Introduction to Tensor Flow

Objectives

- In this presentation, you will learn about:
  - Tensor Flow
  - The SoftMax Regression model
  - How to train the model using cross-entropy
  - How to evaluate the model

The Data

- We will be using the MNIST data set of hand-written digits.
- Please download the data from here: http://yann.lecun.com/exdb/mnist/
- The data should be downloaded to your working directory with your ipython notebook
- You will need the following files:

```
train-images-idx3-ubyte.gz:  training set images (9912422 bytes)
train-labels-idx1-ubyte.gz:  training set labels (28881 bytes)
t10k-images-idx3-ubyte.gz:   test set images (1648877 bytes)
t10k-labels-idx1-ubyte.gz:   test set labels (4542 bytes)
```

Load the Data

- Ensure input_data.py is in your working directory *(NOTE: supply participants with this file)*

- Now, load the data with the following two commands:

```
import input_data

mnist = input_data.read_data_sets("MNIST_data/",
one_hot=True)
```

Softmax Regression

- For each image, assign probability it is one of the digits between 0 - 9
  - e.g image of a 9: 80% probability it is a 9, 5% probability it is an 8, 15% probability it is some other number
- Softmax regression is good for assigning an object to probabilities that it is a one of a set of things

Softmax Regression

- Two steps:
  - Add up evidence input is in a certain class
  - Convert evidence into probabilities

- Evidence: weighted sum of pixel intensities
  - **Negative weight**: pixel with high intensity evidence object **is not** in that class
  - **Positive weight**: pixel with high intensity evidence object **is** in that class

Softmax Regression

- Mathematically:

$$E_i = \sum_j W_{i,j} x_j + b_i$$

W_i = weights, b_i is the bias, x is the output image

- Convert evidence to predicted probabilities:

$$y = softmax(E)$$

Softmax Regression

- Vector notation:

$$\begin{bmatrix} y_1 \\ y_2 \\ ... \\ y_i \end{bmatrix} = softmax\left(\begin{bmatrix} W_{1,1} & W_{1,2} & ... & W_{1,j} \\ W_{2,1} & W_{2,2} & ... & W_{2,j} \\ ... & ... & ... & ... \\ W_{i,1} & W_{i,2} & ... & W_{i,j} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ ... \\ x_i \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ ... \\ b_i \end{bmatrix} \right)$$

- In simpler form:

$$y = softmax(Wx + b)$$

Implementing the Regression

- Tensor flow defines a graph of interacting operations using symbolic variables.
- We will define symbolic variables for x, W and b, which are needed for the softmax regression.

```
import tensorflow as tf


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

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

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

Implementing the Regression

- Now, we define the model using one line of code:

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

Training the Model

- We will train our model by minimizing the cost function
- The cost function we will use is called the cross-entropy function, defined as:

$$H_{y'}(y) = -\sum_i y'_i log(y_i)$$

Training the Model
Implementing Cross-Entropy

- First, we define a new placeholder to store the correct answers

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

- Define the entropy function

```
cross_entropy = -tf.reduce_sum(y_*tf.log(y))
```

Training the Model

- Backpropogation is used to evaluate how variables affect the minimization of the cost function
- The variables are modified using an optimization function, in this case gradient descent

```
train_step =
tf.train.GradientDescentOptimizer(0.01).minimize(cross_entropy)
```

optimizer: gradient descent

cost function: cross entropy

- Gradient descent shifts the variables a small amount in the direction that minimizes the cost function

Training the Model

- Initialize the variables

```
init = tf.initialize_all_variables()
```

- Start a tensor flow session

```
sess = tf.Session()

sess.run(init)
```

Training the Model

- Now, we train our model

```
for i in range(1000):

  batch_xs, batch_ys = mnist.train.next_batch(100)

  sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
```

- Select a batch of 100 random samples from the training set
- Repeat training step 1000 times
- This method is called stochastic training

Evaluate the Model

- Compute how many predictions were correctly made

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

tf.argmax(y,1): Best (highest) prediction in y tensor along axis 1

tf.argmax(y_,1): Correct y label

Evaluate the Model

- Compute the model accuracy

```
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
```

- Convert boolean values in correct_prediction array to floating point numbers
- Then compute mean

Evaluate the Model

- Lastly, tensor flow is used to compute accuracy of the test data

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

- The accuracy should be around 91%

Acknowledgements

This presentation is based off of the MNIST For ML Beginners tensor flow tutorial (https://www.tensorflow.org/versions/master/tutorials/mnist/beginners/index.html)