هوش مصنوعی پیشرفته

شبکه‌های عصبی مصنوعی

Artificial Neural Networks

کاظم فولادی

دانشکده مهندسی برق و کامپیوتر

دانشگاه تهران

http://courses.fouladi.ir/aai

هوش مصنوعی پیشرفته

شبکه‌های عصبی مصنوعی

۱

مقدمه

شبکه‌های عصبی مصنوعی

شبکه‌ی عصبی مصنوعی:
مدلی برای استدلال بر اساس مغز انسان

مغز، از یک مجموعه‌ی به‌شدت به‌هم‌پیوسته از سلول‌های عصبی تشکیل شده است.
سلول عصبی = واحد پایه‌ی پردازش اطلاعات = **نرون (neuron)**

مغز انسان تقریباً ۱۰ میلیارد نرون و ۶۰ تریلیون اتصال (سیناپس) بین آنها دارد.

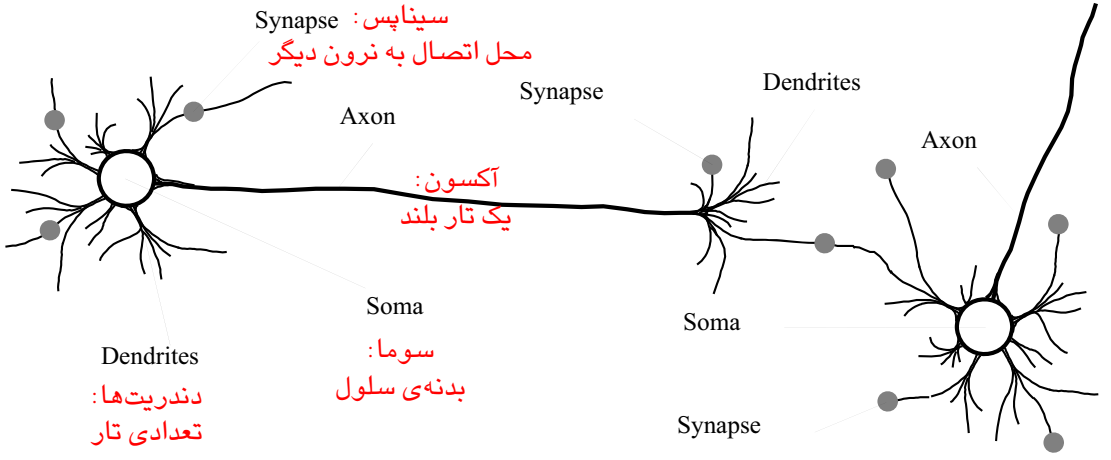
با استفاده‌ی همزمان از چندین نرون،
مغز می‌تواند کارکردهای خود را سریع‌تر از سریع‌ترین کامپیوترهای امروزی انجام دهد.

شبکه‌ی عصبی بیولوژیکی

ساختار نرون

BIOLOGICAL NEURAL NETWORK

هر نرون، ساختار بسیار ساده‌ای دارد،
اما شبکه‌ی چنین عناصری قدرت محاسباتی عظیمی را پدید می‌آورد.



مغز: یک سیستم پردازش موازی و غیرخطی اطلاعات

مغز انسان می‌تواند به‌عنوان یک سیستم بسیار پیچیده، غیرخطی و موازی برای پردازش اطلاعات دیده شود.

اطلاعات به طور همزمان در سراسر کل شبکه‌ی عصبی ذخیره و پردازش می‌شود
(به‌جای محل‌های خاص)
به عبارت دیگر: در شبکه‌های عصبی هم داده‌ها و هم پردازش آنها
سراسری **(global)** است نه محلی **(local)**

یادگیری یک مشخصه‌ی مبنایی و اساسی شبکه‌های عصبی بیولوژیکی است:
از مزایای این ویژگی در شبکه‌های عصبی مصنوعی استفاده می‌شود.

شبکه‌ی عصبی مصنوعی

نرون و اتصالات آن

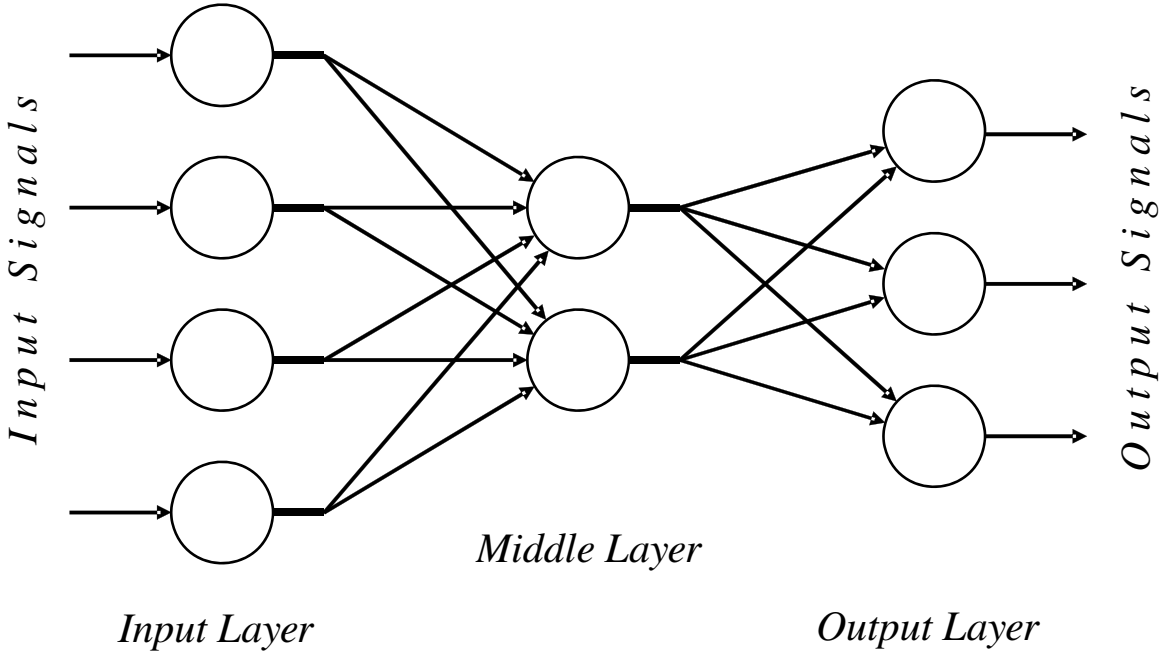
یک شبکه‌ی عصبی مصنوعی،
از تعدادی پردازنده‌ی بسیار ساده به نام **نرون** تشکیل می‌شود.
(مشابه نرون بیولوژیکی در مغز انسان)

نرون‌ها با پیوندهای وزن‌دار به هم متصل می‌شوند.
(برای گذر دادن سیگنال‌ها از یک نرون به دیگری)

سیگنال خروجی از طریق اتصال خروجی نرون انتقال داده می‌شود؛
اتصال خروجی به تعدادی شاخه که اطلاعات یکسانی را انتقال می‌دهند، تقسیم می‌شود؛
انشعابات خروجی در اتصالات ورودی دیگر نرون‌های شبکه پایان می‌یابند.

معماری یک شبکه‌ی عصبی مصنوعی نوعی

ARCHITECTURE OF A TYPICAL ARTIFICIAL NEURAL NETWORK



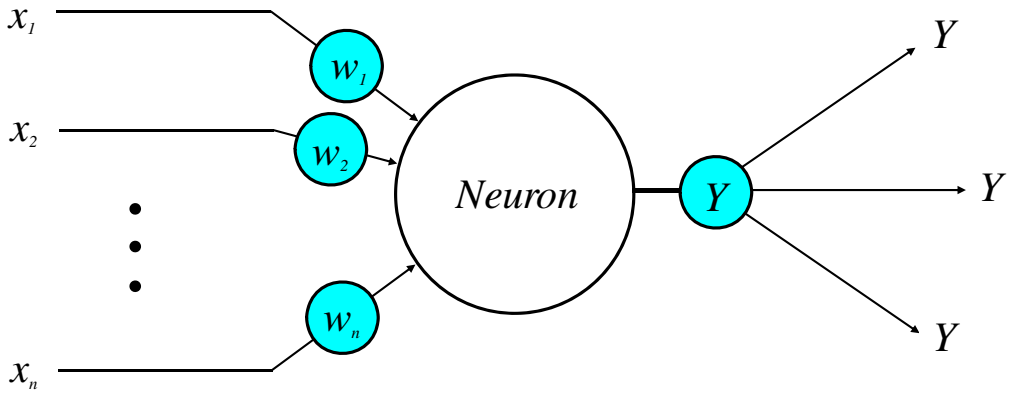
مقایسه‌ی میان شبکه‌ی عصبی بیولوژیکی و شبکه‌ی عصبی مصنوعی

شبکه‌ی عصبی بیولوژیکی *Biological Neural Network*	شبکه‌ی عصبی مصنوعی *Artificial Neural Network*
سوما *Soma*	نرون *Neuron*
دندریت *Dendrite*	ورودی *Input*
آکسون *Axon*	خروجی *Output*
سیناپس *Synapse*	وزن *Weight*

نرون

به‌عنوان یک عنصر محاسباتی ساده

NEURON



نرون

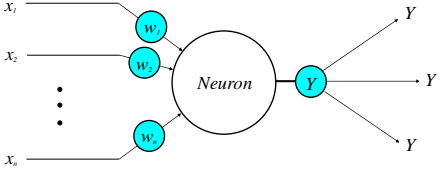
مدل ریاضی

NEURON

نرون، یک مجموع وزن‌دار از سیگنال‌های ورودی را محاسبه می‌کند
و حاصل را با یک **مقدار آستانه** θ مقایسه می‌کند.

- اگر ورودی خالص کوچک‌تر از آستانه بود، خروجی نرون -1 می‌شود.
- اگر ورودی خالص بزرگ‌تر یا مساوی آستانه بود، خروجی نرون $+1$ می‌شود.

$$X = \sum_{i=1}^{n} x_i w_i \qquad Y = \begin{cases} +1, & \text{if } X \geq \theta \\ -1, & \text{if } X < \theta \end{cases}$$

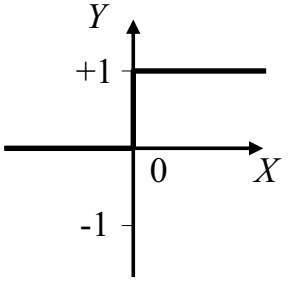
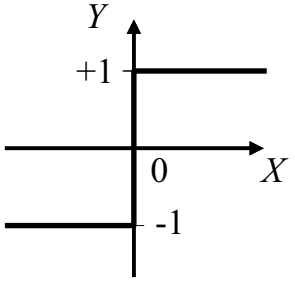
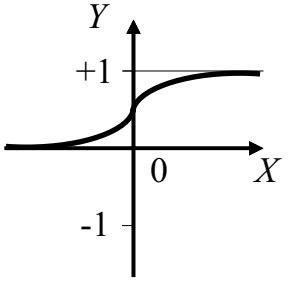
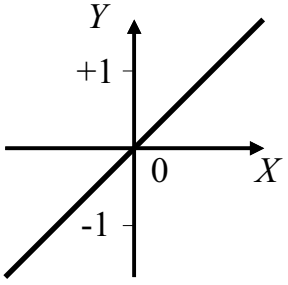


در اینجا نرون از تابع فعال‌سازی از نوع **علامت (sign)** استفاده کرده است.

نرون

تابع فعال‌سازی یک نرون

ACTIVATION FUNCTION

تابع پله	تابع علامت	تابع سیگموئید	تابع خطی
Step function	*Sign function*	*Sigmoid function*	*Linear function*
			
$Y^{step} = \begin{cases} 1, & \text{if } X \geq 0 \\ 0, & \text{if } X < 0 \end{cases}$	$Y^{sign} = \begin{cases} +1, & \text{if } X \geq 0 \\ -1, & \text{if } X < 0 \end{cases}$	$Y^{sigmoid} = \frac{1}{1+e^{-X}}$	$Y^{linear} = X$

هوش مصنوعی پیشرفته

شبکه‌های عصبی مصنوعی

۲

پرسپترون

آیا یک نرون تنها می‌تواند چیزی را یاد بگیرد؟

پرسپترون
Perceptron

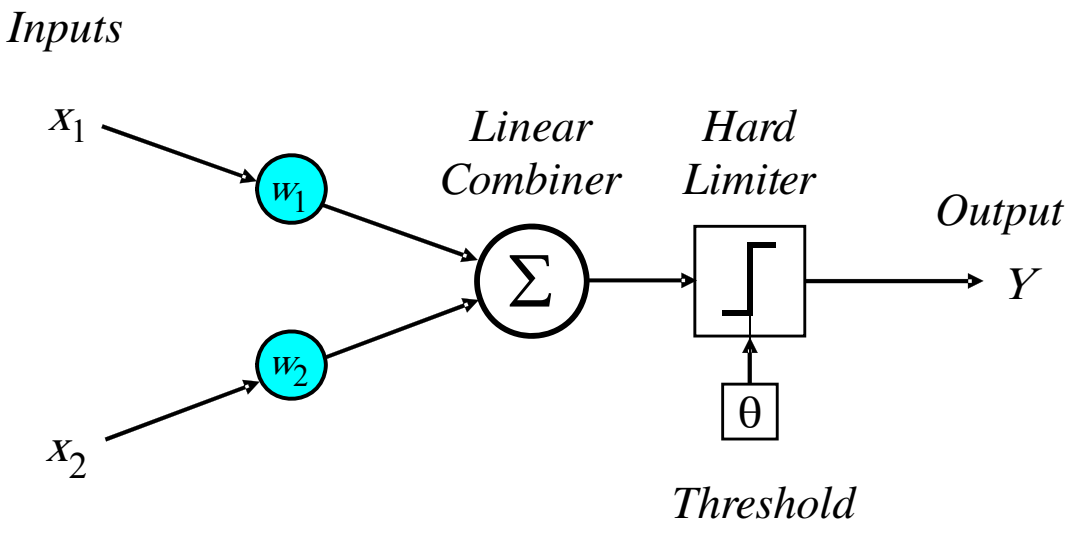
ساده‌ترین شکل از یک شبکه‌ی عصبی:
متشکل از یک نرون واحد با وزن‌های سیناپسی قابل تنظیم
و یک تابع فعال‌سازی Hard Limiter (مثل پله یا علامت)

فرانک روزنبلات (۱۹۵۸)، اولین الگوریتم آموزش شبکه‌ی عصبی
را برای پرسپترون ارائه کرد.

پرسپترون تک-لایه، دو-ورودی

با یک نرون

SINGLE-LAYER TWO-INPUT PERCEPTRON



پرسپترون

THE PERCEPTRON

عملکرد پرسپترون بر اساس مدل نرون مک‌کلوچ و پیتز است:
این مدل متشکل است از از یک ترکیب‌گر خطی که به‌دنبال آن یک hard limiter می‌آید:
مجموع وزن‌دار ورودی به hard limiter اعمال می‌شود:

- خروجی نرون -1 می‌شود اگر ورودی hard limiter منفی باشد.
- خروجی نرون $+1$ می‌شود اگر ورودی hard limiter مثبت (یا صفر) باشد.

پرسپترون

وظیفه‌ی طبقه‌بندی

PERCEPTRON: CLASSIFICATION TASK

هدف پرسپترون، طبقه‌بندی ورودی n بعدی
$(x_1, x_2, \ldots, x_n)$
به یکی از دو طبقه (class)
A_1, A_2
است.

برای یک پرسپترون ابتدائی،
فضای n–بعدی توسط یک ابرصفحه (hyperplan) به دوناحیه‌ی تصمیم‌گیری تقسیم می‌شود:

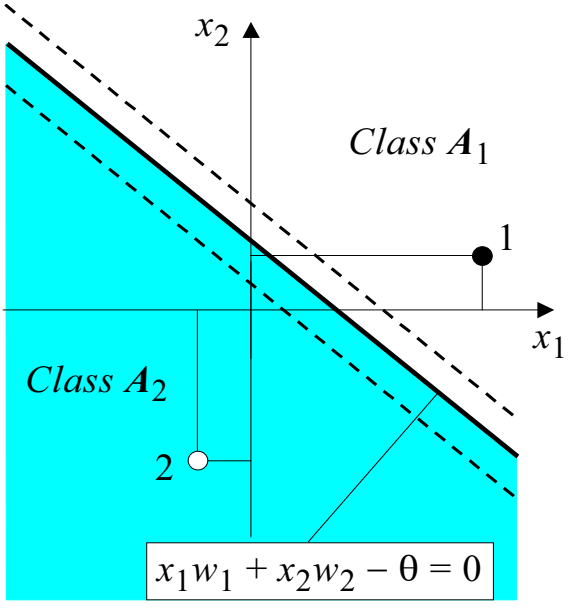
$$\sum_{i=1}^{n} x_i w_i - \theta = 0$$

ابرصفحه توسط **تابع جداپذیر خطی** فوق تعریف می‌شود.

پرسپترون

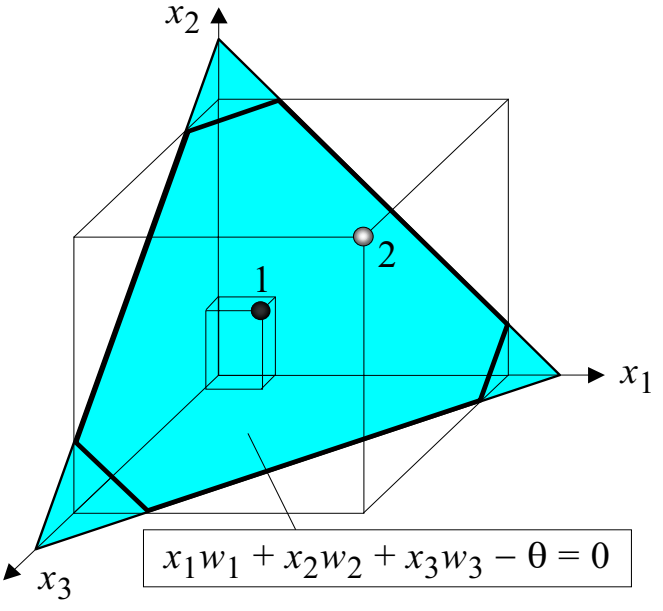
جداپذیری خطی در پرسپترون‌ها

LINEAR SEPARABILITY IN THE PERCEPTRONS



(*a*) Two-input perceptron.

پرسپترون دو–ورودی



(*b*) Three-input perceptron.

پرسپترون سه–ورودی

پرسپترون

یادگیری وظیفه‌ی طبقه‌بندی توسط پرسپترون

برای یادگیری طبقه‌بندی:
تنظیم‌های کوچکی در وزن‌ها انجام می‌شود
تا تفاضل بین خروجی‌های واقعی و خروجی‌های مطلوب پرسپترون کاهش یابد:
وزن‌های آغازین به صورت تصادفی (معمولاً در بازه‌ی $[-0.5, 0.5]$) نسبت‌دهی می‌شوند
و سپس وزن‌ها به‌هنگام می‌شوند تا به خروجی سازگار با نمونه‌های آموزشی دست پیدا کنیم.

پرسپترون

خطای یادگیری در هر تکرار و نقش آن در یادگیری

اگر در تکرار p
خروجی واقعی $Y(p)$ و خروجی مطلوب $Y_d(p)$ باشد،
خطا به صورت زیر به‌دست می‌آید:

$$e(p) = Y_d(p) - Y(p)$$

$$p = 1, 2, 3, \ldots$$

در تکرار p مثال p–ام به پرسپترون نشان داده می‌شود.

- اگر خطا $e(p)$ مثبت بود، باید خروجی پرسپترون $Y(p)$ را **افزایش** دهیم.
- اگر خطا $e(p)$ منفی بود، باید خروجی پرسپترون $Y(p)$ را **کاهش** دهیم.

قاعده‌ی یادگیری پرسپترون

THE PERCEPTRON LEARNING RULE

$$w_i(p+1) = w_i(p) + \alpha \cdot x_i(p) \cdot e(p)$$
$$p = 1, 2, 3, \ldots$$

نرخ یادگیری (learning rate): یک ثابت مثبت کوچک‌تر از واحد

بر اساس قاعده‌ی یادگیری پرسپترون،
الگوریتم آموزش پرسپترون برای وظیفه‌ی طبقه‌بندی استخراج می‌شود.

الگوریتم آموزش پرسپترون

برای وظیفه‌ی طبقه‌بندی

PERCEPTRON'S TRAINING ALGORITHM

(۱) مقداردهی اولیه

وزن‌ها و مقدار آستانه را با اعداد تصادفی در بازه‌ی $[-0.5, 0.5]$ مقداردهی اولیه می‌کنیم؛ و $p = 1$

(۲) فعال‌سازی

پرسپترون را با اعمال ورودی p–ام فعال می‌کنیم و خروجی آن را محاسبه می‌کنیم.

$$Y(p) = step\left[\sum_{i=1}^{n} x_i(p)\, w_i(p) - \theta\right]$$

(۳) آموزش وزن‌ها

وزن‌های پرسپترون را به‌هنگام می‌کنیم:

$$Y(p) = step\left[\sum_{i=1}^{n} x_i(p)\, w_i(p) - \theta\right]$$

تصحیح وزن‌ها با **قاعده‌ی دلتا (delta rule)** انجام می‌شود:

$$w_i(p+1) = w_i(p) + \Delta w_i(p)$$

(۴) تکرار تا همگرایی

یک واحد به p اضافه می‌کنیم، به مرحله‌ی (۲) برمی‌گردیم و تکرار را تا رسیدن به همگرایی ادامه می‌دهیم.

الگوریتم آموزش پرسپترون

مثال: یادگیری تابع AND منطقی

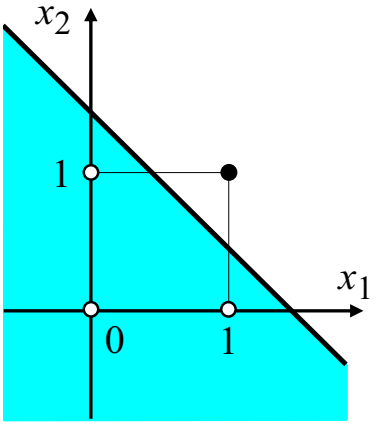
هر اپک، شامل نشان دادن همه‌ی مثال‌ها به شبکه است.

Epoch	Inputs		Desired output	Initial weights		Actual output	Error	Final weights	
	x_1	x_2	Y_d	w_1	w_2	Y	e	w_1	w_2
1	0	0	0	0.3	−0.1	0	0	0.3	−0.1
	0	1	0	0.3	−0.1	0	0	0.3	−0.1
	1	0	0	0.3	−0.1	1	−1	0.2	−0.1
	1	1	1	0.2	−0.1	0	1	0.3	0.0
2	0	0	0	0.3	0.0	0	0	0.3	0.0
	0	1	0	0.3	0.0	0	0	0.3	0.0
	1	0	0	0.3	0.0	1	−1	0.2	0.0
	1	1	1	0.2	0.0	1	0	0.2	0.0
3	0	0	0	0.2	0.0	0	0	0.2	0.0
	0	1	0	0.2	0.0	0	0	0.2	0.0
	1	0	0	0.2	0.0	1	−1	0.1	0.0
	1	1	1	0.1	0.0	0	1	0.2	0.1
4	0	0	0	0.2	0.1	0	0	0.2	0.1
	0	1	0	0.2	0.1	0	0	0.2	0.1
	1	0	0	0.2	0.1	1	−1	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1
5	0	0	0	0.1	0.1	0	0	0.1	0.1
	0	1	0	0.1	0.1	0	0	0.1	0.1
	1	0	0	0.1	0.1	0	0	0.1	0.1
	1	1	1	0.1	0.1	1	0	0.1	0.1

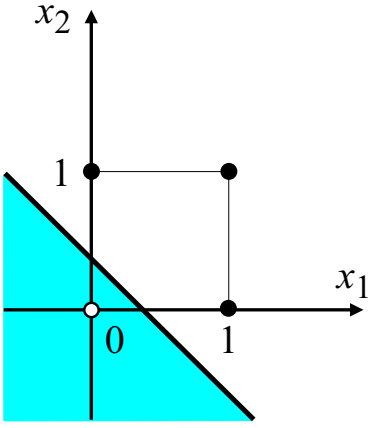
Threshold: $\theta = 0.2$; learning rate: $\alpha = 0.1$

مرز تصمیم برای توابع منطقی پایه

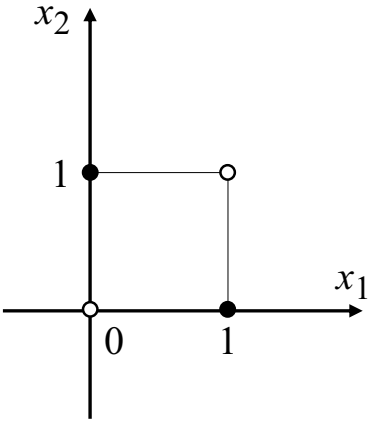
نمودار دوبعدی



(a) *AND* $(x_1 \cap x_2)$



(b) *OR* $(x_1 \cup x_2)$



(c) *Exclusive-OR* $(x_1 \oplus x_2)$

این دو تابع مرز تصمیم خطی دارند!
⇓
پرسپترون می‌تواند آن را یاد بگیرد.

مرز تصمیم خطی ندارد!
⇓
پرسپترون نمی‌تواند آن را یاد بگیرد!

هوش مصنوعی پیشرفته

شبکه‌های عصبی مصنوعی

۳

پرسپترون چندلایه

پرسپترون چندلایه

MULTILAYER NEURAL NETWORKS

پرسپترون چندلایه
Multilayer Perceptron

یک شبکه‌ی عصبی پیش‌خور با یک یا چند لایه‌ی پنهان

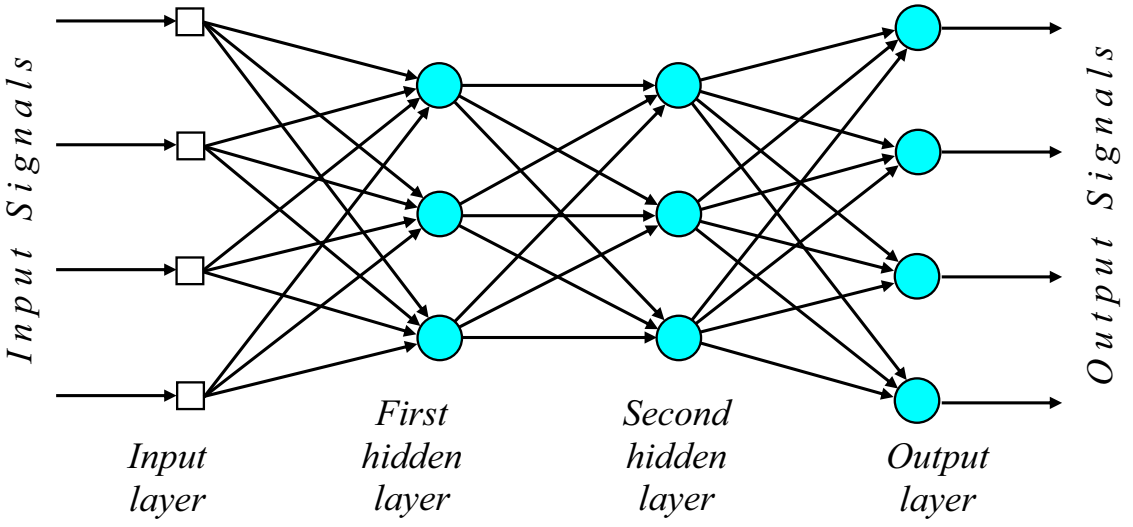
* یک **لایه‌ی ورودی** از نرون‌های مبدأ
* حداقل یک **لایه‌ی میانی** یا **لایه‌ی پنهان** از نرون‌های محاسباتی
* یک **لایه‌ی خروجی** از نرون‌های محاسباتی

سیگنال‌های ورودی در جهت پیش‌رو به صورت لایه به لایه منتشر می‌شوند.

پرسپترون چندلایه

با دو لایه‌ی پنهان

MULTILAYER PERCEPTRON WITH TWO HIDDEN LAYERS



پرسپترون چندلایه

لایه‌ی میانی / لایه‌ی پنهان

لایه‌ی میانی چه چیزی را پنهان می‌کند؟

لایه‌ی پنهان، خروجی مطلوب خودش را پنهان می‌کند:

- نرون‌های این لایه نمی‌توانند از طریق رفتار ورودی/خروجی شبکه مشاهده شوند.
- راه مشخصی برای فهمیدن خروجی مطلوب لایه‌ی میانی وجود ندارد

شبکه‌های عصبی مصنوعی تجاری، ۳ و گاهی ۴ لایه دارند که ۱ یا ۲ لایه‌ی آنها پنهان است.
هر لایه ۱۰ تا چندهزار نرون دارد.

شبکه‌های عصبی مصنوعی آزمایشگاهی، می‌توانند ۵ و یا حتی ۶ لایه داشته باشند، که ۳ یا ۴ لایه‌ی آنها پنهان است و از میلیون‌ها نرون بهره می‌برند.

پرسپترون چندلایه

یادگیری با پس‌انتشار خطا

یادگیری در یک شبکه‌ی چندلایه مشابه یادگیری برای یک پرسپترون است.

یک **مجموعه‌ی آموزشی** از الگوها به شبکه نشان داده می‌شود.
شبکه الگوی خروجی خودش را محاسبه می‌کند:
اگر خطا وجود داشت (تفاضل بین الگوهای خروجی واقعی و مطلوب)
وزن‌ها به‌گونه‌ای تنظیم می‌شوند که این خطا کاهش یابد.

دو مرحله در الگوریتم یادگیری پس انتشار خطا

(۱) انتشار ورودی

الگوی ورودی آموزشی به لایه‌ی ورودی شبکه نشان داده می‌شود.
شبکه آن را به جلو منتشر می‌کند تا الگوی خروجی در لایه خروجی ایجاد شود.

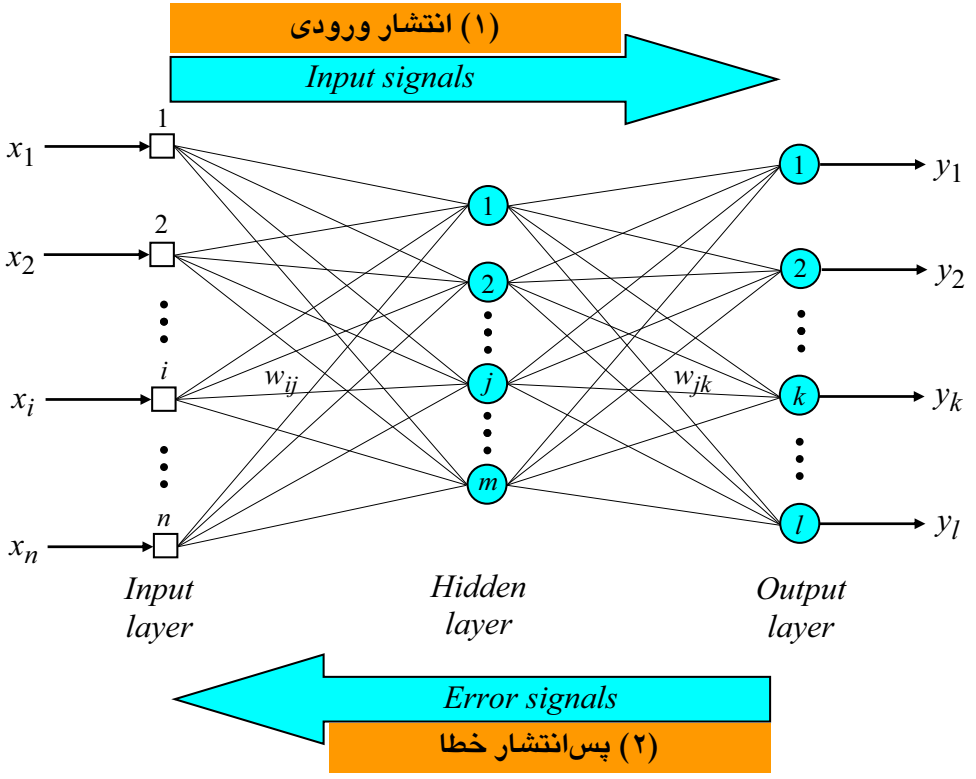
(۲) پس‌انتشار خطا

خطا در لایه‌ی خروجی محاسبه می‌شود و به عقب در شبکه منتشر می‌شود.
با پس‌انتشار خطا از خروجی به سمت ورودی، وزن‌ها اصلاح می‌شوند.

پرسپترون چندلایه

پس انتشار خطا در یک شبکه‌ی عصبی سه لایه

THREE-LAYER BACK-PROPAGATION NEURAL NETWORK



شبکه‌های عصبی مصنوعی

پرسپترون‌های چندلایه: یادگیری پس‌انتشار

BACK-PROPAGATION LEARNING

لایه‌ی خروجی: مشابه لایه‌ی پرسپترون تک‌لایه

$$W_{j,k} \leftarrow W_{j,k} + \alpha \times y_j \times \delta_k$$

$$\delta_k = e_k \times g'(in_k), \quad y_k = g(in_k)$$

لایه‌ی پنهان: خطا از لایه‌ی خروجی به پشت منتشر می‌شود **(back-propagate)**:

$$\delta_j = g'(in_j) \sum_{k=1}^{l} W_{j,k}\delta_k, \quad y_j = g(in_j)$$

قاعده‌ی به‌هنگام‌سازی برای وزن‌ها در لایه‌ی پنهان:

$$W_{i,j} \leftarrow W_{i,j} + \alpha \times x_i \times \delta_j$$

شبکه‌های عصبی مصنوعی

پرسپترون‌های چندلایه: یادگیری پس‌انتشار: سیگموئید لگاریتمی و مشتق آن

BACK-PROPAGATION LEARNING

تابع فعال‌سازی در همه‌ی لایه‌ها، سیگموئید لگاریتمی در نظر گرفته می‌شود.

$$g(u) = sigmoid(u) = \frac{1}{1+e^{-u}}$$

$$g'(u) = \frac{d}{du}g(u)$$

$$\frac{d}{du}g(u) = \frac{d}{du}\frac{1}{1+e^{-u}} = \frac{e^{-u}}{(1+e^{-u})^2} = \frac{1}{1+e^{-u}}\left(1-\frac{1}{1+e^{-u}}\right) = g(u)(1-g(u))$$

شبکه‌های عصبی مصنوعی

پرسپترون‌های چندلایه : یادگیری پس‌انتشار : بازنویسی قواعد به‌هنگام‌سازی وزن‌ها با جایگذاری مشتق

BACK-PROPAGATION LEARNING

لایه‌ی خروجی: مشابه لایه‌ی پرسپترون تک‌لایه

$$W_{j,k} \leftarrow W_{j,k} + \alpha \times y_j \times \delta_k$$

$$\delta_k = e_k \times y_k(1 - y_k)$$

لایه‌ی پنهان: خطا از لایه‌ی خروجی به پشت منتشر می‌شود **(back-propagate)** :

$$\delta_j = y_j(1 - y_j) \sum_{k=1}^{l} W_{j,k}\delta_k \ .$$

قاعده‌ی به‌هنگام‌سازی برای وزن‌ها در لایه‌ی پنهان:

$$W_{i,j} \leftarrow W_{i,j} + \alpha \times x_i \times \delta_j$$

BACK-PROPAGATION LEARNING

هدف این است که برای هر ورودی مقادیر وزن‌ها به‌گونه‌ای اصلاح شوند که خطا می‌نیمم شود؛

مجذور خطا بر روی یک مثال واحد به صورت زیر تعریف می‌شود:

$$E = \frac{1}{2}\sum_{k}(y_{d,k} - y_k)^2$$

که در آن مجموع روی همه‌ی گره‌ها در لایه‌ی خروجی محاسبه می‌شود.

$$\begin{aligned}
\frac{\partial E}{\partial W_{j,k}} &= -(y_{d,k} - y_k)\frac{\partial y_k}{\partial W_{j,k}} = -(y_{d,k} - y_k)\frac{\partial g(in_k)}{\partial W_{j,k}} \\
&= -(y_{d,k} - y_k)g'(in_k)\frac{\partial in_k}{\partial W_{j,k}} = -(y_{d,k} - y_k)g'(in_k)\frac{\partial}{\partial W_{j,k}}\left(\sum_j W_{j,k}y_j\right) \\
&= -(y_{d,k} - y_k)g'(in_k)y_j = -y_j\delta_k
\end{aligned}$$

شبکه‌های عصبی مصنوعی

پرسپترون‌های چندلایه: یادگیری پس‌انتشار: استخراج فرمول (۲): لایه‌ی پنهان

BACK-PROPAGATION LEARNING

$$
\begin{aligned}
\frac{\partial E}{\partial W_{i,j}} &= -\sum_k (y_{d,k} - y_k) \frac{\partial y_{d,k}}{\partial W_{i,j}} = -\sum_k (y_{d,k} - y_k) \frac{\partial g(in_k)}{\partial W_{i,j}} \\
&= -\sum_k (y_{d,k} - y_k) g'(in_k) \frac{\partial in_k}{\partial W_{i,j}} = -\sum_k \delta_k \frac{\partial}{\partial W_{i,j}} \left(\sum_j W_{j,k} y_j \right) \\
&= -\sum_k \delta_k W_{j,k} \frac{\partial y_j}{\partial W_{i,j}} = -\sum_k \delta_k W_{j,k} \frac{\partial g(in_j)}{\partial W_{i,j}} \\
&= -\sum_k \delta_k W_{j,k} g'(in_j) \frac{\partial in_j}{\partial W_{i,j}} \\
&= -\sum_k \delta_k W_{j,k} g'(in_j) \frac{\partial}{\partial W_{i,j}} \left(\sum_i W_{i,j} x_i \right) \\
&= -\sum_k \delta_k W_{j,k} g'(in_j) x_i = -x_i \delta_j
\end{aligned}
$$

پرسپترون چندلایه

الگوریتم آموزش پس‌انتشار خطا

THE BACK-PROPAGATION TRAINING ALGORITHM

(۱) مقداردهی اولیه

وزن‌ها و مقادیر آستانه را با اعداد تصادفی در بازه‌ی $[-0.5, 0.5]$ مقداردهی اولیه می‌کنیم؛ و $p = 1$

(۲) فعال‌سازی

شبکه را با اعمال ورودی p–ام فعال می‌کنیم و خروجی آن را لایه به لایه محاسبه می‌کنیم.

نرون j در لایه‌ی میانی $$y_j(p) = sigmoid\left[\sum_{i=1}^{n} x_i(p)\cdot w_{ij}(p) - \theta_j\right]$$

نرون k در لایه‌ی خروجی $$y_k(p) = sigmoid\left[\sum_{j=1}^{m} x_{jk}(p)\cdot w_{jk}(p) - \theta_k\right]$$

(۳) آموزش وزن‌ها

وزن‌های پرسپترون را به‌هنگام می‌کنیم.

تصحیح وزن‌ها با **قاعده‌ی دلتا (delta rule)** انجام می‌شود (بر اساس مشتق‌گیری زنجیره‌ای).

(۴) تکرار تا همگرایی

یک واحد به p اضافه می‌کنیم، به مرحله‌ی (۲) برمی‌گردیم و تکرار را تا رسیدن به همگرایی ادامه می‌دهیم.

پرسپترون چندلایه

الگوریتم آموزش پس‌انتشار خطا: به‌هنگام‌سازی وزن‌های لایه‌ی خروجی

(۳) آموزش وزن‌ها (لایه‌ی خروجی)

وزن‌های پرسپترون را به‌هنگام می‌کنیم.

$$w_{jk}(p+1) = w_{jk}(p) + \Delta w_{jk}(p)$$

تصحیح وزن‌ها با **قاعده‌ی دلتا (delta rule)** انجام می‌شود (بر اساس مشتق‌گیری زنجیره‌ای).

$$\Delta w_{jk}(p) = \alpha \cdot y_j(p) \cdot \delta_k(p)$$

$$\delta_k(p) = y_k(p) \cdot [1 - y_k(p)] \cdot e_k(p)$$

$$e_k(p) = y_{d,k}(p) - y_k(p)$$

پرسپترون چندلایه

الگوریتم آموزش پس‌انتشار خطا: به‌هنگام‌سازی وزن‌های لایه‌ی میانی

(۳) آموزش وزن‌ها (لایه‌ی میانی)

وزن‌های پرسپترون را به‌هنگام می‌کنیم.

$$w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p)$$

تصحیح وزن‌ها با **قاعده‌ی دلتا (delta rule)** انجام می‌شود (بر اساس مشتق‌گیری زنجیره‌ای).

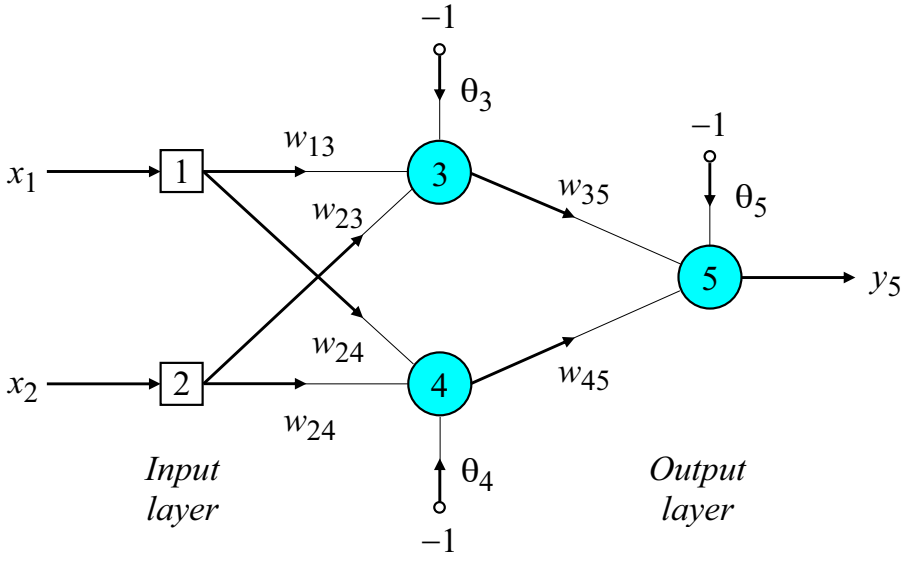
$$\Delta w_{ij}(p) = \alpha \cdot x_i(p) \cdot \delta_j(p)$$

$$\delta_j(p) = y_j(p) \cdot [1 - y_j(p)] \cdot \sum_{k=1}^{l} \delta_k(p)\, w_{jk}(p)$$

پرسپترون چندلایه

مثال: یک شبکه‌ی سه لایه برای انجام عمل یای انحصاری (XOR)

THREE-LAYER NETWORK FOR SOLVING THE EXCLUSIVE-OR OPERATION



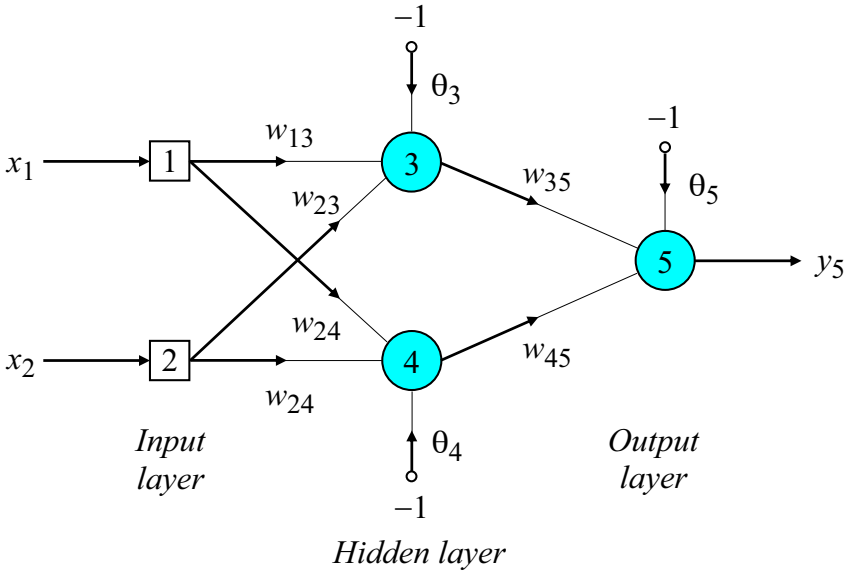
به‌عنوان مثال، یک شبکه‌ی پس‌انتشار خطای سه لایه را در نظر می‌گیریم.
می‌خواهیم این شبکه عمل یای انحصاری (XOR) را یاد بگیرد.

((یادآوری: پرسپترون تک‌لایه نمی‌توانست این عمل را یاد بگیرد.))

پرسپترون چندلایه

مثال: یک شبکه‌ی سه لایه برای انجام عملیات یای انحصاری (XOR): مقداردهی اولیه وزن‌ها

THREE-LAYER NETWORK FOR SOLVING THE EXCLUSIVE-OR OPERATION



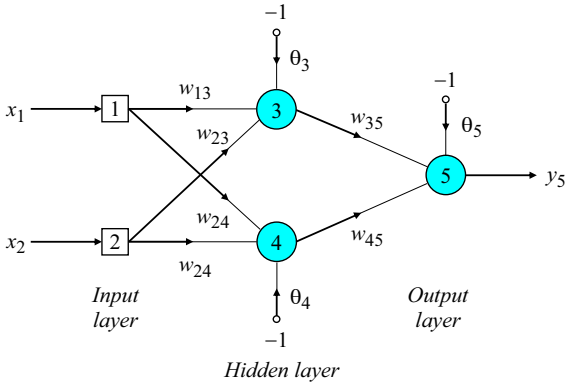
$w_{13} = 0.5$
$w_{14} = 0.9$
$w_{23} = 0.4$
$w_{24} = 1.0$
$w_{35} = -1.2$
$w_{45} = 1.1$
$\theta_3 = 0.8$
$\theta_4 = -0.1$
$\theta_5 = 0.3$

وزن‌های اولیه به صورت تصادفی مقداردهی شده‌اند.

اثر مقادیر آستانه‌ی نرون‌ها با وزن θ که به ورودی ثابت -1 متصل شده است، نمایش داده می‌شود.

پرسپترون چندلایه

مثال: یک شبکه‌ی سه لایه برای انجام عملیات یای انحصاری (XOR): فعال‌سازی نرون‌ها (در جهت پیش‌رو)



یک مثال از مجموعه‌ی آموزشی را به صورت مقابل در نظر می‌گیریم:

$$x_1 = 1$$
$$x_2 = 1$$
$$y_{d,5} = 0$$

خروجی واقعی نرون‌های 3 و 4 در لایه‌ی پنهان می‌شود:

$$y_3 = sigmoid\,(x_1 w_{13} + x_2 w_{23} - \theta_3) = 1/\left[1 + e^{-(1\cdot 0.5 + 1\cdot 0.4 - 1\cdot 0.8)}\right] = 0.5250$$

$$y_4 = sigmoid\,(x_1 w_{14} + x_2 w_{24} - \theta_4) = 1/\left[1 + e^{-(1\cdot 0.9 + 1\cdot 1.0 + 1\cdot 0.1)}\right] = 0.8808$$

خروجی واقعی نرون 5 در لایه‌ی خروجی می‌شود:

$$y_5 = sigmoid\,(y_3 w_{35} + y_4 w_{45} - \theta_5) = 1/\left[1 + e^{-(-0.5250\cdot 1.2 + 0.8808\cdot 1.1 - 1\cdot 0.3)}\right] = 0.5097$$

بنابراین خطای زیر در خروجی به‌دست می‌آید:

$$e = y_{d,5} - y_5 = 0 - 0.5097 = -0.5097$$

پرسپترون چندلایه

مثال: یک شبکه‌ی سه لایه برای انجام عملیات یای انحصاری (XOR): آموزش وزن‌ها: لایه خروجی

حال باید وزن‌های شبکه آموزش ببینند.
برای به‌هنگام‌سازی وزن‌های شبکه، خطای e از لایه‌ی خروجی به سمت لایه‌ی ورودی در جهت پس‌رو منتشر می‌شود:

ابتدا: گرادیان خطا برای نرون 5 در لایه‌ی خروجی محاسبه می‌شود:

$$\delta_5 = y_5\,(1-y_5)\,e = 0.5097\cdot(1-0.5097)\cdot(-0.5097) = -0.1274$$

سپس: میزان تصحیح وزن‌ها با فرض اینکه پارامتر نرخ یادگیری $\alpha = 0.1$ باشد محاسبه می‌شود:

$$\Delta w_{35} = \alpha\cdot y_3\cdot\delta_5 = 0.1\cdot 0.5250\cdot(-0.1274) = -0.0067$$

$$\Delta w_{45} = \alpha\cdot y_4\cdot\delta_5 = 0.1\cdot 0.8808\cdot(-0.1274) = -0.0112$$

$$\Delta\theta_5 = \alpha\cdot(-1)\cdot\delta_5 = 0.1\cdot(-1)\cdot(-0.1274) = -0.0127$$

پرسپترون چندلایه

مثال: یک شبکه‌ی سه لایه برای انجام عملیات یای انحصاری (XOR): آموزش وزن‌ها: لایه پنهان

سپس: گرادیان خطا برای نرون‌های 3 و 4 در لایه‌ی پنهان محاسبه می‌شود:

$$\delta_3 = y_3(1-y_3)\cdot\delta_5\cdot w_{35} = 0.5250\cdot(1-0.5250)\cdot(-0.1274)\cdot(-1.2) = 0.0381$$

$$\delta_4 = y_4(1-y_4)\cdot\delta_5\cdot w_{45} = 0.8808\cdot(1-0.8808)\cdot(-0.127\,4)\cdot 1.1 = -0.0147$$

سپس: میزان تصحیح وزن‌ها با فرض اینکه پارامتر نرخ یادگیری $\alpha = 0.1$ باشد محاسبه می‌شود:

$$\Delta w_{13} = \alpha\cdot x_1\cdot\delta_3 = 0.1\cdot 1\cdot 0.0381 = 0.0038$$
$$\Delta w_{23} = \alpha\cdot x_2\cdot\delta_3 = 0.1\cdot 1\cdot 0.0381 = 0.0038$$
$$\Delta\theta_3 = \alpha\cdot(-1)\cdot\delta_3 = 0.1\cdot(-1)\cdot 0.0381 = -0.0038$$
$$\Delta w_{14} = \alpha\cdot x_1\cdot\delta_4 = 0.1\cdot 1\cdot(-0.0147) = -0.0015$$
$$\Delta w_{24} = \alpha\cdot x_2\cdot\delta_4 = 0.1\cdot 1\cdot(-0.0147) = -0.0015$$
$$\Delta\theta_4 = \alpha\cdot(-1)\cdot\delta_4 = 0.1\cdot(-1)\cdot(-0.0147) = 0.0015$$

پرسپترون چندلایه

مثال: یک شبکه‌ی سه لایه برای انجام عملیات یای انحصاری (XOR): آموزش وزن‌ها: به‌هنگام‌سازی

سرانجام: همه‌ی وزن‌ها به‌هنگام می‌شوند:

$$w_{13} = w_{13} + \Delta w_{13} = 0.5 + 0.0038 = 0.5038$$

$$w_{14} = w_{14} + \Delta w_{14} = 0.9 - 0.0015 = 0.8985$$

$$w_{23} = w_{23} + \Delta w_{23} = 0.4 + 0.0038 = 0.4038$$

$$w_{24} = w_{24} + \Delta w_{24} = 1.0 - 0.0015 = 0.9985$$

$$w_{35} = w_{35} + \Delta w_{35} = -1.2 - 0.0067 = -1.2067$$

$$w_{45} = w_{45} + \Delta w_{45} = 1.1 - 0.0112 = 1.0888$$

$$\theta_3 = \theta_3 + \Delta \theta_3 = 0.8 - 0.0038 = 0.7962$$

$$\theta_4 = \theta_4 + \Delta \theta_4 = -0.1 + 0.0015 = -0.0985$$

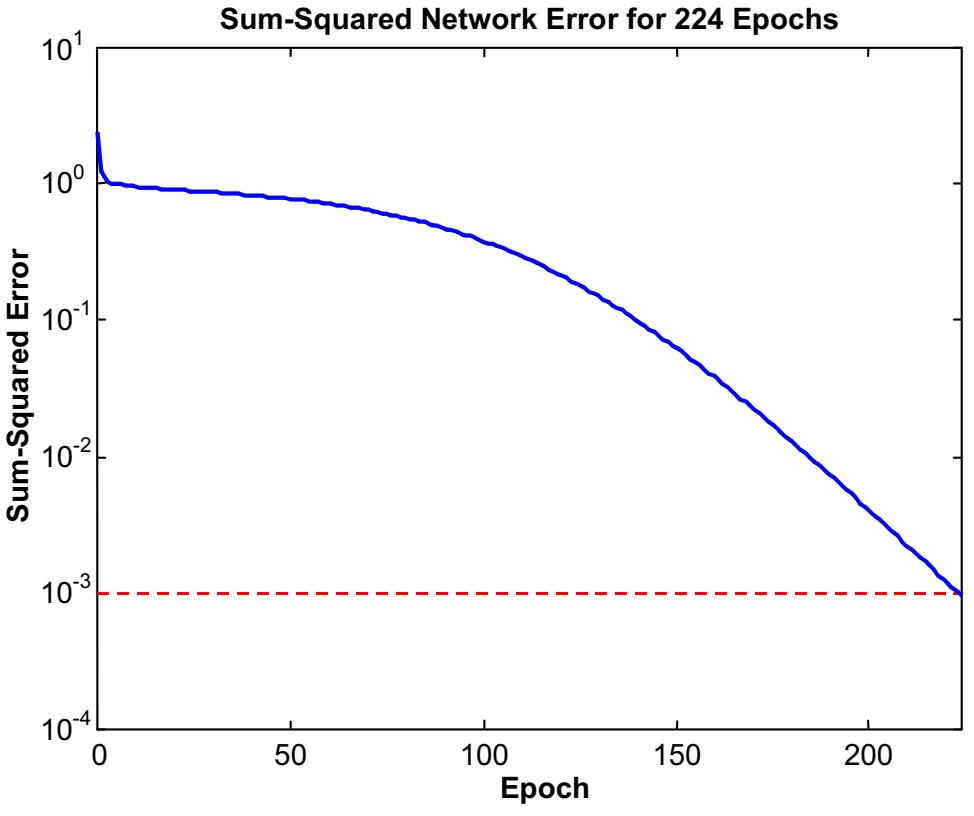
$$\theta_5 = \theta_5 + \Delta \theta_5 = 0.3 + 0.0127 = 0.3127$$

فرآیند آموزش تکرار می‌شود تا مجموع مربعات خطا به کمتر از 0.001 برسد.

پرسپترون چندلایه

مثال: یک شبکه‌ی سه لایه برای انجام عملیات یای انحصاری (XOR): منحنی یادگیری

LEARNING CURVE



پرسپترون چندلایه

مثال: یک شبکه‌ی سه لایه برای انجام عملیات یای انحصاری (XOR): نتایج نهایی یادگیری

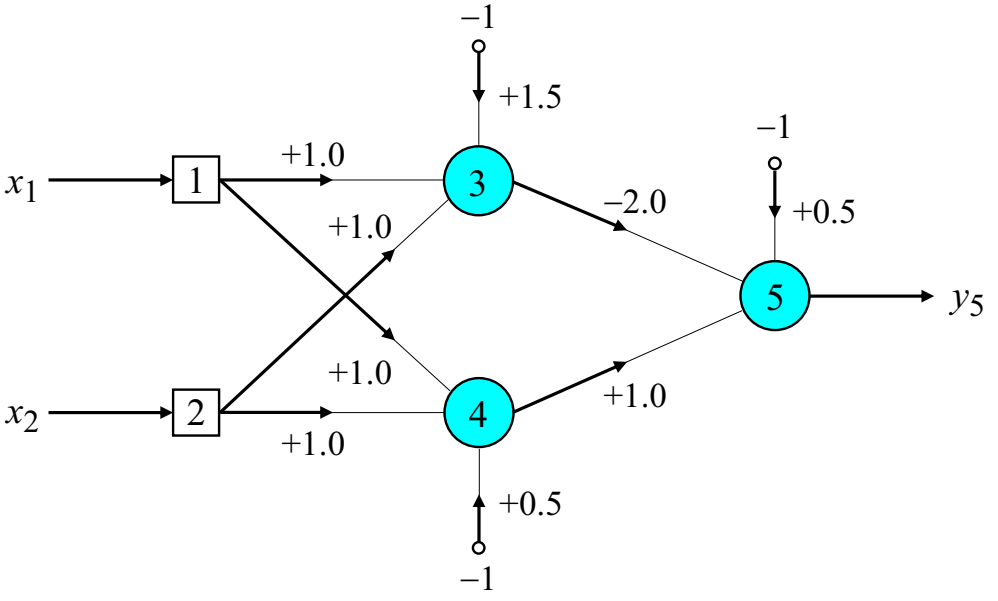
FINAL RESULTS OF LEARNING

Inputs		Desired output	Actual output	Error	Sum of squared errors
x_1	x_2	y_d	y_5	e	
1	1	0	0.0155	–0.0155	0.0010
0	1	1	0.9849	0.0151	
1	0	1	0.9849	0.0151	
0	0	0	0.0175	–0.0175	

پرسپترون چندلایه

مثال: یک شبکه‌ی سه لایه برای انجام عملیات یای انحصاری (XOR): شبکه‌ی نهایی

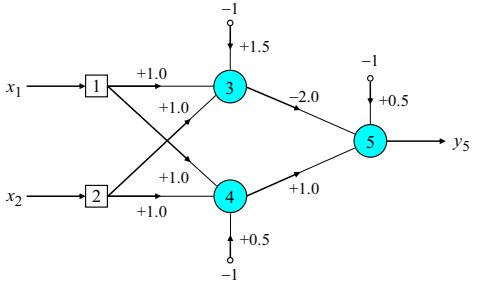
NETWORK REPRESENTED BY MCCULLOCH-PITTS MODEL FOR SOLVING THE EXCLUSIVE-OR OPERATION

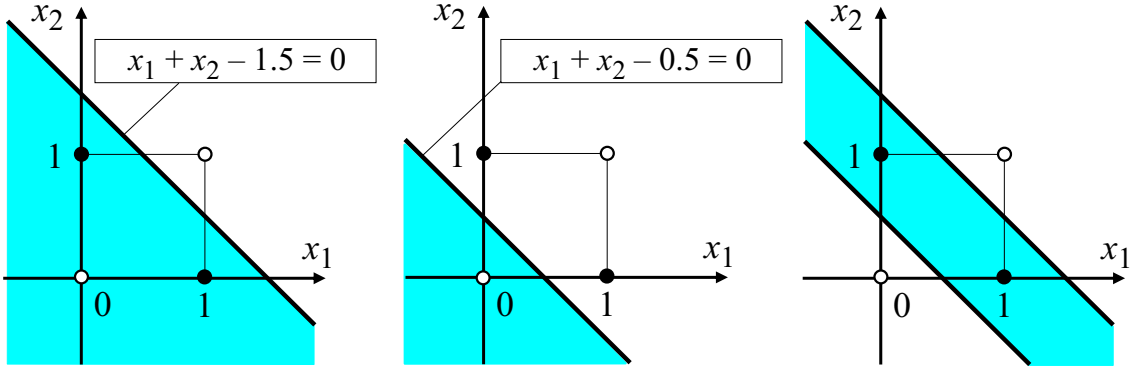


پرسپترون چندلایه

مثال: یک شبکه‌ی سه لایه برای انجام عملیات یای انحصاری (XOR): مرزهای تصمیم

DECISION BOUNDARIES





(*a*) (*b*) (*c*)

(*a*) Decision boundary constructed by hidden neuron 3;
(*b*) Decision boundary constructed by hidden neuron 4;
(*c*) Decision boundaries constructed by the complete three-layer network

پرسپترون چندلایه

یادگیری شتاب‌دهی‌شده در شبکه‌های عصبی چندلایه: استفاده از تابع تانژانت هایپربولیک

ACCELERATED LEARNING IN MULTILAYER NEURAL NETWORKS: USING HYPERBOLIC TANGENT

اگر تابع فعال‌سازی سیگموئید را با **تانژانت هایپربولیک** نمایش دهیم (به‌جای سیگموئید لگاریتمی) شبکه‌ی چندلایه سریع‌تر یاد می‌گیرد.

$$Y^{\tanh} = \frac{2a}{1+e^{-bX}} - a$$

مقادیر مناسب برای a و b ثابت: $a = 1.716,\ b = 0.667$

پرسپترون چندلایه

یادگیری شتاب‌دهی‌شده در شبکه‌های عصبی چندلایه: استفاده از جمله‌ی مومنتوم

ACCELERATED LEARNING IN MULTILAYER NEURAL NETWORKS: USING MOMENTUM TERM

می‌توان آموزش شبکه را با وارد کردن جمله‌ی مومنتوم در قاعده‌ی دلتا شتاب‌دهی کرد؛

قاعده‌ی دلتای تعمیم‌یافته:

تحت تأثیر مقدار قبلی وزن

$$\Delta w_{jk}(p) = \beta \cdot \Delta w_{jk}(p-1) + \alpha \cdot y_j(p) \cdot \delta_k(p)$$

ثابت مومنتوم
Momentum Constant

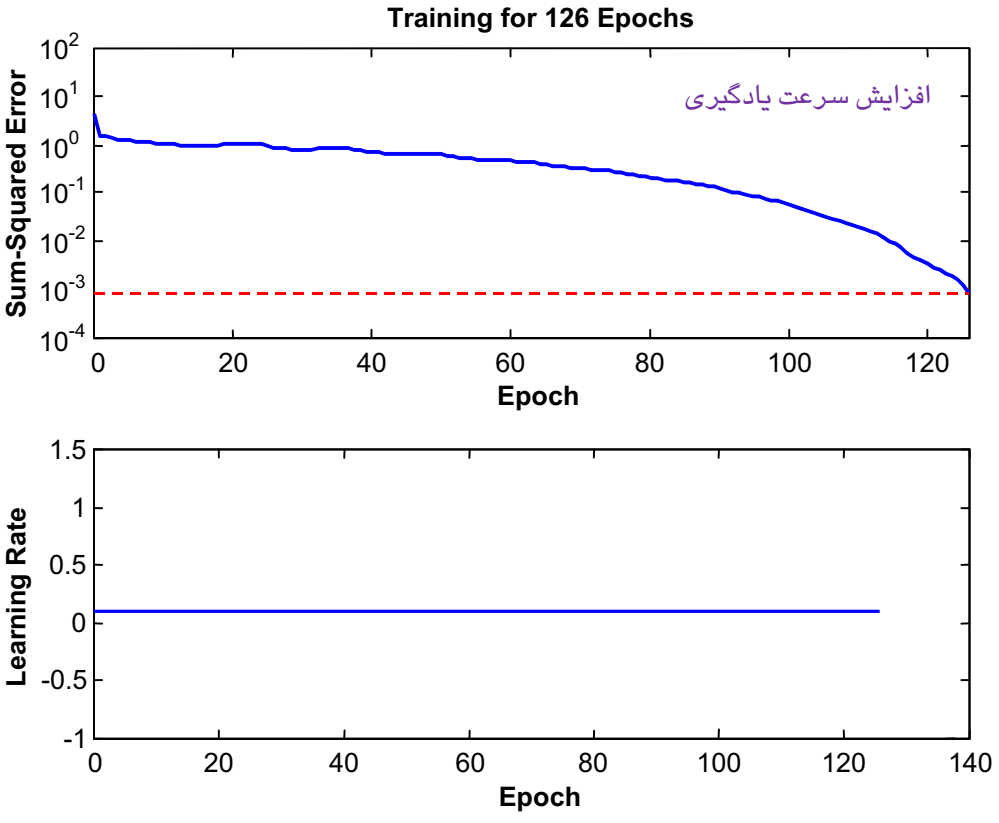
$$0 \le \beta < 1$$

$$\beta \cong 0.95$$

پرسپترون چندلایه

یادگیری شتاب‌دهی‌شده در شبکه‌های عصبی چندلایه: استفاده از جمله‌ی مومنتوم: مثال

LEARNING WITH MOMENTUM FOR OPERATION EXCLUSIVE-OR



پرسپترون چندلایه

یادگیری شتاب‌دهی‌شده در شبکه‌های عصبی چندلایه: استفاده از نرخ یادگیری وفقی

ACCELERATED LEARNING IN MULTILAYER NEURAL NETWORKS: USING ADAPTIVE LEARNING RATE

برای شتاب‌دهی همگرایی و همچنین اجتناب از خطر ناپایداری، می‌توانیم دو هیوریستیک زیر را اعمال کنیم:

هیوریتسیک ۱	اگر **علامت تغییر** مجموع مربعات خطا در چندین اپک متوالی **یکسان** بود، آن‌گاه نرخ یادگیری α باید افزایش یابد.
هیوریتسیک ۲	اگر **علامت تغییر** مجموع مربعات خطا در چندین اپک متوالی **متغیر** بود، آن‌گاه نرخ یادگیری α باید کاهش یابد.

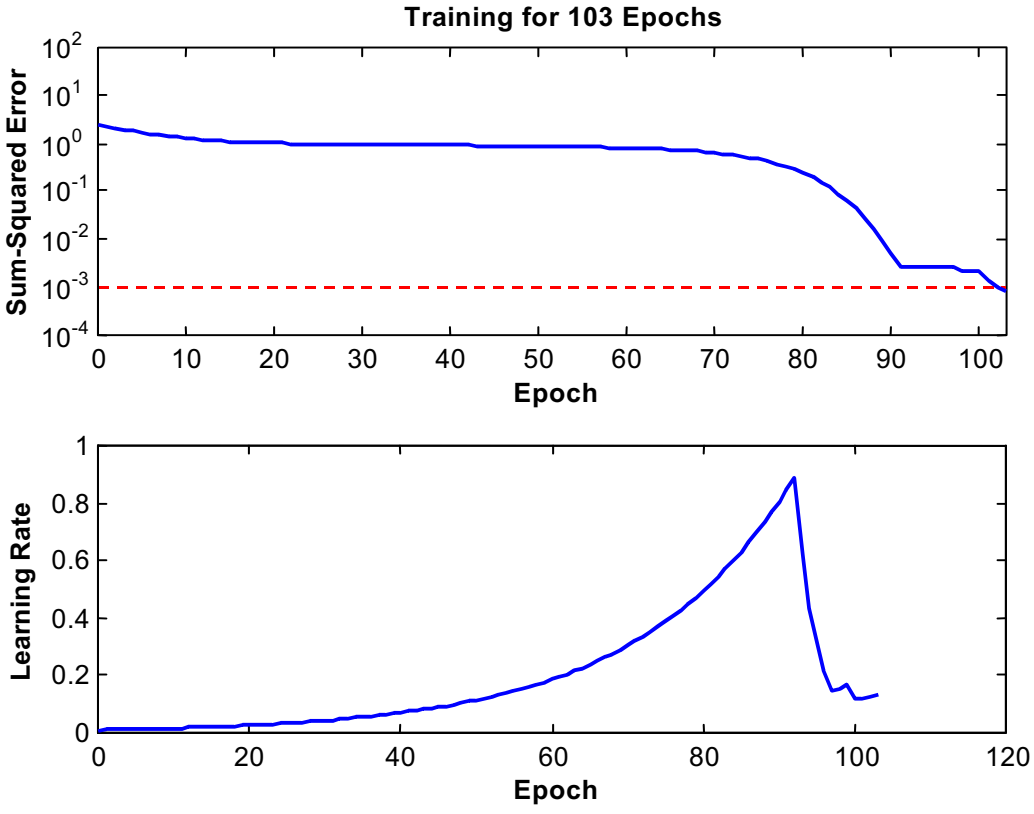
وفق دادن نرخ یادگیری نیازمند تغییراتی در الگوریتم پس‌انتشار است:

- اگر مجموع مربعات خطا در اپک فعلی از مقدار قبلی آن به‌میزان یک نسبت از پیش تعریف‌شده (مثلاً 1.04) بیشتر شود، پارامتر نرخ یادگیری کاهش می‌یابد (معمولاً با ضرب در 0.7) و وزن‌های جدید محاسبه می‌شوند.
- اگر مجموع مربعات خطا در اپک فعلی از مقدار قبلی آن بیشتر شود، پارامتر نرخ یادگیری افزایش می‌یابد (معمولاً با ضرب در 1.05) و وزن‌های جدید محاسبه می‌شوند.

پرسپترون چندلایه

یادگیری با نرخ یادگیری وفقی

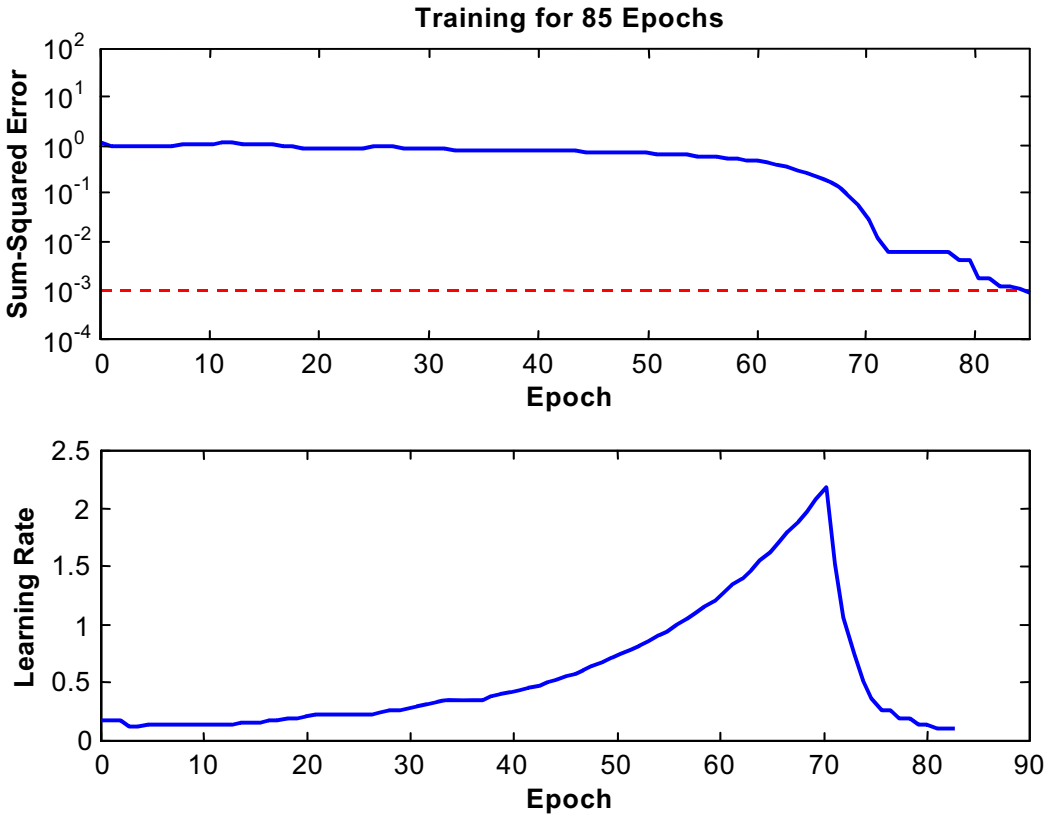
LEARNING WITH ADAPTIVE LEARNING RATE



پرسپترون چندلایه

یادگیری با مومنتوم و نرخ یادگیری وفقی

LEARNING WITH MOMENTUM AND ADAPTIVE LEARNING RATE



هوش مصنوعی پیشرفته

شبکه‌های عصبی مصنوعی

۴

شبکه‌ی هاپفیلد

شبکه‌های بازگشتی

RECURRENT NETWORKS

شبکه‌های عصبی مصنوعی بر اساس قیاس با مغز طراحی شده‌اند.

حافظه‌ی مغز، از طریق **پیونددهی** (association) کار می‌کند.

برای مثال:
می‌توانیم یک چهره‌ی آشنا را در یک محیط ناآشنا در حدود ۱۰۰ تا ۲۰۰ میلی‌ثانیه بازشناسی کنیم.

مثال دیگر:
می‌توانیم یک تجربه‌ی حسی شامل صدا و صحنه را با شنیدن تنها چند بخش کوچکی از موسیقی به‌خاطر بیاوریم.

مغز به صورت عادی یک چیز را به چیز دیگر پیونددهی می‌کند!

برای شبیه‌سازی مشخصه‌های پیونددهی حافظه‌ی انسان، به نوع متفاوتی از شبکه‌های عصبی نیاز داریم: **شبکه‌های بازگشتی**

در یک شبکه‌های عصبی بازگشتی، از خروجی‌ها به ورودی‌های شبکه حلقه‌ی فیدبک وجود دارد.

وجود چنین حلقه‌هایی بر قابلیت یادگیری شبکه تأثیر فراوانی ایجاد می‌کند.

شبکه‌ی هاپفیلد

THE HOPFIELD NETWORK

شبکه‌ی هاپفیلد

Hopfield Network

یک شبکه‌ی عصبی بازگشتی

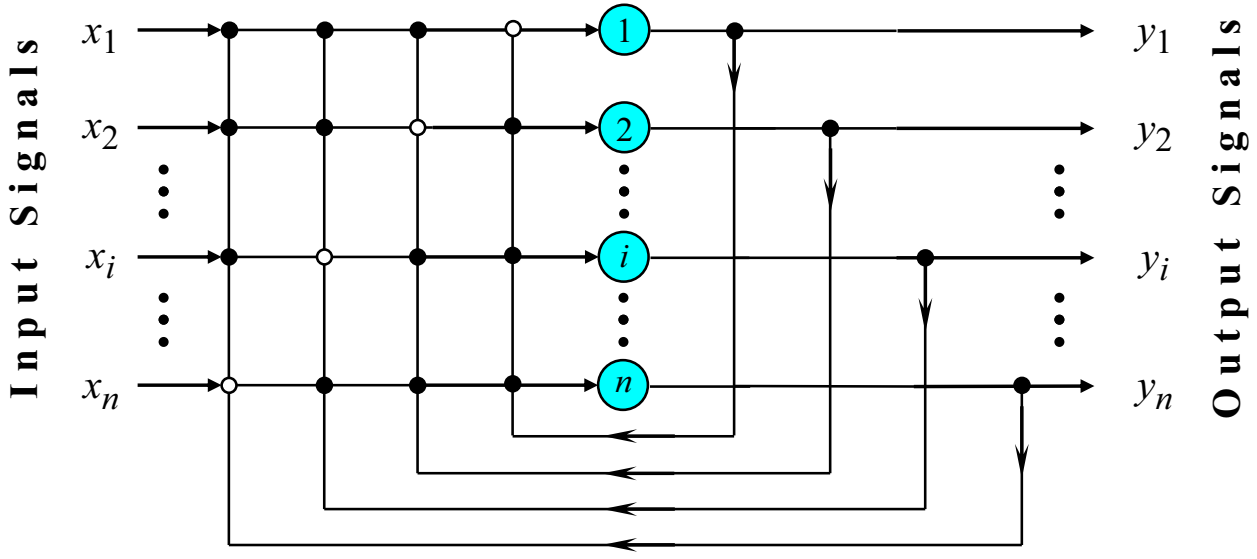
در مورد شبکه‌های بازگشتی یک مسئله‌ی مهم مسئله‌ی پایداری است.

جان هاپفیلد (۱۹۸۲)، مشکل پیش‌بینی‌پذیری پایداری شبکه‌ی بازگشتی را
با فرمول‌بندی اصل فیزیکی ذخیره‌ی اطلاعات در یک شبکه‌ی پایدار دینامیکی حل کرد.

شبکه‌ی هاپفیلد

شبکه‌ی هاپفیلد تک‌لایه با n نرون

SINGLE-LAYER n-NEURON HOPFIELD NETWORK



خروجی هر نرون به ورودی سایر نرون‌ها وصل می‌شود.

شبکه‌ی هاپفیلد

نرون

شبکه‌ی هاپفیلد از نرون‌های مک‌کلوچ–پیتز با تابع فعال‌سازی علامت استفاده می‌کند.

$$Y^{sign} = \begin{cases} +1, & \text{if } X > 0 \\ -1, & \text{if } X < 0 \\ Y, & \text{if } X = 0 \end{cases}$$

مقدار قبلی نرون حفظ می‌شود،
وقتی ورودی صفر است.

شبکه‌ی هاپفیلد

حالت شبکه

NETWORK SATATE

حالت فعلی در شبکه‌ی هاپفیلد با برداری متشکل از خروجی همه‌ی نرون‌ها مشخص می‌شود:
(بردار حالت: State Vector)

$$\mathbf{Y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

برای یک شبکه‌ی هاپفیلد تک‌لایه با n نرون

شبکه‌ی هاپفیلد

وزن‌های سیناپسی بین نرون‌ها

در شبکه‌ی هاپفیلد، وزن سیناپسی بین نرون‌ها
معمولاً در قالب ماتریسی نمایش داده می‌شود:

$$\mathbf{W} = \sum_{m=1}^{M} \mathbf{Y}_m \mathbf{Y}_m^T - M\,\mathbf{I}$$

$\mathbf{Y}_m$: بردار حالت دودویی با n بعد

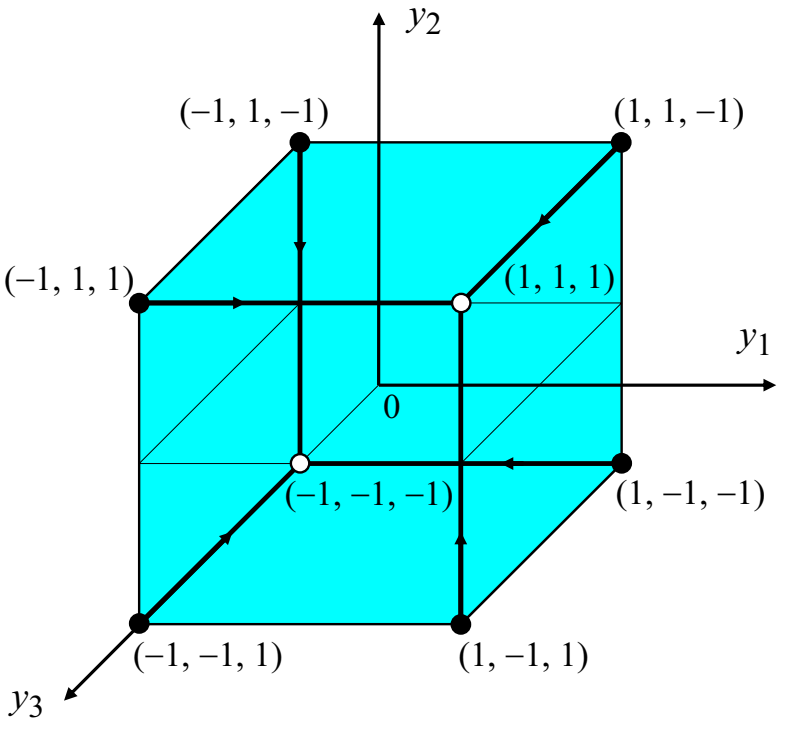
$\mathbf{I}$: ماتریس همانی مرتبه n

M : تعداد حالت‌هایی که باید توسط شبکه حفظ شود.

شبکه‌ی هاپفیلد

حالت‌های ممکن برای یک شبکه‌ی هاپفیلد با ۳ نرون

POSSIBLE STATES FOR THE THREE-NEURON HOPFIELD NETWORK



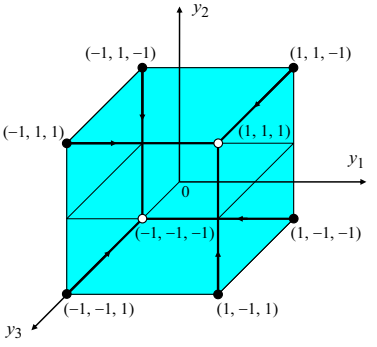
شبکه‌ی هاپفیلد

حالت پایدار (حافظه‌ی مبنا)

STABLE SATATE (FUNDAMENTAL MEMORY)

یک حالت/رأس پایدار با موارد زیر تعیین می‌شود:
ماتریس وزن **W**
بردار ورودی جاری **X**
ماتریس آستانه **Θ**

اگر بردار ورودی به طور جزئی نادرست یا ناکامل باشد،
حالت اولیه پس از تعدادی تکرار به حالت/رأس پایدار همگرا می‌شود.



شبکه‌ی هاپفیلد

حالت پایدار: مثال

برای نمونه فرض کنید شبکه‌ی ما بخواهد دو حالت مخالف $\mathbf{Y}_1$ و $\mathbf{Y}_2$ را حفظ کند:

$$\mathbf{Y}_1=\begin{bmatrix}1\\1\\1\end{bmatrix} \qquad \mathbf{Y}_2=\begin{bmatrix}-1\\-1\\-1\end{bmatrix} \qquad \Rightarrow \qquad \mathbf{Y}_1^T=\begin{bmatrix}1 & 1 & 1\end{bmatrix} \qquad \mathbf{Y}_2^T=\begin{bmatrix}-1 & -1 & -1\end{bmatrix}$$

ماتریس وزن‌ها به‌صورت زیر تعیین می‌شود:

$$\mathbf{W}=\sum_{m=1}^{M}\mathbf{Y}_m\mathbf{Y}_m^T-M\,\mathbf{I} \qquad \Rightarrow \qquad \mathbf{W}=\begin{bmatrix}1\\1\\1\end{bmatrix}\begin{bmatrix}1 & 1 & 1\end{bmatrix}+\begin{bmatrix}-1\\-1\\-1\end{bmatrix}\begin{bmatrix}-1 & -1 & -1\end{bmatrix}-2\begin{bmatrix}1 & 0 & 0\\0 & 1 & 0\\0 & 0 & 1\end{bmatrix}=\begin{bmatrix}0 & 2 & 2\\2 & 0 & 2\\2 & 2 & 0\end{bmatrix}$$

حال شبکه را با بردارهای ورودی $\mathbf{X}_1=\mathbf{Y}_1$ و $\mathbf{X}_2=\mathbf{Y}_2$ آزمایش می‌کنیم:

$$\mathbf{Y}_1=sign\left\{\begin{bmatrix}0 & 2 & 2\\2 & 0 & 2\\2 & 2 & 0\end{bmatrix}\begin{bmatrix}1\\1\\1\end{bmatrix}-\begin{bmatrix}0\\0\\0\end{bmatrix}\right\}=\begin{bmatrix}1\\1\\1\end{bmatrix} \qquad \mathbf{Y}_2=sign\left\{\begin{bmatrix}0 & 2 & 2\\2 & 0 & 2\\2 & 2 & 0\end{bmatrix}\begin{bmatrix}-1\\-1\\-1\end{bmatrix}-\begin{bmatrix}0\\0\\0\end{bmatrix}\right\}=\begin{bmatrix}-1\\-1\\-1\end{bmatrix}$$

مشاهده می‌شود که در هر دو مورد خروجی با ورودی مساوی است.

۶ حالت دیگر همگی ناپایدار هستند.

شبکه‌ی هاپفیلد

حالت‌های ناپایدار

حالت‌های پایدار (حافظه‌های مبنایی)، قادرند حالت‌هایی که به آنها نزدیک هستند را جذب کنند.

$\mathbf{Y}_1^T = [1 \quad 1 \quad 1]$ $\qquad$ $\mathbf{Y}_2^T = [-1 \quad -1 \quad -1]$

حالت‌های پایدار (حافظه‌های مبنایی)

جذب می‌کند $\qquad$ جذب می‌کند

$[-1 \quad 1 \quad 1]$ $\qquad$ $[-1 \quad -1 \quad 1]$

$[1 \quad -1 \quad 1]$ $\qquad$ حالت‌های ناپایدار $\qquad$ $[1 \quad -1 \quad -1]$

$[1 \quad 1 \quad -1]$ $\qquad$ $[-1 \quad 1 \quad -1]$

هر حالت ناپایدار نسبت به حالت پایدار جاذب آنها یک خطا دارد.

شبکه‌ی هاپفیلد می‌تواند به‌عنوان یک **شبکه‌ی تصحیح خطا** عمل کند.

شبکه‌ی هاپفیلد

ظرفیت ذخیره‌ی شبکه‌ی هاپفیلد

STORAGE CAPACITY OF THE HOPFIELD NETWORK

ظرفیت ذخیرهی یک شبکه‌ی هاپفیلد برابر است با
بزرگ‌ترین تعداد حافظه‌های مبنایی که می‌توانند به‌درستی ذخیره و بازیابی شوند.

حداکثر تعداد حافظه‌های مبنایی که می‌تواند در یک شبکه‌ی بازگشتی با n نرون ذخیره شود:

$$M_{max} = 0.15\, n$$

(اثبات با آزمایش تجربی)

هوش مصنوعی پیشرفته

شبکه‌های عصبی مصنوعی

۵

حافظه‌ی پیوندی دوطرفه

حافظه‌ی پیوندی

ASSOCIATIVE MEMORY (AM)

حافظه‌ی پیوندی

حافظه‌ی خودپیوندی
Autoassociative

یک الگو را به همان الگو پیوند می‌دهد.

مثل **شبکه‌ی هاپفیلد**:

می‌تواند یک حافظه‌ی خراب‌شده یا ناکامل را بازیابی کند،
اما نمی‌تواند یک حافظه را به حافظه‌ی دیگر پیونددهی کند.

حافظه‌ی دگرپیوندی
Hetroassociative

یک الگو را به یک الگوی دیگر پیوند می‌دهد.

مثل **حافظه‌ی انسان**:

یک چیز می‌تواند چیز دیگری را به‌خاطر بیاورد
و همین طور چیز دیگر ...

ما از یک زنجیره از پیوندهای ذهنی برای بازیابی یک حافظه‌ی فراموش شده استفاده می‌کنیم.

نیاز به یک شبکه‌ی عصبی بازگشتی داریم که بتواند الگوی ورودی را بر روی مجموعه‌ای از نرون‌ها دریافت کند و الگوی خروجی را بر روی مجموعه‌ی دیگری از نرون‌ها بسازد.

حافظه‌ی پیوندی دوطرفه

BIDIRECTIONAL ASSOCIATIVE MEMORY (BAM)

حافظه‌ی پیوندی دوطرفه

Bidirectional Associative Memory (BAM)

یک شبکه‌ی عصبی بازگشتی
شبکه‌ی دگرپیوندی

الگوهایی از یک مجموعه A را به الگوهایی از یک مجموعه‌ی دیگر B پیوند می‌دهد و برعکس.

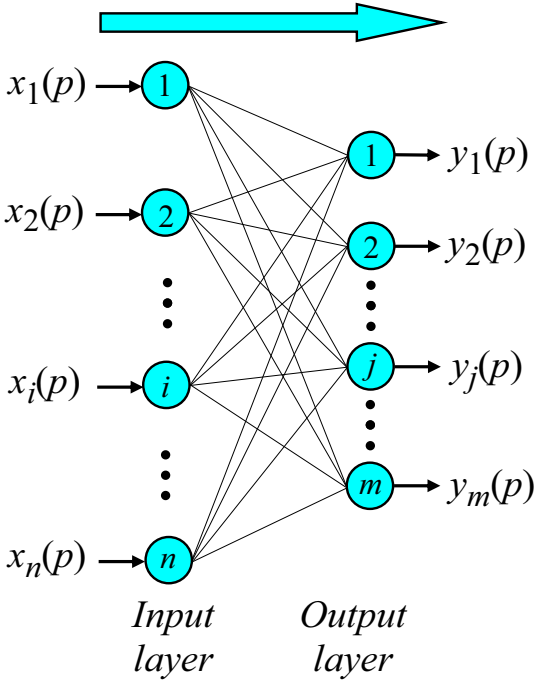
طراحی شده توسط بارت کاسکو

مانند شبکه‌ی هاپفیلد، BAM می‌تواند تعمیم‌دهی کند و نیز خروجی صحیح تولید کند حتی اگر ورودی خراب‌شده یا ناکامل باشد.

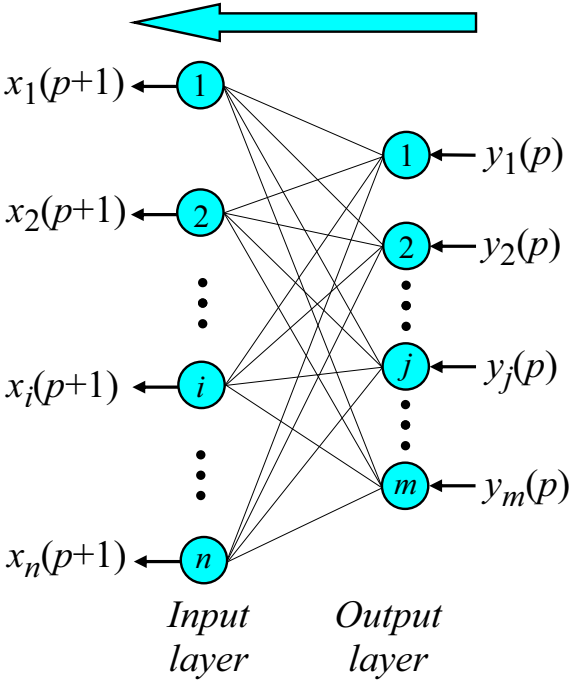
حافظه‌ی پیوندی دوطرفه

عملکرد

BAM OPERATION



(*a*) Forward direction.



(*b*) Backward direction.

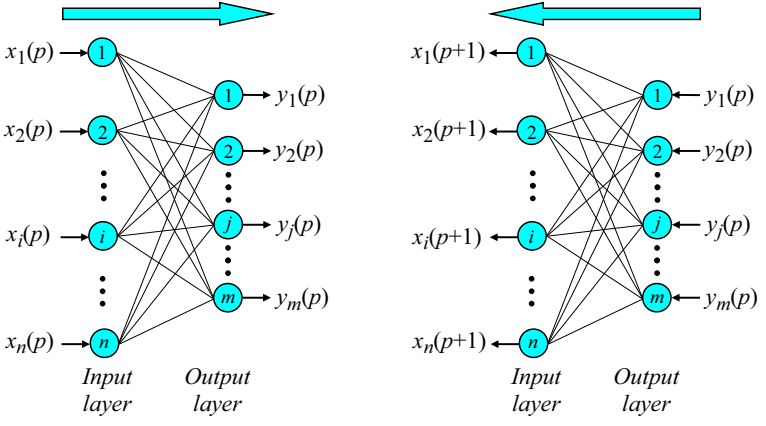
حافظه‌ی پیوندی دوطرفه

عملکرد BAM

ایده‌ی پایه‌ی BAM، ذخیره کردن جفت الگوها به‌گونه‌ای است که:

هرگاه بردار n بعدی $\mathbf{X}$ از مجموعه‌ی A به‌عنوان ورودی به شبکه نشان داده شد، بردار m بعدی $\mathbf{Y}$ از مجموعه‌ی B به‌عنوان خروجی به‌خاطر آورده شود.

برعکس هرگاه $\mathbf{Y}$ به‌عنوان ورودی نشان داده شد، $\mathbf{X}$ به‌خاطر آورده شود.



(*a*) Forward direction. (*b*) Backward direction.

حافظه‌ی پیوندی دوطرفه

ایجاد BAM

برای ساخت BAM باید یک **ماتریس همبستگی (correlation)** برای هر الگویی که می‌خواهیم ذخیره کنیم ایجاد کنیم:

ماتریس همبستگی هر پیوند = ضرب ماتریسی بردار ورودی و ترانهاده‌ی بردار خروجی

ماتریس وزن BAM = مجموع همه‌ی ماتریس‌های همبستگی

M : تعداد جفت الگوهایی که باید در BAM ذخیره شود.

$$\mathbf{W} = \sum_{m=1}^{M} \mathbf{X}_m \ \mathbf{Y}_m^T$$

برای بازیابی الگو:

$$\mathbf{Y}_m = sign\,(\mathbf{W}^T\,\mathbf{X}_m), \qquad m = 1, 2, \ldots, M$$

$$\mathbf{X}_m = sign\,(\mathbf{W}\,\mathbf{Y}_m), \qquad m = 1, 2, \ldots, M$$

حافظه‌ی پیوندی دوطرفه

مثال (۱ از ۳)

$$\text{Set } A\text{: } \mathbf{X}_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \quad \mathbf{X}_2 = \begin{bmatrix} -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \end{bmatrix} \quad \mathbf{X}_3 = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \\ 1 \\ 1 \end{bmatrix} \quad \mathbf{X}_4 = \begin{bmatrix} -1 \\ -1 \\ 1 \\ 1 \\ -1 \\ -1 \end{bmatrix}$$

$$\text{Set } B\text{: } \mathbf{Y}_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \quad \mathbf{Y}_2 = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix} \quad \mathbf{Y}_3 = \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} \quad \mathbf{Y}_4 = \begin{bmatrix} -1 \\ 1 \\ -1 \end{bmatrix}$$

حافظه‌ی پیوندی دوطرفه

مثال (۲ از ۳)

$$\mathbf{W} = \sum_{m=1}^{4} \mathbf{X}_m \, \mathbf{Y}_m^T$$

$$\mathbf{W} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} + \begin{bmatrix} -1 \\ -1 \\ -1 \\ -1 \\ -1 \\ -1 \end{bmatrix} \begin{bmatrix} -1 & -1 & -1 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \\ 1 \\ 1 \end{bmatrix} \begin{bmatrix} 1 & -1 & 1 \end{bmatrix}$$

$$+ \begin{bmatrix} -1 \\ -1 \\ 1 \\ 1 \\ -1 \\ -1 \end{bmatrix} \begin{bmatrix} -1 & 1 & -1 \end{bmatrix} = \begin{bmatrix} 4 & 0 & 4 \\ 4 & 0 & 4 \\ 0 & 4 & 0 \\ 0 & 4 & 0 \\ 4 & 0 & 4 \\ 4 & 0 & 4 \end{bmatrix}$$

حافظه‌ی پیوندی دوطرفه

مثال (۳ از ۳)

$$\mathbf{Y}_m = sign\left(\mathbf{W}^T \mathbf{X}_m\right), \qquad m = 1, 2, \ldots, M$$

$$\mathbf{Y}_1 = sign\left(\mathbf{W}^T \mathbf{X}_1\right) = sign\left\{ \begin{bmatrix} 4 & 4 & 0 & 0 & 4 & 4 \\ 0 & 0 & 4 & 4 & 0 & 0 \\ 4 & 4 & 0 & 0 & 4 & 4 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \right\} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}$$

$$\mathbf{X}_m = sign\left(\mathbf{W}\,\mathbf{Y}_m\right), \qquad m = 1, 2, \ldots, M$$

$$\mathbf{X}_3 = sign\left(\mathbf{W}\,\mathbf{Y}_3\right) = sign\left\{ \begin{bmatrix} 4 & 0 & 4 \\ 4 & 0 & 4 \\ 0 & 4 & 0 \\ 0 & 4 & 0 \\ 4 & 0 & 4 \\ 4 & 0 & 4 \end{bmatrix} \begin{bmatrix} 1 \\ -1 \\ 1 \end{bmatrix} \right\} = \begin{bmatrix} 1 \\ 1 \\ -1 \\ -1 \\ 1 \\ 1 \end{bmatrix}$$

حافظه‌ی پیوندی دوطرفه

پایداری و ظرفیت ذخیره‌ی BAM

STABILITY AND STORAGE CAPACITY OF THE BAM

BAM پایدار غیرشرطی است.

یعنی اینکه هر مجموعه‌ای از پیوندها بدون نگرانی از ناپایداری می‌تواند یاد گرفته شود.

ظرفیت ذخیره‌ی BAM

حداکثر تعداد پیوندهای ذخیره شده‌ی BAM نباید از تعداد نرون‌ها لایه‌ی کوچکتر بیشتر شود.

مشکل جدی BAM

مشکل جدی‌تر BAM همگرایی نادرست است:
BAM همیشه نزدیک‌ترین پیوند را تولید نمی‌کند!

در واقع، یک پیوند پایدار ممکن است تنها اندکی به بردار ورودی اولیه وابستگی داشته باشد.

هوش مصنوعی

۶

یادگیری بدون نظارت

یادگیری بدون نظارت

UNSUPERVISED LEARNING

یادگیری بدون نظارت
به ناظر/مربی بیرونی نیازی ندارد.

در حین فرآیند آموزش، شبکه‌ی عصبی تعداد متفاوتی الگوی ورودی دریافت می‌کند،
ویژگی‌های مهم را در این الگوها کشف می‌کند،
و یاد می‌گیرد که چگونه داده‌های ورودی را در دسته‌های مناسب قرار دهد.

- یادگیری بدون نظارت تمایل دارد از سازمان‌دهی نروبیولوژیکی مغز پیروی کند.
- الگوریتم‌های یادگیری بدون نظارت، گرایش به یادگیری **سریع** و قابلیت استفاده‌ی **بلادرنگ** دارند.

یادگیری هبی

HEBBIAN LEARNING

قانون هب
Hebb's Law

اگر نرون *i* به طور مکرر در فعال‌سازی نرون *j* شرکت کند،
اتصال سیناپسی بین این دو نرون تقویت می‌شود، و
نرون *j* به تحریک‌های دریافت شده از نرون *i* حساس‌تر می‌شود.

دونالد هب (۱۹۴۹) این ایده‌ی کلیدی در یادگیری بیولوژیکی را مطرح کرد.

۱) اگر دو نرون در هر سمت یک اتصال به طور **همزمان فعال** شوند، وزن آن اتصال **افزایش** می‌یابد.

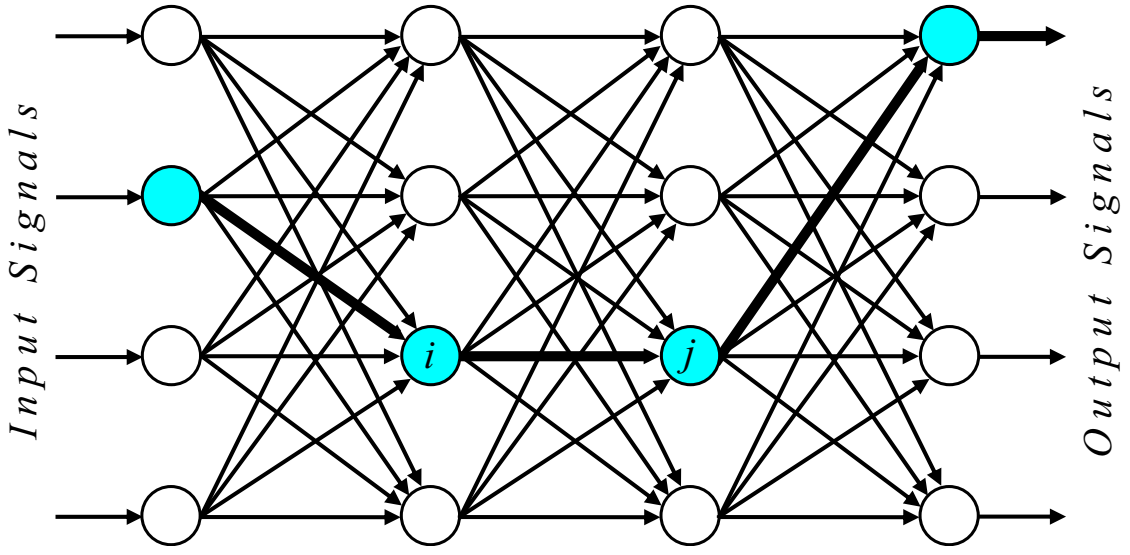
۲) اگر دو نرون در هر سمت یک اتصال به طور **ناهمزمان فعال** شوند، وزن آن اتصال **کاهش** می‌یابد.

قانون هب، پایه‌ای برای یادگیری بدون مربی فراهم می‌کند.
یادگیری در اینجا یک **پدیده‌ی محلی** است که بدون فیدبک از محیط انجام می‌شود.

یادگیری هبی

در یک شبکه‌ی عصبی

HEBBIAN LEARNING IN A NEURAL NETWORK



یادگیری هبی

قاعده‌ی ضرب فعالیت‌ها

ACTIVITY PRODUCT RULE

با استفاده از قانون هب، فرمول تنظیم وزن w_{ij} در تکرار p به صورت زیر نوشته می‌شود:

$$\Delta w_{ij}(p) = F[y_j(p), x_i(p)]$$

y_j: خروجی نرون j

x_i: ورودی نرون i

به عنوان یک حالت خاص، قانون هب می‌تواند به صورت زیر بازنمایی شود:

$$\Delta w_{ij}(p) = \alpha\, y_j(p)\, x_i(p)$$

(قانون ضرب فعالیت‌ها)

α: نرخ یادگیری

یادگیری هبی

قاعده‌ی ضرب فعالیت‌ها با فاکتور فراموشی

قانون هب اصلی، موجب می‌شود که وزن‌ها فقط افزایش پیدا کنند.
برای رفع این مشکل، می‌توانیم حدی بر روی رشد وزن‌های سیناپسی قرار بدهیم.
این کار با وارد کردن یک **فاکتور فراموشی غیرخطی** در قانون هب انجام می‌شود.

$$\Delta w_{ij}(p) = \alpha\, y_j(p)\, x_i(p) - \varphi\, y_j(p)\, w_{ij}(p)$$

فاکتور فراموشی
Forgetting Factor

$$0 < \varphi < 1$$

$$[0.01, 0.1]$$

الگوریتم یادگیری هبی

HEBBIAN LEARNING ALGORITHM

(۱) مقداردهی اولیه

وزن‌های سیناپسی و مقادیر آستانه را با اعداد تصادفی کوچک در $[0,1]$ مقداردهی می‌کنیم؛ و $p = 1$

(۲) فعال‌سازی

خروجی نرون در تکرار p–ام را محاسبه می‌کنیم. (n تعداد ورودی‌های نرون و θ_j مقدار آستانه‌ی نرون j)

$$y_j(p) = \sum_{i=1}^{n} x_i(p)\, w_{ij}(p) - \theta_j$$

(۳) آموزش وزن‌ها

وزن‌های شبکه را به‌هنگام می‌کنیم:

$$w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p)$$

تصحیح وزن‌ها با **قاعده‌ی دلتای تعمیم‌یافته (generalized delta rule)** انجام می‌شود:

$$\Delta w_{ij}(p) = \varphi\, y_j(p)[\lambda\ x_i(p) - w_{ij}(p)]$$

(۴) تکرار تا همگرایی

یک واحد به p اضافه می‌کنیم، به مرحله‌ی (۲) برمی‌گردیم و تکرار را تا رسیدن به همگرایی ادامه می‌دهیم.

الگوریتم یادگیری هبی

مثال

HEBBIAN LEARNING EXAMPLE

یک شبکه‌ی پیش‌خور با اتصال کامل با یک لایه از ۵ نرون محاسباتی را در نظر می‌گیریم.

هر نرون با یک مدل مک‌کلوچ و پیتز با تابع فعال‌سازی علامت بازنمایی می‌شود.

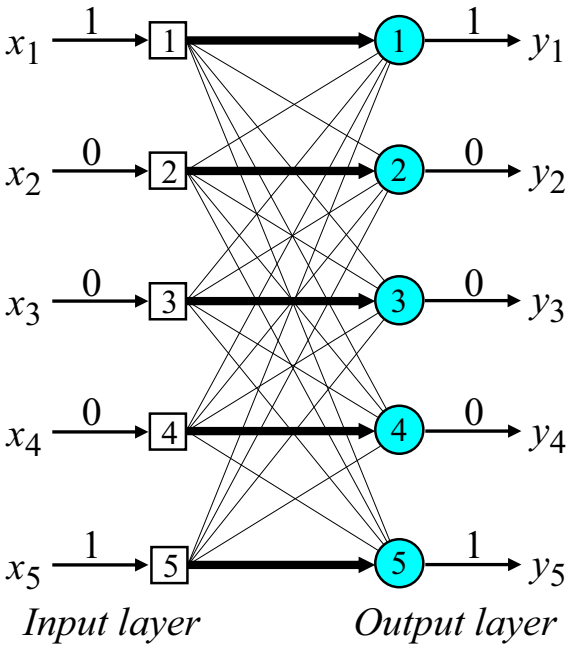
شبکه با مجموعه‌ی زیر از برداری ورودی آموزش می‌بیند:

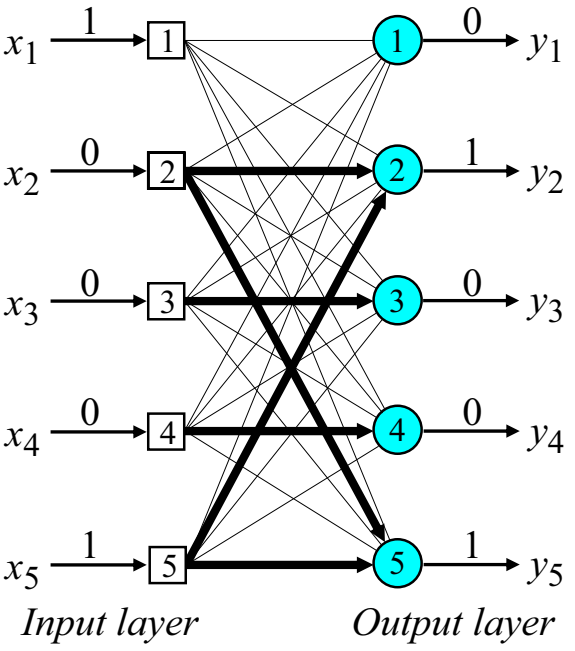
$$\mathbf{X}_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{X}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix} \quad \mathbf{X}_3 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} \quad \mathbf{X}_4 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{X}_5 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

الگوریتم یادگیری هبی

مثال: حالت‌های آغازین و نهایی شبکه

INITIAL AND FINAL STATES OF THE NETWORK





الگوریتم یادگیری هبی

مثال: ماتریس‌های وزن آغازین و نهایی

INITIAL AND FINAL WEIGHT MATRICES

Input layer \ Output layer	1	2	3	4	5
1	1	0	0	0	0
2	0	1	0	0	0
3	0	0	1	0	0
4	0	0	0	1	0
5	0	0	0	0	1

Input layer \ Output layer	1	2	3	4	5
1	0	0	0	0	0
2	0	2.0204	0	0	2.0204
3	0	0	1.0200	0	0
4	0	0	0	0.9996	0
5	0	2.0204	0	0	2.0204

الگوریتم یادگیری هبی

مثال: آزمایش شبکه

بردار ورودی آزمایشی $\mathbf{X}$ را به صورت مقابل در نظر می‌گیریم:

$$\mathbf{X} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$\mathbf{Y} = sign\left(\mathbf{W}\,\mathbf{X} - \boldsymbol{\theta}\right)$$

اگر بردار $\mathbf{X}$ به شبکه نشان داده شود، خروجی می‌شود:

$$\mathbf{Y} = sign\left\{ \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 2.0204 & 0 & 0 & 2.0204 \\ 0 & 0 & 1.0200 & 0 & 0 \\ 0 & 0 & 0 & 0.9996 & 0 \\ 0 & 2.0204 & 0 & 0 & 2.0204 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} - \begin{bmatrix} 0.4940 \\ 0.2661 \\ 0.0907 \\ 0.9478 \\ 0.0737 \end{bmatrix} \right\} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

هوش مصنوعی

۷

یادگیری رقابتی

یادگیری رقابتی

COMPETITIVE LEARNING

در یادگیری رقابتی
نرون‌ها برای فعال شدن با یکدیگر رقابت می‌کنند.

در حالی که در یادگیری هبی، نرون‌های خروجی مختلف می‌توانند همزمان با هم فعال شوند،
در یادگیری رقابتی، تنها یک نرون واحد در یک زمان فعال می‌شود.

نرون خروجی که «رقابت» را می‌برد، نرون «برنده–همه–را–می‌خورد» نامیده می‌شود.

برنده همه را می‌خورد
Winner-Takes-All

نقشه‌های ویژگی خود–سازمان‌ده

SELF-ORGANIZING FEATURE MAPS

ایده‌ی پایه‌ی **یادگیری رقابتی** در اواخر دهه‌ی ۱۹۷۰ معرفی شد.

در اواخر دهه‌ی ۱۹۸۰، کوهونن طبقه‌ی خاصی از شبکه‌های عصبی مصنوعی را با عنوان
نقشه‌های ویژگی خود–سازمان‌ده
معرفی کرد که بر اساس یادگیری رقابتی طراحی شده بود.

مغز انسان تحت سلطه‌ی یک غشای مغزی است
که یک ساختار بسیار پیچیده از میلیاردها نرون و صدها میلیون سیناپس است.

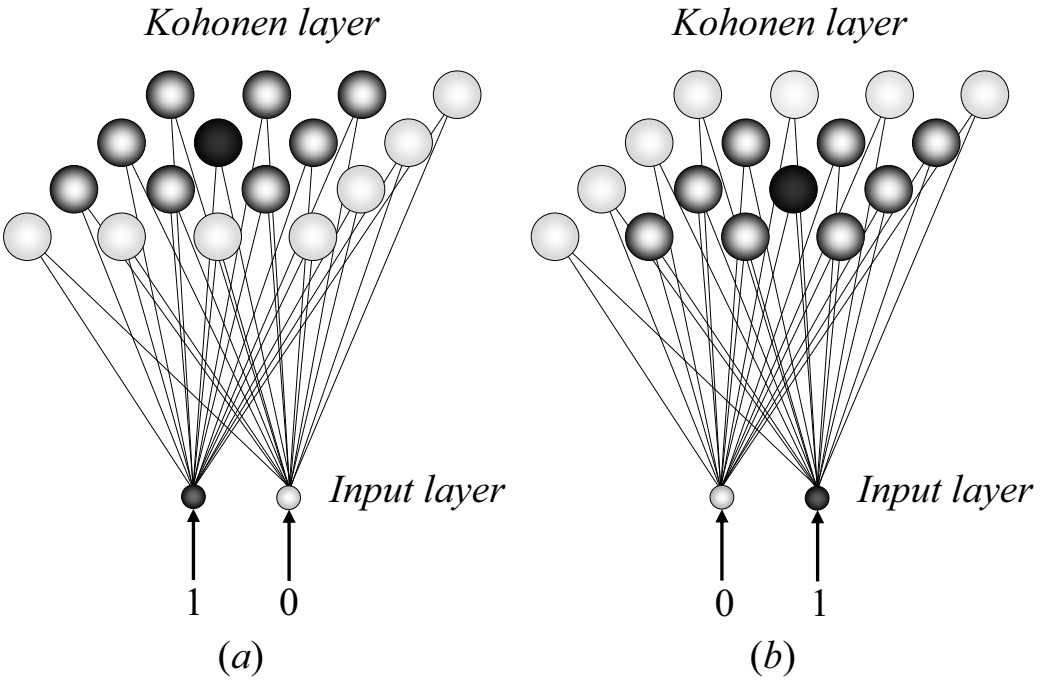
غشاء (cortex) حاوی ناحیه‌هایی است که مسئول فعالیت‌های مختلف انسانی هستند
(موتوری، بینایی، شنوایی، لامسه، ...)
و به ورودی‌های حسگری متفاوتی پیوند خورده‌اند.

می‌توانیم بگوییم هر ورودی حسگری به ناحیه‌ی متناظر در غشای مغزی نگاشت می‌یابد.
کورتکس، یک نگاشت محاسباتی خود–سازمان‌ده در مغز انسان است.

نقشه‌های ویژگی خود-سازمان‌ده

مدل نگاشت ویژگی کوهونن

FEATURE-MAPPING KOHONEN MODEL



هوش مصنوعی

۸

شبکه‌ی کوهونن

شبکه‌ی کوهونن

THE KOHONEN NETWORK

شبکه‌ی کوهونن

Kohonen Network

یک شبکه‌ی عصبی رقابتی

مدل کوهونن یک نقشه‌ی توپولوژیکی را فراهم می‌کند.
این نقشه، تعداد ثابتی از الگوهای ورودی از لایه‌ی ورودی را
در یک لایه‌ی خروجی با بعد بالاتر (لایه‌ی کوهونن) قرار می‌دهد.

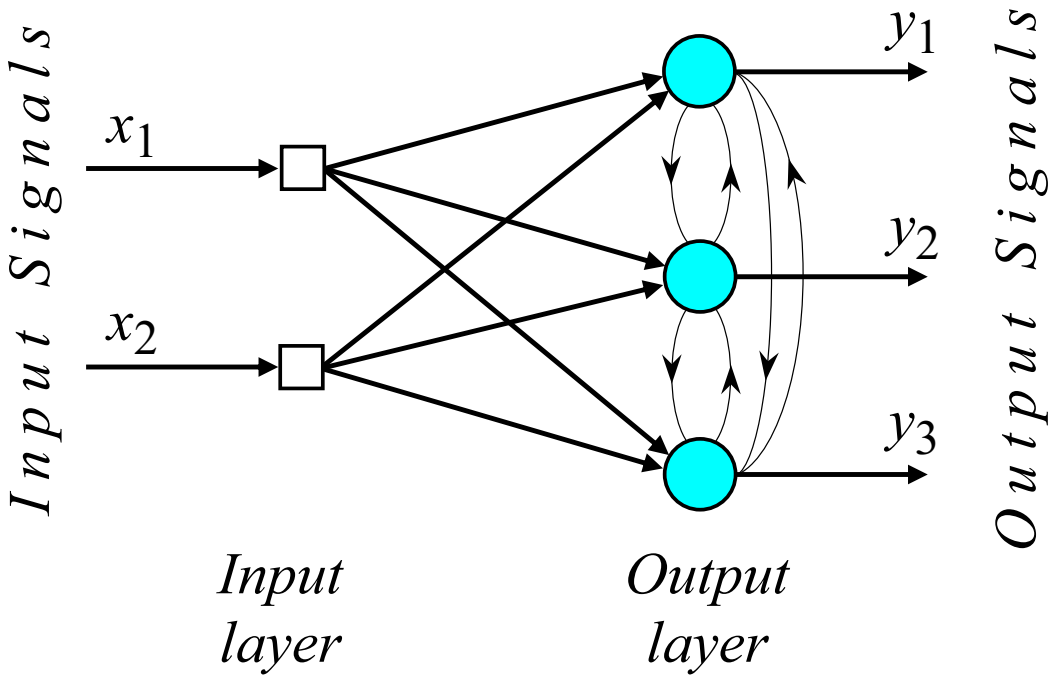
طراحی شده توسط تیوو کوهونن (دهه ۱۹۸۰)

آموزش در شبکه‌ی کوهونن با یک همسایگی به اندازه‌ی نسبتاً بزرگ از برنده شروع می‌شود و
با پیشرفت آموزش، اندازه‌ی همسایگی به‌تدریج کاهش می‌یابد.

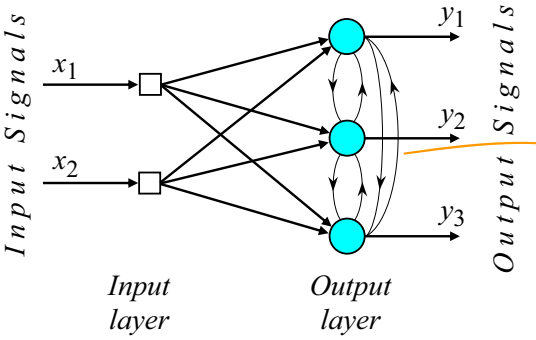
شبکه‌ی کوهونن

معماری یک شبکه‌ی کوهونن

ARCHITECTURE OF THE KOHONEN NETWORK



شبکه‌ی کوهونن



- اتصالات جانبی بین نرون‌ها برای ایجاد رقابت بین نرون‌ها به‌کار می‌رود.
- نرون با بزرگ‌ترین سطح فعال‌سازی بین همه‌ی نرون‌ها در لایه‌ی خروجی، برنده می‌شود.
- نرون برنده تنها نرونی است که یک سیگنال خروجی تولید می‌کند.
- فعالیت سایر نرون‌ها در رقابت سرکوب می‌شود.

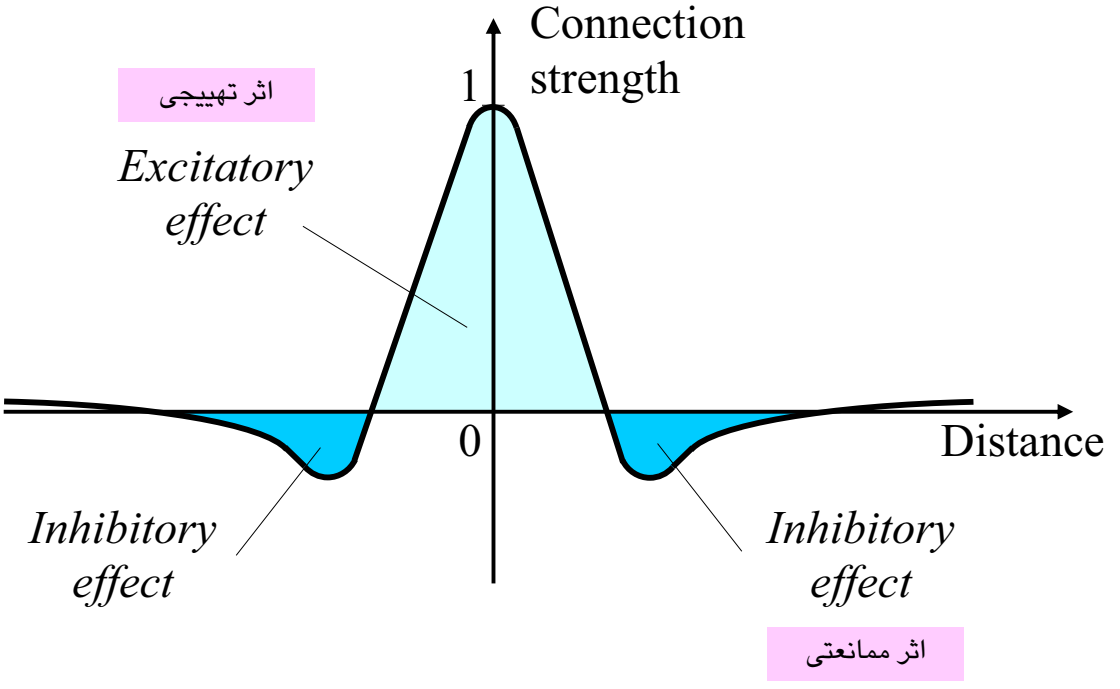
اتصالات فیدبک جانبی اثر **تهییجی** یا **ممانعتی** دارند
(بسته به فاصله تا نرون برنده)

این اثر با استفاده از یک **تابع کلاه مکزیکی** به‌دست می‌آید
که وزن‌های سیناپسی بین نرون‌ها در لایه‌ی کوهونن را توصیف می‌کند.

شبکه‌ی کوهونن

تابع کلاه مکزیکی (برای اتصالات جانبی)

THE MEXICAN HAT FUNCTION OF LATERAL CONNECTION



شبکه‌ی کوهونن

قاعده‌ی یادگیری رقابتی

COMPETITIVE LEARNING RULE

در شبکه‌ی کوهونن،
یک نرون با شیفت دادن وزن‌هایش از اتصالات غیرفعال به اتصالات فعال یاد می‌گیرد.

فقط نرون برنده و همسایگانش اجازه‌ی یادگیری دارند.

اگر یک نرون به یک الگوی ورودی داده شده پاسخی ندهد،
در این صورت یادگیری در آن نرون خاص نمی‌تواند اتفاق بیفتد.

قاعده‌ی یادگیری رقابتی:
تعریف تغییر وزن اعمال شده به یک وزن سیناپسی:

نرخ یادگیری — سیگنال ورودی

$$\Delta w_{ij} = \begin{cases} \alpha (x_i - w_{ij}), & \text{if neuron } j \text{ wins the competition} \\ 0, & \text{if neuron } j \text{ loses the competition} \end{cases}$$

شبکه‌ی کوهونن

اثر کلی قاعده‌ی یادگیری رقابتی

اثر کلی قاعده‌ی یادگیری رقابتی،
در حرکت دادن بردار وزن سیناپسی نرون برنده به سمت بردار الگوی ورودی است.

ضابطه‌ی تطابق، معادل با **می‌نیمم فاصله‌ی اقلیدسی** بین بردارهاست:

$$d = \left\| \mathbf{X} - \mathbf{W}_j \right\| = \left[\sum_{i=1}^{n} (x_i - w_{ij})^2 \right]^{1/2}$$

بردار ورودی $\mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$

بردار وزن نرون j $\mathbf{W}_j = \begin{bmatrix} w_{1j} \\ w_{2j} \\ \vdots \\ w_{nj} \end{bmatrix}$

شبکه‌ی کوهونن

تعیین نرون برنده

برای تعیین نرون برنده،
که بهترین تطابق را با بردار ورودی دارد،
از می‌نیمم فاصله‌ی اقلیدسی استفاده می‌کنیم:

$$j_{\mathbf{X}} = \min_{j} \left\| \mathbf{X} - \mathbf{W}_j \right\|, \qquad j = 1, 2, \ldots, m$$

نرون برنده

تعداد نرون‌ها
در لایه‌ی کوهونن

شبکه‌ی کوهونن

مثال (۱ از ۳)

برای نمونه، فرض می‌کنیم یک بردار ورودی دوبعدی
به یک شبکه‌ی کوهونن با سه نرون نشان داده شود:

$$\mathbf{X} = \begin{bmatrix} 0.52 \\ 0.12 \end{bmatrix}$$

بردارهای وزن اولیه به صورت زیر داده شده است:

$$\mathbf{W}_1 = \begin{bmatrix} 0.27 \\ 0.81 \end{bmatrix} \qquad \mathbf{W}_2 = \begin{bmatrix} 0.42 \\ 0.70 \end{bmatrix} \qquad \mathbf{W}_3 = \begin{bmatrix} 0.43 \\ 0.21 \end{bmatrix}$$

شبکه‌ی کوهونن

مثال (۲ از ۳)

برای یافتن نرون برنده (بهترین تطابق)،
از ضابطه‌ی می‌نیمم فاصله‌ی اقلیدسی استفاده می‌کنیم.

$$d_1 = \sqrt{(x_1 - w_{11})^2 + (x_2 - w_{21})^2} = \sqrt{(0.52 - 0.27)^2 + (0.12 - 0.81)^2} = 0.73$$

$$d_2 = \sqrt{(x_1 - w_{12})^2 + (x_2 - w_{22})^2} = \sqrt{(0.52 - 0.42)^2 + (0.12 - 0.70)^2} = 0.59$$

$$d_3 = \sqrt{(x_1 - w_{13})^2 + (x_2 - w_{23})^2} = \sqrt{(0.52 - 0.43)^2 + (0.12 - 0.21)^2} = 0.13$$

نرون 3 رقابت را می‌برد،
پس وزن آن یعنی $\mathbf{W}_3$ باید با توجه به قاعده‌ی یادگیری رقابتی به‌هنگام شود:

$$\Delta w_{13} = \alpha (x_1 - w_{13}) = 0.1 (0.52 - 0.43) = 0.01$$

$$\Delta w_{23} = \alpha (x_2 - w_{23}) = 0.1 (0.12 - 0.21) = -0.01$$

شبکه‌ی کوهونن

مثال (۳ از ۳)

نرون 3 رقابت را می‌برد،
پس وزن آن یعنی $\mathbf{W}_3$ باید با توجه به قاعده‌ی یادگیری رقابتی به‌هنگام شود:

$$\Delta w_{13} = \alpha (x_1 - w_{13}) = 0.1(0.52 - 0.43) = 0.01$$

$$\Delta w_{23} = \alpha (x_2 - w_{23}) = 0.1(0.12 - 0.21) = -0.01$$

پس بردار وزن در تکرار بعدی به صورت زیر تعیین می‌شود:

$$\mathbf{W}_3(p+1) = \mathbf{W}_3(p) + \Delta\mathbf{W}_3(p) = \begin{bmatrix} 0.43 \\ 0.21 \end{bmatrix} + \begin{bmatrix} 0.01 \\ -0.01 \end{bmatrix} = \begin{bmatrix} 0.44 \\ 0.20 \end{bmatrix}$$

بردار وزن $\mathbf{W}_3$ مربوط به نرون برنده (3) در هر تکرار به بردار ورودی $\mathbf{X}$ نزدیک‌تر می‌شود.

شبکه‌ی کوهونن

الگوریتم یادگیری رقابتی

COMPETITIVE LEARNING ALGORITHM

(۱) مقداردهی اولیه

- وزن‌های سیناپسی و مقادیر آستانه را با اعداد تصادفی کوچک در $[0,1]$ مقداردهی می‌کنیم؛ و $p = 1$
- به پارامتر نرخ یادگیری α یک مقدار کوچک مثبت نسبت می‌دهیم.

(۲) فعال‌سازی و شباهت‌سنجی

با اعمال بردار ورودی $\mathbf{X}$ شبکه‌ی کوهونن را فعال می‌کنیم و بر اساس ضابطه‌ی می‌نیمم فاصله‌ی اقلیدسی نرون برنده در تکرار p–ام را می‌یابیم. (n تعداد نرون‌های لایه‌ی ورودی، m تعداد نرون‌های لایه‌ی کوهونن)

$$j_{\mathbf{X}}(p) = \min_{j} \left\| \mathbf{X} - \mathbf{W}_j(p) \right\| = \left\{ \sum_{i=1}^{n} [x_i - w_{ij}(p)]^2 \right\}^{1/2}, \qquad j = 1, 2, \ldots, m$$

(۳) آموزش وزن‌ها

وزن‌های شبکه را به‌هنگام می‌کنیم (تصحیح وزن‌ها با **قاعده‌ی یادگیری رقابتی** انجام می‌شود):

$$w_{ij}(p+1) = w_{ij}(p) + \Delta w_{ij}(p) \qquad \Delta w_{ij}(p) = \begin{cases} \alpha [x_i - w_{ij}(p)], & j \in \Lambda_j(p) \\ 0, & j \notin \Lambda_j(p) \end{cases}$$

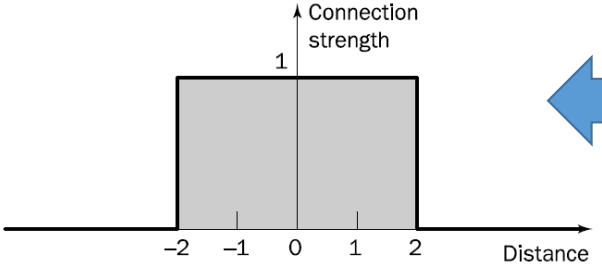
(۴) تکرار تا همگرایی

یک واحد به p اضافه می‌کنیم، به مرحله‌ی (۲) برمی‌گردیم و تکرار را ادامه می‌دهیم تا ۱. به می‌نیمم فاصله‌ی اقلیدسی برسیم یا ۲. تغییر قابل توجهی در نقشه‌ی ویژگی رخ ندهد.

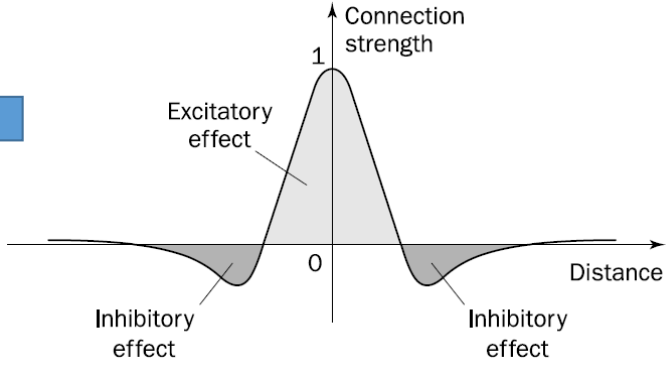
شبکه‌ی کوهونن

تابع همسایگی

NEIGHBOURHOOD FUNCTION



تابع همسایگی مستطیلی



$$\Delta w_{ij}(p) = \begin{cases} \alpha\left[x_i - w_{ij}(p)\right], & j \in \Lambda_j(p) \\ 0, & j \notin \Lambda_j(p) \end{cases}$$

شبکه‌ی کوهونن

مثال: یادگیری رقابتی در شبکه‌ی کوهونن (۱ از ۶)

COMPETITIVE LEARNING IN THE KOHONEN NETWORK

یک شبکه‌ی کوهونن با ۱۰۰ نرون را در نظر می‌گیریم
که به صورت یک مشبکه‌ی ۱۰ سطر در ۱۰ ستون آرایش یافته‌اند.

می‌خواهیم این شبکه بردارهای ورودی دوبعدی را خوشه‌بندی کند.
(هر نرون در شبکه باید فقط به بردارهای ورودی واقع در محدوده‌ی خودش پاسخ بدهد)

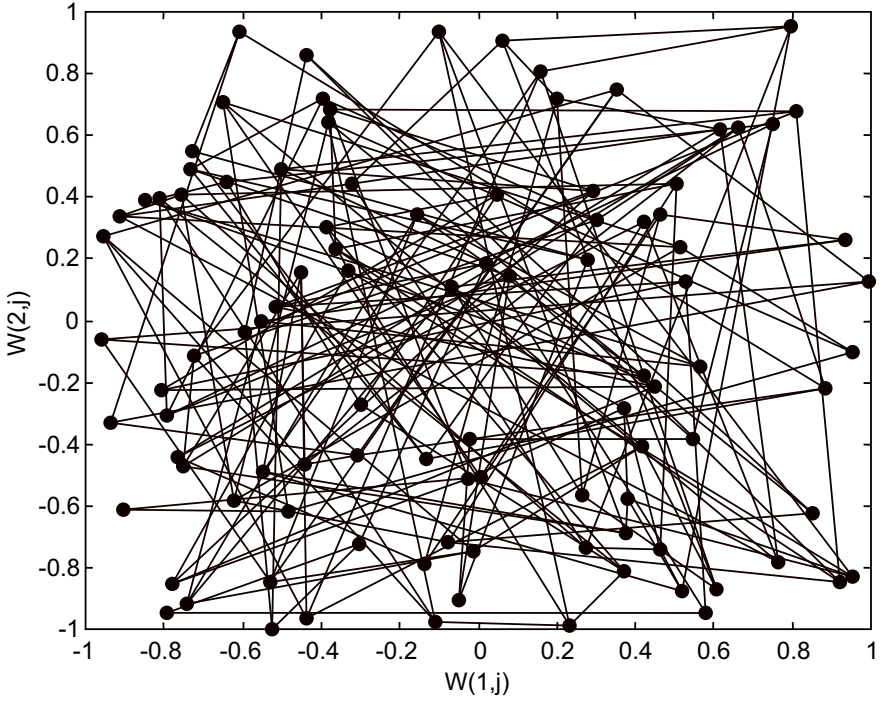
شبکه با ۱۰۰۰ بردار ورودی دوبعدی
تولید شده به صورت تصادفی در یک ناحیه‌ی مربعی بین 1- و 1+
آموزش داده می‌شود.

پارامتر نرخ یادگیری اولیه:
$$\alpha = 0.1$$

شبکه‌ی کوهونن

مثال: یادگیری رقابتی در شبکه‌ی کوهونن (۲ از ۶): وزن‌های آغازین تصادفی

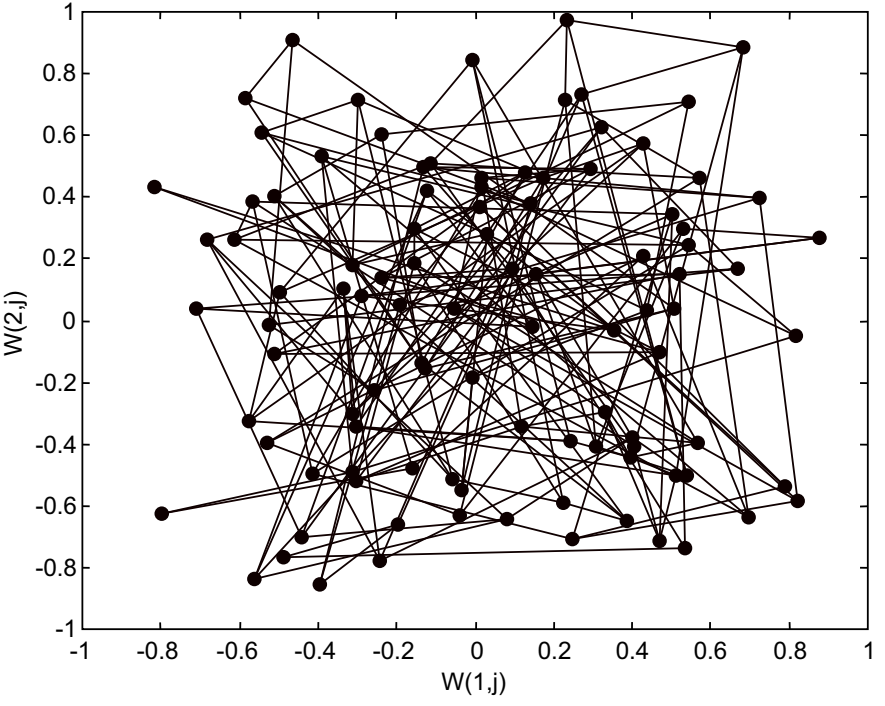
INITIAL RANDOM WEIGHTS



شبکه‌ی کوهونن

مثال: یادگیری رقابتی در شبکه‌ی کوهونن (۳ از ۶): شبکه پس از ۱۰۰ تکرار

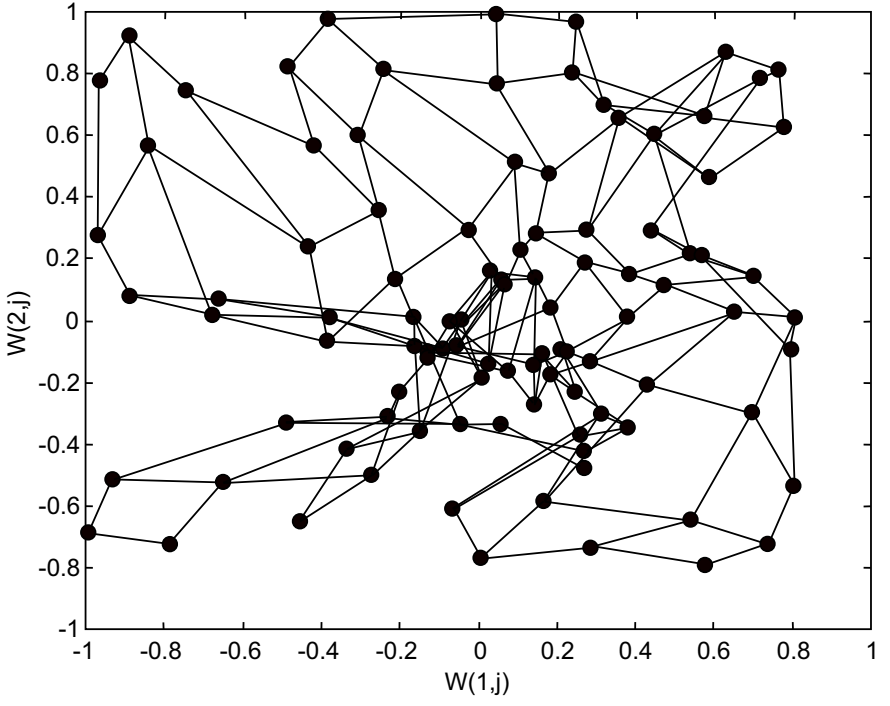
NETWORK AFTER 100 ITERATIONS



شبکه‌ی کوهونن

مثال: یادگیری رقابتی در شبکه‌ی کوهونن (۴ از ۶): شبکه پس از ۱۰۰۰ تکرار

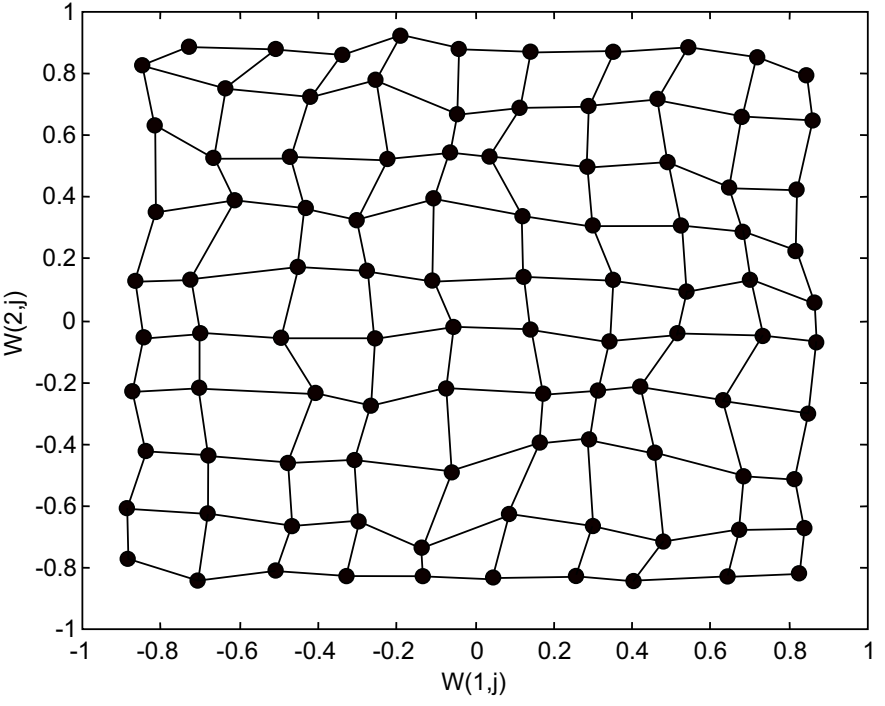
NETWORK AFTER 1000 ITERATIONS



شبکه‌ی کوهونن

مثال: یادگیری رقابتی در شبکه‌ی کوهونن (۵ از ۶): شبکه پس از ۱۰۰۰۰ تکرار

NETWORK AFTER 10,000 ITERATIONS

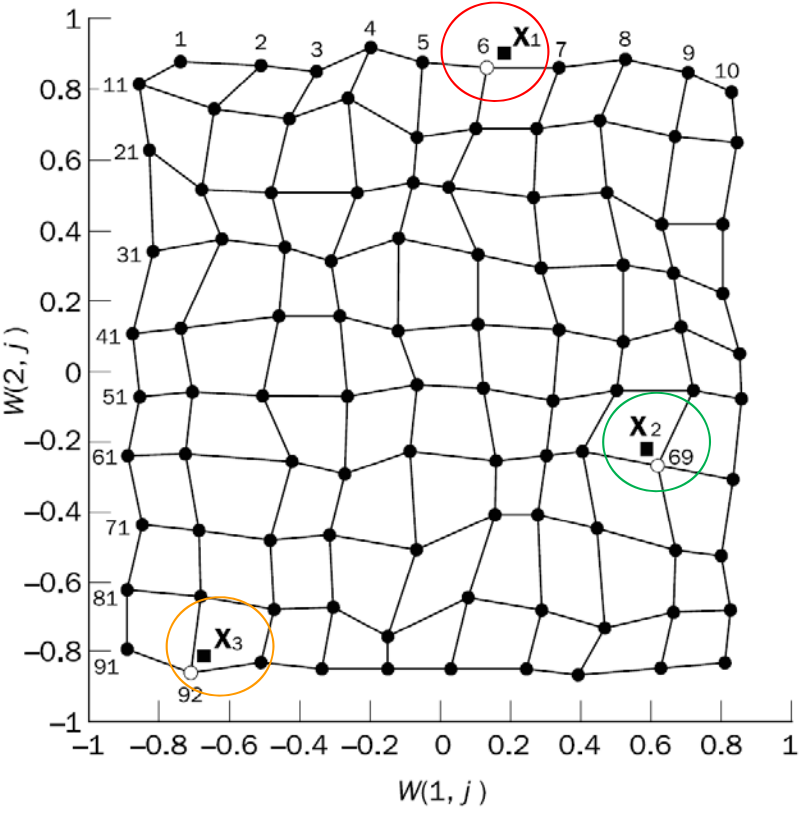


شبکه‌ی کوهونن

مثال: یادگیری رقابتی در شبکه‌ی کوهونن (۶ از ۶): پاسخ شبکه به سه ورودی جدید

NETWORK AFTER 10,000 ITERATIONS

$$X_1 = \begin{bmatrix} 0.2 \\ 0.9 \end{bmatrix} \quad X_2 = \begin{bmatrix} 0.6 \\ -0.2 \end{bmatrix} \quad X_3 = \begin{bmatrix} -0.7 \\ -0.8 \end{bmatrix}$$



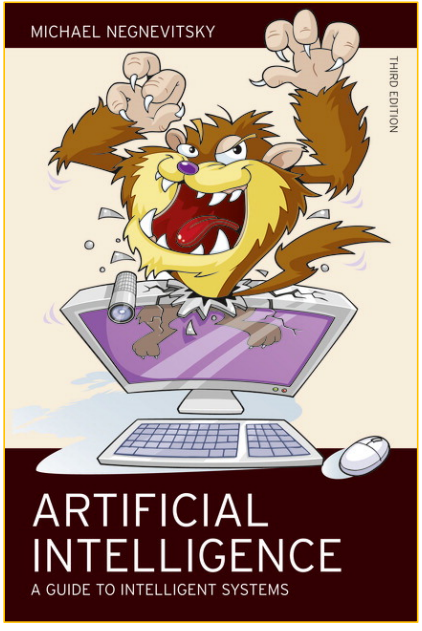
هوش مصنوعی پیشرفته

شبکه‌های عصبی مصنوعی

۹

منابع، مطالعه، تکلیف

منبع اصلی



Michael Negnevitsky,
Artificial Intelligence: A Guide to Intelligent Systems,
Pearson Education Canada, 2011.
Chapter 6

Artificial neural networks 6

In which we consider how our brains work and how to build and train artificial neural networks.

6.1 Introduction, or how the brain works

'The computer hasn't proved anything yet,' angry Garry Kasparov, the world chess champion, said after his defeat in New York in May 1997. 'If we were playing a real competitive match, I would tear down Deep Blue into pieces.'

But Kasparov's efforts to downplay the significance of his defeat in the six-game match was futile. The fact that Kasparov – probably the greatest chess player the world has seen – was beaten by a computer marked a turning point in the quest for intelligent machines.

The IBM supercomputer called Deep Blue was capable of analysing 200 million positions a second, and it appeared to be displaying intelligent thoughts. At one stage Kasparov even accused the machine of cheating!

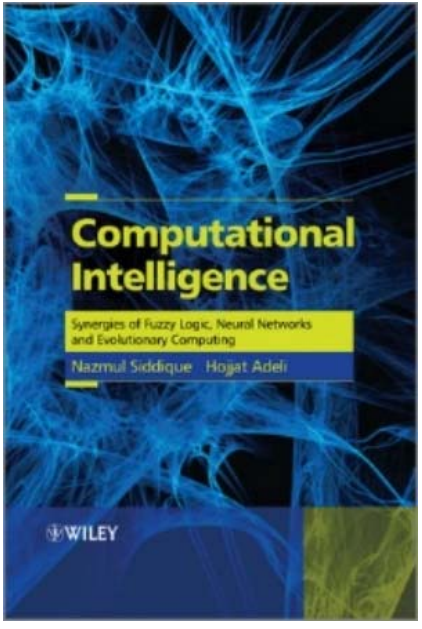
> 'There were many, many discoveries in this match, and one of them was that sometimes the computer plays very, very human moves.
> It deeply understands positional factors. And that is an outstanding scientific achievement.'

Traditionally, it has been assumed that to beat an expert in a chess game, a computer would have to formulate a strategy that goes beyond simply doing a great number of 'look-ahead' moves per second. Chess-playing programs must be able to improve their performance with experience or, in other words, a machine must be capable of learning.

What is machine learning?

In general, machine learning involves adaptive mechanisms that enable computers to learn from experience, learn by example and learn by analogy. Learning capabilities can improve the performance of an intelligent system over time. Machine learning mechanisms form the basis for adaptive systems. The most popular approaches to machine learning are **artificial neural networks** and **genetic algorithms**. This chapter is dedicated to neural networks.

منبع دوم



Nazmul Siddique, Hojjat Adeli,
Computational Intelligence: Synergies of Fuzzy Logic, Neural Networks and Evolutionary Computing,
John Wiley & Sons, 2013.
Chapter 4

4

Neural Networks

4.1 Introduction

The study of the human brain is hundreds of years old. Advances in brain research promise an initial understanding of the mechanism of cognitive process in the brain. This shows that the brain stores information as patterns. Some of these patterns are very complicated, for example the ability to recognize individual faces from different angles. This process of storing information as patterns, utilizing those patterns, and then solving problems encompasses a new field in computing, which does not utilize traditional programming. This involves the creation of massively parallel networks and the training of those networks to solve specific tasks.

The exact workings of the human brain are still a mystery. Yet, some aspects of this amazing processor are known. In particular, the most basic element of the human brain is a specific type of cell, which provides us with our abilities to remember, think and apply previous experiences to our every action. These cells, all approximately 100 billion of them, are known as neurons. Each of these neurons can connect with up to 200,000 other neurons, although 1,000 to 10,000 is typical. The individual neurons convey information via a host of electrochemical pathways. Together, these neurons and their connections form a process which is not binary, not stable and not synchronous. This building block of the human brain has a few general capabilities. Basically, a biological neuron receives inputs from other sources, combines them in some way, performs a generally nonlinear operation on the result and then outputs the final result. Figure 4.1 shows a biological neuron and Figure 4.2 shows the four main functional parts and their relationships in a neuron. Recent experimental data has provided further evidence that biological neurons are structurally more complex than the simplistic explanation above.

The first model of artificial neural networks came in 1943 when Warren McCulloch, a neurophysiologist and Walter Pitts, a young mathematician outlined the first formal model of an elementary computing neuron (McCulloch and Pitts, 1943). McCulloch and Pitts' artificial

Computational Intelligence: Synergies of Fuzzy Logic, Neural Networks and Evolutionary Computing, First Edition.
Nazmul Siddique and Hojjat Adeli.