Erik Hallström Follow
Studied Engineering Physics and in Machine Learning at Royal Institute of Technology in Stockholm. Also…
Nov 14, 2016 · 2 min read

Using the RNN API in TensorFlow (2/7)

This post is the follow up of the article "How to build a Recurrent Neural Network in TensorFlow", where we built a RNN from scratch, building up the computational graph manually. Now we will utilize the native TensorFlow API to simplify our script.

Simple graph creation

Remember where we made the unpacking and forward passes in the vanilla RNN?

```
# Unpack columns
inputs_series = tf.unpack(batchX_placeholder, axis=1)
labels_series = tf.unpack(batchY_placeholder, axis=1)
```

```
# Forward pass
current_state = init_state
states_series = []
for current_input in inputs_series:
    current_input = tf.reshape(current_input, [batch_s
    input_and_state_concatenated = tf.concat(1, [curre

    next_state = tf.tanh(tf.matmul(input_and_state_con
```

Replace the piece of code above with this:

```
# Unpack columns
inputs_series = tf.split(1, truncated_backprop_length,
labels_series = tf.unpack(batchY_placeholder, axis=1)

# Forward passes
```

You may also remove the weight- and bias matrices `W` and `b` declared earlier. The inner workings of the RNN are now hidden “under the hood”. Notice the usage of `split` instead of `unpack` when assigning the `x_inputs` variable. The `tf.nn.rnn` accepts a list of inputs of shape `[batch_size, input_size]` , and the `input_size` is simply one in our case (input is just a series of scalars). Split doesn’t remove the singular dimension, but unpack does, you can read more about it here. It doesn’t really matter anyways, since we still had to reshape the inputs in our previous example before the matrix multiplication. The `tf.nn.rnn` unrolls the RNN and creates the graph automatically, so we can remove the for-loop. The function returns a series of previous states as well as the last state in the same shape as we did before manually, here is the printed output of these variables.

Whole program

Here is the full code:

```
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//trunca

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

    x = x.reshape((batch_size, -1))  # The first inde
    y = y.reshape((batch_size, -1))

    return (x, y)

batchX_placeholder = tf.placeholder(tf.float32, [batc
batchY_placeholder = tf.placeholder(tf.int32, [batch_

init_state = tf.placeholder(tf.float32, [batch_size,

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

# Unpack columns
inputs_series = tf.split(1, truncated_backprop_length
labels_series = tf.unpack(batchY_placeholder, axis=1)

# Forward passes
cell = tf.nn.rnn_cell.BasicRNNCell(state_size)
states_series, current_state = tf.nn.rnn(cell, inputs

logits_series = [tf.matmul(state, W2) + b2 for state
```

```
predictions_series = [tf.nn.softmax(logits) for logit

losses = [tf.nn.sparse_softmax_cross_entropy_with_log
total_loss = tf.reduce_mean(losses)

train_step = tf.train.AdagradOptimizer(0.3).minimize(

def plot(loss_list, predictions_series, batchX, batch
    plt.subplot(2, 3, 1)
    plt.cla()
    plt.plot(loss_list)

    for batch_series_idx in range(5):
        one_hot_output_series = np.array(predictions_
        single_output_series = np.array([(1 if out[0]

        plt.subplot(2, 3, batch_series_idx + 2)
        plt.cla()
        plt.axis([0, truncated_backprop_length, 0, 2]
        left_offset = range(truncated_backprop_length
        plt.bar(left_offset, batchX[batch_series_idx,
        plt.bar(left_offset, batchY[batch_series_idx,
        plt.bar(left_offset, single_output_series * 0

```

Next step

In the next post we will improve the RNN by using another architecture called “Long short-term memory” or LSTM. Actually this is not necessary since our network already can solve our toy problem. But remember that our goal is to learn to use TensorFlow properly, not to solve the actual problem which is trivial :)