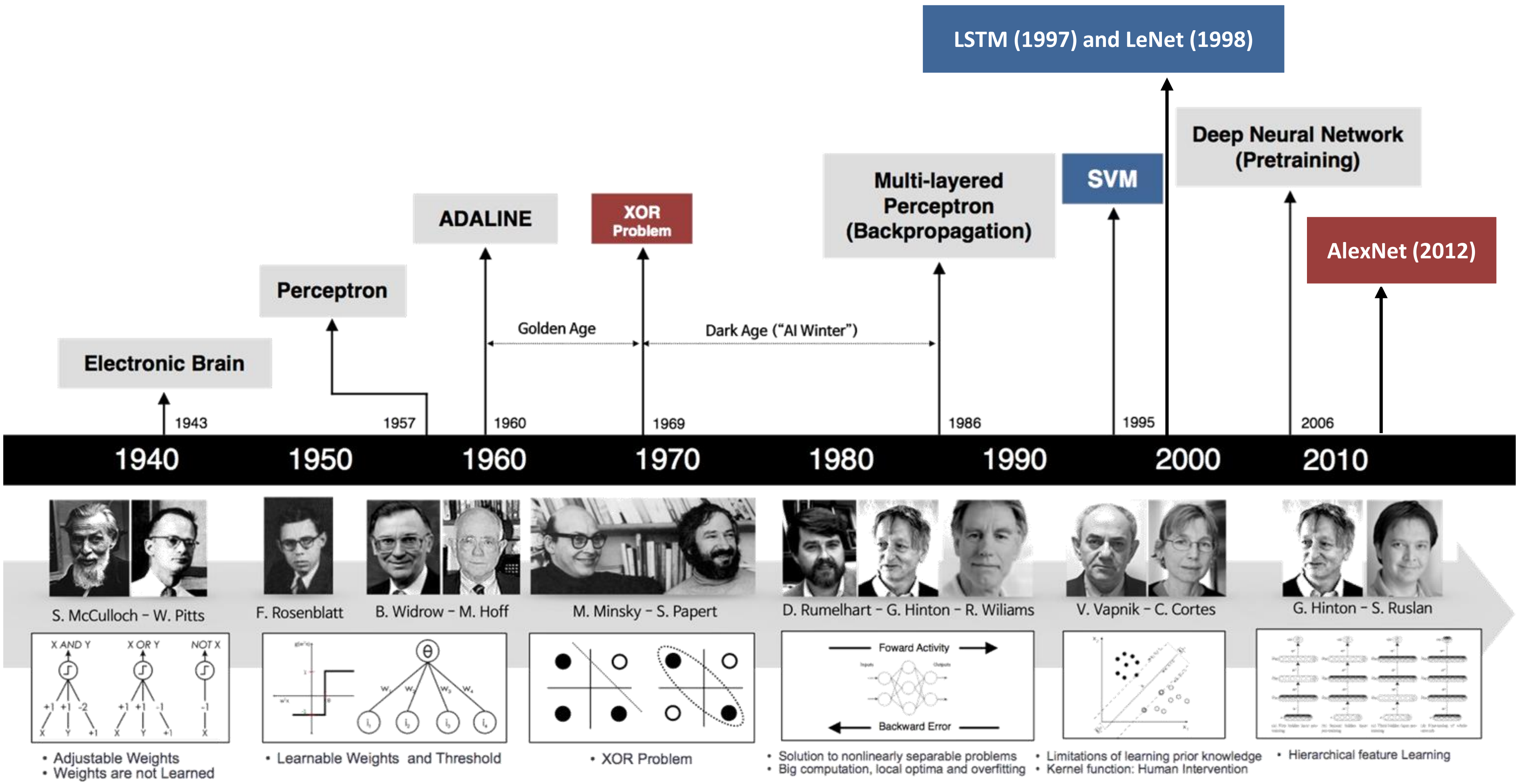
Neural Networks Basics

Data Intelligence and Learning (DIAL) Lab
Prof. Jongwuk Lee

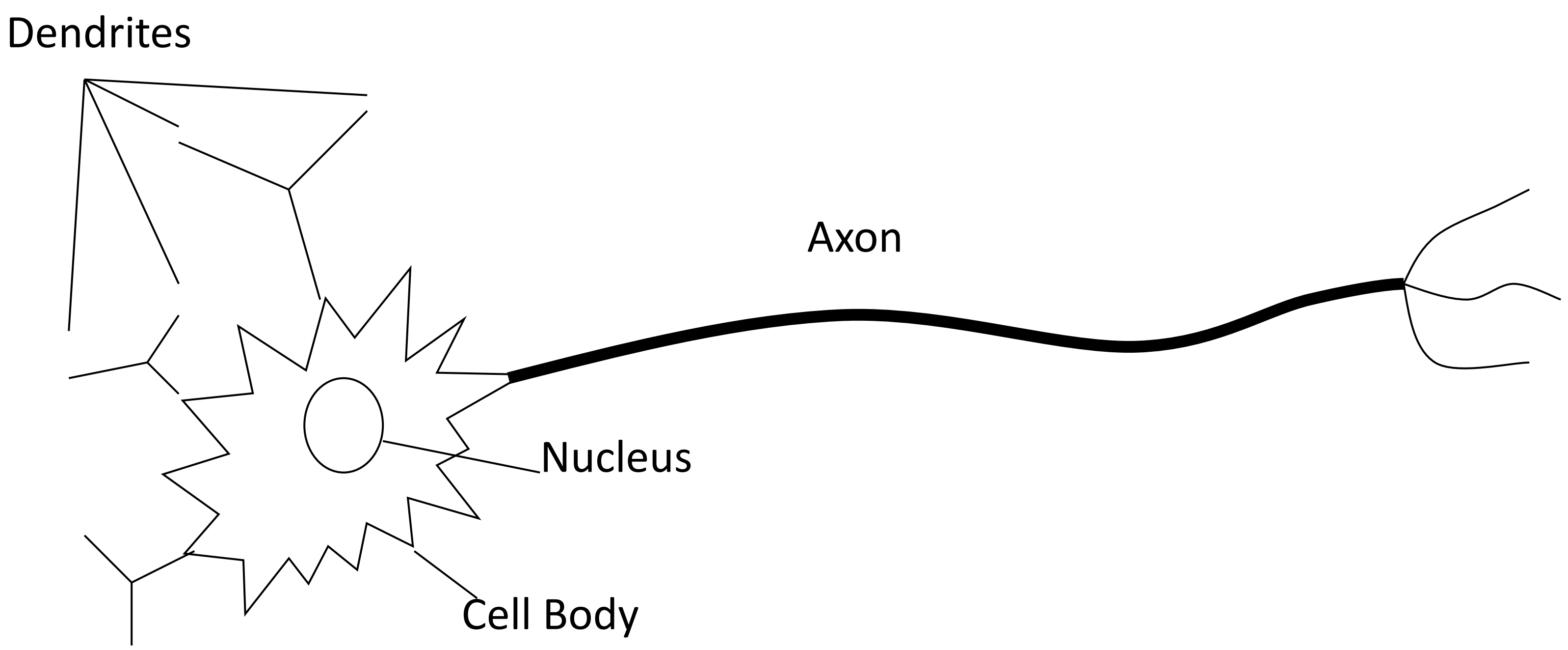
Perceptron

Brief History of Neural Networks



Concept of Neurons

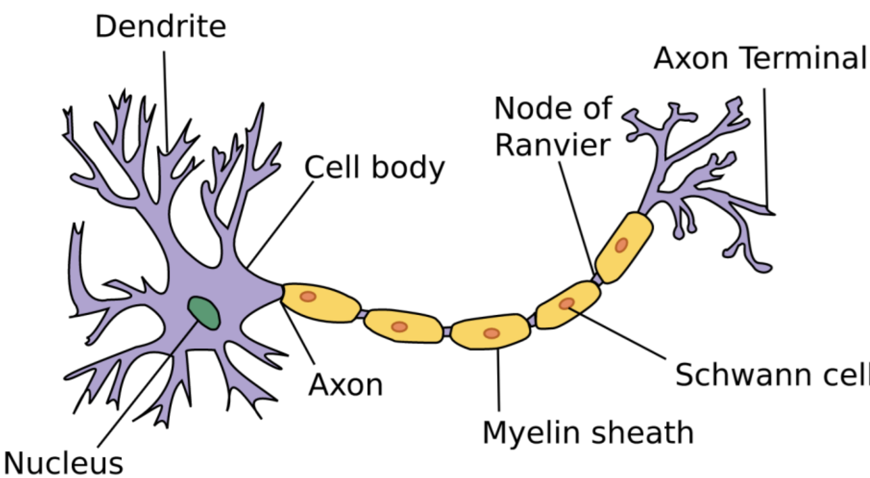
- **Receive inputs from other neurons (via synapses).**
- **When input exceeds a threshold, “fires.”**
    - Sends output along axon to other neurons.
- **Human brain: 10^{11} neurons**

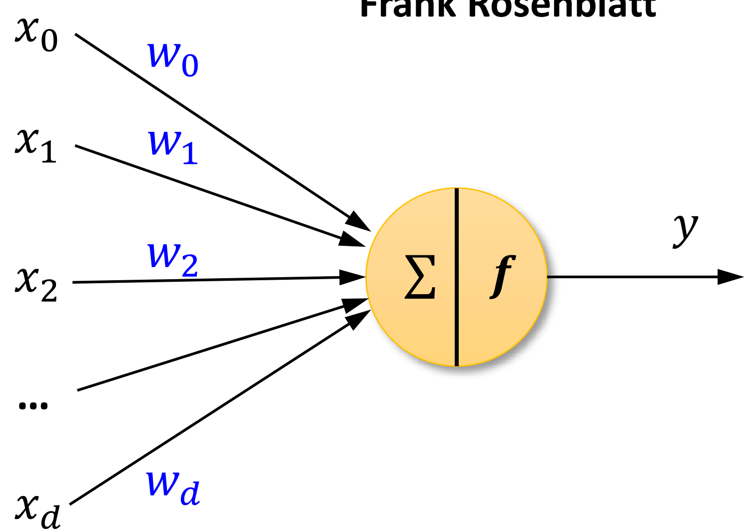


Perceptron: Artificial Neuron (1957)

- **A neuron is activated when the correlation between the input $\mathbf{x}$ and a pattern $\mathbf{w}$ exceeds a threshold.**
  - It is called an **artificial neuron**.
  - It mimics the function of human **neurons**.

Frank Rosenblatt





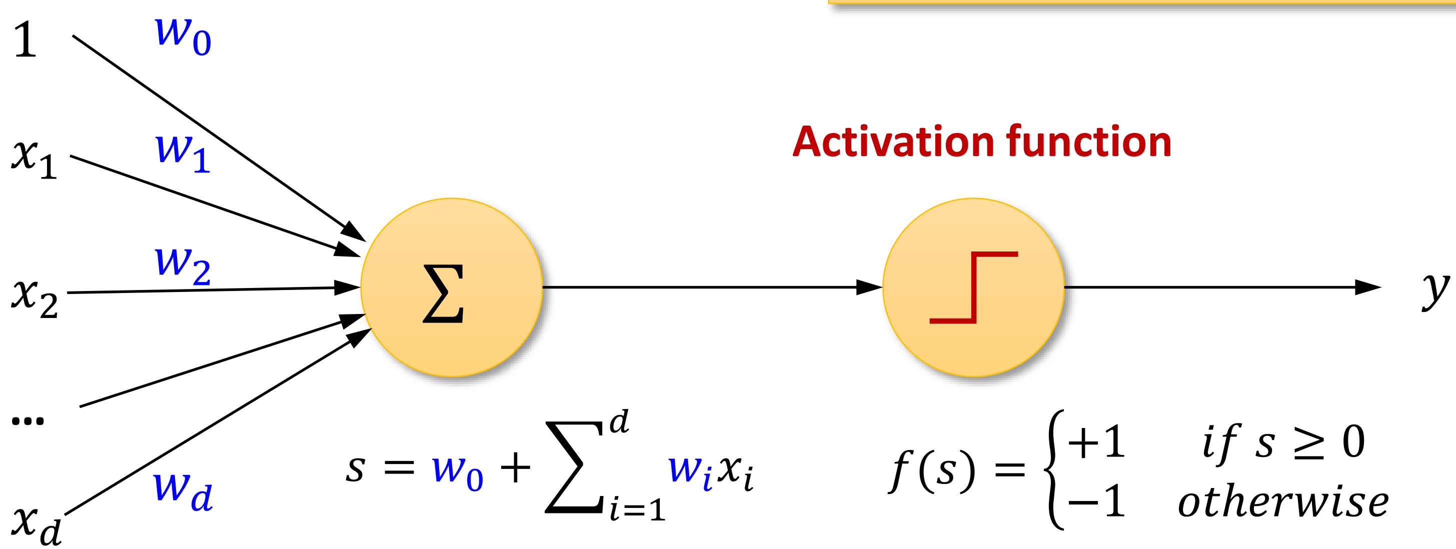
What is the Perceptron?

- **Linear combination of input x:**

$$s = w_0 + \sum_{i=1}^{d} w_i x_i$$

- **Nonlinear transformation of s:**

$$y = f(s), \text{where } \begin{cases} +1 & \text{if } s \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

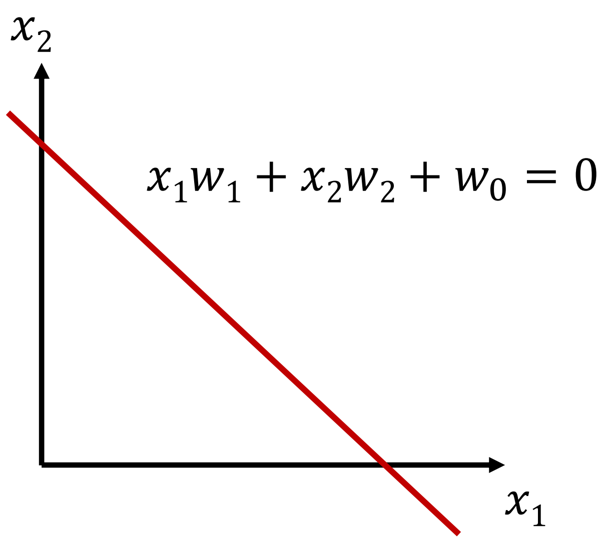


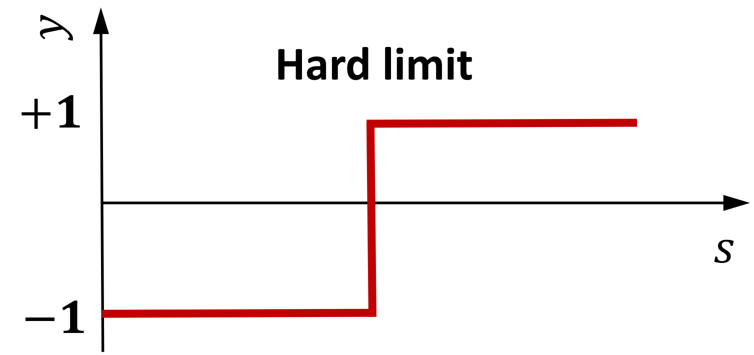
What is the Perceptron?

- **Linear combination of input x:**

$$s = \mathbf{w}^{\mathrm{T}}\mathbf{x} = \sum_{i=0}^{d} w_i x_i$$

- **Nonlinear transformation of s:**





$$f(s) = \begin{cases} +1 & if\ s \geq 0 \\ -1 & otherwise \end{cases}$$

Formulating a Learning Classifier

- **We regard the output y into $\{+1, -1\}$.**

- **Find $f(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x}$ that satisfies the condition.**
  - $\boldsymbol{y} = \mathbf{+1}$ if $\mathbf{w}^{\mathrm{T}}\mathbf{x} \geq 0$
  - $\boldsymbol{y} = \mathbf{-1}$ if $\mathbf{w}^{\mathrm{T}}\mathbf{x} < 0$

- **Prediction: $\hat{y} = \mathrm{sign}\big(f(\mathbf{x})\big) = \mathrm{sign}(\mathbf{w}^{\mathrm{T}}\mathbf{x})$**
  - Note: $\mathrm{sign}(\cdot)$ returns $+1$ or -1.

Formulating a Learning Classifier

- **Given training data** $\left\{\left(\mathbf{x}^{(i)}, y^{(i)}\right): 1 \leq i \leq n\right\}$

- **Hypothesis function:** $\hat{y} = \mathrm{sign}(\mathbf{w}^{\mathrm{T}}\mathbf{x})$

- **0-1 loss function: # of inputs such that** $y^{(i)} \neq \mathrm{sign}(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)})$

$$E(\mathbf{w}) = -\sum_{\left(\mathrm{x}^{(i)}, y^{(i)}\right) \in \mathcal{D}} y^{(i)}\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)}\mathbb{I}[mistake\ on\ \mathbf{x}^{(i)}]$$

$$\mathbb{I}\left[mistake\ on\ \mathbf{x}^{(i)}\right] = \begin{cases} 1 & if\ y^{(i)} \neq \mathrm{sign}\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(i)}\right) \\ 0 & otherwise \end{cases}$$

Formulating a Learning Classifier

- **Representing a simplified a 0-1 loss function**

$$E(\mathbf{w}) = -\sum_{(\mathrm{x}^{(i)}, y^{(i)}) \in \mathcal{D}} y^{(i)} \mathbf{w}^{\mathbf{T}} \mathbf{x}^{(i)} \mathbb{I}[mistake\ on\ \mathbf{x}^{(i)}]$$

$$\Downarrow$$

$$E(\mathbf{w}) = \sum_{(\mathbf{x}^{(k)}, y^{(k)}) \in \mathcal{S}} -y^{(k)} \left(\mathbf{w}^{\mathbf{T}} \mathbf{x}^{(k)}\right)$$

$\mathcal{S}$ is a set of **incorrect samples**.

Property of the 0-1 Loss Function

- **For any sample** $(\mathbf{x}^{(k)}, y^{(k)}) \in \mathcal{S}$, $-y^{(k)}\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(k)}\right) > 0$.

- $\boldsymbol{E}(\mathbf{w}) \geq 0$

$$E(\mathbf{w}) = \sum_{\left(\mathbf{x}^{(k)}, y^{(k)}\right) \in \mathcal{S}} -y^{(k)}\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(k)}\right)$$

- **If w is optimal, then** $\boldsymbol{E}(\mathbf{w}) = 0$.
  - $E(\mathbf{w})$ tends to increase with the number of **incorrect samples**.

Computing the Derivative of w

- **How to compute the derivative of w_j**

$$\frac{\partial E(\mathbf{w})}{\partial w_j} = \sum_{(\mathbf{x}^{(k)}, y^{(k)}) \in \mathcal{S}} \frac{\partial}{\partial w_j}\left(-y^{(k)}\left(w_0 x_0^{(k)} + \cdots + w_j x_j^{(k)} + \cdots + w_d x_d^{(k)}\right)\right)$$

Other values are regarded as constants.

$$\frac{\partial E(\mathbf{w})}{\partial w_j} = \sum_{(\mathbf{x}^{(k)}, y^{(k)}) \in \mathcal{S}} -y^{(k)} x_j^{(k)}, \qquad \forall j = 0, 1, \ldots, d$$

Perceptron Learning Algorithm (PLA)

➢ **Batch version**

Initialize a random weight $\mathbf{w}$.
Repeat
$\mathcal{S} \leftarrow \emptyset$

Step 1: finding a set of incorrect samples

for $i = 1$ to n
$\hat{y} \leftarrow \mathrm{sign}\left(\mathbf{w}^{\mathbf{T}}\mathbf{x}^{(i)}\right)$
$if\ \left(\hat{y} \neq y^{(i)}\right)\ \mathcal{S} \leftarrow \mathcal{S} \cup (\mathbf{x}^{(i)}, y^{(i)})$

Step 2: updating the weight w

if $(\mathcal{S} \neq \emptyset)$
$\Delta\mathbf{w} = -\sum_{(\mathbf{x}^{(k)}, y^{(k)}) \in \mathcal{S}} y^{(k)}\mathbf{x}^{(k)}$
$\mathbf{w} \leftarrow \mathbf{w} - \eta\Delta\mathbf{w}$

Until $(\mathcal{S} = \emptyset)$

Perceptron Learning Algorithm (PLA)

➢ Stochastic version

Initialize a random weight $\mathbf{w}$.

Repeat

quit $\leftarrow$ true

for $i \leftarrow 1$ to n

Step 1: finding an incorrect sample

$\hat{y} \leftarrow \text{sign}\left(\mathbf{w}^{\mathbf{T}}\mathbf{x}^{(i)}\right)$

if $\left(\hat{y} \neq y^{(i)}\right)$

quit $\leftarrow$ false

$\Delta\mathbf{w} = -y^{(i)}\mathbf{x}^{(i)}$

$\mathbf{w} \leftarrow \mathbf{w} - \eta\Delta\mathbf{w}$

Step 2: updating the weight w

Until (quit = true)

Perceptron Learning Algorithm (PLA)

- **Start with a random weight vector.**
- **Given an input, predict its class.**
  - For the mistake on a **positive sample**,

$$\mathbf{w}_{new} = \mathbf{w} - \eta \nabla E\left(\mathbf{w}; (\mathbf{x}^{(i)}, y^{(i)})\right) = \mathbf{w} + \eta \mathbf{x}^{(i)}$$

  - For the mistake on a **negative sample**,

$$\mathbf{w}_{new} = \mathbf{w} - \eta \nabla E\left(\mathbf{w}; (\mathbf{x}^{(i)}, y^{(i)})\right) = \mathbf{w} - \eta \mathbf{x}^{(i)}$$

- **Repeat it until there is no mistake.**

Perceptron Learning Algorithm (PLA)

- **Start with a random weight vector.**
- **Given an input, predict its class.**
  - For the mistake on a **positive sample**,

$$\mathbf{w}_{new} = \mathbf{w} - \eta \nabla E\left(\mathbf{w}; (\mathbf{x}^{(i)}, y^{(i)})\right) = \mathbf{w} + 1\mathbf{x}^{(i)}$$

When $\eta = 1$

  - For the mistake on a **negative sample**,

$$\mathbf{w}_{new} = \mathbf{w} - \eta \nabla E\left(\mathbf{w}; (\mathbf{x}^{(i)}, y^{(i)})\right) = \mathbf{w} - 1\mathbf{x}^{(i)}$$

- **Repeat it until there is no mistake.**

Recap: Learning a Linear Classifier

- Execute the algorithm until no mistakes are encountered.

Randomly choose an initial solution $\mathbf{w}^{\mathbf{0}}$.

For $t = 0, 1, \dots$

Find a mistake sample $(\mathbf{x}^{(i)}, y^{(i)})$ of $\mathbf{w}^{t}$

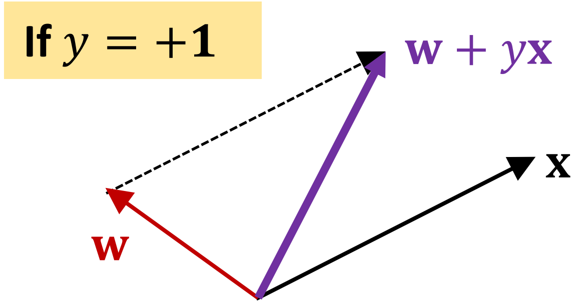
$$\mathrm{sign}\left(\mathbf{w}^{\mathbf{T}}\mathbf{x}^{(i)}\right) \neq y^{(i)}$$

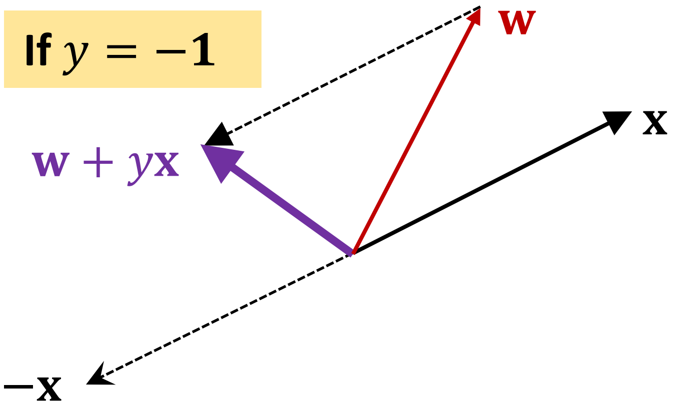
Correct the mistake by

$$\mathbf{w}^{t+1} = \mathbf{w}^{t} + y^{(i)}\mathbf{x}^{(i)}$$

Until no more mistake is found

Return last $\mathbf{w}^{t}$ as the learned model.

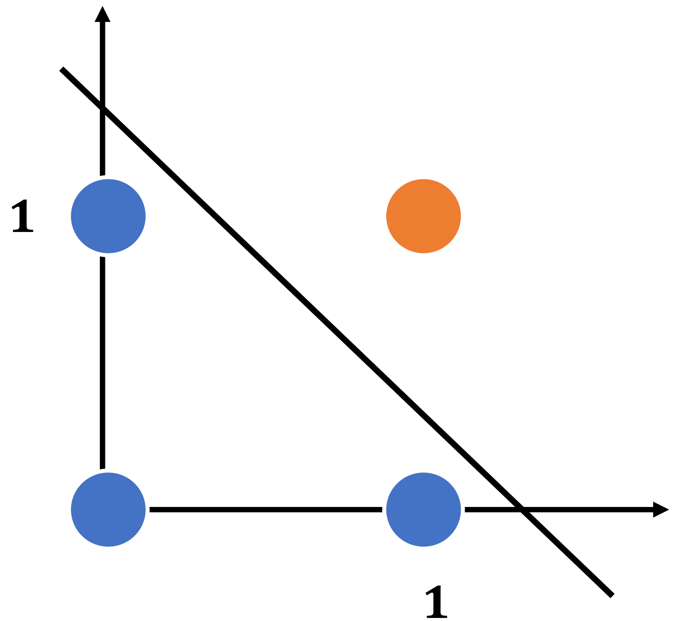




AND Operation

➢ **How to represent the AND operation**

$$w_1 = 1.0, w_2 = 1.0, w_0 = -1.5$$

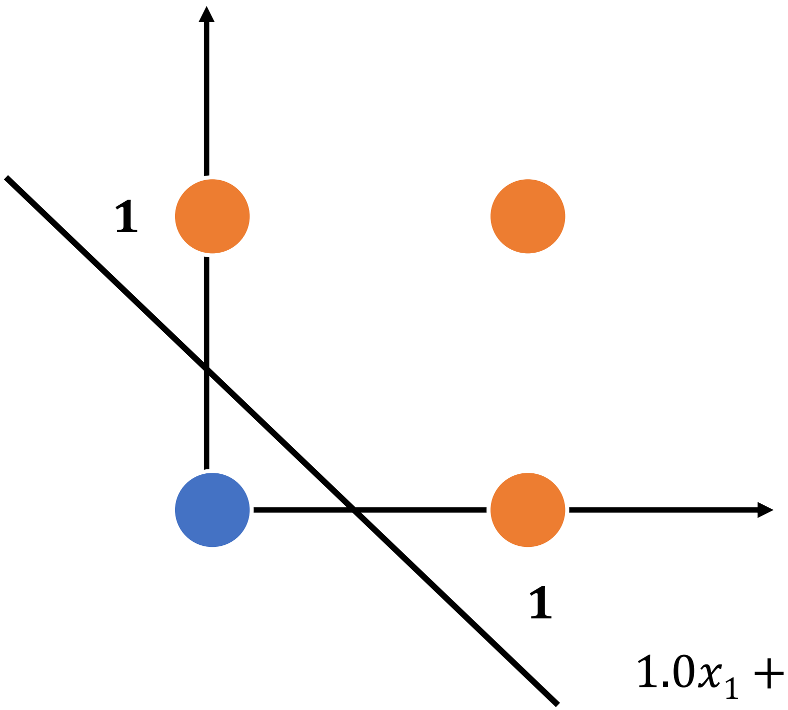


x_1	x_2	Σ	y
0	0	-1.5	0
0	1	-0.5	0
1	0	-0.5	0
1	1	$+0.5$	1

$$1.0x_1 + 1.0x_2 - 1.5 = 0$$

OR Operation

- How to represent the OR operation



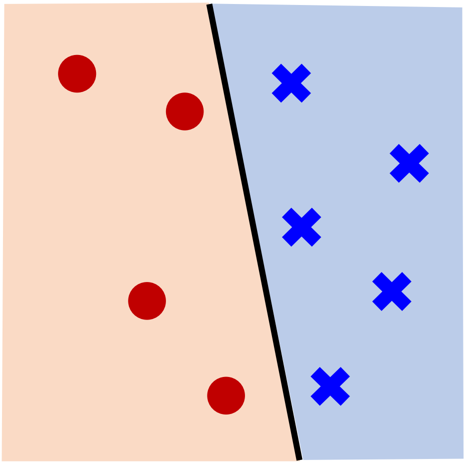
$1.0x_1 + 1.0x_2 - 0.5 = 0$

$w_1 = 1.0, w_2 = 1.0, w_0 = -0.5$

x_1	x_2	$\sum$	y
0	0	-0.5	0
0	1	0.5	1
1	0	0.5	1
1	1	1.5	1

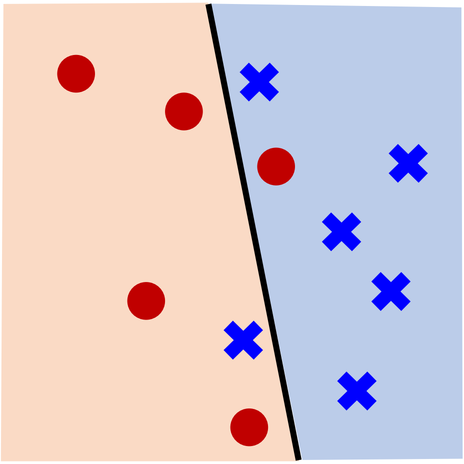
Linear Separability

- If PLA halts (i.e., no more mistakes),
  - (necessary condition) $\boldsymbol{D}$ allows some $\mathbf{w}$ to make no mistake.
- Call such D **linearly separable.**



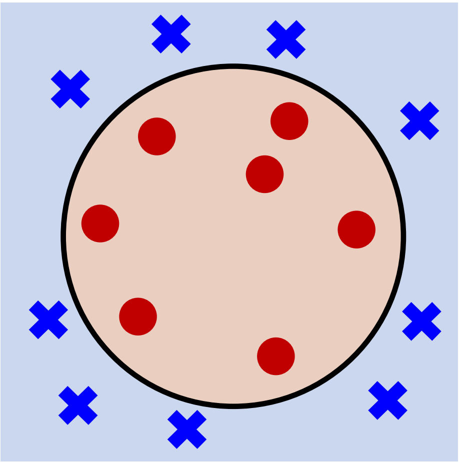
Linear separable

Good!



Linear non-separable

Need a linear model that allows some errors.

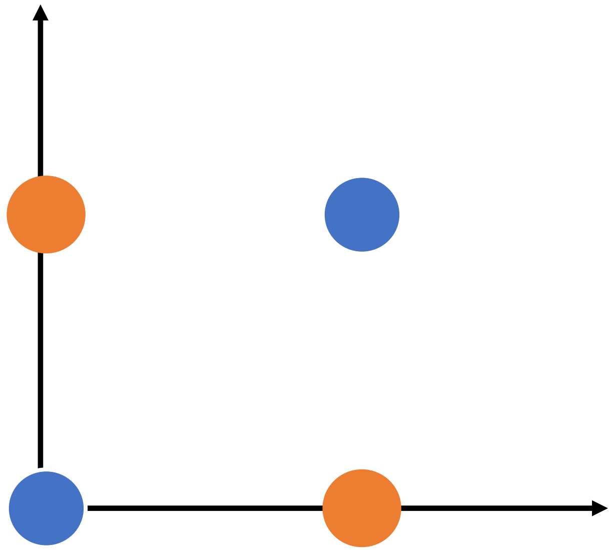


Linear non-separable

Need a non-linear model.

XOR Operation (1969)

- How to train a linear decision boundary?



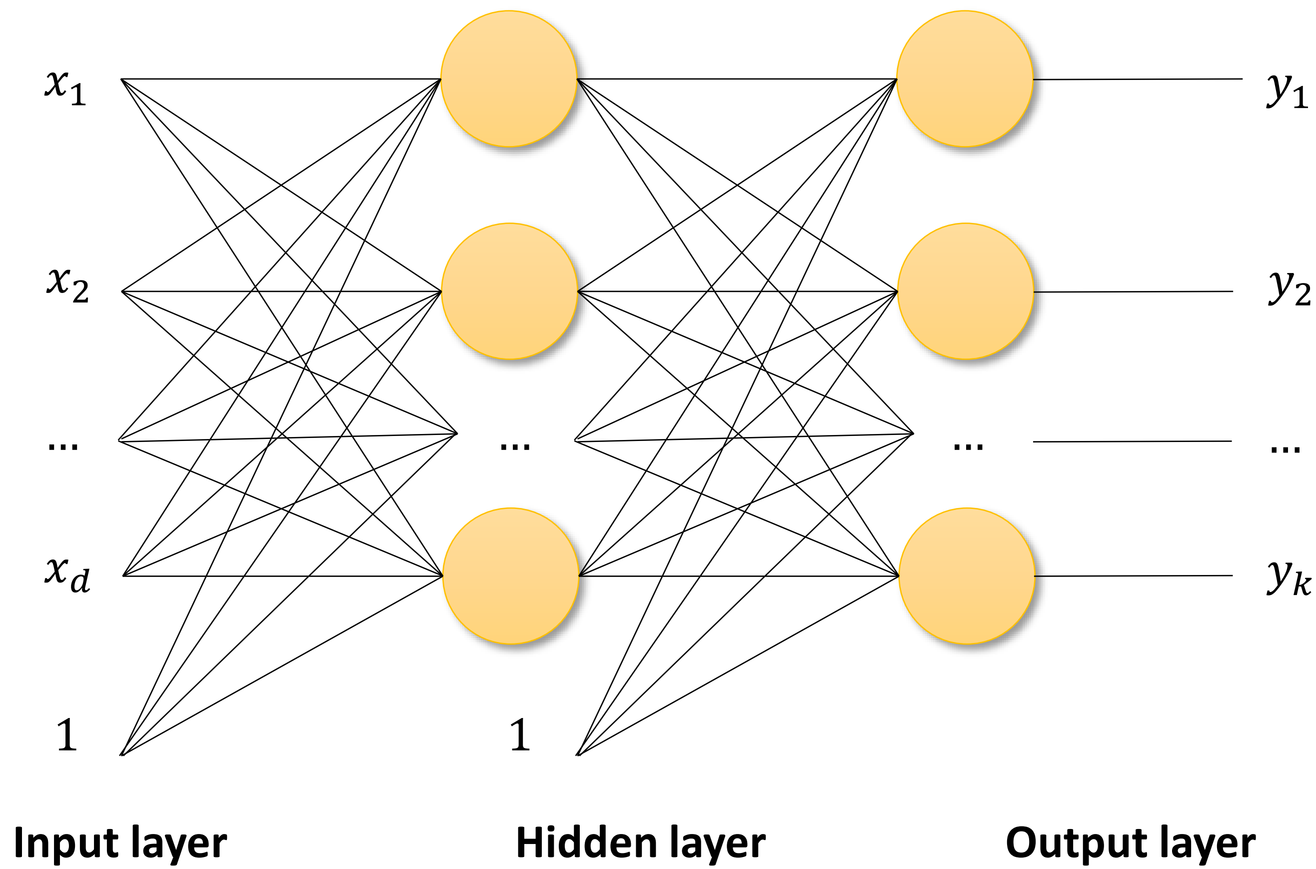
"No one on earth had found a viable way to train" by Marvin Minsky

- Guaranteed to converge if linearly separable.
- Otherwise, many simple functions are NOT learnable.

Multilayer Perceptron (MLP)

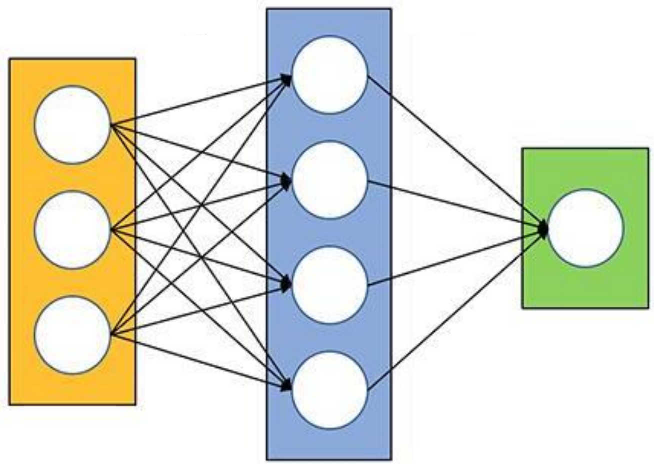
Multilayer Perceptron (MLP)

- It is a neural network of **multiple** artificial neurons.



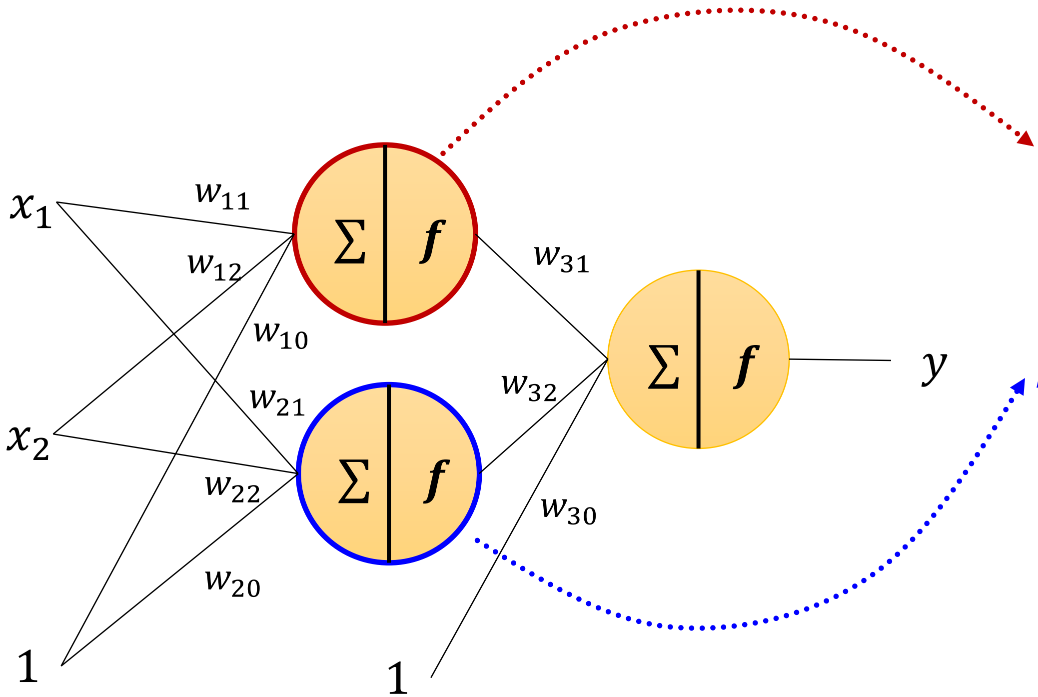
Multilayer Perceptron (MLP)

- **Structure**
  - **Input layer**
    - Simply pass the input values to the next layer.
    - # of nodes = # of inputs
  - **Hidden layer**
    - There can be several hidden layers.
    - # of nodes should be given.
  - **Output layer**
    - # of nodes = # of outputs



- **MLP is also called the feed-forward neural network.**

Multilayer Perceptron (MLP)



$$h_1 = \frac{1}{1 + \exp\big(-(w_{11}x_1 + w_{12}x_2 + w_{10})\big)}$$

$$h_2 = \frac{1}{1 + \exp\big(-(w_{21}x_1 + w_{22}x_2 + w_{20})\big)}$$

$$y = \frac{1}{1 + \exp\big(-(w_{31}h_1 + w_{32}h_2 + w_{30})\big)}$$

$$y = \frac{1}{1 + \exp\left(-\left(w_{31}\underbrace{\left(\frac{1}{1 + \exp(-(w_{11}x_1 + w_{12}x_2 + w_{10}))}\right)}_{h_1} + w_{32}\underbrace{\left(\frac{1}{1 + \exp(-(w_{21}x_1 + w_{22}x_2 + w_{20}))}\right)}_{h_2} + w_{30}\right)\right)}$$

How to Solve XOR Operation?

$w_{11} = 1.0, w_{12} = 1.0, w_{10} = -1.5$ $w_{21} = 1.0, w_{22} = 1.0, w_{20} = -0.5$ $w_{31} = -1.0, w_{32} = 1.0, w_{30} = -0.5$

$h_1 = \text{sigmoid}(x_1 + x_2 - 1.5)$

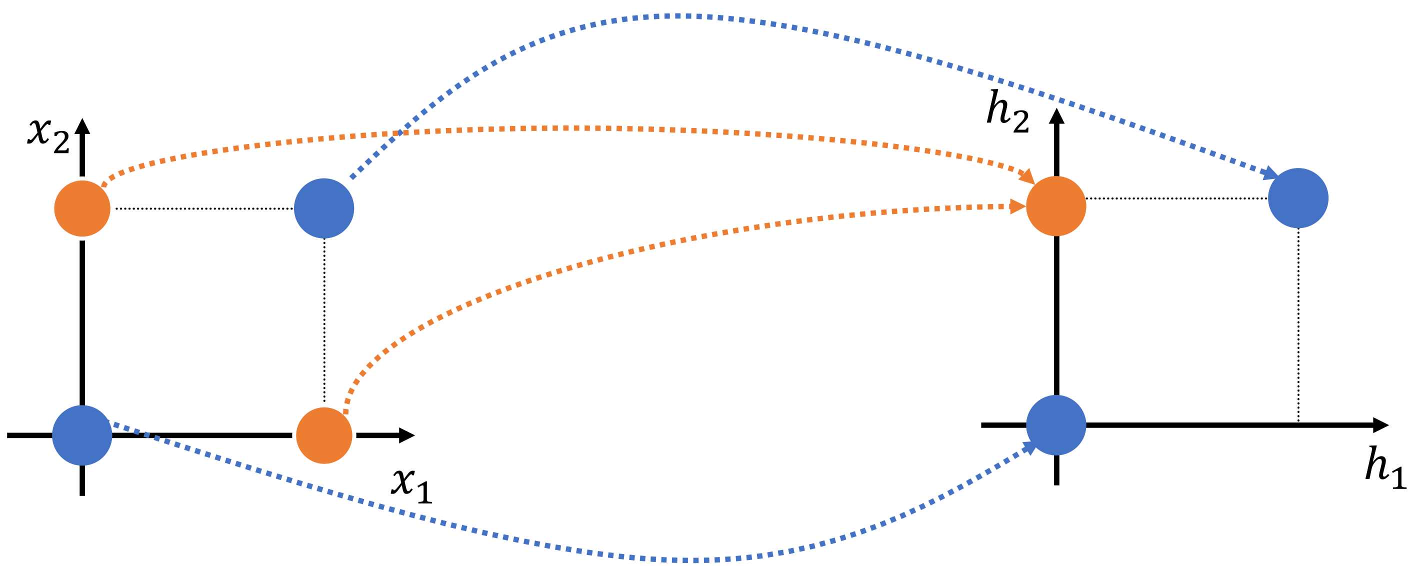
x_1	x_2	$\sum$	h_1
0	0	-1.5	0
0	1	-0.5	0
1	0	-0.5	0
1	1	+0.5	1

$h_2 = \text{sigmoid}(x_1 + x_2 - 0.5)$

x_1	x_2	$\sum$	h_2
0	0	-0.5	0
0	1	0.5	1
1	0	0.5	1
1	1	1.5	1

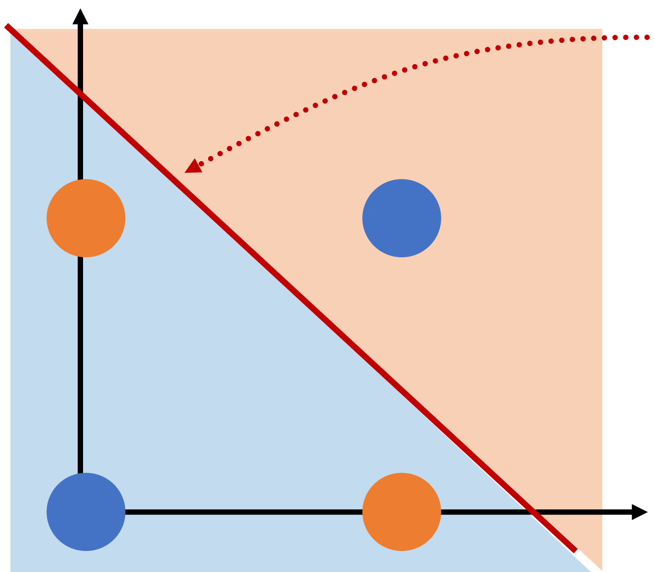
$\hat{y} = \text{sigmoid}(-x_1 + x_2 - 0.5)$

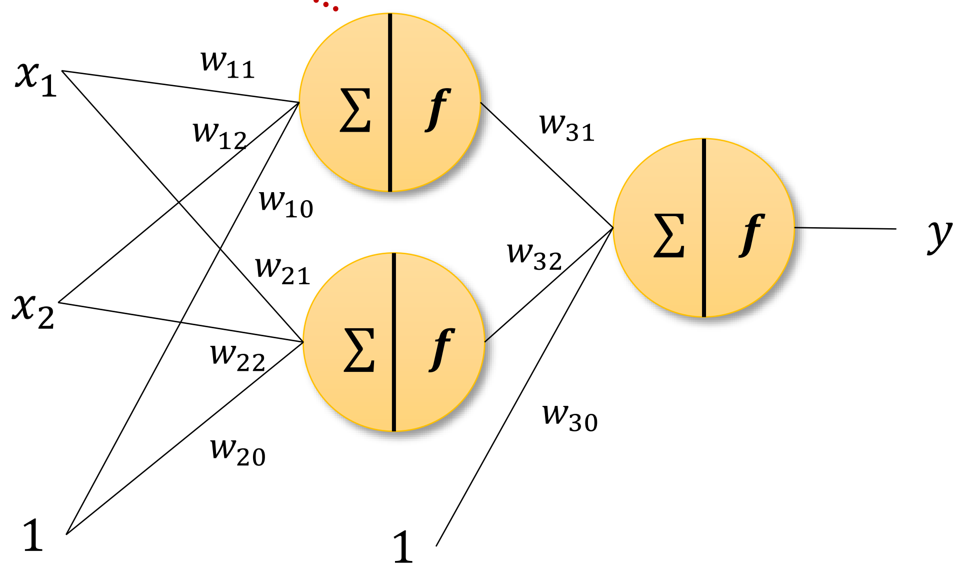
h_1	h_2	$\sum$	$\hat{y}$
0	0	-0.5	0
0	1	0.5	1
0	1	0.5	1
1	1	-0.5	0



How to Solve XOR Operation?

- A single neuron divides a region into **two subregions**.

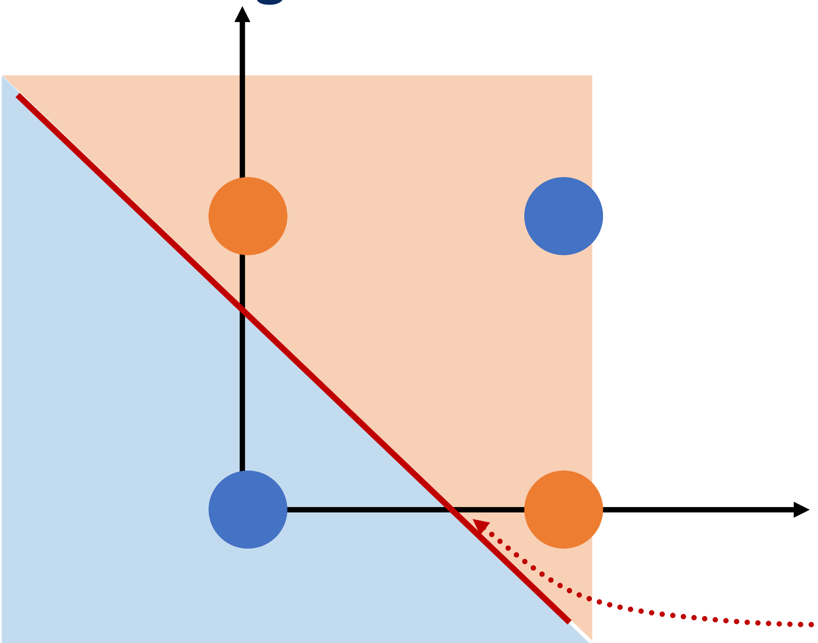


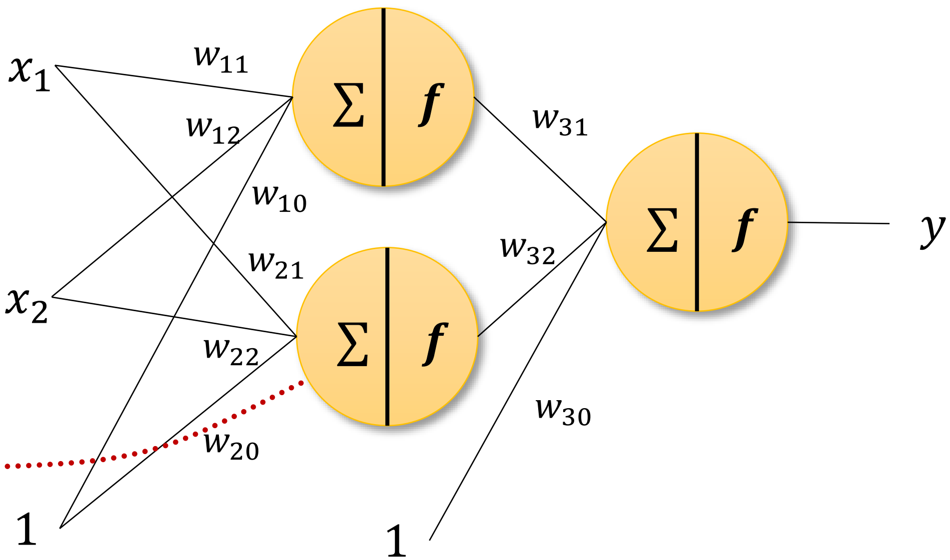


x_1	x_2	Σ	h_1
0	0	-1.5	0
0	1	-0.5	0
1	0	-0.5	0
1	1	+0.5	1

How to Solve XOR Operation?

- A single neuron divides a region into **two subregions**.

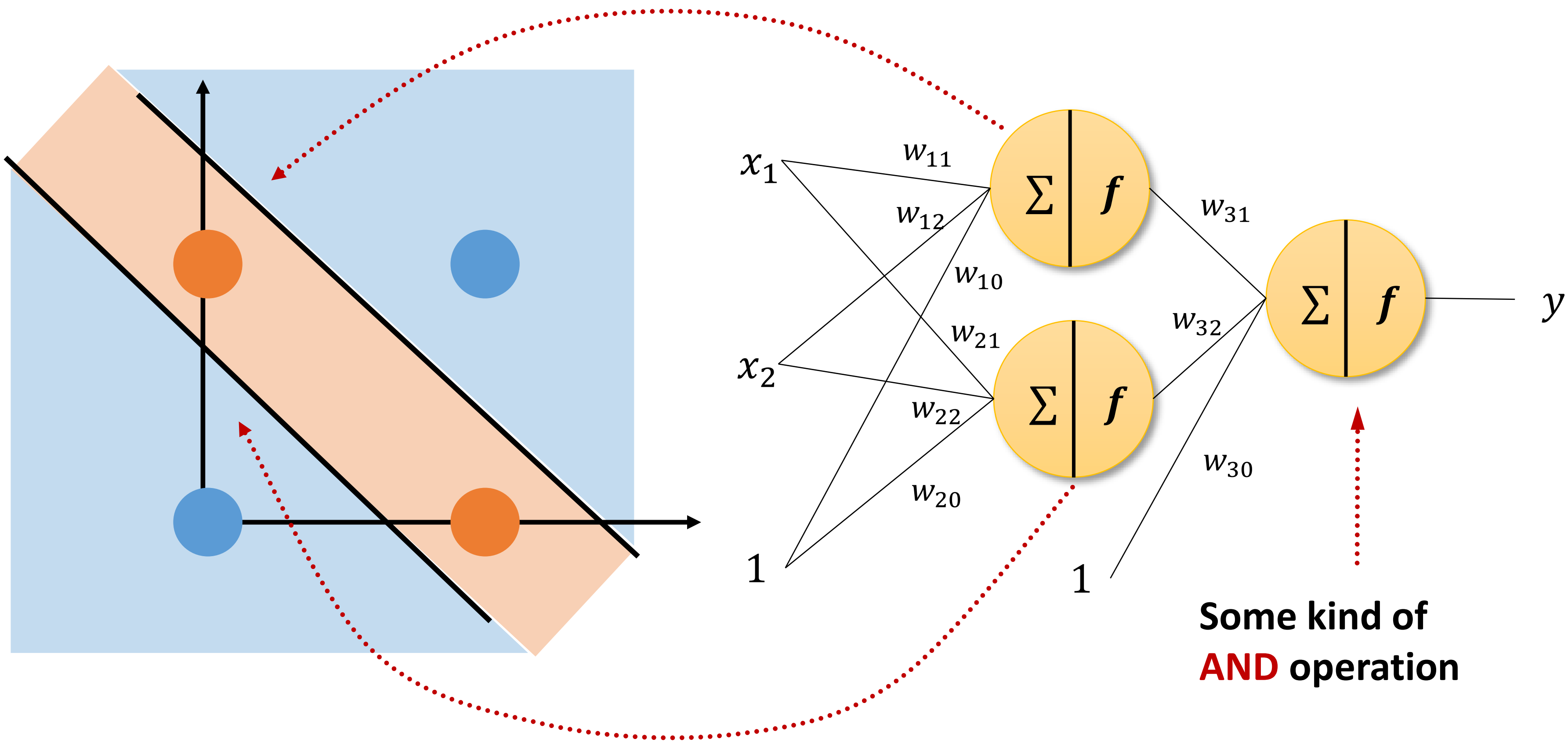




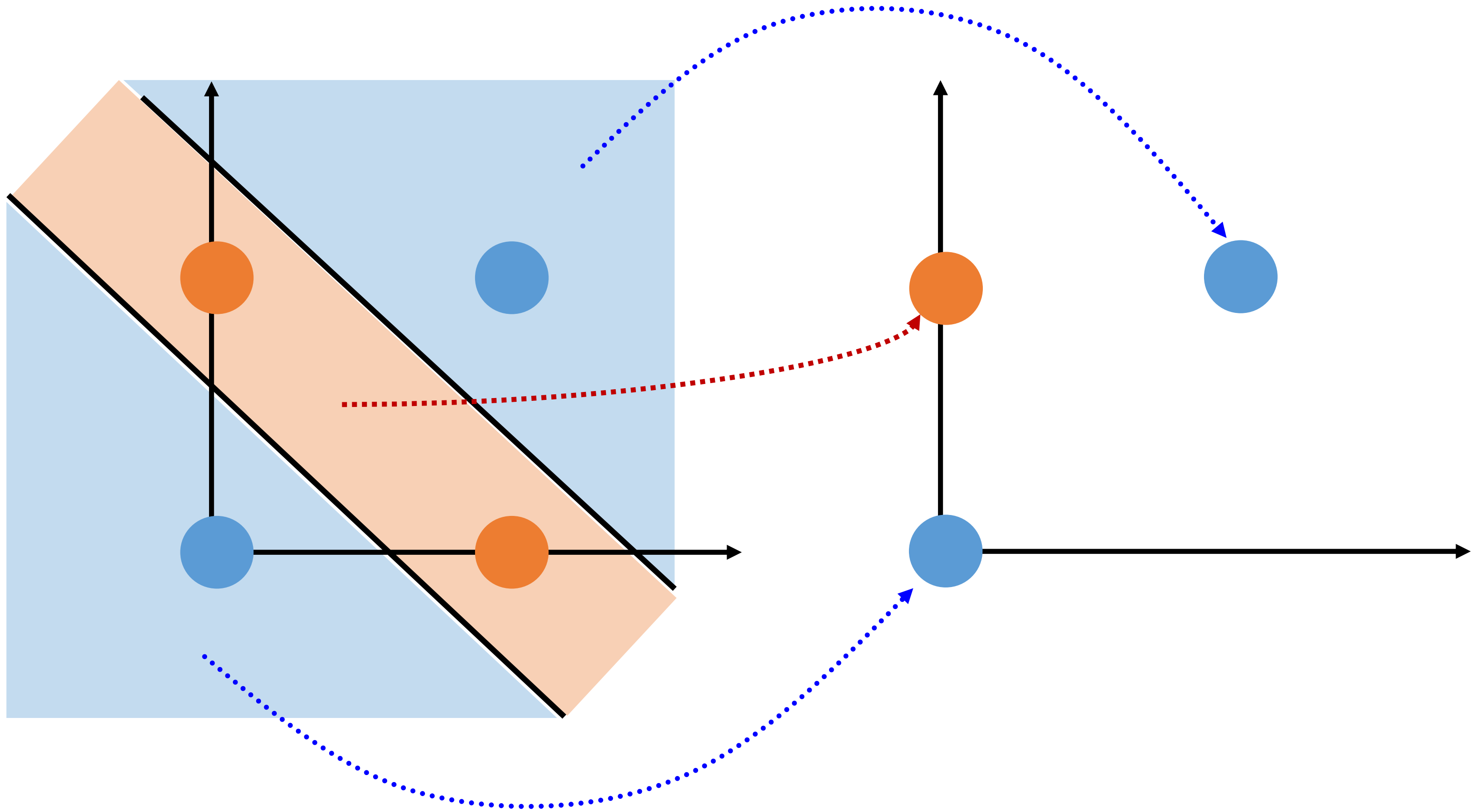
x_1	x_2	$\sum$	h_2
0	0	-1.5	0
0	1	0.5	1
1	0	0.5	1
1	1	1.5	1

How to Solve XOR Operation?

➢ **Combining two neurons with AND operation**



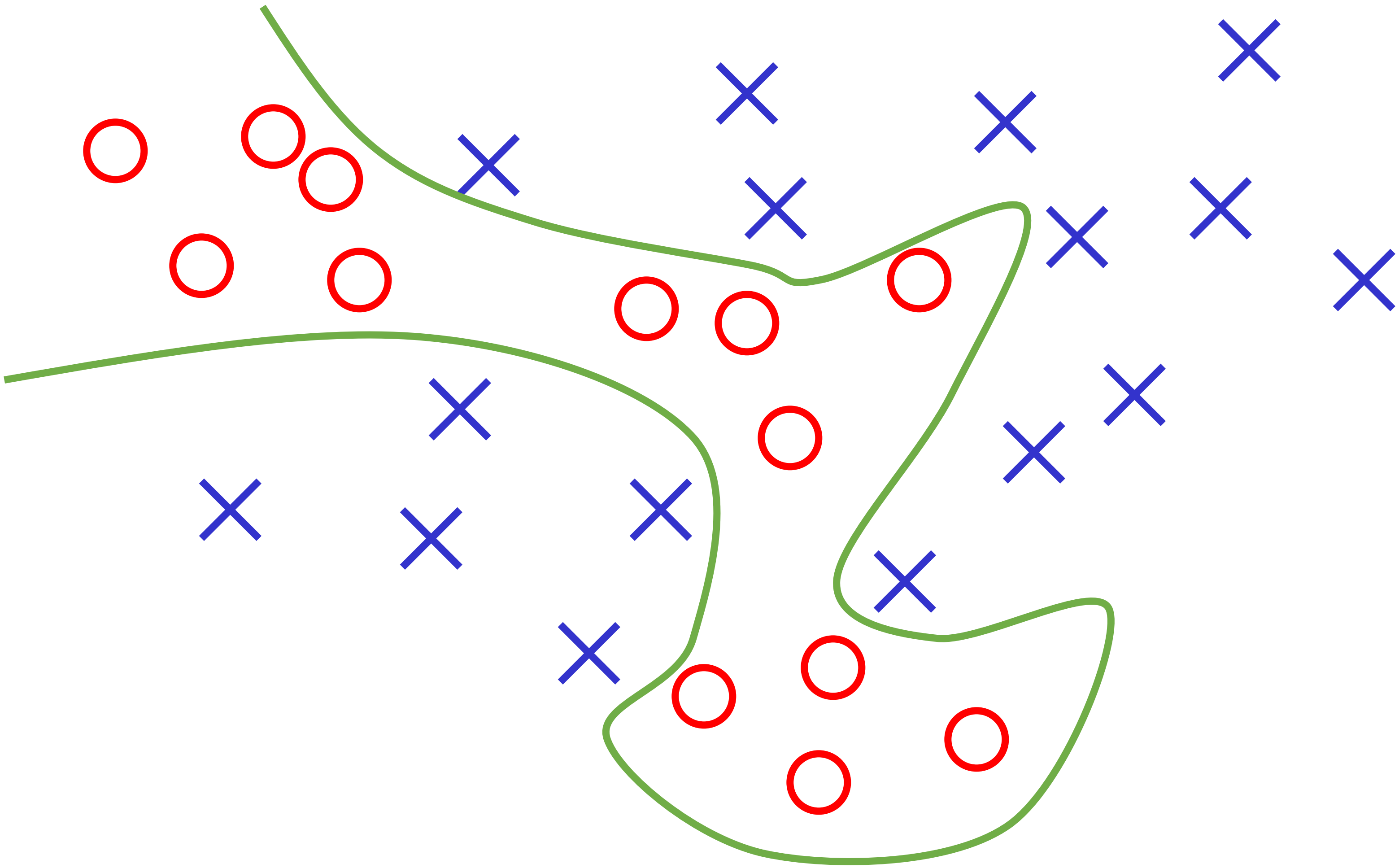
How to Solve XOR Operation?



Mapping the regions in the original space to the points in another space

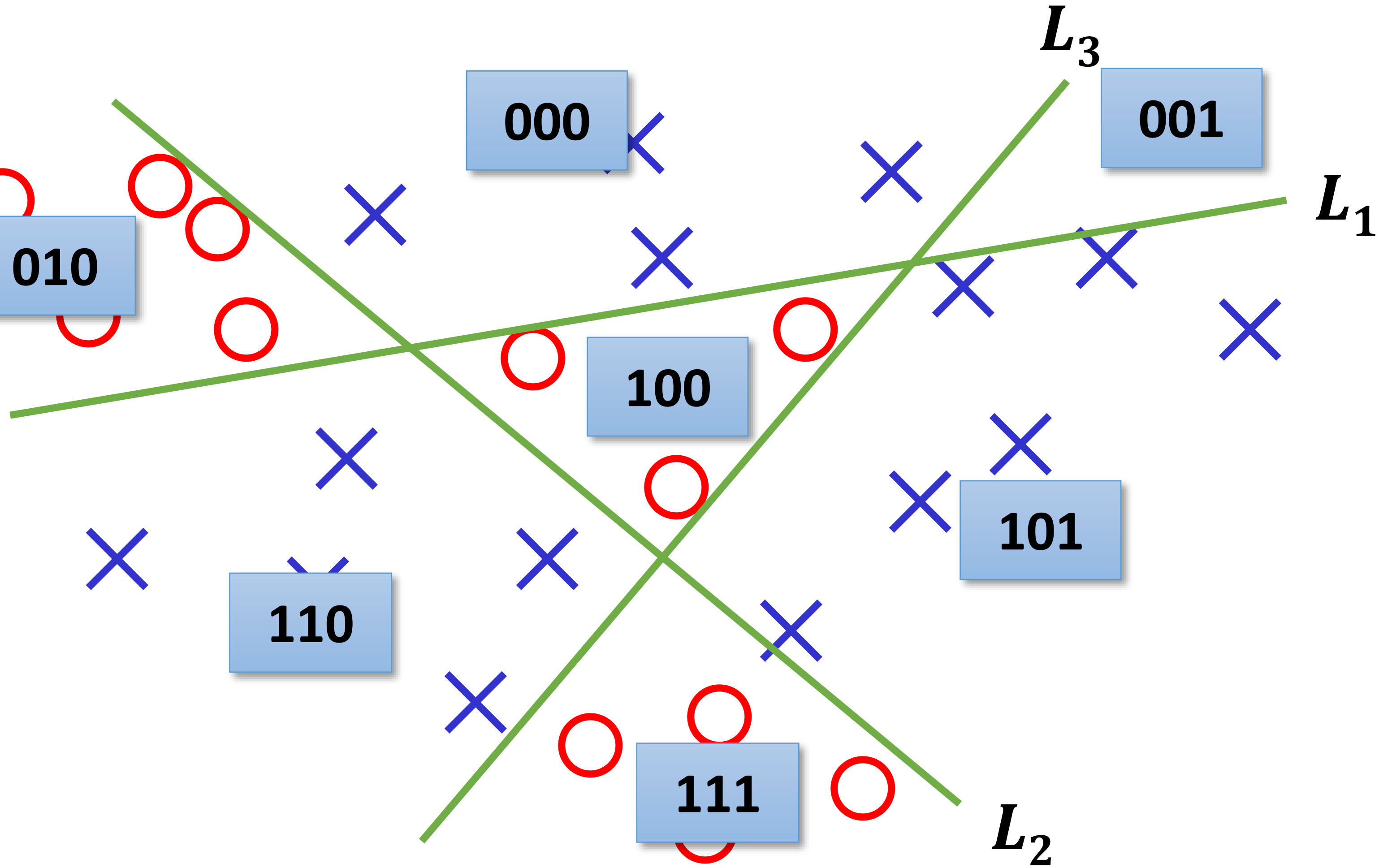
Non-linear Classifier using MLP

- How to classify two classes?



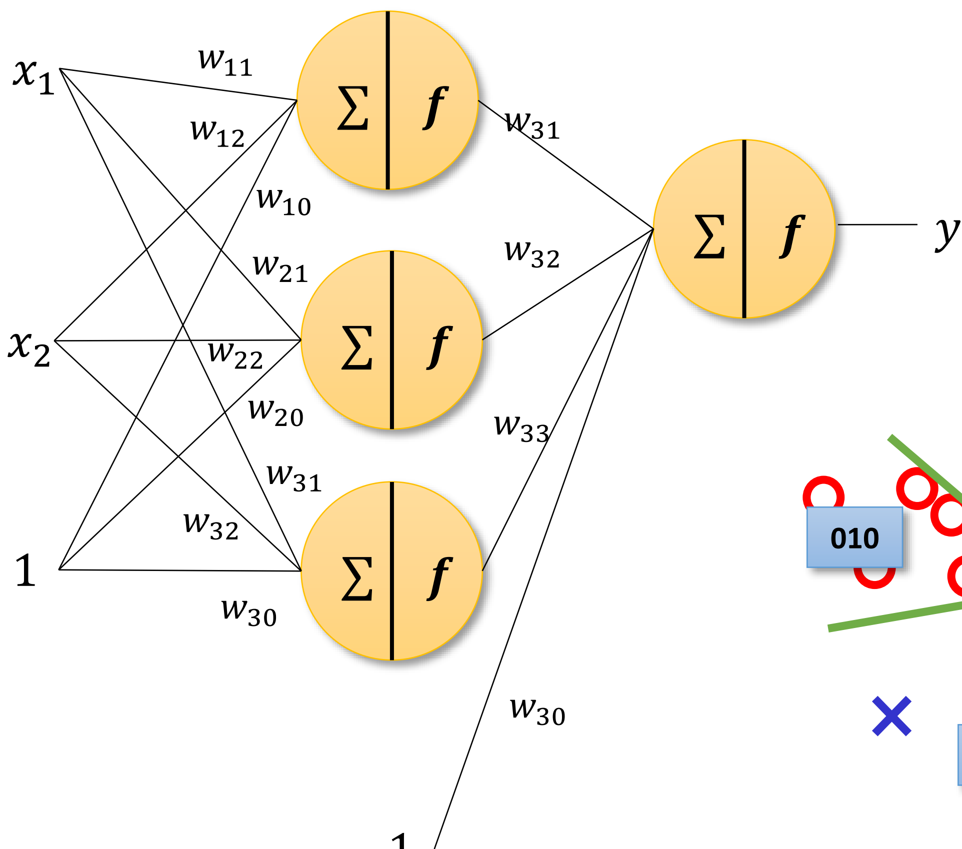
Non-linear Classifier using MLP

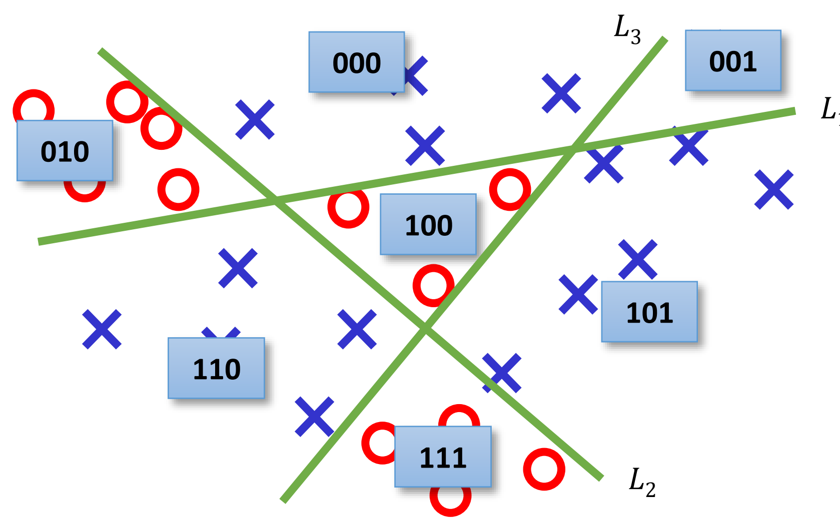
- Hyperspace partitioning



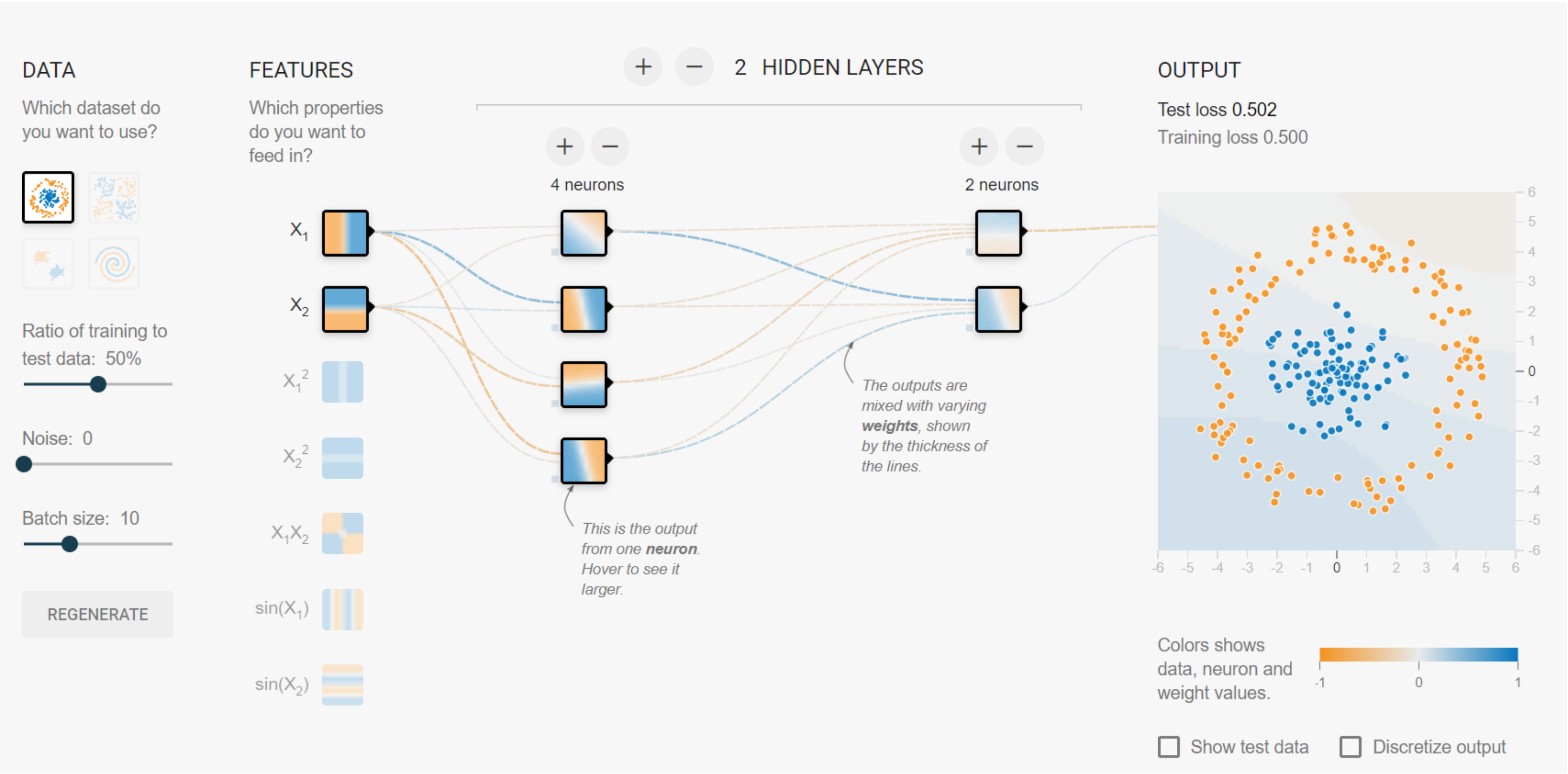
Non-linear Classifier using MLP

- Use three nodes in the hidden layer.





Example: Visualization of MLP



https://playground.tensorflow.org/

Deep Neural Networks (DNNs)

Recap: Machine Learning 1-2-3

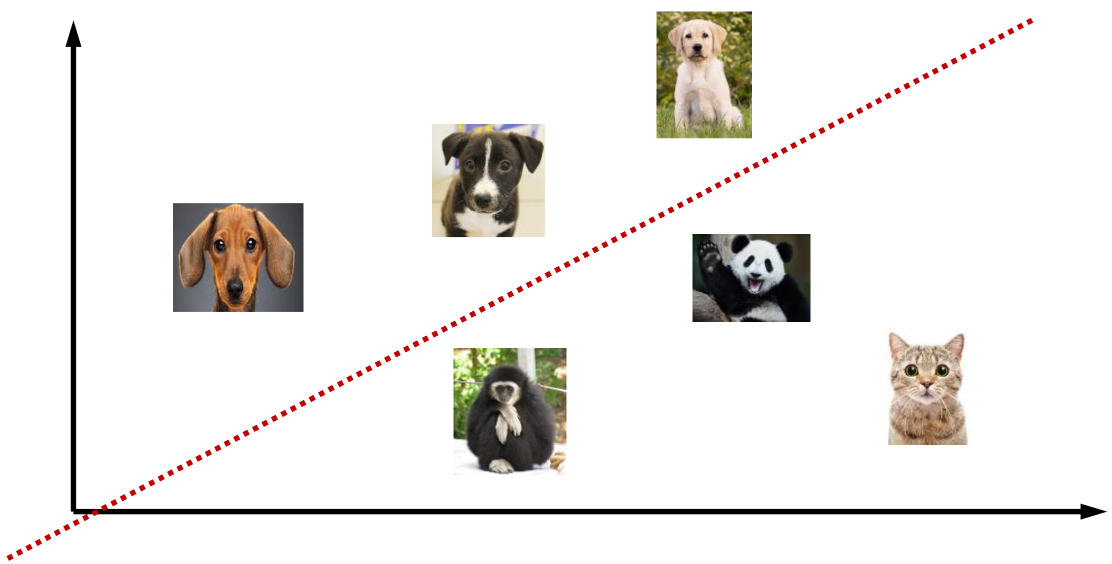
- Collect data and extract features.
- Choose the hypothesis function $\mathcal{H}$ and the loss function $\mathcal{L}$.
- Find an optimal parameter that minimizes the empirical loss.

- Q: How to represent feature vectors for images?
- A: It is difficult to design the vector.

Simplest Case: Linear Classifiers

- Consider the original perceptron model on raw data.

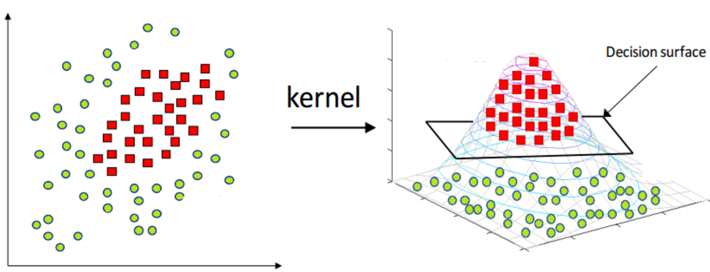
- To classify the image of “dog” (+1) or “not dog” (-1), find the line that separates the +1’s from the -1’s.

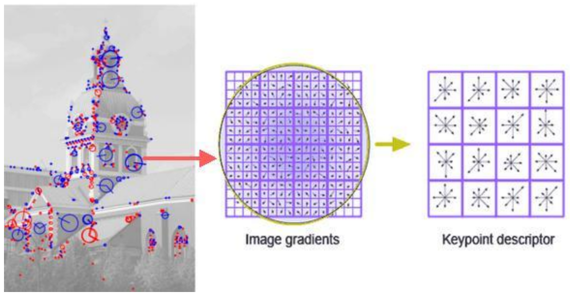


Non-Linear Features, Linear Classifiers

- **Most problems need non-linear features.**
  - Image classification, machine translation, speech recognition, etc.

- **How to represent non-linear features?**
  - Non-linear kernels in SVM
  - Explicit design of features, e.g., SIFT, HOG



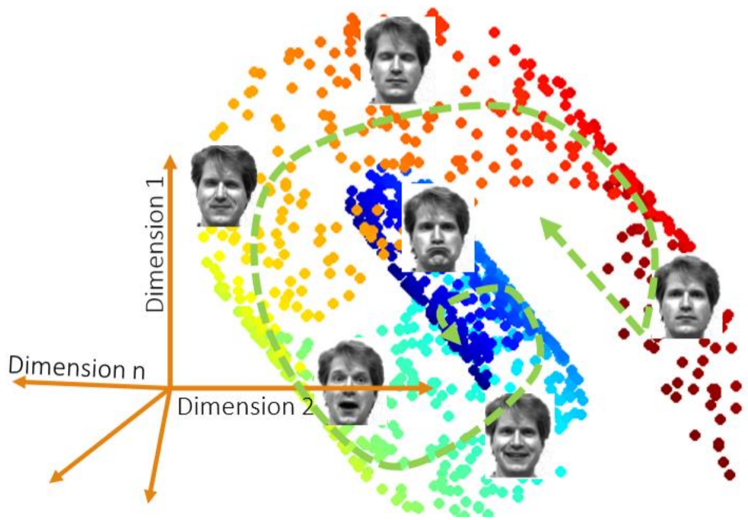


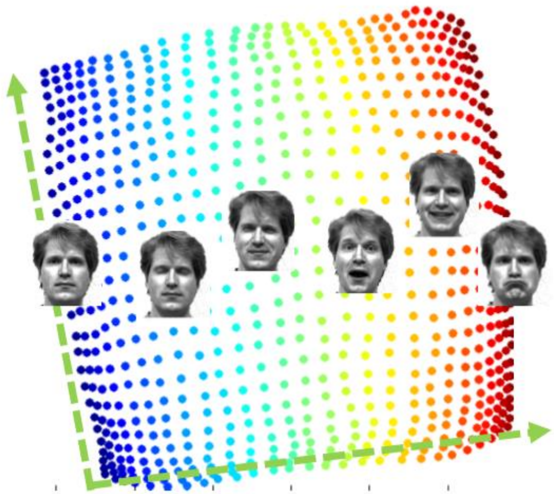
- **Then, linear machines are good enough.**

How to Represent Good Features?

- **Raw data live high dimensionalities. However, data lie in low-dimensional manifolds.**

- **Can we discover a manifold for our data?**
  - Hypothesis: **Semantically similar things** lie **closer** together than **semantically dissimilar things**.

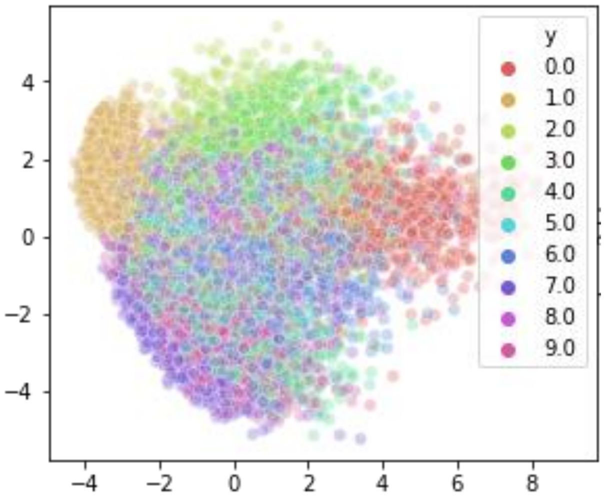




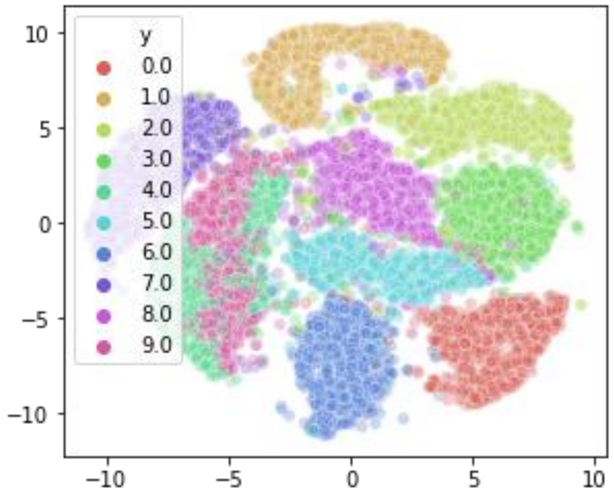
Example: Manifolds of Digits

- There are good features (manifolds).

- Each image has 784 dimensions on the MNIST dataset.
  - 28 pixels x 28 pixels = 784 dimensions



Entangled features
(PCA)



Disentangled features
(t-SNE)

Why Learn Features?

- **Manually designed features**
  - Often, take **a lot of time to** implement
  - Often, take a lot of time to validate
  - Often, they are **incomplete**, as one cannot know if they are optimal for the task

- **Learned features**
  - If data is enough, easy to learn
  - Compact and specific **to the task**

- **Time spent for designing features vs. Time spent on designing architectures**

Shallow Learning vs. Deep Learning

Artificial intelligence

Hard-coded knowledge & the rule-driven approach is used.

Machine learning

Extract the patterns & co-relation between features.
It is difficult to know whether all features are effective and necessary.

Representation learning

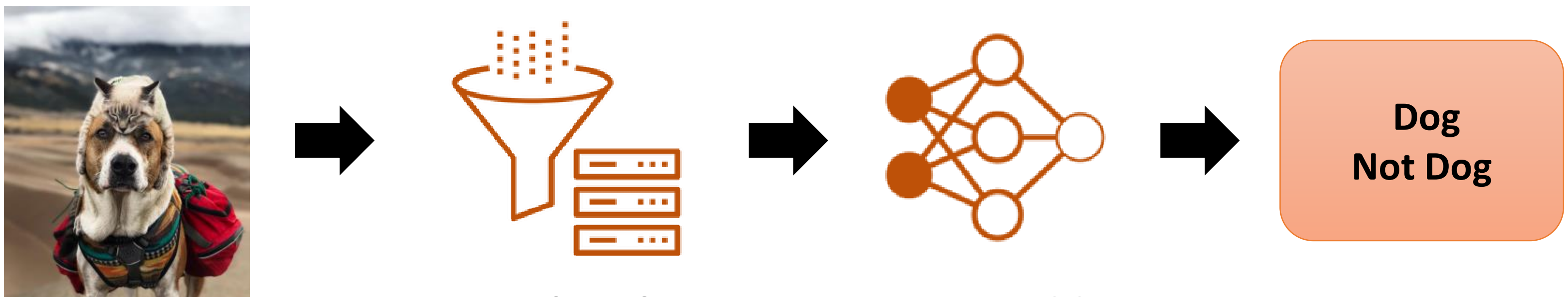
Learns the representation instead of the features.
There are many factors of variations for a given representation.

Deep learning

Complex representations are expressed in terms of simple representations.

Shallow Learning vs. Deep Learning

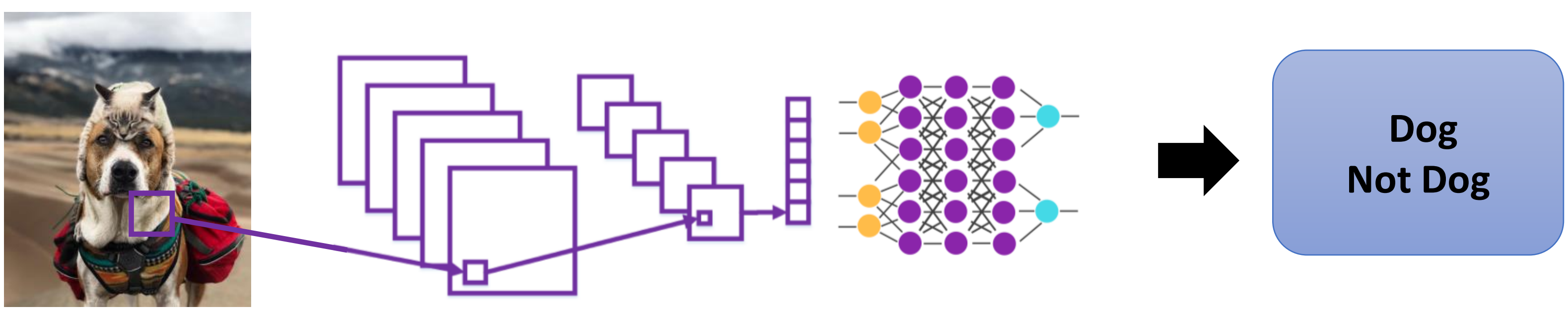
- Traditional machine learning



Hand-crafted feature extractor

Trainable classifier

- End-to-end learning → features are also **learned from data!**



Trainable feature extractor

Trainable classifier

Deep Neural Networks (DNNs)

- **It consists of a pipeline of successive, differentiable modules (transformations).**
  - Each module’s output is used as the input for the next module.

- **Learns hierarchical representation from data.**
- **Each sequential module produces a higher abstraction feature.**

→ **Initial modules** → **Middle modules** → **Last modules** → “Dog”

Deep Neural Networks (DNNs)

- A family of **parametric**, **non-linear**, and **hierarchical** representation learning functions
  - They are optimized with **stochastic gradient descent** to **encode domain knowledge**, i.e., domain invariances and stationarity.

$$\boldsymbol{a_L}\left(\boldsymbol{x};\boldsymbol{\theta}_{\mathbf{1},\ldots,\boldsymbol{L}}\right) = \boldsymbol{h_L}(\boldsymbol{h_{L-1}}(\ldots\boldsymbol{h_1}(\boldsymbol{x},\boldsymbol{\theta_1}),\boldsymbol{\theta_{L-1}}),\boldsymbol{\theta_L})$$

- Given training corpus $\{X, Y\}$, find optimal parameters.

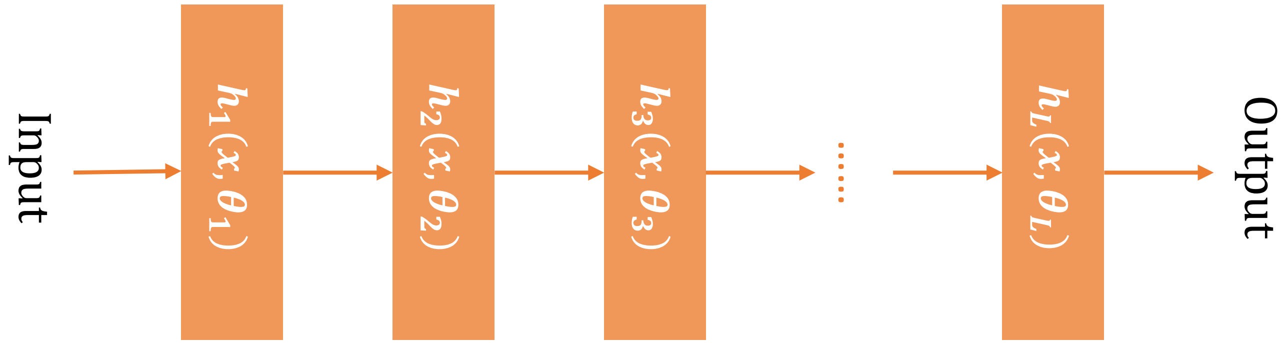
$$\boldsymbol{\theta}^* \leftarrow \operatorname*{\mathbf{argmin}}_{\boldsymbol{\theta}} \sum_{(\boldsymbol{x},\boldsymbol{y})\subseteq(\boldsymbol{X},\boldsymbol{Y})} \mathcal{L}\left(\boldsymbol{y},\boldsymbol{a_L}\left(\boldsymbol{x};\boldsymbol{\theta}_{\mathbf{1},\ldots,\boldsymbol{L}}\right)\right)$$

Deep Neural Networks (DNNs)

- A series of **hierarchically** connected functions

- This hierarchy can be very **complex**!

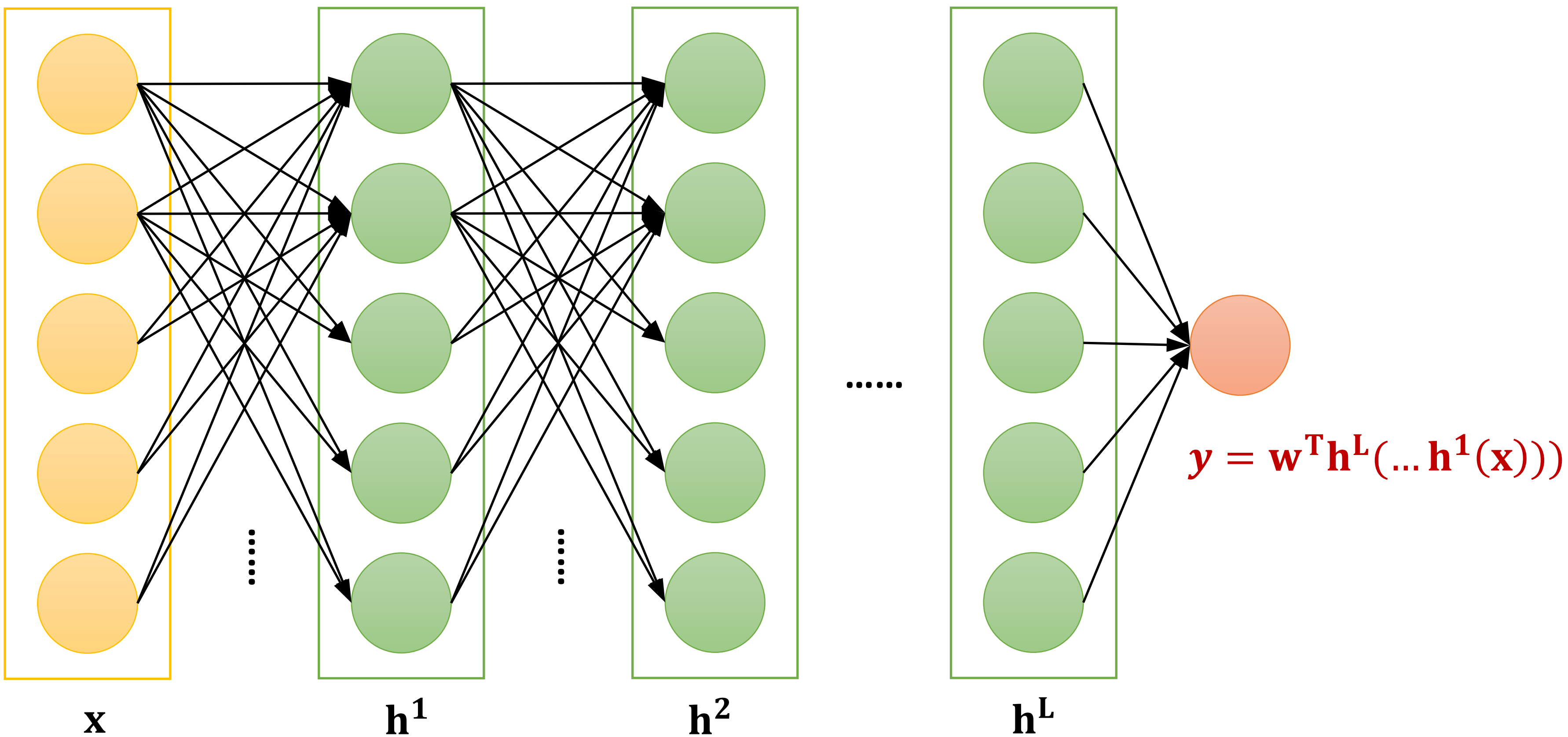
$$a_L(x;\theta_{1,\ldots,L}) = h_L(h_{L-1}(\ldots h_1(x,\theta_1),\theta_{L-1}),\theta_L)$$



Feedforward architecture

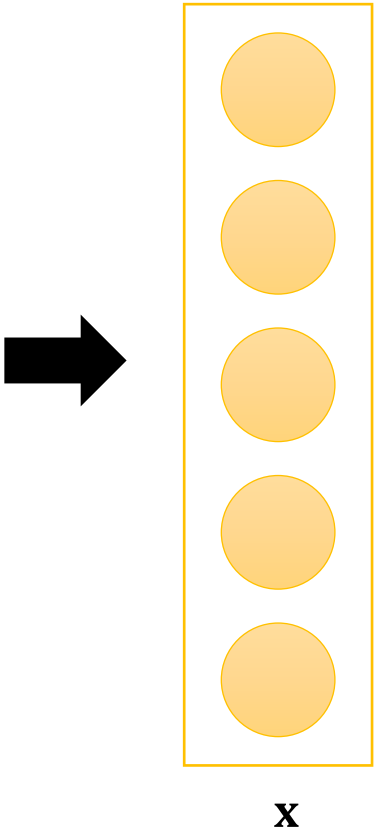
Deep Neural Networks (DNNs)

- What if we go deeper?



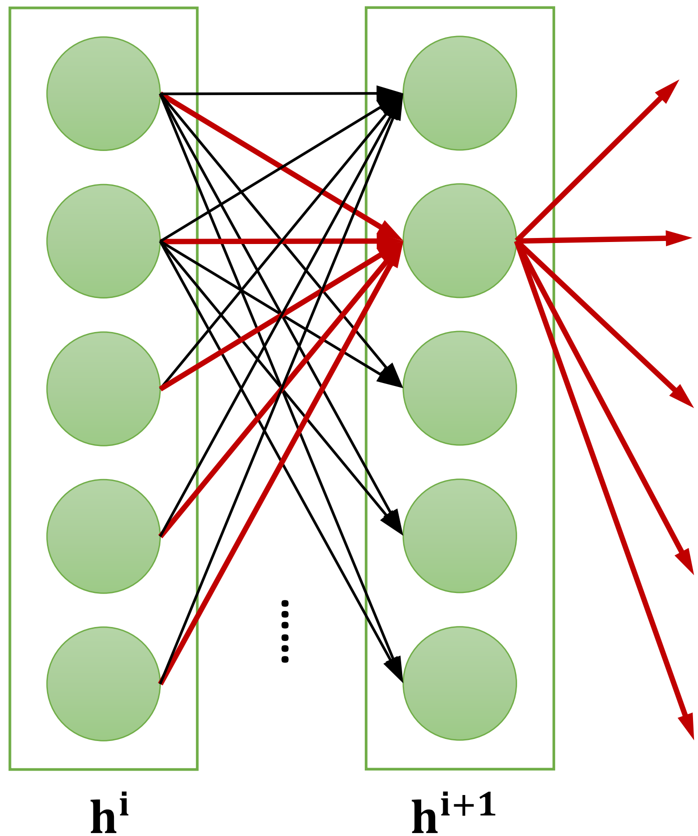
Input Layer of DNNs

- **The input is represented as a vector.**
  - Sometimes, it requires preprocessing, e.g., **normalization**

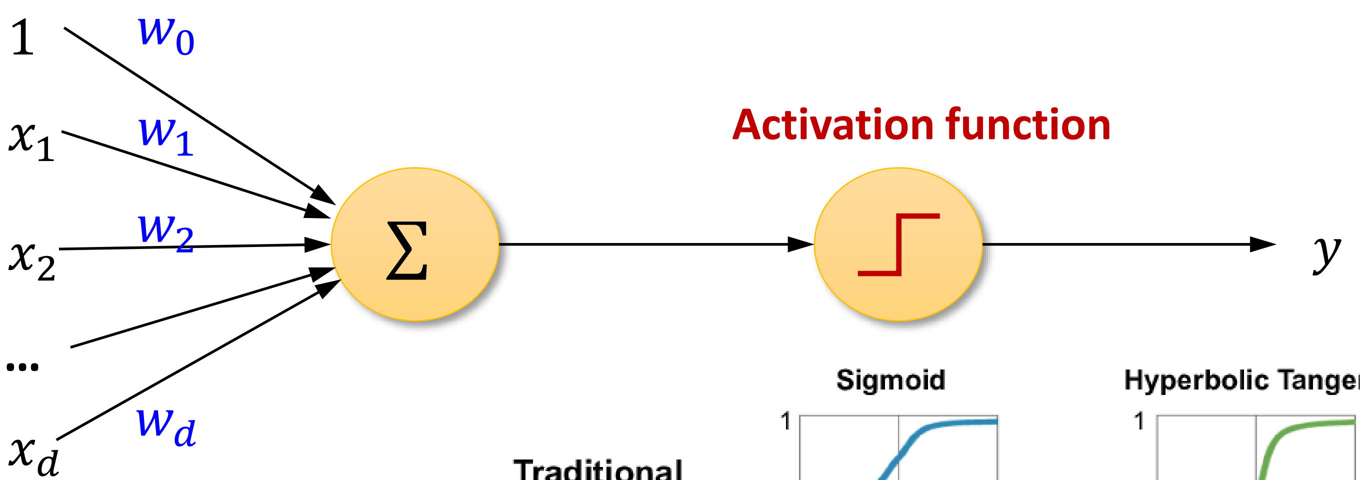


Hidden Layers of DNNs

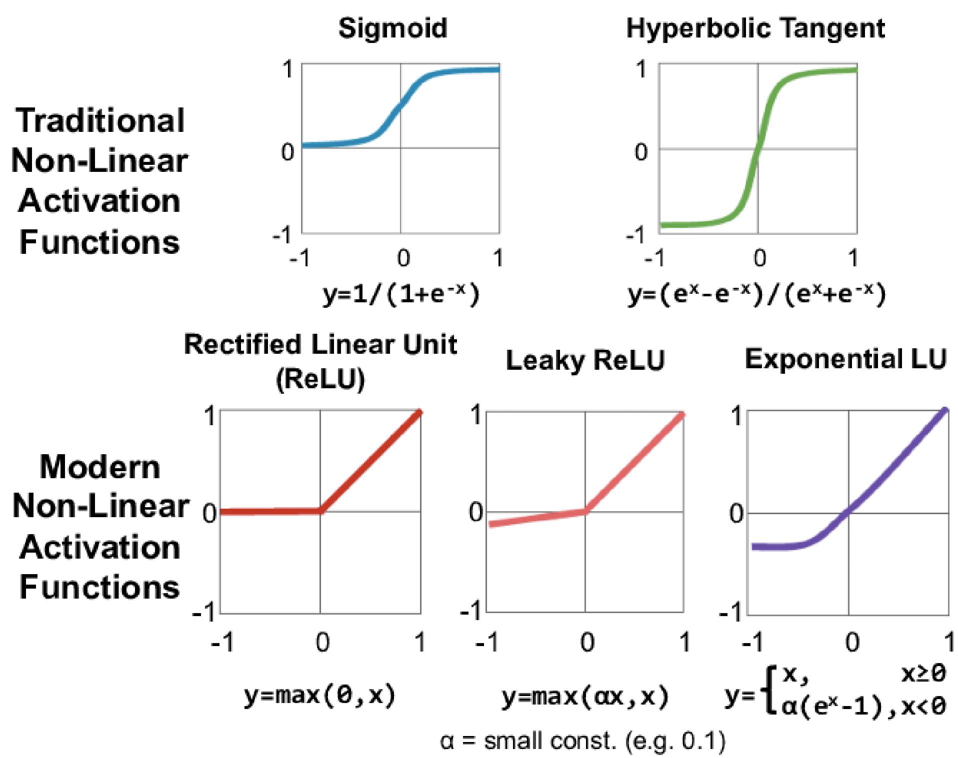
- **Each neuron takes a weighted linear combination of the previous layer.**
  - Provide a single **aggregated** value to the next layer.



Nodes at the Hidden Layers

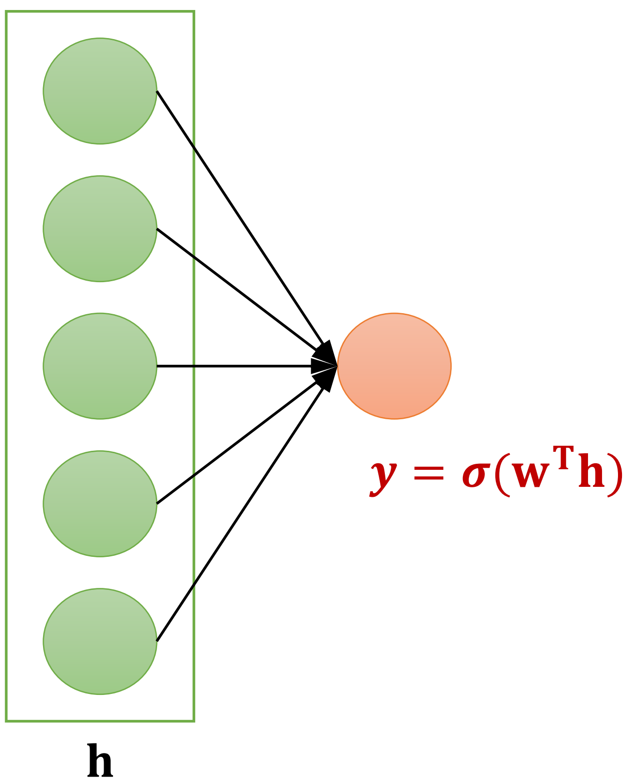


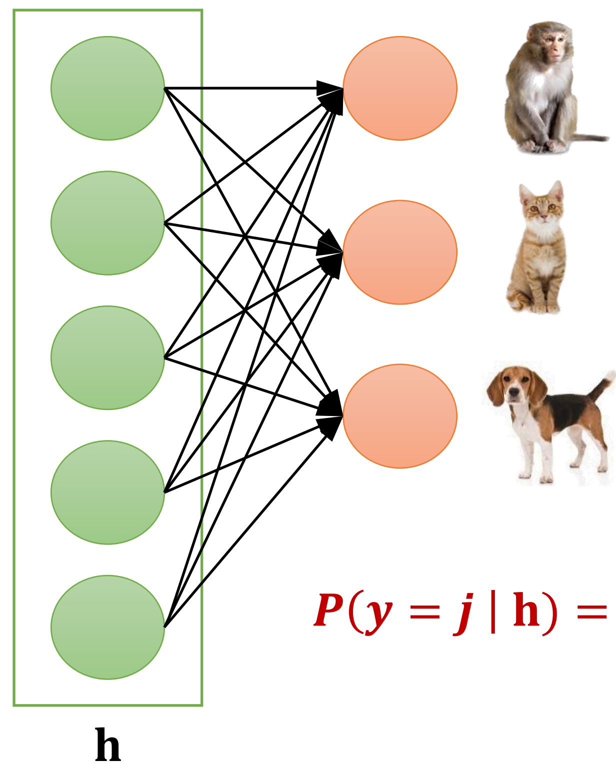
There are various activation functions.



Output Layer of DNNs

- **For output, there are three cases:**
  - Regression: $y = \mathrm{w}^{\mathrm{T}}\mathrm{h}$
  - Binary classification: $y = \sigma\left(\mathrm{w}^{\mathrm{T}}\mathrm{h}\right)$
  - Multi-class classification: $y = \mathrm{softmax}\left(\mathrm{w}^{\mathrm{T}}\mathrm{h}\right)$





$$P(y = j \mid \mathbf{h}) = \frac{e^{\mathbf{w}_j^{\mathrm{T}}\mathbf{h}}}{\sum_{i=1}^{k} e^{\mathbf{w}_i^{\mathrm{T}}\mathbf{h}}}$$

Q&A