COSC4337_CNN_1

[1]:
```python
# Convolutional Neural Network.

#Build and train a convolutional neural network with TensorFlow.


from __future__ import division, print_function, absolute_import

import tensorflow as tf

# Import MNIST data
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("/tmp/data/", one_hot=True)

# Training Parameters
learning_rate = 0.001
num_steps = 200
batch_size = 128
display_step = 10

# Network Parameters
num_input = 784 # MNIST data input (img shape: 28*28)
num_classes = 10 # MNIST total classes (0-9 digits)
dropout = 0.75 # Dropout, probability to keep units

# tf Graph input
X = tf.placeholder(tf.float32, [None, num_input])
Y = tf.placeholder(tf.float32, [None, num_classes])
keep_prob = tf.placeholder(tf.float32) # dropout (keep probability)


# Create some wrappers for simplicity
def conv2d(x, W, b, strides=1):
    # Conv2D wrapper, with bias and relu activation
    x = tf.nn.conv2d(x, W, strides=[1, strides, strides, 1], padding='SAME')
    x = tf.nn.bias_add(x, b)
    return tf.nn.relu(x)
```

```python
def maxpool2d(x, k=2):
    # MaxPool2D wrapper
    return tf.nn.max_pool(x, ksize=[1, k, k, 1], strides=[1, k, k, 1],
                          padding='SAME')


# Create model
def conv_net(x, weights, biases, dropout):
    # MNIST data input is a 1-D vector of 784 features (28*28 pixels)
    # Reshape to match picture format [Height x Width x Channel]
    # Tensor input become 4-D: [Batch Size, Height, Width, Channel]
    x = tf.reshape(x, shape=[-1, 28, 28, 1])

    # Convolution Layer
    conv1 = conv2d(x, weights['wc1'], biases['bc1'])
    # Max Pooling (down-sampling)
    conv1 = maxpool2d(conv1, k=2)

    # Convolution Layer
    conv2 = conv2d(conv1, weights['wc2'], biases['bc2'])
    # Max Pooling (down-sampling)
    conv2 = maxpool2d(conv2, k=2)

    # Fully connected layer
    # Reshape conv2 output to fit fully connected layer input
    fc1 = tf.reshape(conv2, [-1, weights['wd1'].get_shape().as_list()[0]])
    fc1 = tf.add(tf.matmul(fc1, weights['wd1']), biases['bd1'])
    fc1 = tf.nn.relu(fc1)
    # Apply Dropout
    fc1 = tf.nn.dropout(fc1, dropout)

    # Output, class prediction
    out = tf.add(tf.matmul(fc1, weights['out']), biases['out'])
    return out

# Store layers weight & bias
weights = {
    # 5x5 conv, 1 input, 32 outputs
    'wc1': tf.Variable(tf.random_normal([5, 5, 1, 32])),
    # 5x5 conv, 32 inputs, 64 outputs
    'wc2': tf.Variable(tf.random_normal([5, 5, 32, 64])),
    # fully connected, 7*7*64 inputs, 1024 outputs
    'wd1': tf.Variable(tf.random_normal([7*7*64, 1024])),
    # 1024 inputs, 10 outputs (class prediction)
    'out': tf.Variable(tf.random_normal([1024, num_classes]))
}
```

```python
biases = {
    'bc1': tf.Variable(tf.random_normal([32])),
    'bc2': tf.Variable(tf.random_normal([64])),
    'bd1': tf.Variable(tf.random_normal([1024])),
    'out': tf.Variable(tf.random_normal([num_classes]))
}

# Construct model
logits = conv_net(X, weights, biases, keep_prob)
prediction = tf.nn.softmax(logits)

# Define loss and optimizer
loss_op = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(
    logits=logits, labels=Y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate)
train_op = optimizer.minimize(loss_op)


# Evaluate model
correct_pred = tf.equal(tf.argmax(prediction, 1), tf.argmax(Y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))

# Initialize the variables (i.e. assign their default value)
init = tf.global_variables_initializer()

# Start training
with tf.Session() as sess:

    # Run the initializer
    sess.run(init)

    for step in range(1, num_steps+1):
        batch_x, batch_y = mnist.train.next_batch(batch_size)
        # Run optimization op (backprop)
        sess.run(train_op, feed_dict={X: batch_x, Y: batch_y, keep_prob: 0.8})
        if step % display_step == 0 or step == 1:
            # Calculate batch loss and accuracy
            loss, acc = sess.run([loss_op, accuracy], feed_dict={X: batch_x,
                                                                 Y: batch_y,
                                                                 keep_prob: 1.
 ↪0})
            print("Step " + str(step) + ", Minibatch Loss= " + \
                  "{:.4f}".format(loss) + ", Training Accuracy= " + \
                  "{:.3f}".format(acc))

    print("Optimization Finished!")
```

```python
# Calculate accuracy for 256 MNIST test images
print("Testing Accuracy:", \
    sess.run(accuracy, feed_dict={X: mnist.test.images[:256],
                                  Y: mnist.test.labels[:256],
                                  keep_prob: 1.0}))
```

```
WARNING:tensorflow:From <ipython-input-1-8305f85ca654>:12: read_data_sets (from
tensorflow.contrib.learn.python.learn.datasets.mnist) is deprecated and will be
removed in a future version.
Instructions for updating:
Please use alternatives such as official/mnist/dataset.py from
tensorflow/models.
WARNING:tensorflow:From C:\Users\RizkN\.conda\envs\tf1\lib\site-
packages\tensorflow_core\contrib\learn\python\learn\datasets\mnist.py:260:
maybe_download (from tensorflow.contrib.learn.python.learn.datasets.base) is
deprecated and will be removed in a future version.
Instructions for updating:
Please write your own downloading logic.
WARNING:tensorflow:From C:\Users\RizkN\.conda\envs\tf1\lib\site-
packages\tensorflow_core\contrib\learn\python\learn\datasets\mnist.py:262:
extract_images (from tensorflow.contrib.learn.python.learn.datasets.mnist) is
deprecated and will be removed in a future version.
Instructions for updating:
Please use tf.data to implement this functionality.
Extracting /tmp/data/train-images-idx3-ubyte.gz
WARNING:tensorflow:From C:\Users\RizkN\.conda\envs\tf1\lib\site-
packages\tensorflow_core\contrib\learn\python\learn\datasets\mnist.py:267:
extract_labels (from tensorflow.contrib.learn.python.learn.datasets.mnist) is
deprecated and will be removed in a future version.
Instructions for updating:
Please use tf.data to implement this functionality.
Extracting /tmp/data/train-labels-idx1-ubyte.gz
WARNING:tensorflow:From C:\Users\RizkN\.conda\envs\tf1\lib\site-
packages\tensorflow_core\contrib\learn\python\learn\datasets\mnist.py:110:
dense_to_one_hot (from tensorflow.contrib.learn.python.learn.datasets.mnist) is
deprecated and will be removed in a future version.
Instructions for updating:
Please use tf.one_hot on tensors.
Extracting /tmp/data/t10k-images-idx3-ubyte.gz
Extracting /tmp/data/t10k-labels-idx1-ubyte.gz
WARNING:tensorflow:From C:\Users\RizkN\.conda\envs\tf1\lib\site-
packages\tensorflow_core\contrib\learn\python\learn\datasets\mnist.py:290:
DataSet.__init__ (from tensorflow.contrib.learn.python.learn.datasets.mnist) is
deprecated and will be removed in a future version.
Instructions for updating:
Please use alternatives such as official/mnist/dataset.py from
tensorflow/models.
```

```
WARNING:tensorflow:From <ipython-input-1-8305f85ca654>:68: calling dropout (from
tensorflow.python.ops.nn_ops) with keep_prob is deprecated and will be removed
in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 -
keep_prob`.
WARNING:tensorflow:From <ipython-input-1-8305f85ca654>:99:
softmax_cross_entropy_with_logits (from tensorflow.python.ops.nn_ops) is
deprecated and will be removed in a future version.
Instructions for updating:

Future major versions of TensorFlow will allow gradients to flow
into the labels input on backprop by default.

See `tf.nn.softmax_cross_entropy_with_logits_v2`.

Step 1, Minibatch Loss= 61103.8516, Training Accuracy= 0.117
Step 10, Minibatch Loss= 25802.3828, Training Accuracy= 0.227
Step 20, Minibatch Loss= 11076.3027, Training Accuracy= 0.516
Step 30, Minibatch Loss= 6006.2930, Training Accuracy= 0.711
Step 40, Minibatch Loss= 4226.5283, Training Accuracy= 0.805
Step 50, Minibatch Loss= 4199.2275, Training Accuracy= 0.828
Step 60, Minibatch Loss= 5109.8262, Training Accuracy= 0.812
Step 70, Minibatch Loss= 1290.2396, Training Accuracy= 0.938
Step 80, Minibatch Loss= 2960.1343, Training Accuracy= 0.867
Step 90, Minibatch Loss= 2245.3245, Training Accuracy= 0.898
Step 100, Minibatch Loss= 1517.3873, Training Accuracy= 0.922
Step 110, Minibatch Loss= 3113.6824, Training Accuracy= 0.859
Step 120, Minibatch Loss= 2153.8550, Training Accuracy= 0.906
Step 130, Minibatch Loss= 1314.6887, Training Accuracy= 0.898
Step 140, Minibatch Loss= 1413.1868, Training Accuracy= 0.930
Step 150, Minibatch Loss= 818.0255, Training Accuracy= 0.922
Step 160, Minibatch Loss= 1505.7168, Training Accuracy= 0.891
Step 170, Minibatch Loss= 873.8877, Training Accuracy= 0.953
Step 180, Minibatch Loss= 1286.6614, Training Accuracy= 0.930
Step 190, Minibatch Loss= 1718.3186, Training Accuracy= 0.922
Step 200, Minibatch Loss= 764.5211, Training Accuracy= 0.953
Optimization Finished!
Testing Accuracy: 0.94140625
```

[ ]: