Code:

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

dependencies: implementation in tensorflow and numpy

```
class parameters():

    def __init__(self):
        self.DATA_LENGTH = 10000
        self.LEARNING_RATE = 1e-10
        self.REG = 1e-10
        self.NUM_EPOCHS = 500
        self.NUM_BATCHES = 10
        self.DISPLAY_STEP = 100 # epoch
```

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

LINEAR REGRESSION - CODE

```python
def generate_data(config):
    """
    Load the data.
    """
    X = np.array(range(config.DATA_LENGTH))
    y = 3.657*X + np.random.randn(*X.shape) * 0.33
    return X, y
```

data: X is sequential and y is a linear output with a big of noise added in.

```python
def generate_batches(config):
    """
    Create <num_batches> batches from X and y
    """
    X, y = generate_data(config)

    # Create batches
    batch_size = config.DATA_LENGTH // config.NUM_BATCHES
    data_X = np.zeros([config.NUM_BATCHES, batch_size], dtype=np.float32)
    data_y = np.zeros([config.NUM_BATCHES, batch_size], dtype=np.float32)
    for batch_num in range(config.NUM_BATCHES):
        data_X[batch_num,:] = X[batch_num*batch_size:(batch_num+1)*batch_size]
        data_y[batch_num,:] = y[batch_num*batch_size:(batch_num+1)*batch_size]
    yield data_X.reshape(-1, 1), data_y.reshape(-1, 1)
```

batches: Separating the data into batches, each of length batch_size and feeding in all batches simultaneously.

```python
def generate_epochs(config):
    """
    Create batches for <num_epochs> epochs.
    """
    for epoch_num in range(config.NUM_EPOCHS):
        yield generate_batches(config)
```

epochs: Feed in same data for num_epochs in order to continuously train and update the weights to minimize the cost function

LINEAR REGRESSION - CODE

placeholders: X and y are of size (?, 1)

weights: should be initialize to different random values so they don't all update same way

```
def model(config):
    """
    Train the linear model to minimize L2 loss function.
    """
    # Inputs
    X = tf.placeholder(tf.float32, [None, 1], "X")
    y = tf.placeholder(tf.float32, [None, 1], "y")

    # Set model weights
    with tf.variable_scope('weights'):
        W = tf.Variable(tf.truncated_normal([1,1], stddev=0.01), name="W", dtype=tf.float32)
        b = tf.Variable(tf.truncated_normal([1,1], stddev=0.01), name="b", dtype=tf.float32)

    # Forward pass
    prediction = tf.add(tf.matmul(X, W), b)

    # L2 loss
    batch_size = config.DATA_LENGTH // config.NUM_BATCHES
    cost = tf.reduce_sum(tf.pow(prediction-y, 2))/(2*batch_size) + config.REG * np.sum(W * W)

    # Gradient descent (backprop)
    optimizer = tf.train.GradientDescentOptimizer(config.LEARNING_RATE).minimize(cost)

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

prediction: get y_hat via forward pass

MSE + L2 Reg

optimizer: takes care of backprop and updates our weights.

LINEAR REGRESSION - CODE

```
def train(g):

    with tf.Session() as sess:
        sess.run(tf.initialize_all_variables())

        for epoch_num, epoch in enumerate(generate_epochs(config)):
            for simult_batch_num, (input_X, labels_y) in enumerate(epoch):
                prediction, training_loss, _ = sess.run([g['prediction'],
                                                g['cost'],
                                                g['optimizer']],
                                        feed_dict={g['X']:input_X, g['y']:labels_y})
            # Display
            if epoch_num%config.DISPLAY_STEP == 0:
                print "EPOCH %i: \n Training loss: %.3f, W: %.3f, b:%.3f" % (
                    epoch_num, training_loss, sess.run(g['W']), sess.run(g['b']))
```

train: run the optimizer to update the weights. Call for other variables for performance indication.

```
if __name__ == '__main__':
    import time
    start = time.time()
    config = parameters()
    g = model(config)
    train(g)
    print time.time() - start
```

execute: it is faster to use simultaneous batches than one batch at a time, depending on the size of the batch based on your CPU/ GPU constraints