Deep Learning Lab - Assignment 4

Ufuk DOGAN

2018/12/21

1 Introduction and Preprocessing

In this assignment, our purpose is translating the Spanish words to the English. For this assignment, we used the LSTM.

Firstly, we downloaded the dataset which is needed for our traning process. However, using this dataset, we should process them. So, I did the preprocessing part of the assignment at the beginning. As we did previous assignments, we will needed the dictionary for converting the words to the integers and also converting back. That's why I needed to split the words to an array, but before this pretending any conflicts I lowecased the all sentences.

After splitting sentences to arrays, I added the <START> and <END> tokens to array, by this my model will understand the beginning and end of the sentence. Below, you can find screenshots of my spanish set and also, English set.

Figure 1: Spanish Sentence Set

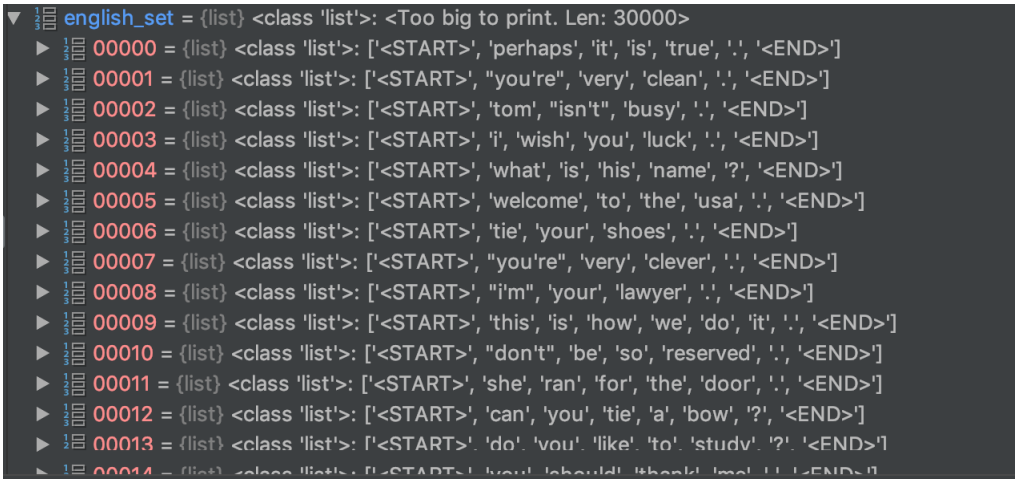


Figure 2: English Sentence Set

However, process does not enough for the model. I need the split the array elements and receive all words in the array, then create a dictionary for each language which contains the words and integers which corresponds to the words. Below, you can find the English language dictionary screenshot.
The advantages of these processes are,

- Specifying the beginning and the end of the word is useful to our model
- By this processes, we have the all data we will needed in the future parts
- This processes are for just one time job

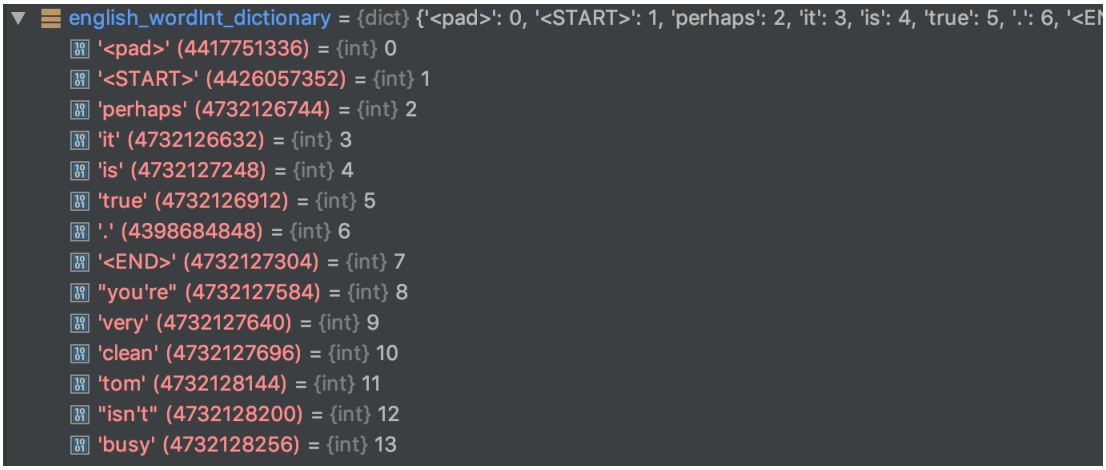


Figure 3: English Word to Int Dictionary

We have to use just 30.000 sentenes to train, validate and test our model. I thought that the training part should has the largest amount of the sentence that's why I used 18.000 sentence during the training, 9.000 sentence during validation and 3.000 sentence during the test.
For train set, I need to put together the words which I splitted. Below, there is a screenshot of my train set.

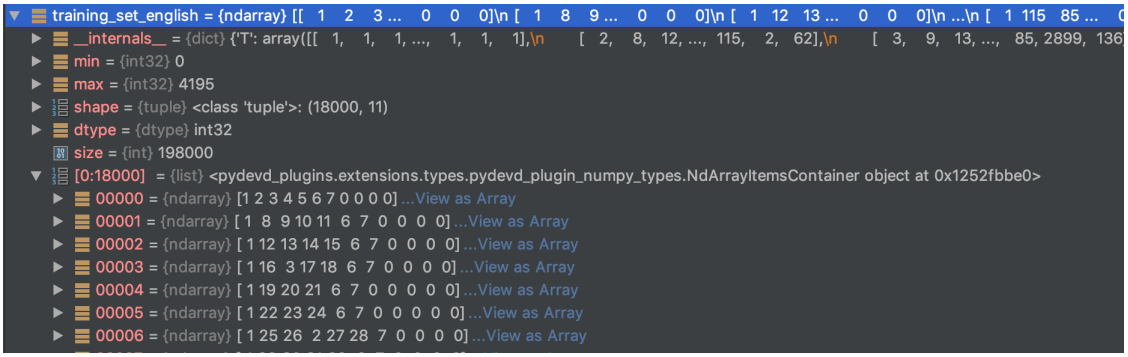


Figure 4: Training Dataset

Now, I should do the padding process. However, our sentences do not have the same length and it will cause a problem if we do not set a size. So, to solve this issue I calculated the largest sentence length of the English set and Spanish set. Then, I computed the length of the sentence, after all I substracted the length of the current sentence from the maximum length of the English or Spanish sentence. By this way, my sentences have the same lengths and I added $< PAD >$ token after my sentences ends. This is important because, if I did not do this, my model will not train due to the inconstintancy of the length of the targets.

By this I finished the preprocess.

2 Model

Our model is an encoder-decoder RNN. The encoder receives the input sentence and produce representations. On the other hand, decoder receives the ouput of the encoder as an output and produces the target sentence.

I applied two processes the decoder.

- Training
- Inferene

2.1 Encoder

The decoder during training needs the target sentence, the embeddings, encoder and decoder RNN.
Below, I added the encoder part of my code and it's explanation.

```
dynamic_batch_size = tf.shape(input_sentences_placeholder)[0]

word_embeddings_spanish = tf.get_variable("word_embeddings_spanish",
                                          shape=[vocabulary_size_spanish, embedding_size])

embedded_input_sentences_spanish_training = tf.nn.embedding_lookup(word_embeddings_spanish,
                                                                   input_sentences_placeholder)

word_embeddings_english = tf.get_variable("word_embeddings_english",
                                          shape=[vocabulary_size_english, embedding_size])

embedded_target_sentences_english_training = tf.nn.embedding_lookup(word_embeddings_english,
                                                                    target_sentences_placeholder)

encoder_cell = tf.contrib.cudnn_rnn.CudnnCompatibleLSTMCell(num_units=hidden_size)

encoder_outputs, encoder_final_state = tf.nn.dynamic_rnn(encoder_cell, embedded_input_sentences_spanish_training,
                                                         dtype=tf.float32)
```

Figure 5: Encoder Part of The Code

In here, first I embedded the length of the spanish and english dictionaries between embedding size which is 256. Then, I used the look up function in order to lookup a word based on my dictionary.
Encoder cell is a LSTM cell. Then, I recive the outputs of the encoder by the encoder cell.

2.2 Training Decoder

The decoder during training needs, the target sentence to train the embeddings, encoder and decoder RNNs.
Below, I added the training decoder part of my code and it's explanation.

```
decoder_input = embedded_target_sentences_english_training[:, :-1]

attention_mechanism = tf.contrib.seq2seq.BahdanauAttention(hidden_size, encoder_outputs)
decoder_cell = tf.contrib.cudnn_rnn.CudnnCompatibleLSTMCell(num_units=hidden_size)
attention_wrapper = tf.contrib.seq2seq.AttentionWrapper(decoder_cell, attention_mechanism, alignment_history=True)

decoder_init_state = attention_wrapper.zero_state(dynamic_batch_size, tf.float32).clone(cell_state=encoder_final_state)

training_decoder_helper = tf.contrib.seq2seq.TrainingHelper(decoder_input,
                                                            target_sentences_lengths_placeholder - 1)
projection_layer = tf.layers.Dense(vocabulary_size_english)

training_decoder = tf.contrib.seq2seq.BasicDecoder(cell=attention_wrapper,
                                                   helper=training_decoder_helper,
                                                   initial_state=decoder_init_state,
                                                   output_layer=projection_layer)

decoder_outputs, decoder_final_state, decoder_final_sequence_length = tf.contrib.seq2seq.dynamic_decode(training_decoder)
training_logits = decoder_outputs.rnn_output
train_predictions = decoder_outputs.sample_id

stack_logits_tensors = tf.stack([tf.shape(target_sentences_placeholder)[1], tf.shape(training_logits)[1]])
maximum_logit_tensor = stack_logits_tensors[tf.argmax(stack_logits_tensors)]
logit_difference = maximum_logit_tensor - tf.shape(training_logits)[1]

padded_training_logits = tf.pad(tensor=training_logits,
                                paddings=[[0, 0], [0, logit_difference], [0, 0]])

training_mask = tf.sequence_mask(lengths=target_sentences_lengths_placeholder - 1,
                                 dtype=tf.float32)

mask_difference = maximum_logit_tensor - tf.shape(training_mask)[1]

training_padded_mask = tf.pad(tensor=training_mask,
                              paddings=[[0, 0], [0, mask_difference]])

training_loss = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=target_sentences_placeholder,
                                                               logits=padded_training_logits)

training_loss = tf.reduce_sum((training_loss*training_padded_mask))/tf.reduce_sum(training_padded_mask)
```

Figure 6: Training Decoder Part of The Code

The decoder's input is the target sentence which is embedded in the encoder part of my code. Then I used the BahdanauAttention and AttentionWrapper in order to fill my traning decoder. Training decoder is a Basic Decoder and the outputs are,

- Decoder Outputs (rnn outputs and sample id)
- Decoder Final State
- Length of the Final Sequence

Before using these outputs, I padded the rnn outputs because of the same reason I mentioned above in preprocessing part. If I do not do the padding process, the outputs and the target's length will not match and I will receive an error. To pretend this, I padded the rnn outputs.

Before calculating the error of the traning decoder, I created a variable named mask. Mask helps to me about the computation of the average error over the predictions that we actually care about. Since I padded the outputs to a uniform length I have several predictions that are not of interest.

The first training loss of the untrained model is, 8.5388

2.3 Inference Decoder

During inference, the model cannot be provided with target tokens

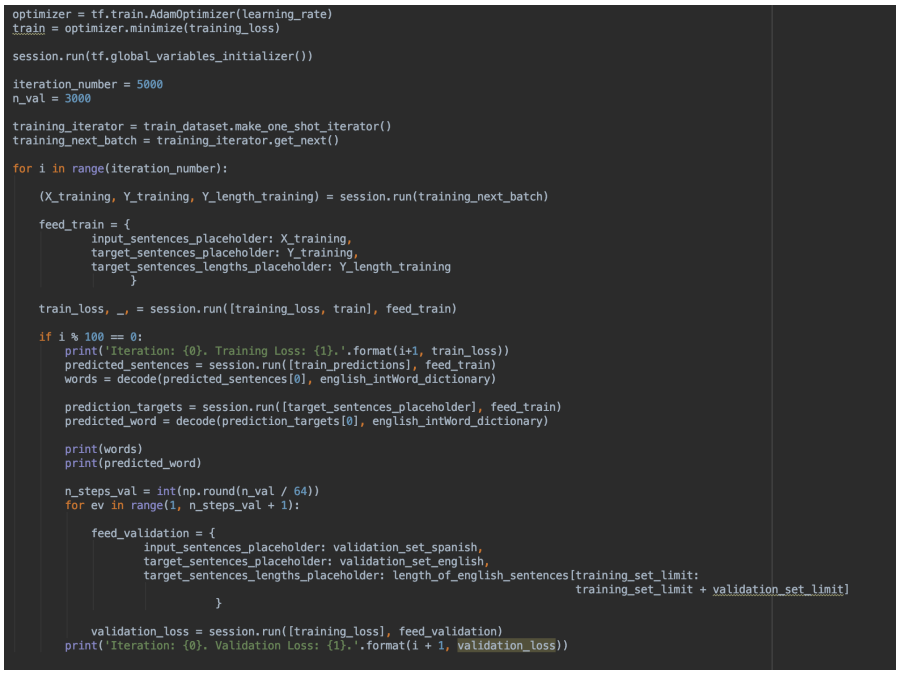
Below, I added the training decoder part of my code.

```
inference_decoder_input = word_embeddings_english

inf_decoder_init_state = attention_wrapper.zero_state(batch_size=dynamic_batch_size, dtype=tf.float32).clone(
    cell_state=encoder_final_state)

inference_decoder_helper = tf.contrib.seq2seq.GreedyEmbeddingHelper(inference_decoder_input,
                                                                    tf.fill([dynamic_batch_size],
                                                                    english_wordInt_dictionary["<START>"]),
                                                                    english_wordInt_dictionary["<END>"])

inference_decoder = tf.contrib.seq2seq.BasicDecoder(cell=attention_wrapper,
                                                    helper=inference_decoder_helper,
                                                    initial_state=inf_decoder_init_state,
                                                    output_layer=projection_layer)

inference_decoder_max_iterations = tf.round(tf.reduce_max(max_length_english) * 2)

inference_decoder_outputs, inference_decoder_final_state, inference_decoder_final_sequence_length = tf.contrib.seq2seq.dynamic_decode(inference_decoder,
                                                                                                                                       maximum_iterations=inference_decoder_max_iteratio
                                                                                                                                       )


inference_logits = inference_decoder_outputs.rnn_output
inf_predictions = inference_decoder_outputs.sample_id

stack_inference_logit_tensors = tf.stack([tf.shape(target_sentences_placeholder)[1], tf.shape(inference_logits)[1]])
maximum_inference_logits_tensor = stack_inference_logit_tensors[tf.argmax(stack_inference_logit_tensors)]
inference_logits_difference = maximum_inference_logits_tensor - tf.shape(inference_logits)[1]

padded_inference_logits = tf.pad(tensor=inference_logits,
                                 paddings=[[0, 0], [0, inference_logits_difference], [0, 0]])

inference_mask = tf.sequence_mask(lengths=target_sentences_lengths_placeholder - 1,
                                  dtype=tf.float32)

mask_inference_difference = maximum_inference_logits_tensor - tf.shape(inference_mask)[1]

inference_padded_mask = tf.pad(tensor=inference_mask,
                               paddings=[[0, 0], [0, mask_inference_difference]])

placeholder_difference = maximum_inference_logits_tensor - tf.shape(target_sentences_placeholder)[1]

placeholder_padded = tf.pad(tensor=target_sentences_placeholder,
                            paddings=[[0, 0], [0, placeholder_difference]])

inference_loss = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=placeholder_padded,
                                                                logits=padded_inference_logits)

inference_loss = tf.reduce_sum((inference_loss*inference_padded_mask))/tf.reduce_sum(inference_padded_mask)
```

Figure 7: Inference Decoder Part of The Code

2.4 Training and Visualizing the Alignment

Below, I added the training decoder part of my code.



```
optimizer = tf.train.AdamOptimizer(learning_rate)
train = optimizer.minimize(training_loss)

session.run(tf.global_variables_initializer())

iteration_number = 5000
n_val = 3000

training_iterator = train_dataset.make_one_shot_iterator()
training_next_batch = training_iterator.get_next()

for i in range(iteration_number):

    (X_training, Y_training, Y_length_training) = session.run(training_next_batch)

    feed_train = {
            input_sentences_placeholder: X_training,
            target_sentences_placeholder: Y_training,
            target_sentences_lengths_placeholder: Y_length_training
                }

    train_loss, _, = session.run([training_loss, train], feed_train)

    if i % 100 == 0:
        print('Iteration: {0}. Training Loss: {1}.'.format(i+1, train_loss))
        predicted_sentences = session.run([train_predictions], feed_train)
        words = decode(predicted_sentences[0], english_intWord_dictionary)

        prediction_targets = session.run([target_sentences_placeholder], feed_train)
        predicted_word = decode(prediction_targets[0], english_intWord_dictionary)

        print(words)
        print(predicted_word)

        n_steps_val = int(np.round(n_val / 64))
        for ev in range(1, n_steps_val + 1):

            feed_validation = {
                    input_sentences_placeholder: validation_set_spanish,
                    target_sentences_placeholder: validation_set_english,
                    target_sentences_lengths_placeholder: length_of_english_sentences[training_set_limit:
                                                                                      training_set_limit + validation_set_limit]
                            }

            validation_loss = session.run([training_loss], feed_validation)
        print('Iteration: {0}. Validation Loss: {1}.'.format(i + 1, validation_loss))
```

Figure 8: Training Part of The Code

In here, I am using the placeholders I created before to train my model. I am using the input sentence set, target sentence set and the length of the target sentence. Then, I feed the session.run with these informations and I receive the loss of the train.

My first totally correct translation comes at 700. iteration. Below, I added the screenshot.

```
Iteration: 701. Training Loss: 2.073451280593872.
[['<START>', 'is', 'this', 'your', 'drink', '?',
[['<START>', 'is', 'this', 'your', 'beer', '?', '
```

Figure 9: Output Sample

The first sentence is my model's prediction and the other one is the target sentence.

I could not finish the whole assignment. That's why I can not write answers of the questions 7 and 8 which belongs in this section. However, I finished until 5th bullet of this subsection.

3 Parameters

Below, I added the parameters I used for this assignment.

- word limit = 30.000
- training set limit = 18.000
- validation set limit = 3.000
- test set limit = 9.000
- batch size = 64
- buffer size = 10
- embedding size = 256
- hidden size = 1.024
- learning rate = 0.001
- iteration number = 5.000