Activities 08

Deep Learning Lab

November 23, 2018

1 Assignment 4

In this assignment, you will implement and train a Neural Machine Translation (NMT) model from scratch. NMT is a machine learning approach to machine translation using artificial neural networks such as the Long Short-Term Memory (LSTM).

Your model will be an encoder-decoder model (Sutskever et al., 2014) with an attention mechanism for phrase-based translation (Bahdanau et al., 2014). Make sure you have read and understood both articles before you start working through the assignment.

The assignment is split into 4 parts with a maximum of 100 points. Successfully solving the first three parts will be awarded 80 points. To achieve this you will have to implement your model using the standard TensorFlow graph code (as taught during this course). Submissions with code using TensorFlow eager or another ML framework will not be accepted. Make sure your code is well commented and easy to read. Your code should be accompanied by a report which documents your decisions and relevant experiments.

1.1 Preprocessing

You will translate Spanish sentences into English sentences. In order to make things work, you'll have to perform several preprocessing steps before feeding the data into your model.

1.1.1 Prepare the Sentences

1. Use the keras command in Listing 1 to easily download the dataset of English and Spanish sentences.

   Listing 1: Downloads the translation dataset.

   ```
   path_to_zip = tf.keras.utils.get_file('spa-eng.zip',
   origin='http://download.tensorflow.org/data/spa-eng.zip', extract=True)
   path_to_file = os.path.dirname(path_to_zip)+"/spa-eng/spa.txt"
   ```

2. The file will contain an English and a Spanish sentence on every line separated by a tab (\t). Make sure to open the file using *encoding='UTF-8'*.

3. For the development of the model you should limit your data to only the first 30'000 sentences (118'964 in total).

4. For simplicity, we'll transform every sentence into lower case format.

5. Use the provided unicode_to_ascii function in Listing 2 in order to convert your Unicode sentences into the ASCII format.

   Listing 2: Converts unicode into ascii.

   ```
   def unicode_to_ascii(s):
     return ''.join(c for c in unicodedata.normalize('NFD', s)
         if unicodedata.category(c) != 'Mn')
   ```

6. Your model will be a word-level NMT model. This means that the input tokens at every time step will refer to a word or concatenations of such (e.g. "it's" would be a different token from "it" and "is"). We will also treat the main punctuation as an individual token. For this reason, you have to add a whitespace before and after "?", ".", "!", ",", "¡", and "¿". Make sure to also remove unnecessary white space before proceeding.

7. Add to every sentence a respective start-of-sentence token and end-of-sentence token. This will be necessary for the model to indicate when a sentence is over.

8. Given the previous steps, you can simply split each sentence string along the whitespaces which will give us for every sentence a list of tokens. Given a sentence like "¡Quédate tranquilo!" the preprocessing will result in a list like ['<SOS>', '¡', 'quedate', 'tranquilo', '!', '<EOS>']. The same steps are applied to English sentences.

9. Provide in your report a few preprocessing examples that display raw sentences and their respective preprocessed result up to this point. Make sure they highlight the transformations that we mentioned. Describe the positive and negative impact of these preprocessing steps in the context of training an NMT model.

10. Finally, assign a unique integer number to every unique word and create a word2int and an int2word dictionary for both languages. Make sure to introduce a padding token such as '<pad >' using the integer 0. On top of that, write a function for both languages that translates from a string to a list of numbers and a function that can translate back from a list of numbers to a string. Use these functions to vectorize all your English and Spanish sentences. Using the first 30'000 samples, the vocabulary for Spanish and English should be 9464 and 5107 words respectively. Similarly, the longest sentence should be 16 and 11 tokens respectively.

1.1.2 Load the Data into TensorFlow using the tf.data API

Now that you have preprocessed all of the data we'll prepare an input pipeline using the TensorFlow tf.data API. Have a look at the official guide if there are some details that are not clear (www.tensorflow.org/guide/datasets).

1. Before you pad the sequences to a uniform length you should keep track of the target sentence length. This will later allow us to unroll the RNN with dynamic sequence lengths which is computationally more efficient.

2. Use the *tf.keras.preprocessing.sequence.pad_sequences* function in order to pad all your sentences to a uniform length such that the data can be represented using a numpy array.

3. Make sure to shuffle your data (without mixing inputs with wrong labels).

4. Split the data into a train, validation, and test set. Use a reasonable split and document your argumentation.

5. Use the function *tf.data.Dataset.from_tensor_slices* to create a new dataset. In the end, you should have a tensor which will result in a random batch of vectorised input sentences, and labels. (Hint: make use of the functions shuffle, batch, and repeat)

1.2 Model

The model is an encoder-decoder RNN. The encoder will process the input sentence and produce some fixed-sized vector representations. The decoder then takes these encoder representations and produces the target sentence. During training, you should use a method called *teacher-forcing*. In this setting, the target sentence is known and the decoder receives at every step the previous target as the current input. For the first input, you simply hardcode the chosen start-of-sentence token as the input. Decoding stops once the end of the target sentence is reached.

During inference, the target sentence is not known. In this setting, the decoder receives during the first step the start-of-sentence token and in all following steps the token predicted by the previous step. The decoding stops after a maximum number of steps or when the decoder produced the end-of-sentence token.

1.2.1 Encoder

The encoder is the same during training and inference. Its job is to encode the input sentence.

1. You will need at least three placeholders for the batch of input sentences, target sentences, and target sentence lengths.

2. Use the *tf.nn.embedding_lookup* function in order to lookup a distributed token representation (that is learned) based on your token integer number.

3. Use a single LSTMCell layer with dynamic_rnn to encode the sentence. Make sure you end up with the final encoder state (in this case an LSTM tuple consisting of the c and h of the LSTMCell) and the stack of all outputs of your encoder which you should pass to your decoder.

1.2.2 Training Decoder

The decoder during training requires a target sentence and uses teacher-forcing to train the embeddings and encoder and decoder RNNs.

1. Similarly to the encoder, you should look up the distributed token representation for the target language.
2. The decoder RNN will be another single LSTMCell layer but with the attention mechanism introduced by Bahdanau et. al. (Bahdanau et al., 2014). You can use the se2seq API in TensorFlow for this. The function *tf.contrib.seq2seq.AttentionWrapper* takes an RNN cell and returns a new RNN cell that has been decorated with an attention mechanism such as *tf.contrib.seq2seq.BahdanauAttention*.
3. Make sure you initialize the initial state of your decoder to the final state of the encoder. You can use the clone function of your wrapped cell as in Listing 3 to do so.

Listing 3: Makes the initial state of the attention-decoder the final encoder state.

```
decoder_init_state = wrapped_decoder_cell \
                     .zero_state(batch_size, tf.float32) \
                     .clone(cell_state=final_encoder_state)
```

4. We recommend that you use *tf.contrib.seq2seq.dynamic_decode* in order to unroll the RNN and feed in the previous prediction or target token. During training, this means the input at the current step is either the start-of-sentence token or the previous target token. Dynamic_decode requires three additional objects. The first is the *tf.contrib.seq2seq.TrainingHelper* which is responsible for feeding the previous target token, i.e. the embedded target sentences as well as the sequence length of that sentence.
5. The next object is a linear transformation from the RNNCell output to the number of valid tokens. You can simply use *tf.layers.Dense* for this purpose.
6. The third is the actual *tf.contrib.seq2seq.BasicDecoder*. It takes as parameters the objects you just have defined i.e. the attention augmented RNN cell, the helper object, the initial state of the decoder, and the projection layer.
7. Your decoder can now be unfolded using *tf.contrib.seq2seq.dynamic_decode*. As you can look up in the official documentation, dynamic_decoder returns the decoder outputs in *final_outputs.rnn_output*. These outputs are often also called logits because they represent the unnormalized log-probabilities for each possible class (or English word token, in this case).
8. Before you can use the logits to compute the error with respect to the actual targets you have to make sure that they are the same length. You need to pad the outputs such that their batch_size and sequence_length dimensions fit.
9. You can now use *tf.nn.sparse_softmax_cross_entropy_with_logits* to compute the error using your logits and your integer targets. This error function takes your logits and computes for each element in the batch the probability distribution over all possible English words using the softmax function. It also transforms every target sentence integer in your batch into a one-hot vector which represents your target probability distribution. The error between your predicted word distribution and your target word distribution is then computed using the discrete cross-entropy (en.wikipedia.org/wiki/Cross_entropy).
10. Make sure you compute the average error over the predictions that we actually care about. Since you padded the outputs to a uniform length you have several predictions that are not of interest. You can use a mask such as *tf.sequence_mask* to easily generate a mask. Make sure to provide a section in your report that explains how you compute this average.
11. Assume that your untrained model produces the same probability for every output token. What is the error value that you would expect the network to have before you start training? Make sure to provide an explanation for your estimate.

1.2.3 Inference Decoder

During inference, you are not in possession of the correct target sentence. In this case, the model cannot be provided with target tokens (i.e. teacher-forcing). Instead, the model takes the token that was predicted in the previous step as the current input. This will require a different decoder computation but you'll be using the same parameters (the same RNN cell).

1. Similar to the training decoder you will require a helper object and a decoder object for your inference decoder. You can use *tf.contrib.seq2seq.GreedyEmbeddingHelper* to greedily pick the current input embedding according to the most probable token of the previous step. You can also use *tf.contrib.seq2seq.SampleEmbeddingHelper* which will instead sample from the previous probability distribution. Both have to be provided with the start-of-sentence token of your target language, as well as the end-of-sentence token of your target language after which the decoding process will stop. Make sure to look at the official API documentation.

2. You can use the same *tf.contrib.seq2seq.BasicDecoder* function to create your inference decoder. Make sure to limit *tf.contrib.seq2seq.dynamic_decode* to a maximum number of iterations using the *maximum_iterations* parameter to force the decoder to stop decoding at some point. Don't forget to also provide the same projection layer as in the previous section! Otherwise, your helper function will not be able to select the correct token.

3. Similar to the previous section, you should pad your inference logits before computing your average error with regards to the inference decoding.

4. Create a prediction tensor using *tf.argmax* which will pick the highest value of your logit tensor and return an integer value for every word in your sequence and every sequence in your batch. Once you have computed these predictions using *session.run*, you can use the translation function that you have implemented previously to transform your predictions into a readable string.

1.3 Training and Visualizing the Alignment

1. Train your network using *tf.train.AdamOptimizer* with the default learning rate.

2. Create a train function which will train your model using batches from your training set. Make sure it logs current train and validation errors after a certain number of steps.

3. Create an evaluation function which will use the inference decoding on the full validation or test set and log the total average error.

4. Create a translation function that will take a sentence in Spanish and use your model to output the English translation. You can also implement this in a separate python script which will load your model from a checkpoint if you wish.

5. Augment the translation function from the previous point with a visualization of the alignment from the attention mechanism. For every step of the decoding phase, the RNNCell uses a context vector for the current prediction. This context vector is a weighted sum of all encoder outputs. The attention vector is essentially the weight vector of that weighted sum of encoder outputs. Ideally, when the decoder has to produce the output token "dog" it will put a hight weight on the encoder output at the time step in which it received the input token "perro". Using *tf.contrib.seq2seq.AttentionWrapper* the alignment matrix (i.e. all attention vectors stacked) can be easily extracted by setting the parameter *alignment_history* to true. This will result in an additional output *decoder_states.alignment_history.stack()* which will result in a tensor with the dimensions [input_sequence_size, batch_size, output_sequence_size]. Use matplotlib and the store_attention function below 4 to save this alignment matrix as a png image. You may adapt this function to your own needs.

Listing 4: A possible alignment visualisation script.

```
def store_attention_plot(attn_map, input_tokens, output_tokens, step_id):
    """
    Stores the alignment matrix plot as a png using the
    alignment matrix of a single sample.

    attn_map: numpy matrix of dimensions [input_length, output_length]
              representing the alignment matrix of one input-output sample.
    input_tokens: a list of strings to be plotted as y-axis labels.
    output_tokens: a list of strings to be plotted as x-axis labels.
    step_id: the current training iteration in order to not overwrite
             previous images generated.
    """
    input_len = len(input_tokens)
    output_len = len(output_tokens)
    attn_map = attn_map[:,:output_len]
```

```
attn_map = attn_map.T

fig = plt.figure(figsize=(10,10))
ax = fig.add_subplot(1, 1, 1)
i = ax.imshow(attn_map, interpolation="nearest", cmap="Blues")

# add words
ax.set_yticks(range(input_len))
ax.set_yticklabels(input_tokens)

ax.set_xticks(range(output_len))
ax.set_xticklabels(output_tokens, rotation=45)

ax.set_ylabel("input sequence")
ax.set_xlabel("output sequence")

ax.grid()

plt.savefig("alignment-{}.png".format(step_id), bbox_inches='tight')
```

6. Train your encoder-decoder implementation only 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 your model should achieve a training error lower than 0.2.

7. Provide in your report the English translation of the sentence *¿todavía están en casa?* (translation: *Are they still at home?*) at different points in your training. Explain why this sentence is harder for the model to translate compared to e.g. *esta es mi vida.* (translation: *This is my life.*).

8. Give an interpretation of the alignment matrix of the input sentence *¿todavía están en casa?*. How does the alignment model change if you train the model using all sample points instead of just the first 30'000? Can you explain the difference?

1.4 Extension

1. The previous three sections will be awarded at most 80 points. To achieve the maximum 100 points you have to provide a meaningful extension to this assignment. Make sure to document all relevant decisions, conclusions, and results in your report.

References

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112.