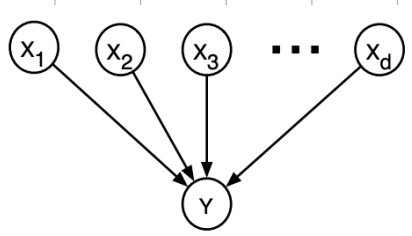
5.1 Gradient-based Learning



We consider the BN shown above with binary random variable $Y \in \{0,1\}$ and CPT: $P(Y=1|x_1,\dots,x_d) = g\left(\sum_{i=1}^{d} \omega_i x_i\right) = g(\vec{\omega}\cdot\vec{x})$.

Here, we assume $\omega_i \in \mathbb{R}$ are real-valued parameters to be estimated from data and that $g: \mathbb{R} \to [0,1]$ is a differentiable function that maps its real-valued argument to the unit interval.

We consider a data set of i.i.d. examples $\{(\vec{x}_t, y_t)\}_{t=1}^{T}$ where $\vec{x}_t = (x_{1t}, x_{2t}, \dots, x_{dt})$ denotes the observed vector of values from the t^{th} example for the network's inputs.

Let $p_t = P(Y=1 \mid \vec{x}_t)$

a) Show that the gradient of the conditional log-likelihood $\mathcal{L} = \sum_t \log P(y_t|\vec{x}_t)$ is $\frac{\partial \mathcal{L}}{\partial \omega_i} = \sum_{t=1}^{T} \left[\frac{g'(\vec{\omega}\cdot\vec{x}_t)}{p_t(1-p_t)}\right](y_t - p_t)x_{it}$.

$$\mathcal{L} = \sum_t \log P(y_t|\vec{x}_t)$$

$$= \sum_{t=1}^{T} \log P(y_t|\vec{x}_t)$$

$$= \sum_{t=1}^{T} \log\left((p_t)^{y_t}(1-p_t)^{1-y_t}\right)$$

$$= \sum_{t=1}^{T} \left(y_t \log(p_t) + (1-y_t)\log(1-p_t)\right)$$

$$\Rightarrow \frac{\partial \mathcal{L}}{\partial \omega_i} = \sum_{t=1}^{T} \left(\frac{y_t}{p_t} g'(\vec{\omega}\cdot\vec{x})x_{it} + (1-y_t)\frac{-1}{1-p_t} g'(\vec{\omega}\cdot\vec{x})x_{it} \right)$$

$$= \sum_{t=1}^{T} \left(\frac{y_t}{p_t} + (y_t - 1)\cdot\frac{1}{1-p_t} \right) g'(\vec{\omega}\cdot\vec{x})\, x_{it}$$

$$= \sum_{t=1}^{T} \left(\frac{y_t(1-p_t) + (y_t-1)p_t}{p_t(1-p_t)} \right) g'(\vec{\omega}\cdot\vec{x})\, x_{it}$$

$$= \sum_{t=1}^{T} \left(\frac{g'(\vec{\omega}\cdot\vec{x})}{p_t(1-p_t)} \right)(y_t - p_t)\, x_{it}$$ ■

b) Show that the general result in a) reduces to the result in lecture when $g(z) = \frac{1}{1+e^{-z}}$ is the Sigmoid function.

$$\frac{\partial \mathcal{L}}{\partial \omega_i} = \sum_{t=1}^{T} \left(\frac{g'(\vec{\omega}\cdot\vec{x})}{p_t(1-p_t)} \right)(y_t - p_t)\, x_{it}$$

Sigmoid property: $g'(z) = g(z)g(-z)$

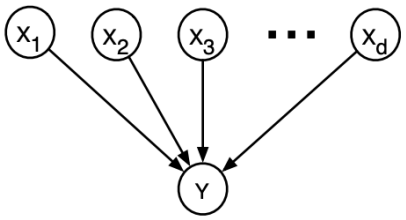
$$= \sum_{t=1}^{T} \frac{g(\vec{\omega}\cdot\vec{x})\, g(-\vec{\omega}\cdot\vec{x})\,(y_t - p_t)}{p_t(1-p_t)}\, x_{it}$$

$$= \sum_{t=1}^{T} \frac{\cancel{p_t(1-p_t)}\,(y_t - p_t)}{\cancel{p_t(1-p_t)}}\, x_{it}$$

$p_t = g(\vec{\omega}\cdot\vec{x})$

$$= \sum_{t=1}^{T} x_{it}\left[y_t - g(\vec{\omega}\cdot\vec{x})\right]$$ ■

5.2 Multinomial logistic regression



A simple generalization of logistic regression is to predict a discrete (but non-binary) label $y \in \{1, 2, \ldots, k\}$ from a real-valued vector $\vec{x} \in \mathcal{R}^d$. Here k is the number of classes. For the belief network shown above, consider the following parameterized conditional probability table (CPT):

$$P(Y=i|\vec{X}=\vec{x}) \;=\; \frac{e^{\vec{w}_i \cdot \vec{x}}}{\sum_{j=1}^{k} e^{\vec{w}_j \cdot \vec{x}}}.$$

The parameters of this CPT are the weight vectors $\vec{w}_i$ which must be learned for each possible label. The denominator normalizes the distribution so that the elements of the CPT sum to one.

Consider a data set of T examples $\{(\vec{x}_t, y_t)\}_{t=1}^{T}$, which you can assume to be identically, independently distributed (i.i.d.). As shorthand, let $y_{it} \in \{0, 1\}$ denote the $k \times T$ matrix of target assignments with elements

$$y_{it} \;=\; \begin{cases} 1 & \text{if} \quad y_t = i, \\ 0 & \text{otherwise.} \end{cases}$$

Also, let $p_{it} \in [0, 1]$ denote the conditional probability that the model classifies the tth example by the ith possible label:

$$p_{it} \;=\; \frac{e^{\vec{w}_i \cdot \vec{x}_t}}{\sum_{j=1}^{k} e^{\vec{w}_j \cdot \vec{x}_t}}.$$

The weights can be obtained by maximum likelihood estimation using gradient ascent.

Show that the gradient of conditional log-likelihood $\mathcal{L} = \sum_t \log P(y_t|\vec{x}_t)$ is given by $\frac{\partial \mathcal{L}}{\partial \vec{w}_i} = \sum_t (y_{it} - p_{it})\vec{x}_t$

$$\begin{aligned}
\mathcal{L} &= \sum_{t=1}^{T} \log P(y_t|\vec{x}_t) \\
&= \sum_{t=1}^{T} \log\left(\prod_{i=1}^{k} (p_{it})^{y_{it}}\right) \\
&= \sum_{t=1}^{T} \sum_{i=1}^{k} y_{it} \log p_{it} \\
&= \sum_{t=1}^{T} \sum_{i=1}^{k} y_{it} \log\left[\frac{e^{\vec{w}_i \cdot \vec{x}_t}}{\sum_{j=1}^{k} e^{\vec{w}_j \cdot \vec{x}_t}}\right] \\
&= \sum_{t=1}^{T} \sum_{i=1}^{k} \left(y_{it} \log(e^{\vec{w}_i \cdot \vec{x}_t}) - y_{it} \log\left(\sum_{j=1}^{k} e^{\vec{w}_j \cdot \vec{x}_t}\right)\right) \\
&= \sum_{t=1}^{T} \left[\sum_{i=1}^{k} y_{it} \log(e^{\vec{w}_i \cdot \vec{x}_t}) - \underbrace{\sum_{i=1}^{k} y_{it}}_{=1} \cdot \log\left(\sum_{j=1}^{k} e^{\vec{w}_j \cdot \vec{x}_t}\right)\right] \\
&= \sum_{t=1}^{T} \left[\sum_{i=1}^{k} y_{it} \log(e^{\vec{w}_i \cdot \vec{x}_t}) - \log\left(\sum_{j=1}^{k} e^{\vec{w}_j \cdot \vec{x}_t}\right)\right]
\end{aligned}$$

From here, we can do the gradient

$$\begin{aligned}
\frac{\partial \mathcal{L}}{\partial \vec{w}_i} &= \sum_{t=1}^{T} \frac{y_{it}\, e^{\vec{w}_i \cdot \vec{x}_t}\, \vec{x}_t}{e^{\vec{w}_i \cdot \vec{x}_t}} - \underbrace{\frac{e^{\vec{w}_i \cdot \vec{x}_t}}{\sum_{j=1}^{k} e^{\vec{w}_j \cdot \vec{x}_t}}}_{p_{it}} \vec{x}_t \\
&= \sum_{t=1}^{T} (y_{it} - p_{it})\vec{x}_t
\end{aligned}$$

■

5.3 Convergence of gradient descent

One way to gain intuition for gradient descent is to analyze its behavior in simple settings. For a 1D function $g(x)$ over the real line, gradient descent takes the form:

$$x_{new} = x_o - \eta g'(x_o)$$

a) We consider minimizing the function $g(x) = \frac{\alpha}{2}(x - x_*)^2$ by gradient descent, where $\alpha > 0$. In this case, we know that the minimum occurs at $x = x_*$; our goal is to analyze the rate of convergence to this minimum.

Derive an expression for the error $\epsilon_n = x_n - x_*$ at the n^{th} iteration in terms of the initial error ϵ_0 and the step size $\eta > 0$.

We know that $g'(x) = \alpha(x - x_*)$. We also have $\epsilon_0 = x_0 - x_*$.

The first iteration will then be

$$x_1 = x_0 - \eta\alpha(x_0 - x_*) = x_0 - \eta\alpha\epsilon_0 = (1-\eta\alpha)\epsilon_0 - x_* \Rightarrow \epsilon_1 = x_1 - x_* = (1-\eta\alpha)\epsilon_0$$

$$x_2 = x_1 - \eta\alpha(x_1 - x_*) = x_0 - \eta\alpha\epsilon_0 - \eta\alpha(x_0 - \eta\alpha\epsilon_0 - x_*) = \epsilon_0 - x_* - \eta\alpha\epsilon_0 - \eta\alpha(\epsilon_0 - \eta\alpha\epsilon_0) = (1-\eta\alpha)\epsilon_0 - \eta\alpha(1-\eta\alpha)\epsilon_0 - x_* = (1-\eta\alpha)(1-\eta\alpha)\epsilon_0 - x_*$$

$$\Rightarrow \epsilon_2 = x_2 - x_* = (1-\eta\alpha)^2\epsilon_0$$

$$x_3 = x_2 - \eta\alpha(\underbrace{x_2 - x_*}_{\epsilon_2}) = (1-\eta\alpha)(1-\eta\alpha)\epsilon_0 - x_* - \eta\alpha(1-\eta\alpha)^2\epsilon_0 = (1-\eta\alpha)(1-\eta\alpha)^2\epsilon_0 - x_* = (1-\eta\alpha)^3\epsilon_0 - x_*$$

$$\Rightarrow \epsilon_3 = x_3 - x_* = (1-\eta\alpha)^3\epsilon_0$$

We see that we have a pattern which yields

$$\underline{\epsilon_n = (1-\eta\alpha)^n\epsilon_0}$$

b) For what values of η does the update rule converge to the minimum at x_*? What η leads to fastest convergence, and how is it related to $g''(x)$?

We need that $\epsilon_n = 0$ after n iterations for convergence, as there will be no error at that point.

Thus, the update rule converges when $|1-\eta\alpha| < 1$, as this will tend $\epsilon_n \to 0$.

For $\eta\alpha > 1$, we have $-1 < 1-\eta\alpha \Rightarrow \eta < \frac{2}{\alpha} = \frac{2}{g''(x)}$.

For $\eta\alpha < 1$, we have $1-\eta\alpha < 1 \Rightarrow \eta > 0$.

$\Rightarrow \underline{0 < \eta < \frac{2}{g''(x)} \text{ for convergence}}$

$\epsilon_n = (1-\eta\alpha)^n\epsilon_0 = 0$

$\Rightarrow 1 - \eta\alpha = 0 \Leftrightarrow \eta\alpha = 1$

Thus, $\underline{\eta = \frac{1}{\alpha}}$ is the step size needed for fastest convergence, and $n = 1$ will yield convergence. (1 iteration)

We observe that $\eta = \frac{1}{\alpha} = \frac{1}{g''(x)}$.

In practice, the gradient descent (GD) learning rule is often modified to dampen oscillations at the end of the learning procedure.

A common variant of GD involves adding a so-called momentum term:

$$\vec{x}_{n+1} = \vec{x}_n - \eta \nabla g + \beta(\vec{x}_n - \vec{x}_{n-1}) \quad \text{where} \quad \beta > 0$$

In one dimension, the rule simplifies to

$$x_{n+1} = x_n - \eta g'(x_n) + \beta(x_n - x_{n-1}).$$

c) Consider minimizing the quadratic function in a) by gradient descent with a momentum term. Show that $\epsilon_{n+1} = (1 - a\eta + \beta)\epsilon_n - \beta\epsilon_{n-1}$.

We have

$$\begin{aligned} x_{n+1} &= x_n - \eta g'(x_n) + \beta(x_n - x_{n-1}) \\ &= x_n - \eta a(x_n - x_*) + \beta(x_n - x_{n-1}) \\ &= (1 - \eta a + \beta)x_n + \eta a x_* - \beta x_{n-1} \end{aligned}$$

We then remember that $\epsilon_{n+1} = x_{n+1} - x_*$, $\epsilon_n = x_n - x_*$ and $\epsilon_{n-1} = x_{n-1} - x_*$ to get

$$\epsilon_{n+1} + x_* = (1 - \eta a + \beta)(\epsilon_n + x_*) + \eta a x_* - \beta(\epsilon_{n-1} + x_*)$$

$$\epsilon_{n+1} + x_* = (1 - \eta a + \beta)\epsilon_n + (1 - \eta a + \beta)x_* + \eta a x_* - \beta\epsilon_{n-1} - \beta x_*$$

$$\epsilon_{n+1} + \cancel{x_*} = (1 - \eta a + \beta)\epsilon_n - \beta\epsilon_{n-1} + \cancel{x_*}$$

$$\Rightarrow \epsilon_{n+1} = (1 - \eta a + \beta)\epsilon_n - \beta\epsilon_{n-1}$$

■

d) We suppose that $\alpha = 1$, $\eta = \frac{4}{9}$ and $\beta = \frac{1}{9}$.

Show that one solution to the recursion in c) is given by $\varepsilon_n = \lambda^n \varepsilon_0$ where λ is a numerical constant to be determined.

$$\varepsilon_{n+1} = (1 - \eta\alpha + \beta)\varepsilon_n - \beta\varepsilon_{n-1}$$

$\varepsilon_n = \lambda^n \varepsilon_0$

$$= (1 - \eta\alpha + \beta)\lambda^n \varepsilon_0 - \beta\lambda^{n-1}\varepsilon_0$$

Insert values

$$= \frac{2}{3}\lambda^n \varepsilon_0 - \frac{1}{9}\lambda^{n-1}\varepsilon_0 = \lambda^{n+1}\varepsilon_0$$

$$\Rightarrow \lambda^{n+1} - \frac{2}{3}\lambda^n + \frac{1}{9}\lambda^{n-1} = 0$$

For $n=1$, we get

$$\lambda^2 - \frac{2}{3}\lambda + \frac{1}{9} = 0$$

$$\Rightarrow \lambda = \frac{1}{3} \text{ which is a constant.}$$

Thus, we get $\varepsilon_n = \left(\frac{1}{3}\right)^n \varepsilon_0$ as rate of convergence with $\lambda = \frac{1}{3}$.

How does this rate of convergence compare to that of GD with $\eta = \frac{4}{9}$ but $\beta = 0$?

In this case, we have

$$\varepsilon_{n+1} = \frac{5}{9}\lambda^n \varepsilon_0 = \lambda^{n+1}\varepsilon_0$$

For $n=1$, we get

$$\lambda^2 - \frac{5}{9}\lambda = 0 \Leftrightarrow \lambda(\lambda - \frac{5}{9}) = 0$$

As $\lambda = 0$ is not a solution, we have the rate of convergence

$$\varepsilon_n = \left(\frac{5}{9}\right)^n \varepsilon_0.$$

Because $\frac{5}{9} > \frac{1}{3}$, this will converge slower than with a momentum term.

5.4 Newton's method

One way to gain intuition for Newton's method is to analyze its behavior in simple settings. For a twice-differentiable function $g(x)$ over the real line, Newton's method takes the form

$$x_{n+1} = x_n - \frac{g'(x_n)}{g''(x_n)}$$

a) We consider the polynomial function $g(x) = (x - x_*)^{2p}$ for $p > 0$, whose minimum occurs at $x = x_*$.

Suppose that Newton's method is used to minimize this function, starting from some initial estimate x_0.

Derive an expression for the error $\epsilon_n = |x_n - x_*|$ at the n^{th} iteration in terms of the initial error ϵ_0.

$$g'(x) = 2p(x - x_*)^{2p-1}$$

$$g''(x) = 2p(2p-1)(x - x_*)^{2p-2}$$

We use $x_{n+1} = x_n - \frac{g'(x_n)}{g''(x_n)}$ and that $\epsilon_n = |x_n - x_*|$ and $\epsilon_0 = |x_0 - x_*|$, to get

$$\epsilon_n = \left| x_{n-1} - \underbrace{\frac{x_{n-1} - x_*}{(2p-1)}}_{\frac{g'(x)}{g''(x)}} - x_* \right|$$

$$= \left| \underbrace{\left(1 - \frac{1}{2p-1}\right)}_{>0 \ \forall p} (x_{n-1} - x_*) \right|$$

$$= \left(1 - \frac{1}{2p-1}\right) |x_{n-1} - x_*|$$

$$= \left(1 - \frac{1}{2p-1}\right) \epsilon_{n-1}$$

In the same way as in the first problem, every error term will base itself in the constant and the last term

Thus, we will get

$$\epsilon_n = \left[1 - \frac{1}{2p-1}\right]^n \epsilon_0$$

b) How many iterations of Newton's method are required to reduce the initial error by $\delta < 1$ such that $\epsilon_n \leq \delta \epsilon_0$? Show that $n \geq (2p-1)\log(\frac{1}{\delta})$ iters are sufficient.

We have $\epsilon_n = \left(1 - \frac{1}{2p-1}\right)^n \epsilon_0$ and we have $\epsilon_n \leq \delta \epsilon_0$ which implies

$$\left(1 - \frac{1}{2p-1}\right)^n \leq \delta$$

$$\log\left(1 - \frac{1}{2p-1}\right)^n \leq \log \delta$$

$$n \log\left(1 - \frac{1}{2p-1}\right) \leq \log \delta$$

$$n \log\left(1 - \frac{1}{2p-1}\right) \leq \log \delta$$

$$-n \log\left(1 - \frac{1}{2p-1}\right) \geq \log \frac{1}{\delta}$$

$$n \geq \frac{\log \frac{1}{\delta}}{-\log\left(1 - \frac{1}{2p-1}\right)}$$ which is the number of iterations needed.

We use the hint $\log z \leq z - 1 \quad \forall z > 0$

$$-\log\left(1 - \frac{1}{2p-1}\right) \geq -\left(1 - \frac{1}{2p-1}\right) + 1 = \frac{1}{2p-1}$$

Thus, we get that

$n \geq \log \frac{1}{\delta} (2p-1)$ iterations are sufficient.

c) Consider $g(x) = x_* \log\left(\frac{x_*}{x}\right) - x_* + x$, where $x_* > 0$. Show that the minimum occurs at $x = x_*$ and sketch the function in region $|x - x_*| < x_*$.

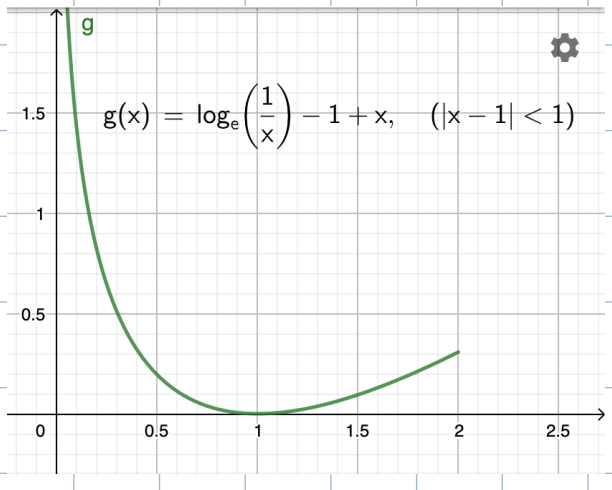
$$g(x) = x_* \log\left(\frac{x_*}{x}\right) - x_* + x$$

$$g'(x) = x_* \frac{x}{x_*} \cdot \frac{-x_*}{x^2} + 1$$

$$= -\frac{x_*}{x} + 1 = 0 \quad \text{for the minimum}$$

$$\Rightarrow x_{min} = x_* \quad \text{QED}$$

We let $x_* = 1$ for the sketch, and sketch it with Geogebra.



d) Consider minimizing $g(x)$ by Newton's method. Derive an expression for the relative error $\rho_n = \frac{x_n - x_*}{x_*}$ at the n^{th} iteration in terms of the initial relative error ρ_0.

We have $g'(x) = -\frac{x_*}{x} + 1$ and $g''(x) = \frac{x_*}{x^2}$

Newton

$$\Rightarrow x_{n+1} = x_n - \frac{-\frac{x_*}{x_n} + 1}{\frac{x_*}{x_n^2}}$$

$$= x_n - \frac{x_n^2}{x_*} + x_n$$

$$= 2x_n - \frac{x_n^2}{x_*} \qquad -\frac{1}{x_*}(x_n - x_*)^2 = -\left(x_n^2 - 2x_n x_* + x_*^2\right)\frac{1}{x_n}$$

$$= -\frac{1}{x_*}(x_n - x_*)^2 + x_*$$

$$\rho_n = \frac{x_n - x_*}{x_*} = \frac{-\frac{1}{x_*}(x_{n-1} - x_*)^2 + x_* - x_*}{x_*} = -\left(\frac{x_{n-1} - x_*}{x_*}\right)^2$$

$$= -(\rho_{n-1})^2$$

$$\underline{\rho_n = -\rho_0^{2^n}}$$

For what range of initial values does Newton's method converge to the correct answer?

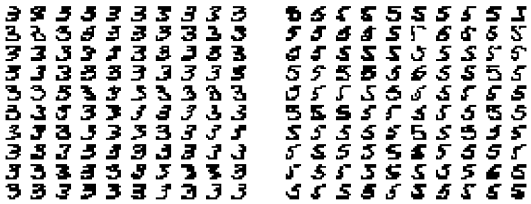
ρ_n will converge when $|\rho_0| < 1$

$$\left|\frac{x_0 - x_*}{x_*}\right| < 1$$

$\Rightarrow |x_0| < 2|x_*|$ will make Newton's method converge.

5.5 Handwritten digit classification

In this problem, you will use logistic regression to classify images of handwritten digits. From the course web site, download the files `train3.txt`, `test3.txt`, `train5.txt`, and `test5.txt`. These files contain data for binary images of handwritten digits. Each image is an 8x8 bitmap represented in the files by one line of text. Some of the examples are shown in the following figure.

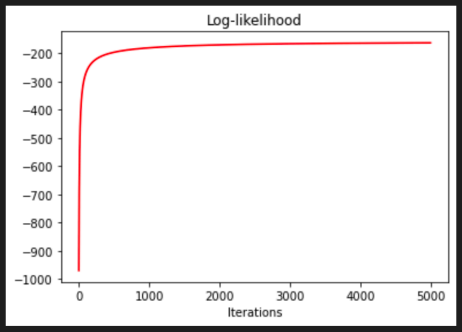


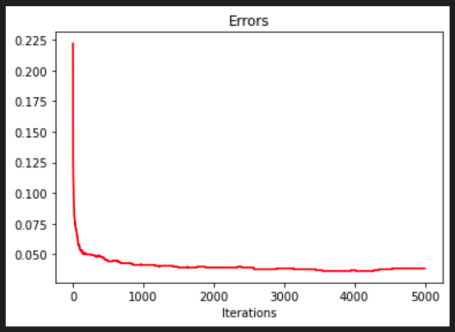
a) Training

Perform a logistic regression (using gradient ascent or Newton's method) on the images in files `train3.txt` and `train5.txt`. Indicate clearly the algorithm used, and provide evidence that it has converged (or nearly converged) by plotting or printing out the log-likelihood on several iterations of the algorithm, as well as the percent error rate on the images in these files. Also, print out the 64 elements of your solution for the weight vector as an 8x8 matrix.

In this task, I will be using the Gradient Ascent method of logistic regression to classify handwritten digits from the MNIST database.

Log-likelihood and error output





Weight matrix

−0.8797974	−1.4877361	−1.1609024	−1.2031431	−0.711716	−0.8885907	0.8056814	1.7719741
0.0007422	−0.1388434	0.2383485	−0.0676125	−0.4321559	0.7737291	−1.3078541	−1.3306284
3.5719539	1.3666871	1.4310638	0.1802701	0.733658	−2.0656259	−2.4048238	−2.5712567
0.7751998	0.3807532	0.6289521	−0.3051188	−0.4752028	−2.3491697	0.4132579	−0.0314203
0.521081	1.1247831	0.0576895	−0.3606556	−0.639655	−0.1502495	−0.4792577	−0.2629605
1.1833087	−0.2161415	−0.3391773	−0.1164656	0.0660542	−0.9162373	0.8473631	−1.5487284
1.4700634	−0.6591165	1.299139	0.5986168	0.4468133	−0.348847	0.1911971	−1.2810957
0.5834873	0.2840508	0.8985841	1.9238005	0.4840218	0.6502586	0.6400514	−0.4983526

b) Testing

Use the model learned in part (a) to label the images in the files `test3.txt` and `test5.txt`. Report your percent error rate on these images.

```
Error rate on test3 images:
0.065
Error rate on test5 images:
0.055
```

c) Source code

Turn in a print-out of your source code. Once again, you may program in the language of your choice. You should write your own routines for computing the model's log-likelihood, gradient, and/or Hessian, as well as for updating its weight vector.

handwritten_digit

November 3, 2021

1 Problem 5 - Handwritten digit classification

a) Perform a logistic regression (using gradient ascent or Newton's method) on the images in files ***train3.txt*** and ***train5.txt***. Indicate clearly the algorithm used, and provide evidence that it has converged (or nearly converged) by plotting or printing out the log-likelihood on several iterations of the algorithm, as well as the percent error rate on the images in these files. Also, print out the 64 elements of your solution for the weight vector as an 8x8 matrix.

1.0.1 Part a

1.0.2 In this task, I will be using the Gradient Ascent method of logistic regression to classify handwritten digits from the MNIST database.

[39]:
```
import numpy as np
import matplotlib.pyplot as plt
from pandas import *
```

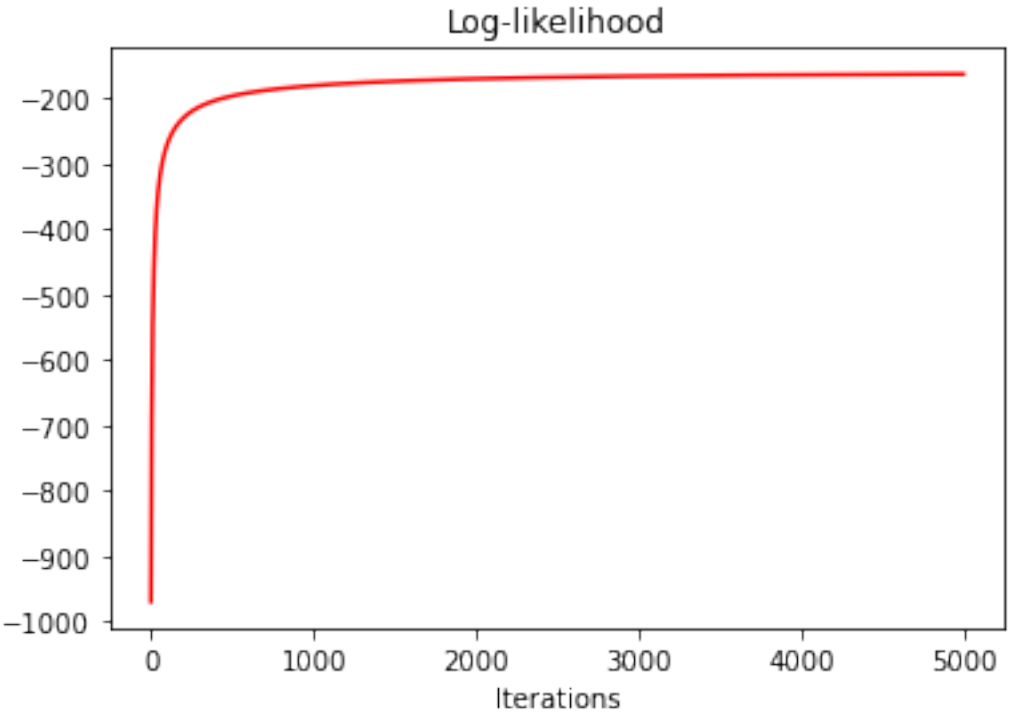
[40]:
```
# Load files and concatenate train and test files together
train3 = np.loadtxt('train3.txt', dtype=int)
train5 = np.loadtxt('train5.txt', dtype=int)
test3 = np.loadtxt('test3.txt', dtype=int)
test5 = np.loadtxt('test5.txt', dtype=int)

train = np.concatenate((train3, train5), axis=0)
test = np.concatenate((test3, test5), axis=0)

train_labels = [0] * train3.shape[0] + [1] * train5.shape[0]
test_labels = [0] * test3.shape[0] + [1] * test5.shape[0]
```

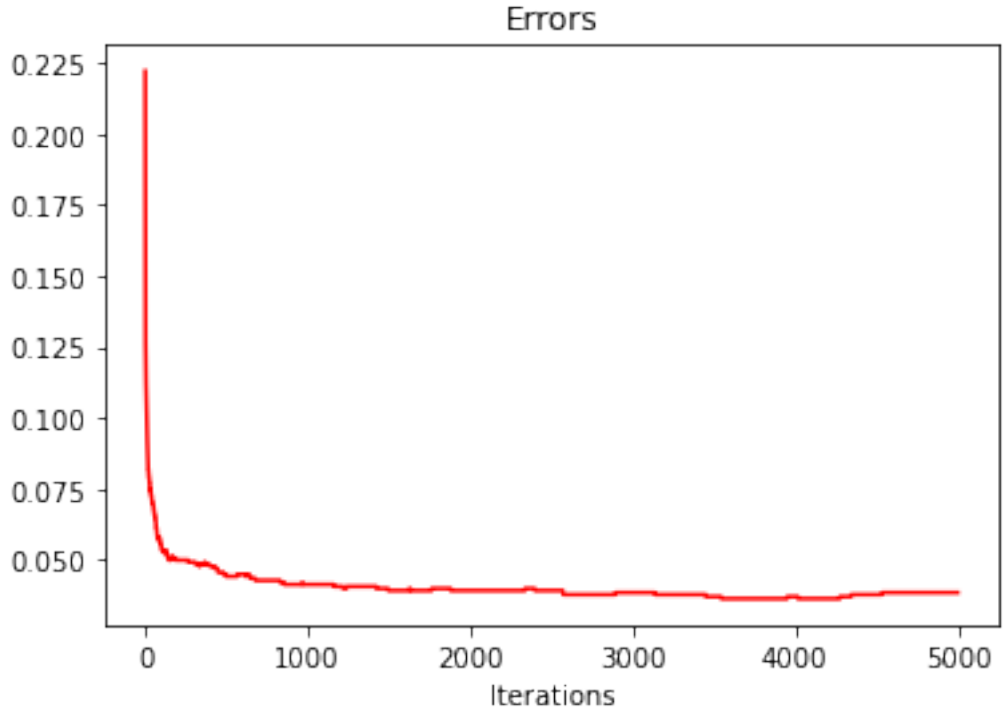
[41]:
```
def sigmoid(x,w):
    return (1.0/(1+np.exp(-np.dot(w,x))))

def gradient(x,y,w):
    return(np.multiply((y - sigmoid(w, x)), x))

def loglikelihood(x,y,w):
    return (y * np.log(sigmoid(w,x)) + (1-y) * np.log(1-sigmoid(w,x)))
```

[42]:
```
def error_prediction(data, labels, w):
    correct = 0
    for t in range(data.shape[0]):
        sig = sigmoid(w, data[t])
        if (labels[t]==1 and sig>=0.5) or (labels[t]==0 and sig<0.5):
            correct += 1
    error = (data.shape[0] - correct)*1.0 / data.shape[0]
    return error

def gradient_ascent(data, labels, steps):
    alpha = 0.0002
    weights = np.zeros(data.shape[1])
    likelihood_weights = []
    errors = []

    for i in range(steps):
        lw_sum = 0
        grad = 0
        for j in range(data.shape[0]):
            lw_sum += loglikelihood(data[j],labels[j],weights)
            grad += gradient(data[j],labels[j],weights)
        weights = weights + alpha * grad
        likelihood_weights.append(lw_sum)
        errors.append(error_prediction(data,labels,weights))
    return weights, likelihood_weights, errors
```

Log-likelihood and error output

[43]:
```
steps = 5000
weights, likelihood_weights, errors = gradient_ascent(train, train_labels, steps)
plt.plot(likelihood_weights, 'r-')
plt.xlabel("Iterations")
plt.title("Log-likelihood")
plt.show()
plt.plot(errors, 'r-')
plt.xlabel("Iterations")
plt.title("Errors")
plt.show()
```


Log-likelihood
−200
−300
−400
−500
−600
−700
−800
−900
−1000
0
1000
2000
3000
4000
5000
Iterations


Errors
0.225
0.200
0.175
0.150
0.125
0.100
0.075
0.050
0
1000
2000
3000
4000
5000
Iterations

Weight matrix

[44]:
```
print('\n'.join(['\t'.join([str(round(cell,7)) for cell in row]) for row in␣
 ↪weights.reshape([8,8])]))
```

```
-0.8797974      -1.4877361      -1.1609024      -1.2031431      -0.711716
-0.8885907      0.8056814       1.7719741
0.0007422       -0.1388434      0.2383485       -0.0676125      -0.4321559
0.7737291       -1.3078541      -1.3306284
3.5719539       1.3666871       1.4310638       0.1802701       0.733658
-2.0656259      -2.4048238      -2.5712567
0.7751998       0.3807532       0.6289521       -0.3051188      -0.4752028
-2.3491697      0.4132579       -0.0314203
0.521081        1.1247831       0.0576895       -0.3606556      -0.639655
-0.1502495      -0.4792577      -0.2629605
1.1833087       -0.2161415      -0.3391773      -0.1164656      0.0660542
-0.9162373      0.8473631       -1.5487284
1.4700634       -0.6591165      1.299139        0.5986168       0.4468133
-0.348847       0.1911971       -1.2810957
0.5834873       0.2840508       0.8985841       1.9238005       0.4840218
0.6502586       0.6400514       -0.4983526
```

1.0.3 Part b

[45]:
```
print("Error rate on test3 images:")
print(error_prediction(test3, [0] * test3.shape[0], weights))

print("Error rate on test5 images:")
print(error_prediction(test5, [1] * test5.shape[0], weights))
```

```
Error rate on test3 images:
0.065
Error rate on test5 images:
0.055
```