CSCI567 Machine Learning (Fall 2016)

Dr. Yan Liu

yanliu.cs@usc.edu

September 19, 2016

Outline

Logistic classification

Setup for two classes

- Input: $\boldsymbol{x} \in \mathbb{R}^D$
- Output: $y \in \{0, 1\}$
- Training data: $\mathcal{D} = \{(\boldsymbol{x}_n, y_n), n = 1, 2, \ldots, N\}$
- Model:

$$p(y = 1|\boldsymbol{x}; b, \boldsymbol{w}) = \sigma[g(\boldsymbol{x})]$$

where

$$g(\boldsymbol{x}) = b + \sum_d w_d x_d = b + \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}$$

and $\sigma[\cdot]$ stands for the *sigmoid* function

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

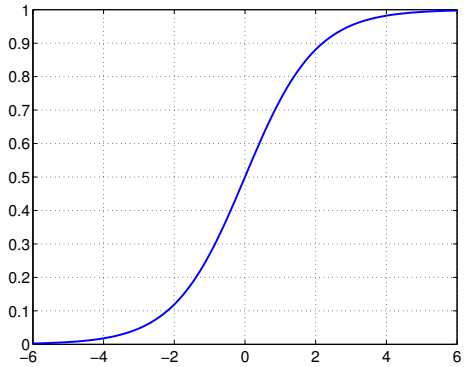
Why the sigmoid function?

What does it look like?

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

where

$$a = b + \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}$$



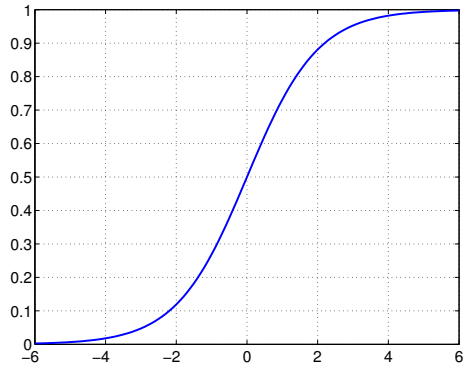
Why the sigmoid function?

What does it look like?

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

where

$$a = b + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}$$



Properties

- Bounded between 0 and 1 ← thus, interpretable as probability
- Monotonically increasing thus, usable to derive classification rules
  1. $\sigma(a) > 0.5$, positive (classify as '1')
  2. $\sigma(a) < 0.5$, negative (classify as '0')
  3. $\sigma(a) = 0.5$, undecidable
- Nice computationally properties These will unfold in the next few slides

Linear or nonlinear?

$\sigma(a)$ **is nonlinear**, however, the decision boundary is determined by

$$\sigma(a) = 0.5 \Rightarrow a = 0 \Rightarrow g(\boldsymbol{x}) = b + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x} = 0$$

which is a *linear* function in $\boldsymbol{x}$

We often call b the bias term.

Contrast Naive Bayes and our new model

Similar

Both look at the linear function of features for classification

Difference

Naive Bayes models the *joint* distribution

$$P(X, Y) = P(Y)P(X|Y)$$

Logistic regression models the *conditional* distribution

$$P(Y|X)$$

Likelihood function

Probability of a single training sample $(\boldsymbol{x}_n, y_n)$

$$p(y_n|\boldsymbol{x}_n; b; \boldsymbol{w}) = \begin{cases} \sigma(b + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) & \text{if} \quad y_n = 1 \\ 1 - \sigma(b + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) & \text{otherwise} \end{cases}$$

Likelihood function

Probability of a single training sample $(\boldsymbol{x}_n, y_n)$

$$p(y_n|\boldsymbol{x}_n; b; \boldsymbol{w}) = \begin{cases} \sigma(b + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) & \text{if} \quad y_n = 1 \\ 1 - \sigma(b + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) & \text{otherwise} \end{cases}$$

Compact expression, exploring that y_n is either 1 or 0

$$p(y_n|\boldsymbol{x}_n; b; \boldsymbol{w}) = \sigma(b + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)^{y_n}[1 - \sigma(b + \boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)]^{1-y_n}$$

Cross-entropy error

Log-likelihood of the whole training data $\mathcal{D}$

$$\log P(\mathcal{D}) = \sum_n \{y_n \log \sigma(b + \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n) + (1 - y_n) \log[1 - \sigma(b + \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n)]\}$$

Cross-entropy error

Log-likelihood of the whole training data $\mathcal{D}$

$$\log P(\mathcal{D}) = \sum_n \{y_n \log \sigma(b + \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n) + (1 - y_n) \log[1 - \sigma(b + \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n)]\}$$

It is convenient to work with its negation, which is called

cross-entropy error function

$$\mathcal{E}(b, \boldsymbol{w}) = -\sum_n \{y_n \log \sigma(b + \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n) + (1 - y_n) \log[1 - \sigma(b + \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n)]\}$$

Maximum likelihood estimation

Cross-entropy error (negative log-likelihood)

$$\mathcal{E}(b, \boldsymbol{w}) = -\sum_n \{y_n \log \sigma(b + \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n) + (1 - y_n) \log[1 - \sigma(b + \boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n)]\}$$

Numerical optimization

- Gradient descent: simple, scalable to large-scale problems
- Newton method: fast but not scalable

Shorthand notation

This is for convenience

- Append 1 to $\boldsymbol{x}$

$$\boldsymbol{x} \leftarrow [1 \quad x_1 \quad x_2 \quad \cdots \quad x_D]$$

- Append b to $\boldsymbol{w}$

$$\boldsymbol{w} \leftarrow [b \quad w_1 \quad w_2 \quad \cdots \quad w_D]$$

- Cross-entropy is then

$$\mathcal{E}(\boldsymbol{w}) = -\sum_n \{y_n \log \sigma(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n) + (1 - y_n) \log[1 - \sigma(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n)]\}$$

NB. We are not using the $\tilde{\boldsymbol{x}}$ and $\tilde{\boldsymbol{w}}$ (as in several textbooks) for cosmetic reasons.

How to find the optimal parameters for logistic regression?

We will minimize the error function

$$\mathcal{E}(\boldsymbol{w}) = -\sum_n \{y_n \log \sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) + (1-y_n)\log[1-\sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n)]\}$$

However, this function is complex and we cannot find the simple solution as we did in Naive Bayes. So we need to use *numerical* methods.

- Numerical methods are messier, in contrast to cleaner analytic solutions.
- In practice, we often have to tune a few optimization parameters — patience is necessary.

An overview of numerical methods

We describe two

- Gradient descent (our focus in lecture): simple, especially effective for large-scale problems
- Newton method: classical and powerful method

Gradient descent is often referred to as an *first-order* method as it requires only to compute the gradients (i.e., the first-order derivative) of the function.

In contrast, Newton method is often referred as to an *second-order* method.

Example: $\min f(\boldsymbol{\theta}) = 0.5(\theta_1^2 - \theta_2)^2 + 0.5(\theta_1 - 1)^2$

- We compute the gradients

$$\frac{\partial f}{\partial \theta_1} = 2(\theta_1^2 - \theta_2)\theta_1 + \theta_1 - 1 \tag{1}$$

$$\frac{\partial f}{\partial \theta_2} = -(\theta_1^2 - \theta_2) \tag{2}$$

- Use the following *iterative* procedure for *gradient descent*
  1. Initialize $\theta_1^{(0)}$ and $\theta_2^{(0)}$, and $t = 0$
  2. do

$$\theta_1^{(t+1)} \leftarrow \theta_1^{(t)} - \eta \left[2({\theta_1^{(t)}}^2 - \theta_2^{(t)})\theta_1^{(t)} + \theta_1^{(t)} - 1 \right] \tag{3}$$

$$\theta_2^{(t+1)} \leftarrow \theta_2^{(t)} - \eta \left[-({\theta_1^{(t)}}^2 - \theta_2^{(t)}) \right] \tag{4}$$

$$t \leftarrow t + 1 \tag{5}$$

  3. until $f(\boldsymbol{\theta}^{(t)})$ *does not change much*

Gradient descent

General form for minimizing $f(\theta)$

$$\boldsymbol{\theta}^{t+1} \leftarrow \boldsymbol{\theta} - \eta \frac{\partial f}{\partial \boldsymbol{\theta}}$$

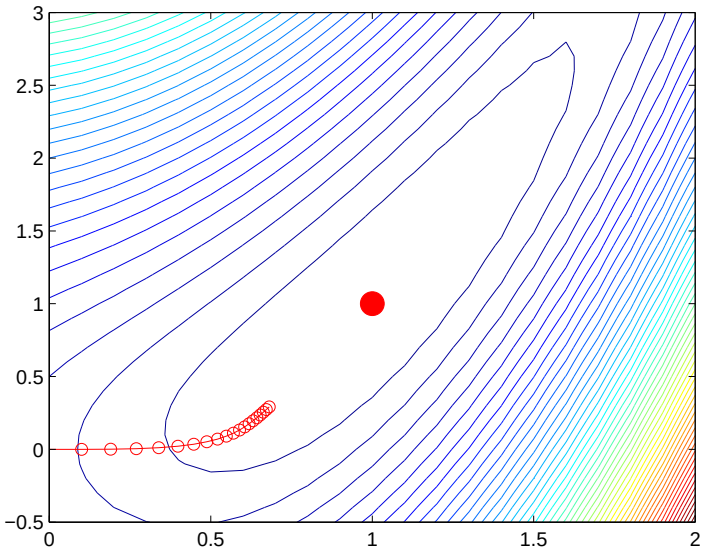
Remarks

- η is often called *step size* – literally, how far our update will go along the direction of the negative gradient
- Note that this is for *minimizing* a function, hence the subtraction $(-\eta)$
- With a *suitable* choice of η, the iterative procedure converges to a stationary point where
$$\frac{\partial f}{\partial \boldsymbol{\theta}} = 0$$
- A stationary point is only necessary for being the minimum.

Seeing in action

Choose the right η is important

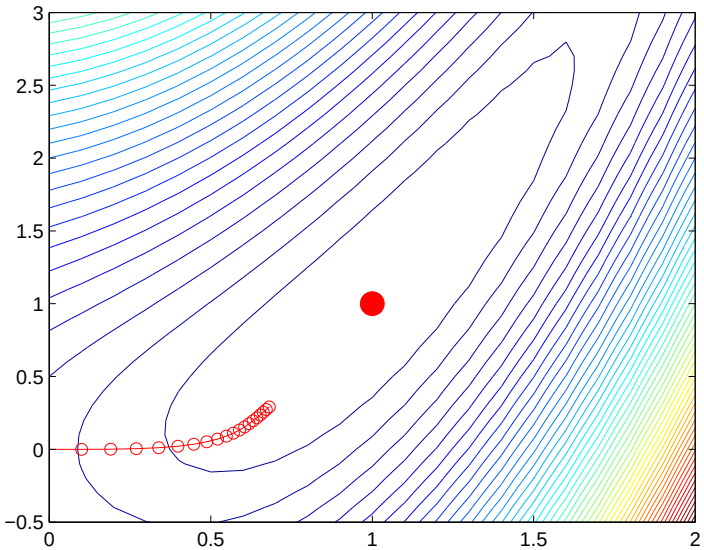
small η is too slow?



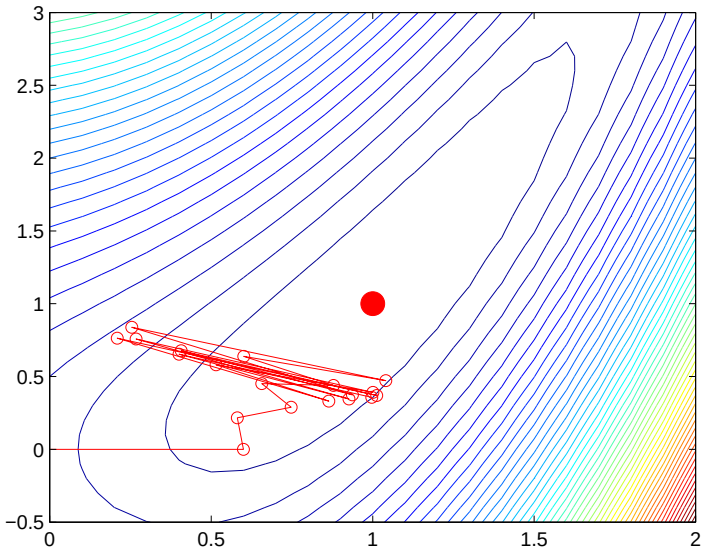
Seeing in action

Choose the right η is important

small η is too slow?



large η is too unstable?



How do we do this for logistic regression?

Simple fact: derivatives of $\sigma(a)$

$$\frac{d\,\sigma(a)}{d\,a} = \frac{d}{d\,a}\left(\frac{1}{1+e^{-a}}\right) = \frac{-(1+e^{-a})'}{(1+e^{-a})^2}$$

How do we do this for logistic regression?

Simple fact: derivatives of $\sigma(a)$

$$\frac{d\,\sigma(a)}{d\,a} = \frac{d}{d\,a}\left(\frac{1}{1+e^{-a}}\right) = \frac{-(1+e^{-a})'}{(1+e^{-a})^2}$$
$$= \frac{e^{a}}{(1+e^{-a})^2} = \frac{1}{1+e^{-a}}\left(1-\frac{1}{1+e^{-a}}\right)$$

How do we do this for logistic regression?

Simple fact: derivatives of $\sigma(a)$

$$\begin{aligned}\frac{d\,\sigma(a)}{d\,a} &= \frac{d}{d\,a}\left(\frac{1}{1+e^{-a}}\right) = \frac{-(1+e^{-a})'}{(1+e^{-a})^2} \\ &= \frac{e^{a}}{(1+e^{-a})^2} = \frac{1}{1+e^{-a}}\left(1-\frac{1}{1+e^{-a}}\right) \\ &= \sigma(a)[1-\sigma(a)] \\ \frac{d\,\log\sigma(a)}{d\,a} &= 1-\sigma(a)\end{aligned}$$

Gradients of the cross-entropy error function

Gradients

$$\frac{\partial \mathcal{E}(\boldsymbol{w})}{\partial \boldsymbol{w}} = -\sum_n \left\{ y_n [1 - \sigma(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n)] \boldsymbol{x}_n - (1 - y_n) \sigma(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n) \boldsymbol{x}_n \right\} \tag{6}$$

$$= \sum_n \left\{ \sigma(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n) - y_n \right\} \boldsymbol{x}_n \tag{7}$$

Remarks

- $e_n = \left\{ \sigma(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n) - y_n \right\}$ is called *error* for the nth training sample.
- Stationary point (in this case, the optimum):

$$\sum_n \sigma(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n) \boldsymbol{x}_n = \sum_n \boldsymbol{x}_n y_n$$

Intuition: *on average, the error is zero*.

Numerical optimization

Gradient descent

- Choose a proper step size $\eta > 0$

Numerical optimization

Gradient descent

- Choose a proper step size $\eta > 0$
- Iteratively update the parameters following the negative gradient to minimize the error function

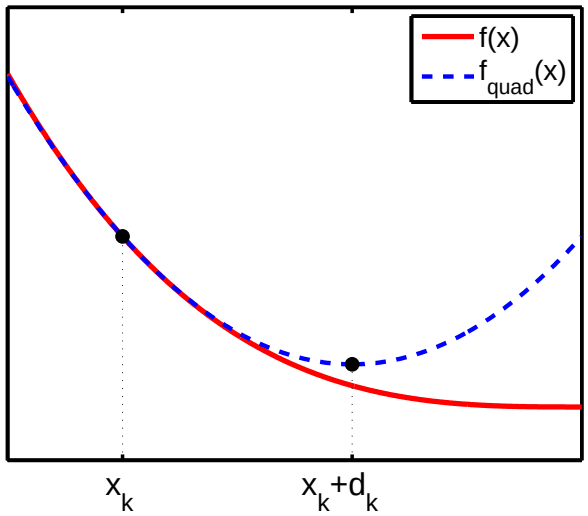
$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \eta \sum_n \left\{ \sigma(\boldsymbol{w}^{\mathrm{T}} \boldsymbol{x}_n) - y_n \right\} \boldsymbol{x}_n$$

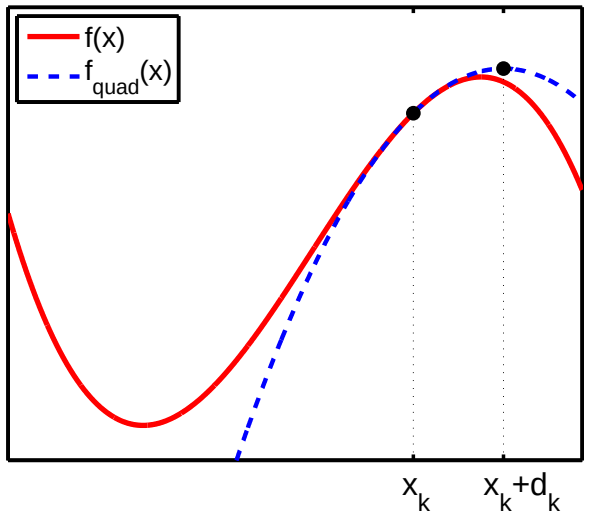
Remarks

- The step size needs to be chosen carefully to ensure convergence.
- The step size can be adaptive (i.e. varying from iteration to iteration). For example, we can use techniques such as *line search*
- There is a variant called *stochastic* gradient descent, also popularly used (later in this semester).

Intuition for Newton method

Approximate the true function with an easy-to-solve optimization problem





Approximation

Taylor expansion of the cross-entropy function

$$\mathcal{E}(\boldsymbol{w}) \approx \mathcal{E}(\boldsymbol{w}^{(t)}) + (\boldsymbol{w} - \boldsymbol{w}^{(t)})^{\mathrm{T}} \nabla \mathcal{E}(\boldsymbol{w}^{(t)}) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^{(t)})^{\mathrm{T}} \boldsymbol{H}^{(t)} (\boldsymbol{w} - \boldsymbol{w}^{(t)})$$

where

- $\nabla \mathcal{E}(\boldsymbol{w}^{(t)})$ is the gradient
- $\boldsymbol{H}^{(t)}$ is the Hessian matrix evaluated at $\boldsymbol{w}^{(t)}$

Example: a scalar function

$$\sin(\theta) \approx \sin(0) + \theta \cos(\theta = 0) + \frac{1}{2}\theta^2[-\sin(\theta = 0)] = \theta$$

where $\nabla \sin(\theta) = \cos(\theta)$ and $\boldsymbol{H} = \nabla \cos(\theta) = -\sin(\theta)$

So what is the Hessian matrix?

The matrix of second-order derivatives

$$\boldsymbol{H} = \frac{\partial^2 \mathcal{E}(\boldsymbol{w})}{\partial \boldsymbol{w} \boldsymbol{w}^{\mathrm{T}}}$$

In other words,

$$H_{ij} = \frac{\partial}{\partial w_j} \left(\frac{\partial \mathcal{E}(\boldsymbol{w})}{\partial w_i} \right)$$

So the Hessian matrix is $\mathbb{R}^{\mathsf{D} \times \mathsf{D}}$, where $\boldsymbol{w} \in \mathbb{R}^{\mathsf{D}}$.

Optimizing the approximation

Minimize the approximation

$$\mathcal{E}(\boldsymbol{w}) \approx \mathcal{E}(\boldsymbol{w}^{(t)}) + (\boldsymbol{w} - \boldsymbol{w}^{(t)})^{\mathrm{T}} \nabla \mathcal{E}(\boldsymbol{w}^{(t)}) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^{(t)})^{\mathrm{T}} \boldsymbol{H}^{(t)} (\boldsymbol{w} - \boldsymbol{w}^{(t)})$$

and use the solution as the new estimate of the parameters

$$\boldsymbol{w}^{(t+1)} \leftarrow \min_{\boldsymbol{w}} (\boldsymbol{w} - \boldsymbol{w}^{(t)})^{\mathrm{T}} \nabla \mathcal{E}(\boldsymbol{w}^{(t)}) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^{(t)})^{\mathrm{T}} \boldsymbol{H}^{(t)} (\boldsymbol{w} - \boldsymbol{w}^{(t)})$$

Optimizing the approximation

Minimize the approximation

$$\mathcal{E}(\boldsymbol{w}) \approx \mathcal{E}(\boldsymbol{w}^{(t)}) + (\boldsymbol{w} - \boldsymbol{w}^{(t)})^{\mathrm{T}} \nabla \mathcal{E}(\boldsymbol{w}^{(t)}) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^{(t)})^{\mathrm{T}} \boldsymbol{H}^{(t)} (\boldsymbol{w} - \boldsymbol{w}^{(t)})$$

and use the solution as the new estimate of the parameters

$$\boldsymbol{w}^{(t+1)} \leftarrow \min_{\boldsymbol{w}} (\boldsymbol{w} - \boldsymbol{w}^{(t)})^{\mathrm{T}} \nabla \mathcal{E}(\boldsymbol{w}^{(t)}) + \frac{1}{2}(\boldsymbol{w} - \boldsymbol{w}^{(t)})^{\mathrm{T}} \boldsymbol{H}^{(t)} (\boldsymbol{w} - \boldsymbol{w}^{(t)})$$

The quadratic function minimization has a *closed* form, thus, we have

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} - \left(\boldsymbol{H}^{(t)}\right)^{-1} \nabla \mathcal{E}(\boldsymbol{w}^{(t)})$$

i.e., the Newton method.

Contrast gradient descent and Newton method

Similar

Both are iterative procedures.

Difference

- Newton method requires second-order derivatives.
- Newton method does not have the magic η to be set.

Other important things about Hessian

Our cross-entropy error function is convex

$$\frac{\partial \mathcal{E}(\boldsymbol{w})}{\partial \boldsymbol{w}} = \sum_n \{\sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) - y_n\}\boldsymbol{x}_n \tag{8}$$

$$\Rightarrow \boldsymbol{H} = \frac{\partial^2 \mathcal{E}(\boldsymbol{w})}{\partial \boldsymbol{w}\boldsymbol{w}^{\mathrm{T}}} = \text{Take home exercise} \tag{9}$$

Other important things about Hessian

Our cross-entropy error function is convex

$$\frac{\partial \mathcal{E}(\boldsymbol{w})}{\partial \boldsymbol{w}} = \sum_n \{\sigma(\boldsymbol{w}^{\mathrm{T}}\boldsymbol{x}_n) - y_n\}\boldsymbol{x}_n \tag{8}$$

$$\Rightarrow \boldsymbol{H} = \frac{\partial^2 \mathcal{E}(\boldsymbol{w})}{\partial \boldsymbol{w}\boldsymbol{w}^{\mathrm{T}}} = \text{Take home exercise} \tag{9}$$

For any vector $\boldsymbol{v}$,

$$\boldsymbol{v}^{\mathrm{T}}\boldsymbol{H}\boldsymbol{v} = \text{Take home exercise} \geq 0$$

Thus, positive definite. Thus, the cross-entropy error function is convex, with only one global optimum.

Good about Newton method

Fast!

Suppose we want to minimize $f(x) = x^2 + 2x$ and we have its current estimate at $x^{(t)} \neq -1$. So what is the next estimate?

$$x^{(t+1)} \leftarrow x^{(t)} - [f''(x)]^{-1} f'(x) = x^{(t)} - \frac{1}{2}(2x^{(t)} + 2) = -1$$

Namely, the next step (of iteration) immediately tells us the global optimum! (In optimization, this is called *superlinear convergence rate*).

In general, the better our approximation, the faster the Newton method is in solving our optimization problem.

Bad about Newton method

Not scalable!

- Computing and inverting Hessian matrix can be very expensive for large-scale problems where the dimensionality D is very large.
- Newton method does not guarantee convergence if your starting point is far away from the optimum

NB. There are fixes and alternatives, such as Quasi-Newton/Quasi-second order method.