```python
import numpy as np
import numpy.random as npr
import matplotlib.pyplot as plt
from funkyyak import grad
npr.seed(1)

class WeightsParser(object):
    def __init__(self):
        self.idxs_and_shapes = {}
        self.N = 0

    def add_weights(self, name, shape):
        start = self.N
        self.N += np.prod(shape)
        self.idxs_and_shapes[name] = (slice(start, self.N), shape)

    def get(self, vect, name):
        idxs, shape = self.idxs_and_shapes[name]
        return np.reshape(vect[idxs], shape)

def make_batches(N_total, N_batch):
    start = 0
    batches = []
    while start < N_total:
        batches.append(slice(start, start + N_batch))
        start += N_batch
    return batches

def logsumexp(X, axis):
    max_X = np.max(X)
    return max_X + np.log(np.sum(np.exp(X - max_X), axis=axis, keepdims=True))

def make_nn_funs(layer_sizes, L2_reg):
    parser = WeightsParser()
    for i, shape in enumerate(zip(layer_sizes[:-1], layer_sizes[1:])):
        parser.add_weights(('weights', i), shape)
        parser.add_weights(('biases', i), (1, shape[1]))

    def predictions(W_vect, X):
        cur_units = X
        for i in range(len(layer_sizes) - 1):
            cur_W = parser.get(W_vect, ('weights', i))
            cur_B = parser.get(W_vect, ('biases', i))
            cur_units = np.tanh(np.dot(cur_units, cur_W) + cur_B)
        return cur_units - logsumexp(cur_units, axis=1)

    def loss(W_vect, X, T):
        log_prior = -L2_reg * np.dot(W_vect, W_vect)
        log_lik = np.sum(predictions(W_vect, X) * T)
        return - log_prior - log_lik

    def frac_err(W_vect, X, T):
        return np.mean(np.argmax(T, axis=1) != np.argmax(pred_fun(W_vect, X), axis=1))

    return parser.N, predictions, loss, frac_err

```

```python
if __name__ == '__main__':
    # Network parameters
    layer_sizes = [784, 200, 100, 10]
    L2_reg = 1.0

    # Training parameters
    param_scale = 0.1
    learning_rate = 1e-3
    momentum = 0.9
    batch_size = 256
    num_epochs = 50

    # Load and process MNIST data (borrowing from Kayak)
    import imp, urllib
    partial_flatten = lambda x : np.reshape(x, (x.shape[0], np.prod(x.shape[1:])))
    one_hot = lambda x, K : np.array(x[:,None] == np.arange(K)[None, :], dtype=int)
    source, _ = urllib.urlretrieve(
        'https://raw.githubusercontent.com/HIPS/Kayak/master/examples/data.py')
    data = imp.load_source('data', source).mnist()
    train_images, train_labels, test_images, test_labels = data
    train_images = partial_flatten(train_images) / 255.0
    test_images  = partial_flatten(test_images)  / 255.0
    train_labels = one_hot(train_labels, 10)
    test_labels = one_hot(test_labels, 10)
    N_data = train_images.shape[0]

    # Make neural net functions
    N_weights, pred_fun, loss_fun, frac_err = make_nn_funs(layer_sizes, L2_reg)
    loss_grad = grad(loss_fun)

    # Initialize weights
    W = npr.randn(N_weights) * param_scale

    # Check grads
    rand_dir = npr.randn(N_weights) * param_scale
    rand_dir = rand_dir / np.sqrt(np.dot(rand_dir, rand_dir))
    test_fun = lambda x : loss_fun(W + x * rand_dir, train_images, train_labels)
    nd = (test_fun(1e-4) - test_fun(-1e-4)) / 2e-4
    ad = np.dot(loss_grad(W, train_images, train_labels), rand_dir)
    print "Checking grads. Relative diff is: {0}".format((nd - ad)/np.abs(nd))

    print "    Epoch      |    Train err  |   Test error  "
    def print_perf(epoch, W):
        test_perf  = frac_err(W, test_images, test_labels)
        train_perf = frac_err(W, train_images, train_labels)
        print "{0:15}|{1:15}|{2:15}".format(epoch, train_perf, test_perf)

    # Train with sgd
    batch_idxs = make_batches(N_data, batch_size)
    cur_dir = np.zeros(N_weights)
    for epoch in range(num_epochs):
        print_perf(epoch, W)
        for idxs in batch_idxs:
            grad_W = loss_grad(W, train_images[idxs], train_labels[idxs])
            cur_dir = momentum * cur_dir + (1.0 - momentum) * grad_W
            W -= learning_rate * cur_dir
```