Neural Network Hw1

Hao-en Sung (wrangle1005@gmail.com), Haifeng Huang (hah086@ucsd.edu)
Department of Computer Science
University of California, San Diego
San Diego, CA 92092

Abstract

In this assignment, we get familiar with Logistic Regression and Softmax Regression. Logistic Regression is used to classify two classes while Softmax Regression is just an extension for more classes. In both regression algorithms, we use gradient descent to update our weight vector w and stop training when the error on the hold-out set, which is 10% of the original training set and a stand-in for the unseen test set, is increasing by 3 epochs contiguously. We also introduce another regularization factor $\lambda C(w)$ to loss function to penalize too complex model, which is potentially going to over-fit the training set. To Sum up, we achieve 97% accuracy with Logistic Regression on discriminating test handwritten digits for two task cases — 2 v.s. 3 and 2 v.s. 8, while we achieved 86.15% accuracy with Softmax Regression on discriminating test handwritten digits for 10-class classification.

1 Problems from Bishop (20 points)

1.1 Hao-en Sung's Work

1.1.1 Problem 1.1

With (1.41), it is clear that one can replace $\lambda = 2$ in the left hand side of (1.42) and get

$$\prod_{i=1}^{d} \int_{-\infty}^{\infty} e^{-x_i^2} dx_i = \prod_{i=1}^{d} (\frac{2\pi}{2})^{\frac{1}{2}} = \pi^{\frac{d}{2}}. \tag{1}$$

One then can rewrite (1.42) as

$$\pi^{\frac{d}{2}} = S_d \cdot \int_0^{\infty} e^{-r^2} r^{d-1} dr. \tag{2}$$

Let $u = r^2$, it then can be derived that $du = 2rdr$. After that, one can again rewrite Eq. 1.1.1 as

$$\pi^{\frac{d}{2}} = \frac{1}{2} \cdot S_d \cdot \int_0^{\infty} e^{-r^2} r^{d-2} \cdot 2r dr \tag{3}$$

$$2\pi^{\frac{d}{2}} = S_d \cdot \int_0^{\infty} e^{-u} u^{\frac{d-2}{2}} du \tag{4}$$

$$S_d = \frac{2\pi^{\frac{d}{2}}}{\int_0^{\infty} e^{-u} u^{\frac{d-2}{2}} du} = \frac{2\pi^{\frac{d}{2}}}{\Gamma(\frac{d}{2})}. \tag{5}$$

On one hand, when $d = 2$, $S_d = 2\pi$, which is the well-known circumference of a unit circle in 2D plane. On the other hand, when $d = 3$, $S_d = \frac{2\pi^{\frac{3}{2}}}{\frac{\sqrt{\pi}}{2}} = 4\pi$, which is a well-known surface of a unit ball area in 3D plane.

30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

1.1.2 Problem 1.2

In polar coordinate, one can write down the volume of d-dimensional ball as

$$V_d = \int_0^a \int_{S_d} r^{d-1} d\Omega dr \tag{6}$$

$$= \int_{S_d} d\Omega \cdot \int_0^a r^{d-1} dr \tag{7}$$

$$= \frac{S_d \cdot a^d}{d}. \tag{8}$$

With this conclusion, one then can calculte the ratio between sphere volume and cube volume as

$$\frac{\frac{S_d \cdot a^d}{d}}{(2a)^d} = \frac{\frac{2\pi^{\frac{d}{2}}}{\Gamma(\frac{d}{2})}}{2^d d} \tag{9}$$

$$= \frac{\pi^{\frac{d}{2}}}{d2^{d-1}\Gamma(\frac{d}{2})}. \tag{10}$$

On top of that, with Stirling's approximation, one have

$$\frac{\pi^{\frac{d}{2}}}{d2^{d-1}\Gamma(\frac{d}{2})} = \frac{\pi^{\frac{d}{2}}}{d2^{d-1} \cdot (2\pi)^{\frac{1}{2}} e^{-\frac{d}{2}+1} (\frac{d}{2}-1)^{\frac{d-1}{2}}} \tag{11}$$

$$= \frac{\pi^{\frac{d}{2}} \cdot e^{\frac{d}{2}}}{d2^{d-1} \cdot e(2\pi)^{\frac{1}{2}} (\frac{d}{2}-1)^{\frac{d-1}{2}}} \tag{12}$$

$$= \frac{(\pi \cdot e)^{\frac{d}{2}}}{d2^{d-1} \cdot (2\pi)^{\frac{1}{2}} (\frac{d}{2}-1)^{\frac{d-1}{2}}} \tag{13}$$

Since it is known that $(\frac{d}{2}-1)^{\frac{d-1}{2}}$ increases much faster than $(\pi \cdot e)^{\frac{d}{2}}$ while d approaches infinity, we have $\lim_{d\to\infty} \frac{\pi^{\frac{d}{2}}}{d2^{d-1}\Gamma(\frac{d}{2})} = 0$.

Similarly, the ratio of the distance from the centre of the hypercube to one of the corners, divided by the perpendicular distance to one of the edges can be written as

$$\frac{\sqrt{\sum_{i=1}^{d} a^2}}{a} = \sqrt{d}. \tag{14}$$

Thus, we have $\lim_{d\to\infty} \sqrt{d} = \infty$.

1.1.3 Problem 1.3

From (1.45), one can write down the fraction of the volume of the sphere which lies at values of the radius between $a - \epsilon$ and ϵ as

$$f = \frac{\int_{a-\epsilon}^{a} \int_{S_d} \cdot r^{d-1} d\Omega dr}{\int_0^a \int_{S_d} \cdot r^{d-1} d\Omega dr} \tag{15}$$

$$= \frac{\frac{a^d - (a-\epsilon)^d}{d}}{\frac{a^d}{d}} \tag{16}$$

$$= 1 - (1 - \frac{\epsilon}{a})^d. \tag{17}$$

It is clear that when $d \to \infty$, $(1 - \frac{\epsilon}{d})^d \to 0$. Thus, $f \to 1$.

When $\frac{\epsilon}{a} = 0.01$, one can calculate

$$d = 2 \Rightarrow 1 - (1 - 0.01)^2 = 0.019900 \tag{18}$$
$$d = 10 \Rightarrow 1 - (1 - 0.01)^{10} \approx 0.095618 \tag{19}$$
$$d = 1000 \Rightarrow 1 - (1 - 0.01)^{1000} \approx 0.999957. \tag{20}$$

Similarly, the fraction of the volume of the sphere which lies inside the radius $\frac{a}{2}$ is

$$f = \frac{\int_{S_d} d\Omega \cdot \int_0^{\frac{a}{2}} r^{d-1} dr}{\int_{S_d} d\Omega \cdot \int_0^{a} r^{d-1} dr} \tag{21}$$
$$= \frac{\frac{(\frac{a}{2})^d}{d}}{\frac{a^d}{d}} \tag{22}$$
$$= \frac{1}{2^d}. \tag{23}$$

One then can calculate

$$d = 2 \Rightarrow \frac{1}{2^2} = 0.250000 \tag{24}$$
$$d = 10 \Rightarrow \frac{1}{2^10} \approx 0.00977 \tag{25}$$
$$d = 1000 \Rightarrow \frac{1}{2^1000} \approx 0.000000. \tag{26}$$

1.1.4 Problem 1.4

Just similar to what we did in Problem 1.2, we need to integral a shell but with thickness ϵ. Thus, the probability mass can be calculated as

$$\int_{S_a} r^{d-1} p(x) d\Omega \cdot \epsilon = \frac{S_d r^{d-1}}{(2\pi\sigma^2)^{\frac{1}{2}}} \exp\left(-\frac{r^2}{2\sigma^2}\right) \cdot \epsilon \tag{27}$$
$$= \rho(r) \cdot \epsilon. \tag{28}$$

Since Gaussian distribution forms like a bell, it has exactly one maximum when $\frac{\partial \rho(r)}{\partial r} = 0$, which can be derived as follows.

$$\frac{\partial \rho(r)}{\partial r} = 0 = \frac{S_d}{(2\pi\sigma^2)^{\frac{1}{2}}} \cdot \left((d-1)r^{d-2} - r^{d-1}\frac{r}{\sigma^2}\right) \cdot \exp\left(-\frac{r^2}{2\sigma^2}\right) \tag{29}$$
$$r^d = \sigma^2 (d-1) r^{d-2} \tag{30}$$
$$r^2 = \sigma^2 (d-1) \tag{31}$$
$$r = \sqrt{d-1}\sigma \approx \sqrt{d}\sigma \tag{32}$$

Finally, when $\epsilon << \hat{(}r)$, we have

$$\rho(\hat{(}r) + \epsilon) = \frac{S_d(\hat{r} + \epsilon)^{d-1}}{(2\pi\sigma^2)^{\frac{1}{2}}} \exp\left(-\frac{(\hat{r} + \epsilon)^2}{2\sigma^2}\right) \tag{33}$$

$$\approx \frac{S_d\hat{r}^{d-1}}{(2\pi\sigma^2)^{\frac{1}{2}}} \exp\left(-\frac{(\hat{r} + \epsilon)^2}{2\sigma^2}\right) \tag{34}$$

$$= \frac{S_d\hat{r}^{d-1}}{(2\pi\sigma^2)^{\frac{1}{2}}} \exp\left(-\frac{\hat{r}^2}{2\sigma^2}\right) \cdot \exp\left((\frac{\hat{r}\epsilon}{\sigma^2}\right) \cdot \exp\left((-\frac{\epsilon^2}{2\sigma^2}\right) \tag{35}$$

$$= \frac{S_d\hat{r}^{d-1}}{(2\pi\sigma^2)^{\frac{1}{2}}} \exp\left(-\frac{\hat{r}^2}{2\sigma^2}\right) \cdot \left(1 + \frac{\hat{r}\epsilon}{\sigma^2} + \cdots\right) \cdot \exp\left(-\frac{\epsilon^2}{2\sigma^2}\right) \tag{36}$$

$$\approx \frac{S_d\hat{r}^{d-1}}{(2\pi\sigma^2)^{\frac{1}{2}}} \exp\left(-\frac{\hat{r}^2}{2\sigma^2}\right) \cdot (1) \cdot \exp\left(-\frac{\epsilon^2}{2\sigma^2}\right) \tag{37}$$

$$= \rho(\hat{r}) \cdot \exp\left(-\frac{\epsilon^2}{2\sigma^2}\right). \tag{38}$$

1.2 Haifeng Huang's Work

1.2.1 Problem 1.1

(1) According to (1.41), set $\lambda = 2$, we can get

$$\prod_{i=1}^{d} \int_{-\infty}^{\infty} e^{-x_i^2} dx_i = \prod_{i=1}^{d} (\frac{2\pi}{2})^{\frac{1}{2}} = \pi^{\frac{d}{2}}. \tag{39}$$

According to (1.42), set $x = \frac{d}{2}$, we get

$$\Gamma(\frac{d}{2}) = \int_0^{\infty} u^{\frac{d}{2}-1} e^{-u} du. \tag{40}$$

Set $r = \sqrt{u}$,

$$S_d \cdot \int_0^{\infty} e^{-r^2} r^{d-1} dr = S_d \cdot \int_0^{\infty} e^{-u} \cdot u^{\frac{1}{2}(d-1)} \cdot \frac{1}{2} u^{-\frac{1}{2}} du = \tag{41}$$

$$\frac{S_d}{2} \int_0^{\infty} u^{\frac{d}{2}-1} e^{-u} du = \frac{S_d}{2} \cdot \Gamma(\frac{d}{2}) = \pi^{\frac{d}{2}}. \tag{42}$$

$$S_d = \frac{2\pi^{\frac{d}{2}}}{\Gamma(\frac{d}{2})}. \tag{43}$$

(2) When d=2,

$$S_d = \frac{2\pi}{\Gamma(1)} = 2\pi. \tag{44}$$

which is the perimeter of unit circle in 2-dimensions.
When d=3,

$$S_d = \frac{2\pi^{\frac{3}{2}}}{\Gamma(\frac{3}{2})} = \frac{2\pi^{\frac{3}{2}}}{\frac{\sqrt{\pi}}{2}} = 4\pi. \tag{45}$$

which is the surface area of unit sphere in 3-dimensions.

1.2.2 Problem 1.2

(1) Surface area of hypersphere of radius r in d-dimension is

$$S_{dr} = S_d \cdot r^{d-1} \tag{46}$$

So volume of hypersphere of radius a in d-dimension is

$$V_d = \int_0^a S_d \cdot r^{d-1} dr = \frac{S_d r^d}{d}|_0^a = \frac{S_d a^d}{d} \tag{47}$$

(2) The ratio of the volume of a hypersphere of radius a to the volume of a hypercube of side $2a$ is

$$\frac{\frac{S_d \cdot a^d}{d}}{(2a)^d} = \frac{S_d}{d \cdot 2^d} = \frac{\pi^{\frac{d}{2}}}{d \cdot 2^{d-1} \cdot \Gamma(\frac{d}{2})} \tag{48}$$

(3) According to Stirling's approximation

$$\Gamma(x+1) \approx (2\pi)^{\frac{1}{2}} e^{-x} x^{x+\frac{1}{2}} \tag{49}$$

Replace x with $\frac{d}{2} - 1$, we can get

$$\frac{\pi^{\frac{d}{2}}}{d 2^{d-1} \Gamma(\frac{d}{2})} = \frac{\pi^{\frac{d}{2}}}{d 2^{d-1} (2\pi)^{\frac{1}{2}} e^{-(\frac{d}{2}-1)} (\frac{d}{2}-1)^{\frac{d-1}{2}}} \tag{50}$$

$$= \frac{(\pi e)^{\frac{d}{2}}}{e d 2^{d-1} (2\pi)^{\frac{1}{2}} (\frac{d}{2}-1)^{\frac{d-1}{2}}} \tag{51}$$

Because $(\frac{d}{2}-1)^{\frac{d-1}{2}}$ is increasing as $(\frac{d}{2})^{\frac{d}{2}}$ as d approaches ∞, so

$$\lim_{d\to\infty}\frac{(\pi e)^{\frac{d}{2}}}{(\frac{d}{2})^{\frac{d}{2}}}=\lim_{d\to\infty}(\frac{2\pi e}{d})^{\frac{d}{2}}=(\lim_{d\to\infty}(\frac{2\pi e}{d}))^{\frac{d}{2}}=0^{\frac{d}{2}}=0 \tag{52}$$

$$\lim_{d\to\infty}\frac{\pi^{\frac{d}{2}}}{d2^{d-1}\Gamma(\frac{d}{2})}=0 \tag{53}$$

(4) The ratio of the distance from the centre of the hypercube to one of the corners, divided by the perpendicular distance to one of the edges can be written as

$$\frac{\sqrt{\sum_{i=1}^{d}a^2}}{a}=\sqrt{d}. \tag{54}$$

1.2.3 Problem 1.3

(1) According to (1.45), the fraction of the volume of the sphere which lies at values of the radius between $a-\epsilon$ and ϵ is

$$f=\frac{\frac{S_d\cdot a^d}{d}-\frac{S_d\cdot(a-\epsilon)^d}{d}}{\frac{S_d\cdot a^d}{d}}=1-(\frac{a-\epsilon}{a})^d=1-(1-\frac{\epsilon}{a})^d. \tag{55}$$

(2) Because $\lim_{x\to\infty}a^x=0$ when $|a|<1$, so

$$\lim_{d\to\infty}[1-(1-\frac{\epsilon}{a})^d]=1-\lim_{d\to\infty}(1-\frac{\epsilon}{a})^d=1-0=1 \tag{56}$$

(3) Since $0<\frac{\epsilon}{a}<1$, $0<1-\frac{\epsilon}{a}<1$.
When d = 2,

$$f=1-(1-0.01)^2=0.0199 \tag{57}$$

When d = 10,

$$f=1-(1-0.01)^{10}\approx 0.096 \tag{58}$$

When d = 100,

$$f=1-(1-0.01)^{100}\approx 0.99996 \tag{59}$$

(4) The fraction of the volume of the sphere which lies inside the radius $\frac{a}{2}$ is

$$f_2=\frac{\frac{(S_d\cdot\frac{a}{2})^d}{d}}{\frac{S_d\cdot a^d}{d}}=(\frac{1}{2})^d. \tag{60}$$

When d = 2,

$$f_2=(\frac{1}{2})^2=0.25 \tag{61}$$

When d = 10,

$$f_2=(\frac{1}{2})^{10}\approx 9.77\times 10^{-4} \tag{62}$$

When d = 100,

$$f=(\frac{1}{2})^{100}\approx 0 \tag{63}$$

1.2.4 Problem 1.4

(1) Because it is a thin shell, so ϵ is very small, we can assume that probability density function is constant for radius from r to $r+\epsilon$.
So the volume of this thin shell is

$$V = S_d \cdot r^{d-1} \cdot \epsilon \tag{64}$$

The probability mass inside this thin shell is

$$P = p(r) \cdot V = \frac{S_d \cdot r^{d-1} \cdot \epsilon}{(2\pi\sigma^2)^{\frac{d}{2}}} exp(-\frac{r^2}{2\sigma^2}) = \rho(r)\epsilon. \tag{65}$$

So

$$\rho(r) = \frac{S_d \cdot r^{d-1}}{(2\pi\sigma^2)^{\frac{d}{2}}} exp(-\frac{r^2}{2\sigma^2}). \tag{66}$$

(2)

$$\frac{d\rho(r)}{dr} = \frac{S_d}{(2\pi\sigma^2)^{\frac{1}{2}}} \cdot (r^{d-1} \cdot e^{-\frac{r^2}{2\sigma^2}})' = \tag{67}$$

$$\frac{S_d}{(2\pi\sigma^2)^{\frac{1}{2}}} \cdot [(d-1) \cdot r^{d-2} \cdot e^{-\frac{r^2}{2\sigma^2}} + r^{d-1} \cdot e^{-\frac{r^2}{2\sigma^2}} \cdot (-\frac{r}{\sigma^2})] = \tag{68}$$

$$\frac{S_d}{(2\pi\sigma^2)^{\frac{1}{2}}} \cdot e^{-\frac{r^2}{2\sigma^2}} [(d-1) \cdot r^{d-2} - \frac{r^d}{\sigma^2}] = 0. \tag{69}$$

So

$$(d-1) \cdot r^{d-2} = \frac{r^d}{\sigma^2}. \tag{70}$$

If $r \neq 0$,

$$(d-1)\sigma^2 = r^2 \tag{71}$$

$$\hat{r} = \sqrt{d-1}\sigma \approx \sqrt{d}\sigma. \tag{72}$$

(3) When $\epsilon << \hat{r}$ and d is large

$$\rho(\hat{r}+\epsilon) = \frac{S_d(\hat{r}+\epsilon)^{d-1}}{(2\pi\sigma^2)^{\frac{d}{2}}} \exp\left(-\frac{(\hat{r}+\epsilon)^2}{2\sigma^2}\right) \tag{73}$$

$$\approx \frac{S_d\hat{r}^{d-1}}{(2\pi\sigma^2)^{\frac{d}{2}}} \exp\left(-\frac{(\hat{r}+\epsilon)^2}{2\sigma^2}\right) \tag{74}$$

$$= \frac{S_d\hat{r}^{d-1}}{(2\pi\sigma^2)^{\frac{d}{2}}} \exp\left(-\frac{\hat{r}^2}{2\sigma^2}\right) \cdot \exp\left(-\frac{\hat{r}\epsilon}{\sigma^2}\right) \cdot \exp\left(-\frac{\epsilon^2}{2\sigma^2}\right) \tag{75}$$

$$\approx \frac{S_d\hat{r}^{d-1}}{(2\pi\sigma^2)^{\frac{d}{2}}} \exp\left(-\frac{\hat{r}^2}{2\sigma^2}\right) \cdot \exp\left(-\frac{\hat{r}\epsilon}{\sigma^2}\right) \tag{76}$$

$$\approx \frac{S_d\hat{r}^{d-1}}{(2\pi\sigma^2)^{\frac{d}{2}}} \exp\left(-\frac{\hat{r}^2}{2\sigma^2}\right) \cdot \exp\left(-\frac{\epsilon^2}{\sigma^2}\right) \tag{77}$$

$$= \rho(\hat{r}) \cdot \exp\left(-\frac{\epsilon^2}{\sigma^2}\right). \tag{78}$$

2 Logistic and Softmax Regression (35 points)

2.1 Derive the gradient for Logistic Regression

For sigmoid function $\sigma(z) = \frac{1}{1+\exp(-z)}$, we know its derivative is $\frac{\partial \sigma(z)}{\partial z} = \sigma(z)(1-\sigma(z))$.
We then can derive

$$y^n(w) = g_w(x^n) = \frac{1}{1+\exp(-w^T x^n)} \tag{79}$$

$$\frac{\partial y^n(w)}{\partial w_j} = y^n(w)(1-y^n(w))x_j^n. \tag{80}$$

Since we also know that

$$E^n(w) = -[t^n \ln y^n + (1-t^n)\ln(1-y^n)], \tag{81}$$

its derivative can be written as

$$\frac{\partial E^n(w)}{\partial y^n(w)} = -(\frac{t^n}{y^n(w)} - \frac{1-t^n}{1-y^n(w)}) \tag{82}$$

$$\frac{\partial E^n(w)}{\partial w_j} = \frac{\partial E^n(w)}{\partial y^n(w)} \cdot \frac{\partial y^n(w)}{\partial w_j} \tag{83}$$

$$= -(\frac{t^n}{y^n(w)} - \frac{1-t^n}{1-y^n(w)})y^n(w)(1-y^n(w))x_j^n \tag{84}$$

$$= [-t^n(1-y^n(w)) + (1-t^n)y^n(w)]x_j^n = (y^n - t^n)x_j^n \tag{85}$$

$$-\frac{\partial E^n(w)}{\partial w_j} = (t^n - y^n)x_j^n. \tag{86}$$

2.2 Derive the gradient for Softmax Regression.

If the n-th sample x^n belongs to class k, then the probability t_k^n that the n-th sample belongs to class k is 1 and the probability $t_{k'}^n$ $(k' \neq k)$ that the n-th sample belongs to class k' is 0, so $\sum_{k'} t_{k'}^n = 1$.

Since we know that $E^n(w) = -\sum t_k^n \ln y_k^n$ and $y_k^n = \frac{\exp(a_k^n)}{\sum_{k'} \exp(a_{k'}^n)}$, their deviatives can be derive as follows.

$$\frac{\partial E^n(w)}{\partial w_{jk}} = \frac{\partial E^n(w)}{\partial a_k^n} \frac{\partial a_k^n}{\partial w_{jk}} \tag{87}$$

$$\frac{\partial E^n(w)}{\partial a_k^n} = \sum_{k'} \frac{\partial E^n(w)}{\partial y_{k'}^n} \frac{\partial y_{k'}^n}{\partial a_k^n} \tag{88}$$

$$\frac{\partial E^n(w)}{\partial y_{k'}^n} = -\frac{t_{k'}^n}{y_{k'}^n} \tag{89}$$

$$\frac{\partial y_{k'}^n}{\partial a_k^n} = \frac{\delta_{kk'} \exp(a_{k'}^n)(\sum_{k'} \exp(a_{k'}^n)) - \exp(a_{k'}^n) \exp(a_k^n)}{(\sum_{k'} \exp(a_{k'}^n))^2} \tag{90}$$

$$= \delta_{kk'} \frac{\exp(a_{k'}^n)}{\sum_{k'} \exp(a_{k'}^n)} - \frac{\exp(a_{k'}^n)}{\sum_{k'} \exp(a_{k'}^n)} \frac{\exp(a_k^n)}{\sum_{k'} \exp(a_{k'}^n)} \tag{91}$$

$$= \delta_{kk'} y_{k'}^n - y_{k'}^n y_k^n \tag{92}$$

$$\frac{\partial E^n(w)}{\partial a_k^n} = \sum_{k'} (-\frac{t_{k'}^n}{y_{k'}^n})(\delta_{kk'} y_{k'}^n - y_{k'}^n y_k^n) \tag{93}$$

$$= \sum_{k'} (-\delta_{kk'} t_{k'}^n + t_{k'}^n y_k^n) \tag{94}$$

$$= -t_k^n + (\sum_{k'} t_{k'}^n) y_k^n \tag{95}$$

$$= y_k^n - t_k^n \tag{96}$$

$$\frac{\partial a_k^n}{\partial w_{jk}} = x_j^n \tag{97}$$

$$-\frac{\partial E^n(w)}{\partial w_{jk}} = (t_k^n - y_k^n) x_j^n \tag{98}$$

2.3 Read in Data.

We use the first $20,000$ training images and the first $2,000$ test images.

2.4 Logistic Regression via Gradient Descent.

2.4.1 Introduction to the problem

In this problem, we are going to plot the loss function and accuracy over training for both cases 2 v.s. 3 and 2 v.s. 8, respectively. After that, we are going to plot weights as an image for two classifiers (2 v.s. 3 and 2 v.s. 8) and the difference between them.

2.4.2 Methods and Results

We are going to use gradient descent to update the weight vector, which is initialized to a zero vector. For every iteration, we will iterate through the training set, use the formula $w_{t+1} = w_t - \eta \sum_{n=1}^{N} \nabla E^n(w)$, where the j-th element of $\nabla E^n(w)$ is $\frac{\partial E^n(w)}{\partial w_j}$ and we have proven in the first problem that $-\frac{\partial E^n(w)}{\partial w_j} = (t^n - y^n) x_j^n$. The learning rate is denoted as η, which is used to describe the rate of moving toward the opposite direction of gradient. To make it converge to the minimum rather than rush through it, we use the formula $\eta(t) = \frac{\eta(0)}{1+t/T}$ to reduce it over iteration epoch t, where we use $\eta(0) = 0.001$ and $T = 100$.

For case 2 v.s. 3, we extract those samples whose labels are 2 and 3 respectively for both images and labels, and then stack those labeled 2 and those labeled 3 for both images and labels. In the beginning, we initialize weight vector to 0 and use the formula $-\frac{\partial E^n(w)}{\partial w_j} = (t^n - y^n) x_j^n$ to update weight vector

in every iteration. Then we calculate the loss function for this weight vector on the validation set. If the loss function decreases, we continue the iteration over all the training set; otherwise, if the loss function increases for 3 epochs consistently, we stop training to avoid overfitting.

After we update weight vector in every iteration, we calculate the loss function and accuracy on training set, hold-out set, and test set, then append them to different lists. At the end of training process, we return these lists and plot the loss function and accuracy over training. Please refer Fig. 1 and Fig. 2 for the loss function and accuracy of case 2 v.s. 3; while refer Fig. 3 and Fig. 4 for the loss function and accuracy of case 2 v.s. 8.

We also retrieve the weighting vector and plot it in a 28×28 image without bias term. Please refer Fig. 5 for case 2 v.s. 3 and Fig. 6 for case 2 v.s. 8. Difference between two weighting vectors are also plotted as Fig. 7.

All implementation codes are included in Appendix as Code 1.

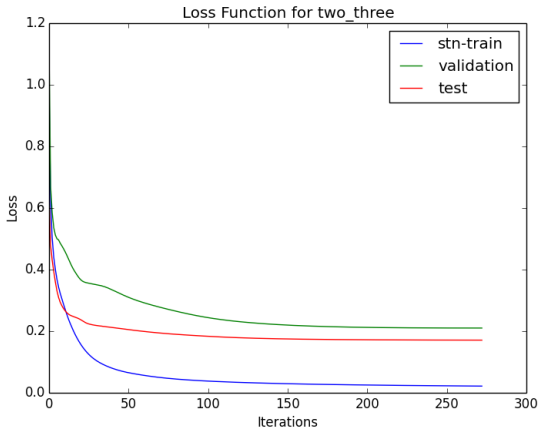

Figure 1: Loss Function for case 2 v.s. 3

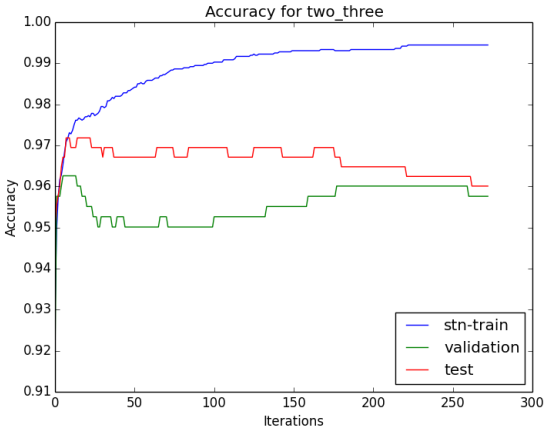

Figure 2: Accuracy for case 2 v.s. 3

2.4.3 Discussion

(a) From Fig. 1, Fig. 2, Fig. 3, and Fig. 4, we can see that validation set is a good stand-in for test set because for both loss function and accuracy, they have the same trend of increasing and decreasing. Actually, we can see that test set have a better performance than validation set.

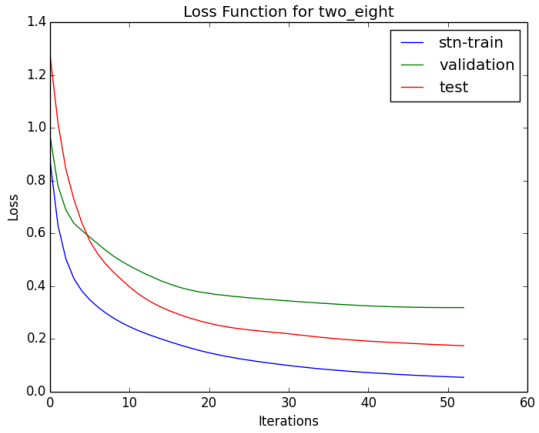

Figure 3: Loss Function for case 2 v.s. 8

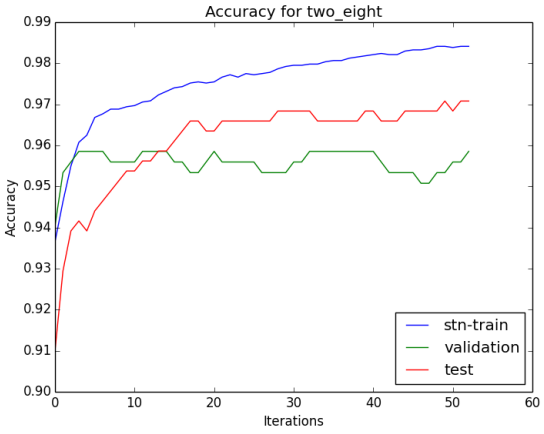

Figure 4: Accuracy for case 2 v.s. 8

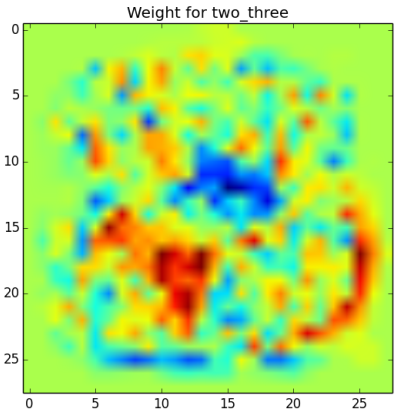

Figure 5: Weight Image for case 2 v.s. 3

(d) In the weight difference image in Fig. 7, the red part represents class 1 and the blue part represents class 0. We can see that the red part is the residual of 2 by subtracting weight image for case 2 v.s. 3

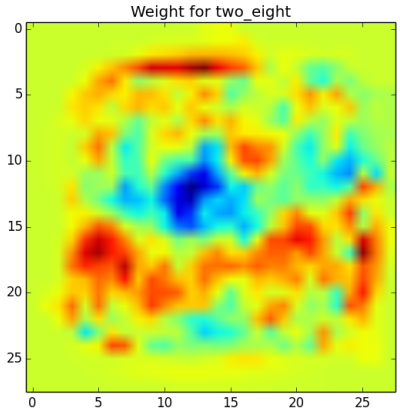

Figure 6: Weight Image for case 2 v.s. 8

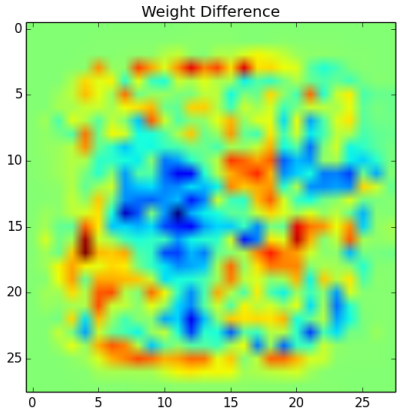

Figure 7: Weight Difference between case 2 v.s. 3 and case 2 v.s. 8

from weight image for case 2 v.s. 8, which is almost offset. The central blue part is the result of left half of 8 which is absent in 3. So the weight difference can be used to classify 3 and 8, actually, our weight difference achieves 85.2% accuracy on test set.

2.5 Regularization

2.5.1 Introduction to the problem

In this problem, we introduce a penalty term $\lambda C(w)$ for loss function, that is $J(w) = E(w) + \lambda C(w)$ is our new loss function. There are two kinds of regularization: $C(w) = \sum_{i,j} \|w_{i,j}\|_1$ for L1 regularization and $C(w) = \sum_{i,j} \left\|w_{i,j}^2\right\|_2$ for L2 regularization.

We are going to train a classifier for case 2 v.s. 3 and plot the accuracy over training, length of weight vector, test set error with different values of $\lambda = 0.01, 0.001, 0.0001$.

2.5.2 Methods and Results

Methods with two regularizations are the same as the fourth problem, except that the update term for weight vector should be deducted by $\lambda \frac{\partial C}{\partial w}$ before multiplying by η.

Please refer Fig. 8 and Fig. 9 for the loss function and accuracy of case 2 v.s. 3 over training. Error rate for test set is recorded in Fig. 10 with different λ. Moreover, the weight images for L1 and L2 regularization with different λ are included as Fig. 11 to Fig. 16.

All implementation codes are included in Appendix as Code 1.

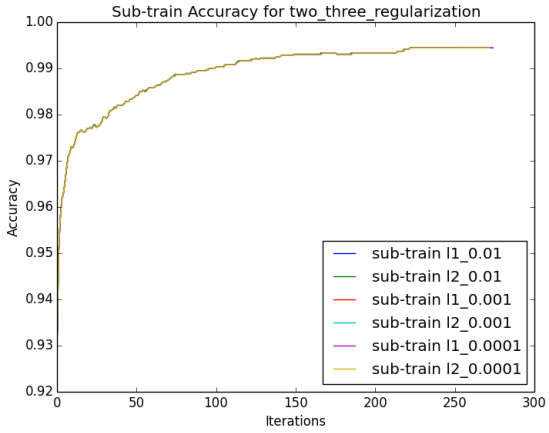

Figure 8: Accuracy of training set for case 2 v.s. 3 for different values of λ

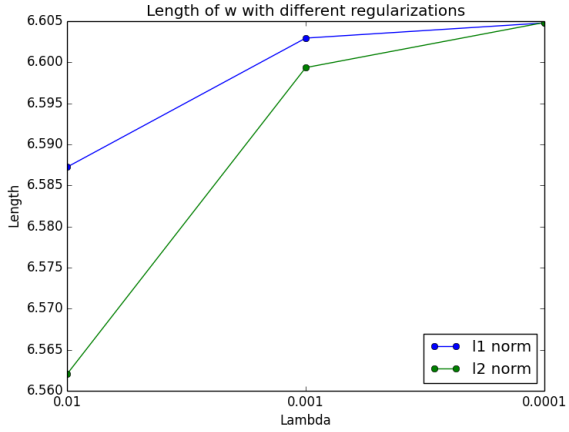

Figure 9: Weight length of training set for case 2 v.s. 3 for different values of λ

2.5.3 Discussion

(a) For L1 regularization, the (i,j)-th element of $\frac{\partial C}{\partial w}$ can be written as

$$\frac{\partial C}{\partial w} = \begin{cases} 1, & w_{i,j} > 0 \\ -1, & w_{i,j} < 0. \end{cases} \tag{99}$$

Since it is not differentiable when $w_{i,j} = 0$, so we define $\frac{\partial C}{\partial w} = 0$ when $w_{i,j} = 0$.

For L2 regularization, the (i,j)-th element of $\frac{\partial C}{\partial w}$ is $\frac{\partial C}{\partial w_{i,j}} = 2w_{i,j}$. Thus, $\frac{\partial C}{\partial w} = 2w$.

(b) For different values of λ and different kinds of regularization, accuracy are almost the same on the training set and are above 99% after 150 iterations.

(c) It is clear that with larger λ, which means the stronger regularization, we can observe the smaller length of the weight. We can also tell that L2 regularization are stronger because they have smaller length of weight when λ is equal.

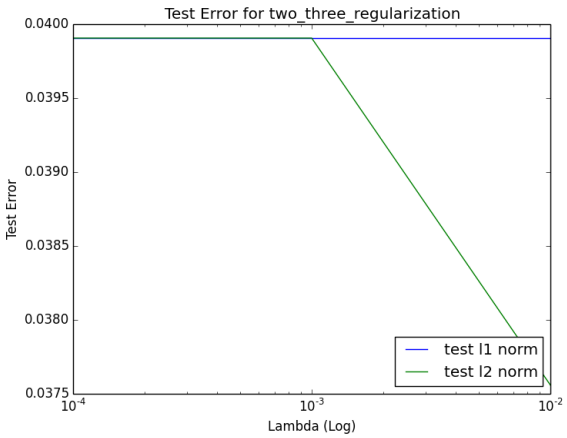

Figure 10: Test error rate for case 2 v.s. 3 for different values of λ

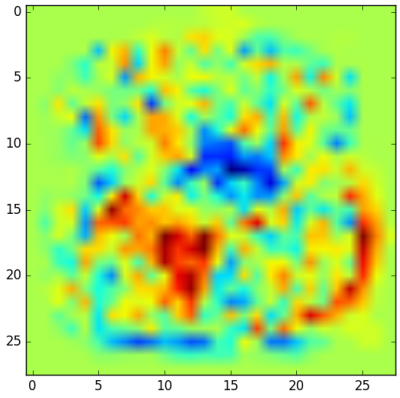

Figure 11: Weight Image for case 2 v.s. 3 using L1 regularization and $\lambda = 0.01$

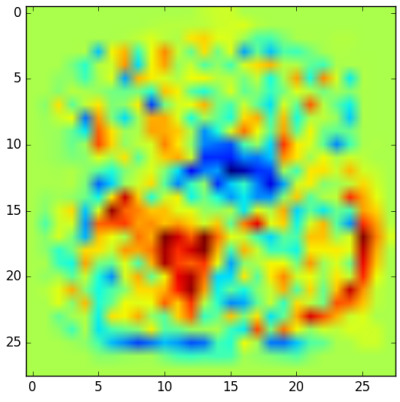

Figure 12: weight image for case 2 v.s. 3 using L2 regularization and $\lambda = 0.01$

(d) The larger λ, the stronger regularization, which also means the smaller test error rate. We can tell from Fig. 10 that it satisfies our own theory to some extent: for L2 regularization, $\lambda = 0.01$ has

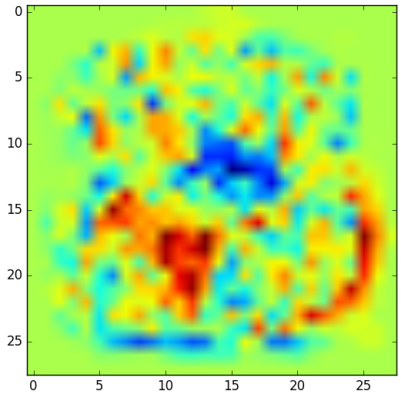

Figure 13: weight image for case 2 v.s. 3 using L1 regularization and $\lambda = 0.001$

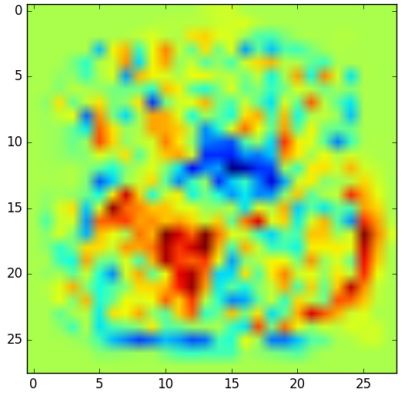

Figure 14: weight image for case 2 v.s. 3 using L2 regularization and $\lambda = 0.001$

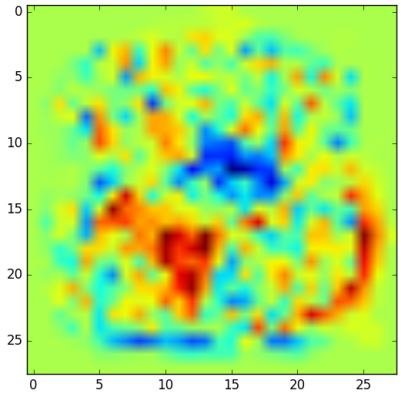

Figure 15: weight image for case 2 v.s. 3 using L1 regularization and $\lambda = 0.0001$

smaller test error rate than $\lambda = 0.001$ and 0.0001, but for L1 regularization, different λ has the same

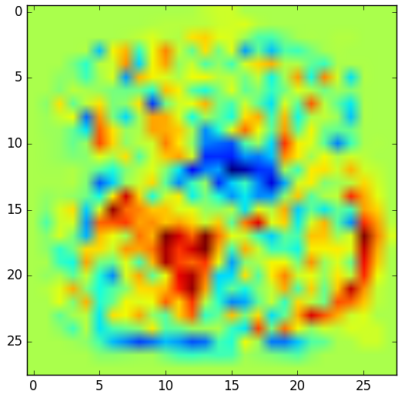

Figure 16: weight image for case 2 v.s. 3 using L2 regularization and $\lambda = 0.0001$

test error rate. We can also see that L2 regularization are stronger because they have smaller test error rate when $\lambda = 0.01$.

(e) They are almost the same. This means that for $\lambda = 0.01, 0.001, 0.0001$ and for L1 and L2 regularization, the outcome is almost the same, which can be seen from figure 8.

2.6 Softmax Regression via Gradient Descent.

2.6.1 Introduction to the problem

In this problem, we are going to plot the loss function and the accuracy over the number of training iterations for the training set, hold-out set, and test set.

2.6.2 Methods and Results

The methods for this problem is almost the same as the fourth problem except for the following aspects. First, the weight is not a vector of size 785 by 1 but a matrix of size 785 by 10, because we have 10 classes now. Thus, for every class, the weight vector is different. On top of that, the true label and the prediction are not vectors now, they should be converted to a matrix of size: number of samples by 10. For true label matrix, if a sample belongs to class k, we make the k-th element of the row vector corresponding to this sample to be 1, which denotes that the probability of this sample belonging to class k is 1, and set all other probabilities to 0. One can use the formula $y_k^n = \frac{exp(a_k^n)}{\sum_{k'} exp(a_{k'}^n)}$ and $a_k^n = w_k^T x^n$ to get the prediction y_k^n. Please refer Fig. 17 and Fig. 18 for the loss function and accuracy of 10-class classification.

All implementation codes are included in Appendix as Code 2.

2.6.3 Discussion

I use $\lambda = 0.01$ and L2 regularization for the reasoning in the discussion of the fifth problem.

3 Summary

In this assignment, we use Logistic regression for to classify case 2 v.s. 3 and case 2 v.s. 8 and get 97% accuracy and Softmax Regression to classify 10 classes and get 86.15% accurate rate.

From this assignment, we have learned how to update weight vectors or weight matrices by calculating gradient descent and reducing loss function, how to do L1 and L2 regularization, how to use Softmax Regression to classify samples of more than two classes.

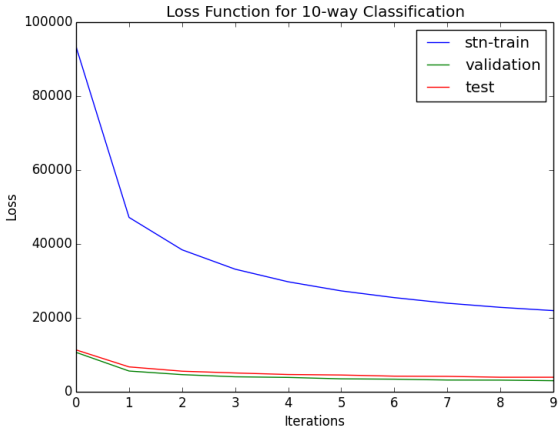

Figure 17: Loss function for 10 digits classifier

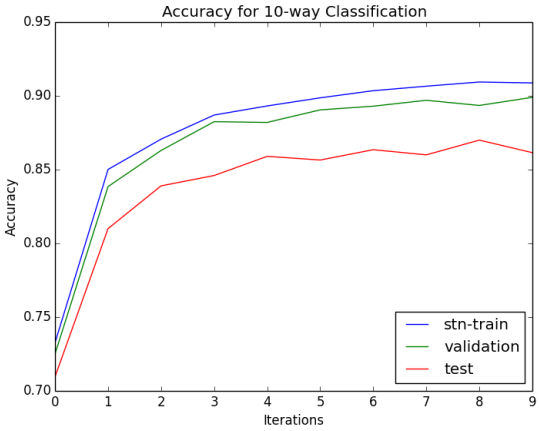

Figure 18: Accuracy for 10 digits classifier

4 Distribution

- Hao-en Sung is responsible for problem 3, 4, and 5.
- Haifeng Huang is responsible for problem 1, 2, 6, and 7.

5 Reference

Neural Networks for Pattern Recognition - Christopher Bishop

6 Appendix

6.1 Implementation Codes

Code Listing 1: Code for Logistic Regression

```
import os, struct
from array import array as pyarray
import numpy as np
import matplotlib.pyplot as plt

from sklearn import preprocessing
from sklearn import model_selection
from sklearn.model_selection import ShuffleSplit

tn_n = 20000
tt_n = 2000
ft_n = 784

INF = 10000000
MAX_ITER = 1000
eta = 0.001
eta_decay = 0.01
eps = 1e-10

img_folder = '../res/'

ss = ShuffleSplit(n_splits=1, test_size=0.1, random_state=0)


def load_mnist(dataset, digits=np.arange(10), path="../dat/"):
    """
    Loads MNIST files into 3D numpy arrays

    Adapted from: http://abel.ee.ucla.edu/cvxopt/_downloads/mnist.py
    """

    if dataset == "training":
        fname_img = os.path.join(path, 'train-images-idx3-ubyte')
        fname_lbl = os.path.join(path, 'train-labels-idx1-ubyte')
    elif dataset == "testing":
        fname_img = os.path.join(path, 't10k-images-idx3-ubyte')
        fname_lbl = os.path.join(path, 't10k-labels-idx1-ubyte')
    else:
        raise ValueError("dataset must be 'testing' or 'training'")

    flbl = open(fname_lbl, 'rb')
    magic_nr, size = struct.unpack(">II", flbl.read(8))
    lbl = pyarray("b", flbl.read())
    flbl.close()

    fimg = open(fname_img, 'rb')
    magic_nr, size, rows, cols = struct.unpack(">IIII", fimg.read(16))
    img = pyarray("B", fimg.read())
    fimg.close()

    ind = [ k for k in range(size) if lbl[k] in digits ]
    N = len(ind)

    images = np.zeros((N, rows, cols), dtype=np.uint8)
    labels = np.zeros((N, 1), dtype=np.int8)
    for i in range(len(ind)):
        images[i] = np.array(img[ ind[i]*rows*cols : \
                (ind[i]+1)*rows*cols ]).reshape((rows, cols))
        labels[i] = lbl[ind[i]]
```

```
    return images, labels


def calLOSS(w, x, y, p, lb1, lb2):
    assert(x.shape[0] == y.shape[0])

    val = 0.0
    for i in xrange(x.shape[0]):
        val -= y[i] * np.log(p[i]+eps) + (1-y[i]) * np.log(1-p[i]+eps)

    val *= 1.0 / x.shape[0]
    val += lb1 * np.linalg.norm(w, 1)
    val += lb2 * np.linalg.norm(w, 2)
    return val


def calACC(w, x, y, p):
    assert(x.shape[0] == y.shape[0] and y.shape[0] == p.shape[0])
    return 1.0 / x.shape[0] * np.sum(y == (p > 0.5))


def sigmoid(z):
    return 1.0 / (1 + np.exp(-z))


def predict(w, x):
    v = np.dot(x, np.transpose(w))
    return np.apply_along_axis(sigmoid, 0, v)


def LogisticRegression(sx, sy, vx, vy, tx, ty, lb1=0, lb2=0):
    assert(sx.shape[0] == sy.shape[0])
    assert(vx.shape[0] == vy.shape[0])
    assert(tx.shape[0] == ty.shape[0])

    w = np.zeros((1, ft_n+1))
    bw = np.zeros((1, ft_n+1))
    sp = predict(w, sx);
    vld_v = INF
    cnt = 0

    # results
    stn_loss = []
    vld_loss = []
    tt_loss = []
    stn_acc = []
    vld_acc = []
    tt_acc = []

    for t in xrange(MAX_ITER):
        nw = np.dot((sy-sp).transpose(),sx) - lb1*np.sign(w) - lb2*2*w

        n_eta = eta / (1 + t * eta_decay)
        w += n_eta * nw

        # predict
        sp = predict(w, sx)
        vp = predict(w, vx)
        tp = predict(w, tx)

        # calculate loss
        n_stn_v = calLOSS(w, sx, sy, sp, lb1, lb2)
        n_vld_v = calLOSS(w, vx, vy, vp, lb1, lb2)
        n_tt_v = calLOSS(w, tx, ty, tp, lb1, lb2)
```

```
        stn_loss.append(n_stn_v)
        vld_loss.append(n_vld_v)
        tt_loss.append(n_tt_v)

        # calculate acc
        n_stn_a = calACC(w, sx, sy, sp)
        n_vld_a = calACC(w, vx, vy, vp)
        n_tt_a = calACC(w, tx, ty, tp)
        stn_acc.append(n_stn_a)
        vld_acc.append(n_vld_a)
        tt_acc.append(n_tt_a)

        # stop condition
        if n_vld_v >= vld_v:
            cnt += 1
        else:
            bw = np.copy(w)
            vld_v = n_vld_v
            cnt = 0

        if cnt == 3:
            print 'Stop at #' + str(t)
            return bw, stn_loss[:-3], vld_loss[:-3], tt_loss[:-3], \
                    stn_acc[:-3], vld_acc[:-3], tt_acc[:-3]

    print 'Maximum iterations are reached'
    return bw, stn_loss[:-3], vld_loss[:-3], tt_loss[:-3], \
            stn_acc[:-3], vld_acc[:-3], tt_acc[:-3]


## read dataset
tn_x, tn_y = load_mnist('training')
tn_x, tn_y = tn_x[:tn_n, :, :], tn_y[:tn_n, :]
tn_x = tn_x.reshape(tn_n, ft_n)
tn_x = np.hstack([tn_x, np.ones((tn_n, 1))])

tt_x, tt_y = load_mnist('testing')
tt_x, tt_y = tt_x[:tt_n, :, :], tt_y[:tt_n, :]
tt_x = tt_x.reshape(tt_n, ft_n)
tt_x = np.hstack([tt_x, np.ones((tt_n, 1))])


## preprocessing
tn_x = preprocessing.scale(tn_x)
tt_x = preprocessing.scale(tt_x)


## Problem 4 (a), (b): 2 v.s 3

# training part
tn_two_idx = np.transpose(tn_y == 2)[0]
tn_three_idx = np.transpose(tn_y == 3)[0]
tn_two_x = tn_x[tn_two_idx, :]
tn_two_y = np.ones((tn_two_x.shape[0], 1))
tn_three_x = tn_x[tn_three_idx, :]
tn_three_y = np.zeros((tn_three_x.shape[0], 1))
tn_two_three_x = np.vstack((tn_two_x, tn_three_x))
tn_two_three_y = np.vstack((tn_two_y, tn_three_y))

# testing part
tt_two_idx = np.transpose(tt_y == 2)[0]
tt_three_idx = np.transpose(tt_y == 3)[0]
tt_two_x = tt_x[tt_two_idx, :]
tt_two_y = np.ones((tt_two_x.shape[0], 1))
tt_three_x = tt_x[tt_three_idx, :]
```

```
tt_three_y = np.zeros((tt_three_x.shape[0], 1))
tt_two_three_x = np.vstack((tt_two_x, tt_three_x))
tt_two_three_y = np.vstack((tt_two_y, tt_three_y))

# hold out 10% train as validation
stn_two_three_idx, vld_two_three_idx = \
        list(ss.split(np.arange(tn_two_three_x.shape[0])))[0]
stn_two_three_x = tn_two_three_x[stn_two_three_idx, :]
stn_two_three_y = tn_two_three_y[stn_two_three_idx, :]
vld_two_three_x = tn_two_three_x[vld_two_three_idx, :]
vld_two_three_y = tn_two_three_y[vld_two_three_idx, :]

# main model
two_three_w, stn_loss, vld_loss, tt_loss, stn_acc, vld_acc, tt_acc =\
        LogisticRegression(stn_two_three_x, stn_two_three_y, \
        vld_two_three_x, vld_two_three_y, tt_two_three_x, \
        tt_two_three_y, lb1=0, lb2=0)

# plot loss
plt.figure()
plt.plot(stn_loss, label='stn-train')
plt.plot(vld_loss, label='validation')
plt.plot(tt_loss, label='test')
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.title('Loss Function for two_three')
plt.legend(loc=1)
plt.savefig(img_folder + 'loss_two_three.png')

# plot acc
plt.figure()
plt.plot(stn_acc, label='stn-train')
plt.plot(vld_acc, label='validation')
plt.plot(tt_acc, label='test')
plt.xlabel('Iterations')
plt.ylabel('Accuracy')
plt.title('Accuracy for two_three')
plt.legend(loc=4)
plt.savefig(img_folder + 'acc_two_three.png')


## Problem 4 (c): 2 v.s 8

# training part
tn_two_idx = np.transpose(tn_y == 2)[0]
tn_eight_idx = np.transpose(tn_y == 8)[0]
tn_two_x = tn_x[tn_two_idx, :]
tn_two_y = np.ones((tn_two_x.shape[0], 1))
tn_eight_x = tn_x[tn_eight_idx, :]
tn_eight_y = np.zeros((tn_eight_x.shape[0], 1))
tn_two_eight_x = np.vstack((tn_two_x, tn_eight_x))
tn_two_eight_y = np.vstack((tn_two_y, tn_eight_y))

# testing part
tt_two_idx = np.transpose(tt_y == 2)[0]
tt_eight_idx = np.transpose(tt_y == 8)[0]
tt_two_x = tt_x[tt_two_idx, :]
tt_two_y = np.ones((tt_two_x.shape[0], 1))
tt_eight_x = tt_x[tt_eight_idx, :]
tt_eight_y = np.zeros((tt_eight_x.shape[0], 1))
tt_two_eight_x = np.vstack((tt_two_x, tt_eight_x))
tt_two_eight_y = np.vstack((tt_two_y, tt_eight_y))

# hold out 10% train as validation
stn_two_eight_idx, vld_two_eight_idx = \
```

```
        list(ss.split(np.arange(tn_two_eight_x.shape[0])))[0]
stn_two_eight_x = tn_two_eight_x[stn_two_eight_idx, :]
stn_two_eight_y = tn_two_eight_y[stn_two_eight_idx, :]
vld_two_eight_x = tn_two_eight_x[vld_two_eight_idx, :]
vld_two_eight_y = tn_two_eight_y[vld_two_eight_idx, :]

# main model
two_eight_w, stn_loss, vld_loss, tt_loss, stn_acc, vld_acc, tt_acc =\
        LogisticRegression(stn_two_eight_x, stn_two_eight_y, \
        vld_two_eight_x, vld_two_eight_y, tt_two_eight_x, \
        tt_two_eight_y, lb1=0, lb2=0)

# plot loss
plt.figure()
plt.plot(stn_loss, label='stn-train')
plt.plot(vld_loss, label='validation')
plt.plot(tt_loss, label='test')
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.title('Loss Function for two_eight')
plt.legend(loc=1)
plt.savefig(img_folder + 'loss_two_eight.png')

# plot acc
plt.figure()
plt.plot(stn_acc, label='stn-train')
plt.plot(vld_acc, label='validation')
plt.plot(tt_acc, label='test')
plt.xlabel('Iterations')
plt.ylabel('Accuracy')
plt.title('Accuracy for two_eight')
plt.legend(loc=4)
plt.savefig(img_folder + 'acc_two_eight.png')


## Problem 4 (d): plot weights

# weight for two_three
plt.figure()
plt.imshow(two_three_w[0,:-1].reshape(28, 28))
plt.title('Weight for two_three')
plt.savefig(img_folder + 'weight_two_three.png')

# weight for two_eight
plt.figure()
plt.imshow(two_eight_w[0,:-1].reshape(28, 28))
plt.title('Weight for two_eight')
plt.savefig(img_folder + 'weight_two_eight.png')

# weight difference
plt.figure()
plt.imshow((two_eight_w[0,:-1]-two_three_w[0,:-1]).reshape(28, 28))
plt.title('Weight Difference')
plt.savefig(img_folder + 'weight_difference.png')


## Problem 5
lb_lst = [0.01, 0.001, 0.0001]

w_l1 = []
w_l2 = []
err_tt_l1 = []
err_tt_l2 = []

# Problem 5 (b)
```

```
plt.figure()
for lb in lb_lst:
    # l1 norm
    w, stn_loss, vld_loss, tt_loss, stn_acc, vld_acc, tt_acc =\
            LogisticRegression(stn_two_three_x, stn_two_three_y, \
            vld_two_three_x, vld_two_three_y, tt_two_three_x, \
            tt_two_three_y, lb1=lb, lb2=0)
    plt.plot(stn_acc, label='sub-train l1_' + str(lb))
    w_l1.append(w)
    err_tt_l1.append(1 - tt_acc[-1])

    # l2 norm
    w, stn_loss, vld_loss, tt_loss, stn_acc, vld_acc, tt_acc =\
            LogisticRegression(stn_two_three_x, stn_two_three_y, \
            vld_two_three_x, vld_two_three_y, tt_two_three_x, \
            tt_two_three_y, lb1=0, lb2=lb)
    plt.plot(stn_acc, label='sub-train l2_' + str(lb))
    w_l2.append(w)
    err_tt_l2.append(1 - tt_acc[-1])

plt.xlabel('Iterations')
plt.ylabel('Accuracy')
plt.title('Sub-train Accuracy for two_three_regularization')
plt.legend(loc=4)
plt.savefig(img_folder + 'acc_stn_two_three_regularization.png')

# Problem 5 (c)
plt.figure()
plt.plot(range(len(lb_lst)), [np.linalg.norm(w, 2) for w in w_l1], \
        marker='o', label='l1 norm')
plt.plot(range(len(lb_lst)), [np.linalg.norm(w, 2) for w in w_l2], \
        marker='o', label='l2 norm')
plt.xticks(range(len(lb_lst)), ['0.01', '0.001', '0.0001'])
plt.xlabel('Lambda')
plt.ylabel('Length')
plt.title('Length of w with different regularizations')
plt.legend(loc=4)
plt.savefig(img_folder + 'len_w_two_three_regularization.png')

# Problem 5 (d)
plt.figure()
plt.plot(lb_lst, err_tt_l1, label='test l1 norm')
plt.plot(lb_lst, err_tt_l2, label='test l2 norm')
plt.xscale('log')
plt.xlabel('Lambda (Log)')
plt.ylabel('Test Error')
plt.title('Test Error for two_three_regularization')
plt.legend(loc=4)
plt.savefig(img_folder + 'err_tt_two_three_regularization.png')

# Problem 5 (e)
for w, lb in zip(w_l1, lb_lst):
    plt.figure()
    plt.imshow(w[0,:-1].reshape(28, 28))
    plt.title('')
    plt.savefig(img_folder + 'weight_two_three_l1_' + str(lb) + '.png'
                                        )

for w, lb in zip(w_l2, lb_lst):
    plt.figure()
    plt.imshow(w[0,:-1].reshape(28, 28))
    plt.title('')
    plt.savefig(img_folder + 'weight_two_three_l2_' + str(lb) + '.png'
                                        )
```

Code Listing 2: Code for Softmax Regression

```
import os, struct
from array import array as pyarray
import numpy as np
import matplotlib.pyplot as plt

from sklearn import preprocessing
from sklearn import model_selection
from sklearn.model_selection import ShuffleSplit

tn_n = 20000
tt_n = 2000
ft_n = 784

INF = 10000000
MAX_ITER = 1000
eta = 0.001
eta_decay = 0.01
eps = 1e-10

img_folder = '../res/'

ss = ShuffleSplit(n_splits=1, test_size=0.1, random_state=0)


def load_mnist(dataset, digits=np.arange(10), path="../dat/"):
    """
    Loads MNIST files into 3D numpy arrays

    Adapted from: http://abel.ee.ucla.edu/cvxopt/_downloads/mnist.py
    """

    if dataset == "training":
        fname_img = os.path.join(path, 'train-images-idx3-ubyte')
        fname_lbl = os.path.join(path, 'train-labels-idx1-ubyte')
    elif dataset == "testing":
        fname_img = os.path.join(path, 't10k-images-idx3-ubyte')
        fname_lbl = os.path.join(path, 't10k-labels-idx1-ubyte')
    else:
        raise ValueError("dataset must be 'testing' or 'training'")

    flbl = open(fname_lbl, 'rb')
    magic_nr, size = struct.unpack(">II", flbl.read(8))
    lbl = pyarray("b", flbl.read())
    flbl.close()

    fimg = open(fname_img, 'rb')
    magic_nr, size, rows, cols = struct.unpack(">IIII", fimg.read(16))
    img = pyarray("B", fimg.read())
    fimg.close()

    ind = [ k for k in range(size) if lbl[k] in digits ]
    N = len(ind)

    images = np.zeros((N, rows, cols), dtype=np.uint8)
    labels = np.zeros((N, 1), dtype=np.int8)
    for i in range(len(ind)):
        images[i] = np.array(img[ ind[i]*rows*cols : \
                (ind[i]+1)*rows*cols ]).reshape((rows, cols))
        labels[i] = lbl[ind[i]]

    return images, labels
```

```
def calLOSS(w, x, y, p, lb):
    assert(x.shape[0] == y.shape[0])
    val = 0.0
    E_mat = y * np.log(p + eps)
    val = - E_mat.sum()

    val += lb * np.linalg.norm(w, 2)
    return val

def calACC(w, x, y, p):
    assert(x.shape[0] == y.shape[0] and y.shape[0] == p.shape[0])
    num=0
    for row in range(p.shape[0]):
        largest=0
        index=0
        for col in range(10):
            if p[row,col]>largest:
                index=col
                largest=p[row,col]
        if y[row,index]==1:
            num+=1
    return 1.0 / x.shape[0] * num


def predict(w, x):
    v = np.dot(x, np.transpose(w))
    v=np.matrix(np.exp(v))
    return np.array(v/(v.sum(1)+eps))

def SoftmaxRegression(sx, sy, vx, vy, tx, ty, lb):
    assert(sx.shape[0] == sy.shape[0])
    assert(vx.shape[0] == vy.shape[0])
    assert(tx.shape[0] == ty.shape[0])

    w = np.zeros((10, ft_n+1))
    bw = np.zeros((10, ft_n+1))
    sp = predict(w, sx);
    vld_v = INF
    cnt = 0

    # results
    stn_loss = []
    vld_loss = []
    tt_loss = []
    stn_acc = []
    vld_acc = []
    tt_acc = []

    for t in xrange(MAX_ITER):
        nw = np.dot((sy-sp).transpose(), sx) - lb * 2 * w

        n_eta = eta / (1 + t * eta_decay)
        w += n_eta * nw

        # predict
        sp = predict(w, sx)
        vp = predict(w, vx)
        tp = predict(w, tx)

        # calculate loss
        n_stn_v = calLOSS(w, sx, sy, sp, lb)
        n_vld_v = calLOSS(w, vx, vy, vp, lb)
        n_tt_v = calLOSS(w, tx, ty, tp, lb)
        stn_loss.append(n_stn_v)
        vld_loss.append(n_vld_v)
```

```
        tt_loss.append(n_tt_v)

        # calculate acc
        n_stn_a = calACC(w, sx, sy, sp)
        n_vld_a = calACC(w, vx, vy, vp)
        n_tt_a = calACC(w, tx, ty, tp)
        stn_acc.append(n_stn_a)
        vld_acc.append(n_vld_a)
        tt_acc.append(n_tt_a)

        # stop condition
        if n_vld_v >= vld_v:
            cnt += 1
        else:
            bw = np.copy(w)
            vld_v = n_vld_v
            cnt = 0

        if cnt == 3:
            print 'Stop at #' + str(t)
            return bw, stn_loss[:-3], vld_loss[:-3], tt_loss[:-3], \
                    stn_acc[:-3], vld_acc[:-3], tt_acc[:-3]

    print 'Maximum iterations are reached'
    return bw, stn_loss[:-3], vld_loss[:-3], tt_loss[:-3], \
            stn_acc[:-3], vld_acc[:-3], tt_acc[:-3]


## read dataset
tn_x, tn_y = load_mnist('training')
tn_x, tn_y = tn_x[:tn_n, :, :], tn_y[:tn_n, :]
tn_x = tn_x.reshape(tn_n, ft_n)
tn_x = np.hstack([tn_x, np.ones((tn_n, 1))])
tn_y_new = np.zeros((tn_n,10))
for i in range(tn_n):
    tn_y_new[i,tn_y[i,0]]=1.0

tt_x, tt_y = load_mnist('testing')
tt_x, tt_y = tt_x[:tt_n, :, :], tt_y[:tt_n, :]
tt_x = tt_x.reshape(tt_n, ft_n)
tt_x = np.hstack([tt_x, np.ones((tt_n, 1))])
tt_y_new = np.zeros((tt_n,10))
for i in range(tt_n):
    tt_y_new[i,tt_y[i,0]]=1.0

## preprocessing
tn_x = preprocessing.scale(tn_x)
tt_x = preprocessing.scale(tt_x)


# hold out 10% train as validation
stn_idx, vld_idx = \
        list(ss.split(np.arange(tn_x.shape[0])))[0]
stn_x = tn_x[stn_idx, :]
stn_y = tn_y_new[stn_idx, :]
vld_x = tn_x[vld_idx, :]
vld_y = tn_y_new[vld_idx, :]


# main model
w, stn_loss, vld_loss, tt_loss, stn_acc, vld_acc, tt_acc =\
      SoftmaxRegression(stn_x, stn_y, vld_x,\
      vld_y, tt_x, tt_y_new, 0.01)

# plot loss
```

```
plt.figure()
plt.plot(stn_loss, label='stn-train')
plt.plot(vld_loss, label='validation')
plt.plot(tt_loss, label='test')
plt.xlabel('Iterations')
plt.ylabel('Loss')
plt.title('Loss Function for 10-way Classification')
plt.legend(loc=1)
plt.savefig(img_folder + 'loss_10_way.png')

# plot acc
plt.figure()
plt.plot(stn_acc, label='stn-train')
plt.plot(vld_acc, label='validation')
plt.plot(tt_acc, label='test')
plt.xlabel('Iterations')
plt.ylabel('Accuracy')
plt.title('Accuracy for 10-way Classification')
plt.legend(loc=4)
plt.savefig(img_folder + 'acc_10_way.png')
```