شبکه های عصبی مصنوعی

جواد سلیمی

پاییز ۱۳۹۸

تاریخچه

- در ۱۹۴۰ به این نتیجه رسیدند که شبکه عصبی یک واحد پردازشگر است.
- در ۱۹۵۰ شبکه عصبی مصنوعی پرسپترون توسط رزنبلت ارائه شد.
- در دهه ۶۰ تا ۷۰ شبکه های عصبی ساده روی کار آمدند.
- در ۱۹۸۶ شبکه چند لایه با یادگیری پس انتشار خطا داده شد.

کاربردها

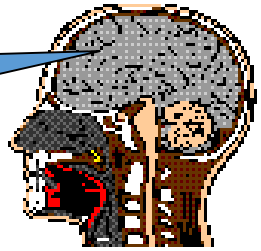
- بازشناسی دست نوشته ها
- بازشناسی لغات تلفظ شده
- بازشناسی چهره
- پیش بینی سریهای زمانی
- بازشناسی الگو

کاربرد

- امور مالی، بیمه ، بانکها
  - تخمین سهام، تجویز وام
- بازاریابی، تجارت
  - پیش بینی فروش، نمایش نمودار مشتری
- پزشکی
  - تشخیص و تجویز درمان
- صنعت
  - کنترل کیفیت، کنترل ماشین آلات
- بازشناسی الگو

یادگیری در مغز انسان

مغز انسان چگونه الگوها را یاد می گیرد



فرایند یادگیری مبتنی بر اصول زیر است

Massive parallelism

- مغز انسان از تعداد زیادی نرون تشکیل شده
- نرونها با لینکهای مستقیم به یکدیگر ارتباط دارند.

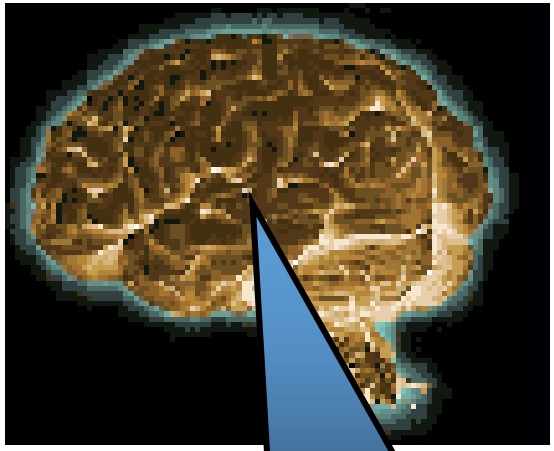
Connectionism

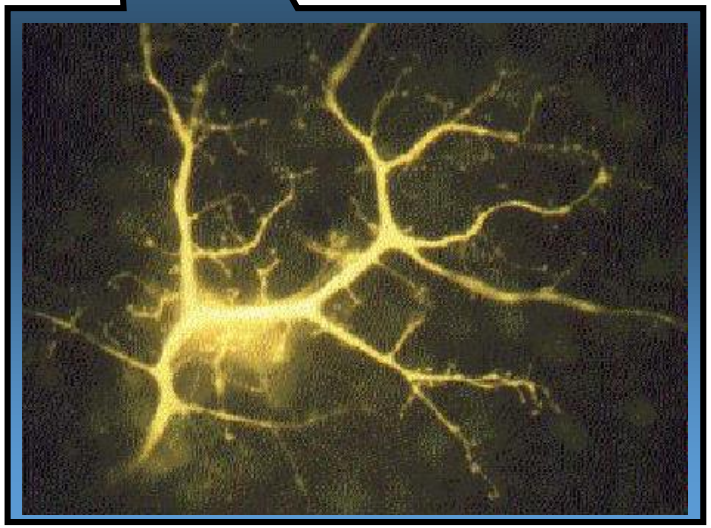
- نرونها با لینکهای زیادی به هم متصلند

Associative distributed memory

- اطلاعات در اتصالات بین نرونها و در سیناپسها ذخیره می شود
- الگوها در این اتصالات و قدرت (وزن) این اتصالات ذخیره می شود.

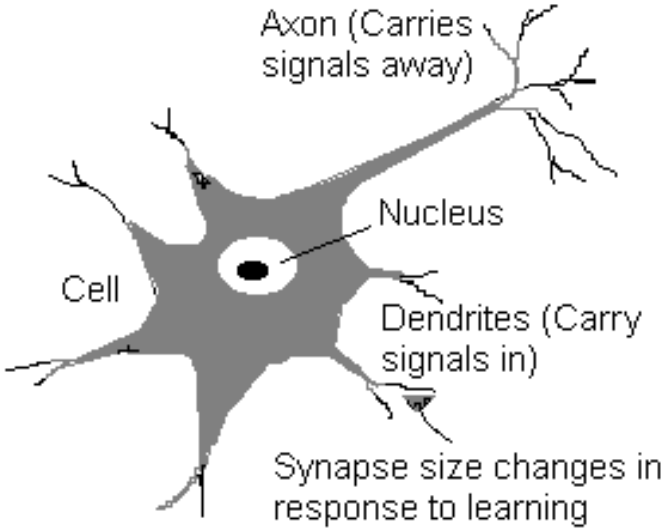
مغز

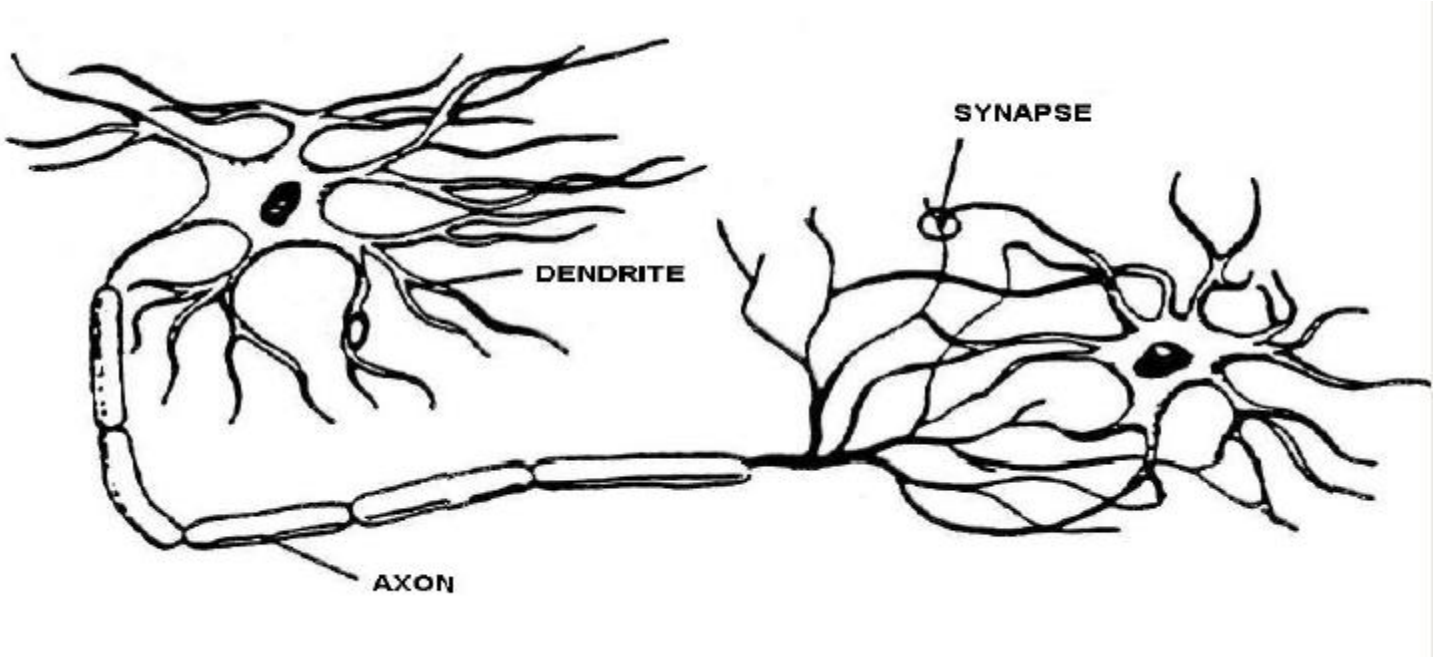




بیولوژیک نرونها

- **بدنه سلولی:** تحقق کارهای منطقی (پردازشگر)
- **دندریت:** شاخه های نرون که تحریک را از نرونهای دیگر دریافت می کنند. (ورودی)
- **آکسون:** یک شاخه عصبی که کانال خروجی نرون است.
- **سیناپس:** اتصال بین آکسون یک نرون و دندریت نرون دیگر. (وزن اتصالات)

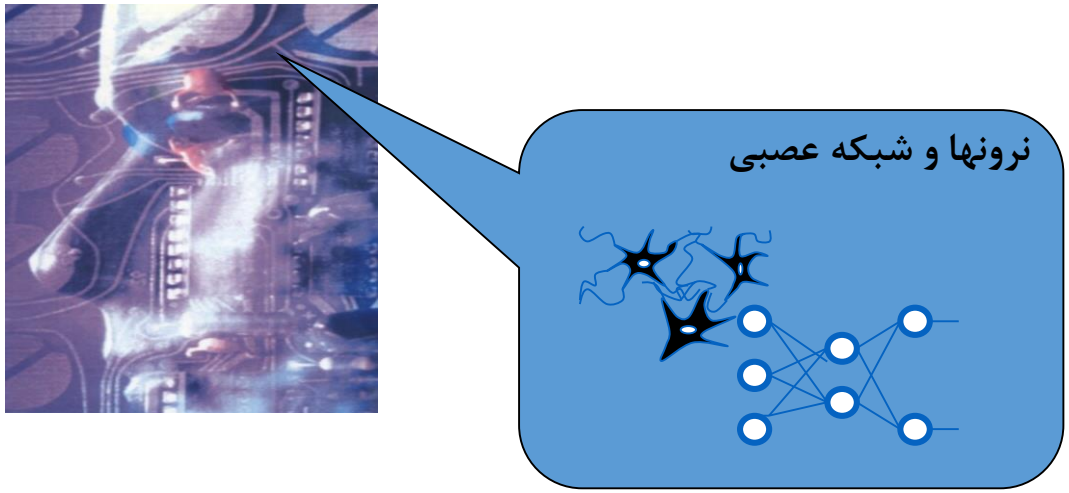

Axon (Carries signals away)
Nucleus
Cell
Dendrites (Carry signals in)
Synapse size changes in response to learning

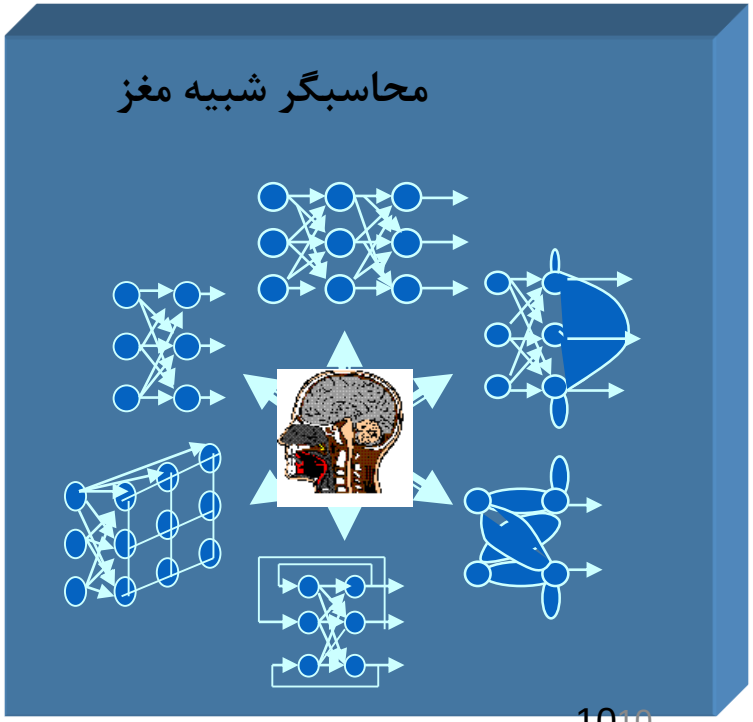

SYNAPSE
DENDRITE
AXON

شبکه های عصبی زیستی

- مجموعه ای بسیار عظیم از پردازشگرهایی موازی به نام نورون اند و توسط سیناپس ها اطلاعات را منتقل می کنند.
- این شبکه ها قادر به یادگیری اند. مثلا با اعمال سوزش به سلولهای عصبی لامسه، سلولها یاد می گیرند که به طرف جسم داغ نروند.
- یادگیری در این سیستم ها به صورت تطبیقی صورت می گیرد، یعنی با استفاده ازمشاهدات ها وزن سیناپس ها به گونه ای تغییر می کند که در صورت دادن ورودی های جدید سیستم پاسخ درستی تولید کند.

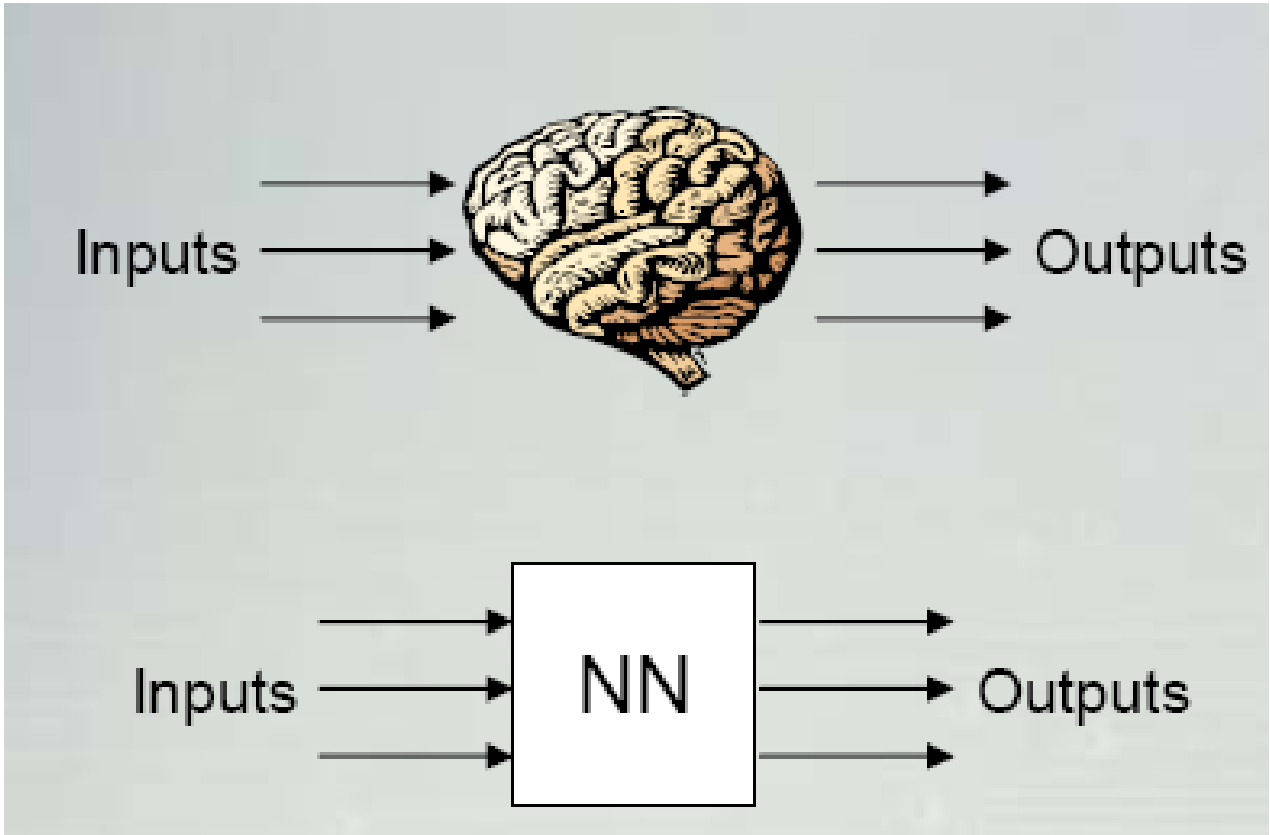
مغز مانند یک محاسبه گر



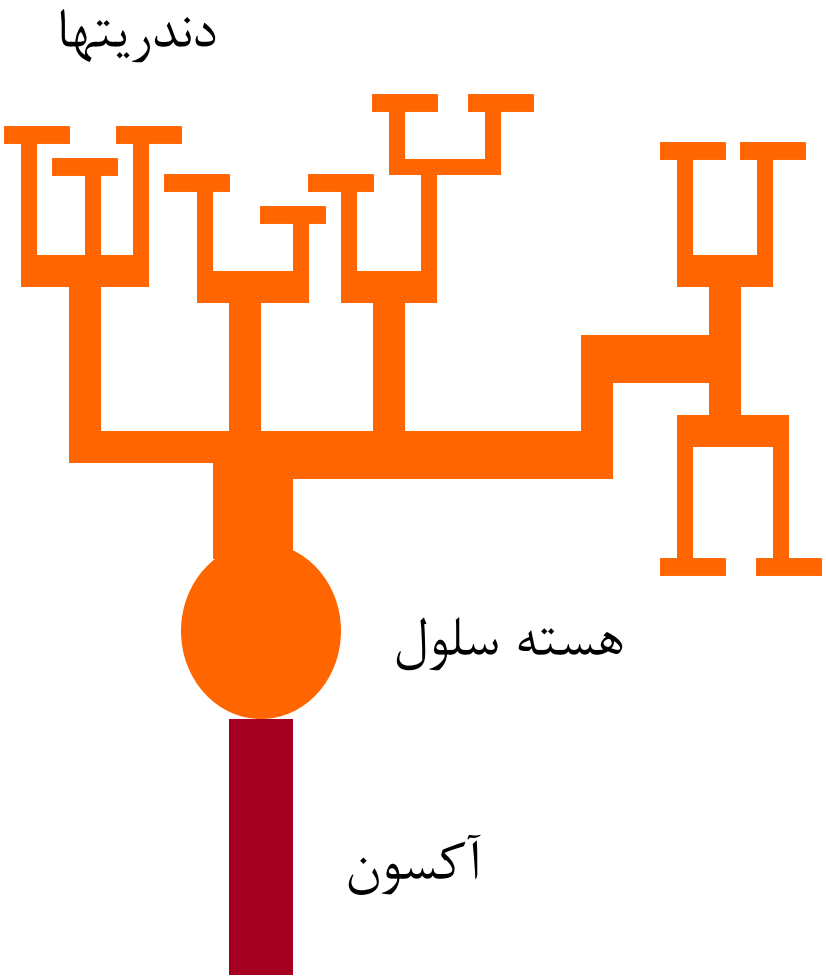


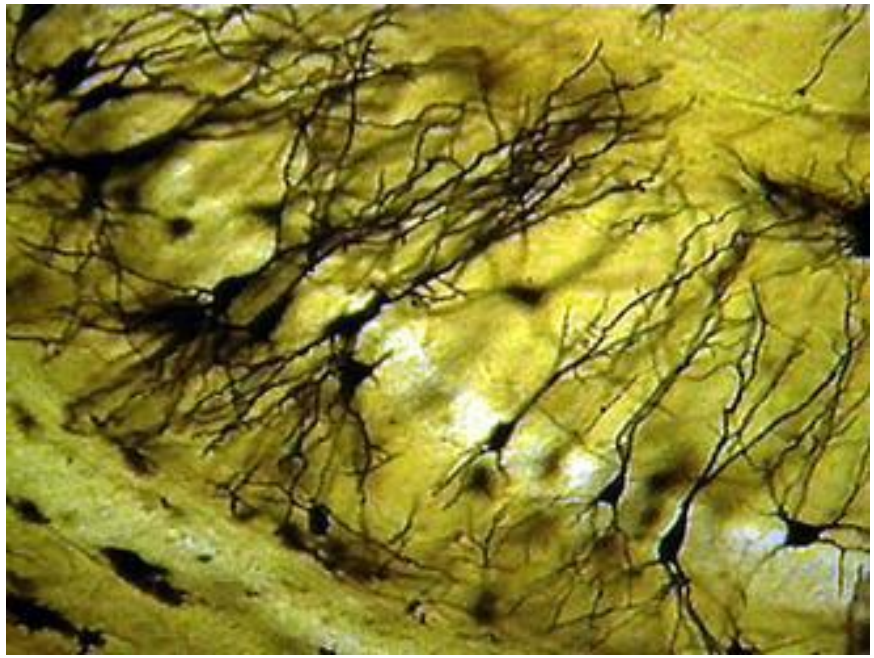
مدلسازی پردازش موازی در مغز

الهام از یادگیری در مغز

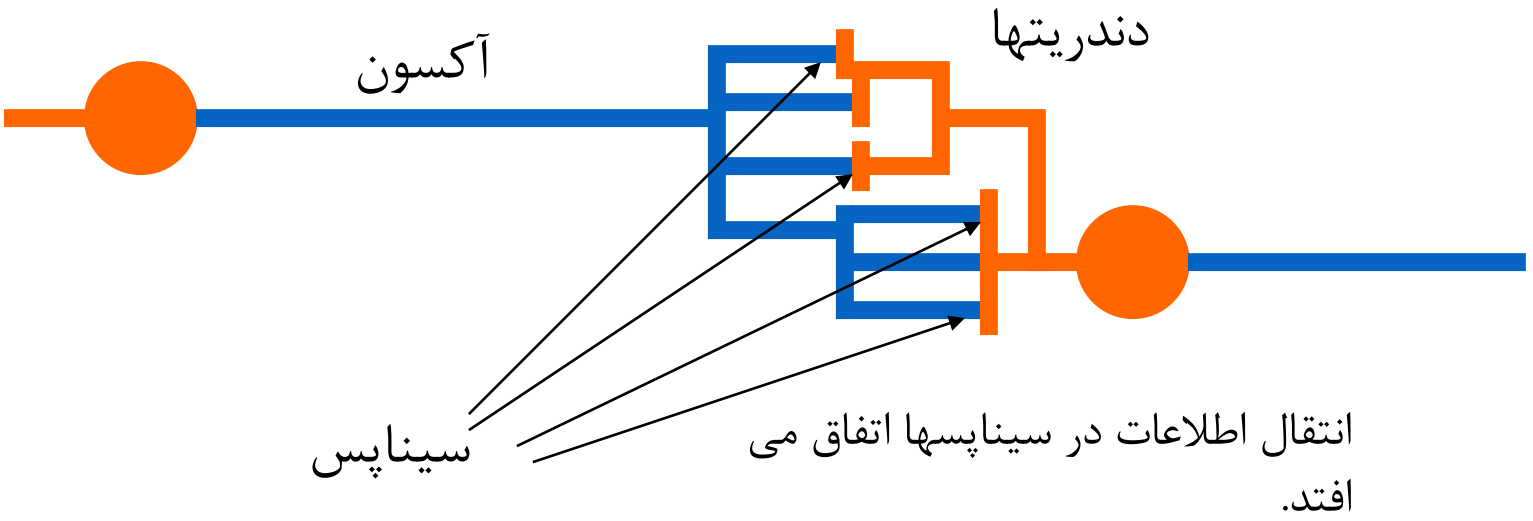


الهام از بیولوژیک





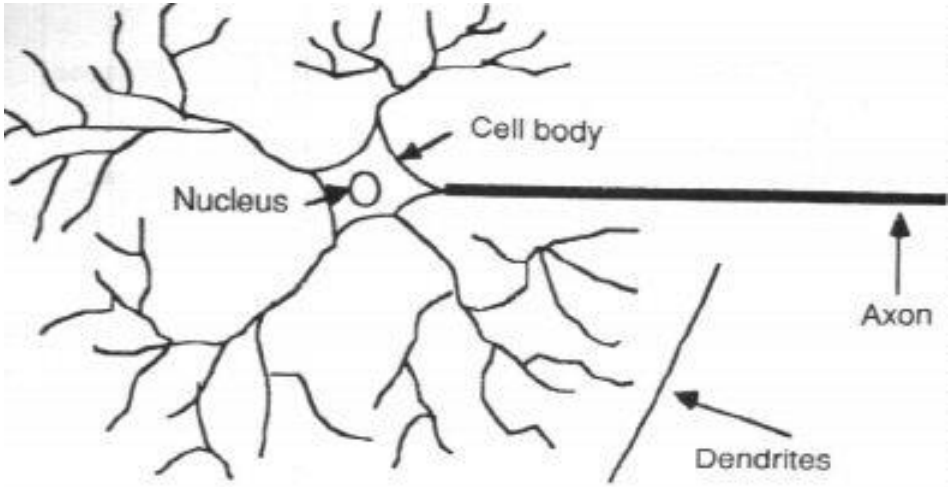
الهام از بیولوژیک

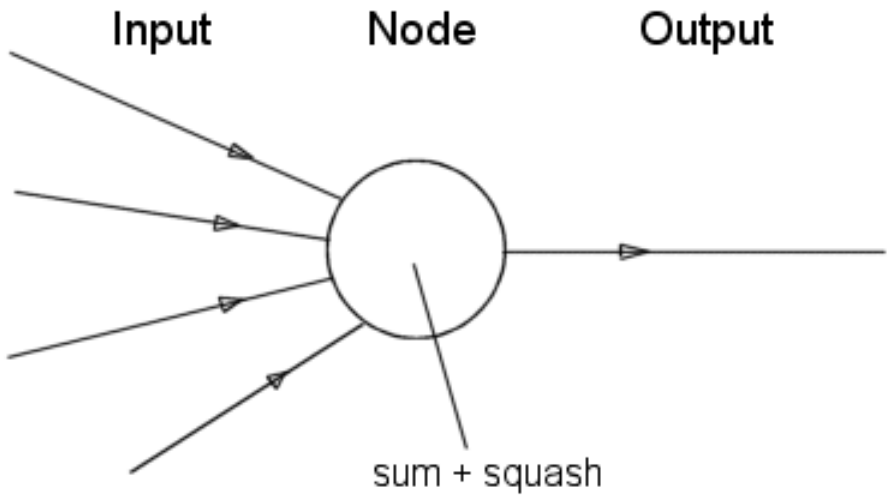


گمان می‌رود که مغز انسان از تعداد 10^{11} نرون تشکیل شده باشد که هر نرون با تقریبا 10^4 نرون دیگر در ارتباط است.

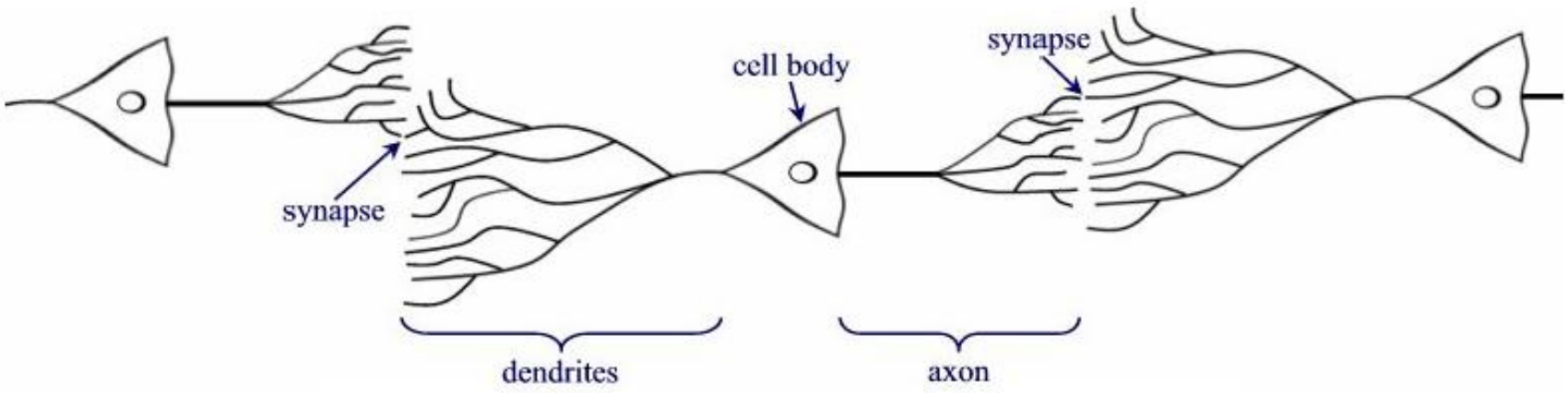
سرعت سوئیچنگ نرونها در حدود 10^{-3} ثانیه است که در مقایسه با کامپیوترها (10^{-10} ثانیه) بسیار ناچیز می‌نماید .با این وجود آدمی قادر است در 0.1 ثانیه تصویر یک انسان را بازشناسائی نماید .این قدرت فوق العاده باید از پردازش موازی توزیع شده در تعدادی زیادی از نرونها حاصل شده باشد.

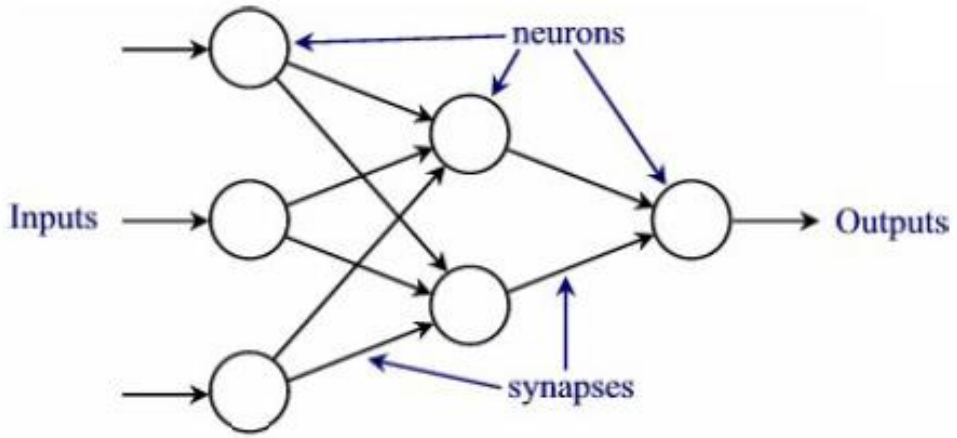
نرون در مقابل گره





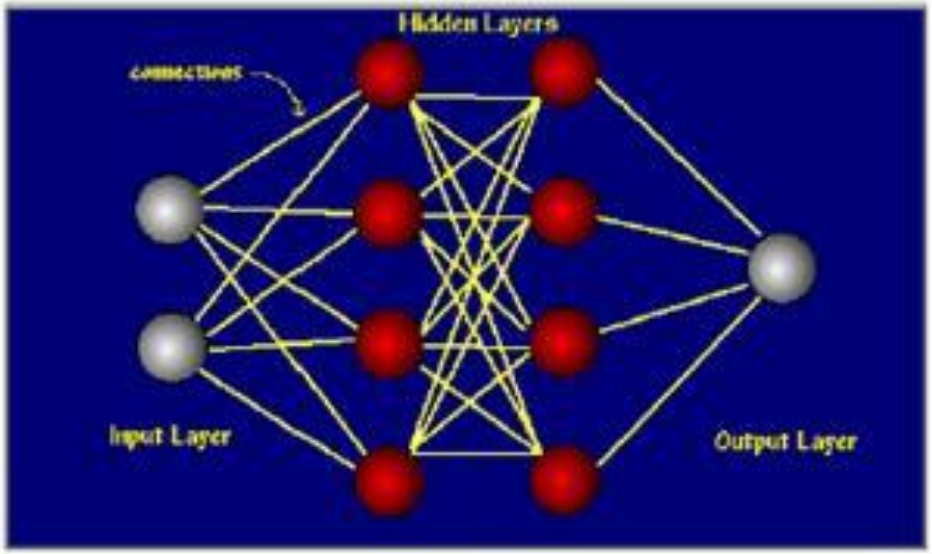
یادگیری در ارتباطات وزن‌دار نرون‌ها





معرفی ANN ها

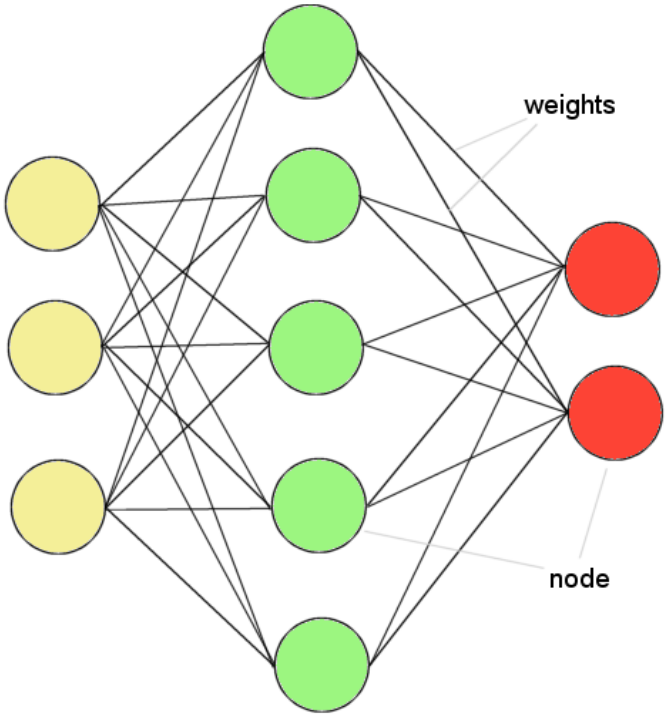
- یک سیستم پردازشی داده ها که از مغز انسان ایده گرفته است.
- در این شبکه ها ساختاری طراحی می شود که به آن node یا گره گفته می شود و می تواند همانند نورون عمل کند.
- با ایجاد شبکه بین این گرهها و اعمال یک الگوریتم آموزشی، شبکه الگویی را یاد می گیرد .
- در این حافظه یا شبکه ی عصبی هر لینک دارای یک وزن می باشد.



ANN

- شبکه عصبی مصنوعی، از دو نکته اصلی در شبکه عصبی اقتباس شده است.

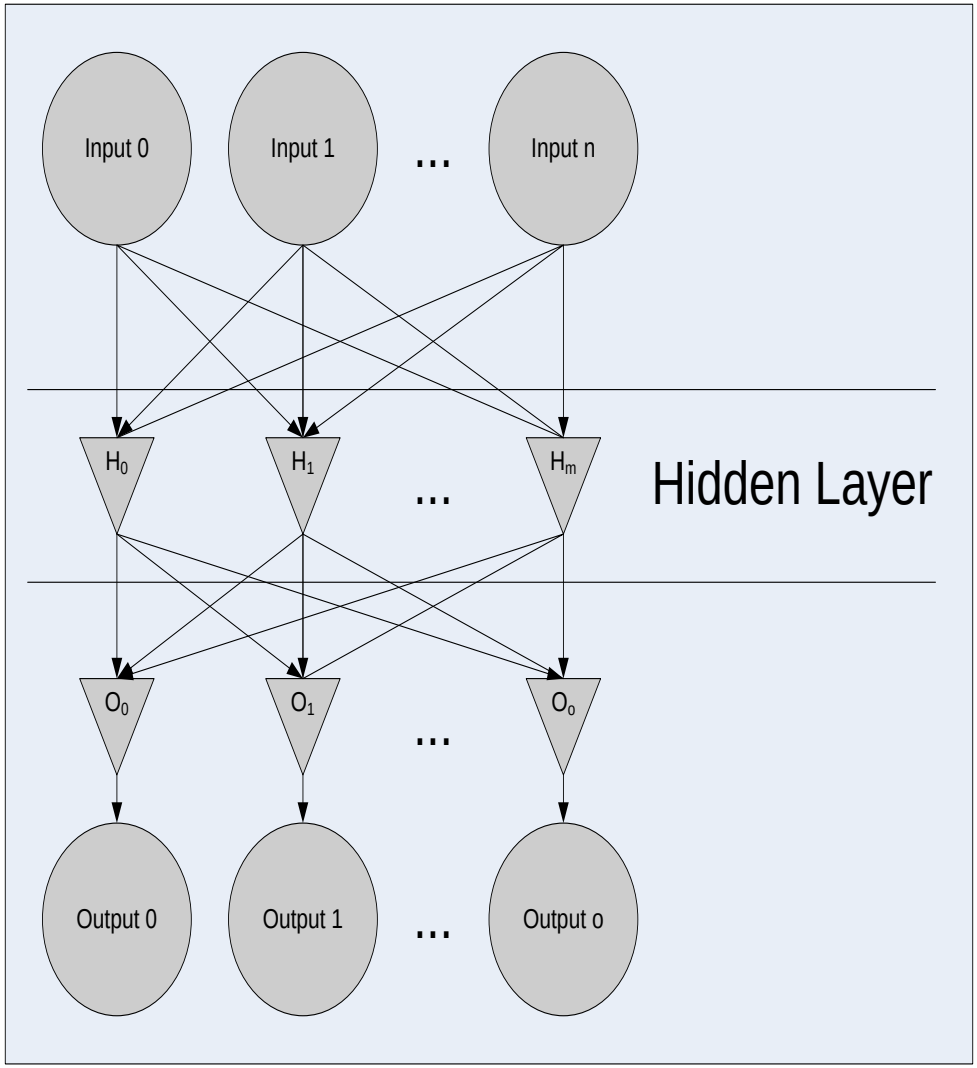
1. گره ها → نرون ها
2. وزن ها → سیناپس ها



مقدمه‌ای بر شبکه عصبی مصنوعی

- شبکه عصبی مصنوعی روشی عملی برای یادگیری توابع گوناگون نظیر توابع با مقادیر حقیقی، توابع با مقادیر گسسته و توابع با مقادیر برداری می‌باشد.
- یادگیری شبکه عصبی در برابر خطاهای داده های آموزشی مصون بوده و اینگونه شبکه ها با موفقیت به مسائلی نظیر شناسائی گفتار، شناسائی و تعبیر تصاویر، و یادگیری روبات اعمال شده است.

شبکه عصبی چیست؟



- روشی برای محاسبه است که بر پایه اتصال به هم پیوسته چندین واحد پردازشی ساخته می‌شود.
- شبکه از تعداد دلخواهی سلول یا گره یا واحد یا نرون تشکیل می‌شود که مجموعه ورودی را به خروجی ربط می‌دهند.

شبکه عصبی چه قابلیت‌هایی دارد؟

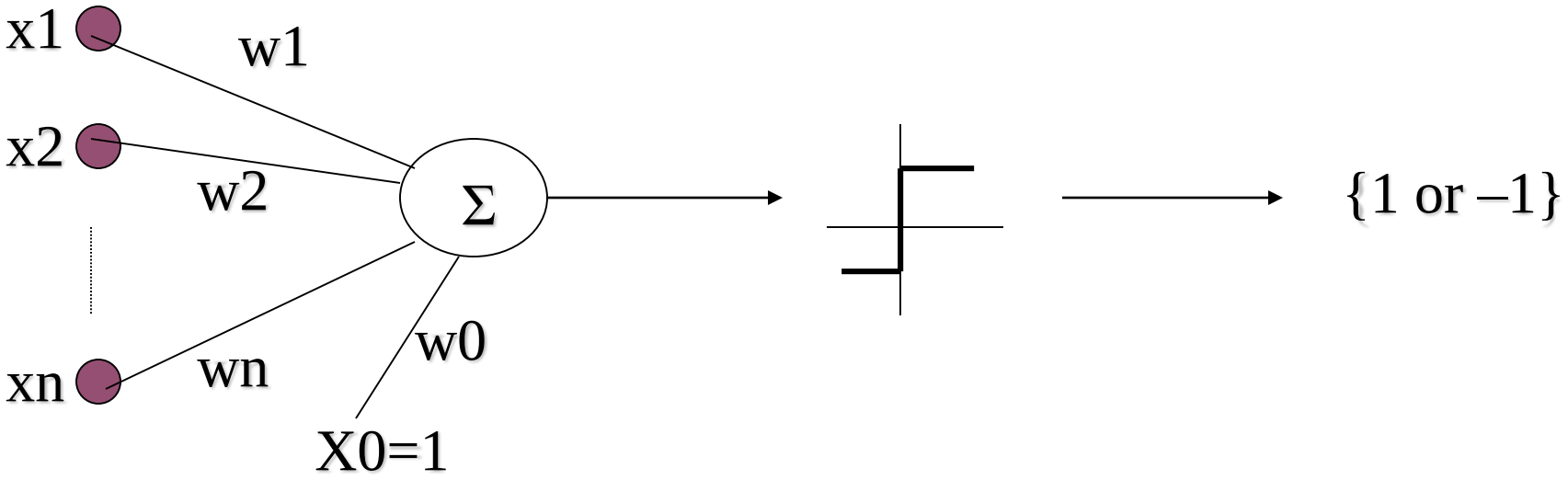
- محاسبه یک تابع معلوم
- تقریب یک تابع ناشناخته
- شناسائی الگو
- پردازش سیگنال
- یادگیری

مسائل مناسب برای یادگیری شبکه های عصبی

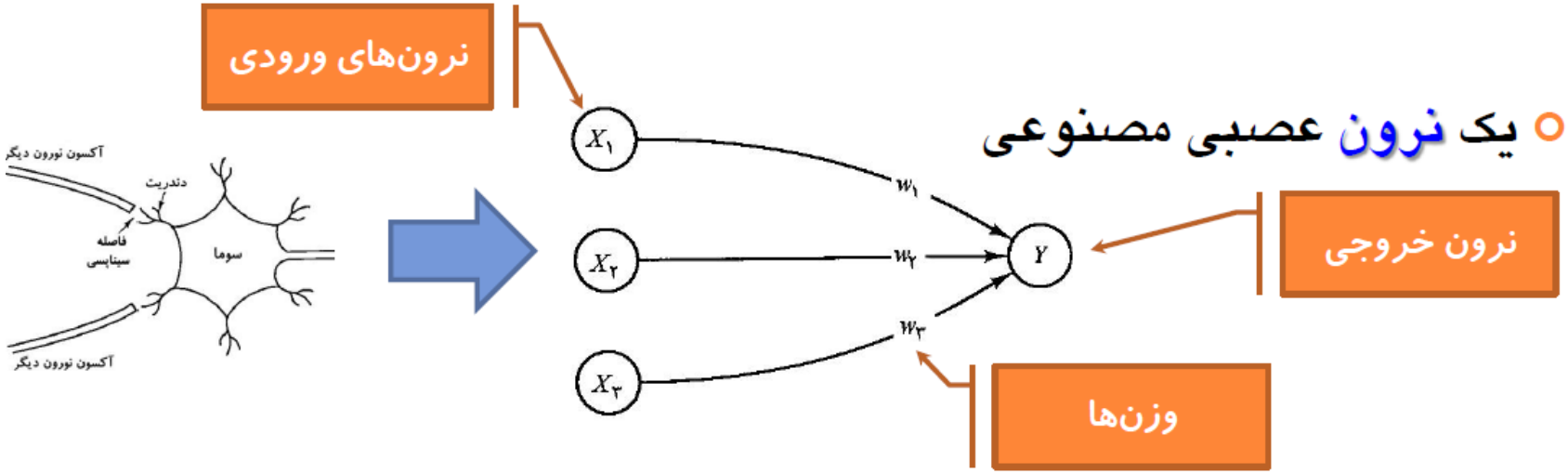
- خطا در داده های آموزشی وجود داشته باشد. مثل مسائلی که داده های آموزشی دارای نویز حاصل از دادهای سنسورها نظیر دوربین و میکروفن ها هستند.
- مواردی که نمونه‌ها توسط مقادیر زیادی زوج ویژگی-مقدار نشان داده شده باشند. نظیر داده های حاصل از یک دوربین ویدئویی.
- تابع هدف دارای مقادیر پیوسته باشد.
- زمان کافی برای یادگیری وجود داشته باشد. این روش در مقایسه با روش‌های دیگر نظیر درخت تصمیم نیاز به زمان بیشتری برای یادگیری دارد.
- نیازی به تعبیر تابع هدف نباشد ،زیرا به سختی می‌توان اوزان یادگرفته شده توسط شبکه را تعبیر نمود.

Perceptron

- نوعی از شبکه عصبی برمبنای یک واحد محاسباتی به نام پرسپترون ساخته میشود .یک پرسپترون برداری از ورودیهای با مقادیر حقیقی را گرفته و یک ترکیب خطی از این ورودیها را محاسبه میکند .اگر حاصل از یک مقدار آستانه بیشتر بود خروجی پرسپترون برابر با 1 و در غیر اینصورت معادل 1- خواهد بود.



یک نرون عصبی مصنوعی



- فعال‌سازی‌ها یا سیگنال‌های خروجی نرون‌های ورودی به ترتیب x1، x2 و x3 هستند
- ورودی شبکه به نرون Y، حاصل‌جمع وزن‌دار سیگنال‌های ورودی و وزن‌هاست:

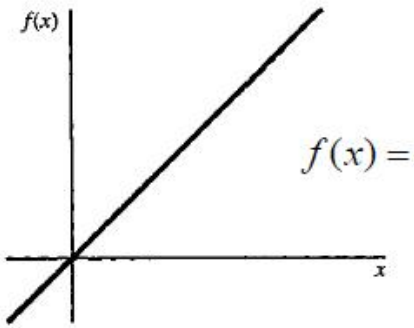
$$y\_in = w_1 x_1 + w_2 x_2 + w_3 x_3 = \sum_i w_i x_i$$

- فعال‌سازی نرون Y با اعمال تابع فعال‌سازی f روی ورودی آن به دست می‌آید
  - تابع سیگموید (Sigmoid)

$$y = f(y\_in) \qquad f(x) = \frac{1}{1+\exp(-x)}$$

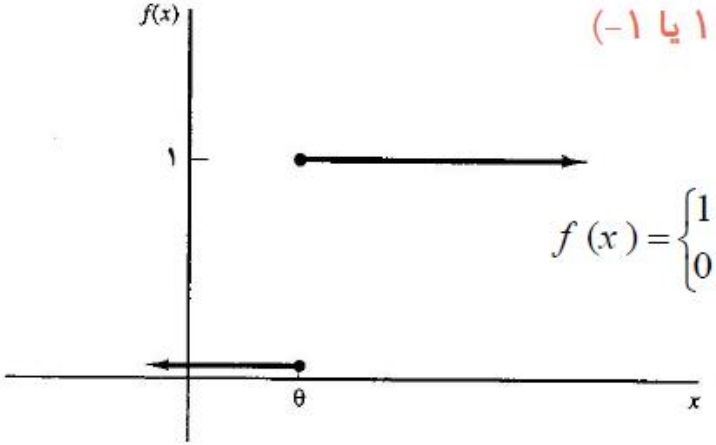
توابع فعال‌سازی متداول . . .

- **تابع همانی** (Identity Function)
  - برای واحدهای ورودی



$$f(x) = x \qquad for\ all\ x$$

- **تابع پله‌ای دودویی** (Step Function)
  - تابع آستانه (Threshold Function) یا تابع هویساید (Heaviside Function)
  - خروجی = سیگنال دودویی (۱ یا ۰) یا دوقطبی (Bipolar) (۱ یا ۱-)



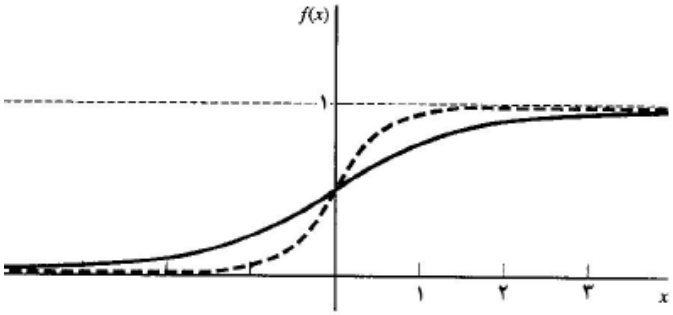
$$f(x) = \begin{cases} 1 & if \quad x > \theta \\ 0 & if \quad x < \theta \end{cases}$$

توابع فعال‌سازی متداول

- **توابع سیگموید** (Sigmoid Functions)
- **منحنی‌هایی به شکل S**
  - استفاده در شبکه‌های عصبی پس‌انتشار (نیاز به مشتق‌گیری)
  - سیگموید دودویی- تابع لجستیک (Logistic Function)
    - دامنهٔ 0 تا 1 ، مقادیر مطلوب خروجی یا دودویی است و یا بین 0 و 1 است
  - سیگموید دوقطبی- شبیه به تابع تانژانت هیپربولیک (Hyperbolic Tangent Function)
    - دامنهٔ 1- تا 1

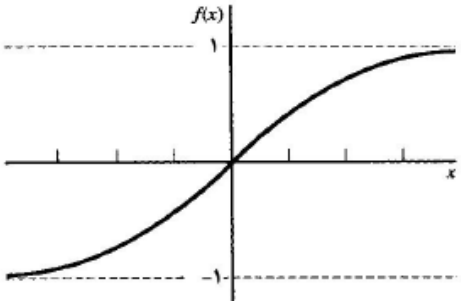
$$f(x)=\frac{1}{1+\exp(-\sigma x)}$$

$$f'(x)=\sigma f(x)[1-f(x)]$$



$$g(x)=2f(x)-1=\frac{2}{1+\exp(-\sigma x)}-1=\frac{1-\exp(-\sigma x)}{1+\exp(-\sigma x)}$$

$$g'(x)=\frac{\sigma}{2}[1+g(x)][1-g(x)]$$



یادگیری یک پرسپترون

- خروجی پرسپترون توسط رابطه زیر مشخص می‌شود:

$$O(x1,x2,\ldots,xn) = \begin{cases} 1 \text{ if } w0 + w1x1 + w2x2 + \ldots + wnxn > 0 \\ -1 \text{ otherwise} \end{cases}$$

- که برای سادگی آنرا می‌توان بصورت زیر نشان داد:

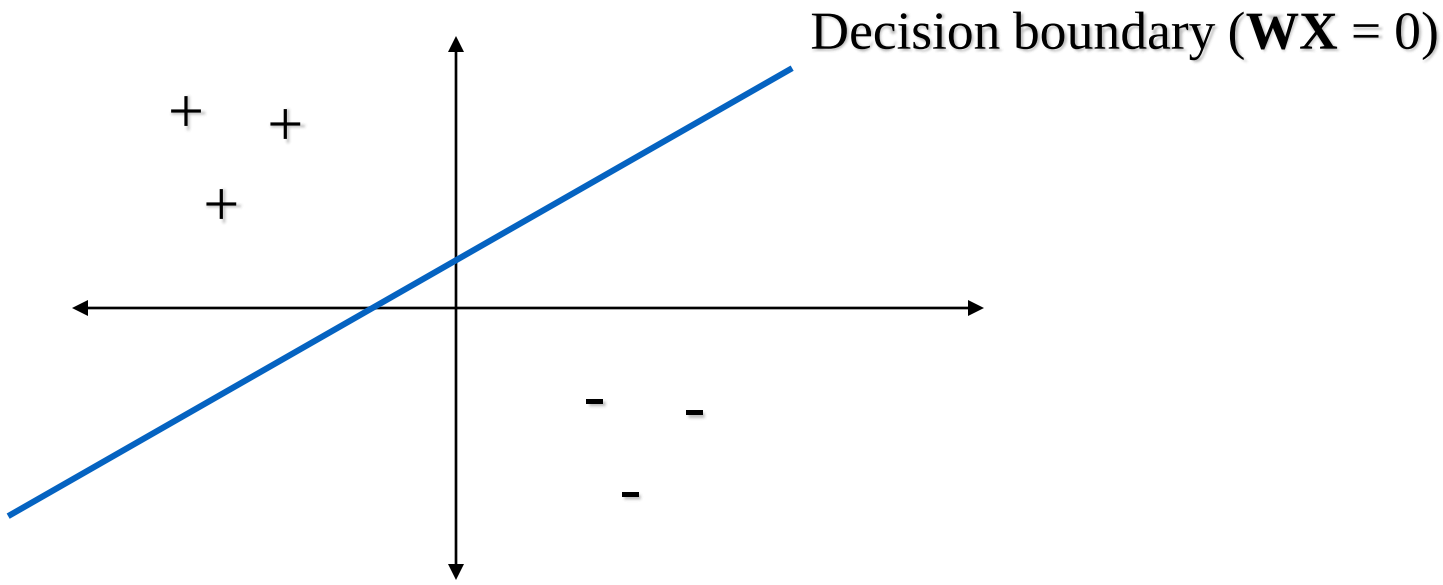
$$O(\mathbf{X}) = \text{sgn}(\mathbf{WX}) \text{ where}$$

$$\text{Sgn}(y) = \begin{cases} 1 \text{ if } y > 0 \\ -1 \text{ otherwise} \end{cases}$$

یادگیری پرسپترون عبارت است از: پیدا کردن مقادیردرستی برای W
بنابراین فضای فرضیه H در یادگیری پرسپترون عبارت است ازمجموعه تمام مقادیر حقیقی ممکن برای بردارهای وزن.

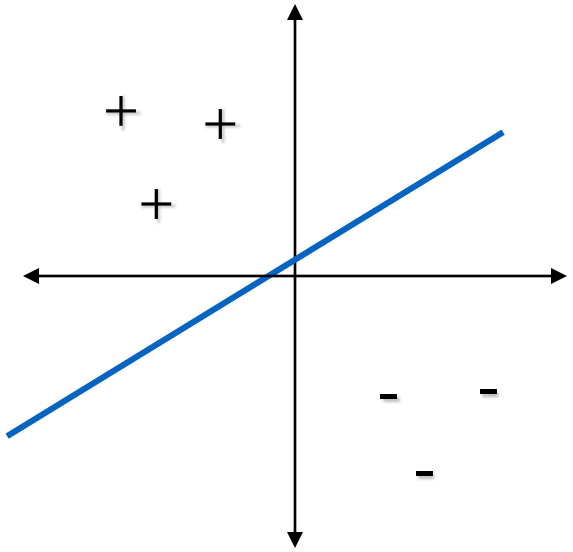
توانائ پرسپترون

- پرسپترون را می‌توان بصورت یک سطح تصمیم hyperplane در فضای n بعدی نمونه ها در نظر گرفت. پرسپترون برای نمونه‌های یک طرف صفحه مقدار 1 و برای مقادیر طرف دیگر مقدار 1- بوجود می‌آورد.

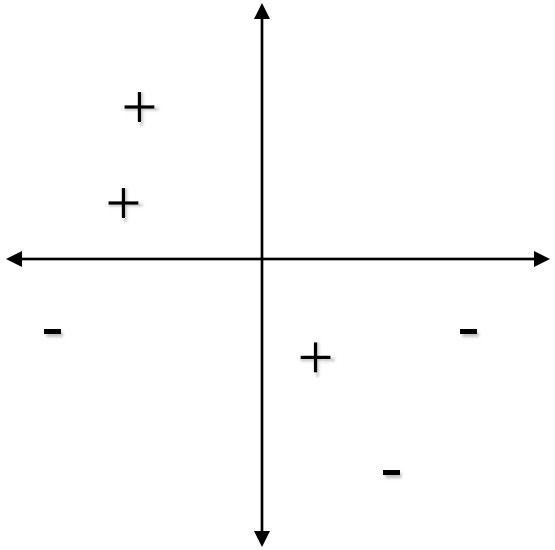


توابعی که پرسپترون قادر به یادگیری آنها می‌باشد

- یک پرسپترون فقط قادر است مثالهائی را یاد بگیرد که بصورت خطی جداپذیر باشند .اینگونه مثال‌ها مواردی هستند که بطور کامل توسط یکhyperplane قابل جدا سازی می‌باشند.



Linearly separable

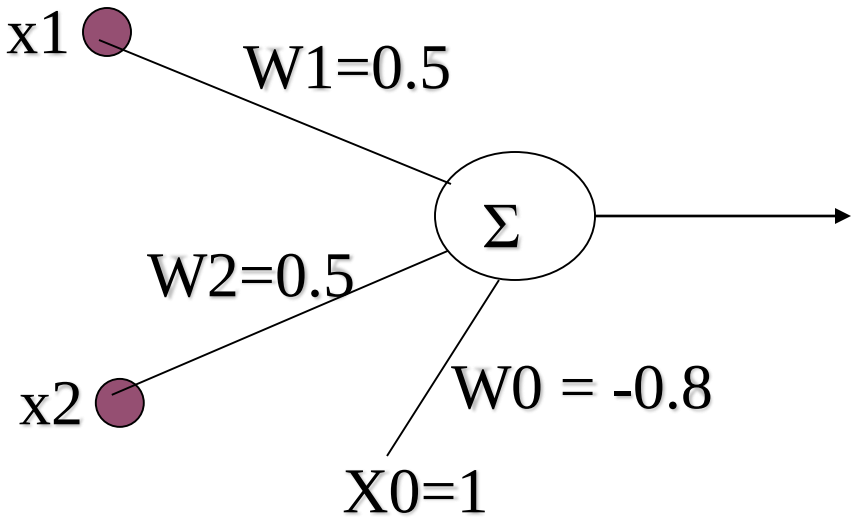


Non-linearly separable

توابع بولی و پرسپترون

- یک پرسپترون میتواند بسیاری از توابع بولی را نمایش دهد نظیر AND, OR, NAND, NOR
- اما نمی‌تواند XOR را نمایش دهد.

AND:

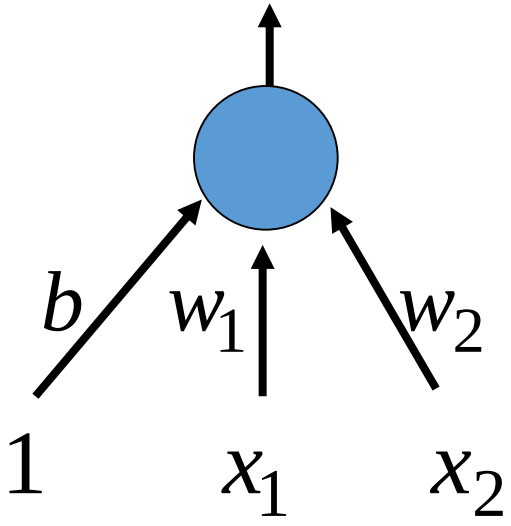


- در واقع هر تابع بولی را میتوان با شبکه ای دوسطحی از پرسپترونها نشان داد.

اضافه کردن بایاس

- افزودن بایاس موجب میشود تا استفاده از شبکه پرسپترون با سهولت بیشتری انجام شود.
- برای اینکه برای یادگیری بایاس نیازی به استفاده از قانون دیگری نداشته باشیم بایاس را بصورت یک ورودی با مقدار ثابت ۱ در نظر گرفته و وزن W0 را به آن اختصاص میدهیم.

$$\hat{y} = b + \sum_i x_i w_i$$



$$\hat{y} = w_0 + \sum_{i=1} x_i w_i$$

آموزش پرسپترون

- چگونه وزنهای یک پرسپترون واحد را یاد بگیریم به نحوی که پرسپترون برای مثالهای آموزشی مقادیر صحیح را ایجاد نماید؟
- دو راه مختلف:
  - قانون پرسپترون
  - قانون دلتا

آموزش پرسپترون

الگوریتم یادگیری پرسپترون

1. مقادیری تصادفی به وزنها نسبت می‌دهیم
2. پرسپترون را به تک تک مثالهای آموزشی اعمال میکنیم .اگر مثال غلط ارزیابی شود مقادیر وزنهای پرسپترون را تصحیح میکنیم.
3. آیا تمامی مثال‌های آموزشی درست ارزیابی می‌شوند:
   - بله ← پایان الگوریتم
   - خیر←به مرحله 2 برمیگردیم

قانون پرسپترون

- مرحلهٔ ۵- اگر خطایی رخ داده است، وزن‌ها و بایاس را به‌روز کنید.

به‌روز کردن مشروط وزن‌ها

اگر $y \neq t$ است، آنگاه:

$$w_i(new) = w_i(old) + \alpha x_i t$$
$$b(new) = b(old) + \alpha t$$

در غیراین‌صورت:

$$w_i(new) = w_i(old)$$
$$b(new) = b(old)$$

- مرحلهٔ ۶- شرایط توقف را آزمایش کنید:
  - اگر در مرحلهٔ ۲ هیچ وزنی تغییر نکرد، الگوریتم را متوقف کنید، در غیراین‌صورت ادامه دهید.

خطا = برابر نبودن پاسخ شبکه و مقدار هدف

نرخ یادگیری

قانون پرسپترون

- برای یک مثال آموزشی $X = (x1, x2, ..., xn)$ در هر مرحله وزنها بر اساس قانون پرسپترون بصورت زیر تغییر می‌کند:

$$wi = wi + \Delta wi$$

که در آن

$$\Delta wi = \eta (t - o) xi$$

t: target output

o: output generated by the perceptron

η: constant called the learning rate (e.g., 0.1)

اثبات شده است که برای یک مجموعه مثال جداپذیرخطی این روش همگرا شده و پرسپترون قادر به جدا سازی صحیح مثال‌ها خواهد شد.

قانون دلتا (Delta Rule)

- وقتی که مثال‌ها بصورت خطی جداپذیر نباشند قانون پرسپترون همگرا نخواهد شد .برای غلبه بر این مشکل از قانون دلتا استفاده می‌شود .
- ایده اصلی این قانون استفاده از gradient descent برای جستجو در فضای فرضیه وزن‌های ممکن می‌باشد .این قانون پایه روش Backpropagation است که برای آموزش شبکه با چندین نرون به هم متصل بکار می‌رود.
- همچنین این روش پایه ای برای انواع الگوریتم‌های یادگیری است که باید فضای فرضیه‌ای شامل فرضیه های مختلف پیوسته را جستجو کنند.

قانون دلتا (Delta Rule)

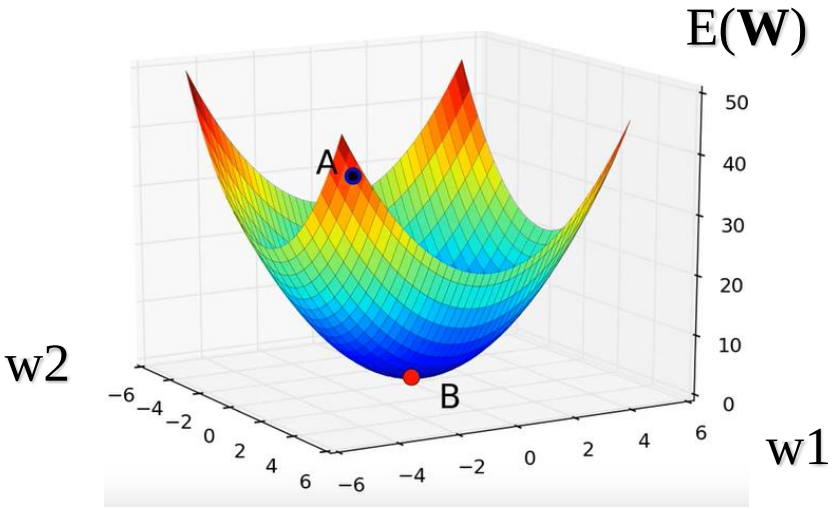
- برای درک بهتر این روش آنرا به یک پرسپترون فاقد حد آستانه اعمال می‌کنیم .در انجا لازم است ابتدا تعریفی برای خطا ی آموزش ارائه شود .یک تعریف متداول این چنین است:

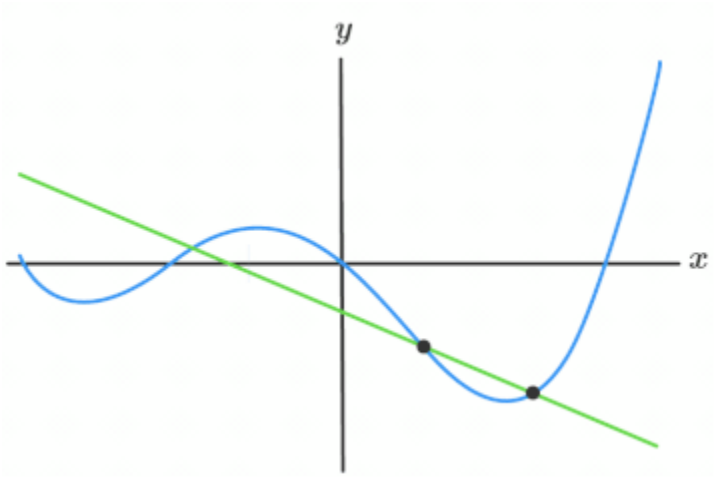
$$E = \frac{1}{2} \Sigma_i (t_i - o_i)^2$$

- که این مجموع برای تمام مثالهای آموزشی انجام می‌شود.

الگوریتم gradient descent

- با توجه به نحوه تعریف E سطح خطا بصورت یک سهمی خواهد بود .ما بدنبال وزنهائی هستیم که حداقل خطا را داشته باشند . الگوریتم gradient descent در فضای وزنها بدنبال برداری میگردد که خطا را حداقل کند .این الگوریتم از یک مقدار دلخواه برای بردار وزن شروع کرده و در هر مرحله وزنها را طوری تغییر می‌دهد که در جهت شیب کاهشی منحنی فوق خطا کاهش داده شود.





بدست آوردن قانون gradient descent

- ایده اصلی :گرادیان همواره در جهت افزایش شیب E عمل میکند.
- گرادیان E نسبت به بردار وزن w بصورت زیر تعریف میشود:

$$\nabla E(W) = [\, E'/w0,\ E'/w1,\ \ldots,\ E'/wn]$$

- که در آن $\nabla E(W)$ یک برداروE’ مشتق جزئی نسبت به هر وزن می‌باشد.

قانون دلتا Delta Rule

- برای یک مثال آموزشی $X = (x1, x2, ..., xn)$ در هر مرحله وزنها بر اساس قانون دلتا بصورت زیر تغییر می‌کند:

$$wi = wi + \Delta wi$$

$$\text{Where } \Delta wi = -\eta \; E'(\mathbf{W})/wi$$

η: learning rate (e.g., 0.1)

علامت منفی نشان دهنده حرکت در جهت کاهش شیب است.

قانون دلتا Delta Rule

$$\frac{\partial E}{\partial w_i} = \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

$$= \frac{1}{2} \sum_{d \in D} \frac{\partial}{\partial w_i} (t_d - o_d)^2$$

$$= \frac{1}{2} \sum_{d \in D} 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d)$$

$$= \sum_{d \in D} (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x}_d)$$

$$\frac{\partial E}{\partial w_i} = \sum_{d \in D} (t_d - o_d)(-x_{id}) \qquad \Delta w_i = \eta \sum_{d \in D} (t_d - o_d)\, x_{id}$$

محاسبه گرادیان

- با مشتق گیری جزئی از رابطه خطا می‌توان بسادگی گرادیان را محاسبه نمود:

$$E'(W)/ w_i = \Sigma_i (t_i - O_i) (-x_i)$$

- لذا وزنها طبق رابطه زیر تغییر خواهند نمود.

$$\Delta wi = \eta \Sigma i (ti - oi) xi$$

خلاصه یادگیری قانون دلتا

GRADIENT-DESCENT(*training_examples*, η)

Each training example is a pair of the form $\langle \vec{x}, t \rangle$, *where* $\vec{x}$ *is the vector of input values, and* t *is the target output value.* η *is the learning rate (e.g., .05).*

- Initialize each w_i to some small random value
- Until the termination condition is met, Do
  - Initialize each Δw_i to zero.
  - For each $\langle \vec{x}, t \rangle$ in *training_examples*, Do
    - Input the instance $\vec{x}$ to the unit and compute the output o
    - For each linear unit weight w_i, Do

$$\Delta w_i \leftarrow \Delta w_i + \eta(t - o)x_i \qquad \text{(T4.1)}$$

  - For each linear unit weight w_i, Do

$$w_i \leftarrow w_i + \Delta w_i \qquad \text{(T4.2)}$$

مشکلات روش gradient descent

1. ممکن است همگرا شدن به یک مقدار مینیمم زمان زیادی لازم داشته باشد.
2. اگر در سطح خطا چندین مینیمم محلی وجود داشته باشد تضمینی وجود ندارد که الگوریتم مینیمم مطلق را پیدا بکند.

در ضمن این روش وقتی قابل استفاده است که:

- فضای فرضیه دارای فرضیه های پارامتریک پیوسته باشد.
- رابطه خطا قابل مشتق گیری باشد

تقریب افزایشی gradient descent

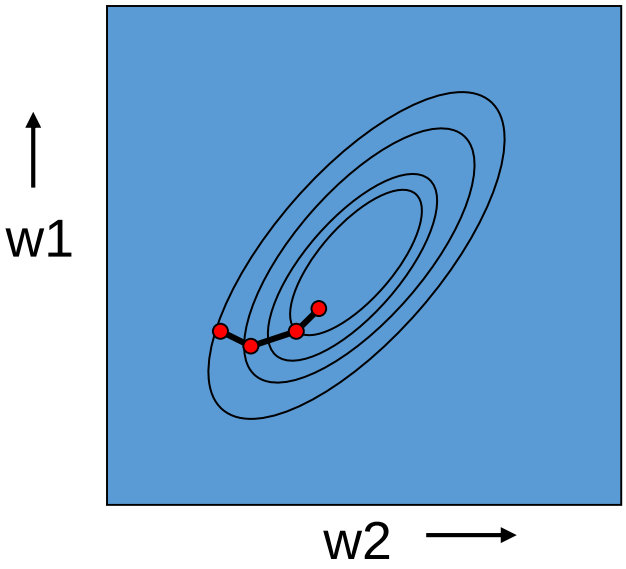
- میتوان بجای تغییر وزنها پس از مشاهده همه مثالها، آنها را بازا هر مثال مشاهده شده تغییر داد . در این حالت وزنها بصورت افزایشی incremental تغییر میکنند .این روش را stochastic gradient descentنیزمینامند.

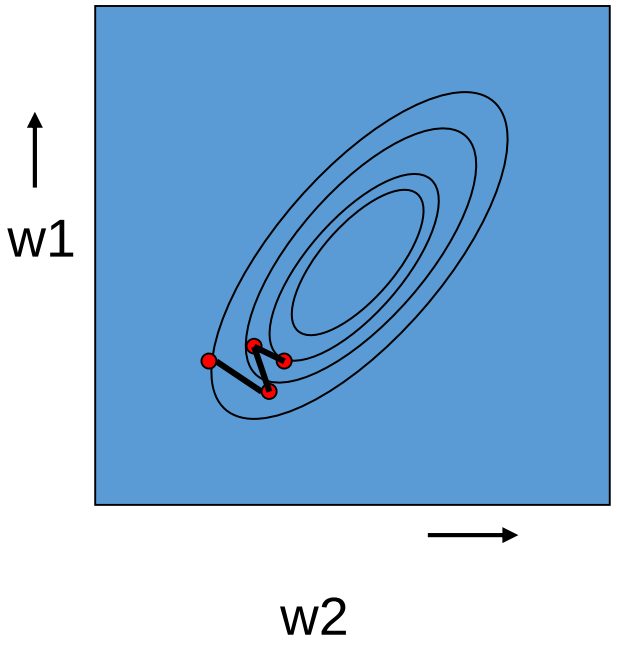
$$\nabla w_i = \eta (t-o) x_i$$

در بعضی موارد تغییر افزایشی وزنها میتواند از بروز مینیمم محلی جلوگیری کند .روش استاندارد نیاز به محاسبات بیشتری دارد درعوض میتواند طول step بزرگتری هم داشته باشد.

مقایسه آموزش یکجا و افزایشی

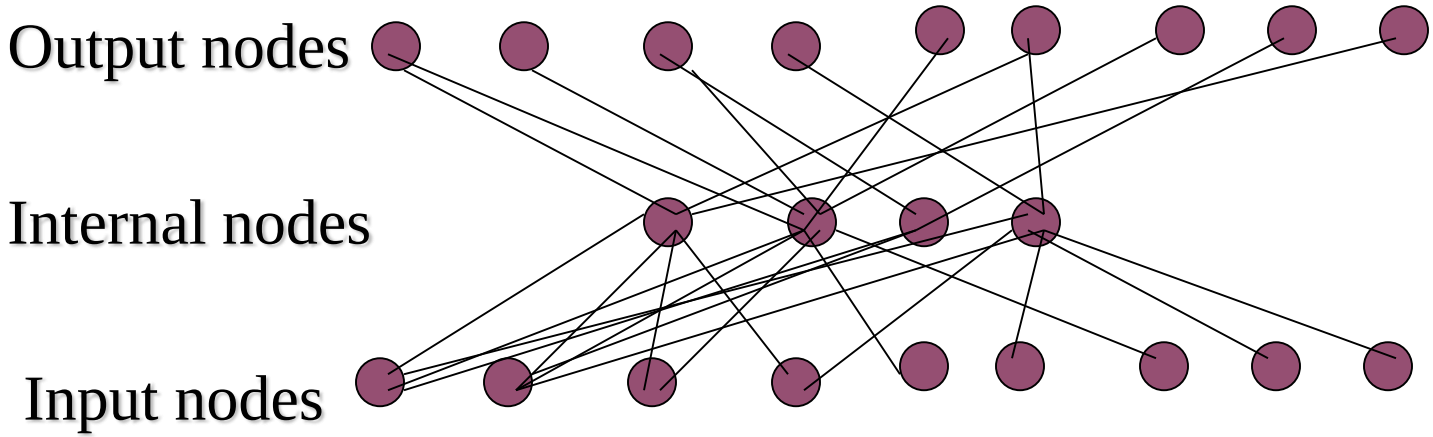
- آموزش یکجا (Batch learning)
- آموزش افزایشی (Online learning)



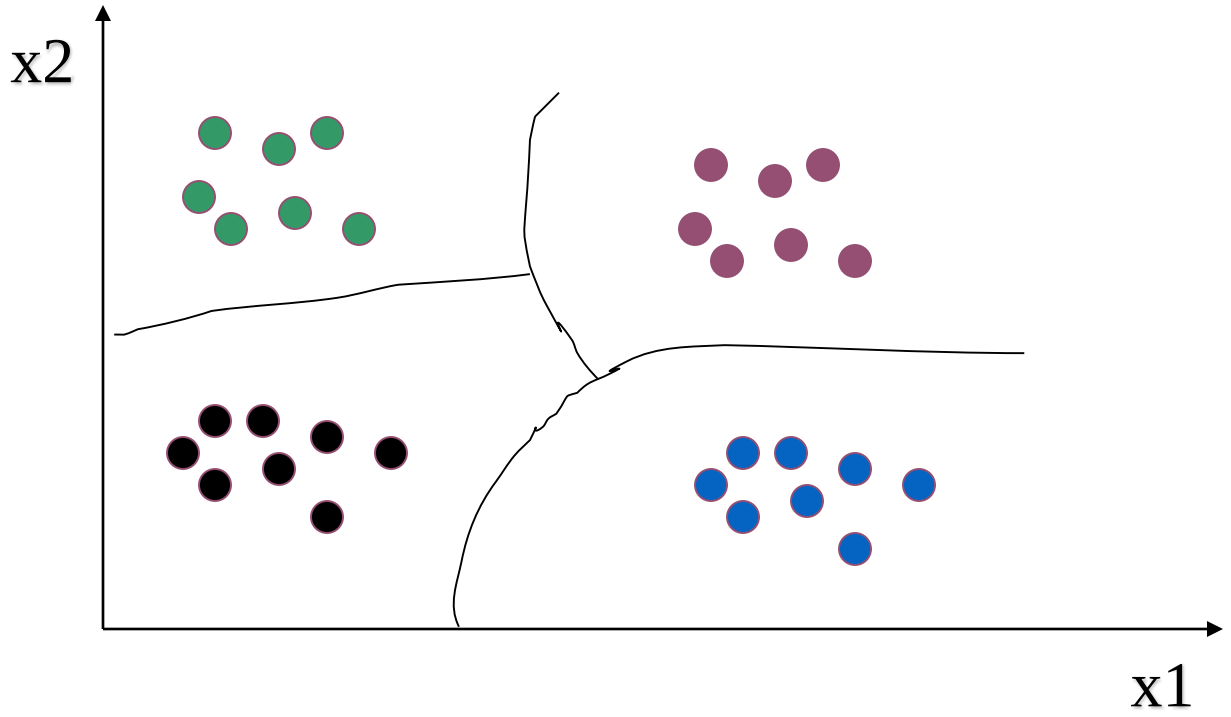


شبکه های چند لایه

بر خلاف پرسپترونها شبکه های چند لایه میتوانند برای یادگیری مسائل غیر خطی و همچنین مسائلی با تصمیم گیری های متعدد بکار روند.

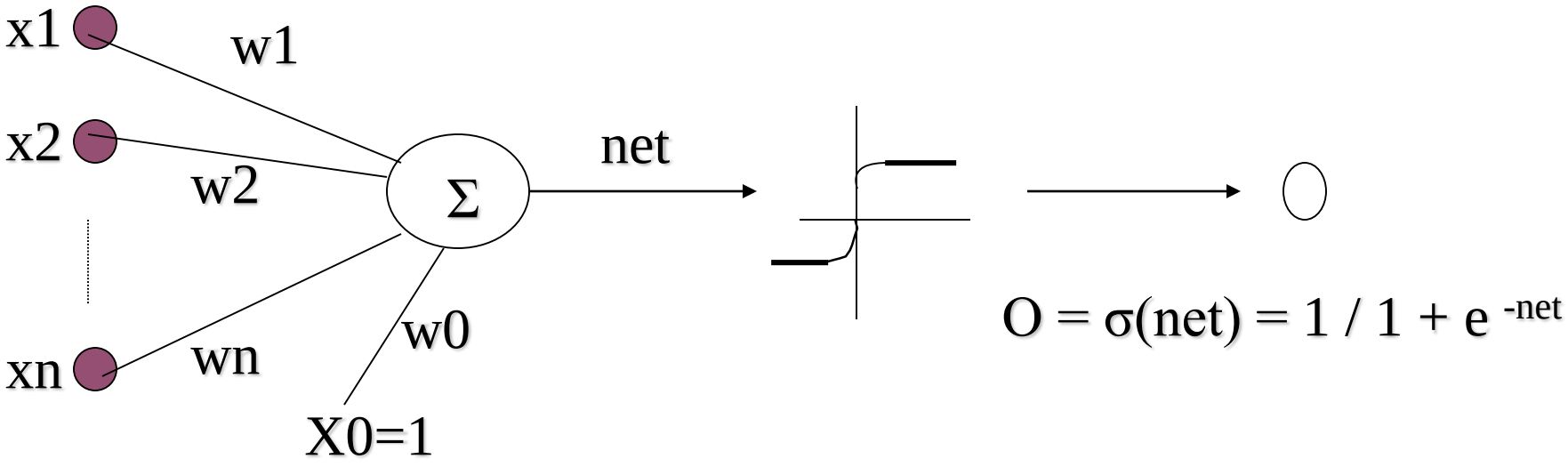


مثال



یک سلول واحد

برای اینکه بتوانیم فضای تصمیم گیری را بصورت غیر خطی از هم جدا بکنیم، لازم است تا هر سلول واحد را بصورت یک تابع غیر خطی تعریف نمائیم .مثالی از چنین سلولی میتواند یک واحد سیگموئید باشد:



تابع سیگموئید

خروجی این سلول واحد را بصورت زیر میتوان بیان نمود:

$$O(x1,x2,\ldots,xn) = \sigma (WX)$$

where: $\sigma (WX) = 1 / 1 + e^{-WX}$

تابع σ تابع سیگموئید یا لجستیک نامیده میشود .این تابع دارای خاصیت زیر است:

$$d\,\sigma(y) / dy = \sigma(y)\ (1 - \sigma(y))$$

الگوریتم Back propagation

- برای یادگیری وزن های یک شبکه چند لایه از روش Back Propagation استفاده میشود . در این روش با استفاده از gradient descent سعی میشود تا مربع خطای بین خروجی های شبکه و تابع هدف مینیمم شود.
- خطا بصورت زیر تعریف میشود:

$$E\left(\vec{W}\right) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} \left(t_{kd} - o_{kd} \right)^2$$

مراد از outputsخروجیهای مجموعه واحد های لایه خروجی وt_{kd} و o_{kd} مقدار هدف و خروجی متناظر با k امین واحد خروجی و مثال آموزشی d است.

الگوریتم Back propagation

- فضای فرضیه مورد جستجو در این روش عبارت است از فضای بزرگی که توسط همه مقادیر ممکن برای وزنها تعریف می‌شود .روش gradient descent سعی میکند تا با مینیمم کردن خطا به فرضیه مناسبی دست پیدا کند .اما تضمینی برای اینکه این الگوریتم به مینیمم مطلق برسد وجود ندارد.

الگوریتم BP

1. شبکه ای با n_{in} گره ورودی، n_{hidden} گره مخفی، و n_{out} گره خروجی ایجاد کنید .
2. همه وزنها را با یک مقدار تصادفی کوچک عدد دهی کنید.
3. تا رسیدن به شرط پایانی (کوچک شدن خطا)مراحل زیر را انجام دهید:

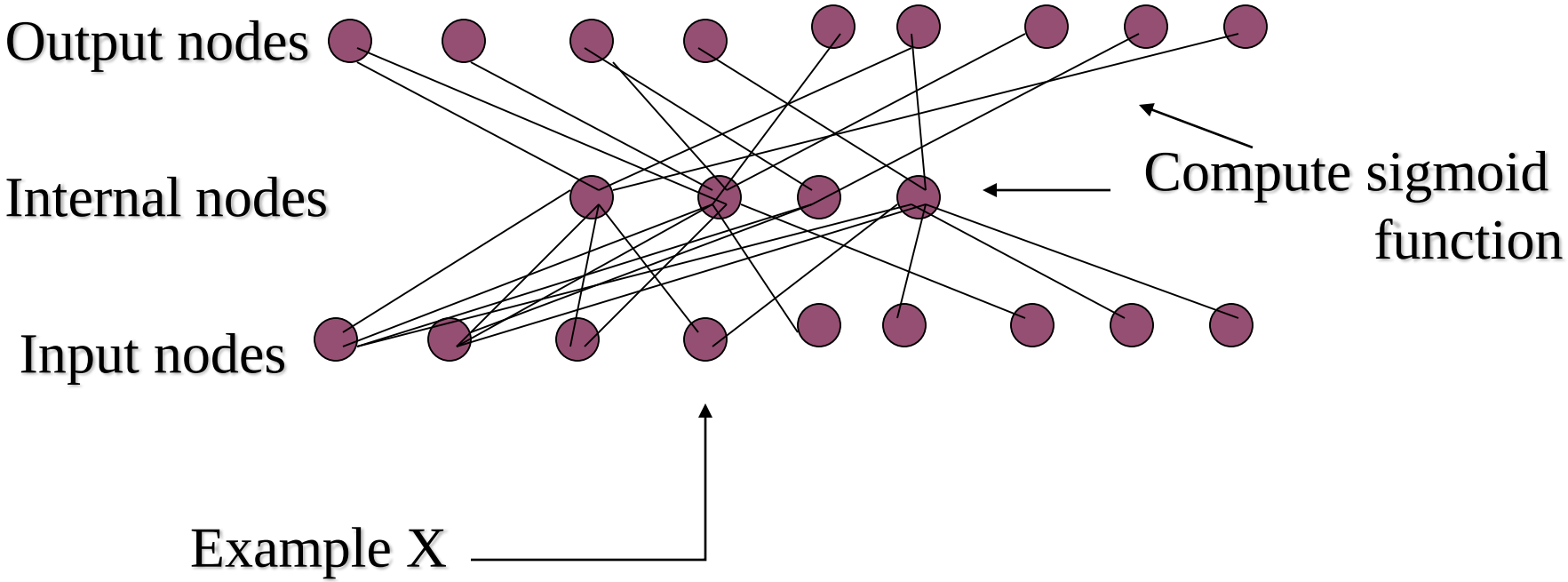
برای هر x متعلق به مثالهای آموزشی:

- مثال X را به سمت جلو در شبکه انتشار دهید
- خطای E را به سمت عقب در شبکه انتشار دهید.

هر مثال آموزشی بصورت یک زوج (x,t) ارائه می‌شود که بردار x مقادیر ورودی و بردار t مقادیر هدف برای خروجی شبکه را تعیین میکنند.

انتشار به سمت جلو

- برای هر مثال X مقدار خروجی هر واحد را محاسبه کنید تا به گره های خروجی برسید.



انتشار به سمت عقب

1. برای هر واحد خروجی جمله خطا را بصورت زیر محاسبه کنید:
2. $\delta_k = O_k (1-O_k)(t_k - O_k)$
3. برای هر واحد مخفی جمله خطا را بصورت زیر محاسبه کنید:
4. $\delta_h = O_h (1-O_h) \Sigma_k W_{kh} \delta_k$
5. مقدارهر وزن را بصورت زیر تغییر دهید:

$$W_{ji} = W_{ji} + \Delta W_{ji}$$

که در آن:

$$\Delta W_{ji} = \eta \delta_j X_{ji}$$

η عبارت است از نرخ یادگیری

BACKPROPAGATION($training\_examples, \eta, n_{in}, n_{out}, n_{hidden}$)

Each training example is a pair of the form $\langle \vec{x}, \vec{t} \rangle$, where $\vec{x}$ is the vector of network input values, and $\vec{t}$ is the vector of target network output values.

η is the learning rate (e.g., .05). n_{in} is the number of network inputs, n_{hidden} the number of units in the hidden layer, and n_{out} the number of output units.

The input from unit i into unit j is denoted x_{ji}, and the weight from unit i to unit j is denoted w_{ji}.

- Create a feed-forward network with n_{in} inputs, n_{hidden} hidden units, and n_{out} output units.
- Initialize all network weights to small random numbers (e.g., between $-.05$ and .05).
- Until the termination condition is met, Do
  - For each $\langle \vec{x}, \vec{t} \rangle$ in *training_examples*, Do

Propagate the input forward through the network:

1. Input the instance $\vec{x}$ to the network and compute the output o_u of every unit u in the network.

Propagate the errors backward through the network:

2. For each network output unit k, calculate its error term δ_k

$$\delta_k \leftarrow o_k(1 - o_k)(t_k - o_k) \tag{T4.3}$$

3. For each hidden unit h, calculate its error term δ_h

$$\delta_h \leftarrow o_h(1 - o_h) \sum_{k \in outputs} w_{kh}\delta_k \tag{T4.4}$$

4. Update each network weight w_{ji}

$$w_{ji} \leftarrow w_{ji} + \Delta w_{ji}$$

where

$$\Delta w_{ji} = \eta \, \delta_j \, x_{ji} \tag{T4.5}$$

- x_{ji} = the ith input to unit j
- w_{ji} = the weight associated with the ith input to unit j
- $net_j = \sum_i w_{ji} x_{ji}$ (the weighted sum of inputs for unit j)
- o_j = the output computed by unit j
- t_j = the target output for unit j
- σ = the sigmoid function
- $outputs$ = the set of units in the final layer of the network
- $Downstream(j)$ = the set of units whose immediate inputs include the output of unit j

Training Rule for Output Unit Weights

$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}}$$

$$= \frac{\partial E_d}{\partial net_j} x_{ji}$$

$$\frac{\partial E_d}{\partial net_j} = \frac{\partial E_d}{\partial o_j}\frac{\partial o_j}{\partial net_j}$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j}\frac{1}{2}\sum_{k \in outputs}(t_k - o_k)^2$$

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j}\frac{1}{2}(t_j - o_j)^2$$

$$= \frac{1}{2}2(t_j - o_j)\frac{\partial (t_j - o_j)}{\partial o_j}$$

$$= -(t_j - o_j)$$

Training Rule for Output Unit Weights

$$\frac{\partial o_j}{\partial net_j} = \frac{\partial \sigma(net_j)}{\partial net_j}$$

$$= o_j(1 - o_j)$$

$$\frac{\partial E_d}{\partial net_j} = -(t_j - o_j)\, o_j(1 - o_j)$$

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}} = \eta\, (t_j - o_j)\, o_j(1 - o_j) x_{ji}$$

Training Rule for Hidden Unit Weights

$$\frac{\partial E_d}{\partial net_j} = \sum_{k \in Downstream(j)} \frac{\partial E_d}{\partial net_k} \frac{\partial net_k}{\partial net_j}$$

$$= \sum_{k \in Downstream(j)} -\delta_k \frac{\partial net_k}{\partial net_j} \qquad \delta_j = o_j(1 - o_j) \sum_{k \in Downstream(j)} \delta_k \, w_{kj}$$

$$= \sum_{k \in Downstream(j)} -\delta_k \frac{\partial net_k}{\partial o_j} \frac{\partial o_j}{\partial net_j}$$

$$\Delta w_{ji} = \eta \, \delta_j \, x_{ji}$$

$$= \sum_{k \in Downstream(j)} -\delta_k \, w_{kj} \frac{\partial o_j}{\partial net_j}$$

$$= \sum_{k \in Downstream(j)} -\delta_k \, w_{kj} \, o_j(1 - o_j)$$

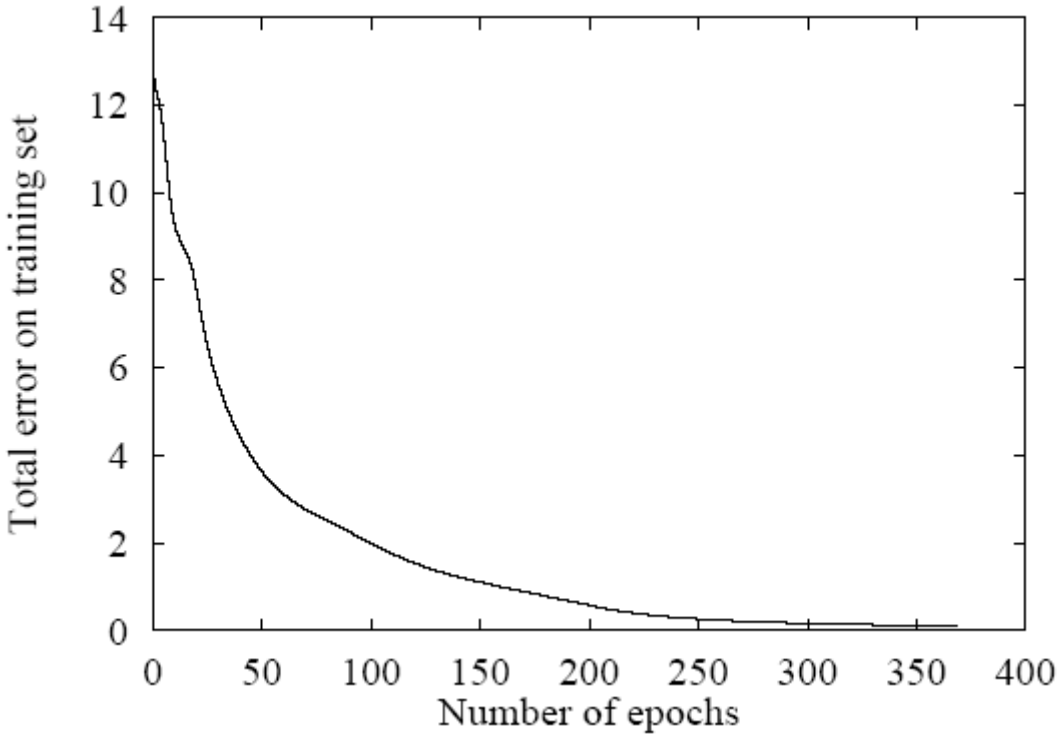
شرط خاتمه

معمولا الگوریتم BP پیش از خاتمه هزاران بار با استفاده همان داده های آموزشی تکرار میگردد شروط مختلفی را میتوان برای خاتمه الگوریتم بکار برد:

- توقف بعد از تکرار به دفعات معین
- توقف وقتی که خطا از یک مقدار تعیین شده کمتر شود.
- توقف وقتی که خطا در مثالهای مجموعه تائید از قاعده خاصی پیروی نماید.

اگر دفعات تکرار کم باشد خطا خواهیم داشت و اگر زیاد باشد مسئله Overfitting رخ خواهد داد.

محنی یادگیری



http://playground.tensorflow.org/

مرور الگوریتم BP

- این الگوریتم یک جستجوی gradient descent در فضای وزنها انجام می‌دهد.
- ممکن است در یک مینیمم محلی گیر بیافتد
- در عمل بسیار موثر بوده است

برای پرهیز از مینیمم محلی روشهای مختلفی وجود دارد:

- افزودن ممنتم
- استفاده از stochastic gradient descent
- استفاده ازشبکه های مختلف با مقادیر متفاوتی برای وزنهای اولیه

افزودن ممنتم

- می‌توان قانون تغییر وزنها را طوری در نظر گرفت که تغییر وزن در تکرار n ام تا حدی به اندازه تغییروزن در تکرار قبلی بستگی داشته باشد.

$$\Delta W_{ji}(n) = \eta\, \delta_j\, X_{ji} + \alpha \Delta W_{ji}(n-1)$$

قانون تغییر وزن — عبارت ممنتم

که در آن مقدارممنتم α بصورت $0 <= \alpha <= 1$ میباشد.

افزودن ممنتم باعث میشود تا با حرکت در مسیر قبلی در سطح خطا:

- از گیر افتادن در مینیم محلی پرهیز شود
- از قرارگرفتن در سطوح صاف پرهیز شود
- با افزایش تدریجی مقدار پله تغییرات، سرعت جستجو افزایش یابد.

قدرت نمایش توابع

- گرچه قدرت نمایش توابع به توسط یک شبکه feedforward بسته به عمق و گستردگی شبکه دارد، با این وجود موارد زیر را می‌توان به صورت قوانین کلی بیان نمود:
- توابع بولی :هر تابع بولی را می‌توان توسط یک شبکه دو لایه پیاده سازی نمود.
- توابع پیوسته :هر تابع پیوسته محدود را می‌توان توسط یک شبکه دو لایه تقریب زد .تئوری مربوطه در مورد شبکه هائی که از تابع سیگموئید در لایه پنهان و لایه خطی در شبکه خروجی استفاده می‌کنند صادق است.
- توابع دلخواه :هر تابع دلخواه را می‌توان با یک شبکه سه لایه تا حد قابل قبولی تفریب زد.

با این وجود باید درنظر داشت که فضای فرضیه جستجو شده توسط روش gradient deescent ممکن است در برگیرنده تمام مقادیر ممکن وزنها نباشد.

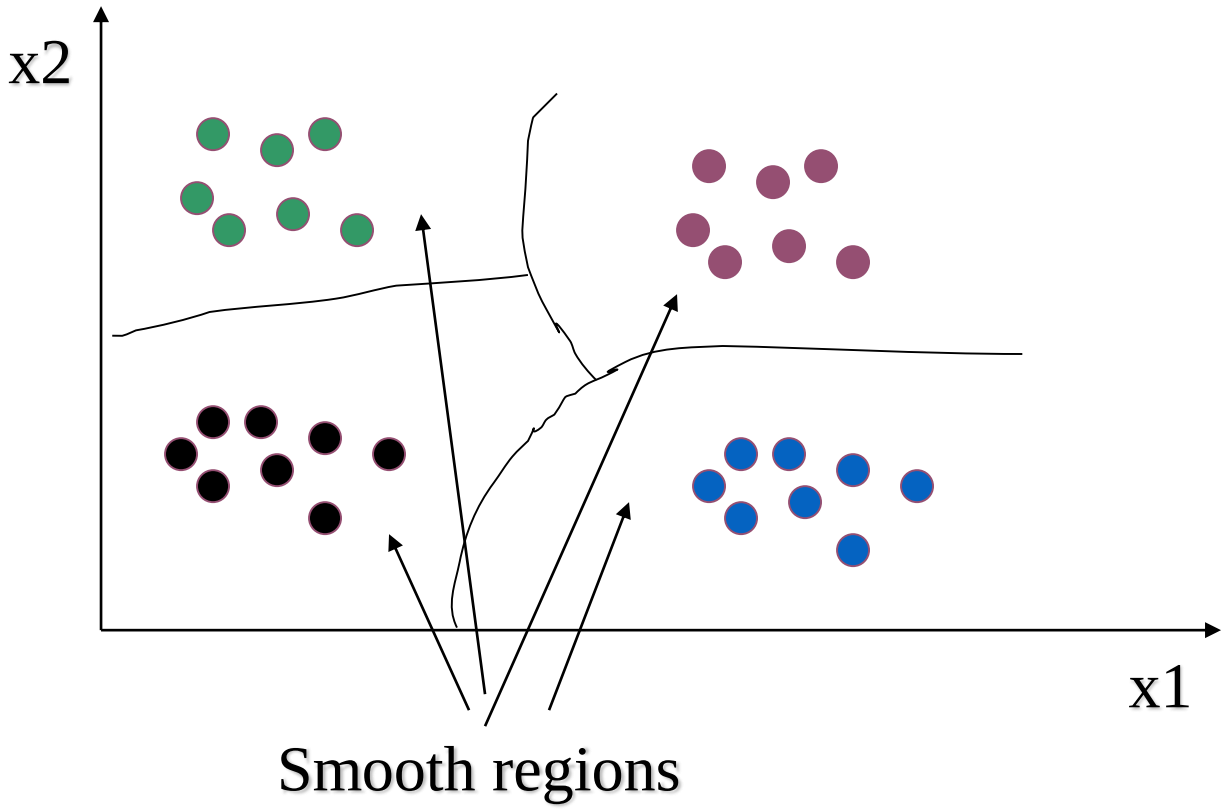
فضای فرضیه و بایاس استقرا

- فضای فرضیه مورد جستجو را می‌توان بصورت یک فضای فرضیه اقلیدسی n بعدی از وزن‌های شبکه در نظر گرفت(که nتعداد وزنهاست)
- این فضای فرضیه بر خلاف فضای فرضیه درخت تصمیم یک فضای پیوسته است.
- بایاس استقرا این روش را می‌توان بصورت زیر بیان کرد:

“smooth interpolation between data points”

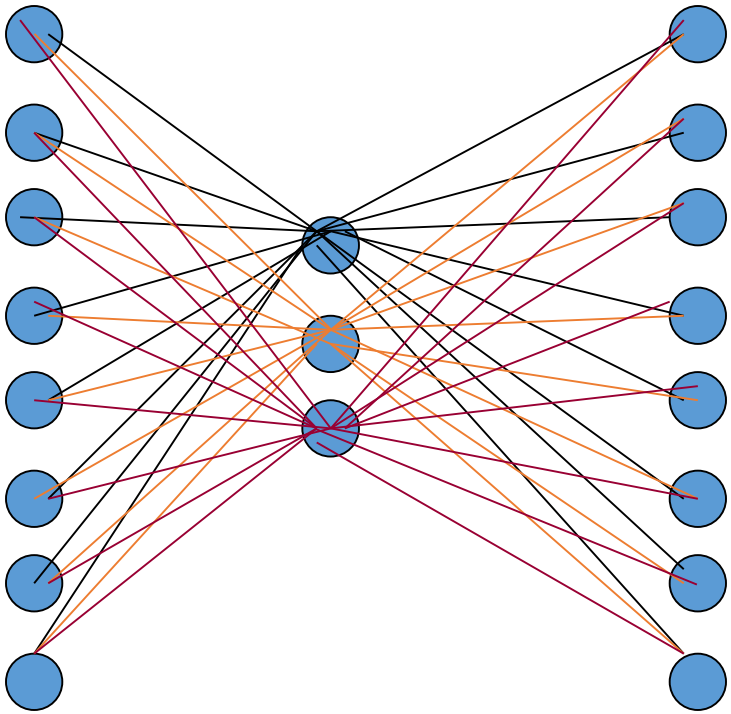
به این معنا که الگوریتم BP سعی می‌کند تا نقاطی را که به هم نزدیکتر هستند در یک دسته‌بندی قرار دهد.

مثال



قدرت نمایش لایه پنهان

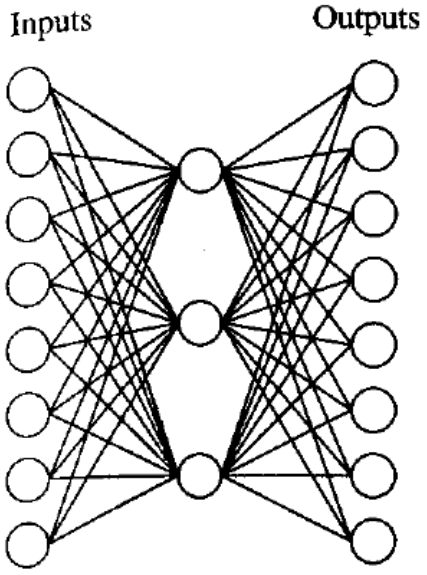
- یکی از خواص BP این است که می‌تواند در لایه های پنهان شبکه ویژگیهای نا آشکاری از داده ورودی نشان دهد.



برای مثال شبکه 8x3x8 زیر طوری آموزش داده می‌شود که مقدارهرمثال ورودی را عینا در خروجی بوجو د آورد(تابع f(x)=x را یاد بگیرد .)ساختار خاص این شبکه باعث میشود تا واحد های لایه وسط ویژگی های مقادیر ورودی را به نحوی کد بندی کنند که لایه خروجی بتواند از آنان برای نمایش مجدد داده ها استفاده نماید .

قدرت نمایش لایه پنهان

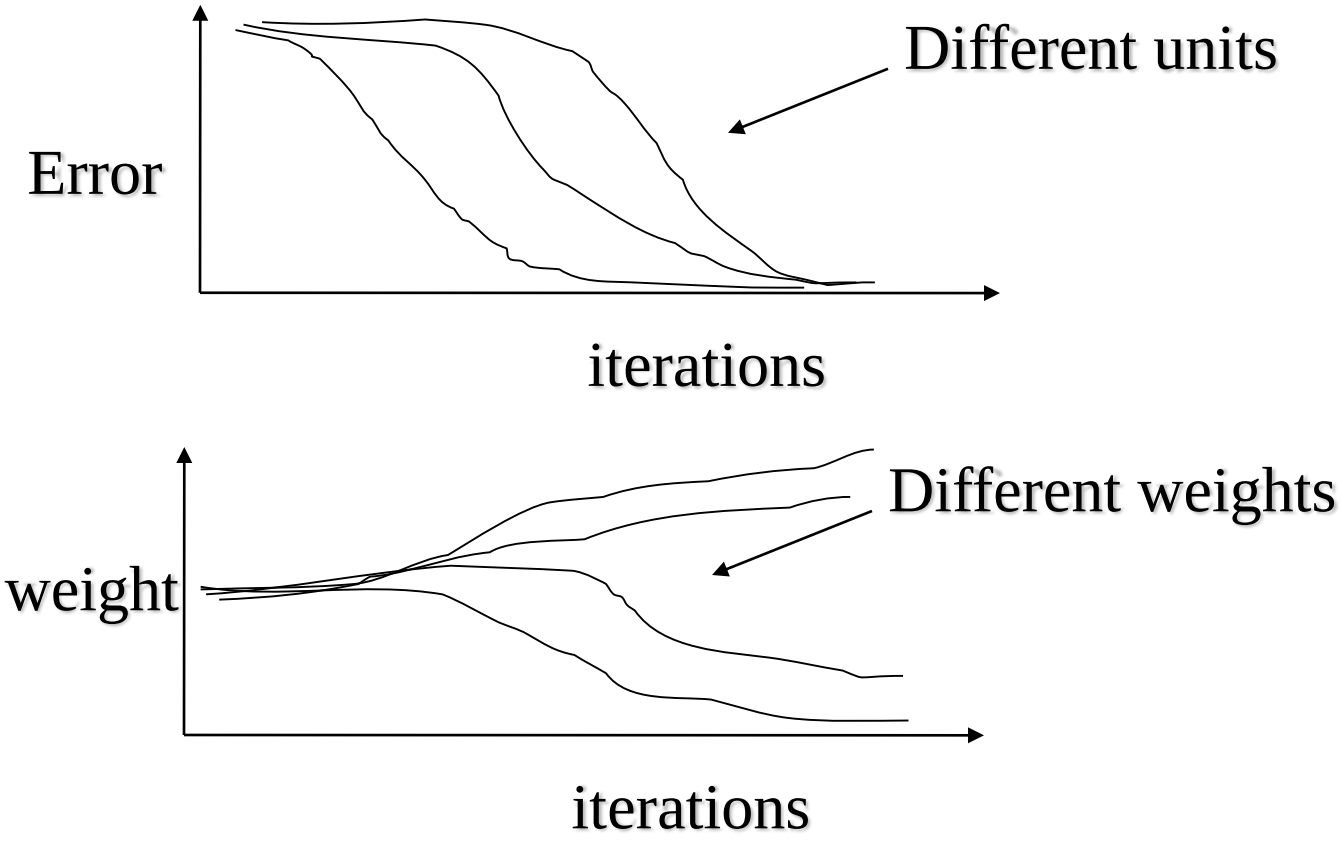
در این آزمایش که به تعداد 5000 بار تکرار شده از 8 داده مختلف به عنوان ورودی استفاده شده و شبکه با استفاده از الگوریتم BP موفق شده تا تابع هدف را بیاموزد.



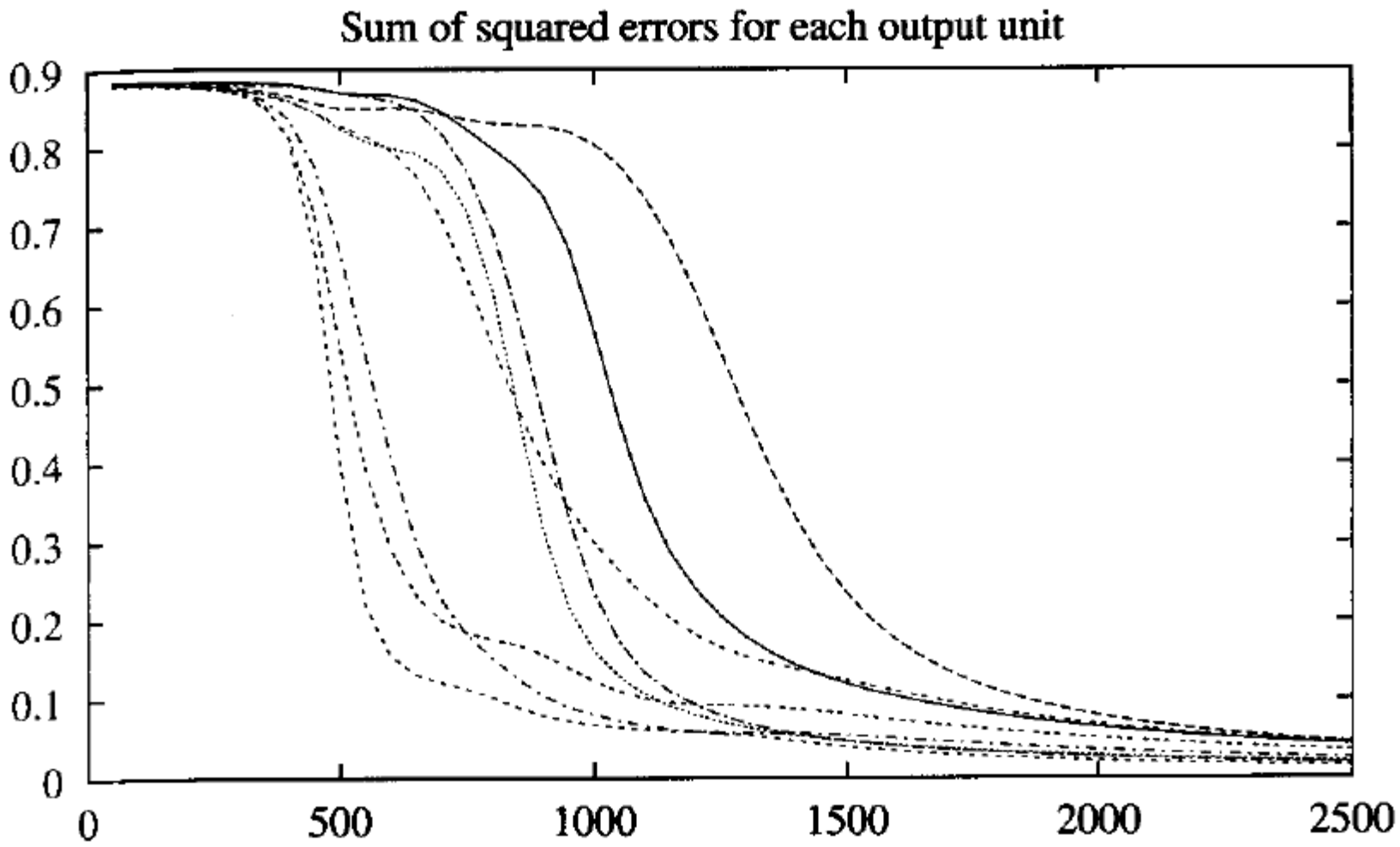
Input		Hidden Values				Output
10000000	→	.89	.04	.08	→	10000000
01000000	→	.15	.99	.99	→	01000000
00100000	→	.01	.97	.27	→	00100000
00010000	→	.99	.97	.71	→	00010000
00001000	→	.03	.05	.02	→	00001000
00000100	→	.01	.11	.88	→	00000100
00000010	→	.80	.01	.98	→	00000010
00000001	→	.60	.94	.01	→	00000001

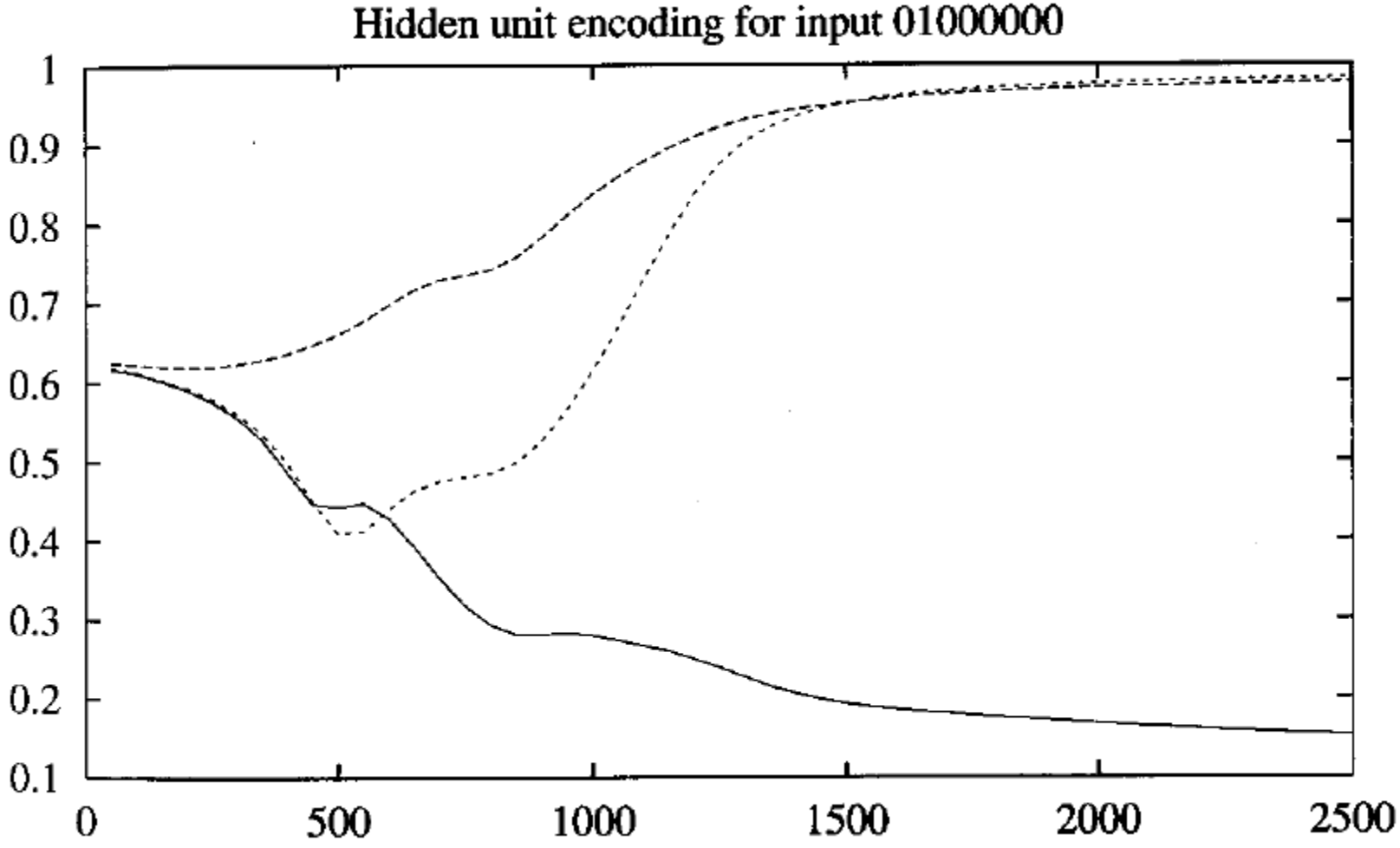
با مشاهده خروجی واحد های لایه میانی مشخص میشود که بردار حاصل معادل انکدینگ استاندارد داده‌های ورودی بوده است(000,001,,...,111)

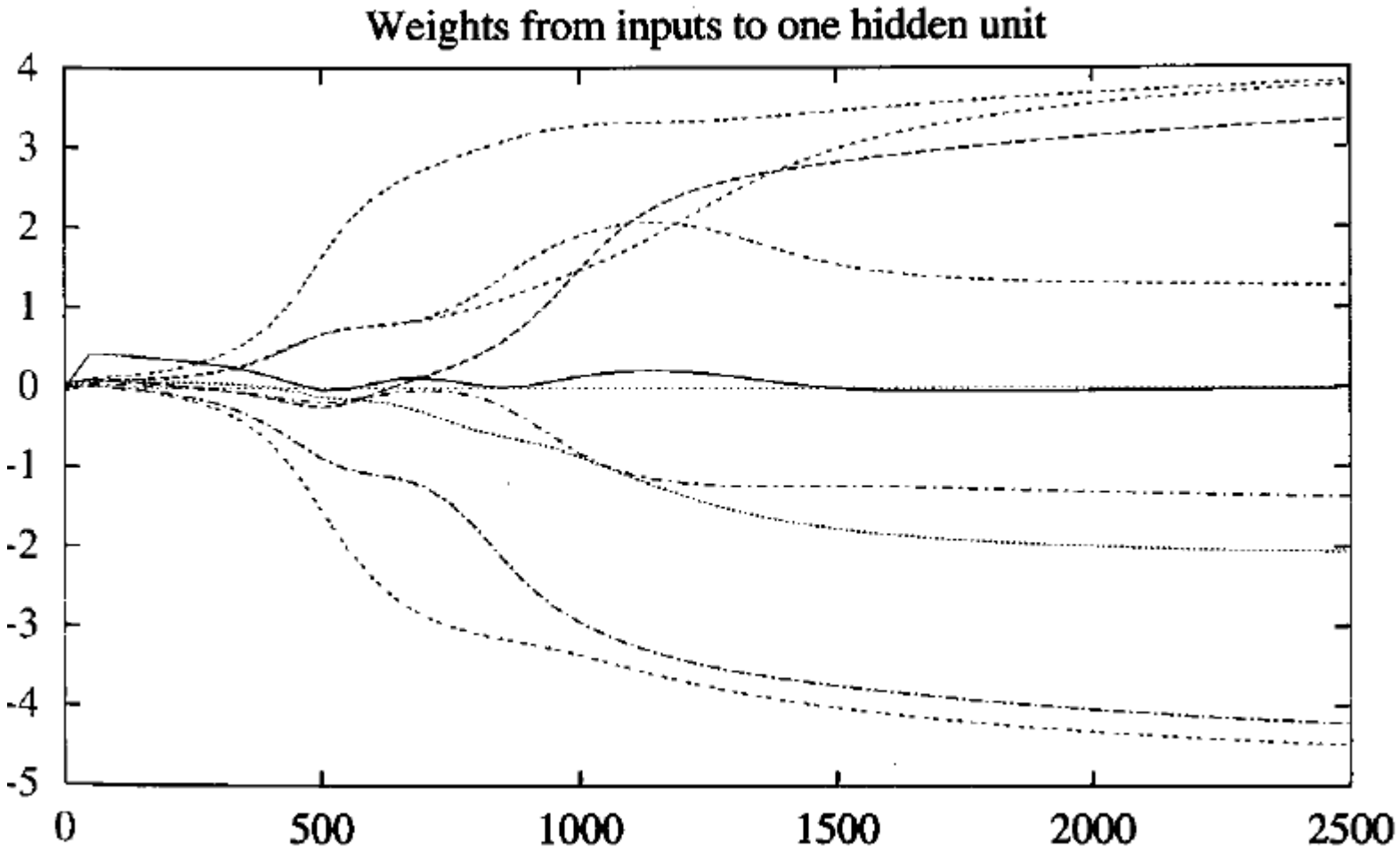
نمودار خطا


Different units
Error
iterations
Different weights
weight
iterations

نمودار خطا

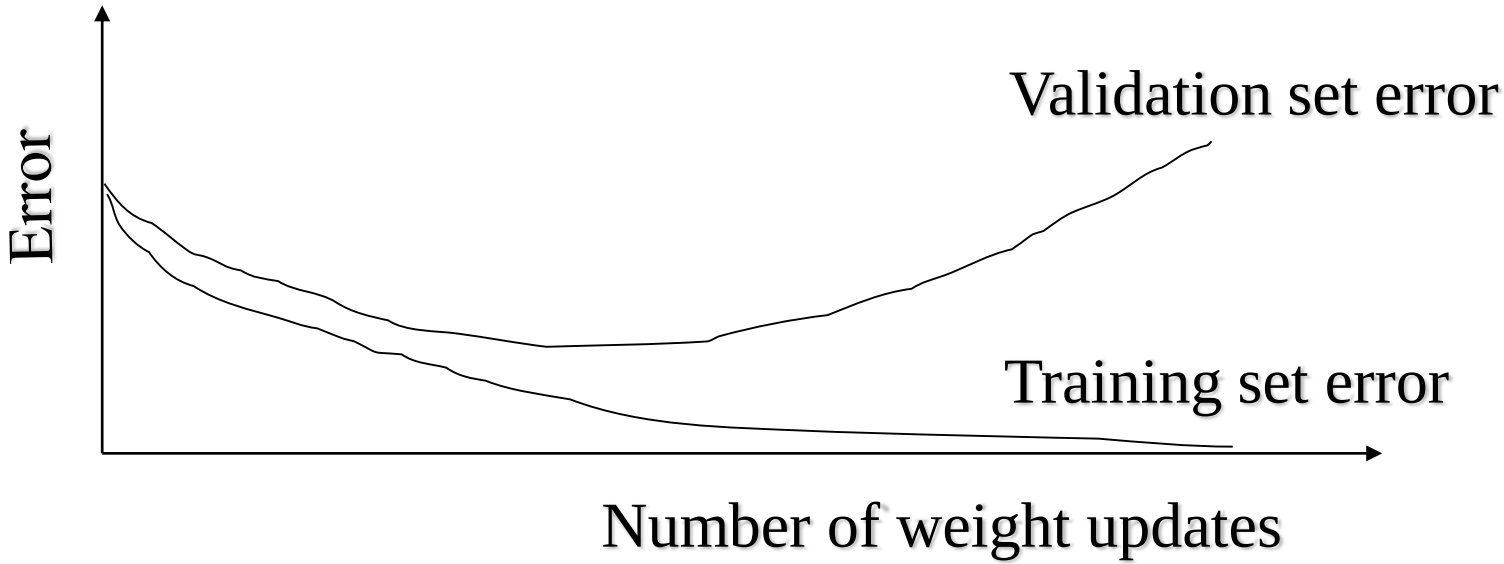

Sum of squared errors for each output unit
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0
500
1000
1500
2000
2500


Hidden unit encoding for input 01000000
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
500
1000
1500
2000
2500


Weights from inputs to one hidden unit
4
3
2
1
0
-1
-2
-3
-4
-5
0
500
1000
1500
2000
2500

قدرت تعمیم و Overfitting

- شرط پاین الگوریتم BP چیست؟
- یک انتخاب این است که الگوریتم را آنقدر ادامه دهیم تا خطا از مقدار معینی کمتر شود .این امر میتواند منجر به overfitting شود.



دلایل رخ دادن overfitting

- overfittingناشی از تنظیم وزنها برای در نظر گرفتن مثالهای نادری است که ممکن است با توزیع کلی داده ها مطابقت نداشته باشند .تعداد زیاد وزنهای یک شبکه عصبی باعث میشود تا شبکه درجه آزادی زیادی برای انطباق با این مثالها داشته باشد.
- با افزایش تعداد تکرار، پیچیدگی فضای فرضیه یادگرفته شده توسط الگوریتم بیشتر و بیشتر میشود تا شبکه بتواند نویز و مثالهای نادر موجود در مجموعه آموزش را بدرستی ارزیابی نماید.

راه حل

- استفاده از یک مجموعه تائید Vallidation و توقف یادگیری هنگامی که خطا در این مجموعه به اندازه کافی کوچک میشود.
- بایاس کردن شبکه برای فضاهای فرضیه ساده تر :یک راه می‌تواند استفاده ازweight decay باشد که در آن مقدار وزنها در هر بارتکرار باندازه خیلی کمی کاهش داده میشود.
- k-fold cross validation وقتی که تعداد مثالهای آموزشی کم باشد میتوان m داده آموزشی را به K دسته تقسیم بندی نموده و آزمایش را به تعداد k دفعه تکرار نمود .در هر دفعه یکی از دسته ها بعنوان مجموعه تست و بقیه بعنوان مجموعه آموزشی استفاده خواهند شد .تصمیم گیری بر اساس میانگین نتایج انجام میشود.

روش‌های دیگر

راه های بسیار متنوعی برای ایجاد شبکه های جدید وجود دارد از جمله:

- استفاده از تعاریف دیگری برای تابع خطا
- استفاده از روشهای دیگری برای کاهش خطا در حین یادگیری
  - Hybrid Global Learning
  - Simulated Annealing
  - Genetic Algorithms
- استفاده از توابع دیگری در واحدها
  - Radial Basis Functions
- استفاده از ساختار های دیگری برای شبکه
  - Recurrent Network

روش‌های دیگر

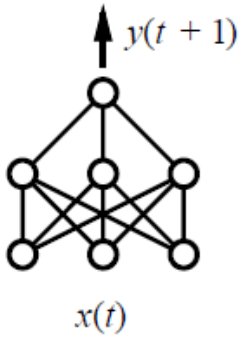
Penalize large weights:

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} (t_{kd} - o_{kd})^2 + \gamma \sum_{i,j} w_{ji}^2$$

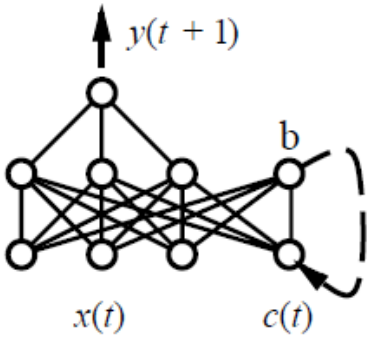
Train on target slopes as well as values:

$$E(\vec{w}) \equiv \frac{1}{2} \sum_{d \in D} \sum_{k \in outputs} \left[(t_{kd} - o_{kd})^2 + \mu \sum_{j \in inputs} \left(\frac{\partial t_{kd}}{\partial x_d^j} - \frac{\partial o_{kd}}{\partial x_d^j} \right)^2 \right]$$

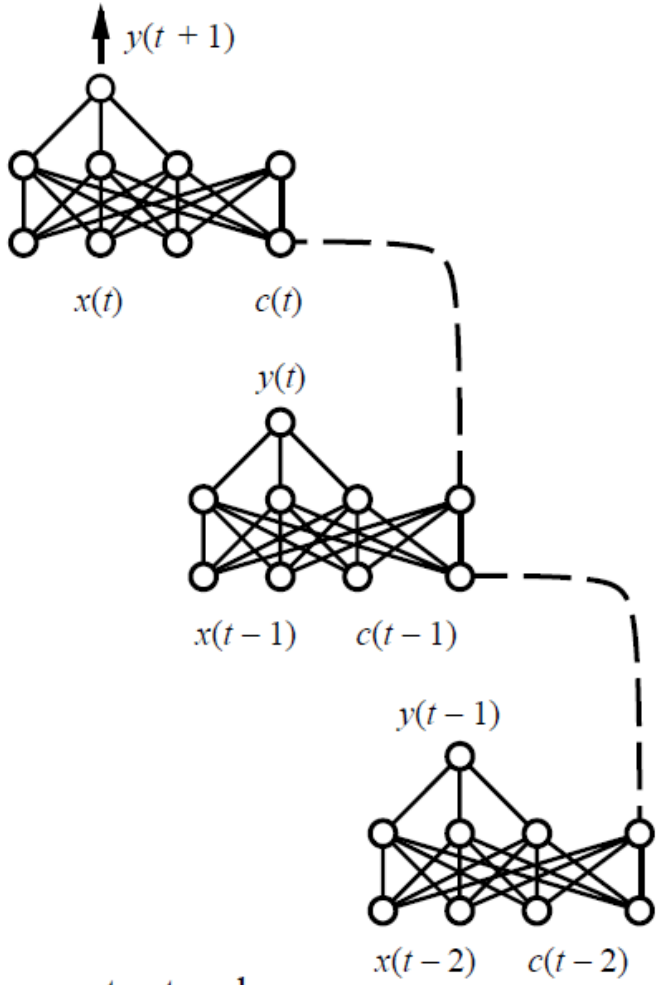
Recurrent Network



(a) Feedforward network

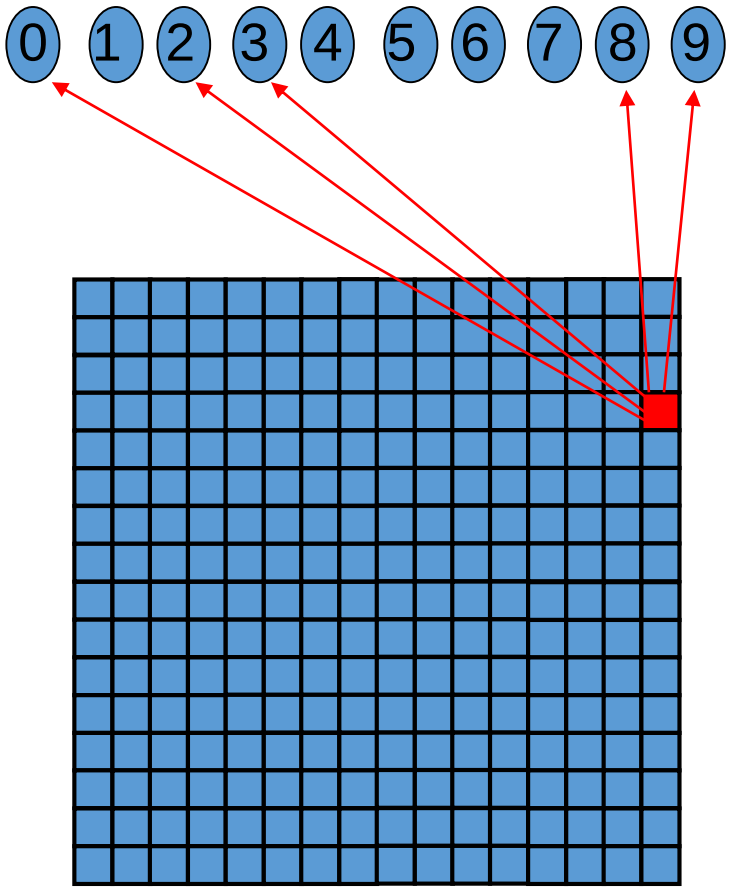


(b) Recurrent network



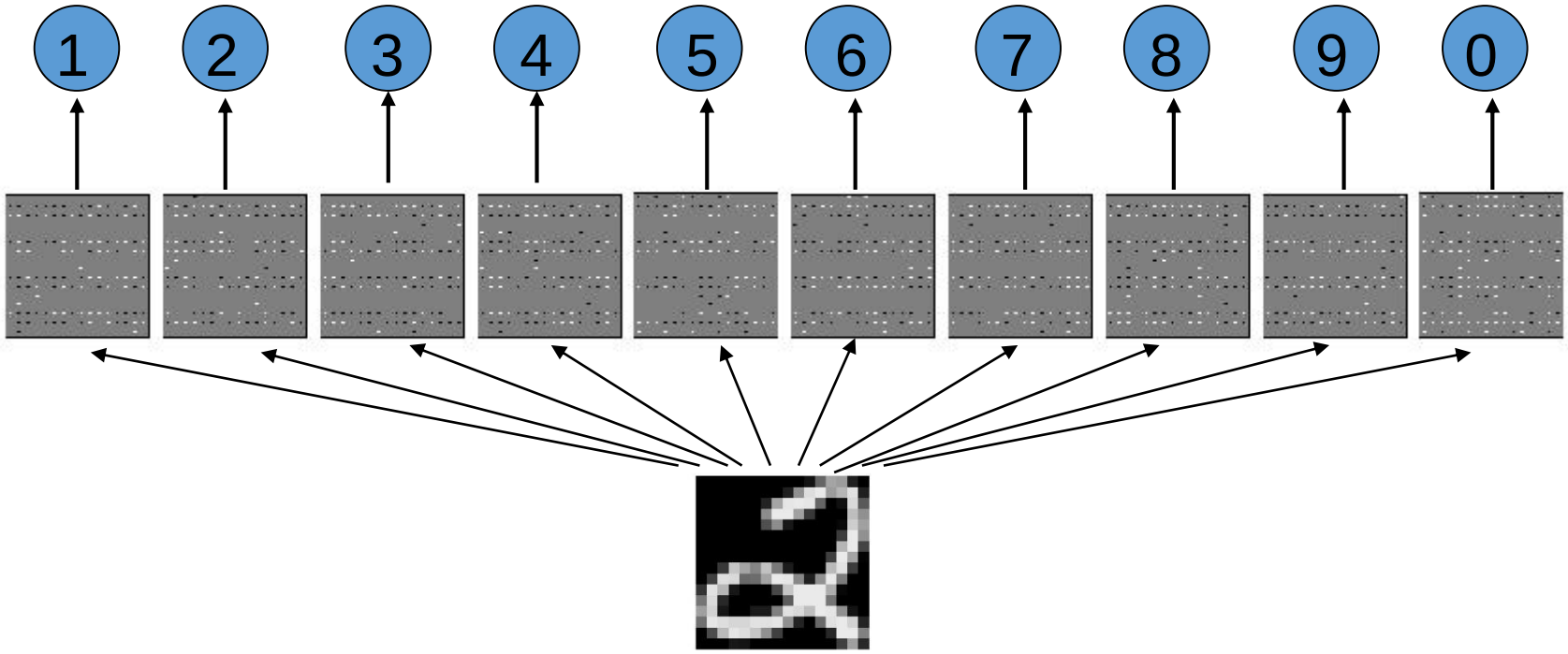
(c) Recurrent network unfolded in time

مثال: تشخیص ارقام



- فرض کنید بخواهیم با استفاده از یک شبکه دو لایه ارقام دستنویس را تشخیص دهیم.
- نرونهای لایه اول شدت روشنائی پیکسل‌ها را تقریب می‌زنند و
- نرونهای لایه آخر شکل ارقام را تعیین می کنند.

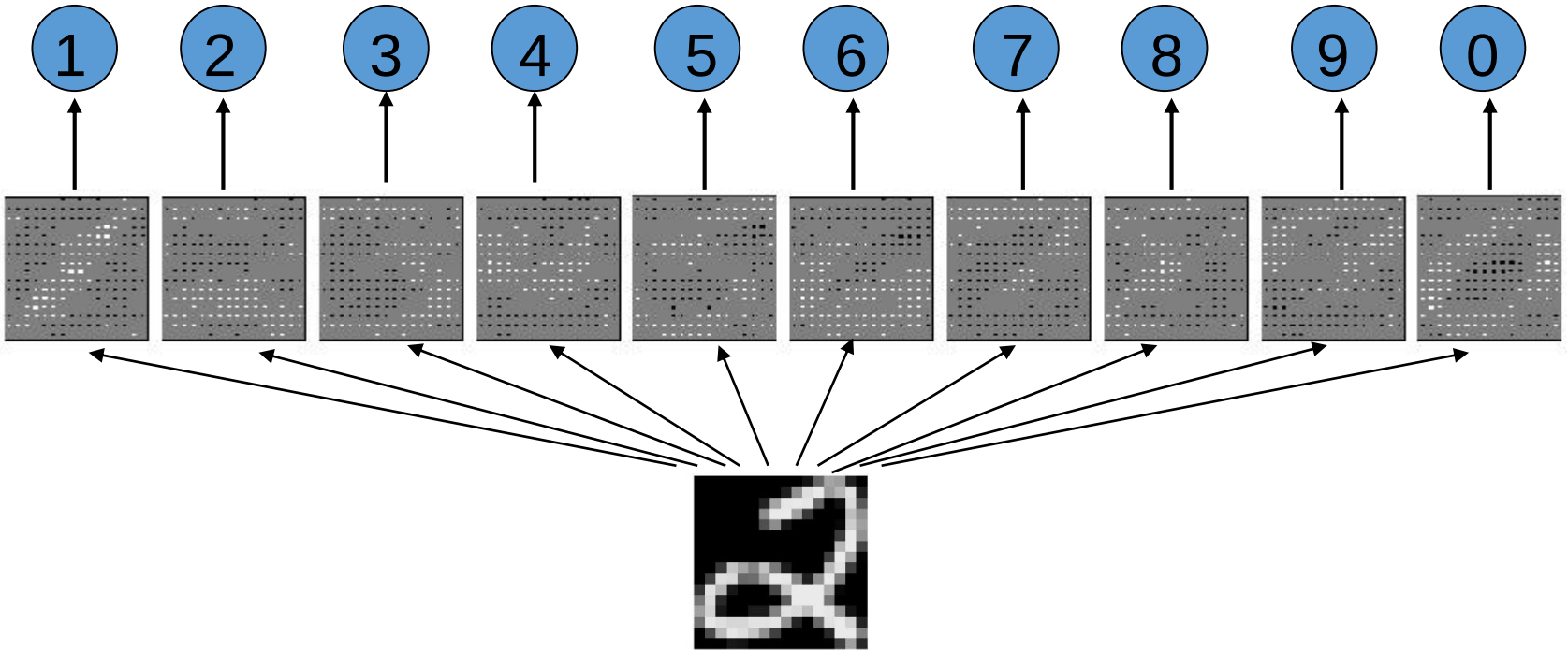
روشی که وزن‌ها یاد گرفته می‌شوند



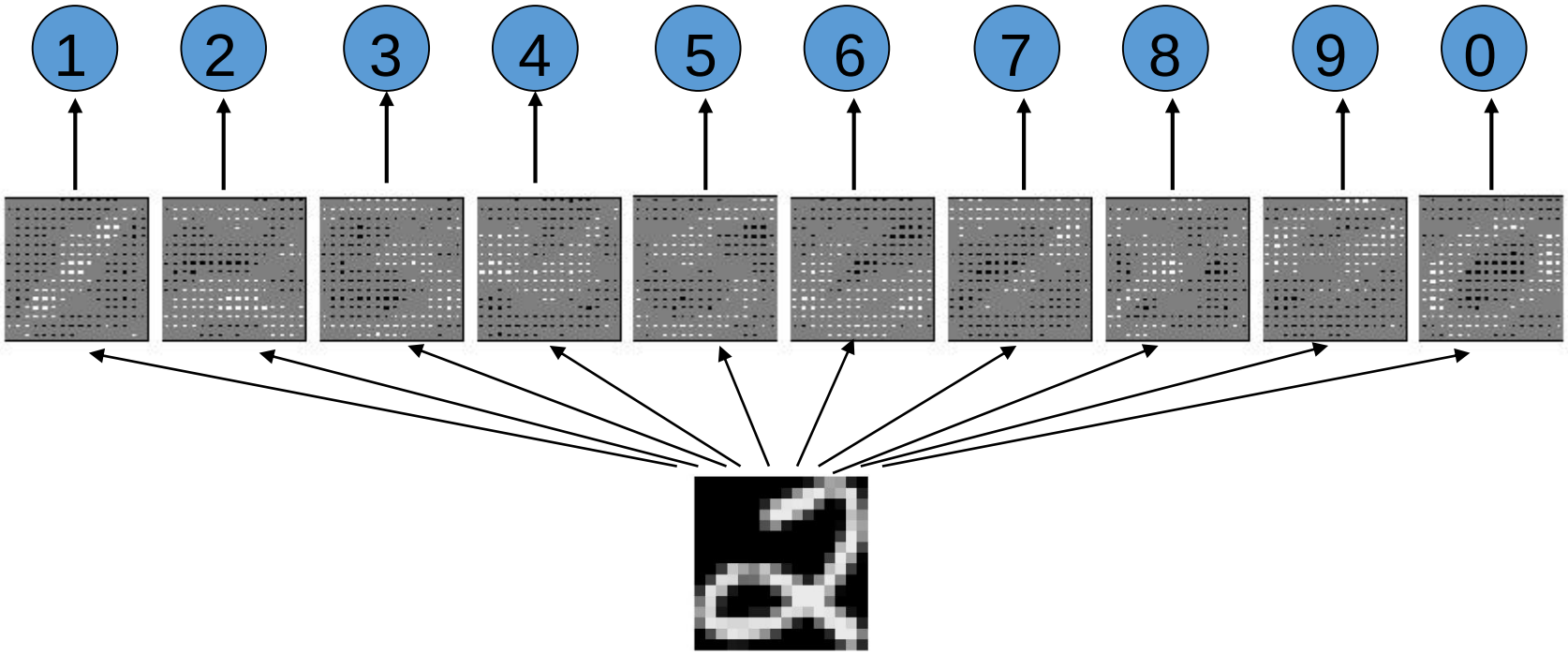
تصویر ورودی

تصویر به شبکه ارائه شده و وزنهای پیکسل‌های فعال بتدریج اضافه می‌شوند. وزن پیکسل‌های غیر موثر نیز بتدریج کاهش می‌یابد.

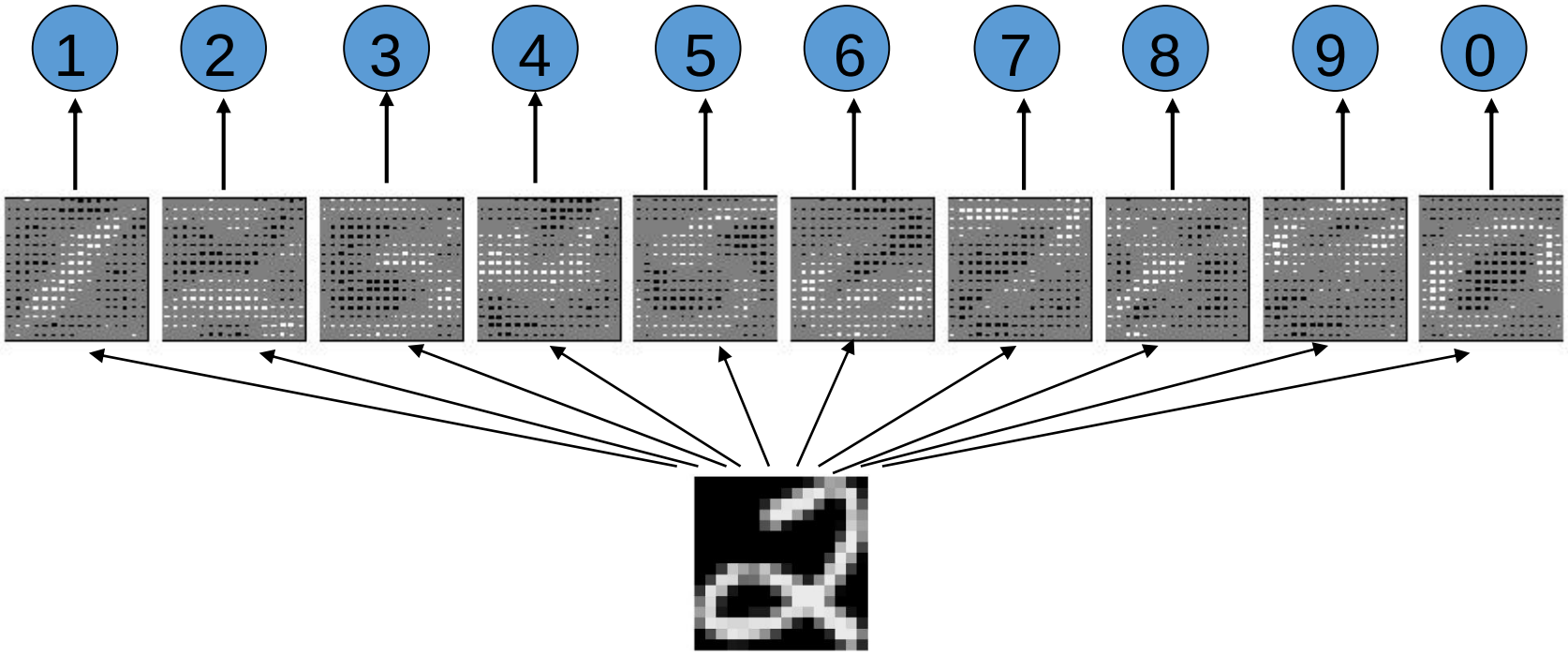
شکل گیری وزن‌ها



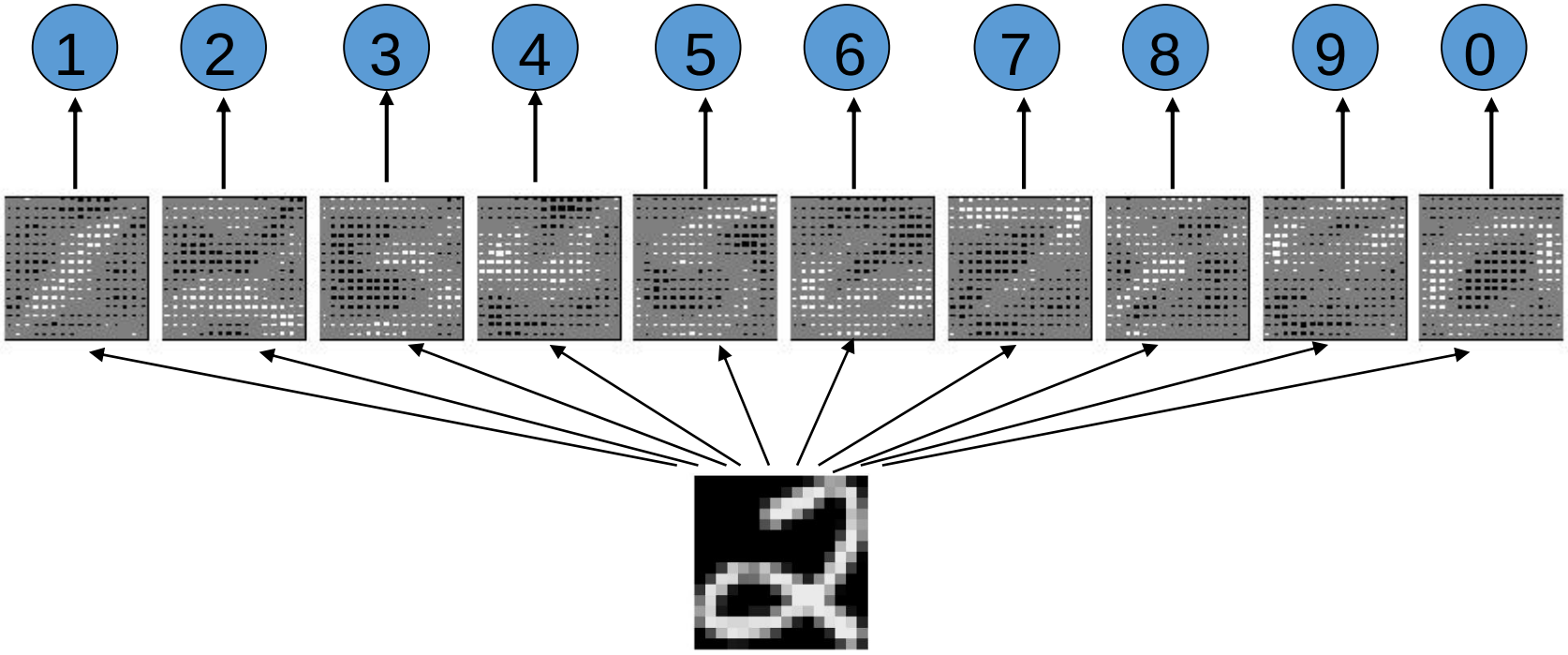
تصویر ورودی



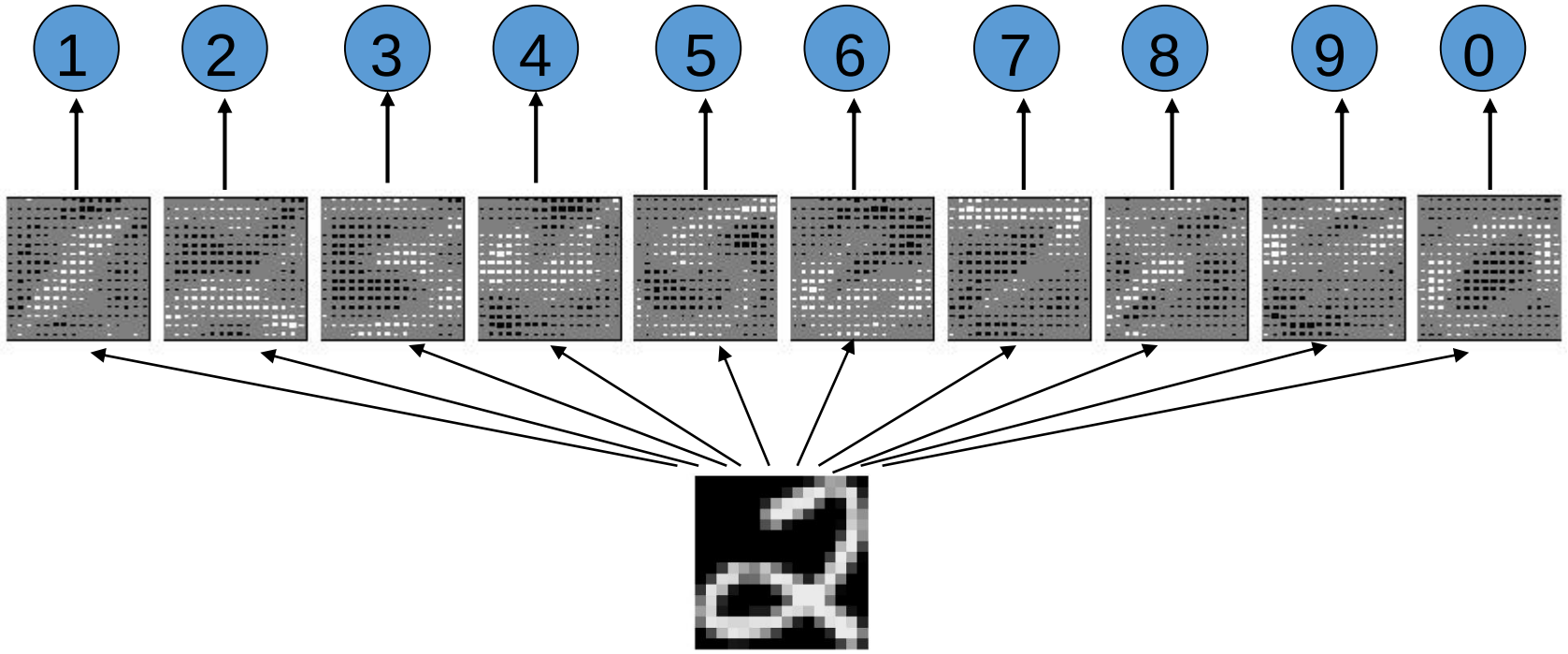
تصویر ورودی



تصویر ورودی

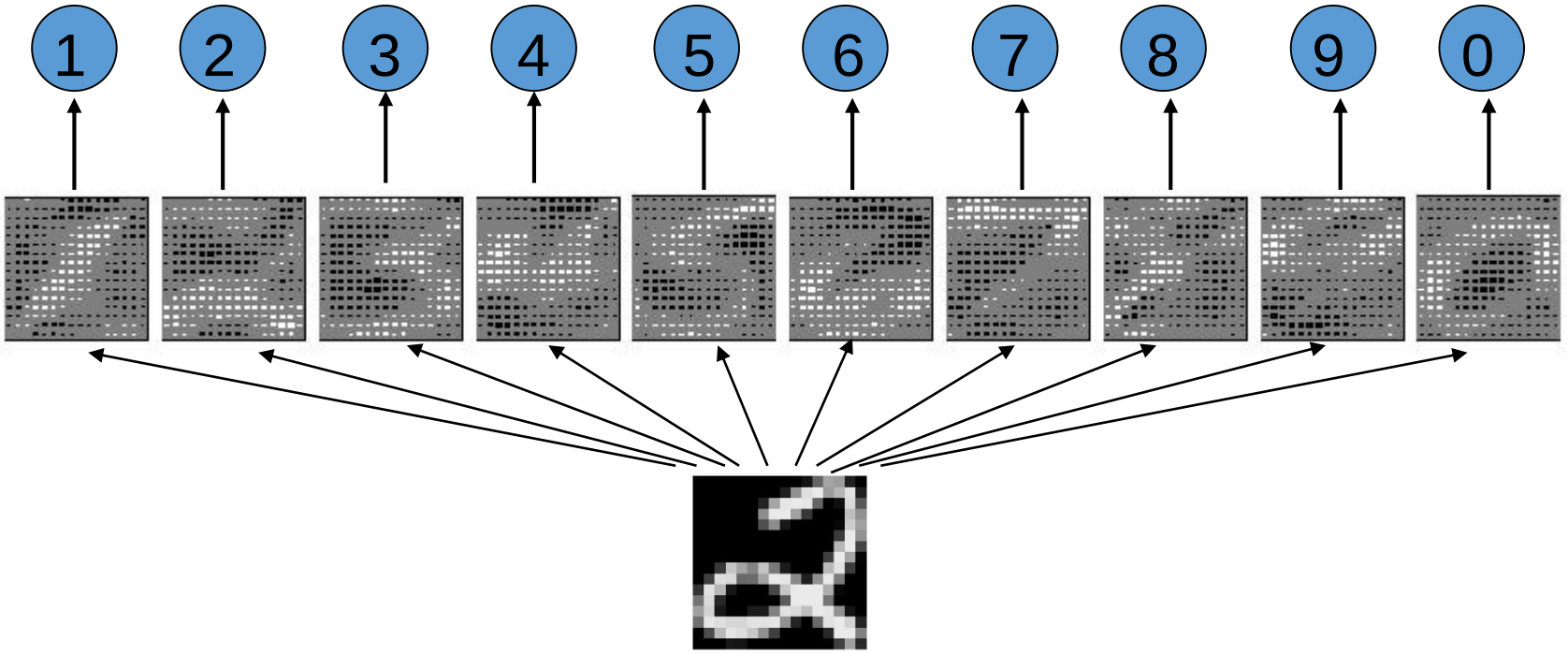


تصویر ورودی



تصویر ورودی

The learned weights



تصویر ورودی

شبکه چه چیزی را یاد میگیرد؟

- در این مثال یک شبکه با دو لایه معادل با استفاده از یک سری template یا قالب است که شبکه قالبی را که بهترین تطبیق با ورودی را داشته باشد بر میگزینند!
- اما برای مسئله ارقام دستنویس شکلهای ورودی بسیار متنوع هستند لذا یک قالب ساده که با همه ورودیها سازگار باشد وجود ندارد. در نتیجه چنین شبکه ای هم نمیتواند راه حل مسئله در حالت کلی باشد!
- برای اینکه بتوان مسئله را در حالت کلی حل نمود بایدشکل های ورودی به مجموعه ای از ویژگی ها تبدیل شده و شبکه را بر اساس ویژگی ها آموزش داد.

مثالی از تنوع ارقام دست‌نویس