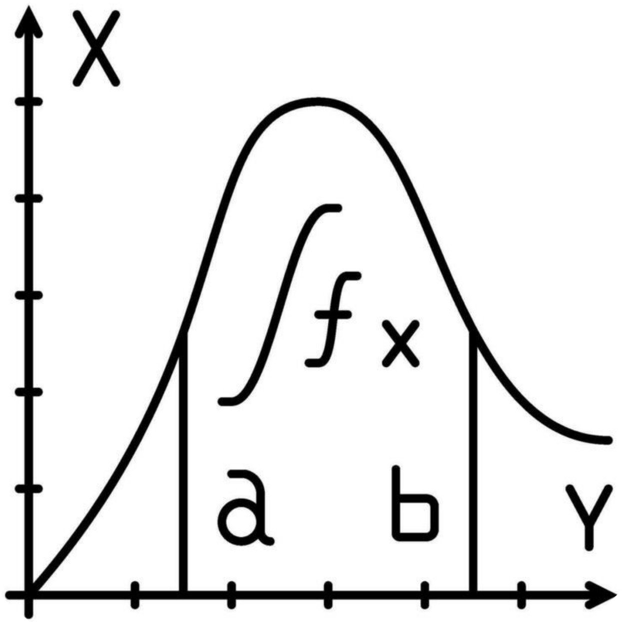
مبانی بهینه‌سازی

هادی عاشری



مقدمه

- هر مدل یادگیری ماشین نیازمند کمینه‌سازی تابع هزینه است.
- مشتق و گرادیان ابزاری اساسی برای یافتن نقاط کمینه هستند.
- درک عمیق این مفاهیم باعث تسلط بهتر بر الگوریتم‌ها و ساخت مدل‌های دقیق‌تر می‌شود.

تابع هزینه

- معیاری است برای سنجش «اشتباه» پیش‌بینی‌های مدل. وقتی مدل پاسخی می‌دهد که با حقیقت فاصله دارد، امتیاز اشتباه (هزینه) افزایش می‌یابد. هدف کمینه کردن مجموع یا میانگین همین امتیازهاست.
- مثل معلمی که هر پاسخ نادرست را با خط خوردن یا کاهش نمره جریمه می‌کند. تابع هزینه با نمره پایین، به ما می‌گوید کدام پارامترها یا وزن‌ها باید تغییر کنند تا خطا کمتر شود.
- بدون تابع هزینه نمی‌دانیم مدل چقدر خوب یا بد عمل کرده. این معیار مسیر بهینه را به ما نشان می‌دهد تا با الگوریتم‌هایی مثل گرادیان کاهشی، مدل را به سمت کمترین خطا هدایت کنیم

Iran Annual AI Award

مشتق

- مشتق را می‌توان به‌عنوان سرعت لحظه‌ای تغییر یک کمیت تصور کرد.
- درست همان‌طور که سرعت خودرو نشان می‌دهد چقدر سریع فاصله طی می‌شود، مشتق نشان می‌دهد در یک نقطه خاص، تابع با چه نرخی بالا یا پایین می‌رود.
- تصور کنید روی سطح یک تپه ایستاده‌اید. خط مماس بر نقطه‌ای که روی زمین ایستاده‌اید جهت حرکت شیب‌دار آن نقطه را نشان می‌دهد. زاویه و تندی این خط مماس همان مشتق تابع در آن نقطه است.

Iran Annual AI Award

کاربرد مشتق در یادگیری ماشین

- هنگام آموزش مدل، تابع هزینه به پارامترها وابسته است.
- مشتق تابع هزینه نسبت به هر پارامتر به ما می‌گوید با چه شدتی باید آن پارامتر را تغییر دهیم تا خطا کاهش یابد.
- الگوریتم گرادیان کاهشی از همین اطلاعات سرعت (مشتق) استفاده می‌کند تا مدام پارامترها را به‌سمت بهینه حرکت دهد.

Iran Annual AI Award

تعریف مشتق

تعریف

$$f'(x) = lim_{h\to 0}\frac{f(x+h)-f(x)}{h}$$

مثال

$$f(x) = x^3 \Rightarrow 3x^2$$

Iran Annual AI Award

قوانین مشتق‌گیری

- جمع

$$(f+g)' = f' + g'$$

- ضرب

$$(f.g)' = f'.g + f.g'$$

- قاعده زنجیره‌ای

$$(f(g(x)))' = f'\big(g(x)\big)g'(x)$$

مشتق چندمتغیره و جزئی

- مشتق جزیی $\frac{\partial f}{\partial x_i}$
- مثال

$$f(x,y) = x^2 y + \sin(y)$$

$$\frac{\partial f}{\partial x} = 2xy, \qquad \frac{\partial f}{\partial y} = x^2 + \cos(y)$$

- asd

گرادیان

- تعریف: برداری از مشتقات جزئی

$$f=f(x_1,x_2,\dots,x_n)\Rightarrow \nabla f=\left[\frac{\partial f}{\partial x_1},\frac{\partial f}{\partial x_2},\dots,\frac{\partial f}{\partial x_n}\right]$$

- با استفاده از آن می‌توان جهت بیشترین صعود تابع را مشخص نمود.
- بردار گرادیان را می‌توان به‌صورت شهودی به‌عنوان «قطب‌نما تغییرات» یک تابع چندمتغیره تصور کرد
- جهتی که اگر در آن حرکت کنیم، بیشترین افزایش مقدار تابع را تجربه خواهیم کرد.
- بردار گرادیان دقیقاً همین جهت را نشان می‌دهد: جهت و شدت بیشترین افزایش ارتفاع.
- اگر بخواهید به پایین‌ترین نقطه برسید، کافی‌ست در جهت «منفی گرادیان» حرکت کنید—یعنی همان چیزی که الگوریتم گرادیان کاهشی انجام می‌دهد.

Iran Annual AI Award

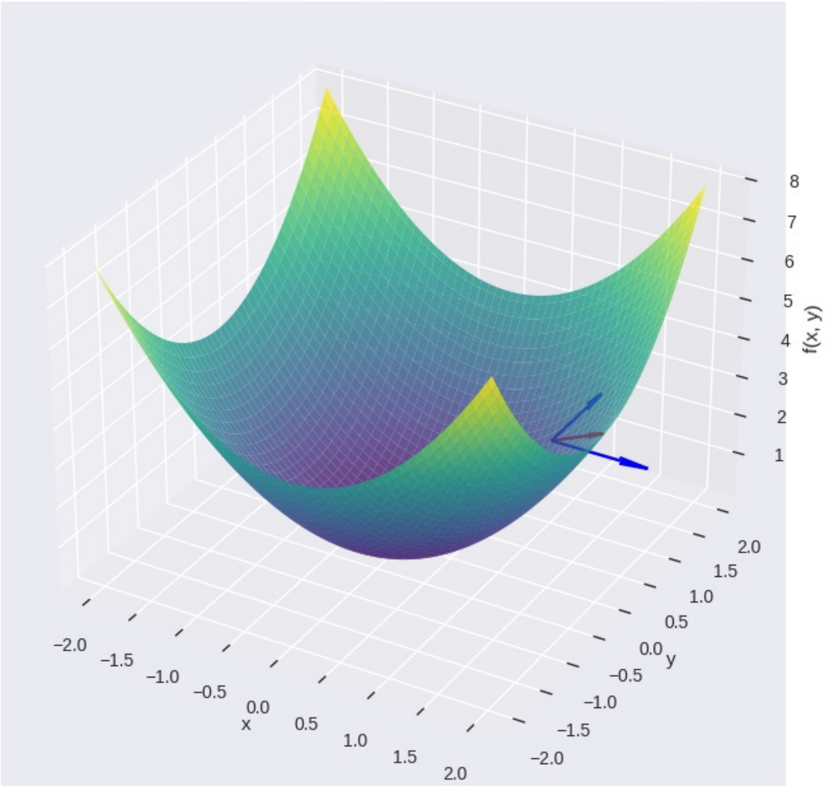
هسیان

- تعریف: ماتریس مشتقات مرتبه دوم $\frac{\partial f}{\partial x_i \partial x_j}$
- هسین یک ماتریس مربعی است که هر عنصر آن نشان‌دهندهٔ میزان خمیدگی تابع در یک جهت خاص یا ترکیبی از جهت‌هاست.
  - قطر اصلی ماتریس: میزان خمیدگی در هر متغیر به‌تنهایی
  - عناصر غیرقطری: تعامل بین متغیرها و اینکه تغییر در یکی چگونه روی دیگری اثر می‌گذارد
- در بهینه‌سازی، هسین به ما می‌گوید آیا نقطه‌ای که گرادیان صفر شده واقعاً کمینه است یا نه.
- اگر همهٔ خمیدگی‌ها مثبت باشند (یعنی ماتریس مثبت‌تعریف باشد)، آن نقطه یک کمینهٔ محلی است.

Iran Annual AI Award

هسیان

3D Surface Plot of f(x, y) = x² + y² with Gradient and Hessian at (1,1)



معیارهای همگرایی

- اگر وزنهای شبکه عصبی را با بردار θ نشان دهـیم بایـد تصـمیم بگیریم چه زمانی به نقطه بهینه رسیده‌ایم:
  - شرط گرادیان صفر: $\nabla f = 0$
  - تغییرات تابع هزینه کمتر از مقدار آستانه $|f(\theta_i) - f(\theta_{i+1})| < \epsilon$

Iran Annual AI Award

الگوریتم نزول گرادیان

- فرمول به‌روزرسانی:

$$\theta_{t+1} = \theta_t - \eta \nabla J(\theta_t)$$

- پارامترهای مدل: θ
- تابع هزینه: J
- نرخ یادگیری: η

Iran Annual AI Award

روش‌های نزول گرادیان

- تابع هزینه:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2$$

- نزول گرادیان پایه:

$$\nabla J(\theta) \cong \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)\nabla h_\theta(x^{(i)})$$

  - پارامترهای مدل: θ
  - تابع هزینه: J
  - نرخ یادگیری: η

- در هر مرحله بروزرسانی از همه داده‌ها استفاده می شود.

روش‌های نزول گرادیان

- نزول گرادیان تصادفی:

$$\nabla J(\theta) = \left(h_\theta(x^{(i)}) - y^{(i)}\right)\nabla h_\theta(x^{(i)})$$

  - پارامترهای مدل: θ
  - تابع هزینه: J
  - نرخ یادگیری: η
- در هر مرحله بروزرسانی از یک داده استفاده می شود.
- نوسان زیاد دارد ولی سریع است.

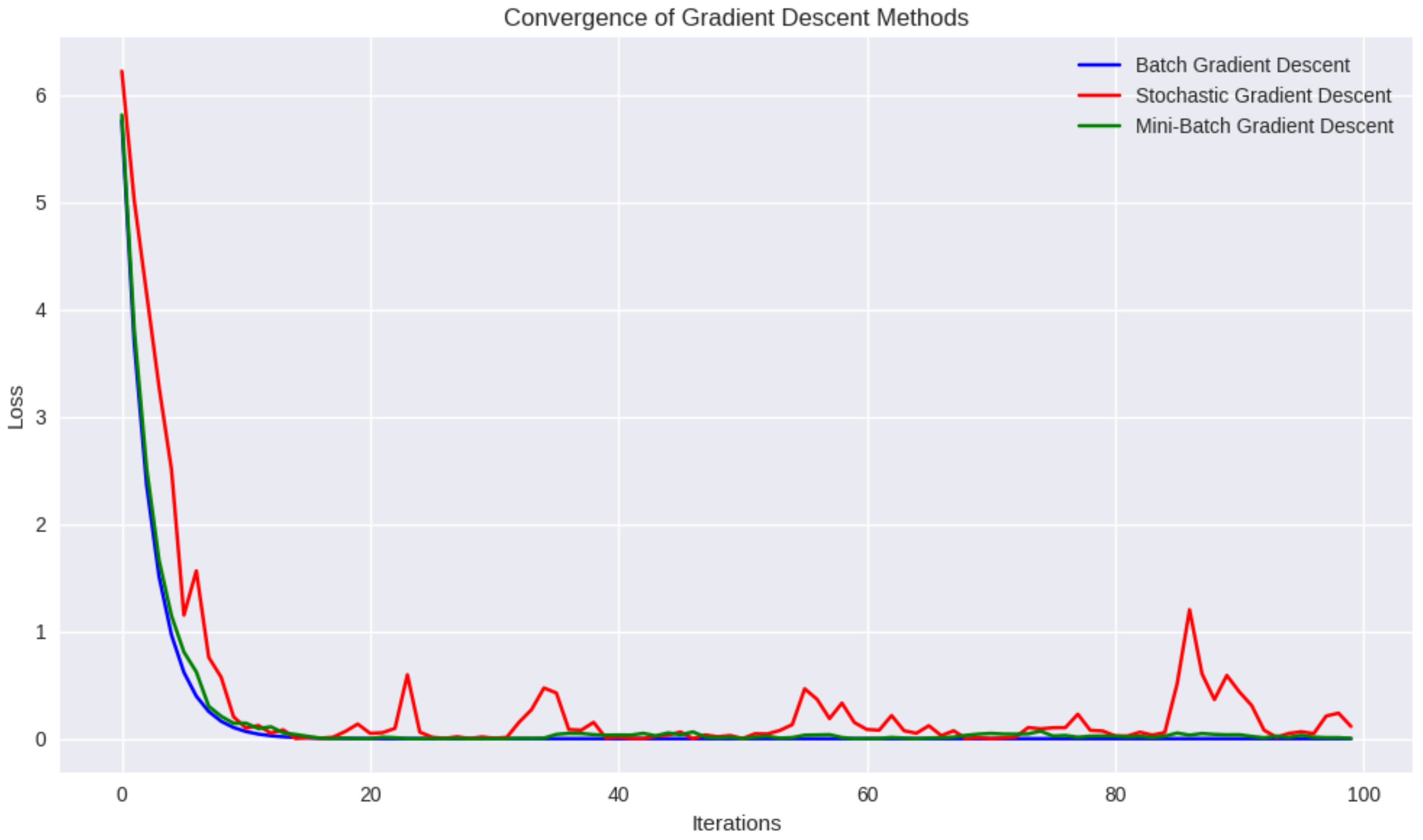
روش‌های نزول گرادیان

- نزول گرادیان مینی‌بچ:

$$\nabla J(\theta) \cong \frac{1}{b}\sum_{i=1}^{b}\left(h_\theta(x^{(i)}) - y^{(i)}\right)\nabla h_\theta(x^{(i)})$$

  - اندازه بچ: b
- در هر مرحله بروزرسانی از دسته‌ای از داده استفاده می شود.
- تعادل خوبی بین سرعت و دقت دارد.
- در عمل رایج‌ترین روش است، مخصوصاً در آموزش شـبکه‌های عصبی.

روش‌های نزول گرادیان



Iran Annual AI Award

انواع روش نزول گرادیان

ویژگی	Batch Gradient Descent	Stochastic Gradient Descent (SGD)	Mini-Batch Gradient Descent
📊 داده استفاده‌شده در هر مرحله	کل دیتاست	یک نمونه تصادفی	گروه کوچکی از نمونه‌ها
⏱️ سرعت هر آپدیت	کند (محاسبات زیاد)	بسیار سریع	متوسط
📉 نوسان در مسیر بهینه‌سازی	کم (مسیر صاف)	زیاد (نوسان بالا)	کنترل‌شده
🧠 حافظه مورد نیاز	زیاد	کم	متوسط
🎯 دقت نهایی	بالا	ممکن است نوسان کند	تعادل بین دقت و سرعت
🔄 مناسب برای	مدل‌های کوچک با داده کم	داده‌های بزرگ و آنلاین	اغلب کاربردهای عملی

تنظیم نرخ یادگیری

مقدار نرخ یادگیری	رفتار الگوریتم
خیلی کوچک	همگرایی کند، ولی پایدار
مناسب	همگرایی سریع و دقیق
خیلی بزرگ	نوسان زیاد، احتمال واگرایی (divergence)

تنظیم نرخ یادگیری

- ثابت (Constant):
  - ساده‌ترین حالت
  - مناسب برای مسائل ساده یا زمانی که دامنه تغییرات گرادیان کم است.
- کاهشی (Decay)
  - روش رایج $\eta = \frac{\eta_0}{1+kt}$
  - نرخ یادگیری با گذشت زمان کاهش می‌یابد و باعث می‌شود که مدل در مراحل اولیه با سرعت بیشتری یاد بگیرد و در مراحل پایانی با دقت بیشتری به سمت نقطهٔ بهینه حرکت کند

Iran Annual AI Award

تنظیم نرخ یادگیری

- **Warm-up + Annealing**
  - در ابتدا نرخ یادگیری کم است ← مدل با ثبات شروع می‌کند چون در مراحل اولیه آموزش، مدل هنوز در حال «یادگیری ساختار» داده‌هاست. اگر نرخ یادگیری زیاد باشد، ممکنه مدل ناپایدار یا از مسیر بهینه منحرف شود.
  - سپس افزایش می‌یابد ← یادگیری سریع‌تر
  - در نهایت کاهش می‌یابد ← بعد از اینکه مدل به نرخ یادگیری مطلوب رسید، باید به‌تدریج آن را کاهش بدهیم تا مدل با دقت بیشتری به نقطهٔ بهینه همگرا شود/
  - مثل خنک کردن فلز در فرآیند آنیل‌سازی: اول داغ و شکل‌پذیر است، بعد به‌آرامی سرد می‌شود تا ساختار پایدار پیدا کند.

Iran Annual AI Award

تنظیم نرخ یادگیری

- Cosine Annealing
- تصور کن نرخ یادگیری مثل سرعت حرکت در یک مسیر کوهستانی است:
  - در ابتدا با سرعت بالا حرکت می‌کنید (نرخ یادگیری زیاد).
  - به‌تدریج سرعتتان کم می‌شود تا به نقطه بهینه نزدیک‌تر شوید.
  - این کاهش سرعت به‌صورت نرم و پیوسته اتفاق می‌افتد، نه ناگهانی.
  - اگر از نسخه Warm Restart استفاده کنید، بعد از رسیدن به کمترین نرخ، دوباره سرعتتان زیاد می‌شود تا از مینیمم‌های محلی فرار کنید.
- کاهش نرم و پیوسته نرخ یادگیری ← همگرایی پایدارتر
- مناسب برای مدل‌های پیچیده با landscape چندمینیمومی
- قابل ترکیب با Warm Restarts برای فرار از مینیمم‌های محلی

Iran Annual AI Award

تنظیم نرخ یادگیری

- الگوریتم Adaptive Gradient — AdaGrad
- در الگوریتم‌های کلاسیک مثل Gradient Descent، نرخ یادگیری برای همهٔ پارامترها یکسان است.
- در بسیاری از مسائل، برخی پارامترها نیاز به تغییرات بزرگ‌تری دارند و برخی دیگر باید با احتیاط تنظیم شوند.
- AdaGrad این مشکل را حل می‌کند:
  - برای هر پارامتر، نرخ یادگیری جداگانه تنظیم می‌کند.
  - پارامترهایی که گرادیان زیادی دارند، نرخ یادگیری‌شان کاهش می‌یابد.
  - پارامترهایی که گرادیان کمی دارند، نرخ یادگیری‌شان حفظ می‌شود یا حتی افزایش می‌یابد.

تنظیم نرخ یادگیری

- الگوریتم AdaGrad — Adaptive Gradient
- مجموع مربعات گرادیان‌ها تا مرحله t برای هر پارامتر

$$r_i^{(t)} = r_i^{(t-1)} + \left(\frac{\partial J}{\partial \theta_i}\right)^2$$

$$\theta_i^{(t+1)} = \theta_i^{(t)} - \frac{\eta}{\sqrt{r_i^{(t)} + \epsilon}} \frac{\partial J}{\partial \theta_i}$$

تنظیم نرخ یادگیری

- الگوریتم Adaptive Gradient — AdaGrad
- تصور کنید دارید در یک مسیر کوهستانی با شیب‌های مختلف حرکت می‌کنید:
  - در جاهایی که شیب زیاد بوده (گرادیان بزرگ)، AdaGrad سرعت حرکت را کم می‌کند چون آن مسیر قبلاً زیاد به‌روزرسانی شده است.
  - در جاهایی که شیب کم بوده (گرادیان کوچک)، سرعت حرکت بیشتر می‌شود چون آن مسیر کمتر به‌روزرسانی شده است.
- این یعنی AdaGrad به‌صورت هوشمندانه‌تر تعادل یادگیری را بین پارامترها رعایت می‌کند.

Iran Annual AI Award

تنظیم نرخ یادگیری

- الگوریتم Momentum
- در الگوریتم Gradient Descent ساده، هر گام فقط به گرادیان لحظه‌ای وابسته است. این باعث می‌شود:
  - در مسیرهای پرشیب، مدل نوسان کند.
  - در دره‌های کم‌شیب، حرکت کند و گیر بیفتد.
- مثل توپ سنگینی که روی سطح شیب‌دار در حال حرکت است. حتی اگر شیب لحظه‌ای کم شود، توپ به‌خاطر شتاب قبلی‌اش به حرکت ادامه می‌دهد.
- Momentum مثل اضافه کردن جرم و شتاب به حرکت است:
  - اگر در یک جهت حرکت کرده‌ایم، تمایل داریم در همان جهت ادامه دهیم
  - این باعث می‌شود از نوسانات عبور کنیم و سریع‌تر به نقطهٔ بهینه برسیم

تنظیم نرخ یادگیری

- الگوریتم Momentum

$$v^{(t)} = \beta v^{(t-1)} + (1-\beta)\frac{\partial J}{\partial \theta}$$

$$\theta^{(t+1)} = \theta^{(t)} - \eta v^{(t)}$$

- $v^{(t)}$: شتاب تجمعی گرادیان‌ها
- β: ضریب ماندگاری شتاب (معمولاً بین 0.9 تا 0.99)
- η: نرخ یادگیری

Iran Annual AI Award

تنظیم نرخ یادگیری

ویژگی	Gradient Descent	Momentum
وابستگی به گرادیان لحظه‌ای	کامل	جزئی
سرعت همگرایی	کندتر	سریع‌تر
نوسان در جهت‌های پرشیب	زیاد	کمتر
عبور از دره‌های کم‌شیب	دشوار	آسان‌تر

Iran Annual AI Award

تنظیم نرخ یادگیری

- الگوریتم RMSProp: تنظیم تطبیقی نرخ یادگیری
  - با نرمال‌سازی گرادیان‌ها، نرخ یادگیری را برای هر پارامتر تطبیق می‌دهد
  - در جهت‌هایی که گرادیان زیاد است، نرخ یادگیری کاهش می‌یابد
  - در جهت‌هایی که گرادیان کم است، نرخ یادگیری حفظ می‌شود
- مثل رانندگی در جاده‌ای با شیب‌های مختلف—اگر شیب زیاد باشد، سرعت را کم می‌کنیم تا کنترل حفظ شود؛ اگر شیب کم باشد، با سرعت مناسب ادامه می‌دهیم.

تنظیم نرخ یادگیری

- الگوریتم RMSProp: تنظیم تطبیقی نرخ یادگیری

$$r_i^{(t)} = \gamma r_i^{(t-1)} + (1-\gamma)\left(\frac{\partial J}{\partial \theta_i}\right)^2$$

$$\theta_i^{(t+1)} = \theta_i^{(t)} - \frac{\eta}{\sqrt{r_i^{(t)} + \epsilon}} \frac{\partial J}{\partial \theta_i}$$

- $r_i^{(t)}$: میانگین نمایی مربعات گرادیان‌ها

تنظیم نرخ یادگیری

- الگوریتم Adaptive Moment Estimation — Adam:
  - ترکیب Momentum و RMSProp
  - Momentum: حفظ جهت حرکت با میانگین گرادیان‌ها

$$v_t = \beta_1 v_{t-1} + (1-\beta_1)\frac{\partial J}{\partial \theta}$$

  - RMSProp: تنظیم نرخ یادگیری با نرمال‌سازی گرادیان‌ها

$$r_t = \beta_2 r_{t-1} + (1-\beta_2)\left(\frac{\partial J}{\partial \theta}\right)^2$$

تنظیم نرخ یادگیری

- الگوریتم Adaptive Moment Estimation — Adam:
  - اصلاح بایاس

$$\hat{v}_t = \frac{v_t}{1-\beta_1^t}, \hat{r}_t = \frac{r_t}{1-\beta_2^t}$$

  - بروزرسانی پارامترها:

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

توابع محدب

- توابع محدب مثل یک کاسه هستند که اگر توپ را در هر نقطه‌ای رها کنیم، همیشه به پایین‌ترین نقطه می‌رسد.
- 🔍ویژگی کلیدی: هیچ دره یا قلهٔ محلی گمراه‌کننده وجود ندارد—فقط یک کمینهٔ واقعی.
- تابع f(x)محدب است اگر برای هر دو نقطه x_1 و x_2 و هر λ∈[0,1]

$$f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda\backslash\mathrm{f}(x_1) + (1-\lambda)f(x_2)$$

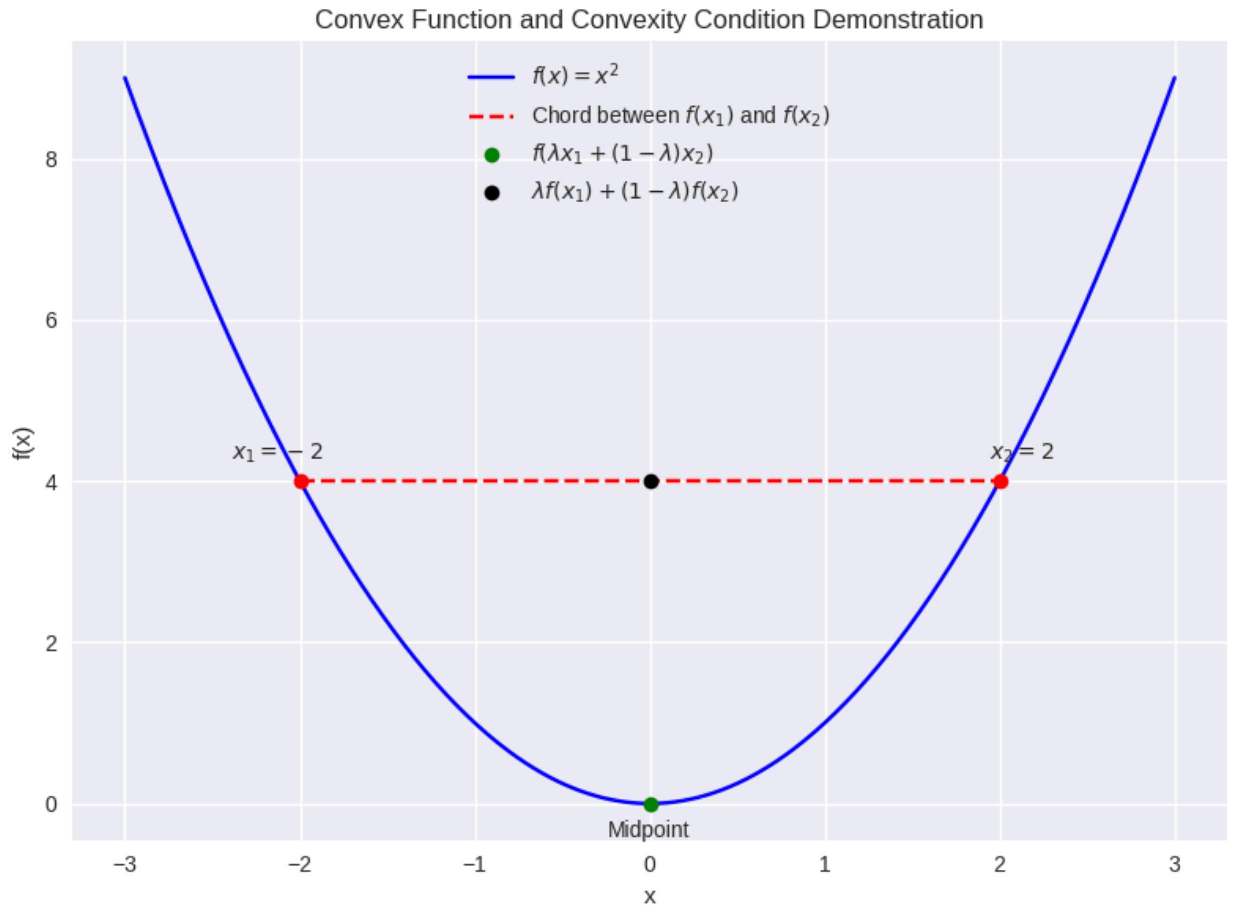
- یعنی اگر بین دو نقطه خط بکشیم، آن خط همیشه بالاتر از منحنی تابع قرار می‌گیرد.
- تضمین همگرایی، مشتق دوم مثبت، همگرایی سریع الگوریتم‌ها

Iran Annual AI Award

توابع محدب

- در رگرسیون خطی، تابع هزینه (مثل MSE) محدب است ← تضمین همگرایی
- در SVM، تابع هدف محدب است
- در طراحی الگوریتم‌های یادگیری ماشین، توابع هزینه محدب باعث می‌شوند مدل‌ها قابل اعتماد و قابل تحلیل باشند

توابع محدب


Convex Function and Convexity Condition Demonstration
$f(x) = x^2$
Chord between $f(x_1)$ and $f(x_2)$
$f(\lambda x_1 + (1-\lambda)x_2)$
$\lambda f(x_1) + (1-\lambda)f(x_2)$
$x_1 = -2$
$x_2 = 2$
Midpoint
f(x)
x
8
6
4
2
0
−3
−2
−1
0
1
2
3

روش نیوتن

- در الگوریتم‌های معمول مثل Gradient Descent، ما فقط از شیب تابع (گرادیان) استفاده می‌کنیم تا بفهمیم در کدام جهت حرکت کنیم.
- اما روش نیوتن یک قدم جلوتر می‌رود: نه‌تنها شیب را در نظر می‌گیرد، بلکه انحنای تابع را هم بررسی می‌کند
- با استفاده از مشتق دوم، تخمین می‌زند که نقطهٔ بهینه دقیقاً کجاست
- مثل راننده‌ای که فقط با دیدن شیب جاده تصمیم نمی‌گیرد، بلکه نقشهٔ انحنای مسیر را هم دارد

$$\theta_{t+1} = \theta_t - H^{-1}(\theta_t)\nabla J(\theta_t)$$

روش نیوتن

ویژگی	Gradient Descent	Newton's Method
استفاده از مشتق اول	✅	✅
استفاده از مشتق دوم	❌	✅
سرعت همگرایی	متوسط	بسیار سریع (در نزدیکی کمینه)
نیاز به محاسبه هسین	❌	✅
مناسب برای توابع محدب	✅	✅

Iran Annual AI Award

روش شبه‌نیوتنی

- در روش نیوتن، برای هر مرحله باید ماتریس هسین (مشتقات دوم) را محاسبه و معکوس کنیم. این کار در مسائل چندمتغیره بسیار پرهزینه است.
- روش BFGS (Broyden–Fletcher–Goldfarb–Shanno) این مشکل را حل می‌کند:
  - بدون محاسبه مستقیم هسین، آن را تقریبی و به‌روزشونده نگه می‌دارد
  - با استفاده از گرادیان‌های قبلی، تخمین می‌زند که انحنای تابع چگونه تغییر کرده
  - همگرایی سریع و پایدار دارد، مخصوصاً در مسائل محدب

iAAA
Iran Annual AI Award

www.iaaa.ai
support@iaaa.ai
021-91096992
iaaa.event
iaaa_ai

با تشکر
از توجه شما