جلسه دوم:

جلسه سوم:

اگر تابع فعالیت یکنوا افزایشی باشد به این صورت کاری میکنیم که خروجی ما بیشتر بشود یعنی اگر خروجی درست ما 1 بوده و خروجی ما کمتر بوده مثلا -1 کمک میکند بیشتر شود و همگرا میشود و بدین ترتیب تغییر وزن ها باعث کاهش خطا میشود. فرض جدایی پذیر خطی مهم هست در استفاده از پرسپترون. اگر وزن بهینه را در بیاوریم یعنی معادله خط یا مرز تصمیم گیری را بدست آورده ایم حالا بستگی به بعد هم دارد.

نحوه محاسبه مثال:

وزن را که ابتدا صفر قرار دادیم نظیر به نظیر با ورودی ضرب میکنیم از تابع فعالیت عبور میدهیم اگر خروجی با خروجی واقعی یکی بود که هیچی اگر نبود اپدیت میکنیم، چطوری؟ ورودی را در خروجی واقعی ضرب میکنیم بعلاوه یا منهای وزن فعلی میکنیم.

قضیه همگرا اگر مسئله جدایی پذیر خطی باشد در صورت وجود وزن ها مسئله به همگرایی میرسد و پاسخ همگرا خواهد بود.

Adaline تابع خروجی آن تابع خطی است چرا؟ در LMS میگفت میزان مقدار خطا مهم هست و اگر خطا بالا بود اپدیت بیشتر باشد بر خلاف پرسپترون که میزان خطا تاثیری نداشت و در نتیجه در LMS همگرایی سریع تر هست. Adaline تابع خطی دارد دیگر -1 و +1 نیست چرا؟ وقتی میخواهد تاثیر خطا در نظر بگیرد دیگر تابع نباید دو سطحی یا این یا اون باشد باید میزان خطا را در نظر بگیرد پس خروجی خطی هست. در Adaline از یک تابع فعالیت خطی رد میشود با وجود اینکه خروجی آن دو سطحی هست تنها نکته اینجا هست که بر خلاف پرسپترون اینجا مهم میشود که چه قدر خطا داریم و فقط داشتن یا نداشتن خطا مهم نیست.

جلسه چهارم:

LMS برخلاف پرسپترون میزان خطا در آن مهم هست در صورتی که در پرسپترون صرفا اگر خطا داشت میگفتی کم کن یا زیاد کن. در نتیجه همگرایی زودتر است. واسه همین هست که ما از تابع خطی استفاده میکنیم و feedback میدهیم. بعد حالا به صورت دو سطحی ها غیر خطی خروجی میدهیم. در واقع اولین تابع فعالیت ما خطی هست بر خلاف پرسپترون تا بتوانیم فیدبک بدهیم. نرخ آموزش بزرگی تغییرات وزن ها را مشخص میکند اگر زیاد باشد یعنی میزان تغییر وزن باید زیاد باشد.

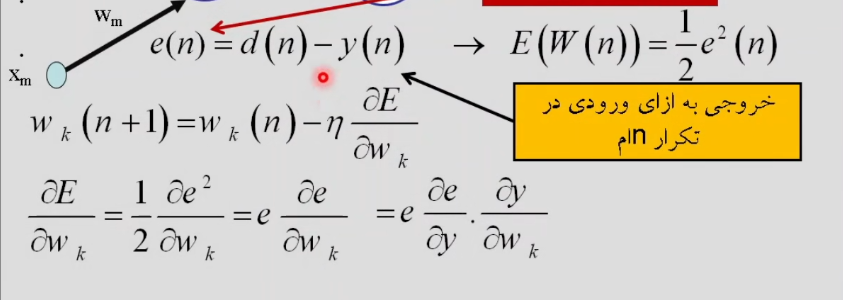
Epoch: یکبار همه ورودی ها را ببینی و خطا را حساب بکنی و MSE آن را بدست بیاوری بعد وزن ها را اپدیت بکنی، میشود یک دوره. البته batch گفته میشود به این سیستم که همه ورودی ها را ببینی و خطا را حساب بکنی و اپدیت وزن انجام بدهی که در مخالف sequence mode هست.

وزن ها پارامتر های آزاد و قابل تغییر ما هستند برای تابع خطا چون با تغییر وزن میتوانیم کاهش خطا بدهیم.

یک وزنی در نظر میگیریم تابع خطا را حساب میکنیم در عکس گرادیان یا همان مشتق وزن را بروز رسانی میکنیم یعنی اگر شیب منفی بود یا گرادیان منفی بود وزن را زیاد میکنیم و اگر شیب مثبت بود و گرادیان مثبت بود وزن را کم میکنیم. به این میگوییم steepest descent. اگر بیشتر از یک متغیر بود بردار گرادیان داریم.

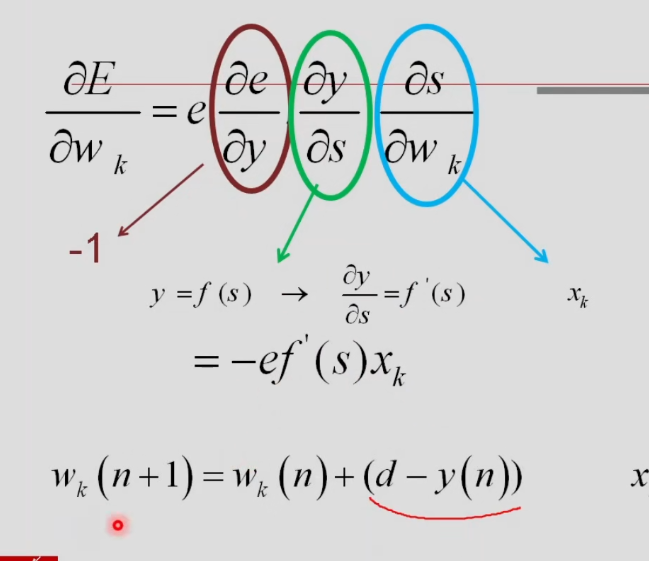
در نرخ یادگیری کم همگرایی کند و نرخ یادگیری زیاد ناپایدار میشود سیستم.

Sequential یعنی ورودی ها دونه به دونه میان و خطا حساب میشود و وزن اپدیت میشود. روند اپدیت شدن تابع خطا روند نرم تری در batch mode داریم ولی در sequential تابع ما زیاد نرم نیست. چون batch دارد میانگین خطا میگیرد دیگه.



در اینجا از مشتق زنجیره ای استفاده میکنیم بخاطر اینکه ما مشتق ارور را به صورت جزئی میخواهیم از وزن بگیریم اما مستقیم به آن دسترسی نداریم بلکه در y قرار دارد پس باید زنجیره ای بریم تا به w برسیم.

دقت کن d ثابت هست در حین مشتق گیری صفر میشود.



طبق عکس بالا واسه همین هست که مشتق تابع سیگموید در حین محاسبه خطا نیاز داریم.

پرسپترون 3 ورودی مرز جدا سازش صفحه میشود.

جلسه پنجم:

مسئله XOR جدایی پذیر خطی نیست پس با یک نورون نمیتوانیم این تابع را جدا کنیم و با یک خط نمیتوانیم پس به 2 خط حداقل نیاز داریم. میتوانیم یک نگاشت کنیم به یک فضای دیگه که جدای پذیر خطی میشود در نتیجه با یک نورون میتوانیم جدا کنیم آنها را. لایه میانی آن استخراج کننده فیچر هست که توانسته هست مسئله را ببرد به یک فضای دیگر که جدایی پذیر خطی هست. fully connected یعنی هر واحد قبلی به تمام واحدهای لایه بعدی متصل هست. تعداد واحد ها و لایه های مخفی مشخص است و تمامی اتصالات رو به جلو است یا feed forward. تابع فعالیت لایه های مخفی باید غیر خطی باشد. ترکیب یک سری لایه خطی و تابع خطی، خطی هست و دیگر عملا فایده ندارد اضافه کردن لایه های جدید و دیگر ورودی میشود یک ماتریس دیگر ماتریس بیشتر نداریم.

تک نورون: مسائل جدایی پذیر خطی را حل میکردیم که تک لایه هم گفته میشود.

دو لایه: یعنی یک لایه مخفی داشته باشیم به ازای هر نورون میتوانیم فضا را به 2 قسمت ترکیب بکنیم و نورون لایه آخر فقط خط ها را ترکیب میکند بسته به تعداد نود لایه مخفی خط ها زیاد یا کم میشوند.

سه لایه: اینجا ترکیبی از نواحی محدب داریم نسبت به لایه قبلی. و انگار یک سری ناحیه محدب میتوانیم داشته باشیم.

مشتق یک تابع نسبت به یک پارامتر یعنی تغییرات اون تابع نسبت به اون پارامتر.

پس انتشار خطا یا backpropagation:

اول forward:

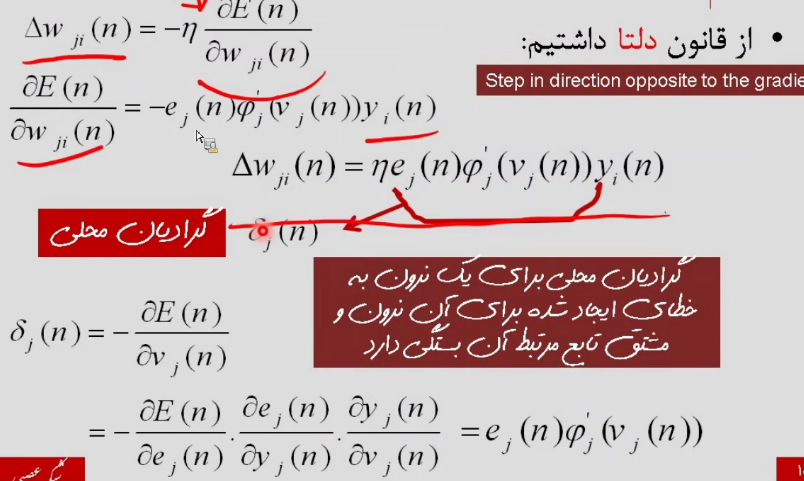
بردار ورودی به شبکه اعمال شده و خروجی واقعی محاسبه میشود.

Backward:

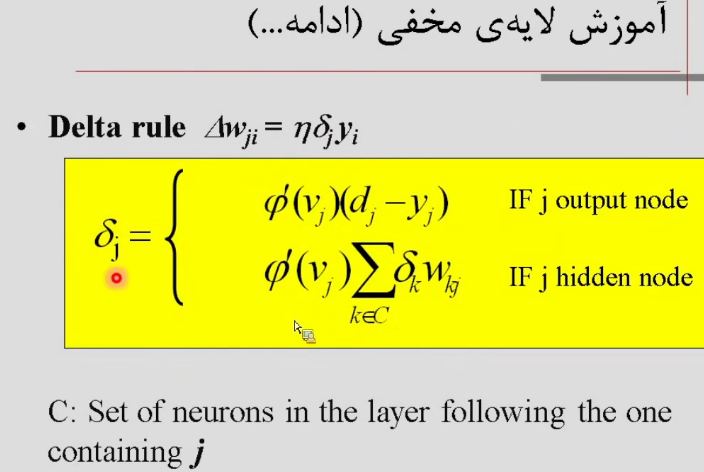
یک خروجی مطلوب داریم و یک خروجی واقعی که توسط شبکه ما تولید شده است که اختلاف این 2 تا را محاسبه میکنیم بر حسب تابع معیار سیگنالی متناسب با خطا تولید و لایه به لایه حرکت میکند و وزن را اصلاح میکند. پس از اصلاح وزن ها میگوییم iteration.

دقت کن در اسلاید میزان خطا نورون j ام نورون خروجی ما هست. خطا را برای اون نورون داریم حساب میکنیم. بعد که جمع میکنیم مجموع خطا های همه لایه های آخر هست یا خروجی به ازای هر ورودی و این لحظه ای است. حالا همین رو سیگما بگیریم تقسیم بر n کنیم میانگین خطا لحظه کل ورودی ها میشود نه فقط یک ورودی. دقت کن این با فرمول batch یکسان هست و چون میانگین میگیریم تابع خیلی نرم تر خواهد بود.   
خلاف جهت گرادیان یعنی به سمت مینیموم حرکت میکنی. به فرمول دلتا این شکلی نگاه کنی که با مشتق خطا با در نظر گرفتن وزن جهت مینیموم را پیدا میکنیم و با نرخ آموزش طول گام را مشخص میکنیم حالا به سمت منهای این باید برویم تا در جهت مینیموم باشد یا سمت خلاف آن.

گرادیان محلی را فقط یکبار محاسبه میکنی چون ثابت و مشترک هست.



نورون لایه قبلی به همه نورون های لایه بعد وصل هست و خطا آن نسبت به همه نورون های لایه بعد تاثیر گذار هست. چرا فرمول سیگما داریم روی ek؟ چون نورون لایه قبل به همه نورون های لایه بعد وصل هست پس باید مجموعه e ها را جمع کنیم برای نورون فعلی در لایه مخفی.



جلسه ششم:

Epoch: یک دوره کامل همه مجموعه آموزشی را دیده باشیم. iteration: در روش ترتیبی هست که به ازای هر ورودی که اصلاح وزن صورت میگیرد. روش ترتیبی و روش دسته ای تقریبا فرمول مشابه ای در اپدیت دارند.

شیوه ترتیبی به حافظه کمتری نیاز دارد چون برای دسته ای نیاز هست کل داده های آموزشی لود کنیم. در حالت دسته ای رسیدن به همگرایی مطمئن هست. روش ترتیبی نسبت به داده های تکراری عملکرد بهتری دارد چرا؟ چون دسته ای میانگین میگیرد و میانگین یک سری تکراری یک چیز است و فایده ای ندارد.

پردازش موازی در شیوه دسته ای مناسب تر هست. پس در کل بهتر هست یک ترکیبی از این 2 استفاده کنیم به نام mini batch که از یک سری دسته های کوچک تر تشکیل شده است. پیاده سازی ترتیبی ساده تر هست.

SGD: همان خلاف جهت گرادیان برای رسیدن مینیموم و این stochastic یعنی به صورت تصادفی داده ها وارد شوند و به جای استفاده از همه داده از یک زیر مجموعه به صورت تصادفی استفاده میکنیم.

Early stopping: اگر نسبت به مجموعه validation خطا نسبت به دفعه قبلی بیشتر شود یعنی مدل دارد به سمت overfit شدن حرکت میکند بنابراین فرآیند آموزش را متوقف میکنیم. به overfit شدن واریانس بالا هم گفته میشود و مدل مستعد overfit شدن است. high bias یعنی سوگیری مدل بالا هست یا underfit مستعد شدن هست و انگار مدل متعصب هست و خودش را تغییر نمیدهد و سوگیری و تعصب آن بالا هست.

جلسه هفتم:

3 فاکتور تصمیم پذیری: حجم مجموعه آموزشی 2. ساختار شبکه عصبی 3. پیچیدگی مساله که فقط مورد آخر دست ما هست و اگر پیچیدگی بالا باشد شبکه overfit میشود و به نویز هم حساس میشود.

Cross validation: کل مجموعه را آموزش نمیدهیم دو بخش میکنیم یک بخش آموزشی یک بخش validation ضمن اینکه برای توزیع داده ها نباید بهم بریزد و توزیع آنها باید حفظ شود و رندوم نباید باشد.

k-fold cross validation: در این کار مجموعه به k تا شکسته میشود و هر بار یک زیر مجموعه k مورد validation قرار میگیرد و k-1 دیگر برای آموزش میروند. برای محاسبه خطا کلی هم میانگین میگیریم.

n-fold cross validation: یا leave – one – out یعنی یکی رو بردار رو برا test بعد بقیه رو برای آموزش برمیدارد.

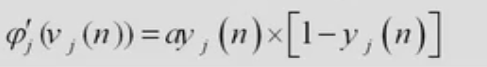
باید در cross validation تعداد افراز بالا باشد تا به یک جواب خاص بستگی نداشته باشد و median آن را گزارش میدهیم یا میانه آن ها را.

توابع فعال سازی باید مشتق پذیر و غیر خطی باشند.

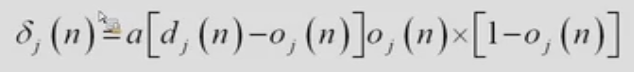
برای regression از توابع فعالیت خطی استفاده میکنیم. مقدار خروجی مطلوب باید توی برد تابع فعال سازی باشند و گرنه پارامتر های آزاد به سمت بی نهایت میروند چرا چون اگر بیشتر از برد باشد مشتق صفر میشود و اصلا آموزش صورت نمیگیرد و از طرف دیگر خروجی حالا از تابع فعال سازی بیشتر باشد تابع هعی میخواد بهش برسد پس باید وزن را بیشتر کند و نمیرسد ولی وزن به بینهایت میرود.

Vanishing gradient: در عمق شبکه که جلو میرویم و شبکه عمیق میشود گرادیان از یک محلی خیلی کوچک میشود چرا چون مشتق اون خیلی کوچک میشود و لایه های اولی خیلی خوب آموزش نمیبیند چون الگوریتم back propagation دارد خطا را بر میگرداند دیگه حالا فرض کن مشتق خیلی کوچک شود خوب مشخص است لایه های اول خوب آموزش نمیبیند.

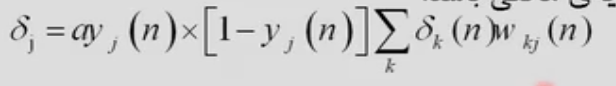
تابع سیگموید: مشتق این خیلی ساده است.

 مشتق تابع سیگموید. برای محاسبه گرادیان محلی.

فرمول گرادیان محلی با تابع سیگموید برای لایه آخر:



فرمول گرادیان محلی با تابع سیگموید برای لایه میانی:



در نتیجه اصلا دیگر نیازی به مشتق تابع سیگموید نداریم.

در تابع فعالیت tanh اگر به مقدار اشباع برویم یعنی خیلی نزدیک به 1 یا صفر آموزش کند میشود چون مشتق نزدیک به صفر میشود و vanishing gradient رخ میدهد.

Relu: بر خلاف tanh دیگر اشباع ندارد و عدم اشباع دارد. محاسبات ساده ای ام دارد. منتها منفی ها را اثر نمیدهد و نورون آموزش نمیدهد و نورون مرده است یعنی نورون هیچگاه فعال نمیشود و آموزش نمیبیند.

تاثیر نرخ آموزش:

پایین باشد دیر همگرا میشود. میتوان نرخ آموزش را متغیر در نظر گرفت یا وفقی یعنی مثلا در نزدیکی نقطه بهینه نرخ آموزش را کوچک کنیم. بردار گرادیان میخواهد بیشترین تغییرات را داشته باشد و ما باید خلاف آن حرکت کنیم پس باید بر روی کانتور ها بردار گرادیان را عمود کنیم تا بیشترین تغییر رخ بدهد . و دوست داریم این مسیر زودتر طی شود. اگر در حرکت ما به سمت بهینه تاریخچه کار را هم در نظر بگیریم و تاریخچه حرکت را همگرایی ما سریعتر میشود که به این momentum یا نرخ یادگیری وفقی گفته میشود.

Momentum:

اگر ثابت صفر باشد بروز رسانی عادی است اگر به یک باشد یعنی بیشتر از الگوی قبلی تبعیت کند. در صورتی که تغییرات هم جهت باشد در نظر گرفتن این باعث تسریع آموزش میشود که به accelerating effect گفته میشود. اگر تغییرات وزن در مرحله قبلی با مرحله فعلی هم جهت نباشد یعنی از مینیموم عبور کرده ایم و باید برگردیم و با در نظر گرفتن آن موجب پایداری میشود و این حرکت جهت مخالف صورت نمیگیرد. دقت کن هر چه قدر به ضریب آلفا توان بدهیم در هر مرحله باعث میشود تاثیرات اخیر را بهتر ببینیم و تاثیرات دورتر کمرنگ شوند یعنی یک پنجره ای از گذشته را استفاده میکنیم نه همه گذشته را. این ابزار برای تسریع همگرایی صورت میگیرد.

نرخ یادگیری خیلی پایین با ضریب مومنتوم خیلی بالا مناسب هست و باعث افزایش سرعت همگرایی میشود. در طرف مقابل ثبات یادگیری خیلی پایین یا صفر با نرخ یادگیری خیلی بالا مثل 1 مناسب هست و باعث ثبات یادگیری میشود. باید یک تناسبی بین این 2 باشد و گرنه اگر هر دو زیاد باشند باعث ناپایداری شبکه میشود و میزان خطا به صورت نوسانی تغییر کند.

اگر جهت حرکت نامناسب باشد ضریب مومنتوم را صفر بزار تا بدانی مسیر درست نیست و نرخ یادگیری را تغییر نده. اگر هم جهت بود یک مقداری نرخ یادگیری را زیاد تر بکن اما بصورت کلی مومنتوم را به مقدار اصلی برگردان. این شیوه برای آموزش دسته ای مفید است. در صورت استفاده در ترتیبی منجر به واگرایی میشود.

Annealing: به جای نرخ یادگیری وفقی نرخ آموزش به صورت نزولی در نظر گرفته میشود. بدین ترتیب در شروع نرخ یادگیری ثابت است بعد از رسیدن به محدوده مینیمم نرخ آموزش کاهش میباید. K هر چه قدر بزرگتر شود نرخ یادگیری کمتر میشود چون کسر بزرگتر میشود.

روش بهبود کارایی:

روش ترتیبی برای جاهایی که مجموعه آموزشی بزرگ داریم و افزونگی داده ها بالاست روش ترتیبی مناسب هست یعنی تکراری دارند این داده ها چون در دسته ای میانگین میگیریم برای تکراری ها این میانگین تاثیر خاصی ندارد. در مواردی که مجموعه آموزشی ثابت نیست و داده های جدید به مجموعه اضافه میشود هم روش ترتیبی مناسب است.

در آموزش شبکه بهتر هست از نمونه های آموزشی استفاده کنیم که بیشترین خطا را تولید میکنند. نمونه هایی که با هم متفاوت هستند استفاده کنیم تا طیف وسیعی از وزن ها مورد استفاده قرار بگیرد. نمونه های دشوارتر را به شبکه بیشتر اعمال کنیم. این روش ها در جهت بهبود کارایی است. 2 مشکل دارد استفاده از نمونه های دشوار تر : 1. ترتیب اعمال نمونه های آموزشی بهم میریزد. 2. Outlier ها اگر باشند باعث مشکل در تعمیم پذیری میشوند.

پیش پردازش ورودی:

داده های ورودی باید پیش پردازش بشوند و میانگین ورودی ها صفر شود و گرنه تابع خطا به سمت مینیموم زیگزاگی میشوند. بعد از، صفر کردن میانگین ورودی، ارتباط داده ها را از بین میبریم تا داده ها بهم ربطی نداشته باشند. در مرحله بعدی این داده را به شبکه میدهیم. چون اگر ورودی فقط از یک کلاس باشد قانون دلتا یا یهو بالا میره یا یهو پایین میره ولی اگه میانگین صفر باشد دلتا وزن معقول تر هست و زیگزاگی نمیشود و حرکت معقول میشود.

Covariance: داده ها را در هر جهت به واریانس تقسیم بکنی سرعت آموزش بهتر میشود چون پراکندگی داده ها در جهت یکسان میشود و اپدیت شدن تمام وزن ها به یک شیوه خواهد بود.

یکی دیگر از روش های بهبود کارایی مقدار دهی اولیه های وزن ها هست، چون از روی نقطه اولیه شروع میکنیم به مینیموم پس نقطه اولیه مهم هست اگر مقادیر بزرگ باشد میریم به ناحیه اشباع و آموزش کند میشود. اگر مقادیر کوچک باشد نزدیکی به مبدا داریم و مبدا به صورت نقطه زینی است این نقطه از هر دو جهت از یک جهت مینیموم داریم از یک جهت ماکسیموم و ما دنبال این نیستیم باید یک چیزی بین این 2 نقطه باشد.

جلسه هشتم:

Corolation یا وابستگی یعنی یکی زیاد شود اون یکی هم زیاد شود این میشود مثبت اگر اولی کم شد بقیه کم شوند میشود وابستگی منفی ولی اگر decorrelation روش انجام بدهی و وابستگی رو بگیری ارتباط از بین میرود و پراکندگی حول یک محور میشود و آموزش حول همون محور هست و آموزش بهتر میشود. پراکندگی اگر در جهت های مختلف باشد با تقسیم بر واریانس پراکندگی از هر جهت یکسان میشود و وزن ها به نوعی اصلاح میشود و آموزش بهتر خواهد شد.

Covariance باعث میشود سرعت آموزش در همه جا یکسان باشد در بخش های مختلف اگر سرعت آموزش یکسان باشد در بخش های مختلف آموزش بهتری خواهیم داشت.

اگر مجموعه آموزشی با تست متفاوت باشد نرمالایز کردن بهتر باشد.

Batch normalization هم گفته میشود که ورودی به گونه ای نرمال شود با اضافه کردن یک لایه جدید که آموزش بهتری خواهیم داشت.

میانگین ورودی صفر میشود mean removal. واریانس هم در همه جهت یکسان باشد پس 1 در نظر میگیریم.

در اثبات دقت کن که از دو تا sum یکی که اون e هست اگر k=i میشود 1 و وزن ها هم که یکی هستند پس کلی یک sum میماند.

در tanh بهتر است ما بین ناحیه 1 و -1 باشیم تا مشتق بگیریم خوب باشد و گرنه میرویم داخل ناحیه اشباع. اگر واریانس وزن m^-1/2 باشد مقدار v واریانس یک خواهد داشت. M تعداد اتصالات نورون ها هست.

همه نورون ها باید با یک سرعت آموزش ببینید. گرادیان لایه های آخر بزرگتر هست پس نرخ آموزش اون لایه ها بهتر است کمتر باشد. بهتر است نرخ آموزش بر اساس تعداد اتصالات نورون ها هم باشد.

روش های سرعت بخشیدن به همگرایی:

اگر اپدیت شدن وزن کم هست 3 دلیل دارد 1. یا خطا کم هست که خیلی خوب هست یا yj ما صفر است و یا یک است. یعنی خطا بزرگ هست ولی اون yj در سیگموید خیلی کوچک هست و نزدیک صفر هست. روش پیشنهادی این است که تابع معیار خطا را عوض کنی به جای MSE بیا و روش VAN OYEN را استفاده کن.

برای binary cross entropy حتما تابع فعالیت باید سیگموید و گرنه اون فرمول اصلا شکل نمیگیرد. هر تابع تحلیلی را به صورت سری از مشتقات مرتبه های مختلف میتوانی بنویسی.

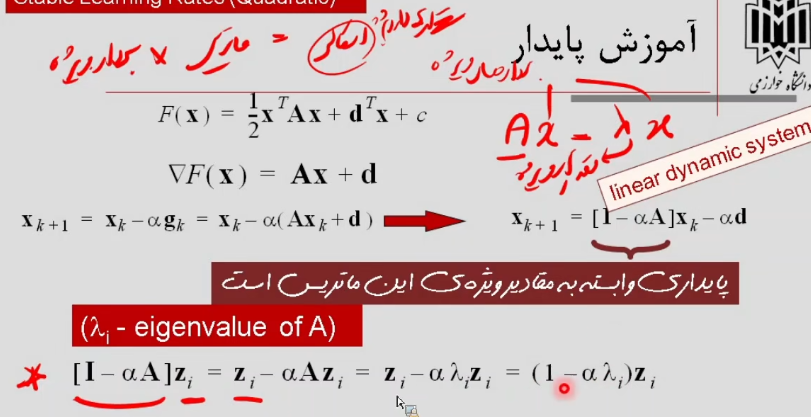
مشتقات مرتبه اول تو بردار گرادیان و مشتقات مرتبه دوم در ماتریس هسین هست.

الگوریتم های بهینه سازی برای یافتن نقطه مینیمم تابع خطا:

در اینجا پارامتر های آزاد x ها هستند. p جهت جستجو و a اندازه گام ما هست که همان نرخ یادگیری خواهد بود. دو بردار اگر عمود باشند ضرب داخلی صفر میشود. اگر کاملا روی هم باشند میشوند 1. اگر 180 درجه باشند منفی ترین حالت ممکن بدست میاد ما هم که دنبال منفی بودن gk هستیم پس این 2 خلاف همدیگه باید باشند. Pk باید خلاف جهت گرادیان باشد اگر خلاف جهت گرادیان بروی به نقطه مینیمم میرسی پس باید 180 درجه باشد. G0 بردار گرادیان هست.

آموزش پایدار:

Gk همان pk هست به همون فرمول گرادیان فکر کن.



به یک منهای الفا در لاندا میشود مقدار ویژه ماتریس هسین یا همان A. مقادیر ویژه قطر اصلی ماتریس میشوند. بردار ویژه یا 1 و صفر یا صفر و یک. ماتریس هسین رو روی قطر اصلی مشتق مرتبه دوم بزار. بعد از این فرمول آلفا اگر از اون مقدار مکس بیشتر باشد ناپایدار میشود و به مینیمم نمیرسد.

جلسه نهم:

تنظیم نرخ یادگیری:

Pk میشود search direction و ak میشود اندازه گام هست. و ما دوست داریم این عبارت مینیمم شود.

اگر بخواهیم یک تابع درجه دوم را کاهش بدهیم باید گرادیان حساب بکنیم یا مشتق بگیریم. هدف یافتن ak بهینه هست. در روش نیوتون به دنبال نقطه ای هستیم که مشتق صفر باشد.

دقت کن روش نیوتن ما را سریع به جواب میرساند اما 1. باید ماتریس ما معکوس پذیر باشد و 2. باید همیشه معادله تابع روی خطا ما درجه 2 باشد به نقطه ابتدایی خیلی بستگی دارد. تابع درجه 2 نباشد نمیتوانیم همگرایی را تضمین کنیم. در این صورت همگرایی وابسته به تابع هزینه و حدس اولیه خواهد بود. شرایط اولیه متفاوت جواب های متفاوت خواهد داشت.

P تعداد کل مجهولات هست در اسلاید ماتریس ژاکوبی. روش gaus newton از اونجایی که در مشتق دوم صفر میکند و صرف نظر میکند چون کوچک هست برای ساده کردن تا مشتق دوم نگیرد. مشکل چی هست؟ از ماتریس هسین در رفتیم ولی نیاز داریم معکوس ماتریس را بدست بیاوریم.

در Levenberg میاد سعی میکند معکوس پذیری را حل کند و میاد از یک gk استفاده میکند hk خودش تخمین ماتریس هسین هست. gk همین تخمین هست منتها عناصر قطر اصلیش با یک میو جمع شده است اگر میو صفر باشد همان روش نیوتن است. برای معکوس پذیری کافی است مقادیر ویژه ماتریس مثبت باشد. چرا g حتما معکوس پذیر هست در صورتی که نمیتوانستیم راجب h همین را بگوییم؟ چون میو را خودمان اضافه کردیم و میتوانیم اینقدر تغییر بدهیم بر خلاف نیوتن که ثابت بود تا معکوس پذیر شود و مقدار ویژه مثبت باشد. در روش نیوتن تابع خطا باید SSE باشد یا sum of squared error بعد وکتورایز کردیم بعد گفتیم a معکوس جی را باید درست کنیم.

ماتریس ژاکوبی ضربدر error میشود گرادیان.

در نهایت Levenberg میشود یک چیزی ضربدر گرادیان که همون به کاهش گرادیان مربوط است با افزایش میو در واقع رابطه میشود کاهش گرادیان و مشتق مرتبه اول. میو را کم بکنیم میشود مشتق دوم و روش نیوتن در واقع میو انگار همان نرخ یادگیری هست یا مرتبط به آن هست. میو اگه خطا کم شد کاهش میدن با ضریب تتا اگر خطا کم نشد افزایش میدهیم میو را. بهتر است با مقدار بزرگ شروع کنیم و در محدوده مورد نظر رسیدیم میو را کوچک کنیم تا همگرایی سریع شود و تابع خطا حتما باید SSE باشد. درست از حساب ماتریس هسین فرار میکنی اما حجم محاسبات بالاس برای شبکه های کوچک مناسب است و بهتر از گرادیان کاهشی هست.

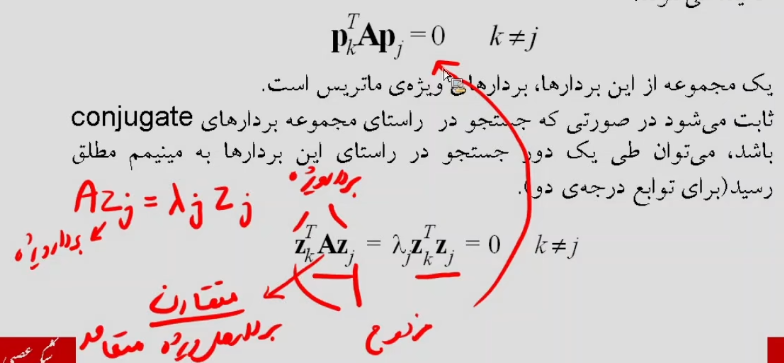
گرادیان مزدوج:

در صورتی که در راستای بردار های ویژه ماتریس هسین حرکت کنیم میتوان انتظار داشت که سرعت همگرایی افزایش یابد. F(x) تابع روی خطا درجه 2 هست.

محور های اصلی کانتور بردار های ویژه ماتریس هسین هستند و در راستای محور های اصلی کانتور میخواهیم حرکت کنیم.

ماتریس هسین یک ماتریس متقارن است.

Ax = lambda x که x میشود بردار ویژه و lambda میشود مقدار ویژه. بردار های ویژه بر هم متعامدند چون ماتریس هسین متقارن هست. ضرب دو بردار متعامد صفر میشود.



این روش مزدوج میاد و از تابع درجه 2 گرادیان میگیرد و یکبار دیگه نسبت به ایکس مشتق میگیرد و میشود A چون هر جور هست میخواهیم از محاسبات ماتریس هسین فرار کنیم. X همان پارامتر های آزاد هستند مثل وزن ها. دلتا XK همان AKPK خودمان هست. بعد به جاش تغییرات گرادیان گذاشتیم و شده صفر پس چه خاصیت باید داشته باشد؟ pj باید روی تغییرات گرادیان عمود باشد این همان بردار مزدوج هست و دیگر A را حساب نمیکنیم. این باید روی همه تغییرات G عمود باشد.

حل گرادیان مزدوج:

مرحله اول کاملا مشابه قبلی یک نقطه اولیه داریم بر خلاف جهت گرادیان حرکت میکنیم میرویم نقطه دوم حالا در نقطه دوم میگیم جهت کدام سمتی باید باشد، در هر دور بردار p باید طوری باشد که عمود بر تغییرات گرادیان باشد. با اون فرمول که pk-1 اثر گرادیان های قبلی را اثر میدهد استفاده میکنیم تا جهت جدید را بدست بیاوریم. Gk که هم گرادیان در همین دور هست.

بهتر است برای نرخ یادگیری از یک روش تکرار شونده استفاده بکنیم. یعنی میخواهیم بدونیم با چه گامی پیش بریم اگر جلو رفتیم خطا کمتر شد نقطه جدید نسبت به قبلی میگیم الفا کا مقدار خوبی هست اپسیلون رو دو برابر میکنیم نقطه بعدی بهتر بود چهار برابر میکنیم بهتر بود هشت برابر میکنیم اگر نقطه جدید بدتر بود بر میگردیم از لحاظ خطا دارم میگم نسبت به نقطه قبلی.

RBF:

MLP با اضافه کردن لایه شبکه را به فضای جدیدی میبردند برای MAP کردن. تابع linear در خروجی داریم چون مثلا برای regression است. لایه مخفی در RBF سعی میکند یک ویژگی یا feature از ورودی استخراج کند. در تابع گوسی هر چه قدر به center نزدیک تر باشیم مقدار بیشتر است. هر داده ما یک سنتر دارد به نام xi و همه داده ها به نسبت فاصله ای که دارند نسبت به سنتر یک ارزیابی انجام میشود و خروجی میدهد پس فاصله نسبت به خروج هست و هایپر پارامتر هست شکل تابع هم داریم که بهتر هست متقارن باشد.

واریانس بیشتر گوسی چاق تر. هر چه واریانس کوچک تر انتخاب ها محدود تر و هر چه بزرگتر انتخاب ها گسترده تر میشود. سیگما میزان پراکندگی را مشخص میکند.

خاصیت تقارن از مرکز برای فضای n بعدی وجود دارد.

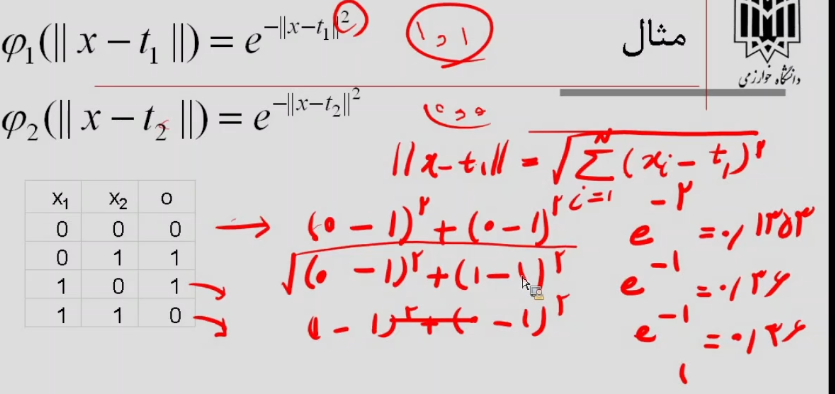
جلسه دهم:

در RBF بر اساس فاصله محاسبه میشود و به شکل متقارن است. واریانس مهم هست یا همان عرض تا پراکندگی داده را چه قدر در نظر بگیرد یا نگیرد.

به ازای هر فی در لایه مخفی باید یک سنتر و یک تابع شعاعی داشته باشیم. پس m1 تا فی در نتیجه m1 تا سنتر داریم. و محاسبه فی سنتر آن هم تاثیر دارد. در نهایت در خروجی مسئله به تابع جدا پذیر خطی تبدیل میشود تا با یک خط بتوانیم جدا کنیم یعنی خروجی یک ترکیب خطی هست. پس میگیم حول اون دو تا نقطه مثبتی که داریم تابع شعاعی در نظر بگیریم. باینری در نظر گرفتیم یا داخل دایره هستند یا نیستند. دقت کن اون 2 تا دایره مثلا برای اولی خروجی فی 1 آن 1 هست ولی خروجی فی 2 آن صفر هست. برای نمونه های منفی هم خروجی آنها صفر هست. در نهایت خروجی مسئله جدایی پذیر خطی شده است.

به فی 1 و فی 2 فضای ویژگی گفته میشود.

پس کل روند این شکلی هست که 2 تا دایره یا هر چند تا برای اون مقادیر یک کلاس در نظر میگیریم و بقیه میشوند کلاس دیگر بعد حل میکنی.



دقت کن اینجا برای فی 1 محاسبه کرده و سنتر 1و1 بوده است. بعد واسه فی 2 حساب میکنی بعد پلات میکنی روی نمودار.

پس میخواهیم از یک فضای m0 بعدی به m1 بعدی برویم. پس فی مهم است که بتواند از یک فضا به فضای بعدی ببرد. مرز تصمیم گیری هم میشه برابر با صفر همین حاصل ضرب.

حول حوش داده های ورودی یک تابع گوسی در نظر میگیریم تا بتوانیم آنها را تخمین بزنیم.

دقت کن در تابع هسته وزن ها که قرار است تابع را تخمین بزنیم، y یا قد هر کدام از داده ها هست البته وزن باید آموزش داده شود. و برای نقطه جدید که آموزش داده نشده باید توابع پایه نزدیک اون این رو تخمین بزنند و بر اساس سنتر اون وزن دارد و هر چه از این فاصله بگیریم اثر آن کمتر میشود و بقیه توابع شعاعی کاربردی ندارند. از این شیوه برای تقریب تابع و کلاس بندی هم میتوانیم استفاده بکنیم. پارامتر های آزاد RBF، سنتر ها و واریانس و وزن ها. F که قرار است پیدا کنیم قرار است دقیقا خروجی مطلوب را تقریب بزند.

دقت کن ما N تا ورودی داریم و N تا مخفی و N تا وزن و N تا خروجی مطلوب در آموزش دقیق. و دقت کن برای محاسبه خروجی باید هر ورودی فاصله اش نسبت به تمام مراکز حساب شود نظیر به نظیر ضربدر ماتریس شود و در نهایت برابر با خروجی مطلوب است. ماتریس فی ها N\*N هست. دقت کن در اینجا وزن ها مجهول هستند. پس طرفین را در یک فی معکوس ضرب میکنیم. پس نیاز داریم که معکوس را حساب کنیم و ماتریس فی باید معکوس پذیری باشد. اگر فی مربعی باشد که مشکلی ندارد و گرنه باید به مربعی تبدیلش کنیم. مثلا برای گوسی ماتریس معکوس پذیر وجود دارد. اما در کاربرد های عملیات تعداد ورودی با تعداد فی ها برابر نیست چون آموزش دقیق نیست. دقت کن خارج از بحث اگر خروجی شبکه خطی باشد برای مسائل regression مورد استفاده قرار میگیرد. اگر classifier باشد دیگر خطی نیست و مثلا step function هست برای کلاس بندی. در لایه مخفی ما ساب کلاس ها را درست میکنیم عین همون کاری که MLP با اضافه کردن لایه انجام میداد. پس چون پیچیدگی با افزایش N زیاد میشود میایم تعداد واحد های لایه مخفی را کمتر از ورودی در نظر میگیریم. مراکز میتوانند برابر با ورودی باشند یا نباشند و ممکن است یک جای دیگر باشد و نسبت به اون حساب کنیم. و دقت کن خروجی مطلوب را نمیتوانیم دقیقا حساب بکنیم بر عکس آموزش دقیق ولی میخواهیم حداکثر نزدیک باشد. هدف کمترین میزان خطا هست. ماتریس جدید N\*M است. وزن ها میشوند M\*1 و خروجی میشود N\*1. تعداد مجهولات دیگر N معادله M مجهول نیست و بیشتر است. هنگامی جواب بهینه است که حداقل خطا را داشته باشیم. برای مینیمم کردن یک چیزی هم مشتق میگیریم. دقت کن بردار توان 2 را باید بشکنی به یک transpose و یکی خودش تا بتوانی ضرب کنی.

در حین مشتق دقت کن خروجی مطلوب عدد هستند و مشتق بگیری میروند.

جلسه یازدهم:

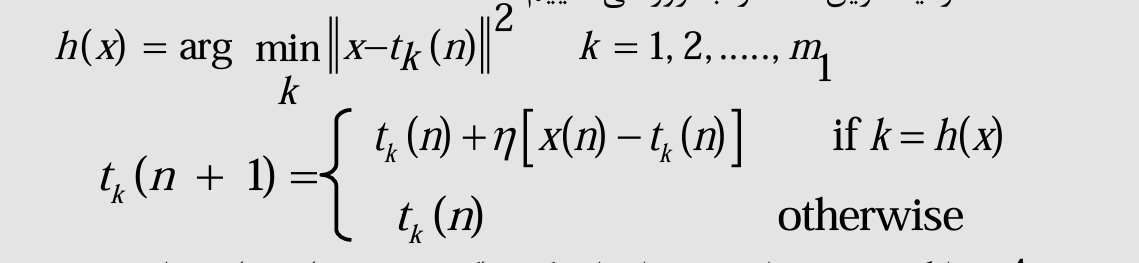
هر تصویر را میتوان به صورت جمع یک سری تصویر پایه نوشت مثل اینکه هر سیگنالی را میتوان با جمع یک سری سینوسی نوشت. تصاویر content based هستند یعنی تصویر پایه برای یک منظره فرق میکند با یک میمون.

واریانس باید به گونه ای انتخاب شود که هم همه نقاط را پوشش دهد و هم متمرکز روی یک نقطه نباشد. این همان بخاطر ماکسیموم هست.

Vu^t بردار های ویژه هستند.

محاسبه psedo inverse را سرچ کن.

با این الگوریتم سنتر به ورودی نزدیک تر میشود:



در هر iteration سنتر عوض میشود فاصله هر ورودی با همه سنتر ها محاسبه میشود و سنتر نزدیک ترین انتخاب میشود و به نوعی سعی میکند به آن نزدیک تر شود. بعد از چند دور، سنتر ها در جای درست قرار میگیرند. این شبیه به k mean است که سنتر هعی سعی میکند عوض شود و محاسبات میرود بالاتر و همچنین سعی میکند مراکز بهینه را انتخاب کند بخاطر همین پیچیدگی بالاتر میرود.

سیستم نظارت شده:

وزن ها بر اساس گرادیان کاهشی بدست میان پس ابتدا تابع خطا تعریف میشود. در این رویکرد یا الگوریتم چه مجهول وزن باشد که باید یاد بگیریم چه مراکز چه سیگما فرقی نمیکند همه را با گرادیان کاهشی حل میکند.

PNN: این شبکه برای دسته بندی استفاده میشود. MLP ها از اینها کند تر هستند و پتانسیل این را دارند MLP را از لحاظ دقت شکست بدهند. Outlier ها خیلی مهم نیستند.

معایب: وقتی یک داده جدید وارد میشود از mlp کند تر میشود. بخاطر استفاده از RBF حافظه بیشتری نیاز دارند.

Pattern layer: همان فی ها در RBF هستند.

یک حالت رقابتی در خروجی این شبکه ها به وجود میاد. در لایه جمع اینها با هم جمع میشوند و خروجی یکی از اینها برنده است.

One hot بخون.

هر اندیس را یک بردار 3 تایی میکند که متناظر آن یک میشود.

Generalized regression network:

یک لایه خطی داریم و برای regression استفاده میشود. یک راه برای انتخاب کردن وزن ها همون y هست یا قد آن. خروجی مطلوب میشود خروجی مراکز.

شبکه های عمیق:

استفاده از سیستم های قدرت پردازش بالا و حجم دیتا از عوامل موفقیت شبکه های عمیق بود.

یادگیری عمیق: معدل لایه های مخفی زیادی دارد. لایه های مخفی وظیفه استخراج ویژگی و شناسایی را دارند.

Shallow nets: عمق شبکه کم هست و تعداد لایه های مخفی کم هست. افزایش تعداد لایه ها باعث ناپدید شدن گرادیان میشود. بیش بردازش اتفاق میفتد و پیچیدگی محاسبات زیاد میشود چون هر لایه وزن و بیش بردازش میشود چون تعداد دیتا کم هست و باید دیتا زیاد کنیم و لایه زیاد باعث افزایش پارامتر های آزاد میشود. برای کنترل پارامتر های آزاد یا باید از پارامتر های مشترک استفاده بکنیم و یا از اتصالات تنک استفاده بکنیم. یک روش دیگر استفاده از یادگیری بی نظارت خصیصه ها هست. اتصالات تنک در مقابل اتصالات کامل هستند fully connectivity.

به مثال عدد نگاه کن ما یک افقی میخواهیم ولی ممکن است یکی چهار را یا بالا بنویسد یا پایین ولی برای نورون نباید مهم باشد که کجای تصویر است که به اصطلاح به این ها میگویند باید translation invariant باشد پس یک convolution میزند و میاد سطح به سطح یا افقی به افقی یا هر چی بررسی میکند در جای به جای تصویر.

اینکه لایه های ابتدایی خوب آموزش نمیبینند مشکل بیش بردازش و محو شدن گرادیان دارد. اسم این پنجره که افقی حرکت میکند از طریق اون پنجره یک پیکسل یک پیکسل بررسی میکنیم و در نورون بعدی همه این موارد را تجمیع میکنیم و اگر در همه آنها خط افقی بود یعنی بلاخره یک افقی دید در هر کدام از پیکسل اون تجمیع میکند که slide ring هم گفته میشود.

در این شبکه ها استخراج ویژگی به صورت اتوماتیک هست بر خلاف شیوه سنتی که دستی بود و کلاس بندی هم در همان مرحله هست و دیگر مثل ماشین نیست که خروجی فیچر را بدهیم که کلاس بندی شود. شبکه به گونه ای آموزش میبیند که بهترین ویژگی را استخراج کند و کلاس بندی میکند در واقع فیچر ها یاد گرفته میشوند. به اون پنجره sliding view گفته میشود که روی ورودی میزنیم و اون خروجی یک اسکالر هست. خروجی سایزش از ورودی کمتر هست چون فیلتر میشود اگر قرار است که فیلتر یا خروجی با ورودی برابر شود باید padding انجام بدهیم.

coloration va convolution: فیلتر کردن این 2 عکس همدیگر است. در کانولوشن اول قرینه میکنیم بعد همپوشانی در اون یکی این قرینه کردن صورت نمیگیرد. کانولوشن میزان شباهت اون ناحیه با فیلتر را نشان میدهد و خواص اون ناحیه بدست میاد.

کانولوشن یک بعدی: ضرب کن حالا با خودش یا قرینه آن حاصل جمع ضرب را بزار داخل مرکز فیلتر مثلا 3 تا بود مرکز میشود دومی بزار تو دومی بعد پنجره را یکی ببر جلو.

نرمالایز کردن یعنی تقسیم بر جمع وزن دار فیلتر بکن مثلا در مثال 4 هست چون از اون حدی که داریم بیشتر نشود. اگر فیلتر قرینه باشد این 2 روش کانولوشن و اون یکی با هم فرقی نمیکند. دقت کن اون 2 ماتریس ضرب کردی حاصلش کلا میشه یک عدد اسکالر دقت کن مهم هست بعد حاصل هم میدونی دیگه کلا نظیر به نظیر ضرب میکنی و جمع میکنی کل ضرب ها را . در مثال دو بعدی صفحه بعد دقت کن اگر correlation هست اولین چیزی که با یک تلاقی میکند 9 هست بعد میبریم سمت راست میشود 8 اما در convolution اولین چیزی که تلاقی میکند 1 است چون قرینه میشود. بعد جای 2 و هشت عوض میشود بعد جای 3 و 7 عوض میشود.

LeNet1: ورودی 28 در 28 است. برای داده های 2 بعدی است. برای داده هایی مناسب است که به صورت محلی ساختار منظمی دارند و relation بین آنها اهمیت دارد. اتصالات محلی است نه fully connected که اسپارس یا تنک هم گفته میشود. feature map به لایه بعد ورودی گفته میشود و وزن های به این یکسان است یعنی چی یعنی وقتی فیلتر در نظر میگیریم وزن ها برای اون فیلتر هستش پس انگار کل اون لایه میشود 3 تا وزن بخاطر فیلتر 3 تاییش و گرنه به ازای تمام ورودی ها وزن نداریم و پارامتر ها کم شده اند و مشترک هستند. یعنی فیلتر قرمز برای همه وزن ها یکسان هست. فیلتر سبز همشون وزن یکسان دارند. وقتی وزن ها یکسان باشد، و اون مشکل تعداد پارامتر های زیاد از بین میرود. چون فیلتر هعی دارد convolve میشود دیگه هعی از ورودی یک به یک شیفت پیدا میکند و محاسبه میشود پس وزن یکسان و کم هست نسبت به تمام ورودی ها نیست بنابراین انگار وزن ها یکسان هست و در جای به جای لایه میخواهد یک ویژگی استخراج شود پس ویژگی نسبت به جابجایی مقاوم است. در این حالت خروجی این لایه برای الگویی که تکرار میشود مقدار زیادی خواهد شد چون ویژگی دارد استخراج میشود. بعد که ویژگی های استخراج شد لایه بعدی روی این لایه فیلتر میزند و سعی میکند استخراج بالاتری استخراج کند در نهایت هر لایه فیچر سطح بالاتری دارد. دقت کن اون وزن ها بر اساس آموزش بدست میاد از اول مشخص نیست.

تصاویری 3 کاناله همان RGB است. 3 در عمق بخاطر همین RGB است. دقت کن در 3 کاناله کانولوشن 2 بعدی است چون فقط در 2 بعد هر کت میکند افقی و عمودی. چرا عمق از 3 شد یک؟ چون هر سری فیلتر میزنی یک اسکالر هست بعد چرا شده از 12 شده 8 تایی؟ چون padding ندارد. دقت کن اگر در عمق حرکت میکرد هم یک نتیجه اسکالر میداد و دیگر عمق 1 نمیشد و 3 میشد. دقت کن در بعدی 256 تا از اون بالایی داریم انگار 256 تا فیلتر مختلف زدیم ولی تو ابعاد هشت در هشت با عمق 1.

میتوانیم به جای یک پیکسل یک پیکسل از بیش از یک پیکسل گام برداری یا stride مثلا 2 باشد 2 تا میپرد.

Max pooling: لایه کانولوشن باعث میشود تعداد فیچر ها زیاد شود با این روش کاهش بعد میکنیم و جلوی پیچیدگی را میگیریم. خوبیش چی بود؟ جایی که نتیجه convolution بالا باشد یعنی شباهت بالاست به اون فیلتر و همین بیشترین شباهت را پاس بده به مرحله بعد و اگر یک مقداری فیچر و detail از دست دادی اهمیتی ندارد.

Pooling: یا sub-sampling در این نوع در لایه فیچر سایز تغییر نمیکند یعنی ماکس هر 3 تا رو که گذاشتیم بعد رفتیم بعدیم سایز همان هست. اما چیزی که مد نظر ما هست downsampling هست که سایز هم کوچکتر میشود در لایه فیچر.

وقتی ورودی 32 هست حاصل فیچر مپ 28 تایی هست یعنی 5 در 5 بوده است که دو تا از بالا و دو تا از راست را از دست داده است. چرا 6 تا؟ چون 6 تا فیلتر داشتیم 5 در 5.

فیلتر 5 در 5 بعلاوه 1 که میشود بایاس تعداد وزن ها برای یک فیلتر، چند تا فیلتر داریم ؟ 6 تا. حاصل این رو بردار ضرب ورودی بکن که 28 در 28 هست میشود تعداد connection ها. دقت کن تعداد پارامتر بر اساس فیلتر هست نه بر اساس فیچر مپ.

در کانولوشن 2 بعدی ما در عمق حرکت نداریم پس عمق میشود عمق همان ورودی که روی آن فیلتر زده ایم.

جلسه سیزدهم:

شبکه کانولوشن شبکه ای هست که دست کم یک لایه کانولوشن داشته باشد و کارش این است که میچرخد و سعی میکند structure یا فیچر از تصویر استخراج کند مثلا افقی یا عمودی. دقت کن فیلتر ها یاد گرفته میشود. وزن فقط برای فیلتر محاسبه میشود. و پارامتر ها کم میشود و روش دیگر اسپارس یا تنک هست و fully connected وصل نیست و به صورت محلی وصل هست. و داخل اون پنجره وزن همه یکسان است.

فریز کردن لایه های اول یعنی اینکه وزن هاش عوض نمیشوند برای یک داده دیگری بوده ولی الان روی دیتا ست خودمان میخواهیم آموزش بدهیم و وزن ها ثابت هستند. برای فریز شدن لایه های اول بهتر است یا آخر؟ مشخص است لایه های اول یا فیچر های سطح پایین تو اکثر مسائل یکسان هستند. آخرش که ادغام میشوند برای هر شبکه مهم هست یا fine tunning هم گفته میشود. شبکه های 3 بعدی حجم محاسبات خیلی زیادی دارند یعنی تو عمق هم دارد فیچر استخراج میکند و نمیصرفد.

Max pooling سایز را کوچکتر میکند تا با حجم دیتا کمتری سرو کار داشته باشیم.

جلسه چهاردهم:

Dropout: یک سری نورون را خاموش میکنیم در جریان train، یعنی وزن آن هم صفر میشود. از overfitting جلوگیری میشود و این خاموش کردن به صورت تصادفی خواهد بود تا نورون با الگوی خاصی تطبیق پیدا نکند تا جلوی overfitting را بگیرد.

Data augmentation: بیش بردازش کی بود؟ تعداد داده کم بود و شبکه تطبیق پیدا میکرد و گاهی اوقات dropout مشکل را حل نمیکرد پس نیاز داریم به افزایش تعداد داده ها و باعث تنوع داده های مورد استفاده میشود و rotate , flip کردن از این موارد است.

Transfer learning: یک مدل کامل آموزش دیده شده و وزن های آن آماده هستش مثلا برای image detection حالا میخواهیم ازش به عنوان action recognition استفاده میکنیم لایه های ابتدایی که چون دارن فیچر ابتدایی در میارن شبیه به هم هستند و حالا میخواهیم fine tunning انجام ندهیم تا وزن های لایه آخر باز باشد و دوباره train شود و وزن ها بدست بیاد. لایه کانولوشن وظیفه استخراج ویژگی را دارند. یک راه دیگر این است که بخش کانولوشن را فریز کنیم و فقط با قسمت کلاس بندی کار داشته باشیم و اون رو آموزش بدهیم. اینجوری از مدل آماده استفاده میکنیم.

Fine tune: لایه های ابتدایی که خصوصیات عمومی یا فیچر سطح پایین استخراج میکنند و مشترک هستند فریز میکنیم. برای وقتی که داده ها کم هست خوب نیست از اول آموزش بدهیم. بهتر است یک مدل دیگر بیاریم که لایه های اولیه را دست نمیزنیم و در لایه های بعدی تعداد پارامتر ها نسبت به داده دیگر حساسیت ندارد و لایه های آخر را فقط train میکنیم. یا مثلا امکانات سخت افزاری آموزش یک شبکه از ابتدا را نداشته باشیم. دقت کن فرد باید باشن چون وسط داشته باشیم.

در alexnet پنجاه درصد dropout داشته است در لایه های 8 و 9 برای جلوگیری از بیش بردازش.

Local response normalization را بخوان.

Batch normalization: فرقش با LRN این است که این scale , shiftیی که برای normalization میدهد قابل آموزش هست و بهتر عمل میکند. LRN الان منسوخ شده و از این استفاده میشود. میخواهیم داده های تستی که توزیع متفاوتی از داده های train داشته باشید اختلاف را کمتر بکنیم.