CS 461: Machine Learning Principles

Class 10: Oct. 7

Perceptron & Multiclass Logistic Regression

Instructor: Diana Kim

In the last two classes,
we studied how the two discriminant functions for binary classifications can be designed by learning $P[C_0 \mid x]$ and $P[C_1 \mid x]$ $(posterior)$

- Generative: GDA, Naïve Bayes

- Discriminative: Logistic Regression

Both aim to learn posterior and use it in inference stage but
generative estimates cov and mean
discriminative directly learns the posterior

The discriminant functions were the linear functions under some assumptions (shared covariance)

- binary decision rule

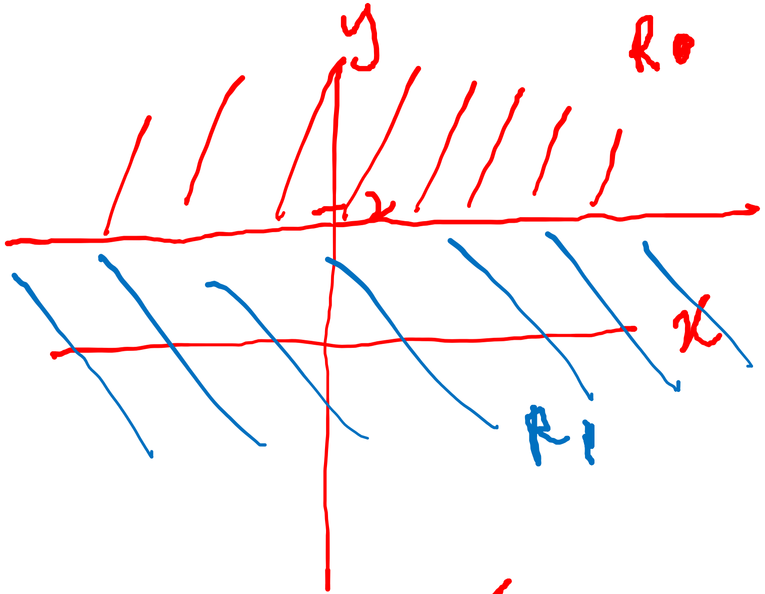
$$f_0(x) \underset{H_1}{\overset{H_0}{\gtreqless}} f_1(x)$$

- one decision boundary : a hyperplane

$$\boxed{\overrightarrow{W}\phi(\mathrm{x}) = 0}$$

Classification algorithms learn discriminant functions and they define the hyperplanes for decision boundaries.

Example) Suppose we learned two discriminant functions over $2-D$ space (x, y). Compute the hyperplane to define the decision boundary.



$$\underline{\overset{f_0(x)}{2x + y - 1}} \underset{H_1}{\overset{H_0}{\gtrless}} \underline{\overset{f_1(x)}{x + y + 1}}$$

$$f_0(x) - f_1(x) \Rightarrow 2x + \cancel{y} - 1 - x - \cancel{y} - 1 = 0$$

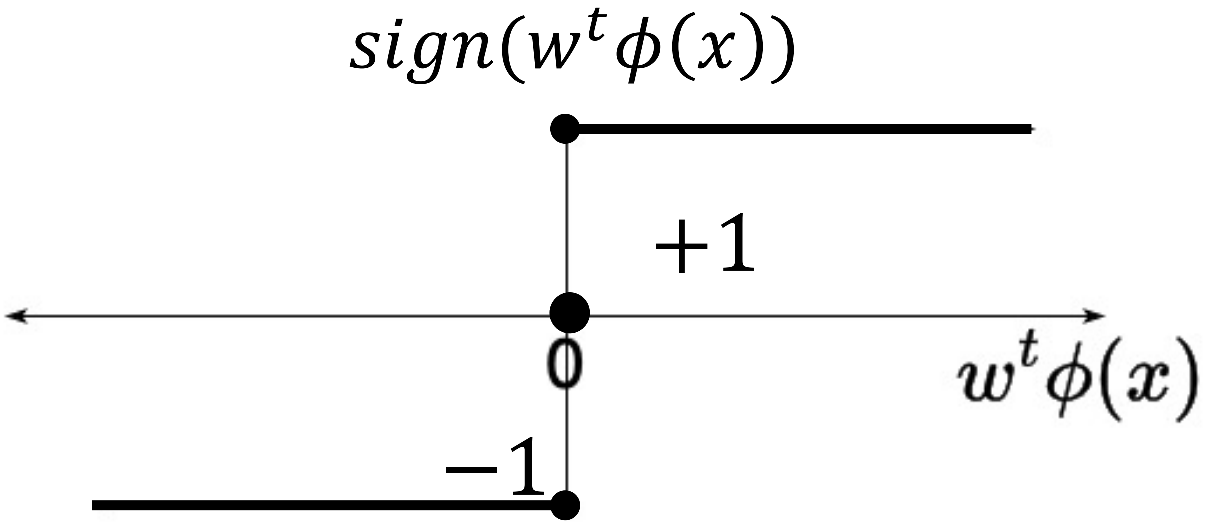
$$\Rightarrow x - 2 = 0$$

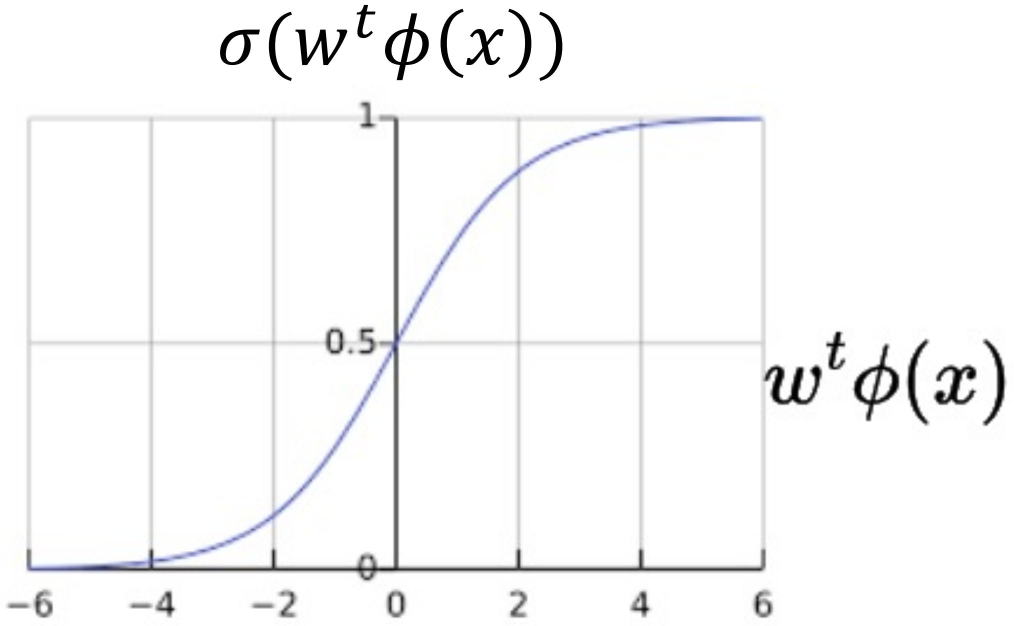
Today we are going to study **Perceptron Algorithm** [Rosenblatt (1962)]
It learns a decision boundary for binary classification
without considering any probabilistic modeling.

$$f(x) \underset{H_1}{\overset{H_0}{\gtreqless}} 0$$

- Perceptron algorithm directly learns a hyperplane (decision boundary).
- Perceptron algorithm does not give any probabilistic interpretation.

Perceptron outcome does not give probabilistic interpretation.





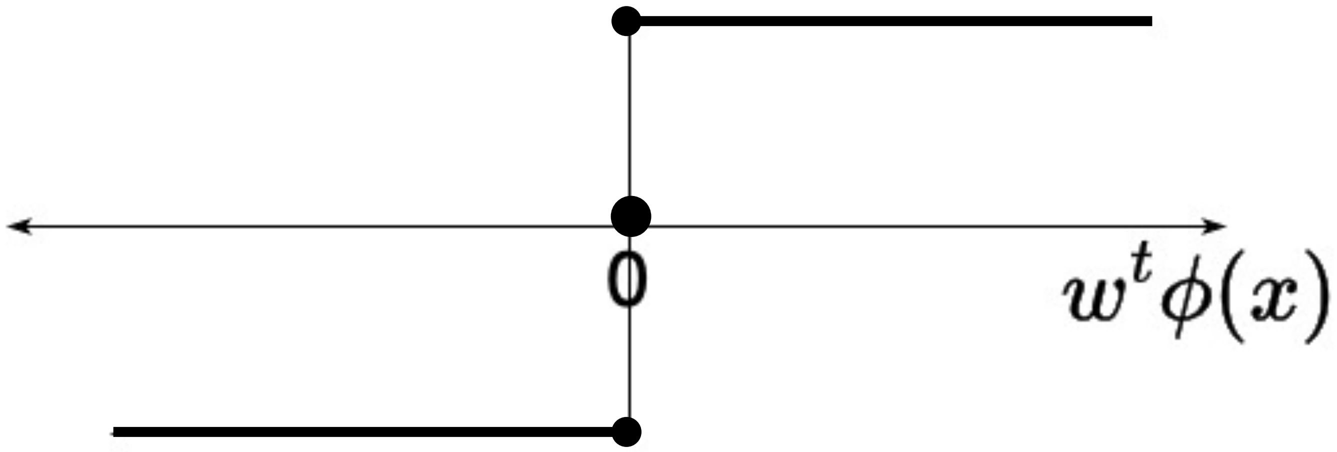
- Perceptron Activation

: <u>hard</u> decision (non-probabilistic)

- Logistic Regression Activation

: <u>soft</u> decision (probabilistic)

Activation Function $sign(w^t\phi(x))$ for Perceptron



$$P(t|w) = \prod_{n=1} \sigma(w^t x_n)^{t_n}(1-\sigma(w^t x_n))^{1-t_n}$$

In logistic regression, it learns $\sigma((w^t\phi(x))$ by using MLE for its probabilistic representation.
How about perceptron? How can we learn the $sign\ (w^t\phi(x))$?

- Suppose we have a data set $\{x_n, t_n\}$ $where\ t_n \in \{-1, +1\}\ and\ n = \{1,2, \dots, N\}$
- One way to define the objective function to minimize is "misclassification error"

$$E(\vec{w}) = \frac{\sum_{n=1}^{N} ||f(\vec{w}^t \phi(x)) - t_n||^2}{N}$$

Q: Is the objective function differentiable?

+ the objective function is the piece-wise constant function, so non-differentiable.

Perceptron Algorithm [Rosenblatt (1962)]

- The Objective Function (to minimize) in Perceptron

$$E(\vec{w}) = -\sum_{n\in\mathcal{M}} \vec{w}^t\phi(x_n)\cdot t_n - \sum_{n\in\mathcal{M}^c} \cdot 0$$

$\mathcal{M}$: misclassification Samples

- Motivation : increase $(w^t\phi(x_n)t_n) > 0$

- Gradient

$$-\sum_{n\in\mathcal{M}} \phi(\vec{x_n})t_n$$

Perceptron Algorithm

$$E(\vec{w}) = -\sum_{n \in \mathcal{M}} \vec{w}^t \phi(x_n) \cdot t_n - \sum_{n \in \mathcal{M}^c} \cdot 0$$

$\mathcal{M}$: Misclassification Samples

- Gradient (steepest increase direction)

$$-\sum_{n \in \mathcal{M}} \phi(\vec{x_n}) t_n$$

- Update rule

- $t_n > 0$: *add* positive
- $t_n < 0$: *subtract* negative sample

$$w(t+1) = w(t) - \eta \nabla E(w) = w(t) + \eta \cdot \boxed{\phi(\vec{x_n}) t_n}$$

Perceptron Algorithm

Perceptron Algorithm

Initialize $\vec{w} = \vec{0}$
while TRUE **do**
  $m = 0$
  for $(x_i, y_i) \in D$ **do**
    if $y_i(\vec{w}^T \cdot \vec{x}_i) \leq 0$ **then**
      $\boxed{\vec{w} \leftarrow \vec{w} + y\vec{x}}$ update one sample by one!
      $m \leftarrow m + 1$
    end if
  end for
  if $m = 0$ **then**
    break
  end if
end while

Source from: https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote03.html

Ex) Assume a data set consists only of a single data point { (x, +1) }. How often can a Perceptron misclassify this point x repeatedly? What if the initial vector w was initialized randomly and not as the all-zero vector?

Perceptron Algorithm

Initialize $\vec{w} = \vec{0}$
while TRUE **do**
  $m = 0$
  for $(x_i, y_i) \in D$ **do**
    if $y_i(\vec{w}^T \cdot \vec{x}_i) \leq 0$ **then**
      $\vec{w} \leftarrow \vec{w} + y\vec{x}$
      $m \leftarrow m + 1$
    end if
  end for
  if $m = 0$ **then**
    break
  end if
end while

Suppose $W_0 = \{0,0\}$ $\rightarrow$ $[0.\ 0] \cdot \begin{bmatrix} x \end{bmatrix} \cdot (+1) = 0$

$w^T x \cdot y = 0$

$\times \rightarrow$ misclassification

$W_1 = w_0 + \eta \cdot (+1) \cdot x$

$= \eta \cdot x$

$w_1^T \cdot x \cdot y$

$= \eta \cdot x^T \cdot x \cdot 1$

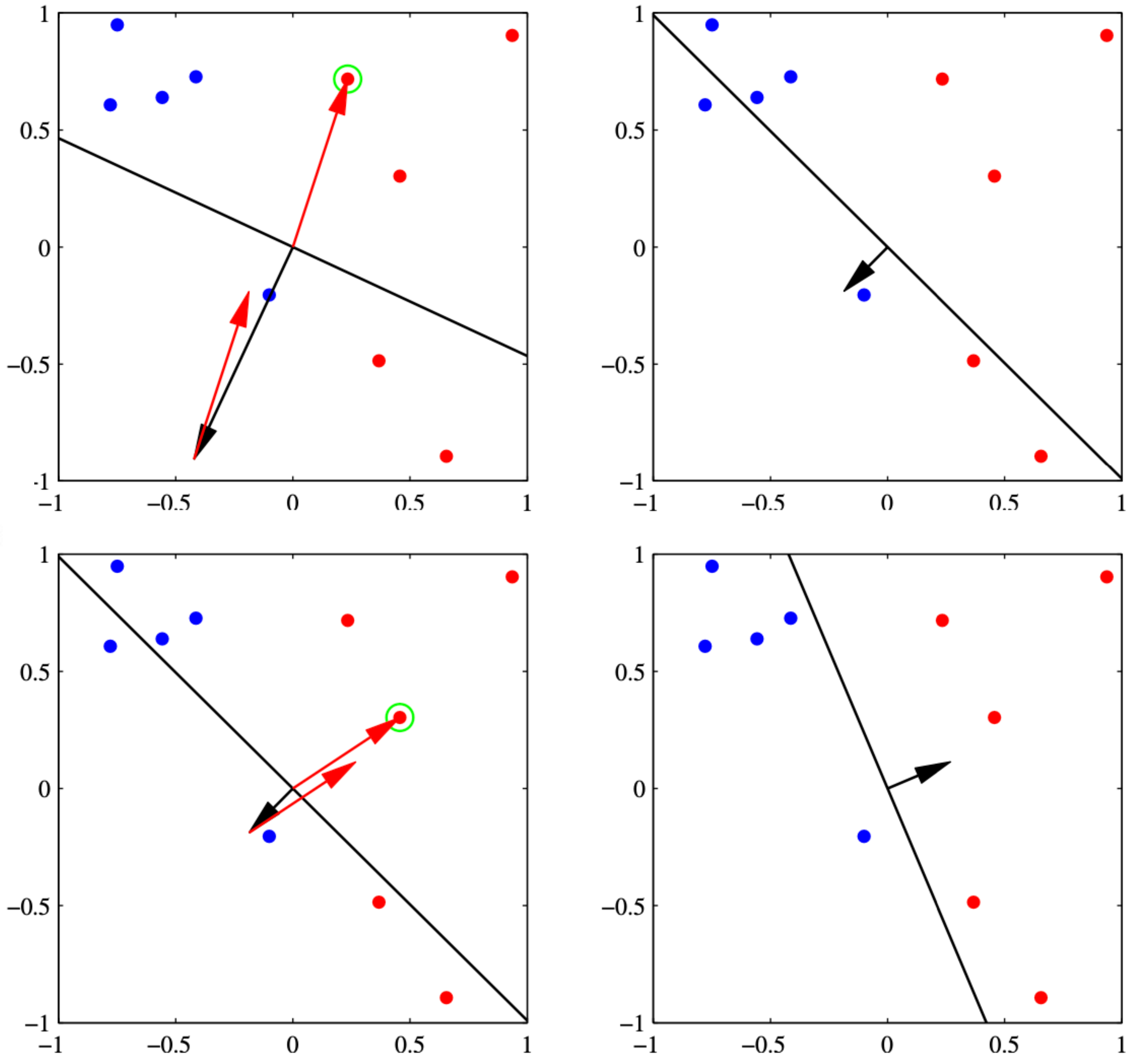
$= \eta \cdot \|x\|^2 > 0$

Source from: https://www.cs.cornell.edu/courses/cs4780/2018fa/lectures/lecturenote03.html

- Update rule

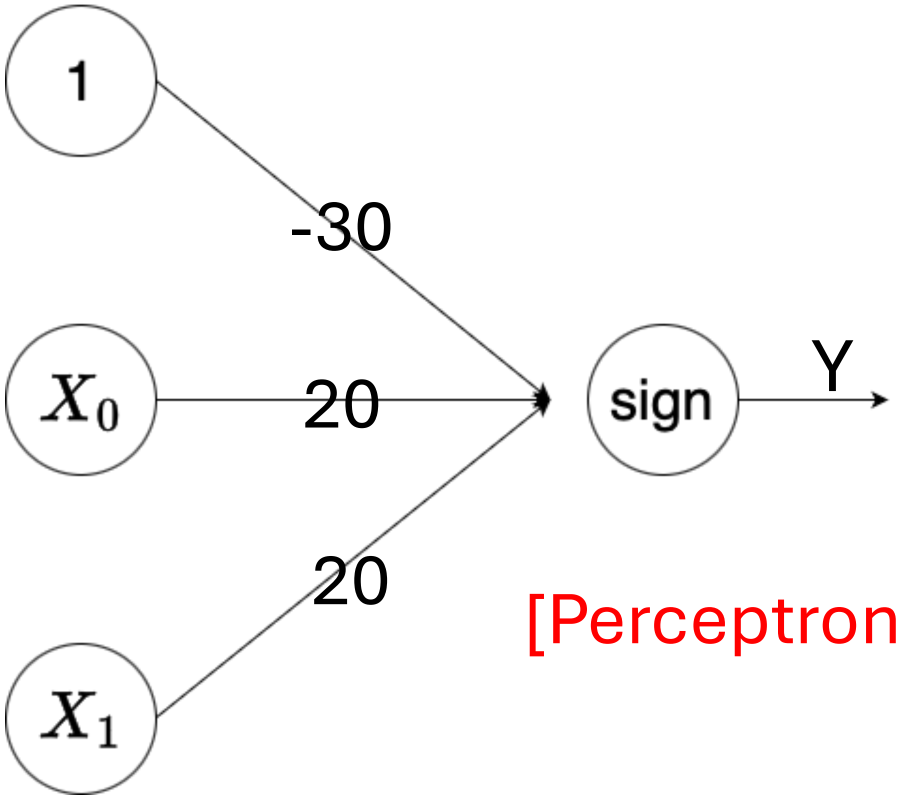
$$w(t+1) = w(t) - \eta \nabla E(w) = w(t) + \eta \cdot \phi(\vec{x_n}) t_n$$

The Convergence of Perceptron from Text Bishop Figure 4.7



Perceptron Convergence theorem

If training data set is linearly separable,
then the perceptron algorithm is guaranteed
to find a solution in finite number of iterations.



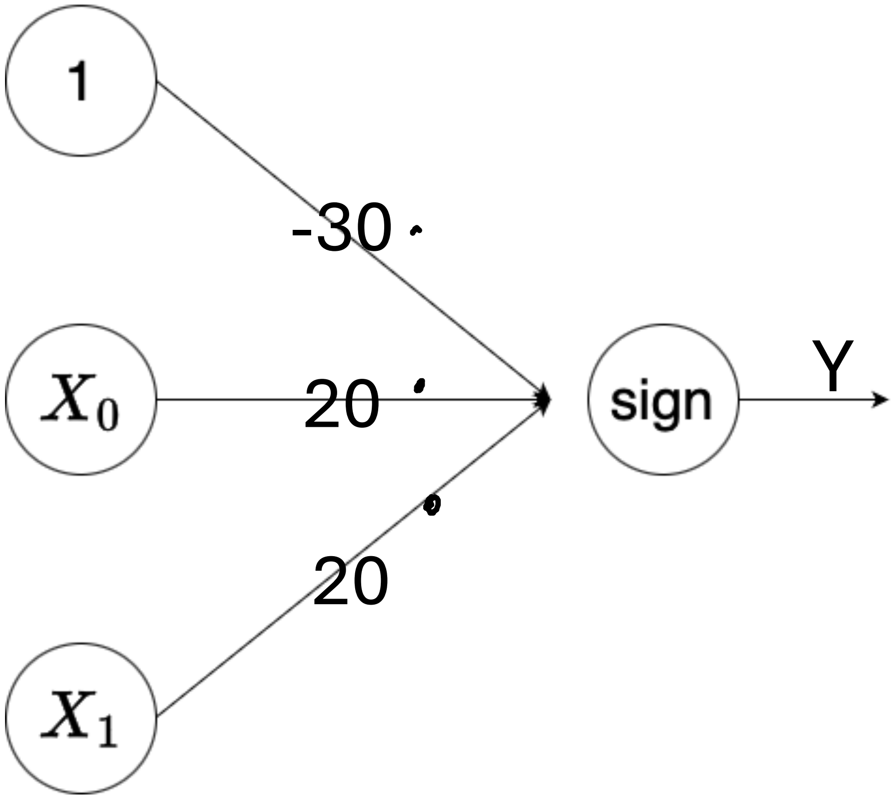
[Perceptron: A Founding Elements of Neural Net]

Perceptron is a building block for the neural net but had not had attention since the work of Rosenblatt in spite of the convergence proof. The problem is that Perceptron has no way to represent the knowledge (feature) required for solving certain problems. It started to work right as having multiple-layers.
Q: how could we design the feature map to be linearly separable ?

Perceptron Examples

[1] AND gate

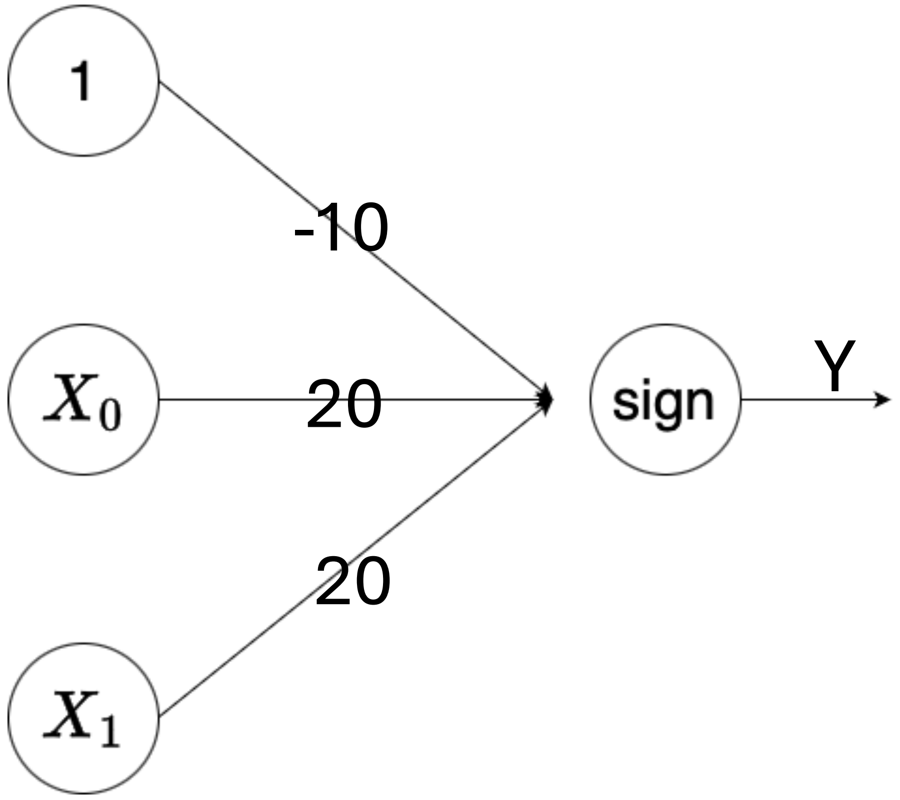
$$[-30,\ 20,\ 20]\begin{bmatrix}1\\ x_0\\ x_1\end{bmatrix} = -30 + \underbrace{20\overset{1}{x_0} + 20\overset{1}{x_1}}_{40} \begin{cases} > 0 \\ < 0 \end{cases}$$



X_0	X_1	$X_0 \wedge X_1$
0	0	0
0	1	0
1	0	0
1	1	1

Perceptron Examples

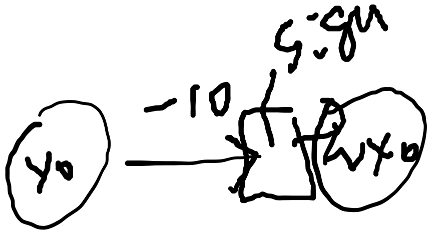
[2] OR gate



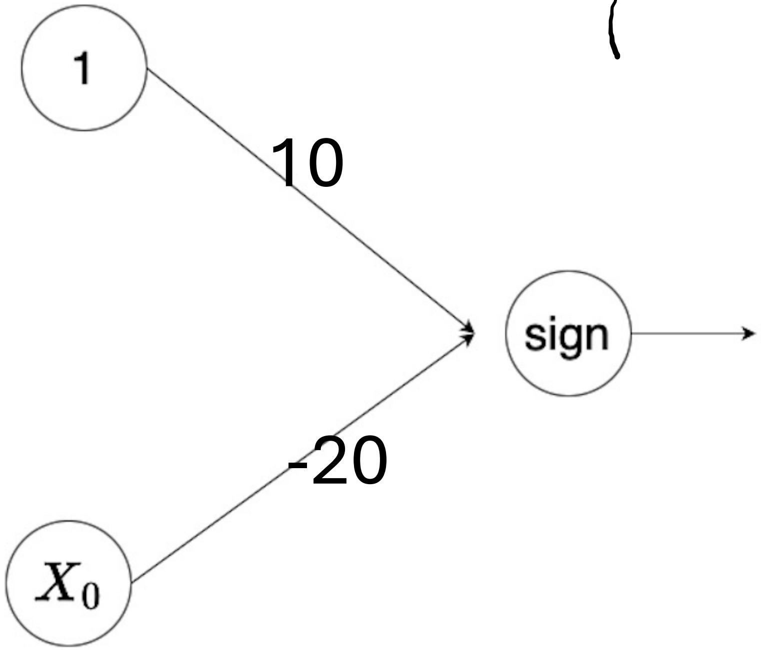
X_0	X_1	$X_0 \vee X_1$
0	0	0
0	1	1
1	0	1
1	1	1

Perceptron Examples

[3] Negation ~



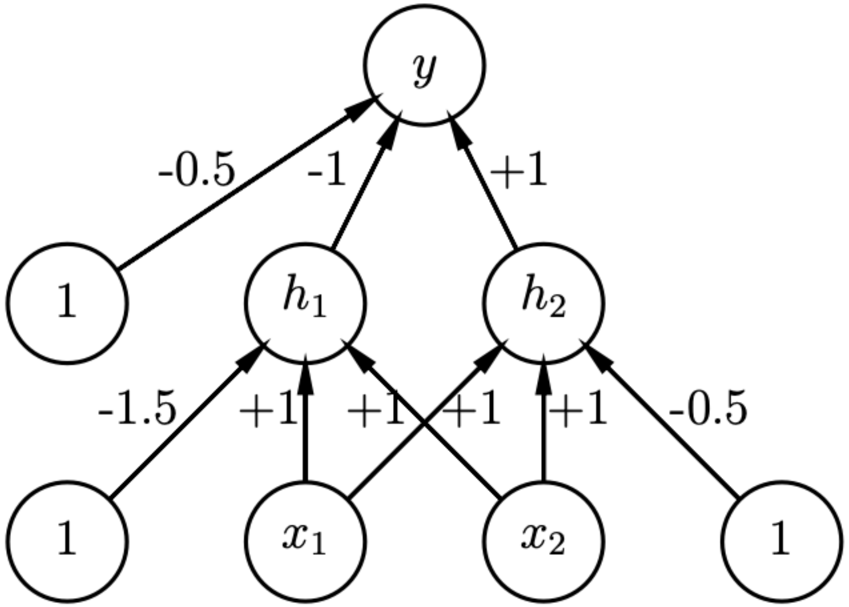
+ we can implement the negation gate buying using one node.



X_0	$\sim X_0$
0	1
1	0

Perceptron Examples

[4] XOR Problem [by MLP: **M**ulti-**L**ayer **P**erceptron]



X_0	X_1	$X_0 \oplus X_1$
0	0	0
0	1	1
1	0	1
1	1	0

$$\overline{X_0 \land X_1} \land (X_0 \lor X_1) = X_0 \oplus X_1$$

Probabilistic and Discriminative Classification

- Logistic Regression
- Multiple Class Logistic Regression

Discriminative Classification

: can we directly learn the posterior

- with a linear function of feature map? $\vec{W}\phi(\mathrm{x})$
- but without estimating likelihood / prior?

Representation of Posterior using Logistic Sigmoid

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

$$P[C_1|x] = \frac{1}{1 + \frac{P[x|C_0]P[C_0]}{P[x|C_1]P[C_1]}}$$

$$\ln P[x|C0] + \ln P[C0] - \ln P[x|C1] - \ln P[C0]$$

$$\ln P[\mathcal{H}_0] - \ln P[\mathcal{H}_1] + (\mu_0 - \mu_1)^t E\Lambda^{-1}E^t x - \frac{1}{2}\mu_0^t E\Lambda^{-1}E^t\mu_0 + \frac{1}{2}\mu_1^t E\Lambda^{-1}E^t\mu_1$$

$$P[C_1|x] = \frac{1}{1 + \exp\left(\ln \frac{P[x|C_0]P[C_0]}{P[x|C_1]P[C_1]}\right)}$$

$$P[C_1|x] = \frac{1}{1 + \exp\left(-\boxed{\ln \frac{P[x|C_1]P[C_1]}{P[x|C_0]P[C_0]}}\right)}$$

- $when\ \Sigma_0 = \Sigma_1$,
  it becomes a linear function of $\vec{x}$
- $\vec{W}\phi(\mathrm{x})$ or $\vec{W}x$

Complementary Posterior $P[C_0|x]$

is the symmetric function of Logistic Sigmoid Function.

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

$\sigma(x) = P(C_0|x)$

$= 1 - P(C_0|x)$

$\sigma(-x) = P(C_1|x)$

Learning one posterior function will be enough!

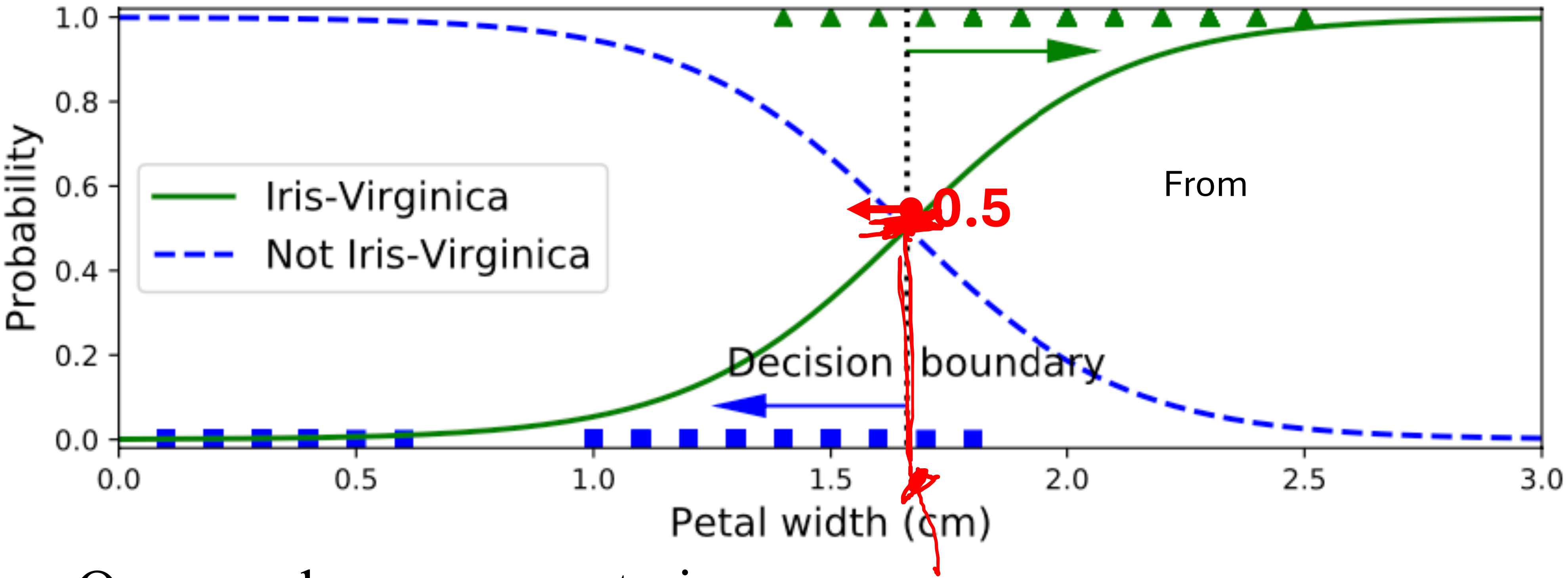
$$P[C_1|x] = \frac{1}{1 + \exp(-\ln \frac{P[x|C_1]P[C_1]}{P[x|C_0]P[C_0]})}$$

$$P[C_0|x] = \frac{1}{1 + \exp(\ln \frac{P[x|C_1]P[C_1]}{P[x|C_0]P[C_0]})}$$

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

Form Murphy Figure 2.11

Ex) P [Not Iris Virginica| x] *and* P [Iris Virginica| x]



From

Once we learn one posterior,
the other is automatically defined.
The $\sigma\ (\overrightarrow{W}x) = 0.5$ defines the decision boundary.

Logistic Sigmoid Function

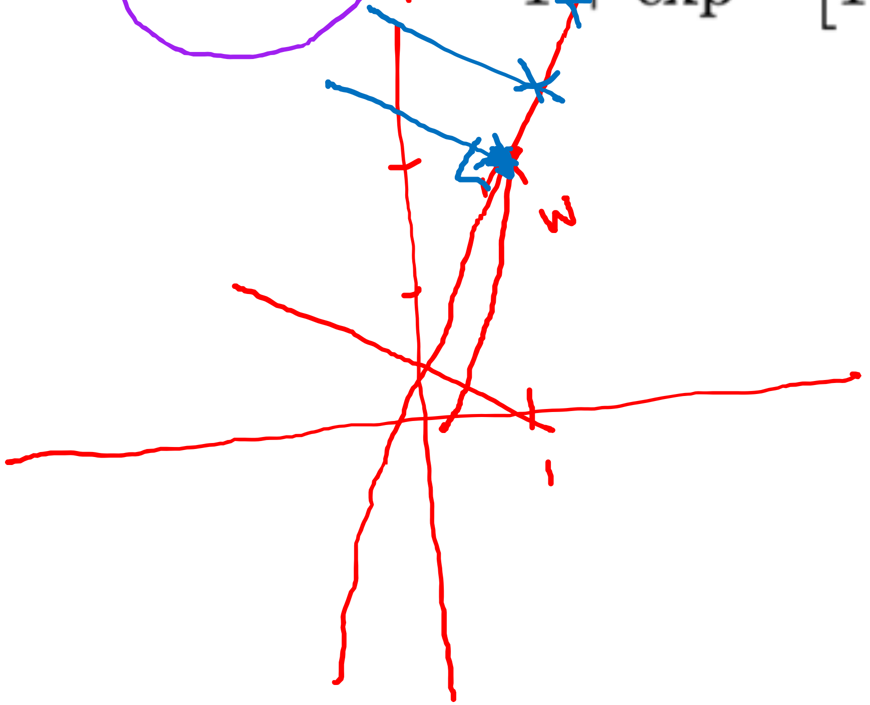
- decision boundary
- steepness

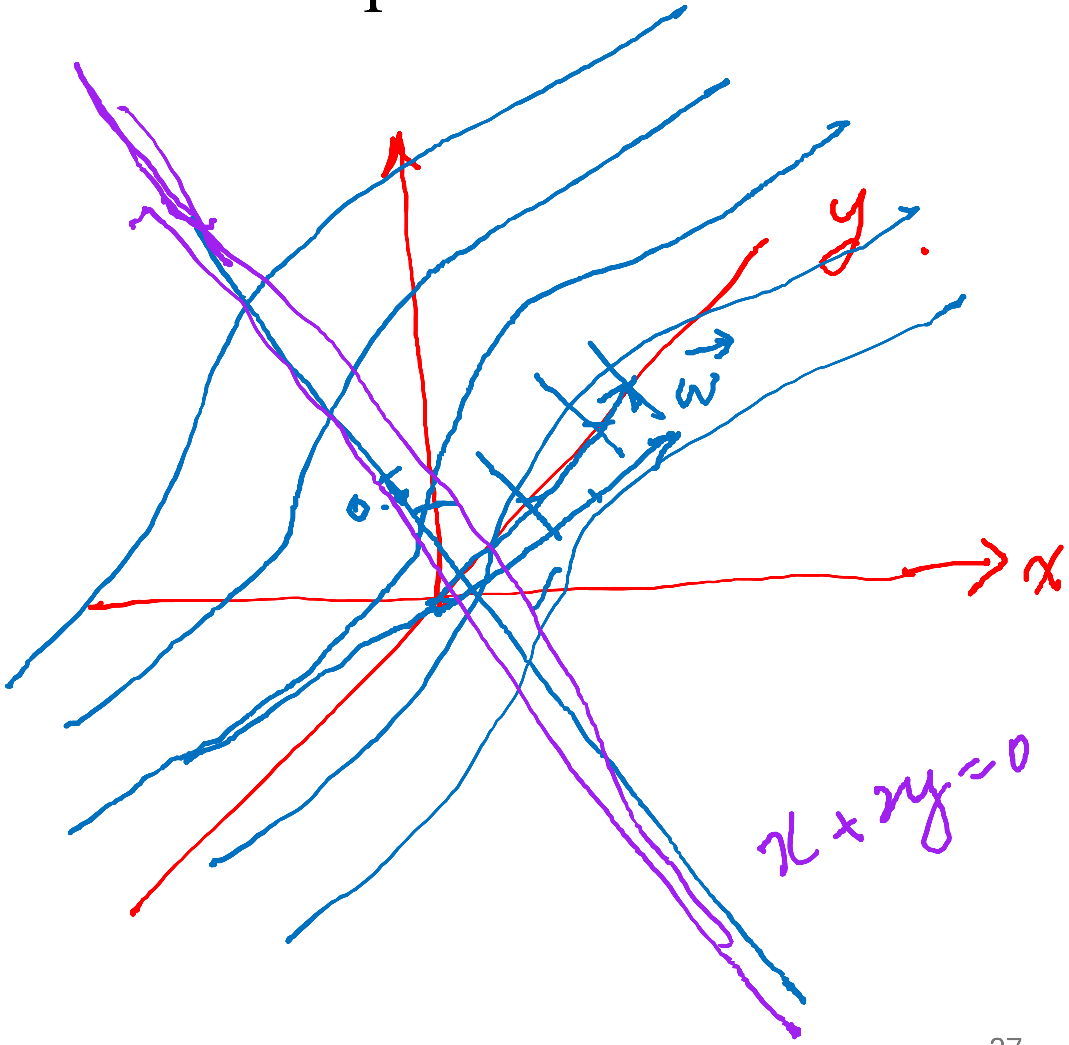
Example of Decision Boundary)

- Plot the logistic sigmoid function on the 2-D data space.

$[w_0, w_1]$

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

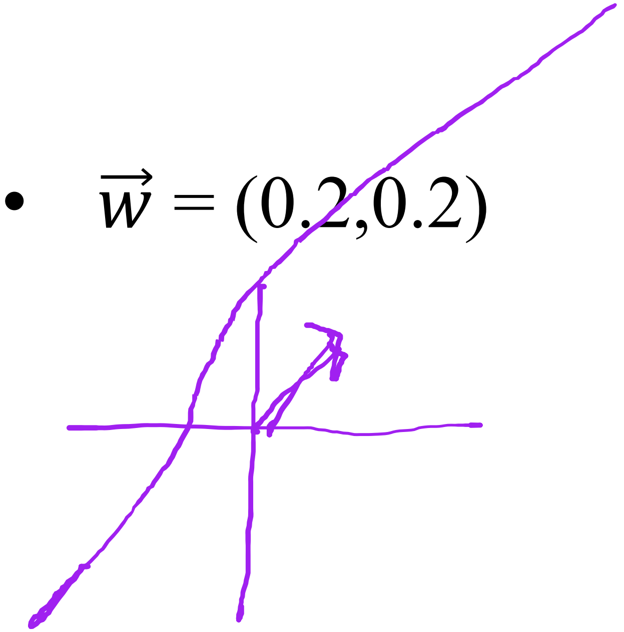




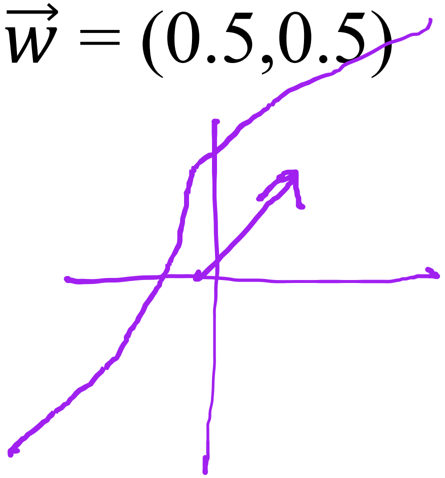
Example of Steepness)

- Plot the logistic sigmoid function on the 2-D data space.

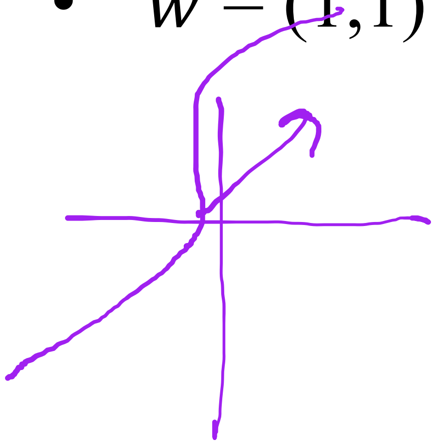
- $\vec{w} = (0.2, 0.2)$



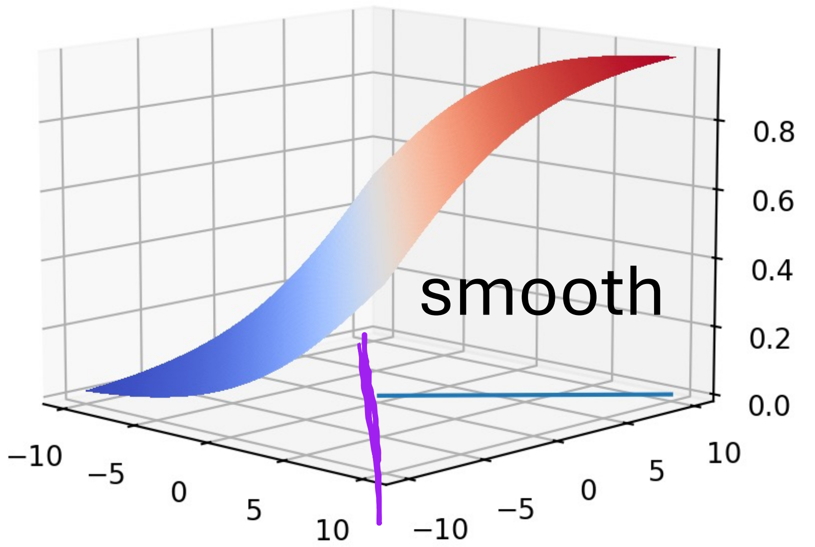
- $\vec{w} = (0.5, 0.5)$



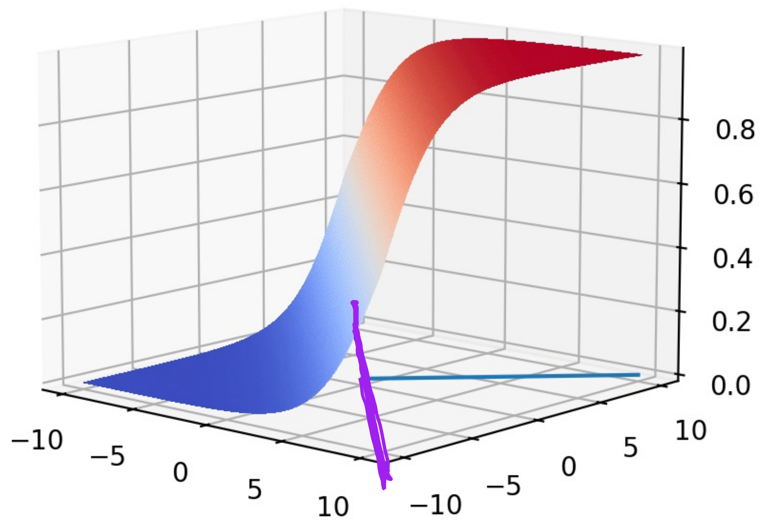
- $\vec{w} = (1, 1)$



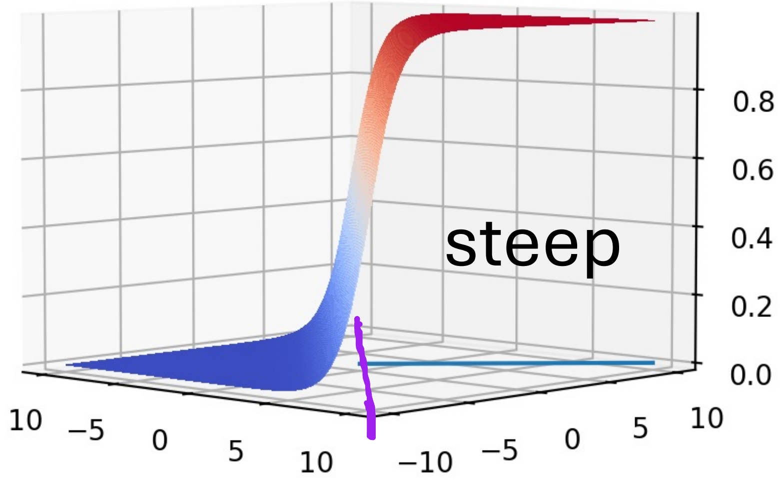
Example of Steepness)



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



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



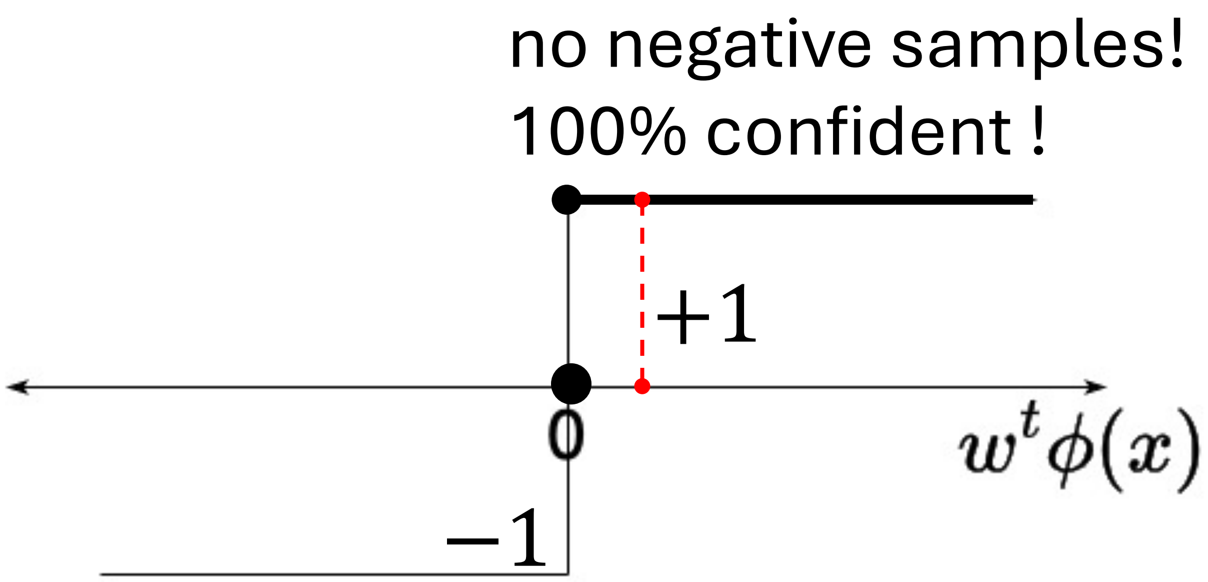
$\vec{w} = (1,1)$ · $||w|| = 1$

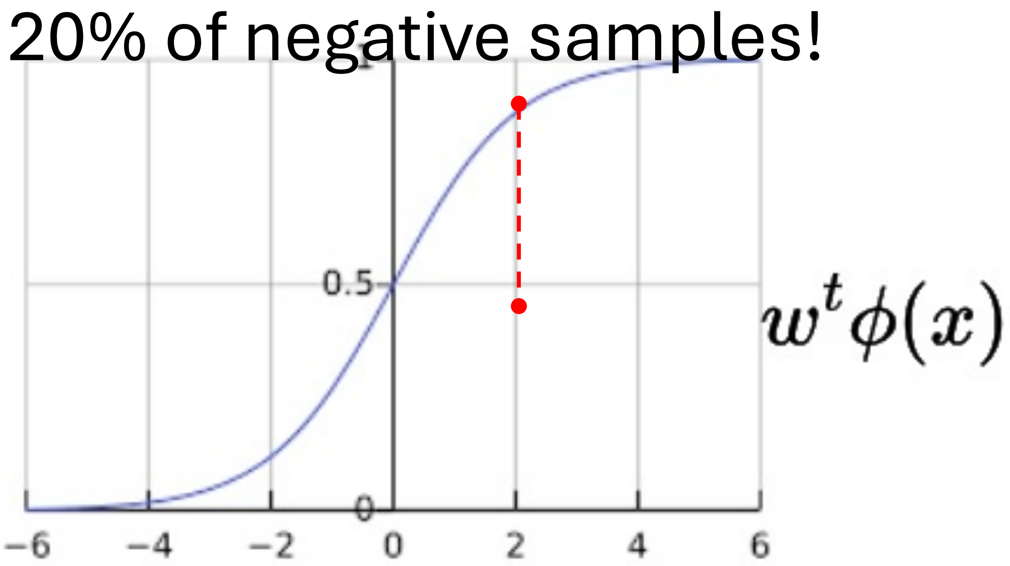
Sigmod Function $\sigma\ (\vec{w}^t x)$ for different $||\vec{w}||$

Decision boundaries will be the same $X_1 = -X_2 \leftrightarrow \boxed{X_1 + X_2 = 0}$

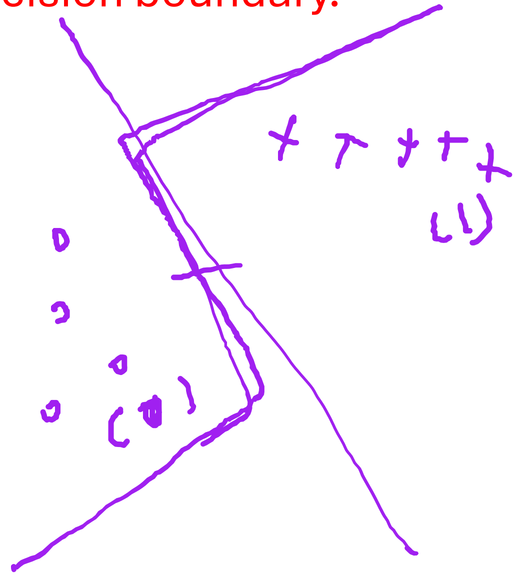
Decision boundaries are same but the different steepness?
What does it mean?

[1] Smooth vs. Steep Transition
can be regarded as the different level of confidence about classification.





+ when data is linearly separable, the logistic function has steep curve around decision boundary.



[2] Steep Sigmoid happens

when training data set is linearly separable!
by the nature of MLE.

- Objective Function

$$P(t|w) = \prod_{n=1} \sigma(w^t x_n)^{t_n} (1 - \sigma(w^t x_n))^{1-t_n}$$

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

- at large ||w||, the logistic value is1 or 0.

[3] Steep sigmoid needs careful examination!

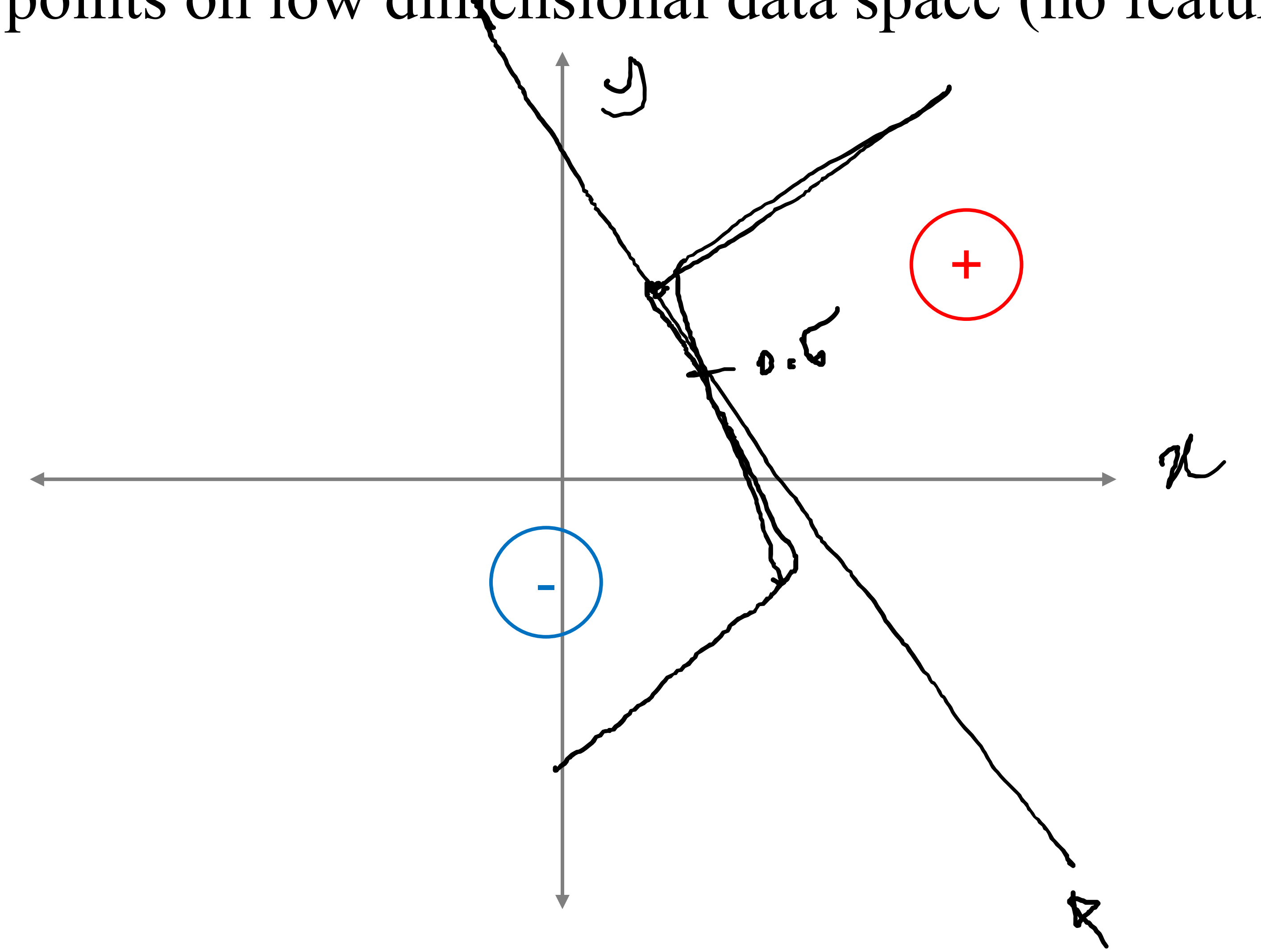
why?

+ Not always, but there is possibility of overfitting!

[3] A steep sigmoid function requires careful examination
as linear separability on the training data may suggest potential overfitting!

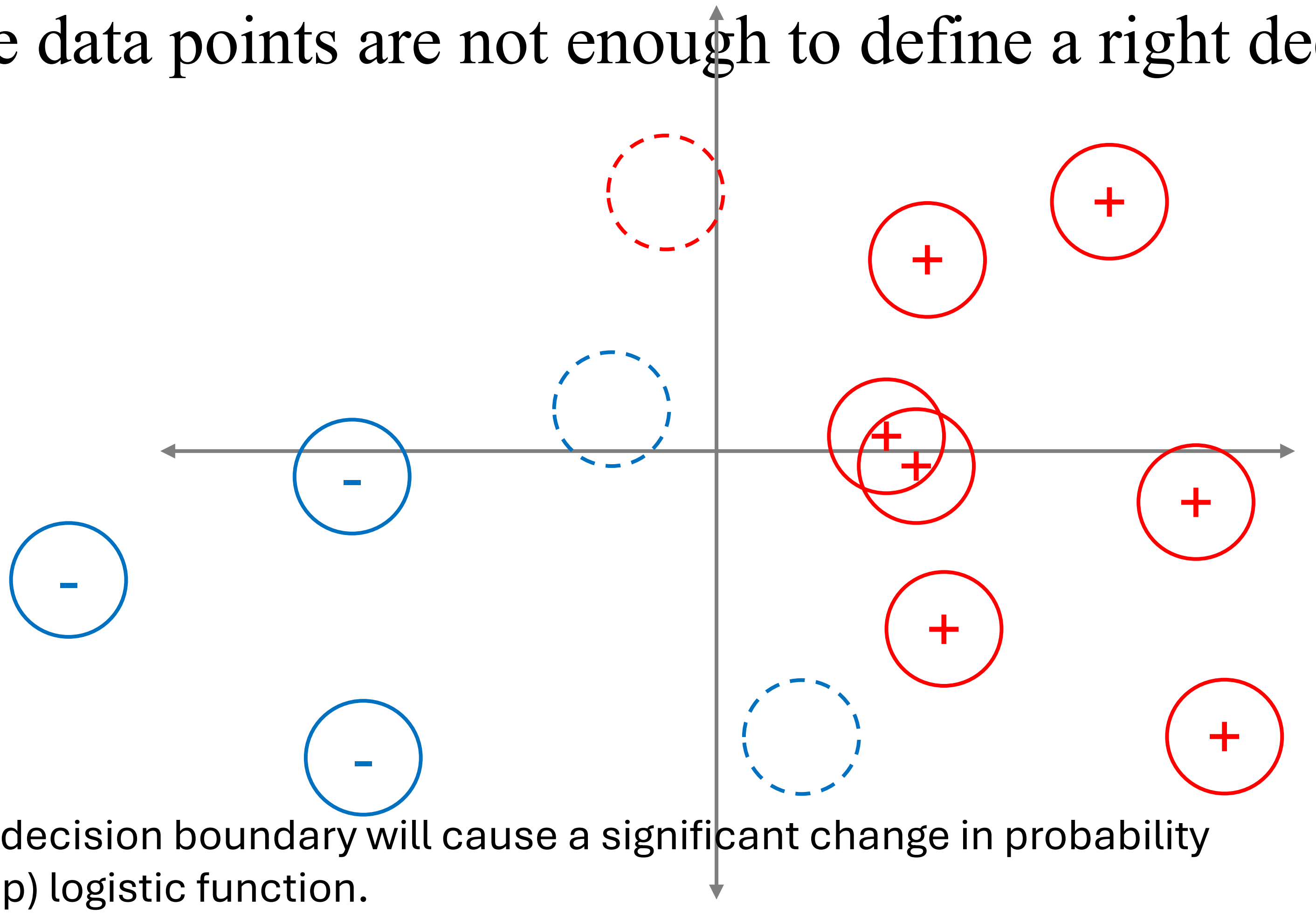
[3] A steep sigmoid function requires careful examination
as linear separability on the training data may indicate potential overfitting!

- case1] a few data points on low dimensional data space (no feature mapping).



[3] A steep sigmoid function requires careful examination as linear separability on the training data may indicate potential overfitting!

- case2] very high dimensional space
  but the data points are not enough to define a right decision boundary.



+ any slight difference in decision boundary will cause a significant change in probability for high confidence (steep) logistic function.

[4] A steep sigmoid function is appropriate!

if we have well-designed feature map that makes data space linearly separable

if we we have a sufficient number of data points.

<u>However, it is hard to achieve in practice.</u>

[4] A steep sigmoid function is appropriate!

if we have well-designed feature map that makes data space linearly separable

if we we have a sufficient number of data points.

However, it is hard to achieve in practice.

Regularization is needed to constrain ||W||.

MAP Estimation

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

MAP Estimation ($\vec{w} \sim \mathcal{N}(0, \sigma^2)I)$

prior term

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

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

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

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

ridge regression for various λs

MAP: Logistic Regression

Finding the optimal point $\overrightarrow{W}$

- Computing $\nabla J(\overrightarrow{W})$

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

$$= \sum_{n=1}^{N} \{-t_n(1-\sigma(w^t x_n)) - (1-t_n)(-\sigma(w^t x_n)\} x_n + 2\lambda\vec{w}$$

$$= \sum_{n=1}^{N} (\sigma(w^t x_n) - t_n) x_n + 2\lambda\vec{w}$$

- Gradient descent

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

Logistic regression has a global minimum for both MLE / MAP
any initial point will converge to the optimal solution $W*$
as we have a proper step size.

Multiclass Logistic Regression

Learning K posteriors

SoftMax Representation of Multiclass Posterior

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

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

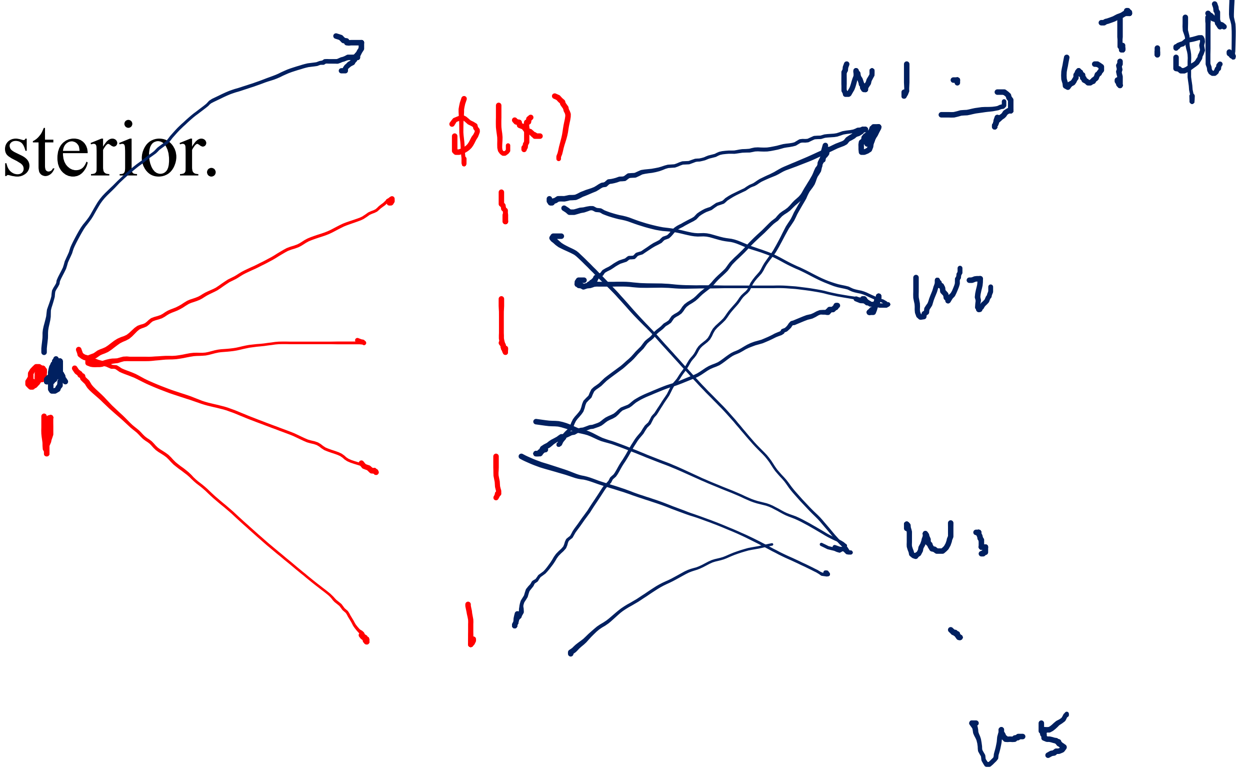
+ K= 3 discriminant functions

The posterior will be designed with softmax & $K = 3$ linear function of x

& $K = 3$ linear function of $\phi(x)$

Example) Compute the softmax posterior.

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

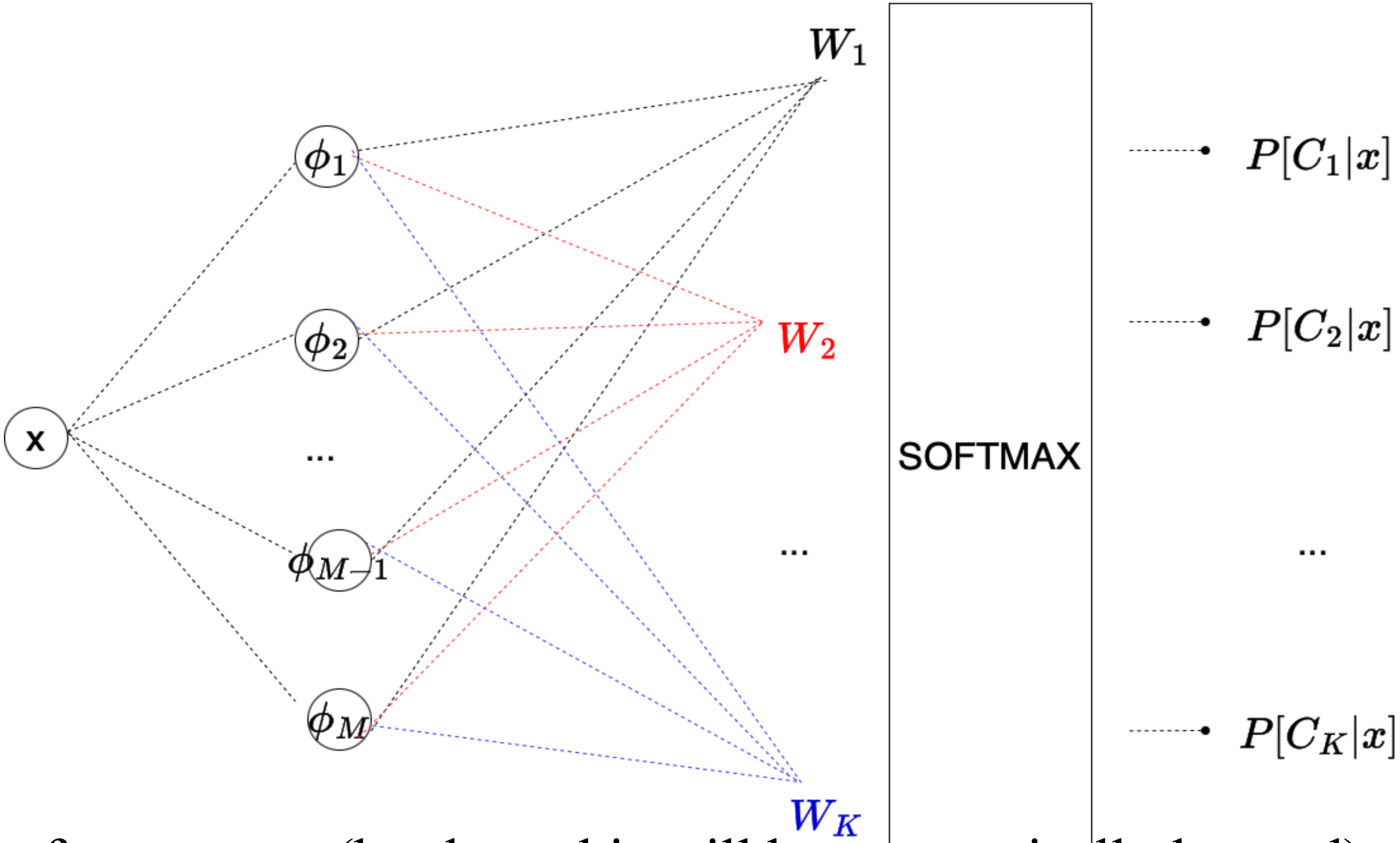


Example]

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

$$\begin{bmatrix} 1 & 1 & 1 \\ 2 & 2 & 2 \\ 3 & 3 & 3 \\ 4 & 4 & 4 \\ 5 & 5 & 5 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 6 \\ 9 \\ 12 \\ 15 \end{bmatrix}$$

The Classifier Structure for Multiclass Logistic



- Given a feature map (but later this will be automatically learned)
- Learning linear K discriminant functions

MultiClass Logistic Regression
learns K discriminant linear functions
and the functions are translated into posterior by softmax in the last layer.
(deep neural net classifier follows this structure!)
how can we learn the discriminant linear functions ? + using gradient descent!

- MLE for Multiclass Logistic Regression

  - Likelihood

$$P[T|W_1, W_2, ...W_K] = \prod_{n=1}^{N} \prod_{k=1}^{K} P[C_k|\phi(x)]^{t_{nk}}$$

  - Negative log

$$J(W_1, W_2, ...W_K) = -P[T|W_1, W_2, ...W_K] = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \ln P[C_k|\phi(x)]$$

  - Gradient

$$\nabla_{W_j} J(W_1, W_2, ...W_K) = -\sum_{n=1}^{N} \sum_{k=1}^{K} t_{nk} \nabla_{W_j} \ln P[C_k|\phi(x)]$$

Logistic Sigmoid Properties

- Symmetric

$$\boxed{\begin{aligned}\sigma(-x) &= 1 - \sigma(x) \\ \sigma(x) &= 1 - \sigma(-x)\end{aligned}}$$

- Derivative

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

SoftMax Derivative

- $\mathrm{P}[C_j|x]\ r.t\ \nabla W_j$

j =j

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

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

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

- $\mathrm{P}[C_j|x]\ r.t\ \nabla W_i$

i≠j

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

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

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

SoftMax Derivative

$$\nabla_{W_j} P[C_j|x] = P[C_j|x] \cdot (1 - P[C_j|x])\phi(x)$$
$$\nabla_{W_i} P[C_j|x] = P[C_j|x] \cdot (-P[C_i|x])\phi(x)$$

+ we can compute the P[Cj|x] at current W!

MLE: Multiclass Logistic Regression

- Gradient

$$\nabla_{W_j} J(W_1, W_2, ...W_K) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \nabla_{W_j} \ln P[C_k|\phi(x)]$$

$$= \sum_{n=1}^{N} t_{n1} P[C_j|\phi(x)] + ... + t_{nj}(P[C_j|\phi(x)] - 1) + ... + t_{nK} P[C_j|\phi(x)]$$

$$= \sum_{n=1}^{N} \{P[C_j|\phi(x)](\underbrace{t_{n1} + t_{n2} + ...t_{nK}}_{\text{Sum of one hot vector } =1}) - t_{nj}\}\phi(x)$$

$$= \sum_{n=1}^{N} \{P[C_j|\phi(x)] - t_{nj}\}\phi(x)$$

- Gradient descent

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

$$\nabla J(W) = \begin{bmatrix} \nabla_{W_1} J(W) \\ \nabla_{W_2} J(W) \\ \dots \\ \dots \\ \nabla_{W_{K-1}} J(W) \\ \nabla_{W_K} J(W) \end{bmatrix}$$

In the next class

- Perceptron Convergence Theorem and wrap-up logistic and perceptron
- Kernel Methods and Gaussian Process Regression
- Push the classification metrics (TA recitations/ deep CNN)