Introduction to Neural Networks

Biological and artificial networks

The Brain:

consists of 10^{12} nerve cells (also called neurons) interconnected through about 10^{15} *synaptic junctions* to form millions of neural networks. Hundreds specialized cortical areas are formed based on these networks for different information processing tasks;

The Neuron:

- *Dendrites:*

  conduct impulses (inputs) from other cells toward the cell body;

- *Axon:*

  conducts impulses (output) away from the cell body to other cells;

- *Synapse:*

  the point at which impulses pass from one cell to another;

Model of neuron:

A neuron is modeled mathematically as the following:

- **activation – the net input signal:**

  The net input to the ith node is a weighted sum of all inputs:

  $$net_i = \sum_j w_{ij} x_j$$

  where o_j is the input signal from the jth node, w_{ij} is the synaptic connectivity between the jth node and the ith node.

  $$w_{ij} = \begin{cases} > 0 & excitatory \;\; input \\ < 0 & inhibitory \;\; input \\ = 0 & no \;\; connection \end{cases}$$

- **the output signal:**

  The output of the ith node is a function of the net input:

  $$y_i = f(net_i) = f(\sum_j w_{ij} x_j)$$

  where the function f can be a sigmoid, a threshold function, or an identical function.

Model of neural network:

The artificial neural network is a group of neurons organized in several layers:

- *input layer:* receives inputs from sources external to the network;
- *output layer:* generates outputs to the external world.
- *hidden layer(s):* layers in between of the input and output layers, not visible from outside the network.
- *learning laws:* mathematical rules for modifying the weights of a network iteratively according the inputs (and outputs if the learning is supervised).

Computation

Assume a simple 2-layer network has n input nodes receiving stimuli (input patterns) $X = [x_1, \cdots, x_n]^T$ and m output nodes generating responses (output patterns) $Y = [y_1, \cdots, y_m]^T$. Also assume the output function f is an identical function. Then the output of the ith output node is:

$$y_i = \sum_{j=1}^{n} w_{ij} x_j = W_i^T X \qquad (i = 1, \cdots, m)$$

where

$$W_i \triangleq [w_{i1}, \cdots, w_{in}]^T$$

is a weight vector formed by all the weights leading from the n input nodes to the ith output node.

The output Y can also be represented in the matrix form:

$$Y = WX$$

where W is the weight matrix

$$W = \begin{bmatrix} W_1^T \\ \cdots \\ \cdots \\ W_m^T \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} & \cdots & w_{1n} \\ w_{21} & w_{22} & \cdots & w_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ w_{m1} & w_{m2} & \cdots & w_{mn} \end{bmatrix}_{m \times n}$$

Paradigms of Learning

1. **Pattern Associator**

   Training (supervised):

   A set of pairs of patterns $\{(X_i, Y_i), i = 1, 2, \cdots, L\}$ is repeatedly presented to the NN which then learns to establish the relationship (association) between two sets of patterns:

   $$f : X \in R^n \rightarrow Y \in R^m$$

   i.e., the associtive relationship between the two patterns of the pairs is stored in the NN.

   Testing:

   When one pattern in a pair is presented, the NN will produce the other.

2. **Autoassociator**

   Training:

   A set of patterns $\{X_i, i = 1, 2, \cdots, L\}$ is repeatedly presented to the NN which learns and remembers them, i.e., the patterns are stored in the NN.

   Testing:

   When part of a pattern, or a similar pattern (pattern with noise) is presented to the NN, the complete original pattern will be retrieved through pattern completion by the NN.

3. **Pattern Classifier**

   This is a variation of the first type of NN. The input patterns are classified by the NN into a set of classes (represented by names or any other symbols). This can be considered as a special type of mapping:

   $$f : X \in R^n \rightarrow \omega_i \in \{\omega_1, \cdots, \omega_m\}$$

4. **Regularity Detector**

   The NN discovers the regularity in the inputs so that patterns of various types can be automatically detected and classified into a set of classes. This is an unsupervised learning process.

Examples of association:

- X_1: Evolution $\rightarrow$ Y_1: Darwin
- X_2: Einstein $\rightarrow$ Y_2: $E = mc^2$
- X_3: sounding bell $\rightarrow$ Y_3: food

Hebb's Learning

D. Hebb speculated in 1949 that

> "When neuron A repeatedly and persistently takes part in exciting neuron B, the synaptic connection from A to B will be strengthened."

So a Hebbian network can be used as an associator which will establish the association between two sets of patterns $\{X_i, \quad i = 1, \cdots, L\}$ and $\{Y_i, \quad j = 1, \cdots, L\}$.

An explanation of the classical conditioning (Pavlov, 1927):

- F: sight of food – unconditioned stimulus
- B: sound of bell – conditioned stimulus
- S: Salivation – response

$$(F \to S) \Rightarrow (F \cap B \to S) \Rightarrow (B \to S)$$

Synaptic connections between pattern B and pattern S are strengthened as both are repeatedly excited simultaneously.

The structure

The Hebbian network model has an n-node input layer $X = [x_1, \cdots, x_n]^T$ and an m-node output layer $Y = [y_1, \cdots, y_m]^T$. Each output node is connected to all input nodes:

$$y_i = \sum_{j=1}^{n} w_{ij} x_j \qquad (i = 1, \cdots, m)$$

or in matrix form, we have

$$Y = WX$$

The learning law

$$w_{ij}^{new} = w_{ij}^{old} + \eta x_j y_i \quad (i = 1, \cdots, n,\ j = 1, \cdots, m)$$

or in matrix form:

$$W^{new} = W^{old} + \eta Y X^T$$

Here η is the learning rate, a parameter controlling how fast the weights get modified.

Training

If we assume $w_{ij} = 0$ initially, $\eta = 1$ and a set of L pairs of patterns $\{(X_k, Y_k),\ \ k = 1, \cdots, L\}$ are presented repeatedly during training, we have

$$w_{ij} = \sum_{k=1}^{L} x_j^{(k)} y_i^{(k)} \quad (i, j = 1, \cdots, n, \ \ j = 1, \cdots, m)$$

or in matrix form

$$W = \sum_{k=1}^{L} Y_k X_k^T$$

Classification

When presented with one of the patterns X_l, the network will produce the output:

$$Y = W X_l = (\sum_{k=1}^{L} Y_k X_k^T)\ X_l = Y_l (X_l^T X_l) + \sum_{k \neq l} Y_k (X_k^T X_l)$$

We first consider the ideal case where

- $X's$ are orthogonal to each other:

  $$X_i^T X_j = |X_i||X_j| cos\phi = \delta_{ij} = \left\{ \begin{array}{ll} 1 & i = j \\ 0 & i \neq j \end{array} \right.$$

  where $|X_i|$ is the length of vector X_i and ϕ is the angle between vectors X_i and X_j.

- The number of pattern pairs is smaller than the number of input nodes:

$$L \leq n$$

The first condition implies that the L input patterns are totally unrelated to each other. Intuitively, how much two vectors are related to each other can be measured by the angle ϕ between them. If ϕ is close to 0 the two vectors are closely related, because their elements are very similar to each other so that they almost coincide. When negative values are allowed for the elements and ϕ is close to 180 degrees, the vectors are also highly related because their elements are the opposite of each other. In either case, the $cos\phi$ is close to 1 and the inner product is maximinzed indicating that the two vectors are either positively or negatively related to each other. On the other hand, if the two vectors are perpendicular to each other ($\phi = \pi/2$, $cos\phi = 0$), the inner product of the two vectors is very small or even zero indicating the elements of the a vector are irrelevant to those of the other vector.

How much 2 patterns X and Y are related to each other can also be measured quantitatively, we define *correlation coefficient* as:

$$r_{xy} \triangleq \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2}\sqrt{\sum_i y_i^2}}$$

r_{xy} is the inner product of 2 normalized vectors representing 2 patterns in the n-D feature space, and can take the following values

$0 < r_{xy} \leq 1$	if X and Y are positively correlated ($r_{xy} = 1$ iff $X = Y$)
$r_{xy} = 0$	if X and Y are not correlated
$-1 < r_{xy} \leq 0$	if X and Y are negatively correlated ($r_{xy} = -1$ iff $X = -Y$)

In other words, if X and Y are totally unrelated, then $r_{xy} = 0$ and they are orthogonal to each other.

The second condition implies that the capacity of the network (n input nodes) is large enough for representing L different patterns. This can be translated into mathematical language that there can be no more than n orthogonal vectors in an n-D space.

If these conditions are true, then the response of the network to X_l will be Y_l because

$$X_k^T X_l = 0 \quad for \;\; all \;\; k \neq l$$

In other words, a one-to-one correspondence relationship between X_i and Y_i has been established for all $i = 1, \cdots, L$. In non-ideal case, the summation term (called cross talk) is non-zero and we have an error $Y - Y_l \neq 0$.

Although two patterns $X = B$ and $Y = S$ don't have causal relationship (as S is caused by another pattern F), but if they always appear simultaneously, an association relationship can be established in the Hebbian network.

Hopfield Network

Hopfield network (Hopfield 1982) is *recurrent* network composed of a set of n nodes and behaves as an associator (content addressable memory) with a set of L patterns $\{Y_k, \quad k = 1, \cdots, L\}$ stored in it. Here each pattern Y_k is an n dimensional vector:

$$Y_k = [y_1^{(k)}, \cdots, y_n^{(k)}]^T \qquad y_i \in \{-1,\ 1\}$$

Presented a new pattern X, the network responds by iteratively updating its output

$$X = [x_1, \cdots, x_n]^T \qquad x_i \in \{-1,\ 1\}$$

until finally convergence is reached when one of the stored patterns which most closely resembles X is produced as the output.

The weights and iterative computation

The weight matrix of the network are obtained from the L patterns to be stored

$$W = \frac{1}{n} \sum_{k=0}^{L} X_k X_k^T$$

Equivalently, the weight connecting node i and node j is

$$w_{ij} = \frac{1}{n} \sum_{k=1}^{L} x_i^{(k)} x_j^{(k)} = w_{ji}$$

And no self-connection exists

$$w_{ii} = 0 \qquad (i = 1, \cdots, n)$$

The way to obtain the weights is very similar to Hebbian learning, except here the two patterns X_k and Y_k in each pair are the same.

After an input pattern, an n dimensional vector X, is presented to the n nodes of the network, the outputs x_i are updated *iteratively* and *asynchronously*. Only one of the n nodes randomly selected is updated at a time:

$$x_i' = sgn(\sum_{j=1}^{n} w_{ij} x_j)$$

where x_i' represents the new output of the ith node, and

$$sgn(x) \triangleq \begin{cases} +1 & x \geq 0 \\ -1 & x < 0 \end{cases}$$

We will show later that the iteration will always converge to produce one of the stored patterns as the output.

Energy function

The *energy* of a pair of nodes i and j is defined as

$$e_{ij} \stackrel{\triangle}{=} -w_{ij}x_ix_j$$

and the interaction between these two nodes can be summarized by this table

x_i	x_j	$w_{ij} > 0$	$w_{ij} < 0$
-1	-1	$e_{ij} < 0$	$e_{ij} > 0$
-1	1	$e_{ij} > 0$	$e_{ij} < 0$
1	-1	$e_{ij} > 0$	$e_{ij} < 0$
1	1	$e_{ij} < 0$	$e_{ij} > 0$

We see that whenever the two nodes are reinforcing each other's state (e.g., in the last row when $w_{ij} > 0$, $x_i = 1$ will help keep x_j to stay at state 1 and vice versa), the energy is negative; and whenever the two nodes are trying to change each other's state (e.g., in the same row when $w_{ij} < 0$, $x_i = 1$ will tend to reverse x_j from 1 to -1 and vice versa), the energy is positive. In other words, low energy level of e_{ij} corresponds to a stable interaction between x_i and x_j, and high energy level corresponds to an unstable interaction.

The *energy function* $E(X)$ of all n nodes in the network is defined as the sum of all the pair-wise energies:

$$\begin{aligned} E(X) &\stackrel{\triangle}{=} \sum_{i>j} e_{ij} = -\sum_{i>j} w_{ij}x_ix_j \\ &\stackrel{*}{=} -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} w_{ij}x_ix_j \end{aligned}$$

(* $w_{ij} = w_{ji}$)

Againg, lower energy function level corresponds to more stable condition of the network, and vice versa.

Convergence of iterations

Here we show that the total energy $E(X)$ always decreases whenever the state of any node changes. Assume x_k has just been updated, i.e., $x'_k \neq x_k$ while all others remain the same $x'_{l\neq k} = x_{l\neq k}$. The energy before x_k changes state is

$$\begin{aligned} E(X) &= -\frac{1}{2}[\sum_{i\neq k}\sum_{j\neq k} w_{ij}x_ix_j + \sum_i w_{ik}x_ix_k + \sum_j w_{kj}x_kx_j] \\ &= -\frac{1}{2}\sum_{i\neq k}\sum_{j\neq k} w_{ij}x_ix_j - \sum_i w_{ik}x_ix_k \end{aligned}$$

and the energy after x_k changes state is

$$E'(X) = -\frac{1}{2}\sum_{i\neq k}\sum_{j\neq k} w_{ij}x_ix_j - \sum_i w_{ik}x_ix'_k$$

The energy difference is

$$\triangle E \stackrel{\triangle}{=} E'(X) - E(X) = -(x'_k - x_k)\sum_i w_{ik}x_i$$

Consider two cases:

- case 1: if $\sum_i w_{ik}x_i \geq 0$, then $x'_k = 1$, we have $(x'_k - x_k) > 0$ and $\triangle E \leq 0$.
- case 2: if $\sum_i w_{ik}x_i < 0$, then $x'_k = -1$, we have $(x'_k - x_k) < 0$ and $\triangle E \leq 0$.

As $\triangle E(X) \leq 0$ is always true throughout the iteration, (and also note the magnitude of $E(X)$ is finite), we conclude that the energy function $E(X)$ will eventually reach one of the minima of the "energy landscape" and, therefore, the iteration will always converge.

Retrieval of stored patterns

Here we show the L pre-stored patterns correspond to the minima of the energy function. First recall the weights of the network are obtained by Hebbian learning:

$$w_{ij} = \frac{1}{n} \sum_{k=1}^{L} y_i^{(k)} y_j^{(k)}$$

the energy function $E(X)$ can now be written as

$$\begin{aligned} E(X) &= -\frac{1}{2n} \sum_i \sum_j [\sum_{k=1}^{L} y_i^{(k)} y_j^{(k)}] x_i x_j = -\frac{1}{2n} \sum_{k=1}^{L} [\sum_i \sum_j y_i^{(k)} x_i y_j^{(k)} x_j] \\ &= -\frac{1}{2n} \sum_{k=1}^{L} [(\sum_i y_i^{(k)} x_i)(\sum_j y_j^{(k)} x_j)] = -\frac{1}{2n} \sum_{k=1}^{L} [(\sum_i y_i^{(k)} x_i)^2] \end{aligned}$$

If X is different from any of the stored patterns, all L terms of the summation will contribute to the sum only moderately. However, if X is the same as any of the stored patterns, their inner product reaches maximum, and thus causing the total energy to be minimized to reach one of the minima. In other words, the patterns stored in the net correspond to the local minima of the energy function. i.e., these patterns become attractors.

Note that it is possible to have other local minima, called *spurious states*, which do not represent any of the stored patterns, i.e., the associative memory is not perfect.

Perceptron Learning

A perceptron network (F. Rosenblatt, 1957) also has 2 layers as in the previous Hebb's network, except the learning law is different. We first consider one of the output nodes:

In addition to the n inputs $x_i \quad (i = 1, \cdots, n)$, the output node also receives an input y representing the desired output for the current input used for training. The node generates its output y':

$$y' = \begin{cases} 1 & if \;\; X^T W \geq 0 \\ 0 & if \;\; X^T W < 0 \end{cases}$$

Here both y and y' are binary:

$$y \in \{0, 1\}, \qquad y' \in \{0, 1\}$$

and this simple network is a 2-class classifier.

The learning law

$$W^{new} = W^{old} + \triangle W = W^{old} + \eta (y - y') X$$

where

$$\triangle W \stackrel{\triangle}{=} \eta (y - y') X = \eta \delta X$$

Here $\delta \stackrel{\triangle}{=} y - y'$ represents the error, the difference between the actual output and the desired output. And the learning law is called the δ-rule.

How the learning law works can be shown by the following 2 cases:

- If $X^T W^{old} > 0$ and therefore $y' = 1$, but $y = 0$, then $\delta = -1$ and

  $$W^{new} = W^{old} - \eta X$$

  so that next time when the same X is presented, y' will be more likely to be 0 as desired because:

  $$X^T W^{new} = X^T W^{old} - \eta X^T X < X^T W^{old}$$

- If $X^T W^{old} < 0$ and therefore $y' = 0$, but $y = 1$, then $\delta = 1$ and

  $$W^{new} = W^{old} + \eta X$$

  so that next time when the same X is presented, y' will be more likely to be 1 as desired because:

  $$X^T W^{new} = X^T W^{old} + \eta X^T X > X^T W^{old}$$

Perceptron convergence theorem:

> If ω_1 and ω_2 are two linearly separable clusters of patterns, then a perceptron will develop a W in finite number of training trials to classify them.

The condition of linear separability is obvious as the output node of the perceptron network is only capable of linear operations. When multiple output nodes are available, the network can associate multiple classes to different binary output patterns.

Constraint of linear classifier

The use of linear classifier is limited, as shown by the following example. Consider a 2-class problem in a 2D feature space as shown:

Back Propagation Network (BPN)

The basic idea

This is a typical supervised learning network, composed of input, hidden and output layers with n, l, m nodes, respectively. Each node is fully connected to all nodes in the previous layer.

The two phase classification:

- **Training:** The training data are a set of pattern pairs $\{(X_p, Y_p), p = 1, 2, ..., L\}$, where X_p's are the input patterns and Y_p's are the desired output patterns — the correct responses.

  In training phase the following two steps are repeated for all input patterns presented to the input layer in random order:

  - Present an input pattern X to the input layer and generate the output Y' at the output layer, which is compared with the desired output Y to find the error $E = Y - Y'$;
  - Propagate the error backwards from the output layer to input layer to modify the weights so that the error, as a function of the weights, can be minimized.

  The weights are updated iteratively until eventually every input pattern presented to the input layer will cause the correct response pattern to appear at the output layer.

- **Classification** The network responds to various input patterns by generating the patterns associated to the input.

Derivation of training

The following is the 2-step training process for a particular pattern pair (X_p, Y_p).

- Define error energy E_p at the output layer:

$$E_p = \frac{1}{2}\sum_{i=1}^{m}(y_{pi} - y'_{pi})^2 = \frac{1}{2}\sum_{i=1}^{m}(\delta^o_{pi})^2$$

  where

$$\delta^o_{pi} \triangleq y_{pi} - y'_{pi}$$

  y_{pi} is the desired output, and y'_{pi} is the actual output

$$y'_{pi} = f(net_{pi}) = f(\sum_{j=1}^{l} w^o_{ij} z_{pj} + T_i)$$

- Find gradient of E_p in output weight space $\{w^o_{ij}\}$:

$$\frac{\partial E_p}{\partial w^o_{ij}} = \frac{\partial E_p}{\partial y'_{pi}} \frac{\partial y'_{pi}}{\partial net_{pi}} \frac{\partial net_{pi}}{\partial w^o_{ij}} = -\delta^o_{pi}\; f'(net_{pi})\; z_{pj}$$

- Update w^o_{ij} to minimize E_p with gradient descent method:

$$w^{o(new)}_{ij} = w^{o(old)}_{ij} + \Delta w^o_{ij} = w^{o(old)}_{ij} - \eta\frac{\partial E_p}{\partial w^o_{ij}}$$

  where

$$\Delta w^o_{ij} \triangleq -\eta\frac{\partial E_p}{\partial w^o_{ij}}$$

  and η is the learning rate.

- Relate E_p to the hidden layer weights w^h_{jk}

$$\begin{aligned} E_p &= \frac{1}{2}\sum_{i=1}^{m}(y_{pi} - y'_{pi})^2 \\ &= \frac{1}{2}\sum_{i=1}^{m}(y_{pi} - f(\sum_{j=1}^{l} w^o_{ij} z_{pj} + T_i))^2 \end{aligned}$$

$$
\begin{aligned}
&= \frac{1}{2}\sum_{i=1}^{m}(y_{pi} - f(\sum_{j=1}^{l} w^o_{ij} f(net_{pj}) + T_i))^2 \\
&= \frac{1}{2}\sum_{i=1}^{m}(y_{pi} - f(\sum_{j=1}^{l} w^o_{ij} f(\sum_{k=1}^{n} w^h_{jk} x_{pk} + T_j) + T_i))^2
\end{aligned}
$$

- Find gradient of E_p in hidden weight space $\{w^h_{jk}\}$:

$$
\begin{aligned}
\frac{\partial E_p}{\partial w^h_{jk}} &= -\sum_{i=1}^{m}(y_{pi} - y'_{pi})\frac{\partial y'_{pi}}{\partial net_{pi}}\frac{\partial net_{pi}}{\partial z_{pj}}\frac{\partial z_{pj}}{\partial net_{pj}}\frac{\partial net_{pj}}{\partial w^h_{jk}} \\
&= -\sum_{i=1}^{m}\delta^o_{pi} f'(net_{pi}) w^o_{ij}\ f'(net_{pj}) x_{pk} \\
&= -f'(net_{pj}) x_{pk} \delta^h_{pj}
\end{aligned}
$$

where

$$
\delta^h_{pj} \triangleq \sum_{i=1}^{m}\delta^o_{pi} f'(net_{pi}) w^o_{ij}
$$

- Update w^h_{ij} to minimize E_p with gradient descent method

$$
w^{h(new)}_{jk} = w^{h(old)}_{jk} + \Delta w^h_{jk} = w^{h(old)}_{jk} - \eta\frac{\partial E_p}{\partial w_{jk}}
$$

where

$$
\Delta w^h_{jk} \triangleq -\eta\frac{\partial E_p}{\partial w^h_{ji}} = \eta f'(net_{pj}) x_k \delta^h_{pj}
$$

Summary of BP training

The following is for one training pattern pair $\{X_p, Y_p\}$.

1. Apply $X_p = (x_{p1}, x_{p2}, ..., x_{pn})^t$ to the input nodes;
2. Compute net input to hidden nodes
$$net_{pj} = \sum_{k=1}^{n} w_{jk}^h x_{pk} + T_j$$
3. Compute output from hidden nodes
$$z_{pj} = f(net_{pj}) \qquad (j = 1, \cdots, l)$$
4. Compute net input to output nodes
$$net_{pi} = \sum_{j=1}^{l} w_{ij}^o z_{pj} + T_i \qquad (i = 1, \cdots, m)$$
5. Compute output from output nodes
$$y'_{pi} = f(net_{pi}) \qquad (i = 1, \cdots, m)$$
6. Find error terms for all output nodes (not quite the same as defined previously)
$$\delta_{pi}^o = f'(net_{pi})(y_{pi} - y'_{pi}) \qquad (i = 1, \cdots, m)$$
where $Y_p = (y_{p1}, y_{p2}, ..., y_{pm})^t$ is the desired output for X_p.
7. Find error terms for all hidden nodes (not quite the same as defined previously)
$$\delta_{pj}^h = f'(net_{pj}) \sum_{i=1}^{m} \delta_{pi}^o w_{ij}^o \qquad (j = 1, \cdots, l)$$
8. Update weights to output nodes
$$w_{ij}^o \leftarrow w_{ij}^o + \eta \delta_{pi}^o z_{pj} \qquad (i = 1, \cdots, m, \quad j = 1, \cdots, l)$$

9. Update weights to hidden nodes

$$w_{jk}^h \leftarrow w_{jk}^h + \eta \delta_{pj}^h x_{pk} \qquad (j = 1, \cdots, l, \quad k = 1, \cdots, n)$$

10. Compute

$$E_p = \frac{1}{2} \sum_{i=1}^{m} (y_{pi} - y'_{pi})^2$$

When this error is acceptably small for all of the training pattern pairs, training can be discontinued.

Competitive Learning Network

This is a typical unsupervised learning network, similar to the statistical clustering analysis methods (k-means, Isodata). The simplest form of competitive learning network has 2 layers, the input layer composed of n nodes to which an input pattern $X = [x_1, \cdots, x_n]^T$ is presented and the output layer composed of m nodes $y_i, \quad (i = 1, \cdots, m)$. The net input (activation) of the ith output node is

$$net_i = \sum_{j=1}^{n} w_{ij} x_j = W_i^T X$$

The outputs of the network are determined by a "winner-take-all" competition such that only the output node receiving the maximal net input will output 1 while all others output 0:

$$y_k = \begin{cases} 1 & iff \;\; net_k = max\{net_i, \; i = 1, \cdots, m\} \\ 0 & else \end{cases}$$

For simplicity, we assume

- the input patterns are binary: $x_i = 0 \;\; or \;\; 1$
- all weights are positive

  $$w_{ij} > 0$$

  and the weight vector W_i is normalized in a certain sense:

  $$\sum_j w_{ij} = 1$$

The net input net_i can be considered as the inner product of two vectors W_i and X:

$$net_i = \sum_j w_{ij} x_j = W_i^T X = |W_i| \, |X| \, cos\, \theta$$

which is closely related to the difference between the two vectors:

$$|W_i - X|^2 = |W_i|^2 + |X|^2 - 2|W_i| \, |X| \, cos\, \theta$$

where θ is the angle between the two vectors.

We see that the weight vector W_k of the winning node is closest to the current input pattern vector X in the n-D feature space (with smallest angle θ and therefore smallest distance $|W_i - X|$).

The competitive learning law is

$$W_i^{new} = W_i^{old} + u_i \, \eta \, (X - W_i^{old}) = W_i^{old} + u_i \, \triangle W_i$$

where

$$u_i \triangleq \begin{cases} 1 & iff \;\; y_i \;\; is \;\; the \;\; winner \\ 0 & else \end{cases}$$

and

$$\triangle W_i \triangleq \eta (X - W_i^{old})$$

According to this learning law, only the weights of the winning node will get updated:

$$W_k^{new} = W_k^{old} + \triangle W = (1 - \eta) W_k^{old} + \eta X$$

while all other weights stay the same:

$$W_i^{new} = W_i^{old} \quad (for \;\; all \;\; i \neq k)$$

It can be seen that the effect of this learning process is to pull the weight vector W_k closest to the current input pattern X even closer to X (note that W_k^{new} is a point between W_k^{old} and X).

Every time a different pattern X is presented to the input layer, a certain output node will win and be drawn closer to X. After many patterns have been repeatedly presented to the input layer, each group of similar patterns — a cluster of points in the n-D feature space — will draw a certain weight vector to its center, and, as a result, the corresponding node will always win and produce a non-zero output whenever a member of the cluster is presented to the input. In other words, each group of similar patterns is represented by a certain output node.

Vector Support Machines (SVM

Linear separation of a feature space

This is a variation of the perceptron learning algorithm. Consider a hyper plane in an n-dimensional (n-D) feature space:

$$f(X) = X^T W + b = \sum_{i=0}^{n} x_i w_i + b = 0$$

where the weight vector W is normal to the plane, and $b/|W|$ is the distance from the origin to the plane. The n-D space is partitioned by the plane into two regions. We further define a mapping function $y = sign(f(X)) \in \{1, -1\}$, i.e.,

$$f(X) = X^T W + b = \begin{cases} > 0, & y = sign(f(X)) = 1, \ X \in P \\ < 0, & y = sign(f(X)) = -1, \ X \in N \end{cases}$$

Any point $X \in P$ on the positive side of the plane is mapped to 1, while any point $X \in N$ on the negative side is mapped to -1. A point X of unknown class will be classified to P if $f(X) > 0$, or N if $f(X) < 0$.

The learning problem:

Given a set m training samples from two linearly separable classes P and N:

$$\{(X_i, y_i), i = 1, \cdots, m\}$$

where $y_i \in \{1, -1\}$ labels X_i to belong to either of the two classes. we want to find a hyperplane in terms of W and b, that linearly separates the two classes.

Before W is properly trained, the actual output $y' = sign(f(X))$ may not be the same as the desired output y. There are four possible cases:

	Input (X, y)	Output $y' = sign(f(X))$	result
1	$(X, y = 1)$	$y' = 1 = y$	corrrect
2	$(X, y = -1)$	$y' = 1 \neq y$	incorrect
3	$(X, y = 1)$	$y' = -1 \neq y$	incorrect
4	$(X, y = -1)$	$y' = -1 = y$	corrrect

The weight vector W is updated whenever the result is incorrect (mistake driven):

- If $(X, y=-1)$ but $y' = 1 \neq y$ (case 2 above), then

  $$W^{new} = W^{old} + \eta y X = W^{old} - \eta X$$

  When the same X is presented again, we have

  $$f(X) = X^T W^{new} + b = X^T W^{old} - \eta X^T X + b < X^T W^{old} + b$$

  The output $y' = sign(f(X))$ is more likely to be $y = -1$ as desired. Here $0 < \eta < 1$ is the learning rate.

- If $(X, y=1)$ but $y' = -1 \neq y$ (case 3 above), then

  $$W^{new} = W^{old} + \eta y X = W^{old} + \eta X$$

  When the same X is presented again, we have

  $$f(X) = X^T W^{new} + b = X^T W^{old} + \eta X^T X + b > X^T W^{old} + b$$

  The output $y' = sign(f(X))$ is more likely to be $y = 1$ as desired.

Summaring the two cases, we get the learning law:

$$\text{if } \; y f(X) = y(X^T W^{old} + b) < 0, \;\; \text{then } \; W^{new} = W^{old} + \eta y X$$

The two correct cases can also be summaried as

$$y f(X) = y(X^T W + b) \geq 0$$

which is the condition a successful classifier should satisfy.

We assume initially $W = 0$, and the m training samples are presented repeatedly, the learning law during training will yield eventually:

$$W = \sum_{j=1}^{m} \alpha_j y_j X_j$$

where $\alpha_i > 0$. Note that W is expressed as a linear combination of the training samples. After receiving a new sample (X_i, y_i), vector W is updated by

$$\text{if } \; y_i f(X_i) = y_i(X_i^T W^{old} + b) = y_i(\sum_{j=1}^{m} \alpha_j y_j (X_i^T X_j) + b) < 0,$$

$$\text{then } \; W^{new} = W^{old} + \eta y_i X_i = \sum_{j=1}^{m} \alpha_j y_j X_j + \eta y_i X_i, \quad \text{i.e. } \; \alpha_i^{new} = \alpha_i^{old} + \eta$$

Now both the decision funciton

$$f(X) = X^T W + b = \sum_{j=1}^{m} \alpha_j y_j (X_i^T X_j) + b$$

and the learning law

$$\text{if } \; y_i(\sum_{j=1}^{m} \alpha_j y_j (X_i^T X_j) + b) < 0, \quad \text{then } \; \alpha_i^{new} = \alpha_i^{old} + \eta$$

are expressed in terms of the inner production of input vectors.

Support Vector Machine

For a decision hyperplane $X^T W + b = 0$ to separate the two classes P $(X_i, 1)$ and N $(X_i, -1)$, it has to satisfy

$$y_i(X_i^T W + b) \geq 0$$

for both $X_i \in P$ and $X_i \in N$. Among all such planes satisfying this condition, we want to find the optimal one that separtes the two classes with the maximal margin (the smallest distance between the two classes).

The optimal plane should be in the middle of the two classes, so that the distance from the plane to the cloeset point on either side is the same. We define two additional planes H_+ and H_- that are parallel to H_0 and go through the point cloeset to the plane on either side:

$$X^T W + b = 1, \quad \text{and} \quad X^T W + b = -1$$

All points $X_i \in P$ on the positive side should satisfy

$$X_i^T W + b \geq 1, \quad y_i = 1$$

and all points $X_i \in N$ on the negative side should satisfy

$$X_i^T W + b \leq -1, \quad y_i = -1$$

These can be combined into one inequality:

$$y_i(X_i^T W + b) \geq 1, \quad (i = 1, \cdots, m)$$

The equality holds for those points on the planes H_+ or H_-. Such points are called *supprot vectors*, for which

$$X_i^T W + b = y_i$$

i.e., the following holds for all support vectors:

$$b = y_i - X_i^T W = y_i - \sum_{j=1}^{m} \alpha_j y_j (X_i^T X_j)$$

Moreover, the distances from the origin to the three planes H_-, H_0 and H_+ are, respectively, $|b-1|/||w||$, $|b|/||w||$, and $|b+1|/||w||$, and the distances between planes H_- and H_+ is $2/||W||$, which is to be maximized. Now the problem of finding the optimal decision plane in terms of W and b can be formulated as:

$$\begin{aligned} &\text{minimize} \quad \frac{1}{2}||W||^2 \\ &\text{subject to} \quad y_i(X_i^T W + b) \geq 1, \quad (i = 1, \cdots, m) \end{aligned}$$

This constrained optimization problem can be solved by Lagrange multipliers method to minimize

$$L(W, b) = \frac{1}{2}||W||^2 - \sum_{i=1}^{m} \alpha_i (y_i (X_i^T W + b) - 1)$$

with respect to W and b. Here $\alpha_i \geq 0, \quad (i = 1, \cdots, \alpha_m)$ are m Lagrange coefficients. We let

$$\frac{\partial}{\partial W} L(W, b) = 0, \quad \frac{\partial}{\partial b} L(W, b) = 0$$

These lead, respectively, to

$$W = \sum_{j=1}^{m} \alpha_j y_j X_j, \quad \text{and} \quad \sum_{i=1}^{m} \alpha_i y_i = 0$$

Substituting these two equations back into the expression of $L(W, b)$, we get the dual problem of finding α_i:

$$\begin{aligned} &\text{maximize} \quad L(\alpha) = \sum_{i=0}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j X_i^T, X_j \\ &\text{subject to} \quad \alpha_i \geq 0, \quad \sum_{i=0}^{m} \alpha_i y_i = 0 \end{aligned}$$

Those points X_i on either of the two planes H_+ and H_- (for which the equality $y_i(W^T X_i + b) = 1$ holds) are called *support vectors* and they correspond to positive Lagrange multipliers $\alpha_i > 0$. The training depends mostly on the support vectors, while all other samples far away from the planes H_+ and H_- are not important.

Kernel Mapping

The algorithm above converges only for linearly separabel data. If the data set is not linearly separable, we can map the the samples X into a feature space of higher dimensions:

$$X \longrightarrow \Phi(X)$$

in which the classes can be linearly separated. The decision function in the new space becomes:

$$f(X) = \Phi(X)^T W^* + b^* = \sum_{j=1}^{m} \alpha_j^* y_j (\phi(X_i)^T \phi(X_j)) + b^*$$

where

$$W^* = \sum_{j=1}^{m} \alpha_j y_j \Phi(X_j)$$

and b^* are the parameters of the desicion plane in the new space. As the vectors X_i appear only in inner products in both the decision function and the learning law, the mapping function $\Phi(X)$ does not need to be explicitly specified. Instead, all we need is the inner product of the vectors in the new space. The function $\Phi(X)$ is a kernel-induced *implicit* mapping.

Definition: A kernel is a function that takes two vectors X_i and X_j as arguments and returns the value of the inner product of their images $\Phi(X_i)$ and $\Phi(X_j)$:

$$K(X_1, X_2) = \phi(X_1)^T \phi(X_2)$$

As only the inner product of the two vectors in the new space is returned, the dimensionality of the new space is not important.

The learning algorithm in the kernel space can be obtained by replacing all inner products in the learning algorithm in the original space with the kernels:

$$f(X) = \Phi(X)^T W^* + b^* = \sum_{j=1}^{m} \alpha_j^* y_j K(X_i, X_j) + b^*$$

The parameter b^* can be found from any support vectors X_i:

$$b^* = y_i - \Phi(X_i)^T W^* = y_i - \sum_{j=1}^{m} \alpha_j^* y_j (\Phi(X_i)^T \Phi(X_j)) = y_i - \sum_{j=1}^{m} \alpha_j y_j K(X_i, X_j)$$

Example 0: linear kernel
Assume $X = (x_1, \cdots, x_n)^T$, $Z = (z_1, \cdots, z_n)^T$,

$$K(X, Z) = X^T Z = \sum_{i=1}^{n} x_1 z_1$$

Example 1: polynomial kernels
Assume $X = (x_1, x_2)^T$, $Z = (z_1, z_2)^T$,

$$\begin{aligned} K(X, Z) = (X^T Z)^2 &= (x_1 z_1 + x_2 z_2)^2 = x_1^2 z_1^2 + x_2^2 z_2^2 + 2 x_1 z_1 x_2 z_2 \\ &= <(x_1^2, x_2^2, \sqrt{2} x_1 x_2), (z_1^2, z_2^2, \sqrt{2} z_1 z_2)> = \phi(X)^T \phi(Z) \end{aligned}$$

This is a mapping from a 2-D space to a 3-D space. The order can be changed from 2 to general d.

Example 2:

$$K(X, Z) = e^{-||X-Z||^2/2\sigma}$$

Example 3:

$$K(X, Z) = K(X, Z) K(X, X)^{-1/2} K(Z, Z)^{-1/2}$$