昆山杜克大学
DUKE KUNSHAN UNIVERSITY

training nn

stats403_deep_learning

spring_2025

lecture_2

2.1 optimization methods

gradient descent

- consider the optimization task we face in deep learning

$$\min_{\boldsymbol{x} \in \mathbb{R}^d} f(\boldsymbol{x})$$

in the context of deep learning,
this is the vector of parameters

- the gradient descent update is

$$\boldsymbol{x}^{t+1} = \boldsymbol{x}^t - \eta^t \nabla f(\boldsymbol{x}^t)$$

learning rate

gradient descent

- intuition:

$$f(\boldsymbol{x}^{t+1}) = f(\boldsymbol{x}^t - \eta^t \nabla f(\boldsymbol{x}^t))$$

gradient descent

- assume that the eigenvalues of the Hessian of $f(\boldsymbol{x})$ is uniformly bounded by β: $\lambda\left(\nabla^2(f(\boldsymbol{x}))\right) \leq \beta$ for any $\boldsymbol{x}$

- $f(\boldsymbol{x}^t)$ descreases with t for a proper choice of η^t

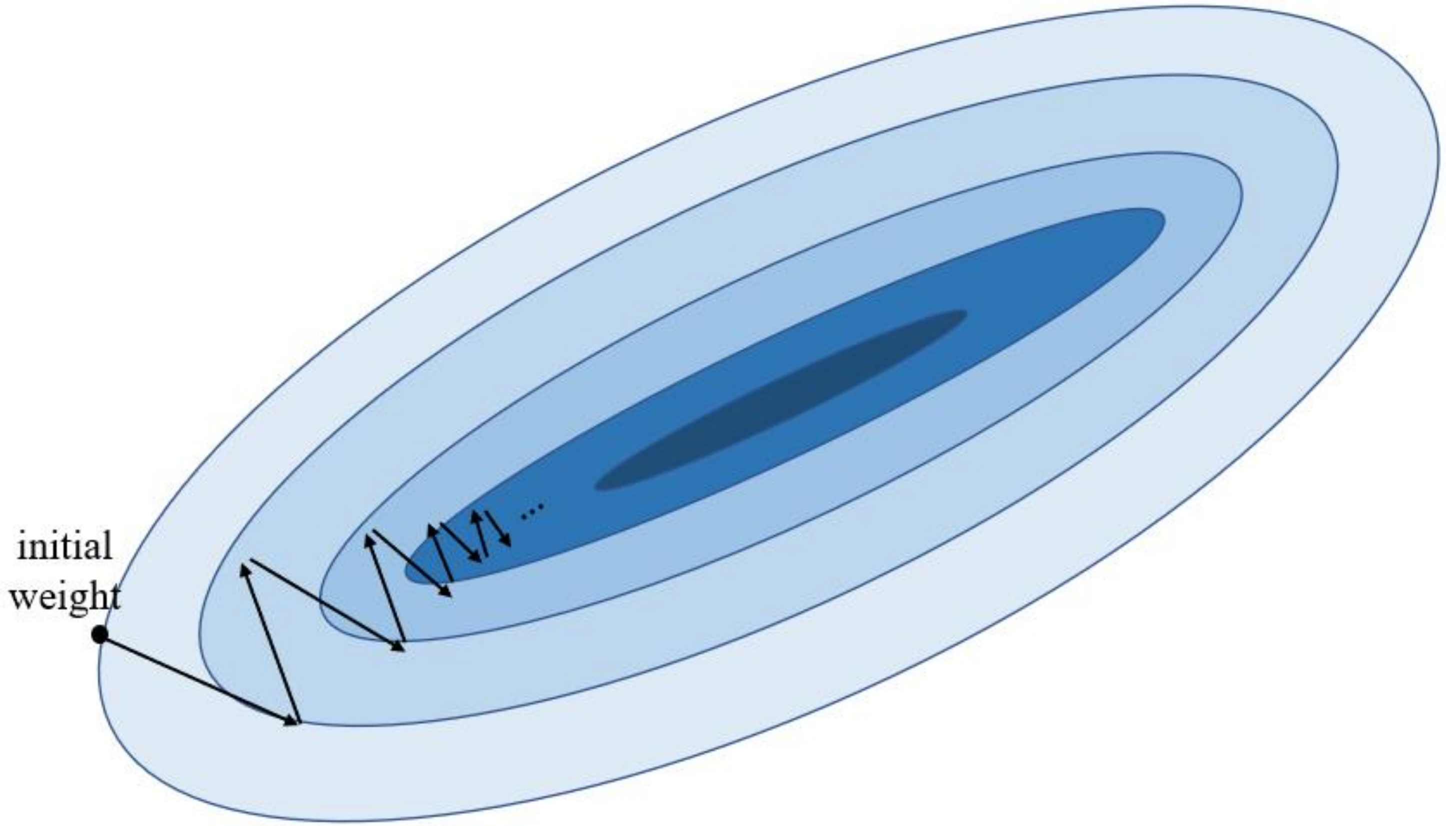
gradient descent

- further assume that f is strongly convex: the eigenvalues of the Hessian of $f(\boldsymbol{x})$ is uniformly bounded below by $\alpha > 0$: $\lambda\left(\nabla^2(f(\boldsymbol{x}))\right) \geq \alpha$ for any $\boldsymbol{x}$

- let $\boldsymbol{x}^*$ be the minimizer of f,

$$f(\boldsymbol{x}^T) - f(\boldsymbol{x}^*) \leq \left(1 - \frac{\alpha}{\beta}\right)^T \left(f(\boldsymbol{x}^0) - f(\boldsymbol{x}^*)\right)$$

problem with gradient descent



momentum

new direction old direction target direction

$$\mathbf{u}^{t+1} = \mathbf{u}^{t} - \beta^{t} \nabla f(\mathbf{x}^{t})$$

$$\mathbf{x}^{t+1} = \mathbf{x}^{t} + \gamma^{t} \mathbf{u}^{t+1}$$

momentum

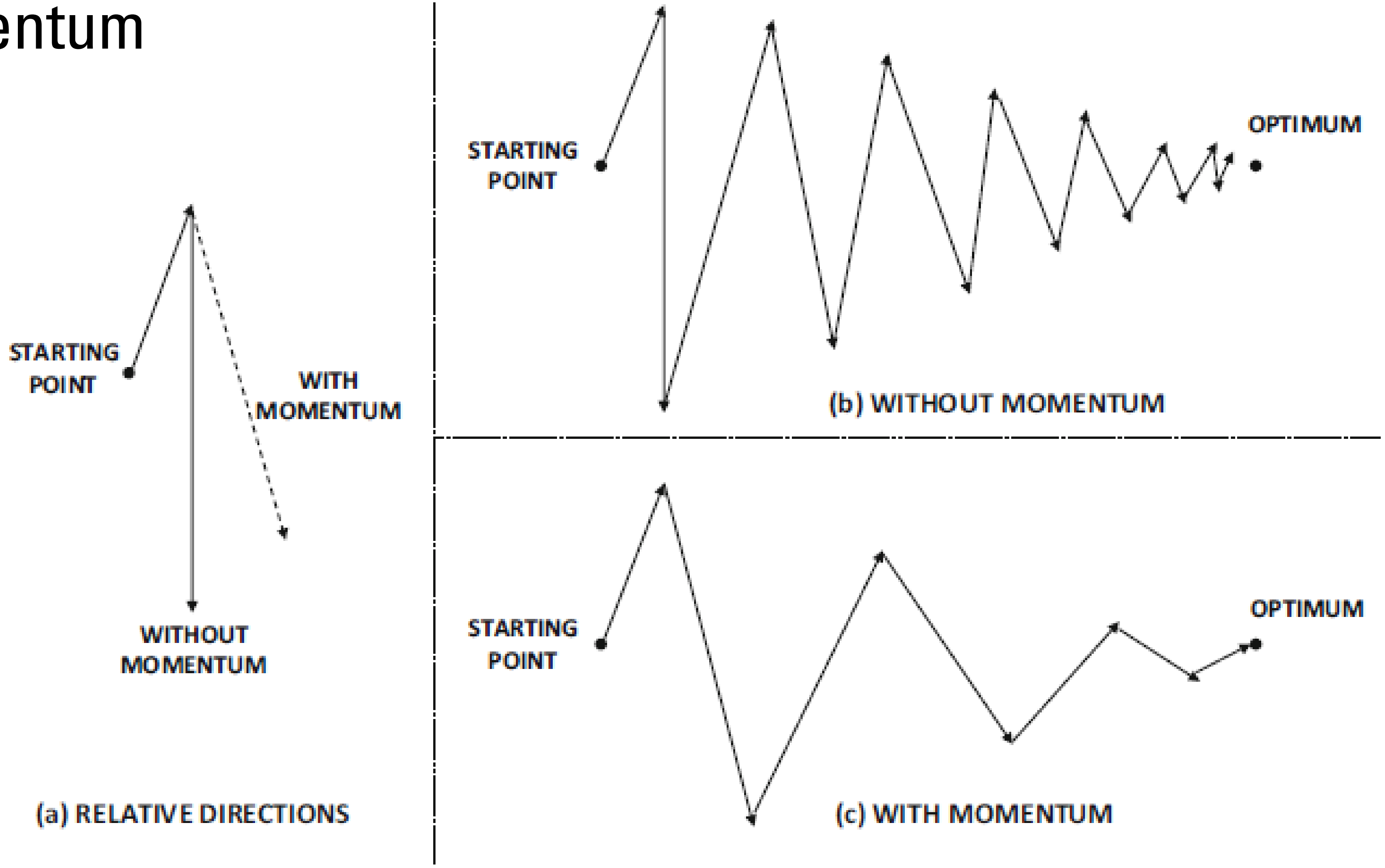


image credit: Aggarwal, Neural Networks and Deep Learning

Nesterov's acceleration

$$\mathbf{u}^{t+1} = \mathbf{x}^t - \beta^t \nabla f(\mathbf{x}^t)$$

$$\mathbf{x}^{t+1} = \mathbf{u}^{t+1} + \frac{t}{t+3}(\mathbf{u}^{t+1} - \mathbf{u}^t)$$

Gradient descent

$$x_{k+1} = x_k - \tau \nabla f(x_k)$$

$$\tau \to 0 \quad k\tau \to t$$

$$\frac{\mathrm{d}x(t)}{\mathrm{d}t} = -\nabla f(x(t))$$

Nesterov's acceleration

$$x_{k+1} = y_k - \tau \nabla f(y_k)$$
$$y_{k+1} = x_{k+1} + \frac{k}{k+3}(x_{k+1} - x_k)$$

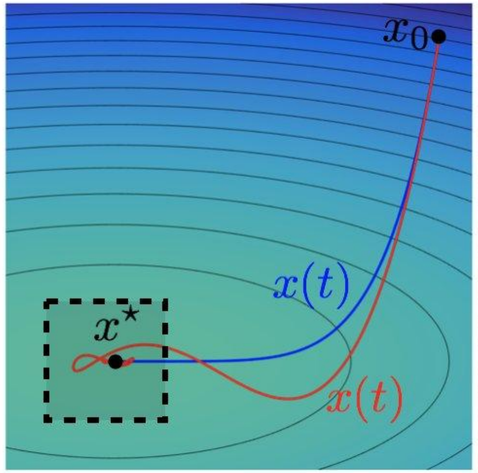
$$\tau \to 0 \quad k\sqrt{\tau} \to t$$

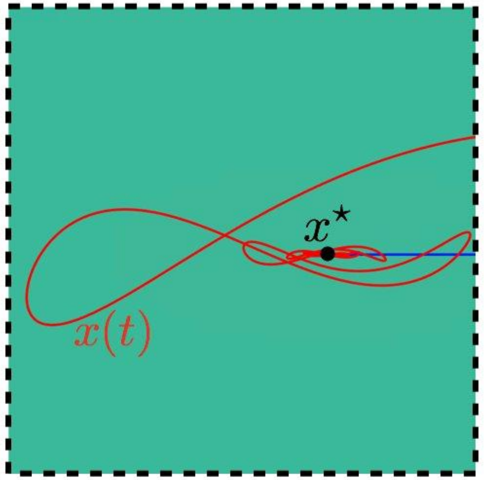
$$\frac{\mathrm{d}^2 x(t)}{\mathrm{d}t^2} + \frac{3}{t}\frac{\mathrm{d}x(t)}{\mathrm{d}t} = -\nabla f(x(t))$$

Theorem:

$$f(x_k) - f(x^\star) = O(1/k)$$

$$f(x_k) - f(x^\star) = O(1/k^2)$$





Yurii Nesterov

image credit:

Gabriel Peyré @gabrielpeyre · 23 Dec 2017 ···
Gradient descent is consistent with a first order ODE. Nesterov's acceleration is consistent with a second order ODE. Nesterov is better in the worse case. Gradient is better for strongly convex. #NoFreeLunch
arxiv.org/abs/1503.01243

preconditioner

$$\mathbf{x}^{t+1} = \mathbf{x}^t - \eta^t \mathbf{H}_t^{-1} \nabla f(\mathbf{x}^t)$$

preconditioner $\mathbf{x}^{t+1} = \mathbf{x}^t - \eta^t \mathbf{H}_t^{-1} \nabla f(\mathbf{x}^t)$

- a simple quadratic example: $f(\mathbf{x}) = c + \mathbf{b}^{\mathrm{T}}\mathbf{x} + \frac{1}{2}\mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x}$
- $\mathbf{x}^t \in \mathbb{R}^d$
- set $\nabla f(\mathbf{x}^{t+1}) = 0$ ……

AdaGrad: adaptive preconditioner

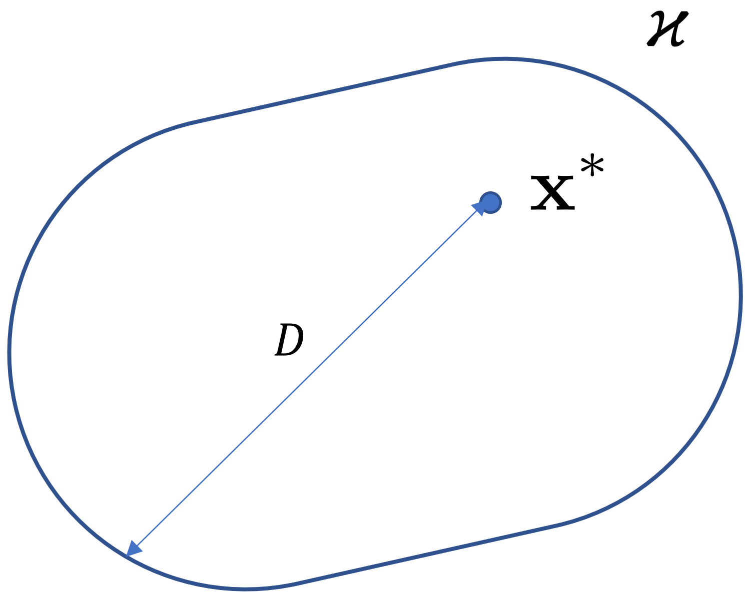
$$\mathbf{G}_t = \left(\sum_{i=1}^{t} \nabla f(\mathbf{x}^i) \nabla f(\mathbf{x}^i)^{\mathrm{T}} \right)^{\frac{1}{2}}$$

$$\mathbf{x}^{t+1} = \mathbf{x}^t - \eta \mathbf{G}_t^{-1} \nabla f(\mathbf{x}^t)$$

AdaGrad: convergence* (see draft book!)

assume

- f: convex objective function
- $\varkappa$: bounded convex set for searching $\mathbf{x}^*$
- D = $\max\limits_{\mathbf{x} \in K} ||\mathbf{x} - \mathbf{x}^*||$



define $||\mathbf{x}||^2_{\mathbf{A}} = \mathbf{x}^{\mathrm{T}}\mathbf{A}\mathbf{x}$

examining $||\mathbf{x}^{t+1} - \mathbf{x}^*||^2_{\mathbf{G}_t}$ yields a convergence bound (see draft book!)

AdaGrad (element-wise version)

$$\boldsymbol{g}^{(t)} = \left(\sum_{i=0}^{t} \nabla f(\boldsymbol{x}^{(i)}) \odot \nabla f(\boldsymbol{x}^{(i)}) \right)$$

$$\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} - \eta \nabla f(\boldsymbol{x}^{(t)}) \oslash \sqrt{\boldsymbol{g}^{(t)}},$$

Algorithm 8.4 The AdaGrad algorithm

Require: Global learning rate ϵ
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant δ, perhaps 10^{-7}, for numerical stability
  Initialize gradient accumulation variable $\boldsymbol{r} = \boldsymbol{0}$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$.
    Accumulate squared gradient: $\boldsymbol{r} \leftarrow \boldsymbol{r} + \boldsymbol{g} \odot \boldsymbol{g}$.
    Compute update: $\Delta \boldsymbol{\theta} \leftarrow -\frac{\epsilon}{\delta + \sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$. (Division and square root applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$.
  end while

---

algorithm taken from *Deep Learning*

RMSProp: modified AdaGrad

$$\boldsymbol{G}^{(t)} = \operatorname{diag}\left(\sum_{i=0}^{t} \nabla f(\boldsymbol{x}^{(i)}) \odot \nabla f(\boldsymbol{x}^{(i)})\right)$$

$$\boldsymbol{R}^{(t)} = \alpha \boldsymbol{R}^{(t-1)} + (1-\alpha)\boldsymbol{G}^{(t)}$$

$$\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} - \eta(\boldsymbol{R}^{(t)})^{-1/2}\nabla f(\boldsymbol{x}^{(t)}).$$

Algorithm 8.5 The RMSProp algorithm

Require: Global learning rate ϵ, decay rate ρ
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant δ, usually 10^{-6}, used to stabilize division by small numbers
Initialize accumulation variables $\boldsymbol{r} = 0$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$.
Accumulate squared gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1-\rho)\boldsymbol{g} \odot \boldsymbol{g}$.
Compute parameter update: $\Delta\boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}} \odot \boldsymbol{g}$. ($\frac{1}{\sqrt{\delta+\boldsymbol{r}}}$ applied element-wise)
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$.
end while

changing the gradient accumulation into an exponentially weighted moving average

algorithm taken from *Deep Learning*

Adam

$$
\begin{aligned}
\boldsymbol{g}^{(t)} &= \nabla f(\boldsymbol{x}^{(t)}) \\
\boldsymbol{m}^{(t)} &= \beta_1 \boldsymbol{m}^{(t-1)} + (1-\beta_1)\boldsymbol{g}^{(t)} \\
\boldsymbol{s}^{(t)} &= \beta_2 \boldsymbol{s}^{(t-1)} + (1-\beta_2)\boldsymbol{g}^{(t)} \odot \boldsymbol{g}^{(t)} \\
\hat{\boldsymbol{m}}^{(t)} &= \boldsymbol{m}^{(t)}/(1-\beta_1^t) \\
\hat{\boldsymbol{s}}^{(t)} &= \boldsymbol{s}^{(t)}/(1-\beta_2^t) \\
\boldsymbol{x}^{(t+1)} &= \boldsymbol{x}^{(t)} - \eta \hat{\boldsymbol{m}}^{(t)} \oslash \sqrt{\hat{\boldsymbol{s}}^{(t)}},
\end{aligned}
$$

Adam

Algorithm 8.7 The Adam algorithm

Require: Step size ϵ (Suggested default: 0.001)
Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$. (Suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant δ used for numerical stabilization (Suggested default: 10^{-8})
Require: Initial parameters $\boldsymbol{\theta}$
  Initialize 1st and 2nd moment variables $\boldsymbol{s} = \boldsymbol{0}$, $\boldsymbol{r} = \boldsymbol{0}$
  Initialize time step $t = 0$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    $t \leftarrow t + 1$
    Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1) \boldsymbol{g}$
    Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2) \boldsymbol{g} \odot \boldsymbol{g}$
    Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t}$
    Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t}$
    Compute update: $\Delta \boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$ (operations applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta \boldsymbol{\theta}$
  end while

algorithm taken from *Deep Learning*

Let’s compare the performance of these optimization methods!

- see colab notebook

SGDW and AdamW

Algorithm 1 SGD with L_2 regularization and SGD with decoupled weight decay (SGDW), both with momentum

given initial learning rate $\alpha \in \mathbb{R}$, momentum factor $\beta_1 \in \mathbb{R}$, weight decay/L_2 regularization factor $\lambda \in \mathbb{R}$

initialize time step $t \leftarrow 0$, parameter vector $\boldsymbol{\theta}_{t=0} \in \mathbb{R}^n$, first moment vector $\boldsymbol{m}_{t=0} \leftarrow \boldsymbol{0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$

repeat

$\quad t \leftarrow t + 1$

$\quad \nabla f_t(\boldsymbol{\theta}_{t-1}) \leftarrow \text{SelectBatch}(\boldsymbol{\theta}_{t-1})$ ▷ select batch and return the corresponding gradient

$\quad \boldsymbol{g}_t \leftarrow \nabla f_t(\boldsymbol{\theta}_{t-1}) + \lambda\boldsymbol{\theta}_{t-1}$

$\quad \eta_t \leftarrow \text{SetScheduleMultiplier}(t)$ ▷ can be fixed, decay, be used for warm restarts

$\quad \boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + \eta_t \alpha \boldsymbol{g}_t$

$\quad \boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \boldsymbol{m}_t - \eta_t \lambda \boldsymbol{\theta}_{t-1}$

until *stopping criterion is met*

return optimized parameters $\boldsymbol{\theta}_t$

SGDW and AdamW

Algorithm 2 Adam with L_2 regularization and Adam with decoupled weight decay (AdamW)

given $\alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}, \lambda \in \mathbb{R}$
initialize time step $t \leftarrow 0$, parameter vector $\boldsymbol{\theta}_{t=0} \in \mathbb{R}^n$, first moment vector $\boldsymbol{m}_{t=0} \leftarrow \boldsymbol{0}$, second moment vector $\boldsymbol{v}_{t=0} \leftarrow \boldsymbol{0}$, schedule multiplier $\eta_{t=0} \in \mathbb{R}$
repeat
  $t \leftarrow t + 1$
  $\nabla f_t(\boldsymbol{\theta}_{t-1}) \leftarrow \text{SelectBatch}(\boldsymbol{\theta}_{t-1})$ ▷ select batch and return the corresponding gradient
  $\boldsymbol{g}_t \leftarrow \nabla f_t(\boldsymbol{\theta}_{t-1}) + \lambda\boldsymbol{\theta}_{t-1}$
  $\boldsymbol{m}_t \leftarrow \beta_1 \boldsymbol{m}_{t-1} + (1 - \beta_1)\boldsymbol{g}_t$ ▷ here and below all operations are element-wise
  $\boldsymbol{v}_t \leftarrow \beta_2 \boldsymbol{v}_{t-1} + (1 - \beta_2)\boldsymbol{g}_t^2$
  $\hat{\boldsymbol{m}}_t \leftarrow \boldsymbol{m}_t / (1 - \beta_1^t)$ ▷ β_1 is taken to the power of t
  $\hat{\boldsymbol{v}}_t \leftarrow \boldsymbol{v}_t / (1 - \beta_2^t)$ ▷ β_2 is taken to the power of t
  $\eta_t \leftarrow \text{SetScheduleMultiplier}(t)$ ▷ can be fixed, decay, or also be used for warm restarts
  $\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta_t \left(\alpha \hat{\boldsymbol{m}}_t / (\sqrt{\hat{\boldsymbol{v}}_t} + \epsilon) + \lambda\boldsymbol{\theta}_{t-1} \right)$
until *stopping criterion is met*
return optimized parameters $\boldsymbol{\theta}_t$

initialization

- good initialization is important in iterative optimization methods for both convergence and efficiency

- this is especially true when it comes to training neural networks since the loss landscape is very complicated

- more importantly, since gradients propagate from the output to the input, a good initialization must avoid vanishing or exploding gradients in this process

initialization

- to achieve this, a good heuristic for initializing the weights of a dense network is to keep the variance of each layer the same after applying activation functions

- for instance, one popular initialization method is the Xavier normal initialization, where each weight is initialized randomly according to $\mathcal{N}(0, \sigma^2)$ where $\sigma^2 = \text{gain} \times \frac{2}{\#\text{in} + \#\text{out}}$

- "gain"' is a scalar normalization factor which takes into account the activation function used (https://pytorch.org/docs/stable/nn.init.html).

Thank you!

Reference

- Ch 6, 7.12, 8.3-8.5, 8.7.1, *Deep Learning*.
- Ch 20.3, *Understanding Machine Learning*.
- Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4), 303-314.
- Arora, S., Lectures 3 & 5, *Towards Theoretical Understanding of Deep Learning (COS597G)*.
- https://pytorch.org/docs/stable/optim.html
- https://github.com/jettify/pytorch-optimizer
- Défossez, A., Bottou, L., Bach, F., & Usunier, N. (2020). *A Simple Convergence Proof of Adam and Adagrad*. https://arxiv.org/pdf/2003.02395.pdf
- https://blogs.princeton.edu/imabandit/orf523-the-complexities-of-optimization/
- Hazan, *Introduction to Online Convex Optimization*. https://arxiv.org/pdf/1909.05207.pdf