دانشگاه آزاد اسلامی واحد تبریز

نام درس: یادگیری ماشین

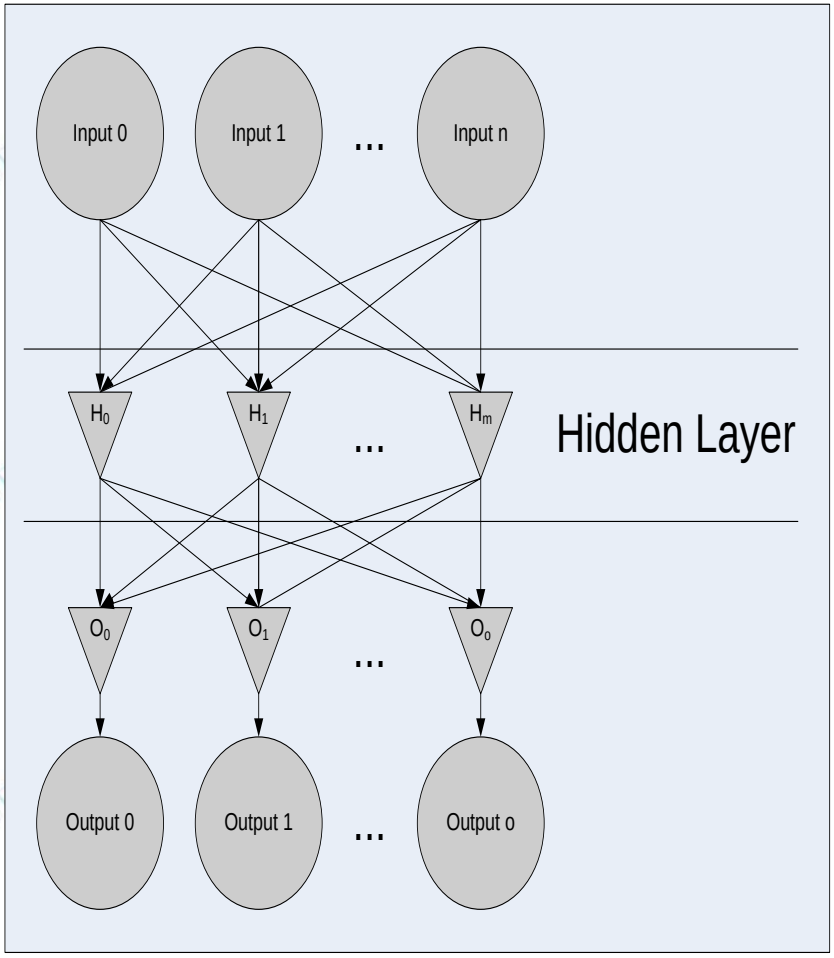
بخش: شبکه های عصبی مصنوعی

نام استاد: دکتر مسعود کارگر

مقدمه

- 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 داده‌های آموزشی مقاوم بوده و اینگونه شبکه‌ها با موفقیت به مسایلی نظیر شناسایی گفتار، شناسایی و تعبیر تصاویر، و یادگیری روبات اعمال شده است.

شبکه عصبی چیست؟



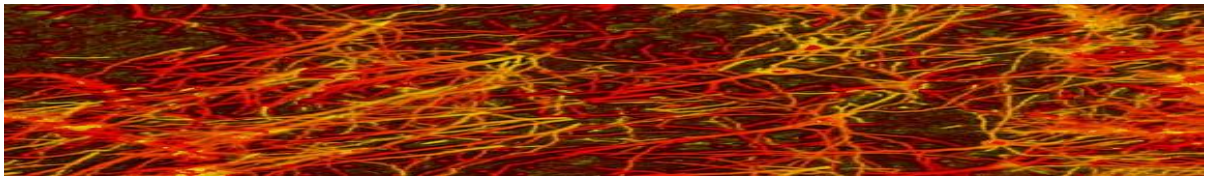
- روشی برای محاسبه است که بر پایه اتصال به هم پیوسته چندین واحد پردازشی ساخته می‌شود.
- شبکه از تعداد دلخواهی سلول یا گره یا واحد یا نرون تشکیل می‌شود که مجموعه ورودی را به خروجی ربط می‌دهند.

شبکه عصبی چه قابلیتهایی دارد؟

- محاسبه یک تابع معلوم
- تقریب یک تابع ناشناخته
- شناسایی الگو
- پردازش سیگنال
- یادگیری انجام موارد فوق

الهام از طبیعت

- مطالعه شبکه‌های عصبی مصنوعی تا حد زیادی ملهم از سیستم‌های یادگیر طبیعی است که در آنها یک مجموعه پیچیده از نرونهای به هم متصل در کار یادگیری دخیل هستند.
- 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 سوییچنگ 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 بازشناسایی نماید. این قدرت فوق العاده باید از پردازش موازی توزیع شده در تعدادی زیادی از نرونها حاصل شده باشد.

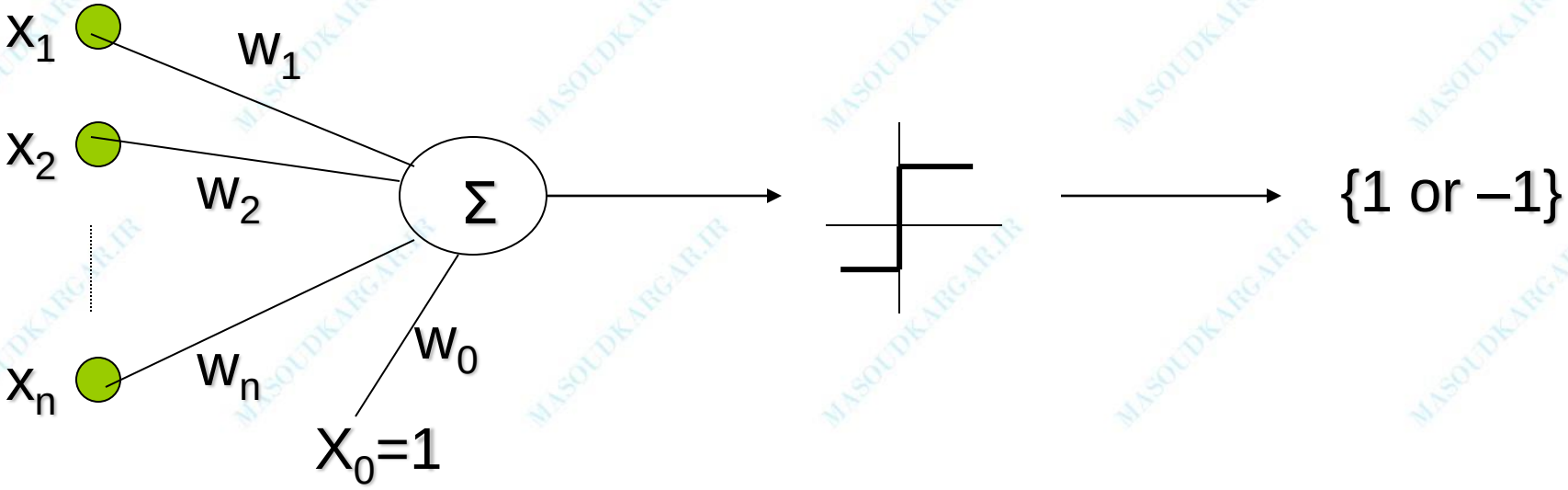


مسایل مناسب برای یادگیری شبکه‌های عصبی

- خطا در داده‌های آموزشی وجود داشته باشد. مثل مسایلی که داده‌های آموزشی دارای نویز حاصل از داده‌های سنسورها نظیر دوربین و میکروفن‌ها هستند.
- مواردی که نمونه‌ها توسط مقادیر زیادی زوج ویژگی-مقدار نشان داده شده باشند. نظیر داده‌های حاصل از یک دوربین ویدیویی.
- 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 یاد گرفته شده توسط شبکه را تعبیر نمود.

Perceptron

- نوعی از شبکه عصبی بر مبنای 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.



یادگیری یک پرسپترون

- خروجی پرسپترون توسط رابطه زیر مشخص می‌شود:

$$O(x_1,x_2,\ldots,x_n) = \begin{cases} 1 \text{ if } w_0 + w_1x_1 + w_2x_2 + \ldots + w_nx_n > 0 \\ -1 \text{ otherwise} \end{cases}$$

- که برای سادگی آنرا می‌توان به صورت زیر نشان داد:

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

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

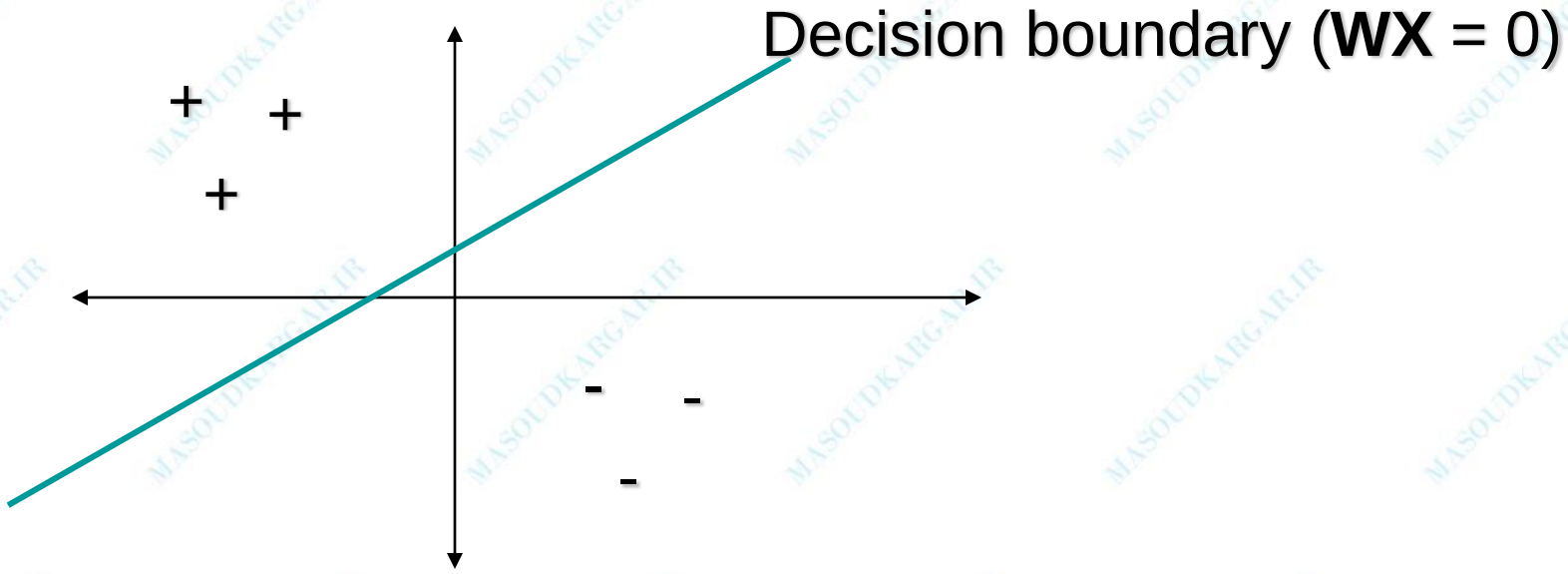
یادگیری پرسپترون عبارت است از:

پیدا کردن مقادیر درستی برای W

بنابراین فضای فرضیه H در یادگیری پرسپترون عبارت است ازمجموعه تمام مقادیر حقیقی ممکن برای بردارهای وزن.

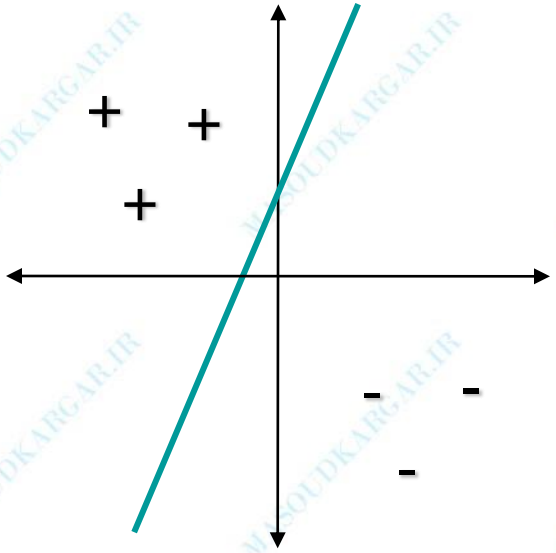
توانایی پرسپترون

- پرسپترون را می‌توان به صورت یک سطح تصمیم hyper plane در فضای nبعدی نمونه‌ها در نظر گرفت. پرسپترون برای نمونه‌های یک طرف صفحه مقدار 1 و برای مقادیر طرف دیگر مقدار 1- بوجود می‌آورد.

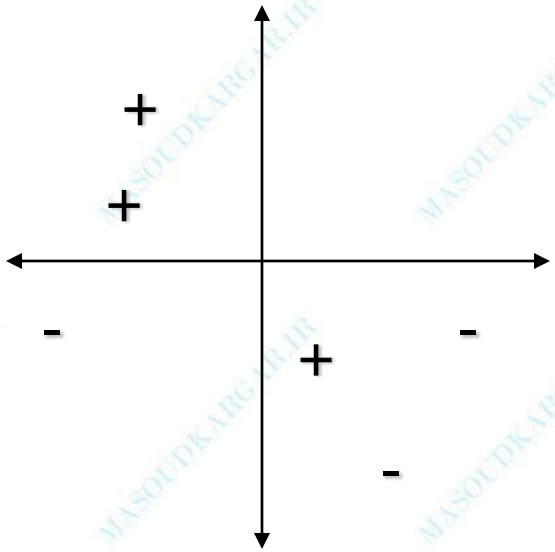


توابعی که پرسپترون قادر به یادگیری آنها می‌باشد

- یک پرسپترون فقط قادر است مثالهایی را یاد بگیرد که به صورت خطی جداپذیر باشند. اینگونه مثالها مواردی هستند که بطور کامل توسط یک hyper plane قابل جداسازی می‌باشند.



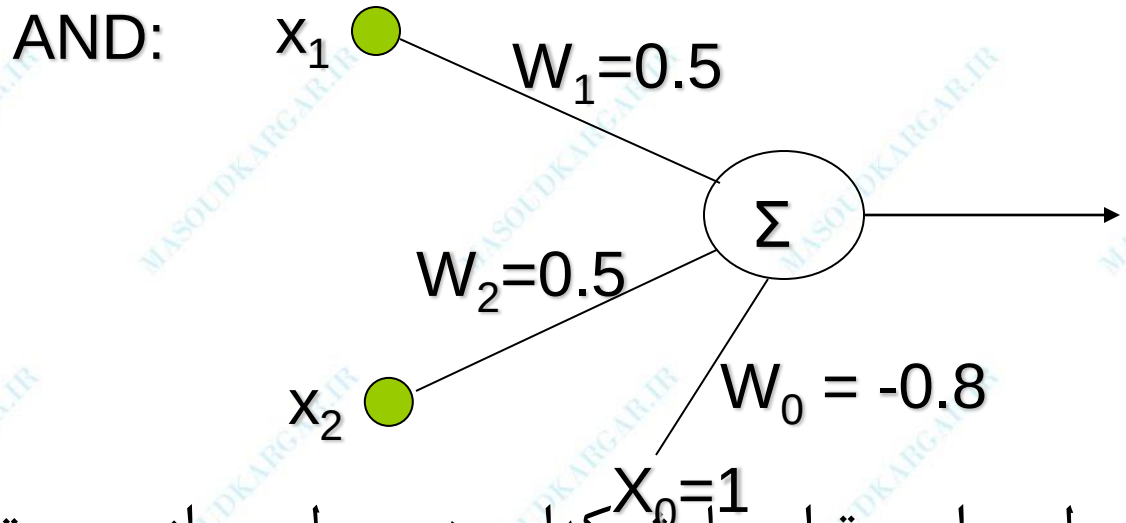
Linearly separable



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.



- در واقع هر تابع بولی را می‌توان با شبکه‌ای دو سطحی از پرسپترونها نشان داد.

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:
  - قاعده پرسپترون
  - قاعده 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 ۲ برمی‌گردیم

قاعده پرسپترون

- برای یک مثال آموزشی $X = (x_1, x_2, \ldots, x_n)$ در هر مرحله وزنها بر اساس قاعده پرسپترون به صورت زیر تغییر می‌کند:

$$w_i = w_i + \Delta w_i$$

که در آن

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

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 جداپذیر خطی این روش همگرا شده و پرسپترون قادر به جداسازی صحیح مثالها خواهد شد.

Delta Rule قاعده دلتا

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

Delta Rule قاعده دلتا

- برای درک بهتر این روش آنرا به یک پرسپترون فاقد حد آستانه اعمال می‌کنیم. یعنی داریم:

$$O(\vec{x}) = \vec{w}.\vec{x}$$

- ابتدا لازم است تعریفی برای خطای آموزش ارایه شود. یک تعریف متداول این چنین است:

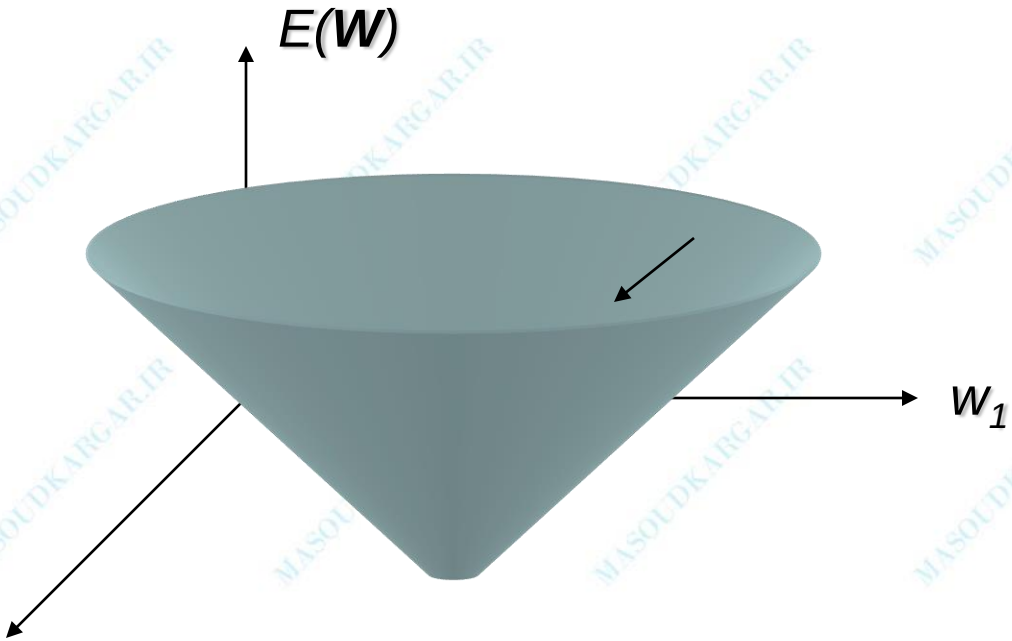
$$E(\vec{w}) \equiv \frac{1}{2}\sum_{d \in D}(t_d - o_d)^2$$

$$D = Training\ Set$$

- که این مجموع برای تمام مثالهای آموزشی انجام می‌شود.

الگوریتم Gradient descent

- با توجه به نحوه تعریف E سطح خطا به صورت یک سهمی خواهد بود. ما بدنبال وزنهایی 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 قاعده Gradient descent

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

$$\nabla E(\vec{w}) \equiv \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \ldots, \frac{\partial E}{\partial w_n} \right]$$

- که در آن $\nabla E(\vec{w})$ یک بردار و $\frac{\partial E}{\partial w_i}$ مشتق جزیی نسبت به هر وزن می‌باشد.

قاعده دلتا Delta Rule

- برای یک مثال آموزشی $X = (x_1, x_2, \ldots, x_n)$ در هر مرحله وزنها بر اساس قاعده دلتا به صورت زیر تغییر می‌کند:

$$w_i \leftarrow w_i + \Delta w_i$$

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$

η: learning rate (e.g., 0.1)

علامت منفی نشاندهنده حرکت در جهت کاهش شیب است.

محاسبه گرادیان

- با مشتق‌گیری جزیی از رابطه خطا می‌توان بسادگی گرادیان را محاسبه نمود:

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

- لذا وزنها طبق رابطه زیر تغییر خواهند نمود.

$$\Delta w_i = \eta \sum_{d \in D} (t_d - O_d) x_{id}$$

خلاصه یادگیری قاعده دلتا

الگوریتم یادگیری با استفاده از قاعده دلتا به صورت زیر می‌باشد.

1. به وزنها مقدار تصادفی کوچک نسبت دهید.
2. تا رسیدن به شرایط توقف (کوچک شدن خطا) مراحل زیر را ادامه دهید:
   - هر وزن Δw_i را با مقدار صفر مقداردهی اولیه کنید.
   - برای همه نمونه‌ها (x_i, t) در مجموعه آموزشی وزن Δw_i را به صورت زیر تغییر دهید:

$$\Delta w_i = \Delta w_i + \eta(t - o)x_i$$

   - برای هر w_i آنرا به صورت زیر تغییر دهید:

$$w_i = w_i + \Delta w_i$$

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 کند.

در ضمن این روش وقتی قابل استفاده است که:

- فضای فرضیه دارای فرضیه‌های پارامتریک پیوسته باشد.
- رابطه خطا قابل مشتق‌گیری باشد.

تقریب افزایشی Gradient descent

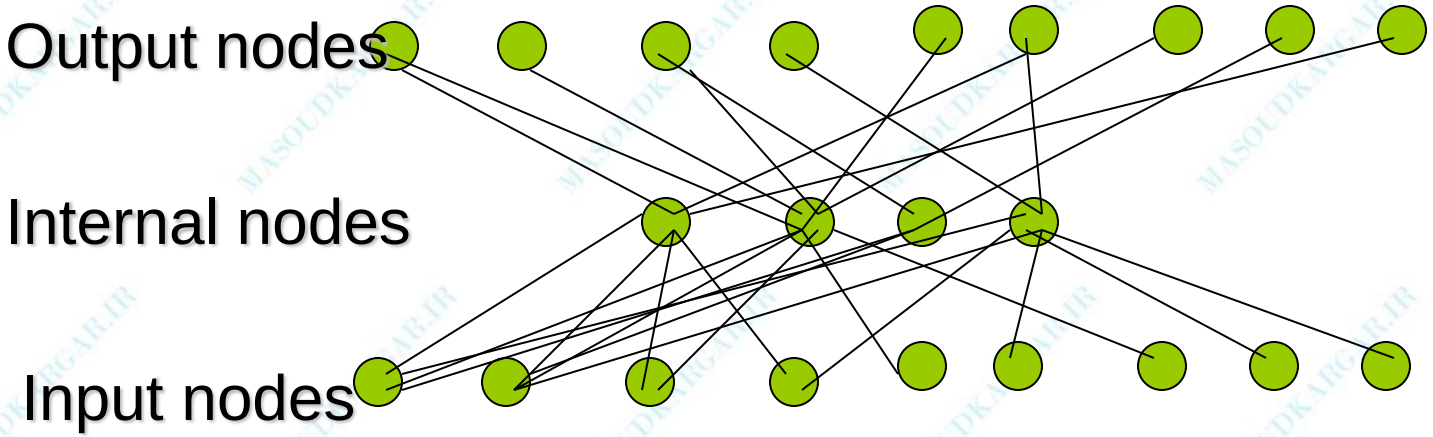
- می‌توان بجای تغییر وزنها پس از مشاهده همه مثالها، آنها را به ازای هر مثال مشاهده شده تغییر داد. در این حالت وزنها به صورت افزایشی *(Incremental)* تغییر می‌کنند. این روش را *Stochastic gradient descent* نیز می‌نامند.

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

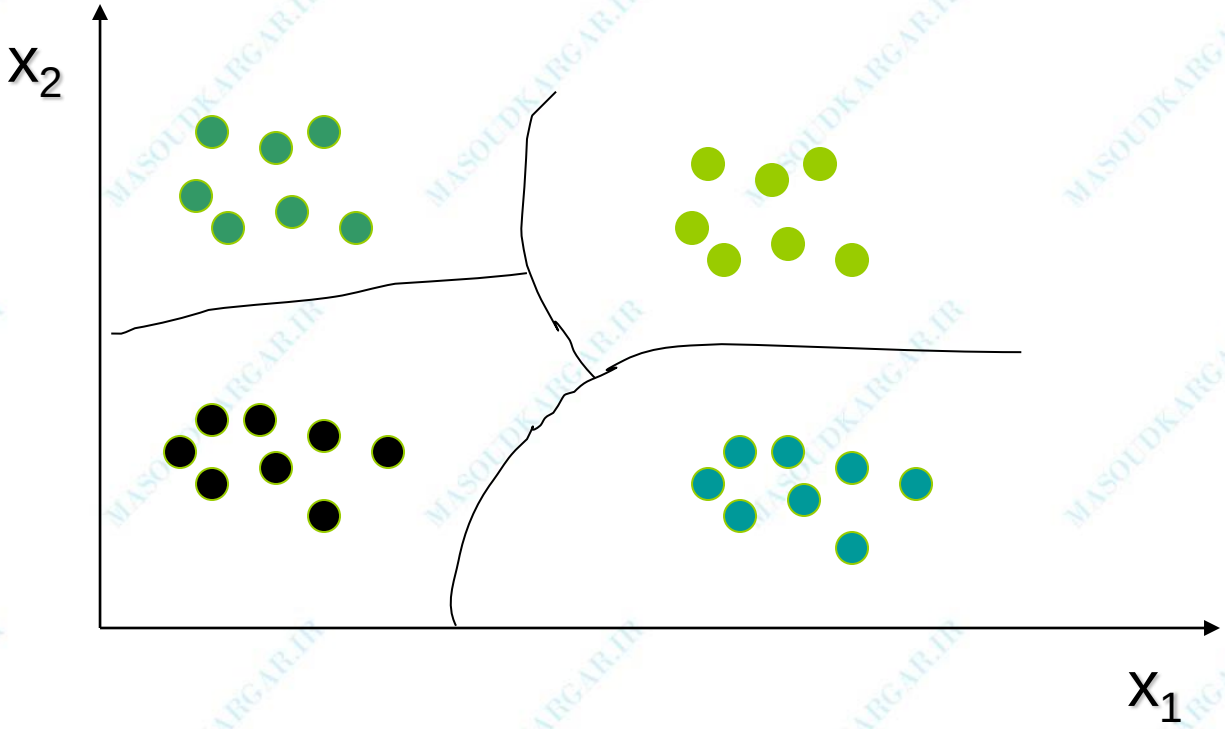
- در بعضی موارد تغییر افزایشی وزنها می‌تواند از بروز مینیمم محلی جلوگیری کند. روش استاندارد نیاز به محاسبات بیشتری دارد درعوض می‌تواند طول *step* بزرگتری هم داشته باشد.

لایه چند شبکه‌های

بر خلاف پرسپترونها شبکه‌های چند لایه می‌توانند برای یادگیری مسایل غیرخطی و همچنین مسایلی با تصمیم‌گیری‌های متعدد بکار روند.

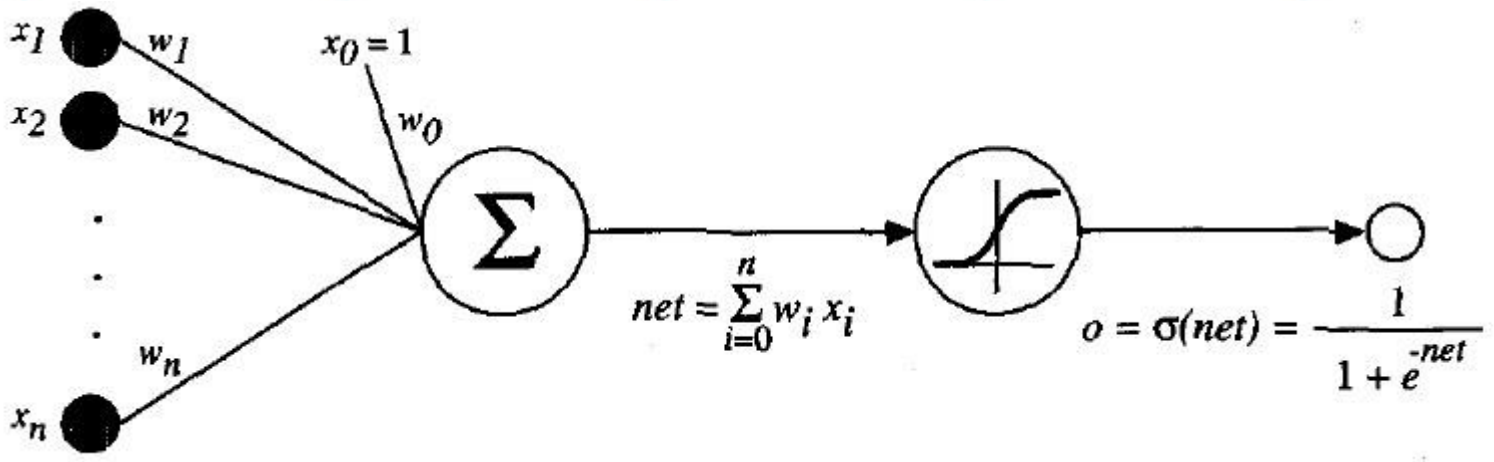


مثال



یک سلول واحد

برای اینکه بتوانیم فضای تصمیم‌گیری را به صورت غیرخطی از هم جدا بکنیم، لازم است تا هر سلول واحد را به صورت یک تابع غیرخطی تعریف نماییم. مثالی از چنین سلولی می‌تواند یک واحد سیگموید باشد:



تابع سیگموید

خروجی این سلول واحد را به صورت زیر می‌توان بیان نمود:

$$o = \sigma(\vec{w} \cdot \vec{x})$$

$$\sigma(y) = \frac{1}{1 + e^{-y}}$$

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

$$\frac{d\sigma(y)}{dy} = \sigma(y) \cdot (1 - \sigma(y))$$

الگوریتم Back propagation

- برای یادگیری وزن‌های یک شبکه چند لایه از روش *Back Propagation* استفاده می‌شود. در این روش با استفاده از *Gradient descent* سعی می‌شود تا مربع خطای بین خروجی‌های شبکه و تابع هدف مینیمم شود.

- خطا به صورت زیر تعریف می‌شود:

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

مراد از *outputs* خروجیهای مجموعه واحدهای لایه خروجی و t_{kd} و o_{kd} مقدار هدف و خروجی متناظر با k امین واحد خروجی و مثال آموزشی d است.

Back propagation الگوریتم

- 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.

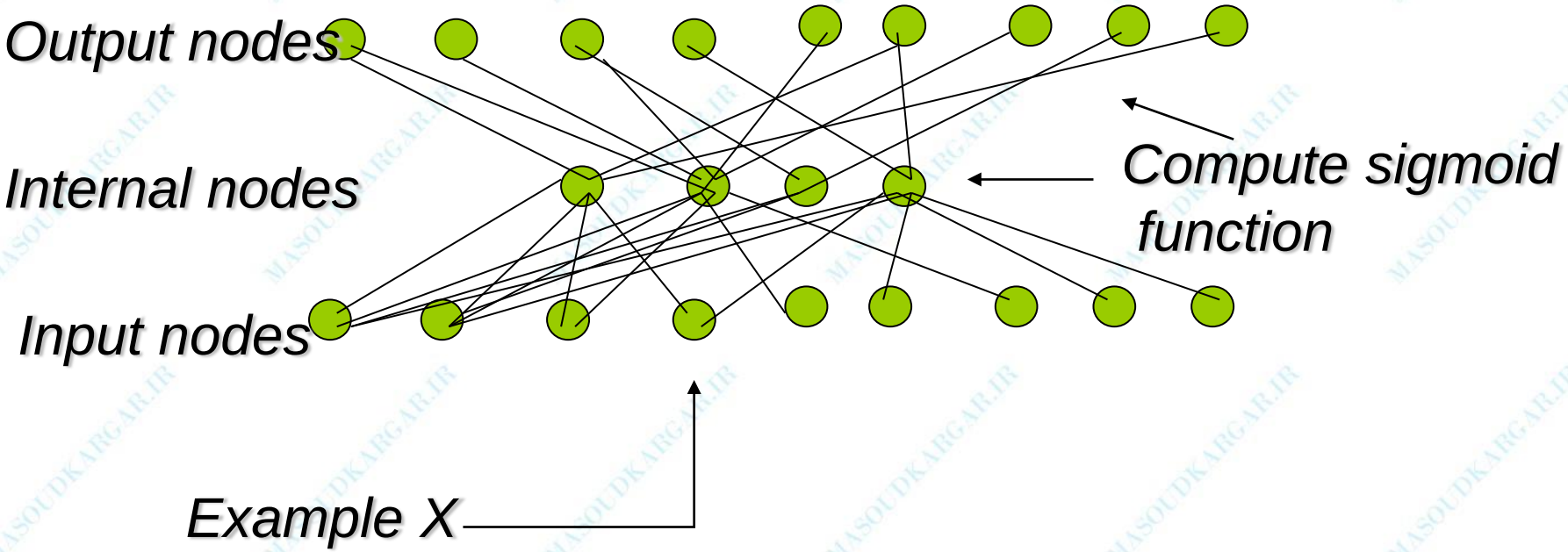
الگوریتم BP

1. شبکه‌ای با n_{in} گره در لایه ورودی، n_{hidden} گره مخفی، و n_{out} گره در لایه خروجی ایجاد کنید.
2. همه وزنها را با یک مقدار تصادفی کوچک مقداردهی کنید.
3. تا رسیدن به شرط پایانی (کوچک شدن خطا) مراحل زیر را انجام دهید:
   a. برای هر X متعلق به مثالهای آموزشی:
   b. مثال X را به سمت جلو در شبکه انتشار دهید.
   c. خطای E را به سمت عقب در شبکه انتشار دهید.

هر مثال آموزشی به صورت یک زوج (x,t) ارایه 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 گره‌های خروجی برسید.



انتشار به سمت عقب

1. برای هر واحد خروجی جمله خطا را به صورت زیر محاسبه کنید : δ_k

$$= o_k\,(1\text{-}o_k)(t_k - o_k)$$

2. برای هر واحد مخفی جمله خطا را به صورت زیر محاسبه کنید : δ_h

$$= o_h\,(1\text{-}o_h)\,\Sigma_k\, w_{kh}\,\delta_k$$

3. مقدارهر وزن را به صورت زیر تغییر دهید:

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

که در آن:

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

η = نرخ یادگیری x_{ji} = ورودی واحد i به j w_{ji} = وزن واحد i به j

نسخه *Stochastic gradient descent*.

شرط خاتمه

معمولا الگوریتم *BP* پیش از خاتمه هزاران بار با استفاده همان داده‌های 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 تایید از قاعده خاصی پیروی نماید.
- اگر دفعات تکرار کم باشد خطا خواهیم داشت و اگر زیاد باشد مساله *Over fitting*رخ خواهد داد.

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 ازشبکه‌های مختلف با مقادیر متفاوتی برای وزنهای اولیه.

افزودن ممنتم

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

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

قاعده تغییر وزن — عبارت ممنتم

که در آن مقدارممنتم α به صورت $0 <= \alpha <= 1$ 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 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 پیاده‌سازی نمود.
- توابع پیوسته :هر تابع پیوسته محدود را می‌توان توسط یک شبکه دو لایه تقریب زد. تئوری مربوطه در مورد شبکه‌هایی که از تابع سیگموید 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 descent ممکن است در برگیرنده تمام مقادیر ممکن وزنها نباشد.

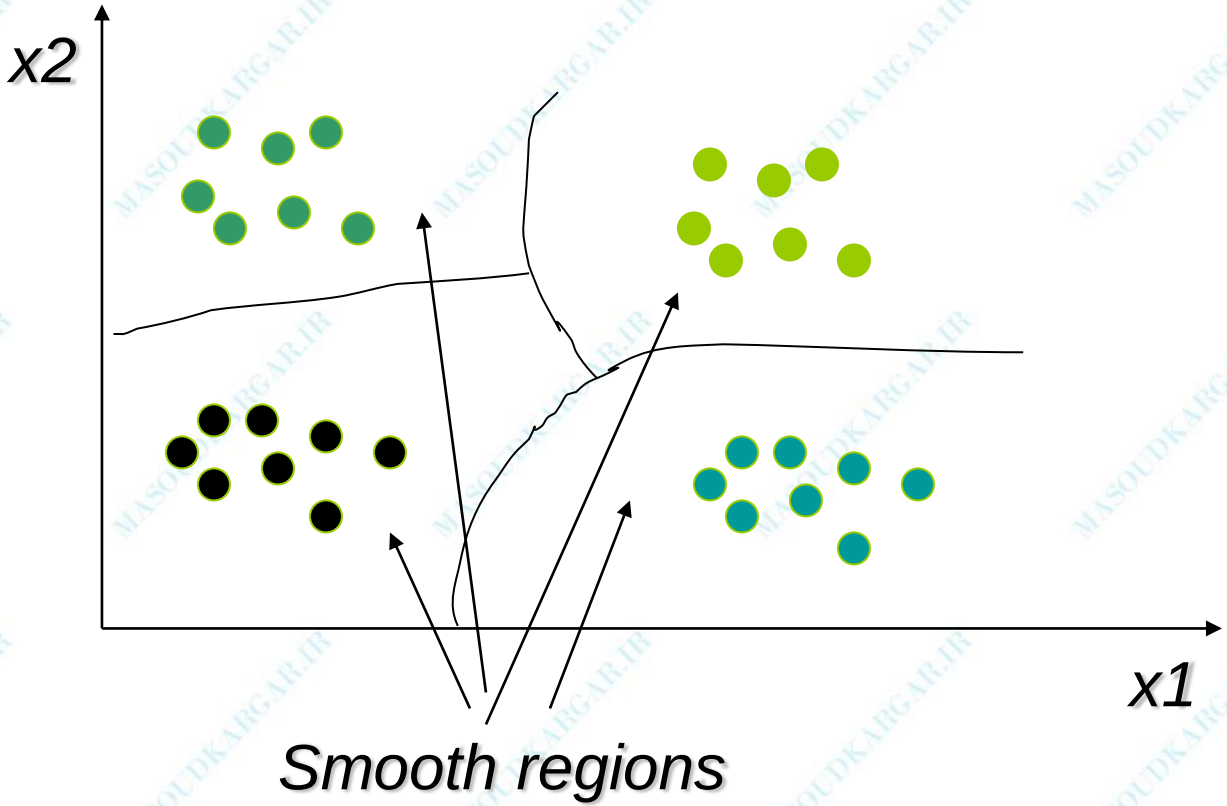
فضای فرضیه و بایاس استقرا

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

"*smooth interpolation between data points*"

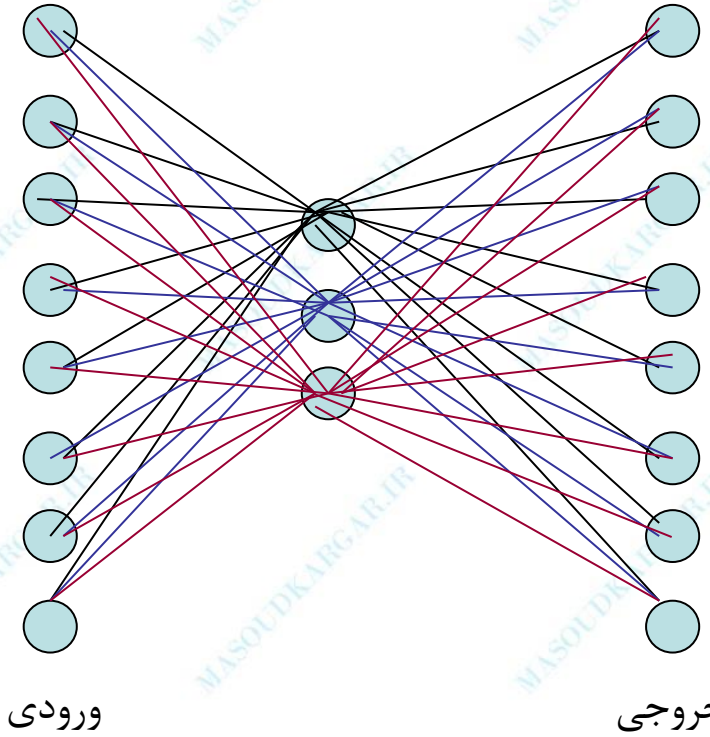
به این معنا که الگوریتم *BP نقاط درونیابی شده بین دو نقطه با کلاس معین را به همان کلاس متعلق می‌داند*. به عبارت دیگر سعی می‌کند تا نقاطی را که به هم نزدیکتر هستند در یک دسته‌بندی قرار دهد.

مثال



قدرت نمایش لایه پنهان

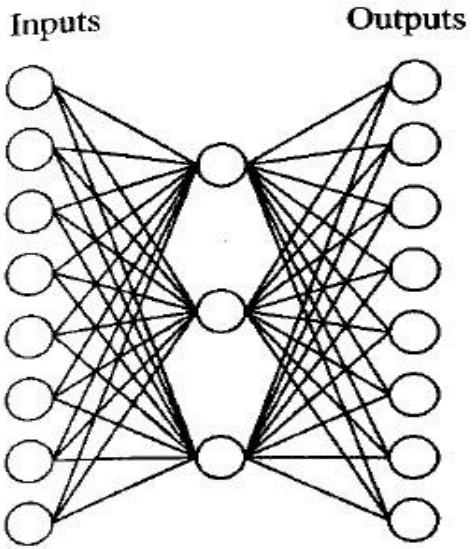
- یکی از خواص BP این است که می‌تواند در لایه‌های پنهان شبکه ویژگیهای نا آشکاری از داده ورودی نشان دهد.



برای مثال شبکه $8x3x8$ زیر طوری آموزش داده می‌شود که مقدار هر مثال ورودی را عینا در خروجی بوجود آورد(تابع $f(x)=x$ را یاد بگیرد.)ساختار خاص این شبکه باعث می‌شود تا واحدهای لایه وسط ویژگی‌های مقادیر ورودی را به نحوی کدبندی کنند که لایه خروجی بتواند از آنان برای نمایش مجدد داده‌ها استفاده نماید.

قدرت نمایش لایه پنهان

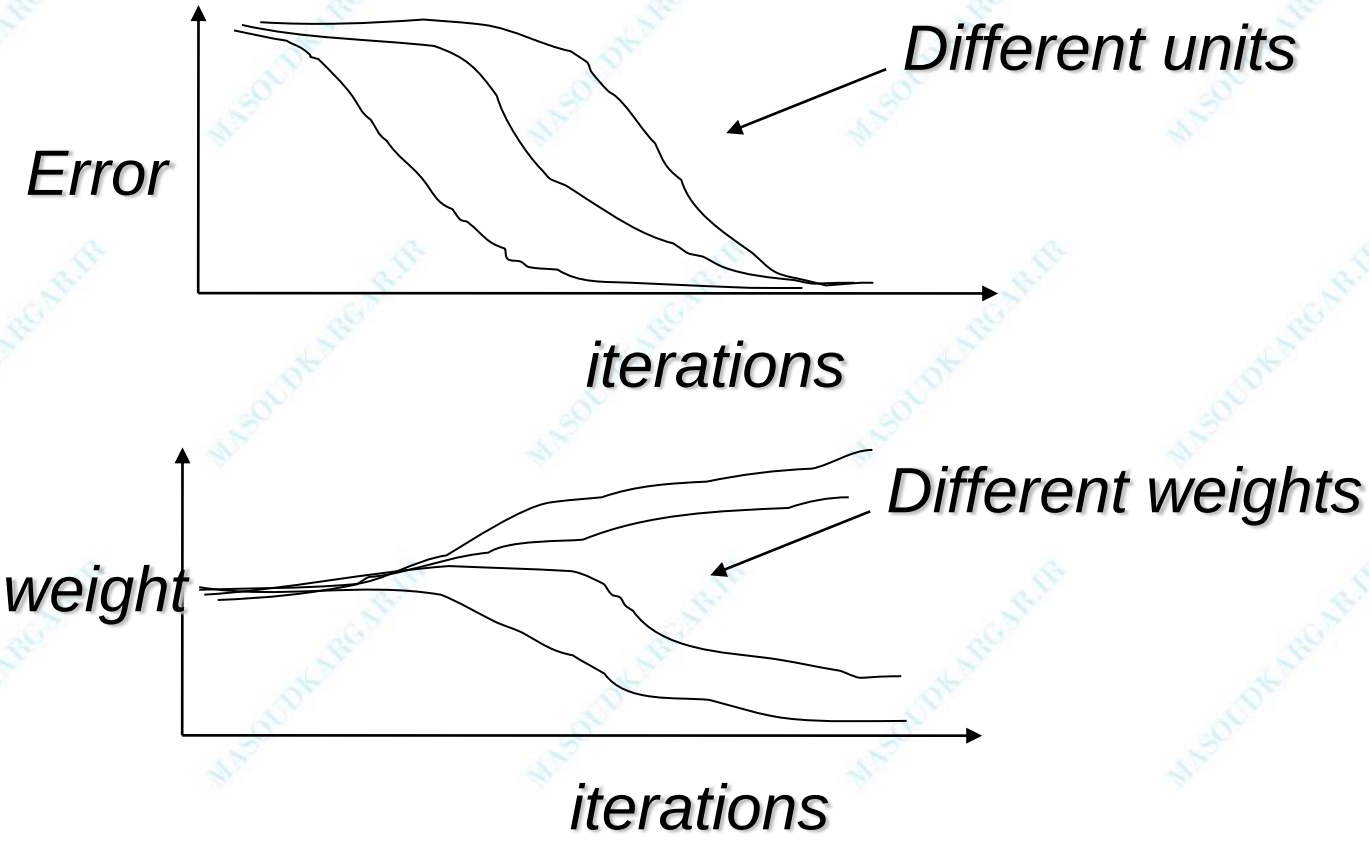
در این آزمایش که به تعداد *5000* بار تکرار شده از *8* داده مختلف به عنوان ورودی استفاده شده و شبکه با استفاده از الگوریتم *BP* موفق شده تا تابع هدف را بیاموزد.



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

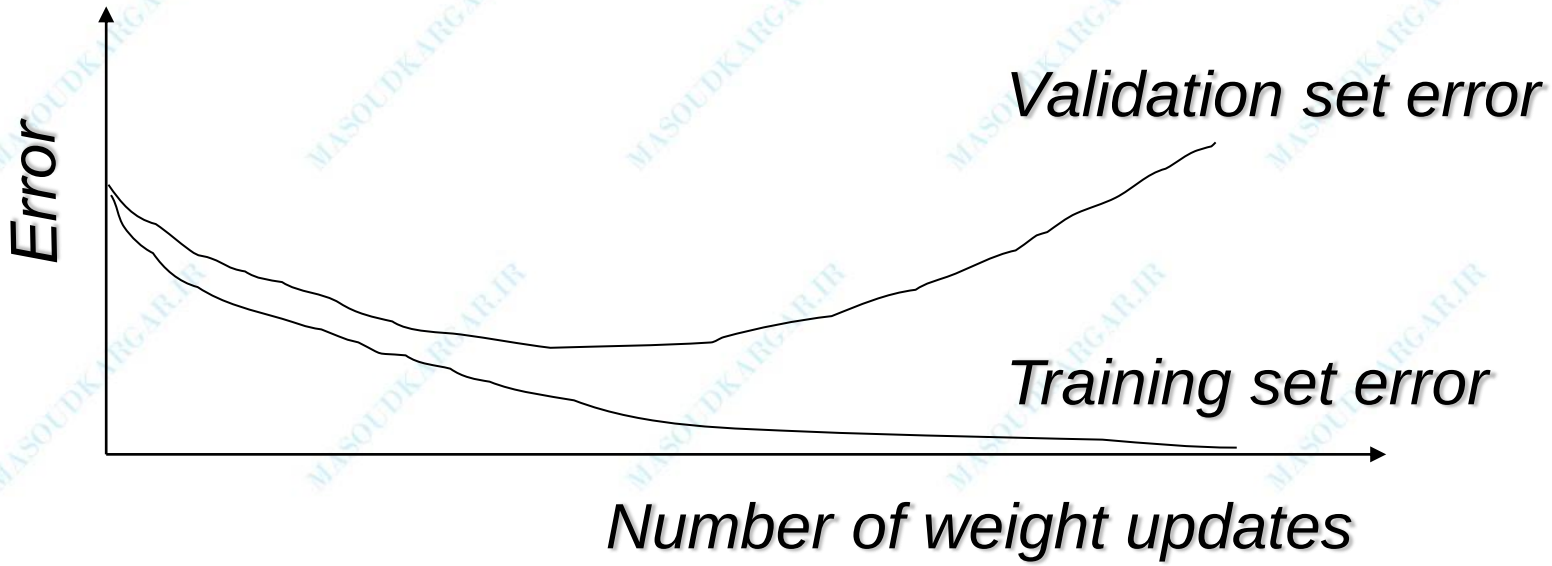
با مشاهده خروجی واحدهای لایه میانی مشخص می‌شود که بردار حاصل معادل انکدینگ استاندارد داده‌های ورودی بوده است (۱۱۱، ... ، ۰۰۱ ، ۰۰۰)

نمودار خطا


Different units
Error
iterations
Different weights
weight
iterations

قدرت تعمیم و Over fitting

- شرط پایان 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 *Over fitting* شود.



Over fitting دادن رخ دلایل

- *Over fitting* ناشی از تنظیم وزنها برای در نظر گرفتن مثالهای نادری است که ممکن است با توزیع کلی داده‌ها 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 یاد گرفته 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 تایید *Validation* و توقف یادگیری هنگامی که خطا در این مجموعه به اندازه کافی کوچک می‌شود.
- بایاس کردن شبکه برای فضاهای فرضیه ساده‌تر: یک راه می‌تواند استفاده از *weight decay* باشد که در آن مقدار وزنها در هر بار تکرار به اندازه 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 kدفعه تکرار نمود. در هر دفعه یکی از دسته‌ها بعنوان مجموعه تست و بقیه بعنوان مجموعه آموزشی استفاده خواهند شد. تصمیم‌گیری بر اساس میانگین نتایج انجام می‌شود.

دیگر روشهای

راه‌های بسیار متنوعی برای ایجاد شبکه‌های 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 ساختارهای دیگری برای شبکه
  - *Recurrent Network*

با سپاس

- ازآقای دکتر شیری – دانشگاه امیرکبیر

مراجع