

آموزش شبکه‌های عصبی مصنوعی



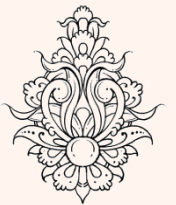
کارگاه مقدماتی «امنیت هوش مصنوعی»

زمستان ۱۴۰۲

احمد محمودی ازناوه

فهرست مطالب

- نزول گرادیانی (الگوریتم گرادیان)
- آموزش تک‌نورون
- آموزش شبکه‌های عصبی چندلایه
 - الگوریتم پس‌انتشار خطا
 - روش‌های بهبود کارایی
 - روش‌های مطرح بهینه‌سازی
 - Forward-forward algorithm



نزول گرادیانی

Gradient Descend



ابعاد متفاوت الگوریتم‌های یادگیری ماشین

- مدل $g(\mathbf{x}|\theta)$
- تابع هزینه (cost or loss function)

$$E(\theta | \mathcal{X}) = \sum_t L(r^t, g(\mathbf{x}^t | \theta))$$

- فرآیند بهینه‌سازی

$$\theta^* = \arg \min_{\theta} E(\theta | \mathcal{X})$$

- در صورتی که مدل پیچیده‌تر شود، به روش‌های پیچیده‌تری برای یافتن پارامترهای بهینه احتیاج خواهیم داشت.
- برای انجام مناسب آموزش به مدلی با ظرفیت مناسب، تعداد نمونه‌های آموزشی مناسب و یک فرآیند بهینه‌سازی خوب احتیاج داریم.



رادهای آموزشی

$$\mathcal{X} = \{x^t, r^t\}_{t=1}^N$$

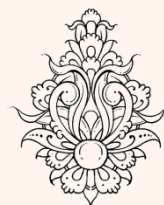
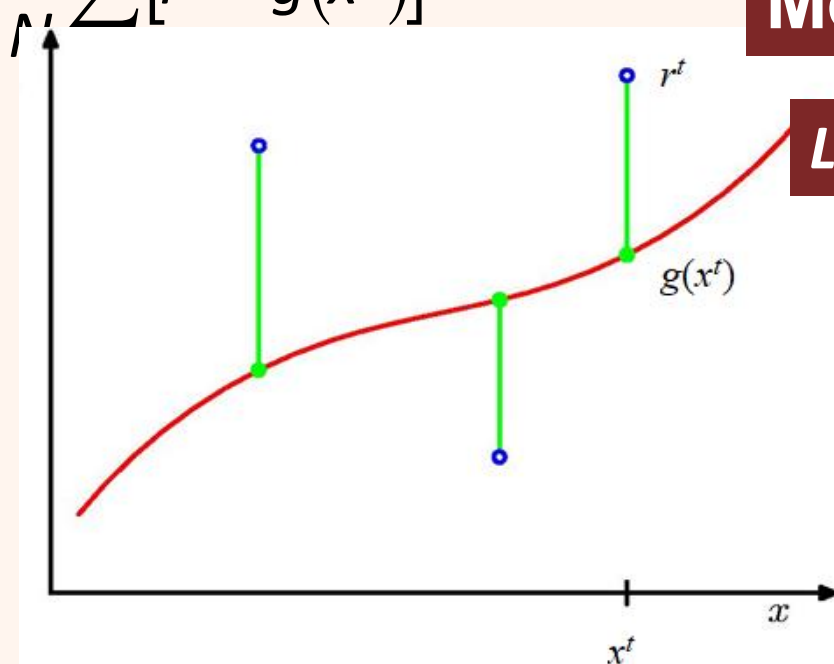
تابع هزینه

- به توان مثال فضای داده‌های آموزشی را می‌توان به صورت زیر تعریف کرد:

$$E(g | \mathcal{X}) = \frac{1}{N} \sum [r^t - g(x^t)]^2$$

Mean Square Error

Least Squares estimates



• قدر مطلق خطا

Absolute Error

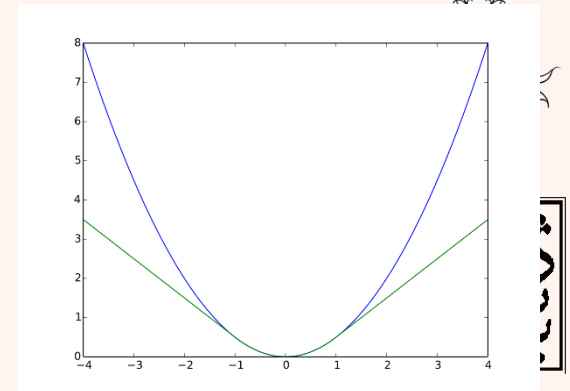
$$E(\theta | \mathcal{X}) = \frac{1}{2} \sum_{t=1}^N |r^t - g(x^t | \theta)|$$

ϵ -sensitive(hing) L

$$E(\theta | \mathcal{X}) = \frac{1}{2} \sum_{t=1}^N \mathbf{1}(|r^t - g(x^t | \theta)| > \epsilon) (|r^t - g(x^t | \theta)| - \epsilon)$$

Huber Loss

$$a = r^t - g(x^t | \theta), \quad L_\delta(a) = \begin{cases} \frac{1}{2} a^2 & \text{for } |a| \leq \delta \\ \delta(|a| - \frac{1}{2} \delta) & \text{otherwise} \end{cases}$$



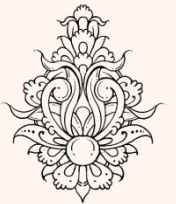
تابع هزینه

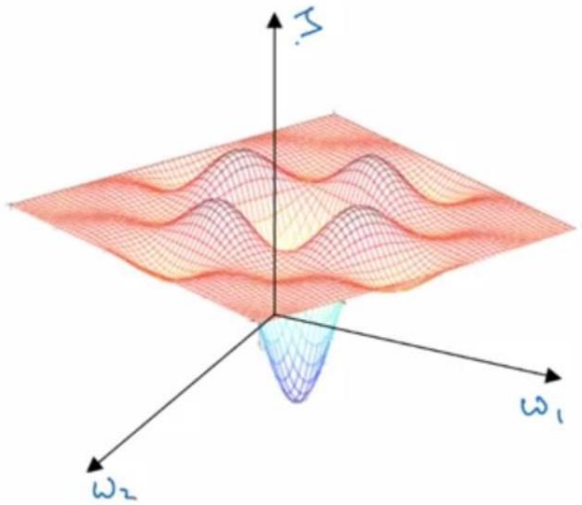
Binary cross entropy

$$E(\mathbf{w}, w_0 | \mathcal{X}) = -\sum_t r^t \log y^t + (1 - r^t) \log (1 - y^t)$$

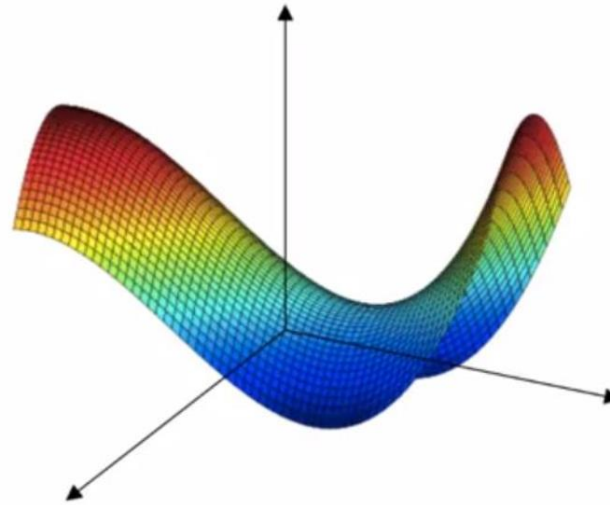
cross entropy

$$E(\{\mathbf{w}_i, w_{i0}\}_i | \mathcal{X}) = -\sum_t \sum_i r_i^t \log y_i^t$$

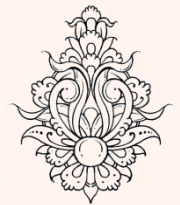




What about very high-dimensional spaces?



For high dimensional parameter space, most points of zero gradients are not local optima and are saddle points.



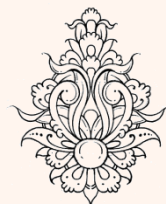
Andrew Ng, Deep learning specialization

- در دسته‌بندی مبتنی بر جداساز، پارامترها به گونه‌ای بهینه‌سازی می‌شوند که فضای دسته‌بندی حداقل شود:

$$w^* = \arg \min_w E(w | X)$$

- در بیشتر مواقع، راه حل تحلیلی برای یافتن پارامترها وجود ندارد و چاره‌ای جز استفاده از یک روش بهینه‌سازی تکرارشونده نخواهد بود.
- استفاده از نزول گرادیان یکی از پرکاربردترین راه‌کارهاست.
- بردار گرادیان به صورت زیر تعریف می‌شود:

$$\nabla_w E = \left[\frac{\partial E}{\partial w_1}, \frac{\partial E}{\partial w_2}, \dots, \frac{\partial E}{\partial w_d} \right]^T$$

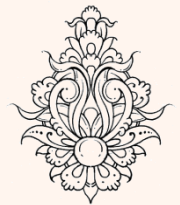
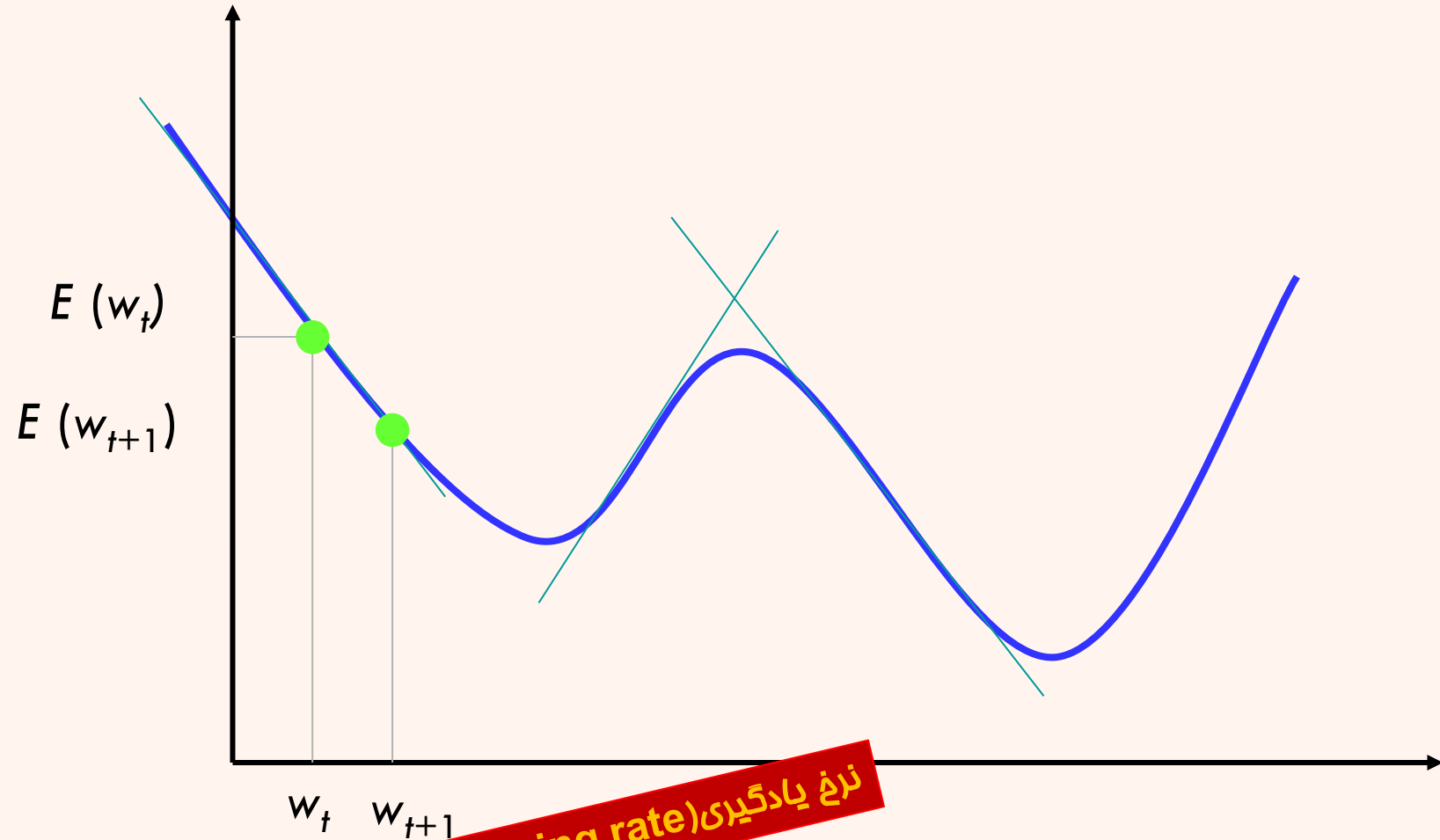


Gradient-Descent

$$\Delta w_t = -\eta \frac{\partial E}{\partial w_t}, \forall i$$

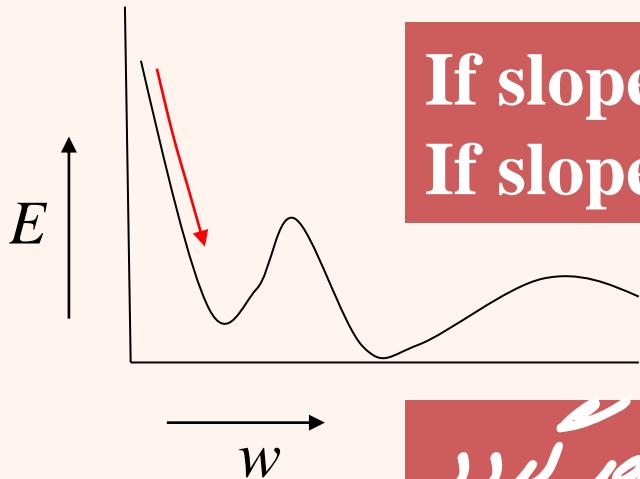
$$w_{t+1} = w_t + \Delta w_t$$

- پارامترها با یک مقدار تصادفی مقداردهی می‌شوند.
- بر خلاف جهت گرادیان مقدار پارامترها را به روز می‌شوند.



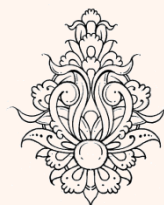
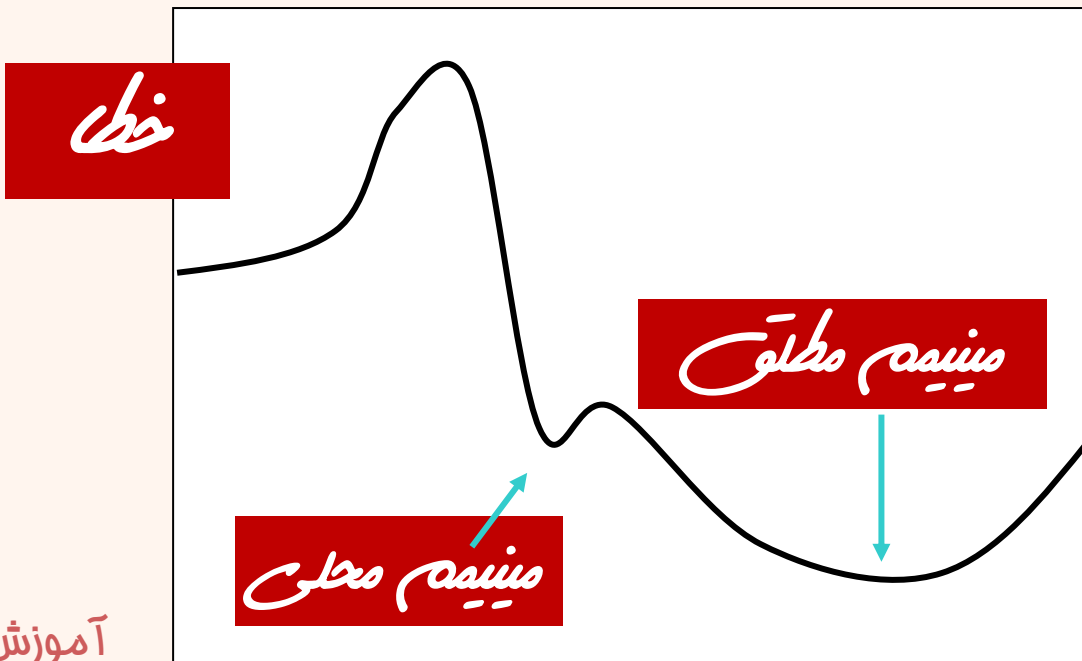
نرخ یادگیری (learning rate) η آموزش شبکه‌های عصبی
گیر کردن در مینیمم محلی از معایب این روش است.

Gradient Descent



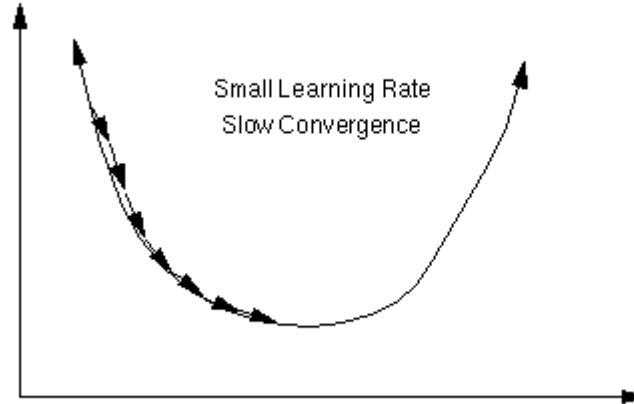
If slope is negative \rightarrow increase w
If slope is positive \rightarrow decrease w

مینیمم محلی جایی است که مشتق صفر گردد

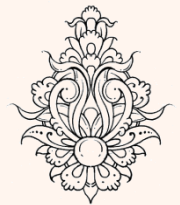
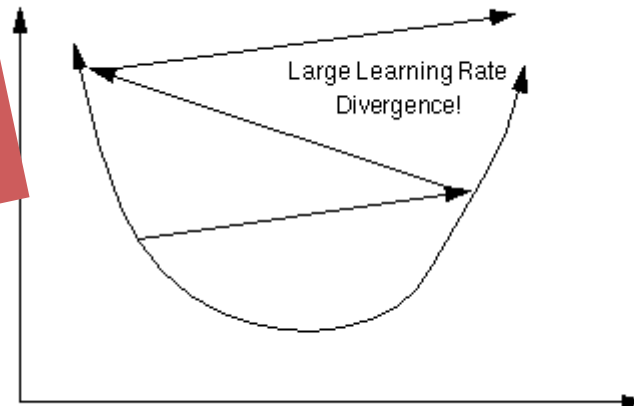


تنظیم نرخ یادگیری

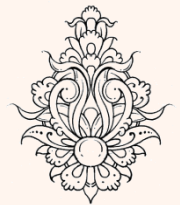
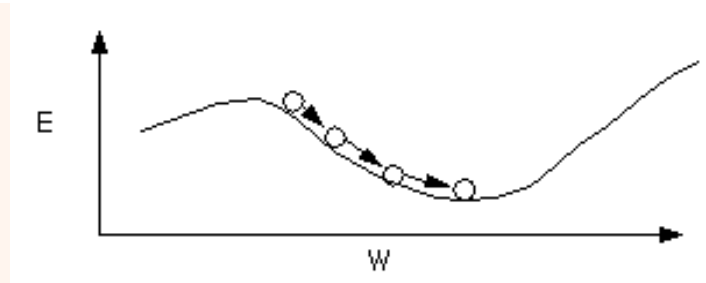
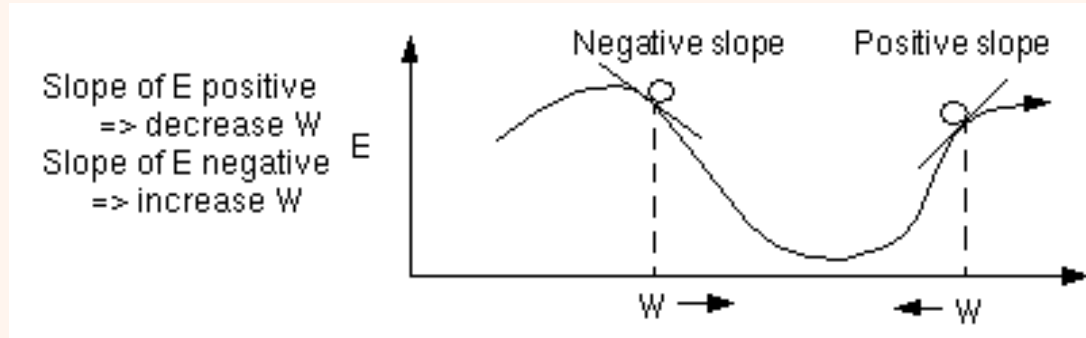
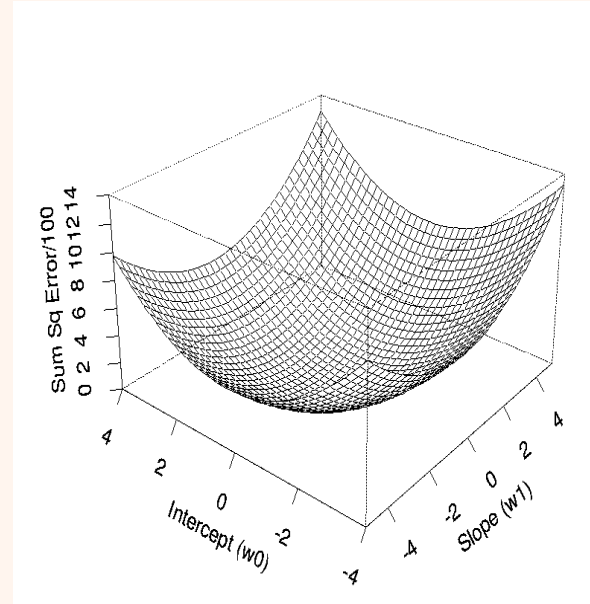
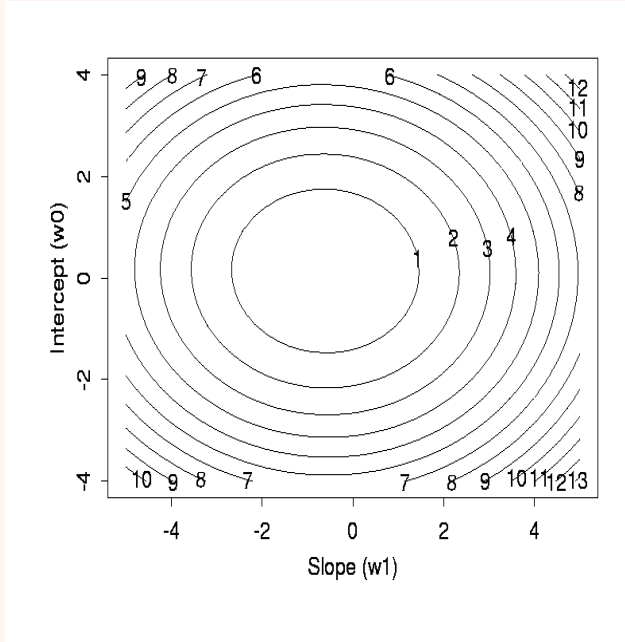
همه را این کند است.



بیتم ناپیدار است.



Steepest descent (کمینه کردن خطا) (ادامه...)



انواع شیوه‌های آموزش

- آموزش به دو صورت قابل انجام است:

Batch

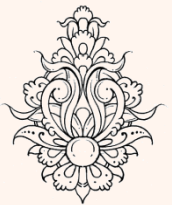
- **دسته‌ای:**

در این شیوه تابع خطایی که بنیاست مورد استفاده قرار گیرد، تابع خطا به ازای اعمال همه‌ی ورودی‌هاست.

Sequential

- **ترتیبی:**

در این شیوه ورودی‌های تک‌تک اعمال شده، پس از محاسبه‌ی خطا، پارامترها اصلاح شده و به همین ترتیب ورودی بعدی اعمال می‌شود.



آموزش تک‌نورون



McCulloch and Pitts 1943

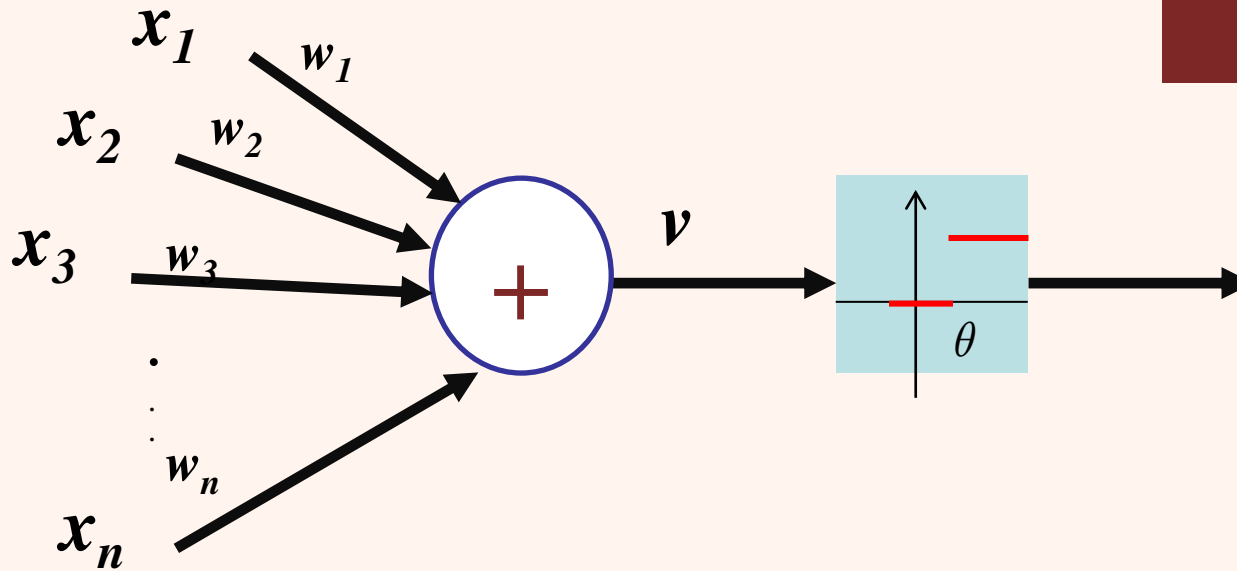
A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY

WARREN S. McCULLOCH and WALTER H. PITTS

مدل نورون

• کوچک‌ترین واحد پردازشگر اطلاعات

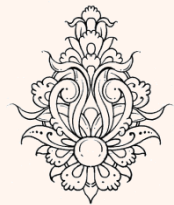
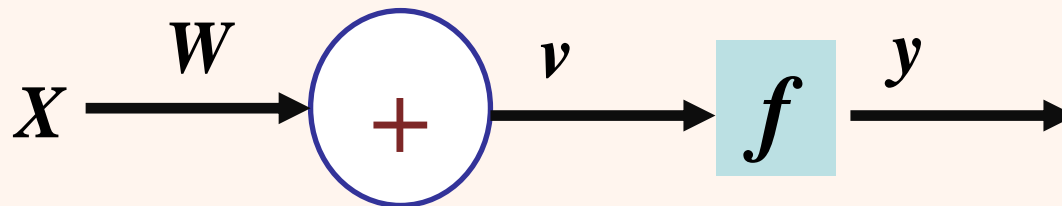
ساختار نورون



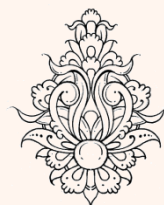
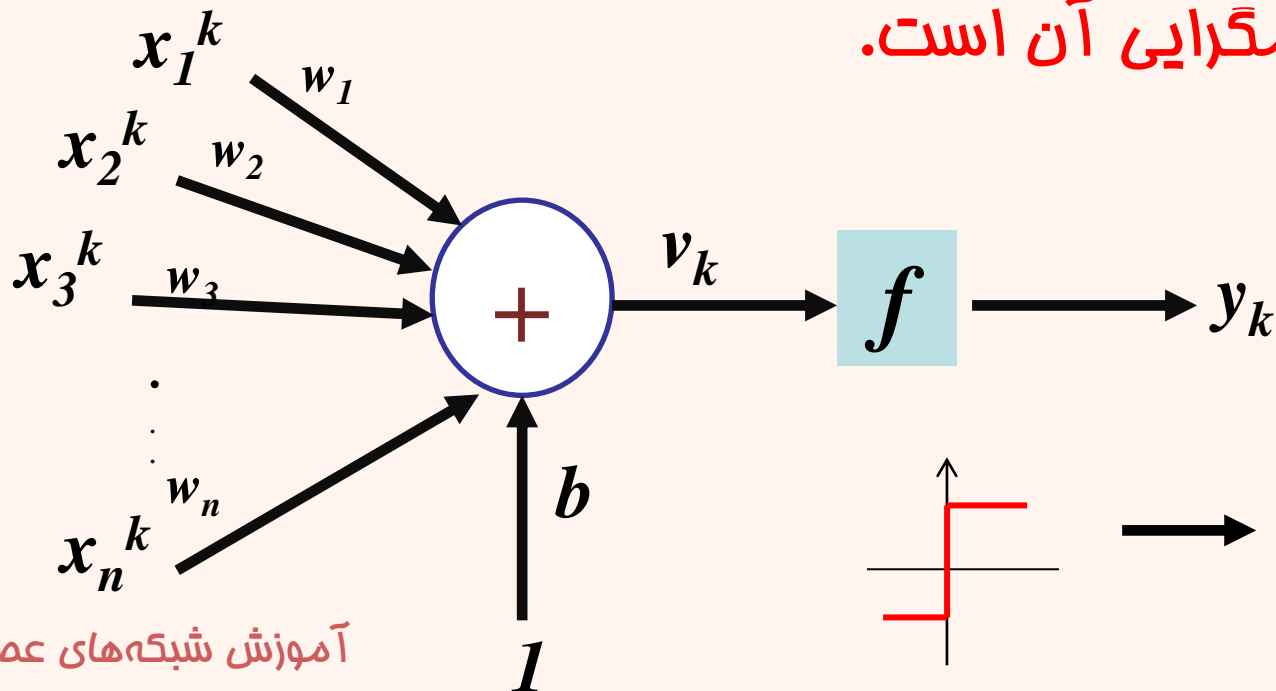
$$v = W \cdot X$$

$$y = f(v)$$

$$y = f(W \cdot X)$$

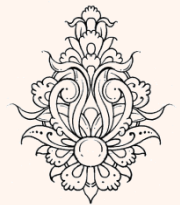
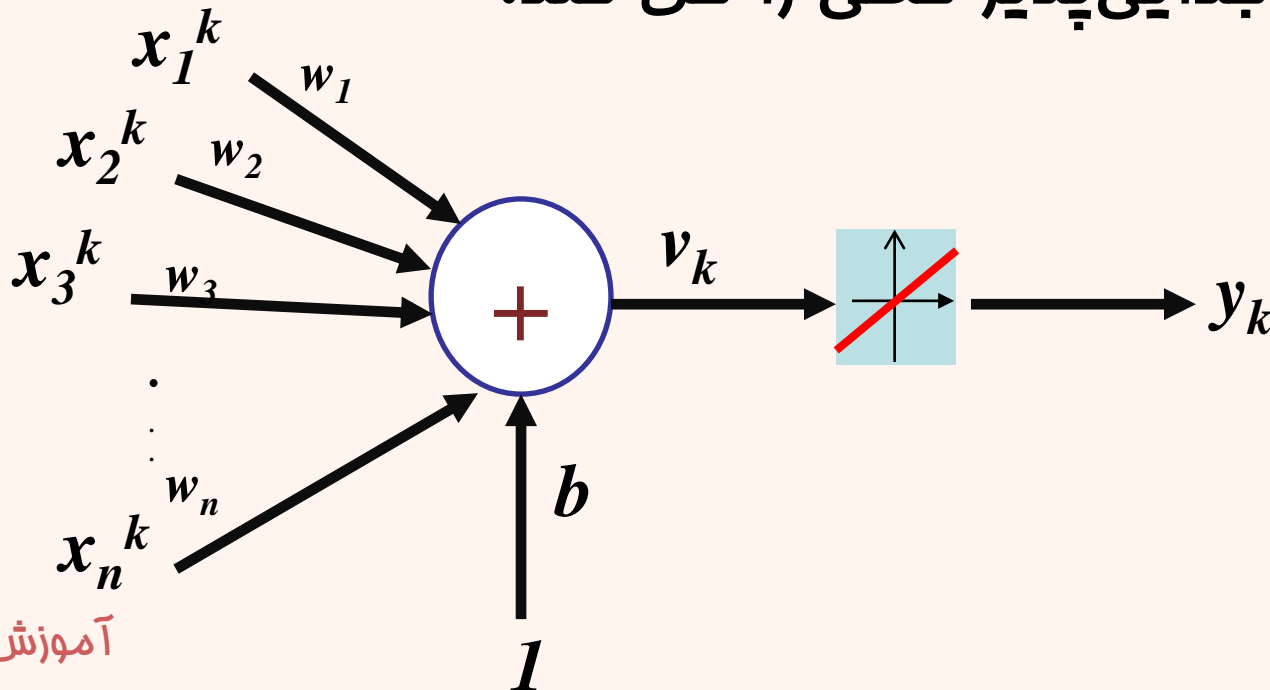


- یک پرسپترون یک بردار ورودی را گرفته، ترکیبی خطی از آن‌ها را محاسبه نموده، خروجی را فراهم می‌آورد.
- اگر خروجی از میزان آستانه‌ای بالاتر بود **یک** و در غیر این صورت **صفر** (منهای یک) باز می‌گرداند.
- **مهمترین مساله مطرح شدن قانون آموزش پرسپترون و قضیه همگرایی آن است.**



ADALINE

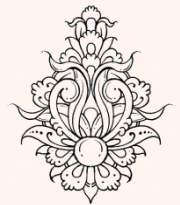
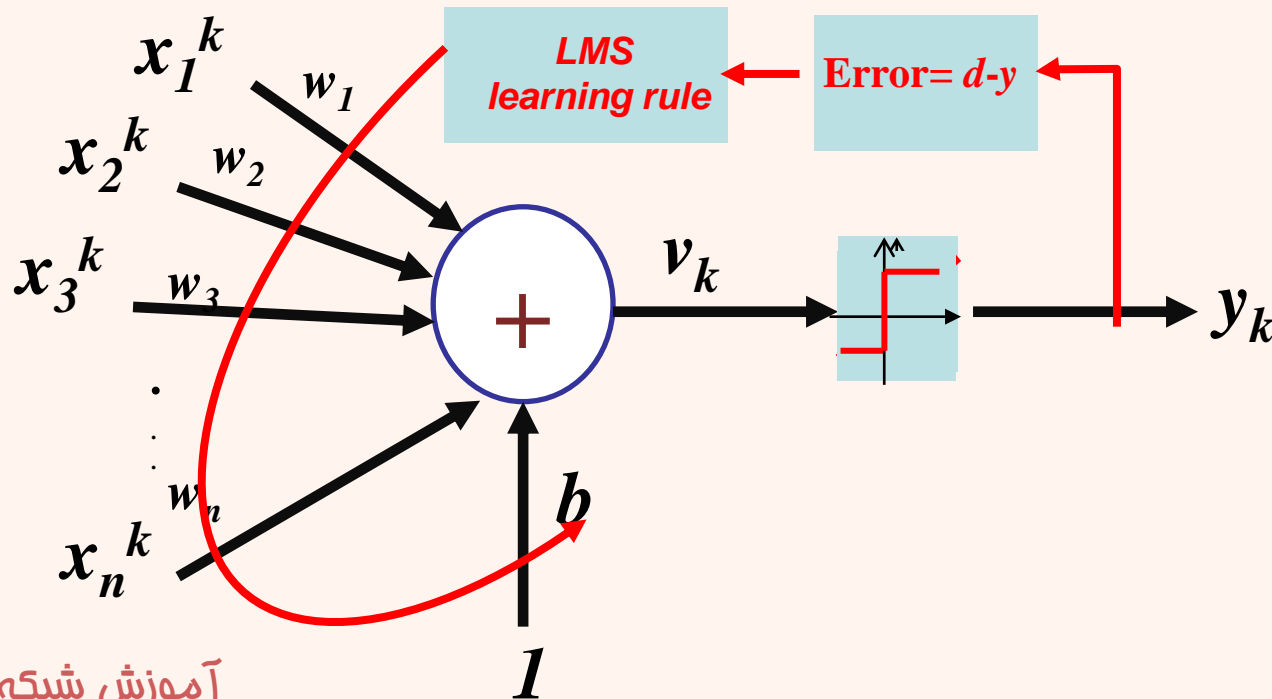
- ADALINE همانند پرسپترون است تنها تابع آن به جای دوسطحی بودن (که مقادیر ۱ و -۱- (0) را به خود اختصاص می‌دهد) تابعی خطی است.
- ADALINE همانند پرسپترون می‌تواند مسائل جدایی‌پذیر خطی را حل کند.



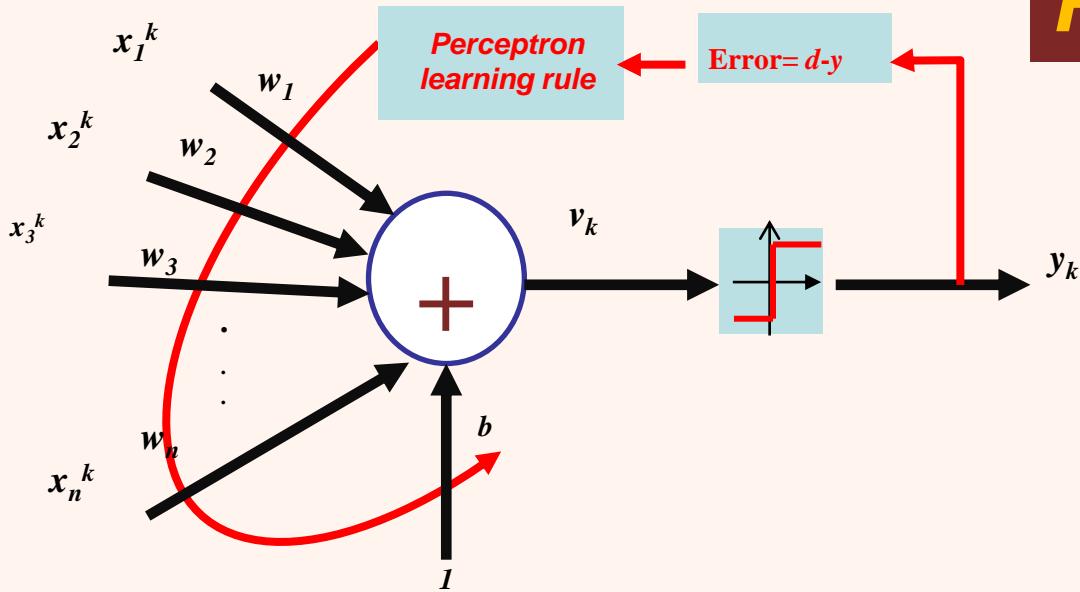
Widrow-Hoff Learning Rule

LMS(Least Mean Square)

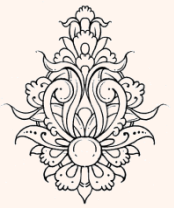
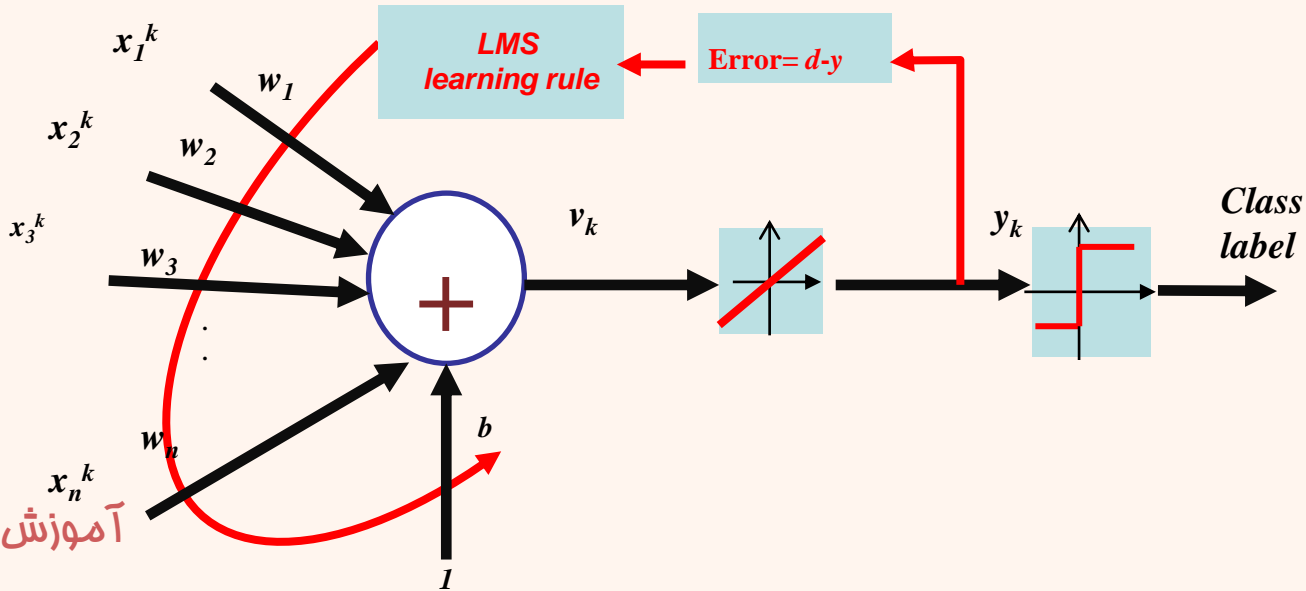
- الگوریتم **LMS** وزن‌ها و بایاس را به گونه‌ای تغییر می‌دهد که میانگین مربعات خطا (بین خروجی مطلوب و خروجی واقعی) سیستم را به حداقل برساند.



Perceptron

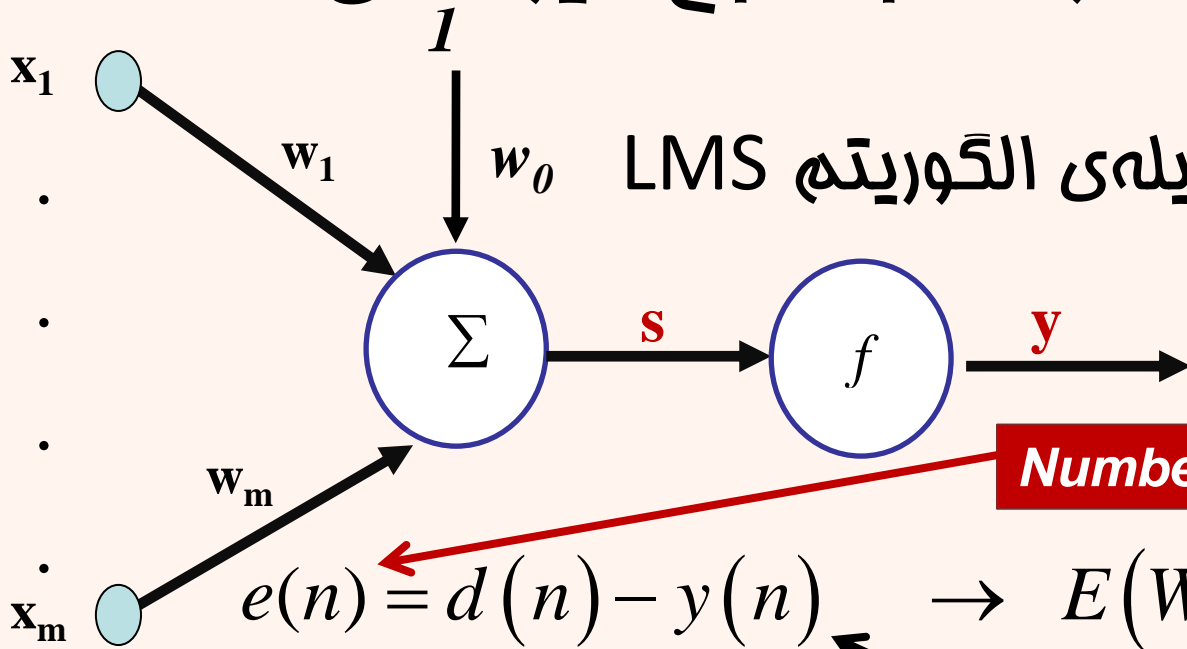


Adaline



آموزش شبکه‌های عصبی

تک‌لایه تک‌واحد با تابع غیر خطی



• حل به وسیله‌ی الگوریتم LMS

Sequential Mode

Number of iteration

$$e(n) = d(n) - y(n) \rightarrow E(W(n)) = \frac{1}{2} e^2(n)$$

فروجهی به ازای ورودی در تکرار nام

$$w_k(n+1) = w_k(n) - \eta \frac{\partial E}{\partial w_k}$$

$$\begin{aligned} \frac{\partial E}{\partial w_k} &= \frac{1}{2} \frac{\partial e^2}{\partial w_k} = e \frac{\partial e}{\partial w_k} = e \frac{\partial e}{\partial y} \cdot \frac{\partial y}{\partial w_k} \\ &= e \frac{\partial e}{\partial y} \cdot \frac{\partial y}{\partial s} \cdot \frac{\partial s}{\partial w_k} \end{aligned}$$

Chain rule



$$\frac{\partial E}{\partial w_k} = e \frac{\partial e}{\partial y} \frac{\partial y}{\partial s} \frac{\partial s}{\partial w_k}$$

-1

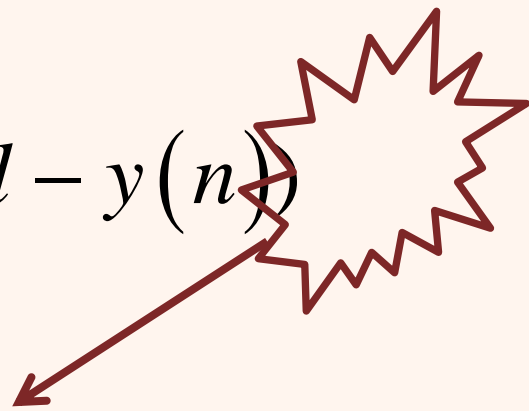
$$y = f(s) \rightarrow \frac{\partial y}{\partial s} = f'(s)$$

$$= -ef'(s)x_k$$

تابع انگیزش باید مشتق پذیر باشد



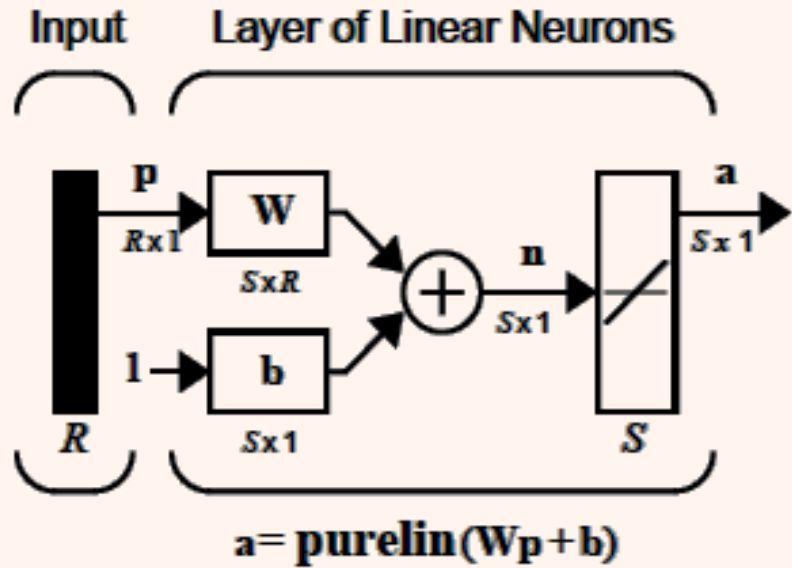
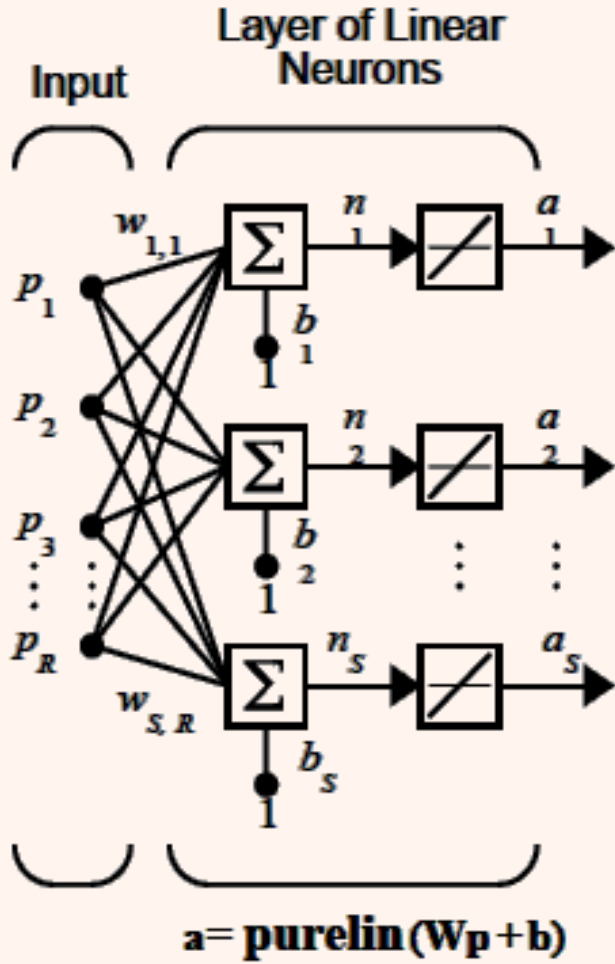
$$w_k(n+1) = w_k(n) + (d - y(n))x_k$$



بسته به تابع f متفاوت است آموزش شبکه‌های عصبی

شبکه‌ی تک‌لایه با چند خروجی

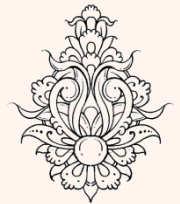
Single-Layer Linear Network

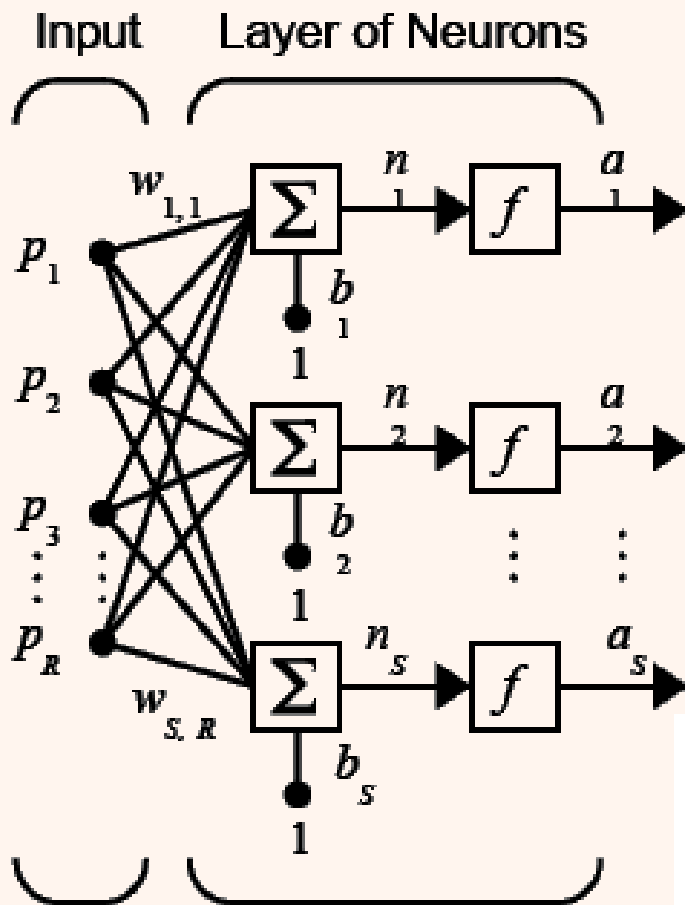


Where...

R = number of elements in input vector

S = number of neurons in layer





R

تعداد المان های ورودی

S

تعداد نرون های موجود در یک لایه

$$\mathbf{a} = \mathbf{f}(\mathbf{Wp} + \mathbf{b})$$

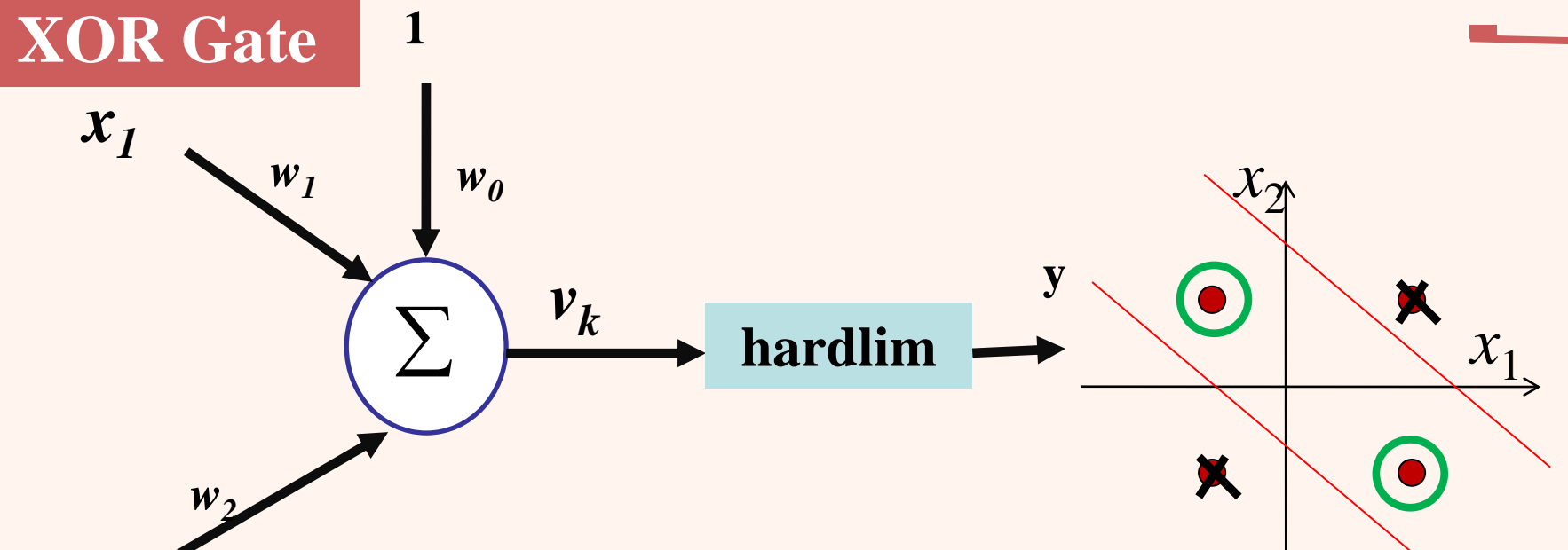
$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \dots & w_{1,R} \\ w_{2,1} & w_{2,2} & \dots & w_{2,R} \\ \vdots & \vdots & \ddots & \vdots \\ w_{S,1} & w_{S,2} & \dots & w_{S,R} \end{bmatrix}$$



الگوریتم پس انتشار خطا

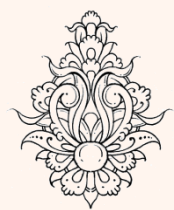


XOR Gate



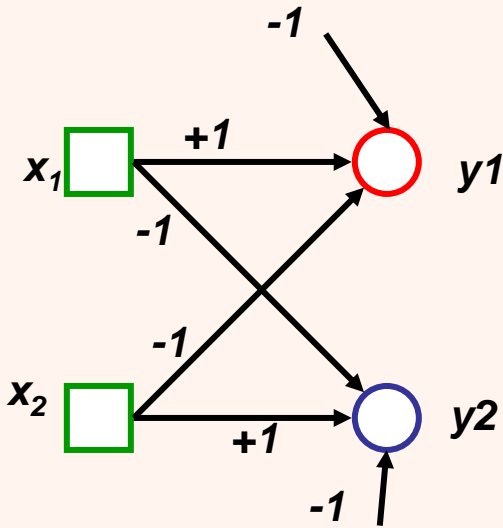
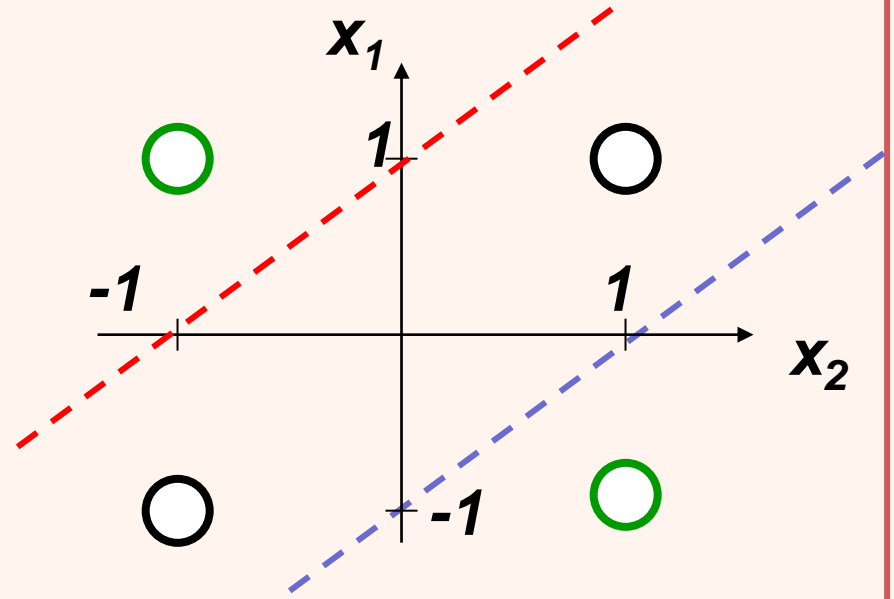
x_1	x_2	y
1	1	-1
-1	-1	-1
1	-1	1
-1	1	1

- $W_1 + W_2 + W_0 < 0$
- $W_0 < 0$
- $W_1 + W_0 > 0$
- $W_2 + W_0 > 0$



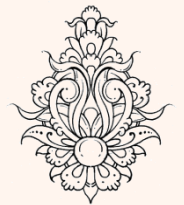
مثال

x_1	x_2	$x_1 \text{ XOR } x_2$
-1	-1	-1
-1	1	1
1	-1	1
1	1	-1



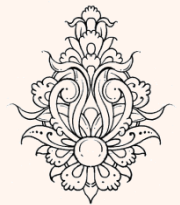
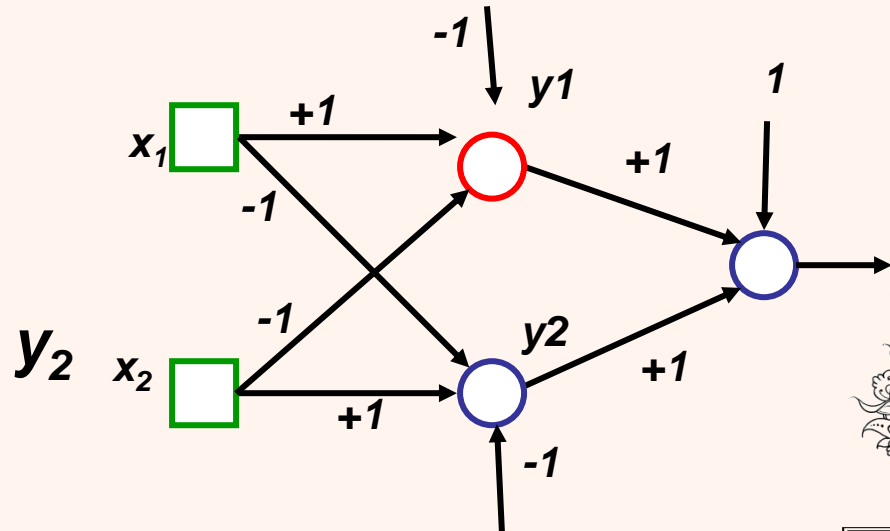
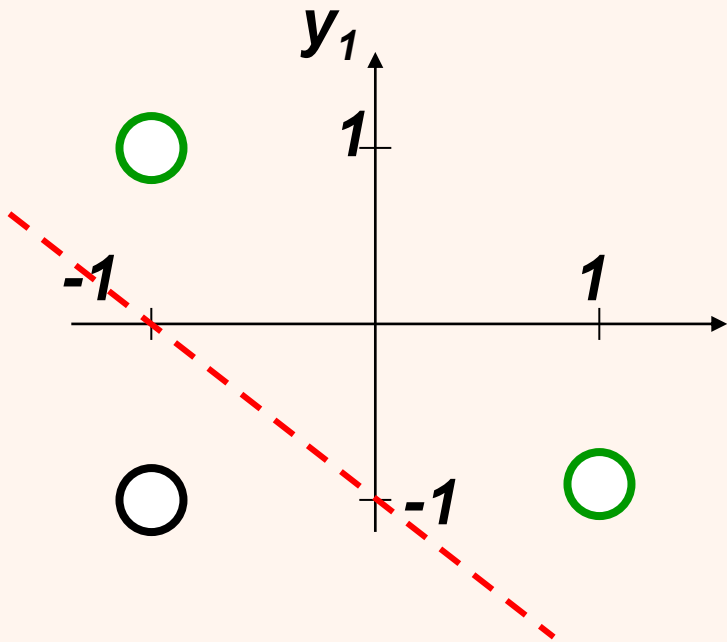
$$\varphi(v) = \begin{cases} 1 & \text{if } v > 0 \\ -1 & \text{if } v \leq 0 \end{cases}$$

φ is the sign function.



مثال (ادامه...)

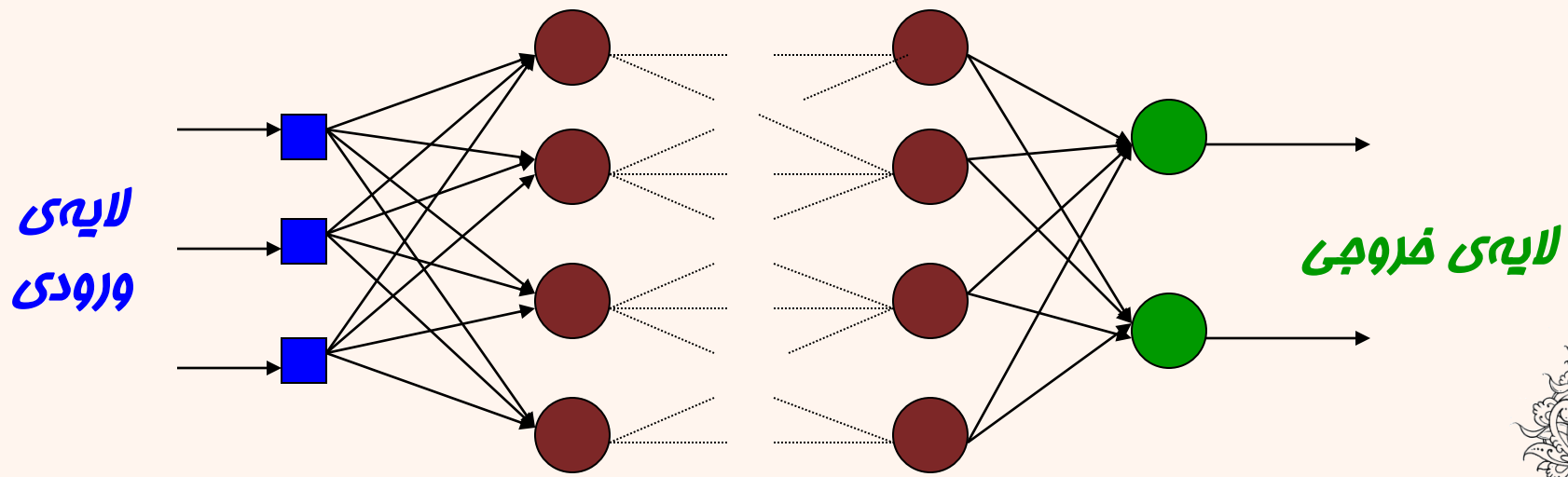
x_1	x_2	y_1	y_2	$x_1 \text{ XOR } x_2$
-1	-1	-1	-1	-1
-1	1	-1	1	1
1	-1	1	-1	1
1	1	-1	-1	-1



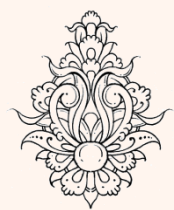
Minsky & Papert (1969) offered solution to XOR problem by combining perceptron unit responses using a second layer of units

آموزش شبکه‌های عصبی

Multilayer Perceptron (MLP)



لایه های پنهان

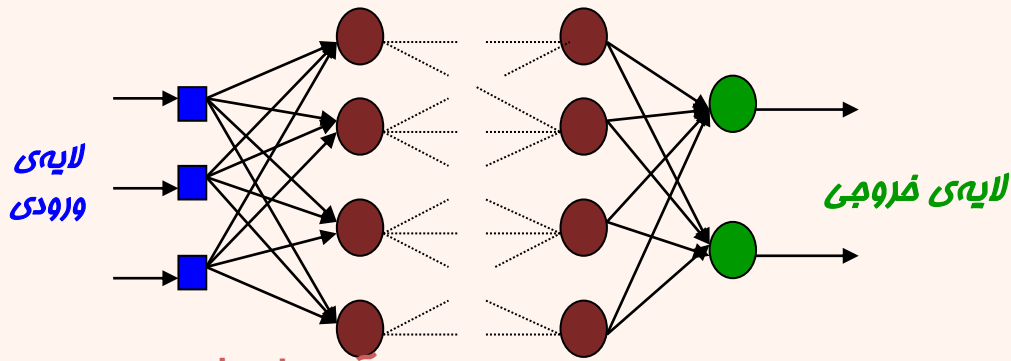


شبکه‌ی عصبی چند لایه (ادامه...)

- ورودی‌ها به صورت مستقیم به خروجی متصل نیستند.
- هر واحد از لایه‌ی قبلی به تمامی واحدهای لایه‌ی بعدی متصل است. (وزن صفر مجاز است)
- تعداد لایه‌ها و همچنین تعداد واحدهای مخفی در هر لایه باید از پیش مشخص باشد.
- تمام اتصالات **رو به جلو** است.
- تابع فعال‌سازی نورون‌ها (جز لایه خروجی) باید تابعی **غیرخطی** باشد.

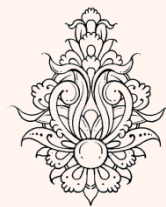
Fully connected

Feed Forward



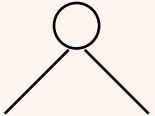
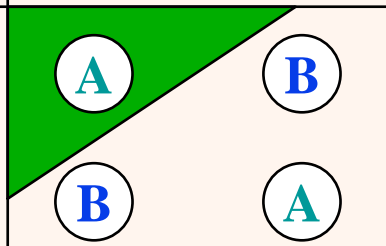
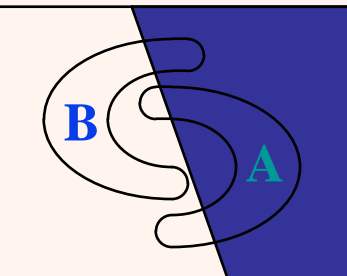
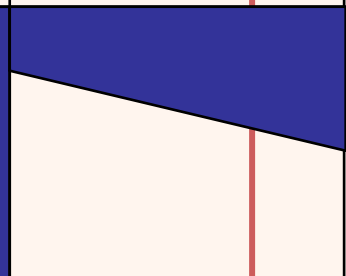
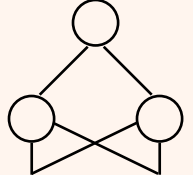
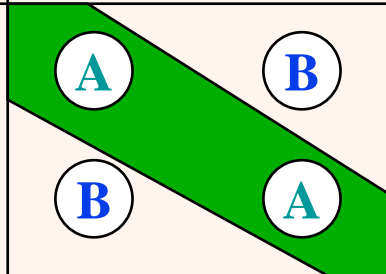
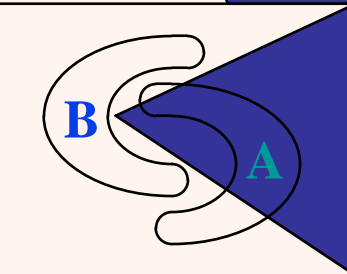
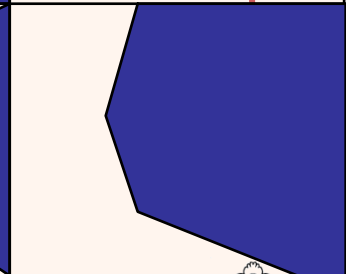
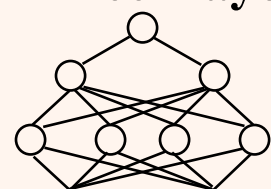
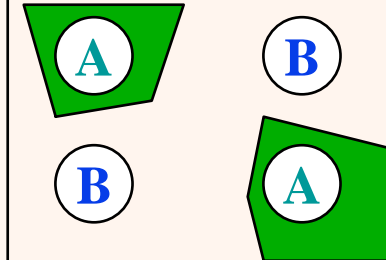
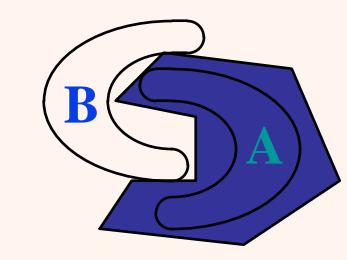
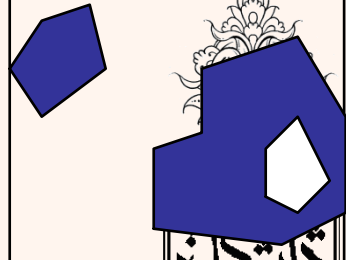
آموزش شبکه‌های عصبی

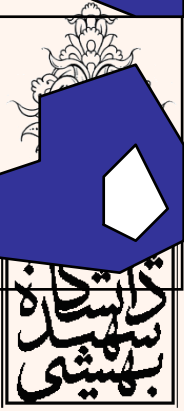
لایه‌های مخفی



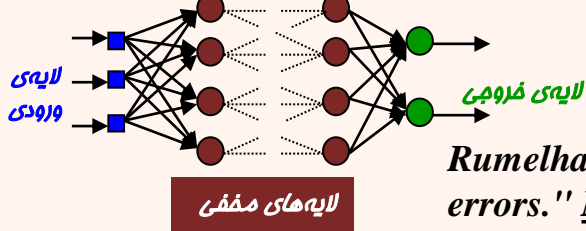
شبکه عصبی چند لایه

An introduction to computing with neural nets (Lippmann, R. P. 1987)

Structure	Types of Decision Regions	Exclusive-OR Problem	Classes with Meshed regions	Most General Region Shapes
Single-Layer 	Half Plane Bounded By Hyperplane			
Two-Layer 	Convex Open Or Closed Regions			
Three-Layer 	Arbitrary (Complexity Limited by No. of Nodes)			



یادگیری



Rumelhart, D. E., et al. (1986). "Learning representations by back-propagating errors." *Nature* 323(6088): 533-536.

- برای آموزش این شبکه‌ها از الگوریتم «پس‌انتشار

Back Propagation

خطا» استفاده می‌شود.

- برای آموزش این دست شبکه‌ها به طریقه‌ی زیر عمل می‌شود:

– رو به جلو (Forward)

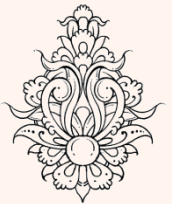
- بردار ورودی به شبکه اعمال شده و خروجی واقعی محاسبه می‌شود.

– رو به عقب (Backward)

- خطا (خروجی واقعی - خروجی مطلوب) محاسبه شده و بر حسب تابع معیار، سیگنالی متناسب با خطا تولید می‌شود. این سیگنال لایه لایه حرکت کرده و وزن‌ها را تا لایه‌ی ورودی

sequential mode

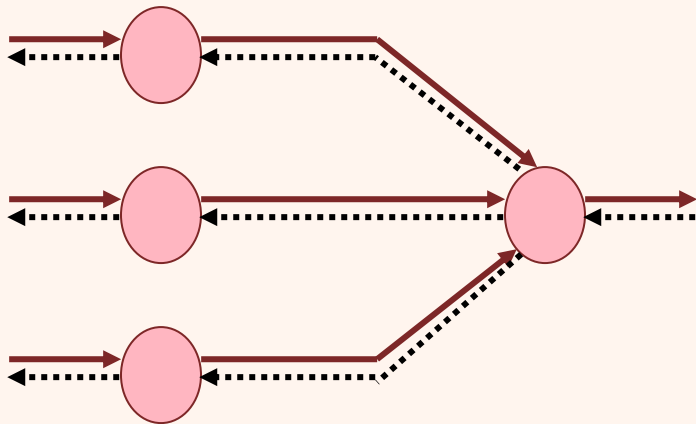
پس از اصلاح وزن‌ها می‌گوییم یک iteration صورت گرفته است.



عملکرد شبکه

• فرض شبکه

– m لایه بدون در نظر گرفتن لایه ورودی



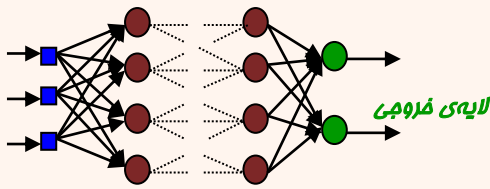
—→ **Function signals**
Forward Step

←..... **Error signals**
Backward Step



وزن‌ها به گونهای اصلاح می‌شوند که تابع خطا (به عنوان مثال میانگین مجموع مربعات) خطا کمینه گردد

لایه‌ی ورودی



لایه‌ی خروجی

میزان خطا

sequential mode

- میزان خطا و میانگین مربعات خطا برای نورون‌ها در تکرار n ام (به ازای ورودی n -ام) به شیوه‌ی زیر محاسبه می‌شود:

$$e_j(n) = d_j(n) - y_j(n)$$

خطای خروجی نورون‌ها
(گره‌ی خروجی)

sequential mode

$$E(n) = \frac{1}{2} \sum_{j \in C} e_j^2(n)$$

instantaneous error energy

$$E_{AV} = \frac{1}{N} \sum_{n=1}^N E(n)$$

averaged squared error energy

Batch mode

هدف آموزش
کمینه نمودن

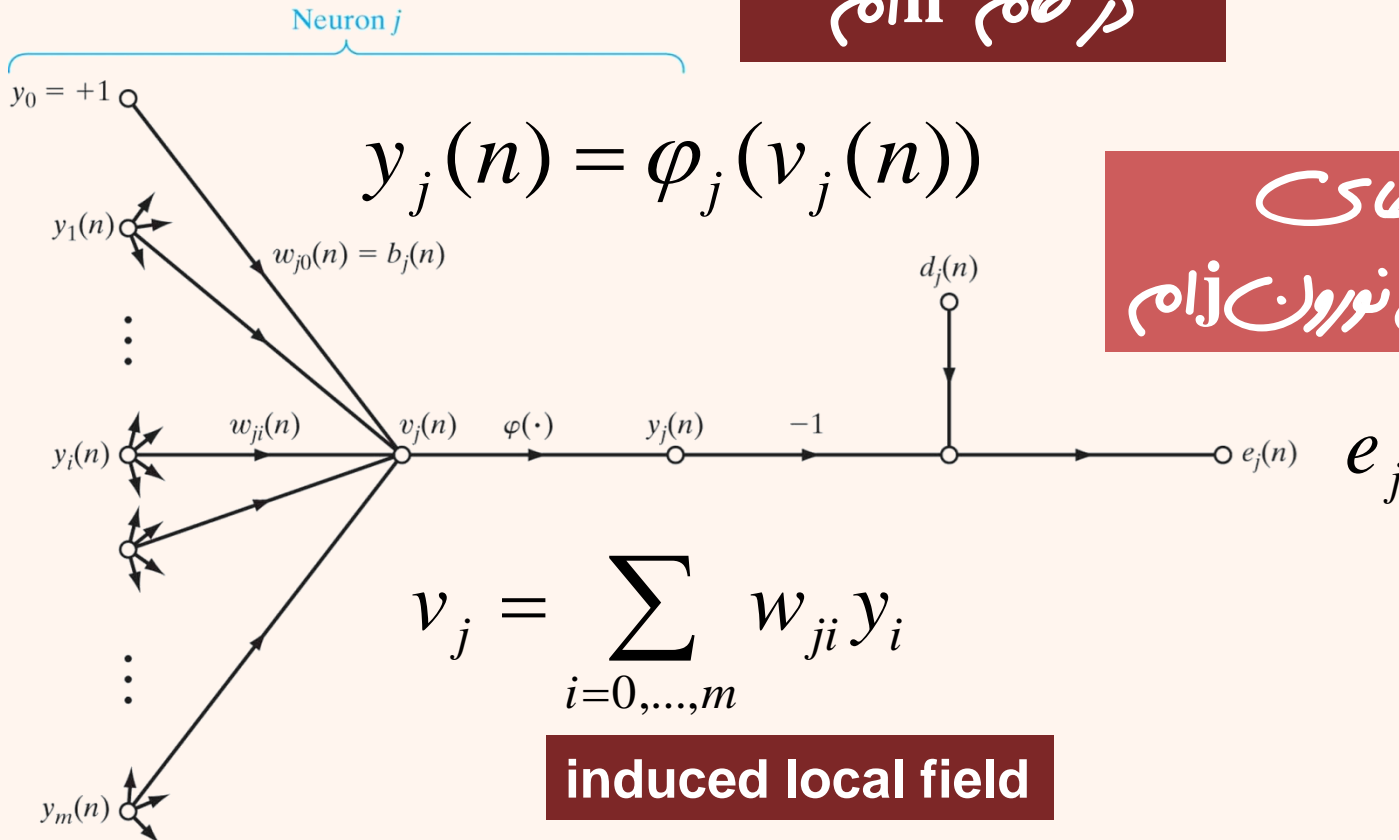
E_{AV}



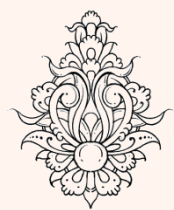
آموزش شبکه‌های چندلایه

- به روز نمودن وزن‌ها همانند LMS است:

خروجی نورون زام
در گام nام

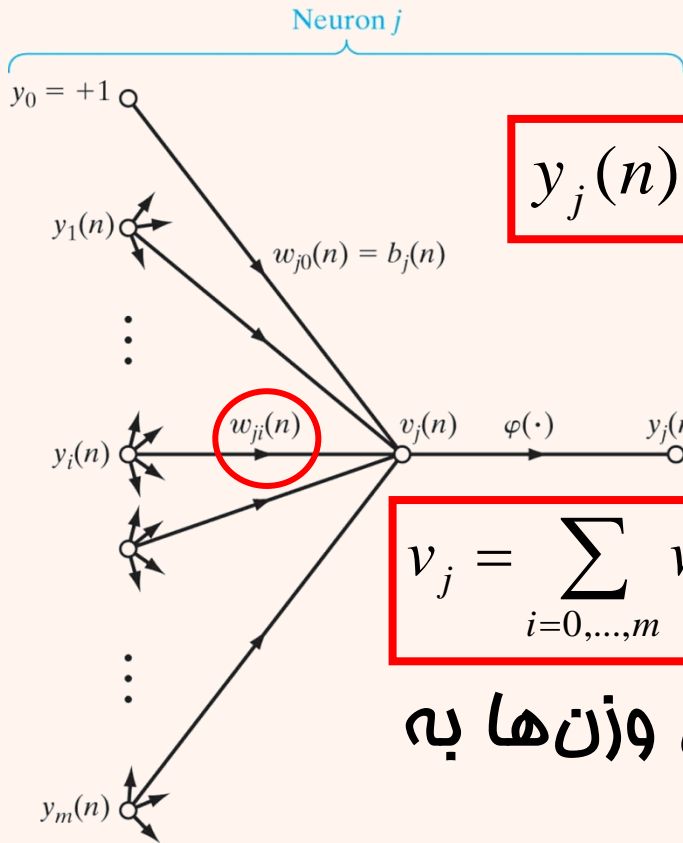


خطای
خروجی نورون زام



آموزش لایه‌ی خروجی

sequential mode



$$y_j(n) = \varphi_j(v_j(n))$$

$$E(n) = \frac{1}{2} \sum_{j \in C} e_j^2(n)$$

$$v_j = \sum_{i=0, \dots, m} w_{ji} y_i$$

$$e_j(n) = d_j - y_j(n)$$

• همانند LMS برای به روز کردن وزن‌ها به شیوه‌ی زیر عمل می‌کنیم:

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = \frac{\partial E(n)}{\partial e_j(n)} \cdot \frac{\partial e_j(n)}{\partial y_j(n)} \cdot \frac{\partial y_j(n)}{\partial v_j(n)} \cdot \frac{\partial v_j(n)}{\partial w_{ji}(n)}$$

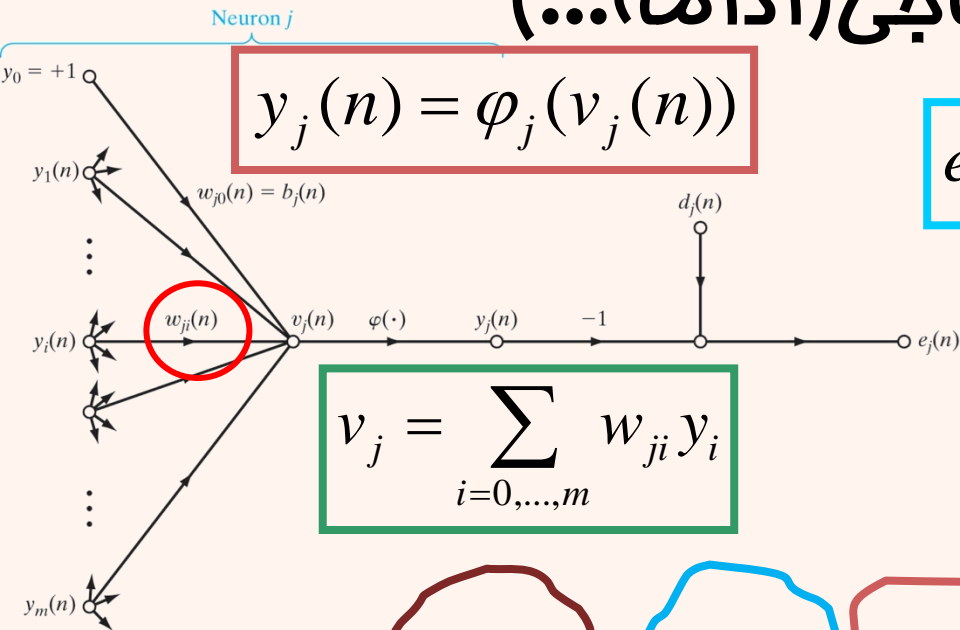
Chain rule



sensitivity factor

آموزش شبکه‌های عصبی

آموزش لایه خروجی (ادامه...)



$$y_j(n) = \varphi_j(v_j(n))$$

$$e_j(n) = d_j - y_j(n)$$

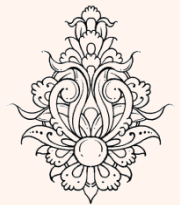
$$E(n) = \frac{1}{2} \sum_{j \in C} e_j^2(n)$$

$$v_j = \sum_{i=0, \dots, m} w_{ji} y_i$$

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = \frac{\partial E(n)}{\partial e_j(n)} \cdot \frac{\partial e_j(n)}{\partial y_j(n)} \cdot \frac{\partial y_j(n)}{\partial v_j(n)} \cdot \frac{\partial v_j(n)}{\partial w_{ji}(n)}$$

$e_j(n)$ -1 $\phi'_j(v_j(n))$ $y_i(n)$

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = -e_j(n) \phi'_j(v_j(n)) y_i(n)$$



آموزش لایه خروجی (ادامه...)

• از قانون **دلتا** داشتیم:

$$\Delta w_{ji}(n) = -\eta \frac{\partial E(n)}{\partial w_{ji}(n)}$$

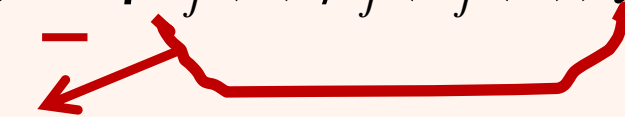
Step in direction opposite to the gradient

$$\frac{\partial E(n)}{\partial w_{ji}(n)} = -e_j(n) \phi'_j(v_j(n)) y_i(n)$$

$$\Delta w_{ji}(n) = \eta e_j(n) \phi'_j(v_j(n)) y_i(n)$$

گرادینان محلی

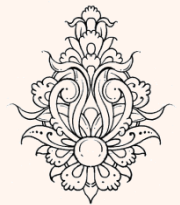
$$\delta_j(n)$$



گرادینان محلی برای یک نورون به فضای ایجاد شده برای آن نورون و مشتق تابع مرتباً آن بستگی دارد

$$\delta_j(n) = -\frac{\partial E(n)}{\partial v_j(n)}$$

$$= -\frac{\partial E(n)}{\partial e_j(n)} \cdot \frac{\partial e_j(n)}{\partial y_j(n)} \cdot \frac{\partial y_j(n)}{\partial v_j(n)} = e_j(n) \phi'_j(v_j(n))$$



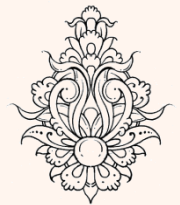
آموزش لایه خروجی (ادامه...) $e_j = d_j - y_j$

$$\Delta w_{ji}(n) = \eta e_j(n) \varphi'_j(v_j(n)) y_i(n)$$

$$\delta_j(n)$$

$$\delta_j = (d_j - y_j) \varphi'(v_j)$$

$$\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n)$$



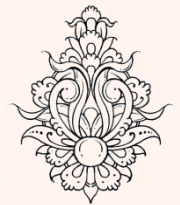
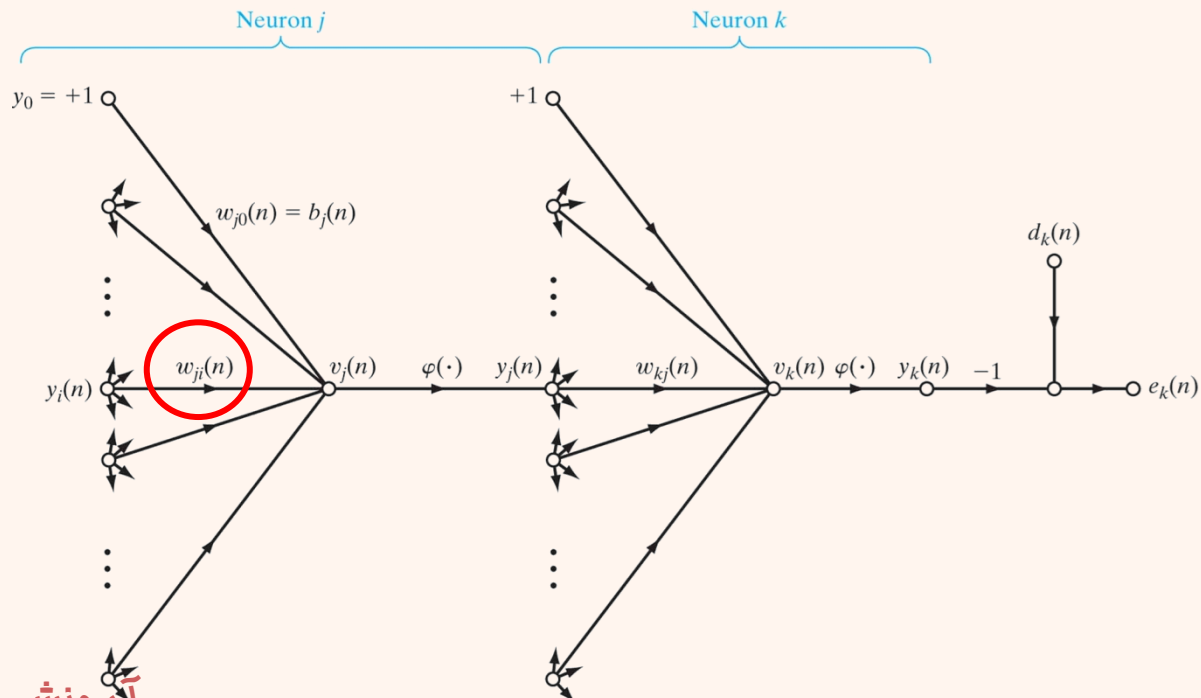
دانشگاه

در ادامه به شیوه آموزش لایه مخفی خواهیم پرداخت

آموزش لایه مخفی

- اگر نورون ز را نورونی از لایه مخفی در نظر بگیریم برای محاسبه خطا:

– برای محاسبه گرادیان محلی نورون مورد نظر می‌باید تمامی گرادیان‌های محلی نورون‌هایی که با نورون مورد نظر در ارتباط هستند را لحاظ نماییم.

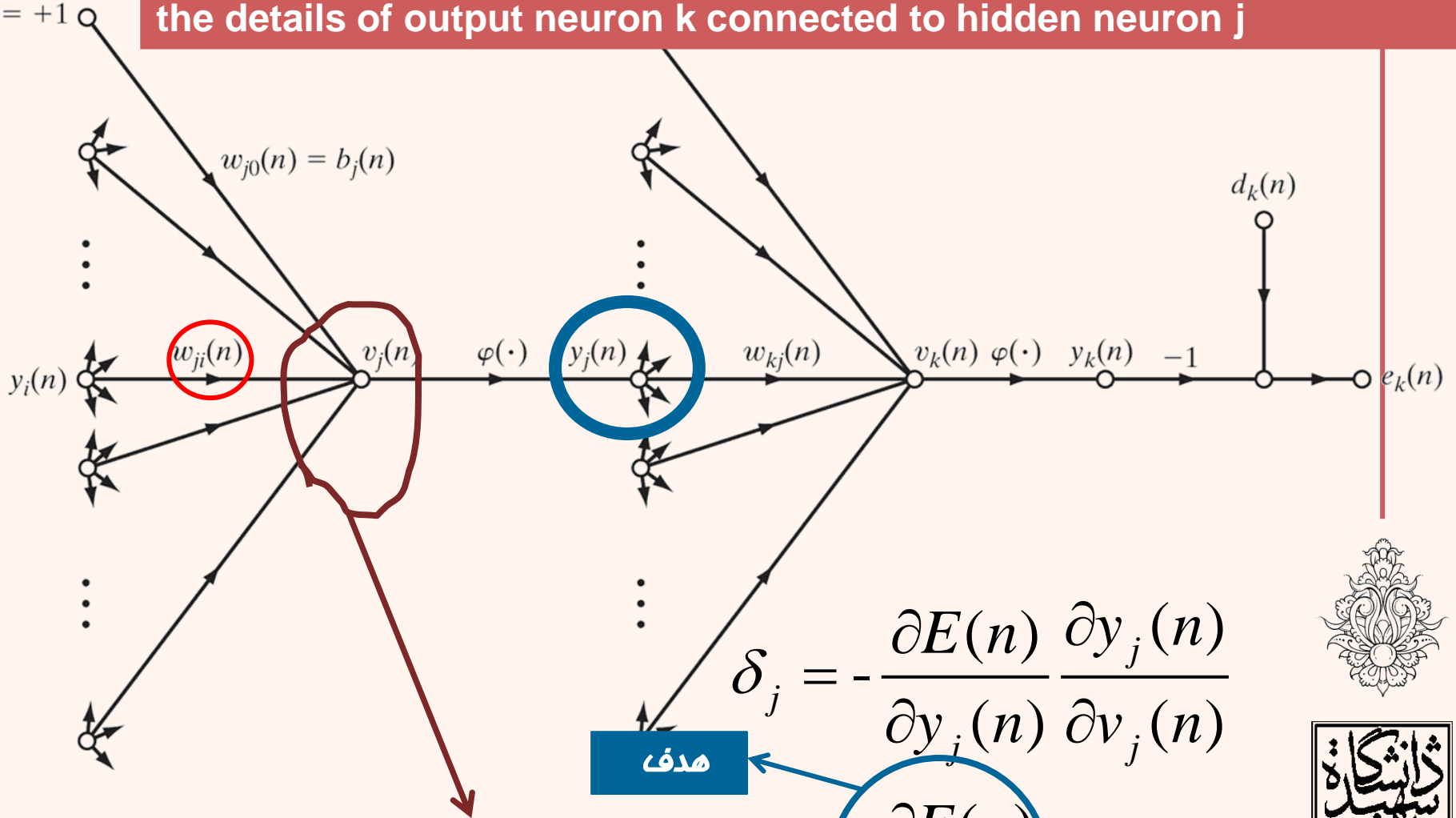


آموزش لایه مخفی (ادامه...)

Neuron j

Neuron k

the details of output neuron k connected to hidden neuron j



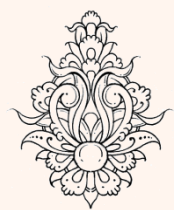
$$\delta_j = - \frac{\partial E(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)}$$

$$= - \frac{\partial E(n)}{\partial y_j(n)} \varphi'_j(v_j(n))$$

هدف

نورونز متعلق به لایه مخفی است

آموزش شبکه‌های عصبی



آموزش لایه مخفی (ادامه...)

$$\delta_j = - \frac{\partial E(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)}$$

$$= - \frac{\partial E(n)}{\partial y_j(n)} \varphi_j'(v_j(n))$$

$$E(n) = \frac{1}{2} \sum_{k \in C} e_k^2(n)$$

نورون کایک
node خروجی
است

$$\frac{\partial E(n)}{\partial y_j(n)} = \sum_k e_k \frac{\partial e_k(n)}{\partial y_j(n)}$$

$$v_k(n) = \sum_{j=0}^m w_{kj}(n) y_j(n)$$

$$\frac{\partial E(n)}{\partial y_j(n)} = \sum_k e_k \left(\frac{\partial e_k(n)}{\partial v_k(n)} \frac{\partial v_k(n)}{\partial y_j(n)} \right)$$

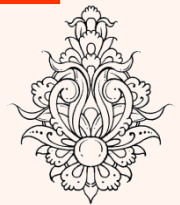
$w_{kj}(n)$

$$e_k(n) = d_k(n) - y_k(n)$$

$$-\varphi_k'(v_k(n))$$

$$d_k(n) - \varphi_k(v_k(n))$$

آموزش شبکه‌های عصبی



آموزش لایه مخفی (ادامه...)

$$\frac{\partial E(n)}{\partial y_j(n)} = \sum_k e_k \frac{\partial e_k(n)}{\partial v_k(n)} \frac{\partial v_k(n)}{\partial y_j(n)} \quad \rightarrow w_{kj}(n)$$

$$-\phi'_k(v_k(n))$$

$$\frac{\partial E(n)}{\partial y_j(n)} = - \sum_k e_k(n) \phi'_k(v_k(n)) w_{kj}(n)$$

$$= - \sum_k \delta_k(n) w_{kj}(n)$$



$$\delta_j = \phi'_j(v_j(n)) \sum_k \delta_k(n) w_{kj}(n)$$

$$\delta_j = - \frac{\partial E(n)}{\partial y_j(n)} \frac{\partial y_j(n)}{\partial v_j(n)}$$

$$= - \frac{\partial E(n)}{\partial y_j(n)} \phi'_j(v_j(n))$$

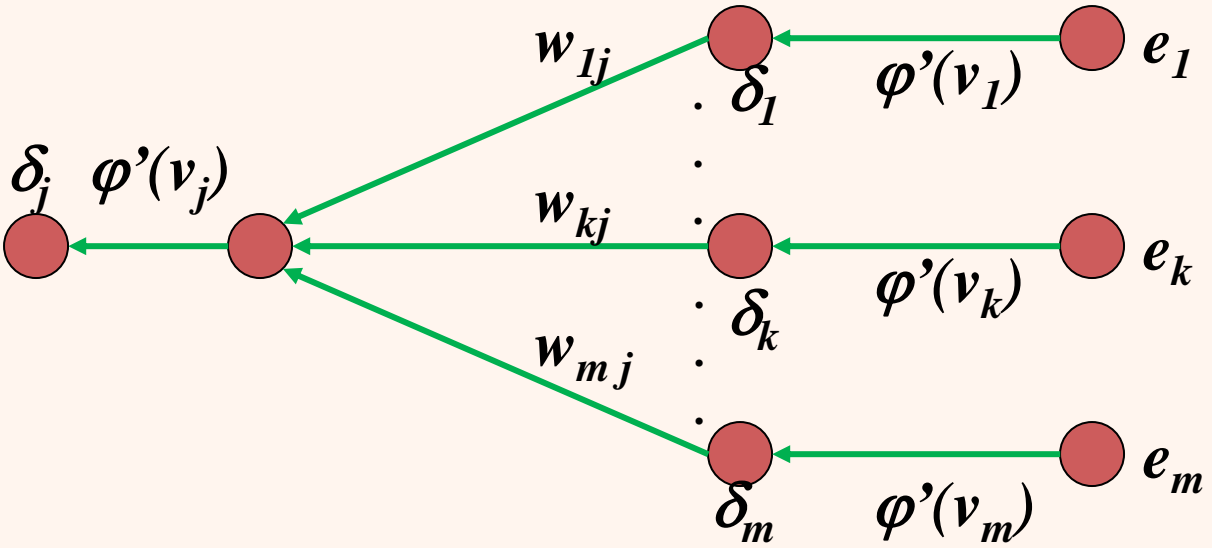


آموزش لایه مخفی *Local Gradient of hidden neuron j*

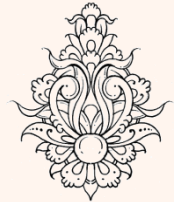
$$\delta_j = \varphi'_j(v_j) \sum_{k \in C} \delta_k w_{kj}$$

n شماره Iteration

Signal-flow graph of back-propagation error signals to neuron *j*



$$\begin{pmatrix} \text{Weight} \\ \text{Correction} \\ \Delta w_{ij}(n) \end{pmatrix} = \begin{pmatrix} \text{learning} \\ \text{rate} \\ \eta \end{pmatrix} \cdot \begin{pmatrix} \text{local} \\ \text{gradient} \\ \delta_j(n) \end{pmatrix} \cdot \begin{pmatrix} \text{input} \\ \text{of neuron } j \\ y_i(n) \end{pmatrix}$$



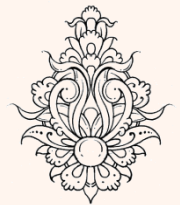
در این شیوه تمام نمونه‌های آموزشی اعمال شده، سپس اصلاح وزن‌ها صورت می‌پذیرد.

$$E_{AV} = \frac{1}{N} \sum_{n=1}^N E(n)$$

$$\Delta w_{ji} = -\eta \frac{\partial E_{AV}}{\partial w_{ji}}$$

$$\frac{\partial E_{AV}}{\partial w_{ji}} = \frac{\partial E_{AV}}{\partial E} \cdot \frac{\partial E}{\partial w_{ji}}$$

$$= \frac{1}{N} \sum_{n=1}^N \frac{\partial E(n)}{\partial w_{ji}}$$



انواع شیوه‌های آموزش

Sequential Mode

pattern or **stochastic** mode

online

• شیوهی ترتیبی:

– در این شیوه نمونه‌ها تک‌تک برای اصلاح وزن‌ها به کار می‌روند.

یک دوره کامل ارائه‌ی نمونه‌های آموزشی در فراآیند آموزش را **epoch** می‌نامند.

• شیوهی دسته‌ای: **Batch Mode**

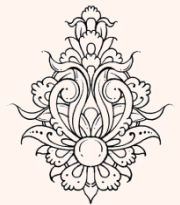
– در این شیوه تمام نمونه‌های آموزشی اعمال شده، سپس اصلاح وزن‌ها صورت می‌پذیرد.

$$E_{AV} = \frac{1}{2N} \sum_{n=1}^N \sum_{j \in C} e_j^2(n) \quad \Delta w_{ji} = -\eta \frac{\partial E_{AV}}{\partial w_{ji}}$$

$$\Delta w_{ji} = -\frac{\eta}{N} \sum_{n=1}^N e_j(n) \frac{\partial e_j(n)}{\partial w_{ji}}$$

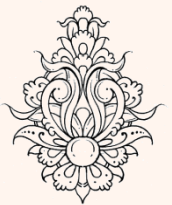
آموزش شبکه‌های عصبی

مماسه‌ی این بخش به همان شیوه‌ای که پیش از این گفته شد، انجام می‌شود



انواع شیوه‌های آموزش (ادامه...)

- شیوهی ترتیبی به مافضهی کمتری احتیاج دارد.
- در صورتی که نمونه‌ها به صورت ترتیبی و تصادفی اعمال شود، احتمال این که الگوریتم در دام مینیمم محلی بیفتند، کمتر خواهد بود.
- هر چند این تصادفی بودن، تحلیل نظری شرایط همگرایی را دشوارتر می‌کند، در حالی که با استفاده از شیوهی دسته‌ای تقریب بهتری از بردار گرادیان به دست می‌آید و همگرایی به سوی مینیمم محلی تضمین شده است.
- وقتی که داده‌های تکراری داشته باشیم، روش ترتیبی به صورت مؤثرتری از داده‌های تکراری استفاده می‌کند.
- استفاده از پردازش موازی در شیوهی دسته‌ای به مراتب ساده‌است.



انواع شیوه‌های آموزش (ادامه...)

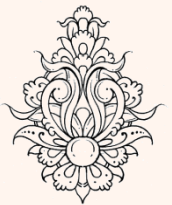
علیرغم، معایب روش‌های تدریجی، این روش در عمل ترجیح داده می‌شود، از این جهت که پیاده‌سازی آن ساده‌تر است. برای مسائل دشوار و بزرگ راه حل موثری است.

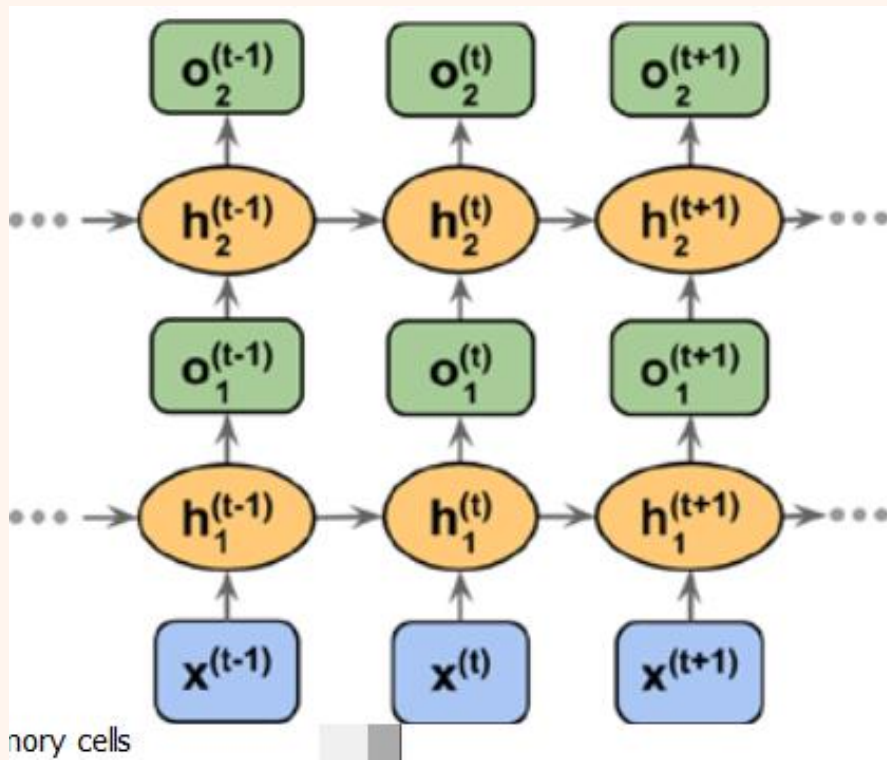
- برای بهره‌برداری از مزایای هر دو شیوه، آموزش به صورت «**minibatch**» نیز معمول است که به‌روزرسانی وزن‌ها، بعد از هر n ($n > 1$) گام صورت می‌گیرد.

Stochastic Gradient Descent (SGD)

stochastic approximation of gradient descent

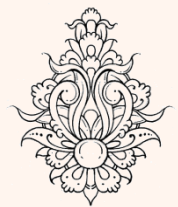
هنگامی برای بهینه‌سازی به جای بردار گرادیان تابع از تقریب آن استفاده شود (به جای استفاده از همه داده از یک زیرمجموعه که به صورت تصادفی انتخاب شده است استفاده شود)





Truncated Backpropagation through time

در این شیوه بازگشت در زمان و اصلاح وزن‌ها به صورت ممدود تا بازه‌ای خاص انجام می‌شود

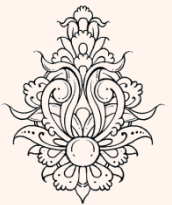


بهبود کارایی



معیارهای توقف آموزش

- در عمل، به راحتی نمی‌توان مشخص نمود که آیا فرآیند آموزش به نقطه‌ی مورد نظر رسیده است. اما ضوابطی برای پایان دادن به آموزش مطرح شده است.
- آموزش تا جایی پیش رود که اندازه‌ی بردار گرادیان از یک حد آستانه کمتر شود.
- عیب این روش این است که فرآیند آموزش ممکن است طولانی شود.
- همچنین لازم است اندازه بردار گرادیان مناسبه شود.
- آموزش زمانی متوقف می‌شود که اختلاف خطای میانگین به دست آمده در دو دوره (epoch) متوالی به اندازه‌ی کافی کوچک باشد.
- این شیوه ممکن است به توقف زودهنگام فرآیند آموزش منجر شود.
- یک شیوه‌ی دیگر این است که پس از هر فاز آموزش، شبکه با داده‌هایی غیر از داده‌های آموزشی بررسی شود و قدرت «تعمیم‌پذیری» آن ملاکی برای توقف آموزش باشد برای این بررسی لزومی ندارد از تابع خطا مشتق‌پذیر استفاده شود و می‌توان معیار دیگری به جز تابع خطا را استفاده کرد.



Early stopping

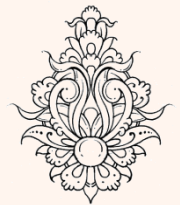
آموزش شبکه‌های عصبی

یکی از شیوه‌های کم‌هزینه‌ی پیش‌گیری از بیش‌براش توقف زودهنگام فرآیند آموزش است.

توابع فعال‌ساز

- چنانچه پیش از این مطرح شد، در شبکه‌های عصبی چندلایه، لازم است توابع فعال‌سازی نورون‌های لایه‌های پنهان **غیرخطی** باشند.
 - در صورت خطی بودن نورون‌های لایه مخفی کل شبکه به یک سیستم خطی تبدیل می‌شود.
 - معمولاً برای لایه‌های مخفی یک نوع تابع فعال‌ساز در نظر گرفته می‌شود.
- نوع تابع فعال‌ساز لایه آخر عملکرد شبکه را مشخص می‌کند.
- از طرفی برای محاسبه‌ی گرادیان محلی هر نورون، به مشتق تابع فعال‌سازی آن نیاز است. از این جهت لازم است **مشتق‌پذیر** باشند.

$$\delta_j = \begin{cases} \varphi'(v_j)(d_j - y_j) & \text{IF } j \text{ output node} \\ \varphi'(v_j) \sum_{k \in C} \delta_k w_{kj} & \text{IF } j \text{ hidden node} \end{cases}$$

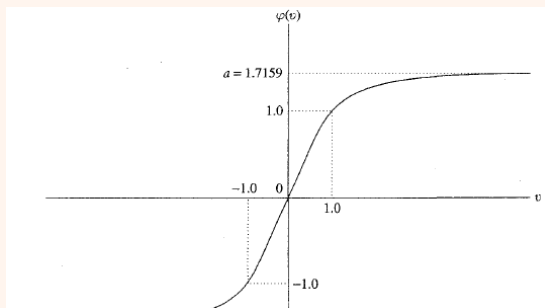


- تابع فعال سازی:

- انتخاب تابع فعال سازی مناسب در کارایی موثر است.

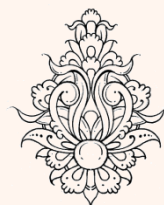
- مقدار خروجی مطلوب:

- این مقدار باید در محدوده‌ی برد تابع فعال سازی باشد، وگرنه پارامترهای آزاد به سمت بی‌نهایت می‌روند.



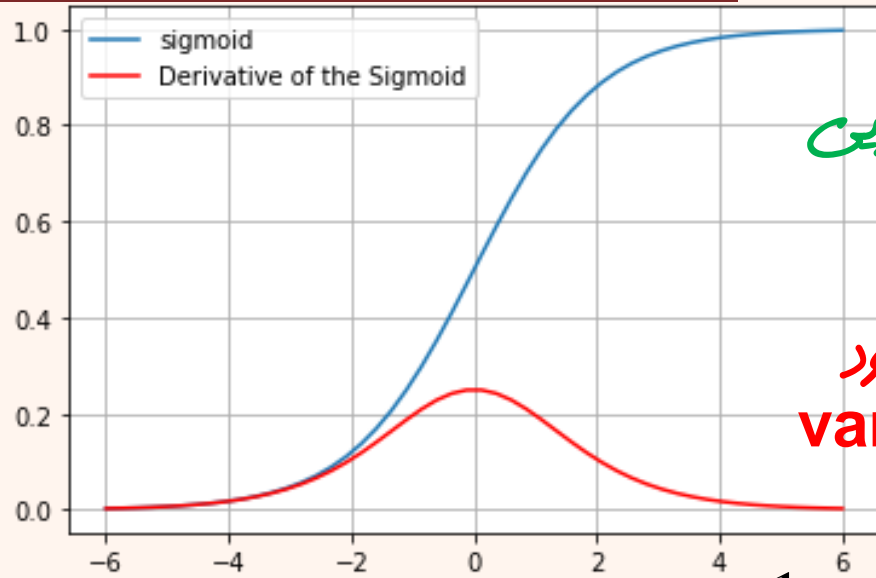
Vanishing Gradients(Exploding Gradients)

هنگامی که عمق شبکه افزایش یابد، گرادینت وزن‌ها لایه ابتدایی (نزدیک به ورودی) کوچک (بزرگ) می‌شوند.



Sigmoid function(logistic)

تابع سیگموئید



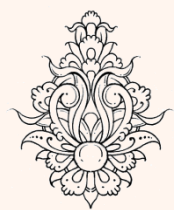
- مناسب برای دسته‌بندی دو طبقه
- نمونه پیوسته تابع پله
- در ناحیه اشباع آموزش کند می‌شود
- احتمال vanishing gradient

$$y_j(n) = \varphi_j(v_j(n)) = \frac{1}{1 + \exp(-av_j(n))} \quad a > 0$$

$$\varphi'_j(v_j(n)) = \frac{a \exp(-av_j(n))}{[1 + \exp(-av_j(n))]^2}$$

$$\varphi'_j(v_j(n)) = a \times \frac{1}{1 + \exp(-av_j(n))} \times \left[1 - \frac{1}{1 + \exp(-av_j(n))} \right]$$

$$\varphi'_j(v_j(n)) = ay_j(n) \times [1 - y_j(n)]$$



توابع فعال‌سازی (ادامه...)

Sigmoid function

- اگر نورون در لایه‌ی خروجی باشد:

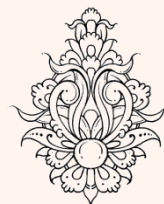
$$\delta_j(n) = e_j(n) \phi'_j(v_j(n))$$

$$\delta_j(n) = a [d_j(n) - o_j(n)] o_j(n) \times [1 - o_j(n)]$$

- و اگر در لایه‌ی مخفی باشد:

$$\delta_j = \alpha y_j(n) \times [1 - y_j(n)] \sum_k \delta_k(n) w_{kj}(n)$$

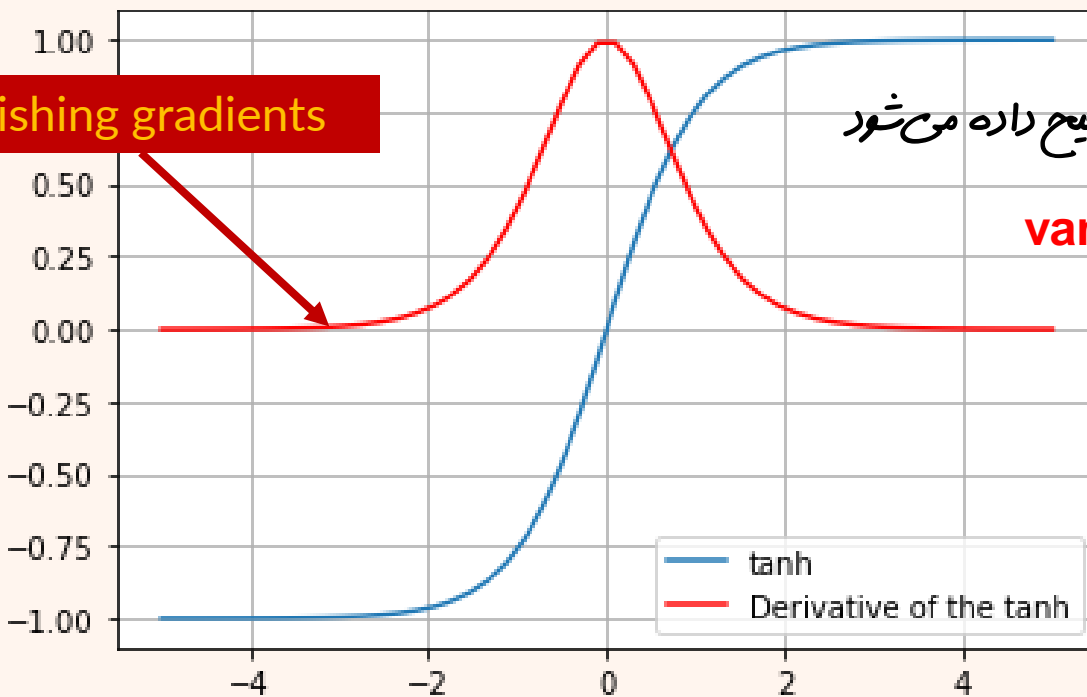
بدین ترتیب گرادیان ممی بدون نیاز به محاسبه صریح مشتق تابع فعال‌سازی قابل محاسبه می‌باشد.



توابع فعال‌سازی (ادامه...) *hyperbolic tangent*

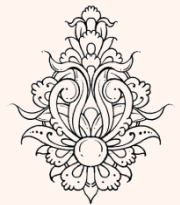
$$\varphi_j(v_j(n)) = a \tanh(bv_j(n)) \quad (a, b) > 0$$

$$\varphi'_j(v_j(n)) = \frac{b}{a} [a - y_j(n)][a + y_j(n)]$$



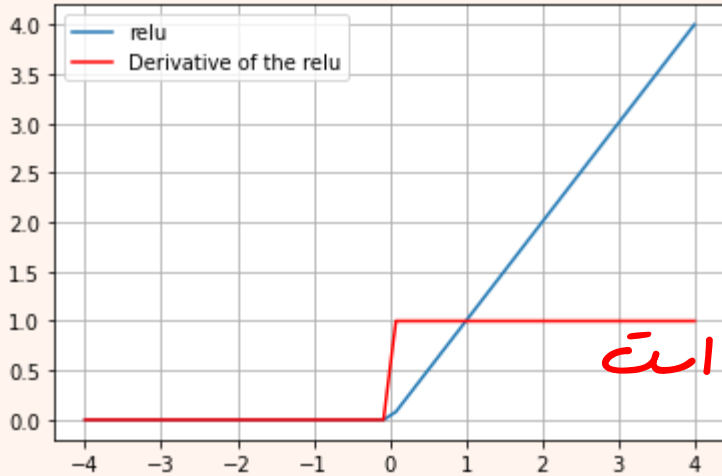
خروجی میانگین صفر دارد
گرادیان قوی‌تری دارد
در کایه منفی به sigmoid ترجیح داده می‌شود

احتمال vanishing gradient



ReLU(rectified linear activation function)

Biological Cybernetics1975, ICML2010



$$relu(x) = \begin{cases} x & x \geq 0 \\ 0 & x < 0 \end{cases}$$

- محاسبات ساده
- بازنمایی تنگ
- عدم اشباع

تنها برای لایه‌های مخفی مناسب است

Exploding gradient

مشکل نرون مرده

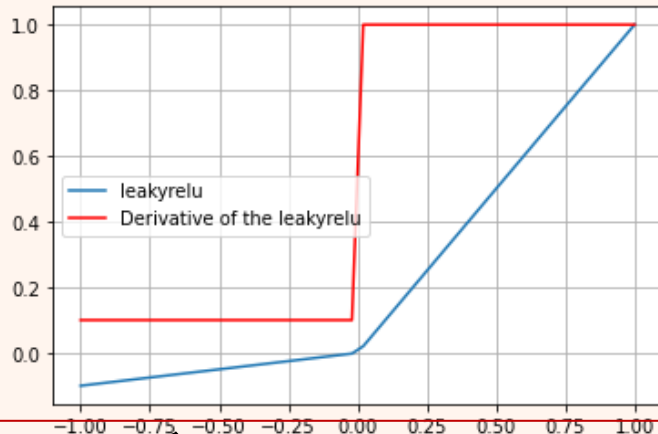
dead neurons

Dying relu

چنانچه مقدار ورودی نرون منفی باشد، وزن‌های نرون آموزش نمی‌بینند. این شرایط می‌تواند منجر به ایجاد **نرون مرده** شود، نرونی که هیچ‌گاه فعال نمی‌شود.

Leaky ReLU
(ICML2013)

$$lrelu(x) = \begin{cases} x & x > 0 \\ \alpha x & x \leq 0 \end{cases}$$

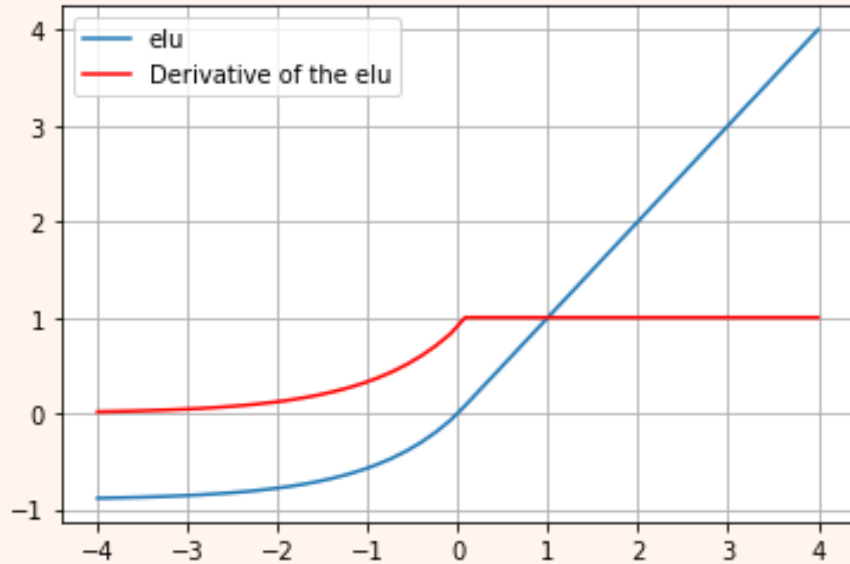


Parameterised ReLU
(ICCV2015)



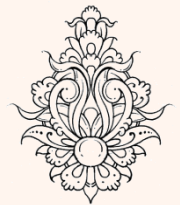
آموزش شبکه‌های عصبی در این شیوه پارامتر a نیز به پارامترهای قابل آموزش اضافه می‌شود.

Exponential Linear Unit(ELU) (ICLR2016)



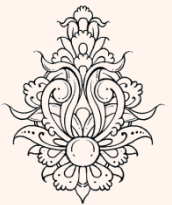
تخیرات متنوع نرم است •

Exploding gradient •



روش‌های بهبود کارایی

- غالباً گفته می‌شود که طراحی شبکه‌ی عصبی بیش از آن که «علم» باشد، نوعی «هنر» است، چرا که تنظیم پارامترهای زیادی که در طراحی دخیل هستند، تا حد زیادی به تجربه‌ی شخص بستگی دارد.
- با این وجود روش‌هایی برای بهبود کارایی شبکه وجود دارد:
- روش ترتیبی در مقابل دسته‌ای:
 - جاهایی که مجموعه‌ی آموزشی بزرگی در اختیار داریم و افزودگی در داده‌ها بالاست روش ترتیبی مناسب‌تر است.
 - مواردی که مجموعه‌ی آموزشی ثابت نیست، داده‌های جدید به مجموعه اضافه شود.



روش‌های سرعت بخشیدن به همگرایی

- برای تصحیح وزن w_{ji} ابتدا خطا را محاسبه می‌کردیم:

$$E(k) = \frac{1}{2} \sum_{j \in C} e_j^2(k) \quad e_j(k) = d_j(k) - y_j(k)$$

$$y_j = \varphi(v_j)$$

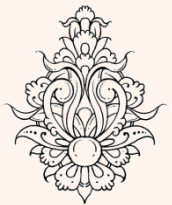
$$\Delta w_{ji}(k) = -\frac{\partial E(k)}{\partial w_{ji}(k)} = -\frac{1}{2} \frac{\partial e_j^2(k)}{\partial w_{ji}(k)}$$

$$= -e_j(k) \cdot \frac{\partial e_j(k)}{\partial w_{ji}(k)}$$

$$= e_j(k) \cdot \frac{\partial y_j(k)}{\partial w_{ji}(k)}$$

واضحی

مطلوب



روش‌های سرعت بخشیدن به همگرایی

$$\Delta w_{ji}(k) = e_j(k) \cdot \frac{\partial y_j(k)}{\partial v_j(k)} \cdot \frac{\partial v_j(k)}{\partial w_{ji}(k)}$$

اگر f تابع sigmoid باشد

$$y_j = \varphi(v_j)$$

ورودی به شافعی زا

$$\Delta w_{ji}(k) = e_j(k) y_j(k) (1 - y_j(k)) \cdot \frac{\partial v_j(k)}{\partial w_{ji}(k)}$$

مناسب

$$e_j \ll$$

$$y_j \approx 0$$

بزرگ ولی

$$e_j$$

$$y_j \approx 1$$

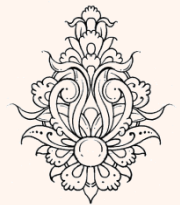
بزرگ ولی

$$e_j$$

$$\Delta w_{ji} \ll$$

نامناسب

کند شدن همگرایی



• تعویض تابع معیار خطا

– در روش عادی B.P تابع معیار خطا

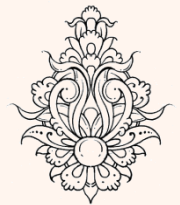
$$E(k) = \sum_{h=1}^M e_h^2(k)$$

– روش (V-O) *Van Oyen, Nienhuis*

$$E(k) = - \sum_{h=1}^M [d_h \ln(y_h(k)) + (1 - d_h) \ln(1 - y_h(k))]$$

فروبی مطلوب

فروبی واقعی



$$E(k) = - \sum_{h=1}^M [d_h \ln(y_h(k)) + (1 - d_h) \ln(1 - y_h(k))]$$

$$\Delta w_{ji}(k) = - \frac{\partial E(k)}{\partial w_{ji}(k)} = - \frac{\partial E(k)}{\partial y_j(k)} \cdot \frac{\partial y_j(k)}{\partial v_j(k)} \cdot \frac{\partial v_j(k)}{\partial w_{ji}(k)}$$

$$= \left[d_j \times \frac{1}{y_j(k)} - (1 - d_j) \times \frac{1}{(1 - y_j(k))} \right] \cdot y_j(k) (1 - y_j(k)) \cdot y_i$$

$$= e_j \cdot \frac{\partial v_j(k)}{\partial w_{ji}(k)}$$

e_j

فروجه لایه ما قبل آخر



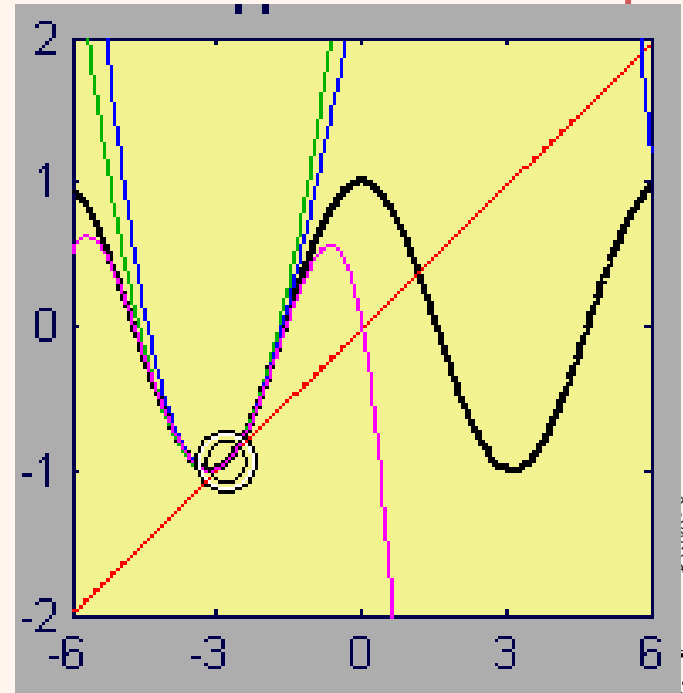
روش‌های مرتبه دوم



بسط تیلور

- توابع تحلیلی قابل تقریب زدن با چندجمله‌ای‌ها هستند.

$$F(x) = F(x^*) + \frac{d}{dx}F(x) \Big|_{x=x^*} (x - x^*) + \frac{1}{2} \frac{d^2}{dx^2} F(x) \Big|_{x=x^*} (x - x^*)^2 + \frac{1}{n!} \frac{d^n}{dx^n} F(x) \Big|_{x=x^*} (x - x^*)^n + \dots$$



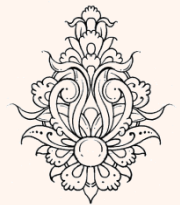
nnd8ts1



حالت برداری

$$F(\mathbf{X}) = F(x_1, x_2, \dots, x_n)$$

$$\begin{aligned} F(\mathbf{X}) = & F(\mathbf{X}^*) + \frac{\partial}{\partial x_1} F(\mathbf{X}) \Big|_{\mathbf{X}=\mathbf{X}^*} (x_1 - x_1^*) + \frac{\partial}{\partial x_2} F(\mathbf{X}) \Big|_{\mathbf{X}=\mathbf{X}^*} (x_2 - x_2^*) \\ & + \dots + \frac{\partial}{\partial x_n} F(\mathbf{X}) \Big|_{\mathbf{X}=\mathbf{X}^*} (x_n - x_n^*) + \frac{1}{2} \frac{\partial^2}{\partial x_1^2} F(\mathbf{X}) \Big|_{\mathbf{X}=\mathbf{X}^*} (x_1 - x_1^*)^2 \\ & + \frac{1}{2} \frac{\partial^2}{\partial x_1 \partial x_2} F(\mathbf{X}) \Big|_{\mathbf{X}=\mathbf{X}^*} (x_1 - x_1^*) (x_2 - x_2^*) + \dots \end{aligned}$$



فرم ماتریسی

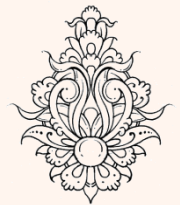
$$F(\mathbf{x}) = F(\mathbf{x}^*) + \nabla F(\mathbf{x})^T \Big|_{\mathbf{x}=\mathbf{x}^*} (\mathbf{x} - \mathbf{x}^*) + \frac{1}{2} (\mathbf{x} - \mathbf{x}^*)^T \nabla^2 F(\mathbf{x}) \Big|_{\mathbf{x}=\mathbf{x}^*} (\mathbf{x} - \mathbf{x}^*) + \dots$$

Gradient

$$\nabla F(\mathbf{x}) = \begin{bmatrix} \frac{\partial}{\partial x_1} F(\mathbf{x}) \\ \frac{\partial}{\partial x_2} F(\mathbf{x}) \\ \vdots \\ \frac{\partial}{\partial x_n} F(\mathbf{x}) \end{bmatrix}$$

Hessian

$$\nabla^2 F(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2}{\partial x_1^2} F(\mathbf{x}) & \frac{\partial^2}{\partial x_1 \partial x_2} F(\mathbf{x}) & \dots & \frac{\partial^2}{\partial x_1 \partial x_n} F(\mathbf{x}) \\ \frac{\partial^2}{\partial x_2 \partial x_1} F(\mathbf{x}) & \frac{\partial^2}{\partial x_2^2} F(\mathbf{x}) & \dots & \frac{\partial^2}{\partial x_2 \partial x_n} F(\mathbf{x}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2}{\partial x_n \partial x_1} F(\mathbf{x}) & \frac{\partial^2}{\partial x_n \partial x_2} F(\mathbf{x}) & \dots & \frac{\partial^2}{\partial x_n^2} F(\mathbf{x}) \end{bmatrix}$$



$$F(\mathbf{x}) = \frac{1}{2} \mathbf{x}^T \mathbf{A} \mathbf{x} + \mathbf{d}^T \mathbf{x} + c$$

فرم کلی یک تابع درجه ۲

الگوریتم‌های بهینه‌سازی

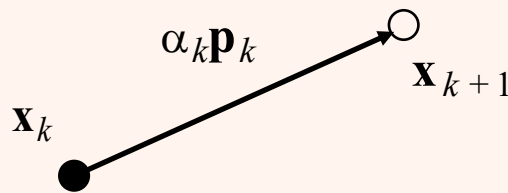
برای یافتن نقطه‌ی مینیمم خطا بر روی رویه‌ی کارایی (performance(error) surface) شیوه‌های متفاوتی وجود دارد.

هدف یافتن نقطه‌ی مینیمم تابع خطا ($F(X)$) می‌باشد با استفاده از الگوریتم‌های تکرار شونده است. از یک حدس اولیه شروع خواهیم کرد.

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$$

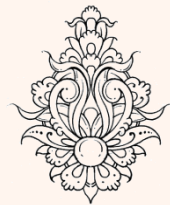
۱

$$\Delta \mathbf{x}_k = (\mathbf{x}_{k+1} - \mathbf{x}_k) = \alpha_k \mathbf{p}_k$$



\mathbf{p}_k - Search Direction

α_k - Learning Rate



- روند به‌روزرسانی می‌باید به گونه‌ای باشد که

$$F(\mathbf{x}_{k+1}) < F(\mathbf{x}_k)$$

داریم:

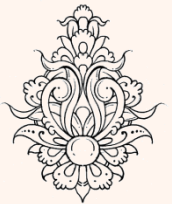
$$F(\mathbf{x}_{k+1}) = F(\mathbf{x}_k + \Delta\mathbf{x}_k) \approx F(\mathbf{x}_k) + \mathbf{g}_k^T \Delta\mathbf{x}_k$$

سری تیلور مرتبه یک

- که در آن \mathbf{g}_k از رابطه‌ی زیر به دست می‌آید:

$$\mathbf{g}_k \equiv \nabla F(\mathbf{x}) \Big|_{\mathbf{x} = \mathbf{x}_k}$$

گرادیان



- برای این $F(\mathbf{x}_{k+1}) < F(\mathbf{x}_k)$ بخش آخر رابطه‌ی زیر باید کوچک‌تر از صفر باشد.

$$F(\mathbf{x}_{k+1}) = F(\mathbf{x}_k + \Delta\mathbf{x}_k) \approx F(\mathbf{x}_k) + \mathbf{g}_k^T \Delta\mathbf{x}_k$$

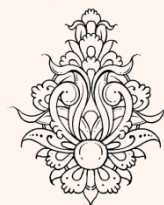
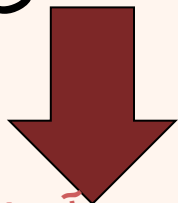
- پس داریم:

$$\mathbf{g}_k^T \Delta\mathbf{x}_k = \alpha_k \mathbf{g}_k^T \mathbf{p}_k < 0$$

- اگر α بین صفر و یک باشد، خواهیم داشت:

$$\mathbf{g}_k^T \mathbf{p}_k < 0$$

- به هر بردار \mathbf{p}_k که شرط بالا صدق کند، «descent direction» می‌گویند.



الگوریتم‌های بهینه‌سازی

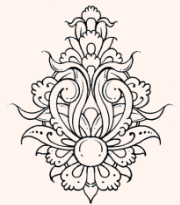
$$g_k^T p_k < 0$$

- در رابطه‌ی فوق می‌باید ضرب داخلی دو بردار گرادیان و بردار جهت نزولی (descent direction) منفی باشد، هر چه عبارت فوق منفی‌تر باشد، سریع‌تر به نقطه‌ی مزبور نزدیک می‌شویم.
- چگونه می‌توان به سریع‌ترین کاهش دست یافت؟

$$p_k = -g_k$$

- برای متد steepest decent داریم:

$$x_{k+1} = x_k - \alpha_k g_k$$



• Steepest decent را برای مساله‌ی زیر اعمال

$$F(x) = x_1^2 + 25x_2^2$$

کنید:

• با در نظر گرفتن $x_0 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$ خواهیم داشت:

$$\nabla F(x) = \begin{bmatrix} \frac{\partial}{\partial x_1} F(x) \\ \frac{\partial}{\partial x_2} F(x) \end{bmatrix} = \begin{bmatrix} 2x_1 \\ 50x_2 \end{bmatrix} \rightarrow \nabla F(x) \Big|_{x=x_0} = \begin{bmatrix} 1 \\ 25 \end{bmatrix}$$



• با در نظر گرفتن $\alpha=0.01$ خواهیم داشت:

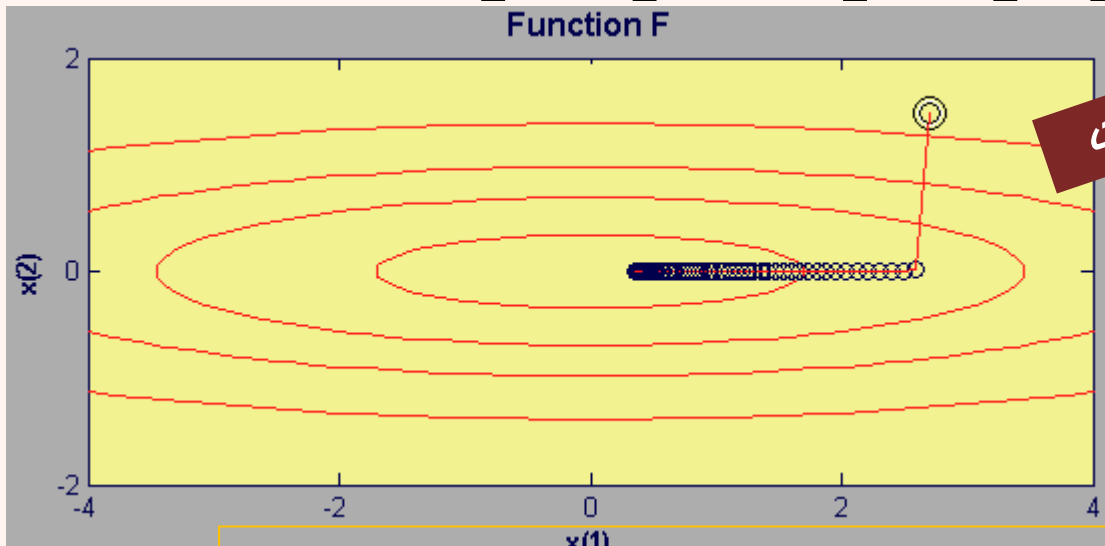
First iteration

$$x_1 = x_0 - \alpha g_0 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} - 0.01 \begin{bmatrix} 1 \\ 25 \end{bmatrix} = \begin{bmatrix} 0.49 \\ 0.25 \end{bmatrix}$$

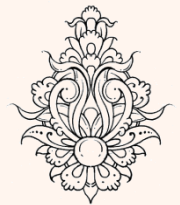
Second iteration

$$x_2 = x_1 - \alpha g_1 = \begin{bmatrix} 0.49 \\ 0.25 \end{bmatrix} - 0.01 \begin{bmatrix} 0.98 \\ 12.5 \end{bmatrix} = \begin{bmatrix} 0.4802 \\ 0.125 \end{bmatrix}$$

در صورت ادامه‌ی روند خواهیم داشت

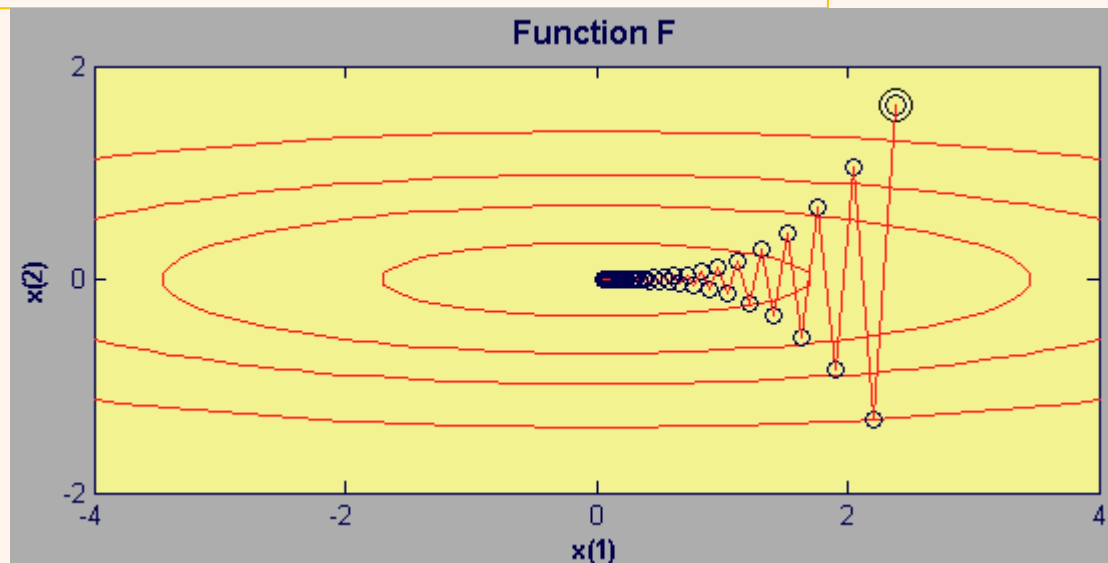


Trajectory for Steepest Descent with $\alpha = 0.01$

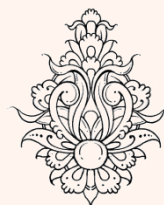


- اگر نرخ آموزش را بالا ببریم چیزی شبیه به شکل زیر خواهیم داشت:
- در این صورت نوسان بیشتری خواهیم داشت و ناپایداری بیشتری خواهد شد.

Trajectory for Steepest Descent with $\alpha = 0.035$



همواره جهت تخیرات بر مسیر عمود است
و این به دلیل استفاده از گرادیان است.



مثال

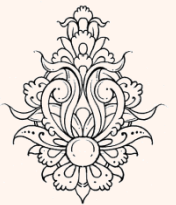
$$F(\mathbf{x}) = x_1^2 + 2x_1x_2 + 2x_2^2 + x_1$$

$$\mathbf{x}_0 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \quad \alpha = 0.1$$

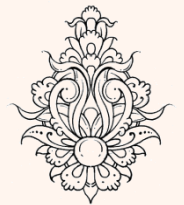
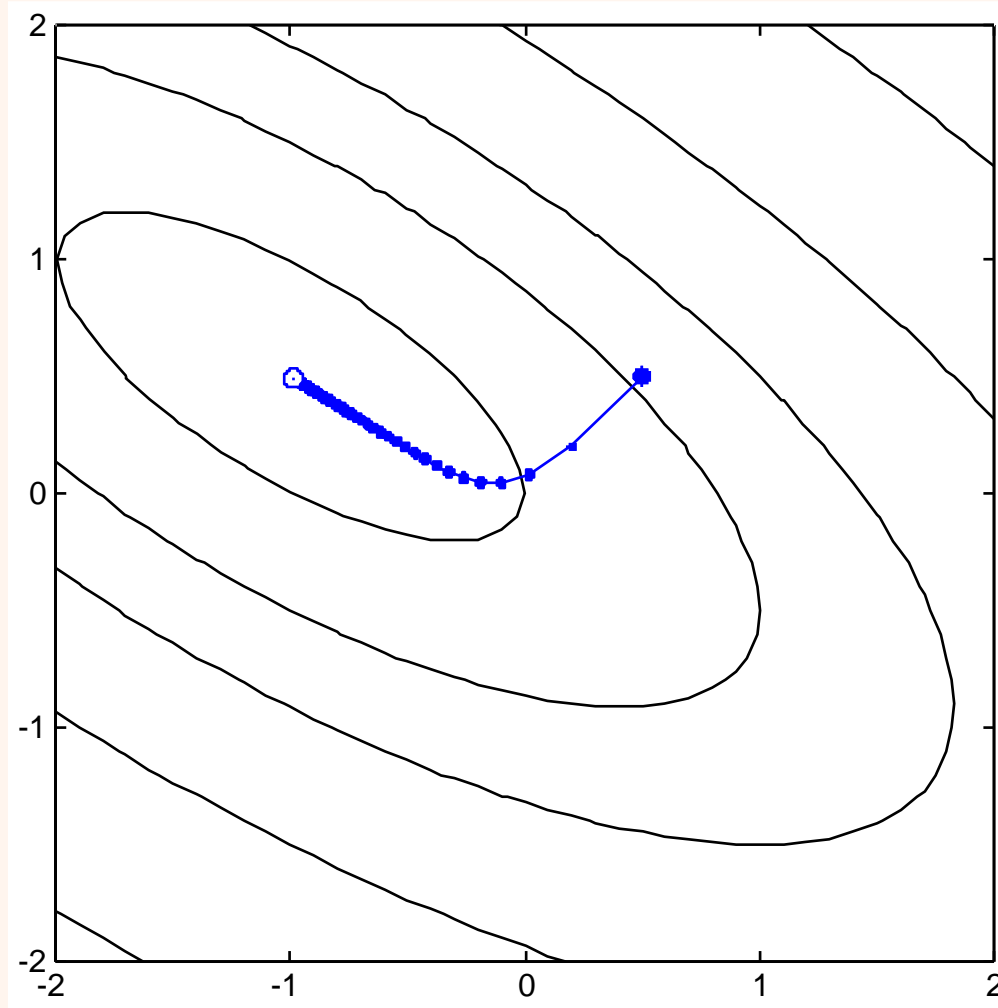
$$\nabla F(\mathbf{x}) = \begin{bmatrix} \frac{\partial}{\partial x_1} F(\mathbf{x}) \\ \frac{\partial}{\partial x_2} F(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} 2x_1 + 2x_2 + 1 \\ 2x_1 + 4x_2 \end{bmatrix} \quad \mathbf{g}_0 = \nabla F(\mathbf{x})|_{\mathbf{x} = \mathbf{x}_0} = \begin{bmatrix} 3 \\ 3 \end{bmatrix}$$

$$\mathbf{x}_1 = \mathbf{x}_0 - \alpha \mathbf{g}_0 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} - 0.1 \begin{bmatrix} 3 \\ 3 \end{bmatrix} = \begin{bmatrix} 0.2 \\ 0.2 \end{bmatrix}$$

$$\mathbf{x}_2 = \mathbf{x}_1 - \alpha \mathbf{g}_1 = \begin{bmatrix} 0.2 \\ 0.2 \end{bmatrix} - 0.1 \begin{bmatrix} 1.8 \\ 1.2 \end{bmatrix} = \begin{bmatrix} 0.02 \\ 0.08 \end{bmatrix}$$



نمودار



$$F(\mathbf{x}) = \frac{1}{2} \mathbf{x}^T \mathbf{A} \mathbf{x} + \mathbf{d}^T \mathbf{x} + c$$

$$\nabla F(\mathbf{x}) = \mathbf{A} \mathbf{x} + \mathbf{d}$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha \mathbf{g}_k = \mathbf{x}_k - \alpha (\mathbf{A} \mathbf{x}_k + \mathbf{d}) \longrightarrow \mathbf{x}_{k+1} = \underbrace{[\mathbf{I} - \alpha \mathbf{A}]}_{\text{linear dynamic system}} \mathbf{x}_k - \alpha \mathbf{d}$$

پایداری وابسته به مقادیر ویژه‌ی این ماتریس است

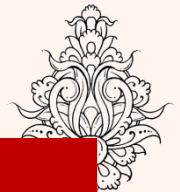
$(\lambda_i$ - eigenvalue of $\mathbf{A})$

$$[\mathbf{I} - \alpha \mathbf{A}] \mathbf{z}_i = \mathbf{z}_i - \alpha \mathbf{A} \mathbf{z}_i = \mathbf{z}_i - \alpha \lambda_i \mathbf{z}_i = \underbrace{(1 - \alpha \lambda_i)}_{\text{Eigenvalues of } [\mathbf{I} - \alpha \mathbf{A}]} \mathbf{z}_i$$

Eigenvalues of $[\mathbf{I} - \alpha \mathbf{A}]$.

با فرض آن که دارای که مینم مطلوب با

$$|(1 - \alpha \lambda_i)| < 1 \longrightarrow \alpha < \frac{2}{\lambda_i} \longrightarrow \alpha < \frac{2}{\lambda_{max}}$$



مثال

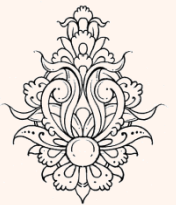
- با اعمال این مساله بر مثال قبلی برآنیه
بیشترین میزان نرخ آموزش مجاز را محاسبه کنید

$$F(x) = x_1^2 + 25x_2^2$$

- همان‌گونه که مشاهده می‌شود مثالی درجه دو
است پس Hessian Matrix به صورت زیر خواهد
بود:

$$A = \begin{bmatrix} 2 & 0 \\ 0 & 50 \end{bmatrix}$$

- پس برای مقادیر ویژه داریم:
 $\{(\lambda_1 = 2), \left(z_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}\right)\}, \{(\lambda_2 = 50), \left(z_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}\right)\}$



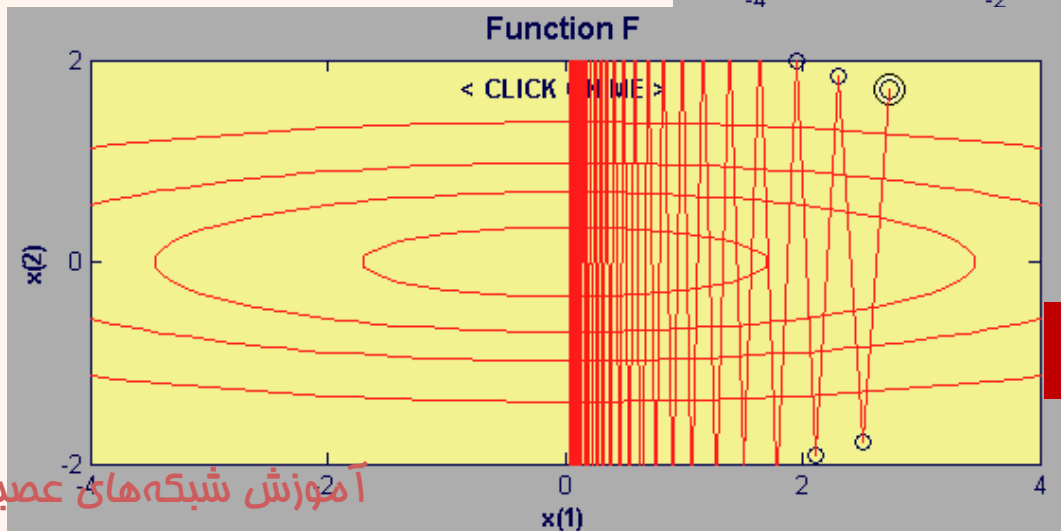
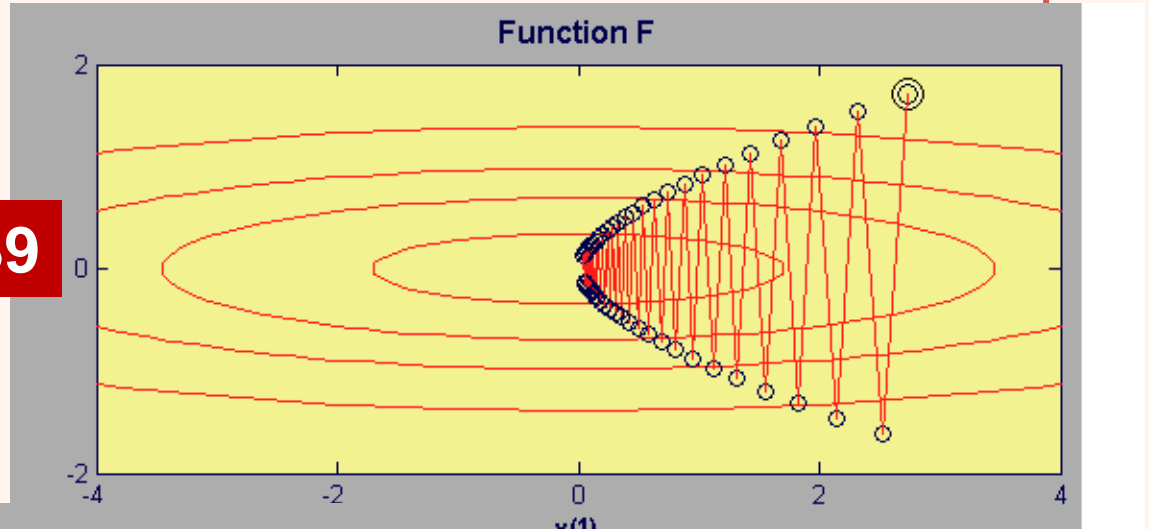
مثال-ادامه

• پس بیشترین میزان نرخ آموزش از رابطه‌ی زیر به

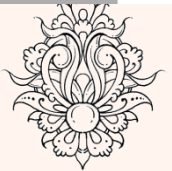
$$\alpha < \frac{2}{\lambda_{\max}} = \frac{2}{50} = 0.04$$

دست می‌آید:

$\alpha=0.039$



$\alpha=0.041$



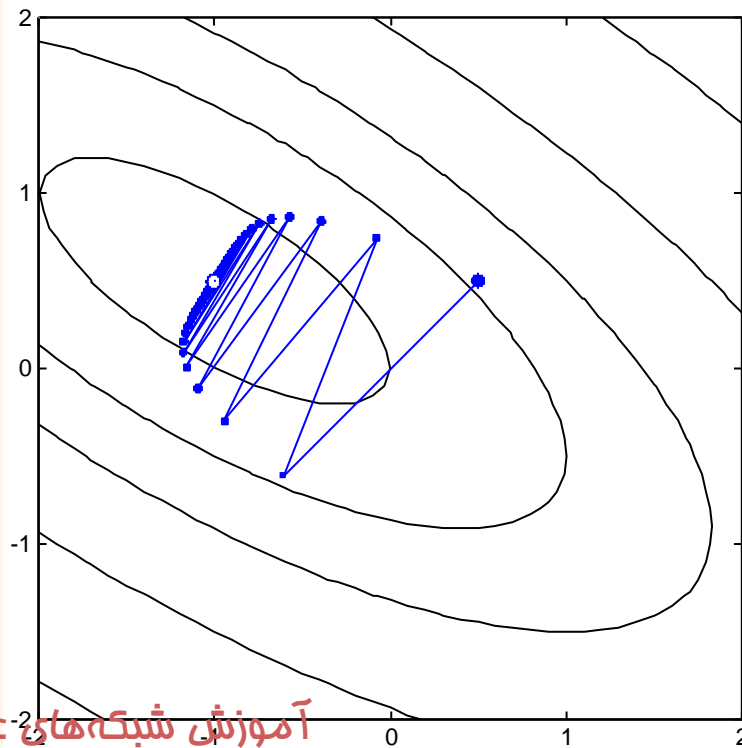
مثال

$$F(\mathbf{x}) = x_1^2 + 2x_1x_2 + 2x_2^2 + x_1$$

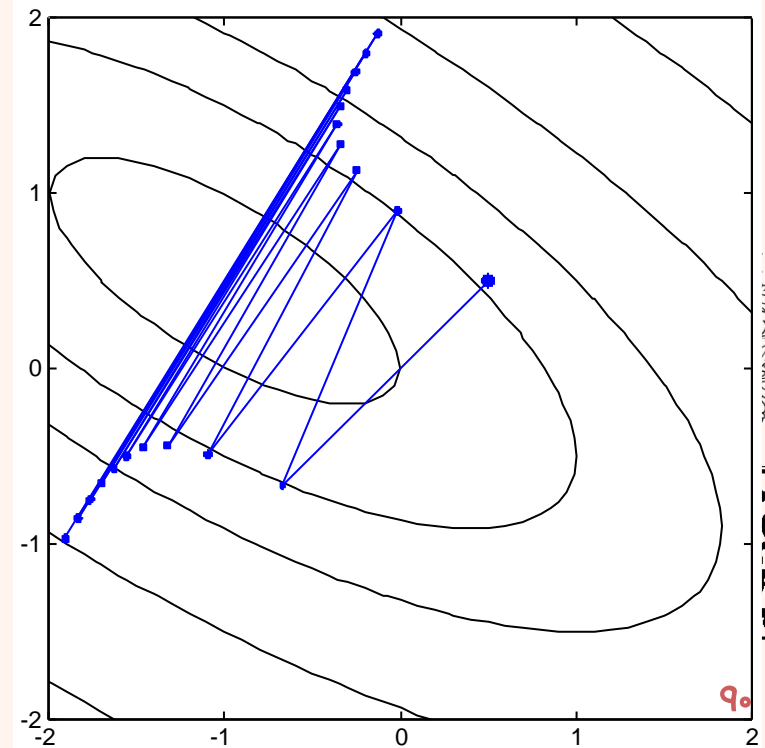
$$\mathbf{A} = \begin{bmatrix} 2 & 2 \\ 2 & 4 \end{bmatrix} \quad \left\{ (\lambda_1 = 0.764), \left(\mathbf{z}_1 = \begin{bmatrix} 0.851 \\ -0.526 \end{bmatrix} \right) \right\}, \left\{ \lambda_2 = 5.24, \left(\mathbf{z}_2 = \begin{bmatrix} 0.526 \\ 0.851 \end{bmatrix} \right) \right\}$$

$$\alpha < \frac{2}{\lambda_{max}} = \frac{2}{5.24} = 0.38$$

$\alpha = 0.37$



$\alpha = 0.39$

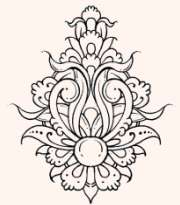


- در مورد انتخاب نرخ آموزش به صورت افقی پیش از این صحبت شد.
- راه دیگر انتخاب نرخ آموزش به گونه‌ای است که عبارت زیر مینیمم شود:

$$F(\mathbf{x}_k + \alpha_k \mathbf{p}_k)$$

- برای این منظور لازم است در راستای \mathbf{p}_k جستجویی صورت پذیرد.
- برای توابع درجه‌ی دوم می‌توان راه حلی تحلیلی ارائه نمود:

$$\frac{d}{d\alpha_k}(F(\mathbf{x}_k + \alpha_k \mathbf{p}_k)) = \nabla F(\mathbf{x})^T \Big|_{\mathbf{x} = \mathbf{x}_k} \mathbf{p}_k + \alpha_k \mathbf{p}_k^T \nabla^2 F(\mathbf{x}) \Big|_{\mathbf{x} = \mathbf{x}_k} \mathbf{p}_k$$



تنظیم نرخ یادگیری (ادامه...)

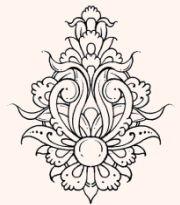
$$\frac{d}{d\alpha_k}(F(\mathbf{x}_k + \alpha_k \mathbf{p}_k)) = \nabla F(\mathbf{x})^T \Big|_{\mathbf{x} = \mathbf{x}_k} \mathbf{p}_k + \alpha_k \mathbf{p}_k^T \nabla^2 F(\mathbf{x}) \Big|_{\mathbf{x} = \mathbf{x}_k} \mathbf{p}_k$$

در نتیجه

$$\alpha_k = - \frac{\nabla F(\mathbf{x})^T \Big|_{\mathbf{x} = \mathbf{x}_k} \mathbf{p}_k}{\mathbf{p}_k^T \nabla^2 F(\mathbf{x}) \Big|_{\mathbf{x} = \mathbf{x}_k} \mathbf{p}_k} = - \frac{\mathbf{g}_k^T \mathbf{p}_k}{\mathbf{p}_k^T \mathbf{A}_k \mathbf{p}_k}$$

که در آن

$$\mathbf{A}_k \equiv \nabla^2 F(\mathbf{x}) \Big|_{\mathbf{x} = \mathbf{x}_k}$$



مثال

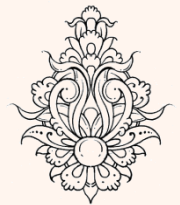
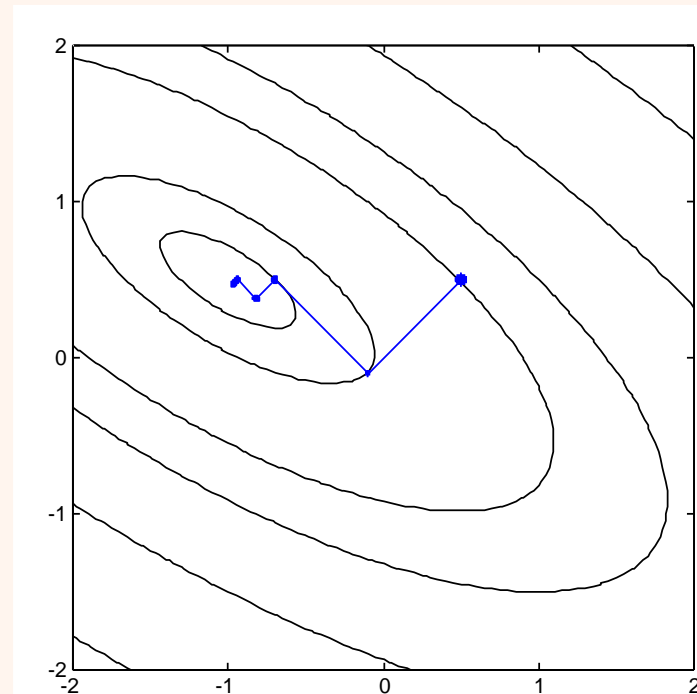
$$F(\mathbf{x}) = \frac{1}{2} \mathbf{x}^T \begin{bmatrix} 2 & 2 \\ 2 & 4 \end{bmatrix} \mathbf{x} + [1 \ 0] \mathbf{x} \quad \mathbf{x}_0 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$$

$$\nabla F(\mathbf{x}) = \begin{bmatrix} \frac{\partial}{\partial x_1} F(\mathbf{x}) \\ \frac{\partial}{\partial x_2} F(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} 2x_1 + 2x_2 + 1 \\ 2x_1 + 4x_2 \end{bmatrix}$$

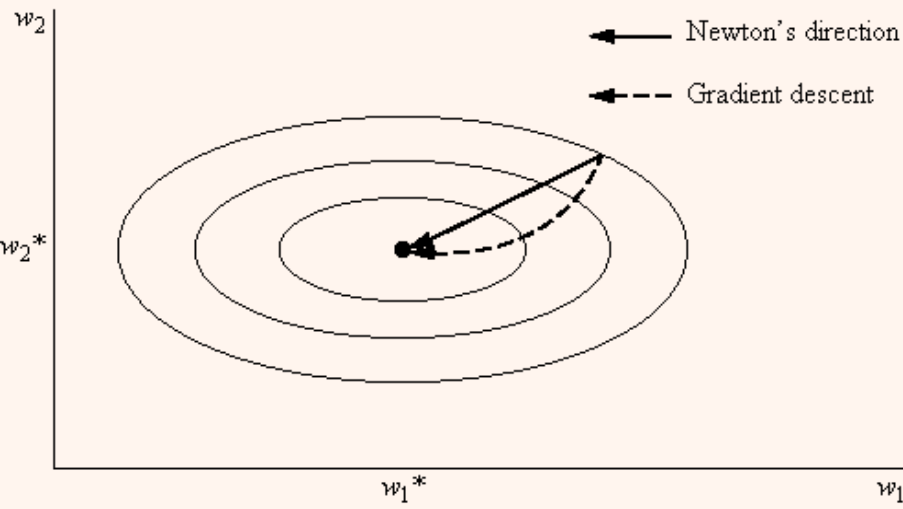
$$\mathbf{p}_0 = -\mathbf{g}_0 = -\nabla F(\mathbf{x}) \Big|_{\mathbf{x}=\mathbf{x}_0} = \begin{bmatrix} -3 \\ -3 \end{bmatrix}$$

$$\alpha_0 = -\frac{\begin{bmatrix} 3 & 3 \end{bmatrix} \begin{bmatrix} -3 \\ -3 \end{bmatrix}}{\begin{bmatrix} -3 & -3 \end{bmatrix} \begin{bmatrix} 2 & 2 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} -3 \\ -3 \end{bmatrix}} = 0.2$$

$$\mathbf{x}_1 = \mathbf{x}_0 - \alpha_0 \mathbf{g}_0 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} - 0.2 \begin{bmatrix} 3 \\ 3 \end{bmatrix} = \begin{bmatrix} -0.1 \\ -0.1 \end{bmatrix}$$



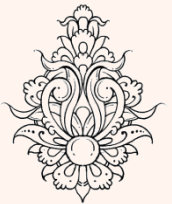
روش نیوتن



در ریاضیات از روش نیوتن جهت یافتن ریشه عبارت ریاضی به وسیله الگوریتمی تکراری استفاده می‌شود.

در مسأله‌ی بهینه‌سازی از این الگوریتم برای یافتن نقاط مانا (Stationary Point) به گونه‌ای که مشتق را صفر کند، استفاده می‌شود.

در این حالت با شروع از نقطه‌ی x_0 به دنبال نقطه‌ی x^* هستیم به گونه‌ای که $f'(x^*)=0$



روش نیوتن

- در بسط سری تیلور با استفاده از روابط زیر خواهیم داشت:

$$\Delta x = x_{k+1} - x_k$$

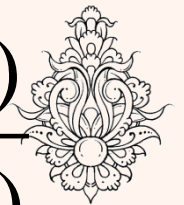
$$f_T(x_k + \Delta x) \approx f(x_k) + f'(x_k)\Delta x + \frac{1}{2}f''(x_k)\Delta x^2$$

- با مشتق گرفتن از رابطه‌ی فوق و قرار دادن آن برابر با صفر داریم:

$$f'(x_k) + f''(x_k)\Delta x = 0$$

$$\Delta x = -\frac{f'(x_k)}{f''(x_k)}$$

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)} \quad k = 0, 1, \dots$$



روش نیوتن

$$f_T(x_k + \Delta x) = f(x_k) + f'(x_k)\Delta x + \frac{1}{2}f''(x_k)\Delta x^2$$

• در نمایش برداری داریم:

$$F(\mathbf{x}_{k+1}) = F(\mathbf{x}_k + \Delta \mathbf{x}_k) \approx F(\mathbf{x}_k) + \mathbf{g}_k^T \Delta \mathbf{x}_k + \frac{1}{2} \Delta \mathbf{x}_k^T \mathbf{A}_k \Delta \mathbf{x}_k$$

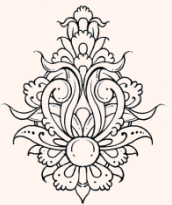
سری تیلور مرتبه ۲

با مشتق گرفتن از این تقریب و قرار دادن رابطه برابر با صفر، نقطه‌ی
مانا را بیابیم:

$$\mathbf{g}_k + \mathbf{A}_k \Delta \mathbf{x}_k = \mathbf{0}$$

$$\Delta \mathbf{x}_k = -\mathbf{A}_k^{-1} \mathbf{g}_k$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{A}_k^{-1} \mathbf{g}_k$$

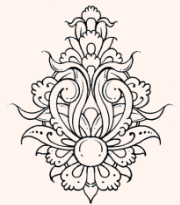
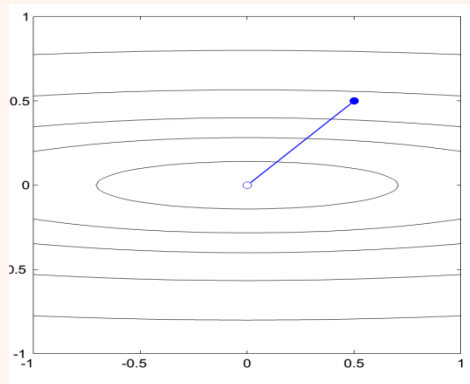


$$F(x) = x_1^2 + 25x_2^2$$

• با در نظر گرفتن $x_0 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$ خواهیم داشت:

$$\nabla F(x) = \begin{bmatrix} \frac{\partial}{\partial x_1} F(x) \\ \frac{\partial}{\partial x_2} F(x) \end{bmatrix} = \begin{bmatrix} 2x_1 \\ 50x_2 \end{bmatrix} \quad \nabla^2 F(x) = \begin{bmatrix} 2 & 0 \\ 0 & 50 \end{bmatrix}$$

$$x_1 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} - \begin{bmatrix} 2 & 0 \\ 0 & 50 \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ 25 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} - \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} = 0$$



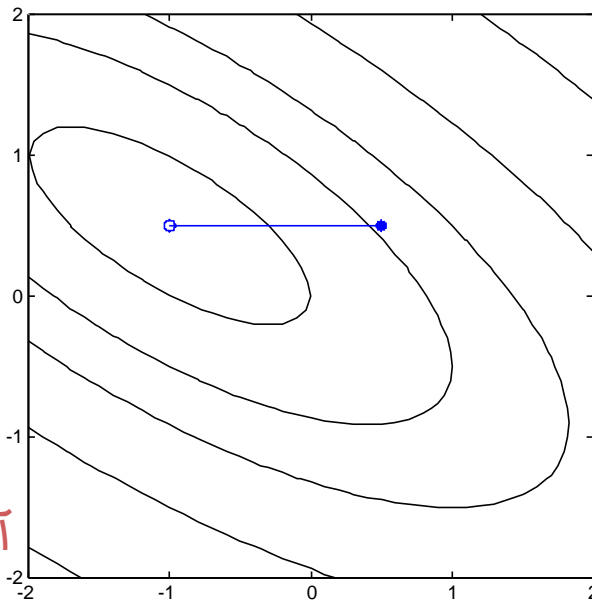
مثال ۲

$$F(\mathbf{x}) = x_1^2 + 2x_1x_2 + 2x_2^2 + x_1$$

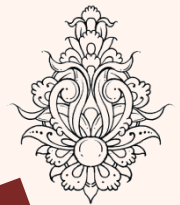
$$\mathbf{x}_0 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$$

$$\nabla F(\mathbf{x}) = \begin{bmatrix} \frac{\partial}{\partial x_1} F(\mathbf{x}) \\ \frac{\partial}{\partial x_2} F(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} 2x_1 + 2x_2 + 1 \\ 2x_1 + 4x_2 \end{bmatrix} \quad \mathbf{g}_0 = \nabla F(\mathbf{x})|_{\mathbf{x} = \mathbf{x}_0} = \begin{bmatrix} 3 \\ 3 \end{bmatrix} \quad \mathbf{A} = \begin{bmatrix} 2 & 2 \\ 2 & 4 \end{bmatrix}$$

$$\mathbf{x}_1 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} - \begin{bmatrix} 2 & 2 \\ 2 & 4 \end{bmatrix}^{-1} \begin{bmatrix} 3 \\ 3 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} - \begin{bmatrix} 1 & -0.5 \\ -0.5 & 0.5 \end{bmatrix} \begin{bmatrix} 3 \\ 3 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} - \begin{bmatrix} 1.5 \\ 0 \end{bmatrix} = \begin{bmatrix} -1 \\ 0.5 \end{bmatrix}$$

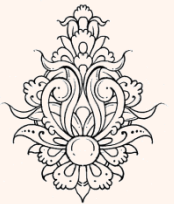


Quadratic Termination



مشکلات روش نیوتن

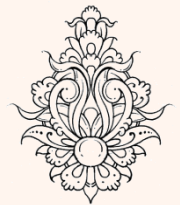
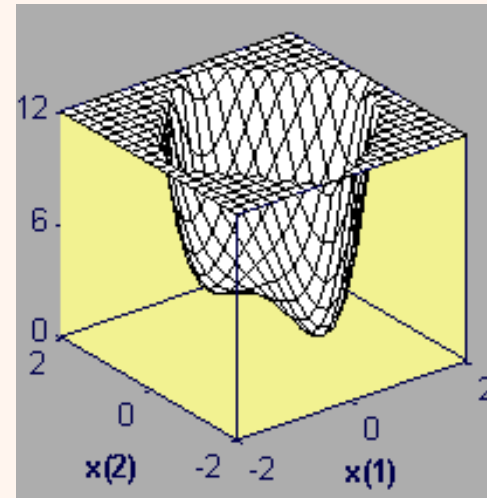
- این روش نیاز به محاسبه ماتریس **معکوس** Hessian دارد.
- ممکن است در شرایطی این ماتریس **معکوس پذیر** نباشد.
- هنگامی که تابع خطا (کارایی)، درجهی دو نباشد، تضمینی مبنی بر **همگرایی** وجود نخواهد داشت.



همگرایی

- در صورتی که تابع هزینه (کارایی) درجه‌ی دوم نباشد، با استفاده از روش نیوتن نمی‌توان همگرایی روش را تضمین کرد.
- در این صورت، همگرایی وابسته به تابع هزینه و حتی حدس اولیه است.

$$F(\mathbf{x}) = (x_2 - x_1)^4 + 8x_1x_2 - x_1 + x_2 + 3$$

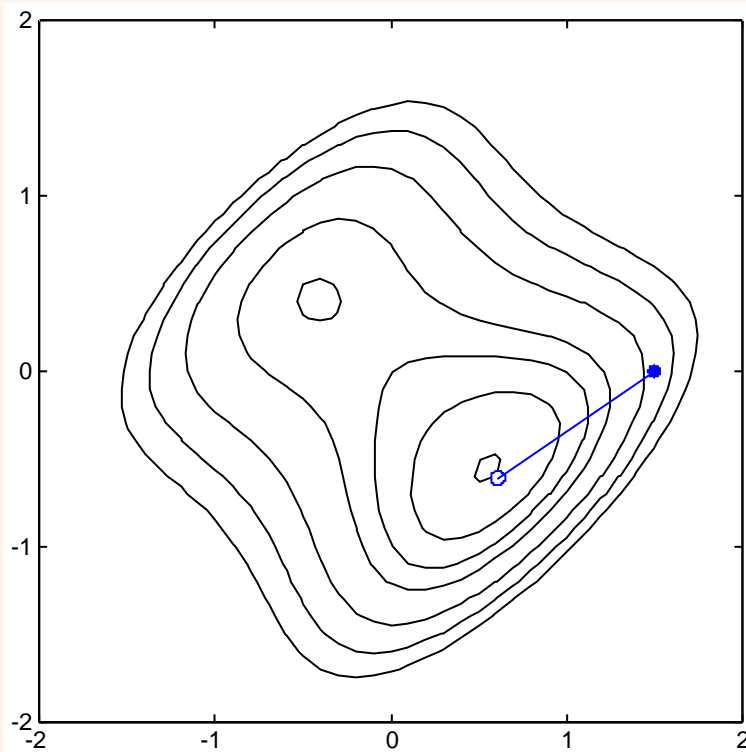


شرایط اولیه متفاوت

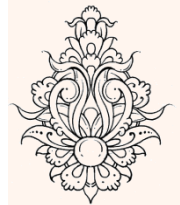
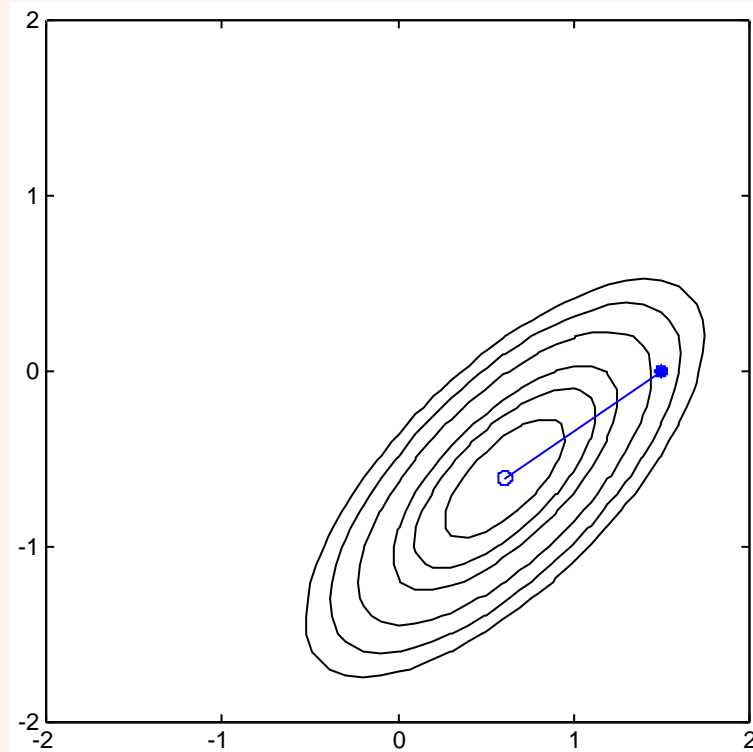
$$F(\mathbf{x}) = (x_2 - x_1)^4 + 8x_1x_2 - x_1 + x_2 + 3$$

Stationary Points: $\mathbf{x}^1 = \begin{bmatrix} -0.42 \\ 0.42 \end{bmatrix}$ $\mathbf{x}^2 = \begin{bmatrix} -0.13 \\ 0.13 \end{bmatrix}$ $\mathbf{x}^3 = \begin{bmatrix} 0.55 \\ -0.55 \end{bmatrix}$

$F(\mathbf{x})$

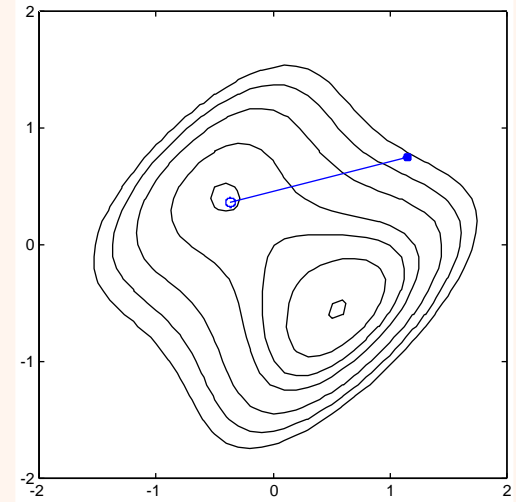
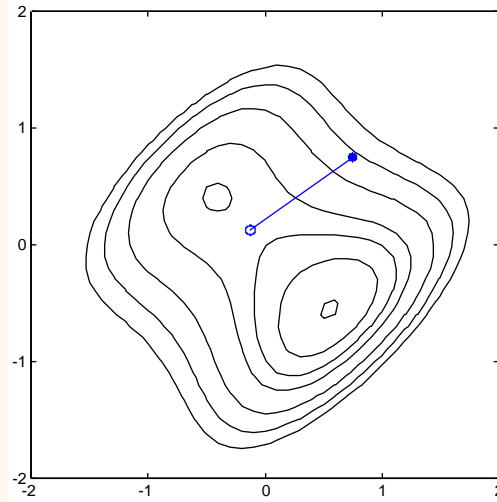
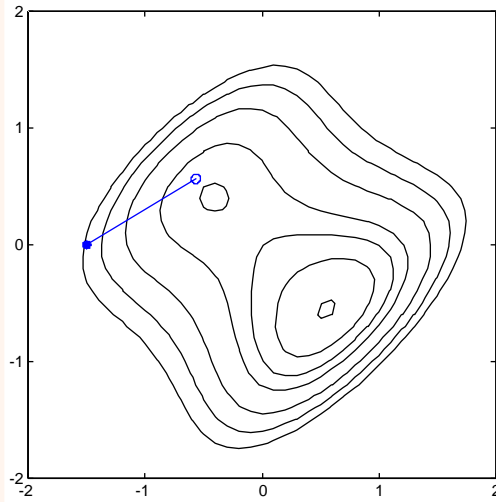


$F_2(\mathbf{x})$

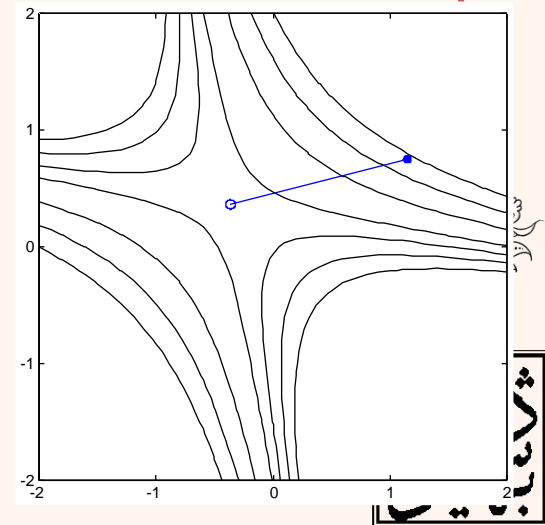
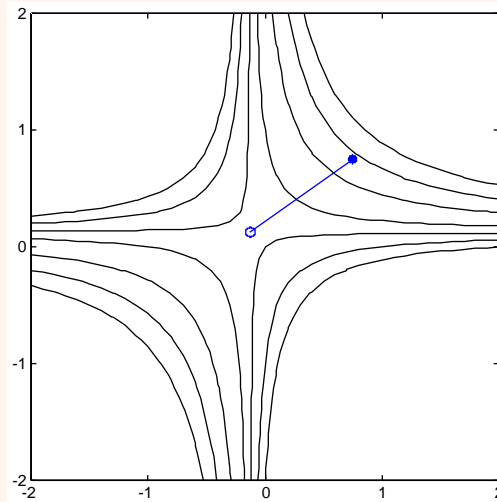
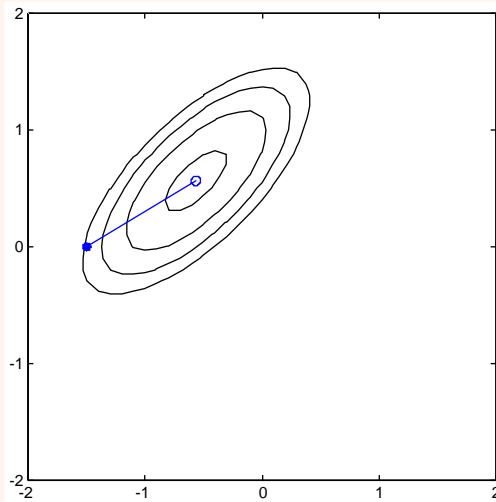


شرایط اولیه متفاوت

$F(\mathbf{x})$



$F_2(\mathbf{x})$



$X = \{w \text{ and } b \text{ of layer 1, } w \text{ and } b \text{ of layer 2, } \dots\}$

• در روش عادی B.P داشتیم:

$$X_{k+1} = X_k - \mu \nabla F(X_k)$$

• در روش نیوتن داریم:

$$X_{k+1} = X_k - A_k^{-1} \nabla F(X_k)$$

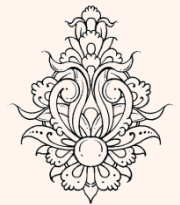
تابع معیار خطا

$$F(X) = \sum_{i=1}^M e_i^2(X)$$

• که در آن

$$g_k = \nabla F(X) \Big|_{X=X_k}$$

$$A_k = \nabla^2 F(X) \Big|_{X=X_k}$$



$X = \{w \text{ and } b \text{ of layer } 1, w \text{ and } b \text{ of layer } 2, \dots\}$

تابع معیار خطا

$$F(X) = \sum_{i=1}^M e_i^2(X) = E^T(X)E(X)$$

$$E(X) = [e_1 \ e_2 \ \dots \ e_M]^T$$

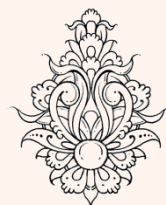
$$[\nabla F(X)]_{x_j} = \frac{\partial F(X)}{\partial x_j} = 2 \sum_{i=1}^M e_i(X) \cdot \frac{\partial e_i(X)}{\partial x_j}$$

$$\nabla F(X) = 2J^T(X)E(X)$$

ماتریس ژاکوبی

$J(X) =$

$$\begin{bmatrix} \frac{\partial e_1(X)}{\partial x_1} & \dots & \frac{\partial e_1(X)}{\partial x_p} \\ \vdots & & \vdots \\ \frac{\partial e_M(X)}{\partial x_1} & \dots & \frac{\partial e_M(X)}{\partial x_p} \end{bmatrix}$$



محاسبه مشتق دوم

$$\frac{\partial F(X)}{\partial x_j} = 2 \sum_{i=1}^M e_i(X) \cdot \frac{\partial e_i(X)}{\partial x_j}$$

$$\left[\nabla^2 F(X) \right]_{x_k, j} = \frac{\partial^2 F(X)}{\partial x_k \partial x_j} = \frac{\partial}{\partial x_k} \left[\frac{\partial F(X)}{\partial x_j} \right]$$

$$= 2 \sum_{i=1}^M \left[\frac{\partial e_i(X)}{\partial x_k} \cdot \frac{\partial e_i(X)}{\partial x_j} + e_i(X) \cdot \frac{\partial^2 e_i(X)}{\partial x_k \partial x_j} \right]$$

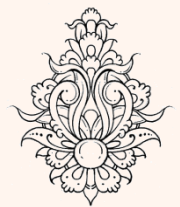
$$\nabla^2 F(X) = 2J^T(X)J(X) + 2S(X)$$

$$S(X) = \sum_{i=1}^M e_i(X) \cdot \frac{\partial^2 e_i(X)}{\partial x_k \partial x_j}$$

$S(X)$ معمولاً بسیار کوچک است به همین دلیل در محاسبات می‌توان از آن صرف‌نظر کرد

$$\nabla^2 F(X) = 2J^T(X)J(X)$$

در این صورت متد را **Gauss-Newton** گویند



• در روش نیوتن داشتیم

$$X_{k+1} = X_k - A_k^{-1} g_k$$

$$g_k = \nabla F(X) \Big|_{X=X_k} \longrightarrow \nabla F(X) = 2J^T(x)E(X)$$

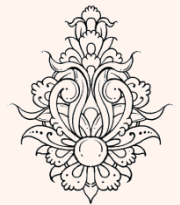
$$A_k = \nabla^2 F(X) \Big|_{X=X_k} \longrightarrow \nabla^2 F(X) = 2J^T(X)J(X)$$

برای مناسبی رابطه‌ی مذکور تنها از مشتق اول استفاده می‌شود

Gauss–Newton

$$X_{k+1} = X_k - \left[2J^T(X) \cdot J(X) \right]^{-1} \cdot 2J^T(x)E(X)$$

آیا این ماتریس همواره معکوس پذیر است؟



Levenberg–Marquardt

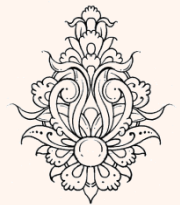
$$X_{k+1} = X_k - \underbrace{\left[2J^T(X_k) J(X_k) \right]^{-1}}_{H_k} \cdot 2J^T(x) E(X)$$

- در این روش به جای استفاده از H_k از ماتریس G_k استفاده می‌شود.

$$G_k = H_k + \mu_k I$$

بسیار کوچک

- اگر $\mu = 0$ باشد روش گاوس-نیوتن است.



Levenberg–Marquardt

مقدار ویژه

یادآوری

$$Aq_i = \lambda_i q_i$$

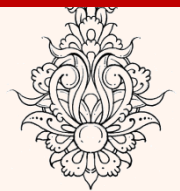
بردار ویژه

$$\begin{aligned} Gq_i &= [H + \mu I]q_i \\ &= Hq_i + \mu q_i \\ &= \lambda_i q_i + \mu q_i \\ &= (\lambda_i + \mu)q_i \end{aligned}$$

فرض می‌کنیم H دارای مقادیر ویژه λ_i و بردارهای ویژه q_i باشد

G دارای مقادیر ویژه $\lambda_i + \mu$ و بردارهای ویژه q_i خواهد بود.

$$Gq_i = (\lambda_i + \mu)q_i$$



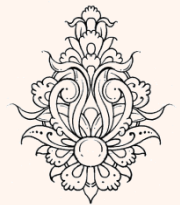
Levenberg–Marquardt

G دارای مقادیر ویژه $\mu + \lambda_i$ و بردارهای ویژه q_i است

$$Gq_i = (\lambda_i + \mu)q_i$$

- برای محکوس‌پذیری کافی است مقادیر ویژه ماتریس مثبت باشد.
- می‌توان آنقدر μ را تغییر داد تا مقادیر ویژه مثبت گردد.
- در روش گاوس نیوتن این مقدار ثابت و غیرقابل تغییر بود.

$$X_{k+1} = X_k - \left[2J^T(X_k) \cdot J(X_k) + \mu_k I \right]^{-1} \cdot 2J^T(X_k) E(X_k)$$



Levenberg–Marquardt

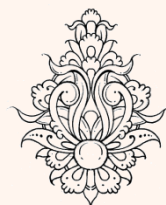
$$X_{k+1} = X_k - \left[2J^T(X_k) \cdot J(X_k) + \mu_k I \right]^{-1} \cdot 2J^T(X_k) E(X_k)$$

• می‌توان نشان داد، با افزایش میزان μ رابطه همانند زیر می‌شود:

$$X_{k+1} = X_k - \frac{1}{\mu_k} J^T(X_k) E(X_k) = X_k - \frac{1}{2\mu_k} \nabla F(X_k)$$

• معمولاً الگوریتم را با μ کوچک در حدود 0.01 شروع کرده در صورت کاهش خطا (موفقیت) با ضریب θ (در حدود ۱۰) کاهش می‌دهند.

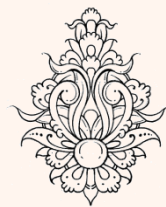
• در صورت عدم موفقیت μ را با ضریب θ افزایش می‌دهند.



این روش برای شبکه‌های کوچک تعداد تکرارهای بسیار کمتری از BP دارد اما حجم محاسباتی آن بالا و زمان‌گیر است

Conjugate gradient

- در روش نیوتن احتیاج به محاسبه‌ی ماتریس Hessian داریم؛ در واقع محاسبه و ذخیره‌سازی مشتق دوه لازم است.
- هدف یافتن روشی است که **بدون نیاز به محاسبه‌ی مشتق دوه** همگرایی را افزایش دهد.
- از طرفی استفاده از steepest descent باعث حرکت زیزایی به سمت مینیمم می‌شود.
- در صورتی که در راستای بردارهای ویژه‌ی ماتریس Hessian حرکت کنیم، می‌توان انتظار داشت که سرعت همگرایی افزایش یابد.



Conjugate gradient

$$F(\mathbf{x}) = \frac{1}{2} \mathbf{x}^T \mathbf{A} \mathbf{x} + \mathbf{d}^T \mathbf{x} + c$$

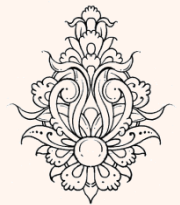
دو بردار نسبت به ماتریس \mathbf{A} (که positive definite است)، conjugate نامیده می‌شوند:

$$\mathbf{p}_k^T \mathbf{A} \mathbf{p}_j = 0 \quad k \neq j$$

ثابت می‌شود در صورتی که جستجو در راستای مجموعه بردارهای conjugate باشد، می‌توان طی یک دور جستجو در راستای این بردارها به مینیمم مطلق رسید (برای توابع درجه‌ی دو).
یک مجموعه از این بردارها، بردارهای ویژه‌ی ماتریس است.

$$\mathbf{z}_k^T \mathbf{A} \mathbf{z}_j = \lambda_j \mathbf{z}_k^T \mathbf{z}_j = 0 \quad k \neq j$$

در صورتی که ماتریس متقارن باشد، بردارهای ویژه‌ی آن متعامد است.



Conjugate gradient

$$\nabla F(\mathbf{x}) = \mathbf{Ax} + \mathbf{d}$$

$$\nabla^2 F(\mathbf{x}) = \mathbf{A}$$

تغییرات گرادیان در گام k -ام

$$\Delta \mathbf{g}_k = \mathbf{g}_{k+1} - \mathbf{g}_k = (\mathbf{Ax}_{k+1} + \mathbf{d}) - (\mathbf{Ax}_k + \mathbf{d}) = \mathbf{A}\Delta \mathbf{x}_k$$

که

$$\Delta \mathbf{x}_k = (\mathbf{x}_{k+1} - \mathbf{x}_k) = \alpha_k \mathbf{p}_k$$

$$\alpha_k \mathbf{p}_k^T \mathbf{A} \mathbf{p}_j = \Delta \mathbf{x}_k^T \mathbf{A} \mathbf{p}_j = \Delta \mathbf{g}_k^T \mathbf{p}_j = 0 \quad k \neq j$$

بدون نیاز به محاسبه بردار Hessian بردار conjugate مناسبه شد.
توجه داشته باشید که به یک مجموعه بردار conjugate نیاز است، در
واقع در گام k -ام به جهتی نیاز است که بر تمام جهتهای زیر عمود
باشد:



Conjugate gradient

- مرحله‌ی اول مشابه steepest decent است.

F تابع معیار فضا است

$$g_0 = \nabla F \Big|_{x=x(0)}$$

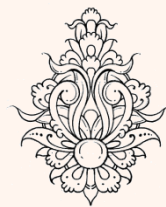
- Direction را به گونه‌ای انتخاب می‌کنیم که بر خلاف مشتق باشد.

$$p_0 = -g_0$$

- در هر iteration بردار p را به گونه‌ای محاسبه می‌کنیم که عمود بر تخییرات گرادیان واقع شود.

$$p_k = -g_k + \beta_k p_{k-1}$$

اثر گرادیان‌های قبلی



آموزش شبکه‌های عصبی **Gram-Schmidt Orthogonalization**

Conjugate gradient

$$p_k = -g_k + \beta_k p_{k-1}$$

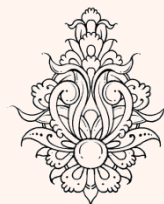
$$p_{k-1} \mathbf{A} p_k = -p_{k-1} \mathbf{A} g_k + \beta_k p_{k-1} \mathbf{A} p_{k-1}$$

صفر

$$\beta_k p_{k-1} \mathbf{A} p_{k-1} = p_{k-1} \mathbf{A} g_k$$

$$\beta_k = \frac{p_{k-1} \mathbf{A} g_k}{p_{k-1} \mathbf{A} p_{k-1}}$$

با توجه به نیاز به ماتریس Hessian از این شیوه
نمی‌توان استفاده کرد



Conjugate gradient

- ضریب β_k می‌تواند از یکی از روش‌های زیر محاسبه گردد:

$$\Delta \mathbf{g}_{k-1}^T = (\mathbf{g}_k - \mathbf{g}_{k-1})^T$$

$$\beta_k = \frac{\Delta \mathbf{g}_{k-1}^T \mathbf{g}_k}{\Delta \mathbf{g}_{k-1}^T \mathbf{p}_{k-1}}$$

Hestenes and Steifel

$$\beta_k = \frac{\mathbf{g}_k^T \mathbf{g}_k}{\mathbf{g}_{k-1}^T \mathbf{g}_{k-1}}$$

Fletcher and Reeves

Polak and Ribiere

$$\beta_k = \frac{\Delta \mathbf{g}_{k-1}^T \mathbf{g}_k}{\mathbf{g}_{k-1}^T \mathbf{g}_{k-1}}$$



Conjugate gradient

• الگوریتم conjugate gradient به صورت خلاصه:

– جهت جستجوی اولیه را به گونه‌ای که بر جهت مشتق عمود

باشد در نظر می‌گیریم
 $p_0 = -g_0$

– مقدار α را که همان نرخ یادگیری است مناسبه می‌نماییم تا

رابطه‌ی زیر به دست آید، با توجه به این که این شیوه به نرخ

آموزش حساس است و نرخ آموزش نیز به ماتریس Hessian

وابسته است، برای تخمین این میزان از روش‌های تکرارشونده

برای مناسبه‌ی نرخ آموزش استفاده می‌شود.

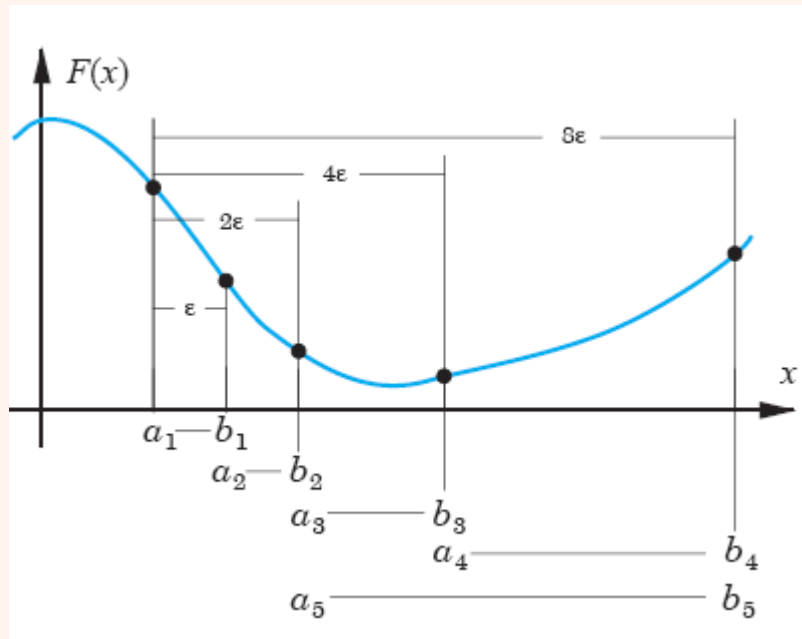
(For quadratic functions.)

$$x_{k+1} = x_k + \alpha_k p_k \quad \alpha_k = - \frac{\nabla F(\mathbf{x})^T \big|_{\mathbf{x} = \mathbf{x}_k} \mathbf{p}_k}{\mathbf{p}_k^T \nabla^2 F(\mathbf{x}) \big|_{\mathbf{x} = \mathbf{x}_k} \mathbf{p}_k} = - \frac{\mathbf{g}_k^T \mathbf{p}_k}{\mathbf{p}_k^T \mathbf{A}_k \mathbf{p}_k}$$

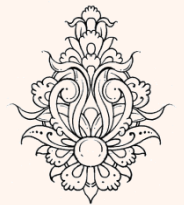


• تخمین نرخ آموزش در دو مرحله صورت می‌گیرد:

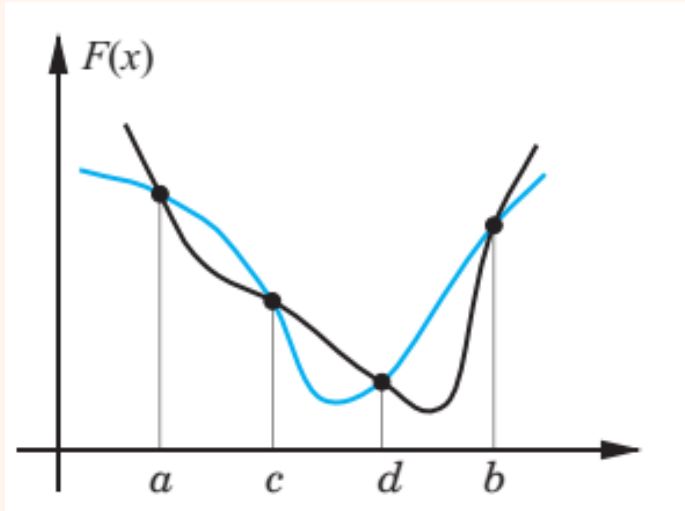
- Interval location
- Interval reduction



Interval location



Conjugate gradient

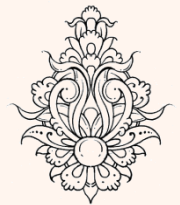


Interval reduction

- با محاسبه β مقدار زیر را که همان جهت بهینه‌ی جستجو است محاسبه می‌شود.

$$p_k = -g_k + \beta_k p_{k-1}$$

- اگر الگوریتم به همگرایی نرسید از گام دو دوباره شروع می‌کنیم.

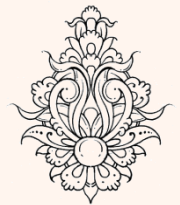


مثال

$$F(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T \begin{bmatrix} 2 & 2 \\ 2 & 4 \end{bmatrix} \mathbf{x} + [1 \ 0]\mathbf{x} \quad \mathbf{x}_0 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$$

$$\nabla F(\mathbf{x}) = \begin{bmatrix} \frac{\partial}{\partial x_1} F(\mathbf{x}) \\ \frac{\partial}{\partial x_2} F(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} 2x_1 + 2x_2 + 1 \\ 2x_1 + 4x_2 \end{bmatrix} \quad \mathbf{p}_0 = -\mathbf{g}_0 = -\nabla F(\mathbf{x})|_{\mathbf{x}=\mathbf{x}_0} = \begin{bmatrix} -3 \\ -3 \end{bmatrix}$$

$$\alpha_0 = -\frac{[3 \ 3] \begin{bmatrix} -3 \\ -3 \end{bmatrix}}{[-3 \ -3] \begin{bmatrix} 2 & 2 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} -3 \\ -3 \end{bmatrix}} = 0.2 \quad \mathbf{x}_1 = \mathbf{x}_0 - \alpha_0 \mathbf{g}_0 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} - 0.2 \begin{bmatrix} 3 \\ 3 \end{bmatrix} = \begin{bmatrix} -0.1 \\ -0.1 \end{bmatrix}$$



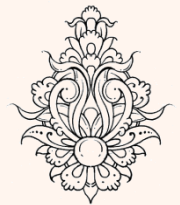
مثال-ادامه

$$\mathbf{g}_1 = \nabla F(\mathbf{x}) \Big|_{\mathbf{x} = \mathbf{x}_1} = \begin{bmatrix} 2 & 2 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} -0.1 \\ -0.1 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0.6 \\ -0.6 \end{bmatrix}$$

$$\beta_1 = \frac{\mathbf{g}_1^T \mathbf{g}_1}{\mathbf{g}_0^T \mathbf{g}_0} = \frac{\begin{bmatrix} 0.6 & -0.6 \end{bmatrix} \begin{bmatrix} 0.6 \\ -0.6 \end{bmatrix}}{\begin{bmatrix} 3 & 3 \end{bmatrix} \begin{bmatrix} 3 \\ 3 \end{bmatrix}} = \frac{0.72}{18} = 0.04$$

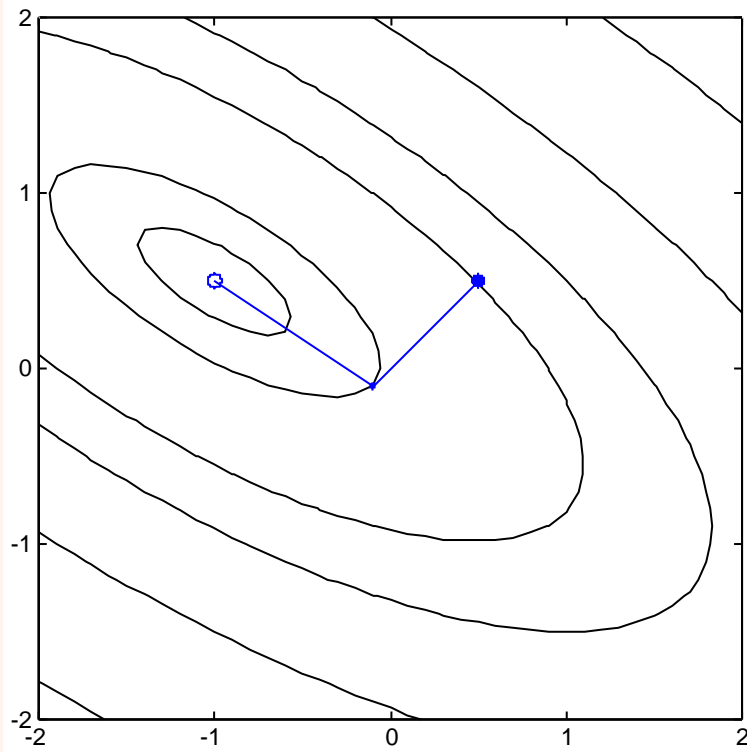
$$\mathbf{p}_1 = -\mathbf{g}_1 + \beta_1 \mathbf{p}_0 = \begin{bmatrix} -0.6 \\ 0.6 \end{bmatrix} + 0.04 \begin{bmatrix} -3 \\ -3 \end{bmatrix} = \begin{bmatrix} -0.72 \\ 0.48 \end{bmatrix}$$

$$\alpha_1 = -\frac{\begin{bmatrix} 0.6 & -0.6 \end{bmatrix} \begin{bmatrix} -0.72 \\ 0.48 \end{bmatrix}}{\begin{bmatrix} -0.72 & 0.48 \end{bmatrix} \begin{bmatrix} 2 & 2 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} -0.72 \\ 0.48 \end{bmatrix}} = -\frac{-0.72}{0.576} = 1.25$$

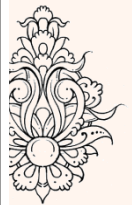
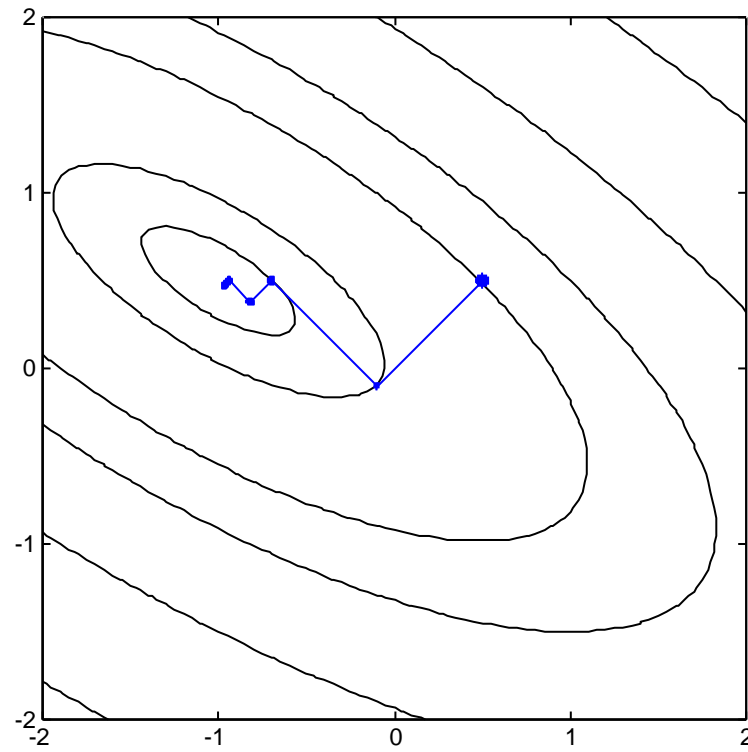


$$\mathbf{x}_2 = \mathbf{x}_1 + \alpha_1 \mathbf{p}_1 = \begin{bmatrix} -0.1 \\ -0.1 \end{bmatrix} + 1.25 \begin{bmatrix} -0.72 \\ 0.48 \end{bmatrix} = \begin{bmatrix} -1 \\ 0.5 \end{bmatrix}$$

Conjugate Gradient



Steepest Descent



تازشک
بهبهشت

روش‌های مطرح بهینه‌سازی



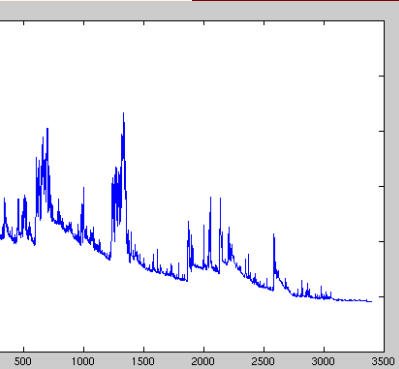
Gradient descent variants

Batch gradient descent (Vanilla gradient descent)

- لازم است برای کل دسته داده گرادیان مناسبه و سپس وزن‌ها به روز شود.
- سرعت به روز شدن پایین است.
- برای موارد آنلاین مناسب نیست.
- امکان انجام محاسبات به صورت موازی

Stochastic gradient descent (SGD)

- به روز شدن برای هر نمونه آموزشی است (در به روز شدن از تقریب گرادیان استفاده می‌شود).
- در مدل Batch افزونگی محاسبات وجود دارد.
- در SGD به روز رسانی برای هر نمونه است و به همین دلیل سرعت بالایی دارد
- برای موارد آنلاین مناسب است.
- به دلیل واریانس زیاد پارامترها در روند به روزرسانی نوسانات زیاد دیده می‌شود.



Mini-batch gradient descent

- از مزایای دو روش قبل استفاده می‌شود.
- واریانس به روزرسانی پارامترها کاهش می‌یابد
- معمولاً تعداد نمونه‌ها در هر batch بین ۵۰ الی ۲۵۶ لحاظ می‌شود.

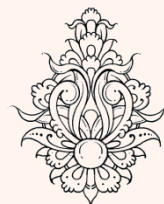


آیا می‌توان به طریقی میزان نوسانات در رسیدن به نقطه‌ی بهینه را کاهش داد؟

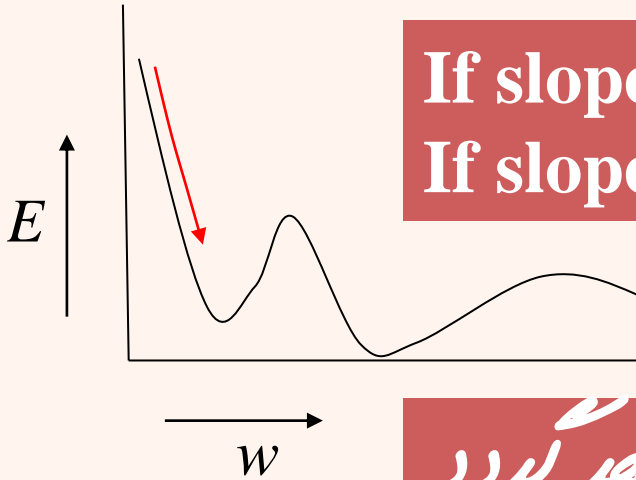
- Momentum
- Nesterov accelerated gradient
- Adagrad
- Adadelta
- RMSprop
- Adam
- Nadam
- AMSGrad

Optimiser	Year	Learning Rate	Gradient
Momentum	1964		✓
AdaGrad	2011	✓	
RMSprop	2012	✓	
Adadelta	2012	✓	
Nesterov	2013		✓
Adam	2014	✓	✓
AdaMax	2015	✓	✓
Nadam	2015	✓	✓
AMSGrad	2018	✓	✓

<https://towardsdatascience.com/10-gradient-descent-optimisation-algorithms-86989510b5e9>

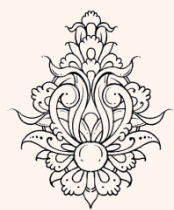
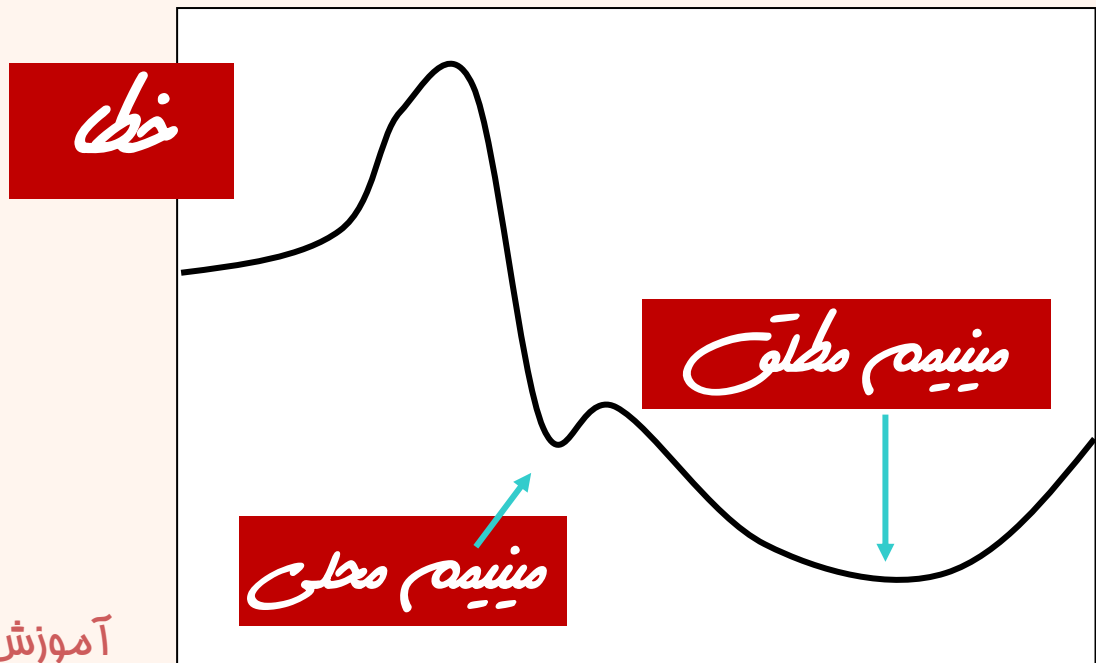


Gradient Descent



If slope is negative \rightarrow increase w
If slope is positive \rightarrow decrease w

مینیمم محلی جایی است که مشتق صفر گردد



تأثیر نرخ آموزش

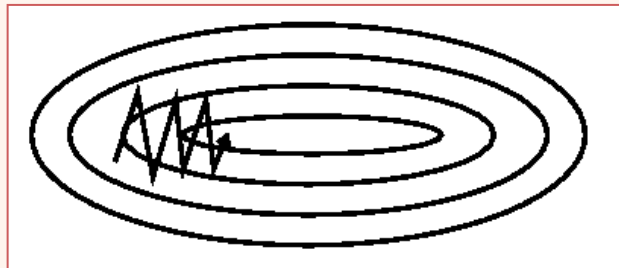
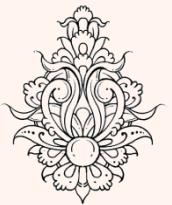
- چنانچه پیش از این مطرح شد، نرخ آموزش بالا موجب ناپایداری می‌شود، در حالی که نرخ آموزش پایین فرآیند آموزش را طولانی خواهد کرد.

نرخ آموزش کوچک: همگرایی کند است ولی روند حرکت بدون تغییرات زیاد

نرخ آموزش بزرگ: همگرایی تند است ولی روند حرکت با تغییرات زیاد

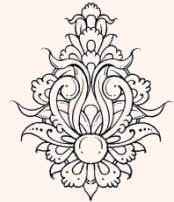
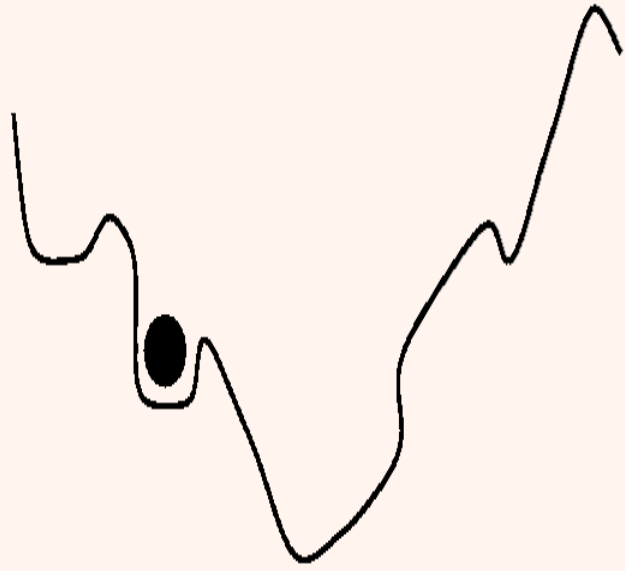
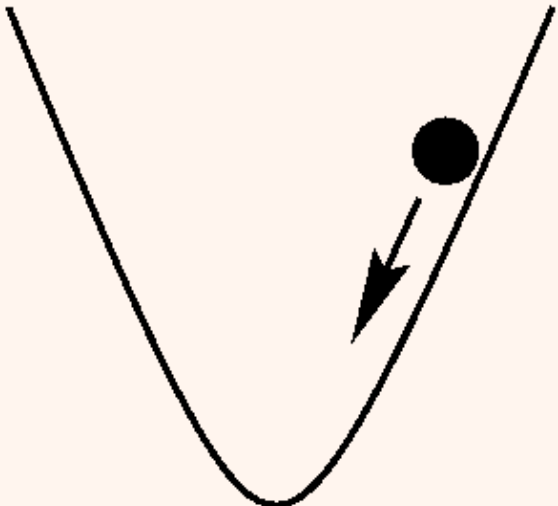
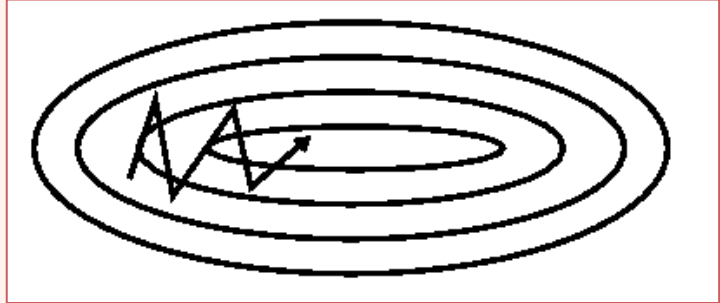
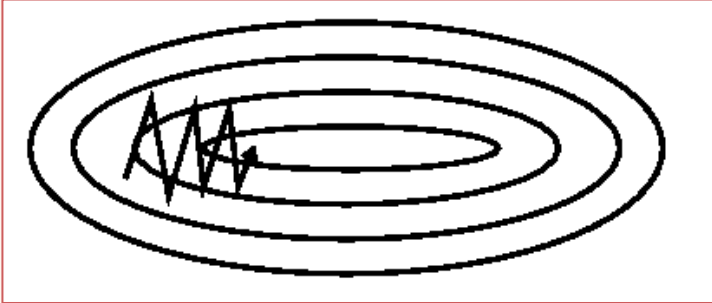
- همچنین می‌توان نرخ آموزش را برای وزن‌های مختلف **متغیر** در نظر گرفت.

Connection dependent



Momentum and Learning Rate Adaptation

Local Minima



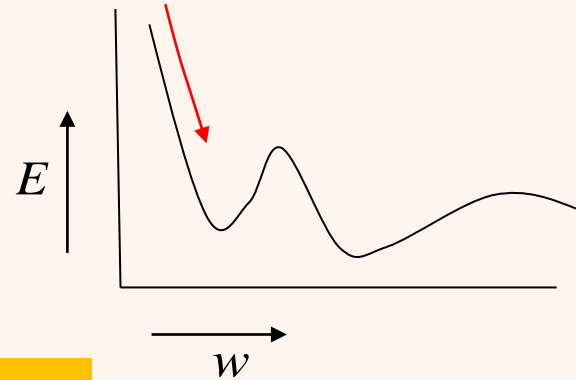
روش استفاده از momentum

$$w_{new} = w_{old} + \Delta w$$

• داشتیم:

generalized delta rule

$$\Delta w_{ji}^l(n) = \alpha \Delta w_{ji}^l(n-1) - \eta \frac{\partial E(n)}{\partial w_{ji}^l(n)}$$

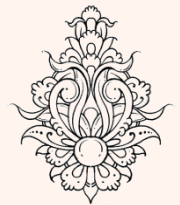


ثابت Momentum $0 \leq \alpha < 1$

تخیرات وزن در مرحله پیشین

$$\Delta w_{ji}^l(n) = \alpha \Delta w_{ji}^l(n-1) + \eta \delta_j^l(n) \cdot y_i^{l-1}(n)$$

$$w_{ji}^l(n+1) = w_{ji}^l(n) + \alpha \Delta w_{ji}^l(n-1) + \eta \delta_j^l(n) y_i^{l-1}(n)$$



به روز رسانی وزن‌ها

روش استفاده از momentum

تغییرات وزن در مرحله پیشین

$$\Delta w_{ji}^l(n) = \alpha \Delta w_{ji}^l(n-1) + \eta \delta_j^l(n) \cdot y_i^{l-1}(n)$$

برای iteration اول بررسی می کنیم:

$$\Delta w_{ji}^l(0) = \eta \delta_j^l(0) \cdot y_i^{l-1}(0)$$

$$\Delta w_{ji}^l(1) = \alpha \eta \delta_j^l(0) \cdot y_i^{l-1}(0) + \eta \delta_j^l(1) \cdot y_i^{l-1}(1)$$

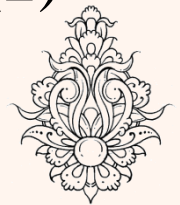
$$\Delta w_{ji}^l(2) = \alpha^2 \eta \delta_j^l(0) \cdot y_i^{l-1}(0) + \alpha \eta \delta_j^l(1) \cdot y_i^{l-1}(1) + \eta \delta_j^l(2) \cdot y_i^{l-1}(2)$$

تاثیر Momentum

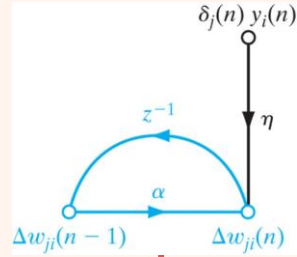
$$\Delta w_{ji}^l(n) = \eta \sum_{h=0}^n \alpha^{n-h} \delta_j^l(h) \cdot y_i^{l-1}(h)$$

نرخ آموزش

ثابت Momentum



روش استفاده از momentum



• اگر ثابت ممنتوم صفر باشد به روز رسانی عادی است.
• هر اندازه این ثابت به سمت یک میل کند به روز رسانی بیشتر از الگوی قبلی تبعیت می‌کند.

- در صورتی که که تاخیرات هم‌جهت باشند، در نظر گرفتن momentum باعث **تسریع** آموزش می‌شود.

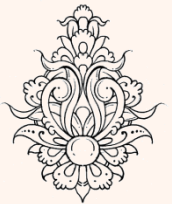
accelerating effect

- در صورت تاخیرات یکسان نباشند، موجب **پایداری** فرآیند آموزش می‌شود.

stabilizing effect

• واضح است که ثابت کمتر از صفر و بیشتر از یک ناکارآمد است.

• با تاخیر میزان نرخ آموزش می‌توان پاسخی بهینه دریافت کرد.



ادامه...

$$\Delta w_{ij}^l(n) = -\eta \sum_{h=0}^n \alpha^{n-h} \frac{\partial E(n)}{\partial w_{ji}(n)}$$

ثابت Momentum	iteration
α	1
α^2	2
.	.
α^{n-h}	n

هرچه iteration بالاتر رود ضریب کوچکتر خواهد شد بدین ترتیب از مقدار بهینه کمتر فاصله خواهیم گرفت



روش استفاده از momentum

$$\Delta w_{ji}^l(n) = -\eta \sum_{h=0}^n \alpha^{n-h} \delta_j^l(h) \cdot y_i^{l-1}(h)$$

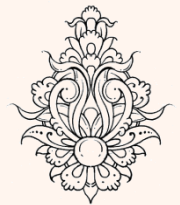
نرخ آموزش

ثابت Momentum

$$\Delta w_{ji}^l(n) = -\eta \sum_{h=0}^n \alpha^{n-h} \frac{\partial E(n)}{\partial w_{ji}(n)}$$

- میانگین زمان یادگیری برای یک شبکه بدون/با استفاده از Momentum

momentum	Training time
۰	۲۱۷
۰.۹	۹۵



نرخ یادگیری و ضریب گشتاور

- هر اندازه نرخ یادگیری کوچکتر باشد تغییرات نمودار کندتر است، اما کمینه‌ی محلی بهتری را می‌تواند به دست آورد. در این شرایط فضای بیشتری از رویه‌ی خطا مورد بررسی قرار می‌گیرد.

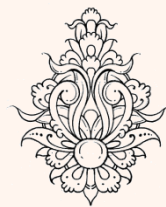
For $\eta \rightarrow 0$, $\alpha \rightarrow 1$

باعث افزایش سرعت همگرایی

For $\eta \rightarrow 1$, $\alpha \rightarrow 0$

باعث ثبات یادگیری

- در صورتی که برای نرخ یادگیری و ضریب گشتاور اعداد ثابت و بزرگی در نظر گرفته شود، باعث خواهد شد میزان خطا به صورت نوسانی تغییر کند و یا به مقدار بالاتری همگرا شود.



روش نرخ یادگیری متغیر (وفاقی)

Adaptive Learning Rate

Bold Driver

ضریب momentum را صفر می‌گذاریم

if $E(k+1) > (1 + \xi)E(k) \longrightarrow w(k+1) = w(k)$

$$\xi > 0.01$$

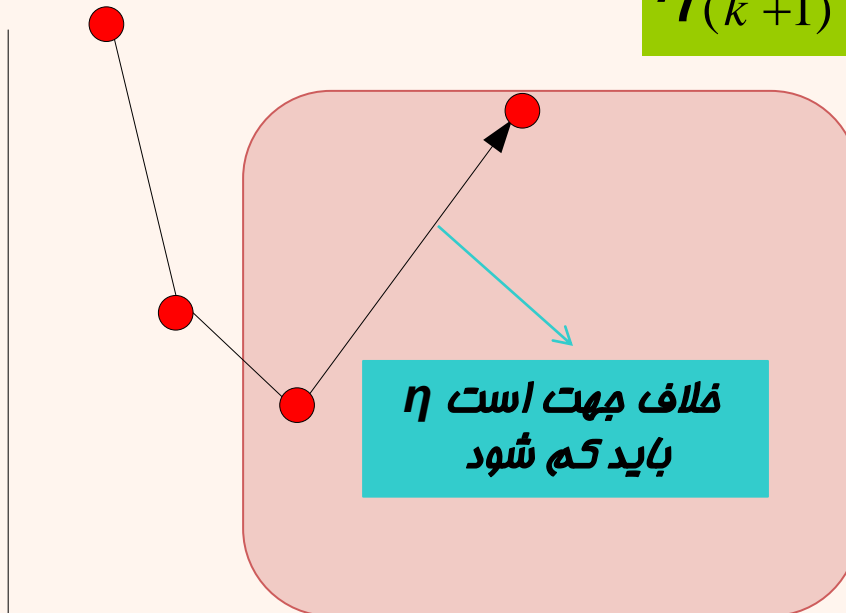
در این حالت جهت حرکت نامناسب است

$$\eta_{(k+1)} = \eta_{(k)} \cdot \rho$$

$$0 < \rho < 1$$

معمولا ۰.۵

SSE



روش نرخ یادگیری متغیر (وفاقی)

ضریب momentum به مقدار اصلی برمی‌گردد

$$\text{if } E(k+1) < E(k) \longrightarrow w(k+1) = w(k) + \Delta w(k)$$

در این حالت جهت حرکت مناسب است

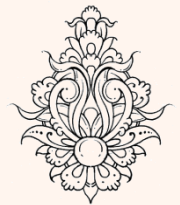
$$\eta_{(k+1)} = \eta_{(k)} \cdot d$$

$$1 < d < 2$$

معمولا ۱.۰۱ تا ۱.۰۵

SSE

η بیشتر شود



روش نرخ یادگیری متغیر (وفاقی)

$$\text{if } E(k+1) > (1 + \beta)E(k)$$

$$\beta < \xi$$

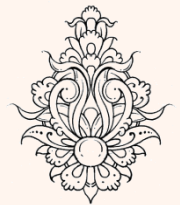
$$w(k+1) = w(k) + \Delta w(k)$$

$$\eta_{(k+1)} = \eta_{(k)}$$

چنانچه شرط قبلی برقرار نباشد

وزن‌های جدید را قبول کرده
ولی η ثابت می‌ماند و α به مقدار
اولیه تغییر داده می‌شود.

این شیوه تنها برای آموزش دسته‌ای مفید است، در
صورت استفاده در روش ترتیبی منجر به واگرایی می‌شود.

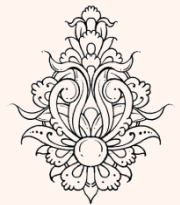


- در روش ترتیبی، استفاده از Bold driver منجر به واگرایی الگوریتم می‌شود، از این رو به جای در نظر گرفتن نرخ و فقی، نرخ آموزش به صورت نزولی در نظر گرفته می‌شود.

$$\eta_{(k)} = \frac{\eta_{(0)}}{1 + \frac{k}{T}}$$

- بدین ترتیب، در گام‌های اولیه نرخ آموزش تقریباً ثابت است. بعد از رسیدن به محدوده می‌نیم، نرخ آموزش کاهش می‌یابد.

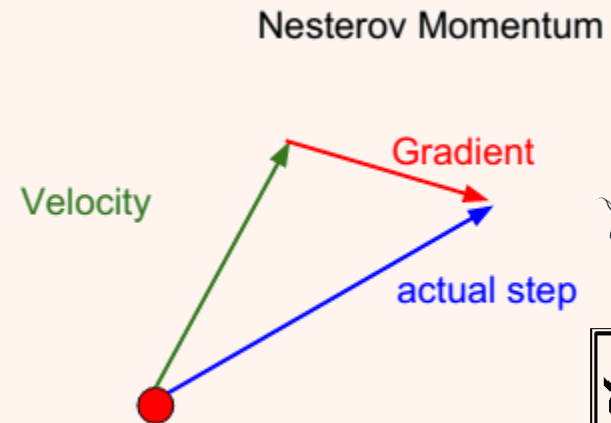
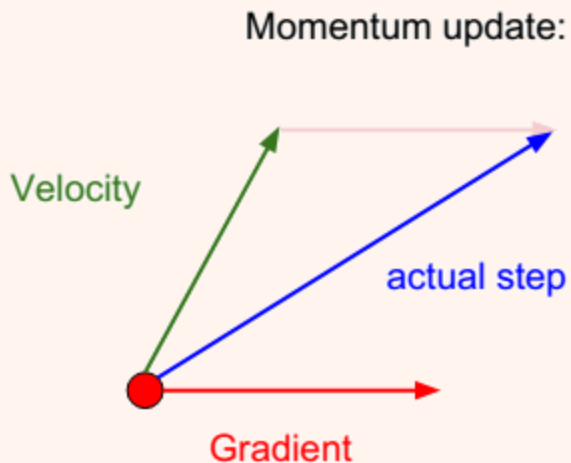
- در این حالت یک هایپرپارامتر آزاد به مجموعه پارامترها افزوده خواهد شد.



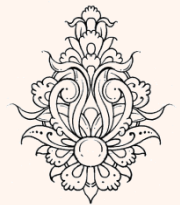
Nesterov accelerated gradient

Nesterov Accelerated Gradient (NAG), is to measure the gradient of the cost function not at the local position but slightly ahead in the direction of the momentum

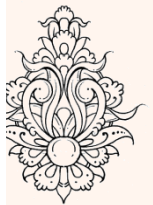
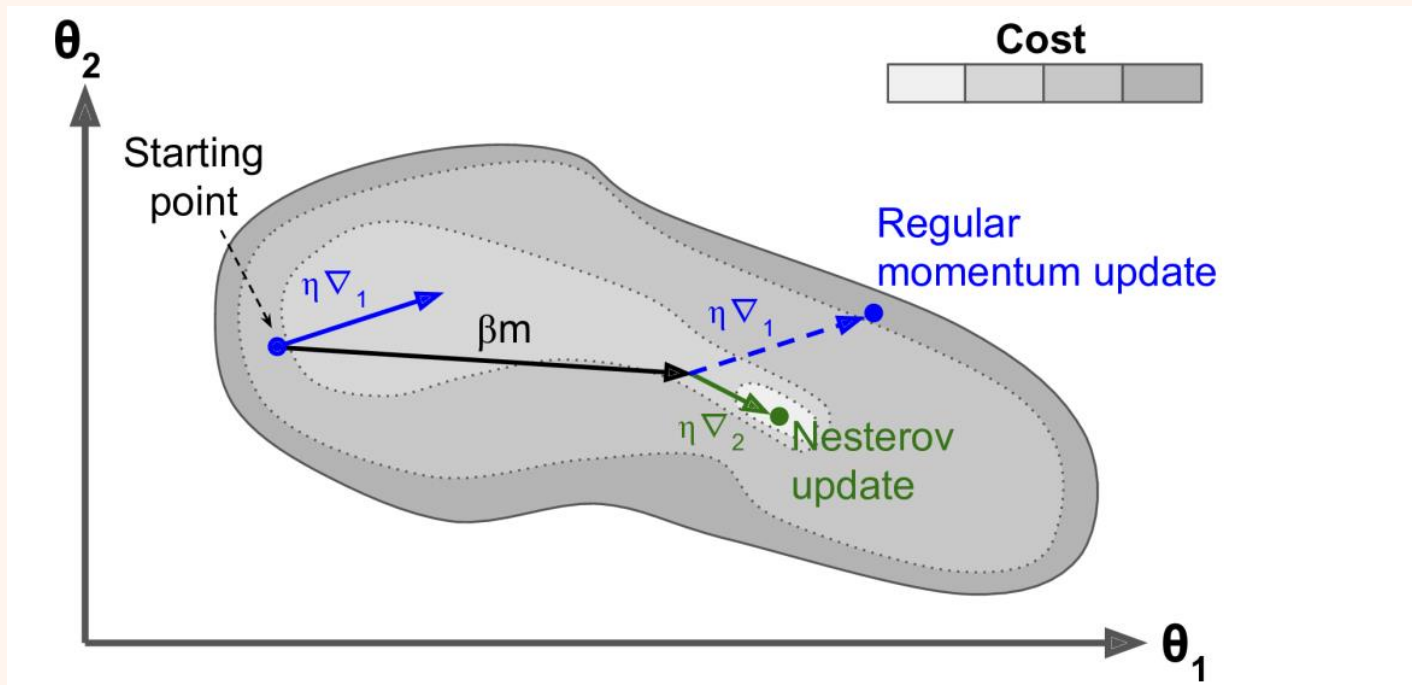
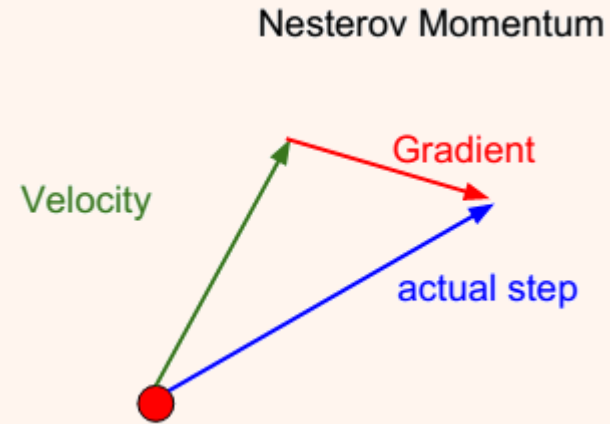
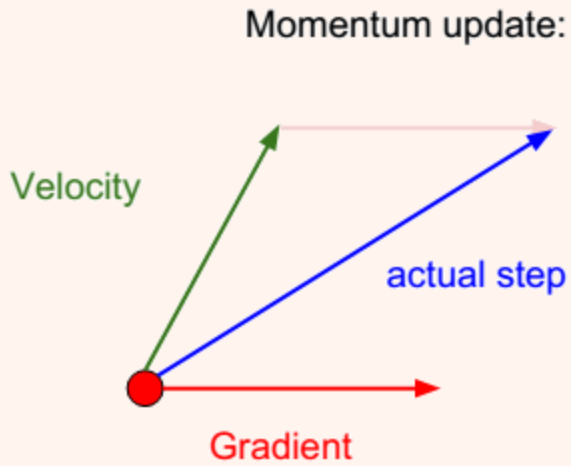
در این شیوه ابتدا بر اساس بردار سرعت موقعیت اصلاح می‌شود و سپس تخیرات گرادیان اعمال می‌شود.

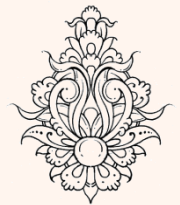
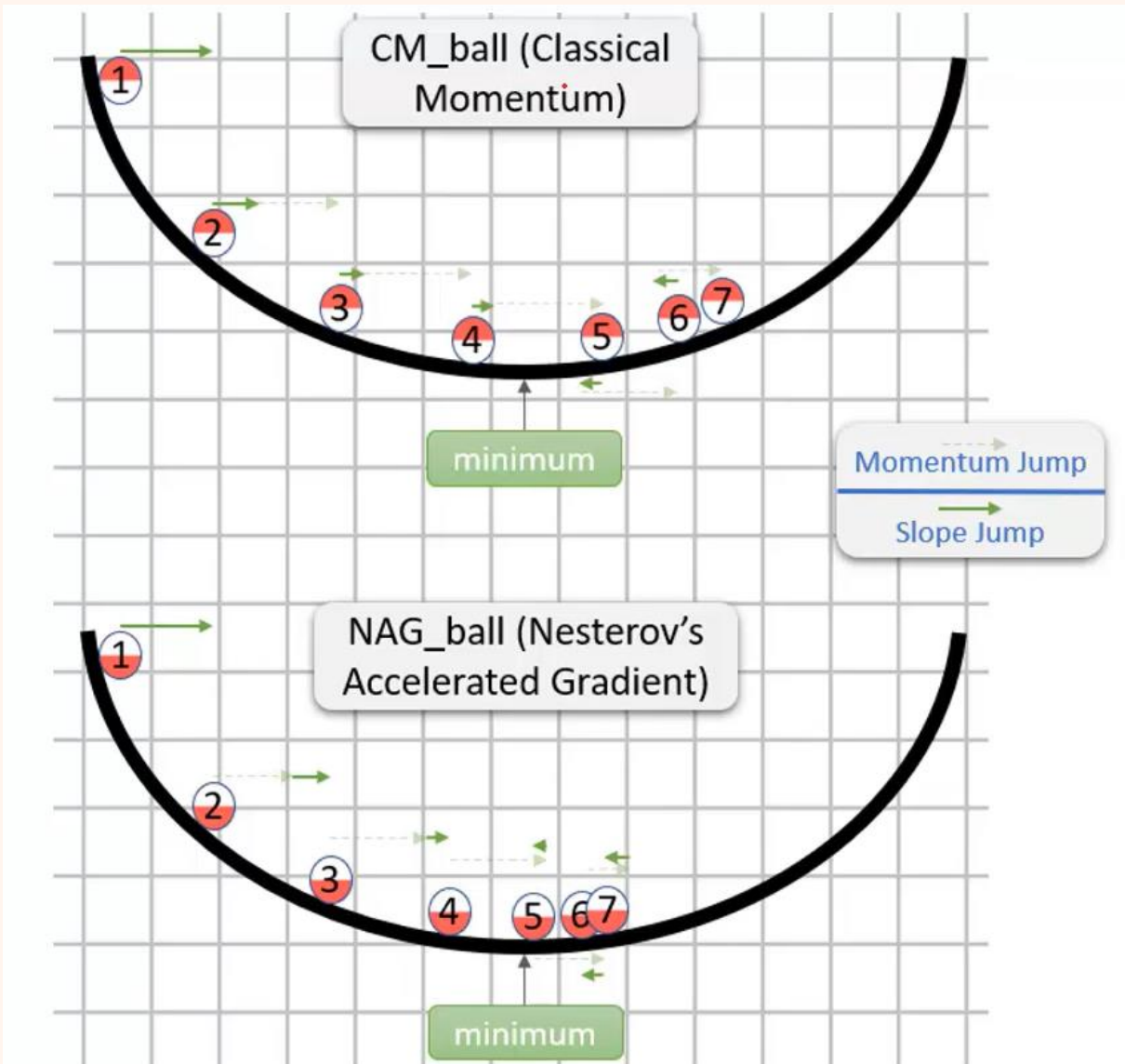


Sutskever, I., et al. (2013). On the importance of initialization and momentum in deep learning. International conference on machine learning, PMLR.

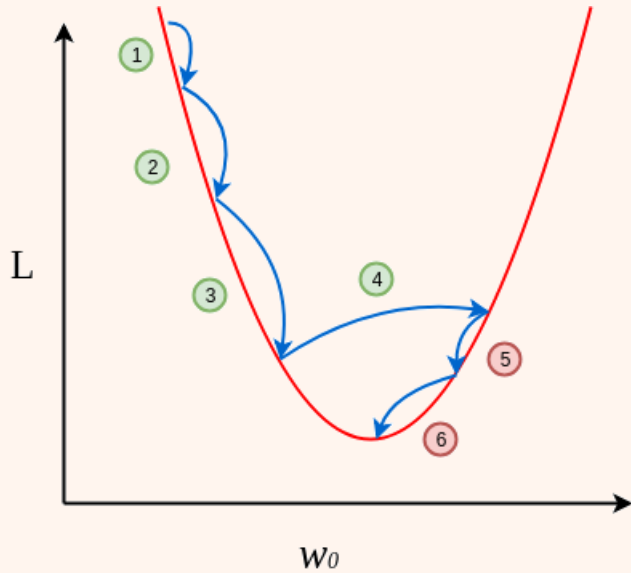


optimizer = keras.optimizers.SGD(lr=0.001, momentum=0.9, nesterov=True)

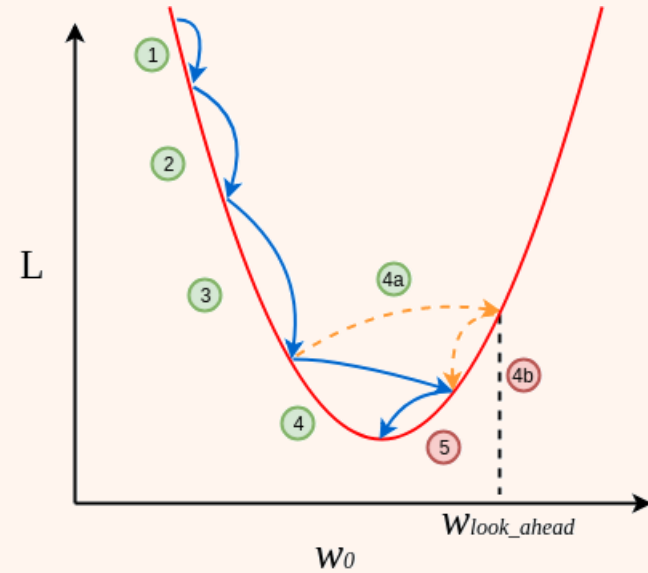




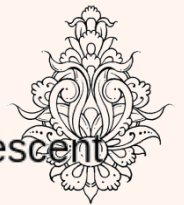
مقایسه



(a) Momentum-Based Gradient Descent



(b) Nesterov Accelerated Gradient Descent



شیوه‌های بهینه‌سازی

SGD

Stochastic gradient descent optimizer.

در این شیوه می‌توان از *momentum*، ضریب یادگیری کاهش‌ی و *nesterov momentum* استفاده کرد.

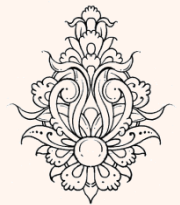
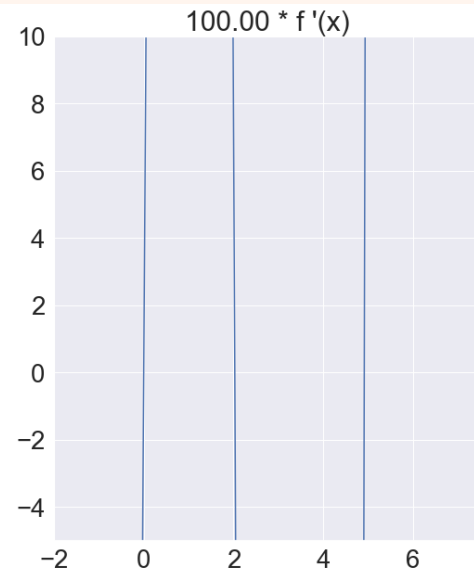
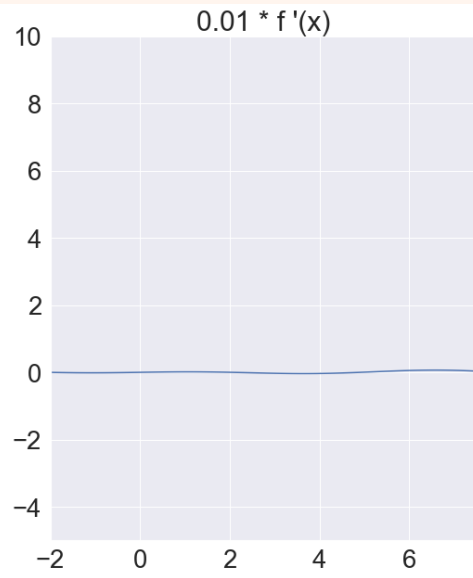
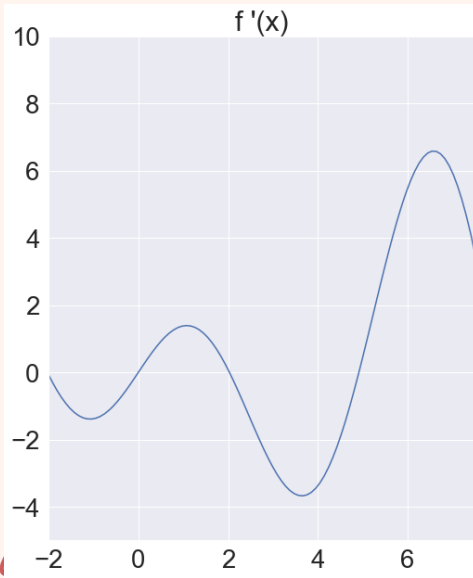
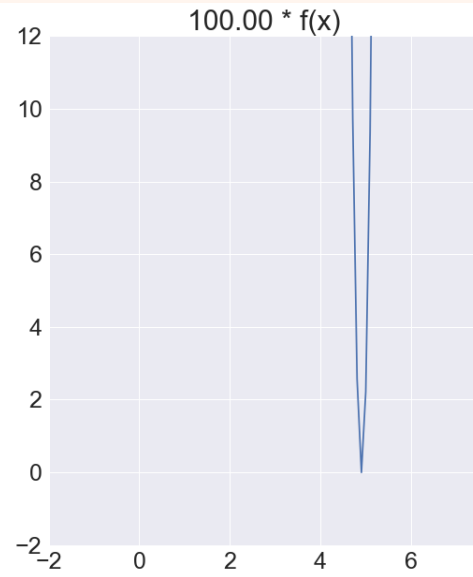
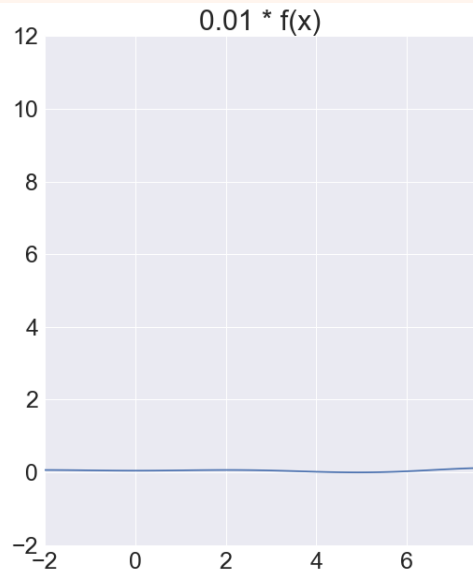
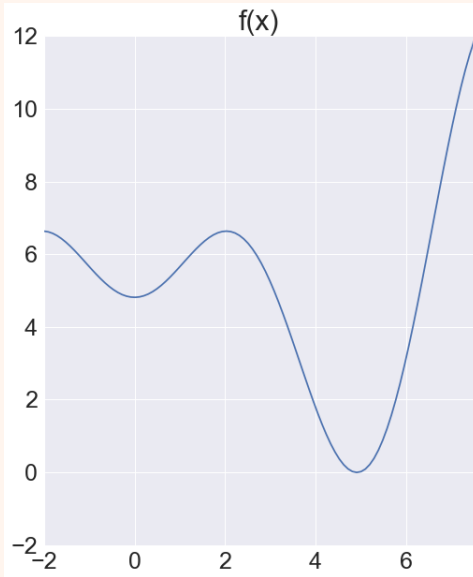
```
keras.optimizers.SGD(lr=0.01, momentum=0.0, decay=0.0, nesterov=False)
```

ضریب یادگیری کاهش‌ی

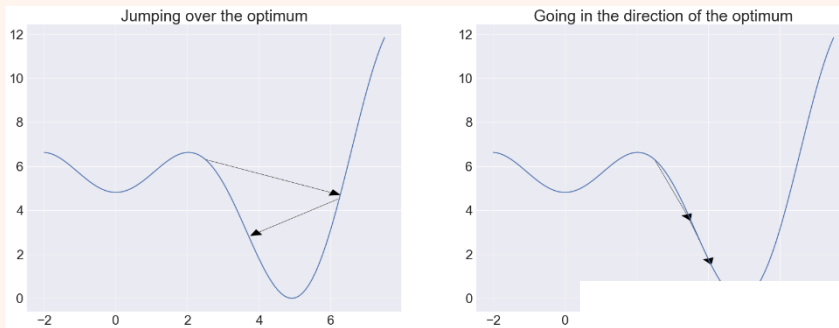
$lr = self.lr * (1. / (1. + self.decay * self.iterations))$

$$\eta_{(k)} = \frac{\eta_{(0)}}{1 + \frac{k}{T}}$$

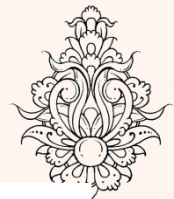




اندازه گرادیان برای وزن‌های مختلف می‌تواند بسیار متفاوت باشد. یک شیوه استفاده از **علامت گرادیان** به ازای هر وزن و در نظر گرفتن اندازه‌ی یکسان برای وزن‌هاست. این شیوه در آموزش دسته‌ای به کار می‌رود. در صورتی که این شیوه به این گونه اصلاح شود که میزان اصلاح بر اساس همسان بودن علامت گرادیان دو مرحله‌ی قبل افزایش/کاهش باید این شیوه **rprop** نامیده می‌شود.



$$w_i^{(t)} = w_i^{(t-1)} - \eta_i^{(t-1)} * \text{sgn} \left(\frac{\partial E^{(t-1)}}{\partial w_i^{(t-1)}} \right)$$



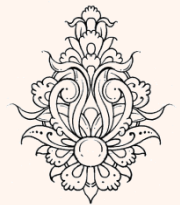
• مزایا:

- در نظر گرفتن گام با اندازه‌های متفاوت برای پارامترها
- این روش در شرایطی که پارامترها در یک بعد به مینیمم نزدیک و در بعد دیگر دور، خوب عمل می‌کند.

• معایب:

- به دسته‌های بزرگ امتیاج دارد، چنانچه تغییرات تصادفی زیاد باشد (در تابع خطا)، همگرایی خوبی نخواهد داشت.

$$\eta_i^{(t)} = \begin{cases} \min(\eta_i^{(t-1)} * \alpha, \eta_{\max}) & \text{if } \frac{\partial E^{(t)}}{\partial w_i^{(t)}} * \frac{\partial E^{(t-1)}}{\partial w_i^{(t-1)}} > 0 \\ \max(\eta_i^{(t-1)} * \beta, \eta_{\min}) & \text{if } \frac{\partial E^{(t)}}{\partial w_i^{(t)}} * \frac{\partial E^{(t-1)}}{\partial w_i^{(t-1)}} < 0 \\ \eta_i^{(t-1)} & \text{otherwise} \end{cases}$$



Adagrad

Adaptive gradient

`keras.optimizers.Adagrad(lr=0.01, epsilon=None, decay=0.0)`

In its update rule, Adagrad modifies the general learning rate η at each time step t for every parameter based on the past gradients

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{S_t + \epsilon}} \cdot \frac{\partial L}{\partial w_t} \quad S_t = S_{t-1} + \left[\frac{\partial L}{\partial w_t} \right]^2$$

باعث می شود یادگیری در همه ابعاد به صورت یکنواخت صورت گیرد. این مساله در مدل های عمیق که گرادیان لایه های مختلف می تواند بسیار متفاوت باشد، بسیار مفید است. عملاً عملکردی مشابه **annealing** دارد.

به مقادیر اولیه حساس است. اگر گرادیان در ابتدای فرآیند بزرگ باشد، در کل فرآیند نرخ آموزش پایین خواهد بود.



Duchi, John, Elad Hazan, and Yoram Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization." *Journal of machine learning research* 12, no. 7 (2011).



RMSprop

```
optimizer = keras.optimizers.RMSprop(lr=0.001, rho=0.9)
```

RMSprop and Adadelta have both been developed independently around the same time stemming from the need to resolve Adagrad's radically diminishing learning rates.

RMSprop as well divides the learning rate by an exponentially Moving average of squared gradients.

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{S_t + \epsilon}} \cdot \frac{\partial L}{\partial w_t}$$

$$S_t = \beta S_{t-1} + (1 - \beta) \left[\frac{\partial L}{\partial w_t} \right]^2$$

Exponentially weighted averages:

$$v_t = \beta v_{t-1} + (1 - \beta) \theta_t$$

$$v_{100} = 0.9v_{99} + 0.1\theta_{100}$$

$$v_{99} = 0.9v_{98} + 0.1\theta_{99}$$

$$v_{98} = 0.9v_{97} + 0.1\theta_{98}$$

دانشگاه
شهید
بهشتی

Exponentially Weighted Averages

Exponentially weighted averages

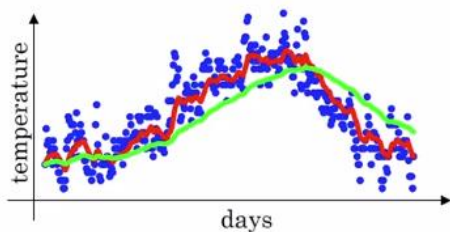
$$v_t = \beta v_{t-1} + (1-\beta)\theta_t$$

$\beta = 0.9$: ≈ 10 days' temper.

$\beta = 0.98$: ≈ 50 days

v_t is an exponentially
moving aver.
 $\approx \frac{1}{1-\beta}$ days'
temperature.

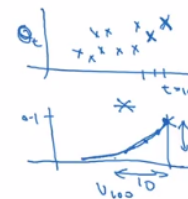
$$\frac{1}{1-0.98} = 50$$



Andrew Ng

Exponentially weighted averages

$$v_t = \beta v_{t-1} + (1-\beta)\theta_t$$



$$v_{100} = 0.9v_{99} + 0.1\theta_{100}$$

$$v_{99} = 0.9v_{98} + 0.1\theta_{99}$$

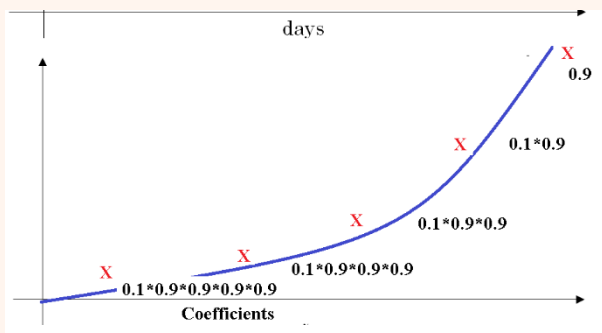
$$v_{98} = 0.9v_{97} + 0.1\theta_{98}$$

...

$$v_{100} = 0.1\theta_{100} + 0.9(0.1\theta_{99} + 0.9v_{98})$$

$$= 0.1\theta_{100} + 0.1 \times 0.9 \theta_{99} + 0.1(0.9)^2 \theta_{98} + 0.1(0.9)^3 \theta_{97} + \dots$$

$0.9^{10} \approx 0.35 \approx \frac{1}{2}$ $\frac{(1-\epsilon)^{1/\epsilon}}{\epsilon} = \frac{1}{\epsilon}$ $0.98^{50} \approx \frac{1}{2}$



Andrew Ng



RMSprop

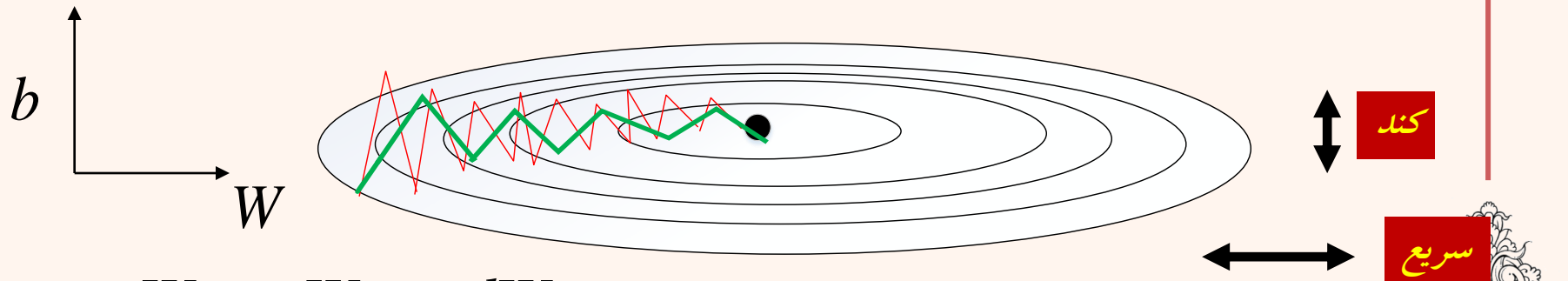
A mini-batch version of rprop

در این شیوه باید دسته‌های داده‌ی مورد استفاده متعادل باشد.

minibatch از نظر محاسباتی به صرفه‌تر است.

گرایان به دست آمده بر مجذور میانگین مربعات تقسیم می‌شود!

استفاده از نرخ یادگیری بالاتر باعث همگرایی سریع‌تر میشود.



$$W_{t+1} = W_t - \eta dW$$

$$b_{t+1} = b_t - \eta db$$

$$w_{t+1} = w_t - \frac{\alpha}{\sqrt{S_t + \epsilon}} \cdot \frac{\partial L}{\partial w_t}$$

ضریب یادگیری برای **dw** بهتر است بالاتر لحاظ شود



$$S_t = \beta S_{t-1} + (1 - \beta) \left[\frac{\partial L}{\partial w_t} \right]^2$$

Accumulation of the squared gradients in the denominator causes the learning rate to shrink and eventually become infinitesimally small, at which point we acquire additional

3.1. Idea 1: Accumulate Over Window

Instead of accumulating all past squared gradients, Adadelta restricts the window of accumulated past gradients to some fixed size w .

`keras.optimizers.Adadelta(lr=1.0, rho=0.95, epsilon=None, decay=0.0)`

$$w_{t+1} = w_t - \frac{\sqrt{D_{t-1} + \epsilon}}{\sqrt{v_t + \epsilon}} \cdot \frac{\partial L}{\partial w_t}$$

where

$$D_t = \beta D_{t-1} + (1 - \beta) [\Delta w_t]^2$$
$$v_t = \beta v_{t-1} + (1 - \beta) \left[\frac{\partial L}{\partial w_t} \right]^2$$

with D and v initialised to 0, and

$$\Delta w_t = w_t - w_{t-1}$$



Zeiler, Matthew D. "Adadelta: An Adaptive Learning Rate Method."

arXiv preprint arXiv:1212.5701 (2012).

3.2. Idea 2: Correct Units with Hessian Approximation

$$\Delta\theta = -H^{-1}g. \quad \Delta\theta = \frac{\frac{\partial f}{\partial\theta}}{\frac{\partial^2 f}{\partial\theta^2}}$$

$$\Delta\theta = -\frac{\Delta\theta}{\frac{\partial f}{\partial\theta}}g.$$

$$\frac{1}{\frac{\partial^2 f}{\partial\theta^2}} = \frac{\Delta\theta}{\frac{\partial f}{\partial\theta}},$$
$$H^{-1} = \frac{\Delta\theta}{\frac{\partial f}{\partial\theta}},$$

$$w_{t+1} = w_t - \frac{\sqrt{D_{t-1} + \epsilon}}{\sqrt{v_t + \epsilon}} \cdot \frac{\partial L}{\partial w_t}$$

where

$$D_t = \beta D_{t-1} + (1 - \beta)[\Delta w_t]^2$$

$$v_t = \beta v_{t-1} + (1 - \beta) \left[\frac{\partial L}{\partial w_t} \right]^2$$

with D and v initialised to 0, and

$$\Delta w_t = w_t - w_{t-1}$$



ADAM

Adaptive moment estimation



momentum

RMSprop

```
keras.optimizers.Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=None, decay=0.0, amsgrad=False)
```

Adaptive Moment Estimation (Adam) is another method that computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients like Adadelta and RMSprop, Adam also keeps an exponentially decaying average of past gradients, similar to momentum. Whereas momentum can be seen as a ball running down a slope, Adam behaves like a heavy ball with friction, which

thus prefers flat minima on the error surface

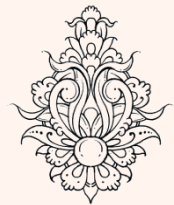
$$w_{t+1} = w_t - \frac{V_t}{\sqrt{\hat{S}_t + \epsilon}} \cdot \hat{V}_t$$

Bias Correction

$$\hat{V}_t = \frac{V_t}{1 - \beta_1^t}$$

$$\hat{S}_t = \frac{S_t}{1 - \beta_2^t}$$

$$V_t = \beta_1 V_{t-1} + (1 - \beta_1) \frac{\partial L}{\partial w_t}$$
$$S_t = \beta_2 S_{t-1} + (1 - \beta_2) \left[\frac{\partial L}{\partial w_t} \right]^2$$



Kingma, Diederik P, and Jimmy Ba. "Adam: A Method for Stochastic Optimization." arXiv preprint arXiv:1412.6980 (2014).

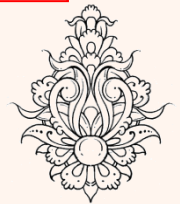
NADAM

Nadam ([Dozat, 2015](#)) is an acronym for Nesterov and Adam optimiser.

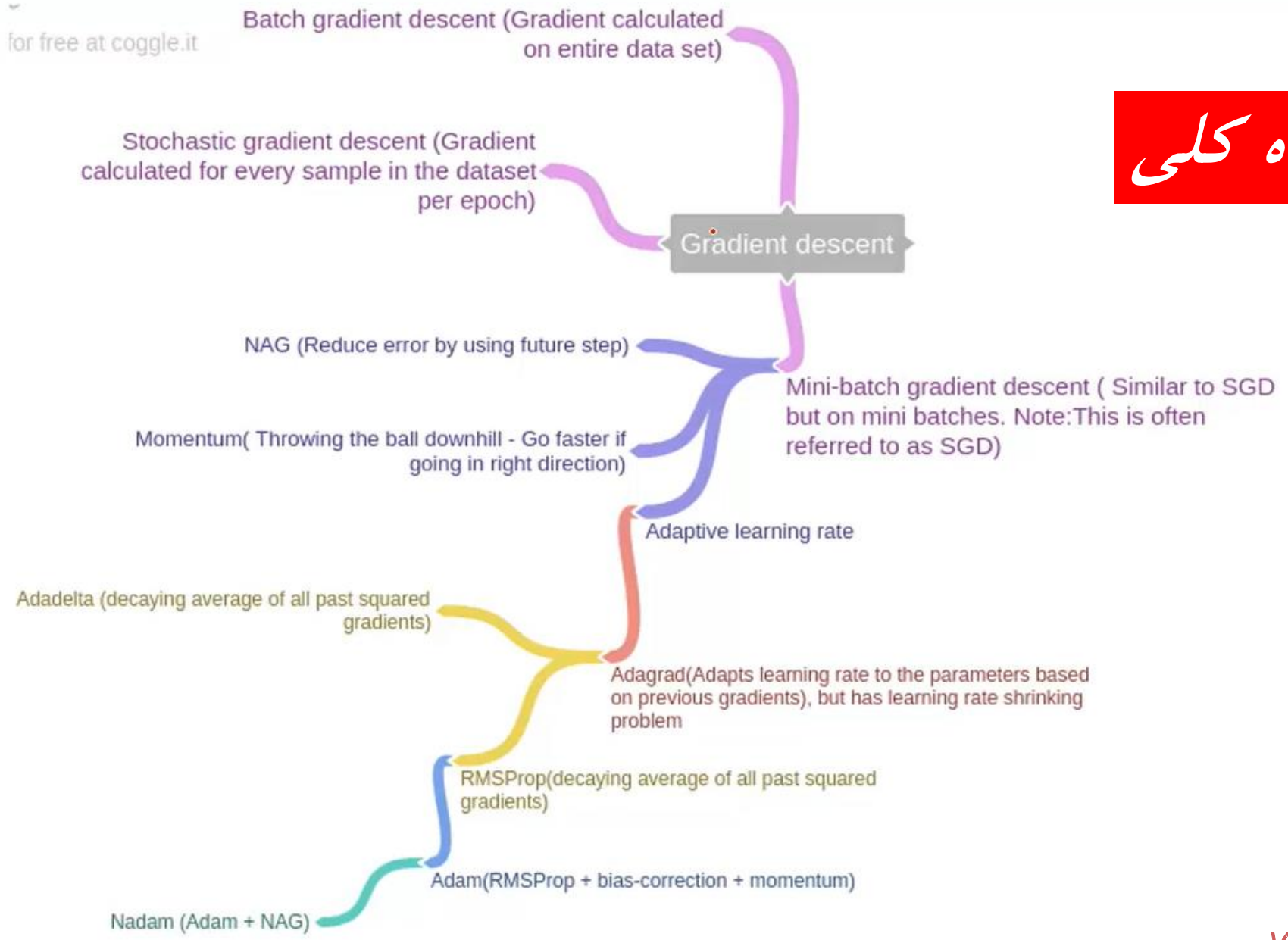
AMSGrad

```
avg_grads = beta1 * avg_grads + (1-beta1) * w.grad
avg_squared = beta2 * (avg_squared) + (1-beta2) * (w.grad ** 2)
w = w - lr * avg_grads / sqrt(avg_squared)
```

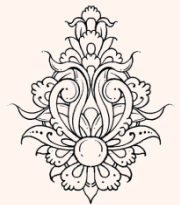
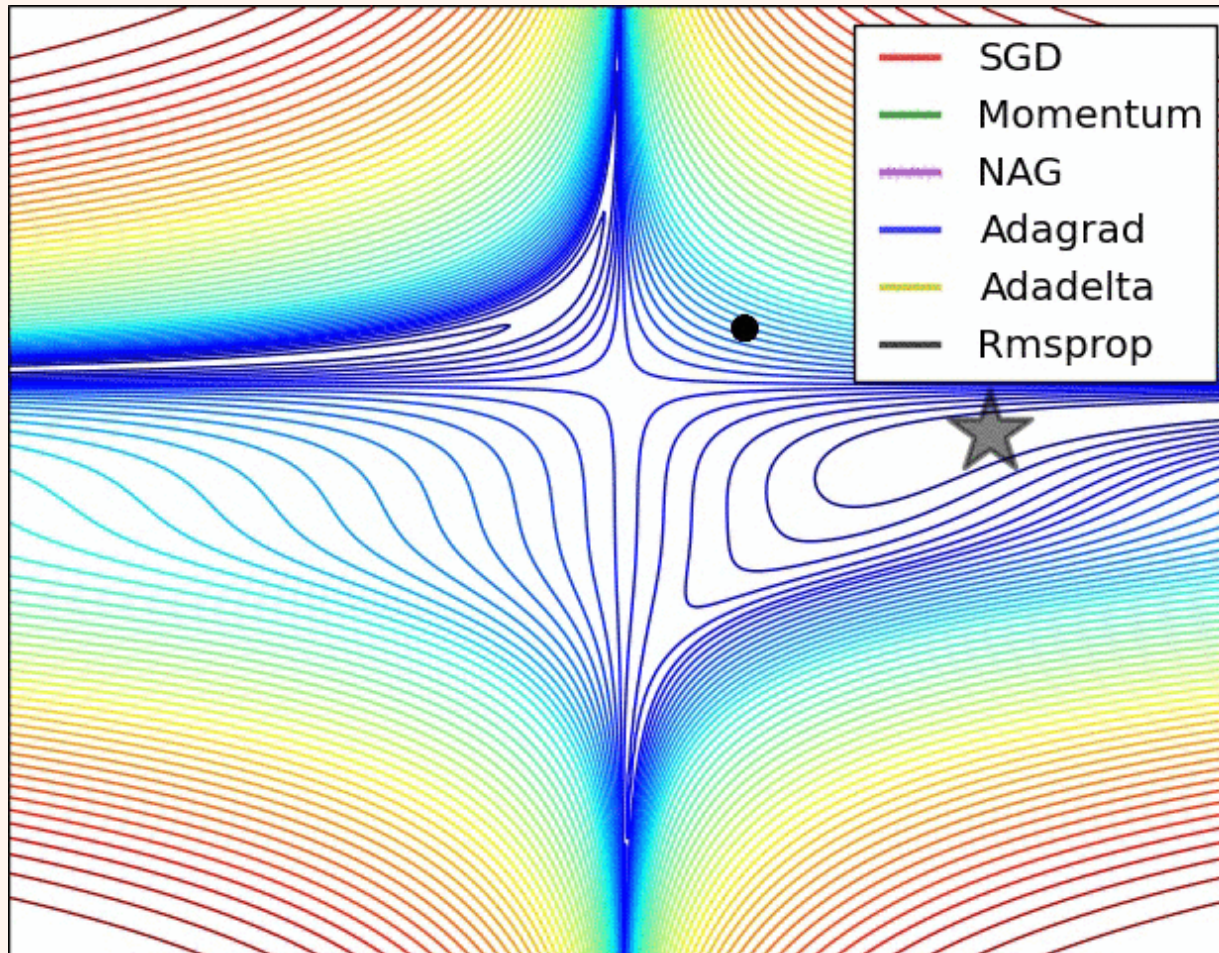
```
avg_grads = beta1 * avg_grads + (1-beta1) * w.grad
avg_squared = beta2 * (avg_squared) + (1-beta2) * (w.grad ** 2)
max_squared = max(avg_squared, max_squared)
w = w - lr * avg_grads / sqrt(max_squared)
```



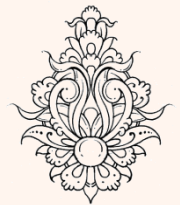
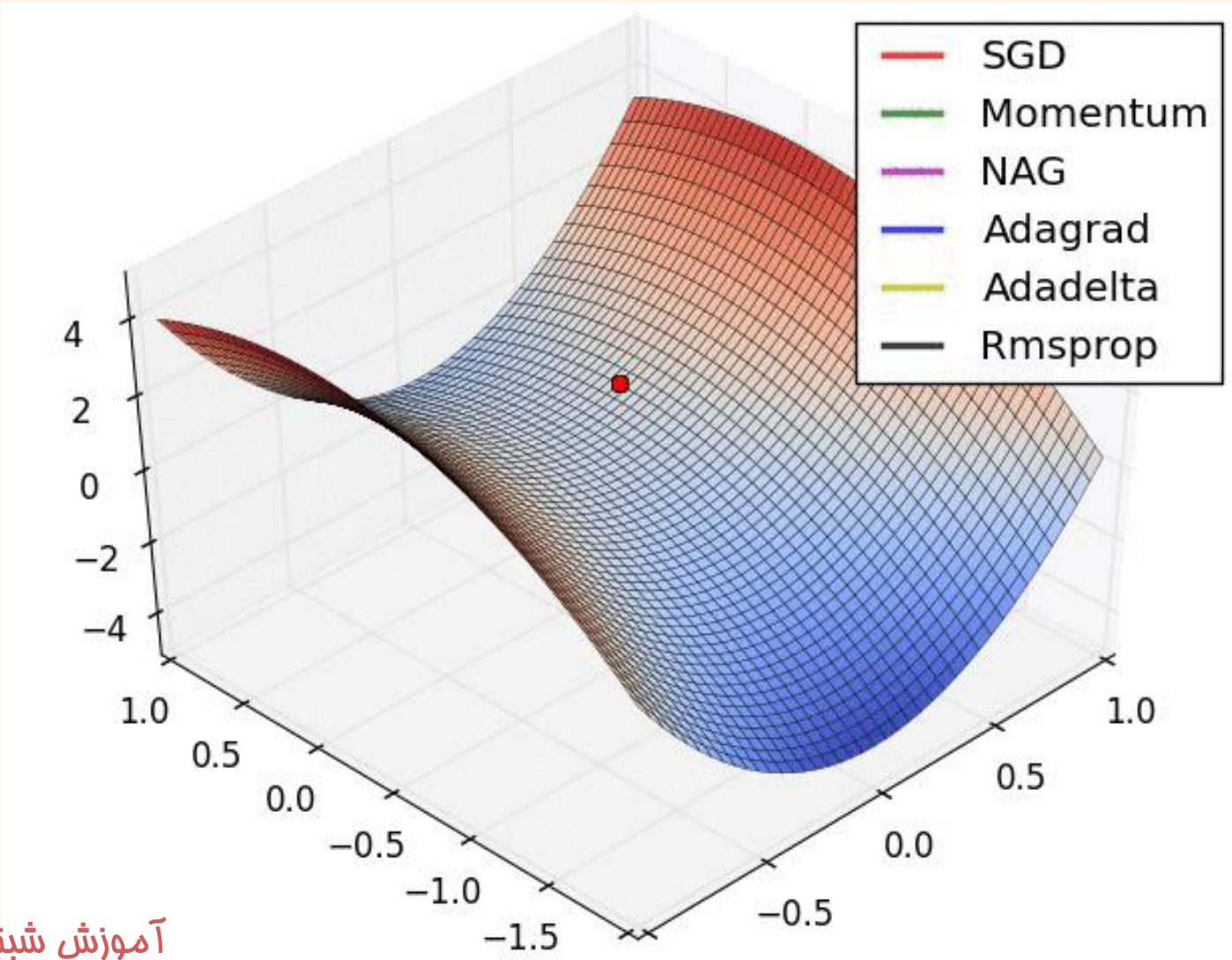
for free at coggle.it



<http://ruder.io/optimizing-gradient-descent/>



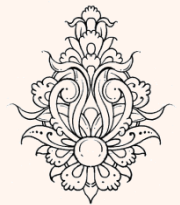
<http://ruder.io/optimizing-gradient-descent/>



مقایسه

<http://ruder.io/optimizing-gradient-descent/>

- در شرایط یکسان RMSprop, Adadelta, Adam عملکرد خوبی دارند.
- مساله‌ی bias-correction باعث می‌شود عملکرد Adam نسبت به RMSprop کمی بهتر باشد.
- در بسیاری موارد SGD با در نظر گرفتن Annealing انتخاب مناسبی است. هر چند این روش به نقطه مطلوب می‌رسد، سرعت آن پایین است.
- در مواردی که آموزش عمیق مد نظر است یا هدف بالا بردن سرعت همگرایی باشد انتخاب الگوریتمی با نرخ آموزش و فقی مناسب است.

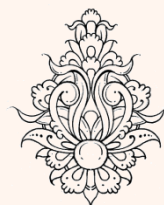


The Forward-Forward algorithm (FF)



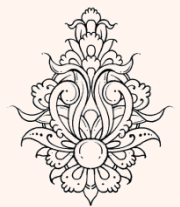
“As a model of how cortex learns, backpropagation remains implausible despite considerable effort to invent ways in which it could be implemented by real neurons,” Hinton writes. “There is no convincing evidence that cortex explicitly propagates error derivatives or stores neural activities for use in a subsequent backward pass.”

“the perceptual system needs to perform inference and learning in real time without stopping to perform backpropagation,” Hinton writes.

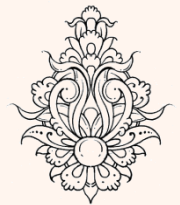


Backpropagation also doesn't work if the computation done in the forward pass is **not differentiable**. "If we insert a black box into the forward pass, it is no longer possible to perform backpropagation unless we learn a differentiable model of the black box," Hinton writes.

Unlike backpropagation, the FF algorithm also works if it contains black-box modules. Since the algorithm does not require differentiable functions, it can still tune its trainable parameters without knowing the inner workings of every layer in the model.

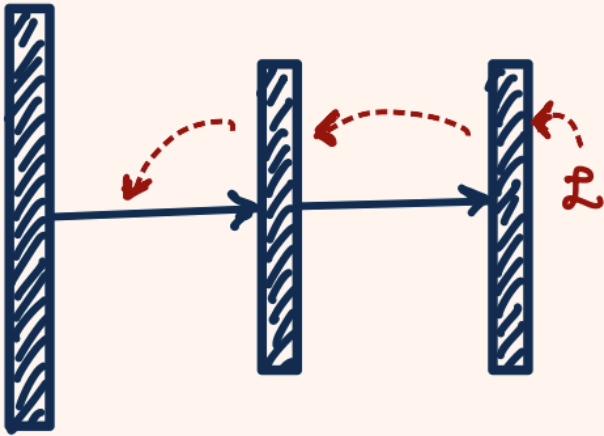


“The Forward-Forward algorithm (FF) is comparable in speed to backpropagation but has the advantage that it can be used when the precise details of the forward computation are unknown,” Hinton writes.

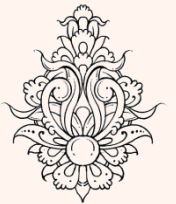
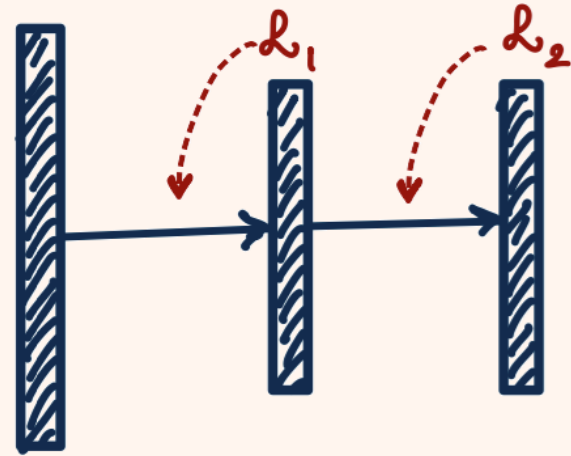


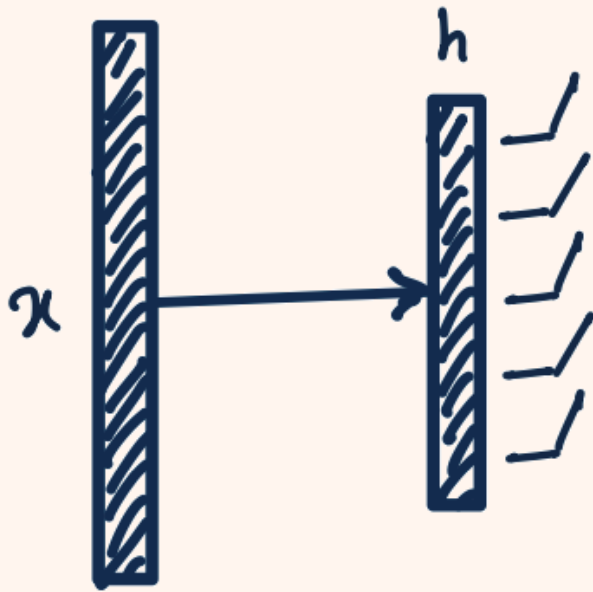
Forward-Forward

Back Propagation



Forward-Forward





↗ Goodness

$$G(x) := \sum \text{relu}(h_i)^2$$

We want:

$$G(x_{\text{pos}}) \gg \text{threshold}$$

$$G(x_{\text{neg}}) \ll \text{threshold}$$

