Activity 7

Deep Learning Lab

November 15, 2019

1 Assignment 3

Consider a long short-term memory network trained to predict the next textual character given a sequence of characters. Such network can be used to generate text by sampling a character according to its output and feeding it back as an input. An accessible introduction to this idea is available here.

1. Download a large book from Project Gutenberg, which should be in plain English text (e.g., The Count of Monte Cristo).

2. Preprocess the text: convert characters to lower case, count the number of unique characters, count the frequency of each character, and choose one integer to represent each character.

3. The text is too long to allow backpropagation through time, so it must be broken down into smaller sequences.

   In order to allow backpropagation for a batch of sequences, the text may first be broken down into a number of large blocks, which corresponds to the batch size.

   Each of these blocks may be broken down further into subsequences, such that batch i contains the i-th subsequence of each block.

   During training, batches must be presented in order, and the state corresponding to each block must be preserved across batches. *Important*: In TensorFlow, this requires feeding the current state to *initial_state*, fetching *final_state*, and feeding it to *initial_state* in the next iteration.

   The technique described above is called *truncated backpropagation through time*.

4. Listing 1 partially illustrates the batch creation process. For simplicity, each resulting batch will be used to derive both inputs and targets, leaving no target for the last character of each subsequence.

5. For "The Count of Monte Cristo", you may use 16 blocks with subsequences of size 256. In that case, the input tensor dimension is $(16, 256, k)$, where k is the number of unique characters (one-hot encoding). You may use a MultiRNNCell with two LSTMCells, each containing 256 units, and a softmax output layer with k units. In that case, training would take at least 5 epochs with a learning rate of 10^{-2}.

CHECK GPT-2 ON THE WEB FOR FURTHER DETAILS

6. Train your network and document the evolution of the training loss function.

7. After training, generate and document 20 sequences composed of 256 characters to evaluate your network. Here is a short example text:

   "say it was in the high ignorance to long, a single viastly châteaus his deceives, distinguished to his daughter; "but in the most exacing for faria, to whole of him is satisfière, or this time he carefully that?".

8. **Bonus:** Read more about text generation using long short-term memory networks and try to improve your model.

9. **Bonus:** Implement a long short-term memory network cell by completing the code provided in Listing 2. Recall that a long short-term memory network layer is given by

$$\begin{aligned}
\mathbf{g}[t]_I &= \sigma(\mathbf{W}_I \mathbf{x}[t] + \mathbf{U}_I \mathbf{h}[t-1] + \mathbf{b}_I) \\
\mathbf{g}[t]_O &= \sigma(\mathbf{W}_O \mathbf{x}[t] + \mathbf{U}_O \mathbf{h}[t-1] + \mathbf{b}_O) \\
\mathbf{g}[t]_F &= \sigma(\mathbf{W}_F \mathbf{x}[t] + \mathbf{U}_F \mathbf{h}[t-1] + \mathbf{b}_F) \\
\mathbf{f}[t] &= \phi(\mathbf{W}_f \mathbf{x}[t] + \mathbf{U}_f \mathbf{h}[t-1] + \mathbf{b}_f) \\
\mathbf{s}[t] &= \mathbf{g}[t]_F \odot \mathbf{s}[t-1] + \mathbf{g}[t]_I \odot \mathbf{f}[t] \\
\mathbf{h}[t] &= \mathbf{g}[t]_O \odot \phi(\mathbf{s}[t]),
\end{aligned}$$

   where $\mathbf{x}[t]$ is the input to the recurrent layer at time step t, $\mathbf{h}[t]$ is the output of the recurrent layer at time step t, ϕ is the hyperbolic tangent function, and σ is the sigmoid function.

You should deliver the following by the deadline stipulated on iCorsi3:

- Report: a single *pdf* file that clearly and concisely provides evidence that you have accomplished each of the tasks listed above. The report should not contain source code (not even snippets). Instead, if absolutely necessary, briefly mention which functions were used to accomplish a task.

- Source code: a single Python script that could be easily adapted to accomplish each of the tasks listed above. The source code will be read superficially and checked for plagiarism. Unless this reveals that your code is suspicious, your grade will be based entirely on the report. Therefore, if a task is accomplished but not documented in the report, it will be considered missing. Note: Jupyter notebook files are not acceptable.

Listing 1: Batch creation process (incomplete).

```
def generate_batches(self, batch_size, sequence_length):
    block_length = len(self.text) // batch_size

    batches = []
    for i in range(0, block_length, sequence_length):
        batch = []

        for j in range(batch_size):
            start = j*block_length + i
            end = min(start + sequence_length,
                      j*block_length + block_length)
            # ... (Incomplete)

        batches.append(np.array(batch, dtype=int))

    return batches
```

Listing 2: LSTM Cell (incomplete).

```
from tensorflow.contrib.rnn import LayerRNNCell

class LSTM(LayerRNNCell):
  def __init__(self,
               num_units,
               reuse=None,
               name=None,
               dtype=None):
    super(LayerRNNCell, self)
      .__init__(_reuse=reuse, name=name, dtype=dtype)
    self._num_units = num_units

  @property
  def state_size(self):
    return (self._num_units, self._num_units)

  @property
  def output_size(self):
    return self._num_units

  def build(self, inputs_shape):
    """
    Parameters
    ----------
    inputs_shape : tuple
        Contains the batch size and the number of input units
    """
    if inputs_shape[1].value is None:
      raise ValueError(
        "Expected inputs.shape[-1] to be known, saw shape: %s"
        % inputs_shape)
    input_size = inputs_shape[1].value

    # Incomplete: create parameter matrices and vectors

    self.built = True

  def call(self, inputs, state):
    """
    Parameters
    ----------
    inputs : tensor
        Contains the batch of inputs for a single time step
    state : tuple
        Contains a pair that constitutes a batch of states for a single step
    Returns
    -------
    output : tensor
```

```
        Contains the batch of outputs for a single time step
    new_state : tuple
        Contains a pair that constitutes a batch of states for a single step
    """
    c, h = state

    new_c = # Incomplete: compute the new state of the cell
    new_h = # Incomplete: compute the output of this layer

    new_state = (new_c, new_h)

    return new_h, new_state
```