دانشگاه علم و صنعت
دانشکده مهندسی کامپیوتر

شبکه عصبی مصنوعی

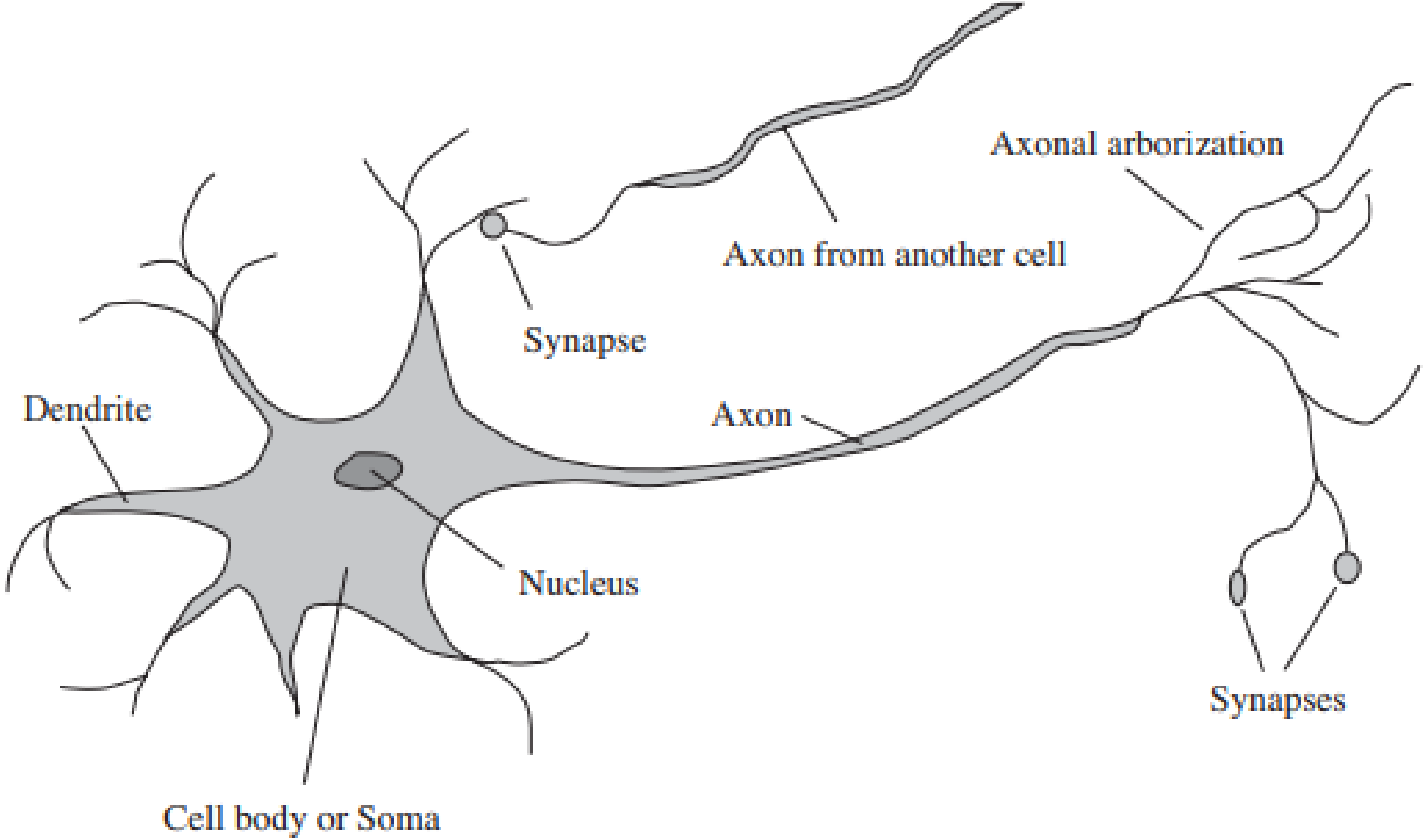
دانشگاه علم و صنعت
دانشکده مهندسی کامپیوتر

نیم سال اول ۱۴۰۱-۱۴۰۲

شبکه عصبی

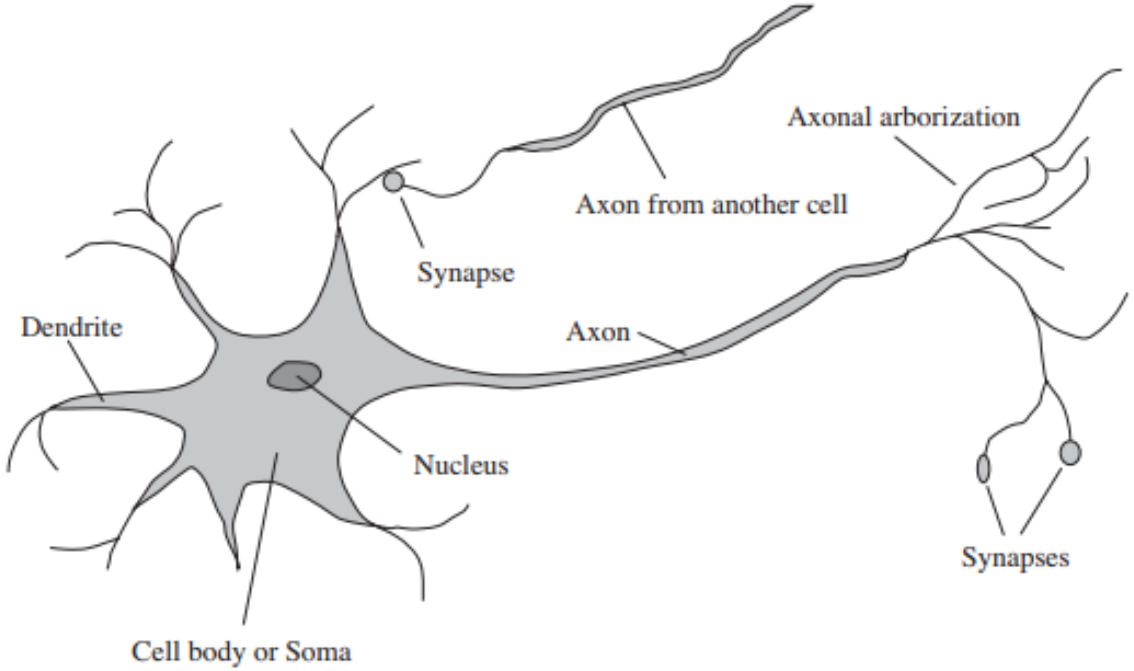
- ادامه بحث یادگیری ماشین
- پیچیده‌ترین یادگیرنده در طبیعت: مغز (انسان)
- مغز چگونه یاد می‌گیرد؟
- آیا می‌توان با الهام از یادگیری مغز، مدلی محاسباتی/ریاضیاتی تولید کرد؟

- شبکه‌های عصبی مصنوعی


Axonal arborization
Axon from another cell
Synapse
Dendrite
Axon
Nucleus
Synapses
Cell body or Soma

ساختار سلول‌های عصبی مغز

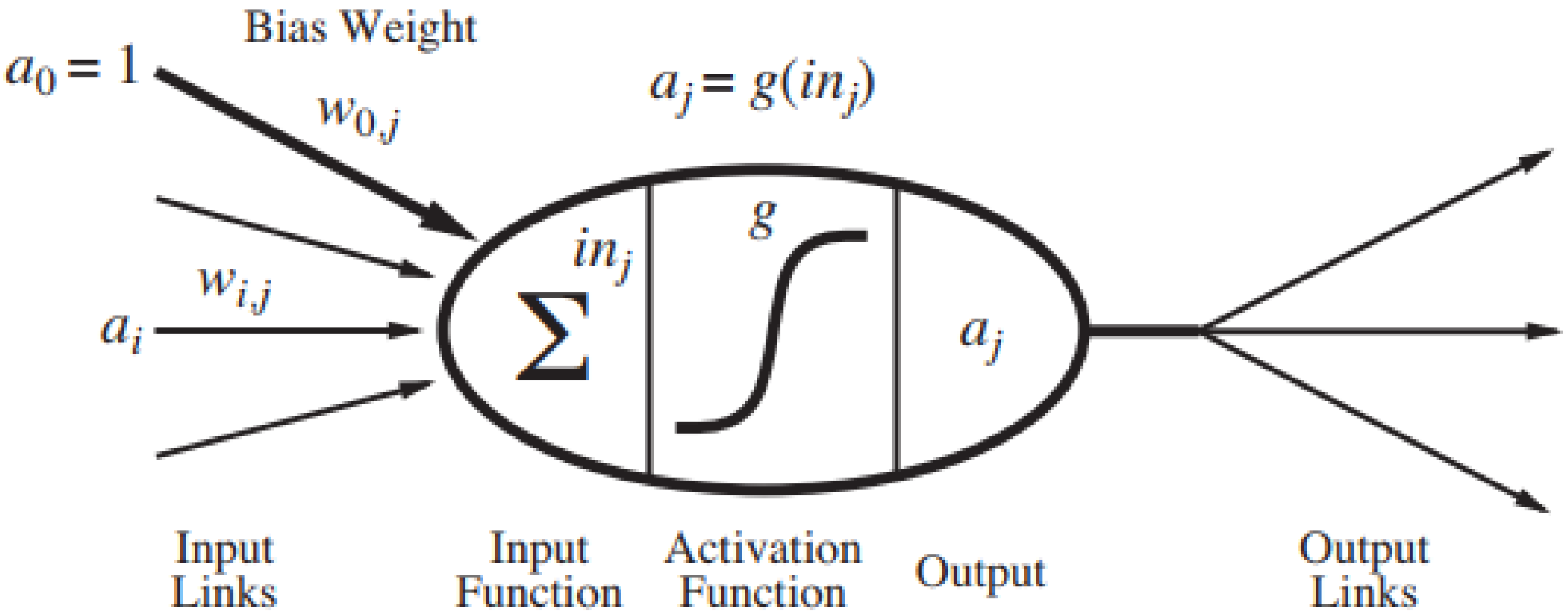
- نورون: سلول عصبی
- هر سلول عصبی از هسته و بدنه سلول (سوما) تشکیل شده است.
- تعدادی دندانه فیبری از سلول خارج شده است (دندریت‌ها) که ورودی‌های سلول را تشکیل می‌دهند
- یک دندریت خیلی بلند از هر نورون خارج شده است (آکسون) که خروجی سلول است.
  - طول آکسون: یک سانتی متر تا یک متر!
- دندریت‌ها و آکسون ارتباط بین سلول‌ها را برعهده دارند.
- هر نورون با ۱۰ تا ۱۰۰۰۰۰ نورون دیگر ارتباط دارد
  - محل ارتباطات: سیناپس
- از طریق همین ارتباط‌ها، پیغام‌ها یا سیگنال‌هایی بین سلول‌های عصبی جابجا می‌شود.

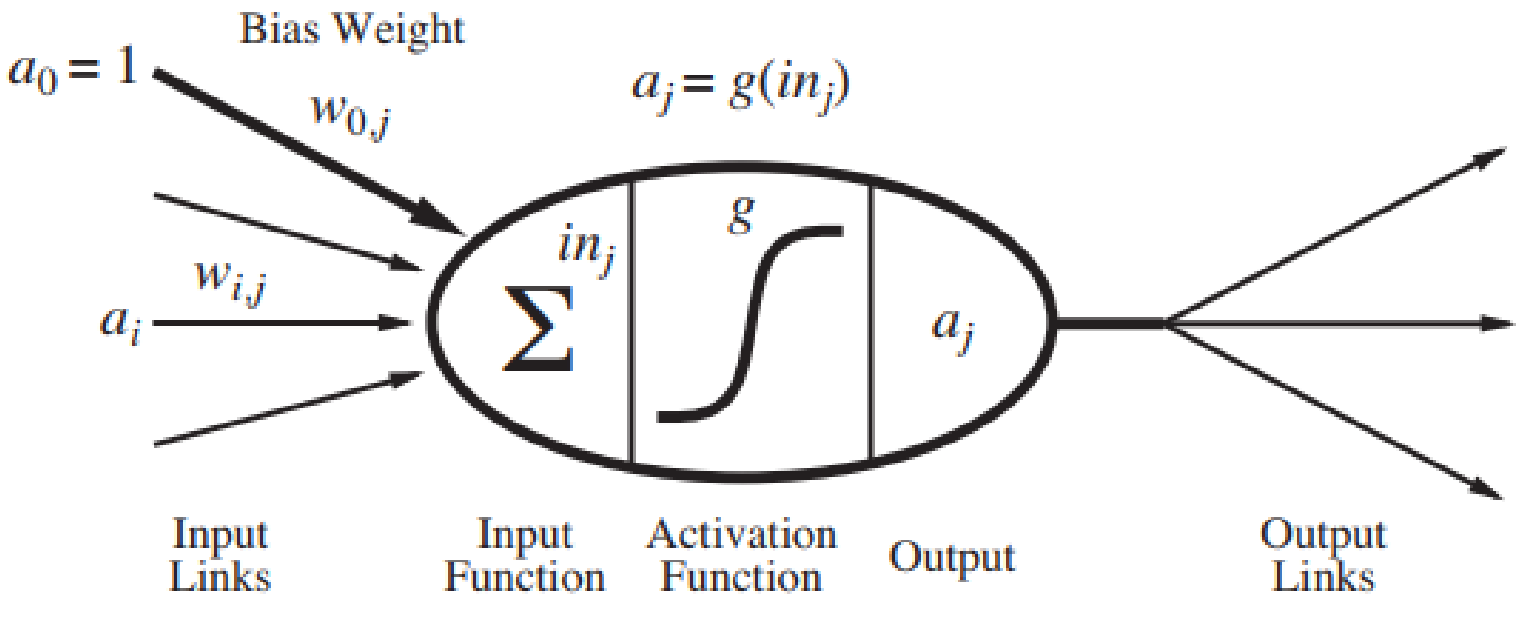


- سیگنال‌ها بین نورون ها با یک سری فعل و انفعالات پیچیده الکتروشیمیایی انتقال می‌یابند.
- این سیگنال‌ها، فعالیت مغز را کنترل می‌کنند (کوتاه مدت).
- همین سیگنال‌ها، اتصالات بین نورون‌ها را نیز تغییر می‌دهند (بلند مدت)
- مغز انسان حدود ۶۰۸ میلیارد نورون دارد.

- فعال شدن خروجی یک سلول (از طریق آکسون) یک پیغام عصبی را به سلول‌های بعدی منتقل می‌کند.
- لازمه فعال شدن خروجی سلول، رسیدن مجموع ورودی‌ها (از طریق دندریت‌ها) به یک حد مشخص است.
  - ورودی یا خروجی (الکتروشیمیایی):
    - اختلاف ولتاژ از طریق انتقال یونهای سدیم و پتاسیم
    - انتقال از طریق مولکول‌های neurotransmitter

شبیه سازی نورون





- نورون j طبق رابطه زیر فعال می‌شود:

$$a_j = g\left(\sum_{i=0}^{n} w_{i,j} a_i\right)$$

i اندیس نورون های قبلی است

W وزن لینک بین نورون ها است.

- نورون وقتی فعال می‌شود که ترکیب خطی ورودی‌هایش از یک حد دقیق یا تقریبی بیشتر شود.
- یک شبکه عصبی، مجموعه‌ای از نورون‌هاست که با ساختاری معین به هم متصل هستند.

ساختار شبکه عصبی

- شبکه عصبی از واحدها (گره‌ها- نورون‌ها) یی متصل به هم تشکیل شده است.
- یک لینک از واحد i به j به معنای انتقال مقدار فعالسازی a_i از واحد i به j است.
- هر لینک، یک مقدار عددی به نام وزن $w_{i,j}$ دارد که قدرت و علامت اتصال را نشان می‌دهد.
- هر واحد یک بایاس نیز دارد که از طریق فرض کردن یک ورودی اضافی به مقدار $a_0 = 1$ و وزن $w_{0,j}$ مدل می‌شود.
- هر واحد j ابتدا جمع وزن‌دار ورودی هایش را محاسبه می‌کند:

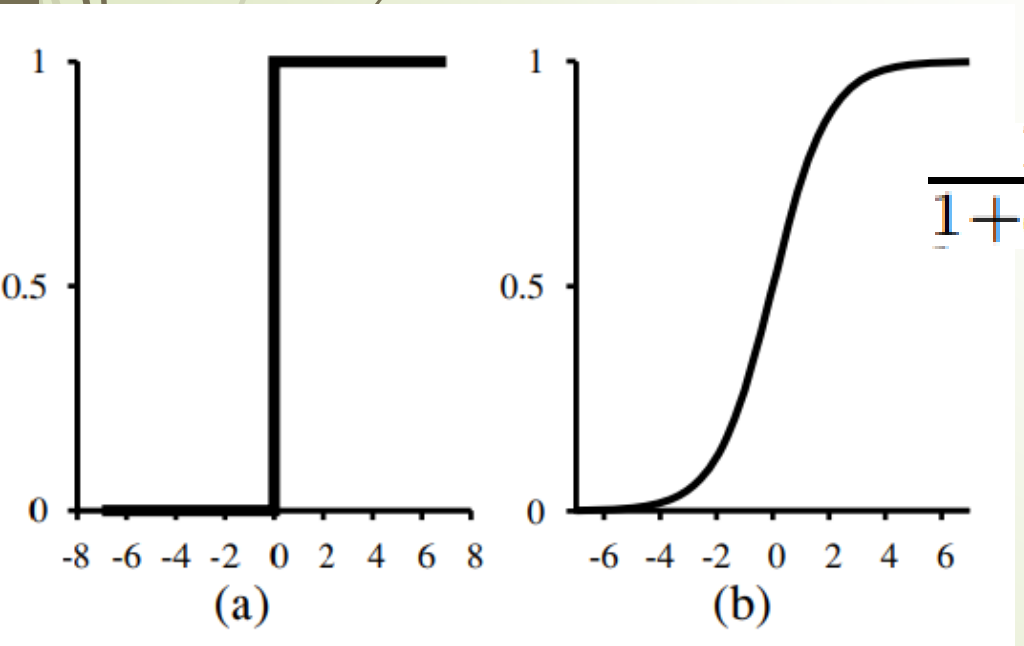
$$in_j = \sum_{i=0}^{n} w_{i,j} a_i$$

- سپس خروجیش را از طریق اعمال یک تابع فعال‌ساز به جمع ورودی‌ها به دست می‌آورد:

$$a_j = g(in_j) = g\left(\sum_{i=0}^{n} w_{i,j} a_i\right)$$

تابع فعال‌ساز

- تابع فعالسازی g یا دارای آستانه سخت (شکل a) است . . .
  - در این حالت، واحد (نورون) یک پرسپترون (Perceptron) نام دارد.
- . . . و یا دارای آستانه نرم است (تابع Logistic یا Sigmoid، شکل b)
  - پرسپتورن سیگموئیدی



$$\frac{1}{1+e^{-z}}$$

- دومی، مشتق پذیر است
- در هر صورت، کل شبکه، بیانگر تابعی غیرخطی است.

اتصالات در شبکه Feed-Forward

- شبکه Feed-Forward
  - اتصالات در یک مسیر (از ورودی به خروجی)
  - شبکه = گراف فاقد حلقه جهت دار
  - هر واحد، از واحدهای قبلی ورودی می‌گیرد و به واحدهای بعدی ورودی می‌دهد.
    - بدون حلقه
  - چنین شبکه‌ای، تابعی بر حسب ورودی فعلی را تشکیل می‌دهد.
  - شبکه دارای وضعیت درونی، غیر از وزن‌های یالها، نیست.

اتصالات در شبکه Recurrent

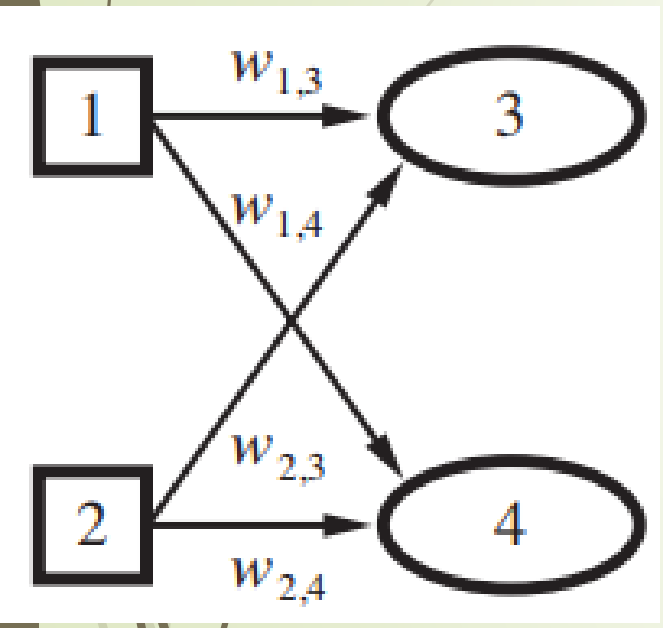
- شبکه Recurrent
  - خروجیش را مجدد به عنوان ورودی تزریق می‌کند به شبکه
  - خروجی سیستم، ممکن است
    - همگرا به نقطه‌ای ثابت شود
    - نوسانی باشد
    - دارای نظم خاصی نباشد
  - پاسخ شبکه به یک ورودی خاص، ممکن است به ورودی قبلی وابسته باشد.
    - شبکه دارای وضعیت درونی یا حافظه کوتاه مدت است.
    - جذاب تر، و پیچیده تر
    - کاربرد در داده‌هایی با ماهیت سری زمانی (مانند پردازش زبان طبیعی)
      - خروجی بعدی، وابستگی به خروجی‌های قبلی دارد. (در یک جمله، کلمه بعدی، مستقل از کلمه قبلی نیست)
  - در اینجا (کجا؟!) فقط شبکه Feed-Forward بررسی می‌شود.

ساختار شبکه Feed-Forward

- شبکه Feed-Forward دارای ساختار لایه‌ای است
- نورون در هر لایه صرفا از لایه قبل ورودی می‌گیرد.
- شبکه ممکن است تک لایه یا چند لایه باشد
- شبکه چند لایه دارای لایه یا لایه‌های نورون پنهان است
- خروجی شبکه (به عنوان یک ابزار یادگیری) می‌تواند چندگانه باشد
  - مثلا هر خروجی مشخص کننده یک دسته خاص، در کاربرد دسته‌بندی
  - یا ...

شبکه پرسپترون

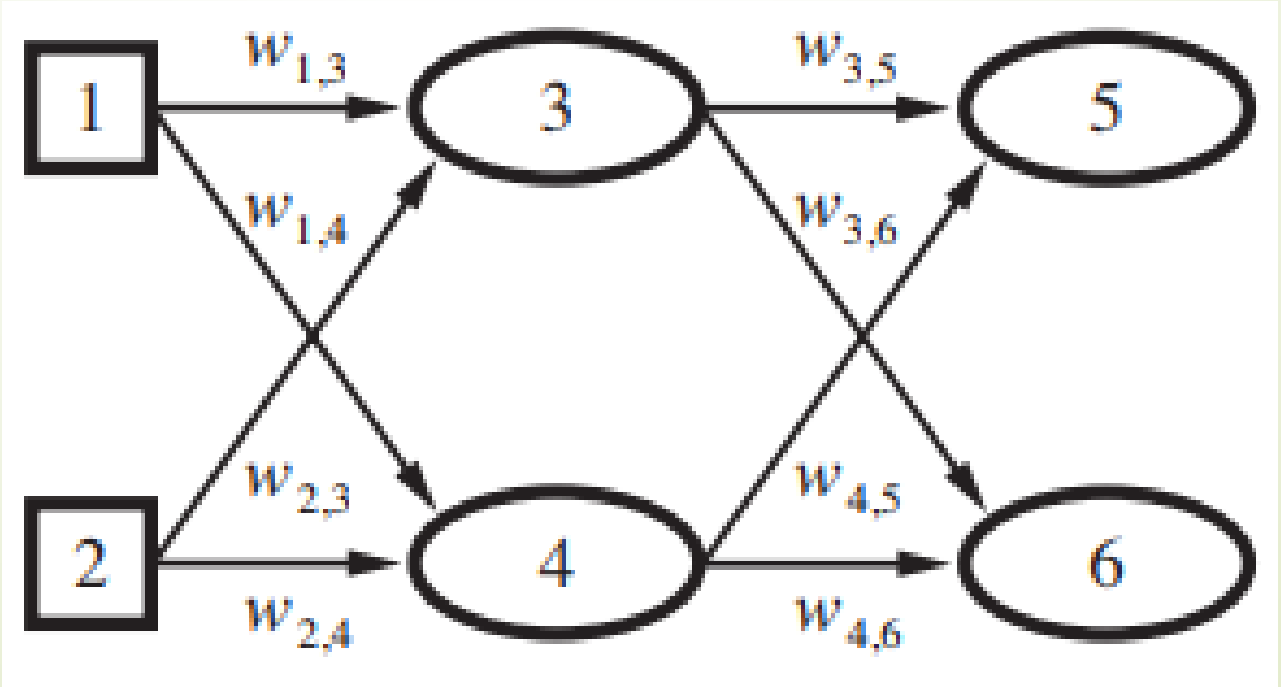
- شبکه پرسپترون، شبکه عصبی‌ای تک لایه و از نوع Feed-Forward است.
  - تک لایه: تمام ورودی‌هایش مستقیما به خروجی‌ها وصل هستند.
- مثال: شبکه زیر:
  - مثلا برای تابع جمع‌کننده دو بیتی
- داده‌های آموزشی مورد نیاز:



x_1	x_2	y_3 (carry)	y_4 (sum)
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

- هر خروجی شبکه پرسپترون (تک لایه) به طور مستقل از سایر خروجی ها آموزش می‌بیند و وزن یالهای ورودیش تعیین می‌شود.

- در شبکه چندلایه، یادگیری وزن‌ها برای نورون‌های لایه خروجی از هم مستقل نیستند.
- هرچند، محاسبات مربوطه، مستقل انجام خواهد شد.
  - در بخش‌های آینده خواهیم دید.



تعیین وزن یالها–آموزش شبکه

- در شبکه پرسپترون، برای به‌روزرسانی وزن یال‌های متصل به هر خروجی، می‌توان از دو روش استفاده کرد:
  - قاعده یادگیری پرسپترون – برای تابع فعال‌ساز با آستانه سخت
  - رگرسیون Logistic – برای تابع فعال‌ساز با آستانه نرم (سیگموئید)
- وزن‌ها باید به گونه‌ای تعیین شوند (آموزش داده شوند) که دقت خروجی شبکه بالاتر رود
- دقت؟
  - برای هر بردار ورودی X، یک بردار خروجی تولید می‌شود.
  - هر نمونه داده آموزشی، دارای بردار خروجی صحیح است.
  - خطا: اختلاف برداری خروجی صحیح با بردار خروجی تولید شده برای تمام داده‌های آموزشی

تعیین وزن یالها–آموزش شبکه

- اگر $h_w(x) = {w_1}^T x + w_0$ تابع فعالسازی باشد که در آن $w = [w_1, w_0]$ و نیز w_1 و x بردارهای هم‌طول هستند و ${w_1}^T x$ ضرب داخلی دو بردار را نشان می‌دهد

$${w_1}^T x = \sum_i w_{1,i} x_i$$

در این صورت، خطا برای خروجی شبکه پرسپترون برای N داده آموزشی خواهد بود:

$$Loss(h_{\mathbf{w}}) = \sum_{j=1}^{N} L_2(y_j, h_{\mathbf{w}}(x_j)) = \sum_{j=1}^{N} (y_j - h_{\mathbf{w}}(x_j))^2 = \sum_{j=1}^{N} (y_j - (w_1 x_j + w_0))^2$$

- تابع فوق برحسب متغیرهای وزن، تابعی **محدب** است و جواب معادله با صفر کردن مشتق رابطه فوق به دست می‌آید.
- اما اگر تابع فعال‌سازی، خطی نباشد، پیدا کردن وزن‌هایی که تابع خطا را کمینه کنند، دارای پاسخ محاسباتی به شکل فوق نخواهد بود.
  - تابع فعال‌ساز شبکه، نوعا خطی نیست.
- در این صورت، با یک مساله بهینه سازی (جستجو) در فضای پیوسته مواجه هستیم.

نزول در راستای گرادیان

- می‌توان از جستجوی تپه نوردی بر اساس **گرادیان** تابع هدف استفاده کرد.
  - تابع هدف: **خطای** خروجی
  - **نزول** در جهت گرادیان، هدف خواهد بود: Gradient Descent
- یک نقطه دلخواه برای وزن‌های W انتخاب می‌کنیم.
- در راستای کاهش گرادیان، به یک نقطه همسایه می‌رویم.
- تا وقتی که به یک نقطه کمینه همگرا بشویم، ادامه می‌دهیم.

$\mathbf{w} \leftarrow$ any point in the parameter space

loop until convergence **do**

  for each w_i **in** w **do**

$$w_i \leftarrow w_i - \alpha \frac{\partial}{\partial w_i} Loss(\mathbf{w})$$

- α پارامتر طول گام یا نرخ یادگیری نام دارد و می‌تواند ثابت باشد یا به مرور تغییر کند (چگونه؟)

یادگیری با رگرسیون Logistic

برای نمونه داده j در داده‌های آموزشی، تابع فعالسازی (سیگموئید) و با در نظر گرفتن $x_{j,0} = 1$:

$$h_w(x_j) = g\left(\sum_{i=0}^{n} w_i x_{j,i}\right) = g(w^T.x_j) = \frac{1}{1+e^{-w^T x_j}}$$

$$\frac{\partial}{\partial w_i} Loss(w) = \frac{\partial}{\partial w_i} \sum_j^N (y_j - h_w(x_j))^2 = 2\sum_j^N \left(y_j - h_w(x_j)\right).\frac{\partial}{\partial w_i}\left(y_j - h_w(x_j)\right)$$

$$= 2\sum_j^N \left(y_j - g(w^T.x_j)\right).\frac{\partial}{\partial w_i}\left(y_j - g(w^T.x_j)\right) = -2\sum_j^N \left(y_j - g(w^T.x_j)\right).g'(w^T.x_j)\frac{\partial}{\partial w_i}(w^T.x_j)$$

$$= -2\sum_j^N \left(y_j - g(w^T.x_j)\right).\left(g(w^T.x_j)\left(1 - g(w^T.x_j)\right)\right).x_{j,i}$$

برای تابع سیگموئید داریم: $g'(z)=g(z)(1-g(z))$

$$w_i \leftarrow w_i + \alpha \sum_j^N \left(y_j - g(w^T.x_j)\right).\left(g(w^T.x_j)\left(1 - g(w^T.x_j)\right)\right).x_{j,i}$$

قاعده یادگیری پرسپترون

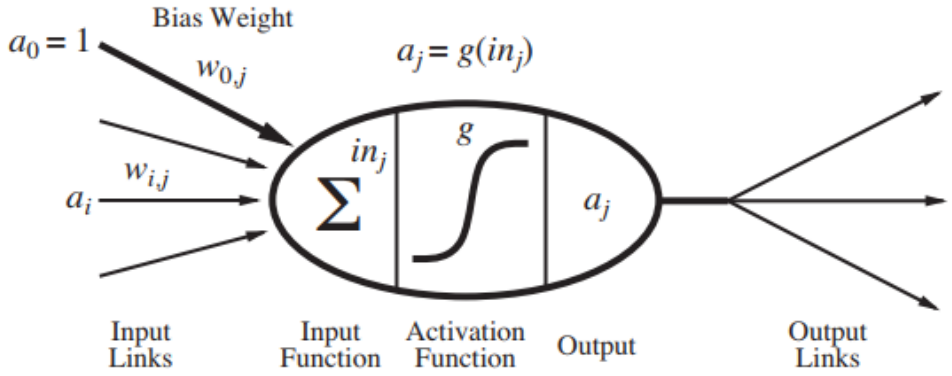
- نحوه به‌روزرسانی وزن‌های شبکه با تابع فعال‌سازی نرم طبق اسلاید قبل، رگرسیون Logistic نام دارد.
- اگر تابع فعال‌سازی با آستانه سخت به کار برده شود (مشتق ناپذیر) از قاعده یادگیری پرسپترون استفاده می‌شود:

$$w_i \leftarrow w_i + \alpha \sum_j^N \left(y_j - h_w(x_j)\right).x_{j,i}$$

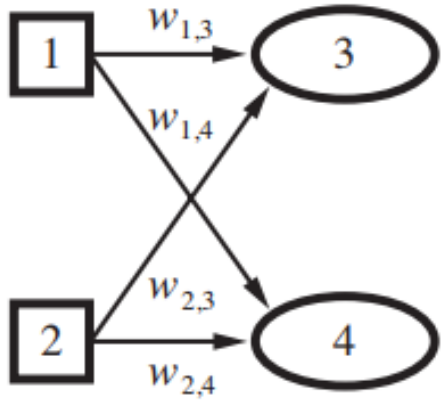
- اگر فقط یک نمونه داده آموزشی را در نظر بگیریم:
  - اگر خروجی شبکه برای نمونه درست باشد: $y_j = h_w(x_j)$، وزن‌ها تغییر نمی‌کنند
  - اگر y=1 و خروجی شبکه 0 باشد، وزن مربوطه بزرگتر می‌شود اگر ورودی متناظر با وزن $(x_{j,i})$ مثبت باشد (و برعکس)
    - احتمال آنکه خروجی برابر 1 شود بیشتر می‌شود
    - و برعکس!
    - احتمال آنکه خروجی برابر 0 شود بیشتر می‌شود

توان یادگیری شبکه پرسپترون

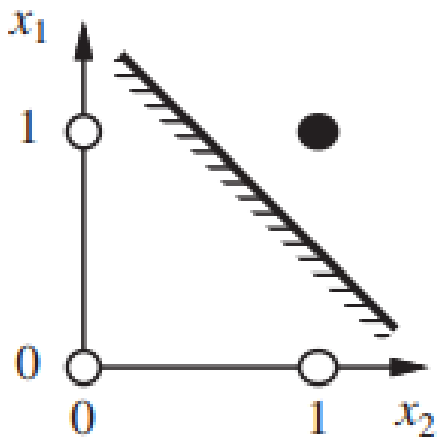
- شبکه پرسپترون، برای یادگیری جمع کننده باینری
  - نورون ۳، بخش carry را به درستی یادخواهد گرفت
  - اما نورون ۴، بخش sum را نمی‌تواند یادبگیرد. چرا؟
- نورون، یک جداکننده خطی است، (نه یک تابع خطی).
  - یک مرزجداکننده خطی در فضای ورودی‌هایش را بیان می‌کند.
  - خروجی غیرخطی را نمی‌تواند یادبگیرد.
  - تابع g عملا یک if روی خروجی تعریف می‌کند، نه چیز بیشتر.



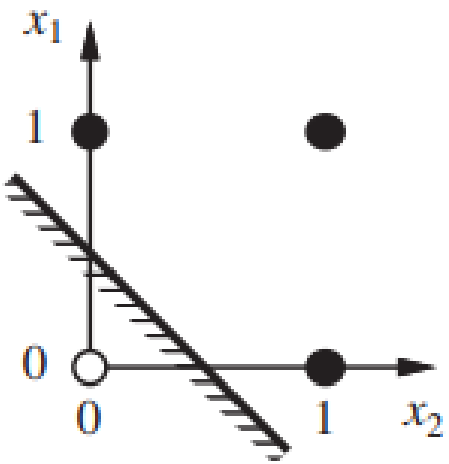
$$a_j = g\left(\sum_{i=0}^{n} w_{i,j} a_i\right)$$



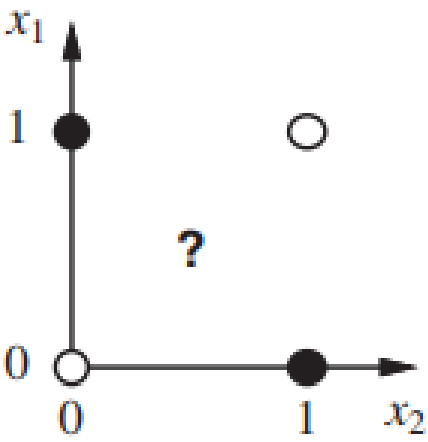
x_1	x_2	y_3 (carry)	y_4 (sum)
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



(a) x_1 **and** x_2



(b) x_1 **or** x_2

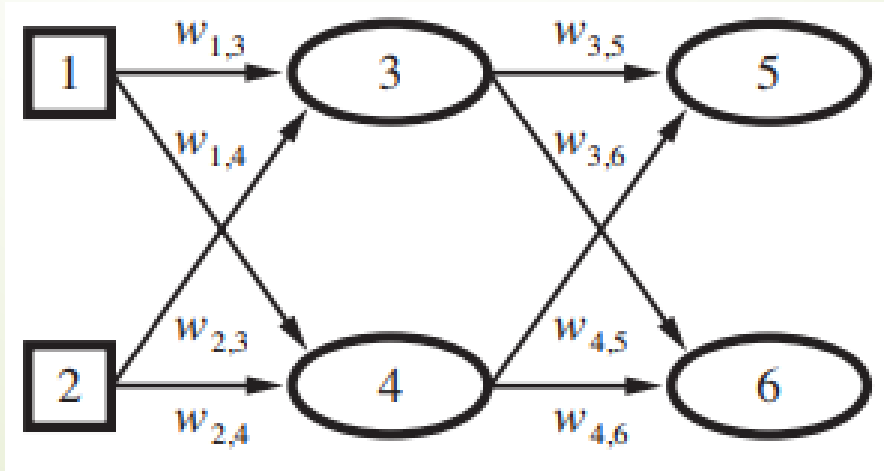


(c) x_1 **xor** x_2

شبکه عصبی چند لایه

- قدرت جداکنندگی و کارایی یک نورون به تنهایی، پایین است.
- اما شبکه ای از نورون‌ها با عمق لازم، می‌توانند هر کارایی دلخواهی را فراهم کنند!
- شبکه عصبی چندلایه، تابعی بر حسب ورودی‌ها و با پارامترهای وزن می‌باشد.
- مثلا در شبکه زیر، برای ورودی دو بعدی $\boldsymbol{x} = (x_1, x_2)$ خروجی نورون 5 خواهد بود:

$$
\begin{aligned}
a_5 &= g(w_{0,5,+} w_{3,5}\, a_3 + w_{4,5}\, a_4) \\
&= g(w_{0,5,+} w_{3,5}\, g(w_{0,3} + w_{1,3}\, a_1 + w_{2,3}\, a_2) + w_{4,5}\, g(w_0 4 + w_{1,4}\, a_1 + w_{2,4}\, a_2)) \\
&= g(w_{0,5,+} w_{3,5}\, g(w_{0,3} + w_{1,3}\, x_1 + w_{2,3}\, x_2) + w_{4,5}\, g(w_0 4 + w_{1,4}\, x_1 + w_{2,4}\, x_2))
\end{aligned}
$$

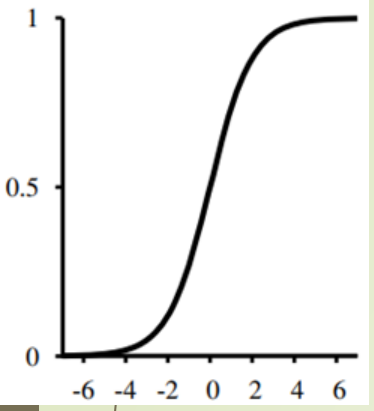


شبکه عصبی چند لایه

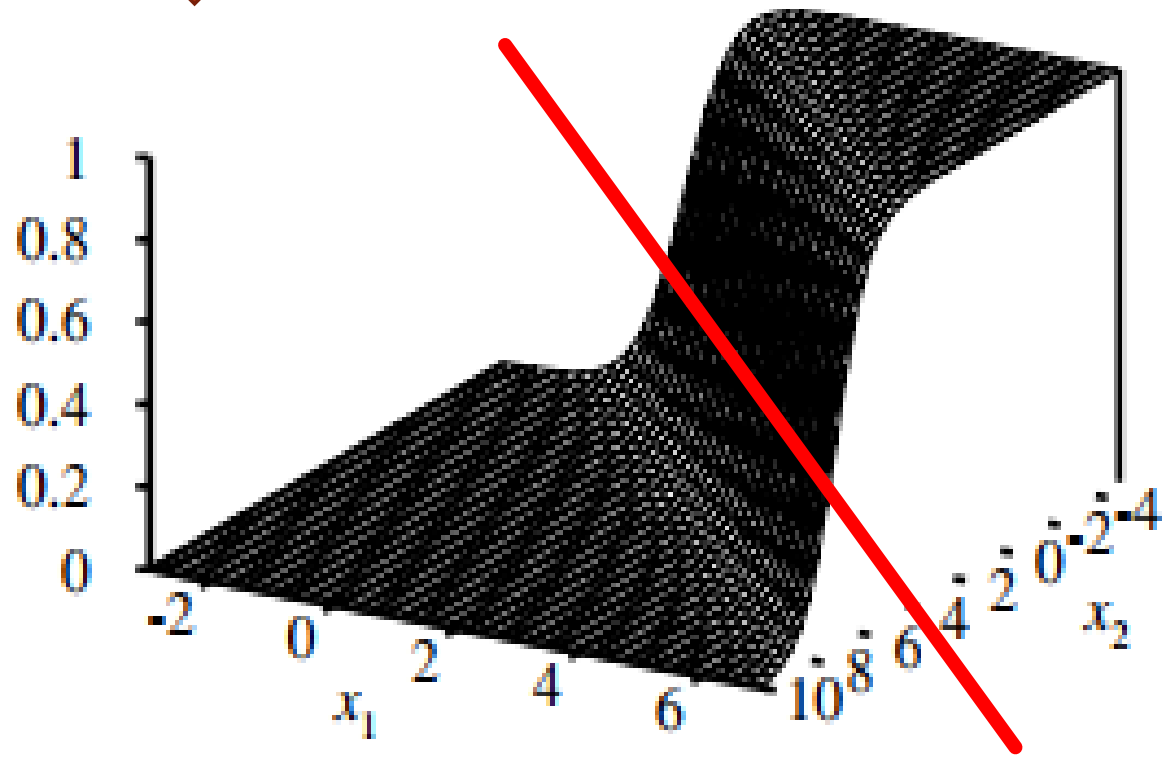
- خروجی در مثال قبل، برحسب وزن‌ها قابل مشتق‌گیری است.
- روش مبتنی بر نزول گرادیان برای بهینه سازی (یادگیری-آموزش) وزن‌های شبکه همچنان قابل استفاده است.
- در شبکه چندلایه، به دلیل ترکیب توابع آستانه‌گذاری متعدد (سیگموئید) در نورون‌های با عمق (شماره لایه) مختلف، تابعی که شبکه آن را بازنمایی می‌کند می‌تواند شدیدا غیرخطی باشد.
- شبکه چندلایه ابزاری منعطف برای تقریب توابع غیرخطی به شمار می‌آید.
  - آچار فرانسه!
- یک شبکه چندلایه چگونه قادر است توابع غیرخطی را بازنمایی کند؟

بازنمایی غیر خطی با شبکه عصبی چندلایه

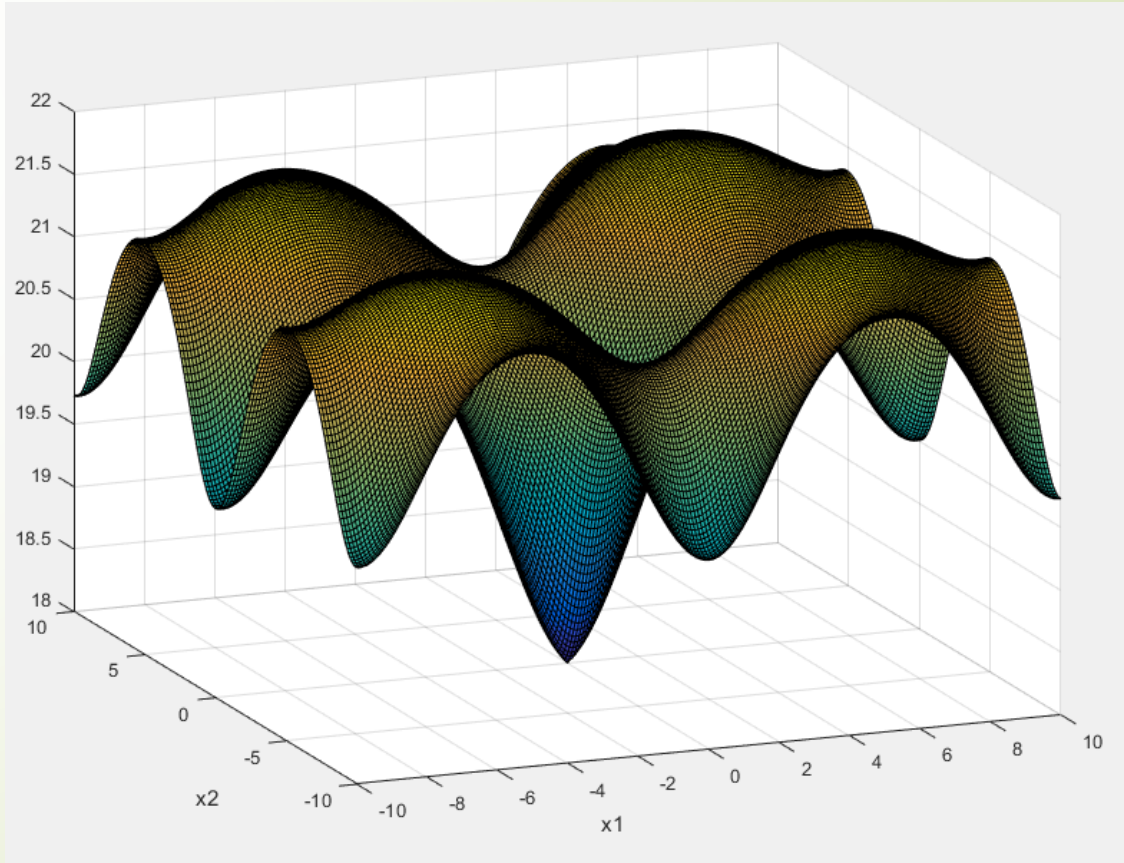
- فرض: ورودی دو بعدی (برای قابلیت نمایش)
- خروجی می‌تواند هر منحنی بسیار پیچیده شدیدا غیرخطی درب و داغانی(!) باشد
  - داغان با «غ» است؟ از داغ می‌آید؟
- انواع کاربردها



یک سیگموئید با ورودی دو بعدی

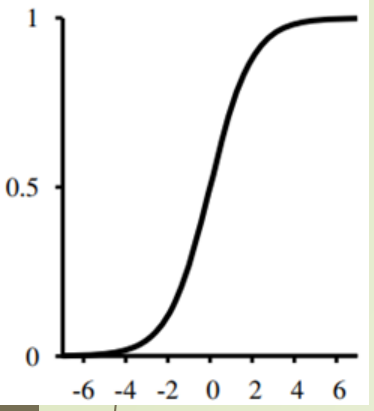


?

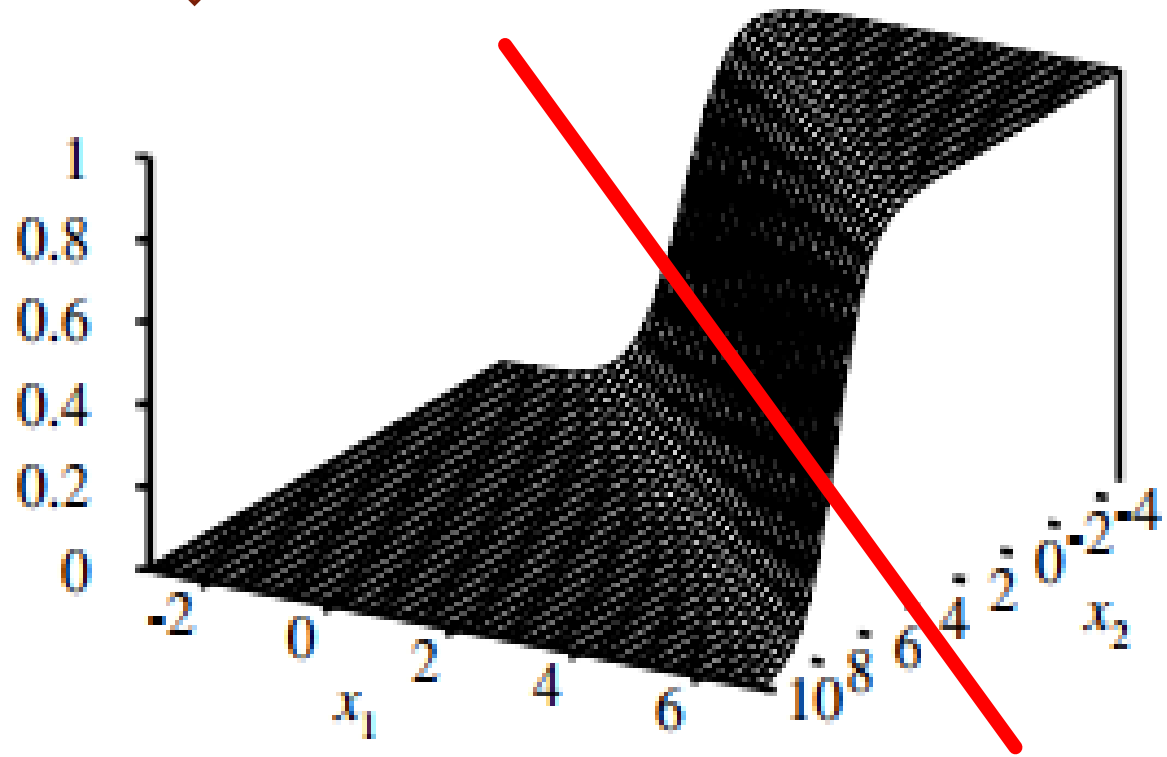


بازنمایی غیر خطی با شبکه عصبی چندلایه

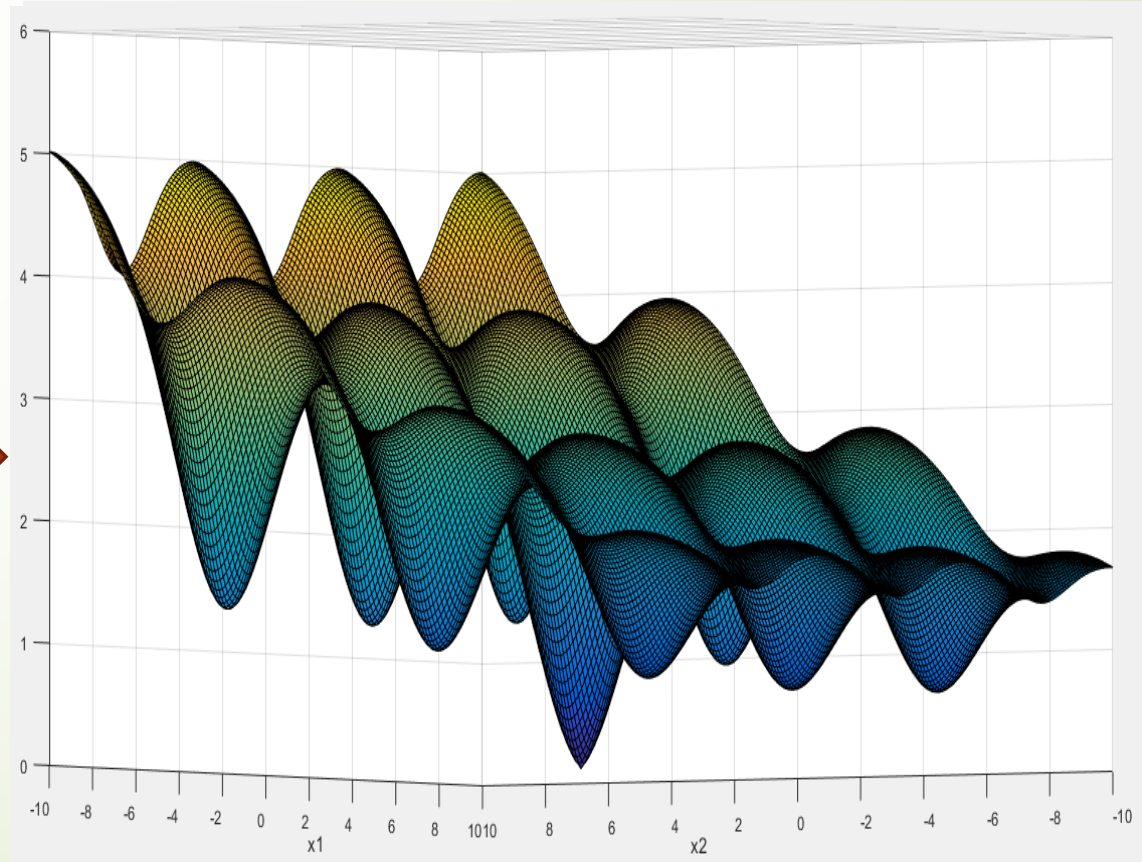
- فرض: ورودی دو بعدی (برای قابلیت نمایش)
- خروجی می‌تواند هر منحنی بسیار پیچیده شدیدا غیرخطی درب و داغانی(!) باشد
  - داغان با «غ» است؟ از داغ می‌آید؟
- انواع کاربردها



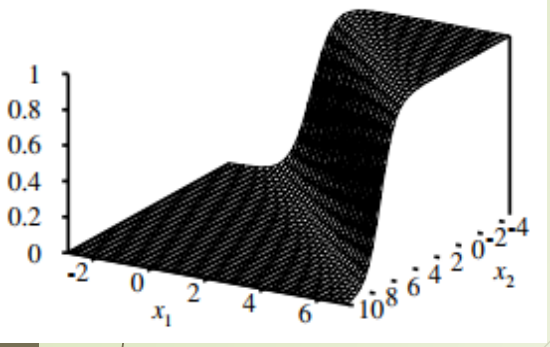
یک سیگموئید با ورودی دو بعدی



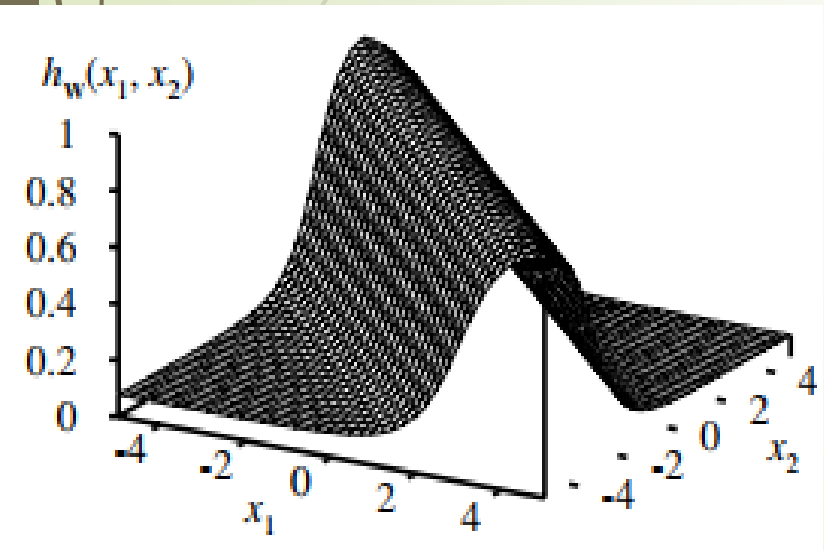
?



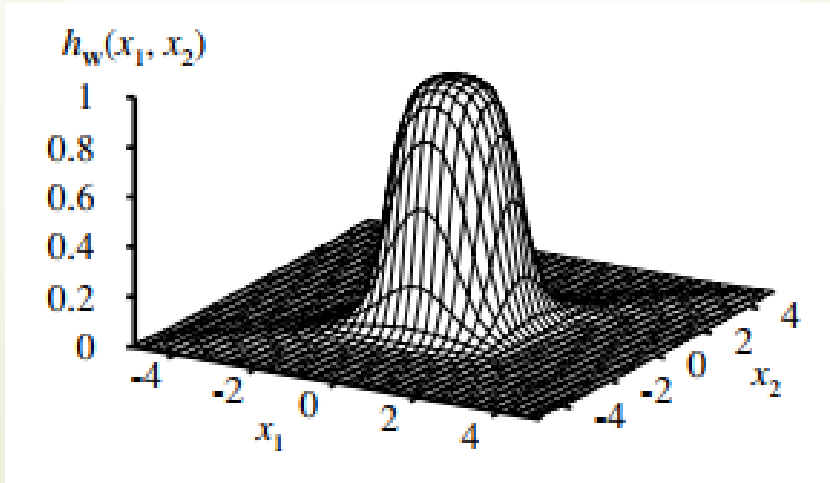
بازنمایی غیر خطی با شبکه عصبی چندلایه



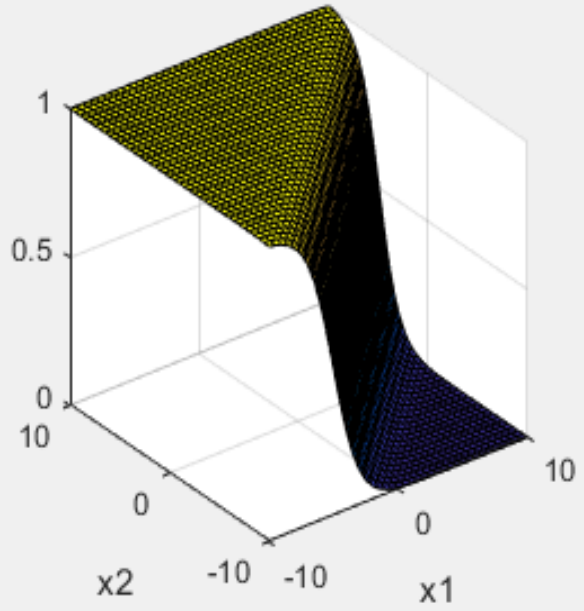
- یک نورون تنها: یک سیگموئید
- اگر یک لایه نورون پنهان داشته باشیم، هر نورون در لایه خروجی، یک آستانه‌گذاری نرم (سیگموئیدی) روی ترکیب خطی تعدادی سیگموئید خواهد بود.
- مثلا با ترکیب دو سیگموئید در خلاف جهت هم، یک منحنی به شکل روبرو در لایه دوم (خروجی) می‌تواند پدید آید.
  - آیا؟
  - با ترکیب ۳ سیگموئید یا بیشتر؟

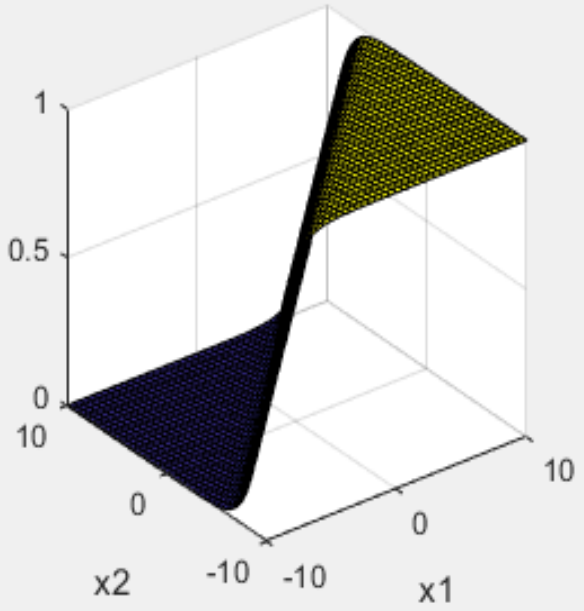


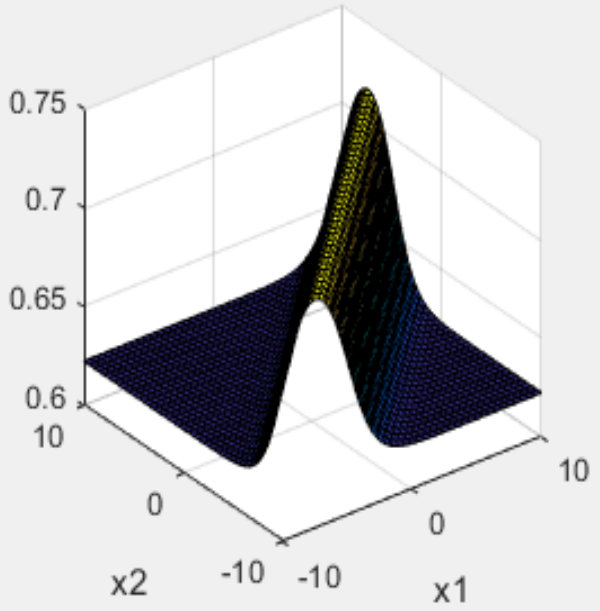
- اگر یک لایه پنهان دیگر نیز اضافه شود، ترکیب دو منحنی از لایه دوم (شکل روبرو) شکل زیر را در لایه سوم (خروجی) می‌تواند پدید آورد
  - الله اکبر! هزار ماشاءالله!

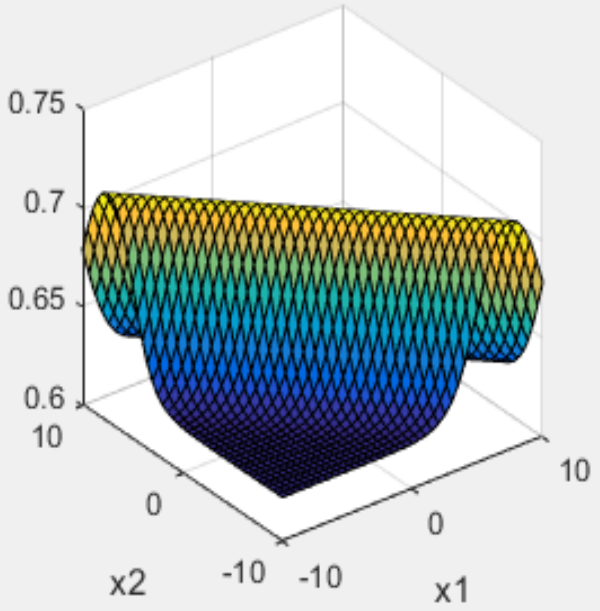


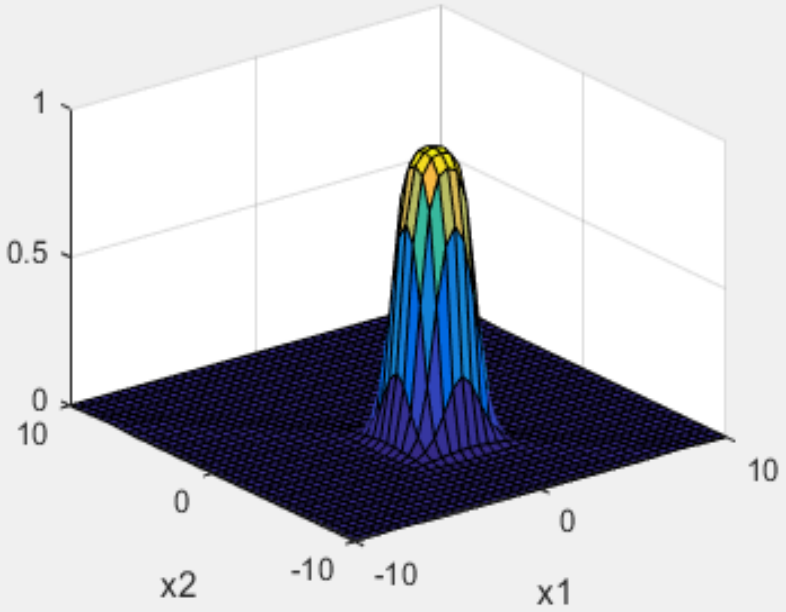
- با همان یک لایه پنهان، و با ترکیب۴ نورون پنهان هم می‌توان شکل روبرو را ایجاد کرد.
  - ترکیب دو جفت سیگموئید در خلاف جهت هم

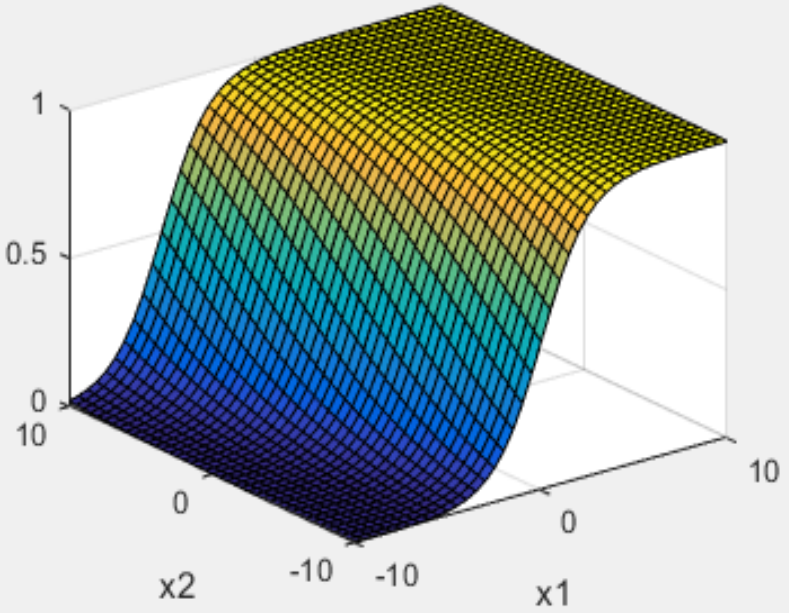

1
0.5
0
10
0
-10
x2
-10
0
10
x1

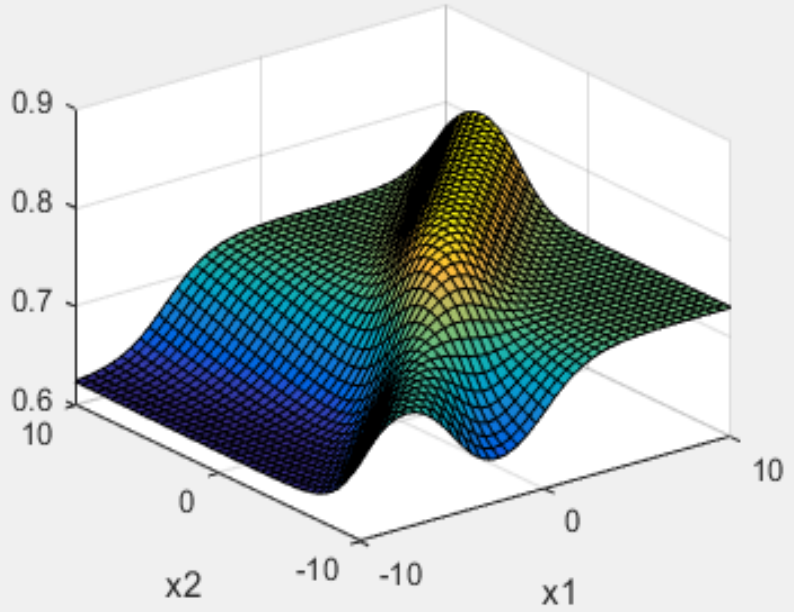

1
0.5
0
10
0
-10
x2
-10
0
10
x1

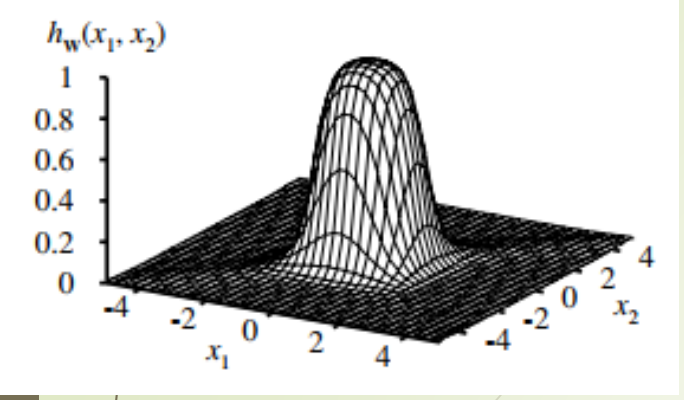

0.75
0.7
0.65
0.6
10
0
-10
x2
-10
0
10
x1


0.75
0.7
0.65
0.6
10
0
-10
x2
-10
0
10
x1


1
0.5
0
10
0
-10
x2
-10
0
10
x1

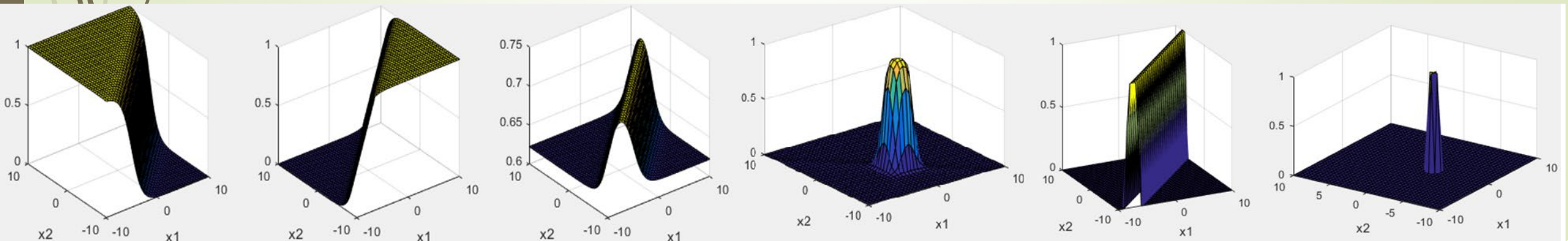

1
0.5
0
10
0
-10
x2
-10
0
10
x1


0.9
0.8
0.7
0.6
10
0
-10
x2
-10
0
10
x1



در پیشگاه شبکه عصبی مصنوعی

- با ترکیب نتایج نورون‌های پنهان بیشتر، می‌توان قله‌های بیشتری در نواحی مختلف ایجاد کرد. (در مسیر بازنمایی تابع دلخواه)
- با **یک لایه پنهان** به اندازه کافی بزرگ، می‌توان **هر** تابع پیوسته‌ای را با دقت مورد نیاز، بازنمایی کرد.
  - آیا ایمان نمی‌آورید؟
  - با بزرگ گرفتن وزن متناظر با سیگموئیدها (و احتمالا در نظر گرفتن عدد منفی بزرگ برای W_0) منحنی‌ها و قله‌های تیزتر (با ارتفاع دلخواه) قابل ایجاد است.
  - در بدترین حالت، برای هر نقطه از دامنه ورودی، یک قله خیلی تیز با ارتفاع برابر با مقدار صحیح تابع هدف ایجاد می‌شود و کل تابع هدف با مجموعه تعداد زیادی از این قله‌ها تقریب زده می‌شود.
    - صرفا برای سخت ایمان آورندگان! در عمل، مساله بسیار ساده‌تر حل می‌شود.

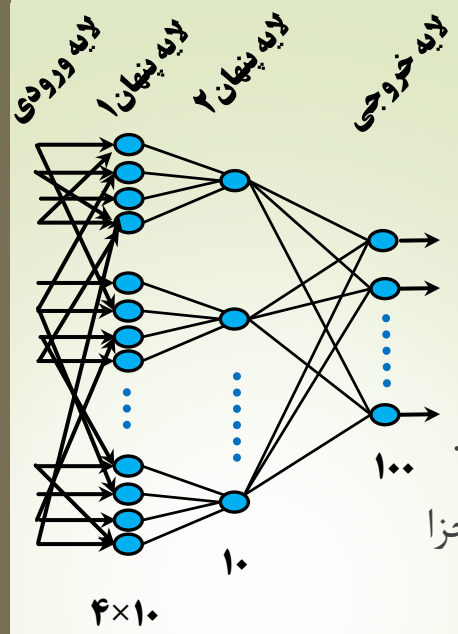


شبکه عمیق‌تر

- اگر با یک لایه پنهان (=شبکه دولایه) تقریبا می‌توان هر تابعی را بازسازی کرد، چرا و چه موقع و آیا(!) لایه‌های بیشتر نیاز می‌شود؟
- با بیش از یک لایه پنهان، ساختار شبکه برای رسیدن به مقاصد مورد نظر، طراحی می‌شود.
  - شبکه‌های عمیق
  - AutoEncoders
- مثلا در شبکه سه‌لایه:
  - در لایه اول، تعدادی سیگموئید یادگرفته و ذخیره می‌شود.
  - در لایه دوم، تعدادی ساختار دارای قابلیت استفاده مجدد در مسائل مختلف یادگرفته و ذخیره می‌شود.
    - مثلا تعدادی تابع که بیانگر نوعی خوشه‌بندی فضای ورودی هستند. (در حالت ساده، هر کدام، یک قله در بخشی از فضای ورودی)
  - حالا برای هر مساله جدید، در لایه سوم، وزن‌های مربوطه بر اساس ساختارهای موجود در لایه دوم، هر بار جداگانه یادگرفته می‌شود
    - مساله جدید؟ تقریب یک تابع جدید. یک دسته‌بندی جدید.
- در شبکه‌های عمیق، مثلا در پردازش تصویر، ساختارهای دارای پیچیدگی مختلف در لایه‌های متعدد شبکه یادگرفته و ذخیره می‌شوند، و برای هر دسته بندی خاص، صرفا وزن‌های لایه آخر (لایه دسته بندی) یادگرفته می‌شوند.

شبکه عمیق‌تر

- کاهش هزینه یادگیری با ایجاد ساختارهای دارای قابلیت استفاده مجدد در لایه دوم
  - هزینه اگر بالا باشد، ممکن است باعث عدم استفاده از شبکه **در عمل** شود (علی‌رغم قدرت بالای شبکه **در تئوری**)
- یک مثال ساده (ابعاد ورودی = n):
  - لایه اول دارای ۴۰=۱۰×۴ نورون سیگموئیدی (هر ۴ نورون ایجاد کننده یک قله در لایه بعدی)
  - لایه دوم دارای ۱۰ نورون ناحیه بندی (۱۰ قله/ناحیه در نقاط مختلف)
  - لایه سوم، فعلا یک نورون خروجی، که می‌تواند با کمی ساده سازی، برای ۱۰ ناحیه ورودی، 2^{10} تابع را بازنمایی کند.
    - نورون خروجی در کدام نواحی فعال باشد و در کدام ها غیر فعال = 2^{10} حالت مختلف، هر حالت، یک مساله (دسته‌بندی) مجزا
  - برای یادگیری هر خروجی(هر مساله)، ۱۰ وزن ورودی از لایه دوم به نورون خروجی لازم است یادگرفته شود.
    - وزن‌های لایه‌های قبل به تعداد «۴۰×۱۰ + ۴۰×n» قبلا یادگرفته و ذخیره شده‌اند.
  - اگر ساختار دو لایه بود (حذف لایه دوم)، برای هر مساله، ۴۰ وزن مجزا باید یادگرفته می‌شد (به جای ۱۰)
  - اگر لایه خروجی دارای ۱۰۰ نورون باشد (خروجی ۱۰۰ بعدی)، حتی بدون فرض ذخیره سازی وزن‌های لایه‌های قبل:
    - تعداد وزن‌های شبکه ۳ لایه = ۱۰×۱۰۰ + ۴۰×۱۰ + n×۴۰ = ۱۴۰۰ + n×۴۰
    - تعداد وزن‌های شبکه ۲ لایه = ۴۰×۱۰۰ + n×۴۰ = ۴۰۰۰ + n×۴۰

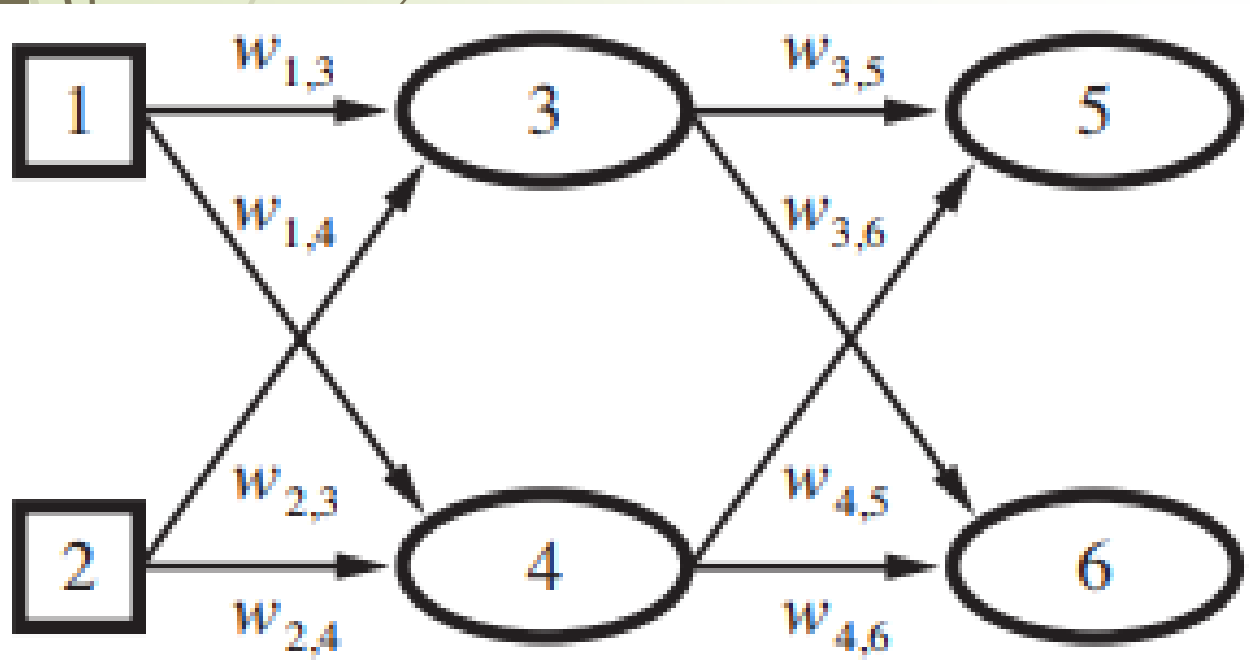


یادگیری وزن در شبکه چندلایه

- برای شبکه‌ای چندلایه با k خروجی در لایه آخر، (مانند شکل زیر)، هر وزن طبق روش نزول در راستای گرادیان به‌روز خواهد شد

$$\frac{\partial}{\partial w} Loss(\mathbf{w}) = \frac{\partial}{\partial w} |\mathbf{y} - \mathbf{h_w}(\mathbf{x})|^2 = \frac{\partial}{\partial w} \sum_k (y_k - a_k)^2 = \sum_k \frac{\partial}{\partial w} (y_k - a_k)^2$$

- می‌توان گرادیان هر وزن را با توجه به صرفا یکی از خروجی‌ها محاسبه کرد (مستقل از سایر خروجی‌ها)
- بدین ترتیب، یادگیری وزن‌های شبکه به k مساله یادگیری مستقل تقسیم می‌شود.
- نهایتا هنگام تغییر هر وزن، همه k گرادیان محاسبه شده با هم جمع می‌شوند.



محاسبه گرادیان برای یادگیری وزن ها

- گرادیانی که قبلا برای به روز کردن وزنها محاسبه کردیم، برای یک تک نورون بود.

$$w_i \leftarrow w_i + \alpha \sum_j \left(y_j - g(w^T.x_j)\right).\left(g(w^T.x_j)\left(1 - g(w^T.x_j)\right)\right).x_{j,i}$$

  - بر حسب وزنهای ورودی نورون (به عنوان متغیر)
  - و با فرض ثابت بودن خود ورودی ها (x_i ها).
- در شبکه چند لایه، ورودی نورون ها نیز ثابت نیستند و برحسب وزن های لایه های قبل تر متغیرند.
- بنابراین، یا باید محاسبات گرادیان برای شبکه چندلایه مجددا با فرض متغیر بودن ورودی نورونها تکرار شود
  - و برای هر شبکه با تعداد لایه و نورون مختلف محاسبات جدیدی انجام داد
- یا ...

یادگیری وزن‌ها با انتشار رو به عقب
Back-Propagation

- به‌روز رسانی وزن‌ها با کمک روش گرادیان، مبتنی بر گرادیان **خطا**
- در لایه‌های پنهان، خطا چیست؟
  - در لایه خروجی، خطا = اختلاف خروجی صحیح نمونه داده با خروجی نورون لایه آخر
- روش انتشار رو به عقب خطا، خطای لایه آخر را بین نورون‌های لایه‌های قبلی، به طور عادلانه تقسیم می‌کند!
- سهم نورون‌های لایه قبل از لایه آخر از خطا، متناسب با وزن متناظر با هر نورون است.
  - وزن بالاتر ← اثرگذاری بیشتر در نورون بعدی، سهم بیشتر از دقت/خطا

یادگیری وزن‌ها با انتشار رو به عقب

$$w_i \leftarrow w_i + \alpha \sum_j \big(y_j - g(w^T.x_j)\big).\Big(g(w^T.x_j)\big(1-g(w^T.x_j)\big)\Big).x_{j,i}$$

- یادگیری وزن برای یک نورون خروجی (ورودی‌ها به جای x_i، خروجی a_i از نورون‌های لایه قبل است):

$$w_i \leftarrow w_i + \alpha \underbrace{(y - h_w)}_{Err} \underbrace{\big(h_w(1-h_w)\big)}_{g'(in)}.a_i$$

- لایه خروجی با K نورون ← خطا = بردار K تایی ← مولفه k ام خطا = Err_k
- اگر داشته باشیم: $\Delta_k = Err_k.g'(in_k)$ (که در آن g تابع سیگموئید است)، به‌روزرسانی وزن‌های بین لایه آخر و لایه قبل آخر چنین خواهد بود:

$$w_{i,k} \leftarrow w_{i,k} + \alpha.a_i.\Delta_k$$

- برای وزن‌های لایه‌های قبل‌تر، Δ (خطا) باید بازتعریف شود (تا رابطه فوق قابل استفاده باشد)
- برای این کار از انتشار رو به عقب خطای Δ_k متناسب با وزن استفاده می‌شود:

$$\Delta_i = g'(in_i)\sum_k w_{i,k}\Delta_k$$

سهم خطای نورون i از خطای همه k نورون‌لایه بعدی متناسب با وزن فیمابین، محاسبه می‌شود.

function BACK-PROP-LEARNING(*examples*, *network*) **returns** a neural network
inputs: *examples*, a set of examples, each with input vector **x** and output vector **y**
network, a multilayer network with L layers, weights $w_{i,j}$, activation function g
local variables: Δ, a vector of errors, indexed by network node

repeat
for each weight $w_{i,j}$ in *network* **do**
$w_{i,j} \leftarrow$ a small random number
for each example (**x**, **y**) **in** *examples* **do**
/* *Propagate the inputs forward to compute the outputs* */
for each node i in the input layer **do**
$a_i \leftarrow x_i$
for $\ell = 2$ **to** L **do**
for each node j in layer ℓ **do**
$in_j \leftarrow \sum_i w_{i,j}\, a_i$
$a_j \leftarrow g(in_j)$
/* *Propagate deltas backward from output layer to input layer* */
for each node j in the output layer **do**
$\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$
for $\ell = L - 1$ **to** 1 **do**
for each node i in layer ℓ **do**
$\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j}\, \Delta[j]$
/* *Update every weight in network using deltas* */
for each weight $w_{i,j}$ in *network* **do**
$w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$
until some stopping criterion is satisfied
return *network*

الگوریتم انتشار رو به عقب

- انتساب مقدار تصادفی اولیه به وزن‌ها
- اعمال الگوریتم برای هر نمونه آموزشی به طور مجزا در یک حلقه تکرارشونده
  - محاسبه خروجی نورون‌ها رو به جلو
  - محاسبه خطای لایه آخر
  - انتشار رو به عقب خطا برای همه لایه‌ها
  - به‌روزرسانی همه وزن‌ها برحسب Δ ها
- تکرار چرخه تا رسیدن به شرط خاتمه
  - حداکثر تکرار مجاز
  - میزان تغییر اندک وزن‌ها

یادگیری ساختار شبکه

- شبکه خیلی بزرگ، می‌تواند تمام نمونه‌های آموزشی را حفظ کند
  - برای هر نمونه، تعدادی نورون پنهان ایجاد می‌شوند که باعث ایجاد خروجی صحیح برای آن نمونه می‌شوند.
  - چنین شبکه‌ای احتمالا دارای تعمیم (به نمونه‌های دیده نشده) نخواهد بود ← بیش برازش (Overfitting)
- در شبکه‌های fully connected تنها انتخاب برای ساختار، تعداد لایه‌ها و اندازه هر لایه است.
- راه حل: آزمودن چند حالت مختلف و انتخاب بهترین
- برای جلوگیری از بیش برازش و منطقی‌تر شدن روند آموزش، از تکنیک **Cross-Validation** استفاده می‌شود.
- برای شبکه‌های غیر fully connected با مساله جستجو (برای یافتن روابط سودمند) مواجه هستیم.
  - می‌توان ابتدا از شبکه fully connected شروع کرد و با ایده‌هایی روابط زائد را cut کرد.
- اندازه شبکه هم می‌تواند با ایده‌های خلاقانه تعیین شود (به جای صرفا آزمون و خطا)
  - مثلا شروع از یک شبکه کوچک و افزودن نورون برای ایجاد مقدار صحیح برای نمونه‌هایی که قبلا خروجی شبکه برایشان خطا داشته است.
  - یا...