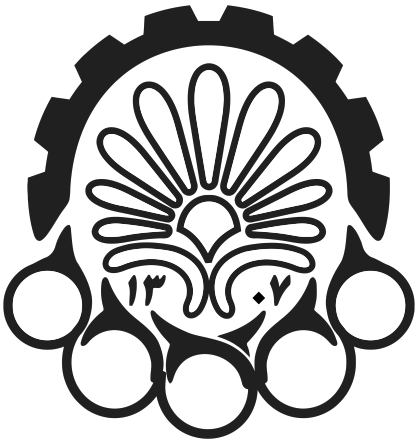
شبکه‌های عصبی و یادگیری عمیق
دکتر صفابخش



دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)
دانشکده مهندسی کامپیوتر

رضا آدینه پور ۴۰۲۱۳۱۰۵۵

تمرین پنجم
شبکه‌های RNN

۱۰ خرداد ۱۴۰۳

شبکه‌های عصبی و یادگیری عمیق

تمرین پنجم

رضا آدینه پور ۴۰۲۱۳۱۰۵۵

دانشکده مهندسی کامپیوتر

سوال اول - نظری

به سوالات زیر بصورت خلاصه و برای هر یک حداکثر در سه بند پاسخ دهید:

۱. به‌طور کلی بهینه‌سازها[۱] (نظیر ADAM) به دنبال یافتن وزن‌های شبکه‌های عصبی هستند بطوریکه توابع هزینه[۲] کمینه شود. مشتق‌پذیر بودن توابع یاد شده چه تاثیری در بهینه‌ساز دارد؟ اگر مشتق‌پذیر نباشد، چه رویکردهایی برای بهینه‌سازی آن وجود دارد؟ یک مورد را به دلخواه توضیح دهید.

پاسخ

بهینه‌سازهایی مانند ADAM از گرادیان توابع هزینه برای به‌روزرسانی وزن‌های شبکه استفاده می‌کنند. مشتق‌پذیر بودن این توابع به معنای وجود گرادیان است که به بهینه‌سازها کمک می‌کنند جهت حرکت به سمت مینیمم سراسری را پیدا کنند. بدون مشتق‌پذیری، تعیین دقیق جهت و میزان تغییر وزن‌ها دشوار می‌شود.
در صورتی که تابع هزینه مشتق‌پذیر نباشد، روش‌های دیگری نظیر الگوریتم‌های مبتنی بر مشتقات تقریبی یا تکنیک‌های بهینه‌سازی بدون مشتق مانند الگوریتم ژنتیک یا بهینه‌سازی ازدحام ذرات (Particle Swarm Optimization) مورد استفاده قرار می‌گیرند.
بهینه‌سازی ازدحام ذرات (PSO) یک روش الهام گرفته از طبیعت است که بدون نیاز به مشتق تابع کار می‌کند. این الگوریتم با استفاده از حرکت ذرات در فضای جستجو و به‌روزرسانی موقعیت‌های آنها بر اساس بهترین موقعیت‌های خود و همسایگانشان، به سمت بهینه و پیدا کردن مینیمم سسراسری حرکت می‌کند.

*

References

[1] I. Goodfellow, Y. Bengio & A. Courville, (2016). Deep Learning. MIT Press (Link)

[2] Rios, Luis Miguel, and Nikolaos V. Sahinidis. ”Derivative-free optimization: a review of algorithms and comparison of software implementations.” Journal of Global Optimization 56.3 (2013): 1247-1293.

۲. محدب[۳] بودن توابع به چه معناست و چرا مطلوب است که در بهینه‌سازی، توابع هزینه محدب باشد؟ اگر محدب نباشد، چگونه می‌توان آن را بهینه نمود؟

[۱] Optimizer
[۲] Loss Functions
[۳] Convex

پاسخ

یک تابع محدب است اگر خط واصل بین هر دو نقطه از نمودار آن تابع، همیشه بالای نمودار تابع قرار گیرد. این ویژگی باعث می‌شود که هر مینیمم محلی، مینیمم سراسری نیز باشد، که جستجو برای یافتن نقطه بهینه را آسان می‌کند.
توابع محدب از این جهت برای ما مفید هستند چون تضمین می‌کنند که بهینه‌سازها می‌توانند به راحتی و با اطمینان به نقطه بهینه سراسری برسند، بدون اینکه در مینیمم‌های محلی گیر کنند. این ویژگی فرآیند بهینه‌سازی را کارآمدتر و قابل اعتمادتر می‌سازد.
در صورت محدب نبودن توابع هزینه، می‌توان از تکنیک‌هایی نظیر الگوریتم‌های تصادفی (Stochastic Algorithms)، چندین شروع تصادفی (Multiple Random Starts)، و روش‌های بهینه‌سازی مبتنی بر شبیه‌سازی (Simulated Annealing) برای جستجوی بهینه سراسری استفاده کرد.

*

References

[1] S. Boyd & L. Vandenberghe, (2004). Convex Optimization. Cambridge University Press. (Link)

[2] J. Nocedal, & S. j. Wright, (2006). Numerical Optimization. Springer. (Chapters on non-convex optimization)

۳. الگوریتم بهینه‌سازی نیوتن را مطالعه کرده و آن را با نزول در راستای گرادیان[۴] مقایسه کنید. در چه نوع مسائلی استفاده از الگوریتم نیوتن ارجحیت دارد؟

پاسخ

الگوریتم نیوتون، از مشتق دوم تابع هزینه (hessian) برای بهبود به‌روزرسانی وزن‌ها استفاده می‌کند. بروزرسانی وزن‌ها با استفاده از فرمول زیر انجام می‌شود که H همان ماتریس hessian است.

$$\theta_{\text{new}} = \theta_{\text{old}} - H^{-1}\nabla L(\theta_{\text{old}})$$

نزول گرادیان فقط از مشتق مرتبه اول استفاده می‌کند و به‌روزرسانی وزن‌ها را با توجه به جهت و میزان مشتق انجام می‌دهد. الگوریتم نیوتن به دلیل استفاده از اطلاعات مشتق مرتبه دوم می‌تواند به سرعت به نقطه بهینه نزدیک شود، اما محاسبه و بدست آوردن وارون ماتریس hessian هزینه‌بر است.
الگوریتم نیوتن برای مسائلی با تعداد پارامترهای کم و توابع ساده، که محاسبه و وارون‌سازی ماتریس hessian را دشوار نکند، مناسب‌تر است. این الگوریتم در مسائلی که به دقت بالاتر و همگرایی سریع‌تر نیاز داریم، ارجحیت دارد.

[۴] Gradient Descent

پاسخ

*

References

[1] I. Goodfellow, Y. Bengio & A. Courville, (2016). Deep Learning. MIT Press, (Chapter on Optimization) (Link)

[2] J. Nocedal, & J. S. Wright, (2006). Numerical Optimization. Springer. (Chapters on second-order methods)

۴. ضمن مطالعه کلی الگوریتم AdaGrad، بیان کنید که چگونه می‌توان از آن برای بهینه ساختن نرخ یادگیری[۵] بهره گرفت.

فرض کنید مسئله‌ی دسته بندی دودویی بحرانی بودن/نبودن شرایط یک کارگاه صنعتی بر اساس اطلاعاتی محیطی آن را در اختیار دارید که داده‌های دما، رطوبت، فشار و ذرات معلق بر اساس سنسورهای نصب شده در هر یک ثانیه ارسال می‌گردد. شما بایستی با در نظر گرفتن دنباله‌ای از داده‌های ارسالی بتوانید تشخیص دهید که شرایط بحرانی است یا خیر.

پاسخ

AdaGrad (Adaptive Gradient) الگوریتم بهینه‌سازی‌ای است که نرخ یادگیری را به صورت داینامیک و متناسب با تاریخچه گرادیان تنظیم می‌کند. این الگوریتم با تقسیم نرخ یادگیری اولیه بر مجموع ریشه مربع گرادیان‌های قبلی، نرخ یادگیری را برای هر پارامتر به صورت جداگانه تنظیم می‌کند.
در AdaGrad، هر پارامتر نرخ یادگیری خاص خود را دارد که با توجه به میزان نوسانات آن پارامتر تنظیم می‌شود. این کار به الگوریتم اجازه می‌دهد تا در مسیرهای با گرادیان زیاد نرخ یادگیری را کاهش دهد و در مسیرهای با گرادیان کم آن را افزایش دهد، که منجر به بهینه‌سازی دقیق‌تر و جلوگیری از نوسانات شدید می‌شود.

*

References

[1] J. Duchi, E. Hazan, & Y. Singer, (2011). Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research, 12(Jul), 2121-2159.

[2] S. Ruder, (2016). An overview of gradient descent optimization algorithms. (Link)

۵. یک شبکه‌ی بازخوردی Elman که با دولایه‌ی مخفی که به ترتیب سه و دو نورون تعبیه شده است، طراحی نمایید و تعداد وزن‌های مورد نیاز برای یادگیری در این شبکه را با بیان علت محاسبه نموده و ابعاد تمامی بردارهای (Vectors & Tensors) مشاهده شده در شبکه (ورودی‌ها/میانی‌ها/خروجی‌ها) را با محاسبات و استدلال نمایش دهید. انتظار می‌رود که شما بتوانید سیر تغییرات ابعاد بردارها و چگونگی آن را نشان دهید؛ مثلا شکل بردار ورودی برای یک دسته (batch) چگونه تعیین می‌شود و تا رسیدن به خروجی شکل آن چرا و چگونه تغییر پیدا کرده است و با چه وزن‌هایی متاثر شده است.

[۵] Learning Rate

پاسخ

طبق صورت مسئله، فرضیات و نوتیشن‌های زیر را درنظر می‌گیریم:

(آ) ورودی $x(t)$ با ابعاد n_x است

(ب) لایه مخفی اول ($h_1(t)$) دارای ۳ نورون با ابعاد ۳

(ج) لایه مخفی دوم ($h_2(t)$) دارای ۲ نورون مخفی با ابعاد ۲

(د) خروجی $y(t)$ با ابعاد n_y

در مرحله اول تعداد وزن‌های لایه‌های مختلف را با بیان جزئیات محاسبه می‌کنیم:

(آ) **لایه ورودی به لایه مخفی اول:**

- وزن‌های بین ورودی و لایه مخفی اول: $\mathbf{W}_{xh1}$ با ابعاد $3 \times n_x$
- وزن‌های بازگشتی از خروجی لایه مخفی اول به خودش: $\mathbf{W}_{h1h1}$ با ابعاد 3×3
- بایاس‌های لایه مخفی اول: $\mathbf{b}_{h1}$ با ابعاد ۳

$$\text{تعداد وزن‌های لایه اول} = 3 \times n_x + 3 \times 3 + 3$$

(ب) **لایه مخفی اول به لایه مخفی دوم:**

- وزن‌های بین لایه مخفی اول و دوم: $\mathbf{W}_{h1h2}$ با ابعاد 2×3
- وزن‌های بازگشتی از خروجی لایه مخفی دوم به خودش: $\mathbf{W}_{h2h2}$ با ابعاد 2×2.
- بایاس‌های لایه مخفی دوم: $\mathbf{b}_{h2}$ با ابعاد ۲

$$\text{تعداد وزن‌های لایه دوم} = 2 \times 3 + 2 \times 2 + 2 = 12$$

(ج) **لایه مخفی دوم به لایه خروجی:**

- وزن‌های بین لایه مخفی دوم و خروجی: $\mathbf{W}_{h2y}$ با ابعاد $n_y \times 2$
- بایاس‌های لایه خروجی: $\mathbf{b}_y$ با ابعاد n_y

$$\text{تعداد وزن‌های لایه خروجی} = n_y \times 2 + n_y$$

در ادامه ابعاد بردارها در شبکه را محاسبه می‌کنیم:

(آ) **ورودی:**

- بردار ورودی $\mathbf{x}(t)$ با ابعاد n_x

پاسخ

(آ) **لایه مخفی اول:**

- ورودی به لایه مخفی اول:

$$\mathbf{h}_1(t) = \sigma(\mathbf{W}_{xh1}\mathbf{x}(t) + \mathbf{W}_{h1h1}\mathbf{h}_1(t-1) + \mathbf{b}_{h1})$$

- ابعاد $\mathbf{h}_1(t)$: ۳

(ب) **لایه مخفی دوم:**

- ورودی به لایه مخفی دوم:

$$\mathbf{h}_2(t) = \sigma(\mathbf{W}_{h1h2}\mathbf{h}_1(t) + \mathbf{W}_{h2h2}\mathbf{h}_2(t-1) + \mathbf{b}_{h2})$$

- ابعاد $\mathbf{h}_2(t)$: ۲.

(ج) **خروجی:**

- خروجی:

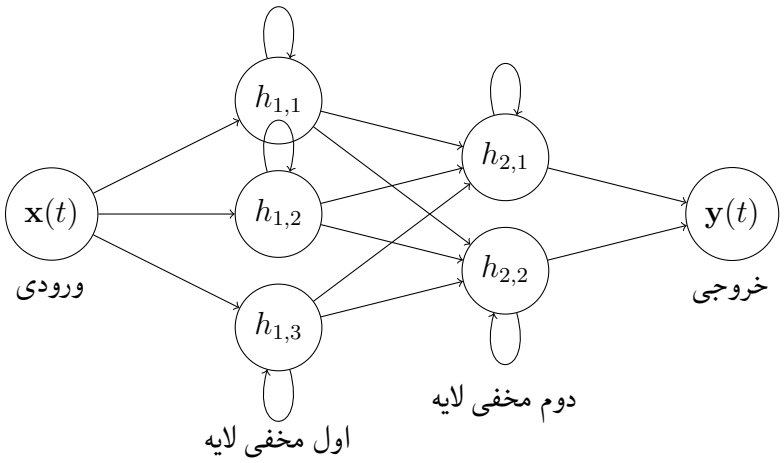
$$\mathbf{y}(t) = \mathbf{W}_{h2y}\mathbf{h}_2(t) + \mathbf{b}_y$$

- ابعاد $\mathbf{y}(t)$: n_y.

درنهایت با ترکیب همه وزن‌ها و بایاس‌ها تعداد کل وزن‌های شبکه به‌صورت زیر می‌شود:

$$\text{تعداد کل وزن‌ها} = (3 \times n_x + 3 \times 3 + 3) + (2 \times 3 + 2 \times 2 + 2) + (n_y \times 2 + n_y)$$
$$= (3n_x + 9 + 3) + (6 + 4 + 2) + (2n_y + n_y)$$
$$= 3n_x + 3n_y + 24$$

درنهایت دیاگرام شبکه طراحی شده به‌صورت زیر است:



شکل ۱: دیاگرام شبکه Elman با دو لایه مخفی