Dropout

Use dropout to help train a better LSTM model.

Chapter Goals:

- Understand the purpose of dropout in the context of RNNs

Note: While the models used in the remainder of this course are all LSTMs, the concepts from this section of the course are general to any RNN.

A. Regularization

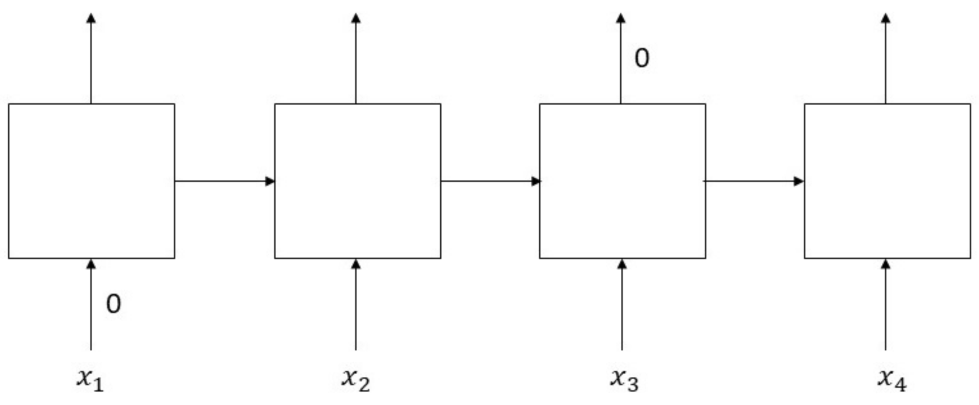
When an RNN has many weight parameters (e.g. the number of hidden units per cell is large or there are multiple RNN layers), it can have a tendency to overfit the training set and therefore generalize poorly to other data. To combat this, we need to *regularize* the RNN during training.

Regularization refers to modifying the neural network in ways that mitigate the risk of overfitting during training. A particularly popular method of regularization for feed-forward neural networks is dropout. Dropout can also be used in recurrent neural networks, although it works slightly differently.

B. Dropout in RNNs

In feed-forward neural networks, dropout refers to randomly "dropping" or zeroing-out certain hidden neurons during training. This helps each neuron in the network become more proficient and less reliant on the other neurons in the network. The fraction of neurons that are randomly dropped out is referred to as the *dropout rate*.

In recurrent neural networks, we apply dropout to the input and/or output of each cell unit. When dropout is applied to a cell's input/output at a particular time step, the cell's connection at that time step is zero'd out. So whatever, the previous input/output value was, it would be set to 0 due to the dropout.



Dropout applied to an RNN with a single cell layer and 4 time steps. Each x_i represents a time step's input. Connections labeled with 0 have dropout applied to them (i.e. zero'd out).

Applying dropout decreases the amount of cell computation that is used in the overall RNN output, thereby reducing the risk that the RNN will overfit the data.

In TensorFlow, the function we use for recurrent neural network dropout is `tf.nn.rnn_cell.DropoutWrapper`, which takes in an RNN/LSTM cell as its required argument. A couple of the more important keyword arguments for the function are `input_keep_prob` and `output_keep_prob`.

The `input_keep_prob` argument represents the probability of *not* dropping (i.e. keeping) the cell's input at each time step. The `output_keep_prob` argument represents the same thing for the cell's outputs. Their default values are both 1.0, which represents no dropout.

When training your model, you can set either argument (or both), depending on how much regularization you want. Similarly, the argument values will also influence the amount of regularization for your model. Usually, a good starting point is around 0.5 (randomly dropping half the inputs/outputs), and then adjusting based on how the model trains.

Time to Code!

In this chapter, you'll be modifying the `make_lstm_cell` function to include dropout.

We can apply (output-only) dropout to the created LSTM cell from the previous chapter. The node keep rate (i.e. 1 minus the dropout rate) is given previous chapter. The node keep rate (i.e. 1 minus the dropout rate) is given by `dropout_keep_prob`.

Set `dropout_cell` equal to `tf.nn.rnn_cell.DropoutWrapper` applied with `cell` as the required argument and `dropout_keep_prob` as the `output_keep_prob` keyword argument. Then return `dropout_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.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):
        cell = tf.nn.rnn_cell.LSTMCell(
            self.num_lstm_units) # CODE UNDER THIS LINE
```