Sequence To Sequence Model For Machine Translation

Introduction:

The goal of this project is to build a sequence-to-sequence (Seq2Seq) model with attention mechanism using TensorFlow for translating Spanish sentences to English sentences. Seq2Seq models are widely used in natural language processing (NLP) tasks, such as machine translation, summarization, and dialogue generation.

DataSet:

The dataset used for this project is the Spanish-English parallel corpus from the Open Parallel Corpus (OPUS). It contains over 6 million sentence pairs for English-Spanish translations. We will use the TensorFlow dataset API to load and pre-process the data.

WorkFlow:

Data Preparation:

Loaded the dataset into target raw and context raw: In this step, we acquired the dataset and loaded it into your program. Target raw refers to the English data, and context raw refers to the Spanish data. We made sure the data is in a suitable format for further processing.

Preprocess the data by removing special symbols and adding the Start of Sentence (SOS) and End of Sentence (EOS) tokens at the beginning and end of each statement: In this step, we preprocessed the data to make it suitable for modeling. Preprocessing involves removing any special characters or symbols that are not relevant to the task at hand. Then we added the SOS and EOS tokens at the beginning and end of each statement. These tokens help the model learn the beginning and end of each sentence and facilitate better language modeling.

Contraction Expansion:

Used a contraction file to expand all the contraction expressions in the dataset: Contraction expansion is the process of converting contractions (e.g., "don't" to "do not") to their full forms. This step is important because contractions are not always recognized as full words by tokenizers and models. Therefore, expanding contractions helps in better tokenization and improves model accuracy.

Tokenization:

Tokenized the target data (English) and context data (Spanish) using a suitable tokenizer: Tokenization involves breaking up the text into smaller units, such as words or subwords, to make it easier for the model to process. A suitable tokenizer needs to be chosen based on the specific task and language.

Data Splitting:

Divide the data into training and testing sets: In this step, we have split the data into training and testing sets to evaluate the model's performance. The training set is used to train the model, while the testing set is used to evaluate the model's performance on unseen data.

We then Padded the data to ensure uniform length across all sequences in the dataset: Paddings are added to the data to ensure that all the sequences have the same length. This is important for the model to process the data efficiently. Padding can be done by adding zeros at the end of the sequences until they have the same length.

Word Embedding:

Download pre-trained Word2Vec embeddings for English and GloVe embeddings for Spanish: Word embeddings are dense vector representations of words that capture their meaning and context. Applied the embeddings to the corresponding tokenized data to obtain dense vector representations for each word.

Encoder:

Input layer: The input layer takes the Spanish sentence as input. The shape of the input tensor is (51,), which is the maximum length of the Spanish sentence in the dataset.   SHAPE: (none,51). In place of none the no. of the spanish sentence will come.

Embedding layer: The input tensor is then passed through an embedding layer, which maps each word in the sentence to a vector representation. The embedding layer is initialized with pre-trained Spanish word embeddings, which are represented by the variable 'spanish_embedding_weights'. The length of the word embedding is 300. SHAPE: (none,51,300).

Bidirectional LSTM layer: The output of the embedding layer is passed through a bidirectional LSTM layer. The LSTM layer has 256 hidden units, which means that every words 300 embedding is connected to 256 hidden units for every time step. The LSTM layer is bidirectional, meaning that it processes the input sequence in both directions (forward and backward). This allows the layer to capture both the past and future context of each word in the input sequence. The 'return_sequences=True' parameter indicates that the LSTM layer should return the output for each time step of the input sequence, instead of just the final output.

Output of LSTM layer: The output of the LSTM layer is a tensor of shape (batch_size, max_spa_len, 512), where 512 is the size of the concatenated hidden state of the forward and backward LSTM cells (256+256=512). The last dimension represents the output features of the LSTM layer.

Concatenate both h and c: The forward and backward hidden states and cell states are concatenated to create the final hidden state and cell state respectively. This is done by concatenating the forward and backward states along the last dimension using the 'Concatenate()' function. The final hidden state and cell state are tensors of shape (batch_size, 512).

Context vector: The final hidden state and cell state are the context vector, which represents the semantic meaning of the input sequence. The context vector is passed to the decoder portion of the Seq2Seq model.

Encoder states: The final hidden state and cell state are stored in a list called 'encoder_states', which is used to initialize the decoder portion of the Seq2Seq model.

Decoder:

After processing the input sequence with the encoder, the decoder is responsible for generating the output sequence. The decoder takes the output sequence from the attention as input and generates the output. At each time step, the decoder takes in the previous context and the previous hidden state as input, and generates the next output word and hidden state.

Decoder Input: The decoder_inputs is an input tensor of shape (None,), where None represents variable-length input sequences. This means that the decoder can take in input sequences of any length.

Decoder Embedding: A decoder embedding layer is created with the same number of neurons as the encoder embedding layer. The decoder_inputs tensor is passed through the decoder embedding layer to get the decoder embedding tensor. The decoder embedding tensor is a 3D tensor of shape (batch_size, sequence_length, embedding_dimension), where sequence_length is the length of the longest sequence in the batch and embedding_dimension is the number of neurons in the decoder embedding layer.

Decoder LSTM: A decoder LSTM layer is created with 512 hidden units. This layer uses the encoder_states as initial state. The decoder LSTM layer takes the decoder embedding tensor as input and generates the decoder outputs, decoder hidden state, and decoder cell state at each time step.
The decoder_outputs tensor is a 3D tensor of shape (batch_size, sequence_length, 512), where sequence_length is the length of the longest sequence in the batch and 512 is the number of hidden units in the decoder LSTM layer.

Attention Layer: An attention layer is created using the AttentionLayer() function. This layer will be used to compute the attention scores between the decoder hidden state and the encoder outputs. The decoder_outputs tensor and encoder_outputs1 tensor are passed into the AdditiveAttention() function as inputs. The AdditiveAttention() function computes the attention scores using a feedforward neural network with two layers and returns the attention result tensor.

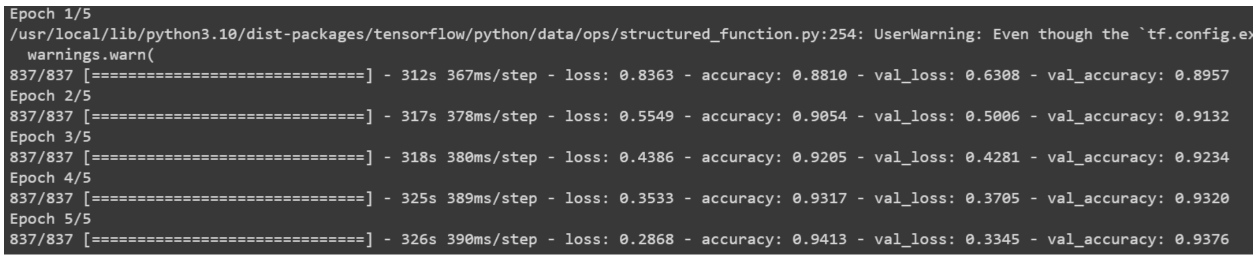
Concatenation: The attention result tensor is concatenated with the decoder_outputs tensor using the Concatenate() function.
This results in a tensor of shape (batch_size, sequence_length, 1024), where 1024 = 512 (hidden units in decoder LSTM layer) + 512 (hidden units in encoder LSTM layer after bidirectional).

Dense Layer: A dense layer with softmax activation is created with a number of neurons equal to the size of the English vocabulary.
The concatenated tensor is passed through the dense layer to generate the final output tensor. The final output tensor is a 3D tensor of shape (batch_size, sequence_length, VOCABULARY_SIZE_ENGLISH), where VOCABULARY_SIZE_ENGLISH is the size of the English vocabulary.

Model: Finally, a Keras model is created with the encoder_inputs, decoder_inputs, and decoder_outputs as inputs and outputs.
To summarize, the decoder part of the sequence-to-sequence model with attention takes the encoder outputs as input, and generates the output sequence one word at a time using the decoder LSTM layer with attention mechanism. The attention mechanism helps the decoder to focus on relevant parts of the encoder outputs while generating the output sequence.

Training:

We are then training the seq2seq model for 5 epocs in which the training accuracy is:



```
Epoch 1/5
/usr/local/lib/python3.10/dist-packages/tensorflow/python/data/ops/structured_function.py:254: UserWarning: Even though the `tf.config.ex
  warnings.warn(
837/837 [==============================] - 312s 367ms/step - loss: 0.8363 - accuracy: 0.8810 - val_loss: 0.6308 - val_accuracy: 0.8957
Epoch 2/5
837/837 [==============================] - 317s 378ms/step - loss: 0.5549 - accuracy: 0.9054 - val_loss: 0.5006 - val_accuracy: 0.9132
Epoch 3/5
837/837 [==============================] - 318s 380ms/step - loss: 0.4386 - accuracy: 0.9205 - val_loss: 0.4281 - val_accuracy: 0.9234
Epoch 4/5
837/837 [==============================] - 325s 389ms/step - loss: 0.3533 - accuracy: 0.9317 - val_loss: 0.3705 - val_accuracy: 0.9320
Epoch 5/5
837/837 [==============================] - 326s 390ms/step - loss: 0.2868 - accuracy: 0.9413 - val_loss: 0.3345 - val_accuracy: 0.9376
```

After training an Inference model is needed for testing as for training the gold_answer is available but for testing only the last state output is available. So, the decoder part of the model needs to be changed to take into account the difference.

Testing:

After making the inference model the testing accuracy is for some reason very low and we are not able to understand the problem because of which it is happening. We think it is because of the inference model that has some problems with it.