Homework 9 - RNNs

December 2020

1 Homework summary

In this weeks homework we are not looking into any real data - most problems we would be using RNNs for simply are a bit too large in scope to easily train in reasonable time for a homework (and you will have to do this once later anyway when we discuss NLP & DL!). Instead we will create a simple toy task and write our own implementation of an LSTM to solve it.

The learning goal for you here is twofold: First and foremost this allows you to go through implementing an LSTM in detail, which is really important in understanding how it works after all. Secondly it is the perfect playground to work on some advanced implementation skills, which while not necessary for a running LSTM cell are very useful for one that runs fast - and understanding these tools will help you to generally build better running and scaling Tensorflow models. Generally these advanced implementation approaches are not necessary to pass this homework, but they are required to obtain an *outstanding* this week.

2 The task

The task is simple: Generate sequences of digits and let the network find out which digit out of two queried digits is the more common one in the sequence, or whether the two queried digits were shown the same number of times. Specifically, the network is first shown the two queried digits at every timestep, and additionally the respective sequence element from the random sequence:

```
 context:  49
sequence:  82347235461234873524879
```

- where the cyan and the red digit are the queries
- and the sequence is evaluated for which one of the two digits is more common (or whether they are found the same number of times within)

The task can easily be written in a few lines of python. For an outstanding we expect you to implement the task as a generator pattern and create a tensorflow dataset from that -check out this link to see how that's done in a really easy way with the yield statement!

3 The network

While there are great implementations in tensorflow/keras for LSTMs, this week you are asked to build your own! So you are restricted from using any tensorflow/keras inbuilts of LSTM (or RNN!). Specifically we ask you to go at this problem in two steps:

1. Implement a LSTM_Cell whis is called by providing the input for a single timestep and a tuple containing (hidden_state, cell_state)

- According to Josezefowicz et al. Setting the bias of the forget gate to one initially is very important for performance in training LSTMs. Think about it for a moment - what effect would that have on the initial output values of this gate, and what effect will that have on the recurrent cell state? This is actually implemented in the default LSTM cell in Tensorflow - check the $unit_forget_biasargumentthereformoreinformation$!

2. Implement a LSTM, which is created from one (or multiple for a multi-layer LSTM) LSTM Cell. The call function takes the input over multiple timesteps and creates (and returns!) the outputs over multiple timesteps. The input is expected to be of shape [batch_size, timesteps, input_size], the respective output of shape [batch_size, timesteps, output_size] To achieve this you will have to "unroll" the LSTM.

   Unrolling the network is quite easy in Eager Exectution mode. To do this correctly (i.e. efficiently) in graph mode (graph mode is the name for what happens if you decorate a function with the "@tf.function" decorator) you will have to understand a bit more of how graph mode works. Specifically you will probably have to solve the following subproblems:

   - In graph mode we have to be extra careful to appropriately design *conditional - if_else* and *loops*. Specifically for proper handling in graph mode the respective condition (or sequence) arguments have to be tensors. For unrolling the LSTM over multiple timesteps you will probably need this. Check out this part of the Tensorflow guide to get familiar with the details!
   - Typically you would use a list to aggregate results over multiple timesteps. Appending to typical python lists won't work (efficiently or not at all, depending on implementation details) in graph mode. Check out the documentation of the so called TensorArray to find the typical solution

   For just passing this week you won't need to actually implement the points above (even though we strongly recommend you at least read about them), but for an outstanding we expect proper graph mode implementations!

   Also don't forget that you probably want to use both a read-in layer and read-out layer.

4 Training

Think of this task as a proof of concept task - getting some reasonable accuracy is really enough, no need to try and push it this week. So don't feel pressured to use any sort of more advanced optimization. There are a few things we would like you to think about (and for an outstanding quickly comment on those questions):

- Can / should you use truncated BPTT here?
- Should you rather take this as a regression, or a classification problem?