Sharif University of Technology
Department of Computer Engineering
June 2022

Hadis Ahmadian
400211524
ML for Bioinformatics Assignment #4

Question ONE .....

۱. (غلط) : **L2 loss function** به دلیل داشتن توان ۲ نسبت به خطا ها حساس تر است و این یعنی هرچقدر یک داده پرت تر باشد مقدار **L2 loss** به طور توان۲ افزایش میابد و باعث میشود مدل بیشتر سعی کند برای کم کردن خطا داده ی پرت را نیز در نظر بگیرد و از داده های اصلی دور شود.

$$L1LossFunction = \sum_{i=1}^{n} |y_{true} - y_{predicted}|$$

$$L2LossFunction = \sum_{i=1}^{n} (y_{true} - y_{predicted})^2$$

۲. (غلط) : در هر ایپاک چون داریم در جهت گرادیان تابع **loss** حرکت میکنیم **loss** روی داده های آموزش حتما کمتر میشود پس اگر **Loss** در فرایند آموزش در حال بیشتر شدن باشد یعنی **loss** روی داده های ولیدیشن دارد زیاد میشود و این به معنای **overfit** شدن است زیرا مدل بیش از حد روی داده های آموزش فیت شده و آن ها را حفظ کرده پس قدرت **generalization** خود را از دست داده است و نمیتواند داده هایی که برای آموزش ندیده یعنی داده های ولیدیشن را خوب دسته بندی کند.

۳. (غلط) : در **SGD** ما داریم روی تابع **loss** با توجه به گرادیان حرکت میکنیم. حال اگر تابع به گونه ای باشد که فقط یک بهینه ی سراسری داشته باشیم و بهینه های محلی نداشته باشیم، از هر نقطه ای شروع کنیم به آن بهینه ی سراسری خواهیم رسید (فقط تعداد ایپاک ها با توجه به نقاط شروع و هایپرپارامترها ممکن است متفاوت باشد) اما اگر بهینه های محلی داشته باشیم با توجه به نقطه ی شروع ممکن است به بهینه های محلی متفاوتی برسیم و پارامترهای متفاوتی به عنوان خروجی برگردانده شوند.

۴. (غلط): در **SGD** ما در هر مرحله گرادیان را نسبت به فقط یک داده حساب میکنیم و با توجه به آن حرکت میکنیم در حالی که در **bach GD** گرادیان هر مرحله بر حسب تمام داده های **bach** آن مرحله حساب میشود و با توجه به میانگین آن ها حرکت میکنیم. پس در حالت اول ممکن است داده های **outlier** یا داده هایی که گرادیان خیلی بزرگی را حاصل میشوند باعث یک حرکت ناگهانی شوند ولی در حالت دوم چون این داده ها با داده های دیگر بچ در نظر گرفته میشوند اثرشان کم میشود و حرکت نرم خواهد بود. پس نمودار خطا بر حسب ایپاک در **SGD** میتواند دندانه دندانه باشد اما در **bach GD** هموار تر خواهد بود.

۵. در متد **bacth normalization** با نرمال کردن (**re-center** و **re-scale** کردن)ورودی لایه ها در شبکه ی عصبی، باعث میشود شبکه **stable** تر باشد. در واقع خروجی نورون ها قبل از اینکه به ورودی لایه ی بعد داده شود نرمالایز میشود تا بیش از اندازه بزرگ یا کوچک نشود. اگر اینطور در نظر بگیریم که نرمالایز شدن روی خروجی های لایه ی قبل انجام میشود پس خروجی ها کنترل میشود، عبارت صحیح است اما اگر اینطور در نظر بگیریم که درواقع ورودی های لایه ی بعدی هستند که دارند با نرمالایز شدن خروجی لایه ی قبل کنترل میشوند عبارت غلط است (ابهام دارد)

■

Question TWO .....

۱. خیر، زیرا میتواند باعث این شود که مشتق ها برابر با ۰ شده و در نتیجه شبکه **learn** شود یعنی در هر ایپاک پارامترها فریز شده و تغییر نکنند.

۲.

خطوط عمودی:

1	0	-1
1	0	-1
1	0	-1

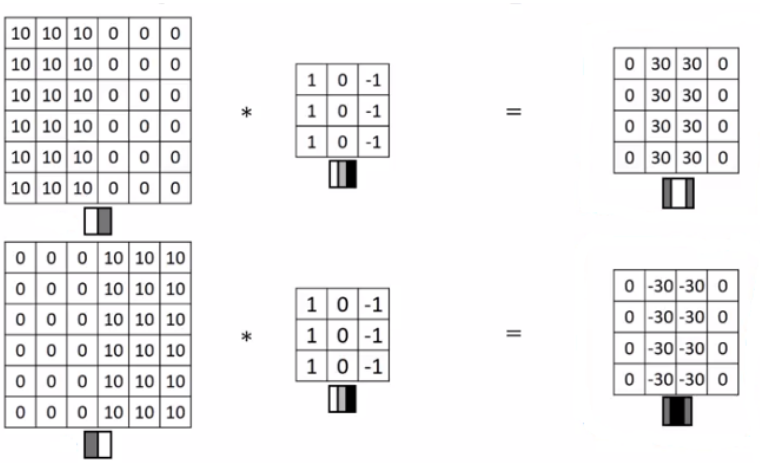
خطوط افقی:

1	1	1
0	0	0
-1	-1	-1

یک تعبیر از مرز عمودی این است که در ناحیه ای عمودی پیکسل ها به طور قابل توجهی از روشن به تاریک تغییر کنند یعنی از مقادیر بزرگتر به مقادیر کمتر. حال وقتی فیلتر فوق در چنین ناحیه ای ضرب میشود، در اطراف مرز این تغییر، مقدار زیاد (روشن) به خود میگیرد.

تعبیر دیگر نیز تغییر از تاریک به روشن یعنی مقادیر زیاد به مقادیر کم است که حاصل ضرب این فیلتر در اطراف این مرز مقادیر کوچکتر (تاریک) خواهد داشت.

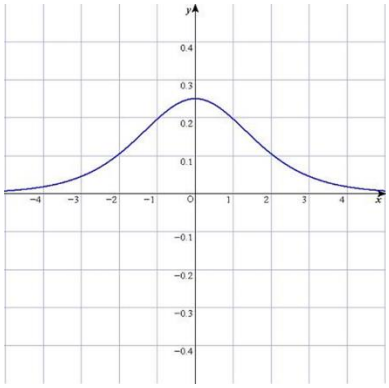
مثالی از هر دو حالت در تصویر زیر آمده است.



توضیحات برای حالت افقی مشابه حالت عمودی است با این تفاوت که تغییرات روشنایی در مرز ها به طور افقی است.

۳. فرمول مشتق تابع سیگموید به شکل زیر است و نمودار آن نیز در شکل زیر قابل مشاهده است. میتوان دید برای مقادیر بزرگ مقدار گرادیان سیگموید تقریبا ۰ است که باعث مشکل **vanishing gradiant** میشود یعنی مشتقات در بک پروپگیشین طی ضرب در محاسبات بسیار کوچک میشود و شبکه **learn** نخواهد شد. برای حل این مشکل میتوان توابع فعال ساز دیگر مانند **ReLU** یا **Leacky ReLU** را جایگزین کرد.

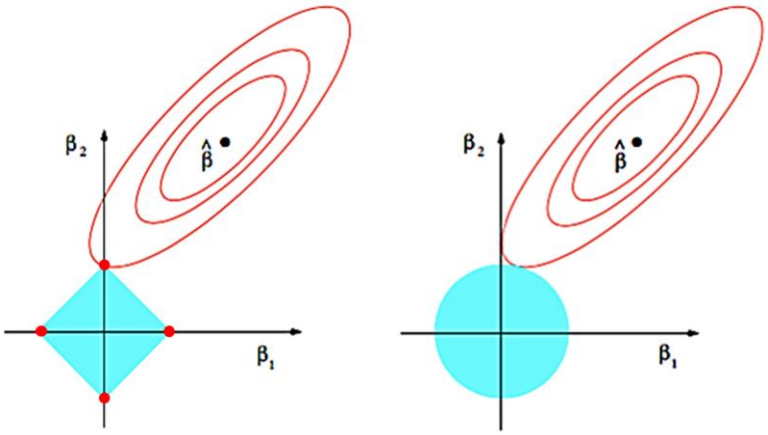
$$\frac{d\sigma}{dx} = -(1+e^{-x})^{-2} \times -e^{-x} = \frac{e^{-x}}{(1+e^{-x})^2} = \sigma(x).(1-\sigma(x))$$



۴. دقت های ذکر شده نشان میدهد که مدل روی داده های آموزش **Overfit** شده و آن ها را کاملا به خاطر سپرده در حالی که به خوبی جنرالایز نمیشود و روی داده های جدید خوب عمل نمیکند. **Dropout** یک مکانیزم برای کنترل کردن **overfit** شدن و درواقع **regulization** است. این روش به این صورت عمل میکند که در لایه های **dropout** به طور رندوم اثر بعضی نورون ها را خنثی کرده و آن ها را در محاسبات پیش رو برای لایه ی بعدی در نظر نمیگیریم (یعنی مقدار خروجی آن ها را ۰ یا غیرفعال در نظر میگیریم). این کمک میکند در هر ایپاک از آموزش، بعضی از پارامترها آپدیت نشوند و یعنی به ازای هر ایپاک بعضی از ویژگی ها **learn** نشده و ایگنور شوند که این مانع **overfitting** خواهد شد.

۵. **L1-regularization**، توضیحات : به طور کلی هدف **regularization** جلوگیری از اورفیتینگ با پنالتی دادن به وزن هاست بنابراین باعث میشود وزن فیچر های غیرضروری کم شده و فقط فیچر های مفید در نظر گرفته شود. حال **L2-regularization** وزن فیچر های غیرضروری را بسیار به صفر نزدیک میکند اما برابر با ۰ نه، این در حالی است که **L1** وزن فیچر های غیرضروری را دقیقا برابر با ۰ قرار میدهد و درواقع این یعنی به یک پاسخ اسپارس میرسد. از لحاظ هندسی علت این امر این است که **L1-norm** در فضا لبه هایی دارد که دقیقا روی پاسخ های اسپارس هستند (در مثال دو بعدی این لبه ها روی محور های مختصات هستند که برای محور **x** باعث میشود **y=0** باشد و برعکس یعنی وزن یکی از دو فیچر در دو بعد ۰ درنظر گرفته میشود و این به ابعاد بیشتر هم قابل تعمیم است) و به دلیل این شکل هندسی، ثابت میشود احتمال اینکه به نقاط اسپارس ذکر شده **converge** کنیم بیشتر است.

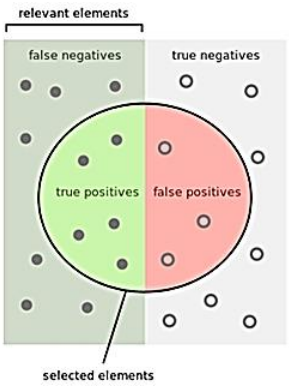
در شکل زیر یک مثال آورده شده که در آن کانتورهای تابع **loss** و هم چنین نقاط اسپارس ذکر شده در **L1** مشخص شده اند .

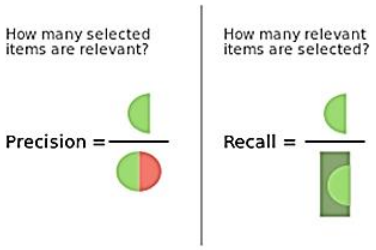


۶. **recall** : چیزی که ما میخواهیم در واقع این است که مطمین باشیم تا حد امکان تمام بیماران مشخص شوند. با توجه به تعریف **recall** و **precesion** میتوان دید **recall** گزینه ی مناسب تری برای ما است زیرا **recall** درواقع مشخص میکند چند درصد از کسانی که بیماری داشته اند تشخیص داده شده اند اما هدف **precesion** این است که ببیند چند درصد از افرادی که بیمار تشخیص داده شده اند واقعا بیمار بوده اند. با توجه به هدف بیان شده برای ما مهم تر است که تمام بیماران تشخیص داده شوند تا حد امکان افراد بیمار سالم تشخیص داده نشده و **FN** کم باشد پس بالا بودن **recall** مهم تر است تا **precesion**.

$$\textbf{Precision} = \frac{\text{True Positive}}{\text{Actual Results}} \quad \text{or} \quad \frac{\text{True Positive}}{\text{True Positive + False Positive}}$$

$$\textbf{Recall} = \frac{\text{True Positive}}{\text{Predicted Results}} \quad \text{or} \quad \frac{\text{True Positive}}{\text{True Positive + False Negative}}$$

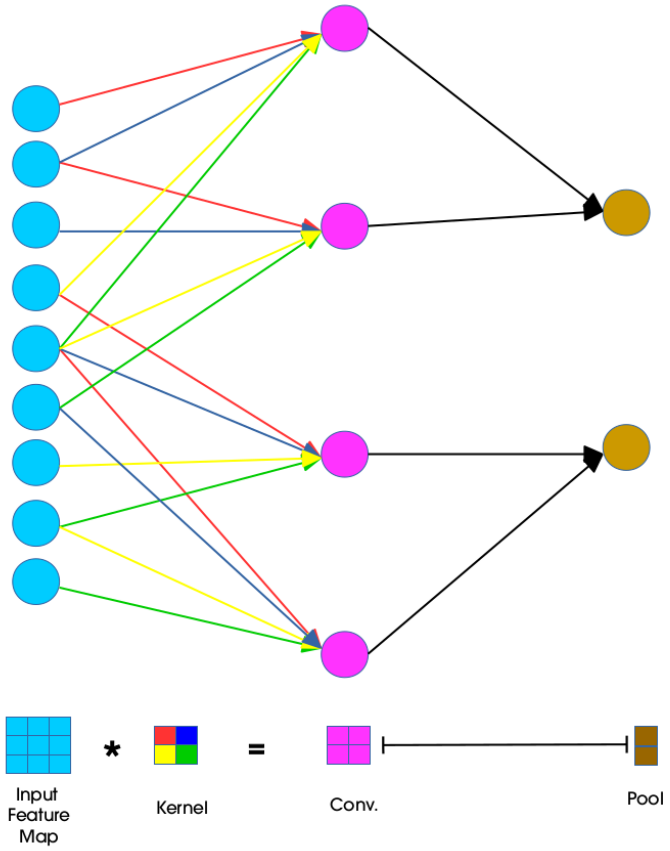




۷. مزیت اول این است که لایه های کانولوشن تعداد پارامتر ها (وزن) ی کمتری دارند و به همین علت برای داده هایی که ابعاد خیلی زیادی دارند مناسب هستند زیرا پارامترهای کمتری باید لرن شوند و این بار محاسباتی را کمتر میکند و سرعت را بیشتر.

مزیت دوم این است که در لایه ی کانولوشنال تمام ورودی ها به تمام خروجی ها متصل نیستند. این باعث میشود تمام ورودی ها روی تمام خروجی ها تاثیر نگذارند و ورودی های لوکال روی هر خروجی موثر باشند و در نتیجه فیچر های لوکال تری لرن شود و فرایند آموزش انعطاف پذیر تر شود.

در تصویر زیر میتوان هر دو مزیت را دید، هم اینکه تعداد زیادی از یال ها وزن مشترک دارند یعنی کلا تعداد فیچر هایی که باید لرن شوند کمتر هستند و اینکه همه ی ورودی ها به همه ی خروجی ها متصل نبوده و فقط ورودی های لوکال با هم به یک نورون از خروجی متصل میشوند.



۸. سیگموید به ازای مقادیر مثبت بزرگتر از ۰,۵ و به ازای مقادیر منفی کوچکتر از ۰,۵ است. حال آنکه **ReLU** به ازای مقادیر منفی صفر برمیگرداند پس ورودی سیگموید در عبارت داده شده هرگز منفی نمیشود و همواره مقادیر ۰,۵ یا بالاتر را برمیگرداند و عملا کلاس بندی ای انجام نمیشود .

۹. در قسمت ۳، مشکل سیگموید در مواجهه با مقادیر بزرگ توضیح داده شد، با توجه به شکل مشتق سیگموید واضح است این مشکل در مواجهه با مقادیر بسیار کوچک هم وجود دارد و به طور کلی باعث **vanishing gradiant** میشود. وقتی ما ورودی را

نرمال میکنیم یعنی مقادیر را در یک بازه با میانگین=۰ و std=1 نگه داریم این مشکل ورودی های خیلی بزرگ یا خیلی کوچک را تا حد خوبی حل میکنیم و باعث میشویم در بک پروپگیشن دیگر گرادیان vanish نشود.

■

Question THREE ...,,,.....

۱. استفاده از اکتیویشن سیگموید مشکل اصلی است و میتوان برای حل مشکل از ReLU استفاده کرد. چون سیگموید نمیتواند خروجی بین ۰ تا ۱۵ داشته باشد. راه حل دیگر این است که اعداد ۰ تا ۱۵ را قبل از دادن به AE، نرمالایز کنیم تا بین ۰و۱ قرار بگیرد و سیگموید بتواند آن ها را تولید کند.

۲. اگر خطا روی داده های آموزش نیز در همین حدود باشد، یعنی مدل خوب کار کرده اما اگر خطا روی داده های آموزش مقدار بسیار کمتری باشد میتواند به این معنا باشد که مدل overfit شده و جای بهبود دارد.

۳. این نشان میدهد که آتوانکودر ما robust نیست و این میتواند ناشی از این باشد که داده ی ورودی outlier های زیاد یا نویز زیادی داشته که رفع این مشکلات باعثrobust تر شدن آتوانکودر میشود و دیگر با تغییرات جزیی لایه ی latent تغییر زیاد نخواهد کرد. درواقع با از بین بردن نویز ها و outlier ها میتوانیم باعث شویم آتوانکودر در لایه های latent فیچر های مناسبی از تصاویر ورودی را یاد بگیرد به نحوی که از روی آن ها قادر به ساخت مجدد تصاویر باشد و با تغییرات کوچک تغییر زیادی نداشته و robust باشد. یک تکنیک خوب برای رفع نویز میتواند median filter یا FFT باشد.

۴. این که یک AE بتواند داده های ورودی را به شکل بسیار خوبی در خروجی بازسازی کند ناشی از این است که ورودی ما این قابلیت را داشته که در ابعاد کمتر نیز بیان شود. یعنی ۲۰۰۰ ویژگی ورودی با تنها ۲۰ ویژگی قابل بیان بوده اند. این میتواند به این معنا باشد که تعداد زیادی از فیچر های ورودی correlation بالایی داشتند و قابل تولید از روی یکدیگر بوده و تمام آن ها برای بیان داده ضروری نبوده اند.

■

Question FOUR .....

۱. vanishing gradiant به این معناست که هنگام back-propagate کردن در شبکه، گرادیان به مرور کوچکتر و کوچکتر میشود و به مقداری بسیار نزدیک به صفر میرسد که علت آن اصولا مقدار کوچک تابع فعال ساز است (برای مثال مقادیر مشتق تابع سیگموید بین ۰ تا ۰,۲۵ هستند) و چون برای محاسبه ی مشتق از قانون زنجیره ای استفاده میکنیم چندین بار ضرب این مقادیر کوچک در طی back-propagate کردن منجر به این میشود که گرادیان تقریبا ۰ شود (پس منطقی است که هرچه شبکه عمسق تر شود یعنی تعداد این لایه ا و در نتیجه ضرب ها بیشتر شود این مشکل جدی تر میشود) و در نتیجه وزن ها و بایاس ها تغییر نکنند و در نتیجه GD هرگز به مقدار بهینه converge نکند.

Exploding gradiant نیز دقیقا برعکس مشکل قبل است و به این معناست که گرادیان بیش از حد بزرگ شود و در back propagation در طی لایه های متوالی آن قدر بزرگ شود که باعث شود وزن ها و بایاس ها در هر آپدیت diverge کنند. و ممکن است در طی training مقادیر وزن ها NaN شود. شرایطی که تحت آن این مشکل رخ میدهد میتواند ناشی از مقداردهی نامناسب وزن های اولیه باشد به این صورت که اگر وزن های اولیه بسیار بد مقدار دهی شده باشند و منجر به یک خطای بسیار زیاد شوند، قاعدتا گرادیان خطا در راستای وزن ها نیز مقدار بزرگی خواهد داشت و این ممکن است منجر به explode شدن گرادیان شود به خصوص در صورت استفاده از توابع فعال سازی مانند ReLU که برای مقادیر مثبت مشتقش ۱ است و بنابراین ضرب آن در مقادیر بزرگ کمکی به کوچک تر کردن آن ها نمیکند.

برای حل این دو مشکل باید توجه داشت که:

الف) برای حل exploding gradint باید مقدار دهی اولیه وزن ها مناسب باشد. یک روش مناسب برای این مقدار دهی میتواند Xavier Glorot initialization باشد (لینک مقاله ی مربوط).

به طور خلاصه طبق این مقاله مقداردهی وزن ها در هر لایه میتواند به صورت زیر باشد: (fan_{in} تعداد ورودی های یک لایه و fan_{out} تعداد نورون های یک لایه (خروجی ها) و fan_{avg} میانگین این دو مقدار برای آن لایه است)

- توزیع نرمال با میانگین ۰ و واریانس $\sigma^2 = 1/ fan_{avg}$
- یا توزیع یونیفرم بین $-r$ و r که $r = sqrt(3 / fan_{avg})$

ب) برای پیش نیامدن vanishing gradiant به جای sigmoid از توابع فعال ساز دیگر مثل ReLU استفاده کنیم (البته چون مشتق ReLU در مقادیر منفی ۰ است، بهتر است از سایر فرم های آن مانند Leaky ReLU استفاده کنیم که برای مقادیر منفی شیبی بسیار کوچک ولی غیرصفر خواهد داشت.)

* * *

۲.

۲.(آ). در ابتدا داده ها را شافل میکنیم سپس به طور رندوم درصد کمتری از آن ها (برای مثال، ۲۰ درصد) را به عنوان داده ی ولیدیشن انتخاب کرده و سایر آن ها (۸۰ درصد باقی مانده) را به عنوان داده ی آموزش برای آموزش استفاده خواهیم کرد.

۲.(ب). خروجی های اسکنر های متفاوت با یکدیگر تفاوت هایی دارند و اگر شبکه فقط داده های اسکنر نوع ۱ را به عنوان داده های آموزش دیده باشد خصوصیات خاص مختص به آن اسکنر در روند آموزشش تاثیر میگذارد و به دلیل تفاوتی که داده های اسکنر نوع ۲ دارد، ولیدیشن دارای loss زیادی خواهد بود.

میتوان از رویکرد cross-validation استفاده کرد یعنی طی چندین iteration که هربار بخش متفاوتی از داده ها ولیدیشن و بخش متفاوتی آموزش هستند استفاده کرد تا احتمال اینکه هربار ولیدیشن و آموزش داده های اسکنرهای مجزا باشند کم شود.

راهکار دیگر این که قبل از تقسیم داده ها روی آنها پیش پردازشی صورت گیرد که تاثیر این تفاوت اسکنر ها را خنثی کند برای مثال داده ها در ابتدا **normalize** شوند (میانگین ۰ و **std=1**) و سایر تبدیلات.

۲.(ج). روش های زیادی برای داده افزایی وجود دارد به خصوص در مورد عکس. از جمله ی آنها میتوان به تبدیلات **flip** کردن، **rotate** کردن،**crop** کردن، اضافه کردن نویز گاوسی، **scale** کردن روی تصاویر اصلی اشاره کرد. حتی یک راه دیگر استفاده از **Conditional GAN** ها برای تغییر عکس های موجود از یک دامنه به دامنه ی دیگر استفاده کرد (برای مثال تبدیل فصل یک عکس از طبیعت) علت اینکه این روش های داده افزایی موثر واقع میشوند و درواقع به آموزش شبکه کمک میکنند این است که در لایه های اولیه ی شبکه ، تفاوت های کوچک ذکر شده رو داده ها برای شبکه اینگونه دیده میشود که این داده ها کاملا جدید هستند زیرا در لایه های اولیه فقط پترن های لوکال در حال کشف شدن هستند و مثلا تصاویر زیر از دید شبکه یک تصویر نیستند و داده ی جدید محسوب میشوند.

استفاده از این تبدیلات برای داده افزایی و افزودن عکس های تغییر یافته به دیتابیس یک مزیت دیگر نیز دارد و آن اینکه ممکن است در دیتابیس ما تمام تصاویر از یک زاویه ی خاص یا تحت شرایط یکسانی گرفته شده باشند حال آنکه نمونه های دیگر که به عنوان تست قرار است به شبکه داده شوند در شرایط متفاوتی به دست آمده باشند و این موضوع که ما به ازای هر داده شرایط متفاوتی از آن را تغییر داده ایم باعث میشود شرایط مختلف در نمونه های دنیای واقعی کاور شود و شبکه ی **train** شده، نسبت به این تغییرات **roubost** تر باشد.

۲.(د).۱.

از ابعاد ورودی تصویر (۱۲۸*۱۲۸) این برداشت شده است که تصویرورودی یک کانال دارد.

	وزن ها	بایاس ها	ابعاد ورودی	ابعاد خروجی
Conv-9-32	9*9*32	32	۱۲۸*۱۲۸	۳۲*۱۲۰*۱۲۰
POOL-2	0	0	۳۲*۱۲۰*۱۲۰	۳۲*۶۰*۶۰
CONV-5-64	5*5*64	64	۳۲*۶۰*۶۰	۶۴*۵۶*۵۶
POOL-2	0	0	۶۴*۵۶*۵۶	۶۴*۲۸*۲۸
CONV-5-64	5*5*64	64	۶۴*۲۸*۲۸	۶۴*۲۴*۲۴
POOL-2	0	0	۶۴*۲۴*۲۴	۶۴*۱۲*۱۲
FC-3	3*12*12*64	3	۶۴*۱۲*۱۲	۳

۲.(د).۲.

یک شبکه ی عصبی شامل **n** لایه که هر کدام **m** واحد دارند که تمام فعالساز های آن خطی هستند ولایه ی غیرخطی ندارد، معادل خواهد بود با یک شبکه بدون لایه های **hidden**، و در نهایت تنها میتواند یک **classifier** خطی باشد.

$$y=h(x)=b_n+W_n(b_{n-1}+W_{n-1}(\dots(b_1+W_1x)\dots))=b_n+W_nb_{n-1}+W_nW_{n-1}b_{n-2}+\dots+W_nW_{n-1}\dots W_1x=b'+W'x$$

این به این معناست که افزودن لایه های بیشتر بدون لایه های غیر خطی، به پیچیدگی مدل اضافه نمیکند و قدرت مدل را افزایش نمیدهد به همین دلیل به لایه های غیر خطی نیاز خواهیم داشت.

۲.(د).۳.

از آنجا که با یک مساله ی **Multi-class classification** روبه رو هستیم و تعداد کلاس ها بیشتر از ۲ تاست، (۳ کلاس) **softmax** میتواند انتخاب خوبی برای لایه ی آخر باشد. در آخرین لایه ی تعریف شده در مساله سه نورون داریم که خروجی آن ها را به لایه ی سافت مکس میدهیم و نتیجه سه مقدار با مجموع یک میشود که میتوان آن را به عنوان احتمال قرار گیری ورودی در هر یک از یه دسته تعبیر کرد. هم چنین **soft maax** این مزیت را دارد که وزنی که به مقادیر بزرگ تر میدهد به طور نمایی بیشتر از وزنی خواهد بود که به مقادیر کوچک تر میدهد.

فرمول سافت مکس:

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

۲.(د).۴.

استفاده از **stride** بزرگ تر یعنی **overlap** کمتر در کانولوشن ها و این یعنی اندازه ی کمتر خروجی لایه که منجر به نیازمندی های حافظه ی کمتر و سرعت بیشتر میشود. از طرفی وقتی خروجی یک لایه کوچکتر باشد شبکه سبک تر میشود و محاسبات کمتر میشود.

علاوه بر فواید ذکر شده کمتر شدن **overlap** میتواند باعث شود شبکه بیشتر **generalization** داشته باشد و درواقع کمتر **overfit** شود.

۲.(د).۵.

در روش های کلاسیک ماشین لرنینگ وقتی یک سری داده داریم ، شبکه ای را از ابتدا طراحی کرده و با داده ها آن را آموزش میدهیم. اشکال این رویکرد این است که اگر داده های در دسترس ما برای آموزش خیلی کم باشند احتمال شبکه به دقت خوبی نخواهد رسید یا حتی اگر داده های زیادی داشته باشیم برای رسیدن به دقت بالا به زمان زیاد و قدرت محاسباتی بالا نیاز داشته باشیم. در این شرایط میتوانیم از **transfer learning** استفاده کنیم. در این روش از این نکته استفاده میشود که بسیاری از تسک

ها به هم شبیه هستند و اگر یه شبکه از قبل طراحی شده باشد و آموزش دیده باشد که بتواند یک سری تصویر با تسک مرتبط (مثلا نوع دیگری از سرطان) را دسته بندی کند (یا تسک های مربوط دیگر) میتوانیم برای دسته بندی سرطان جدید خود از آن بهره ببریم. ایده ی کلی این است که شبکه در لایه های اولیه ی خود فیچر های ابتدایی و ساده را یاد میگیرد مانند تشخیص خطوط و **edge** ها و ... و به مرور در طی لایه های مختلف نسبت به آن تسک خاص، فیچر های پیچیده تری را تشخیص میدهد. حال فیچر های ساده تر در لایه های ابتدایی در تسک های مرتبط احتمالا بسیار شبیه به هم خواهند بود و اگر شبکه ای برای تسک۱ آموزش دیده در لایه های اولیه ی خود آن پترن ها را تشخیص میدهد و ما میتوانیم از آن لایه های از پیش آموزش دیده استفاده کنیم (دیگر لازم نیست خودمان آن لایه ها را آموزش دهیم خصوصا اگر دیتای کافی نداشته باشیم) و لایه های نهایی را بنابر تسک جدید خود تغییر دهیم و با داده های جدید مربوط به تسک خود آموزش دهیم. مواردی که در این رویکرد نیاز است نعیین شود این است که کدام لایه ها ثابت بمانند و کدام لایه ها تغییر کنند. لایه های ابتدایی اصولا باید **freeze** شوند یعنی در طی **backpropagation** پارامترهای آن ها تغییر نکند و به مرور که به لایه های نهایی میرویم پارامترها با توجه به داده های جدید، بیشتر و بیشتر تغییر کنند و **tune** شوند. پس باید برای هر لایه هایپرپارامتر **learning rate** را معین کنیم مثلا برای لایه هایی که تصمیم میگیریم فریز شوند $LR=0$ در نظر گرفته میشود و برای لایه های نهایی مقدار LR باید نسبتا بزرگتر باشد تا طبق داده های جدید آموزش ببیند و از ترکیب فیچر های از پیش آموزش دیده ی لایه های اولیه بتواند پترن های تسک جدید را تشخیص دهد.

■

Question FIVE ……………………………………………………………………………………

۱.

$$dE / dw(f', c', k', l') = \sum_{i=0}^{Ow-1} \sum_{j=0}^{Oh-1} \frac{dE}{dO(b,f',i,j)} \times \frac{dO(b,f',i,j)}{dw(f',c',k',l')}$$

به ازای یک ورودی

این از لایه بعدی تو backpropagate رسیده به ما

این بخش رو باید محاسبه کنیم

$$dO(b, f', i, j) / dw(f', c', k', l') =$$

$$\frac{d\left(\sum_{r=0}^{C-1} \sum_{k=0}^{f_h-1} \sum_{l=0}^{f_w-1} w(f', r, k, l)\, x(b, r, S \times i + k, S \times j + l)\right)}{dw(f', c', k', l')}$$

$$= x(b, c', S \times i + k', S \times j + l')$$

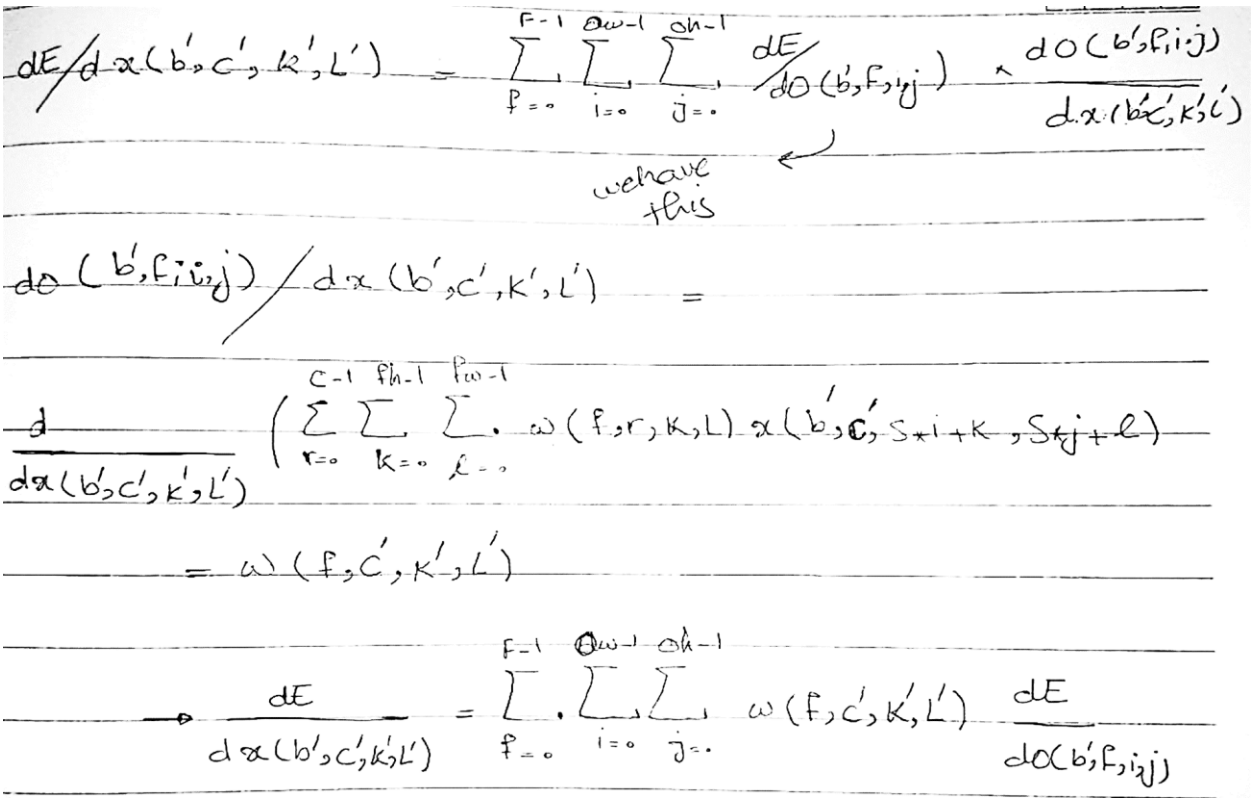
مشتق تابع هزینه به ازای یک ورودی نسبت به یک وزن در یک فیلتر محاسبه شد

حالا به ازای تمام ورودی‌ها: $\sum_{b=0}^{N-1}$ <=

به ازای تمام ورودی‌ها

$$dE / dw(f', c', k', l') = \sum_{b=0}^{N-1} \sum_{i=0}^{Ow-1} \sum_{j=0}^{Oh-1} x(b, c', S \times i + k', S \times j + l')\, \frac{dE}{dO(b,f',i,j)}$$

۲.



■

Question SIX .....

۱. از آنجا که با فشردن دکمه نمودار خطا بر حسب **iteration** کاهش میابد، میتوان حدس زد فشردن دکمه معادل افزایش هایپرپارامتر **batch size** است. زیرا هرچقدر **batch size** بزرگ تر باشد گرادیان بر حسب تعداد بیشتری داده اندازه گیری میشود و میانگین آنها حرکت نرم تری در سطح تابع **loss** را فراهم میکند و همچنین اثر **outlier** ها و نویز در میان سایر داده های **batch** خنثی میشود.

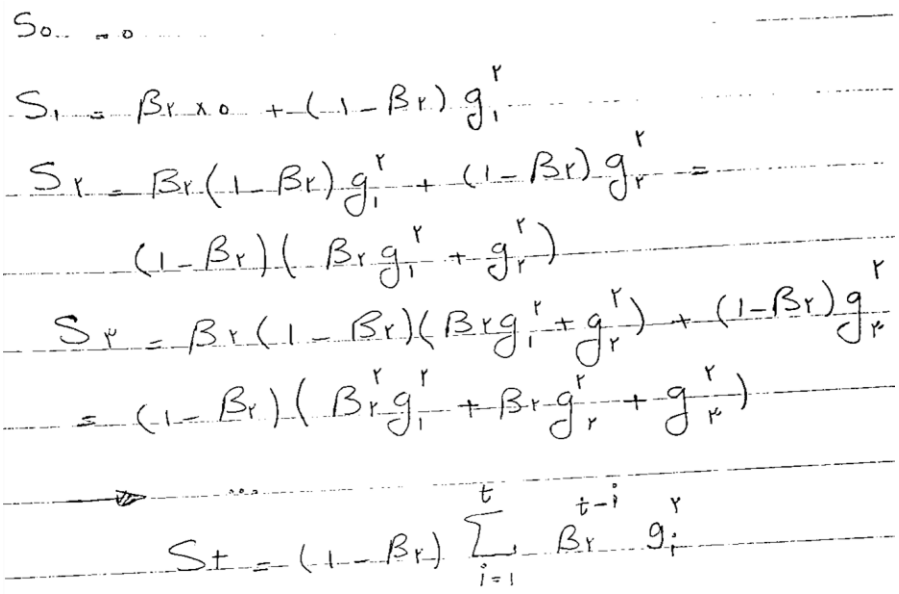
۲. اولین فایده این است که از لحاظ محاسباتی **efficient** تر است زیرا محاسبه ی گرادیان بر حسب داده های یک **batch** خیلی بزرگ یا کل داده ها توان محاسباتی بیشتری میخواهد در حالی که با **batch size** کوچکتر این عمل از لحاظ **computational** راحت تر خواهد بود.

دومین مزیت این است که اگر به هر دلیلی در بهینه ی محلی گیر کنیم، چند **bach** که داده های نویزی دارند میتوانند باعث شوند از لوکال مینیما خارج شویم این در حالیست که احتمال تاثیر داده های نویزی در **batch** های بزرگتر، کمتر میشود.

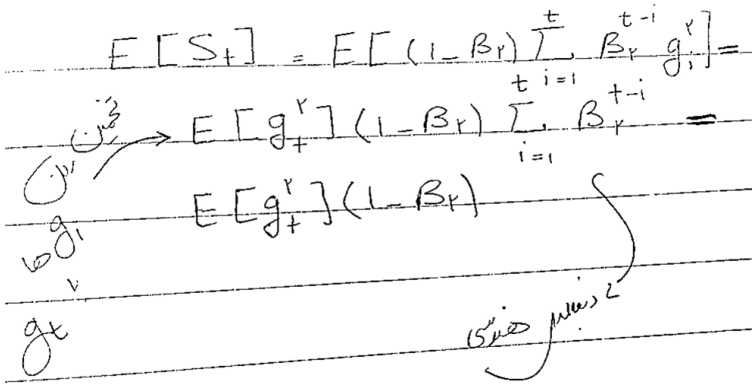
مزیت های دیگر مانند نیازمندی کمتر **memory** برای هر ایپاک و سرعت بیشتر آموزش.

۳.(آ). کم کردن β_1 به این معناست که در محاسبه ی **mt** ها، مقادیر قبلی g که مربوط به **iteration** ها ی قبل بودند، تاثیر کمتری خواهند داشت و مقدار جدید **gt** مربوط به لحظه ی **t** یعنی **iteration** فعلی، تاثیر بیشتری خواهد داشت. این یعنی اگر داده ای که در لحظه ی **t** پرادیان بر حسب آن حساب میشود نویز داشته باشد یا **oulier** باشد و گرادیان آن خیلی تغییرات ناگهانی داشته باشد، اثرش کمتر خنثی میشود این یعنی **mt** نیز بیشتر به نویز ها حساس است و بیشتر تغییر میکند پس $\frac{m_t}{\sqrt{s_t+\epsilon}}$ نیز تغییرات بیشتری خواهد داشت و مقادیر متفاوت تری خواهد گرفت (یعنی توزیع به یک توزیع یکنواخت نزدیک تر میشود) پس ارتفاع پیک کاهش یافته و عرض نمودار اضافه میشود.

۳.(ب).۱.



۳.(ب).۲.



۳.(ب).۳. چون $S_0=0$ در نظر گرفته میشود، و در **iteration** های اول هنوز مقادیر زیادی از داده ها دیده نشده و $\mathbf{gt}^2$ آنها در **St** تاثیر داده نشده، با توجه به مقدار بالای β_2 ، مقدار S به صفر نزدیک خواهد ماند وکمتر از حد واقعی خواهد بود (در واقع نسبت به ۰ بایاس میماند) و باید خیلی داده ببیند تا کم کم مقدارش به یک تخمین درست از $\mathbf{gt}^2$ برسد . با انجام اصلاح بایاس این مشکل را برطرف میکنیم.

با توجه به امید ریاضی هم میتوان دید برای اینکه S پس از چندین **iteration** به g نزدیک شود و تخمین دقیقی از آن باشد باید اصلاح بایاس را انجام داد. اگر این کار انجام نشود.

■
