Erik Hallström Follow
Studied Engineering Physics and in Machine Learning at Royal Institute of Technology in Stockholm. Also…
Nov 19, 2016

Using the Dropout API in TensorFlow (6/7)

In the previous part we built a multi-layered LSTM RNN. In this post we will make it less prone to overfitting (called regularizing) by adding a something called dropout. It's a weird trick to randomly turn off activations of neurons during training, and was pioneered by Geoffrey Hinton among others, you can read their initial article here.

Fortunately this is very simple to do in TensorFlow, between the lines 41–42 you simply add a `DropoutWrapper` with the probability to *not* drop out, called `output_keep_prob`. Change lines 41–42 to the code below.

```
1  cell = tf.nn.rnn_cell.LSTMCell(state_size, state_is_tup
2  cell = tf.nn.rnn_cell.DropoutWrapper(cell, output_keep_
3  cell = tf.nn.rnn_cell.MultiRNNCell([cell] * num_layers,
```

Don't drop out too much or you will need a large state to be sure to keep some of the information (in our toy example at least). As you can read in this article dropout is implemented between RNN layers in TensorFlow, not on recurrent connections.

Whole program

This is the whole self-contained script, just copy and run.

```
from __future__ import print_function, division
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt

num_epochs = 100
total_series_length = 50000
truncated_backprop_length = 15
state_size = 4
num_classes = 2
echo_step = 3
batch_size = 5
num_batches = total_series_length//batch_size//truncated_backprop_length
num_layers = 3

def generateData():
    x = np.array(np.random.choice(2, total_series_length, p=[0.5, 0.5]))
    y = np.roll(x, echo_step)
    y[0:echo_step] = 0

    x = x.reshape((batch_size, -1))  # The first index changing slowest, subseries as rows
    y = y.reshape((batch_size, -1))

    return (x, y)

batchX_placeholder = tf.placeholder(tf.float32, [batch_size, truncated_backprop_length])
batchY_placeholder = tf.placeholder(tf.int32, [batch_size, truncated_backprop_length])

init_state = tf.placeholder(tf.float32, [num_layers, 2, batch_size, state_size])

state_per_layer_list = tf.unpack(init_state, axis=0)
rnn_tuple_state = tuple(
    [tf.nn.rnn_cell.LSTMStateTuple(state_per_layer_list[idx][0], state_per_layer_list[idx][1])
     for idx in range(num_layers)]
)

W = tf.Variable(np.random.rand(state_size+1, state_size), dtype=tf.float32)
b = tf.Variable(np.zeros((1,state_size)), dtype=tf.float32)

W2 = tf.Variable(np.random.rand(state_size, num_classes),dtype=tf.float32)
b2 = tf.Variable(np.zeros((1,num_classes)), dtype=tf.
```

```
42
43  # Forward passes
44  cell = tf.nn.rnn_cell.LSTMCell(state_size, state_is_t
45  cell = tf.nn.rnn_cell.DropoutWrapper(cell, output_kee
46  cell = tf.nn.rnn_cell.MultiRNNCell([cell] * num_layer
47  states_series, current_state = tf.nn.dynamic_rnn(cell
48  states_series = tf.reshape(states_series, [-1, state_
49
50  logits = tf.matmul(states_series, W2) + b2 #Broadcast
51  labels = tf.reshape(batchY_placeholder, [-1])
52
53  logits_series = tf.unpack(tf.reshape(logits, [batch_s
54  predictions_series = [tf.nn.softmax(logit) for logit
55
56  losses = tf.nn.sparse_softmax_cross_entropy_with_logi
57  total_loss = tf.reduce_mean(losses)
58
59  train_step = tf.train.AdagradOptimizer(0.3).minimize(
60
61  def plot(loss_list, predictions_series, batchX, batch
62      plt.subplot(2, 3, 1)
63      plt.cla()
64      plt.plot(loss_list)
65
66      for batch_series_idx in range(5):
67          one_hot_output_series = np.array(predictions_
68          single_output_series = np.array([(1 if out[0]
69
70          plt.subplot(2, 3, batch_series_idx + 2)
71          plt.cla()
72          plt.axis([0, truncated_backprop_length, 0, 2]
73          left_offset = range(truncated_backprop_length
74          plt.bar(left_offset, batchX[batch_series_idx,
```

Next step

In the next part we will further regularize it by using something called batch normalization. Stay tuned, it will be coming soon :)