RNN/LSTM

Learn about recurrent neural networks and LSTM cells.

Chapter Goals:

- Understand the differences between feed-forward and recurrent neural networks
- Learn about the long short-term memory (LSTM) cell

A. Neural network types

As mentioned in the previous chapter, the MLP is a type of *feed-forward neural network*. This means that each of its layers is a fixed size, and each layer's output is fed into the next layer as input. While feed-forward neural networks are great for tasks involving fixed size input data, they aren't as great in dealing with sequences of text data.

For the remainder of this course, we'll be focusing on *recurrent neural networks*, which are specially designed to work with sequential data of varying lengths. The main component of a recurrent neural network (RNN) is its *cell*.

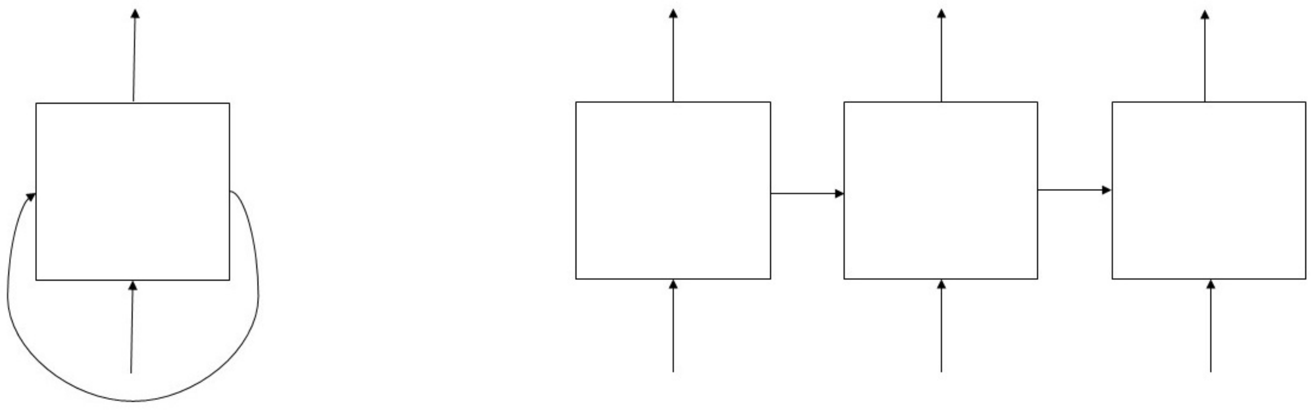

Diagram of an RNN. On the left is the rolled RNN, on the right is the unrolled RNN with 3 time steps. The square represents the RNN cell.

The rolled RNN diagram shows the "true" depiction of the network. It consists of a single cell (although a multi-layer RNN will have multiple stacked cells).

and 3 types of connections: input, output, and recurrent. The unrolled diagram gives us a better look at what each of these connections represent.

In the unrolled diagram above, the RNN consists of 3 time steps, meaning that the length of the RNN's input sequence is 3. This also means that the RNN will output a sequence of length 3. At each time step, the arrow going into the cell represents the token at that particular index of the input sequence. The arrow coming out of the cell represents the cell's output.

> Note: The diagram shows the RNN process for a single input. For batch data, the RNN will handle each individual input independently, in parallel.

The left-to-right arrows connecting the cell at each time step are the recurrent connections. These are the foundation of an RNN, and they represent the transmission of each time step's *state*. The state gives the cell at the current time step information about the cell inputs and outputs from previous time steps. This is incredibly useful for capturing dependencies in the text that make it easier to calculate probabilities and predictions.

B. LSTM

While theoretically a regular RNN should be able to capture all the necessary dependencies in an input text sequence, this is not usually the case in real life. Specifically, it is difficult for RNNs to handle *long-term* dependencies, i.e. dependencies between words that are far apart in a text sequence.

Let's take a look at the following text sequence:

"She is currently a doctor. She chose this career path because her favorite subject in high school was biology."

Here, "biology" and "doctor" have a dependency and are relatively far from each other in the text sequence. If a human were asked to predict the final word in the text sequence from the previous words, they would likely say "biology", given that the subject is a doctor. However, an RNN would have a difficult time picking up this dependency due to its distance.

In order to handle long-term dependencies, we use a variation of the regular

RNN cell known as the LSTM (long short-term memory). While the default RNN cell has issues with long-term dependencies, the LSTM cell is specifically designed to keep track of all the useful dependencies in a text sequence. . When an RNN uses the LSTM cell variation, we normally just refer to the entire model as an LSTM.

C. Inside the cell

The inside of a default RNN cell consists of two fully-connected layers. The first layer has tanh activation, and it's used to compute the cell's state at a particular time step, based on the previous state and the time step's input.

The second fully-connected layer is unactivated, and it's used to compute the cell's output at the time step. You can think of the number of hidden units in the cell as the number of hidden units in the fully-connected layers.

An LSTM cell adds a few additional layers to the default RNN cell. We can refer to these additional layers as "gates", since they help regulate the information that is added or removed from the cell state. These additional gates give LSTM cells the boost needed to handle long-term dependencies.

Time to Code!

In this chapter, you'll be completing the `make_lstm_cell` function, which creates an LSTM cell with `self.num_lstm_units` as the number of hidden units in the cell.

We use `tf.nn.rnn_cell.LSTMCell` to create the LSTM cell. This function is an initializer for an `LSTMCell` object, and the only required argument is the number of hidden units in the cell.

Side note: For creating a basic RNN cell, you can use the `tf.nn.rnn_cell.BasicRNNCell` function.

Set `cell` equal to `tf.nn.rnn_cell.LSTMCell` applied with `self.num_lstm_units` as the only argument. Then return `cell`.

```
import tensorflow as tf

# LSTM Language Model
class LanguageModel(object):
    # Model Initialization
    def __init__(self, vocab_size, max_length, num_lstm_units, num_lstm_layers):
        self.vocab_size = vocab_size
        self.max_length = max_length
        self.max_length = max_length
        self.num_lstm_units = num_lstm_units
        self.num_lstm_layers = num_lstm_layers
        self.tokenizer = tf.keras.preprocessing.text.Tokenizer(num_words=vocab_size)

    def make_lstm_cell(self, dropout_keep_prob):
        # CODE HERE
        pass
```