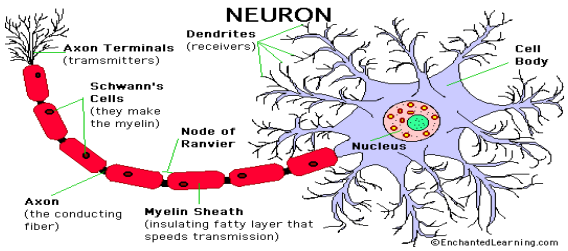
K. N. Toosi University of Technology

Fundamental intelligence



Computational neural

Computational fuzzy

Computational evolution

Machine Learning

Computational intelligence

Multi-agent systems

Expert sysyetm

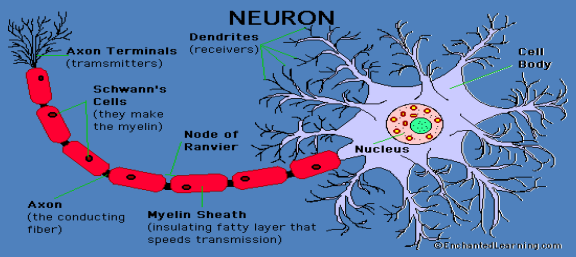
Swarm intelligence

دانشگاه خواجه نصیرالدین طوسی- دانشکده برق

K. N. Toosi University of Technology

بسمه تعالی





شبکه های عصبی

دانشگاه خواجه نصیرالدین طوسی- دانشکده برق

فهرست مطالب

فصل اول : مقدمه ای بر شبکه های عصبی

فصل دوم : ارائه ساختارهای مختلف شبکه های عصبی

فصل سوم : ارائه مفاهیم پایه در آموزش شبکه های عصبی بر اساس الگوریتم پس انتشار خطا

فصل چهارم : بررسی روشهای مختلف آموزش در شبکه های عصبی

فصل پنجم : بهبود الگوریتم پس انتشار خطا

فصل ششم : شبکه های عصبی بازگشتی

فهرست مطالب

فصل هفتم : شبکه های حافظه دار و گاما

فصل هشتم : شبکه های عصبی مدل مخچه

فصل نهم : شبکه عصبی بیزین

فصل دهم : شبکه عصبی مثلثاتی

فصل یازدهم : شبکه عصبی ویولت

فصل دوازدهم : شبکه عصبی کانولوشنال

فصل سیزدهم : کاربرد شبکه های عصبی در شناسایی، پیش بینی و کنترل سیستم ها

فصل اول : مقدمه ای بر شبکه های عصبی

سیستم های پیچیده

سیستم های پیچیده از بخش های مختلف تشکیل شده اند که کلیه بخش ها با یکدیگر در ارتباطند و همچنین بخش های سیستم ها با خارج در ارتباط هستند و در کنش و واکنش می باشند.

انواع سیستم های پیچیده

- **سیستم های حمل و نقل(زمینی، دریایی و هوایی)**
- **سیستم های هواشناسی (منطقه ای، محلی و جهانی)**
- **سیستم های بیمارستانی (بخش ها، تخصصها و ارتباط بین آنها و بیمارستانهای دیگر و بیمه و...)**
- **سیستم های اقتصادی(تحت تأثیرعواملی چون نیازها، درخواستها، موجودیها و بحرانها ... می باشد.)**
- **جوامع (فرهنگهای مختلف، تحصیلات، دارایی، فقر و امکانات فردی و اجتماعی)**
- **سیستم های بیولوژیکی(شامل مغز، مخچه، سلولها و ... می باشد)**

شبکه عصبی:

شبکه های عصبی بر اساس رفتار شناسی ماشین های بیولوژیکی موجودات زنده طراحی شده است.

هدف از شبکه های عصبی مصنوعی :

ارائه روشهائی جهت استفاده از سخت افزارها (مدارات) و نرم افزارها (الگوریتم ها) برای ایجاد قابلیت های هوشمند به دستگاه ها، روبوت ها، برنامه ها و غیره .... می باشد که قادر به یادگیری حین فرآیند می باشد.

در این درس هدف آشنایی فقط با الگوریتم ها و نحوه چگونگی اضافه نمودن آن به نرم/ سخت افزار می باشدکه در این راستا برنامه ها (الگوریتم) قادر به یادگیری می باشند.

مفهوم هوش چیست؟

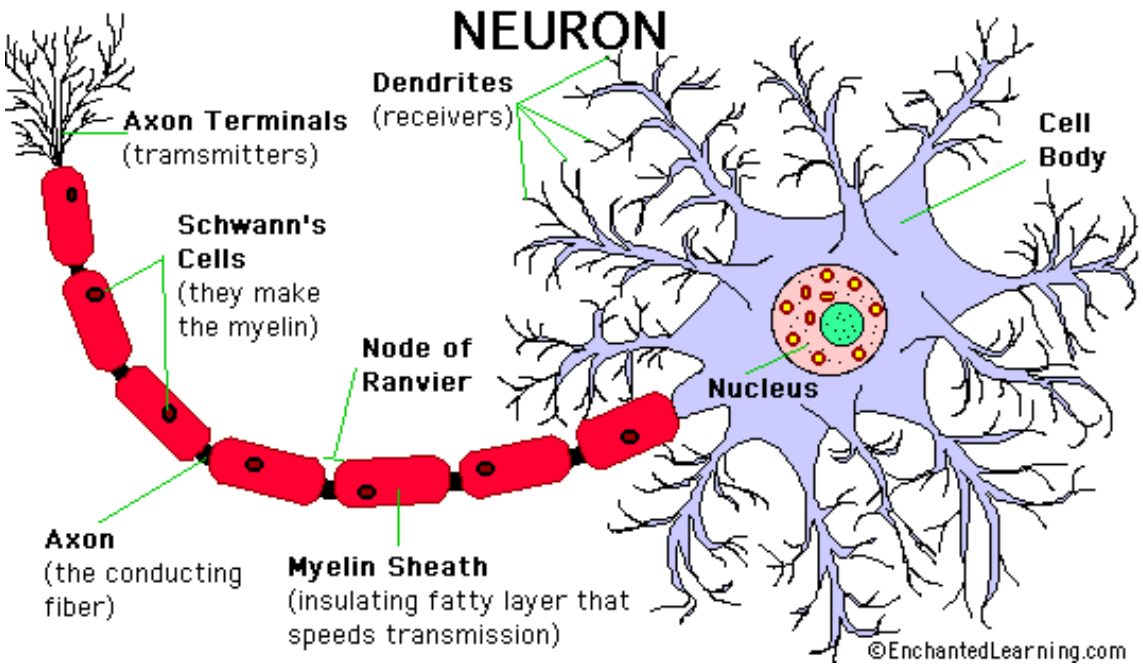
با توجه به اینکه محققان تا کنون نتوانسته اند درباره عملکرد مغز انسان که در واقع محل تجزیه و تحلیل(اتاق فرمان) ماشین بیولوژیکی موجودات زنده خصوصا انسان تعریف واقعی ارائه نمایند، لذا پاسخ به این سوال به سادگی امکان پذیر نمی باشد و نمی توان پاسخی روشن و شفاف ارائه نمود. در هر حال ماشین بیولوژیکی را با قابلیت های زیر می توان مورد بررسی قرار داد:

مفهوم هوش چیست؟

- آموزشی پذیری از طریق دریافت داده ها (اطلاعات) و یا الگوها یا ترکیبی از داده ها و الگو ها
- در حافظه نگهداری نمودن تجربیات آموزش و غیره و قابل بازسازی نمودن آنها.
- تصمیم گیری مناسب و منطقی براساس یادگیری و تجربیات آموزش دیده گذشته.
- ابراز نظر(احساسات) براساس وجود واقعیت گذشته و حال.
- خود سازمانده هی/ شکل پذیری درونی براساس نیازهای فردی و جمعی.
- ارتباط برقرار کردن منطقی الگوها براساس آموزش گذشته، به عنوان مثال، هوای ابری تصویری از امکان وجود بارندگی را در ذهن انسان تداعی می کند.
- ارائه پاسخی مناسب به نحوی که محرک های خارجی از خود یک عکس العمل مناسبی ارائه دهند.

بررسی ساختمان سلول عصبی بیولوژیک

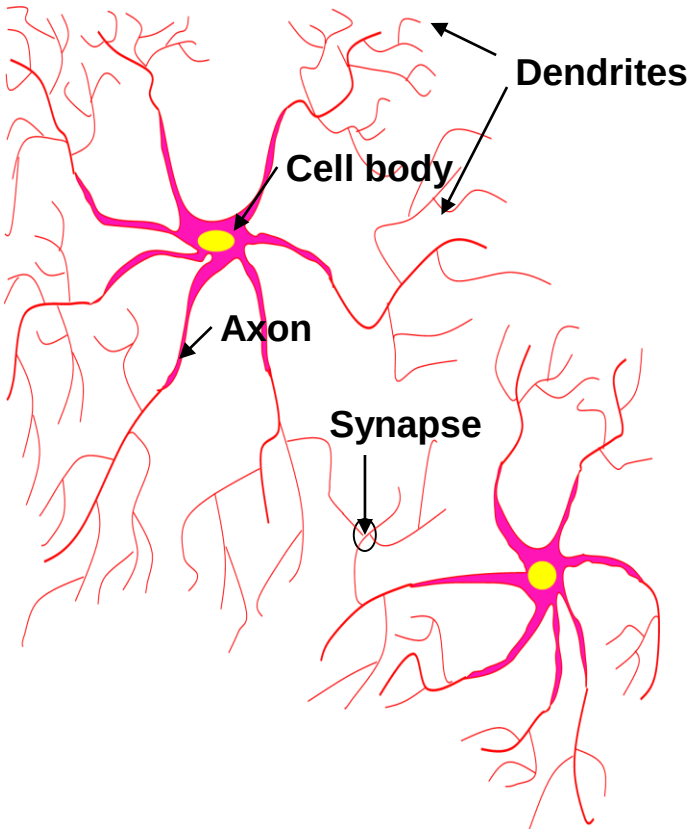
- شکل و ساختمان جسم سلولی نورون
  - سوما cell body(soma)
  - دندریت input cell(dendrite)
  - آکسون output cell(axon)

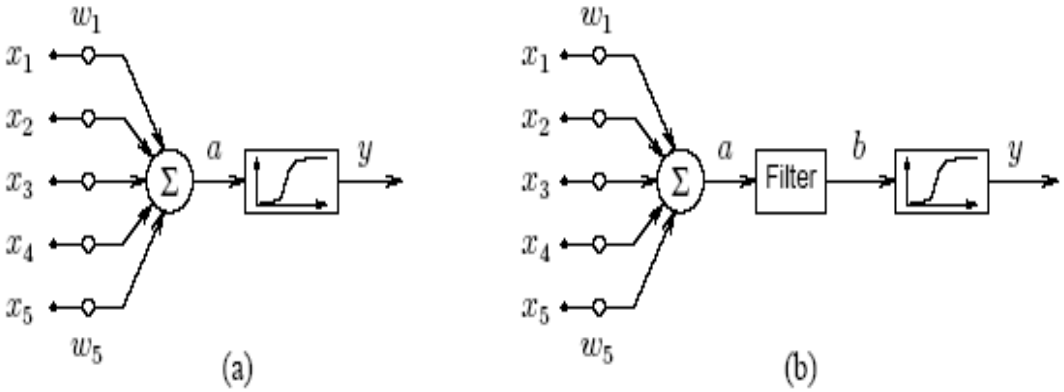


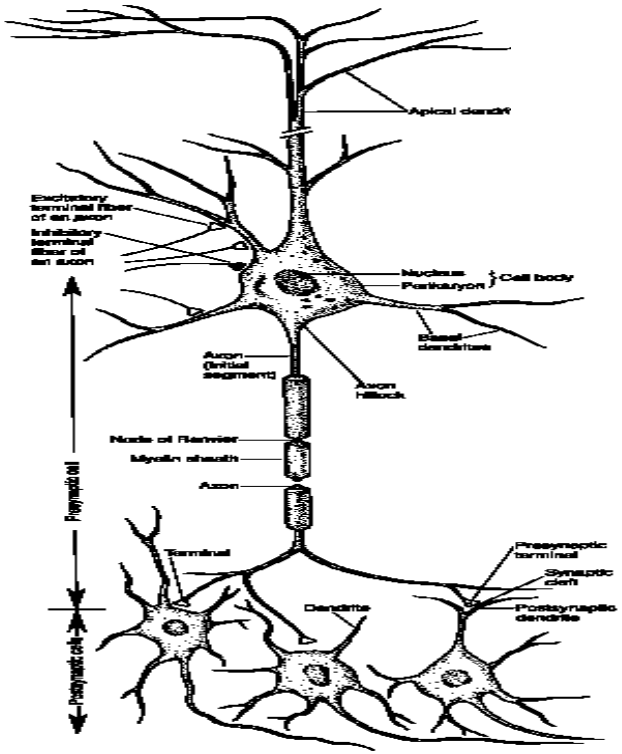
Biology Inspiration

- Neurons respond slowly
  - 10^{-3} s compared to 10^{-9} s for electrical circuits
- The brain uses massively parallel computation
  - »10^{11} neurons in the brain
  - »10^{4} connections per neuron

Human nervous system is built of cells call neuron Each neuron can receive, process and transmit electrochemical signals Dendrites extend from the cell body to other neurons, and the connection point is called synapse Signals received among dendrites are transmitted to and summed in the cell body If the cumulative excitation exceed a threshold, the cell fires, which sends a signal down the axon to other neurons







Simple neuron models

مقدمه‌ای بر شبکه‌های عصبی و قابلیت‌ها و تاریخچه آن

- مفهوم پایه شبکه‌های عصبی چیست؟
- چرا شبکه‌های عصبی قابلیت یادگیری دارند؟
- چرا شبکه‌های عصبی قابلیت تعمیم پذیری ساختار دارند؟

- ✓ شبکه‌های عصبی تقریبی از رفتار مغز و اعصاب در قسمت‌های مختلف بدن موجودات زنده هستند.
- ✓ شبکه‌های عصبی دارای قابلیت‌های مختلف بر اساس کاربرد می‌باشند.
- ✓ شبکه‌های عصبی یک پردازشگر موازی است که در یک لحظه دارای آموزش و اعمال می‌باشند.

ارزش و قابلیت یادگیری در شبکه‌های عصبی

- ☑ دارای محاسبات قوی و ارزشمند از داده و اطلاعات هستند.
- ☑ در ساختار شبکه‌های عصبی که دارای نرون‌ها و لایه‌های زیاد هستند باعث انعطاف پذیری و درجه آزادی زیاد در سیستم عصبی می‌شوند ولی زمان محاسبات طولانی‌تر خواهد شد.
- ☑ دارای توانایی زیاد در اصلاح و تعمیم پذیری می‌باشند.
- ☑ دارای قابلیت حذف اغتشاش و نویز هستند.
- ☑ قابلیت تولید الگوریتم تطبیق پذیر هوشمند بر پایه کاربرد را دارند.

ویژگیهای شبکه های عصبی

تاکنون مدل های مختلف با ساختار و الگوریتم های متنوعی از شبکه های عصبی ارائه شده است و هر چند این مدل ها با یکدیگر تفاوت دارند، اما تمام این مدل ها یک هدف مشترک را دنبال می کنند. به طور کلی سلول های عصبی که تشکیل دهنده یک شبکه عصبی می باشند ماشین های محاسباتی هستند، که از اجزای ساده(سلول) و زنجیره ای تشکیل می شوند و دارای خواص زیرند:

- ☑ **قابلیت یادگیری و تطبیق پذیری**
- ☑ **قابلیت تعمیم پذیری**
- ☑ **پردازش موازی**
- ☑ **مقاوم بودن**
- ☑ **قابلیت تقریب عمومی**

قابلیت یادگیری و تطبیق پذیری

قابلیت یادگیری یعنی توانایی تنظیم پارامترهای شبکه عصبی. برای این منظور نمونه های اولیه را به شبکه اعمال می کنند شبکه، پارامترها را بر اساس این نمونه ها تنظیم می کند. اگر نمونه های جدید به این شبکه که به این طریق آموزش دیده، اعمال شود، خروجی مناسب را با درصد خطای کوچک می توان بدست آورد. با این ترتیب شبکه های عصبی می توانند با تغییر شرایط به صورت هوشمندانه، خود را تطبیق یا اصلاح نماید.

قابلیت تعمیم پذیری:

پس از آنکه نمونه های اولیه به شبکه آموزش داده شد، شبکه می تواند در مقابل ورودیهای آموزش داده نشده (ورودیهای جدید) قرار گیرد و یک خروجی مناسب تولید نماید. این خروجی بر اساس مکانیسم تعمیم، که چیزی جز فرایند درونیایی نیست به دست می آید.

پردازش موازی:

هنگامی که شبکه عصبی در قالب سخت افزار پیاده می شود سلولهایی که در یک تراز قرار می گیرند می توانند به طور همزمان به ورودیهای آن تراز پاسخ دهند. این ویژگی باعث افزایش سرعت پردازش می شود. در واقع در چنین سیستمی، وظیفه کلی پردازش بین پردازنده های کوچکتر مستقل از یکدیگر توزیع می گردد.

مقاوم بودن:

در یک شبکه عصبی رفتار کلی آن مستقل از رفتار هر سلول در شبکه می باشد درواقع رفتار کلی شبکه برآیند رفتارهای محلی تک تک سلولهای شبکه می باشد که این امر باعث می شود تا خطا های محلی سلولها از چشم خروجی نهایی دور بمانند. این خصوصیت باعث افزایش قابلیت مقاوم بودن در شبکه عصبی می گردد.

قابلیت تقریب عمومی:

شبکه های عصبی چند لایه، با یک یا چند لایه مخفی به شرط آن که تعداد نرونهای لایه ها مخفی کافی داشته باشند، می توانند هر تابع غیر خطی پیوسته ای را در فضای ترکیبی تخمین بزنند.

Applications

- **Aerospace**
  - High performance aircraft autopilots, flight path simulations, aircraft control systems, autopilot enhancements, aircraft component simulations, aircraft component fault detectors
- **Automotive**
  - Automobile automatic guidance systems, warranty activity analyzers
- **Banking**
  - Check and other document readers, credit application evaluators
- **Defense**
  - Weapon steering, target tracking, object discrimination, facial recognition, new kinds of sensors, sonar, radar and image signal processing including data compression, feature extraction and noise suppression, signal/image identification
- **Electronics**
  - Code sequence prediction, integrated circuit chip layout, process control, chip failure analysis, machine vision, voice synthesis, nonlinear modeling





Applications

- **Financial**
  - Real estate appraisal, loan advisor, mortgage screening, corporate bond rating, credit line use analysis, portfolio trading program, corporate financial analysis, currency price prediction
- **Manufacturing**
  - Manufacturing process control, product design and analysis, process and machine diagnosis, real-time particle identification, visual quality inspection systems, beer testing, welding quality analysis, paper quality prediction, computer chip quality analysis, analysis of grinding operations, chemical product design analysis, machine maintenance analysis, project bidding, planning and management, dynamic modeling of chemical process systems
- **Medical**
  - Breast cancer cell analysis, EEG and ECG analysis, prosthesis design, optimization of transplant times, hospital expense reduction, hospital quality improvement, emergency room test advisement

Applications

- **Robotics**
  - Trajectory control, forklift robot, manipulator controllers, vision systems
- **Speech**
  - Speech recognition, speech compression, vowel classification, text to speech synthesis
- **Securities**
  - Market analysis, automatic bond rating, stock trading advisory systems
- **Telecommunications**
  - Image and data compression, automated information services, real-time translation of spoken language, customer payment processing systems
- **Transportation**
  - Truck brake diagnosis systems, vehicle scheduling, routing systems

فصل دوم :

ارائه ساختارهای مختلف شبکه های عصبی
Multi-layer Perceptron (MPL)

❖ هر شبکه عصبی دارای سه ویژگی زیر می باشد:

- مدل سلول عصبی (نوع تابع)
- ساختار شبکه عصبی (نوع توپولوژی)
- آموزش در شبکه عصبی (نوع آموزش)

خودسازماندهی ساختار درحین آموزش

مدل سلول عصبی

توابع انتقال

خروجی واقعی به تابع انتقال ویژه ای که انتخاب شده بستگی دارد و باید معیار های مورد نظر مسئله ای که سلول عصبی برای حل آن استفاده می شود را، برآورده کند.
سه نوع از پرکاربرد ترین آنها عبارتند از:

- تابع انتقال سخت محدود
- تابع انتقال خطی
- تابع انتقال لگاریتمی سیگموئید

مدل سلول عصبی

توابع انتقال

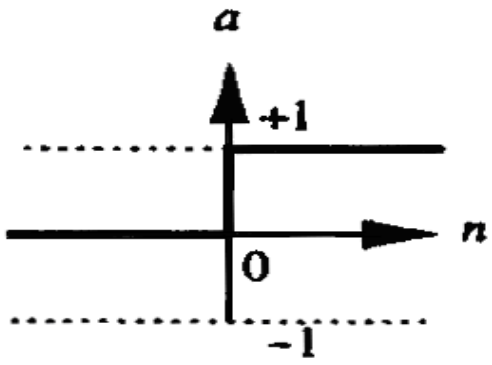
از آنجایی که تابع انتقال لگاریتمی سیگموئید، یک تابع مشتق پذیر است، عموماً از آن در شبکه های چند لایه ای استفاده می شود که با استفاده از الگوریتم پس انتشار خطا (Backpropagation) آموزش می پذیرند. نمونه ای از این تابع به صورت زیر است:

$$a = \frac{1}{1 + e^{-net \cdot g}}$$

$$g > o$$

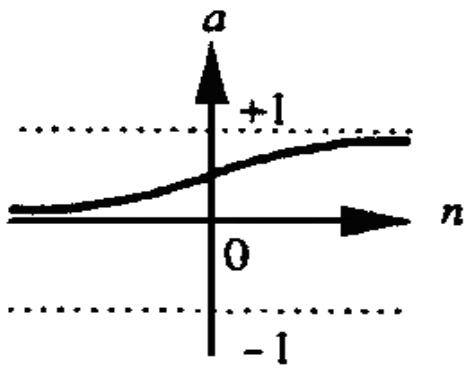
مطابق این عبارت، ورودی این تابع انتقال می تواند هر مقداری بین منفی بینهایت تا مثبت بینهایت باشد در حالیکه خروجی آن در بازه صفر و ۱ محدود شده است.

توابع انتقال



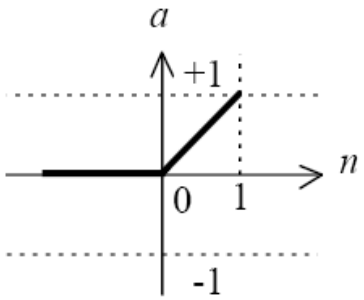
$a = hardlim(n)$

hardlim(n) = 1 if n >=0
=0 otherwise



$a = logsig(n)$

logsig(n) = 1 / (1 + exp(-n))

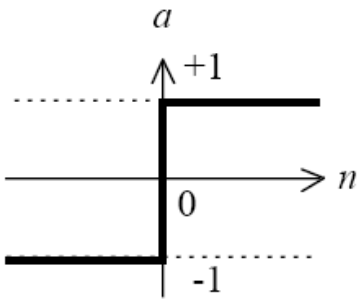




$a = poslin(n)$

Positive Linear Transfer Funct.

poslin(n) = n, if n >= 0
= 0, if n <= 0



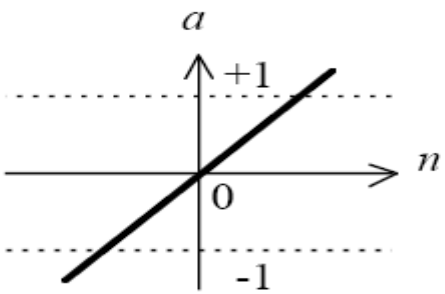


$a = hardlims(n)$

Symmetric Hard Limit Trans. Funct.

hardlims(n) = 1 if n >= 0, -1 otherwise.

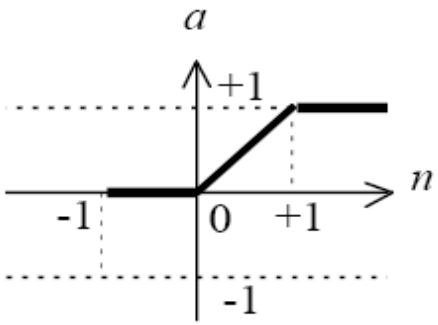
توابع انتقال



$a = purelin(n)$



purelin(n) = n

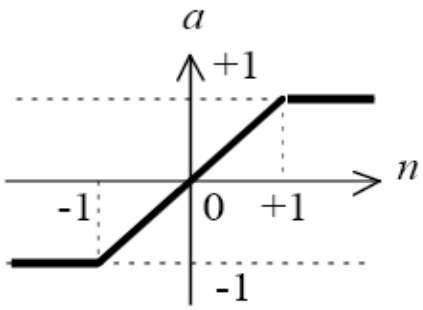


$a = satlin(n)$



Satlin Transfer Function

satlin(n) = 0, if n <= 0
= n, f 0 <= n <= 1
=1, if 1 <= n



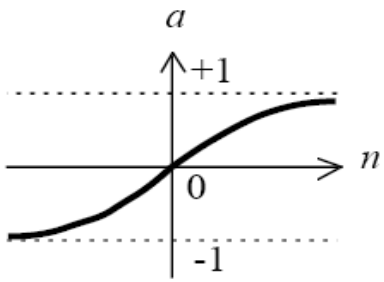
$a = satlins(n)$



Satlins Transfer Function

satlins(n) = -1, if n <= -1
= n, if -1 <= n <= 1
= 1, if 1 <= n

توابع انتقال

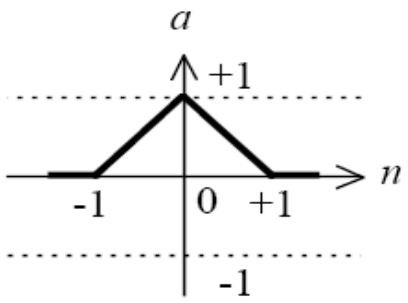




$a = tansig(n)$

Tan-Sigmoid Transfer Function

tansig(n) = 2/(1+exp(-2n))-1

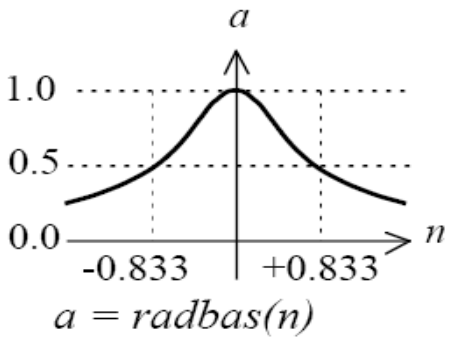




$a = tribas(n)$

Triangular Basis Function

tribas(n) = 1 - abs(n), if -1 <= n <= 1
= 0, otherwise





$a = radbas(n)$

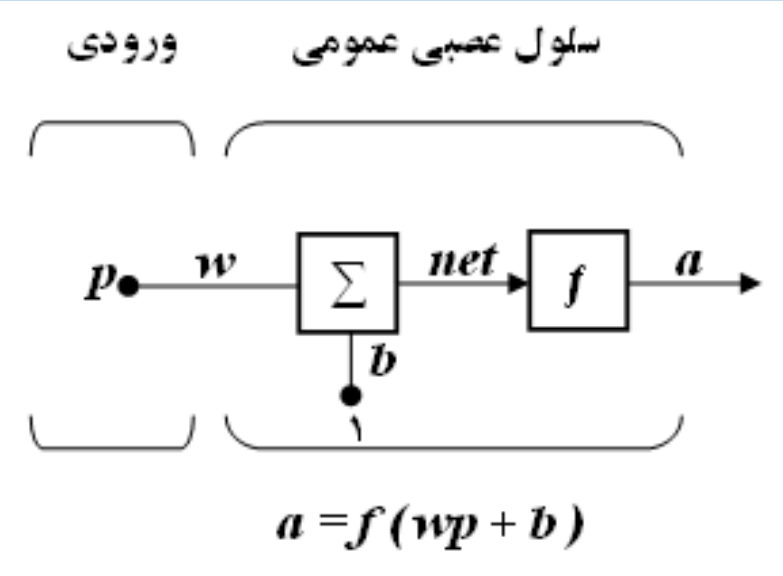
Radial Basis Function

radbas(n) = exp(-n^2)

مدل سلول عصبی

مدل سلول عصبی تک ورودی

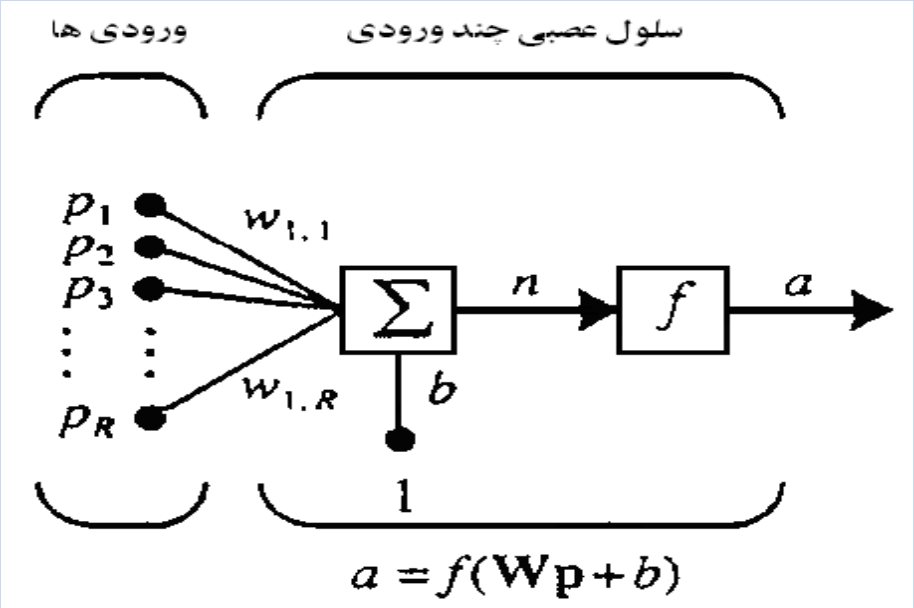
اگر این مدل ساده را با سلول عصبی بیولوژیکی که پیش از این شرح دادیم مقایسه کنیم، وزن W مطابقت دارد با سیناپس، بدنه سلول به وسیله عمل جمع و تابع انتقال بیان شده و خروجی سلول عصبی یا همان a نمایانگر سیگنال آکسون است.



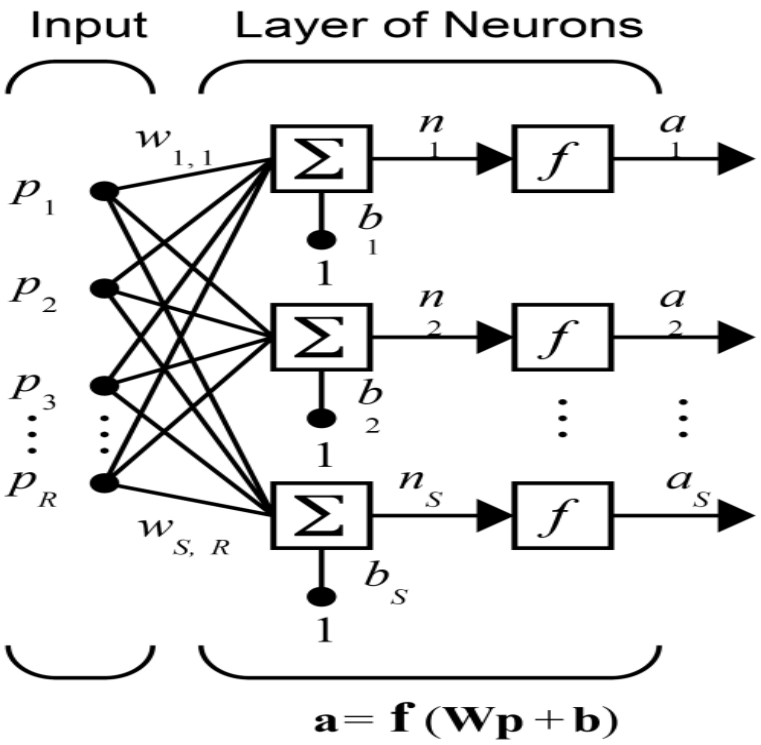
مدل سلول عصبی با چند ورودی

سلول عصبی چند ورودی

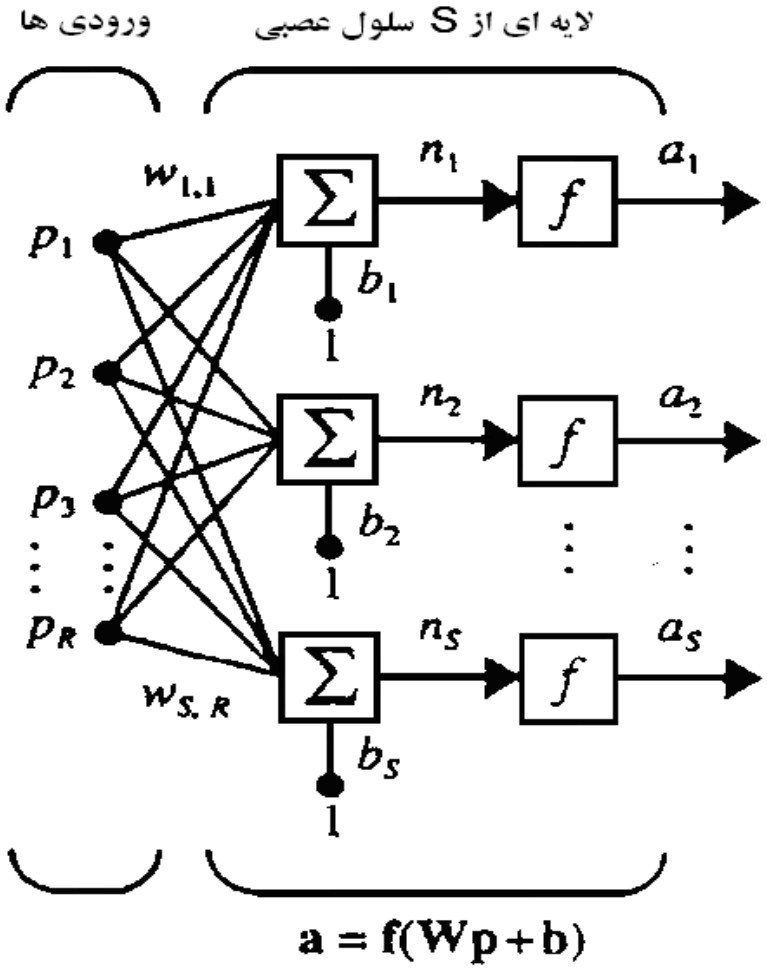
عموماً یک سلول عصبی بیش از یک ورودی دارد. هر کدام از ورودی های مجزا در وزن متناظر خود ضرب می شوند. بنابراین می توان ورودی ها را به صورت بردار **p** و وزن ها را به صورت ماتریس **W** تعریف نمود.



مدل سلول عصبی با چند ورودی و چند خروجی



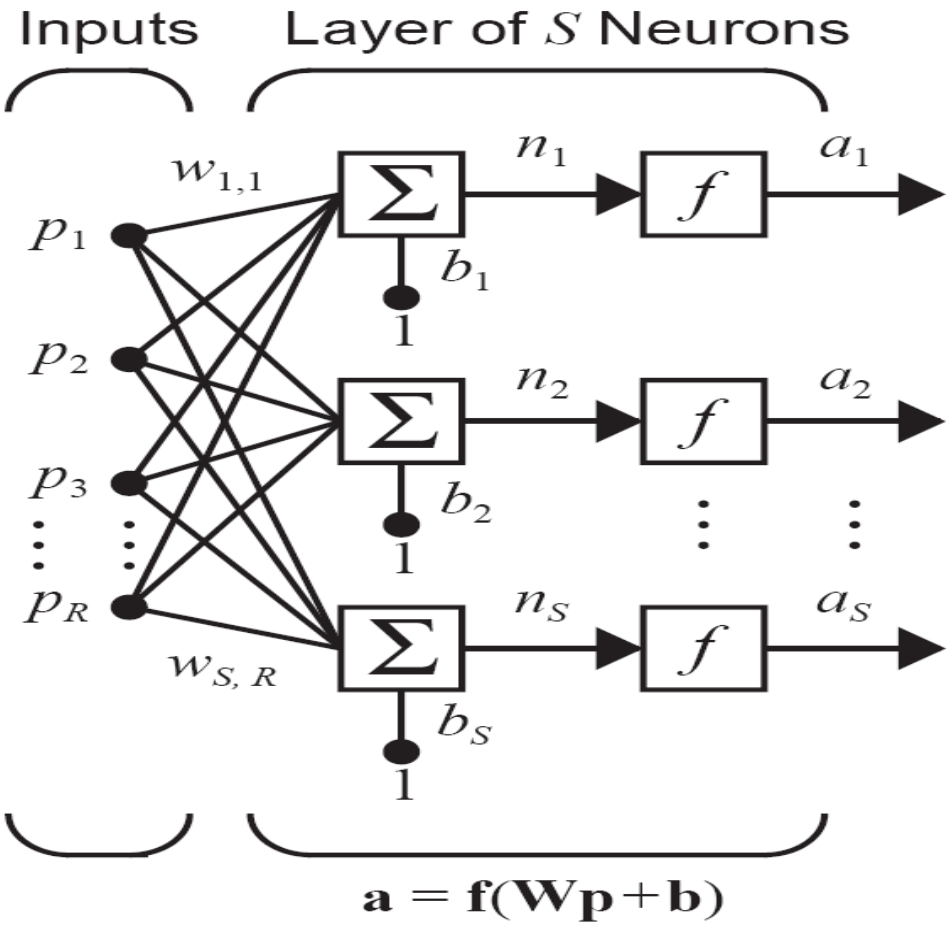
ساختار های شبکه عصبی



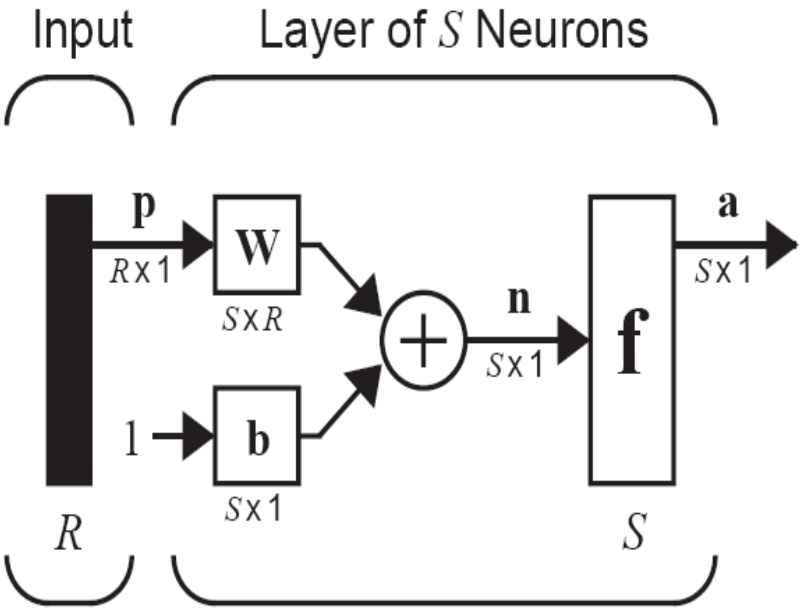
یک لایه از سلول عصبی

یک سلول عصبی، حتی با ورودی های زیاد ممکن است نیاز ما را براورده نکند. معمولاً به تعدادی از آنها که به صورت موازی عمل می کنند نیار می باشد که به آن یک شبکه یک لایه گفته می شود.

Layer of Neurons



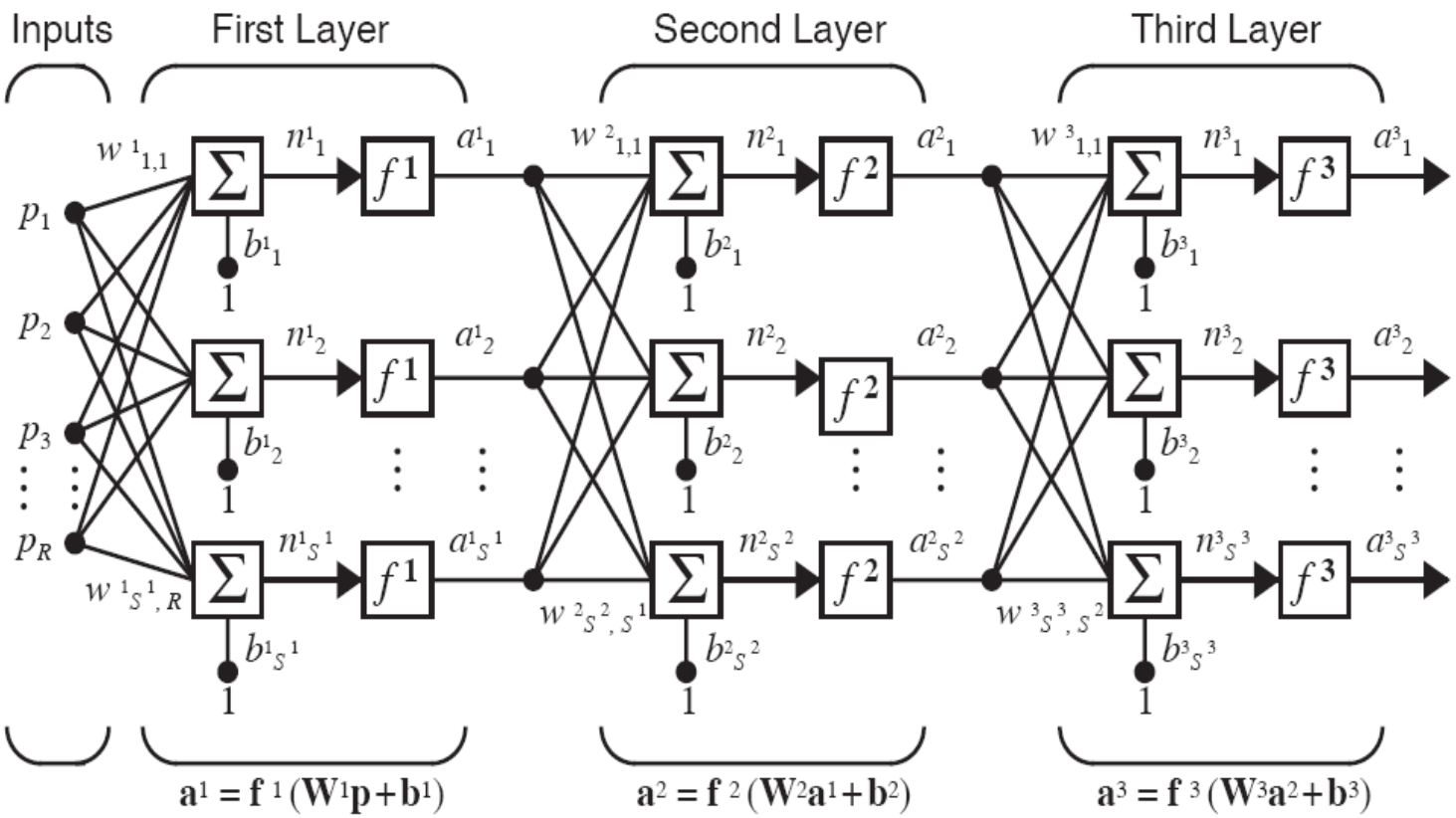
Abbreviated Notation



$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,R} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,R} \\ \vdots & \vdots & & \vdots \\ w_{S,1} & w_{S,2} & \cdots & w_{S,R} \end{bmatrix}$$

$$\mathbf{p} = \begin{bmatrix} p_1 \\ p_2 \\ \vdots \\ p_R \end{bmatrix} \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_S \end{bmatrix} \quad \mathbf{a} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_S \end{bmatrix}$$

ساختار شبکه عصبی چند لایه



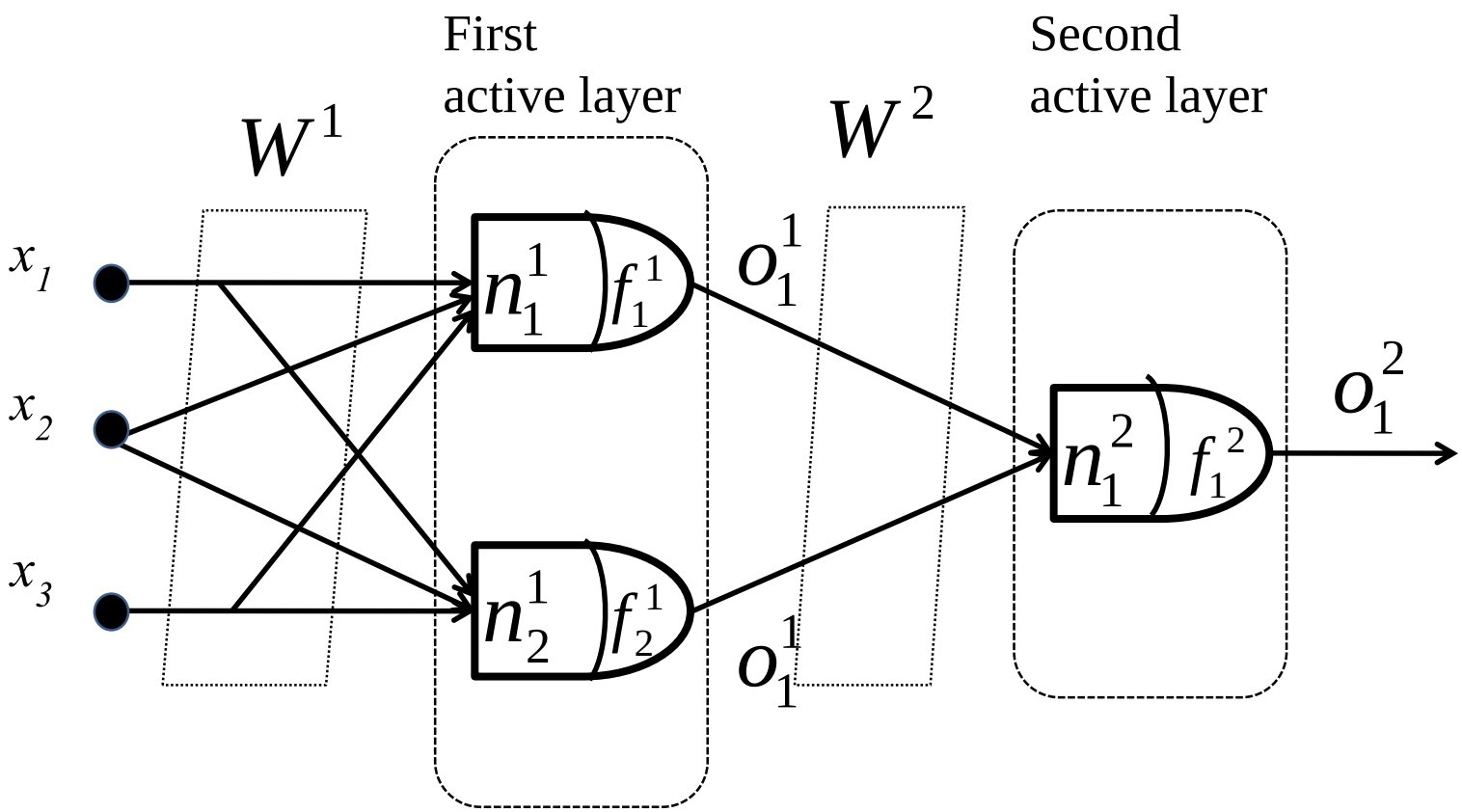
$$a^3 = f^3(W^3 f^2(W^2 f^1(W^1 p + b^1) + b^2) + b^3)$$

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

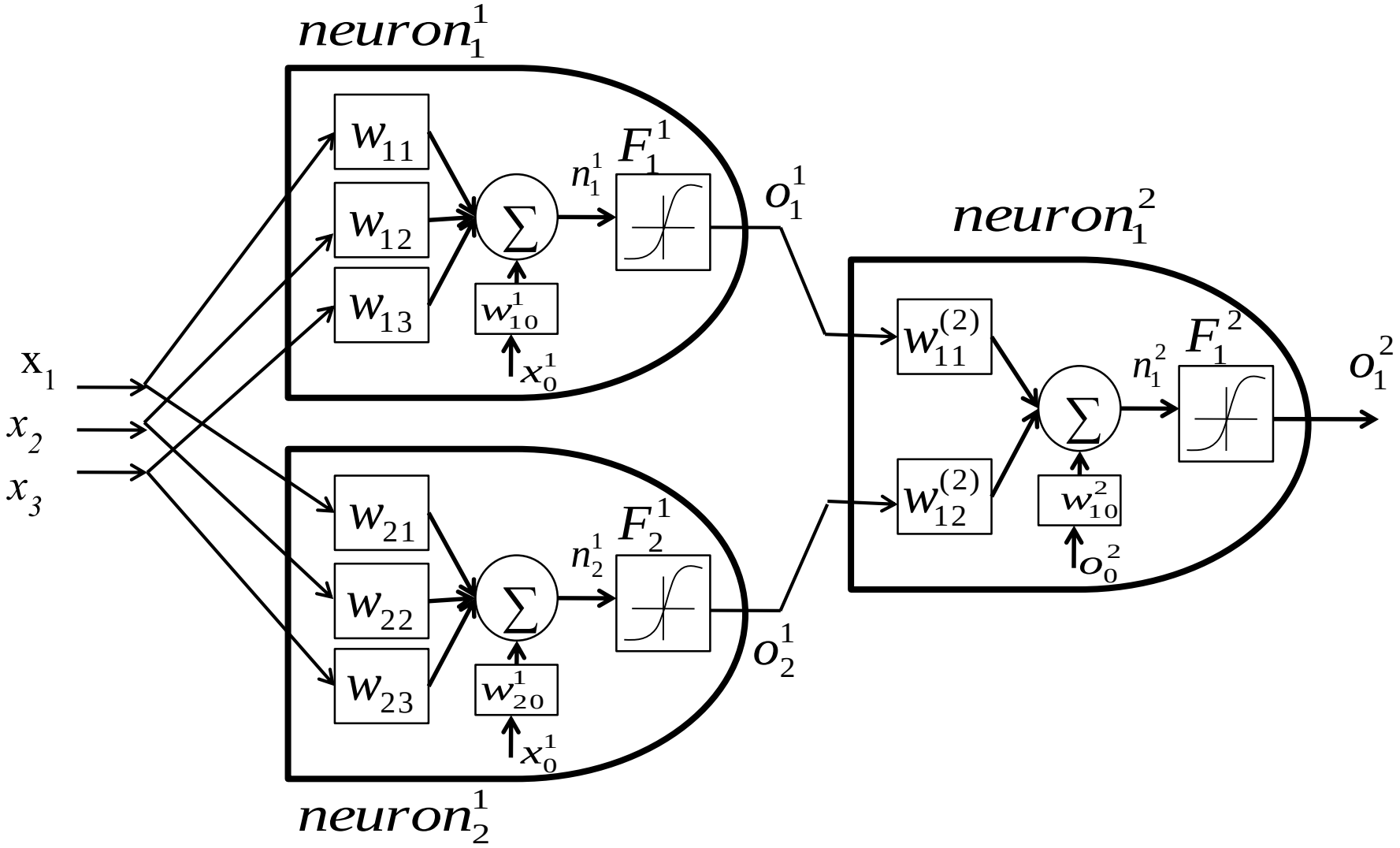
ساختار های شبکه

- **لایه های چند گانه از سلول عصبی**

در یک شبکه چند لایه، هر لایه ماتریس وزن **W** ویژه خود، بردار بایاس **b** خود، یک بردار ورودی **n** مربوط به خود و یک بردار خروجی **a** ویژه خود را دارد. لایه های مختلف می توانند تعداد نرون های متفاوتی داشته باشند. لایه ای که خروجی آن، خروجی شبکه است، یک لایه خروجی نامیده می شود. لایه های دیگر، لایه های پنهان نام دارند. شبکه های چند لایه قدرتمند تر از شبکه های تک لایه هستند. برای نمونه، یک شبکه دو لایه که شامل یک لایه اول سیگموئید و یک لایه دوم خطی می باشد، می تواند به منظور تقریب اکثر توابع اختیاری آموزش داده شود. بیشتر شبکه های کاربردی تنها دو یا سه لایه دارند.

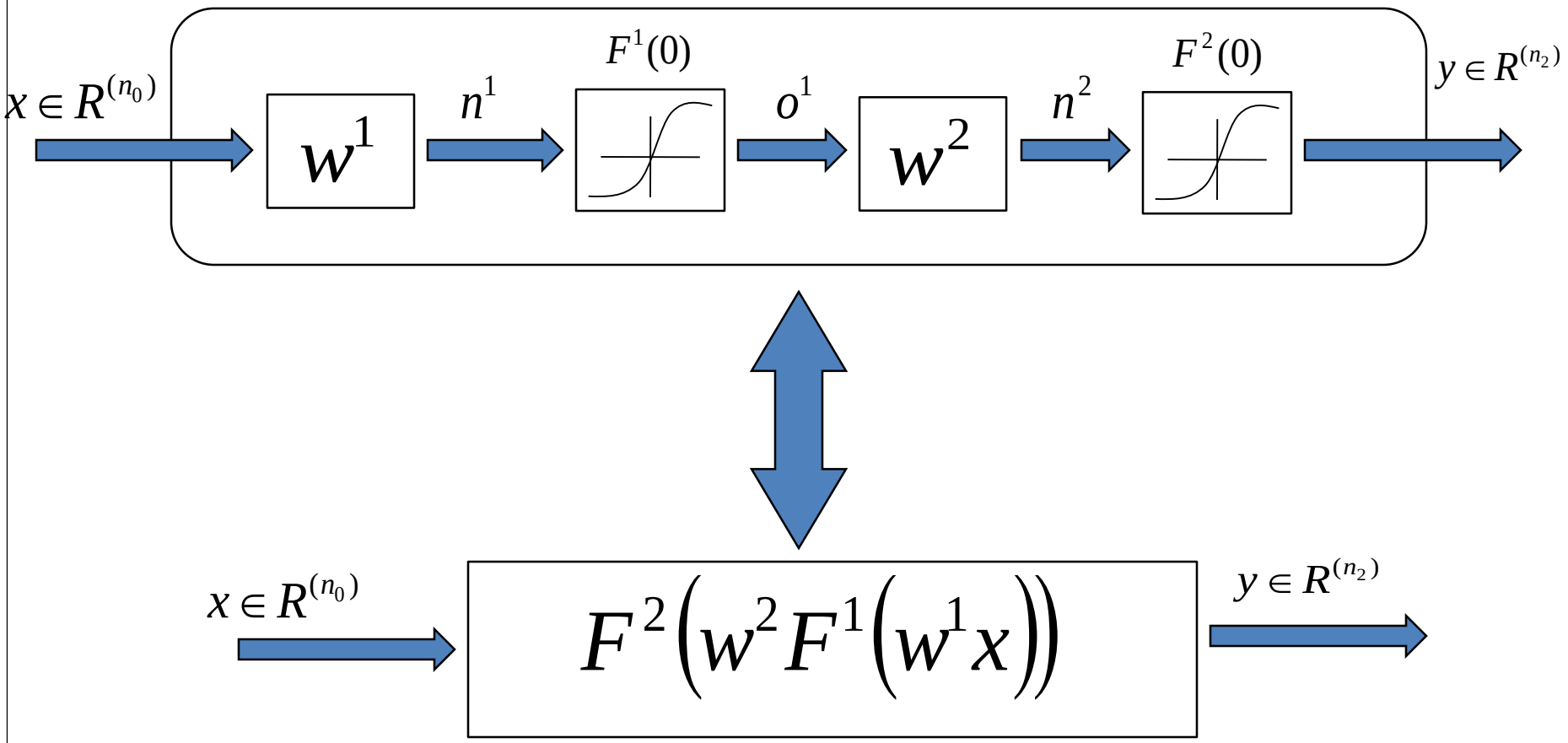


(a) General structure with two activation layers

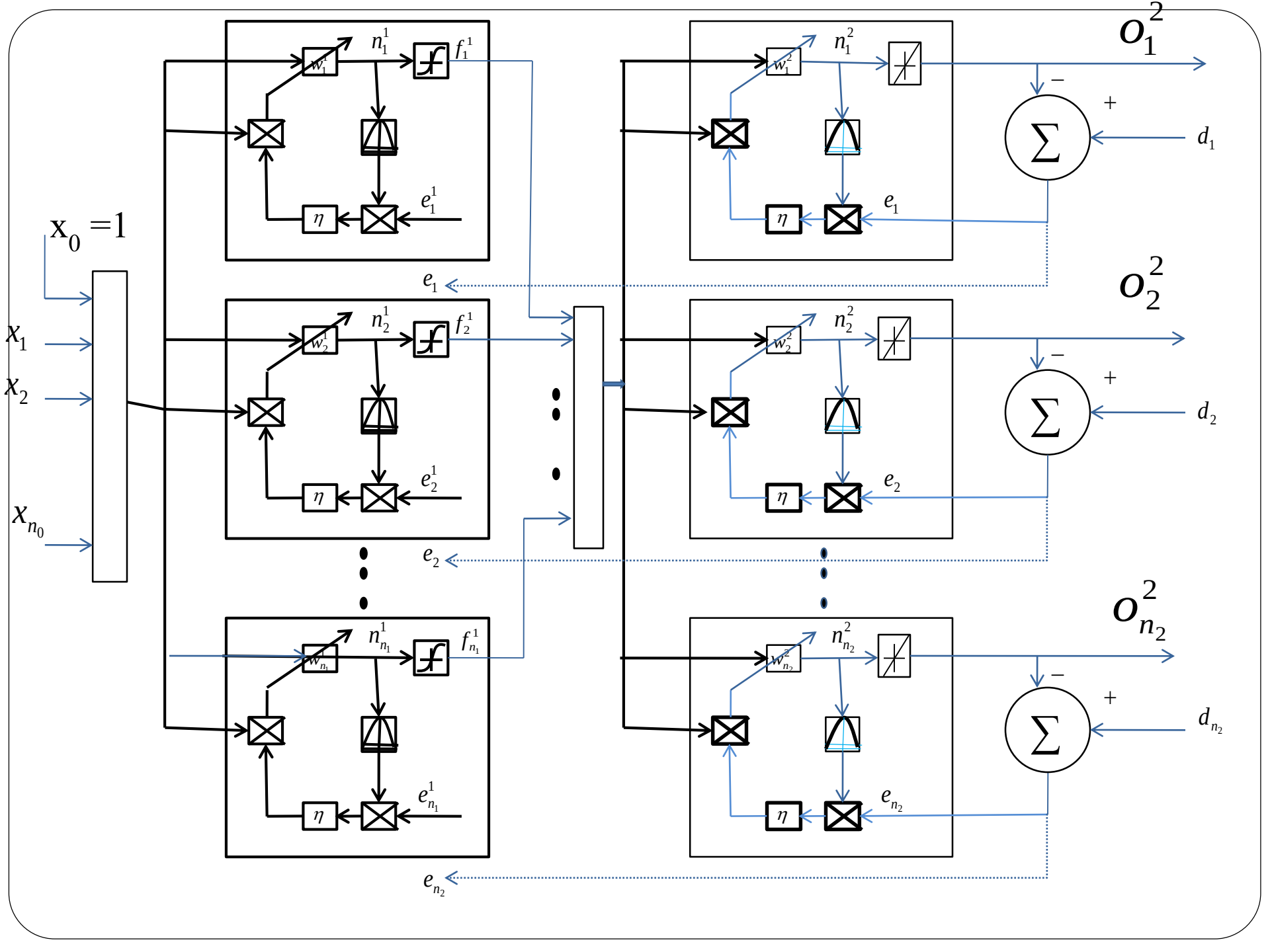


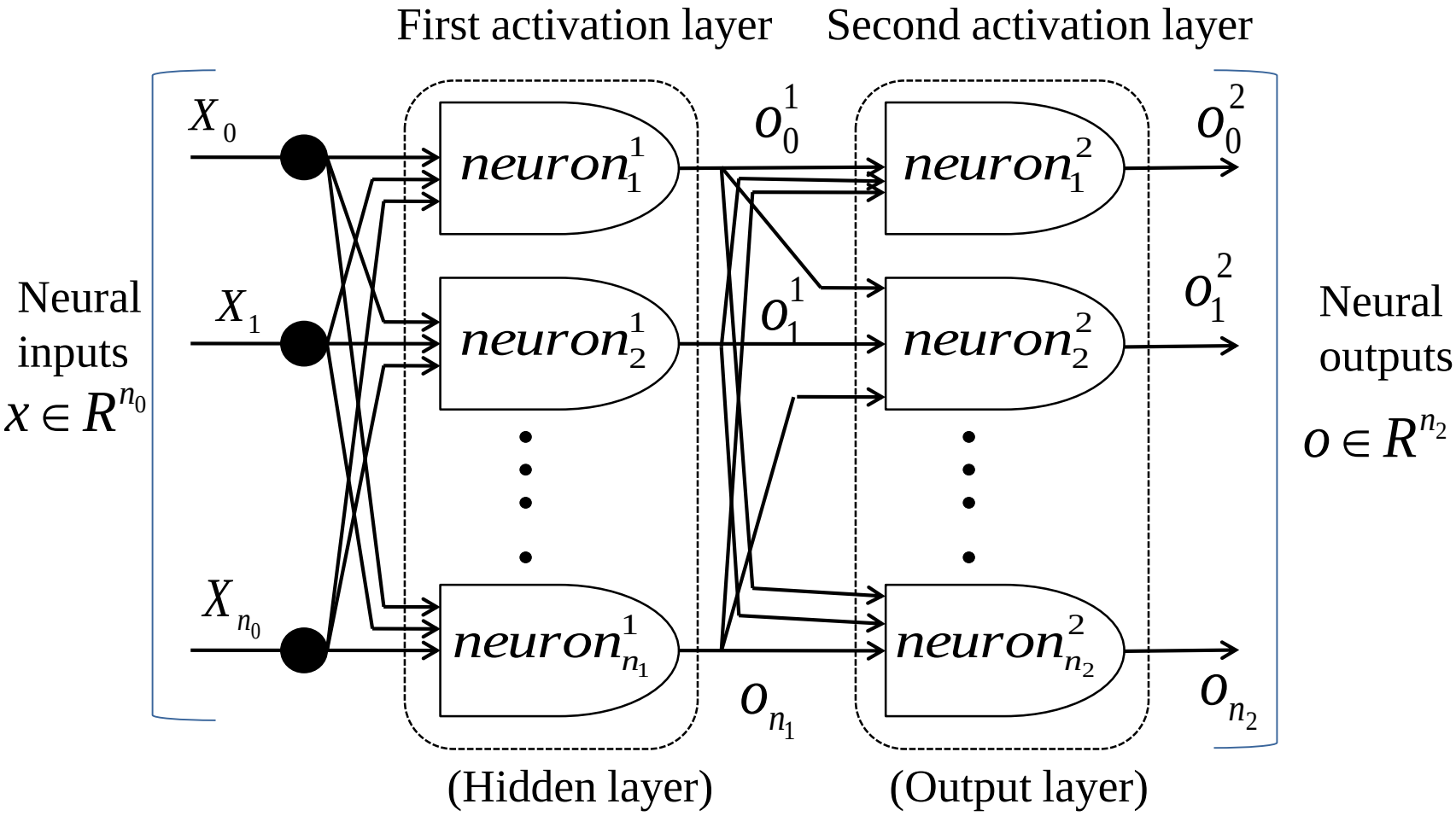
(b) Specific structure with two activation layers

Nonlinear mapping



4.3 : **Two-layered neural network: nonlinear mapping for input ($x \in R^{(n_0)}$) to output ($y \in R^{(n_2)}$)**

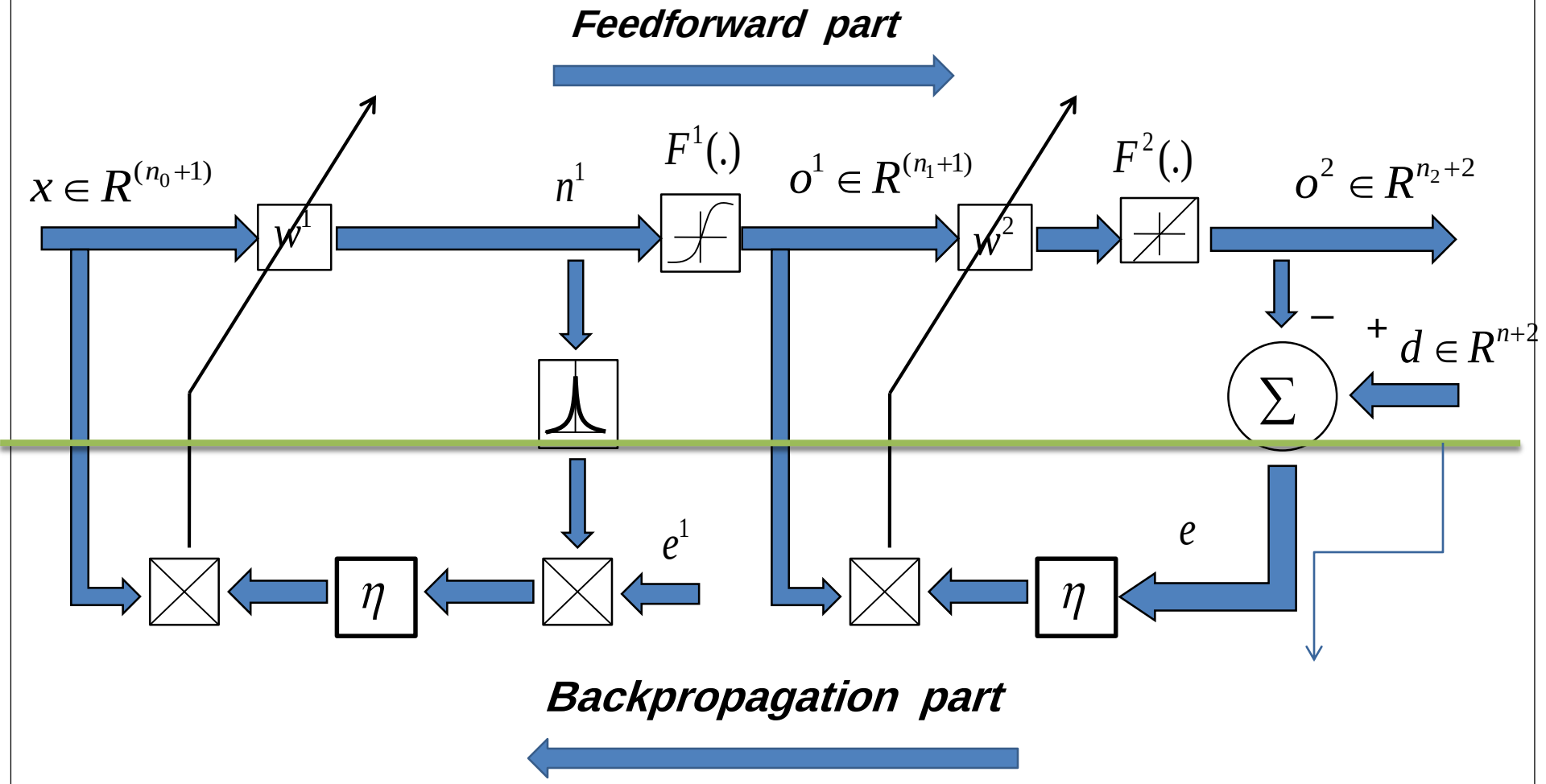

$x_0 = 1$
x_1
x_2
x_{n_0}
n_1^1
f_1^1
w_1^1
e_1^1
η
n_2^1
f_2^1
w_2^1
e_2^1
$n_{n_1}^1$
$f_{n_1}^1$
$w_{n_1}^1$
$e_{n_1}^1$
n_1^2
w_1^2
e_1
o_1^2
d_1
n_2^2
w_2^2
e_2
o_2^2
d_2
$n_{n_2}^2$
$w_{n_2}^2$
e_{n_2}
$o_{n_2}^2$
d_{n_2}
$\sum$
$-$
$+$



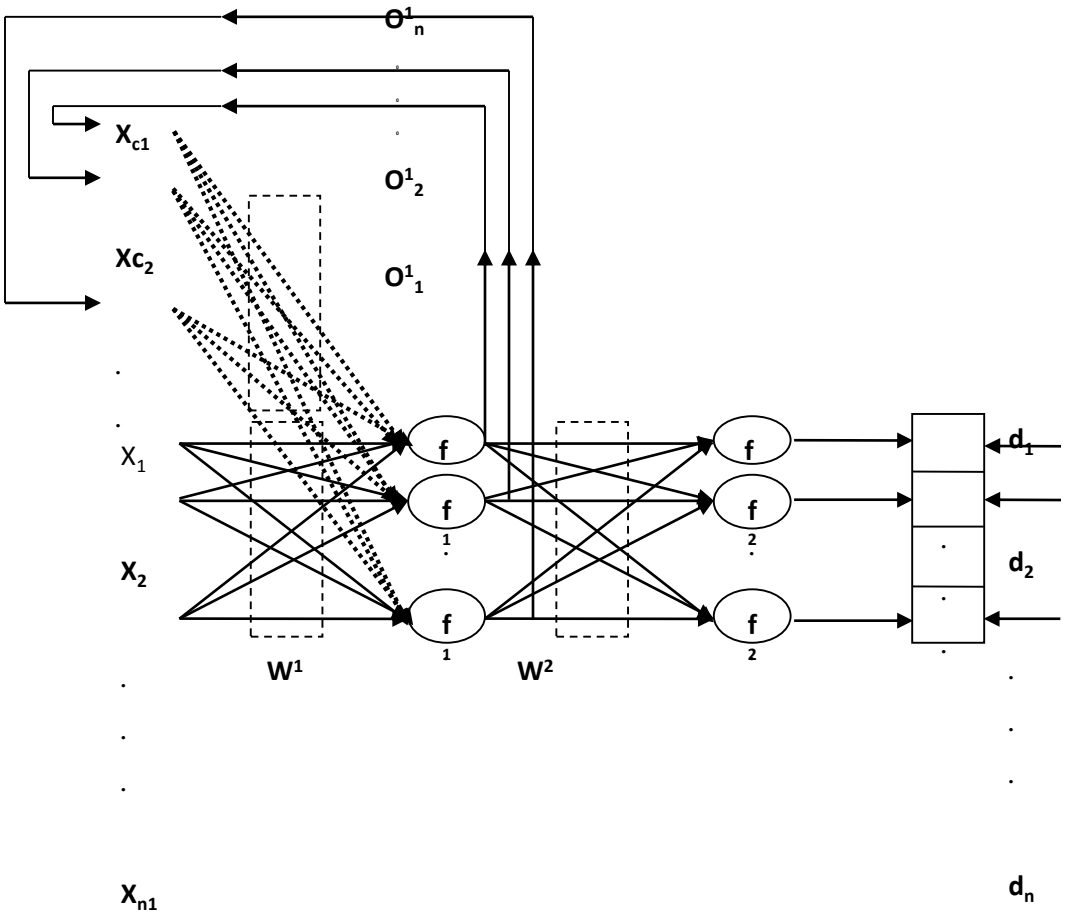
A two-layered feedforward neural network : n_1 hidden neurons, and n_2 output neurons

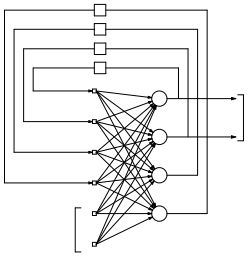
$$x = [x_1 \ldots\ldots x_i \ldots\ldots x_{n_0}]^T \in R^{n_0}$$

$$o = [o_1 \ldots\ldots o_i \ldots\ldots o_{n_1}]^T \in R^{n_1}$$



4.8 : **The matrix representation of backpropagation learning for two-layered feedforward neural network with linear output neurons**


O^1_n
X_{c1}
O^1_2
Xc_2
O^1_1
X_1
X_2
X_{n1}
f
W^1
W^2
d_1
d_2
d_n



Z^{-1}

Z^{-1}

Z^{-1}

Z^{-1}

خروجی ها

عملگر تاخیر

ورودی ها

فصل سوم :

ارائه مفاهیم پایه در آموزش شبکه های عصبی

بر اساس الگوریتم پس انتشار

قوانین یادگیری شبکه های عصبی

- ☑ یادگیری بدون مربی
- ☑ یادگیری تقویتی
- ☑ یادگیری با مربی

یادگیری با مربی

در یادگیری با مربی قانون یادگیری به وسیله مجموعهٔ مثالهایی (مجموعهٔ آموزشی) از رفتارهای مناسب شبکه، ایجاد می شود:

$$\{x_1 , t_1\} , \{x_2 , t_2\} , \dots , \{x_Q , t_Q\}$$

الگوریتم انتشار به عقب

این الگوریتم می تواند در آموزش پرسپترونهای چند لایه که قادر به حل مسائل غیرخطی هستند بکار رود. پرسپترون چند لایه که با الگوریتم انتشار به عقب آموزش دیده باشد در حال حاضر پرکاربرد ترین شبکه عصبی است که به طور گسترده استفاده می شود.

- تقریب تابع

شبکه‌های عصبی به عنوان شناساگر و کنترلر سیستمهای دینامیکی کاربردهای فراوانی دارند. شبکه‌های عصبی پرسپترون چند لایه به عنوان یک تقریب گر عمومی شناخته می‌شوند و این قابلیت در آنها نهفته است که به عنوان یک گزینه بسیار مناسب و مشهور جهت کنترل غیرخطی و همچنین شناسایی و مدلسازی سیستمهای غیرخطی مطرح هستند.

بنابراین شبکه های عصبی به عنوان تقریب زننده های توابع به کار می روند. برای مثال در سیستمهای کنترل، هدف یافتن یک تابع بازخورد مناسب است که خروجی های اندازه گیری شده را به ورودی های کنترل نگاشت کند.

الگوریتم انتشار به عقب

- **روش آموزش بر مبنای الگوریتم انتشار به عقب**

در شبکه های چند لایه، خروجی یک لایه، ورودی لایه بعدی را تشکیل می دهد.

$$o^{s+1} = f^{s+1}\left(W^{s+1} o^{s} + b^{s+1} \right)$$

$$\text{For} \quad s = 0,1,\ldots, M-1$$

- **شاخص عملکرد**

شاخص عملکرد در این الگوریتم، خطای میانگین مربعات می باشد.

$$J(x) = E[e^T e] = E[(t-o)^T (t-o)]$$

الگوریتم انتشار به عقب

- الگوریتم شیبدارترین نزول (GD) برای تقریب خطای میانگین (w)

$$net_i^s(k) = W_i^s(k) O^{s-1}(k)$$

For s=1, It means in first active layer the summation of weigthed input is;

$$Net^1(k) = W^1(k) O^0(k)$$

of is; W^1 $[n_1 \times n_0] \rightarrow W^1 =$ The **size**

$$\begin{bmatrix} w_{1,1}, w_{1,2}, w_{1,3}, \ldots\ldots, w_{1,n_{0-1}}, w_{1,n_0} \\ w_{2,1}, w_{2,2}, w_{2,3}, \ldots\ldots, w_{2,n_{0-1}}, w_{1,n_0} \\ w_{3,1}, w_{3,2}, \ldots\ldots\ldots\ldots\ldots\ldots, w_{3,n_0} \\ w_{4,1}, \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots, w_{4,n_0} \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ \ldots\ldots\ldots\ldots\ldots\ldots \\ w_{n_1,1}, w_{n_1,2}, w_{n_1,3}, \ldots\ldots\ldots\ldots, w_{n_1,n_0} \end{bmatrix}$$

of **is;** O^0 $[n_0] \rightarrow O^0 =$

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ . \\ . \\ x_{n_0-1} \\ x_{n_0} \end{bmatrix}$$

The **size**

And the output of first active layer is:

$$O^1 = \begin{bmatrix} o_1^1 \\ o_2^1 \\ o_3^1 \\ . \\ . \\ o_{n_1-1}^1 \\ o_{n_1}^1 \end{bmatrix} = \begin{bmatrix} f_1^1(.),0,0,0,0,0,0 \\ 0, f_2^1(.),0,0,0,0,0 \\ 0,0, f_3^1(.),0,0,0,0 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ 0,0,0,0,0, f_{n_1-1}^1(.),0 \\ 0,0,0,0,0,0, f_{n_1}^1(.) \end{bmatrix} \begin{bmatrix} net_1^1 \\ net_2^1 \\ net_3^1 \\ \ldots\ldots \\ \ldots\ldots \\ net_{n_1-1}^1 \\ net_{n_1}^1 \end{bmatrix}$$

الگوریتم انتشار به عقب

- الگوریتم شیبدارترین نزول (GD) برای تقریب خطای میانگین (w)

$$w_{i,j}^{m}(k+1) = w_{i,j}^{m}(k) + \Delta w_{i,j}^{m}(k)$$

$$\Delta w_{i,j}^{m}(k) = -\eta_w \frac{\partial J(k)}{\partial w_{i,j}^{m}} = s_i^m(k) o_j^{m-1}(k)$$

$$s_i^m(k) \equiv \frac{\partial J(k)}{\partial net_i^m(k)} \qquad o_j^{m-1}(k) = \frac{\partial net_i^m(k)}{\partial w_{i,j}^{m}(k)}$$

الگوریتم انتشار به عقب

- الگوریتم شیبدارترین نزول (GD) برای تقریب خطای میانگین (b)

$$b_i^m(k+1) = b_i^m(k) + \Delta b_i^m(k)$$

$$\Delta b^m(k) = -\eta_b \frac{\partial J}{\partial b_i^m} = s_i^m$$

$$s_i^m(k) \equiv \frac{\partial J(k)}{\partial b_i^m(k)}$$

الگوریتم انتشار به عقب

قانون زنجیره ای

برای شبکه چند لایه، خطا تابع صریحی از وزن ها در لایه های پنهان نمی باشد، بنابراین مشتقات به آسانی قابل محاسبه نیستند.

$$\frac{\partial J}{\partial w_{i,j}^{m}} = \frac{\partial J}{\partial net_{i}^{m}} \times \frac{\partial net_{i}^{m}}{\partial w_{i,j}^{m}}$$

$$\frac{\partial J}{\partial b_{i}^{m}} = \frac{\partial J}{\partial net_{i}^{m}} \times \frac{\partial net_{i}^{m}}{\partial b_{i}^{m}}$$

الگوریتم انتشار به عقب

- قانون زنجیره ای

$$net_{i,j}^{m} = \sum_{j=1}^{S^{m-1}} w_{i,j}^{m} o_{j}^{m-1} + b_{i}^{m}$$

$$\frac{\partial net_{i}^{m}}{\partial w_{i,j}^{m}} = o_{j}^{m-1} \quad , \quad \frac{\partial net_{i}^{m}}{\partial b_{i}^{m}} = 1$$

$$s_{i}^{m} \equiv \frac{\partial J}{\partial net_{i}^{m}}$$

$$w_{i,j}^{m}(k+1) = w_{i,j}^{m}(k) - \eta s_{i}^{m} o_{i}^{m-1}$$

$$\frac{\partial J}{\partial w_{i,j}^{m}} = s_{i}^{m} o_{j}^{m-1}$$

$$b_{i}^{m}(k+1) = b_{i}^{m}(k) - \eta s_{i}^{m}$$

$$\frac{\partial J}{\partial b_{i}^{m}} = s_{i}^{m}$$

الگوریتم انتشار به عقب حساسیت ها

برای محاسبه حساسیت ها به کاربرد دیگری از قانون زنجیره ای نیاز داریم:

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

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

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

الگوریتم انتشار به عقب حساسیت ها

اکنون می توانیم بخشی را مشاهده کنیم که الگوریتم انتشار به عقب نام خود را از آن گرفته است. حساسیت ها در شبکه به سمت عقب از آخرین لایه به اولین لایه انتشار می یابند:

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

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

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

الگوریتم انتشار به عقب حساسیت ها

نقطه آغاز برای ارتباط بازگشتی معادله قبل، از معادله زیر به دست می آید:

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

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

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

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

الگوریتم انتشار به عقب

- **انتخاب ساختار شبکه**

شبکه های چند لایه تقریباً می توانند برای تقریب هر تابعی استفاده شوند به شرطی که به تعداد کافی نورون در لایه های پنهان داشته باشیم. اگر بخواهیم تابعی را تقریب بزنیم که تعداد بسیار زیادی نقطه خمیدگی دارد، نیازمند آن خواهیم بود که تعداد بسیار زیادی نورون در لایه پنهان داشته باشیم.

الگوریتم انتشار به عقب

همگرایی

هنگامی که الگوریتم انتشار به عقب همگرا می شود، نمی توانیم مطمئن باشیم که یک جواب بهینه داریم. بهتر این است که چندین وضعیت اولیه متفاوت را به منظور اطمینان از اینکه جواب بهینه به دست آمده است، امتحان کرد.

تعمیم دادن

شبکه باید بتواند آنچه را یاد گرفته است به کلیه حالت ها (جمعیت ورودی) تعمیم دهد. برای اینکه یک شبکه بتواند تعمیم دهد باید پارامتر های کمتری از نقاط داده موجود در مجموعه آموزش داشته باشد. در شبکه های عصبی، در تمام مسائل مدل کردن، ما می خواهیم از ساده ترین شبکه ای استفاده کنیم که بتواند به طور مناسب مجموعه آموزش را ارائه دهد.

اصلاح های اکتشافی الگوریتم انتشار به عقب

الگوریتم انتشار به عقب یک جهش بزرگ در پژوهش های شبکه عصبی بود. به هر حال الگوریتم پایه برای اکثر کاربرد های عملی بسیار کند است.

☒ ایرادهای انتشار به عقب

سطح کارایی برای یک شبکه چند لایه ممکن است شامل چندین نقطه کمینه محلی باشد و انحنای آن می تواند در نواحی مختلف از فضای پارامتر کاملاً تغییر کند. در مسائلی که انحنا شدیداً روی فضای پارامتر تغییر می کند، انتخاب یک نرخ آموزش مناسب برای الگوریتم آموزش مشکل خواهد بود. برای نواحی مسطح یک نرخ آموزش بزرگ نیاز می باشد درحالیکه برای نواحی با انحنای زیاد نیازمند یک نرخ آموزش کوچک هستیم.

☑ رویه هایی جهت بهبود الگوریتم انتشار به عقب

- جنبش آنی

این روش با هموار کردن نوسانات منحنی فضای پارامترها همگرایی را بهبود می دهد که این کار را می توان با استفاده از یک فیلتر پایین گذر انجام داد.

اصلاح های اکتشافی الگوریتم انتشار به عقب

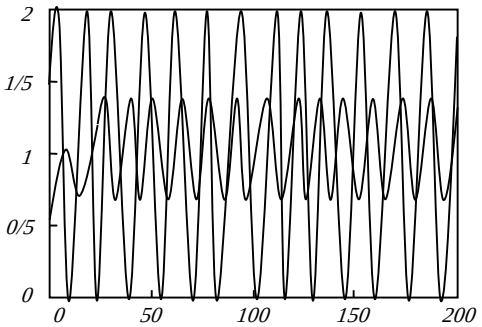
مثالی از استفاده فیلتر پایین گذر

در فرمول زیر $w(k)$ ورودی فیلتر و $y(k)$ خروجی فیلتر وγ ضریب جنبش آنی است.

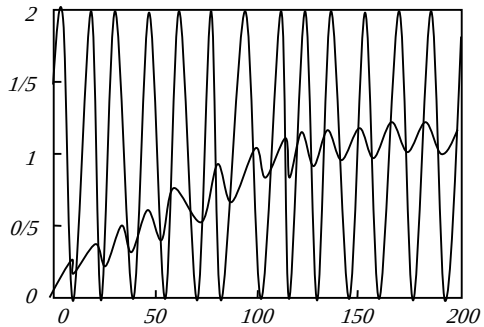
$$y(k) = \gamma y(k-1) + (1-\gamma) w(k)$$

$$0 \leq \gamma < 1$$

برای این مثال ورودی فیلتر موج سینوسی در نظر گرفته شده است. تاثیر این فیلتر در شکلهای زیر دیده می شود:



a) γ=0/9



b) γ=0/98

$$w(k) = 1 + \sin\left(\frac{2\pi k}{16}\right)$$

اصلاح های اکتشافی الگوریتم انتشار به عقب

◈استفاده از فیلتر پایین گذر درمورد شبکه عصبی

زمانی که فیلتر جنبش آنی به تغییرات پارامتر در به روز رسانی وزن ها و بایاس ها اضافه می شود، معادلات زیر برای جنبش آنی اصلاحی انتشار به عقب به دست می آید:

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

$$\Delta \mathbf{b}^m(k) = -\alpha \mathbf{s}^m$$

با استفاده از جنبش آنی توانایی استفاده از نرخ آموزش بزرگتر با حفظ ثبات الگوریتم را داریم خصوصیت دیگر جنبش آنی این است که در مدتی که منحنی در یک جهت ثابت باشد تمایل به تسریع همگرایی دارد و دلیل نامگذاری جنبش آنی به این خاطر است که تمایل به واداد کردن منحنی به ادامه در جهت مشابه دارد. هرچه مقدار γ بیشتر باشد، منحنی دارای جنبش آنی بیشتری است.

اصلاح های اکتشافی الگوریتم انتشار به عقب

✓نرخ آموزش متغیر

با استفاده از افزایش نرخ آموزش در سطوح هموار و سپس کاهش آن زمانیکه شیب افزایش می یابد می توان همگرایی را بیشتر کرد. بنابراین می توان همگرایی را توسط تنظیم نرخ آموزش، در مدت آموزش افزایش داد. تعیین زمان تغییر نرخ آموزش و اینکه چه مقدار آن را تغییر دهیم، یک لم است.

قوانین الگوریتم انتشار به عقب با نرخ آموزش متغییر

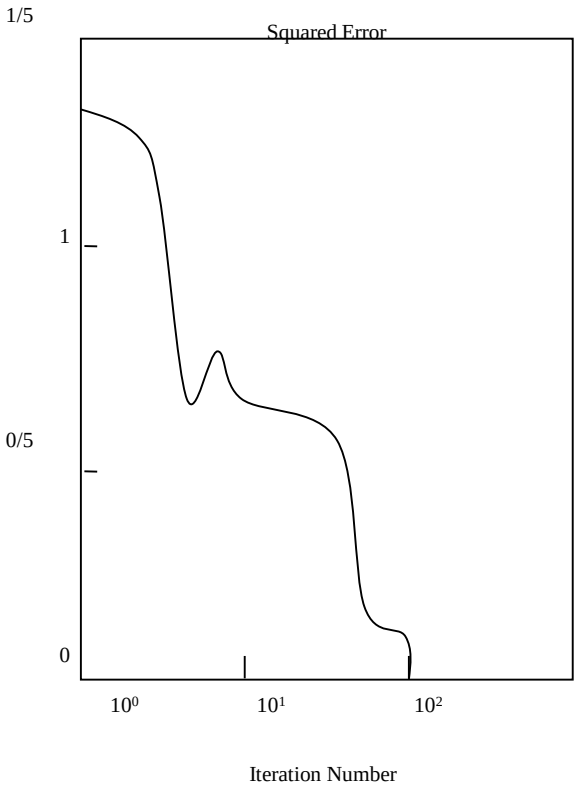
- اگر خطای آموزشی بیشتر از یک میزان معین ζ (بطور نمونه ۱ تا ۵ درصد) بعد از به روز رسانی وزن ها افزایش یابد، به روز رسانی وزن ها رها شده، نرخ آموزش در فاکتور $0< \rho <1$ ضرب می شود و ضریب جنبش آنی صفر مقدار دهی می شود.

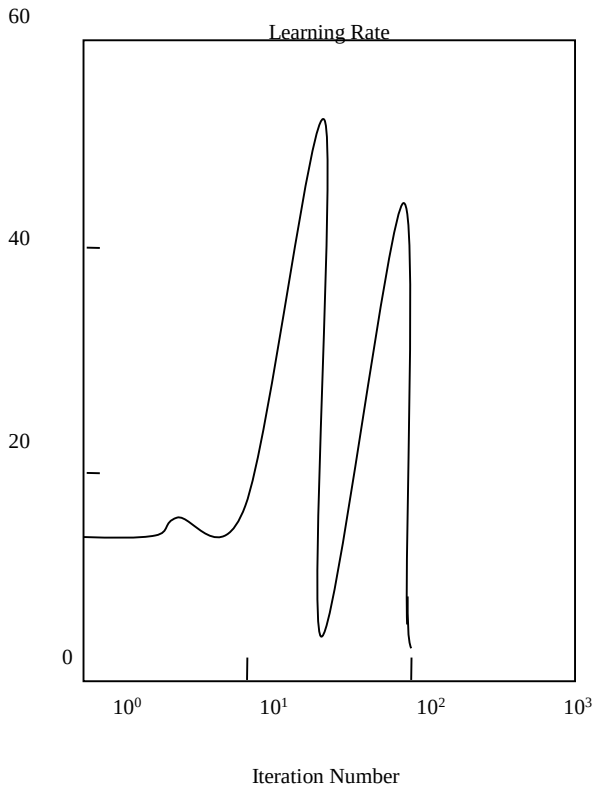
- اگر خطای مربعی بعد از به روز رسانی وزن ها کاهش یابد، به روز رسانی وزنها مورد قبول بوده و نرخ آموزشی در فاکتور $\acute{\eta} >1$ ضرب می شود. اگر ضریب جنبش آنی قبلاً صفر شده باشد، با مقدار اصلیش دوباره مقدار دهی می شود.

- اگر خطای مربعی کمتر از ζ کاهش یابد، به روز رسانی وزن ها مورد قبول واقع شده اما نرخ آموزش و ضریب جنبش آنی بدون تغییر می مانند.

اصلاح های اکتشافی الگوریتم انتشار به عقب

- مثالی از تغییر ترخ آموزش در شکل زیر دیده می شود:





اصلاح های اکتشافی الگوریتم انتشار به عقب

اشکالات روش های اصلاحی

اصلاحهای اکتشافی می تواند معمولاً همگرایی بسیار سریعتری را برای برخی از مسائل فراهم کند. به هرحال دو اشکال اصلی برای این روشها وجود دارد:

- ✓ این اصلاحات نیاز به مقدار دهی چندین پارامتر دارند و کارایی الگوریتم حساس به تغییرات این پارامتر هاست.
- ✓ این اصلاحات بعضی مواقع ممکن است در همگرایی مسئله موفق نشود.

روش پیشنهادی برای افزایش سرعت همگرایی

الگوریتم ترکیبی

در این روش یک الگوریتم ترکیبی برای آموزش با مربی شبکه های عصبی پیشخور ارائه شده است. این الگوریتم برای یک شبکه دو لایه ارائه شده اما می توان آن را برای شبکه های چند لایه نیز به کار برد.

الگوریتم ارائه شده برای افزایش سرعت آموزش، در الگوریتم انتشار به عقب پایه استفاده شده است.

در این الگوریتم دو روش مختلف برای به روز کردن وزن ها استفاده شده است:

۱. یکی از روش های آموزش استاندارد برای لایه اول

۲. استفاده از رابطه زیر در آموزش لایه دوم

e و d از روابط زیر به دست می آیند:

$$e = d - (Wx + b) \qquad d = f^{-1}(d)$$

$$\min E\left[(d - y)^T (d - y)\right] \approx \min E\left[\left(f'(\bar{d}) * \bar{e}\right)^T \left(f'(\bar{d}) * \bar{e}\right)\right]$$

روش پیشنهادی برای افزایش سرعت همگرایی

- با استفاده از جایگزینی طرف راست رابطه اصلی، در الگوریتم آموزش، وزن های لایه آخر در تابع غیر خطی لایه آخر ظاهر نمی شوند.

- در این روش، ابعاد فضای جستجوی مؤثر برای الگوریتم غیر خطی کاهش می یابد. بنابراین تعداد تکرار های آموزش کمتر خواهد شد.

روش پیشنهادی برای افزایش سرعت همگرایی

گام های الگوریتم ترکیبی

I. انتخاب وزن ها و بایاس ها به طور تصادفی و یا با استفاده از روش های مقدار اولیه
II. به روز کردن وزن ها و بایاس ها با اسفاده از یک روش استاندارد مانند الگوریتم انتشار به عقب پایه
III. بررسی قانون توقف و پایان دادن به آموزش در صورت ارضا شدن این قانون یا رفتن به قدم ۴ در غیر اینصورت

- در الگوریتم ارائه شده ابتدا پارامتر های هر دو لایه از روش استاندارد برای آموزش استفاده می کنند اما در صورت کم شدن تغییرات خطا، از روش ترکیبی استفاده شده که روش آموزش پارامترهای لایه دوم تغییر کرده و به صورتی که ارائه شد آموزش می بیند.

فصل چهارم :

بررسی روشهای مختلف آموزش در شبکه های عصبی

Learning Rules

Supervised Learning

Network is provided with a set of examples of proper network behavior (inputs/targets)

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

Reinforcement Learning

Network is only provided with a grade, or score, which indicates network performance

Unsupervised Learning

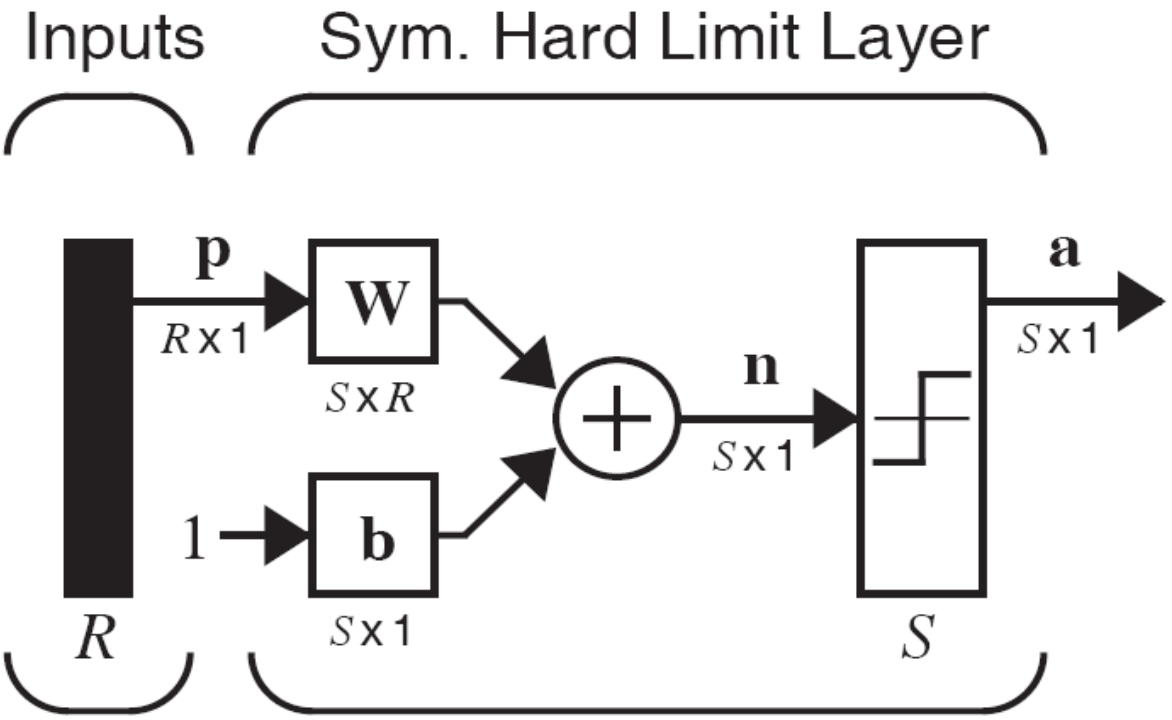
Only network inputs are available to the learning algorithm. Network learns to categorize (cluster) the inputs.

Learning Rules

- Supervised Learning
  - Early learning algorithms
  - First order gradient methods
  - Second order gradient methods

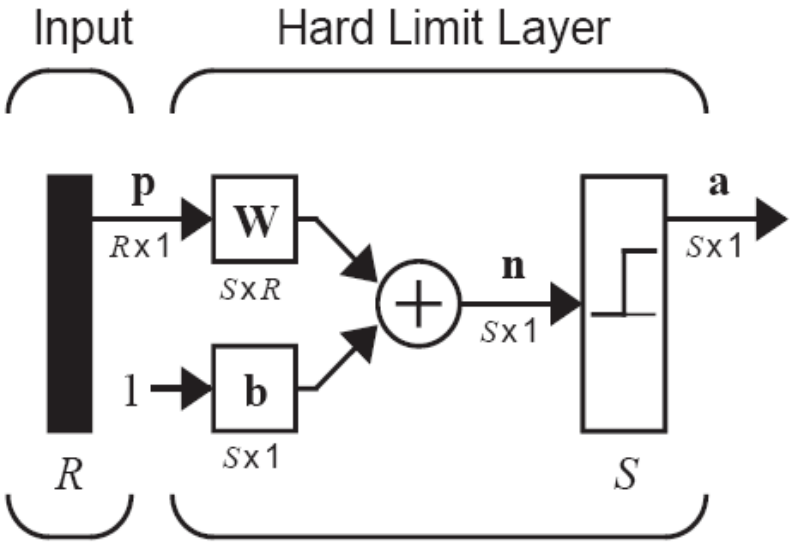
- Early learning algorithms
  - Designed for single layer neural networks
  - Generally more limited in their applicability
  - Some of them are
    - Perceptron learning
    - LMS or Widrow- Hoff learning
    - Grossberg learning

McCulloch-Pitts Perceptron



$$\mathbf{a} = \mathbf{hardlims}(\mathbf{Wp} + \mathbf{b})$$

Perceptron Architecture AGAIN!!!!



$$\mathbf{a} = \mathbf{hardlim}(\mathbf{Wp}+\mathbf{b})$$

$$\mathbf{W} = \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,R} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,R} \\ \vdots & \vdots & & \vdots \\ w_{S,1} & w_{S,2} & \cdots & w_{S,R} \end{bmatrix}$$

$${}_i\mathbf{w} = \begin{bmatrix} w_{i,1} \\ w_{i,2} \\ \vdots \\ w_{i,R} \end{bmatrix} \qquad \mathbf{W} = \begin{bmatrix} {}_1\mathbf{w}^T \\ {}_2\mathbf{w}^T \\ \vdots \\ {}_S\mathbf{w}^T \end{bmatrix}$$

$$a_i = hardlim(n_i) = hardlim({}_i\mathbf{w}^T\mathbf{p} + b_i)$$

Unified Learning Rule

If $t = 1$ and $a = 0$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} + \mathbf{p}$

If $t = 0$ and $a = 1$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} - \mathbf{p}$

If $t = a$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old}$

$$e = t - a$$

If $e = 1$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} + \mathbf{p}$

If $e = -1$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} - \mathbf{p}$

If $e = 0$, then ${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old}$

$${}_1\mathbf{w}^{new} = {}_1\mathbf{w}^{old} + e\mathbf{p} = {}_1\mathbf{w}^{old} + (t-a)\mathbf{p}$$

$$b^{new} = b^{old} + e$$

A bias is a weight with an input of 1.

Multiple-Neuron Perceptrons

To update the ith row of the weight matri

$$ {}_i\mathbf{w}^{new} = {}_i\mathbf{w}^{old} + e_i\mathbf{p} $$

$$ b_i^{new} = b_i^{old} + e_i $$

Matrix form:

$$ \mathbf{W}^{new} = \mathbf{W}^{old} + \mathbf{e}\mathbf{p}^T $$

$$ \mathbf{b}^{new} = \mathbf{b}^{old} + \mathbf{e} $$

Perceptron Rule Capability

The perceptron rule will always converge to weights which accomplish the desired classification, assuming that such weights exist.

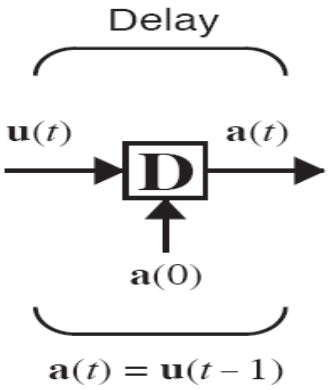
❑Rosenblatt's single layer perceptron is trained as follow:

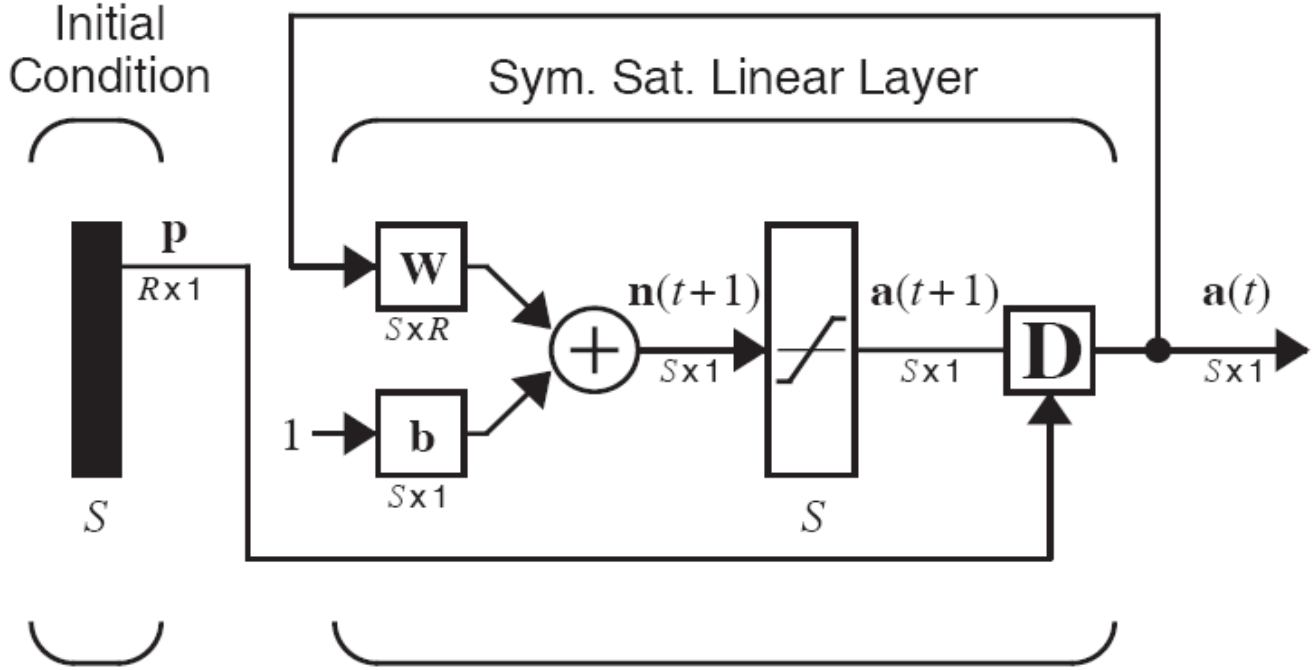
1. Randomly initialize all the networks weights.
2. Apply inputs and find outputs (feedforward).
3. compute the errors.
4. Update each weight as

$$w_{ij}(k+1) = w_{ij}(k) + \eta\, p_i(k) e_j(k)$$

5. Repeat steps 2 to 4 until the errors reach the satisfictory level.

Recurrent Network



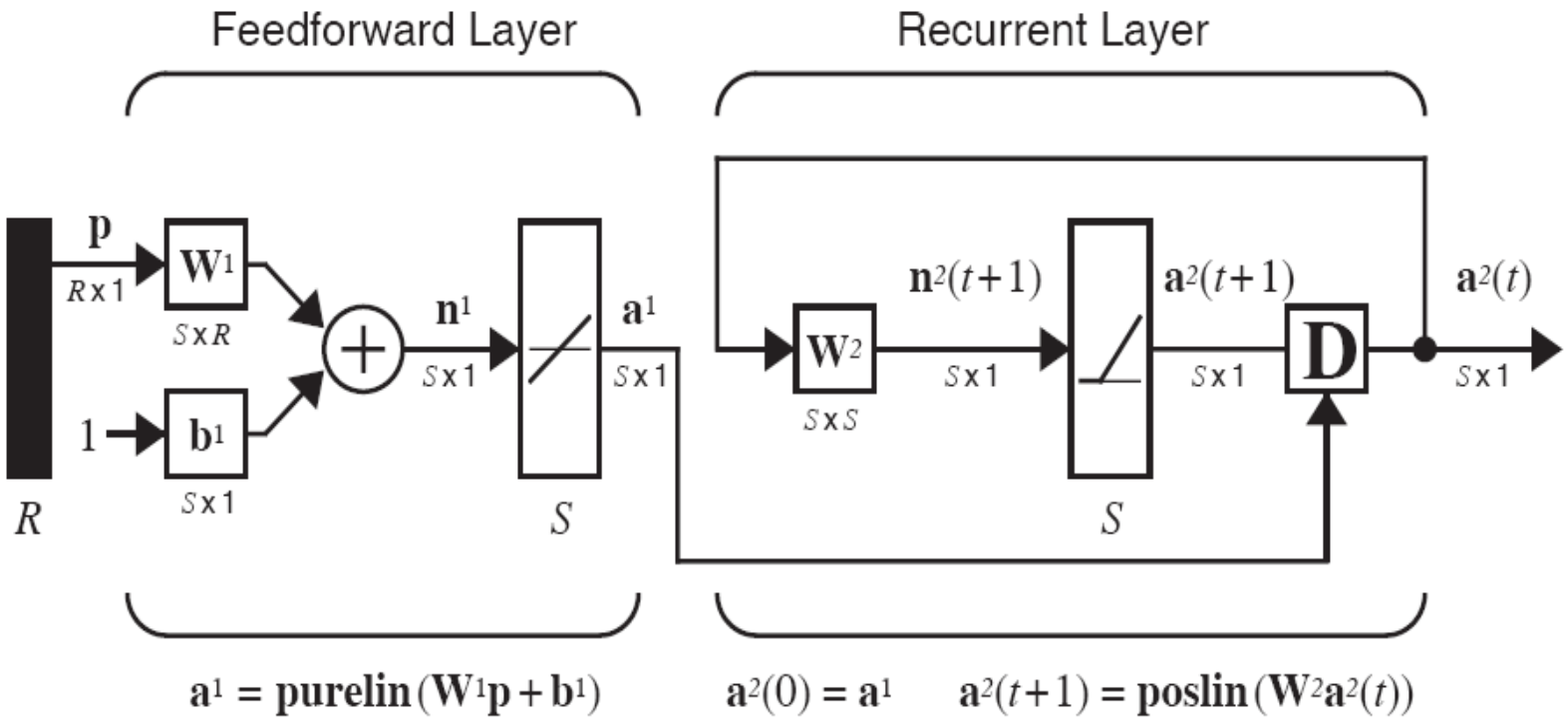


$$\mathbf{a}(0) = \mathbf{p} \qquad \mathbf{a}(t+1) = \mathbf{satlin}(\mathbf{W}\mathbf{a}(t)+\mathbf{b})$$

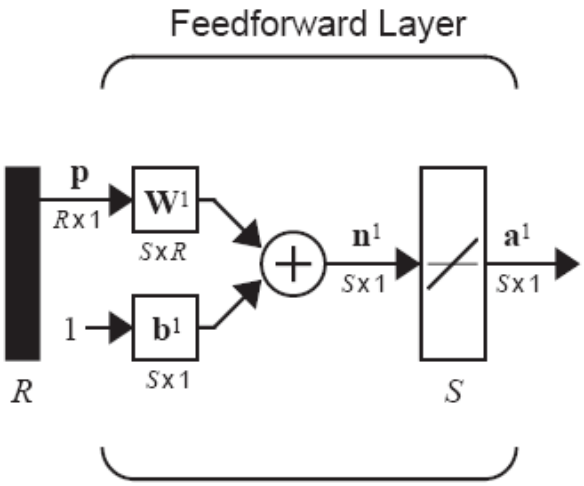
$$\mathbf{a}(1) = \mathbf{satlins}(\mathbf{W}\mathbf{a}(0)+\mathbf{b}) = \mathbf{satlins}(\mathbf{W}\mathbf{p}+\mathbf{b})$$

$$\mathbf{a}(2) = \mathbf{satlins}(\mathbf{W}\mathbf{a}(1)+\mathbf{b})$$

Hamming Network



Feed Forward Layer



$$\mathbf{a}^1 = \text{purelin}(\mathbf{W}^1\mathbf{p} + \mathbf{b}^1)$$

For Banana/Apple Recognition

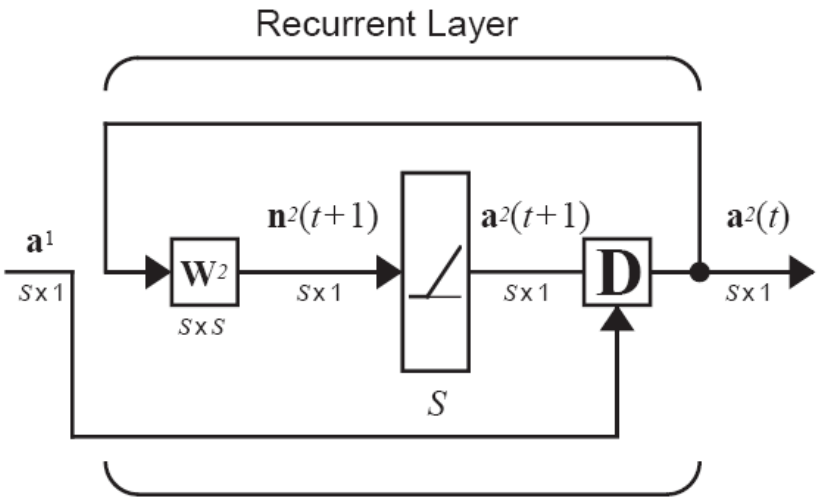
$$S = 2$$

$$\mathbf{W}^1 = \begin{bmatrix} \mathbf{p}_1^T \\ \mathbf{p}_2^T \end{bmatrix} = \begin{bmatrix} -1 & 1 & -1 \\ 1 & 1 & -1 \end{bmatrix}$$

$$\mathbf{b}^1 = \begin{bmatrix} R \\ R \end{bmatrix} = \begin{bmatrix} 3 \\ 3 \end{bmatrix}$$

$$\mathbf{a}^1 = \mathbf{W}^1\mathbf{p} + \mathbf{b}^1 = \begin{bmatrix} \mathbf{p}_1^T \\ \mathbf{p}_2^T \end{bmatrix}\mathbf{p} + \begin{bmatrix} 3 \\ 3 \end{bmatrix} = \begin{bmatrix} \mathbf{p}_1^T\mathbf{p} + 3 \\ \mathbf{p}_2^T\mathbf{p} + 3 \end{bmatrix}$$

Recurrent Layer



$$\mathbf{W}^2 = \begin{bmatrix} 1 & -\varepsilon \\ -\varepsilon & 1 \end{bmatrix} \qquad \varepsilon < \frac{1}{S-1}$$

$$\mathbf{a}^2(t+1) = \mathbf{poslin}\left(\begin{bmatrix} 1 & -\varepsilon \\ -\varepsilon & 1 \end{bmatrix}\mathbf{a}^2(t)\right) = \mathbf{poslin}\left(\begin{bmatrix} a_1^2(t) - \varepsilon a_2^2(t) \\ a_2^2(t) - \varepsilon a_1^2(t) \end{bmatrix}\right)$$

Hamming Operation

First Layer

Input (Rough Banana)

$$\mathbf{p} = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}$$

$$\mathbf{a}^1 = \begin{bmatrix} -1 & 1 & -1 \\ 1 & 1 & -1 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix} + \begin{bmatrix} 3 \\ 3 \end{bmatrix} = \begin{bmatrix} (1+3) \\ (-1+3) \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \end{bmatrix}$$

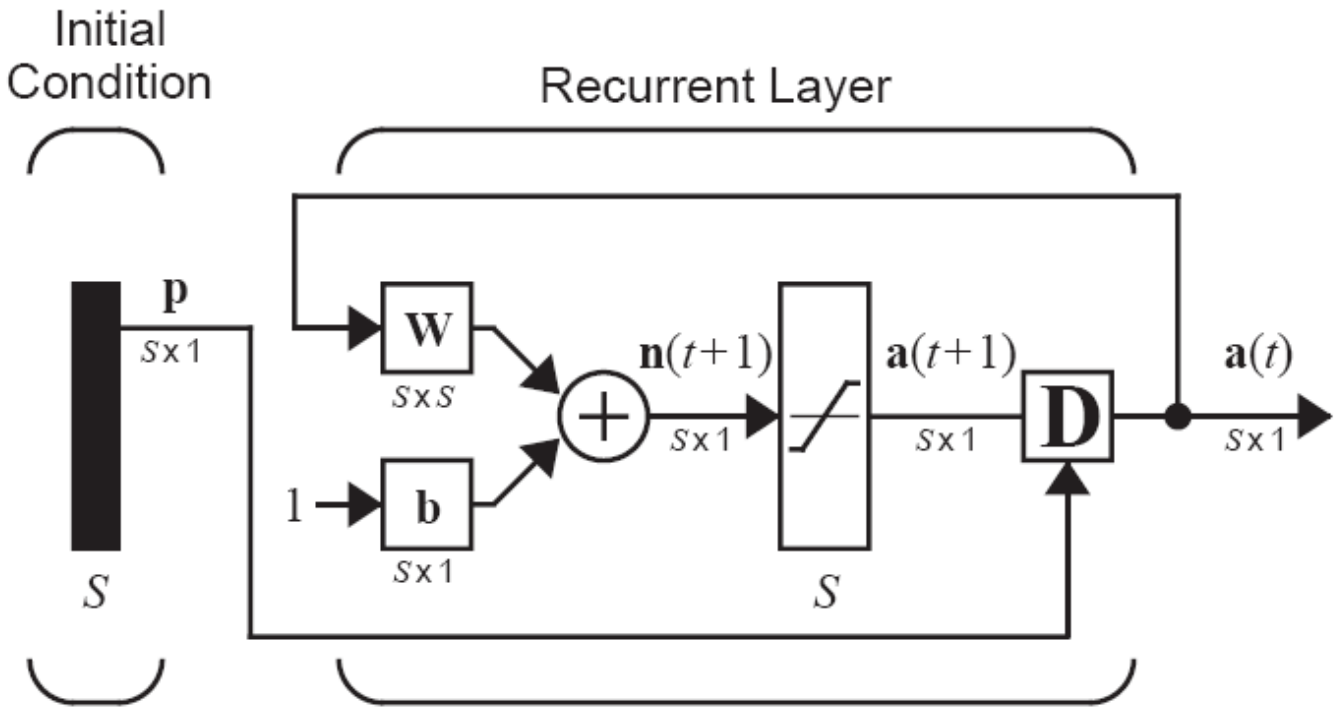
Hamming Operation

Second Layer

$$\mathbf{a}^2(1) = \mathbf{poslin}(\mathbf{W}^2\mathbf{a}^2(0)) = \begin{cases} \mathbf{poslin}\left(\begin{bmatrix} 1 & -0.5 \\ -0.5 & 1 \end{bmatrix}\begin{bmatrix} 4 \\ 2 \end{bmatrix}\right) \\ \mathbf{poslin}\left(\begin{bmatrix} 3 \\ 0 \end{bmatrix}\right) = \begin{bmatrix} 3 \\ 0 \end{bmatrix} \end{cases}$$

$$\mathbf{a}^2(2) = \mathbf{poslin}(\mathbf{W}^2\mathbf{a}^2(1)) = \begin{cases} \mathbf{poslin}\left(\begin{bmatrix} 1 & -0.5 \\ -0.5 & 1 \end{bmatrix}\begin{bmatrix} 3 \\ 0 \end{bmatrix}\right) \\ \mathbf{poslin}\left(\begin{bmatrix} 3 \\ -1.5 \end{bmatrix}\right) = \begin{bmatrix} 3 \\ 0 \end{bmatrix} \end{cases}$$

Hopfield Network



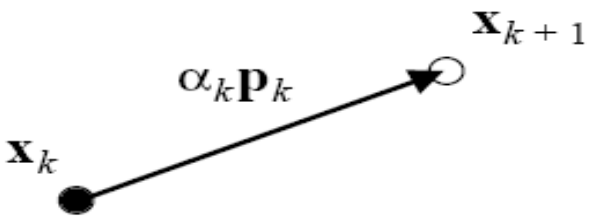
Performance Optimization
Gradient based methods

Basic Optimization Algorithm

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$$

or

$$\Delta \mathbf{x}_k = (\mathbf{x}_{k+1} - \mathbf{x}_k) = \alpha_k \mathbf{p}_k$$



$\mathbf{p}_k$ - Search Direction

α_k - Learning Rate

Steepest Descent (first order Taylor expansion)

Choose the next step so that the function decreases:

$$F(\mathbf{x}_{k+1}) < F(\mathbf{x}_k)$$

For small changes in $\mathbf{x}$ we can approximate $F(\mathbf{x})$:

$$F(\mathbf{x}_{k+1}) = F(\mathbf{x}_k + \Delta\mathbf{x}_k) \approx F(\mathbf{x}_k) + \mathbf{g}_k^T \Delta\mathbf{x}_k$$

where

$$\mathbf{g}_k \equiv \nabla F(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}_k}$$

If we want the function to decrease:

$$\mathbf{g}_k^T \Delta\mathbf{x}_k = \alpha_k \mathbf{g}_k^T \mathbf{p}_k < 0$$

We can maximize the decrease by choosing:

$$\mathbf{p}_k = -\mathbf{g}_k$$

$$\boxed{\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{g}_k}$$

Example

$$F(\mathbf{x}) = x_1^2 + 2x_1x_2 + 2x_2^2 + x_1$$

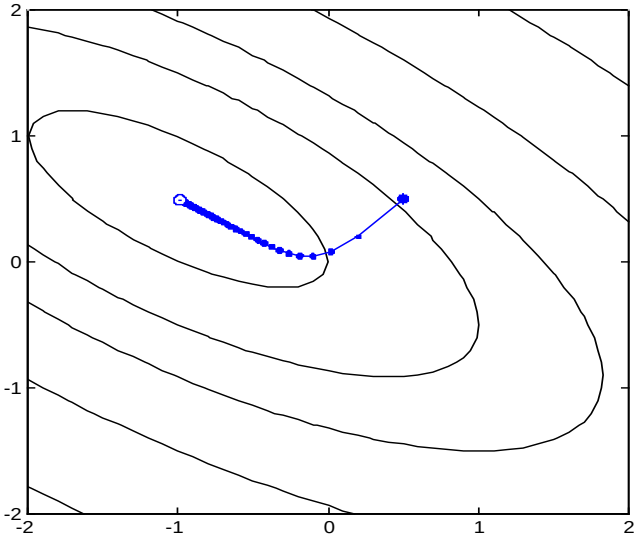
$$\mathbf{x}_0 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \qquad \alpha = 0.1$$

$$\nabla F(\mathbf{x}) = \begin{bmatrix} \frac{\partial}{\partial x_1} F(\mathbf{x}) \\ \frac{\partial}{\partial x_2} F(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} 2x_1 + 2x_2 + 1 \\ 2x_1 + 4x_2 \end{bmatrix} \qquad \mathbf{g}_0 = \nabla F(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}_0} = \begin{bmatrix} 3 \\ 3 \end{bmatrix}$$

$$\mathbf{x}_1 = \mathbf{x}_0 - \alpha \mathbf{g}_0 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} - 0.1 \begin{bmatrix} 3 \\ 3 \end{bmatrix} = \begin{bmatrix} 0.2 \\ 0.2 \end{bmatrix}$$

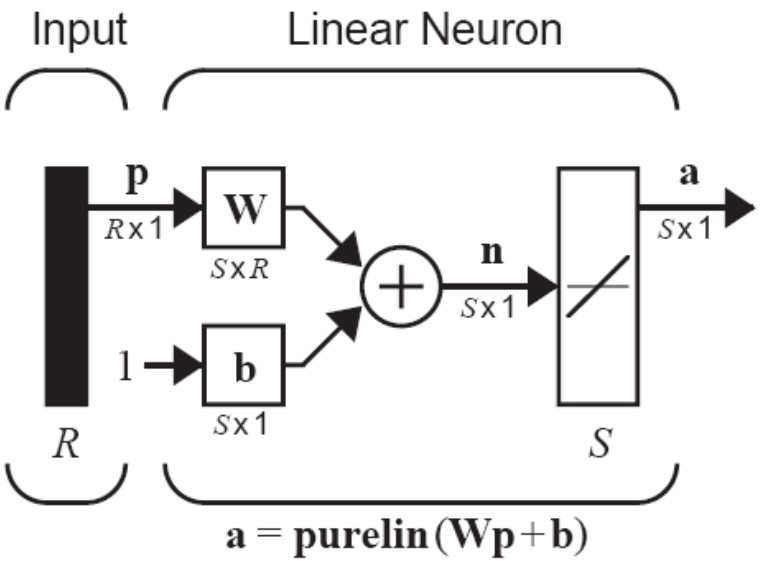
$$\mathbf{x}_2 = \mathbf{x}_1 - \alpha \mathbf{g}_1 = \begin{bmatrix} 0.2 \\ 0.2 \end{bmatrix} - 0.1 \begin{bmatrix} 1.8 \\ 1.2 \end{bmatrix} = \begin{bmatrix} 0.02 \\ 0.08 \end{bmatrix}$$

Plot



LMS or Widrow- Hoff learning

- First introduce ADALINE (ADAptive LInear NEuron) Network

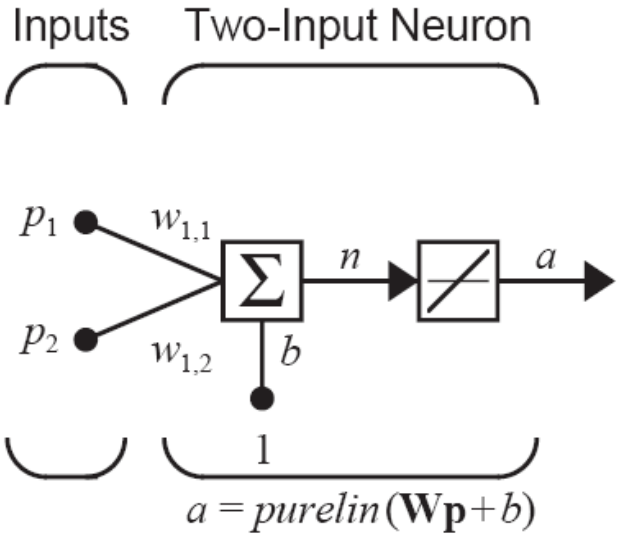


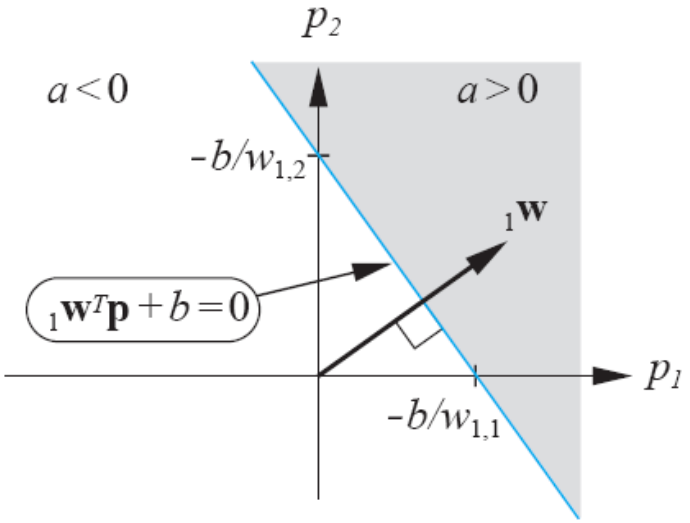
$$\mathbf{a} = \mathbf{purelin}(\mathbf{Wp}+\mathbf{b}) = \mathbf{Wp}+\mathbf{b}$$

$$a_i = purelin(n_i) = purelin({}_i\mathbf{w}^T\mathbf{p} + b_i) = {}_i\mathbf{w}^T\mathbf{p} + b_i \qquad {}_i\mathbf{w} = \begin{bmatrix} w_{i,1} \\ w_{i,2} \\ \vdots \\ w_{i,R} \end{bmatrix}$$

LMS or Widrow- Hoff learning
or
Delta Rule

- ADALINE network same basic structure as the perceptron network





$$a = purelin(n) = purelin({}_1\mathbf{w}^T\mathbf{p} + b) = {}_1\mathbf{w}^T\mathbf{p} + b$$

$$a = {}_1\mathbf{w}^T\mathbf{p} + b = w_{1,1}p_1 + w_{1,2}p_2 + b$$

Approximate Steepest Descent

Approximate mean square error (one sample):

$$\hat{F}(\mathbf{x}) = (t(k) - a(k))^2 = e^2(k)$$

Approximate (stochastic) gradient:

$$\hat{\nabla} F(\mathbf{x}) = \nabla e^2(k)$$

$$[\nabla e^2(k)]_j = \frac{\partial e^2(k)}{\partial w_{1,j}} = 2e(k)\frac{\partial e(k)}{\partial w_{1,j}} \qquad j = 1, 2, \ldots, R$$

$$[\nabla e^2(k)]_{R+1} = \frac{\partial e^2(k)}{\partial b} = 2e(k)\frac{\partial e(k)}{\partial b}$$

Approximate Gradient Calculation

$$\frac{\partial e(k)}{\partial w_{1,j}} = \frac{\partial [t(k) - a(k)]}{\partial w_{1,j}} = \frac{\partial}{\partial w_{1,j}} [t(k) - ({}_1\mathbf{w}^T \mathbf{p}(k) + b)]$$

$$\frac{\partial e(k)}{\partial w_{1,j}} = \frac{\partial}{\partial w_{1,j}} \left[t(k) - \left(\sum_{i=1}^{R} w_{1,i} p_i(k) + b \right) \right]$$

$$\frac{\partial e(k)}{\partial w_{1,j}} = -p_j(k) \qquad \frac{\partial e(k)}{\partial b} = -1$$

$$\hat{\nabla} F(\mathbf{x}) = \nabla e^2(k) = -2e(k)\mathbf{z}(k)$$

LMS Algorithm

This algorithm inspire from steepest descent algorithm

$$\mathbf{x}_{k+1} = \mathbf{x}_k + 2\alpha e(k)\mathbf{z}(k)$$

$$_1\mathbf{w}(k+1) = {}_1\mathbf{w}(k) + 2\alpha e(k)\mathbf{p}(k)$$

$$b(k+1) = b(k) + 2\alpha e(k)$$

Multiple-Neuron Case

$$_i\mathbf{w}(k+1) = {}_i\mathbf{w}(k) + 2\alpha e_i(k)\mathbf{p}(k)$$

$$b_i(k+1) = b_i(k) + 2\alpha e_i(k)$$

Matrix Form:

$$\mathbf{W}(k+1) = \mathbf{W}(k) + 2\alpha\mathbf{e}(k)\mathbf{p}^T(k)$$

$$\mathbf{b}(k+1) = \mathbf{b}(k) + 2\alpha\mathbf{e}(k)$$

Difference between perceptron learning and LMS learning

- DERIVATIVE
- Linear activation function has derivative

  but
- sign (bipolar, unipolar) has not derivative

Grossberg learning (associated learning)

- Sometimes known as instar and outstar training
- Updating rule:

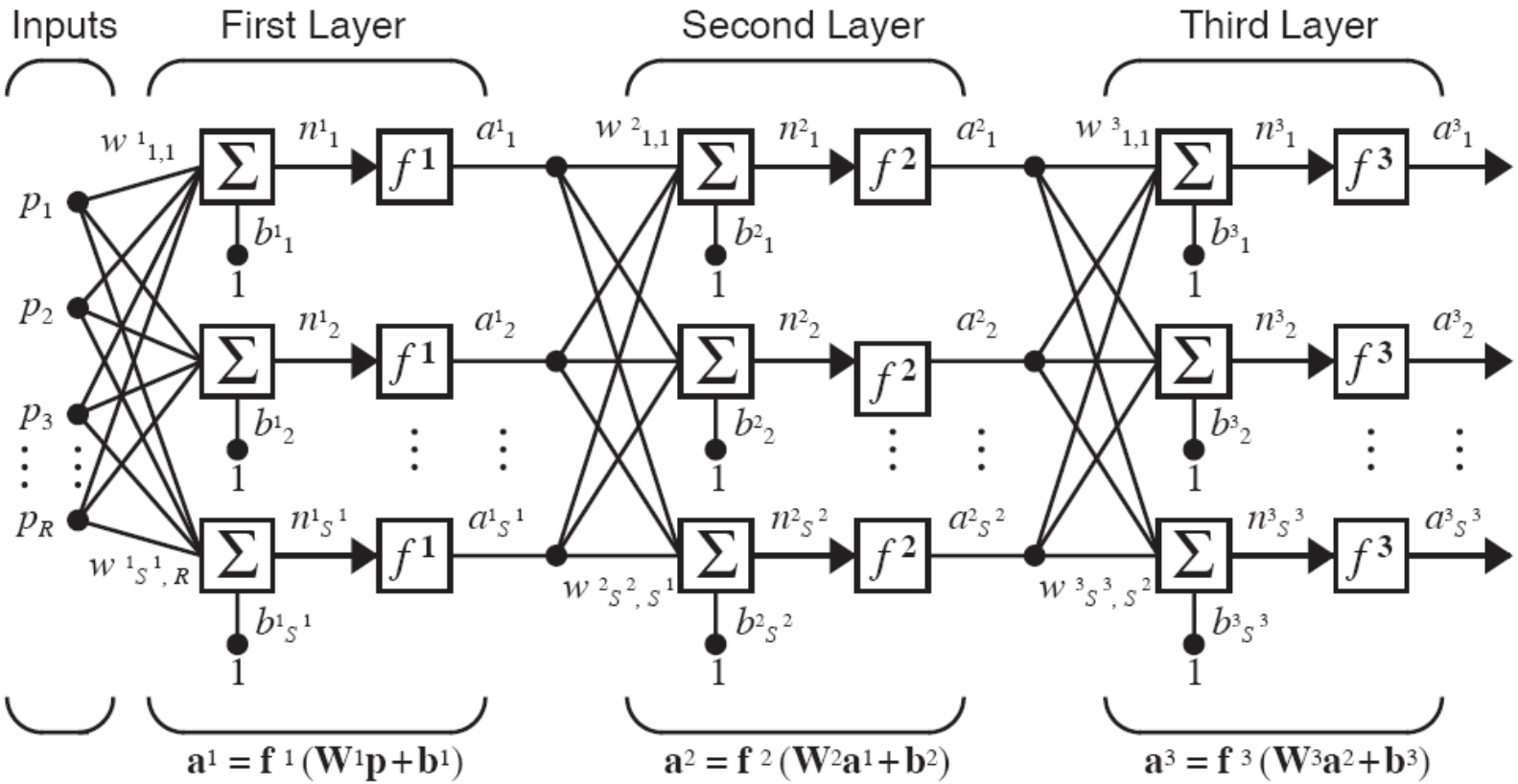
$$w_i(k+1) = w_i(k) + \eta\left[x_i(k) - w_i(k)\right]$$

- Where x_i could be the desired input values (instar training, example: clustering) or the desired output values (outstar) depending on network structure.
- Grossberg network (use Hagan to more details)

First order gradient method

Back propagation

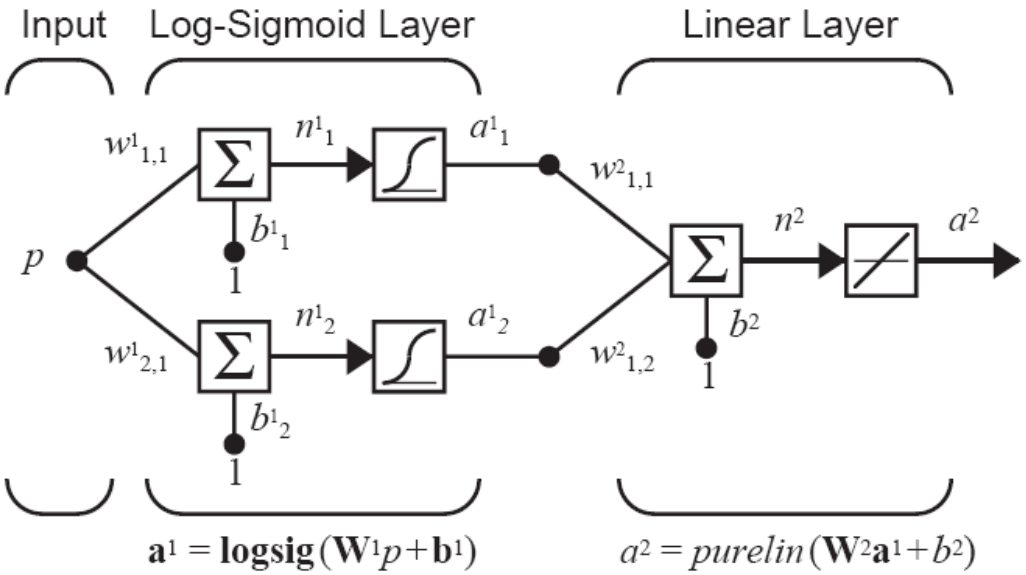
Multilayer Perceptron



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

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

Function Approximation Example



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

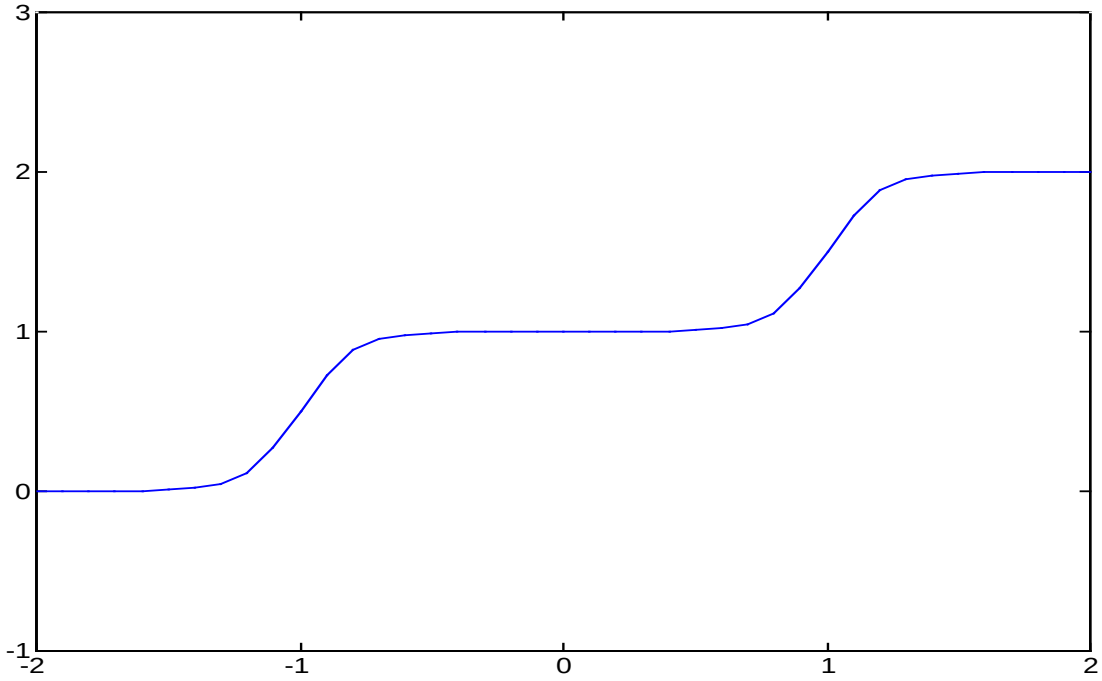
$$f^2(n) = n$$

Nominal Parameter Values

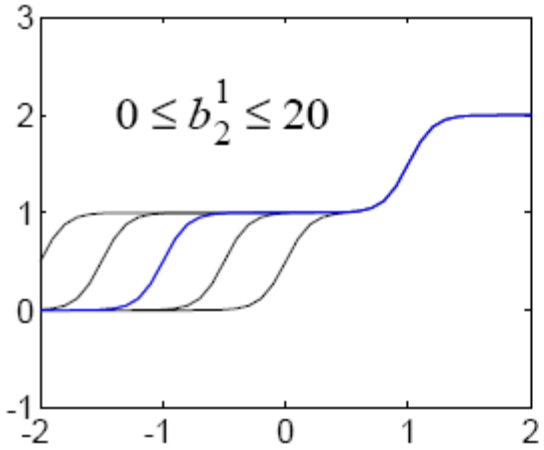
$$w^1_{1,1} = 10 \qquad w^1_{2,1} = 10 \qquad b^1_1 = -10 \qquad b^1_2 = 10$$

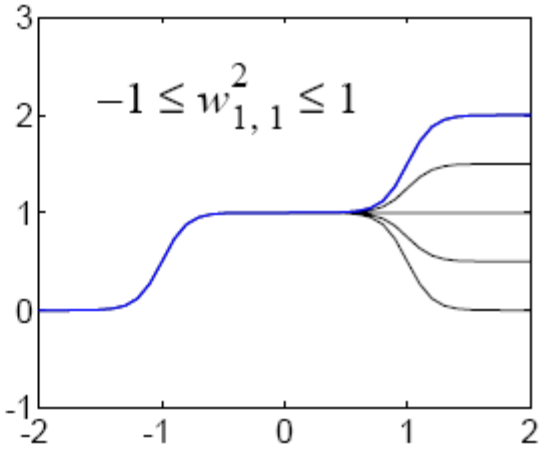
$$w^2_{1,1} = 1 \qquad w^2_{1,2} = 1 \qquad b^2 = 0$$

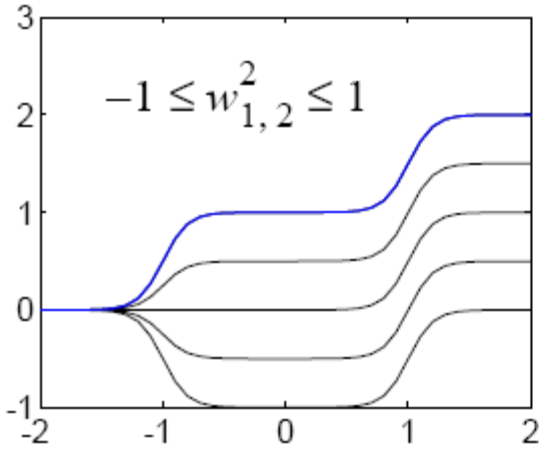
Nominal Response

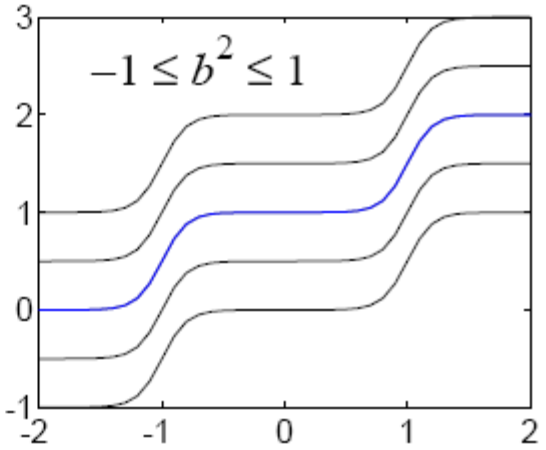


Parameter Variations

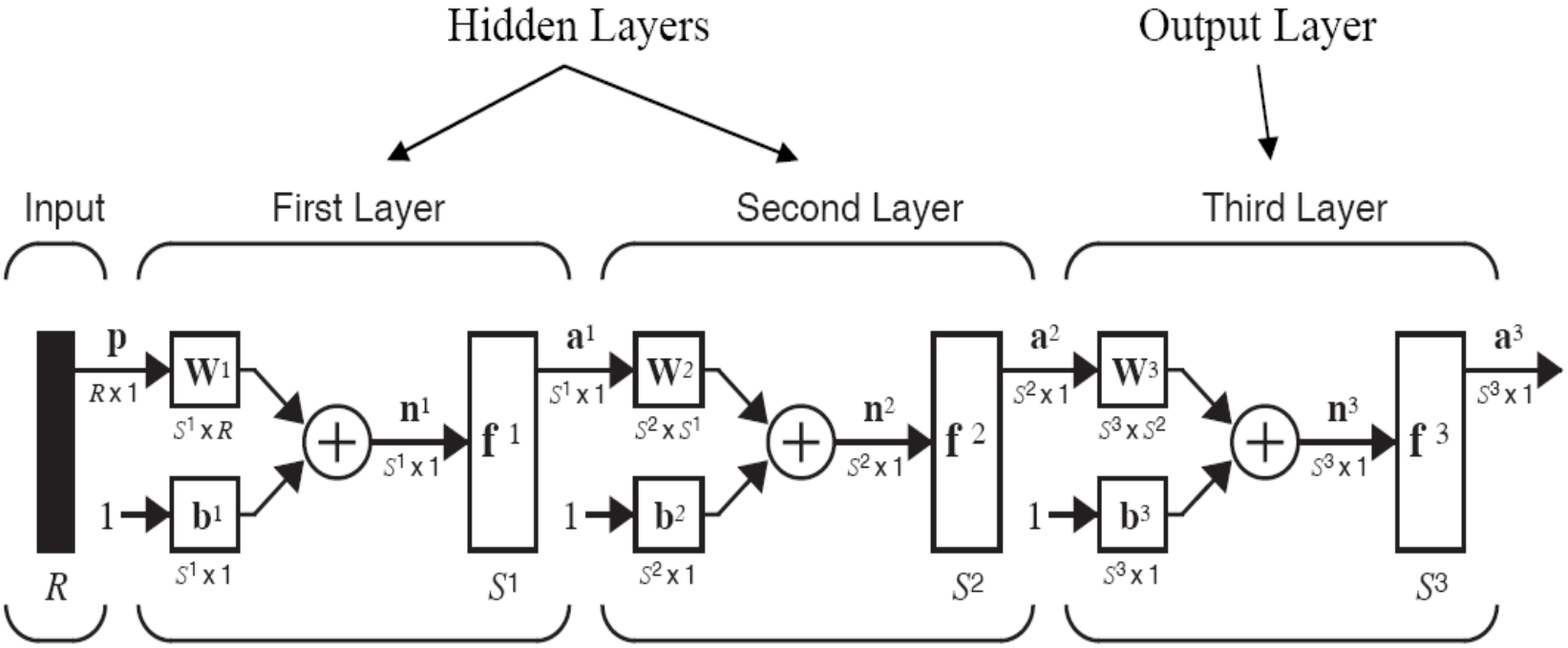








Multilayer Network



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

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

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

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

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

Performance Index

Training Set

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

Mean Square Error

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

Vector Case

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

Approximate Mean Square Error (Single Sample)

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

Approximate Steepest Descent

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

Chain Rule

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

Example

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

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

Application to Gradient Calculation

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

Gradient Calculation

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

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

Sensitivity

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

Gradient

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

Steepest Descent

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

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

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

Next Step: Compute the Sensitivities (Backpropagation)

Jacobian Matrix

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

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

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

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

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

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

Backpropagation (Sensitivities)

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

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

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

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

Initialization (Last Layer)

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

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

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

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

Summary

Forward Propagation

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

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

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

Backpropagation

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

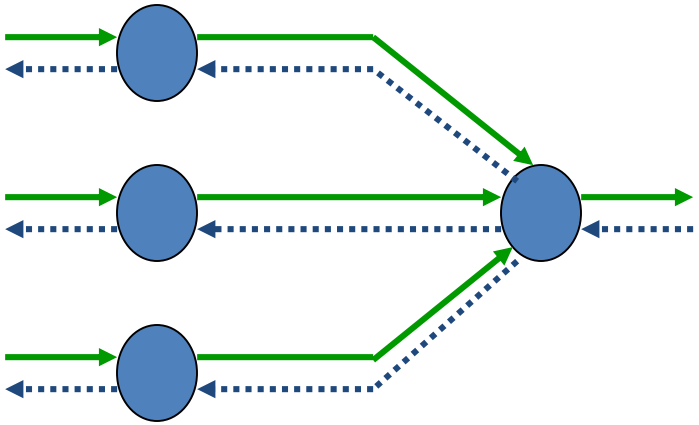
$$\mathbf{s}^m = \dot{\mathbf{F}}^m(\mathbf{n}^m)(\mathbf{W}^{m+1})^T\mathbf{s}^{m+1} \qquad m = M-1, \ldots, 2, 1$$

Weight Update

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

Summary

- Back-propagation training algorithm

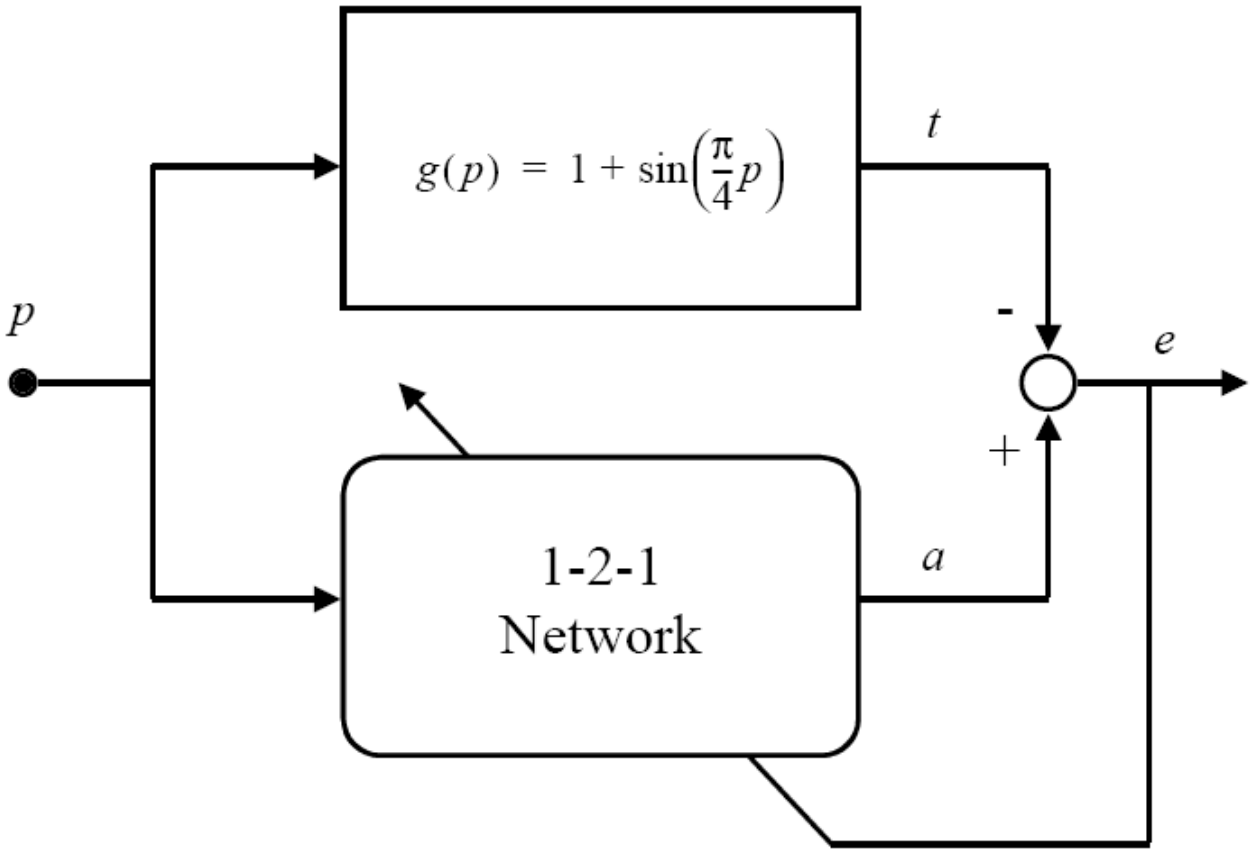


Network activation
Forward Step

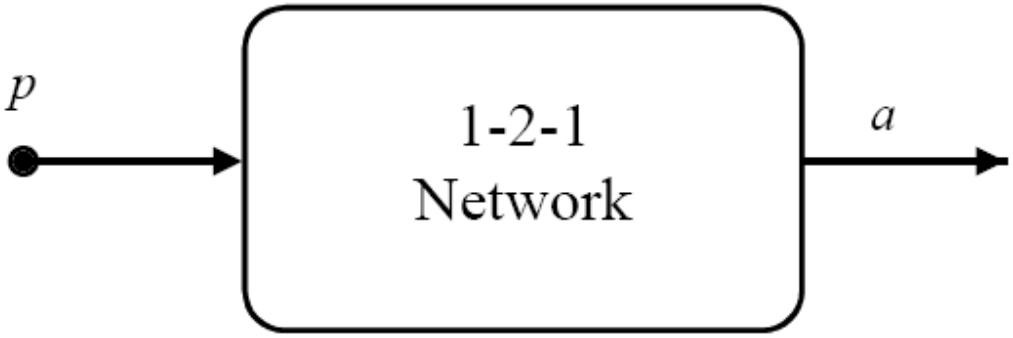
Error propagation
Backward Step

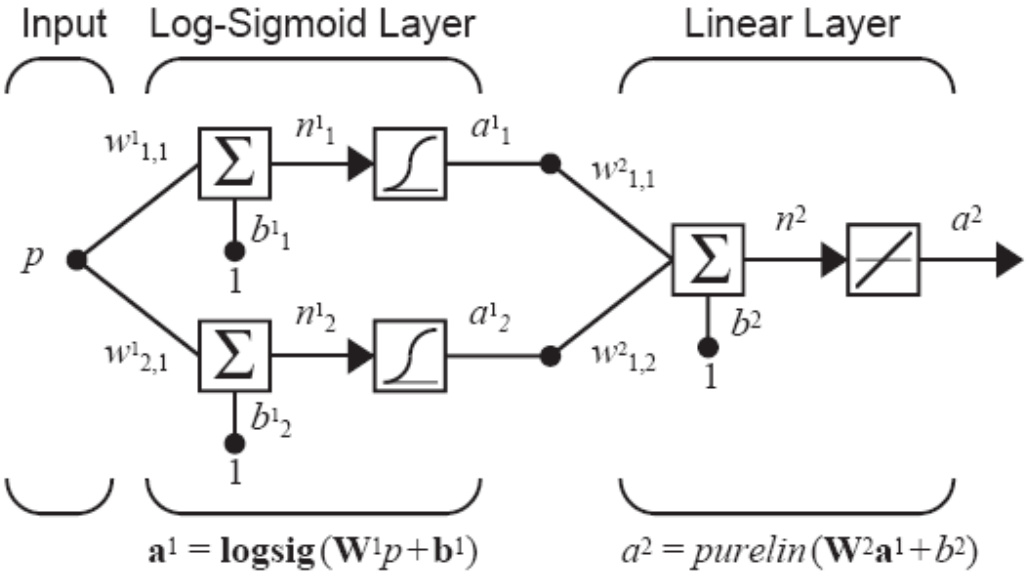
- Backprop adjusts the weights of the NN in order to minimize the network total mean squared error.

Example: Function Approximation



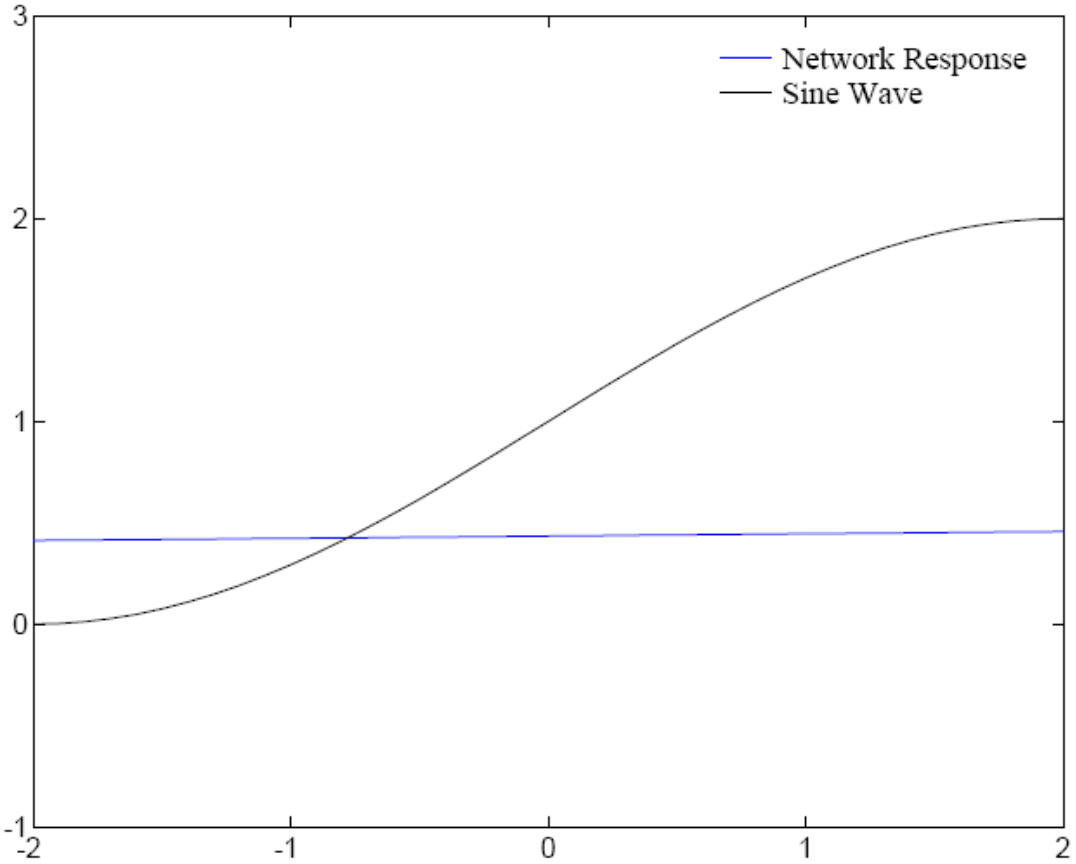
Network


p
1-2-1
Network
a


Input
Log-Sigmoid Layer
Linear Layer
p
1
1
1
$w^1_{1,1}$
$w^1_{2,1}$
b^1_1
b^1_2
n^1_1
n^1_2
a^1_1
a^1_2
$w^2_{1,1}$
$w^2_{1,2}$
b^2
n^2
a^2
$\mathbf{a}^1 = \mathbf{logsig}(\mathbf{W}^1 p + \mathbf{b}^1)$
$a^2 = purelin(\mathbf{W}^2 \mathbf{a}^1 + b^2)$

Initial Conditions

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



Forward Propagation

$$a^0 = p = 1$$

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

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

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

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

Transfer Function Derivatives

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

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

Backpropagation(BP)

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

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

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

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

Weights Update

$$\alpha = 0.1$$

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

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

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

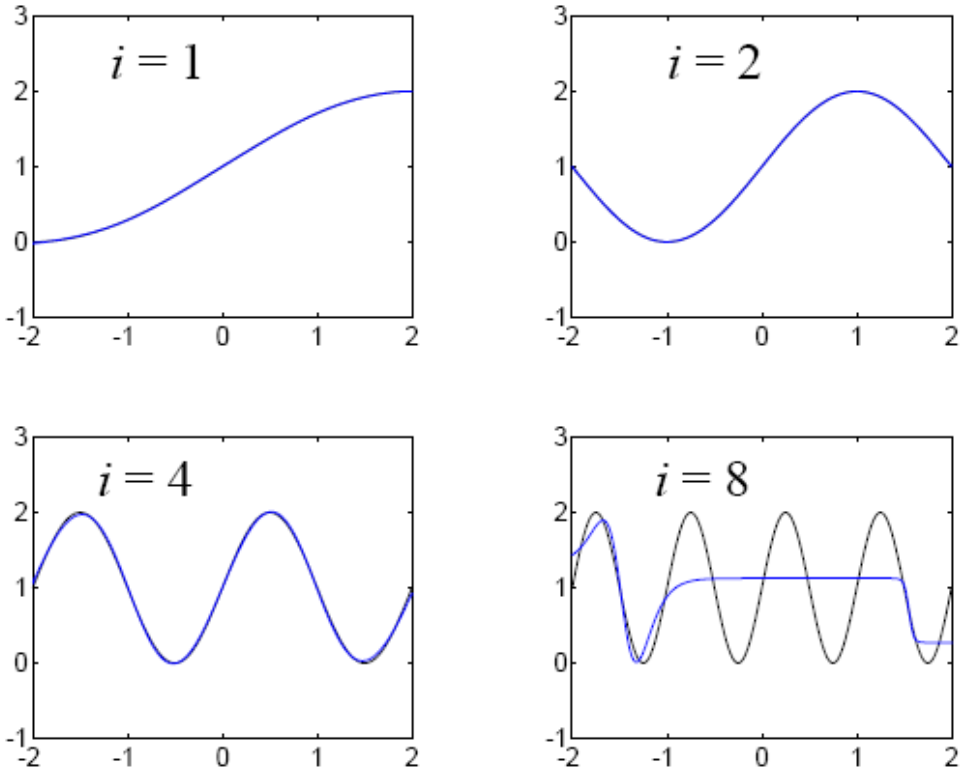
$$\mathbf{W}^1(1) = \mathbf{W}^1(0) - \alpha \mathbf{s}^1 (\mathbf{a}^0)^T = \begin{bmatrix} -0.27 \\ -0.41 \end{bmatrix} - 0.1 \begin{bmatrix} -0.0495 \\ 0.0997 \end{bmatrix} \begin{bmatrix} 1 \end{bmatrix} = \begin{bmatrix} -0.265 \\ -0.420 \end{bmatrix}$$

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

Choice of Architecture

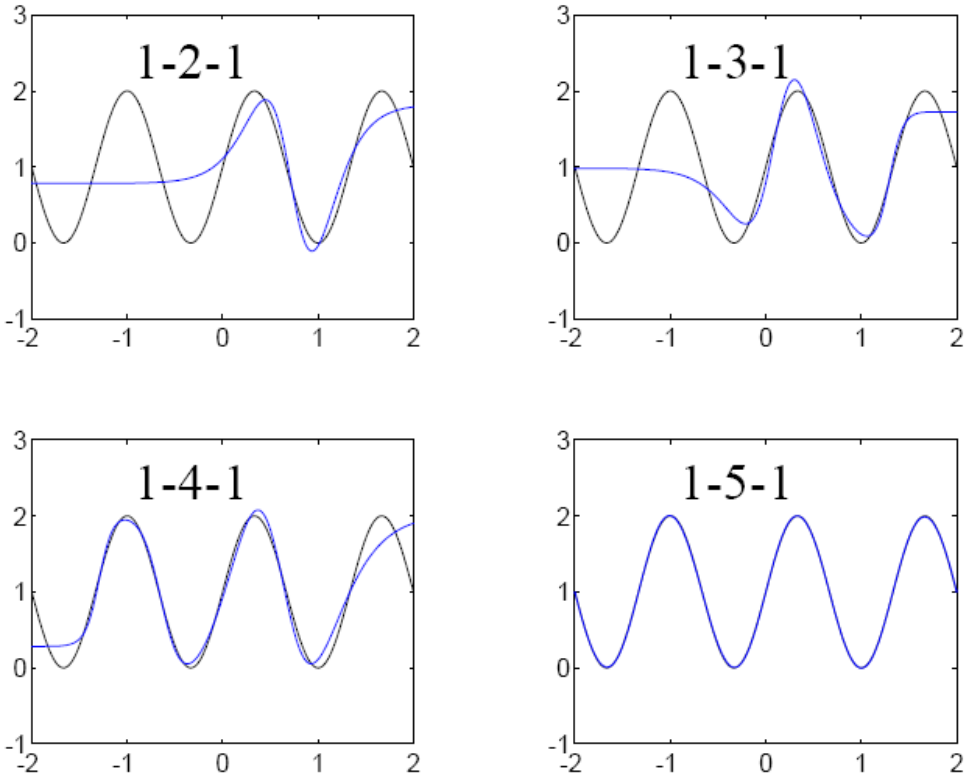
$$g(p) = 1 + \sin\left(\frac{i\pi}{4}p\right)$$

1-3-1 Network



Choice of Network Architecture

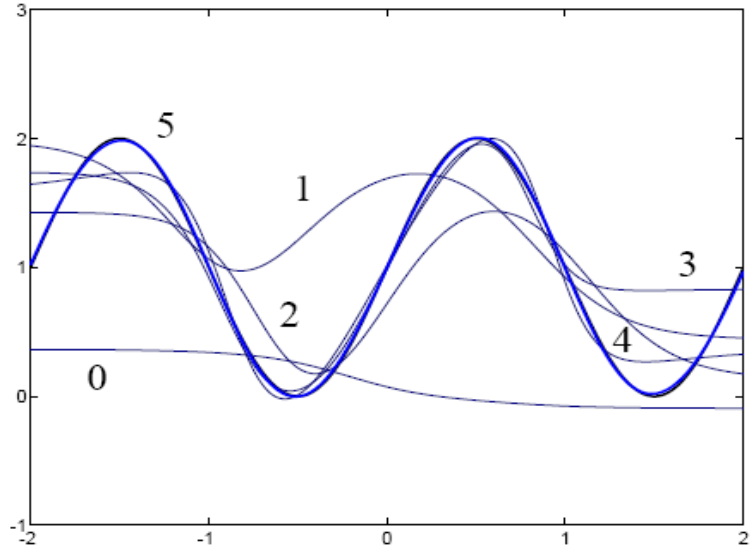
$$g(p) = 1 + \sin\left(\frac{6\pi}{4}p\right)$$

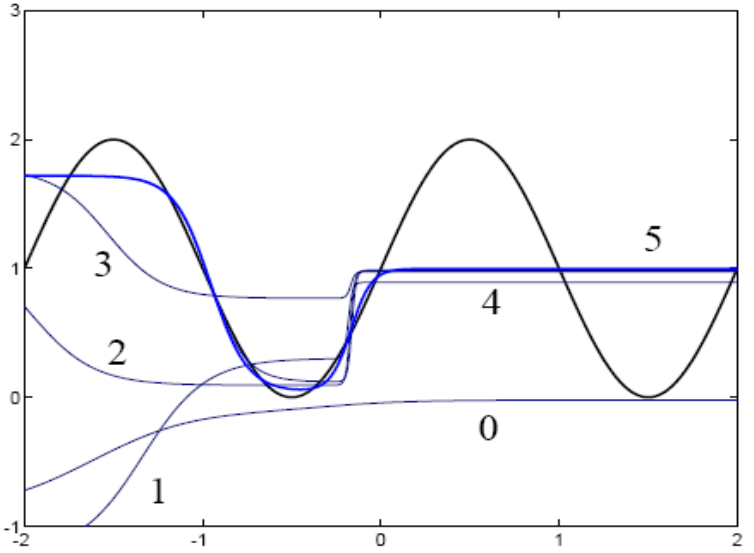


Convergence

Global minimum (left) local minimum (right)

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

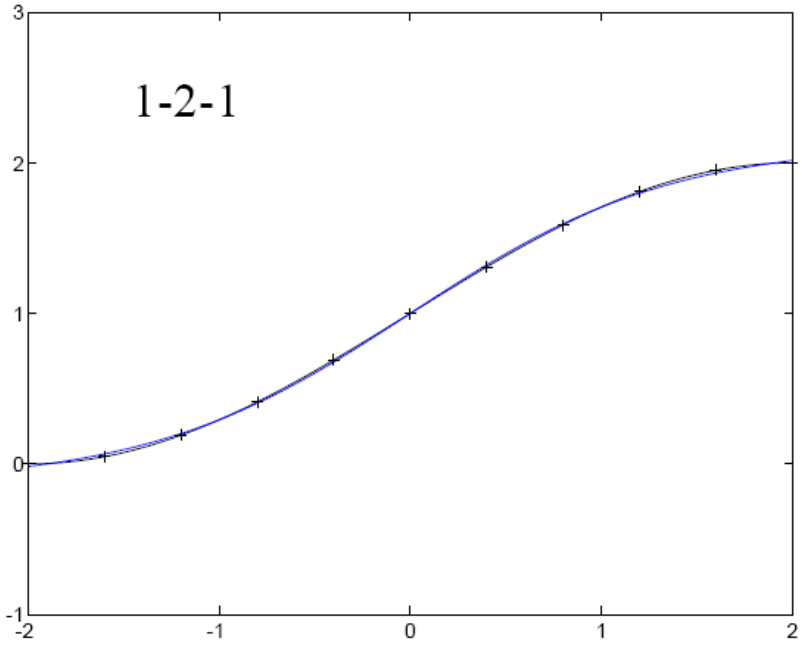


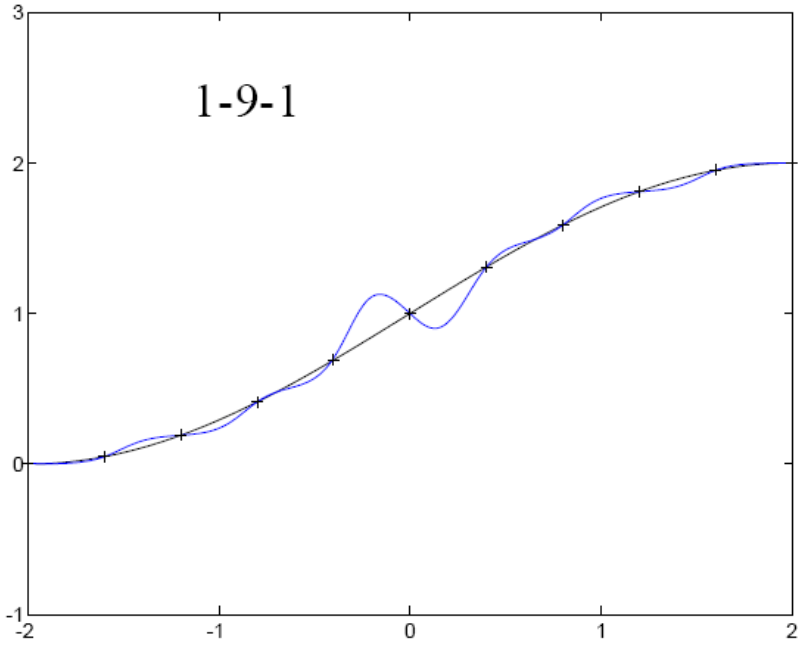


Generalization

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

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





Disadvantage of BP algorithm

- Slow convergence speed
- Sensitivity to initial conditions
- Trapped in local minima
- Instability if learning rate is too large

- Note: despite above disadvantages, it is popularly used in control community. There are numerous extensions to improve BP algorithm.

Improved BP algorithms

- First Order Gradient Method
- Second Order Gradient Method

فصل پنجم :

بهبود الگوریتم پس انتشار خطا

Improved BP algorithms

Improved BP algorithms

First Order Gradient Method

1. BP with momentum
2. Delta- bar- delta
3. Decoupled momentum
4. RProp
5. Adaptive BP
6. Trianary BP
7. BP with adaptive gain
8. Extended BP

1- BP with momentum (BPM)

The basic improvement to BP (Rumelhart 1986)

$$W(k) = W(k-1) + (-\eta \frac{\partial E(k)}{\partial W(k-1)}) + \alpha \Delta W(k-1)$$

$$\Delta \mathbf{W}^m(k) = \gamma \Delta \mathbf{W}^m(k-1) - (1-\gamma)\alpha \mathbf{s}^m (\mathbf{a}^{m-1})^T$$

$$\Delta \mathbf{b}^m(k) = \gamma \Delta \mathbf{b}^m(k-1) - (1-\gamma)\alpha \mathbf{s}^m$$

Momentum factor alpha selected between zero and one

Adding momentum improves the convergence speed and helps network from being trapped in a local minimum.

Modification form:

$$W(k) = W(k-1) + (-\eta \frac{\partial E(k)}{\partial W(k-1)}) + \alpha \Delta W(k-1) + \beta \Delta W(k-2)$$

- Proposed by nagata 1990
- Beta is a constant value decided by user
- Nagata claimed that beta term reduce the possibility of the network being trapped in the local minimum
- This seems beta repeating alpha rule again!!! But not clear

2- Delta-bar-delta (DBD)

- Use adaptive learning rate to speed up the convergence
- The adaptive rate adopted base on local optimization
- Use gradient descent for the search direction, and use individual step sizes for each weight

Let $\eta_{ij}(k)$ denote the learning rate for the weight $w_{ij}(k)$, then

$$\eta_{ij}(k+1) = \eta_{ij}(k) + \Delta\eta_{ij}(k)$$

$$\Delta\eta_{ij}(k) = \begin{cases} \kappa & \bar{\psi}_{ij}(k-1)\psi_{ij}(k) > 0 \\ -\beta\eta_{ij}(k) & \bar{\psi}_{ij}(k-1)\psi_{ij}(k) < 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\psi_{ij}(k) = \frac{\partial E(k)}{\partial w_{ij}}$$

$$\bar{\psi}_{ij}(k) = (1-\varepsilon)\psi_{ij}(k-1) + \varepsilon\bar{\psi}_{ij}(k-1)$$

3- RProp

- Jervis and Fitzgerald (1993)
- $\eta_{\max}$ an $\eta_{\min}$ limit the size of the step

$$\eta_{ij}(k) = \begin{cases} \min[1.2\eta_{ij}(k-1), \eta_{\max}] & \bar{\psi}_{ij}(k-1)\psi_{ij}(k) > 0 \\ \max[0.5\eta_{ij}(k-1), \eta_{\min}] & \bar{\psi}_{ij}(k-1)\psi_{ij}(k) < 0 \\ \eta_{ij}(k-1) & \text{otherwise} \end{cases}$$

$$\Delta w_{ij}(k) = \begin{cases} -\text{sign}(\psi_{ij})\eta_{ij}(k) & \bar{\psi}_{ij}(k-1)\psi_{ij}(k) > 0 \\ 0 & \text{otherwise} \end{cases}$$

Improved BP algorithms

Second Order Gradient Method

1. Newton
2. Gauss-Newton
3. Levenberg-Marquardt
4. Quickprop
5. Conjugate gradient descent
6. Broyde –Fletcher-Goldfab-Shanno

Performance Surfaces

- Taylor Series Expansion

$$F(x) = F(x^*) + \frac{d}{dx}F(x)\Big|_{x=x^*}(x-x^*)$$

$$+ \frac{1}{2}\frac{d^2}{dx^2}F(x)\Big|_{x=x^*}(x-x^*)^2 + \cdots$$

$$+ \frac{1}{n!}\frac{d^n}{dx^n}F(x)\Big|_{x=x^*}(x-x^*)^n + \cdots$$

Example

$$F(x) = e^{-x}$$

Taylor series of $F(x)$ about $x^* = 0$:

$$F(x) = e^{-x} = e^{-0} - e^{-0}(x-0) + \frac{1}{2}e^{-0}(x-0)^2 - \frac{1}{6}e^{-0}(x-0)^3 + \ldots$$

$$F(x) = 1 - x + \frac{1}{2}x^2 - \frac{1}{6}x^3 + \ldots$$

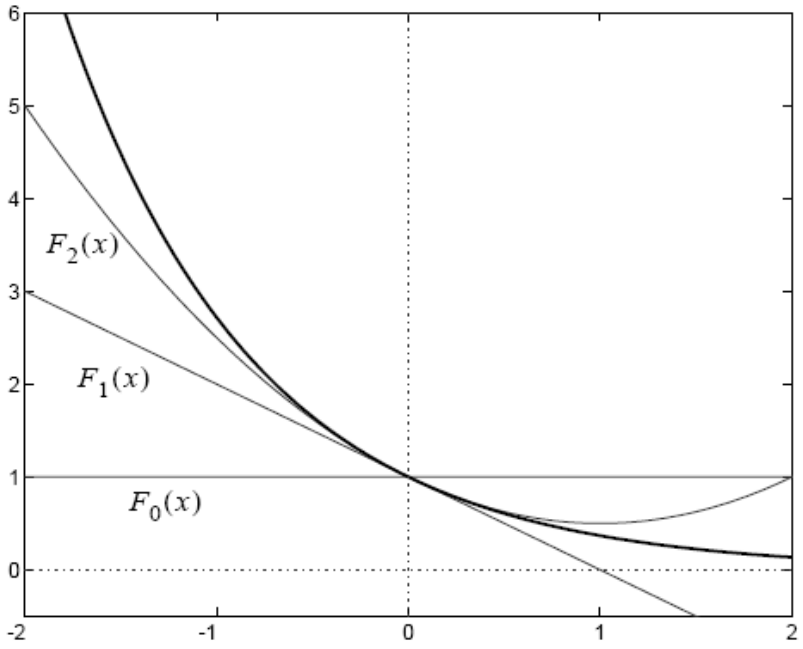
Taylor series approximations:

$$F(x) \approx F_0(x) = 1$$

$$F(x) \approx F_1(x) = 1 - x$$

$$F(x) \approx F_2(x) = 1 - x + \frac{1}{2}x^2$$

Plot of Approximations



Vector Case

$$F(\mathbf{x}) = F(x_1, x_2, \ldots, x_n)$$

$$F(\mathbf{x}) = F(\mathbf{x}^*) + \frac{\partial}{\partial x_1} F(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}^*} (x_1 - x_1^*) + \frac{\partial}{\partial x_2} F(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}^*} (x_2 - x_2^*)$$

$$+ \cdots + \frac{\partial}{\partial x_n} F(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}^*} (x_n - x_n^*) + \frac{1}{2}\frac{\partial^2}{\partial x_1^2} F(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}^*} (x_1 - x_1^*)^2$$

$$+ \frac{1}{2}\frac{\partial^2}{\partial x_1 \partial x_2} F(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}^*} (x_1 - x_1^*)(x_2 - x_2^*) + \cdots$$

Matrix Form

$$F(\mathbf{x}) = F(\mathbf{x}^*) + \nabla F(\mathbf{x})^T\Big|_{\mathbf{x}=\mathbf{x}^*}(\mathbf{x}-\mathbf{x}^*)$$

$$+\frac{1}{2}(\mathbf{x}-\mathbf{x}^*)^T\nabla^2 F(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}^*}(\mathbf{x}-\mathbf{x}^*)+\cdots$$

Gradient

$$\nabla F(\mathbf{x}) = \begin{bmatrix} \frac{\partial}{\partial x_1}F(\mathbf{x}) \\ \frac{\partial}{\partial x_2}F(\mathbf{x}) \\ \vdots \\ \frac{\partial}{\partial x_n}F(\mathbf{x}) \end{bmatrix}$$

Hessian

$$\nabla^2 F(\mathbf{x}) = \begin{bmatrix} \frac{\partial^2}{\partial x_1^2}F(\mathbf{x}) & \frac{\partial^2}{\partial x_1 \partial x_2}F(\mathbf{x}) & \cdots & \frac{\partial^2}{\partial x_1 \partial x_n}F(\mathbf{x}) \\ \frac{\partial^2}{\partial x_2 \partial x_1}F(\mathbf{x}) & \frac{\partial^2}{\partial x_2^2}F(\mathbf{x}) & \cdots & \frac{\partial^2}{\partial x_2 \partial x_n}F(\mathbf{x}) \\ \vdots & \vdots & & \vdots \\ \frac{\partial^2}{\partial x_n \partial x_1}F(\mathbf{x}) & \frac{\partial^2}{\partial x_n \partial x_2}F(\mathbf{x}) & \cdots & \frac{\partial^2}{\partial x_n^2}F(\mathbf{x}) \end{bmatrix}$$

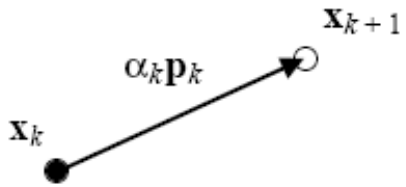
Performance Optimization

- Basic Optimization Algorithm

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$$

or

$$\Delta \mathbf{x}_k = (\mathbf{x}_{k+1} - \mathbf{x}_k) = \alpha_k \mathbf{p}_k$$



$\mathbf{p}_k$ - Search Direction

α_k - Learning Rate

- Steepest Descent

Choose the next step so that the function decreases:

$$F(\mathbf{x}_{k+1}) < F(\mathbf{x}_k)$$

For small changes in $\mathbf{x}$ we can approximate $F(\mathbf{x})$:

$$F(\mathbf{x}_{k+1}) = F(\mathbf{x}_k + \Delta \mathbf{x}_k) \approx F(\mathbf{x}_k) + \mathbf{g}_k^T \Delta \mathbf{x}_k$$

where

$$\mathbf{g}_k \equiv \nabla F(\mathbf{x})\Big|_{\mathbf{x} = \mathbf{x}_k}$$

If we want the function to decrease:

$$\mathbf{g}_k^T \Delta \mathbf{x}_k = \alpha_k \mathbf{g}_k^T \mathbf{p}_k < 0$$

We can maximize the decrease by choosing:

$$\mathbf{p}_k = -\mathbf{g}_k$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha_k \mathbf{g}_k$$

Minimizing Along a Line

Choose α_k to minimize $F(\mathbf{x}_k + \alpha_k \mathbf{p}_k)$

$$\frac{d}{d\alpha_k}(F(\mathbf{x}_k + \alpha_k \mathbf{p}_k)) = \nabla F(\mathbf{x})^T \Big|_{\mathbf{x} = \mathbf{x}_k} \mathbf{p}_k + \alpha_k \mathbf{p}_k^T \nabla^2 F(\mathbf{x}) \Big|_{\mathbf{x} = \mathbf{x}_k} \mathbf{p}_k$$

$$\alpha_k = - \frac{\nabla F(\mathbf{x})^T \Big|_{\mathbf{x} = \mathbf{x}_k} \mathbf{p}_k}{\mathbf{p}_k^T \nabla^2 F(\mathbf{x}) \Big|_{\mathbf{x} = \mathbf{x}_k} \mathbf{p}_k} = - \frac{\mathbf{g}_k^T \mathbf{p}_k}{\mathbf{p}_k^T \mathbf{A}_k \mathbf{p}_k}$$

where

$$\mathbf{A}_k \equiv \nabla^2 F(\mathbf{x}) \Big|_{\mathbf{x} = \mathbf{x}_k}$$

1- Newton's Method

$$F(\mathbf{x}_{k+1}) = F(\mathbf{x}_k + \Delta\mathbf{x}_k) \approx F(\mathbf{x}_k) + \mathbf{g}_k^T \Delta\mathbf{x}_k + \frac{1}{2}\Delta\mathbf{x}_k^T \mathbf{A}_k \Delta\mathbf{x}_k$$

Take the gradient of this second-order approximation and set it equal to zero to find the stationary point:

$$\mathbf{g}_k + \mathbf{A}_k \Delta\mathbf{x}_k = \mathbf{0}$$

$$\Delta\mathbf{x}_k = -\mathbf{A}_k^{-1}\mathbf{g}_k$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{A}_k^{-1}\mathbf{g}_k$$

Example

$$F(\mathbf{x}) = x_1^2 + 2x_1x_2 + 2x_2^2 + x_1$$

$$\mathbf{x}_0 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix}$$

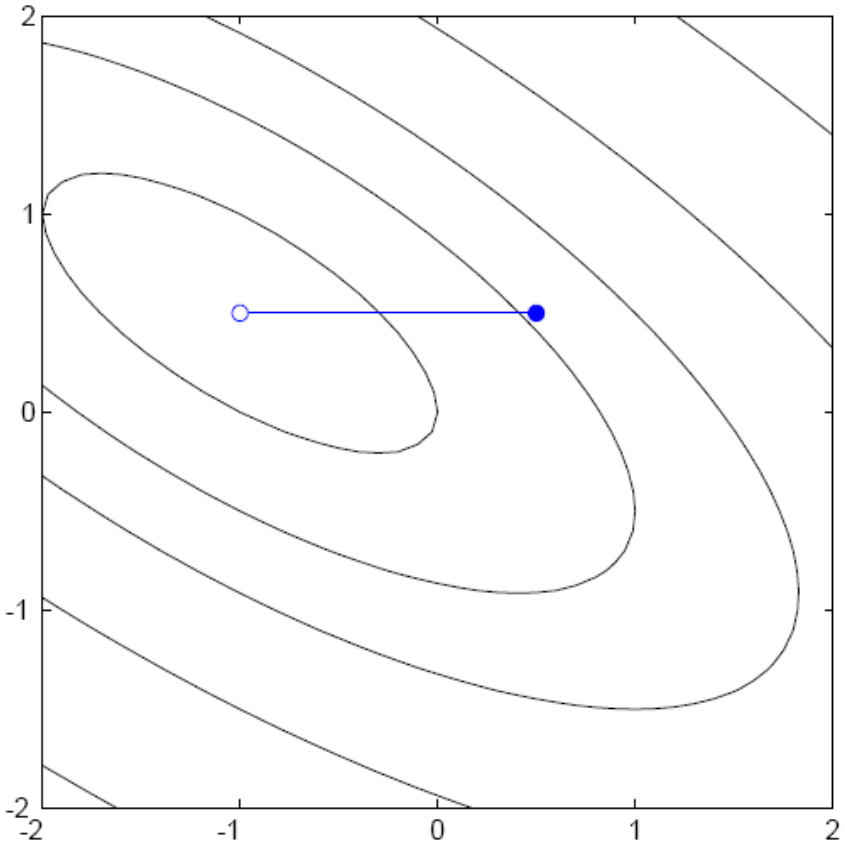
$$\nabla F(\mathbf{x}) = \begin{bmatrix} \frac{\partial}{\partial x_1} F(\mathbf{x}) \\ \frac{\partial}{\partial x_2} F(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} 2x_1 + 2x_2 + 1 \\ 2x_1 + 4x_2 \end{bmatrix}$$

$$\mathbf{g}_0 = \nabla F(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}_0} = \begin{bmatrix} 3 \\ 3 \end{bmatrix}$$

$$\mathbf{A} = \begin{bmatrix} 2 & 2 \\ 2 & 4 \end{bmatrix}$$

$$\mathbf{x}_1 = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} - \begin{bmatrix} 2 & 2 \\ 2 & 4 \end{bmatrix}^{-1} \begin{bmatrix} 3 \\ 3 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} - \begin{bmatrix} 1 & -0.5 \\ -0.5 & 0.5 \end{bmatrix} \begin{bmatrix} 3 \\ 3 \end{bmatrix} = \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} - \begin{bmatrix} 1.5 \\ 0 \end{bmatrix} = \begin{bmatrix} -1 \\ 0.5 \end{bmatrix}$$

Plot

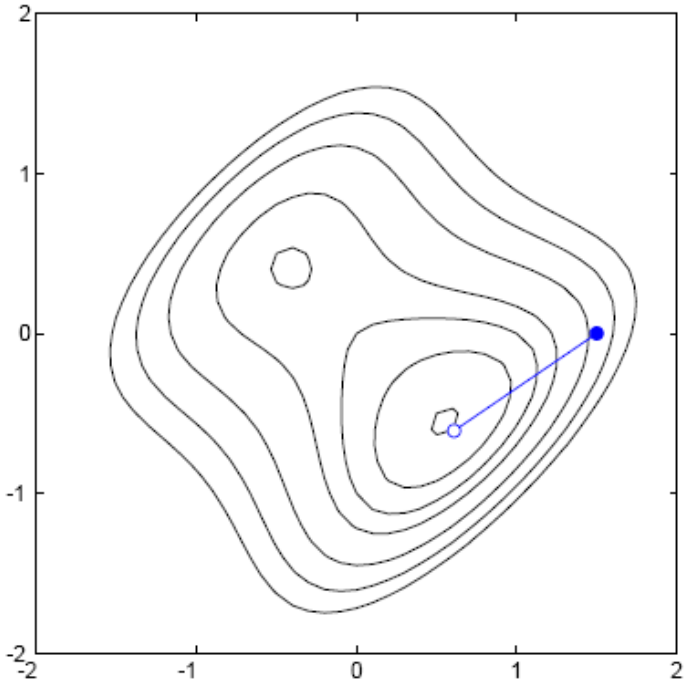


Non-Quadratic Example

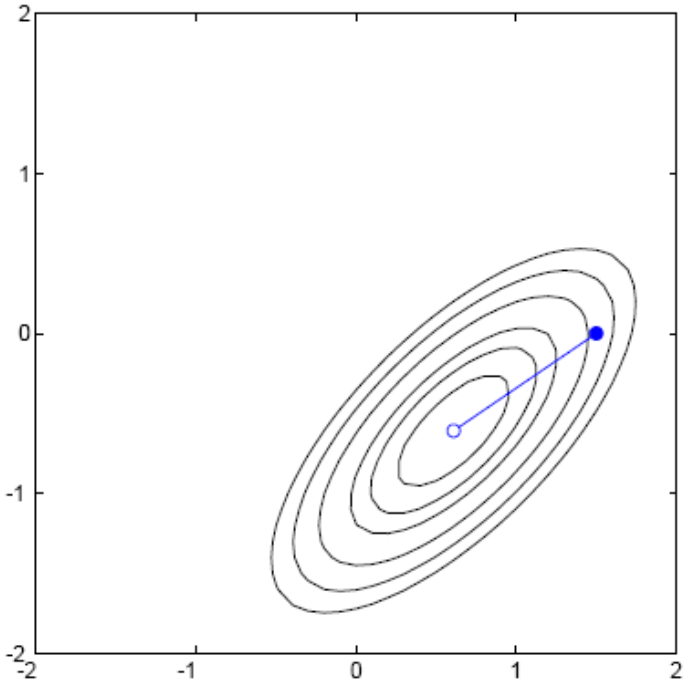
$$F(\mathbf{x}) = (x_2 - x_1)^4 + 8x_1x_2 - x_1 + x_2 + 3$$

Stationary Points: $\mathbf{x}^1 = \begin{bmatrix} -0.42 \\ 0.42 \end{bmatrix}$ $\mathbf{x}^2 = \begin{bmatrix} -0.13 \\ 0.13 \end{bmatrix}$ $\mathbf{x}^3 = \begin{bmatrix} 0.55 \\ -0.55 \end{bmatrix}$

$F(\mathbf{x})$

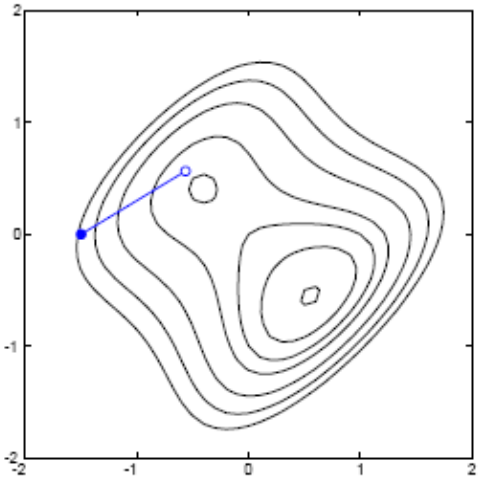


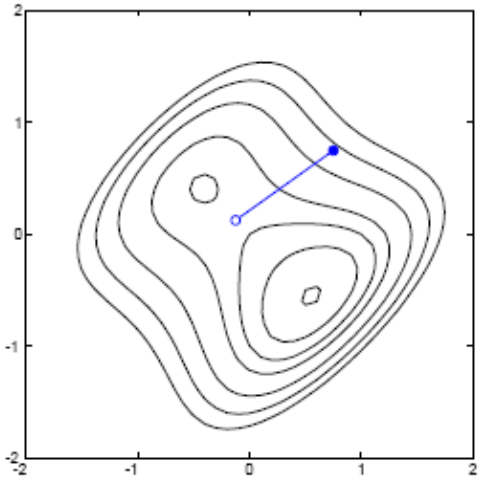
$F_2(\mathbf{x})$

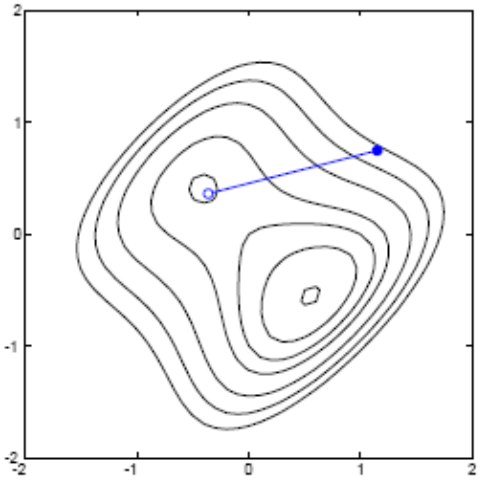


Different Initial Conditions

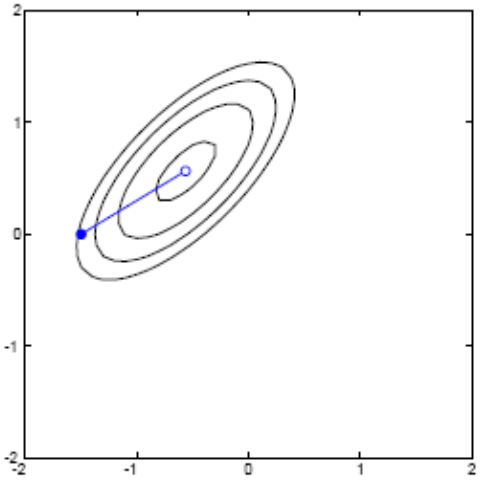
$F(\mathbf{x})$

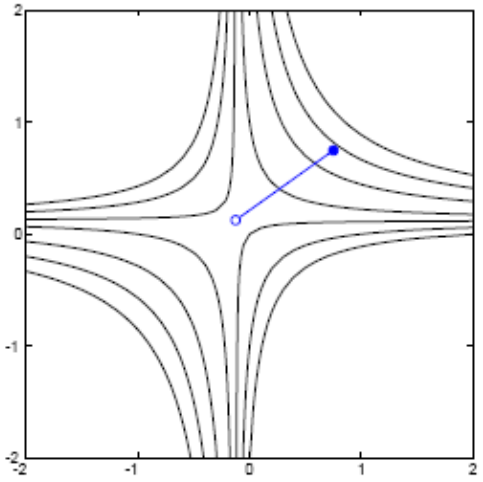


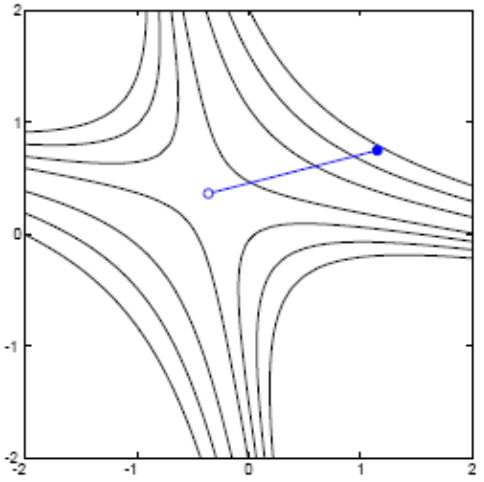




$F_2(\mathbf{x})$







DIFFICULT

- Inverse a singular matrix!!!
- Complexity

Newton's Method

$$\mathbf{x}_{k+1} = \mathbf{x}_k - \mathbf{A}_k^{-1}\mathbf{g}_k$$

$$\mathbf{A}_k \equiv \nabla^2 F(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}_k} \qquad \mathbf{g}_k \equiv \nabla F(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}_k}$$

If the performance index is a sum of squares function:

$$F(\mathbf{x}) = \sum_{i=1}^{N} v_i^2(\mathbf{x}) = \mathbf{v}^T(\mathbf{x})\mathbf{v}(\mathbf{x})$$

then the jth element of the gradient is

$$[\nabla F(\mathbf{x})]_j = \frac{\partial F(\mathbf{x})}{\partial x_j} = 2\sum_{i=1}^{N} v_i(\mathbf{x})\frac{\partial v_i(\mathbf{x})}{\partial x_j}$$

Matrix Form

The gradient can be written in matrix form:

$$\nabla F(\mathbf{x}) = 2\mathbf{J}^T(\mathbf{x})\mathbf{v}(\mathbf{x})$$

where **J** is the Jacobian matrix:

$$\mathbf{J}(\mathbf{x}) = \begin{bmatrix} \frac{\partial v_1(\mathbf{x})}{\partial x_1} & \frac{\partial v_1(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial v_1(\mathbf{x})}{\partial x_n} \\ \frac{\partial v_2(\mathbf{x})}{\partial x_1} & \frac{\partial v_2(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial v_2(\mathbf{x})}{\partial x_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial v_N(\mathbf{x})}{\partial x_1} & \frac{\partial v_N(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial v_N(\mathbf{x})}{\partial x_n} \end{bmatrix}$$

Hessian

$$[\nabla^2 F(\mathbf{x})]_{k,j} = \frac{\partial^2 F(\mathbf{x})}{\partial x_k \partial x_j} = 2\sum_{i=1}^{N}\left\{\frac{\partial v_i(\mathbf{x})}{\partial x_k}\frac{\partial v_i(\mathbf{x})}{\partial x_j} + v_i(\mathbf{x})\frac{\partial^2 v_i(\mathbf{x})}{\partial x_k \partial x_j}\right\}$$

$$\nabla^2 \mathrm{F}(\mathbf{x}) = 2\mathbf{J}^T(\mathbf{x})\mathbf{J}(\mathbf{x}) + 2\mathbf{S}(\mathbf{x})$$

$$\mathbf{S}(\mathbf{x}) = \sum_{i=1}^{N} v_i(\mathbf{x})\nabla^2 v_i(\mathbf{x})$$

2- Gauss-Newton Method

Approximate the Hessian matrix as:

$$\nabla^2 F(\mathbf{x}) \cong 2\mathbf{J}^T(\mathbf{x})\mathbf{J}(\mathbf{x})$$

Newton's method becomes:

$$\mathbf{x}_{k+1} = \mathbf{x}_k - [2\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x}_k)]^{-1} 2\mathbf{J}^T(\mathbf{x}_k)\mathbf{v}(\mathbf{x}_k)$$

$$= \mathbf{x}_k - [\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x}_k)]^{-1} \mathbf{J}^T(\mathbf{x}_k)\mathbf{v}(\mathbf{x}_k)$$

3- Levenberg-Marquardt

Gauss-Newton approximates the Hessian by:

$$\mathbf{H} = \mathbf{J}^T\mathbf{J}$$

This matrix may be singular, but can be made invertible as follows:

$$\mathbf{G} = \mathbf{H} + \mu\mathbf{I}$$

If the eigenvalues and eigenvectors of $\mathbf{H}$ are:

$$\{\lambda_1, \lambda_2, \ldots, \lambda_n\} \qquad \{\mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_n\}$$

then

$$\mathbf{G}\mathbf{z}_i = [\mathbf{H} + \mu\mathbf{I}]\mathbf{z}_i = \mathbf{H}\mathbf{z}_i + \mu\mathbf{z}_i = \lambda_i\mathbf{z}_i + \mu\mathbf{z}_i = \overbrace{(\lambda_i + \mu)}^{\text{Eigenvalues of } \mathbf{G}}\mathbf{z}_i$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k - [\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x}_k) + \mu_k\mathbf{I}]^{-1}\mathbf{J}^T(\mathbf{x}_k)\mathbf{v}(\mathbf{x}_k)$$

Adjustment of μ_k

As $\mu_k \to 0$, LM becomes Gauss-Newton.

$$\mathbf{x}_{k+1} = \mathbf{x}_k - [\mathbf{J}^T(\mathbf{x}_k)\mathbf{J}(\mathbf{x}_k)]^{-1}\mathbf{J}^T(\mathbf{x}_k)\mathbf{v}(\mathbf{x}_k)$$

As $\mu_k \to \infty$, LM becomes Steepest Descent with small learning rate.

$$\mathbf{x}_{k+1} \cong \mathbf{x}_k - \frac{1}{\mu_k}\mathbf{J}^T(\mathbf{x}_k)\mathbf{v}(\mathbf{x}_k) = \mathbf{x}_k - \frac{1}{2\mu_k}\nabla F(\mathbf{x})$$

Therefore, begin with a small μ_k to use Gauss-Newton and speed convergence. If a step does not yield a smaller $F(\mathbf{x})$, then repeat the step with an increased μ_k until $F(\mathbf{x})$ is decreased. $F(\mathbf{x})$ must decrease eventually, since we will be taking a very small step in the steepest descent direction.

Application to Multilayer Network

The performance index for the multilayer network is:

$$F(\mathbf{x}) = \sum_{q=1}^{Q} (\mathbf{t}_q - \mathbf{a}_q)^T (\mathbf{t}_q - \mathbf{a}_q) = \sum_{q=1}^{Q} \mathbf{e}_q^T \mathbf{e}_q = \sum_{q=1}^{Q} \sum_{j=1}^{S^M} (e_{j,q})^2 = \sum_{i=1}^{N} (v_i)^2$$

The error vector is:

$$\mathbf{v}^T = \begin{bmatrix} v_1 & v_2 & \cdots & v_N \end{bmatrix} = \begin{bmatrix} e_{1,1} & e_{2,1} & \cdots & e_{S^M,1} & e_{1,2} & \cdots & e_{S^M,Q} \end{bmatrix}$$

The parameter vector is:

$$\mathbf{x}^T = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix} = \begin{bmatrix} w^1_{1,1} & w^1_{1,2} & \cdots & w^1_{S^1,R} & b^1_1 & \cdots & b^1_{S^1} & w^2_{1,1} & \cdots & b^M_{S^M} \end{bmatrix}$$

The dimensions of the two vectors are:

$$N = Q \times S^M \qquad n = S^1(R+1) + S^2(S^1+1) + \cdots + S^M(S^{M-1}+1)$$

Jacobian Matrix

$$\mathbf{J}(\mathbf{x}) = \begin{bmatrix} \frac{\partial e_{1,1}}{\partial w^1_{1,1}} & \frac{\partial e_{1,1}}{\partial w^1_{1,2}} & \cdots & \frac{\partial e_{1,1}}{\partial w^1_{S^1,R}} & \frac{\partial e_{1,1}}{\partial b^1_1} & \cdots \\ \frac{\partial e_{2,1}}{\partial w^1_{1,1}} & \frac{\partial e_{2,1}}{\partial w^1_{1,2}} & \cdots & \frac{\partial e_{2,1}}{\partial w^1_{S^1,R}} & \frac{\partial e_{2,1}}{\partial b^1_1} & \cdots \\ \vdots & \vdots & & \vdots & \vdots & \\ \frac{\partial e_{S^M,1}}{\partial w^1_{1,1}} & \frac{\partial e_{S^M,1}}{\partial w^1_{1,2}} & \cdots & \frac{\partial \mathrm{e}_{e_{S^M,1}}}{\partial w^1_{S^1,R}} & \frac{\partial \mathrm{e}_{e_{S^M,1}}}{\partial b^1_1} & \cdots \\ \frac{\partial e_{1,2}}{\partial w^1_{1,1}} & \frac{\partial e_{1,2}}{\partial w^1_{1,2}} & \cdots & \frac{\partial e_{1,2}}{\partial w^1_{S^1,R}} & \frac{\partial e_{1,2}}{\partial b^1_1} & \cdots \\ \vdots & \vdots & & \vdots & \vdots & \end{bmatrix}$$

Computing the Jacobian

SDBP computes terms like:

$$\frac{\partial \hat{F}(\mathbf{x})}{\partial x_l} = \frac{\partial \mathbf{e}_q^T \mathbf{e}_q}{\partial x_l}$$

using the chain rule:

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

where the sensitivity

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

is computed using backpropagation.

For the Jacobian we need to compute terms like:

$$[\mathbf{J}]_{h,l} = \frac{\partial v_h}{\partial x_l} = \frac{\partial e_{k,q}}{\partial x_l}$$

Marquardt Sensitivity

If we define a Marquardt sensitivity:

$$\tilde{s}^m_{i,h} \equiv \frac{\partial v_h}{\partial n^m_{i,q}} = \frac{\partial e_{k,q}}{\partial n^m_{i,q}} \qquad h = (q-1)S^M + k$$

We can compute the Jacobian as follows:

weight

$$[\mathbf{J}]_{h,l} = \frac{\partial v_h}{\partial x_l} = \frac{\partial e_{k,q}}{\partial w^m_{i,j}} = \frac{\partial e_{k,q}}{\partial n^m_{i,q}} \times \frac{\partial n^m_{i,q}}{\partial w^m_{i,j}} = \tilde{s}^m_{i,h} \times \frac{\partial n^m_{i,q}}{\partial w^m_{i,j}} = \tilde{s}^m_{i,h} \times a^{m-1}_{j,q}$$

bias

$$[\mathbf{J}]_{h,l} = \frac{\partial v_h}{\partial x_l} = \frac{\partial e_{k,q}}{\partial b^m_i} = \frac{\partial e_{k,q}}{\partial n^m_{i,q}} \times \frac{\partial n^m_{i,q}}{\partial b^m_i} = \tilde{s}^m_{i,h} \times \frac{\partial n^m_{i,q}}{\partial b^m_i} = \tilde{s}^m_{i,h}$$

Computing the Sensitivities

Initialization

$$\tilde{s}^{M}_{i,h} = \frac{\partial v_h}{\partial n^{M}_{i,q}} = \frac{\partial e_{k,q}}{\partial n^{M}_{i,q}} = \frac{\partial (t_{k,q} - a^{M}_{k,q})}{\partial n^{M}_{i,q}} = -\frac{\partial a^{M}_{k,q}}{\partial n^{M}_{i,q}}$$

$$\tilde{s}^{M}_{i,h} = \begin{cases} -\dot{f}^{M}(n^{M}_{i,q}) & \text{for } i = k \\ 0 & \text{for } i \neq k \end{cases}$$

$$\tilde{\mathbf{S}}^{M}_{q} = -\dot{\mathbf{F}}^{M}(\mathbf{n}^{M}_{q})$$

Backpropagation

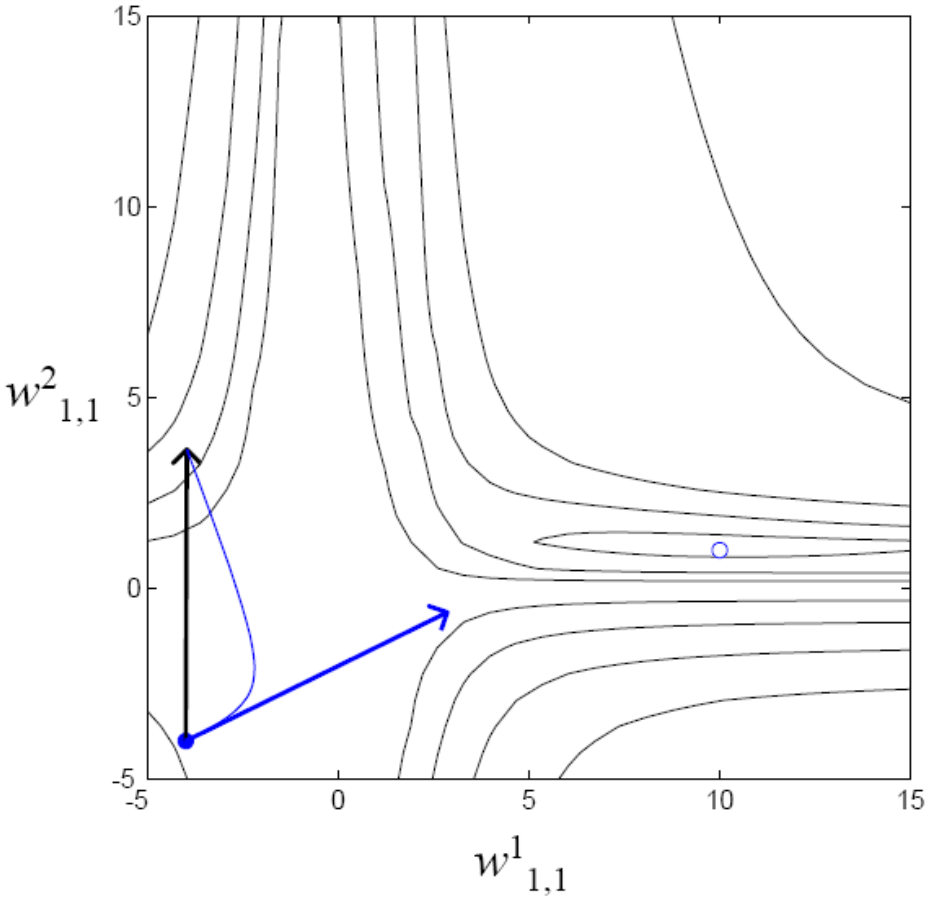
$$\tilde{\mathbf{S}}^{m}_{q} = \dot{\mathbf{F}}^{m}(\mathbf{n}^{m}_{q})(\mathbf{W}^{m+1})^{T}\tilde{\mathbf{S}}^{m+1}_{q}$$

$$\tilde{\mathbf{S}}^{m} = \left[\tilde{\mathbf{S}}^{m}_{1} \middle| \tilde{\mathbf{S}}^{m}_{2} \middle| \cdots \middle| \tilde{\mathbf{S}}^{m}_{Q}\right]$$

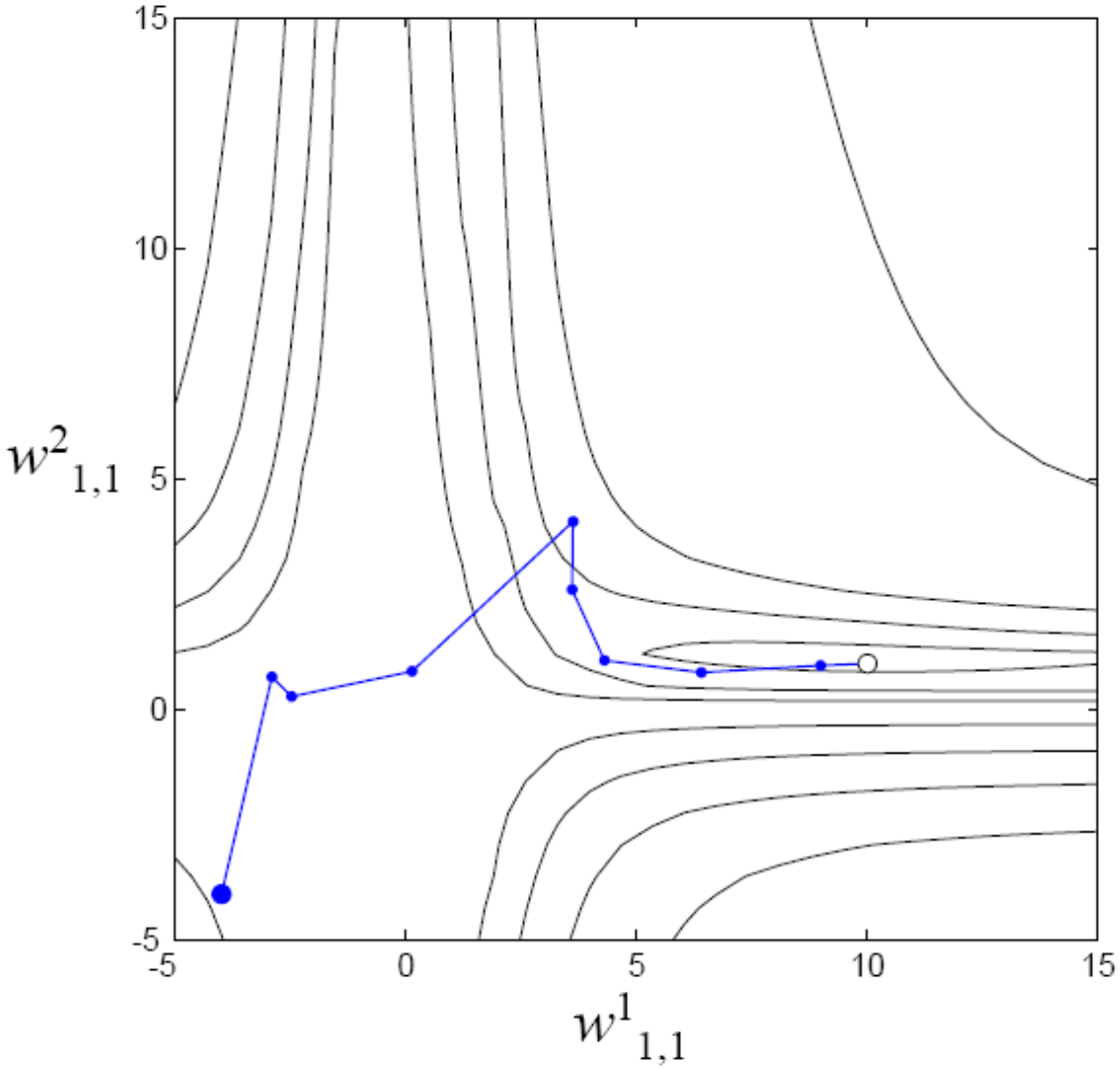
LMBP

- Present all inputs to the network and compute the corresponding network outputs and the errors. Compute the sum of squared errors over all inputs.
- Compute the Jacobian matrix. Calculate the sensitivities with the backpropagation algorithm, after initializing. Augment the individual matrices into the Marquardt sensitivities. Compute the elements of the Jacobian matrix.
- Solve to obtain the change in the weights.
- Recompute the sum of squared errors with the new weights. If this new sum of squares is smaller than that computed in step 1, then divide μ_k by υ, update the weights and go back to step 1. If the sum of squares is not reduced, then multiply μ_k by υ and go back to step 3.

Example LMBP Step



LMBP Trajectory



۵- Conjugate Gradient

1. The first search direction is steepest descent.

$$\mathbf{p}_0 = -\mathbf{g}_0 \qquad \mathbf{g}_k \equiv \nabla F(\mathbf{x})\Big|_{\mathbf{x}=\mathbf{x}_k}$$

2. Take a step and choose the learning rate to minimize the function along the search direction.

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$$

3. Select the next search direction according to:

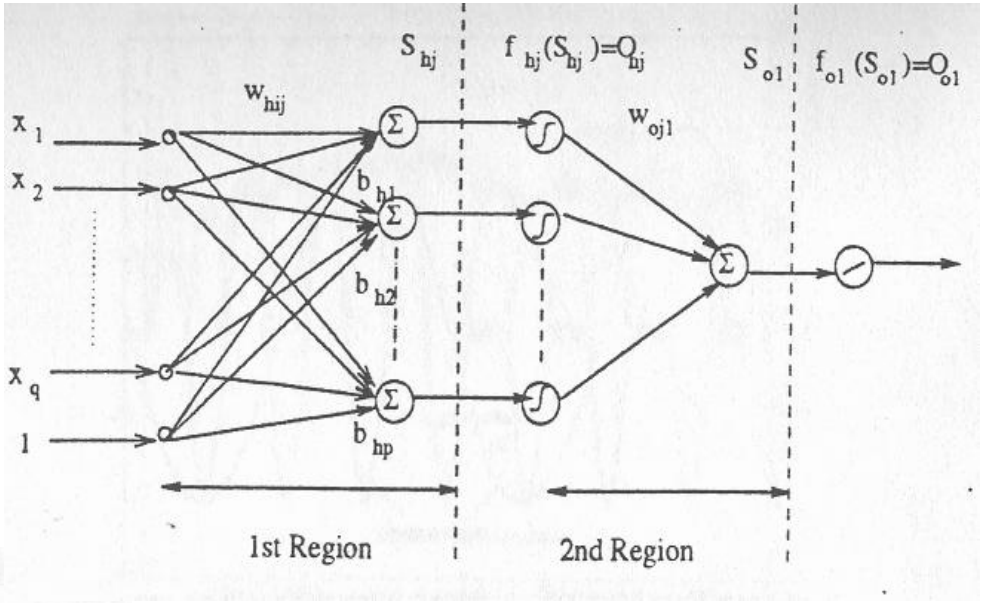
$$\mathbf{p}_k = -\mathbf{g}_k + \beta_k \mathbf{p}_{k-1}$$

where

$$\beta_k = \frac{\Delta\mathbf{g}_{k-1}^T \mathbf{g}_k}{\Delta\mathbf{g}_{k-1}^T \mathbf{p}_{k-1}} \quad \text{or} \quad \beta_k = \frac{\mathbf{g}_k^T \mathbf{g}_k}{\mathbf{g}_{k-1}^T \mathbf{g}_{k-1}} \quad \text{or} \quad \beta_k = \frac{\Delta\mathbf{g}_{k-1}^T \mathbf{g}_k}{\mathbf{g}_{k-1}^T \mathbf{g}_{k-1}}$$

LRLS Method

- Recursive least square method based method, does not need gradient descent.



$$\begin{aligned}\phi_h^T(k) &= [x_1(k), x_2(k), ..., x_q(k), 1] \\ &= [y_d(k+1), y_p(k), ..., y_p(k-n_y+1) \\ &\quad u(k-1), ..., u(k-n_u+1), 1]\end{aligned}$$

$$W_{hj}^T(k) = [w_{h1j}(k), w_{h2j}(k), \ldots, w_{hqj}(k), b_{hj}(k)]$$

$$\begin{aligned}S_{hj}(k) &= \phi_h^T(k) W_{hj}(k) \\ S_{o1}(k) &= \phi_o^T(k) W_{o1}(k)\end{aligned}$$

$$\phi_o^T(k) = [O_{h1}(k), O_{h2}(k), \ldots, O_{hp}(k)]$$

$$W_{o1}^T(k) = [w_{o11}(k), w_{o21}(k), \ldots, w_{op1}(k)]$$

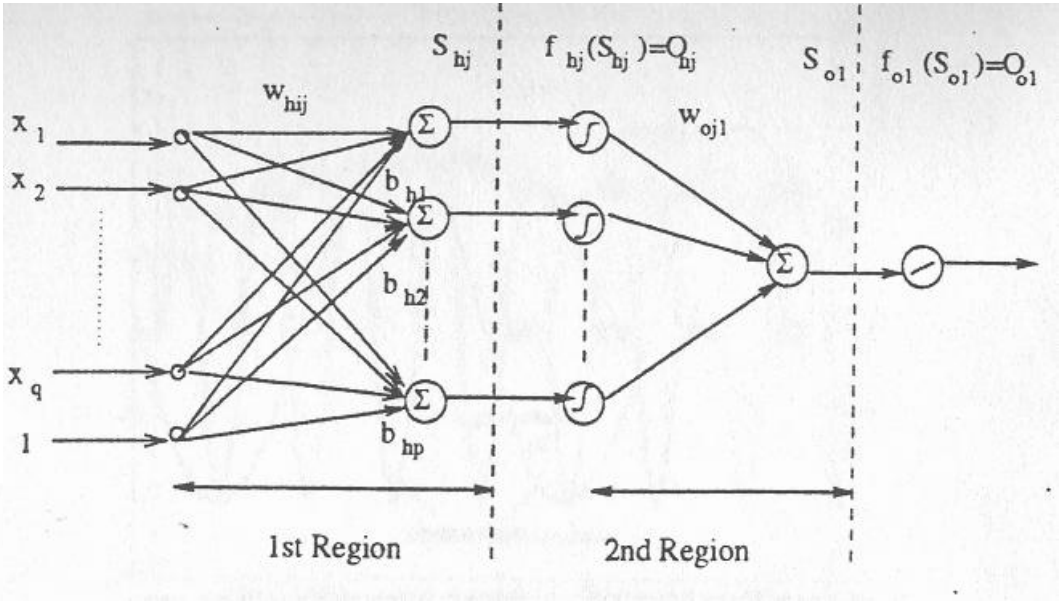
$$e_{hj}(k) = S_{hj}^*(k) - S_{hj}(k)$$

$$e_{o1}(k) = S_{o1}^*(k) - S_{o1}(k)$$

LRLS Method Cont.

$$E_c(k) = \frac{1}{2}[y_d(k) - y_p(k)]^2$$

$$e_c(k) = y_d(k) - y_p(k)$$



$$\begin{aligned} e_{hj}(k) &\approx -\frac{\partial E_c(k)}{\partial S_{hj}(k)} \\ &\approx e_{o1}(k) w_{oj1}(k) f'_{hj}(k) \end{aligned}$$

$$\begin{aligned} e_{o1}(k) &\approx -\frac{\partial E_c(k)}{\partial S_{o1}(k)} \\ &\approx e_c(k) f'_{o1}(k) \end{aligned}$$

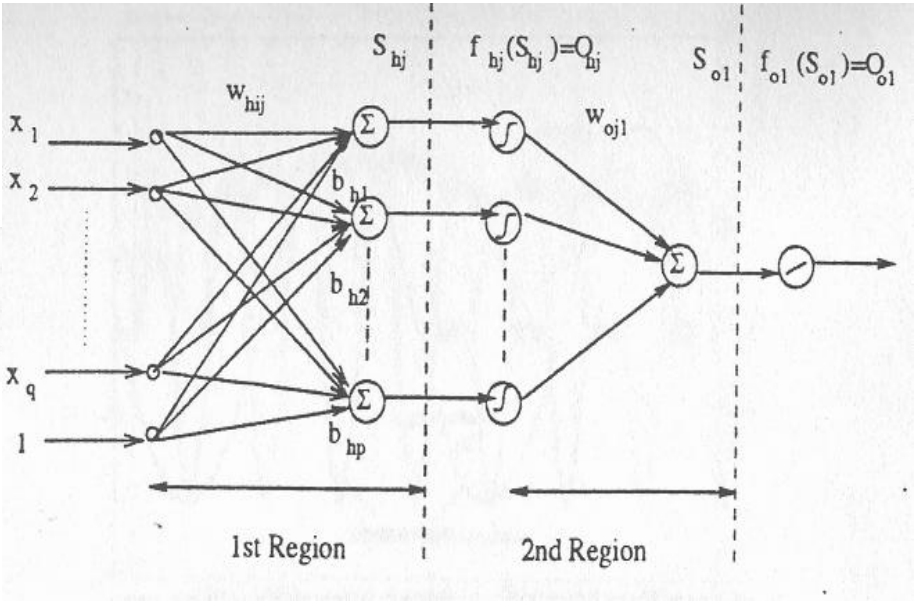
LRLS Method Cont.

$$P(0) = \sigma I_m$$

$$P(k) = \frac{1}{\lambda} P(k-1) \left\{ I_m - \frac{\phi(k)\phi^T(k)P(k-1)}{\lambda + \phi^T(k)P(k-1)\phi(k)} \right\}$$

$$W_{hj}(k) = W_{hj}(k-1) + P_h(k)\phi_h(k)(e_{hj}(k))$$

$$W_{o1}(k) = W_{o1}(k-1) + P_o(k)\phi_o(k)(e_{o1}(k))$$



Example for compare methods

$$r(k) = sin(\frac{0.1\pi k}{25})$$

$$y_p(k+1) = \frac{y_p(k)}{1+y_p^2(k)} + u^3(k) + e_n(k)$$

Algorithms	Average time steps (cut off at 0.15)	Average time steps (cut off at 0.1)	Average time steps (cut off at 0.05)
BPM	101	175	594
DBD	64	167	2382
LM	2	123	264
LRLS	3	9	69

IGLS (Integration of the Gradient and Least Square)

- The LRLS algorithm has faster convergence speed than BPM algorithm. However, it is computationally more complex and is not ideal for very large network size. The learning strategy describe here combines ideas from both the algorithm to achieve the objectives of maintaining complexity for large network size and reasonably fast convergence.

  - The out put layer weights update using BPM method

$$W(k) = W(k-1) + (-\eta \frac{\partial E(k)}{\partial W(k-1)}) + \alpha \Delta W(k-1)$$

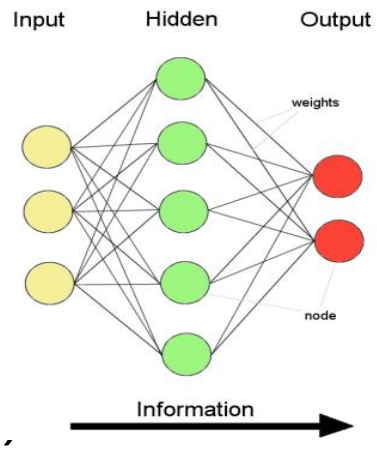
  - The hidden layer weights update using BPM method

$$W_{hj}(k) = W_{hj}(k-1) + P_h(k)\phi_h(k)(e_{hj}(k))$$

فصل ششم :

شبکه عصبی بازگشتی Recurrent Networks

Recurrent Networks



- **Feed forward networks:**
  - Information only flows one way
  - One input pattern produces one output
  - No sense of time (or memory of previous s

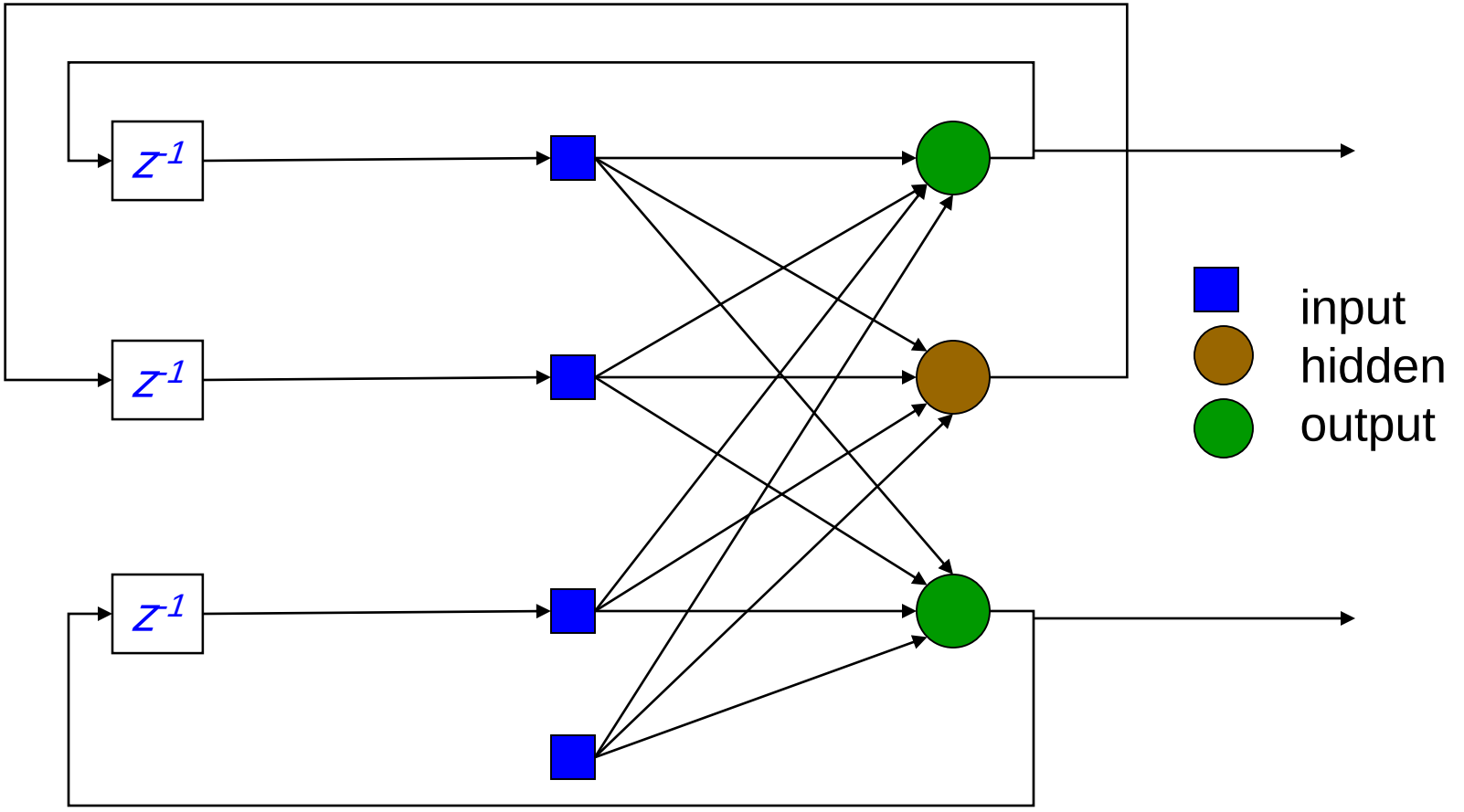
- **Recurrency**
  - Nodes connect back to other nodes or themselves
  - Information flow is multidirectional
  - Sense of time and memory of previous state(s)

- **Biological nervous systems show high levels of recurrency**

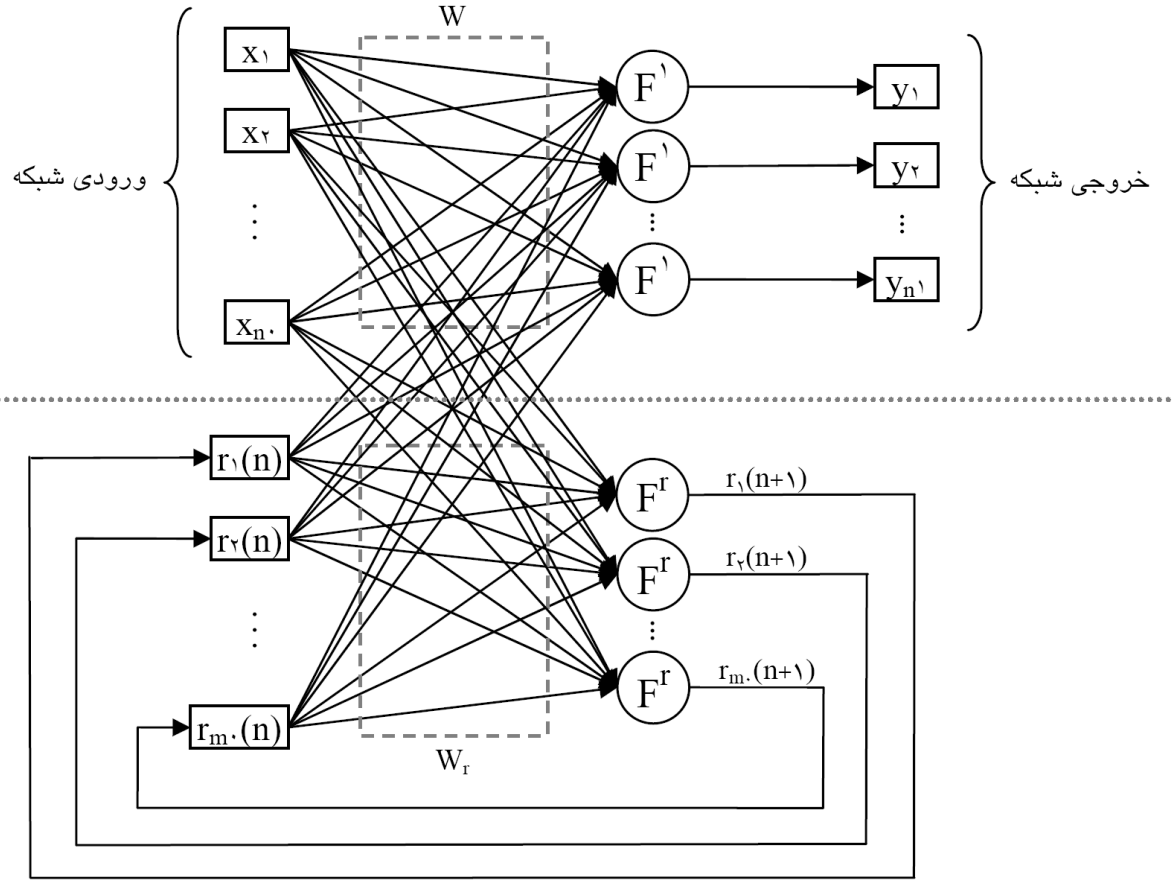
  (but feed-forward structures exists too)

Recurrent Networks

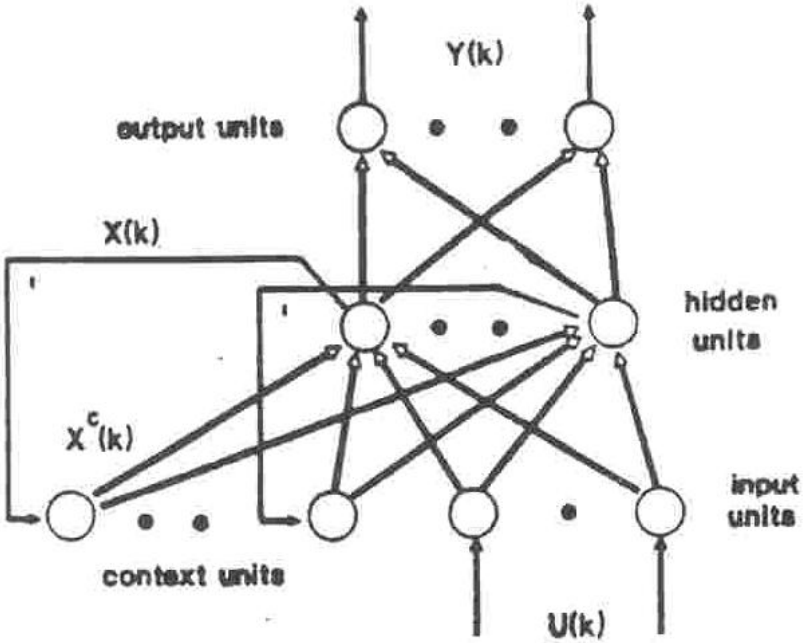
Recurrent Network with ***hidden neuron***: unit delay operator z^{-1} is used to model a dynamic system

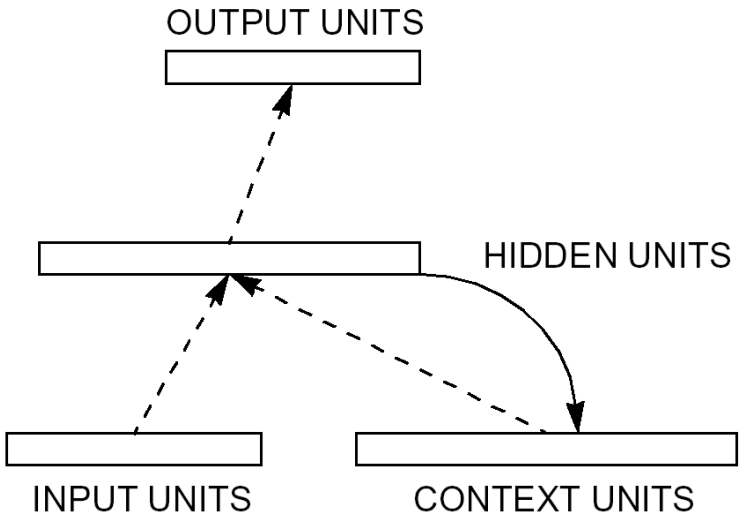


Rabinson and Fallside

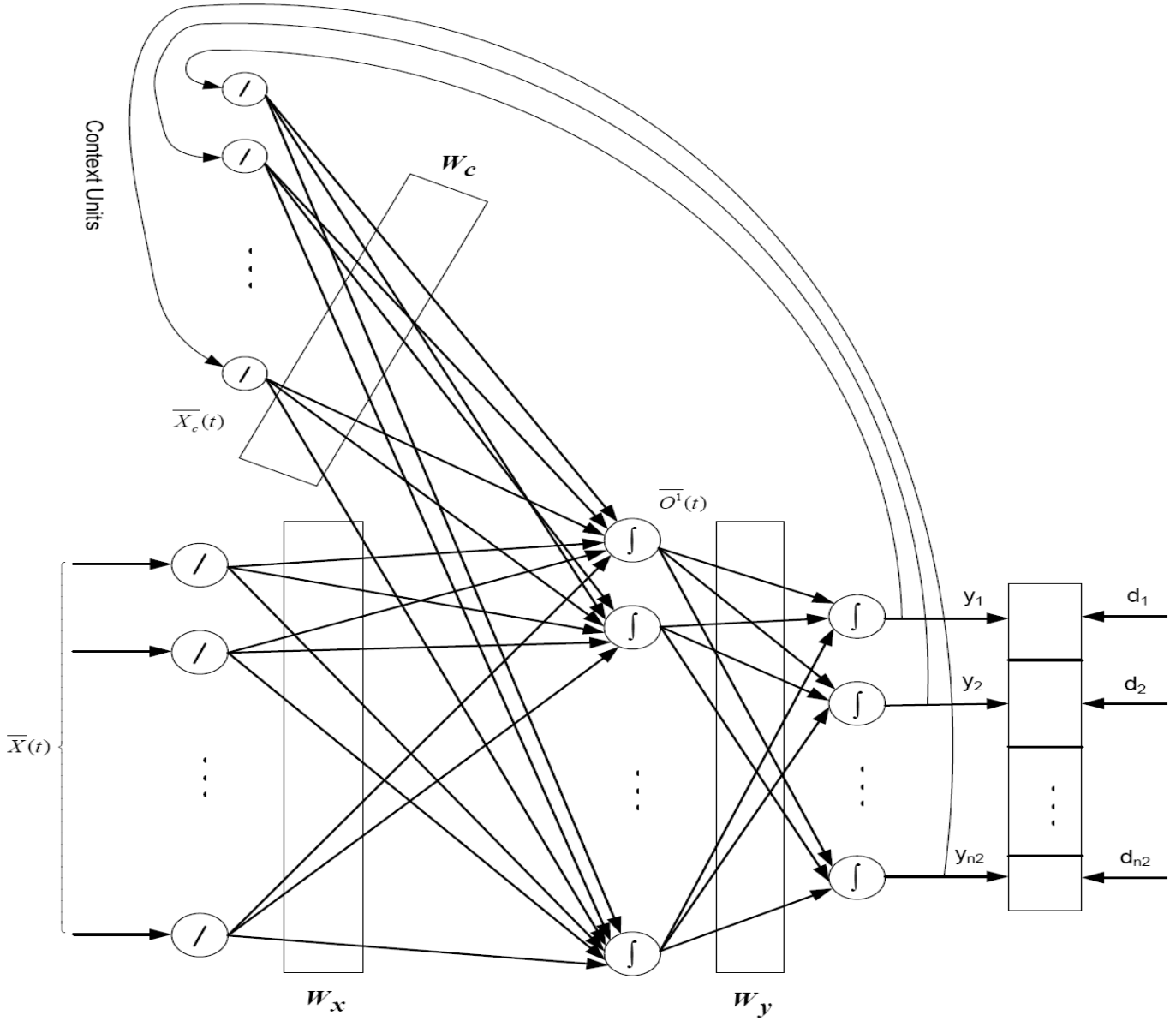


Elman

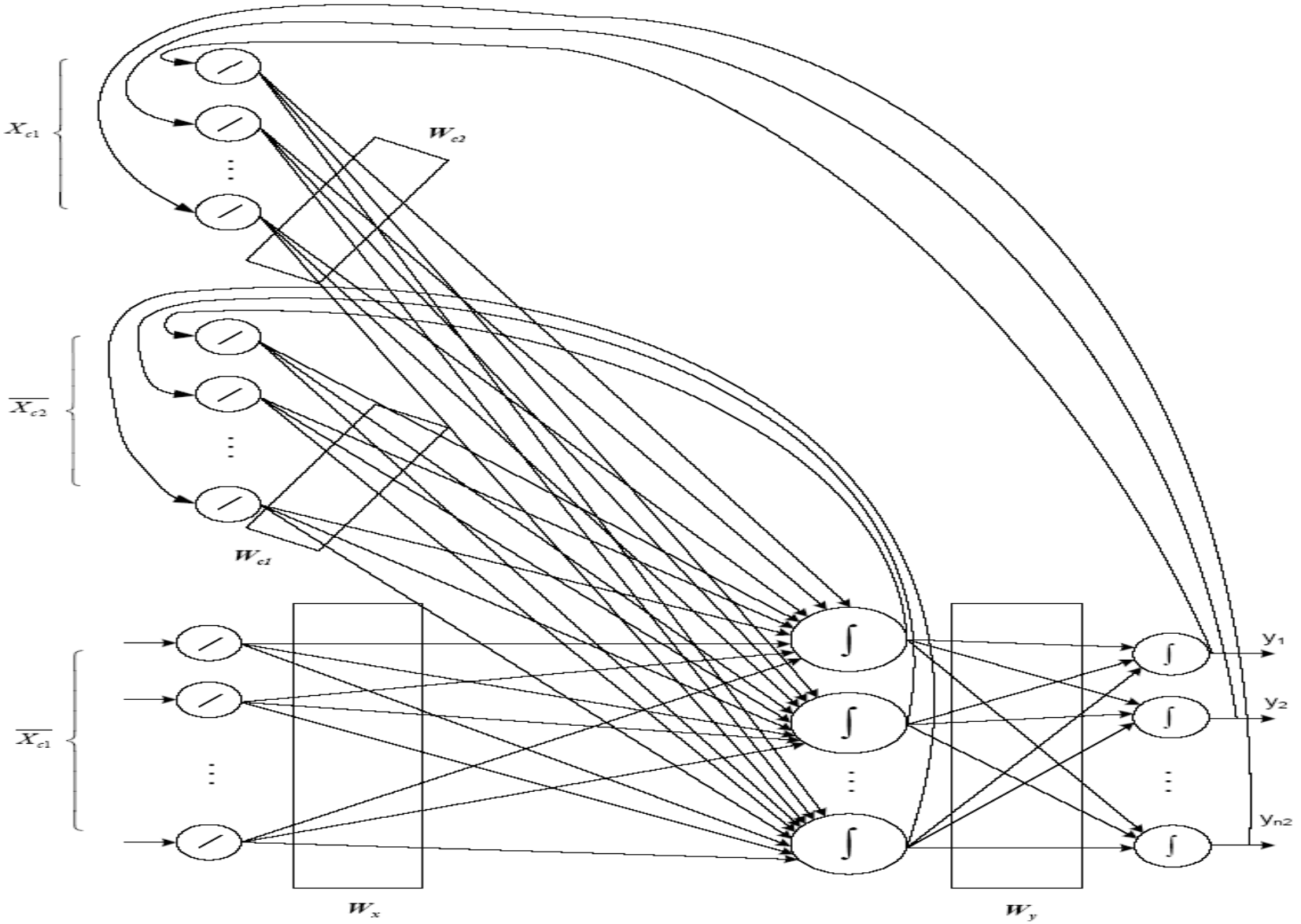

Y(k)
output units
X(k)
hidden units
$X^c(k)$
input units
context units
U(k)


OUTPUT UNITS
HIDDEN UNITS
INPUT UNITS
CONTEXT UNITS

Jordan


Context Units
W_c
$\overline{X_c}(t)$
$\overline{O^1}(t)$
$\overline{X}(t)$
y_1
y_2
y_{n2}
d_1
d_2
d_{n2}
W_x
W_y

Elman and Jordan

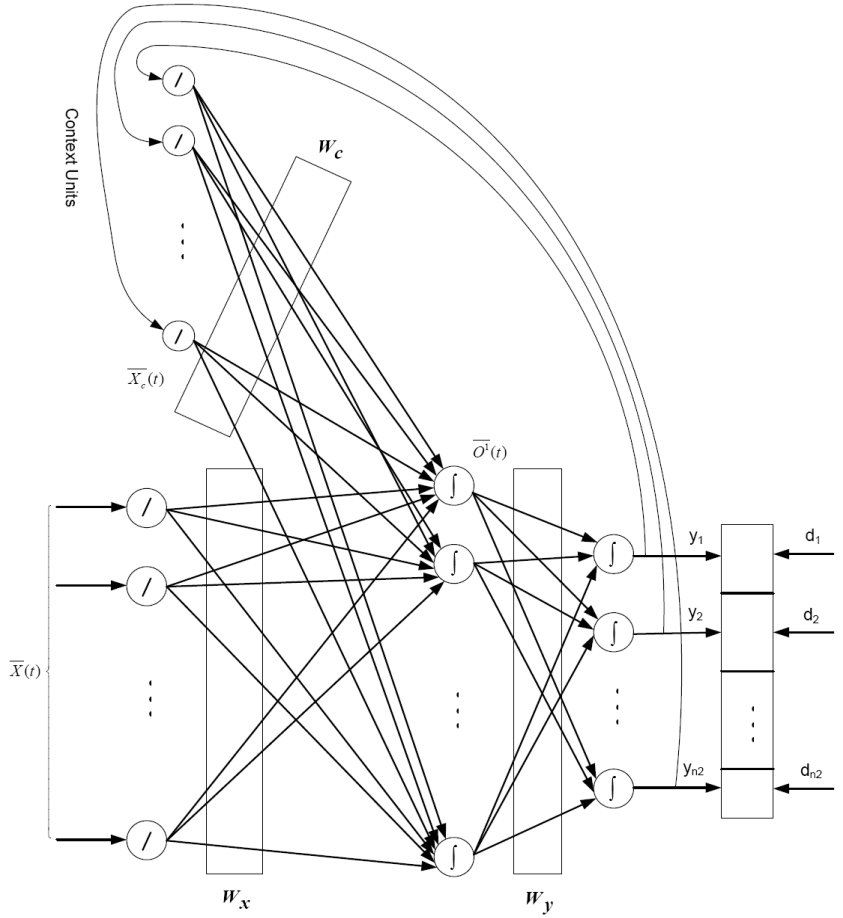


Jordan Learning

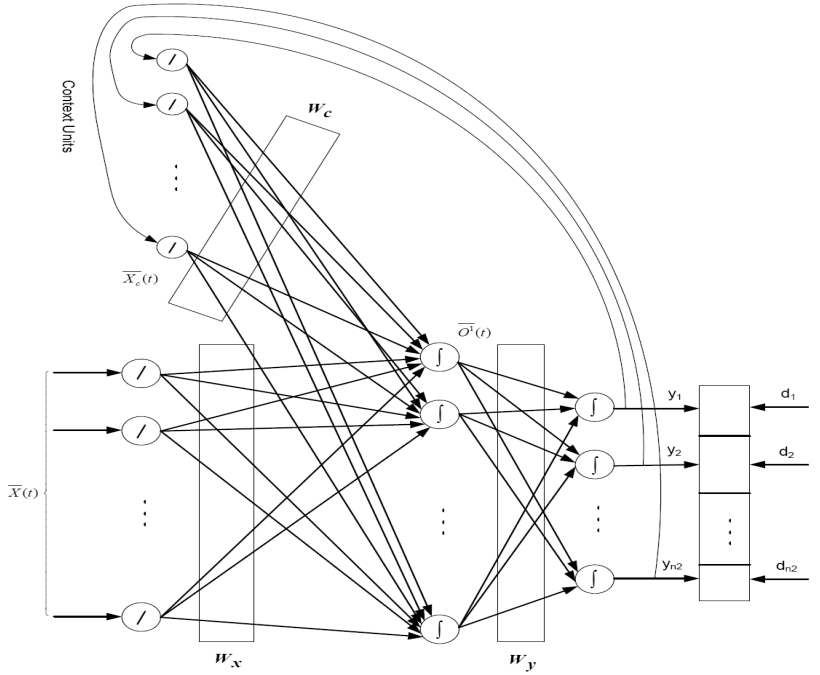
$$\begin{cases} \overline{net^1} = \boldsymbol{W_x}.\overline{X}(t) + \boldsymbol{W_c}.\overline{X_c}(t) \\ where: \\ \overline{X_c}(t) = \overline{O^2}(t-1) = \overline{y}(t-1) \end{cases}$$

$$\overline{F^1}(\overline{net^1}) = \overline{O^1}(\overline{net^1})$$

$$\begin{cases} \overline{net^2} = \boldsymbol{W_y}.\overline{O^1}(t) \\ \overline{F^2}(\overline{net^2}) = \overline{O^2}(\overline{net^2}) = \overline{y}(\overline{net^2}) \end{cases}$$



Jordan Learning



$$\begin{cases} \Delta W_x = -\eta \dfrac{\partial E}{\partial W_x} \\ \Delta W_y = -\eta \dfrac{\partial E}{\partial W_y} \\ \Delta W_c = -\eta \dfrac{\partial E}{\partial W_c} = -\eta . \dfrac{\partial E}{\partial \overline{O^2}} . \dfrac{\partial \overline{O^2}}{\partial \overline{net^2}} . \dfrac{\partial \overline{net^2}}{\partial \overline{O^1}} . \dfrac{\partial \overline{O^1}}{\partial \overline{net^1}} . \dfrac{\partial \overline{net^1}}{\partial W_c} \end{cases}$$

$$= \eta . \underbrace{\overline{e} . \overline{F^{2\prime}} . W_y . \overline{F^{1\prime}}}_{\overline{\delta^1}} . \left\{ \overline{X_c}(t) + W_c . \frac{\partial \overline{X_c}(t)}{\partial W_c} \right\}$$

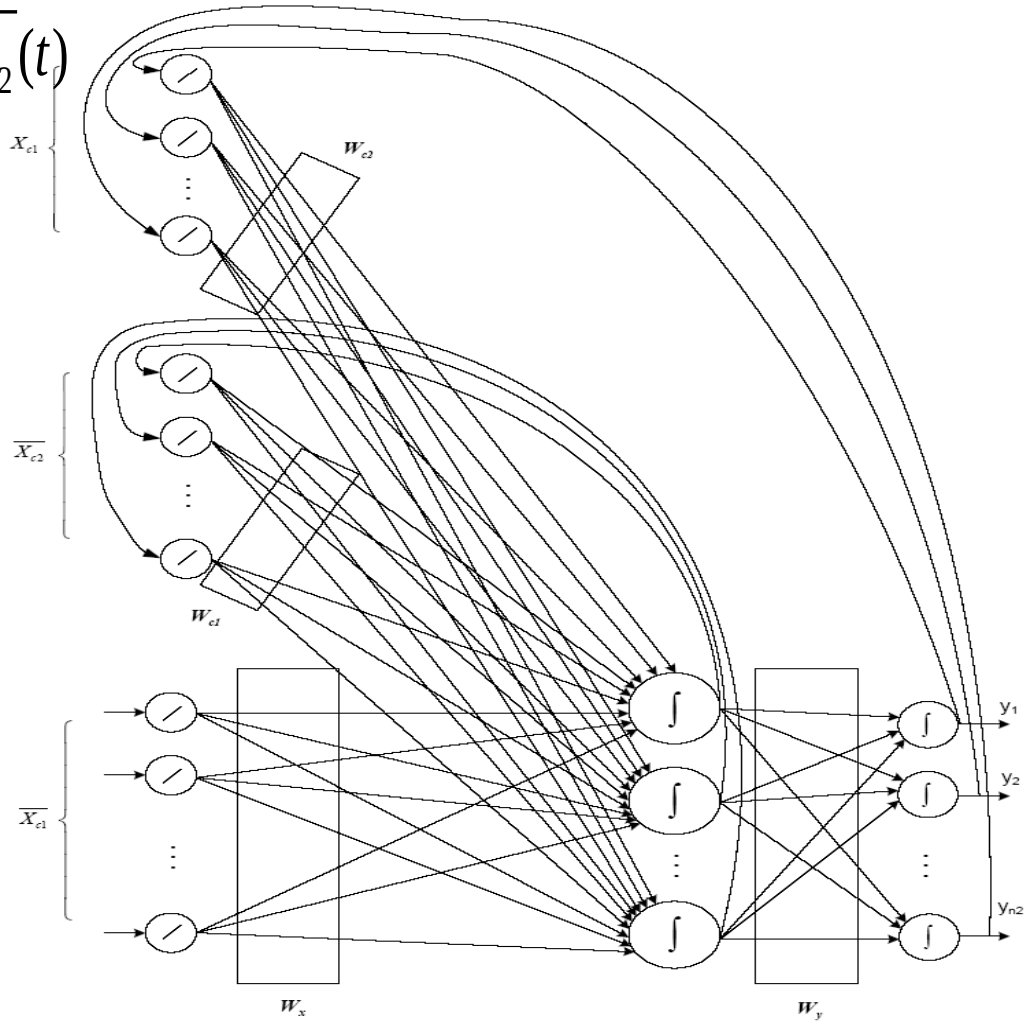
$$\Delta W = \eta . \overline{\delta^1} . \left\{ \overline{X_c}(t) + W_c . \frac{\partial \overline{X_c}(t)}{\partial W_c} \right\}$$

Elman and Jordan Learning

$$\begin{cases} \overline{net^1} = W_x.\overline{X}(t) + W_{c1}.\overline{X_{c1}}(t) + W_{c2}.\overline{X_{c2}}(t) \\ where: \\ \overline{X_{c1}}(t) = \overline{O^1}(t-1) = \overline{y}(t-1) \\ \overline{X_{c2}}(t) = \overline{O^2}(t-1) = \overline{y}(t-1) \end{cases}$$

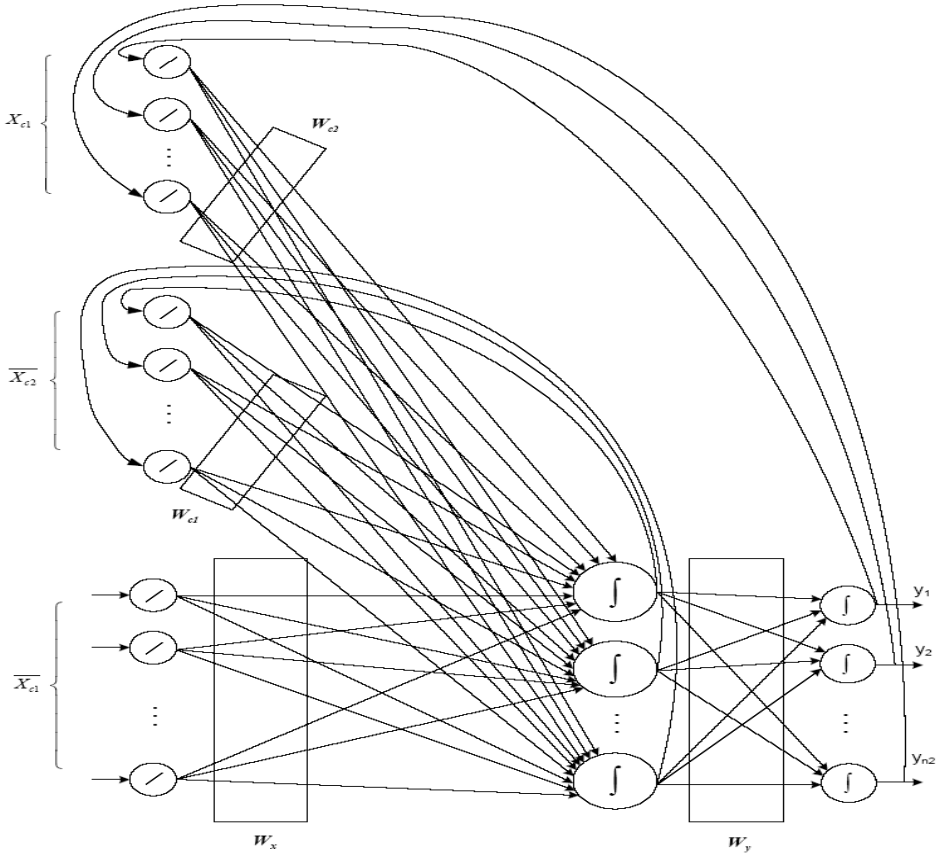
$$\overline{F^1}(\overline{net^1}) = \overline{O^1}(\overline{net^1})$$

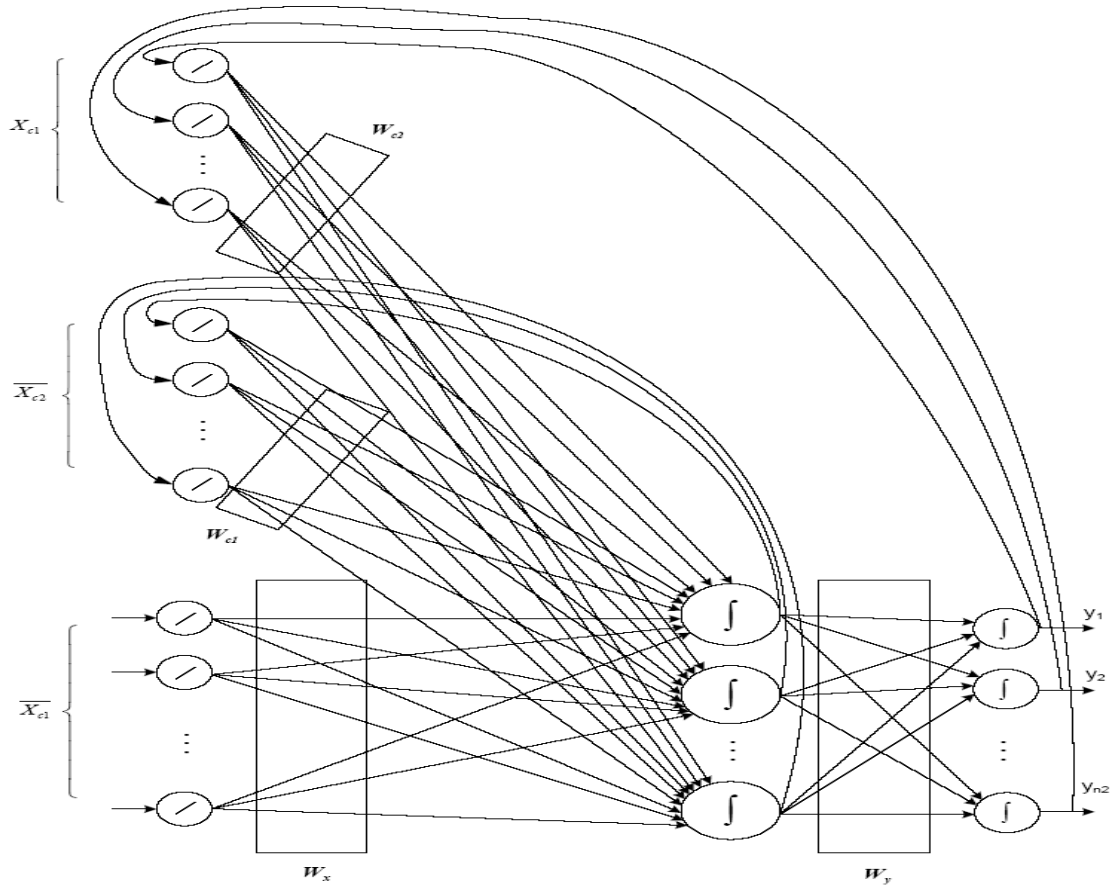
$$\begin{cases} \overline{net^2} = W_y.\overline{O^1}(t) \\ \overline{F^2}(\overline{net^2}) = \overline{O^2}(\overline{net^2}) = \overline{y}(\overline{net^2}) \end{cases}$$



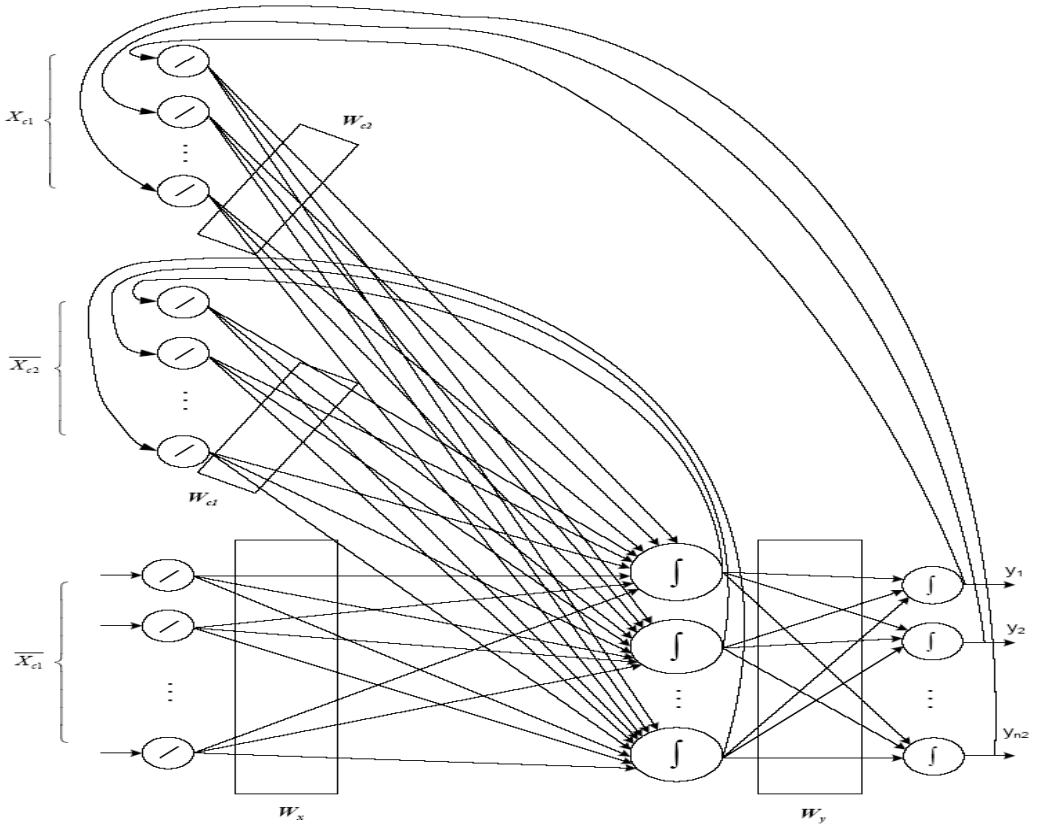
Elman and Jordan Learning

$$\begin{cases} \Delta W_y = -\eta \dfrac{\partial E}{\partial W_y} & (1) \\ \Delta W_x = -\eta \dfrac{\partial E}{\partial W_x} & (2) \\ \Delta W_{c1} = -\eta \dfrac{\partial E}{\partial W_{c1}} & (3) \\ \Delta W_{c2} = -\eta \dfrac{\partial E}{\partial W_{c2}} & (4) \end{cases}$$





$$1)\quad \Delta \boldsymbol{W}_y = -\eta \frac{\partial E}{\partial \boldsymbol{W}_y} = -\eta . \frac{\partial E}{\partial \overline{O^2}} . \frac{\partial \overline{O^2}}{\partial \overline{net^2}} . \frac{\partial \overline{net^2}}{\partial \boldsymbol{W}_y} = \eta . \underbrace{\overline{e} . \overline{F^{2\prime}}(.)}_{\overline{\delta^2}} . \overline{O^1}$$



2) $$\Delta \boldsymbol{W}_x = -\eta \frac{\partial E}{\partial \boldsymbol{W}_x} = -\eta . \frac{\partial E}{\partial \overline{O^2}} . \frac{\partial \overline{O^2}}{\partial \overline{net^2}} . \frac{\partial \overline{net^2}}{\partial \overline{O^1}} . \frac{\partial \overline{O^1}}{\partial \overline{net^1}} . \frac{\partial \overline{net^1}}{\partial \boldsymbol{W}_x}$$

$$= \eta . \underbrace{\overline{e} . \overline{F^{2\prime}} . \boldsymbol{W}_y . \overline{F^{1\prime}}}_{\overline{\delta^1}} . \overline{X}(t)$$

$$\Delta \boldsymbol{W}_x = \eta . \overline{\delta^1} . \overline{X}(t)$$

3) $$\Delta \boldsymbol{W}_{c1} = -\eta \frac{\partial E}{\partial \boldsymbol{W}_{c1}} = -\eta . \frac{\partial E}{\partial \overline{O^2}} . \frac{\partial \overline{O^2}}{\partial \overline{net^2}} . \frac{\partial \overline{net^2}}{\partial \overline{O^1}} . \frac{\partial \overline{O^1}}{\partial \overline{net^1}} . \frac{\partial \overline{net^1}}{\partial \boldsymbol{W}_{c1}}$$

$$= \eta . \underbrace{\overline{e} . \overline{F^{2\prime}}(.) . \boldsymbol{W}_y . \overline{F^{1\prime}}(.)}_{\overline{\delta^1}} . \left\{ \overline{X_{c1}}(t) + \boldsymbol{W}_{c1} . \frac{\partial \overline{X_{c1}}(t)}{\partial \boldsymbol{W}_{c1}} \right\}$$

$$\Delta \boldsymbol{W}_{c1} = \eta . \overline{\delta^1} . \left\{ \overline{X_{c1}}(t) + \boldsymbol{W}_{c1} . \frac{\partial \overline{X_{c1}}(t)}{\partial \boldsymbol{W}_{c1}} \right\}$$

$$= \eta . \overline{\delta^1} . \left\{ \overline{O^1}(t-1) + \boldsymbol{W}_{c1} . \frac{\partial \overline{O^1}(t-1)}{\partial \boldsymbol{W}_{c1}} \right\}$$

4) $$\Delta W_{c2} = -\eta \frac{\partial E}{\partial W_{c2}} = -\eta . \frac{\partial E}{\partial \overline{O^2}} . \frac{\partial \overline{O^2}}{\partial \overline{net^2}} . \frac{\partial \overline{net^2}}{\partial \overline{O^1}} . \frac{\partial \overline{O^1}}{\partial \overline{net^1}} . \frac{\partial \overline{net^1}}{\partial W_{c2}}$$

$$= \eta . \underbrace{\overline{e} . \overline{F^{2\prime}}(.) . W_y . \overline{F^{1\prime}}(.)}_{\overline{\delta^1}} . \left\{ \overline{X_{c2}}(t) + W_{c2} . \frac{\partial \overline{X_{c2}}(t)}{\partial W_{c2}} \right\}$$

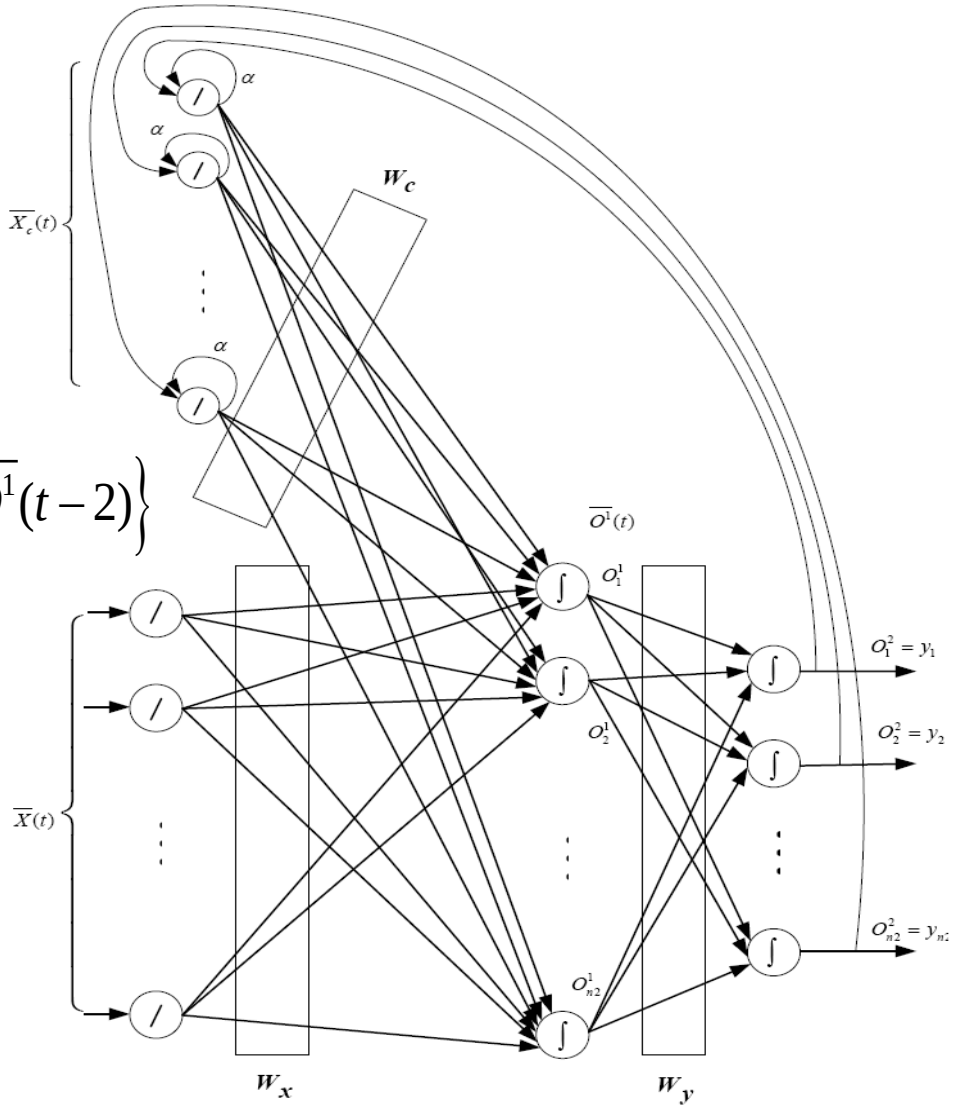
$$\Delta W_{c2} = \eta . \overline{\delta^1} . \left\{ \overline{X_{c2}}(t) + W_{c2} . \frac{\partial \overline{X_{c2}}(t)}{\partial W_{c2}} \right\}$$

$$= \eta . \overline{\delta^1} . \left\{ \overline{O^2}(t-1) + W_{c2} . \frac{\partial \overline{O^2}(t-1)}{\partial W_{c2}} \right\}$$

Jordan with Neuron Feedback

$$\begin{cases} \overline{X_c}(t) = \overline{O^1}(t-1) + \alpha \overline{X_c}(t-1) \\ \qquad\quad = \overline{O^1}(t-1) + \alpha \overline{O^1}(t-2) \end{cases}$$

$$\begin{cases} \overline{net^1}(t) = W_x . \overline{X}(t) + W_c . \overline{X_c}(t) \\ \qquad\quad = W_x . \overline{X}(t) + W_c . \left\{ \overline{O^1}(t-1) + \alpha \overline{O^1}(t-2) \right\} \end{cases}$$



Jordan with Neuron Feedback

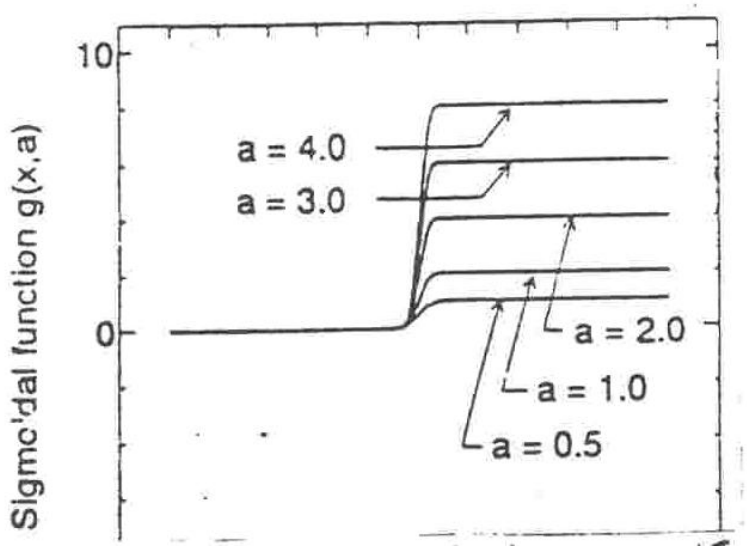
$$\begin{cases} \overline{net^1}(t) = \boldsymbol{W}_x.\overline{X}(t) + \boldsymbol{W}_c.\overline{X_c}(t) \\ \qquad\qquad = \boldsymbol{W}_x.\overline{X}(t) + \boldsymbol{W}_c.\left\{\overline{O^1}(t-1) + \alpha\overline{O^1}(t-2)\right\} \\ \Delta \boldsymbol{W}_c = -\eta.\dfrac{\partial E}{\partial \boldsymbol{W}_c} = -\eta.\dfrac{\partial E}{\partial \overline{O^2}}.\dfrac{\partial \overline{O^2}}{\partial \overline{net^2}}.\dfrac{\partial \overline{net^2}}{\partial \overline{O^1}}.\dfrac{\partial \overline{O^1}}{\partial \overline{net^1}}.\dfrac{\partial \overline{net^1}}{\partial \boldsymbol{W}_c} \\ \qquad\qquad = \eta.\bar{e}.\overline{F^{2\prime}}(.).\boldsymbol{W}_y.\overline{F^{1\prime}}(.).\left\{\overline{X_c}(t) + \boldsymbol{W}_c.\dfrac{\partial \overline{X_c}(t)}{\partial \boldsymbol{W}_c}\right\} \end{cases}$$

Flexible NN

Neuron Functions (Unipolar)

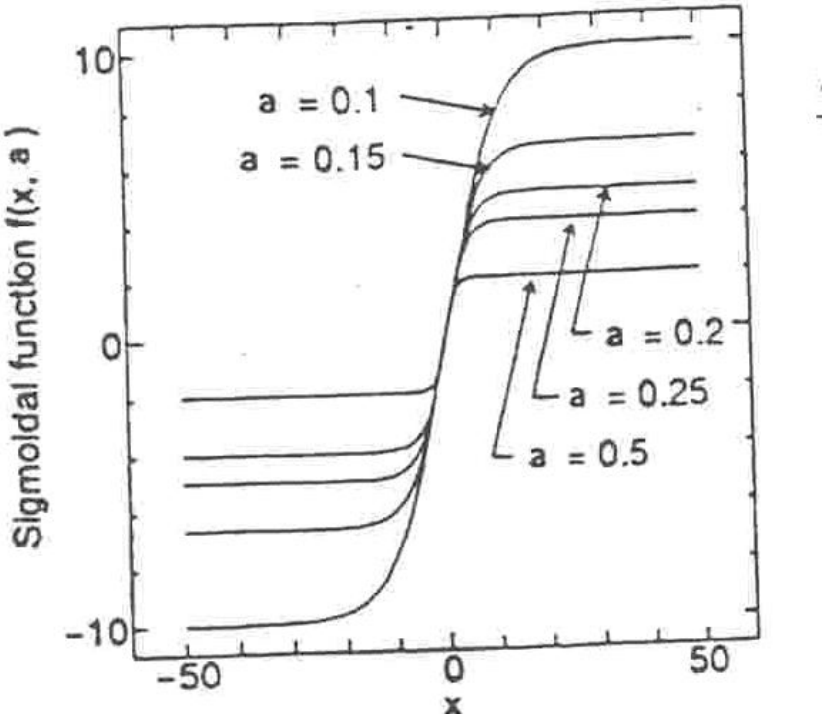
$$f(x) = \frac{1}{1 + \exp(-gx)}$$

$$F(a, x) = \frac{2|a|}{1 + e^{-2|a|.x}}$$



Flexible NN

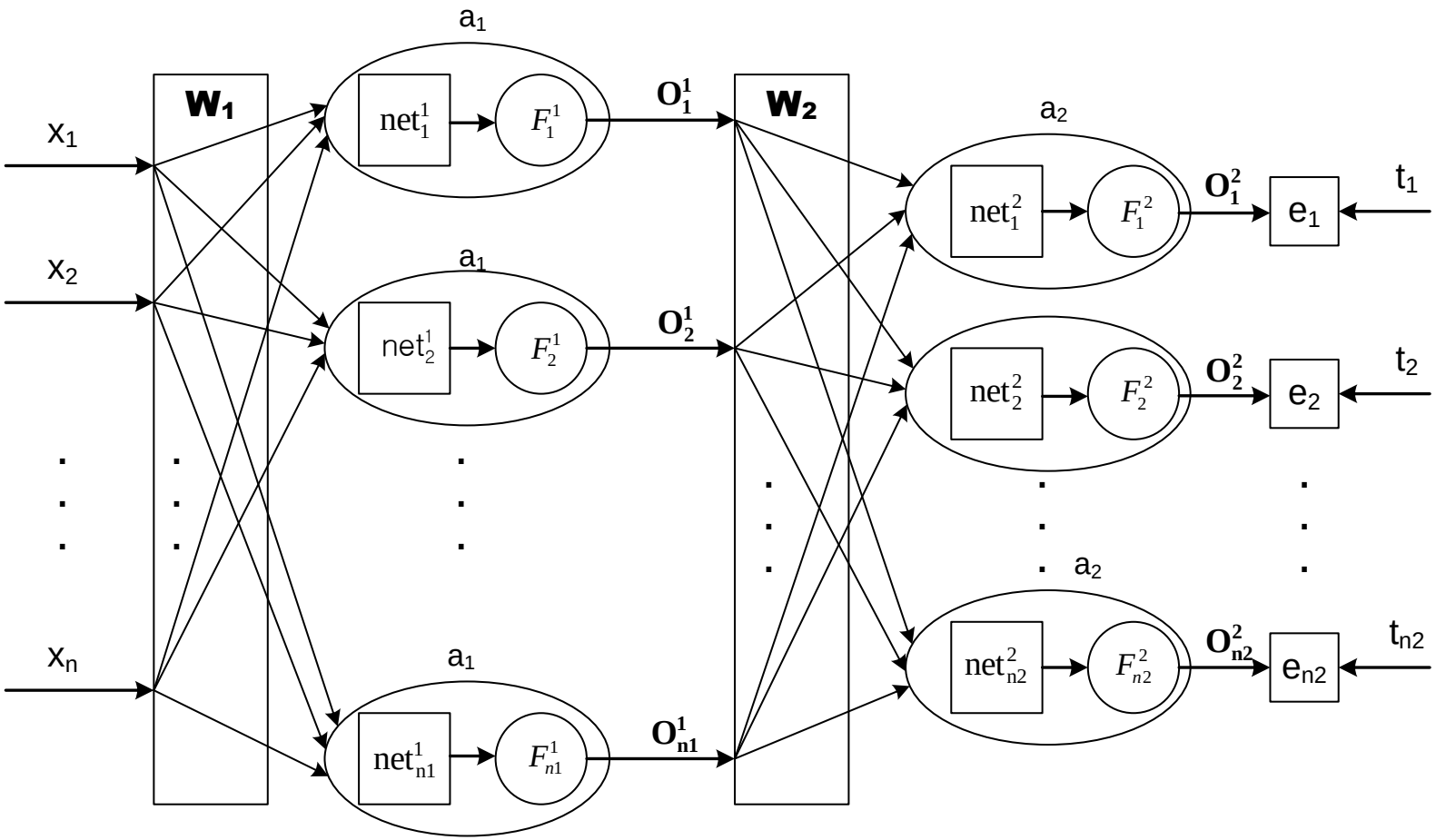
Neuron Functions (Bipolar)



$$g(x) = \frac{1 - \exp(-gx)}{1 + \exp(-gx)}$$

$$F(a,x) = \frac{1 - e^{-2|a|.x}}{1 + e^{-2|a|.x}}$$

Flexible NN Structure



Feed Forward of FNN

$$\overline{net^1}(k) = \mathbf{W_1^T}.\overline{X} = \begin{bmatrix} net_1^1 \\ net_2^1 \\ \vdots \\ net_{n1}^1 \end{bmatrix} \qquad \overline{net^2}(k) = \mathbf{W_2^T}.\overline{O^1} = \begin{bmatrix} net_1^2 \\ net_2^2 \\ \vdots \\ net_{n1}^2 \end{bmatrix}$$

$$\overline{O^1}(k) = \overline{F^1}(\overline{a^1}, \overline{net^1}(k)) \qquad \overline{O^2}(k) = \overline{F^2}(\overline{a^2}, \overline{net^2}(k))$$

$$\begin{bmatrix} O_1^2 \\ O_2^2 \\ \vdots \\ O_{n2}^2 \end{bmatrix} = \begin{bmatrix} F_1^2(a^2, net_1^2) & 0 & \cdots & 0 \\ 0 & F_2^2(a^2, net_2^2) & 0 & 0 \\ \vdots & 0 & \ddots & \vdots \\ 0 & 0 & \cdots & F_{n1}^2(a^2, net_{n1}^2) \end{bmatrix}$$

Learning

$$E = \frac{1}{2}\sum_{i=1}^{n2} e_i^2 = \frac{1}{2}\sum_{i=1}^{n2}(t_i - o_i^2)^2 \qquad \Delta a(k) = -\eta.\frac{\partial E}{\partial a(k)}$$

$$\overline{\Delta a^2}(k) = -\eta.\frac{\partial E}{\partial \overline{a^2}(k)} = -\eta\frac{\partial E}{\partial \overline{O^2}(k)}\frac{\partial \overline{O^2}(k)}{\partial \overline{a^2}(k)} = -\eta.(-\bar{e}).\overline{F_2'}(\overline{a^2}, \overline{net^2}) = \eta.\bar{e}.\overline{F_2'}(\overline{a^2}, \overline{net^2})$$

$$\overline{F'^2}(\overline{a^2}, \overline{net^2}) = \frac{(2\overline{net^2}e^{-2\overline{a^2}.*\overline{net^2}})(1+e^{-2\overline{a^2}.*\overline{net^2}}).*\overline{a^2} - [(1+e^{-2\overline{a^2}.*\overline{net^2}}) + \overline{a^2}.*(-2\overline{net^2}e^{-2\overline{a^2}.*\overline{net^2}})].*(1-e^{-2\overline{a^2}.*\overline{net^2}})}{(\overline{a^2})^2.*(1+e^{-2\overline{a^2}.*\overline{net^2}})^2}$$

$$= \frac{2\overline{a^2}\,\overline{net^2}e^{-2\overline{a^2}.*\overline{net^2}}(1+e^{-2\overline{a^2}.*\overline{net^2}}+1-e^{-2\overline{a^2}.*\overline{net^2}})}{(\overline{a^2})^2.*(1+e^{-2\overline{a^2}.*\overline{net^2}})^2} - \frac{(1-e^{-2\overline{a^2}.*\overline{net^2}})(1+e^{-2\overline{a^2}.*\overline{net^2}})}{(\overline{a^2})^2.*(1+e^{-2\overline{a^2}.*\overline{net^2}})^2}$$

$$= \frac{4\overline{net^2}e^{-2\overline{a^2}.*\overline{net^2}}}{\overline{a^2}.*(1+e^{-2\overline{a^2}.*\overline{net^2}})^2} - \frac{1}{=}\overline{F^2}(\overline{a^2}, \overline{net^2})$$

Learning

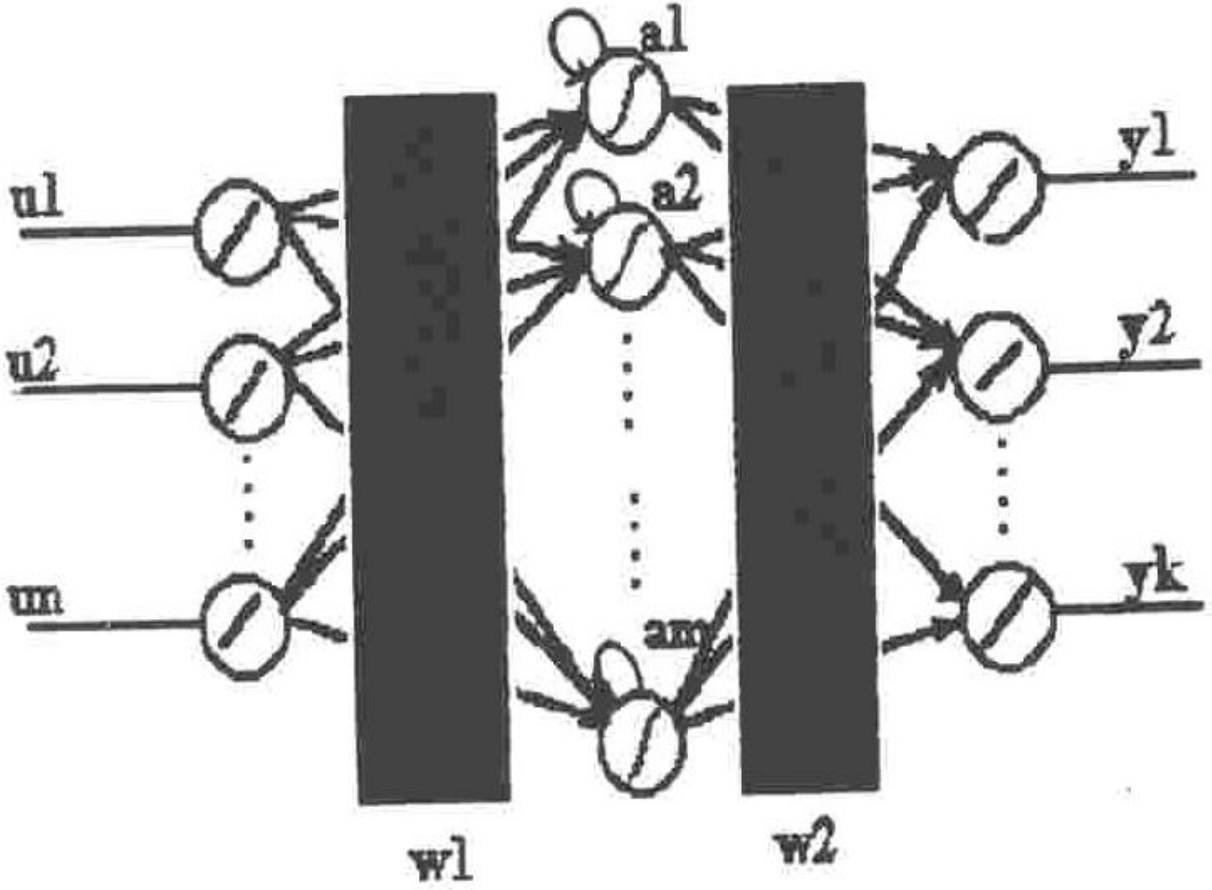
$$E = \frac{1}{2}\sum_{i=1}^{n2} e_i^2 = \frac{1}{2}\sum_{i=1}^{n2}(t_i - o_i^2)^2 \qquad \Delta a(k) = -\eta . \frac{\partial E}{\partial a(k)}$$

$$\overline{\Delta a^1}(k) = -\eta . \frac{\partial E}{\partial \overline{a^1}(k)} = -\eta \frac{\partial E}{\partial \overline{O^2}(k)} \frac{\partial \overline{O^2}(k)}{\partial \overline{net^2}(k)} \frac{\partial \overline{net^2}(k)}{\partial \overline{O^1}(k)} \frac{\partial \overline{O^1}(k)}{\partial \overline{a^1}(k)}$$

$$= \eta . \overline{e} . \underbrace{\overline{F_2'}(\overline{a^2}, \overline{net^2})}_{\text{Derived respect to } \overline{net^2}} . \mathbf{W^2} . \underbrace{\overline{F_1'}(\overline{a^1}, \overline{net^1})}_{\text{Derived respect to } \overline{a^2}}$$

Recurrent Flexible NN

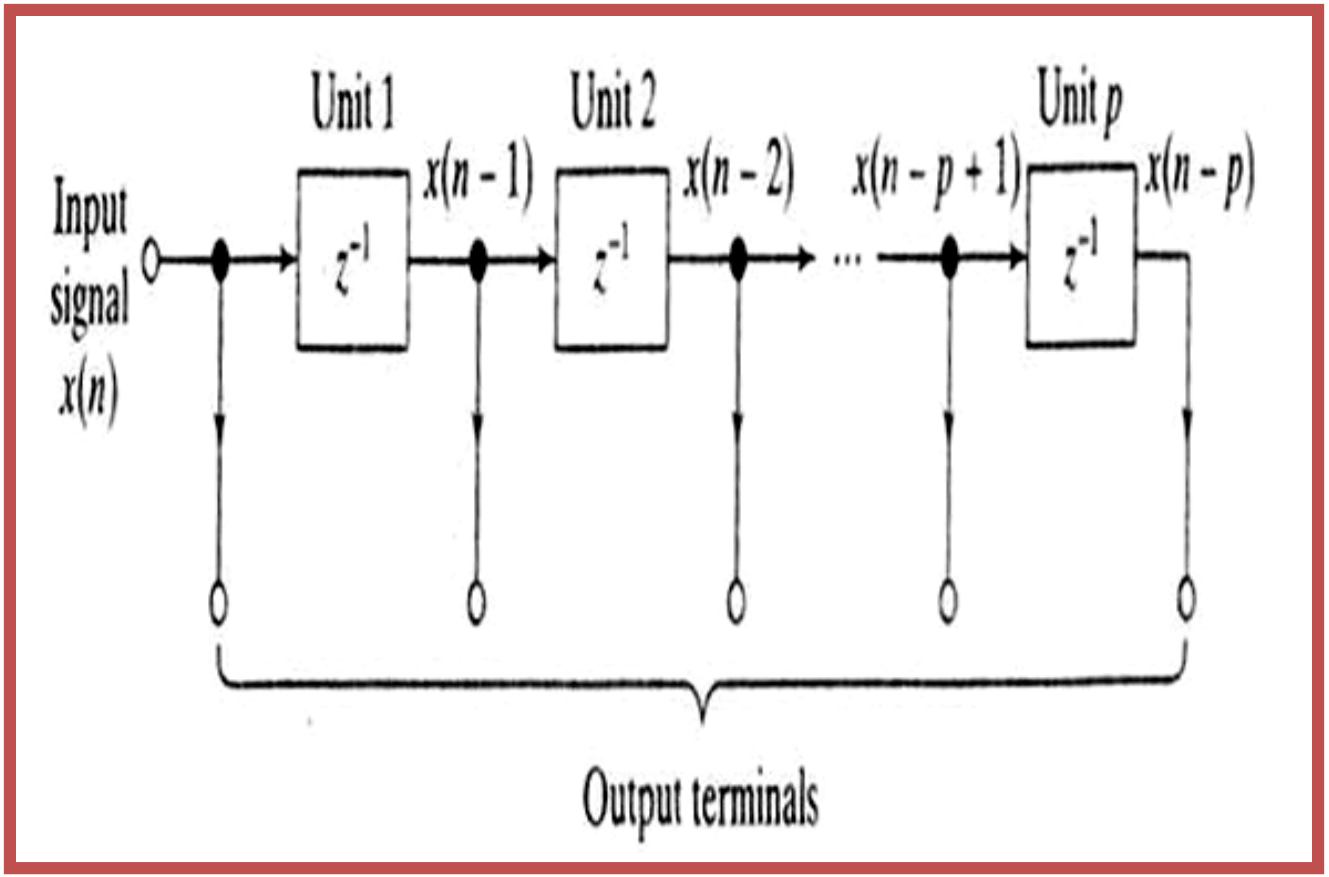
$$\Delta a(k) = -\eta . \frac{\partial E}{\partial a(k)}$$

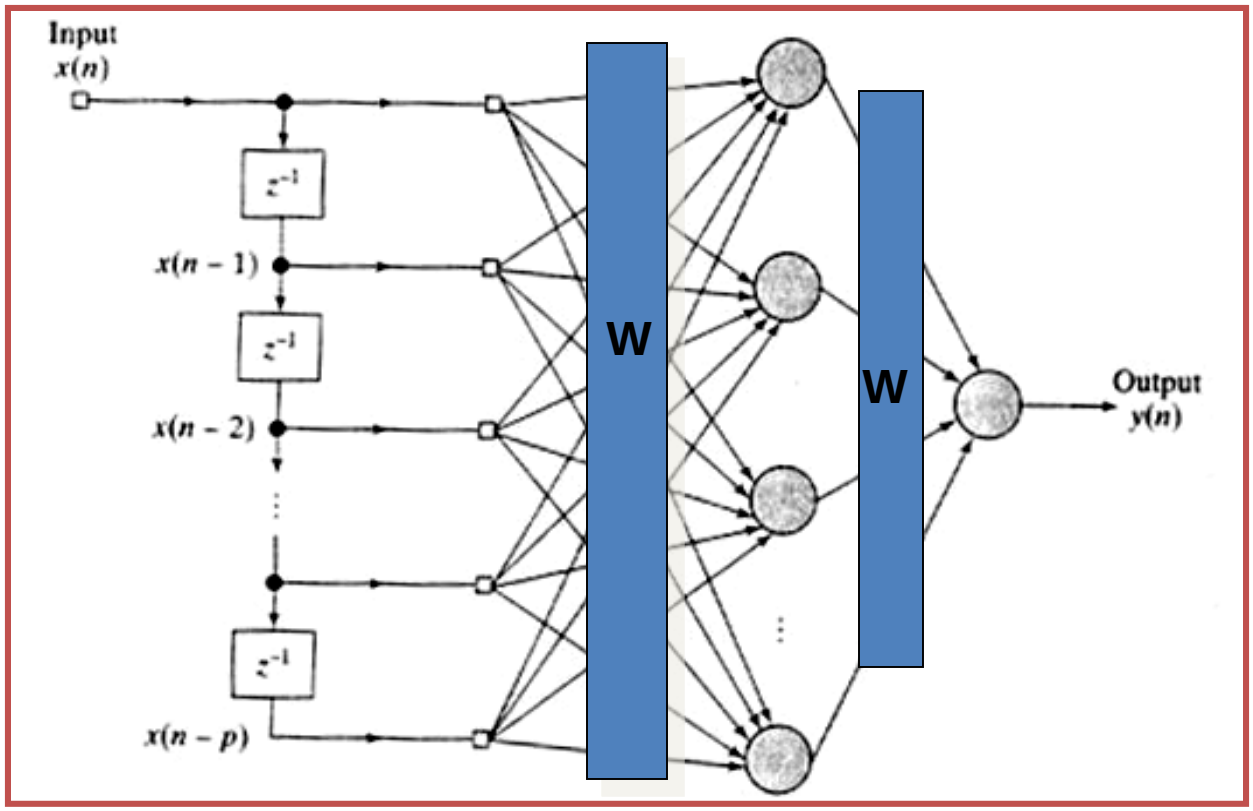


فصل هفتم :شبکه های حافظه دار TDLV و گاما

حافظه درشبکه عصبی

- ☑ شبکه‌های عصبی می توانند بهتر از روشهای کلاسیک خطی مشخصات پیچیده سری‌های زمانی را مدل کنند .
- ☑ به صورت تئوری نیازی به این که آیا سری زمانی ایستا است یا خیر جهت مدلسازی با شبکه‌های عصبی نداریم .
- ☑ شبکه‌های عصبی نیاز به تعداد عظیمی از داده‌های به عنوان ورودی ندارند بر عکس روشAR .


Unit 1
Unit 2
Unit p
Input signal x(n)
x(n - 1)
x(n - 2)
x(n - p + 1)
x(n - p)
z^-1
z^-1
...
z^-1
Output terminals


Input
x(n)
z⁻¹
x(n − 1)
z⁻¹
x(n − 2)
z⁻¹
x(n − p)
W
W
Output
y(n)

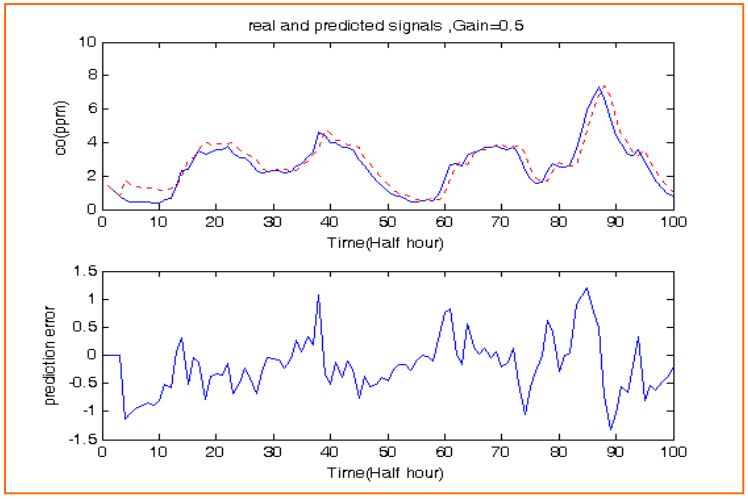
پیش بینی با کمک MLP

شکل اعمال ورودی

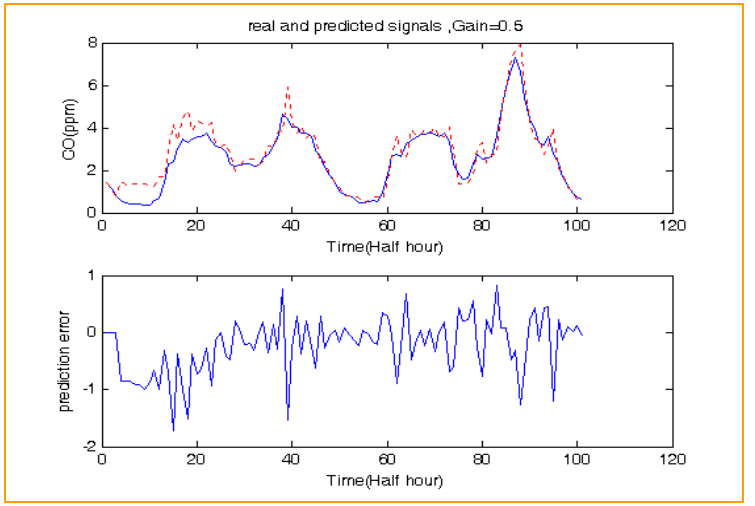
$$X = \left[x(n) \;\; x(n)-x(n-1) \;\; x(n-1) \;\; x(n-1)-x(n-2) \cdots \right]^T$$

شکل اعمال خروجی

$x(n+1)-x(n)$



$x(n+1)-2x(n)+x(n-1)$



شبکه های حافظه دار TDL وگاما

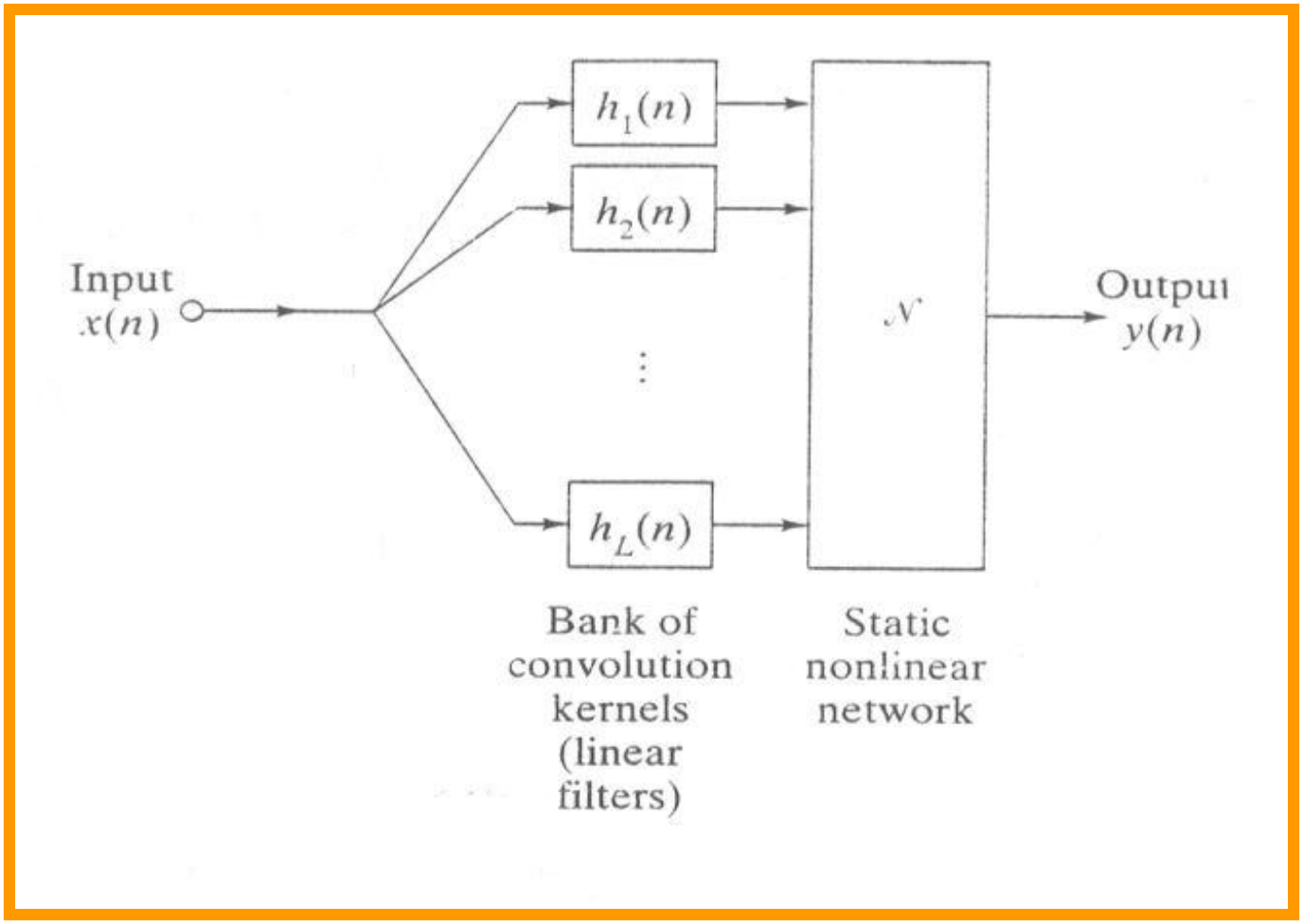
- ***سند برگ (۱۹۹۷) :*** هر تغییر زمانی در یک نگاشت دینامیکی می‌تواند به طور تقریباً کاملی توسط یک ساختاری که دارای ۲ بلوک است که بلوک اول بانک فیلترهای خطی شبکه می باشد و بلوک استاتیکی بعدی خود را تغذیه می‌کند مدل شود.
- فیلترهای ***FIR , IIR***

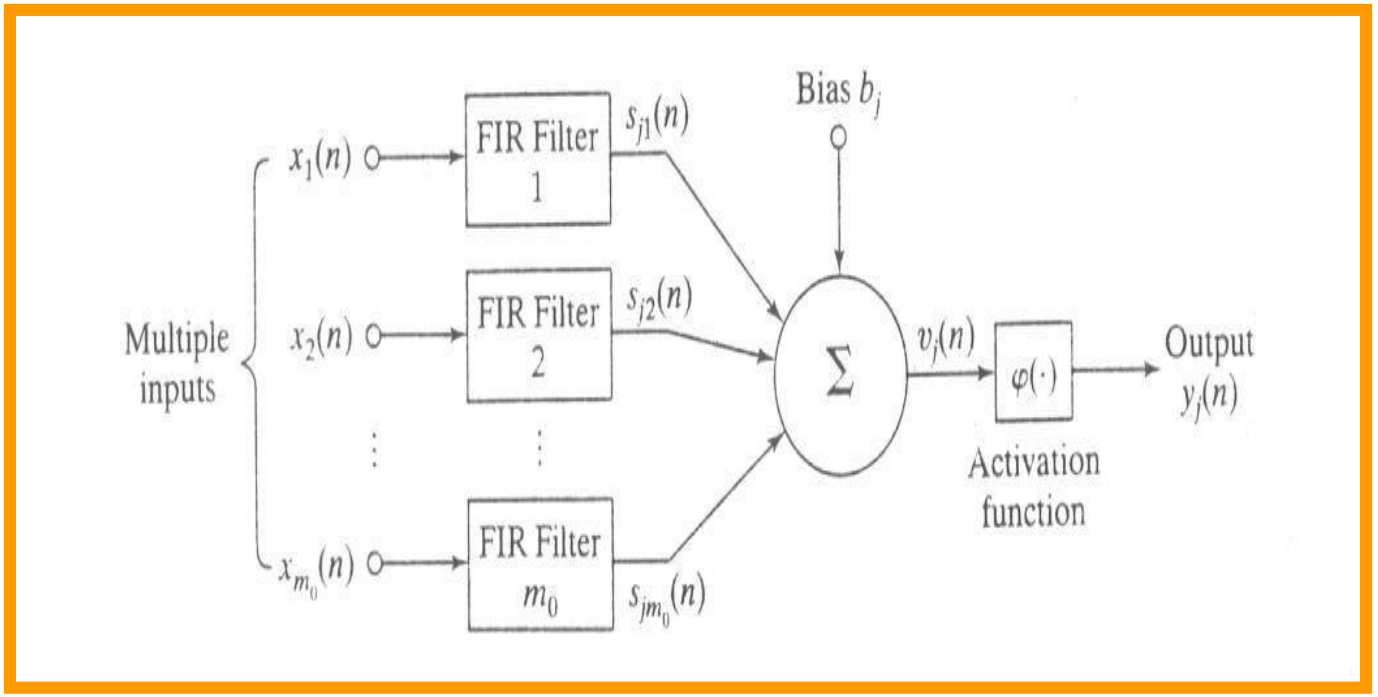
مشکلات

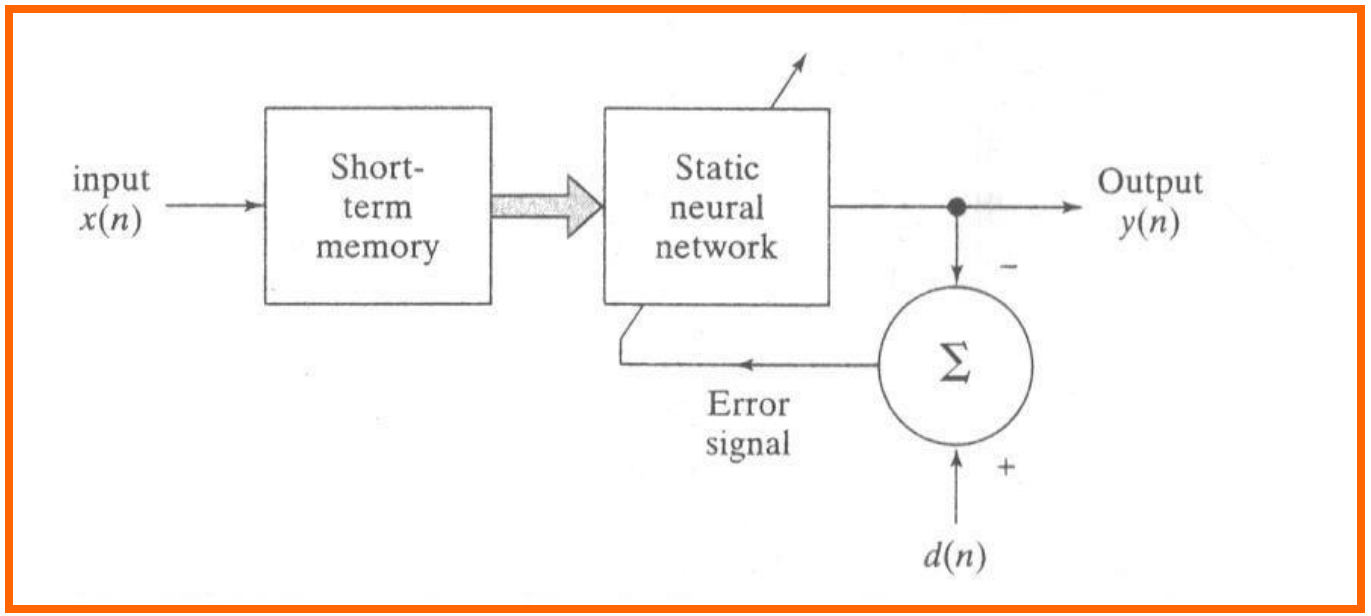
- ***IIR : بوجود آمدن ناپایداری ها در آموزش***
- ***FIR : ظرفیت محدود مدلسازی***

مزایا

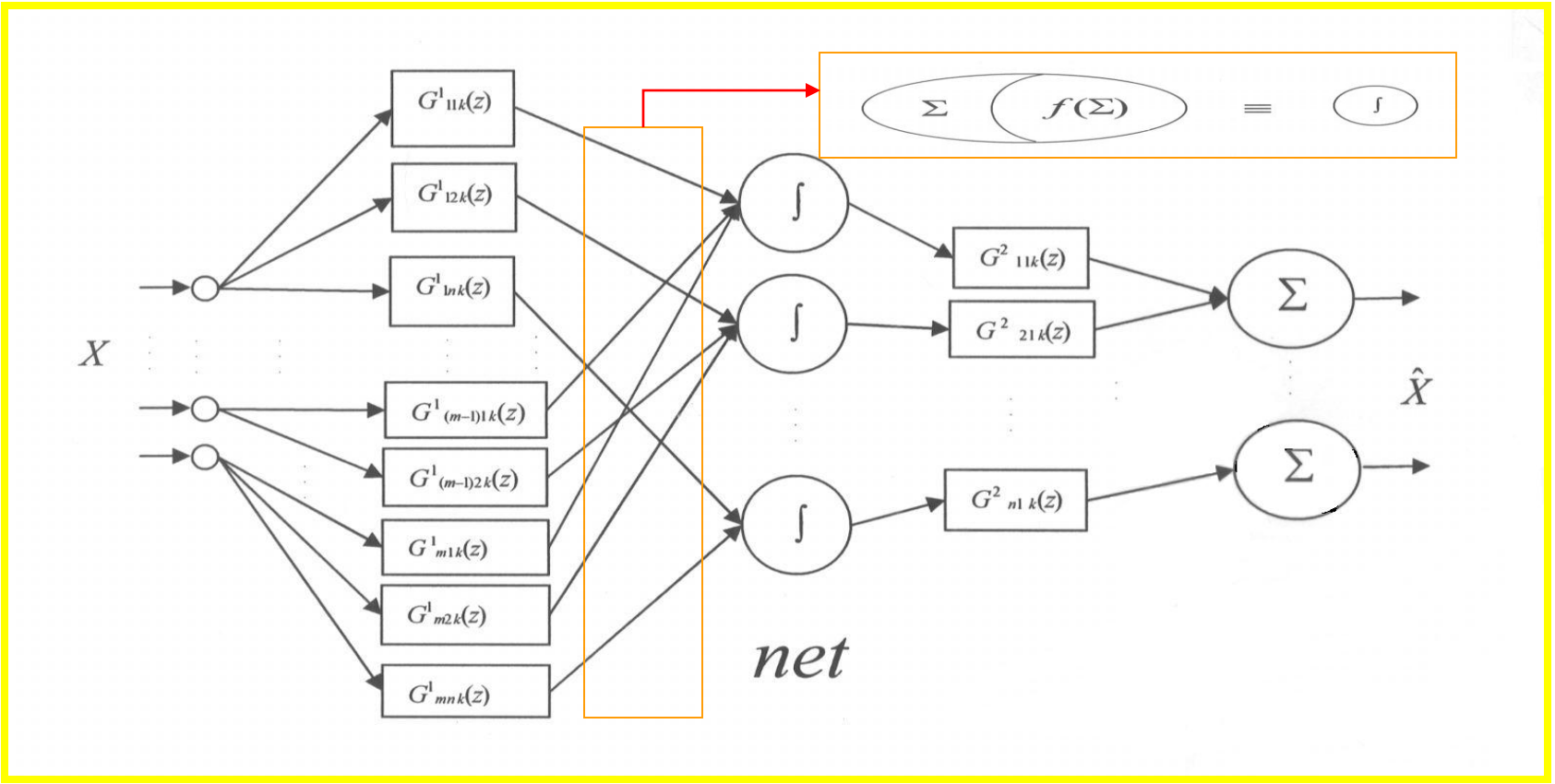
- ***IIR : کم بودن پارامترها در حوزه های بزرگ زمان***
- ***FIR : پایداری شبکه***


$h_1(n)$
$h_2(n)$
$h_L(n)$
Input
$x(n)$
$\mathcal{N}$
Output
$y(n)$
Bank of convolution kernels (linear filters)
Static nonlinear network


Bias b_j
$x_1(n)$
FIR Filter 1
$s_{j1}(n)$
Multiple inputs
$x_2(n)$
FIR Filter 2
$s_{j2}(n)$
$x_{m_0}(n)$
FIR Filter m_0
$s_{jm_0}(n)$
Σ
$v_j(n)$
$\varphi(\cdot)$
Output $y_j(n)$
Activation function


input $x(n)$
Short-term memory
Static neural network
Output $y(n)$
Σ
Error signal
$d(n)$

اعمال حافظه به لایه های میانی و خروجی



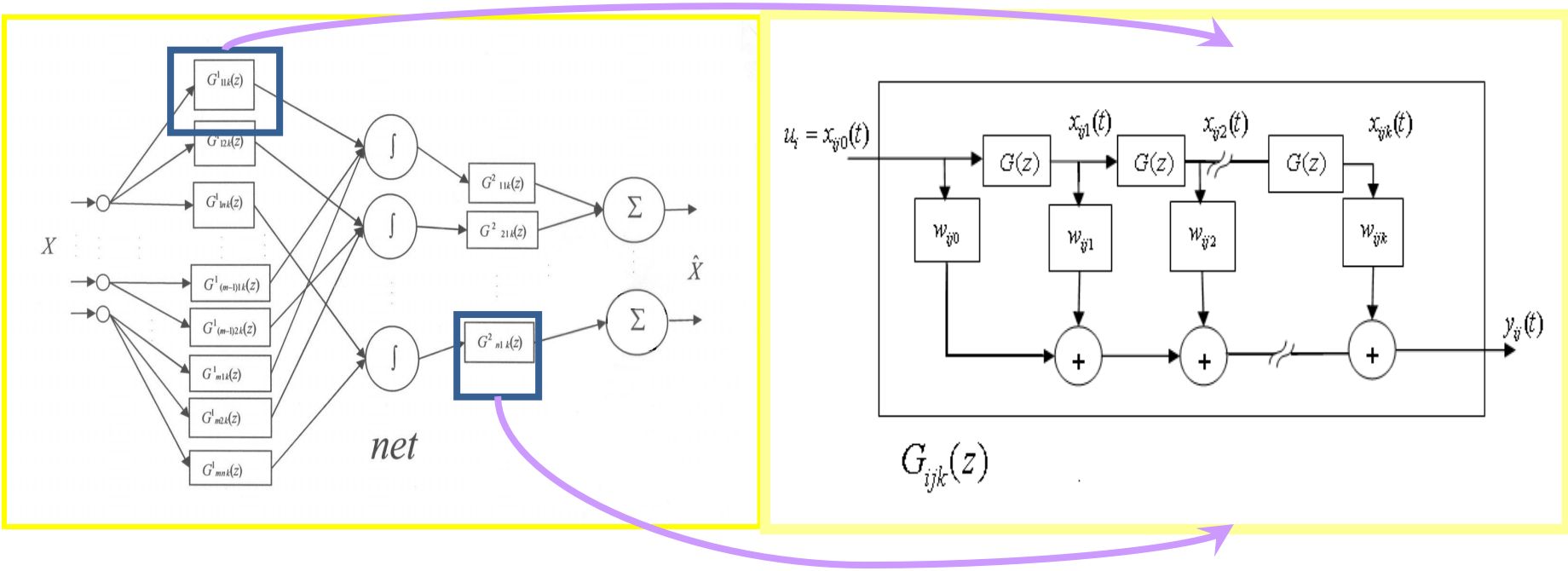
$$X = [x(t)\ ,\ x(t-1)\ ,\ \cdots\ ,\ x(t-m+1)]^{T}$$

$$\hat{X}(t+1) = [O(t)]_{1\times p} = \left[G^{2^{T}}\right]_{1\times n} \cdot \left[F(Net^{1})\right]_{n\times p}$$

$$\left[Net^{1}\right]_{n\times 1} = \left[G^{T}\right]_{n\times m} \cdot [X]_{m\times 1}$$

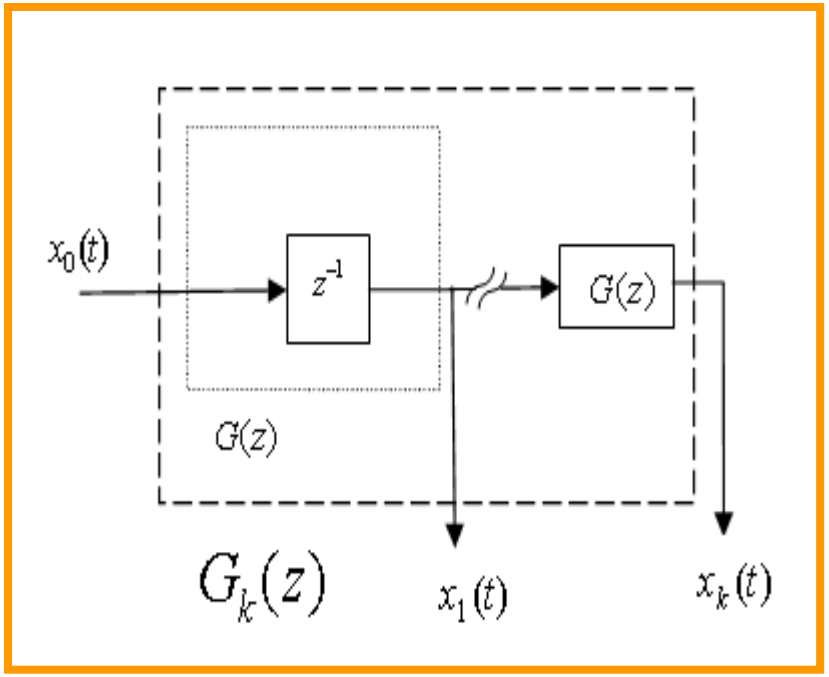
$$F\left(\left[Net^{1}\right]_{n\times 1}\right) = F\left(\left[G^{T}\right]_{n\times m} \cdot [X]_{m\times 1}\right)$$

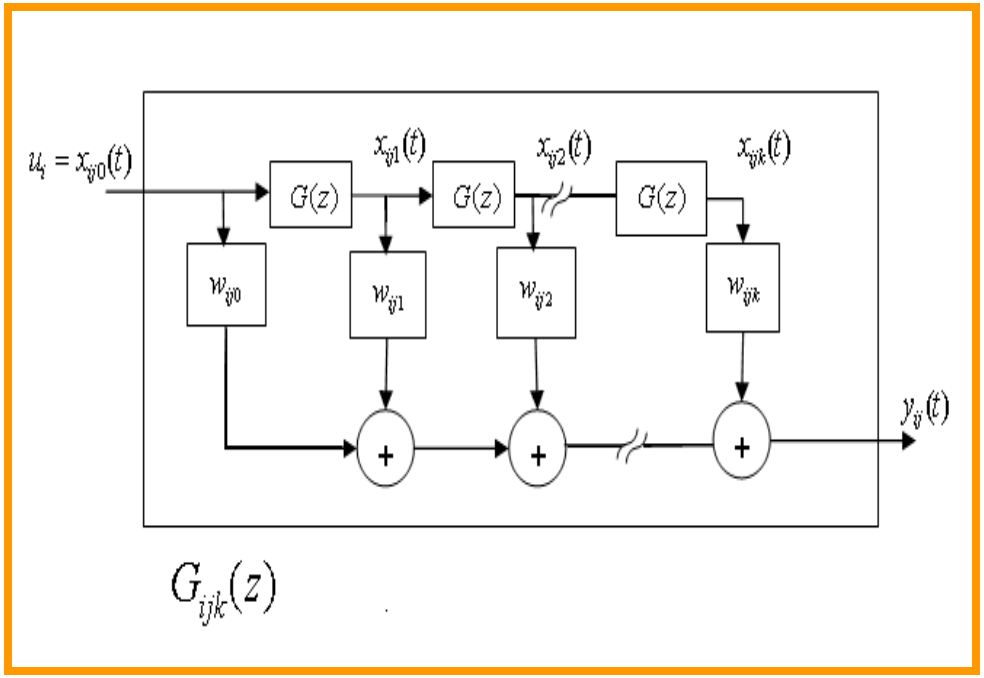
مزایا و توپولوژی اعمال فیلتر



- ☑ داشتن بیش از یک وزن تحت آموزش در هر سیناپس
- ☑ در دسترس بودن داده های زمانهای گذشته
- ☑ آموزش پارامترهای فیلتر دیجیتال در صورت وجود

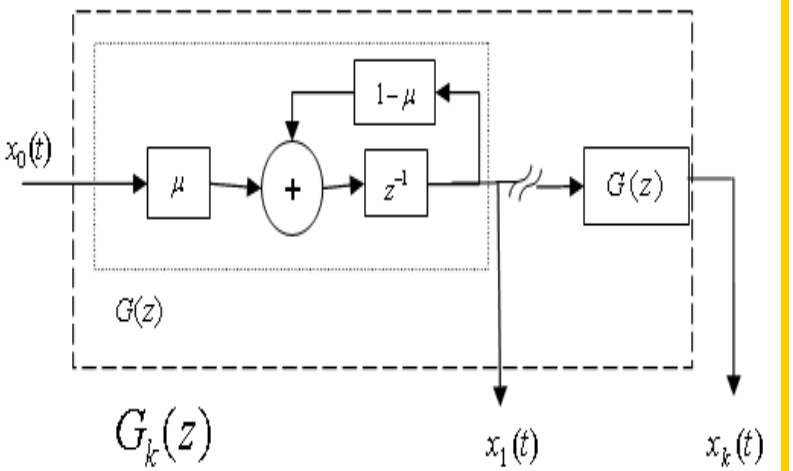
توپولوژی و محاسبات پیشرو TDL

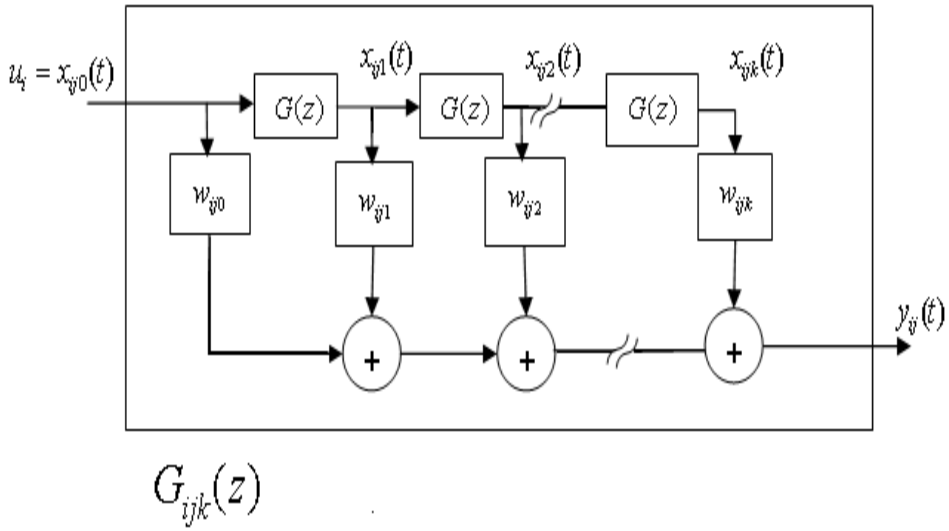




$$
\begin{aligned}
x_{ij1}(t) &= x_{ij0}(t-1)\\
x_{ij2}(t) &= x_{ij1}(t-1)\\
x_{ij3}(t) &= x_{ij2}(t-1)\\
&\vdots \qquad\quad \vdots\\
x_{ijk}(t) &= x_{ij,k-1}(t-1)
\end{aligned}
\longrightarrow
y_{ij}(t) = w_{ij0}x_{ij0}(t) + w_{ij1}x_{ij1}(t) + \cdots + w_{ijk}x_{ijk}(t) = \sum_{l=0}^{k} w_{ijl}x_{ijl}(t)
$$

توپولوژی و محاسبات پیشرو TDL





$$x_1(t) = G(z)\cdot x_0(t) = \frac{\mu z^{-1}}{1-(1-\mu)z^{-1}}\cdot x_0(t)$$

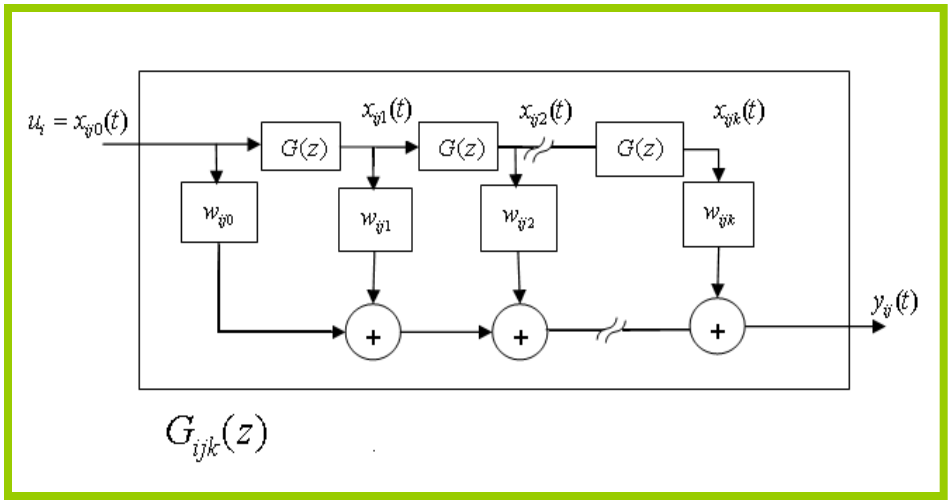
$$x_1(t) - (1-\mu)x_1(t-1) = \mu x_0(t-1)$$

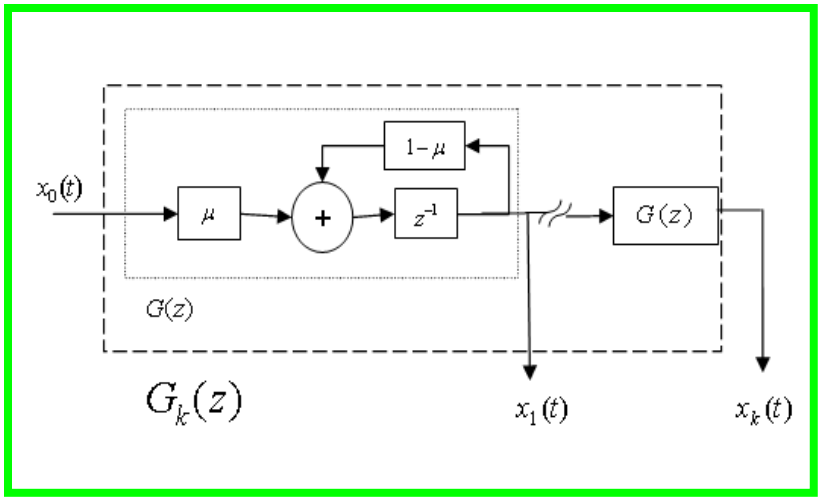
$$x_1(t) = (1-\mu)x_1(t-1) + \mu x_0(t-1)$$

$$x_{ijl}(t) = (1-\mu_{ij})\cdot x_{ijl}(t-1) + \mu_{ij}\cdot x_{ij,l-1}(t-1)$$

$$y_{ij}(t) = w_{ij0}x_{ij0}(t) + w_{ij1}x_{ij1}(t) + \cdots + w_{ijk}x_{ijk}(t) = \sum_{l=0}^{k} w_{ijl}x_{ijl}(t)$$

آموزش شبکه گاما (BP)





$$\frac{\partial y_{ij}(t)}{\partial \mu_{ij}} = \sum_{l=0}^{k} w_{ijl} \cdot \frac{\partial x_{ijl}(t)}{\partial \mu_{ij}} = \sum_{l=0}^{k} w_{ijl} \cdot \alpha_{ijl}(t)$$

$$\frac{\partial x_{ijl}(t)}{\partial \mu_{ij}} = \alpha_{ijl}(t) = (1-\mu_{ij})\alpha_{ijl}(t-1) + \mu_{ij}\alpha_{ij,l-1}(t-1) + x_{ij,l-1}(t) - x_{ijl}(t-1)$$

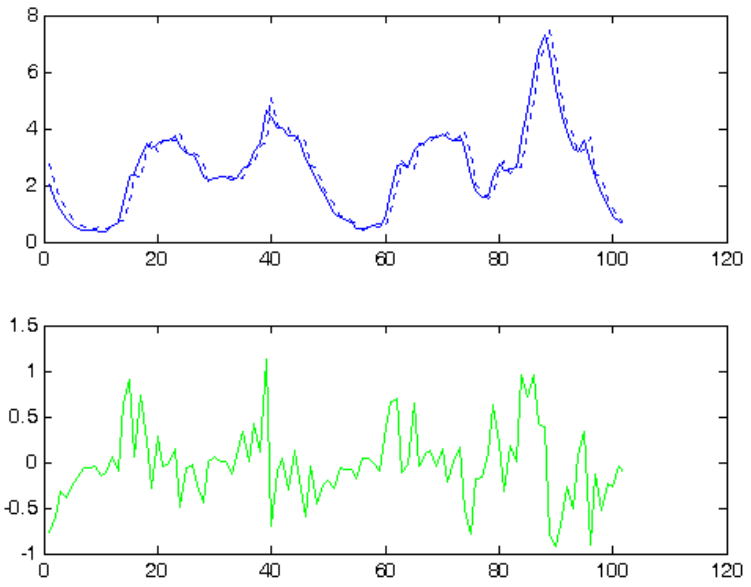
$$\frac{\partial y_{ij}}{\partial w_{ijl}(t)} = x_{ijl}(t) \qquad \frac{\partial y_{ij}(t)}{\partial x_{ij0}} = \frac{\partial y_{ij}(t)}{\partial u_i} = w_{ij0}$$

$$\alpha_{ijl}(t) = \frac{\partial(1-\mu_{ij})}{\partial \mu_{ij}} \cdot x_{ijl}(t-1) + \frac{\partial x_{ijl}(t-1)}{\partial \mu_{ij}} \cdot (1-\mu_{ij}) + \frac{\partial \mu_{ij}}{\partial \mu_{ij}} \cdot x_{ij,l-1}(t-1) + \frac{\partial x_{ij,l-1}(t-1)}{\partial \mu_{ij}} \cdot \mu_{ij} =$$

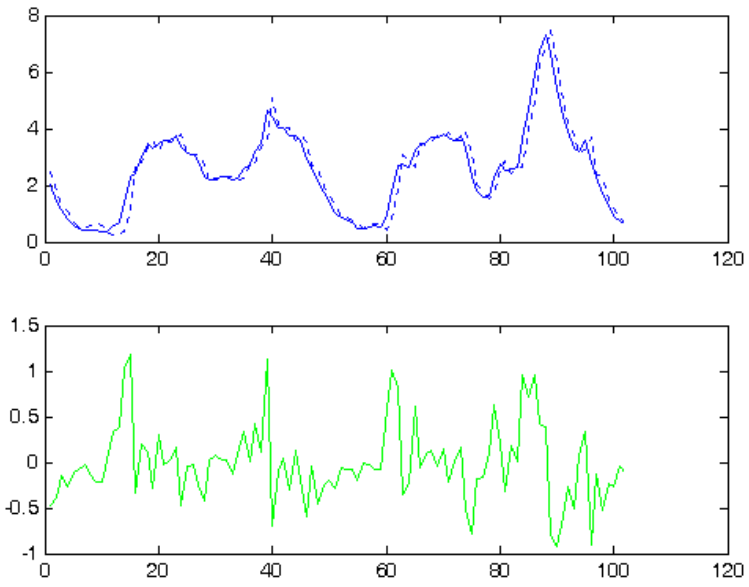
$$-x_{ijl}(t-1) + \frac{\partial x_{ijl}(t-1)}{\partial \mu_{ij}} \cdot (1-\mu_{ij}) + x_{ij,l-1}(t-1) + \frac{\partial x_{ij,l-1}(t-1)}{\partial \mu_{ij}} \cdot \mu_{ij}$$

نمودار پیش بینی با شبکه های حافظه دار

شبکه گاما



شبکهTDL



مقایسه روشها

مقایسه کلیه روشها برای منوکسید کربن					
CO	AR	MLP1	MLP2	TDL	Gamma
RMSE	۲/۳۲۸۷	۰/۴۹۶۰	۰/۴۴۴۶	۰/۴۱۵۹	۰/۴۰۰۳
NMSE	۰/۸۱۶۸	۰/۱۱۹۴	۰/۰۹۵۹	۰/۰۷۸۶	۰/۰۷۲۹
MB	-۰/۴۹۷۳	۰/۱۲۴۹	۰/۱۱۹۹	۰/۰۱۲۰	۰/۰۲۵۹
MA	۱/۰۴۵۴	۰/۳۸۶۰	۰/۳۲۷۸	۰/۳۰۰۶	۰/۲۸۹۸

مقایسه کلیه روشهابرای منوکسیدنیتروژن					
NO	AR	MLP1	MLP2	TDL	Gamma
RMSE	۲۶/۳۵۷۸	۱۴/۴۸۶۶	۱۹/۸۹۲۳	۱۳/۶۴۲۵	۱۳/۶۳۸۶
NMSE	۰/۸۶۹۷	۰/۶۰۸۶	۱/۱۹۱۸	۰/۵۳۷۶	۰/۵۳۷۳
MB	-۲/۶۳۰۲	۱/۱۰۱۶	۰/۹۲۵۳	-۰/۰۶۰۶۰	-۰/۰۵۱۵
MA	۱۴/۵۸۴۷	۸/۲۲۲	۱۱/۱۹۷۹	۷/۹۱۵۶	۷/۹۰۴۴

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{x}(i) - x(i))^2}$$

$$NMSE = \frac{\sum_{i=1}^{N}(\hat{x}(i) - x(i))^2}{\sum_{i=1}^{N}(x(i) - \bar{x}(i))^2}$$

$$MBE = \frac{1}{N}\sum_{i=1}^{N}\hat{x}(i) - x(i)$$

$$MAE = \frac{1}{N}\sum_{i=1}^{N}\left|\hat{x}(i) - x(i)\right|$$

نتیجه گیری

- روشهای سری زمانی غیرخطی شبکه های عصبی برتری خاصی برروشهای غیرشبکه عصبی دارند (همانطورکه انتظارداشتیم).
- بزرگترین مشکل در روش کلاسیک یافتن جواب بهینه به صورت سعی وخطا میباشد و شبکه های عصبی نیزاین مشکل را دریافتن تعداد لایه های میانی بهینه دارند.

- هنگامی که واحد حافظه را به داخل شبکه انتقال دادیم جوابها بسیار بهتر شدند.اما نکته مهم اینجاست که نتایج روش TDL تقریباً به گاما بسیارنزدیک بوده این امر میزان اهمیت و رجحان وزنها را برپارامترعمق حافظه می رساند.
- درشبکه های حافظه دار(TDL، گاما) اهمیت فوق العاده لایه میانی دارا می باشد.

فصل هشتم :

شبکه عصبی مدل مخچه

CMAC Neural Network and TD-CMAC

CMAC Neural Network and TD-CMAC

- Introduction
- Time Series
- Cerebellar Model Arithmetic Computer (CMAC)
- Time-Delay CMAC (TD-CMAC)
  - 1- Output Mapping
  - 2- Learning Algorithm
- Evaluation of TD-CMAC
- Conclusion

مدل مصنوعی، شبیه ساز ساختار مخچه

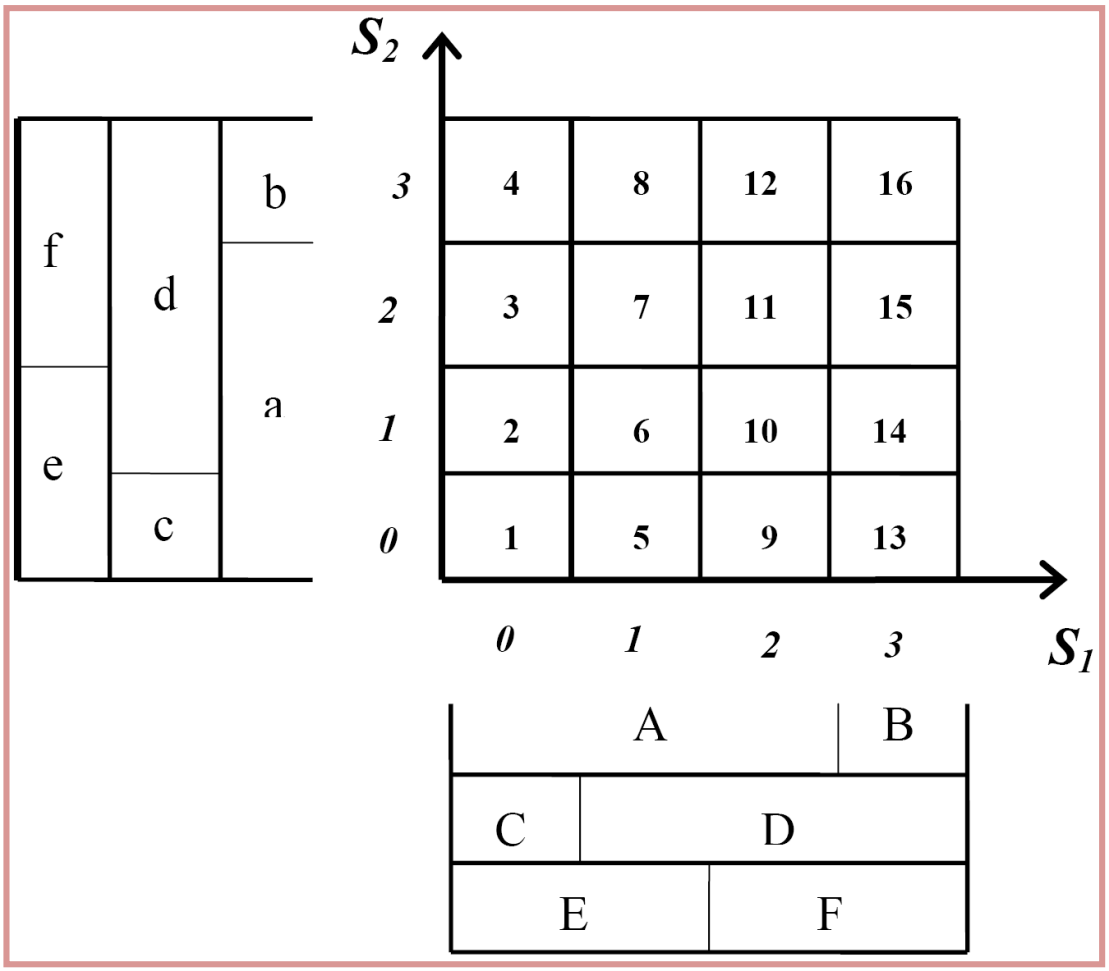
- **معماری CMAC**

کوانتیزه کردن ورودیها ⇦ بلوک

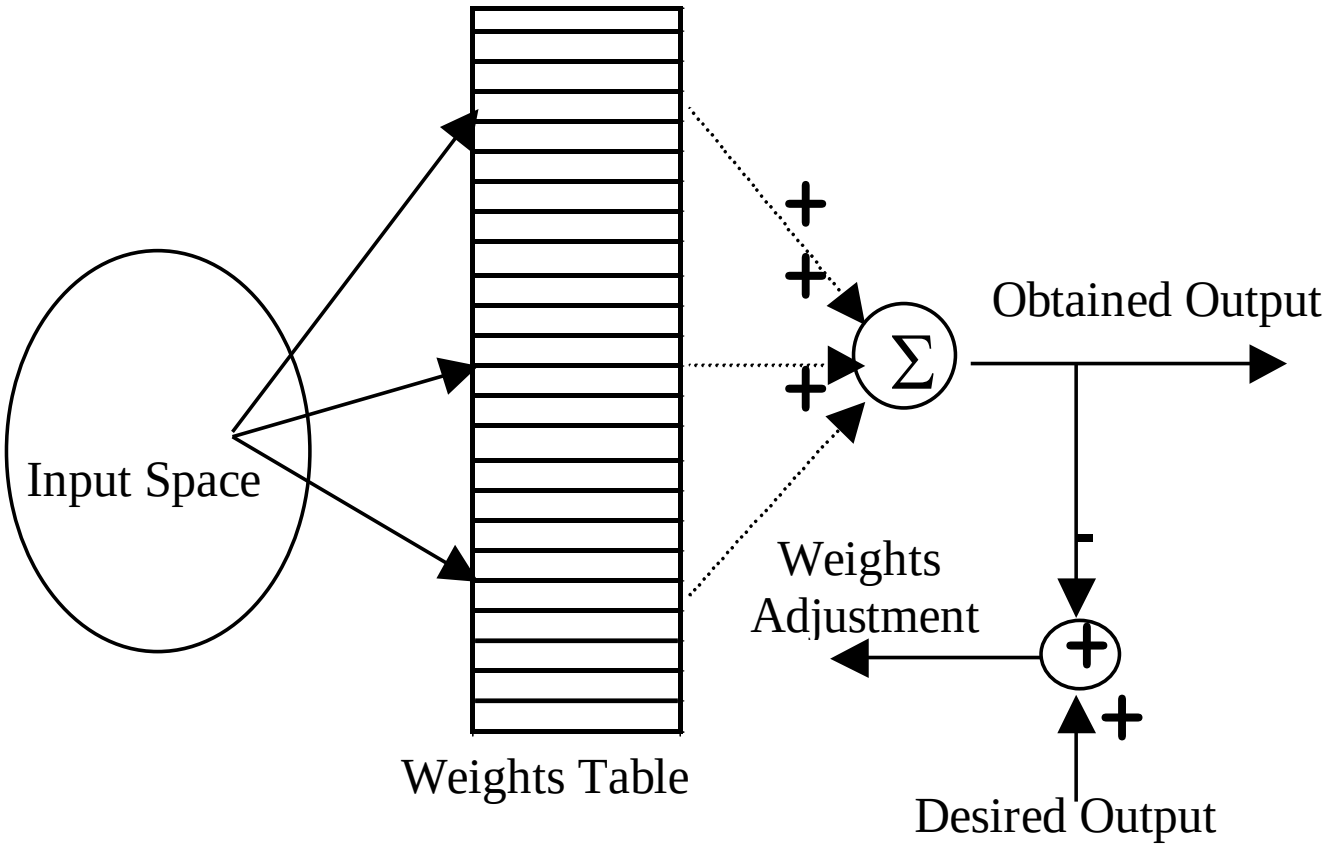
شیفت بلوکها ⇦ لایه‌ها

مساحت ایجاد شده توسط بلوکهای در لایه های متناظر ⇦ **فوق مکعب یا سلول حافظه**

- ☑ **کد کردن متغیرهای دقت بالا و انتقال روی تعداد زیادی کانالهای با دقت کم**
- ☑ **حمل تنها قسمت کوچکی از اطلاعات محتوی یک متغیر در هر کانال**
- ☑ **در حالت ماکزیمم فعال بودن هر بلوک، تنها در یک دامنه محدود**

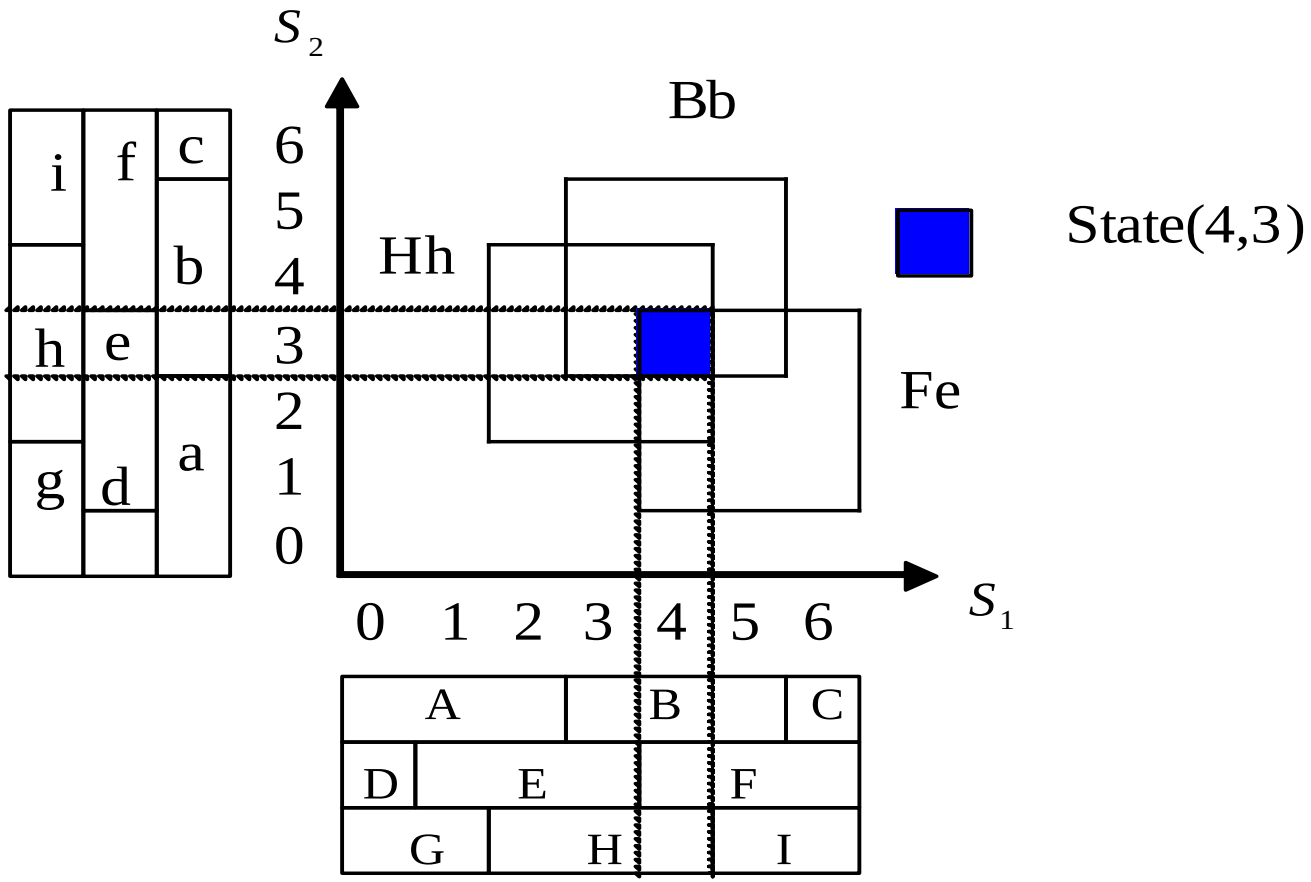

S_2
3
2
1
0
4
8
12
16
3
7
11
15
2
6
10
14
1
5
9
13
0
1
2
3
S_1
f
d
b
a
e
c
A
B
C
D
E
F

CMAC Model



Block diagram of CMAC

CMAC Model (Example)



The CMAC structure with two inputs

بکارگیری جدول جستجوی هوشمند

نگاشت خروجی و الگوریتم یادگیری در CMAC

- خروجی، مجموع وزنهای فوق مکعبهای فعال شده است.

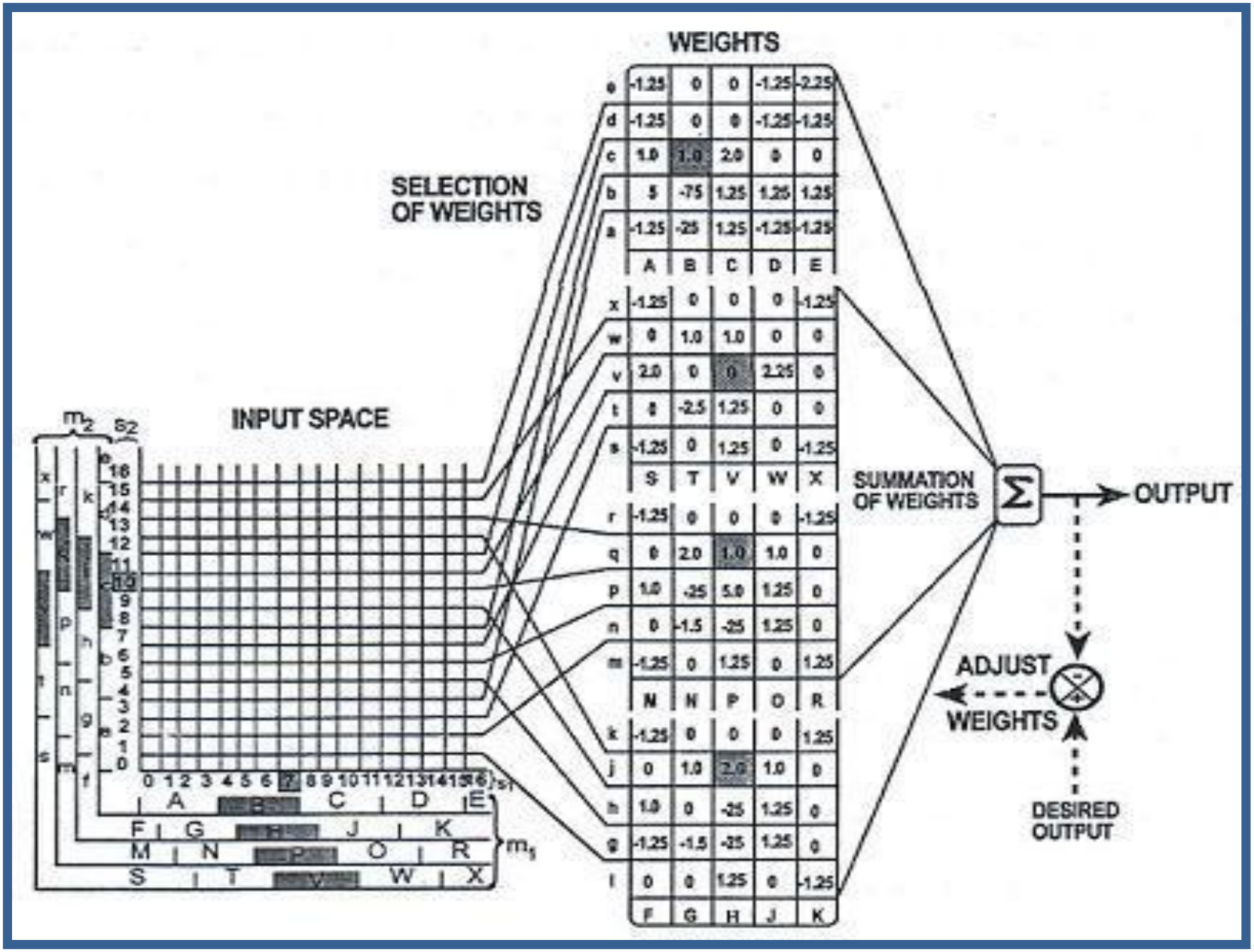
$$y(s) = a^T(s)\mathbf{W} = \sum_{j=1}^{M} a_j(s) W_j$$

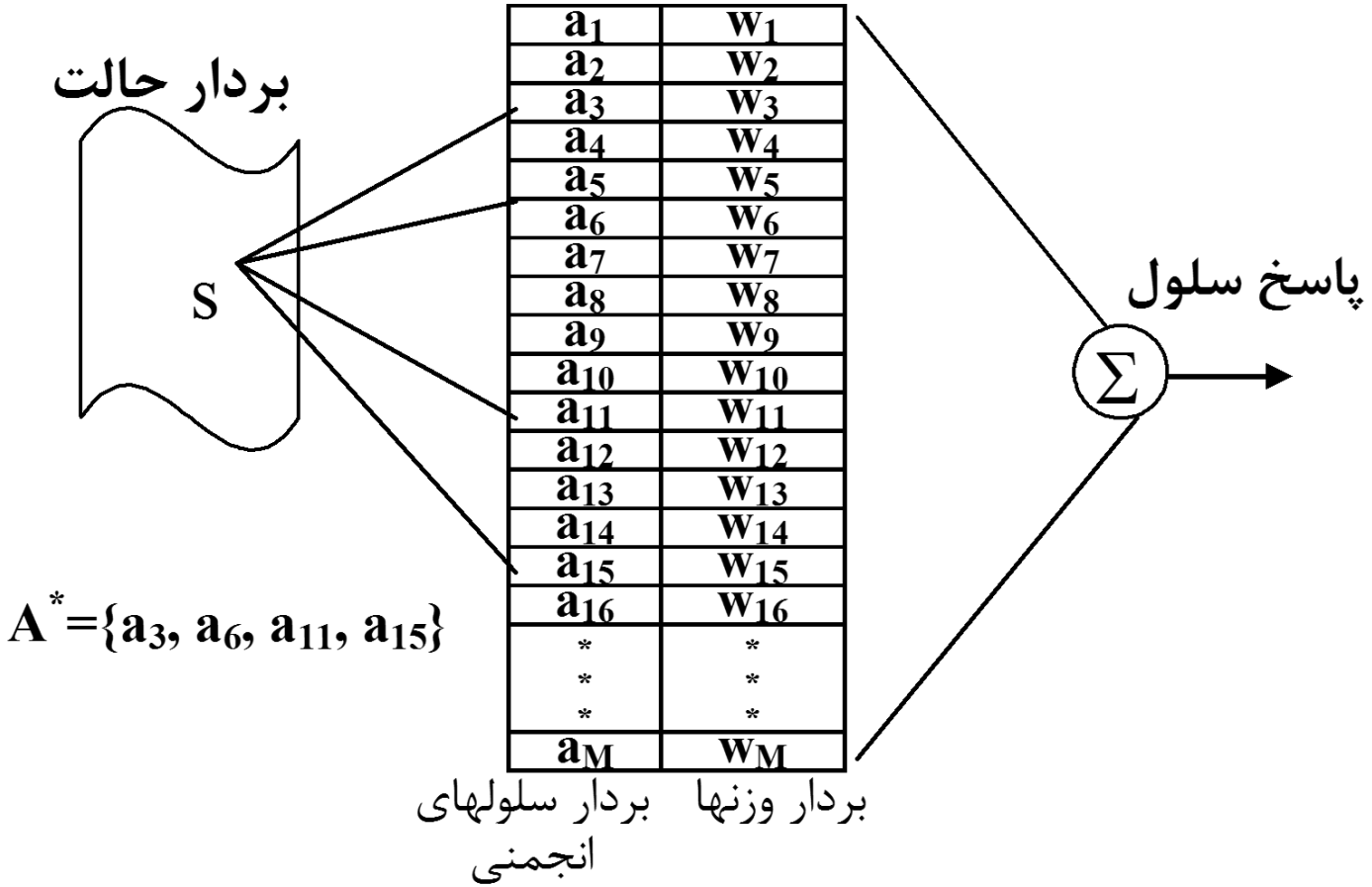
- تنظیم وزنها

$$W_{new} = W_{old} + \frac{\alpha}{N_e} a(s)(\hat{y}(s) - a^T(s) W_{old})$$

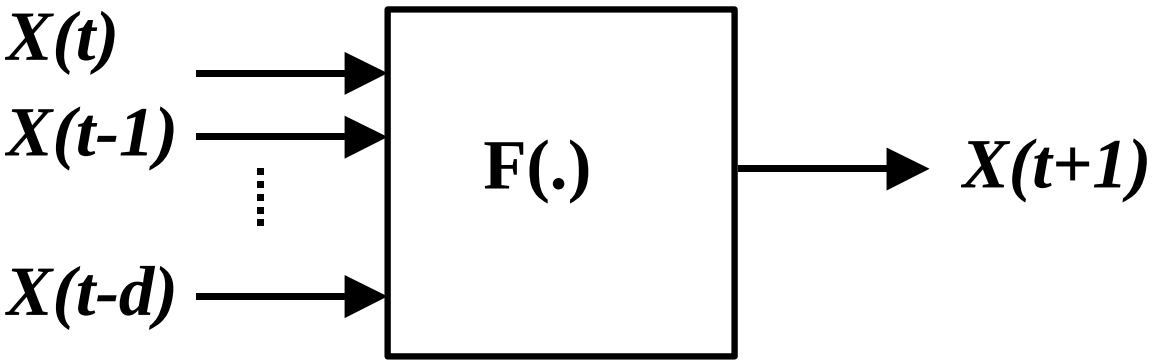
ویژگیهای CMAC:

- تنها تنظیم وزنهای فوق مکعبهای فعال شده


WEIGHTS
SELECTION OF WEIGHTS
INPUT SPACE
SUMMATION OF WEIGHTS
OUTPUT
ADJUST WEIGHTS
DESIRED OUTPUT


بردار حالت
S
a_1 w_1
a_2 w_2
a_3 w_3
a_4 w_4
a_5 w_5
a_6 w_6
a_7 w_7
a_8 w_8
a_9 w_9
a_{10} w_{10}
a_{11} w_{11}
a_{12} w_{12}
a_{13} w_{13}
a_{14} w_{14}
a_{15} w_{15}
a_{16} w_{16}
a_M w_M
$A^*=\{a_3, a_6, a_{11}, a_{15}\}$
بردار سلولهای انجمنی
بردار وزنها
پاسخ سلول
Σ

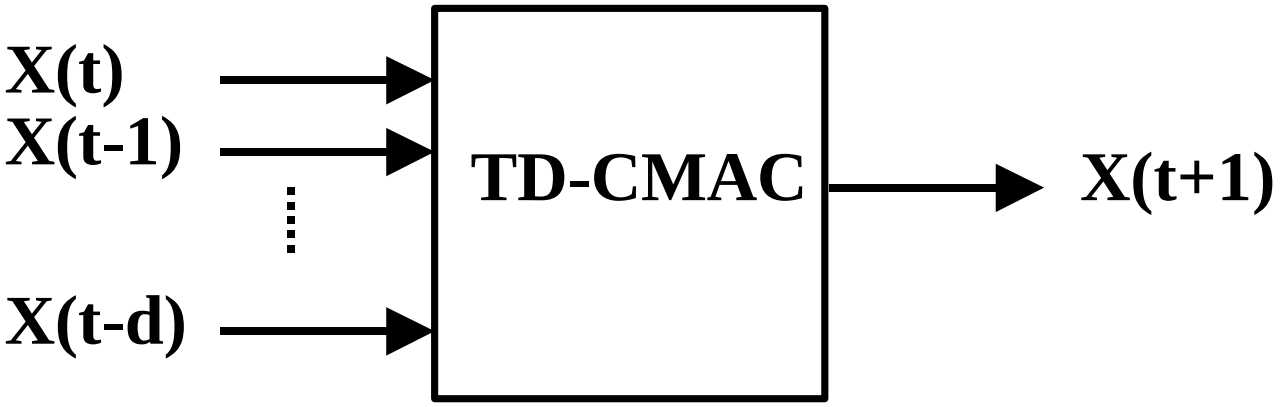
Time Series Modeling



$$X(t+1) = F(\, X(t) + X(t-1) + X(t-2) + X(t-d)\,) + e(t)$$

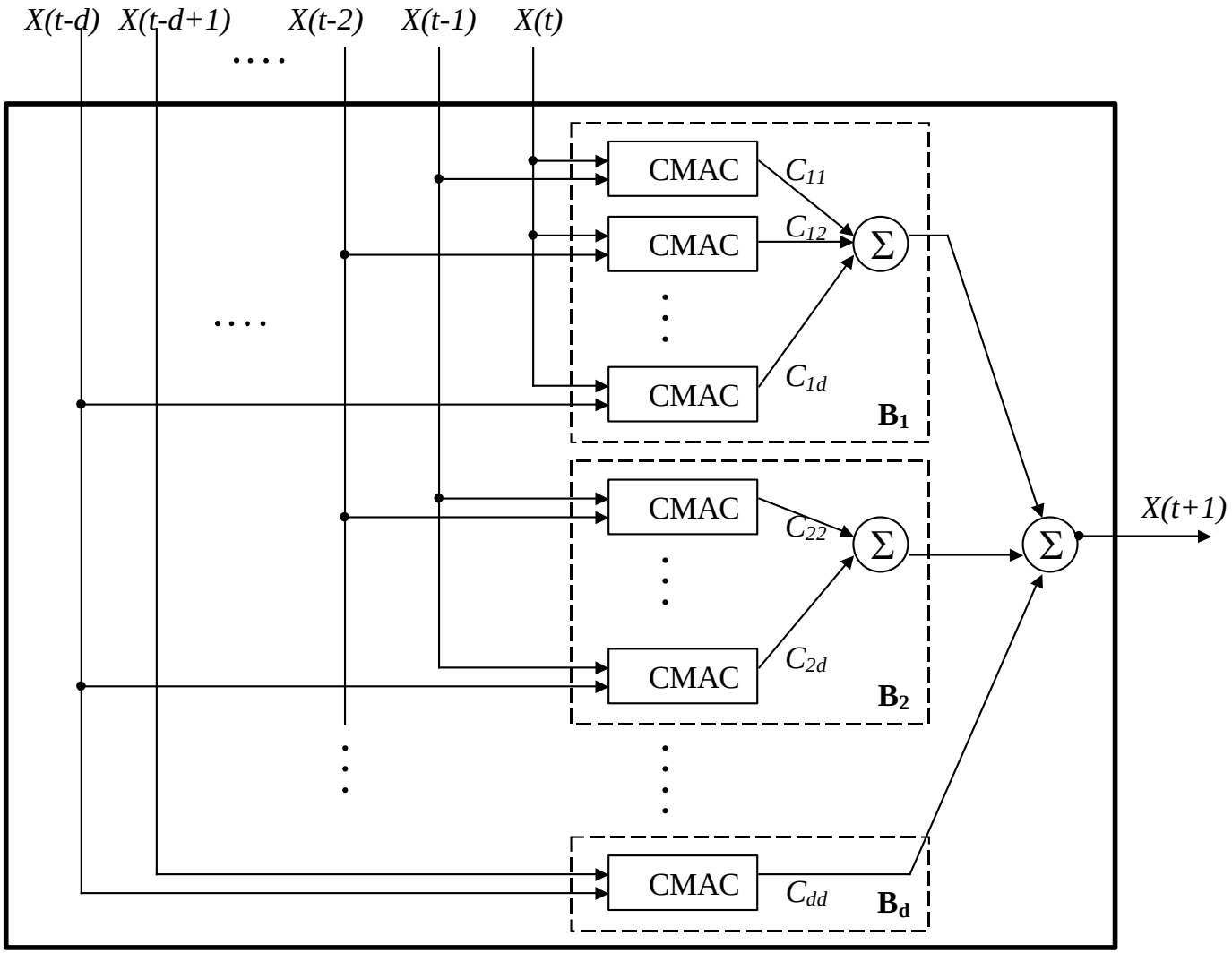
Converting prediction of time series to function approximation

TD-CMAC Model



block diagram of TD-CMAC model

TD-CMAC Model



The structure details of TD-CMAC model

TD-CMAC Model

1- Output Mapping

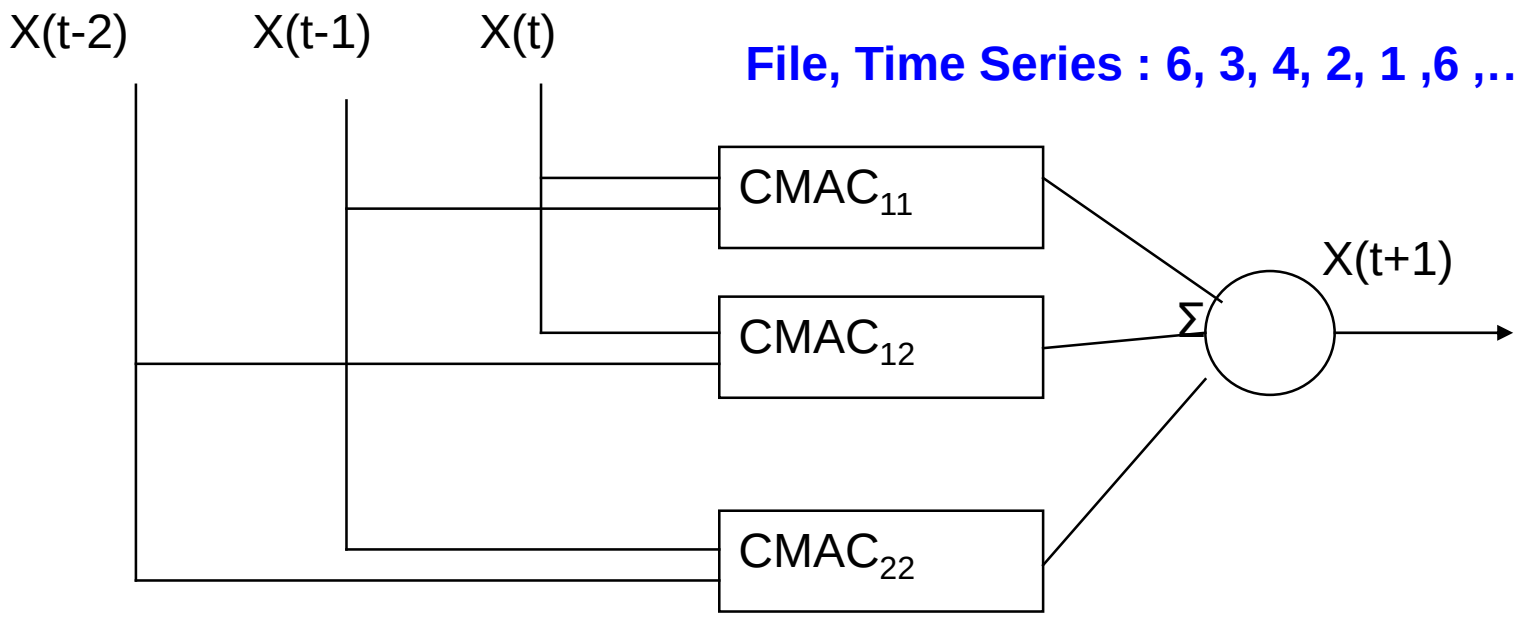
$$y(s) = a(s)W = \sum_{i=1}^{d}\sum_{j=i}^{d}\sum_{k=1}^{M} a_{ijk}(s)w_{ijk}$$

2- Learning Algorithm

$$W_{new} = W_{old} + \frac{\alpha}{N_e} c(s)a(s)(\hat{y}(s) - y(s))$$

$$c_{ij}(s) = \frac{1}{ij}$$

TD-CMAC (Example)

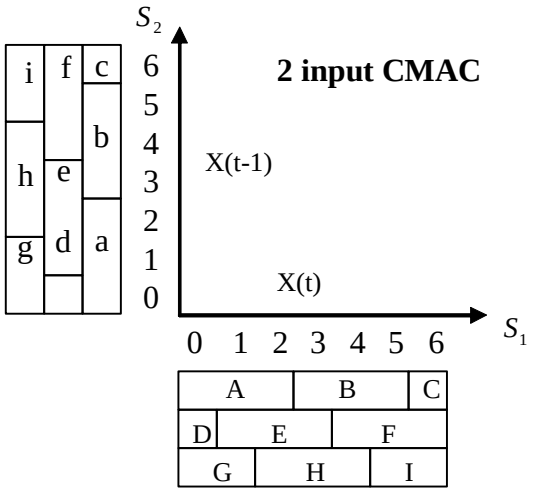


Weight table:
$W[i][j][N_b][N_b][N_e]$

For (i=1,2) For (j=i,2)

Time Series: S[d+2]

W[2][2][3][3][3] , S[4]



TD-CMAC Model

d=2 (Number of delay)

N_b=3 (Number of block)

N_e=3 (Number of layer)

α= (Learning rate)

Error = 0.001

TD-CMAC (Example)

File, Time Series : 6, 3, 4, 2, 1 ,6 ,…

- The first time series data for training of model
- The next data for evaluation of model (prediction)

1) d+1 input for X(t) to X(t-1)

- X(t-2) -> S[1]=6
- X(t-1) -> S[2]=3
- X(t) -> S[3]=4

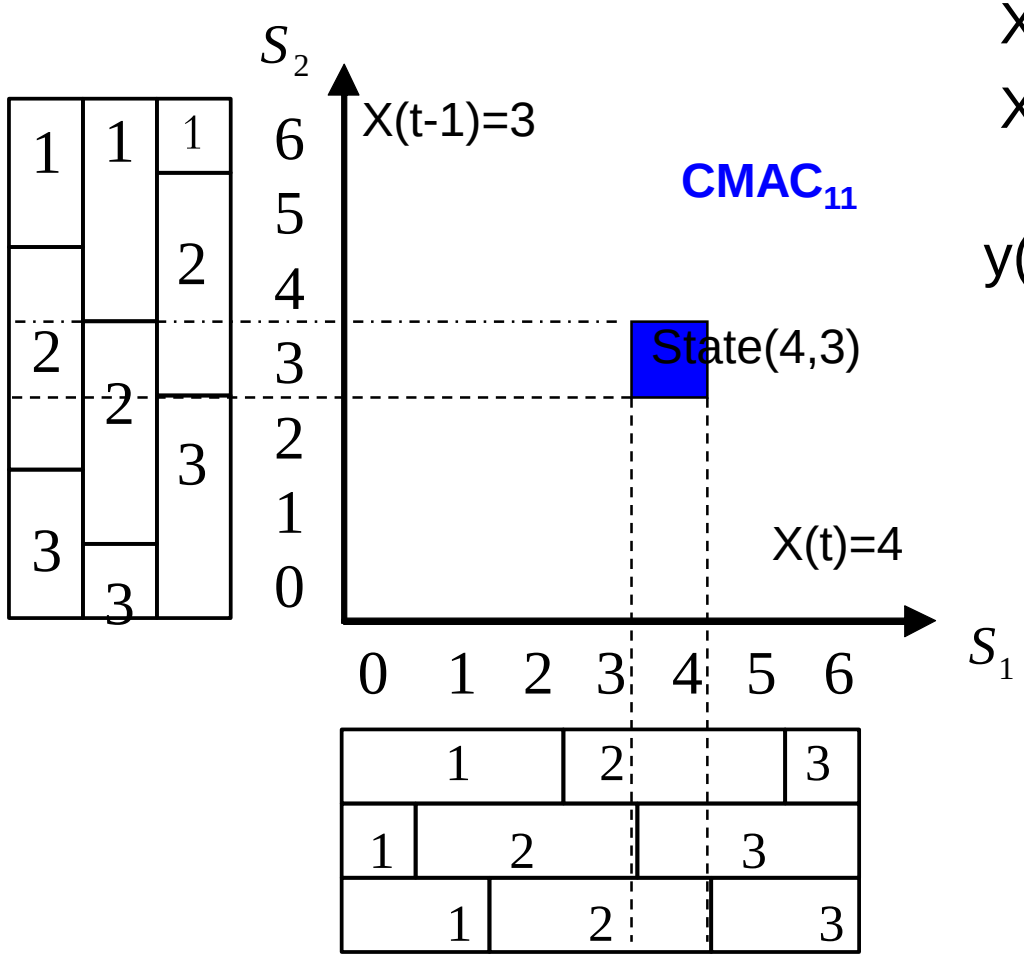
2) Next data for X(t+1)

- X(t+1) -> S[4]=2

3) Weight Initialization to zero

A) Output Calculating

$$y(s) = a(s)W = \sum_{i=1}^{d}\sum_{j=i}^{d}\sum_{k=1}^{M} a_{ijk}(s)w_{ijk}$$

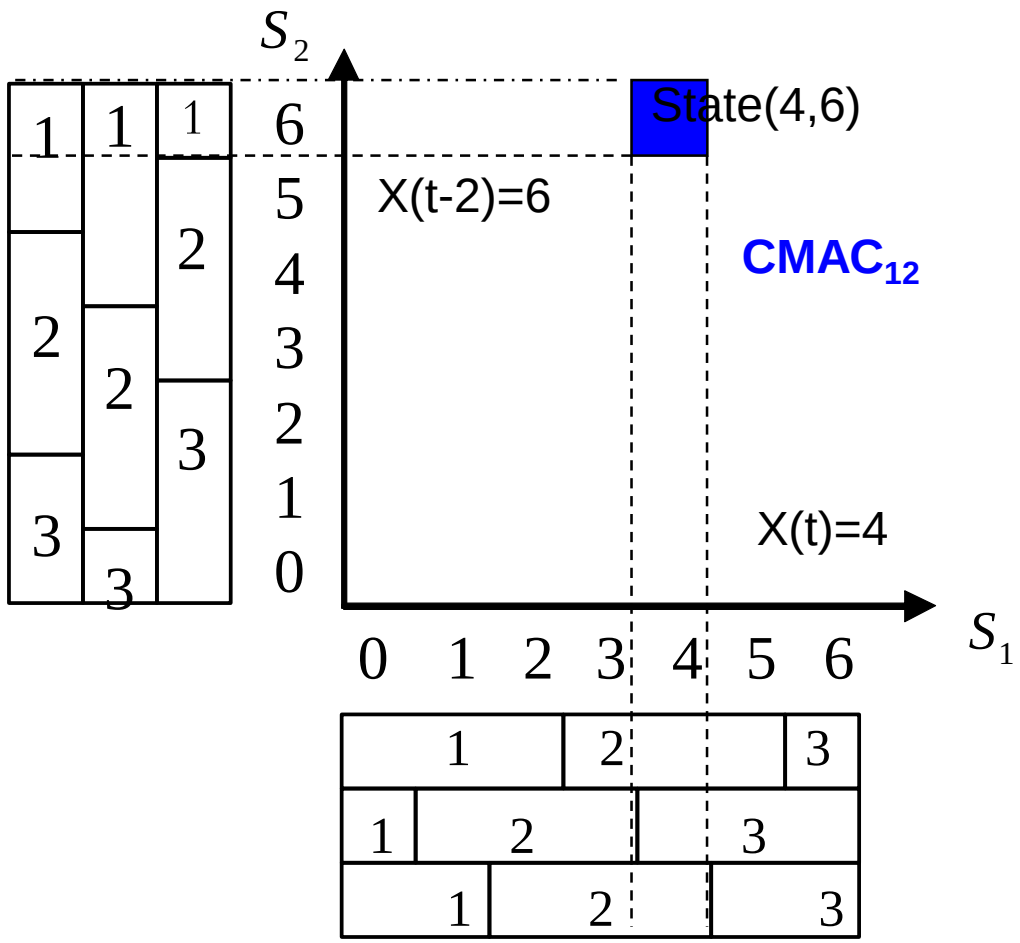


X(t-2) -> S[1]=6
X(t-1) -> S[2]=3
X(t) -> S[3]=4
X(t+1)-> S[4]=2

y(s) = w[1][1][2][2][1]
+ w[1][1][3][2][2]
+ w[1][1][2][2][3]
+…

A) Output Calculating

$$y(s) = a(s)W = \sum_{i=1}^{d}\sum_{j=i}^{d}\sum_{k=1}^{M} a_{ijk}(s)w_{ijk}$$



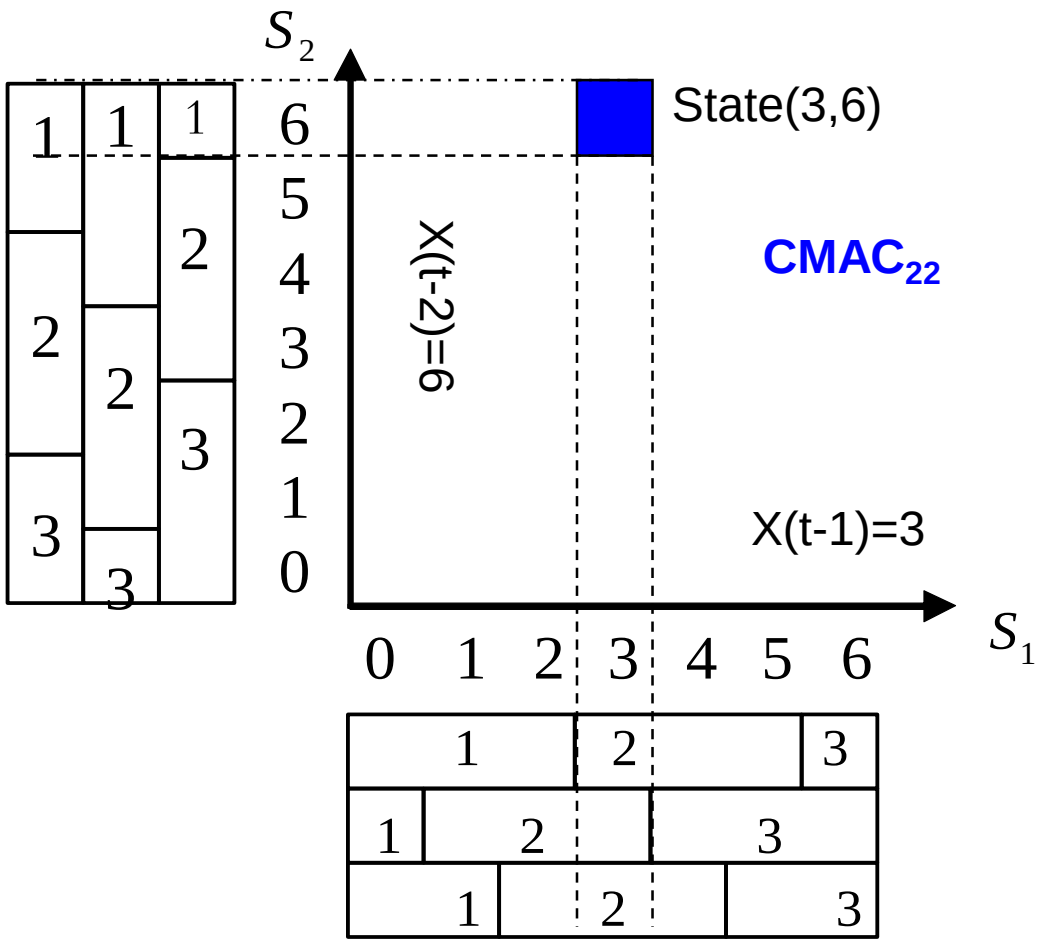
X(t-2) -> S[1]=6
X(t-1) -> S[2]=3
X(t) -> S[3]=4
X(t+1)-> S[4]=2

y(s) = +…
+ w[1][2][2][1][1]
+ w[1][2][3][1][2]
+ w[1][2][2][1][3]
+…

A) Output Calculating

$$y(s) = a(s)W = \sum_{i=1}^{d}\sum_{j=i}^{d}\sum_{k=1}^{M} a_{ijk}(s)w_{ijk}$$

X(t-2) -> S[1]=6
X(t-1) -> S[2]=3
X(t) -> S[3]=4
X(t+1)-> S[4]=2



CMAC$_{22}$

y(s) = +…
+ w[2][2][2][1][1]
+ w[2][2][2][1][2]
+ w[2][2][2][1][3]
= 0

B) Weight Adjusting

$$W_{new} = W_{old} + \frac{\alpha}{N_e} c(s)a(s)(\hat{y}(s) - y(s))$$

$$c_{ij}(s) = \frac{1}{ij}$$

CMAC$_{ij}$

y(s) = 0

$\hat{y}(s)$ = X(t+1)=S[4]=2

CMAC$_{11}$

w[1][1][2][2][1]$_{new}$=0+(0.1/3)*(1/(1*1))*(2-0) = 0.066

w[1][1][3][2][2]$_{new}$=0+(0.1/3)*(1/(1*1))*(2-0) = 0.066

w[1][1][2][2][3]$_{new}$=0+(0.1/3)*(1/(1*1))*(2-0) = 0.066

CMAC$_{12}$

w[1][2][2][1][1]$_{new}$=0+(0.1/3)*(1/(1*2))*(2-0) = 0.033

w[1][2][3][1][2]$_{new}$=0+(0.1/3)*(1/(1*2))*(2-0) = 0.033

w[1][2][2][1][3]$_{new}$=0+(0.1/3)*(1/(1*2))*(2-0) = 0.033

CMAC$_{22}$

w[2][2][2][1][1]$_{new}$=0+(0.1/3)*(1/(2*2))*(2-0) = 0.016

w[2][2][2][1][2]$_{new}$=0+(0.1/3)*(1/(2*2))*(2-0) = 0.016

w[2][2][2][1][3]$_{new}$=0+(0.1/3)*(1/(2*2))*(2-0) = 0.016

C) Repeat A, B with new weights

A) Output Calculating

$$y(s) = a(s)W = \sum_{i=1}^{d}\sum_{j=i}^{d}\sum_{k=1}^{M} a_{ijk}(s)w_{ijk}$$

y(s) =
0.066+0.066+0.066+0.033+0.033+0.033+0.016+0.016+0.016 = 0.35

y(s) = 0.35

$\hat{y}(s)$ = X(t+1)=S[4]=2

error = 0.001

$$ERROR = \left|\hat{y}(s) - y(s)\right| = \left|2 - 0.35\right| = 1.65 > error$$

End) Until (*ERROR*< error)

C) Repeat A, B with new weights

B) Weight Adjusting

$$W_{new} = W_{old} + \frac{\alpha}{N_e} c(s) a(s) (\hat{y}(s) - y(s))$$

$$c_{ij}(s) = \frac{1}{ij}$$

y(s) = 0

$\hat{y}(s)$ = X(t+1)=S[4]=2

$w[1][1][2][2][1]_{new}$=$w[1][1][3][2][2]_{new}$=$w[1][1][2][2][3]_{new}$=
0.066+(0.1/3)*(1/(1*1))*(2-0.35) = 0.1

$w[1][2][2][1][1]_{new}$=$w[1][2][3][1][2]_{new}$=$w[1][2][2][1][3]_{new}$=
0.033+(0.1/3)*(1/(1*2))*(2-0.35) = 0.060

$w[2][2][2][1][1]_{new}$=$w[2][2][2][1][2]_{new}$=$w[2][2][2][1][3]_{new}$=
0+(0.1/3)*(1/(2*2))*(2-0) = 0.030

C) Repeat A, B with new weights

A) Output Calculating

$$y(s) = a(s)W = \sum_{i=1}^{d}\sum_{j=i}^{d}\sum_{k=1}^{M} a_{ijk}(s)w_{ijk}$$

y(s) = 0.1+0.1+0.1+0.06+0.06+0.06+0.03+0.03+0.03 = 0.57

y(s) = 0.57

$\hat{y}(s)$ = X(t+1)=S[4]=2

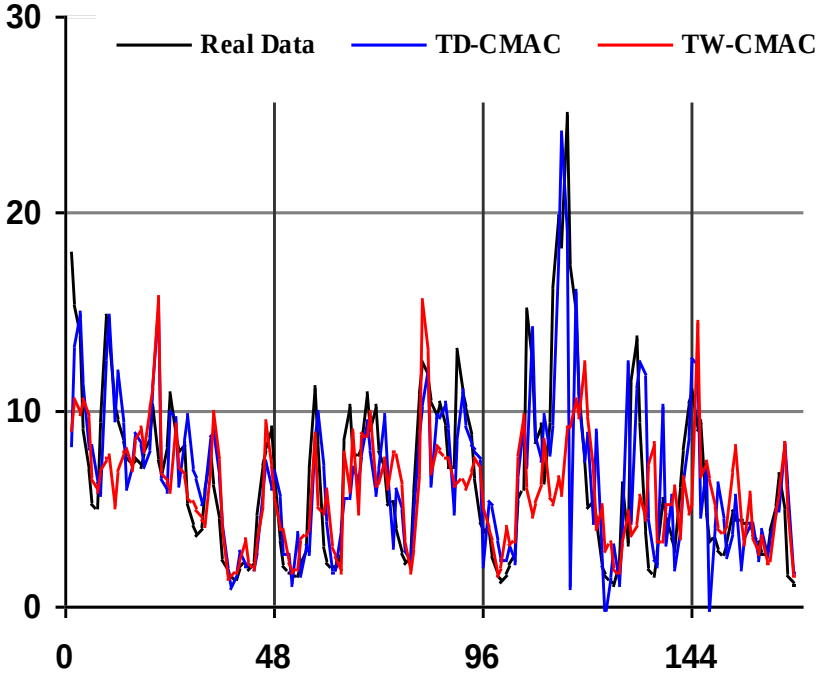
error = 0.001

$$ERROR = \left|\hat{y}(s) - y(s)\right| = \left|2 - 0.57\right| = 1.43 > error$$

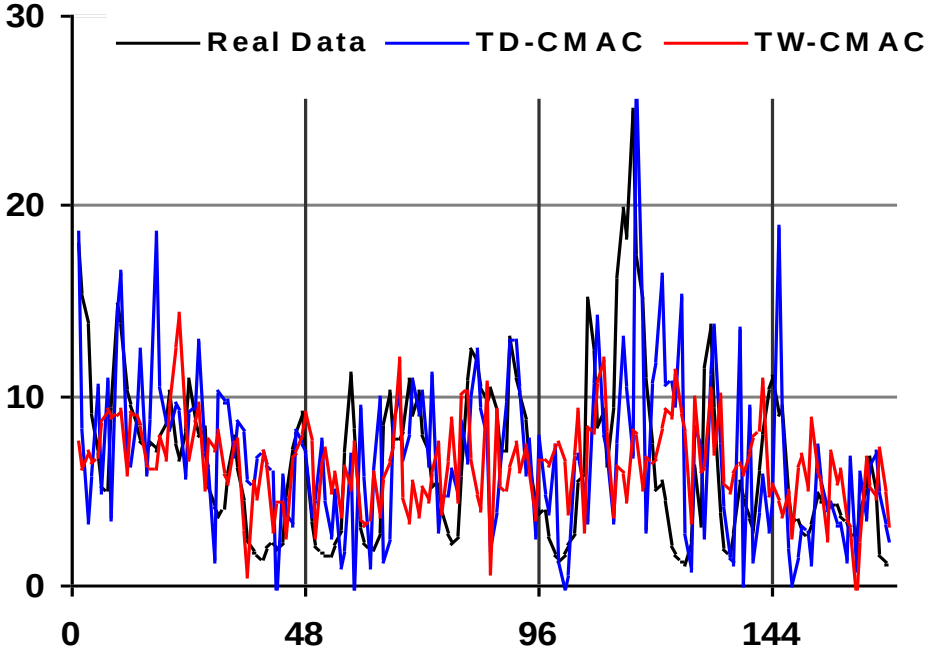
End) Until (*ERROR*< error)

CO (ppm)

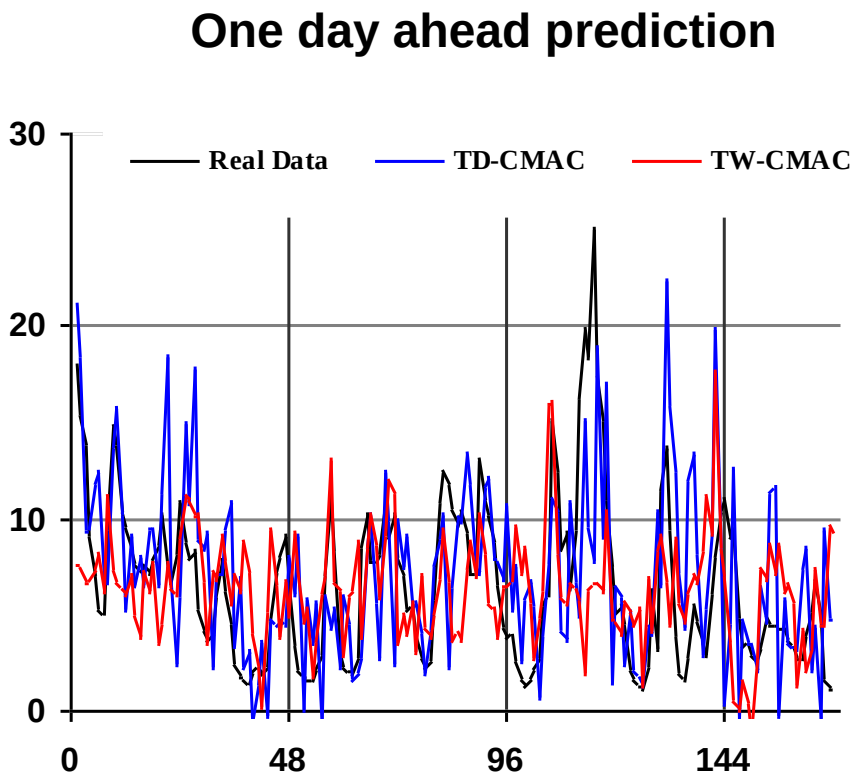
One hour ahead prediction



Six hour ahead prediction

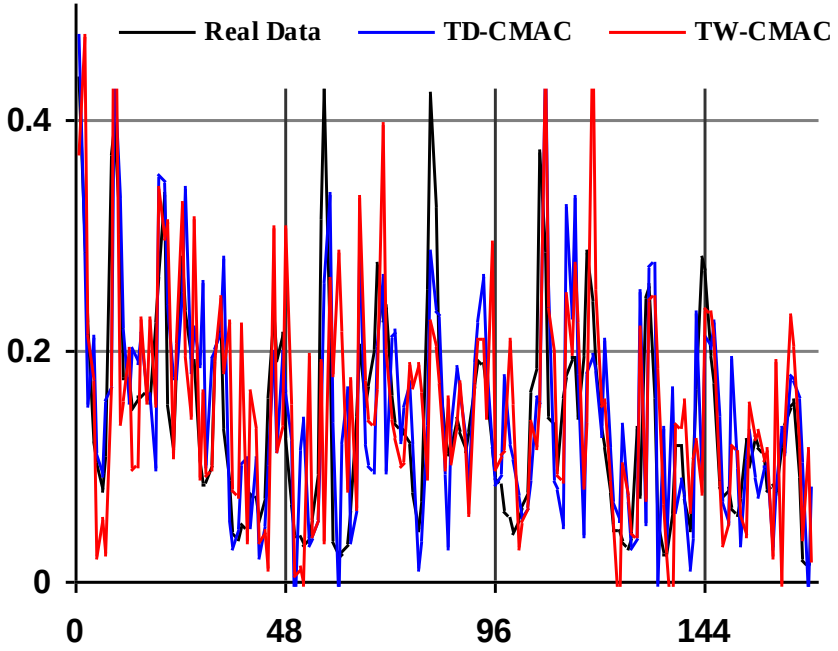


CO (ppm)


One day ahead prediction
Real Data
TD-CMAC
TW-CMAC
30
20
10
0
0
48
96
144

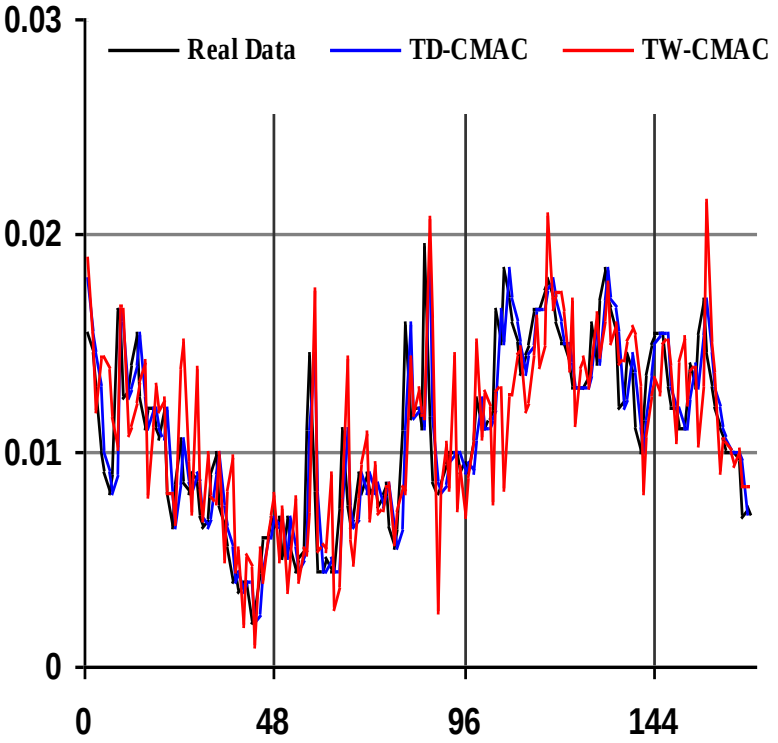
SO_2 (ppm)

One hour ahead prediction



NO_x (ppm)

One hour ahead prediction



فصل نهم :

شبکه عصبی بیزین Bayesian Neural Network

۱ . معرفی شبکه عصبی بیزین تک لایه

۲ . توسعه شبکه عصبی بیزین تک لایه به چندلایه

۲-۱ روش قسمت بندی (Partitioning)

۲-۲ روش روی هم افتادگی (Overlapping)

۳ . ویژگی های پیوسته و شبکه عصبی بیزین

معرفی شبکه عصبی بیزین تک لایه:

هدف نهایی که در این شبکه به دنبال آن هستیم، محاسبه احتمال تمامی کلاس ها به ازای داده های ورودی به شبکه، و انتخاب کلاس با بیشترین مقدار احتمال برای هر داده، به عنوان کلاسی که داده متعلق به آن است، می باشد.

معرفی شبکه عصبی بیزین تک لایه:

تئوری بیز برای محاسبه P(y|x):

$$p(y|x) = p(y)\frac{p(x|y)}{p(x)}$$

معرفی شبکه عصبی بیزین تک لایه:

فرض کنید مقادیر N ویژگی مستقل مربوط به داده x، $x=\{x_1 , x_2 , \ldots, x_N\}$، در دست باشد. در اینصورت احتمال x می تواند به صورت زیر نوشته شود :

$$p(x) = p(x_1).p(x_2) \ldots\ldots p(x_N)$$

همچنین احتمال شرطی p(x|y) نیز بصورت زیر نوشته می شود:

$$p(x|y) = p(x_1|y).p(x_2|y) \ldots\ldots p(x_N|y)$$

معرفی شبکه عصبی بیزین تک لایه:

احتمال هر کلاس به ازای مشاهده داده ورودی، به صورت زیر محاسبه خواهد شد :

$$p(y|x) = p(y)\frac{p(x|y)}{p(x)} = p(y)\prod_{i=1}^{N}\frac{p(x_i|y)}{p(x_i)}$$

معرفی شبکه عصبی بیزین تک لایه:

$$p(y|x) = p(y)\frac{p(x|y)}{p(x)} = p(y)\prod_{i=1}^{N}\frac{p(x_i|y)}{p(x_i)}$$

معادله اخیر پایه **کلاسیفیر ساده بیز** می باشد.
طراحی بیز ساده مربوط به اوقاتی می شود که **فرض مستقل بودن ورودی ها** را داریم.

معرفی شبکه عصبی بیزین تک لایه:

رابطه زیر همواره برقرار است :

$$\frac{p(x_i|y)}{p(x_i)} = \frac{p(x_i,y)}{p(y)p(x_i)} = \frac{p(y|x_i)}{p(y)}$$



$$p(y|x) = p(y)\frac{p(x|y)}{p(x)} = p(y)\prod_{i=1}^{N}\frac{p(x_i|y)}{p(x_i)}$$

معرفی شبکه عصبی بیزین تک لایه:

با جایگذاری و لگاریتم گرفتن از رابطه (۱) به رابطه زیر خواهیم رسید:

$$logP(y|x) = logP(y) + \sum_i \log\left(\frac{p(y, x_i)}{p(y)p(x_i)}\right)$$

مزیت فرم لگاریتمی خطی بودن آن است. بدین معنی که کلاسیفیر ساده بیز می تواند با یک تابع تمایز خطی و بنابراین با یک شبکه عصبی تک لایه پیاده سازی شود.

معرفی شبکه عصبی بیزین تک لایه:

معادله معمولی برای پخش سیگنال در یک شبکه عصبی صورت زیر است:

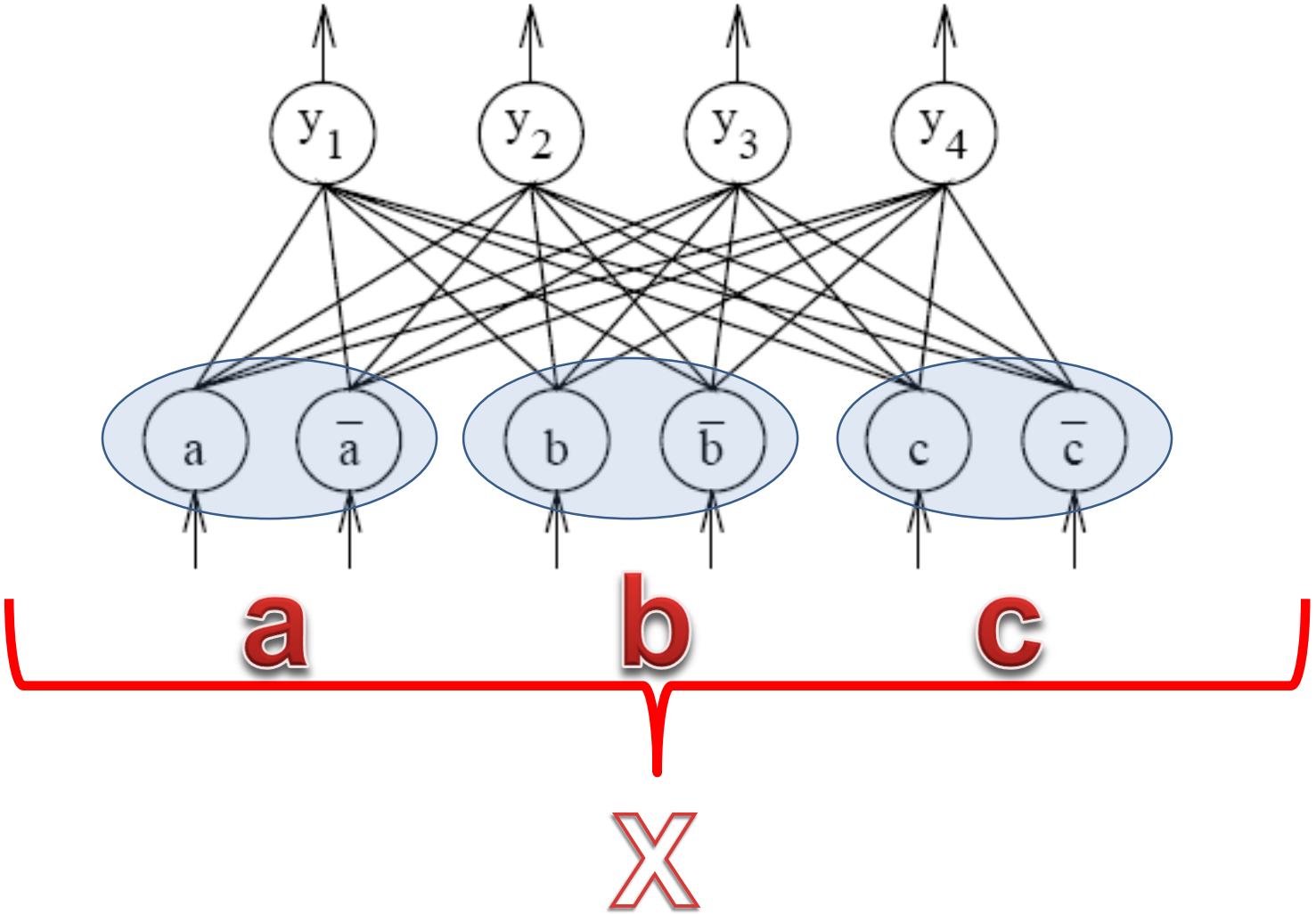
$$s_j = \beta_j + \sum_i w_{ji} o_i$$

$$o_i = f(s_j)$$

معرفی شبکه عصبی بیزین تک لایه:

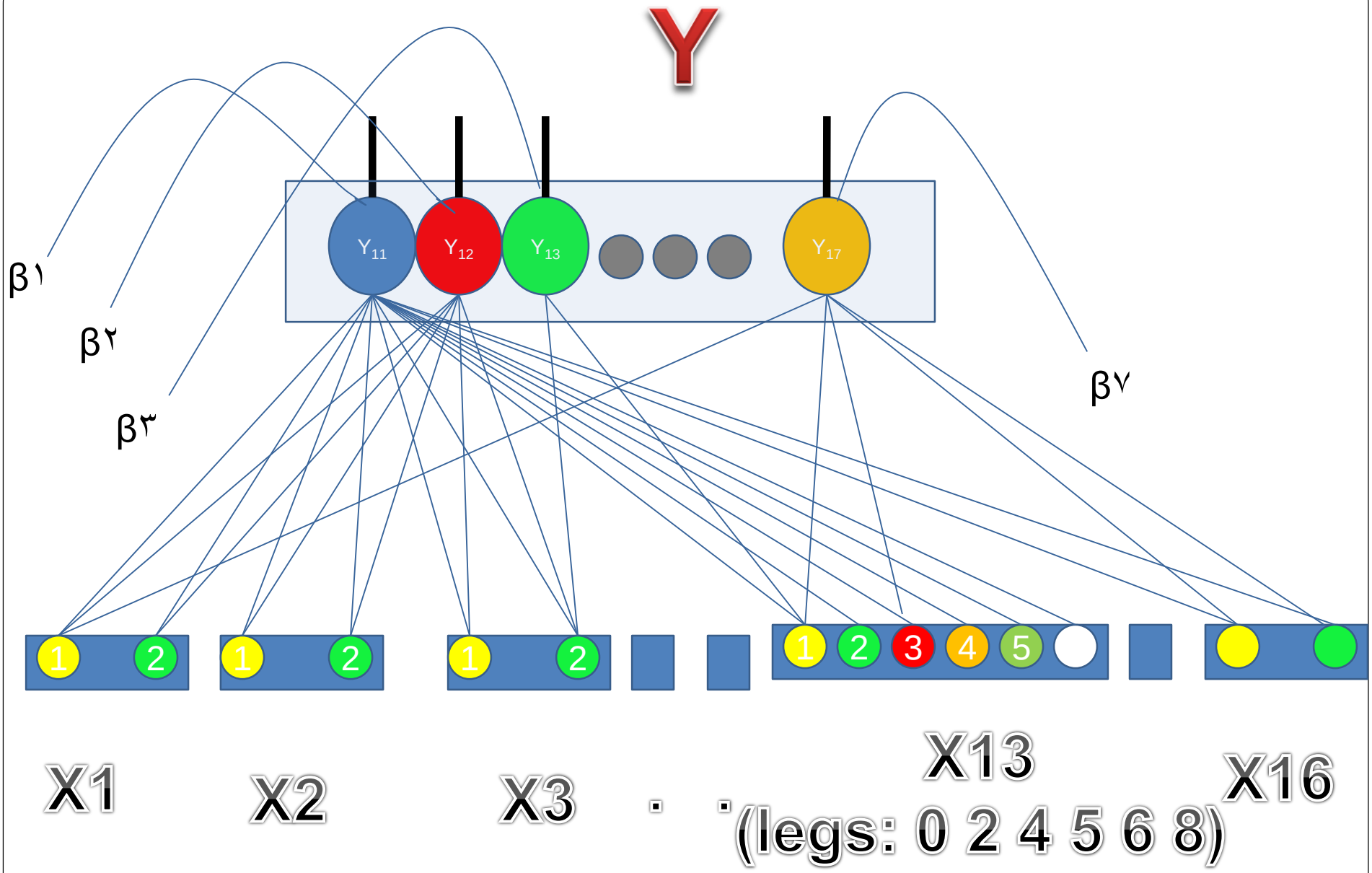
نمای شبکه عصبی بیزین تک لایه برای حالتی که در آن ویژگی های مربوط به داده ها، گسسته (دارای چند مقدار محدود) می باشند:

معرفی شبکه عصبی بیزین تک لایه:



معرفی شبکه عصبی بیزین تک لایه:

1. hair	**Boolean**	
2. feathers		Boolean
3. eggs		Boolean
4. milk	Boolean	
5. airborne		Boolean
6. aquatic		Boolean
7. predator		Boolean
8. toothed		Boolean
9. backbone		Boolean
10. breathes		Boolean
11. venomous		Boolean
12. fins	Boolean	
13. legs		**Numeric (set of values: {0,2,4,5,6,8})**
14. tail	Boolean	
15. domestic		Boolean
16. capsize		Boolean
17. type		**Numeric (integer values in range [1,7])**


Y
Y11
Y12
Y13
Y17
β۱
β۲
β۳
β۷
1
2
1
2
1
2
1
2
3
4
5
X1
X2
X3
X13
X16
(legs: 0 2 4 5 6 8)

معرفی شبکه عصبی بیزین تک لایه:

$\pi_{jj'}$ به احتمال شرطی نتیجه متغیر y_i به شرط مشاهده ویژگی داده x_i، یعنی $P(y_j|x_i)$ اشاره کند:

$$logP(y|x) = logP(y) + \sum_i \log\left(\frac{p(y,x_i)}{p(y)p(x_i)}\right)$$

$$\log(\pi_{jj'}) = log\ P(y_{jj'}) + \sum_{x_{ii'} \in A} \log\left(\frac{p(y_{jj'},x_{ii'})}{p(y_{jj'})p(x_{ii'})}\right)$$

$$= log\ P(y_{jj'}) + \sum_{i,i'} \log\left(\frac{p(y_{jj'},x_{ii'})}{p(y_{jj'})p(x_{ii'})}\right) o_{ii'}$$

معرفی شبکه عصبی بیزین تک لایه:

حال می توانیم دو فرم رابطه زیر را با هم مقایسه کنیم:

$$\log(\pi_{jj'}) = log\ P(y_{jj'}) + \sum_{x_{ii'} \epsilon A} \log\left(\frac{p(y_{jj'}, x_{ii'})}{p(y_{jj'})p(x_{ii'})}\right)$$

$$= log\ P(y_{jj'}) + \sum_{i,i'} \log\left(\frac{p(y_{jj'}, x_{ii'})}{p(y_{jj'})p(x_{ii'})}\right) o_{ii'}$$

$$s_j = \beta_j + \sum_i w_{ji}\, o_i$$

$$\beta_{j\,j'} = \log P(y_{jj'})$$

$$w_{jj',ii'} = \log\left(\frac{p(y_{jj'}, x_{ii'})}{p(y_{jj'})p(x_{ii'})}\right)$$

$$s_{jj'} = \log \pi_{jj'}$$

معرفی شبکه عصبی بیزین تک لایه:

آموزش شبکه عصبی بیزین:

$$C = \sum_{\gamma} \kappa^{(\gamma)}$$

$$c_{ii'} = \sum_{\gamma} \kappa^{(\gamma)} \xi_{ii'}^{(\gamma)}$$

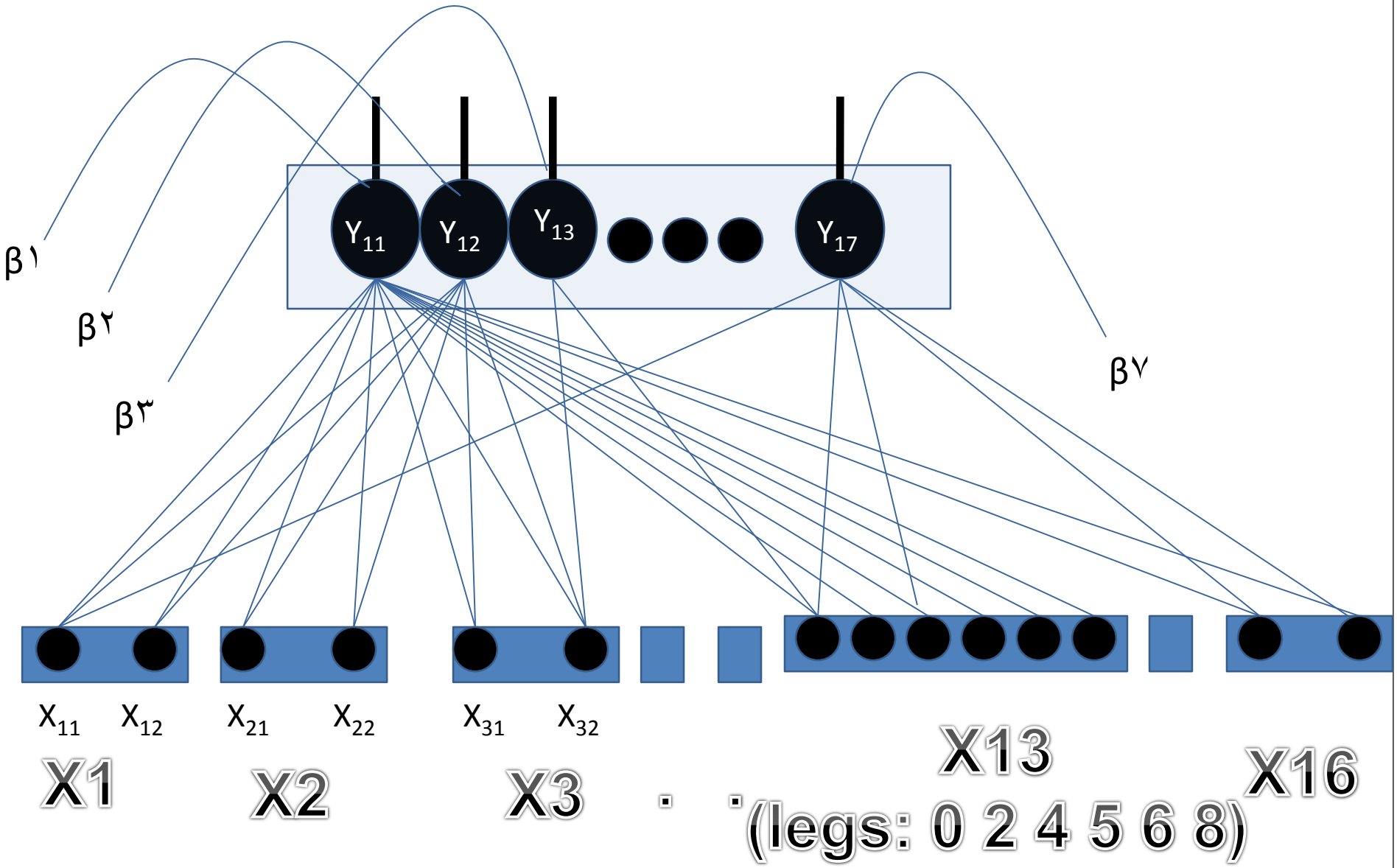
$$c_{ii',jj'} = \sum_{\gamma} \kappa^{(\gamma)} \xi_{ii'}^{(\gamma)} \xi_{jj'}^{(\gamma)}$$

معرفی شبکه عصبی بیزین تک لایه:

مثال:

فرض کنید داده ورودی به شبکه به صورت زیر باشد:

Antelope(بز کوهی):
1,0,0,1,0,0,0,1,1,1,0,0,4,1,0,1,1


β۱
β۲
β۳
β۷
Y11
Y12
Y13
Y17
X11
X12
X21
X22
X31
X32
X1
X2
X3
X13
X16
(legs: 0 2 4 5 6 8)

$Y_1 = 1 \Rightarrow$

$Y_{1\,1}$

$X_1 = 1 \Rightarrow X_{1\,2}$

$X_2 = 0 \Rightarrow X_{2\,1}$

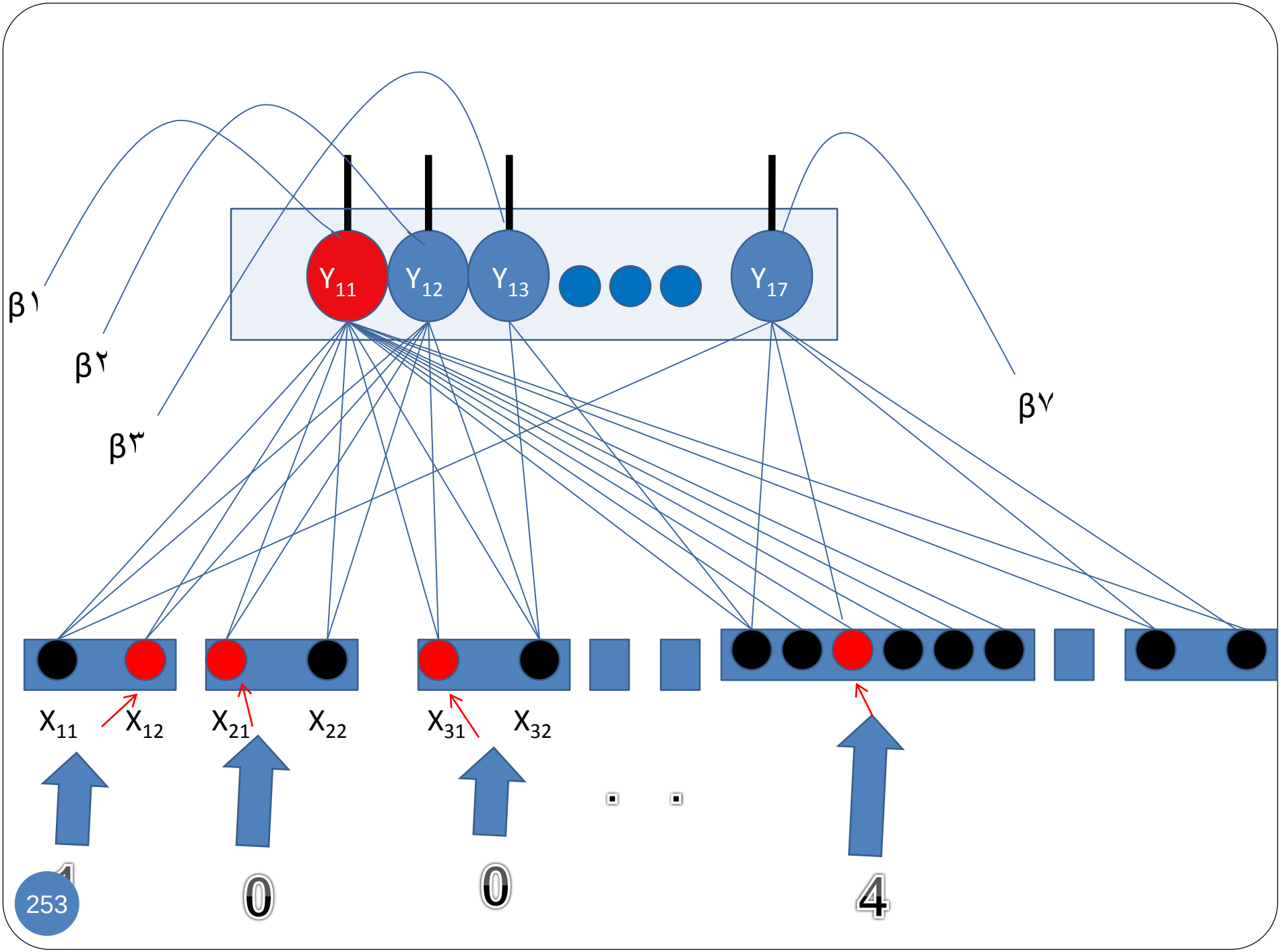
$X_3 = 0 \Rightarrow X_{3\,1}$

....

.....

$X_{13} = 4 \Rightarrow X_{13\,3}$

....


β۱
β۲
β۳
β۷
Y11
Y12
Y13
Y17
X11
X12
X21
X22
X31
X32
0
0
4
253

شمارنده های ورودی (C_I)

$$C_{12} = C_{12} + 1$$

$$C_{21} = C_{21} + 1$$

.....

$$C_{133} = C_{133} + 1$$

شمارنده های توام($C_{ll',JJ'}$)

$$C_{1\,2,1\,1} = C_{1\,2,1\,1} + 1$$

$$C_{1\,2,1\,1} = C_{1\,2,1\,1} + 1$$

$$C_{13\,3,1\,1} = C_{13\,3,1\,1} + 1$$

شمارنده های خروجی(C_J)

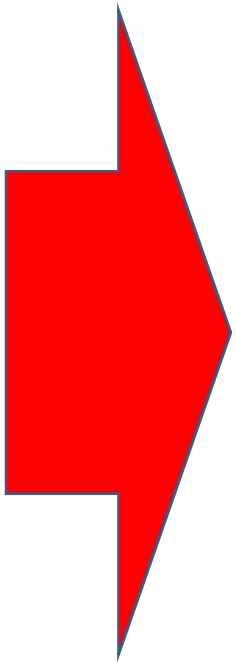
$$C_{11} = C_{11} + 1$$

معرفی شبکه عصبی بیزین تک لایه:

تخمین کلاسیک:

$$\hat{p}_i = \frac{c_i}{C}$$

$$\hat{p}_{ij} = \frac{c_{ij}}{C}$$



$$w_{ij} = \begin{cases} 0 & c_i = 0 \vee c_j = 0 \\ \log 1/C & c_{ij} = 0 \\ \log \frac{c_{ij}C}{c_i c_j} & \text{otherwise} \end{cases}$$

$$\beta_i = \begin{cases} \log 1/C^2 & c_i = 0 \\ \log c_i/C & \text{otherwise} \end{cases}$$

معرفی شبکه عصبی بیزین تک لایه:

تخمین بیزین:

$$\hat{p}_i = \frac{c_i + \alpha/n_i}{C + \alpha}$$

$$\hat{p}_{ij} = \frac{c_{ij} + \alpha/(n_i m_j)}{C + \alpha}$$

$$w_{ij} = \log \frac{(c_{ij} + \alpha/(n_i m_j))(C + \alpha)}{(c_i + \alpha/n_i)(c_j + \alpha/m_j)}$$

$$\beta_i = \log \frac{c_i + \alpha/n_i}{C + \alpha}$$

توسعه شبکه عصبی بیزین
تک لایه به چندلایه

توسعه شبکه عصبی بیزین تک لایه به چندلایه:

اگر ویژگی های ورودی به یکدیگر وابسته باشند، شبکه عصبی تک لایه جواب خوبی بدست نمی دهد.

برای جبران وابستگی ها در میان لایه ورودی و لایه خروجی از یک لایه پنهان استفاده می کنیم که این لایه از ترکیب ستون های ورودی (آنهایی که به هم وابسته هستند) حاصل می شود.

توسعه شبکه عصبی بیزین تک لایه به چندلایه:

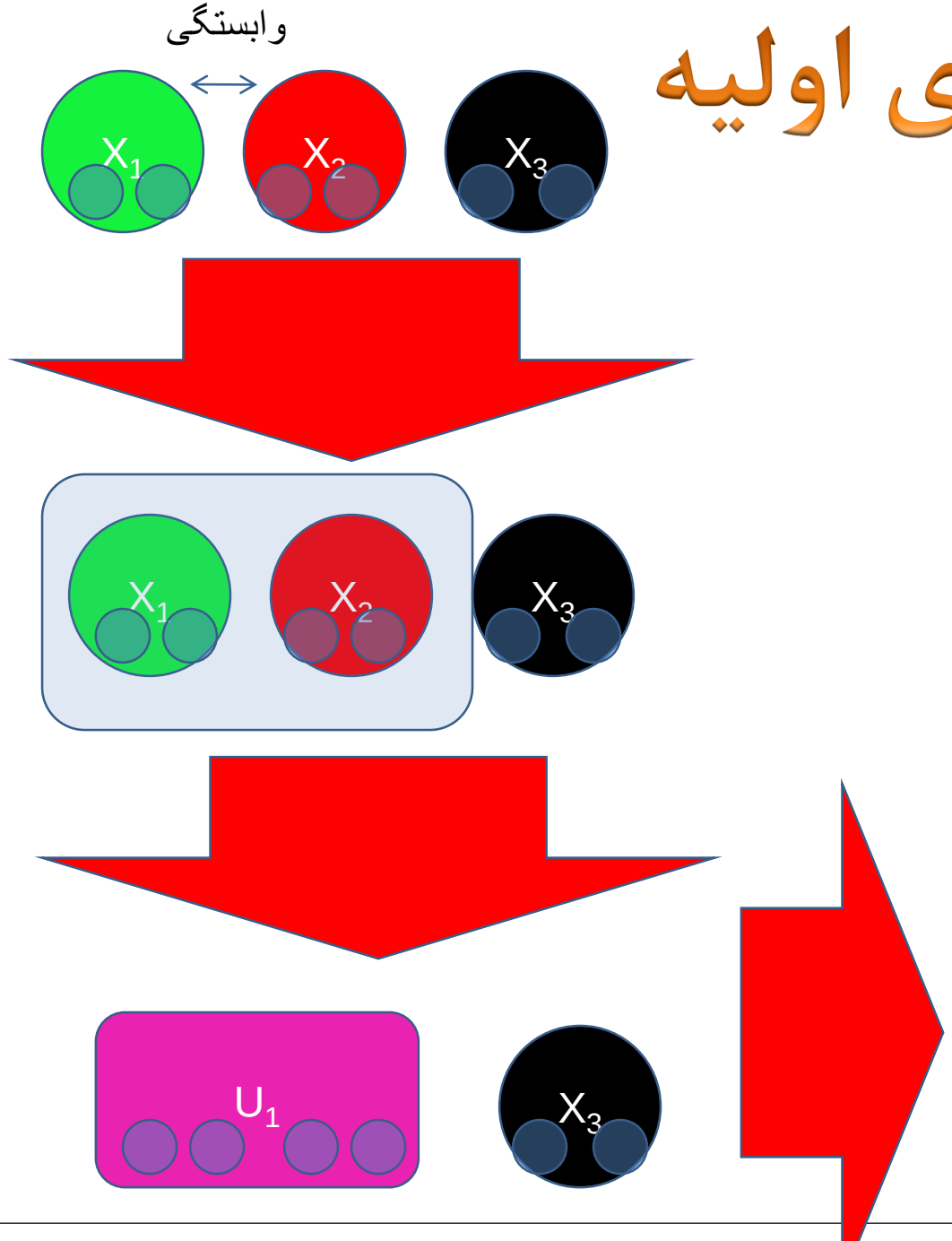
بطور کلی برای تشکیل شبکه عصبی بیزین چند لایه از دو معماری می توان سود جست :

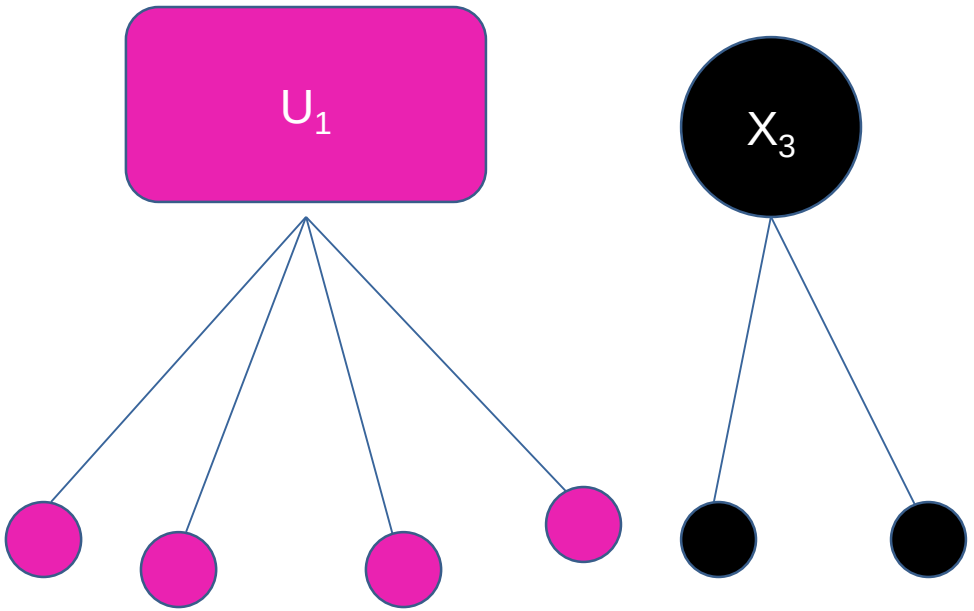
الف) Partitioning

ب) Overlapping

شبکه عصبی چند لایه
(Partitioning)

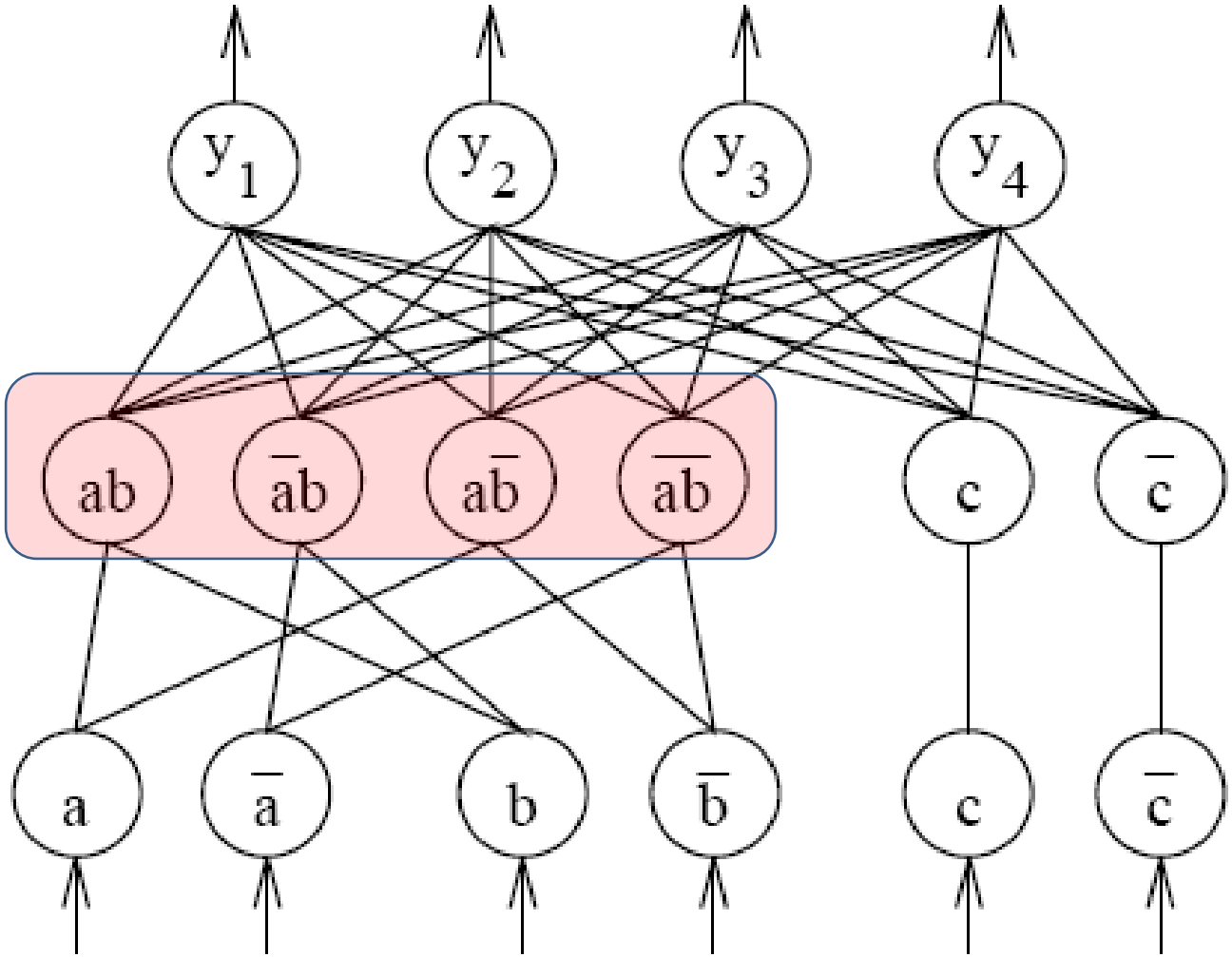
ویژگی های اولیه

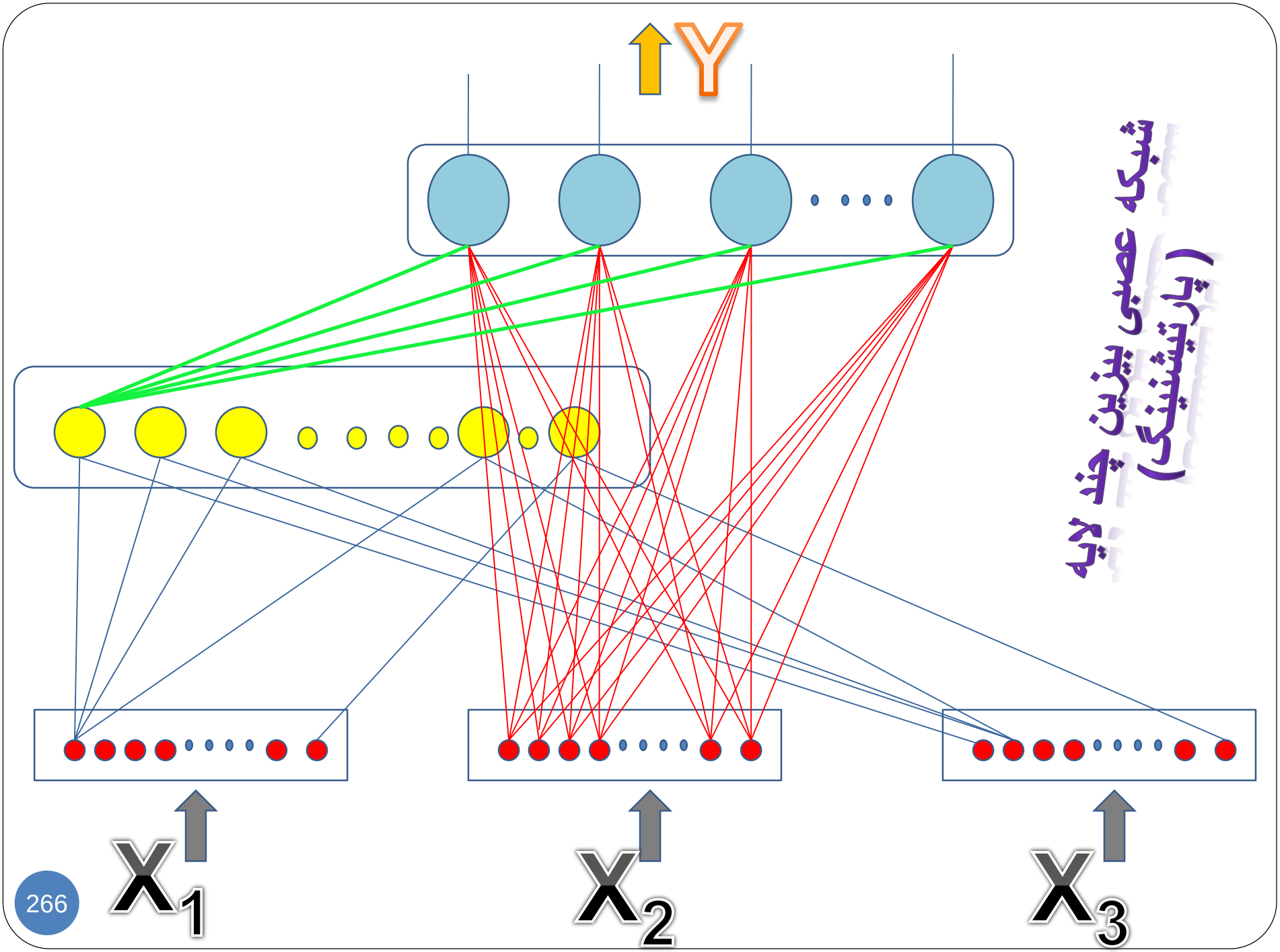




لایه پنهان

مثالی از ترکیب ویژگی های ورودی در حالت قسمت بندی




Y
شبکه عصبی پیش خور چند لایه
(پارتیشنینگ)
X1
X2
X3

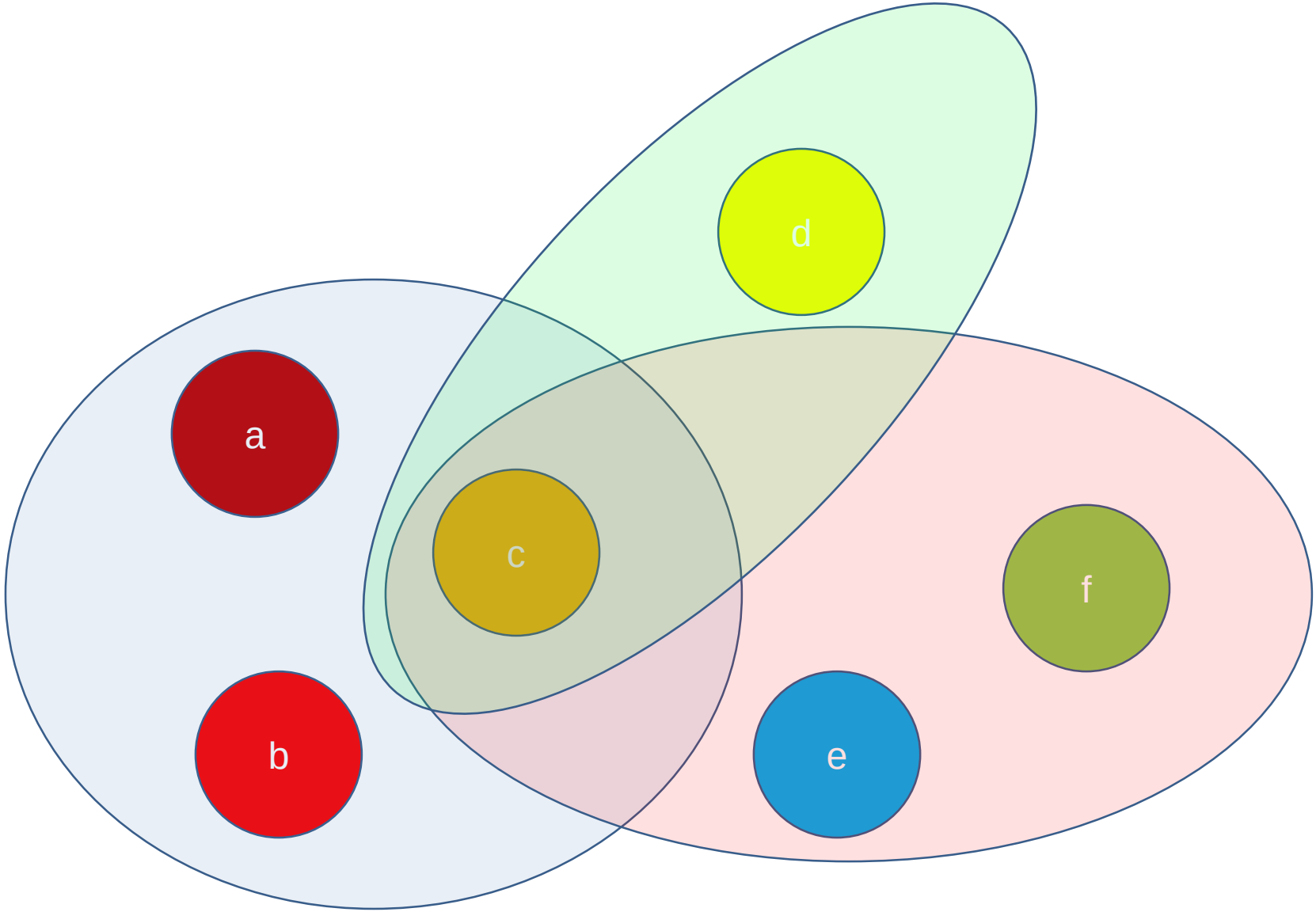
توسعه شبکه عصبی بیزین تک لایه به چندلایه:

می توان توزیع را بروی تمام حوزه به عنوان یک ضرب که شامل متغیرهای u_k می شود بیان کنیم :

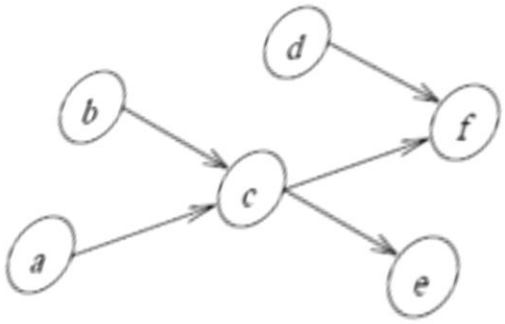
$$p(y|x) = P(y)\frac{p(x|y)}{P(x)} = P(y)\prod_{i=1}^{M}\frac{P(u_i|y)}{P(u_i)}$$

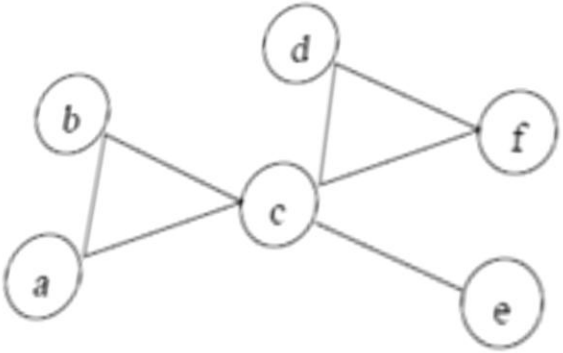
شبکه عصبی چند لایه (Overlapping)

نمونه ای از وابستگی بین ویژگی ها



گراف وابستگی





نوشتن وابستگی ها:

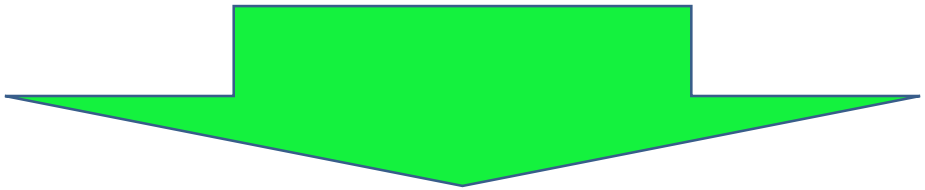
$$P(x) = P(a)P(b)P(c|ab)P(d)P(e|c)P(f|cd)$$

توجه به وابستگی های مستقیم :

$$P(x) = P(a)P(b|a)P(c|ab)P(d|abc)P(e|abcd)P(f|abcde)$$

$$P(x) = P(a)P(b)P(c|ab)P(d)P(e|c)P(f|cd)$$

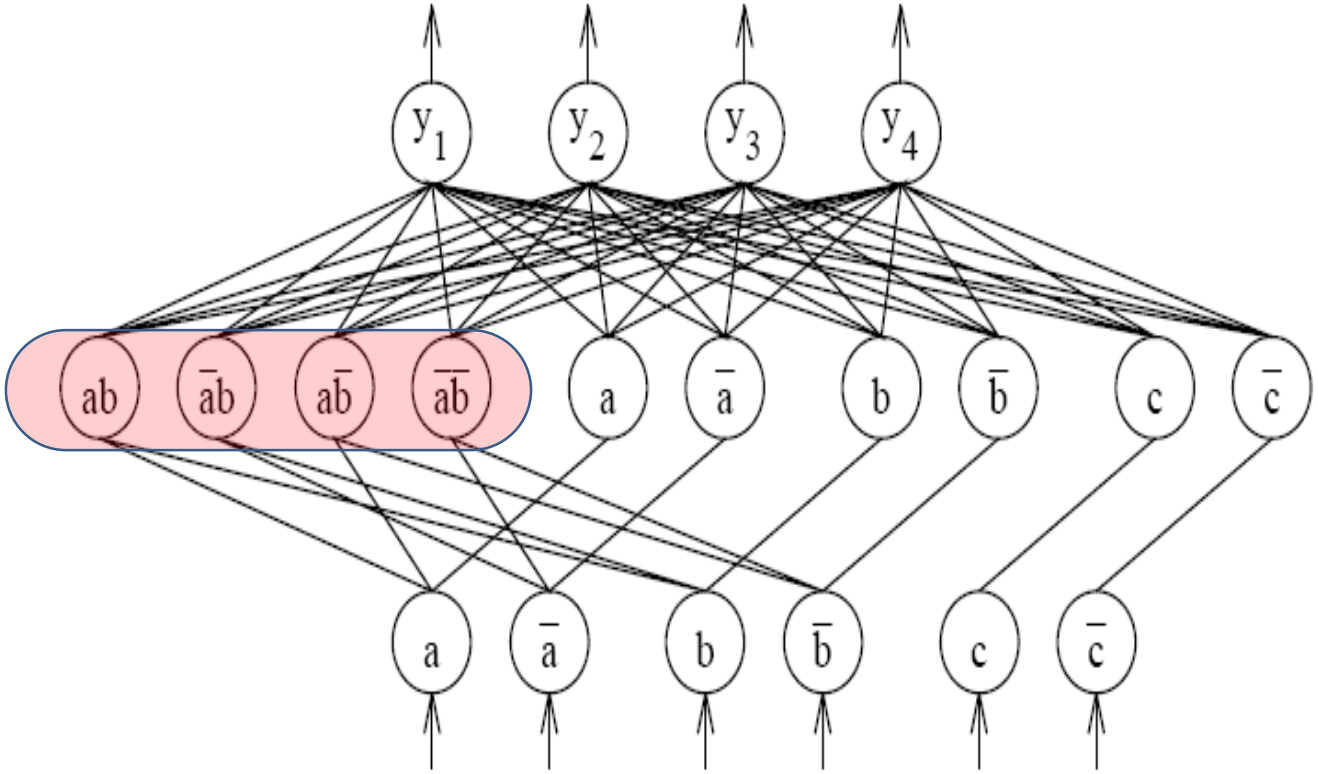
$$P(x) = P(a)P(b)\frac{P(abc)}{P(ab)}P(d)\frac{P(ce)}{P(c)}\frac{P(cdf)}{P(cd)}$$



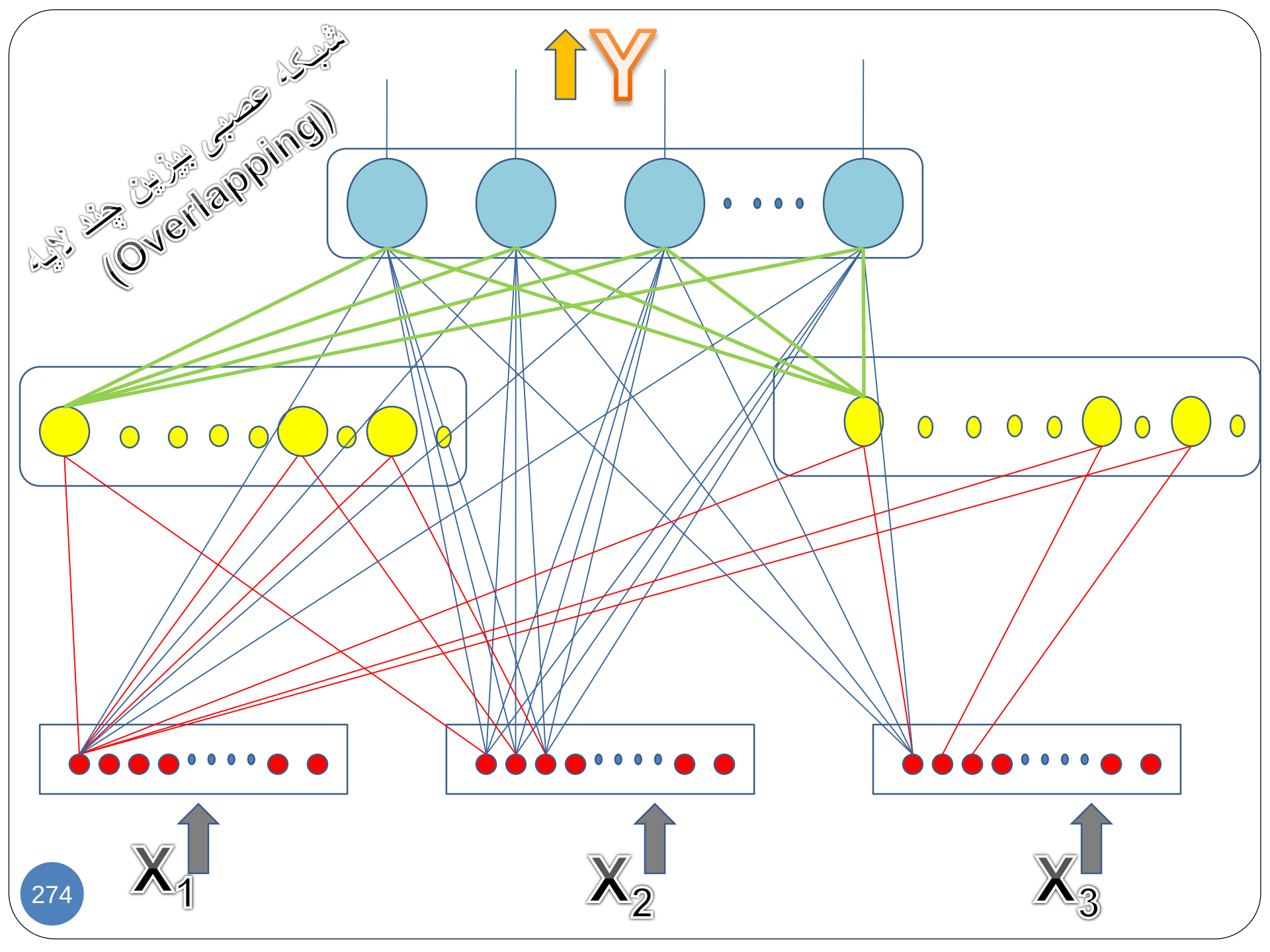
$$P(x) = P(a)P(b)P(c)P(d)P(e)P(f).$$

$$\cdot\left(\frac{P(abc)}{P(a)P(b)P(c)}\right)\left(\frac{P(ce)}{P(c)P(e)}\right)\left(\frac{P(cdf)}{P(c)P(d)P(f)}\right)$$

یک شبکه عصبی بیزین چند لایه در حالت روی هم افتادگی:



در این نوع معماری شبکه عصبی بیزین چند لایه ستون (ویژگی) های اولیه پس از ترکیب بطور مستقیم نیز با لایه خروجی ارتباط دارد .


شبکه عصبی پیرنین چند تایه
(Overlapping)
Y
X1
X2
X3

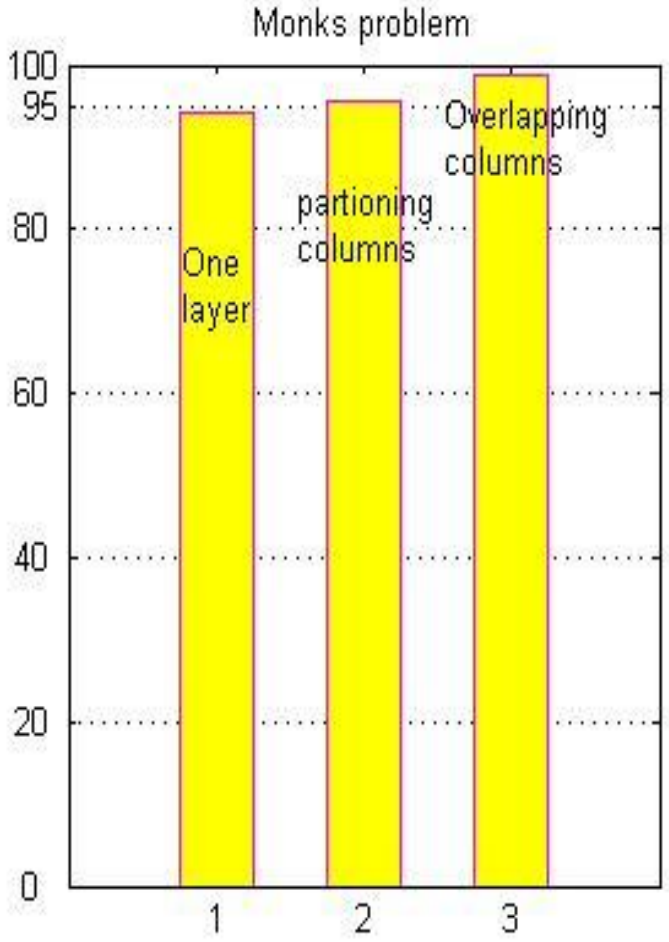
بدست آوردن وابستگی میان ویژگی ها :

اطلاعات متقابل میان دو ویژگی X,Y به صورت زیر تعریف می شود :

$$I(X,Y) = \sum_{i,j} P(x_i, y_j) \log \left(\frac{P(x_i, y_j)}{P(x_i)P(y_j)} \right)$$

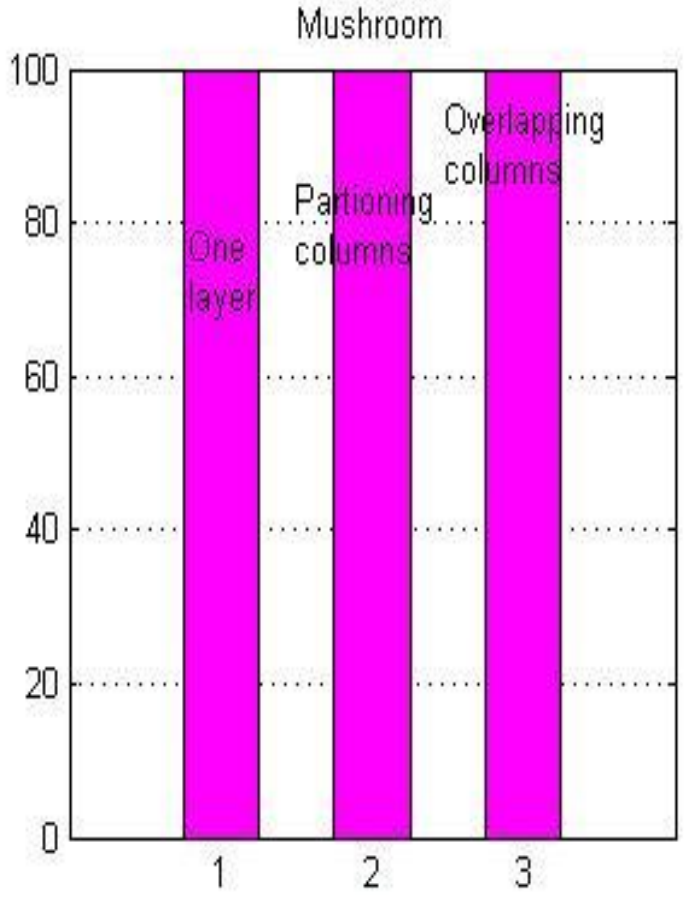
اگر مقدار معیار بالا از یک threshold بیشتر شود ، دو ویژگی وابسته تلقی می شوند و یک ستون پیچیده برای ترکیب آنها درنظر گرفته می شود .

نتایج پیاده سازی برای پایگاه داده monk’s problem:



Bayesian Solution	Classic Solution	Number of training data
97.22	97.22	122
96.99	96.99	61
88.88	88.42	16
84.49	81.01	13
77.31	75.00	10
61.34	59.49	7

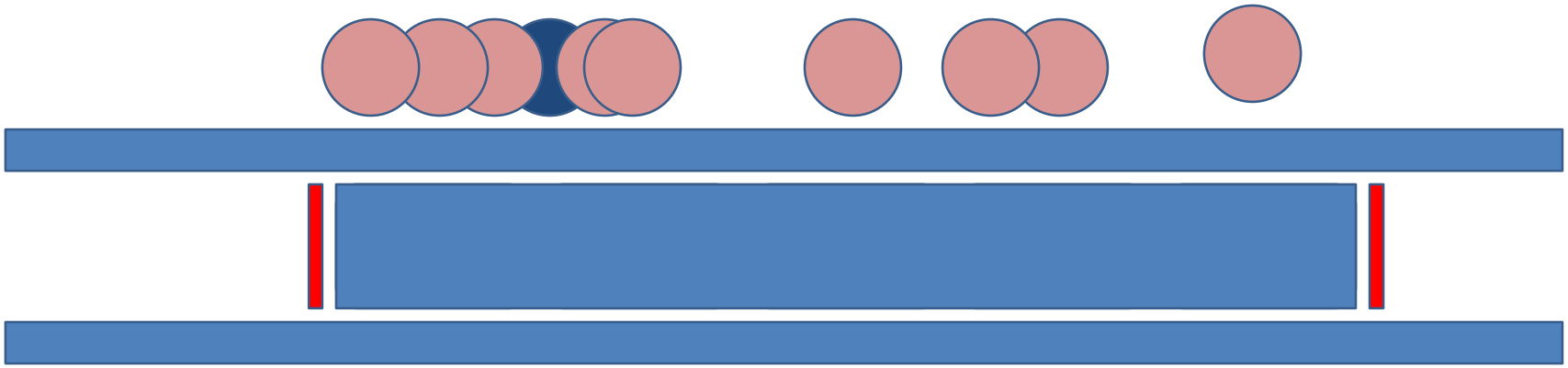
نتایج پیاده سازی برای پایگاه داده Mushroom:



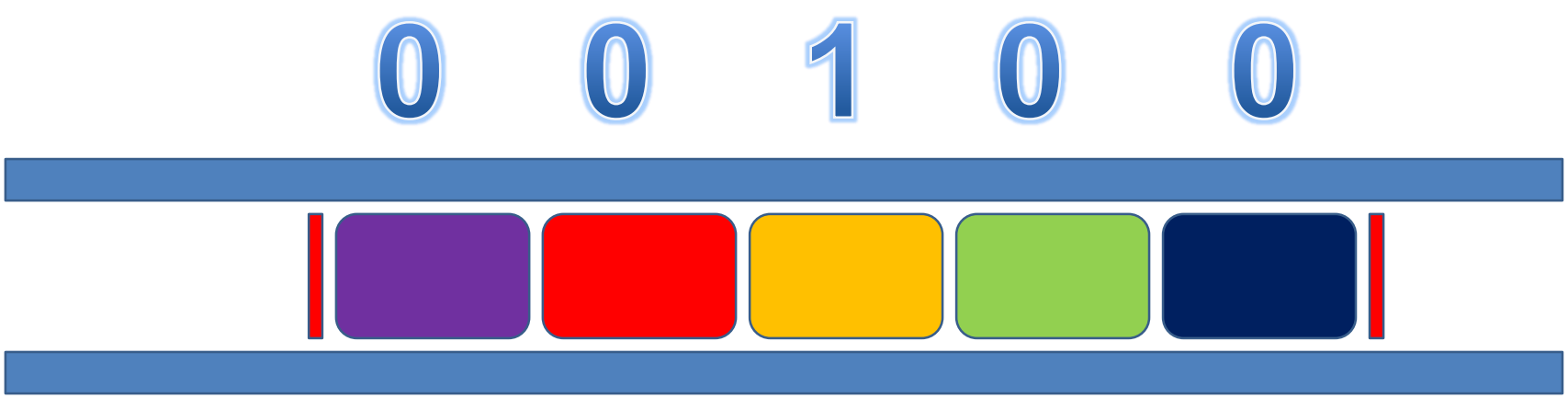
Bayesian Solution	Classic Solution	Number of training data
99.68	99.13	4000
99.51	98.74	2708
99.20	98.20	1354
99.42	98.02	903
98.94	96.95	677
98.48	96.93	542
99.11	96.63	452
98.15	96.32	387
97.23	94.38	339
97.87	97.11	301
97.96	96.27	271
97.00	93.85	247
95.73	92.70	226
97.96	94.92	209
96.95	95.19	194
96.80	93.92	170
97.35	94.59	138
97.30	95.43	118
95.34	92.96	94
95.43	94.71	83
93.42	92.42	62
92.92	91.19	41

ویژگی های پیوسته و شبکه عصبی بیزین

تقسیم بندی سخت بازه و آموزش شبکه عصبی :



فعال و غیر فعال شدن واحدها در حین دسته بندی:

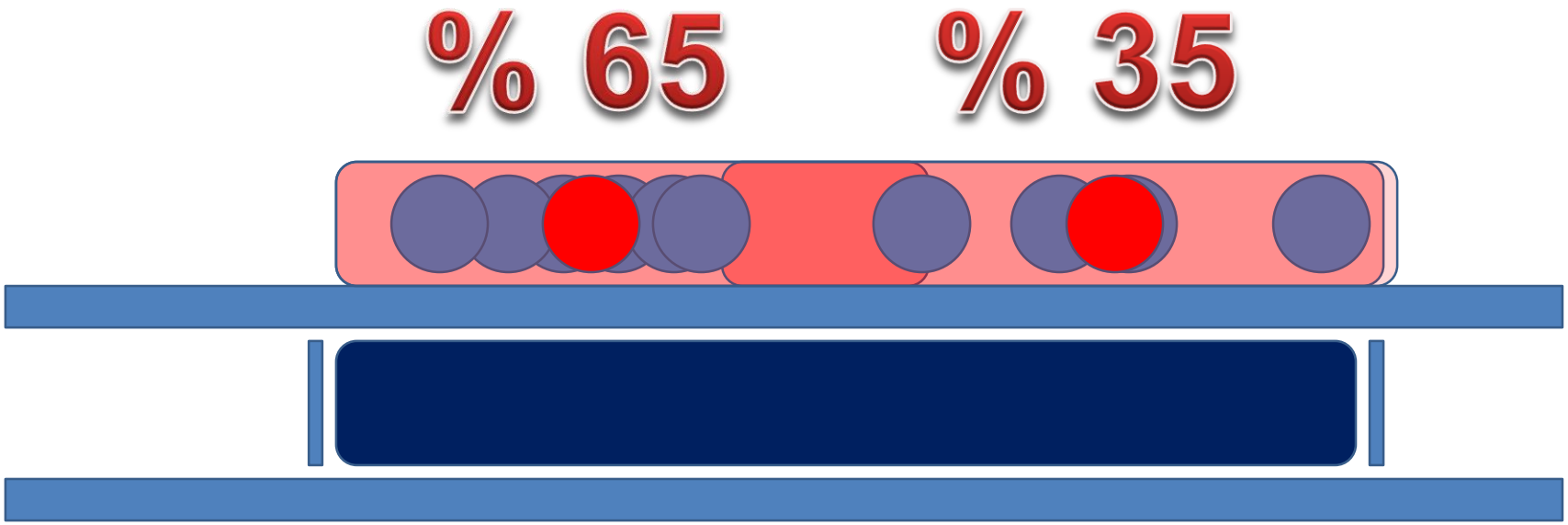


تقسیم بندی نرم بازه:

تابع چگالی احتمال بروی متغیر پیوسته z می تواند به وسیله یک تعداد متناهی از جمع تقریب زده شود:

$$P(z) = \sum_{i=1}^{n} P(v_i) P(z \mid v_i)$$

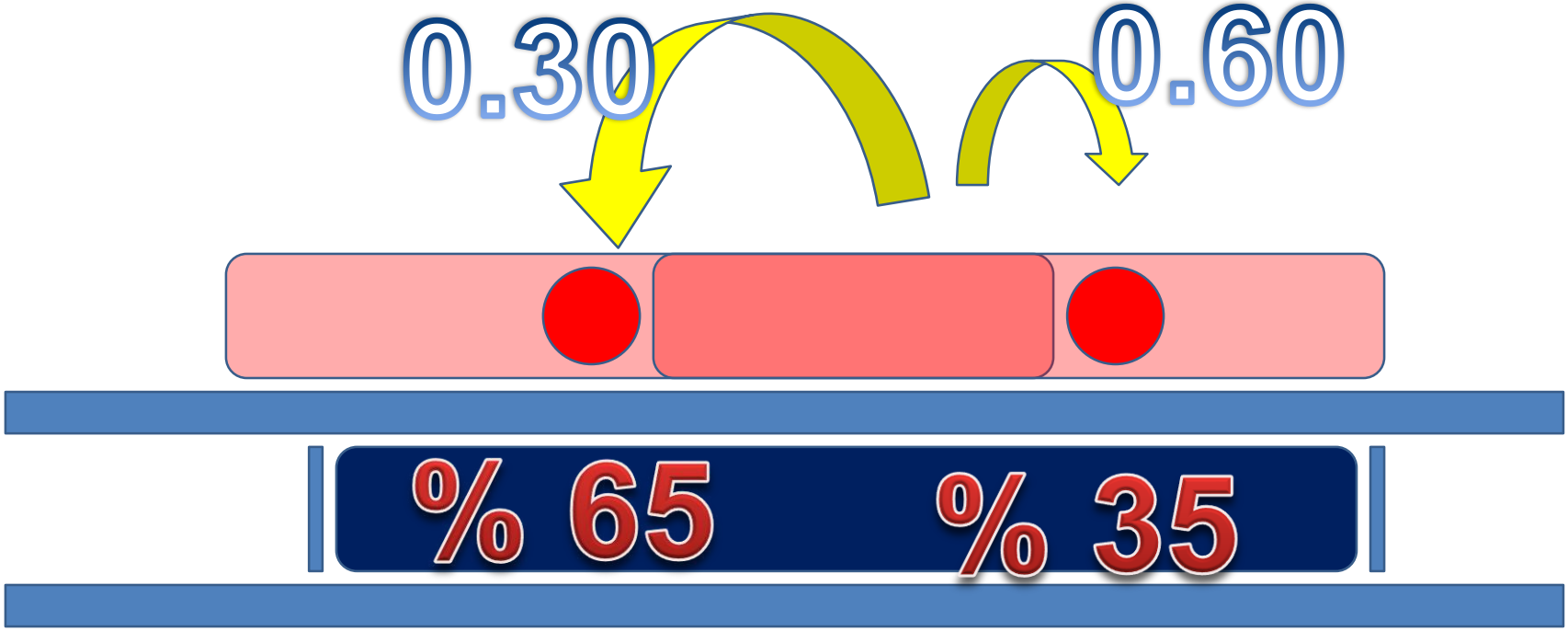
تقسیم بندی نرم بازه و آموزش شبکه عصبی :



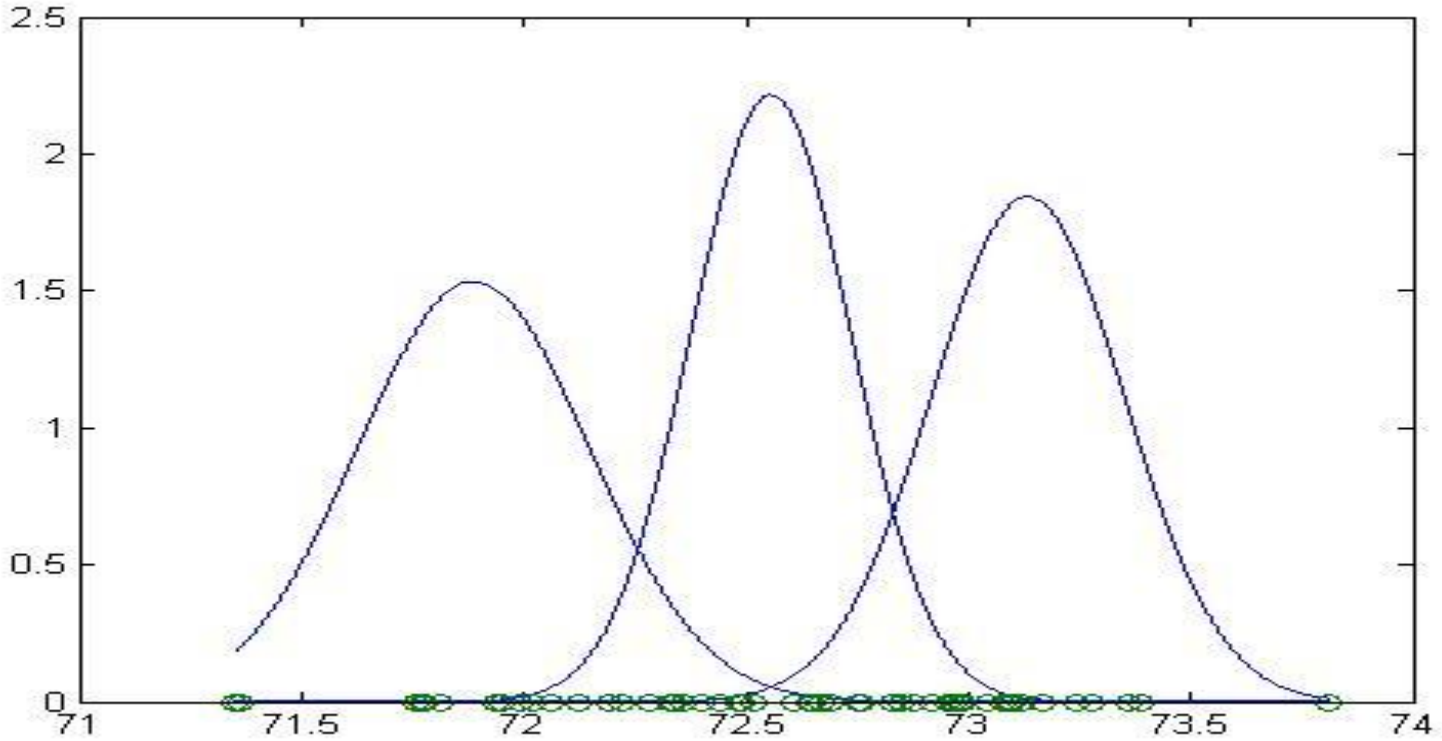
فعال و غیر فعال شدن واحدها در حین دسته بندی:

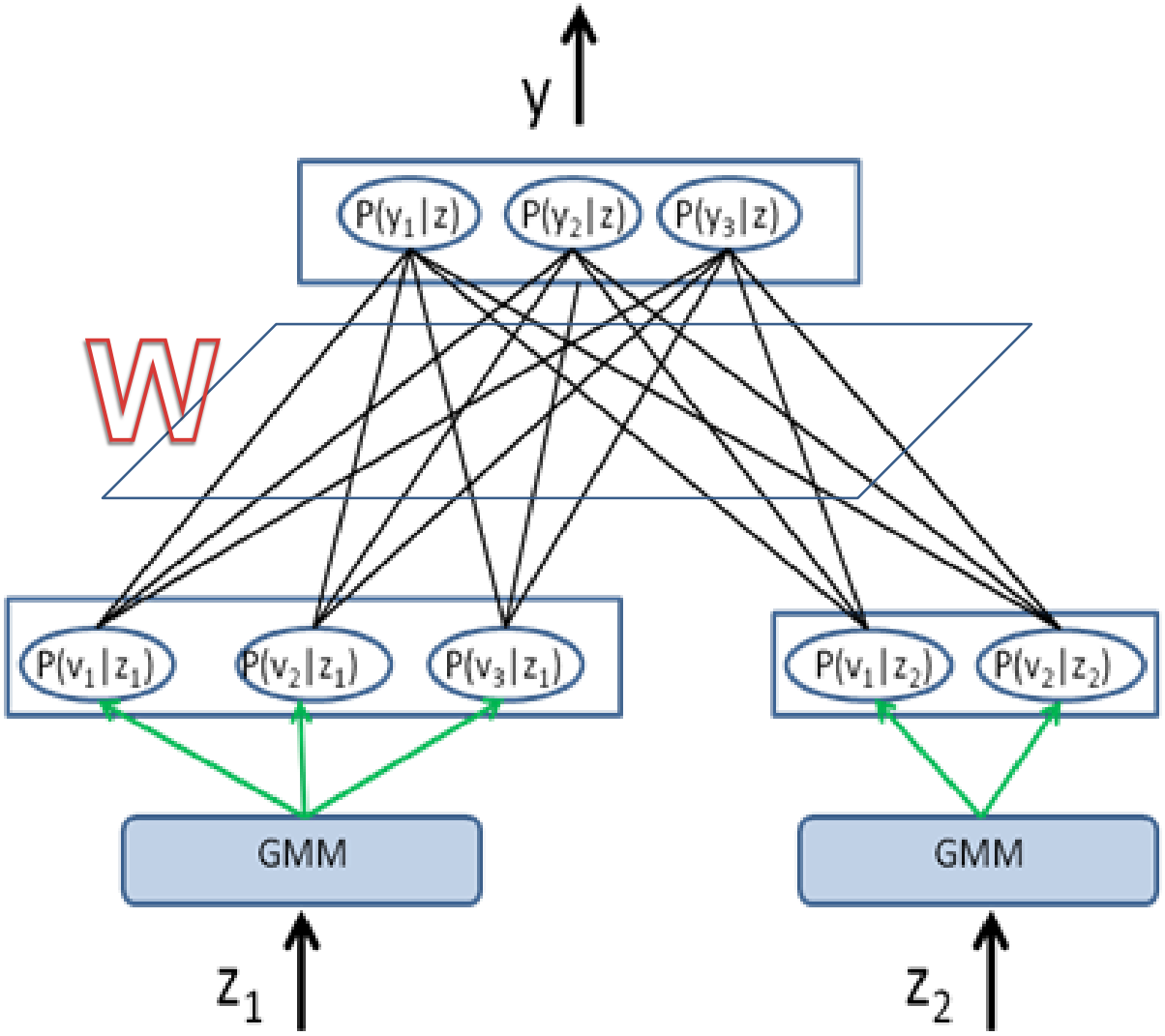
0. 65 * 0.30 =0.195

0.35 * 0.60 = 0.21



هر مقدار در بازه به درجه متفاوتی از جزء متفاوتی "متعلق" خواهد بود :




y
P(y1|z)
P(y2|z)
P(y3|z)
W
P(v1|z1)
P(v2|z1)
P(v3|z1)
P(v1|z2)
P(v2|z2)
GMM
GMM
z1
z2

$$\log(p(y \mid z)) =$$

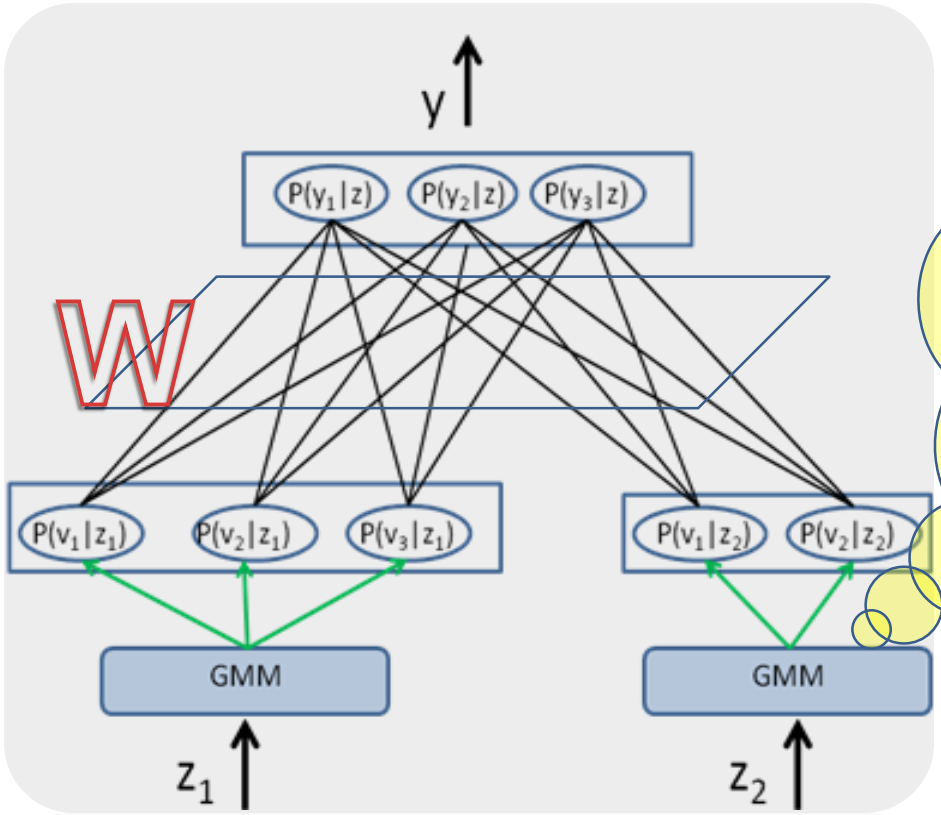
$$\log(\, p(y)) + \sum_i \log(\sum_{i'} \frac{P(y \mid v_{ii'})}{P(y)P(v_{ii'})} P(v_{ii'} \mid z_i))$$

$$S_j = \beta_j + \sum_i W_{ij} o_i$$

$$S_j = \log(P(y_j \mid z))$$

$$\beta_j = \log(P(y_j))$$

$$W_{i,j} = \frac{P(y_j, v_i)}{P(y_j)P(v_i)}$$



مدل ترکیبی

مدل ترکیبی :

یک سری از توابع که اجتماع دامنه های آنها یک بازه را پوشش می دهد.

در این کار، تبدیل یک متغیر پیوسته به تعدادی متغیر گسسته که بتوانند به عنوان ورودی شبکه عصبی بیزین استفاده شوند، توسط ***مدل ترکیبی*** با ***توابع گوسین*** صورت می گیرد.

توزیع گوسین با بـردار میـانگین خـود، μ و مـاتریس کوواریانس خود، Σ، شناخته می شـود و دارای تـابع توزیع زیر است :

$$f_i(z:\mu_i,\Sigma_i) = \frac{1}{\sqrt{(2\pi)^d \mid \Sigma_i \mid}} \exp\left(-\frac{1}{2}(z-\mu_i)\Sigma_i^{-1}(z-\mu_i)^{'}\right)$$

مسئله ای که مطرح است، تخمین پارامترهای ترکیبات مدل ترکیبی گوسین است.

$\pi_i = P(v_i)$

۴

$$f_i(z;\mu_i,\Sigma_i) = \frac{1}{\sqrt{(2\pi)^d \mid \Sigma_i \mid}} \exp\left(-\frac{1}{2}(z-\mu_i)\Sigma_i^{-1}(z-\mu_i)^{'}\right)$$

۱

$$\pi_i = \sum_{\gamma=1}^{c} P(v_i \mid z^{(\gamma)}) / C$$

۲

$$\mu_i = \frac{\sum_{\gamma=1}^{c} P(v_i \mid z^{(\gamma)}) z^{(\gamma)}}{C\pi_i}$$

۳

$$\sum_i = \frac{\sum_{\gamma=1}^{c} P(v_i \mid z^{(\gamma)})(z^{(\gamma)}-\mu_i)(z^{(\gamma)}-\mu_i)^{'}}{C\pi_i}$$

Expectation Maximization

با فرض برخی مقادیر اولیه تصادفی برای پارامترها، از معادله (۴) می توان برای محاسبه مقدار تعلق هر نمونه به هر جزء، استفاده کرد. این مرحله، مقدار مورد انتظار است.

سپس پارامترهای تخمین زده شده با معادلات (۱) تا (۳) مقادیر معادله (۴) را بدست می دهند. این مرحله، ماکزیمم سازی است.

این دو مرحله تا زمانی که پارامترها همگرا شوند، تکرار می شوند.

بررسی الگوریتم ماکزیمم سازی مقدار مورد انتظار:

الگوریتم ماکزیمم سازی مقدار مورد انتظار دارای ضعف هایی همانند زیر می باشد.

*)این روش یک روش محلی، و نسبت به پارامترهای اولیه انتخاب شده بسیار حساس می باشد و ممکن است به یک محدوده پارامترها که تخمین های نادرست را ارائه می دهد همگرا شود.

برای حل این مسئله چندین روش مورد بررسی قرار گرفته است که از یک یا ترکیبی از استراتژی هایی همانند چندین نقطه شروع تصادفی و انتخاب تخمین نهایی بطوریکه آن تخمین دارای بالاترین احتمال باشد و یا انتخاب اولیه پارامترها به وسیله الگوریتم خوشه بندی استفاده می کنند.

بررسی الگوریتم ماکزیمم سازی مقدار مورد انتظار:

*)مسئله دیگری که کارایی ماکزیمم سازی مقدار مورد انتظار را تحت تاثیر قرار می دهد مشخص نبودن تعداد ترکیبات است.

برای تخمین تعداد بهینه ترکیبات نیز مطالعات زیادی صورت گرفته است. برای مثال معیار اطلاعات Akaike (AIC)، استنتاج بیزین Schwarz (BIC) ، معیار احتمال دسته بندی مجتمع (ICL).

الگوریتم K-Means

۱. مرحله اول : پیدا کردن مرکز دسته اول به صورتی که رابطه زیر را ماکزیمم کند :

$$E_n^1 = \sum_{t=1}^{T} \exp\{-||x_t - x_n||^2\} \quad , \quad 1 \le n \le T \, , \, k = 1$$

$$\mu_1 = x_{n^*} \quad where \quad n^* = \arg_{1 \le n \le T} \max E_n^1$$

الگوریتم K-Means

۲. مرحله دوم : به شرط داشتن (k-1)امین ترکیب، معیار هر داده کاندید برای مرکز ترکیب kام می تواند بصورت زیر نوشته شود:

$$E_n^k = \sum_{i=1}^{T} \sum_{m=1}^{K} I(x_i \in c_m)|x_i - m_k|^2$$

تخمین تعداد بهینه ترکیبات :

هنگامی که یک جزء اضافه می شود، اگر ارتباط میان kامین جزء و اجزاء قبلی هنوز هم مستقل باشد، پارامترها با تخمین دوباره از k جزء بدست می آیند. برعکس اگر حداقل یکی از اجزا با جزء اضافه شده همبسته باشد، باید پارامترها را در مدل ترکیبی (k-1) ملاحظه کنیم و تعداد ترکیبات را حداقل (k-1) مشخص کنیم.

رابطه متقابل میان اجزاء :

$$\varphi(i,k) = p(i,k) log_2 \frac{P(i,k)}{P(i)P(k)} \quad , \quad i = 1, \dots, k$$

$$P(i) = \frac{1}{T} \sum_{i=1}^{T} P(i|x_t)$$

$$P(i,k) = \frac{1}{T} \sum_{i=1}^{T} P(i|x_t)\, P(k|x_t)$$

رابطه متقابل میان اجزاء :

رابطه ی متقابل ممکن است دارای سه مقدار باشد: منفی، صفر و یا مثبت

اگر دارای مقدار مثبت باشد، بدین معنی است که i و k بطور آماری به یکدیگر وابسته اند.

الگوریتم تخمین تعداد بهینه ترکیبات :

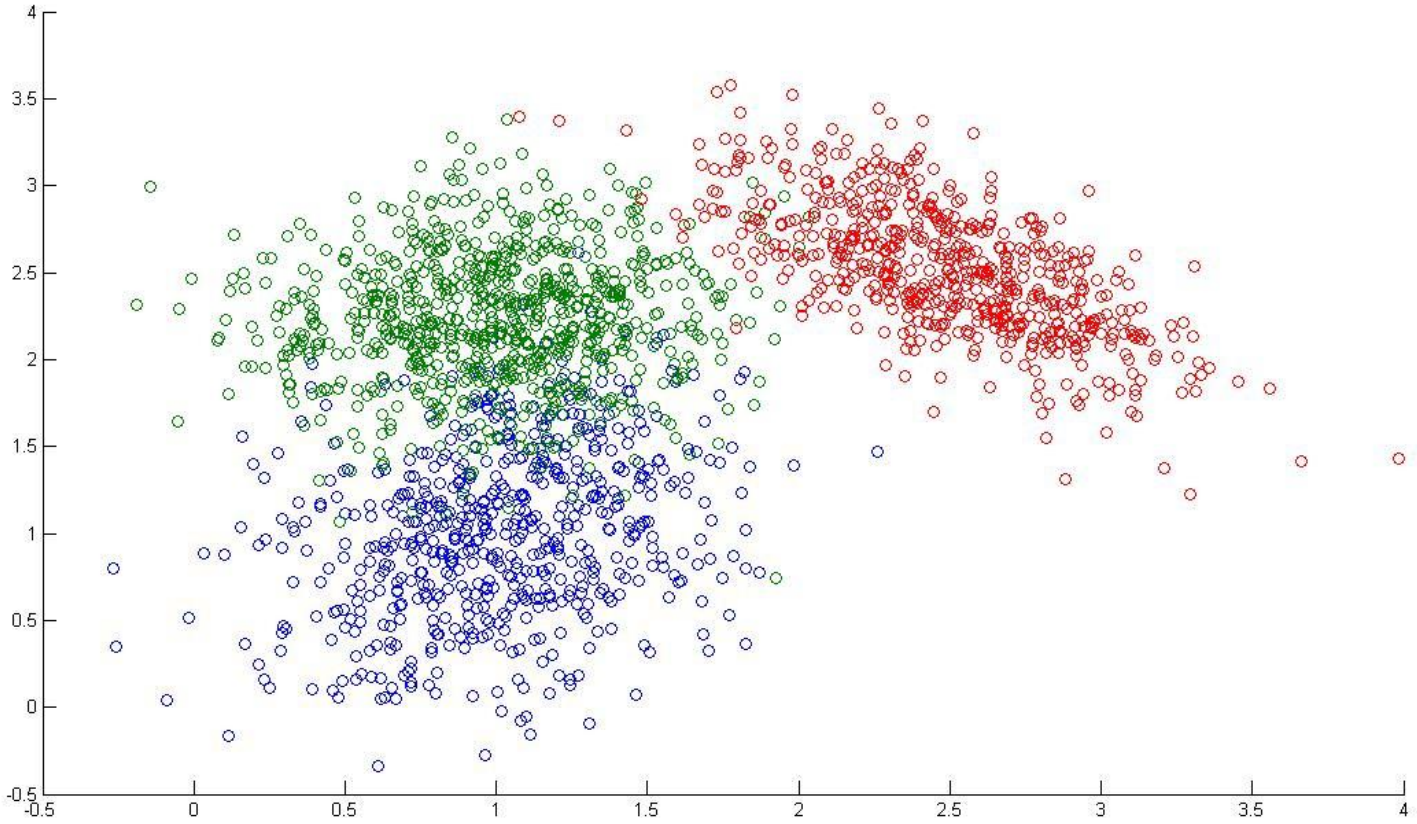
مرحله ی ۱ : برای $n=1,\ldots,T$ ، $k=1$ در نظر بگیرید و خطای فاصله ی E_n^1 را حساب کنید و مرکز μ_1 ترکیب را مشخص کنید.

مرحله ی ۲ : برای اضافه کردن یک جزء، با مینیمم کردن E_n^k مرکز ترکیب دیگر را بر اساس الگوریتم افزایشی k-میانه جستجو کنید.

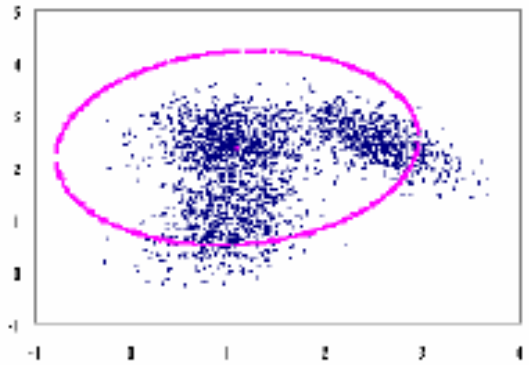
مرحله ی ۳ : به شرط داشتن مقادیر اولیه k مدل، برای تخمین زدن دوباره پارامترها، الگوریتم EM را با استفاده از معادلات (۲۴-۲)، (۲۶-۲) و (۲۸-۲) تکرار کنید تا احتمالات همگرا شود. سپس کمیت های P(i) و P(k) و P(i,k) برای محاسبه ی ارتباط متقابل آماده می باشند.

مرحله ی ۴ : برای $i=1,..,k$ ارتباط متقابل میان اجزاء را به وسیله ی (۳۲-۲) محاسبه کنید. اگر $\varphi(i,k)$ مثبت بود، آنگاه توقف کنید و عداد بهینه ترکیبات را $M=k-1$ قرار دهید وگرنه به مرحله ی (۲) بازگردید.

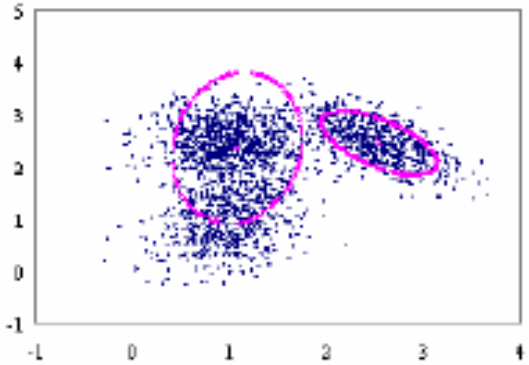
پیاده سازی بروی داده های مصنوعی :



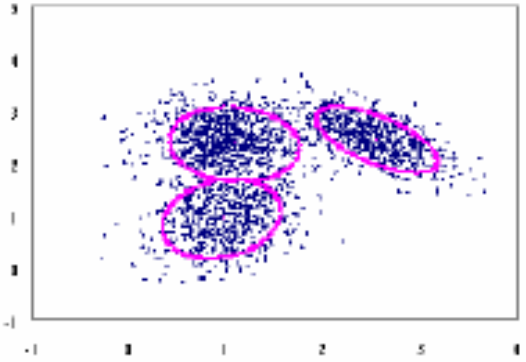
$$0.3N\left[x|\begin{pmatrix}1\\1\end{pmatrix},\begin{pmatrix}0.15 & 0.05\\0.05 & 0.25\end{pmatrix}\right]+0.4N\left[x|\begin{pmatrix}1\\1.25\end{pmatrix},\begin{pmatrix}0.15 & 0\\0 & 0.15\end{pmatrix}\right]+0.3N\left[x|\begin{pmatrix}2.5\\2.5\end{pmatrix},\begin{pmatrix}0.15 & -0.1\\-0.1 & 0.15\end{pmatrix}\right]$$



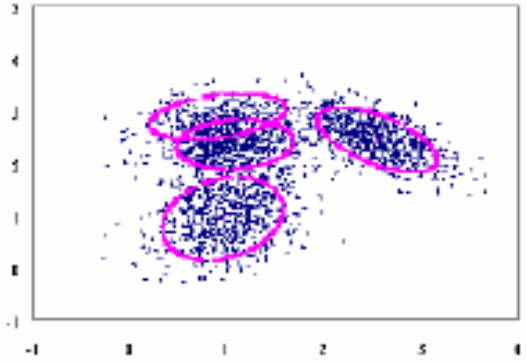
(a) k=1



(b) k=2



(c) k=3



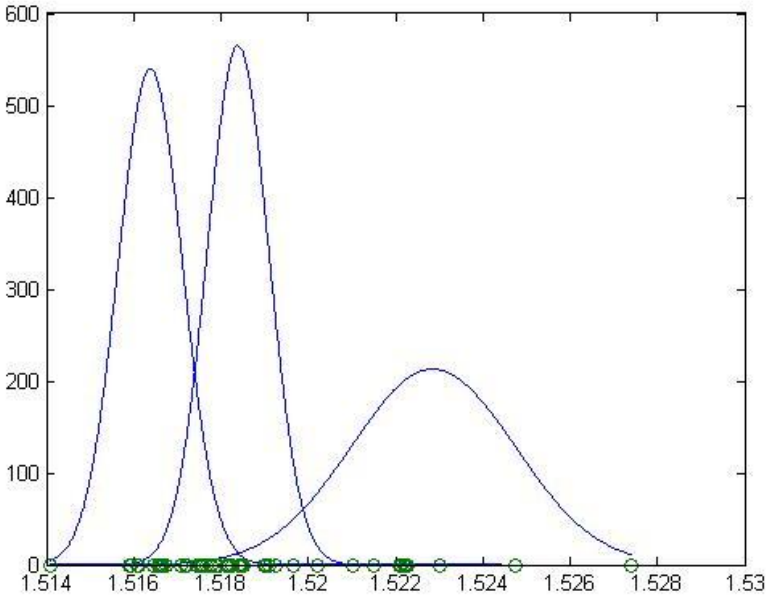
(d) k=4

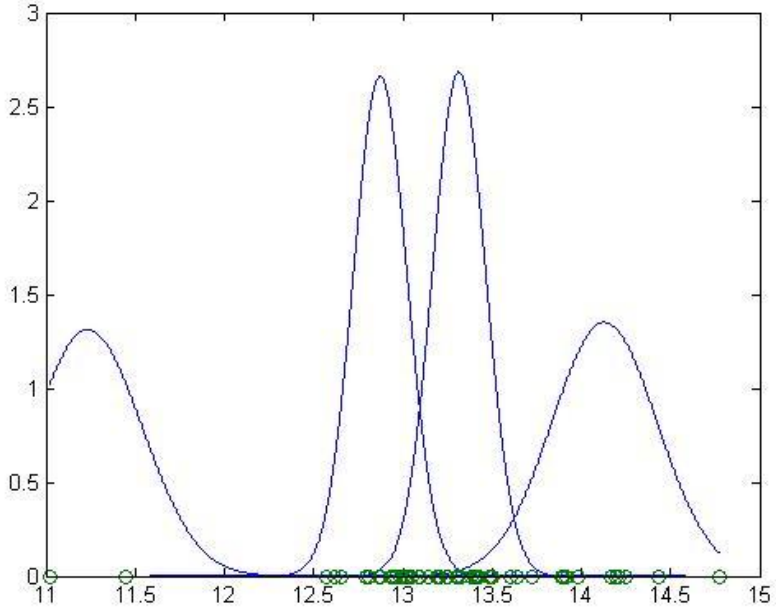
$\varphi(i,k)$

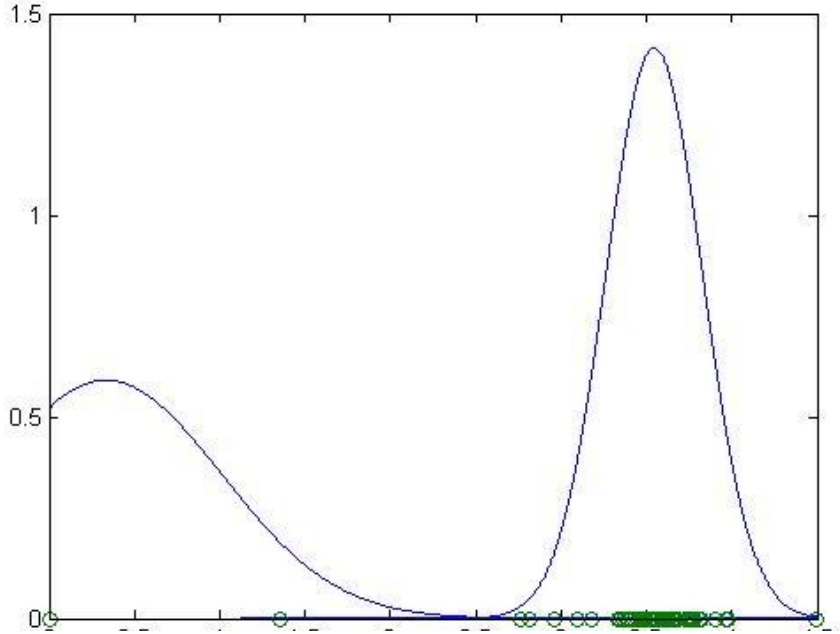
تعداد ترکیبات (k)	میانه ی iامین ترکیب	i	
1	(1.1620,2.0100)	1	~
2	(2.4915,2.5147)	1	-0.0674
3	(0.9851,0.8554)	1	-0.0346
		2	-0.0042
4	(0.8065,2.4638)	1	0.0255
		2	-0.0092
		3	-0.0080

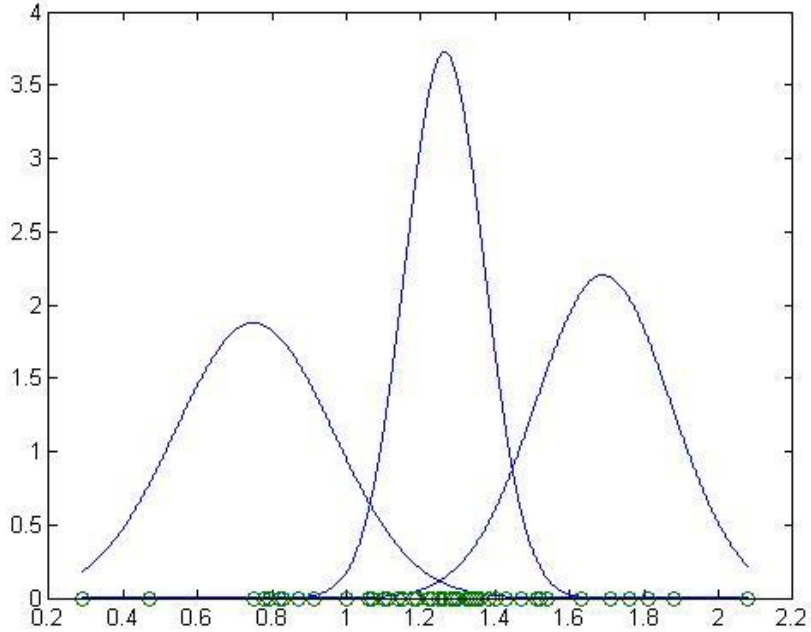
پیاده سازی و آزمایش GMM :

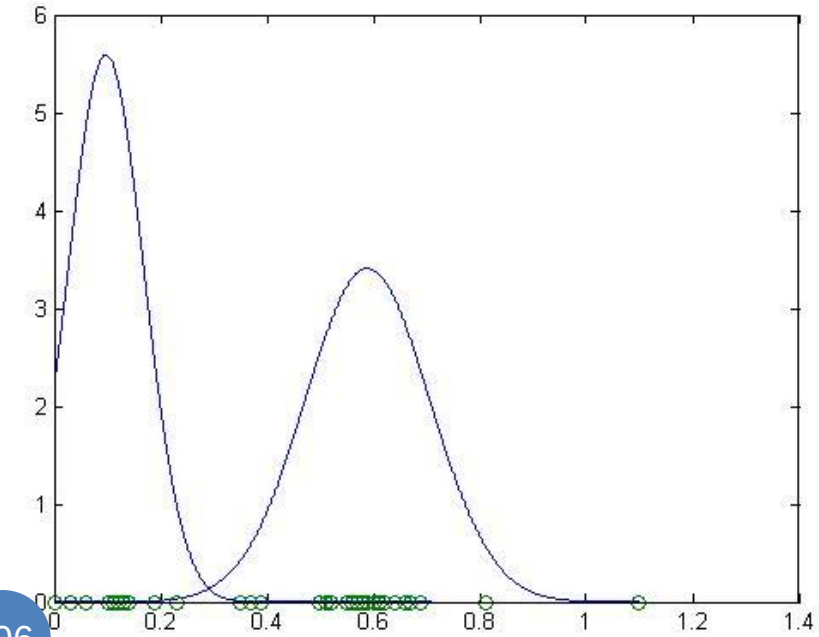
داده glass :

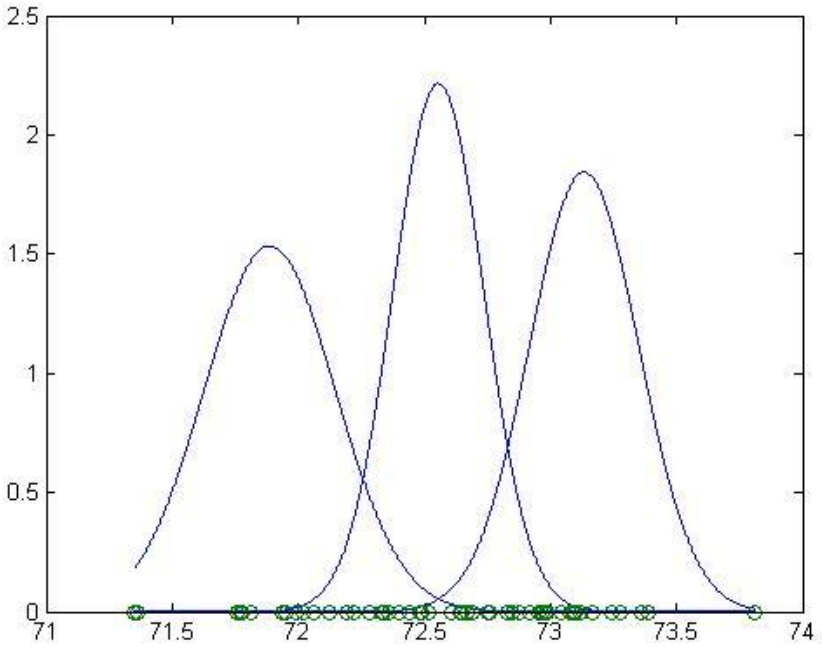


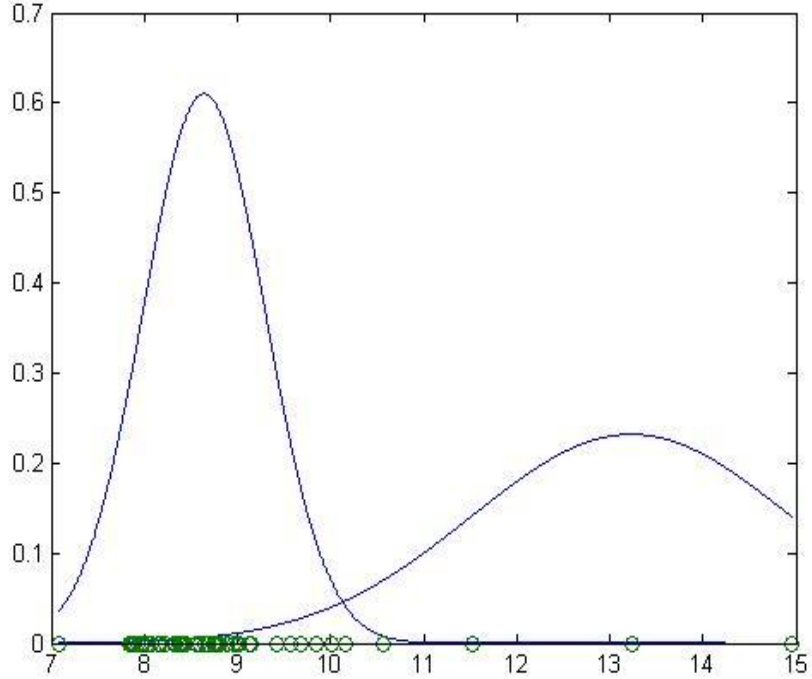


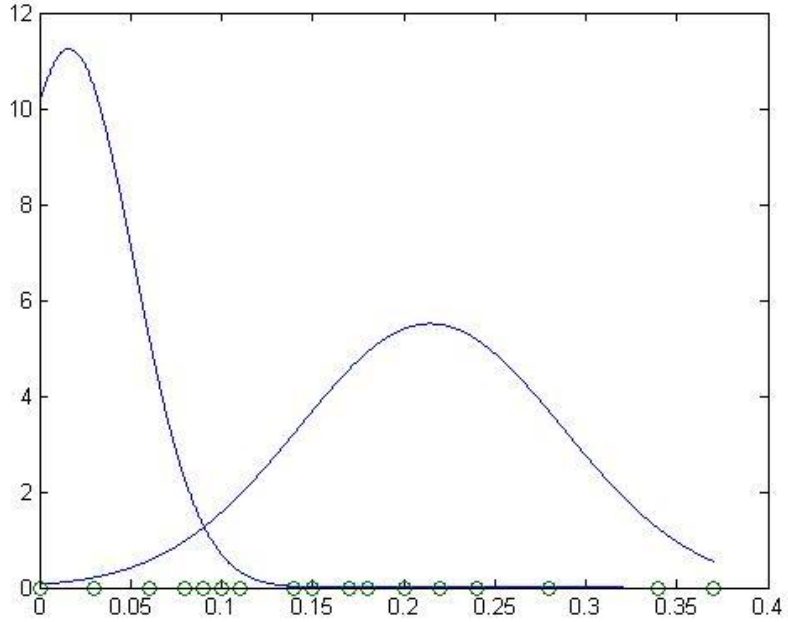

1.5
1
0.5
0

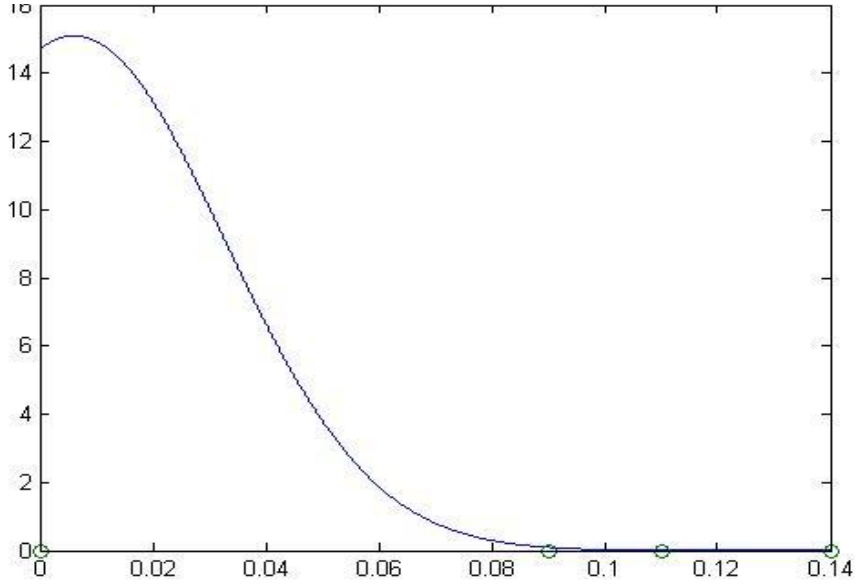

4
3.5
3
2.5
2
1.5
1
0.5
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
2.2


6
5
4
3
2
1
0
0.2
0.4
0.6
0.8
1
1.2
1.4

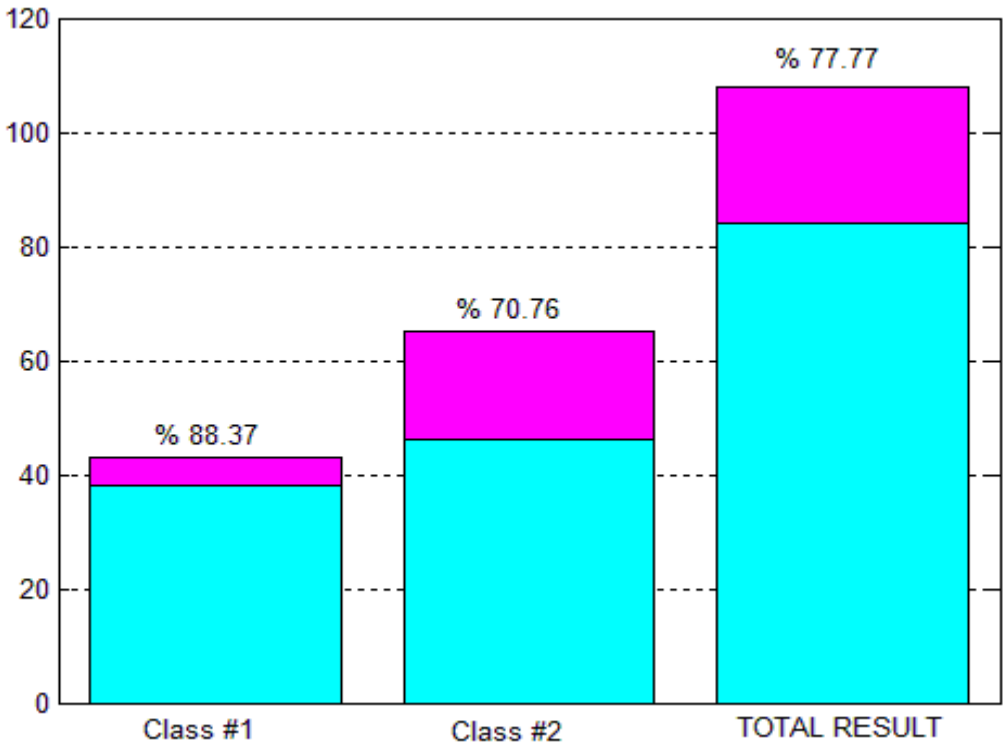

2.5
2
1.5
1
0.5
0
71
71.5
72
72.5
73
73.5
74


0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
7
8
9
10
11
12
13
14
15

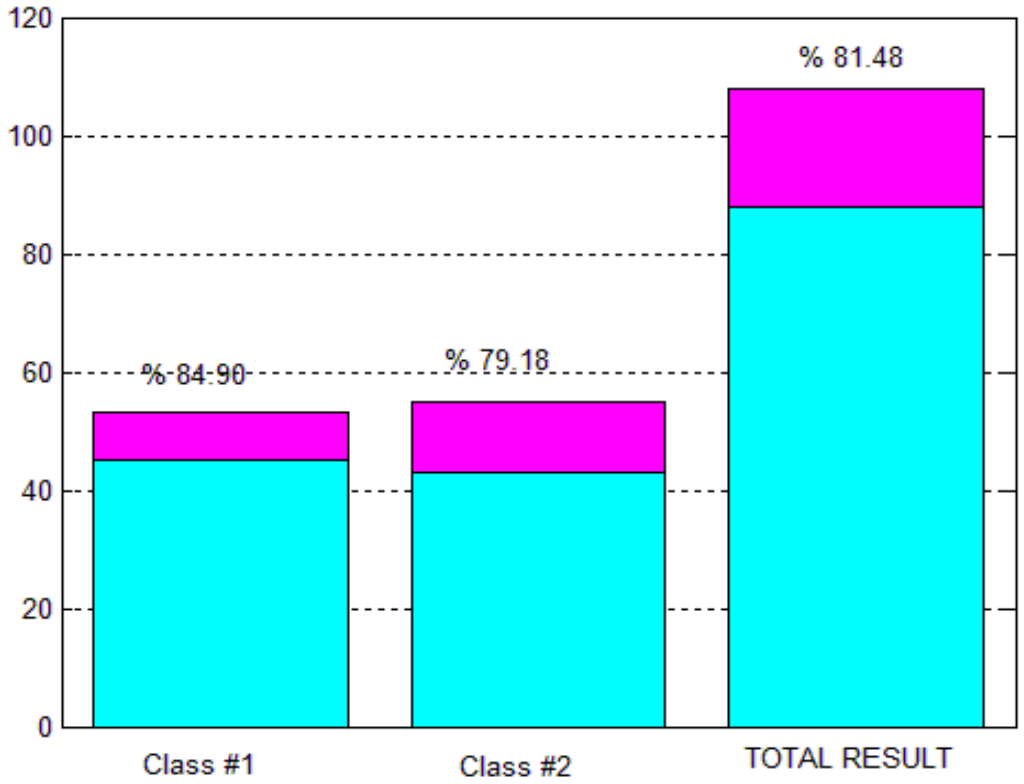

12
10
8
6
4
2
0
0
0.05
0.1
0.15
0.2
0.25
0.3
0.35
0.4


14
12
10
8
6
4
2
0
0
0.02
0.04
0.06
0.08
0.1
0.12
0.14

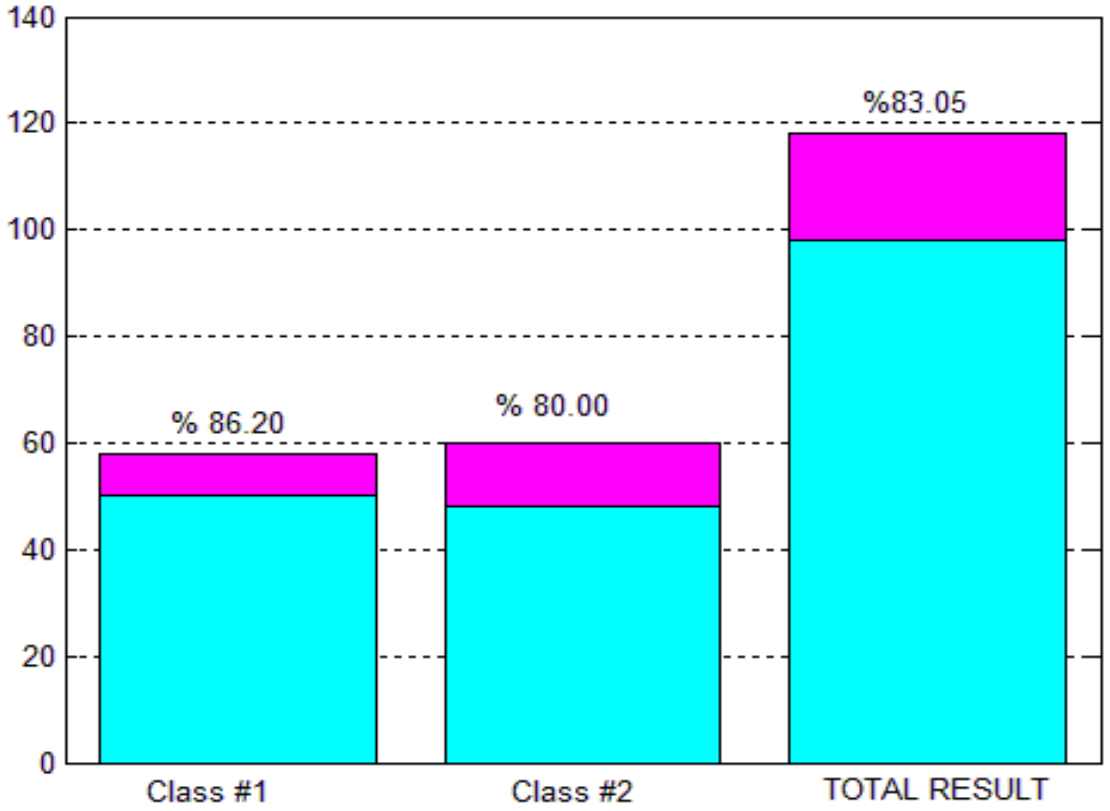
نتیجه دسته‌بندی با استفاده از شبکه عصبی بیزین تک‌لایه



نتیجه دسته‌بندی با استفاده از شبکه عصبی بیزین چندلایه در حالت قسمت بندی



دسته‌بندی با استفاده از شبکه عصبی بیزین چندلایه در حالت روی هم افتادگی



فصل دهم : شبکه عصبی مثلثاتی

فصل یازدهم :

شبکه عصبی ویولت Wavelet Neural Network

فصل دوازدهم : شبکه عصبی کانولوشنال

فصل دوازدهم :

کاربرد شبکه های عصبی

در شناسایی، پیش بینی و کنترل سیستم ها

شناسایی، پیش بینی و کنترل

- شبکه های عصبی به عنوان شناساگر
- شبکه های عصبی در پیش بینی سیستم ها
- شبکه های عصبی به عنوان کنترلر

کنترلر عصبی

- کنترلر عصبی تقلیدگر
- کنترلر و شناساگر مستقیم
- کنترلر و شناساگر معکوس
- کنترل معکوس تطبیقی
- کنترل مدل داخلی غیرخطی
- کنترل تطبیقی عصبی
- نقاد تطبیقی
- کنترلر PIDخود تنظیم
- شبکه عصبی به عنوان جبرانگر
- کنترلر تطبیقی مدل مرجع عصبی مستقیم
- کنترل مدل پیش بین

کنترلر ترکیبی

- کنترلر ترکیبی از کنترل ساختار متغیر و کنترل عصبی
- کنترل تطبیقی مستقیم پایدار
- خطی سازی فیدبک تطبیقی عصبی
- کنترلر ترکیبی از کنترل فیدبک و کنترل عصبی
- آموزش خطای پسخور
- کنترلر ترکیبی از کنترل کنترل بهره ثابت و کنترل عصبی
- کنترلر شبکه عصبی بر پایه مُد لغزشی
- کنترلر بهینه عصبی

پایان