Preliminary Exercise
Sobel Filters

a) Write the Sobel\_x and Sobel\_y filters

b) Show a numerical example of how Sobel filters detect edges

c) What operation is needed before applying Sobel filters?

d) What problems can you get into in the previous operation? Show a numerical example

e) Applying the Sobel filters, we obtain the Dx\_image and Dy\_image. How can we create an ‘only edge’ image?
General overview of Neural Network and Tensorflow up to now
What’s a Neural Network?

• Biological Inspiration

• An approximation to make them work

• Neurons, connections, activation functions, weights and bias
Dimension of the Weights

- Blackboard examples
The key ideas behind a network

- Forward and Backward step
- Gradient Descent
- Partial derivation w.r.t. weights variables
- Steps and Epochs
Regression vs. Classification (1/3)

- Classification is the task of predicting a discrete class label.

- Regression is the task of predicting a continuous quantity.
A classification algorithm may predict a continuous value, but the continuous value is in the form of a probability for a class label.

A regression algorithm may predict a discrete value, but the discrete value is in the form of an integer quantity.
• Classification predictions can be evaluated using accuracy, whereas regression predictions cannot.

• Regression predictions can be evaluated using root mean squared error, whereas classification predictions cannot.
Base elements of a Network

tf.Variable(value, name="exampleVar")
tf.constant(value, name="exampleConst")
tf.placeholder(dtype=tf.int32, size=(), name="scalarPlaceholder")

Lots of operations:
tf.transpose(), tf.reduce_mean() and etc.
TF.NN and TF.Layer

• Tensorflow.nn offers elements for your network. If the component needs a weight and/or bias vector, you need to create the actual tf.Variable()
  
  e.g. tf.nn.batch_normalization()
  tf.nn.conv2d(input, filters,....)

• Tensorflow.layer offers a tool to make your life easier. Weights and bias are automatically managed. Save and restore works on them as a single variable.
  
  e.g. tf.layer.conv2d(input, #filters, filter_size, ...)
Manual Implementation - basic setting for MNIST

```python
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot = True)

a0 = tf.placeholder(tf.float32, [None, 784])

y = tf.placeholder(tf.float32, [None, 10])

eta = tf.constant(0.5)
```
Manual Implementation - WEIGHTS and BIAS

numNodi_middle = 30

w1 = tf.Variable(tf.truncated_normal([784, numNodi_middle]))
b1 = tf.Variable(tf.truncated_normal([1, numNodi_middle]))
w2 = tf.Variable(tf.truncated_normal([numNodi_middle, 10]))
b2 = tf.Variable(tf.truncated_normal([1, 10]))
SIGMOID ACTIVATION FUNCTION

```python
def sigmoid_f(x):
    return tf.div(tf.constant(1.0), tf.add(tf.constant(1.0), tf.exp(tf.negative(x)))))
```

DERIVATIVE OF THE SIGMOID

```python
def d_sig(x):
    return tf.multiply(sigmoid_f(x), tf.subtract(tf.constant(1.0), sigmoid_f(x)))
```
Manual Implementation - FORWARD PROPAGATION

• #Primo Layer
  • $z_1 = \text{tf.add}(\text{tf.matmul}(a_0, w_1), b_1)$
  • $a_1 = \text{sigmoid}_f(z_1)$

• #Secondo Layer
  • $z_2 = \text{tf.add}(\text{tf.matmul}(a_1, w_2), b_2)$
  • $a_2 = \text{sigmoid}_f(z_2)$
diff = tf.subtract(a2, y)

d_z2 = tf.multiply(diff, d_sig(z2))
d_b2 = d_z2
d_w2 = tf.matmul(tf.transpose(a1), d_z2)
d_a1 = tf.matmul(d_z2, tf.transpose(w2))

d_z1 = tf.multiply(d_a1, d_sig(z1))
d_b1 = d_z1

d_w1 = tf.matmul(tf.transpose(a0), d_z1)
Manual Implementation

```
step = [
    tf.assign(w1, tf.subtract(w1, tf.multiply(eta, d_w1))),
    tf.assign(b1, tf.subtract(b1, tf.multiply(eta,
       tf.reduce_mean(d_b1, axis=[0])))),
    tf.assign(w2, tf.subtract(w2, tf.multiply(eta, d_w2))),
    tf.assign(b2, tf.subtract(b2, tf.multiply(eta,
       tf.reduce_mean(d_b2, axis=[0]))))
]
```
Manual Implementation

acct_mat = tf.equal(tf.argmax(a2, 1), tf.argmax(y, 1))
acct_res = tf.reduce_sum(tf.cast(acct_mat, tf.float32))

s = tf.InteractiveSession()
s.run(tf.global_variables_initializer())
totRange= 10000
for i in range(totRange):
    batch_x, batch_y = mnist.train.next_batch(10)
    s.run(step, feed_dict = {a0: batch_x, y: batch_y})
An Important Issue

• Vanishing Gradient

• Exploding Gradient
Regularization

- Regularization in the loss function
- Dropout
Convolutional NN
General concepts

• Wrap up everything the course presented up to now

• Powerful tool to work with images

• The most used network along with RNN (next lecture)

• Convolution can be 1D, 2D, 3D, ...

• SPARSE NEURONS CONNECTIONS
Convolution net comes in two steps

• Every convolution layer is followed by a Pooling:
  - Max
  - Average
  - Many others

• The whole convolutional part of the network is usually followed by a fully connected to offer an actual classification
Key concept: Meaningful Dimensions

• Conv2d layers have 3d weights and outputs:
  Spatial information kept (w.r.t. Width and Height)

  Filter informations kept (a slice w.r.t. Depth)
Stride recall

• It generally appears in two component that state how much the filter moves in each direction (x, y)
• In signal processing is assumed to have the form (1, 1)
• An increase of the stride values, reduce the output dimension
• Particular cases are: stride (1, 1) and non overlapping stride (filter_x_dim, filter_y_dim)
• In tensorflow, it has 4 components: it considers also channel and batch movement directions (but usually set to 1)
Key role of Padding

- In tensorflow, we have: padding='VALID' and padding='SAME' as parameters of conv2d function.

- **padding='VALID'** = no padding applied (dimensionality reduction or 1x1 filters case)

- **padding='SAME'** = zero padding applied

- Keep in mind that you can applied the tf.pad() function to decide to use other types of padding.
Semi-Manual Implementation - Libraries

%matplotlib inline
import matplotlib.pyplot as plt
import tensorflow as tf
import numpy as np
from sklearn.metrics import confusion_matrix
import time
from datetime import timedelta
import math
from tensorflow.examples.tutorials.mnist import input_data

data = input_data.read_data_sets("MNIST_data/", one_hot=True)
Semi-Manual Implementation - Placeholders

```python
x = tf.placeholder(tf.float32, shape=[None, 784], name='x')
x_image = tf.reshape(x, [-1, 28, 28, 1])
y_true = tf.placeholder(tf.float32, shape=[None, 10], name='y_true')
y_true_cls = tf.argmax(y_true, axis=1)
```
with tf.name_scope("conv_1"):
    weights_1 = tf.Variable(tf.truncated_normal(shape=[5,5,1,16], stddev=0.05))
    biases_1 = tf.Variable(tf.constant(0.05, shape=[16]))
    layer_1 = tf.nn.conv2d(input = x_image, filter =
                            weights_1, strides=[1,1,1,1], padding='SAME')
    layer_1 += biases_1
    layer_1 = tf.nn.max_pool(value=layer_1, ksize=[1,2,2,1], strides=[1,2,2,1],
                              padding='SAME')
Semi-Manual Implementation - Printing the First layer

In Jupyter:
layer_1

Output:
<tf.Tensor 'conv_1/MaxPool:0' shape=(?, 14, 14, 16) dtype=float32>
with tf.name_scope("conv_2"):  
    weights_2 = tf.Variable(tf.truncated_normal(shape=[5,5,16,36], stddev=0.05))  
    biases_2 = tf.Variable(tf.constant(0.05,shape=[36]))  
    layer_2 = tf.nn.conv2d(input = layer_1,filter = weights_2,strides=[1,1,1,1],padding='SAME')  
    layer_2 += biases_2  
    layer_2 = tf.nn.max_pool(value=layer_2, ksize=[1,2,2,1],strides=[1,2,2,1], padding='SAME')
Semi-Manual Implementation - Flattening needed for a FC layer

```python
with tf.name_scope("flatten"):
    layer_shape = layer_2.get_shape()
    num_features = layer_shape[1:4].num_elements()
    layer_flat = tf.reshape(layer_2, [-1, num_features])
```
with tf.name_scope("fc_1"):  
weights_3 = tf.Variable(tf.truncated_normal([num_features, 128]))  
biases_3 = tf.Variable(tf.constant(0.05, shape=[128]))  
layer_3 = tf.add(tf.matmul(layer_flat, weights_3), biases_3)  
layer_3 = tf.nn.relu(layer_3)
Semi-Manual Implementation - Fully connected two

with tf.name_scope("fc_2"):
weights_4 = tf.Variable(tf.truncated_normal([128, 10]))
biases_4 = tf.Variable(tf.constant(0.05, shape=[10]))
layer_4 = tf.add(tf.matmul(layer_3, weights_4), biases_4)
y_pred = tf.nn.softmax(layer_4)

y_pred_cls = tf.argmax(y_pred, axis=1)

cross_entropy = 
    tf.nn.softmax_cross_entropy_with_logits(logits=layer_4, labels=y_true)
Semi-Manual Implementation - Towards the loss and accuracy (2/2)

cost = tf.reduce_mean(cross_entropy)
optimizer = tf.train.AdamOptimizer(learning_rate=1e-4).minimize(cost)

correct_prediction = tf.equal(y_pred_cls, y_true_cls)
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
Semi-Manual Implementation - Pre-Training settings

session = tf.Session()
session.run(tf.global_variables_initializer())
batch_size = 64
num_iterations = 1
start_time = time.time()

for i in range(num_iterations):
    print(i)
    x_batch, y_true_batch = data.train.next_batch(batch_size)

    feed_dict_train = {x: x_batch,
                      y_true: y_true_batch}
session.run(optimizer, feed_dict=feed_dict_train)
    if i % 100 == 0:
        acc = session.run(accuracy, feed_dict=feed_dict_train)
        print("Optimization Iteration:", str(i), "Training Accuracy:", acc)

end_time = time.time()
time_dif = end_time - start_time