```
Tensorflow Example: Logistic Regression
======================================
```

In [1]:
```python
# Run this cell to import all the libraries you need for this exercise
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
```

In [2]:
```python
# STEP 1: Read in data
from tensorflow.examples.tutorials.mnist import input_data
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 [3]:
```python
def TRAIN_SIZE(num):
    print ('Total Training Images in Dataset = ' + str(mnist.train.images.shape))
    print ('--------------------------------------------------')
    x_train = mnist.train.images[:num,:]
    print ('x_train Examples Loaded = ' + str(x_train.shape))
    y_train = mnist.train.labels[:num,:]
    print ('y_train Examples Loaded = ' + str(y_train.shape))
    print('')
    return x_train, y_train

def TEST_SIZE(num):
    print ('Total Test Examples in Dataset = ' + str(mnist.test.images.shape))
    print ('--------------------------------------------------')
    x_test = mnist.test.images[:num,:]
    print ('x_test Examples Loaded = ' + str(x_test.shape))
    y_test = mnist.test.labels[:num,:]
    print ('y_test Examples Loaded = ' + str(y_test.shape))
    return x_test, y_test
```

In [4]:
```python
# STEP 2: Define parameters for the model
X_train, Y_train = TRAIN_SIZE(5500)
X_test, Y_test = TEST_SIZE(1000)
learning_rate = 0.01
train_steps = 2500
```

```
Total Training Images in Dataset = (55000, 784)
--------------------------------------------------
x_train Examples Loaded = (5500, 784)
y_train Examples Loaded = (5500, 10)

Total Test Examples in Dataset = (10000, 784)
--------------------------------------------------
x_test Examples Loaded = (1000, 784)
y_test Examples Loaded = (1000, 10)
```

```
Tensorflow functions used:

placeholder       for input data
Variable          for parameters to be learnt
matmul            matrix multiplication
nn.softmax        softmax function
random_normal     outputs random values from a normal distribution
zeros             creates a tensor with all elements set to zero
log(x)            computes natural logarithm of x element-wise
reduce_sum        computes the sum of elements across dimensions of a tensor
argmax            returns the index with the largest value across axes of a tensor
equal(x,y)        returns the truth value of (x == y) element-wise
cast              casts a tensor to a new type
reduce_mean       computes the mean of elements across dimensions of a tensor
```

In [5]:
```python
# n features, k classes
n = 784
k = 10

# STEP 3: Create placeholders for features and labels
X = tf.placeholder(tf.float32, [None, n])
Y = tf.placeholder(tf.float32, [None, k])

# STEP 4: Create weights and bias
W = tf.Variable(tf.random_normal(shape=[n,k], stddev=0.01), name="weights")
b = tf.Variable(tf.zeros([k]), name="bias")

# STEP 5: Predict y from X and W, b
y = tf.nn.softmax(tf.matmul(X, W) + b)

# STEP 6: Define loss function
loss = tf.reduce_mean(-tf.reduce_sum(Y * tf.log(y), reduction_indices=[1]))

# STEP 7: Gradient Descent
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(loss)

# STEP 8: Calculate accuracy
correct_prediction = tf.equal(tf.argmax(Y,1), tf.argmax(y,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

In [6]:
```python
# Initializing the variables
init = tf.global_variables_initializer()

# Launch the graph
with tf.Session() as sess:
    sess.run(init)

    # Train model
    for i in range(train_steps+1):
        sess.run([optimizer, loss], feed_dict={X: X_train,Y: Y_train})
        if i%100 == 0:
            print('Training Step: ', i)
            print('Accuracy:', sess.run(accuracy, feed_dict={X: X_test, Y: Y_test}))

    # Obtain weights
    W_values = sess.run(W)
```

```
Training Step:  0
Accuracy: 0.084
Training Step:  100
Accuracy: 0.728
Training Step:  200
Accuracy: 0.763
Training Step:  300
Accuracy: 0.78
Training Step:  400
Accuracy: 0.793
Training Step:  500
Accuracy: 0.803
Training Step:  600
Accuracy: 0.812
Training Step:  700
Accuracy: 0.82
Training Step:  800
Accuracy: 0.83
Training Step:  900
Accuracy: 0.834
Training Step:  1000
Accuracy: 0.839
Training Step:  1100
Accuracy: 0.842
Training Step:  1200
Accuracy: 0.847
Training Step:  1300
Accuracy: 0.851
Training Step:  1400
Accuracy: 0.852
Training Step:  1500
Accuracy: 0.852
Training Step:  1600
Accuracy: 0.853
Training Step:  1700
Accuracy: 0.856
Training Step:  1800
Accuracy: 0.856
Training Step:  1900
Accuracy: 0.858
Training Step:  2000
Accuracy: 0.858
Training Step:  2100
Accuracy: 0.859
Training Step:  2200
Accuracy: 0.86
Training Step:  2300
Accuracy: 0.86
Training Step:  2400
Accuracy: 0.861
Training Step:  2500
Accuracy: 0.861
```

In [7]:
```
print(W_values)
```

```
[[ 0.00824239  0.01554364  0.00204719 ..., -0.02057871  0.01179873
  -0.00500055]
 [-0.00026903  0.00970864 -0.00314839 ...,  0.00593256 -0.00174163
  -0.00043793]
 [-0.00216794  0.01184921 -0.01560484 ..., -0.01186555 -0.02076972
  -0.00705317]
 ...,
 [ 0.01165243 -0.00494103  0.01567424 ..., -0.00410011 -0.00310922
   0.01497457]
 [-0.01311398 -0.00242065 -0.00950717 ..., -0.02925431 -0.01496194
   0.01571412]
 [ 0.01072468  0.01011477 -0.02105916 ..., -0.00809694  0.01144223
  -0.00033818]]
```

In [8]:
```
for i in range(10):
    plt.subplot(2, 5, i+1)
    weight = W_values[:,i]
    plt.title(i)
    plt.imshow(weight.reshape([28,28]), cmap=plt.get_cmap('seismic'))
    frame1 = plt.gca()
    frame1.axes.get_xaxis().set_visible(False)
    frame1.axes.get_yaxis().set_visible(False)
plt.show()
```