Assignment 4: Deep Learning Lab

Wei Huang

In Spanish2English folder there are four python code files: data_helpers.py, model.py, train.py, predict.py, and each of them accomplish different tasks. Besides, there is one folder inside named model which contain my pretrained model parameters.

1 Preprocessing data

In data_helpers.py we define class of Data which could download the first 30000 pairs of Spanish to English sentences, tokenize each sentence, and final convert word sentences to corresponding integer list.

```
class Data():
  def __init__(self):
    self.load_data()
    self.transform()
    self.token_container
    self.num_container
    self.eng2int
    self.spa2int
    self.int2eng
    self.int2spa
```

After implementing:

```
  data = Data()
```

The summary information of our raw data is printed below:

```
  The total number of English word is 5108
  The total number of Spanish word is 9470
  The maximum length of English sentence is 11
  The maximum length of Spanish sentence is 16
```

The data.token_container is a list of lists. Each element contain a pair of Enlish and Spanish tokenized sentences. and data.num_container is also a list of lists which is gotten by transforming words to integers. Moreover, data.eng2int and spa2int are dictionary which maps words to corresponding integers, and data.int2eng, int2spa are list which map integers to corresponding words, and I introduce a padding token using the integer 0. For example I print out the first element of token_container and num_container below:

```
The first element of token_container is
[['<SOS>', 'go', '.', '<EOS>'], ['<SOS>', 've', '.', '<EOS>']]
The first element of num_container is
[[1, 2, 3, 4], [1, 2, 3, 4]]
```

Then, I define a class of Batch:

```
class Batch:
  def __init__(self):
    self.encoder_inputs = []
    self.encoder_inputs_length = []
    self.decoder_targets = []
    self.decoder_targets_length = []
```

One instance batch of class Batch contain pairs of English and Spanish and their length of sentence which will help us calculate correct error. the batch.encoder_inputs contain Spanish sentences coded by integer lists with uniform length, and batch.decoder_targets contain English sentences coded by integer lists with uniform length. By calling getBatches(data, batch_size) defined in data_helpers.py we could get a list of instances od class Batch. Each element contain one instance of Batch class.
Finally, I split data into three parts: trainset, validationset, and testset:

dataset	number
trainset	23000
validationset	1000
testset	6000

2 Model

In model.py we define a class named Seq2SeqModel:

```
class Seq2SeqModel():
  def __init__(self, rnn_size, num_layers, embedding_size, eng2int,
                                    spa2int, learning_rate,
                                    use_attention, max_gradient_norm=5.0
                                    ):
    self.learing_rate = 0.001
    self.embedding_size = 256
    self.rnn_size = 1024
    self.num_layers = 1
    self.eng_vocab_size = len(eng2int)
    self.spa_vocab_size = len(spa2int)
    self.use_attention = True
    self.max_gradient_norm = max_gradient_norm
    self.eng2int = eng2int
    self.spa2int = spa2int
    self.build_model()
```

And then I implement the encoder part:

```
#====1, Defining the placeholder
self.encoder_inputs = tf.placeholder(tf.int32, [None, None], name='
                                     encoder_inputs')
self.encoder_inputs_length = tf.placeholder(tf.int32, [None], name='
                                     encoder_inputs_length')

self.batch_size = tf.placeholder(tf.int32, [], name='batch_size')
self.keep_prob_placeholder = tf.placeholder(tf.float32, name='
                                     keep_prob_placeholder')

self.decoder_targets = tf.placeholder(tf.int32, [None, None], name='
                                     decoder_targets')
self.decoder_targets_length = tf.placeholder(tf.int32, [None], name='
                                     decoder_targets_length')
self.max_target_sequence_length = tf.reduce_max(self.
                                     decoder_targets_length, name='
                                     max_target_len')
self.mask = tf.sequence_mask(self.decoder_targets_length, self.
                                     max_target_sequence_length, dtype=tf
                                     .float32, name='masks')

#====2, Defining Encode
with tf.variable_scope('encoder'):
  #using LSTM to encode input sentence.
  encoder_cell = self._create_rnn_cell()
  #define word embedding, we could also use existing word embedding
  spa_embedding = tf.get_variable('spa_embedding', [self.spa_vocab_size
                                     , self.embedding_size])
  encoder_inputs_embedded = tf.nn.embedding_lookup(spa_embedding, self.
                                     encoder_inputs)
  # encoder_outputs = [batch_size, encoder_inputs_length, rnn_size]
  # final_encoder_state = [batch_size, rnn_szie]
  encoder_outputs, final_encoder_state = tf.nn.dynamic_rnn(encoder_cell
                                     , encoder_inputs_embedded,
  sequence_length=self.encoder_inputs_length,
  dtype=tf.float32)
```

Thirdly, I implement attention:

```
#Standard way to implementing attention model
attention_mechanism = tf.contrib.seq2seq.BahdanauAttention(num_units=
                                     self.rnn_size, memory=
                                     encoder_outputs,
memory_sequence_length=encoder_inputs_length)

decoder_cell = self._create_rnn_cell()
decoder_cell = tf.contrib.seq2seq.AttentionWrapper(cell=decoder_cell,
                                     attention_mechanism=
                                     attention_mechanism,
attention_layer_size=self.rnn_size, alignment_history = True, name='
                                     Attention_Wrapper')
#If we use beam search, we have to change the batch size
```

```
#use final_encoder_state as decoder_initial_state
decoder_initial_state = decoder_cell.zero_state(batch_size=self.
                                    batch_size, dtype=tf.float32).clone(
                                    cell_state=final_encoder_state)
output_layer = tf.layers.Dense(self.eng_vocab_size, kernel_initializer=
                                    tf.truncated_normal_initializer(mean
                                    =0.0, stddev=0.1))
```

Fourthly I implement Training Decoder and Inference Decoder:

```
#-----Training Decoder
training_helper = tf.contrib.seq2seq.TrainingHelper(inputs=
                                    decoder_inputs_embedded,
sequence_length=self.decoder_targets_length,
time_major=False, name='training_helper')
training_decoder = tf.contrib.seq2seq.BasicDecoder(cell=decoder_cell,
                                    helper=training_helper,
initial_state=decoder_initial_state, output_layer=output_layer)
train_decoder_outputs, _, _ = tf.contrib.seq2seq.dynamic_decode(
                                    decoder=training_decoder,
impute_finished=True,
maximum_iterations=self.max_target_sequence_length)
#-----Inference Decoder
start_tokens = tf.ones([self.batch_size,], tf.int32) * self.eng2int['
                                    <SOS>']
end_token = self.eng2int['<EOS>']
decoding_helper = tf.contrib.seq2seq.GreedyEmbeddingHelper(embedding=
                                    eng_embedding,
start_tokens=start_tokens, end_token=end_token)
inference_decoder = tf.contrib.seq2seq.BasicDecoder(cell=decoder_cell,
                                     helper=decoding_helper,
initial_state=decoder_initial_state,
output_layer=output_layer)
inference_decoder_outputs, inference_decoder_state, _= tf.contrib.
                                    seq2seq.dynamic_decode(decoder=
                                    inference_decoder,
impute_finished=True,
maximum_iterations=20)
```

Finally, I compute the error by using tf.contrib.seq2seq.sequence_loss and tf.sequence_mask:

```
self.mask = tf.sequence_mask(self.decoder_targets_length, self.
                                    max_target_sequence_length, dtype=tf
                                    .float32, name='masks')
self.decoder_logits_train = tf.identity(train_decoder_outputs.
                                    rnn_output)
self.decoder_predict_train = tf.argmax(self.decoder_logits_train, axis=
                                    -1, name='decoder_pred_train')
self.train_loss = tf.contrib.seq2seq.sequence_loss(logits=self.
                                    decoder_logits_train,
targets=self.decoder_targets, weights=self.mask)
```

3 Training and Visualizing the Alignment

Defining our train_op and train, evaluation, translation function:

```
#-----Define train_op
optimizer = tf.train.AdamOptimizer(self.learing_rate)
trainable_params = tf.trainable_variables()
gradients = tf.gradients(self.train_loss, trainable_params)
clip_gradients, _ = tf.clip_by_global_norm(gradients, self.
                                    max_gradient_norm)
self.train_op = optimizer.apply_gradients(zip(clip_gradients,
                                    trainable_params))
#-----Define train, evaluation, translation function
def train(self, sess, batch):
  feed_dict = {self.encoder_inputs: batch.encoder_inputs,
    self.encoder_inputs_length: batch.encoder_inputs_length,
    self.decoder_targets: batch.decoder_targets,
    self.decoder_targets_length: batch.decoder_targets_length,
    self.keep_prob_placeholder: 1.0,
    self.batch_size: len(batch.encoder_inputs)}
  _, loss = sess.run([self.train_op, self.train_loss], feed_dict=
                                    feed_dict)
  return loss



def evaluation(self, sess, batch):
  feed_dict = {self.encoder_inputs: batch.encoder_inputs,
    self.encoder_inputs_length: batch.encoder_inputs_length,
    self.decoder_targets: batch.decoder_targets,
    self.decoder_targets_length: batch.decoder_targets_length,
    self.keep_prob_placeholder: 1.0,
    self.batch_size: len(batch.encoder_inputs)}
  loss, summary = sess.run([self.inference_loss, self.summary_op],
                                    feed_dict=feed_dict)
  return loss, summary

def translation(self, sess, batch):
  feed_dict = {self.encoder_inputs: batch.encoder_inputs,
    self.encoder_inputs_length: batch.encoder_inputs_length,
    self.decoder_targets: batch.decoder_targets,
    self.decoder_targets_length: batch.decoder_targets_length,
    self.keep_prob_placeholder: 1.0,
    self.batch_size: len(batch.encoder_inputs)}
  predict, alignments = sess.run([self.decoder_predict_inference, self.
                                    decoder_alignments], feed_dict=
                                    feed_dict)
  return predict, alignments
```

d

Training my encoder-decoder implementation on the first 30'000 samples using single-layer LSTMs with 1024 units, a batch size of 64, and an embedding size of 256. After roughly 5000 steps the training error is 0.14696860313415527.
I print translation result of two Spanish sentences every 500 steps:

```
1st: 'esta es mi vida.'
2nd: 'todavia estan en casa?'
```

The result is below:

```
after 0 steps:
<SOS> <SOS> <SOS> <SOS> . <EOS>
<SOS> <SOS> <SOS> <SOS> . <EOS>
after 500 steps:
<SOS> this is my life . <EOS>
<SOS> are you in home ? <EOS>
after 100 steps:
<SOS> this is my life . <EOS>
<SOS> are you still home ? <EOS>
after 1500 steps:
<SOS> this is my life . <EOS>
<SOS> are you still at home ? <EOS>
after 2000 steps:
<SOS> this is my life . <EOS>
<SOS> are you still at home ? <EOS>
after 2500 steps:
<SOS> this is my life . <EOS>
<SOS> are you still home ? <EOS>
after 3000 steps:
<SOS> this is my life . <EOS>
<SOS> are you still at home ? <EOS>
after 3500 steps:
<SOS> this is my life . <EOS>
<SOS> are you still home ? <EOS>
after 4000 steps:
<SOS> this is my life . <EOS>
<SOS> are you still at home ? <EOS>
after 4500 steps:
<SOS> this is my life . <EOS>
<SOS> are you still at home ? <EOS>
after 5000 steps:
<SOS> this is my life . <EOS>
<SOS> are you still at home ? <EOS>
```

And the cross entropy with respect to step is the following graph: Finally, we plot the alignment matrix of the two input Spanish sentences: For the alignment graph for the first sentence, we could see when we getting the translated English word life, the weight of vida is very big because the meaning of vida is life in English. In term of the second sentence, estan influence the word home more than others, and todavia influence the words are and you the most. This is because in attention model there is different weight of input sequence at different time for output sentence. The first sentence is easy to translate because the order of Spanish words is the same as the order of corresponding

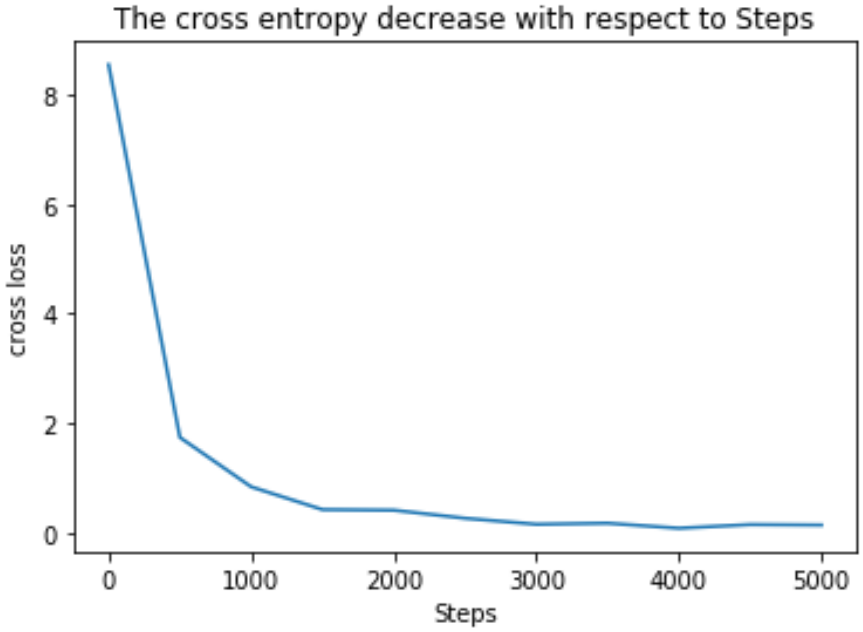

Figure 1: Loss decreasing Graph

English words. But for the second sentence, the order of Spanish words and English words is different. In order to get the correct English sentence, we have to rearrange the English words.

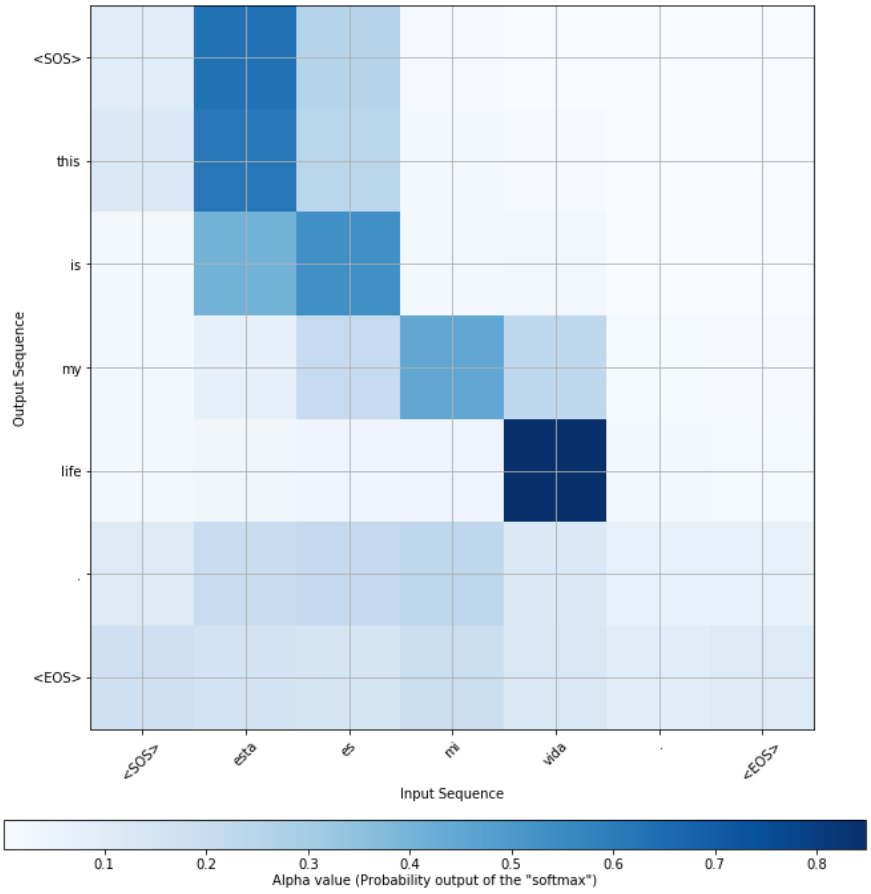

Figure 2: The first sentence

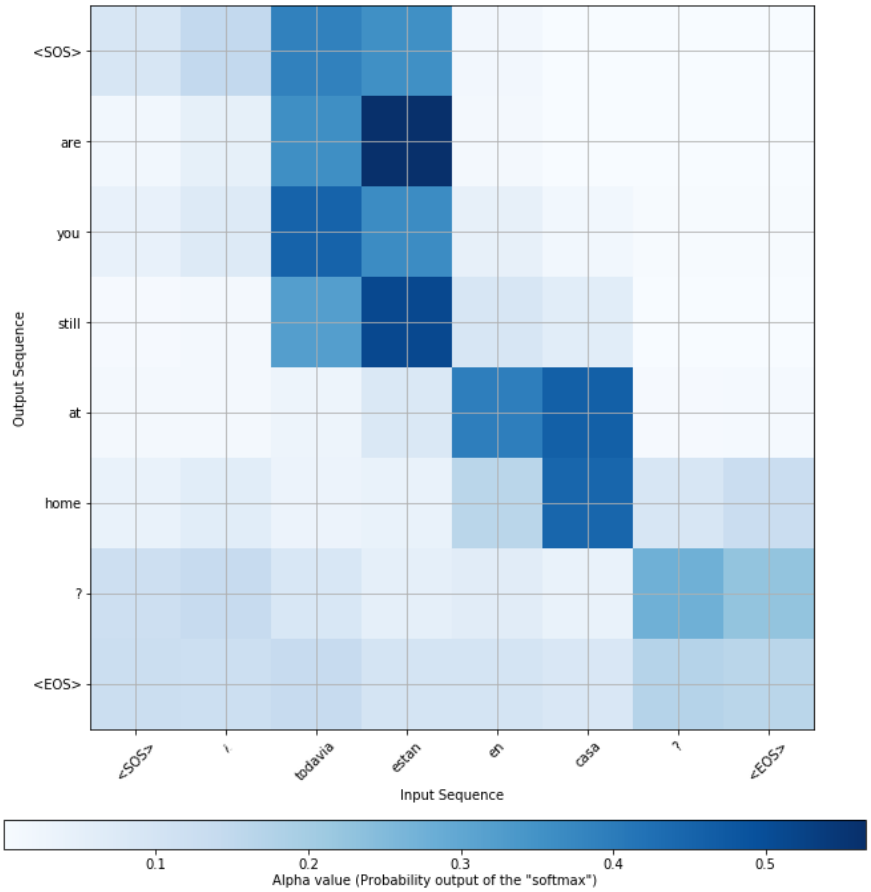

Figure 3: The second sentence