Code:

```
import tensorflow as tf
import numpy as np
import input_data
```

dependencies: implementation in tensorflow and numpy

```
class parameters():

    def __init__(self):
        self.LEARNING_RATE = 0.05
        self.REG = 0
        self.NUM_EPOCHS = 500
        self.DISPLAY_STEP = 1 # epoch
```

hyperparameters: the learning rate and regularization rates are very important. Should be finalized using empirical evidence.

```
def load_data():
    mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)
    trainX, trainY, testX, testY = mnist.train.images, mnist.train.labels, mnist.test.images, mnist.test.labels
    return trainX, trainY, testX, testY
```

load data: create train and test sets with y as one hot encoded vectors

LOGISTIC REGRESSION - CODE

```
def model(config):

    # Placeholders
    X = tf.placeholder("float", [None, 784])
    y = tf.placeholder("float", [None, 10])

    # Weights
    with tf.variable_scope('weights'):
        W = tf.Variable(tf.random_normal([784, 10], stddev=0.01), "W")

    logits = tf.matmul(X, W)
    cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits, y))
    optimizer = tf.train.GradientDescentOptimizer(config.LEARNING_RATE).minimize(cost)

    # Prediction
    prediction = tf.argmax(logits, 1)

    return dict(
    X=X, y=y, W=W,
    prediction=prediction,
    cost=cost, optimizer=optimizer)
```

placeholders: X and y are of size (?, 784) and (?, 10)

loss: softmax loss with cross entropy, can add regularization if needed

LOGISTIC REGRESSION - CODE

```
def train(config, g):

    with tf.Session() as sess:
        sess.run(tf.initialize_all_variables())
        trainX, trainY, testX, testY = load_data()

        for epoch_num in range(config.NUM_EPOCHS):
            prediction, training_loss, _ = sess.run([g['prediction'],
                                                     g['cost'],
                                                     g['optimizer']],
                                                    feed_dict={g['X']:trainX, g['y']:trainY})
            # Display
            if epoch_num%config.DISPLAY_STEP == 0:
                print "EPOCH %i: \n Training loss: %.3f, Test loss: %.3f" % (
                    epoch_num, training_loss, sess.run(g['cost'], feed_dict={g['X']:testX, g['y']:testY}))

if __name__== '__main__':
    config = parameters()
    g = model(config)
    train(config, g)
```

test: compute test loss