# Lecture 12

# Machine Learning

k nearest neighbors

neural networks

## 12.1 Machine Learning

Machine Learning is a field of study focusing on how to make computer system learn from a set of examples. To be more precise, Machine Learning focuses on *automatically constructing models from the collected dataset without requiring explicit programming.*

Several types of machine learning techniques have been developed until now, e.g. supervised learning, unsupervised learning, reinforcement learning, etc. In this course, we focus only on *supervised machine learning.*

(learn from training examples)

### 12.1.1 Supervised Machine Learning

Given a set of input-output pairs

an input vector

$$\mathcal{D} = \{(\mathbf{x}_i, t_i)\}_{i=1}^{N}$$

a target output

where,

$\mathcal{D}$ is a training set,

$N$ is the number of examples,

$\mathbf{x}_i \in \mathcal{X}$ is an input vector, and

$t_i \in \mathcal{T}$ is a target output,

$$\vec{x} \longrightarrow \boxed{\hat{f}} \longrightarrow t$$

this function is constructed automatically by an ML algorithm.

find a function $\boxed{\hat{f}} : \mathcal{X} \to \mathcal{T}$ that predicts the value of $t$ from an unknown $\mathbf{x}$.

Each input vector $\mathbf{x}_i$ is usually represented as a $D$-dimensional feature vector, i.e.

dimension of input vectors

$$\mathbf{x}_i = [x_{i1}, x_{i2}, \ldots, x_{iD}]^T \in \mathbb{R}^D$$

It can be represented as a point in the $D$-dimensional space.

It is assumed that each pair $(\mathbf{x}_i, t_i)$ is drawn i.i.d. (independently and identically distributed) from an unknown distribution on $\mathcal{X}$.

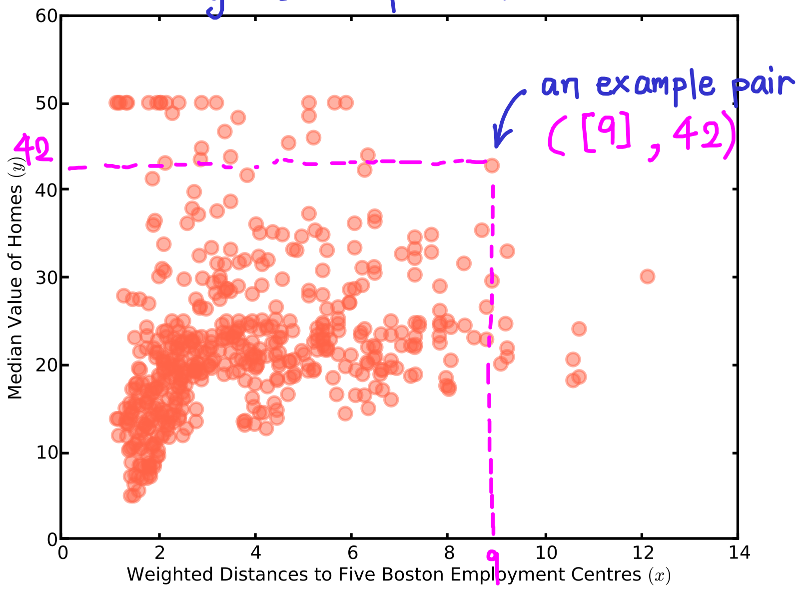
### 12.1.2 Classification and Regression

#possible outputs is finite — #possible outputs is infinite

When $\mathcal{T}$ is a finite set or $t_i$ is categorical, the task of the supervised learning is called *classification*. The task is called *regression* when $t_i$ is real-valued i.e. $\mathcal{T} = \mathbb{R}$ or $\mathcal{T} = \mathbb{R}^D$. In this course, we mainly focus on the classification task where the *binary classification* is the most frequently studied problem. It is the classification task when $\mathcal{T} = \{-1, +1\}$.

**Example 12.1** Predict the housing values in suburbs of Boston.[1]

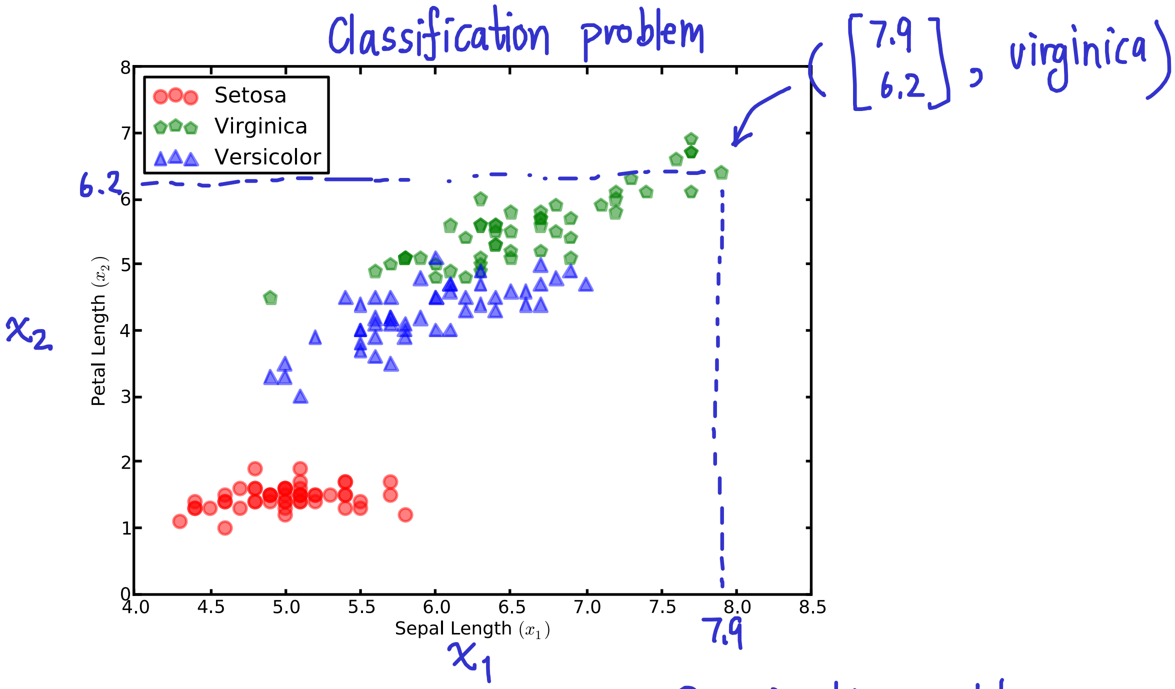
Regression problem




**Example 12.2** Predict the class of iris using its shape.[2]

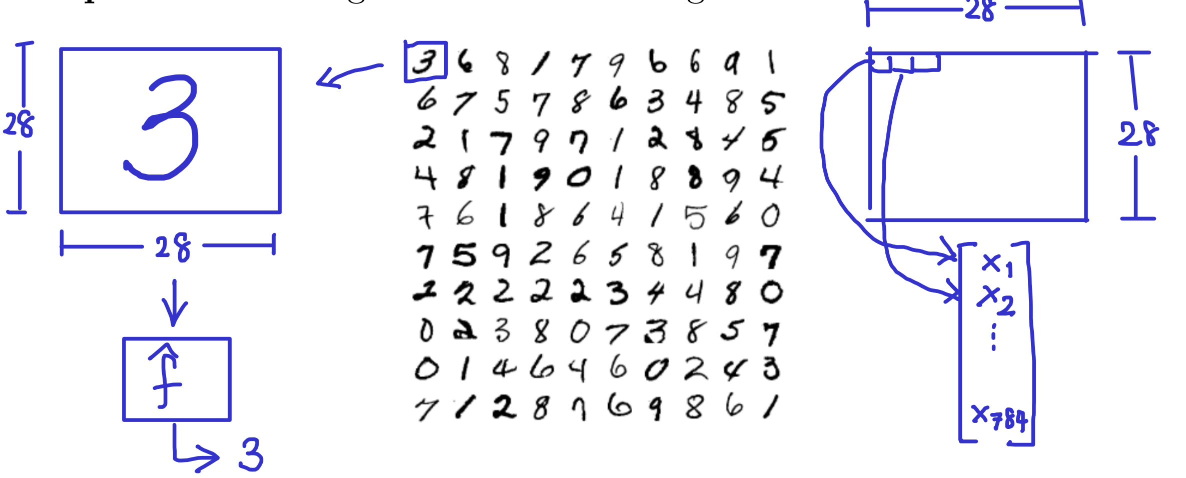
[1]Housing Data Set – `http://archive.ics.uci.edu/ml/datasets/Housing`

[2]Iris Data Set – `http://archive.ics.uci.edu/ml/datasets/Iris`




**Example 12.3** Recognize handwritten digits.[3]

Classification problem




Each handwritten digit is preprocessed and transformed into a $28 \times 28$ greyscale image. Therefore, each input vector composes of 784 features.

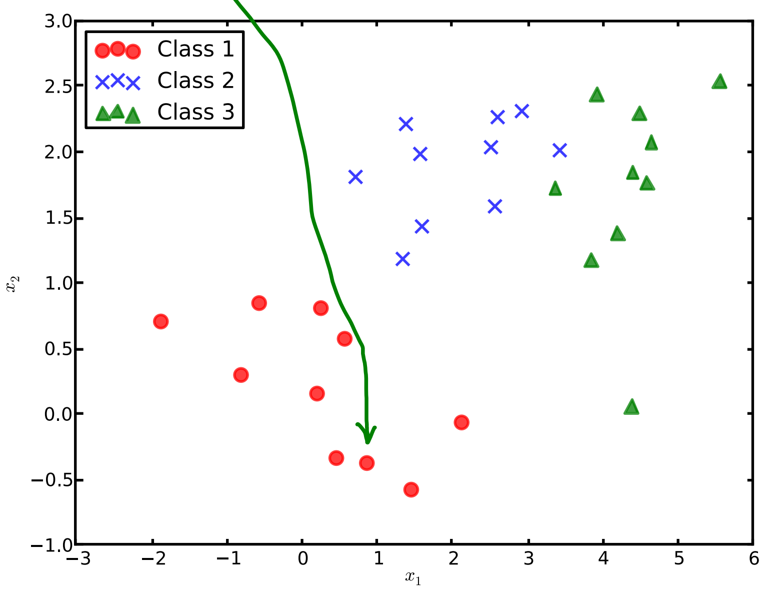
[3] The MNIST Database of Handwritten Digits (`http://yann.lecun.com/exdb/mnist/`); Y. LeCun, L. Bottou, Y. Bengio and P. Haffner: Gradient-Based Learning Applied to Document Recognition, Proceedings of the IEEE, 86(11):2278-2324, November 1998

**Example 12.4** Classify a set of artificially generated examples

| $x_1$ | $x_2$ | $t$ |
|---|---|---|
| 2.124 | −0.065 | 1 |
| 0.253 | 0.807 | 1 |
| 1.454 | −0.578 | 1 |
| 0.569 | 0.573 | 1 |
| 0.458 | −0.337 | 1 |
| −0.809 | 0.297 | 1 |
| 0.864 | −0.375 | 1 |
| 0.202 | 0.155 | 1 |
| −1.875 | 0.705 | 1 |
| $\boxed{-0.569}$ | $\boxed{0.845}$ | $\boxed{1}$ |

| $x_1$ | $x_2$ | $t$ |
|---|---|---|
| 1.342 | 1.182 | 2 |
| 2.568 | 1.583 | 2 |
| 2.515 | 2.034 | 2 |
| 1.384 | 2.211 | 2 |
| 2.926 | 2.310 | 2 |
| 0.714 | 1.808 | 2 |
| 3.430 | 2.011 | 2 |
| 1.575 | 1.983 | 2 |
| 1.597 | 1.430 | 2 |
| 2.604 | 2.264 | 2 |

| $x_1$ | $x_2$ | $t$ |
|---|---|---|
| 4.393 | 0.059 | 3 |
| 4.584 | 1.761 | 3 |
| 3.370 | 1.720 | 3 |
| 4.192 | 1.379 | 3 |
| 4.649 | 2.073 | 3 |
| 3.849 | 1.173 | 3 |
| 4.401 | 1.839 | 3 |
| 4.494 | 2.292 | 3 |
| 5.567 | 2.538 | 3 |
| 3.928 | 2.438 | 3 |



### 12.1.3 Probabilistic Prediction

Using the probability theory, we can find a prediction function from

input

$$y = \hat{f}(\mathbf{x}) = \operatorname*{argmax}_{c=1}^{C} P(\underbrace{t = c}_{\text{output } = 0}|\mathbf{x}, \mathcal{D})$$

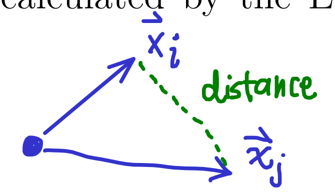
a training set

This is known as a **MAP estimate**.[4]

## 12.2 $K$-Nearest Neighbors

A simple technique to predict the class of an unknown feature vector $\mathbf{x}$ is to check the classes of the $K$ nearest vectors in the training set. Then, predicting the class of $\mathbf{x}$ using the majority class.

The distance between two vectors can be simply calculated by the Euclidean norm of the difference of the two vectors i.e.




$$\begin{aligned} \text{dist}(\mathbf{x}_i, \mathbf{x}_j) &= \|\mathbf{x}_i - \mathbf{x}_j\| \\ &= \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + \cdots + (x_{iD} - x_{jD})^2} \end{aligned}$$

$$\vec{x}_i = \begin{bmatrix} x_{i1} \\ x_{i2} \\ \vdots \\ x_{iD} \end{bmatrix} \qquad \vec{x}_j = \begin{bmatrix} x_{j1} \\ x_{j2} \\ \vdots \\ x_{jD} \end{bmatrix}$$

Formally, we can compute the prediction probability from

$$P(t = c|\mathbf{x}, \mathcal{D}, K) = \frac{1}{K} \sum_{i \in N_K(\mathbf{x}, \mathcal{D})} \mathbb{I}(t_i = c)$$

where $N_K(\mathbf{x}, \mathcal{D})$ is the set of $K$ nearest points to $\mathbf{x}$ in $\mathcal{D}$, and $\mathbb{I}(e)$ is the indicator function defined as

$$\mathbb{I}(e) = \begin{cases} 1 & \text{if } e \text{ is true} \\ 0 & \text{if } e \text{ is false} \end{cases}$$

[4]MAP stands for *maximum a posteriori*

**Exercise 12.1** From the following dataset, use the $K$-nearest neighbors technique to predict the classes of the following examples with different values of $K$.

| $x_1$ | $x_2$ | $t$ | $x_1$ | $x_2$ | $t$ |
|---|---|---|---|---|---|
| 0.5 | 0.5 | 0 | 2.0 | 1.5 | 1 |
| 0.2 | 1.0 | 0 | 3.0 | 1.0 | 1 |
| 1.0 | 0.8 | 0 | 4.0 | 3.2 | 1 |

nearest neighbors

1. $\overset{x_1}{[0.0}, \overset{x_2}{0.0]}^{\mathsf{T}}$ using $K = 3$

$d\left(\begin{bmatrix}0.0\\0.0\end{bmatrix}, \begin{bmatrix}0.5\\0.5\end{bmatrix}\right) = \sqrt{(0.0-0.5)^2+(0.0-0.5)^2} = \sqrt{0.5} \ \ast \rightarrow 0$

$d\left(\begin{bmatrix}0.0\\0.0\end{bmatrix}, \begin{bmatrix}0.2\\1.0\end{bmatrix}\right) = \sqrt{(0.0-0.2)^2+(0.0-1.0)^2} = \sqrt{1.04} \ \ast \rightarrow 0$

$d\left(\begin{bmatrix}0.0\\0.0\end{bmatrix}, \begin{bmatrix}1.0\\0.8\end{bmatrix}\right) = \sqrt{(0.0-1.0)^2+(0.0-0.8)^2} = \sqrt{1.64} \ \ast \rightarrow 0$

$d\left(\begin{bmatrix}0.0\\0.0\end{bmatrix}, \begin{bmatrix}2.0\\1.5\end{bmatrix}\right) = \sqrt{(0.0-2.0)^2+(0.0-1.5)^2} = \sqrt{6.25}$

$d\left(\begin{bmatrix}0.0\\0.0\end{bmatrix}, \begin{bmatrix}3.0\\1.0\end{bmatrix}\right) = \sqrt{(0.0-3.0)^2+(0.0-1.0)^2} = \sqrt{10}$

$d\left(\begin{bmatrix}0.0\\0.0\end{bmatrix}, \begin{bmatrix}4.0\\3.2\end{bmatrix}\right) = \sqrt{(0.0-4.0)^2+(0.0-3.2)^2} = \sqrt{26.24}$

Predict "0" for $\begin{bmatrix}0.0\\0.0\end{bmatrix}$

2. $[1.0, 1.0]^{\mathsf{T}}$ using $K = 5$

$d\left(\begin{bmatrix}1.0\\1.0\end{bmatrix}, \begin{bmatrix}0.5\\0.5\end{bmatrix}\right) = \sqrt{0.5} \ \ast \rightarrow 0$

$d\left(\begin{bmatrix}1.0\\1.0\end{bmatrix}, \begin{bmatrix}0.2\\1.0\end{bmatrix}\right) = \sqrt{0.64} \ \ast \rightarrow 0$

$d\left(\begin{bmatrix}1.0\\1.0\end{bmatrix}, \begin{bmatrix}1.0\\0.8\end{bmatrix}\right) = \sqrt{0.04} \ \ast \rightarrow 0$

$d\left(\begin{bmatrix}1.0\\1.0\end{bmatrix}, \begin{bmatrix}2.0\\1.5\end{bmatrix}\right) = \sqrt{1.25} \ \ast \rightarrow 1$

$d\left(\begin{bmatrix}1.0\\1.0\end{bmatrix}, \begin{bmatrix}3.0\\1.0\end{bmatrix}\right) = \sqrt{4} \ \ast \rightarrow 1$

$d\left(\begin{bmatrix}1.0\\1.0\end{bmatrix}, \begin{bmatrix}4.0\\3.2\end{bmatrix}\right) = \sqrt{13.84}$

Predict "0" for $\begin{bmatrix}1.0\\1.0\end{bmatrix}$

input = $\begin{bmatrix}2.0\\2.5\end{bmatrix}$

$d\left(\begin{bmatrix}2.0\\2.5\end{bmatrix}, \begin{bmatrix}0.5\\0.5\end{bmatrix}\right) = \sqrt{8.5}$

$d\left(\begin{bmatrix}2.0\\2.5\end{bmatrix}, \begin{bmatrix}0.2\\1.0\end{bmatrix}\right) = \sqrt{5.49}$

$0 \leftarrow \ast\, d\left(\begin{bmatrix}2.0\\2.5\end{bmatrix}, \begin{bmatrix}1.0\\0.8\end{bmatrix}\right) = \sqrt{3.89}$

$1 \leftarrow \ast\, d\left(\begin{bmatrix}2.0\\2.5\end{bmatrix}, \begin{bmatrix}2.0\\1.5\end{bmatrix}\right) = \sqrt{1}$

$1 \leftarrow \ast\, d\left(\begin{bmatrix}2.0\\2.5\end{bmatrix}, \begin{bmatrix}3.0\\1.0\end{bmatrix}\right) = \sqrt{3.25}$

$d\left(\begin{bmatrix}2.0\\2.5\end{bmatrix}, \begin{bmatrix}4.0\\3.0\end{bmatrix}\right) = \sqrt{4.25}$

input → calculate distances to all training vectors → find vectors with K smallest distances → voting

## 12.3 Model Selection

Since we can run KNN using different value of $K$, which value of $K$ should we use?

A basic way to evaluate a classifier is to use the *misclassification rate* on the training set. It can be defined as:

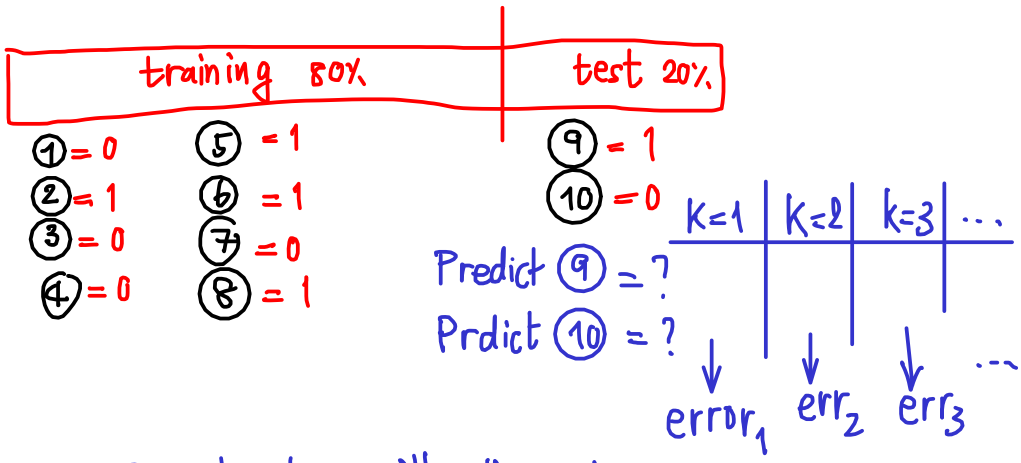
$$\mathrm{err}(f, \mathcal{D}) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}(f(\mathbf{x}_i) \neq t_i)$$

However, the error on the training set should not represent the *generalization error*.

### 12.3.1 Holdout Method

The collected dataset is split into two subsets, i.e. *training* and *validation* sets. Generally, about 80% of the data are for the training set, and 20% of the data are for the validation set.

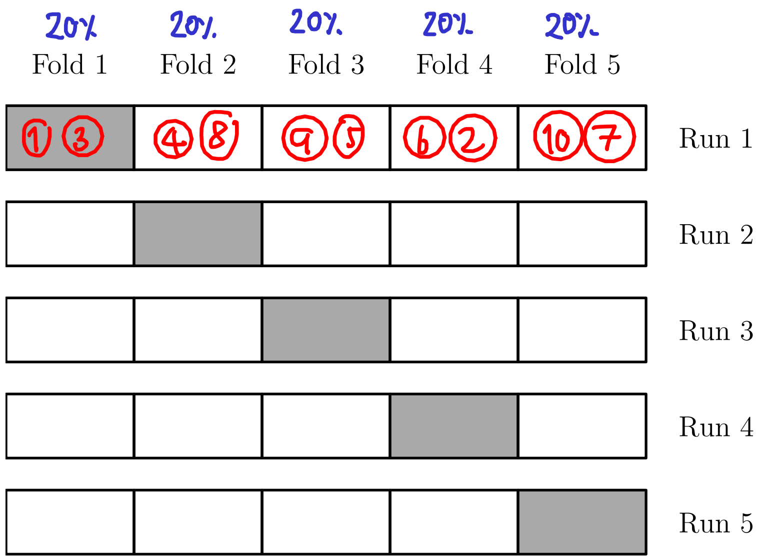
However, the evaluation result depends heavily on the examples in the training set and the validation set. We may get too good or too bad result easily.




Select K with the minimum error

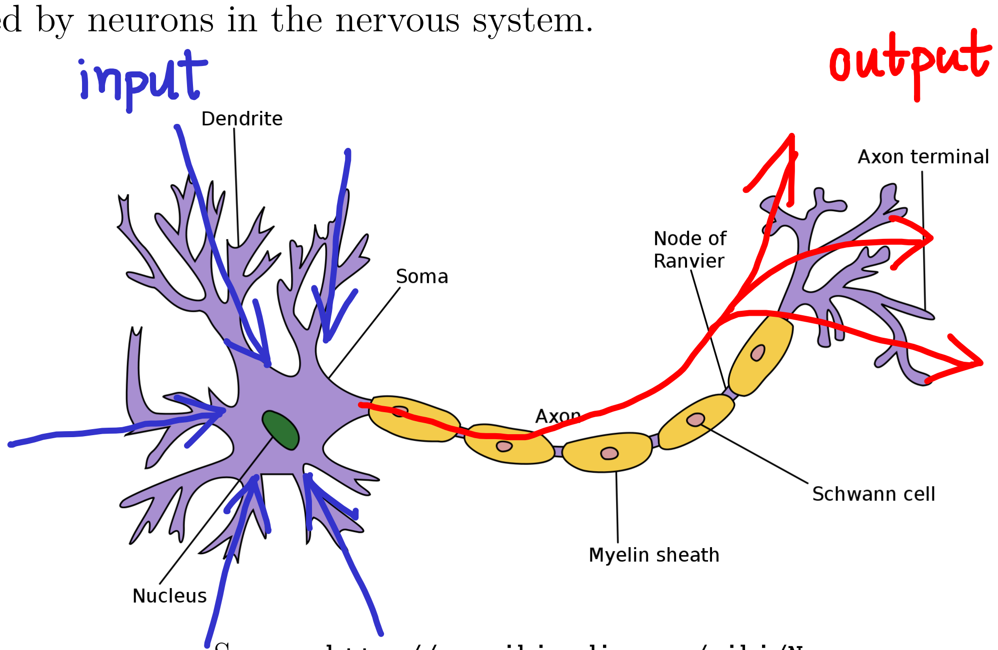
### 12.3.2 Cross Validation Method

The cross validation method splits the dataset into $k$ folds. Then, it runs the holdout evaluation $k$ runs. In each run, the $k$th fold is used as the validation set, while the combination of the rest folds are used as the training set in that run.



## 12.4 Artificial Neural Networks

**Artificial Neural Network** (ANN) is a supervised learning technique inspired by neurons in the nervous system.

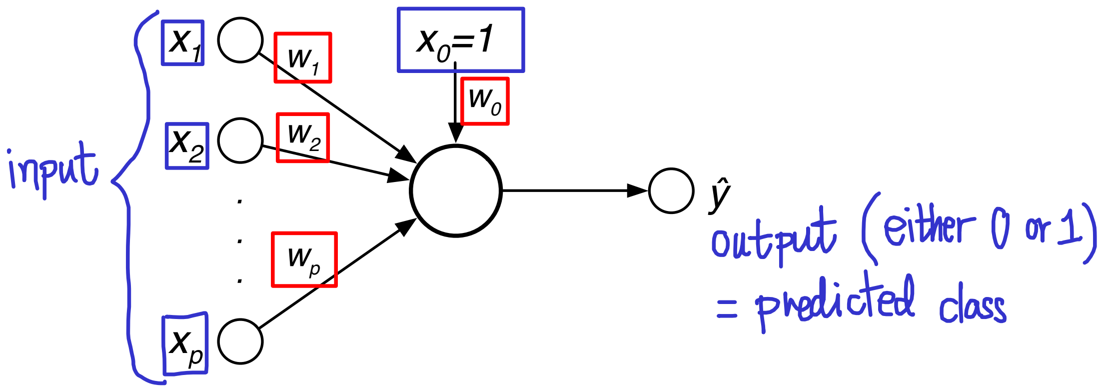



Source: `http://en.wikipedia.org/wiki/Neuron`

two classes of output

**Perceptron** is a type of ANN working as a kind of binary classification based on a linear model.

$x_0$ is always 1.




Each perceptron has several inputs expressing instance features denoting $(x_0, x_1, x_2, \ldots, x_p)^\mathsf{T}$. A weight is attached to each input as a vector $(w_0, w_1, w_2, \ldots, w_p)^\mathsf{T}$. These weights are the important part denoting the main characteristic of each perceptron. An output of perceptron denotes the prediction result.

A perceptron works as **a linear classifier** getting features as inputs and estimating the class. The prediction is done by

Given an input vector $(x_1, x_2, \ldots, x_p)^T$, we have

predicted output

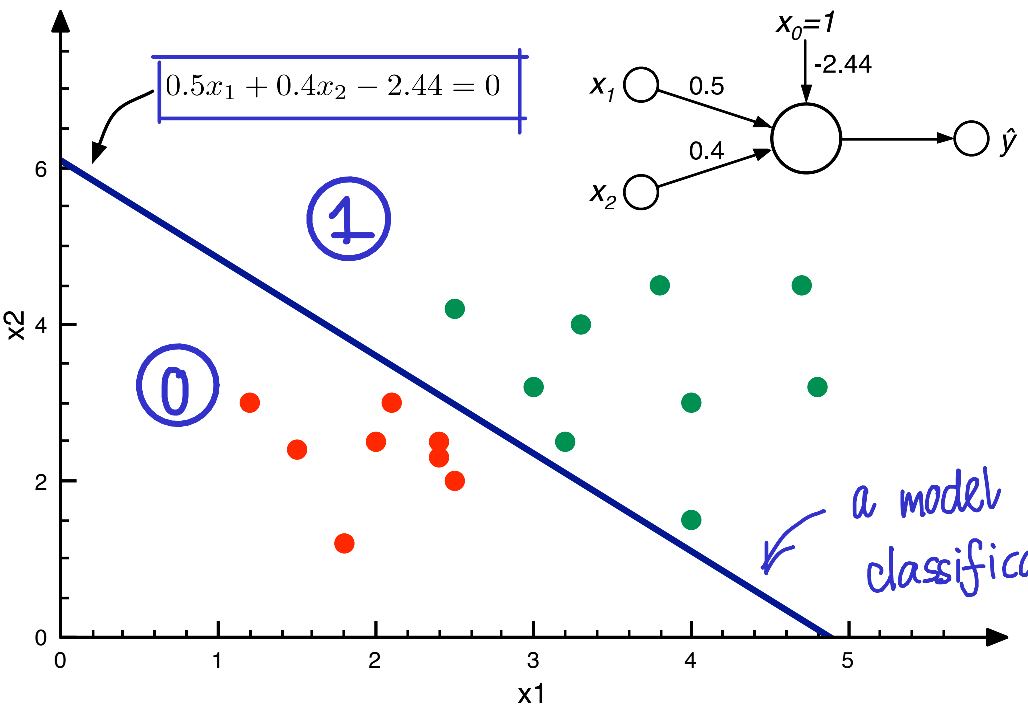
$$\boxed{y} = \mathrm{sgn}\left(\sum_{i=0}^{p} w_i x_i\right)$$
$$= \mathrm{sgn}(w_0 + \underbrace{\boxed{w_1}\boxed{x_1} + \cdots + \boxed{w_p}\boxed{x_p}}_{\text{weighted sum of the input vector}})$$

where, $\mathrm{sgn}(x) = \begin{cases} 1 & \text{if } x > 0, \\ 0 & \text{otherwise} \end{cases}$

the sign function

This classifier deals with a two-class problem (1 and 0). We have $w_0 + w_1x_1 + \cdots + w_px_p = 0$ as a linear hyperplane.

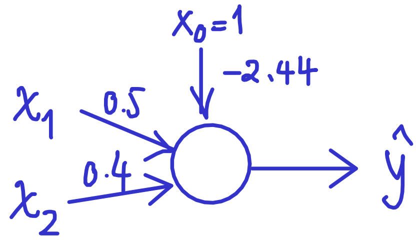
$\hat{y} = \mathrm{sgn}(-2.44 + 0.5x_1 + 0.4x_2)$




a model for classification

a perceptron (with appropriate weights) = a line that classified the space into two areas (1, 0)

**Exercise 12.2** Use the perceptron in the previous figure to predict the following examples.




1. $[0.0, 0.0]^{\mathsf{T}}$ ($x_1$, $x_2$)

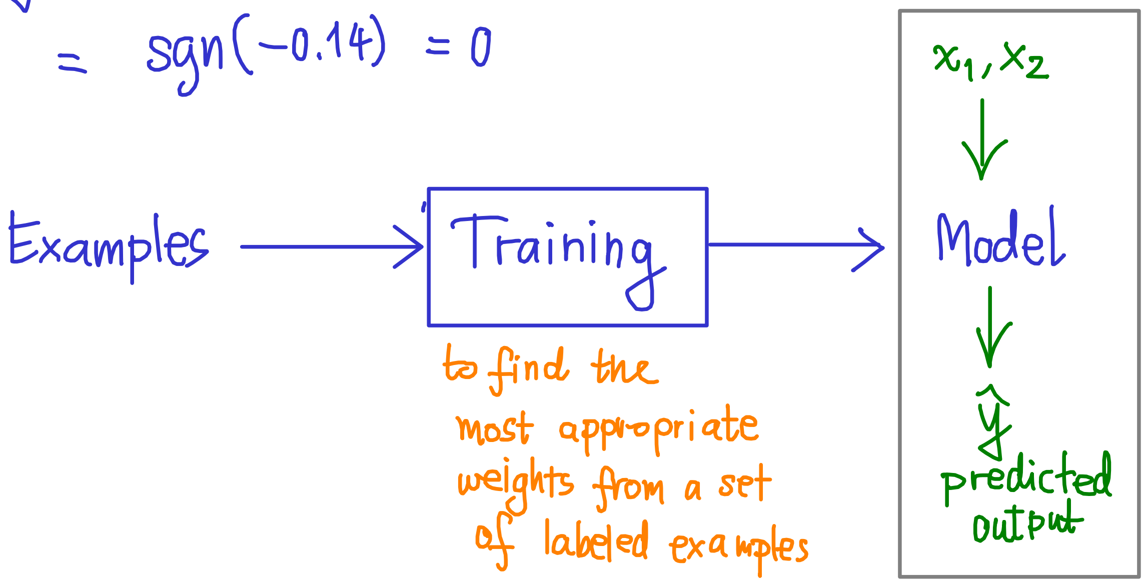
$$\hat{y} = \mathrm{sgn}(-2.44 + (0.5)(0) + (0.4)(0))$$
$$= \mathrm{sgn}(-2.44) = 0$$

2. $[4.0, 4.0]^{\mathsf{T}}$

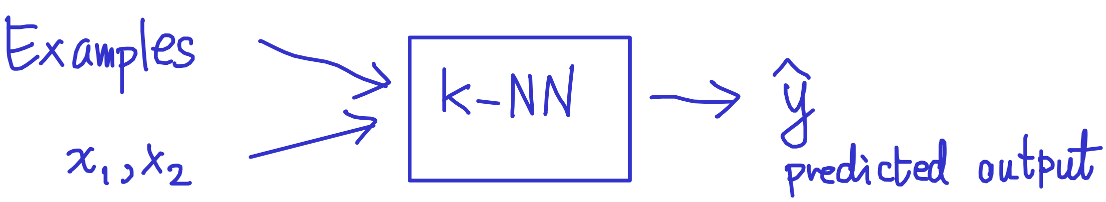
$$\hat{y} = \mathrm{sgn}(-2.44 + (0.5)(4.0) + (0.4)(4.0))$$
$$= \mathrm{sgn}(1.16) = 1$$

3. $[3.0, 2.0]^{\mathsf{T}}$

$$\hat{y} = \mathrm{sgn}(-2.44 + (0.5)(3.0) + (0.4)(2.0))$$
$$= \mathrm{sgn}(-0.14) = 0$$




---



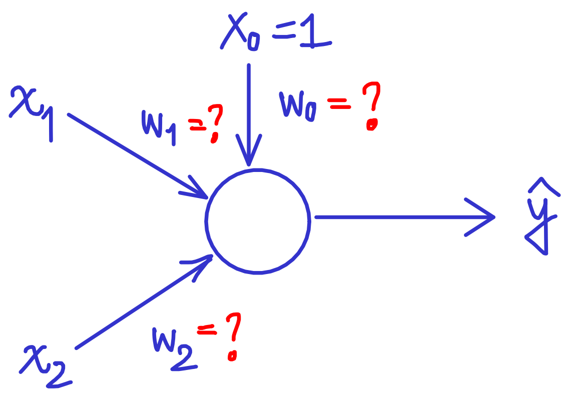
inputs target output

**Exercise 12.3** Answer the questions from the following dataset.

Training examples

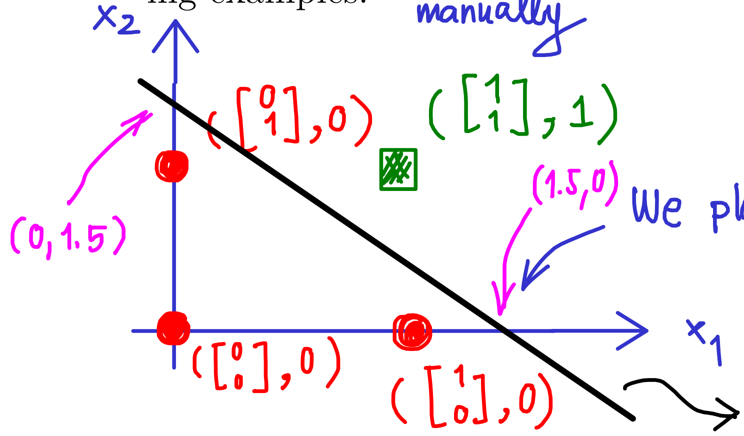
| $x_1$ | $x_2$ | t |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

1. Draw the structure of a perceptron for learning from this dataset.




We need to find an appropriate vector of weights from the given dataset.

2. Find the weights that makes the perceptron correctly classify the training examples. manually




We place a line here to make it correctly classify the training examples.

$w_0 + w_1x_1 + w_2x_2 = 0$

Let's find the weights from this line.

From (0, 1.5), we have $w_0 + w_1(0) + w_2(1.5) = 0$

$w_0 + 1.5w_2 = 0$ —— (1)

From (1.5, 0), we have $w_0 + w_1(1.5) + w_2(0) = 0$

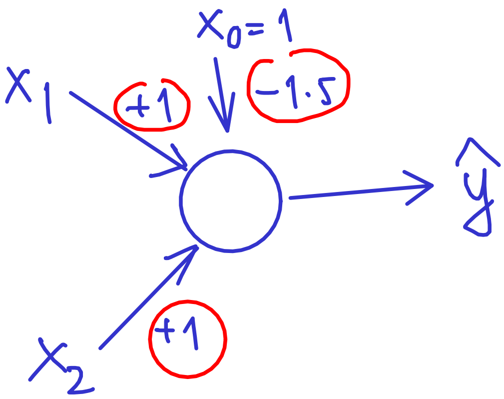
$w_0 + 1.5w_1 = 0$ —— (2)

(2)−(1) $1.5w_1 - 1.5w_2 = 0 \longrightarrow w_1 = w_2$

when $w_2 = +1 \longrightarrow w_1 = +1$

substitute $w_2$ to (1) $\longrightarrow w_0 = -1.5$

$$-1.5 + x_1 + x_2 = 0$$



### 12.4.1 Perceptron Training Algorithm

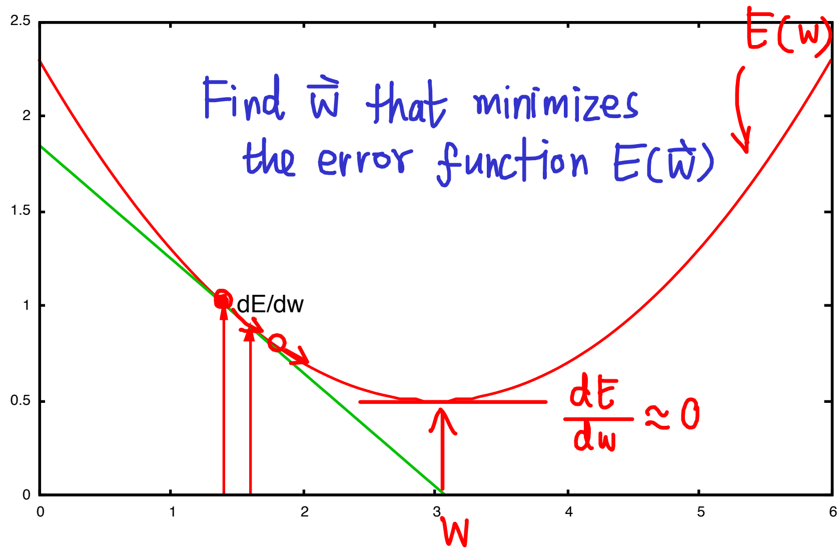
Weights are the main component of each perceptron. If each weight is set to a suitable value, the perceptron will be able to classify unlabeled instances. We need a technique to assign or adjust weights until we obtain the most suitable values.

## Obtaining the Weight Updating Rule

*Gradient Descent* technique can be used to find the most suitable weights. It is a simple technique to solve optimization problem on a continuous environment.

weight vector

1. Define an objective function. In this case, it should be a function of **w** i.e. $E(\mathbf{w})$.
   error function
2. Randomly select initial values for **w**.
3. Update weights using $\boxed{\mathbf{w} - \eta \frac{dE}{d\mathbf{w}}}$ where $\eta$ is a positive small number.
4. Iteratively update until $\frac{dE}{d\mathbf{w}} \approx 0$




We can define an objective function using weights **w** as a parameter. The function should return 0 if the weights are adjusted to the best values. In the simplest way, we can define an objective function as

Error function

target output

$$E(\mathbf{w}) \stackrel{d}{=} \frac{1}{2}\sum_{d=1}^{N}(t_d - y_d)^2$$

predicted output

$$= \frac{1}{2}\sum_{d=1}^{N}\left(t_d - \boxed{\sum_{i=0}^{p} w_i x_{di}}\right)$$

→ a perceptron

We compute gradient of the objective function which is denoted as

$$\boxed{\nabla E(\mathbf{w})} \stackrel{d}{=} \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_p}\right]$$

Then, we find the partial differentiate of $E$ by $w_i$:

$$\begin{aligned}
\frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i}\left(\frac{1}{2}\sum_{d=1}^{N}(t_d - y_d)^2\right) \\
&= \frac{1}{2}\sum_{d=1}^{N}\frac{\partial}{\partial w_i}\left((t_d - y_d)^2\right) \\
&= \frac{1}{2}\sum_{d=1}^{N}2(t_d - y_d)\frac{\partial}{\partial w_i}\left(t_d - y_d\right) \\
&= \sum_{d=1}^{N}(t_d - y_d)\frac{\partial}{\partial w_i}\left(t_d - \sum_{i=0}^{p} w_i x_{di}\right) \\
&= \boxed{\sum_{d=1}^{N}(t_d - y_d)(-x_{di})} = \nabla E(w_i)
\end{aligned}$$

a new weight $w_i$

the current $w_i$

From the gradient of $E$, we have

$$\boxed{w_i^* = w_i - \eta \nabla E(w_i)}$$

input

We then update $w_i$ by

$$\boxed{w_i \leftarrow w_i + \eta(t - y)(x_i)}$$

→ Weight updating rule

target output

predicted output

feed an input vector ⟶ make a prediction

if there is a error ⟶ update weights

$(t \neq y)$

find the most appropriate weights for a perceptron.

A training algorithm for perceptron is as below:

1. Start with random weights.

$t \neq y$

2. Apply each training instance. If the perceptron misclassifies, then adjust all the weights. Each weight $w_i$ for an input $x_i$ is adjusted by

$$\boxed{w_i \leftarrow w_i + \eta(t - y)x_i}$$

where $\eta$ is a positive constant called the *learning rate*.

3. Iteratively, perform step 2 through the training set until the perceptron classifies all training instances correctly.

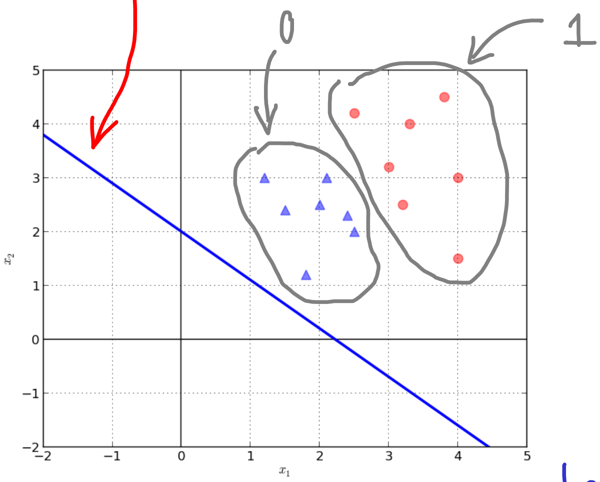
**Example 12.5** Find the most suitable weights from the following examples:

| No | $x_1$ | $x_2$ | $t$ | No | $x_1$ | $x_2$ | $t$ |
|---|---|---|---|---|---|---|---|
| 1 | 2.5 | 2.0 | 0 | 8 | 4.0 | 3.0 | 1 |
| 2 | 1.2 | 3.0 | 0 | 9 | 3.8 | 4.5 | 1 |
| 3 | 2.1 | 3.0 | 0 | 10 | 3.2 | 2.5 | 1 |
| 4 | 2.4 | 2.3 | 0 | 11 | 3.3 | 4.0 | 1 |
| 5 | 2.0 | 2.5 | 0 | 12 | 2.5 | 4.2 | 1 |
| 6 | 1.5 | 2.4 | 0 | 13 | 4.0 | 1.5 | 1 |
| 7 | 1.8 | 1.2 | 0 | 14 | 3.0 | 3.2 | 1 |

We start from random weights and the learning rate:

$$\boxed{w_0 = -0.40; \quad w_1 = 0.18; \quad w_2 = 0.20;} \quad \eta = 0.01$$

From this setting, we have a linear classifier as the following figure.




#1

We start from applying an instance $([2.5, 2.0]^{\mathsf{T}}), 0)$ to the perceptron, then

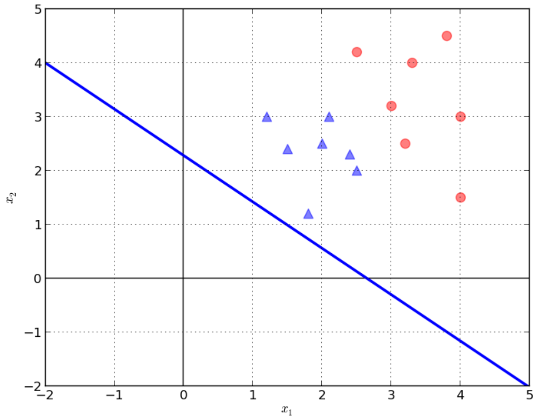
target output

$$\begin{aligned} y &= \mathrm{sgn}(0.18 \times 2.5 + 0.2 \times 2.0 - 0.4) \\ &= \mathrm{sgn}(0.45) \\ &= 1 \end{aligned}$$

predicted output

Since $y \neq t$, then we adjust weight as follow:

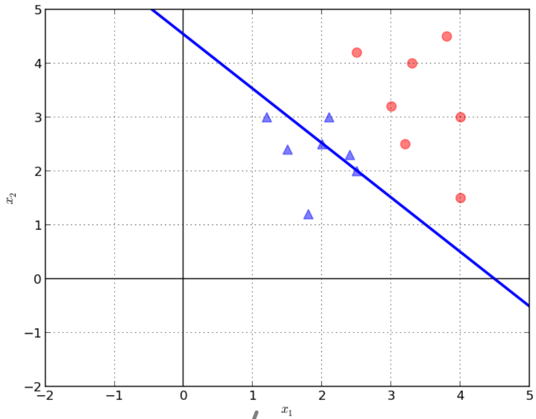
$$\begin{aligned} w_0 &\leftarrow -0.40 + ((0.01)(0-1)(1.0)) = -0.410 \\ w_1 &\leftarrow +0.18 + ((0.01)(0-1)(2.5)) = +0.155 \\ w_2 &\leftarrow \underbrace{+0.20}_{\text{current } w_i} + ((\underbrace{0.01}_{\eta})(\underbrace{0}_{t}-\underbrace{1}_{y})(\underbrace{2.0}_{x_i})) = \underbrace{+0.180}_{\text{new } w_i} \end{aligned}$$




#2 ~ #14

After applying all the remaining 13 examples, we will get weights as:

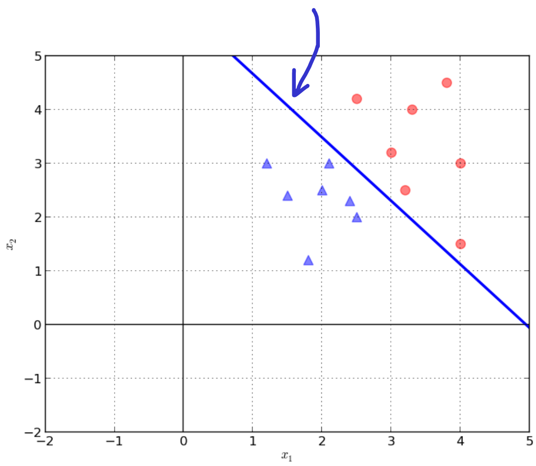
$$w_0 = -0.440; \quad w_1 = 0.098; \quad w_2 = 0.097$$




#1 ~ #14 (5 times)

After updating the weights for five rounds, the weights become:

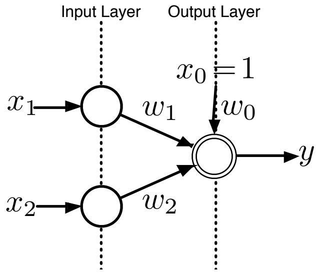
$$w_0 = -0.450; \quad w_1 = 0.091; \quad w_2 = 0.077$$



## 12.4.2 Threshold Function

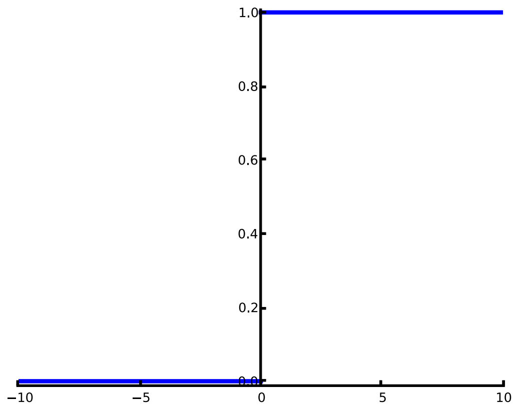
A perceptron outputs either '0' or '1' based on the sum of the inputs and the weights, as well as the threshold function (e.g. sgn(·)).

$$y = \text{sgn}(w_0 + w_1x_1 + w_2x_2)$$




The theshold function sgn(·) is a discontinuous function. It cannot be differentiated. Therefore, it cannot be used together with the gradient descent, or other optimization techniques.

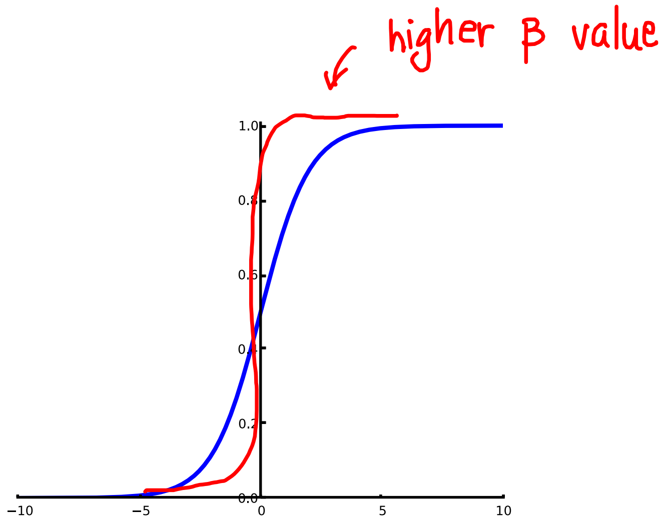
$$\text{sgn}(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$




The sigmoid function is an S-shaped function. It can be in place of the function sgn(·).

$$\boxed{\text{sigmoid}(x) = \frac{1}{1 + e^{-\beta x}}}$$

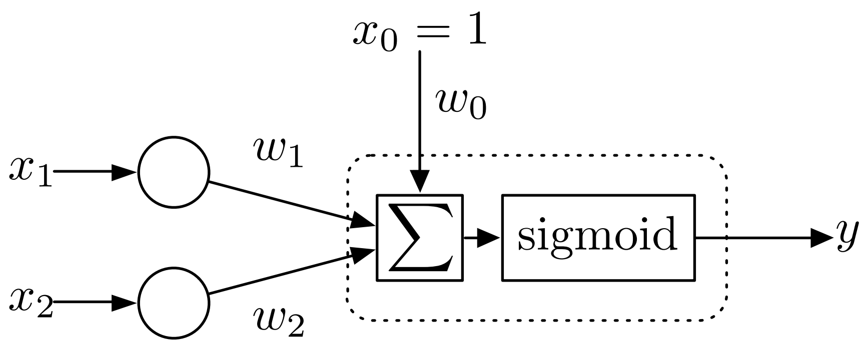
β = a positive constant




### 12.4.3 Sigmoid Unit

Now, an output of a perceptron can be calculated from:

$$y = \text{sigmoid}(w_0 + w_1 x_1 + w_2 x_2)$$

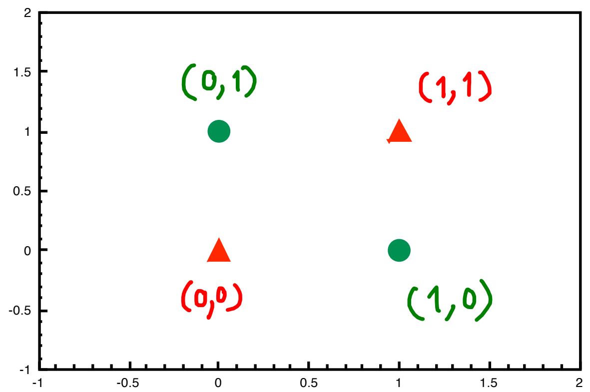


## 12.5 Linear Separability and Multilayer Perceptron

Since a perceptron represents a linear model for classification, it works very well on a training set that is **linearly separable**. However, a perceptron cannot deal with the training set that cannot be separated by just a line.

XOR Problem

| $x_1$ | $x_2$ | $t$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |



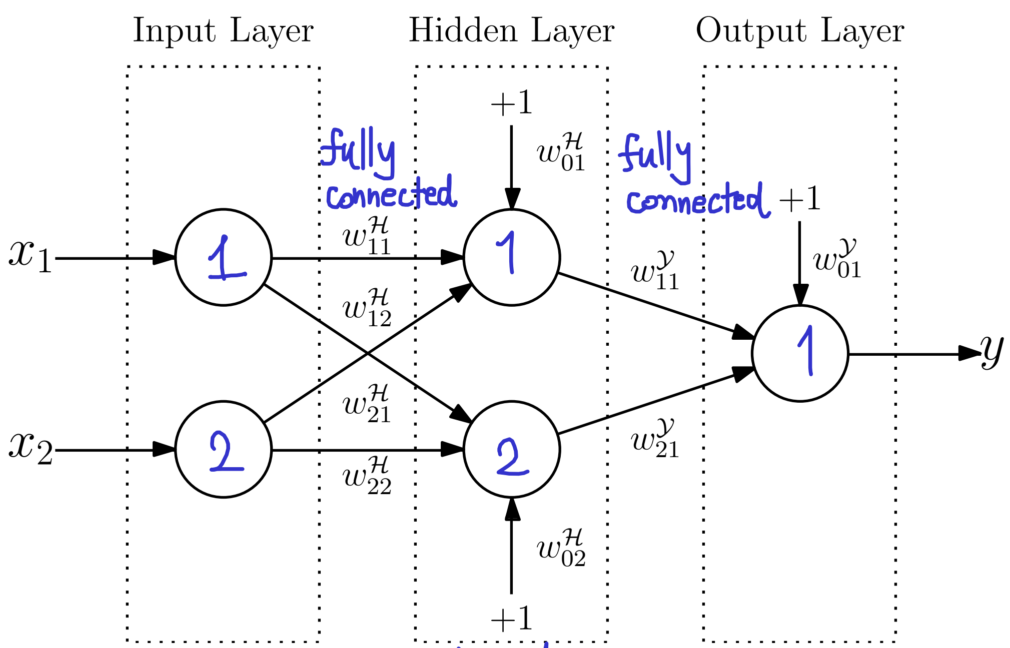

non-linearly separable

a perceptron = a line (a hyperplane)

How can the perceptron be improved to handle this situation?

To solve the non-linearly separable problem, we connect multiple sigmoid units to become a neural network.

The units are arranged into more than two layers. The technique is called *multilayer perceptron*. = a neural network




sigmoid units

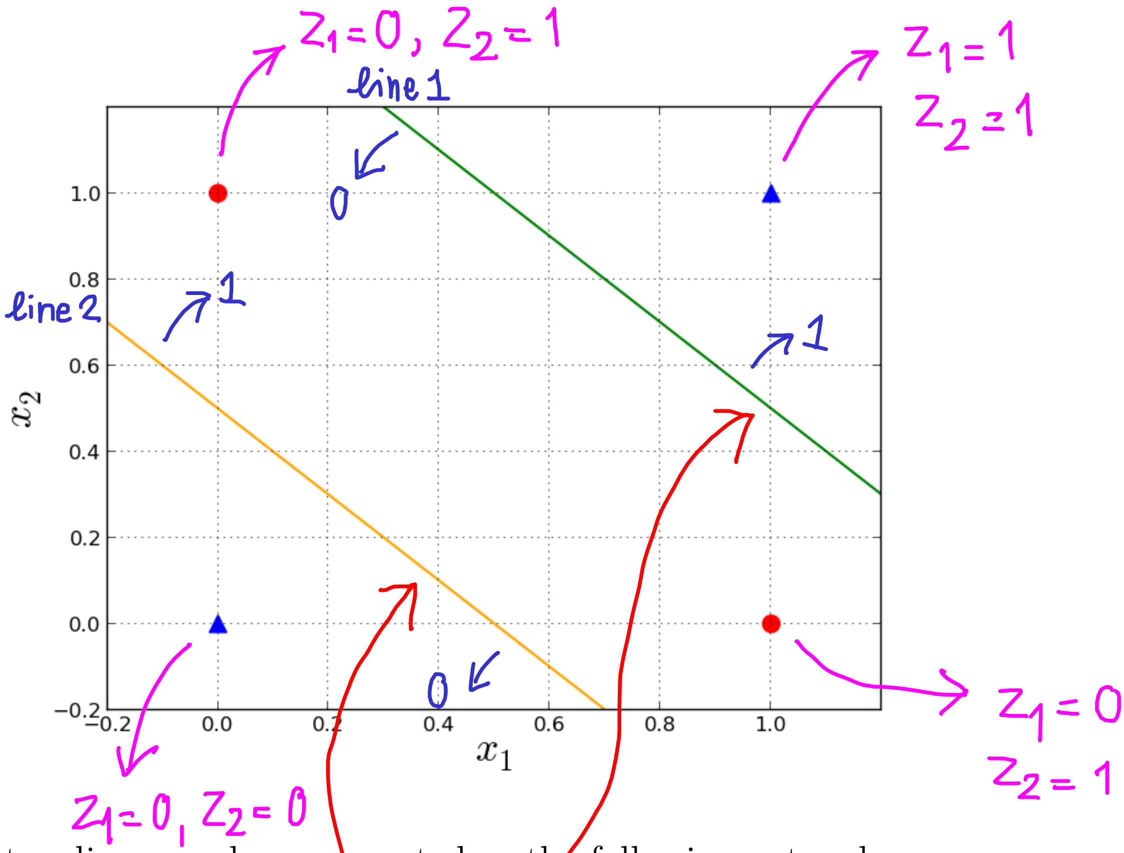
sigmoid unit

$w^{H}_{xx}$ = a weight for a hidden unit

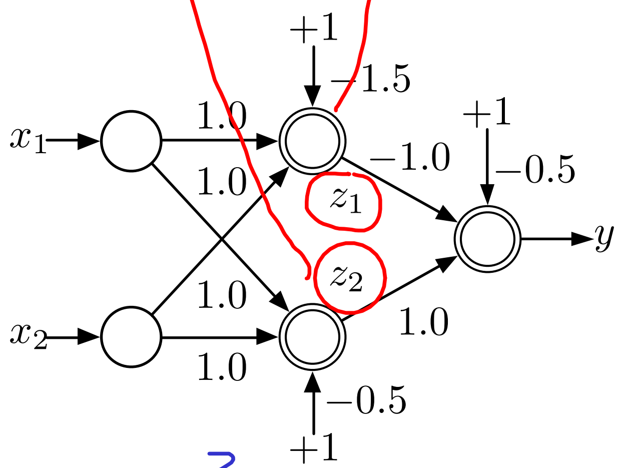
$w^{Y}_{xx}$ = a weight for an output unit

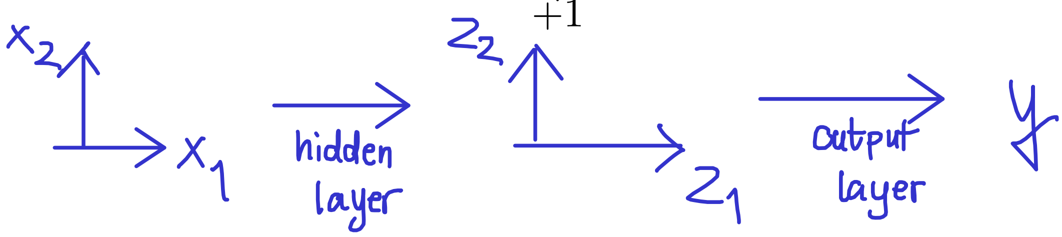
Each perceptron works as a linear discriminant. The inputs are classified by the linear discriminants in the hidden layer. Either 0 or 1 are yielded from each unit in the hidden layer. The outputs from the hidden layer are fed as inputs to the output layer. Here, the perceptron in the output is a linear discriminate that classifies the pre-classified inputs.

**Example 12.6** We uses two lines to correctly discriminate the examples:




These two lines can be represented as the following network:



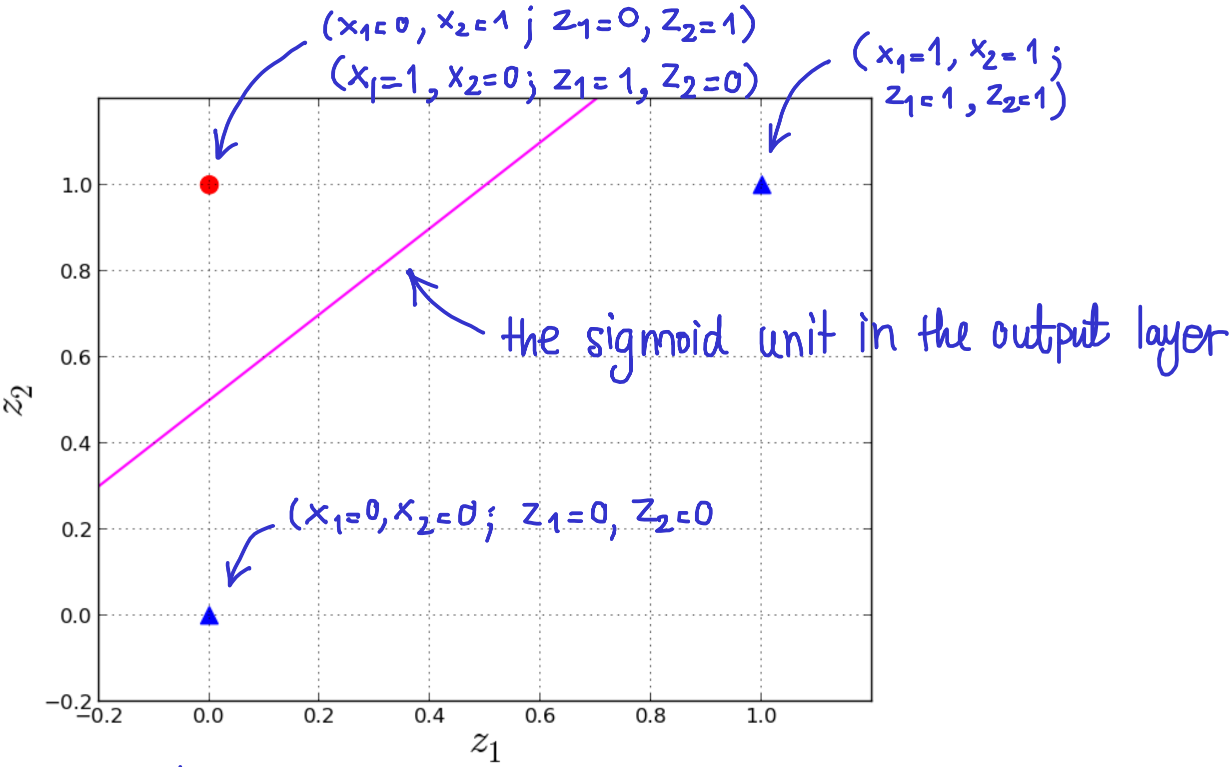



$$z_1 = \text{sigmoid}(x_1 + x_2 - 0.5)$$
$$z_2 = \text{sigmoid}(x_1 + x_2 - 1.5)$$

From the network, we have two intermediate outputs (i.e. $z_1$, and $z_2$) from the perceptrons in the hidden layer: hidden layer

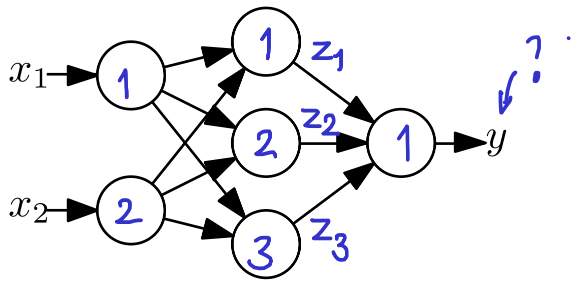
| $x_1$ | $x_2$ | $z_1$ | $z_2$ | $t$ |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 |

We can use two lines to correctly discriminate the examples:




The $z_1$–$z_2$ space is now linearly separable.

**Exercise 12.4** Use the multilayer perceptron with the following weights to predict the output of $[1.0, 1.0]^\mathsf{T}$. Set $\beta = 1$.




$$w^{\mathcal{H}}_{01} = +0.50, \quad w^{\mathcal{H}}_{11} = -0.50, \quad w^{\mathcal{H}}_{21} = +0.10,$$
$$w^{\mathcal{H}}_{02} = +1.00, \quad w^{\mathcal{H}}_{12} = +0.20, \quad w^{\mathcal{H}}_{22} = -0.20,$$
$$w^{\mathcal{H}}_{03} = -1.00, \quad w^{\mathcal{H}}_{13} = +0.10, \quad w^{\mathcal{H}}_{23} = +0.50,$$
$$w^{\mathcal{Y}}_{01} = -1.00, \quad w^{\mathcal{Y}}_{11} = +1.50, \quad w^{\mathcal{Y}}_{21} = -1.00, \quad w^{\mathcal{Y}}_{31} = 1.00$$

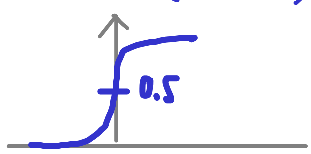
$$z_1 = \text{sigmoid}(w^H_{01} + w^H_{11}x_1 + w^H_{21}x_2) = \text{sigmoid}(0.5 + (-0.5)(1.0) + (0.1)(1.0))$$
$$= \text{sigmoid}(0.1) = \frac{1}{1+e^{-(1)(0.1)}} = 0.52$$

$$z_2 = \text{sigmoid}(w^H_{02} + w^H_{12}x_1 + w^H_{22}x_2) = \text{sigmoid}(1 + (0.2)(1) + (-0.2)(1))$$
$$= \text{sigmoid}(1) = 0.73$$

$$z_3 = \text{sigmoid}(w^H_{03} + w^H_{13}x_1 + w^H_{23}x_2) = \text{sigmoid}(-1 + (0.1)(1) + (0.5)(1))$$
$$= \text{sigmoid}(-0.4) = 0.40$$

$$y = \text{sigmoid}(w^Y_{01} + w^Y_{11}z_1 + w^Y_{21}z_2 + w^Y_{31}z_3)$$
$$= \text{sigmoid}(-1 + (1.5)(0.52) + (-1)(0.73) + (1)(0.40))$$
$$= \text{sigmoid}(-0.58) = 0.36$$

using the threshold of 0.5,
the predicted output is 0
(0.36 < 0.5)



### 12.5.1 Backpropagation Algorithm

The gradient descent technique is used to train a multilayer perceptron in the same manner as training a perceptron.

We first define the error function:

$$\boxed{E(\mathbf{w}^{\mathcal{H}}, \mathbf{w}^{\mathcal{Y}})} = \frac{1}{2}\sum_{d=1}^{N}(t_d - y_d)^2$$

We can use the chain rule to calculate the gradient

$$\frac{\partial E}{\partial w_{jl}^{\mathcal{H}}} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial z_l}\frac{\partial z_l}{\partial w_{jl}^{\mathcal{H}}}$$

Here, the error propagates from the output back to the inputs. Thus, the algorithm is named *backpropagation*.

For each example in $D$:

**Forward phase:** Given an input vector, compute the predicted output

1. compute the output of each perceptron in the hidden layer:

$$z_l = \text{sigmoid}\Big(\sum_i w_{il}^{\mathcal{H}} x_i\Big)$$

2. compute the output of each perceptron in the output layer:

$$y_k = \text{sigmoid}\Big(\sum_l w_{lk}^{\mathcal{Y}} z_l\Big)$$

**Backward phase:** Update $\vec{w}^H$ and $\vec{w}^Y$ according to the error $(t-y)$

1. compute the error at the output using: $\delta^Y$

$$\delta_k^{\mathcal{Y}} = (t_k - y_k)y_k(1 - y_k)$$

2. update the output layer weights using:

$$w_{lk}^{\mathcal{Y}} \leftarrow w_{lk}^{\mathcal{Y}} + \eta\delta_k^{\mathcal{Y}} z_l$$

3. compute the error in the hidden layer using:

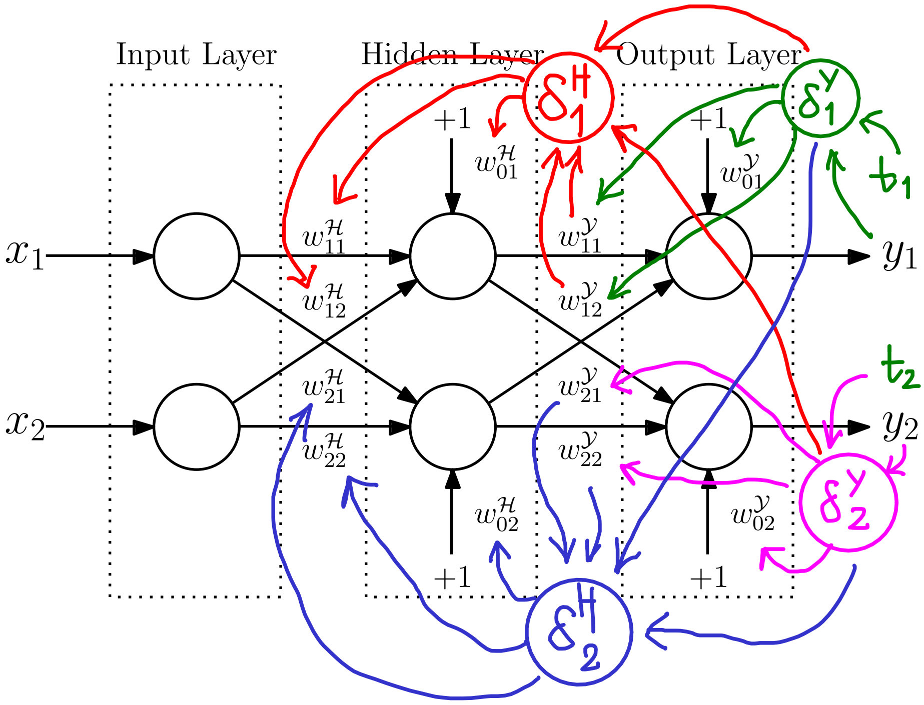
$\delta^H$

$$\delta_l^{\mathcal{H}} = z_l(1 - z_l)\sum_k w_{lk}^{\mathcal{Y}}\delta_k^{\mathcal{Y}}$$

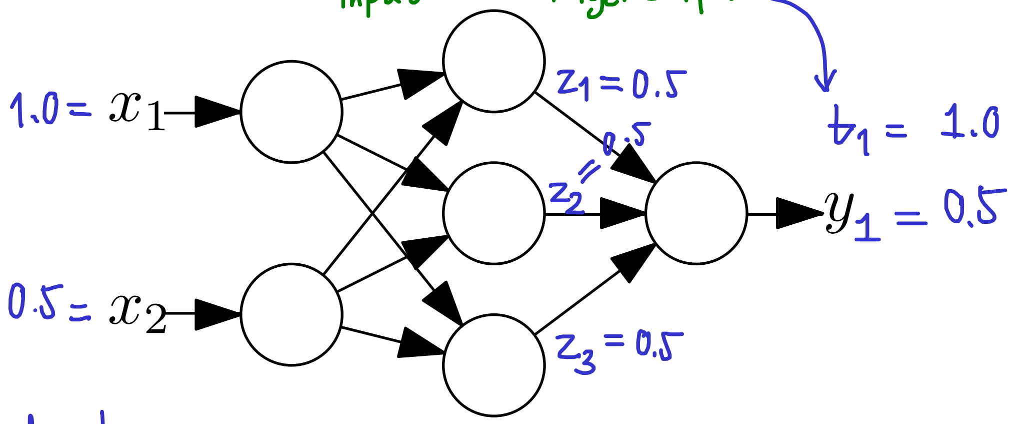
4. update the hidden layer weights using:

error in hidden layer

$$w_{il}^{\mathcal{H}} \leftarrow w_{il}^{\mathcal{H}} + \eta \delta_l^{\mathcal{H}} x_i$$



**Exercise 12.5** Given the following multilayer perceptron with all initial weights of $\boxed{0\text{'s}}$. Find the all updated values of the weights of all layers when we feed a labeled example $(\underbrace{[1.0, 0.5]^\top}_{\text{input}}, \underbrace{1}_{\text{target output}})$. Set $\beta = 1$ and $\eta = 0.01$.




Backward phase

$$\delta_1^y = (t_1 - y_1)(y_1)(1-y_1) = 0.125$$

$$w_{11}^y = 0 + \eta\delta_1^y z_1 = 0 + (0.01)(0.125)(0.5) = 0.000625$$

$$w_{21}^y = 0 + \eta\delta_1^y z_2 = 0.000625$$

$$w_{31}^y = 0 + \eta\delta_1^y z_3 = 0.000625$$

$$w_{01}^y = 0 + \eta\delta_1^y (1) = 0.00125$$

$$\delta_1^H = z_1(1-z_1)\sum_k w_{1k}^y \delta_k^y$$

k → #output units

$$= (0.5)(1-0.5)(0.000625)(0.125) = 1.95 \times 10^{-5}$$

$$w_{01}^H = 0 + \eta\delta_1^H(1) = 0 + (0.01)(1.95\times10^{-5})(1) = 1.95\times10^{-7}$$

$$w_{11}^H = 0 + \eta\delta_1^H x_1 = 0 + (0.01)(1.95\times10^{-5})(1) = 1.95\times10^{-7}$$

$$w_{21}^H = 0 + \eta\delta_1^H x_2 = 0 + (0.01)(1.95\times10^{-5})(0.5) = 9.75\times10^{-8}$$

$$\delta_2^H \rightarrow w_{02}^H,\ w_{12}^H,\ w_{22}^H$$

$$\delta_3^H \rightarrow w_{03}^H,\ w_{13}^H,\ w_{23}^H$$

# References


- Satish Kumar, “Neural networks: a classroom approach”, McGraw-Hill, 2005.
- Stephen Marsland, “Machine Learning: an algorithm perspective”, CRC Press, 2009.
- Ethem Alpaydin, “Introduction to Machine Learning”, The MIT Press, 2004.
- Kevin P. Murphy, “Machine Learning: A Probabilistic Perspective”, The MIS Press, 2012.