Recurrent neural nets: training RNNs

Glenn Bruns
CSUMB

Much material in this deck from Géron, Hands-on Machine Learning with Scikit-Learn and TensorFlow

Learning outcomes

After this lecture you should be able to:

- build an RNN using TensorFlow's dynamic_rnn operation
- train an RNN

Review: a manually-created RNN

```
n_inputs = 3
n_neurons = 5

X0 = tf.placeholder(tf.float32, [None, n_inputs])
X1 = tf.placeholder(tf.float32, [None, n_inputs])

Wx = tf.Variable(tf.random_normal(shape=[n_inputs, n_neurons],
                                  dtype=tf.float32))
Wy = tf.Variable(tf.random_normal(shape=[n_neurons,n_neurons],
                                  dtype=tf.float32))
b = tf.Variable(tf.zeros([1, n_neurons], dtype=tf.float32))

# the RNN is manually unrolled over two time steps
Y0 = tf.tanh(tf.matmul(X0, Wx) + b)
Y1 = tf.tanh(tf.matmul(Y0, Wy) + tf.matmul(X1, Wx) + b)

init = tf.global_variables_initializer()
```

Letting TensorFlow do the unrolling

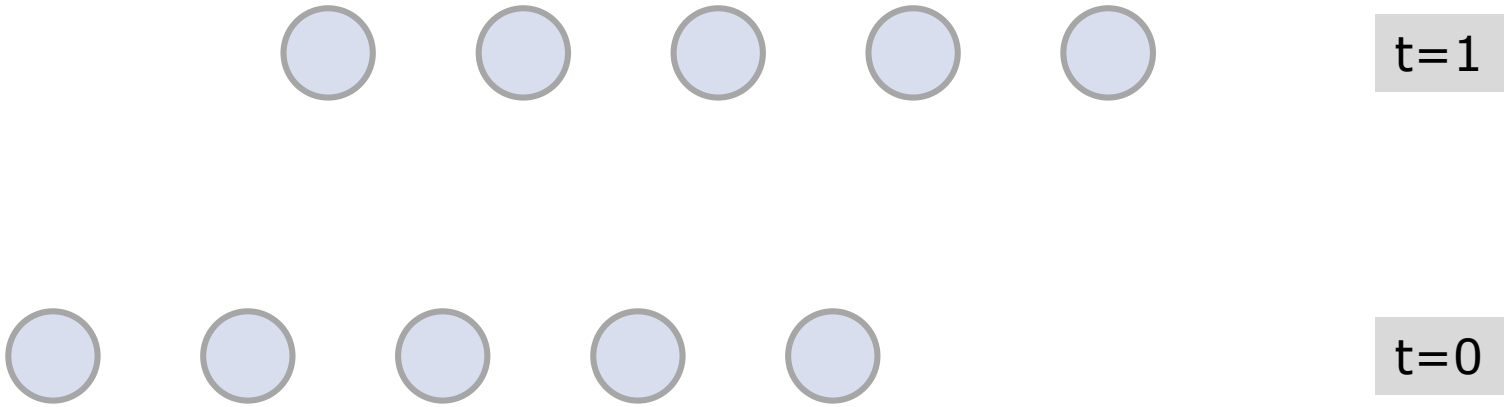
graph construction:

```
X0 = tf.placeholder(tf.float32, [None, n_inputs])
X1 = tf.placeholder(tf.float32, [None, n_inputs])

basic_cell = tf.contrib.rnn.BasicRNNCell(num_units=n_neurons)
output_seqs, states = tf.contrib.rnn.static_rnn(basic_cell,
                                                [X0, X1], dtype=tf.float32)
Y0, Y1 = output_seqs

init = tf.global_variables_initializer()
```

execution:

```
X0_batch = np.array([[0, 1, 2], [3, 4, 5], [6, 7, 8], [9, 0, 1]]) # t=0
X1_batch = np.array([[9, 8, 7], [0, 0, 0], [6, 5, 4], [3, 2, 1]]) # t=1

with tf.Session() as sess:
    init.run()
    Y0_val, Y1_val = sess.run([Y0, Y1],
                              feed_dict={X0: X0_batch, X1: X1_batch})
```

Visualizing the neurons



n_inputs = 3, n_neurons = 5

Dynamic unrolling

graph construction:

```
X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])

basic_cell = tf.contrib.rnn.BasicRNNCell(num_units=n_neurons)
outputs, states = tf.nn.dynamic_rnn(basic_cell, X,
                                    dtype=tf.float32)

init = tf.global_variables_initializer()
```

execution:

```
X_batch = np.array([  # t=0         t=1
                    [[0, 1, 2], [9, 8, 7]], # instance 1
                    [[3, 4, 5], [0, 0, 0]], # instance 2
                    [[6, 7, 8], [6, 5, 4]], # instance 3
                    [[9, 0, 1], [3, 2, 1]], # instance 4
    ])

with tf.Session() as sess:
    init.run()
    outputs_val = outputs.eval(feed_dict={X: X_batch})
```

Variable-length input sequences

graph construction:

```
# X, basic_cell, init as before
seq_length = tf.placeholder(tf.int32, [None])
outputs, states = tf.nn.dynamic_rnn(basic_cell, X, d
                type=tf.float32, sequence_length=seq_length)
```

execution:

```
X_batch = np.array([
        # step 0     step 1
        [[0, 1, 2], [9, 8, 7]], # instance 1
        [[3, 4, 5], [0, 0, 0]], # instance 2 (zero padded)
        [[6, 7, 8], [6, 5, 4]], # instance 3
        [[9, 0, 1], [3, 2, 1]], # instance 4
    ])
seq_length_batch = np.array([2, 1, 2, 2])
with tf.Session() as sess:
    init.run()
    outputs_val, states_val = sess.run([outputs, states],
       feed_dict={X: X_batch, seq_length: seq_length_batch})
```

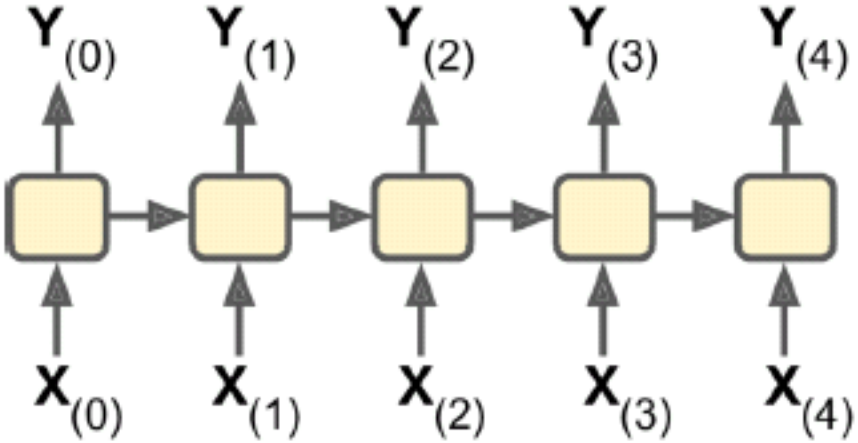
Output

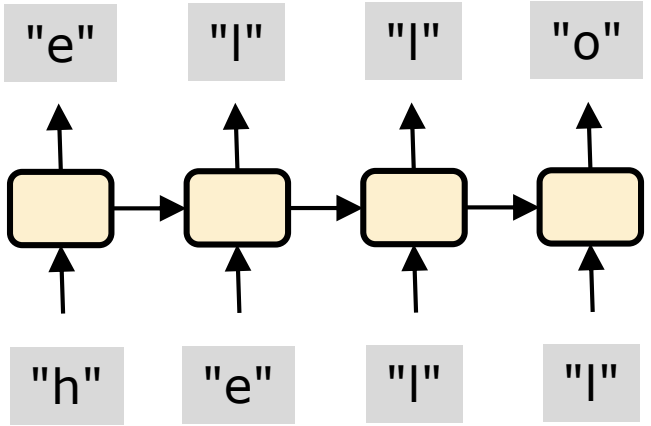
```
print(outputs_val)
[[[-0.68579942 -0.25901747 -0.80249101 -0.18141514 -0.37491533]
  [-0.99996686 -0.94501185  0.98072106 -0.96897626  0.99966902]]

 [[-0.99099374 -0.64768541 -0.67801034 -0.74154466  0.77195084]
  [ 0.          0.          0.          0.          0.        ]]

 [[-0.99978048 -0.85583007 -0.49696964 -0.93838578  0.98505181]
  [-0.99951071 -0.89148796  0.94170517 -0.38407671  0.97499222]]

 [[-0.02052618 -0.94588047  0.99935198  0.37283331  0.99981642]
  [-0.91052353  0.05769406  0.47446662 -0.44611037  0.89394677]]]
```

The RNN ouputs zero vectors for every time step past the input sequence length.

Input and output sequences: case 1

Sequence to sequence:





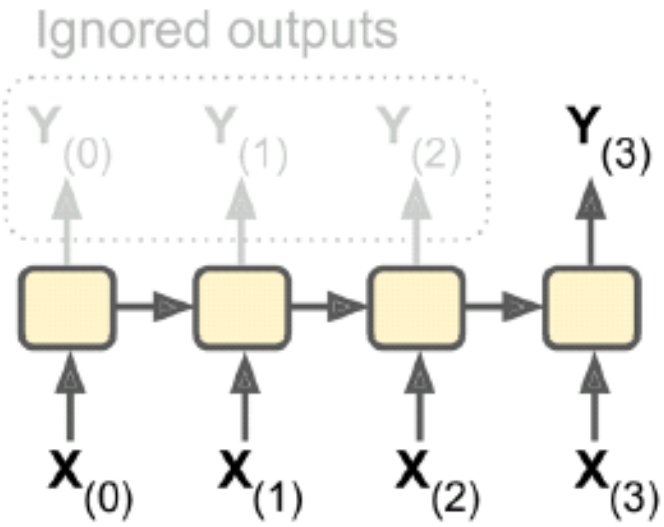
Example: inputs are daily stock prices, outputs are daily stock prices shifted by one day.

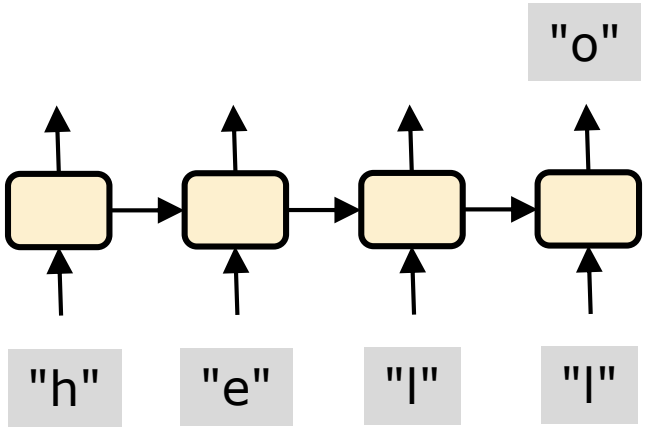
Example: character-level RNN

source: Géron, figure and text on left

Input and output sequences: case 2

Sequence to vector:





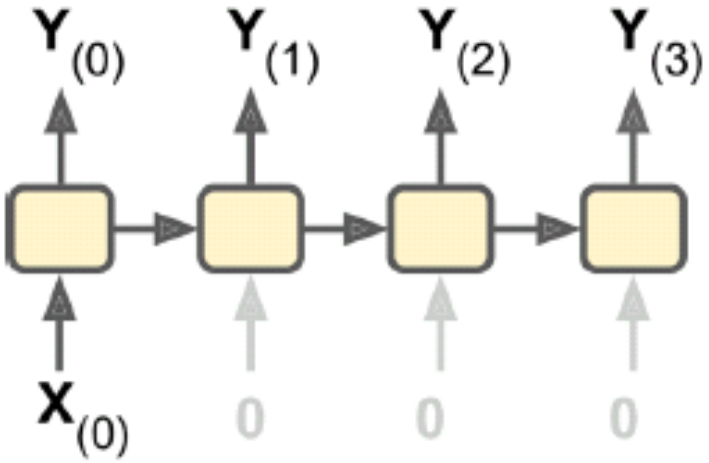
Example: inputs are a sequence of words in a movie review, output is a sentiment score.

Example: another character-level RNN

source: Géron, figure and text on left

Input and output sequences: case 3

Vector to sequence:



Example: inputs is an image, output is a caption for the image

Training an RNN

Idea:

- unroll the RNN
- use regular backpropagation for training

See text for details. This is called Backpropogation through time (BPTT).

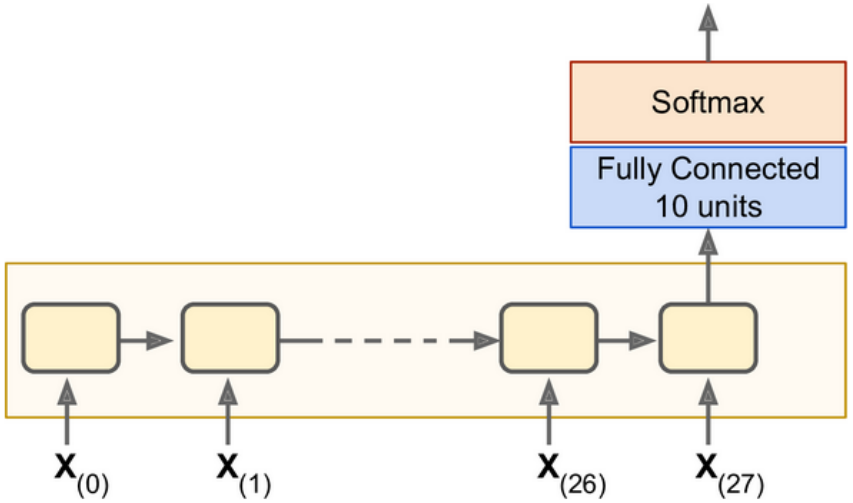
Training a sequence classifier

You can do digit classification with an RNN

- treat an 28 x 28 pixel image as a sequence of 28 rows, each with 28 pixels

The TF RNN has:

- cells of 150 recurrent neurons
- a fully connected layer of 10 neurons



Note: better to use a convolutional neural network for this task.

TensorFlow code

```
n_steps, n_inputs = 28, 28
n_neurons, n_outputs = 150, 10

learning_rate = 0.001

X = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
y = tf.placeholder(tf.int32, [None])

basic_cell = tf.contrib.rnn.BasicRNNCell(num_units=n_neurons)
outputs, states = tf.nn.dynamic_rnn(basic_cell, X, dtype=tf.float32)

logits = tf.layers.dense(states, n_outputs)    # fully connected layer
xentropy = tf.nn.sparse_softmax_cross_entropy_with_logits(labels=y,
                                                          logits=logits)
loss = tf.reduce_mean(xentropy)
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
training_op = optimizer.minimize(loss)
correct = tf.nn.in_top_k(logits, y, 1)
accuracy = tf.reduce_mean(tf.cast(correct, tf.float32))

init = tf.global_variables_initializer()
```

Summary

- much easier to build RNNs using TensorFlow's dynamic_rnn() method
- dynamic_rnn() supports variable length inputs
- to train an RNN, can use backpropagation through time (simply backprop after unrolling)
- it's possible to use RNNs for tasks like image classification