Lab5

December 11, 2023

[ ]:
```
import numpy as np
```

0.0.1 Questions 1

A: Principle of backpropagation algorithm: Backpropagation is a process involved in training a neural network. It involves taking the error rate of a forward propagation and feeding this loss backward through the neural network layers to fine-tune the weights. #### B: The meaning and the role of the Softmax function: It converts the neural networks predictions into probability.

C: Name typically used non-linear output functions and implications of choosing one or another for implementation:

- **ReLu**: It is very efficient computationally
- **Sigmoid**: It is used for probabalistic predictions because of it's range (between 0 and 1)
- **SoftMax**: Similar to Sigmoid in that it gives a probability, but because of its summed nature it is usually used in the last layer since you usually cannot move forward with the result if it goes beyond 1
- **Hyperbolic Tanget**: Very good for mapping outputs to states between "negative", "neutral" or "positive" and is solid for hidden layers as an activation function. Since tanh is zero-centered (meaning its outputs are centered around 0), it can help in reducing the bias shift effect during training. If the activations are not zero-centered, the gradients can consistently be all positive or all negative in certain layers, which can lead to inefficient gradient descent

[ ]:
```
#functions of non-linear activations
def f_sigmoid(X, deriv=False):
    if not deriv:
        return 1 / (1 + np.exp(-X))
    else:
        return f_sigmoid(X)*(1 - f_sigmoid(X))


def f_softmax(X):
    Z = np.sum(np.exp(X), axis=1)
    Z = Z.reshape(Z.shape[0], 1)
    return np.exp(X) / Z

def f_relu(X, deriv=False):
    if not deriv:
```

```
        return np.maximum(0, X)
    else:
        return (X > 0).astype(float)
```

[ ]:
```
def exit_with_err(err_str):
    print >> sys.stderr, err_str
    sys.exit(1)
```

[ ]:
```
#Functionality of a single hidden layer
class Layer:
    def __init__(self, size, batch_size, is_input=False, is_output=False,
                 activation=f_sigmoid):
        self.is_input = is_input
        self.is_output = is_output

        # Z is the matrix that holds output values
        self.Z = np.zeros((batch_size, size[0]))
        # The activation function is an externally defined function (with a
        # derivative) that is stored here
        self.activation = activation

        # W is the outgoing weight matrix for this layer
        self.W = None
        # S is the matrix that holds the inputs to this layer
        self.S = None
        # D is the matrix that holds the deltas for this layer
        self.D = None
        # Fp is the matrix that holds the derivatives of the activation function
        self.Fp = None

        if not is_input:
            self.S = np.zeros((batch_size, size[0]))
            self.D = np.zeros((batch_size, size[0]))

        if not is_output:
            self.W = np.random.normal(size=size, scale=1E-4)

        if not is_input and not is_output:
            self.Fp = np.zeros((size[0], batch_size))

    def forward_propagate(self):
        if self.is_input:
            return self.Z.dot(self.W)

        self.Z = self.activation(self.S)
        if self.is_output:
            return self.Z
```

```
        else:
            # For hidden layers, we add the bias values here
            self.Z = np.append(self.Z, np.ones((self.Z.shape[0], 1)), axis=1)
            self.Fp = self.activation(self.S, deriv=True).T
            return self.Z.dot(self.W)
```

[ ]:
```
class MultiLayerPerceptron:
    def __init__(self, layer_config, batch_size=100):
        self.layers = []
        self.num_layers = len(layer_config)
        self.minibatch_size = batch_size

        for i in range(self.num_layers-1):
            if i == 0:
                print ("Initializing input layer with size {0}.".
 ↪format(layer_config[i]))
                # Here, we add an additional unit at the input for the bias
                # weight.
                self.layers.append(Layer([layer_config[i]+1, layer_config[i+1]],
                                         batch_size,
                                         is_input=True))
            else:
                print ("Initializing hidden layer with size {0}.".
 ↪format(layer_config[i]))
                # Here we add an additional unit in the hidden layers for the
                # bias weight.
                # self.layers.append(Layer([layer_config[i]+1,␣
 ↪layer_config[i+1]],
                #                          batch_size,
                #                          activation=f_sigmoid))

                self.layers.append(Layer([layer_config[i]+1, layer_config[i+1]],
                                         batch_size,
                                         activation=f_relu))

        print ("Initializing output layer with size {0}.".
 ↪format(layer_config[-1]))
        self.layers.append(Layer([layer_config[-1], None],
                                 batch_size,
                                 is_output=True,
                                 activation=f_softmax))
        print ("Done!")

    def forward_propagate(self, data):
        # We need to be sure to add bias values to the input
        self.layers[0].Z = np.append(data, np.ones((data.shape[0], 1)), axis=1)
```

```
        for i in range(self.num_layers-1):
            self.layers[i+1].S = self.layers[i].forward_propagate()
        return self.layers[-1].forward_propagate()

    def backpropagate(self, yhat, labels):

        # exit_with_err("FIND ME IN THE CODE, What is computed in the next line of code?\n")

        # It calculates the initial gradient of the loss function
        # with respect to the output of the last layer (the output predictions)
        # and stores it in the last layer's D property.

        self.layers[-1].D = (yhat - labels).T
        for i in range(self.num_layers-2, 0, -1):
            # We do not calculate deltas for the bias values
            W_nobias = self.layers[i].W[0:-1, :]

            # exit_with_err("FIND ME IN THE CODE, What does this 'for' loop do? \n")

            # It goes through the network of layers from the back and updates the deltas for the hidden layers.

            self.layers[i].D = W_nobias.dot(self.layers[i+1].D) * self.layers[i].Fp

    def update_weights(self, eta):
        for i in range(0, self.num_layers-1):
            W_grad = -eta*(self.layers[i+1].D.dot(self.layers[i].Z)).T
            self.layers[i].W += W_grad

    def evaluate(self, train_data, train_labels, test_data, test_labels,
                 num_epochs=70, eta=0.05, eval_train=False, eval_test=True):

        N_train = len(train_labels)*len(train_labels[0])
        N_test = len(test_labels)*len(test_labels[0])

        print ("Training for {0} epochs...".format(num_epochs))
        for t in range(0, num_epochs):
            out_str = "[{0:4d}] ".format(t)

            for b_data, b_labels in zip(train_data, train_labels):
                output = self.forward_propagate(b_data)
                self.backpropagate(output, b_labels)
```

```
                # exit_with_err("FIND ME IN THE CODE, How does weight update is␣
↪implemented? What is eta?\n")

                # ETA is the learning rate.
                # The update function is implemented by multiplying the␣
↪gradient by the learning rate and subtracting it from the weights.

                self.update_weights(eta=eta)

            if eval_train:
                errs = 0
                for b_data, b_labels in zip(train_data, train_labels):
                    output = self.forward_propagate(b_data)
                    yhat = np.argmax(output, axis=1)
                    errs += np.sum(1-b_labels[np.arange(len(b_labels)), yhat])

                out_str = ("{0} Training error: {1:.5f}".format(out_str,
                                                        float(errs)/N_train))

            if eval_test:
                errs = 0
                for b_data, b_labels in zip(test_data, test_labels):
                    output = self.forward_propagate(b_data)
                    yhat = np.argmax(output, axis=1)
                    errs += np.sum(1-b_labels[np.arange(len(b_labels)), yhat])

                out_str = ("{0} Test error: {1:.5f}").format(out_str,
                                                    float(errs)/N_test)

            print (out_str)
```

[ ]:
```
def label_to_bit_vector(labels, nbits):
    bit_vector = np.zeros((labels.shape[0], nbits))
    for i in range(labels.shape[0]):
        bit_vector[i, labels[i]] = 1.0

    return bit_vector
```

[ ]:
```
def create_batches(data, labels, batch_size, create_bit_vector=False):
    N = data.shape[0]
    print ("Batch size {0}, the number of examples {1}.".format(batch_size,N))

    if N % batch_size != 0:
        print ("Warning in create_minibatches(): Batch size {0} does not " \
              "evenly divide the number of examples {1}.".format(batch_size,N))
    chunked_data = []
    chunked_labels = []
```

```
    idx = 0
    while idx + batch_size <= N:
        chunked_data.append(data[idx:idx+batch_size, :])
        if not create_bit_vector:
            chunked_labels.append(labels[idx:idx+batch_size])
        else:
            bit_vector = label_to_bit_vector(labels[idx:idx+batch_size], 10)
            chunked_labels.append(bit_vector)

        idx += batch_size

    return chunked_data, chunked_labels
```

[ ]:
```
def prepare_for_backprop(batch_size, Train_images, Train_labels, Valid_images,␣
 ↪Valid_labels):

    print ("Creating data...")
    batched_train_data, batched_train_labels = create_batches(Train_images,␣
 ↪Train_labels,
                                                batch_size,
                                                create_bit_vector=True)
    batched_valid_data, batched_valid_labels = create_batches(Valid_images,␣
 ↪Valid_labels,
                                                batch_size,
                                                create_bit_vector=True)
    print ("Done!")


    return batched_train_data, batched_train_labels,  batched_valid_data,␣
 ↪batched_valid_labels

def get_accuracy(model, X, y):
    yhat = model.forward_propagate(X)
    yhat = np.argmax(yhat, axis=1)
    accuracy = np.sum(yhat == y) / float(len(y))
    print("Accuracy: {0:.4f}".format(accuracy))
    return accuracy
```

[ ]:
```
from keras.datasets import mnist
```

[ ]:
```
(Xtr, Ltr), (X_test, L_test)=mnist.load_data()

Xtr = Xtr.reshape(60000, 784)
X_test = X_test.reshape(10000, 784)
Xtr = Xtr.astype('float32')
X_test = X_test.astype('float32')
Xtr /= 255
```

```
X_test /= 255
print(Xtr.shape[0], 'train samples')
print(X_test.shape[0], 'test samples')
```

```
60000 train samples
10000 test samples
```

[ ]:
```
batch_size=100

train_data, train_labels, valid_data,␣
 ↪valid_labels=prepare_for_backprop(batch_size, Xtr, Ltr, X_test, L_test)

mlp = MultiLayerPerceptron(layer_config=[784, 100, 100, 10],␣
 ↪batch_size=batch_size)

mlp.evaluate(train_data, train_labels, valid_data, valid_labels,
             eval_train=True)

get_accuracy(mlp, X_test, L_test)

print("Done:)\n")
```

```
Creating data…
Batch size 100, the number of examples 60000.
Batch size 100, the number of examples 10000.
Done!
Initializing input layer with size 784.
Initializing hidden layer with size 100.
Initializing hidden layer with size 100.
Initializing output layer with size 10.
Done!
Training for 70 epochs…
[   0]  Training error: 0.46972 Test error: 0.47070
[   1]  Training error: 0.07760 Test error: 0.07590
[   2]  Training error: 0.04945 Test error: 0.05290
[   3]  Training error: 0.04372 Test error: 0.04740
[   4]  Training error: 0.03355 Test error: 0.03800
[   5]  Training error: 0.03038 Test error: 0.03970
[   6]  Training error: 0.02590 Test error: 0.03690
[   7]  Training error: 0.02168 Test error: 0.03310
[   8]  Training error: 0.02067 Test error: 0.03440
[   9]  Training error: 0.02007 Test error: 0.03360
[  10]  Training error: 0.02150 Test error: 0.03700
[  11]  Training error: 0.01632 Test error: 0.03440
[  12]  Training error: 0.01758 Test error: 0.03280
[  13]  Training error: 0.02112 Test error: 0.03510
[  14]  Training error: 0.01668 Test error: 0.03280
```

```
[  15]  Training error: 0.01080 Test error: 0.03160
[  16]  Training error: 0.01162 Test error: 0.03210
[  17]  Training error: 0.01190 Test error: 0.03230
[  18]  Training error: 0.01195 Test error: 0.03170
[  19]  Training error: 0.01633 Test error: 0.03440
[  20]  Training error: 0.00995 Test error: 0.03040
[  21]  Training error: 0.00918 Test error: 0.03190
[  22]  Training error: 0.01158 Test error: 0.03160
[  23]  Training error: 0.01117 Test error: 0.03230
[  24]  Training error: 0.00945 Test error: 0.03130
[  25]  Training error: 0.00598 Test error: 0.02940
[  26]  Training error: 0.00977 Test error: 0.03100
[  27]  Training error: 0.00615 Test error: 0.02880
[  28]  Training error: 0.00538 Test error: 0.02850
[  29]  Training error: 0.00877 Test error: 0.03090
[  30]  Training error: 0.00315 Test error: 0.02880
[  31]  Training error: 0.00598 Test error: 0.03130
[  32]  Training error: 0.00407 Test error: 0.02820
[  33]  Training error: 0.00422 Test error: 0.02910
[  34]  Training error: 0.00788 Test error: 0.02870
[  35]  Training error: 0.00722 Test error: 0.03100
[  36]  Training error: 0.00388 Test error: 0.02710
[  37]  Training error: 0.00672 Test error: 0.02970
[  38]  Training error: 0.00623 Test error: 0.03030
[  39]  Training error: 0.01087 Test error: 0.03440
[  40]  Training error: 0.00490 Test error: 0.02880
[  41]  Training error: 0.00515 Test error: 0.03040
[  42]  Training error: 0.00547 Test error: 0.02980
[  43]  Training error: 0.00242 Test error: 0.02730
[  44]  Training error: 0.00215 Test error: 0.02850
[  45]  Training error: 0.00113 Test error: 0.02670
[  46]  Training error: 0.00093 Test error: 0.02650
[  47]  Training error: 0.00033 Test error: 0.02540
[  48]  Training error: 0.00018 Test error: 0.02600
[  49]  Training error: 0.00007 Test error: 0.02490
[  50]  Training error: 0.00005 Test error: 0.02530
[  51]  Training error: 0.00005 Test error: 0.02520
[  52]  Training error: 0.00005 Test error: 0.02560
[  53]  Training error: 0.00005 Test error: 0.02560
[  54]  Training error: 0.00003 Test error: 0.02540
[  55]  Training error: 0.00002 Test error: 0.02510
[  56]  Training error: 0.00002 Test error: 0.02520
[  57]  Training error: 0.00002 Test error: 0.02490
[  58]  Training error: 0.00002 Test error: 0.02490
[  59]  Training error: 0.00002 Test error: 0.02490
[  60]  Training error: 0.00000 Test error: 0.02490
[  61]  Training error: 0.00000 Test error: 0.02480
[  62]  Training error: 0.00000 Test error: 0.02480
```

```
[  63]  Training error: 0.00000 Test error: 0.02480
[  64]  Training error: 0.00000 Test error: 0.02500
[  65]  Training error: 0.00000 Test error: 0.02490
[  66]  Training error: 0.00000 Test error: 0.02510
[  67]  Training error: 0.00000 Test error: 0.02490
[  68]  Training error: 0.00000 Test error: 0.02490
[  69]  Training error: 0.00000 Test error: 0.02480
Accuracy: 0.9752
Done:)
```

[ ]:
```
batch_size=100

train_data, train_labels, valid_data,␣
 ↪valid_labels=prepare_for_backprop(batch_size, Xtr, Ltr, X_test, L_test)

mlp = MultiLayerPerceptron(layer_config=[784, 100, 100, 10],␣
 ↪batch_size=batch_size)

mlp.evaluate(train_data, train_labels, valid_data, valid_labels,
             eval_train=True, eta=0.5)

get_accuracy(mlp, X_test, L_test)

print("Done:)\n")
```

```
Creating data…
Batch size 100, the number of examples 60000.
Batch size 100, the number of examples 10000.
Done!
Initializing input layer with size 784.
Initializing hidden layer with size 100.
Initializing hidden layer with size 100.
Initializing output layer with size 10.
Done!
Training for 70 epochs…
[   0]  Training error: 0.89782 Test error: 0.89900
[   1]  Training error: 0.90085 Test error: 0.89910
[   2]  Training error: 0.90128 Test error: 0.90200
[   3]  Training error: 0.90085 Test error: 0.89910
[   4]  Training error: 0.90137 Test error: 0.90420
[   5]  Training error: 0.90128 Test error: 0.90200
[   6]  Training error: 0.88763 Test error: 0.88650
[   7]  Training error: 0.90137 Test error: 0.90420
[   8]  Training error: 0.89782 Test error: 0.89900
[   9]  Training error: 0.90263 Test error: 0.90180
[  10]  Training error: 0.90085 Test error: 0.89910
[  11]  Training error: 0.89782 Test error: 0.89900
```

```
[  12]  Training error: 0.89782 Test error: 0.89900
[  13]  Training error: 0.90137 Test error: 0.90420
[  14]  Training error: 0.90128 Test error: 0.90200
[  15]  Training error: 0.90128 Test error: 0.90200
[  16]  Training error: 0.90263 Test error: 0.90180
[  17]  Training error: 0.90965 Test error: 0.91080
[  18]  Training error: 0.90263 Test error: 0.90180
[  19]  Training error: 0.89558 Test error: 0.89720
[  20]  Training error: 0.90128 Test error: 0.90200
[  21]  Training error: 0.90248 Test error: 0.90260
[  22]  Training error: 0.90263 Test error: 0.90180
[  23]  Training error: 0.89782 Test error: 0.89900
[  24]  Training error: 0.90128 Test error: 0.90200
[  25]  Training error: 0.90248 Test error: 0.90260
[  26]  Training error: 0.90248 Test error: 0.90260
[  27]  Training error: 0.90965 Test error: 0.91080
[  28]  Training error: 0.90085 Test error: 0.89910
[  29]  Training error: 0.90965 Test error: 0.91080
[  30]  Training error: 0.89558 Test error: 0.89720
[  31]  Training error: 0.90137 Test error: 0.90420
[  32]  Training error: 0.90137 Test error: 0.90420
[  33]  Training error: 0.90248 Test error: 0.90260
[  34]  Training error: 0.90248 Test error: 0.90260
[  35]  Training error: 0.90248 Test error: 0.90260
[  36]  Training error: 0.90128 Test error: 0.90200
[  37]  Training error: 0.90248 Test error: 0.90260
[  38]  Training error: 0.90263 Test error: 0.90180
[  39]  Training error: 0.89782 Test error: 0.89900
[  40]  Training error: 0.90263 Test error: 0.90180
[  41]  Training error: 0.89782 Test error: 0.89900
[  42]  Training error: 0.90085 Test error: 0.89910
[  43]  Training error: 0.90085 Test error: 0.89910
[  44]  Training error: 0.90128 Test error: 0.90200
[  45]  Training error: 0.88763 Test error: 0.88650
[  46]  Training error: 0.90263 Test error: 0.90180
[  47]  Training error: 0.90248 Test error: 0.90260
[  48]  Training error: 0.90248 Test error: 0.90260
[  49]  Training error: 0.90248 Test error: 0.90260
[  50]  Training error: 0.90137 Test error: 0.90420
[  51]  Training error: 0.90070 Test error: 0.89680
[  52]  Training error: 0.90248 Test error: 0.90260
[  53]  Training error: 0.90248 Test error: 0.90260
[  54]  Training error: 0.88763 Test error: 0.88650
[  55]  Training error: 0.88763 Test error: 0.88650
[  56]  Training error: 0.89558 Test error: 0.89720
[  57]  Training error: 0.90248 Test error: 0.90260
[  58]  Training error: 0.90965 Test error: 0.91080
[  59]  Training error: 0.90263 Test error: 0.90180
```

```
[  60]  Training error: 0.89782 Test error: 0.89900
[  61]  Training error: 0.88763 Test error: 0.88650
[  62]  Training error: 0.90263 Test error: 0.90180
[  63]  Training error: 0.90085 Test error: 0.89910
[  64]  Training error: 0.90137 Test error: 0.90420
[  65]  Training error: 0.90248 Test error: 0.90260
[  66]  Training error: 0.90085 Test error: 0.89910
[  67]  Training error: 0.90137 Test error: 0.90420
[  68]  Training error: 0.90137 Test error: 0.90420
[  69]  Training error: 0.90248 Test error: 0.90260
Accuracy: 0.0974
Done:)
```

[ ]:
```
batch_size=100

train_data, train_labels, valid_data,␣
 ↪valid_labels=prepare_for_backprop(batch_size, Xtr, Ltr, X_test, L_test)

mlp = MultiLayerPerceptron(layer_config=[784, 100, 100, 10],␣
 ↪batch_size=batch_size)

mlp.evaluate(train_data, train_labels, valid_data, valid_labels,
             eval_train=True, eta=0.005)

get_accuracy(mlp, X_test, L_test)

print("Done:)\n")
```

```
Creating data…
Batch size 100, the number of examples 60000.
Batch size 100, the number of examples 10000.
Done!
Initializing input layer with size 784.
Initializing hidden layer with size 100.
Initializing hidden layer with size 100.
Initializing output layer with size 10.
Done!
Training for 70 epochs…
[   0]  Training error: 0.70343 Test error: 0.70080
[   1]  Training error: 0.64710 Test error: 0.64380
[   2]  Training error: 0.59973 Test error: 0.59910
[   3]  Training error: 0.45603 Test error: 0.46660
[   4]  Training error: 0.20322 Test error: 0.19170
[   5]  Training error: 0.11390 Test error: 0.11040
[   6]  Training error: 0.09023 Test error: 0.08900
[   7]  Training error: 0.07532 Test error: 0.07420
[   8]  Training error: 0.06420 Test error: 0.06520
```

```
[   9]  Training error: 0.05513 Test error: 0.05750
[  10]  Training error: 0.04828 Test error: 0.05070
[  11]  Training error: 0.04290 Test error: 0.04760
[  12]  Training error: 0.03870 Test error: 0.04340
[  13]  Training error: 0.03508 Test error: 0.04050
[  14]  Training error: 0.03197 Test error: 0.03690
[  15]  Training error: 0.02883 Test error: 0.03520
[  16]  Training error: 0.02600 Test error: 0.03400
[  17]  Training error: 0.02405 Test error: 0.03220
[  18]  Training error: 0.02220 Test error: 0.03120
[  19]  Training error: 0.02065 Test error: 0.03080
[  20]  Training error: 0.01920 Test error: 0.03020
[  21]  Training error: 0.01782 Test error: 0.02920
[  22]  Training error: 0.01678 Test error: 0.02790
[  23]  Training error: 0.01555 Test error: 0.02800
[  24]  Training error: 0.01435 Test error: 0.02820
[  25]  Training error: 0.01355 Test error: 0.02790
[  26]  Training error: 0.01268 Test error: 0.02770
[  27]  Training error: 0.01195 Test error: 0.02730
[  28]  Training error: 0.01125 Test error: 0.02680
[  29]  Training error: 0.01063 Test error: 0.02680
[  30]  Training error: 0.00990 Test error: 0.02710
[  31]  Training error: 0.00955 Test error: 0.02670
[  32]  Training error: 0.00915 Test error: 0.02700
[  33]  Training error: 0.00860 Test error: 0.02730
[  34]  Training error: 0.00822 Test error: 0.02740
[  35]  Training error: 0.00793 Test error: 0.02750
[  36]  Training error: 0.00740 Test error: 0.02760
[  37]  Training error: 0.00687 Test error: 0.02760
[  38]  Training error: 0.00653 Test error: 0.02790
[  39]  Training error: 0.00612 Test error: 0.02780
[  40]  Training error: 0.00570 Test error: 0.02780
[  41]  Training error: 0.00530 Test error: 0.02810
[  42]  Training error: 0.00483 Test error: 0.02830
[  43]  Training error: 0.00447 Test error: 0.02830
[  44]  Training error: 0.00388 Test error: 0.02780
[  45]  Training error: 0.00368 Test error: 0.02780
[  46]  Training error: 0.00340 Test error: 0.02800
[  47]  Training error: 0.00310 Test error: 0.02760
[  48]  Training error: 0.00285 Test error: 0.02760
[  49]  Training error: 0.00255 Test error: 0.02710
[  50]  Training error: 0.00228 Test error: 0.02740
[  51]  Training error: 0.00208 Test error: 0.02720
[  52]  Training error: 0.00185 Test error: 0.02740
[  53]  Training error: 0.00168 Test error: 0.02740
[  54]  Training error: 0.00148 Test error: 0.02780
[  55]  Training error: 0.00132 Test error: 0.02770
[  56]  Training error: 0.00127 Test error: 0.02740
```

```
[  57]  Training error: 0.00113 Test error: 0.02760
[  58]  Training error: 0.00108 Test error: 0.02750
[  59]  Training error: 0.00100 Test error: 0.02700
[  60]  Training error: 0.00093 Test error: 0.02700
[  61]  Training error: 0.00083 Test error: 0.02710
[  62]  Training error: 0.00077 Test error: 0.02720
[  63]  Training error: 0.00073 Test error: 0.02730
[  64]  Training error: 0.00060 Test error: 0.02740
[  65]  Training error: 0.00055 Test error: 0.02730
[  66]  Training error: 0.00052 Test error: 0.02730
[  67]  Training error: 0.00045 Test error: 0.02720
[  68]  Training error: 0.00042 Test error: 0.02720
[  69]  Training error: 0.00040 Test error: 0.02710
Accuracy: 0.9729
Done:)
```

[ ]:
```
batch_size=100

print("With ReLu\n")

train_data, train_labels, valid_data,␣
 ↪valid_labels=prepare_for_backprop(batch_size, Xtr, Ltr, X_test, L_test)

mlp = MultiLayerPerceptron(layer_config=[784, 100, 100, 10],␣
 ↪batch_size=batch_size)

mlp.evaluate(train_data, train_labels, valid_data, valid_labels,
             eval_train=True)

get_accuracy(mlp, X_test, L_test)

print("Done:)\n")
```

```
With ReLu

Creating data…
Batch size 100, the number of examples 60000.
Batch size 100, the number of examples 10000.
Done!
Initializing input layer with size 784.
Initializing hidden layer with size 100.
Initializing hidden layer with size 100.
Initializing output layer with size 10.
Done!
Training for 70 epochs…
[   0]  Training error: 0.90137 Test error: 0.90420
[   1]  Training error: 0.90137 Test error: 0.90420
```

```
[   2]  Training error: 0.90137 Test error: 0.90420
[   3]  Training error: 0.90137 Test error: 0.90420
[   4]  Training error: 0.90137 Test error: 0.90420
[   5]  Training error: 0.90137 Test error: 0.90420
[   6]  Training error: 0.90137 Test error: 0.90420
[   7]  Training error: 0.90137 Test error: 0.90420
[   8]  Training error: 0.90137 Test error: 0.90420
[   9]  Training error: 0.90137 Test error: 0.90420
[  10]  Training error: 0.90137 Test error: 0.90420
[  11]  Training error: 0.90137 Test error: 0.90420
[  12]  Training error: 0.90137 Test error: 0.90420
[  13]  Training error: 0.90137 Test error: 0.90420
[  14]  Training error: 0.90137 Test error: 0.90420
[  15]  Training error: 0.90137 Test error: 0.90420
[  16]  Training error: 0.90137 Test error: 0.90420
[  17]  Training error: 0.90137 Test error: 0.90420
[  18]  Training error: 0.90137 Test error: 0.90420
[  19]  Training error: 0.90137 Test error: 0.90420
[  20]  Training error: 0.90137 Test error: 0.90420
[  21]  Training error: 0.90137 Test error: 0.90420
[  22]  Training error: 0.90137 Test error: 0.90420
[  23]  Training error: 0.90137 Test error: 0.90420
[  24]  Training error: 0.90137 Test error: 0.90420
[  25]  Training error: 0.90137 Test error: 0.90420
[  26]  Training error: 0.90137 Test error: 0.90420
[  27]  Training error: 0.90137 Test error: 0.90420
[  28]  Training error: 0.90137 Test error: 0.90420
[  29]  Training error: 0.90137 Test error: 0.90420
[  30]  Training error: 0.90137 Test error: 0.90420
[  31]  Training error: 0.90137 Test error: 0.90420
[  32]  Training error: 0.90137 Test error: 0.90420
[  33]  Training error: 0.90137 Test error: 0.90420
[  34]  Training error: 0.90137 Test error: 0.90420
[  35]  Training error: 0.90137 Test error: 0.90420
[  36]  Training error: 0.90137 Test error: 0.90420
[  37]  Training error: 0.90137 Test error: 0.90420
[  38]  Training error: 0.90137 Test error: 0.90420
[  39]  Training error: 0.90137 Test error: 0.90420
[  40]  Training error: 0.90137 Test error: 0.90420
[  41]  Training error: 0.90137 Test error: 0.90420
[  42]  Training error: 0.90137 Test error: 0.90420
[  43]  Training error: 0.90137 Test error: 0.90420
[  44]  Training error: 0.90137 Test error: 0.90420
[  45]  Training error: 0.90137 Test error: 0.90420
[  46]  Training error: 0.90137 Test error: 0.90420
[  47]  Training error: 0.90137 Test error: 0.90420
[  48]  Training error: 0.90137 Test error: 0.90420
[  49]  Training error: 0.90137 Test error: 0.90420
```

```
[  50]  Training error: 0.90137 Test error: 0.90420
[  51]  Training error: 0.90137 Test error: 0.90420
[  52]  Training error: 0.90137 Test error: 0.90420
[  53]  Training error: 0.90137 Test error: 0.90420
[  54]  Training error: 0.90137 Test error: 0.90420
[  55]  Training error: 0.90137 Test error: 0.90420
[  56]  Training error: 0.90137 Test error: 0.90420
[  57]  Training error: 0.90137 Test error: 0.90420
[  58]  Training error: 0.90137 Test error: 0.90420
[  59]  Training error: 0.90137 Test error: 0.90420
[  60]  Training error: 0.90137 Test error: 0.90420
[  61]  Training error: 0.90137 Test error: 0.90420
[  62]  Training error: 0.90137 Test error: 0.90420
[  63]  Training error: 0.90137 Test error: 0.90420
[  64]  Training error: 0.90137 Test error: 0.90420
[  65]  Training error: 0.90137 Test error: 0.90420
[  66]  Training error: 0.90137 Test error: 0.90420
[  67]  Training error: 0.90137 Test error: 0.90420
[  68]  Training error: 0.90137 Test error: 0.90420
[  69]  Training error: 0.90137 Test error: 0.90420
Accuracy: 0.0958
Done:)
```