CNN_adversarial

June 10, 2018

In [103]:
```
%matplotlib inline
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import ImageGrid
import numpy as np
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
```

In [104]:
```
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 [105]:
```
sess = tf.InteractiveSession()
```

In [106]:
```
# Functions for creating weights and biases
# https://www.tensorflow.org/get_started/mnist/pros
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')
```

In [107]:
```
# Create placeholders nodes for images and label inputs# Creat
x = tf.placeholder(tf.float32, shape=[None, 784])
y_ = tf.placeholder(tf.float32, shape=[None, 10])
```

In [108]:
```
# y = (Wx +b)
# https://www.tensorflow.org/get_started/mnist/pros

# Input layer
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
```

In [109]:
```
# Probabilities - output from model (not the same as logits)
y = tf.nn.softmax(y_conv)
```

In [110]:
```
# Loss and optimizer# Loss a
cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y_, log
train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
```

In [111]:
```
# Setup to test accuracy of model
correct_prediction = tf.equal(tf.argmax(y_conv,1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

In [112]:
```
# Initilize all global variables
sess.run(tf.global_variables_initializer())
```

In [113]:
```
# Train model
# Run once to get the model to a good confidence level
for i in range(500):
    batch = mnist.train.next_batch(100)
    if i%100 == 0:
        train_accuracy = accuracy.eval(feed_dict={x:batch[0], y_: batch[1], keep_prob
        print("step %d, training accuracy %g"%(i, train_accuracy))

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

```
step 0, training accuracy 0.2
step 100, training accuracy 0.89
step 200, training accuracy 0.92
step 300, training accuracy 0.93
step 400, training accuracy 0.96
```

In [114]:
```
# Run trained model against test data
print("test accuracy %g"%accuracy.eval(feed_dict={x: mnist.test.images[0:500],
                                                  y_: mnist.test.labels[0:500], keep_
```

```
test accuracy 0.944
```

In [115]:
```
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))

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

Adversial accuracy 0.62

In [116]:
```
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*
#print(x)
y_logistic = tf.placeholder(tf.float32, [None, 10]) # 0-9 digits recognition => 10 c

# 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) # Soft

# Minimize error using cross entropy
cost_logistic = tf.reduce_mean(-tf.reduce_sum(y_logistic*tf.log(pred_logistic), redu
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
            eps = 0.1
            gradients = sess.run(grads_logistic,feed_dict={x_logistic: mnist.test.images ,y_logis
            adversial_image = list()
            for i in range(100):
                adversial_image.append(mnist.test.images[i] + eps * np.sign(gradients[mnist.test
            print("Accuracy for adversarial", accuracy_logistic.eval(feed_dict={x_logistic: adver
```

```
Epoch: 0001 cost= 1.184346928
Epoch: 0002 cost= 0.666558521
Epoch: 0003 cost= 0.552374808
Epoch: 0004 cost= 0.499382836
Epoch: 0005 cost= 0.465472445
Epoch: 0006 cost= 0.442538566
Epoch: 0007 cost= 0.425703498
Epoch: 0008 cost= 0.411542967
Epoch: 0009 cost= 0.401541156
Epoch: 0010 cost= 0.392360053
Epoch: 0011 cost= 0.385793734
Epoch: 0012 cost= 0.377451270
Epoch: 0013 cost= 0.371519373
Epoch: 0014 cost= 0.368496240
Epoch: 0015 cost= 0.363197807
Epoch: 0016 cost= 0.356623211
Epoch: 0017 cost= 0.355426335
Epoch: 0018 cost= 0.352173921
Epoch: 0019 cost= 0.348494050
Epoch: 0020 cost= 0.345959686
Epoch: 0021 cost= 0.340833946
Epoch: 0022 cost= 0.341859334
Epoch: 0023 cost= 0.338874917
Epoch: 0024 cost= 0.334670528
Epoch: 0025 cost= 0.334672520
Optimization Finished!
Accuracy: 0.9138
```

```
Accuracy for adversarial 0.7
```

```
In [117]: #using Cnn adversarial images for logistic regression model
          print("Accuracy for adversarial", accuracy_logistic.eval(feed_dict={x_logistic: new_
```

```
Accuracy for adversarial 0.91
```

```
In [118]: # Using adversarial images of logistic for CNN
          print("Adversial accuracy %g"%accuracy.eval(feed_dict={x: new_img_list,
                                                                  y_: mnist.test.labels[0:100], keep
```

```
Adversial accuracy 0.12
```

```
In [119]: sess.close()
```

```
In [ ]: #Creating adversarials images
        adversarial_images_ultimate = []
        num_adversaries_ultimate = len(mnist.test.images)
        with tf.Session() as sess:
            sess.run(init)
            print("Creating ... {} training examples" .format(num_adversaries_ultimate))
           #Creation of same number of training data of adversarial examples
           # for logistic regression
            grad = sess.run([grad_input], feed_dict={x:  mnist.train.images,y:  mnist.train.lab
            grad = np.array(grad)
            grad = grad.reshape(55000,784)

            for i in range(num_adversaries_ultimate):
                img_ad = mnist.train.images[i] + eps * np.sign(grad[mnist.train.labels.tolist()
                adversarial_images_ultimate.append(img_ad)

            print("Finishing generating the adversarial examples")
            print("Number of adversarials", len(adversarial_images_ultimate))
        adversarial_images_ultimate_bk = adversarial_images_ultimate[1:10001]
        print(len(adversarial_images_ultimate_bk))
        adversarial_images_ultimate = adversarial_images_ultimate_bk
        adversarial_images_ultimate_label = []
        print(len(adversarial_images_ultimate))

        for i in range(num_adversaries_ultimate):
            adversarial_images_ultimate.append(mnist.train.images[i])
           #adding a set of training images
        for i in range(num_adversaries_ultimate):
            adversarial_images_ultimate_label.append(list(mnist.train.labels[i]))
        for i in range(num_adversaries_ultimate):
        #     print(mnist.train.labels[i])
```

```
            adversarial_images_ultimate_label.append(list(mnist.train.labels[i]))
        print(len(adversarial_images_ultimate))
        print(len(adversarial_images_ultimate_label))
        #Training with the ultimate adversarials list of 20,000 images
        # Start training
        with tf.Session() as sess:
            sess.run(init)

            print("Starting training")
            # Training cycle
            for epoch in range(training_epochs):
                avg_cost = 0.
                total_batch = int(len(adversarial_images_ultimate)/batch_size)
                print(total_batch)
                # Loop over all batches
        #         for i in range(total_batch):
        #         batch_xs = adversarial_images_ultimate[i*100:(i+1)*100]
                batch_xs = adversarial_images_ultimate
        #         batch_ys = adversarial_images_ultimate_label[i*100:(i+1)*100]
                batch_ys = adversarial_images_ultimate_label

                # Fit training using batch data
                _, c = sess.run([optimizer, loss], feed_dict={x: batch_xs,
                                                              y: 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 ("Training Finished!")

            print("Starting testing")
            # Test model
            correct_prediction = tf.equal(tf.argmax(y_pred, 1), tf.argmax(y, 1))

            # Accuracy model
            accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

            #Calculating the accuracy with the testing data.
            print ("Accuracy Testing:", accuracy.eval({x: mnist.test.images, y: mnist.test.labe
```

Reference: https://github.com/jasonicarter/MNIST-adversarial-images/blob/master/MNIST-adversarial-images.ipynb