CNN

June 10, 2018

```
In [1]: import tensorflow as tf
        import numpy as np
        import math
        import matplotlib.pyplot as plt
        from mpl_toolkits.axes_grid1 import ImageGrid
        from tensorflow.examples.tutorials.mnist import input_data
```

```
/home/swaroop/anaconda2/envs/tf-env/lib/python2.7/site-packages/h5py/__init__.py:36: FutureWarni
  from ._conv import register_converters as _register_converters
```

```
WARNING:tensorflow:From /home/swaroop/anaconda2/envs/tf-env/lib/python2.7/site-packages/tensorfl
Instructions for updating:
Use the retry module or similar alternatives.
```

1 Building Adversarial for CNN

```
In [2]: mnist = input_data.read_data_sets('MNIST_data', one_hot=True)
```

```
WARNING:tensorflow:From <ipython-input-2-1390ba3ba838>:1: read_data_sets (from tensorflow.contri
Instructions for updating:
Please use alternatives such as official/mnist/dataset.py from tensorflow/models.
WARNING:tensorflow:From /home/swaroop/anaconda2/envs/tf-env/lib/python2.7/site-packages/tensorfl
Instructions for updating:
Please write your own downloading logic.
WARNING:tensorflow:From /home/swaroop/anaconda2/envs/tf-env/lib/python2.7/site-packages/tensorfl
Instructions for updating:
Please use tf.data to implement this functionality.
Extracting MNIST_data/train-images-idx3-ubyte.gz
WARNING:tensorflow:From /home/swaroop/anaconda2/envs/tf-env/lib/python2.7/site-packages/tensorfl
Instructions for updating:
Please use tf.data to implement this functionality.
Extracting MNIST_data/train-labels-idx1-ubyte.gz
WARNING:tensorflow:From /home/swaroop/anaconda2/envs/tf-env/lib/python2.7/site-packages/tensorfl
Instructions for updating:
Please use tf.one_hot on tensors.
```

```
Extracting MNIST_data/t10k-images-idx3-ubyte.gz
Extracting MNIST_data/t10k-labels-idx1-ubyte.gz
WARNING:tensorflow:From /home/swaroop/anaconda2/envs/tf-env/lib/python2.7/site-packages/tensorfl
Instructions for updating:
Please use alternatives such as official/mnist/dataset.py from tensorflow/models.
```

1.1 CNN model

```
In [3]: sess = tf.InteractiveSession()
        def weight_variable(shape):
            initial = tf.truncated_normal(shape, stddev=0.1)
            return tf.Variable(initial)

        def bias_variable(shape):
            initial = tf.constant(0.1, shape=shape)
            return tf.Variable(initial)

        # Functions for convolution and pooling functions
        def conv2d(x, W):
            return tf.nn.conv2d(x, W, strides=[1,1,1,1], padding='SAME')

        def max_pooling_2x2(x):
            return tf.nn.max_pool(x, ksize=[1,2,2,1], strides=[1,2,2,1], padding='SAME')

        x = tf.placeholder(tf.float32, shape=[None, 784])
        y_ = tf.placeholder(tf.float32, shape=[None, 10])

        x_image = tf.reshape(x, [-1,28,28,1]) # mnist image comes in as 784 vector

        # Conv layer 1 - 32x5x5
        W_conv1 = weight_variable([5, 5, 1, 32])
        b_conv1 = bias_variable([32])
        x_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
        x_pool1 = max_pooling_2x2(x_conv1)

        # Conv layer 2 - 64x5x5
        W_conv2 = weight_variable([5, 5, 32, 64])
        b_conv2 = bias_variable([64])
        x_conv2 = tf.nn.relu(conv2d(x_pool1, W_conv2) + b_conv2)
        x_pool2 = max_pooling_2x2(x_conv2)

        # Flatten - keras 'flatten'
        x_flat = tf.reshape(x_pool2, [-1, 7*7*64])

        # Dense fully connected layer
        W_fc1 = weight_variable([7 * 7 * 64, 1024]) # max pooling reduced image to 7x7
        b_fc1 = bias_variable([1024])
```

```
        x_fc1 = tf.nn.relu(tf.matmul(x_flat, W_fc1) + b_fc1)

        # Regularization with dropout
        keep_prob = tf.placeholder(tf.float32)
        x_fc1_drop = tf.nn.dropout(x_fc1, keep_prob)

        # Classification layer
        W_fc2 = weight_variable([1024, 10])
        b_fc2 = bias_variable([10])
        y_conv = tf.matmul(x_fc1_drop, W_fc2) + b_fc2
        y = tf.nn.softmax(y_conv)

        cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits
        train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
        correct_prediction = tf.equal(tf.argmax(y_conv,1), tf.argmax(y_,1))
        accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))


        sess.run(tf.global_variables_initializer())


        for i in range(600):
            batch = mnist.train.next_batch(100)
            if i%200 == 0:
                train_accuracy = accuracy.eval(feed_dict={x:batch[0], y_: batch[1], keep_prob: 1
                print("step %d, training accuracy %g"%(i, train_accuracy))

            train_step.run(feed_dict={x: batch[0], y_: batch[1], keep_prob: 0.4})

        print("test accuracy %g"%accuracy.eval(feed_dict={x: mnist.test.images[0:500],
                                                          y_: mnist.test.labels[0:500], keep_pro
```

```
WARNING:tensorflow:From <ipython-input-3-2aa7045217ae>:52: softmax_cross_entropy_with_logits (fr
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 0, training accuracy 0.07
step 200, training accuracy 0.93
step 400, training accuracy 0.9
test accuracy 0.958
```

1.2 Adversarial for CNN

In [4]:
```
img_adv_list = list()
y_label = [0,0,0,0,0,0,1,0,0,0]
eps = 0.1
index = 1

for index in range(100):
    image_norm = mnist.test.images[index]
    image_norm = np.reshape(image_norm, (1, 784))

    original_image = image_norm

    loss =  tf.nn.softmax_cross_entropy_with_logits(labels=y_label, logits=y)
    deriv = tf.gradients(loss, x)

    image_adv = tf.stop_gradient(x - tf.sign(deriv)*eps)
    image_adv = tf.clip_by_value(image_adv, 0, 1)

    #dydx = sess.run(deriv, {x: x_image, keep_prob: 1.0})
    x_adv = sess.run(image_adv, {x: original_image, keep_prob: 1.0})
    x_image = np.reshape(x_adv, (1, 784))
    img_adv_list.append(np.squeeze(x_image))

a = np.array(img_adv_list)
print("Adversial accuracy %g"%accuracy.eval(feed_dict={x: a,
                                                       y_: mnist.test.labels[0:100], keep_pr
```

```
Adversial accuracy 0.67
```

We could see that the accuracy dropped to 67 percent after adding adversarial noise for CNN.

2 Building Adversarial for logistic regression

In [6]:
```
x_train = mnist.train.images
y_train = mnist.train.labels
x_test = mnist.test.images
y_test = mnist.test.labels

learning_rate = 0.01
training_epochs = 25
batch_size = 100
display_step = 1

# tf Graph Input
x_logistic = tf.placeholder(tf.float32, [None, 784]) # mnist data image of shape 28*28=7
#print(x)
```

```
y_logistic = tf.placeholder(tf.float32, [None, 10]) # 0-9 digits recognition => 10 class

# Set model weights
W_logistic = tf.Variable(tf.zeros([784, 10]))
b_logistic = tf.Variable(tf.zeros([10]))

# Construct model
pred_logistic = tf.nn.softmax(tf.matmul(x_logistic, W_logistic) + b_logistic) # Softmax

# Minimize error using cross entropy
cost_logistic = tf.reduce_mean(-tf.reduce_sum(y_logistic*tf.log(pred_logistic), reductio
grads_logistic = tf.gradients(cost_logistic, [x_logistic])[0]



# Gradient Descent
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost_logistic)

# Initialize the variables (i.e. assign their default value)

# Run the initializer
sess.run(tf.global_variables_initializer())

# Training cycle
for epoch in range(training_epochs):
    avg_cost = 0.
    total_batch = int(mnist.train.num_examples/batch_size)
    # Loop over all batches
    for i in range(total_batch):
        batch_xs, batch_ys = mnist.train.next_batch(batch_size)
        # Run optimization op (backprop) and cost op (to get loss value)
        _, c = sess.run([optimizer, cost_logistic], feed_dict={x_logistic: batch_xs,
                                                        y_logistic: batch_ys})
        # Compute average loss
        avg_cost += c / total_batch
    # Display logs per epoch step
    if (epoch+1) % display_step == 0:
        print("Epoch:", '%04d' % (epoch+1), "cost=", "{:.9f}".format(avg_cost))

print("Optimization Finished!")

# Test model
correct_prediction = tf.equal(tf.argmax(pred_logistic, 1), tf.argmax(y_logistic, 1))
# Calculate accuracy
accuracy_logistic = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print("Accuracy:", accuracy_logistic.eval(feed_dict={x_logistic: mnist.test.images, y_lo
eps = 0.1
gradients = sess.run(grads_logistic,feed_dict={x_logistic: mnist.test.images ,y_logistic
```

```
        print np.shape(gradients)
        adversial_image = list()
        for i in range(100):
            adversial_image.append(mnist.test.images[i] + eps * np.sign(gradients[mnist.test.lab
        print("Accuracy for adversarial", accuracy_logistic.eval(feed_dict={x_logistic: adversia
```

```
('Epoch:', '0001', 'cost=', '1.184220838')
('Epoch:', '0002', 'cost=', '0.665903673')
('Epoch:', '0003', 'cost=', '0.551804216')
('Epoch:', '0004', 'cost=', '0.498392707')
('Epoch:', '0005', 'cost=', '0.464794250')
('Epoch:', '0006', 'cost=', '0.440419118')
('Epoch:', '0007', 'cost=', '0.429076707')
('Epoch:', '0008', 'cost=', '0.412730855')
('Epoch:', '0009', 'cost=', '0.400757651')
('Epoch:', '0010', 'cost=', '0.392332228')
('Epoch:', '0011', 'cost=', '0.384225340')
('Epoch:', '0012', 'cost=', '0.378427407')
('Epoch:', '0013', 'cost=', '0.371771220')
('Epoch:', '0014', 'cost=', '0.369578379')
('Epoch:', '0015', 'cost=', '0.359863018')
('Epoch:', '0016', 'cost=', '0.359782431')
('Epoch:', '0017', 'cost=', '0.355925974')
('Epoch:', '0018', 'cost=', '0.351616819')
('Epoch:', '0019', 'cost=', '0.347447850')
('Epoch:', '0020', 'cost=', '0.343778826')
('Epoch:', '0021', 'cost=', '0.344152260')
('Epoch:', '0022', 'cost=', '0.339711116')
('Epoch:', '0023', 'cost=', '0.338246835')
('Epoch:', '0024', 'cost=', '0.337168307')
('Epoch:', '0025', 'cost=', '0.332706073')
Optimization Finished!
('Accuracy:', 0.9139)
(10000, 784)
('Accuracy for adversarial', 0.7)
```

We could see that the accuracy dropped to 70 percent for logistic regression

3 Using adversarial of CNN for logistic regression

```
In [8]: print("Accuracy for adversarial", accuracy_logistic.eval(feed_dict={x_logistic: a, y_log
```

```
('Accuracy for adversarial', 0.93)
```

4 Using adversarial of logistic for CNN

```
In [9]: print("Adversial accuracy %g"%accuracy.eval(feed_dict={x: a,
                                                              y_: mnist.test.labels[0:100], keep_pr
```

```
Adversial accuracy 0.22
```

5 Creating 60000 samples for logistic regression model

```
In [13]: adversial_image_500 = list()

         for i in range(500):
             adversial_image_500.append(mnist.train.images[i] + eps * np.sign(gradients[mnist.tr
             adversial_image_500.append(mnist.train.labels[i])
```

```
In [14]: total_train_set = [mnist.train.images, adversial_image_5000]
         total_label_set = [mnist.train.labels, adversial_image_5000]
```

5.1 Training logistic with the adversarial and training data

```
In [17]: for epoch in range(training_epochs):
             avg_cost = 0.
             total_batch = int(len(total_train_set)/batch_size)
             # Loop over all batches
             for i in range(total_batch-1):
                 batch_xs = total_train_set[i*batch_size:(i+1)*batch_size]
                 batch_ys = total_label_set[i*batch_size:(i+1)*batch_size]
                 # Run optimization op (backprop) and cost op (to get loss value)
                 _, c = sess.run([optimizer, cost_logistic], feed_dict={x_logistic: batch_xs,
                                                                        y_logistic: batch_ys})
                 # Compute average loss
                 avg_cost += c / total_batch
             # Display logs per epoch step
             if (epoch+1) % display_step == 0:
                 print("Epoch:", '%04d' % (epoch+1))

         print("Optimization Finished!")
         print("Accuracy:", accuracy_logistic.eval(feed_dict={x_logistic: mnist.test.images, y_l
```

```
('Epoch:', '0001', 'cost=', '0.000000000')
('Epoch:', '0002', 'cost=', '0.000000000')
('Epoch:', '0003', 'cost=', '0.000000000')
('Epoch:', '0004', 'cost=', '0.000000000')
('Epoch:', '0005', 'cost=', '0.000000000')
('Epoch:', '0006', 'cost=', '0.000000000')
('Epoch:', '0007', 'cost=', '0.000000000')
('Epoch:', '0008', 'cost=', '0.000000000')
```

```
('Epoch:', '0009', 'cost=', '0.000000000')
('Epoch:', '0010', 'cost=', '0.000000000')
('Epoch:', '0011', 'cost=', '0.000000000')
('Epoch:', '0012', 'cost=', '0.000000000')
('Epoch:', '0013', 'cost=', '0.000000000')
('Epoch:', '0014', 'cost=', '0.000000000')
('Epoch:', '0015', 'cost=', '0.000000000')
('Epoch:', '0016', 'cost=', '0.000000000')
('Epoch:', '0017', 'cost=', '0.000000000')
('Epoch:', '0018', 'cost=', '0.000000000')
('Epoch:', '0019', 'cost=', '0.000000000')
('Epoch:', '0020', 'cost=', '0.000000000')
('Epoch:', '0021', 'cost=', '0.000000000')
('Epoch:', '0022', 'cost=', '0.000000000')
('Epoch:', '0023', 'cost=', '0.000000000')
('Epoch:', '0024', 'cost=', '0.000000000')
('Epoch:', '0025', 'cost=', '0.000000000')
Optimization Finished!
('Accuracy:', 0.9139)
```

5.2 Training CNN with adversarial and training data

```
In [20]: for i in range(1200):
             batch_xs = total_train_set[i*batch_size:(i+1)*batch_size]
             batch_ys = total_label_set[i*batch_size:(i+1)*batch_size]
             train_step.run(feed_dict={x: batch[0], y_: batch[1], keep_prob: 0.4})

         print("test accuracy %g"%accuracy.eval(feed_dict={x: mnist.test.images[0:500],
                                                           y_: mnist.test.labels[0:500], keep_pr
```

```
test accuracy 0.866
```

Does classification performance improve? The classification performance should improve for the following reasons: The number of training samples is increased. Hence it will be able to better learn the underlying features better. Also, the model will be trained on images in which it is prone to making mistakes as adversarials are meant to trick the model. In the code above we used only 500 adversarial examples as it was taking a vey long time to generate 60000 examples.

I am guessing the reasoning with the dropped accuracy is that since the number of adversarial examples were very less, it was not able to find the underlying pattern. If higher number of examples were used then it training accuracy could be more.

Is the new model less or more susceptible to adversarial examples? The model is less prone to adversrial examples for the same reason as mentioned above.

Do you think you can use a regularization method in order to make the model less susceptible to adversarial examples? Regularization such as dropout can be used for CNNs to make it less susceptible for adversarial attacks.