یادگیری عمیق

دکتر فاطمی زاده

مهندسی برق

برنا خدابنده ۴۰۰۱۰۹۸۹۸

تمرین ۲

۲۷ آبان ۱۴۰۲

یادگیری عمیق

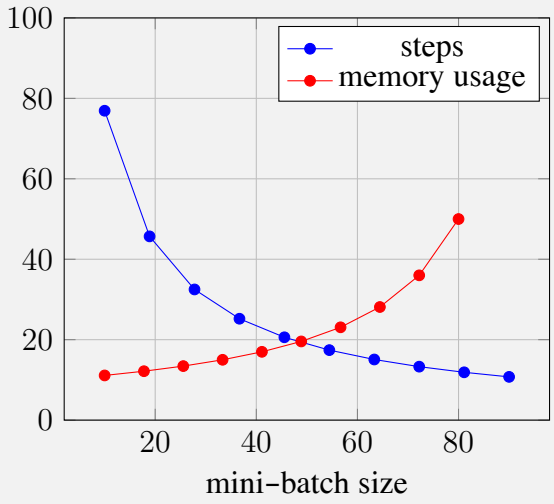
تمرین ۲

برنا خدابنده ۴۰۰۱۰۹۸۹۸

سوالات نظری

سوال ۱

۱. نموداری رسم کنید که محور افقی آن تعداد داده در batch-mini و محور عمودی آن تعداد گام لازم برای همگرایی توسط الگوریتم SGD جهت رسیدن به مقدار از پیش تعیین شده ای از خطای آموزش باشد. بخش آغازین نمودار (بسته های کوچک داده) و بخش پایانی نمودار (بسته های بزرگ داده) را با ذکر دلیل توجیه نمایید.



به طور کلی، افزایش اندازه بسته mini-batch می‌تواند به سریع تر به همگرایی منجر شود، اما همچنین به افزایش مصرف حافظه محاسباتی نیاز دارد. این تجزیه و تحلیل به معنای این است که انتخاب اندازه مناسب برای mini-batch در SGD نیازمند تعادل بین سرعت همگرایی و میزان حافظه مورد نیاز است.

$$\theta_{i+1} = \theta_i - \eta \nabla J(\theta_i, x^{(i)}, y^{(i)})$$

افزایش سایز mini-batch منجر به سخت تر شدن محاسبه $\nabla J(\theta_i, x^{(i)}, y^{(i)})$ شده و همچنین مقدار θ_i بیشتری باید آپدیت شوند، در نتیجه حجم محاسبه بیشتر شده، ولی گرادیان محاسبه شده پرمعنا تر است.
بررسی بیشتر در: [۱]

۲. آیا می توان گفت که در لایه Batch Normalization و در هنگام آموزش، مقداری نویز به توابع فعالیت لایه های مخفی تزریق می شود؟ چرا؟

به این موضوع در مقاله ی [۲] به طور عمیق پرداخته شده است.
به طور خلاصه ولی، انتخاب batch در روش Batch Normalization یک عمل تصادفی است، صورت مسئله روش BN به صورت زیر است:

$$y = g(\hat{h}), \quad \hat{h} = \gamma \bar{h} + \beta, \quad \bar{h} = \frac{h - \mu_{\mathcal{B}}}{\sigma_{\mathcal{B}}}$$

حال چون انتخاب $\mathcal{B}$ خود یک کار تصادفی است، خود $\mu_{\mathcal{B}}$ و $\sigma_{\mathcal{B}}$ متغیرهای تصادفی خواهند بود.
اگر سایز batch را بزرگ بگیریم، میتوان طبق قضیه حد مرکزی نشان داد که(اثبات دقیق تر در [۳]):

$$\mu_{\mathcal{B}} \sim \mathcal{N}(\mu_{\mathcal{P}}, \frac{\sigma_{\mathcal{P}}}{M}), \quad \sigma_{\mathcal{B}} \sim \mathcal{N}(\sigma_{\mathcal{P}}, \frac{\rho + 2}{4M}), \quad \rho : \text{kurtosis}, \quad \mathcal{P} : \text{population}$$

حال به گونه ای میتوان دید که $y = g(\hat{h})$ نیز یک متغیر تصادفی خواهد شد، باز اگر M بزرگ باشد، میتوان به طور تقریبی از حد مرکزی دید که:

$$y \sim \mathcal{N}(y_{\mathcal{P}}, \sigma) = y_{\mathcal{P}} + \mathcal{N}(0, \sigma), \quad y_{\mathcal{P}} = g\left(\gamma \frac{h - \mu_{\mathcal{P}}}{\sigma_{\mathcal{P}}} + \beta\right)$$

پس میتوان چنین تعبیری کرد که در واقع این عملیات BN مانند نویزی کردن توابع فعالیت است، جزئیات محاسبه σ در اینجا آورده نشده است.

$$g(\hat{h}) = g(\hat{h_{\mathcal{P}}}) + \delta, \quad \delta \sim \mathcal{N}(0, \sigma)$$

۳. شبکه ی عصبی Fully Connected ای را درنظر بگیرید که تمام توابع فعالیت آن سیگموید باشد و وزن های اولیه آن مقادیر مثبت بزرگ باشد. آیا این شبکه مناسبی برای طبقه بندی می باشد؟ چرا؟

یک شبکه عصبی Fully Connected با تمام توابع فعالیت سیگموید و وزن‌های اولیه مثبت بزرگ، عموماً مناسب برای تمام مسائل طبقه‌بندی نیست. این مسئله به عوامل زیر بستگی دارد:

- پیچیدگی توابع هدف: برای مسائل با توابع هدف پیچیده و غیرخطی، توانایی یک شبکه با توابع فعالیت سیگموید محدودتر است و نیاز به توانایی‌های شبکه‌های عمیق‌تر مانند شبکه‌های عصبی با لایه‌های ReLU دارد.
- وزن‌های اولیه بزرگ ممکن است باعث مشکلاتی مانند سریع برخورد به واحدهای اشباع (saturation) در توابع سیگموید شوند و منجر به کندی در آموزش یا برخورد به مشکل گرادیان نزولی شود.
- نوع داده‌ها نیز تأثیر دارد. برای مسائل مانند تشخیص تصاویر، شبکه‌های کانولوشنال با ویژگی‌های مربوط به تصاویر بهتر عمل می‌کنند.
- اگر تعداد ورودی‌ها بسیار بزرگ باشد، این نوع شبکه ممکن است به مشکلات مصرف

به طور خلاصه، این شبکه ممکن است برای مسائل ساده و کوچک مناسب باشد، اما برای بسیاری از مسائل پیچیده و مهم، نیاز به شبکه‌های عمیق‌تر و تنوع‌تر با توابع فعالیت متنوع‌تر داریم. همچنین وزن های اولیه بزرگ در توابع فعالیت سیگموید منجر به saturation و کم شدن گرادیان ها و vanishing gradient شده پس این نوع مقدار دهی نیز کار ما را سخت تر میکند.

۴. شبکه ی عصبی Fully Connected ای با ۵ لایه مخفی، که در هر کدام از لایه ها، ۱۰ نورون وجود دارد را در نظر بگیرید. ورودی این شبکه ۲۰ بعدی و خروجی آن اسکالر می باشد. تعداد کل پارامتر های قابل آموزش را در این شبکه محاسبه کنید.

$$\begin{aligned}\text{Total Trainable Parameters} = (I_0 \times H_1 + H_1) + (\overbrace{H_1 \times H_2}^{\text{Weights}} + \overbrace{H_2}^{\text{biases}}) + (H_2 \times H_3 + H_3) \\ + (H_3 \times H_4 + H_4) + (H_4 \times H_5 + H_4) + (H_5 \times O + O)\end{aligned}$$

پس برای این مسئله داریم که:

$$\begin{aligned}\text{Total Trainable Parameters} = (20 \times 10 + 10) + (10 \times 10 + 10) + (10 \times 10 + 10) \\ + (10 \times 10 + 10) + (10 \times 10 + 10) + (10 \times 1 + 1) = 661\end{aligned}$$

۵. یک مسئله طبقه بندی باینری می تواند با دو روش زیر حل شود:

روش اول: Logistic regression ساده (یک نورون)

خروجی: $\hat{y} = \sigma(W_l x + b_l)$

اگر $\hat{y} \leq 0.5$ کلاس صفر در غیر این صورت کلاس یک طبقه بندی می شود.

روش ذوم: Softmax regression ساده (دو نورون)

خروجی: $\hat{y} = Softmax(W_s x + b_s) = [\hat{y}_1, \hat{y}_2]^T$

اگر $\hat{y}_1 \geq \hat{y}_2$ کلاس صفر در غیر این صورت کلاس یک طبقه بندی می شود.

روش دوم دو برابر روش اول پارامتر دارد. آیا می توان گفت که روش دوم مدل های پیچیده تری نسبت به روش اول یاد می گیرد؟

اگر بله، پارامترهای (W_s, b_s) تابعی که روش دوم می تواند آن را مدل کند مثال بزنید در غیر این صورت نشان دهید که (W_s, b_s) همیشه میتواند برحسب (W_l, b_l) نوشته شود.

مدل‌های Logistic Regression و Softmax Regression به عنوان دو روش مختلف در مسائل طبقه‌بندی باینری و چند دسته‌ای به کار می‌روند. اما اگر به دقت به معادلات داخلی این دو مدل نگاه کنیم، متوجه می‌شویم که این دو مدل در واقع دقیقاً یکی هستند و تنها با تغییراتی در وزن‌ها و بایاس‌ها به یکدیگر تبدیل می‌شوند.
در Softmax Regression، ما داریم:

$$\hat{y}_1 = \frac{e^{(W_{s1}x+b_{s1})}}{e^{(W_{s1}x+b_{s1})} + e^{(W_{s2}x+b_{s2})}} = \frac{1}{1 + e^{[(W_{s2}-W_{s1})x+b_{s2}-b_{s1}]}}$$

$$\hat{y}_2 = \frac{e^{(W_{s2}x+b_{s2})}}{e^{(W_{s1}x+b_{s1})} + e^{(W_{s2}x+b_{s2})}} = 1 - \hat{y}_1 \Rightarrow \hat{y}_1 \geq \hat{y}_2 \sim \hat{y}_1 \geq 0.5$$

میتوان دید مدل softmax معادل یک Logistic با پارامتر های زیر است.

$$\hat{y} = \sigma(W_l x + b_l), \quad W_l = W_{s2} - W_{s1}, \quad b_l = b_{s2} - b_{s1}$$

با این تغییرات در وزن‌ها و بایاس‌ها، دو مدل به یکدیگر تبدیل می‌شوند. به عبارت دیگر، توابع Softmax و Logistic در این حالت به یکدیگر معادل می‌شوند و هر دو مدل به یک دسته‌ی مدل تبدیل می‌شوند. این نشان می‌دهد که این دو مدل اساساً یکی هستند و تفاوت اصلی بین آنها تعداد پارامترهای مدل است.
در نتیجه مدل softmax یک مدل پیچیده تر نیست، زیرا هر تابعی که میتواند مدل کند را تابع سیگموید یا Logistic هم میتواند آن را مدل کند.

سوال ۲

می دانیم حتی یک شبکه عصبی یک لایه نیز می تواند طبقه بندی ارقام را با دقت خوبی انجام دهد. راه های متعددی برای ارتقای دقت مدل وجود دارد.این مقاله روشی ساده برای ارتقای عملکرد مدل، بدون تغییر در ساختار آن پیشنهاد می دهد. این پیشنهاد آموزش چند مدل مشابه است. مقاله را بخوانید و به سوالات زیر پاسخ دهید:

۱. کمیته چیست و چطور به بهبود عملکرد مدل کمک میکند؟

به طور خلاصه، کمیته به یک مجموعه از چندین طبقه‌بند مانند شبکه‌های عصبی اشاره دارد. آموزش این طبقه‌بندها روی نسخه‌های مختلفی از داده منجر به ایجاد مدل‌هایی می‌شود که خطاهای مختلفی دارند. ترکیب خروجی‌های آن‌ها از این تنوع بهره می‌برد؛ هر مدل اطلاعات تکمیلی ارائه می‌دهد که عملکرد کلی را بهبود می‌بخشد. کلید اینجا این است که خطاها به گونه‌ای تا حد امکان از یکدیگر مستقل باشند. این امر به کمیته اجازه می‌دهد خطاهای انفرادی مدل‌ها را اصلاح کند.

$$y^k_{com}(x) = \sum_{n=1}^{N} w_{nk} y_{nk}(x)$$

که در اینجا هر کدام از $y_{nk}(x)$ ها یک طبقه بند است، در کمیته ای با خطا های مستقل، خطای کل کمینه میشود.

۲. پیش پردازش انجام شده در مقاله را شرح دهید. چگونه این پیش پردازش از وابستگی زیاد خطای مدل ها جلوگیری می کند؟

تکنیک‌های کلیدی پیش‌پردازش داده که در این مقاله برای دی‌کرله کردن خطاها در میان مدل‌ها استفاده شده‌اند عبارتند از:

- **نرمالیزه کردن ابعاد اعداد:** آموزش مدل‌های مختلف بر روی داده‌هایی با bounding-box به عرض متفاوت از ۸ تا ۲۰ پیکسل، تنوعی در نسبت ابعاد وجود دارد. این باعث می‌شود هر مدل بر اساس ویژگی‌های مختلف مرتبط با نسبت ابعاد تکمیلی اعتماد کند، که منجر به خطاهای غیرهمبسته می‌شود.
- **رفع انحنای ارقام:** رفع انحنا باعث می‌شود که اصلی‌ترین مولفه عمودی شود. این باعث کاهش انحراف در انحنا در میان نمونه‌ها می‌شود. مدل‌هایی که بر روی داده‌های رفع انحنا آموزش داده شده‌اند، خطاهای متفاوتی نسبت به مدل‌های آموزش داده شده بر روی داده‌های انحراف دارند.
- **انحراف الاستیک:** اعمال انحراف‌های مختلف خطی و انعطافی در طول آموزش مدل‌ها را نسبت به تغییرات مقاوم می‌کند. استفاده از پارامترهای مختلف انحراف برای هر مدل تنوع در داده‌های آموزش را افزایش می‌دهد و منجر به خطاهای غیرهمبسته می‌شود.
- **مجموعه‌های آموزش مختلف:** برخی از مدل‌ها بر روی زیرمجموعه‌های تصادفی مختلفی از داده‌های آموزش آموزش می‌بینند. این کار باعث دی‌کرله کردن خطاها بین مدل‌ها می‌شود.

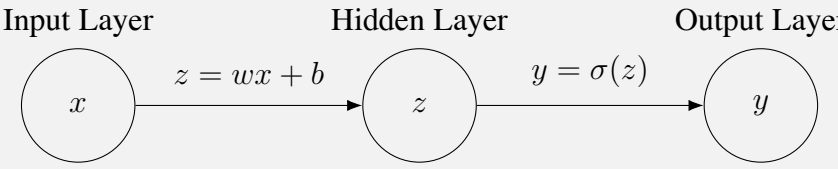
به طور خلاصه، با تغییراتی که هر مدل از طریق پیش‌پردازش و انحراف‌ها مشاهده می‌کند، آنها بر تمرکز بر روی یادگیری ویژگی‌ها و تثبیت‌های مختلف تمرکز می‌کنند. این تنوع در چیزهایی که یاد می‌گیرند منجر به خطاهای غیرهمبسته در مثال‌های تست می‌شود. ترکیب این مدل‌های متنوع باعث پوشش خطاهای یکدیگر می‌شود و عملکرد کلی را بهبود می‌بخشد.

سوال ۳

۱. یک شبکه ی عصبی با ورودی x را در نظر بگیرید. برای بدست آوردن خروجی محاسبات زیر بر روی x انجام می شود.

$$z = wx + b$$
$$y = \sigma(z)$$
$$L = \frac{1}{2}(y - t)^2$$
$$R = \frac{1}{2}w^2$$
$$L_{reg} = L + \lambda R$$

گراف محاسباتی این مسئله را رسم کنید و مشتقات L_{reg} را نسبت به همه متغیرها بدست آورید.



خروجی y باید به مقدار واقعی t نزدیک باشد، در نتیجه تابع هدف را به صورت زیر تعریف کرده ایم:

$$\mathcal{L}(x; w, b) = \frac{1}{2}(y(x; w, b) - t)^2 + \frac{\lambda}{2}w^2$$

شکل ۱: گراف محاسباتی

$$\frac{\partial L_{reg}}{\partial w} = \lambda w + \frac{\partial L}{\partial w} = \lambda w + \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial w} = \lambda w + x(y - t)\sigma'(z)$$
$$\frac{\partial L_{reg}}{\partial b} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial b} = (y - t)\sigma'(z)$$
$$\frac{\partial L_{reg}}{\partial x} = \frac{\partial L}{\partial x} = \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial x} = (y - t) \cdot \sigma'(z) \cdot w$$
$$\frac{\partial L_{reg}}{\partial t} = \frac{\partial L}{\partial t} = -(y - t)$$
$$\frac{\partial L_{reg}}{\partial \lambda} = R = \frac{1}{2}w^2$$
$$\sigma'(z) = \frac{\mathrm{d}\sigma(z)}{\mathrm{d}z} = \sigma(z)(1 - \sigma(z))$$

که البته برای مسئله ما، تنها $\frac{\partial L_{reg}}{\partial w}$ و $\frac{\partial L_{reg}}{\partial b}$ اهمیت دارند، زیرا آنها پارامتر هایی اند که بهینه سازی میکنیم، باقی مشتقات ممکن است برای تحلیل های متفاوتی مورد استفاده قرار بگیرند.
روند بهینه سازی به شکل زیر است:

$$w_{t+1} = w_t - \alpha \frac{\partial L_{reg}}{\partial w}$$
$$b_{t+1} = b_t - \alpha \frac{\partial L_{reg}}{\partial b}$$

۲. پارامتر های یک شبکه عصبی در ابتدا به صورت تصادفی و با مقادیر کوچک مقداردهی می شوند. توضیح دهید در صورت عدم رعایت این دو ویژگی در مقداردهی چه مشکلاتی بروز پیدا می کند.

مقدار دهی بزرگ پارامتر ها ۲ مشکل همراه با خود دارد، تابع فعالیت هایی مانند تابع سیگموید در محاسبات استفاده کرده ایم، بزرگ بودن وزن ها موجع به اشباع شدن این تابع فعالیت و از بین رفتن گرادیان آن در نتیجه یادگیری کند میشود، همچنین بزرگ بودن این وزن ها با توجه به جمله های رگولارازیسیون ممکن است موجب ناپایداری گرادیان ها و مشکل در میل کردن بشود.

۳. وزن های شبکه ی عصبی بدست آمده در قسمت اول را با مقادیر تصادفی دلخواه مقداردهی کنید و برای یک ورودی دلخواه، با توجه به مشتقاتی که در قسمت اول بدست آوردید، با اعمال بهینه سازی گرادیان کاهشی برای یک ایپاک با نرخ یادگیری ۰/۱ ، وزن های شبکه را آپدیت کنید.

$$
\begin{aligned}
\text{let: } & \lambda = 0.1, \quad \alpha = 0.1 \\
t = 0: \quad & w_0 = 0.1, \quad b_0 = 0.1 \\
y = \sigma(wx+b) &\Rightarrow y_0 = \sigma(0.1x - 0.1), \quad \sigma'(z_0) = y_0(1-y_0) \\
\frac{\partial L_{reg}}{\partial w} &= \lambda w + x(y-t)\sigma'(z) \Rightarrow 0.1 \cdot 0.1 + x(y_0 - t)y_0(1-y_0) \\
\frac{\partial L_{reg}}{\partial b} &= (y-t)\sigma'(z) = (y_0 - t)y_0(1-y_0) \\
&\Rightarrow \begin{cases} w_1 = w_0 - 0.1\frac{\partial L_{reg}}{\partial w} & = 0.1 \cdot (0.99) + xy_0(1-y_0)(y_0-t) \\ b_1 = b_0 - 0.1\frac{\partial L_{reg}}{\partial b} & = 0.1 - y_0(1-y_0)(y_0-t) \end{cases} \\
y_1 &= \sigma(w_1 x + b_1) \\
L'_{reg} &= \frac{(y_1-t)^2}{2} + \frac{\left(\frac{y_0\, x\, (y_0-t)\, (y_0-1)}{10} + \frac{99}{1000}\right)^2}{20}
\end{aligned}
$$

سوال ۴

الگوریتم آدام برای آموزش وزن های یک شبکه عصبی به صورت تکراری گام های زیر را اجرا میکند:

$$
\begin{aligned}
g_t &\leftarrow \nabla_\theta f_t(\theta_{t-1}) \\
m_t &\leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t \\
v_t &\leftarrow \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \\
\hat{m}_t &\leftarrow \frac{m_t}{1-\beta_1^t} \\
\hat{v}_t &\leftarrow \frac{v_t}{1-\beta_2^t} \\
\theta_t &\leftarrow \theta_{t-1} - \frac{\alpha \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}
\end{aligned}
$$

۱. الگوریتم بالا را خط به خط توضیح دهید.

- $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$: محسابه گرادیان و ذخیره آن در g_t، که گرادیان در زمان t است.
- $m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t$: محاسبه m_t که بردار سرعت ما در زمان t است، این سرعت رو با استفاده از تکانه(momemntum) محاسبه میکنیم، و ضریب تکانه را β_1 در نظر گرفته ایم. اینگونه که برای آپدیت کردن سرعت، سرعت لحظه قبل را نیز در نظر میگیریم، بطوری این میانگین نمایی گرادیان است.(exponential smoothing)
- $v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2) g_t^2$: در اینجا تخمین توان ۲ تکانه خود را آپدیت میکنیم، اینجا نیز مانند مرحله قبل میانگین نمایی را استفاده میکنیم و مقدار قبلی را نیز در نظر میگیریم.
- $\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t}$: اینجا بایاس به صفر را در تکانه را حذف میکنیم.
- $\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t}$: اینجا بایاس به صفر را در تکانه دوم را حذف میکنیم.
- $\theta_t \leftarrow \theta_{t-1} - \frac{\alpha \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$ در اینجا هم استپ نهایی و هدف اپتیمایزیشن، یعنی آپدیت کردن پارامتر ها طبق تکانه است. α در اینجا نرخ یادگیری است، با محاسبه $\frac{\hat{m}_t}{\sqrt{\hat{v}_t}}$ در واقع گرادیان نورمالیزه را حساب میکنیم تا منجر به ناپایداری نشود، اضافه کردن ϵ به مخرج صرفا برای جلوگیری از زیاد شدن سرعت در صورت کوچک بودن $\sqrt{\hat{v}_t}$ می باشد.

۲. نشان دهید چرا مقادیر m_t به سمت صفر بایاس دارند؟ چرا مقدار $\hat{m}_t$ که به شکل $\frac{m_t}{1-\beta_1^t}$ محاسبه میشود (در t های با مقادیر کمتر) با این مشکل رو به رو نمیشود؟
(توجه کنید که مقدار اولیه $m_0 = 0$ است.)

در واقع ما در محاسبه m_t(تحلیل مشابه برای v_t) هدف بر تخمین زدن گرادیان واقعی را داریم، حال ما در تخمین خود با یک moving average، وقتی با $m_0 = 0$ شروع میکنیم واضح است که تخمین ما در زمان های اولیه بسیار به سمت این m_0 بایاس شده است.
اینگونه بایاس را درست میکنیم.

$$
\mathbb{E}[m_t] = \mathbb{E}\left[(1-\beta_1)\sum_{i=1}^{t}\beta_1^{t-i} g_i\right] = \mathbb{E}[[]\, g_t] \cdot (1-\beta_1)\sum_{i=1}^{t}\beta_1^{t-i} + \zeta = \mathbb{E}[g_t]\,(1-\beta_2^t) + \zeta
$$

حال میتوان دید که اگر بر $\frac{1}{1-\beta_1^t}$ تقسیم کنیم، آنگاه $\hat{m}_t$ تخمینی بر گرادیان واقعی میزند.
شهود در زمان $t = 1$:

$$m_1 = \beta m_0 + (1-\beta) g_t \longrightarrow \hat{m}_1 = \frac{m_1 - \beta m_0}{1-\beta} = \frac{m_1}{1-\beta}$$

سوال ۵

تصور کنید که تابع هدف یک مدل یادگیری ماشین به صورت $w^T H w$ باشد که اگر از تجزیه مقادیر ویژه استفاده کنیم خواهیم داشت:

$$H = Q\lambda Q^T$$

۱. اگر از روش گرادیان کاهشی با طول گام ϵ استفاده کنیم، فرمول یادگیری ضرایب به چه صورت است؟

کافیست گرادیان را محاسبه کنیم و سپس از رابطه $w_{t+1} = w_t - \alpha \nabla_t \mathcal{L}$ استفاده کنیم.

$$H^T = (Q\lambda Q^T)^T = Q\lambda^T Q^T = Q\lambda Q^T = H$$
$$\frac{\partial \mathcal{L}}{\partial w} = \frac{\partial w^T H w}{\partial w} = (H + H^T)w = 2Hw$$
$$w_{t+1} = w_t - \alpha(2Hw_t) \Rightarrow w_{t+1} = (I - 2\alpha H)w_t = Q(I - 2\alpha\lambda)Q^T w_t$$

$$\boxed{w_{t+1} = Q(I - 2\alpha\lambda)Q^T w_t}$$

۲. با شروع از حالت اولیه w_0 ضرایب در گام t به چه صورت خواهد بود؟

$$w_t = \left[Q(I - 2\alpha\lambda)Q^T\right]^t w_0 = \overbrace{Q(I - 2\alpha\lambda)Q^T \dots Q(I - 2\alpha\lambda)Q^T}^{t} w_0$$
$$= Q \underbrace{(I - 2\alpha\lambda)^t}_{S} Q^T w_0 = Q \begin{bmatrix} (1-2\alpha\lambda_1)^t & 0 & \dots & 0 \\ 0 & (1-2\alpha\lambda_2)^t & \dots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & \dots & & (1-2\alpha\lambda_n)^t \end{bmatrix} Q^T w_0$$

$$\boxed{w_t = QSQ^T w_0}$$

۳. تحت چه شرایطی این الگوریتم همگرا می شود؟

برای همگرایی، نیاز است ماتریس S همگرا شود، که نیاز است تمامی المان های روی قطر همگرا شوند، برای این منظور لازم است که:

$$\forall i \in \{1, \dots, n\} : |1 - 2\alpha\lambda_i| \leq 1 \Rightarrow -1 \leq 1 - 2\alpha\lambda_i \leq 1 \Rightarrow \begin{cases} \lambda_i \geq 0 \\ \alpha\lambda_i \leq 1 \end{cases}$$

که به بیان دیگر میتوان گفت که باید H یک ماتریس مثبت معین باشد و نرخ یادگیری کوچک تر از $\frac{1}{\lambda_{max}}$ باشد.

۴. حال بررسی کنید اگر از روش نیوتن استفاده کنیم، یادگیری به چه صورت خواهد بود؟ چند گام طول می کشد تا همگرا شویم؟

ابتدا ماتریس ژاکوبی را محاسبه میکنیم.

$$J = \left[\frac{\partial \nabla \mathcal{L}(W)}{\partial W}\right]^T = \left[\frac{\partial}{\partial W}(2Hw)\right]^T = [2H]^2 = 2H^T = 2H$$
$$w \leftarrow w - J^{-1}[\nabla \mathcal{L}] = w - 2^{-1}H^{-1}(2Hw) = w - w = 0$$

مسئله در اولین مرحله همگرا میشود، زیرا تابع هدف تابعی درجه ۲ است و روش نیوتون حل با تقریب درجه دوم برای تابع هدف است.

۵. چرا با وجود اینکه روش مرتبه ۲ نیوتن از روش مرتبه ۱ گرادیان کاهشی بسیار سریع تر همگرا می شود، در آموزش شبکه های عمیق از آن استفاده نمی شود؟

زیرا هم از لحاظ زمان محاسباتی سنگین است، زیرا بطورکلی محاسبه ماتریس ژاکوبی $\mathcal{O}(n^2)$ زمان برده و حال محاسبه معکوس آن $\mathcal{O}(n^3)$ زمان میبرد که در شبکه های عمیق اصلا قابل انجام نیست.
از لحاظ استفاده مموری نیز ما نیاز به ذخیره کل ماتریس ژاکوبی داریم برای روش نیوتون در نتیجه برای شبکه های عمیق که تعداد پارامتر ها از مرتبه $1M \sim 1B$ پارامتر است، اصلا قابل انجام نیست زیرا $\mathcal{O}(n^2)$ مقدار باید ذخیره و محاسبه شوند.
روش های مرتبه ۲ ای برای شبکه های عمیق معرفی شده اند که برخی از آنها با تخمین زدن ماتریس هسین این کار را انجام میدهند مانند. [۴]

سوال ۶

تابع خطا در یک شبکه با اعمال Dropout گوسی-جمعی به شکل زیر است:

$$J_1 = \frac{1}{2}\left(y_d - \sum_{k=1}^{n}(w_k + \delta_k)x_k\right)^2$$

که در آن $\delta_k \sim N(0, \alpha w_k^2)$ می باشد.

۱. مقدار امید ریاضی گرادیان تابع هدف نسبت به متغیر w_k را محاسبه و تا حد امکان ساده کنید.

$$\mathbb{E}\left[\frac{\partial J_1}{\partial w_k}\right] = \frac{\partial \mathbb{E}[J_1]}{\partial w_k}$$

$$\mathbb{E}[J_1] = \mathbb{E}\left[\frac{1}{2}\left(y_d - \sum_{k=1}^{n}\overbrace{(w_k+\delta_k)}^{\tilde{w}_k}x_k\right)^2\right] = \mathbb{E}\left[\frac{1}{2}\left(y_d - \tilde{w}^T x\right)^2\right]$$

$$= \frac{1}{2}\mathbb{E}\left[y_d^2 - 2y_d\tilde{w}^T x + (\tilde{w}^T x)\tilde{w}^T x\right] = \frac{1}{2}\mathbb{E}\left[y_d^2 - 2y_d\tilde{w}^T x + x^T\tilde{w}\tilde{w}^T x\right]$$

فرض میکنیم کوواریانس δ_k ها با هم صفر باشد، در آن صورت:

$$\mathbb{E}\left[\tilde{w}^T\right] = w^T$$

$$\mathbb{E}\left[(\tilde{w}\tilde{w}^T)_{ij}\right] = \mathbb{E}\left[(w_i+\delta_i)(w_j+\delta_j)\right] = \mathbb{E}\left[w_iw_j + \delta_i\delta_j + \delta_iw_j + \delta_jw_i\right] = w_iw_j + \mathbb{E}\left[\delta_i\delta_j\right]$$

$$\mathbb{E}\left[\tilde{w}\tilde{w}^T\right] = ww^T + \alpha\Sigma, \quad \Sigma_{i,j} = \begin{cases} 0 & i \neq j \\ w_i^2 & i = j \end{cases}$$

$$\mathbb{E}[J_1] = \frac{1}{2}\left(y_d^2 - 2y_dw^Tx + x^T(ww^T + \alpha\Sigma)x\right)$$

$$= \frac{1}{2}\left(y_d^2 - 2y_dw^Tx + (w^Tx)^2 + \alpha x^T\Sigma x\right)$$

$$= \frac{1}{2}\left(y_d^2 - 2y_d\sum_{i=1}^{n}w_ix_i + \left(\sum_{i=1}^{n}w_ix_i\right)^2 + \alpha\sum_{i=1}^{n}w_i^2x_i^2\right)$$

$$= \frac{1}{2}\left(y_d - \sum_{i=1}^{n}w_ix_i\right)^2 + \frac{1}{2}\alpha\sum_{i=1}^{n}w_i^2x_i^2 = J_0 + \frac{1}{2}\alpha\sum_{i=1}^{n}w_i^2x_i^2$$

$$\mathbb{E}\left[\frac{\partial J_1}{\partial w_k}\right] = \frac{\partial \mathbb{E}[J_1]}{\partial w_k} = -\left(y_d - \sum_{i=1}^{n}w_ix_i\right)x_k + \alpha w_kx_k^2 = \nabla J_0 + \alpha w_kx_k^2$$

$$\mathbb{E}\left[\frac{\partial J_1}{\partial w_k}\right] = -\left(y_d - \sum_{i=1}^{n}w_ix_i\right)x_k + \alpha w_kx_k^2$$

۲. آیا می توانید تعبیری از رگولاسیون با استفاده از این نوع Dropout ارائه دهید؟

از روی رابطه بدست آمده $\mathbb{E}[J_1] = J_0 + \frac{1}{2}\alpha\sum_{i=1}^{n}w_i^2x_i^2$ واضح است که به طور میانگین تاثیر Dropout معرفی شده دقیقا مانند l_2 − regularization عمل میکند و گرادیان ها و تابع هزینه را تغییر میدهد.
صرفا با این تفاوت که چون به صورت جمعی انجام داده ایم، با ضریب اهمیت هر داده x_k اعمال میشود.

*

References

[1] Y. Tsukada and H. Iiduka, “Relationship between batch size and number of steps needed for nonconvex optimization of stochastic gradient descent using armijo line search,” 2023.

[2] P. Luo, X. Wang, W. Shao, and Z. Peng, “Towards understanding regularization in batch normalization,” 2019.

[3] M. Teye, H. Azizpour, and K. Smith, “Bayesian uncertainty estimation for batch normalized deep networks,” 2018.

[4] Z. Yao, A. Gholami, S. Shen, M. Mustafa, K. Keutzer, and M. W. Mahoney, “Adahessian: An adaptive second order optimizer for machine learning,” 2021.