Tensorflow_Highway2

March 21, 2016

```
In [1]: from __future__ import absolute_import
        from __future__ import division
        from __future__ import print_function

        import time
        import tensorflow as tf

        # https://github.com/tensorflow/tensorflow/blob/master/tensorflow/examples/tutorials/mnist/inpu
        import input_data
        def weight_bias(W_shape, b_shape, bias_init=0.1):
            W = tf.Variable(tf.truncated_normal(W_shape, stddev=0.1))
            b = tf.Variable(tf.constant(bias_init, shape=b_shape), name='bias')
            return W, b

In [2]: def dense_layer(x, W_shape, b_shape, activation):
            W, b = weight_bias(W_shape, b_shape)
            return activation(tf.matmul(x, W) + b)

        def conv2d_layer(x, W_shape, b_shape, strides, padding):
            W, b = weight_bias(W_shape, b_shape)
            return tf.nn.relu(tf.nn.conv2d(x, W, strides, padding) + b)

        def highway_conv2d_layer(x, W_shape, b_shape, strides, padding, carry_bias=-1.0):
            W, b = weight_bias(W_shape, b_shape, carry_bias)
            W_T, b_T = weight_bias(W_shape, b_shape)
            H = tf.nn.relu(tf.nn.conv2d(x, W, strides, padding) + b, name='activation')
            T = tf.sigmoid(tf.nn.conv2d(x, W_T, strides, padding) + b_T, name='transform_gate')
            C = tf.sub(1.0, T, name="carry_gate")
            return tf.add(tf.mul(H, T), tf.mul(x, C), 'y') # y = (H * T) + (x * C)

In [3]: sess = tf.InteractiveSession()

In [4]: mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

```
Extracting MNIST_data/train-images-idx3-ubyte.gz
Extracting MNIST_data/train-labels-idx1-ubyte.gz
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
```

```
In [5]: x = tf.placeholder("float", [None, 784])
        y_ = tf.placeholder("float", [None, 10])

In [6]: carry_bias_init = -1.0

        x_image = tf.reshape(x, [-1, 28, 28, 1]) # reshape for conv
        keep_prob1 = tf.placeholder("float", name="keep_prob1")
        x_drop = tf.nn.dropout(x_image, keep_prob1)

        prev_y = conv2d_layer(x_drop, [5, 5, 1, 32], [32], [1, 1, 1, 1], 'SAME')
        prev_y = highway_conv2d_layer(prev_y, [3, 3, 32, 32], [32], [1, 1, 1, 1], 'SAME', carry_bias=ca
        prev_y = highway_conv2d_layer(prev_y, [3, 3, 32, 32], [32], [1, 1, 1, 1], 'SAME', carry_bias=ca

        prev_y = tf.nn.max_pool(prev_y, ksize=[1, 3, 3, 1], strides=[1, 2, 2, 1], padding='SAME')

        keep_prob2 = tf.placeholder("float", name="keep_prob2")
        prev_y = tf.nn.dropout(prev_y, keep_prob2)

        prev_y = highway_conv2d_layer(prev_y, [3, 3, 32, 32], [32], [1, 1, 1, 1], 'SAME', carry_bias=ca
        prev_y = highway_conv2d_layer(prev_y, [3, 3, 32, 32], [32], [1, 1, 1, 1], 'SAME', carry_bias=ca
        prev_y = highway_conv2d_layer(prev_y, [3, 3, 32, 32], [32], [1, 1, 1, 1], 'SAME', carry_bias=ca

        prev_y = tf.nn.max_pool(prev_y, ksize=[1, 3, 3, 1], strides=[1, 2, 2, 1], padding='SAME')

        keep_prob3 = tf.placeholder("float", name="keep_prob3")
        prev_y = tf.nn.dropout(prev_y, keep_prob3)

        prev_y = highway_conv2d_layer(prev_y, [3, 3, 32, 32], [32], [1, 1, 1, 1], 'SAME', carry_bias=ca
        prev_y = highway_conv2d_layer(prev_y, [3, 3, 32, 32], [32], [1, 1, 1, 1], 'SAME', carry_bias=ca
        prev_y = highway_conv2d_layer(prev_y, [3, 3, 32, 32], [32], [1, 1, 1, 1], 'SAME', carry_bias=ca

        prev_y = tf.nn.max_pool(prev_y, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')

        keep_prob4 = tf.placeholder("float", name="keep_prob4")
        prev_y = tf.nn.dropout(prev_y, keep_prob4)

        prev_y = tf.reshape(prev_y, [-1, 4 * 4 * 32])
        y = dense_layer(prev_y, [4 * 4 * 32, 10], [10], tf.nn.softmax)
```

```
In [7]: with tf.name_scope("loss") as scope:
            cross_entropy = -tf.reduce_sum(y_ * tf.log(y))

        with tf.name_scope("train") as scope:
            train_step = tf.train.GradientDescentOptimizer(1e-2).minimize(cross_entropy)

        with tf.name_scope("test") as scope:
            correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
            accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
```

```
In [8]: tf.initialize_all_variables().run()
```

```
In [9]: batch_size = 50

        for i in range(2000):
            batch_xs, batch_ys = mnist.train.next_batch(batch_size)

            if i % 100 == 0:
                train_accuracy = accuracy.eval(feed_dict={
                    x: batch_xs,
                    y_: batch_ys,
                    keep_prob1: 1.0,
                    keep_prob2: 1.0,
                    keep_prob3: 1.0,
                    keep_prob4: 1.0,
                })
                print("step %d, training accuracy %g" % (i, train_accuracy))

            train_step.run(feed_dict={
                x: batch_xs,
                y_: batch_ys,
                keep_prob1: 0.8,
                keep_prob2: 0.7,
                keep_prob3: 0.6,
                keep_prob4: 0.5,
            })
```

```
step 0, training accuracy 0.1
step 100, training accuracy 0.1
step 200, training accuracy 0.16
step 300, training accuracy 0.14
step 400, training accuracy 0.12
step 500, training accuracy 0.12
step 600, training accuracy 0.12
step 700, training accuracy 0.12
step 800, training accuracy 0.34
step 900, training accuracy 0.6
step 1000, training accuracy 0.74
step 1100, training accuracy 0.44
step 1200, training accuracy 0.64
step 1300, training accuracy 0.64
step 1400, training accuracy 0.74
step 1500, training accuracy 0.72
step 1600, training accuracy 0.98
step 1700, training accuracy 0.96
step 1800, training accuracy 0.92
step 1900, training accuracy 0.94
```

```
In [10]: print("test accuracy %g" % accuracy.eval(feed_dict={
             x: mnist.test.images,
             y_: mnist.test.labels,
             keep_prob1: 1.0,
             keep_prob2: 1.0,
             keep_prob3: 1.0,
             keep_prob4: 1.0,
         }))
```

```
test accuracy 0.9431
```

```
In [ ]:
```