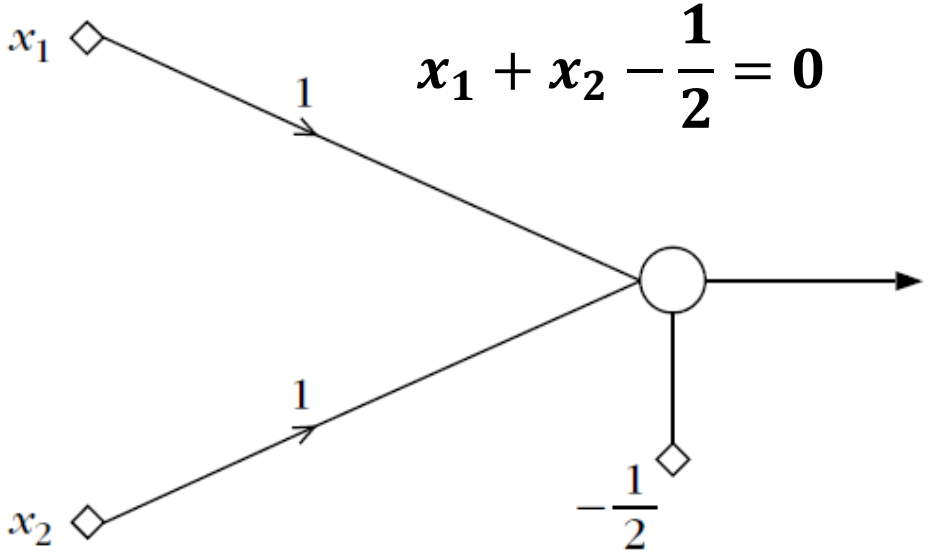
فصل چهارم
شناسایی الگو
طبقه‌بندی کننده‌های غیرخطی
NONLINEAR CLASSIFIERS

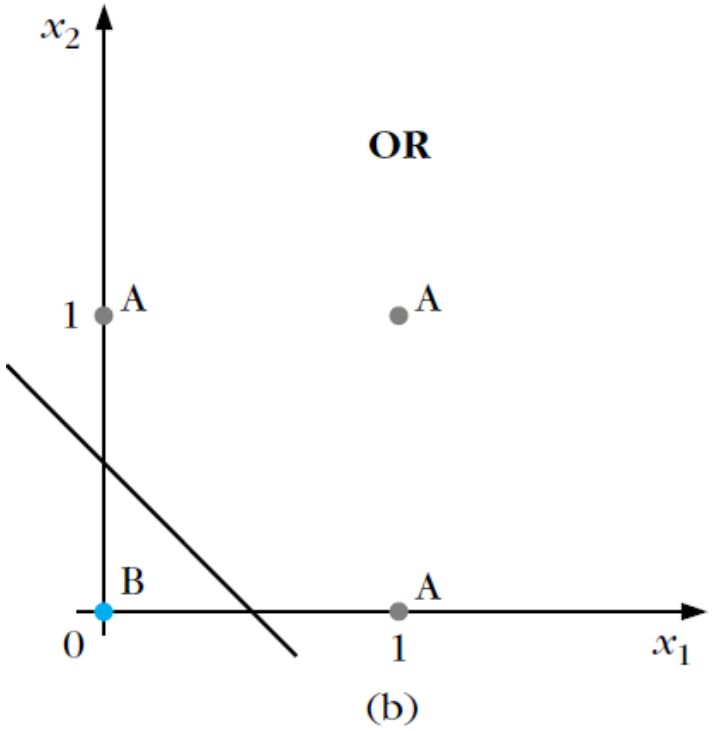
محمدجواد فدائی‌اسلام

مساله OR



$$x_1 + x_2 - \frac{1}{2} = 0$$

A perceptron realizing an OR gate.



مساله XOR

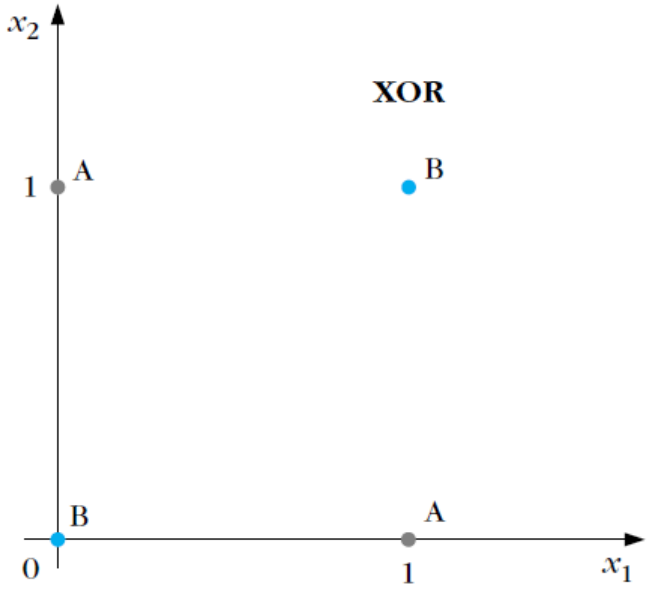


Table 4.1 Truth Table for the XOR Problem

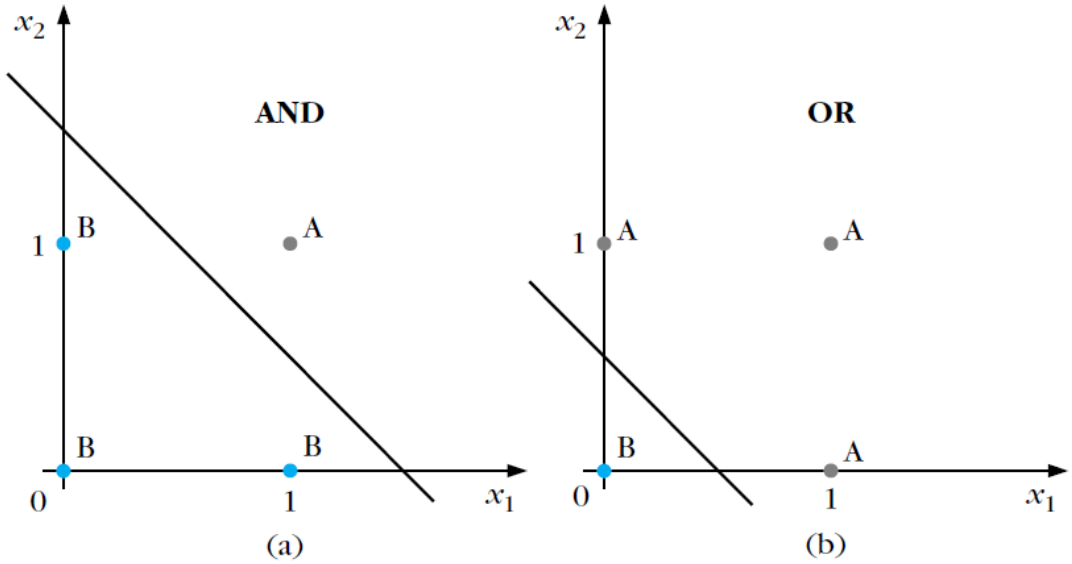
x_1	x_2	XOR	Class
0	0	0	B
0	1	1	A
1	0	1	A
1	1	0	B

هیچ خط مستقیمی وجود ندارد که این دو کلاس را از هم جدا کند.

AND, OR

Table 4.2 Truth Table for AND and OR Problems

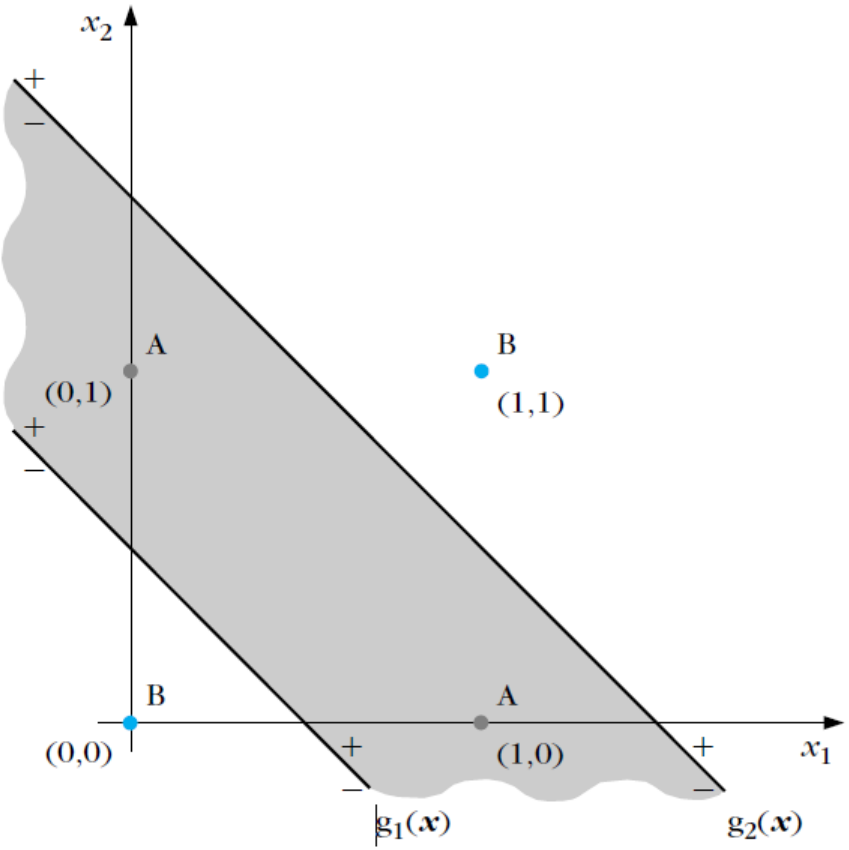
x_1	x_2	AND	Class	OR	Class
0	0	0	B	0	B
0	1	0	B	1	A
1	0	0	B	1	A
1	1	1	A	1	A



حل مساله XOR با دو خط

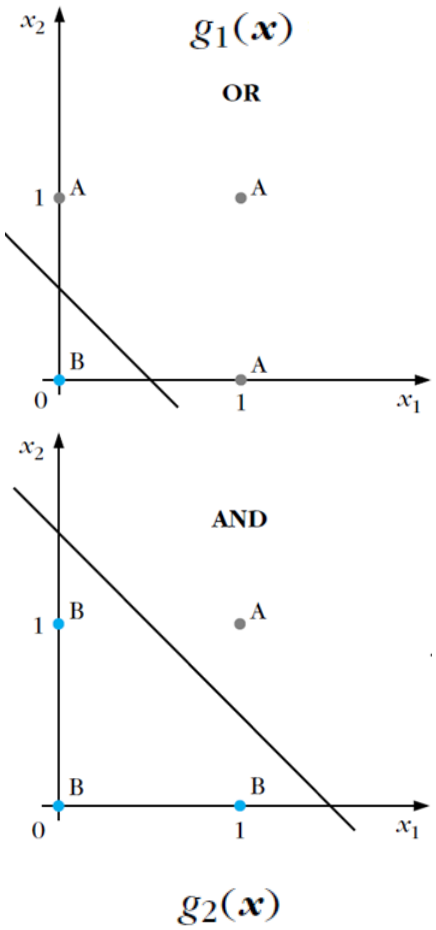
Table 4.3 Truth Table for the Two Computation Phases of the XOR Problem

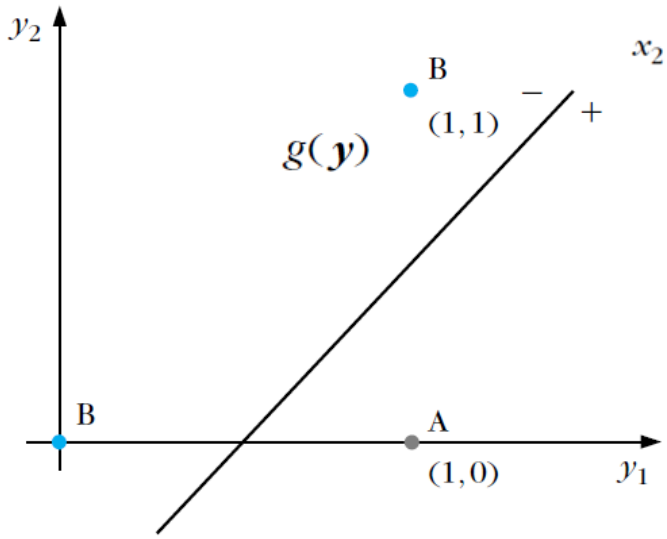
1st Phase				2nd Phase
x_1	x_2	y_1	y_2	
0	0	0 (−)	0 (−)	B (0)
0	1	1 (+)	0 (−)	A (1)
1	0	1 (+)	0 (−)	A (1)
1	1	1 (+)	1 (+)	B (0)

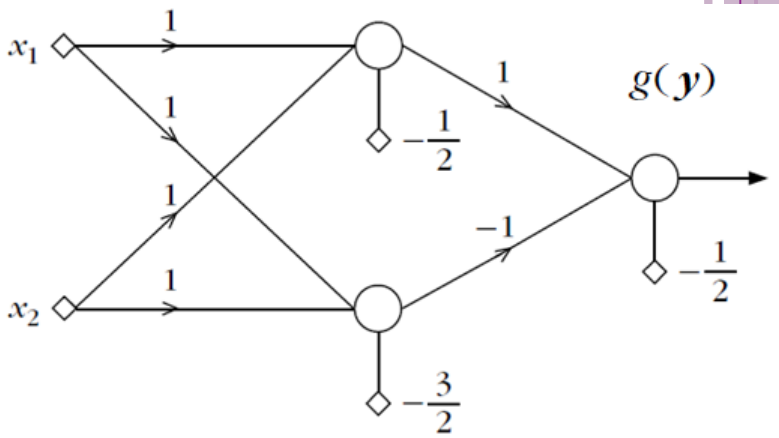


THE TWO-LAYER PERCEPTRON

پرسپترون دو لایه







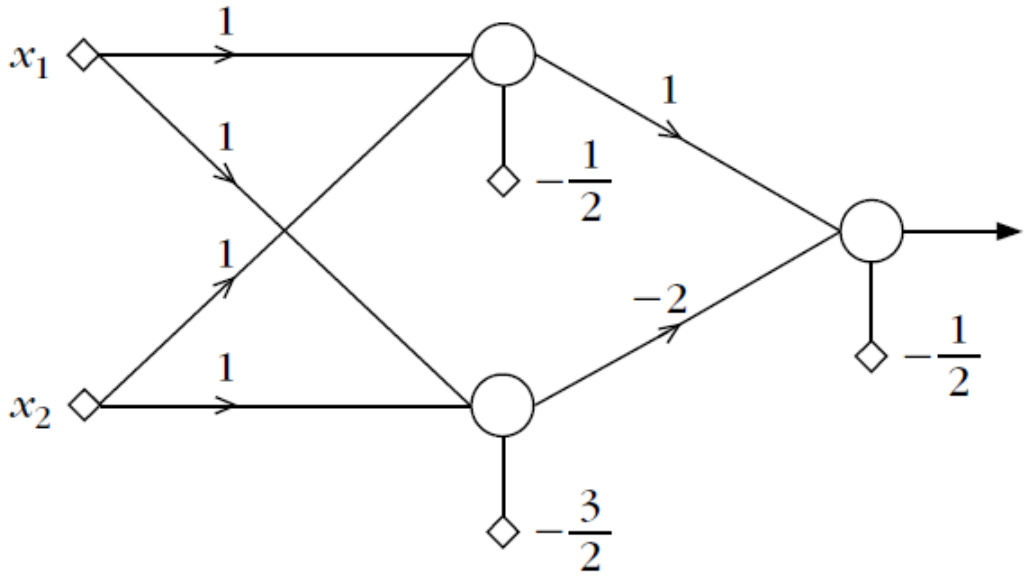
$$g_1(\boldsymbol{x}) = x_1 + x_2 - \frac{1}{2} = 0$$

$$g_2(\boldsymbol{x}) = x_1 + x_2 - \frac{3}{2} = 0$$

$$g(\boldsymbol{y}) = y_1 - y_2 - \frac{1}{2} = 0$$

THE TWO-LAYER PERCEPTRON

پرسپترون دولایه

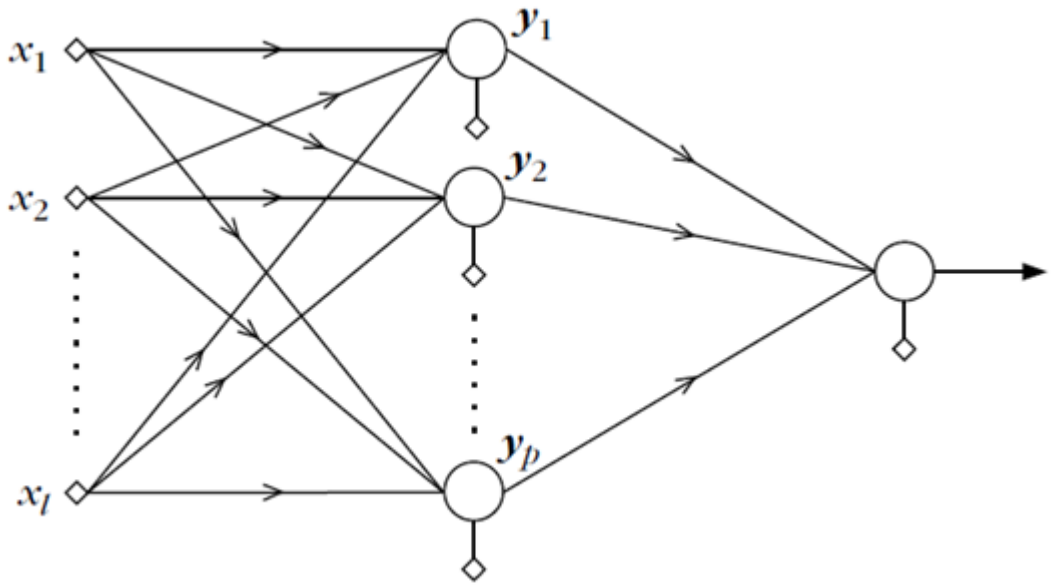


Input layer **Hidden layer** **Output layer**

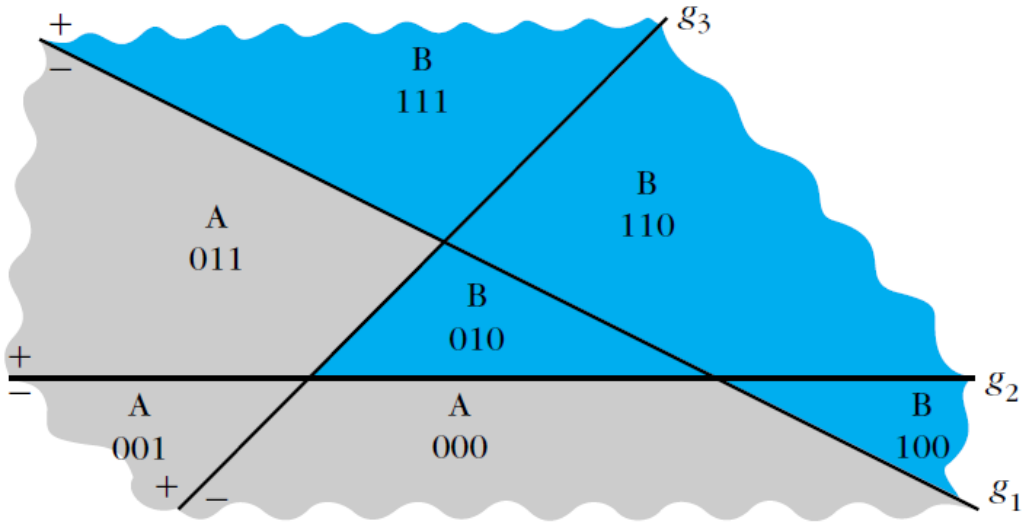
نگاهی دقیق به پرسپترون دو لایه نشان می‌دهد که عمل نورون‌های لایه پنهان در واقع نگاشت فضای ورودی بروی یک فضای جداسازی‌پذیر خطی است.

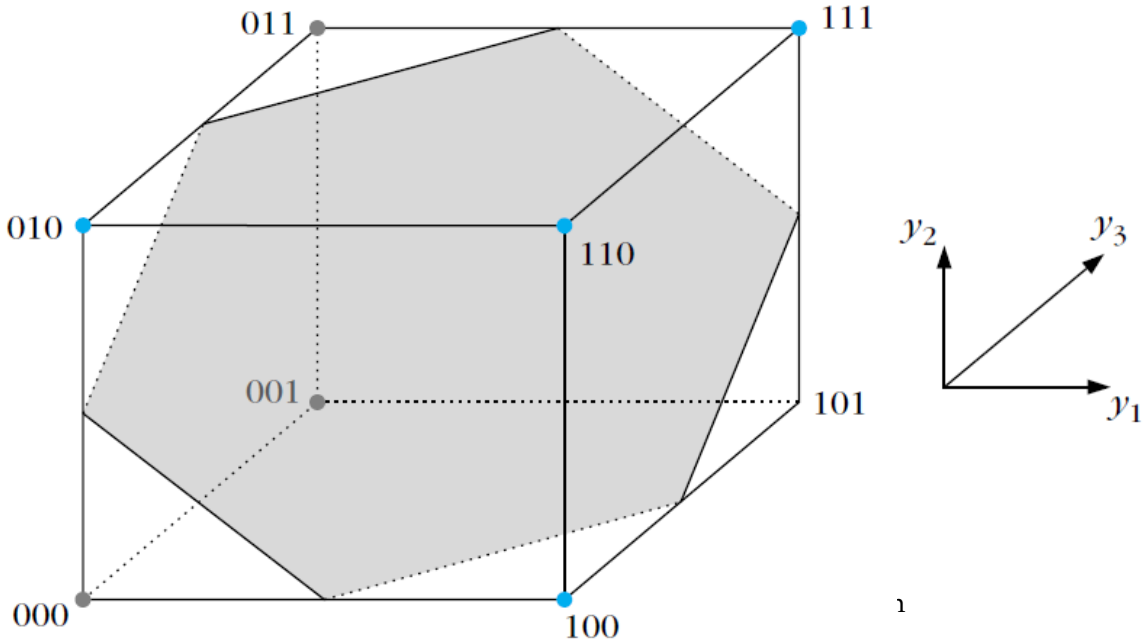
توانایی کلاس‌بندی یک پرسپترون دو لایه

- در حالت کلی، ورودی می‌تواند l بعدی در نظر گرفته شود و p نرون هم در لایه پنهان قرار داشته باشد. هر نرون میانی که خروجی صفر و یک دارد فضای مساله را به دو نیم تقسیم می‌نماید.
- این نرون‌ها فضای ورودی را بر روی رئوس یک ابرمکعب p بعدی نگاشت می‌کنند.



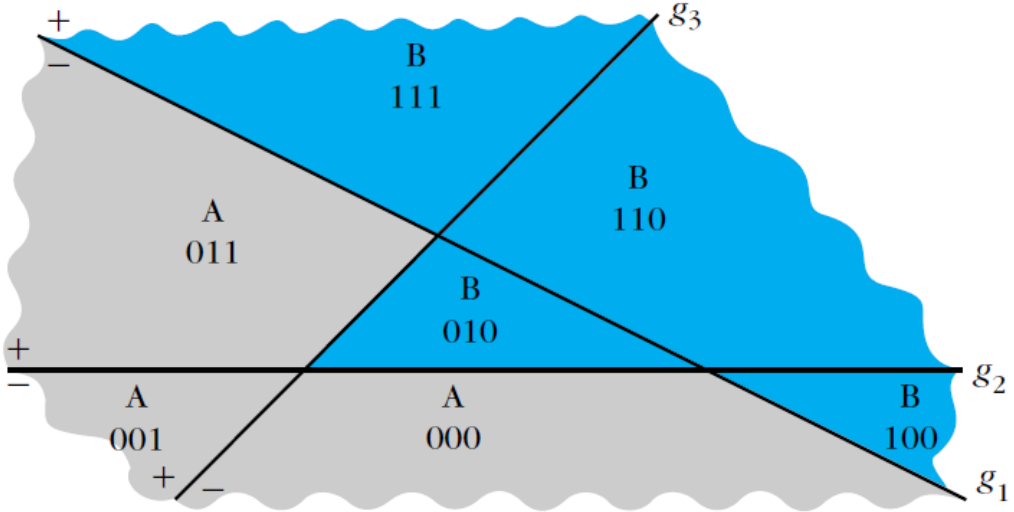
نگاشت فضای ورودی بر روی راس‌های یک ابرمکعب

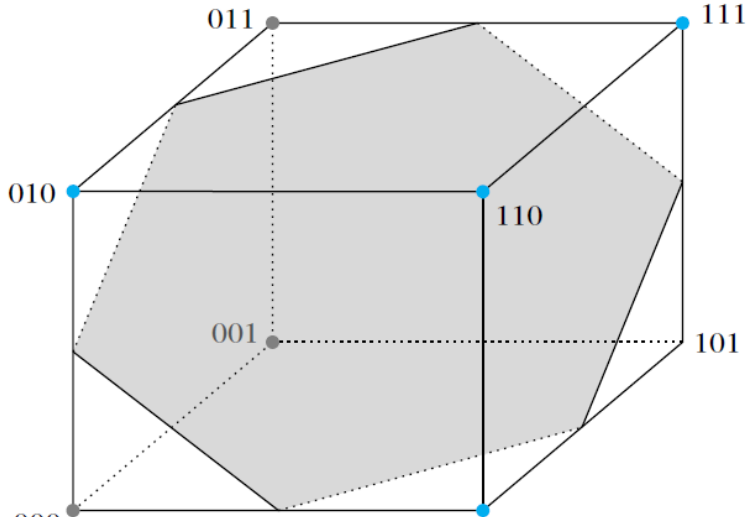




توانایی کلاس‌بندی یک پرسپترون دو لایه

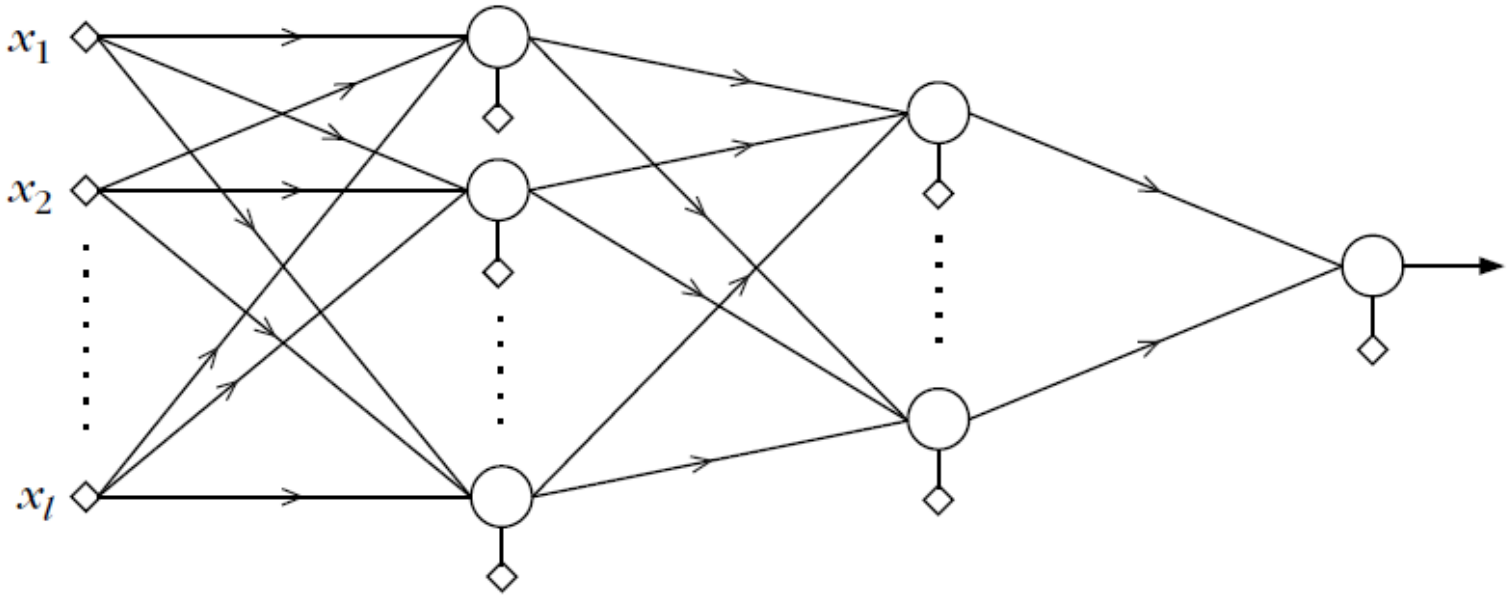
- در این مثال، کلاس A از اجتماع مناطق نگاشت شده بر روی رئوس ۰۰۰، ۰۰۱، ۰۱۱ و کلاس B از بقیه مناطق تشکیل شده است. ابرصفحه می‌تواند فضا را به درستی به دو ناحیه جدا کند.
- اگر کلاس A از اجتماع ۰۰۰، ۱۱۱، ۱۱۰ و کلاس B از بقیه تشکیل شده باشد، نمی‌توان یک ابرصفحه یافت که دو کلاس را از هم جدا کند.
- بنابراین، می‌توان نتیجه گرفت که یک پرسپترون دو لایه نمی‌تواند هر اجتماع از رئوس را جدا کند.
- لازم به ذکر است که راس ۱۰۱ مکعب با هیچ یک از مناطق چند وجهی مطابقت ندارد. گفته می‌شود چنین رئوسی با چند وجهی مجازی مطابقت دارند و بر طبقه‌بندی تأثیری ندارند.





THREE-LAYER PERCEPTRON

پرسپترون سه‌لایه



یک پرسپترون سه لایه می‌تواند هر اجتماعی از راس‌های چند وجهی را جدا کند.

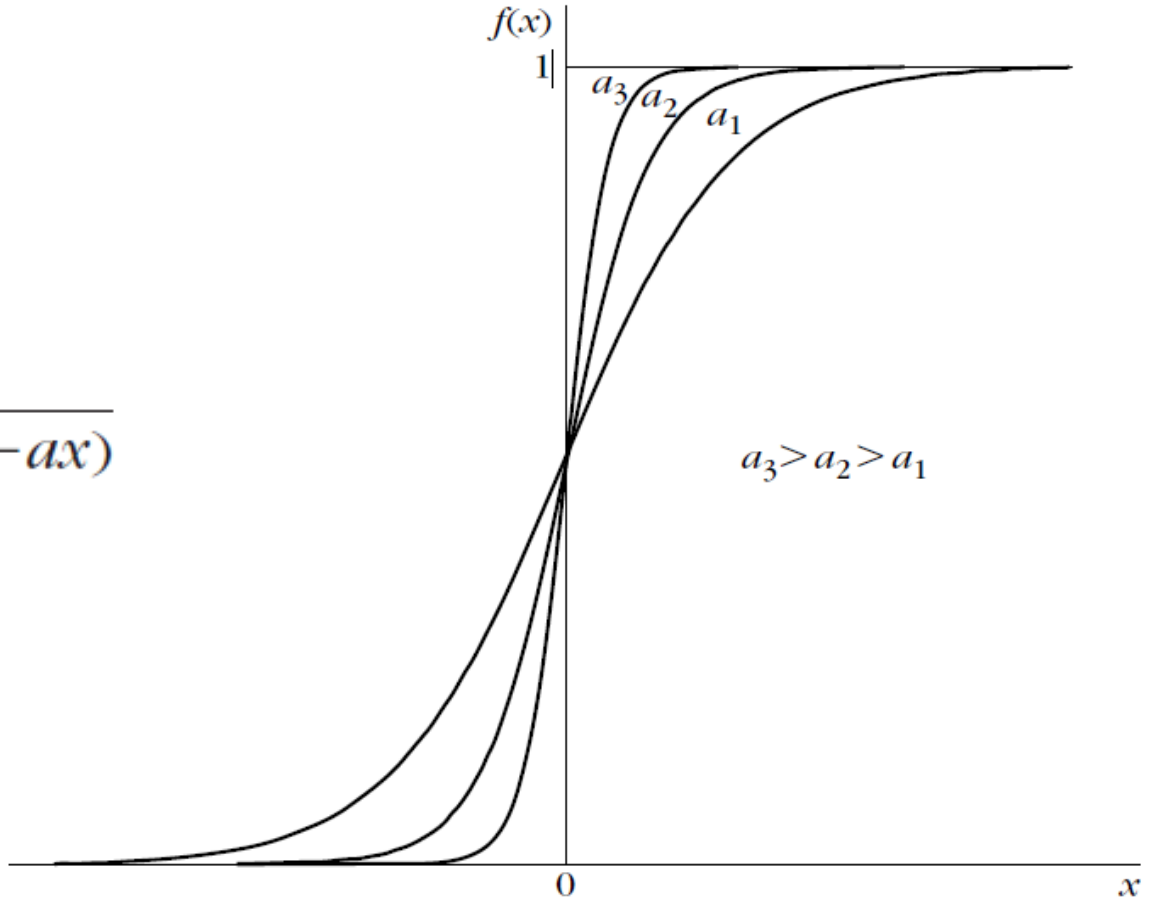
پرسپترون چندلایه و الگوریتم پس‌انتشار خطا
Backpropagation Algorithm

- فرض اینکه در عمل ما مناطقی را که داده ها در آن قرار دارند می‌شناسیم و می‌توانیم معادلات ابرصفحه‌های مربوط را محاسبه کنیم، بدون شک دور انتظار است.
- تمام آنچه که ما در عمل می‌دانیم مجموعه‌ای از نمونه‌های با برچسب معین است.
- الگوریتم پس‌انتشار خطا برای یافتن وزن شبکه عصبی از داده‌ها معرفی شد.
- این روش با محاسبه گرادیان تابع اتلاف، با توجه به هر وزن توسط قانون زنجیره‌ای کار می‌کند.

ACTIVATION FUNCTION
تابع فعال‌سازی

- The backpropagation algorithm requires a differentiable activation function. The family of sigmoid function are used for this purpose.

$$f(x) = \frac{1}{1 + \exp(-ax)}$$



POLYNOMIAL CLASSIFIERS
طبقه‌بندهای چندجمله‌ای

$$g(\boldsymbol{x}) = w_0 + \sum_{i=1}^{l} w_i x_i + \sum_{i=1}^{l-1} \sum_{m=i+1}^{l} w_{im} x_i x_m + \sum_{i=1}^{l} w_{ii} x_i^2$$

If $\boldsymbol{x} = [x_1, x_2]^T$, then the general form of $\boldsymbol{y}$ will be

$$\boldsymbol{y} = [x_1, x_2, x_1 x_2, x_1^2, x_2^2]^T$$

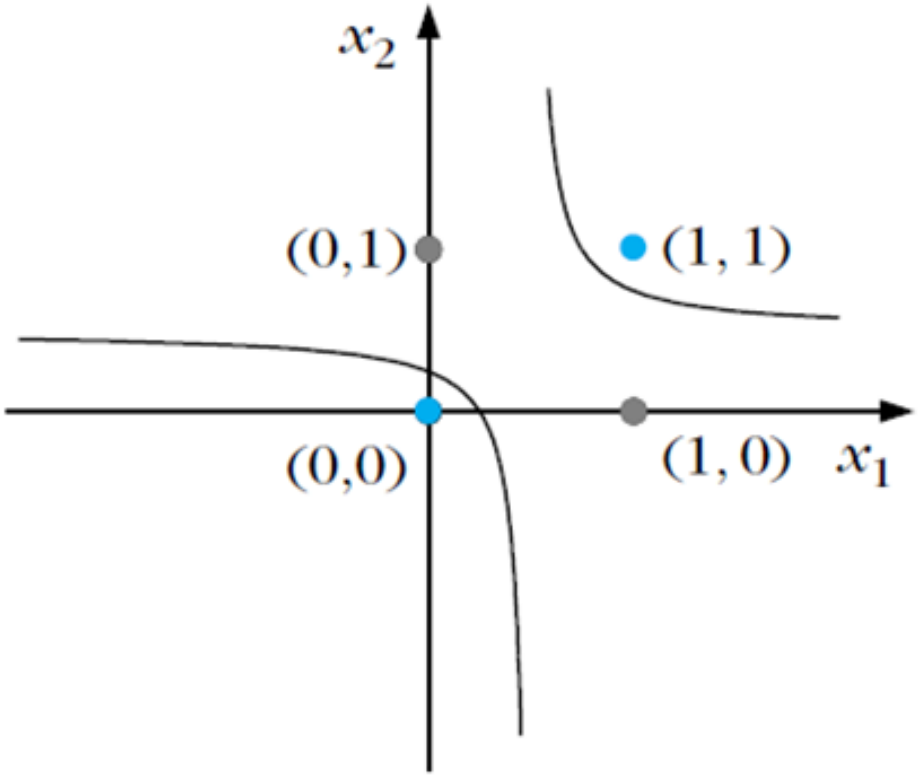
and

$$g(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{y} + w_0$$

$$\boldsymbol{w}^T = [w_1, w_2, w_{12}, w_{11}, w_{22}]$$

Polynomial classifiers (XOR)
طبقه‌بندهای چندجمله‌ای – مساله XOR

- تبدیل مساله با دو ورودی به مساله با سه ورودی و حل آن در فضای بالاتر به صورت خطی



$$g(\boldsymbol{x}) = -\frac{1}{4} + x_1 + x_2 - 2x_1x_2 \quad \begin{matrix} >0 & \boldsymbol{x} \in A \\ <0 & \boldsymbol{x} \in B \end{matrix}$$

RADIAL BASIS FUNCTION NETWORKS-RBF

$$f(\|\boldsymbol{x} - \boldsymbol{c}_i\|)$$

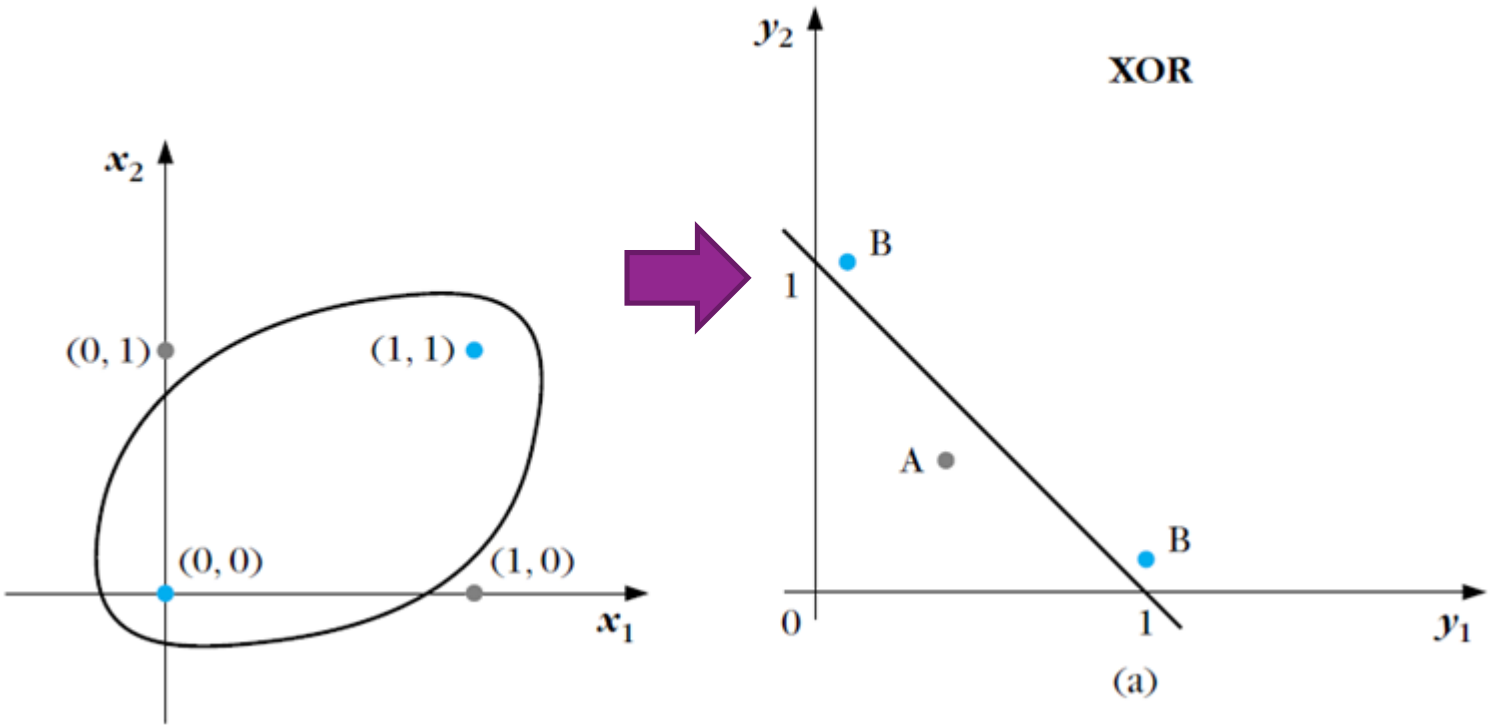
That is, the argument of the function is the Euclidean distance of the input vector $\boldsymbol{x}$ from a center $\boldsymbol{c}_i$, which justifies the name *radial basis function (RBF)*. Function f can take various forms, The Gaussian form is more widely used.

$$f(\boldsymbol{x}) = \exp\left(-\frac{1}{2\sigma_i^2}\|\boldsymbol{x} - \boldsymbol{c}_i\|^2\right) \tag{4.53}$$

For a large enough value of k, it can be shown that the function $g(\boldsymbol{x})$ is sufficiently approximated by

$$g(\boldsymbol{x}) = w_0 + \sum_{i=1}^{k} w_i \exp\left(-\frac{(\boldsymbol{x} - \boldsymbol{c}_i)^T(\boldsymbol{x} - \boldsymbol{c}_i)}{2\sigma_i^2}\right) \tag{4.55}$$

RBF – XOR PROBLEM



$$\boldsymbol{c}_1 = [1,1]^T \qquad \boldsymbol{c}_2 = [0,0]^T$$

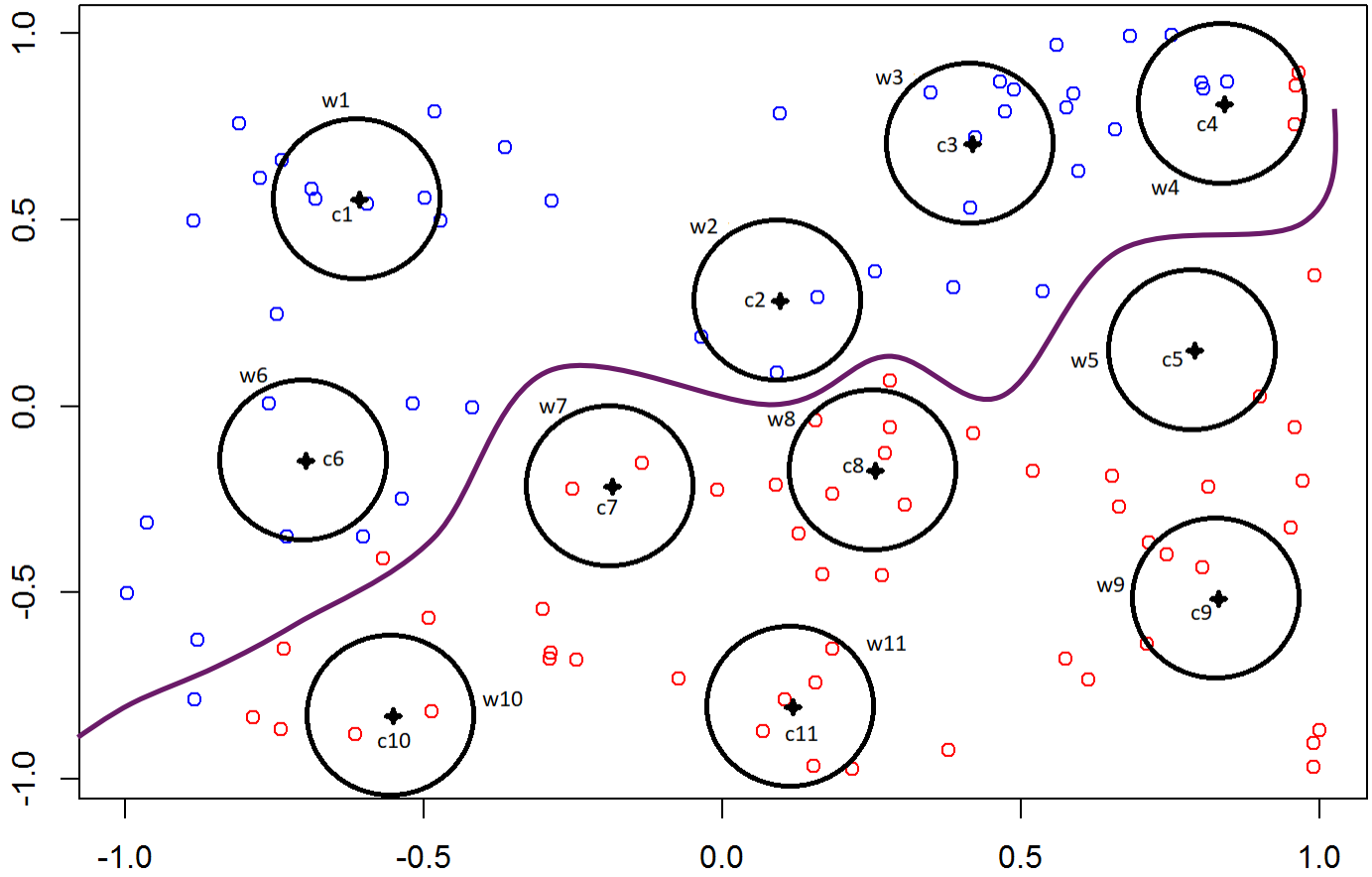
$$\boldsymbol{y} = \boldsymbol{y}(\boldsymbol{x}) = \begin{bmatrix} \exp(-\|\boldsymbol{x}-\boldsymbol{c}_1\|^2) \\ \exp(-\|\boldsymbol{x}-\boldsymbol{c}_2\|^2) \end{bmatrix}$$

$$\begin{aligned} (0,0) &\rightarrow (0.135, 1) \\ (1,1) &\rightarrow (1, 0.135) \\ (1,0) &\rightarrow (0.368, 0.368) \\ (0,1) &\rightarrow (0.368, 0.368) \end{aligned}$$

$$g(\boldsymbol{x}) = \exp(-\|\boldsymbol{x}-\boldsymbol{c}_1\|^2) + \exp(-\|\boldsymbol{x}-\boldsymbol{c}_2\|^2) - 1 = 0$$

$$g(\boldsymbol{y}) = y_1 + y_2 - 1 = 0$$

RBF EXAMPLE


1.0
0.5
0.0
-0.5
-1.0
-1.0
-0.5
0.0
0.5
1.0
w1
c1
w2
c2
w3
c3
w4
c4
w5
c5
w6
c6
w7
c7
w8
c8
w9
c9
w10
c10
w11
c11

PARZEN VS RBF

- به راحتی می توان رابطه نزدیکی را که بین RBF و روش تقریب پارزن برای توابع چگالی احتمال فصل ۲ وجود دارد مشاهده کرد.
- اما باید توجه نمود که در آنجا تعداد هسته‌ها برابر با تعداد نقاط آموزشی انتخاب شده است $k = N$. در اینجا $k \ll N$.
- علاوه بر دستاورد کاهش پیچیدگی محاسباتی، این کاهش در تعداد هسته‌ها برای قابلیت تعمیم مدل تقریبی حاصل سودمند است.

MLP vs RBF

- به طور کلی، پرسپترون‌های چندلایه کندتر از همتایان RBF خود یاد می گیرند. در مقابل، پرسپترون‌های چندلایه تعمیم بهتری دارند. به ویژه برای مناطقی که به اندازه کافی در مجموعه آموزشی نمونه ندارند.
- نتایج شبیه‌سازی نشان می‌دهد که برای دستیابی به عملکردی مشابه عملکرد پرسپترون‌های چندلایه، یک شبکه RBF باید از مرتبه بالاتری برخوردار باشد. این به دلیل محلی بودن توابع فعال سازی RBF است که استفاده از تعداد زیادی از مراکز را ضروری می کند.

SUPPORT VECTOR MACHINE-LINEAR CASE

$$\max_{\boldsymbol{\lambda}} \left(\sum_{i=1}^{N} \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \boldsymbol{x}_i^T \boldsymbol{x}_j \right)$$

$$\text{subject to} \quad \sum_{i=1}^{N} \lambda_i y_i = 0$$

$$\boldsymbol{\lambda} \geq \mathbf{0}$$

$$g(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + w_0$$

$$= \sum_{i=1}^{N_s} \lambda_i y_i \boldsymbol{x}_i^T \boldsymbol{x} + w_0$$

$$\boldsymbol{w} = \sum_{i=1}^{N} \lambda_i y_i \boldsymbol{x}_i$$

ition

SVM-Nonlinear case

$$g(\boldsymbol{x}) = \sum_{i=1}^{N_s} \lambda_i y_i K(\boldsymbol{x}_i, \boldsymbol{x}) + w_0$$

با توجه به غیر خطی بودن تابع هسته، طبقه‌بندی‌کننده به‌دست‌آمده در فضای اولیه R^l غیرخطی است.

TYPICAL KERNELS

Polynomials

$$K(\boldsymbol{x}, \boldsymbol{z}) = (\boldsymbol{x}^T \boldsymbol{z} + 1)^q, \quad q > 0$$

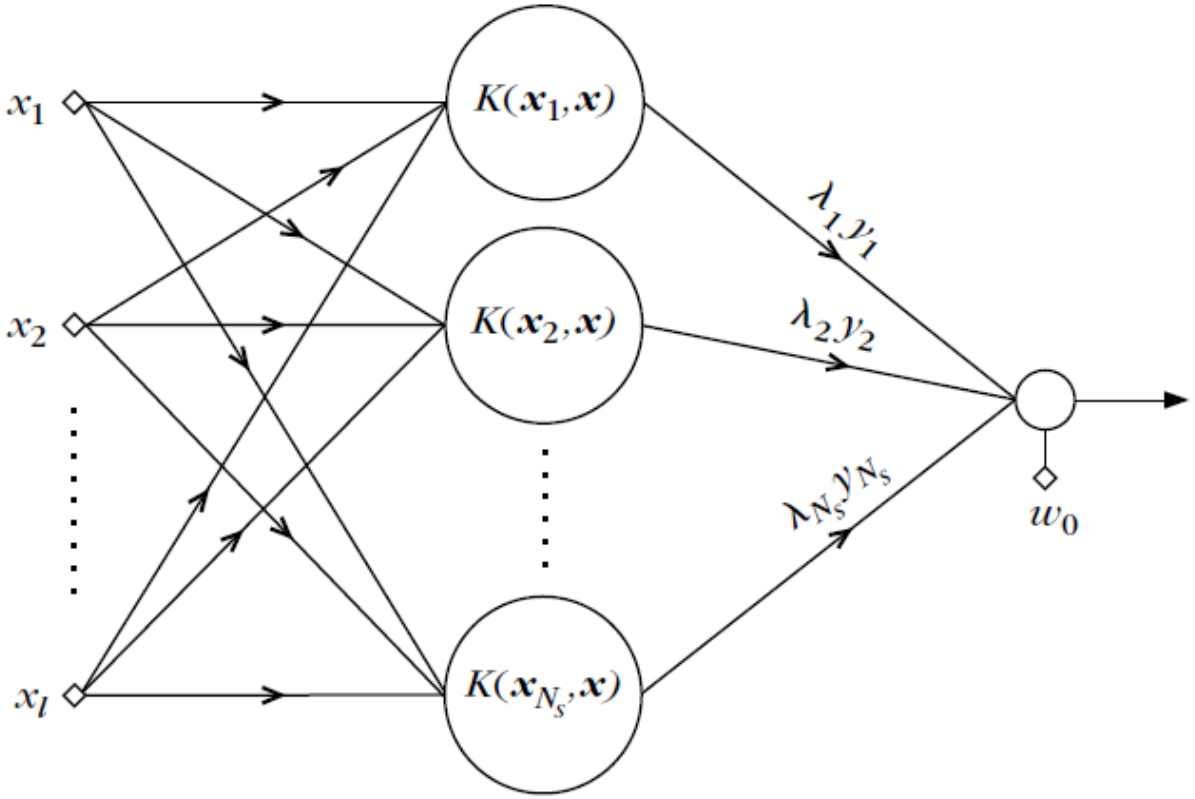
Radial Basis Functions

$$K(\boldsymbol{x}, \boldsymbol{z}) = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{z}\|^2}{\sigma^2}\right)$$

Hyperbolic Tangent

$$K(\boldsymbol{x}, \boldsymbol{z}) = \tanh\left(\beta \boldsymbol{x}^T \boldsymbol{z} + \gamma\right)$$

THE ARCHITECTURE OF NONLINEAR SVM



NONLINEAR SVM

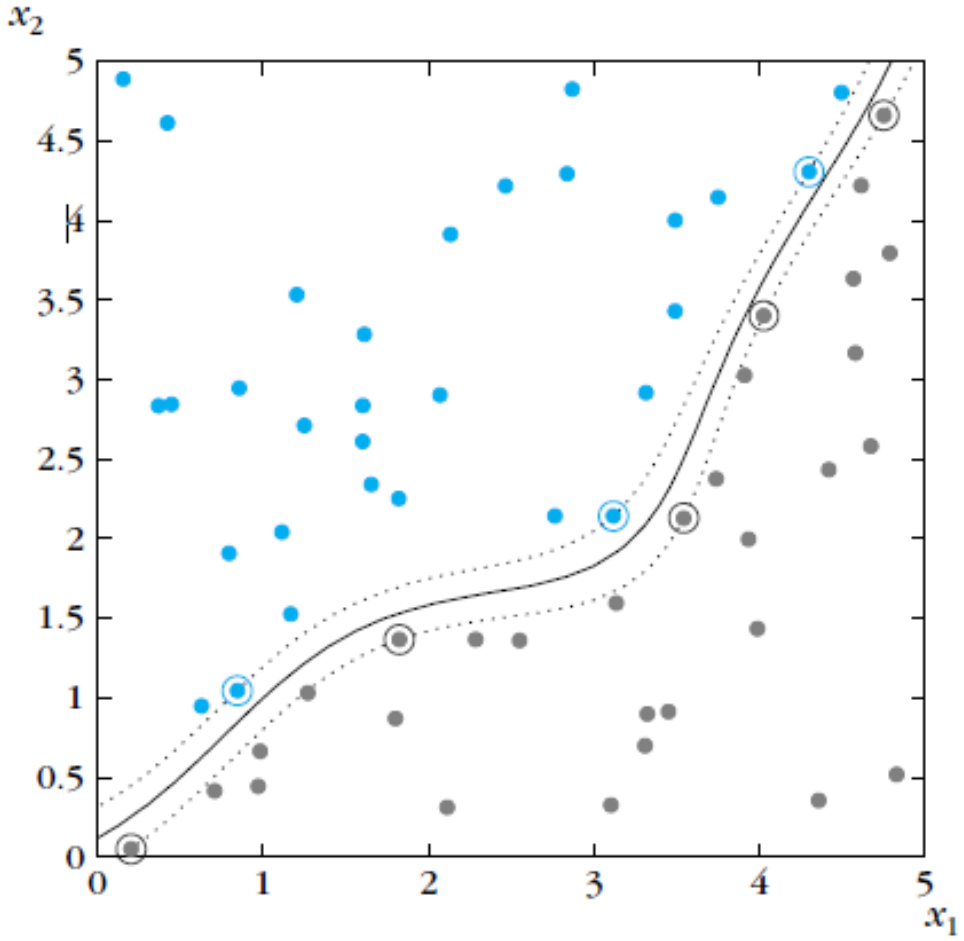


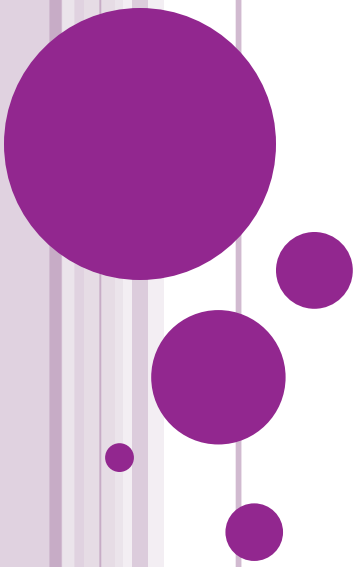
FIGURE 4.24

Example of a nonlinear SVM classifier for the case of two nonlinearly separable classes. The Gaussian RBF kernel was used. Dotted lines mark the margin and circled points the support vectors.

NONLINEAR SVM FEATURES

- If the kernel function is the RBF, then the architecture is the same as the RBF network architecture. In the SVM approach, the number of nodes and centers are the result of the optimization procedure.
- The hyperbolic tangent function is a sigmoid one. If it is chosen as a kernel, the resulting architecture is a special case of a two-layer perceptron. Once more, the number of nodes is the result of the optimization procedure. This is important. Although the SVM architecture is the same as that of a two-layer perceptron, the training procedure is entirely different for the two methods. The same is true for the RBF networks.
- A notable characteristic of the support vector machines is that the computational complexity is independent of the dimensionality of the kernel space, where the input feature space is mapped. Thus, the curse of dimensionality is bypassed.
- A major limitation of the support vector machines is that up to now there has been no efficient practical method for selecting the best kernel function.

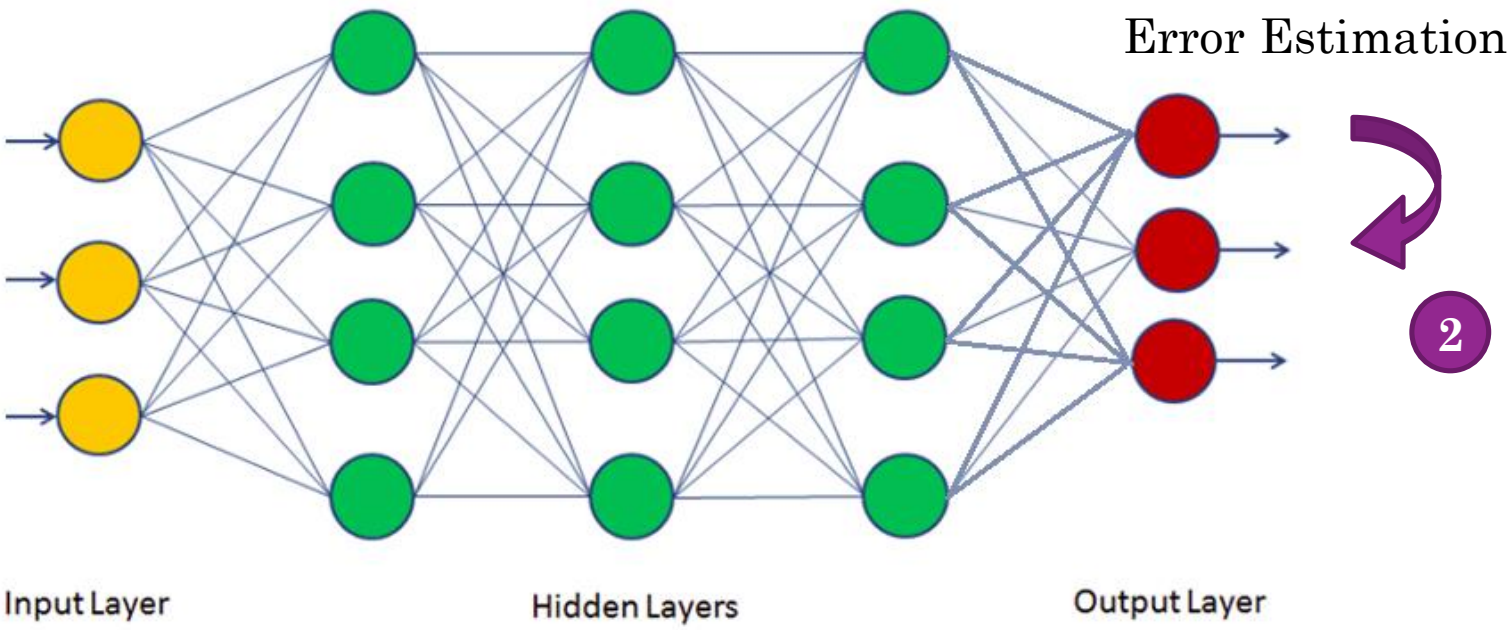
الگوریتم پس‌انتشار خطا
Backpropagation Algorithm



پس‌انتشار خطا

Forward Propagation -FP

1



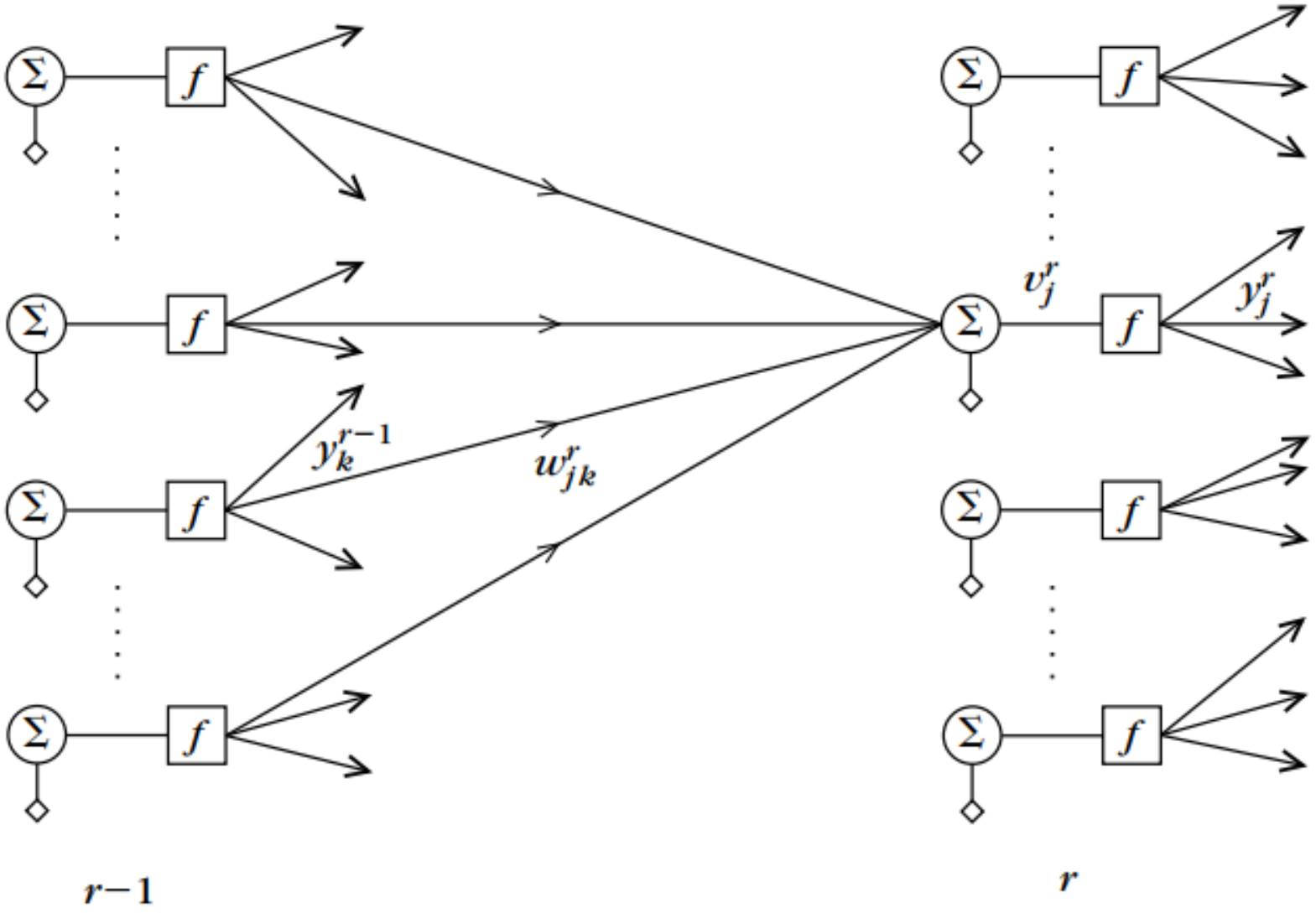
Backward Propagation -BP

3

BACKPROPAGATION
پس‌انتشار خطا

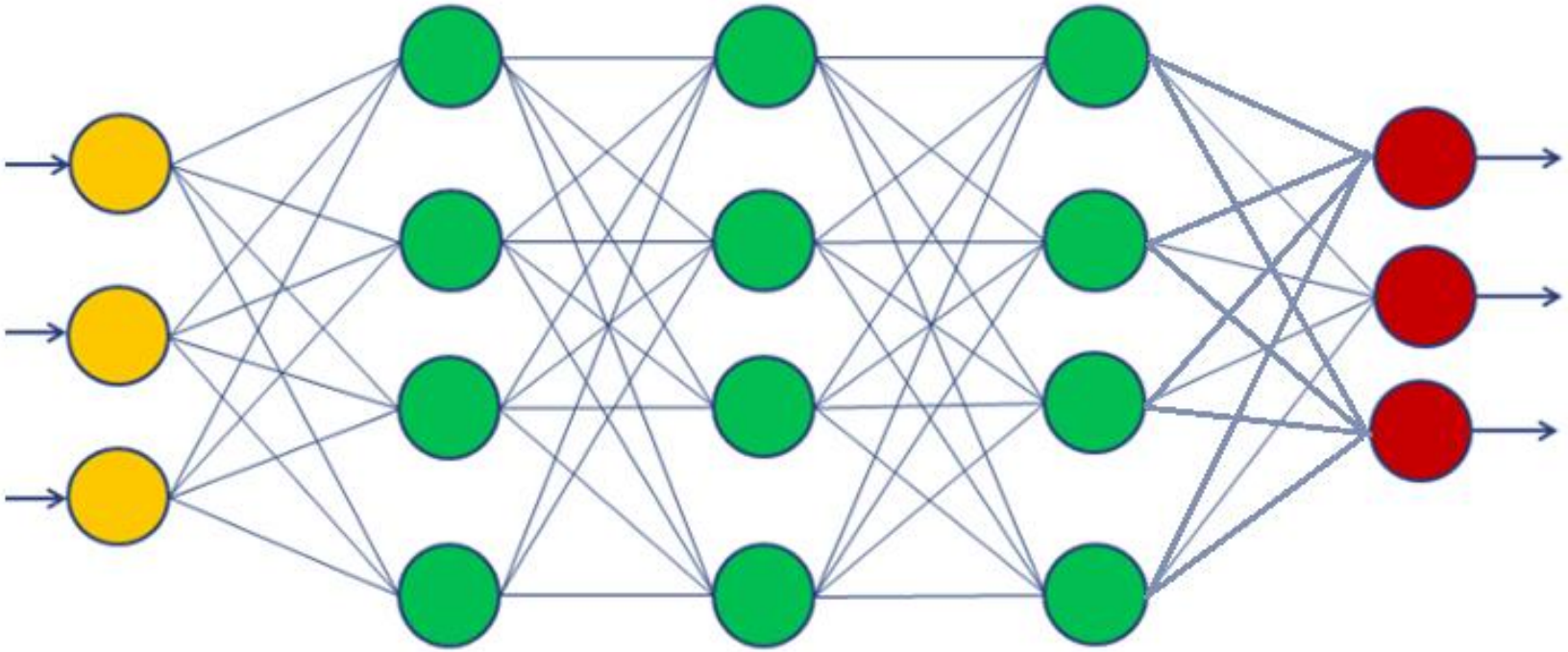
- پس‌انتشار یک الگوریتم یادگیری شبکه عصبی است که در دهه ۱۹۸۰ شهرت یافت.
- برای هر نمونه آموزشی، وزن‌ها به گونه‌ای اصلاح می‌شوند که خطا بین پیش‌بینی شبکه و مقدار هدف واقعی به حداقل برسد.
- این تغییرات در جهت «عقب» (یعنی از لایه خروجی) از طریق هر لایه پنهان به سمت لایه پنهان اول انجام می‌شود (از این رو پس‌انتشار نامیده می شود).
- اگرچه تضمین نشده است، اما عموما وزن‌ها در نهایت همگرا می‌شوند و فرآیند یادگیری متوقف می‌شود.
- این الگوریتم، با محاسبه گرادیان تابع اتلاف برای هر وزن توسط قانون زنجیره‌ای کار می‌کند.

BACKPROPAGATION



$f = activation\ function$

BACKPROPAGATION - PARAMETERS



N training pairs $= (\boldsymbol{y}(i), \boldsymbol{x}(i)), i = 1, 2, \ldots, N$

$$\boldsymbol{y}(i) = [y_1(i), \ldots, y_{k_L}(i)]^T$$

$$\boldsymbol{x}(i) = [x_1(i), \ldots, x_{k_0}(i)]^T$$

$\hat{\boldsymbol{y}}(i)$ = the output of the network

L = number of layers

BACKPROPAGATION – COST FUNCTION

$$J = \sum_{i=1}^{N} \mathcal{E}(i)$$

$$\mathcal{E}(i) = \frac{1}{2} \sum_{m=1}^{k_L} e_m^2(i) \equiv \frac{1}{2} \sum_{m=1}^{k_L} (y_m(i) - \hat{y}_m(i))^2, \quad i = 1, 2, \ldots, N$$

If $\varepsilon(i)$ is considered as the sum of squared errors.

BACKPROPAGATION – MINIMIZING COST FUNCTION

- Minimization of the cost function can be achieved via iterative techniques. Gradient descent method is adopted for this purpose.
- $\boldsymbol{W}_j^r$ = the weight vector of the jth neuron in the rth layer

$$\boldsymbol{w}_j^r = [w_{j0}^r, w_{j1}^r, \ldots, w_{jk_{r-1}}^r]^T.$$

The basic iteration step will be of the form

$$\boldsymbol{w}_j^r(\text{new}) = \boldsymbol{w}_j^r(\text{old}) + \Delta \boldsymbol{w}_j^r$$

with

$$\Delta \boldsymbol{w}_j^r = -\mu \frac{\partial J}{\partial \boldsymbol{w}_j^r}$$

where $\boldsymbol{w}_j^r(\text{old})$ is the current estimate of the unknown weights and $\Delta \boldsymbol{w}_j^r$ the corresponding correction to obtain the next estimate $\boldsymbol{w}_j^r(\text{new})$.

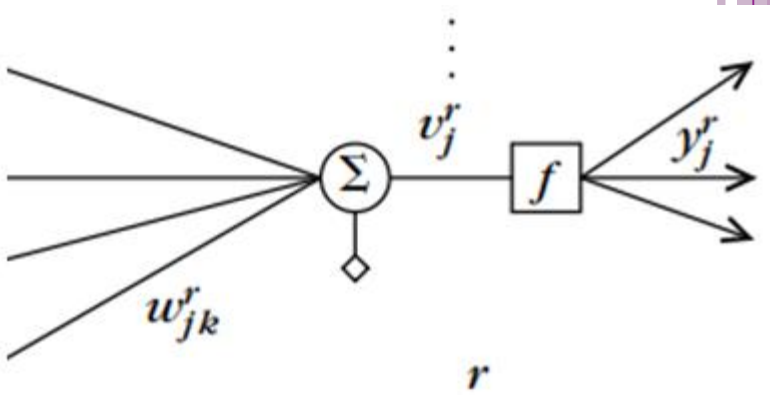
MINIMIZING COST FUNCTION- CHAIN RULE

$$\frac{\partial \mathcal{E}(i)}{\partial \boldsymbol{w}_j^r} = \frac{\partial \mathcal{E}(i)}{\partial v_j^r(i)} \frac{\partial v_j^r(i)}{\partial \boldsymbol{w}_j^r}$$

the dependence of $\mathcal{E}(i)$ on $\boldsymbol{w}_j^r$ passes through $v_j^r(i)$.

$$v_j^r(i) = \sum_{k=1}^{k_{r-1}} w_{jk}^r \, y_k^{r-1}(i) + w_{jo}^r \equiv \sum_{k=0}^{k_{r-1}} w_{jk}^r \, y_k^{r-1}(i)$$

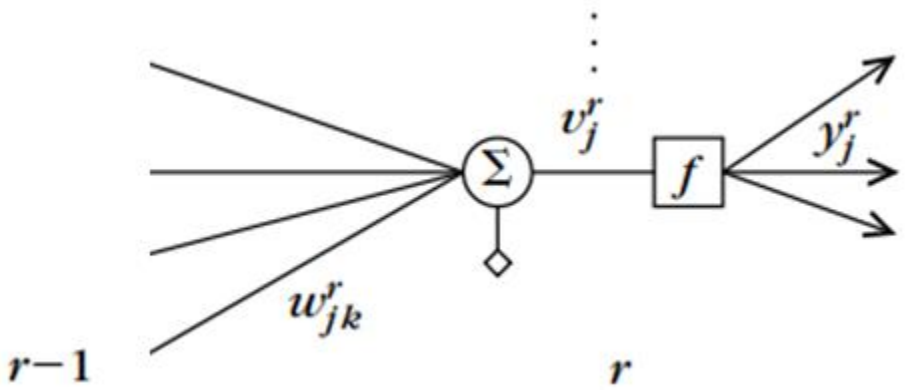
$$\frac{\partial}{\partial \boldsymbol{w}_j^r} v_j^r(i) \equiv \begin{bmatrix} \frac{\partial}{\partial w_{j0}^r} v_j^r(i) \\ \vdots \\ \frac{\partial}{\partial w_{jk_{r-1}}^r} v_j^r(i) \end{bmatrix} = \boldsymbol{y}^{r-1}(i) \qquad \boldsymbol{y}^{r-1}(i) = \begin{bmatrix} +1 \\ y_1^{r-1}(i) \\ \vdots \\ y_{k_{r-1}}^{r-1}(i) \end{bmatrix}$$



MINIMIZING COST FUNCTION- CHAIN RULE (2)

$$\frac{\partial \mathcal{E}(i)}{\partial v_j^r(i)} \equiv \delta_j^r(i)$$

$$\Delta \boldsymbol{w}_j^r = -\mu \sum_{i=1}^{N} \delta_j^r(i) \boldsymbol{y}^{r-1}(i)$$



THE BACKPROPAGATION ALGORITHM

- *Initialization:* Initialize all the weights with small random values from a pseudorandom sequence generator.
- *Forward computations:* For each of the training feature vectors $\boldsymbol{x}(i)$, compute all the $v_j^r(i)$, $y_j^r(i) = f(v_j^r(i))$,
  Compute the cost function for the current estimate of weights
- *Backward computations:* For each $i = 1, 2, \ldots, N$ and $j = 1, 2, \ldots, k_L$ compute $\delta_j^L(i)$ and in the sequel compute $\delta_j^{r-1}(i)$
- *Update the weights:* For $r = 1, 2, \ldots, L$ and $j = 1, 2, \ldots, k_r$

$$\boldsymbol{w}_j^r(\text{new}) = \boldsymbol{w}_j^r(\text{old}) + \Delta\boldsymbol{w}_j^r$$

$$\Delta\boldsymbol{w}_j^r = -\mu \sum_{i=1}^{N} \delta_j^r(i)\boldsymbol{y}^{r-1}(i)$$

The Backpropagation Algorithm
Cost Function = sum of squared error

1. $r = L$

$$\delta_j^L(i) = \frac{\partial \mathcal{E}(i)}{\partial v_j^L(i)} \tag{4.13}$$

$$\mathcal{E}(i) \equiv \frac{1}{2}\sum_{m=1}^{k_L} e_m^2(i) \equiv \frac{1}{2}\sum_{m=1}^{k_L}(f(v_m^L(i)) - y_m(i))^2 \tag{4.14}$$

Hence

$$\delta_j^L(i) = e_j(i) f'(v_j^L(i)) \tag{4.15}$$

where f' is the derivative of $f(\cdot)$. In the last layer, the dependence of $\mathcal{E}(i)$ on $v_j^L(i)$ is explicit, and the computation of the derivative is straightforward.

The Backpropagation Algorithm
Cost Function = sum of squared error

2. $r < L$. Due to the successive dependence among the layers, the value of $v_j^{r-1}(i)$ influences all $v_k^r(i), k = 1, 2, \ldots, k_r$, of the next layer. Employing the chain rule in differentiation once more, we obtain

$$\frac{\partial \mathcal{E}(i)}{\partial v_j^{r-1}(i)} = \sum_{k=1}^{k_r} \frac{\partial \mathcal{E}(i)}{\partial v_k^r(i)} \frac{\partial v_k^r(i)}{\partial v_j^{r-1}(i)} \tag{4.16}$$

and from the respective definition (4.11)

$$\delta_j^{r-1}(i) = \sum_{k=1}^{k_r} \delta_k^r(i) \frac{\partial v_k^r(i)}{\partial v_j^{r-1}(i)} \tag{4.17}$$

The Backpropagation Algorithm
Cost Function = sum of squared error

$$\frac{\partial v_k^r(i)}{\partial v_j^{r-1}(i)} = \frac{\partial \left[\sum_{m=0}^{k_{r-1}} w_{km}^r y_m^{r-1}(i)\right]}{\partial v_j^{r-1}(i)} \tag{4.18}$$

with

$$y_m^{r-1}(i) = f(v_m^{r-1}(i)) \tag{4.19}$$

Hence,

$$\frac{\partial v_k^r(i)}{\partial v_j^{r-1}(i)} = w_{kj}^r f'(v_j^{r-1}(i)) \tag{4.20}$$

From (4.20) and (4.17) the following results:

$$\delta_j^{r-1}(i) = \left[\sum_{k=1}^{k_r} \delta_k^r(i) w_{kj}^r\right] f'(v_j^{r-1}(i)) \tag{4.21}$$

The Backpropagation Algorithm
Cost Function = sum of squared error

and for uniformity with (4.15)

$$\delta_j^{r-1}(i) = e_j^{r-1}(i) f'(v_j^{r-1}(i)) \tag{4.22}$$

where

$$e_j^{r-1}(i) = \sum_{k=1}^{k_r} \delta_k^r(i) w_{kj}^r \tag{4.23}$$

Relations (4.15), (4.22), and (4.23) constitute the iterations leading to the computation of $\delta_j^r(i), r = 1, 2, \ldots, L, j = 1, 2, \ldots, k_r$.

Logistic function and its derivative

$$f(x) = \frac{1}{1+e^{-x}} = \frac{e^x}{1+e^x},$$

$$\frac{\mathrm{d}}{\mathrm{d}x} f(x) = \frac{e^x \cdot (1+e^x) - e^x \cdot e^x}{(1+e^x)^2} = \frac{e^x}{(1+e^x)^2} = f(x)\big(1 - f(x)\big)$$

The Backpropagation Algorithm
Cost Function = sum of squared error

$$w_j^r(\text{new}) = w_j^r(\text{old}) + \Delta w_j^r$$

$$\Delta w_j^r = -\mu \sum_{i=1}^{N} \delta_j^r(i) y^{r-1}(i)$$

1. $r = L$ $\qquad \delta_j^L(i) = e_j(i) f'(v_j^L(i))$

$$\delta_j^L(i) = \left[f\left(v_j^L(i)\right) - y_j(i) \right] f(v_j^L(i)) \left[1 - f(v_j^L(i))\right]$$

2. $r < L$ $\qquad \delta_j^{r-1}(i) = \left[\sum_{k=1}^{k_r} \delta_k^r(i) w_{kj}^r \right] f'(v_j^{r-1}(i))$

TERMINATION AND μ

- الگوریتم زمانی که تابع هزینه از یک آستانه خاص کوچکتر شود یا زمانی که گرادیان آن نسبت به وزن‌ها کوچک شود، خاتمه می یابد.
- تمام الگوریتم‌هایی که از روش گرادیان نزولی سرچشمه می‌گیرند، سرعت همگرایی روش پس‌انتشار به مقدار ثابت یادگیری بستگی دارد. مقدار آن باید به اندازه کافی کوچک باشد تا همگرایی را تضمین کند اما خیلی کوچک نباشد، زیرا سرعت همگرایی بسیار کند می‌شود.
- بهترین انتخاب μ بستگی زیادی به مساله و شکل تابع هزینه در فضای وزن‌ها دارد.
- پهنه‌های وسیع شیب کمی دارند. بنابراین مقادیر زیاد μ منجر به همگرایی سریعتر می شود.از سوی دیگر، نقاط کمینه با عرض کم (باریک)، مقادیر کوچکی از μ مورد نیاز است تا از کمینه عبور نکند.

BATCH MODE & SAMPLE MODE

- الگوریتم توضیح داده شده در این بخش وزن‌ها را پس از دیدن تمام نمونه‌های آموزشی بروز رسانی می‌کند. این حالت کار به حالت دسته‌ای معروف است.
- یکی از انواع این رویکرد بروزرسانی وزنه‌ها به ازای هر نمونه است.
- بروزرسانی دسته‌ای یک فرآیند میانگین‌گیری در ذات خود دارد. این منجر به تخمین بهتر گرادیان و در نتیجه همگرایی با رفتار بهتر می‌شود.
- از سوی دیگر، حالت بروزرسانی به ازای هر نمونه، در حین آموزش درجه بالاتری از تصادفی بودن را نشان می‌دهد. این ممکن است به الگوریتم کمک کند تا از گیرافتادن در کمینه محلی جلوگیری کند.

Inside the Black Box: Backpropagation and Interpretability

- شبکه‌های عصبی مانند یک جعبه سیاه هستند. چگونه می‌توانم آنچه را که شبکه پس‌انتشار آموخته است، درک کنیم؟
- یکی از معایب اصلی شبکه‌های عصبی در بازنمایی دانش آنها نهفته است. کسب دانش از شبکه‌ای از واحدهای متصل شده با پیوندهای وزنی برای انسان دشوار است.
- برخی از روش‌ها شبکه‌ها را هرس می‌کنند. این شامل ساده‌سازی ساختار شبکه با حذف پیوندهای وزنی است که کمترین تأثیر را بر شبکه آموزش دیده دارند.

- بر اساس اصل Occam razor «ساده ترین توضیح معمولا بهترین توضیح» است.

تابع اتلاف
Loss Function

- معیاری که نشان‌دهنده آن است که یک شبکه عصبی چقدر خروجی واقعی را با توجه به یک ورودی پیش‌بینی می‌کند. آن اختلاف بین خروجی پیش‌بینی شده و خروجی واقعی را اندازه‌گیری می‌کند. اتلاف کمتر به معنای تناسب بهتر و دقت بالاتر است. اتلاف بیشتر به معنای تناسب بدتر و دقت کمتر است.
  - تابع اتلاف میانگین مربعات خطا

Mean Square Error

$$\varepsilon(i) = \frac{1}{2}\sum_{m=1}^{k}(\hat{y}_m - y_m)^2$$

  - آنتروپی متقابل دودیی

Cross-Entropy

$$\varepsilon(i) = -\sum_{m=1}^{k} y_m \ln \hat{y}_m + (1 - y_m)\ln(1 - \hat{y}_m)$$

تابع اتلاف – ادامه

- آنتروپی متقاطع همگرایی سریع‌تری نسبت به میانگین مربعات خطا دارد، زیرا زمانی که خروجی پیش‌بینی‌شده از خروجی واقعی فاصله دارد، شیب تندتری دارد. این بدان معناست که شبکه عصبی می تواند به سرعت خطاهای خود را تصحیح کند.
- اگر مقیاس داده‌ها یکسان نباشد، در روش میانگین مربعات خطا اثر بیشتر را به داده‌های بزرگتر خواهد داد.
- در برخی از مسائل، نشان داده شده است که الگوریتم نزول گرادیان با معیار خطای مربع در کمینه محلی گیر می‌افتد.

سه نوع روش نزولی گرادیان

- **Batch Gradient Descent**

  Parameters are updated after computing the gradient of the error with respect to the entire training set

- **Stochastic Gradient Descent-SGD**

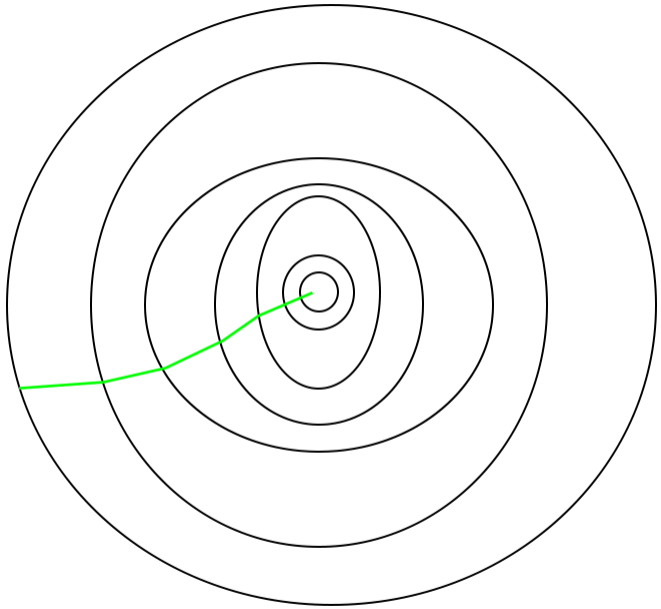
  Parameters are updated after computing the gradient of the error with respect to a single training example

- **Mini-Batch Gradient Descent**

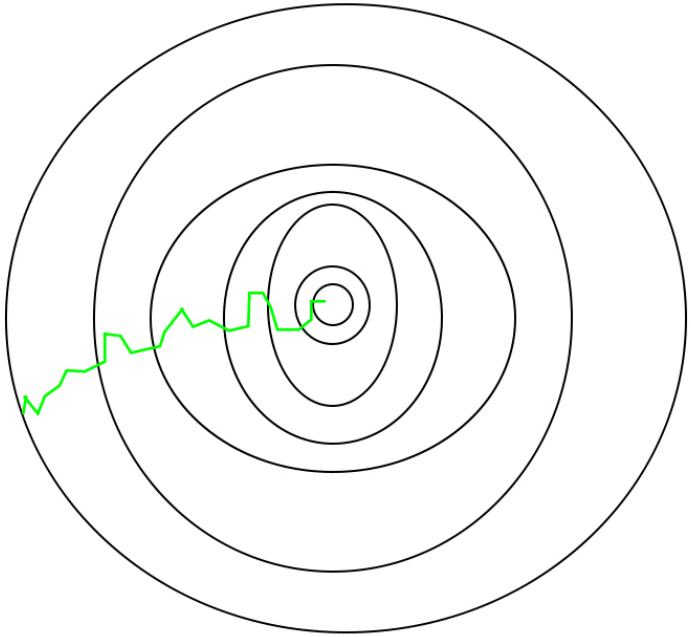
  Parameters are updated after computing the gradient of the error with respect to a subset of the training set

سه نوع روش نزولی گرادیان
مسیر رسیدن به کمینه

Batch

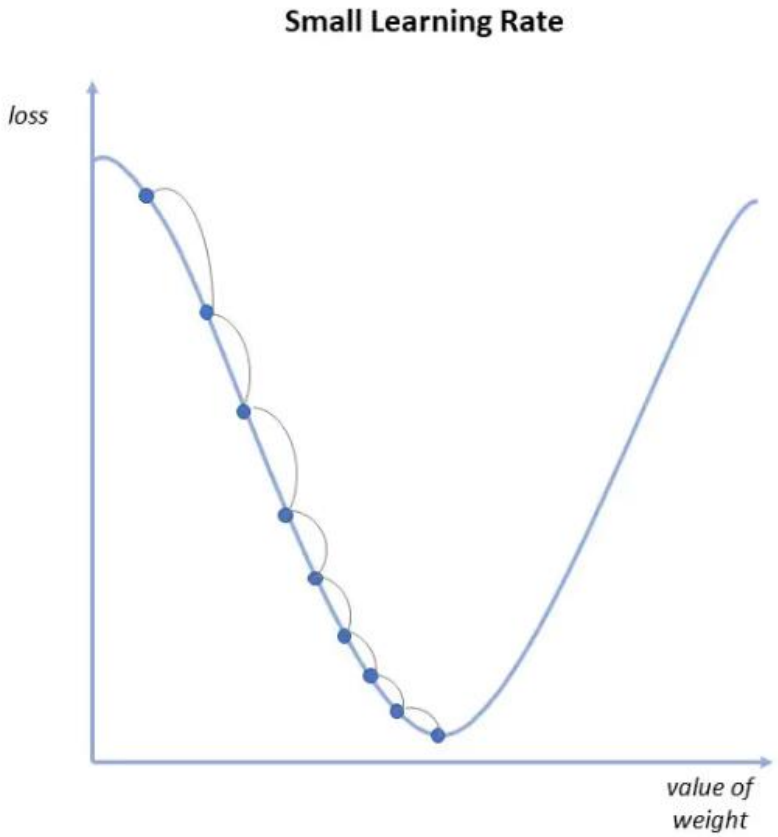


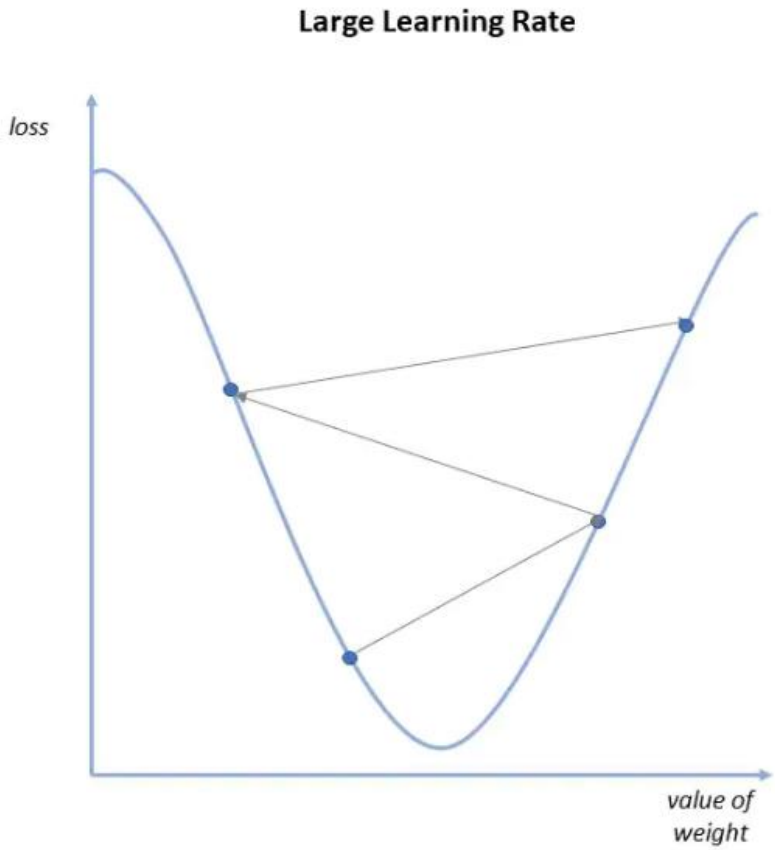
Stochastic



The learning rate in Gradient Descent

- The learning rate in Gradient Descent is constant throughout the training process for all the parameters. This can slow the convergence.





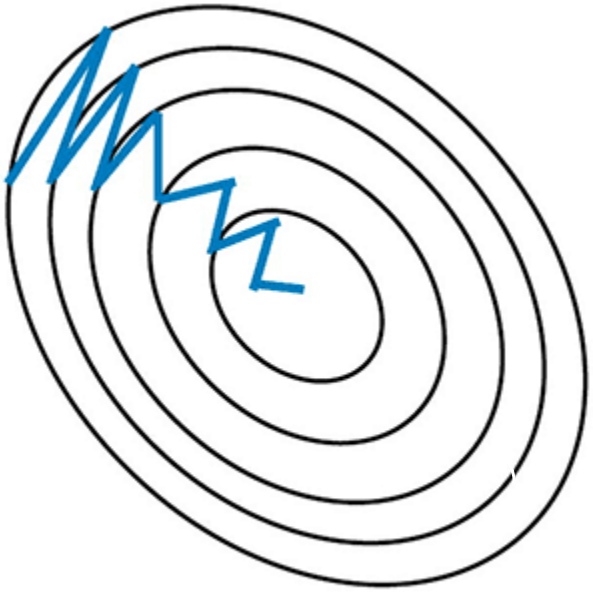
اثر ممنتم
MOMENTUM TERM

- یکی از راه‌های غلبه بر مشکل کند بودن همگرایی، استفاده از ممنتم است. بردار تصحیح نه تنها به عبارت گرادیان بلکه به مقدار آن در مرحله قبل نیز بستگی دارد. ثابت α ضریب ممنتم نامیده می‌شود و در عمل بین ۰/۸ و ۰/۱ انتخاب می‌شود.

$$\Delta \boldsymbol{w}_j^r(\text{new}) = \alpha \Delta \boldsymbol{w}_j^r(\text{old}) - \mu \sum_i^N \delta_j^r(i) \boldsymbol{y}^{r-1}(i)$$

$$\boldsymbol{w}_j^r(\text{new}) = \boldsymbol{w}_j^r(\text{old}) + \Delta \boldsymbol{w}_j^r(\text{new})$$

withhout Momentum



with Momentum