M2 MIAGE
ID/IDApp

OPTIMIZATION FOR DATA SCIENCE

December 8, 2025

Today (Session 4/8)

- Recap on gradient-based methods (through exercise)
- Basics of stochastic gradient
- Next week (hopefully): lab session

2] Recap on gradient descent (Exercise 1 in Tutorial 2)

One-layer neural network (aka linear regression)

- Data $\{(x_i, y_i)\}_{i=1..m}$, $x_i \in \mathbb{R}^d$, $y_i \in \mathbb{R}$
- Linear model $x \mapsto x^T w$, parameterized by $w \in \mathbb{R}^d$

$w^T x = \sum [w]_j [x]_j$

- Goal (regression): Find w such that $x_i^T w - y_i \simeq 0 \ \forall i$

Optimization problem

$$(P) \quad \min_{w \in \mathbb{R}^d} f(w) = \frac{1}{2m} \sum_{i=1}^{m} \left(x_i^T w - y_i \right)^2$$

- $x_i^T w$: Model output given x_i
- y_i: True output
- $\left(x_i^T w - y_i \right)^2$: Loss function (squared loss)
- $\frac{1}{2m} \sum_{i=1}^{m}$: Average over all data points

a) What class of problems does (P) belong to?

→ (P) is a smooth optimization problem (f is C^1, C^2)

→ (P) is a convex optimization problem (f is convex)

→ (P) is a quadratic optimization problem (f is a degree-2 polynomial of the coordinates of w)

→ (P) is a linear least-squares problem

$$X = \begin{bmatrix} x_1^T \\ \vdots \\ x_m^T \end{bmatrix} \in \mathbb{R}^{m \times d}, \quad y = \begin{bmatrix} y_1 \\ \vdots \\ y_m \end{bmatrix} \in \mathbb{R}^m, \quad f(w) = \frac{1}{2m} \|Xw - y\|^2$$

$$X_w = \begin{bmatrix} x_1^T w \\ \vdots \\ x_n^T w \end{bmatrix} \qquad \|v\|^2 = \sum [v]_j^2$$

b) $f \in C^{1,1}_L$, i.e. • $f \in C^1$ $(\forall w \in \mathbb{R}^d, \exists \nabla f(w) \in \mathbb{R}^d)$

• $\forall (w,v) \in (\mathbb{R}^d)^2$, $\|\nabla f(w) - \nabla f(v)\| \leq L \|w - v\|$

How can we use L in an algorithm such as gradient descent?

Gradient descent for minimizing f: Starting from w_0, compute

$$w_{k+1} = w_k - \alpha_k \nabla f(w_k) \qquad \alpha_k > 0$$

→ A good choice for α_k is $\alpha_k = \frac{1}{L}$ $\forall k \in \mathbb{N}$

↳ Gives theoretical guarantees

↳ Works well in practice especially for linear regression (cf Session 2)

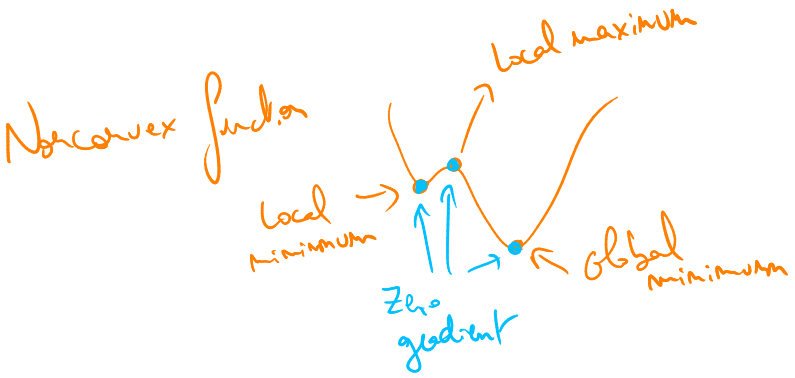
→ Using $\alpha_k \in \left(0, \frac{2}{L}\right)$ guarantees $f(w_{k+1}) < f(w_k)$ when $\nabla f(w_k) \neq 0$, and thus it is also a valid choice in theory (N.B.: In practice, using a very small α_k is not efficient)

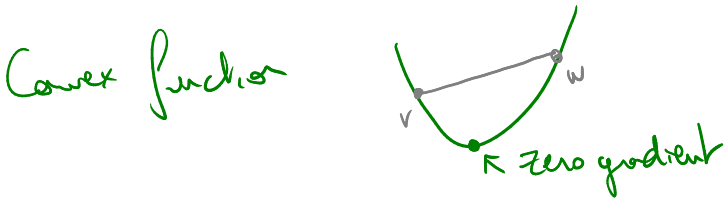
c) Problem (P) is convex (i.e. f is a convex function)

i) What can we say about $\bar{w} \in \mathbb{R}^d$ such that $\nabla f(\bar{w}) = 0_{\mathbb{R}^d}$?

Since f is convex, $\nabla f(\bar{w}) = 0_{\mathbb{R}^d} \iff \bar{w}$ is a global minimum of f

$\left(\bar{w} \in \operatorname{argmin}_{w \in \mathbb{R}^d} f(w)\right)$





f convex

$$f(\alpha v + (1-\alpha) w) \leq \alpha f(v) + (1-\alpha) f(v)$$

ii) What is the convergence rate of gradient descent on (P) ?

(P) is convex / f is convex

Convergence rate: Given a budget of K iterations, how good { is the last iterate } computed by gradient { are the iterates } descent ?

Convergence rate for gradient descent in the convex setting

After $K \geq 1$ iterations,

$$f(w_K) - \min_{w \in \mathbb{R}^d} f(w) \leq O\left(\frac{1}{K}\right)$$

A constant that does not depend on K (depends on $L, w_0, \dots$)

$O(A) = C \times A$

where $C > 0$ does not depend on A

Here, $\frac{L}{2}\|w_0 - w^*\|^2$

$w^* \in \operatorname{argmin}_{w \in \mathbb{R}^d} f(w)$

Important parts of that result
- the rate $\left(\frac{1}{K}\right)$
- the quantity the rate applies to $\left(f(w_K) - \min_{w \in \mathbb{R}^d} f(w)\right)$

iii) What is the convergence rate for accelerated gradient on (P)? Is it better or worse than the rate of gradient descent?

Accelerated gradient idea: Combine gradient descent steps with momentum steps
↓
involve the previous iteration

$$w_{k+1} = w_k + \beta_{k+1}(w_k - w_{k-1}) - \alpha_k \nabla f\left(w_k + \beta_{k+1}(w_k - w_{k-1})\right)$$

Momentum step: moving in the direction of the previous step $(w_k - w_{k-1})$

Additional parameter of the method

$\alpha_k > 0$

$\beta_{k+1} > 0$ (see formulas in notebook)

Convergence rate of accelerated gradient (on convex functions)

After $K \geq 1$ iterations,

$$f(w_K) - \min_{w \in \mathbb{R}^d} f(w) \leq O\left(\frac{1}{K^2}\right)$$

↑ Same quantity than for gradient descent

↑ Rate $\frac{1}{K^2}$ faster than $\frac{1}{K}$ (goes more quickly to 0)

as $(K \to +\infty)$

→ The rate of accelerated gradient is better than the rate of gradient descent

(Remark: In general for accelerated gradient, we may have $f(w_{k+1}) > f(w_k)$ for some iterations but overall the function values $\{f(w_k)\}$ decrease more quickly than in gradient descent)

d) Under assumptions on the problem data (on the x_i s), f can be μ-strongly convex. with $\mu > 0$.

i) If $\nabla f(w) = \nabla f(v) = 0_{\mathbb{R}^d}$, what can we say about w and v?

A strongly convex function has a unique global minimum, and it is the only point with zero gradient.

Hence $w = v$

ii) Convergence rate of accelerated gradient on this problem?

If f μ-strongly convex and $C^{1,1}_L$, after K iterations of accelerated gradient, we have

$$f(w_K) - \min_{w \in \mathbb{R}^d} f(w) \leq O\left(\left(1 - \sqrt{\frac{\mu}{L}}\right)^K \right)$$

- $\mu \leq L \Rightarrow \left(1-\sqrt{\frac{\mu}{L}}\right) \in (0,1)$

  $\Rightarrow \left(1-\sqrt{\frac{\mu}{L}}\right)^K$ converges to 0 faster than $\frac{1}{K}$ and $\frac{1}{K^2}$

- The convergence rate for accelerated gradient on strongly convex problems is
  1) faster than on convex problems: $\left(1-\sqrt{\frac{\mu}{L}}\right)^K$ vs $\frac{1}{K^2}$
  2) faster than the rate of GD on strongly convex problems: $\left(1-\sqrt{\frac{\mu}{L}}\right)^K$ vs $\left(1-\frac{\mu}{L}\right)^K$

  ↓ Rate for GD (gradient descent)

1) From gradient descent to stochastic gradient

↳ Gradient descent is a very generic method that applies to any problem of the form $\underset{w \in \mathbb{R}^d}{\text{minimize}}\ f(w)$ with $f($

↳ But problems in ML are more structured, and are typically finite-sum problems

$$\underset{w \in \mathbb{R}^d}{\text{minimize}}\ f(w) = \frac{1}{n} \sum_{i=1}^{n} f_i(w)$$

where every f_i depends on the i^{th} data point in a dataset with n elements (usually f_i is the loss with respect to that data point)

Ex) $f(w) = \frac{1}{2n} \sum_{i=1}^{n} (x_i^T w - y_i)^2$ $\qquad f_i(w) = \frac{1}{2}(x_i^T w - y_i)^2$

$= \frac{1}{n} \sum_{i=1}^{n} f_i(w)$

↳ Suppose that all f_i functions are C^1.

Then $\nabla f(w) = \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(w)$

Thus, to compute a gradient of f (as in gradient descent), we need to compute all gradients of f_i (n "sample gradients")

$\Rightarrow$ To do this, we need to look at the entire dataset, which is expensive when $n \gg 1$

$\Rightarrow$ If there is an underlying distribution of data points, not all data points are necessary to compute a good approximation of the gradient of f

Stochastic gradient: (SG) Use a single data point per iteration

Starting from $w_0 \in \mathbb{R}^d$, compute

$$w_{k+1} = w_k - \alpha_k \nabla f_{i_k}(w_k) \qquad \alpha_k > 0$$

↑ Gradient of f_{i_k}

where i_k is an index drawn randomly in $\{1, \dots, n\}$

→ Every iteration of SG accesses 1 data point only

⇒ Much cheaper than an iteration of GD, that accesses n data points

With a budget of N accesses to data points, can do:

- $\frac{N}{n}$ iterations of GD) ⊕ accurate gradients ⊖ few iterations
- N iterations of SG) ⊖ inexact/stochastic gradients ⊕ many iterations

Metric: Epochs

Def: An epoch is a unit of cost that corresponds to n accesses to data points in a dataset with n elements.

1 iteration of GD costs 1 epoch

1 iteration of SG costs $\frac{1}{n}$ epoch

⇔ 1 epoch = the cost of n iterations of SG

2) Theory of stochastic gradient

→ Stochastic gradient is a randomized algorithm

⇒ Running it twice gives different results!

⇒ It might not work on some runs!

⇒ We can show that the method works | on average / in expectation

→ To obtain convergence rates on stochastic gradient, we need assumptions on the randomness of the method

Assumptions At every iteration k, the random index i_k is drawn such that:

- i_k is independent of $i_0, \dots, i_{k-1}$) i_k is drawn independently from the past iterations
- $\mathbb{E}_{i_k}\left[\nabla f_{i_k}(w_k)\right] = \nabla f(w_k)$] on average, get the true gradient
- $\mathbb{E}_{i_k}\left[\|\nabla f_{i_k}(w_k)\|^2\right] \leq \|\nabla f(w_k)\|^2 + \sigma^2$] the norm of the stochastic gradient does not vary too much from the true gradient norm

for $\sigma^2 \geq 0$

Expected value (aka Average)

$$\mathbb{E}_{i_k}\left[\varphi(i_k)\right] = \sum_{i=1}^{n} \mathbb{P}(i_k = i) \times \varphi(i)$$ for every function φ of i_k

→ In the theory of gradient descent, we use the descent lemma

$$f\left(w_k - \alpha_k \nabla f(w_k)\right) \leq f(w_k) - \left(\alpha_k - \frac{L\alpha_k^2}{2}\right)\|\nabla f(w_k)\|^2$$

for deriving convergence rates.

→ Under the assumptions above i_k, we get an expected descent lemma for stochastic gradient

$$\mathbb{E}_{i_k}\left[f\left(w_k - \alpha_k \nabla f_{i_k}(w_k)\right)\right] \leq f(w_k) - \left(\alpha_k - \frac{L\alpha_k^2}{2}\right)\|\nabla f(w_k)\|^2 + \frac{L\alpha_k^2\sigma^2}{2}$$

- $\mathbb{E}_{i_k}[\cdots]$: the function value decreases on average
- $-\left(\alpha_k - \frac{L\alpha_k^2}{2}\right)\|\nabla f(w_k)\|^2$: Gradient descent "descent lemma"
- $\frac{L\alpha_k^2\sigma^2}{2}$: Accounts for the randomness of $\nabla f_{i_k}(w_k)$

→ From this lemma we can prove (expected) convergence rates for stochastic gradient, and compare them to rates for gradient descent on the same problem.

Illustration: Stochastic gradient with fixed stepsize on strongly convex functions

Suppose that the problem is $\min_{w \in \mathbb{R}^d} \frac{1}{n}\sum_{i=1}^n f_i(w)$ where

- f_i C^1 $\forall i$
- f $C_L^{1,1}$ with $L > 0$
- f μ-strongly convex with $\mu > 0$

① GD with stepsize $1/L$:

After K iterations, $f(w_K) - \min_{w \in \mathbb{R}^d} f(w) \leq \left(1 - \frac{\mu}{L}\right)^K \left(f(w_0) - \min_{w \in \mathbb{R}^d} f(w)\right)$

"$\mathcal{O}\left(\left(1-\frac{\mu}{L}\right)^K\right)$"

② SG with stepsize $1/L$

$$\mathbb{E}\left[f(w_K) - \min_{w \in \mathbb{R}^d} f(w)\right] \leq \left(1 - \frac{\mu}{L}\right)^K \left(f(w_0) - \min_{w \in \mathbb{R}^d} f(w) - \frac{\sigma^2}{2\mu}\right) + \frac{\sigma^2}{2\mu}$$

$\mathbb{E}[\cdot]$ expected value with respect to $i_0, \dots, i_{K-1}$

$\frac{\sigma^2}{2\mu}$: "Noise level"
Appears because we use stochastic gradients

Q) which rate is the best one?

GD	SG
$f(w_K) - \min_{w \in \mathbb{R}^d} f(w) \leq \left(1 - \frac{\mu}{L}\right)^K \left(f(w_0) - \min_{w \in \mathbb{R}^d} f(w)\right)$	$\mathbb{E}\left[f(w_K) - \min_{w \in \mathbb{R}^d} f(w)\right] \leq \left(1 - \frac{\mu}{L}\right)^K \left(f(w_0) - \min_{w \in \mathbb{R}^d} f(w) - \frac{\sigma^2}{2\mu}\right) + \frac{\sigma^2}{2\mu}$
• $f(w_K) - \min_{w \in \mathbb{R}^d} f(w) \to 0$ (convergence)	• $f(w_K) - \min_{w \in \mathbb{R}^d} f(w) \to \left[0, \frac{\sigma^2}{2\mu}\right]$ (convergence to a neighborhood of the solution)
• True deterministically	• True in expectation
• Rate $O\left(\left(1 - \frac{\mu}{L}\right)^K\right)$	• Rate : $O\left(\left(1 - \frac{\mu}{L}\right)^K\right)$

Conclusion: GD has better guarantees than SG!
(convergence rate; in terms of iterations)

BUT an iteration of GD is more expensive than an iteration of SG in terms of accesses to data points

$\Rightarrow$ To account for the cost of an iteration, we can derive convergence rates for a fixed number of epochs N_E

GD

N_E epochs $\equiv N_E$ iterations

$$f(w_{N_E}) - \min_{w \in \mathbb{R}^d} f(w) \leq \left(1 - \frac{\mu}{L}\right)^{N_E} \left(f(w_0) - \min_{w \in \mathbb{R}^d} f(w)\right)$$

- Deterministic convergence
- CV rate $\left(1 - \frac{\mu}{L}\right)^{N_E}$

SG

N_E epochs $\equiv m N_E$ iterations

$$\mathbb{E}\left[f(w_{mN_E}) - \min_{w \in \mathbb{R}^d} f(w)\right]$$

$$\leq \left(1 - \frac{\mu}{L}\right)^{mN_E} \left(f(w_0) - \min_{w \in \mathbb{R}^d} f(w) - \frac{\sigma^2}{2\mu}\right) + \frac{\sigma^2}{2\mu}$$

- Convergence in expectation to a neighborhood
- CV rate $\left(1 - \frac{\mu}{L}\right)^{mN_E}$

$\Rightarrow$ Faster when $m > 1$

$\Rightarrow$ Much faster when m is large compared to 1 ($m \gg 1$)

Takeaway

- SG guarantees fast convergence to a neighborhood of the solution
- GD guarantees slower convergence but to the exact solution
- When $m \gg 1$ and N_E small, SG is preferred.

$m \geq 10^9$

N_E can be as small as 1!

3) Advanced variants of stochastic gradient

$$w_{k+1} = w_k - \alpha_k \nabla f_{i_k}(w_k) \quad , \alpha_k > 0$$

"Vanilla stochastic gradient"

- **Batch stochastic gradient** Use more than 1 sample per iteration

$$w_{k+1} = w_k - \alpha_k \left(\frac{1}{|S_k|} \sum_{i \in S_k} \nabla f_i(w_k) \right)$$

where S_k is a set of indices drawn randomly in $\{1, \dots, n\}$ with or without replacement

$|S_k|$: number of elements in S_k

→ Generalizes SG ($S_k = \{i_k\}$, $|S_k| = 1$)

GD ($S_k = \{1, \dots, n\}$, $|S_k| = n$ n indices drawn without replacement)

⊕ Helps reducing the variance of the stochastic gradient estimates (σ^2 parameter)

⊕ Helps exploit parallelism

Ex) 64 cores ⇒ $|S_k| = 64$

⊕ Using mini-batch SG ($|S_k| \ll n$) can lead to faster convergence than SG

⊖ There are cases where SG ($|S_k|=1$) perform best

⊖ An iteration of batch SG (with $|S_k|>1$) is always more expensive than an iteration of SG in terms of accesses to data points/epochs
⇒ Expensive without parallelism

⊖ In a large batch regime ($|S_k| \gg 1$), batch methods tend to perform like GD, and thus to converge much more slowly than SG.

→ Finding the optimal batch size $|S_k|$ is a difficult task that is problem-dependent

Popular stochastic gradient techniques for deep learning

Generic format

$$w_{k+1} = w_k - \alpha_k \, m_k \, / \, v_k$$

$\alpha_k > 0$, $m_k \in \mathbb{R}^d$, $v_k \in \mathbb{R}^d$

/ : componentwise division

$\forall j = 1..d$, $[w_{k+1}]_j = [w_k]_j - \alpha_k \dfrac{[m_k]_j}{[v_k]_j}$

Vanilla SG: $m_k = \nabla f_{i_k}(w_k)$ $\quad v_k = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix}$

→ Advanced variants for deep learning use a different formula for m_k (direction) and/or for V_k (normalization of the stepsize)

(a) S.G. with momentum

→ Inspired by accelerated gradient

→ $m_k = \beta_1 m_{k-1} + (1-\beta_1) \nabla f_{i_k}(w_k)$

m_{k-1}: Previous direction

$\nabla f_{i_k}(w_k)$: New stochastic gradient

$\beta_1 \in (0,1)$ ($\beta_1 = 0$: Vanilla SG)

$\beta_1 \simeq 0.9$ in default PyTorch implementation

→ Original method for succesfully training neural networks (2012)

(b) AdaGrad / RMSProp

$$m_k = \nabla f_{i_k}(w_k)$$

$$\forall j = 1..d, \quad [V_k]_j = \frac{1}{\sqrt{\sum_{l=0}^{k-1} [\nabla f_{i_l}(w_l)]_j^2 + [\nabla f_{i_k}(w_k)]_j^2}}$$

Accumulating magnitude of the j^{th} coordinate of all stochastic gradients

→ If the gradients have coordinates that vary in magnitude, scales the stepsize appropriately for every coordinate

RMS Prop: Variant of Adagrad

$$[v_k]_j = \frac{1}{\sqrt{\beta_2 \sum_{l=0}^{k-1} [\nabla f_{i_l}(w_l)]_j^2 + (1-\beta_2)\,[\nabla f_{i_k}(w_k)]_j^2}}$$

Information from past iterations ← $\beta_2 \sum_{l=0}^{k-1} [\nabla f_{i_l}(w_l)]_j^2$

New stochastic gradient → $(1-\beta_2)\,[\nabla f_{i_k}(w_k)]_j^2$

$\beta_2 \in (0,1)$

→ Weigh the importance of the new coordinate

→ Typically $\beta_2 \simeq 0.9$

(RMSProp 2012-2013, highly popular at the time for very deep neural networks)

Adam (Kingma & Ba 2015)

$$m_k = \frac{1-\beta_1^{k+1}}{1-\beta_1} \sum_{l=0}^{k} \beta_1^l \nabla f_{i_l}(w_l) \qquad \beta_1 \in (0,1)$$

$$[v_k]_j = \left(\frac{1-\beta_2^{k+1}}{1-\beta_2} \sum_{l=0}^{k} \beta_2^l [\nabla f_{i_l}(w_l)]_j^2 \right)^{-1/2} \qquad \beta_2 \in (0,1)$$

→ Weighted combination of information from all iterations

→ $\beta_1 = 0.9$ and $\beta_2 = 0.999$ in PyTorch

(More weight to past iterations)

↳ Mainstream algorithm for training modern neural networks (along with variants like AdamW)

- Because it worked very well in practice
- Not because of the theory
  - * Original 2015 paper had theory
  - * In 2018, the theory was proven to be false

Remark: New algorithms are always being proposed (Ex: Muon (2024))