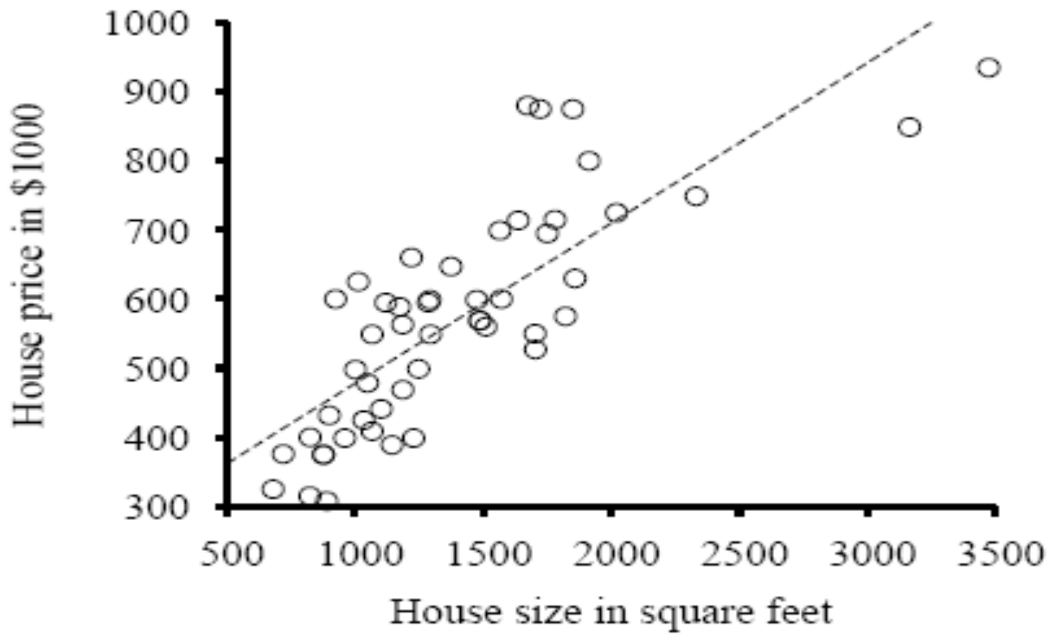
Linear Classification

Kietikul Jearanaitanakij

Department of Computer Engineering, KMITL

Regression and Classification with Linear Models

- Regression analysis is a statistical process for estimating the relationships between a dependent variable y and independent variable(s) x.
- Given data points of (x, y), linear regression consists of finding the best-fitting straight line (function) through the points. The best-fitting line is called a regression line.



Univariate linear regression

- Univariate linear function (a straight line) with input x and output y has the form $h_w(x) = w_1x+w_0$, where w_0 and w_1 are real-value coefficients (weights) to be learned.
- To find h_w that best fit input data x, we have to find the values of the weights $[w_0,w_1]$ that minimize the loss (e.g. training errors).

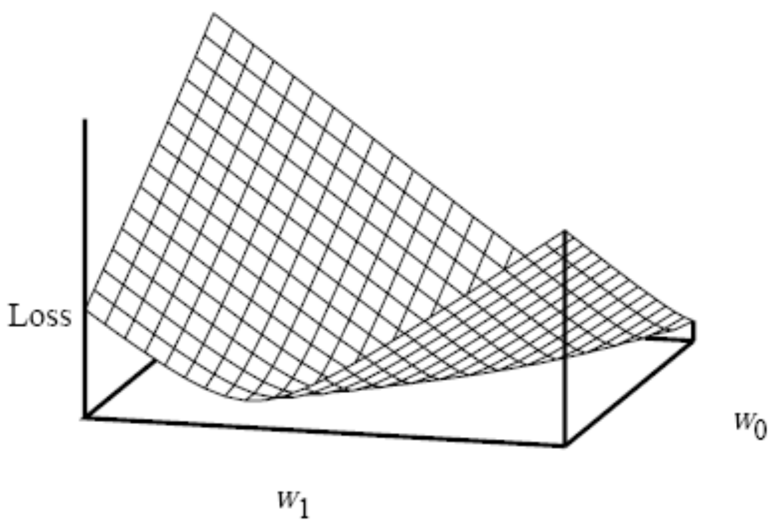
Target value

mean (true - pred)**2

Number of examples

Actual value

- Loss(h_w) = $\sum_{j=1}^{N}\left(y_j - h_w(x_j)\right)^2 = \sum_{j=1}^{N}\left(y_j - \underbrace{(w_1x_j + w_0)}_{h_w(x_j)}\right)^2$



Note that this loss function is convex, with a single global minimum.

- The sum $\sum_{j=1}^{N}\left(y_j - (w_1 x_j + w_0)\right)^2$ is minimized when its partial derivatives with respect to w_0 and w_1 are zero.

$$\underbrace{\frac{\partial}{\partial w_0}\sum_{j=1}^{N}\left(y_j - (w_1 x_j + w_0)\right)^2 = 0 \ , \frac{\partial}{\partial w_1}\sum_{j=1}^{N}\left(y_j - (w_1 x_j + w_0)\right)^2 = 0}$$

Optimization problem: solve equations for w0 and w1

เพราะเราปรับ w ได้

- For example in the above figure, the solution is w_1=0.232, w_0= 246, and the line with those weights is shown as a dashed line in the figure.

- In a general optimization problem, this optimization problem can be addressed by a **hill-climbing algorithm** that follows the gradient of the function to be optimized. We will use **gradient descent** to minimize the loss. Algo for derivative

w = any point in the parameter space

w = {w0, w1, w2, w3, …, wn}

loop until convergence

for each w_i in w do

$$w_i \leftarrow w_i - \propto \frac{\partial}{\partial w_i} Loss(\boldsymbol{w}) \qquad (1)$$

$Note$: ∝ is usually called the learning rate.

- For univariate regression, the loss function is a quadratic function, so the partial derivative will be a linear function.

$$
\begin{aligned}
\frac{\partial}{\partial w_i} Loss(\boldsymbol{w}) &= \frac{\partial}{\partial w_i} (\boxed{y - h_w(x)})^2 \\
&= 2(y - h_w(x)) \times \frac{\partial}{\partial w_i}(y - h_w(x)) \\
&= 2(y - h_w(x)) \times \frac{\partial}{\partial w_i}(y - \underbrace{(w_1 x + w_0)}_{h_w(x)})
\end{aligned}
$$

- Applying this to both w_0 and w_1 we get:

$2(y-h_w(x)) \times \frac{\partial}{\partial w_0}(\overset{k}{\cancel{y}} - (\overset{k}{\cancel{w_1}} x + \overset{-1}{w_0}))$

$2(y-h_w(x))(0-(\cancel{0}+1))$

$$
(1)\left\{
\begin{aligned}
\frac{\partial}{\partial w_0} Loss(\boldsymbol{w}) &= -2\big(y - h_w(x)\big) \\
\frac{\partial}{\partial w_1} Loss(\boldsymbol{w}) &= -2\big(y - h_w(x)\big) \times x
\end{aligned}
\right.
$$

$2(y-h_w(x)) \times \frac{\partial}{\partial w_1}(\overset{k}{\cancel{y}} - (w_1 x + \overset{k}{\cancel{w_0}}))$

$2(y-h_w(x)) \times (0-(x+0))$

online training $w_i \leftarrow w_i - \propto \frac{\partial}{\partial w_i} Loss(\boldsymbol{w})$

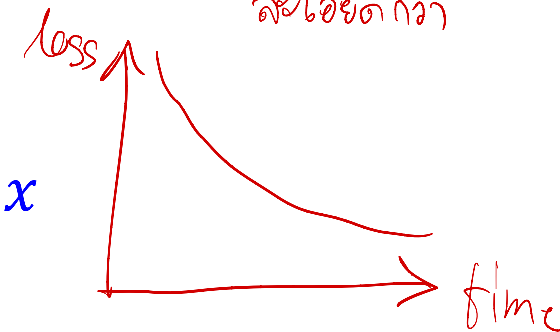
- Then, plugging this back into Equation (1), and folding the 2 into the unspecified learning rate ∝, we get the following learning rule for the weights:

ละเอียดกว่า

-2 ถูกยัดไปไว้ใน ∝

$$w_0 = w_0 + \propto \left(y - h_w(x)\right)$$
$$w_1 = w_1 + \propto \left(y - h_w(x)\right) \times x$$

Target value   Actual value

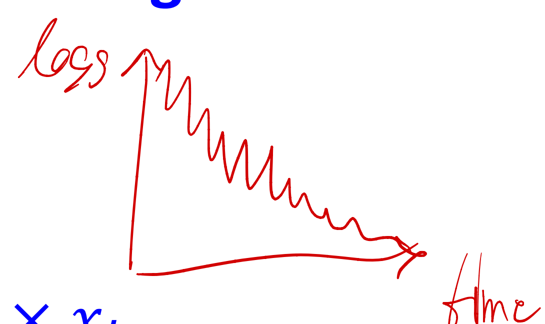


This is the weight updating for a single training example.

batch training

- For N training examples, we want to minimize the sum of the individual losses for every example (**batch gradient descent** learning rule).

ประหยัดเวลา



$$w_0 = w_0 + \propto \sum_j \left(y_j - h_w(x_j)\right)$$
$$w_1 = w_1 + \propto \underbrace{\sum_j \left(y_j - h_w(x_j)\right) \times x_j}_{\text{Sum derivatives of the loss values from N training examples.}}$$

Multivariate linear regression

- We can easily extend to multivariate linear regression problems, in which each example x_j is an n-element vector. Our hypothesis space is:

$$h_{sw}(x_j) = w_0 + w_1\mathbf{x}_{j,1} + \ldots + w_n\mathbf{x}_{j,n} = w_0 + \sum_i w_i \boldsymbol{x}_{j,i}$$

- The w_0 term, *the intercept*, stands out as different from the others.
- If we introduce input attribute ($x_{j,0}$) which is defined as always equal to 1, then h is simply the dot product of the weights and the input vector:

$$w_0 \overset{1}{x_{j,0}} + \sum_i w_i x_{j,i}$$

$$h_{sw}(x_j) = \mathbf{w}.\mathbf{x}_j = \mathbf{w}^{\mathrm{T}}.\mathbf{x}_j = \sum_i w_i x_{j,i}$$

$$\begin{bmatrix}3\\4\\1\end{bmatrix} \cdot [1\ 5\ 2] \rightarrow [3\ 4\ 1] \cdot [1\ 5\ 2] = \overset{3}{\cancel{3 \cdot 1}} + \overset{20}{\cancel{4 \cdot 5}} + \overset{2}{\cancel{1 \cdot 2}} = 25 \quad \text{(scalar)}$$

- The update weight w_i by substitute $h_{sw}(x_j)$ into the equation in page 7:

Target value

Actual value

$$w_i = w_i + \propto \underbrace{\sum_j x_{j,i} \times \left(y_j - h_{sw}(x_j)\right)}_{\text{Loss (multivariate)}}$$

- This process is equivalent in finding the best vector of weights, **w***, that minimizes squared-error loss over the examples:

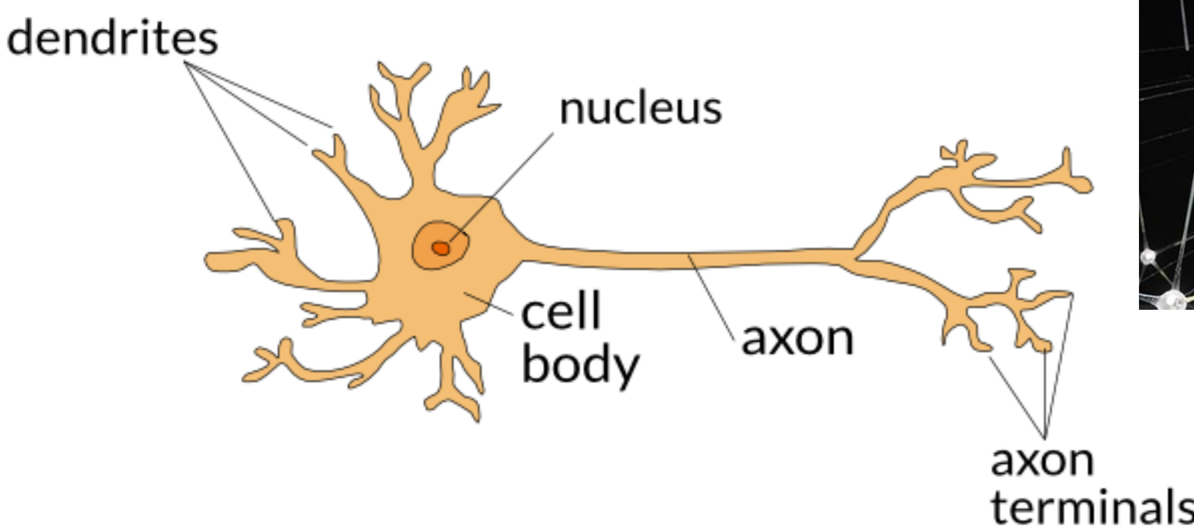
$$\boldsymbol{w}^* = \underset{w}{\operatorname{argmin}} \boxed{\sum_j (y_j - h_{sw}(x_j))^2}$$

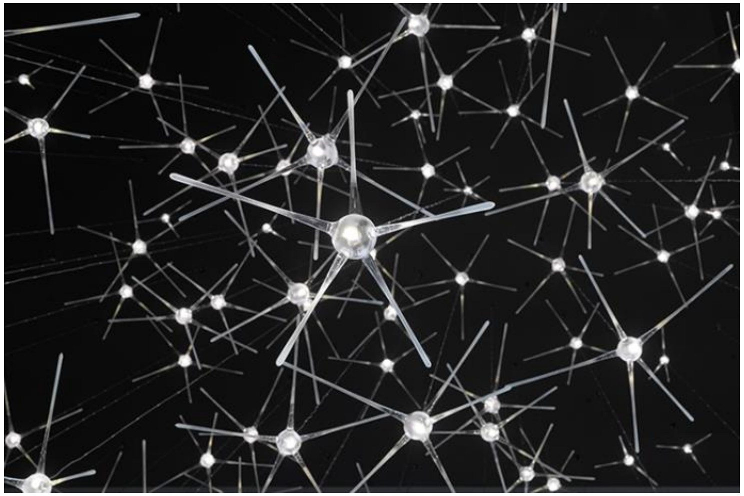
"แค่หา param ของ w ที่ทำให้ Loss ต่ำที่สุด"

(argument)

Linear Classification With Perceptron

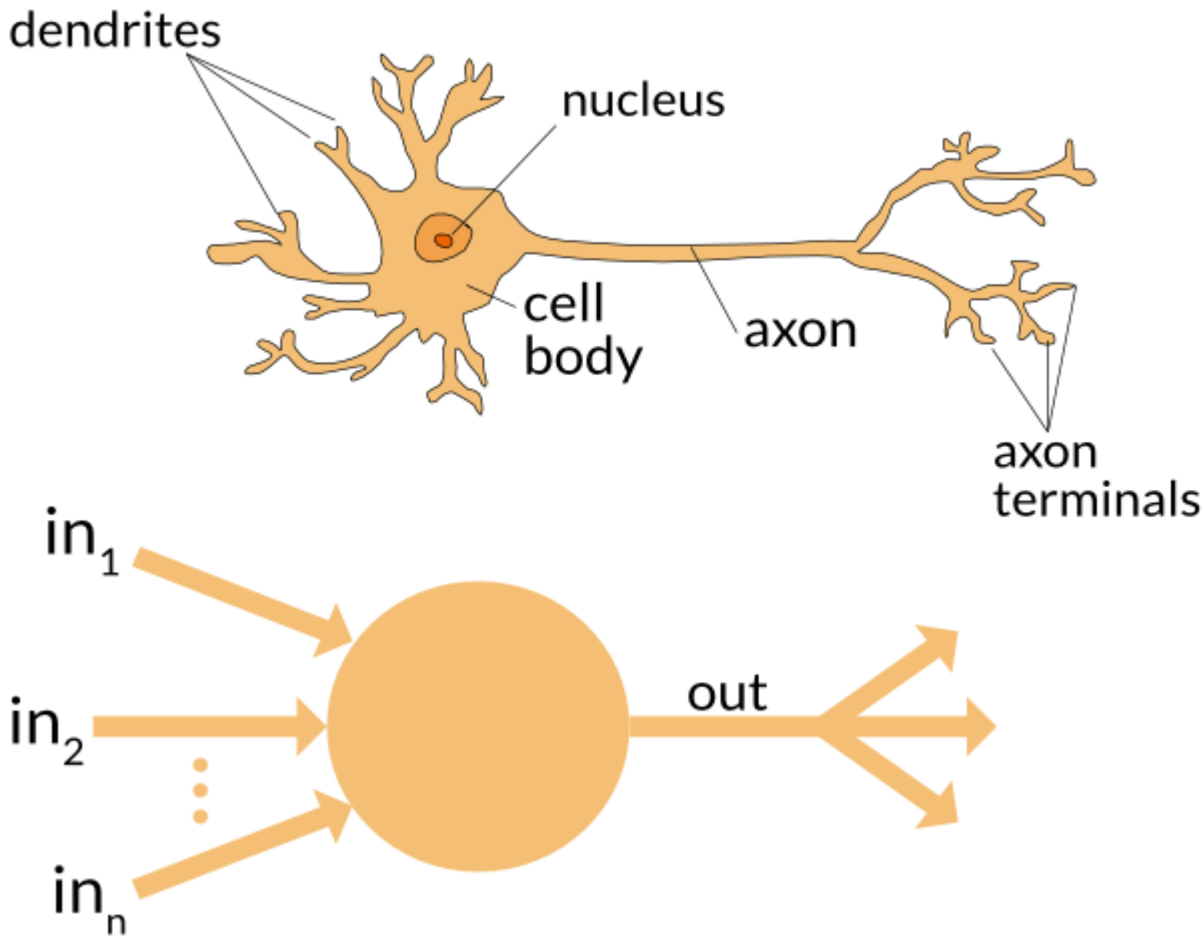
Biological neuron





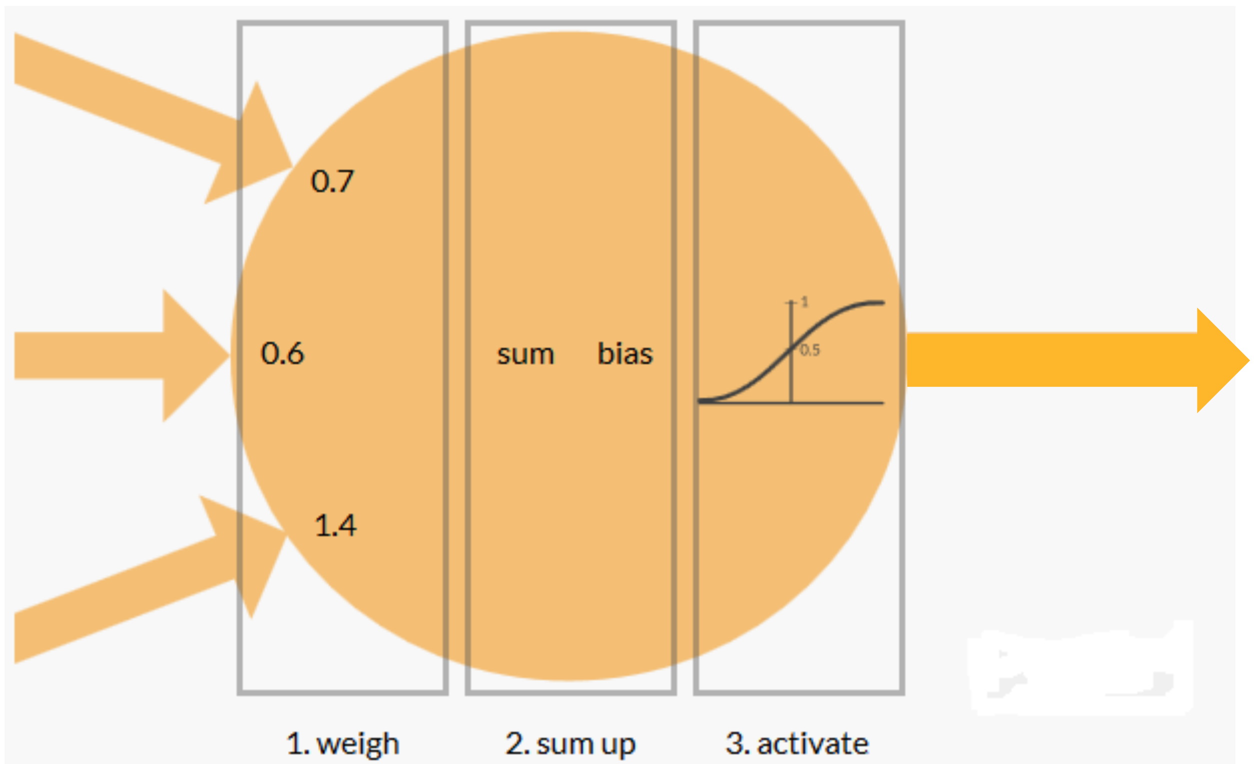
- Dendrites receive signals
- Axon sends signals out to other neurons

Artificial neuron



Inside artificial neuron

Input signals

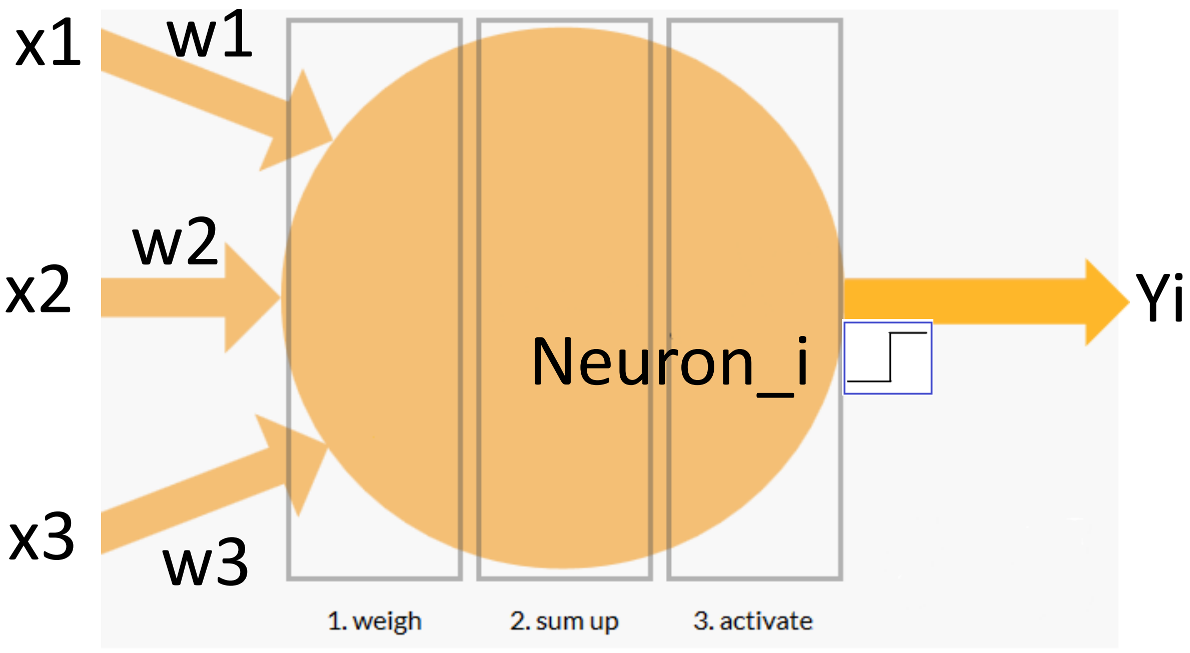


Output signal

We will call this artificial neuron as a **perceptron**.

Notations

Perceptron is the simplest form of a neural network. It consists of a single neuron with adjustable weights and a hard limit activation function.

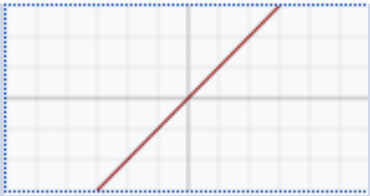
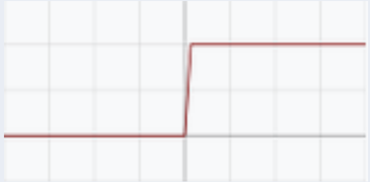
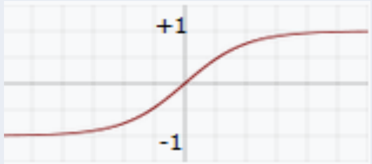


There are many kinds of activation function.

Frank Rosenblatt

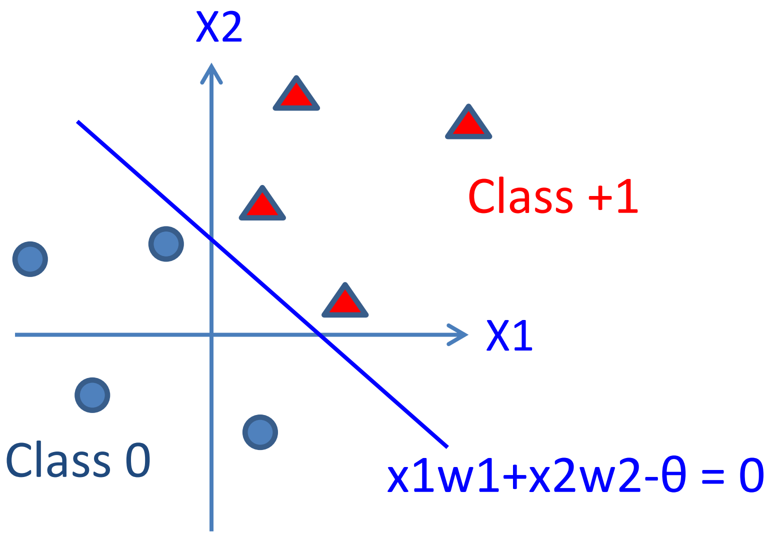
invented perceptron in 1957

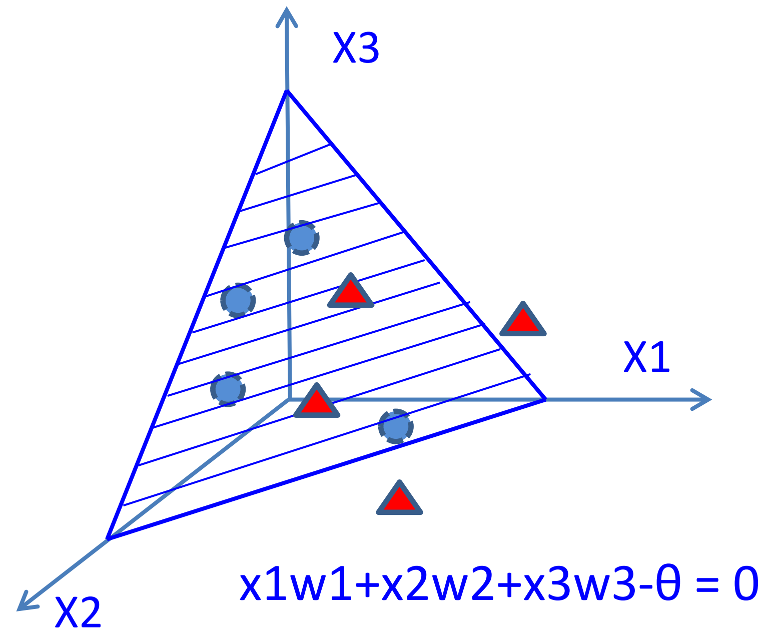
Activation functions

Name	Equation	Plot
Identity (Linear)	$f(x) = x$	
Binary step	$f(x) = \begin{cases} 0 & \text{for} \quad x < 0 \\ 1 & \text{for} \quad x \geq 0 \end{cases}$	
Sigmoid (Logistic)	$f(x) = \dfrac{1}{1 + e^{-x}}$	
TanH	$f(x) = \tanh(x) = \dfrac{2}{1 + e^{-2x}} - 1$	
Softmax	$f_i(\vec{x}) = \dfrac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}}$ for $i = 1, \ldots, J$	

Linear separability

Data in the n-dimensional space is **linearly separable** if two classes of data are divided by a hyperplane into two decision regions.





In general, the hyperplane is defined by the linearly separable function.

$$\sum_{i=1}^{n} x_i w_i - \theta = 0$$

(The threshold θ can be used to shift the decision boundary.)

- The perceptron learns its classification task by making small adjustments in the weights to reduce the difference between the actual and desired (target) outputs of the perceptron.

$$e(p) = \underbrace{Y_d(p)}_{\text{Target value}} - \underbrace{Y(p)}_{\text{Actual value}}$$

ยุคบุกเบิก

Where

p is the training pattern (example)

$Y_d(p)$ is the desired (target) output of pattern p

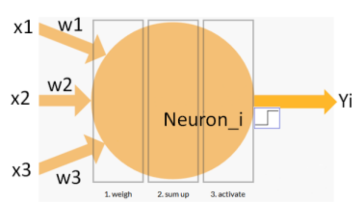
$Y(p)$ is the actual output

$e(p)$ is the difference (error) between $Y_d(p)$ and $Y(p)$

- It uses e(p) to update weights of the next iteration,

$$w_i(p+1) = w_i(p) - \alpha . x_i(p) . e(p)$$

where α is the learning rate (0~1)



Perceptron learning algorithm

Step 1: Initialization

- Set initial weights w1,w2,…,wn and thresholds (θ) to small random numbers in the range [-0.5,+0.5]
- p = 1   # the first pattern

Step 2: Activation

$$Y(p) = step\left(\sum_{i=1}^{n} x_i(p).w_i(p) - \theta\right)$$

where n is the number of perceptron inputs.

Step 3: Weight training by using gradient descent

−2k

$$w_i(p+1) = w_i(p) + \Delta w_i(p),$$

เปรียบเทียบกับ

$$\boxed{w_1 = w_1 + \propto \left(y - h_w(x)\right) \times x}$$

Target value   Actual value

$$\Delta w_i(p) = -\alpha . x_i(p) . e(p)$$

where $e(p) = Yd(p) - Y(p)$

Step 4: Iteration

Increase p by one, go back to step 2 until each pattern is trained.

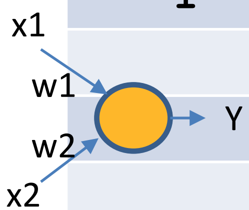
Step 5: If the perceptron doesn't converge, p = 1 and repeat steps 2 – 4.

Example: Train a perceptron on AND

ΔW_i

$$w_i(p+1) = w_i(p) - \alpha . x_i(p) . e(p)$$

$\overset{0}{x_1} * \overset{0.3}{w_1} + \overset{0}{x_2} * \overset{-0.1}{w_2} < \overset{0.3}{\theta}$ θ= 0.3, α = 0.1



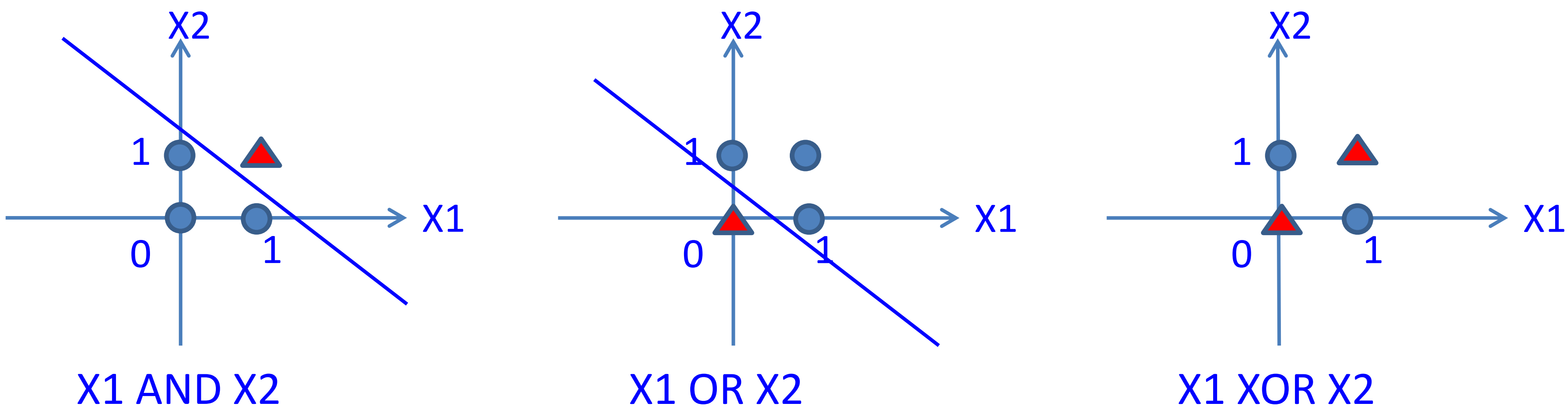
Epoch	Inputs x1	Inputs x2	Yd (desired)	Weights w1	Weights w2	Y (actual)	Error	Weights w1	Weights w2
1	0	0	0	0.3	-0.1	0	0	0.3	-0.1
	0	1	0	0.3	-0.1	0	0	0.3	-0.1
	1	0	0	0.3	-0.1	1	-1	0.4	-0.1
	1	1	1	0.4	-0.1	1	0	0.4	-0.1
2	0	0	0	0.4	-0.1	0	0	0.4	-0.1
	0	1	0	0.4	-0.1	0	0	0.4	-0.1
	1	0	0	0.4	-0.1	1	-1	0.3	-0.1
	1	1	1	0.3	-0.1	0	1	0.2	-0.2
.	.		.	.		.	.	.	
.	.		.	.		.	.	.	
.	.		.	.		.	.	.	
5	0	0	0	0.15	0.15	0	0	0.15	0.15
	0	1	0	0.15	0.15	0	0	0.15	0.15
	1	0	0	0.15	0.15	0	0	0.15	0.15
	1	1	1	0.15	0.15	1	0	0.15	0.15

$\Delta W_1 = -\overset{0.1}{\alpha} \cdot \overset{1}{X_1}(p) \cdot \overset{-1}{e}(p) = 0.1$

$\Delta W_2 = -0.1 * 0 * (-1) = 0$

Problem of perceptron

- Perceptron can learn only simple linear separable problems, e.g., AND, OR. It failed on XOR problem.

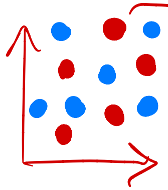


- A single perceptron can classify only linear separable problems, regardless of whether we use a hard-limit or soft-limit activation functions.
- Moreover, increasing the number of perceptrons in the same layer doesn't help.

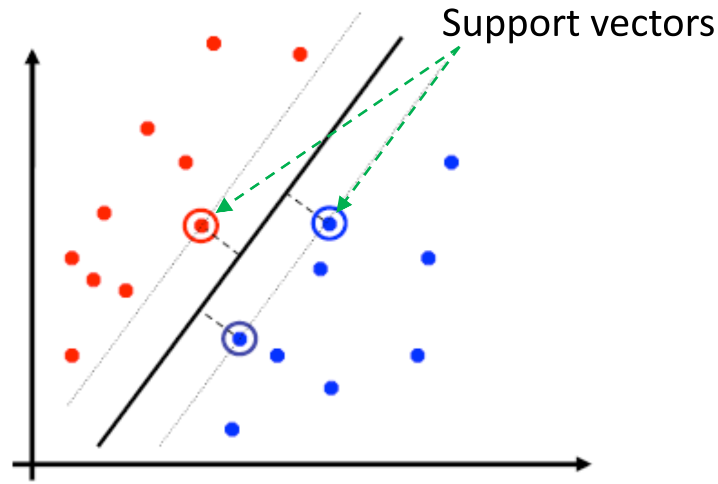
Linear Classification by Support Vector Machine

- SVMs (Vapnik, 1990's) choose the linear separator with the **largest margin**

Kernel function



Robust to outliers!



V. Vapnik

- Good according to intuition, theory, practice

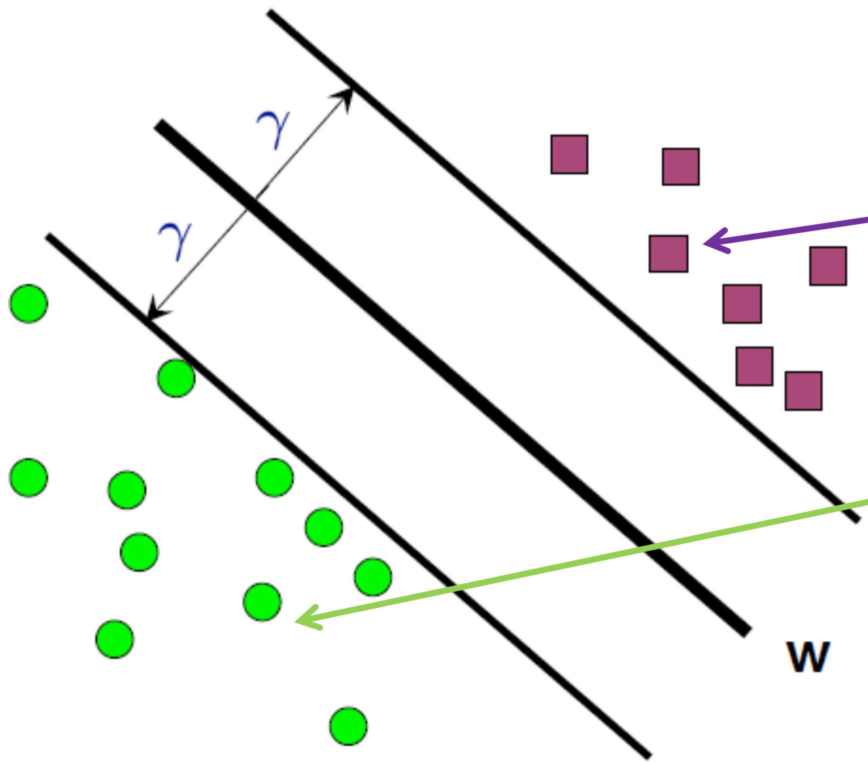
- SVM became famous when, using images as input, it gave accuracy comparable to neural-network

Pages 22-24 are taken from Introduction To Machine Learning @MIT

Linear Classification by Support Vector Machine

$\exists \mathbf{w}$ such that $\forall t$ $\quad y_t(\mathbf{w} \cdot \mathbf{x}_t) \geq \gamma > 0$

Called the *margin*



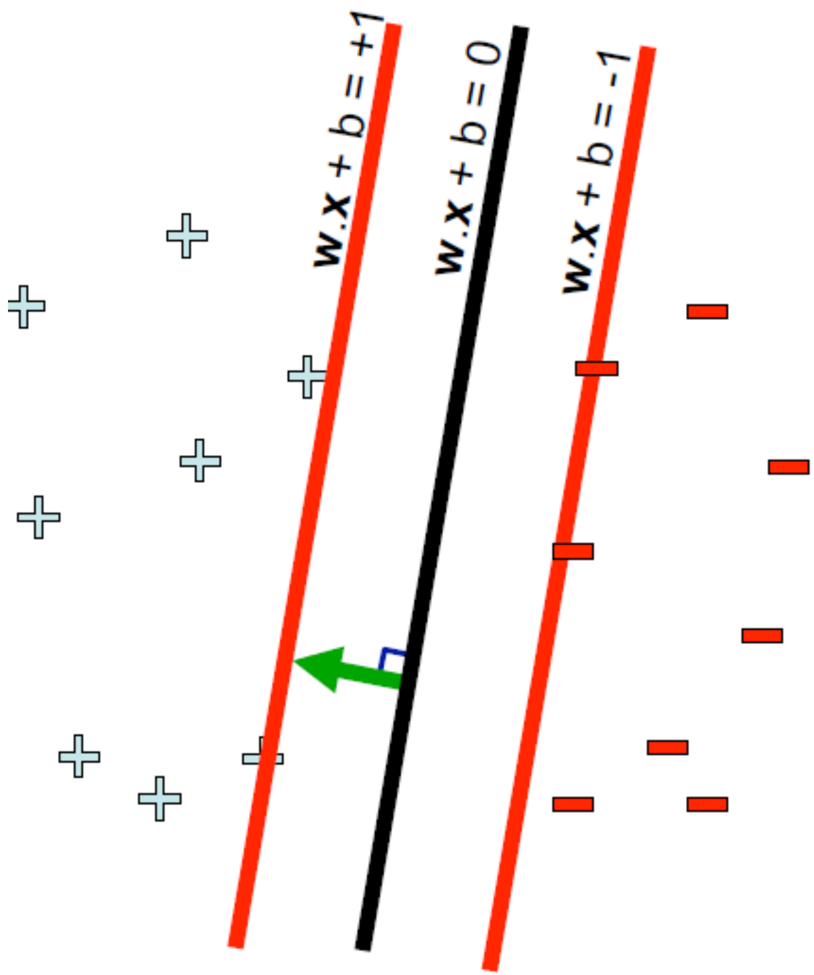
Equivalently, for $y_t = +1$,

$$w \cdot x_t \geq \gamma$$

and for $y_t = -1$,

$$w \cdot x_t \leq -\gamma$$

Linear Classification by Support Vector Machine



Suppose that the value of margin is 1.

We are asked to find a set of weights (w) such that, for all pattern t,

for y_t = +1, $w \cdot x_t + b \geq 1$

and for y_t = -1, $w \cdot x_t + b \leq -1$

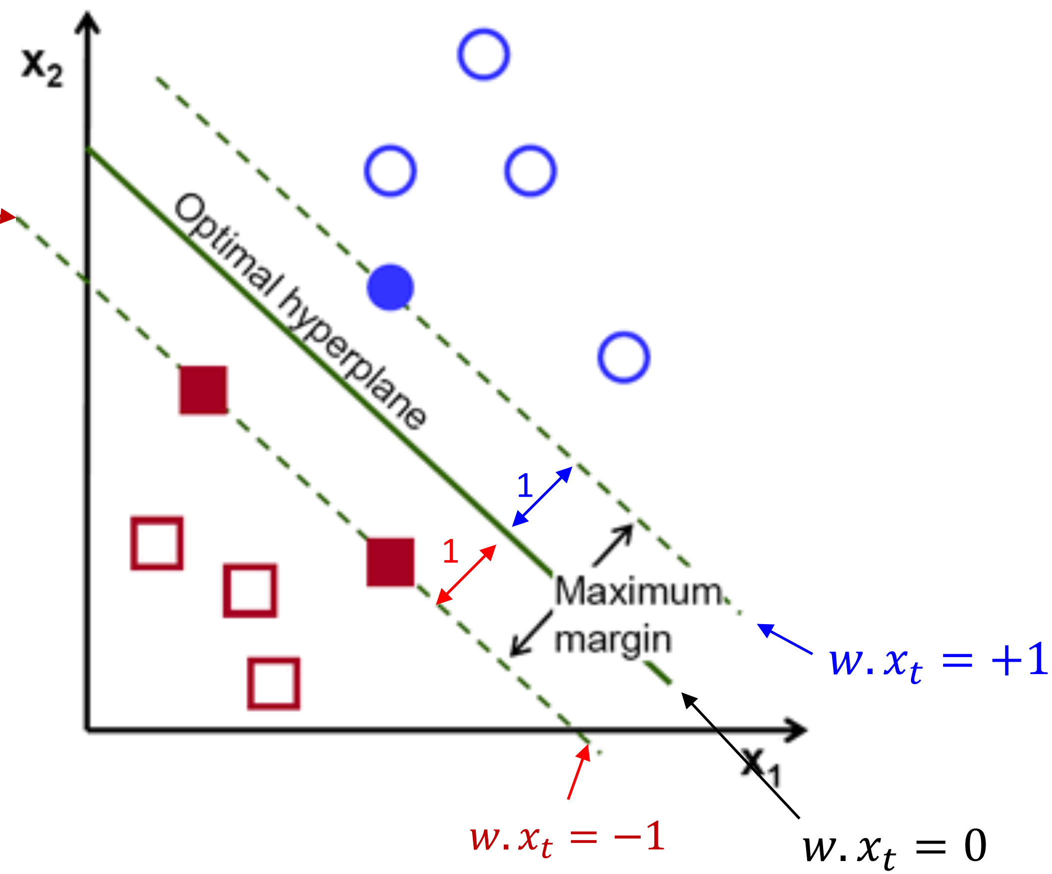
That is, we want to satisfy all of the **linear** constraints

$$y_t\,(w \cdot x_t + b) \geq 1 \quad \forall t$$

SVM Loss

- Loss of SVM

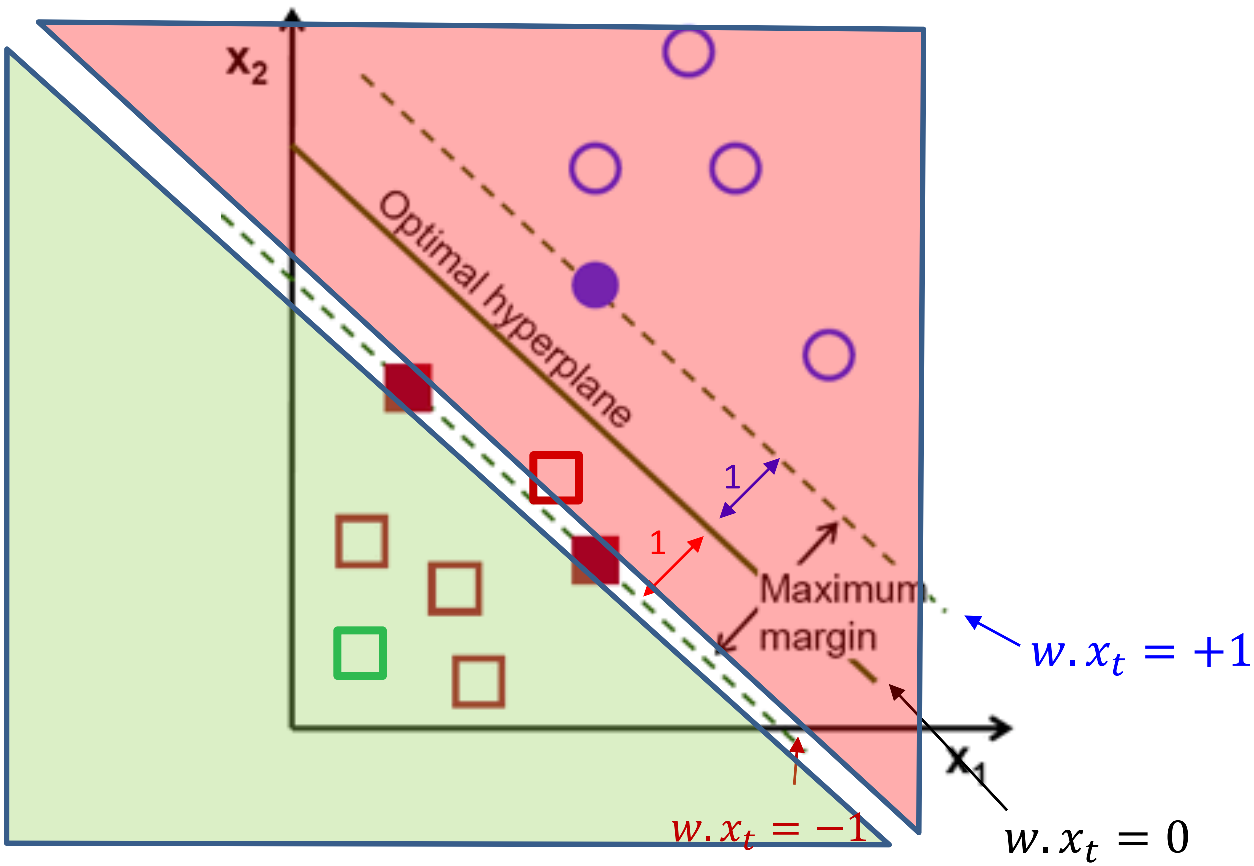
Any rectangle instance that crosses the maximum margin (dash line) will be penalized.



SVM Loss

Penalty = 0

Penalty > 0



SVM Loss

Let $s = f(x_i, w)$; score of input x_i

s_{yi} is the score of the target class

Score of other class

the SVM loss has the form:

Score of the target class

$$L_i = \sum_{j \neq y_i} \begin{cases} 0 & \text{if } s_{y_i} \geq s_j + 1 \\ s_j - s_{y_i} + 1 & \text{otherwise} \end{cases}$$

$$= \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$$

cat	$\overset{s_{yi}}{3.2}$
car	$\overset{s_j}{5.1}$
frog	$\overset{s_j}{-1.7}$

$$\boxed{L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)}$$

= max(0, 5.1 - 3.2 + 1)
+max(0, -1.7 - 3.2 + 1)
= max(0, 2.9) + max(0, -3.9)
= 2.9 + 0
= 2.9

- Next class
  - Multilayer neural networks