شبکه های عصبی - دکتر صفا بخش - پاییز ۱۴۰۰

جلسه اول - ۱۴۰۰/۶/۲۸

ایمیل درس : safacourses1400@gmail.com

بارم بندی : کار مطالعاتی : ۲۰٪ پروژه ها : ۵۰٪ امتحان پایانترم : ۳۰٪

حداقل ۵ مقاله، حداقل ۵۰ صفحه، سعی کنید مقالات ژورنال انتخاب کنید نه کنفرانس

مقدمه (Introduction)

اسامی مختلف شبکه های عصبی : Neural Networks, Neural Nets, NN

Artificial Neural Networks, ANN, connectionist Models,

parallel Distributed Processing (PDP) systems,

Neuromorphic systems, Neural Computing, Neurocomputing

Deep Neural Networks, Deep Learning

← سوال : شبکه های عصبی چه چیزی هستند؟

یک جواب خیلی کلی برای این سوال این است که شبکه های عصبی تلاش انسان است برای ایجاد کردن امکان و شرایطی برای تقلید از عملکرد مغز انسان.

← یک ویژگی مغز انسان این است که هم زمان می تواند کارهای متعددی را انجام بدهد و می تواند از روی اطلاعات مبهم یا ناقص یا متناقض نتیجه گیری کند. هم چنین در انجام کارهای طبیعی مثلا بینایی، خیلی بهتر از ماشین های امروزی کار می کند. مثال های دیگر : زبان، بازیابی اطلاعات، برنامه ریزی، مغز انسان در همه این موارد بهتر از ماشین ها عمل می کند. به همین دلیل علاقه داریم که در ماشین ها از عملکرد مغز انسان تقلید کنیم.

Scanned with CamScanner

روش های رایانش عصبی (Neural Computing) یک فرق اساسی با روش های کلاسیک حل مسئله دارند. در روش های سنتی باید یک الگوریتم ابداع کنیم و آن را پیاده سازی کنیم اما در رایانش عصبی نیاز به داشتن الگوریتم نداریم، هم چنین نیازی به پیاده سازی هم نداریم چون گاهی اوقات الگوریتم یافت می شود اما پیاده سازی آن سخت است. روش آن به این صورت است که با تعامل با محیط مسئله یاد می گیرد که پاسخی که شما انتظار دارید را پیدا کند.

نکته دیگری که رایانش عصبی را ارزشمند می کند این است که در روش های سنتی اگر افتی در هر بخش از سیستم رخ دهد، سیستم از کار می افتد اما در شبکه های عصبی اگر بخشی از شبکه دچار مشکل شود (مثلا گره ای از دست برود) شبکه به کار خودش ادامه می دهد فقط ممکن است نتیجه ای که شبکه می دهد بدتر شود. این ویژگی فوق العاده مهم است به خصوص در سیستم های حساس. به این ویژگی gracefull degradation (خراب شدن آبرومندانه) گفته می شود.

پیاده سازی شبکه های عصبی

① شبیه سازی نرم افزاری ← کتابخانه های متعدد

② پیاده سازی سخت افزاری ← مزیت این روش ها سرعت بیشتر نسبت به شبیه سازی های نرم افزاری است، هم چنین می توانند به عنوان اجزایی در شبکه های پیچیده مورد استفاده قرار بگیرند

Scanned with CamScanner

جلسه سوم - ۱۴۰۰/۷/۴

فرض کنید یک نورون فعال می‌شود این فعال شدن چگونه بر نورون های دیگر تأثیر می‌گذارد؟

وقتی پتانسیل به سیناپس می‌رسد کیسه های مواد انتقال دهنده به سمت شکاف سیناپسی می‌روند و مواد خود را تخلیه می‌کنند و این مواد بعد می‌شکنند در دیواره ی شکاف و جلوی عبور و رسیدن به نورون پس سیناپسی

مولکول های گیرنده در سلول پس سیناپسی منتظر دریافت ماده انتقال دهنده اند

اگر نوع انتقال دهنده محرک یا فاداننده باشد پتانسیل استراحت سلول پس سیناپسی به سمت صفر می‌رود (Depolarization) اختلاف پتانسیل در دو سر غشا این سلول کاهش می‌یابد در نتیجه سلول آماده ی تحریک شدن می‌شود

در یک زمان خاص اگر تحریکات مختلف از ورودی های مختلف نورون وارد شود، اثر آن ها با هم جمع می‌شود و اگر از یک حد آستانه بیشتر شود سلول فعال می‌شود

اگر ماده منتقل کننده در ناحیه پس سیناپسی تجمع پیدا کند سلول به یک حالتی می‌رود که به آن حد آستانه تحریک یا دپولاریزاسیون ثابت می‌گویند. مواد پیوسته جریان دارد از این سلول به سلول های متصل به هم. در یک حالتی طرف مولکول های گیرنده تکمیل می‌شود دیگر نمی‌توانند اثر ایجاد کنند اگر بیشتر شود اگر ماده شرایط آن را فراهم کند که بالای این ترکیب را می‌شکند و ماده شرایط بعدی آن را فراهم می‌کند

اگر مواد انتقال دهنده از نوع بازدارنده باشند (inhibitory): (می‌خواهد مانع ایجاد تحریک شود)

اختلاف پتانسیل را می‌خواهد افزایش دهد. هایپرپولاریزاسیون سلول پس سیناپسی زیاد می‌شود مانع را زیاد می‌کند برای فعال شدن سلول بعدی.

جمع جبری این پتانسیل ها اگر به حد آستانه برسد سلول فعال می‌شود.

دریافت اطلاعات در مغز

مدارهای واگرا: یک نورون چند نورون بعدی را تحریک می‌کند و دوباره هر کدام از آن ها چند نورون بعدی را و ... ساختار درختی ایجاد می‌شود. مثال: احساسات

مدارهای همگرا: چند نورون می‌خواهند یک نورون را تحریک کنند مثلا در کارهای حرکتی (motor)

مدار موازی: کنار هم اند، چند یک در آن ها وجود دارد. باعث می‌شود عمل تکراری مدت تکرار بشه مثل ضربان قلب

Scanned with CamScanner

مدار موازی پس از تخلیه ← به سری مسیرهای موازی که می‌خواهند یک نورون را فعال کنند.

مثال: ارسال هم زمان یک سری ضربات سریع

شبکه‌های عصبی مصنوعی (Artificial Neural Networks)

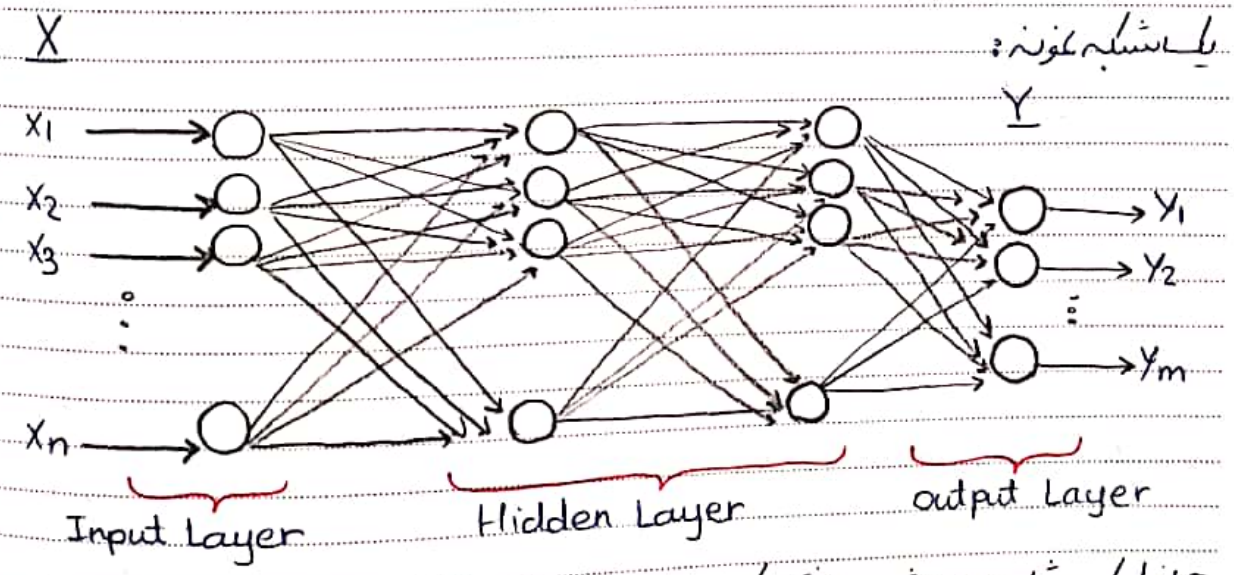
← یک شبکه عصبی یک ساختار موازی توزیع شده به شکل یک گراف جهت دار است. در این گراف هر گره نشان دهنده یک نورون یا واحد پردازشگر است (Neuron or processing unit)

← هر کدام از نورون‌ها فقط یک خروجی دارد ولی تعداد دلخواهی ورودی می‌تواند داشته باشند. خروجی هر نورون می‌تواند به تعداد دلخواهی نورون‌های دیگر برود. همه‌ی این خروجی‌ها با هم برابرند.

← هر اتصال در این گراف یک جهت و یک وزن دارد. جهت نشان می‌دهد که در این اتصال کدام گره روی کدام گره اثر می‌گذارد و مبدأ و مقصد را مشخص می‌کند و قوت آن اثر را وزن نشان می‌دهد.

← گره‌ها گروه‌بندی می‌شوند به یک سری لایه‌هایی (در مورد مغز هم همین طور است)

یک شبکه نمونه:



همانطور که مشاهده می‌شود ساختار به گونه‌ای است که گره‌ها لایه لایه هستند

یک لایه هست که از خارج ورودی‌ها را قبول می‌کند که به آن لایه ورودی می‌گویند. نورون‌های این لایه سیگنال‌ها را از خارج می‌گیرند و به لایه‌های بعدی در شبکه می‌دهند.

Scanned with CamScanner

یک لایه هست که نتیجه شبکه را به بیرون می دهد. (لایه خروجی) و نورون های آن را نورون های خروجی می نامند

لایه هایی که بین این دو لایه قرار دارند لایه های مخفی (hidden layer) نامیده می شود چون

نه از ورودی و نه از خروجی دقتی نگاه می کنیم این لایه ها را می بینیم مخفی هستند.

معمولا ورودی ها را با x و خروجی ها را با y نشان می دهیم. معمولا تعداد بعد ورودی را با n و

تعداد بعد خروجی را با m نشان می دهیم.

هر نوع نود می تواند به هر نوع نود در لایه دیگر وصل باشد (مثلا از لایه اول میتونه به لایه چهارم وصل بشه)

فرض می کنیم که همه ی نورون های یک لایه شبیه هم هستن

هیچ تأخیری در نظر نمی گیریم (برخلاف مدارهای منطقی) از ورودی به خروجی.

عملکرد نورون ها ۲ حالت می تواند باشد:

① synchronous (سنکرون) ← در گرافی که داریم نورون ها در زمان های خاصی فعال می شوند (episodical)

② Asynchronous (آسنکرون) ← عملکرد نورون ها پیوسته است و هر وقت در ورودی تغییری (continuous)

ایجاد شود بلافاصله اثر آن در خروجی ظاهر می شود

← می توان گفت که شبکه ما در این حالت مثل مدارهای منطقی است.

← برای راحتی فرض می کنیم همه ی نورون های یک شبکه سنکرون یا آسنکرون است

* وزن ها در یک فرآیند یادگیری، یادگرفته می شوند. این یادگیری به چند چیز نیاز دارد:

① قانون یادگیری: شبکه با محیط تعامل می کند تا یاد بگیرد و این تعامل از طریق داده هایی که از محیط

وارد شبکه می شود صورت می گیرد.

← سلول های واقعی عملکرد بسیار پیچیده تری دارند. همیشه آسنکرون هستند. همیشه ترکیب غیرخطی از

ورودی ها را در نظر می گیرند.

Scanned with CamScanner

← خروجی در نورون های واقعی ضربه یا دنباله ای از ضربه هاست در حالی که در شبکه عصبی مصنوعی به شکل سطح در نظر گرفته می شود. مثلا سطح ۱ و ۰

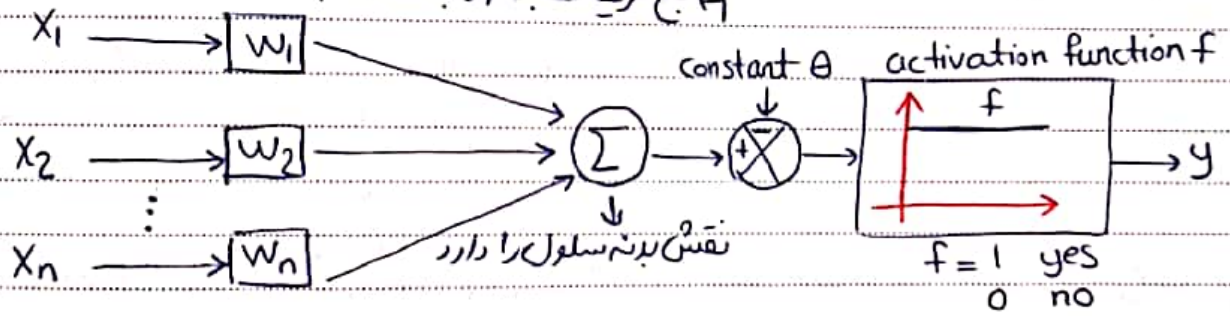
← در نورون های واقعی مثل مدار منطقی تاخیر وجود دارد و پیچیدگی های ایجاد می کند

← تخلیه مواد در سیناپس می تواند به صورت اتفاقی عرض شود تحت عوامل مختلف و خیلی قابل پیش بینی نیست.

← مدل : McCulloch & Pitts

اثر سیناپس را با وزن هاشان می دهیم. اگر وزن مثبت باشد اثر تحریک کنندگی دارد و اگر منفی باشد اثر بازدارندگی دارند.

← جمع تحریکات بدنه را با مقدار ثابت θ مقایسه می کند



اگر نتیجه تحریکات منهای θ مثبت باشد تابع فعال سازی ۱ می دهد در غیر اینصورت ۰ می دهد

$$w_1x_1 + w_2x_2 + \dots + w_nx_n - \theta \xrightarrow{f} y = f(w_1x_1 + \dots + w_nx_n - \theta)$$

$$y = f\left(\sum_{i=1}^{n} w_i x_i - \theta\right) = f\left(\sum_{i=0}^{n} w_i x_i\right) \qquad x_0 = 1, w_0 = -\theta$$

باید $w_0 x_0 = \theta$ باشد به همین دلیل x_0 را ۱ می گیریم

در این مدل که متعلق به McCulloch و Pitts است تابع فعال سازی تابع پله است.

می توانیم در این مدل تغییراتی ایجاد کنیم:

۱) تابع فعال سازی را تغییر بدهیم

اسامی مختلفی برای تابع فعال سازی عنوان شده است:

- Activation function
- Squashing function
- Transfer function
- Gain function

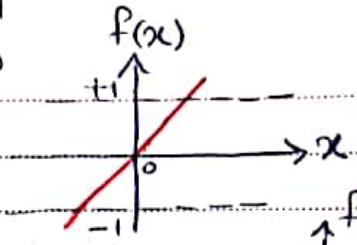
روز سلامتی

Scanned with CamScanner

انواع تابع فعال سازی :

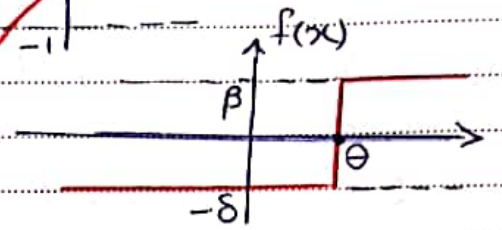
الف) تابع خطی

$$f(x) = a.x \quad \alpha > 0$$



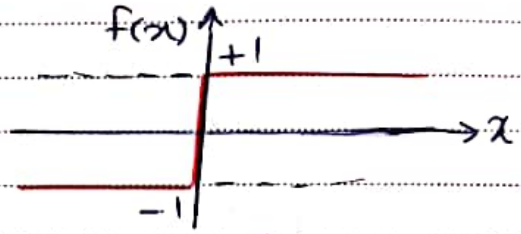
ب) تابع پله

$$f(x) = \begin{cases} \beta & x \geq \theta \\ -\delta & x < \theta \end{cases}$$



اگر $\beta = \delta = 1$ و $\theta = 0$ باشد تابع علامت ایجاد می شود:

$$f(x) = \begin{cases} 1 & x \geq 0 \\ -1 & x < 0 \end{cases} = sgn(x)$$



فرقش با تابع پله در این است که به جای صفر، ۱- داریم که بهتر عمل می کند.

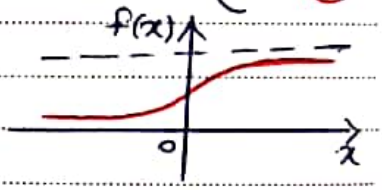
ج) تابع Sigmoid

$$f(x) = \frac{1}{1+e^{-\alpha x}} \qquad \alpha > 0 \quad \alpha \approx 1$$

به آن binary Sigmoid یا Logistic هم گفته می شود

$$f'(x) = \alpha f(x)\left[1 - f(x)\right]$$

مشتقش برحسب خودش قابل بیان



① ایراد دارد ← از یه جایی به بعد هم در قسمت مثبت هم منفی به یک حالت ثابتی می رسد. چون مشتق این تابع در این نواحی کوچک است و این مشتق هست که در آموزش کاربرد دارد و خیلی کوچک می شود و آموزش نمی تواند انجام بگیرد.

② ایراد دوم: نسبت به مبدا متقارن نیست.

د) تابع Bipolar Sigmoidal (دوقطبی) ← همان تابع قبلی است که scale شده و شیفت داده شده

$$f(x) = \frac{2}{1+e^{-\alpha x}} - 1 = \frac{1-e^{-\alpha x}}{1+e^{-\alpha x}} = \tanh\left(\frac{\alpha x}{2}\right)$$

$$f'(x) = \left(\frac{\alpha}{2}\right)\left[1 + f(x)\right]\left[1 - f(x)\right]$$

Scanned with CamScanner

جلسه چهارم - ۱۴۰۰/۷/۶

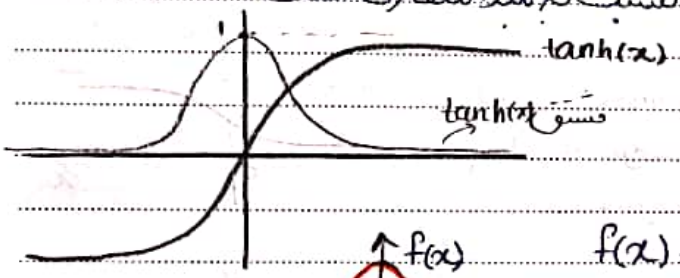
به طور کلی در توابع فعال سازی دو مقداری بهتر است که یک مقدار مثبت و یک مقدار منفی باشد تا اینکه یک مقدار مثبت و یکی صفر باشد. صفر محو لا مشکل سازی می شود.

ادامه توابع فعال سازی:

ه) tanh Units

$$\tanh(x) = \frac{1-e^{-2x}}{1+e^{-2x}} \qquad \tanh(x) = 2\sigma(2x) - 1$$

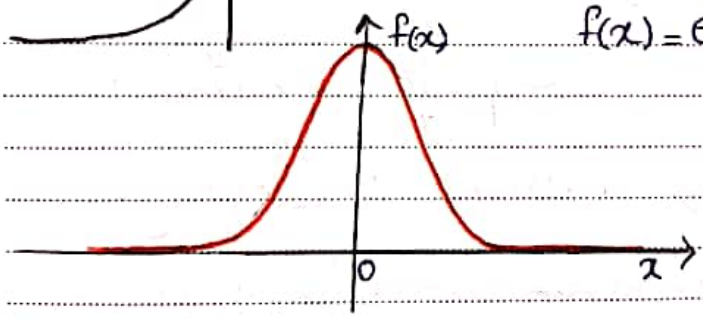
همان ویژگی های Bipolar Sigmoid را دارد، نسبت به مبدا متقارن است



و) تابع گاوسی

$$f(x) = e^{-x^2/\sigma^2}$$

مشتق تابع گوسی، باز هم گوسی می شود و از این جهت مطلوب است

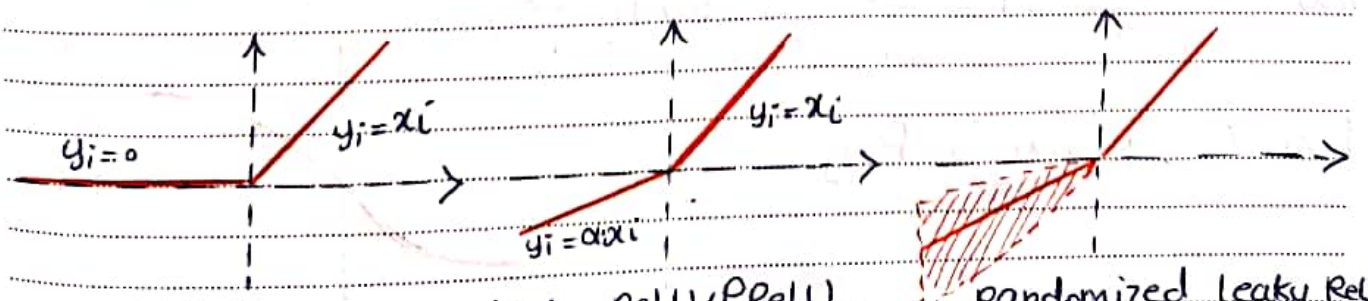


ز) ReLU

$$f(x) = \max(0, x)$$

$$f(x) = \max(x, \alpha x) \qquad \begin{cases} f(x) = \alpha x & x < 0 \\ f(x) = x & x \geq 0 \end{cases}$$

$$\begin{cases} f(x_{ji}) = x_{ji} & x_{ji} \geq 0 \\ f(x_{ji}) = \alpha_{ji} x_{ji} & x_{ji} < 0 \end{cases}$$



ReLU

Leaky ReLU / PReLU
پارامتری

Randomized Leaky ReLU
پارامتری تصادفی

Scanned with CamScanner

مزیت ReLU : مشکلی که tanh یا sigmoid داشت را کمتر دارد

هزینه محاسباتش کمتر است، همگرایی اش سریعتر است و پیاده سازی اش راحت تر است

مسئله ی به اشباع رفتن sigmoid ها در دو انتها که باعث می شد مشتق کوچک باشد و یادگیری کند شود را ندارد چون برای xها مثبت مشتق ۱ است

ایراد ReLU : بعضی واحدها انگار می میرند و آموزش x در مورد آن ها غیرفعال می شود.

علتش این است که برخی وزن ها به گونه ای می شوند که نورون هیچ وقت دیگر فعال نمی شود

این نتیجه ی جریان مشتق های بزرگ در شبکه حین آموزش است

به خصوص اگر ضریب نرخ یادگیری بالا باشد تا حدود ۴۰ درصد از نورون های شبکه ممکن است بمیرند به همین دلیل به سراغ leaky ReLU رفتند.

مردن نورون به این معناست وزن ها مقادیری پیدا می کنند که این نورون دیگر نمی تواند فعال شود

برای رفع این ایراد به سراغ Leaky ReLU رفتند که اجازه می دهد مقدار خروجی برای xهای منفی یک مقدار منفی کوچک پیدا کند (نشت کند)

← Randomized ReLU : گفتند شیب leaky ReLU را به صورت رندوم انتخاب کنیم

و این رندوم کردن اثر بهتری در آموزش دارد.

α را از یک توزیع یکنواخت (uniform) در فاصله ۰ و ۱ انتخاب می کنند. در فرآیند آموزش، α هر بار با نمونه برداری از چنین توزیعی انتخاب می شود و موقع تست تمام αهایی که در آموزش به کار رفته را میانگین گیری می کنند

و این میانگین را به صورت ثابت در زمان تست به کار می برند

$$\alpha_{ji} \sim U(l, u) \quad l < u \text{ and } l, u \in [0, 1)$$

رندوم کردن برای xهای مثبت صورت نمی گیرد چون می خواهیم مشتق ۱ باشد

Scanned with CamScanner

Softmax (Layer)

کاری که امروزه در شبکه های عمیق انجام می دهیم این است که یک لایه Softmax قرار می دهیم در واقع مثل این است که روی خروجی های نورون ها یک عمل دیگر هم انجام می دهیم، به عنوان لایه softmax به کار می برد (در واقع لایه نیست)

علتش این است که در مسائل دسته بندی چند کلاسه می خواهیم به جای اینکه برای کلاس ها مختلف اعداد مختلف داشته باشیم، خروجی ها اعداد بین ۰ و ۱ باشند که احتمال تعلق به آن کلاس را نشان دهند.

فرض کنید خروجی شبکه K بعدی است. Z ها خروجی اند و یک مقدار حقیقی تولید می کنند

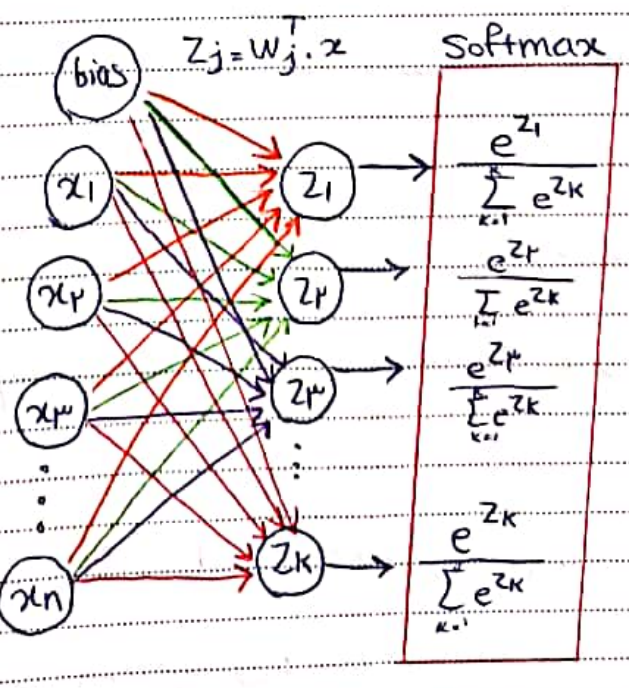
یک تابع $\sigma(Z)$ تولید می کنیم که این کار را انجام می دهد:

$$\sigma(Z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \qquad j=1,2,\dots,K$$

در واقع یک توزیع احتمال روی خروجی به ما می دهد

و احتمال تعلق به هر کلاس را نشان می دهد.

$$Z = \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_K \end{bmatrix} = \begin{bmatrix} w_1^T \\ w_2^T \\ w_3^T \\ \vdots \\ w_K^T \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$$



Scanned with CamScanner

دومین تغییر:

تغییر دیگری که در شبکه های عمیق می توانیم بدهیم:

برای لایه های مختلف توابع فعال سازی مختلفی تعریف کنیم

سومین تغییر:

نورون را غیرخطی کنیم. ترکیبات غیرخطی ورودی ها را وارد کنیم:

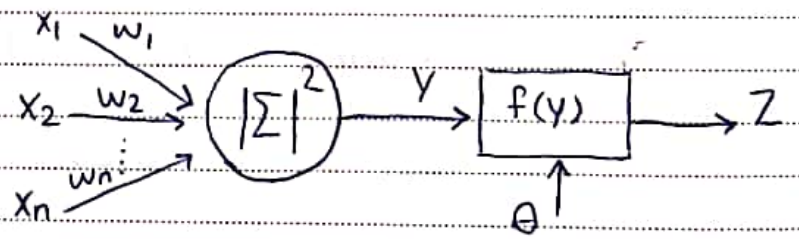
$$y = f\left(\sum_{i=0}^{n} w_i x_i + \sum_{k=1}^{n_i}\sum_{l=1}^{n_i} w_{kl} x_k x_l + \dots\right)$$

مثلا یک راه این است که جفت جفت x ها را اضافه کنیم (درجه ۲)

چهارمین تغییر: نورون آسنکرون شود

پنجمین تغییر: وزن ها اعداد حقیقی نباشند، اعداد مختلط باشند. در تئوری فقط قابل اجرا است.

$$w_i = a_i + j b_i$$



$$y = |w_1 x_1 + w_2 x_2 + \dots + w_n x_n|^2 = (a_1 x_1 + a_2 x_2 + \dots + a_n x_n)^2$$
$$+ (b_1 x_1 + b_2 x_2 + \dots + b_n x_n)^2$$

قدر مطلق به توان ۲ می شود قسمت حقیقی

به توان ۲ به اضافه ی قسمت موهومی به توان ۲

y می شود یک عبارت درجه ۲ از ورودی ها

$$Z = f(y) = \frac{1}{1+e^{-\alpha(y-\theta)}}$$

Scanned with CamScanner

تغییر ششم: فعال شدن نورون تصادفی باشد نه قطعی

I ← ورودی خالص به نورون (مجموع $x_i w_i$ ها) Y ← خروجی نورون

P(Y) ← احتمال فعال شدن نورون

$$p(I) = \frac{1}{1+e^{-I/T}}$$

T ← پارامتری که میزان randomness را کنترل می کند

$$Y = \begin{cases} +1 & \text{با احتمال } P(I) \\ -1 & \text{با احتمال } 1-P(I) \end{cases}$$

تغییر هفتم: گره های گراف، نورون های فازی باشند. به ۳ صورت قابل انجام است:

① ورودی ها اعداد فازی باشند

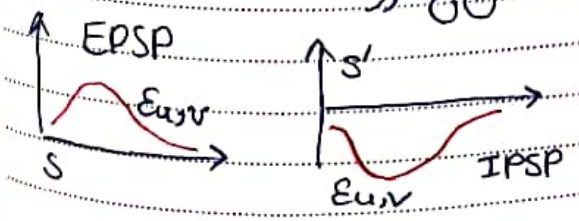
② پارامترهای شبکه اعداد فازی باشند (مثلا وزن ها)

③ هم ورودی ها و هم پارامترها فازی باشند

تغییر هشتم: خروجی نورون ها به جای ۰ و ۱، ضربه باشد. به آن ها نورون های ضربه ای یا spiking Neurons گفته می شود.

برای یک نورون پتانسیل در نظر می گیریم که آن را با P_V نشان می دهیم. از نورون طبیعی به خاطر داریم که P_V مجموع EPSP ها و IPSP هایی است که در آن لحظه به نورون می رسد.

اگر این پتانسیل (P_V) به مقدار آستانه θ برسد نورون فعال می شود



EPSP تاثیر کم کردن پتانسیل منفی را دارند

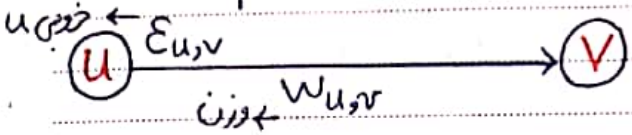
IPSP ها " زیاد "

" " "

روز بزرگداشت عطار نیشابوری

Scanned with CamScanner

فرض کنید در شبکه مان نورون های ضربه ای داریم و دو تا نورون u و v داریم که به شکل زیر



هم متصل است:

وقتی نورون u فعال می شود و شلیک می کند

تغییرات پتانسیل با زمان ظاهر می شود و این مقادیر در وزن $w_{u,v}$ ضرب می شوند و وارد می شوند

به نورون بعدی (v). معادله آن: $w_{u,v} \cdot \varepsilon_{u,v}(t-s)$

s ← زمان شلیک

چون در زمان s شلیک کرده، اثری که وارد نورون v می شود $(t-s)$ است

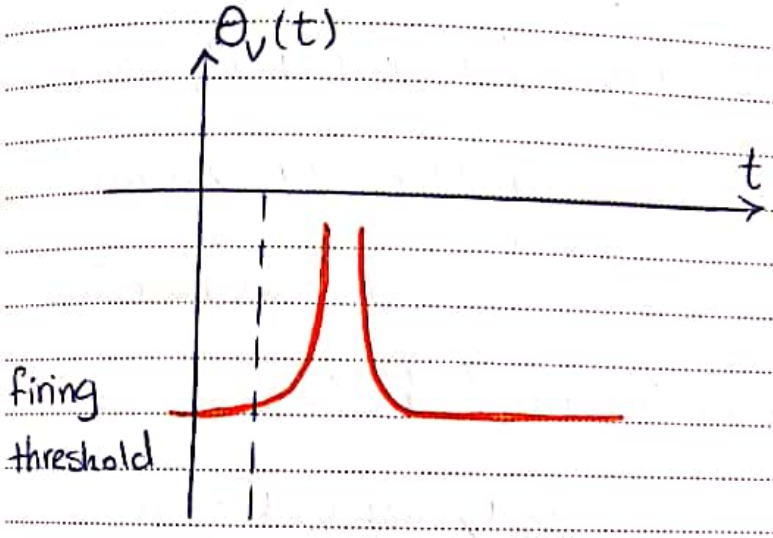
w می تواند ثابت یا متغیر باشد

پتانسیل استراحت نورون حدود $-70mv$ است اگر به $-50mv$ برسد نورون فعال می شود

پس مجموع اثرهایی که به یک نورون می رسد اگر پتانسیل را به $-50mv$ یا بالاتر برساند فعال می شود

Scanned with CamScanner

وقتی که نورون شلیک می کند (نورون طبیعی)، تا یک مدت کوتاهی بعد از آن نمی تواند فعال شود و نمی تواند به تحریک ورودی حتی پاسخ بدهد چون در حال طی کردن دوره گذار به وجود آوردن ضربه در خروجی می باشد.



مدل آن به این صورت است:

در این تابع وقتی نورون شلیک می شود به طور ناگهانی θ افزایش پیدا می کند یعنی مقدار آستانه به حدی بالا می رود که هیچ تحریکی نمی تواند از آن عبور کند به همین دلیل نورون در این بازه فعال نمی شود.

بعد از سپری شدن این زمان کوتاه، مجددا θ کاهش پیدا می کند و به مقدار قبلی اش برمی گردد بنابراین این تابع، ناتوان شدن نورون بعد از شلیک را مدل می کند.

$$P_v(t) = \sum_u \sum_{s \in F_u} w_{u,v} \cdot \varepsilon_{u,v}(t-s)$$

u ها تمام نورون هایی هستند که به نورون v وصل اند و خروجی شان وارد نورون v می شود

s ها زمان های شلیک نورون های u اند

مدلی که گفتیم، مدل نورون ضربه ای بود.

Scanned with CamScanner

دسته بندی نورون ها یا گره های گراف :

① نورون های نسل اول ← binary / bipolar ← یعنی خروجی شان یا ۰ و ۱ بود یا ۱- و ۱. مثال : پرسپترون ، شبکه Hopfield ، شبکه Boltzman

② نورون های نسل دوم ← نورون هایی که خروجی شان پیوسته است مثلا در شبکه های MLP ، RBF و RNN چنین نورون هایی داریم.

③ نورون های نسل سوم ← نورون های ضربه ای ، مثلا در شبکه های feed forward ضربه ای یا در شبکه های RBF ضربه ای . پیچیدگی بالایی دارند . عملکردشان بهتر از نورون های غیر ضربه ای است

اگر فرض کنیم تغییرات خروجی نورون به طور پیوسته صورت می گیرد می توانیم آن را با یک معادله دیفرانسیل مرتبه اول مدل کنیم :

$$\frac{dy}{dt} = -g(y) + \sum_{j=0}^{n} w_j . x_j = -g(y) + I \qquad \boxed{I = \sum_{j=0}^{n} w_j . x_j}$$

Net Input

می خواهیم نشان بدهیم این مدل با مدل استاتیک که قبلا هم داشتیم هماهنگ است :

در حالت تعادل : $dy/dt = 0$ چون خروجی ثابت است $\Rightarrow g(y) = I$

$$\Rightarrow \boxed{y = g^{-1}(I) = g^{-1}\left(\sum_{j=0}^{n} w_j . x_j\right) = f\left(\sum_{j=0}^{n} w_j . x_j\right)}$$

$$\Rightarrow \boxed{y = f\left(\sum_{j=0}^{n} w_j . x_j\right)}$$ همان رابطه ای است که برای نورون استاتیک داشتیم

به عنوان مثال اگر $f(x) = \tanh(\beta . x)$ آنگاه g که معکوس f است می شود :

$$g(x) = -\left(\frac{1}{2}\beta\right)\left[\ln(1-x)/(1+x)\right]$$

Scanned with CamScanner

Network Connections (links)

اتصالات شبکه

گفتیم که یال ها جهت و وزن دارند. این وزن ها قرار است در یک فاز یادگیری یا آموزش، یاد گرفته شوند. تنها مجهولات ما در شبکه همین وزن ها هستند.

برای نوع داده ای وزن ها هیچ محدودیتی نداریم (حقیقی، مختلط و ...)

وزن مثبت ← تحریک $w>0$
وزن منفی ← ممانعت $w<0$
وزن صفر ← اتصالی وجود ندارد $w=0$

در مورد نوع داده ای وزن ها خودمان با توجه به مسئله باید تصمیم بگیریم

ارائه (بازنمایی) وزن ها:

در انتخاب درایه ها برای ماتریس وزن دو تا قرارداد وجود دارد:

① w_{ij} وزن اتصال نورون i به نورون j را نشان دهد $i \xrightarrow{w_{ij}} j$

② w_{ij} یعنی وزنی که به مقصد j می آید از مبدأ i (جهتش برعکس قرارداد اول است) $j \xrightarrow{w_{ij}} i$

روش بیشتر به کار رفته چون بعضی از روابط ریاضی آسان تر می شود

ماتریس وزن در این دو حالت Transpose همدیگر است.

نوع اتصالات:

① full connection (تمام اتصال): خروجی هر نورون به تمام نورون های لایه بعد داده می شود

② uniformly Random Connection

تعدادی از خروجی های این لایه به صورت تصادفی یکنواخت انتخاب شوند و به هر نورون در لایه بعد وصل شود (برای هر نورون خروجی هایی که وارد آن می شود متفاوت خواهد بود و به صورت اتفاقی یکنواخت از خروجی لایه قبل انتخاب می شود، تعداد ورودی های نورون ها یکسان است)

Scanned with CamScanner

③ اتصال یک به یک ← خروجی هر نورون فقط به یک نورون در لایه بعد متصل می‌شود

Network Architectures ساختار (معماری) شبکه

① Feed forward (جلورو)

سیگنال همیشه رو به جلو می‌رود و برنمی‌گردد. پس گراف شبکه مسیر برگشتی و مسیر بسته ندارد

این شبکه‌ها، شبکه‌های استاتیک هستند. وقتی یک ورودی به آن می‌دهیم یک خروجی برای ما تولید می‌کند

خروجی تولید شده به ازای این ورودی وابسته به حالت قبلی نیست، به عبارت دیگر شبکه حافظه ندارد

② Recurrent (بازخوردی)

برخلاف شبکه FF در گراف شبکه مسیر بسته وجود دارد. می‌گوییم شبکه دینامیک است.

وقتی یک ورودی اعمال می‌شود خروجی شبکه ممکن است مرتب عوض شود. علت این تغییرات همان فیدبک‌هاست مثلا یک سیگنال تا یک جایی جلو می‌رود و بعد برمی‌گردد به ورودی نورون‌ها وارد می‌شود و ممکن است خروجی این‌ها را عوض کند و وقتی خروجی‌ها عوض شد اثر آن مجددا در شبکه جلو می‌رود و خروجی شبکه را عوض می‌کند. بنابراین این شبکه‌ها حالتی دارند که خروجی آنقدر تغییر می‌کند تا جایی که ثابت شود و این مقدار ثابت می‌شود پاسخ شبکه به ورودی مورد نظر.

این شبکه‌ها این خطر را هم دارند که خروجی هیچ‌گاه ثابت نشود و ناپایدار باشند، به طور کلی در تئوری سیستم‌ها وجود فیدبک می‌تواند باعث ناپایداری شود

Scanned with CamScanner

جلسه پنجم - ۱۴۰۰/۷/۱۱

ساختار شبکه باید تعیین شود و پارامترهای آن نیز باید مشخص شود. پارامترها در یک فاز یادگیری در اثر تعامل با محیط (در اثر تعامل با داده‌هایی که از محیط گرفته شده) یادگرفته می‌شوند

انواع یادگیری (Learning paradigms):

① supervised Learning (یادگیری با نظارت (با معلم))

در زمان آموزش، هر ورودی که به شبکه داده می‌شود، خروجی متناظرش هم داده می‌شود.

② unsupervised Learning (یادگیری بی نظارت)

در فاز آموزش، داده‌های ورودی را به شبکه می‌دهیم اما خروجی را برای آن مشخص نمی‌کنیم

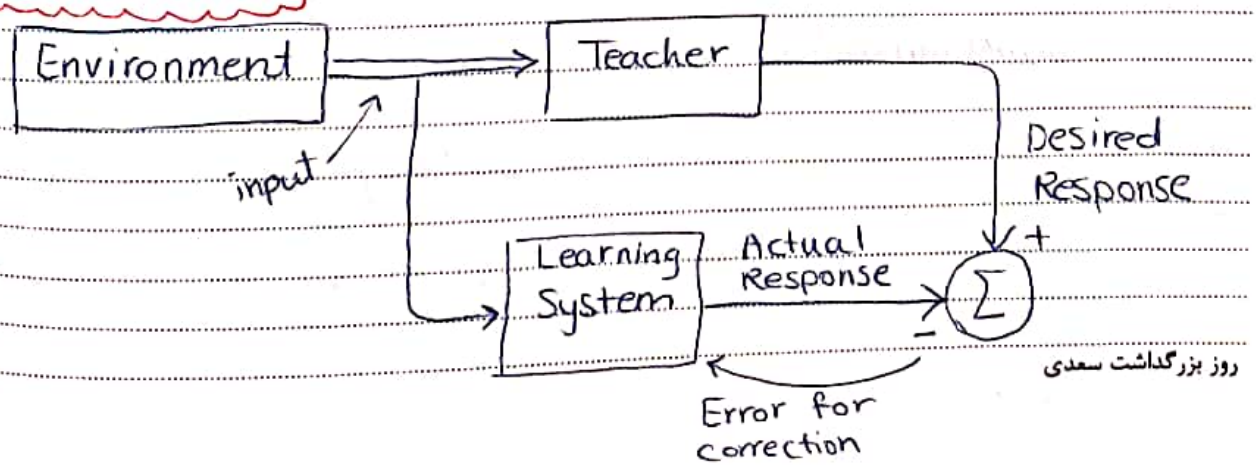
فقط داده‌های ورودی را می‌دهیم و شبکه بایستی خودش یاد بگیرد که چه خروجی ای به ما بدهد

Graded Learning - Learning with a critic

③ Reinforcement Learning (یادگیری تقویتی)

بین دو حالت بالا است. به ازای ورودی‌ها، خروجی را مشخص نمی‌کنیم برای سیستم اما یک سیگنالی می‌دهیم که سیستم را راهنمایی می‌کند. در واقع یک فیدبکی از جانب محیط به سیستم داده می‌شود تا بفهمد خوب عمل کرده یا نه، که می‌تواند همراه با تاخیر هم باشد.

Supervised Learning:



روز بزرگداشت سعدی

Scanned with CamScanner

supervised Learning

یک دیتا داریم که اطلاعات از آن دریافت می‌شود و می‌شود input برای سیستم ما. این اطلاعات (ورودی‌ها) به معلم (Teacher) هم داده می‌شود. معلم مشخص می‌کند خروجی مطلوبی که این شبکه باید داشته باشد چه چیزی است. این خروجی مطلوب با پاسخی که سیستم در ازای این ورودی تولید کرده مقایسه می‌شود و یک خطایی به دست می‌آید.

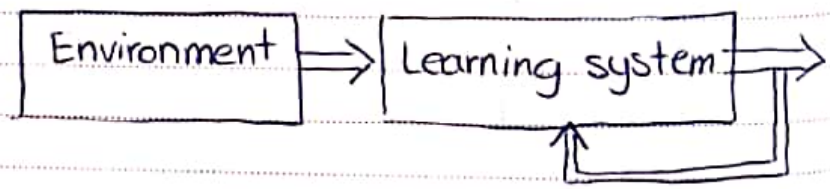
این خطا به یادگیری سیستم ما کمک می‌کند. سیستم سعی می‌کند به گونه‌ای عمل کند که این خطا را به حداقل برساند.

معمولا در این نوع آموزش یک جفت ورودی و خروجی مطلوب داریم. یعنی هرگاه یک ورودی به سیستم می‌دهیم، به ازای آن ورودی، خروجی مطلوب را هم می‌دهیم.

برای اینکه فرآیند آموزش صورت بگیرد، لازم است که سیستم کار آموزش را خیلی تکرار کند.

مثال‌ها: perceptron و LMS, BP

unsupervised Learning



داده‌ها از محیط به سیستم داده می‌شود و سیستم باید خودش یاد بگیرد. مثلا سیستم ممکن است یاد بگیرد ویژگی‌هایی را از داده‌ها استخراج کند. یعنی یادگرفتن ویژگی‌های داده‌ها: ما داده‌هایی را به سیستم می‌دهیم و سیستم در خروجی ویژگی‌هایی از داده‌ها را به ما می‌دهد. یا مثلا در داده‌ها نظمی پیدا می‌کند یا الگویی پیدا می‌کند. وابستگی‌هایشان را پیدا می‌کند. خوشه‌هایی که در داده‌ها هست را پیدا کند و...

(گاهی اوقات برای استخراج ویژگی‌های پایان از شبکه‌های بی‌نظارت استفاده می‌کنن)

Scanned with CamScanner

ممکنه سیستم یادبگیره که میزان شباهت ورودی به یک الگوی خاص را که ارش کند

یا ممکنه چندین الگو رو در نظر بگیره و میزان شباهت ورودی به هر کدام را در خروجی نشان دهد

ممکنه خوشه بندی کنه داده ها رو.

ممکنه یه نقشه دو بعدی از داده های که ابعاد بالا دارند تولید کند و این نقشه معنی دار باشد. فضای

n بعدی را به فضای ۲ بعدی نگاشت کند

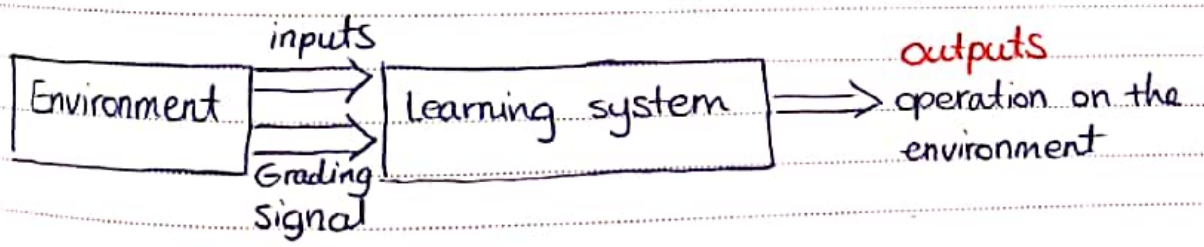
یادگیری تقویتی (Reinforcement Learning)

سیستم یادگیرنده از محیط اطلاعاتی را می گیرد هم چنین یک ورودی دیگر هم دارد که این ورودی

یک امتیاز یا ارزیابی است که راجع به کار خودش دریافت می کند.

این سیستم در ازای هر ورودی که دریافت می کند یک عمل انتخاب می کند که باید روی محیط انجام شود

در واقع سیستم یادگیری می خواهد یاد بگیرد که عملی روی این محیط انجام دهد.



عملی که سیستم روی محیط انجام می دهد، روی محیط اثر می گذارد و منجر به این می شود که یک سری

داده های جدیدی از محیط تولید شود و به سیستم یادگیرنده داده شود و در عین حال یک ارزیابی هم

از عمل تعیین شده توسط سیستم مشخص می شود و به سیستم کمک می کند که بتواند عمل های بهتری

را انتخاب کند

از آن جایی که برای سیستم مشخص نمی کنیم که به ازای هر ورودی، خروجی چه چیزی باید باشد، قاعدتا ضعیف تر

از supervised Learning باید کار بکند.

این نوع یادگیری معمولا برای ربات ها استفاده می شود.

Scanned with CamScanner

مسائل دسته‌بندی (classification problems)

← دسته‌بندی یک راه تشخیص یا شناسایی است. مثلا برای شناخت شیء x می‌توانیم بررسی کنیم به کدام دسته تعلق دارد. در انسان خیلی از کارهای شناختی با دسته‌بندی صورت می‌گیرد.

← یک دسته‌بندی کننده (classifier) یاد می‌گیرد که نمونه‌هایی را از دسته‌های مختلف ببیند و یاد بگیرد که نمونه‌های جدید به کدام دسته تعلق دارد.

داده‌هایی که به دسته‌بندی کننده داده می‌شود ممکن است داده‌های خام باشند یا ویژگی‌هایی باشد که از داده‌های خام استخراج شده است.

دسته‌بندی معادل بخش‌بندی کردن فضای ورودی است. مثل این است که فضای ورودی را مثل یک صفحه سفید در نظر بگیریم و در آن خطوطی را رسم کنیم و ناحیه‌هایی را مشخص کنیم و هر کدام از این ناحیه‌ها، فضایی را که متعلق به یک دسته خاص هستند را مشخص کند.

مسئله دسته‌بندی: فرض کنیم مجموعه‌ای از نمونه‌ها داریم که با بردارهای x نشان می‌دهیم و این‌ها از k تا دسته هستند. هدف این است که دسته‌بندی کننده‌ای طراحی کنیم که با استفاده از داده‌های x یاد بگیرد که چگونه دسته‌بندی نمونه‌ها را به این k تا کلاس انجام بدهد

فرض کنیم مسئله ۲ دسته‌ای باشد. تابعی به نام تابع تمایز خطی در نظر می‌گیریم:

$$y(x) = w^t x + w_0$$

اگر $y(x) \geq 0$ باشد x به دسته‌ی C_1 و اگر $y(x) < 0$ باشد x به دسته‌ی C_2 تعلق دارد.

فرض کنیم فضای ورودی ۲ بعدی باشد (برای اینکه در صفحه به راحتی نشان بدهیم)

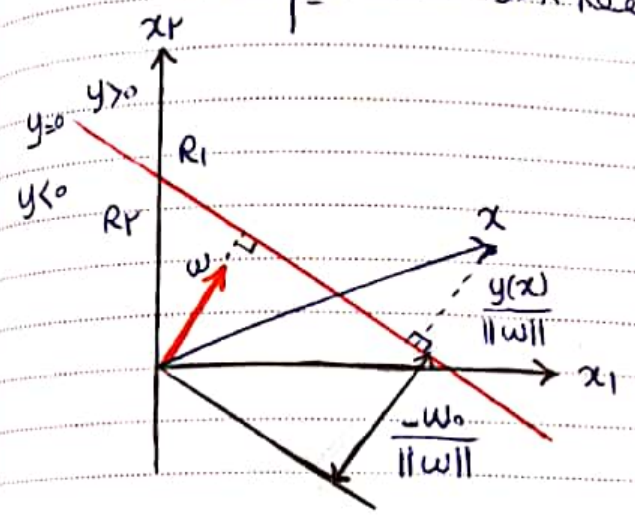
در این صورت مرز تصمیم (decision boundry) می‌شود: $y(x) = 0$

در واقع w عمود بر خط مرز تصمیم است.

Scanned with CamScanner

$\frac{-w_0}{\|w\|}$ ← مشخص می‌شود w_0 اینکه فاصله این خط تا مبدأ چقدر باشد توسط

و یک بخش علامتدار از فاصله‌ای است که نقطه x از خط مرز تصمیم دارد



در حالت K دسته‌ای، K تا تابع داریم:

$$y_k(X) = w_k^T X + w_{k0} \qquad k=0,1,\dots,K$$

x را متعلق به کلاس k می‌دانیم اگر: $y_k(X) > y_j(X)$ for all $j \neq k$

یعنی x ای که دارید را در K تا معادله می‌گذارید و معادله‌ای که بزرگترین مقدار y را می‌دهد، دسته را تعیین می‌کند

مرز تصمیم بین دو کلاس C_k و C_j دارای معادله روبه‌رو است: $y_k(X) = y_j(X)$

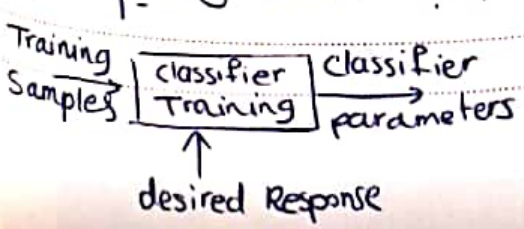
معادله مرز بین دسته k و j: $(w_k - w_j)^t X + (w_{k0} - w_{j0}) = 0$

۲ فاز دسته‌بندی:

① Classifier Training (Learning) (آموزش دسته‌بندی کننده)

یعنی همان یادگیری پارامترها، اینکه خط‌ها چگونه رسم شوند در صفحه و تقسیم‌بندی فضا چگونه صورت بگیرد.

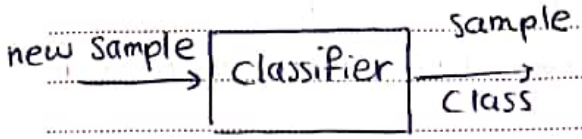
در این فاز نمونه‌های آموزشی را به همراه مقادیر یا پاسخ‌های مطلوب به classifier می‌دهیم و در این فاز یادگیری پارامترهای classifier مشخص می‌شود.



Scanned with CamScanner

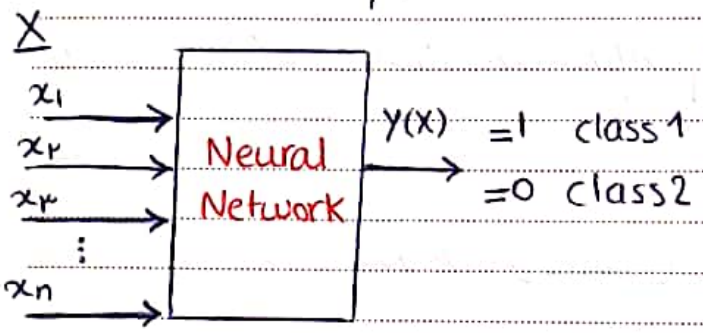
② classifier Testing (use)

② فاز تست (فاز استفاده از دسته‌بندی کننده)



یک نمونه به classifier می‌دهیم و در خروجی دسته‌ی نمونه را برای ما مشخص می‌کند

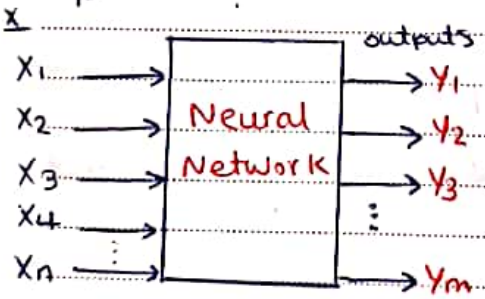
اگر بخواهیم عمل دسته‌بندی را با استفاده از یک شبکه عصبی انجام بدهیم:



فرض کنید داده ورودی n بعدی باشد و دسته‌بندی ۲ کلاسه می‌خواهیم انجام بدهیم

دو دسته را با یک خط خروجی می‌توان نشان داد. اگر بود کلاس ۱ و اگر نه بود کلاس ۲

اگر m تا دسته داشته باشیم: ① یک راه این است که به ازای هر دسته ۱ خروجی قرار بدهیم و



در این صورت m تا خط خروجی خواهیم داشت

در این صورت اگر ورودی به کلاس i ام تعلق داشته باشد خط خروجی مربوط به آن فعال می‌شود

② یک راه دیگر این است که از coding استفاده کنیم. برای نشان دادن m تا خروجی، $\log_2 m$ خط کافی است. مثلا اگر ۸ تا دسته داشته باشیم یک کد ۳ بیتی کافی است و با ۳ خط می‌توانیم نشان بدهیم.

معمولا از حالت اول استفاده می‌کنیم و به ازای هر خروجی یک خط می‌گذاریم چون ساده‌تر است.

Scanned with CamScanner

۱۴۰۰

مثال: می‌خواهیم سیستمی بسازیم که هر فردی به آن عرضه می‌کنیم مشخص کند که آیا یک بسکتبالیست است یا وزنه بردار. فرض هم بر این است که هر فردی که به این سیستم می‌دهیم حتما یا وزنه بردار است یا بسکتبالیست. (دسته‌بندی دو کلاسه)

راه حل:

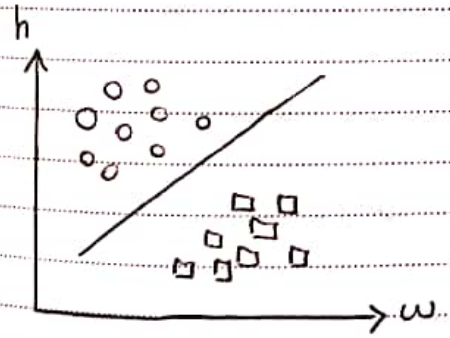
چه ویژگی‌هایی هست که این دو دسته را از هم متمایز می‌کند؟ انتخاب ویژگی‌ها یک مسئله مهم است.

در این مسئله دو ویژگی قد و دور کمر را در نظر می‌گیریم: $X = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$

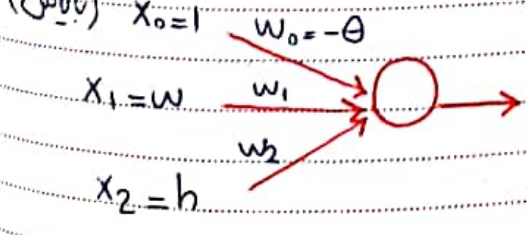
چون این مسئله دو کلاسه است فضای ورودی باید به ۲ قسمت تقسیم شود.

فاز آموزش شبکه به این صورت است: تعداد زیادی نمونه از وزنه بردارها و بسکتبالیست‌ها نیاز داریم و باید دو فیچر مورد نظرمان را در مورد هر کدام اندازه‌گیری کنیم.

اگر داده‌ها را رسم کنیم چنین شکلی داریم:



اگر یک نورون داشته باشیم و بخواهیم مسئله را حل کنیم:



$$y = f(I) : \begin{cases} 1 & I \geq 0 \quad \text{class 1} \\ -1 & I < 0 \quad \text{class 2} \end{cases}$$

$$y = f(I) = f\left(\sum_{i=0}^{2} w_i x_i\right)$$

$$X = \begin{bmatrix} 1 \\ x_1 \\ x_2 \end{bmatrix} \qquad W = \begin{bmatrix} -\theta \\ w_1 \\ w_2 \end{bmatrix} \Rightarrow I = -\theta + w_1 x_1 + w_2 x_2$$

Net Input

ولادت امام حسن مجتبی (ع) (۳ هـ ق) و روز اکرام و تکریم خیرین

Scanned with CamScanner

برای یافتن مرز تصمیم $I=0$ قرار می‌دهیم:

$$w_1x_1 + w_2x_2 - \theta = 0 \Rightarrow x_2 = -\frac{w_1}{w_2}x_1 + \frac{\theta}{w_2}$$ ← معادله خط مرز تصمیم

اگر این خط را بکشیم انتظار داریم که سکتالیست‌ها یک طرف و وزنه‌بردارها طرف دیگر آن قرار گیرند

اگر مسئله به گونه‌ای باشد که پس از یافتن خط مرز تصمیم، همه‌ی داده‌های (داده‌های آموزشی) دسته اول یک طرف خط و همه‌ی نمونه‌های دسته‌ی دوم طرف دیگر خط قرار بگیرند، می‌گوییم این مسئله جدایی‌پذیر خطی است

در غیر این صورت جدایی‌پذیر خطی نیست

Minsky نشان داد که پرسپترون فقط می‌تواند یک خط بکشد و فقط می‌تواند مسائل جدایی‌پذیر خطی را حل کند. بنابراین پرسپترون بسیاری از مسائل را نمی‌تواند حل کند.

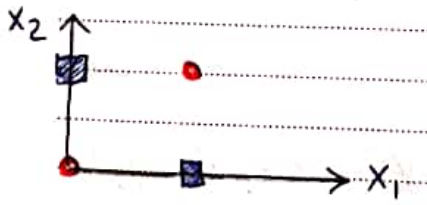
مسئله XOR

روز شوراها

اردیبهشت ۱۰ 30 Apr
جمعه ۱۷ رمضان

مثلاً مسئله‌ی ساده‌ای مثل XOR جدایی‌پذیر خطی نیست

x_1x_2	output	class
00	0	1
11	0	1
01	1	2
10	1	2



این مسئله ماهیتاً linearly separable نیست یعنی نمی‌توان هیچ خطی در صفحه رسم کرد که داده‌های ۲ کلاس را از هم جدا کند.

مسئله XOR با یک perceptron (پرسپترون) قابل حل نیست.

روز ملی خلیج فارس

Scanned with CamScanner

Single Layer FeedForward Neural Networks (شبکه‌های جلوروی تک لایه)

① واحد پردازشگر Hebb (Hebb processing Unit)

یادگیری هم وقوعی coincidence learning:
- اگر دو نورون هم زمان فعال باشند اتصال بین آن‌ها باید تقویت شود
- اگر دو نورون هم زمان خاموش باشند اتصال بین شان باید تقویت شود

الگوریتم Hebb:

۱) مقداردهی اولیه وزن‌ها $w_i = 0 \quad (i=1 \text{ to } n)$

۲) فرض کنیم یک نمونه آموزشی (S, t) داریم، S ورودی و t خروجی مطلوب متناظر با آن است.

برای هر نمونه ورودی آموزشی مثل (S, t) این دو مرحله را انجام بده:

۳) $x_i = S_i \quad i=1,2,\dots,n$

$y = t$

۴) مقدار وزن‌ها را اصلاح کن با این قاعده:

$$w_i^{new} = w_i^{old} + x_i y \qquad i=0,1,2,\dots,n$$

۵) پایان

سوال: قاعده‌ی به هنگام شدن وزن‌ها چه ارتباطی با قانون Hebb دارد؟

با آمدن هر نمونه، جمله‌ی $x_i y$ به وزن قبلی اضافه می‌شود، اگر $x_i y$ یک عدد مثبت باشد وزن مورد نظر افزایش پیدا می‌کند و تقویت می‌شود. $x_i y$ در صورتی مثبت است که نه x_i و نه y صفر نباشند. x_i و y یا ۰ اند یا ۱ اند. اگر هر کدام‌شان ۰ باشد وزن تقویت نمی‌شود اما اگر هر دو یک باشند تقویت می‌شود. چون مانند شکل روبه رو،

$$① \xrightarrow{x_i} ② \xrightarrow{y}$$

ورودی و خروجی نورون دوم ۱ است. خروجی نورون اول هم ۱ است پس وزن بینشان باید تقویت شود

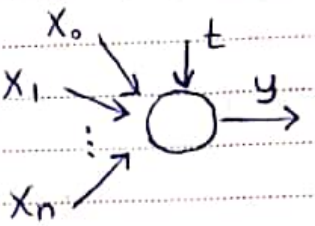
نکته: در قاعده Hebb نورون دو حالته است. خروجی اش ۰ و ۱ است.

Scanned with CamScanner

جلسه ششم - ۱۴۰۰/۷/۱۸

قانون آموزش perceptron:

$$\boxed{w^{new} = w^{old} + (t-y)X}$$



مقدار مطلوب در فاز آموزش از طریق ورودی t به نورون گفته می‌شود.

Roseblatt ثابت کرد که این الگوریتم از هر مقدار اولیه‌ی w^{old} که شروع بکنیم در یک تعداد محدود iteration همگرا می‌شود و به جواب می‌رسد. در نتیجه این قانون شد قانون یادگیری پرسپترون و این اولین مدل نورونی بود که به طور جدی معرفی شد.

در این مدل نورون n تا ورودی می‌گیرد ← $X_1, \dots, X_n$ و یک ورودی bias ← X_0

و یک ورودی t هم دارد.

در ادامه Minsky پیشنهاد کرد که یک ضریب α در جمله $(t-y)X$ ضرب شود که $\alpha < 1$ و در واقع هر بار اصلاح با مقدار کمتری صورت بگیرد و به همگرایی سریعتر کمک کند

$$w^{new} = w^{old} + \alpha(t-y)X$$

و پیشنهاد داد که مقدار α این باشد:

$$\alpha = \frac{X \cdot W}{\|X\|^2} \qquad \alpha = \frac{1}{\|X\|}$$

در ادامه Rosenblatt در دانشگاه Cornell یک سیستم درست کرد که اسمش رو Mark I perceptron گذاشت.

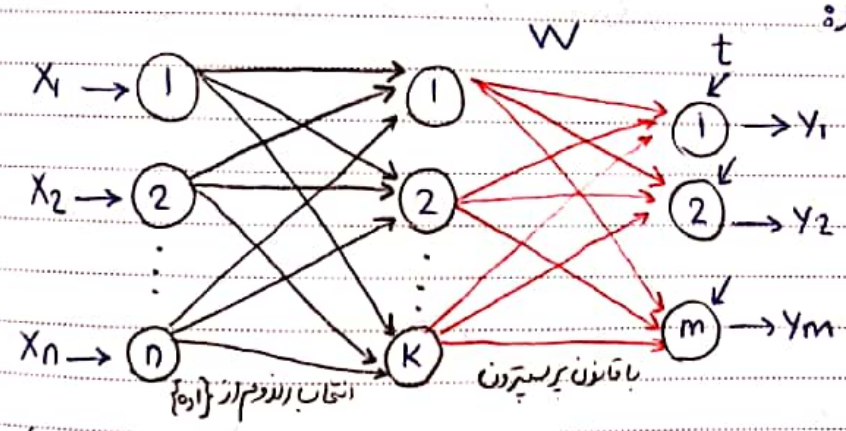
می‌خواست شبکیه چشم انسان را شبیه‌سازی کند با یک نورون. می‌خواست تصویر حروف را روی آرایه‌ای از سنسورها بیندازد و می‌خواست از روی این تصویر نورون تشخیص بدهد که چه حرفی بوده است.

در آن زمان همه چیز هنوز الکتریکی بود، الکترونیک و دیجیتال اصلا پیشرفتی نکرده بود. خودش سنسوری درست کرد که بتونه تصویر حرف رو sense بکنه.

Scanned with CamScanner

آرایه‌ای از سلول‌های حساس به نور را ایجاد کرد که 20*20 بود یعنی ۴۰۰ تا سلول.
۴۰۰ تا سنسور را در کنار هم قرار داد و ۴۰۰ تا سیگنالی که این‌ها دریافت می‌کردند را به سیستمش می‌داد
تا سیستم بتواند حرف مورد نظر را تشخیص بدهد. برای این تشخیص لازم بود تا شبکه‌ای از پرسپترون‌ها
ایجاد شود و با یک پرسپترون این عمل ممکن نبود. یک شبکه ۳ لایه از پرسپترون‌ها ایجاد کرد.
مقدار هر وزن را با یک مقاومت نشان می‌داد و این وزن‌ها را تغییر می‌داد. به این صورت که پتانسیومتر
قرار داده بود و شفت آن را به شفت یک موتور متصل کرده بود. وقتی موتور تحریک می‌شد خود آن می‌چرخید
و این چرخش شفت پتانسیومتر را می‌چرخاند و نتیجه این می‌شد که مقاومت تغییر میکرد. بدین ترتیب این
سیستم می‌توانست به صورت سخت‌افزاری یاد بگیرد که این مقدار مقاومت چقدر باید باشد.

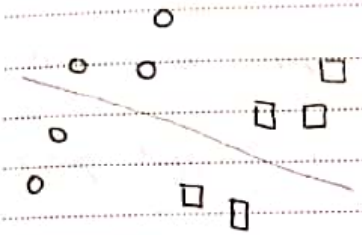
ساختار شبکه‌ای که ایجاد کرده بود:



در لایه دوم K تا نورون و در لایه سوم m تا نورون قرار دارد. m تعداد حروفی است که می‌خواهد شناسایی کند.
در نتیجه m تا خروجی داریم. نورون‌های لایه اول فقط وظیفه‌شان این است که ۴۰۰ تا ورودی را به لایه بعد توزیع
کنند. نورون‌های لایه دوم روی ورودی عمل می‌کنند و نتیجه‌شان به لایه بعد داده می‌شود. لایه آخر هم باز روشن به
عملی انجام میده و خروجی ایجاد می‌شه. برای پرسپترون‌های لایه آخر مقادیر t را داریم چون می‌دانیم وقتی یک ورودی
داده می‌شود خروجی درست چه چیزی باید باشد پس لایه آخر را می‌توان با قانون پرسپترون آموزش داد. اما در مورد
لایه وسط مقدار t را نداریم پس نمی‌توان با قانون پرسپترون آموزش داد. به همین دلیل گفت که من وزن‌های این
لایه را به صورت رندوم از ۰ و ۱ انتخاب می‌کنم. پس در لایه وسط یادگیری انجام نمی‌شود.

Scanned with CamScanner

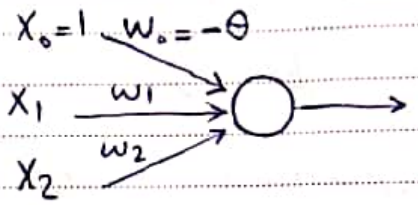
دسته‌بندی با یک پرسپترون



داده‌ها ۲ دسته‌اند ← دایره و مربع

فضای ورودی ۲ بُعدی است، هر نقطه با ۲ عدد قابل نمایش است

پس نورون به این صورت است:



Net Input: $I=-\theta+w_1x_1+w_2x_2$

می‌خواهیم این نورون را آموزش بدهیم پس از قانون یادگیری پرسپترون استفاده می‌کنیم:

$$w^{new}=w^{old}+(t-y)x$$

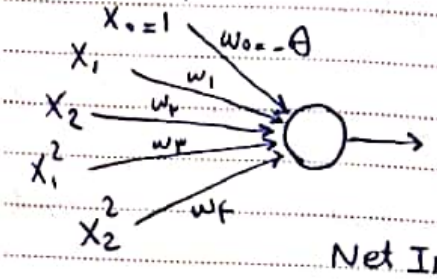
بار اول w^{old} را نداریم به همین دلیل خودمان یک مقداری برایش در نظر می‌گیریم. در نتیجه w_0 و w_1 و w_2 را داریم در شروع کار. پس معادله مرز ($I=0$) را داریم که یک خط است و می‌توان آن را رسم کرد.

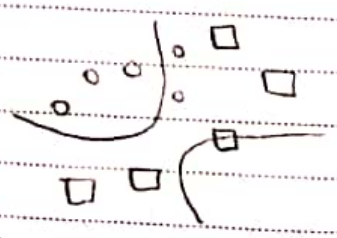
در ادامه یکی یکی ورودی‌های مختلف را اعمال می‌کنیم و چک می‌کنیم آیا مرز تغییر کرده یا نه. وقتی که یک بار همه‌ی ورودی‌ها را اعمال کنیم و مرز تغییری نکند یعنی همگرا شده است. (مرز تغییر نکند یعنی وزن‌ها تغییری نکرده)

دسته‌بندی با پرسپترون مرتبه دوم

داده‌های این مثال به صورت خطی جدایی پذیر نیستند و این مسئله با یک نورون خطی قابل حل نیست

می‌توانیم یک نورون درجه ۲ در نظر بگیریم. به این معنا که علاوه بر ورودی‌ها، درجه دوم آن‌ها را هم به عنوان ورودی بدهیم.





Net Input: $I=-\theta+w_1x_1+w_2x_2+w_3x_1^2+w_4x_2^2$

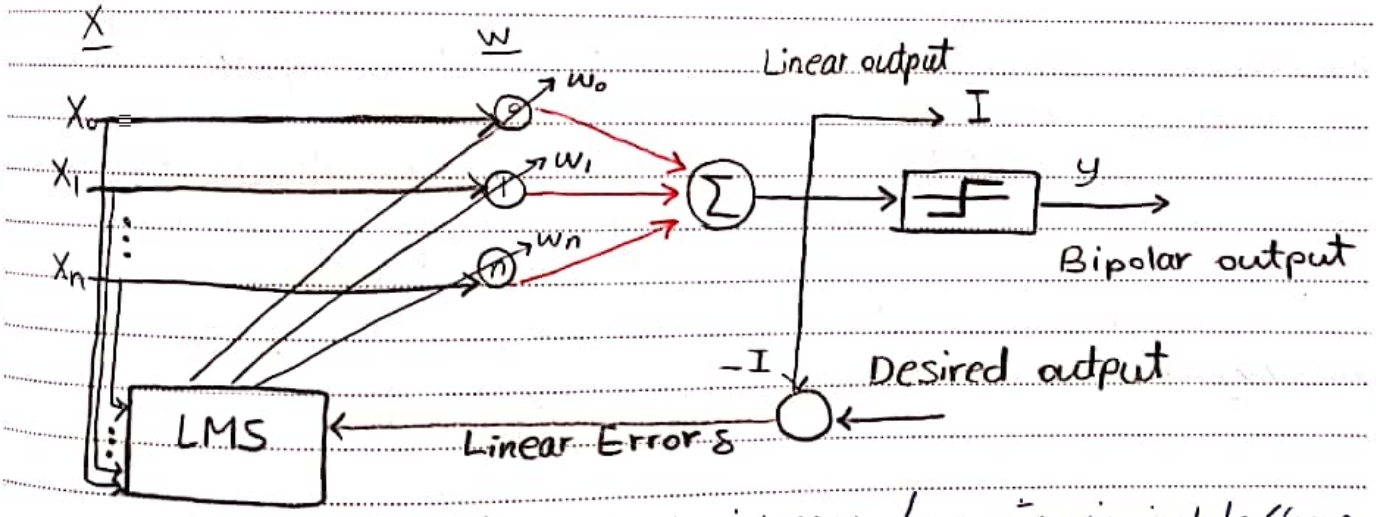
Scanned with CamScanner

مجدداً یک مقدار اولیه برای w^{old} در نظر می‌گیریم و در ادامه ورودی‌های مختلف را اعمال می‌کنیم و تغییرات مدرز را بررسی می‌کنیم تا زمانی که همگرا شود.

نورون غیرخطی قدرت بیشتری دارد و می‌تواند مسائلی که جدایی‌پذیر خطی نیستند را حل کند اما در عوض هزینه‌ای دارد. در نورون خطی ۳ تا پارامتر بود که باید یادگرفته می‌شد اما در این نورون مرتبه‌دوم تعداد پارامترها ۵ تاست بنابراین آموزش هزینه بیشتری دارد. اگر شبکه‌ای از این نورون‌ها داشته باشیم تفاوت زیادی با شبکه‌ای از نورون‌های خطی دارد. (تعداد وزن‌ها یکباره خیلی می‌رود بالا و بادیادگیری سنگین‌تری می‌شود)

The Adaline processing Unit

Adaptive Linear Element → Adaline



ورودی‌ها وارد نورون می‌شود، در یک سری وزن‌های ضرب می‌شود، نتایج باهم جمع می‌شوند و اسم خروجی را I می‌گذارد که Linear output است. I را به یک تابع علامت می‌دهد و خروجی را Bipolar output می‌نامد و با y نشان می‌دهد که یا ۱ است یا ۱-. خروجی خطی را با مقدار مطلوب خروجی مقایسه می‌کند و تفاضل آن‌ها می‌شود خطای خطی (δ). این خطا را به همراه X_iها به الگوریتم LMS می‌دهد و تعیین می‌کند که وزن‌ها چگونه باید تغییر کند

Scanned with CamScanner

رابطه ریاضی برای AdaLine :

$$I = w_0 x_0 + w_1 x_1 + \dots + w_n x_n = X.W = X^T W = W^T X$$

$$y = sgn(I)$$

در دسته‌ی performance Learning از قوانین یادگیری قرار می‌گیرد. براساس عملکرد کار می‌کند و خطا را حداقل می‌کند

در این مدل، وزن‌ها توسط قانون LMS تعیین می‌شوند. گاهی از این قانون با عنوان LMSE (Least Mean Square Error) نیز یاد می‌شود.

قاعده LMS ← Least Mean Square ← وزن‌ها به مقدار بهینه همگرا می‌شوند، فارغ از مقدار اولیه وزن‌ها را به گونه‌ای تغییر می‌دهد که متوسط مربع خطا، حداقل شود

قاعده یادگیری AdaLine :

(N تعداد نمونه‌ها)

تابع خطا: $F(W) = \lim_{N\to\infty} \frac{1}{N} \sum_{k=1}^{N} (t_k - I_k)^2 = E[(t_k - I_k)^2]$

$$\Rightarrow F(W) = E[(t_k - I_k)^2] = E[(t_k - W^T X_k)^2]$$

$$= E[t_k^2 - 2 t_k W^T X_k + W^T X_k X_k^T W]$$

$$= \underbrace{E[t_k^2]}_{\substack{P \\ \text{constant scalur} \\ \text{هر } t_k \text{ یک عدد است}}} - 2W^T \underbrace{E[t_k X_k]}_{\substack{q \\ \text{vector} \\ \text{بردار}}} + W^T \underbrace{E[X_k X_k^T]}_{\substack{R \\ \text{matrix} \\ \text{ماتریس}}} W = \underbrace{P - 2W^T q + W^T R W}_{\substack{\text{quadratic form} \\ \text{معادله درجه ۲ نسبت به } W}}$$

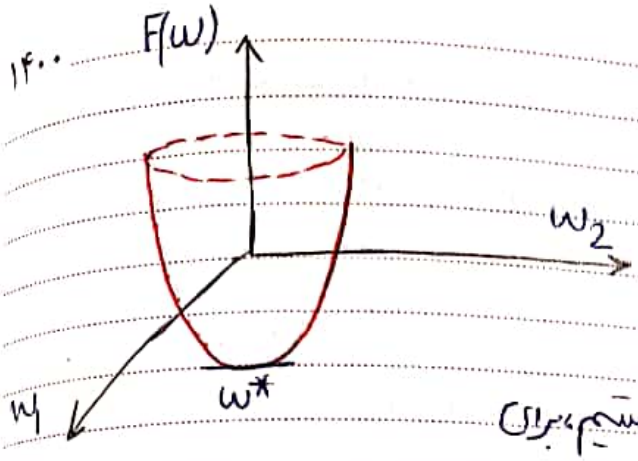
ماتریس R یک ماتریس متقارن است و مثبت معین است و این رابطه در مورد عناصر آن برقرار است:

$R : (n+1) \times (n+1)$

$$\forall X : \sum\sum r_{ij} x_i x_j \geq 0$$

روز جهانی قدس (آخرین جمعه ماه رمضان)

Scanned with CamScanner



همانطور که گفتیم F نسبت به w درجه ۲ است

در فضای وزن ها، F چنین شکلی دارد:

شکل سهمی گون (paraboloidal) دارد

این سطح یک minimum دارد که ما به دنبال آن هستیم، برای

یافتن min، از F(w) نسبت به w مشتق می گیریم:

$$\nabla_w F(w) = -2q + 2Rw = 0$$

$$\Rightarrow Rw^* = q \quad \Rightarrow w^* = R^+ q$$

R^+ ← شبه معکوس یا معکوس تعمیم یافته R (pseudo-inverse or generalized inverse)

اگر R معکوس پذیر باشد: $R^+ = R^{-1}$ و یک minimum یکتا دارد شبیه شکل بالا.

این سطح را می گویند non-degenerate ← هموار $w^* = R^{-1} q$

اگر R معکوس پذیر نباشد سطح متفاوت می شود و هموار نیست، ممکنه minimum های مختلف

داشته باشه و اگر چند min باشه نزدیکترین min به مبدا را به ما می دهد: $w^* = R^+ q$

در این حالت سطح را degenerate می گویند ← ناهموار است

ما به R و q نیاز داریم اما پیدا کردن آن ها کار راحتی نیست. به همین دلیل راه حل دیگری پیشنهاد شد

اینکه بیایم روی سطح حرکت کنیم و آنقدر پایین برویم که به min برسیم. اگر در جهت عکس گرادیان F نسبت

به w حرکت کنیم در هر iteration به min نزدیکتر می شویم و بهترین جهت برای حرکت است.

گرادیان F جهت بیشترین نرخ افزایش را نشان می دهد و جهت عکس آن، بیشترین نرخ کاهش

را نشان می دهد

Scanned with CamScanner

$$\nabla_w F(w) = \nabla_w \left[\lim_{N\to\infty} \frac{1}{N} \sum (t_k - I_k)^2 \right] = \lim_{N\to\infty} \frac{1}{N} \nabla_w \left[\sum (t_k - I_k)^2 \right]$$

$$= \lim_{N\to\infty} \frac{1}{N} \left[\sum 2 \underbrace{(t_k - I_k)}_{\delta_k} \underbrace{\nabla_w (-I_k)}_{-X_k} \right] \qquad I_k = w^T x_k$$

$$\boxed{\nabla_w F(\mathbf{w}) = -\lim_{N\to\infty} \frac{1}{N} \sum_{k=1}^{N} 2\delta_k X_k = -2E[\delta_k X_k]}$$

$$\boxed{w_{k+1} = w_k + \alpha \delta_k X_k} \qquad 0 < \alpha < \frac{2}{\lambda_{max} R}$$

$E[\delta_k X_k]$ یعنی $\delta_k X_k$ های مختلف را برای بی‌نهایت نمونه X_k میانگین بگیریم. widrow گفت بجای اینکه امید ریاضی $\delta_k X_k$ را در نظر بگیریم، خود $\delta_k X_k$ را در نظر می‌گیریم در هر iteration، بنابراین قاعده به شکل $w_{k+1} = w_k + \alpha \delta_k X_k$ درآمد. بنابراین این شد قاعده‌ای که x را اعمال می‌کنیم وزن‌ها را به صورت تکراری به هنگام در می‌آورید تا به min برسید.

widrow, Hoff Learning law
LMS learning law
Delta rule

$\lambda_{max} R$ ← بزرگترین مقدار ویژه‌ی ماتریس R، اما ماتریس R را نداریم

خود widrow پیشنهاد داد که مقدار α در این بازه باشد: $0 < \alpha < \frac{2}{\lambda_{max} R}$

گفتند برای α یک مقدار اولیه‌ای بین $0.01 \leq \alpha \leq 10$ در نظر بگیریم مثلا $\alpha = 0.1$

و با سعی و خطا سعی کنید α ای بیابید که مناسب باشد

اگر α خیلی بزرگ باشد باعث می‌شود الگوریتم همگرا نشود و اگر خیلی کوچک باشد مشکلی که پیش می‌آید این است که الگوریتم خیلی کند می‌شود.

یه پیشنهاد دیگه‌ای که داد این بود که: $\frac{1}{10n} \leq \alpha \leq \frac{1}{n}$ که n تعداد ابعاد ورودی (X) است

Scanned with CamScanner

دو نسخه دیگر هم از قانون آموزش widrow وجود دارد:

① The batching Version

در این نسخه به جای اینکه به ازای هر x اعمال شده w ها را به هنگام درآورد، این کار را نمی‌کند. میاد x را اعمال می‌کند، δ را پیدا می‌کند ولی w را به هنگام در نمی‌آورد، دوباره x بعدی را اعمال می‌کند δ را می‌یابد ولی w را به هنگام در نمی‌آورد و ... تا به تعداد ورودی. وقتی همه ورودی‌ها اعمال شدند (یک batch از ورودی)، حالا حاصل ضرب $\delta_k x_k$ را برای همه حساب می‌کند و میانگین آن را به دست می‌آورد و w را اصلاح می‌کند.

در واقع با میانگین از $\delta_k x_k$ ها w ها را اصلاح می‌کند و مزیتش این است که سریعتر است.

② The Momentum Version

در این نسخه یک جمله با ضریب μ به قانون اضافه می‌کند:

$$w_{k+1} = w_k + \alpha(1-\mu)\delta_k x_k + \mu(w_k - w_{k-1})$$

$\mu \gg \alpha$
(eg. $\mu = 0.9, \alpha = 0.1$)

$w_k - w_{k-1}$ تغییرات w در iteration قبلی است. برای اعمال تغییر در w به ضریبی از تغییر قبل را هم در نظر می‌گیرد علاوه بر ورودی‌های این دفعه.

یک ضریب $(1-\mu)$ هم گذاشته برای کنترل نقش این دو جمله

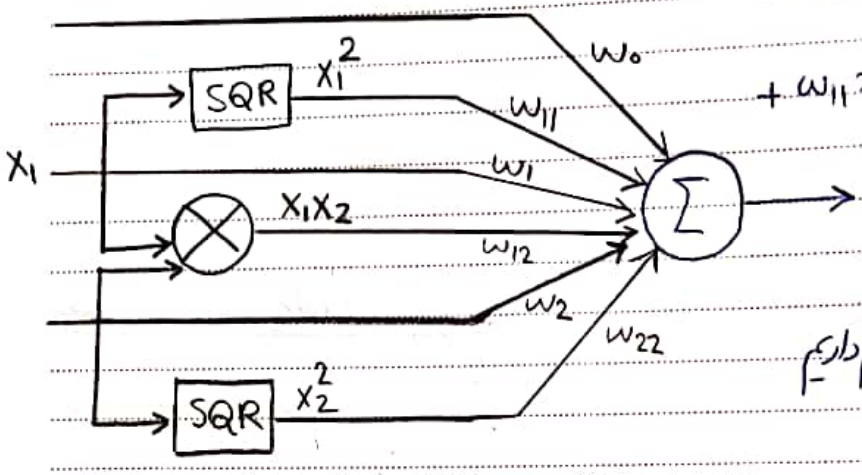
مزیت: کمک می‌کند از نوسانات زیاد برای وزن‌ها در یک امتداد جلوگیری شود

این دو نسخه هر کدام در یک جایی بهتر عمل می‌کنند و به طور کلی نمی‌توان گفت کدام یک بهتر است

Scanned with CamScanner

Second order AdaLine

$$I = w_0 + w_1 x_1 + w_2 x_2 + w_{11} x_1^2 + w_{22} x_2^2 + w_{12} x_1 x_2$$



مثل پرسپترون درجه ۲، Adaline درجه ۲ هم داریم

جلسه هفتم - ۱۴۰۰/۷/۲۰

MultiLayer Feedforward Neural Networks — شبکه‌های جلورو چندلایه

این شبکه‌ها می‌توانند تعداد لایه‌های زیاد هم داشته باشند و عمیق شوند. هم‌چنین به آن‌ها MLP Multilayer Perceptrons هم گفته می‌شود. (چون از گذشته با پرسپترون آغاز شد)

می‌تواند برای افزایش توانایی شبکه‌های تک لایه در حالتی که داده‌های ما به صورت خطی جدایی‌پذیر نیستند، مورد استفاده قرار بگیرد.

در این شبکه‌ها تابع فعال شدن نورون‌ها حتماً باید پیوسته باشد. تابع پله‌ای یا علامت نمی‌تواند باشد. علتش هم این است که لازم داریم از این تابع مشتق بگیریم.

این شبکه‌ها حداقل دارای ۳ لایه هستند: ورودی، یک لایه مخفی، خروجی

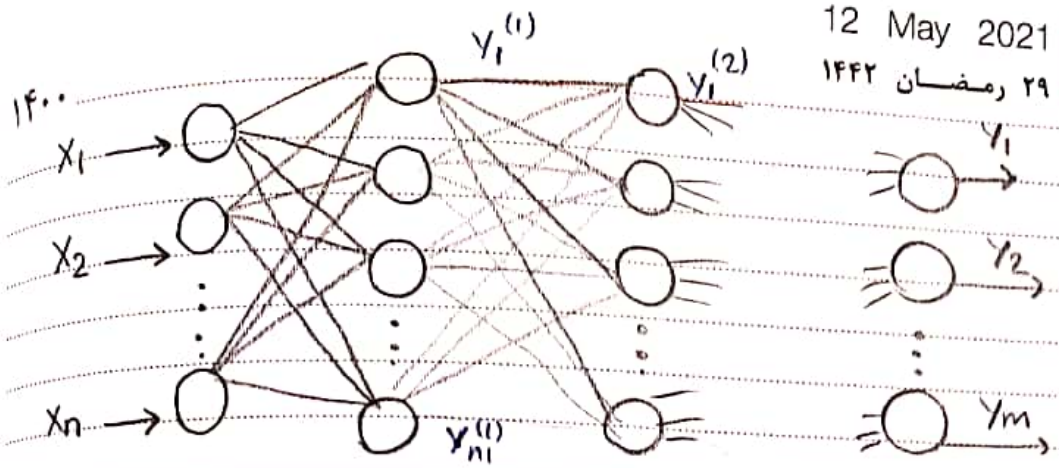
تعداد نورون‌ها در لایه‌ی مخفی K را با n_K نشان می‌دهیم

اتصالات از یک لایه به لایه بعد کامل است. یعنی هر خروجی از نورون در یک لایه به تمام نورون‌های لایه‌ی بعدی متصل می‌شود.

نورون‌های لایه ورودی هیچ تغییری روی x نمی‌دهند و فقط ورودی را به لایه بعد توزیع می‌کنند

Scanned with CamScanner

۱۴۰۰



خروجی لایه مخفی اول: $Y_1^{(1)}$ ← شماره لایه ، $Y_1^{(1)}$ ← شماره نورون ، $Y_2^{(1)} \dots Y_{n_1}^{(1)}$

خروجی لایه مخفی دوم: $Y_1^{(2)}$ و $Y_2^{(2)}$ و ... و $Y_{n_2}^{(2)}$

قاعده ای که ابتدا برای آموزش این شبکه پیشنهاد شد:

قاعده پس انتشار خطا — Error backpropagation rule (BP)

قاعده دلتای تعمیم یافته — Generalized Delta Rule

در آموزش این شبکه ۳ مرحله باید طی شود:

① یک ورودی به شبکه اعمال می شود، این ورودی باید تا انتهای شبکه پیش برود و تمام نورون های شبکه، خروجی شان حساب شود. با مشخص بودن وزن ها و توابع فعال سازی، خروجی ها حساب می شود.

② خروجی هایی که در لایه های خروجی تولید شده با خروجی نظیر مطلوبش مقایسه شود و مقدار خطا محاسبه شود. این خطا می شود، خطا در خروجی شبکه.

از این خطا می توانیم میزان خطا در یک لایه قبل را محاسبه کنیم و مجددا با استفاده از این خطا می توانیم خطاهای لایه قبل را محاسبه کنیم و ... تا برسیم به اولین لایه مخفی. اسم این روش از این جا آمده. خطا را از خروجی به سمت عقب انتشار می دهد.

③ با استفاده از خطا، تغییر وزن ها را در همه ی اتصالات محاسبه می کنیم

Scanned with CamScanner

این ۳ مرحله آنقدر تکرار می‌شود روی ورودی‌های مختلف تا اینکه الگوریتم همگرا شود و خطا به حد قابل قبولی در خروجی برسد.

استخراج قانون BP :

فرض ← در فاز آموزش نمونه‌های ورودی را با S نشان می‌دهیم

$$S_p = \begin{bmatrix} S_{p_1} \\ S_{p_2} \\ \vdots \\ S_{p_n} \end{bmatrix}$$ ← p امین نمونه در مجموعه آموزشی

$$Y_p^{(l)} = \begin{bmatrix} Y_{p_1}^{(l)} \\ y_{p_2}^{(l)} \\ \vdots \\ y_{p_{nl}}^{(l)} \end{bmatrix}$$ ← وقتی S_p اعمال می‌شود در خروجی لایه‌های مخفی این را داریم

$$T_p = \begin{bmatrix} t_{p_1} \\ t_{p_2} \\ \vdots \\ t_{p_m} \end{bmatrix}$$ ← خروجی مطلوب برای S_p

خطا در خروجی ← E_p (وقتی ورودی S_p اعمال شده باشد)

$w_{ij}^{(l)}$ ← وزن اتصال از واحد i از لایه‌ی l-1 به واحد j در لایه‌ی l

عید سعید فطر (تعطیل)

تابع خطا:

$$E_p = \frac{1}{2} \sum_j (t_{pj} - y_{pj})^2$$

خطای تک تک نورون‌های لایه خروجی را با هم جمع کرده‌ایم و به توان ۲ رسانده‌ایم

فعالیت یا ورودی خالص نورون j در لایه‌ی l به واسطه‌ی ورودی p :

$$I_{pj}^{(l)} = \sum_k w_{kj}^{(l)} y_{pk}^{(l-1)}$$

خروجی نورون j برای سمپل p در لایه l :

$$y_{pj}^{(l)} = f_j(I_{pj}^{(l)})$$

می‌خواهیم مشتق خطا را نسبت به وزن w_{ij} در لایه‌ی l پیدا بکنیم.

تغییرات خطا در خروجی با یک وزنی که از نورون i در لایه‌ی l-1 رفته به نورون j در لایه l، را می‌خواهیم بیابیم.

تعطیل به مناسبت عید سعید فطر

Scanned with CamScanner

از قاعده زنجیره‌ای مشتق استفاده می‌کنیم:

$$\frac{\partial E}{\partial w_{ij}^{(l)}} = \frac{\partial E_p}{\partial I_{pj}^{(l)}} \cdot \frac{\partial I_{pj}^{(l)}}{\partial w_{ij}^{(l)}}$$

$$-\frac{\partial E_p}{\partial I_{pj}^{(l)}} = \delta_{pj}^{(l)}$$

$$\frac{\partial I_{pj}^{(l)}}{\partial w_{ij}^{(l)}} = \frac{\partial}{\partial w_{ij}^{(l)}}\left[\sum_k w_{kj}^{(l)} y_{pk}^{(l-1)}\right] = y_{pi}^{(l-1)}$$

فقط یکی از جمله‌ها که $k=i$ است مشتق آن غیر صفر می‌شود

$$\Rightarrow -\frac{\partial E_p}{\partial w_{ij}^{(l)}} = -\nabla_w E_p = \delta_{pj}^{(l)} y_{pi}^{(l-1)}$$

این رابطه نشان می‌دهد که تغییرات خطا در خروجی چگونه وابسته است به تغییرات وزن‌ها در شبکه.

رابطه‌ی update شدن وزن‌ها در BP:

$$\boxed{w_{ij}^{new(l)} = w_{ij}^{old(l)} + \eta\, \delta_{pj}^{(l)} y_{pi}^{(l-1)}}$$

η با سعی و خطا تعیین می‌شود

روش نزول در امتداد گرادیان

مشخص کردن $\delta_{pj}^{(l)}$:

$$\delta_{pj}^{(l)} = -\frac{\partial E_p}{\partial I_{pj}^{(l)}} \Rightarrow \delta_{pj}^{(l)} = -\frac{\partial E_p}{\partial y_{pj}^{(l)}} \cdot \frac{\partial y_{pj}^{(l)}}{\partial I_{pj}^{(l)}}$$

$$E_p = -\frac{1}{2}\sum_j (t_{pj} - y_{pj})^2$$

$$y_{pj}^{(l)} = f_j(I_{pj}^{(l)})$$

$$\frac{\partial E_p}{\partial y_{pj}} = -(t_{pj} - y_{pj})$$

$$f_j'(I_{pj}^{(l)})$$

در این جا l ها را ننوشته‌ایم چون E_p را فقط برای لایه خروجی داریم

برای واحدهای خروجی داریم ←

$$\boxed{\delta_{pj} = (t_{pj} - y_{pj}) \cdot f_j'(I_{pj})}$$

Scanned with CamScanner

برای لایه‌های مخفی:
(E_p را نداریم)

$$\delta_{pj}^{(l)} = -\frac{\partial E_p}{\partial I_{pj}^{(l)}} \Rightarrow \delta_{pj}^{(l)} = -\frac{\partial E_p}{\partial y_{pj}^{(l)}} \cdot \frac{\partial y_{pj}^{(l)}}{\partial I_{pj}^{(l)}}$$

$$f_j'(I_{pj}^{(l)})$$

$$\frac{\partial E_p}{\partial y_{pj}^{(l)}} = \sum_k \frac{\partial E_p}{\partial I_{pk}^{(l+1)}} \cdot \frac{\partial I_{pk}^{(l+1)}}{\partial y_{pj}^{(l)}}$$

$$-\delta_{pk}^{(l+1)}$$

$$\frac{\partial}{\partial y_{pj}^{(l)}} \sum w_{ik}^{(l+1)} y_{pi}^{(l)} = w_{jk}^{(l+1)}$$

$$\Rightarrow \frac{\partial E_p}{\partial y_{pj}^{(l)}} = -\sum_k \delta_{pk}^{(l+1)} \cdot w_{jk}^{(l+1)}$$

$$\boxed{\delta_{pj}^{(l)} = f_j'(I_{pj}^{(l)}) \cdot \sum_k \delta_{pk}^{(l+1)} \cdot w_{jk}^{(l+1)}}$$

برای لایه‌های مخفی ←

الان برای خطا در لایه l یک فرمولی پیدا شده که می‌شه با استفاده از خطا در لایه $l+1$ آن را محاسبه کرد

در نتیجه با استفاده از این رابطه می‌توانیم خطا را از لایه بعدی به لایه قبلی منتشر کنیم. (قاعده BP)

همانطور که گفتیم تابع f باید مشتق‌پذیر باشد چون در روابطمان f' داشتیم.

تابع سیگموئید باینری (Binary Sigmoid) مشتق دارد و می‌توان برحسب خودش نوشت:

$$f(z) = \frac{1}{1+e^{-kz}} \rightarrow f'(z) = k f(z)\left[1 - f(z)\right]$$

تابع Bipolar Sigmoid هم می‌تواند استفاده شود:

$$f(z) = \frac{2}{1+e^{-kz}} - 1 \rightarrow f'(z) = \frac{k}{2}\left[1 + f(z)\right]\left[1 - f(z)\right]$$

Scanned with CamScanner

الگوریتم آموزش BP :

① ابتدا مقادیر تصادفی کوچکی به وزن ها اختصاص بدهید

① تا زمانی که شرط توقف غلط است قدم های ۲ تا ۷ را انجام بده

② برای هر زوج آموزشی (S_p, T_p) قدم های ۳ تا ۶ را تکرار کن

③ خروجی همه ی واحدها را از اول شبکه تا آخر شبکه حساب کن:

$$y_{pj}^{(1)} = f_j\left[\sum_{i=0}^{n} w_{ij} S_{pi}\right]$$ ← برای لایه مخفی اول که ورودی اش S است

$$y_{pj}^{(l)} = f_j\left[\sum w_{ij}^{(l)} y_{pi}^{(l-1)}\right] \quad l>1$$ ← برای لایه های بعدی به جای S، y داریم

④ دلتا را برای واحدهای خروجی حساب کن:

$$\delta_{pj} = (t_{pj} - y_{pj})\, f_j'(I_{pj})$$

⑤ دلتا را برای همه ی واحدهای مخفی پیدا کن:

$$\delta_{pj}^{(l)} = \left(\sum_k \delta_{pk}^{(l+1)} w_{jk}^{(l+1)}\right) f_j'(I_{pj}^{(l)})$$

به این صورت محاسبه می شود:

δ را در لایه خروجی داریم، با استفاده از آن δ را برای یک لایه قبل حساب کن و مجددا با استفاده از δ بدست آمده برای یک لایه قبل حساب کن و همینطور رو به عقب.

⑥ همه ی وزن ها را update کن:

$$w_{ij}^{new(l)} = w_{ij}^{old(l)} + \eta\, \delta_{pj}^{(l)} y_{pi}^{(l-1)}$$

⑦ چک کردن شرط خاتمه

انقدر برای ورودی های مختلف این مراحل را تکرار می کنیم تا شرط خاتمه صحیح شود

Scanned with CamScanner

مجموعه آموزشی فقط یکبار به شبکه اعمال نمی‌شود بلکه باید چندین بار کل داده‌های آموزشی به شبکه اعمال شود تا به همگرایی برسیم. وقتی یک دور همه‌ی داده‌های آموزشی را اعمال کنیم به آن یک دور اعمال ورودی یا epoch گفته می‌شود.

الگوریتم BP یک الگوریتم کند است چون باید مکرر ورودی‌ها را به شبکه اعمال کنیم و معمولا داده‌های آموزشی زیاد هستند، هم‌چنین محاسبات زیاد است و بارها باید تکرار شود. در نتیجه آموزش شبکه‌های feed forward یک آموزش زمان‌بر و کند است و تعداد پارامترها زیاد است.

فرض کنید آموزش انجام شده، چگونه از شبکه آموزش دیده استفاده کنیم؟

⓪ وزن‌ها را برابر با وزن‌های فاز آموزش قرار بدهید

① ورودی را بدهید و از این رابطه خروجی نورون‌ها را در لایه اول حساب کنید:

$$y_i^{(1)} = f\left(\sum_{j=0}^{n} w_{ji}^{(1)} x_j\right)$$

② از روی خروجی نورون‌های لایه اول، لایه‌های بعدی را حساب کنید:

$$y_i^{(l)} = f\left(\sum_j w_{ji}^{(l)} y_j^{(l-1)}\right)$$

③ خروجی لایه آخر بدست می‌آید

شبکه feed forward آموزشش سنگین است، استفاده از آن راحت است. وقتی وزن‌ها مشخص شد کافیه یه بار مسیر از ورودی تا خروجی پیمایش بشه تا خروجی مشخص شود.

Network operation:

تابع انرژی شبکه (تابع خطای شبکه) ← E_{rms}

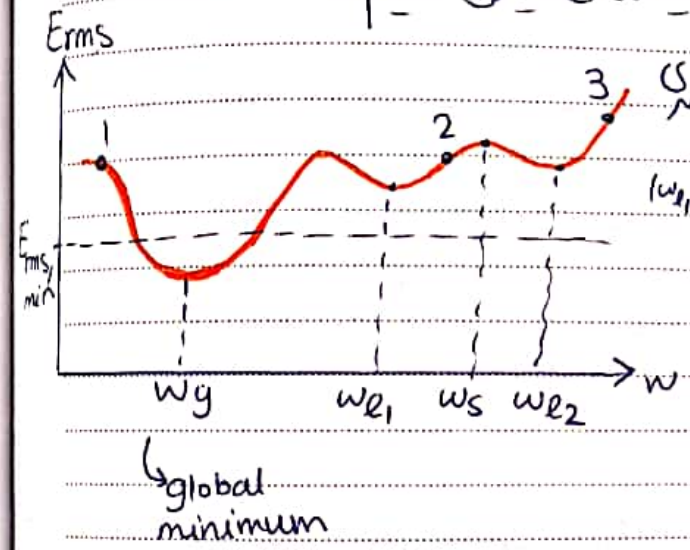
در این شبکه هدف ما این است وزن‌ها به گونه‌ای پیدا شوند که این خطا حداقل شود و این عمل را به این صورت انجام می‌دهیم که در جهت خلاف گرادیان روی سطح انرژی (خطا) می‌آییم پایین.

در این شبکه تابع خطا یک سطح بسیار پیچیده‌ای در فضای وزن‌ها دارد بنابراین ممکن است min های محلی

Scanned with CamScanner

۱۴۰۰

زیادی داشته باشد و احتمالش زیاد است که در این min های محلی گیر بیفتیم و وقتی که در این min های محلی افتاد دیگر نمی‌تواند خارج شود و الگوریتم ما متوقف می‌شود و به یک min محلی همگرا می‌شود. اینکه این اتفاق بیفتد خیلی به این بستگی دارد که نقطه شروع ما روی سطح خطا کجا باشد مثلا اگر از نقطه‌ای در نزدیکی min سراسری آغاز کنیم ممکنه به راحتی min سراسری را پیدا کنیم اما اگر از یک نقطه در نزدیکی یک min محلی شروع کنیم به احتمال زیاد در آن گیر می‌افتیم.



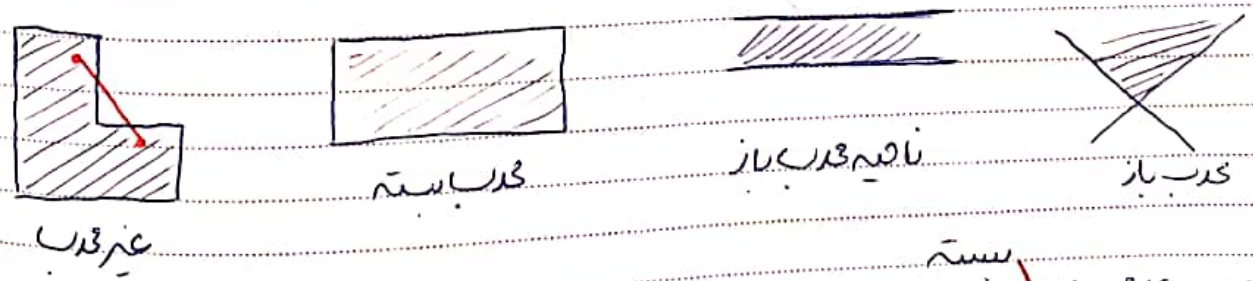
مثال: در شکل روبه‌رو اگر از نقطه 1 شروع کنیم به min سراسری می‌رسیم. اگر از نقطه 2 شروع کنیم در min محلی گیر میفتیم (w_{l_1}) در مورد نقطه 3 هم در min محلی گیر می‌افتیم (w_s)

← بنابراین انتخاب مقادیر اولیه وزن‌ها مهم است

جلسه هشتم - ۱۴۰۰/۷/۲۵

عملکرد شبکه (MLP) به عنوان دسته‌بند

یک ناحیه را محدب (convex) می‌گوییم اگر هر دو نقطه از ناحیه را که با یک خط بهم متصل می‌کنیم، تمام آن خط، درون ناحیه بیفتد



ناحیه‌های محدب: بسته / باز

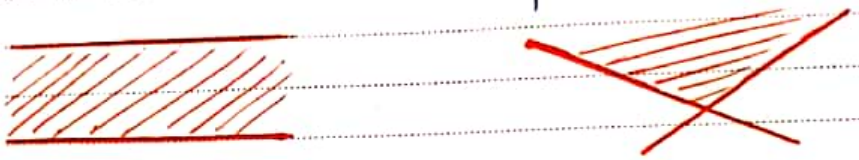
وقتی یک نورون داشته باشیم میاد در فضای ورودی یک خط می‌کشد و ناحیه را به ۲ بخش تقسیم می‌کند

Scanned with CamScanner

وقتی در لایه مخفی اول چندتا نورون داریم، هرکدام از نورون ها یک خط در فضا ترسیم می کند

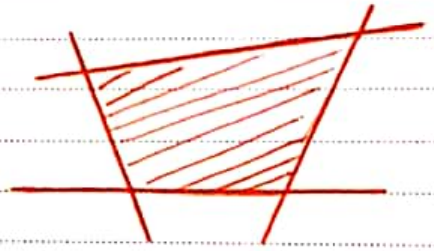
در لایه مخفی دوم، این خطوط رسم شده به گونه ای که مطلوب است ترکیب می شوند تا ناحیه مورد نظر ما را تشکیل دهند

فرض کنید یک شبکه با معماری 1-2-n داشته باشیم:



معماری 1-3-n:

معماری 1-4-n:



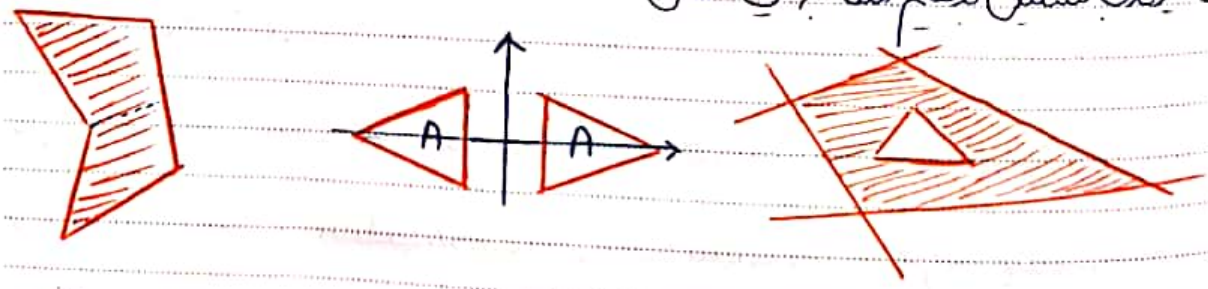
⇐ نتیجه: تعداد اضلاع ناحیه ای که تشکیل می شود برابر تعداد نورون های لایه مخفی اول است.

روز ملی جمعیت

اردیبهشت ۳۱ جمعه
21 May ■ ۹ شوال

اینکه خط رسم شده یا منحنی، تابع فعال سازی نورون هست که این را مشخص می کند. ربطی به بعد ورودی ندارد.

همانطور که دیدیم با معماری های ۳ لایه، نواحی به دست آمده محدب هستند. خیلی اوقات نیاز داریم ناحیه ی غیرمحدب تشکیل بدهیم مثلا به این شکل:



راه حل: یک لایه اضافه کنیم. مثلا در مورد شکل سمت راست، مثلث و ۴ ضلعی در لایه سوم ایجاد می شود و در لایه چهارم با هم ترکیب می شوند. یا مثلا سمت چپ: از دو شکل محدب تشکیل شده، هرکدام در لایه ۳ تولید می شود و در لایه ۴ ترکیب می شوند. پس با شبکه ۴ لایه می توان هر شکلی با هر پیچیدگی را ایجاد کرد.

روز اهدای عضو، اهدای زندگی

Scanned with CamScanner

تعداد لایه‌ها	نوع نواحی	مسئله XOR	کلاس‌های با نواحی Meshed	نواحی با شکل کلی‌تر
2	half plane bounded by Hyperplane	A B B A	B A	
3	Convex Open or closed Regions	A B B A	B A	
۲ تا خط در فضا میتونه باشه	Arbitrary (Complexity Limited by No. of Nods)	A B B A	B A	

Memorization and Generalization

Memorization: وقتی شبکه را آموزش داریم، مشخص می‌کند شبکه چقدر این‌ها را خوب یاد گرفته

یعنی میزان صحت پاسخ‌ها به داده‌هایی که برای آموزش به کار رفته

یعنی این شبکه با آموزشی که دیده قدرت حفظ کردنش چقدر بوده

Generalization: عبارت است از قدرت شبکه برای پاسخ صحیح دادن به نمونه‌های جدیدی که با آن‌ها آموزش ندیده است.

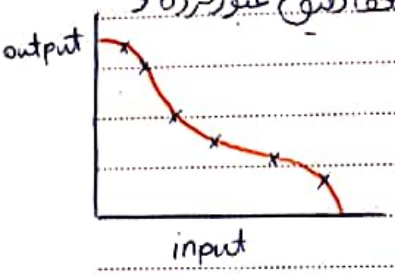
← برای ما مطلوب است که Memorization و Generalization هر دو بالا باشند.

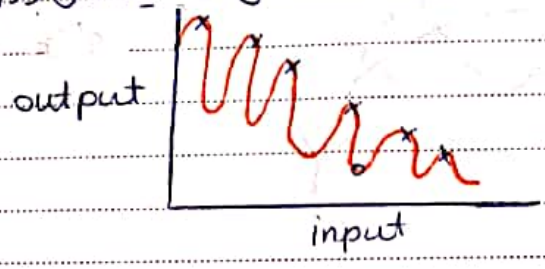
اما مشاهده می‌شود که Memorization و Generalization ضد هم کار می‌کنند. یعنی در آموزش شبکه اگر سعی کنیم Memorization را بالا ببریم Generalization پایین می‌آید و برعکس. معمولا در طراحی شبکه باید به یک تعادلی میان این‌ها برسیم.

Scanned with CamScanner

یک سری راه‌هایی پیدا شده که هم Memorization و هم Generalization بالا باشد. که این راهکارها با تغییراتی در شبکه همراه است مثلا تغییراتی در تابعی که minimize می‌کنیم برای آموزش شبکه، ایجاد کنیم.

آموزش این شبکه خیلی شبیه برازش منحنی است. داده‌های ما نقاطی هستند که در برازش منحنی داریم و تابعی که شبکه باید یاد بگیرد همان منحنی است که برازش می‌کنیم. اگر بخواهیم تشابه آموزش شبکه را با مسئله‌ی برازش منحنی در نظر بگیریم مسئله‌ی Memorization می‌شود اینکه منحنی برازش شده چقدر از داده‌ها دقیق عبور کرده و





مسئله Generalization می‌شود اینکه شبکه چقدر خوب می‌تواند با آموزشی که دیده interpolation انجام دهد.

با توجه به دو منحنی رسم شده در بالا، در حالت سمت چپ Generalization مناسبی داریم چون هر نقطه‌ای به عنوان تست به مدل بدهیم یک جواب معقول در محدوده‌ی داده‌هایی که داریم به ما می‌دهد. اما در شکل سمت راست Generalization پایین است. ممکنه برای یک داده تست که می‌دهیم جواب دارای خطای بالایی باشد. در شکل سمت راست Memorization خوب است یعنی برای داده‌های آموزشی خوب جواب می‌دهد.

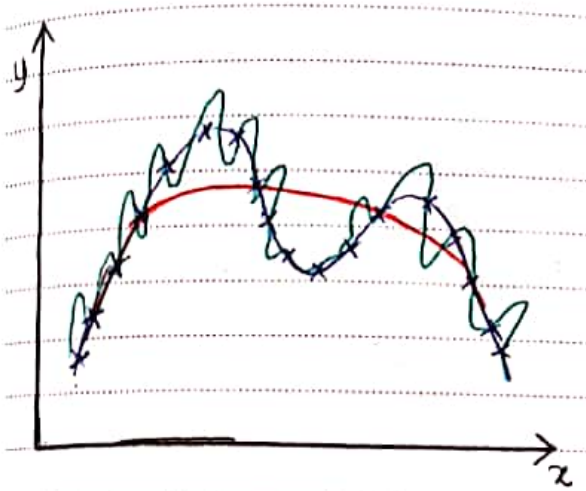
این دقیقاً مسئله‌ای است که در curve fitting هم داریم. در سمت چپ interpolation خوب است در سمت راست نه.

اگر شبکه را با تعداد زیادی نمونه آموزشی و تعداد epoch های زیاد آموزش بدهیم، شبکه به سمت حفظ کردن داده‌های آموزشی (Memorization) می‌رود و Generalization کم می‌شود.

در این حالت معمولا ممکنه لازم بشه مجبور بشویم نورون‌ها را زیاد کنیم. وقتی نورون‌ها زیاد می‌شود در واقع تعداد پارامترها زیاد می‌شود. مشابه آن در curve fitting هم داریم یعنی اگر تعداد پارامترهای منحنی که می‌خواهید فیت کنید زیاد باشد overfitting پیش می‌آید.

Scanned with CamScanner

در این حالت شبکه به داده‌های نویز هم حساس است. چون کوچکترین مقادیر نویز را هم یاد خواهد گرفت (حفاظ کند در مسئله (مدل سعی دارد منحنی ای یافت کند که از همه نقاط عبور می‌کند) در این حالت در برازش منحنی interpolation و extrapolation ضعیف است.



در شکل زیر اگر دو نورون در لایه مخفی بگذاریم، منحنی قرمز به دست می‌آید. اگر ۵ تا نورون بگذاریم منحنی آبی به دست می‌آید.

اگر ۱۴ تا بگذاریم، منحنی سبز به دست می‌آید. از نقاط عبور می‌کند اما چون تعداد پارامترها زیاد است، منحنی حاصل یک منحنی درجه بالا است در نتیجه نوسانی می‌شود.

در برازش منحنی با منحنی‌های ساده شروع می‌کردیم. این جا هم دوست داریم تابعی که برای شبکه یادگرفته می‌شود ساده‌ترین حالت ممکن باشد. چون در این صورت محاسبات و تعداد نورون‌ها کمتر است.

ما خواهیم بی‌جهت تعداد نورون‌ها و پارامترها را زیاد نکنیم. ترجیحمان این است که تعداد نورون‌ها کم باشد. می‌خواهیم بررسی کنیم انتظاری که از شبکه feed forward داریم که پاسخ خوب بدهد برای نمونه‌های جدید تا چه حد معقول و منطقی است؟

فرض کنید مجموعه همه داده‌های ورودی و خروجی یک مسئله در مجموعه U باشد (مستطیل) یک سری از نقاط این مجموعه با تابع R هماهنگ هستند و ما می‌خواهیم شبکه‌ای را آموزش بدهیم که این را تشخیص بدهد. شبکه را با نمونه‌های آموزشی که انتخاب می‌کنیم آموزش می‌دهیم. فرض کنیم نقاط داخل ناحیه T، مجموعه آموزشی باشند. نقاط مجموعه X هم به عنوان تست به کار می‌بریم. آموزش شبکه باید آن چه را با مجموعه T یادگرفته تعمیم بدهد تا برای نقاط جدید جواب بدهد.

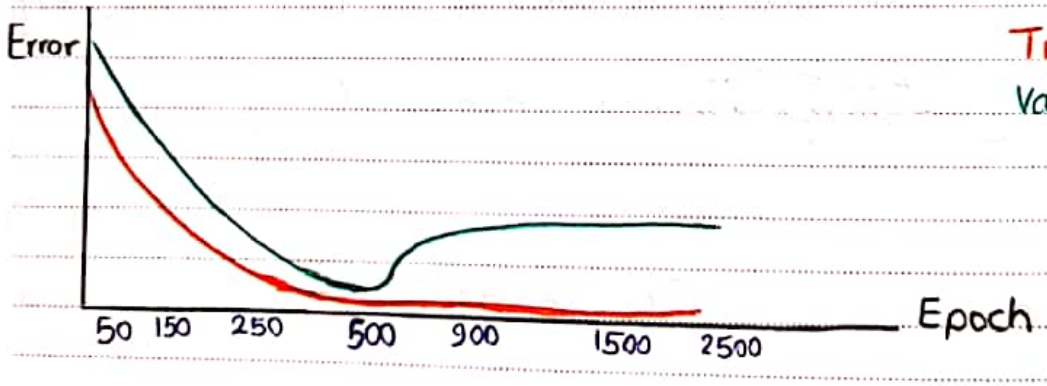
Scanned with CamScanner

به شکل‌های مختلفی می‌تواند آموزشش از روی مجموعه T را تعمیم بدهد. مانند شکل، نقاط خط‌چین نشان می‌دهد آموزش داده شده به شبکه این ناحیه را یاد گرفته و خارج از آن را نه. در خیلی از حالات تعمیم یادگرفته برای T اصلا شامل X نمی‌شود. آن چه که شبکه دیده نقاط مجموعه T بوده، چطور انتظار داریم برای تمام نقاط جواب صحیح بدهد؟!

باید انتظارات خود را کم کنیم و نکته دیگر اینکه چقدر آموزش شبکه مهم است

نکته دیگر: نقاطی که در مجموعه آموزش می‌گذاریم خیلی مهم است مثلا در این جا اگر نقاط آموزشی که انتخاب می‌کنیم در سرتاسر R پخش بودند، نتیجه بهتری حاصل می‌شد. چون ناحیه X را هم پوشش می‌دهد.

در یک سیستم واقعی متناظر این است که برای همه حالاتی که سیستم می‌تواند داشته باشد ورودی اعمال کنیم و به خروجی حاصل، به سیستم بدهیم. شامل همه حالات است حتی وقتی سیستم خراب است یا مشکل دارد...



در این نمودار برای یک مسئله، خطای مجموعه آموزش و Validation به ازای تعداد epoch های مختلف رسم شده است. تا حدود epoch=۵۰۰ آموزش به خوبی یاد گرفته که هم برای Validation set و هم برای Training set خطا را حداقل کند اما از ۵۰۰ به بعد خطای Validation set در حال افزایش است یعنی شبکه برای نمونه‌های جدید تصمیم‌گیری غلط می‌کند. یعنی تا epoch=۵۰۰، Memorization و Generalization خوب هستند اما از ۵۰۰ به بعد Generalization شروع به خراب شدن می‌کند و Memorization همچنان در حال بهبود است. پس نتیجه می‌گیریم که نباید آموزش را بیشتر از ۵۰۰ تا epoch ادامه بدهیم.

Scanned with CamScanner

نکته: همانطور که گفتیم با ۲ لایه در شبکه FF می‌توانیم ناحیه‌بندی فضا را به هر شکلی که می‌خواهیم انجام بدهیم این حرف به این معنا است که می‌توانیم هر مسئله‌ی دسته‌بندی را با شبکه ۲ لایه حل بکنیم. اما ممکن آموزش شبکه خیلی سخت باشد و با اضافه کردن لایه، آسانتر حل شود و آموزش صورت بگیرد.

جلسه نهم - ۱۴۰۰/۷/۲۷

مسئله دوم: تخمین تابع (Regression or Function Approximation)

در بسیاری از کاربرها با این وضعیت روبه‌رو هستیم که مقدار زیادی داده از یک پدیده جمع‌آوری کرده‌ایم و می‌خواهیم مدلی برای این پدیده بسازیم و یک تابع به این داده‌ها فیت کنیم

فرض کنید مجموعه‌ای از مشاهدات را از پدیده‌ای داریم که شامل بردار X و y است. بردار X تحریکی است که به آن پدیده وارد می‌شود و y پاسخی است که آن پدیده می‌دهد. هدف این است که برای این پدیده مدلی را توسعه بدهیم که به کمک آن بتوانیم اگر X جدیدی داریم y ای که پاسخ آن است را پیش‌بینی کنیم.

این مسئله همیشه بوده و رویکردی که از پیش استفاده می‌شده، برازش منحنی بوده است.

شبکه‌های عصبی این عمل را به خوبی انجام می‌دهند

تخمین تابع با شبکه‌های FF:

فرض کنید نگاشت f را داریم که: $f: A \subset R^n \to R^m$ که با مجموعه‌ای از سمپل‌ها (S_i, T_i) $i=1,2,\dots,L$ نشان داده می‌شود.

این رابطه را هم داریم: $T_k = f(S_k)$ $k=1,2,\dots,L$ یعنی مقدار مطلوبی که ما داریم، با تابع f مرتبط می‌شود با مقدار تحریک. این تفاوتی است که با مسئله‌ی دسته‌بندی دارد.

Scanned with CamScanner

فرض کنید X_k یکی از نمونه‌های آموزشی است. مربع خطا برای آن اینگونه بدست می‌آید:

$$f_k(X_k, W) = |f(X_k) - y_k(X_k, W)|^2$$

y خروجی پیش‌بینی شده توسط شبکه در صورت اعمال تحریک X_k است

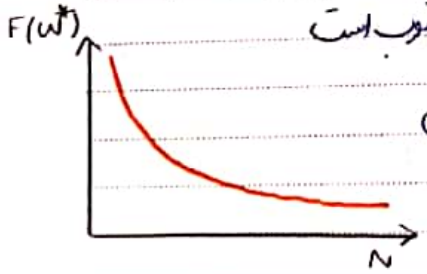
مقدار مطلوب یا T_k برابر $f(X_k)$ ← فرقی که با مسائل قبلی دارد این است که به جای T_k، $f(X_k)$ میگذاریم

فرض کنید تعداد زیادی از این نمونه‌ها را داریم و آن‌ها را اعمال می‌کنیم. خطای کل:

$$F(W) = \lim_{N\to\infty} \frac{1}{N} \sum_{k=1}^{N} f_k(X_k, W) \geq 0$$

این میانگین یا مثبت است یا صفر است. مطلوب با اینه که خطا به عدد خیلی کوچکی شود.

لازم است شبکه را ارزیابی کنیم تا بفهمیم تخمینی که این تابع به ما می‌دهد چقدر خوب است



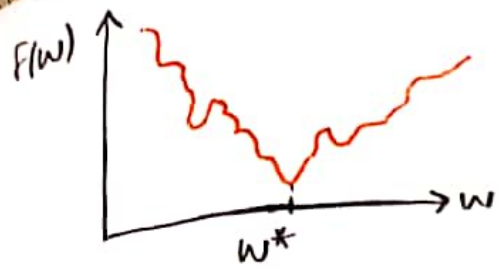
برای اینکار فرض کنید یک نمونه به شبکه می‌دهیم، اگر درست جواب بدهد دقت ۱۰۰٪ و اگر غلط باشد ۰٪ است. برای ۲ نمونه، اگر هر ۲ درست باشد ۱۰۰٪، هر دو غلط ۰٪، یکی غلط یکی درست ۵۰٪ و ...

نتیجه ارزیابی با داشتن تعداد نمونه‌های مختلف، متفاوت می‌شود. نمی‌توان تعداد نمونه‌های داده برداشت، باید بر یک اساسی باشد.

با توجه به نمودار، هرچقدر تعداد نمونه‌ها را افزایش می‌دهیم خطای کل کاهش پیدا می‌کند.

کاهش پیدا می‌کند تا اینکه در جایی ثابت می‌شود، در این جا نشان می‌دهد که تعداد نمونه‌های تست کافی است که در این حد باشد. بنابراین برای اینکه بفهمم مجموعه تستم چقدر خوب ارزیابی می‌کند و آیا کافی است یا نه، عیام یه مجموعه اختیار می‌کنم و بعد به مرور اندازه‌اش را افزایش می‌دهم. در هر بار افزایش سوال می‌کنم آیا مقدار خطا کمتر شد؟ اگر کمتر شد یعنی مقدار قبلی کافی نبوده و باید تعداد داده‌های تست را افزایش بدهیم. هرگاه دیگر تغییر آنچنانی در خطا ایجاد نشد می‌گوییم مقدار داده‌های تست کافی است.

Scanned with CamScanner



در این مسئله هم مثل دسته‌بندی می‌توانیم برای شبکه در حین آموزش یک سطح در نظر بگیریم، سطح خطا با ۱۴۰۰
که در فضای وزن‌ها قرار می‌گیرد و دنبال این هستیم که با حرکت روی این سطح به min خطا برسیم

قضیه: برای هر $\varepsilon > 0$ و هر تابع که L_2 باشد (یعنی به صورت مربع روی مکعب قابل انتگرال‌گیری است)
$f: [0,1]^n \to R^m$، آن‌گاه یک شبکه‌ی سه‌لایه‌ی feedforward وجود دارد که می‌تواند f را با
متوسط خطای (mse accuracy) ε تخمین بزند.

نکته: ε هر مقدار دلخواهی می‌تواند باشد یعنی تا حد دلخواهی می‌توان دقت را به ۱۰۰ درصد نزدیک کرد
← اینکه بتوان با شبکه ۳ لایه پیاده‌سازی کرد می‌تواند سخت باشد (آموزش سخت است یعنی
همگرا شدنش و رسیدن آن به خطای مطلوب، به راحتی صورت نمی‌گیرد)
برای اینکه راحت‌تر باشیم می‌توانیم شبکه ۴ لایه یا ۵ لایه استفاده کنیم

Scanned with CamScanner

عوامل تاثیرگذار روی عملکرد شبکه FF

① تعیین وزن های اولیه

اگر به خوبی انتخاب شود هم سریع همگرا می شویم و هم به min سراسری می رسیم. اما متاسفانه قاعده ای به دست نیاوردن، فقط یک سری راهنماها داریم. مثلا اینکه وزن های اولیه نباید خیلی بزرگ باشند. اگر خیلی بزرگ باشند، نورون ها، به خصوص نورون هایی که در انتهای شبکه و نزدیک به خروجی هستند، عملشان در نواحی که تابع فعال سازی نسبتاً ثابت است قرار می گیرد در نتیجه مشتق بسیار کوچک است و یادگیری کم است اصلاح وزن ها توسط ضریب δ صورت می گیرد که در روابط دیدیم δ با f' مربوط است. f' به صورت یک عامل ضرب شونده در یک تابع دیگری اثرگذار روی δ است.

$$w_{ij}^{new(l)} = w_{ij}^{old(l)} + \eta \delta_{pj}^{(l)} y_{pi}^{(l-1)}$$

توصیه دیگر اینکه وزن های اولیه خیلی کوچک هم نباشند چون در این صورت net Input به نورون ها کوچک می شود و خروجی ها به صفر نزدیک می شود و باز هم موجب کاهش سرعت همگرایی می شود.

$$\delta_{pj} = (t_{pj} - y_{pj}) f_j'(I_{pj}) \qquad \delta_{pj}^{(l)} = f_j'(I_{pj}^{(l)}) \cdot \sum_k \delta_{pk}^{(l+1)} \cdot w_{jk}^{(l+1)}$$

معمولا وزن ها را در فاصله [-0.5,0.5] یا [1 و -1] به صورت رندوم انتخاب می کنند.

اگر هنگام آموزش شبکه خطای ما به جایی رسید که دیگر کاهش پیدا نکرد (که به آن یک plateau گفته می شود) توصیه این است که آموزش را ادامه ندهید و بهتر است آموزش را از نو شروع کنید با مقادیر اولیه جدید برای وزن ها

چون تغییرات نامطلوبی در وزن ها پیش می آید

② تابع فعال سازی

بهتر هست توابع Antisymmetric (فرد) داشته باشیم. یعنی خروجی هم مقادیر مثبت و هم مقادیر منفی داشته باشد و نسبت به محور قائم متقارن باشد. وقتی اینطور باشد یادگیری سریعتر صورت می گیرد. مثل این است که در

Scanned with CamScanner

یک جستجو هم داریم تشویق می کنیم و هم تنبیه. وقتی مقدار منفی نداشته باشد (nonsymmetric) مثل
این است که فقط داریم تشویق می کنیم و تنبیه نمی کنیم.

← مشتق تابع f برای تابع sigmoid ← عاملی است که روی میزان اصلاح وزن ها تاثیرگذار است

$f'(z) = \alpha f(z)\left[1 - f(z)\right]$ $\qquad f(z) = \dfrac{1}{1+e^{-\alpha z}}$: α مشتق تابع فعال سازی است

α در f' تاثیرگذار است پس α در δ تاثیرگذار است، بنابراین در میزان اصلاح تاثیرگذار است.

α نباید خیلی بزرگ یا خیلی کوچک باشد. α در واقع سرعت بالا رفتن منحنی Sigmoid را کنترل می کند
اینکه نزدیکتر می شود به تابع پله یا دورتر (تدریجی تر) می شود.

تصحیح وزن ها با عبارت $\eta\, \delta_{pj}^{(l)}\, y_{pi}^{(l-1)}$ صورت می گیرد که δ خودش یک عامل ضرب شونده ی f' دارد
f' هم عامل ضرب شونده ی α دارد. پس در نهایت α و η که هر دو پارامترهای کنترلی اند در هم ضرب
می شوند. پیشنهاد شد که بیاییم α=1 بگیریم و بعد کنترل را با η اعمال کنیم.

۳) ارائه ی داده ها (Data Representation)

خیلی اوقات به خصوص در مسائل پیچیده این سوال مطرح است که ورودی ها را چه چیزی در نظر بگیریم.
یعنی چه داده هایی به شبکه بدهیم؟ نکته دیگر این است که چگونه این داده ها را به شبکه بدهیم؟
این انتخاب تا حدی مهم است که حتی می تواند تعیین کند مسئله قابل حل باشد یا خیر.
مثلا یک انتخاب این است که داده ها را پیوسته بدهیم یا گسسته سازی انجام بشود. (مثال درجه حرارت)
وقتی توابع فعال سازی را دوقطبی در نظر می گیریم داده ها هم باید دوقطبی (Bipolar) باشند. داده ها
باید 1- و 1 باشند و تابع فعال سازی هم بین 1- و 1 تغییر می کند.

Scanned with CamScanner

۴) مجموعه‌های Training و Validation

دو حالت را در نظر بگیرید: ۱) مسئله ما به گونه‌ای است که به وفور برای آن داده داریم و هیچ مشکلی در تهیه داده زیاد نداریم. در این حالت ۳ مجموعه Train، Validation و Test درست می‌کنیم و سعی می‌کنیم انواع داده‌ها را در هر ۳ مجموعه قرار بدهیم.

۲) نمی‌توانیم نمونه‌های زیادی داشته باشیم. در مسائل عملی معمولا این حالت پیش می‌آید و ما هم از نظر تعداد داده و هم تنوع داده، محدودیت داریم. مثلا نمی‌توانیم یک سیستم را خراب کنیم برای اینکه داده‌های مربوط به خراب شدن را داشته باشیم و در دنیای واقعی یک سری از داده‌ها را اصلا نمی‌توان تهیه کرد.

در این حالت تمام داده‌ها را برای آموزش به کار می‌بریم. در حین آموزش به یک جایی می‌رسیم که کاهش خطا کُند می‌شود، در این جا آموزش را قطع می‌کنیم. یعنی به جای استفاده از validation set، از افت شدن منحنی تابع خطای مجموعه آموزشی استفاده می‌کنیم

پیشنهاد دیگر این است که یک نمونه را کنار بگذاریم و با n-1 نمونه‌ی دیگر شبکه را آموزش بدهیم و با آن 1 نمونه تست کنیم. بعد دوباره این کار را برای همه نمونه‌ها انجام بده (n بار). هر بار خطا را محاسبه کن بعد میانگین مربع این خطا را حساب کن و به عنوان خطای generalization شبکه اعلام کن. به این روش leave one out گفته می‌شود

سومین پیشنهاد: از روی داده‌هایی که داریم یک سری داده به صورت دستی و مصنوعی ایجاد کنیم. مثلا به شکل‌های مختلف به داده‌هایی که داری نویز اضافه کن تا داده جدید به دست آید. یا تغییراتی در داده ایجاد کن مثلا در مورد داده تصویری می‌توان با چرخاندن آن یک تصویر جدید ایجاد کرد. یا مثلا scale کردن (کوچک و بزرگ کردن شیء در تصویر) یا با تغییر شدت روشنایی تصویر. به این روش‌ها data augmentation گفته می‌شود.

Scanned with CamScanner

۵) تعداد نمونه‌های آموزشی

Baum و Haussler نشان دادند که در یک شبکه FF تعداد وزن‌ها (یا اتصالاتی) که در آن است اگر N_W باشد و خطای قابل قبول برای تست E باشد آن‌گاه باید شبکه را با تعداد $L = \frac{N_W}{E}$ نمونه آن قدر آموزش بدهید که خطا به کمتر از $\frac{E}{2}$ برسد. در این صورت می‌توانیم مطمئن باشیم که خطا برای داده‌های تست از E بیشتر نمی‌شود.

مثال:

$N_W = 80$, $E = 10\%$ $\Rightarrow$ $L = 80/0.1 = 800$

پس ۸۰۰ تا داده باید انتخاب شود و آن قدر آموزش داده شود که خطا کمتر از ۵ درصد شود

۶) چقدر آموزش بدهیم (تعداد epoch)

همانطور که گفتیم در طراحی‌مان باید بین Memorization و Generalization تعادل برقرار کنیم. (قدرت حفظ کردن و قدرت تعمیم دادن شبکه).

معنی‌اش این است که نباید زیاد آموزش بدهیم چون به سمت حفظ کردن می‌رود.

نباید دنبال این باشیم که حتما خطا را min کنیم. ترجیح می‌دهیم خطای جزئی را بپذیریم در ازای اینکه Generalization بهتری داشته باشیم. ۳ معیار برای توقف ذکر شده:

۱ اندازه گرادیان را در هر مرحله به دست آوریم. در min واقعی مشتق صفر است. وقتی گرادیان نزدیک به صفر شود ما به پاسخ نزدیکیم. یک مقدار آستانه در نظر بگیرید، اگر گرادیان از آن کوچکتر شد متوقف شوید

۲ نرخ تغییرات متوسط مربع خطا برای epoch اگر از یک حدی کمتر شد متوقف شویم. پس در این جا داریم از معیار خطا برای توقف آموزش استفاده می‌کنیم

Scanned with CamScanner

۳. Early Stopping ← وقتی خطا برای مجموعه آموزشی در حال ثابت شدن است می‌توانید آموزش را متوقف کنید

جلسه دهم ـ ۱۴۰۰/۸/۴

۷) نرخ یادگیری (Learning Rate)

تنظیم نرخ یادگیری وابسته به مسئله‌ای است که می‌خواهیم حل کنیم. یعنی این ضریب η (eta) که روی سرعت همگرایی تاثیرگذار است و الگوریتم نسبت به مقدار آن حساس است، به مسئله وابسته است.

علاکاری نمی‌توانیم بکنیم جز اینکه از طریق تجربی آن را پیدا کنیم. نکته مهمی که در مورد آن می‌دانیم این است که مقادیر کوچک η تضمین می‌کنند که پایین رفتن در امتداد گرادیان صورت بگیرد. در گزارشات انجام شده مقدار مناسب برای η در بازه ۰/۰۰۱ تا ۱۰ گفته شده است.

۸) معماری شبکه (Network Architecture)

قدم اول و مهم و اصلی است.

تعداد نورون‌های لایه ورودی باید معلوم باشد (سایز ورودی را داریم در نتیجه تعداد نورون‌های لایه اول را داریم)

چون می‌دانیم چه پاسخی از مسئله انتظار داریم تعداد نورون‌های لایه خروجی را هم می‌دانیم که البته در مورد لایه خروجی قبلا گفتیم که m تا یا $\log m$ تا خروجی می‌توانیم استفاده کنیم

در مورد لایه‌های مخفی: باید از مقادیر نه چندان بزرگ برای لایه‌های مخفی شروع کنیم و جلو برویم. یعنی مثلا یک لایه مخفی با یک تعداد نورون که فکر می‌کنید برای این مسئله کفایت می‌کند قرار دهید، بعد بررسی می‌کنید. تغییر دهید و این کار تکرار می‌شود

در واقع طراحی شبکه‌های پیش رو، یک بار نیست. یک فرآیندی است که به دفعات با مشخصات مختلف برای لایه‌های مخفی تکرار می‌شود که شما بتوانید به یک معماری بهتر دسترسی پیدا کنید.

Scanned with CamScanner

۱۴۰۰

مثلا ممکنه شبکه‌ای طراحی کنید که ۸۸ درصد دقت دارد و شبکه‌ی دیگری را بررسی نکنید. سوالی که این جا جواب داده نشده این است که آیا معماری بهتر از این وجود نداشت؟ این مورد باید بررسی شود. هنگام طراحی باید معماری های مختلف را امتحان کنید و از بین آن ها بهترین شان را انتخاب کنید.

شیب های مختلف شبکه های FF

مشکلات این شبکه: ① گیر بودن آموزش (گیر بودن همگرایی در فاز آموزش)

② همگرا شدن به min محلی

برای حل این دو مشکل کارهای زیادی صورت گرفته است

① یک کار این بود که قانون به هنگام درآوردن وزن ها را تغییر دهند (یعنی قانون یادگیری را عوض کنند)

② راه دیگر: با تغییر تابع فعال شدن نورون ها، بهبود ایجاد کنیم

③ راه دیگر: روش های ترکیبی هوش محاسباتی

دسته اول: تغییر قانون اصلاح وزن ها (BP)

۱. روش های درجه ۱

ویژگی متدهای درجه ۱ این است که همه بر اساس گرادیان هستند. یعنی بر اساس شیب حرکت روی سطح خطا هستند. چند تا از روش هایی که پیشنهاد شده در این دسته:

۱) یک جمله به قانون یادگیری اضافه می شود که به آن جمله momentum گفته می شود و در اصلاح وزن ها momentum ایجاد می کند

۲) روش دیگر: به هنگام درآوردن دسته ای

۳) روش دیگر: روش هایی هستند که نرخ یادگیری را تطبیقی می کنند. یعنی نرخ یادگیری η که اندازه قدم حرکت به سمت پاسخ را برای ما مشخص می کند با شرایطی که داریم تطبیقی می کند.

قیام خونین ۱۵ خرداد (۱۳۴۲ هـ ش) (تعطیل) - روز جهانی محیط زیست

Scanned with CamScanner

۴) روش آخر: روش تندترین شیب (steepest descent)

A.1 اضافه کردن جمله‌ی Momentum

خیلی اوقات ممکنه در بین داده‌ها، داده‌های پرت داشته باشیم که مقدارشان خیلی با داده‌های دیگر تفاوت دارند. وجود این داده‌ها به آموزش صدمه می‌زند و توصیه می‌شود قبل از آموزش این داده‌ها حذف شوند.

جمله‌ی Momentum سعی دارد اگر به جهت دلیلی داده پرت در داده‌های آموزشی وجود دارد، صدمه آن را کم کند

$$w_{ij}^{(l)}(t+1) = w_{ij}^{(l)}(t) + \alpha \delta_{pj}^{(l)} y_{pj}^{(l-1)} + \mu \left[w_{ij}^{(l)}(t) - w_{ij}^{(l)}(t-1) \right] \qquad 0<\mu<1$$

تغییر وزن این دفعه با ضریبی از تغییر وزن دفعه قبل جمع می‌شود. یعنی اگر مثلا در امتداد یک بردار در جهت گرادیان در حال حرکت به سمت پایین هستیم و یک دفعه یک داده پرت یک جهت انحرافی به این بردار می‌دهد. حالا اگر این بردار منحرف شده با ضریبی از تغییر وزن دفعه‌ی قبلی جمع شود، حاصلش می‌شود برداری که کمتر انحراف دارد و بیشتر متمایل هست به سمت جهت گرادیان تا جهتی که این داده پرت پیشنهاد داده است.

بنابراین حرکت در این حالت در جهت گرادیان نقطه نخواهد بود بلکه در جهت گرادیان قبلی به اضافه جهت این نقطه خواهد بود.

ضریب μ را معمولا نزدیک ۱ می‌گیرند و α هم مقدار کوچکی است

وقتی از Momentum استفاده می‌کنیم این احتمال وجود دارد که اصلاح وزن‌ها جاهایی که داده پرت نداریم بزرگ می‌شود (در نتیجه با قدم‌های بزرگ‌تری حرکت می‌کنیم و سرعت همگرایی بالا می‌رود. هم چنین وقتی قدم‌ها بزرگ است این احتمال وجود دارد که از برخی min های محلی عبور کنیم. پس اثرات مثبتی از جهت‌های دیگر هم دارد

Scanned with CamScanner

مثال از انتخاب مقادیر η و μ

بهتر است η را کوچک و μ را نزدیک ۱ انتخاب کنیم. با کوچک شدن η بهبود ایجاد می‌شود ولی از یک جدی وقتی کوچکتر شد شروع به بدتر شدن می‌کند. چون یادگیری ضعیف می‌شود

A.2 به روز آوردن دسته‌ای وزن‌ها

۱) pattern mode : هر تک نمونه‌ای که اعمال می‌شود، همه‌ی وزن‌ها را اصلاح کنیم

۲) یک راه دیگر این است که نمونه اعمال شود، خطا را حساب کنیم اما وزن‌ها را آپدیت نکنیم. این کار را به تعداد نمونه‌های درون batch تکرار می‌کنیم. بعد از آن از خطاها میانگین می‌گیریم و اصلاح وزن‌ها بر حسب میانگین خطا صورت گیرد

یعنی برای هر sample ای که داریم خطا : $e_j(p) = t_j(p) - y_j(p)$ را ذخیره می‌کنیم

و بعد میانگین می‌گیریم :

$$E_{avg} = \frac{1}{2L}\sum_{p=1}^{L}\sum_{j\in C} e_j^2(p)$$

هدف این است که سرعت را بالا ببریم.

با انجام این کار، تعداد زیادی به هنگام در آوردن وزن انجام نمی‌شود و در نتیجه محاسبات کاهش پیدا می‌کند

یک batch می‌تواند شامل همه‌ی نمونه‌ها باشد یعنی مساوی یک epoch باشد. یعنی یک دور تمام نمونه‌ها را اعمال کنیم، خطا را بیابیم و آخر سر وزن‌ها را آپدیت کنیم

A.3 تطبیق کردن نرخ یادگیری

دو دیدگاه وجود دارد: ① با یک η بزرگ شروع کنیم و آن را تدریجا کاهش بدهیم به طوری که وقتی به ناحیه همگرایی می‌رسیم η کوچک شده باشد.

در ابتدا اگر قدم‌ها بزرگ باشد خوب است چون معمولا از کجا سرا سری دور هستیم، با قدم‌های بزرگ به سمت نقطه min می‌رویم و ممکنه از روی min های محلی هم بپریم که سرعت همگرایی بالا می‌رود اما وقتی به ناحیه پاسخ نزدیک می‌شویم η بزرگ نامناسب است به همین دلیل می‌گوییم η را کاهش بدهیم به صورت تدریجی.

Scanned with CamScanner

۲) Delta_bar_Delta : برای هر وزن یک نرخ یادگیری مختص خودش دارد و این نرخ یادگیری در حین آموزش یادگیرنده می شود. بنابراین شبکه در حین آموزش این مورد را هم یاد می گیرد که نرخ یادگیری برای هر نورون چقدر باشد.

A.4 تندترین شیب (steepest Descent)

روش ترکیبی است: در روش delta_bar_delta نرخ یادگیری برای هر وزن مختص خودش تعیین می شود و در BP استاندارد گفتیم η یک عدد ثابت است که برای تمام نورون ها در کل آموزش ثابت است. در steepest descent کاری که می کنیم بین این دو راه است. نرخ یادگیری را ε می نامیم و برای تمام وزن ها مثل BP استاندارد، یکسان است اما در حین آموزش مقدارش تعیین می شود

روش آموزش: یک مقدار اولیه برای ε نسبت می دهید

در هر مرحله نرخ یادگیری را ضرب در 2 کن، وزن ها را محاسبه کن و خطا را با این وزن های جدید محاسبه کن. اگر خطا نسبت به دفعه قبل کاهش پیدا کرده این تغییر وزن ها خوب است

اگر کاهش پیدا نکرده این اصلاح وزن ها و تغییر نرخ یادگیری خوب نیست، مقادیر وزن ها را به حالت قبلی برگردان و نرخ یادگیری را نصف کن. مجددا MSE را محاسبه کن اگر MSE کاهش پیدا نمی کند، دوباره ε را نصف کن. به صورت مکرر این کار انجام می شود. انقدر ε نصف می شود تا به حالتی برسیم که MSE کاهش پیدا کند نسبت به قبل. وقتی این اتفاق افتاد نتیجه می گیریم این ε و وزن هایی که با آن اصلاح شده اند خوب هستند، آن ها را نگه می داریم و به قدم بعدی می رویم.

بنابراین به این ترتیب مقدار ε مکررا کم و زیاد می شود تا به یک مقدار مناسب در این قدم برای آن برسیم و در این قدم خطا در حال کاهش است و وزن ها را اصلاح می کنیم و به دور بعدی آموزش می رویم و این مراحل تکرار می شود. سوالی که مطرح می شود این است که آموزش را چه زمانی متوقف کنیم؟

Scanned with CamScanner

شروط توقف: ① تغییر خطای این بار نسبت به دفعه قبل اگر از مقدار مشخصی کمتر شد متوقف شویم:

$$\frac{E(w_m) - E(w_{m+1})}{E(w_m)} \leq E_{min}$$

② ε اگر از مقدار آستانه کمتر شد:

$$\varepsilon \leq \varepsilon_{min}$$

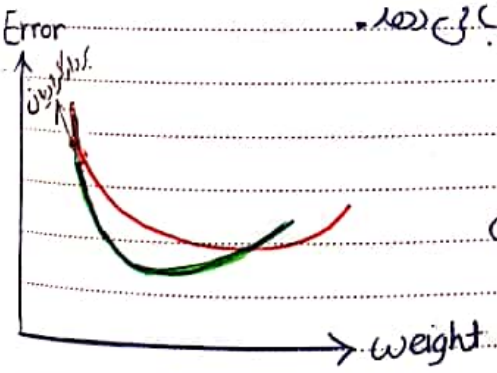
مقایسه روش steepest Descent با delta-bar-delta و BP نشان داده که steepest descent بهتر عمل می‌کند

B. روش‌های درجه 2

به جای اینکه پایین رفتن روی سطح را براساس شیب انجام بدهیم، براساس انحنا پیش برویم که مشتق درجه ۲ انحنای سطح خطا در نقطه را به ما می‌دهد

۳ روش وجود دارد:
- QuickProp
- Gauss-Newton Method
- The Levenberg-Marquardt Method

در عمل دیده شده که اگر براساس انحنا عمل کنیم بهتر از گرادیان جواب می‌دهد.



در شکل روبه‌رو، منحنی‌ها تابع خطا را نشان می‌دهند. منحنی قرمز رنگ برای حالتی است که از گرادیان استفاده می‌کنیم و منحنی سبز رنگ حالتی است که از انحنا استفاده می‌کنیم.

دو منحنی در ابتدا روی هم هستند و بردار گرادیان برای هر دوی آن‌ها یک جهت را نشان می‌دهد اما اگر از انحنا استفاده کنیم منحنی خط سبز رنگ انحنای بیشتری در این نقطه دارد. بنابراین انحنایی که به دست می‌آوریم ما را به سمت منحنی سبز هدایت می‌کند و زودتر به پایین می‌رسیم و خطا حداقل می‌شود.

بنابراین یک جاهایی ممکن است گرادیان جهتی را نشان دهد که احتمالا بهترین جهت برای آن نقطه نباشد و انحنا جهت بهتری را مشخص می‌کند.

Scanned with CamScanner

به طور کلی در روش های درجه ۲ فاصله ی تا نقطه ی min یا نقطه بهینه، در فضای وزن ها، با رابطه ی زیر تخمین زده می شود: مشتق اول خطا تقسیم بر مشتق دوم خطا ←

$$\Delta w = -\frac{\partial E / \partial w}{\partial^2 E / \partial w^2}$$

خطا برای شبکه MLP با N تا نمونه آموزشی:

$$E(w) = \frac{1}{N}\sum_{i=1}^{N}[t_i - z_i]^2 = \frac{1}{N}\sum_{i=1}^{N}[t_i - f(w, x_i)]^2$$

در روش های درجه ۲ با توجه به نکته ای که در مورد فاصله Δw گفتیم، یک قاعده کلی می توانیم برای تغییر وزن ها به کار ببریم که به این شکل است:

$$\boxed{w_m = w_{m-1} - \varepsilon R d_m}$$

که در آن ε نرخ یادگیری، R یک تابع از مشتق دوم است و d_m مجموع مشتقات خطا برای یک batch یا epoch است یعنی d_m مرتبط با مشتق اول و R مرتبط با مشتق دوم است و اصلاح بایاس مزبور از Δw دارد صورت می گیرد.

B.2 روش Gauss_Newton

$$\boxed{d^s = \frac{\partial^2 E}{\partial w^2}}$$ ← مشتق دوم E نسبت به w

در این روش R ای که در معادله کلی داشتیم برابر $1/d^s$ است. $R = 1/d^s$

بنابراین قاعده آموزش اینگونه است:

$$w_m = w_{m-1} - \varepsilon R d_m \Rightarrow \boxed{\Delta w_m = -\varepsilon \frac{d_m}{d^s_m}}$$

روز صنایع دستی

در هر Epoch ابتدا ε را برابر ۱ قرار می دهید و اگر MSE با این ε کوچکتر از آخرین MSE که داشتیم باشد، این ε را می پذیریم. اگر کوچکتر نیست ε را تقسیم بر ۲ کنید و وزن ها را به مقدار قبلی بازگردانید و دوباره بررسی کنید که MSE کوچکتر شد یا نه.

یعنی همان فرآیندی که در steepest Descent داشتیم در این جا در این قسمت تعیین ε، مشابهش به کار گرفته شده است.

Scanned with CamScanner

وقتی به حالتی برسیم که خطا کاهش پیدا نکرده باشد، وزن‌ها را به عنوان وزن‌های جدید قبول می‌کنیم همان طور که گفتیم ε را ابتدا ۱ می‌گذاریم و آن فرآیند را تکرار می‌کنیم که باعث می‌شود نرخ یادگیری در حین آموزش خودش تعیین شود و نیاز نداریم نرخ آموزش را تعیین کنیم.

شرط توقف مشابه قبل:

$$\boxed{\begin{array}{ll} ① & \dfrac{E(w_m) - E(w_{m+1})}{E(w_m)} \leq E_{min} \\ ② & \varepsilon \leq \varepsilon_{min} \end{array}}$$

تا این جا گفتیم که برای تک وزن کار را به این شکل می‌توانیم انجام بدهیم اما برای حالتی که وزن‌های مختلف را می‌خواهیم داشته باشیم یک راه این است که همه‌ی مشتقات درجه ۲ را در یک ماتریس قرار بدهیم و سعی کنیم به صورت عملیات ماتریسی یکباره برای همه‌ی وزن‌ها این کار را انجام بدهیم.

ماتریسی که مشتقات درجه ۲ را در خودش دارد ماتریس Hessian نام دارد.

چون مشتق درجه ۲ را در مخرج کسر داشتیم باید از معکوس ماتریس H باید استفاده کنیم: $R = H^{-1}$

برای یک شبکه با تابع خطای MSE به این صورت است:

$$\boxed{E = \frac{1}{N}\sum_{i=1}^{N}\left(t_i - z_i(x_i, w_i)\right)^2}$$

درایه‌های ماتریس H اینگونه به دست می‌آیند:

$$\boxed{H = \frac{\partial^2 E}{\partial w_i \partial w_j} = \frac{2}{N}\sum_{i=1}^{N}\left\{(t_i - z_i)\left(\frac{-\partial^2 z_i}{\partial w_i \partial w_j}\right) + \left(-\frac{\partial z_i}{\partial w_i}\right)\left(-\frac{\partial z_i}{\partial w_j}\right)\right\}}$$

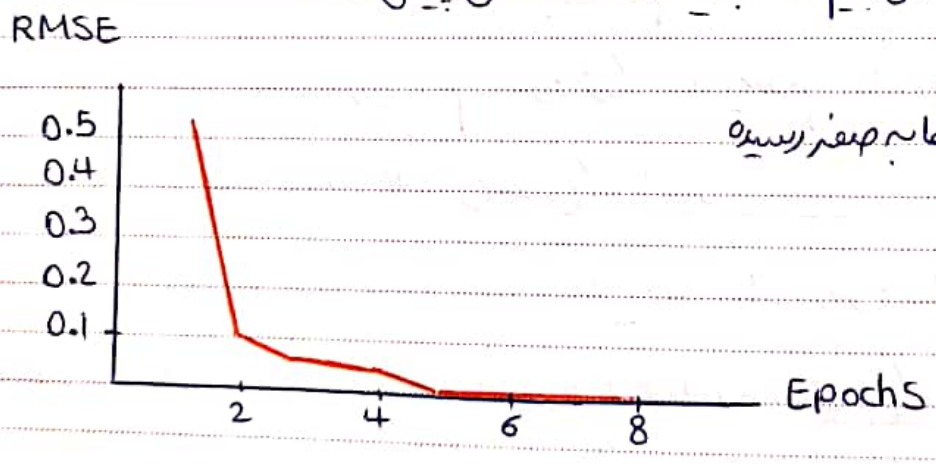
Scanned with CamScanner

رابطه‌ها از مشتق گرفتن از رابطه‌ی E حاصل می‌شوند.

نکته‌ای که مشاهده می‌شود این است که جمله‌ی اول به گونه‌ای حساس است که می‌تواند از نظر محاسباتی ناپایداری به وجود بیاورد و در محاسبه‌ی H^{-1} و معکوس کردن H دچار مشکل شویم. به همین دلیل از جمله‌ی اول که مشتق درجه ۲ است صرف نظر می‌شود و H را به این صورت تقریب می‌زنیم:

$$\boxed{H = \frac{\partial^2 E}{\partial w_i \partial w_j} \approx \frac{2}{N} \sum_{i=1}^{N} \left\{ \left(-\frac{\partial z_i}{\partial w_i}\right)\left(-\frac{\partial z_i}{\partial w_i}\right) \right\}}$$

مثال: وقتی این روش را به کار می‌بریم خطا به این صورت کاهش پیدا می‌کند:



حدودا در 5 تا epoch خطا به صفر رسیده

B.3 روش Levenberg_Marquardt (LM)

به جای اینکه E را در نظر بگیریم و هر بار آن را تغییر بدهیم یک جمله‌ی دیگر اضافه می‌کنیم. E را ۱ در نظر می‌گیرد. جمله‌ی e^{λ} را به مشتق اضافه می‌کنیم:

$$\boxed{R = \frac{1}{d_m^s + e^{\lambda}}}$$

قاعده‌ی Δw_m برای یک وزن به این صورت در می‌آید $\rightarrow$

$$\boxed{\Delta w_m = -\frac{d_m}{d_m^s + e^{\lambda}}}$$

Scanned with CamScanner

ماتریس Hessian به این صورت در می آید:

$$\boxed{H' = H + e^{\lambda} I}$$

R در حالت ماتریسی:

$$\boxed{R = \frac{1}{(H + e^{\lambda} I)}}$$

Δw_m در حالت ماتریسی ↓

$$\boxed{\Delta w_m = -\frac{d_m}{(H_m + e^{\lambda} I)}}$$

در این حالت باید λ را در فرآیند آموزش تعیین کنیم.

یک مقدار اولیه برای λ در نظر می گیریم و همین طور که جلو می رویم ابتدا به اندازه $\Delta\lambda$ آن را کاهش بدهید و بررسی کنید که MSE کاهش پیدا می کند یا خیر.

اگر کاهش پیدا کرد این حرکت خوب بوده و اصلاح وزن ها را به این شکل قبول می کنیم. اگر MSE کاهش پیدا نکرده، کاهش λ خوب نبوده و $\Delta\lambda$ را به λ اضافه می کنیم. مجددا MSE را محاسبه می کنیم و بررسی می کنیم که قابل قبول هست یا نه.

هرچقدر λ بزرگتر شود چون در مخرج است اصلاح کمتر می شود.

در روش LM نیمه های steepest Descent و Gauss_Newton باهم ترکیب شده اند.

شرط خاتمه به این صورت است:

$$\boxed{\lambda > 10\,\Delta\lambda + Max[H]}$$

Max[ ] ← بزرگترین مقدار ویژه ی ماتریس H

Scanned with CamScanner

شرط خاتمه‌ی دیگر:

$$\frac{E(w_m) - E(w_{m+1})}{E(w_m)} < E_{min}$$

شکلی از مقایسه روش BP و وزن های مختلف آن در اسلاید آمده، به ترتیب عملکرد:

BP < delta_bar_delta < steepest Descent < Gauss_Newton < LM

مشاهده می‌شود که این دو روش درجه ۲ بهتر از همه‌ی روش های درجه ۱ عمل می‌کنند و LM از همه بهتر.

سوالی که مطرح می‌شود این است که روش های درجه ۲ که اینقدر خوب عمل می‌کنند چه هزینه‌ای دارند؟

هزینه‌شان محاسبات سنگینی است که دارند: ماتریس Hessian و معکوس کردن آن و ...

II تغییر تابع فعال سازی نورون ها

پیشنهاد دیگری که دارند عوض کردن تابع فعال سازی نورون ها بود. اینکه سعی کنیم تابع anti_symmetric بکاربریم. یعنی این خاصیت را داشته باشند: $f(-x) = -f(x)$

مثال: Binary and Bipolar Sigmoid، تانژانت هایپربولیک (Tanh(x))، arctan(x)، تابع لگاریتمی Log(1+x)، تابع گاوسی $\exp(-x^2)$

بخشی از مشکلاتی که شبکه‌های FF دارند برمی‌گردد به تابع فعال سازی و اگر آن را عوض کنیم امکان بهبود هست

III تکنیک های ترکیبی

۱) مثلا استفاده از تکنیک های پردازش تکاملی: الگوریتم ژنتیک می‌تونه برای بهینه کردن معماری و پارامترهای شبکه به کار بره، یا Genetic programming را به کار ببریم، روش هایی مثل PSO (هوش جمعی).

نگاه این روش ها این است که طراحی به سمت طراحی بهینه برود و حتی باید هزینه‌اش را هم بدهیم، مثلا کند بودن

۲) استفاده از مفاهیم فازی در شبکه ها (neuro fazy) ← علت پدیده ۴، ورودی ۴ یا هر ورودی آن ها اعداد فازی باشند. نتیجه خوب داد

۳) هم فازی و هم الگوریتم ژنتیک را بیاوریم. حسن فازی این است که جزئیات آزار دهنده را با رفتن به یک فضای دیگر حذف می‌کند.

Scanned with CamScanner

جلسه یازدهم - ۱۴۰۰/۸/۹

pattern clustering Neural Network

مسئله‌ی ۳: مسئله‌ی pattern clustering (خوشه‌بندی)

خوشه‌بندی ساختارهایی را که در داده‌های ورودی وجود دارد کشف می‌کند و گروه‌ها (خوشه‌هایی) تشکیل می‌دهد که سازمان‌دهی آن‌ها معنی‌دار است.

داده‌های ورودی می‌توانند داده‌های خام باشند و یا ویژگی‌هایی باشند که از داده‌های خام استخراج شده‌اند و به هر کدام از آن‌ها یک pattern گفته می‌شود. بنابراین کاری که خوشه‌بندی انجام می‌دهد این است که پترن‌ها یا الگوها را برای ما گروه‌بندی می‌کند به گروه‌هایی (خوشه‌هایی) که معنی‌دار هستند.

هدف این کار این است که بتوانیم در بین الگوها تشابه‌ها را بیابیم و تفاوت‌ها را بیابیم و بتوانیم براساس آن‌ها نتیجه‌گیری‌های مفیدی برای مسئله‌مان کنیم. (هدف خوشه‌بندی)

برای خوشه‌بندی به ۳ چیز نیاز داریم:

1. سنجشگر نزدیکی (proximity Measure)
2. معیار خوشه‌بندی (clustering Criterion)
3. الگوریتم خوشه‌بندی (clustering Algorithm)

معیار خوشه‌بندی ← مبنایی است که خوشه‌بندی براساس آن انجام می‌شود

مثال: فرض کنید می‌خواهیم تعدادی حیوان را خوشه‌بندی کنیم. سوالی که مطرح می‌شود این است که برچه اساسی می‌خواهیم خوشه‌بندی را انجام بدهیم؟ مثلا: نحوه تولید مثل، پشم دارند یا نه؟، معیار خوشه‌بندی روی تعداد دسته‌ها هم تاثیرگذار است

این دسته‌بندی در واقع براساس سنجشگری که خواص این پترن‌ها را اندازه‌گیری می‌کند و نزدیک بودنشان به یکدیگر را بررسی می‌کند، انجام می‌گیرد. (proximity Measure)

Scanned with CamScanner

تعریف رسمی: فرض کنید $X = \{x_1, x_2, \dots, x_N\}$ این x_i ها الگوهای ما هستند.

m-خوشه‌بندی برای X اینگونه تعریف می‌شود: X را به m تا زیرمجموعه $C_1, C_2, \dots, C_m$ تقسیم کنیم

به طوری که:

1) $C_i \neq \emptyset \quad i = 1, 2, \dots, m$ — هر C_i حداقل یک عضو داشته باشد

2) $\bigcup_{i=1}^{m} C_i = X$ — اجتماع C_i ها بشود X

3) $C_i \cap C_j = \emptyset \quad i \neq j, \ i, j = 1, 2, \dots, m$ — هر پترن فقط در یک خوشه می‌تواند باشد

در هر کدام از این خوشه‌ها الگوهایی که داریم به هم شبیه‌تر هستند نسبت به الگوهایی که در خوشه‌های دیگرند.
یعنی الگوهای خوشه‌های مختلف با هم متفاوت اند و الگوهای درون یک خوشه کم شبیه هستند.

موارد استفاده از خوشه‌بندی:

① در جاهایی که حجم داده‌ها زیاد است، می‌توان با خوشه‌بندی به جای هر الگو (پترن)، هر خوشه را در نظر بگیریم. به آن data Reduction (کاهش حجم داده) می‌گویند. مثلا به جای همه اعضای یک خوشه میانگین آن‌ها را در نظر بگیریم

② در مواردی که نسبت به داده‌ها آشنایی نداریم می‌توانیم یک آشنایی پیدا کنیم و می‌توانیم یک سری فرضیه‌هایی پیشنهاد بدهیم و آن‌ها را verify کنیم با داده‌های بیشتر (تولید و تست فرضیه)

۲۸ | 18 Jun

④ می‌توانیم برای داده‌های جدید پیش‌بینی کنیم که به کدام خوشه تعلق دارد.

پیش‌پردازش داده‌ها

اگر داده‌ها شرایط مطلوب برای کاری که ما می‌خواهیم انجام بدهیم را نداشته باشند، سعی می‌کنیم شرایطشان را تغییر دهیم به گونه‌ای که مناسب‌تر شوند برای هدف ما، به این کار Data preprocessing می‌گویند.

وقتی می‌خواهیم مسئله را حل کنیم یک سوالی که در ابتدای کار مطرح می‌شود این است که کدام متغیرها را ورودی بدهیم؟ وقتی تعداد ورودی‌ها زیاد باشد تعداد وزن‌ها زیاد می‌شود و آموزش شبکه سخت‌تر می‌شود بنابراین تلاش می‌کنیم که ممکن است بهتر است ورودی‌ها را کم کنیم.

Scanned with CamScanner

۲) مسئله دیگر این است که ورودی ها ممکن است باهم correlation داشته باشند که موجب می شود جواب واحدی برای مسئله وجود نداشته باشد و آموزش سخت می شود و مشکله در واقع در یک فضای با بعد کمتر کار می کند زیرا دقتی داده ها باهم correlation دارند ما بدون اینکه نیاز باشد رفته ایم به بُعد بالاتر. در این حالت می توانیم از بین ورودی هایی که باهم correlation دارند یکی را انتخاب کنیم و قرار بدهیم. پس مطلوب است که ورودی ها از هم مستقل باشند.

۳) بسیاری از اوقات جمع آوری داده برای مسئله غیرممکن است یا هزینه زیادی دارد به طوری که مانع انجام کار می شود. در این حالت تعداد داده هایی که داریم کمتر از تعداد مطلوب است.

باید سعی کنیم ورودی ها تعدادش کم باشد تا تعداد وزن ها کمتر باشد و آموزش آسان تر شود. یعنی اگر تعداد متغیرها (بُعد ورودی) زیاد و داده ها کم باشد تعیین صحیح وزن ها مشکل می شود.

۴) ممکن است اندازه اعدادی که داریم خیلی باهم متفاوت باشند. مثلا یکی از متغیرها در بازه [۱۰۰۰, ۱۰۰۰۰] و متغیر دیگر در بازه [۰, ۰.۰۰۱] باشد. در نورون ها برای فعال شدن نیاز بود که Net Input محاسبه شود. که x_i ها را با هم جمع می کند. اگر یکی از x_i ها خیلی بزرگ و یکی خیلی کوچک باشد، مقدار بزرگ غلبه می کند و کوچکتر ها بی اثر می شوند بنابراین اثر متغیرهای کوچکتر وارد نمی شود و یک کار مهمی که در پیش پردازش باید انجام شود این است که همه متغیرها را به یک بازه خاص بیاوریم. مثلا بازه [۰, ۱] ← نرمال سازی (بهنجارسازی)

نرمال کردن داده ها (Normalization of Data)

① Standardization

هر کدام از اندازه گیری ها را با یک تبدیل خطی به یک فضای جدید ببریم که میانگین متغیرها در آن فضا برابر صفر و انحراف معیارشان ۱ است.

تبدیلی که این کار را انجام می دهد:

$$x_{T_i}^n = \frac{x_i^n - \bar{x}_i}{\sigma_i}$$

$$\bar{x}_i = \frac{1}{N}\sum_{n=1}^{N} x_i^n$$

$$\sigma_i^2 = \frac{1}{N-1}\sum_{n=1}^{N}(x_i^n - \bar{x}_i)^2$$

Scanned with CamScanner

با انجام این تبدیل اکثر داده‌ها در فاصله ± 26 قرار می‌گیرند. چون میانگین آن‌ها را صفر کردیم عددی داده‌ها در اطراف مبدا می‌افتند و در فاصله‌ی $[-26, +26]$

اگر همبسته‌سازی را برای داده‌هایی انجام بدهیم که مقادیر مطلوب خروجی را هم داریم مثلا عمل دسته‌بندی، باید بر روی داده‌های خروجی هم همان عمل (همبسته‌سازی) انجام شود. (چون ورودی‌های متناظرش نرمالیزه شده اند)

اگر این عمل را انجام بدهیم باعث می‌شود بعضی از وزن‌ها بزرگ نشوند که مورد مطلوبی است. چون به همگرایی سریع آموزش صدمه می‌زنند.

② Simple Range Scaling

با این رابطه همه داده‌ها در بازه $[0,1]$ قرار می‌گیرند ←

$$\boxed{x_{Ti} = \frac{x_i - x_{imin}}{x_{imax} - x_{imin}}}$$

③ Normalization of correlated Multivariate Data

می‌توانیم عمل نرمالیزه کردن را به جای اینکه برای تک تک درایه‌ها انجام دهیم، برای یک بردار انجام بدهیم.

داده‌های چند متغیره‌ای داریم که ممکنه بین شان همبستگی هم وجود داشته باشد.

اگر K تا ورودی داشته باشیم. میانگین انگوهای مختلف را حساب می‌کنیم. N تا انگو داریم و بردار میانگین داریم

$$X = \{x_1, x_2, \dots, x_K\} \qquad \bar{x} = \{\bar{x}_1, \bar{x}_2, \dots, \bar{x}_K\} \qquad \bar{x} = \frac{1}{N}\sum_{n=1}^{N} x^n$$

هر کدام از مولفه‌های بردار $\bar{x}$، میانگین یکی از متغیرهای ماست.

ماتریس کوواریانس را با این تشکیل بدهیم:

$$\boxed{C_X = \frac{1}{N-1}\sum_{n=1}^{N}(x^n - \bar{x})(x^n - \bar{x})^T}$$

C_X یک ماتریس متقارن است، ابعادش $K \times K$ است که K بعد ورودی است

می‌خواهیم داده‌ها را به فضای جدیدی ببریم که واریانس واحد باشد و متغیرها از هم مستقل باشند. (در واقع می‌خواهیم همبستگی میان متغیرها را از بین ببریم. این کار را می‌توان با eigen value decomposition (تجزیه مقدار ویژه) یا PCA (تحلیل مولفه اصلی) انجام داد.

Scanned with CamScanner

۱۴۰۰

یعنی با اعمال تبدیل PCA روی داده ها می توان از فضای X به یک فضای Y رفت که فضای Y

این ویژگی ها را دارد: ① میانگین الگوها در آن ۰ است ② ماتریس cov قطری است (بین متغیرها

هم بستگی وجود ندارد)

$$\boxed{Y = A(X - \bar{X})} \qquad \boxed{m_Y = 0}$$

تبدیل را ماتریس A انجام می دهد که از روی بردارهای ویژه ی ماتریس C_X به دست می آید.

$$\boxed{C_Y = \frac{1}{N-1}\sum_{i=1}^{N}(Y_i - \bar{Y})(Y_i - \bar{Y})^T = A C_X A^T}$$

• تشخیص داده پرت (outlier Detection)

معمولا در داده هایی که از دنیای واقعی جمع آوری می کنیم داده هایی وجود دارند که با بقیه خیلی تفاوت دارند.

به آن ها outlier یا داده ی پرت گفته می شود که به آموزش ضربه می زنند بنابراین باید داده های پرت را حذف کنیم. در مقابل داده های پرت داده های inlier یا غیر پرت هستند.

قبل از تشخیص و حذف outlier ها باید روی داده ها نرمال سازی انجام شود.

چگونه داده های پرت را تشخیص بدهیم؟

باید داده ها را برحسب اندازه مرتب کنیم و آن هایی که در ابتدا و انتهای بازه قرار می گیرند را بررسی کنیم زیرا خیلی کوچک یا خیلی بزرگ هستند و می توانند جزو داده های پرت باشند

دو روش کلی وجود دارد: یکی اینکه داده های پرت را حذف کنیم (Trimming)

یکی اینکه به جای آن ها داده های جدیدی قرار دهیم ← تغییر داده ها (winsorizing)

← در این جا مقادیر انتهایی بازه ها را که تشخیص داده ایم داده های پرت هستند به جای آن ها مقداری قرار می دهیم که نزدیکتر است به میانگین داده ها مثلا به جای همه ی داده هایی که از میانگین + 3σ بزرگتر هستند همین مقدار mean + 3σ را قرار می دهیم. یا بزرگترین مقداری که داده پرت نیست و inlier است را به جای داده های پرت خیلی بزرگ قرار دهیم (برای مقادیر کوچک هم همین طور: mean - 3σ و کوچکترین غیرپرت)

Scanned with CamScanner

به راه دیگر: تمام اعدادی که بزرگتر از ۹۹ امین صدک یا کوچکتر از اولین صدک

بجای آن ها مقادیر بزرگ را قرار بدهیم. (همان مقدار ۹۹ امین صدک را بگذاریم)

تشخیص داده های پرت در داده های چند متغیره

باید به سراغ تبدیل PCA برویم. PCA یک دستگاه مختصات متعامد به ما می دهد. مولفه های اصلی

که در این فضای جدید خواهیم داشت به این صورت هستند که بیشترین واریانس در اولین مولفه ها

هست و همین طور که به جلو می رویم واریانس کم می شود.

داده های پرت در تغییرات تاثیرگذار هستند و در واریانس تاثیر می گذارند بنابراین می توان گفت مولفه های

اول بیشترین میزان تغییرات را دارند و این تغییرات می تواند تاثیر داده های پرت باشد یعنی اگر داده پرت

داشته باشیم تاثیرش بیشتر در مولفه های اول دیده می شود. به این ترتیب می توان از مولفه های اصلی استفاده

کنیم و داده های پرت را تشخیص بدهیم.

$var(a'_k X) = \lambda_k$

α_k بردار ویژه ی k ام

مقدار ویژه ی k ام

k امین مولفه اصلی X را با $a'_k X$ نشان می دهیم و

مولفه های اول بیشترین واریانس را دارند

Scanned with CamScanner

self_organizing (Feature) Maps (SOM)(SOFM)

در این جا وارد مدل آموزش بدون نظارت می شویم و تفاوت این مسئله با مسئله های قبل در این است.

یکی از کارهایی که این شبکه ها می توانند انجام دهند این است که داده ها را از فضاهای با ابعاد بالا Map می کنند به فضای ۱ یا ۲ یا ۳ بعدی.

شبکه هایی که برای این مسئله داریم:

معماری این شبکه های بدون نظارت یک معماری ۲ لایه است: لایه ورودی و لایه خروجی

معماری ساده ای دارند. لایه ورودی متغیرهای ورودی را از خارج شبکه می گیرد و به شبکه اعمال می کند

در این شبکه نورون های لایه خروجی می توانند به صورت خط (آرایه ۱ بعدی)، آرایه ۲ بعدی و یا آرایه ۳ بعدی چیده شوند. ورودی ها به تمام این نورون های خروجی متصل می شوند.

آموزش این شبکه ها اینگونه است که بر اساس یک قانون یادگیری ورودی ها را یکی یکی می گیرد و وزن ها را آپدیت می کند. تا به همگرایی برسد.

Scanned with CamScanner

انتظار داریم که آموزش به گونه ای انجام شود که هر کدام از نورون های لایه خروجی نماینده (مرکز) یک خوشه باشند

وقتی آموزش شبکه تمام می شود، مجموعه بردارهای وزن نورون ها CodeBook را تشکیل می دهند. هر بردار یک مرکز خوشه را تعیین می کند.

یادگیری رقابتی ← در این جا یادگیری بی نظارت ما یادگیری رقابتی است

هر ورودی که به شبکه اعمال می شود نورون ها با هم رقابت می کنند برای اینکه در پاسخ به این ورودی فعال شوند.

در انتهای یک دوره رقابت یک نورون فعال می شود و بقیه بازنده می شوند

در پایان دوره، وزن نورون برنده را بر اساس یک قاعده و وزن نورون های بازنده بر اساس یک قاعده دیگر اصلاح صورت می گیرد (یا اینکه اصلا اصلاحی صورت نمی گیرد برای نورون های بازنده).

وقتی اصلاح وزن انجام شد دوباره ورودی بعدی می آید و دوباره رقابت داریم و برنده که برنده اصلاح می شود و بازنده همانند ... همین طور این فرآیند ادامه پیدا می کند تا شرط خاتمه ارضا شود. به این نوع یادگیری، یادگیری رقابتی گفته می شود.

جلسه دوازدهم _ ۱۴۰۰/۸/۱۱

در یادگیری رقابتی فرق نورون برنده و بازنده در آموزشی است که می بینند.

اصلاح وزن ها در نورون برنده با نورون های بازنده متفاوت است.

ایده ی یادگیری رقابتی از مغز انسان گرفته شده است. اطلاعات مختلفی که وارد مغز انسان می شوند هر کدام به یک منطقه از مغز انسان نگاشت می شوند. اولین بار Rosenblatt مطرح کرد که یادگیری رقابتی در مغز انسان اتفاق می افتد و موجب می شود نگاشت اطلاعات به این شکل انجام شود.

Scanned with CamScanner

رقابت به این صورت است که هر نورون تلاش می کند Net Input خودش یا Activation اش max شود. وقتی اینطور شود خروجی مناسب و قوی ای خواهد داشت.

هر نورون سعی می کند خودش برنده شود و بقیه برنده نشوند. تلاش می کند Net Input بقیه نورون ها را پایین بیاورد به همین دلیل یک سیگنال منفی و مهارکننده به بقیه نورون ها می دهد. در نهایت یک نورون برنده می شود.

وقتی نورون برنده مشخص شد، وزن های نورون برنده به این صورت اصلاح می شود: این بردار x که الان به عنوان ورودی آمده یک نقطه در فضاست، وزن نورون برنده هم یک بردار که به مقداری جذب می شه به سمت x. (تولید شده) وزن بقیه نورون ها (غیر برنده ها) هیچ تغییری نمی کند. سپس یک ورودی دیگر اعمال می شود، یک نورون برنده می شود، نورون برنده ی جدید به سمت ورودی جدید جذب می شود و ... این فرآیند ادامه پیدا می کند و وزن ها تغییر می کنند

لایه ورودی ← به تعداد ابعاد ورودی، نورون دارد

لایه خروجی ← در انتهای آموزش هر نورون در لایه خروجی نشان دهنده ی یک خوشه است و بخشی از داده ها را به خودش اختصاص می دهد. بنابراین تعداد نورون ها در لایه خروجی برابر با تعداد خوشه ها در داده ها است. اگر تعداد خوشه ها را بدانیم به همان تعداد نورون می گذاریم. اگر ندانیم، از مقدار حدسی که می زنیم، نورون بیشتری می گذاریم. در این صورت تعداد خوشه ممکن است دو تکه شوند که در نهایت باید ترکیبشان کنیم.

برای قانون یادگیری وزن ها باید به مقدار یکوچک اتفاقی مقدار دهی اولیه شوند یا اینکه از بردارهای ورودی m تا را به صورت تصادفی برداریم و به عنوان وزن اولیه بدهیم. مزیتش این که سرعت آموزش را سریع کند.

انتخاب برنده (Winner Selection)

۱- استفاده از فعالیت نورون ها

اون نورونی برنده است که حداکثر فعالیت را از خودش نشان بدهد یا به عبارت دیگر، ورودی خالصش بزرگترین ورودی خالص در بین نورون ها باشد. ورودی خالص به این صورت است:

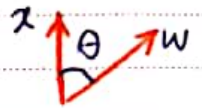
$$I_j = \sum_{i=1}^{n} w_{ij} x_i$$

Scanned with CamScanner

همه‌ی نورون‌ها، وقتی یک ورودی وارد می‌شود، مقدار I_j را محاسبه می‌کنند. هر نورون یک سیگنال عاملیت کننده برای بقیه نورون‌ها ارسال می‌کند. هر نورون مجددا I_j را با وجود این سیگنال‌های عاملیت کننده محاسبه می‌کند. یک حد آستانه داریم که اگر activation برای نورون کمتر از آن شد آن نورون را حذف می‌کنیم چون شرایط رقابت با بقیه نورون‌ها را ندارد و از رقابت خارج می‌شود.

۲) استفاده از فاصله تا بردار ورودی ← وقتی یک ورودی وارد می‌شود یک نقطه‌ای از فضای وزن‌ها را نشان می‌دهد که با یک بردار هم می‌توان نشان داده شود. Net Input یک نورون را می‌توان به صورت ضرب داخلی x و w_j نوشت:

$$\boxed{I_j = \sum_{i=1}^{n} w_{ij} x_i = X.W_j = \|X\| \, \|W_j\| \cos\theta}$$



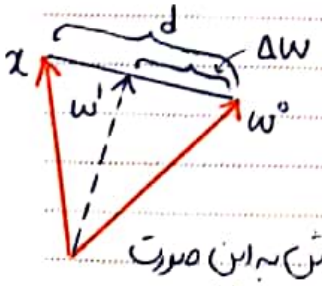
اگر اندازه وزن‌ها را نرمالیزه کنیم به 1، حداکثر شدن I بستگی به $\cos\theta$ دارد. یعنی θ حداقل باشد. که حداقل بودن θ یعنی نقطه w نزدیکتر باشد به نقطه x. بنابراین:

وقتی یک x ورودی می‌آید از فاصله‌ی بردارهای وزن مختلف تا این بردار x استفاده کنیم. آن که نزدیکتر به x باشد برنده است و اصلاح وزنش هم با جذب شدن w به سمت x انجام می‌شود یعنی θ کوچکتر می‌شود.

فاصله‌ای که حساب می‌کنیم می‌تواند اقلیدسی باشد:

$$\boxed{d_j = x - w_j = \sqrt{\sum_{i=1}^{n} (x_i - w_{ij})^2}}$$

ممکن است نخواهیم این عملیات را انجام دهیم چون نیاز به توان رساندن و جذر گرفتن به تعداد دفعات زیاد دارد. می‌توان از فاصله‌های دیگر هم استفاده کرد.



به شکل روبه‌رو فرض کنید w^0 نورون برنده باشد وقتی که x وارد شده است. این بدین معناست که w^0 از تمام wهای نورون‌های دیگر به x نزدیکتر بوده است. می‌خواهیم w^0 جذب شود به سمت x. بردار $x - w^0$ را در نظر می‌گیریم. قاعده آموزش به این صورت است که بردار وزن نورون برنده برابر بشود با مقداری که قبلا داشت به اضافه‌ی کسری از بردار تفاضل (Δw). در این صورت مقداری که w^0 تغییر می‌کند برابر بردار Δw است و نتیجه‌اش بردار w' است. که به سمت x کشانده شده.

Scanned with CamScanner

۱۴۰۰

$$\Delta w_j = \beta(x - w_j) = \beta d_j$$

قاعده آموزش:

$$w_{j+1} = w_j + \beta(X - w_j)$$

$(0 < \beta < 1)$ ← نشان می‌دهد چقدر جذب شود

β نرخ یادگیری است و اندازه‌ای که w نزدیک می‌شود به بردار ورودی را کنترل می‌کند. بقیه وزن‌ها تغییری نمی‌کنند. در نهایت هر کدام از وزن‌ها محصول ورودی یک سری بردار x است. که مجموعا آن‌ها این بردار وزن، با هم یک خوشه را تشکیل می‌دهند. فاصله دیگری که می‌توانیم استفاده کنیم:

$$d_j = \sum_{i=1}^{n} |x_i - w_{ij}|$$

آموزش در این شبکه به ۲ شکل می‌تواند باشد:

1) pattern or recursive mode ← یعنی وزن‌ها برای هر ورودی اصلاح می‌شوند.

2) Batch Mode به ازای هر ورودی وزن‌ها اصلاح نمی‌شوند بلکه اصلاح وزن‌ها بعد از تعدادی ورودی (که به این تعداد یک Batch گفته می‌شود) انجام می‌شود.

Batch می‌تواند آنقدر بزرگ باشد که به اندازه همه ورودی‌ها باشد. در نتیجه اعمال یک Batch می‌شود یک epoch. یعنی یک epoch ورودی را اعمال می‌کنم و اصلاح وزن‌ها را پیدا می‌کنم ولی اعمال نمی‌کنم تا epoch کامل شود. در انتهای epoch با میانگین‌گیری اصلاح را انجام می‌دهم.

اعمال ورودی‌ها epoch به epoch انجام می‌شود تا جایی که تغییر مهم دیگری مشاهده نشود. یعنی بردار وزن‌های نورون‌ها به نقطه‌ای که باید منتقل می‌شدند رسیدند.

یک معیار توقف دیگر: وقتی متوسط فاصله میان نورون برنده و همه ورودی‌هایی که این نورون نماینده آن‌هاست، به یک مقدار minimum برسد. به این معناست که این بردار وزن به اندازه کافی به همه نورون‌های مربوط به خودش نزدیک است. این فاصله وقتی min شود می‌تواند معیاری برای توقف باشد. این فاصله به این صورت محاسبه می‌شود:

(مجموع)

$$D = \sum_{i=1}^{K} \sum_{n \in C_i} (x^n - w_i)^2$$

برای همه نورون‌ها (K تا)

و برای همه ورودی‌هایی که در هر خوشه است (C_i)

Scanned with CamScanner

نکته: نورون هایی که در شبکه در کنار هم و نزدیک هستند لزومی ندارد که خوشه هایی را نشان دهند که در فضای ورودی نزدیک بهم هستند.

اگر مراکز خوشه ها را هم وصل کنیم و عمود منصف خط حاصل را رسم کنیم، این عمود منصف ها در نقاطی یکدیگر را قطع می کنند و یک ناحیه بندی در فضا ایجاد می کنند. این اتفاق در دسته بندی هم می افتاد.

می توانیم برای هر کدام از نورون ها یک محدوده جذب تعریف کنیم. (محدوده تاثیر یا جذب)

محدوده ای که نقاط موجود در آن به این وزن نورون یا مرکز خوشه نزدیکترند تا سایر مرکز خوشه ها، محدوده جذب این نورون است و اگر یک ورودی جدید در این محدوده وارد شود متعلق به این خوشه خواهد بود.

با این ناحیه بندی می توانیم مشخص کنیم بعد از این هر ورودی ای باید متعلق به کدام خوشه خواهد بود.

به این سلول ها Voronoi cells و به این ناحیه بندی Voronoi tessellation می گویند.

self_organizing Feature Maps

یک کار دیگری که کوهونن انجام داد این بود که گفت می توانیم نقشه هایی از داده ها تشکیل بدهیم که داده های با بُعد بالا را در بُعد پایین تری برای ما ارائه می دهند. این ایده هم از اتفاقی که در مغز انسان می افتد گرفته شد. هر بخشی در مغز مربوط به کار مشخصی است و در یک ناحیه نیز کارهای شبیه به هم در نواحی نزدیک بهم انجام می شود. به حفظ سازمان دهی مکانی topology preservation گفته می شود. اگر در فضای n بعدی داده هم نزدیک باشند، این داده ها وقتی در مغز نگاشت می شوند، نزدیک بهم نگاشت می شوند یعنی سازمان داده ها هم نمی ریزد.

نقشه ویژگی کوهونن (نقشه ویژگی خودسازمانده) نیز این ویژگی را دارد یعنی داده ها به فضای با ابعاد کمتر و ساده تر نگاشت می شوند اما این نگاشت به گونه ای است که توپولوژی حفظ شود.

Scanned with CamScanner

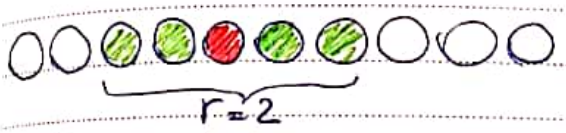
برای اینکه خاصیت حفظ توپولوژی ایجاد شود اینطور عمل می‌کنیم:

وقتی یک نورون برنده می‌شود، نه تنها وزن خودش بلکه وزن همسایه‌هایش را هم اصلاح کنیم.

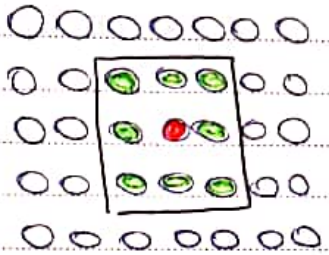
اصلاح وزن همسایه‌ها متناسب با میزان فاصله‌ای که از نورون برنده دارند صورت می‌گیرد. یعنی نورون‌هایی که دورتر هستند اصلاح وزنشان کم‌تر است.

در این جا نیاز داریم که همسایگی را هم تعریف کنیم:

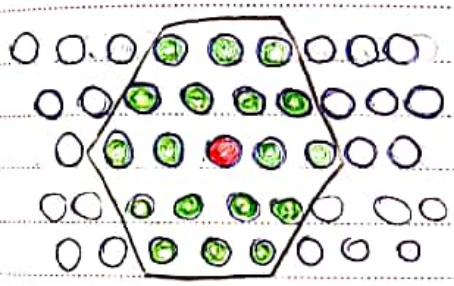
اگر آرایه ۱ بعدی باشد:



اگر آرایه 2 بعدی باشد:



همسایگی مربعی با $r=1$



همسایگی ۶ ضلعی با $r=2$

این یک راه است برای تعیین همسایگی‌ها، راه دیگری هم وجود دارد.

فرض کنید ورودی x را داریم: $x = \{x_1, x_2, \dots, x_n\}$

این ورودی در نقطه‌ای از نقشه مانند $\{i_{win}, j_{win}\}$ وارد می‌شود ← محل نورون برنده است

وزن‌های نورون برنده و همسایه‌هایش با این قاعده به روز می‌شود:

$$w'_j = w_j + \beta\, NS[x - w_j]$$

قاعده قبلی که گفتیم برای خوشه‌بندی بود و این قاعده برای تشکیل نقشه است.

یک عامل NS داریم که در β و در $x - w_j$ ضرب شده است. NS → Neighbor strength

NS برای اعمال تاثیر فاصله‌ی نورون از نورون برنده بر روی میزان اصلاح وزن اضافه شده است.

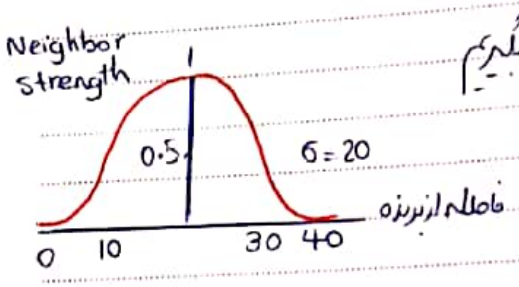
β هم قبلا گفتیم ضریبی است که میزان اصلاح را مشخص می‌کند.

Scanned with CamScanner

انواع NS ها:

۱) NS می‌تواند یک تابع خطی که با فاصله کاهشی است، باشد. مقدار این تابع در همسایگی به صورت خطی کاهش پیدا می‌کند (ساده‌ترین نوع NS)

۲) Guassian decay function ← NS را تابع گاوسی در نظر بگیریم



$$NS = Exp\left[\frac{-d_{i,j}^2}{2\sigma^2}\right]$$

قله روی نورون برنده قرار می‌گیرد (مقدار ۱)

همسایه‌ها به نسبت فاصله‌ای که از نورون برنده دارند این تابع مشخص می‌کند که در چه ضریبی ضرب شوند (یه عدد کوچکتر از ۱) هرچه فاصله بیشتر شود این ضریب کوچکتر می‌شود.

σ در این تابع پهنی و باریکی این تابع را کنترل می‌کند و اندازه‌ی همسایگی را هم مشخص می‌کند. بین σ و اندازه‌ی همسایگی که در نظر داریم باید رابطه‌ای برقرار باشد.

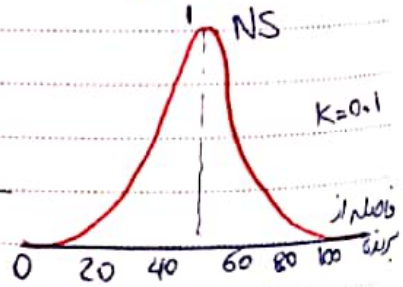
روز صنعت و معدن

تیر | ۱۱ | جمعه
2 Jul ■ ۲۱ ذی القعده

۳) Exponential Decay Function ← NS تابع نمایی باشد

$$NS = Exp[-k d_{i,j}]$$

← با سرعت بیشتری NS کاهش پیدا کرده



اثر کمتری برای نورون‌های همسایه دارد. قله محل نورون برنده است.

اصلاح وزن همسایه‌ها به مرور زمان باید کم و متوقف شود و فقط نورون برنده را اصلاح کنیم. به همین دلیل اندازه‌ی همسایگی همین طور که آموزش جلو می‌رود باید با یک قاعده‌ای کوچک شود. ابتدا همسایگی را خیلی بزرگ می‌گیریم، حتی گفته شد در ابتدا بیشترین تعداد نورونی که می‌تواند در همسایگی قرار بگیرد را قرار بدهیم. زیرا در ابتدا بردارهای وزن نورون‌ها به صورت تصادفی داده شده اند و می‌توانند در جاهای خیلی پرتی باشند

Scanned with CamScanner

۱۴۰۰

و با آمدن ورودی ها ممکن است یک نورون هیچ گاه برنده نشود. در بین این وزن های تصادفی اولیه ی نورون ها ممکن است یک نورون آنقدر دور باشد از ورودی ها که هیچ گاه نتواند برنده شود. همیشه نورون های دیگری هستند که فاصله شان از ورودی کمتر است نسبت به این نورون. و ما تا آخر پیش می رویم و وزن این نورون همان وزن اولیه ی تصادفی باقی می ماند و این نامطلوب است. به این نورون، نورون مرده می گویند که وزن هایش اصلا اصلاح نمی شوند.

برای اینکه جلوی به وجود آمدن نورون های مرده گرفته شود کوهونن گفت در ابتدا هر همسایگی را باید بزرگترین همسایگی ممکن بگیرید و بعد به تدریج اندازه همسایگی را کاهش بدهید. در این صورت در ابتدا همه ی نورون ها اصلاح می شوند و از شکل گیری نورون مرده جلوگیری می شود.

برای کوچکتر کردن اندازه ی همسایگی یک رابطه خطی معرفی کرد:

$$\sigma_t = \sigma_0 \left(1 - \frac{t}{T}\right)$$

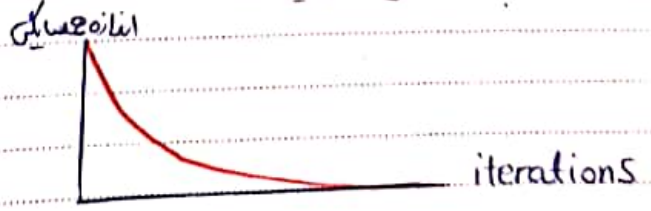
σ_0 ← اندازه ی اولیه ی همسایگی

σ_t ← اندازه همسایگی در قدم t ام

T ← تعداد کل iteration هایی که قرار است σ_0 را به مقدار ۰ برساند، یعنی فقط نورون برنده بماند

می توان رابطه ی نمایی تعریف کرد:

$$\sigma_t = \sigma_0 \, Exp\left[\frac{-t}{T}\right]$$



کاهش اندازه همسایگی را می توان NS (تغییر دامنه) ترکیب کرد:

$$NS(d,t) = Exp\left[\frac{-d_{ij}^2}{2\sigma_t^2}\right] = Exp\left[-d_{ij}^2 / 2\{\sigma_0 \, Exp(-t/T)\}^2\right]$$

Scanned with CamScanner

Learning Rate Decay

دیگر از روش‌های کوهونن گفت خود نرخ یادگیری هم نباید ثابت باشد. در شبکه FF هم همینطور بود. می‌خواهیم وقتی به همگرایی می‌رسیم، میزان تغییرات جزئی باشد.

خطی

$$\beta_t = \beta_0 \left(1 - \frac{t}{T}\right)$$

یا

$$\beta_t = \beta_0 \exp\left(-\frac{t}{T}\right)$$

نمایی

$$\beta_t = \frac{2}{3+t}$$

باید β را کاهش بدهیم اما جوری که از ۰ کمتر نشود.

قاعده اصلاح وزن‌ها در شبکه کوهونن:

$$w_j(t) = w_j(t-1) + \beta(t)\, NS(d, t)\, [x(t) - w_j(t-1)]$$

این‌ها هم آموزش می‌تواند Batch Mode یا Recursive Mode باشد.

آموزش شبکه SOM را می‌توان در ۲ فاز در نظر گرفت:

① ordering (مرتب شدن) ← برای بردارهای وزن topological ordering پیش می‌آید. در ابتدا بردارهای وزن چون تصادفی انتخاب می‌شن ممکنه خیلی پرت و پلا باشند جابه‌جایی‌های بزرگی باید صورت بگیره تا در حدود ناحیه‌ای که باید باشند قرار بگیرند. در این فاز نرخ یادگیری باید بالا باشد، بعد کم کم کاهش پیدا کند و به صفر برسد. اندازه همسایگی باید ابتدا بزرگ باشه (حداکثر نورون‌ها رو پوشش بده) و بعد کاهش پیدا کنه. برای این فاز چندین هزار iteration لازمه.

② فاز همگرایی ← این فاز، فاز تنظیم دقیق است یعنی وزن هر نورون دقیق مشخص می‌شه. تغییرات خیلی کمه. جابه‌جایی‌ها خیلی کوچیکه. شاید چند صد یا چند هزار iteration داشته باشه. نرخ یادگیری در کل این فاز باید ۰ باشه که تغییرات جزئی باشه.

Fine-tunning

Scanned with CamScanner

مقداردهی اولیه نقشه ← ① مقادیر تصادفی

Map

② انتخاب از بردارهای ورودی

③ ابتدا با وزن‌های کوچک شروع می‌کنیم سپس مقدار iteration که جلو رفتیم به سمت وزن‌ها بدست میاد

اینارو به عنوان وزن‌های اولیه در نظر بگیریم و آموزش را از سر بگیریم

جلسه سیزدهم - ۱۴۰۰/۸/۱۶

گفتیم در نقشه‌های کوهونن توپولوژی حفظ می‌شود یعنی این نقشه معنی دار است. وقتی ورودی آن حرکت کند به سمت وزن‌ها کم و زیاد می‌شود.

خاصیت این نقشه این است که اگر ۲ تا ورودی در فضای n بعدی کنار هم نزدیک باشند در نقشه هم به هم نزدیک هستند و آن‌هایی که دور باشند در نقشه هم دور از هم.

مثال مجموعه داده iris ← هدف اینه که یک نقشه ۲ بعدی تشکیل بدهیم و داده‌های ۴ بعدی را به ۲ بعد map کنیم

اندازه map ← 8x8

همسایگی را گاوسی و ضعیف در نظر گرفته $\beta_0 = 0.5$ $T = 20 \times N$ $N = 150$

اندازه همسایگی به تدریج کاهش پیدا می‌کند تا اینکه به ۱ نورون می‌رسد

در نهایت هر کدام از نورون‌ها یک بردار مرکز جوشه شده‌اند. هر کدام یک کد هستند در codebook

یک سری از ورودی‌ها به آن نورون map شده‌اند و مرکز آن ورودی‌ها را وزن‌های این نورون ارائه می‌دهد

مثال: Input distribution estimation

گفتیم که در قانون یادگیری، وقتی ورودی می‌آید نزدیک‌ترین بردار وزن نورون‌ها، به سمت آن جذب می‌شود. یعنی بردارهای وزن نورون‌های ما کشانده می‌شوند به سمت جاهایی که داده وجود دارد و جاهایی که چگالی داده‌ها بالاست.

روز قلم

Scanned with CamScanner

این بردارهای وزن توانند برای تخمین توزیع ورودی به کار گرفته شوند. یعنی بردارهای وزن نورون‌های شبکه می‌توانند تقریب توزیع ورودی را به ما بدهند.

مثلاً ورودی را اگر از یک توزیع یکنواخت در بازه [۰و۱] بگیریم هر چقدر آموزش جلو می‌رود سعی می‌کند به صورت مساوی پوششش یکنواخت به صفحه بدهد. بردارهای وزن نورون‌ها

!. SOM می‌توانیم مدل‌هایی برای تولید شکل‌های مختلف ایجاد کنیم

Time_adaptive SOM ← فاکتور زمان را در SOM آورده اند. اگر شکل با زمان تغییر کند SOM هم بتواند تغییر کند با آن. کاربرش. دنبال کردن اشیاء متحرک

Evolving self_organizing Maps

گفتیم SOM خیلی مفید در خوشه‌بندی و مصورسازی داده‌های با ابعاد بالا.

اما این شبکه محدودیت‌هایی دارد: ① در مورد اندازه‌ی شبکه باید از اول تصمیم بگیریم و انتخاب کنیم. برای یک مسئله هیچ ایده‌ای نداریم که چند تا نورون باید بدهیم.

② گفتیم در شبکه خودسازمانده و نقشه‌ی حاصل از آن، هر نورون یک خوشه را ارائه می‌دهد و آن‌هایی که نزدیک به هم واقع می‌شوند چیزهای شبیه به هم را ارائه می‌دهند. اگر بخواهیم به سری داده‌ها را دقیقاً خوشه‌بندی کنیم به سری داده‌های شبیه به هم که روی نقشه به چند تا نورون map شده اند، در حقیقت باید به یک نورون map بشوند.

به عبارتی، خوشه‌هایی که روی نقشه تشکیل شده یک مقدار کار اضافی لازم دارد برای اینکه خوشه‌های مرتبط با هم ترکیب شوند.

③ وقتی یک نقشه داریم و آن را آموزش داده‌ایم، ورودی جدیدی که وارد می‌شود نمی‌تواند به هر جای نقشه که

Scanned with CamScanner

کمترین جا برایش هست map نیست بلکه مجبوره به یکی از نورون ها map بشه. یعنی بعد این
مکانیزم حالت گسسته دارد و پیوسته نیست

برای حل این مشکلات کارهایی صورت گرفته است. یه سری از شبکه ها به نام SOM رشد یافته به
وجود آمده. چند دسته هستند. به صورت پویا نقشه را تشکیل می دهند. یک نقشه اولیه از تشکیل می شود
و رشد پیدا می کند.

جلسه چهاردهم - ۱۴۰۰/۸/۱۸

(GSOM) Dynamic self-organizing maps with Controlled Growth

در این روش همین طور که جلو می رویم نقشه رشد می کند. این رشد توسط یک عامل گسترش یا
عامل توزیع (Spread factor) کنترل می شود.

اگر در این سیستم یک نقشه ی کوچکی داشته باشیم یک تقریب کلی به ما می دهد از خوشه ها.
این ویژگی به خصوص متن مفید است که دیتاست های بزرگ داشته باشیم و بخواهیم یک دید
کلی از داده ها به دست بیاوریم. در این حالت یک نقشه کوچک با یک SF پایین ایجاد می کنیم.
(broader) (macro)
اگر بخواهیم نقشه ای با جزئیات بیشتر داشته باشیم باید شبکه را بیشتر رشد بدهیم.
به کار دیگه ای که همیشه کرد: به تیکه کوچکش را گرفت و بسطش داد. این متنی که علاقه مندیم
و بزرگش میکنه.

رشد این شبکه داشته خوشه هایی است که در داده ها وجود دارد. در واقع مشکلی که این رشد به خودش
می گیرد منتج از خوشه هایی است که در داده ها است.
مثلا مشاهده می کنیم شبکه در یک جهت های خاصی بیشتر رشد کرده و همین جهت ها به ما خوشه ها را
نشان می دهند.

Scanned with CamScanner

ویژگی دیگر این شبکه این است که شکلی که رشد آن به خودش می‌گیرد در نتیجه‌ی خوشه‌های آن ۱۴۰۰
است. مثلا در شکل می‌بینیم

شروع کار از یک نقشه‌ی اولیه‌ی ۲×۲ (چهار نورون) است. در ۴ راس یک مربع نورون‌ها قرار می‌گیرد.
افزایش نورون‌ها در حین رشد شبکه، در نورون‌های مرزی انجام می‌گیرد که این نورون‌ها بالاترین خطای
مجتمع (highest accumulated error) را دارند. اگر جای نورون باشد که خطای بالایی را نشان می‌دهد،
در واقع نشانه‌ی این است که این جا به نورون بیشتری نیاز داریم و بیاراست خوشه‌ها بیشتری تشکیل بشه که خطا بیاد پایین.
خوشه‌های این جا برای کنترل دارند. پس در مرزها هر جا خطا ناشی است اضافه می‌کنیم
اضافه شدن نورون با یک قاعده‌ای است که ساختار شبکه بهم نمی‌ریزد. همان ساختار توری مانندی که
از ابتدا با آن شروع کرده بسطش می‌دهد.

پارامتر spread factor برای کنترل رشد به کار می‌رود. اگر در ابتدا مقدار پایین برای آن در نظر بگیریم
یک نقشه کلی می‌دهد و بعد با افزایش آن جزئیات بیشتری را می‌توانیم داشته باشیم و در واقع بسط پیدا می‌کند
در نواحی. ابتدا با ۴ نورون شروع می‌شه و به مقداری که جلوی رو مشاهده می‌شه به نورون خطای بالایی دارد از آن جا بسط می‌دهیم
بسط به این صورت است که در تمام جهت‌های آزاد نورون که دارای خطای بالاست، نورون اضافه می‌شود.

فرآیند آموزش

1. فاز مقداردهی اولیه: ۴ تا نورون در شبکه قرار می‌دهیم. باید وزن‌های اولیه‌ای به شان بدهیم. می‌توانیم رندوم انتخاب کنیم
همچنین می‌توانیم ۴ تا از بردارهای ورودی را انتخاب کنیم و به عنوان وزن اولیه بدهیم.
بهتر است روی داده‌ها یک عمل Normalization انجام بدهیم تا همه را در فاصله‌ی ۰ و ۱ بیاوریم و مقیاس‌هایی
که از نظر مقیاس با هم خیلی فرق دارند در یک محدوده قرار بگیرند و این‌ها را به عنوان وزن اولیه بدهیم.

Scanned with CamScanner

۲) فاز رشد (Growth phase)

ورودی ها را به شبکه می دهیم و هر کدام به یک نورون تخصیص پیدا می کند.

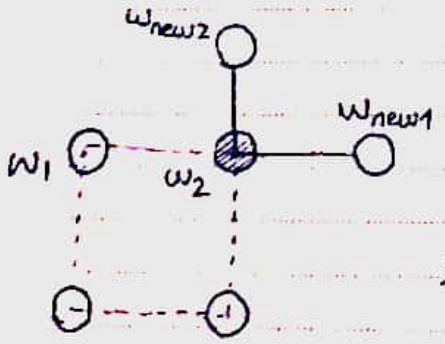
می توانیم در هر مرحله خطای مجتمع را برای هر نورون پیدا کنیم. خطای مجتمع یعنی یک نورون داریم که وزن هایش عوض می شود که خوشه حساب باشد، فاصله هر کدام از ورودی هایی که به این خوشه اختصاص پیدا کرده اند را تا مرکز خوشه حساب می کنیم و با هم جمع می کنیم. جایی که بزرگترین خطای مجتمع را داریم باید بررسی شود برای اضافه کردن نورون.

در ابتدا که ۴ تا نورون داریم اگر بگوییم که فضای ورودی ها را به ۴ ناحیه تقسیم می کنند. نورونی که بیشترین خطا را دارد نشان می دهد که ورودی هایی که به ناحیه ی مربوط به آن می آیند پراکندگی شان زیاد است و ناحیه ای است که پتانسیل آن را دارد که شبکه از آن جا رشد کند.

خطای مجتمع اگر از یک تریشولد ((GT) growth threshold) بزرگتر باشد، می گوییم شبکه بایستی در این قسمت رشد بکند. یعنی x رقمت های آزاد مربوط به این نورون، نورون اضافه شود.

وزن های اولیه ی نورون های جدید را چگونه بدهیم؟

وزن های اولیه برای نورون های تازه اضافه شده باید به گونه ای باشد که تغییرات وزن در ابتدا اثر در عملکرد حالت هموار داشته باشد و نوسان نکند. (کم و زیاد نشود)



توصیه کرده اند: اگر w_2 از w_1 کوچکتر، وزن w_{new1} را هم کوچکتر از w_2 بگیرید. همانطور که w_2 از w_1 کوچکتر، w_{new1} هم از w_2 کوچکتر باشد.

$$\text{if } w_1 < w_2 \rightarrow w_{new1} = w_2 + (w_2 - w_1)$$

$$\text{if } w_1 > w_2 \rightarrow w_{new1} = w_2 - (w_1 - w_2)$$

Scanned with CamScanner

ابتدای کار ۴ تا نورون داریم و هر کدام وزنی رندوم می‌شوند چون هر کدام $\frac{1}{4}$ صفحه را برعهده دارند و ورودی‌هایی که به هر کدام اختصاص پیدا می‌کند مختلف و متفاوت می‌باشند.

نتیجه این مسئله این است که اگر از قانون کمتگام سازی وزن‌ها استفاده کنیم وزن‌ها مختلف نوسان کنند به طور مداوم چون ورودی‌ها متفاوت اند و وزن‌ها همراه آن‌ها کشیده می‌شوند.

در نتیجه گفته اند نرخ یادگیری را تغییر کنند براساس اینکه در چه مرحله‌ای از آموزش هستیم و اندازه شبکه چقدر است پیشنهاد دادند η که نرخ یادگیری است

$$\eta(t+1) = \alpha \varphi(n) \eta(t)$$

را اینگونه آپدیت کنیم:

α یک ضریب ثابت است $(0<\alpha<1)$ — تعداد نورون‌های شبکه

$\varphi(n)$ یک تابع است که مقدار آن وابسته به اندازه شبکه است و رابطه مستقیم با اندازه شبکه دارد. یعنی وقتی اندازه شبکه کوچیکه، کوچک و برعکس. یک پیشنهاد برای φ:

$$\varphi(n) = 1 - \frac{R}{n(t)}$$

در ابتدا این ضریب کوچک است و با افزایش $n(t)$ افزایش پیدا می‌کند

R را خودشان پیشنهاد دادند که 3.8 بگیریم. n تعداد نورون‌ها در قدم t است. در ابتدا $1-\frac{3.8}{4}$ عدد کوچکی است. وقتی ضریب در ابتدا کوچکه یعنی جابجا شدن بردار وزن نورون با آمدن ورودی‌ها، کم است اما وقتی نورون‌ها زیاد شد بزرگ حرکت می‌کند.

در شبکه SOM گفتیم که در ابتدا همسایگی را تا حای ممکن بزرگ می‌گیریم به طوری که شامل خیلی از همسایه‌ها شود و بعد رفته رفته آن را کوچک می‌کنیم. گفتیم که شبکه یک فاز ordering دارد که نورون‌ها جابجایی‌های اساسی انجام می‌دهند و می‌روند در محدوده‌ای که جایشان هست قرار می‌گیرند. و بعد هم فاز convergence داریم که دقیق تنظیم می‌شوند.

اما این فاز ordering به اندازه شبکه چون همین که نورون اضافه می‌کنیم خودمان ترتیب را رعایت می‌کنیم و تا حدود زیادی نورون‌ها خودشان مرتب هستند. در این حالت همسایگی‌های کوچکی در نظر می‌گیریم مثلاً ۲ تا ۴ در نظر می‌گیریم و بعد کاهش می‌دهیم به ۱.

روز گفت و گو و تعامل سازنده با جهان

Scanned with CamScanner

گفتیم که در این شبکه بسط از طریق نورون های مرزی صورت می گیرد حال اگر خطای مجتمع زیاد برای یکی از نورون های مرکزی رخ دهد باید چه کار کرد؟ در این حالت در واقع وسط نقشه ی ما خلل رسد و نورون های همسایه پیشرو دارند که ما این فشار را از این خلل کم کنیم.

برای کاهش این فشار پیشنهاد شدن که فشار را منتقل کنیم به همسایه هایش. برای همسایه ها هم این روند تکرار شود تا کمابیش به مرز برسد.

برای انجام این کار، نورون که خطای مجتمعش بیشتر از GT شده، خطای مجتمع آن را برابر نصف آستانه قرار می دهد

$$E^{\omega}_{t+1} = \frac{GT}{2}$$

و نورون های همسایه ی آن را در نظر گرفته و خطایشان را با کسری از خطا جمع کند

$$E^{i}_{t+1} = E^{i}_{t} + \gamma E^{i}_{t}$$

به این ترتیب اثر این نورون منتقل می شود به همسایه هایش.

γ عددی بین ۰ و ۱ است

۲۵ | تیر جمعه

16 Jul ■ ۵ ذی الحجه

گفتیم که Growth threshold تکلیف این که شبکه رشد بکند یا نه را مشخص می کند و کنترل می کند. یک عامل گسترش یا SF تعریف می کنیم: Spread Factor (SF) عددی بین ۰ و ۱ است. تعیین کننده ی این است که شبکه رشد می کند و مرتبط با GT است

SF باید با GT در ارتباط باشد چون هر دو رشد شبکه را کنترل می کنند:

$$GT = -D \ln(SF)$$

D ← تعداد داده های ورودی $0 < SF < 1$.

$-D \ln(SF)$ ← عددی مثبت است که ضریبی از بعد ورودی است

بر اساس این رابطه وقتی SF کوچک باشد مجبور می شود که GT بزرگ تر شود و وقتی GT بزرگ باشد جلوی رشد شبکه گرفته می شود بنابراین SF کوچک یعنی رشد کمتر. وقتی SF بزرگتر شود GT کوچکتر می شود در نتیجه رشد آسانتر انجام می شود. شبکه بیشتر رشد می کند جزئیات بیشتری می شود

Scanned with CamScanner

فاز رشد به این صورت ادامه پیدا می‌کند اما زمانی می‌رسد که دیگر نیازی نیست شبکه رشد پیدا کند و این حالت پیش می‌آید که خطای مجتمع نورون‌ها به GT برسد یا اگر می‌رسد با فرکانس پایین اتفاق می‌افتد. در این حالت می‌گوییم رشد دیگه تمام شده و وارد فاز دیگری می‌شویم:

فاز هموارسازی (Smoothing)

در این فاز آموزش را ادامه می‌دهیم با یک نرخ یادگیری پایین. هدفش اینه که به حرکات جزئی اگر نورون‌ها نیاز داشتند بتونه انجام بشه و تنظیمات ظریف شبکه صورت بگیرد.

همسایگی در این فاز فقط شامل خود نورون و ۴ تا همسایه‌ی اطرافش می‌شود.

مثال: zoo dataset

در ابتدا SF=0.1، تقریباً صفر گره شبکه ابتدا یک طرح کلی به دست آورد. مشاهده می‌شه در ۳ جهت رشد کرده. دوباره با SF=0.25 روشی دهد نقشه‌ای با جزئیات بیشتر به دست می‌آید. در هر شافل کلی به سراغ خوشه‌های هم دیده می‌شود.

یک کار دیگه که بیشتر انجام داده: یه کمیش قسمتی از نقشه را با SF بالا بسط بدهیم

وقتی شبکه را استرش دهیم اگر داریم روی لوشن را بالا می‌بریم. لازم نیست تعداد نورون‌ها حتماً برابر تعداد خوشه‌ها باشه.

لذا از این بحث وارد آموزش supervised می‌شویم دوباره

Scanned with CamScanner

جلسه پانزدهم - ۱۴۰۰/۸/۲۳

جدایی پذیری الگوها

وقتی مسئله خیلی پیچیده می‌شود، آن را ببریم به یک فضای دیگر که احتمال می‌دهیم در آن فضا مسئله راحت‌تر حل شود. یک تبدیل غیرخطی به داده‌ها اعمال می‌کنیم و آن را به فضایی با ابعاد بزرگتر مساوی فضای قبلی می‌بریم. برای یک مسئله پیچیده اگر آن را به فضای با بُعد بالا با تبدیل غیرخطی ببریم این احتمال وجود دارد که بتواند به صورت خطی جدایی پذیر شود. (و این احتمال در فضای با بعد بالاتر بیشتر است تا فضای با بعد پایین‌تر) به شرطی که فضا خیلی پرچگال نباشد (نقاط به صورت dense نباشند)

x x_1 x_2

فرض کنید x مجموعه‌ای از n تا بردار x_i باشد که هر کدام متعلق به x_1 یا x_2 که زیرمجموعه‌های x اند، می‌باشند.

$$\phi(x) = [\varphi_1(x), \varphi_2(x), \dots, \varphi_m(x)]^T \qquad \phi(x): R^n \longrightarrow R^m \qquad m \geq n$$

برای هر بردار x یک بردار تعریف می‌کنیم که شامل مقادیر حقیقی توابعی است که این توابع را با φ_i نشان می‌دهیم. کاری که این بردار $\phi(x)$ انجام می‌دهد این است که x را از فضای n بعدی می‌برد به یک فضای جدید که با ϕ نشان داده می‌شود و m بُعدی است.

نام دیگر فضای ϕ فضای ویژگی است. می‌توان گفت φ_iها ویژگی‌های x هستند.

یک دسته‌بندی X به دو مجموعه X_1 و X_2 را می‌گوییم φ separable (φ جدایی پذیر) است اگر بردار m بعدی w وجود دارد به طوری که:

$$w^T\phi(x) > 0 \qquad x \in \mathcal{X}_1$$

$$w^T\phi(x) < 0 \qquad x \in \mathcal{X}_2$$

یعنی با محاسبه‌ی $w^T\phi(x)$ و بررسی علامت آن می‌توانیم تصمیم بگیریم x متعلق به دسته $\mathcal{X}_1$ یا $\mathcal{X}_2$ است.

مرزی که این ۲ دسته را در فضای φ از هم جدا می‌کند با معادله روبه‌رو مشخص می‌شود: $w^T\phi(x) = 0$

در فضای اولیه‌مان هم x هایی که در $w^T\phi(x) = 0$ صدق می‌کنند مرز را تشکیل می‌دهند.

Scanned with CamScanner

2 تا مثال در اسلاید (انتگرال و ...)

۱۴۰۰

این ایده می‌تواند در شبکه‌های عصبی به کار گرفته شود و شبکه‌ای که این کار را می‌کند RBF است.

Function approximation with radial basis functions

ما می‌توانیم یک تابع را با یک سری توابع دیگر تخمین بزنیم که این توابع یک سری توابع پایه هستند و یک نوع از آن‌ها توابع پایه شعاعی است.

فرض کنید ما تا نقطه‌ی X_i داریم و L تا مقدار متناظر این X_iها که با d_i نشان می‌دهیم:

$$\{X_i \in \mathbb{R}^n \quad i=1,2,3,\dots,L\}$$

$$\{d_i \in \mathbb{R} \quad i=1,2,3,\dots,L\}$$

X_iها n بعدی اند و d_i اسکالر حقیقی است و مقدار تابعی است که می‌خواهیم تخمین بزنیم.

هدف این است که تابع f را بیابیم به گونه‌ای که:

$$f(X_i)=d_i \qquad i=1,2,3,\dots,L$$

یک راه که پیشنهاد شد: f را برابر مجموع وزن دار یک سری توابع نشان دهیم که به این توابع، توابع پایه شعاعی می‌گوییم و آن‌ها را با φ نشان می‌دهیم

$$\boxed{f(x)=\sum_{j=1}^{L} w_j\, \varphi(\|X-X_j\|)}$$

بنابراین مجموعه توابعی که به کار گرفته می‌شود را به این شکل نشان می‌دهیم:

$$\{\varphi(\|X-X_j\|),\ j=1,2,\dots,L\}$$

$\|\cdot\|$ ← نرم اقلیدسی

داده‌های ما X_jها هستند که آن‌ها را به عنوان مرکز این توابع در نظر می‌گیریم. یعنی علت اینکه به جای $\varphi(\|X\|)$ داریم $\varphi(\|X-X_j\|)$ این است که مراکز روی نقاط X_j باشند.

$$\boxed{f(X_i)=\sum_{j=1}^{L} w_j\, \varphi(\|X_i-X_j\|)=\sum_{j=1}^{L} w_j \varphi_{ij}} \qquad i=1,2,\dots,L$$

اگر برای iهای مختلف این رابطه را بنویسیم، ما تا رابطه به دست می‌آید که به جای $f(x_i)$ هم می‌توان d_i را در آن‌ها قرار داد.

ما معادله را می‌توان در یک فرم ماتریسی به شکل روبه‌رو ترکیب کرد:

$$\begin{bmatrix} \varphi_{11} & \varphi_{12} & \cdots & \varphi_{1L} \\ \vdots & \vdots & \ddots & \vdots \\ \varphi_{L1} & \varphi_{L2} & \cdots & \varphi_{LL} \end{bmatrix} * \begin{bmatrix} w_1 \\ \vdots \\ w_L \end{bmatrix} = \begin{bmatrix} d_1 \\ \vdots \\ d_L \end{bmatrix}$$

Scanned with CamScanner

رابطه‌ی ماتریسی: (بردار مقادیر مطلوب)

interpolation matrix	weight	desired values
Φ	w	d

$$\boxed{w = \Phi^{-1} d}$$

d برای ما معلوم است، اگر φ ها هم معلوم باشند w به دست می‌آید. با مشخص شدن w، f به دست می‌آید.

قضیه Micchelli

فرض کنید مجموعه‌ای از نقاط متمایز $\{X_i, i=1,2,\dots,L\}$ را در فضای n بعدی داریم، در این صورت ماتریس interpolation : $\Phi : L \times L$ که عضو ij ام آن φ_{ij} است: $\varphi_{ij} = \varphi(\|x_i - x_j\|)$ و $i,j = 1,2,\dots,L$

nonsingular (غیرمنفرد) است یعنی معکوس دارد و برای تعیین معکوس Φ در رابطه‌ی ماتریسی که بالا گفته شد مشکلی نداریم.

توابعی که می‌توانند به عنوان تابع RBF انتخاب شوند:

① Multiquadrics $\quad \varphi(r) = (r^2 + c^2)^{1/2} \quad c>0,\ r \in \mathbb{R}$

② Inverse Multiquadrics $\quad \varphi(r) = \dfrac{1}{(r^2+c^2)^{1/2}} \quad c>0,\ r \in \mathbb{R}$

③ Gaussian functions $\quad \varphi(r) = \exp\left(-\dfrac{r^2}{2\sigma^2}\right) \quad c>0,\ r \in \mathbb{R}$

(مثال در اسلاید)

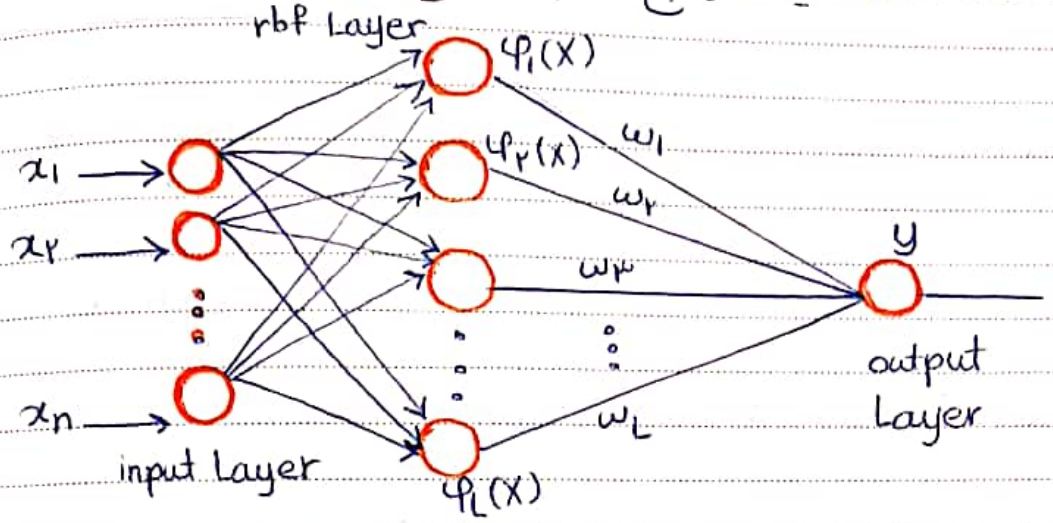
در شبکه RBF ابتدا در لایه‌ی مخفی ای که شبکه دارد (لایه دوم شبکه)، x ها را به فضای Φ می‌بریم

در نتیجه معماری آن به این صورت است: ① یک لایه ورودی داریم که همه‌ی ورودی‌های x را به لایه‌ی بعدی می‌دهد. ② لایه‌ی مخفی که عمل تبدیل به فضای Φ در آن صورت می‌گیرد و خروجی آن به این صورت است: $\varphi_1(x), \varphi_2(x), \dots, \varphi_L(x)$ و بعد این φ ها با یک سری وزن‌هایی باهم

Scanned with CamScanner

ترکیب می‌شوند و خروجی را به ما می‌دهند.

بنابراین این شبکه ساده است و ۳ لایه دارد و تابع f را برای ما تخمین می‌زند



n تا نود در لایه‌ی ورودی داریم چون X، n بُعدی است. L تا نود در لایه‌ی مخفی داریم که L تعداد نمونه‌های آموزشی ماست. یعنی شبکه به تعداد نمونه‌های آموزشی نورون دارد که مرکز تابع φ در هر کدام از نورون‌ها، روی آن داده قرار می‌گیرد.

این توابع σ (سیگما) هم دارند که پهنا را نشان می‌دهد. معمولا σ را در همه نورون‌ها یکسان می‌گیرند.

معمولا چون یک خروجی را می‌خواهیم تخمین بزنیم در لایه خروجی 1 نورون کافی است.

به لایه‌ی مخفی، لایه‌ی RBF هم گفته می‌شود.

$$\varphi_j(X) = \varphi(\|X - X_j\|)$$

$$= \exp\left(-\frac{1}{2\sigma_j^2}\|x - x_j\|^2\right)$$

گفتیم که مراکز توابع RBF روی داده‌ها قرار می‌گیرد و به ازای هر داده یک نورون RBF خواهیم داشت.

اگر داده‌ها زیاد باشند تعداد نورون‌های لایه RBF خیلی زیاد می‌شود و این می‌تواند مشکل‌ساز شود.

از طرفی ممکن است داده‌ها نویزی باشند و باعث شوند نتایج نامربوط و ناخواسته‌ای از شبکه حاصل شوند. انگار داریم به نویزها هم در اهمیت می‌دهیم که به داده‌های اصلی اهمیت می‌دهیم

بنابراین این طرح که به ازای هر داده یک نورون RBF داشته باشیم جالب نیست. پس چه کاری می‌توانیم به جای آن انجام بدهیم؟

Scanned with CamScanner

جواب این است که تعداد نورون های RBF را کمتر از تعداد داده ها بگیریم $(K<L)$

$$\boxed{f(X)=\sum_{j=1}^{K} w_j \varphi(\|X-\mu_j\|)=\sum_{j=1}^{K} w_j \exp\left(-\frac{1}{2\sigma_j^2}\|X-\mu_j\|^2\right)}$$

بنابراین فرمول $f(X)$ به صورت بالا در می آید. j از ۱ تا K جلو می رود. به جای X_j، یه μ ای گذاشتیم

μ مرکز تابع گاوسی است که نمی دانیم μ چیست. اگر φ را تابع گاوسی در نظر بگیریم (که معمولا اینطور هست)

رابطه ی تابع گاوسی را جایگذاری می کنیم که در بالا آورده شده.

در این جا K تا سیگما (σ) و K تا μ داریم که پارامترهای ما هستند. این ها باید مشخص بشوند

تا توابع پایه هم مشخص شوند. بعد از مشخص شدن توابع پایه، وزن ها هم باید مشخص شوند تا مسئله

حل شود. (پس برای حل مسئله μ ها برای K تا نورون، σ_j ها و w_j ها باید تعیین شوند)

یک ایده این است که K تا نورون که توابع RBF را مشخص می کنند در جایی قرار بدهیم که داده ها خوشه شده اند

یعنی هر جا که یک خوشه مشاهده می شود یک نورون قرار بدهیم. بنابراین راه همان

اینکه μ ها چه چیزی باید باشند این است که داده ها چگونه در فضای ورودی خوشه می شوند.

به همین دلیل سراغ الگوریتم K_means می رویم

الگوریتم K میانگین (K_means)

فرض کنید مجموعه نقاطی که در فضای n بعدی حقیقی داریم X_i ها هستند: $\{X_i \in \mathbb{R}^n, i=1,2,\dots,L\}$

می خواهیم این X_i ها را به K تا خوشه، خوشه بندی کنیم. پس تعداد خوشه ها K است و به عنوان ورودی به الگوریتم

داده می شود.

$d(x_i, x_{i'})$ ← معیار شباهت جفت جفت داده ها

یک نگاشت کننده یا encoder در نظر می گیریم که نمونه های i را به یکی از K تا خوشه map می کند

در واقع تعیین می کند که هر نمونه به کدام خوشه باید متعلق باشد و این را بر اساس معیار شباهت

Scanned with CamScanner

انجام می‌دهد.

$j = C(i) \quad i = 1, 2, \dots, L \rightarrow$ many_to_many mapper (encoder)

↳ داده‌ی i ام را به خوشه‌ی j ام assign می‌کند

وقتی خوشه‌بندی انجام شود مرکز خوشه‌ها پیدا می‌شوند و مرکز خوشه‌ها را همان μ ها در نظر می‌گیریم

یعنی برای مراکز توابع RBF، مرکز خوشه‌ها را در نظر می‌گیریم.

الگوریتم K_means سعی دارد تابع زیر را کمینه کند:

$$\boxed{J(C) = \frac{1}{2} \sum_{j=1}^{K} \sum_{C(i)=j} \sum_{C(i')=j} d(x_i, x_{i'})}$$

d را برابر فاصله‌ی اقلیدسی می‌گیریم و می‌خواهیم

مجموع فواصل را حداقل کنیم. به عبارتی وقتی فاصله‌ی بین دو نمونه حداقل شود یعنی شباهتشان حداکثر می‌شود

و این مسئله باعث می‌شود تا نمونه‌های شبیه به هم در یک خوشه قرار بگیرند.

بنابراین هدف ما این است که $J(C)$ را min کنیم $\quad d(x_i, x_{i'}) = \|x_i - x_{i'}\|^2 = \|x_{i'} - x_i\|^2$

با قرار دادن d در $J(C)$ داریم:

$$J(C) = \frac{1}{2} \sum_{j=1}^{K} \sum_{C(i)=j} \underbrace{\sum_{C(i')=j} \|x_i - x_{i'}\|^2}$$

$\hat{\mu}_j \leftarrow$ تخمین بردار مرکز خوشه‌ی j

$$\boxed{J(C) = \sum_{j=1}^{K} \sum_{C(i)=j} \|x_i - \hat{\mu}_j\|^2}$$

بنابراین مجموع فواصل تا مرکز را min می‌کند.

$$\boxed{\hat{\sigma}_j^2 = \sum_{C(i)=j} \|x_i - \hat{\mu}_j\|^2}$$

می‌توان نشان داد با احتساب یک ضریب برابر واریانس است. چون $x_i - \mu$ یک تخمین از واریانس در خوشه‌ی j است پس مجموع واریانس‌ها را داریم حداقل می‌کنیم.

Scanned with CamScanner

الگوریتم K_means :

ابتدا یک مقادیر اولیه برای μ انتخاب می‌کنیم ← تخمین مراکز خوشه‌ها

تکرار قدم های زیر:

برای داده ها از 1 تا L تصمیم بگیر که هر داده به کدام مرکز از همه نزدیکتر است. (نزدیکترین مرکز به هر داده را مشخص کن) و این مقدار را برای این داده ثبت کن: Set $b(i)=q$

و این کار را برای همه ی داده ها تکرار کن.

با انجام این عمل هر کدام از داده ها به خوشه ای تعلق می‌گیرند، حالا x های هر خوشه را میانگین بگیر و اسم آن را μ برای آن خوشه بگذار

انقدر این مراحل را تکرار کن که مراکز دیگر تغییری نکنند ← در این صورت مراکز خوشه ها پیدا می‌شود

الگوریتم K_means در واقع یک نگاشت غیرخطی به ورودی X اعمال می‌کند

توصیه می‌کنند الگوریتم را چندین بار برای مقادیر اولیه مختلف μ اجرا کنید و $J(C)$ را در آن ها مقایسه کنید.

هر کدام $J(C)$ کوچک تری داشت آن را انتخاب کنیم.

بنابراین μ ها را اینگونه حساب می‌کنیم. مسئله دیگر این است که w های لایه آخر که در φ ها ضرب می‌شدند را هم می‌خواهیم محاسبه کنیم.

یک روش ارائه شده که با الگوریتم تکراری متوسط مربع خطا را حداقل کنیم:

$$\Phi(X_i)=[\varphi(X_i,\mu_1),\varphi(X_i,\mu_2),\dots,\varphi(X_i,\mu_K)]^T$$

$$\varphi(x_i,\mu_j)=\exp\left(-\frac{1}{2\sigma_j^2}\|x_i-\mu_j\|^2\right) \quad j=1,2,\dots,K$$

نمونه های آموزشی که برای لایه خروجی به کار می‌گیریم:

$$\{(\Phi(X_i),d_i),\ i=1,2,\dots,L\}$$

$$\{(\Phi(i),d(i)),\ i=1,2,\dots,L\}$$

Scanned with CamScanner

الگوریتم RLS (Recursive least Squares)

با نمونه‌های $(\phi(i), d(i))$ که داریم

$$p(n) = P(n-1) - \frac{P(n-1)\phi(n)\phi^T(n)P(n-1)}{1+\phi^T(n)P(n-1)\phi(n)}$$

این محاسبات را از 1 تا L انجام می‌دهیم:

$$g(n) = P(n)\phi(n)$$

$$\alpha(n) = d(n) - \hat{w}^T(n-1)\phi(n)$$ → $\hat{w}(n-1)$ را داریم، $d(n)$ را هم داریم پس α را داریم

$$\hat{w}(n) = \hat{w}(n-1) + g(n)\alpha(n)$$ → w ای که به دنبال آن هستیم $\hat{w}(n)$ است که بر حسب مقدار قبلی اش و مقادیر g و α محاسبه می‌شود که مقادیر α و g قابل محاسبه اند

↓

این فرمولی است که برای اصلاح w بکار برده می‌شود

در اولین قدم $\hat{w}(0) = 0$ و $P(0) = \lambda^{-1} I$ که λ یک عدد مثبت کوچک است

الگوریتم K_means, RLS

لایه ورودی: بعد لایه ورودی توسط بعد بردار x مشخص می‌شود که n است

لایه مخفی: ① تعداد نورون‌ها در لایه مخفی برابر تعداد خوشه‌ها که K است، می‌باشد

② میانگین خوشه‌ی j ام که $\hat{\mu}_j$ است را با استفاده از الگوریتم K_means که روی نمونه‌های x_i $i=1,2,\dots,L$ کار می‌کند، به دست می‌آوریم. این $\hat{\mu}_j$ ها مرکز توابع φ هستند.

③ انحراف معیار توابع گاوسی

$$\sigma = \frac{d_{max}}{\sqrt{2K}}$$

برای همه‌ی نورون‌های rbf یکسان است. K تعداد خوشه‌ها یا همان تعداد نورون‌های rbf است و d_{max} بزرگترین فاصله‌ی بین جفت جفت مراکز خوشه‌هاست. پس تا این جا μ و σ مشخص شدند و توابع پایه مشخص شدند

لایه خروجی: ورودی این لایه، خروجی نورون‌های مخفی است

$$\phi(x_i) = [\varphi(x_i, \mu_1), \varphi(x_i, \mu_2), \dots, \varphi(x_i, \mu_K)]^T$$

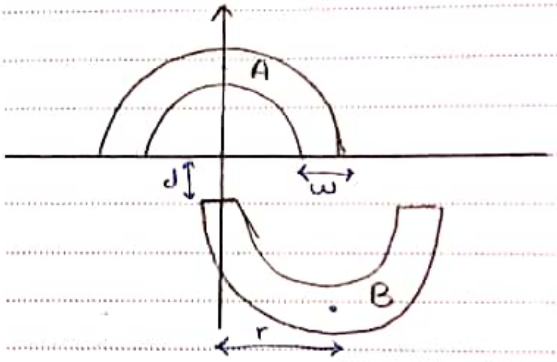
مجموعه‌ی آموزشی که به کار می‌رود $(\phi(x_i), d_i)$ $i=1,2,\dots,L$ است و الگوریتم RLS برای آموزش استفاده می‌شود.

Scanned with CamScanner

یک حُسن این روش این است که محاسباتش خیلی پیچیده نیست و نسبتاً سریع است. خودش شبکه هم ساده است و اینکه دو تا لایه کارشان جدا از هم انجام می‌شود و تداخلی ندارند به سریع کمک می‌کند.

یک سوالی که مطرح می‌شود: ما در مورد لایه rbf و لایه خروجی به صورت جداگانه تصمیم می‌گیریم و یک بهینه‌سازی در کل شبکه انجام نمی‌شود بلکه در هر کدام از این لایه‌ها سعی می‌کنیم جداگانه به صورت بهینه عمل کنیم. ولی به هر حال جواب‌های خوبی می‌دهد این شبکه، سریع است و خوب کار می‌کند.

مثال: مسئله‌ی دسته‌بندی ۲ دسته‌ای، داده‌های دسته A را از ناحیه A و داده‌های دسته B را از ناحیه B برمی‌دارد.



w ← پهنای نیم حلقه
r ← فاصله‌ی مراکز دو تا نیم حلقه
d ← اینکه حلقه‌ی پایینی چقدر زیر محور افقی است

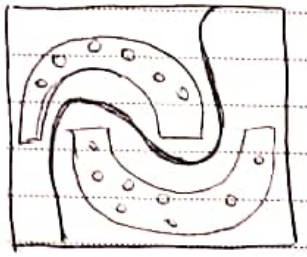
در این مثال: $r=10$ $w=6$
برای d دو حالت را در نظر می‌گیریم: $d=-6$ $d=-5$

تعداد نمونه‌های آموزش: ۱۰۰۰ تعداد نمونه‌های تست: ۲۰۰۰

حالت اول: $d=-5$ معماری این شبکه ← 2_20_1 2 تا ورودی چون داده‌ها ۲ بعدی هستند

۱ خروجی برای اینکه ۲ دسته داریم، ۲۰ تا هم نورون rbf در نظر گرفته. واحدهای مخفی را هم گاوسی در نظر می‌گیریم

تعداد خوشه‌ها = K = ۲۰ پس از اجرای K-means می‌توان σ را محاسبه کرد که 2.6 بدست آمده



برای یافتن وزن‌های w الگوریتم RLS را به کار می‌برد

در این توابع rbf اند، مرکز آن‌ها مرکز تابع گاوسی را نشان می‌دهد و شعاع هم σ را نشان می‌دهد.

حالت دوم $d=-6$ هم مشابه اول است. سپس همین دسته‌بندی را با شبکه MLP هم آموزش داده و معماری را همان 2,20,1 گرفته. learning rate را از 10^{-1} تا 10^{-5} کاهش داده است عملکرد rbf بهتر بوده هم از نظر سرعت هم صحت. (مثال از تخمین تابع توسط rbf در اسلاید آمده)

۱۴۰۰

جلسه شانزدهم - ۱۴۰۰/۸/۲۵

Convolutional NNs

شبکه های کانولوشنی (پیچشی)

مسائلی که تا این جا مطرح کرده ایم: ① دسته بندی (classification)

② تقریب تابع (function approximation)

③ خوشه بندی (clustering)

مسئله ای که الان می خواهیم بحث کنیم: ④ استخراج ویژگی (feature extraction)

استخراج ویژگی اگر به صورت اتوماتیک در یک فرآیند یادگیری صورت بگیرد خیلی ارزشمند است زیرا در گذشته قبل از اینکه بتوانیم استخراج ویژگی را در یک شبکه یاد بگیریم، باید خودمان تصمیم می گرفتیم که چگونه ویژگی استخراج کنیم و ویژگی باید چگونه باشد که برای مسئله ی ما خوب باشد که بحث مفصل و مشکلی است شبکه های کانولوشنی برای استخراج ویژگی مورد استفاده قرار می گیرند و می توانند ویژگی هایی با درجه های تجرید مختلف استخراج کنند. در انتهای شبکه ی کانولوشنی یک دسته بندی کننده هم قرار دارد بنابراین در خروجی این شبکه ما شاهد حل مسئله مان خواهیم بود. هم ناز استخراج ویژگی و هم ناز تصمیم گیری بر اساس آن ویژگی را دارد.

مزیت شبکه کانولوشنی این است که نسبت به انتقال، مقیاس شدن، و اعوجاج در ورودی مقاوم است. در بحث استخراج ویژگی و representation داده ها یکی از هدف ها این است که تا حد ممکن نسبت به یک سری تغییرات غیر حساس باشد. مثلا دو تا تصویر از یک شیء که فقط اندازه شیء در آن ها متفاوت است یا تصویر چرخیده یا کج شده یا ... لازم است که representation یکسانی داشته باشند. آموزش این شبکه به صورت supervised است.

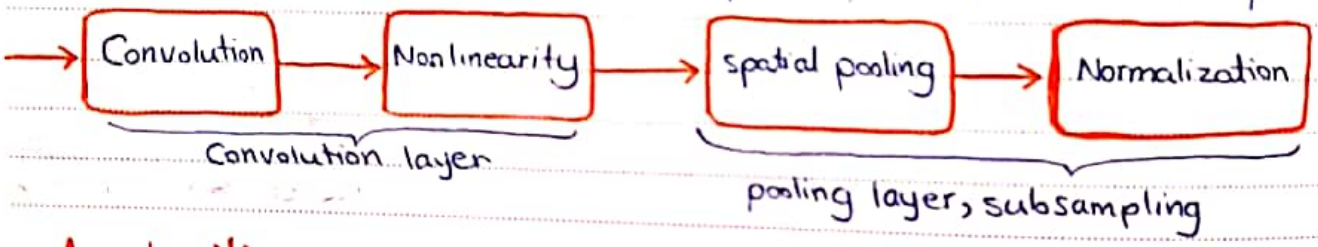
Scanned with CamScanner

معماری شبکه

معماری شبکه دو بخش اصلی دارد: ① استخراج ویژگی ← شامل N تا مرحله است که ویژگی استخراج می‌کند
② دسته‌بندی ← بعد از بخش استخراج ویژگی وجود دارد، مثلاً می‌تواند یک شبکه‌ی MLP باشد. یک شبکه FF ساده مثلاً ۳ یا ۴ لایه.

در بخش استخراج ویژگی که N تا stage داریم وقتی داده‌ی ورودی اعمال می‌شود همین طور که در این مراحل جلو می‌رود، ویژگی‌ای که استخراج می‌کند خردتر می‌شود. وقتی به سمت خروجی می‌رویم درجه‌ی تجرید ویژگی استخراج شده بالاتر می‌رود یا به عبارتی ویژگی استخراج شده سطح بالاتری می‌شود.

هر کدام از stage ها شامل همه یا برخی از فرم‌های زیر است:



ابتدا یک عمل Convolution انجام می‌شود، سپس یک تابع غیرخطی به نتیجه اعمال می‌شود. بعد ادغام مکانی و در آخر Normalization صورت می‌گیرد.

دو مرحله‌ی اول به عنوان یک لایه شناخته می‌شود که به آن لایه‌ی کانولوشن گفته می‌شود. دو مرحله‌ی بعدی به عنوان لایه‌ی ادغام (pooling) یا لایه‌ی subsampling (زیرنمونه برداری) شناخته می‌شود

stage 1: Convolution layer 1 + pooling layer 2

stage 2: Convolution layer 2 + pooling layer 2

⋮

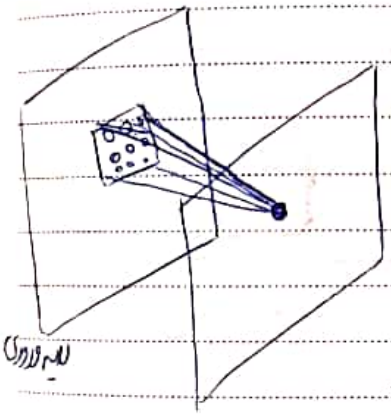
stage N: Convolution layer N + pooling layer N

Scanned with CamScanner

Convolution

در اطراف هر نورون محدوده ای را در نظر می گیرد و در ورودی آن، یک عمل کانولوشن را در آن همسایگی انجام می دهد. همسایگی ای که برای این نورون اثرگذار است به همین دلیل به این همسایگی receptive field می گویند که به معنای محدوده ای است که این نورون می بیند و ورودی های این محدوده روی این نورون اثر می گذارند.

بنابراین در شبکه کانولوشن اینگونه نیست که هر نورون از تمام ورودی ها، ورودی بگیرد. بلکه هر نورون از تعدادی ورودی، ورودی می گیرد. پس در این شبکه اتصالات کامل نیستند بلکه تعدادشان خیلی کمتر از اتصالات کامل است.



$$I_j^{(l)} = \sum_{i \in \text{receptive field}} w_{ij}^{(l)} x_i^{(l-1)} + b_j^{(l)}$$

کانولوشن برای یک نورون

وزن های اتصالاتی که برای هر نورون داریم با سایر نورون ها مشابه است.

برای مثال ماتریس وزن همه ی نورون ها در شکل بالا، ۳×۳ و یکسان خواهد بود. به این ویژگی weight sharing گفته می شود. بنابراین اگر یک مجموعه وزن داشته باشیم برای تمام نورون های صفحه می توانیم I را حساب کنیم. می توان به جای اینکه یکی از این مجموعه وزن ها را داشته باشیم یک تعدادی از این مجموعه وزن ها را داشته باشیم در اثر به کارگیری هر مجموعه یک صفحه ویژگی یافت می شود. مثل این است که ویژگی های مختلفی از ورودی استخراج می شود. این عملی که با یک مجموعه از این وزن ها انجام می دهیم معادل این است که یک فیلتر به ورودی اعمال شود و وزن هایی که استفاده می شد در واقع همسایگی فیلتر را تشکیل می دهند.

مثلا اگر ورودی تصویر باشد، یک فیلتر می تواند برای تشخیص لبه های افقی اشیاء ایجاد شود. یک فیلتر برای

Scanned with CamScanner

لبه های عمودی، یکی برای لبه های با زاویه ۴۵ درجه و ...

مطلب دیگر اینکه خود ورودی می تواند بیشتر از یک ورودی باشد. مثلا اگر تصویر رنگی داشته باشیم ۳ تا کانال دارد ← R, G, B و این عملیات برای هر کانال جداگانه اعمال می شود و تعداد صفحات ویژگی که استخراج می شود ۳ برابر می شود.

فرض کنیم فیلتری که استفاده می کنیم F×F باشد ← انتخاب آن با خود طراح شبکه است

موردی که باید مشخص کنیم این است که فیلتر با receptive field روی ورودی چگونه حرکت کند. یعنی وقتی الان یک receptive field داریم، بعدی چگونه تعیین شود. ممکن است بعدی را بین ۱ تا F خانه جلو ببریم و قرار بدهیم. وقتی فقط ۱ خانه جلو ببریم هم پوشانی زیادی داریم و وقتی F تا جلو ببریم هم پوشانی نداریم اینکه چند قدم ببریم و بعدی را قرار بدهیم stride نام دارد که می تواند 1, 2, 3, ..., F باشد. (اندازه قدم)

وقتی میدان های دریافت هم پوشانی داشته باشند نتایج بهتری حاصل می شود.

هر چقدر stride بزرگتر باشد تعداد کمتری محاسبه نیاز است و در نتیجه در لایه ی کانولوشن تعداد کمتری گره یا نورون خواهیم داشت. (و برعکس) stride کوچکتر به معنای محاسبات بیشتر و نقشه های ویژگی بزرگتر است.

وزن های فیلترها با آموزش supervised یادگرفته می شود. قانون آموزش آن هم BP است.

NonLinearity

روی نتیجه ی کانولوشن یک تابع غیرخطی اعمال می کنیم. این تابع می تواند Sigmoid، Tanh، ReLU و ... باشد. حسن ReLU نسبت به بقیه در این است که آموزش و انتشار خطا به عقب را آسانتر می کند، یادگیری با آن سریعتر صورت می گیرد و توابع دیگر که به اشباع می روند مشکلاتی را ایجاد می کنند که این تابع ایجاد نمی کند.

خروجی لایه کانولوشن ← $\boxed{x_j^{(l)} = f(I_j^{(l)})}$

(مثال در اسلاید)

Scanned with CamScanner

Spatial Pooling

گفتیم که اگر ورودی t تا کانال داشته باشد و k تا صفحه ویژگی بخواهیم استخراج کنیم، t×k تا صفحه خواهیم داشت و تعداد ورودی های لایه ی بعدی می شود tk تصویر و همین طور تعداد این ها در هر لایه زیاد می شود. بنابراین باید از یک راهی این مورد را کنترل کنیم که یکی از وظایف لایه ی ادغام مکانی همین است که حجم این داده ها را کم کند و آن ها را فشرده کند.

لایه ادغام یک ناحیه را می گیرد و به جای مقادیر آن ها یک مقدار جایگزین می کند. این مقدار مثلا می تواند max مقادیر آن ناحیه باشد یا مثلا مجموع یا میانگین آن ها.

به عنوان مثال اگر ناحیه ای که در نظر می گیریم ۲×۲ باشد، هر سلول ۲×۲ در خروجی با ۱ عدد جایگزین می شود. پس هر بعد داده تقسیم بر ۲ می شود و در کل داده ها $\frac{1}{4}$ می شود برای هر کدام از صفحات.

چرا این ناحیه دارد بدون اتصالات کامل، از ناحیه ی قبلی ورودی می گیرد یعنی یک receptive field دارد که همان نورون هایی هستند که از لایه ی قبلی در نظر گرفته می شوند (مثلا ۲×۲)

در این جا هم میدان های دریافت می توانند هم پوشانی داشته باشند یا نداشته باشند. (stride)

معمولا receptive field را ۲×۲ و stride را ۲ می گیرند که هم پوشانی نداشته باشند.

کاری که pooling انجام می دهد به جابه جایی های کوچک غیر حساس می شود چون برای یک ناحیه max یا مجموع و ... را در نظر می گیرد.

نکته دیگر اینکه چون مثلا یک ناحیه ۲×۲ رو در نظر می گیره، در حقیقت مثل این است که میدان های دریافت بزرگتری را در نظر می گیرد و ناحیه ی بزرگتری از ورودی ها دارند روی گسوای که در لایه ی pooling قرار دارد اثر می گذارند.

(مثال از max pooling در اسلاید)

Scanned with CamScanner

منطق max pooling ← در واقع جاهایی که فیلتر پاسخ قوی‌تری می‌دهد برای ما مهم است و این جاها همان max و عددهای بزرگتر هستند. این نقاط هستند که برای استخراج ویژگی مفیدند.

Normalization

برای کنترل مقادیر و اینکه contrast بالا برود یعنی تمایزها بیشتر بشود. یعنی داده‌های خروجی pooling، به گونه‌ای نباشد که خیلی مقادیر متفاوتی پیدا کنند و در بین نتایجی که حاصل شده یک تمایز خوب بتوانیم ایجاد بکنیم.

نکته: اگر ورودی N×N باشد و فیلتر F×F و stride داشته باشیم اندازه خروجی به این صورت محاسبه می‌شود:

$$\boxed{(N-F)/\text{stride}+1}$$

مثال: N=7, F=3, stride=1 — output size $= \frac{7-3}{1}+1 = 5$

N=7, F=3, stride=2 — " " $=$ $= 3$

" " stride=3 — " " $=$ $= 2.33$

این حالت را نباید انتخاب کنیم

نکته: اگر تعداد مراحل در شبکه کانولوشن زیاد باشد چه اتفاقی می‌افتد؟ مثلا اگر ۲×۲ pooling به کار ببریم در مرحله اول اندازه ورودی تقسیم بر ۲ می‌شود، در مرحله دوم دوباره تقسیم بر ۲ می‌شود و $\frac{1}{4}$ می‌شود و مرحله بعدی $\frac{1}{8}$ می‌شود... در حالی که می‌خواهیم حجم داده‌ها را کنترل کنیم در عین حال نمی‌خواهیم داده‌ها نابود شوند.

یک راه این است که در برخی مراحل pooling نداشته باشیم. یعنی ۲ تا لایه کانولوشن و یک pooling و...

یک راه دیگر: چون علاوه بر pooling خود کانولوشن هم داده‌ها را کوچک می‌کند، بیاییم به ورودی سطرها و ستون‌های صفر اضافه کنیم به طوری که اندازه‌ی خروجی کانولوشن برابر اندازه‌ی ورودی اش باشد. به آن zero padding می‌گویند.

Scanned with CamScanner

zero padding کمک می‌کند که مرکز receptive field روی تمام نقاط داده‌ی ورودی قرار بگیرد. به عنوان مثال اگر ورودی 7x7 و فیلتر 3x3 و strid=1 باشد، یک padding با اندازه ۱ مناسب است. یعنی یک سطر و یک ستون صفر اضافه شود.

F=5 → zero pad = 2
F=7 ⇒ zero pad = 3

تعداد سطر و ستون که نیاز داریم $\frac{F-1}{2}$ است.

فرمول‌های کلی‌تر:

ورودی: $W_1 \times H_1 \times D_1$ → تعداد کانال‌ها

تعداد فیلترها: K

اندازه فیلترها: F×F

stride = S

P = تعداد سطر یا ستون صفر

اگر خروجی لایه کانولوشن $W_2 \times H_2 \times D_2$ باشد:

$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

$$D_2 = K$$

$$H_2 = \frac{H_1 - F + 2P}{S} + 1$$

(مثال در اسلاید)

آموزش شبکه:

با الگوریتم BP. این شبکه به مجموعه داده بزرگ نیاز دارد به خصوص اگر تعداد لایه‌ها زیاد باشد. نتیجه این است که آموزش این شبکه بسیار کند است.

بخشی از مسائل مشکل ما این است که این حجم داده را اصلا نمی‌توانیم داشته باشیم (لیبل هم باید داشته باشد) یک راه برای بالا بردن تعداد داده‌ها این است که همین داده‌های موجود را تغییر بدهیم و به عنوان داده جدید اضافه کنیم. مثلا برای تصویر: چرخش، بازاریه‌های مختلف.

یک کاری که انجام شده این است که افرادی که به سیستم‌های خیلی قوی دسترسی داشته‌اند شبکه‌هایی را آموزش داده‌اند و در اختیار عموم گذاشته‌اند. حتی برای مجموعه داده‌های متفاوت هم تا حدی می‌تواند کمک کند.

Scanned with CamScanner

بنابراین از شبکه‌های پیش آموزش دیده (pre_trained Network) استفاده می‌کنیم. که عدتاً آموزش را دیده‌اند، این‌ها را با داده‌های خودمون دوباره آموزش (re-training) می‌دهیم و به مقداری برای داده‌های ما آموزششان بهتر می‌شود که این کار تحت عنوان Transfer learning یاد می‌شود. که به این معناست که شبکه‌ای با یک مجموعه داده آموزش دیده و حالا در مسئله‌ی دیگری با مجموعه داده دیگری مورد استفاده قرار می‌گیرد.

جلسه هفدهم ـ ۱۴۰۰/۸/۳۰

تابع زیان در این شبکه، تابع cross entropy است:

$$J(\theta)=\sum_i y^{(i)} \log(\hat{y}^{(i)})$$

مثلاً وقتی بخواهم روی ارقام دست‌نویس فارسی کار کنم بهتر است از شبکه‌ای که با MNIST پیش آموزش دیده کار کنم یا مثلاً برای تشخیص مکان‌ها خوب است با شبکه‌ای که با دیتاست places آموزش دیده کار کنم.

مثال: شناسایی حروف دست‌نویس

لایه ورودی، لایه‌ای است که تصویر یک حرف به آن داده می‌شود

یک ماتریس ۲۸×۲۸ است، تصویر حرف در وسط تصویر است و از نظر اندازه هم حروف نرمالیزه شده‌اند

از کانولوشن فیلتر برای استخراج ویژگی استفاده می‌کنیم، هر فیلتر ویژگی مشخصی را استخراج می‌کند.

ابعاد خروجی فیلترها ۲۴×۲۴ است زیرا receptive field = 5×5

به اندازه‌ی $\frac{F-1}{2}$ از ابعاد تصویر کم می‌شود

در مرحله‌ی اولی که استخراج ویژگی انجام می‌شود یک pooling صورت می‌گیرد که ۲×۲ است. در نتیجه ۴ تا نقشه‌ی ۲۴×۲۴ تبدیل به ۴ تا نقشه‌ی ۱۲×۱۲ می‌شود.

Scanned with CamScanner

در مرحله‌ی بعد کانولوشن ۵×۵ صورت می‌گیرد که باعث می‌شود ۴ تا از بعد کم بشود.
۱۲ تا نقشه‌ی ۸×۸ خواهیم داشت، بعد از pooling، ۱۲ تا نقشه‌ی ۴×۴ خواهیم داشت.
بعد از این مرحله، این ۱۲ تا نقشه را به عنوان ورودی به لایه‌ی out put می‌دهیم. بعد از یک کانولوشن
صورت می‌گیرد که ۴×۴ است.
۲۶ تا نقشه‌ی ۱×۱ می‌خواهیم خروجی output باشد. ۲۶ تا عدد که اولین عدد احتمال اینکه ورودی A باشد،
دومی احتمال اینکه B باشد و ... ۲۶ امی احتمال اینکه Z باشد را نشان می‌دهد.
این شبکه حدود ۱۰۰ هزار تا اتصال دارد. به دلیل ویژگی اشتراک گذاری وزن ها، این اتصالات به
۲۶۰۰ کاهش پیدا می‌کند.

popular CNNs

① LeNet: اولین پیاده سازی از CNN را انجام داد. برای کاربرد خواندن ارقام و Zipcode و ... استفاده شد.

② AlexNet: توسط یک تیم ۳ نفره معرفی شد. در سال ۲۰۱۲ در یک مسابقه این شبکه را شرکت دادند و مشخص شد شبکه شان به مراتب بهتر از بقیه است. همان LeNet بود که تغییراتی در آن داده بودند

③ ZF Net: یک نسخه دیگر از CNN ارائه دادند که بهبودهایی در AlexNet داده بودند.

④ GoogleNet: در ۲۰۱۴ این شبکه برنده شد که توسط فردی که در گوگل کار می‌کرد معرفی شد. از ماژول‌هایی به نام Inception استفاده کرد که تعداد پارامترها را کاهش می‌داد. مثلا این شبکه

⑤ VGGNet: در همان سال 2014 این شبکه دوم شد. در دانشگاه oxford این شبکه توسعه داده شده بود. کاری که کردند این بود که عمق را زیاد کردند و نشان دادند که افزایش عمق تاثیر زیادی دارد. علاوه بر این، با اینکه این شبکه به GoogleNet باخت اما نشان داده شد که این شبکه ویژگی های بهتری

Scanned with CamScanner

را استخراج می‌کند به همین دلیل در Transfer learning خیلی مفید است و هنوز هم افراد ترجیح می‌دهند که از VGG استفاده کنند. هزینه‌ای که این شبکه دارد تعداد پارامترهای زیاد آن است.

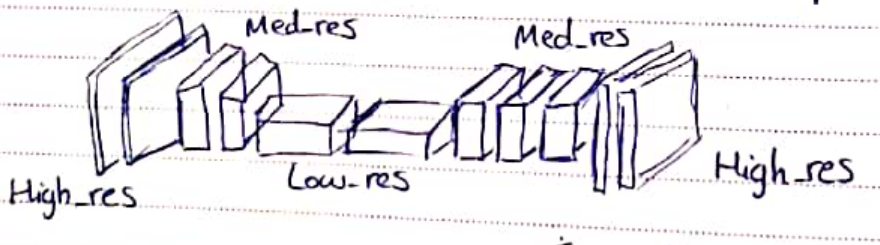
در نسخه‌های اولیه، شروع به استفاده از CNN شد و نسخه‌های جدیدی هم ارائه شد. مثلا یک نسخه R_CNN (Region CNN) است که در واقع روی نواحی تصویر کار می‌کند و مثلا برای یافتن شرح تصویر به کار گرفته شده.

کاربردهای رگرسیون و تقطیع تصویر، object detection، عنوان نویسی برای تصاویر

Semantic Segmentation : FCNs

FCN : Fully Connected ابتدا رزولوشن را کاهش می‌دهد بعد دوباره افزایش می‌دهد.

برای Semantic Segmentation هم استفاده شده که پیکسل‌های مربوط به هر شیء یا ناحیه از تصویر را به یک رنگ خاص در می‌آورد.



ماشین‌های خودران از Segmentation استفاده می‌شود

علاوه بر Semantic Segmentation ممکن بخواهیم عمق را مشخص کنیم (تخمین بزنیم) یا ممکن بخواهیم نمونه‌های از یک شیء را که در تصویر وجود دارد جداگانه تشخیص بدهیم که به آن Instant Segmentation می‌گویند

در Image Captioning یک شبکه CNN تصویر را تحلیل می‌کند و نتیجه را به یک RNN می‌دهد که کپشن را تولید می‌کند

در کاربردهای پزشکی هم استفاده می‌شود. مثلا از روی تصویر چهره اختلالات را تشخیص می‌دهند. خروجی شبکه احتمال ابتلای فرد به بیماری‌های مختلف را نشان می‌دهد.

Scanned with CamScanner

Convolution over volumes

یک تصویر RGB سه تا کانال دارد. اگر بخواهیم کانولوشن انجام بدهیم باید فیلتر را روی هرکدام از کانال‌ها قرار بدهیم و عمل کانولوشن را انجام بدهیم.

یک راه دیگر این است که تعداد کانال‌ها را به عنوان مولفه سوم تصویر در نظر بگیریم و تصویر حجم داشته باشد فیلترها را هم می‌توانیم به صورت حجم در نظر بگیریم که مولفه‌ی سوم آن همان تعداد کانال‌هاست مثلا یک فیلتر ۳×۳ برای ۳ کانال می‌شود یک حجم ۳×۳×۳

حجم فیلتر در داخل حجم تصویر قرار می‌گیرد و حرکت می‌کند و کانولوشن محاسبه می‌شود.

اگر بیشتر از یک فیلتر داشته باشیم، هرکدام از فیلترها وقتی با تصویر convolve می‌شوند یک صفحه در خروجی ایجاد می‌کنند. مجددا صفحات خروجی را می‌توان به صورت حجم در نظر گرفت که مثلا بعد سوم آن ۲ است

فرض کنید ورودی $n \times n$ و تعداد کانال‌ها n_c باشد:

ورودی: $n \times n \times n_c$

فیلتر $f \times f$ است:

فیلتر: $f \times f \times n_c$

ابعاد صفحات خروجی $n-f+1$ است:

خروجی: $(n-f+1) \times (n-f+1) \times n_c'$

این جا اضافه کردن ۰ نداریم

n_c' ← تعداد کانال در خروجی = تعداد فیلترها

جلسه هجدهم - ۱۴۰۰/۹/۲

وقتی کانولوشن ۱×۱ را با عمق تعداد کانال‌ها استفاده کنیم تصویر حاصل انگار خلاصه‌ای از همه‌ی کانال‌ها را در خودش دارد. انگار از همه‌ی کانال‌ها را با هم ترکیب کرده و عصاره‌ای از همه‌ی آن‌هاست.

وقتی کانولوشن ۱×۱ را حرکت می‌دهم در واقع ترکیب وزن‌دار پیکسل‌ها در کانال‌های مختلف محاسبه می‌شود و در خروجی ظاهر می‌شود

Scanned with CamScanner

لایه سوم فیلتر ۱×۱ در اینجا با تعداد کانال‌ها است

تعداد صفحه‌ها در خروجی ← تعداد فیلترها

اگر بخواهیم از یک فیلتر ۱×۱ با عمق تعداد کانال‌ها داشته باشیم، صفحات خروجی هم بیش از ۱ می‌شود

به تعداد فیلترها، صفحه‌ی خروجی داریم.

← مشکل دسترسی اطلاعات است، اطلاعات ۱۹۲ کانال در ۳۲ کانال خلاصه می‌شود

هم چنین برای کاهش تعداد کانال می‌توان از کانولوشن ۱×۱ استفاده کرد. به عنوان مثال تصویر ۲۸×۲۸

با ۱۹۲ کانال داریم: ۲۸ × ۲۸ × ۱۹۲ ← می‌خواهیم تعداد کانال‌ها را به ۳۲ کاهش دهیم

پس از ۳۲ تا فیلتر ۱×۱ استفاده می‌کنیم: ۱×۱×۱۹۲ ← خروجی: ۲۸×۲۸×۳۲

کانولوشن عمقی جداپذیر

تصویر: $D_F \times D_F \times M$

فیلتر: $D_K \times D_K \times M$ $\xrightarrow[N \text{ filters}]{\text{Conv}}$ خروجی: $D_G \times D_G \times N$

D_G از روی D_F و D_K محاسبه می‌شود

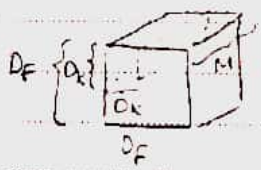
محاسبه‌ی تعداد ضرب‌ها:

در حالت استاندارد

$$\text{Mults ones} = D_K^2 \times M$$

$$\text{Mults per kernel} = D_G^2 \times D_K^2 \times M$$

$$\text{Mults N kernel} = N \times D_G^2 \times D_K^2 \times M$$



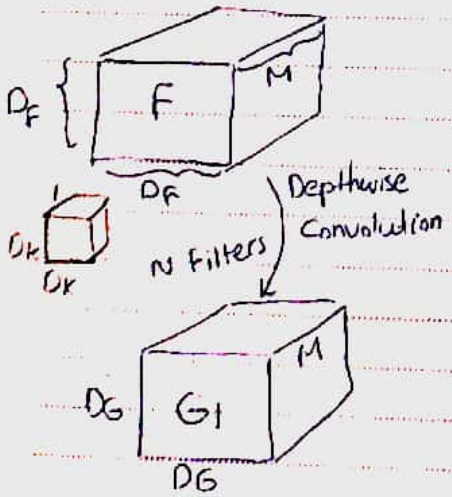
کانولوشن جداپذیر عمقی:

در این جا فیلتر را $D_K \times D_K \times 1$ در نظر می‌گیریم و کانولوشن را ابتدا

برای این فیلتر انجام می‌دهیم. تعداد دفعاتی که فیلتر در آن می‌گنجید

$D_G \times D_G$ است و یک صفحه $D_G \times D_G$ در خروجی نوا ایجاد می‌شود

اگر M تا فیلتر داشته باشیم M تا از این صفحات در خروجی داریم

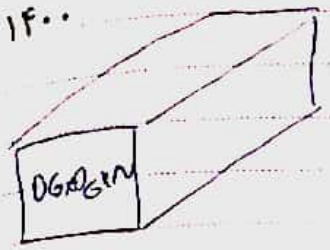


در ادامه‌ی این کار یک کانولوشن ۱×۱ عمقی هم انجام می‌دهیم.

به این صورت که G1 را که نتیجه‌ی مرحله‌ی قبل بود با فیلترهای ۱×۱×M کانوالو می‌کنیم. در اثر یک

Scanned with CamScanner

۱۴۰۰



برای انجام این کار، یک صفحه‌ی $D_G \times D_G$ خواهیم داشت و اگر N تا از این فیلترها داشته باشیم می‌شود $D_G \times D_G \times N$

همان نتیجه‌ی قبلی به دست می‌آید. با این تفاوت که این جا ۲ مرحله داریم. یک بار کار اعمال فیلتر در تک تک انجام می‌شه و بعد کانال‌ها با کانولوشن ۱×۱ ترکیب می‌شوند

محاسبه‌ی تعداد ضرب‌ها در این حالت:

$\text{Mults ones} = D_K^2$

برای یک کانال ← $\text{Mults 1 channel} = D_G^2 \times D_K^2$

تعداد ضرب‌های مرحله اول ← برای M کانال ← $\text{DC Mults} = M \times D_G^2 \times D_K^2$

مرحله دوم:

وقتی یکی از فیلترها در M تا ضرب داریم $\text{Mults once} = M$ (Convolve می‌شود)

$\text{Mults 1 kernel} = D_G \times D_G \times M$

$\text{PC Mults} = N \times D_G \times D_G \times M$

تعداد کل ضرب $= \text{DC Mults} + \text{PC Mults} = M \times D_G^2 \times D_K^2 + N \times D_G^2 \times M$

مقایسه حالت جدایی پذیر عمق و حالت استاندارد:

$$\frac{M \times D_G^2 (D_K^2 + N)}{N \times D_G \times D_G \times D_K \times D_K \times M} = \frac{D_K^2 + N}{D_K^2 \times N} = \frac{1}{N} + \frac{1}{D_K^2}$$

مثلا اگر $D_K = 3$ و $N = 1024$ ⇐ 0.112

تعداد ضرب‌های این روش حدود یک دهم تعداد ضرب‌های روش استاندارد می‌شود (در این مثال) افزایش سرعت قابل توجهی به دست می‌آید

Scanned with CamScanner

محاسبه‌ی مقدار پارامترها

param 1 kernel $= D_k^2 \times M$

param N kernels $= N \times M \times D_k^2$

مشخص می‌شود که مقدار ضرب‌ها بود

یکی از کاربردهای شبکه‌های کانولوشنی در شبکه‌های Multi Modal است. یعنی شبکه‌هایی که ورودی و خروجی‌شان نوعاً مختلف باشند.

مثلاً ورودی تصویر باشد صوت هم باشد.

Xception

MobileNet ← مثال این بودند که شبکه کانولوشنی را کاهش حجم دهند تا در موبایل به کار گرفته شود. عملکرد دقت پیدا می‌کند اما حجم را کاهش می‌دهند که روی موبایل بتواند کار کند.

underfitting ← در مسئله‌ی رگرسیون یا کلاسیفیکیشن با این وضعیت روبه‌رو شویم که مدل تعمیم نتوانسته اونطور که باید منعطف بشه high bias

overfitting ← تغییرات مدل خیلی زیاد است، داده‌ها را خوب جدا کرده ولی تغییراتش خیلی بالاست نویز روی مدل تعمیم خیلی اثر دارد high variance

Scanned with CamScanner

Bias, Variance

high variance ← overfitting

high bias ← underfitting

برای ما هیچ کدام از این دو حالت مطلوب نیست

Training Set error	1 %	15 %	15 %	0.5 %
Testing Set error	11 %	16 %	30 %	1 %
	overfitting واریانس بالا	underfitting بایاس بالا خطا بالاست	بایاس بالا واریانس بالا	بایاس پایین واریانس پایین

مطلوب ✓

چگونه می‌توان در یک حالت هم بایاس بالا هم واریانس بالا باشد؟

در طراحی شبکه هم موقعی که داریم آموزش می‌دهیم باید این سوال را بپرسیم

براساس خطای داده‌های آموزش مشخص می‌شود که بایاس بالا هست یا نه و بعد براساس خطای داده‌های تست آیا واریانس بالا هست یا نه؟ اگر نیست خوبه.

اگر بایاس بالا نیست

اگر بایاس بالا بود

برای پایین آوردن بایاس می‌توان شبکه را بزرگتر کرد یا مدت طولانی‌تری آموزش را ادامه دهید و یا اینکه معماری شبکه را عوض کنید.

اگر واریانس بالا باشد توصیه می‌شود ① سعی کنید داده‌های بیشتری به کار ببرید

② Regularization

③ معماری شبکه را تغییر دهید

مسئله‌ی Memorization و Generalization را در شبکه‌های FF بررسی کردیم و زمانی که شبکه کم‌عمق بود این مسئله را داشتیم. این مسئله در شبکه‌های عمیق حل شده و ما می‌توانیم حالتی را داشته باشیم که در عین اینکه در آموزش خطا پایین، خطای تست هم پایین بیاید. این یکی از دلایل موفقیت شبکه‌های عمیق است.

Scanned with CamScanner

Regularization (تنظیم)

تغییری در تابع هدف می‌دهیم و عملکرد روش ما را تغییر می‌کند.

تعریف Regularization: هر تغییری که در الگوریتم یادگیری می‌دهیم به طوری که خطای تعمیم‌پذیری را (بدون اینکه بدانیم) پایین بیاوریم (با این هدف که) خطای آموزش تأثیر منفی بگذارد.

Generalization یا خطای روی داده تست

خیلی وقت‌ها مدلی که پیدا می‌کنیم یک مدل بزرگ هست که توانسته خطای آموزش را پایین بیاورد و به طور مناسبی عمل Regularization روی آن انجام شده است.

روش‌های Regularization:

کاری که در Regularization انجام می‌شود به این شکل است:

$$\tilde{J}(\theta; X, y) = J(\theta; X, y) + \alpha \Omega(\theta)$$

پارامترها — ورودی

یک جمله‌ای با تابع هدف جمع می‌شود

که این جمله معمولا یک ضریب ثابت هست ضرب پارامترهای مسئله (تابعی از پارامترها)

α که معمولا یک عدد مثبت وزن جمله‌ی Regularization را نسبت به تابع بهینه‌سازی اصلی کنترل می‌کند

وقتی $\tilde{J}$ را min می‌کنیم هم J min می‌شود و هم $\Omega(\theta)$ ـ به نوعی پارامترها را کنترل می‌کند. معمولا داخل Ω وزن‌ها را قطعی گذاریم و بایاس‌ها را رها گذاریم که اثر منفی دارد

L^2 Regularization (weigh decay)

$$\Omega(\theta) = \frac{1}{2}\|w\|_2^2 \qquad \tilde{J}(w; X, y) = \frac{\alpha}{2} w^T w + J(w; X, y)$$

نرم 2 وزن‌ها در $\tilde{J}$ قرار می‌گیرد

$$\nabla_w \tilde{J}(w; X, y) = \alpha w + \nabla_w J(w; X, y)$$

$$w \leftarrow w - \epsilon(\alpha w + \nabla_w J(w; X, y)) \Rightarrow w \leftarrow (1 - \epsilon\alpha) w - \epsilon \nabla_w J(w; X, y)$$

Scanned with CamScanner

L1 Regularization

$$\Omega(\theta) = \|w\|_1 = \sum_i |w_i|$$

از نرم L1 وزن‌ها استفاده می‌کند

$$\tilde{J}(w; X, y) = \alpha \|w\|_1 + J(w; X, y)$$

$$\nabla_w \tilde{J}(w; X, y) = \alpha \, \mathrm{sign}(w) + \nabla_w J(X, y; w)$$

منظور از sign، تابع sign به صورت element wise اعمال می‌شود

وقتی از L1 استفاده کنیم به سری وزن‌ها در نتیجه به اجبار صفر شده‌اند. پاسخ به دست آمده اسپارس است به خصوص اگر α بزرگ باشد

مقدار زیادی صفر

وقتی یک سری وزن‌ها صفر باشند یعنی به سری اتصالات نباشند، یعنی ورودی‌های مربوط به این اتصالات هم می‌توانند نباشند. یعنی آن ورودی‌هایی که صفر نیستند خوب و کافی هستند، ویژگی‌های خوبی هستند و اونایی که اتصالاتشون صفره می‌تونن حذف شن. به جور Feature Selection انجام میده

Data Augmentation

برای آموزش شبکه‌های زیاد لازم داریم. همیشه در عمل نمی‌شه داده زیاد داشته باشیم به دلیل فنی یا اقتصادی چون لیبل زدن داده در بعضی مسائل خیلی هزینه‌بره.

توصیه می‌شه از روی داده‌هایی که دارید داده‌های جدیدی بسازید و به کار ببرید. مثلا در تشخیص اشیاء یا در رگرسیون. در مورد داده‌های تصویری مثلا می‌دونیم شیء رو اگر ازش عکس بگیریم یا مثلا با درجات مختلف چرخش اگر بگیریم یا crop کنیم

در مسئله‌ای مثل تشخیص موضوع این کار را می‌توانیم انجام بدهیم. در این موارد یک راه این است که یک نویز مناسب به داده‌ها اضافه کنیم و نویز را جزو داده در نظر بگیریم اگر خوب اعمال شده باشد.

Scanned with CamScanner

نکته: می‌توانیم چند نوع Regularization را با هم به کار ببریم

مرداد ۲۶ سه‌شنبه | 17 Aug 2021 | ۸ محرم ۱۴۴۳

early stopping

یعنی خطاهای training set و validation set همین طور که جلو می‌رویم کنترل کنیم

در ابتدا مشاهده می‌کنیم که هر دو دارند کاهش پیدا می‌کنند اما از یه جایی به بعد خطای ولیدیشن شروع به افزایش می‌کند. همین جا آموزش رو متوقف می‌کنیم

خود early stopping هم نوعی Regularization است. در الگوریتم Regularization به روش early stopping برای یک مدت چک می‌کند که خطای Validation افزایش پیدا نکند. اگر برای مدت مشخصی افزایشی باشد آموزش را متوقف می‌کند.

برای early stopping نیاز به validation set داریم یعنی کمتری از داده‌ها را شبکه ندیده و اگر آن‌ها را ببیند ممکنه عملکردش بهتر بشه.

یک راه این است که پس از پایان آموزش، آموزش را با داده‌های validation ادامه بدهیم. تا شبکه داده‌هایی که ندیده بود را ببیند. می‌توانیم به این شکل انجام بدهیم:

شبکه را با دوباره از اول با همه‌ی داده‌های آموزش بده با همان تعدادی که early stopping تعیین کرده بود.

یک راه دیگه: دوباره از اول آموزش ندهیم بلکه آموزش را ادامه بدهیم و فقط این داده‌های جدید را به شبکه بدهیم و این آموزش را آنقدر ادامه بدهیم که مقدار خطا به زیر مقدار متوسط خطا در داده‌ی ولیدیشن برسد

Scanned with CamScanner

Dropout (حذف تصادفی)

برای هر نورون یک سکه پرتاب می‌کنیم که آن را نگه داریم یا نه!

در این بخش از آموزش (در این دوره)

هر ورودی که می‌آید به صورت تصادفی تصمیم می‌گیریم یک بخش‌هایی از شبکه در آموزش استفاده نشود

می‌تواند برای همه یا برخی از لایه‌ها انجام شود. احتمال حذف برای لایه‌های مختلف می‌تواند متفاوت باشد.

مشابه عملکرد L2 Regularization را دارد. نزدیک به آن بها است. کار آسانی است و هزینه کمی دارد

Dropout در شبکه‌های دیگر مثل بولتزمن، RNN، FF می‌تواند به کار برود

از طرفی روش ارزانی هست و محاسبات سنگین ندارد.

مثال از Dropout در شبکه CNN:

$$a_j^{(l+1)} = pool(a_1^{(l)}, \dots, a_i^{(l)}, \dots, a_n^{(l)}) \qquad i \in R_j^{(l)}$$

اتفاقی که می‌خواهیم روی آن‌ها pooling انجام بدهیم را در یک $m^{(l)}$ ضرب می‌کنیم و اسمش را $\hat{a}$ می‌گذاریم و بعد pooling را روی آن انجام می‌دهیم.

m^l در واقع یک ماسک باینری که هر عضوش از یک توزیع برنولی گرفته شده، یعنی به صورت تصادفی یک سری از این مقادیر صفر می‌شوند و چون max pooling انجام می‌دهیم حذف می‌شوند و انتخاب نمی‌شوند

$$\hat{a}^{(l)} \sim m^{(l)} * a^{(l)}$$

$$a_j^{(l+1)} = pool(\hat{a}_1^{(l)}, \dots, \hat{a}_i^{(l)}, \dots, \hat{a}_n^{(l)}) \qquad i \in R_j^{(l)}$$

در لایه کانولوشن هم Dropout به همین صورت با ماسک باینری انجام می‌شود

$$m_k^{(l)}(i) \sim Bernouli(p) \qquad z_j^{(l+1)} = \sum conv(w_j^{(l+1)}, \hat{a}_k^{(l)})$$

$$\hat{a}_k^{(l)} = a_k^{(l)} * m_k^{(l)} \qquad a_j^{(l+1)} = f(z_j^{(l+1)})$$

در فاز تست ← در هر لایه پولینگ و کانولوشن

$$a_j^{(l+1)} = p \times max(a_1^{(l)} \dots a_i^{(l)} \dots a_n^{(l)})$$

p ← احتمال اینکه خروجی واحد فعال بماند

Scanned with CamScanner

جلسه نوزدهم - ۱۴۰۰/۹/۷

صحبت در مورد شبکه های کانولوشنی را به صورت غیر رسمی ادامه می دهیم. در حوزه این شبکه توسعه هایی داشته ایم و کارهایی در جهت بهبود آن ها انجام شده است. از جمله آن ها: شبکه ی باقی ماندهای

Deep Residual Neural Networks (ResNets)

در شبکه ی کانولوشنی اگر تعداد لایه ها خیلی زیاد شود و شبکه خیلی عمیق شود، آموزش آن دچار مشکل می شود. شبکه ResNet برای حل این مشکل معرفی شد.

عملکردی که این شبکه در کاربردهای مختلف مثل دسته بندی تصاویر، تشخیص اشیاء، تقطیع موضوعی تصاویر از خودشان داده، بهترین بوده است.

ایده ای که پیشنهاد دهنده های این شبکه داشتند این بود که با افزایش تعداد لایه ها قدرت توصیف و ارائه ای که شبکه پیدا می کند قوی تر می شود و هم چنین قدرت تعمیم هم بالاتر می رود.

سوالی که در این جا مطرح شد این بود که تا چند لایه اگر افزایش بدهیم نتایج بهتر می شود؟ آیا حدی دارد؟

جواب این بود که به حدی دارد و از یک جایی به بعد عملکرد افت می کند. شبکه ای که خودشان در مسابقات ارائه دادند ۱۵۲ لایه داشت.

منظورمان از لایه، لایه های کانولوشنی است یعنی batch Normalization و pooling و ... را نشمردیم.

vanishing / exploding gradient problems

مسئله ای که گفتیم در شبکه کانولوشنی وجود دارد و در شبکه ResNet حل شده این است: وقتی شبکه FF

Scanned with CamScanner

۱۴۰۰

با لایه های خیلی زیاد داریم، آموزش این شبکه ها با روش BP صورت می گیرد که از گرادیان استفاده می کند، در این حالت به دلیل وجود تعداد لایه های زیاد آموزش شبکه دچار مشکل می شود.

مشکلی که ایجاد می شود بیشتر به این شکل است:

لایه های ابتدایی یعنی لایه هایی که به ورودی نزدیکتر هستند، آموزششان خیلی کند می شود. خیلی دیر یاد می گیرند یا به عبارتی تغییرات وزن هایشان خیلی کند می شود.

دلیل این مسئله این است:

همان طور که قبلا گفتیم اصلاح وزن ها متناسب با گرادیان خطا صورت می گیرد در قانون BP.

فرمول اصلاح وزن ها:

$$w_{ij}^{new(l)} = w_{ij}^{old(l)} + \eta\, \delta_{pj}^{(l)}\, y_{pi}^{(l-1)}$$

$$\delta_{pj}^{(l)} = f_j'(I_{pj}^{(l)}) \cdot \sum_k \delta_{pk}^{(l+1)} \cdot w_{jk}^{(l+1)}$$

$$-\frac{\partial E_p}{\partial w_{ij}^{(l)}} = -\nabla_w E_p = \delta_{pj}^{(l)}\, y_{pi}^{(l-1)}$$

وقتی در شبکه توابع فعالیتی مانند Tanh داریم که تغییراتشان در یک محدوده ی خاصی است و در یک نواحی هم اشباع می شوند یعنی تغییراتشان خیلی جزئی می شود. گرادیان این توابع در محدوده ۱- و ۱ قرار می گیرد. وقتی تعداد زیادی لایه داریم، خطای خروجی لایه به لایه در گرادیان ها که اعداد کوچکتر از ۱ هستند ضرب می شود و به سمت ورودی می آید و باعث می شود هر چه به سمت ورودی می آییم این عدد کوچکتر شود.

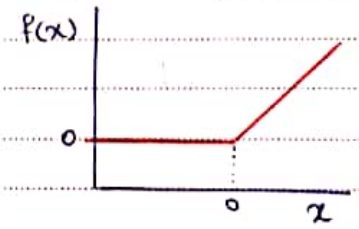
بخاطر گرادیانی که در لایه های ابتدایی به کار می رود بسیار جزئی است در نتیجه اصلاحی که در وزن ها صورت می گیرد بسیار جزئی خواهد بود یعنی جمله ی $\eta \delta y$ در لایه هایی که به ورودی نزدیکند خیلی جزئی

Scanned with CamScanner

خواهد بود. به طوری که انگار اصلا یادگیری صورت نمی‌گیرد. به این مشکل vanishing gradient problem گفته می‌شود. یعنی گرادیان آنقدر ناچیز می‌شود که انگار از بین می‌رود.

اگر از تابع فعال‌سازی استفاده کنیم که گرادیانش بزرگتر از ۱ باشد، همینطور که به سمت ورودی می‌آییم یک سری عدد بزرگتر از ۱ در هم ضرب می‌شوند و وقتی به لایه‌های اولیه می‌رسیم عددی که از این حاصل‌ضرب‌ها ایجاد شده عدد خیلی بزرگی است. بزرگی این عدد باعث می‌شود در محاسبات وزن‌ها ناپایداری به وجود بیاید که به آن exploding gradient problem گفته می‌شود. که این مسئله هم آموزش را دچار مشکل می‌کند.

Hinton پیشنهاد داد که مشتق نه بزرگتر از ۱ و نه کوچکتر از ۱ باشد و دقیقاً خود ۱ باشد در نتیجه تابع ReLU را پیشنهاد داد.



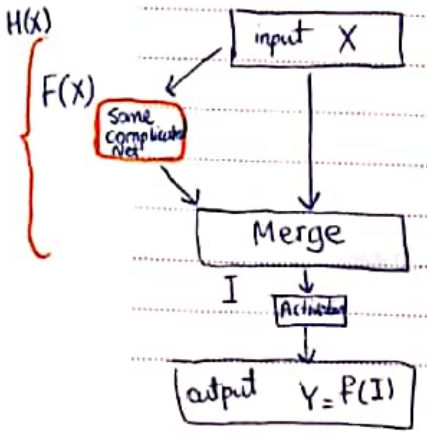
$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$$

$$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$$

ReLU کمک می‌کند که این مشکل حل شود

معماری ResNET

این شبکه از تعدادی بلوک تشکیل شده که به هر کدام از آن‌ها یک بلوک پایه‌ی ResNet گفته می‌شود. (ResNet Basic Block). ساختار بلوک‌ها به این صورت است:



ورودی وارد یک بخشی متشکل از تعدادی عملیات می‌شود. خود این بخش یک شبکه است، مثلا می‌تواند تعدادی لایه کانولوشنی داشته باشد. تا بعضی که این شبکه پیاده می‌کند را F(X) می‌نامیم. ورودی و F(X) با هم ترکیب می‌شوند و به نتیجه‌ی آن یک تابع فعالیت اعمال می‌شود و خروجی بلوک را تولید می‌کند.

$$\boxed{Y = f(m(X, F(X)))}$$

m ترکیب بین X و F(X) را نشان می‌دهد:

Scanned with CamScanner

مثلا اگر merge عمل جمع کردن و f یعنی تابع فعال سازی بلوک، ReLU باشد، داریم:

$$Y = ReLU(X + F(X))$$

حالا اگر نتیجه‌ی بعد از merge را با H(X) نشان بدهیم داریم: $Y = ReLU(H(X))$

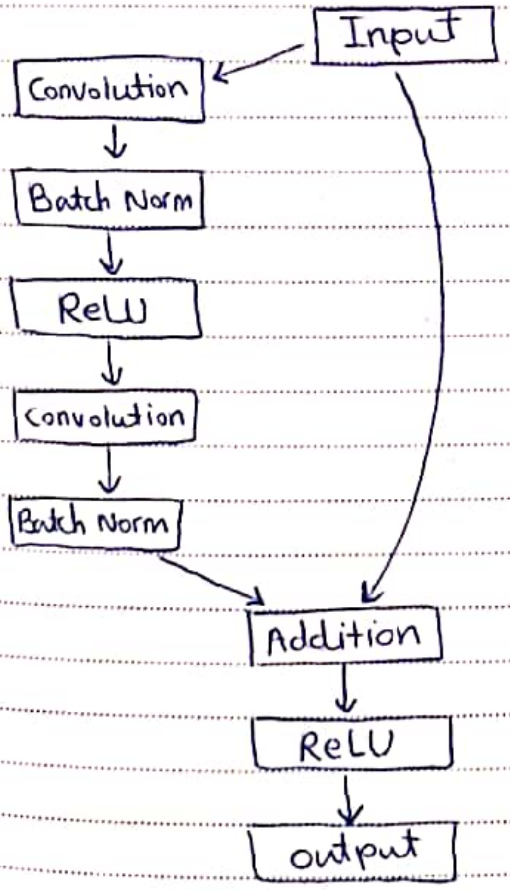
در این آموزش به دلیل ساختاری که انتخاب کرده‌ایم در حقیقت F(X) را یاد می‌گیریم که برابر است با:

$$\boxed{F(X) = H(X) - X}$$

یعنی باقی‌مانده‌ی این تفاضل را در حین آموزش یاد می‌گیریم

به همین دلیل اسم Residual Neural Net را روی آن گذاشته‌اند.

بلوک F(X):



یک نمونه اولیه از آن به این صورت است:

(یک بلوک پایه‌ی ReNet)

فرق عمده‌ی آن با شبکه‌ی کانولوشنی اتصال مستقیمی است که از Input به Addition داریم و امتیازاتی را به وجود می‌آورد.

این جا عمل Batch Norm را قرار داده‌اند که سعی دارد مسئله‌ای را برای ما حل کند. نام این مسئله:

internal covariate shift problem

موجب مشکل در آموزش می‌شود.

کاری که Batch Norm می‌کند این است که خروجی لایه‌ی کانولوشن را نرمالیزه می‌کند.

Scanned with CamScanner

این مسئله این است: همین طور که آموزش جلو می‌رود تغییرات وزن‌ها موجب می‌شود توزیع مقادیر عوض شود، این تغییر توزیع برای آموزش نامطلوب است و باعث می‌شود آموزش کند شود و در یک مسیر مشخص نتواند حرکت کند. بنابراین ترجیح می‌دهیم خروجی لایه‌ی کانولوشن را نرمالیزه کنیم که جلوی این تغییرات را تا حدی بگیریم و آن را کنترل کنیم.

عملی که انجام می‌گیرد به این صورت است: فرض کنید یک Batch داریم: $B = \{x_{1 \dots m}\}$

میانگین آن را محاسبه می‌کنیم:

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i$$

واریانس را محاسبه می‌کنیم:

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_B)^2$$

نحوه‌ی نرمالیزه کردن:

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

scale & shift

خروجی نرمالیزه شده‌ی مطلوب:

$$y_i = \gamma \hat{x}_i + \beta \equiv BN_{\gamma,\beta}(x_i)$$

در حقیقت xها را به yهایی تبدیل می‌کنیم که میانگین و واریانس آن‌ها را کنترل می‌کنیم

مقایسه شبکه FF با ۲ لایه با شبکه رزنتی که ۲ تا بلوک دارد: (شکل ساختار در اسلایدها)

FF Network:

$h_1(X) = Sigm(X.W_1 + b_1)$

$Y_1 = h_2(h_1(X))$

$h_2(X) = Sigm(X.W_2 + b_2)$

$Y_1 = Sigm(h_1(X).W_2 + b_2)$

ResNet Network:

$g_1(X) = Sigm(X.W_1 + b_1 + X)$

$g_2(X) = Sigm(X.W_2 + b_2 + X)$

$Y_2 = g_2(g_1(X))$

$Y_2 = Sigm(g_1(X).W_2 + b_2 + g_1(X))$

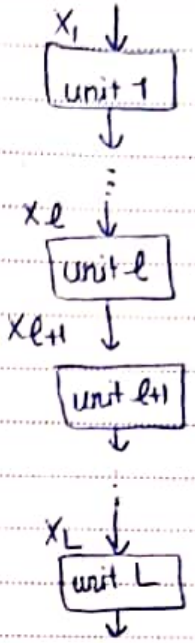
Scanned with CamScanner

تحلیل ResNet

می‌خواهیم ببینیم اگر بلوک‌ها را پشت سر هم قرار بدهیم به چه نتیجه‌ای می‌رسیم. حالت کلی‌تر این است که به جای اینکه خود x مستقیماً انتقال پیدا کند، یک تابعی روی آن اعمال شود:

$$y_\ell = h(x_\ell) + F(x_\ell, w_\ell)$$

$$x_{\ell+1} = f(y_\ell)$$



w_ℓ ← مجموعه‌ی وزن‌های مربوط به ℓ امین بلوک شبکه

$$w_\ell = \{w_{k,\ell} \mid 1 \leq k \leq K\}$$

K (بزرگ) ← تعداد لایه‌هایی که در واحد پایه داریم (۲ یا ۳ است)

F ← تابعی که بلوک پیاده می‌کند، مثلاً ممکن است شامل دو تا کانولوشن ۳×۳ که پشت سر هم هستند باشد.

f ← تابع فعال‌سازی مثلاً ReLU (the operation after element-wise addition)

حالا یک مقدار روابط را ساده‌تر می‌کنیم: فرض کنید $f(x)=x$ و $h(x_\ell)=x_\ell$

آن‌گاه داریم: $x_{\ell+1} \equiv y_\ell$

$$y_\ell = h(x_\ell) + F(x_\ell, w_\ell) \Rightarrow x_{\ell+1} = x_\ell + F(x_\ell, w_\ell)$$

$$x_{\ell+2} = x_{\ell+1} + F(x_{\ell+1}, w_{\ell+1}) = x_\ell + F(x_\ell, w_\ell) + F(x_{\ell+1}, w_{\ell+1})$$

بنابراین داریم:

$$\boxed{x_L = x_\ell + \sum_{i=1}^{L-1} F(x_i, w_i)}$$

یعنی در این شبکه برای واحدی در عمق L خروجی می‌تواند برحسب هر کدام از خروجی‌های بلوک‌های قبلی نوشته شود و خروجی لایه‌ی ابتدایی تر را با مجموع F هایی که در این مسیر دیده شد جمع می‌کنیم.

Scanned with CamScanner

حالا می‌خواهیم خواصی که این شبکه مبتنی بر این رابطه دارد را بررسی کنیم:

راحت‌ای که وجود دارد این است که می‌توانیم $l=0$ قرار بدهیم. در نتیجه x_0 را داریم که ورودی به شبکه است یعنی اولین ورودی است. پس می‌توانیم در هر بلوکی رابطه را برحسب ورودی شبکه بنویسیم:

$$x_L = x_0 + \sum_{i=0}^{L-1} F(x_i, W_i)$$

اگر در شبکه FF که این اتصالات مستقیم وجود ندارد می‌خواستیم این کار را انجام بدهیم می‌شد حاصلضرب تعدادی ماتریس. یعنی بلوک بلوک که جلو می‌آمدیم ضرب می‌شد در یک ماتریس W دیگر و یک مجموعه‌ی ماتریس ضرب شونده در هم را داشتیم که کار را مشکل می‌کند اما این جا جمله‌ی ثابتی که داریم به حل مشکلات کمک می‌کند. حالا چگونه کمک می‌کند؟ خطا را بررسی می‌کنیم:

$$\frac{\partial \mathcal{E}}{\partial x_l} = \frac{\partial \mathcal{E}}{\partial x_L}\frac{\partial x_L}{\partial x_l} = \frac{\partial \mathcal{E}}{\partial x_L}\left(1 + \frac{\partial}{\partial x_l}\sum_{i=l}^{L-1} F(x_i, W_i)\right)$$

یعنی در مشتقی که برای خطا حساب می‌کنیم علاوه بر مشتقی که به صورت معمول در شبکه‌های FF داشتیم یک جمله‌ی دیگر هم پیدا می‌شود که به ما اجازه می‌دهد بتوانیم مشتق را منتقل کنیم در کنار بخشی که مربوط به F ها منتقل می‌شود. این قسمت مستقیماً مشتق را به لایه قبلی منتقل می‌کند و باعث می‌شود مشتق بتواند در لایه‌های اولیه هم بزرگ باشد و مشکل vanishing را حل می‌کند. (این جمله‌ی ⊥ در مشتق)

مقایسه‌ی شبکه‌های کانولوشنی مختلف:

صحت: ResNet > VGG-16 > AlexNet > LeNet-5

تعداد پارامترها: VGG-16 > ResNet > AlexNet > LeNet-5

برای بلوک پایه پیشنهادهای دیگری هم داده شد. مثلاً Constant Scaling ← در هر دو شاخه مقیاس شدن اعمال می‌شود.

Scanned with CamScanner

۲) بلوک پایه دیگر: exclusive gating

۳) shortcut only gating ← در این مورد و قبلی $h(x)$ دیگر برابر x نیست بلکه تابع دیگری است

نکته: در این شبکه چون X را با $F(X)$ جمع می‌کنیم باید دقت کنیم که ابعادشان باید یکی باشد.

ممکن است بُعد $F(X)$ متفاوت از X باشد در این حالت می‌توانیم یک ضرب ماتریسی در X انجام بدهیم به گونه‌ای که هم بُعد با $F(X)$ شود:

$$F(X) + W_s X$$

درایه‌های ماتریس W_s باید مشخص شود که می‌توانیم در طول آموزش اجازه بدهیم که تعیین شود، یعنی درایه‌های این ماتریس را به عنوان پارامترهایی که در حین آموزش باید مشخص شود در نظر بگیریم.

یک روش دیگر هم این است که با اضافه کردن صفر بُعد X را به بُعد $F(X)$ برسانیم.

Densely Connected Convolutional NNs (DenseNets)

توسعه‌ی دیگری که در شبکه‌های کانولوشنی داده شد و باعث بهبود در آن‌ها شد DenseNet است

در این جا برای حل مسئله‌ی vanishing gradient اینگونه عمل می‌کنیم:

فرض کنیم L تا لایه داشته باشیم و هر کدام از این لایه‌ها تابع H_ℓ را پیاده کنند. در خروجی هر کدام از این لایه‌ها یک سری نقشه ویژگی (صفحات خروجی) ایجاد می‌شود.

وقتی به لایه‌ی بعدی می‌رویم این ویژگی‌ها را هم به عنوان ورودی به لایه‌های بعدی می‌دهیم و این کار را در همه‌ی لایه‌ها انجام می‌دهیم.

بنابراین همینطور که جلو می‌رویم تعداد ورودی‌ها در این شبکه زیاد می‌شود.

لایه‌ی ℓ تابع H_ℓ را پیاده می‌کند که شامل Batch Norm، ReLU و کانولوشن ۳x۳ است.

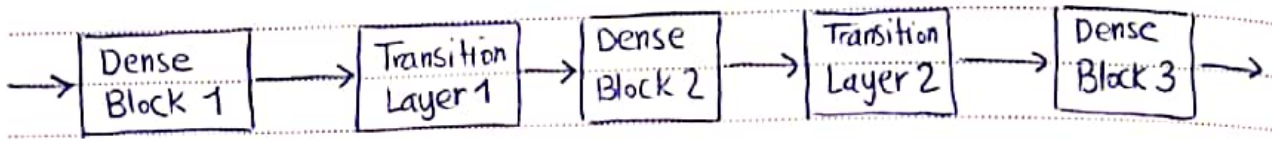
مثلا در اولین لایه این ۳ عمل را داریم، تعداد صفحات ویژگی در خروجی این لایه تولید می‌شود. حالا برای لایه

Scanned with CamScanner

بعدی، این خروجی‌ها می‌شود ورودی و باز به طور مشابه در خروجی لایه دوم این صفحات دیگری تولید می‌شود.
وقتی به لایه سوم می‌رویم نه فقط صفحات خروجی لایه دوم بلکه صفحات دیگری که در لایه اول هم تولید شده را به عنوان ورودی به لایه سوم می‌دهیم.

$$\boxed{X_\ell = H_\ell([X_0, X_1, \dots, X_{\ell-1}])}$$

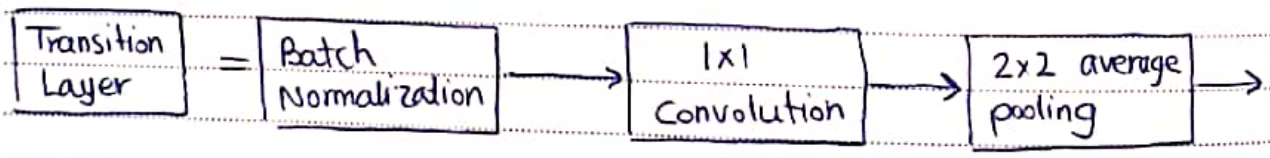
معماری شبکه



بین هر دو تا Dense Block یک Transition Layer قرار می‌گیرد. کاری که Transition layer انجام می‌دهد:

1) Batch Normalization
2) 1x1 Convolution
3) 2x2 Average pooling

یعنی به نوعی اتفاقاتی که در Dense Block‌ها می‌افتد را کنترل می‌کند.



(شکل معماری DenseNet در اسلاید)

Growth Rate

یک نکته دیگر این است که هر کدام از لایه‌ها چند تا نقشه تولید کنند؟
اگر فرض کنیم هر کدام از لایه‌ها K تا نقشه تولید کند، تعداد ورودی‌های لایه‌ی ℓام برابر است با:

$$\boxed{K_0 + K(\ell - 1)}$$

K = نرخ رشد شبکه

تعداد Feature map‌هایی که هر لایه تولید می‌کند

Scanned with CamScanner

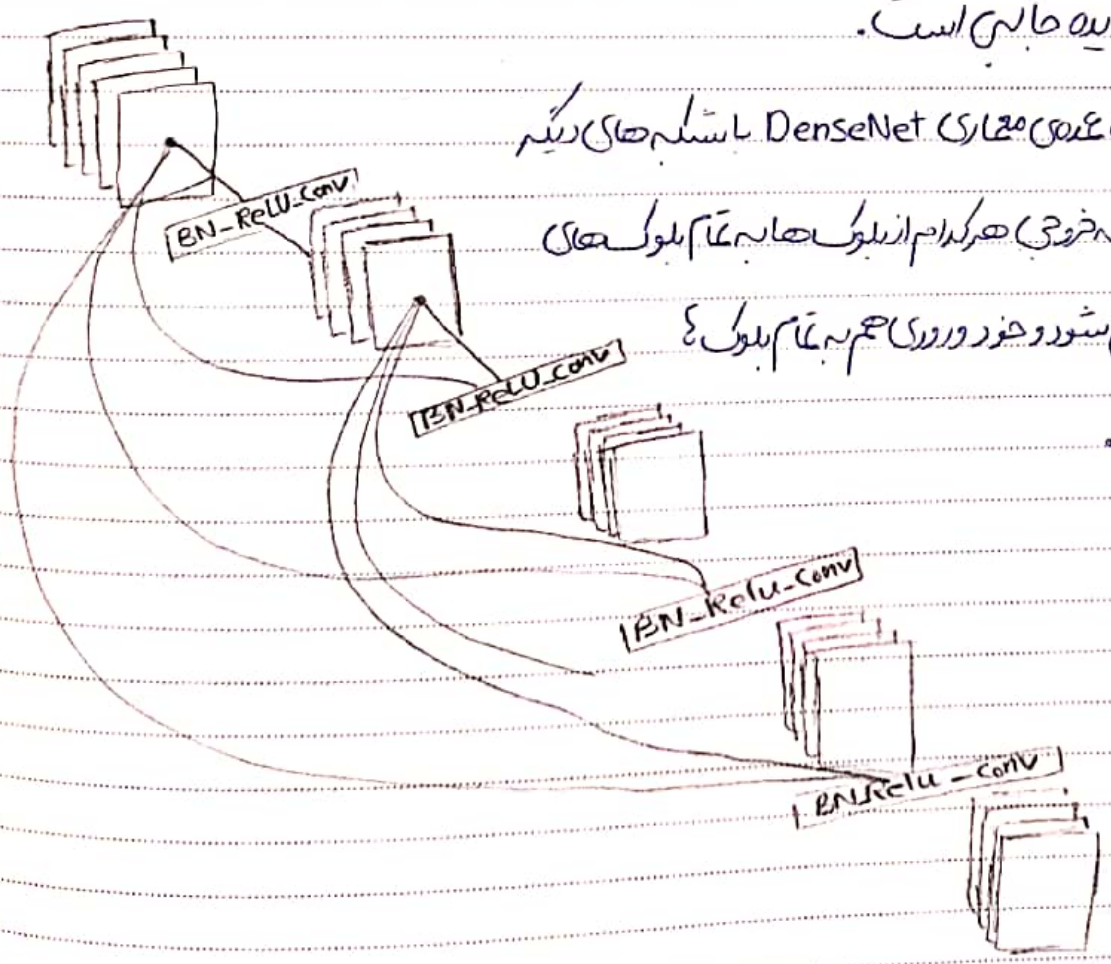
گفته شده که معمولا برای به دست آوردن بهترین نتایج لازم نیست که نرخ رشد خیلی بالا باشد

مثلا در حد k=12 کافی است.

نکته دیگر این است که این Feature Map ها در جاهای مختلف شبکه با همدیگر، حالت شبکه (global state) را مشخص می کنند. شرایطی که شبکه دارد را به ما نشان می دهد. جمع اطلاعاتی که می توانیم از همه ی شبکه بگیریم در این ها هست. می توان گفت که این Feature Map ها به ما ویژگی هایی با درجات تجرید مختلف می دهند. هرچقدر جلوتر می رود، درجه تجریدش بالاتر می رود.

به این ترتیب این ها جمعا اطلاعات خیلی خوبی می دهند.

اصولا این ایده که به هرکدام از لایه های شبکه ویژگی هایی با درجات تجرید مختلف به عنوان ورودی داده شود، ایده جالبی است.



نکته: تفاوت عمده معماری DenseNet با شبکه های دیگر در این است که خروجی هرکدام از بلوک ها به تمام بلوک های بعدی وارد می شود و خود ورودی هم به تمام بلوک ها وارد می شود.

جلسه بیستم - ۱۴۰۰/۶/۹

Growth rate

هر لایه K تا نقشه تولید می‌کند و به ورودی های لایه های بعد اضافه می‌شود

تعداد نقشه‌هایی که به لایه l وارد می‌شود: $K_0 + K(l-1)$ ← تعداد نقشه‌های ورودی

K ← نرخ رشد در این شبکه

اگر K حدود ۱۲ باشد کافی است که عملکرد این شبکه برسد به بهترین عملکردهای شبکه‌های دیگر.

Bottleneck Layers

یک نسخه دیگر از این شبکه: می‌خواهیم به جوری حجم داده‌ای که در شبکه تولید می‌شود را کنترل کنیم

یک لایه گلوگاه می‌گذاریم که اطلاعات را فشرده می‌کند

با قرار دادن یک کانولوشن 1x1 قبل از کانولوشن ۳x۳ این کار را انجام می‌دهند

ابتدا اطلاعات فشرده می‌شود و بعد وارد عملیات سه‌گانه‌ی Batch Norm ، ReLU و Conv(3x3) می‌شود

بنابراین یک Conv(1x1) ReLU Batch Normalization داریم و بعد از آن Batch Norm, Relu, Conv(۳x۳)

اسم این نسخه از شبکه: DenseNet_B

در این جا conv(1x1) طوری اعمال می‌شود که 4K تا feature Map تولید شود. K مقدار صفات دیگر بود

که هر لایه تولید می‌کرد و این جا می‌خواد که ۴ برابر اون باشه

Compression

سومین نسخه این شبکه ← باز هم تعدد دارد فشرده‌سازی روی اطلاعات انجام بدهد. اگر فرض کنیم یک بلوک

denseNet ، m تا نقشه داشته باشد، سعی می‌کنیم که در Transition layer بعدی تعداد این‌ها به

$\lfloor \theta m \rfloor$ برسد.

Scanned with CamScanner

Compression Factor ← $0 < \theta < 1$

بنابراین تعداد feature Map ها کاهش داده می‌شود. نام این نسخه ← DenseNet_C

اگر هم Bottleneck layer را بگذارد و هم این Compression را اعمال کند به آن DenseNet_BC می‌گویند.

DenseNet + Bottleneck layer = DenseNet_B
DenseNet + Compression = DenseNet_C
DenseNet + Bottleneck + Compression = DenseNet_BC

جدول مقایسه DenseNet با سایر روش‌ها در اسلاید

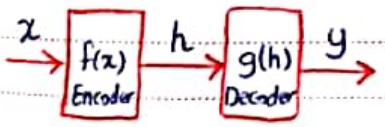
در این جدول نشان داده شده که DenseNet حتی از ResNet هم بهتر عمل می‌کند.

خودکدگذارها (AutoEncoders)

هدف این شبکه: بتواند ورودی را دوباره در خروجی ایجاد کند، یعنی خروجی اش برابر ورودی اش باشد.

این شبکه ۲ بخش دارد: ① کدگذار (کدکننده) Encoder ② کدگشا Decoder

وقتی ورودی x به شبکه داده می‌شود Encoder آن را می‌گیرد و خروجی h ($f(x)$) را می‌دهد. بعد Decoder h را ورودی می‌گیرد و y را خروجی می‌دهد که برابر x ورودی است.



فایده این شبکه چیست؟

فایده این شبکه در y یا همان خروجی ای که تولید می‌کند نیست ولی این شبکه با گرفتن ورودی x ارائه‌ای از آن را پیدا می‌کند که با h نشان می‌دهد و این ارائه است که برای ما مهم است. خود این شبکه این ارائه را جوری پیدا می‌کند که بتوان از روی آن، x اولیه را ساخت.

یعنی h یک representation از x است که برای ما ارزشمند است.

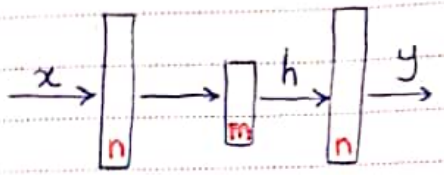
در Auto Encoder های جدیدتر یک نگاه احتمالاتی هم به کار رفته و با زبان احتمالات ارائه داده اند. در این حالت Encoder میاد به یک گاست تصادفی انجام میده: $P_{encoder}(h|x)$

decoder هم اینطور بیان می‌شود: $P_{decoder}(x|h)$

Scanned with CamScanner

معماری

دو حالت کلی می‌تواند داشته باشد:



۱) undercomplete Autoencoder $(m<n)$

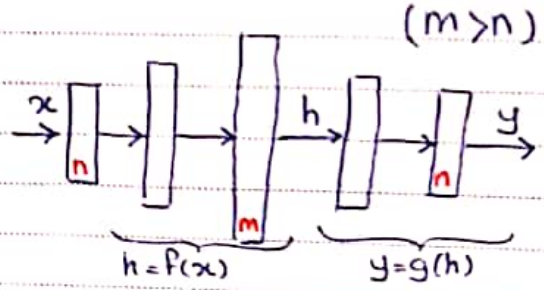
x ورودی با بُعد n به لایه ورودی وارد می‌شود، بعد توزیع می‌شود به لایه‌ی بعدی که m تا نورون دارد و h را تولید می‌کند این لایه h به لایه‌ی بعدی که n تا نورون دارد داده می‌شود و y را تولید می‌کند. اگر $m<n$ ← underComplete ← یعنی تعداد نورون‌ها در Encoder از بُعد x کوچکتر است در این حالت.

xهایی که وارد می‌شوند وقتی در h ارائه می‌شوند، فشرده شده‌اند $(m<n)$. می‌توان گفت که در این حالت شبکه یاد می‌گیرد که ویژگی‌های مشخص ورودی‌ها را استخراج کند و در ارائه‌ی h بیاورد. چون این ویژگی‌ها لازم‌اند برای اینکه x مجدداً بازسازی شود پس می‌توان گفت ویژگی‌های اصلی هستند.

در این جا به صورت ۳ لایه رسم شده اما در حالت کلی Encoder و decoder می‌توانند بیش از ۱ لایه باشند.

۱۲ | 3 Sep

۲) overcomplete Autoencoder $(m>n)$



بررسی‌هایشان داده این شبکه به دلیل اینکه $m>n$ می‌تواند x را در خروجی خودش کپی بکند بدون اینکه h به درد بخوری در خروجی Encoder یاد گرفته باشد. گفته می‌شود این شبکه ظرفیت بالایی دارد به خاطر ابعادش.

حتی می‌توانید با یک رابطه خطی این کار را انجام بدهید و دقیقاً x را تولید کنید بدون اینکه h چیزی باشد که ما می‌خواهیم. این ایرادی است که در معماری overcomplete وجود دارد.

این شبکه‌ها FF هستند و با BP آموزش داده می‌شود. تابع Loss آن به این صورت است:

Scanned with CamScanner

$$\boxed{J_{AE}(\theta) = \sum_{x \in D_n} L(x, g(f(x)))}$$

g عملی است که decoder انجام میدهد

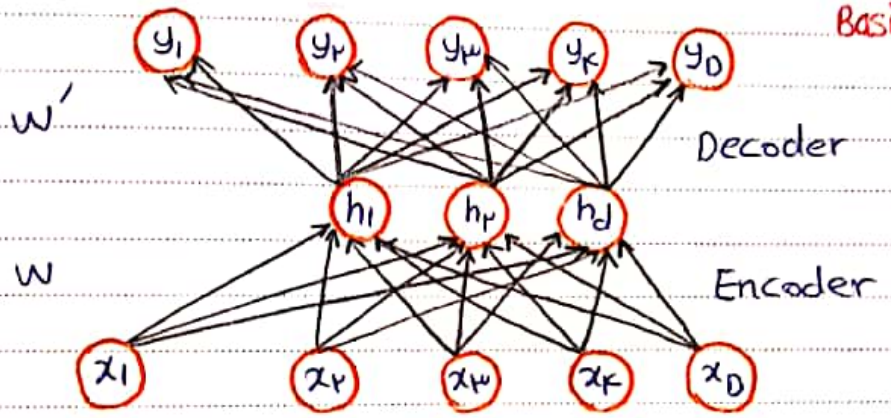
f عملی است که encoder انجام می دهد

تابع Loss براساس تفاوت ورودی و خروجی تعریف می شود

اگر decoder در این شبکه خطی باشد و L را هم برابر با میانگین مربع خطا بگیریم، آن گاه یک شبکه ی Autoencoder undercomplete همان فضایی را یاد می گیرد که PCA یاد می گیرد.

اگر Decoder و Encoder غیرخطی باشند یک تعمیمی از PCA را یاد می گیرد که قوی تر است.

Basic AutoEncoders



x ها در لایه ورودی اعمال می شوند و توزیع می شوند به لایه ی مخفی که همان لایه ی Encoder است (با وزن های W توزیع می شوند). بعد خروجی لایه مخفی با وزن های لایه Decoder وارد می شوند و خروجی نهایی تولید می شود.

Encoder

$$h = f(x) = \overset{\text{Sigmoid}}{S_f}(Wx + \overset{\text{bias}}{b_h})$$

$$S_f = S_g = S(z) = \frac{1}{1+e^{-z}}$$

Decoder

$$y = g(h) = \overset{\text{Sigmoid}}{S_g}(W'h + b_y)$$

Training: Gradient Descent

objective function:

$$J_{AE}(\theta) = \sum_{x \in D_n} L(x, g(f(x)))$$

D_n : Training Set

پارامترهای شبکه $= \theta = \{W, b_h, b_y\}$

Scanned with CamScanner

تابع Loss می‌تواند موارد مختلفی باشد مثلا:

① squared error: $L(x,y) = \|x-y\|^2$

② cross_entropy Loss: $L(x,y) = -\sum_{i=1}^{d_x} x_i \log(y_i) + (1-x_i)\log(1-y_i)$

نکته: وزن‌های w و w' می‌توانند به گونه‌ای انتخاب شوند که: $w' = w^T$. در این حالت تعداد پارامترها کمتر می‌شود و به این حالت Tied weights (وزن‌های به هم گره خورده) گفته می‌شود.

Regularized AutoEncoders

برای اینکه خودکدگذار overComplete را مجبور به تولید یک h مفید کنیم می‌توانیم از Regularization استفاده کنیم. باید در تابع زیان شبکه جمله‌ای اضافه کنیم که چیزی که مورد نظر ماست را تحمیل کند به شبکه.

مثلا می‌توانیم ارائه را به سمت تنک شدن ببریم (sparsity of representation)

یا مثلا می‌توانیم کوچک شدن مشتق ارائه را دنبال کنیم. یا می‌توانیم مقاومت در برابر نویز یا در برابر ورودی‌هایی که نادیده شده اند را به وجود بیاوریم در این شبکه. با این کارها می‌توانیم شبکه overcomplete را تبدیل کنیم به شبکه‌ای که کار مفیدی انجام می‌دهد.

یک روش ساده برای Regularization این است که weight_decay انجام بدهیم.

$$\boxed{J_{WDRAE}(\theta) = \sum_{x \in D_n} L(x, g(f(x))) + \lambda \sum_{i,j} w_{i,j}^2}$$

در این جا یک ویژگی جدید حین فرآیند آموزش روی وزن‌ها اعمال می‌کنیم که وزن‌ها نباید بزرگ شوند.

WDRAE → weight Decay Regularized Auto Encoder

(نسبت جمله‌ی regularization به جمله‌ی اصلی را کنترل می‌کند در واقع میزان منظم‌سازی را کنترل می‌کند.

آزمایش‌ها نشان داده است که با انجام این کار مدلی به دست می‌آید که هم ظرفیت بالایی دارد و هم اینکه یادگیری مفید h در آن انجام می‌شود.

Scanned with CamScanner

Sparse Autoencoders

تعبیر دیگری که می‌توان در خودکدگذار overComplete داد این است که تابع زیان را به گونه‌ای تغییر دهیم که ارایه‌ای که حاصل می‌شود تعداد قابل توجهی از درایه‌هایش صفر باشند. (یعنی sparse یا تنک باشد)

$$J_{SAE}(\theta) = \sum_{x \in D_n} L(x, g(f(x))) + \eta \sum_j KL(\rho | \rho_j)$$

$$KL(\rho|\rho_j) = \rho \log \frac{\rho}{\rho_j} + (1-\rho) \log \frac{1-\rho}{1-\rho_j}$$

KL ← Kullback Leibler distance
← فاصله‌ی دو تا توزیع را با آن می‌سنجند

ρ_j ← میانگین خروجی نورون j روی m تا داده‌ی ورودی

ρ ← یک عدد ثابت در نظر می‌گیریم که خیلی کوچک است مثلا ۰/۰۵

در واقع می‌خواهیم میانگین خروجی نورون j روی ورودی‌های مختلف به سمت ۰/۰۵ (ρ) برود. همین به سمت اپارس شدن می‌برد و به سری از درایه‌های representation ای که یافت می‌شود صفر خواهند بود.

η هم وزن جمله‌ی اضافه شده را کنترل می‌کند

از ابتدای شبکه اگر چند لایه داشته باشیم تا h تولید شود خروجی هر کدام از این لایه‌ها sparse نیست هنوز ویژگی‌های داده را به خوبی encode می‌کند. اما وقتی h که sparse است تولید می‌شود نشان می‌دهد که بعضی از درایه‌ها که صفر شده‌اند مهم نیستند و در ارائه‌ی داده‌ی ورودی نقشی ندارند.

به نوعی feature selection انجام می‌دهد به گونه‌ای که فیچرهایی که صفر نیستند مهم‌ترند و آن‌هایی که صفر هستند فیچرهای کم اهمیت اند.

Scanned with CamScanner

Denoising Autoencoders

در این جا داده‌ها را نویزی می‌کنیم و می‌خواهیم شبکه همان ورودی را بدون نویز خروجی بدهد. برای این کار ورودی‌ها را با نمونه‌هایی که از یک توزیع گاوسی isotropic گرفته می‌شود، نویزی کنیم:

در رابطه‌ی زیر، ϵ از یک توزیع نرمال با میانگین ۰ و واریانس σ^2 گرفته می‌شود

$$\tilde{x} = x + \epsilon, \quad \epsilon \sim N(0, \sigma^2 I)$$

یک راه دیگر، یک ماسک باینری در نظر بگیریم با مقادیر ۰ و ۱ که در ورودی ضرب می‌شود و بعضی از ورودی‌ها را صفر می‌کند یعنی بعضی از ورودی‌ها را نداریم و ورودی ناقص است. اما در خروجی می‌خواهیم ورودی کامل را داشته باشیم. ۰ و ۱هایی که در ماسک است به صورت رندوم انتخاب می‌شود.

$\tilde{x}$ با یک فرآیند تخریب $q(\tilde{x}|x)$ ارائه می‌شود. تابعی که برای آموزش شبکه به کار می‌رود:

$$\boxed{J_{DAE}(\theta) = \sum_{x \in D_n} \mathbb{E}_{\tilde{x} \sim q(\tilde{x}|x)} \left[L(x, g(f(\tilde{x}))) \right]}$$

(شکل در اسلاید)

optimization با استفاده از SGD صورت می‌گیرد. این کار هم باعث می‌شود که h مفید شود علیرغم overcomplet بودن.

stacked Denoising Autoencoders

می‌توانیم پشته‌ای از Denoising AE ها داشته باشیم و اون hای که AE اول یاد می‌گیرد ورودی AE دوم باشد، hای که دومی یاد می‌گیرد ورودی سومی باشد و... به این صورت می‌توانیم representation ها را همین طور که جلو می‌روند بهبود ببخشیم و می‌توانیم تعدادی از این AE ها را داشته باشیم که ما را به یک ارائه خیلی خوب از داده‌ها برسانند.

بعد از آن در یک ساختار supervised دوباره همه‌ی شبکه را آموزش می‌دهیم برای اینکه شبکه عمل دسته‌بندی را یاد بگیرد در عین حال یک آموزش جزئی وزن‌های Encoder ها ببینند.

Scanned with CamScanner

وقتی Autoencoder ها را تک به تک آموزش می‌دهیم هر کدام به طور جداگانه آموزش دیده اند و بعد این ها را به هم می‌چسبانیم در حالی که همه با هم آموزش ندیده اند. به همین دلیل آموزش مرحله‌ی دوم که گفتیم نیاز است. که به آن fine_tuning گفته می‌شود. ممکن است در آن وزن ها تغییرات کوچکی بکنند. (شکل را اسلاید)

Contractive Autoencoders (خودکدگذار انقباضی)

گفتیم در Denoising Autoencoder مقصدمان این است که $g(f(x))$ نسبت به ورودی های نویزی مقاوم باشد. یعنی اگر در ورودی x نویزی بدهیم در خروجی همان x را بدون نویز دریافت کنیم.

به راه دیگر که گفته شده بیاییم خود h را نسبت به نویز مقاوم کنیم. تابع هدف را به این صورت تعریف می‌کنند:

$$\boxed{J_{CAE}(\theta) = \sum_{x \in D_n} \left(L(x, g(f(\hat{x}))) + \lambda \| J_f(x) \|_F^2 \right)}$$

J_f ← Jacobian f ← مشتقات درجه ۱ تابع f

جمله‌ی Regularization ← نرم فروبنیوس جاکوبین f (Frobenius norm of Jacobian of f)

داریم اندازه‌ی مشتق را سعی می‌کنیم که minimize کنیم. مشتق یعنی تغییرات و چون می‌خواهیم تغییرات $f(x)$ یا h نسبت به تغییرات x کم باشد مشتق f نسبت به x را minimize می‌کنیم. یعنی تغییرات h را minimize می‌کنیم

$$\Omega(h, x) = \lambda \sum_i \| \nabla_x h_i \|^2$$

نسبت به نویزی که در ورودی اضافه شده.

علت اینکه به آن Contractive می‌گویند این است که کاری که در فرآیند Minimization انجام می‌دهیم در حقیقت معادل این است که یک همسایگی از ورودی ها را به یک همسایگی کوچکتر نگاشت می‌کند.

Scanned with CamScanner

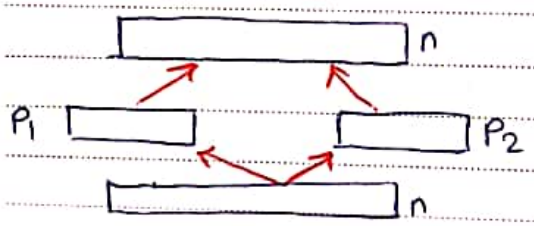
انگاه یک انقباض در اندازه‌ی همسایگی داریم.

استفاده از Encoder و decoder های deep خوب است چون کمک می‌کند تقریب بهتری بزنیم و نگاشت مناسب‌تری داشته باشیم. حتی منجر می‌شود که هزینه‌ها در کل کاهش پیدا کند چون با deep کردنشان آموزش راحت‌تر انجام می‌شود و می‌تواند با داده‌های کم‌تر هم صورت بگیرد.

Combined Representations

می‌توانیم تعدادی ارائه از auto encoder به دست آوریم و این‌ها را با هم ترکیب کنیم و پشت سر هم قرار بدهیم. (Concat) در این صورت می‌توان یک ارائه‌ی قوی‌تری پیدا کرد.

فرض کنید دو تا auto encoder با معماری‌های $n/p_1/n$ و $n/p_2/n$ داشته باشیم. در اثر آموزش آن‌ها دو تا ارائه به دست می‌آید که می‌توان با ترکیب آن‌ها یک ارائه‌ی قوی با سایز p_1+p_2 داشته باشیم.



فرق این دو تا representation که به دست می‌آوریم می‌تواند در این موارد باشد:

1. الگوریتم یادگیری
2. مقداردهی اولیه (initialization)
3. فرایند آموزش
4. نرخ یادگیری
5. معیارهای اندازه‌گیری اعوجاج (distortion measures)

هم‌چنین پیش‌نهاد شده نورون‌های encoder تدریجا کم و زیاد شود، به این لایه نورون اضافه یا از آن کم کنیم.

نکته: در حالت خطی که نورون‌ها خطی‌اند و در فضای حقیقی باشیم، اگر یک واحد مخفی قرار بدهیم اولین principal component را می‌یابد. دومی را که قرار بدهیم دومین PC را می‌یابد.

Scanned with CamScanner

جلسه بیست و یکم - ۱۴۰۰/۹/۱۴

تاکنون ۴ مسئله را مورد بررسی قرار دادیم: دسته‌بندی، تخمین تابع، خوشه‌بندی، استخراج ویژگی

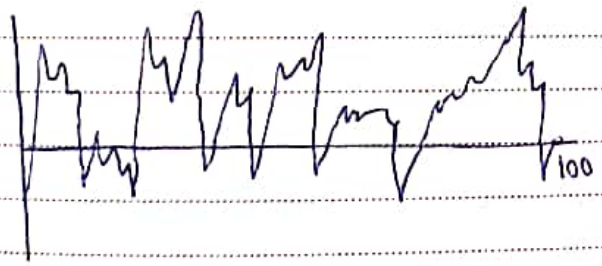
مسئله‌ی پنجمی که بررسی می‌کنیم: پیش‌بینی سری‌های زمانی با شبکه‌های عصبی

این مسئله بسیار مهم است و در مسائل عملی خیلی پیش می‌آید. مثلا: پیش‌بینی بارش باران، پیش‌بینی مصرف برق، پیش‌بینی بازار سهام و ...

در همه‌ی این‌ها یک متغیری داریم که به صورت منظم مقادیر آن را از بازه‌های زمانی گذشته داریم و می‌خواهیم مقادیر آینده‌ی آن را پیش‌بینی کنیم.

تاکنون در شبکه‌هایی که داشتیم نمی‌توانستیم ورودی‌ای داشته باشیم به صورت دنباله‌ای از اعداد که با زمان تغییر پیدا کرده‌اند. بنابراین می‌خواهیم یک ویژگی جدیدی را وارد شبکه‌های عصبی کنیم.

مثال:



مثلا تغییرات یک متغیر را در ۱۰۰ روز داریم و می‌خواهیم مقدار آن را در قدم بعدی پیش‌بینی کنیم.

با توجه به این سری‌ها می‌توان متوجه شد که این مسئله، مسئله‌ی دشواری است.

برای حل مسئله از یک ویژگی که در سری‌های زمانی وجود دارد استفاده می‌کنیم: خودهمبستگی (Auto correlation)

یعنی همبستگی بین نمونه‌ای که الان داریم و نمونه‌های قبلی ← مبنای روش‌های پیش‌بینی

این همبستگی در مورد نمونه‌هایی که در گذشته‌ی نزدیک هستند بیشتر قابل اعتنا است تا آن‌هایی که مربوط به گذشته‌های دورتر هستند. به همین دلیل نمونه‌های قبلی مهم و تاثیرگذار در نظر می‌گیرند و به آن‌ها lags گفته می‌شود.

مدل زمانی: مدلی است که سعی می‌کند مقادیر آینده‌ی یک متغیر را بر اساس نمونه‌های قبلی که از این متغیر داریم پیش‌بینی کند

Temporal Models

Scanned with CamScanner

Spatio_Temporal Mobility : مقدار آینده‌ی متغیر را با استفاده از مقادیر گذشته‌ی خودش و یک متغیر دیگر پیش بینی می‌کند

یعنی در ورودی فقط خود این متغیر نیست، بلکه متغیرهای دیگر هم هستند

پیش بینی می‌تواند بر مبنای مدل‌های خطی یا غیرخطی باشد.

Linear Forecasting

1) Autoregressive (AR) model of order p

مدل خودپس رونده از درجه‌ی p

$$y_t = a_0 + a_1 y_{t-1} + a_2 y_{t-2} + \dots + a_p y_{t-p} + \varepsilon_t$$

در این جا یک ترکیب خطی از نمونه‌های قبلی در نظر می‌گیرد و p قدم به عقب برمی‌گردد. یعنی تعداد نمونه‌های موثر را p در نظر می‌گیرد به همین دلیل می‌گوییم این مدل از درجه‌ی p است.

در حقیقت در هر زمانی که هست، مقدار را برای قدم بعدی از ترکیب خطی و نمایی مقادیر قبلی پیش بینی می‌کند.

ε_t خطا در زمان t است، یک متغیر تصادفی که از یک تابع گاوسی با میانگین صفر و واریانس σ^2 نمونه برداری می‌شود

Moving Average (MA) model of order q

مدل میانگین متحرک از درجه‌ی q

$$y_t = a_0 + \varepsilon_t - b_1 \varepsilon_{t-1} - b_2 \varepsilon_{t-2} - \dots - b_q \varepsilon_{t-q}$$

بازسازی مقدار بعدی به جای اینکه بر اساس نمونه‌های قبلی باشد، بر اساس خطاهای پیش بینی‌های قبلی انجام می‌شود

Autoregressive Integrated Moving Average (ARIMA) model of order (p,q)

$$y_t = a_0 + a_1 y_{t-1} + a_2 y_{t-2} + \dots + a_p y_{t-p} + \varepsilon_t - b_1 \varepsilon_{t-1} - b_2 \varepsilon_{t-2} - \dots - b_q \varepsilon_{t-q}$$

ε_i ها متغیرهای تصادفی مستقل از هم و دارای توزیع یکسان هستند

مدل‌های غیرخطی

$$y_t = f(y_{t-1}, y_{t-2}, \dots, y_{t-p}) + \varepsilon_t$$

این چیزی که مدل را غیرخطی می‌کند تابع f است

چگونه می‌توانیم داده‌هایی که دارای زمان هستند را وارد شبکه‌های عصبی کنیم؟

برای انجام این کار باید بتوانیم حافظه کوتاه مدت به شبکه ایجاد کنیم. به دو راه می‌توان این کار را انجام داد:

① یک مکانیزمی به نام tapped delay Line (خط تاخیر اتصال دار) به شبکه اضافه کنیم

Scanned with CamScanner

② اضافه کردن اتصالات فیدبک دار به شبکه (feedback)

در مقابل حافظه‌ی کوتاه مدت، حافظه‌ی بلندمدت را داریم که در شبکه توسط وزن‌ها مشخص می‌شود.

حافظه‌ی کوتاه مدت اثرش با زمان از بین می‌رود ولی حافظه‌ی بلندمدت ماندگاری دارد.

STM structures

اگر یک سری زمانی داشته باشیم که با یک دنباله $\{x(n)\}$ نشان داده می‌شود و یک سری زمان گسسته است. تبدیل Z برای سری‌های گسسته به کار می‌رود:

$$X(Z) = \sum_{n=-\infty}^{\infty} x(n) z^{-n}$$

z^{-1} = unit delay operator

↓

اپراتور تاخیر واحد، یعنی یک قدم تاخیر

$$y(n) = \sum_{k=-\infty}^{\infty} h(k)\, x(n-k)$$

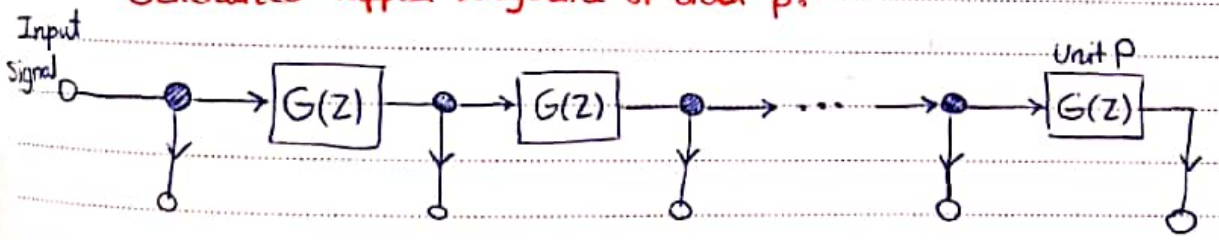
حوزه‌ی Z

برای یک سیستم گسسته زمان این روابط را داریم:

$$Y(Z) = H(Z)\, X(Z)$$

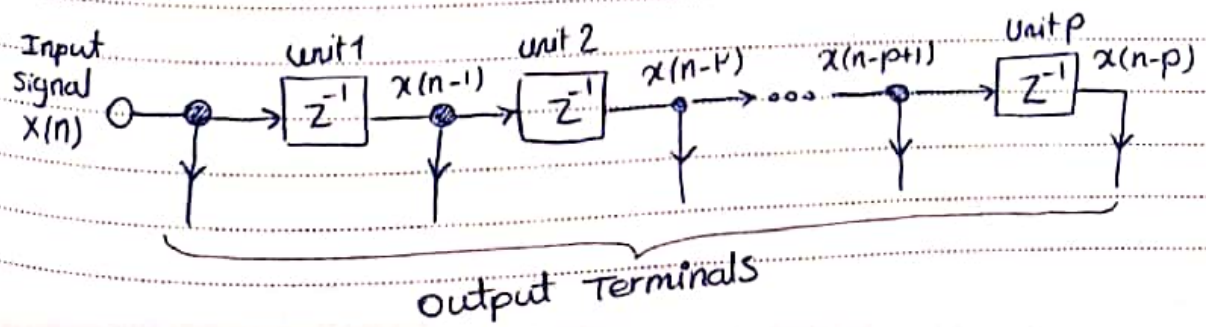
خروجی را می‌توان بر اساس ورودی و تابع h نوشت که h(k) پاسخ ضربه‌ی سیستم است

Generalized Tapped Delay-Line of order p:



output Terminals

حالت کلی خط تاخیر، یک سری بلوک داریم که عمل G(Z) را انجام می‌دهد، p تا از این‌ها را به صورت سری پشت سرهم داریم. چون G را کلی گذاشتیم به آن Generalized می‌گویند. حالا اگر بخواهیم خاصش کنیم: $G(z) = z^{-1}$



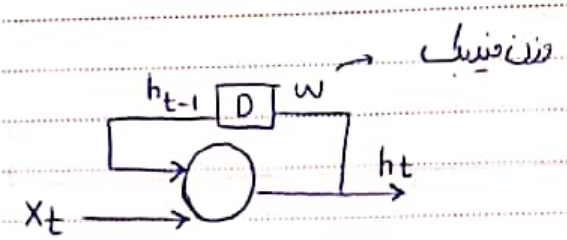
Scanned with CamScanner

وقتی $G(z)=z^{-1}$ یعنی هر unit فقط یک قدم تأخیر ایجاد کند و هیچ عمل دیگری روی سیگنال انجام ندهد جز اینکه یک قدم آن را نگه دارد. شبیه shift register است.

در اتصالات بعد از هر unit خروجی ها را نگه میداریم و همه ی این ها می شوند خروجی های این ساختار برای ما.

به آن Tapped Delay-Line of order p گفته می شود.

Feedback Connection



در شکل روبه رو خروجی نورون را با یک تأخیر واحد به ورودی نورون برگردانده ایم. فرض کنیم نورون خطی باشد:

$f(x_t)=x_t \qquad h_t=f(x_t,h_{t-1})=x_t+wh_{t-1}$

$x_1 \qquad h_1=x_1$

$x_2 \qquad h_2=x_2+wh_1=x_2+wx_1$

$x_3 \qquad h_3=x_3+wh_2=x_3+wx_2+w^2x_1$

$\dots \qquad \dots$

$x_m \qquad h_m=x_m+wx_{m-1}+w^2x_{m-2}+w^3x_{m-3}+\dots+w^{m-1}x_1$

خروجی در هر زمان برابر است با ترکیبی از ورودی ها در این قدم و قدم های قبل.

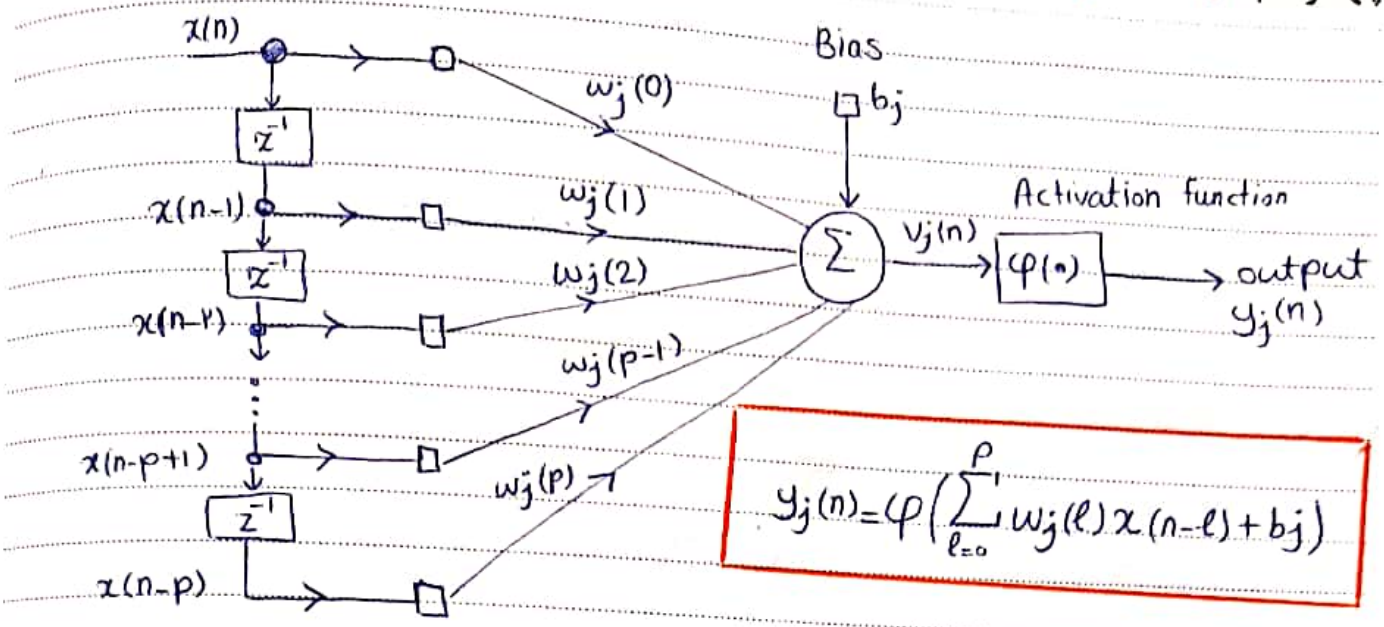
اگر $|w|<1$ ← ضریب ها هر چقدر دورتر می شویم کوچکتر می شود. یعنی نمونه های نزدیکتر اثرگذاری بیشتری دارند.

حالا چگونه از این ها در شبکه ها استفاده کنیم؟

ابتدا فرض کنید یک نورون داشته باشیم که بخواهیم سری های زمانی را به عنوان ورودی به آن بدهیم. از ساختار Tapped Delay-line استفاده می کنیم و آن را به ورودی های نورون وصل می کنیم:

Scanned with CamScanner

Focused Neuronal Filter:



$$y_j(n)=\varphi\left(\sum_{l=0}^{p} w_j(l)\,x(n-l)+b_j\right)$$

هرکدام از خروجی های Tapped Delay-Line مرتبه‌ی p را

با یک وزن ورودی به نورون می دهیم. در واقع برای اینکه بتوانیم یک سیگنال تابع زمان را به نورون بدهیم، نمونه‌ها

زمان‌های مختلف را هم زمان به نورون می دهیم.

Focused Time Lagged FeedForward Network (TLFN)

می توانیم در لایه‌ی مخفی یک شبکه ۳ لایه تعدادی از این نورون ها را به کار ببریم و در لایه‌ی ورودی هم مکانیزم TDL

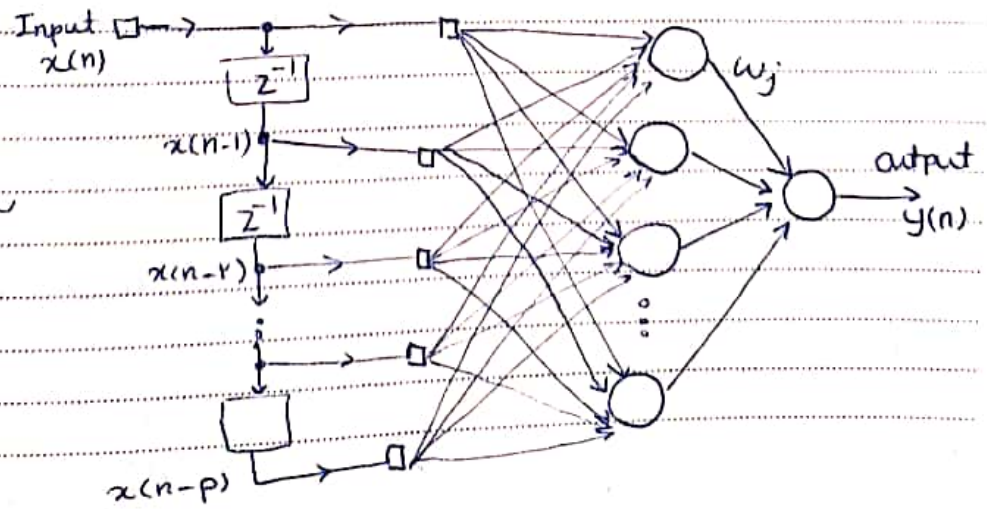
نمونه‌های سری زمان را ایجاد می کند و به گره های ورودی می دهد و توزیع می شود. یک نورون خطی هم در لایه خروجی می گذاریم

$$y(n)=\sum_{j=1}^{m_1} w_j y_j(n)=\sum_{j=1}^{m_1} w_j\,\varphi\left(\sum_{l=0}^{p} w_j(l)\,x(n-l)+b_j\right)+b_0$$

بایاس نورون خروجی

m_1 تا نورون مخفی داریم

آموزش: BP



Scanned with CamScanner

مثال: فرض کنید یک سری زمانی داریم: $\{x(n-\ell)\}_{\ell=0}^{p}$

در اینگونه مسائل وقتی می‌خواهند بسیتیک مدل چقدر خوب کار می‌کند با یک مسئله‌ای که می‌دانند نمونه‌های آینده چه چیزی خواهد بود شروع می‌کنند. در این مسئله سری زمانی از رابطه ریاضی زیر تولید شده:

$$x(n) = \sin(n + \sin(n^2)) \qquad n = 0, 1, 2, \ldots$$

تا جایی که نمونه‌ها را برای مشاهدات نیاز داریم از روی همین تابع حساب می‌کنیم. وقتی هم که مدل پیش‌بینی را انجام داد، مقدار صحیح را با همین تابع حساب می‌کنیم و تفاضلش را با مقدار پیش‌بینی شده می‌سنجیم تا خطا به دست آید.

در این شبکه درجه‌ی خط تاخیر را ۲۰ گرفته‌ایم ($p=20$)، تعداد نورون‌های لایه مخفی = ۱۰، تابع فعال‌سازی: لاجستیک

در لایه‌ی خروجی ۱ نورون خطی داریم. نرخ یادگیری هر دو لایه ← ۰٫۰۱

Momentum هم نداریم.

(شکل مثال در اسلاید)

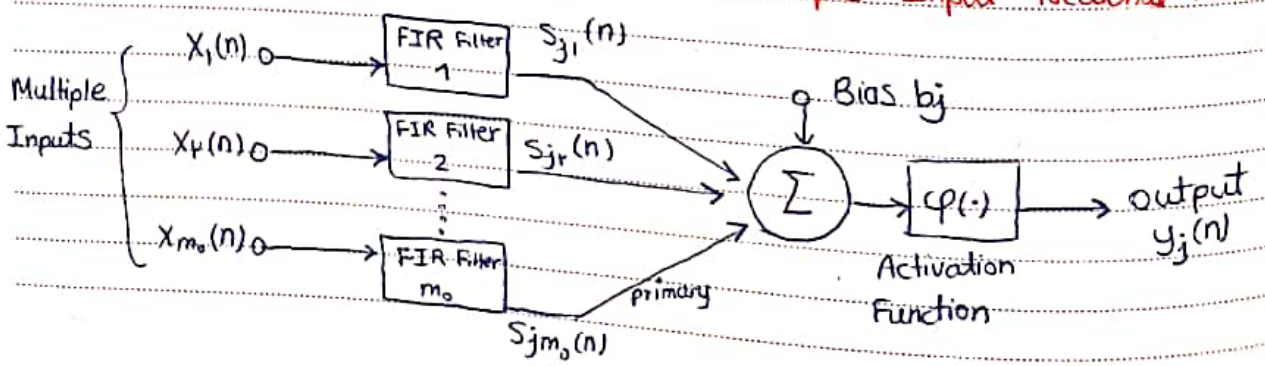
Training Data

۵۰۰ تا سری ۲۰ تایی تولید می‌کنیم با استفاده از رابطه ریاضی که داشتیم (رندم)

پس $\{x(n)\}$ ها ۲۰ تا مقدار دارد و ۵۰۰ تا از این‌ها داریم. پس از آموزش مدل، خطای پیش‌بینی در بازه $[-0.2, 0.2]$ است.

حالا اگر مقدار سری زمانی داشته باشیم که دو ورودی باشند (Spatio_Temporal):

Multiple Input Neuronal Filter



Scanned with CamScanner

فرض کنید m_0 تا سری زمانی داریم که ورودی های شبکه اند: $x_1(n), x_2(n), \dots, x_{m_0}(n)$

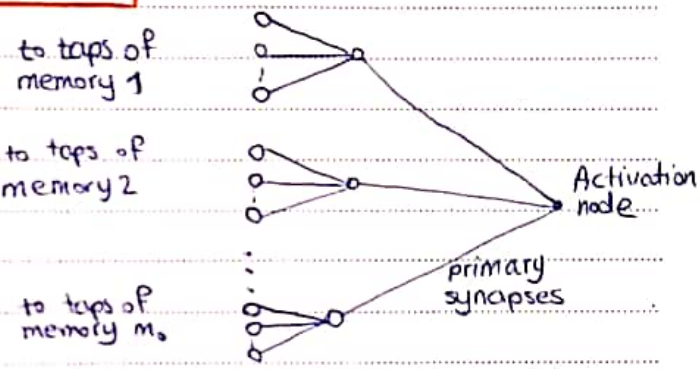
هر کدام از این ها با ساختاری مشابه شبکه ی قبلی یک خروجی تولید می کنند: $s_{j1}(n), s_{j2}(n), \dots, s_{jm_0}(n)$

ساختارها هم با عنوان FIR Filter در شکل نشان داده شده (همان ساختار شبکه ی قبلی است)

یک نورون داریم که نتایج را با هم ترکیب می کند، خروجی آن از تابع فعال سازی عبور می کند و خروجی نهایی را تولید می کند.

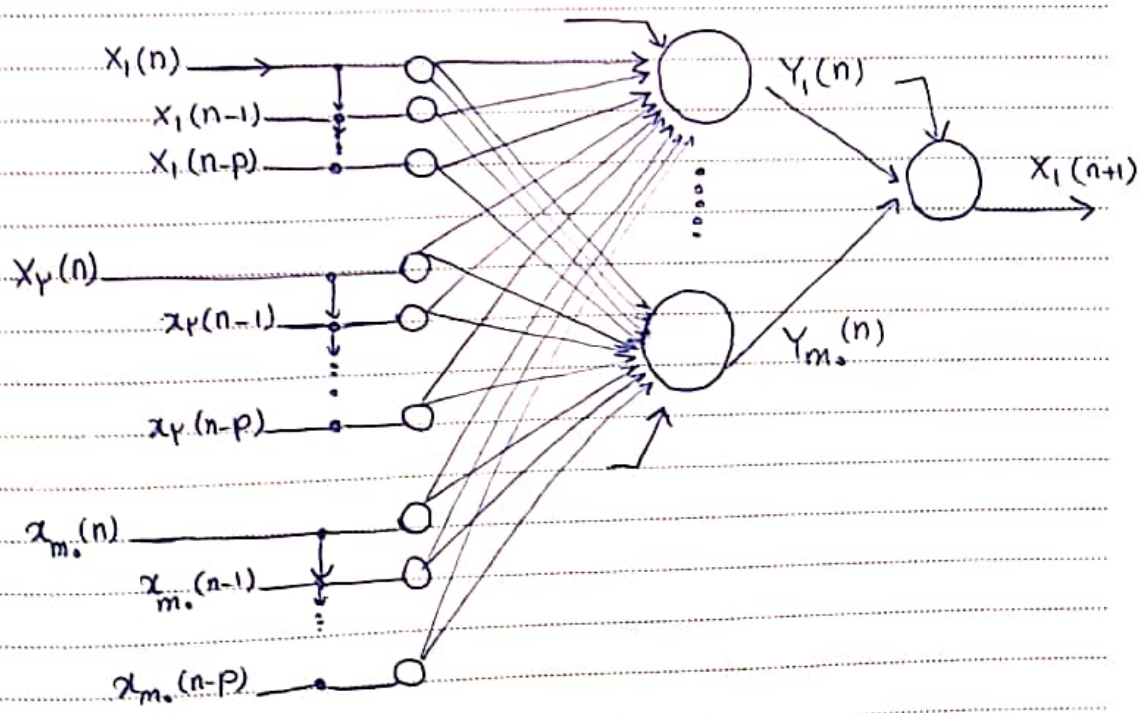
رابطه این شبکه:

$$\boxed{y_j(n) = \varphi\left(\sum_{i=1}^{m_0}\sum_{l=0}^{p} w_{ji}(l)\, x_i(n-l) + b_j\right)}$$



محبطا می خواهیم پیچیده تر کنیم و شبکه ای از نورون هایی که ورودی شان سری های زمانی متعدد داشته باشیم:

Spatio_Temporal (Distributed) Time Lagged FF Network



Scanned with CamScanner

$$S_{ji}(n) = \sum_{l=0}^{p} w_{ji}(l)\, x_i(n-l)$$

$$v_j(n) = \sum_{i=1}^{m_0} S_{ji}(n) + b_j$$

$$y_j(n) = \varphi(v_j(n))$$

$$S(n) = \sum_{j=1}^{m_0} y_j(n)$$

$$x_1(n+1) = \varphi(S(n))$$

در این جا هدف این است که نمونه ی بعدی x_1 را تولید بکنیم ولی نمونه ی بعدی x_1 فقط تحت تاثیر نمونه های قبلی x_1 نیست بلکه تحت تاثیر نمونه های قبلی x_2 و x_3 و ... x_{m_0} هم هست.

وزن های y_j ها را ۱ داریم در نظر می گیریم چون نیازی نیست که این جا دوباره وزن یادگرفته شود.

در این حالت بیشتر از یک سری زمانی پاسخ را تعیین می کند.

آموزش این شبکه مشابه شبکه های Feed Forward است.

این حالت یک گسترش غیرخطی از AR است، با ورودی های خارجی: exogenous Input variables که به آن مدل NARx گفته می شود.

برای آموزش این شبکه سه مجموعه Train ، Validation و Test نیاز است. توصیه می شود که Validation را از جایی نزدیک Train انتخاب کنیم و Test را از جایی دور از Train بردار برای اینکه بفهمیم چقدر خوب عمل می کند. چون رفتار سری های زمانی در حال تغییر است و ممکن است مقداری که در می شویم روند به طور کلی تغییر کند.

در این شبکه اتصالات کامل است.

Scanned with CamScanner

۱۴۰۰ $x(n)$: Forecast values $T(n)$: Desired values

معیارهای سنجش کیفیت پیش بینی:

$$MSE = \frac{1}{N}\sum_{n=1}^{N}(x(n)-T(n))^2$$

$$RMAE = MAE / \bar{T}$$

$\bar{T}$: میانگین مقادیر target

$$MAE = \frac{1}{N}\sum_{n=1}^{N}|x(n)-T(n)|$$

$$PI = 1 - \frac{\sum_{n=1}^{N}(x(n)-T(n))^2}{\sum_{n=2}^{N}(T(n)-T(n-1))^2}$$

← مربعات خطای تخمین نسبت به مربعات تفاضل نمونه‌های متوالی

Hybrid Linear and NonLinear Model

می‌توانیم مدل‌های ترکیبی خطی و غیرخطی داشته باشیم.

فرض کنیم L_t مولفه‌ی خطی و N_t مولفه‌ی غیرخطی مدل است. در این صورت y_t را می‌توانیم به صورت مجموع این دو بنویسیم:

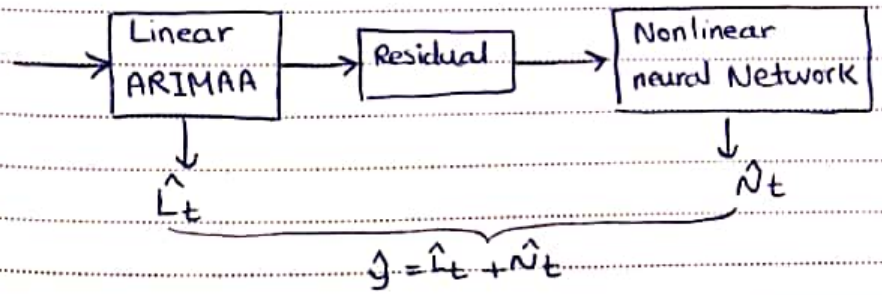
$$\boxed{y_t = L_t + N_t}$$

اگر بخش خطی را با یک مدل خطی تخمین بزنیم $\hat{L}_t$ به دست می‌آید. $\hat{L}_t$ را اگر از y_t کم کنیم خطایی پیدا می‌شود که مربوط به قسمت خطی است:

$$\boxed{e_t = y_t - \hat{L}_t}$$

حالا e_t را با یک تابع غیرخطی تخمین می‌زنیم. e_t در واقع تخمین N_t است:

$$\boxed{e_t = \hat{N} + \varepsilon_t = f(e_{t-1}, e_{t-2}, \dots, e_{t-n}) + \varepsilon_t}$$



توضیح شکل: ابتدا یک تخمین خطی که می‌زنیم $\hat{L}_t$ پیدا می‌شود. باقی مانده‌اش را می‌بریم در یک مدل غیرخطی و $\hat{N}_t$ را می‌یابیم و خروجی جمع این دو تاست. نشان داده شده که این دو با هم عملکرد بهتری از مدل خطی تنها یا مدل غیرخطی تنها دارد و اینکه مدل‌های خطی و غیرخطی ساده‌تری هم نیاز است.

Scanned with CamScanner

آیا می‌توان از همین مدلی که داشتیم برای پیش بینی چند گام آینده استفاده کنیم؟

مثلا ممکن است پیشنهاد شود که از مقادیر پیش بینی شده توسط خود مدل استفاده کنیم و به عنوان ورودی به مدل بدهیم اما این کار ایراد دارد. وقتی یک پیش بینی را به عنوان یک مقدار مشاهده شده وارد مدل می‌کنیم، داریم خطایی را وارد مدل می‌کنیم و همین طور که جلوتر می‌رویم خطا هم بیشتر می‌شود.

مثلا اگر یک قدم جلو برویم ۱ نمونه تقریبی اضافه شده، در ۲ قدم ۲ نمونه تقریبی و ...

بعد یک تاریخچه‌ای از مقادیر تقریبی خواهیم داشت که خطا ایجاد می‌کنند و یک خطای مجتمع داریم.

هم‌چنین هر چقدر اون زمانی که می‌خواهیم پیش بینی کنیم آینده‌ی دورتری باشد قدرت پیش بینی بیشتر کاهش پیدا می‌کند.

اگر مجموعه‌ی تستی داشته باشیم که مشابه مجموعه آموزش نباشد صحت در مورد آن افت می‌کند نسبت به نمونه‌ی تستی که نزدیک باشد به Training Set.

اگر چند سری زمانی ورودی داشته باشید نمونه‌های آن‌ها را هم باید وارد کنید که این جا باز هم تقریب وارد می‌شود و خطا خیلی زیاد می‌شود

⇐ مشاهده شد که اگر تمام سری زمانی که داریم (کل مشاهدات) را برای آموزش به کار ببریم، پیش بینی‌های دراز مدت بهتر انجام می‌شود

جلسه بیست و دوم — ۱۴۰۰/۹/۱۶

مقایسه‌ی مدل‌های خطی و غیرخطی

به طور کلی قدرت پیش بینی مدل‌های غیرخطی بیشتر از مدل‌های خطی است به خصوص وقتی که پیش بینی دراز مدت باشد. اگر داده‌های تست شبیه داده‌های آموزش باشند مدل غیرخطی بهتر عمل می‌کند

Scanned with CamScanner

فقط در یک حالت مدل خطی بهتر عمل می‌کند: در حالتی که داریم قدم بعدی را پیش‌بینی می‌کنیم و داده‌ها از نظر مشخصات با داده‌های آموزش متفاوت باشند، مدل خطی به صورت جزئی بهتر از مدل غیرخطی عمل می‌کند.

اما همین پیش‌بینی اگر برای چند قدم دورتر باشد، باز مدل غیرخطی بهتر عمل می‌کند.

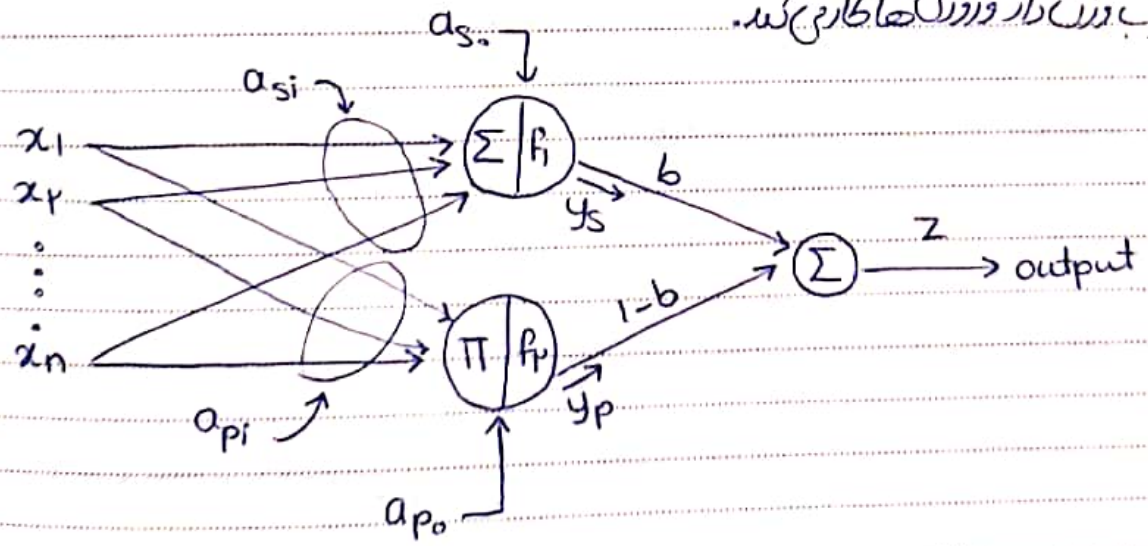
در هر دو حالت استفاده از تمام داده‌ها دقت پیش‌بینی را افزایش می‌دهد.

برای پیش‌بینی‌های بلندمدت مدل‌های غیرخطی بهتر عمل می‌کنند.

Generalized Neuron Network

یک مدل دیگر که پیشنهاد شده مدلی است که ترکیبی از حالات خطی و غیرخطی ورودی‌ها را در نظر می‌گیرد.

شبکه دو شاخه دارد: یک شاخه روی مجموع وزن‌دار ورودی‌ها کار می‌کند و یک شاخه دیگر روی حاصل‌ضرب وزن‌دار ورودی‌ها کار می‌کند.



شاخه‌ی مجموع: $y_s = f_1\left(\sum_{i=1}^{n} a_{si} x_i + a_{s0}\right) = f_1(a_{s1}x_1 + a_{s2}x_2 + \dots + a_{s0})$

شاخه‌ی حاصل‌ضرب: $y_p = f_2\left(\left(\prod_{i=1}^{n} a_{pi} x_i\right) a_{p0}\right) = f_2(a_{p1}x_1 \times a_{p2}x_2 \times \dots \times a_{pn}x_n \times a_{p0})$

خروجی: $z = y_s b + y_p(1-b)$

Scanned with CamScanner

روابط اصلاح وزن‌ها برای یک نورون که دو ورودی دارد: (قسمت حاصل جمع)

$$\Delta a_{s1_new} = \eta \sum_{i=1}^{n} (t_i - z_i)\, b y_{si} (1 - y_{si})\, x_{1i} + \alpha \Delta a_{s1_old}$$

$$\Delta a_{s2_new} = \eta \sum_{i=1}^{n} (t_i - z_i)\, b y_{si} (1 - y_{si})\, x_{2i} + \alpha \Delta a_{s2_old}$$

$$\Delta a_{s0_new} = \eta \sum_{i=1}^{n} (t_i - z_i)\, b y_{si} (1 - y_{si}) + \alpha \Delta a_{s0_old}$$

$$a_{si,new} = a_{si} + \Delta a_{si,new}$$

η ← نرخ یادگیری α ← ضریب momentum

اصلاح وزن‌ها برای قسمت حاصل ضرب:

$$\Delta a_{p1_new} = -2\eta \sum_{i=1}^{n} (t_i - z_i)(1-b)\, y_{pi} \frac{u_{pi}^2}{a_{p1}} + \alpha \Delta a_{p1,old}$$

$$\Delta a_{p2_new} = -2\eta \sum_{i=1}^{n} (t_i - z_i)(1-b)\, y_{pi} \frac{u_{pi}^2}{a_{p2}} + \alpha \Delta a_{p2,old}$$

$$\Delta a_{p0_new} = -2\eta \sum_{i=1}^{n} (t_i - z_i)(1-b)\, y_{pi} \frac{u_{pi}^2}{a_{p0}} + \alpha \Delta a_{p0,old}$$

$$a_{pi,new} = a_{pi} + \Delta a_{pi,new}$$

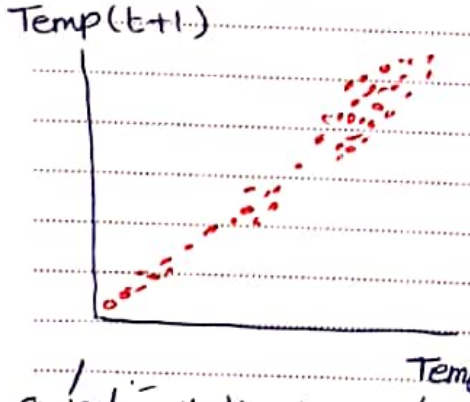
۲ | مهر جمعه

24 Sep ■ ۱۷ صفر

$$\Delta b_{new} = \eta \sum_{i=1}^{n} (t_i - z_i)(y_s - y_p) + \alpha \Delta b_{old}$$

$$b_{new} = b + \Delta b_{new}$$

مثال: پیش‌بینی درجه حرارت



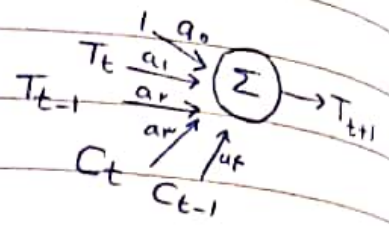
ابتدا نموداری از درجه حرارت در زمان t+1 نسبت به درجه حرارت در زمان t رسم کرده است و مشاهده شده که این دو تا رابطه‌ی خطی دارند.

به همین دلیل یک نورون خطی به صورت یک مدل ARX در نظر گرفته و فقط یک time lag را استفاده کرده (یعنی فقط نمونه‌ی قبلی)

Scanned with CamScanner

بنابراین مدلی که به دست می آید به این صورت است:

$$T_{t+1} = a_0 + a_1 T_t + a_2 T_{t-1} + a_3 C_t + a_4 C_{t-1}$$



C جریان را نشان می دهد T درجه حرارت

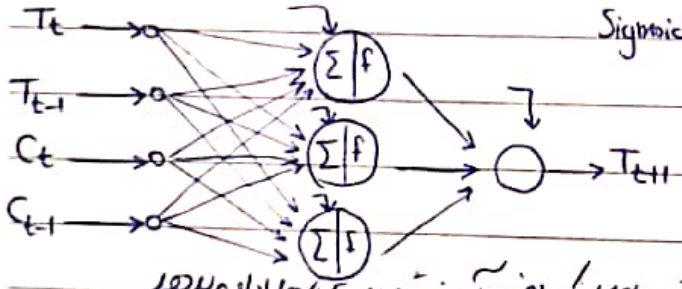
یک مجموعه داده تست داریم که شبیه داده های آموزش است و یک مجموعه داده تست داریم که شبیه نیست

خطا روی دیتاست دوم بیشتر از اولی می شود

وقتی پیش بینی را برای تعداد قدم های بیشتری انجام می دهیم خطا افزایش پیدا می کند

عملکردی که از کل دیتاست برای آموزش استفاده می کنیم خطا برای قدم های آینده کمتر افزایش می یابد

پس برای همین مسئله از یک مدل غیرخطی استفاده کرده ← NARx



در این شبکه از ۳ نورون استفاده کردن با توابع فعالیت Sigmoid

نورون خروجی خطی است

آموزش شبکه با روش LM انجام شده

Levenberg_Marquardt

مجموعه تست اول را برای Validation در نظر گرفته برای اینکه بفهمیم آموزش رو تا کجا ادامه بدهد

مشاهده می شود که عملکرد مدل غیرخطی بهتر است

روش دوم ایجاد حافظه: Feedback

در این شبکه ها اتصالات تنیدگی دار داریم. به این شبکه ها Recurrent (بازگشتی، بازخوردی) می گویند.

در این ها lag ها در اثر feedback ایجاد می شوند و شبکه حافظه دارد.

وزن های lag ها با هم متفاوت است، بهتر است به lag های اخیر وزن بیشتری بدهیم

اگر یک نورون خطی داشته باشیم با feedback ← مدل AR

اگر یک نورون غیرخطی با feedback داشته باشیم ← مدل NAR

اگر ورودی هایی غیر از متغیری که می خواهیم پیش بینی کنیم داشته باشد ← مدل NARx

اگر علاوه بر حالت AutoRegressive بودن از خطا هم استفاده کند ← مدل NARIMA

و اگر ورودی های اضافی هم داشته باشد ← مدل NARIMAx

معماری شبکه های Recurrent

ممکنه شبکه هایی درست کنیم که خیلی پیچیده باشند ولی ممکنه نتونیم تحلیل شان کنیم و هم اینکه ممکنه اصلاً

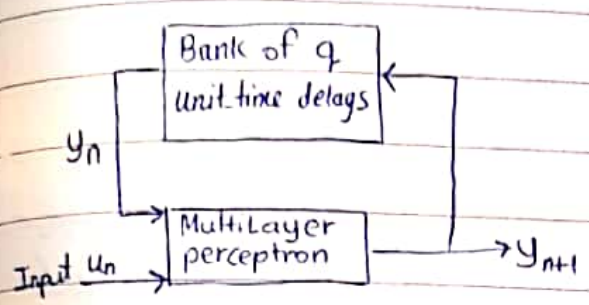
عملکرد خاصی هم نداشته باشند. می‌خواهیم شبکه‌هایی داشته باشیم که هم بتوانیم تحلیل شان کنیم و بفهمیم شان و هم اینکه مفید باشند.

معماری‌هایی که انتخاب می‌شود به این صورت است:

یک شبکه‌ی MLP را در نظر می‌گیرد. ۲ تا حالت می‌توانیم داشته باشیم:

① از خروجی فیدبک داشته باشیم به ورودی ② از لایه‌ی مخفی فیدبک داشته باشیم ← یا هر دو این‌ها

شبکه‌ی جرون (Input_output Model)



ورودی آن دنباله‌ی زمانی u_n است و توسط یک خط تاخیر مثلاً با عمق q، این داده‌ها به ورودی شبکه داده می‌شود و مقدار خروجی برای قدم بعدی (y_{n+1}) را پیش‌بینی می‌کند.

در این پیش‌بینی، خروجی قبلی هم را استفاده قرار می‌گیرد.

با استفاده از بانکی از تاخیرهای تک قدمی y_n را داریم و ورودی می‌شود به شبکه.

ورودی ← دنباله‌ی زمانی با عمق q ← Inputs: $u_n, u_{n-1}, \dots, u_{n-q+1}$

ورودی‌های u ورودی‌های exogenous هستند

یعنی ما u را نمی‌خواهیم پیش‌بینی کنیم بلکه y را می‌خواهیم پیش‌بینی کنیم

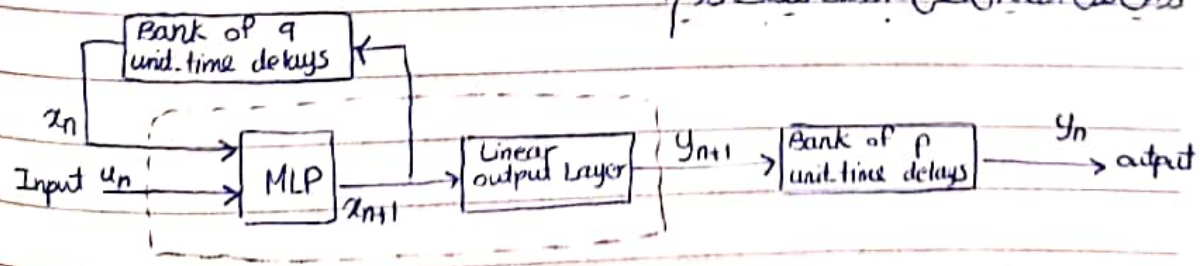
ورودی‌هایی که متغیر هدف هستند: $y_n, y_{n-1}, \dots, y_{n-q+1}$

(شکل دقیق‌تر در اسلاید)

این مدل یک مدل NARX است، رابطه‌ی آن: $y_{n+1} = F(y_n, \dots, y_{n-q+1}; u_n, \dots, u_{n-q+1})$

مدل فضای حالت (state space Model)

در این مدل از لایه‌ی مخفی به ورودی فیدبک داریم.



مجدداً ورودی u_n که دنباله‌ی زمانی است را داریم، exogenous است

در خروجی لایه‌ی مخفی MLP، x_{n+1} ای تولید می‌شود که به ورودی فیدبک داده می‌شود (دنباله q تایی)

x_{n+1} ورودی لایه‌ی آخر شبکه است.

بعد از y_{n+1} هم می‌توانیم یک خط تاخیر بگذاریم و این نمونه‌ها را ذخیره کنیم و چون ورودی u_n است

Scanned with CamScanner

ما هم y_n را به عنوان خروجی بدهیم که از خط تاخیر گذرانده می‌شود. روابط شبکه:

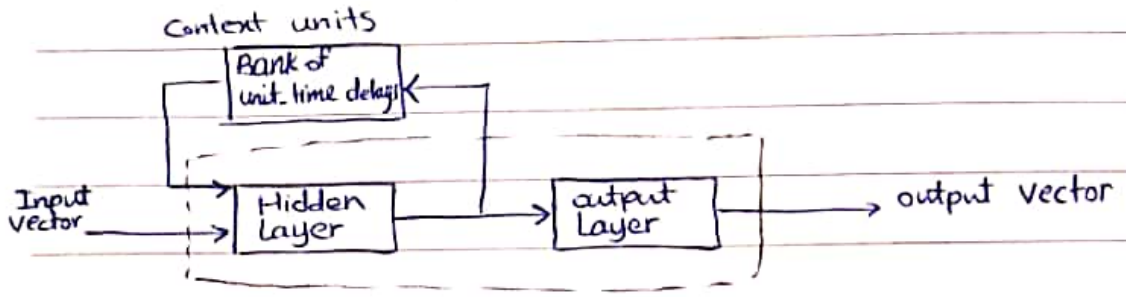
$$x_{n+1} = a(x_n, u_n)$$

$$y_n = B x_n$$

a ← یک تابع غیرخطی است که لایه مخفی را ارائه می‌کند

B → matriz of synaptic weights

Simple Recurrent network (SRN) ← حالت خاصی از این معماری را دارد، خط تأخیر خروجی حذف شده و لایه‌ی خروجی می‌تواند غیرخطی باشد (شبکه‌ی Elman)



Recurrent MultiLayer perceptron (RMLP)

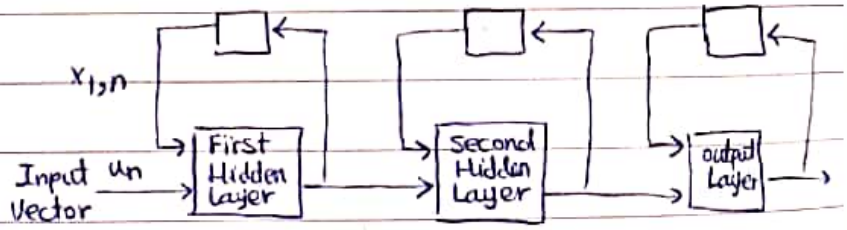
در این شبکه از هر لایه‌ی مخفی و همچنین از خروجی، فیدبک داریم

$$X_{1,n+1} = \Phi_1(X_{1,n}, u_n)$$

$$X_{2,n+1} = \Phi_2(X_{2,n}, X_{1,n+1})$$

$$\vdots$$

$$X_{0,n+1} = \Phi_0(X_{0,n}, X_{k,n+1})$$



مثال از شبکه Elman

این شبکه فقط ۳ لایه داشت و از خروجی لایه مخفی فیدبک داشت

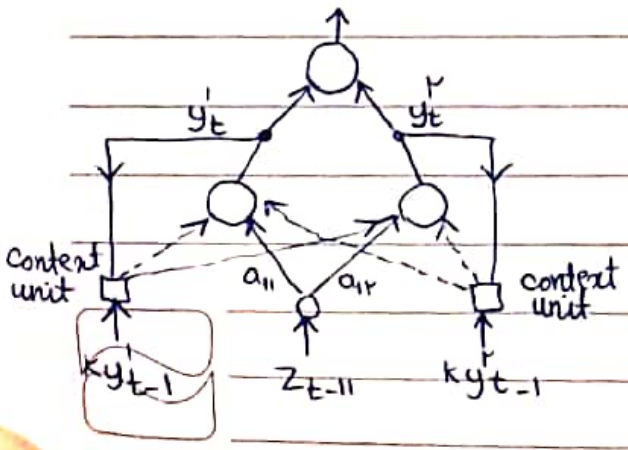
شبکه‌ای که مثال زده: 1 - 2 - 1+2 ← یعنی شبکه یک گره ورودی دارد و دو تا فیدبک که ورودی می‌شوند و اینطوری نوشید که مشخص شود

لایه دوم دو تا نورون دارد و لایه سوم یکی

برای اینکه اشتباه نکنیم گره‌های فرضی مربعی را در نظر می‌گیرند

در حقیقت وجود ندارد و به همین دلیل Context unit نامیده می‌شوند

فیدبک‌ها را می‌گیرند، اتصال از خروجی به این گره‌ها را با ضریب K می‌آورند



می‌توانیم به چشم یک شبکه ۳ لایه به آن نگاه کنیم که ۳ ورودی دارد

Scanned with CamScanner