شبکه‌های عصبی مصنوعی
۱۳-۱۱-۷۱۳-۰۱
بخش دوم

MLP

Error Back propagation

دانشگاه شهید بهشتی

دانشکده‌ی مهندسی برق و کامپیوتر

زمستان ۱۳۹۳

احمد محمودی ازناوه

http://faculties.sbu.ac.ir/~a_mahmoudi/

فهرست مطالب

- شبکه‌ی عصبی چندلایه
- آموزش
- الگوریتم پس‌انتشار خطا
  - Momentum
  - انواع آموزش
  - توقف آموزش
- نکاتی برای تسریع آموزش
  - بردارهای ورودی

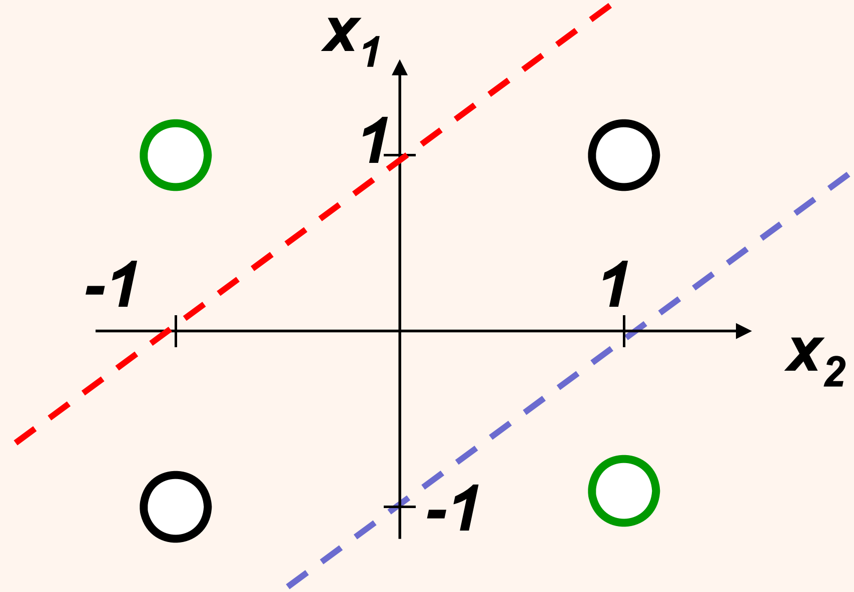
فهرست مطالب(ادامه...)

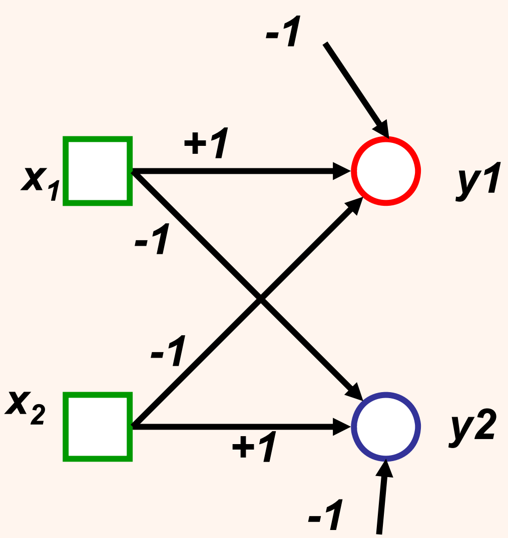
  - تابع انگیزش
  - نرخ یادگیری
  - تابع هزینه
- الگوریتم‌های بهینه‌سازی
  - Gradient descent
- روش نیوتن
- Levenberg-Marqualt
- Conjuagte gradient

مثال

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

x_1	x_2	x_1 xor x_2
-1	-1	**-1**
-1	1	**1**
1	-1	**1**
1	1	**-1**



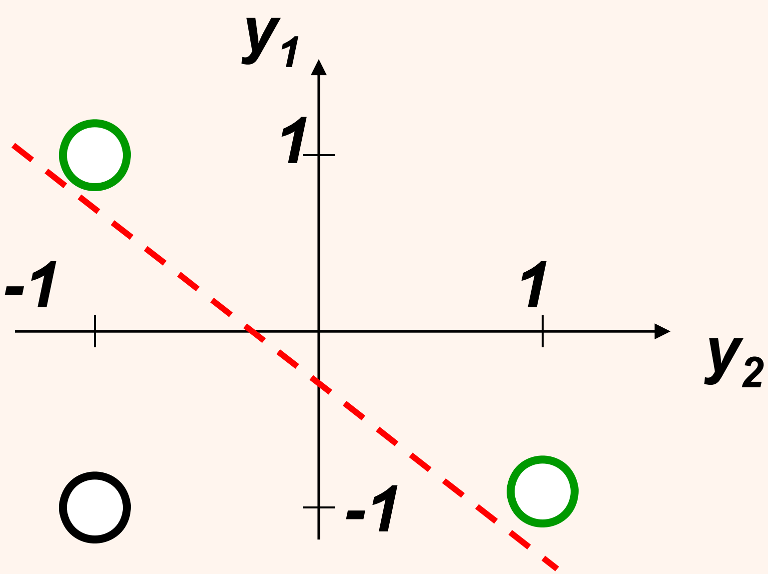


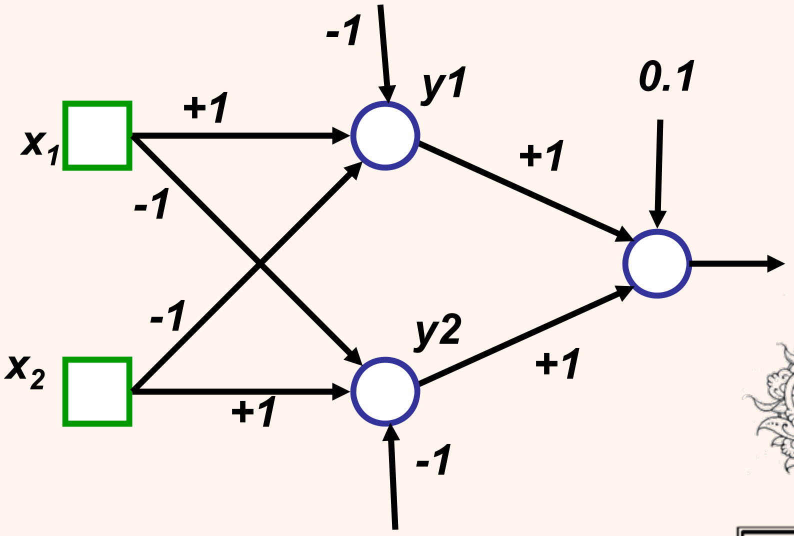
$$\varphi(v) = \begin{cases} 1 & \text{if } v > 0 \\ -1 & \text{if } v \le 0 \end{cases}$$

φ is the sign function.

مثال

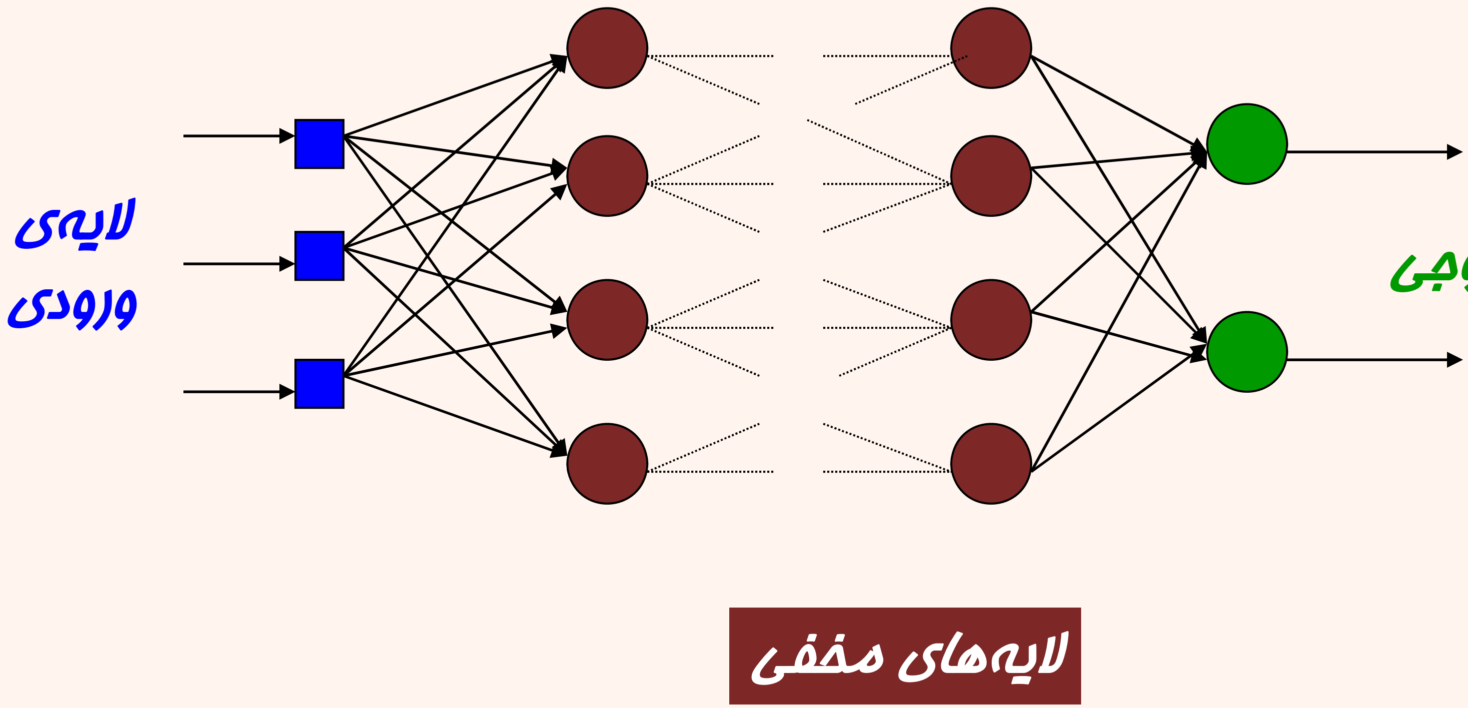
x_1	x_2	y1	y2	x_1 xor x_2
-1	-1	-1	-1	-1
-1	1	-1	1	1
1	-1	1	-1	1
1	1	-1	-1	-1





Multilayer Neural Network

Multilayer Neural Perceptron



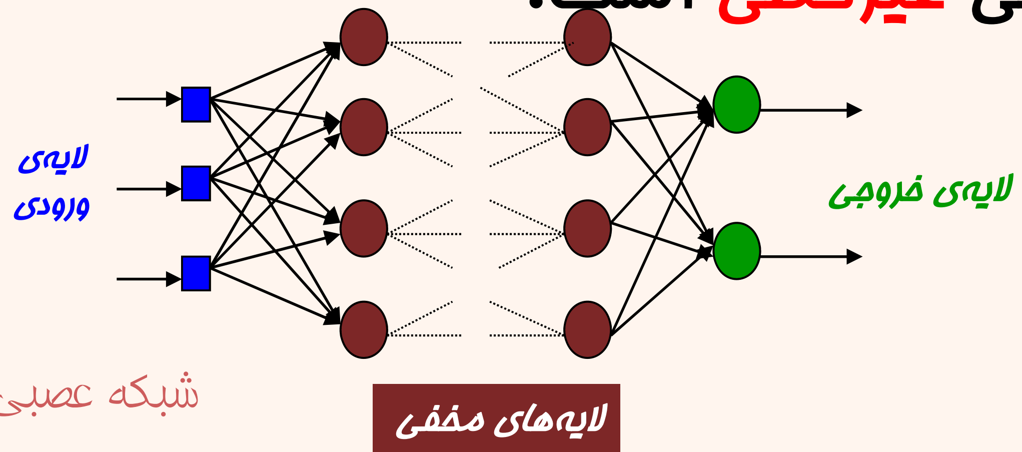
شبکه عصبی چند لایه

Multilayer Neural Network

Multilayer Neural Perceptron(MLP)

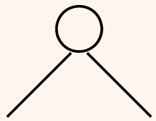
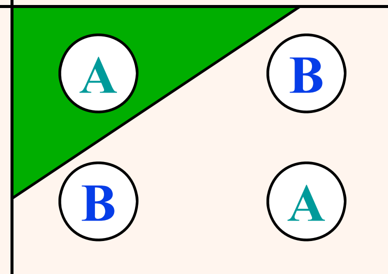
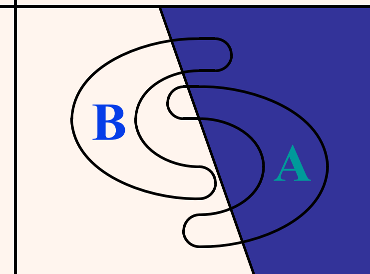
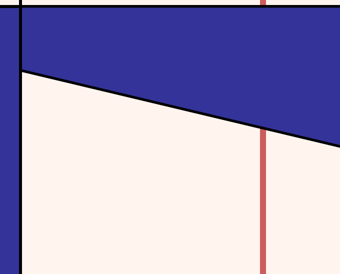
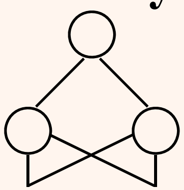
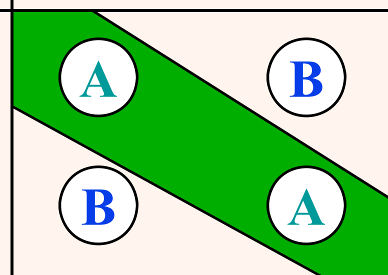
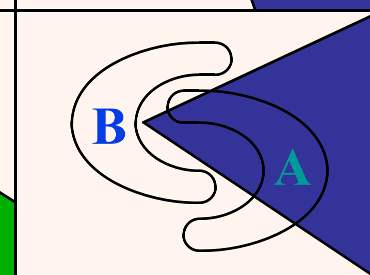
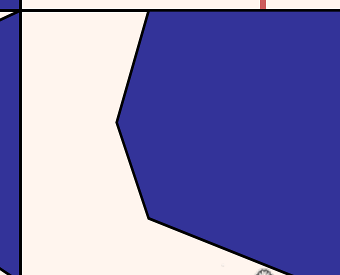
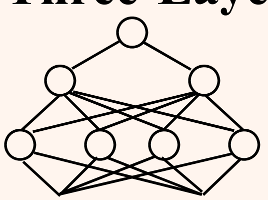
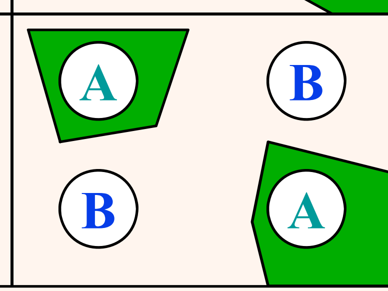
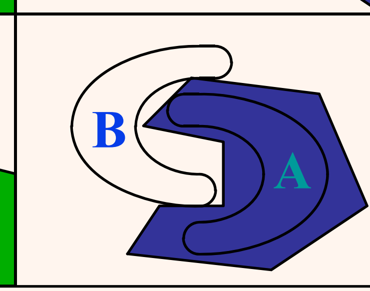
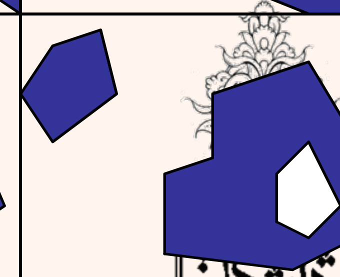
- ورودی‌ها به صورت مستقیم به خروجی متصل نیستند.
- هر واحد از لایه‌ی ۱ به تمامی واحدهای لایه‌ی ۲ متصل است. (وزن صفر مجاز است)
- تعداد واحدهای مخفی مشخص است.
- تمام اتصالات رو به جلو است.
- تابع انگیزش تابعی غیرخطی است.

Feed Forward



شبکه عصبی چند لایه

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

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 	Abitrary (Complexity Limited by No. of Nodes)			

یادگیری

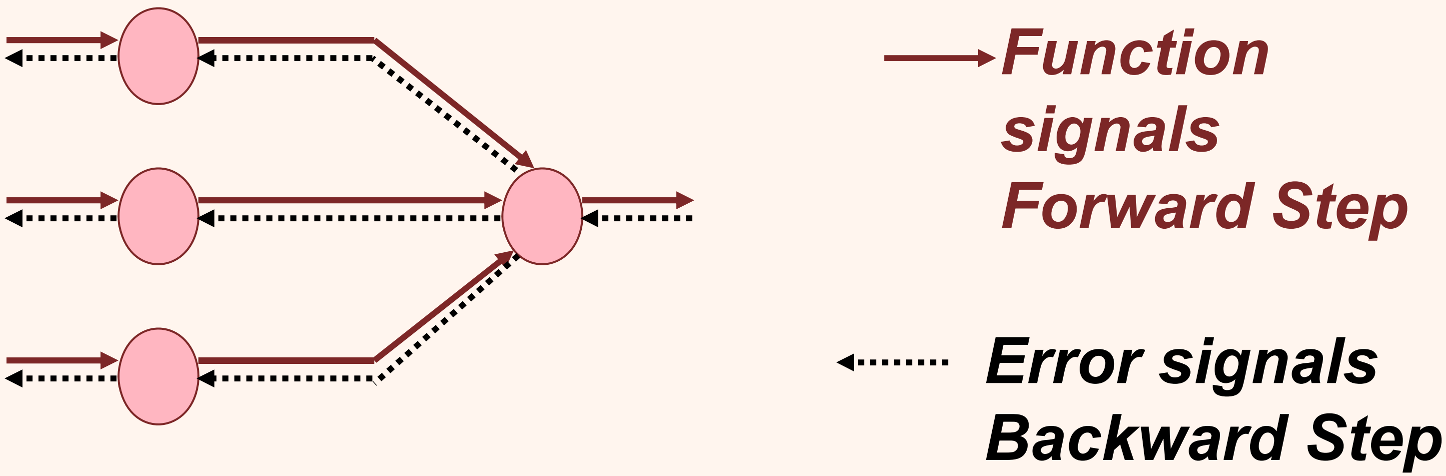
- برای آموزش این شبکه‌ها از الگوریتم Back Propagation استفاده می‌شود.
- برای آموزش این دست شبکه‌ها به طریقه‌ی زیر عمل می‌شود:
  - رو به جلو (Forward)
    - بردار ورودی به شبکه اعمال شده و خروجی واقعی محاسبه می‌شود.
  - رو به عقب (Backward)
    - خطا (خروجی واقعی – خروجی مطلوب) محاسبه شده و بر حسب تابع معیار، سیگنالی متناسب با خطا تولید می‌شود. این سیگنال لایه لایه حرکت کرده و وزن‌ها را تا لایه‌ی ورودی اصلاح می‌نماید.

پس از اصلاح وزن‌ها می‌گوییم یک iteration صورت گرفته است

دانشگاه شهید بهشتی

عملکرد شبکه

- فرض شبکه
  - m لایه بدون در نظر گرفتن لایه‌ی ورودی



وزن‌ها به گونه‌ای اصلاح می‌شوند که میانگین مجموع مربعات خطا کمینه گردد.

میزان خطا

sequential mode

- میزان خطا و میانگین مربعات خطا برای نرون زام خروجی در تکرار nام(به ازای ورودی n–ام) به شیوه‌ی زیر محاسبه می‌شود:

$$\boxed{e_j(n) = d_j(n) - y_j(n)}$$

خطای خروجی نرون jام (گره‌ی خروجی)

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

instantaneous error energy

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

averaged squared error energy

هدف آموزش کمینه نمودن E_{AV}

آموزش شبکه‌های چندلایه

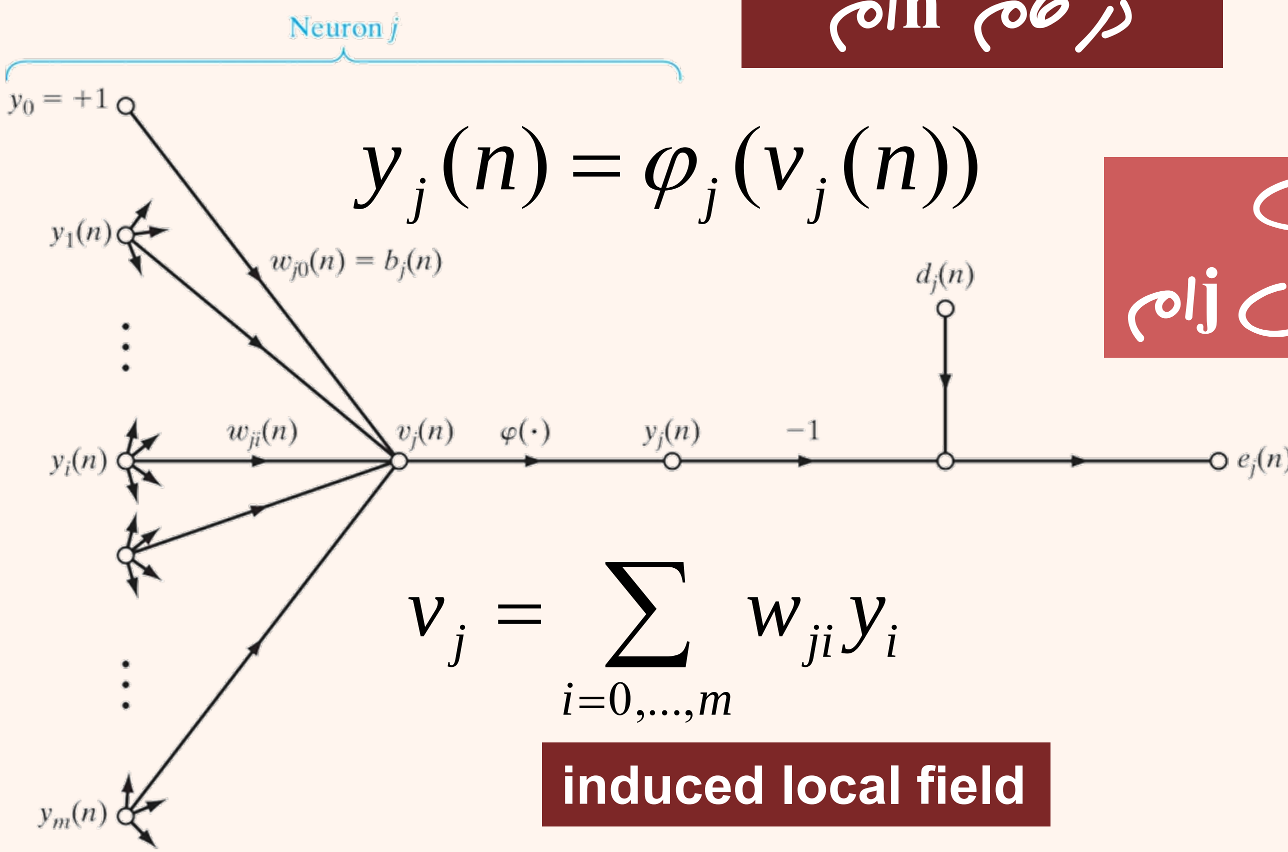
- به روز نمودن وزن‌ها همانند LMS است:

خروجی نرون j ام در گام n ام

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

خطای خروجی نرون j ام

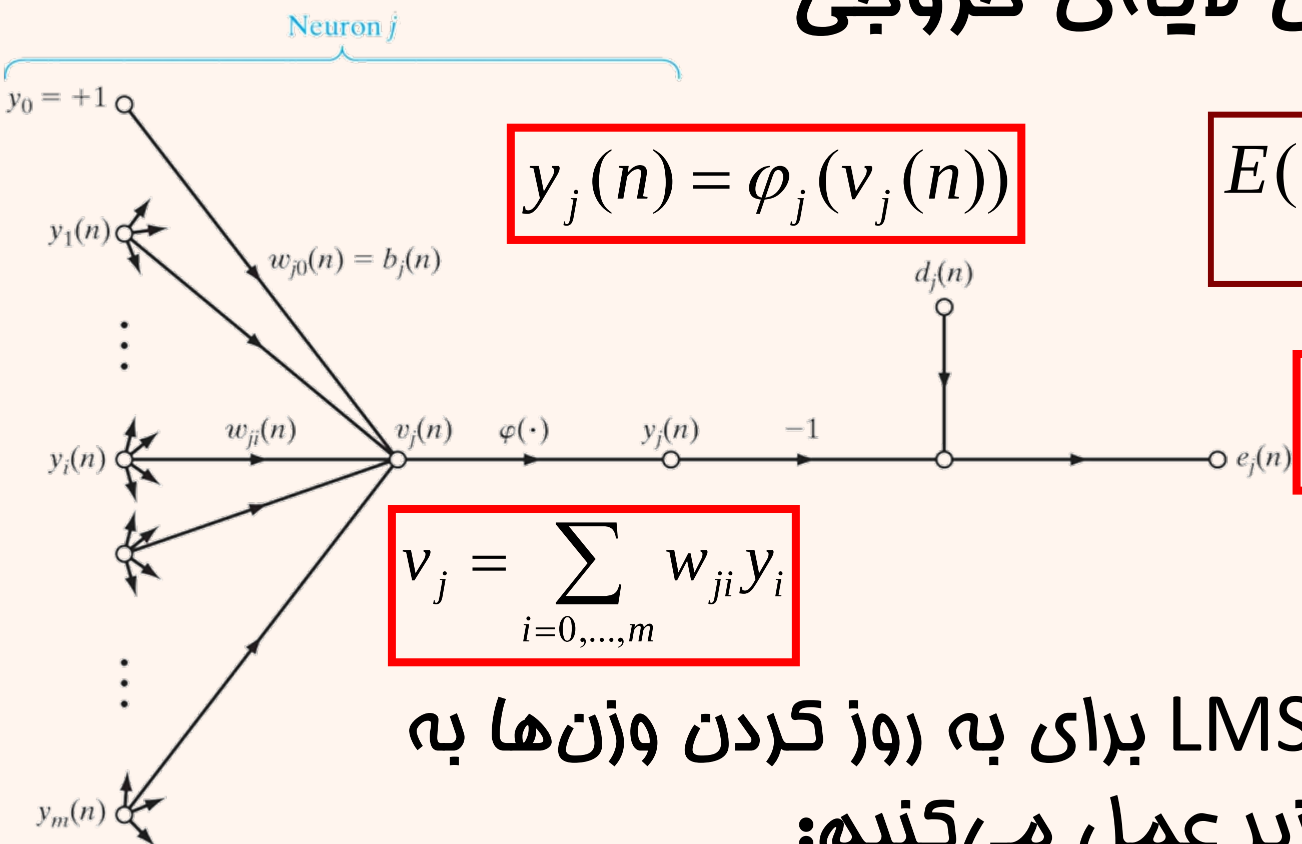
e_j



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

induced local field

آموزش لایه‌ی خروجی



$$\boxed{y_j(n) = \varphi_j(v_j(n))}$$

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

$$\boxed{e_j}$$

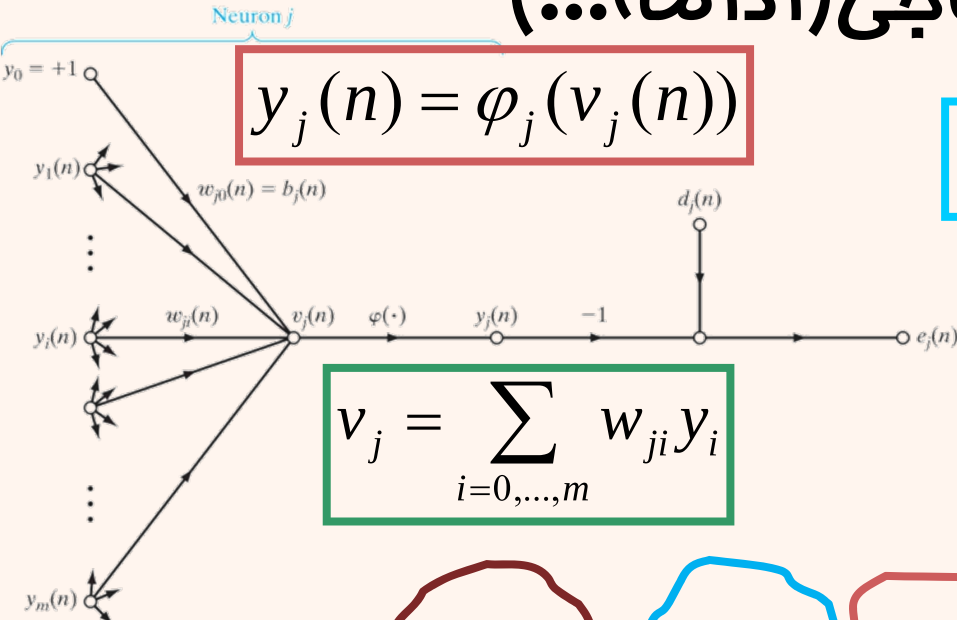
$$\boxed{v_j = \sum_{i=0,\dots,m} w_{ji} y_i}$$

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

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

sensitivity factor

آموزش لایه‌ی خروجی(ادامه...)



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

$$e_i(n) = d_i - y_i(n)$$

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

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

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

$$e_j(n) \qquad -1 \qquad \phi_j'(v_j(n)) \qquad y_i(n)$$

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

آموزش لایه‌ی 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

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

$$\Delta w_{ji}(n) = -\eta \underbrace{e_j(n)\varphi_j'(v_j(n))}_{\delta_j(n)} y_i(n)$$

گرادیان محلی

$\delta_j(n)$

گرادیان محلی برای یک نرون به
خطای ایجاد شده برای آن نرون و
مشتق تابع مرتبط آن بستگی دارد

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

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

آموزش لایه‌ی خروجی(ادامه...)

$$e_j = d_j - y_j$$

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

$$\delta_j(n)$$

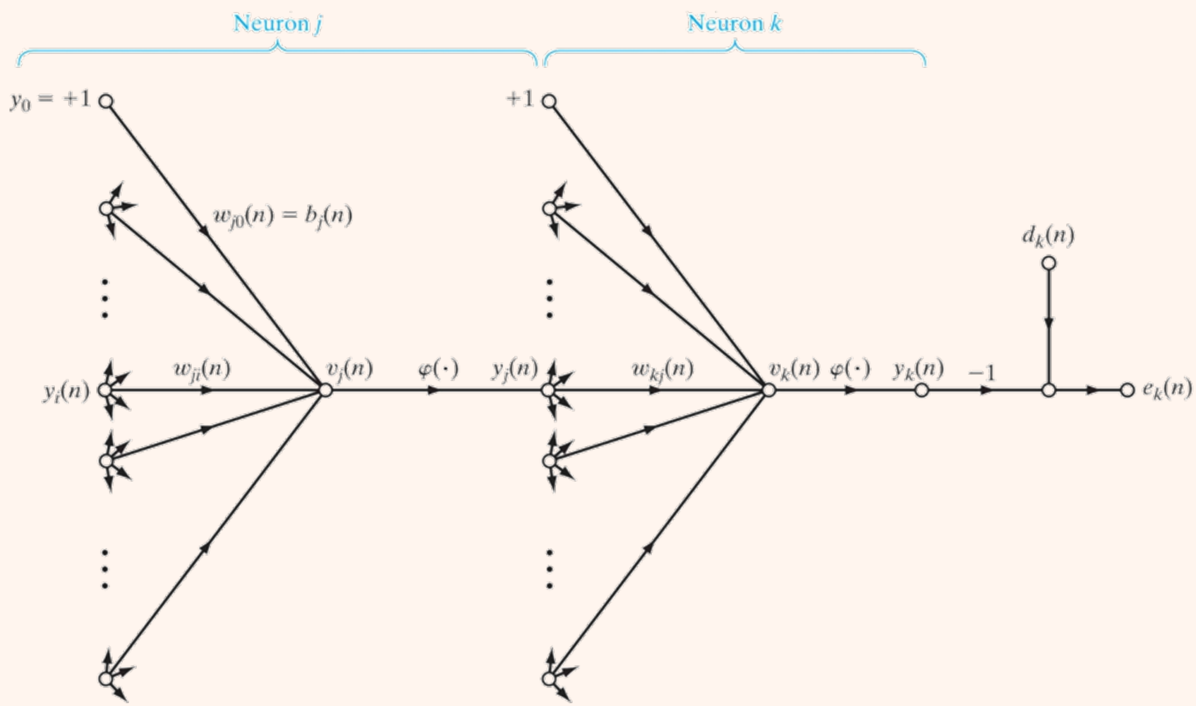
$$\delta_j = (d_j - y_j)\varphi'(v_j)$$

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

در ادامه به شیوه‌ی آموزش لایه‌ی مخفی خواهیم پرداخت

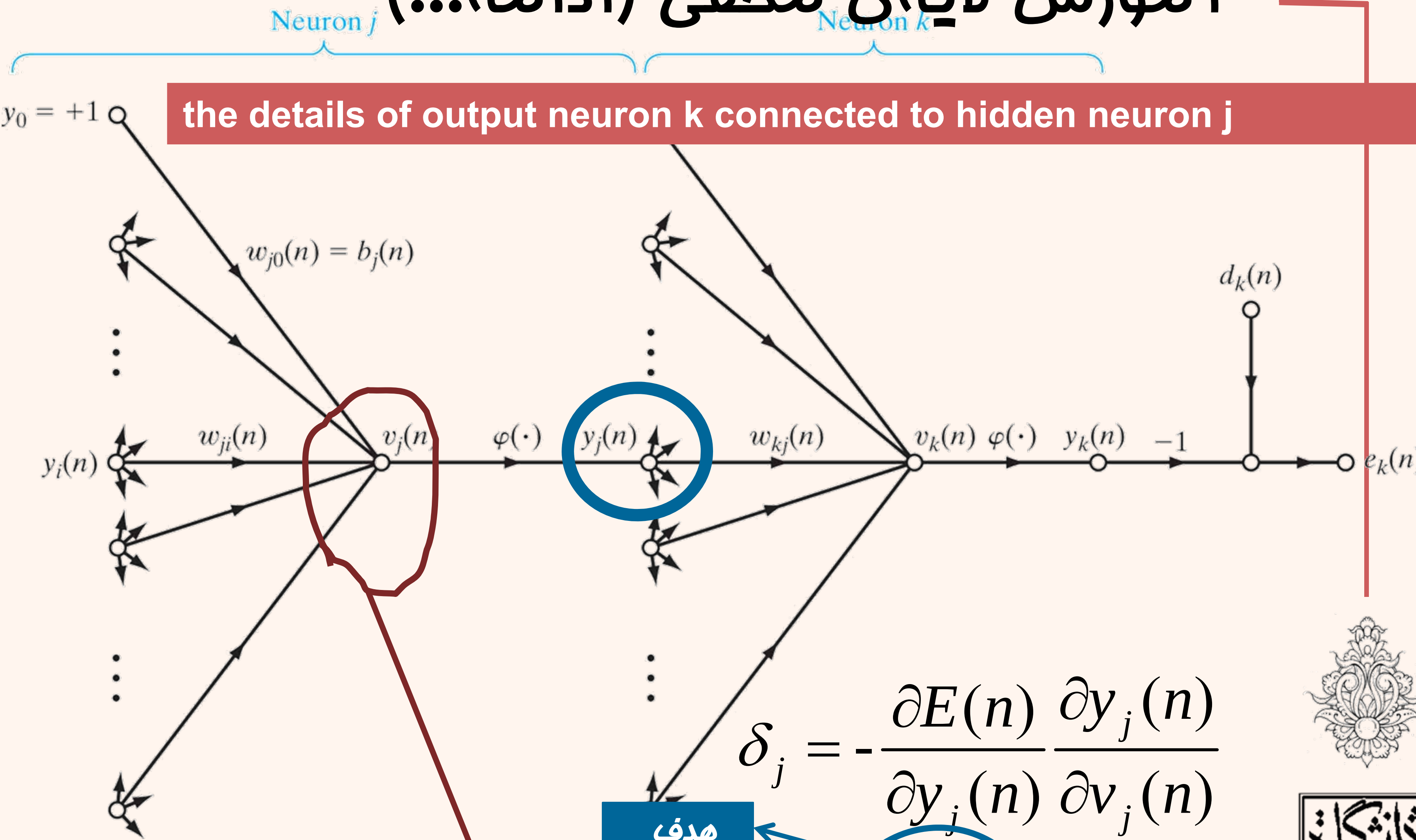
آموزش لایه‌ی مخفی

- اگر نرون j را نرونی از لایه‌ی مخفی در نظر بگیریم برای محاسبه‌ی خطا:
  - برای محاسبه‌ی گرادیان محلی نرون مورد نظر می‌باید تمامی گرادیان‌های محلی نرون‌هایی که با نرون مورد نظر در ارتباط هستند را لحاظ نماییم.



آموزش لایه‌ی مخفی (ادامه...)

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

نرون j متعلق به لایه‌ی مخفی است

آموزش لایه‌ی مخفی (ادامه...)

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

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

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

نرون k یک node خروجی است

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

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

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

$$\boxed{w_{kj}(n)}$$

$$\boxed{-\varphi_k'(v_k(n))}$$

$$e_k(n) = d_k(n) - y_k(n)$$
$$= d_k(n) - \varphi_k(v_k(n))$$

شبکه عصبی

آموزش لایه‌ی مخفی (ادامه...)

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

$$w_{kj}(n)$$

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

$$\frac{\partial E(n)}{\partial y_j(n)} = -\sum_k \underbrace{e_k(n)\varphi_k'(v_k(n))}_{\delta_k(n)} w_{kj}(n)$$

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

$$\boxed{\delta_j = \varphi_j'(v_j(n)) \sum_k \delta_k(n) w_{kj}(n)}$$

نرون j متعلق به لایه‌ی مخفی است

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

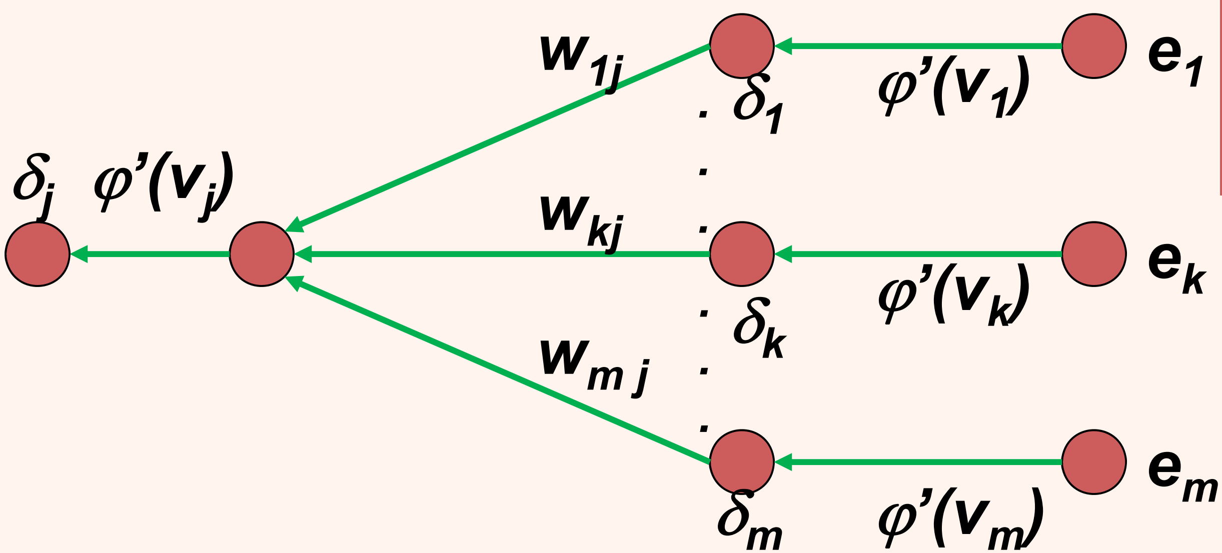
هدف

آموزش لایه‌ی مخفی (

Local Gradient of hidden neuron j

Iteration شماره‌ی n

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



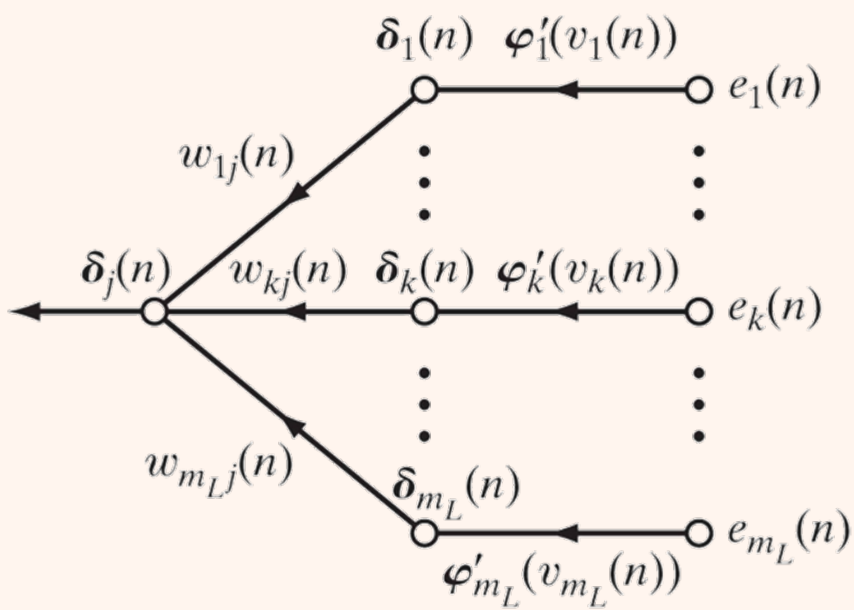
Signal-flow graph of back-propagation error signals to neuron j

آموزش لایه‌ی مخفی (ادامه...)

$$\boxed{\delta_j = \varphi_j'(v_j(n))\sum_k \delta_k(n) w_{kj}(n)}$$

- به صورت کلی خواهیم داشت:

$$\begin{pmatrix} \text{Weight} \\ \text{correction} \\ \Delta w_{ji}(n) \end{pmatrix} = \begin{pmatrix} \text{learning-} \\ \text{rate parameter} \\ \eta \end{pmatrix} \cdot \begin{pmatrix} \text{local} \\ \text{gradient} \\ \delta_j(n) \end{pmatrix} \cdot \begin{pmatrix} \text{input signal} \\ \text{of neuron } j \\ y_i(n) \end{pmatrix}$$



آموزش لایه‌ی مخفی (ادامه...)

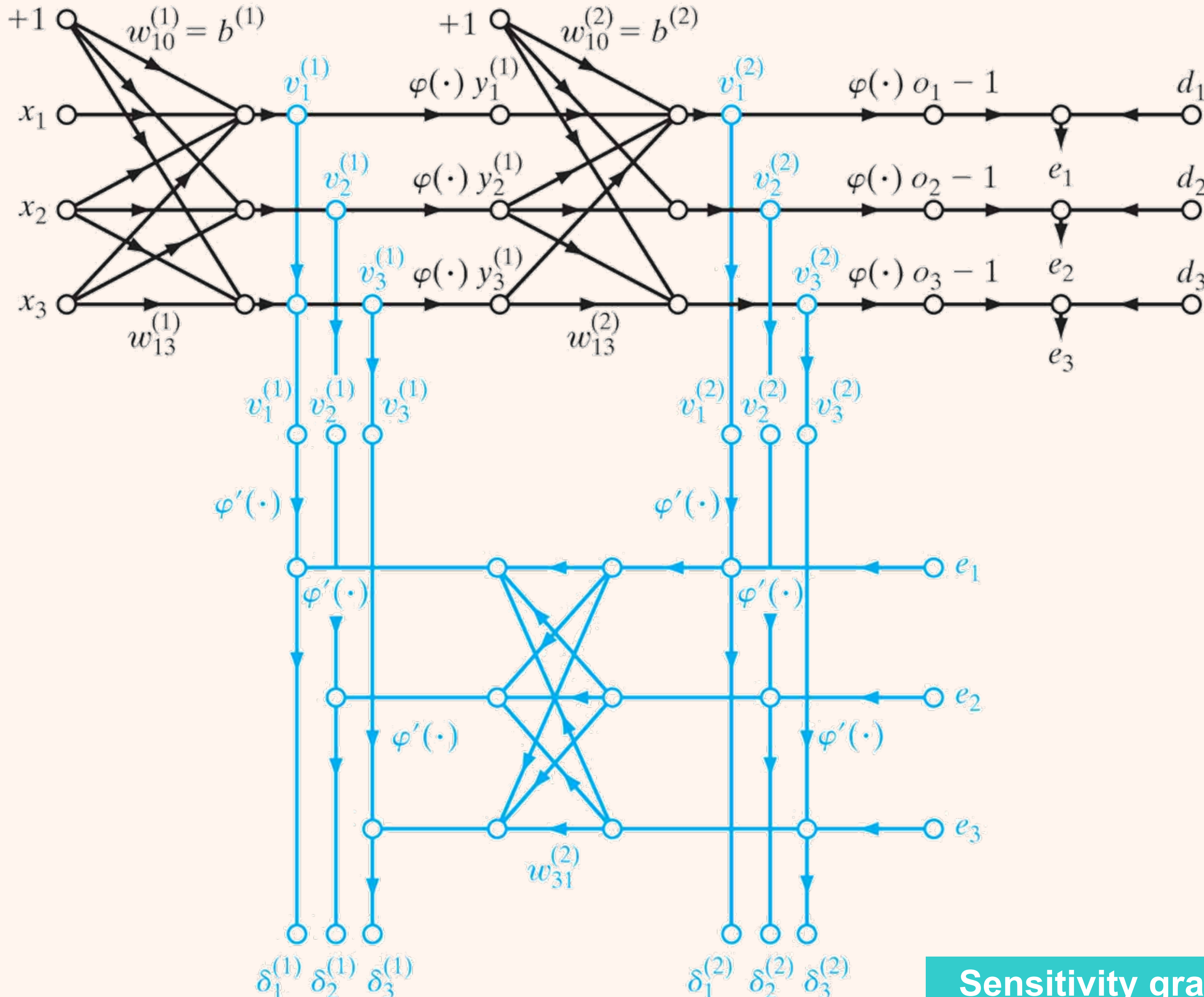
- **Delta rule** $\Delta w_{ji} = \eta \delta_j y_i$

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

C: Set of neurons in the layer following the one containing ***j***

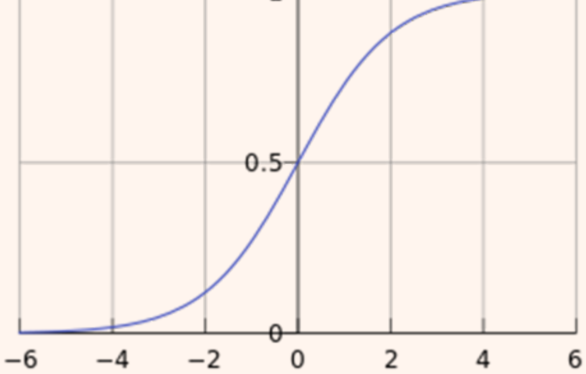
آموزش لایه‌ی مخفی (ادامه...)

دسته‌بندی



Sensitivity graph

تابع انگیزش



Sigmoid function

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

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

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

$$\varphi_j'(v_j(n)) = ay_j(n) \times \left[1 - y_j(n)\right]$$

تابع انگیزش (ادامه...)

- اگر نرون در لایه‌ی خروجی باشد:

$$\delta_j(n)=e_j(n)\varphi_j'(v_j(n))$$

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

- و اگر در لایه‌ی مخفی باشد:

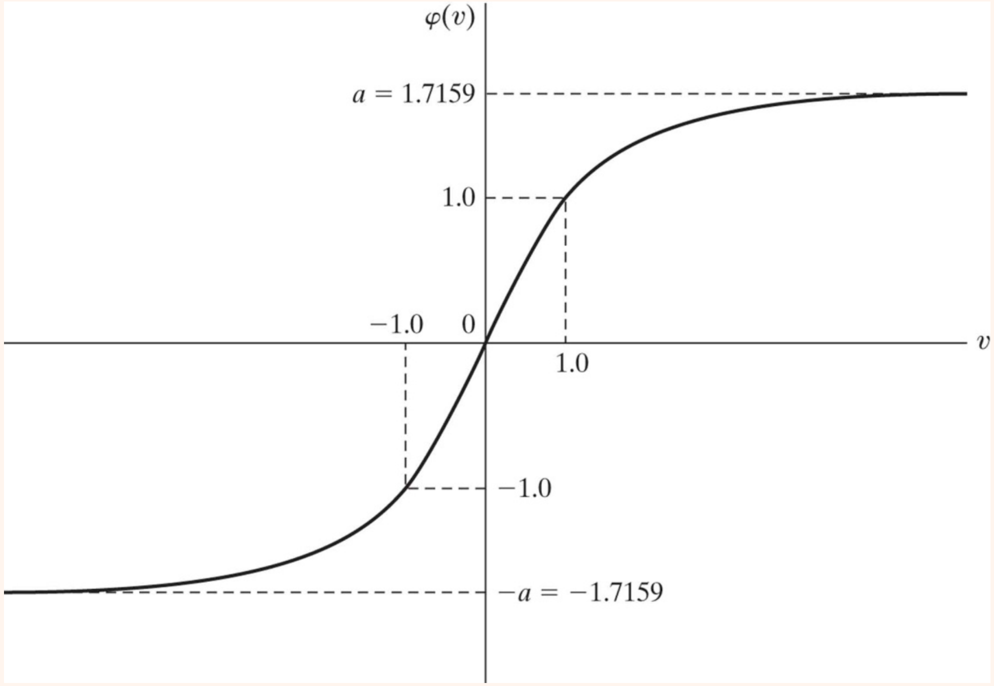
$$\delta_{\mathrm{j}}=ay_j(n)\times\left[1-y_j(n)\right]\sum_k\delta_k(n)w_{kj}(n)$$

بدین ترتیب گرادیان محلی بدون نیاز به استفاده از تابع انگیزش قابل محاسبه می‌باشد.

تابع انگیزش(ادامه...)

$$\varphi_j(v_j(n)) = a\tanh\left(bv_j(n)\right) \qquad (a,b) > 0$$

$$\varphi_j'(v_j(n)) = \frac{b}{a}\left[a - y_j(n)\right]\left[a + y_j(n)\right]$$



α= 1.7159 and b = 2/3

انواع شیوه‌های آموزش

Sequential Mode

online **pattern or stochastic mode**

- شیوه‌ی ترتیبی:
  - در این شیوه نمونه‌ها تک‌تک برای اصلاح وزن‌ها به کار می‌روند.

یک دوره‌ی کامل ارائه‌ی نمونه‌های آموزشی در فرآیند آموزش را **epoch** می‌نامند.

- شیوه‌ی دسته‌ای: **Batch Mode**
  - در این شیوه تمام نمونه‌های آموزشی اعمال شده، سپس اصلاح وزن‌ها صورت می‌پذیرد.

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

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

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

انواع شیوه‌های آموزش(ادامه...)

- شیوه‌ی ترتیبی به حافظه‌ی 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.
- وقتی که داده‌های تکراری داشته باشیم، روش ترتیبی به صورت مؤثرتری از داده‌های تکراری استفاده می‌کند.
- 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.
- استفاده از پردازش موازی در شیوه‌ی دسته‌ای به مراتب ساده‌است.

انواع شیوه‌های آموزش(ادامه...)

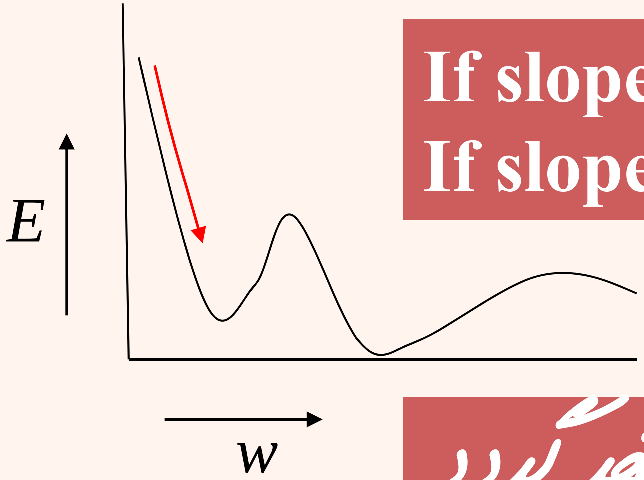
علیرغم، معایب روش ترتیبی، این روش در عمل ترجیح داده می‌شود، از این جهت که پیاده‌سازی آن ساده‌تر است
برای مسائل دشوار و بزرگ راه حل موثری است

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

معیارهای توقف آموزش

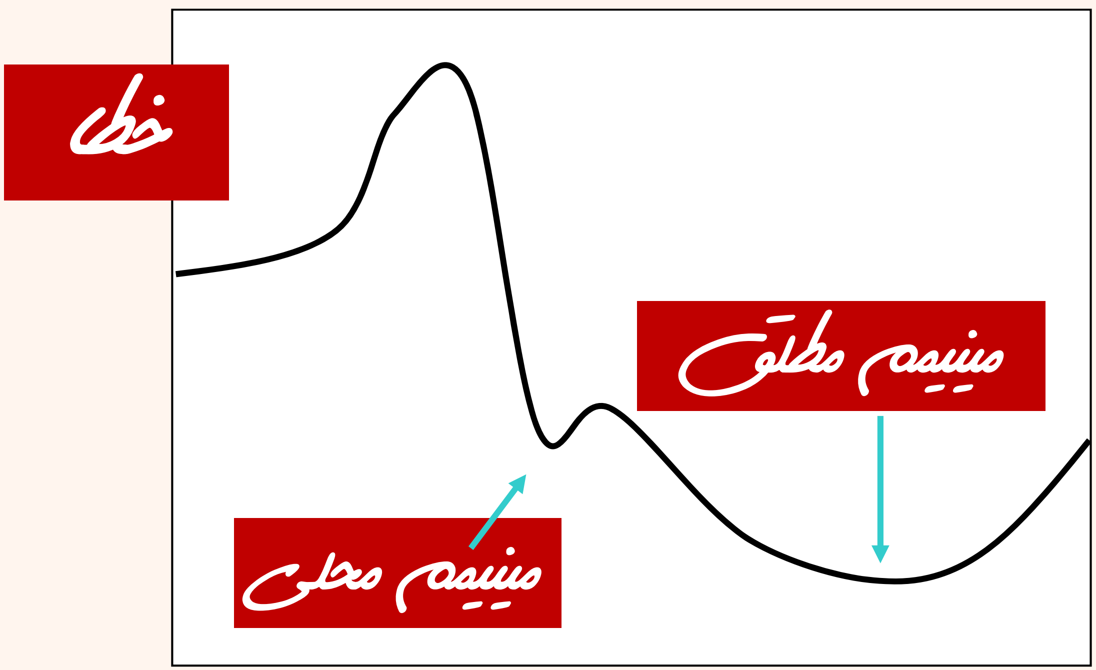
- در عمل، به راحتی نمی‌توان مشخص نمود که آیا فرآیند آموزش به نقطه‌ی مورد نظر رسیده است. اما ضوابطی برای پایان دادن به آموزش مطرح شده است.
  - آموزش تا جایی پیش رود که اندازه‌ی بردار گرادیان از یک حد آستانه کم‌تر شود.
    - عیب این روش این است که فرآیند آموزش ممکن است طولانی شود.
  - آموزش زمانی متوقف می‌شود که اختلاف خطای میانگین به دست آمده در دو دوره(epoch) متوالی به اندازه‌ی کافی کوچک باشد.
  - 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.

Gradient Descent



If slope is negative → increase w
If slope is positive → decrease w

مینیمم محلی جایی است که مشتق صفر گردد



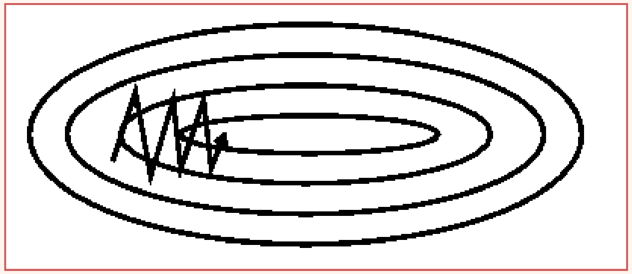
تأثیر نرخ آموزش

- چنان‌چه پیش از این مطرح وشد، نرخ آموزش بالا موجب ناپایداری می‌شود، در حالی که نرخ آموزش پایین فرآیند آموزش را طولانی خواهد کرد.
- همچنین می‌توان نرخ آموزش را برای وزن‌های مختلف متغیر در نظر گرفت.

Connection dependent

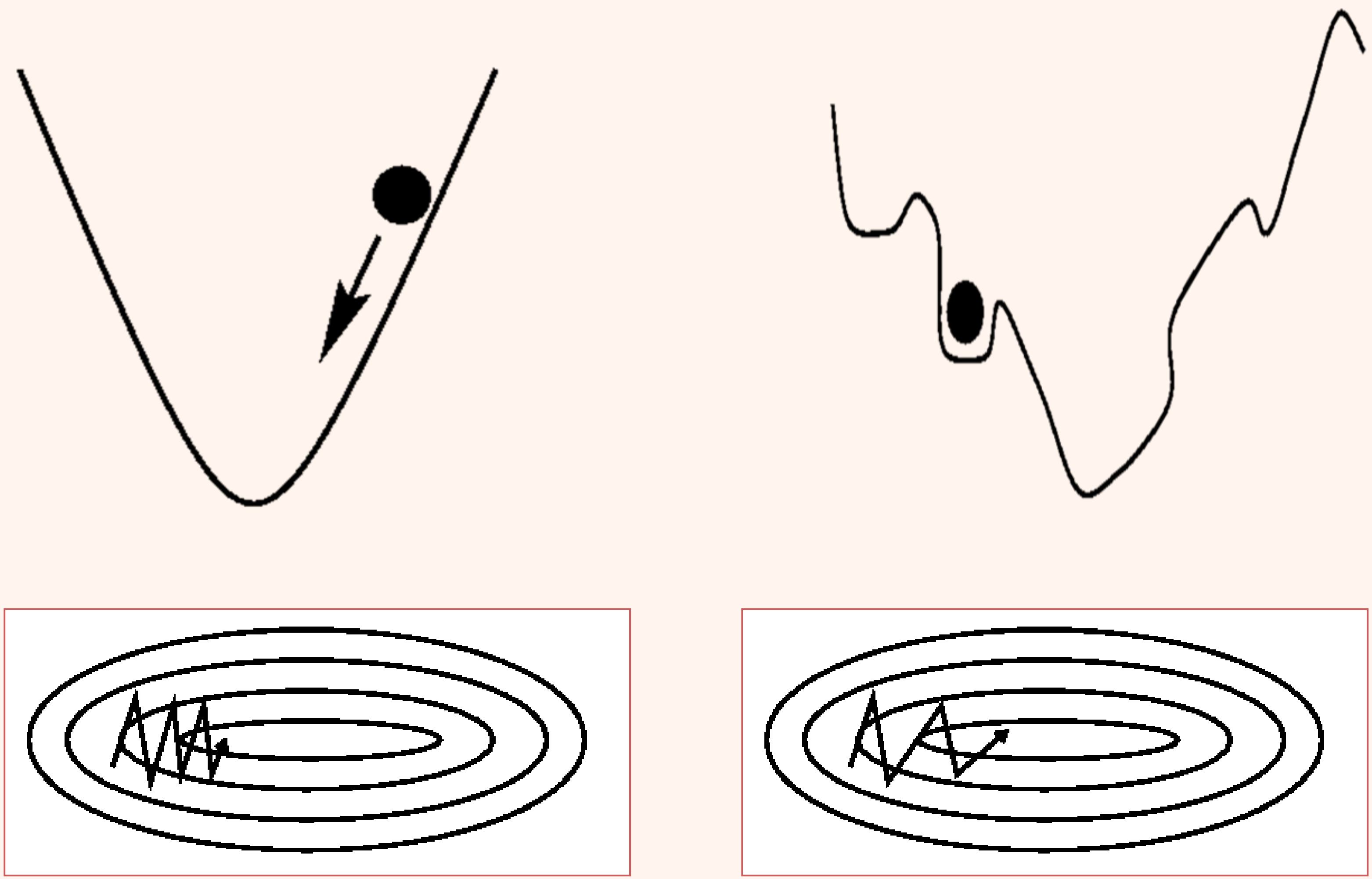
نرخ آموزش کوچک: همگرایی کند است ولی روند حرکت بدون تغییرات زیاد

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



Momentum and Learning Rate Adaptation

Local Minima



http://www.willamette.edu/~gorr/classes/cs449/momrate.html

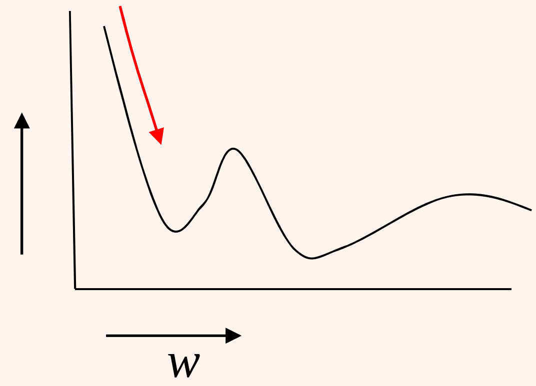
روش استفاده از momentum

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

- داشتیم:

generalized delta rule

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



Momentum ثابت

$$\boxed{0 \le \alpha < 1}$$

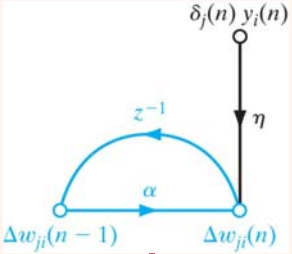
تغییرات وزن در مرحله‌ی پیشین

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

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

به روز رسانی وزن ها

روش استفاده از momentum



تغییرات وزن در مرحله‌ی پیشین

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

- اگر ثابت ممنتوم صفر باشد به‌روز 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 تغییرات هم‌جهت باشند، در نظر گرفتن momentum باعث تسریع آموزش می‌شود.

accelerating effect

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

stabilizing effect

- واضح است که ثابت کم‌تر از صفر و بیشتر از یک ناکارامد است.
- با تغییر میزان نرخ آموزش می‌توان پاسخی بهینه دریافت کرد.

روش استفاده از momentum

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

برای iteration اول بررسی می کنیم:

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

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

$$\Delta w_{ji}^{l}(2) = \underbrace{\alpha^2 \eta \delta_j^l(0).y_i^{l-1}(0) + \alpha \eta \delta_j^l(1).y_i^{l-1}(1)}_{\text{Momentum تاثیر}} + \eta \delta_j^l(2).y_i^{l-1}(2)$$

تاثیر Momentum

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

نرخ آموزش

ثابت Momentum

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

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

نرخ آموزش

ثابت Momentum

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

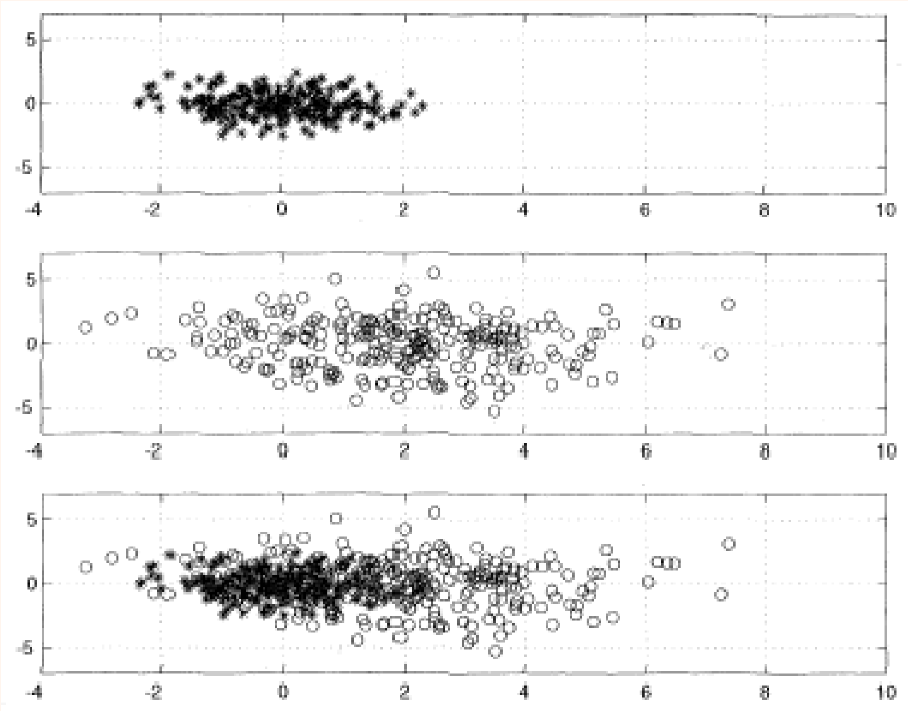
- میانگین زمان یادگیری برای یک شبکه بدون/با استفاده از Momentum

momentum	Training time
۰	۲۱۷
۰.۹	۹۵

http://www.willamette.edu/~gorr/classes/cs449/momrate.html

مثال

- هدف طراحی شبکه‌ای است که دو دسته داده زیر را از هم جدا کند.



مثال

Class C_1: $$f_{\mathbf{x}}(\mathbf{x}|C_1)=\frac{1}{2\pi\sigma_1^2}\exp\left(-\frac{1}{2\sigma_1^2}\|\mathbf{x}-\boldsymbol{\mu}_1\|^2\right)$$

$$\boldsymbol{\mu}_1 = \text{mean vector} = [0,0]^T$$

$$\sigma_1^2 = \text{variance} = 1$$

Class C_2: $$f_{\mathbf{x}}(\mathbf{x}|C_2)=\frac{1}{2\pi\sigma_2^2}\exp\left(-\frac{1}{2\sigma_2^2}\|\mathbf{x}-\boldsymbol{\mu}_2\|^2\right)$$

$$\boldsymbol{\mu}_2 = \text{mean vector} = [2,0]^T$$

$$\sigma_2^2 = \text{variance} = 1$$

Likelihood ratio

$$\Lambda(\mathbf{x})=\frac{f_{\mathbf{x}}(\mathbf{x}|C_1)}{f_{\mathbf{x}}(\mathbf{x}|C_2)}\lessgtr 1$$

ادامه ...

$$\Lambda(\mathbf{x}) = \frac{\sigma_2^2}{\sigma_1^2} \exp\left(-\frac{1}{2\sigma_1^2}\|\mathbf{x}-\boldsymbol{\mu}_1\|^2 + \frac{1}{2\sigma_2^2}\|\mathbf{x}-\boldsymbol{\mu}_2\|^2 \right)$$

$$\frac{\sigma_2^2}{\sigma_1^2} \exp\left(-\frac{1}{2\sigma_1^2}\|\mathbf{x}-\boldsymbol{\mu}_1\|^2 + \frac{1}{2\sigma_2^2}\|\mathbf{x}-\boldsymbol{\mu}_2\|^2 \right) = 1$$

$$\frac{1}{2\sigma_2^2}\|\mathbf{x}-\boldsymbol{\mu}_2\|^2 \quad -\frac{1}{2\sigma_1^2}\|\mathbf{x}-\boldsymbol{\mu}_1\|^2 = 4\log\frac{\sigma_2^2}{\sigma_1^2}$$

$$\|\mathbf{x}-\mathbf{x}_c\|^2 = r^2$$

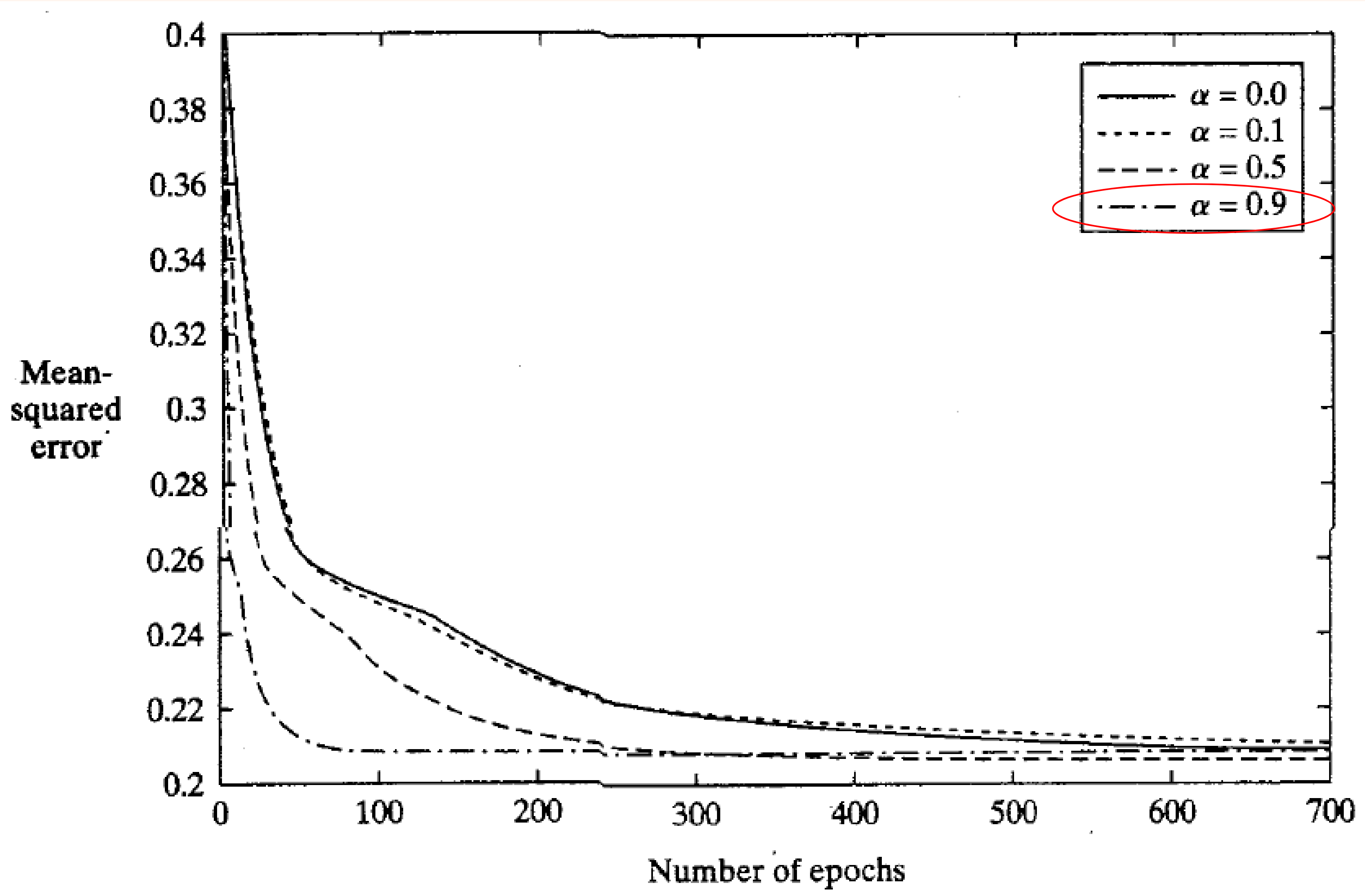
Optimum decision boundary

$$P_c = 0.8151$$

Probability of correct classification

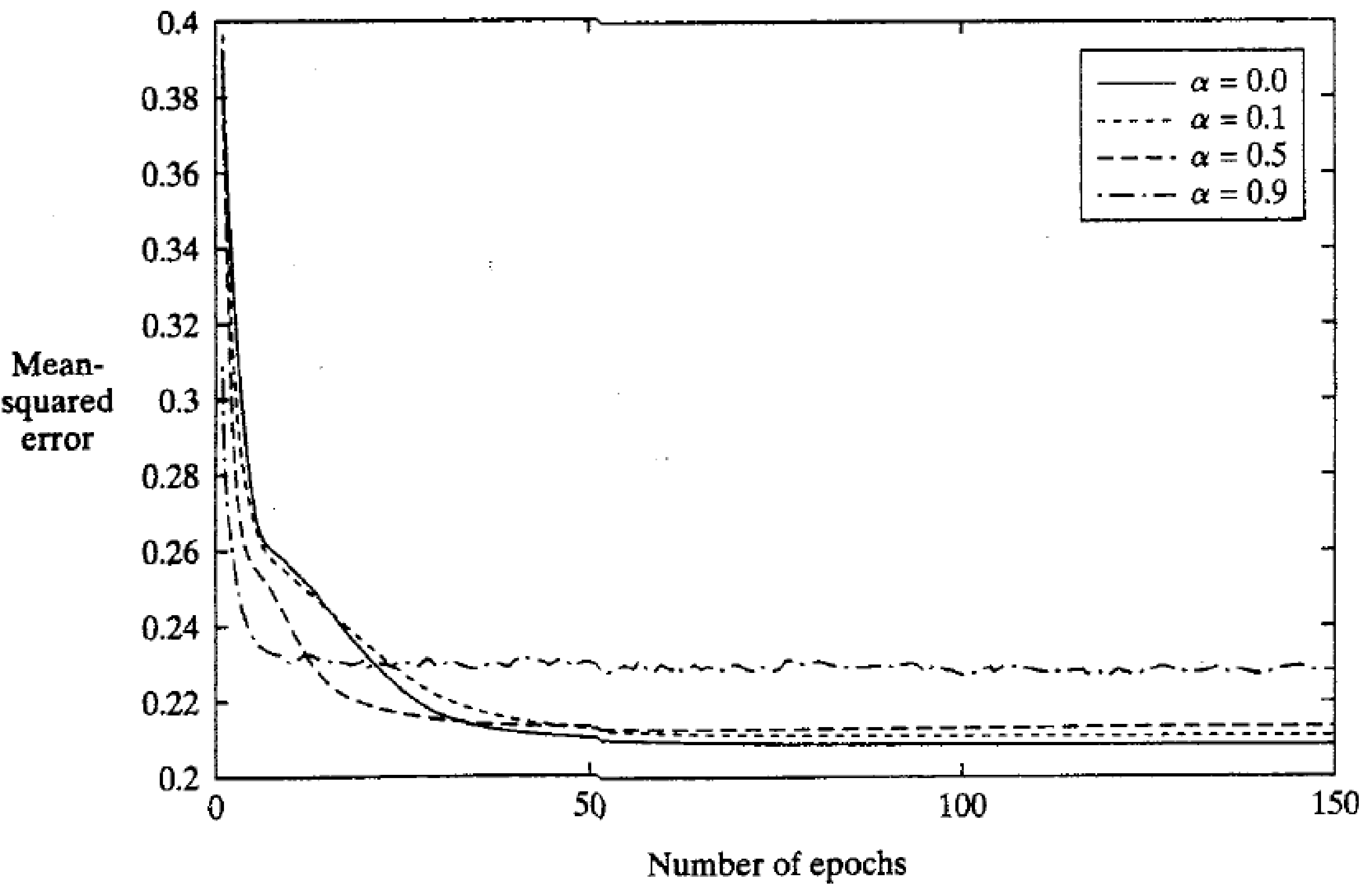
η=0.01

مثال



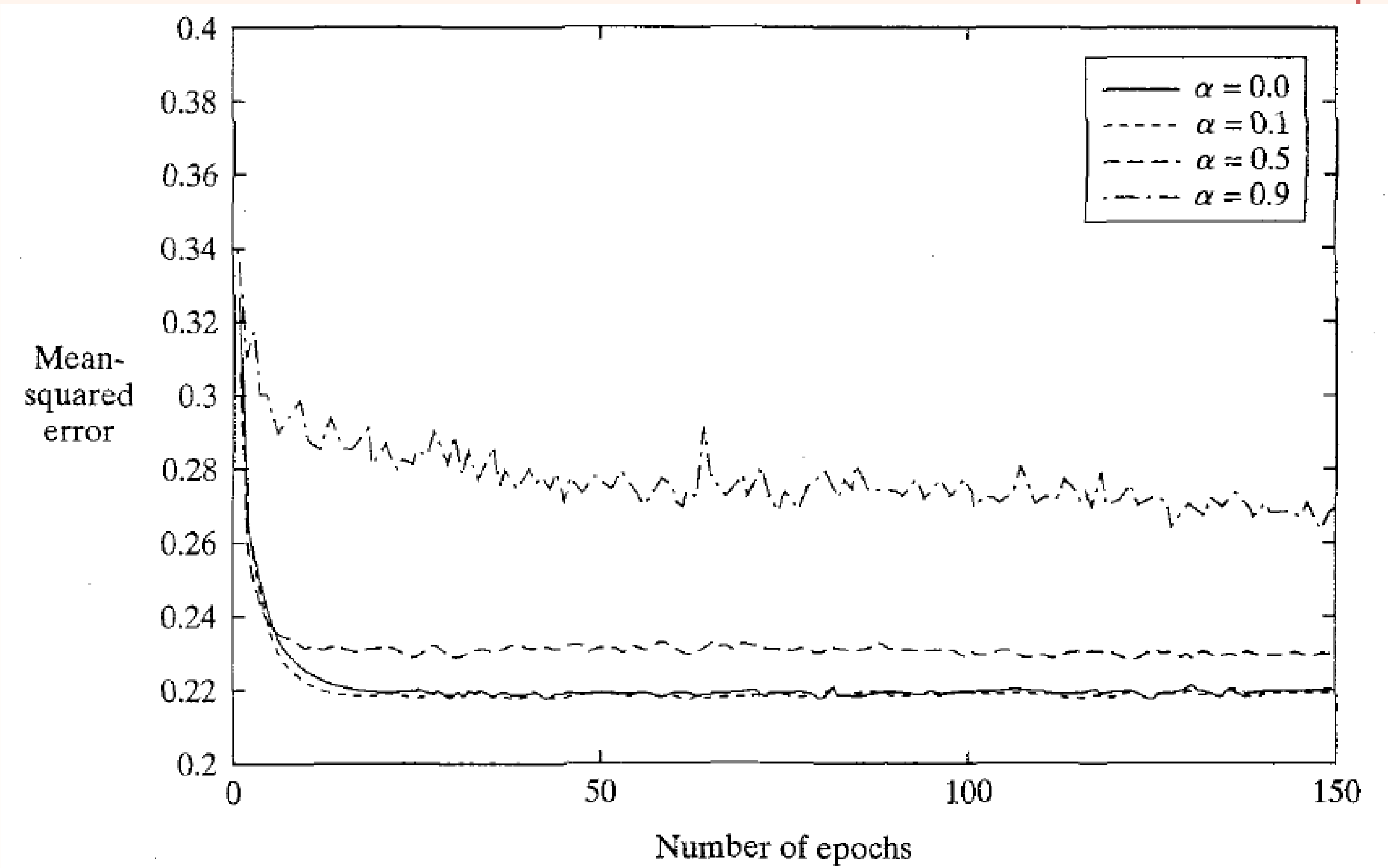
0.4
0.38
0.36
0.34
0.32
0.3
0.28
0.26
0.24
0.22
0.2
Mean-squared error
0
100
200
300
400
500
600
700
Number of epochs
α = 0.0
α = 0.1
α = 0.5
α = 0.9

$\eta = 0.1$



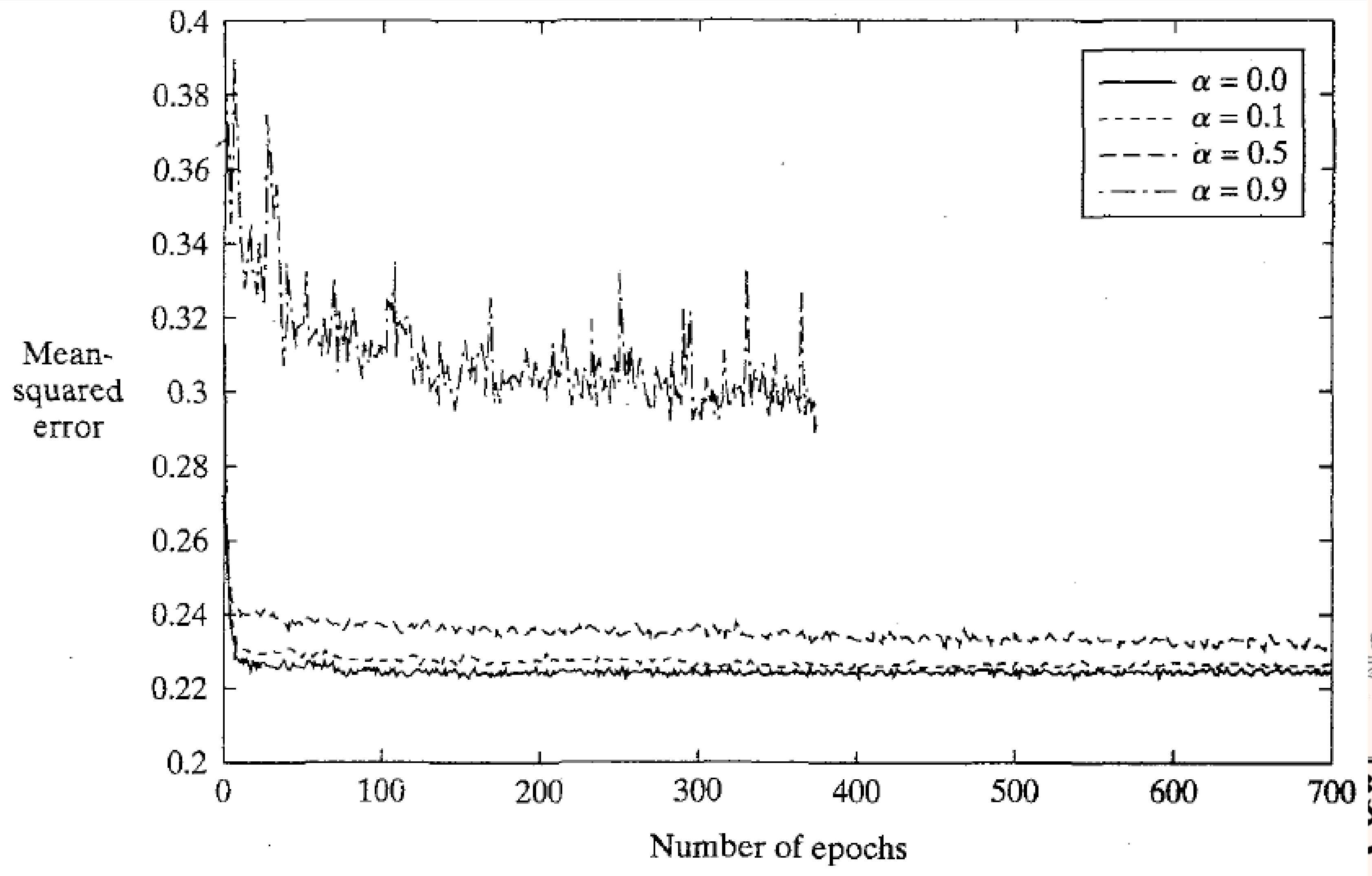
0.4
0.38
0.36
0.34
0.32
0.3
0.28
0.26
0.24
0.22
0.2
Mean-squared error
0
50
100
150
Number of epochs
α = 0.0
α = 0.1
α = 0.5
α = 0.9

η =0.5

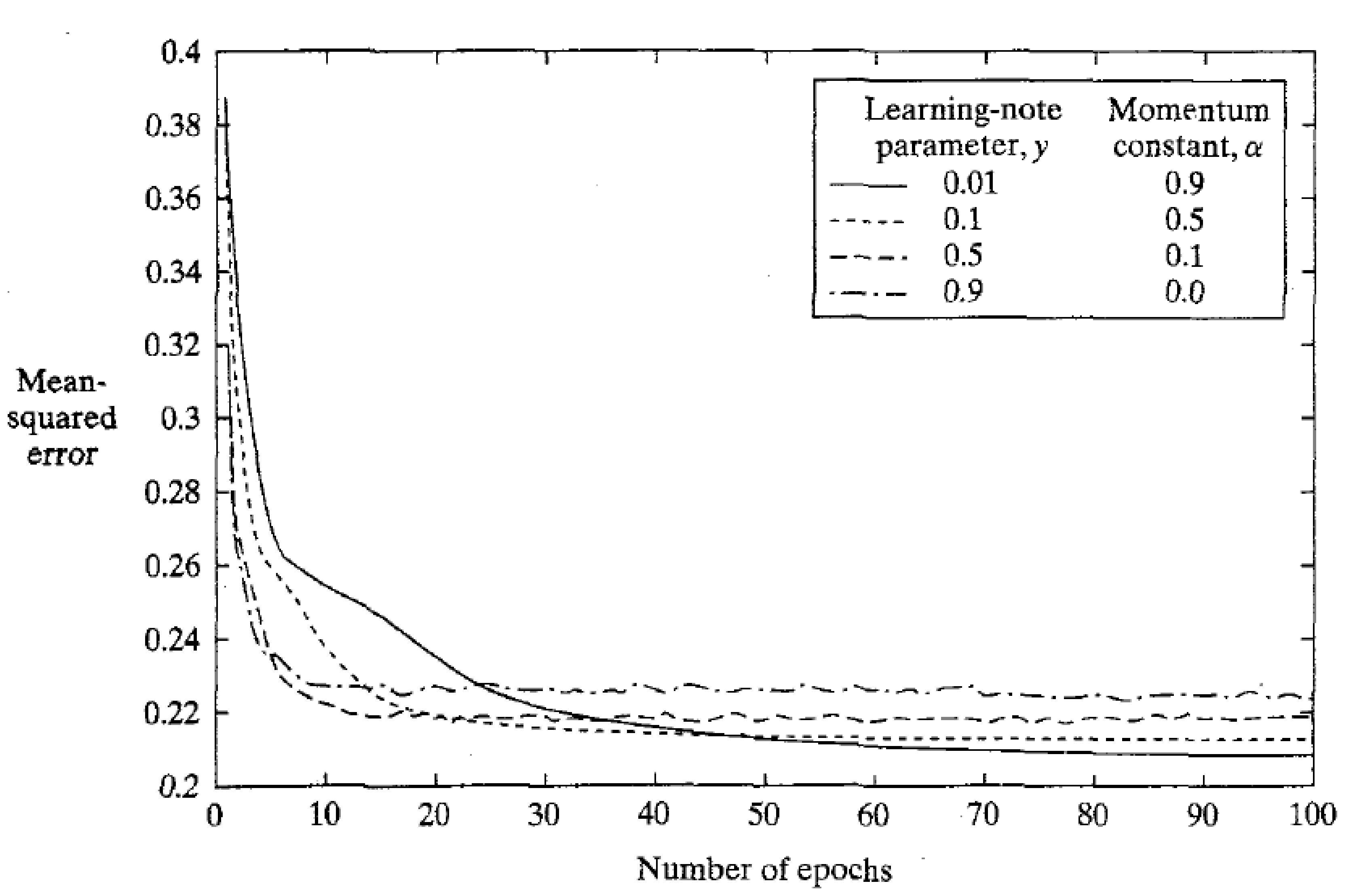


Mean-squared error
0.4
0.38
0.36
0.34
0.32
0.3
0.28
0.26
0.24
0.22
0.2
0
50
100
150
Number of epochs
α = 0.0
α = 0.1
α = 0.5
α = 0.9

η =0.9

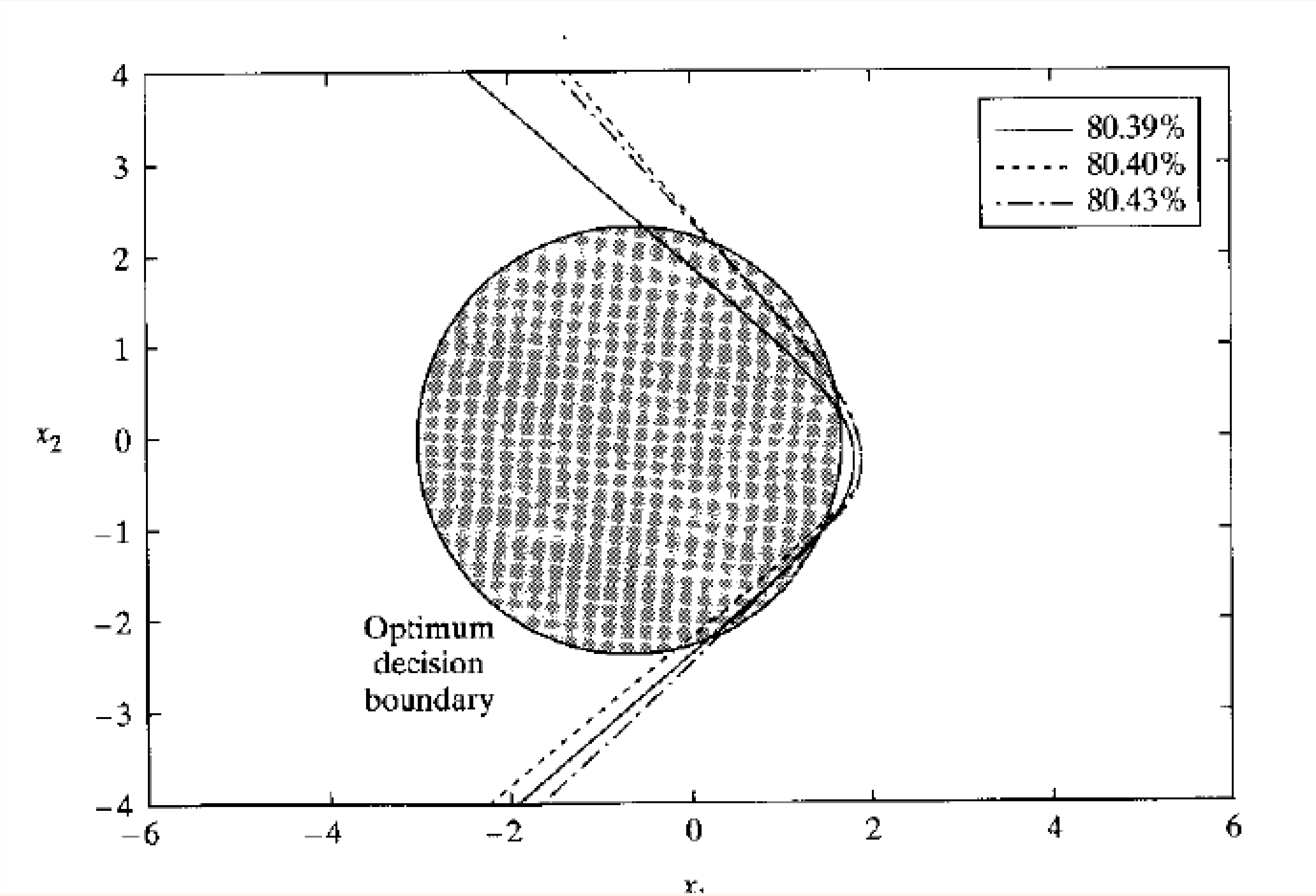


0.4
0.38
0.36
0.34
0.32
0.3
0.28
0.26
0.24
0.22
0.2
Mean-squared error
0
100
200
300
400
500
600
700
Number of epochs
$\alpha = 0.0$
$\alpha = 0.1$
$\alpha = 0.5$
$\alpha = 0.9$



Best learning curves selected from the four parts

مثال


80.39%
80.40%
80.43%
Optimum
decision
boundary
x_2
x_1
4
3
2
1
0
−1
−2
−3
−4
−6
−4
−2
0
2
4
6

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 η ->0 , α->1	*باعث افزایش سرعت همگرایی*
For η ->1, α->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 (وفقی)

Adaptive Learning Rate

Bold Driver

ضریب momentum را صفر می‌گذاریم

$$\text{if} \quad E(k+1) > (1+\xi)E(k) \longrightarrow w(k+1) = w(k)$$

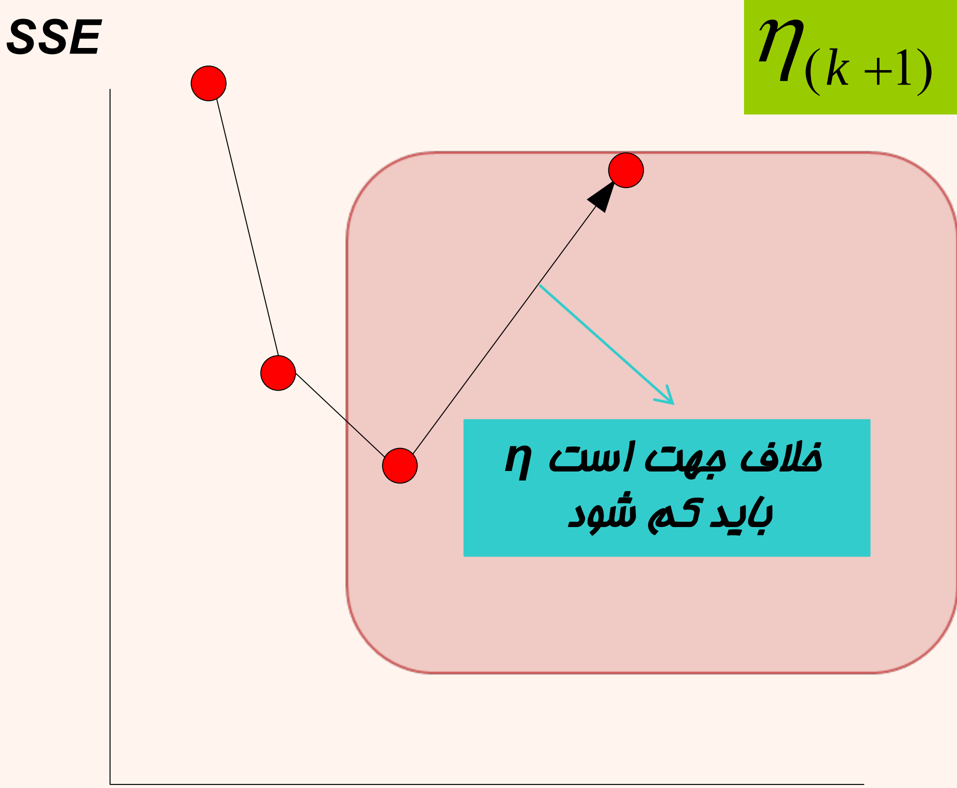
در این حالت جهت حرکت نامناسب است

$$0.01 < \xi < 0.05$$

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

$$0 < \rho < 1$$

معمولا ۱.۵



روش نرخ یادگیری متغیر (وفقی)

ضریب momentum به مقدار اصلی برمی‌گردد

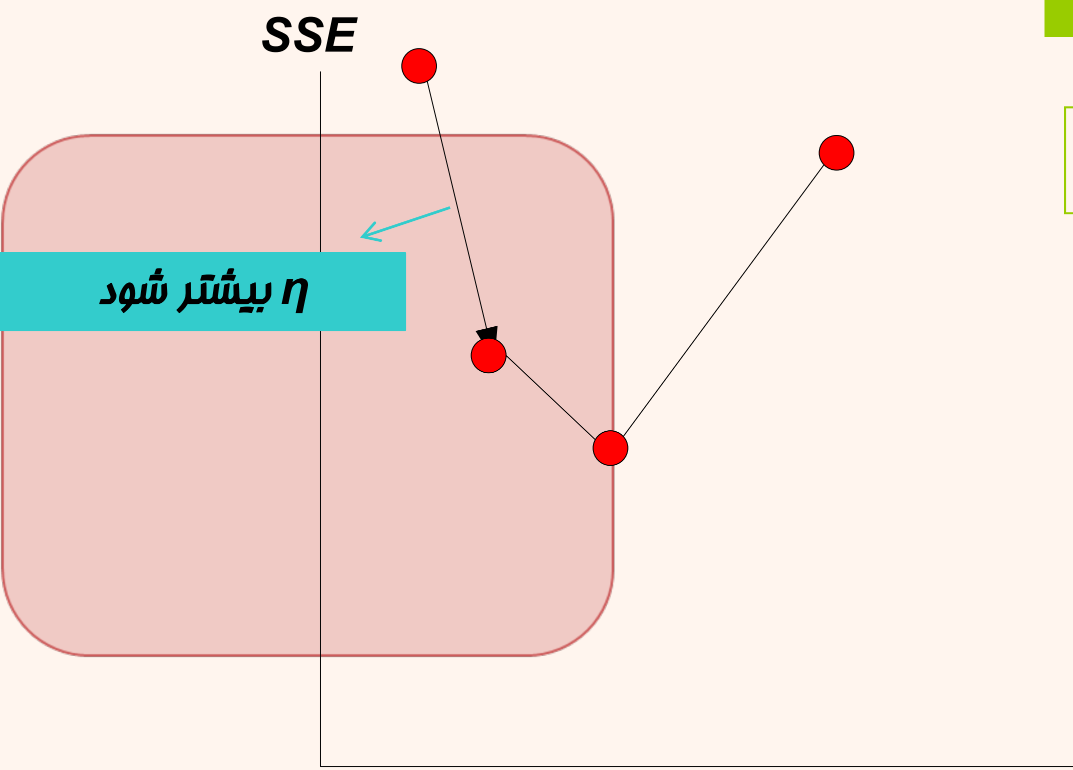
$$if \quad E(k+1) < E(k) \longrightarrow \quad w(k+1) = w(k) + \Delta w(k)$$

در این حالت جهت حرکت مناسب است

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

$$1 < d < 2$$

معمولا ۱.۰۱ تا ۱.۰۵



روش نرخ یادگیری متغیر (وفقی)

$$if \quad E(k+1) > (1+\beta)E(k)$$

چنانچه شرط قبلی برقرار نباشد

$$\beta < \xi$$

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

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

وزن های جدید را قبول کرده ولی η ثابت می‌ماند و α به مقدار اولیه تغییر داده می شود.

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

روش نرخ یادگیری متغیر

- در روش ترتیبی، استفاده از Bold driver منجر به واگرایی الگوریتم می‌شود، از این‌رو به جای در نظر گرفتن نرخ وفقی، نرخ آموزش به صورت نزولی در نظر گرفته می‌شود.

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

- بدین‌ترتیب، در گام‌های اولیه نرخ آموزش تقریبا ثابت است. بعد از رسیدن به محدوده‌ی مینیمم، نرخ آموزش کاهش می‌یابد.
  - در این حالت یک پارامتر آزاد به مجموعه‌ی پارامترها افزوده خواهد شد.

روش‌های بهبود کارایی

- غالبا 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 افزونگی در داده‌ها بالاست روش ترتیبی مناسب‌تر است.
  - مواردی که مجموعه‌ی آموزشی ثابت نیست، داده‌های جدید به مجموعه اضافه شود.

Y. LeCun, L. Bottou, G. Orr and K. Muller: Efficient BackProp, in Orr, G. and Muller K. (Eds), *Neural Networks: Tricks of the trade, Springer, 1998*

روش‌های بهبود کارایی(ادامه...)

Maximizing information content

- استفاده از حداکثر اطلاعات در آموزش:
  - مجموعه‌ی آموزشی نقش بسیار مهمی در همگرایی و تعمیم‌پذیری دارد.
    - بهتر است از نمونه‌هایی استفاده شود که بیشترین خطا در آموزش را ایجاد می‌کند.
    - نمونه‌هایی که با هم متفاوت هستند. این کار باعث می‌شود گستره‌ی وسیع‌تری از وزن‌ها بررسی شوند.
  - در بازشناسی الگو در شیوه‌ی ترتیبی، اعمال الگوها به صورت تصادفی موجب می‌شد تا داده‌های یک کلاس به صورت پشت سرهم انتخاب نشوند.

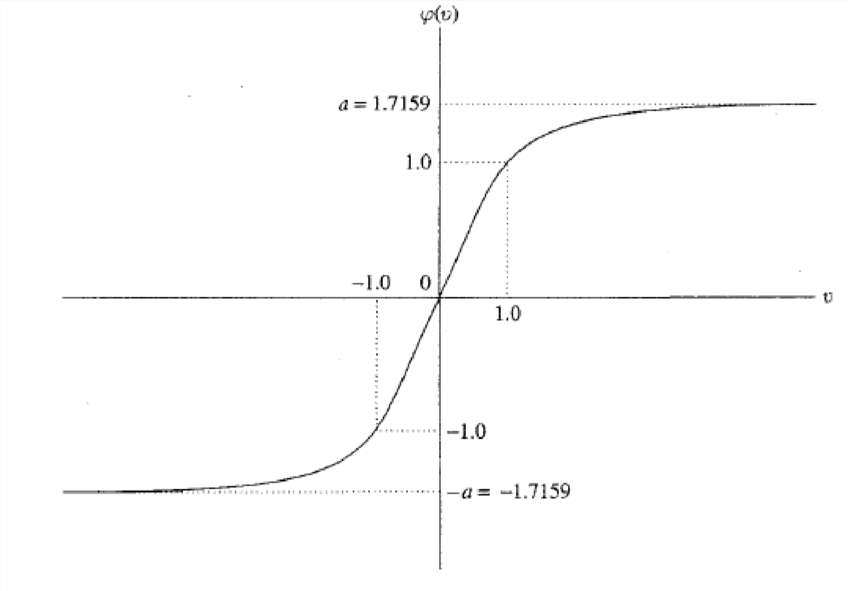
روش‌های بهبود کارایی(ادامه...)

emphasizing scheme

- نمونه‌های دشوارتر به شبکه بیشتر اعمال شوند. این شیوه البته با مشکلاتی هم روبرو است.
  - ترتیب اعمال نمونه‌های آموزشی به هم می‌ریزد.
  - یا داده‌های برون‌هشته(outlier) در مجموعه‌ی آموزشی وجود داشته باشد. چنین داده‌های «تعمیم‌پذیری» را با چالش مواجه می‌کند.

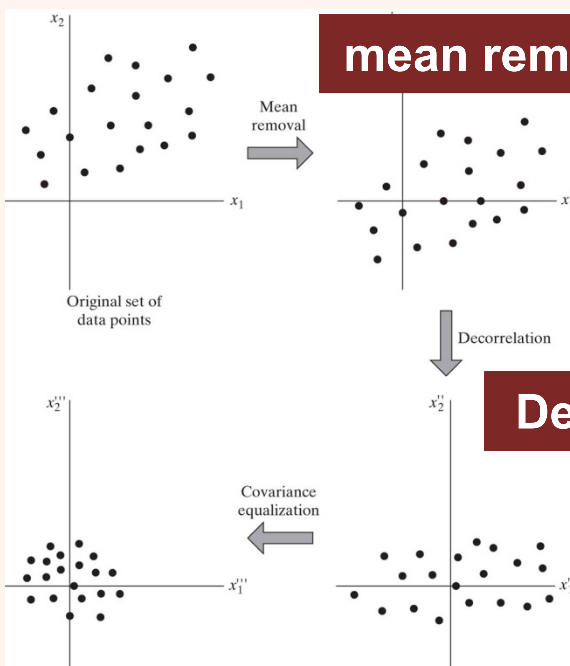
روش‌های بهبود کارایی(ادامه...)

- تابع انگیزش:
  - انتخاب تابع انگیزش مناسب در کارایی موثر است.
- مقدار خروجی مطلوب:
  - این مقدار باید در محدوده‌ی برد تابع فعالیت باشد، وگرنه پارامترهای آزاد به سمت بی‌نهایت می‌روند.



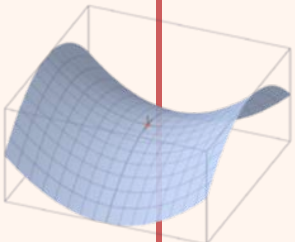
روش‌های بهبود کارایی(ادامه...)

- بهنجارسازی ورودی‌ها:
  - بر روی داده‌های ورودی باید پیش‌پردازش‌های صورت پذیرد. بهتر است میانگین ورودی‌ها صفر شود. در غیر این صورت به صورت زیگزاگ به سمت مینیمم حرکت می‌کند.



روش‌های بهبود کارایی(ادامه...)

- مقداردهی اولیه‌ی وزن‌ها:
  - مقادیر بزرگ←رفتن به ناحیه‌ی اشباع ←کند شدن آموزش
  - مقادیر کوچک←نزدیکی مبدأ ←مبدأ به صورت «نقطه‌ی زینی» است.
  - مقدار مناسب مقداری بین این دو است.



- ثابت می‌شود در صورتی که میانگین مقدار اولیه‌ی وزن‌ها صفر و انحراف معیار آن $\sigma_w = m^{-1/2}$ در نظر گرفته شود،(m تعداد اتصالات به نرون) نتیجه‌ی بهتری به دست می‌آید(زمانی که تابع انگیزش tanh باشد).

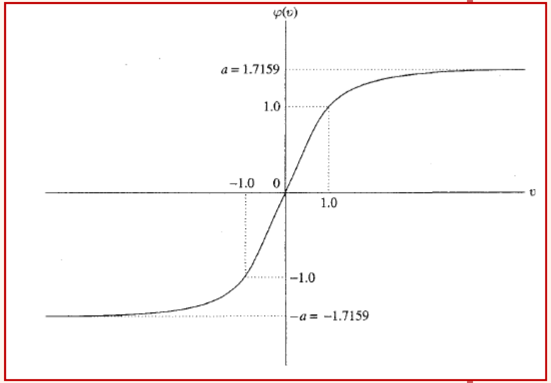
روش‌های بهبود کارایی(ادامه...)

مفروضات

$$v_j = \sum_{i=1}^{m} w_{ji} y_i$$

$$\mu_y = E\left[y_i\right] = 0 \quad \text{for all } i$$

$$\sigma_y^2 = E\left[\left(y_i - \mu_i\right)^2\right] = E\left[y_i^2\right] = 1 \quad \text{for all } i$$



$$E\left[y_i y_k\right] = \begin{cases} 1 & \text{for } k = i \\ 0 & \text{for } k \neq i \end{cases}$$

$$\mu_w = E\left[w_{ji}\right] = 0 \quad \text{for all } \left(j,i\right) \; pairs$$

$$\sigma_w^2 = E\left[\left(w_{ji} - \mu_w\right)^2\right] = E\left[w_{ji}^2\right] \quad \text{for all } \left(i,j\right) \; pairs$$

روش‌های بهبود کارایی(ادامه...)

اثبات

$$\mu_v = E\left[v_j\right] = E\left[\sum_{i=1}^{m} w_{ij} y_i\right] = \sum_{i=1}^{m} E\left[w_{ji}\right] E\left[y_i\right] = 0$$

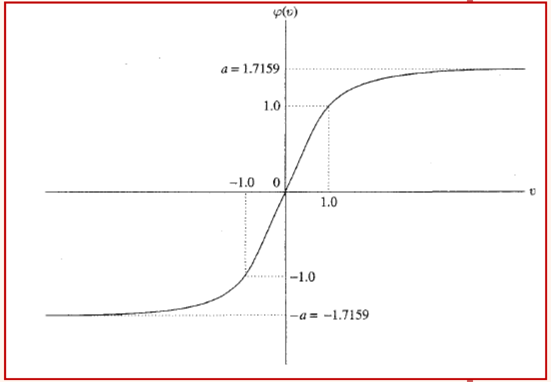
$$\sigma_v^2 = E\left[\left(v_j - \mu_v\right)^2\right] = E\left[v_j^2\right]$$

$$= E\left[\sum_{i=1}^{m}\sum_{k=1}^{m} w_{ji} w_{jk} y_i y_k\right]$$

$$= \sum_{i=1}^{m}\sum_{k=1}^{m} E\left[w_{ji} w_{jk}\right] E\left[y_i y_k\right]$$

$$= \sum_{i=1}^{m} E\left[w_{ji}^2\right]$$

$$= m\sigma_w^2$$



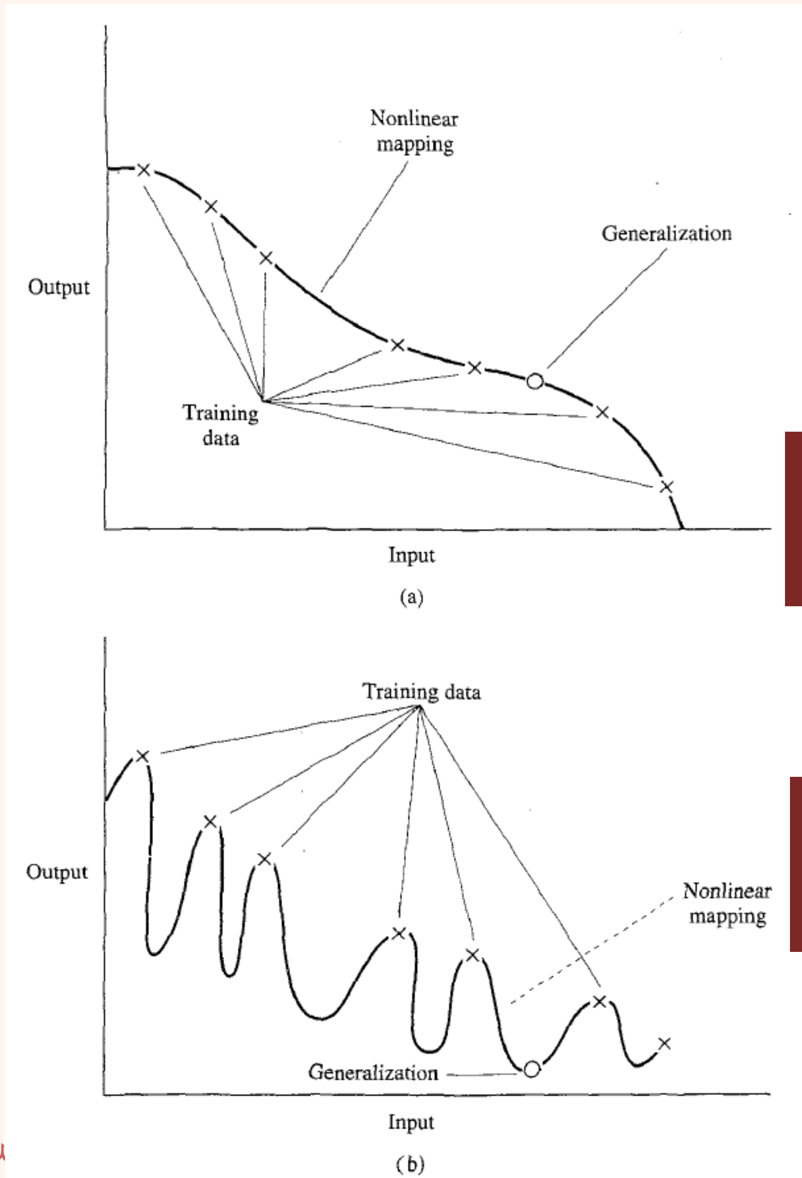
در صورتی که وزن‌ها به گونه‌ای در نظر گرفته شوند که انحراف آن‌ها $m^{-½}$ باشد، مقدار v واریانس یک خواهد داشت.

روش‌های بهبود کارایی(ادامه...)

- نرخ آموزش:
  - در مورد نرخ آموزش وفقی صحبت گردیم.
  - همه‌ی نرون‌ها باید با یک سرعت آموزش ببینند. نرون لایه‌ی آخر معمولا گرادیان بزرگ‌تری دارد. بنابراین بهتر است، نرخ آموزش لایه‌های آخر، کم‌تر در نظر گرفته شود.

تعمیم در شبکه‌های عصبی

- آموزش بیش از حد
- استفاده از لایه‌های مخفی بیش از حد نیاز



properly fitted data (good generalization)

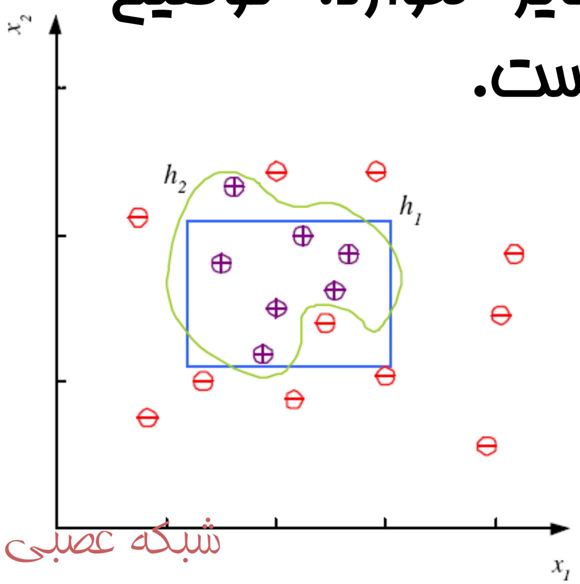
Overfitted data (poor generalization)

تعمیم در شبکه‌های عصبی

- آموزش شبکه
  - از تعدادی الگو برای آموزش شبکه استفاده می‌شود.
  - هنگامی که ورودی‌های دیگری غیر از داده‌های آموزشی به شبکه اعمال شود و شبکه (تقریبا) درست پاسخ دهد، گفته می‌شود «تعمیم‌پذیری» آن خوب است.
  - هنگامی که آموزش بیش از حد صورت گیرد مشکل (overfitting or overtraining) پیش می‌آید.
  - در این حالت ممکن است برای تعداد مشخصی الگو جواب خوب و در صورت تغییر الگوها پاسخ نامناسب دریافت کنیم.
- مرحله‌ی تست شبکه
  - از تعدادی الگو که در آموزش استفاده نشده جهت تست شبکه استفاده می‌کنیم.

تیغ Occam

- تیغ Occam اصلی منسوب به William of Ockham
- منطق‌دان و فیلسوف انگلیسی است. در قرن ۱۴ میلادی ویلیام اوکام اصلی را مطرح کرد که به نام اصل «تیغ Occam» شناخته شد. طبق این اصل، هر گاه دربارهٔ علت بروز پدیده‌ای دو توضیح مختلف ارائه شود، در آن توضیحی که پیچیده‌تر باشد احتمال بروز اشتباه بیشتر است و بنابراین، در شرایط مساوی بودن سایر موارد، توضیح ساده‌تر، احتمال صحیح بودنش بیشتر است.



تعمیم در شبکه‌های عصبی(ادامه...)

- سه فاکتور در تعمیم‌پذیری مؤثر هستند:
  - حجم مجموعه‌ی آموزشی و توزیع آن(تا چه حد گویاست)
  - ساختار شبکه‌ی عصبی (پیچیدگی مدل)
  - پیچیدگی مسأله

  - تعداد نمونه‌های آموزشی برای تعمیم‌پذیری مناسب

$$N = O\left(\frac{W}{\varepsilon}\right)$$

روش‌های سرعت بخشیدن به همگرایی

واقعی

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

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

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

$$y_j=\varphi(v_j)$$

مطلوب

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

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

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

روش‌های سرعت بخشیدن به همگرایی

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

ورودی به شاخه‌ی زام

اگر f تابع sigmoid باشد

$$y_j = \varphi(v_j)$$

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

$\Delta w_{ji} <<$

$e_j <<$ مناسب

e_j بزرگ ولی $y_j \approx 0$

e_j بزرگ ولی $y_j \approx 1$

نامناسب

کند شدن همگرایی

پیشنهاد

- تعویض تابع معیار خطا
  - در روش عادی B.P تابع معیار خطا

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

  - روش *Van Oyen, Nienhuis* (V-O)

$$E(k) = -\sum_{h=1}^{M} [\underbrace{d_h}_{\text{خروجی مطلوب}} \ln(\underbrace{y_h}_{\text{خروجی واقعی}}(k)) + (1 - d_h)\ln(1 - y_h(k))]$$

Improving the convergence of the back-propagation algorithm
Van Ooyen, A., and Nienhuis, B. (1992). *Neural Networks* 5: 465-471

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

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

$$= \underbrace{[d_j \times \frac{1}{y_j(k)} - (1-d_j)\times\frac{1}{(1-y_j(k))}].y_j(k)(1-y_j(k))}_{e_j}.y_i$$

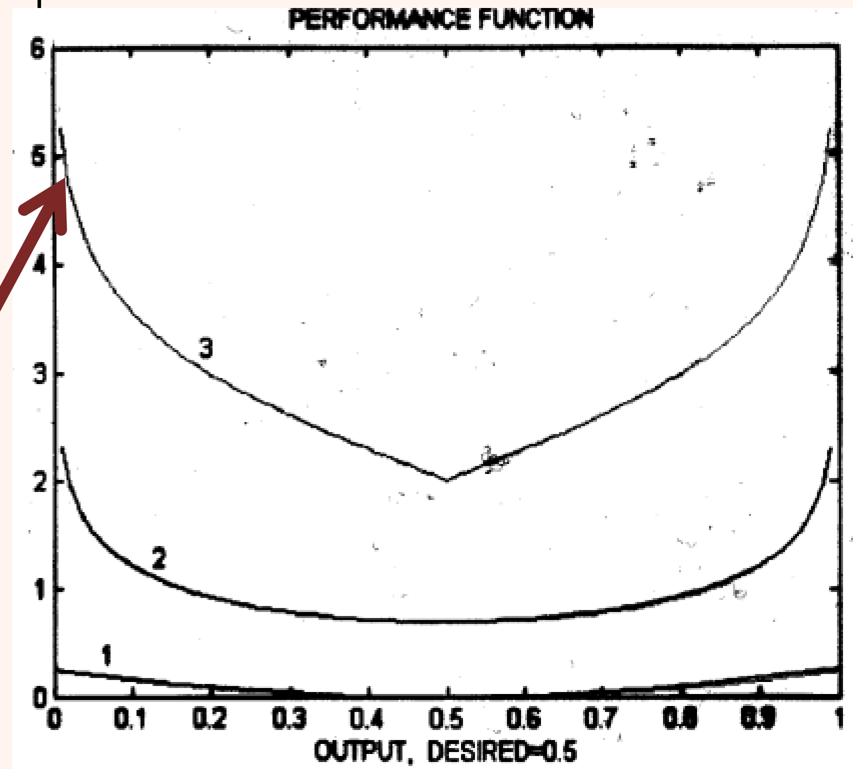
خروجی لایه ما قبل آخر

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

$$\Delta w_{ji}(k) = \frac{\partial E(k)}{\partial w_{ji}(k)} = \underbrace{\frac{\partial E(k)}{\partial y_j(k)} \cdot \frac{\partial y_j(k)}{\partial v_j(k)}}_{sign(e)\left|e\right|^m} \cdot \frac{\partial v_j(k)}{\partial w_{ji}(k)}$$

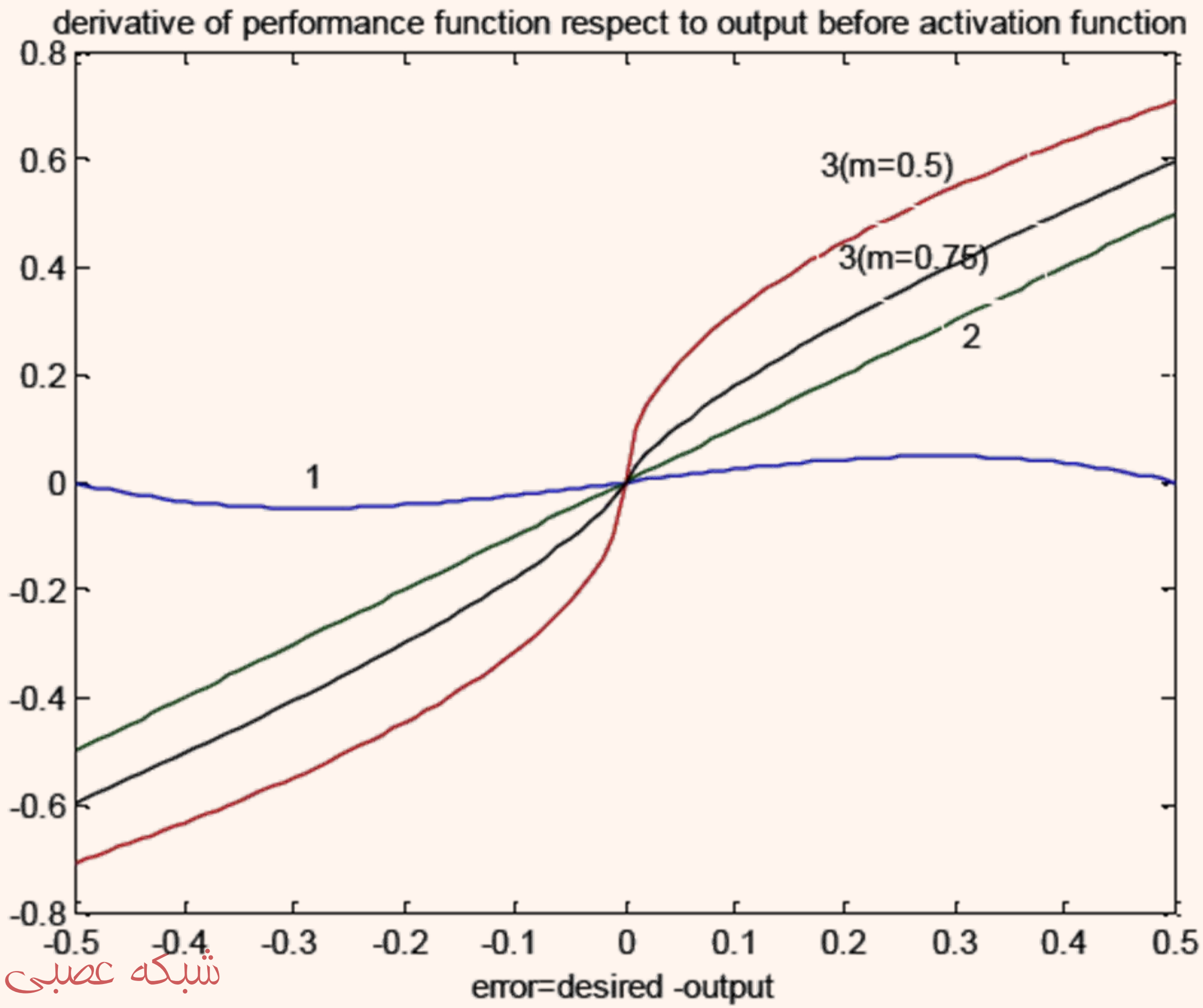
$$0 < m < 1$$

$$\boxed{\begin{aligned}\Phi_{new}(s) = {} & (1-T)^m.\left[\log_e(s) - \log_e(1-s)\right] + \\ & \sum_{j=1}^{p}\left\{\left(\prod_{i=0}^{j-1}(m-i)\right).\frac{(1-T)^{m-j}}{j!}.(-1)^j.\left[\log_e(s) - \binom{j}{1}.s + \binom{j}{2}.\frac{s^2}{2} + \dots \right.\right. \\ & \left.\left. (-1)^r.\binom{j}{r}.\frac{s^r}{r} + \dots + (-1)^j.\frac{s^j}{j}\right]\right\} \\ & \forall \begin{cases} s = O & \text{if } E > 0 \\ s = 2T - O & \text{if } E < 0 \end{cases}\end{aligned}}$$



$$\Delta w_{ji}(k)=\frac{\partial E(k)}{\partial w_{ji}(k)}=\underbrace{\frac{\partial E(k)}{\partial y_j(k)}\cdot\frac{\partial y_j(k)}{\partial v_j(k)}}_{sign(e)|e|^m}\cdot\frac{\partial v_j(k)}{\partial w_{ji}(k)}$$

$$0<m<1$$



1) $y(1-y)e$

2) e

3) $sign(e)|e|^m$

شبکه عصبی

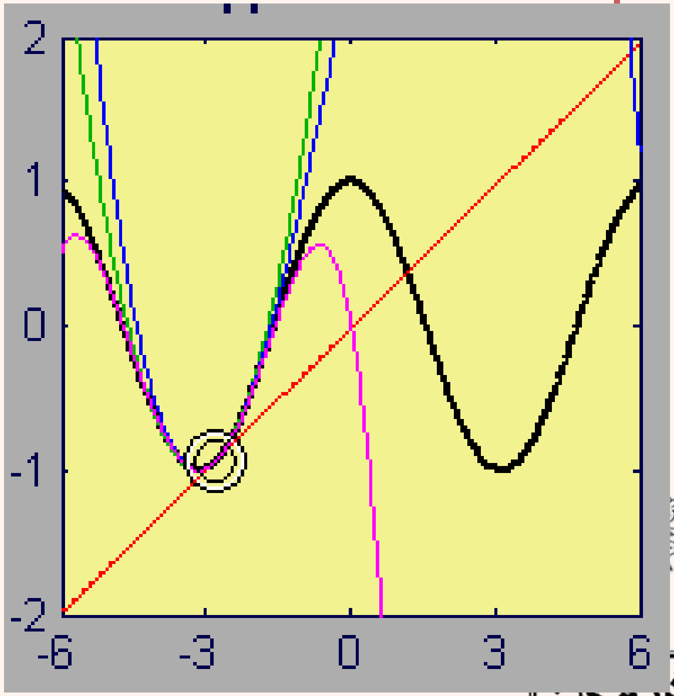
بسط تیلور

- توابع تحلیلی قابل تقریب زدن با چندجمله‌ای‌ها هستند.

$$F(x) = F(x^*) + \frac{d}{dx}F(x)\Big|_{x = x^*}(x - x^*)$$

$$+ \frac{1}{2}\frac{d^2}{dx^2}F(x)\Bigg|_{x = x^*}(x - x^*)^2 + \frac{1}{4}$$

$$+ \frac{1}{n!}\frac{d^n}{dx^n}F(x)\Bigg|_{x = x^*}(x - x^*)^n + \frac{1}{4}$$



حالت برداری

$$F(\mathbf{x}) = F(x_1, x_2, \ldots, x_n)$$

$$F(\mathbf{x}) = F(\mathbf{x}^*) + \frac{\partial}{\partial x_1} F(\mathbf{x}) \Big|_{\mathbf{x} = \mathbf{x}^*} (x_1 - x_1^*) + \frac{\partial}{\partial x_2} F(\mathbf{x}) \Big|_{\mathbf{x} = \mathbf{x}^*} (x_2 - x_2^*)$$

$$+ \cdots + \frac{\partial}{\partial x_n} F(\mathbf{x}) \Big|_{\mathbf{x} = \mathbf{x}^*} (x_n - x_n^*) + \frac{1}{2} \frac{\partial^2}{\partial x_1^2} F(\mathbf{x}) \Big|_{\mathbf{x} = \mathbf{x}^*} (x_1 - x_1^*)^2$$

$$+ \frac{1}{2} \frac{\partial^2}{\partial x_1 \partial x_2} F(\mathbf{x}) \Big|_{\mathbf{x} = \mathbf{x}^*} (x_1 - x_1^*)(x_2 - x_2^*) + \cdots$$

فرم ماتریسی

$$F(\mathbf{x}) = F(\mathbf{x}^*) + \nabla F(\mathbf{x})^T \big|_{\mathbf{x}=\mathbf{x}^*} (\mathbf{x}-\mathbf{x}^*)$$

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

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

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

الگوریتم‌های بهینه‌سازی

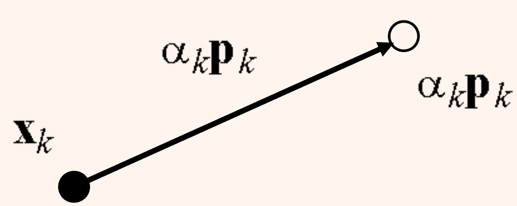
برای یافتن نقطه‌ی مینیمم خطا بر روی رویه‌ی کارایی (performance(error) surface) شیوه‌های متفاوتی وجود دارد.

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

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

یا

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



p_k - Search Direction

α_k - Learning Rate

الگوریتم‌های بهینه‌سازی

- 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 مرتبه‌ی ۱

- 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

الگوریتم‌های بهینه‌سازی

- 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:

$$g_k^T p_k < 0$$

- 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؟

$$\mathbf{p}_k = -\mathbf{g}_k$$

- برای متد steepest decent داریم:

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

مثال

- Steepest decent را برای مساله‌ی زیر اعمال کنید:

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

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

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

مثال

- با در نظر گرفتن α=0.01 خواهیم داشت:

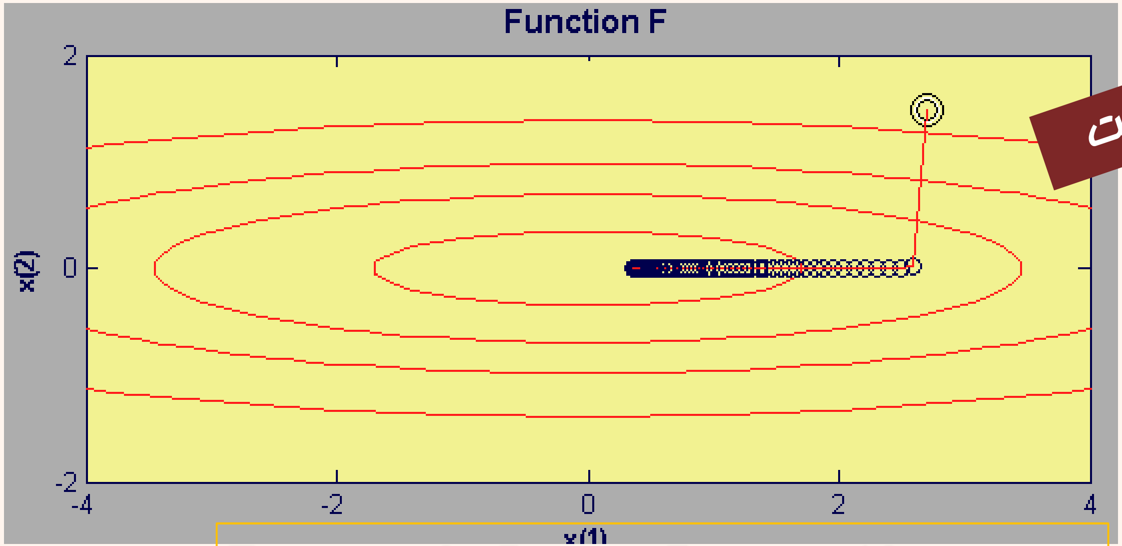
First iteration

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

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

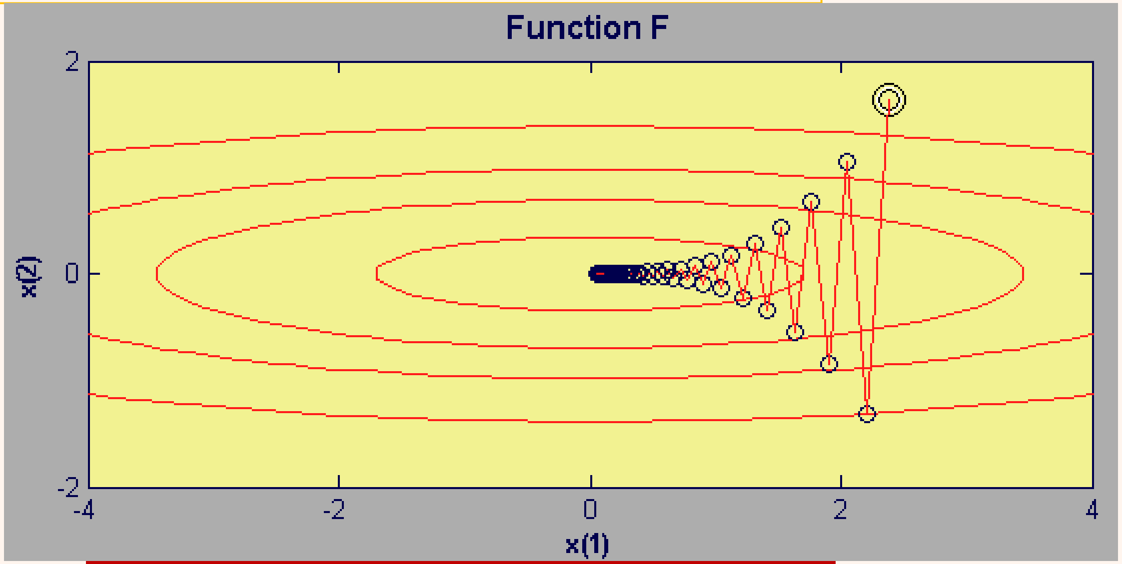


Trajectory for Steepest Descent with α = 0.01

مثال

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

Trajectory for Steepest Descent with α = 0.035



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

مثال

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

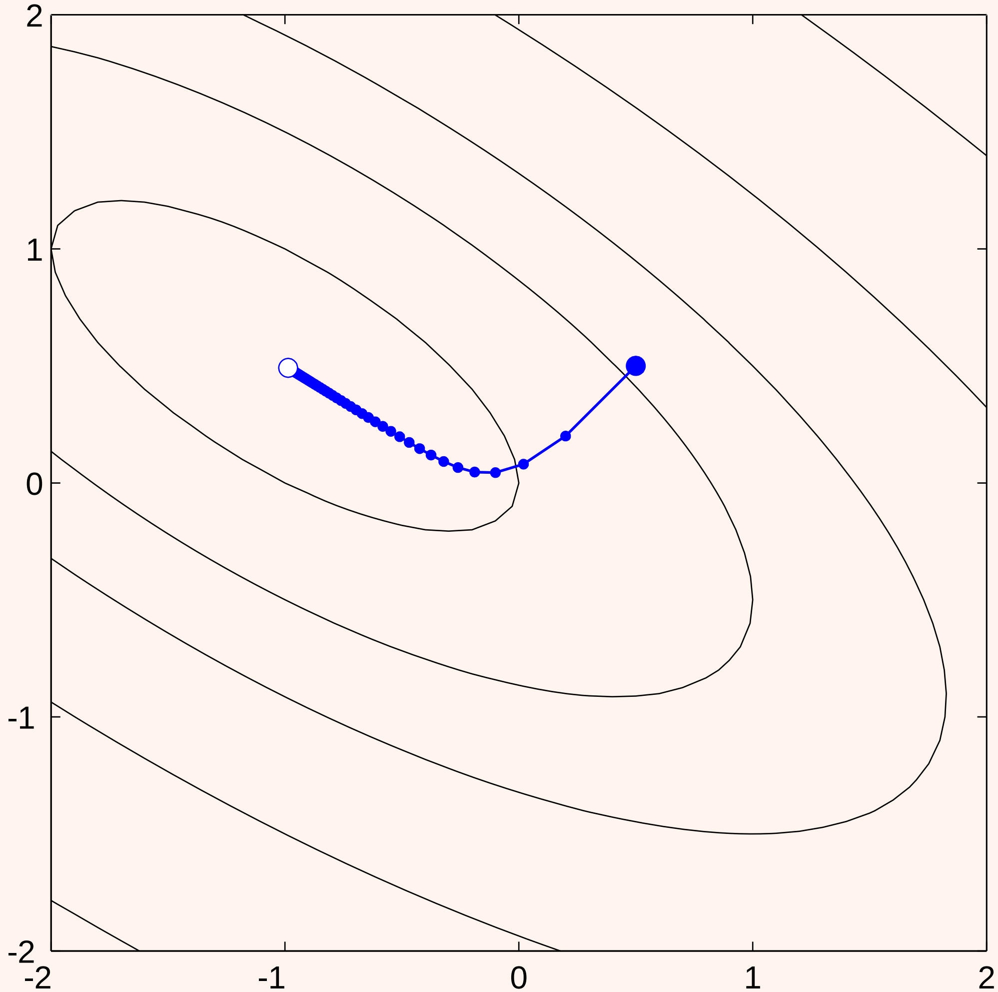
$$\mathbf{x}_0 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \qquad \alpha = 0.1$$

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

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

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

نمودار



Stable Learning Rates (Quadratic)

آموزش پایدار

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

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

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha\mathbf{g}_k = \mathbf{x}_k - \alpha(\mathbf{A}\mathbf{x}_k + \mathbf{d}) \Longrightarrow \mathbf{x}_{k+1} = \underbrace{[\mathbf{I} - \alpha\mathbf{A}]}\mathbf{x}_k - \alpha\mathbf{d}$$

پایداری وابسته به مقادیر ویژه‌ی این ماتریس است

(λ_i - eigenvalue of A)

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

Eigenvalues of [I - αA].

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

Modern Control Theory (3rd dition) William L. Brogan

مثال

- با اعمال این مساله بر مثال قبلی برآنیم بیشترین میزان نرخ آموزش مجاز را محاسبه کنیم

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

- 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:

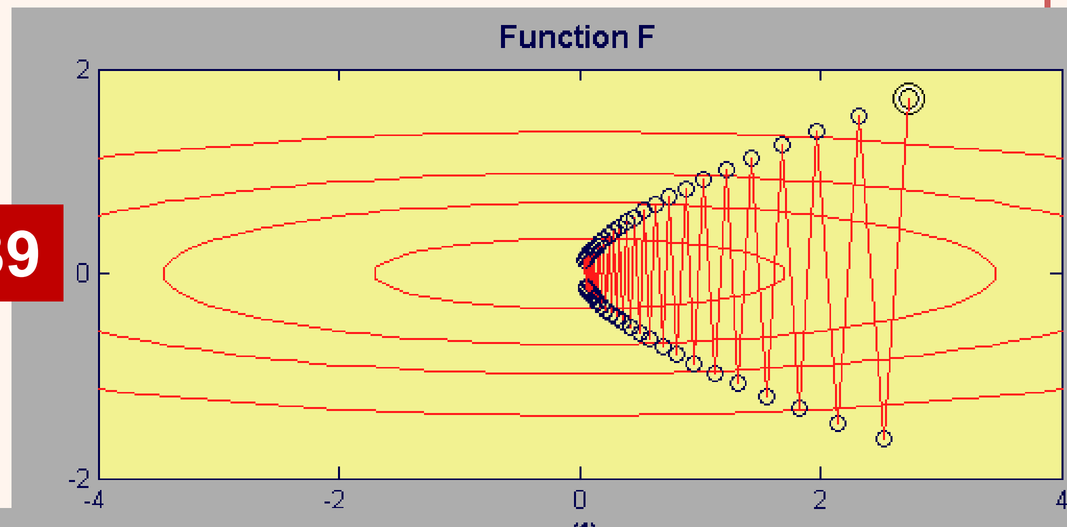
$$\left\{(\lambda_1 = 2), \left(z_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix} \right)\right\}, \left\{(\lambda_2 = 50), \left(z_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \right)\right\}$$

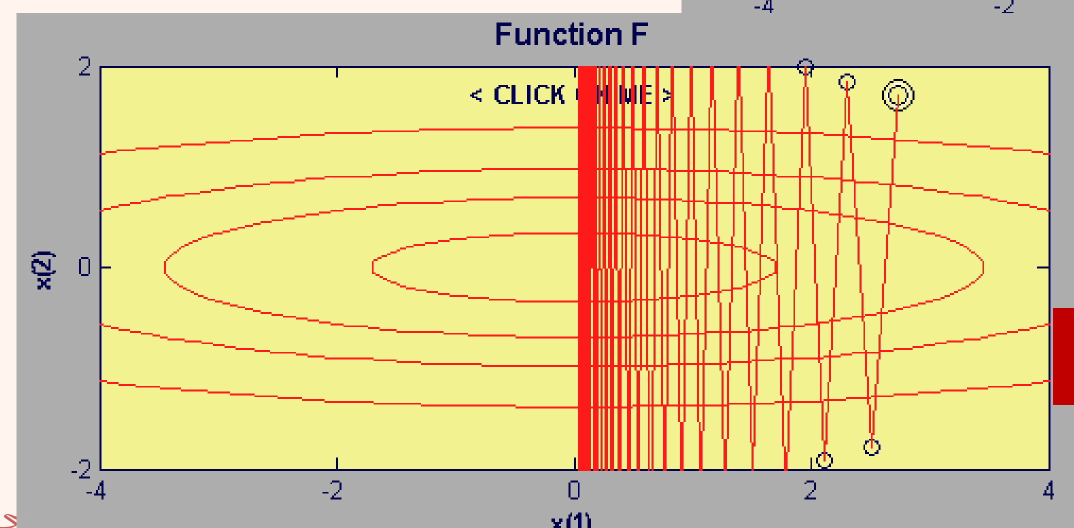
مثال-ادامه

- پس بیشترین میزان نرخ آموزش از رابطه‌ی زیر به دست می‌آید:

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

α=0.039





α=0.041

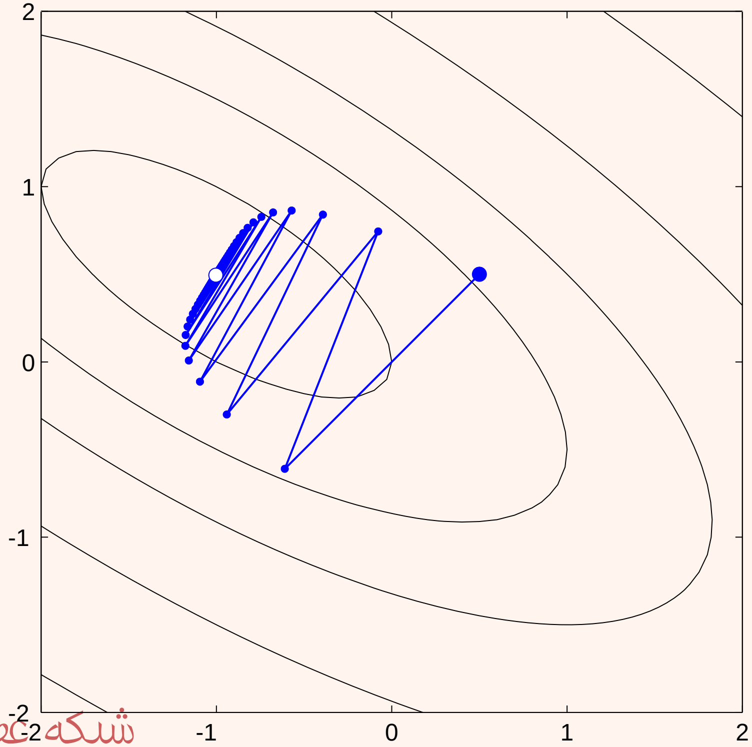
مثال

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

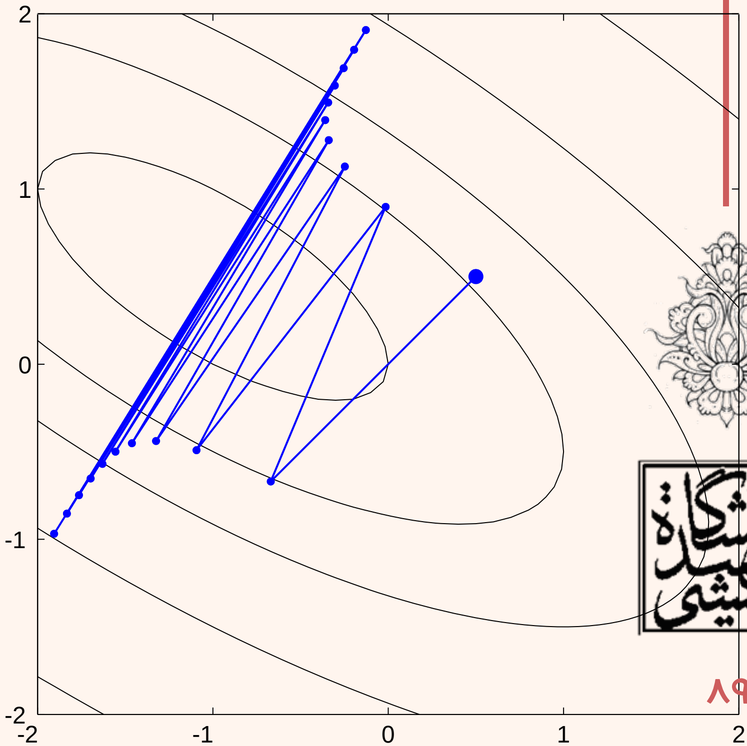
$$\mathbf{A} = \begin{bmatrix} 2 & 2 \\ 2 & 4 \end{bmatrix} \qquad \left\{ (\lambda_1 = 0.764), \left(\mathbf{z}_1 = \begin{bmatrix} 0.851 \\ -0.526 \end{bmatrix} \right) \right\}, \left\{ \lambda_2 = 5.24, \left(\mathbf{z}_2 = \begin{bmatrix} 0.526 \\ 0.851 \end{bmatrix} \right) \right\}$$

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

$\alpha = 0.37$



$\alpha = 0.39$



تنظیم نرخ یادگیری Minimizing Along a Line

- در مورد انتخاب نرخ آموزش به صورت وفقی 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 $\boldsymbol{p}_k$ جستجویی صورت پذیرد.
- برای توابع درجه‌ی دوم می‌توان راه حلی تحلیلی ارائه نمود:

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

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

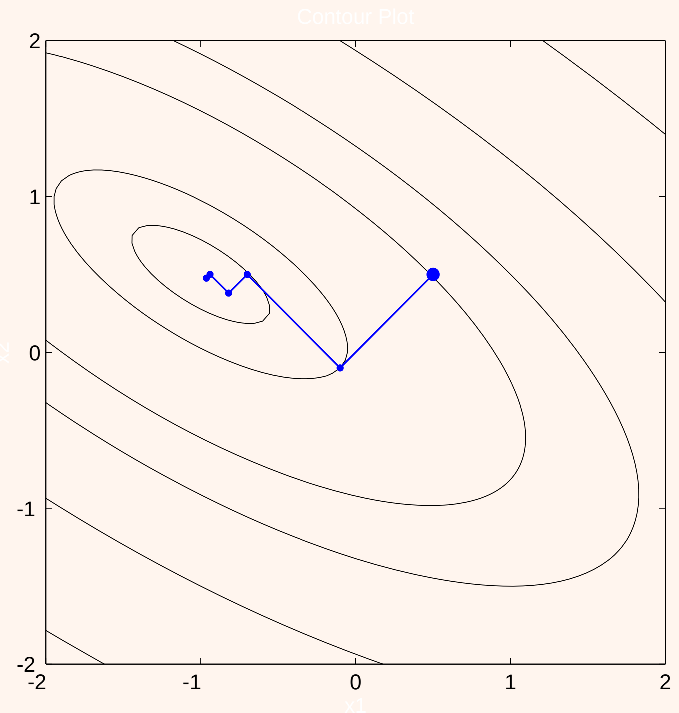
$$\mathbf{A}_k \equiv \nabla^2 F(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}_k}$$

مثال

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

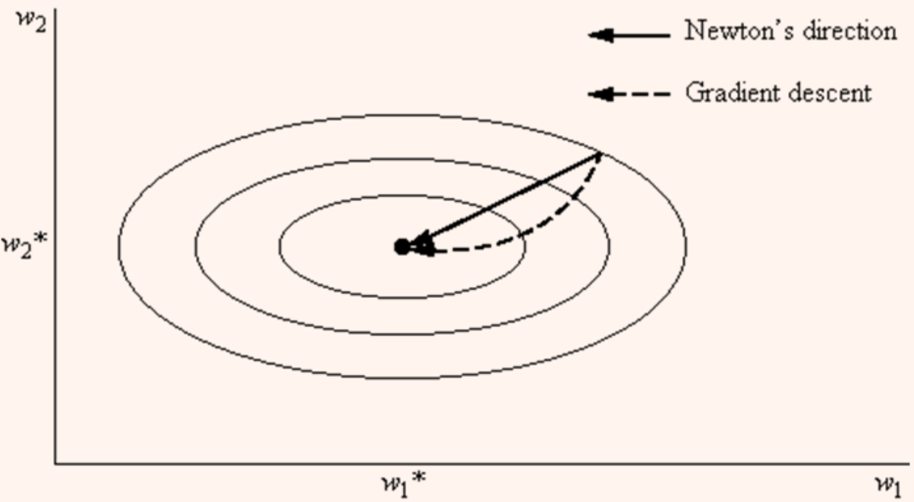
$$\nabla F(\mathbf{x}) = \begin{bmatrix}\frac{\partial}{\partial x_1}F(\mathbf{x})\\ \frac{\partial}{\partial x_2}F(\mathbf{x})\end{bmatrix} = \begin{bmatrix}2x_1 + 2x_2 + 1\\ 2x_1 + 4x_2\end{bmatrix} \qquad \mathbf{p}_0 = -\mathbf{g}_0 = -\nabla F(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}_0} = \begin{bmatrix}-3\\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

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



روش نیوتن

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



در مساله‌ی 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 به‌گونه‌ای که $f'(x^*)=0$

http://www.nd.com/NSBook/NEURAL%20AND%20ADAPTIVE%20SYSTEMS18_Newtons_Method.html

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:

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

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

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

روش نیوتن

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

- به بیانی دیگر 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 مرتبه‌ی ۲

با مشتق گرفتن از این تقریب و قرار دادن رابطه برابر با صفر نقطه ای با ثبات را بیابیم

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

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

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

مثال ۱

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

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

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

$$\mathbf{x}_1 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} - \begin{bmatrix} 2 & 0 \\ 0 & 50 \end{bmatrix}^{-1} \begin{bmatrix} 1 \\ 25 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} - \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} = 0$$

مثال ۲

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

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

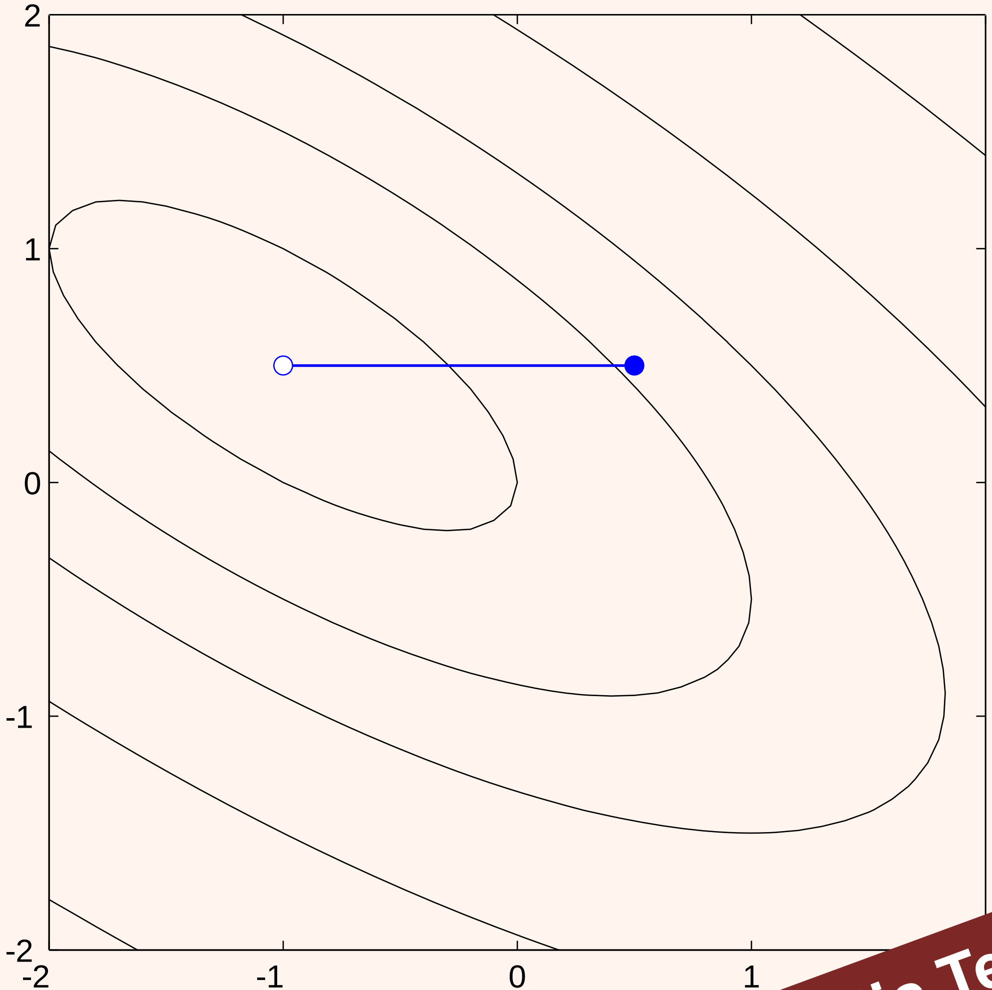
$$\nabla F(\mathbf{x}) = \begin{bmatrix} \frac{\partial}{\partial x_1}F(\mathbf{x}) \\ \frac{\partial}{\partial x_2}F(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} 2x_1 + 2x_2 + 1 \\ 2x_1 + 4x_2 \end{bmatrix}$$

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

$$\mathbf{A} = \begin{bmatrix} 2 & 2 \\ 2 & 4 \end{bmatrix}$$

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

Plot

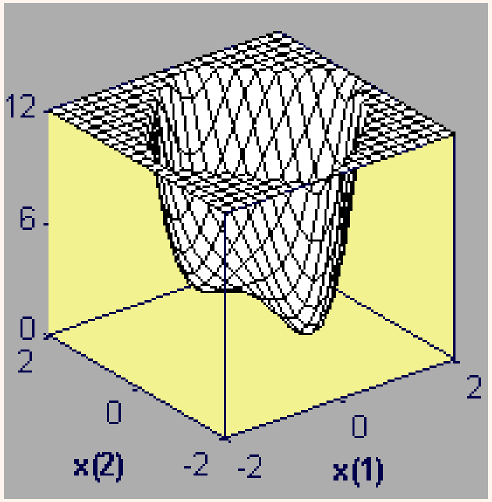


Quadratic Termination

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.

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

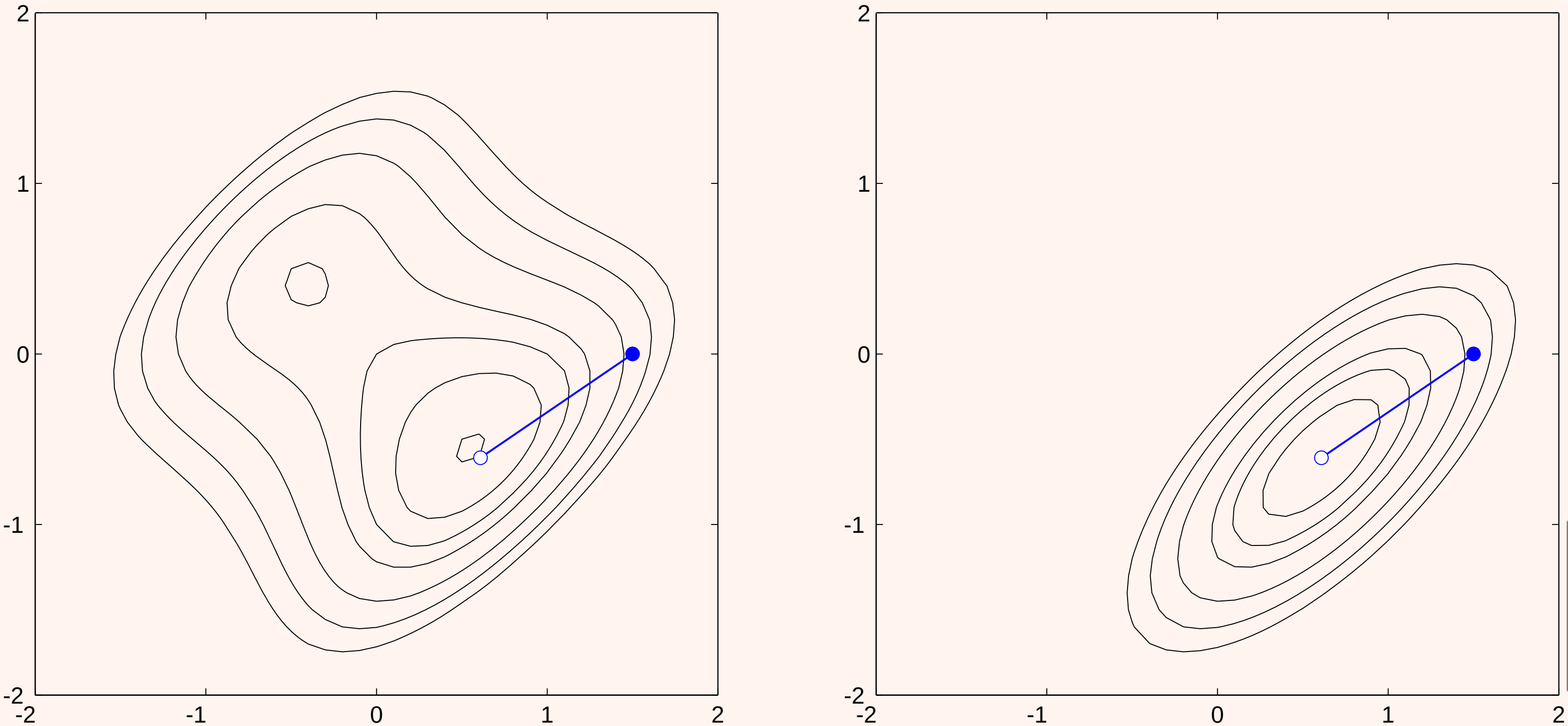


شرایط اولیه‌ی متفاوت

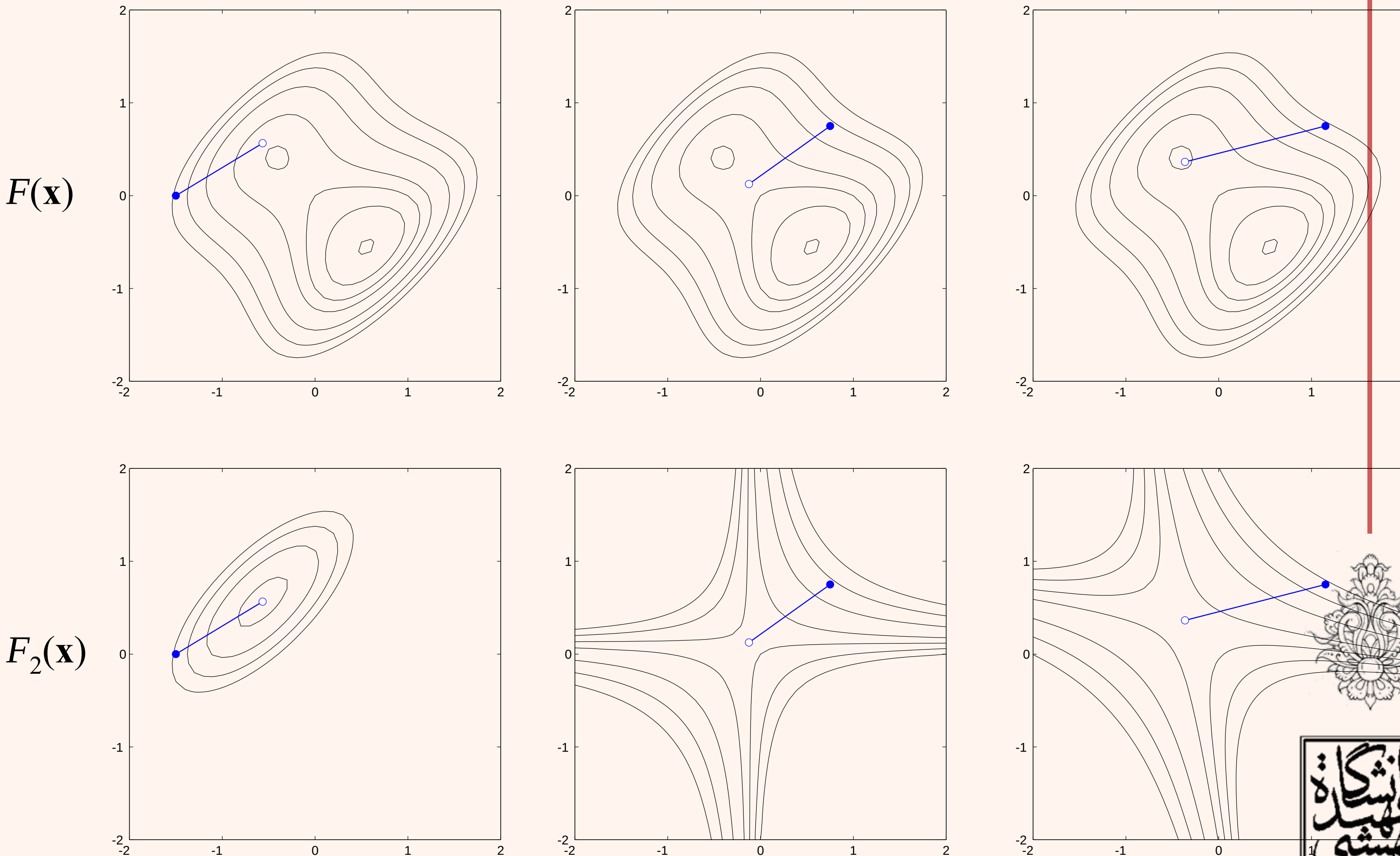
$$F(\mathbf{x}) = (x_2 - x_1)^4 + 8x_1x_2 - x_1 + x_2 + 3$$

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

$F(\mathbf{x})$ $\qquad\qquad F_2(\mathbf{x})$



شرایط اولیه‌ی متفاوت



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

- در روش عادی B.P داشتیم:

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

- در روش نیوتن داریم:

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

$$\boldsymbol{F_k(X) = \sum_{i=1}^{M} e_i^2(k)}$$

تابع معیار خطا

- که در آن

$$g_k = \nabla F_k(x)\Big|_{x = x_k}$$

$$A_k = \nabla^2 F_k(x)\Big|_{x = x_k}$$

P کل تعداد مجهولات

Gauss–Newton algorithm

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

تابع معیار خطا

$$F_k(x) = \sum_{i=1}^{M} e_i^2(k) = E_k^T(X)E_k(X)$$

$$E_k(X) = [\mathrm{e}_1 \ \mathrm{e}_2 \ \dots \ \mathrm{e}_\mathrm{M}]^T$$

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

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

ماتریس ژاکوبین

$$J(X) = \begin{bmatrix} \frac{\partial e_1(x)}{\partial x_1} & \dots & \frac{\partial e_1(x)}{\partial x_P} \\ . & & . \\ . & & . \\ \frac{\partial e_M(x)}{\partial x_1} & \dots & \frac{\partial e_M(x)}{\partial x_P} \end{bmatrix}$$

فرض می‌شود تمامی محاسبات در تکرار k است که برای سادگی از نوشتن آن صرفنظر شده است

محاسبه مشتق دوم

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

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

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

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

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

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 **Quasi Newton** گویند

- در روش نیوتن داشتیم

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

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

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

برای محاسبه‌ی رابطه‌ی مذکور تنها از مشتق اول استفاده می‌شود

Quasi Newton

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

آیا این ماتریس همواره معکوس‌پذیر است؟

Levenberg-Marqualt

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

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

$$G_k = H_k + \mu_k I$$

بسیار کوچک

- اگر μ=0 باشد روش نیوتن است.

Levenberg-Marqualt

یادآوری

مقدار ویژه

$$Aq_i = \lambda_i q_i$$

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

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

Levenberg-Marqualt

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

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

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

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

Levenberg-Marqualt

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

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

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

- 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.

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 دوم لازم است.
- هدف یافتن روشی است که **بدون نیاز به محاسبه‌ی مشتق دوم** همگرایی را افزایش دهد.
- از طرفی استفاده از steepest descent باعث حرکت زیزاگی 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 A (که positive definite) است، conjugate نامیده می‌شوند:

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

یک مجموعه از این بردارها، بردارهای ویژه‌ی ماتریس است.
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(توابع درجه‌ی دو).

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

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

Conjugate gradient

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

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

تغییرات گرادیان در گام kام

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

که

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

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

بدون نیاز به محاسبه‌ی بردار Hessian بردار conjugate محاسبه 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:

$$\{\Delta g_0, \Delta g_1, \Delta g_2, \cdots, \Delta g_{k-1}\}$$

Conjugate gradient

- مرحله‌ی اول مشابه steepest decent است.

F تابع معیار خطاست

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

- 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 تغییرات گرادیان واقع شود.

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

اثر گرادیان‌های قبلی

Gram-Schmidt Orthogonalization

Conjugate gradient

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

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

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:

$$\Delta g_{k-1}^{T} = (g_k - g_{k-1})^{T}$$

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

Hestenes and Steifel

Polak and Ribiere

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

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

Fletcher and Reeves

Conjugate gradient

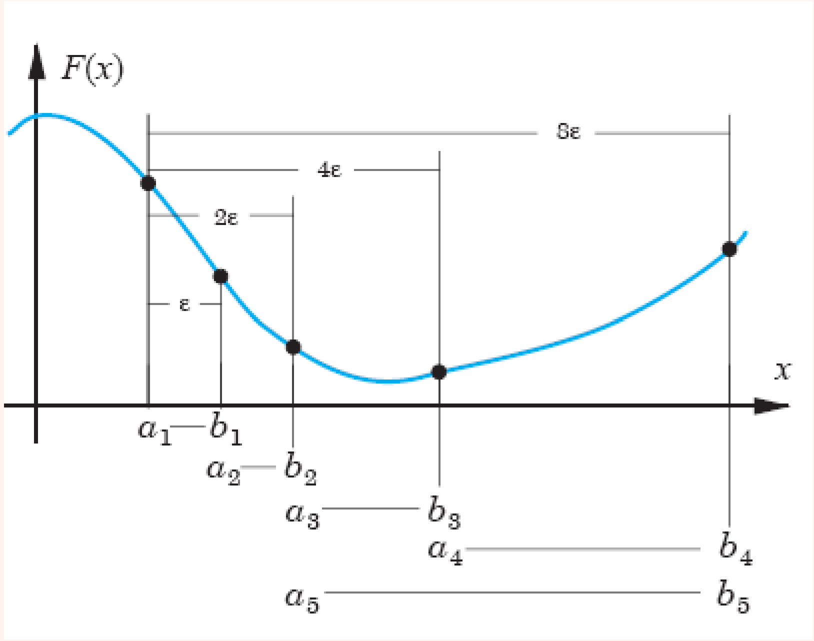
- الگوریتم conjugate gradient به صورت خلاصه:
  - جهت جستجوی اولیه را به‌گونه‌ای که بر جهت مشتق عمود باشد در نظر می‌گیریم $p_0 = -g_0$
  - مقدار α را که همان نرخ یادگیری است محاسبه می‌نماییم تا رابطه‌ی زیر به دست آید، با توجه به این که این شیوه به نرخ آموزش حساس است و نرخ آموزش نیز به ماتریس Hessian وابسته است، برای تخمین این میزان از روش‌های تکرارشونده برای محاسبه‌ی نرخ آموزش استفاده می‌شود.

(For quadratic functions.)

$$x_{k+1} = x_k + \alpha_k p_k$$

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

Conjugate gradient



– با محاسبه‌ی β مقدار زیر را که همان جهت بهینه‌ی جستجوست محاسبه می‌شود.

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

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

مثال

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

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

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

مثال-ادامه

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

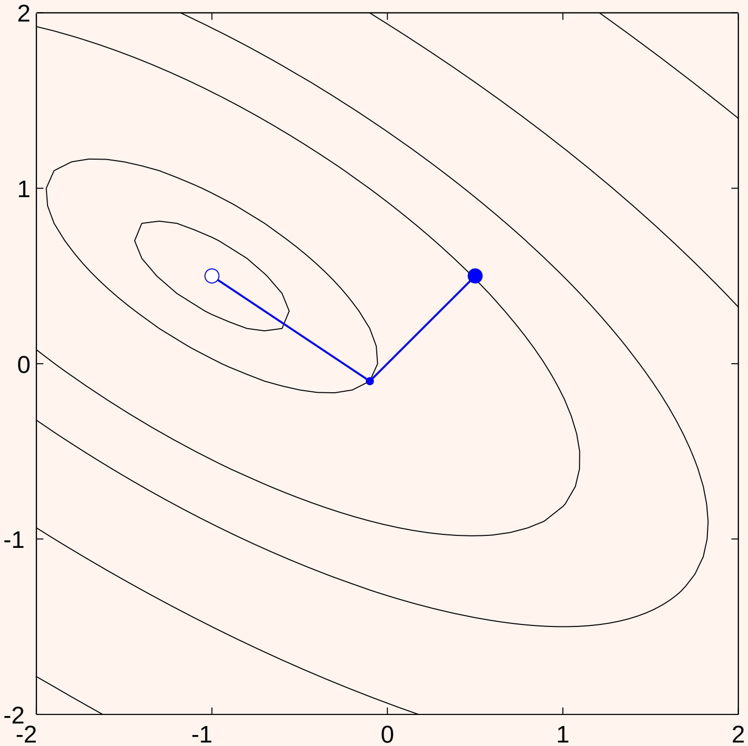
$$\beta_1 = \frac{\mathbf{g}_1^T \mathbf{g}_1}{\mathbf{g}_0^T \mathbf{g}_0} = \frac{\begin{bmatrix} 0.6 & -0.6 \end{bmatrix}\begin{bmatrix} 0.6 \\ -0.6 \end{bmatrix}}{\begin{bmatrix} 3 & 3 \end{bmatrix}\begin{bmatrix} 3 \\ 3 \end{bmatrix}} = \frac{0.72}{18} = 0.04$$

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

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

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

Conjugate Gradient



Steepest Descent