الگوریتم‌های بهینه‌سازی روش گرادیان نزولی*

چکیده: برای تحلیل الگتوریتم‌های بهینه سازی کاهش گرادیان، با وجود محبوبیت فزاینده، از آنالیز جعبه سیاه (BlackBox) استفاده می‌شود زیرا ارائه‌ی توضیح عملی از نقاط ضعف و قوت آن ها دشوار است. هدف این مقاله ایجاد درکی از رفتار الگوریتم‌های متفاوت برای خواننده است تا بتواند از آن‌ها بهره بگیرد. در طول این مقاله، به فرم‌های مختلف روش کاهش گرادیان نگاه می کنیم، چالش‌ها را به طور مختصر بیان می کنیم، پرکاربردترین الگوریتم‌های بهینه سازی را معرفی می کنیم، معماری سیستم‌های موازی و توزیع شده را مرور می‌کنیم و به دنبال روش‌های جدیدی برای بهینه سازی روش کاهش گرادیان خواهیم گشت.

۱ مقدمه

روش گرادیان نزولی یکی از محبوب‌ترین الگوریتم‌ها برای بهینه‌سازی و با فاصله‌ی زیاد، پرکاربردترین روش برای بهینه‌سازی شبکه‌های عصبی است. در عین حال، پیشرفته ترین کتابخانه‌ی یادگیری عمیق ماشین، شامل نمونه‌هایی از پیاده‌سازی الگوریتم‌های متفاوت برای بهینه‌سازی روش گرادیان نزولی است (مثل متن‌های راهنمای لازانیا[۱]، کافه[۲] و کراس[۳]). با اینحال از این الگوریتم‌ها برای بهینه سازی با روش جعبه سیاه (BlackBox) استفاده می‌شود زیرا ارائه‌ی توضیح عملی از نقاط ضعف و قوت آنها مشکل است.

هدف این مقاله ایجاد درکی از رفتار الگوریتم‌های مختلف بهینه‌سازی روش گرادیان نزولی برای خواننده است تا بتواند از آنها استفاده کند. در بخش ۲، ابتدا به بررسی فرم‌های مختلف روش گرادیان نزولی می‌پردازیم. سپس به طور خلاصه چالش‌های این الگوریتم را در خلال آموزش‌های بخش ۳ مرور می کنیم. پس از آن در بخش ۴، معمول ترین الگوریتم‌های بهینه سازی را معرفی می کنیم و نشان می‌دهیم که چگونه تلاش برای حل چالش‌های بهینه‌سازی، سبب ایجاد تغییراتی در رویکردهای استفاده شده در این الگوریتم‌ها شده است. سپس، در بخش ۵، نگاهی کوتاه به الگوریتم‌ها و معماری‌هایی برای بهینه‌سازی گرادیان نزولی در سیستم‌های موازی و توزیع شده می اندازیم. در نهایت، در بخش ۶ رویکردهای تازه‌ای که برای بهینه‌سازی گرادیان نزولی مفید هستند را در نظر می گیریم.

---

* این مقاله در اصل به صورت یک مطلب در یک وبلاگ به آدرس http://sebastianruder.com/optimizing-gradient-descent/index.html در ۱۹ ژانویه ۲۰۱۶ منتشر شده است.

[۱] Lasagne’s Documentation: http://lasagne.readthedocs.org/en/latest/modules/updates.html

[۲] Caffe’s Documentation: http://lasagne.readthedocs.org/en/latest/modules/updates.html

[۳] Keras’ Documentation: http://caffe.berkeleyvision.org/tutorial/solver.html

روش گرادیان نزولی راهی برای یافتن مینیمم یک تابع هدف مثل $J(\theta)$ است که بر اساس بردار پارامترهای مدل یعنی $\theta \in \mathbb{R}^d$ نوشته شده است؛ به این صورت که پارامترها در خلاف جهت گرادیان تابع نسبت به پارامترها یعنی $\nabla_\theta J(\theta)$ تغییر می‌کنند. نرخ یادگیری شبکه عصبی یا η اندازه‌ی گام تغییر پارامترها برای یافتن مینیمم (یا مینیمم نسبی) را تعیین می کند. به عبارت دیگر، در جهت شیب سطح ایجاد شده توسط تابع به سمت پایین حرکت می کنیم تا به یک دره برسیم. [۴]

۲ فرم‌های مختلف روش گرادیان نزولی

سه فرم از روش گرادیان نزولی وجود دارند که در حجم داده‌ای که برای محاسبه‌ی گرادیان تابع هدف استفاده می شود با هم تفاوت دارند. بسته به مقدار داده، تعادلی بین دقت تغییر پارامتر و مدت زمانی که برای تغییر پارامتر نیاز است ایجاد می کنیم.

۲.۱ گرادیان نزولی انبوه

روش گرادیان نزول ساده، که به نام گرادیان نزولی انبوه هم شناخته می‌شود، گرادیان تابع هزینه نسبت به بردار پارامتر θ برای کل مجموعه داده‌های ورودی سیستم محاسبه می شود:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$$

برای اینکه برای ایجاد هر *تغییر* باید گرادیان کل داده‌ها محاسبه شود، روش گرادیان نزولی انبوه می‌تواند بسیار کند باشد و برای مجموعه‌ای از داده ها که به قدری بزرگند که در حافظه جا نمی‌شوند قابل قبول نیست. درضمن این روش به ما اجازه نمی‌دهد که مدل را *لحظه‌ای* تغییر دهیم؛ به این معنا که نمی‌توان داده‌های جدید را حین کار به آن اضافه کرد.

در کدنویسی، روش گرادیان نزولی انبوه چیزی شبیه به این است:

```
for i in range (nb_epochs):
    params_grad = evaluate_gradient ( loss_function, data, params)
    params = params – learning_rate * params_grad
```

برای تعداد تکرار مشخص، ابتدا بردار گرادیان یعنی params_grad را نسبت به بردار پارامترها یعنی params برای تابع ضرر محاسبه می کنیم. دقت کنید که جدیدترین کتابخانه های یادگیری ژرف شامل دیفرانسیل گیری خودکار هستند که گرادیان نسبت به پارامترهای مشخص را به صورت بهینه محاسبه می کنند. اگر گرادیان را خودتان حساب کرده اید، بهتر است که صحت گرادیان ها را بررسی کنید.[۵]

---

[۴] اگر با روش گرادیان نزولی آشنایی ندارید، مطالب خوبی در مورد بهینه سازی سیستم های عصبی را می توانید در اینجا بیابید:
http://cs231n.github.io/optimization-1

[۵] برای چند توصیه عالی برای بررسی درست صحت گرادیان گیری، به آدرس زیر مراجعه کنید:
http://cs231n.github.io/neural-networks-3/

سپس پارامترها را در جهت گرادیان به اندازه‌ی نرخ یادگیری که تعیین می‌کند تغییرات ایجاد شده چقدر بزرگ باشند، تغییر می‌دهیم. روش گرادیان نزولی در سطوح محدب حتما به مینیمم مطلق و در سطوح غیر محدب حتما به مینیمم نسبی همگرا می شود.

۲.۲ گرادیان نزولی تصادفی

روش گرادیان نزولی تصادفی (Stochastic Gradient Descent: SGD) برخلاف روش قبلی، نسبت به *هرکدام* از داده ها مثل $x^{(i)}$ و مقدار متناظر آن $y^{(i)}$ یک گام تغییر ایجاد می کند:

$$\theta = \theta - \eta \cdot \nabla_\theta J\left(\theta; x^{(i)}; y^{(i)}\right)$$

روش گرادیان نزولی انبوه محاسبات اضافی و نالازم برای مجموعه داده‌های بزرگ انجام می‌دهد زیرا گرادیان داده‌های یکسانی را به طور مکرر قبل از ایجاد هر تغییر محاسبه می کند. روش تصادفی این محاسبات اضافی را با ایجاد یک تغییر با هر گرادیان‌گیری حذف می کند. به همین دلیل هم این روش بسیار سریع‌تر است و می تواند یادگیری لحظه ای هم داشته باشد. روش تصادفی تغییرات را با واریانس بالایی ایجاد می‌کند که باعث می شود تابع هدف مثل شکل ۱ به شدت نوسان کند.

در حالیکه روش گرادیان انبوه همواره به مینیمم همان قوسی که پارامترها را در بر می‌گیرد همگرا می‌شود، نوسانات روش تصادفی از یک طرف این روش را قادر می‌سازد که مینیمم های نسبی بهتری را بیابد و از طرف دیگر باعث می شود که درنهایت همگرایی به مقدار دقیق مینیمم سخت تر شود زیرا این روش در هر تکرار انحراف زیادی نسبت به مقدار اصلی دارد. با اینحال، ثابت شده که وقتی نرخ یادگیری را به آرامی زیاد می‌کنیم، این روش همان همگرایی روش انبوه را از خود نشان می‌دهد و با قابلیت اطمینان بالایی، در سطوح محدب به مینیمم مطلق و در سطوح غیر محدب به مینیمم نسبی همگرا می شود. تکه کد مربوط به این الگوریتم خیلی ساده یک حلقه‌ی تکرار (loop) در داده ها ایجاد کرده و گرادیان نسبت به هر نمونه را حساب می‌کند. دقت کنید همانطور که در بخش ۱. ۶ توضیح داده شده، در هر تکرار، ترتیب داده ها به صورت تصادفی تغییر داده می‌شود.

```
for i in range ( nb_eochs):
    np . random . shuffle ( data)
    for example in data:
        params_grad = evaluate_gradient ( loss_function, example, params )
        params = params – learning_rate*params_grad
```

۲.۳ گرادیان نزولی نیمه انبوه

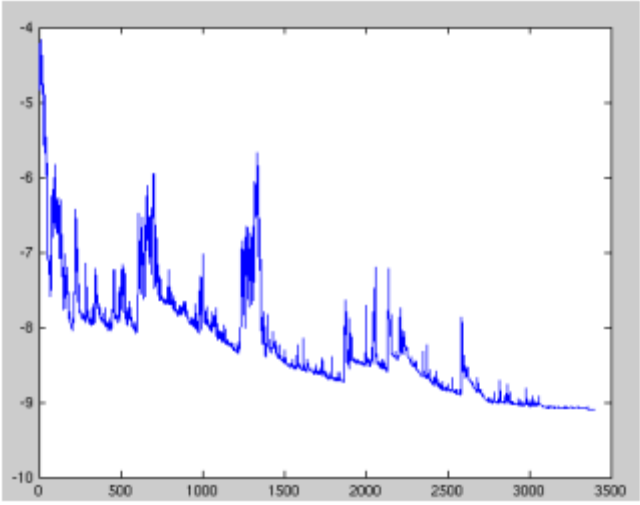
در نهایت روش گرادیان نزولی نیمه انبوه بهترین ویژگی های هردو روش را باهم ترکیب کرده و به ازای گرادیان هر زیرمجموعه n عضوی از نمونه‌ها، یک گام تغییر ایجاد می‌کند:

$$\theta = \theta - \eta \cdot \nabla_\theta J\left(\theta; x^{(i:i+n)}; y^{(i:i+n)}\right)$$

با اینکار، الف) واریانس تغییرات پارامترها کمتر می‌شود، که باعث می‌شود نرخ همگرایی سریع تر شود ب) می‌تواند از روش های بهینه سازی ماتریس که در جدیدترین کتابخانه‌های یادگیری ژرف به سادگی یافت می شوند و به شدت بهینه هستند برای محاسبه‌ی بسیار بهینه‌ی گرادیان نسبت زیرمجموعه‌ها بهره بگیرد. اندازه‌ی معمول زیرمجموعه‌ها بین ۵۰ و ۲۵۶ نمونه است، ولی می‌تواند برای کاربردهای مختلف اندازه‌های متفاوتی داشته باشد. در آموزش دادن شبکه های عصبی به طور معمول از روش گرادیان نزولی نیمه انبوه استفاده می شود و عبارت SGD معمولا برای این روش هم به کار برده می شود. نکته ی مهم: در ادامه‌ی متن در جاهایی که فرمول های مربوط به SGD آمده اند، برای سادگی پارامترهای $x^{(i:i+n)}$ و $y^{(i:i+n)}$ نوشته نشده اند.

در کدنویسی، به جای تکرار کردن حلقه برای کل داده ها، حلقه در یک زیرمجوعه با اندازه ی ۵۰ تکرار می شود:

```
for i in range ( nb_epochs ) :
    np . random . shuffle (data)
    for batch in get_batches ( data, batch_size=50 )
        params_grad = evaluate_gradient ( loss_function, batch, params )
        params = params – learning_rate * params_grad
```



شکل1 تناوب شدید روش تصادفی (منبع: ویکی پدیا)

۳ چالش ها

با وجود موارد گفته شده، روش گرادیان نزولی نیمه انبوه ساده، همگرایی سریع را تضمین نمی‌کند، و در این زمینه چالش‌هایی دارد که باید مورد توجه قرار گیرند:

- انتخاب نرخ یادگیری مناسب می تواند سخت باشد. اگر نرخ یادگیری زیادی کوچک باشد باعث می‌شود همگرایی به شدت کند باشد، و از طرف دیگر اگر زیادی بزرگ باشد برای همگرایی مزاحمت ایجاد کرده و باعث شود تابع ضرر در اطراف مقدار مینیمم نوسان کند یا حتی واگرا شود.
- روندهای نرخ یادگیری [۱۸]، در طول آموزش سیستم عصبی، سعی می کنند با روش هایی مثل الگوریتم تبرید نرخ یادگیری را اصلاح کنند. الگوریتم تبرید، نرخ یادگیری را بر اساس روند تعریف شده، یا زمانی که تغییرات تابع هدف بین دو تکرار کمتر از آستانه‌ی مشخصی می شود، کاهش می‌دهد. با اینحال روند استفاده شده و آستانه‌ی تغییرات، باید از قبل مشخص شده باشند و بنابراین نمی توان آن ها را متناسب با ویژگی های مجموعه داده ها تغییر داد. [۴]
- علاوه بر اینها، یک نرخ یادگیری ثابت برای همه‌ی تغییرات ایجاد شده در پارامترها استفاده می شود. اگر داده‌ها خیلی پراکنده باشند و ویژگی‌های مورد بررسی تعداد رخدادهای متفاوتی داشته باشند، نباید برای همه‌ی آنها را به یک اندازه تغییر دهیم، بلکه باید ویژگی هایی که به ندرت رخ داده‌اند را با گام بزرگتری تغییر دهیم.
- یکی دیگر از چالش‌های کلیدی مینیمم کردن توابع ضرر پرکاربرد در سیستم های عصبی که به شدت غیرمحدب هستند، پیشگیری از گیر کردن در مینیمم های نسبی است که نسبت به مینیمم های اصلی بهینه نیستند. داوفین و همکاران [۵] عنوان کرده اند که چالش اصلی مینیمم های نسبی نیستند بلکه نقاط زینی اند؛ یعنی نقاطی که شیب در یکی از جهت ها مثبت و در جهت دیگر منفی است. این نقاط زینی معمولا با سطحی صاف پر از نقاط مشابه احاطه شده اند که خروج از آن‌ها را برای روش SGD دشوار می‌کند زیرا گرادیان در همه‌ی جهات نزدیک صفر است.

۴ الگوریتم های بهینه سازی با رو ش گرادیان نزولی

در ادامه، مبانی اصلی برخی الگوریتم ها را که به طور گسترده ای توسط افراد فعال در حوزه‌ی یادگیری شرف برای غلبه بر چالش‌های عنوان شده استفاده می‌شوند را بیان می‌کنیم. الگوریتم‌هایی مثل روش‌های مرتبه دو ازجمله روش نیوتون که محاسبات آن‌ها در عمل برای مجموعه داده‌های چند بعدی عملی نیست را در اینجا مطرح نمی‌کنیم.[۶]

۴.۱ روش شتاب (گشتاور)

روش SGD برای حرکت در دره‌ها، یعنی نواحی که در آن فرورفتگی سطح در یک جهت بسیار بیشتر از جهت‌های دیگر است [۲۰] خوب عمل نمی کند در حالیکه که چنین نواحی در اطراف نقاط اکسترمم بسیار معمول هستند. در چنین مواردی، روش

[۶] https://en.wikipedia.org/wiki/Newton%27s_method_in_optimization

SGD بین دو شیب دره تناوب می‌کند و همانطور که در شکل ۲ الف نشان داده شده بسیار کند به سمت اکسترمم موضعی حرکت می‌کند.

(ب) SGD با استفاده از روش شتاب (الف) SGD بدون استفاده از روش شتاب

شکل ۲ منبع: جنویو ب. اور

روش شتاب [۱۷] روشی است که به الگوریتم SGD در جهت مناسب سرعت می‌بخشد و تناوب ها را میرا می کند. این عملکرد در شکل ۲ ب نشان داده شده است. روش شتاب این کار را با اضافه کردن کسری ازبردار تغییرات مرحله‌ی قبل یعنی γ به بردار تغییرات فعلی انجام می‌دهد.[۷]

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$
$$\theta = \theta - v_t$$

ضریب شتاب یعنی γ معمولا ۰/۹ یا همین حدود در نظر گرفته می‌شود.

اساسا وقتی از روش شتاب استفاده می‌کنیم، مثل این است که یک توپ را از یک تپه به پایین هل می‌دهیم. توپ حین قل خوردن به پایین تپه شتاب می‌گیرد و در راه سریع‌تر و سریع‌تر می‌شود (تا جایی که به سرعت نهایی برسد، البته در صورتی که مقاومت هوا وجود داشته باشد که به معنی γ کمتر از ۱ است). همین اتفاق برای تغییرات ایجاد شده در پارامترها هم می‌افتد: ضریب شتاب در جهت هایی که گرادیان نسبت به آنها تغیر علامت نمی دهد افزایش می‌یابد و در جهت هایی که گرادیان تغییر علامت می‌دهد کاهش می‌یابد. در نتیجه، همگرایی سریع‌تر شده و تناوب‌ها کمتر می‌شود.

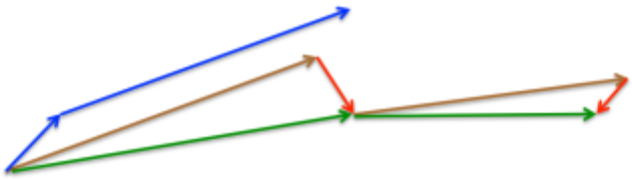
۲. ۴ گرادیان شتاب یافته‌ی نستروف

[۷] در بعضی جاها علامت ضرایب در معادله متفاوت است.

با اینحال، توپی که روی یک تپه قل می‌خورد و تنها شیب روی حرکت آن اثر می‌گذارد، چندان رضایت بخش نیست. ما می‌خواهیم توپی هوشمندتر داشته باشیم، توپی که بداند قرار است به کجا برسد داشته باشد تا قبل از اینکه دوباره به سربالایی تپه برسد سرعتش را کم کند.

روش گرادیان شتاب یافته‌ی نستروف (Nesterov Accelerated Gradient:NAG) [۱۴] روشی است که با آن می توان چنی خاصیتی به ضریب شتاب داد. می دانیم که از ضریب شتاب به صورت γv_{t-1} برای حرکت بردار پارامتر θ استفاده می‌کنیم. بنابراین محاسبه‌ی $\theta - \gamma v_{t-1}$ تقریبی از مقدار بعدی بردار پارامترها را به دست می دهد (گرادیان که در میزان تغییرات موثر است در اینجا در نظر گرفته نشده)؛ با اینکار درکی مناسب از مقادیر بعدی پارامترها به دست می آوریم. اکنون می‌توانیم با محاسبه گرادیان نسبت به مقدار تقریبی بعدی پارامترها، نه مقدار فعلی آن‌ها، به طور موثری مقدار بعدی داده‌ها را در محاسبات دخیل کنیم:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1})$$
$$\theta = \theta - v_t$$



شکل ۳ تغییرات نستروف (منبع: اسلاید 6c ج. هینتون)

باز هم به ضریب شتاب γ مقداری حدود ۰/۹ می‌دهیم. درحالیکه روش شتاب ابتدا گرادیان فعلی را حساب کرده (بردار کوچک آبی در شکل ۳) و سپس جهشی بزرگ در جهت گرادیان جدید تجمع یافته (بردار قهوه ای) می کند، روش نستروف ابتدا جهشی بزرگ در جهت گرادیان تجمع یافته‌ی قبلی می کند (بردار قهوه‌ای)، گرادیان را اندازه می‌گیرد و سپس یک اصلاح کوچک انجام می‌دهد (بردار سبز). این تغییرات ایجاد شده با استفاده از پیش بینی مقادیر بعدی، از زیادی سرعت گرفتن تغییرات جلوگیری می‌کند. این امر باعث پاسخگویی بیشتر الگوریتم می‌شود که کارایی شبکه‌هایی عصبی بازگشتی در انجام بسیاری از فعالیت‌ها را به شدت بهبود داده است. [۲][۸]

اکنون که می‌توانیم تغییرات را با شیب تابع ضرر منطبق کنیم، و لذا سرعت روش SGD را بهبود ببخشیم، بد نیست که تغییرات را با تک تک پارامترها هم انطباق دهیم تا با توجه به میزان اهمیت آنها، تغییرات بزرگتر یا کوچتری بدهیم.

---

[۸] برای توضیحات بیشتر پیرامون مفاهیم پشت روش نستروف به لینک زیر مراجعه کنید. ایلیا استاتسکور هم مروری با جزییات بیشتر در تز دکترای خود نسبت به این مفهوم ارائه کرده است. [۱۹]

http://cs231n.github.io/neural-networks-3/

۴.۳ روش گرادیان انطباقی (AdaGrad)

AdaGrad [۸] الگوریتمی برای بهینه سازی بر پایه‌ی گرادیان است که از این روش استفاده می کند: نرخ یادگیری را با پارامترها انطباق می‌دهد، به این معنی که به ازای پارامترهایی که میزان رخداد کمتری دارند تغییرات بزرگتری داده و به ازای پارامترهایی که بیشتر رخ داده اند تغییرات کمتری می‌دهد. به همین دلیل، برای محاسبات روی داده‌های پراکنده بسیار مناسب است. دین و همکاران [۶] دریافته‌اند که روش AdaGrad اطمینان پذیری روش SGD را به شدت افزایش داده است و از این روش برای آموزش دادن شبکه‌های عصبی گسترده‌ای در گوگل استفاده کردند که –علاوه بر کارهای دیگر- یاد گرفت که گربه ها را در ویدئوهای یوتیوب تشخیص دهد.[۹] بعلاوه، پنینگتون و همکاران [۱۶] از این روش برای یاد دادن تکنیک "کلمه تعبیه شده" (الگوریتمی برای تشخیص جملات توهین آمیز حتی اگر شامل کلمات توهین آمیز نباشند) به سیستم GloVe شدند زیرا در این روش کلمات کم استفاده تر باید تاثیر بیشتری نسبت به کلمات پراستفاده داشته باشند.

در روش های قبلی، ما روی همه‌ی پارامترها یک تغییر یکسان را اعمال می‌کردیم زیرا همه‌ی پارامترها از یک نرخ یادگیری یعنی η استفاده می‌کردند. از آنجایی که روش AdaGrad نرخ یادگیری متفاوتی برای هر پارامتر θ_i با هر گام زمانی t استفاده می‌کند، ابتدا تغییرات هر پارامتر مستقل در AdaGrad را نشان می‌دهیم، سپس پارامترها را برداری می‌کنیم. برای اختصار، $g_{t,i}$ را گرادیان تابع هدف نسبت به θi و در لحظه‌ی t در نظر می گیریم:

$$g_{t.i} = \nabla_{\theta_t} J(\theta_{t.i})$$

پس تغییرات ایجاد شده توسط تابع SGD بر روی هر پارامتر θ_i در هر لحظه‌ی t برابر خواهد بود با:

$$\theta_{t+1.i} = \theta_{t.i} - \frac{\eta}{\sqrt{G_{t.ii} + \epsilon}} \cdot g_{t.i}$$

در اینجا $\mathbf{G_t \in \mathbb{R}^{d \times d}}$ یک ماتریس قطری است که در آن هر درایه‌ی قطری i, i جمع مربعات گرادیان‌ها نسبت به θ_i در گام زمانی t است[۱۰]، و ϵ یک ضریب هموارسازی است که از ایجاد صفر در مخرج جلوگیری می‌کند (معمولا در مقیاس 10^{-8} است). جالب است که بدون جذر گرفتن، الگوریتم عملکرد بسیار ضعیف تری دارد.

از آنجایی که قطر اصلی ماتریس G_t شامل جمع مربعات گرادیان‌های قبلی نسبت به همه‌ی پارامترها یعنی بردار θ است، می‌توانیم فرمول را با یک ضرب درایه به درایه‌ی ماتریس و بردار $\odot$ بین G_t و g_t تبدیل به یک فرمول برداری کنیم:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t$$

---

[۹] http://www.wired.com/2012/06/google-x-neural-network/

[۱۰] دوچی و همکاران [۸] این ماتریس را به عنوان جایگزینی برای ماتریس *کامل* شامل ضرب خارجی تمام گرادیان‌های قبلی ارائه می کند زیرا در غیر این صورت محاسبه ی ماتریس جذر حتی با تعداد کم پارامترها یعنی d هم عملی نیست.

یکی از مزایای اصلی روش AdaGrad این است که نیاز به تعیین نرخ یادگیری را از بین می برد. در بیشتر کاربردها مقدار ۰/۰۱ انتخاب شده و تغییر داده نمی‌شود.

ضعف اصلی روش AdaGrad وجود ماتریس تجمع مربعات گرادیان‌ها در مخرج است: از آنجایی که هر عبارت اضافه شده مثبت است، حاصل جمع همواره در طول آموزش سیستم عصبی افزایش می‌یابد. این امر به نوبه‌ی خود باعث می‌شود که نرخ یادگیری کاهش یابد و درنهایت، بی نهایت کوچک شود که پس از آن الگوریتم نمی‌تواند چیز جدیدی یاد بگیرد. الگوریتم‌های زیر به هدف از بین بردن این نقص به وجود آمده‌اند.

۴.۴ AdaDelta

روش AdaDelta [۲۲] بسطی از روش AdaGrad است که هدف آن کمتر کردن کاهش فزاینده و یکنواخت نزخ یادگیری در این روش است. به جای تجمع مربعات همه‌ی گرادیان‌های قبلی، روش AdaDelta تعداد مربعات جمع شده را به تعداد محدودی مثل wمحدود می کند.

به جای ذخیره‌ی غیربهینه‌ی wتا از مربعات گرادیان‌ها، جمع گرادیان به صورت بازگشتی، میانگین میراشونده‌ی همه‌ی مربعات گرادیان‌های قبلی تعریف می‌شود. میانگین در لحظه‌ی t یعنی $E[g^2]_t$ فقط (با یک کسر γ که شبیه به ثابت شتاب است) به میانگین قبلی و گرادیان فعلی بستگی دارد:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) g_t^2$$

به γ مقداری شبیه به ثابت سرعت یعنی حدود ۰/۹ می‌دهیم. برای شفاف‌تر شدن موضوع، میزان تغییرات پارامترها روش SGD پایه را بر اساس بردار تغییرات یعنی $\Delta\theta_t$ می نویسیم:

$$\Delta\theta_t = -\eta \cdot g_{t.i}$$
$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

پس بردار تغییرات روش AdaGrad که قبلا به دست آوردیم برابر است با:

$$\Delta\theta_t = -\frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t$$

اکنون به سادگی در این فرمول ماتریس قطری G_t را با میانگین میراشونده‌ی مربعات گرادیان‌های قبلی یعنی $E[g^2]_t$ جایگزین می کنیم:

$$\Delta\theta_t = -\frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

از آنجایی که مخرج همان معیار خطای جذر میانگین مربعات گرادیان است، می توانیم آن را با علامت اختصاری این معیار جایگزین کنیم:

$$\Delta\theta_t = -\frac{\eta}{RMS[g]_t} g_t$$

سازندگان این روش متوجه شدند که واحد بردار تغییرات (همانطور که در روش‌های SGD، روش شتاب و AdaGrad هم وضع به همین صورت است) با پارامترها همخوانی ندارد؛ چرا که تغییرات باید واحد فرضی یکسانی با پارامترها داشته باشند. برای حل این مشکل، ابتدا میانگین دیگری تعریف کردند که آن هم به صورت نمایی میراشونده بود. اما این‌بار از میانگین مربعات استفاده نکردند بلکه از میانگین تغییرات پارامترها استفاده کردند:

$$E[\Delta\theta^2]_t = \gamma E\left[\Delta\theta^2\right]_{t-1} + (1-\gamma)\Delta\theta_t^2$$

بنابراین خطای جذر میانگین مربعات تغییرات پارامترها به صورت زیر است:

$$RMS\ [\Delta\theta]_t = \sqrt{E[\Delta\theta^2]_t + \epsilon}$$

از آنجایی که $RMS[\Delta\theta]_t$ معلوم نیست، آن را با مقدار RMS تغییرات پارامترها تا گام زمانی قبلی، تقریب می‌زنیم. با جایگزین کردن نرخ یادگیری η در فرمول تغییرات پارامتر قبلی با $RMS[\Delta\theta]_{t-1}$ در نهایت فرمول تغییرات برای روش AdaDelta به دست می آید:

$$\Delta\theta_t = -\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} g_t$$
$$\theta_{t+1} = \theta_t + \Delta\theta_t$$

با روی AdaDelta، حتی نیاز نیست نرخ یادگیری پیشفرض را تعیین کنیم، زیرا از فرمول تغییرات حذف شده است.

۴.۵ RMSprop

روش RMSprop یک روش بر مبنای نرخ یادگیری انطباق یافته است که در جایی منتشر نشده و جاف هینتون در درس‌هایی که در وبسایت Coursera ارائه کرده آن را معرفی کرده است.[۱۱]

[۱۱] http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

هم RMSprop و هم AdaDelta به طور مستقل و تقریبا در یک زمان در پی نیاز به راه حلی برای حل مشکل کاهش شدید نرخ یادگیری در روش AdaGrad به وجود آمدند. روش RMSprop در واقع کاملا مشابه اولین بردار تغییرات روش AdaDelta که در بالا به دست آوردیم است:

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

روش کسر RMS (RMSprop) نیز نرخ یادگیری را بر میانگین نمایی میراشونده‌ی مربعات گرادیان‌ها تقسیم می‌کند. هینتون پیشنهاد کرده است که مقدار γ برابر ۰/۹ باشد و یک مقدار پیشفرض مناسب برای نرخ یادگیری یا η برابر ۰/۰۰۱ است.

۴.۶ Adam

روش تخمین تطبیق پذیر گشتاور (Adam) [۱۰] یکی دیگر از روش‌هایی است که نرخ یادگیری را منطبق بر داده‌ها محاسبه می کند. علاوه بر ذخیره کردن میانگین میراشونده‌ی نمایی مربعات گرادیان‌های قبلی، یعنی v_t مثل روش‌های AdaDelta و RMSprop، روش Adam میانگین میراشونده‌ی نمایی گرادیان ها m_t را هم مثل روش شتاب حفظ می کند:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2$$

m_t و v_t به ترتیب تخمین گشتاور اول (میانگین) و گشتاور دوم (واریانس غیرمرکزی) گرادیان‌ها هستند که نام‌گذاری روش هم به همین خاطر است. به خاطر این که حالت اولیه ی بردارهای m_t و v_t صفر است، سازندگان روش مشاهده کردند که نتایج به صفر متمایل می‌شوند، به خصوص در گام‌های اولیه و مخصوصاً وقتی نرخ میراشوندگی کوچک است (یا به عبارت دیگر β_1 و β_2 نزدیک ۱ هستند).

آنها این مشکل را با تخمین‌های اصلاح شده‌ی گشتاور اول و دوم حل کردند:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$
$$\hat{v}_t = \frac{v_t}{1-\beta_2^2}$$

سپس از این دو فرمول همانطور که در روش های AdaDelta و RMSprop هم دیدیم، برای محاسبه‌ی تغییرات پارامترها استفاده کردند که فرمول تغییرات را برای این روش به دست می‌دهد:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t$$

سازندگان روش مقادیر پیشفرض ۰/۹ برای β_1، ۰/۹۹۹ برای β_2 و 10^{-8} را برای ϵ پیشنهاد کرده اند. آنها به صورت تجربی نشان داده اند که روش Adam در عمل به خوبی کار می‌کند و نسبت به دیگر روش های یادگیری انطباق پذیر ارجحیت دارد.

۴.۷ AdaMax

ضریب v_t در فرمول تغییرات روش Adam گرادیان را با نسبت معکوس نسبت به میانگین گرادیان‌های قبلی یعنی l_2 (با ضریب v_{t-1}) و گرادیان فعلی مربوط می کند.

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)|g_t|^2$$

می‌توانیم این تغییر را به میانگین l_p هم تعمیم دهیم. دقت کنید که کینگما و با، ضریب β_2 را هم به صورت پارامتری یعنی β_2^p نوشته‌اند:

$$v_t = \beta_2^p v_{t-1} + \left(1 - \beta_2^p\right)|g_t|^p$$

میانگین برای مقادیر بزرگ p معمولا از لحاظ عددی ناپایدار می‌شود. به همین خاطر است که در عمل میانگین های l_1 و l_2 پرکاربردتر هستند. با این وجود، l_∞ نیز معمولا از خود پایداری نشان می‌دهد. به همین دلیل، سازندگان روش AdaMax را پیشنهاد می‌کنند [۱۰] و نشان می‌دهند که v_t نیز مانند l_∞ به مقدار پایدارتر زیر میل می‌کند. برای جلوگیری از اشتباه گرفتن متغیرها با روش Adam، ما برای نشان دادن حد در بینهایت v_t از عبارت u_t استفاده کرده‌ایم:

$$\begin{aligned} u_t &= \beta_2^\infty v_{t-1} + (1 - \beta_2^\infty)|g_t|^\infty \\ &= \max(\beta_2 \cdot v_{t-1}, |g_t|) \end{aligned}$$

اکنون می توانیم با جایگزین کردن $\sqrt{\hat{v}_t} + \epsilon$ با u_t این معادله را با معادله‌ی تغییرات در روش آدام ادغام کنیم تا فرمول تغییرات پارامترها برای روش AdaMax به دست آید:

$$\theta_{t+1} = \theta_t - \frac{\eta}{u_t}\hat{m}_t$$

دقت کنید از آنجایی که u_t حاصل از یک عملیات ماکسیمم‌گیری است، مثل m_t و v_t در روش Adam به صفر همگرا نمی‌شود و نیازی به محاسبه‌ی ضریب تصحیح برای جلوگیری از این امر نیست. مقادیر پیشفرض مناسب برای این روش هم عبارتند از: ۰/۰۰۲ برای η، مقدار ۰/۹ برای β_1 و مقدار ۰/۹۹۹ برای β_2.

۴.۸ Nadam

همانطور که قبلا هم دیدیم، روش Adam را می‌توان ترکیبی از روش RMSprop و روش شتاب دانست: میانگین میراشونده‌ی نمایی مربعات گرایان های قبلی یعنی v_t از روش RMSprop و میانگین میراشونده‌ی نمایی میانگین گرادیان‌های قبلی یعنی m_t . هم چنین دیدیم که روش گرادیان شتاب یافته‌ی نستروف نیز به روش شتاب معمولی ارجحیت دارد.

روش Nadam (Nesterov-accelerated Adaptive Moment Estimation) [۷] روش Adam و نستروف را با هم ترکیب می‌کند. برای استفاده از روش نستروف در الگوریتم Adam، باید عبارت m_t را اصلاح کنیم.

ابتدا، بیایید فرمول تغییرات در روش شتاب را با ضرایب جدید بنویسیم:

$$g_t = \nabla_{\theta_t} J(\theta_t)$$
$$m_t = \gamma m_{t-1} + \eta g_t$$
$$\theta_{t+1} = \theta_t - m_t$$

که در آن J همان تابع هدف، γ ثابت میرایی شتاب و η اندازه‌ی گام است. با ادغام معادله‌ی دم در معادله‌ی سوم داریم:

$$\theta_{t+1} = \theta_t - (\gamma m_{t-1} + \eta g_t)$$

این فرمول نیز نشان می‌دهد که در روش شتاب یک گام در جهت بردار شتاب قبلی و سپس یک یک گام در جهت گرادیان فعلی برداشته می‌شود.

روش نستروف به ما اجازه می‌دهد که با تغییر دادن پارامترها و گشتاورها *پیش از* محاسبه‌ی گرادیان گامی دقیق‌تر در جهت گرادیان برداریم. برای به کارگیری روش نستروف فقط باید گرادیان g_t را اصلاح کنیم:

$$g_t = \nabla_{\theta_t} J(\theta_t - \gamma m_{t-1})$$
$$m_t = \gamma m_{t-1} + \eta g_t$$
$$\theta_{t+1} = \theta_t - m_t$$

دوزارت پیشنهاد کرده است که روش نستروف به این صورت اصلاح شود: به جای دوبار دخیل کردن گشتاور در محاسبات – یک بار برای به تغییر دادن گرادیان g_t و یک بار برای تغییر دادن پارامترها یعنی بردار θ_{t+1} – مستیقما از تخمین بردار گشتاور برای محاسبه‌ی تغییرات پارامترها استفاده می کنیم:

$$g_t = \nabla_{\theta_t} J(\theta_t)$$
$$m_t = \gamma m_{t-1} + \eta g_t$$
$$\theta_{t+1} = \theta_t - (\gamma m_t + \eta g_t)$$

دقت کنید که به جای استفاده از بردار گشتاور قبلی یعنی m_{t-1} در فرمول تغییرات قبلی، اکنون از بردار گشتاور m_t یعنی بردار گشتاور فعلی برای رسیدن به مقادیر بعدی استفاده می کنیم. به طور مشابه برای اضافه کردن روش شتاب نستروف به الگوریتم Adam، می توانیم بردار گشتاور قبلی را با بردار گشتاور فعلی جایگزین کنیم. ابتدا، بیایید فرمول تغییرات روش Adam را به یاد بیاوریم (دقت کنید که نیاز داریم $\hat{v}_t$ را اصلاح کنیم):

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t$$
$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t$$

با جایگذاری کردن فرمول $\hat{m}_t$ و m_t در فرمول آخر داریم:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \left(\frac{\beta_1 m_{t-1}}{1-\beta_1^t} + \frac{(1-\beta_1) g_t}{1-\beta_1^t} \right)$$

توجه داشته باشید که $\frac{\beta_1 m_{t-1}}{1-\beta_1^t}$ تنها یک تخمین اصلاح‌شده از بردار گشتاور در گام زمانی قبلی است. لذا می توانیم آن را با $\hat{m}_{t-1}$ جایگزین کنیم:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \left(\beta_1 \hat{m}_{t-1} + \frac{(1-\beta_1) g_t}{1-\beta_1^t} \right)$$

۴.۹ تجسم تصویری الگوریتم‌ها

دو شکل زیر درکی از عملکرد بهینه‌سازی الگوریتم‌های معرفی شده ارائه می‌کند. [۱۲]

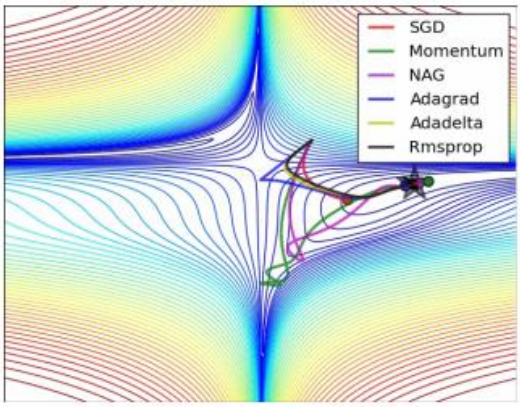
در شکل ۴ الف، مسیری الگوریتم ها بر روی نمای دو بعدی سطح ضرر (تابع Beale) نشان داده شده است. همه‌ی الگوریتم‌ها از یک نقطه شروع کرده‌اند و مسیرهای متفاوتی را برای رسیدن به مینیمم در پیش گرفته اند. توجه داشته باشید که روش AdaGrad، AdaDelta و RMSprop سریعا به جهت درست رفته اند و نسبتا سریع همگرا شده‌اند در حالیکه روش های شتاب و نستروف از مسیر اصلی منحرف شده‌اند که حرکت یک توپ در دامنه‌ی یک تپه را تداعی می‌کند. با اینحال روش نستروف

[۱۲] به لینک زیر هم نگاهی بیندازید. در این لینک کارپاثی توضیحات دیگری برای همین تصاویر و مرور مختصر دیگری بر الگوریتم های ذکر شده ارائه کرده است.

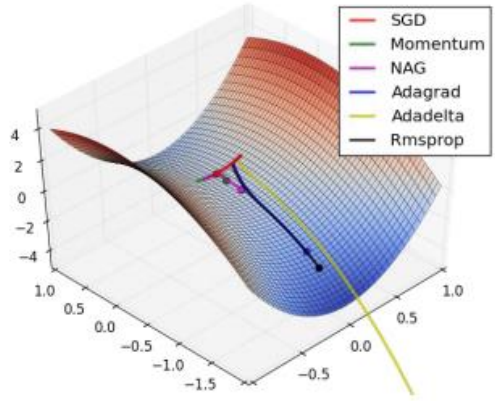
http://cs231n.github.io/neural-networks-3/

قادر بوده است که مسیرش را به خاطر حساسیت بیشتر نسبت به مسیر و استفاده از اطلاعات نقطه‌ی بعدی، قادر بوده است که سریع‌تر مسیرش را اصلاح کند.

شکل ۴ ب رفتار الگوریتم ها را در نقطه‌ی زینی نشان می‌دهد، یعنی نقطه‌ای که شیب در یک جهت مثبت و در جهت دیگر منفی است. این نقطه برای روش SGD همانطور که قبلا اشاره کردیم، دشواری هایی ایجاد کرده است. توجه کنید که روش های SGD، شتاب و نستروف برای خارج شدن از سطح متقارن اطراف این نقطه با دشواری هستند، اگرچه دو الگوریتم آخر توانسته‌اند از نقطه‌ی زینی خارج شوند، در حالیکه AdaGrad، RMSprop و AdaDelta سریعا به سمت شیب منفی سرازیر شده‌اند، و روش AdaDelta در بین این روش ها پیشگام بوده است.



(الف) نمای دو بعدی بهینه سازی SGD بر روی سطح ضرر



(ب) بهینه سازی SGD در نقطه‌ی زینی

شکل ۴ منبع و انیمیشن های کامل: الک ردفورد

همانطور که دیده می‌شود، روش‌های با نرخ یادگیری انطباق یافته، یعنی AdaGrad، AdaDelta، RMSprop و Adam از همه مناسب‌تر هستند و در این مثال‌ها بهترین همگرایی را از خود نشان می‌دهند.

۴.۱۰ کدام روش بهتر است؟

با این اوصاف، باید از کدام روش بهینه‌سازی استفاده کرد؟ اگر داده‌ی ورودی پراکنده است، پس احتمالا بهترین نتایج با استفاده از یکی از روش‌های با نرخ یادگیری انطباق یافته به دست می‌آورید. یکی از مزیت‌های اضافی این روش‌ها این است که دیگر نیاز نیست نرخ یادگیری را تنظیم کنید ولی احتمالا بهترین نتایج را با مقادیر پیش‌فرض به دست خواهید آورد.

به طور خلاصه، روش RMSprop تعمیمی از روش AdaGrad است که کاهش شدید نرخ یادگیری در این روش را حل کرده است. این روش کاملا شبیه به روش AdaDelta است، با این تفاوت که روش AdaDelta از RMS تغییرات پارامترها در صورت فرمول تغییرات استفاده کرده است. در نهایت، روش Adam روشی برای تصحیح میل به صفر و گشتاور را به روش RMSprop اضافه کرده است. تا اینجای کار، RMSprop، AdaDelta و Adam الگوریتم های بسیار مشابهی هستند

که در شرایط مشابه همگی به خوبی کار می‌کنند. کینگما و همکاران [۱۰] نشان داده‌اند که تصحیح میل به صفر در الگوریتم Adam باعث شده که کارایی آن در پایان بهینه سازی که گرادیان ها پراکنده تر می‌شوند کمی از RMSprop بیشتر باشد. پس تا به حال، روش Adam بهترین انتخاب در بین همه‌ی الگوریتم‌هاست.

نکته‌ی جالب اینجاست که بیشتر مقالات اخیر از روش SGD معمولی بدون روش شتاب و یک روند تبرید نرخ یادگیری ساده استفاده می‌کنند. همانطور که نشان داده شد، روش SGD معمولا در یافتن یک مینیمم موفق است، ولی ممکن است به طور قابل ملاحظه‌ای بیش از سایر الگوریتم‌ها طول بکشد، به مقادیر اولیه‌ای که برای داده‌های مختلف کارایی را تضمین کنند وابسته‌تر است و ممکن است به جای رسیدن به مینیمم نسبی در نقطه‌ی زینی گیر بیفتد. درنتیجه، اگر همگرایی سریع برایتان مهم است یا شبکه عصبی عمیق یا پیچیده‌ای را آموزش می‌دهید، باید از یکی روش‌های با نرخ یادگیری انطباق یافته استفاده کنید.

اجرای SGD به صورت موازی و توزیع شده

با فراگیری راه حل های بر پایه‌ی داده های بزرگ مقیاس برای چالش های صنعتی و ظهور خوشه‌های پردازش ابری با تعداد کاربر کم، توزیع الگوریتم SGD به منظور افزایش سرعت آن رویکرد مسلمی است. روش SGD به خودی خود ذاتاً ترتیبی است: قدم به قدم، به سمت مینیمم حرکت می کنیم. استفاده از این الگوریتم همگرایی خوبی فراهم می‌کند ولی می‌تواند به خصوص برای مجموعه داده‌های بزرگ کند باشد. در عوض، اجرای غیرهمزمان (آسنکرون) SGD سریع‌تر است، ولی ارتباط غیر بهینه بین اجزای اجرای کننده‌ی الگوریتم می‌تواند باعث همگرایی ضعیف شود. به علاوه، می‌توانیم SGD را تنها بر روی یک دستگاه بدون نیاز به خوشه‌های پردازشی بزرگ هم به صورت موازی اجرا کنیم. در زیر الگوریتم‌ها و معماری‌هایی که برای اجرای موازی و توزیع‌شده‌ی SGD پیشنهاد شده‌اند آمده‌اند.

۵.۱ HOGWILD!

نیو و همکاران [۱۵] روندی برای ایجاد تغییرات به اسم HOGWILD! را معرفی کرده‌اند که اجرای تغییرات SGD به صورت موازی در پردازنده را امکان‌پذیر می‌کند. هسته‌های پردازشی مجازند بدون ثابت نگه داشتن پارامترها به حافظه‌ی اشتراکی دسترسی داشته باشند. این روش تنها در صورتیکه کارآمد است که داده‌های ورودی پراکنده باشند، زیرا هر تغییر تنها بر روی کسری از داده‌ها پیاده می‌شود. آن‌ها نشان دادند که در این صورت، روند تغییرات تقریبا به نرخ همگرایی بهینه‌ای می‌رسد زیرا احتمال پاک کردن و بازنویسی داده‌های مفید توسط پردازنده‌ها بسیار پایین است.

۵.۲ روش SGD بارشی

روش SGD بارشی یک فُرم غیرهمزمان (آسنکرون) از روش SGD است که توسط دین و همکاران [۶] در فریم‌ورک DistBelief (که TensorFlow از روی آن ساخته شده است) در گوگل از آن استفاده شده است. این الگوریتم چندین کپی از مدل را بر روی زیرمجموعه‌های داده‌های آموزش سیستم به صورت موازی اجرا می‌کند. این مدل‌ها تغییرات خود را به سرور پارامترها ارسال می‌کنند، که بین دستگاه‌های زیادی پخش شده است. هر دستگاه مسئول ذخیره و تغییر دادن کسری از پارامترهای

مدل است. با اینحال، چون کپی های مدل با یکدیگر ارتباط برقرار نمی کنند؛ یعنی مثلاً وزن داده‌ها یا تغییرات را به اشتراک نمی‌گذارند، پارامترهای آن‌ها همواره در معرض خطر واگرایی یا همگرایی کند هستند.

۵.۳ الگوریتم‌های مقاوم در برابر تاخیر برای SGD

مک‌ماهان و استریتر [۱۲] الگوریتم AdaGrad را با توسعه‌ی الگوریتم‌های مقاوم در برابر تاخیر، به سیستم‌های موازی تعمیم دادند. این الگوریتم‌ها نه تنها بر گرادیان‌های قبلی، بلکه با تاخیر در تغییرات هم انطباق می‌یابند. این روش عملکرد خوبی را در عمل نشان داده است.

۵.۴ TensorFlow

TensorFlow[۱۳]، [۲۳] فریم ورک گوگل برای طراحی و پیاده سازی مدل های بزرگ مقیاس یادگیری ماشین است که اخیرا برای استفاده‌ی عموم آزاد شده است. این فریم ورک بر اساس تجربه‌ی گوگل با DistBelief است هم اکنون به صورت داخلی در بسیاری از دستگاه‌های همراه برای انجام پردازش‌های مختلف، هم چنین در سیستم‌های توزیع شده‌ی بزرگ‌مقیاس استفاده می‌شود. نسخه‌ی توزیع‌یافته، که در آوریل ۲۰۱۶ منتشر شده است[۱۴]، به یک گراف محاسباتی وابسته است که به چند زیرگراف در دستگاه‌ها تقسیم شده است، و ارتباط بین دستگاه‌ها با جفت گره‌های ارسال/دریافت برقرار می‌شود.

۵.۵ SGD با میانگین‌گیری فنری

ژانگ و همکاران [۲۳] روش SGD با میانگین‌گیری فنری (Elastic Averaging SGD:EASGD) را پیشنهاد کرده اند، که پارامترهای دستگاه‌های دخیل در SGD غیرهمزمان (آسنکرون) را به یک نیروی فنری مرتبط می کند، به این معنا که همان متغیر مرکزی ذخیره شده در سرور پارامترهاست. این به متغیرهای محلی (Local) اجازه می دهد که فراتر از متغیر مرکزی تناوب کنند، که روی کاغذ جستجوی بیشتر در فضای برداری پارامترها را ممکن می‌کند. آنها به صورت تجربی نشان دادند که این افزایش امکان جستجو به یافتن اسکترمم‌های نسبی جدید و درنتیجه عملکرد بهتر منتهی می‌شود.

۶ رویکردهای بیشتر برای بهینه‌سازی SGD

درنهایت، ما رویکردهای بیشتری معرفی می‌کنیم که می‌توانند در کنار همه‌ی الگوریتم‌هایی که قبلاً نام برده شدند استفاده شوند تا عملکرد SGD را بهبود دهند. برای مروری عالی بر روش‌های معمول دیگر، به مرجع [۱۱] مراجعه کنید.

۶.۱ یادگیری تصادفی و ترتیبی

[۱۳] https://www.tensorflow.org/

[۱۴] http://googleresearch.blogspot.ie/2016/04/announcing-tensorflow-08-now-with.htm

ما عموماً ترجیح می‌دهیم که نمونه‌های آموزشی را با یک ترتیب معنادار به مدل ندهیم زیرا این امر ممکن است در الگوریتم بهینه‌سازی اختلال ایجاد کند. لذا معمولاً توصیه می‌شود که پس از هر تکرار، چینش داده‌ها به صورت تصادفی تغییر کند.

از طرف دیگر، برای بعضی موارد که می‌خواهیم مسائلی که به طور فزاینده دشوارند را حل کنیم، دادن نمونه‌های آموزشی با یک ترتیب معنادار به سیستم، ممکن است واقعا به بهبود عملکرد و همگرایی منجر شود. روش ایجاد این ترتیب معنادار یادگیری ترتیبی (Curriculum Learning) نامیده می‌شود.

زارمبا و ساتسکور [۲۱] برای ارزیابی یادگیری ترتیبی در برنامه‌های ساده تنها توانستند حافظه‌های طولانی کوتاه مدت (LSTM) را با این روش آموزش دهند. آنها نشان دادند که رویکردی ترکیبی که نمونه‌ها را با افزایش دشواری مساله مرتب می‌کند، نسبت به رویکرد خام بهتر است.

۶.۲ نرمالیزاسیون بچ

برای تسهل یادگیری، معمولا مقادیر اولیه‌ی پارامترها را با میانگین صفر و واریانس واحد نرمال می‌کنیم. همینطور که آموزش شبکه عصبی جلوتر می‌رود و پارامترها را به مقادیر جدیدی تغییر می‌دهیم، نرمال‌سازی انجام شده از دست می‌رود که باعث کند شدن یادگیری و شدت یافتن تغییرات با عمیق‌تر شدن شبکه می‌شود.

نرمالیزاسیون بچ [۹] هر دسته از داده ها دوباره نرمال‌سازی کرده و تغییرات را نیز به عقب بازمی‌گرداند. با تبدیل نرمال‌سازی به بخشی از معماری مدل، می‌توانیم از نرخ‌های یادگیری بزرگتری استفاده کنیم و کمتر درگیر مقادیر اولیه‌ی پارامترها شویم. نرمالیزاسیون بچ هم چنین مثل یک رگیولایزر عمل می کند و نیاز به دراپ اوت (Dropout) را کم می‌کند (یا به کلی ازبین می‌برد).

۶.۳ توقف پیش از موعد

بر اساس گفته‌های جاف هینتون، " توقف پیش از موعد، یک رایگان و زیبا است"[۱۵] بنابراین باید همیشه باید خطا را با یک مجموعه داده‌ی صحت سنجی در طول آموزش بسنجید و (با به خرج دادن کمی صبر) اگر خطا به اندازه‌ی کافی زیاد نشد، آموزش را متوقف کنید.

۶.۴ نویز گرادیان

[۱۵] اسلایدهای آموزشی NIPS سال ۲۰۱۵، اسلاید ۶۳، قابل دسترسی در لینک زیر:
http://www.iro.umontreal.ca/~bengioy/talks/DL-Tutorial-NIPS2015.pdf

نیلاکاتان و همکاران [۱۳] به هر تغییرات گرادیان، نویز با توزیع گاوسیِ $N(0.\sigma_t^2)$ اضافه کردند:

$$g_{t.i} = g_{t.i} + N(0.\sigma_t^2)$$

آنها از روند تبرید زیر برای واریانس استفاده کردند:

$$\sigma_t^2 = \frac{\eta}{(1+t)^\gamma}$$

آنها نشان داده اند که اضافه کردن این نویز شبکه‌ها را نسبت به مقادیر اولیه نامناسب مقاوم‌تر کرده و به خصوص به آموزش شبکه‌های عمیق و پیچیده کمک می‌کند. آن‌ها حدس زده‌اند که نویز به مدل برای فرار و یافتن مینیمم های نسبی جدید، که در مدل‌های عمیق‌تر تعدادشان بسیار بیشتر است، شانس بیشتری می‌دهد.

۷ جمع‌بندی

در این مقاله، ابتدا سه فرم از روش گرادیان نزولی را مرور کردیم، که از بین آنها روش گرادیان نزولی نیمه انبوه از همه محبوب‌تر است. الگوریتم‌هایی را بررسی کردیم که برای بهینه‌سازی SGD بیشترین استفاده را دارند: روش شتاب، گرادیان شتاب‌یافته‌ی نستروف، AdaGrad، AdaDelta، RMSprop، Adam، AdaMax، Nadam، و همچنین الگوریتم‌های متفاوتی برای بهینه سازی SGD غیرهمزمان (آسنکرون). درنهایت، رویکردهای دیگری برای بهبود روش SGD مثل یادگیری تصادفی و ترتیبی، نرمالیزاسیون بچ تو توقف پیش از موعد را نیز بررسی کردیم.