Margot Wagner
A53279875
CSE 250a HW5
Due: 11/10/20

5.1 Gradient-based Learning

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

$$= \sum_t \log \left[P(Y=1|x_{..})^{y_t} (1 - P(Y=1|x_{..}))^{1-y_t} \right]$$

$$= \sum_{t=1}^{T} \left[y_t \log f(\vec{w}\cdot\vec{x}_t) + (1-y_t) \log (1 - f(\vec{w}\cdot\vec{x}_t)) \right]$$

$$\frac{\partial \mathcal{L}}{\partial w_i} = \sum_t \left[y_t \frac{1}{f(\vec{w}\cdot\vec{x}_t)} f'(\vec{w}\cdot\vec{x}_t)\, x_{it} + (1-y_t) \frac{1}{1 - f(\vec{w}\cdot\vec{x}_t)} (-f'(\vec{w}\cdot\vec{x}_t))(x_{it}) \right] \qquad p_t = f(\vec{w}\cdot\vec{x}_t)$$

$$= \sum_t \left[y_t \frac{f'(\vec{w}\cdot\vec{x}_t)}{p_t} x_{it} + (1-y_t)\frac{(-f'(\vec{w}\cdot\vec{x}_t))}{(1-p_t)} x_{it} \right]$$ mult to get denom to $p_t(1-p_t)$

$$= \sum_t \left[\frac{(1-p_t) y_t f'(\vec{w}\cdot\vec{x}_t) - (1-y_t) f'(\vec{w}\cdot\vec{x}_t) p_t}{p_t(1-p_t)} x_{it} \right]$$

$$= \sum_t \frac{f'(\vec{w}\cdot\vec{x}_t)}{p_t(1-p_t)} \underbrace{\left[(1-p_t) y_t - (1-y_t) p_t \right]} x_{it}$$ simplify

$$y_t - \cancel{y_t p_t} - p_t + \cancel{y_t p_t} = y_t - p_t$$

$$\boxed{\frac{\partial \mathcal{L}}{\partial w_i} = \sum_t \left[\frac{f'(\vec{w}\cdot\vec{x}_t)}{p_t(1-p_t)} \right] (y_t - p_t)\, x_{it}}$$

(b) $$\frac{\partial \mathcal{L}}{\partial w_i} = \sum_t \left[\frac{f'(\vec{w}\cdot\vec{x}_t)}{p_t(1-p_t)} \right] (y_t - p_t)\, x_{it}$$

$$f(z) = \sigma(z) = \frac{1}{1+e^{-z}}$$

$$f'(z) = \sigma'(z) = \sigma(z)\sigma(-z)$$

$$f'(\vec{w}\cdot\vec{x}_t) = f(\vec{w}\cdot\vec{x}_t) f(-\vec{w}\cdot\vec{x}_t)$$

$$\sigma(-z) = 1 - \sigma(z) \Rightarrow f(-\vec{w}\cdot\vec{x}_t) = 1 - f(\vec{w}\cdot\vec{x}_t)$$

$$\therefore \; f'(\vec{w}\cdot\vec{x}_t) = f(\vec{w}\cdot\vec{x}_t)(1 - f(\vec{w}\cdot\vec{x}_t)) = p_t(1-p_t)$$

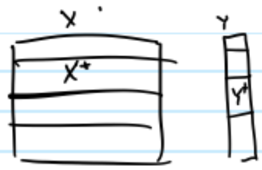
$$\frac{\partial \mathcal{L}}{\partial w_i} = \sum_t \underbrace{\frac{\cancel{p_t(1-p_t)}}{\cancel{p_t(1-p_t)}}}_{1} (y_t - p_t)\, x_{it}$$

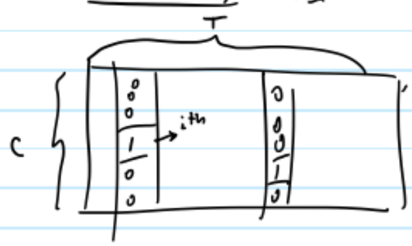
$$= \sum_t [y_t - p_t]\, x_{it}$$

$$\boxed{\frac{\partial \mathcal{L}}{\partial w_i} = \sum_t x_{it} \left[y_t - \sigma(\vec{w}\cdot\vec{x}_t) \right]}$$

5.2 Multinomial Logistic Regression

$$P(Y=i \mid \vec{X}=\vec{x}) = \frac{e^{\vec{w}_i \cdot \vec{x}}}{\sum_{j=1}^{c} e^{\vec{w}_j \cdot \vec{x}}} = p_{it}$$





c outcome possibilities

one-hot encoding of y_t

y^t has c possible vals each w/ a conditional prob

$$\mathcal{L} = \sum_t \log P(y_t \mid \vec{X}_t)$$

$$P(y_t \mid \vec{x}_t) = \frac{\prod_{i=1}^{c} (e^{\vec{w}_i \cdot \vec{x}})^{y_{it}}}{\sum_{j=1}^{c} e^{\vec{w}_j \cdot \vec{x}}}$$

$$\mathcal{L} = \sum_t \log \left[\frac{\prod_{i=1}^{c} (e^{\vec{w}_i \cdot \vec{x}})^{y_{it}}}{\sum_{j=1}^{c} e^{\vec{w}_j \cdot \vec{x}}} \right]$$

$\log\left(\frac{x}{y}\right) = \log x - \log y$

$$= \sum_t \left[\log\left(\prod_{i=1}^{c} (e^{\vec{w}_i \cdot \vec{x}})^{y_{it}} \right) - \log\left(\sum_{j=1}^{c} e^{\vec{w}_j \cdot \vec{x}} \right) \right]$$

$\log a^b = b \log a$

$$= \sum_t \left[y^{it} \log\left(\prod_{i=1}^{c} e^{\vec{w}_i \cdot \vec{x}} \right) - \log\left(\sum_{j=1}^{c} e^{\vec{w}_j \cdot \vec{x}} \right) \right]$$

$$\frac{\partial \mathcal{L}}{\partial w_\alpha} = \sum_t \left[y_{it} \frac{1}{\cancel{e^{\vec{w}_i \cdot \vec{x}}}} \vec{X}_t \cancel{e^{\vec{w}_i \cdot \vec{X}_t}} - \frac{1}{\sum_{j=1}^{c} e^{\vec{w}_j \cdot \vec{x}}} \vec{X}_t e^{\vec{w}_i \cdot \vec{x}} \right]$$

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

$$= \sum_t (y_{it} - p_{it}) \vec{X}_t$$

5.3 Convergence of gradient descent

$$x_{n+1} = x_n - \eta f'(x_n)$$

(a) $x_{n+1} = x_n - \eta f'(x_n)$

$\therefore \; x_n = x_{n-1} - \eta f'(x_{n-1})$

$f(x) = \frac{\alpha}{2}(x - x_*)^2$

$f'(x) = \alpha(x - x_*)$

$f'(x_{n-1}) = \alpha(x_{n-1} - x_*)$ subst

$x_n = x_{n-1} - \eta\alpha(x_{n-1} - x_*)$

$= x_{n-1} - \eta\alpha x_{n-1} + \eta\alpha x_*$

$\varepsilon_n = x_n - x_*$ subst

$= x_{n-1} - \eta\alpha x_{n-1} - x_* + \eta\alpha x_*$

$= (1 - \eta\alpha)(x_{n-1} - x_*)$

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

$\therefore$

$$\varepsilon_n = (1 - \eta\alpha)^n \varepsilon_0$$

(b) $|1 - \eta\alpha| < 1$

$0 < \eta\alpha < 2$

$\eta \in (0, 2/\alpha)$

Fastest: $1 - \eta\alpha = 0$

$$\eta = \frac{1}{\alpha} = \frac{1}{f''(x_n)}$$

(c) $\varepsilon_{n+1} = x_{n+1} - x_*$

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

$= x_n - \eta\alpha x_n + \eta\alpha x_* + \beta x_n - \beta x_{n-1} - x_*$

$= (1 - \eta\alpha + \beta)x_n + (\eta\alpha - 1)x_* - \beta x_{n-1} + \beta x_* - \beta x_*$

$= (1 - \eta\alpha + \beta)(x_n - x_*) - \beta(x_{n-1} - x_*)$

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

(d) $\varepsilon_{n+1} = (1-\alpha\eta+\beta)\varepsilon_n - \beta\varepsilon_{n-1}$ subst α, η, β

$\varepsilon_{n+1} = \left[1-(1)(\frac{4}{9})+\frac{1}{9}\right]\varepsilon_n - (\frac{1}{9})\varepsilon_{n-1}$

$= \frac{6}{9}\varepsilon_n - \frac{1}{9}\varepsilon_{n-1}$

$= \frac{6}{9}(\lambda^n \varepsilon_0) - \frac{1}{9}\lambda^{n-1}\varepsilon_0$

$\lambda^2(\cancel{\lambda^{n-1}}\cancel{\varepsilon_0}) = \frac{6}{9}(\lambda\cancel{\lambda^{n-1}\varepsilon_0}) - \frac{1}{9}\cancel{\lambda^{n-1}}\cancel{\varepsilon_0}$ divide by $\lambda^{n-1}\varepsilon_0$

$\lambda^2 - \frac{6}{9}\lambda + \frac{1}{9} = 0$

$\lambda = 1/3$

$\varepsilon_n = (1/3)^n \varepsilon_0$

$\alpha = 1, \eta = 4/9, \beta = 0$

$\varepsilon_{n+1} = (1 - 4/9)\varepsilon_n$

$\lambda \cancel{\lambda^n \varepsilon_0} = 5/9 \cancel{\lambda^n \varepsilon_0}$

$\lambda = 5/9$

λ is smaller w/o the momentum parameter (when $\beta = 0$), so the rate of convergence will be faster. (error decays faster)

5.4 Newton's Method

(a) $\mathcal{E}_n = |x_n - x_*|$

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

$$f(x) = (x - x_*)^{2p}$$
$$\left.\begin{aligned} f'(x) &= 2p(x - x_*)^{2p-1} \\ f''(x) &= 2p(2p-1)(x - x_*)^{2p-2} \end{aligned}\right\} \text{subst}$$

$$x_n = x_{n-1} - \frac{\cancel{2}p(x_{n-1} - x_*)^{2p-1}}{\cancel{2}p(2p-1)(x_{n-1} - x_*)^{2p-2}}$$

$$= x_{n-1} - \frac{(x_{n-1} - x_*)\cancel{(x - x_*)^{2p-2}}}{(2p-1)\cancel{(x - x_*)^{2p-2}}}$$

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

$$\mathcal{E}_n = |x_n - x_*| = \left| x_{n-1} - \frac{(x_{n-1} - x_*)}{2p-1} - x_* \right| \qquad \mathcal{E}_{n-1} = |x_{n-1} - x_*|$$

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

$$\mathcal{E}_n = \left(1 - \frac{1}{2p-1}\right)\mathcal{E}_{n-1}$$

$$\therefore \mathcal{E}_n = \left| \left(1 - \frac{1}{2p-1}\right)^n \mathcal{E}_0 \right|$$

(b) $\mathcal{E}_n \leq \delta \mathcal{E}_0$

$$\left(1 - \frac{1}{2p-1}\right)^n \cancel{\mathcal{E}_0} \leq \delta \cancel{\mathcal{E}_0}$$

$$\left(1 - \frac{1}{2p-1}\right)^n \leq \delta \qquad \text{take log of both sides}$$

$$n \log\left(1 - \frac{1}{2p-1}\right) \leq \log \delta \qquad \text{mult. by } (-1)$$

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

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

+ so sign doesn't change

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

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

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

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

$$n \geq (2p-1)\log\frac{1}{\delta} \qquad n \geq n_\delta$$

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

$$f'(x) = \frac{d}{dx}\left(x_*\log\left(\frac{x_*}{x}\right) - x_* + x\right) = 1 - \frac{x_*}{x} = 0 \quad \text{min.}$$

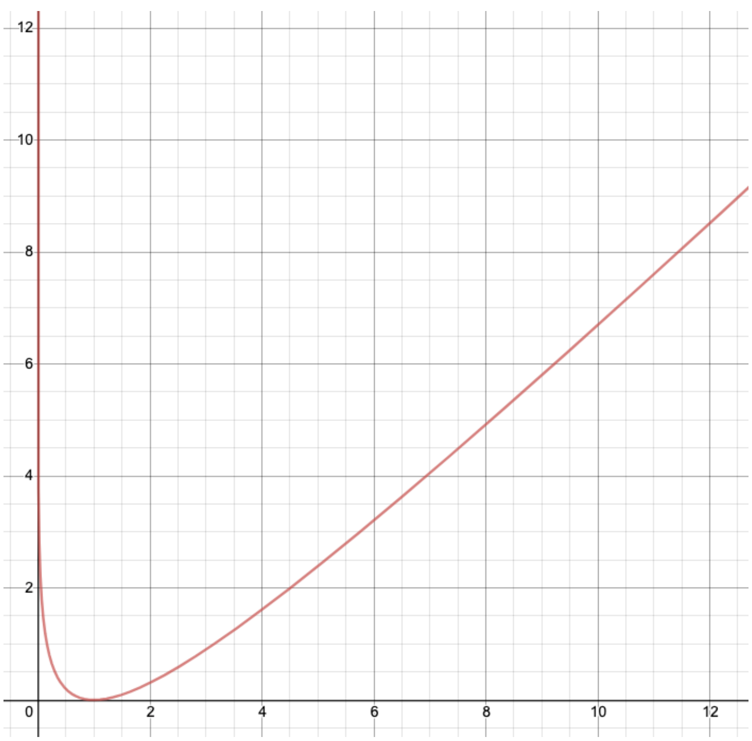
$$1 - \frac{x_*}{x} = 0$$

$$1 = \frac{x_*}{x} \quad \therefore \quad x = x_*$$

$$f''(x) = \frac{d}{dx}\left(1 - \frac{x_*}{x}\right) = \frac{x_*}{x^2} > 0 \rightarrow \text{minimum}$$

Choosing x* = 1

$$y = \ln\left(\frac{1}{x}\right) - 1 + x$$



Minimum is at x = x* = 1.

(d) $p_n = \dfrac{(x_n - x_*)}{x_*}$ Subst x_n

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

$$f'(x_{n-1}) = 1 - \frac{x_*}{x_{n-1}}$$

$$f''(x_{n-1}) = \frac{x_*}{x_{n-1}^2}$$

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

$$= \frac{x_{n-1}(2x_* - x_{n-1})}{x_*}$$

$$p_n = \frac{\frac{x_{n-1}(2x_* - x_{n-1})}{x_*} - x_*}{x_*}$$

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

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

$$p_n = -(p_{n-1})^2$$

$$\therefore \quad p_n = \left(-(p_0)^2\right)^n$$

$$|p_0| < 1$$

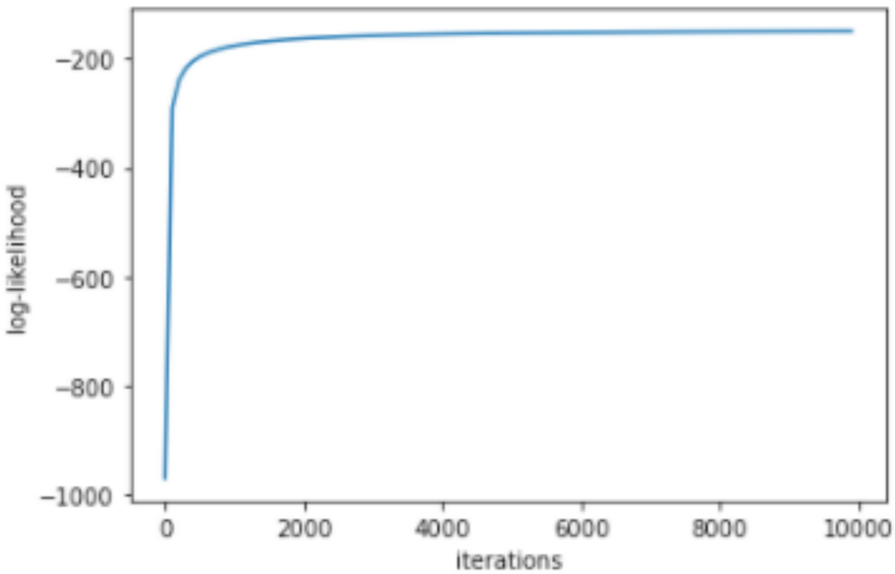
$$\left|\frac{x_0 - x_*}{x_*}\right| < 1$$

$$0 < x_0 < 2x_*$$

5.5 Handwritten digit classification

(a) Training

Method used: **Gradient ascent with 10,000 iterations**



Percent error rate: 3.79%

```
Final weights:
[[-1.25936 -1.37419 -1.93316 -0.94671 -1.63595 -0.47379  0.89571  1.87977]
 [ 0.70296  0.06911  0.79527 -0.54739 -0.10168  0.54825 -1.40402 -0.06085]
 [ 2.68725  1.13078  1.05256  0.35054  0.28219 -2.04812 -3.02995 -3.45483]
 [ 2.12097  0.7162   1.51839 -0.53512 -1.43577 -0.56021  0.36817 -0.18712]
 [ 0.23122  0.27649  0.19447 -0.8278  -0.22176 -0.13688 -0.6316  -0.20239]
 [ 1.22045 -0.94062  0.51926  0.62393  0.45486 -0.67527  0.0177  -1.70624]
 [ 0.42114 -0.22439  0.98362  0.85134  0.05055 -0.26659  0.49028 -1.49941]
 [ 0.25259  0.35063  0.04244  3.7028   0.54296  0.55298 -0.02859 -0.57876]]
```

(b) Testing

Percent error rate: 5.5%

(c) Source code

```
In [39]: import numpy as np
         import matplotlib.pyplot as plt
```
executed in 2ms, finished 16:04:22 2020-11-08

5.3 Handwritten digit classification (Gradient Ascent)

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 [40]: def data(xfname0, xfname1):
             '''
             Create data matrix for train or test data
             '''
             x = np.loadtxt(xfname0)
             x = np.concatenate((x, np.loadtxt(xfname1)))
             y = np.array([0,1]) # 3 is 0, 5 is 1
             y = np.repeat(y, int(len(x)/2))

             return x, y
```
executed in 3ms, finished 16:04:23 2020-11-08

```
In [41]: def sig(z):
             return 1/(1 + np.exp(-z))
```
executed in 2ms, finished 16:04:24 2020-11-08

```
In [42]: def log_likely(x_train, y_train, w):
             '''
             Log-likelihood measurement
             '''
             ll = 0
             for t in range(T):
                 ll += y_train[t]*np.log(sig(np.dot(w, x_train[t]))) + (1 - y_train[t])*np.log(sig(np.dot(-w, x_train[t])))

             return ll
```
executed in 3ms, finished 16:04:24 2020-11-08

```
In [43]: def deriv_ll(x_train, y_train, w):
             '''
             Derivative of log-likelihood (used for gradient ascient)

             '''
             dlldw = []     # partial deriv of log-likelihood wrt weights for each feature
             for a in range(d):
                 dlldw_a = 0     # single feature partial deriv
                 for t in range(T):
                     dlldw_a += x_train[t, a]*(y_train[t] - sig(np.dot(w, x_train[t])))
                 dlldw.append(dlldw_a)

             return np.asarray(dlldw)
```
executed in 3ms, finished 16:04:25 2020-11-08

In [63]:
```python
def grad_ascent(x_train, y_train, w_init=np.zeros(d), maxiter=10000):
    '''
    Implements gradient ascent given an initial set of weights (wo) and maximum iterations

    Returns weights and log-likelihoods for every 100 iterations
    '''
    # Initialize weights
    w = w_init

    # Initial log-likelihood
    prev_ll = 0
    curr_ll = log_likely(x_train, y_train, w)

    # Set max iterations
    i = 0

    # Log likelihood
    lls = []

    print("Beginning gradient ascent...")
    print("Iterations\tLog-Likelihood")

    while (prev_ll != curr_ll) and (i < maxiter):
        # update weights using first partial deriv
        w = w + lr*deriv_ll(x_train, y_train, w)

        if (i%100 == 0):
            lls.append(curr_ll)

        if (i%1000 == 0):
            print("{}:\t\t".format(i), curr_ll)

        # update weights using ll deriv
        prev_ll = curr_ll
        curr_ll = log_likely(x_train, y_train, w)

        i += 1

    print("Done!")

    return w, lls
```
executed in 6ms, finished 20:44:16 2020-11-08

In [45]:
```python
def percent_error(y, y_pred):
    '''
    Percent error between actual y's and predicted y
    '''
    return (1 - sum(y == y_pred)/len(y))*100
```
executed in 3ms, finished 16:04:27 2020-11-08

In [51]:
```python
# Training data
x_train, y_train = data('train3.txt', 'train5.txt')

# Initialize weights and max iterations
maxiter = 50000

# Perform gradient ascent
w, lls = grad_ascent(x_train, y_train, maxiter=maxiter)
```
executed in 3h 20m 51s, finished 19:29:05 2020-11-08

```
Beginning gradient ascent...
Iterations      Log-Likelihood
0:               -970.4060527838883
1000:            -175.72311092694093
2000:            -162.537018816536
3000:            -157.29803908159442
4000:            -154.47199931317115
5000:            -152.71296313269082
6000:            -151.52149261645846
7000:            -150.66806226322777
8000:            -150.03214427055346
9000:            -149.54432033286358
Done!
```

In [59]:

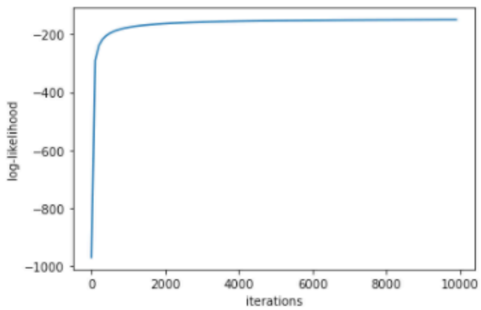
```python
# Results
# Plot the log-likelihood for several iterations
iterations = np.arange(0, 10000, 100)
plt.plot(iterations, lls)
plt.xlabel('iterations')
plt.ylabel('log-likelihood')

# Percent error rate
y_pred_train = np.rint(sig(np.dot(w, x_train.transpose())))
per_train = percent_error(y_train, y_pred_train)
print('Training percent error rate:\t{}%'.format(round(per_train, 2)))

print("Final weights:")
print(np.round(w, decimals=5).reshape((8,8)))
```

executed in 109ms, finished 20:40:40 2020-11-08

```
Training percent error rate:    3.79%
Final weights:
[[-1.25936 -1.37419 -1.93316 -0.94671 -1.63595 -0.47379  0.89571  1.87977]
 [ 0.70296  0.06911  0.79527 -0.54739 -0.10168  0.54825 -1.40402 -0.06085]
 [ 2.68725  1.13078  1.05256  0.35054  0.28219 -2.04812 -3.02995 -3.45483]
 [ 2.12097  0.7162   1.51839 -0.53512 -1.43577 -0.56021  0.36817 -0.18712]
 [ 0.23122  0.27649  0.19447 -0.8278  -0.22176 -0.13688 -0.6316  -0.20239]
 [ 1.22045 -0.94062  0.51926  0.62393  0.45486 -0.67527  0.0177  -1.70624]
 [ 0.42114 -0.22439  0.98362  0.85134  0.05055 -0.26659  0.49028 -1.49941]
 [ 0.25259  0.35063  0.04244  3.7028   0.54296  0.55298 -0.02859 -0.57876]]
```



(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.

In [55]:

```python
# Percent error rate
x_test, y_test = data('test3.txt', 'test5.txt')
y_pred_test = np.rint(sig(np.dot(w, x_test.transpose())))
per_test = percent_error(y_test, y_pred_test)
print('Testing percent error rate:\t{}%'.format(round(per_test, 2)))
```

executed in 31ms, finished 20:36:47 2020-11-08

```
Testing percent error rate:     5.5%
```

In [17]:

```python
w = np.array([-1.25936385, -1.37419156, -1.93315691, -0.94670866, -1.63594695,
       -0.47379369,  0.89570972,  1.87976697,  0.70295908,  0.06910995,
        0.79526537, -0.54738677, -0.10168444,  0.54824705, -1.40401752,
       -0.06085015,  2.68724645,  1.1307781 ,  1.05255621,  0.35054039,
        0.28218726, -2.04811778, -3.02994933, -3.45483084,  2.12096748,
        0.71620149,  1.51838805, -0.53512089, -1.43576657, -0.56020994,
        0.36817054, -0.18712274,  0.23122339,  0.27648621,  0.19446814,
       -0.8277966 , -0.22176086, -0.1368842 , -0.63159855, -0.20238899,
        1.22045228, -0.94061884,  0.51925866,  0.62392667,  0.45485562,
       -0.67526907,  0.01770258, -1.70623856,  0.42114389, -0.22439154,
        0.98362023,  0.85134271,  0.05054536, -0.26658909,  0.49027812,
       -1.49941143,  0.25258644,  0.35062984,  0.04243829,  3.70280437,
        0.54295649,  0.5529752 , -0.02858906, -0.57876249])
```

executed in 6ms, finished 15:57:46 2020-11-08