

# Machine Learning Principles

Class8 : Sept 28

Linear Classification II: Logistic Regression

Instructor: Diana Kim

# Today's Lecture

## 1. Modeling of Logistic Regression :

- $P[C_k|x]$ : sigmoid logistic function embedding “a linear regression function”
- decision rule/ boundary
- orientation and steepness of a sigmoid function

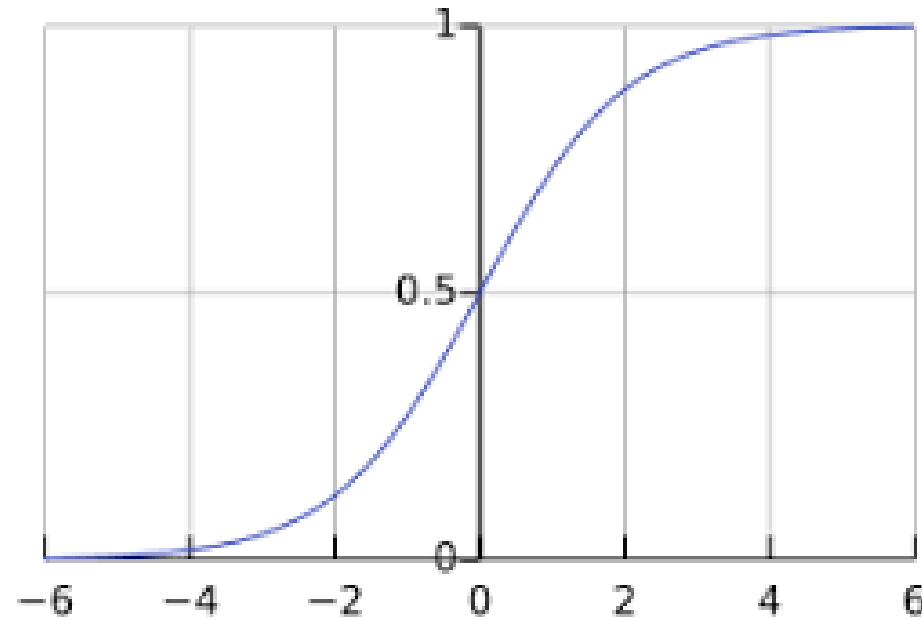
## 2. Training:

- defining objective Function (MLE) and its optimization
- overfitting and regularization

## 3. Multinomial (Multiclass) Logistic Regression

# [1] Logistic Regression (sigmoid function)

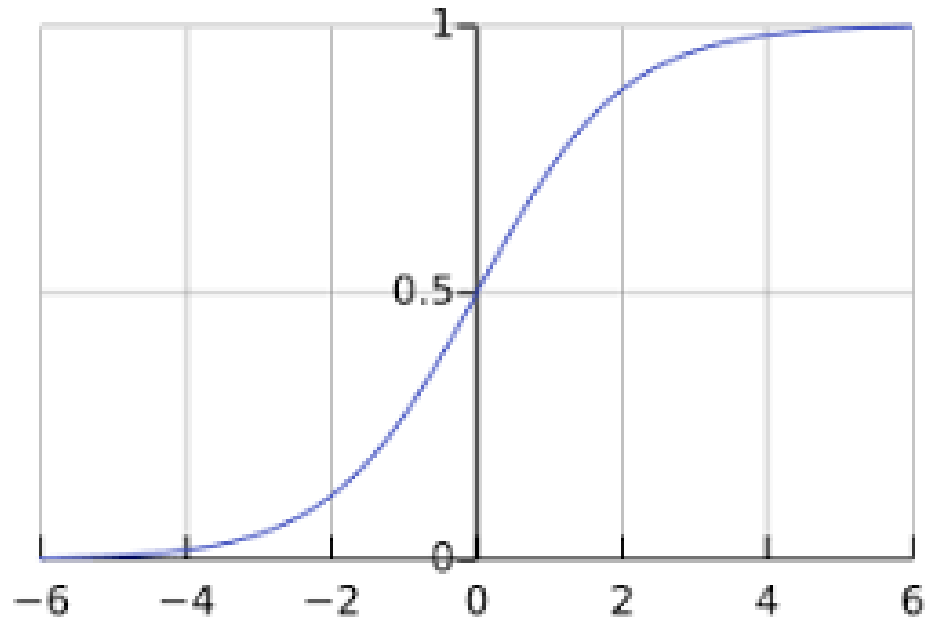
[logistic sigmoid function]



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

## [2] Logistic Regression (sigmoid function + linear function)

[logistic sigmoid function]



$$P[C_1|\vec{x}] = \frac{1}{1 + \exp(-\vec{w}^t \vec{x} - b)}$$
$$P[C_0|\vec{x}] = \frac{\exp(-\vec{w}^t \vec{x} - b)}{1 + \exp(-\vec{w}^t \vec{x} - b)}$$

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

[ decision rule]

$w^t x + b \geq 0$  then  $P[C_1|x] \geq P[C_0|x] \rightarrow x \in C_1$

$w^t x + b < 0$  then  $P[C_1|x] < P[C_0|x] \rightarrow x \in C_0$

### [3] Logistic Regression (example)

ex] suppose we learned logistic regression on the 2-D data space like below.  
what are the decision rule/boundary/ region?

$$P[C_1|x] = \sigma\left([1 \quad 2] \begin{bmatrix} x \\ y \end{bmatrix}\right) = \frac{1}{1 + \exp - [1 \quad 2] \begin{bmatrix} x \\ y \end{bmatrix}}$$

## [4] Logistic Regression (example)

ex] suppose we learned logistic regression on the 2-D data space like below.  
what are the decision rule/boundary/ region?

$$P[C_1|x] = \sigma\left([1 \quad 2] \begin{bmatrix} x \\ y \end{bmatrix}\right) = \frac{1}{1 + \exp - [1 \quad 2] \begin{bmatrix} x \\ y \end{bmatrix}}$$

- Decision Boundary (linear regression function/ hyperplane)
- The direction of the normal vector of the hyperplane
- The magnitude of the normal vector of the hyperplane

## [5] Logistic Regression (normal vector's direction and magnitude)

Q:  $w^t x + b = 0$  defines a decision boundary. do they define the same boundaries ?

$$(1) \quad \sigma\left(\begin{bmatrix} 1 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}\right) = \frac{1}{1 + \exp - \begin{bmatrix} 1 & 2 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}}$$

$$(2) \quad \sigma\left(\begin{bmatrix} 2 & 4 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}\right) = \frac{1}{1 + \exp - \begin{bmatrix} 2 & 4 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}}$$

$$(3) \quad \sigma\left(\begin{bmatrix} 3 & 6 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}\right) = \frac{1}{1 + \exp - \begin{bmatrix} 3 & 6 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}}$$

$$\|w\|(w'^t x + b') = 0$$

$$\text{where } \left\| \begin{bmatrix} w' \\ b' \end{bmatrix} \right\| = 1$$

There could be infinitely many solutions defining the same decision boundary for different  $\|w\|$ , but they differ in the steepness of their logistic sigmoid functions. The larger  $\|w\|$  gives the steeper sigmoid.

## [6] Logistic Regression (properties )

- The **direction** & **magnitude** of the normal vector of the hyperplane determines **orientation** & **steepness** of the sigmoid functions.
  - ✓  $\vec{w}x + b = 0$
  - ✓  $\vec{w}$ : normal vector (direction to be the class 1)

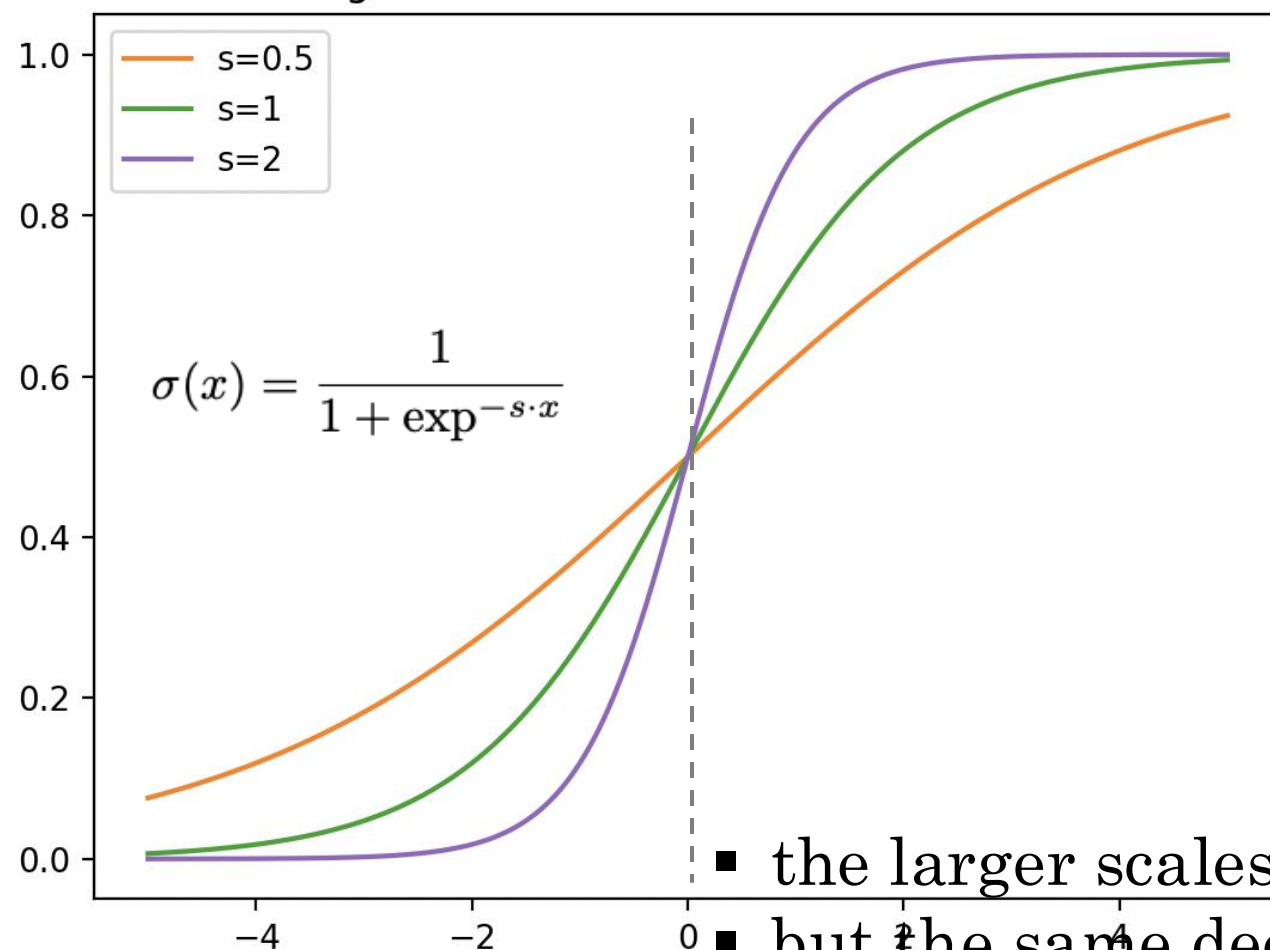
$$P[C_1|\vec{x}] = \frac{1}{1 + \exp(-\vec{w}^t \vec{x} - b)}$$

$$P[C_0|\vec{x}] = \frac{\exp(-\vec{w}^t \vec{x} - b)}{1 + \exp(-\vec{w}^t \vec{x} - b)}$$



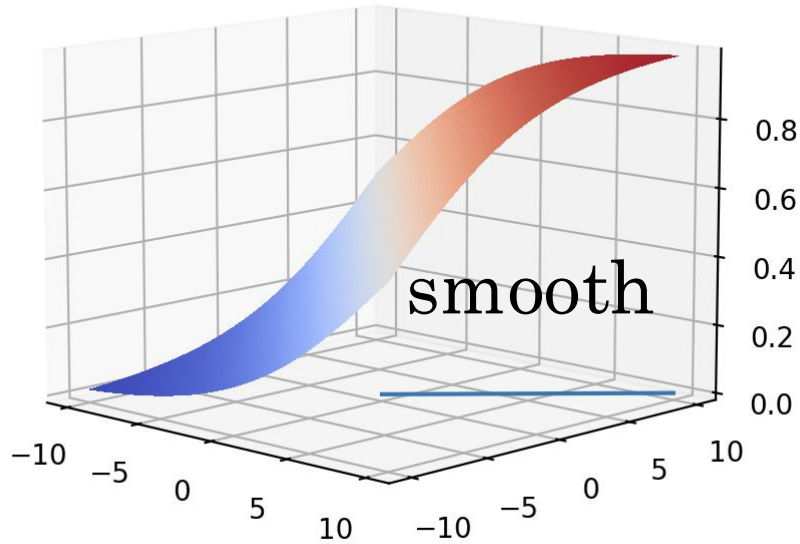
## **\*\*example 1D: sigmoid functions with different scales**

sigmoid function with different scales

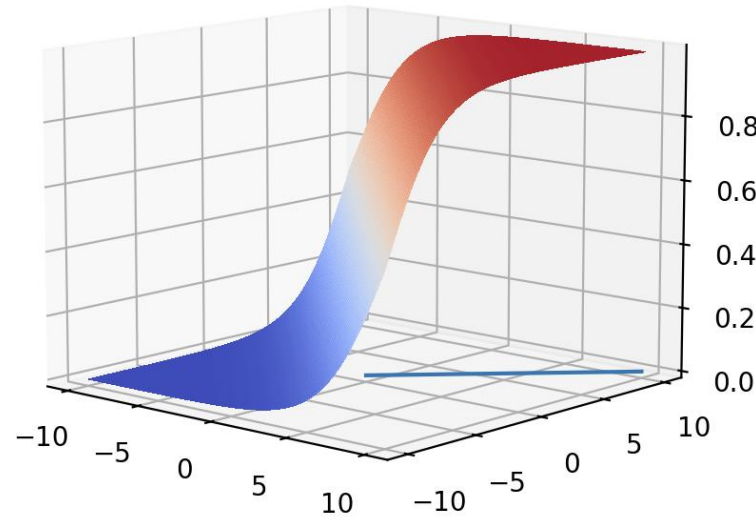


- the larger scales makes the sigmoid steeper.
- but the same decision boundary  $x = 0$

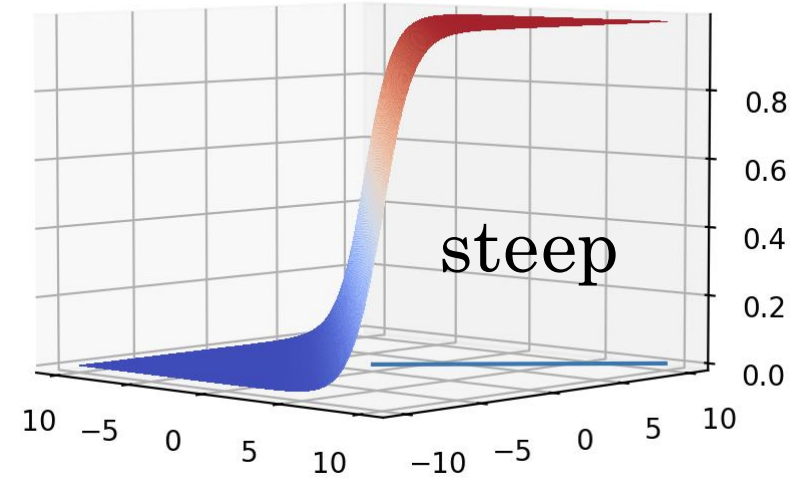
## \*\*example 2D: same decision region but different sigmoid functions



$$\vec{w} = (0.2, 0.2)$$



$$\vec{w} = (0.5, 0.5)$$



$$\vec{w} = (1, 1)$$

- Their decision boundaries are the same as  $X_1 + X_2 = 0$  but their steepness varies according to the magnitude of  $\|w\|$

- Training Logistic Regression

how do we learn the parameters  $\vec{w}$  and  $b$ ?

## [1] Training (Maximum Likelihood Estimator: MLE)

$$P(w|D) = \frac{p(w, D)}{P(D)} = \frac{p(D|w)p(w)}{p(D)}$$

$w \ast = \operatorname{argmax} P(D|w)$ : **Maximum Likelihood Estimation (MLE)**

## [2] Training (MLE)

- suppose we have a data set  $\{\vec{x}_n, t_n\}$  where  $t_n \in \{0,1\}$  &  $n = \{1,2, \dots, N\}$
- like 1,0, 0,...,1,0,1...
- $P(t_1, t_1, t_3, \dots t_N | \vec{w}, b, \vec{x}_n)$   
 $= \prod_{n=1}^N P(t_n | \vec{w}, b, \vec{x}_n) = P(t_n = 1 | \vec{w}, b, \vec{x}_n)^{t_n} P(t_n = 0 | \vec{w}, b, \vec{x}_n)^{1-t_n}$

Q: how we modeled  $P(t_n = 1 | \vec{w}, b)$  in logistic regression?

### [3] Training (cross entropy loss)

$$P(t_1, \dots, t_N | w) = \prod_{n=1}^N \sigma(w^t x_n)^{t_n} (1 - \sigma(w^t x_n))^{1-t_n}$$

$$J(\vec{w}) = -\ln P(t_1, \dots, t_N | w) = \sum_{n=1}^N -t_n \ln \sigma(w^t x_n) - (1 - t_n) \ln (1 - \sigma(w^t x_n))$$

$$w^* = \arg \min_w J(\vec{w})$$

[cross entropy loss]

## \*\* Entropy / Cross Entropy / KL Divergence (Kullback-Leibler)

$$H(X) = E[\log \frac{1}{P(X)}] = \sum_x p(x) \log \frac{1}{p(x)}$$

- **[Entropy]:**  
the smallest number bits to encode R.V  $X$

$$\begin{aligned} D(p||q) &= \sum_x p(x) \log \frac{p(x)}{q(x)} \\ &= \sum_x p(x) \log \frac{1}{q(x)} - \sum_x p(x) \log \frac{1}{p(x)} \end{aligned}$$

- **[KL divergence]:**  
how many bits more needed when using  $q(x)$  instead of the original density  $p(x)$ ?  
■ this measures  
the difference/distance  
between the two densities:  $p(x)$  and  $q(x)$

## \*\* Entropy / Cross Entropy / KL Divergence (Kullback-Leibler)

- KL divergence is always non-negative!

$$D(p||q) = \sum_x p(x) \log \frac{p(x)}{q(x)} \geq 0$$

$$\begin{aligned} -D(p||q) &= \sum_x p(x) \log \frac{q(x)}{p(x)} \quad [\text{by Jensen's Inequality}] \\ &\leq \log \sum_x \frac{p(x)q(x)}{p(x)} = \log 1 = 0 \end{aligned}$$



## \*\* Entropy / Cross Entropy / KL Divergence (Kullback-Leibler)

$$\begin{aligned} H(p, q) &= \sum_x p(x) \log \frac{1}{q(x)} \\ &= \sum_x p(x) \log \frac{p(x)}{q(x)p(x)} = \sum_x p(x) \log \frac{p(x)}{q(x)} + \sum_x p(x) \log \frac{1}{p(x)} \\ &= D(p||q) + H(X) \end{aligned}$$

- **[Cross Entropy]**

minimizing cross entropy directly minimizes KL divergence.

## [4] Training (cross entropy)

$$P(t_1, \dots, t_N | w) = \prod_{n=1}^N \sigma(w^t x_n)^{t_n} (1 - \sigma(w^t x_n))^{1-t_n}$$

$$J(\vec{w}) = -\ln P(t_1, \dots, t_N | w) = \sum_{n=1}^N -t_n \ln \sigma(w^t x_n) - (1 - t_n) \ln (1 - \sigma(w^t x_n))$$

$$w^* = \arg \min_w J(\vec{w})$$

[cross entropy loss]

- as taking the ground truth (1 / 0) as  $p(t)$   
minimizing the cross entropy minimizes the distance  
between  $p(t)$  and  $q(t) = \sigma(w^t x)$

## [5] Training (computing Gradient)

$$\begin{aligned}\nabla_w J(\vec{w}) &= \sum_{n=1}^N -t_n \frac{\sigma(w^t x_n)(1 - \sigma(w^t x_n))}{\sigma(w^t x_n)} x_n - (1 - t_n) \frac{-\sigma(w^t x_n)(1 - \sigma(w^t x_n))}{1 - \sigma(w^t x_n)} x_n \\ &= \sum_{n=1}^N \{-t_n(1 - \sigma(w^t x_n)) - (1 - t_n)(-\sigma(w^t x_n))\} x_n \\ &= \sum_{n=1}^N (\sigma(w^t x_n) - t_n) x_n\end{aligned}$$

## \*\*useful properties of logistic sigmoid function

- symmetric & complement probability

$$\sigma(-x) = 1 - \sigma(x)$$

$$\sigma(x) = 1 - \sigma(-x)$$

- derivative

$$\begin{aligned}\frac{d}{dx}\sigma(x) &= \frac{\exp^{-x}}{(1 + \exp^{-x})^2} \\ &= \frac{1}{(1 + \exp^{-x})} \cdot \frac{\exp^{-x}}{(1 + \exp^{-x})} \\ &= \frac{1}{(1 + \exp^{-x})} \cdot \frac{1 + \exp^{-x} - 1}{(1 + \exp^{-x})} \\ &= \sigma(x) \cdot (1 - \sigma(x))\end{aligned}$$

## [6] Training (computing Hessian)

$$\nabla_w^2 J(\vec{w}) = \sum_{n=1}^N \sigma(w^t x_n) \cdot (1 - \sigma(w^t x_n)) x_n^t x_n \succeq 0$$

non-negative definite

- two possible cases of  $\nabla^2 J(w) = 0$ 
  - no minimum solution  $\vec{w}^*$  when data is linearly separable.
  - multiple minimum points: covariance matrix  $X^t X$  is singular, collinearity exist.

## [7] Training (computing Hessian)

$$\nabla_w^2 J(\vec{w}) = \sum_{n=1}^N \sigma(w^t x_n) \cdot (1 - \sigma(w^t x_n)) x_n^t x_n \succeq 0$$

non-negative definite

- unique solution:  $\nabla^2 J(w) \succ 0$
- when data is not linearly separable and  $X^t X$  is full-rank matrix.  
then  $\nabla^2 J(w) \succ 0 \rightarrow \nabla J(w^*) = 0$  becomes a sufficient condition  
to be an optimal solution.

## [8] Training (summary)

$\nabla^2 J(w) = 0$		$\nabla^2 J(w) > 0$
data is linearly separable	collinearity $X^t X$ is singular	data is not linearly separable & $X^t X$ is a full rank matrix
no minimum exist. $s.t \nabla J(w *) = 0$  $  w  $ goes to $\infty$	multiple $w$ is possible $s.t \nabla J(w *) = 0$  multiple, global minimum	unique solution exist $s.t \nabla J(w *) = 0$  unique, global minimum

## [9] Training (no closed solution $\nabla J(\vec{w}) = 0$ )

Let's try finding the optimal point  $\vec{w}^*$  by  $\nabla J(\vec{w}) = 0$ !

$$\begin{aligned}\nabla_w J(\vec{w}) &= \sum_{n=1}^N -t_n \frac{\sigma(w^t x_n)(1 - \sigma(w^t x_n))}{\sigma(w^t x_n)} x_n - (1 - t_n) \frac{-\sigma(w^t x_n)(1 - \sigma(w^t x_n))}{1 - \sigma(w^t x_n)} x_n \\ &= \sum_{n=1}^N \{-t_n(1 - \sigma(w^t x_n)) - (1 - t_n)(-\sigma(w^t x_n))\} x_n \\ &= \sum_{n=1}^N (\sigma(w^t x_n) - t_n) x_n = 0\end{aligned}$$

no closed solution for  $\nabla_w J(w) = 0$



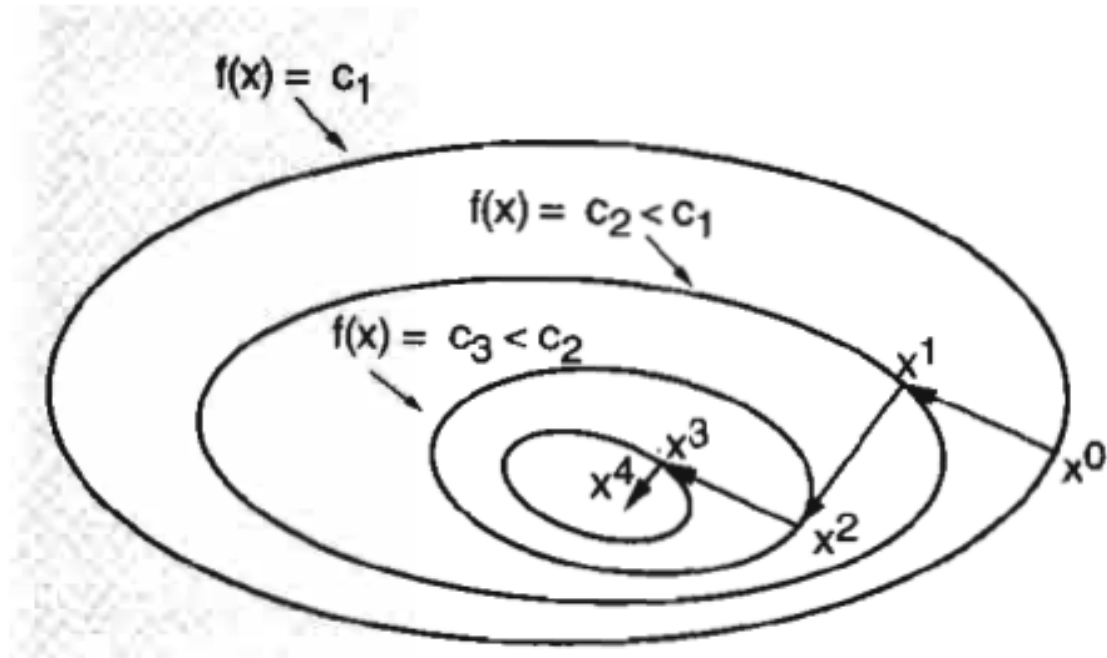
- Steepest Gradient Descent Algorithm  
(an iterative optimization method to minimize  $J(\vec{w})$ )

# [1] Steepest Gradient Descent Algorithm

- it finds **iteratively** a local/ global minimum of  $J(\vec{w})$
- it computes the steepest descent direction at the current and take a step to that direction as amount of predefined and compute the next direction again

$$w_{i+1} = w_i - \eta \nabla J(w) \quad (\text{gradient gives the steepest \& increasing direction})$$

## [2] Steepest Gradient Descent Algorithm (direction by gradient)



**Figure 1.2.1.** Iterative descent for minimizing a function  $f$ . Each vector in the generated sequence has a lower cost than its predecessor.

[Nonlinear Programming, Dimitri P. Bertsekas]

- gradient direction is orthogonal to contours.

- Training Logistic Regression using Gradient Steepest Descent

## [1] Training (gradient/steepest decent algorithm )

$$w_{i+1} = w_i - \eta \nabla J(w) \quad (\text{gradient gives the steepest ascent direction})$$

$$\begin{aligned} \nabla_w J(\vec{w}) &= \sum_{n=1}^N -t_n \frac{\sigma(w^t x_n)(1 - \sigma(w^t x_n))}{\sigma(w^t x_n)} x_n - (1 - t_n) \frac{-\sigma(w^t x_n)(1 - \sigma(w^t x_n))}{1 - \sigma(w^t x_n)} x_n \\ &= \sum_{n=1}^N \{-t_n(1 - \sigma(w^t x_n)) - (1 - t_n)(-\sigma(w^t x_n))\} x_n \\ &= \sum_{n=1}^N (\sigma(w^t x_n) - t_n) x_n \end{aligned}$$

## [5] Training (convergence?)

$\nabla^2 J(w) = 0$		$\nabla^2 J(w) > 0$
data is linearly separable	collinearity $X^t X$ is singular	data is not linearly separable & $X^t X$ is a full rank matrix
no minimum exist. $s.t \nabla J(w *) = 0$  $\ w\ $ goes to $\infty$	multiple $w$ is possible $s.t \nabla J(w *) = 0$  global minimum	unique solution exist $s.t \nabla J(w *) = 0$  global minimum
<p>does gradient descent algorithm converge? (if tolerance set, improvement <math>&lt; \epsilon</math> &amp; if we set the proper step size.)</p>		

- Overfitting

MLE is sensitive to data  
(# data points & data structure)

## [1] Overfitting Scenario (limited data size)

Like regression model,  
logistic regression classifier **overfits to train data**  
**when too many features relative to # training samples.**

Q: even worse, when the training dataset is linearly separable, the overfitted model tends to have excessively high confidence in its predictions.



## [2] Overfitting Scenario (steep logistic sigmoid)

the steep sigmoid function would be appropriate,

- if we have a well-designed feature map making data linearly separable.
- if we we have a sufficient number of data points.
- however, hard to achieve the condition in practice.

- Regularization (MAP rule)

$$w \ast = \operatorname{argmax} p(w|D) \propto p(D|w)p(w)$$

## [1] Regularization (ridge logistic regression)

$$p(w): \vec{w} \sim N(0, \sigma^2 I)$$

prior

$$P(t_1, \dots, t_N | w) P(w) = \prod_{n=1}^N \sigma(w^t x_n)^{t_n} (1 - \sigma(w^t x_n))^{1-t_n} \cdot \frac{1}{\sqrt{2\pi\sigma^{2M}}} \cdot \exp^{-\frac{1}{2\sigma^2} \|W\|^2}$$

$$-\ln P(t_1, \dots, t_N | w) P(w) = \sum_{n=1}^N -t_n \ln \sigma(w^t x_n) - (1 - t_n) \ln (1 - \sigma(w^t x_n)) - \ln \frac{1}{\sqrt{2\pi\sigma^{2M}}} + \frac{1}{2\sigma^2} \|W\|^2$$

$$J(w) = \sum_{n=1}^N -t_n \ln \sigma(w^t x_n) - (1 - t_n) \ln (1 - \sigma(w^t x_n)) + \frac{1}{2\sigma^2} \|W\|^2$$

$$J(w) = \sum_{n=1}^N -t_n \ln \sigma(w^t x_n) - (1 - t_n) \ln (1 - \sigma(w^t x_n)) + \lambda \cdot \|W\|^2$$

regularization parameters /  
set using cross validation.

## [2] Regularization (gradient)

- $\nabla J(\vec{w})$

$$\begin{aligned}\nabla_w J(\vec{w}) &= \sum_{n=1}^N -t_n \frac{\sigma(w^t x_n)(1 - \sigma(w^t x_n))}{\sigma(w^t x_n)} x_n - (1 - t_n) \frac{-\sigma(w^t x_n)(1 - \sigma(w^t x_n))}{1 - \sigma(w^t x_n)} x_n + 2\lambda \vec{w} \\ &= \sum_{n=1}^N \{-t_n(1 - \sigma(w^t x_n)) - (1 - t_n)(-\sigma(w^t x_n))\} x_n + 2\lambda \vec{w} \\ &= \sum_{n=1}^N (\sigma(w^t x_n) - t_n) x_n + 2\lambda \vec{w}\end{aligned}$$

---

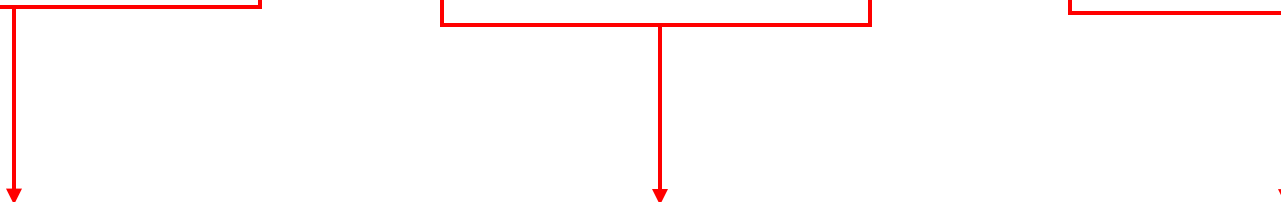
- gradient descent

$$w_{i+1} = w_i - \eta \nabla J(w)$$

- Multinomial Logistic Regression  
(Learning  $K$  posterior  $P[C_k|x]$ )

## [1] Multinomial Logistic Regression (posterior using linear functions)

$$P[C_0|x] = \frac{P[x|C_0]P[C_0]}{P[x|C_0]P[C_0] + P[x|C_1]P[C_1] + P[x|C_2]P[C_2]}$$

$$P[C_0|x] = \frac{\exp[\ln P[x|C_0]P[C_0]]}{\exp[\ln P[x|C_0]P[C_0]] + \exp[\ln P[x|C_1]P[C_1]] + \exp[\ln P[x|C_2]P[C_2]]}$$


when Gaussian assumed, these log elements are quadratic.

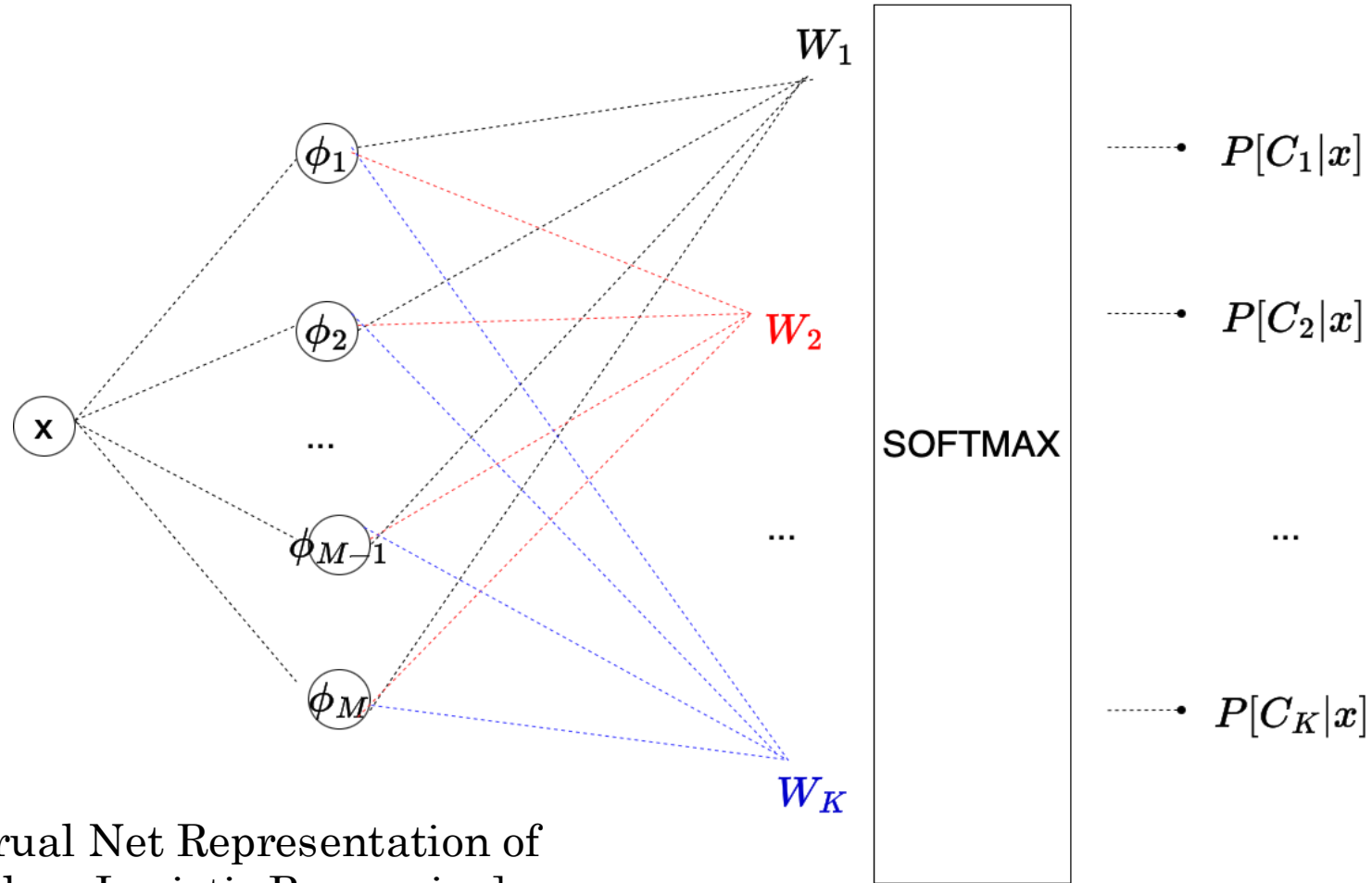
But **each exponent becomes a linear function:  $w^t x + b$** ; the second order terms are cancelled out when covariances are equal.

## [2] Multinomial Logistic Regression (softmax function)

$$P[C_k|x] = \frac{\exp w_k^t \phi(x)}{\sum_{k=1}^K \exp w_k^t \phi(x)}$$

- using normalized exponential (softmax) and  $K$  linear functions, the  $K$  posteriors are represented.

### [3] Multinomial Logistic Regression (softmax function)



[the Nerual Net Representation of  
Multiclass Logistic Regression]



## [4] Multinomial Logistic Regression (softmax example)

ex] compute  $P[C_k|x]$  using softmax

- $x = 1$  (one observation example, this can be any real number)
- $\phi(x): \{1, x, x^2\}$
- $W = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \\ 1 & 2 & 3 & 4 & 5 \end{bmatrix}$

## [5] Multinomial Logistic Regression

Multiclass logistic regression learns  $K$  discriminant linear functions  
the functions are translated into posterior by softmax  
(deep neural net classifier has the same structure in the last layer)

Q: how can we learn the discriminant linear functions?

## [6] Multinomial Logistic Regression (training)

$$P[C_k|x] = \frac{\exp w_k^t \phi(x)}{\sum_{k=1}^K \exp w_k^t \phi(x)}$$

- then how to learn the parameters  $\vec{w}$ ?
- MLE!  
we plug in the posteriors in  $k$ -categorical density and estimate the parameter using MLE as we did in binary logistic regression.

## [7] Multinomial Logistic Regression (cross entropy based on MLE)

- likelihood

$$P[T|W, x] = \prod_{n=1}^N \prod_{k=1}^K P[C_k|W, x_n]^{t_{nk}}$$

- negative-log  
(cross entropy)

$$J(W) = -\ln P[T|W, x] = -\sum_{n=1}^N \sum_{k=1}^K t_{nk} \ln P[C_k|W, x_n]$$

## [8] Multinomial Logistic Regression (softmax gradient)

- $\nabla P[C_j|x] \text{ r.t. } w_i \text{ (} i=j \text{)}$

$$P[C_j|x] = \frac{\exp w_j^t \phi(x)}{\sum_{k=1}^K \exp w_k^t \phi(x)}$$
$$\nabla_{w_j} P[C_j|x] = \left\{ \frac{(\sum_{k=1}^K \exp w_k^t \phi(x))(\exp w_j^t \phi(x)) - (\exp w_j^t \phi(x))^2}{(\sum_{k=1}^K \exp w_k^t \phi(x))^2} \right\} \phi(x)$$
$$\nabla_{w_j} P[C_j|x] = \left\{ \frac{\exp w_j^t \phi(x)}{\sum_{k=1}^K \exp w_k^t \phi(x)} \cdot \left(1 - \frac{\exp w_j^t \phi(x)}{\sum_{k=1}^K \exp w_k^t \phi(x)}\right) \right\} \phi(x)$$

- $\nabla P[C_j|x] \text{ r.t. } w_i \text{ (} i \neq j \text{)}$

$$P[C_j|x] = \frac{\exp w_j^t \phi(x)}{\sum_{k=1}^K \exp w_k^t \phi(x)}$$
$$\nabla_{w_i} P[C_j|x] = \left\{ \frac{-(\exp w_j^t \phi(x))(\exp w_i^t \phi(x))}{(\sum_{k=1}^K \exp w_k^t \phi(x))^2} \right\} \phi(x)$$
$$\nabla_{w_i} P[C_j|x] = \left\{ \frac{\exp w_j^t \phi(x)}{\sum_{k=1}^K \exp w_k^t \phi(x)} \cdot \left(-\frac{\exp w_i^t \phi(x)}{\sum_{k=1}^K \exp w_k^t \phi(x)}\right) \right\} \phi(x)$$

## [9] Multinomial Logistic Regression (softmax gradient for gradient descent)

$$\nabla_{W_j} P[C_j|x] = P[C_j|x] \cdot (1 - P[C_j|x])\phi(x) \quad (i=j)$$

$$\nabla_{W_i} P[C_j|x] = P[C_j|x] \cdot (-P[C_i|x])\phi(x) \quad (i \neq j)$$

Q: can we compute  $P[C_j|x]$  ?

yes, during training, we can compute  $P[C_j|x]$  using the current parameter  $W$ !

## [10] Multinomial Logistic Regression (cross entropy gradient)

$$\begin{aligned}\nabla_{W_j} J(W_1, W_2, \dots, W_K) &= - \sum_{n=1}^N \sum_{k=1}^K t_{nk} \nabla_{W_j} \ln P[C_k | \phi(x)] \\&= \sum_{n=1}^N t_{n1} P[C_j | \phi(x)] + \dots + t_{nj} (P[C_j | \phi(x)] - 1) + \dots + t_{nK} P[C_j | \phi(x)] \\&= \sum_{n=1}^N \{ P[C_j | \phi(x)] (t_{n1} + t_{n2} + \dots + t_{nK}) - t_{nj} \} \phi(x) \\&= \sum_{n=1}^N \{ P[C_j | \phi(x)] - t_{nj} \} \phi(x)\end{aligned}$$

sum = 1

## [11] Multinomial Logistic Regression (gradient descent)

$$w_{i+1} = w_i - \eta \nabla J(w) \qquad \nabla J(W) = \begin{bmatrix} \nabla_{W_1} J(W) \\ \nabla_{W_2} J(W) \\ \dots \\ \dots \\ \nabla_{W_{K-1}} J(W) \\ \nabla_{W_K} J(W) \end{bmatrix}$$