سوال ۱.

الف

مقداردهی اولیه همه پارامترها با صفر در مدل‌های خطی ساده مانند رگرسیون لجستیک، گاهی اوقات می‌تواند باعث مشکلاتی شود. در مورد رگرسیون لجستیک، اگر همه وزن‌ها صفر باشند، مدل در شروع آموزش برای تمام نمونه‌ها پیش‌بینی مشابهی انجام خواهد داد چون خروجی تابع سیگموید (که در رگرسیون لجستیک استفاده می‌شود) در صفر، 0.5 است.

بنابراین، در شروع آموزش، تمام نمونه‌ها با احتمال 0.5 دسته‌بندی می‌شوند. این می‌تواند سرعت یادگیری مدل را کاهش دهد، چون گرادیان‌ها برای تمامی ویژگی‌ها یکسان خواهند بود و همه وزن‌ها با یک سرعت یاد می‌گیرند.

به همین دلیل، معمولا ترجیح می‌دهیم که پارامترها را با اعداد تصادفی کوچک مقداردهی اولیه کنیم. این روش به ما امکان می‌دهد که تنوع بیشتری در آغاز یادگیری داشته باشیم و از گیر کردن در حداقل‌های محلی جلوگیری کنیم.

همچنین لازم به ذکر است که در مدل‌های عمیق‌تر مانند شبکه‌های عصبی، مقداردهی اولیه همه پارامترها با صفر می‌تواند منجر به مشکلات جدی‌تری شود، مانند «مشکل گرادیان‌های ناپدید شونده» یا «مشکل نرون‌های مرده».

مشکل نرون‌های مرده: در شبکه‌های عصبی با توابع فعال‌سازی مانند ReLU (Rectified Linear Unit) ، اگر یک نرون در شروع آموزش یک ورودی منفی دریافت کند، آنگاه خروجی آن 0 خواهد بود (چون ReLU برای ورودی‌های منفی صفر است). اگر این نرون همچنان ورودی‌های منفی دریافت کند، گرادیان آن نیز صفر خواهد بود و وزن‌های آن در فرآیند بروزرسانی با گرادیان کاهش یافته تغییر نمی‌کند. این نرون «مرده» می‌شود و دیگر هیچ تاثیری در شبکه ندارد. اگر همه وزن‌ها با صفر مقداردهی اولیه شوند، همه نرون‌ها در ابتدا «مرده» هستند و یادگیری اتفاق نمی‌افتد.

مشکل گرادیان‌های ناپدید شونده: اگر همه وزن‌ها با یک مقدار ثابت (مانند صفر) مقداردهی اولیه شوند، در هر لایه، همه نرون‌ها یک گرادیان یکسان دریافت می‌کنند و بروزرسانی مشابهی را انجام می‌دهند. این باعث می‌شود که همه نرون‌ها همان فرایند یادگیری را تجربه کنند، که باعث محدود کردن قدرت شبکه می‌شود. همچنین، در شبکه‌های عمیق، این می‌تواند منجر به مشکل گرادیان‌های ناپدید شونده شود، که در آن گرادیان‌ها در هر لایه به طور متوالی کاهش یافته و به لایه‌های اولیه شبکه نمی‌رسند.

به همین دلیل، ما معمولا از مقداردهی اولیه تصادفی برای وزن‌ها استفاده می‌کنیم. این کمک می‌کند تا نرون‌ها در ابتدای یادگیری تفاوت‌هایی داشته باشند و شبکه بتواند از تمام نرون‌ها به طور موثر استفاده کند. برخی از روش‌های مقداردهی اولیه معروف شامل Xavier/Glorot Initialization و He Initialization هستند که بر اساس توزیع‌های خاصی از اعداد تصادفی، وزن‌ها را مقداردهی اولیه می‌کنند.

ب

اگر از Full Batch Gradient Descent استفاده می‌کنید، شافل کردن داده‌ها لزوماً تأثیر زیادی بر عملکرد یادگیری نخواهد داشت. دلیل این است که در Full Batch Gradient Descent ، گرادیان‌ها بر اساس کل داده‌ها محاسبه می‌شوند، بنابراین ترتیب داده‌ها تأثیری بر گرادیان محاسبه شده نخواهد داشت.

با این حال، در متدهای Mini-Batch Gradient Descent یا Stochastic Gradient Descent ، که در هر بروزرسانی فقط از یک زیرمجموعه از داده‌ها یا یک داده استفاده می‌کنند، شافل کردن داده‌ها می‌تواند مهم باشد. این کار می‌تواند به اطمینان برساند که مدل ما از تمامی داده‌ها به طور مناسب یاد بگیرد و از یادگیری ترتیب خاصی از داده‌ها جلوگیری کند. این می‌تواند به جلوگیری از بیش‌برازش کمک کند و عمومیت مدل را افزایش دهد.

شافل کردن داده‌ها در یادگیری Mini-Batch و Stochastic نه تنها از مدل جلوگیری می‌کند که یک الگوی خاص در ترتیب داده‌ها را یاد بگیرد، بلکه باعث می‌شود تا هر بروزرسانی وزن‌ها بر پایه‌ی یک نمونه تصادفی از داده‌ها انجام شود. این می‌تواند کمک کند تا مدل از گیر کردن در حداقل‌های محلی جلوگیری کند و به‌دنبال یافتن یک حداقل جهانی باشد، زیرا هر بروزرسانی تصادفی اجازه می‌دهد تا گرادیان به سمت‌های مختلفی حرکت کند.

این فرایند، همچنین به توزیع یکنواخت بیشتری از داده‌ها در طول آموزش منجر می‌شود، که می‌تواند به بهبود عمومیت مدل کمک کند.

به هر حال، استفاده از شافل داده‌ها یک تکنیک مفید است که معمولاً به عنوان یک قسمت از پیش‌پردازش داده‌ها قبل از آموزش مدل اعمال می‌شود. هرچند در Full Batch Gradient Descent لزومی به آن نیست، اما در آموزش‌هایی که بروزرسانی‌ها بر اساس زیرمجموعه‌هایی از داده‌ها انجام می‌شوند، می‌تواند بسیار مفید باشد.

ج

بله، شافل کردن داده‌ها در این مورد بسیار مهم است.

وقتی از روش Mini-Batch Gradient Descent استفاده می‌کنیم، ما داده‌ها را به چندین بسته (batch) تقسیم می‌کنیم و سپس گرادیان را برای هر بسته محاسبه می‌کنیم. اگر تمامی تصاویر سگ‌ها در ابتدای داده‌ها قرار داشته باشند و تمام تصاویر گربه‌ها در انتها، در طول یک دوره آموزش (epoch) ، مدل ابتدا فقط بر روی تصاویر سگ‌ها آموزش می‌بیند و سپس فقط بر روی تصاویر گربه‌ها. این باعث می‌شود مدل در قسمت‌های مختلف دوره آموزش، یک الگوی خاص را یاد بگیرد و ممکن است باعث بیش‌برازش (overfitting) شود.

وقتی داده‌ها را شافل می‌کنیم، ما از این اطمینان حاصل می‌شویم که در طول هر دوره آموزش، مدل بر روی ترکیبی تصادفی از تصاویر سگ‌ها و گربه‌ها آموزش می‌بیند، به جای یک دسته خاص. این به مدل کمک می‌کند تا تعمیم بیشتری بر روی داده‌ها پیدا کند و باعث کاهش بیش‌برازش می‌شود.

به علاوه، در صورت عدم شافل کردن، مدل ممکن است دچار تغییرات ناگهانی در گرادیان شود زیرا بسته‌های ابتدایی شامل تصاویر متفاوتی با بسته‌های انتهایی هستند. این ممکن است موجب ناپایداری در فرآیند یادگیری شود. بنابراین، شافل کردن داده‌ها می‌تواند به استقرار بیشتر در فرآیند یادگیری کمک کند.

در ادامه، شافل کردن داده‌ها قبل از آموزش می‌تواند به مدل کمک کند تا موثرتر یاد بگیرد. این امر به دو دلیل است:

پیشگیری از ترتیب خاص داده‌ها: با شافل کردن داده‌ها، ما می‌توانیم از یادگیری یک ترتیب خاص داده‌ها توسط مدل جلوگیری کنیم. اگر ترتیب خاصی در داده‌ها وجود داشته باشد و مدل بتواند آن را یاد بگیرد، این ممکن است باعث کاهش توانایی عمومی‌سازی مدل بر روی داده‌های جدید شود.

یادگیری متنوع‌تر: با شافل کردن داده‌ها، ما مطمئن می‌شویم که در طول هر دوره آموزش، مدل بر روی یک ترکیب متنوع از نمونه‌ها آموزش می‌بیند. این به مدل اجازه می‌دهد تا در طول زمان، با یک توزیع بیشتری از داده‌ها مواجه شود، که می‌تواند به بهبود عمومیت مدل کمک کند.

با توجه به این دلایل، شافل کردن داده‌ها قبل از آموزش یک مرحله مهم و پیشنهادی در پردازش داده‌ها است. حتی اگر برخی از موارد (مانند Full Batch Gradient Descent) ممکن است از این تکنیک به طور مستقیم سود نبرند، شافل کردن داده‌ها همچنان می‌تواند برای ارائه یک توزیع متنوع‌تر از داده‌ها و افزایش توانایی عمومی‌سازی مدل مفید باشد.

د

: Batch Gradient Descent (BGD)

نمودار آموزش در این روش به صورت منظم و یکنواخت کاهش می‌یابد. چراکه در BGD ، ما گرادیان را بر اساس کل داده‌های آموزش محاسبه می‌کنیم، بنابراین در هر بروزرسانی، ما به سمتی حرکت می‌کنیم که بر اساس تمام داده‌های آموزش تعیین شده است. این باعث می‌شود تا نمودار آموزش BGD خطی و یکنواخت باشد.

: Mini-Batch Gradient Descent (MBGD)

نمودار آموزش این روش می‌تواند کمی ناپایدارتر باشد. زیرا در MBGD ، ما فقط بر اساس یک زیرمجموعه از داده‌های آموزش گرادیان را محاسبه می‌کنیم. این می‌تواند باعث شود تا نمودار آموزش MBGD کمی نوسان داشته باشد، اما به طور کلی باید همچنان به سمت کاهش کلی خطا حرکت کند.

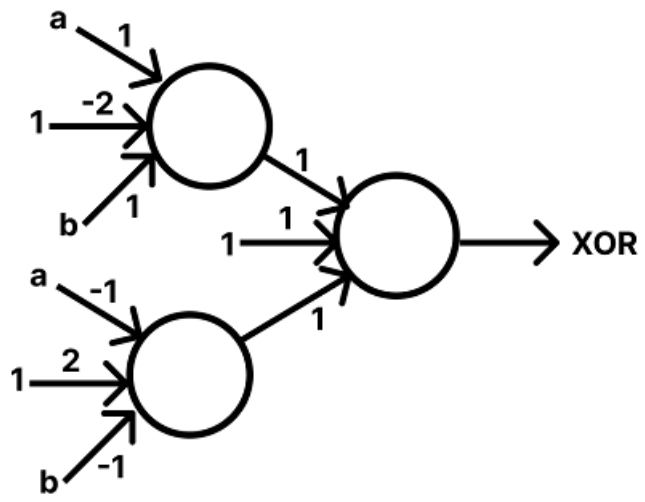
: Stochastic Gradient Descent (SGD)

نمودار آموزش این روش احتمالاً ناپایدارترین است. زیرا در SGD ، ما گرادیان را بر اساس یک نمونه تصادفی از داده‌های آموزش محاسبه می‌کنیم. این باعث می‌شود تا نمودار آموزش SGD نوسانات زیادی داشته باشد. با این حال، اگر مقدار یادگیری به درستی تنظیم شود، SGD همچنان می‌تواند به سمت کاهش کلی خطا حرکت کند، اما ممکن است به یک حالت مینیمم محلی برسد به جای مینیمم جهانی.

در نهایت، باید توجه داشت که این توضیحات فرض می‌کنند که تمامی این روش‌ها با سرعت یادگیری مناسب تنظیم شده‌اند. اگر سرعت یادگیری خیلی بزرگ یا خیلی کوچک باشد، ممکن است باعث شود نمودارها رفتار غیرمنتظره‌ای داشته باشند.

با توجه به این توضیحات، نمودار ۱ برای Batch Gradient Descent (BGD) و نمودار ۲ برای Mini-Batch Gradient Descent (MBGD) و نمودار ۳ برای Stochastic Gradient Descent (SGD) می‌باشد با توجه به میزان نوسان آن‌ها.

سوال ۲. برای طراحی یک شبکه عصبی که تساوی دو ورودی باینری x و y را خروجی دهد از XNOR استفاده می‌کنیم. می‌توانیم از دو لایه مخفی استفاده کنیم. در این حالت، ابتدا می‌توانیم ورودی‌ها را به دو گیت AND و NOR فرستاده و سپس خروجی این دو گیت را به یک گیت OR متصل کنیم. شبکه به شکل زیر خواهد بود:



این شبکه عصبی یک گیت XNOR را شبیه‌سازی می‌کند. وقتی x و y هر دو ۱ یا هر دو ۰ باشند، خروجی شبکه بیش از threshold (مثلا 0.5) خواهد بود.

اما برای تشخیص تساوی برای هر ترکیبی از دو عدد صحیح، مسئله بسیار پیچیده‌تر است. برای تشخیص تساوی دو عدد صحیح، ممکن است باید از شبکه‌های عصبی عمیق‌تر با تعداد بیشتری از نورون‌ها استفاده کنیم و از روش‌های مختلفی برای تشخیص نتایج استفاده کنیم، مثلا باید عدد را به فرم باینری تبدیل کرده و سپس هر بیت را بررسی کنیم. برای این منظور باید از الگوریتم‌های یادگیری ماشین پیچیده‌تری استفاده کنیم و ممکن است نیاز به داده‌های آموزشی بیشتری داشته باشیم.

سوال ۳.

الف

اگر بایاس نداشته باشیم:

$$۲^۶ \times ۲^۶ \times ۲۰۴۸ + ۲ \times ۲۰۴۸ = ۸۳۹۲۷۰۴$$

اگر هم بایاس داشته باشیم:

$$۲۰۴۸ \times (۴۰۹۶ + ۱) + ۲ \times (۲۰۴۸ + ۱) = ۸۳۹۴۷۵۴$$

ب

$$\frac{\partial J}{\partial W_{ij}} = (z - y) \times h$$

ج

$$\frac{\partial J}{\partial V_{ij}} = (z - y) \times W \times W \times x \times r'(Vx) \times r(Vx)$$

سوال ۴.

الف

خیر، غلط است.

Batch normalization نه تنها زمان آموزش را کاهش نمی‌دهد، بلکه ممکن است برای پردازش هر دوره (epoch) زمان بیشتری لازم باشد، زیرا یک عملیات اضافی را اعمال می‌کند: نرمال‌سازی ورودی‌های هر دسته (batch)

اما ویژگی مهم batch normalization این است که باعث می‌شود آموزش شبکه عصبی عمیق پایدارتر و موثرتر شود. این به دو دلیل است:

آن برداشتن مشکل انتقال خودکار (Internal Covariate Shift) کمک می‌کند، که این موضوع می‌تواند کمک کند تا شبکه به سرعت به یادگیری پرداخته و به یک نقطه بهینه برسد.

به طور ناخودآگاه تاثیری مشابه regularization دارد، که می‌تواند به جلوگیری از بیش‌برازش (overfitting) کمک کند.

بنابراین، اگرچه batch normalization ممکن است زمان پردازش هر دوره را افزایش دهد، اما به دلیل اینکه باعث می‌شود آموزش پایدارتر و موثرتر شود، ممکن است در کل، زمان کلی آموزش را کاهش دهد، زیرا کمتر دوره لازم است تا به نتایج مناسب برسیم.

به عبارتی دیگر، Batch Normalization ممکن است زمان اجرای هر epoch را افزایش دهد، اما تعداد کل epochs مورد نیاز برای رسیدن به یک خطای مشخص را کاهش می‌دهد.

ب

بله، این بخش درست است.

Batch Normalization (BN) در شبکه‌های عصبی عمیق، فرآیندی است که به طور کلی توزیع ورودی لایه‌های خاص را به‌صورت مستقل از بقیه لایه‌ها نرمال می‌کند. این فرآیند از تاثیر تغییرات در توزیع ورودی‌های هر لایه به دلیل تغییرات در لایه‌های قبلی (یک مشکل به نام Internal Covariate Shift) جلوگیری می‌کند. بر اساس این تعریف، BN بیشتر به تاثیر تغییرات در وزن‌های شبکه‌های عصبی عمیق روی توزیع ورودی‌های هر لایه مربوط می‌شود، نه به مقداردهی اولیه وزن‌ها.

مقداردهی اولیه وزن‌ها در شبکه‌های عصبی به منظور کنترل شروع فرآیند یادگیری و جلوگیری از مشکلاتی مانند مقادیر خیلی بزرگ یا خیلی کوچک برای گرادیان‌ها است. بنابراین، بسته به نوع مقداردهی اولیه و تابع فعال‌سازی که استفاده می‌کنید، ممکن است به مشکلاتی برخورد کنید.

پس، در حالی که Batch Normalization می‌تواند برخی از تاثیرات نامطلوب تغییرات وزن‌ها را محدود کند، اما این به معنی عدم اهمیت مقداردهی اولیه مناسب وزن‌ها نیست. یک مقداردهی اولیه مناسب همچنان یک بخش مهم در موفقیت یک شبکه عصبی عمیق است.

سوال ۵.

الف

یکی از استفاده‌های اصلی کانولوشن 1x1 در شبکه های عصبی پیچشی یا CNN ، تغییر تعداد کانال‌ها (depth) است. این موضوع در ارتباط با مدل‌های معماری شبکه‌های عمیق مانند GoogLeNet که معماری Inception را ارائه کرد، به‌ویژه مهم است. در این معماری، یک سری کانولوشن‌های 1x1 (که اغلب به عنوان «باتل‌نک» شناخته می‌شوند) مورد استفاده قرار گرفتند تا تعداد کانال‌های ورودی کاهش یابد قبل از اعمال کانولوشن‌های 3x3 یا 5x5

این کاهش در تعداد کانال‌ها باعث کاهش مصرف حافظه و افزایش سرعت آموزش می‌شود.

در مدل‌های معماری دیگر مانند ResNet نیز، کانولوشن‌های 1x1 برای تغییر تعداد کانال‌ها (و گاهی اوقات تغییر اندازه خروجی مکانی) استفاده می‌شوند.

به طور کلی، کانولوشن 1x1 در CNN می‌تواند به چندین شکل مفید باشد:

کاهش تعداد کانال‌ها: مانند چیزی که در بالا توضیح داده شد. افزایش تعداد کانال‌ها: اگر بخواهیم تعداد کانال‌ها را افزایش دهیم، کانولوشن 1x1 می‌تواند یک راه مناسب باشد.

استفاده به عنوان یک تابع غیرخطی: با استفاده از تابع فعال‌سازی غیرخطی بعد از کانولوشن 1x1 ، می‌توانیم مدل را پیچیده‌تر کنیم.

ب

وقتی شما می خواهید سایز ورودی و خروجی لایه کانولوشن برابر باشد، شما باید از "same padding" استفاده کنید. در این حالت، ورودی به گونه‌ای padding می شود تا ابعاد ورودی و خروجی برابر باشند.

مقدار padding وابسته به سایز کرنل کانولوشن است. اگر سایز کرنل K باشد و از گام ۱ (Stride 1) استفاده کنیم، آنگاه مقدار padding باید $\frac{k-۱}{۲}$ باشد. به عنوان مثال، اگر از یک کرنل 3x3 استفاده کنید، مقدار padding باید ۱ باشد تا ابعاد ورودی و خروجی برابر باشند. اگر از یک کرنل 5x5 استفاده کنید، padding باید ۲ باشد.

لطفا توجه داشته باشید که این قاعده برای گام ۱ صادق است. اگر از گام بزرگتر استفاده کنید، ممکن است نتوانید سایز ورودی و خروجی را برابر کنید.

ج

average pooling و max pooling چه تفاوت‌هایی دارند و استفاده از هر کدام در چه مواردی بهتر است؟

جواب: در جواب باید گفت که "Max Pooling" و "Average Pooling" دو روش کاهش سایزیتی در شبکه‌های عصبی پیچشی (CNN) هستند. این دو روش با گرفتن خروجی‌های کانولوشن از قبلی و کاهش ابعاد آنها به وسیله کاهش سایزتی، به استخراج ویژگی‌ها کمک می‌کنند. اما تفاوت‌های اصلی آن‌ها در نحوه عملکردشان و کاربرد آن‌هاست.

Max Pooling : این روش در هر پنجره، بزرگترین عدد (یعنی ویژگی با بیشترین حضور) را انتخاب می‌کند. به این ترتیب، Max Pooling می‌تواند ویژگی‌های برجسته را در نقشه‌های ویژگی حفظ کند، حتی اگر این ویژگی‌ها کوچک باشند. این ویژگی اغلب در شناسایی اشیا و تصاویر مفید است، زیرا ویژگی‌های برجسته مهمتر از میانگین ویژگی‌های دیگر هستند.

Average Pooling : این روش میانگین تمام اعداد در هر پنجره را محاسبه می‌کند. این باعث می‌شود که نقشه‌های ویژگی بیشتری از ویژگی‌های عمومی تصویر را نگه دارند و می‌تواند کمک کند تا شبکه در برابر تغییرات مکانی مقاومتر شود. این روش اغلب در مواردی استفاده می‌شود که نیاز به دید کلی از ویژگی‌های تصویر داریم، نه فقط ویژگی‌های برجسته.

این‌که کدام یک بهتر است بستگی به کاربرد خاص شما دارد. به طور کلی، Max Pooling در بسیاری از شبکه‌های CNN که برای تشخیص تصویر استفاده می‌شوند، محبوب‌تر است زیرا می‌تواند ویژگی‌های برجسته را حفظ کند. با این حال، Average Pooling می‌تواند در بعضی از موارد، مانند شناسایی حضور یا عدم حضور یک ویژگی کلی در تصویر، مفید باشد.

موفق باشید.