بسم الله الرحمن الرحیم

شبکه‌های عصبی مصنوعی

درس ۱۱

پس‌انتشار خطا

Error Backpropagation

کاظم فولادی قلعه
دانشکده مهندسی، پردیس فارابی
دانشگاه تهران

http://courses.fouladi.ir/nn

Backpropagation

پس‌انتشار

BACKPROPAGATION

پس‌انتشار، تعمیم الگوریتم LMS است:
قابل استفاده برای آموزش شبکه‌های چندلایه

پس‌انتشار مانند LMS یک الگوریتم کاهش تندترین شیب تقریبی است:
شاخص کارآیی: MSE
تفاوت با LMS: در نحوه‌ی محاسبه‌ی مشتقات

- **در شبکه‌های خطی تک‌لایه:**
  خطا تابع صریح خطی از وزن‌های شبکه است ⇐ محاسبه‌ی مستقیم مشتق نسبت به وزن‌ها
- **در شبکه‌های چندلایه با توابع انتقال غیرخطی:**
  رابطه‌ی خطا و وزن‌های شبکه پیچیده‌تر است ⇐ محاسبه‌ی مشتق با استفاده از قاعده‌ی زنجیری حسابان

هم روزنبلات (پرسپترون) و هم ویدرو (آدالاین)
شبکه‌های چندلایه را برای عبور از محدودیت‌های شبکه‌ی تک‌لایه پیشنهاد کردند
اما نتوانستند الگوریتم یادگیری خود را برای آموزش این شبکه‌ها تعمیم دهند.

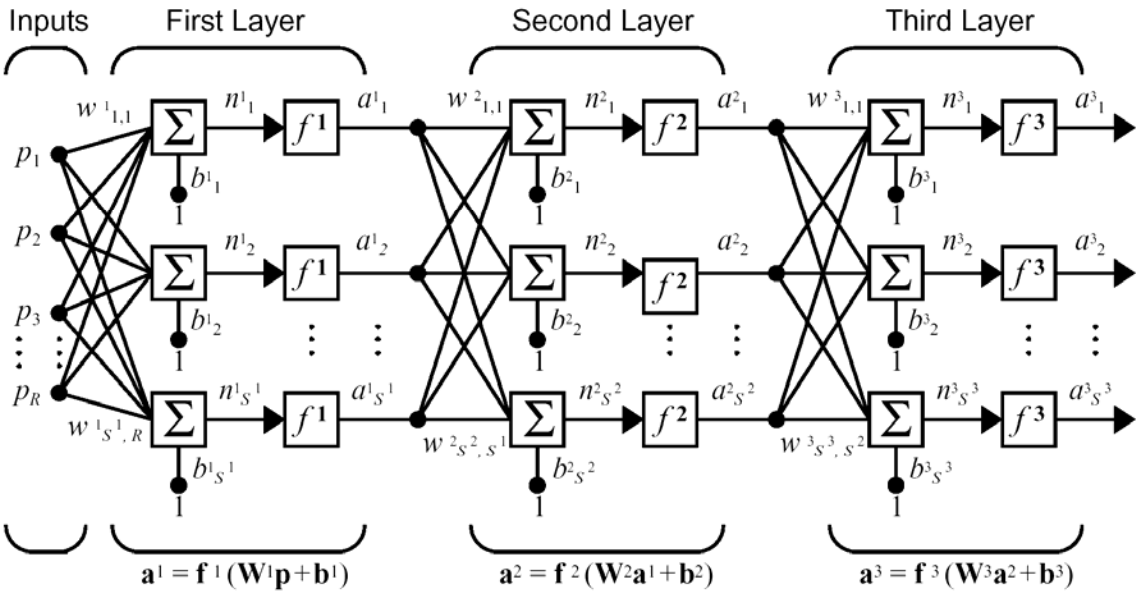
الگوریتم یادگیری شبکه‌ی چند لایه اولین بار توسط Werbos در 1974 ارائه شد
اما عمومی شدن آن با کتاب PDP توسط Parker در 1985 اتفاق افتاد.

پس‌انتشار خطا

۱

پرسپترون چندلایه

Multilayer Perceptron



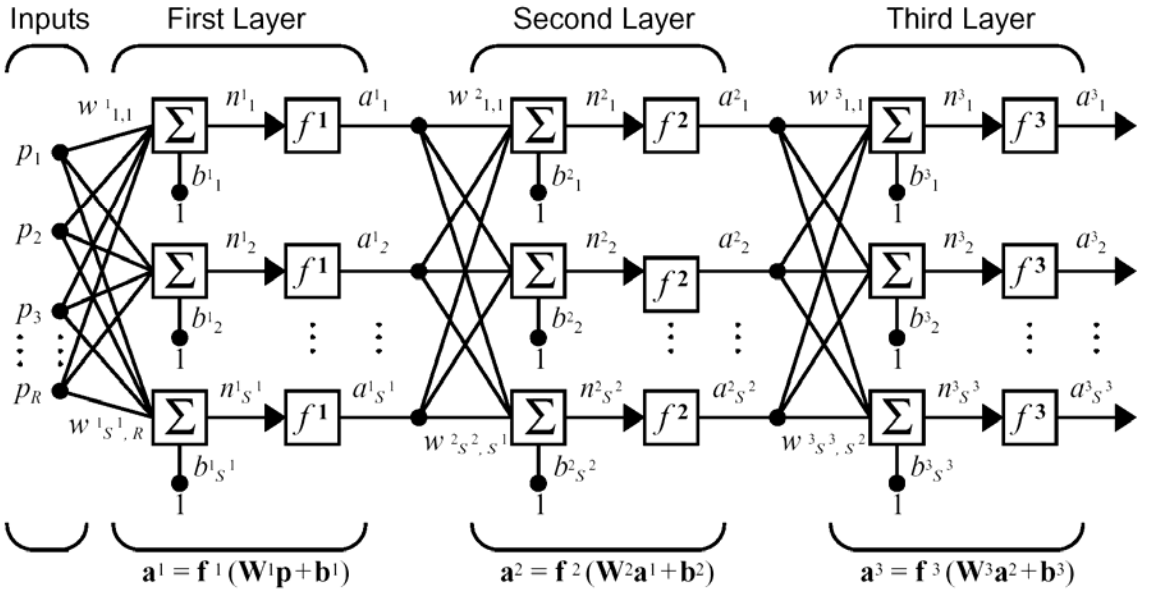
$$a^3 = f^3(W^3 f^2(W^2 f^1(W^1 p + b^1) + b^2) + b^3)$$

$R - S^1 - S^2 - S^3$ Network

پرسپترون چندلایه

اتصال آبشاری سه پرسپترون

MULTILAYER PERCEPTRON



$$\mathbf{a}^3 = \mathbf{f}^3(\mathbf{W}^3\mathbf{f}^2(\mathbf{W}^2\mathbf{f}^1(\mathbf{W}^1\mathbf{p}+\mathbf{b}^1)+\mathbf{b}^2)+\mathbf{b}^3)$$

$$R - S^1 - S^2 - S^3 \quad \text{Network}$$

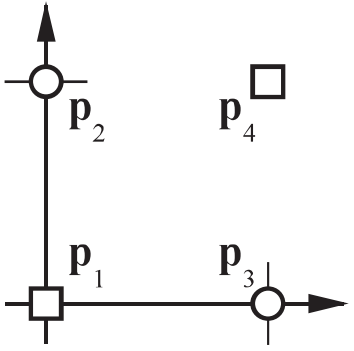
کوته‌نوشت:

پرسپترون چندلایه

مثال: یای انحصاری (xor) (۱ از ۳)

MULTILAYER PERCEPTRON

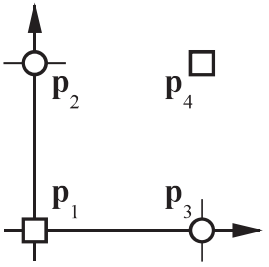
$$\left\{\mathbf{p}_1 = \begin{bmatrix}0\\0\end{bmatrix}, t_1 = 0\right\}\left\{\mathbf{p}_2 = \begin{bmatrix}0\\1\end{bmatrix}, t_2 = 1\right\}\left\{\mathbf{p}_3 = \begin{bmatrix}1\\0\end{bmatrix}, t_3 = 1\right\}\left\{\mathbf{p}_4 = \begin{bmatrix}1\\1\end{bmatrix}, t_4 = 0\right\}.$$

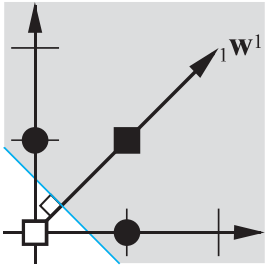


پرسپترون چندلایه

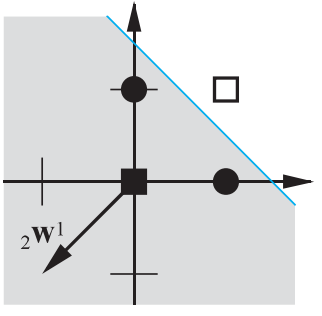
مثال: یای انحصاری (xor) (۲ از ۳)

MULTILAYER PERCEPTRON

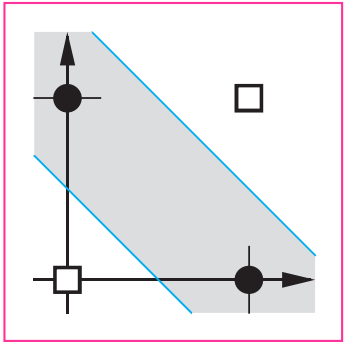




Layer 1/Neuron 1



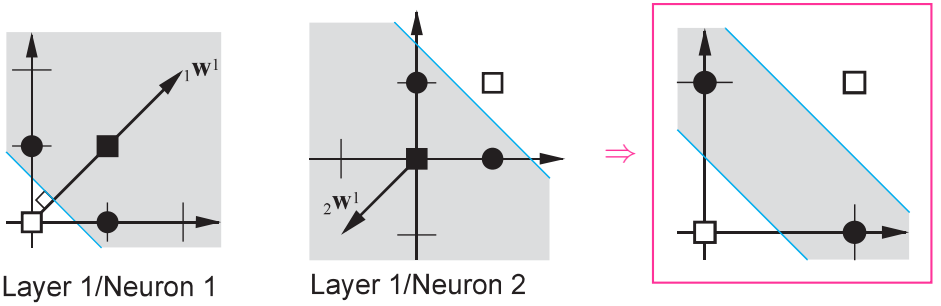
Layer 1/Neuron 2

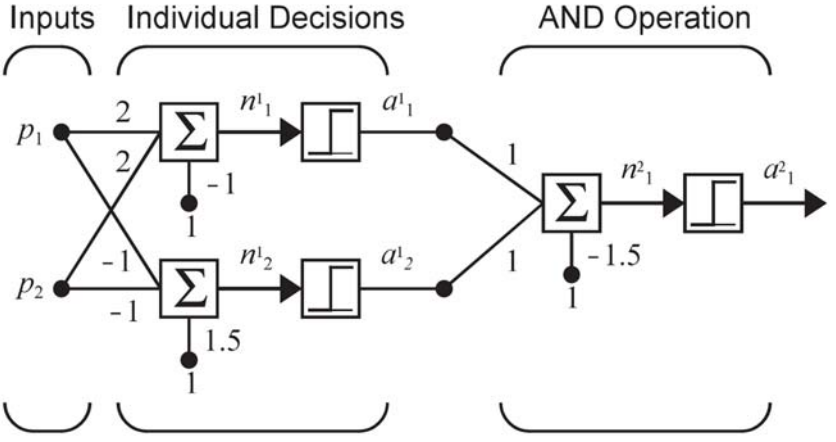


پرسپترون چندلایه

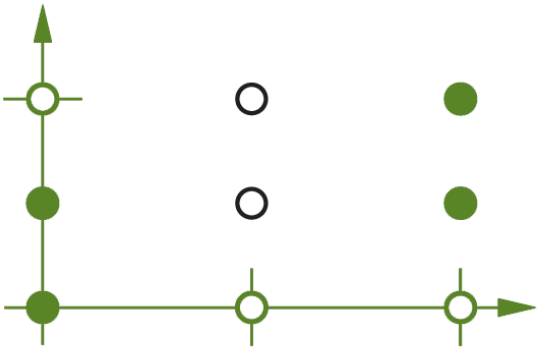
مثال: یای انحصاری (xor) (۳ از ۳)

MULTILAYER PERCEPTRON





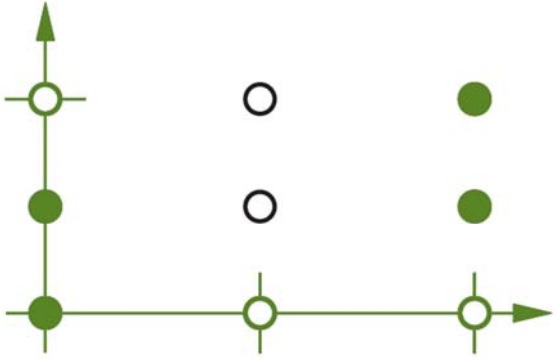
Example



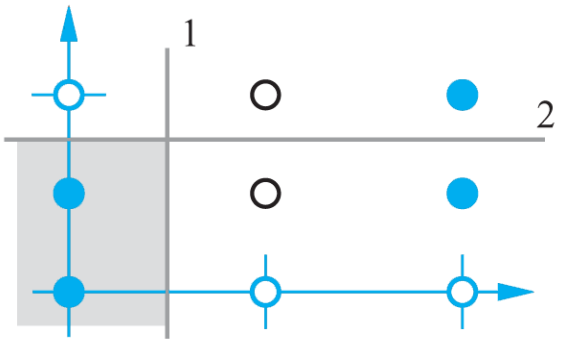
پرسپترون چندلایه

شبکه‌ی سه‌لایه: مثال (۱ از ۴)

MULTILAYER PERCEPTRON



Elementary Decision Boundaries



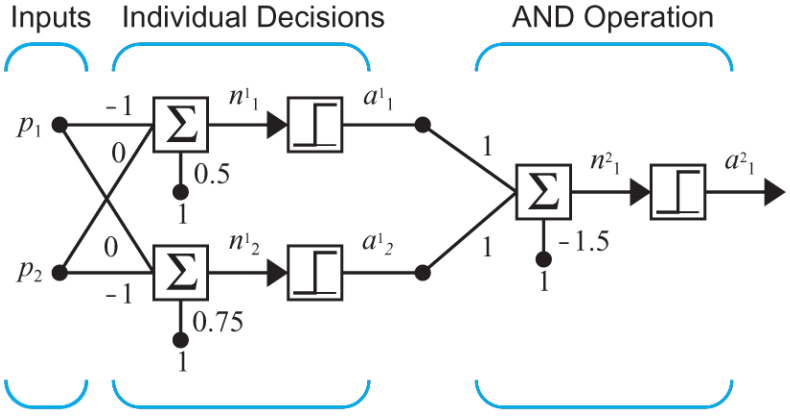
First Boundary:

$$a_1^1 = hardlim(\begin{bmatrix} -1 & 0 \end{bmatrix}\mathbf{p} + 0.5)$$

Second Boundary:

$$a_2^1 = hardlim(\begin{bmatrix} 0 & -1 \end{bmatrix}\mathbf{p} + 0.75)$$

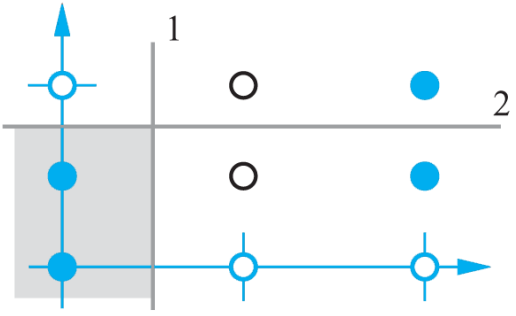
First Subnetwork



پرسپترون چندلایه

شبکه‌ی سه‌لایه: مثال (۲ از ۴): مرزهای تصمیم ابتدایی

ELEMENTARY DECISION BOUNDARIES



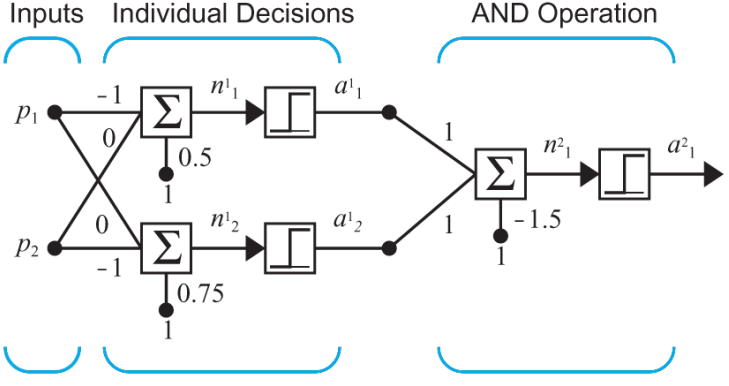
First Boundary:

$$a_1^1 = hardlim(\begin{bmatrix}-1 & 0\end{bmatrix}\mathbf{p} + 0.5)$$

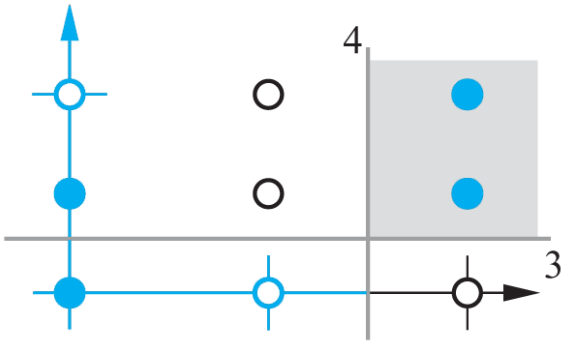
Second Boundary:

$$a_2^1 = hardlim(\begin{bmatrix}0 & -1\end{bmatrix}\mathbf{p} + 0.75)$$

First Subnetwork



Elementary Decision Boundaries



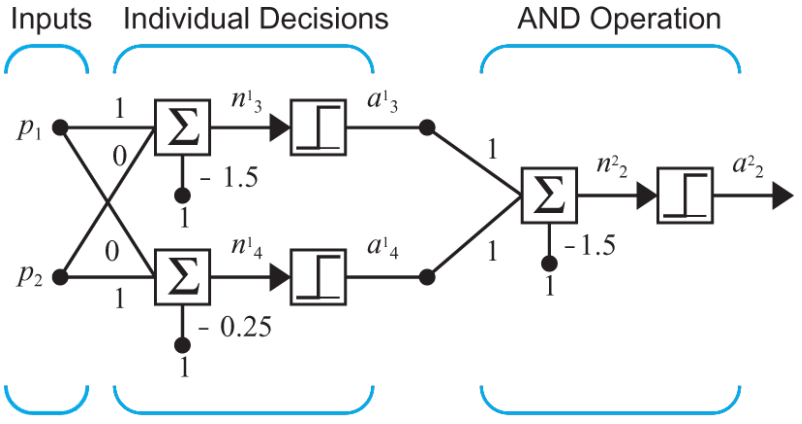
Third Boundary:

$$a_3^1 = hardlim(\begin{bmatrix}1 & 0\end{bmatrix}\mathbf{p} - 1.5)$$

Fourth Boundary:

$$a_4^1 = hardlim(\begin{bmatrix}0 & 1\end{bmatrix}\mathbf{p} - 0.25)$$

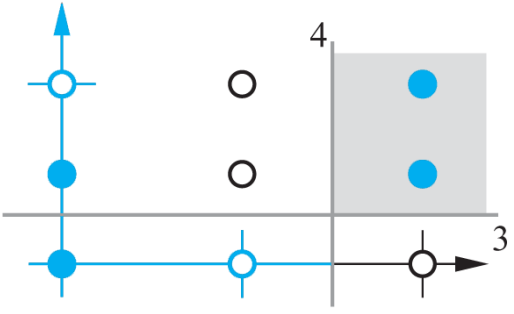
Second Subnetwork



پرسپترون چندلایه

شبکه‌ی سه‌لایه: مثال (۳ از ۴): مرزهای تصمیم ابتدایی

ELEMENTARY DECISION BOUNDARIES



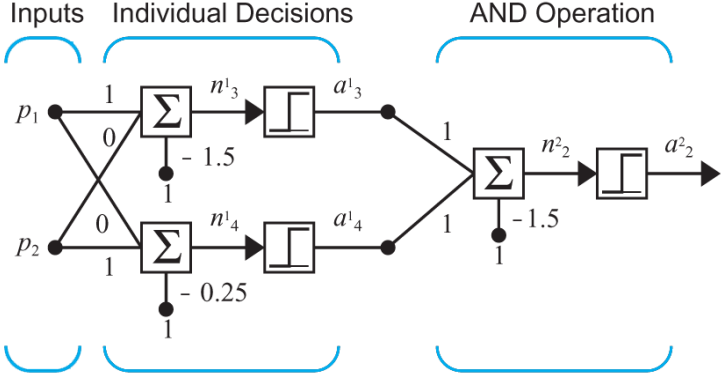
Third Boundary:

$$a_3^1 = hardlim(\begin{bmatrix}1 & 0\end{bmatrix}\mathbf{p} - 1.5)$$

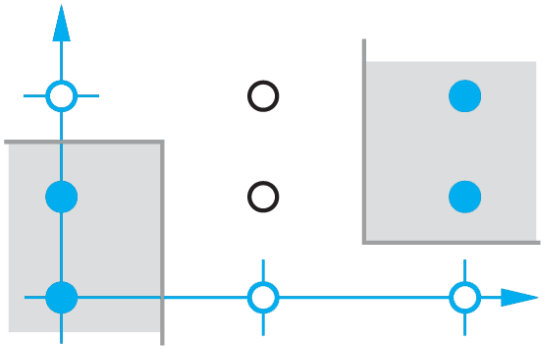
Fourth Boundary:

$$a_4^1 = hardlim(\begin{bmatrix}0 & 1\end{bmatrix}\mathbf{p} - 0.25)$$

Second Subnetwork



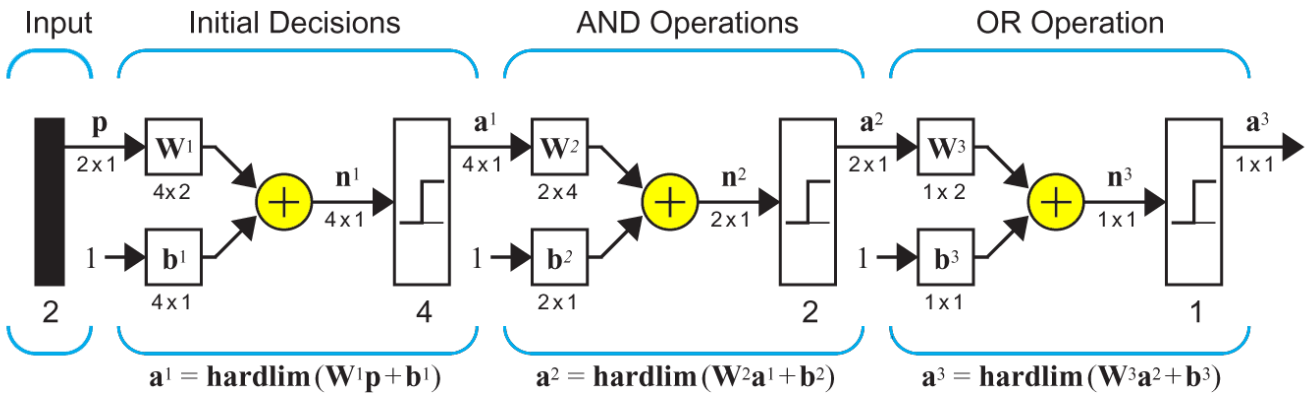
Total Network



$$\mathbf{W}^1 = \begin{bmatrix} -1 & 0 \\ 0 & -1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \mathbf{b}^1 = \begin{bmatrix} 0.5 \\ 0.75 \\ -1.5 \\ -0.25 \end{bmatrix}$$

$$\mathbf{W}^2 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \quad \mathbf{b}^2 = \begin{bmatrix} -1.5 \\ -1.5 \end{bmatrix}$$

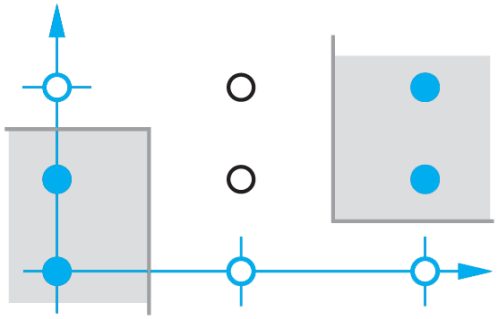
$$\mathbf{W}^3 = \begin{bmatrix} 1 & 1 \end{bmatrix} \quad \mathbf{b}^3 = \begin{bmatrix} -0.5 \end{bmatrix}$$



پرسپترون چندلایه

شبکه‌ی سه‌لایه: مثال (۴ از ۴): شبکه‌ی کامل

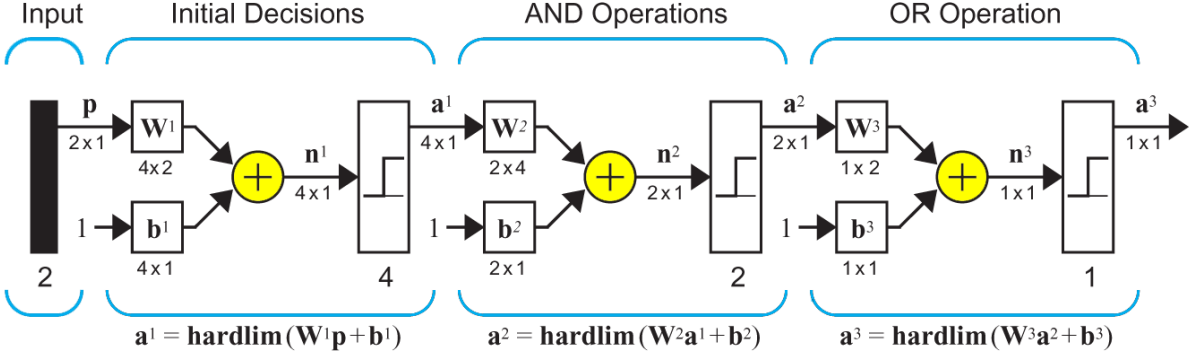
TOTAL NETWORK



$$\mathbf{W}^1 = \begin{bmatrix} -1 & 0 \\ 0 & -1 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \quad \mathbf{b}^1 = \begin{bmatrix} 0.5 \\ 0.75 \\ -1.5 \\ -0.25 \end{bmatrix}$$

$$\mathbf{W}^2 = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 \end{bmatrix} \quad \mathbf{b}^2 = \begin{bmatrix} -1.5 \\ -1.5 \end{bmatrix}$$

$$\mathbf{W}^3 = \begin{bmatrix} 1 & 1 \end{bmatrix} \quad \mathbf{b}^3 = \begin{bmatrix} -0.5 \end{bmatrix}$$



پرسپترون چندلایه

کاربرد تقریب تابع

FUNCTION APPROXIMATION

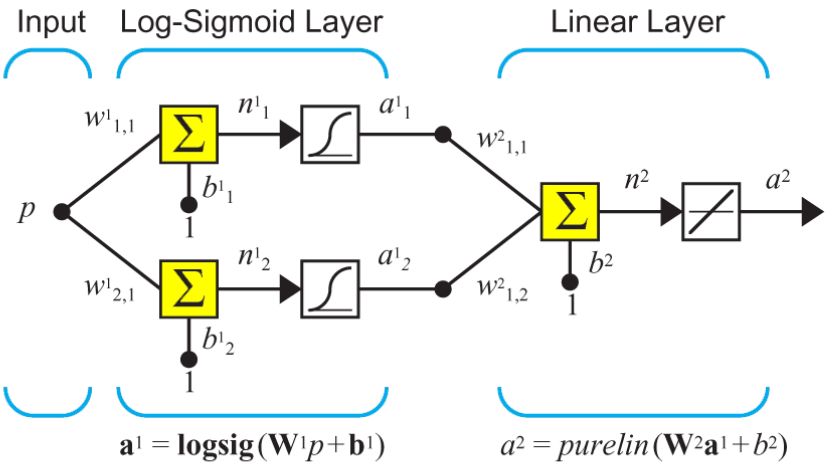
تا اینجا بیشتر از شبکه‌های عصبی به‌عنوان طبقه‌بندی کننده‌ی الگوها استفاده کرده‌ایم.
از اینجا به بعد، به این شبکه‌ها به‌عنوان **تقریب‌زننده‌ی توابع** هم نگاه می‌کنیم

پرسپترون چندلایه، برای پیاده‌سازی توابع انعطاف زیادی دارد.

کاربردها:

- **سیستم‌های کنترل:**
یافتن تابع فیدبک مناسب که خروجی‌های اندازه‌گیری شده را به ورودی‌های کنترلی نگاشت می‌دهد.
- **فیلتر کردن وفقی:**
یافتن نگاشتی از مقادیر تأخیر یافته‌ی سیگنال ورودی به یک سیگنال خروجی مناسب

Function Approximation Example



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

$$f^2(n) = n$$

Nominal Parameter Values

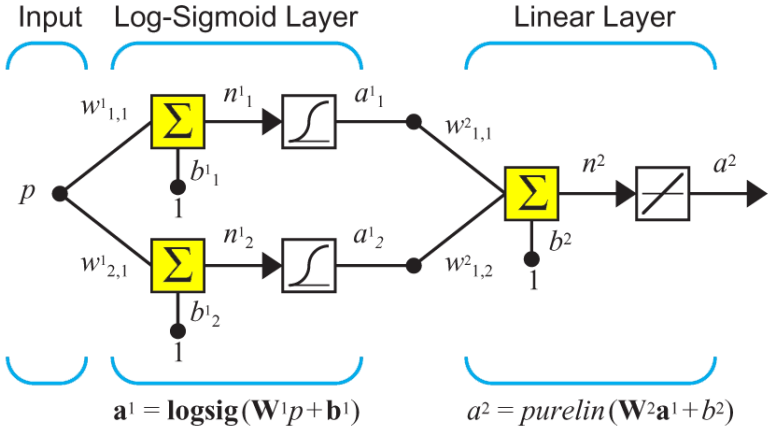
$$w^1_{1,1} = 10 \qquad w^1_{2,1} = 10 \qquad b^1_1 = -10 \qquad b^1_2 = 10$$

$$w^2_{1,1} = 1 \qquad w^2_{1,2} = 1 \qquad b^2 = 0$$

پرسپترون چندلایه

کاربرد تقریب تابع: مثال (۱ از ۳)

FUNCTION APPROXIMATION EXAMPLE



$$f^1(n) = \frac{1}{1+e^{-n}} \qquad \text{logsig}$$

$$f^2(n) = n \qquad \text{purelin}$$

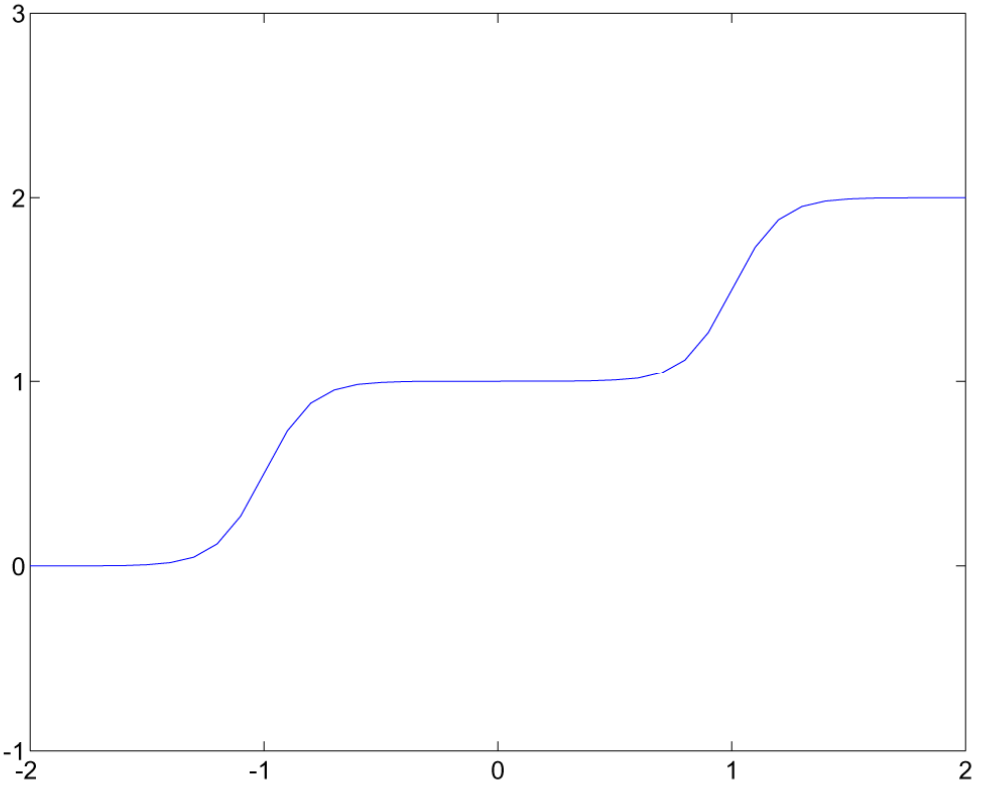
Nominal Parameter Values

$$w^1_{1,1} = 10 \qquad w^1_{2,1} = 10 \qquad b^1_1 = -10 \qquad b^1_2 = 10$$

$$w^2_{1,1} = 1 \qquad w^2_{1,2} = 1 \qquad b^2 = 0$$

11

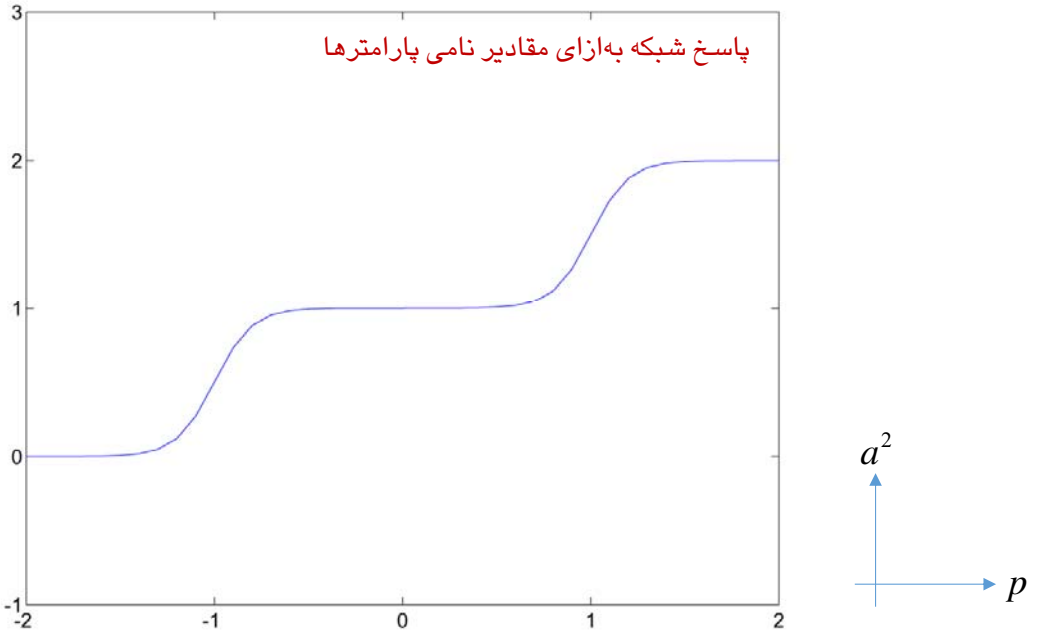
Nominal Response



پرسپترون چندلایه

کاربرد تقریب تابع: مثال (۲ از ۳)

FUNCTION APPROXIMATION EXAMPLE

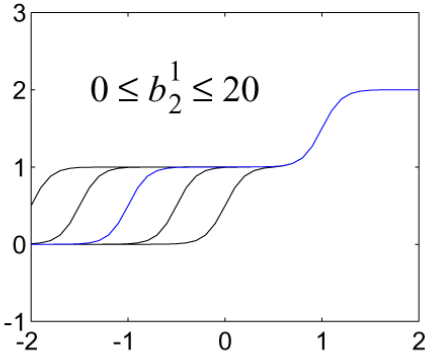


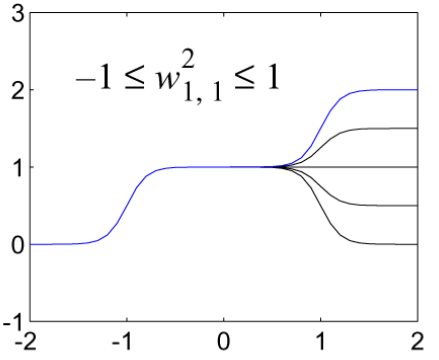
- پاسخ از دو پله تشکیل شده است (برای هر نرون logsig از لایه‌ی ۱، یک پله داریم).
- با تغییر پارامترهای شبکه، شکل و مکان هر پله را تغییر می‌دهیم.
- مرکز هر پله: جایی که ورودی خالص هر نرون لایه‌ی ۱ صفر شود.

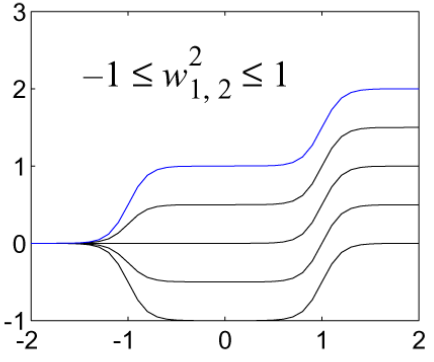
$$n_1^1 = w_{1,1}^1 p + b_1^1 = 0 \Rightarrow p = -b_1^1 / w_{1,1}^1 = -(-10)/10 = 1$$

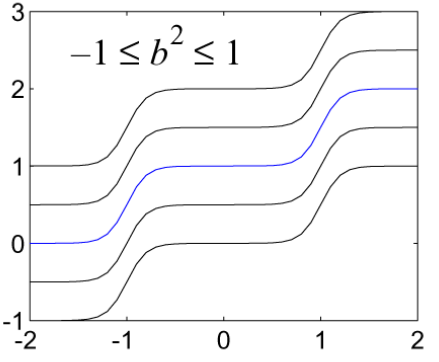
$$n_2^1 = w_{2,1}^1 p + b_2^1 = 0 \Rightarrow p = -b_2^1 / w_{2,1}^1 = -10/10 = -1$$

Parameter Variations







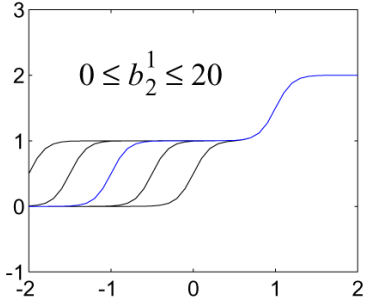


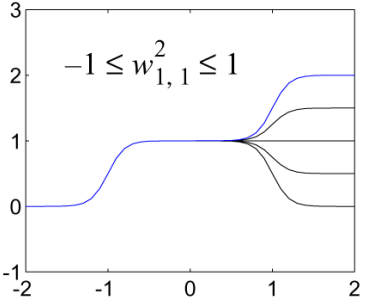
پرسپترون چندلایه

کاربرد تقریب تابع: مثال (۳ از ۳)

FUNCTION APPROXIMATION EXAMPLE

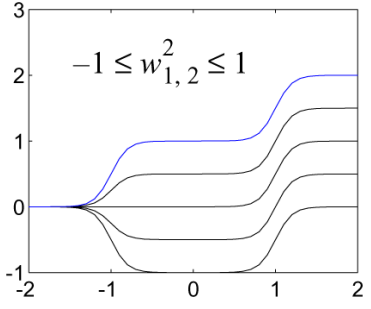
اثر بایاس لایه‌ی ۱

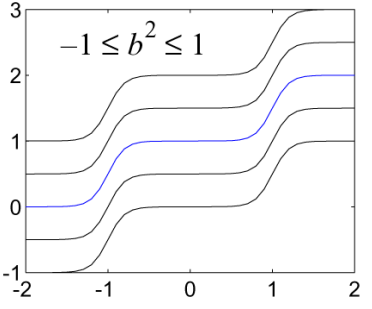




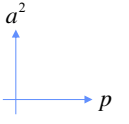
اثر وزن‌ها بر شیب پله‌ها

منحنی آبی با مقادیر نامی رسم شده است. سایر منحنی‌ها با تغییر یک پارامتر در هر مرتبه رسم شده اند.

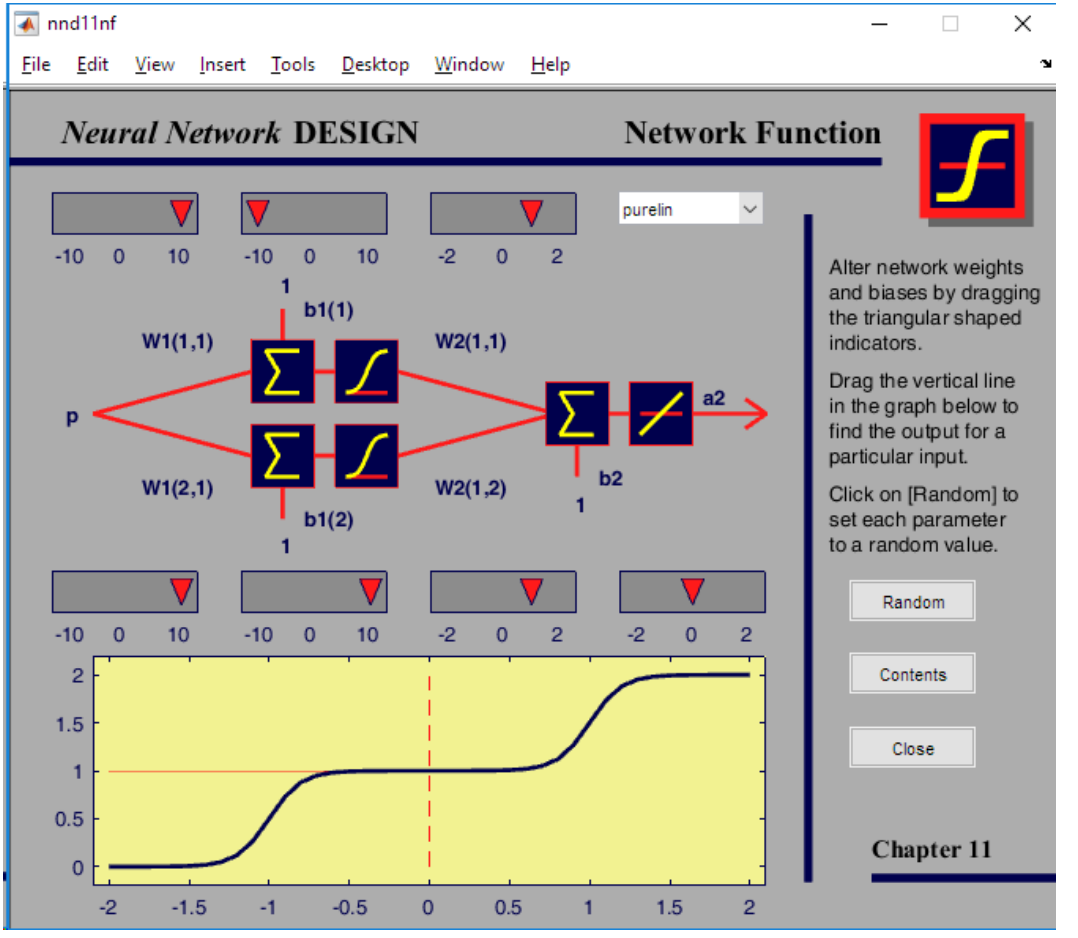




اثر بایاس لایه‌ی ۲ در شیفت خروجی کلی شبکه به بالا یا پایین

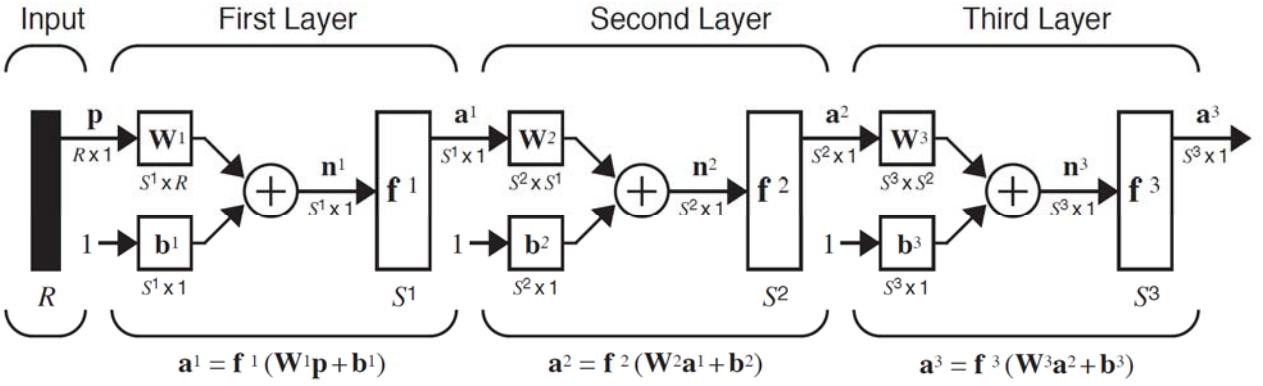


- اگر شبکه تعداد نرون کافی در هر لایه (لایه‌ی پنهان) داشته باشد، می‌تواند تقریباً هر تابعی را تقریب بزند.
- شبکه‌ی دولایه با تابع انتقال سیگموئید در لایه‌ی پنهان و تابع انتقال خطی در لایه‌ی خروجی، می‌تواند مجازاً هر تابع دلخواه با هر درجه دقتی را تقریب بزند، به شرط اینکه تعداد کافی لایه‌ی پنهان وجود داشته باشد.


nnd11nf
File Edit View Insert Tools Desktop Window Help
Neural Network DESIGN
Network Function
purelin
-10 0 10
-10 0 10
-2 0 2
1
b1(1)
W1(1,1)
W2(1,1)
p
a2
W1(2,1)
W2(1,2)
b2
1
b1(2)
1
-10 0 10
-10 0 10
-2 0 2
-2 0 2
2
1.5
1
0.5
0
-2 -1.5 -1 -0.5 0 0.5 1 1.5 2
Alter network weights and biases by dragging the triangular shaped indicators.
Drag the vertical line in the graph below to find the output for a particular input.
Click on [Random] to set each parameter to a random value.
Random
Contents
Close
Chapter 11

```
>> nnd11nf
```

Multilayer Network



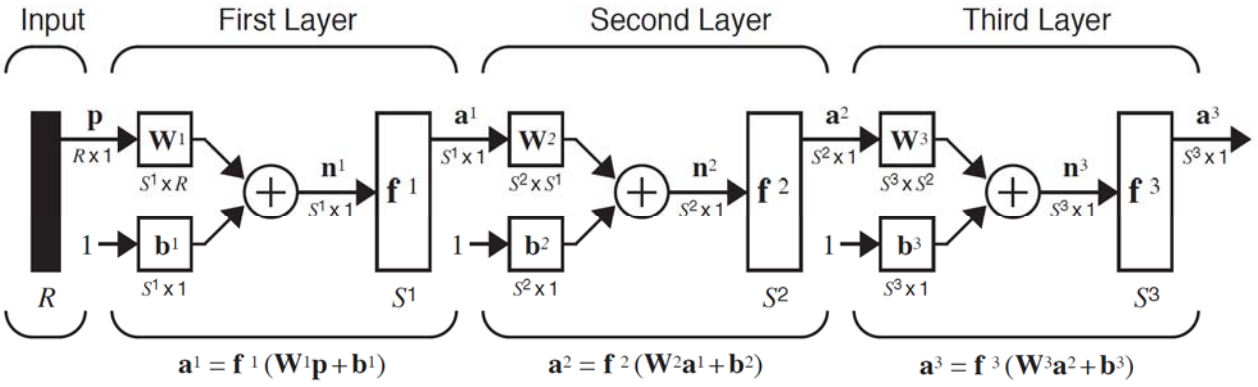
$$\mathbf{a}^{m+1} = \mathbf{f}^{m+1}(\mathbf{W}^{m+1}\mathbf{a}^{m} + \mathbf{b}^{m+1}) \qquad m = 0, 2, \ldots, M-1$$

$$\mathbf{a}^{0} = \mathbf{p}$$

$$\mathbf{a} = \mathbf{a}^{M}$$

شبکه‌ی چندلایه

MULTILAYER NETWORK



$$\mathbf{a}^{m+1} = \mathbf{f}^{m+1}(\mathbf{W}^{m+1}\mathbf{a}^{m} + \mathbf{b}^{m+1}) \qquad m = 0, 2, \ldots, M-1$$

$$\mathbf{a}^{0} = \mathbf{p}$$

$$\mathbf{a} = \mathbf{a}^{M}$$

پس‌انتشار خطا

۲

الگوریتم آموزش: پس‌انتشار

Performance Index

Training Set

$$\{\mathbf{p}_1,\mathbf{t}_1\},\{\mathbf{p}_2,\mathbf{t}_2\},\ldots,\{\mathbf{p}_Q,\mathbf{t}_Q\}$$

Mean Square Error

$$F(\mathbf{x})= E[e^2] = E[(t-a)^2]$$

Vector Case

$$F(\mathbf{x})= E[\mathbf{e}^T\mathbf{e}] = E[(\mathbf{t}-\mathbf{a})^T(\mathbf{t}-\mathbf{a})]$$

Approximate Mean Square Error (Single Sample)

$$\hat{F}(\mathbf{x}) = (\mathbf{t}(k)-\mathbf{a}(k))^T(\mathbf{t}(k)-\mathbf{a}(k)) = \mathbf{e}^T(k)\mathbf{e}(k)$$

Approximate Steepest Descent

$$w^m_{i,j}(k+1) = w^m_{i,j}(k) - \alpha\frac{\partial\hat{F}}{\partial w^m_{i,j}} \qquad b^m_i(k+1) = b^m_i(k) - \alpha\frac{\partial\hat{F}}{\partial b^m_i}$$

الگوریتم آموزش: پس‌انتشار

شاخص کارآیی

PERFORMANCE INDEX

Training Set

$$\{\mathbf{p}_1,\mathbf{t}_1\},\{\mathbf{p}_2,\mathbf{t}_2\},\ldots,\{\mathbf{p}_Q,\mathbf{t}_Q\}$$

Mean Square Error

$$F(\mathbf{x})= E[e^2] = E[(t-a)^2]$$

Vector Case

$$F(\mathbf{x})= E[\mathbf{e}^T\mathbf{e}] = E[(\mathbf{t}-\mathbf{a})^T(\mathbf{t}-\mathbf{a})]$$

Approximate Mean Square Error (Single Sample)

$$\hat{F}(\mathbf{x}) = (\mathbf{t}(k)-\mathbf{a}(k))^T(\mathbf{t}(k)-\mathbf{a}(k)) = \mathbf{e}^T(k)\mathbf{e}(k)$$

Approximate Steepest Descent

$$w^m_{i,j}(k+1) = w^m_{i,j}(k) - \alpha\frac{\partial\hat{F}}{\partial w^m_{i,j}} \qquad b^m_i(k+1) = b^m_i(k) - \alpha\frac{\partial\hat{F}}{\partial b^m_i}$$

Chain Rule

$$\frac{df(n(w))}{dw} = \frac{df(n)}{dn} \times \frac{dn(w)}{dw}$$

Example

$$f(n) = \cos(n) \qquad n = e^{2w} \qquad f(n(w)) = \cos(e^{2w})$$

$$\frac{df(n(w))}{dw} = \frac{df(n)}{dn} \times \frac{dn(w)}{dw} = (-\sin(n))(2e^{2w}) = (-\sin(e^{2w}))(2e^{2w})$$

Application to Gradient Calculation

$$\frac{\partial \hat{F}}{\partial w_{i,j}^{m}} = \frac{\partial \hat{F}}{\partial n_i^m} \times \frac{\partial n_i^m}{\partial w_{i,j}^{m}} \qquad \frac{\partial \hat{F}}{\partial b_i^m} = \frac{\partial \hat{F}}{\partial n_i^m} \times \frac{\partial n_i^m}{\partial b_i^m}$$

الگوریتم آموزش: پس‌انتشار

قاعده‌ی زنجیری

CHAIN RULE

$$\frac{df(n(w))}{dw} = \frac{df(n)}{dn} \times \frac{dn(w)}{dw}$$

Example

$$f(n) = \cos(n) \qquad n = e^{2w} \qquad f(n(w)) = \cos(e^{2w})$$

$$\frac{df(n(w))}{dw} = \frac{df(n)}{dn} \times \frac{dn(w)}{dw} = (-\sin(n))(2e^{2w}) = (-\sin(e^{2w}))(2e^{2w})$$

Application to Gradient Calculation

$$\frac{\partial \hat{F}}{\partial w_{i,j}^{m}} = \frac{\partial \hat{F}}{\partial n_{i}^{m}} \times \frac{\partial n_{i}^{m}}{\partial w_{i,j}^{m}} \qquad \frac{\partial \hat{F}}{\partial b_{i}^{m}} = \frac{\partial \hat{F}}{\partial n_{i}^{m}} \times \frac{\partial n_{i}^{m}}{\partial b_{i}^{m}}$$

جمله‌ی دوم به‌سادگی محاسبه می‌شود زیرا ورودی خالص لایه‌ی m تابع صریح وزن‌ها و بایاس‌های همان لایه است.

Gradient Calculation

$$n_i^m = \sum_{j=1}^{S^{m-1}} w_{i,j}^m a_j^{m-1} + b_i^m$$

$$\frac{\partial n_i^m}{\partial w_{i,j}^m} = a_j^{m-1} \qquad \frac{\partial n_i^m}{\partial b_i^m} = 1$$

Sensitivity

$$s_i^m \equiv \frac{\partial \hat{F}}{\partial n_i^m}$$

Gradient

$$\frac{\partial \hat{F}}{\partial w_{i,j}^m} = s_i^m a_j^{m-1} \qquad \frac{\partial \hat{F}}{\partial b_i^m} = s_i^m$$

الگوریتم آموزش: پس‌انتشار

محاسبه‌ی گرادیان

GRADIENT CALCULATION

$$n_i^m = \sum_{j=1}^{S^{m-1}} w_{i,j}^m a_j^{m-1} + b_i^m$$

$$\frac{\partial n_i^m}{\partial w_{i,j}^m} = a_j^{m-1} \qquad \frac{\partial n_i^m}{\partial b_i^m} = 1$$

Sensitivity

$$s_i^m \equiv \frac{\partial \hat{F}}{\partial n_i^m}$$

حساسیت $\hat{F}$ به تغییرات در iامین عنصر ورودی خالص در لایه‌ی m

Gradient

$$\frac{\partial \hat{F}}{\partial w_{i,j}^m} = s_i^m a_j^{m-1} \qquad \frac{\partial \hat{F}}{\partial b_i^m} = s_i^m$$

Steepest Descent

$$w^m_{i,j}(k+1) = w^m_{i,j}(k) - \alpha s^m_i a^{m-1}_j \qquad b^m_i(k+1) = b^m_i(k) - \alpha s^m_i$$

$$\mathbf{W}^m(k+1) = \mathbf{W}^m(k) - \alpha \mathbf{s}^m (\mathbf{a}^{m-1})^T \qquad \mathbf{b}^m(k+1) = \mathbf{b}^m(k) - \alpha \mathbf{s}^m$$

$$\mathbf{s}^m \equiv \frac{\partial \hat{F}}{\partial \mathbf{n}^m} = \begin{bmatrix} \frac{\partial \hat{F}}{\partial n^m_1} \\ \frac{\partial \hat{F}}{\partial n^m_2} \\ \vdots \\ \frac{\partial \hat{F}}{\partial n^m_{S^m}} \end{bmatrix}$$

Next Step: Compute the Sensitivities (Backpropagation)

الگوریتم آموزش: پس‌انتشار

کاهش تندترین شیب

STEEPEST DESCENT

$$w^m_{i,j}(k+1) = w^m_{i,j}(k) - \alpha s^m_i a^{m-1}_j \qquad b^m_i(k+1) = b^m_i(k) - \alpha s^m_i$$

$$\mathbf{W}^m(k+1) = \mathbf{W}^m(k) - \alpha \mathbf{s}^m (\mathbf{a}^{m-1})^T \qquad \mathbf{b}^m(k+1) = \mathbf{b}^m(k) - \alpha \mathbf{s}^m$$

$$\mathbf{s}^m \equiv \frac{\partial \hat{F}}{\partial \mathbf{n}^m} = \begin{bmatrix} \frac{\partial \hat{F}}{\partial n^m_1} \\ \frac{\partial \hat{F}}{\partial n^m_2} \\ \vdots \\ \frac{\partial \hat{F}}{\partial n^m_{S^m}} \end{bmatrix}$$

Next Step: Compute the Sensitivities (Backpropagation)

Jacobian Matrix

$$\frac{\partial \mathbf{n}^{m+1}}{\partial \mathbf{n}^{m}} \equiv \begin{bmatrix} \frac{\partial n_1^{m+1}}{\partial n_1^m} & \frac{\partial n_1^{m+1}}{\partial n_2^m} & \cdots & \frac{\partial n_1^{m+1}}{\partial n_{S^m}^m} \\ \frac{\partial n_2^{m+1}}{\partial n_1^m} & \frac{\partial n_2^{m+1}}{\partial n_2^m} & \cdots & \frac{\partial n_2^{m+1}}{\partial n_{S^m}^m} \\ \vdots & \vdots & & \vdots \\ \frac{\partial n_{S^{m+1}}^{m+1}}{\partial n_1^m} & \frac{\partial n_{S^{m+1}}^{m+1}}{\partial n_2^m} & \cdots & \frac{\partial n_{S^{m+1}}^{m+1}}{\partial n_{S^m}^m} \end{bmatrix}$$

$$\frac{\partial n_i^{m+1}}{\partial n_j^m} = \frac{\partial \left(\sum_{l=1}^{S^m} w_{i,l}^{m+1} a_l^m + b_i^{m+1} \right)}{\partial n_j^m} = w_{i,j}^{m+1} \frac{\partial a_j^m}{\partial n_j^m}$$

$$\frac{\partial n_i^{m+1}}{\partial n_j^m} = w_{i,j}^{m+1} \frac{\partial f^m(n_j^m)}{\partial n_j^m} = w_{i,j}^{m+1} \dot{f}^m(n_j^m)$$

$$\dot{f}^m(n_j^m) = \frac{\partial f^m(n_j^m)}{\partial n_j^m}$$

$$\frac{\partial \mathbf{n}^{m+1}}{\partial \mathbf{n}^{m}} = \mathbf{W}^{m+1} \dot{\mathbf{F}}^m(\mathbf{n}^m)$$

$$\dot{\mathbf{F}}^m(\mathbf{n}^m) = \begin{bmatrix} \dot{f}^m(n_1^m) & 0 & \cdots & 0 \\ 0 & \dot{f}^m(n_2^m) & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & \dot{f}^m(n_{S^m}^m) \end{bmatrix}$$

الگوریتم آموزش: پس‌انتشار

ماتریس ژاکوبی

JACOBIAN MATRIX

حساسیت لایه‌ی m از روی حساسیت لایه‌ی $(m+1)$ محاسبه می‌شود:
استفاده از ماتریس ژاکوبی ورودی خالص (n) لایه‌ی $(m+1)$ نسبت به n لایه‌ی m

$$\frac{\partial \mathbf{n}^{m+1}}{\partial \mathbf{n}^{m}} \equiv \begin{bmatrix} \frac{\partial n_1^{m+1}}{\partial n_1^m} & \frac{\partial n_1^{m+1}}{\partial n_2^m} & \cdots & \frac{\partial n_1^{m+1}}{\partial n_{S^m}^m} \\ \frac{\partial n_2^{m+1}}{\partial n_1^m} & \frac{\partial n_2^{m+1}}{\partial n_2^m} & \cdots & \frac{\partial n_2^{m+1}}{\partial n_{S^m}^m} \\ \vdots & \vdots & & \vdots \\ \frac{\partial n_{S^{m+1}}^{m+1}}{\partial n_1^m} & \frac{\partial n_{S^{m+1}}^{m+1}}{\partial n_2^m} & \cdots & \frac{\partial n_{S^{m+1}}^{m+1}}{\partial n_{S^m}^m} \end{bmatrix}$$

$$\frac{\partial n_i^{m+1}}{\partial n_j^m} = \frac{\partial \left(\sum_{l=1}^{S^m} w_{i,l}^{m+1} a_l^m + b_i^{m+1} \right)}{\partial n_j^m} = w_{i,j}^{m+1} \frac{\partial a_j^m}{\partial n_j^m}$$

$$a_j^m = f^m(n_j^m)$$

$$\frac{\partial n_i^{m+1}}{\partial n_j^m} = w_{i,j}^{m+1} \frac{\partial f^m(n_j^m)}{\partial n_j^m} = w_{i,j}^{m+1} \dot{f}^m(n_j^m)$$

$$\dot{f}^m(n_j^m) = \frac{\partial f^m(n_j^m)}{\partial n_j^m}$$

$$\frac{\partial \mathbf{n}^{m+1}}{\partial \mathbf{n}^{m}} = \mathbf{W}^{m+1} \dot{\mathbf{F}}^m(\mathbf{n}^m)$$

$$\dot{\mathbf{F}}^m(\mathbf{n}^m) = \begin{bmatrix} \dot{f}^m(n_1^m) & 0 & \cdots & 0 \\ 0 & \dot{f}^m(n_2^m) & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & \dot{f}^m(n_{S^m}^m) \end{bmatrix}$$

Backpropagation (Sensitivities)

$$\mathbf{s}^m = \frac{\partial \hat{F}}{\partial \mathbf{n}^m} = \left(\frac{\partial \mathbf{n}^{m+1}}{\partial \mathbf{n}^m}\right)^T \frac{\partial \hat{F}}{\partial \mathbf{n}^{m+1}} = \dot{\mathbf{F}}^m(\mathbf{n}^m)(\mathbf{W}^{m+1})^T \frac{\partial \hat{F}}{\partial \mathbf{n}^{m+1}}$$

$$\mathbf{s}^m = \dot{\mathbf{F}}^m(\mathbf{n}^m)(\mathbf{W}^{m+1})^T \mathbf{s}^{m+1}$$

The sensitivities are computed by starting at the last layer, and then propagating backwards through the network to the first layer.

$$\mathbf{s}^M \rightarrow \mathbf{s}^{M-1} \rightarrow \ldots \rightarrow \mathbf{s}^2 \rightarrow \mathbf{s}^1$$

الگوریتم آموزش: پس‌انتشار

پس انتشار (حساسیت‌ها)

BACKPROPAGATION (SENSITIVITIES)

$$\mathbf{s}^m = \frac{\partial \hat{F}}{\partial \mathbf{n}^m} = \left(\frac{\partial \mathbf{n}^{m+1}}{\partial \mathbf{n}^m}\right)^T \frac{\partial \hat{F}}{\partial \mathbf{n}^{m+1}} = \dot{\mathbf{F}}^m(\mathbf{n}^m)(\mathbf{W}^{m+1})^T \underbrace{\frac{\partial \hat{F}}{\partial \mathbf{n}^{m+1}}}_{\mathbf{s}^{m+1}}$$

$$\mathbf{s}^m = \dot{\mathbf{F}}^m(\mathbf{n}^m)(\mathbf{W}^{m+1})^T \mathbf{s}^{m+1}$$

The sensitivities are computed by starting at the last layer, and then propagating backwards through the network to the first layer.

حساسیت‌ها با شروع از لایه‌ی آخر محاسبه می‌شوند و سپس در جهت پس‌رو منتشر می‌شوند تا به لایه‌ی اول برسند:

$$\mathbf{s}^M \rightarrow \mathbf{s}^{M-1} \rightarrow \ldots \rightarrow \mathbf{s}^2 \rightarrow \mathbf{s}^1$$

Initialization (Last Layer)

$$s_i^M = \frac{\partial \hat{F}}{\partial n_i^M} = \frac{\partial (\mathbf{t}-\mathbf{a})^T(\mathbf{t}-\mathbf{a})}{\partial n_i^M} = \frac{\partial \sum_{j=1}^{S^M}(t_j - a_j)^2}{\partial n_i^M} = -2(t_i - a_i)\frac{\partial a_i}{\partial n_i^M}$$

$$\frac{\partial a_i}{\partial n_i^M} = \frac{\partial a_i^M}{\partial n_i^M} = \frac{\partial f^M(n_i^M)}{\partial n_i^M} = \dot{f}^M(n_i^M)$$

$$s_i^M = -2(t_i - a_i)\dot{f}^M(n_i^M)$$

$$\mathbf{s}^M = -2\dot{\mathbf{F}}^M(\mathbf{n}^M)(\mathbf{t}-\mathbf{a})$$

الگوریتم آموزش: پس‌انتشار

مقداردهی آغازین (لایه‌ی آخر)

INITIALIZATION (LAST LAYER)

$$s_i^M = \frac{\partial \hat{F}}{\partial n_i^M} = \frac{\partial (\mathbf{t}-\mathbf{a})^T(\mathbf{t}-\mathbf{a})}{\partial n_i^M} = \frac{\partial \sum_{j=1}^{S^M}(t_j-a_j)^2}{\partial n_i^M} = -2(t_i-a_i)\frac{\partial a_i}{\partial n_i^M}$$

$$\frac{\partial a_i}{\partial n_i^M} = \frac{\partial a_i^M}{\partial n_i^M} = \frac{\partial f^M(n_i^M)}{\partial n_i^M} = \dot{f}^M(n_i^M)$$

$$s_i^M = -2(t_i-a_i)\dot{f}^M(n_i^M)$$

$$\boxed{\mathbf{s}^M = -2\dot{\mathbf{F}}^M(\mathbf{n}^M)(\mathbf{t}-\mathbf{a})}$$

Summary

Forward Propagation

$$\mathbf{a}^0 = \mathbf{p}$$

$$\mathbf{a}^{m+1} = \mathbf{f}^{m+1}(\mathbf{W}^{m+1}\mathbf{a}^m + \mathbf{b}^{m+1}) \qquad m = 0, 2, \ldots, M-1$$

$$\mathbf{a} = \mathbf{a}^M$$

Backpropagation

$$\mathbf{s}^M = -2\dot{\mathbf{F}}^M(\mathbf{n}^M)(\mathbf{t} - \mathbf{a})$$

$$\mathbf{s}^m = \dot{\mathbf{F}}^m(\mathbf{n}^m)(\mathbf{W}^{m+1})^T\mathbf{s}^{m+1} \qquad m = M-1, \ldots, 2, 1$$

Weight Update

$$\mathbf{W}^m(k+1) = \mathbf{W}^m(k) - \alpha\mathbf{s}^m(\mathbf{a}^{m-1})^T \qquad \mathbf{b}^m(k+1) = \mathbf{b}^m(k) - \alpha\mathbf{s}^m$$

الگوریتم آموزش: پس‌انتشار

خلاصه

SUMMARY

Forward Propagation

$$\mathbf{a}^0 = \mathbf{p}$$

$$\mathbf{a}^{m+1} = \mathbf{f}^{m+1}(\mathbf{W}^{m+1}\mathbf{a}^m + \mathbf{b}^{m+1}) \qquad m = 0, 2, \ldots, M-1$$

$$\mathbf{a} = \mathbf{a}^M$$

Backpropagation

$$\mathbf{s}^M = -2\dot{\mathbf{F}}^M(\mathbf{n}^M)(\mathbf{t} - \mathbf{a})$$

$$\mathbf{s}^m = \dot{\mathbf{F}}^m(\mathbf{n}^m)(\mathbf{W}^{m+1})^T\mathbf{s}^{m+1} \qquad m = M-1, \ldots, 2, 1$$

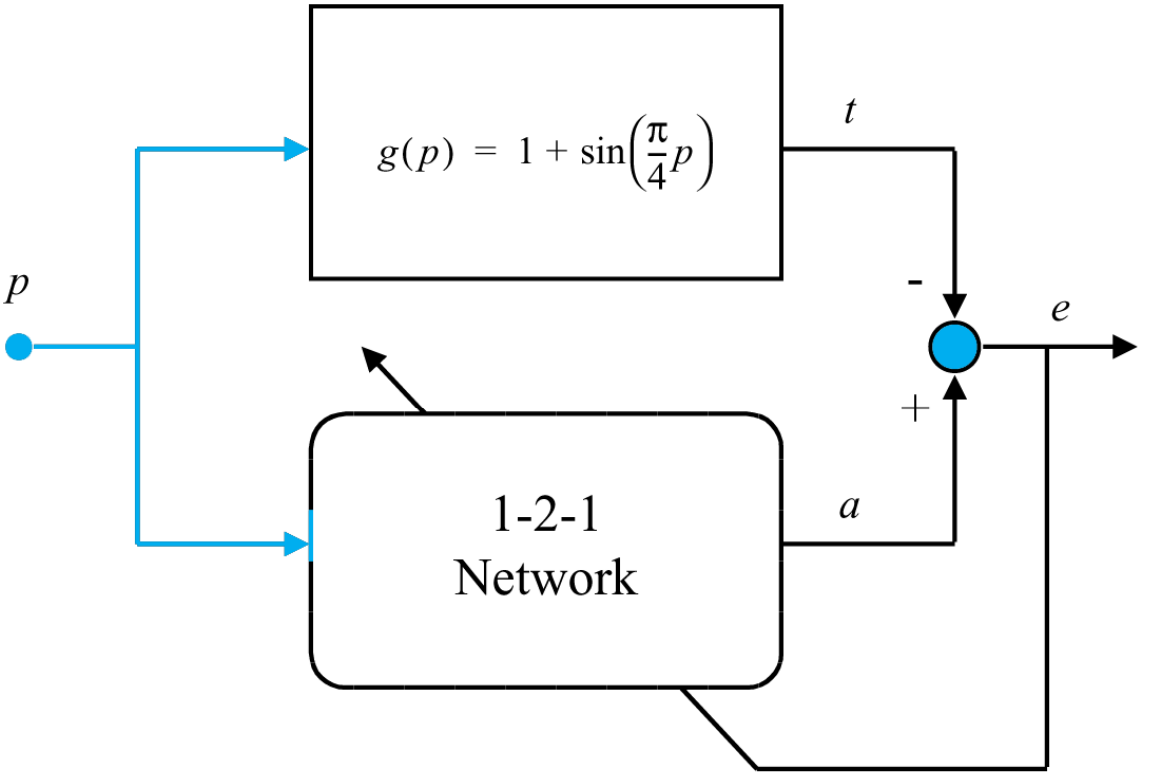
Weight Update

$$\mathbf{W}^m(k+1) = \mathbf{W}^m(k) - \alpha\mathbf{s}^m(\mathbf{a}^{m-1})^T \qquad \mathbf{b}^m(k+1) = \mathbf{b}^m(k) - \alpha\mathbf{s}^m$$

۳

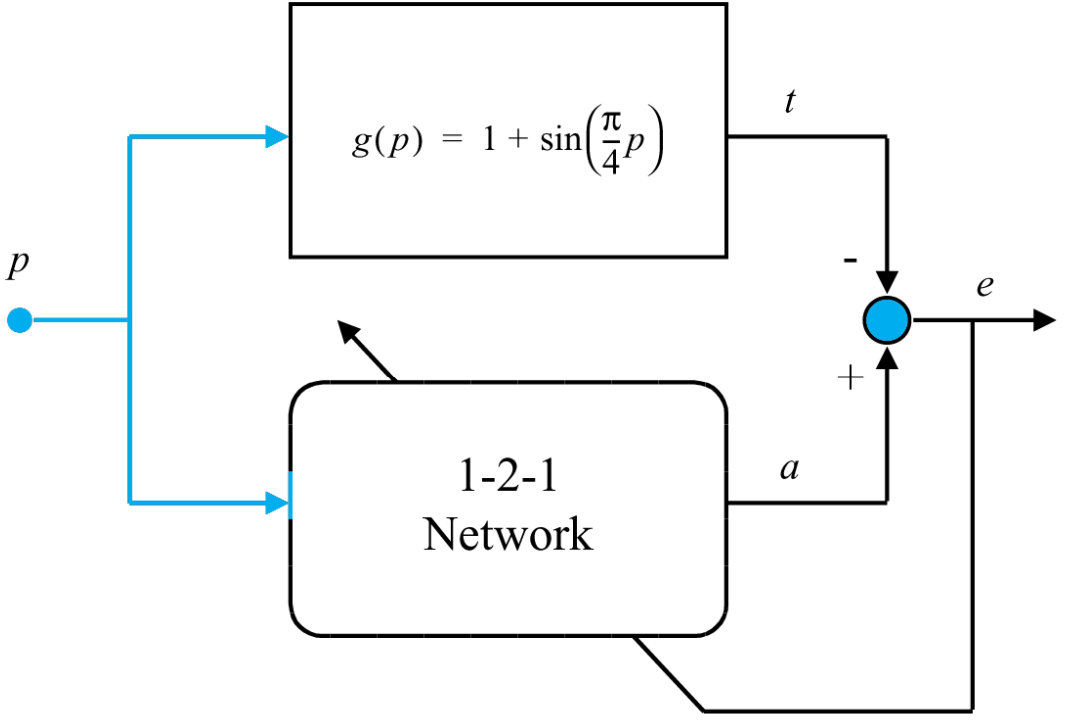
الگوریتم پس‌انتشار: مثال (تقریب تابع)

Example: Function Approximation

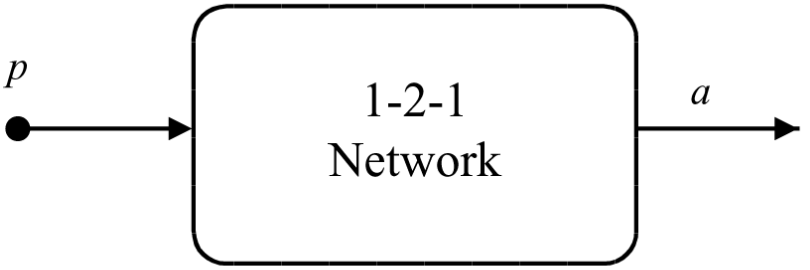


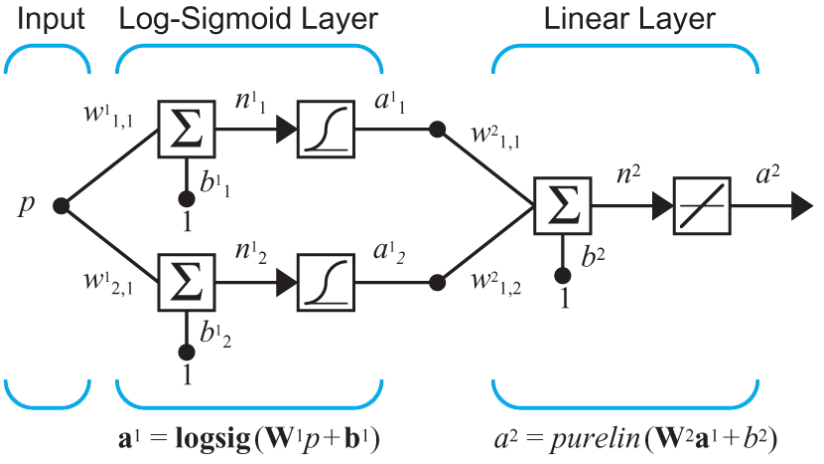
الگوریتم پس‌انتشار: مثال (تقریب تابع)

EXAMPLE: FUNCTION APPROXIMATION



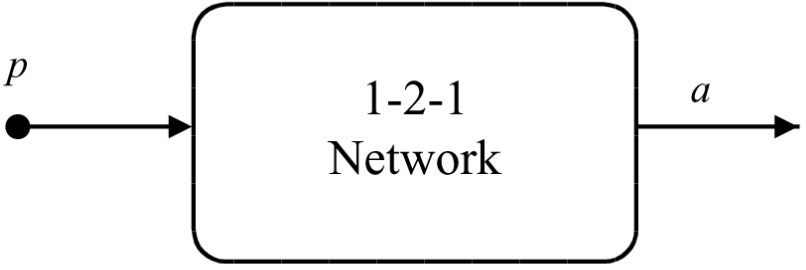
Network

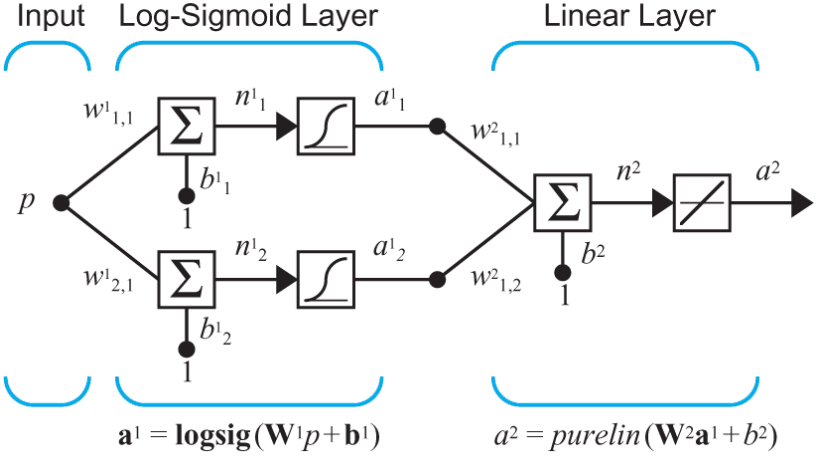

p
1-2-1
Network
a


Input
Log-Sigmoid Layer
Linear Layer
p
w1 1,1
w1 2,1
n1 1
n1 2
a1 1
a1 2
b1 1
b1 2
1
1
w2 1,1
w2 1,2
n2
a2
b2
1
a1 = logsig(W1p+b1)
a2 = purelin(W2a1+b2)

الگوریتم پس‌انتشار: مثال (تقریب تابع)

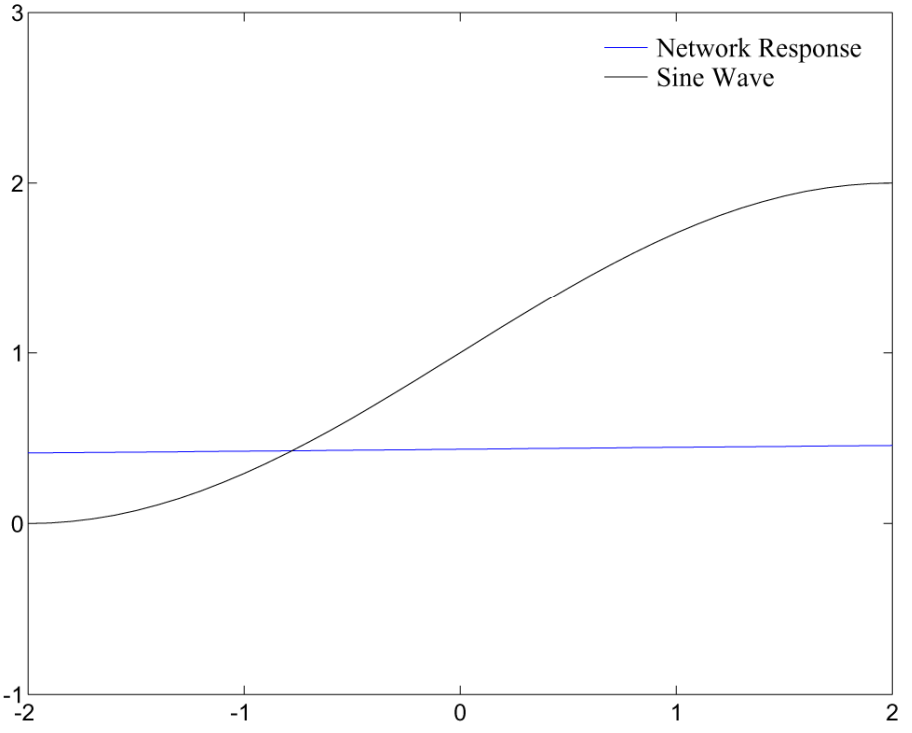
NETWORK





Initial Conditions

$$\mathbf{W}^1(0) = \begin{bmatrix} -0.27 \\ -0.41 \end{bmatrix} \quad \mathbf{b}^1(0) = \begin{bmatrix} -0.48 \\ -0.13 \end{bmatrix} \quad \mathbf{W}^2(0) = \begin{bmatrix} 0.09 & -0.17 \end{bmatrix} \quad \mathbf{b}^2(0) = \begin{bmatrix} 0.48 \end{bmatrix}$$



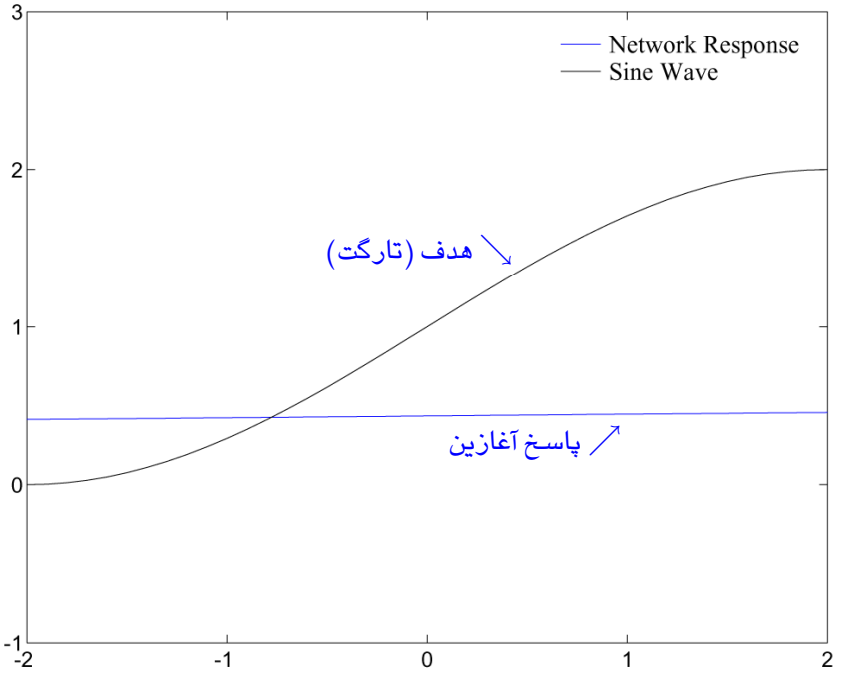
الگوریتم پس‌انتشار: مثال (تقریب تابع)

۳ از ۷ : شرایط آغازین

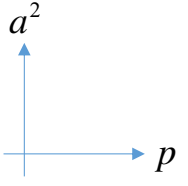
INITIAL CONDITIONS

مقداردهی آغازین با مقادیر کوچک تصادفی

$$\mathbf{W}^1(0) = \begin{bmatrix} -0.27 \\ -0.41 \end{bmatrix} \quad \mathbf{b}^1(0) = \begin{bmatrix} -0.48 \\ -0.13 \end{bmatrix} \quad \mathbf{W}^2(0) = \begin{bmatrix} 0.09 & -0.17 \end{bmatrix} \quad \mathbf{b}^2(0) = \begin{bmatrix} 0.48 \end{bmatrix}$$



داده‌های آموزشی:
۲۱ داده
$\{p_q, t_q\}$
به‌صورت
[-2:0.2:2]



Forward Propagation

$$a^0 = p = 1$$

$$\mathbf{a}^1 = \mathbf{f}^1(\mathbf{W}^1\mathbf{a}^0 + \mathbf{b}^1) = \mathbf{logsig}\left(\begin{bmatrix}-0.27\\-0.41\end{bmatrix}\begin{bmatrix}1\end{bmatrix} + \begin{bmatrix}-0.48\\-0.13\end{bmatrix}\right) = \mathbf{logsig}\left(\begin{bmatrix}-0.75\\-0.54\end{bmatrix}\right)$$

$$\mathbf{a}^1 = \begin{bmatrix}\dfrac{1}{1+e^{0.75}}\\[2ex]\dfrac{1}{1+e^{0.54}}\end{bmatrix} = \begin{bmatrix}0.321\\0.368\end{bmatrix}$$

$$a^2 = \mathrm{f}^2(\mathbf{W}^2\mathbf{a}^1 + \mathbf{b}^2) = purelin\left(\begin{bmatrix}0.09 & -0.17\end{bmatrix}\begin{bmatrix}0.321\\0.368\end{bmatrix} + \begin{bmatrix}0.48\end{bmatrix}\right) = \begin{bmatrix}0.446\end{bmatrix}$$

$$e = t - a = \left\{1 + \sin\left(\frac{\pi}{4}p\right)\right\} - a^2 = \left\{1 + \sin\left(\frac{\pi}{4}1\right)\right\} - 0.446 = 1.261$$

الگوریتم پس‌انتشار: مثال (تقریب تابع)

FORWARD PROPAGATION

$$a^0 = p = 1$$

$$\mathbf{a}^1 = \mathbf{f}^1(\mathbf{W}^1\mathbf{a}^0 + \mathbf{b}^1) = \mathbf{logsig}\left(\begin{bmatrix}-0.27\\-0.41\end{bmatrix}[1] + \begin{bmatrix}-0.48\\-0.13\end{bmatrix}\right) = \mathbf{logsig}\left(\begin{bmatrix}-0.75\\-0.54\end{bmatrix}\right)$$

$$\mathbf{a}^1 = \begin{bmatrix}\frac{1}{1+e^{0.75}}\\ \frac{1}{1+e^{0.54}}\end{bmatrix} = \begin{bmatrix}0.321\\0.368\end{bmatrix}$$

$$a^2 = \mathrm{f}^2(\mathbf{W}^2\mathbf{a}^1 + \mathbf{b}^2) = purelin\left(\begin{bmatrix}0.09 & -0.17\end{bmatrix}\begin{bmatrix}0.321\\0.368\end{bmatrix} + [0.48]\right) = [0.446]$$

$$e = t - a = \left\{1 + \sin\left(\frac{\pi}{4}p\right)\right\} - a^2 = \left\{1 + \sin\left(\frac{\pi}{4}1\right)\right\} - 0.446 = 1.261$$

Transfer Function Derivatives

$$\dot{f}^1(n) = \frac{d}{dn}\left(\frac{1}{1+e^{-n}}\right) = \frac{e^{-n}}{(1+e^{-n})^2} = \left(1-\frac{1}{1+e^{-n}}\right)\left(\frac{1}{1+e^{-n}}\right) = (1-a^1)(a^1)$$

$$\dot{f}^2(n) = \frac{d}{dn}(n) = 1$$

الگوریتم پس‌انتشار: مثال (تقریب تابع)

۵ از ۷ : مشتقات توابع انتقال

TRANSFER FUNCTION DERIVATIVES

$$\dot{f}^1(n) = \frac{d}{dn}\left(\frac{1}{1+e^{-n}}\right) = \frac{e^{-n}}{(1+e^{-n})^2} = \left(1-\frac{1}{1+e^{-n}}\right)\left(\frac{1}{1+e^{-n}}\right) = (1-a^1)(a^1)$$

$$\dot{f}^2(n) = \frac{d}{dn}(n) = 1$$

Backpropagation

$$\mathbf{s}^2 = -2\dot{\mathbf{F}}^2(\mathbf{n}^2)(\mathbf{t}-\mathbf{a}) = -2\left[\dot{f}^2(n^2)\right](1.261) = -2\left[1\right](1.261) = -2.522$$

$$\mathbf{s}^1 = \dot{\mathbf{F}}^1(\mathbf{n}^1)(\mathbf{W}^2)^T\mathbf{s}^2 = \begin{bmatrix}(1-a_1^1)(a_1^1) & 0\\ 0 & (1-a_2^1)(a_2^1)\end{bmatrix}\begin{bmatrix}0.09\\ -0.17\end{bmatrix}\begin{bmatrix}-2.522\end{bmatrix}$$

$$\mathbf{s}^1 = \begin{bmatrix}(1-0.321)(0.321) & 0\\ 0 & (1-0.368)(0.368)\end{bmatrix}\begin{bmatrix}0.09\\ -0.17\end{bmatrix}\begin{bmatrix}-2.522\end{bmatrix}$$

$$\mathbf{s}^1 = \begin{bmatrix}0.218 & 0\\ 0 & 0.233\end{bmatrix}\begin{bmatrix}-0.227\\ 0.429\end{bmatrix} = \begin{bmatrix}-0.0495\\ 0.0997\end{bmatrix}$$

الگوریتم پس‌انتشار: مثال (تقریب تابع)

BACKPROPAGATION

$$\mathbf{s}^2 = -2\dot{\mathbf{F}}^2(\mathbf{n}^2)(\mathbf{t}-\mathbf{a}) = -2\left[\dot{f}^2(n^2)\right](1.261) = -2\left[1\right](1.261) = -2.522$$

$$\mathbf{s}^1 = \dot{\mathbf{F}}^1(\mathbf{n}^1)(\mathbf{W}^2)^T\mathbf{s}^2 = \begin{bmatrix}(1-a_1^1)(a_1^1) & 0\\ 0 & (1-a_2^1)(a_2^1)\end{bmatrix}\begin{bmatrix}0.09\\ -0.17\end{bmatrix}\begin{bmatrix}-2.522\end{bmatrix}$$

$$\mathbf{s}^1 = \begin{bmatrix}(1-0.321)(0.321) & 0\\ 0 & (1-0.368)(0.368)\end{bmatrix}\begin{bmatrix}0.09\\ -0.17\end{bmatrix}\begin{bmatrix}-2.522\end{bmatrix}$$

$$\mathbf{s}^1 = \begin{bmatrix}0.218 & 0\\ 0 & 0.233\end{bmatrix}\begin{bmatrix}-0.227\\ 0.429\end{bmatrix} = \begin{bmatrix}-0.0495\\ 0.0997\end{bmatrix}$$

Weight Update

$$\alpha = 0.1$$

$$\mathbf{W}^2(1) = \mathbf{W}^2(0) - \alpha \mathbf{s}^2 (\mathbf{a}^1)^T = \begin{bmatrix} 0.09 & -0.17 \end{bmatrix} - 0.1 \begin{bmatrix} -2.522 \end{bmatrix} \begin{bmatrix} 0.321 & 0.368 \end{bmatrix}$$

$$\mathbf{W}^2(1) = \begin{bmatrix} 0.171 & -0.0772 \end{bmatrix}$$

$$\mathbf{b}^2(1) = \mathbf{b}^2(0) - \alpha \mathbf{s}^2 = \begin{bmatrix} 0.48 \end{bmatrix} - 0.1 \begin{bmatrix} -2.522 \end{bmatrix} = \begin{bmatrix} 0.732 \end{bmatrix}$$

$$\mathbf{W}^1(1) = \mathbf{W}^1(0) - \alpha \mathbf{s}^1 (\mathbf{a}^0)^T = \begin{bmatrix} -0.27 \\ -0.41 \end{bmatrix} - 0.1 \begin{bmatrix} -0.0495 \\ 0.0997 \end{bmatrix} \begin{bmatrix} 1 \end{bmatrix} = \begin{bmatrix} -0.265 \\ -0.420 \end{bmatrix}$$

$$\mathbf{b}^1(1) = \mathbf{b}^1(0) - \alpha \mathbf{s}^1 = \begin{bmatrix} -0.48 \\ -0.13 \end{bmatrix} - 0.1 \begin{bmatrix} -0.0495 \\ 0.0997 \end{bmatrix} = \begin{bmatrix} -0.475 \\ -0.140 \end{bmatrix}$$

الگوریتم پس‌انتشار: مثال (تقریب تابع)

WEIGHT UPDATE

$$\alpha = 0.1$$

$$\mathbf{W}^2(1) = \mathbf{W}^2(0) - \alpha \mathbf{s}^2 (\mathbf{a}^1)^T = \begin{bmatrix} 0.09 & -0.17 \end{bmatrix} - 0.1 \begin{bmatrix} -2.522 \end{bmatrix} \begin{bmatrix} 0.321 & 0.368 \end{bmatrix}$$

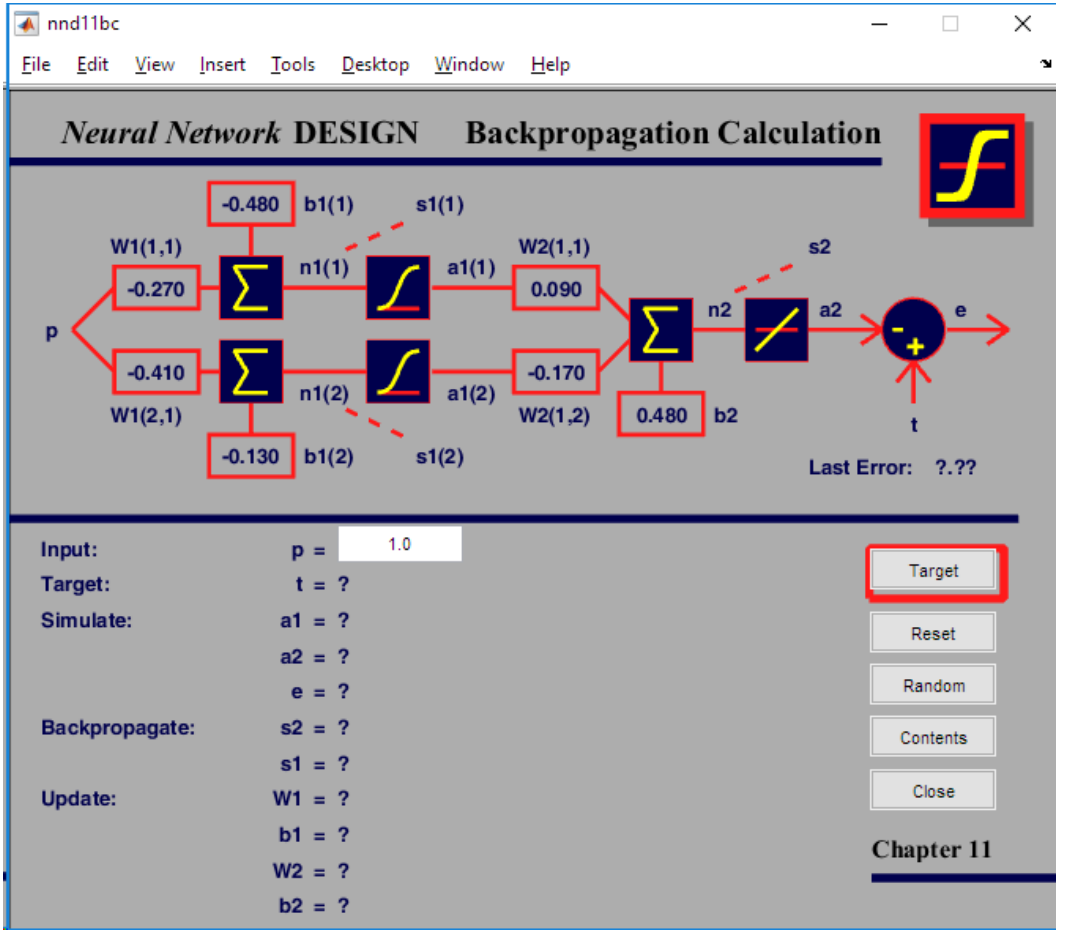
$$\mathbf{W}^2(1) = \begin{bmatrix} 0.171 & -0.0772 \end{bmatrix}$$

$$\mathbf{b}^2(1) = \mathbf{b}^2(0) - \alpha \mathbf{s}^2 = \begin{bmatrix} 0.48 \end{bmatrix} - 0.1 \begin{bmatrix} -2.522 \end{bmatrix} = \begin{bmatrix} 0.732 \end{bmatrix}$$

$$\mathbf{W}^1(1) = \mathbf{W}^1(0) - \alpha \mathbf{s}^1 (\mathbf{a}^0)^T = \begin{bmatrix} -0.27 \\ -0.41 \end{bmatrix} - 0.1 \begin{bmatrix} -0.0495 \\ 0.0997 \end{bmatrix} \begin{bmatrix} 1 \end{bmatrix} = \begin{bmatrix} -0.265 \\ -0.420 \end{bmatrix}$$

$$\mathbf{b}^1(1) = \mathbf{b}^1(0) - \alpha \mathbf{s}^1 = \begin{bmatrix} -0.48 \\ -0.13 \end{bmatrix} - 0.1 \begin{bmatrix} -0.0495 \\ 0.0997 \end{bmatrix} = \begin{bmatrix} -0.475 \\ -0.140 \end{bmatrix}$$

هر پارامتر به میزان سهم خود از خطا تحت تأثیر قرار می‌گیرد.


nnd11bc
File Edit View Insert Tools Desktop Window Help
Neural Network DESIGN Backpropagation Calculation
-0.480
b1(1)
s1(1)
W1(1,1)
-0.270
n1(1)
a1(1)
W2(1,1)
0.090
s2
p
n2
a2
e
-0.410
n1(2)
a1(2)
-0.170
W1(2,1)
W2(1,2)
0.480
b2
t
-0.130
b1(2)
s1(2)
Last Error: ?.??
Input: p = 1.0
Target: t = ?
Simulate: a1 = ?
a2 = ?
e = ?
Backpropagate: s2 = ?
s1 = ?
Update: W1 = ?
b1 = ?
W2 = ?
b2 = ?
Target
Reset
Random
Contents
Close
Chapter 11

```
>> nnd11bc
```

الگوریتم پس‌انتشار

آموزش دسته‌ای در مقابل آموزش افزایشی (۱ از ۲)

BATCH VS. INCREMENTAL TRAINING

می‌توان از محاسبه‌ی گرادیان کامل (پس از ارائه‌ی تمام ورودی‌ها به شبکه) استفاده کرد
و سپس وزن‌ها و بایاس‌ها را به‌هنگام کرد. ⇐ **آموزش دسته‌ای**

The algorithm described above is the stochastic gradient descent algorithm, which involves "on-line" or *incremental training*, in which the network weights and biases are updated after each input is presented (as with the LMS algorithm of Chapter 10). It is also possible to perform *batch training*, in which the complete gradient is computed (after all inputs are applied to the network) before the weights and biases are updated. For example, if each input occurs with equal probability, the mean square error performance index can be written

با فرض احتمال مساوی برای رخداد هر ورودی

$$F(\mathbf{x}) = E[\mathbf{e}^T\mathbf{e}] = E[(\mathbf{t}-\mathbf{a})^T(\mathbf{t}-\mathbf{a})] = \frac{1}{Q}\sum_{q=1}^{Q}(\mathbf{t}_q-\mathbf{a}_q)^T(\mathbf{t}_q-\mathbf{a}_q). \quad (11.49)$$

The total gradient of this performance index is

$$\nabla F(\mathbf{x}) = \nabla\left\{\frac{1}{Q}\sum_{q=1}^{Q}(\mathbf{t}_q-\mathbf{a}_q)^T(\mathbf{t}_q-\mathbf{a}_q)\right\} = \frac{1}{Q}\sum_{q=1}^{Q}\nabla\{(\mathbf{t}_q-\mathbf{a}_q)^T(\mathbf{t}_q-\mathbf{a}_q)\}. \quad (11.50)$$

الگوریتم پس‌انتشار

آموزش دسته‌ای در مقابل آموزش افزایشی (۲ از ۲)

BATCH VS. INCREMENTAL TRAINING

$$\nabla F(\mathbf{x}) = \nabla\left\{\frac{1}{Q}\sum_{q=1}^{Q}(\mathbf{t}_q - \mathbf{a}_q)^T(\mathbf{t}_q - \mathbf{a}_q)\right\} = \frac{1}{Q}\sum_{q=1}^{Q}\nabla\{(\mathbf{t}_q - \mathbf{a}_q)^T(\mathbf{t}_q - \mathbf{a}_q)\}\,. \quad (11.50)$$

Therefore, the total gradient of the mean square error is the mean of the gradients of the individual squared errors. Therefore, to implement a batch version of the backpropagation algorithm, we would step through Eq. (11.41) through Eq. (11.45) for all of the inputs in the training set. Then, the individual gradients would be averaged to get the total gradient. The update equations for the batch steepest descent algorithm would then be

$$\mathbf{W}^m(k+1) = \mathbf{W}^m(k) - \frac{\alpha}{Q}\sum_{q=1}^{Q}\mathbf{s}_q^m(\mathbf{a}_q^{m-1})^T, \quad (11.51)$$

$$\mathbf{b}^m(k+1) = \mathbf{b}^m(k) - \frac{\alpha}{Q}\sum_{q=1}^{Q}\mathbf{s}_q^m. \quad (11.52)$$

چند ملاحظه‌ی در پیاده‌سازی عملی پس‌انتشار

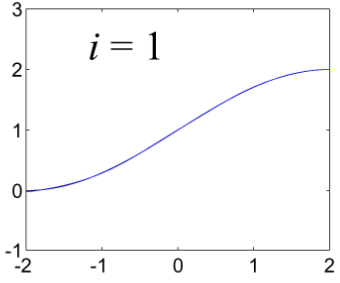
چند ملاحظه‌ی در پیاده‌سازی عملی پس‌انتشار

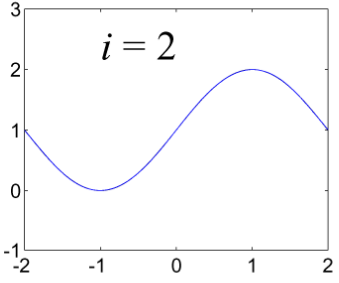
- تعداد لایه‌ها؟
- تعداد نرون‌ها؟
- همگرایی؟
- تعمیم؟

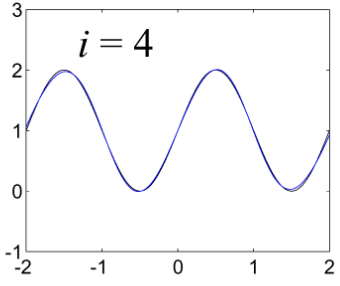
Choice of Architecture

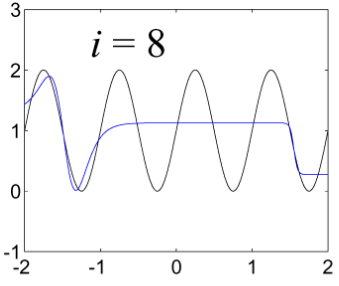
$$g(p) = 1 + \sin\left(\frac{i\pi}{4}p\right)$$

1-3-1 Network







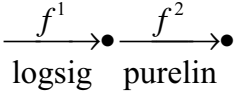


انتخاب معماری

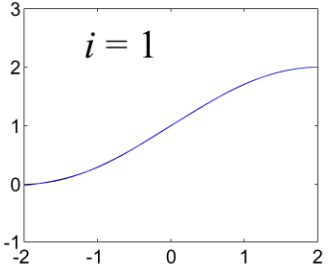
CHOICE OF ARCHITECTURE

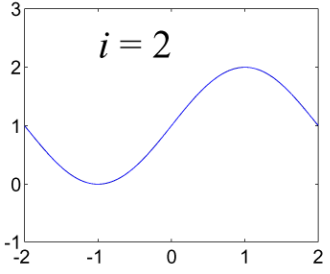
$$g(p) = 1 + \sin\left(\frac{i\pi}{4}p\right) \qquad -2 \le p \le 2$$

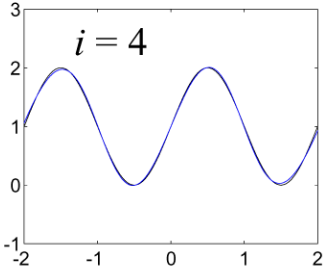
در حالت کلی، تعداد لایه‌ها و تعداد نرون‌های لازم برای رسیدن به کارآیی کافی را نمی‌توان گفت.

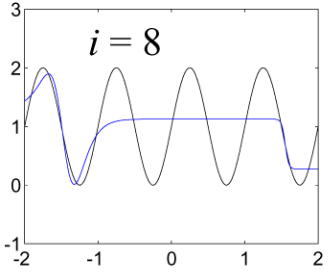


1-3-1 Network





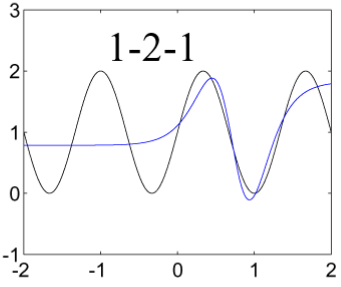


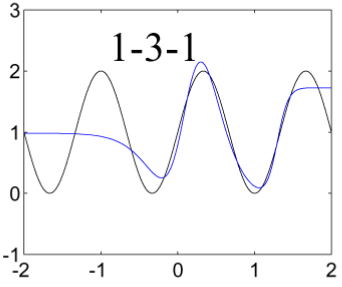


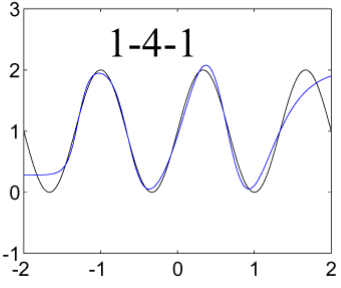
سه پله

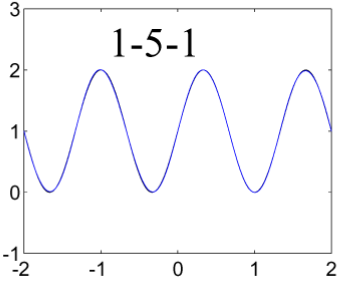
Choice of Network Architecture

$$g(p) = 1 + \sin\left(\frac{6\pi}{4}p\right)$$









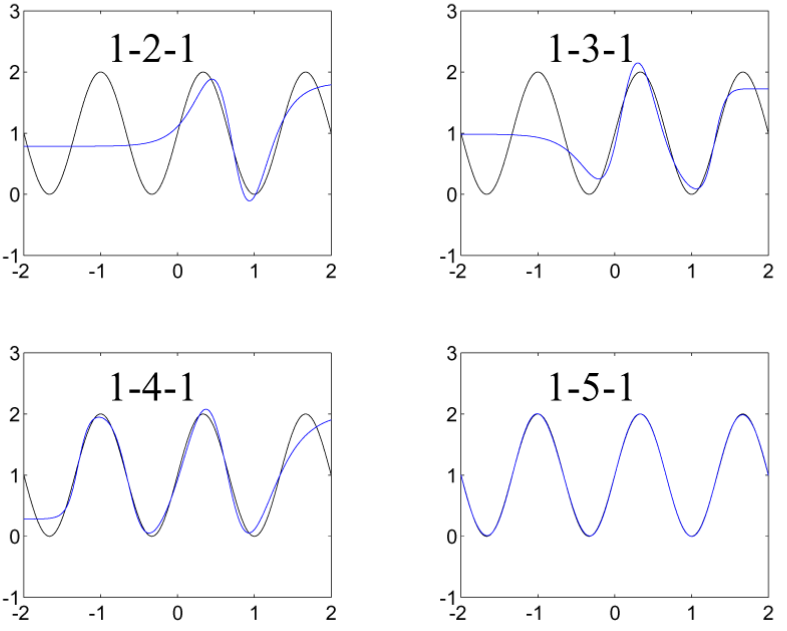
انتخاب معماری شبکه

CHOICE OF NETWORK ARCHITECTURE

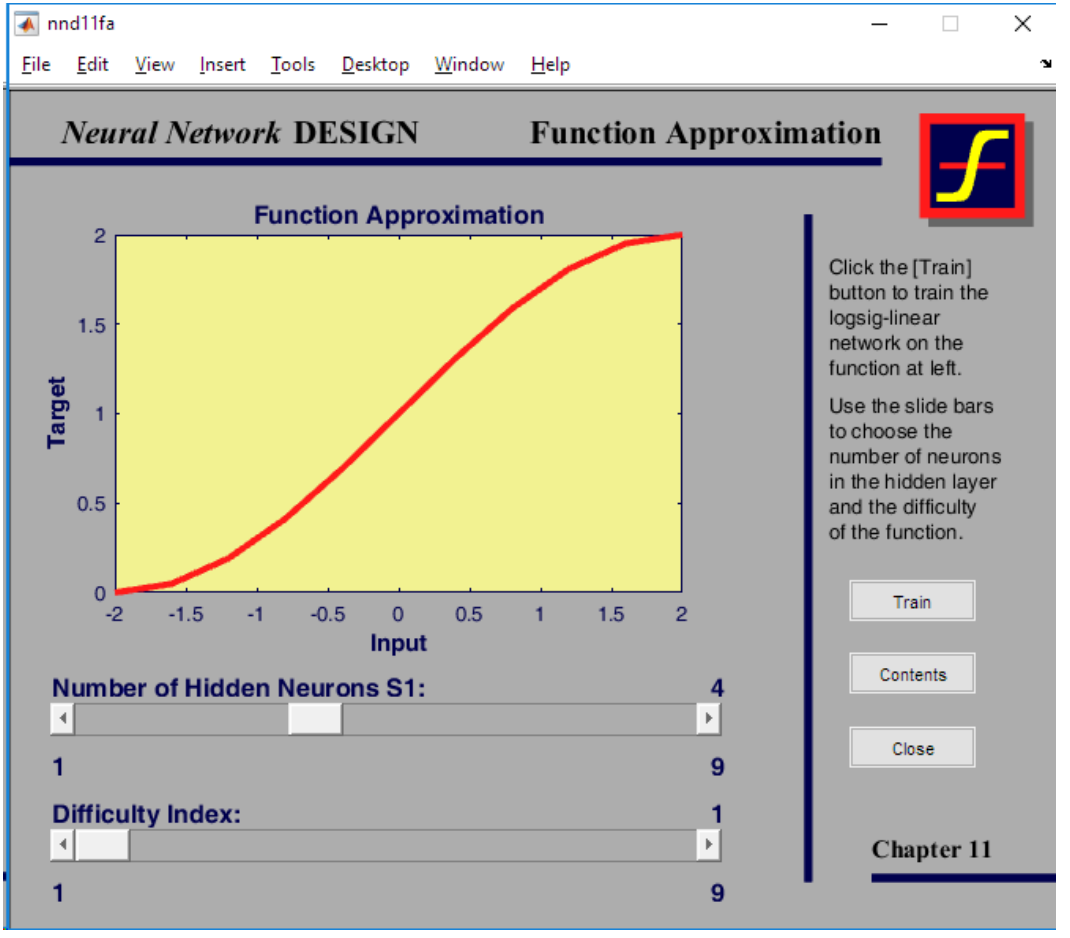
$$g(p) = 1 + \sin\left(\frac{6\pi}{4}p\right) \qquad -2 \le p \le 2$$

مثال قبلی با $i = 6$: تأثیر تعداد نرون‌های لایه‌ی میانی

f^1 logsig → f^2 purelin



هر چه تعداد **نقاط عطف** تابع بیشتر باشد، برای تقریب آن تابع به تعداد نرون بیشتری در لایه‌ی میانی نیاز داریم.

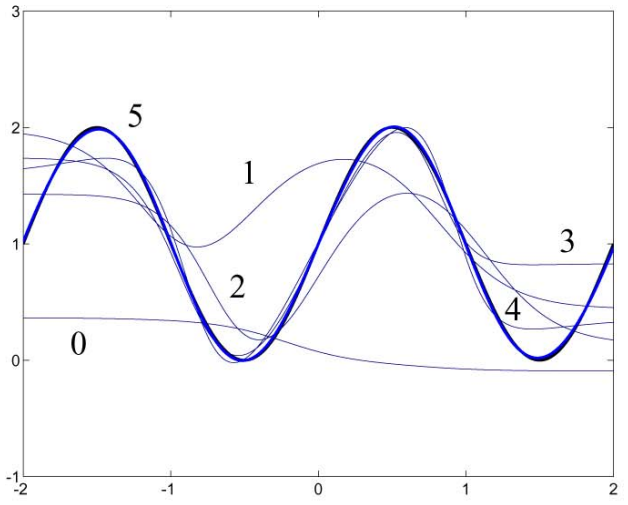


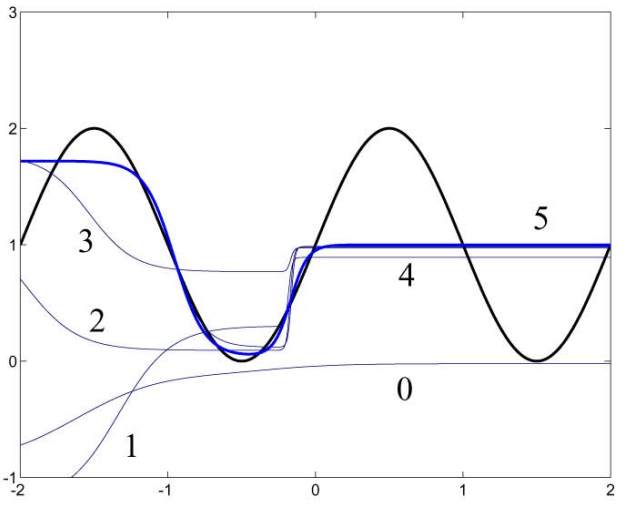
MATLAB

```
>> nnd11fa
```

Convergence

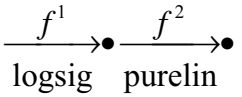
$$g(p) = 1 + \sin(\pi p)$$





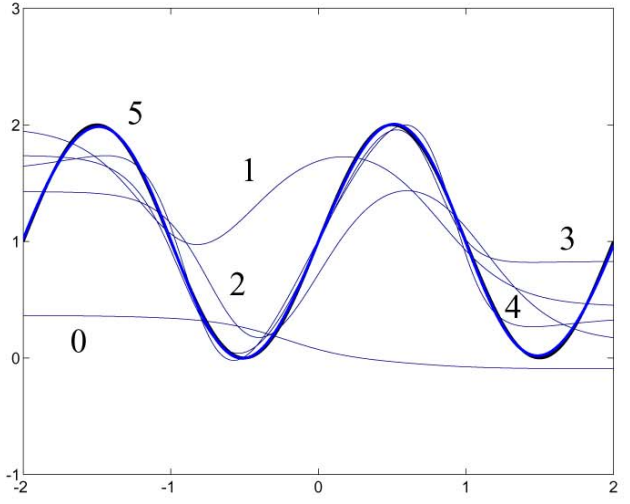
همگرایی

CONVERGENCE

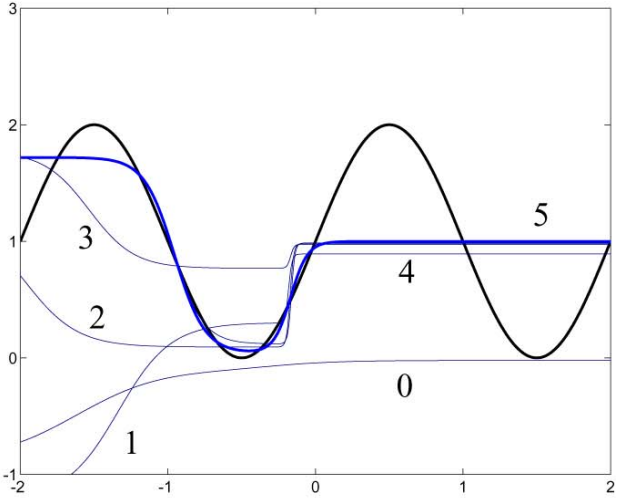


$$g(p) = 1 + \sin(\pi p) \qquad -2 \le p \le 2$$

تفاوت در شرایط اولیه



شرایط برای تکرارهای
0 (حالت اولیه)، 1، 2، 3، 4، 5 (حالت نهایی)



الگوریتم به پاسخی همگرا شده است که خطای MSE را می‌نیمم نمی‌کند.
(الگوریتم به پاسخ می‌نیمم محلی همگرا شده است.)

الگوریتم LMS این مشکل را نداشت. شاخص MSE برای آدالاین، درجه دوم و تحت بیشتر شرایط دارای یک نقطه‌ی می‌نیمم بود.
اما MSE برای شبکه‌ی چندلایه معمولاً پیچیده‌تر است و تعداد زیادی می‌نیمم محلی دارد.
در هنگام همگرایی نمی‌توانیم مطمئن باشیم که به راه‌حل بهینه رسیده‌ایم. (بهترین کار: آزمون چندین شرط اولیه)

تعمیم

GENERALIZATION

شبکه‌های چندلایه در بیشتر موارد با تعداد محدودی نمونه از رفتار مناسب شبکه آموزش داده می‌شود:

$$\{\mathbf{p}_q, \mathbf{t}_q\}$$

مهم این است که شبکه بتواند آنچه دیده و یاد گرفته است را به کل جمعیت تعمیم بدهد.

برای اینکه یک شبکه بتواند تعمیم دهد،
باید تعداد کمتری پارامتر نسبت به تعداد نقاط در مجموعه‌ی آموزشی داشته باشد.

برای تعمیم مناسب، باید اصل مدل‌سازی «تیغه‌ی اوخامی: Ockham's Razor» رعایت شود:

می‌خواهیم ساده‌ترین شبکه‌ای را بیابیم که به‌طور کافی قادر به بازنمایی مجموعه‌ی آموزشی است.

⇓

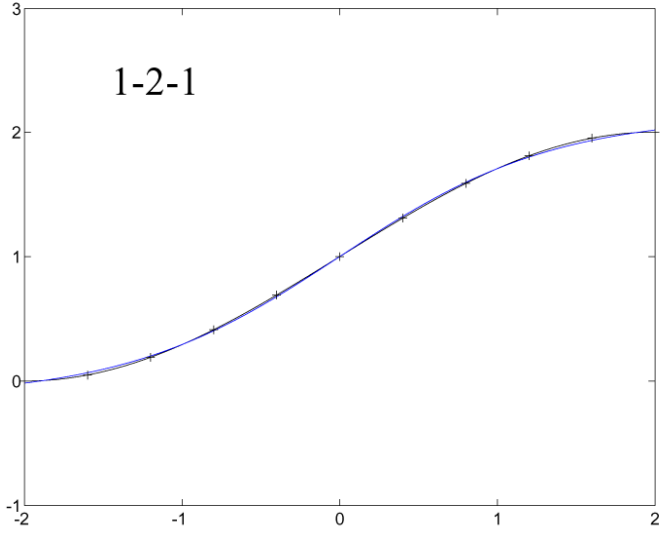
وقتی یک شبکه‌ی کوچک‌تر کار می‌کند، از شبکه‌ی بزرگ‌تر استفاده نکنید.

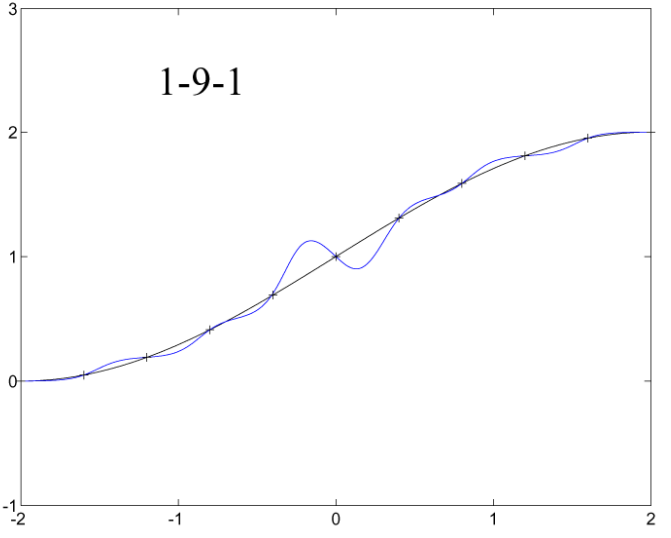
* آموزش را پیش از **بیش‌برازش** (overfitting) شبکه متوقف کنید.

Generalization

$$\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \ldots, \{\mathbf{p}_Q, \mathbf{t}_Q\}$$

$$g(p) = 1 + \sin\left(\frac{\pi}{4}p\right) \qquad p = -2, -1.6, -1.2, \ldots, 1.6, 2$$



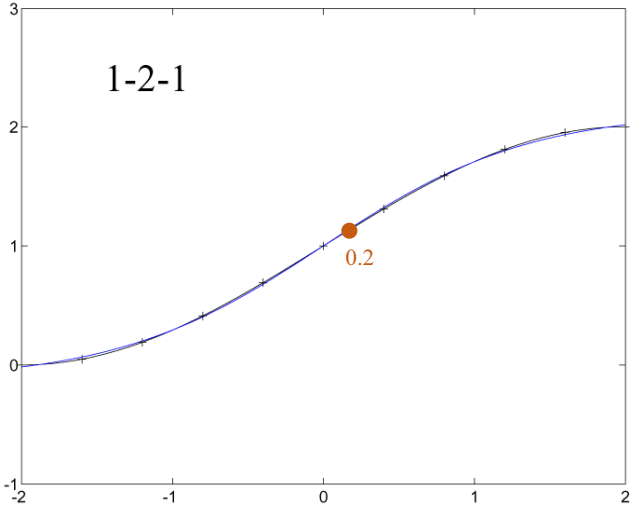


تعمیم

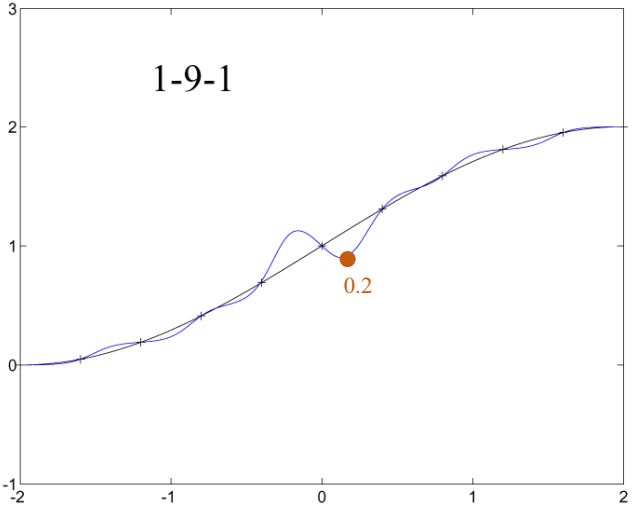
GENERALIZATION

$$\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_2, \mathbf{t}_2\}, \ldots, \{\mathbf{p}_Q, \mathbf{t}_Q\}$$

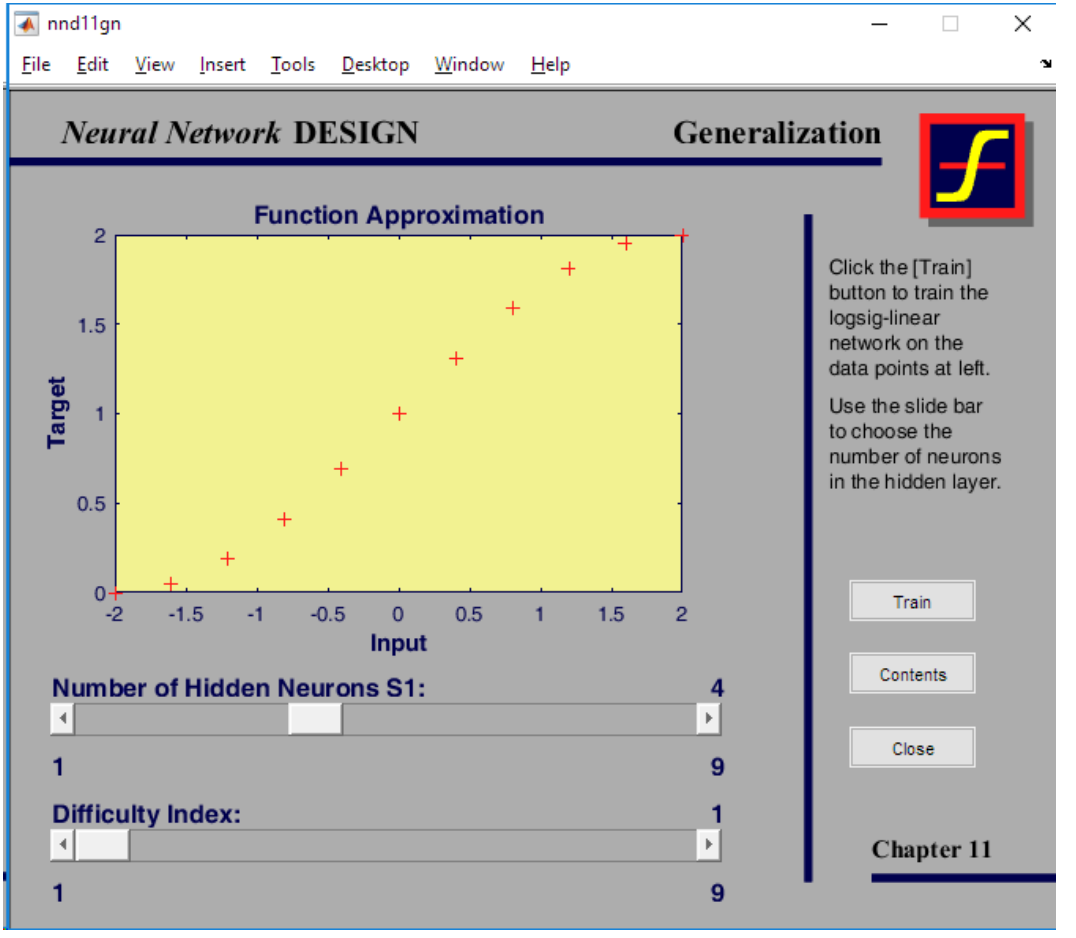
$$g(p) = 1 + \sin\left(\frac{\pi}{4}p\right) \qquad p = -2, -1.6, -1.2, \ldots, 1.6, 2$$



0.2 در مجموعه‌ی آموزشی نیست،
اما پاسخ شبکه برای آن خوب است.



وجود درجات آزادی بیشتر ⇐ تعمیم‌پذیری کمتر



MATLAB

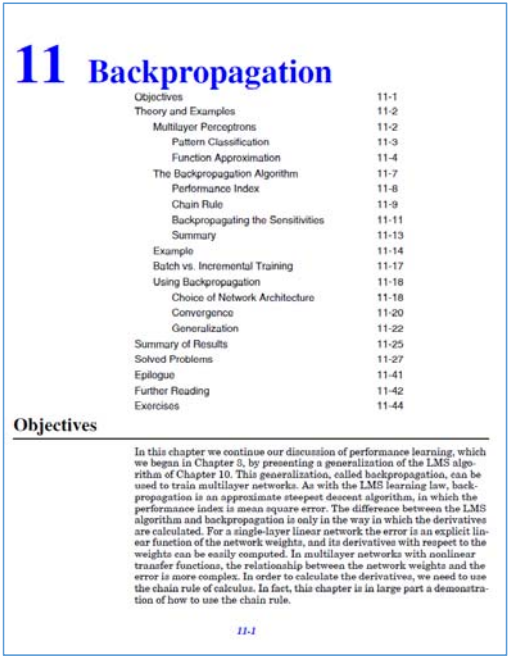
```
>> nnd11gn
```

پس‌انتشار خطا

۴

منابع

منبع اصلی



11 Backpropagation

Objectives 11-1
Theory and Examples 11-2
Multilayer Perceptrons 11-2
Pattern Classification 11-3
Function Approximation 11-4
The Backpropagation Algorithm 11-7
Performance Index 11-8
Chain Rule 11-9
Backpropagating the Sensitivities 11-11
Summary 11-13
Example 11-14
Batch vs. Incremental Training 11-17
Using Backpropagation 11-18
Choice of Network Architecture 11-18
Convergence 11-20
Generalization 11-22
Summary of Results 11-25
Solved Problems 11-27
Epilogue 11-41
Further Reading 11-42
Exercises 11-44

Objectives

In this chapter we continue our discussion of performance learning, which we began in Chapter 8, by presenting a generalization of the LMS algorithm of Chapter 10. This generalization, called backpropagation, can be used to train multilayer networks. As with the LMS learning law, backpropagation is an approximate steepest descent algorithm, in which the performance index is mean square error. The difference between the LMS algorithm and backpropagation is only in the way in which the derivatives are calculated. For a single-layer linear network the error is an explicit linear function of the network weights, and its derivatives with respect to the weights can be easily computed. In multilayer networks with nonlinear transfer functions, the relationship between the network weights and the error is more complex. In order to calculate the derivatives, we need to use the chain rule of calculus. In fact, this chapter is in large part a demonstration of how to use the chain rule.

11-1

Martin T. Hagan, Howard B. Demuth, Mark H. Beale, Orlando De Jesus,
Neural Network Design,
2nd Edition, Martin Hagan, 2014.
Chapter 11

Online version can be downloaded from: http://hagan.okstate.edu/nnd.html