# Short Questions

## ۱.۱

* **استفاده از تکنیک‌های رگولاریزیشن**

تکنیک‌هایی نظیر L2 Regularization (رگرسیون Ridge) یا Dropout که به صورت رندم خروجی برخی لایه‌ها را نادیده می‌گیرد، برای جلوگیری از Overfitشدن مدل استفاده می‌شوند. این تکنیک‌ها در واقع اجازه می‌دهند خطای ترین، در ازای Generalization بهتر افزایش یابد. از آن‌جایی که این تکنیک‌ها هنگام تست مدل با دیتای Validation اعمال نمی‌شوند، ممکن است در نتیجه خطای Validationمان کم‌تر باشد. در واقع، در اینجا مدل‌مان را Over-Regularize کرده‌ایم.

برای حل این مشکل،‌ می‌توان پارامترهای رگولاریزیشن نظیر نرخ Drop Out را طوری تغییر داد که دقت روی دیتای ترین، بیش از اندازه فدای دقت روی دیتای Validation نشود.

* **توزیع متفاوت دیتای Validation و ساده‌تر بودن طبقه‌بندی آن**

ممکن است دیتای Validation، از یک توزیع آماری متفاوت از دیتای ترین گردآوری شده‌باشد که طبقه‌بندی را ساده‌تر می‌کند. برای مثال،‌ دیتای Validation در نتیجه توزیع متفاوت، ساده‌تر باشد و تنها با چند فیچر بتوان خروجی را تخمین زد. در این حالت نیز ممکن است خطای Validation کم‌تر از خطای ترین باشد.

برای حل این مشکل، باید مطمئن شد که هم دیتای ترین، و هم دیتای Validation، به اندازه کافی بزرگ هستند و از یک توزیع یکسان نمونه‌برداری شده‌اند. با روش‌های آماری، می‌توان شاخص‌هایی نظیر میانگین، میانه، واریانس و ... را میان فیچرهای مختلف دیتای ترین و Validation محاسبه کرده، و از متعادل بودن این دو دیتاست اطمینان حاصل کرد.

## ۱.۲

اگر لایه‌های یک شبکه عصبی عمیق، تماما از توابع فعال‌سازی خطی استفاده می‌کردند، این شبکه تنها قادر به تشخیص رابطه خطی میان ورودی و خروجی بود؛ به عبارت دیگر، هر لایه صرفا یک نگاشت خطی از ورودی به خروجی بود، و در نتیجه کل شبکه را می‌شد با یک Perceptron تک‌لایه خطی مدل کرد.

بنابراین، اگر رابطه میان ورودی و خروجی یک رابطه غیر خطی است، لازم است که لایه‌های شبکه عصبی عمیق نیز غیر خطی باشند تا بتوان این روابط را Capture کرد.

## ۱.۳

به نظر می‌رسد که متد انتخاب شده بهینه نباشد، و چند مشکل با آن وجود دارد؛ نخست آنکه مقادیر انتخاب‌شده برای نرخ یادگیری، به نظر سلیقه‌ای می‌آیند و رابطه خاصی (چه خطی چه غیرخطی) بین آن‌ها نیست. همچنین برخی بازه‌های نسبتا بزرگ با این مقادیر انتخابی، نادیده گرفته می‌شوند. (مثلا بازه بین ۰.۲۱ تا ۰.۸۴، و احتمالا بازه بین ۰.۰۱ تا ۰.۱۶).

جدای از مقادیر در نظر گرفته‌شده، خود روش یافتن بهترین مقدار برای نرخ یادگیری نیز احتمالا بهینه نیست؛ یک کار بهتر،‌ تعریف یک دنباله بزرگ‌تر خطی در بازه معین‌شده، و ترین‌کردن کوتاه مدل روی هر یک از مقادیر این رنج است. منظور از ترین‌کردن کوتاه، محدود کردن تعداد Epoch هاست. با تعریف مناسب بازه میان مقادیر مختلف نرخ یادگیری، می‌توان نرخ یادگیری را در برابر خطای مدل پلات کرد، و نقطه مینیمم را، به عنوان نقطه بهینه نرخ یادگیری انتخاب نمود.

## ۱.۴

*منحنی C، در ابتدا کاهش Loss زیادی داشته، اما به نظر مدل چندان خوب همگرا نمی‌شود. این نشان‌دهنده این است که نرخ یادگیری، بالاتر از حد ایده‌آل است.*

*منحنی B، از مقدار Loss بالا شروع می‌کند و از حدود Epoch ۱۲۰م، مقدار Loss به نظر در حال کاهش است و مدل در حال همگرا شدن است. چنین الگویی، نشان‌دهنده اینست که مقدار نرخ یادگیری، زیادی پایین است.*

*منحنی A، مقدار Loss بالایی دارد و این مقدار Loss بالا می‌ماند و حتی در آخرین ایتریشن‌ها، به نظر بیشتر نیز می‌شود. این نشان‌دهنده اینست که میزان نرخ یادگیری، بسیار بالاتر از حد بهینه است.*

*بنابراین، ترتیب منحنی‌ها، به ترتیب نرخ یادگیری از کوچک به بزرگ به صورت زیر است:*

*B < D < C < A*

*با این اوصاف، مقدار آلفای مربوط به هر منحنی نیز به این صورت است:*

*B: 4e1*

*D: 8e3*

*C: 3e4*

*A: 2e5*

## ۱.۵

گزینه سوم صحیح است.

توضیحات در مورد گزینه‌ها:

۱) تقریبا نادرست؛‌ تانژانت هایپربولیک نیز همانند سیگموید، با ورودی خیلی کوچک یا خیلی بزرگ، مستعد مشکل Vanishing Gradient است. البته لازم به ذکر است که tanh اندکی در این موضوع بهتر عمل می‌کند، اما همچنان جزو توابعی است که مستعد مشکل به شمار می‌روند.

۲) نادرست؛ مشکل Vanishing Gradient سبب می‌شود که گرادیان‌های تابع Loss، هنگام Backpropagation، کوچک‌تر و کوچک‌تر شده و به صفر میل کنند. این باعث می‌شود که وزن‌ها در لایه‌های ابتدایی شبکه، بسیار کندتر آپدیت شوند. بنابراین، لایه‌های ابتدایی شبکه، کندتر از لایه‌های در عمق آن یادگیری می‌کنند، که این گزاره عکس گزاره داده شده‌است.

۳) درست؛ گرادیان Leaky ReLU برای ورودی‌های مثبت، یک بوده و برای ورودی‌های منفی، یک مقدار مثبت کوچک؛ این عمل باعث می‌شود که این تابع، در کل کمتر از سیگموید مستعد مشکل Vanishing Gradient باشد.

## ۱.۶

(۱)‌ مینی‌بچ Gradient Descent پیچیدگی محاسباتی کم‌تری نسبت به Full Batch دارد؛ در Full Batch، در هر Epoch کل دیتای ترینینگ پروسس شده و گرادیان آن‌ها محاسبه می‌شود که بسیار هزینه‌بر تر است. علاوه بر این، در Full Batch احتمال بیشتری است که به Local Minima برسیم. علت اینست که مدل وزن‌های مدل با فرکانس بالاتری آپدیت شده و احتمال اینکه مدل از یک مینیای محلی عبور کند بیشتر است.

(۲) با سایز بچ ۱،‌ گرادیانت دیسنت تصادفی واریانس بالایی خواهد داشت و ممکن است شاهد نوسانات زیاد در قدم‌های برداشته‌شده و همگرایی مدل باشیم. در مقابل، گرادیانت دیسنت مینی‌بچ، همگرایی Smoothتری دارد و در آن واریانس قدم‌های برداشته‌شده کم‌تر می‌باشد.

(۳)‌ در گرادیانت دیسنت Vanilla، نرخ یادگیری، برای کل طول ترین‌شدن مدل یک نرخ ثابت است. این درحالی است که در پروسه ترین‌شدن، محتمل است نرخ‌یادگیری بهینه تغییر کند. با انتخاب یک نرخ یادگیری ثابت، ممکن است یادگیری پارامترها بسیار کند باشد، یا اینکه نرخ یادگیری آنقدر بالا باشد که مدل همگرا نشود. با الگوریتم Adam Optimization، نرخ یادگیری، به واسطه متوسط‌گیری نمایی متحرک از گرادیان‌ها، نرخ یادگیری در پروسه ترینینگ، مطابق وضع پیشروی ترینینگ، Adopt می‌شود.

# ‌Backpropagation

## **۲.۱**

K کلاس داریم، و در ضرب می‌شود که سطر دارد. بنابراین، دارای K سطر و ستون خواهد بود.

و نیز طبعا K سطر و یک ستون دارد.

*خروجی لایه پنهان:*

*نورون داریم و بنابراین، ماتریس خروجی لایه پنهان سطر خواهد داشت. از طرفی،‌ ابعاد ماتریس خروجی لایه پنهان، برابر با ابعاد خواهد بود. برای ابعاد ماتریس‌ها داریم:*

*[Da x ?] = [Da x Dx]\*[Dx x m] + [Da x 1]*

*واضح است که تعداد ستون‌ها m بوده و ماتریس خروجی لایه پنهان دارای ابعاد Da x m خواهد بود.*

## ۲.۲

## ۲.۳

## ۲.۴

برای حالت :

برای حالت :

## ۲.۵

## ۲.۶

## ۲.۷

لورم اپسیوم یک دسته‌خر فارسی است که.

## ۲.۸

لورم اپسیوم یک دسته‌خر فارسی است که.

# Two Layer Neural Network

## ۳.۱

لورم اپسیوم یک دسته‌خر فارسی است که.

## ۳.۲

لورم اپسیوم یک دسته‌خر فارسی است که.

## ۳.۳

لورم اپسیوم یک دسته‌خر فارسی است که.

## ۳.۴

لورم اپسیوم یک دسته‌خر فارسی است که.