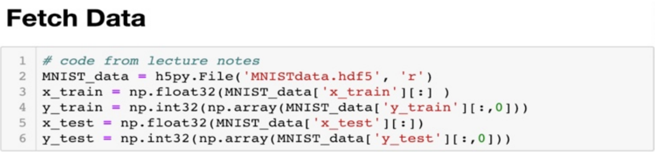
CS398-Deep Learning
Homework 1
Lingyi Xu (lingyix2)

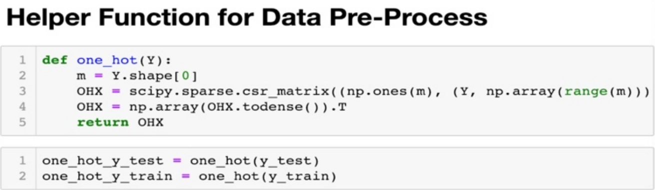
1. Description of Implementation

First, I load the MNIST data from the dataset and preprocess the labels to one-hot format.



Fetch Data

```
# code from lecture notes
MNIST_data = h5py.File('MNISTdata.hdf5', 'r')
x_train = np.float32(MNIST_data['x_train'][:] )
y_train = np.int32(np.array(MNIST_data['y_train'][:,0]))
x_test = np.float32(MNIST_data['x_test'][:])
y_test = np.int32(np.array(MNIST_data['y_test'][:,0]))
```



Helper Function for Data Pre-Process

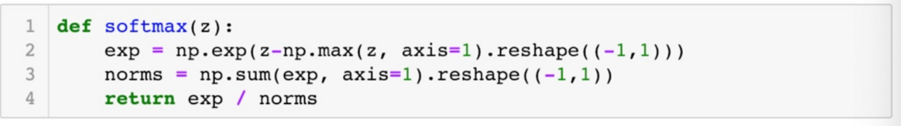
```
def one_hot(Y):
    m = Y.shape[0]
    OHX = scipy.sparse.csr_matrix((np.ones(m), (Y, np.array(range(m))))
    OHX = np.array(OHX.todense()).T
    return OHX
```

```
one_hot_y_test = one_hot(y_test)
one_hot_y_train = one_hot(y_train)
```

Then I build my mini-batch softmax logistic regression model.

I define the softmax function I will use to classify the probabilities:

$$F_{\text{softmax}}(z) = \frac{1}{\sum_{k=0}^{K-1} e^{z_k}} \left(e^{z_0}, e^{z_1}, \dots, e^{z_{K-1}} \right),$$



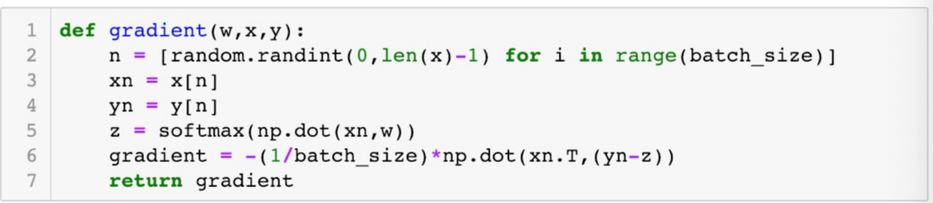
```
def softmax(z):
    exp = np.exp(z-np.max(z, axis=1).reshape((-1,1)))
    norms = np.sum(exp, axis=1).reshape((-1,1))
    return exp / norms
```

And the gradient I will use for update W:

When training, I first initialize W, a zero matrix with shape (784,10). Then I use 2800 iterations to update the W by gradient decent.

For each iteration, I calculate the gradient of F(w) from the sampling:

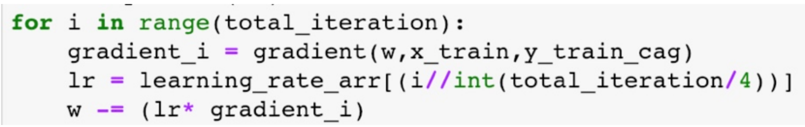
$$\nabla_{\theta_j} J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} \left[x^{(i)} \left(1\{y^{(i)} = j\} - p(y^{(i)} = j | x^{(i)}; \theta) \right) \right]$$



```
def gradient(w,x,y):
    n = [random.randint(0,len(x)-1) for i in range(batch_size)]
    xn = x[n]
    yn = y[n]
    z = softmax(np.dot(xn,w))
    gradient = -(1/batch_size)*np.dot(xn.T,(yn-z))
    return gradient
```

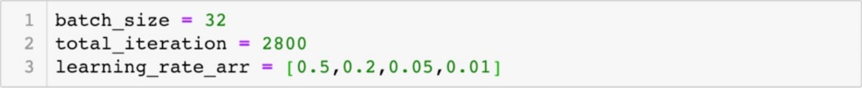
After calculating the gradient of the function, we can update the parameter matrix w by (where "theta" is "W" in my implementation)

$$\theta^{(\ell+1)} = \theta^{(\ell)} - \alpha^{(\ell)} G^{(\ell)},$$



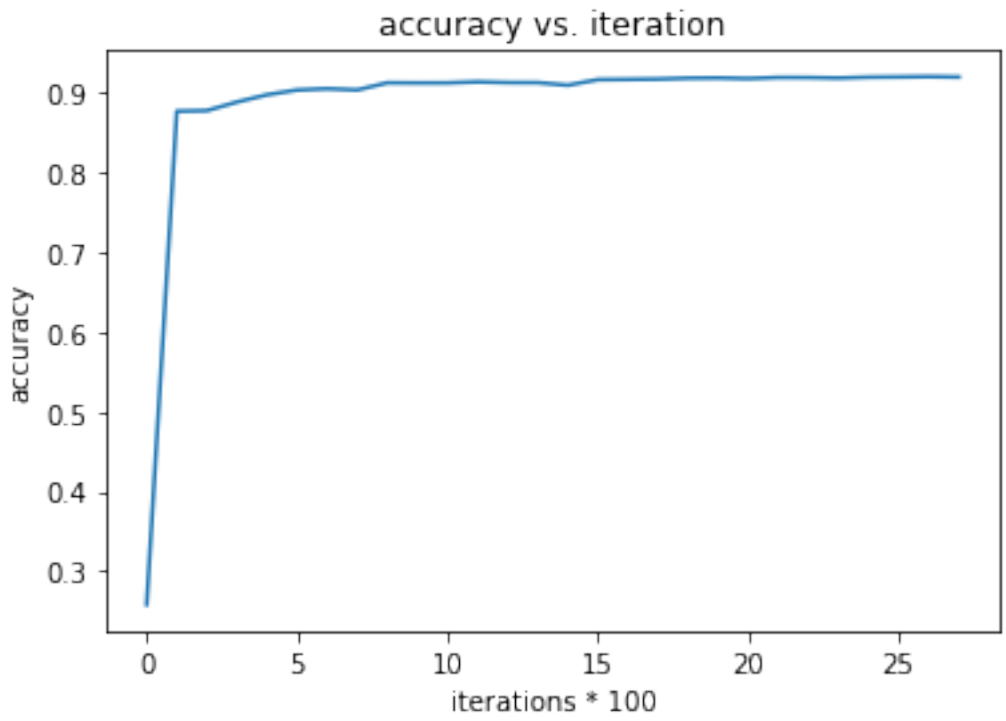
```
for i in range(total_iteration):
    gradient_i = gradient(w,x_train,y_train_cag)
    lr = learning_rate_arr[(i//int(total_iteration/4))]
    w -= (lr* gradient_i)
```

For this particular dataset, I use the following parameters:
Iteration = 2800
Batch size = 32
Learning rate = 0.5 when 0<=itr<700;
0.2 when 700<=itr<1400;
0.05 when 1400<=itr<2100;
0.01 when 2100<=itr<2800



```
batch_size = 32
total_iteration = 2800
learning_rate_arr = [0.5,0.2,0.05,0.01]
```

2. **Final Test Accuracy:**
91.9%


accuracy vs. iteration
accuracy
iterations * 100