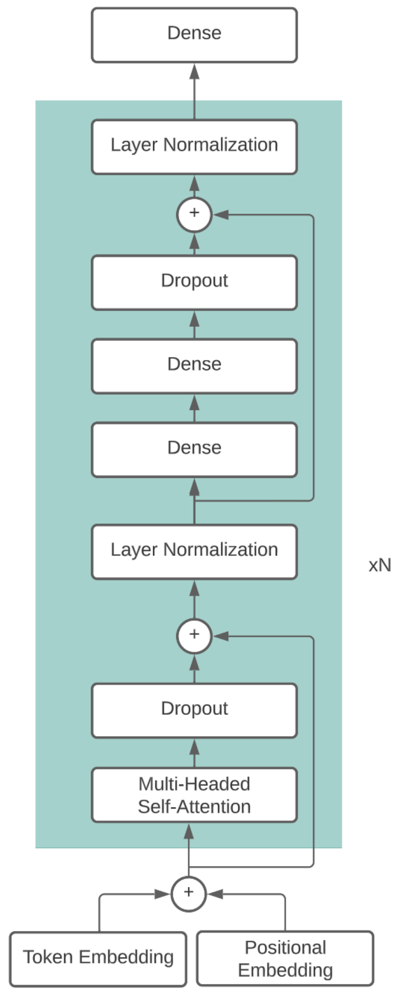
2 Dataset

You will be using a text file which includes lyrics from 246 Beatles songs. All the lyrics are concatenated with each other and will be treated as one long sequence (You do not need to break it up into separate songs). The text file is attached to the assignment on Canvas. The lyrics are taken from the following website: `http://beatlesnumber9.com/lyrics.html`.

3 Problem Description

3.1 Task 1: Implement the TransformerModel Class

1. Use the Jupyter Notebook attached as a file in canvas for structure in your code.
2. The TransformerModel class includes an init, TransformerBlock, EmbeddingLayer, and create_model method.
   - The init method will instantiate self attributes that match the arguments, including a vocabulary size embedding dimensions, number of heads for attention, number of dimensions of the dense layer in the block (ff_dim), a max length (in number of words/tokens), and the rate for dropout layers.



- The TransformerBlock method will be used to create a block of layers that follow the teal background in the figure. This includes several layers: a MultiHeadAttention, dropout, LayerNormalization layer, dense, dropout, sum, and LayerNormalization. Note the connections indicated in the figure that appear before both LayerNormalizations. The first sums the input to the block and the output from the first dropout. The second sums the output of the first LayerNormalization and second dropout.
- The EmbeddingLayer method will construct a layer that combines a token embedding and positional embedding, then sums the output of both. Note that while the input to the token embedding is the token_id the input to the position embedding is simply the position in the sentence.
- The create_model method will return overall model, made by first creating a layers.Input, then adding an EmbeddingLayer, then TransformerBlock, and finally a dense output layer with the same number of parameters as the size of the vocabulary. This method should return a compiled keras.Model and use tf.keras.losses.SparseCategoricalCrossentropy for its loss function.

3.2 Task 2: Implement the DataSet Class

This class is responsible for loading text and generating sequences for training.

1. The DataSet class will include methods init, prep_text, tokenize_text, and create_dataset.

    - The init method will instantiate self attributes of text (by reading in the file contents).
    - The prep_text method will make all text lowercase and remove special characters and apostrophes. It will also replace all whitespace characters (except for newlines) with spaces.
    - The tokenize_text method will set the text to a list of integers identified by a vocabulary. You may use np.unique().
    - The create_dataset method will call prep_text, tokenize_text, and then create and return the x, y, and vocabulary used to train your model. Here, each element of x is a sequence of integers (representing words) and y is offset forward by one, such that x and y have the same length.

3.3 Task 3: Implement a GenerateText Class

This class is responsible for using the model to generate text.

1. The GenerateText will include methods init, generate_text, and generate_random_text.

    - The init method will instantiates the reference to the model and vocabulary. It also create a mapping from the integer representation of tokens/words into a human-readable format.
    - The generate_text method will use a start string and generate a number of additional words. The start string should take in at least one word to initialize the beginning of the return sequence.
    - The generate_random_text method will be the same as generate_text, but provide random words from the vocabulary. This is useful as a baseline.

3.4 Task 4: Train and Qualitatively Evaluate the Model

Implement a train_model function that takes as input the vocabulary, training data, and number of epochs. This function has parameters for a model, vocabulary, x, y, and number of epochs. Use this to create different versions of your model so that you can qualitatively evaluate it. Create models for 1, 50, and 100 epochs as a minimum. For graduate students, try to include an additional model that has enough epochs to be overtrained (e.g. shows too many repeated words or phrases). You should start with different words or phrases to find where at least one of your model "breaks".

1.

4 Additional Information

1. A good Jupyter Notebooks can be re-run from top to bottom without breaking.
2. In Jupyter Notebooks, you can begin a cell with "%%time" to easily time how long it took a cell to run. Here is a link showing how to use it in a Jupyter Notebook
3. Exact learning parameters are not mentioned. You will need to select your own learning rate, momentum etc.
4. Please submit a Jupyter Notebook. Submit both the ipynb file in addition to an html file version. Make sure to design your Notebook such that it can be run from beginning to end without an error.

5 Report

Your Jupyter notebook should include all the following (use markup for text portions) :

1. A short introduction to the problem.
2. Introduction to the network you designed.
3. Results from different models (e.g. different number of epochs and attention heads). Include the calculated loss for each of your models. Be sure to use quotes from the generated models to compare, contrast, and generally evaluate the quality of the generated song content. What do you notice?
4. Conclusion - what did you observe when you ran these experiments?
5. How to run your code.

6 Submission

Please submit the html file and the ipynb file separately (i.e. not multiple files as a compressed zip).