COMP755-Lect06

September 10, 2018

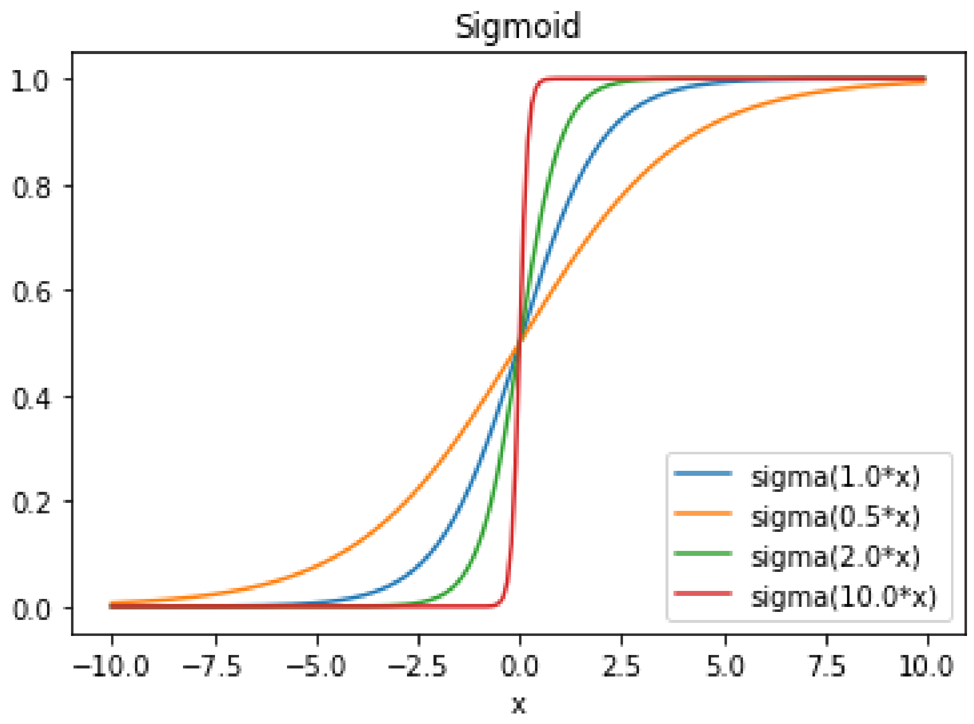
1 COMP 755

Plan for today

1. Review logistic regression
2. Multi-class logistic regression and Softmax
3. BigData and Stochastic gradient descent

```
In [1]: import numpy
        import matplotlib.pyplot as plt
        %matplotlib inline
        x = numpy.arange(-10,10,0.1)
        scales = [1.0,0.5,2.0,10.0]
        labels = []
        for s in scales:
            plt.plot(x,1.0/(1.0 + numpy.exp(-s*x)))
            labels.append('sigma(' + str(s) +'*x)')
        plt.xlabel('x')
        plt.title('Sigmoid')
        plt.legend(labels,loc=4)
```

```
Out[1]: <matplotlib.legend.Legend at 0x107a88080>
```



2 Logistic regression -- log-likelihood for ± 1 labels

Probability of a single sample is when $y \in \{-1, +1\}$:

$$p(y|\mathbf{x}, \beta_0, \beta) = \frac{1}{1 + \exp\{-y(\beta_0 + \mathbf{x}^T\beta)\}}$$

Likelihood function is:

$$\mathcal{L}(\beta_0, \beta|\mathbf{y}, \mathbf{x}) = \prod_i \frac{1}{1 + \exp\{-y_i(\beta_0 + \mathbf{x}_i^T\beta)\}}$$

Log-likelihood function is:

$$\mathcal{LL}(\beta_0, \beta|\mathbf{y}, \mathbf{x}) = -\sum_i \log\left\{1 + \exp\left\{-y_i(\beta_0 + \mathbf{x}_i^T\beta)\right\}\right\}$$

Ridge regularized log-likelihood:

$$\mathcal{PLL}(\beta_0, \beta|\mathbf{y}, \mathbf{x}) = -\sum_i \log\left\{1 + \exp\left\{-y_i(\beta_0 + \mathbf{x}_i^T\beta)\right\}\right\} - \frac{\lambda}{2}\|\beta\|^2$$

3 Logistic regression -- log-likelihood for $0, 1$ labels

Probability of a single sample is when $y \in \{0, 1\}$:

$$p(y|\mathbf{x}, \beta_0, \beta) = \frac{\exp\{y(\beta_0 + \mathbf{x}^T\beta)\}}{1 + \exp\{(\beta_0 + \mathbf{x}^T\beta)\}}$$

Likelihood function is:

$$\mathcal{L}(\beta_0, \beta|\mathbf{y}, \mathbf{x}) = \prod_i \frac{\exp\{y_i(\beta_0 + \mathbf{x}_i^T\beta)\}}{1 + \exp\{(\beta_0 + \mathbf{x}^T\beta)\}}$$

Log-likelihood function is:

$$\mathcal{LL}(\beta_0, \beta|\mathbf{y}, \mathbf{x}) = \sum_i y_i(\beta_0 + \mathbf{x}_i^T\beta) - \log\left\{1 + \exp\left\{(\beta_0 + \mathbf{x}_i^T\beta)\right\}\right\}$$

Ridge regularized log-likelihood:

$$\mathcal{PLL}(\beta_0, \beta|\mathbf{y}, \mathbf{x}) = \sum_i y_i(\beta_0 + \mathbf{x}_i^T\beta) - \log\left\{1 + \exp\left\{(\beta_0 + \mathbf{x}_i^T\beta)\right\}\right\} - \frac{\lambda}{2}\|\beta\|^2$$

4 Decision boundary -- Separating hyperplane

For Logistic regression:

$$p(y = 1|\mathbf{x}, \beta, \beta_0) = 0.5 \iff \beta_0 + \mathbf{x}^T\beta = 0$$

5 Multiclass classification

Given a feature vector $\mathbf{x}$ we wish to predict which of the C classes it came from. If $C = 2$ we can use logistic regression. What if $C > 2$?

Q: Suggest how you might use a two class classifier in training a multiclass classifier? Hint: you can train more than one two-class classifier and aggregate their predictions.

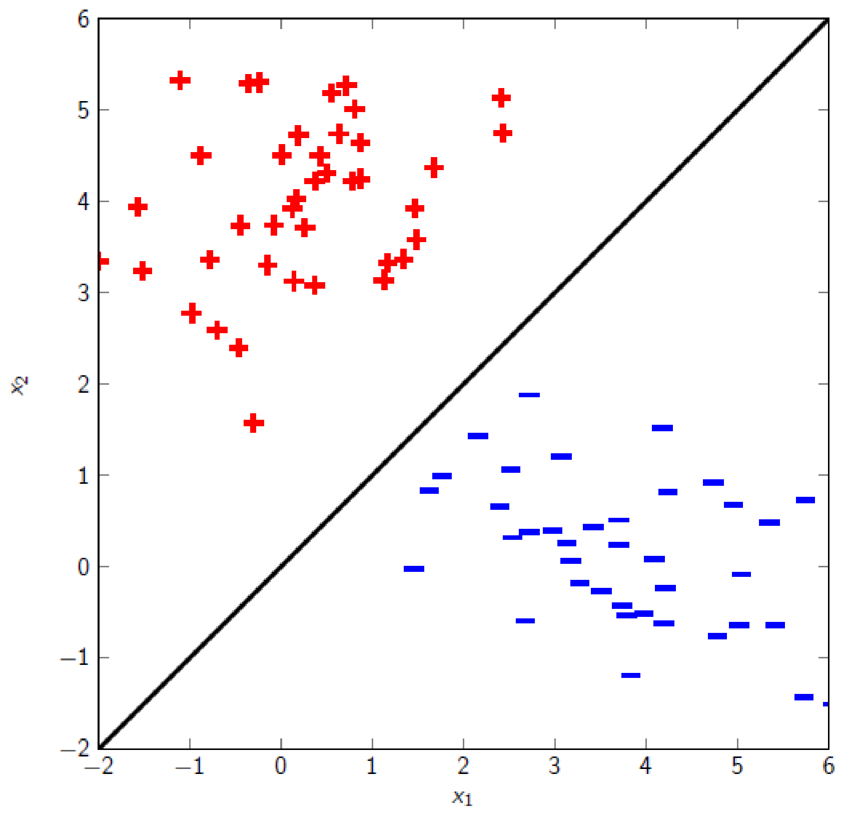
6 Multiclass classification

We need to specify

$$p(y = 1|\mathbf{x}), p(y = 2|\mathbf{x}), ..., p(y = C|\mathbf{x})$$

where we note that

$$\sum_{c=1}^{C} p(y = c|\mathbf{x}) = 1.0$$
$$p(y = c|\mathbf{x}) \geq 0, \quad 1 \leq c \leq C$$



Separating Hyperplane

7 Softmax

Sigmoid:

$$\sigma(z) = \frac{1}{1+\exp\{-z\}} = \frac{\exp\{z\}}{1+\exp\{z\}}$$

Softmax is a generalization of sigmoid:

$$\sigma(\mathbf{z})_j = \frac{\exp\{z_j\}}{\sum_{c=1}^{C} \exp\{z_j\}}$$

For example:

$$\sigma(\mathbf{z})_1 = \frac{\exp\{z_1\}}{\exp\{z_1\}+\exp\{z_2\}+\exp\{z_3\}}$$
$$\sigma(\mathbf{z})_2 = \frac{\exp\{z_2\}}{\exp\{z_1\}+\exp\{z_2\}+\exp\{z_3\}}$$
$$\sigma(\mathbf{z})_3 = \frac{\exp\{z_3\}}{\exp\{z_1\}+\exp\{z_2\}+\exp\{z_3\}}$$

8 Why is it called softmax?

Imagine a function that gives back an indicator vector where 1 for the largest entry, 0 for others.

For example, $[0.1, 0.5, 0.3]$ would map to $[0, 1, 0]$ because the second entry is the largest.

Softmax maps input values into probabilities. The largest value is mapped into the largest probability, but no probability is 0.

Hence, softmax is a "soft" version of the above max function.

```
In [54]: # why softmax
         from __future__ import print_function
         import numpy
         def hardmax(z):
             m = numpy.max(z)
             h = numpy.double(z == m)
             print("z: ",z,"Hardmax(z):",h)
             return h

         def softmax(z,verbose=True):
             s = numpy.exp(z)
             s = s/numpy.sum(s)
             if verbose:
                 print("z: ",z,"Softmax(z):",s)
             return s

         hardmax([0.1,0.5,0.3])
         softmax([0.1,0.5,0.3])
         softmax([0.01,0.05,0.03])
         softmax([10,50,30])
         softmax([50,50,30]);
```

```
z:  [0.1, 0.5, 0.3] Hardmax(z): [ 0.  1.  0.]
z:  [0.1, 0.5, 0.3] Softmax(z): [ 0.2693075   0.40175958  0.32893292]
z:  [0.01, 0.05, 0.03] Softmax(z): [ 0.32668933  0.34002178  0.33328889]
z:  [10, 50, 30] Softmax(z): [  4.24835425e-18   9.99999998e-01   2.06115362e-09]
z:  [50, 50, 30] Softmax(z): [  4.99999999e-01   4.99999999e-01   1.03057681e-09]
```

9 Softmax and temperature

Note that

$$\sigma\left(\mathbf{z}\right) \neq \sigma\left(\frac{1}{T}\mathbf{z}\right), T \neq 1$$

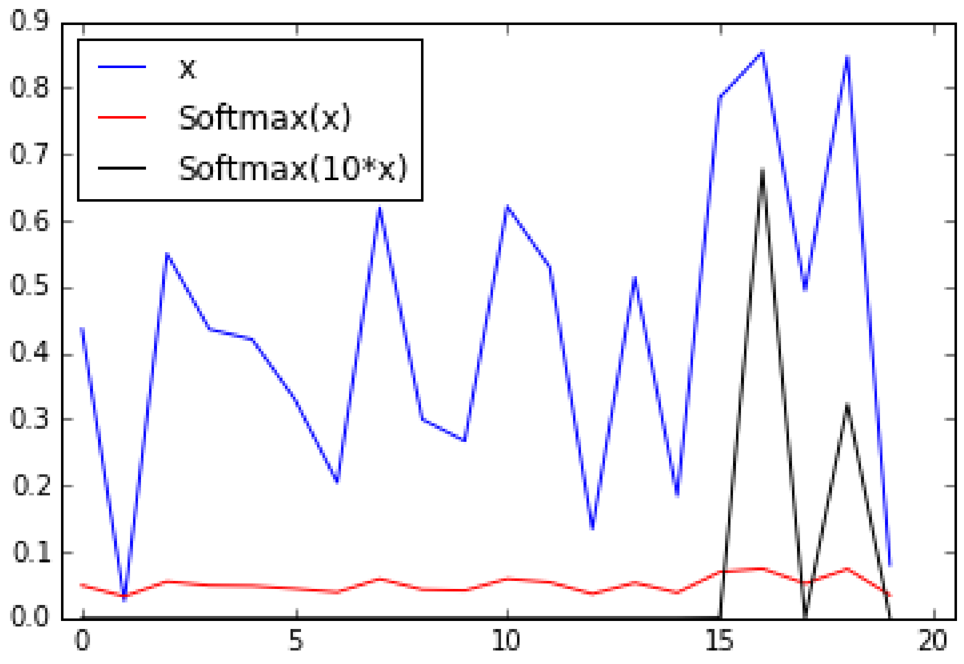
In fact, for small T the softmax output approaches the indicator vector which tells you which entries are the largest.

Hence softmax can be made arbirarily "sharp" or "peaked" by scaling its inputs.

BTW. If you've done any simulated annealing, you have been using softmax and tuning the temperature up to drive the algorithm to find the maximum.

```
In [75]: import matplotlib.pyplot as plt
         %matplotlib inline
         numpy.random.seed(2)
         x = numpy.random.rand(20,1)
         plt.plot(x,'b',label='x')
         plt.plot(softmax(x,verbose=False),'r',label='Softmax(x)')
         plt.plot(softmax(100.0*x,verbose=False),'k',label='Softmax(10*x)');
         plt.xlim([-0.5,20.5])
         plt.legend(loc=0)
         print([x[16],x[18]])
```

```
[array([ 0.85397529]), array([ 0.84656149])]
```



10 Multiclass logistic regression and softmax

We can write out probability of partcular class using softmax

$$p(y = c|\mathbf{x}, \beta_0, B) = \boxed{\frac{\exp\{\beta_{0,c} + \mathbf{x}^T \mathbf{fi}_c\}}{\sum_{k=1}^{C} \exp\{\beta_{0,k} + \mathbf{x}^T \mathbf{fi}_k\}}}$$

where

$$B = [\mathbf{fi}_1 \mathbf{fi}_2 ... \mathbf{fi}_C]$$

and each $\mathbf{fi}_c$ is a vector of class specific feature weights.

Note that the $p(y = c| \cdots)$ is a categorical distribution over C possible states. Probabilities of each state are given by softmax.

11 Excess of parameters

Probability of a single sample in logistic regression assuming where $y \in \{1, 2\}$

$$p(y = 1|\mathbf{x}, \beta_0, \beta) = \frac{1}{1 + \exp\{\beta_0 + \mathbf{x}^T \beta)\}}$$

$$p(y = 2|\mathbf{x}, \beta_0, \beta) = \frac{\exp\{\beta_0 + \mathbf{x}^T \beta\}}{1 + \exp\{\beta_0 + \mathbf{x}^T \beta\}}$$

But in our multiclass logistic regression for $C = 2$ we have

$$p(y = 1|\mathbf{x}, \beta_0, B) \propto \frac{\exp\{\beta_{0,1} + \mathbf{x}^T \mathbf{fi}_1\}}{\exp\{\beta_{0,1} + \mathbf{x}^T \mathbf{fi}_1\} + \exp\{\beta_{0,2} + \mathbf{x}^T \mathbf{fi}_2\}}$$
$$p(y = 2|\mathbf{x}, \beta_0, B) = \frac{\exp\{\beta_{0,2} + \mathbf{x}^T \mathbf{fi}_2\}}{\exp\{\beta_{0,1} + \mathbf{x}^T \mathbf{fi}_1\} + \exp\{\beta_{0,2} + \mathbf{x}^T \mathbf{fi}_2\}}$$

Multiclass logistic regression for $C = 2$ has more paramters than logistic regresssion.

Q: Do we need these parameters? If not, why not and how do we get rid of them? Just need intuition here, we will work it out on the board.

12 Counting parameters

$$p(y = c|\mathbf{x}, \beta_0, B) = \boxed{\frac{\exp\{\beta_{0,c} + \mathbf{x}^T \mathbf{fi}_c\}}{\sum_{k=1}^{C} \exp\{\beta_{0,k} + \mathbf{x}^T \mathbf{fi}_k\}}}$$

If there are C classes and p features

Q: Size of β_0?

Q: Size of

$$B = [\mathbf{fi}_1 \mathbf{fi}_2 ... \mathbf{fi}_C]$$

matrix of **fi** vectors?

Q: Assuming that $\beta_{0,C} = 0$ and $\mathbf{fi}_{j,C} = 0$, total parameter count is?

13 One-hot representation of labels

Thus far we have been using a random variable y and writing probabilities as

$$p(y = c|\cdots) =$$

If you give me a discrete variable $z \in \{1, ...C\}$ I can construct a C-long binary vector

$$y_c = \begin{cases} 1, & z = c \\ 0, & z \neq c \end{cases}$$

This bit vector is **one-hot** representation of original variable z.

For example, if C = 5 1. $z = 3$ then $y = [00100]$ 2. $z = 2$ then $y = [01000]$ 3. $z = 4$ then $y = [00010]$, and

$$\prod_{c=1}^{C} \theta_c^{y_c} = \theta_1^0 \theta_2^0 \theta_3^0 \theta_4^1 \theta_5^0 = \theta_4$$

and

$$\log \prod_{c=1}^{C} \theta_c^{y_c} = \sum_c y_c \log \theta_c = \log \theta_4$$

14 Log-Likelihood

1. There are N samples, each in one of C classes, and p features
2. Labels are represented using one-hot vectors y_i
3. Feature matrix X contains a column of 1s -- corresponding to the bias term.
4. First row of weight matrix B are bias terms.
5. $\mathbf{fi}_k$ is k^{th} column of matrix B

	Variable	Dimensions
feature matrix	X	$N \times (p+1)$
label matrix	Y	$N \times C$
weight matrix	B	$(p+1) \times C$

Likelihood is

$$\mathcal{L}(B|Y,X) = \underbrace{\prod_{i=1}^{N}}_{\text{samples}} \underbrace{\prod_{c=1}^{C}}_{\text{classes}} \left[\frac{\exp\left\{\mathbf{x}_i^T \mathbf{fi}_c\right\}}{\sum_{k=1}^{C} \exp\left\{\mathbf{x}_i^T \mathbf{fi}_k\right\}} \right]^{y_{i,c}}$$

Log-likelihood is

$$\mathcal{LL}(\beta_0, B|Y,X) = \sum_{i=1}^{N} \sum_{c=1}^{C} y_{i,c} \left(\mathbf{x}_i^T \mathbf{fi}_c - \log\left\{ \sum_{k=1}^{C} \exp\left\{ \mathbf{x}_i^T \mathbf{fi}_k \right\} \right\} \right)$$

15 Ridge regularized log-likelihood

Ridge regularized log-likelihood

$$\begin{aligned}\mathcal{PLL}(B|Y,X) = &\sum_{i=1}^{N} \sum_{c=1}^{C} y_{i,c} \left(\mathbf{x}_i^T \mathbf{fi}_c - \log\left\{ \sum_{k=1}^{C} \exp\left\{ \mathbf{x}_i^T \mathbf{fi}_k \right\} \right\} \right) \\ &- \frac{\lambda}{2} \sum_{k=1}^{C} \sum_{j=1}^{p} \beta_{j,k}^2\end{aligned}$$

Note that we keep the last column of B fixed at 0 to get rid of excess parameters.
These parameters will not contribute to the regularization -- sum of their squares is 0.

16 Cross-entropy

Frequently you will encounter mentions of cross-entropy. It is negative log likelihood of multiclass logistic

$$\begin{aligned}\text{crossentropy}(B) &= -\mathcal{LL}(B|Y,X) \\ &= -\sum_{i=1}^{N} \sum_{c=1}^{C} y_{i,c} \left(\mathbf{x}_i^T \mathbf{fi}_c - \log\left\{ \sum_{k=1}^{C} \exp\left\{ \mathbf{x}_i^T \mathbf{fi}_k \right\} \right\} \right)\end{aligned}$$

Ridge regularized cross-entropy

$$\text{crossentropy}(B) = -\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}\left(\mathbf{x}_i^T\mathbf{fi}_c - \log\left\{\sum_{k=1}^{C}\exp\left\{\mathbf{x}_i^T\mathbf{fi}_k\right\}\right\}\right)$$
$$+\frac{\lambda}{2}\sum_{k=1}^{C}\sum_{j=1}^{p}\beta_{j,k}^2$$

Note the sign flip in the regularization.

17 Gradients of log-likelihood

We will work this out in a pedestrian fashion and then obtain a compact expression:

$$\mathcal{LL}(B) = \sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}\left(\underbrace{\mathbf{x}_i^T\mathbf{fi}_c}_{\text{involves only }\beta_c} - \underbrace{\log\left\{\sum_{k=1}^{C}\exp\left\{\mathbf{x}_i^T\mathbf{fi}_k\right\}\right\}}_{\text{involves all columns of }B}\right)$$

Break it down

$$\mathcal{LL}(B) = \sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}\left(\mathbf{x}_i^T\mathbf{fi}_c\right)$$
$$-\sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}\underbrace{\log\left\{\sum_{k=1}^{C}\exp\left\{\mathbf{x}_i^T\mathbf{fi}_k\right\}\right\}}_{\text{does not involve c}}$$

Hence

$$\mathcal{LL}(B) = \sum_{i=1}^{N}\sum_{c=1}^{C} y_{i,c}\left(\mathbf{x}_i^T\mathbf{fi}_c\right)$$
$$-\sum_{i=1}^{N}\log\left\{\sum_{k=1}^{C}\exp\left\{\mathbf{x}_i^T\mathbf{fi}_k\right\}\right\}$$

On board we will work out

$$\frac{\partial}{\partial\beta_{j,l}}\log\left\{\sum_{k=1}^{C}\exp\left\{\mathbf{x}_i^T\mathbf{fi}_k\right\}\right\} = \boxed{\frac{\exp\left\{\mathbf{x}_i^T\mathbf{fi}_l\right\}}{\sum_{k=1}^{C}\exp\left\{\mathbf{x}_i^T\mathbf{fi}_k\right\}}}x_{i,j}$$

Hence

$$\frac{\partial}{\partial\beta_{j,l}}\mathcal{LL}(B) = \sum_i y_{i,l}x_{i,j}$$
$$-\sum_{i=1}^{N}\frac{\exp\left\{\mathbf{x}_i^T\mathbf{fi}_l\right\}}{\sum_{k=1}^{C}\exp\left\{\mathbf{x}_i^T\mathbf{fi}_k\right\}}x_{i,j}$$

For compactness, model's probability of class l for sample i will be denoted $\mu_{i,l}$

$$\mu_{i,l} = \frac{\exp\left\{\mathbf{x}_i^T\mathbf{fi}_l\right\}}{\sum_{k=1}^{C}\exp\left\{\mathbf{x}_i^T\mathbf{fi}_k\right\}}$$

and

$$\frac{\partial}{\partial \beta_{j,l}} \mathcal{LL}(B) = \sum_i y_{i,l} x_{i,j} - \sum_{i=1}^{N} \mu_{i,l} x_{i,j} = \sum_i x_{i,j} \underbrace{(y_{i,l} - \mu_{i,l})}_{\text{residual}}$$

$$\frac{\partial}{\partial \beta_{j,l}} \mathcal{LL}(B) = \sum_i \underbrace{x_{i,j}}_{\text{feature } j} \underbrace{(y_{i,l} - \mu_{i,l})}_{\text{residual in predicting class } l} = \mathbf{x}_{:,j}^T (\mathbf{y}_{:,l} - \mu_{:,l})$$

In words, partial derivative of log-likelihood with respect to j^{th} feature's weight for class l is inner product between the feature and disagreement between prediction and the true label.

Gradient of log likelihood with respect to a column of B

$$\nabla_{\beta_c} \mathcal{LL}(B) = \sum_i (y_{i,c} - \mu_{i,c}) \mathbf{x}_i$$

Gradient of ridge regularized log-likelihood with respect to a column of B

$$\nabla_{\beta_c} \mathcal{PLL}(B) = \sum_i (y_{i,c} - \mu_{i,c}) \mathbf{x}_i - \lambda \begin{bmatrix} 0 \\ \mathbf{1}_p \end{bmatrix}$$

18 BigData and stochastic gradient

If the number of samples N is in thousands rather than millions then computation of gradients like

$$\mu_{i,l} = \frac{\exp\left\{\mathbf{x}_i^T \mathbf{fi}_l\right\}}{\sum_{k=1}^{C} \exp\left\{\mathbf{x}_i^T \mathbf{fi}_k\right\}}$$
$$\nabla_{\beta_c} \mathcal{LL}(B) = \sum_i (y_{i,c} - \mu_{i,c}) \mathbf{x}_i$$

is not overly expensive.

Methods that use the whole dataset on every update of parameters are called **batch** methods.

19 BigData and stochastic gradient

Once the number of samples becomes large iterating over all of them has diminishing returns.

Stochastic gradient methods compute gradients using a portion of data called **mini-batches**.

An extreme example of this is **online learning** -- data is streamed one sample at a time.

20 BigData and stochastic gradient

Updating parameters based on a small set of data if not guaranteed to monotonically improve log-likelihood.

Hence, step-size cannot be chosen using line-search.

Instead, step sizes for each iteration k are computed

$$s^{(k)} = \left(s^{(k-1)}\right)^{1-\epsilon} \qquad \epsilon \in [0, 1]$$
$$s^{(k)} = \frac{1}{\tau + k} \qquad \tau > 0$$

and lead to diminishing step-size.

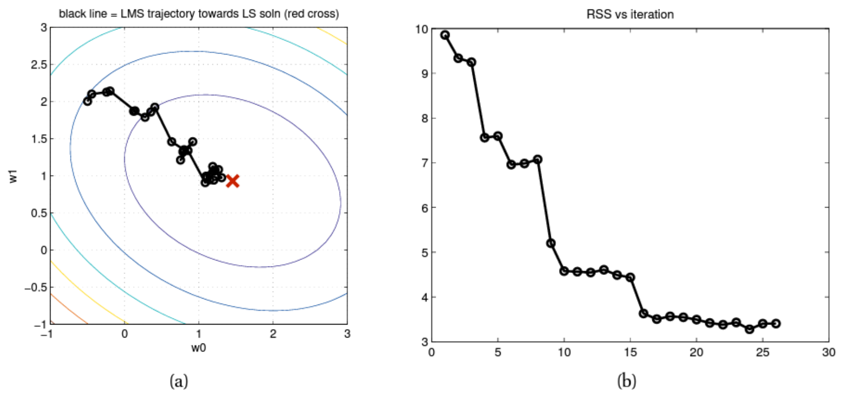


Figure 8.8 Illustration of the LMS algorithm. Left: we start from $\boldsymbol{\theta} = (-0.5, 2)$ and slowly converging to the least squares solution of $\hat{\boldsymbol{\theta}} = (1.45, 0.92)$ (red cross). Right: plot of objective function over time. Note that it does not decrease monotonically. Figure generated by `LMSdemo`.

```
In [ ]: def stochastic_gradient_ascent(f,x,get_next_batch,iters,epsilon=0.99):
            for it in range(iters):
                batch = get_next_batch()
                v,g = f(x,batch)
                x = x + step*g
                step = step**(1.0 - epsilon)
            return x

        def stochastic_gradient_descent(f,x,get_next_batch,iters,epsilon=0.99):
            for it in range(iters):
                batch = get_next_batch()
                v,g = f(x,batch)
                x = x - step*g
                step = step**(1.0 - epsilon)
            return x
```

21 Today

Softmax -- generalization of sigmoid.
Multiclass classification using logistic regression.
Started looking at Stochastic gradients and BigData.