A Brief Introduction to Artificial Intelligence

Yukihiko Yamasita
Dept. of International Development Engineering
Tokyo Institute of Technology

Contents

1 Fields of AI

AI stands on many fields.

Application

- Expert system
  - A computer system that emulates the decision-making ability of a human expert.
  - Prediction, diagnosis, design, planning, monitoring, debugging, repair, instruction, control, etc.
- Natural language processing
  - Understand and generate sentences to communicate between human and computer
  - Language translation
  - Speech recognition and generation
- Pattern recognition
  - Optical character recognition (OCT): Postal number reader
  - Computer vision, face detection and recognition.

- Autonomous vehicle and robots
- Art (music and painting) generation
- Games (chess and Go)
- Big Data
  - Extract information from a huge amount of data (The history of web accesses, tweets, positions, texts, and so on)
  - Marketing
  - Diagnostics : Building, bridge, etc. Airplane, train, etc.

Hardware

- General purpose computer
  - ENIAC : Electronic Numerical Integrator and Computer, 1945
  - IBM System 360, i386, Core, ARM, ...
- LISP machine (1973)
  - LISP (List Processor) : Programming language to handle symbols easily

- PSI (Personal Sequential inference Machine),
  PIM (Parallel Inference Machine)
  - Prolog : Programming language for logical inference
- GPGPU (General Purpose Graphic Processing Unit)
  - NVidia TESLA V100: Over 5,000 calculation cores.
    7 TFlops (FP64), 28 TFlops (FP16),
  - Deep learning and image processing (Linear algebra).
  - CUDA : Compute Unified Device Architecture

Methods

- Symbol processing
- Logic inference
- Statistical learning
- Neural networks
- Deep learning

2 Statistical learning theory

2.1 Framework

- z: pattern
- Pattern are observed randomly.
- $F(z)$: cumulative probability distribution function
- Assumption: each pattern has its category.
- Assumption: a family of (classification) functions $d_\alpha(z)$ ($\alpha \in \Lambda$) that returns a category for an input pattern z.
- We define a function $Q(\alpha, z)$ with a classification function $d_\alpha(z)$.

$$Q(\alpha, z) \equiv \begin{cases} 0 & \text{(the result of } d_\alpha(z) \text{ is correct)} \\ 1 & \text{(the result of } d_\alpha(z) \text{ is incorrect)} \end{cases}.$$

- $R(\alpha)$: the expectation of the risk of a classification function d_α.

$$R(\alpha) = E_z Q(\alpha, z) = \int Q(\alpha, z) dF(z) \left(= \int Q(\alpha, z) f(z) dz \right).$$

The problem is to obtain the classification function d_α that minimizes $R(\alpha)$.

- For learning, we can use only finite number of patterns.
- K : the number of learning patterns.

$$z_1, z_2, \ldots, z_K.$$

- Assumption: we know the category of each learning patterns.
- The risk of a classification function for learning patterns is calculated by

$$R_{\mathrm{emp}}(\alpha) = \frac{1}{K} \sum_{k=1}^{K} Q(\alpha, z_k).$$

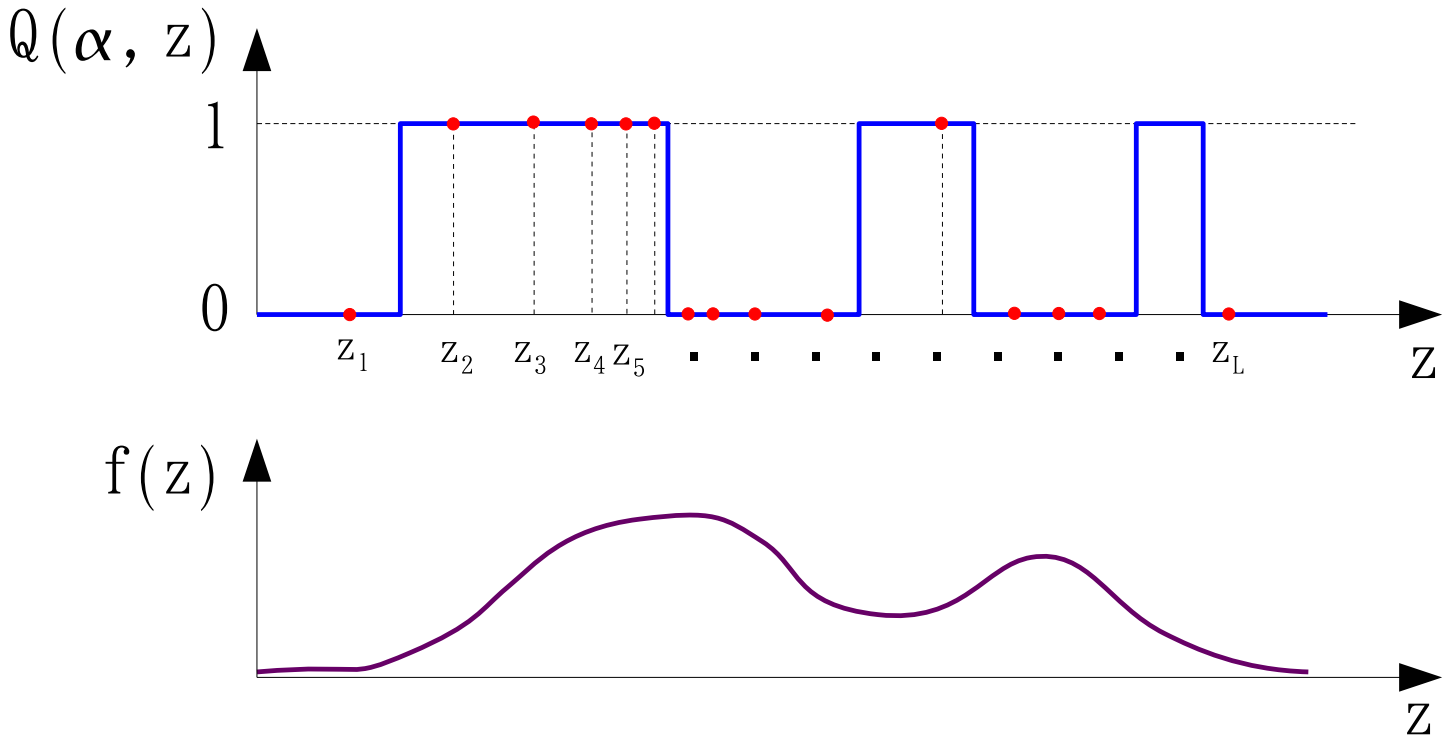
  We call it the empirical risk.
- The difference between two risks is evaluated in statistical learning theory.

$$R(\alpha) = \int Q(\alpha, z) dF(z),$$

$$R_{\mathrm{emp}}(\alpha) = \frac{1}{K} \sum_{k=1}^{K} Q(\alpha, z_k).$$

Definition 1. *ERM (empirical risk minimization) principle*

Use d_α *that minimized* $R_{\mathrm{emp}}(\alpha)$ *instead of* $R(\alpha)$.



- Learning patterns appear according to a p.d.f. $f(z)$.
- Estimate the expectation of $Q(\alpha, z)$ from $Q(\alpha, z_k)$ for learning patterns z_k.

2.2 Entropy, annealed entropy, growth function

- $N^{\Lambda}(z_1, z_2, \ldots, z_K)$: the number of different $(Q(\alpha, z_1), Q(\alpha, z_2), \ldots, Q(\alpha, z_K))$ when we fix $(z_1, z_2, \ldots, z_K)$ and vary $\alpha \in \Lambda$.
- That is the number of different types of risk array of classification functions for a fixed set of input patterns.
- Entropy :

$$H^{\Lambda}(K) = E_{z_1, z_2, \ldots, z_K} \ln N^{\Lambda}(z_1, z_2, \ldots, z_K)$$

- Annealed entropy :

$$H^{\Lambda}_{\text{ann}}(K) = \ln E_{z_1, z_2, \ldots, z_K} N^{\Lambda}(z_1, z_2, \ldots, z_K)$$

- Growth function :

$$G^{\Lambda}(K) = \ln \max_{z_1, z_2, \ldots, z_K} N^{\Lambda}(z_1, z_2, \ldots, z_K)$$

- We have following relation.

$$H^{\Lambda}(K) \leq H^{\Lambda}_{\text{ann}}(K) \leq G^{\Lambda}(K)$$

2.3 Basic inequality

The following theorem is one of the most important theorems for statistical learning theory.

Theorem 1. *We have following relation. (Basic inequality)*

$$P\left\{\sup_{\alpha\in\Lambda}|R(\alpha)-R_{\mathrm{emp}}(\alpha)|>\varepsilon\right\}<4\exp\left\{\left(\frac{H_{\mathrm{ann}}^{\Lambda}(2K)}{K}-\left(\varepsilon-\frac{1}{K}\right)^2\right)K\right\}.$$

- The left hand side is the probability that learning patterns $(z_1, z_2, \ldots, z_K)$ appears, with which the maximum absolute difference between the ensemble risk and the empirical risk among all $\alpha \in \Lambda$ is larger than ε.
- Since $H_{\mathrm{ann}}^{\Lambda}(2K) < G^{\Lambda}(2K)$,

$$P\left\{\sup_{\alpha\in\Lambda}|R(\alpha)-R_{\mathrm{emp}}(\alpha)|>\varepsilon\right\}<4\exp\left\{\left(\frac{G^{\Lambda}(2K)}{K}-\left(\varepsilon-\frac{1}{K}\right)^2\right)K\right\}.$$

2.4 VC (Vapnik-Chervonenkis) dimension

Theorem 2. *For any integer K, we have $G^{\Lambda}(K) = K \ln 2$ or there exists an integer h (VC dimension) such that*

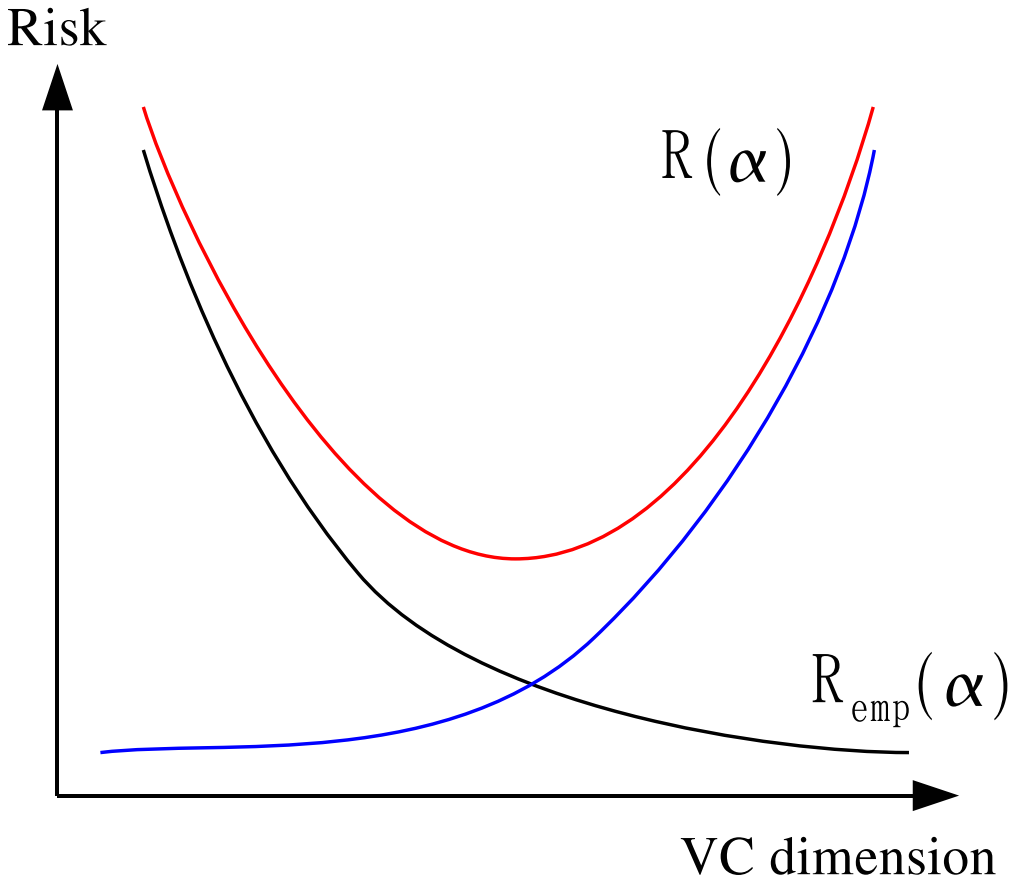
$$\begin{aligned} G^{\Lambda}(K) &= K \ln 2 && (K \leq h) \\ G^{\Lambda}(K) &\leq \ln \left(\sum_{i=0}^{h} {}_{K}C_{i} \right) && (K > h) \end{aligned}$$

- When the number of samples is less than or equal to h, there exists an classifier that splits all samples correctly for any labeling of samples.

Theorem 3.

$$\begin{aligned} & P \left\{ \sup_{\alpha \in \Lambda} |R(\alpha) - R_{\mathrm{emp}}(\alpha)| > \varepsilon \right\} \\ < \; & 4 \exp \left\{ \left(\frac{h(1 + \ln(2L/h))}{K} - \left(\varepsilon - \frac{1}{K} \right)^2 \right) K \right\}. \end{aligned}$$

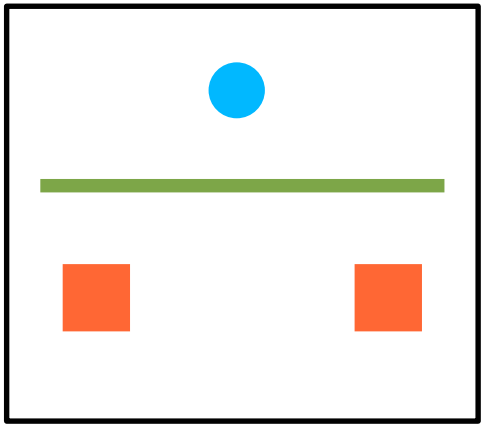
2.5 VC dimension and risk

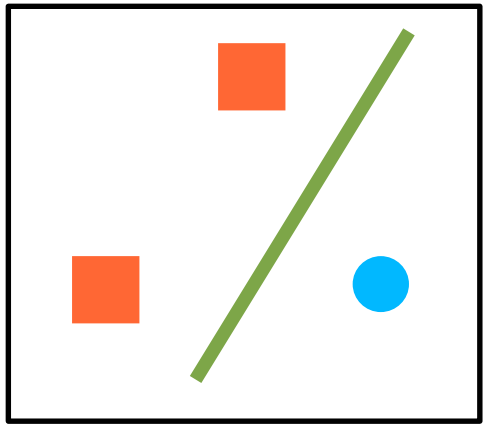


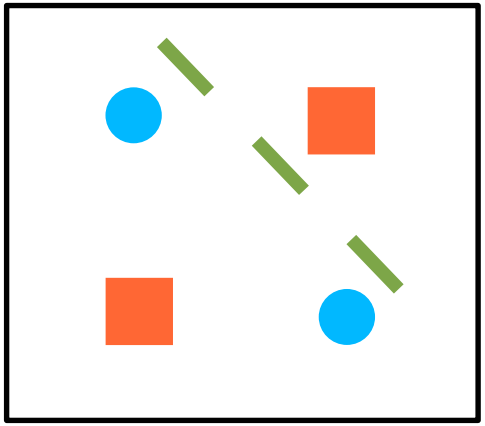
Corollary 1. *With probability* $1 - \eta$ *the following relation holds.*

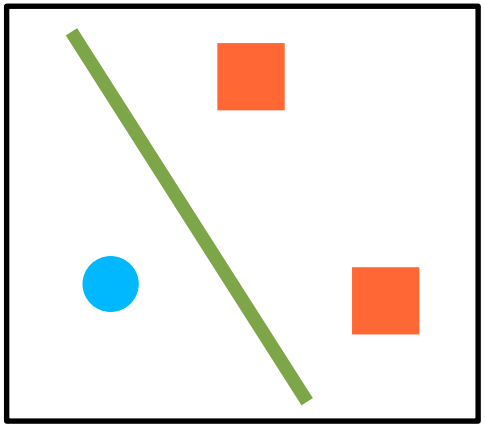
$$R(\alpha) - R_{\text{emp}}(\alpha) \leq \sqrt{\frac{h(1 + \ln(2K/h)) - \ln(\eta/4)}{K}}.$$

2.6 Linear classifier and VC dimension









3 points can be shuttered

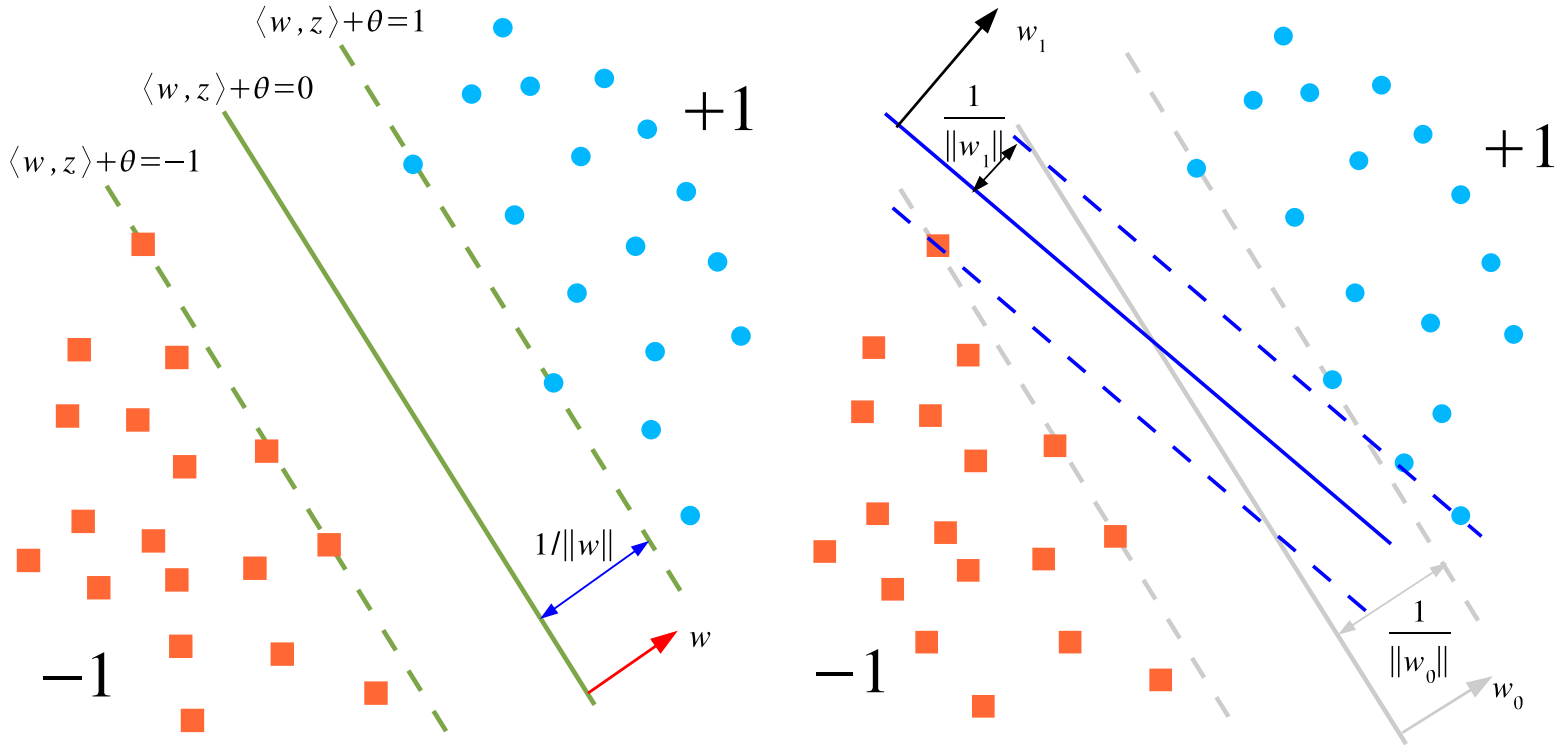
4 points cannot be shuttered with some labeling

2.7 Support vector machine (SVM)

- A linear classifier (of two-class problem):

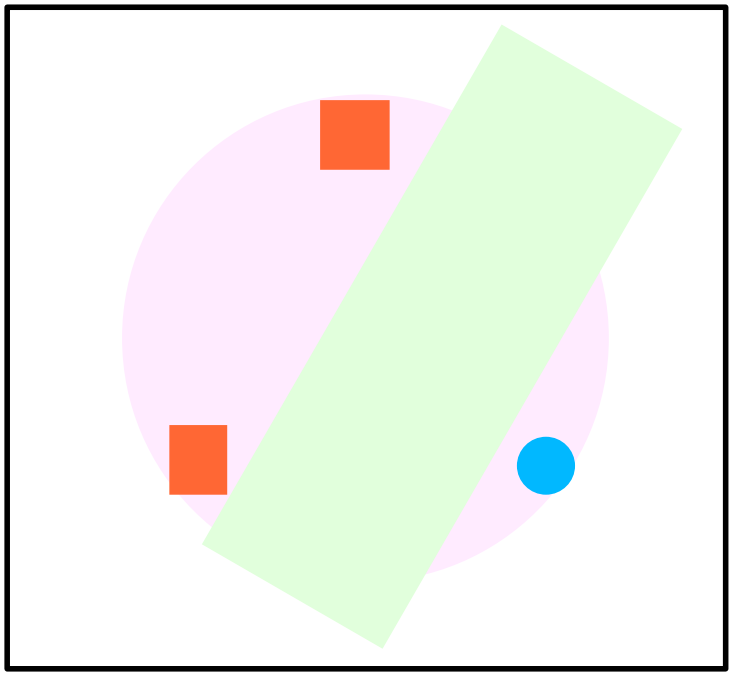
$$d(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + \theta.$$

- $(\boldsymbol{x}_k, y_k)$: a pair of sample $\boldsymbol{x}_k$ and its label $y_k (= \pm 1)$ $(k = 1, 2 \ldots, K)$



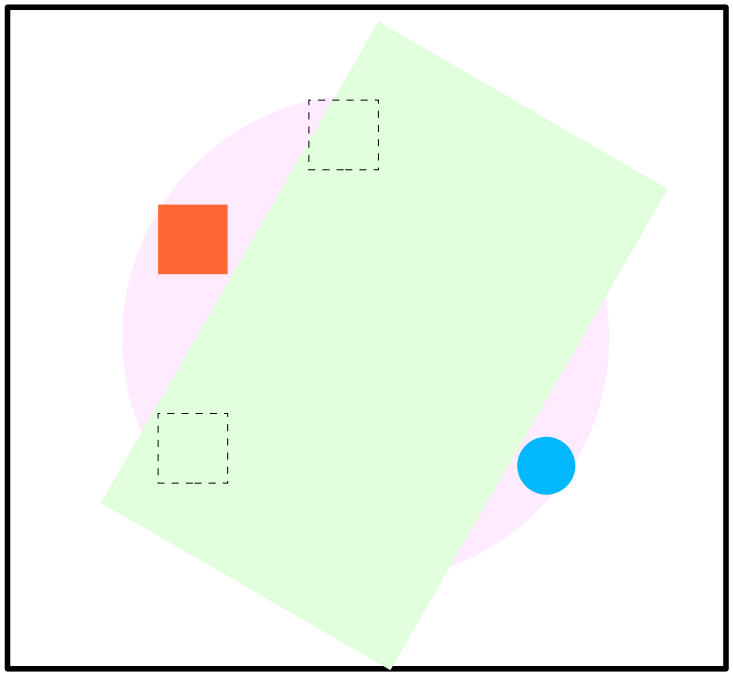
Maximize margin

2.8 Margin and VC dimension



VC dimension = 3

when the margin is small



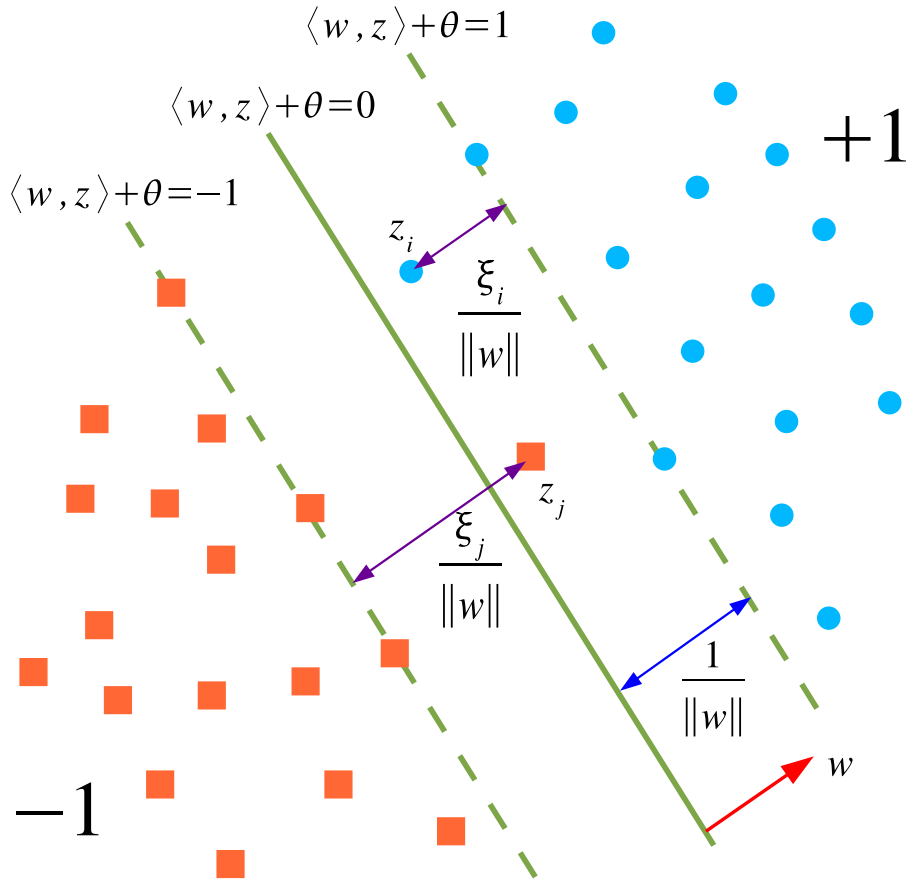
VC dimension = 2

when the margin is large

- Assume that the norm of patterns are upper bounded. The larger margin the fewer VC dimension is.

2.9 Soft margin

- Soft margin : Some errors are allowed.



$$\frac{1}{\text{Margin}} + C\,(\text{Sum of errors}) \to min.$$

2.10 Solution of linear SVM

Lagrange's method :

Subject to

$$0 \leq \alpha_k \leq C, \ (k = 1, 2, \ldots K) \quad \sum_{l=1}^{K} \alpha_k y_k = 0 \tag{1}$$

Maximize

$$L_D = \sum_{k=1}^{K} \alpha_k - \frac{1}{2} \sum_{k=1}^{K} \sum_{l=1}^{K} \alpha_k \alpha_l y_k y_l \langle \boldsymbol{z}_k, \boldsymbol{z}_l \rangle \tag{2}$$

- Variables for optimization problem are only α_l.
- The dimension of this problem is the number of samples.
- The classification function can be calculated by

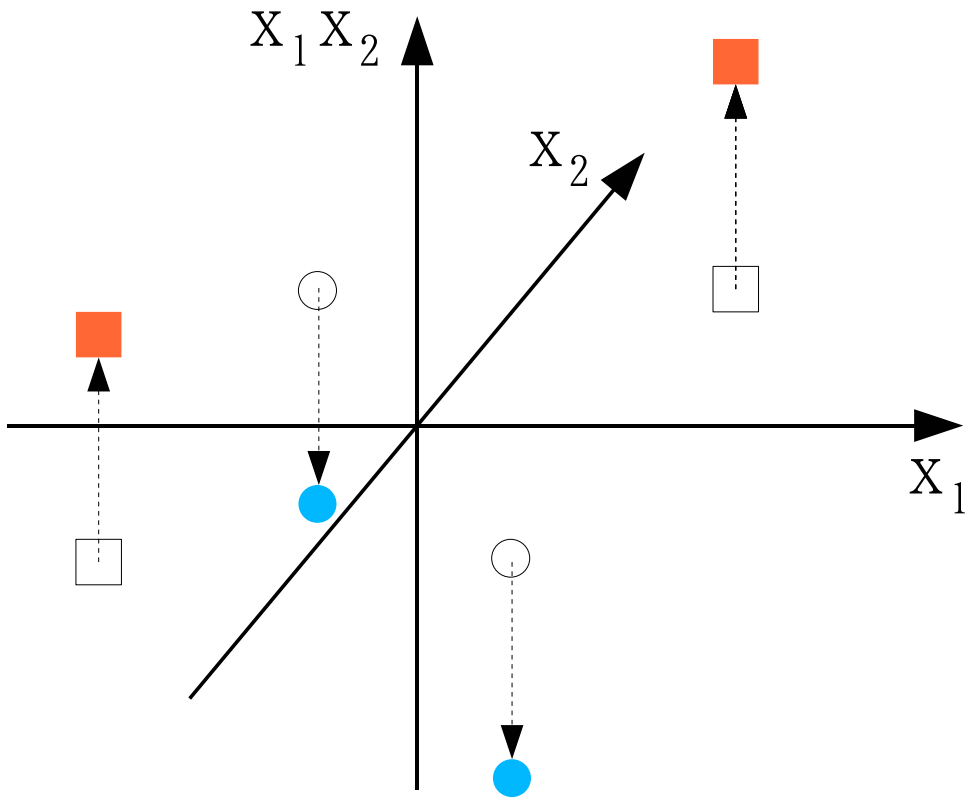
$$d(\boldsymbol{x}) = \sum_{\boldsymbol{z}_l \in SV} \alpha_l y_l \langle \boldsymbol{z}_l, \boldsymbol{x} \rangle + \theta \tag{3}$$

- θ is obtained by using KKT condition.

2.11 Kernel method for nonlinear SVM

- Patterns are mapped by a nonlinear function Φ.

Example: $\Phi(x_1, x_2) = (x_1^2, x_2^2, \sqrt{2}x_1x_2, \sqrt{2}x_1, \sqrt{2}x_2, 1)$



Kernel function : $k(\boldsymbol{x}, \boldsymbol{z}) \equiv \langle \Phi(\boldsymbol{x}), \Phi(\boldsymbol{z}) \rangle = \langle \boldsymbol{x}, \boldsymbol{z} \rangle^2 + 1$

2.12 Nonlinear SVM

- Classification function and its criterion of linear SVM are expressed only by inner products. (Kernel method)
- Classification function

$$d(\boldsymbol{x}) = \sum_{\boldsymbol{z}_l \in SV} \alpha_l y_l k(\boldsymbol{z}_l, \boldsymbol{x}) + \theta \tag{4}$$

- The optimization problem for learning is given as
  Subject to

$$0 \leq \alpha_k \leq C, (l = 1, 2, \ldots K), \quad \sum_{k=1}^{K} \alpha_k y_k = 0 \tag{5}$$

  minimize

$$L_D = \sum_{k}^{K} \alpha_k - \frac{1}{2} \sum_{k=1}^{K} \sum_{l=1}^{K} \alpha_k \alpha_l y_k y_l k(\boldsymbol{z}_k, \boldsymbol{z}_l) \tag{6}$$

  with respect to α_k.

3 Neural networks

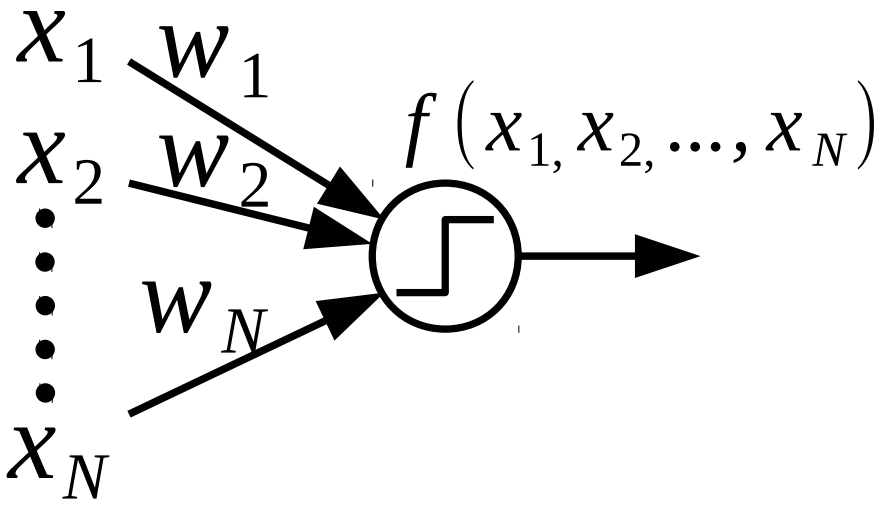
3.1 Perceptron

- An imitation of a biological neuron
  - x_i : inputs to a perceptron. $(i = 1, 2, \cdots, N)$.
  - w_i : weights $(i = 1, 2, \cdots, N)$.
  - θ : threshold.
- We extend an input vector and a weight as $x_0 = 1$ and $w_0 = \theta$, then we have

$$\sum_{i=1}^{N} w_i x_i + \theta = \sum_{i=0}^{N} w_i x_i$$

- The output $f(x_1, \ldots, x_N)$ of perceptron is given by.

$$f(x_1, \ldots, x_N) = \begin{cases} 1 & \left(\sum_{i=0}^{N} w_i x_i > 0\right) \\ 0 & (\text{otherwise}) \end{cases}.$$



Perceptron

Training algorithm

- $(x_i^{(k)}, y^{(k)})$: a pair of sample data $\boldsymbol{x}_i^{(k)}$ and its label $y^{(k)}(= \pm 1)$ $(l = 1, 2 \ldots, K)$
- Update rule:

$$w_i \Leftarrow w_i - \mu(f(x_0, \ldots, x_N) - y^{(k)})x_i^{(k)}$$

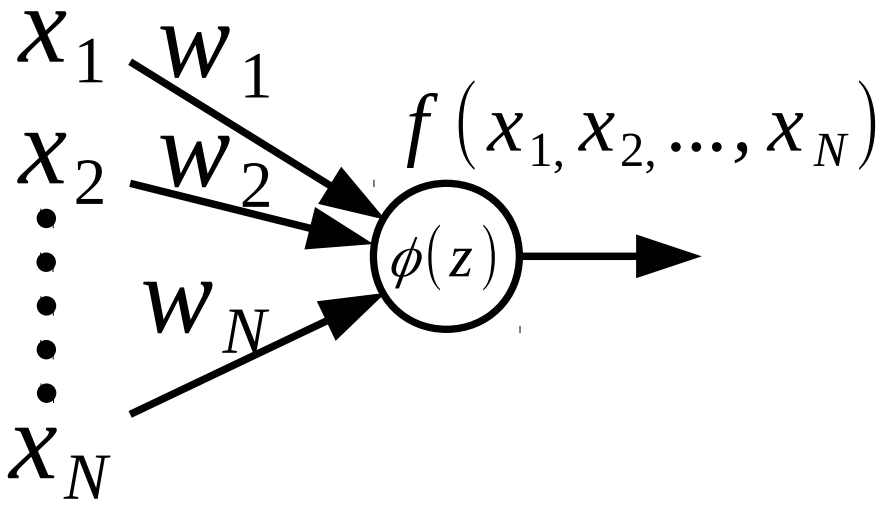
3.2 Neural network

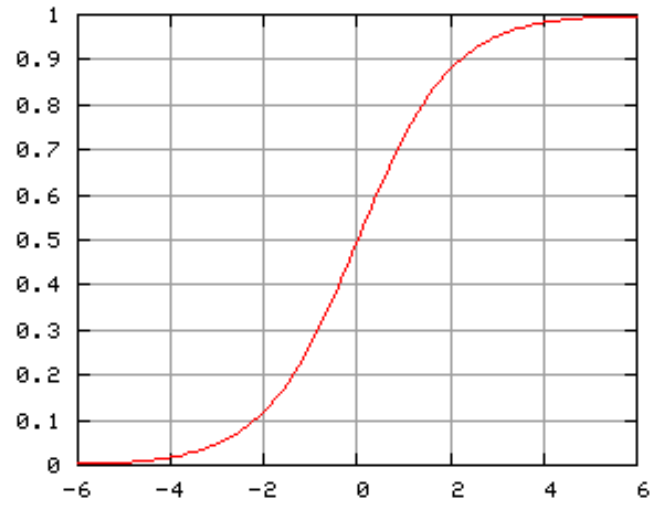
Artificial neuron

$$f(\boldsymbol{x}) = \phi(z), \quad z = \langle \boldsymbol{w}, \boldsymbol{x} \rangle = \sum_{n=1}^{N} w_n x_n$$

- $\phi(z)$ is the activation function.
  Example: sigmoid function

$$\phi(z) = \frac{1}{1 + e^{\gamma z}}$$





https://upload.wikimedia.org/wikipedia/commons/a/ac/Logistic-curve.png

Multi-layer network

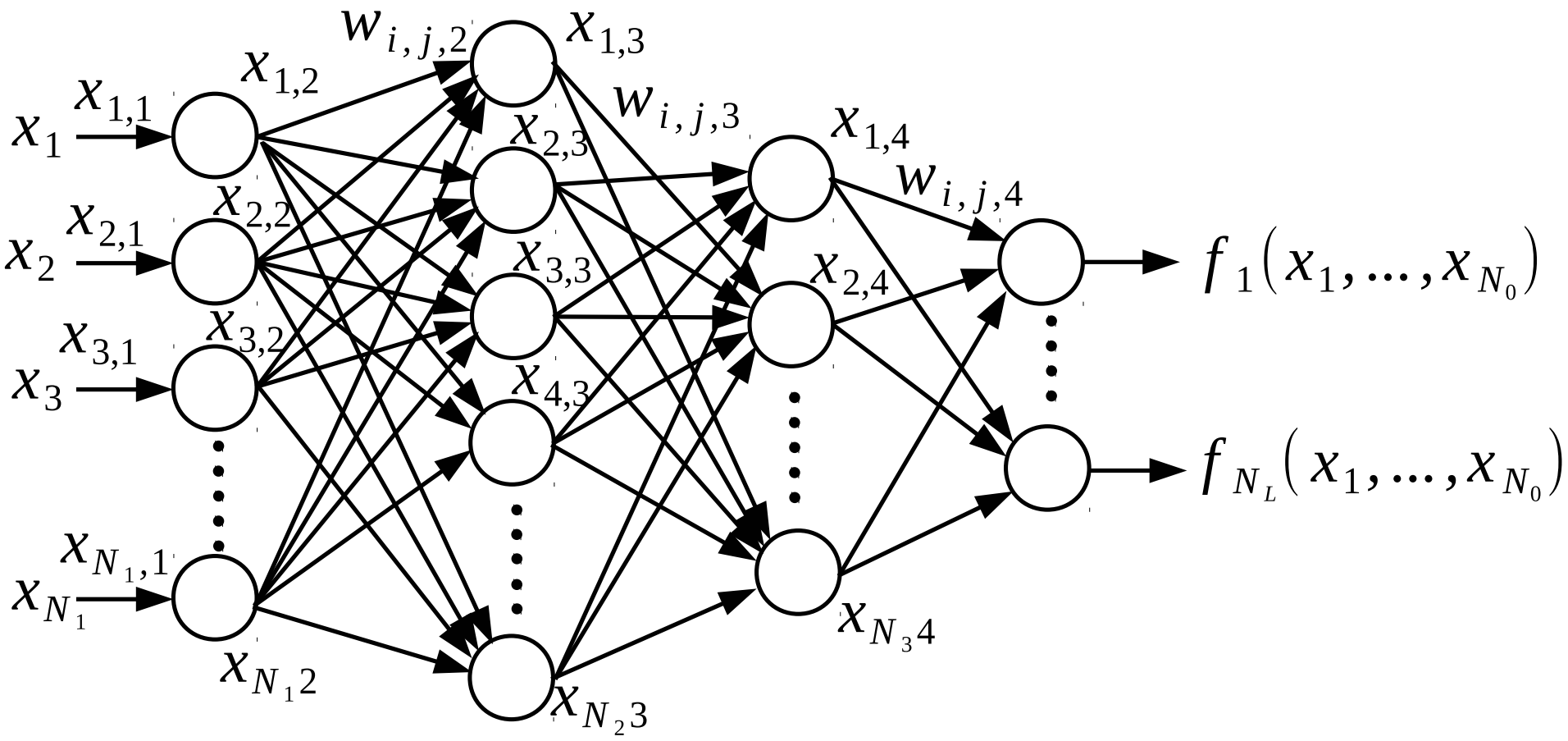
- L : the number of layers.
- N_l : the number of neurons in the l-th layer.
- $\phi_l(z)$: the activation function in the l-th layer.
- $x_{i,\,l}$: the i-th input of a neuron in the l-th layer.
- $w_{i,\,j,\,l}$: the weight from the ouptput of the j-th neurons in the $(l-1)$-th layer to the i-th neurons in the l-th layer.
  (The thresholds are included in weights.)

$$z_{i,\,l} = \sum_{j=1}^{N_{(l-1)}} w_{i,\,j,\,l}\, x_{j,\,l} \tag{7}$$

$$x_{i,\,(l+1)} = \phi_l(z_{i,\,l}) \tag{8}$$

- $x_{j,\,1}$ is the input of the network.
- $x_{j,\,(L+1)}$ is the output of the network.

Example: 4-layer neural network



$$y_m = \sum_{i=1}^{N_3} w_{m,i,4}\phi_3\left(\sum_{j=1}^{N_2} w_{i,j,3}\phi_2\left(\sum_{n=1}^{N_1} w_{j,n,2}x_n\right)\right)$$

3.3 Back propagation

Training algorithm for multi-layer network.

- Neural network: $y_m = f_m(x_0, \ldots, x_{N_1}; \{w_{i,j,l}\})$.
- $(x_0^{(k)}, \ldots, x_{N_1}^{(k)}; y_1^{(k)}, \ldots, y_{N_L}^{(k)})$: a pair of sample $x_n^{(k)}$ and its target output $y_m^{(k)} (= \pm 1)$ $(k = 1, 2 \ldots, K)$
- Minimum squared error (MSE)

$$\text{argmin}_{w_{i,j,l}} \sum_{k=1}^{K} \sum_{m=1}^{N_L} \left| f_m(x_0^{(k)}, \ldots, x_{N_1}^{(k)}; \{w_{i,j,l}\}) - y_m^{(k)} \right|^2$$

- Maximum gradient method is applied for each sample point. Update rule:

$$w_{i,j,l} \Leftarrow w_{i,j,l} - \mu \frac{\partial}{\partial w_{i,j,l}} \sum_{m=1}^{N_L} \left| f_m(x_0^{(k)}, \ldots, x_{N_1}^{(k)}; \{w_{i,j,l}\}) - y_m^{(k)} \right|^2$$

Update rule:

$$w_{i,\,j,\,l} \Leftarrow w_{i,\,j,\,l} - \mu \frac{\partial}{\partial w_{i,\,j,\,l}} \sum_{m=1}^{N_L} \left| f_m(x_0^{(k)}, \ldots, x_{N_1}^{(k)}; \{w_{i,\,j,\,l}\}) - y_m^{(k)} \right|^2$$

- In the L-th layer, the errors of outputs is defined by

$$\varepsilon_{m,\,L} = f_m(x_0^{(k)}, \ldots, x_{N_1}^{(k)}; \{w_{i,\,j,\,l}\}) - y_m^{(k)}. \quad (m = 1, 2, \ldots, N_L)$$

- Then, the update rule of $w_{i,\,j,\,l}$ is given by,

$$w_{i,\,j,\,l} \Leftarrow w_{i,\,j,\,l} - 2\mu \sum_{m=1}^{N_L} \varepsilon_{m,\,L} \frac{\partial}{\partial w_{i,\,j,\,l}} f_m(x_0, \ldots, x_{N_1}; \{w_{i,\,j,\,l}\}).$$

- We describe the derivative of $\phi_l(z)$ as

$$\phi_l'(z) = \left. \frac{d\phi_l}{dz} \right|_z$$

$$\frac{\partial}{\partial w_{i,j,l}} f_m(x_0, \ldots, x_{N_1}; \{w_{i,j,l}\})$$

$$= \sum_{i_L=0}^{N_L} \phi'_L(z_{i_L,L}) w_{i_L,i_{L-1},L} \cdot \sum_{i_{L-1}=0}^{N_{L-1}} \phi'_{L-1}(z_{i_{L-1},(L-1)}) w_{i_{L-1},i_{L-2},(L-1)}$$

$$\cdots \sum_{i_{l+1}=0}^{N_{l+1}} \phi'_{l+1}(z_{i_{l+1},(l+1)}) w_{i_{l+1},i,(l+1)} \phi'_l(z_{i,l}) x_{j,l} \tag{9}$$

- The update rule can be described as

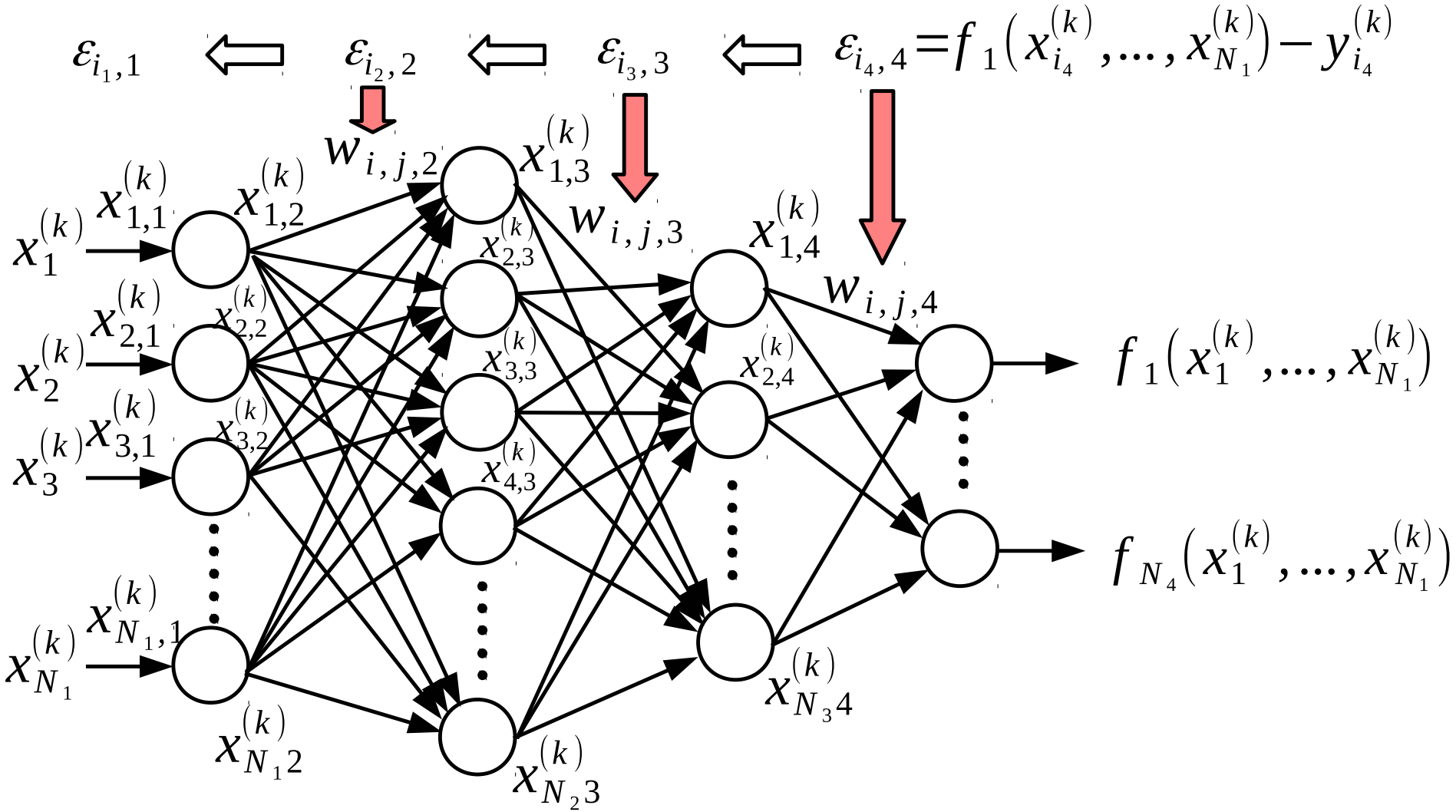
$$w_{i,j,l} \Leftarrow w_{i,j,l} - 2\mu \phi'_l(z_{i,l}) x_{j,l}\, \varepsilon_{i,l},$$

where $\varepsilon_{i,l}$ is the errors of outputs to be reduced in the l-th layer:

$$\varepsilon_{i_l,l} = \sum_{i_{l+1}=0}^{N_{l+1}} \phi'_l(z_{i_{l+1},l}) w_{i_{l+1},i_l,(l+1)}\, \varepsilon_{i_{l+1},(l+1)}.$$

This can be calculated layer by layer from the end to the beginning.

$\Rightarrow$ Back propagation.

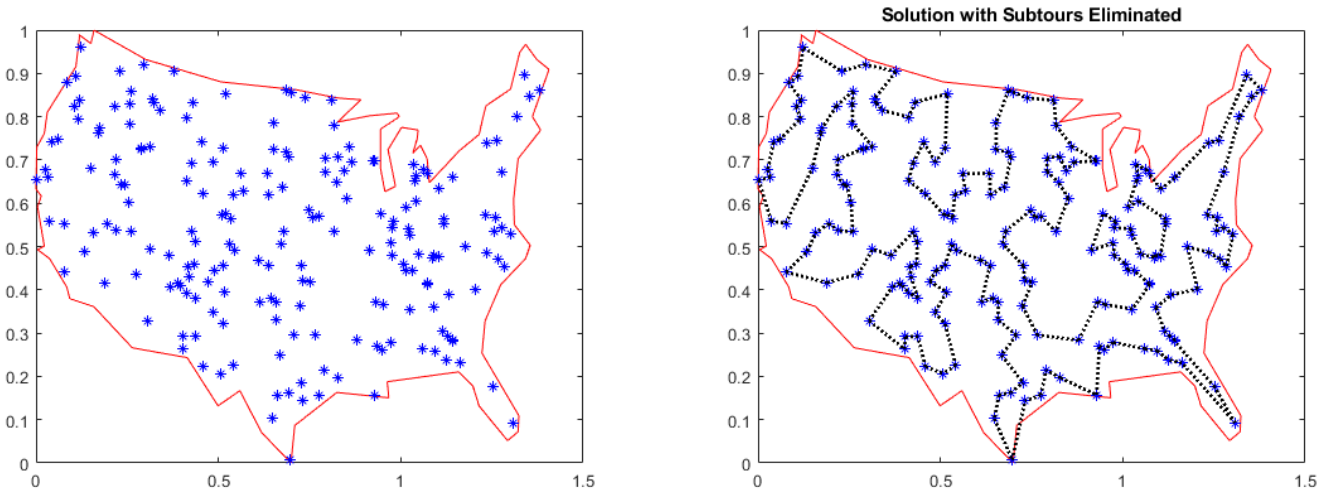


$$\varepsilon_{i_l,l}=\sum_{i_{l+1}=0}^{N_{l+1}}\phi_l'(z_{i_{l+1},l})w_{i_{l+1},i_l,(l+1)}\,\varepsilon_{i_{l+1},(l+1)}.$$

3.4 Hopfield network

Traveling Salesman Problem (TSP)

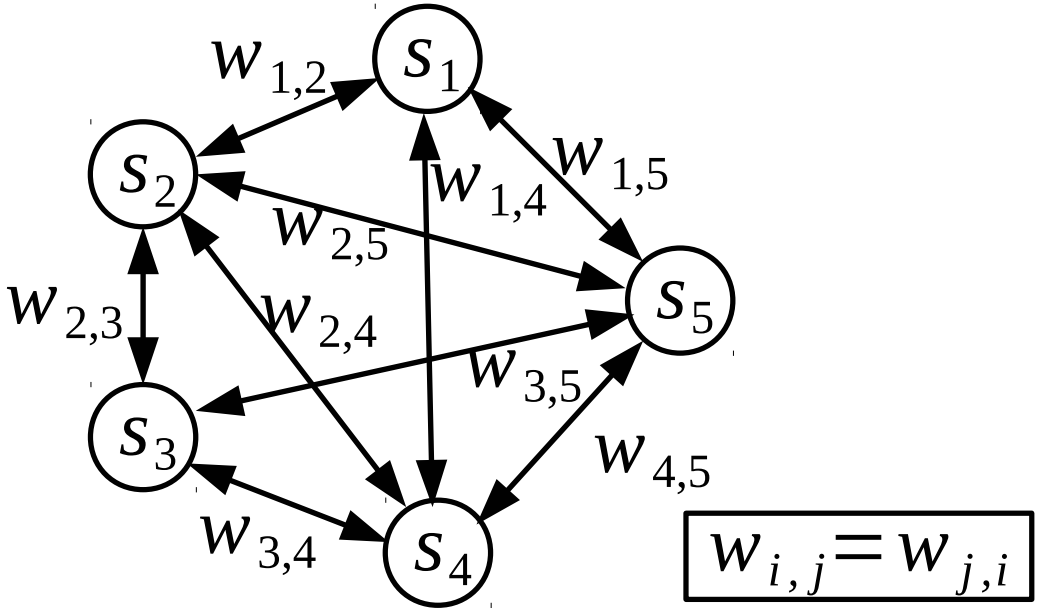
- C_m : city m. ($m = 1, 2, \ldots, M$).
- d_{mn} : the distance between C_m and C_n.
- Search a path of the minimum length that go through all cities and return to the start city.
- Its computational complexity is NP-complete (even if the distances are given by Euclidean distance on a plane).



URL: https://jp.mathworks.com/help/optim/ug/travelling-salesman-problem.html

Hopfield network

- A recurrent network.
- $s_i (= 0, 1)$: the state of the i-th unit (neuron) ($i = 1, 2, \ldots, N$).
- w_{ij} : the strength of connection between the i-th and the j-th units.
- θ_i : the threshold of the i-th unit.
- $w_{ij} = w_{ji}$



Hopfield network

- Update rule:

$$s_i \Leftarrow \begin{cases} 1 & \left(\sum_{j=1}^{N} w_{ij} s_j - \theta_i \geq 0\right) \\ 0 & \text{(otherwise)} \end{cases}.$$

- Energy function:

$$E = -\frac{1}{2} \sum_{i=1}^{N} \sum_{j=1,\, j \neq i}^{N} w_{ij} s_i s_j + \sum_{i=1}^{N} \theta_i s_i$$

- From such a quadratic energy function, we can construct a Hopfield network.
- The energy is not increased by the update.
- At first they consider the Hopfield network can provide the solution of energy minimization problem (such as TSP) quickly.
- However, it is not true. the Hopfield network cannot give the global optimum solution in many cases.
- The decreasing will stop at a local minimum point.

TSP by Hopfield network

- For $m = 1, 2 \ldots, M$ and $n = 0, 1 \ldots, M$, we define a mapping as

$$i(m, n) = \begin{cases} m + Mn & (n < M) \\ m & (n = M) \end{cases}.$$

- The states are defined by

$$s_{i(m,n)} = \begin{cases} 1 & \text{(if the path reaches city } m \text{ after } n \text{ moves)} \\ 0 & \text{(otherwise)} \end{cases}.$$

- Energy function:

$$E = \sum_{n=0}^{M-1} \sum_{m}^{M} \sum_{l}^{M} d_{m,l} s_{i(m,n)} s_{i(l,n+1)}$$
$$+\alpha \sum_{n=1}^{M} \left(\sum_{m=1}^{M} s_{i(m,n)} - 1 \right)^2 + \beta \sum_{m=1}^{M} \left(\sum_{n=1}^{M} s_{i(m,n)} - 1 \right)^2$$

- TSP can be expressed by a Hopfield network.

3.5 Boltzmann machine

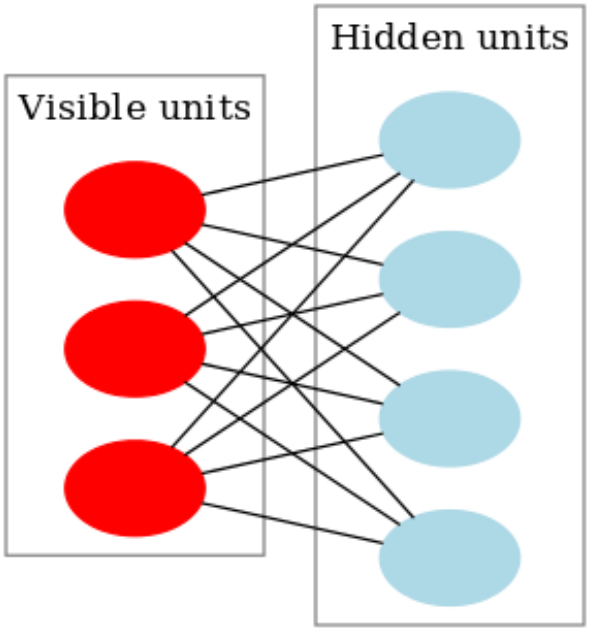
- To solve the problem of local minimum points, the probability is introduced to the Hopfield network.
- T : temperature
- $\Delta E_i = \sum_{j=1}^{N} w_{ij} s_j - \theta_i$: the difference of energy
- Probability $s_i = 1$ at the next stage is given by

$$\frac{1}{1 + \exp\left(-\frac{\Delta E_i}{T}\right)}$$

- There is a possibility to escape from a local minimum point.
- The probability of the set of states of which energy is the minimum is higher than the other sets.
- The temperature T is set to a high value and is decreased gradually to make the convergence faster with a more precise solution.
  $\Rightarrow$ Simulated annealing.

Restricted Boltzmann machine (RBM)

- Two types of units : Visible and Hidden (Latent) units.
- Weights $w_{i,j}$ are zero between units of the same type.
- RBM is used in the training of Deep learning.
  $\Rightarrow$ Weights $w_{i,j}$ are trained.



Restricted Boltzmann machine (RBM)

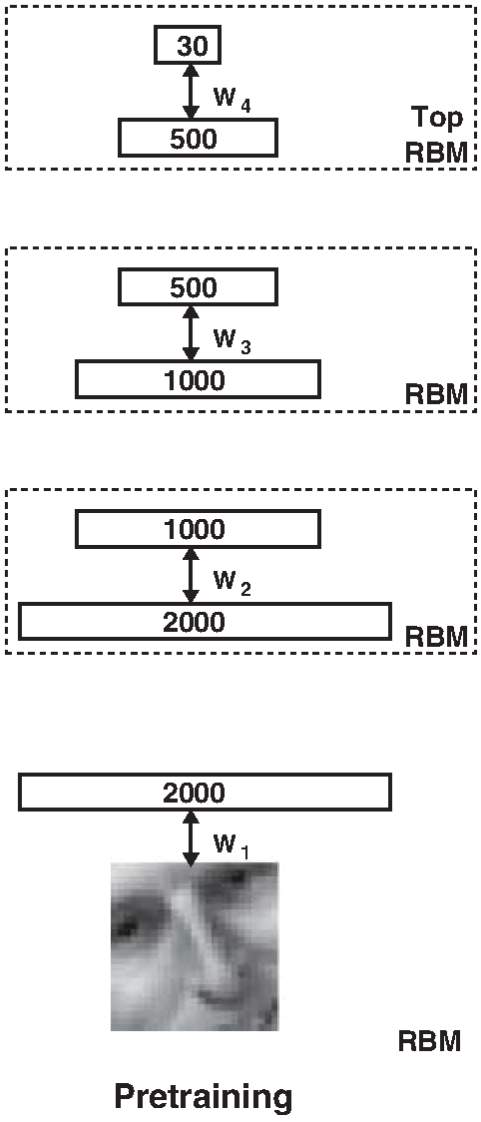
URL: https://en.wikipedia.org/wiki/Boltzmann_machine#/media/File:Restricted_Boltzmann_machine.svg

3.6 Deep learning

- Three layers used be standard for neural network when BP is used.
- Even if the number of layers is increased, the information for learning is dispersed so that it does not transfer properly between lower and upper layers and PB does not work well.
- Deep learning is proposed for 'autoencoder' (automatically tuned encoder for data compression) by a neural network. (G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," Science, vol.313, July 2006.)
- Encoder and decoder are joined. (Input and output are the same.)
- A intermediate layer of which the number of neuron is smaller than that of input/output layer expresses features of patterns
- The deep learning for the autoencoder consists of
  - Pretraining by restricted Boltzmann machine for each layer
  - Unrolling by BP
  - Fine-tuning

Pretraining by Restricted Boltzmann machine (RBM)

- The network for the encoder is trained layer by layer.
- v_i : i-th component of visible unit
  (0 or 1, binary pixel)
  - Input image for the first layer.
  - The output of the $(l-1)$-th layer for l-th layer.
- h_j : j-th component of hidden unit
  (0 or 1, binary feature)



Training of RBM

- Repeat the following steps
  - Obtain h_j from v_i by the following probability

$$P(h_j = 1) = \frac{1}{1 + e^{\sum_i w_{i,j} v_j}}$$

  - Obtain the confabulation $\tilde{v}_j$ from h_i by the following probability

$$P(\tilde{v}_j = 1) = \frac{1}{1 + e^{\sum_j w_{i,j} h_i}}$$

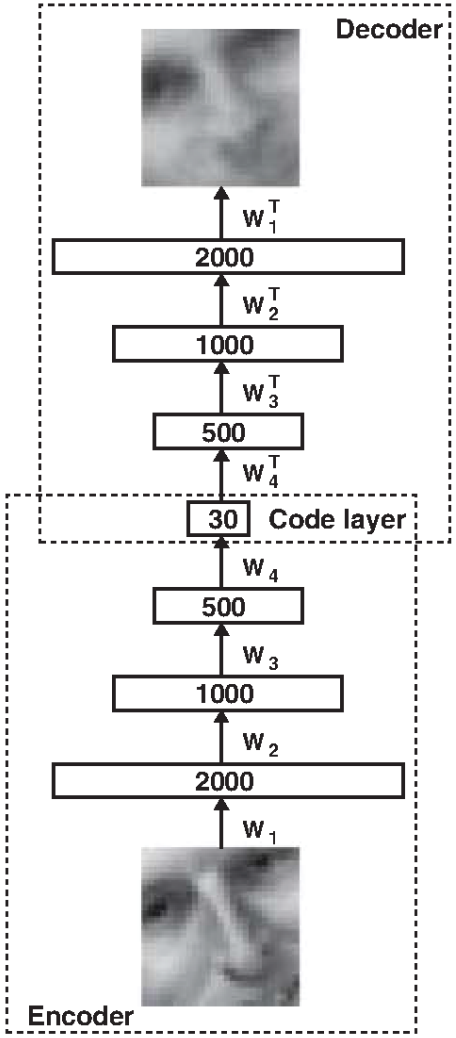
- Calculate $\langle h_i v_j \rangle$ that is the fraction of times when v_j and h_i are on together.
- Calculate $\langle h_i \tilde{v}_j \rangle$ similarly.
- Update the weight :

$$w_{i,j} \Leftarrow w_{i,j} + \lambda(\langle h_i v_j \rangle - \langle h_i \tilde{v}_j \rangle),$$

  where $\lambda(> 0)$ is a learning rate.

Unrolling by BP

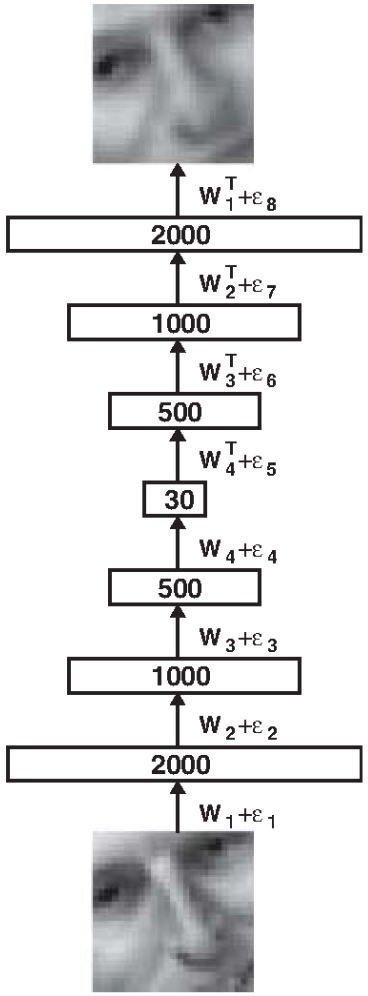
- The transposition of weight in l-th layer is used for $(L - l + 1)$



Unrolling

Fine-tuning

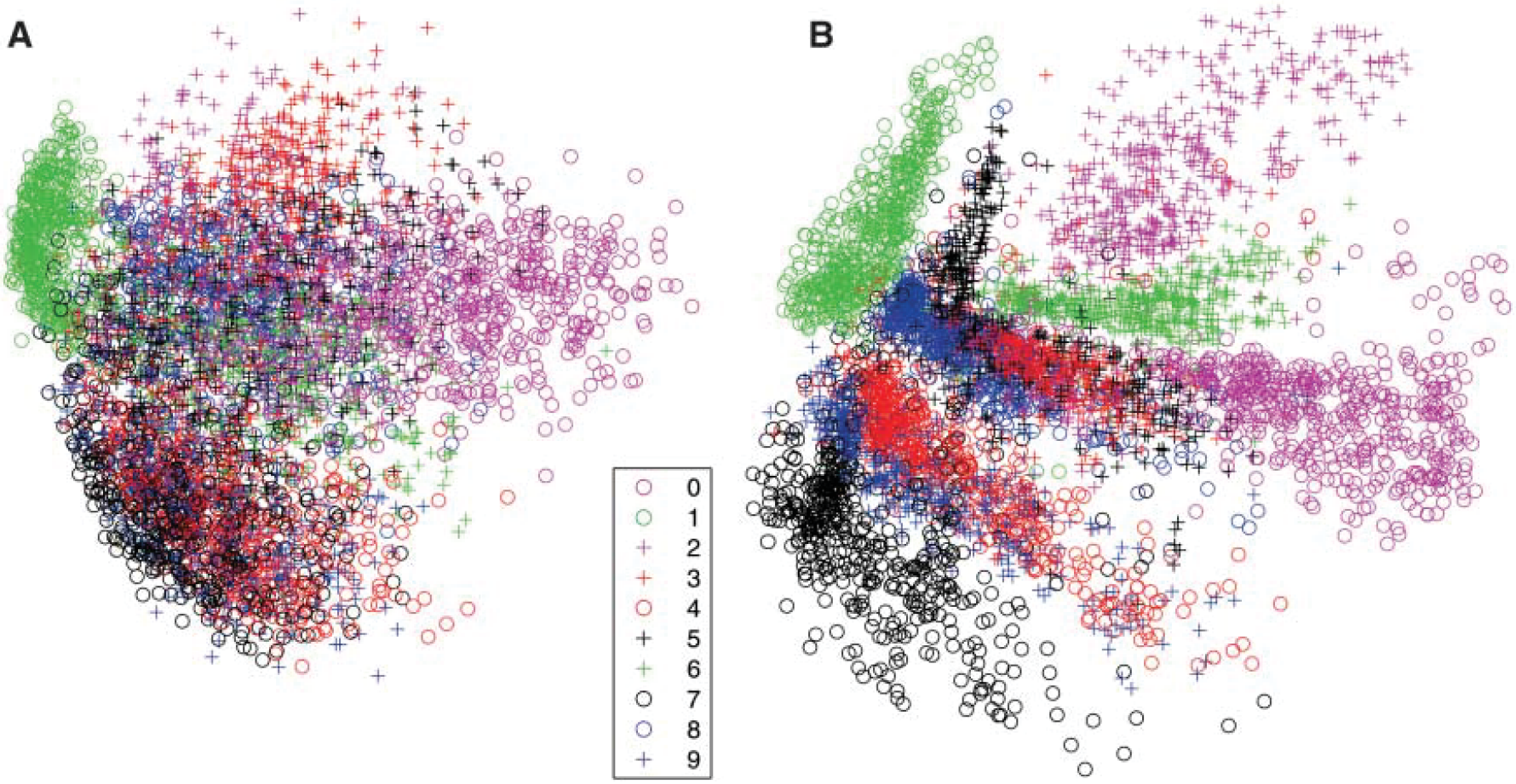
- Standard BP is used for the final tuning.



Fine-tuning

Data compression of MNIST handwritten numeral images

- Nine-layer network (784(= 28 × 28) – 1000 – 500 – 250 – 2)



BP

Deep learning

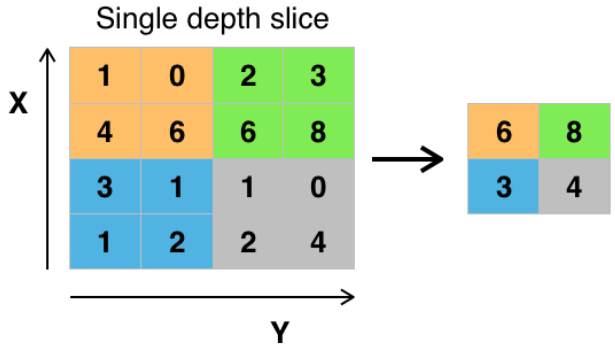
3.7 Convolutional Neural Network (CNN)

CNN consists of

- Convolutional layer
  - A data in a region is mapped to a node of the upper layer similarly to convolution so the weights are sift invariant.
  - Several types of convolutions are applied.
  - Consider 1D case. Let $w_{i,j}$ be the weight of the t-th type from j-th node to i-th node of the upper layer. Then for any k, we have

$$w_{(i-k),(j-k),t} = w_{i,j,t}$$

- Pooling layer : the amount of date is reduced.

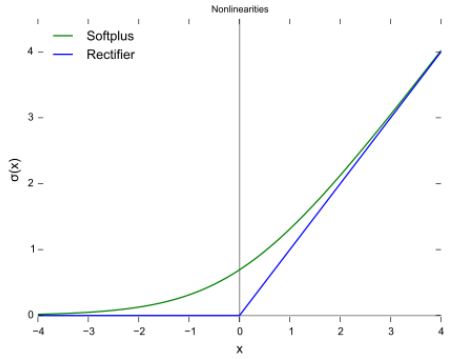


By Aphex34 - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=45673581

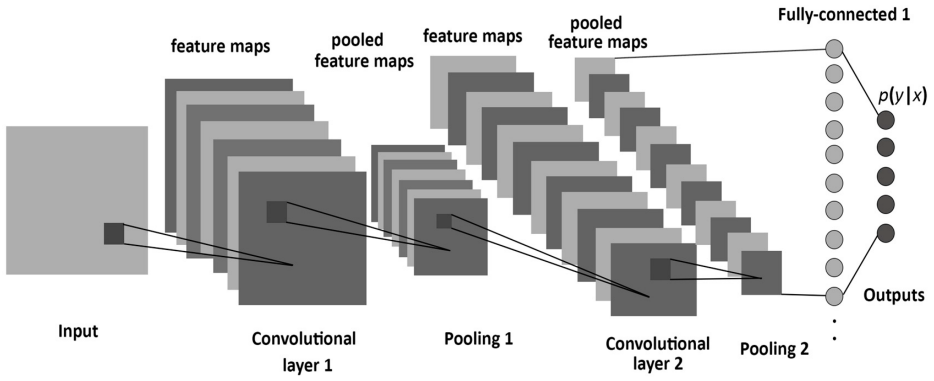
- Fully connected layer (for the last layer)

Activation function $f(x)$

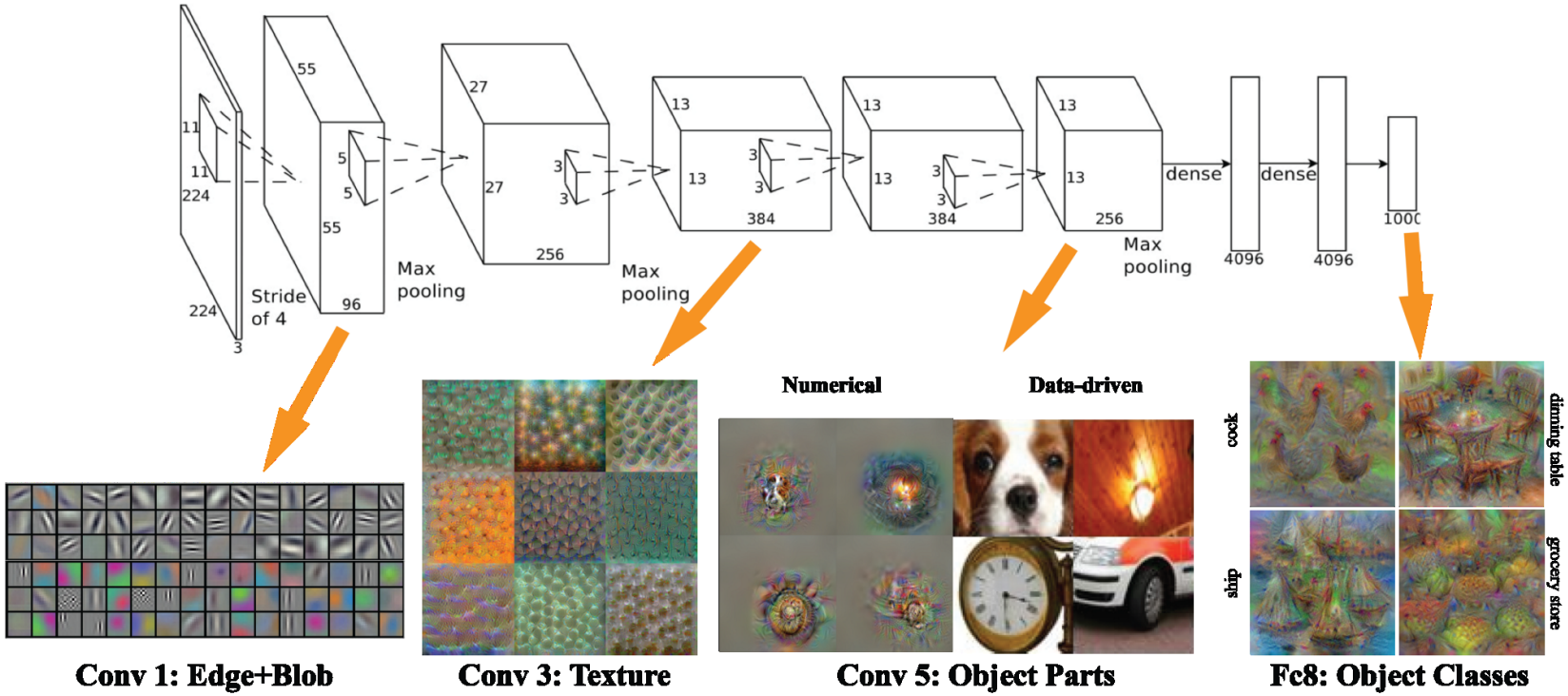
- ReLU : rectified linear units (= hinge loss) : $\max(0, x)$



- Sigmoid function : $1/(1 + e^{-x})$
- Softmax : $e^{y_i} / \sum_j e^{y_j}$: activation values in other units are used.



S. Albelwi and A. Mahmood, "A Framework for Designing the Architectures of Deep Convolutional Neural Networks," Entropy, 2017.

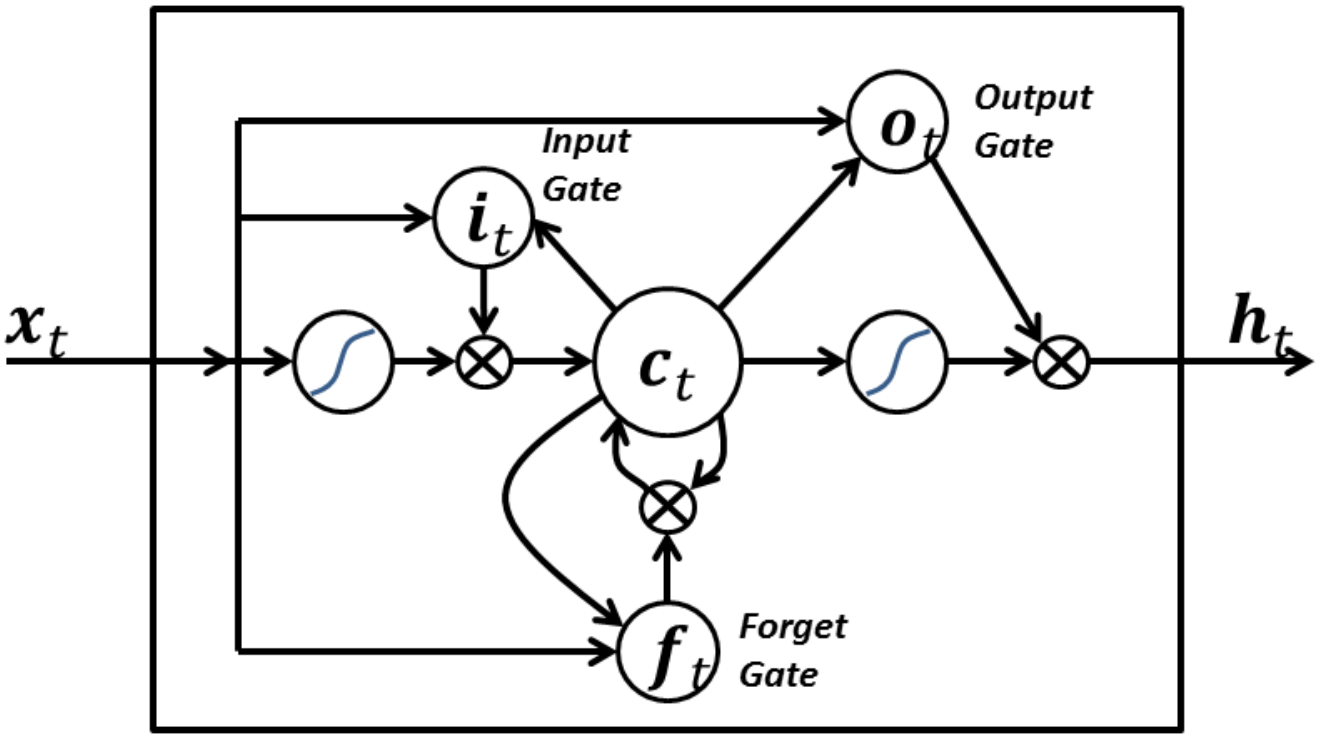


Features in CNN

Daniel Jeffries, "Learning AI if You Suck at Math - P5 -ŁDeep Learning and Convolutional Neural Nets in Plain English!,"
`https://hackernoon.com/learning-ai-if-you-suck-at-math-p5-deep-learning-and-convolutional-neural-nets-in-plain-english-cda79679bbe3`

3.8 Long Short Term Memory (LSTM)

- A type of recurrent networks.
- S. Hochreiter and J. Schmidhuber, “Long Short-term Memory,”Neural Computation 1997.
- Alex Graves, et al. ”Speech recognition with deep recurrent neural networks,” ICASSP 2013.



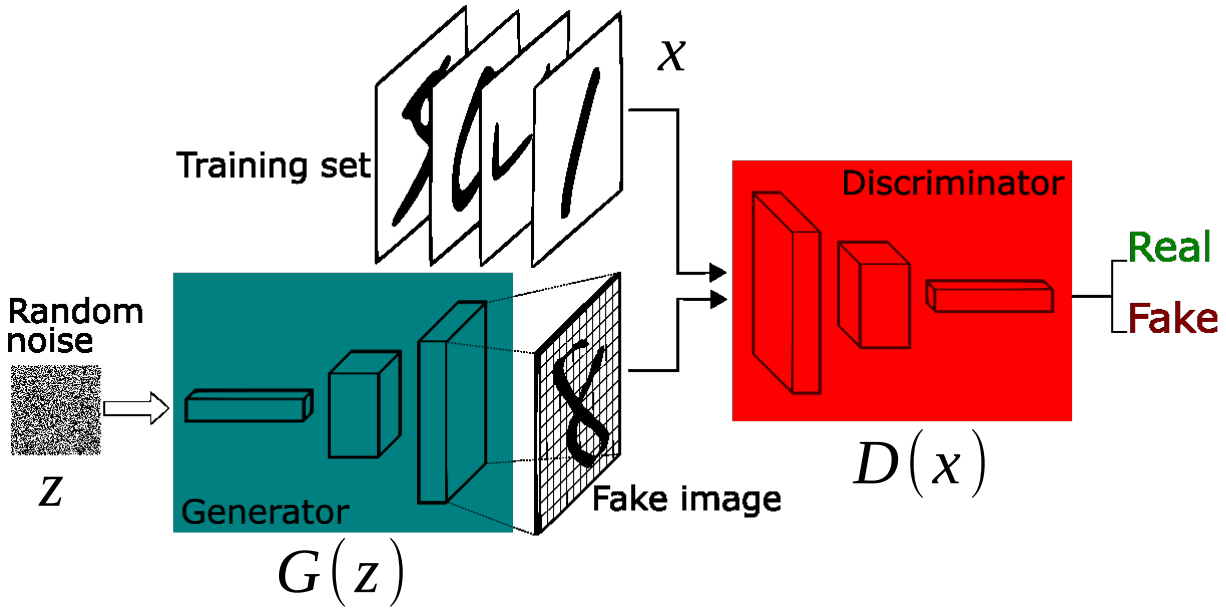
- $\boldsymbol{x}_t$: input vector
- $\boldsymbol{c}_t$: state vector
- $\boldsymbol{f}_t$: forget gate vector
- $\boldsymbol{i}_t$: input gate vector
- $\boldsymbol{o}_t$: output gate Vector
- $\boldsymbol{h}_t$: output vector
- $\boldsymbol{W}_f, \boldsymbol{W}_i, \boldsymbol{W}_o, \boldsymbol{W}_c, \boldsymbol{U}_f, \boldsymbol{U}_i, \boldsymbol{U}_o, \boldsymbol{U}_c, \boldsymbol{b}_f, \boldsymbol{b}_i, \boldsymbol{b}_o, \boldsymbol{b}_c$: parameters
- σ_h : activate function for state
- σ_g : activate function for state, input gate, and output gate
- σ_h : activate function for output

Update rule: ($\circ$: Hadamard product (element-wise product))

- $\boldsymbol{f}_t = \sigma_g(\boldsymbol{W}_f \boldsymbol{x}_t + \boldsymbol{U}_f \boldsymbol{c}_{t-1} + \boldsymbol{b}_f)$
- $\boldsymbol{i}_t = \sigma_g(\boldsymbol{W}_i \boldsymbol{x}_t + \boldsymbol{U}_i \boldsymbol{c}_{t-1} + \boldsymbol{b}_i)$
- $\boldsymbol{o}_t = \sigma_g(\boldsymbol{W}_o \boldsymbol{x}_t + \boldsymbol{U}_o \boldsymbol{c}_{t-1} + \boldsymbol{b}_o)$
- $\boldsymbol{c}_t = \boldsymbol{f}_t \circ \boldsymbol{c}_{t-1} + \boldsymbol{i}_t \circ \sigma_c(\boldsymbol{W}_c \boldsymbol{x}_t + \boldsymbol{U}_c \boldsymbol{c}_{t-1} + \boldsymbol{b}_c)$
- $\boldsymbol{h}_t = \boldsymbol{o}_t \circ \sigma_h(\boldsymbol{c}_t)$

3.9 Generative Adversarial Network (GAN)

- Ian J. Goodfellow, et al. "Generative Adversarial Nets," NIPS, 2014.
- Two types of networks.
  - Discriminative network : $D(\boldsymbol{x})$
    The probability that $\boldsymbol{x}$ came from the data rather than Generator.
  - Generative network : $\boldsymbol{G}(z)$:
    The generated pattern from a noize z.

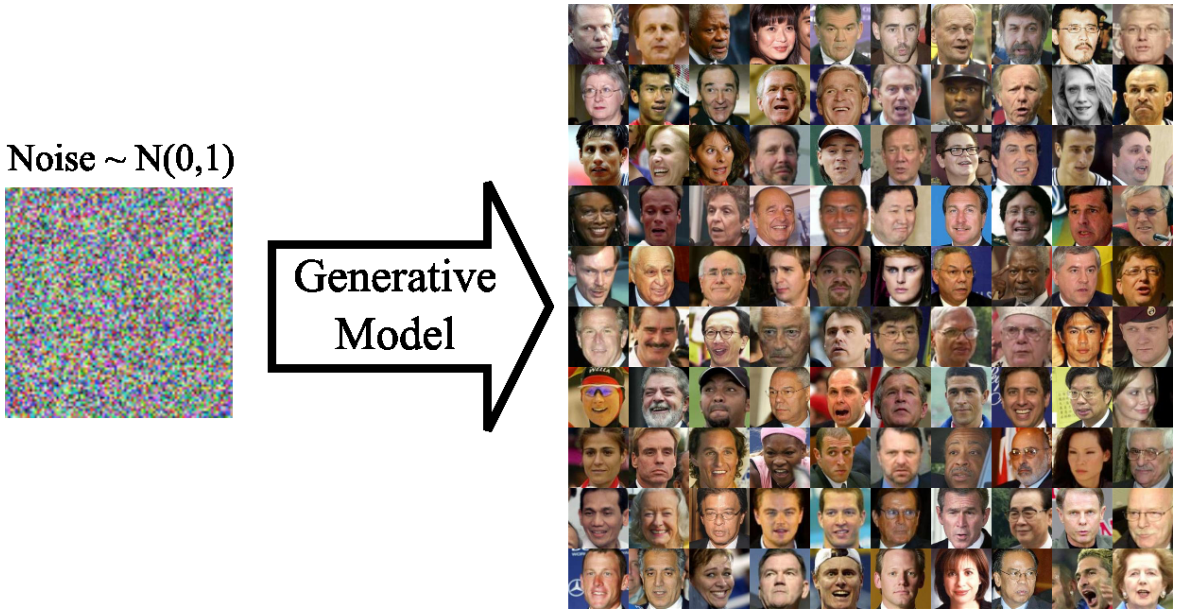


Modified from https://skymind.ai/wiki/generative-adversarial-network-gan

- Criterion:

$$\min_{\boldsymbol{G}} \max_{D} E_{\boldsymbol{x} \simeq p_{\text{data}}(\boldsymbol{x})}[\log D(x)] + E_{\boldsymbol{z} \simeq p_{\boldsymbol{z}}(z)}[\log(1 - D(\boldsymbol{G}(z)))].$$

  - $D(\boldsymbol{x})$ to discriminate precisely.
  - $\boldsymbol{G}(z)$ to generate patterns that cause misclassification of $D(\boldsymbol{x})$.
  - Its original purpose is precise discrimination.
  - But the trained $\boldsymbol{G}(z)$ is used as a pattern generator.



Hunter Heidenreich, "What is a Generative Adversarial Network?,"
URL:https://towardsdatascience.com/what-is-a-generative-adversarial-network-76898dd7ea65

4 Conclusion

- AI stands on many fields
  - Linear algebra
  - Statistical learning
  - Logic inference
  - Image and signal processing
  - Natural language processing
  - Parallel computing.
- Many applications
  - Pattern recognition
    - Character recognition
    - Computer vision (face detection and recognition))
  - Decision making (diagnostics and AlphaGO)
  - Natural language processing (for communication)
  - Big data (marketing and data scientists)
  - Autonomous vehicle and robot

- Week points
  - Logical thinking and integration of information.
  - AlphaGo : strong in earlier and last stages but weak in middle.
  - For autonomous vehicle, a range finder (can measure distance) is necessary although a human can drive a car with only an eye.
- Future
  - Precise recognition
  - Practical autonomous vehicle and robot
  - Language translation
  - Artificial general intelligence.
    Instead of human, machines do everything.

Thank you very much for listening.