Example of Neural Networks and Convolutional Neural Networks using Tensorflow

Sargur Srihari, Jun Chu
University at Buffalo, State University of New York

Topics

- Implement single hidden layer neural network, train it on the MNIST digit images and tune hyperparameters such as the number of units in the hidden layer
- Use a publicly available convolutional neural network package, train it on the MNIST digit images and tune hyperparameters

MNIST Dataset

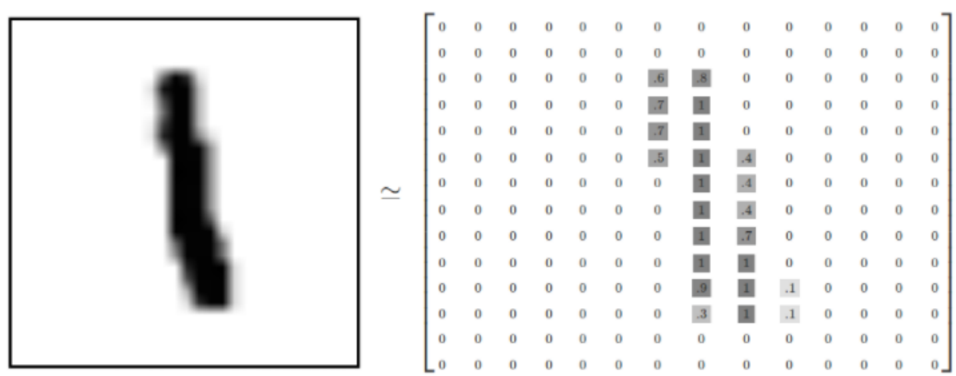
- MNIST is a simple computer vision dataset
- MNIST consists of images of handwritten digits

MNIST Dataset

- The following code will download and read in the data automatically

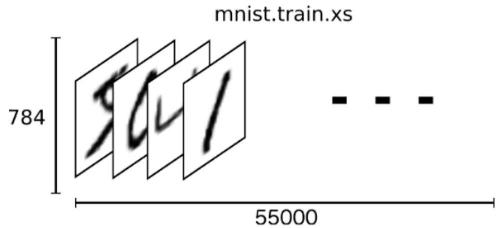
```
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
```

- Each image is 28 pixels by 28 pixels. We can interpret this as a big array of numbers



MNIST Dataset

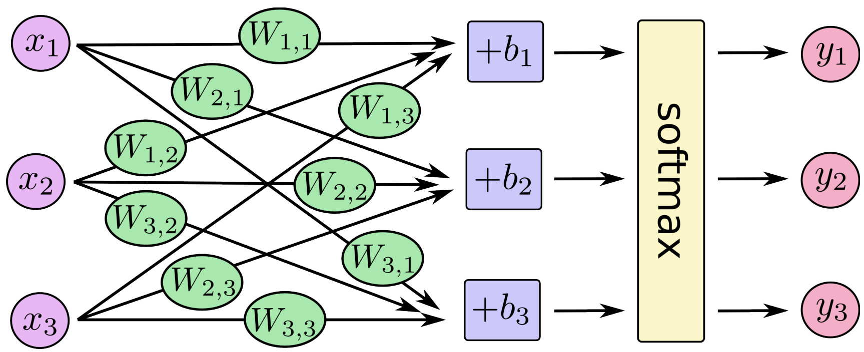
- The result is that `mnist.train.images` is a tensor (an n-dimensional array) with a shape of [55000, 784].



- The first dimension is an index into the list of images and the second dimension is the index for each pixel in each image.
- Each entry in the tensor is a pixel intensity between 0 and 1, for a particular pixel in a particular image.

Softmax Regression

- For each output, we compute a weighted sum of the xs, add a bias, and then apply softmax.



- If we write that out as equations, we get:

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{array}{c} W_{1,1}x_1 + W_{1,2}x_2 + W_{1,3}x_3 + b_1 \\ W_{2,1}x_1 + W_{2,2}x_2 + W_{2,3}x_3 + b_2 \\ W_{3,1}x_1 + W_{3,2}x_2 + W_{3,3}x_3 + b_3 \end{array} \right)$$

Softmax Regression

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\begin{pmatrix} W_{1,1}x_1 + W_{1,2}x_2 + W_{1,3}x_3 + b_1 \\ W_{2,1}x_1 + W_{2,2}x_2 + W_{2,3}x_3 + b_2 \\ W_{3,1}x_1 + W_{3,2}x_2 + W_{3,3}x_3 + b_3 \end{pmatrix}$$

- We can "vectorize" this procedure, turning it into a matrix multiplication and vector addition

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \text{softmax}\left(\begin{bmatrix} W_{1,1} & W_{1,2} & W_{1,3} \\ W_{2,1} & W_{2,2} & W_{2,3} \\ W_{3,1} & W_{3,2} & W_{3,3} \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}\right)$$

Softmax Regression

- import TensorFlow

```
import tensorflow as tf
```

- Input

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

- Parameter Variable

```
W = tf.Variable(tf.zeros([784, 10]))

b = tf.Variable(tf.zeros([10]))
```

- Softmax regression model:

```
y = tf.nn.softmax(tf.matmul(x, W) + b)
```

Softmax Regression

- To implement cross-entropy we need to first add a new placeholder

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

- cross-entropy function

```
cross_entropy = tf.reduce_mean(
    -tf.reduce_sum(y_ * tf.log(y),
    reduction_indices=[1]))
```

- apply your choice of optimization algorithm to modify the variables and reduce the loss

```
train_step =
  tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
```

Softmax Regression

- We can now launch the model in an InteractiveSession

```
sess = tf.InteractiveSession()
```

- create an operation to initialize the variables we created

```
tf.global_variables_initializer().run()
```

- we'll run the training step 1000 times

```
for _ in range(1000):
  batch_xs, batch_ys = mnist.train.next_batch(100)
  sess.run(train_step, feed_dict={x: batch_xs, y_:
batch_ys})
```

Evaluating Our Model

- check if our prediction matches the truth

```
correct_prediction = tf.equal(tf.argmax(y,1),
                              tf.argmax(y_,1))
```

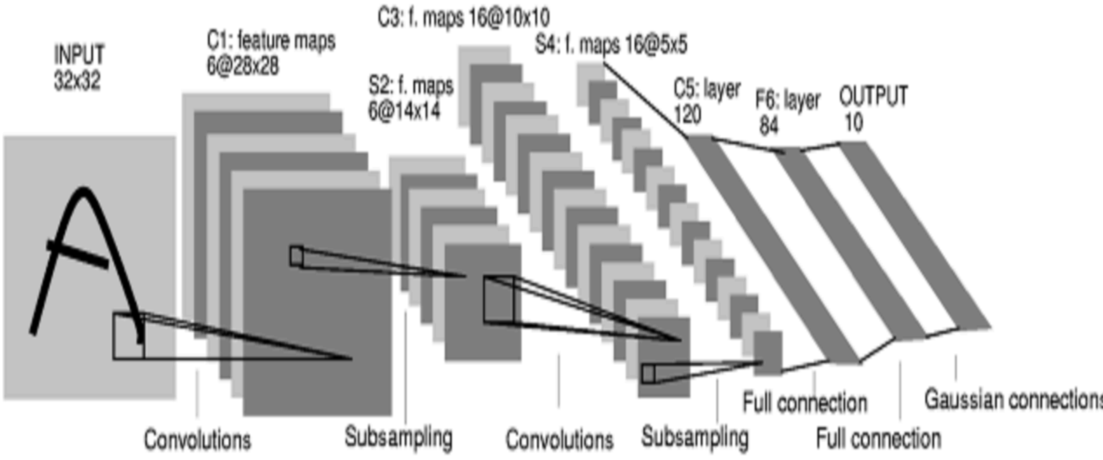
- To determine what fraction are correct, we cast to floating point numbers and then take the mean

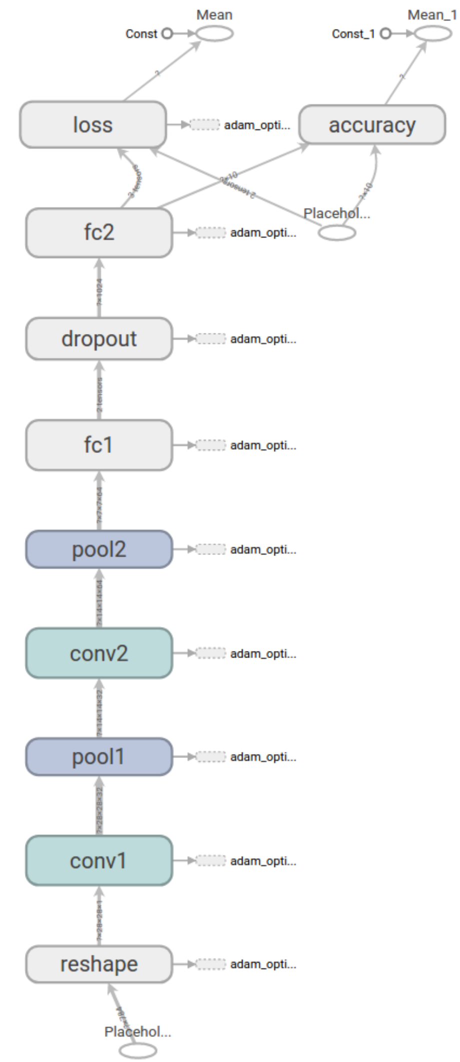
```
accuracy =
tf.reduce_mean(tf.cast(correct_prediction,
                               tf.float32))
```

- Finally, we ask for our accuracy on our test data

```
print(sess.run(accuracy, feed_dict={x:
mnist.test.images, y_: mnist.test.labels}))
```

Multilayer Convolutional Network





Multilayer Convolutional Network

- Weight Initialization

```
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)
```

- Convolution and Pooling

```
def conv2d(x, W):
  return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

def max_pool_2x2(x):
  return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
                        strides=[1, 2, 2, 1], padding='SAME')
```

Multilayer Convolutional Network

- ## First Convolutional Layer

```
W_conv1 = weight_variable([5, 5, 1, 32])
b_conv1 = bias_variable([32])x_image = tf.reshape(x, [-1, 28, 28, 1])

h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
h_pool1 = max_pool_2x2(h_conv1)
```

- ## Second Convolutional Layer

```
W_conv2 = weight_variable([5, 5, 32, 64])
b_conv2 = bias_variable([64])

h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) +
b_conv2)
h_pool2 = max_pool_2x2(h_conv2)
```

Multilayer Convolutional Network

- Densely Connected Layer

```
W_fc1 = weight_variable([7 * 7 * 64, 1024])
b_fc1 = bias_variable([1024])

h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)
```

- Dropout

```
keep_prob = tf.placeholder(tf.float32)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)
```

- Readout Layer

```
W_fc2 = weight_variable([1024, 10])
b_fc2 = bias_variable([10])

y_conv = tf.matmul(h_fc1_drop, W_fc2) + b_fc2
```

Multilayer Convolutional Network

- Train and Evaluate the Model

```
cross_entropy = tf.reduce_mean(
    tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y_conv))
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))

with tf.Session() as sess:
  sess.run(tf.global_variables_initializer())
  for i in range(20000):
    batch = mnist.train.next_batch(50)
    if i % 100 == 0:
      train_accuracy = accuracy.eval(feed_dict={
          x: batch[0], y_: batch[1], keep_prob: 1.0})
      print('step %d, training accuracy %g' % (i, train_accuracy))
    train_step.run(feed_dict={x: batch[0], y_: batch[1], keep_prob:
0.5})

  print('test accuracy %g' % accuracy.eval(feed_dict={
      x: mnist.test.images, y_: mnist.test.labels, keep_prob: 1.0}))
```