8810 Homework 2

Song Liao
Department of Computing
Clemson University
Clemson, USA
liao5@g.clemson.edu

I. Task Description

In this homework, we need to realize a video caption generation based on sequence to sequence model. The input is a video feature and output is corresponding caption that depicts the video. The sequence to sequence model is consists of two recurrent neural networks, one encoder to process the input and one decoder to generate the output. For result, the BLEU is used to generate the baseline.

The dataset is MSVD, which contains 1450 videos for training and 100 videos for testing.

II. Sequence to Sequence Model

The Sequence to Sequence Model mainly contains three parts: encoding layer, decoding layer and seq2seq model. The whole model is shown in Fig.1.

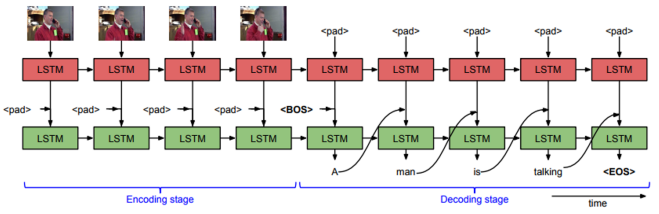

Fig. 1. Sequence to Sequence Model

A. Encoding Layer

First part is encoding layer, which takes features of video and targe sentences as input while outputs RNN output and state. The input placeholder is defined in Fig.2. Because the video feature is 80*4096, so the inputs shape of encoder is [None,80,4096], and the 'None' means the batch size.

```
def enc_dec_model_inputs():
    inputs = tf.placeholder(tf.float32, [None,80,4096], name='input')
    targets = tf.placeholder(tf.int32, [None, None], name='targets')
```

Fig. 2. Encoder input

Then the inputs are passed to encoding layer. Encoding layer uses 'tf.contrib.rnn.MultiRNNCell' to create several layers of cells and then build 'dynamic rnn' with cells and inputs. These code are shown in Fig.3.

```
def encoding_layer(rnn_inputs, rnn_size, num_layers, keep_prob,
                   source_vocab_size):

    stacked_cells = tf.contrib.rnn.MultiRNNCell([tf.contrib.rnn.DropoutWrapper(tf.contrib.rnn.LSTMCell(rnn_size),
                                                                               keep_prob) for _ in range(num_layers)])

    outputs, state = tf.nn.dynamic_rnn(stacked_cells,
                                       rnn_inputs,
                                       dtype=tf.float32)
    return outputs, state
```

Fig. 3. Encoding layer

B. Decoding Layer

Next is the decoding layer. Different with encoding layer, decoding layer can have two different inputs while training phase and testing phase. When training, the target sentences are provided, so the inputs are known; when testing, the input is the output of last step. So the decoding layer contains two parts, decoding_layer_train and decoding_layer_infer. Before that, we need to add the '¡BOS¿' tag, which means the begin of sentence to tell model that the decoding starts. The code of decoding_layer_train and decoding_layer_infer are shown in Fig.4 and Fig.5.

```
def decoding_layer_train(encoder_state, dec_cell, dec_embed_input, dec_embeddings,
                         target_sequence_length, max_summary_length,
                         output_layer, keep_prob):

    dec_cell = tf.contrib.rnn.DropoutWrapper(dec_cell,
                                             output_keep_prob=keep_prob)

    # for only input layer
#     helper = tf.contrib.seq2seq.TrainingHelper(dec_embed_input,
#                                                target_sequence_length)
#
#     helper = tf.contrib.seq2seq.GreedyEmbeddingHelper(dec_embeddings,
#                                                       tf.fill([batch_size], start_of_sequence_id),
#                                                       end_of_sequence_id)

    helper=tf.contrib.seq2seq.ScheduledEmbeddingTrainingHelper(dec_embed_input,
                                                               target_sequence_length,
                                                               dec_embeddings,
                                                               0.5)

    decoder = tf.contrib.seq2seq.BasicDecoder(dec_cell,
                                              helper,
                                              encoder_state,
                                              output_layer)

    # unrolling the decoder layer
    outputs, _, _ = tf.contrib.seq2seq.dynamic_decode(decoder,
                                                      impute_finished=True,
                                                      maximum_iterations=max_summary_length)
    return outputs
```

Fig. 4. Decoding layer train

```
def decoding_layer_infer(encoder_state, dec_cell, dec_embeddings, start_of_sequence_id,
                         end_of_sequence_id, max_target_sequence_length,
                         vocab_size, output_layer, batch_size, keep_prob):

    dec_cell = tf.contrib.rnn.DropoutWrapper(dec_cell,
                                             output_keep_prob=keep_prob)

    helper = tf.contrib.seq2seq.GreedyEmbeddingHelper(dec_embeddings,
                                                      tf.fill([batch_size], start_of_sequence_id),
                                                      end_of_sequence_id)

    decoder = tf.contrib.seq2seq.BasicDecoder(dec_cell,
                                              helper,
                                              encoder_state,
                                              output_layer)

    outputs, _, _ = tf.contrib.seq2seq.dynamic_decode(decoder,
                                                      impute_finished=True,
                                                      maximum_iterations=max_target_sequence_length)
    return outputs
```

Fig. 5. Decoding layer infer

The inputs of decoding_layer_train need to be set as same length. For doing that, while getting each batch of data, we get the length of longest sentence and add other sentence with some ”¡PAD¿” tag so that for each batch, the length of input data are same.

C. seq2seq model

At last, we combine the encoding layer and decoding layer and form the sequence to sequence model. The code is shown in Fig.6.

```
def seq2seq_model(input_data, target_data, keep_prob, batch_size,
                  target_sequence_length,
                  max_target_sentence_length,
                  source_vocab_size, target_vocab_size,
                  dec_embedding_size,
                  rnn_size, num_layers, target_vocab_to_int):

    enc_outputs, enc_states = encoding_layer(input_data,
                                             rnn_size,
                                             num_layers,
                                             keep_prob,
                                             source_vocab_size)

    dec_input = process_decoder_input(target_data,
                                      target_vocab_to_int,
                                      batch_size)

    train_output, infer_output = decoding_layer(dec_input,
                                                enc_states,
                                                target_sequence_length,
                                                max_target_sentence_length,
                                                rnn_size,
                                               num_layers,
                                               target_vocab_to_int,
                                               target_vocab_size,
                                               batch_size,
                                               keep_prob,
                                               dec_embedding_size)

    return train_output, infer_output
```

Fig. 6. Seq2seq model

D. Parameters Setting

Several parameters in graph need to be set before running the model. We use the ’tf.contrib.seq2seq.sequence_loss’ to calculate the loss, and ’tf.train.AdamOptimier’ is used to calculate the gradient descent of the loss. Code of graph is in Fig.7.

```
cost = tf.contrib.seq2seq.sequence_loss(
    training_logits,
    targets,
    masks)

optimizer = tf.train.AdamOptimizer(lr)

gradients = optimizer.compute_gradients(cost)
capped_gradients = [(tf.clip_by_value(grad, -1., 1.), var) for grad, var in gradients if grad is not None]
train_op = optimizer.apply_gradients(capped_gradients)
```

Fig. 7. Graph

For the training parameters, we set batch size as 64, and rnn has two layers with each 128 size. The decoding embedding size is set 500, which is same with paper. Learning rate is 0.001 and probability of ’DropoutWrapper’ is 0.7.

III. Improving Model Performance

After building the model, I tried several methods to improve the model performance. Two methods are mainly used, schedule sampling and beam search. Before that, first I preprocess the target sentence.

A. Data Preprocess

While the features of videos have been processed and can be used directly, the target sentences are raw sentences and we need to process them. We map each sentence into a list of numbers. For doing that, first we get all the words directory and map each word to a number. Here are two ways to realize it, one is using other English word directory and another is only use words in our dataset. I tried to use a directory with 32000 English words, but it is too slow. At last, I only use the words in our sentence. The code is in Fig.8.

```
for i in target:
    sentences_to_int=[]
    for sentence in i:
        words=res = re.findall(r'\w+', sentence.lower())
        sentence_to_int=[]
        for word in words:
            if (word in vocab_to_int)==False:
                vocab_to_int[word]=num
                num+=1
            sentence_to_int.append(vocab_to_int[word])
        sentences_to_int.append(sentence_to_int)
    target_int.append(sentences_to_int)
```

Fig. 8. Vocab to int

B. Beam Search

I also use the Beam Search to improve performance. For each video, there are tens of sentences to describe it. I sorted these sentences with most frequently first word, second word and third word. Then I use the first ten sentences of each video and this ensures that each video is mapped with ten most common sentences. After that, I trained each video with the ten sentences separately, which means my training dataset is 1450*10, so the epochs should be 20 times (suggested 200). The processed sentences sequences are shown in Fig.9.

```
[['a man slices through a two liter plastic bottle of soda pop with a sword.',
  'a man slices a soda bottle with a sword.',
  'a man slices a soda bottle with a sword.',
  'a man slices the top of a plastic bottle of water off with a sword.',
  'a man sliced a plastic bottle with a sword.',
  'a man is cutting a bottle with a sword.',
  'a man is cutting a bottle of water with a sword.',
  'a man is broken a bottle by a sword.',
  'a man uses a sword to cut a two liter plastic bottle in half.',
  'a man breaks a bottle with a sword.'],
 ['a man is stabbing a cardboard cutout with a sword.',
  'a man is hitting a photo with his sword.',
  'a man is demonstrating a sword.',
  'a man is stabbing a target.',
  'a man is stabbing a silhouette with a sword.',
  'a man with a sword stabs a target.',
  'a man with a sword runs ands stabs a cardboard target.',
  'a man with a sword stabs a target.',
  'a man stuck a sword through a piece of paper.',
  'a person stabs a photo print with a knife.']]
```

Fig. 9. Sorted Sentences

C. Schedule Sampling

The second method I used is schedule sampling, which means while training, we feed groundtruth or last time step’s output as input at odds. I use the ’tf.contrib.seq2seq.ScheduledEmbeddingTrainingHelper’ to realize it. The code is shown in Fig.10. After testing, I found that the best result appears when sampling probability is 0.5.

```
helper=tf.contrib.seq2seq.ScheduledEmbeddingTrainingHelper(dec_embed_input,
                                                           target_sequence_length,
                                                           dec_embeddings,
                                                           0.5)

decoder = tf.contrib.seq2seq.BasicDecoder(dec_cell,
                                          helper,
                                          encoder_state,
                                          output_layer)
```

Fig. 10. Schedule Sampling

IV. RESULT

I trained the model with 20 epochs and save the model. For each epoch, I calculated the bleu result to better analyze the model. At last, the bleu score is 0.72, as shown in Fig.11.

```
Epoch  17 Batch   74/226 - Train Accuracy: 0.8339, Validation Accuracy: 0.6230, Loss: 1.0065
Epoch  17 Batch  148/226 - Train Accuracy: 0.7180, Validation Accuracy: 0.6152, Loss: 1.6814
Epoch  17 Batch  222/226 - Train Accuracy: 0.7412, Validation Accuracy: 0.6211, Loss: 1.5847
0.6877816293365947
Epoch  18 Batch   74/226 - Train Accuracy: 0.7917, Validation Accuracy: 0.6123, Loss: 1.2288
Epoch  18 Batch  148/226 - Train Accuracy: 0.7592, Validation Accuracy: 0.6201, Loss: 1.4350
Epoch  18 Batch  222/226 - Train Accuracy: 0.7329, Validation Accuracy: 0.6221, Loss: 1.5342
0.6799422247527938
Epoch  19 Batch   74/226 - Train Accuracy: 0.7463, Validation Accuracy: 0.6240, Loss: 1.4559
Epoch  19 Batch  148/226 - Train Accuracy: 0.7225, Validation Accuracy: 0.6328, Loss: 1.7063
Epoch  19 Batch  222/226 - Train Accuracy: 0.7391, Validation Accuracy: 0.6230, Loss: 1.5185
0.6811880109335801
Epoch  20 Batch   74/226 - Train Accuracy: 0.7262, Validation Accuracy: 0.6191, Loss: 1.5894
Epoch  20 Batch  148/226 - Train Accuracy: 0.7049, Validation Accuracy: 0.6162, Loss: 1.7839
Epoch  20 Batch  222/226 - Train Accuracy: 0.7297, Validation Accuracy: 0.6162, Loss: 1.6017
0.7219022234452269
```

Fig. 11. Result

All the result includes the model and result are in 'https://github.com/songer12345/cpsc_8810/tree/master/HW2'