یادگیری ماشین برای بیوانفورماتیک

تمرین سوم، بخش تئوری

جواد راضی (۴۰۱۲۰۴۳۵۴)

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 W_2 در a_1 ضرب می‌شود که D_a سطر دارد. بنابراین، W_2 دارای K سطر و D_a ستون خواهد بود.

و b_2 نیز طبعا K سطر و یک ستون دارد.

خروجی لایه پنهان:

D_aنورون داریم و بنابراین، ماتریس خروجی لایه پنهان D_a سطر خواهد داشت. از طرفی، ابعاد ماتریس خروجی لایه پنهان، برابر با ابعاد z_1 خواهد بود. برای ابعاد ماتریس‌ها داریم:

$[D_a \times ?] = [D_a \times D_x]*[D_x \times m] + [D_a \times 1]$

واضح است که تعداد ستون‌ها m بوده و ماتریس خروجی لایه پنهان دارای ابعاد $D_a \times m$ خواهد بود.

۲.۲

$$\frac{\partial \hat{y}_k}{\partial z_k^{[2]}} = \frac{\partial SoftMax(z_k^{[2]})}{\partial z_k^{[2]}} = SoftMax\left(z_k^{[2]}\right) * \left(1 - SoftMax\left(z_k^{[2]}\right)\right) = \hat{y}_k * (1 - \hat{y}_k)$$

۲.۳

$$\frac{\partial \hat{y}_k}{\partial z_i^{[2]}} = \frac{\partial}{\partial z_i^{[2]}}\left(\frac{e^{z_k^{[2]}}}{Z}\right) = \frac{\frac{\partial e^{z_k^{[2]}}}{\partial z_i^{[2]}} * Z - \frac{\partial Z}{\partial z_i^{[2]}} * e^{z_k^{[2]}}}{Z^2}$$

$$= \frac{0 * Z - \frac{\partial\left(e^{z_i^{[2]}} + \sum_{j \neq i} e^{z_j^{[2]}}\right)}{\partial z_i^{[2]}} * e^{z_k^{[2]}}}{Z^2} = \frac{-\left(\frac{\partial e^{z_i^{[2]}}}{\partial z_i^{[2]}}\right) * e^{z_k^{[2]}}}{Z^2}$$

$$= \frac{-e^{z_i^{[2]}} * e^{z_k^{[2]}}}{Z^2} = -SoftMax\left(z_k^{[2]}\right) * SoftMax\left(z_i^{[2]}\right) = -y_k * y_i$$

۲.۴

برای حالت $i = k$:

$$\frac{\partial L}{\partial z_i^{[2]}} = \frac{\partial L}{\partial \hat{y}_i} * \frac{\partial \hat{y}_i}{\partial z_i^{[2]}} = \left(\frac{\partial}{\partial \hat{y}_i}\left(-\sum_{k=1}^{K} y_i \log \partial \hat{y}_i\right)\right) * \underbrace{(\hat{y}_i * (1 - \hat{y}_i))}_{From\ 2.2}$$

$$= -\frac{y_i}{\hat{y}_k} * \hat{y}_i * (1 - \hat{y}_i) = -\frac{1}{\hat{y}_k} * \hat{y}_i * (1 - \hat{y}_i) = -(1 - \hat{y}_i)$$

برای حالت $i \mathrel{!}= k$:

$$\frac{\partial L}{\partial z_i^{[2]}} = \frac{\partial L}{\partial \hat{y}_i} * \frac{\partial \hat{y}_i}{\partial z_i^{[2]}} = \left(\frac{\partial}{\partial \hat{y}_i}\left(-\sum_{k=1}^{K} y_i \log \partial \hat{y}_i\right)\right) * \underbrace{\left(\hat{y}_i * (1 - \hat{y}_i)\right)}_{From\ 2.2}$$

$$= -\frac{y_i}{\hat{y}_k} * \hat{y}_i * (1 - \hat{y}_i) = -\frac{0}{\hat{y}_k} * \hat{y}_i * (1 - \hat{y}_i) = 0$$

۲.۵

$$\delta_1 = \frac{\partial z^{[2]}}{\partial a^{[1]}} = \frac{\partial}{\partial a^{[1]}}\left(W^{[2]} a^{[1]} + b\right) = W^{[2]}$$

۲.۶

$$\delta_2 = \frac{\partial a^{[1]}}{\partial z^{[1]}} = \frac{\partial}{\partial z^{[1]}}\left(LeakyReLU(z^{[1]}, \alpha = 0.01)\right) = \begin{cases} 0.01 * z^{[1]}, & z^{[1]} < 0 \\ 0, & z^{[1]} \geq 0 \end{cases}$$

۲.۷

$$\frac{\partial L}{\partial W^{[1]}} = \frac{\partial L}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial a^{[1]}} \cdot \frac{\partial a^{[1]}}{\partial z^{[1]}} \cdot \frac{\partial z^{[1]}}{\partial W^{[1]}} = \delta_0 * \delta_1 * \delta_2 * X$$

$$\frac{\partial L}{\partial b^{[1]}} = \frac{\partial L}{\partial z^{[2]}} \cdot \frac{\partial z^{[2]}}{\partial a^{[1]}} \cdot \frac{\partial a^{[1]}}{\partial z^{[1]}} \cdot \frac{\partial z^{[1]}}{\partial b^{[1]}} = \delta_0 * \delta_1 * \delta_2 * 1$$

۲.۸

مشکل ناپایداری محاسباتی، هنگامی پیش می‌آید که z_i خیلی بزرگ یا خیلی کوچک باشد. در این شرایط، ممکن است مقدار تابع نمایی e^{z_i} را نتوان با آسانی محاسبه و ذخیره کرد. چرا که در حالت اول ممکن است مقدار این تابع بسیار بزرگ باشد و با overflow مواجه شویم، در حالت دوم نیز ممکن است مقدار تابع به صفر میل کند و با underflow مواجه شویم.

فرمول جدید، مقدار بیشینه z_i ها را از هر کدام کم می‌کند. با این کار، مطمئنیم در توان تابع نمایی، هیچ مقدار مثبتی نداریم، پس overflow نخواهیم داشت. مشکل underflow همچنان بالقوه می‌تواند پیش بی‌آید، اما احتمال رخداد آن با فرمول تغییریافته بسیار کم‌تر می‌شود. علت اینست

مطمئنیم حداقل یک مقدار صفر داریم (z_i بیشینه که از خود کم شده). بنابراین، در مخرج کسر که یک جمع است، یک مقدار $e^{z_i} = e^0 = 1$ خواهیم داشت. پس مطمئنیم که مخرج کسر (که جمع عبارات مثبت است)، به صفر میل نمی‌کند و vanish نمی‌شود. البته باز هم در مخرج ممکن است مولفه‌هایی دچار underflow شوند، اما چون اثر ۱ بیشتر است، underflow در یک سری از مولفه‌های مخرج قابل پذیرش است.

Two Layer Neural Network

۳.۱

برای محاسبه مقادیر خروجی، باید لایه به لایه، ورودی هر نورون را که جمع وزن‌دار خروجی لایه قبل است محاسبه کرده، بایاس را اعمال کرده و این مقدار را به تابع فعال‌ساز (ReLU) می‌دهیم.

اکنون گام‌به‌گام روند محاسبه خروجی را شرح می‌دهیم.

یونیت h1:

$$h_1 = W^{h_1}i + b_1 = \ i_1w_{11} + \ i_2w_{21} + b_1 = 2.0 * 1 + \ -1.0 * 0.5 + 0.5 = 2.0$$

یونیت h2:

$$h_2 = W^{h_2}i + b_2 = \ i_1w_{12} + \ i_2w_{22} + b_2 = 2.0 * -0.5 + (-1.0 * -1.0) - 0.5 = -0.5$$

یونیت h3:

$$h_3 = ReLU(h_1) = ReLU(2.0) = 2.0$$

یونیت h4:

$$h_4 = ReLU(h_2) = ReLU(-0.5) = 0.0$$

خروجی o1:

$$o_1 = h_3w_{31} + h_4w_{41} + b_3 = 1.0 + 0 - 1.0 = 0.0$$

خروجی o2:

$$o_2 = h_3w_{32} + h_4w_{42} + b_4 = -2.0 + 0.0 + 0.5 = \ -1.5$$

۳.۲

$$MSE = \frac{1}{2}((o_1 - t_1)^2 + (o_2 - t_2)^2) = \frac{1}{2}(1 + 4) = 2.5$$

۳.۳

$$\begin{aligned}
\frac{\partial MSE}{\partial W_{21}} = & \frac{\partial MSE}{\partial o_1} . \frac{\partial o_1}{\partial h_3} . \frac{\partial h_3}{\partial h_1} . \frac{\partial h_1}{\partial W_{21}} + \frac{\partial MSE}{\partial o_2} . \frac{\partial o_2}{\partial h_3} . \frac{\partial h_3}{\partial h_1} . \frac{\partial h_1}{\partial W_{21}} \\
&= \frac{1}{2}[2 * (o_1 - t_1) + 0].[W_{31}].\left[\frac{\partial}{\partial h_1}\left(ReLU(h_1)\right)\right].[i_2] \\
&+ \frac{1}{2}[2 * (o_2 - t_2) + 0].[W_{32}].\left[\frac{\partial}{\partial h_1}\left(ReLU(h_1)\right)\right].[i_2] \\
&= (o_1 - t_1) * W_{31} * u_{step}(h_1 = 2.0) * i_2 + (o_2 - t_2) * W_{32} \\
&* u_{step}(h_1 = 2.0) * i_2 \\
&= (0.0 - 1.0) * 0.5 * 1.0 * (-1.0) + (-1.5 - 0.5) * -1.0 * 1.0 * -1.0 \\
&= 0.5 - 2 = -1.5
\end{aligned}$$

اکنون با بدست آوردن مشتق جزئی تابع Loss نسبت به W_{21}، مقدار این وزن را با نرخ یادگیری آپدیت می‌کنیم:

$$Learning\ Rate = \alpha = 0.1$$

$$W_{21}^{new} = W_{21} - \alpha * \frac{\partial MSE}{\partial W_{21}} = 0.5 - (0.1 * -1.5) = 0.5 + 0.15 = 0.65$$