گرادیان کاهشی متعامد (OGD) در یادگیری پیوسته

علی احمدی اسفیدی

۸ خرداد ۱۴۰۴

چکیده

چکیده: یادگیری پیوسته، چالش اساسی در توسعه سیستم‌های هوش مصنوعی است که قادر به یادگیری متوالی وظایف جدید بدون فراموش کردن دانش قبلی باشند. شبکه‌های عصبی عمیق، علی‌رغم توانایی‌های چشمگیرشان در حل مسائل پیچیده، با پدیده "فراموشی فاجعه‌بار" (Catastrophic Forgetting) مواجه هستند؛ به این معنی که با آموزش بر روی وظایف جدید، عملکرد آن‌ها بر روی وظایف قبلی به شدت افت می‌کند. در این گزارش، ما به بررسی رویکرد "گرادیان کاهشی متعامد" می‌پردازیم که به منظور مقابله با این چالش معرفی شده است. OGD با محدود کردن جهت به‌روزرسانی گرادیان‌ها در فضای پارامتر، تضمین می‌کند که خروجی شبکه عصبی بر روی وظایف پیشین تغییر نکند، در حالی که همچنان جهت مفیدی برای یادگیری وظیفه جدید ارائه می‌دهد. این روش با استفاده بهینه از ظرفیت بالای شبکه‌های عصبی و بدون نیاز به ذخیره داده‌های قبلی (که ممکن است نگرانی‌های حریم خصوصی ایجاد کند)، کارایی خود را در معیارهای بنچمارک نشان داده است. نتایج تجربی حاکی از اثربخشی OGD در حفظ دانش گذشته و یادگیری موثر وظایف جدید در سناریوهای یادگیری پیوسته است.

۱ مقدمه

در سال‌های اخیر، شبکه‌های عصبی عمیق به پیشرفت‌های قابل توجهی در حوزه‌های مختلف هوش مصنوعی، از جمله بینایی ماشین، پردازش زبان طبیعی و رباتیک دست یافته‌اند. با این حال، یکی از محدودیت‌های اصلی این مدل‌ها در محیط‌های یادگیری پویا، مشکل یادگیری پیوسته (Continual Learning) است. در بسیاری از کاربردهای دنیای واقعی، سیستم‌های هوشمند باید به طور مستمر دانش جدید را بدون بازآموزی از ابتدا و بدون فراموش کردن اطلاعاتی که قبلاً آموخته‌اند، کسب کنند. پدیده "فراموشی فاجعه‌بار" به این واقعیت اشاره دارد که آموزش یک شبکه عصبی بر روی یک وظیفه جدید می‌تواند به طور ناگهانی و شدید عملکرد آن را بر روی وظایف پیشین تخریب کند. این مشکل از تداخل بین به‌روزرسانی‌های پارامترها برای وظایف مختلف ناشی می‌شود.

برای غلبه بر این چالش، رویکردهای متعددی پیشنهاد شده‌اند که می‌توان آن‌ها را به سه دسته اصلی تقسیم کرد: روش‌های مبتنی بر مرور، روش‌های مبتنی بر تنظیم منظم و روش‌های مبتنی بر معماری. هر یک از این رویکردها مزایا و معایب خاص خود را دارند.

در این گزارش، تمرکز ما بر روی روش "گرادیان کاهشی متعامد" است که یک رویکرد نوین برای مقابله با فراموشی فاجعه‌بار در یادگیری پیوسته ارائه می‌دهد. OGD با بهره‌گیری از مفهوم تعامد در فضای گرادیان‌ها، به دنبال به‌روزرسانی پارامترهای شبکه به گونه‌ای است که تاثیر منفی بر عملکرد وظایف قبلی

به حداقل رسانده شود، در حالی که شبکه قادر به یادگیری موثر وظیفه جدید باشد. این روش از طریق پروژکتور (Projection) گرادیان وظیفه جدید به یک زیرفضای خاص، اطمینان حاصل می‌کند که تغییرات پارامترها در جهتی نباشد که به دانش قبلی آسیب بزند. OGD نه تنها به حفظ دانش قبلی کمک می‌کند، بلکه این کار را بدون نیاز به ذخیره‌سازی نمونه‌های داده‌های قبلی انجام می‌دهد، که یک مزیت مهم از منظر حریم خصوصی و کارایی حافظه است.

هدف این گزارش، ارائه یک بررسی جامع از مفهوم OGD، مبانی نظری آن، و ارزیابی عملکرد آن در مقایسه با سایر روش‌های موجود در حوزه یادگیری پیوسته است. همچنین، به کاربردها و محدودیت‌های احتمالی این روش پرداخته خواهد شد.

۲ فرمول‌بندی مسئله

فرض کنید θ بردار پارامترهای شبکه عصبی باشد. در یادگیری پیوسته، شبکه به طور متوالی بر روی یک دنباله از وظایف $T_1, T_2, \ldots, T_k$ آموزش می‌بیند. هنگامی که شبکه در حال آموزش بر روی وظیفه T_k است، هدف این است که عملکرد بر روی T_k بهبود یابد، در حالی که عملکرد بر روی وظایف قبلی $T_1, \ldots, T_{k-1}$ حفظ شود.

تابع هزینه برای وظیفه فعلی T_k را $L_k(\theta)$ در نظر می‌گیریم. گرادیان این تابع هزینه نسبت به پارامترها $\nabla L_k(\theta)$ است. به‌روزرسانی پارامترها در روش گرادیان کاهشی استاندارد به صورت $\theta \leftarrow \theta - \eta \nabla L_k(\theta)$ انجام می‌شود، که در آن η نرخ یادگیری است. این به‌روزرسانی ممکن است باعث تغییرات ناخواسته در خروجی‌های وظایف قبلی شود و منجر به فراموشی گردد.

۳ الگوریتم GD

فرض کنید می‌خواهیم تابع هزینه‌ای به صورت

$$L(w) = \frac{1}{N} \sum_{i=1}^{N} \ell\big(y^{(i)}, f(x^{(i)}; w)\big)$$

را نسبت به پارامترها $w \in \mathbb{R}^p$ مینیمم کنیم. الگوریتم گرادیان دِسِنت کلاسیک به شکل زیر است:

الگوریتم گرادیان کاهشی به‌صورت زیر عمل می‌کند: فرض کنید تابع هزینه $J(w)$، نرخ یادگیری $\eta > 0$، و مقدار اولیه $w^{(0)}$ داده شده باشند. در هر تکرار t، ابتدا گرادیان تابع هزینه نسبت به w محاسبه می‌شود:

$$g^{(t)} = \nabla_w L(w^{(t)})$$

سپس پارامترها طبق رابطه زیر به‌روزرسانی می‌شوند:

$$w^{(t+1)} = w^{(t)} - \eta\, g^{(t)}$$

این روند تا زمانی که شرط توقف برقرار شود ادامه می‌یابد. در پایان، مقدار نهایی $w^{(T)}$ به عنوان خروجی الگوریتم در نظر گرفته می‌شود.

خواص:

- اگر J محدب و مشتق‌پذیر باشد و نرخ یادگیری η مناسب انتخاب شود، الگوریتم به کمینهٔ سراسری همگرا می‌شود.
- در دیتاست‌های بزرگ معمولاً نسخه‌ی Stochastic Gradient Descent (SJD) یا Mini-batch GD به کار می‌رود که به جای مجموع کامل، روی زیرمجموعه‌ای از نمونه‌ها گرادیان را تقریب می‌زنند.

۴ الگوریتم OGD

OGD به دنبال یافتن یک به‌روزرسانی $\Delta\theta$ است که دو شرط اصلی را برآورده کند:

۱. $\Delta\theta$ باید به طور مؤثر وظیفه جدید T_k را یاد بگیرد.

۲. $\Delta\theta$ باید تأثیر کمی بر روی خروجی‌های وظایف قبلی $T_1, \ldots, T_{k-1}$ داشته باشد.

برای تحقق شرط دوم، OGD از مفهوم متعامد بودن (orthogonal) استفاده می‌کند. به طور خاص، OGD فرض می‌کند که تغییرات در خروجی شبکه برای وظایف قبلی می‌تواند به عنوان یک محدودیت خطی بر روی $\Delta\theta$ بیان شود.

۱.۴ تقریب خطی خروجی شبکه

فرض کنید $f(x_i, \theta)$ خروجی شبکه برای نمونه x_i با پارامترهای θ باشد. تغییر در خروجی برای وظیفه قبلی T_j (که $j < k$) ناشی از یک تغییر کوچک $\Delta\theta$ را می‌توان به صورت خطی تقریب زد:

$$\Delta f(x_i, \theta) \approx \nabla_\theta f(x_i, \theta)^T \Delta\theta$$

برای جلوگیری از فراموشی، OGD می‌خواهد $\Delta f(x_i, \theta)$ برای تمام نمونه‌ها و وظایف قبلی نزدیک به صفر باشد. این بدان معناست که $\Delta\theta$ باید متعامد به گرادیان‌های خروجی‌های وظایف قبلی باشد.

۲.۴ تعریف فضای بی‌تأثیر

برای هر وظیفه T_j و یک نمونه x_i از آن وظیفه، می‌توانیم گرادیان خروجی شبکه نسبت به پارامترها را محاسبه کنیم:

$$g_{j,i} = \nabla_\theta f(x_i, \theta)$$

برای حفظ عملکرد بر روی وظایف قبلی، OGD می‌خواهد $\Delta\theta$ در فضای متعامد به مجموعه گرادیان‌های $g_{j,i}$ (برای تمام $j < k$ و نمونه‌های x_i از T_j) باشد. این فضا، "فضای بی‌تأثیر" یا "فضای تعامد" نامیده می‌شود.

۳.۴ تصویرسازی گرادیان وظیفه جدید

هدف OGD این است که گرادیان وظیفه جدید $\nabla L_k(\theta)$ را به زیرفضایی تصویر کند که بر تمام گرادیان‌های وظایف قبلی متعامد باشد.

پایه فضای تعامد به گرادیان‌های وظایف قبلی را می‌توان با استفاده از تجزیه مقادیر منفرد (SVD) یا روش‌های دیگر به دست آورد. در عمل، OGD از مفهوم ماتریس تصویر (projection matrix) استفاده می‌کند.
اگر P_V ماتریس تصویری باشد که بردارها را به فضای V (فضای متعامد به گرادیان‌های وظایف قبلی) تصویر می‌کند، آنگاه به‌روزرسانی گرادیان به صورت زیر انجام می‌شود:

$$\Delta\theta = -\eta P_V g_k$$

ماتریس P_V به گونه‌ای ساخته می‌شود که $P_V G = 0$.

لم: فرض کنید $\mathbf{g}$ گرادیان تابع زیان $L(w)$ باشد و مجموعه $\mathcal{S} = \{\mathbf{v}_1, \ldots, \mathbf{v}_k\}$ پایه‌ای متعامد از یک زیرفضای پارامتری باشد. تعریف کنید:

$$\tilde{\mathbf{g}} = \mathbf{g} - \sum_{i=1}^{k} \mathrm{proj}_{\mathbf{v}_i}(\mathbf{g})$$

آنگاه بردار $-\tilde{\mathbf{g}}$ یک جهت نزولی برای تابع $L(w)$ است.

اثبات: برای اینکه یک بردار $\mathbf{u}$ جهت نزولی برای تابع زیان $L(w)$ باشد، باید داشته باشیم:

$$\langle \mathbf{u}, \mathbf{g} \rangle \leq 0$$

در اینجا:

$$\tilde{\mathbf{g}} = \mathbf{g} - \sum_{i=1}^{k} \mathrm{proj}_{\mathbf{v}_i}(\mathbf{g})$$

و چون $\tilde{\mathbf{g}}$ به زیرفضای $\mathrm{span}(\mathcal{S})$ متعامد است و جمع $\sum_{i=1}^{k} \mathrm{proj}_{\mathbf{v}_i}(\mathbf{g})$ نیز در این زیرفضا قرار دارد، بنابراین:

$$\left\langle \tilde{\mathbf{g}}, \sum_{i=1}^{k} \mathrm{proj}_{\mathbf{v}_i}(\mathbf{g}) \right\rangle = 0$$

حال داریم:

$$\begin{aligned}
\langle -\tilde{\mathbf{g}}, \mathbf{g} \rangle &= \left\langle -\tilde{\mathbf{g}}, \tilde{\mathbf{g}} + \sum_{i=1}^{k} \mathrm{proj}_{\mathbf{v}_i}(\mathbf{g}) \right\rangle \\
&= -\langle \tilde{\mathbf{g}}, \tilde{\mathbf{g}} \rangle - \left\langle \tilde{\mathbf{g}}, \sum_{i=1}^{k} \mathrm{proj}_{\mathbf{v}_i}(\mathbf{g}) \right\rangle \\
&= -\|\tilde{\mathbf{g}}\|^2 - 0 = -\|\tilde{\mathbf{g}}\|^2 \leq 0
\end{aligned}$$

پس $-\tilde{\mathbf{g}}$ یک جهت نزولی معتبر برای $L(w)$ است.

۴.۴ ساخت ماتریس تصویر

فرض کنید G_{prev} ماتریسی باشد که ستون‌های آن گرادیان‌های مربوط به وظایف قبلی را شامل می‌شود. این گرادیان‌ها می‌توانند گرادیان‌های خروجی شبکه برای هر نمونه یا گرادیان‌های متوسط (average gradients) برای هر وظیفه باشند. در مقاله، به طور خاص، از گرادیان‌های تابع هزینه برای هر وظیفه قبلی استفاده می‌شود. یعنی $g_j = \nabla L_j(\theta)$ برای $j < k$.

در هر مرحله آموزش برای وظیفه T_k:

۱. گرادیان وظیفه فعلی $g_k = \nabla L_k(\theta)$ محاسبه می‌شود.

۲. برای هر وظیفه قبلی T_j ($j < k$)، و معمولاً تنها برای آخرین وظیفه یا چند وظیفه قبلی)، گرادیان $g_j = \nabla L_j(\theta)$ محاسبه می‌شود. این گرادیان‌ها باید به روز نگه داشته شوند یا از یک حافظه کوچک بازیابی شوند.

۳. مجموعه گرادیان‌های $G_{prev} = \{g_1, g_2, \ldots, g_{k-1}\}$ (یا زیرمجموعه‌ای از آن‌ها) را در نظر می‌گیریم.

۴. برای تصویرسازی g_k به فضای متعامد به G_{prev}، از فرمول تصویر استفاده می‌کنیم:

$$g_k^{proj} = g_k - \sum_{j \in \text{prev\_tasks}} \frac{g_k^T g_j}{\|g_j\|^2} g_j$$

این یک فرآیند گرام-اشمیت (Gram-Schmidt) است، که در آن g_k از مؤلفه‌های موازی با هر یک از گرادیان‌های وظایف قبلی کاسته می‌شود. اگر گرادیان‌های قبلی متعامد نباشند، نیاز به یک روش تصویرسازی دقیق‌تر داریم. مقاله OGD رویکردی متفاوت را پیشنهاد می‌کند که از ماتریس‌های تصویرسازی پروژه‌ای استفاده می‌کند.

۵.۴ مراحل الگوریتم

مراحل کلی الگوریتم OGD برای یادگیری وظیفه T_k به شرح زیر است:

۱. اولیه:

- پارامترهای شبکه θ را مقداردهی اولیه کنید.
- یک مجموعه گرادیان‌های حافظه (مثلاً M_G) را که گرادیان‌های مهم وظایف قبلی را ذخیره می‌کند، مقداردهی اولیه کنید. در ساده‌ترین حالت، این حافظه می‌تواند خالی باشد و تنها گرادیان وظیفه قبلی (یا وظایف قبلی نزدیک) محاسبه شود.

۲. هر وظیفه T_k در دنباله یادگیری:

- بر روی T_k:
  - برای هر دسته (mini-batch) از داده‌های (X, Y) از وظیفه T_k:
    (آ) گرادیان تابع هزینه برای وظیفه فعلی را محاسبه کنید: $g_k = \nabla_\theta L_k(\theta; X, Y)$.
    (ب) گرادیان‌های وظایف قبلی برای تصویرسازی:

* اگر $k = 1$ (اولین وظیفه)، هیچ وظیفه قبلی وجود ندارد و گرادیان به‌روزرسانی بدون تصویرسازی انجام می‌شود.
* اگر $k > 1$، گرادیان‌های مربوط به وظایف قبلی $G_{prev} = \{g_j\}_{j<k}$ (یا یک زیرمجموعه از آن‌ها که در حافظه M_G ذخیره شده‌اند) را بازیابی کنید. این گرادیان‌ها می‌توانند:
  - گرادیان‌های نهایی پس از آموزش وظیفه T_j.
  - گرادیان‌های مربوط به نمونه‌های مشخصی از وظایف قبلی.
  - در عمل، معمولاً از نمونه‌های کمی از وظایف قبلی استفاده می‌شود تا گرادیان‌های آن‌ها در زمان آموزش وظیفه جدید محاسبه شوند.

  مقاله OGD به طور خاص از این ایده استفاده می‌کند که می‌توان گرادیان‌های تابع هزینه وظایف قبلی را (یا بر اساس یک مجموعه کوچک از داده‌های وظیفه قبلی که در یک "حافظه تکرار" (replay memory) ذخیره شده‌اند) در هر مرحله محاسبه کرد. فرض کنید $g_j = \nabla_\theta L_j(\theta; X_{replay}, Y_{replay})$ برای وظایف T_j که $j < k$ و X_{replay}, Y_{replay} نمونه‌هایی از وظایف قبلی هستند.

(ج) گرادیان:

* گرادیان فعلی g_k را به فضای متعامد به گرادیان‌های قبلی G_{prev} تصویر کنید. این فرآیند به صورت یک عملیات تصویرسازی خطی انجام می‌شود. در مقاله OGD، این تصویرسازی به صورت زیر پیشنهاد می‌شود (که یک راه حل بسته برای محدودیت‌های خطی است): فرض کنید G ماتریسی باشد که ستون‌های آن گرادیان‌های وظایف قبلی g_j هستند (که به طور معمول $g_j = \nabla L_j(\theta)$ برای $j < k$ است). به روز رسانی مطلوب $\Delta\theta$ باید $\nabla L_k(\theta)$ را کمینه کند، با این محدودیت که $G^T \Delta\theta = 0$. این مسئله بهینه‌سازی را می‌توان با ضرب‌کننده‌های لاگرانژ حل کرد که منجر به راه حل زیر برای گرادیان تصویر شده $\tilde{g}_k$ می‌شود:

$$\tilde{g}_k = g_k - \sum_{v \in s} proj_v(g)$$

این فرمول، گرادیان g_k را به فضای متعامد به بردارهای ستون G تصویر می‌کند.

(د) روزرسانی پارامترها:

* پارامترهای شبکه را با استفاده از گرادیان تصویر شده به‌روزرسانی کنید:

$$\theta \leftarrow \theta - \eta \tilde{g}_k$$

که η نرخ یادگیری است.

(ه) روزرسانی حافظه گرادیان:

* پس از اتمام آموزش وظیفه T_k، گرادیان مربوط به T_k (مثلاً گرادیان نهایی $\nabla L_k(\theta)$) را به مجموعه حافظه گرادیان M_G اضافه کنید تا در آینده برای تصویرسازی وظایف بعدی استفاده شود.

۵ نکات عملی و ملاحظات

۱. گرادیان‌ها: ذخیره تمام گرادیان‌های وظایف قبلی ممکن است از نظر حافظه پرهزینه باشد. OGD پیشنهاد می‌کند که تنها یک زیرمجموعه کوچک و مهم از گرادیان‌ها یا گرادیان‌های وظایف اخیر ذخیره شوند.

۲. های حافظه تکرار (Replay Memory): برای محاسبه گرادیان‌های وظایف قبلی در حین آموزش وظیفه جدید (که شبکه بر روی آن وظایف آموزش نمی‌بیند)، OGD از یک حافظه تکرار کوچک برای ذخیره تعداد کمی از نمونه‌های هر وظیفه قبلی استفاده می‌کند. این نمونه‌ها برای محاسبه گرادیان‌های g_j در هر مرحله استفاده می‌شوند.

۳. ارزیابی: عملکرد OGD با اندازه‌گیری دقت بر روی وظایف قبلی پس از آموزش بر روی وظایف جدید (برای ارزیابی میزان فراموشی) و همچنین دقت بر روی وظیفه فعلی (برای ارزیابی توانایی یادگیری) ارزیابی می‌شود.

۴. با روش‌های دیگر: OGD با روش‌هایی مانند EWC (Elastic Weight Consolidation)، LWF (Learning without Forgetting)، A-GEM (Averaged Gradient Episodic Memory) و SGD (Stochastic Gradient Descent) مقایسه می‌شود. مزیت اصلی OGD این است که به طور مستقیم بر روی فضای گرادیان عمل می‌کند و از نگهداری اطلاعات پیچیده‌تر مانند ماتریس‌های فیشر (در EWC) یا مدل‌های جداگانه (در LWF) اجتناب می‌کند.

۶ مثال عددی

۱.۶ مثال عددی برای GD

تابع هزینهٔ یک بعدی ساده:

$$L(w) = \frac{1}{2}(w-3)^2.$$

گرادیان:

$$\nabla L(w) = w - 3.$$

نقطه شروع $w^{(0)} = 0$ و نرخ یادگیری $\eta = 0.1$. مراحل اولیه:

$$w^{(1)} = w^{(0)} - 0.1\,(w^{(0)} - 3) = 0 - 0.1\,(0-3) = 0.3,$$
$$w^{(2)} = 0.3 - 0.1\,(0.3-3) = 0.3 - 0.1\,(-2.7) = 0.57,$$
$$w^{(3)} = 0.57 - 0.1\,(0.57-3) = 0.57 - 0.1\,(-2.43) = 0.813.$$

تا همگرایی به مقدار تقریبی $w^* \approx 3$ ادامه می‌یابد.

۲.۶ مثال عددی برای OGD

در این مثال ساده، فرض کنید پارامتر مدل $w = (w_1, w_2)$ و دو وظیفه داریم:

$$L_1(w) = \frac{1}{2}[(w_1 - 1)^2 + (w_2)^2],$$
$$L_2(w) = \frac{1}{2}[(w_1)^2 + (w_2 - 1)^2].$$

نرخ یادگیری $\eta = 0.1$ و مقدار اولیه $w_0 = (0, 0)$.

۱. **وظیفه ۱:** گرادیان $g_1 = \nabla L_1(w_0) = (w_1 - 1, w_2)\big|_{(0,0)} = (-1, 0)$.

۲. چون $S = \emptyset$، پروژه‌کردن تاثیری ندارد، پس $\widetilde{g}_1 = (-1, 0)$.

۳. به‌روزرسانی: $w \leftarrow w_0 - 0.1\,(-1, 0) = (0.1, 0)$.

۴. ذخیره گرادیان (برای این مثال ساده همان g_1):

$$u_1 = g_1 = (-1, 0), \quad S = \{(-1, 0)\}.$$

۵. **وظیفه ۲:** گرادیان $g_2 = \nabla L_2(w) = (w_1, w_2 - 1)\big|_{(0.1,0)} = (0.1, -1)$.

۶. پروژه کردن روی بردار u_1:

$$\mathrm{proj}_{u_1}(g_2) = \frac{\langle (0.1, -1), (-1, 0) \rangle}{\langle (-1, 0), (-1, 0) \rangle}(-1, 0) = \frac{-0.1}{1}(-1, 0) = (0.1, 0).$$

۷. گرادیان متعامد:

$$\widetilde{g}_2 = g_2 - \mathrm{proj}_{u_1}(g_2) = (0.1, -1) - (0.1, 0) = (0, -1).$$

۸. به‌روزرسانی:

$$w \leftarrow (0.1, 0) - 0.1\,(0, -1) = (0.1, 0.1).$$

۹. ذخیره‌سازی گرادیان جدید پس از پروژه:

$$u_2 = \widetilde{g}_2 = (0, -1), \quad S = \{(-1, 0), (0, -1)\}.$$

۷ پیاده‌سازی OGD در PyTorch

در این بخش قطعات کد اصلی و توضیحات مربوطه آورده شده است.

۱.۷ وارد کردن کتابخانه‌ها و تنظیم بذر تصادفی

```
import torch, torch.nn as nn, torch.optim as optim
import matplotlib.pyplot as plt, numpy as np
from copy import deepcopy

torch.manual_seed(42)
np.random.seed(42)
```

- torch و زیرماژول‌های nn، optim برای تعریف و آموزش مدل.
- matplotlib برای رسم نمودارها و numpy برای محاسبات عددی.
- با manual_seed بذر تصادفی برای تکرارپذیری ثابت می‌شود.

۲.۷ انتخاب دستگاه (CPU یا GPU)

```
device = "cuda" if torch.cuda.is_available() else "cpu"
```

مدل و داده‌ها روی device منتقل می‌شوند تا در صورت وجود کارت گرافیک از آن استفاده شود.

۳.۷ تعریف توابع هدف

```
def f_A(x): return torch.sin(x)
def f_B(x): return torch.sin(x**3)
def f_C(x): return torch.exp(-x**2)
```

هر تابع ویژگی متفاوتی دارد تا وظایف متوالی شبیه‌سازی گردد.

۴.۷ ساخت دیتاست

```
def make_dataset(f, n=128, rng=(5-, 5), noise_std=0.1):
    a, b = rng
    x = torch.rand(n, 1, device=device) * (b - a) + a
    x, _ = torch.sort(x, dim=0)
    y = f(x) + noise_std * torch.randn_like(x)
    return x, y
```

با اضافه کردن نویز گوسی به خروجی، داده‌های آموزشی واقعی‌تر می‌شوند.

۵.۷ تعریف مدل ساده

```
class SimpleNN(nn.Module):
    def __init__(self, hidden=50):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(1, hidden), nn.ReLU(),
```

```
            nn.Linear(hidden, 1)
        )
    def forward(self, x):
        return self.net(x)
```

یک شبکهٔ دو لایه با ۵۰ نورون مخفی و تابع فعال‌سازی ReLU.

۶.۷ توابع کمکی برای OGD

```
def flat_gradients(model):
    return torch.cat([p.grad.view((1-
        for p in model.parameters() if p.grad is not None])

def write_flat_to_grads(model, vec):
    i=0
    for p in model.parameters():
        if p.grad is None: continue
        n=p.numel()
        p.grad.copy_(vec[i:i+n].view_as(p))
        i+=n
```

- flat_gradients: استخراج همهٔ گرادیان‌ها در یک بردار تخت.
- write_flat_to_grads: نوشتن مجدد بردار تخت به گرادیان پارامترها.

۷.۷ تابع آموزش با OGD

```
def train(model, x, y, epochs=250, lr=1e2-, memory=None):
    crit=nn.MSELoss()
    opt=optim.SGD(model.parameters(), lr=lr)
    for _ in range(epochs):
        opt.zero_grad()
        loss=crit(model(x), y)
        loss.backward()
        if memory:
            g=flat_gradients(model)
            for v in memory:
                g -= torch.dot(g,v)/(v.norm()**2+1ev*(8-
            write_flat_to_grads(model, g)
        opt.step()
    return loss.item()
```

- اگر memory (لیست گرادیان‌های وظایف قبلی) داده شود، ابتدا گرادیان فعلی را پروژه می‌کنیم.

۸.۷ ذخیره‌سازی گرادیان هر وظیفه

```
def gradient_of_task(model, x, y):
```

```
    model.zero_grad()
    nn.MSELoss()(model(x), y).backward()
    return flat_gradients(model).detach()
```

برای استفاده در حافظهٔ OGD، گرادیان وظیفه را از گراف محاسباتی جدا می‌کنیم.

۹.۷ شبیه‌سازی شاخه‌های SGD و OGD

کد زیر ابتدا روی وظیفهٔ A آموزش دیده، سپس با و بدون OGD به ترتیب B و C را آموزش می‌دهد و MSE روی A را بعد از هر مرحله گزارش می‌کند.

```
def run_branch(name, use_ogd):
    base_model = SimpleNN().to(device)
    train(base_model, x_A, y_A)
    mse_A_start = nn.MSELoss()(base_model(x_A), y_A).item()
    memory = []
    if use_ogd:
        memory.append(gradient_of_task(base_model, x_A, y_A))
    model_B = deepcopy(base_model)
    model_C = deepcopy(base_model)
    train(model_B, x_B, y_B, memory=memory if use_ogd else None)
    mse_A_after_B = nn.MSELoss()(model_B(x_A), y_A).item()
    if use_ogd:
        memory.append(gradient_of_task(model_B, x_B, y_B))
    train(model_C, x_C, y_C, memory=memory if use_ogd else None)
    mse_A_after_C = nn.MSELoss()(model_C(x_A), y_A).item()
    return base_model, model_B, model_C

model_A0_sgd, model_B_sgd, model_C_sgd = run_branch("SGD branch", False)
model_A0_ogd, model_B_ogd, model_C_ogd = run_branch("OGD branch", True)
...
```

۱۰.۷ رسم نتایج

```
def plot_fit(model, x, y, title):
    model.eval()
    with torch.no_grad(): p = model(x).cpu()
    plt.scatter(x.cpu(), y.cpu(), s=10)
    plt.plot(x.cpu(), p, c="r")
    plt.title(title); plt.grid(); plt.show()

plot_fit(...)
```

نمودارها نشان می‌دهند که در شاخهٔ OGD مدل چگونه اطلاعات وظیفهٔ A را بهتر حفظ می‌کند.

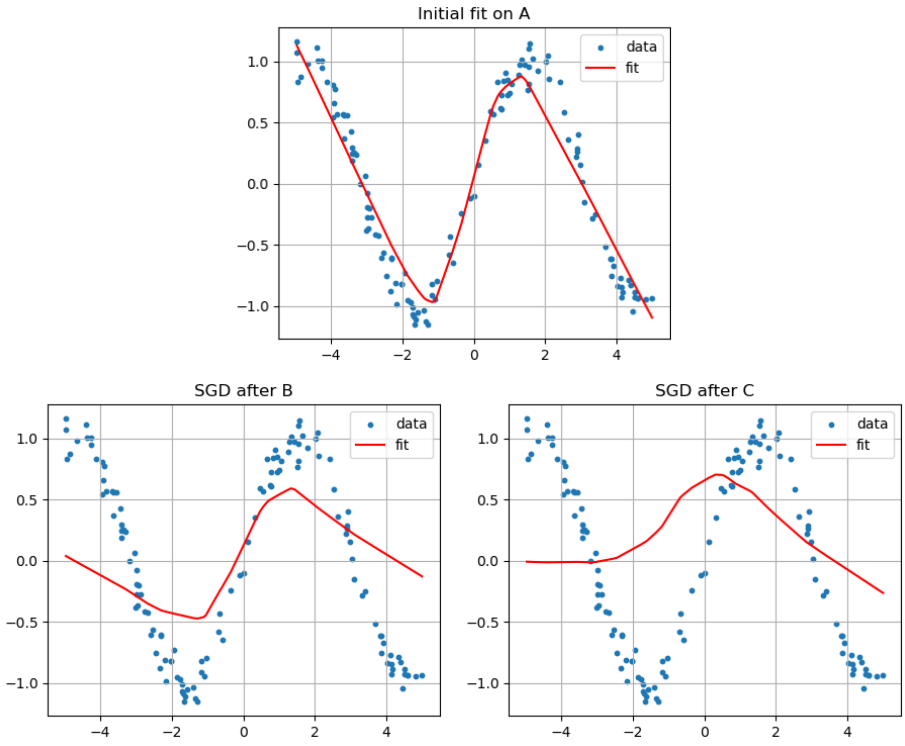
۸ تحلیل نتایج اجرایی

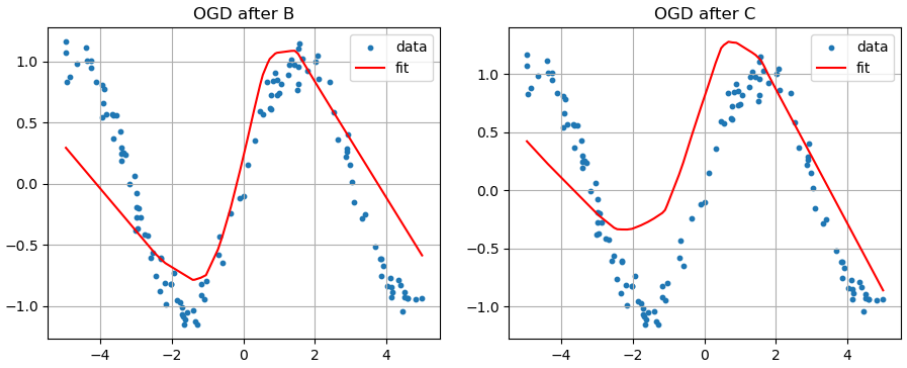
در نتایج اجرا دو شاخهٔ مختلف SGD) و (OGD روی وظیفهٔ A (تابع $\sin(x)$) مقایسه شده‌اند:

```
=== SGD branch | SGD ===
MSE_A after Task A: 0399.0
Training on Task B …
MSE_A after B: 3274.0
Training on Task C …
MSE_A after C: 5061.0

=== OGD branch | OGD ===
MSE_A after Task A: 0563.0
Training on Task B …
MSE_A after B: 1959.0
Training on Task C …
MSE_A after C: 2544.0
```

Plots:





۹ مقایسه

- OGD با پروژه‌سازی عمودی گرادیان‌ها تداخل یادگیری وظایف را کمینه می‌کند.
- خطای نهایی در OGD تقریباً یک‌دوم خطای معادل در SGD ساده است.
- این روش بدون نیاز به ذخیرهٔ داده‌های قبلی، فراموشی فاجعه‌آمیز را کاهش می‌دهد.

۱۰ نتیجه‌گیری

الگوریتم Orthogonal Gradient Descent (OGD) یک رویکرد موثر برای مقابله با فراموشی فاجعه‌بار در یادگیری پیوسته ارائه می‌دهد. این الگوریتم با تصویرسازی گرادیان وظیفه جدید به فضایی که بر گرادیان‌های وظایف قبلی متعامد است، تضمین می‌کند که به‌روزرسانی پارامترها حداقل تأثیر را بر دانش قبلی داشته باشد. OGD یک راه‌حل از دیدگاه فضای پارامتر ارائه می‌دهد که می‌تواند انواع مختلف آن در کاربردهای عملی یادگیری پیوسته مفید باشد.

۱۱ منابع

- *PyTorch: An Imperative Style, High-Performance Deep Learning Library*
- *Orthogonal Gradient Descent for Continual Learning (Farajtabar, et al.)*