Homework 5 - Berkeley STAT 157

Your name: Zhiming, SID 3034485754 (Please add your name, and SID to ease Ryan and Rachel to grade.)

Please submit your homework through [gradescope (http://gradescope.com/)](http://gradescope.com/) instead of Github, so you will get the score distribution for each question. Please enroll in the [class (https://www.gradescope.com/courses/42432)](https://www.gradescope.com/courses/42432) by the Entry code: MXG5G5

Handout 2/19/2019, due 2/26/2019 by 4pm in Git by committing to your repository.

In this homework, we will model covariate shift and attempt to fix it using logistic regression. This is a fairly realistic scenario for data scientists. To keep things well under control and understandable we will use [Fashion-MNIST (http://d2l.ai/chapter_linear-networks/fashion-mnist.html)](http://d2l.ai/chapter_linear-networks/fashion-mnist.html) as the data to experiment on.

Follow the instructions from the Fashion MNIST notebook to get the data.

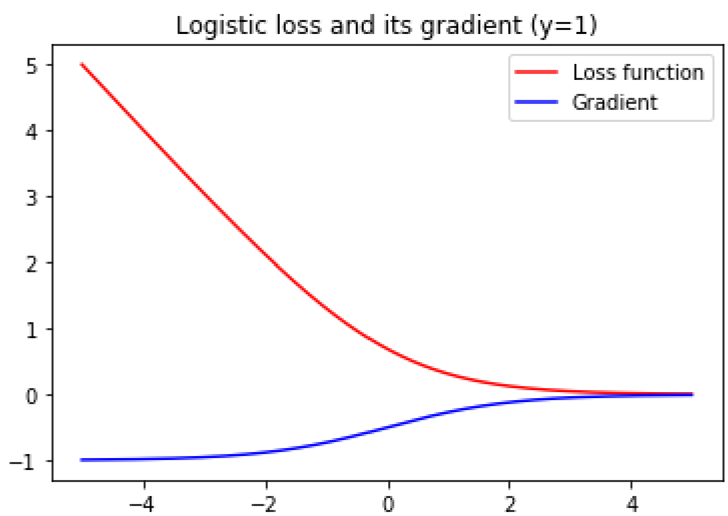
In [1]:
```
%matplotlib inline
from mxnet import autograd, gluon, init, nd
from mxnet.gluon import data as gdata, loss as gloss, nn, utils
import d2l
import numpy as np
import matplotlib.pyplot as plt
import mxnet as mx
import time
```

1. Logistic Regression

1. Implement the logistic loss function $l(y, f) = -\log(1 + \exp(-yf))$ in Gluon.
2. Plot its values and its derivative for $y = 1$ and $f \in [-5, 5]$, using automatic differentiation in Gluon.
3. Generate training and test datasets for a binary classification problem using Fashion-MNIST with class 1 being a combination of `shirt` and `sweater` and class -1 being the combination of `sandal` and `sneaker` categories.
4. Train a binary classifier of your choice (it can be linear or a simple MLP such as from a previous lecture) using half the data (i.e. $12,000$ observations mixed as abvove) and one using the full dataset (i.e. $24,000$ observations as arising from the 4 categories) and report its accuracy.

Hint - you should encapsulate the training and reporting code in a callable function since you'll need it quite a bit in the following.

In [2]:
```
# for question 1 & 2
def logistic_loss(f, y):
    l = nd.log(1.0+nd.exp(-f*y))
    return l
f = nd.arange(-5, 5, 0.01)
f.attach_grad()
# for y = 1
y = nd.ones(shape = f.shape)
with autograd.record():
    l = logistic_loss(f, y)
l.backward()
# for loss function
plt.figure()
plt.title('Logistic loss and its gradient (y=1)')
plt.plot(f.asnumpy(), l.asnumpy(), color = 'r',\
         label = 'Loss function')
# for grad
plt.plot(f.asnumpy(), f.grad.asnumpy(), color = 'b',\
        label = 'Gradient')
plt.legend(loc = 'upper right')
plt.show()
```



In [3]:
```
mnist_train = gdata.vision.FashionMNIST(train=True, transform = lambda d
                                         (data.astype(np.float32), label)
mnist_test = gdata.vision.FashionMNIST(train=False, transform = lambda d
                                         (data.astype(np.float32), label)
```

In [4]:
```
# for question 3
#X, y = train[0:9]
# pick out pullover/shirt, and sneaker/scandal
# a new preprocess function, can produce biased dataset
def preprocess(mnist_train, mnist_test, total_per_label, ratio): # ratio
    X, y = mnist_train[:]
    # pick up the indices
    index_sweater = np.where(y==3)[0]
    index_shirt = np.where(y==6)[0]
    index_scandal = np.where(y==5)[0]
    index_sneaker = np.where(y==7)[0]
    # create the class for training, biased
    class_sweater = X[index_sweater[0:round(total_per_label*ratio)]]
    class_sneaker = X[index_sneaker[0:round(total_per_label*ratio)]]
    class_scandal = X[index_scandal[0:round(total_per_label*(1-ratio))]]
    class_shirt = X[index_shirt[0:round(total_per_label*(1-ratio))]]
    # print(class_sweater.shape, class_shirt.shape, class_scandal.shape,
    train_feature = nd.concat(class_sweater, class_shirt, class_scandal,
    label1 = nd.ones((1, round(total_per_label*2*ratio))).astype(np.floa
    label2 = nd.zeros((1, round(total_per_label*2*(1-ratio)))).astype(np
    train_labels = nd.concat(label1, label2, dim=1).reshape(shape=(-1,))
    train_data = gdata.dataset.ArrayDataset(train_feature, train_labels)
    # create the class for testing, unbiased
    X, y = mnist_test[:]
    index1 = np.where(np.logical_or(y==3, y==6))[0]
    index2 = np.where(np.logical_or(y==5, y==7))[0]
    class1 = X[index1]
    class2 = X[index2]
    test_feature = nd.concat(class1, class2, dim=0)

    label1 = nd.ones((1, 2000)).astype(np.float32)
    label2 = nd.zeros((1, 2000)).astype(np.float32)
    test_labels = nd.concat(label1, label2, dim=1).reshape(shape=(-1,))
    test_data = gdata.dataset.ArrayDataset(test_feature, test_labels)

    return train_data, test_data
```

In [5]:
```
def train_and_test_mnist(train_data, test_data, batch_size, lr, num_epoc
    net = nn.Sequential()
    net.add(nn.Dense(2))
    net.initialize(init.Normal(sigma=0.01))
    loss = gluon.loss.SoftmaxCrossEntropyLoss()
    #loss = logistic_loss
    trainer = gluon.Trainer(net.collect_params(), 'sgd', {'learning_rate
    d2l.train_ch3(net, train_data, test_data, loss, num_epochs, batch_si
```

In [6]:
```
# half the dataset
# note: use half the dataset, test acc is almost the same as using the f
# so I used just 50 per label for train (total 100 for training), which
# observable difference in test acc (~.975 v.s. ~.999)
train_data, test_data = preprocess(mnist_train, mnist_test, total_per_la
batch_size = 64
train_data = gluon.data.DataLoader(train_data, batch_size=batch_size, sh
test_data = gluon.data.DataLoader(test_data, batch_size=batch_size, shuf
train_and_test_mnist(train_data, test_data, batch_size=batch_size, lr=0.
# full dataset
train_data, test_data = preprocess(mnist_train, mnist_test, total_per_la
batch_size = 64
train_data = gluon.data.DataLoader(train_data, batch_size=batch_size, sh
test_data = gluon.data.DataLoader(test_data, batch_size=batch_size, shuf
train_and_test_mnist(train_data, test_data, batch_size=batch_size, lr=0.
```

```
epoch 1, loss 26718.0885, train acc 0.660, test acc 0.526
epoch 2, loss 11235.9929, train acc 0.670, test acc 0.975
epoch 3, loss 11.1763, train acc 0.990, test acc 0.980
epoch 4, loss 0.0000, train acc 1.000, test acc 0.980
epoch 5, loss 0.0000, train acc 1.000, test acc 0.980
epoch 1, loss 435.3934, train acc 0.992, test acc 0.997
epoch 2, loss 44.4580, train acc 0.998, test acc 0.999
epoch 3, loss 26.6787, train acc 0.999, test acc 0.999
epoch 4, loss 17.2049, train acc 0.999, test acc 0.999
epoch 5, loss 15.4026, train acc 0.999, test acc 0.999
```

2. Covariate Shift

Your goal is to introduce covariate shit in the data and observe the accuracy. For this, compose a dataset of $12,000$ observations, given by a mixture of `shirt` and `sweater` and of `sandal` and `sneaker` respectively, where you use a fraction $\lambda \in \{0.05, 0.1, 0.2, \ldots 0.8, 0.9, 0.95\}$ of one and a fraction of $1-\lambda$ of the other datasets respectively. For instance, you might pick for $\lambda = 0.1$ a total of 600 `shirt` and 600 `sweater` images and likewise $5,400$ `sandal` and $5,400$ `sneaker` photos, yielding a total of $12,000$ images for training. Note that the test set remains unbiased, composed of $2,000$ photos for the `shirt + sweater` category and of the `sandal + sneaker` category each.

1. Generate training sets that are appropriately biased. You should have 11 datasets.
2. Train a binary classifier using this and report the test set accuracy on the unbiased test set.

In [7]:
```
total_per_label = 6000
num_epochs = 8
print("Bias ratio now is:", .05)
train_data, test_data = preprocess(mnist_train, mnist_test, total_per_la
batch_size = 64
train_data = gluon.data.DataLoader(train_data, batch_size=batch_size, sh
test_data = gluon.data.DataLoader(test_data, batch_size=batch_size, shuf
train_and_test_mnist(train_data, test_data, batch_size=batch_size, lr=0.
for r in range(1, 9, 1):
    ratio = r / 10.0
    print("Bias ratio now is:", ratio, "(unbiased)" if ratio==.5 else ""
    train_data, test_data = preprocess(mnist_train, mnist_test, total_pe
    batch_size = 64
    train_data = gluon.data.DataLoader(train_data, batch_size=batch_size
    test_data = gluon.data.DataLoader(test_data, batch_size=batch_size,
    train_and_test_mnist(train_data, test_data, batch_size=batch_size, l
print("Bias ratio now is:", .95)
train_data, test_data = preprocess(mnist_train, mnist_test, total_per_la
batch_size = 64
train_data = gluon.data.DataLoader(train_data, batch_size=batch_size, sh
test_data = gluon.data.DataLoader(test_data, batch_size=batch_size, shuf
train_and_test_mnist(train_data, test_data, batch_size=batch_size, lr=0.
```

```
Bias ratio now is: 0.05
epoch 1, loss 1000.1986, train acc 0.924, test acc 0.724
epoch 2, loss 724.9604, train acc 0.934, test acc 0.714
epoch 3, loss 743.7490, train acc 0.935, test acc 0.695
epoch 4, loss 698.5812, train acc 0.934, test acc 0.505
epoch 5, loss 630.6260, train acc 0.940, test acc 0.697
epoch 6, loss 655.4589, train acc 0.938, test acc 0.718
epoch 7, loss 689.7992, train acc 0.935, test acc 0.742
epoch 8, loss 739.3488, train acc 0.934, test acc 0.672
Bias ratio now is: 0.1
epoch 1, loss 1718.0855, train acc 0.872, test acc 0.655
epoch 2, loss 1478.2894, train acc 0.882, test acc 0.599
epoch 3, loss 1485.8477, train acc 0.887, test acc 0.799
epoch 4, loss 1429.8997, train acc 0.888, test acc 0.642
epoch 5, loss 1307.3289, train acc 0.888, test acc 0.538
epoch 6, loss 1196.0399, train acc 0.890, test acc 0.933
epoch 7, loss 1389.5948, train acc 0.889, test acc 0.757
epoch 8, loss 1321.6712, train acc 0.890, test acc 0.578
Bias ratio now is: 0.2
epoch 1, loss 2631.4459, train acc 0.804, test acc 0.715
epoch 2, loss 2487.7274, train acc 0.821, test acc 0.772
epoch 3, loss 2439.9909, train acc 0.815, test acc 0.706
epoch 4, loss 2325.9607, train acc 0.824, test acc 0.870
epoch 5, loss 2220.9590, train acc 0.824, test acc 0.723
epoch 6, loss 2325.0559, train acc 0.824, test acc 0.741
epoch 7, loss 2409.2014, train acc 0.821, test acc 0.724
epoch 8, loss 2370.7292, train acc 0.825, test acc 0.695
Bias ratio now is: 0.3
epoch 1, loss 3033.8086, train acc 0.786, test acc 0.804
epoch 2, loss 2806.9189, train acc 0.800, test acc 0.823
epoch 3, loss 2711.9641, train acc 0.808, test acc 0.962
epoch 4, loss 2621.7852, train acc 0.809, test acc 0.994
epoch 5, loss 2775.3984, train acc 0.806, test acc 0.707
```

```
epoch 6, loss 2596.9733, train acc 0.812, test acc 0.758
epoch 7, loss 2704.1785, train acc 0.809, test acc 0.836
epoch 8, loss 2663.2411, train acc 0.809, test acc 0.962
Bias ratio now is: 0.4
epoch 1, loss 1885.2455, train acc 0.840, test acc 0.971
epoch 2, loss 1816.4795, train acc 0.853, test acc 0.965
epoch 3, loss 1656.1928, train acc 0.855, test acc 0.910
epoch 4, loss 1605.6959, train acc 0.859, test acc 0.788
epoch 5, loss 1773.3373, train acc 0.853, test acc 0.716
epoch 6, loss 1714.5210, train acc 0.857, test acc 0.915
epoch 7, loss 1636.6776, train acc 0.862, test acc 0.847
epoch 8, loss 1834.5523, train acc 0.851, test acc 0.991
Bias ratio now is: 0.5 (unbiased)
epoch 1, loss 46.6982, train acc 0.991, test acc 0.998
epoch 2, loss 1.8334, train acc 0.999, test acc 0.999
epoch 3, loss 1.1898, train acc 0.999, test acc 0.999
epoch 4, loss 0.4084, train acc 1.000, test acc 0.999
epoch 5, loss 0.2590, train acc 1.000, test acc 0.999
epoch 6, loss 0.1962, train acc 0.999, test acc 0.999
epoch 7, loss 0.1693, train acc 1.000, test acc 0.999
epoch 8, loss 0.0618, train acc 1.000, test acc 0.999
Bias ratio now is: 0.6
epoch 1, loss 354.8648, train acc 0.878, test acc 0.982
epoch 2, loss 270.1231, train acc 0.889, test acc 0.861
epoch 3, loss 237.8066, train acc 0.895, test acc 0.897
epoch 4, loss 267.0840, train acc 0.891, test acc 0.938
epoch 5, loss 252.3673, train acc 0.895, test acc 0.943
epoch 6, loss 262.2170, train acc 0.893, test acc 0.900
epoch 7, loss 228.3260, train acc 0.897, test acc 0.840
epoch 8, loss 297.1961, train acc 0.894, test acc 0.864
Bias ratio now is: 0.7
epoch 1, loss 787.2096, train acc 0.852, test acc 0.766
epoch 2, loss 596.7233, train acc 0.871, test acc 0.785
epoch 3, loss 596.5825, train acc 0.878, test acc 0.771
epoch 4, loss 528.9966, train acc 0.883, test acc 0.773
epoch 5, loss 545.9571, train acc 0.883, test acc 0.647
epoch 6, loss 488.5068, train acc 0.887, test acc 0.756
epoch 7, loss 553.4973, train acc 0.881, test acc 0.760
epoch 8, loss 502.5286, train acc 0.887, test acc 0.860
Bias ratio now is: 0.8
epoch 1, loss 1869.1117, train acc 0.769, test acc 0.765
epoch 2, loss 1853.5029, train acc 0.778, test acc 0.503
epoch 3, loss 1641.7631, train acc 0.790, test acc 0.603
epoch 4, loss 1715.6684, train acc 0.786, test acc 0.554
epoch 5, loss 1706.5300, train acc 0.787, test acc 0.652
epoch 6, loss 1708.6808, train acc 0.789, test acc 0.753
epoch 7, loss 1668.6962, train acc 0.789, test acc 0.764
epoch 8, loss 1601.7086, train acc 0.795, test acc 0.524
Bias ratio now is: 0.95
epoch 1, loss 834.2551, train acc 0.907, test acc 0.500
epoch 2, loss 721.4573, train acc 0.913, test acc 0.507
epoch 3, loss 809.8640, train acc 0.910, test acc 0.504
epoch 4, loss 695.2549, train acc 0.911, test acc 0.507
epoch 5, loss 648.2086, train acc 0.913, test acc 0.500
epoch 6, loss 723.6950, train acc 0.911, test acc 0.514
epoch 7, loss 766.9381, train acc 0.910, test acc 0.500
epoch 8, loss 749.4564, train acc 0.910, test acc 0.500
```

3. Covariate Shift Correction

Having observed that covariate shift can be harmful, let's try fixing it. For this we first need to compute the appropriate propensity scores $\frac{dp(x)}{dq(x)}$. For this purpose pick a biased dataset, let's say with $\lambda = 0.1$ and try to fix the covariate shift.

1. When training a logistic regression binary classifier to fix covariate shift, we assumed so far that both sets are of equal size. Show that re-weighting data in training and test set appropriately can help address the issue when both datasets have different size. What is the weighting?
2. Train a binary classifier (using logistic regression) distinguishing between the biased training set and the unbiased test set. Note - you need to weigh the data.
3. Use the scores to compute weights on the training set. Do they match the weight arising from the biasing distribution λ?
4. Train a binary classifier of the covariate shifted problem using the weights obtained previously and report the accuracy. Note - you will need to modify the training loop slightly such that you can compute the gradient of a weighted sum of losses.

Answer

1. We need to minimize the loss function when the data come from a unbiased dataset, where the label x has a distribution function $p(x)$:
$$\min_w \int dxp(x) \int dyp(y|x)l(f(x,w),y)$$
where $p(x)$ is the 'correct' distribution of label x and $q(x)$ is biased.
While the empirical form could be described as:
$$\min_x \frac{1}{n} \sum_i (l(x_i, y_i), f(x_i))$$
When the data come from biased dataset (here is the training set, where x has the distribution $q(x)$, the formula becomes:
$$\min_w \int dxq(x) \int dyp(y|x)l(f(x,w),y)$$
So re-weighting data to adjust the $q(x)$ back to $p(x)$ would help the function become unbiased. Therefore, the weight when training the binary classification should be:
$$\beta(x) = \frac{p(x)}{q(x)}$$
The joint probability distribution is (label data from train as -1, from test as 1:
$$r(x,y) = \frac{N_{test}}{N} p(x)\delta(y,1) + \frac{N_{train}}{N} q(x)\delta(y,-1)$$
$$r(y=1|x) = \frac{r(x,y=1)}{r(x)} = \frac{r(x,y=1)}{r(x|y=1)+r(x|y=-1)} = \frac{r(x,y=1)}{r(x,y=1)+r(x,y=-1)}$$
$$r(y=-1|x) = \frac{r(x,y=-1)}{r(x)} = \frac{r(x,y=-1)}{r(x|y=-1)+r(x|y=1)} = \frac{r(x,y=-1)}{r(x,y=1)+r(x,y=}$$
$$\beta(x) = \frac{p(x)}{q(x)} = \frac{r(y=1|x)}{r(y=-1|x)} * \frac{N_{train}}{N_{test}} = \frac{N_{train}}{N_{test}} exp(f(x))$$
Therefore, the weight is:

$$\frac{N_{train}}{N_{test}} exp(f(x))$$

For question 2, weighted loss for training

In [8]:
```
# preperations
def logistic(z):
    return 1. / (1. + nd.exp(-z))
# loss function, with weight
def log_loss(output, y, ratio):
    yhat = logistic(output)
    yhat = yhat.reshape(shape=y.shape)
    # biased loss
    return  - nd.nansum((1-ratio)/(.5+ratio) * y * nd.log(yhat) + ratio/
    # original loss
    # return  - nd.nansum(y * nd.log(yhat) + (1-y) * nd.log(1-yhat))
# train_model
def train_model(epochs, train_data, net, trainer, batch_size, ratio, f=N
    for e in range(epochs):
        cumulative_loss = 0
        for i, (data, label) in enumerate(train_data):
            with autograd.record():
                if f is not None:
                    ratio = f(data)
                    print('now the ratio is:', ratio)
                output = net(data)
                # print('weigth of net:', net.bias.data(), net.weight.da
                # print('output of net:', output)
                loss = log_loss(output, label, ratio)
            loss.backward()
            trainer.step(batch_size)
            cumulative_loss += nd.sum(loss).asscalar()
        print("Epoch %s, loss: %s" % (e + 1, cumulative_loss ))
# test_model
def test_model(test_data):
    num_correct = 0.0
    num_total = 0
    for i, (data, label) in enumerate(test_data):
        num_total += len(label)
        output = net(data)
        # visual = nd.concat(output.reshape(shape=(64,1)), label.reshape
        # print(visual)
        prediction = ((nd.sign(output).reshape(shape=label.shape) + 1) /
        num_correct += nd.sum(prediction == label)
    print("Accuracy: %0.3f (%s/%s)" % (num_correct.asscalar()/num_total,
```

In [9]:
```
# combine train and test
def train_and_test_mnist_ratio(train_data, test_data, net, trainer, num_
    train_data, test_data = preprocess(mnist_train, mnist_test, total_pe
    train_data = gluon.data.DataLoader(train_data, batch_size=batch_size
    test_data = gluon.data.DataLoader(test_data, batch_size=batch_size,
    train_model(num_epochs, train_data, net, trainer, batch_size, ratio)
    test_model(test_data)
```

In [10]:
```
def preprocess(mnist_train, mnist_test, total_per_label, ratio): # ratio
    X, y = mnist_train[:]
    # pick up the indices
    index_sweater = np.where(y==3)[0]
    index_shirt = np.where(y==6)[0]
    index_scandal = np.where(y==5)[0]
    index_sneaker = np.where(y==7)[0]
    # create the class for training, biased
    class_sweater = X[index_sweater[0:round(total_per_label*ratio)]]
    class_shirt = X[index_shirt[0:round(total_per_label*ratio)]]
    class_scandal = X[index_scandal[0:round(total_per_label*(1-ratio))]]
    class_sneaker = X[index_sneaker[0:round(total_per_label*(1-ratio))]]
    # print(class_sweater.shape, class_shirt.shape, class_scandal.shape,
    train_feature = nd.concat(class_sweater, class_shirt, class_scandal,
    train_feature = nd.flatten(train_feature)
    label = nd.ones((1, total_per_label*2)).astype(np.float32)
    train_labels = label.reshape(shape=(-1,))
    train_data = gdata.dataset.ArrayDataset(train_feature, train_labels)
    # create the class for testing, unbiased
    X, y = mnist_test[:]
    index1 = np.where(np.logical_or(y==3, y==6))[0]
    index2 = np.where(np.logical_or(y==5, y==7))[0]
    class1 = X[index1]
    class2 = X[index2]
    test_feature = nd.concat(class1, class2, dim=0)
    test_feature = nd.flatten(test_feature)
    label = nd.zeros((1, 4000)).astype(np.float32)
    test_labels = label.reshape(shape=(-1,))
    test_data = gdata.dataset.ArrayDataset(test_feature, test_labels)
    return train_data, test_data
def generator(ratio, batch_size):
    mnist_train = gdata.vision.FashionMNIST(train=True,transform=lambda
    mnist_test = gdata.vision.FashionMNIST(train=False,transform=lambda
    bias_train, bias_test = preprocess(mnist_train, mnist_test, 6000, ra
    #train's label should be 1 and test's label should be 0
    f_train, l_train = bias_train[:]
    f_test, l_test = bias_test[:]

    l_train = nd.ones((1, 12000)).astype(np.float32)
    l_test = nd.zeros((1,4000)).astype(np.float32)

    trainLabel = nd.concat(l_train[:,:8000],l_test[:,:3000],dim = 1).res
    testLabel = nd.concat(l_train[:,8000:],l_test[:,3000:],dim = 1).resh
    trainFeature = nd.concat(f_train[:8000], f_test[:3000], dim = 0)
    testFeature = nd.concat(f_train[8000:], f_test[3000:], dim = 0)

    train_data = gdata.dataset.ArrayDataset(trainFeature, trainLabel)
    test_data = gdata.dataset.ArrayDataset(testFeature, testLabel)
    return train_data, test_data
```

In [11]:
```
mnist_train = gdata.vision.FashionMNIST(train=True, transform = lambda d
                                        (data.astype(np.float32)/255.0,
mnist_test = gdata.vision.FashionMNIST(train=False, transform = lambda d
                                        (data.astype(np.float32)/255.0,
net = nn.Dense(1)
net.collect_params().initialize(mx.init.Normal(sigma=.1))
trainer = gluon.Trainer(net.collect_params(), 'sgd', {'learning_rate': 0
batch_size = 64
num_epochs = 80
ratio = 0.1
train_data, test_data = generator(ratio, batch_size)
train_data = gluon.data.DataLoader(train_data, batch_size=batch_size, sh
test_data = gluon.data.DataLoader(test_data, batch_size=batch_size, shuf
train_model(num_epochs, train_data, net, trainer, batch_size, 3./11)
test_model(test_data)
# this is the weight function we want, save to use later
```

```
Epoch 1, loss: 3574.4570150375366
Epoch 2, loss: 3151.593458175659
Epoch 3, loss: 3070.2279958724976
Epoch 4, loss: 3023.494716644287
Epoch 5, loss: 2990.501064300537
Epoch 6, loss: 2964.288585662842
Epoch 7, loss: 2943.6569423675537
Epoch 8, loss: 2925.1764526367188
Epoch 9, loss: 2907.9407787323
Epoch 10, loss: 2893.4413805007935
Epoch 11, loss: 2879.601933479309
Epoch 12, loss: 2867.217098236084
Epoch 13, loss: 2856.6793184280396
Epoch 14, loss: 2846.189125061035
Epoch 15, loss: 2836.723868370056
Epoch 16, loss: 2826.2606382369995
Epoch 17, loss: 2817.787402153015
Epoch 18, loss: 2810.8221797943115
Epoch 19, loss: 2804.0963249206543
Epoch 20, loss: 2796.3563499450684
Epoch 21, loss: 2788.6778297424316
Epoch 22, loss: 2783.7293033599854
Epoch 23, loss: 2777.1588201522827
Epoch 24, loss: 2771.2103633880615
Epoch 25, loss: 2766.2930002212524
Epoch 26, loss: 2760.329577445984
Epoch 27, loss: 2756.365584373474
Epoch 28, loss: 2751.379147529602
Epoch 29, loss: 2747.0357389450073
Epoch 30, loss: 2742.0883922576904
Epoch 31, loss: 2738.3803358078003
Epoch 32, loss: 2734.4318132400513
Epoch 33, loss: 2730.915614128113
Epoch 34, loss: 2727.3973693847656
Epoch 35, loss: 2723.671293258667
Epoch 36, loss: 2719.468644142151
Epoch 37, loss: 2717.504571914673
Epoch 38, loss: 2713.8761711120605
Epoch 39, loss: 2711.096170425415
```

```
Epoch 40, loss: 2706.931444168091
Epoch 41, loss: 2704.6794443130493
Epoch 42, loss: 2702.438359260559
Epoch 43, loss: 2698.7244987487793
Epoch 44, loss: 2697.5324392318726
Epoch 45, loss: 2695.2180557250977
Epoch 46, loss: 2690.636239051819
Epoch 47, loss: 2690.477437019348
Epoch 48, loss: 2687.6634464263916
Epoch 49, loss: 2685.435894012451
Epoch 50, loss: 2682.2733821868896
Epoch 51, loss: 2681.0932216644287
Epoch 52, loss: 2679.2474431991577
Epoch 53, loss: 2676.489851951599
Epoch 54, loss: 2675.4940042495728
Epoch 55, loss: 2674.1509103775024
Epoch 56, loss: 2671.923345565796
Epoch 57, loss: 2670.5356607437134
Epoch 58, loss: 2668.2608137130737
Epoch 59, loss: 2666.223602294922
Epoch 60, loss: 2664.28324508667
Epoch 61, loss: 2663.6015129089355
Epoch 62, loss: 2661.742232322693
Epoch 63, loss: 2660.468403816223
Epoch 64, loss: 2659.1579570770264
Epoch 65, loss: 2655.5567922592163
Epoch 66, loss: 2656.093068599701
Epoch 67, loss: 2653.663408279419
Epoch 68, loss: 2652.369168281555
Epoch 69, loss: 2651.3128700256348
Epoch 70, loss: 2647.1848888397217
Epoch 71, loss: 2649.020571708679
Epoch 72, loss: 2648.1360177993774
Epoch 73, loss: 2646.039858818054
Epoch 74, loss: 2644.6305589675903
Epoch 75, loss: 2643.320749282837
Epoch 76, loss: 2642.3576078414917
Epoch 77, loss: 2641.8019618988037
Epoch 78, loss: 2640.8091259002686
Epoch 79, loss: 2639.210654258728
Epoch 80, loss: 2638.227551460266
Accuracy: 0.800 (4000.0/5000)
```

In [12]:
```
bias_train, bias_test = preprocess(mnist_train, mnist_test, 6000, .2)
X, y = bias_train[:]
weight = nd.clip(nd.exp(net(X)), 0.01, 10)
weight
```

Out[12]:
```
[[ 1.1918093]
 [ 1.629158 ]
 [ 1.3973526]
 ...
 [10.       ]
 [ 8.610544 ]
 [10.       ]]
<NDArray 12000x1 @cpu(0)>
```

In [ ]: