CSC 411: Assignment #2

Xiangyu Kong
kongxi16

February 19, 2018

Problem 1

The data set contains hand-written digits from 0 to 9 (Fig 1). Among these data, most data are labeled accurately. However, some data are hard to be distinguished and even human can't really predict the digit.(Fig 2)

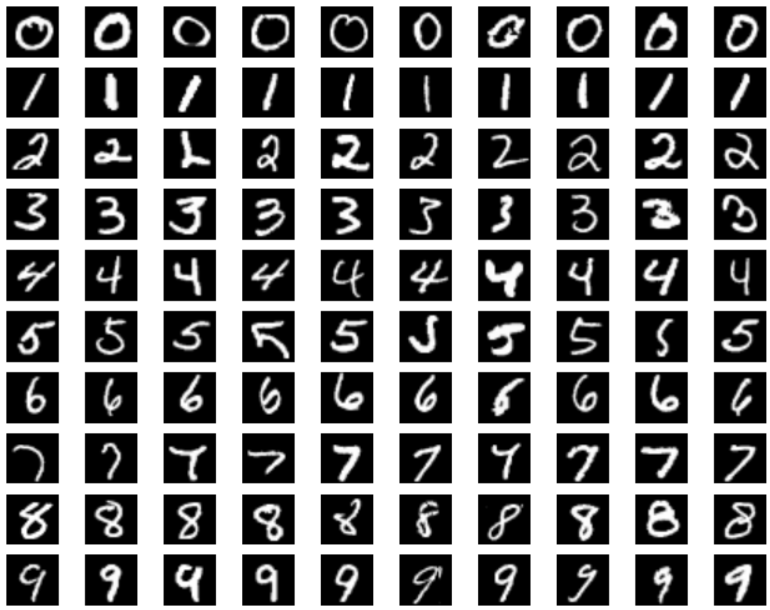
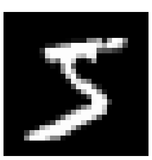
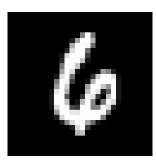


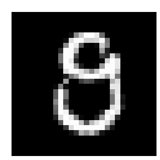
Figure 1: Full data



(a) 5 but looks like a 3



(b) 6 but looks like a 4



(c) 9 but looks like an 8

Figure 2: Inaccurate Labels

Problem 2

The output should be:

$$o^{(i)} = \sum_j w_j x_j^{(i)} + b^{(i)}$$

The listing of the implementation is as follows:

Listing 1: code for linear net output

```
def linear_forward(x, W):
    lin_output = np.dot(W.T, x)
    return softmax(lin_output)
```

Problem 3

1. Let the loss function C be defined as:

$$C = -\sum_i y^{(i)} log(p_i)$$

where p_i is:

$$p_i = \frac{e^{o_i}}{\sum_j e^{o_j}}$$

and o_i is:

$$o_i = \sum_j x_j^{(i)} w_{ij} + b_i$$

The gradient for the loss function with respect to the weight w_{ij} $\frac{\partial C}{\partial w_{ij}}$ is:

$$\frac{\partial C}{\partial w_{ij}} = \frac{\partial C}{\partial o_i}\frac{\partial o_i}{\partial w_{ij}}$$

Note:

$$\frac{\partial p_k}{\partial o_i} = \begin{cases} -p_k p_i & \text{if } k \neq i \\ p_i(1-p_i) & \text{if } k = i \end{cases}$$

Then:

$$\begin{aligned}
\frac{\partial C}{\partial o_i} &= -\frac{\partial C}{\partial p_i}\frac{\partial p_i}{\partial o_i} + \sum_{k\neq i}\frac{\partial C}{\partial p_k}\frac{\partial p_k}{\partial o_i} \\
&= -\frac{y^{(i)}}{p_i} p_i(1-p_i) + \sum_{k\neq i}\frac{y^{(k)}}{p_k} p_k p_i \\
&= p_i y^{(i)} - y^{(i)} + \sum_{k\neq i} y^{(k)} p_i \\
&= \sum_k y^{(k)} p_i - y^{(i)} \\
&= p_i - y^{(i)}
\end{aligned}$$

Also,

$$\frac{\partial o_i}{\partial w_{ij}} = x_j^{(i)}$$

Then

$$\begin{aligned}
\frac{\partial C}{\partial w_{ij}} &= \frac{\partial C}{\partial o_i}\frac{\partial o_i}{\partial w_{ij}} \\
&= x_j^{(i)}(p_i - y^{(i)})
\end{aligned}$$

2. The vectorized implementation is as follows:

Listing 2: Vectorized gradient

```
def linear_forward(x, W):
        lin_output = np.dot(W.T, x)
        return softmax(lin_output)

def loss(x, W, y):
        p = linear_forward(x, W)
        return -np.sum(y * np.log(p)) / x.shape[1]

def dlossdw(x, W, y):
        p = linear_forward(x, W)
        return np.matmul((p - y), x.T).T
```

Problem 4

Problem 5

Problem 6

Problem 7

Problem 8

Problem 9

Problem 10