Optimization in Machine Learning

First order methods
Gradient descent

Learning goals

- Iterative Descent / Line Search
- Descent directions
- GD
- ERM with GD
- Pseudoresiduals

ITERATIVE DESCENT

Let f be the height of a mountain depending on the geographic coordinates (x_1, x_2)

$$f : \mathbb{R}^2 \to \mathbb{R}, \quad f(x_1, x_2) = y.$$

Goal: Reach the valley

$$\arg\min_{\mathbf{x}} f(\mathbf{x})$$

Central idea: Repeat

1. At current location $\mathbf{x} \in \mathbb{R}^d$ search for **descent direction $\mathbf{d}$** $\in \mathbb{R}^d$
2. Move along $\mathbf{d}$ until f „sufficiently“ reduces (**step size control**) and update location

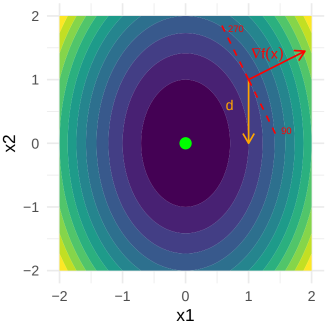
"Walking down the hill, towards the valley."

ITERATIVE DESCENT

Let $f : \mathbb{R}^d \to \mathbb{R}$ continuously differentiable.

Definition: $\mathbf{d} \in \mathbb{R}^d$ is a **descent direction** in $\mathbf{x}$ if

$$D_{\mathbf{d}} f(\mathbf{x}) = \nabla f(\mathbf{x})^T \mathbf{d} < 0 \text{ (neg. directional derivative)}$$



Angle between $\nabla f(\mathbf{x})$ and $\mathbf{d}$ must be $\in (90^\circ, 270^\circ)$.

ITERATIVE DESCENT

Algorithm Iterative Descent / Line search

1: Starting point $\mathbf{x}^{[0]} \in \mathbb{R}^d$
2: **while** Stopping criterion not met **do**
3: Calculate a descent direction $\mathbf{d}^{[t]}$ for current $\mathbf{x}^{[t]}$
4: Find $\alpha^{[t]}$ s.t. $f(\mathbf{x}^{[t+1]}) < f(\mathbf{x}^{[t]})$ for $\mathbf{x}^{[t+1]} = \mathbf{x}^{[t]} + \alpha^{[t]}\mathbf{d}^{[t]}$.
5: Update $\mathbf{x}^{[t+1]} = \mathbf{x}^{[t]} + \alpha^{[t]}\mathbf{d}^{[t]}$
6: **end while**

NB: Terminology is sometimes ambiguous: “line search” can refer to Step 4 (selecting the step size that decreases $f(\mathbf{x})$) and can mean umbrella term for iterative descent algorithms.

Key questions:

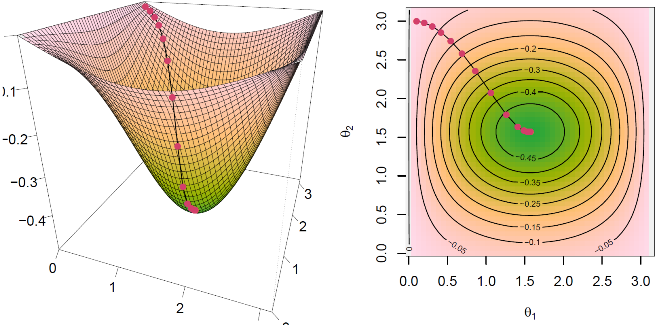
- How to choose $\mathbf{d}^{[t]}$ (now)
- How to choose $\alpha^{[t]}$ (later)

GRADIENT DESCENT

Properties of gradient:

- $\nabla f(\mathbf{x})$: direction of greatest increase
- $-\nabla f(\mathbf{x})$: direction of greatest decrease

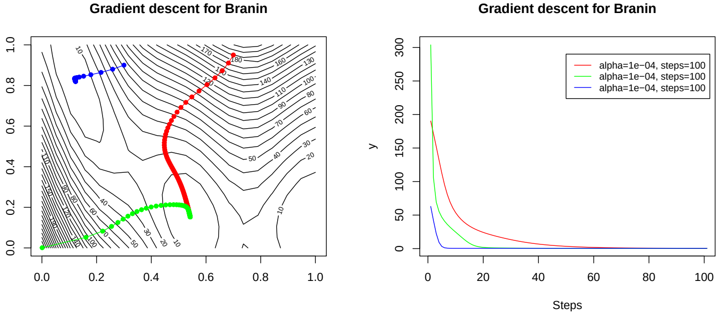
Using $\mathbf{d} = -\nabla f(\mathbf{x})$ is called **gradient descent**.



GD for $f(x_1, x_2) = -\sin(x_1) \cdot \frac{1}{2\pi} \exp\left((x_2 - \pi/2)^2\right)$ with sensibly chosen step size $\alpha^{[t]}$.

GD AND MULTIMODAL FUNCTIONS

Outcome will depend on start point.



100 iters of GD with const $\alpha = 10^{-4}$.

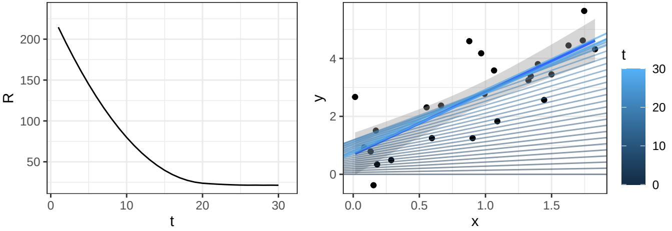
OPTIMIZE LS LINEAR REGRESSION WITH GD

Let $\mathcal{D} = \left(\left(\mathbf{x}^{(1)}, y^{(1)}\right), \ldots, \left(\mathbf{x}^{(n)}, y^{(n)}\right)\right)$ and minimize

$$\mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) = \sum_{i=1}^{n} \left(\boldsymbol{\theta}^\top \mathbf{x}^{(i)} - y^{(i)}\right)^2$$

NB: For illustration, we use GD even though closed-form solution exists. GD-like (more adv.) approaches like this MIGHT make sense for large data, though.

Gradient: $$\nabla_{\boldsymbol{\theta}} \mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) = \frac{\partial \mathcal{R}_{\text{emp}}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = -\sum_{i=1}^{n} 2 \cdot \left(y^{(i)} - \boldsymbol{\theta}^\top \mathbf{x}^{(i)}\right) \mathbf{x}^{(i)}$$

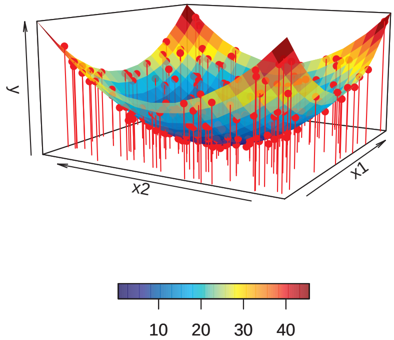


ERM FOR NN WITH GD

Let $\mathcal{D} = \left(\left(\mathbf{x}^{(1)}, y^{(1)} \right), \ldots, \left(\mathbf{x}^{(n)}, y^{(n)} \right) \right)$, with $y = x_1^2 + x_2^2$ and minimize

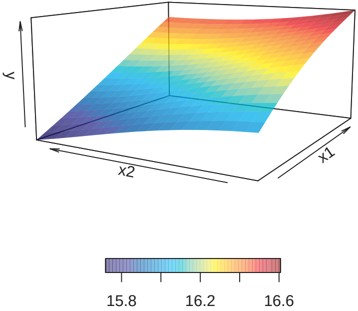
$$\mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) = \sum_{i=1}^{n} \left(f(\mathbf{x} \mid \boldsymbol{\theta}) - y^{(i)} \right)^2$$

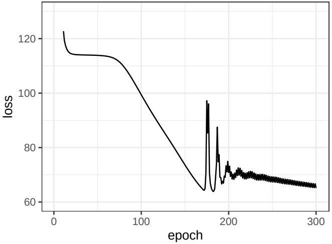
where $f(\mathbf{x} \mid \boldsymbol{\theta})$ is a neural network with 2 hidden layers (2 units each).

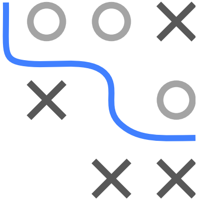


ERM FOR NN WITH GD

After 10 iters of GD:

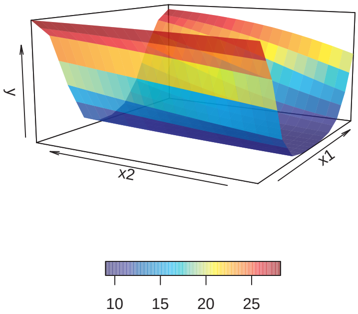


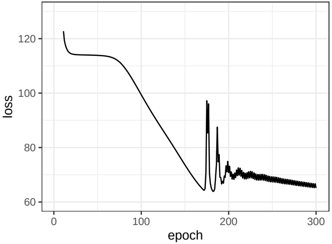




ERM FOR NN WITH GD

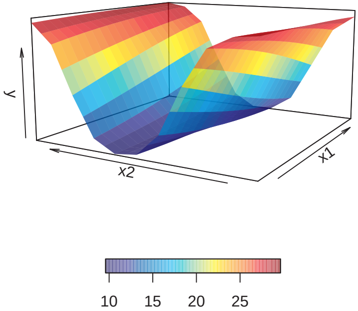
After 100 iters of GD:

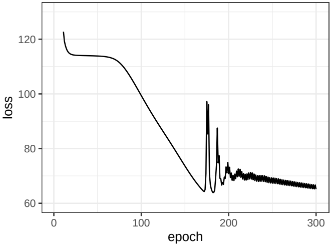




ERM FOR NN WITH GD

After 300 iters of GD (note the zig-zag-behavior after iter. 200):





GD FOR ERM: PSEUDORESIDUALS

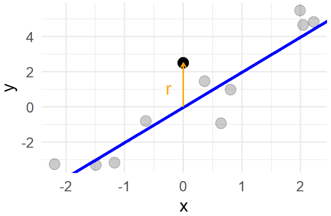
Gradient for ERM problem:

$$-\frac{\partial \mathcal{R}_{\text{emp}}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = -\frac{\partial \sum_{i=1}^{n} L\left(y^{(i)}, f\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right)\right)}{\partial \boldsymbol{\theta}} = -\sum_{i=1}^{n} \underbrace{\frac{\partial L\left(y^{(i)}, f\left(\mathbf{x}^{(i)}\right)\right)}{\partial f}}_{\text{pseudo residual } \tilde{r}^{(i)}(f)} \frac{\partial f\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right)}{\partial \boldsymbol{\theta}}$$

- **pseudo residuals** tell us how to distort $f\left(\mathbf{x}^{(i)}\right)$ to achieve greatest decrease of $L\left(y^{(i)}, f\left(\mathbf{x}^{(i)}\right)\right)$ (best pointwise update)
- $\frac{\partial f\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right)}{\partial \boldsymbol{\theta}}$ tells us how to modify $\boldsymbol{\theta}$ accordingly and wiggle model output
- GD step sums up these modifications across all observations i

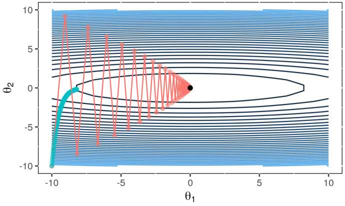
NB: Pseudo-residuals $\tilde{r}(f)$ match usual residuals for L2 loss:

$$\frac{\partial L(y, f(\mathbf{x}))}{\partial f} = \frac{1}{2} \frac{\partial (y-f)^2}{\partial f}\bigg|_{f=f(\mathbf{x})}$$
$$= y - f(\mathbf{x})$$



Optimization in Machine Learning

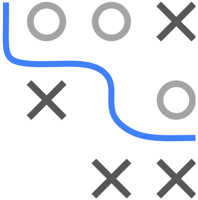
First order methods
Step size and optimality



Learning goals

- Impact of step size
- Fixed vs. adaptive step size
- Exact line search
- Armijo rule & Backtracking
- Bracketing & Pinpointing

CONTROLLING STEP SIZE: FIXED & ADAPTIVE

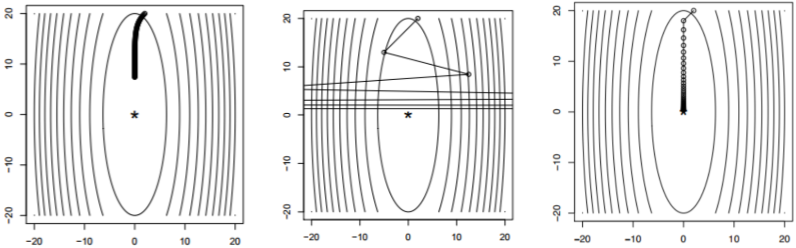


Iteration t: Choose not only descent direction $\mathbf{d}^{[t]}$, but also step size $\alpha^{[t]}$

First approach: **Fixed** step size $\alpha^{[t]} = \alpha > 0$

- If α too small, procedure may converge very slowly (left)
- If α too large, procedure may not converge $\rightarrow$ "jumps" around optimum (middle)

Adaptive step size $\alpha^{[t]}$ can provide better convergence (right)



Steps of line searches for $f(\boldsymbol{x}) = 10x_1^2 + x_2^2/2$

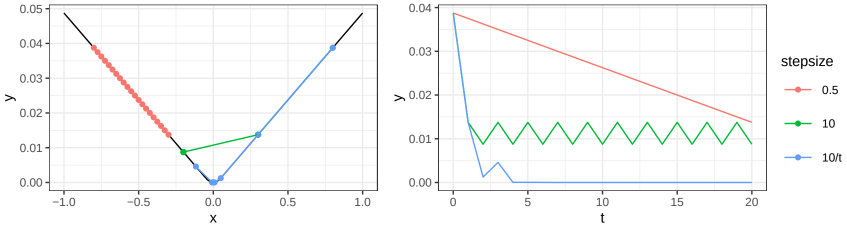
STEP SIZE CONTROL: DIMINISHING STEP SIZE

How can we adaptively control step size?

A natural way of selecting $\alpha^{[t]}$ is to decrease its value over time

Example: GD on

$$f(x) = \begin{cases} \frac{1}{2}x^2 & \text{if } |x| \leq \delta, \\ \delta \cdot (|x| - 1/2 \cdot \delta) & \text{otherwise.} \end{cases}$$



GD with small constant (**red**), large constant (**green**), and diminishing (**blue**) step size

STEP SIZE CONTROL: EXACT LINE SEARCH

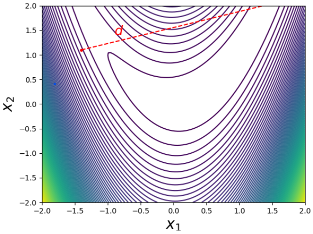
Use **optimal** step size in each iteration:

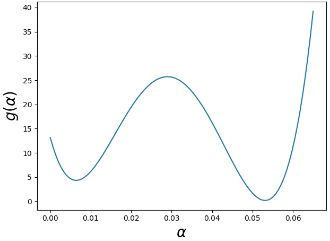
$$\alpha^{[t]} = \arg\min_{\alpha \in \mathbb{R}_{\geq 0}} g(\alpha) = \arg\min_{\alpha \in \mathbb{R}_{\geq 0}} f(\boldsymbol{x}^{[t]} + \alpha \mathbf{d}^{[t]})$$

Need to solve a **univariate** optimization problem in each iteration
$\Rightarrow$ univariate optimization methods

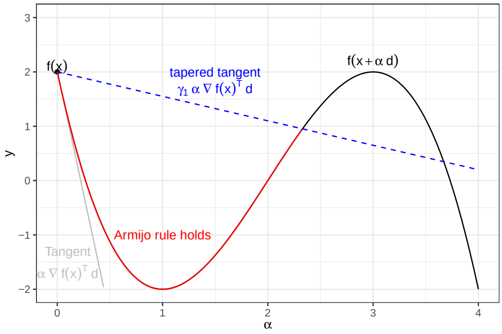
Problem: Expensive, prone to poorly conditioned problems

But: No need for *optimal* step size. Only need a step size that is “good enough”.
Reason: Effort may not pay off, but in some cases slows down performance.





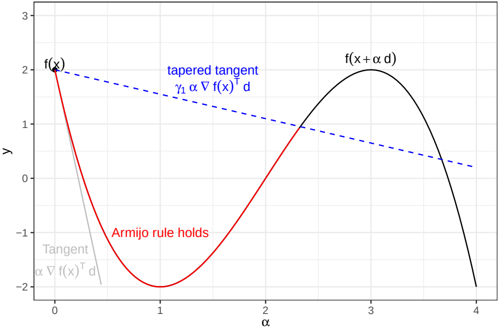
ARMIJO RULE



Inexact line search: Minimize objective “sufficiently” without computing optimal step size exactly

Common condition to guarantee “sufficient” decrease: **Armijo rule**

ARMIJO RULE

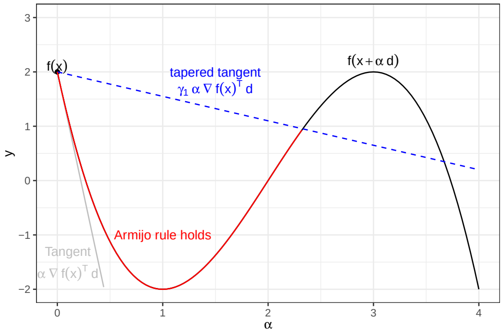


Fix $\gamma_1 \in (0, 1)$. α satisfies **Armijo rule** in $\mathbf{x}$ for descent direction $\mathbf{d}$ if

$$f(\mathbf{x} + \alpha \mathbf{d}) \leq f(\mathbf{x}) + \gamma_1 \alpha \nabla f(\mathbf{x})^\top \mathbf{d}.$$

Note: $\nabla f(\mathbf{x})^\top \mathbf{d} < 0$ ($\mathbf{d}$ *descent* dir.) $\implies f(\mathbf{x} + \alpha \mathbf{d}) < f(\mathbf{x})$.

ARMIJO RULE



Feasibility: For descent direction **d** and $\gamma_1 \in (0, 1)$, there exists $\alpha > 0$ fulfilling Armijo rule. In many cases, Armijo rule guarantees local convergence of GD and is therefore frequently used.

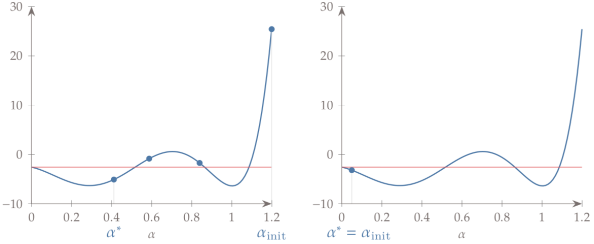
BACKTRACKING LINE SEARCH

Procedure to meet the Armijo rule: **Backtracking** line search

Idea: Decrease α until Armijo rule is met

Algorithm Backtracking line search

Choose initial step size $\alpha = \alpha_{\text{init}}$, $0 < \gamma_1 < 1$ and $0 < \tau < 1$
while $f(\boldsymbol{x} + \alpha \mathbf{d}) > f(\boldsymbol{x}) + \gamma_1 \alpha \nabla f(\boldsymbol{x})^\top \mathbf{d}$ **do**
    Decrease α: $\alpha \leftarrow \tau \cdot \alpha$
end while



(Source: Martins and Ning. *Engineering Design Optimization*, 2021.)

WOLFE CONDITIONS

Backtracking is simple and shows good performance in practice

But: Two undesirable scenarios

1. Initial step size α_{init} is too large $\Rightarrow$ need multiple evaluations of f
2. Step size is too small with highly negative slopes

Solution for small step sizes:

- Fix γ_2 with $0 < \gamma_1 < \gamma_2 < 1$.
- α satisfies **sufficient curvature condition** in $\mathbf{x}$ for $\mathbf{d}$ if

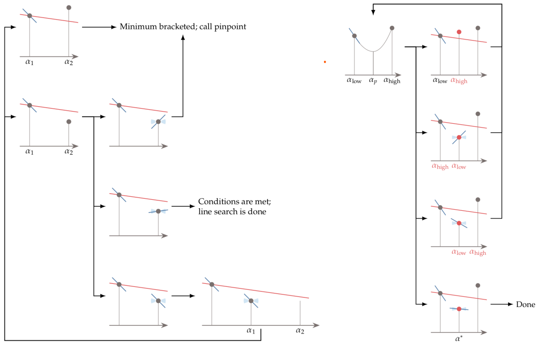
$$|\nabla f(\mathbf{x} + \alpha \mathbf{d})^\top \mathbf{d}| \leq \gamma_2 |\nabla f(\mathbf{x})^\top \mathbf{d}|.$$

Armijo rule + sufficient curvature condition = **Wolfe conditions**

WOLFE CONDITIONS

Algorithm for finding a Wolfe point (point satisfying Wolfe conditions):

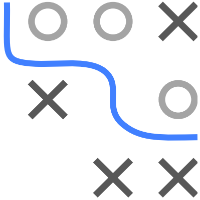
1. **Bracketing:** Find interval containing Wolfe point
2. **Pinpointing:** Find Wolfe point in interval from bracketing



Left: Bracketing. **Right:** Pinpointing.

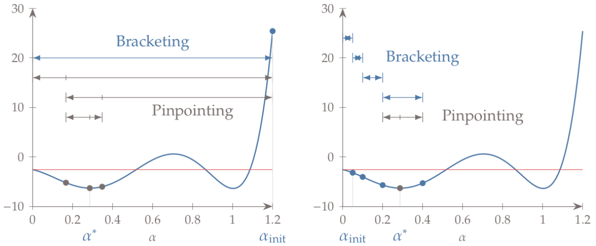
(Source: Martins and Ning. *EDO*, 2021.)

BRACKETING & PINPOINTING



Example:

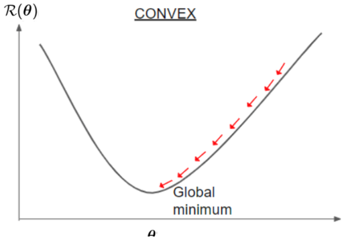
- Large initial step size results in quick bracketing but multiple pinpointing steps (**left**).
- Small initial step size results in multiple bracketing steps but quick pinpointing (**right**).



Source: Martins and Ning. *EDO*, 2021.

Optimization in Machine Learning

Deep dive
Gradient descent and optimality

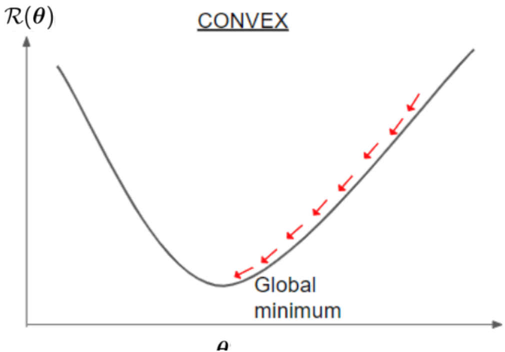


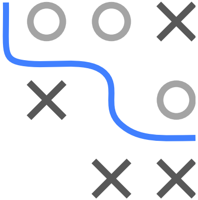
Learning goals

- Convergence of GD
- Proof strategy and tools
- Descent lemma

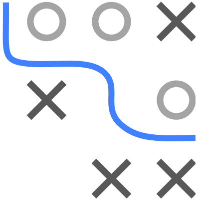
SETTING

- GD is **greedy**: **locally optimal** moves in each iteration
- If f is **convex**, **differentiable** and has a **Lipschitz gradient**, GD converges to global minimum for sufficiently small step sizes.





SETTING



Assumptions:

- f convex and differentiable
- Global minimum $\mathbf{x}^*$ exists
- f has Lipschitz gradient (∇f does not change too fast)

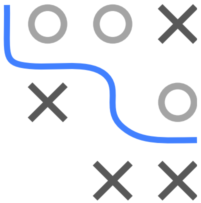
$$\|\nabla f(\mathbf{x}) - \nabla f(\tilde{\mathbf{x}})\| \leq L\|\mathbf{x} - \tilde{\mathbf{x}}\| \quad \text{for all } \mathbf{x}, \tilde{\mathbf{x}}$$

Theorem (Convergence of GD)**.** GD with step size $\alpha \leq 1/L$ yields

$$f(\mathbf{x}^{[k]}) - f(\mathbf{x}^*) \leq \frac{\|\mathbf{x}^{[0]} - \mathbf{x}^*\|^2}{2\alpha k}.$$

In other words: GD converges with rate $\mathcal{O}(1/k)$.

PROOF STRATEGY



❶ Show that $f(\mathbf{x}^{[t]})$ **strictly decreases** with each iteration t

Descent lemma:

at each it'on we at least gain

$$f(\mathbf{x}^{[t+1]}) \leq f(\mathbf{x}^{[t]}) - \frac{\alpha}{2} \|\nabla f(\mathbf{x}^{[t]})\|^2$$

❷ Bound **error of one step**

we're not far more than by

$$f(\mathbf{x}^{[t+1]}) - f(\mathbf{x}^{*}) \leq \frac{1}{2\alpha} \left(\|\mathbf{x}^{[t]} - \mathbf{x}^{*}\|^2 - \|\mathbf{x}^{[t+1]} - \mathbf{x}^{*}\|^2 \right)$$

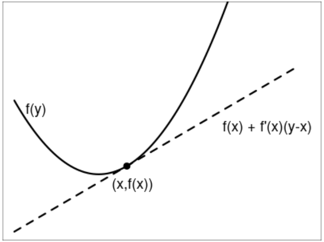
❸ Finalize by **telescoping** argument

MAIN TOOL

Recall: First order condition of convexity

Every tangent line of f is always below f.

$$f(\mathbf{y}) \geq f(\mathbf{x}) + \nabla f(\mathbf{x})^\top (\mathbf{y} - \mathbf{x})$$



DESCENT LEMMA

Recall: ∇f Lipschitz $\implies \nabla^2 f(\mathbf{x}) \preccurlyeq L \cdot \mathbf{I}$ for all $\mathbf{x}$

This gives convexity of $g(\mathbf{x}) := \frac{L}{2}\|\mathbf{x}\|^2 - f(\mathbf{x})$ since

$$\nabla^2 g(\mathbf{x}) = L \cdot I - \nabla^2 f(\mathbf{x}) \succcurlyeq 0.$$

$\nabla g(x) = Lx - \nabla f(x)$

First order condition of convexity of g yields

plugging in definitions

$\|x - x^{[t]}\|^2 = \|x\|^2 - 2x^T x^{[t]} + \|x^{[t]}\|^2$

$$g(\mathbf{x}) \geq g(\mathbf{x}^{[t]}) + \nabla g(\mathbf{x}^{[t]})^\top (\mathbf{x} - \mathbf{x}^{[t]})$$

$$\Leftrightarrow \quad \frac{L}{2}\|\mathbf{x}\|^2 - f(\mathbf{x}) \geq \frac{L}{2}\|\mathbf{x}^{[t]}\|^2 - f(\mathbf{x}^{[t]}) + (L\mathbf{x}^{[t]} - \nabla f(\mathbf{x}^{[t]}))^\top (\mathbf{x} - \mathbf{x}^{[t]})$$

$Lx^{[t]T}x - Lx^{[t]T}x^{[t]}$

$$\Leftrightarrow \quad \vdots$$

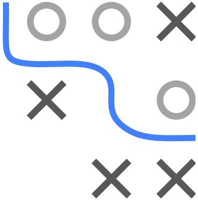
$$\Leftrightarrow \quad f(\mathbf{x}) \leq f(\mathbf{x}^{[t]}) + \nabla f(\mathbf{x}^{[t]})^\top (\mathbf{x} - \mathbf{x}^{[t]}) + \frac{L}{2}\|\mathbf{x} - \mathbf{x}^{[t]}\|^2$$

Now: One GD step with step size $\alpha \leq 1/L$:

$$\mathbf{x} \leftarrow \mathbf{x}^{[t+1]} = \mathbf{x}^{[t]} - \alpha \nabla f\left(\mathbf{x}^{[t]}\right)$$

DESCENT LEMMA

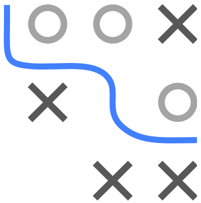
$$
\begin{aligned}
f(\mathbf{x}^{[t+1]}) &\leq f(\mathbf{x}^{[t]}) + \nabla f(\mathbf{x}^{[t]})^\top (\mathbf{x}^{[t+1]} - \mathbf{x}^{[t]}) + \frac{L}{2}\|\mathbf{x}^{[t+1]} - \mathbf{x}^{[t]}\|^2 \\
&= f(\mathbf{x}^{[t]}) + \nabla f(\mathbf{x}^{[t]})^\top (\mathbf{x}^{[t]} - \alpha \nabla f(\mathbf{x}^{[t]}) - \mathbf{x}^{[t]}) \\
&\quad + \frac{L}{2}\|\mathbf{x}^{[t]} - \alpha \nabla f(\mathbf{x}^{[t]}) - \mathbf{x}^{[t]}\|^2 \\
&= f(\mathbf{x}^{[t]}) - \nabla f(\mathbf{x}^{[t]})^\top \alpha \nabla f(\mathbf{x}^{[t]}) + \frac{L}{2}\|\alpha \nabla f(\mathbf{x}^{[t]})\|^2 \\
&= f(\mathbf{x}^{[t]}) - \alpha \|\nabla f(\mathbf{x}^{[t]})\|^2 + \frac{L\alpha^2}{2}\|\nabla f(\mathbf{x}^{[t]})\|^2 \\
&\leq f(\mathbf{x}^{[t]}) - \frac{\alpha}{2}\|\nabla f(\mathbf{x}^{[t]})\|^2
\end{aligned}
$$



Note: $\alpha \leq 1/L$ yields $L\alpha^2 \leq \alpha$

- $\|\nabla f(\mathbf{x}^{[t]})\|^2 > 0$ unless $\nabla f(\mathbf{x}) = \mathbf{0}$
- f **strictly decreases** with each GD iteration until optimum reached
- Descent lemma yields bound on **guaranteed progress** if $\alpha \leq 1/L$ (explains why GD may diverge if step sizes too large)

ONE STEP ERROR BOUND



Again, first order condition of convexity gives

$$f(\mathbf{x}^{[t]}) - f(\mathbf{x}^*) \leq \nabla f(\mathbf{x}^{[t]})^\top (\mathbf{x}^{[t]} - \mathbf{x}^*).$$

This and the descent lemma yields

$$\begin{aligned}
f(\mathbf{x}^{[t+1]}) - f(\mathbf{x}^*) &\leq f(\mathbf{x}^{[t]}) - \frac{\alpha}{2}\|\nabla f(\mathbf{x}^{[t]})\|^2 - f(\mathbf{x}^*) \\
&= f(\mathbf{x}^{[t]}) - f(\mathbf{x}^*) - \frac{\alpha}{2}\|\nabla f(\mathbf{x}^{[t]})\|^2 \\
&\leq \nabla f(\mathbf{x}^{[t]})^\top (\mathbf{x}^{[t]} - \mathbf{x}^*) - \frac{\alpha}{2}\|\nabla f(\mathbf{x}^{[t]})\|^2 \\
&= \frac{1}{2\alpha}\left(\|\mathbf{x}^{[t]} - \mathbf{x}^*\|^2 - \|\mathbf{x}^{[t]} - \mathbf{x}^* - \alpha\nabla f(\mathbf{x}^{[t]})\|^2\right) \\
&= \frac{1}{2\alpha}\left(\|\mathbf{x}^{[t]} - \mathbf{x}^*\|^2 - \|\mathbf{x}^{[t+1]} - \mathbf{x}^*\|^2\right)
\end{aligned}$$

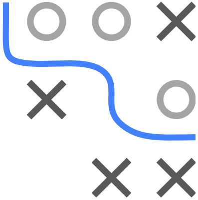
HW: expand and simplify

Note: Line 3 $\rightarrow$ 4 is hard to see (just expand line 4).

FINALIZATION

Summing over iterations yields → because each next iter yields better approx.

$$
\begin{aligned}
k(f(\mathbf{x}^{[k]}) - f(\mathbf{x}^*)) &\leq \sum_{t=1}^{k} [f(\mathbf{x}^{[t]}) - f(\mathbf{x}^*)] \\
&\leq \sum_{t=1}^{k} \frac{1}{2\alpha} \left[\|\mathbf{x}^{[t-1]} - \mathbf{x}^*\|^2 - \|\mathbf{x}^{[t]} - \mathbf{x}^*\|^2 \right] \\
&= \frac{1}{2\alpha} \left(\|\mathbf{x}^{[0]} - \mathbf{x}^*\|^2 - \|\mathbf{x}^{[k]} - \mathbf{x}^*\|^2 \right) \\
&\leq \frac{1}{2\alpha} \left(\|\mathbf{x}^{[0]} - \mathbf{x}^*\|^2 \right).
\end{aligned}
$$

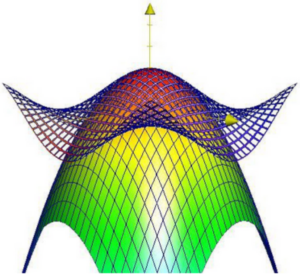


Arguments: Descent lemma (line 1). Telescoping sum (line 2 $\rightarrow$ 3).

$$
\boxed{f(\mathbf{x}^{[\cancel{t-1}\,k]}) - f(\mathbf{x}^*) \leq \frac{\|\mathbf{x}^{[0]} - \mathbf{x}^*\|^2}{2\alpha k}}
$$

Optimization in Machine Learning

First order methods
Weaknesses of GD – Curvature



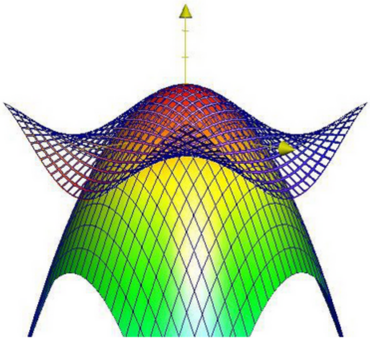
Learning goals

- Effects of curvature
- Step size effect in GD

REMINDER: LOCAL QUADRATIC GEOMETRY

Locally approximate smooth function by quadratic Taylor polynomial:

$$f(\mathbf{x}) \approx f(\tilde{\boldsymbol{x}}) + \nabla f(\tilde{\boldsymbol{x}})^\top(\mathbf{x} - \tilde{\boldsymbol{x}}) + \frac{1}{2}(\mathbf{x} - \tilde{\boldsymbol{x}})^\top \nabla^2 f(\tilde{\boldsymbol{x}})(\mathbf{x} - \tilde{\boldsymbol{x}})$$



Source: `daniloroccatano.blog`.

REMINDER: LOCAL QUADRATIC GEOMETRY

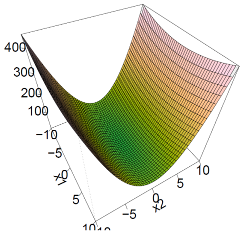
Study Hessian $\mathbf{H} = \nabla^2 f(\mathbf{x}^{[t]})$ in GD to discuss effect of curvature

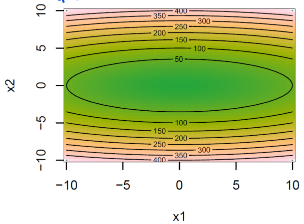
Recall for quadratic forms:

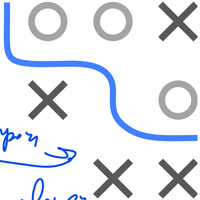
- Eigenvector $\mathbf{v}_{\text{max}}$ ($\mathbf{v}_{\text{min}}$) is direction of largest (smallest) curvature
- $\mathbf{H}$ called ill-conditioned if $\kappa(\mathbf{H}) = |\lambda_{\text{max}}|/|\lambda_{\text{min}}|$ is large

condition number

sounding thing







the sharper

the closer contour lines

$\hat{\theta} = (X^T X)^{-1} X^T y$

$(X^T X)\,\hat{\theta} = X^T y$

Sys. lin equations

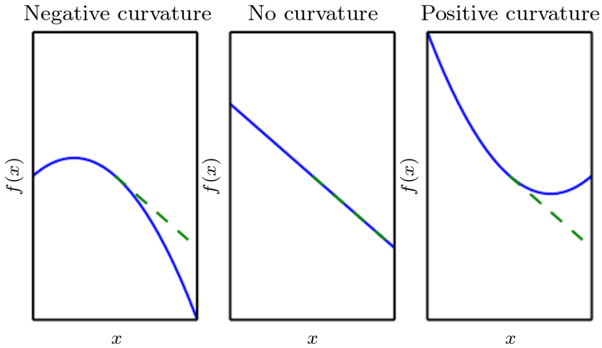
S.L.E.

float – 4 byte

double – 8 byte

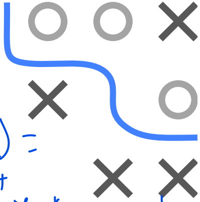
EFFECTS OF CURVATURE

Intuitively, curvature determines reliability of a GD step



Quadratic objective f (blue) with gradient approximation (dashed green). **Left:** f decreases faster than ∇f predicts. **Center:** ∇f predicts decrease correctly. **Right:** f decreases more slowly than ∇f predicts. (Source: Goodfellow et al., 2016)

CURVATURE AND STEP SIZE IN GD



Worst case: H is ill-conditioned. What does this mean for GD?

- Quadratic Taylor polynomial of f around $\tilde{\mathbf{x}}$ (with gradient $\mathbf{g} = \nabla f$)

$$f(\mathbf{x}) \approx f(\tilde{\mathbf{x}}) + (\mathbf{x} - \tilde{\mathbf{x}})^\top \mathbf{g} + \frac{1}{2}(\mathbf{x} - \tilde{\mathbf{x}})^\top \mathbf{H}(\mathbf{x} - \tilde{\mathbf{x}})$$

plug x

- GD step with step size $\alpha > 0$ yields

$$f(\tilde{\boldsymbol{x}} - \alpha \mathbf{g}) \approx f(\tilde{\boldsymbol{x}}) - \alpha \mathbf{g}^\top \mathbf{g} + \frac{1}{2}\alpha^2 \mathbf{g}^\top \boldsymbol{H} \mathbf{g}$$

$\nabla_\alpha f(\tilde{x} - \alpha g) = -g^T g + \dots = 0$

$\alpha^* = \frac{g^T g}{g^T H g}$

- If $\mathbf{g}^\top \boldsymbol{H}\mathbf{g} > 0$, we can solve for optimal step size α^*:

$$\alpha^* = \frac{\mathbf{g}^\top \mathbf{g}}{\mathbf{g}^\top \boldsymbol{H} \mathbf{g}}$$

if ☆ < 0 would α^* be the worst α?

CURVATURE AND STEP SIZE IN GD

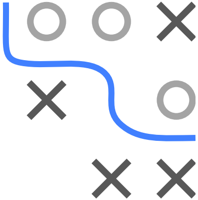
- If $\mathbf{g}$ points along $\mathbf{v}_{\max}$ (largest curvature), optimal step size is

because $Av = \lambda v$

$$\alpha^* = \frac{\mathbf{g}^\top \mathbf{g}}{\mathbf{g}^\top \mathbf{H} \mathbf{g}} = \frac{\mathbf{g}^\top \mathbf{g}}{\lambda_{\max} \mathbf{g}^\top \mathbf{g}} = \frac{1}{\lambda_{\max}}.$$

check chapter 1

$\Rightarrow$ *Large* step sizes can be problematic.



- If $\mathbf{g}$ points along $\mathbf{v}_{\min}$ (smallest curvature), then analogously

$$\alpha^* = \frac{1}{\lambda_{\min}}.$$

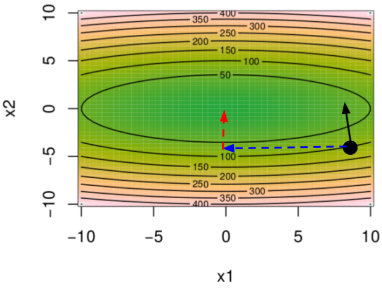
$\Rightarrow$ *Small* step sizes can be problematic.

- **Ideally**: Perform large step along $\mathbf{v}_{\min}$ but small step along $\mathbf{v}_{\max}$.

One dim wants large step, and one small, if $K(H)$ is big then vals in between have a huge range, and when going somewhere else, it's unclear

CURVATURE AND STEP SIZE IN GD

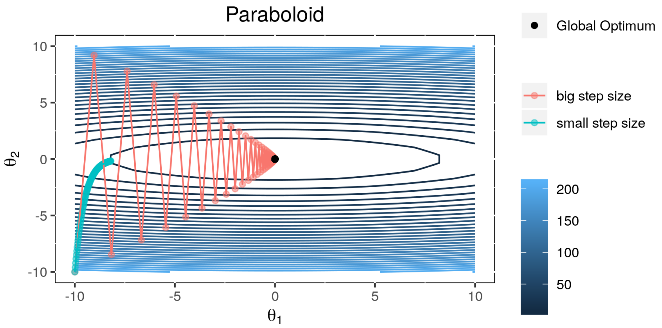
- What if **g** is not aligned with eigenvectors?
- Consider 2D case: Decompose **g** (black) into $\mathbf{v}_{max}$ and $\mathbf{v}_{min}$



- Ideally, perform **large** step along $\mathbf{v}_{min}$ but **small** step along $\mathbf{v}_{max}$
- However, gradient almost only points along $\mathbf{v}_{max}$

CURVATURE AND STEP SIZE IN GD

- GD is not aware of curvatures and can only walk along **g**
- Large step sizes result in “zig-zag” behaviour.
- Small step sizes result in weak performance.



Poorly conditioned quadratic form. GD with large (red) and small (blue) step size. For both, convergence to optimum is slow.

CURVATURE AND STEP SIZE IN GD

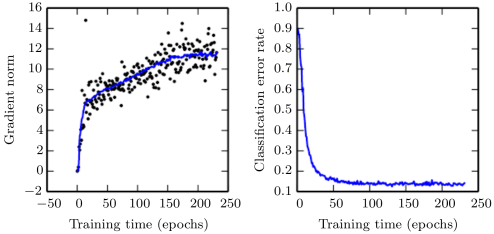
- Large step sizes for ill-conditioned Hessian can even increase

$$f(\tilde{\mathbf{x}} - \alpha \mathbf{g}) \approx f(\tilde{\mathbf{x}}) - \alpha \mathbf{g}^\top \mathbf{g} + \frac{1}{2}\alpha^2 \mathbf{g}^\top \mathbf{H} \mathbf{g}$$

  if

$$\frac{1}{2}\alpha^2 \mathbf{g}^\top \mathbf{H} \mathbf{g} > \alpha \mathbf{g}^\top \mathbf{g} \quad \Leftrightarrow \quad \alpha > 2\frac{\mathbf{g}^\top \mathbf{g}}{\mathbf{g}^\top \mathbf{H} \mathbf{g}}.$$

- Ill-conditioning in practice: Monitor gradient norm and objective



Source: Goodfellow et al., 2016

CURVATURE AND STEP SIZE IN GD

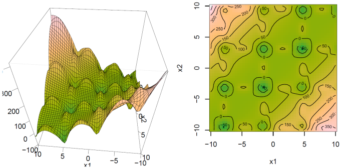
- If gradient norms $\|\mathbf{g}\|$ increase, GD is not converging since $\mathbf{g} \neq 0$.
- Even if $\|\mathbf{g}\|$ increases, objective may stay approximately constant:

$$\underbrace{f(\tilde{\boldsymbol{x}} - \alpha \mathbf{g})}_{\approx \text{ constant}} \approx f(\tilde{\mathbf{x}}) - \alpha \underbrace{\mathbf{g}^\top \mathbf{g}}_{\text{increases}} + \frac{1}{2}\alpha^2 \underbrace{\mathbf{g}^\top \mathbf{H}\mathbf{g}}_{\text{increases}}$$

Optimization in Machine Learning

First order methods

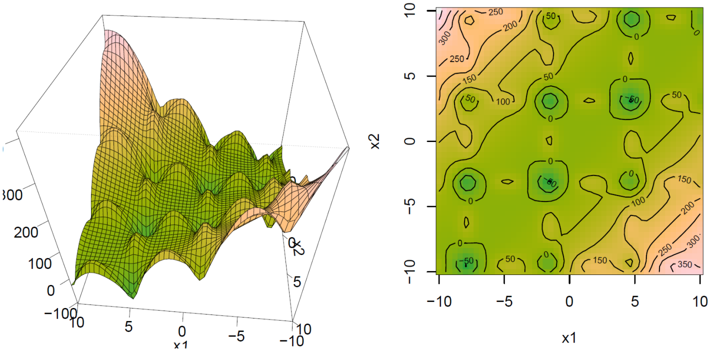
GD – Multimodality and Saddle points



Learning goals

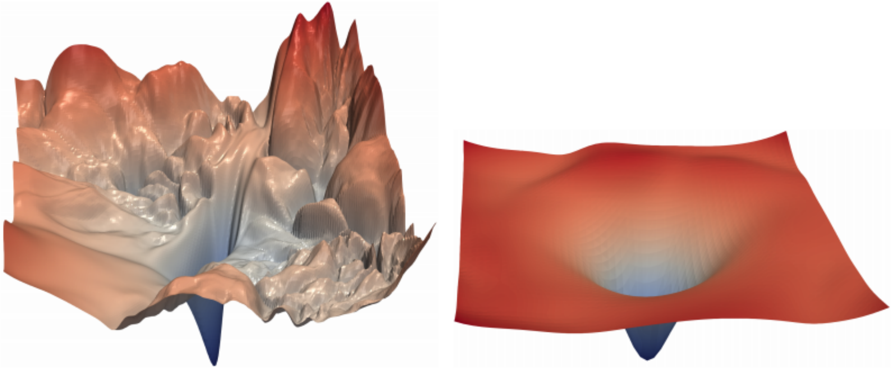
- Multimodality, GD result can be arbitrarily bad
- Saddle points, major problem in NN error landscapes, GD can get stuck or slow crawling

UNIMODAL VS. MULTIMODAL LOSS SURFACES



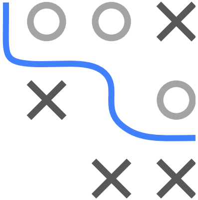
Snippet of a loss surface with many local optima

UNIMODAL VS. MULTIMODAL LOSS SURFACES

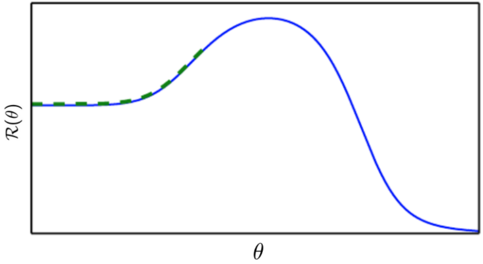


In deep learning, we often find multimodal loss surfaces.
Left: Multimodal loss surface. **Right:** (Nearly) unimodal loss surface.
(Source: Hao Li et al., 2017.

GD: ONLY LOCALLY OPTIMAL MOVES



- GD makes only **locally** optimal moves
- It may move away from the global optimum

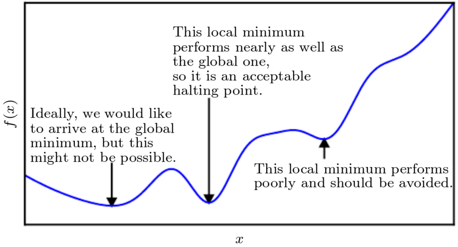


Source: Goodfellow et al., 2016

- Initialization on “wrong” side of the hill results in weak performance
- In higher dimensions, GD may move around the hill (potentially at the cost of longer trajectory and time to convergence)

LOCAL MINIMA

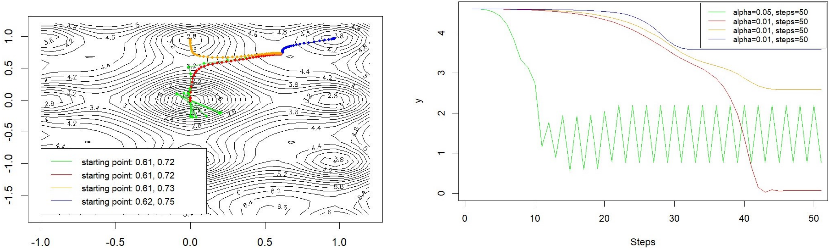
- **In practice:** Only local minima with high value compared to global minimium are problematic.



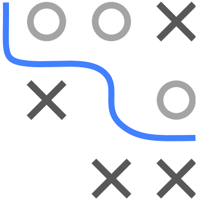
Source: Goodfellow et al., 2016

LOCAL MINIMA

- Small differences in starting point or step size can lead to huge differences in the reached minimum or even to non-convergence



(Non-)Converging gradient descent for Ackley function

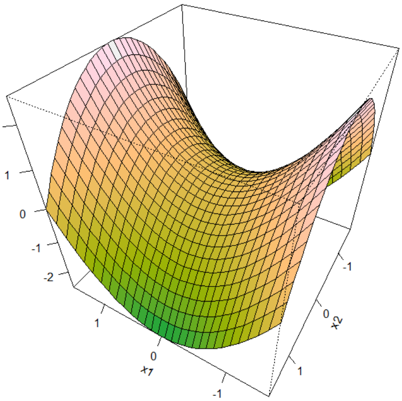


GD AT SADDLE POINTS

Example:

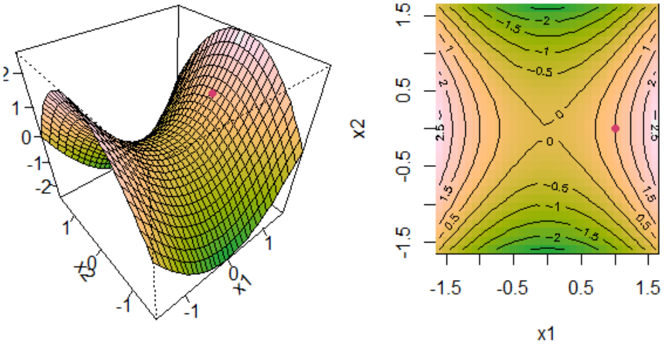
$$
\begin{aligned}
f(x_1, x_2) &= x_1^2 - x_2^2 \\
\nabla f(x_1, x_2) &= (2x_1, -2x_2)^\top \\
\boldsymbol{H} &= \begin{pmatrix} 2 & 0 \\ 0 & -2 \end{pmatrix}
\end{aligned}
$$

- Along x_1, curvature is positive ($\lambda_1 = 2 > 0$).
- Along x_2, curvature is negative ($\lambda_2 = -2 < 0$).



EXAMPLE: SADDLE POINT WITH GD

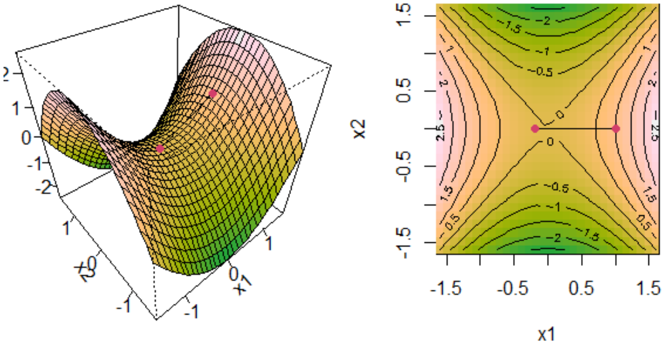
- How do saddle points impair optimization?
- Gradient-based algorithms **might** get stuck in saddle points



Red dot: Starting location

EXAMPLE: SADDLE POINT WITH GD

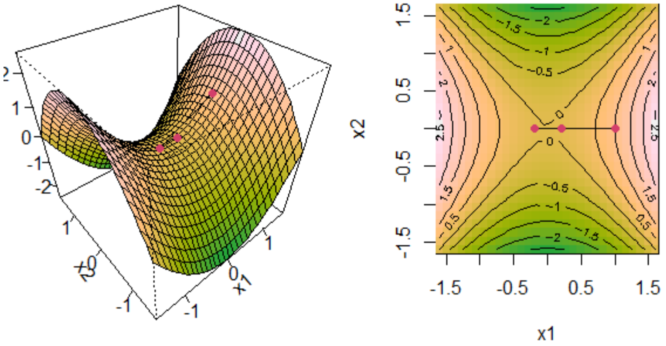
- How do saddle points impair optimization?
- Gradient-based algorithms **might** get stuck in saddle points



Step 1 ...

EXAMPLE: SADDLE POINT WITH GD

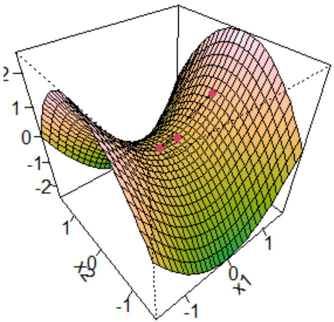
- How do saddle points impair optimization?
- Gradient-based algorithms **might** get stuck in saddle points

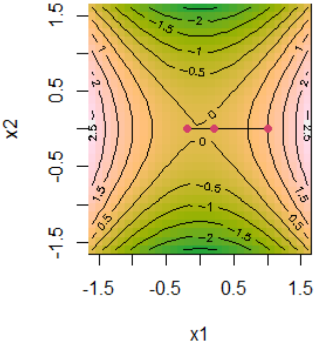


... Step 2 ...

EXAMPLE: SADDLE POINT WITH GD

- How do saddle points impair optimization?
- Gradient-based algorithms **might** get stuck in saddle points

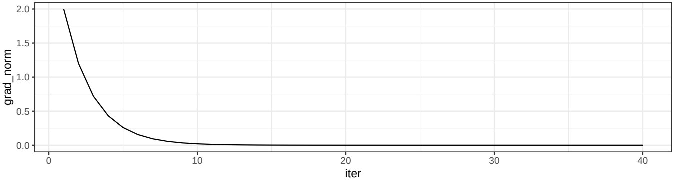




... Step 10 ... got stuck and cannot escape saddle point

EXAMPLE: SADDLE POINT WITH GD

- How do saddle points impair optimization?
- Gradient-based algorithms **might** get stuck in saddle points



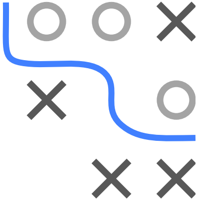
... Step 10 ... got stuck and cannot escape saddle point

SADDLE POINTS IN NEURAL NETWORKS

- For the empirical risk $\mathcal{R} : \mathbb{R}^d \to \mathbb{R}$ of a neural network, the expected ratio of the number of saddle points to local minima typically grows exponentially with d → Is there a proof?

- In other words: Networks with more parameters (deeper networks or larger layers) exhibit a lot more saddle points than local minima

- **Reason:** Hessian at local minimum has only positive eigenvalues. Hessian at saddle point has positive and negative eigenvalues.

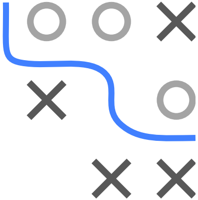


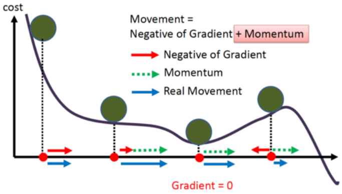
SADDLE POINTS IN NEURAL NETWORKS

- Imagine the sign of each eigenvalue is generated by coin flipping:
  - In a single dimension, it is easy to obtain a local minimum (e.g. “head” means positive eigenvalue).
  - In an *m*-dimensional space, it is exponentially unlikely that all *m* coin tosses will be head.
- A property of many random functions is that eigenvalues of the Hessian become more likely to be positive in regions of lower cost.
- For the coin flipping example, this means we are more likely to have heads *m* times if we are at a critical point with low cost.
- That means in particular that local minima are much more likely to have low cost than high cost and critical points with high cost are far more likely to be saddle points.
- “Saddle points are surrounded by high error plateaus that can dramatically slow down learning, and give the illusory impression of the existence of a local minimum” (Dauphin et al. (2014)).

Optimization in Machine Learning

First order methods
GD with Momentum

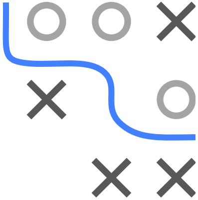




Learning goals

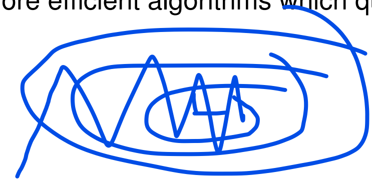
- Recap of GD problems
- Momentum definition
- Unrolling formula
- Examples
- Nesterov

RECAP: WEAKNESSES OF GRADIENT DESCENT



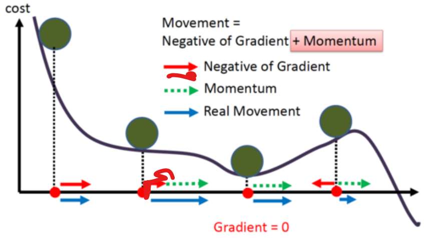
- **Zig-zagging behavior:** For ill-conditioned problems, GD moves with a zig-zag course to the optimum, since the gradient points approximately orthogonal in the shortest direction to the minimum.
- **Slow crawling:** may vanish rapidly close to stationary points (e.g. saddle points) and hence also slows down progress.
- **Trapped in stationary points:** In some functions GD converges to stationary points (e.g. saddle points) since gradient on all sides is fairly flat and the step size is too small to pass this flat part.

Aim: More efficient algorithms which quickly reach the minimum.



GD WITH MOMENTUM

- **Idea:** “Velocity” $\boldsymbol{\nu}$: Increasing if successive gradients point in the same direction but decreasing if they point in opposite directions



Source: Khandewal, *GD with Momentum, RMSprop and Adam Optimizer*, 2020.

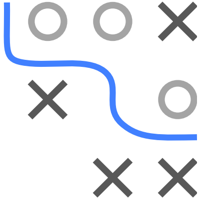
- $\boldsymbol{\nu}$ is weighted moving average of previous gradients:

$$\boldsymbol{\nu}^{[t+1]} = \varphi \boldsymbol{\nu}^{[t]} - \alpha \nabla f(\mathbf{x}^{[t]})$$
$$\mathbf{x}^{[t+1]} = \mathbf{x}^{[t]} + \boldsymbol{\nu}^{[t+1]}$$

(control parameter)

- $\varphi \in [0, 1)$ is additional hyperparameter

GD WITH MOMENTUM



- Length of a single step depends on how large and aligned a sequence of gradients is
- Length of a single step grows if many successive gradients point in the same direction
- φ determines how strongly previous gradients are included in $\boldsymbol{\nu}$
- Common values for φ are 0.5, 0.9 and even 0.99
- In general, the larger φ is in relation to α, the more strongly previous gradients influence the current direction
- **Special case** $\varphi = 0$: "vanilla" gradient descent
- **Intuition:** GD with "short term memory" for the direction of motion

$v^{t+1} = \varphi v^t - \alpha \cdot w^t$

$\varphi = 0$

MOMENTUM: ANALYSIS

$$\boldsymbol{\nu}^{[1]} = \varphi \boldsymbol{\nu}^{[0]} - \alpha \nabla f(\mathbf{x}^{[0]})$$

$$\mathbf{x}^{[1]} = \mathbf{x}^{[0]} + \varphi \boldsymbol{\nu}^{[0]} - \alpha \nabla f(\mathbf{x}^{[0]})$$

$X^{10} =$

$X^2 = X^1 + \varphi v^1 + \alpha \nabla \varphi x^1$

MOMENTUM: ANALYSIS

$$\boldsymbol{\nu}^{[1]} = \varphi\boldsymbol{\nu}^{[0]} - \alpha\nabla f(\mathbf{x}^{[0]})$$

$$\mathbf{x}^{[1]} = \mathbf{x}^{[0]} + \varphi\boldsymbol{\nu}^{[0]} - \alpha\nabla f(\mathbf{x}^{[0]})$$

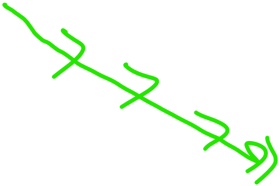
$$\begin{aligned}\boldsymbol{\nu}^{[2]} &= \varphi\boldsymbol{\nu}^{[1]} - \alpha\nabla f(\mathbf{x}^{[1]}) \\ &= \varphi(\varphi\boldsymbol{\nu}^{[0]} - \alpha\nabla f(\mathbf{x}^{[0]})) - \alpha\nabla f(\mathbf{x}^{[1]})\end{aligned}$$

$$\mathbf{x}^{[2]} = \mathbf{x}^{[1]} + \varphi(\varphi\boldsymbol{\nu}^{[0]} - \alpha\nabla f(\mathbf{x}^{[0]})) - \alpha\nabla f(\mathbf{x}^{[1]})$$

MOMENTUM: ANALYSIS

$$
\begin{aligned}
\boldsymbol{\nu}^{[1]} &= \varphi \boldsymbol{\nu}^{[0]} - \alpha \nabla f(\mathbf{x}^{[0]}) \\
\mathbf{x}^{[1]} &= \mathbf{x}^{[0]} + \varphi \boldsymbol{\nu}^{[0]} - \alpha \nabla f(\mathbf{x}^{[0]}) \\
\boldsymbol{\nu}^{[2]} &= \varphi \boldsymbol{\nu}^{[1]} - \alpha \nabla f(\mathbf{x}^{[1]}) \\
&= \varphi(\varphi \boldsymbol{\nu}^{[0]} - \alpha \nabla f(\mathbf{x}^{[0]})) - \alpha \nabla f(\mathbf{x}^{[1]}) \\
\mathbf{x}^{[2]} &= \mathbf{x}^{[1]} + \varphi(\varphi \boldsymbol{\nu}^{[0]} - \alpha \nabla f(\mathbf{x}^{[0]})) - \alpha \nabla f(\mathbf{x}^{[1]}) \\
\boldsymbol{\nu}^{[3]} &= \varphi \boldsymbol{\nu}^{[2]} - \alpha \nabla f(\mathbf{x}^{[2]}) \\
&= \varphi(\varphi(\varphi \boldsymbol{\nu}^{[0]} - \alpha \nabla f(\mathbf{x}^{[0]})) - \alpha \nabla f(\mathbf{x}^{[1]})) - \alpha \nabla f(\mathbf{x}^{[2]}) \\
\mathbf{x}^{[3]} &= \mathbf{x}^{[2]} + \varphi(\varphi(\varphi \boldsymbol{\nu}^{[0]} - \alpha \nabla f(\mathbf{x}^{[0]})) - \alpha \nabla f(\mathbf{x}^{[1]})) - \alpha \nabla f(\mathbf{x}^{[2]}) \\
&= \mathbf{x}^{[2]} + \varphi^3 \boldsymbol{\nu}^{[0]} - \varphi^2 \alpha \nabla f(\mathbf{x}^{[0]}) - \varphi \alpha \nabla f(\mathbf{x}^{[1]}) - \alpha \nabla f(\mathbf{x}^{[2]}) \\
&= \mathbf{x}^{[2]} - \alpha(\varphi^2 \nabla f(\mathbf{x}^{[0]}) + \varphi^1 \nabla f(\mathbf{x}^{[1]}) + \varphi^0 \nabla f(\mathbf{x}^{[2]})) + \varphi^3 \boldsymbol{\nu}^{[0]}
\end{aligned}
$$

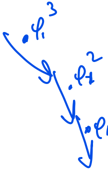
MOMENTUM: ANALYSIS



$$\boldsymbol{\nu}^{[1]} = \varphi\boldsymbol{\nu}^{[0]} - \alpha\nabla f(\mathbf{x}^{[0]})$$

$$\mathbf{x}^{[1]} = \mathbf{x}^{[0]} + \varphi\boldsymbol{\nu}^{[0]} - \alpha\nabla f(\mathbf{x}^{[0]})$$

$$\boldsymbol{\nu}^{[2]} = \varphi\boldsymbol{\nu}^{[1]} - \alpha\nabla f(\mathbf{x}^{[1]})$$
$$= \varphi(\varphi\boldsymbol{\nu}^{[0]} - \alpha\nabla f(\mathbf{x}^{[0]})) - \alpha\nabla f(\mathbf{x}^{[1]})$$



the older the step the less weight it gets

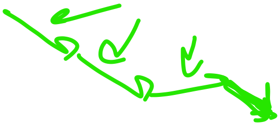
$$\mathbf{x}^{[2]} = \mathbf{x}^{[1]} + \varphi(\varphi\boldsymbol{\nu}^{[0]} - \alpha\nabla f(\mathbf{x}^{[0]})) - \alpha\nabla f(\mathbf{x}^{[1]})$$

$$\boldsymbol{\nu}^{[3]} = \varphi\boldsymbol{\nu}^{[2]} - \alpha\nabla f(\mathbf{x}^{[2]})$$
$$= \varphi(\varphi(\varphi\boldsymbol{\nu}^{[0]} - \alpha\nabla f(\mathbf{x}^{[0]})) - \alpha\nabla f(\mathbf{x}^{[1]})) - \alpha\nabla f(\mathbf{x}^{[2]})$$

$$\mathbf{x}^{[3]} = \mathbf{x}^{[2]} + \varphi(\varphi(\varphi\boldsymbol{\nu}^{[0]} - \alpha\nabla f(\mathbf{x}^{[0]})) - \alpha\nabla f(\mathbf{x}^{[1]})) - \alpha\nabla f(\mathbf{x}^{[2]})$$
$$= \mathbf{x}^{[2]} + \varphi^3\boldsymbol{\nu}^{[0]} - \varphi^2\alpha\nabla f(\mathbf{x}^{[0]}) - \varphi\alpha\nabla f(\mathbf{x}^{[1]}) - \alpha\nabla f(\mathbf{x}^{[2]})$$
$$= \mathbf{x}^{[2]} - \alpha(\varphi^2\nabla f(\mathbf{x}^{[0]}) + \varphi^1\nabla f(\mathbf{x}^{[1]}) + \varphi^0\nabla f(\mathbf{x}^{[2]})) + \varphi^3\boldsymbol{\nu}^{[0]}$$

exponential smoothing

$$\mathbf{x}^{[t+1]} = \mathbf{x}^{[t]} - \underbrace{\alpha\sum_{j=0}^{t}\varphi^j\nabla f(\mathbf{x}^{[t-j]})}_{\varphi^j} + \varphi^{t+1}\boldsymbol{\nu}^{[0]}$$



MOMENTUM: INTUITION

Suppose momentum always observes the same gradient $\nabla f(\mathbf{x}^{[t]})$:

$\varphi < \upsilon$

$$\begin{aligned}
\mathbf{x}^{[t+1]} &= \mathbf{x}^{[t]} - \alpha \sum_{j=0}^{t} \varphi^j \nabla f(\mathbf{x}^{[j]}) + \varphi^{t+1} \boldsymbol{\nu}^{[0]} \\
&= \mathbf{x}^{[t]} - \alpha \nabla f(\mathbf{x}^{[t]}) \sum_{j=0}^{t} \varphi^j + \varphi^{t+1} \boldsymbol{\nu}^{[0]} \\
&= \mathbf{x}^{[t]} - \alpha \nabla f(\mathbf{x}^{[t]}) \frac{1 - \varphi^{t+1}}{1 - \varphi} + \varphi^{t+1} \boldsymbol{\nu}^{[0]} \\
&\to \mathbf{x}^{[t]} - \alpha \nabla f(\mathbf{x}^{[t]}) \frac{1}{1 - \varphi} \qquad \text{for } t \to \infty.
\end{aligned}$$

$t \to \infty$ 0

$$\frac{\alpha}{1-\varphi} \approx 10\alpha$$

Momentum accelerates along $-\nabla f(\mathbf{x}^{[t]})$ to terminal velocity yielding step size $\alpha/(1 - \varphi)$.

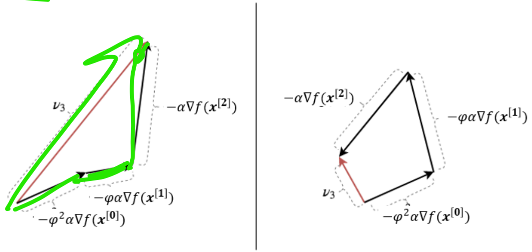
→ we get acceleration: this is equivalent to changing α step size

Example: Momentum with $\varphi = 0.9$ corresponds to a tenfold increase in original step size α compared to vanilla gradient descent

MOMENTUM: INTUITION

Vector $\boldsymbol{\nu}^{[3]}$ (for $\boldsymbol{\nu}^{[0]} = 0$):

$$\begin{aligned}
\boldsymbol{\nu}^{[3]} &= \varphi(\varphi(\varphi\boldsymbol{\nu}^{[0]} - \alpha\nabla f(\mathbf{x}^{[0]})) - \alpha\nabla f(\mathbf{x}^{[1]})) - \alpha\nabla f(\mathbf{x}^{[2]}) \\
&= -\varphi^2\alpha\nabla f(\mathbf{x}^{[0]}) - \varphi\alpha\nabla f(\mathbf{x}^{[1]}) - \alpha\nabla f(\mathbf{x}^{[2]})
\end{aligned}$$

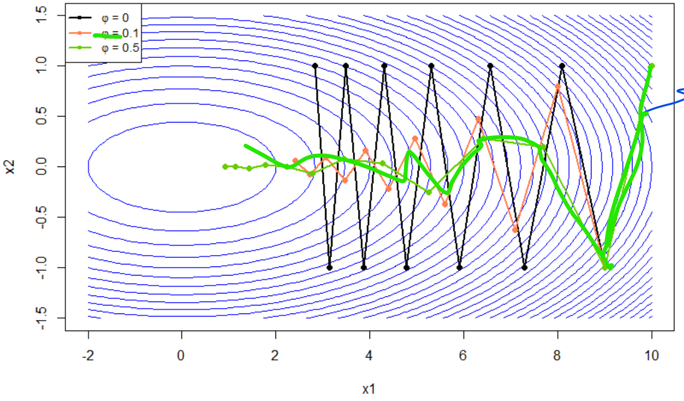


Successive gradients pointing in same/different directions increase/decrease velocity.
Further geometric intuitions and detailed explanations:
`https://distill.pub/2017/momentum/`

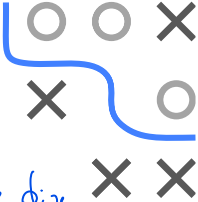
GD WITH MOMENTUM: ZIG-ZAG BEHAVIOUR

Consider a two-dimensional quadratic form $f(\mathbf{x}) = x_1^2/2 + 10x_2$.

Let $\mathbf{x}^{[0]} = (10, 1)^\top$ and $\alpha = 0.1$.



all 3 same dir



GD shows stronger zig-zag behaviour than GD with momentum.

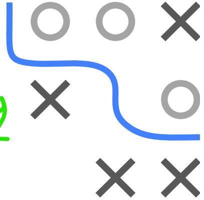
GD WITH MOMENTUM: ZIG-ZAG BEHAVIOUR

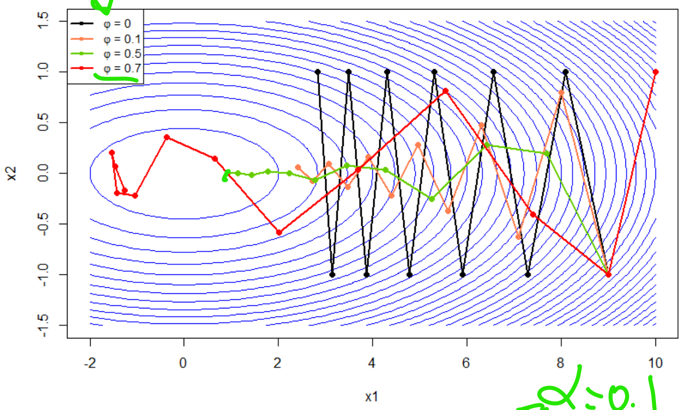
Caution:

- If momentum is too high, minimum is possibly missed
- We might go back and forth around or between local minima

α

$\varphi = 0.4$





φ too close to 1 can't guarantee convergence

$\theta_0 + \theta_1 x$

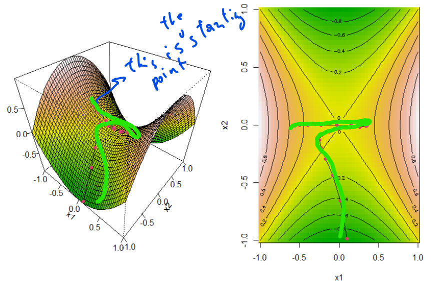
$\alpha = 0.1$

GD WITH MOMENTUM: SADDLE POINTS

Consider the two-dimensional quadratic form $f(\mathbf{x}) = x_1^2 - x_2^2$ with a saddle point at $(0, 0)^\top$.

Let $\mathbf{x}^{[0]} = (-1/2, 10^{-3})^\top$ and $\alpha = 0.1$.

this is the starting point



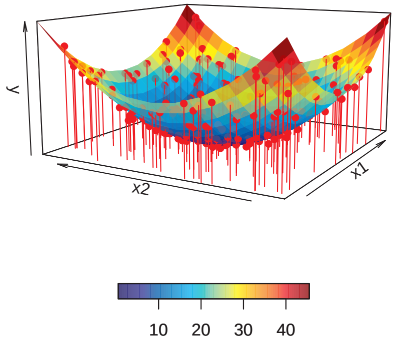
GD was slowing down at the saddle point (vanishing gradient).
GD with momentum “breaks out” of the saddle point and moves on.

ERM FOR NN WITH GD

Let $\mathcal{D} = \left(\left(\mathbf{x}^{(1)}, y^{(1)}\right), \ldots, \left(\mathbf{x}^{(n)}, y^{(n)}\right) \right)$, with $y = x_1^2 + x_2^2$ and minimize

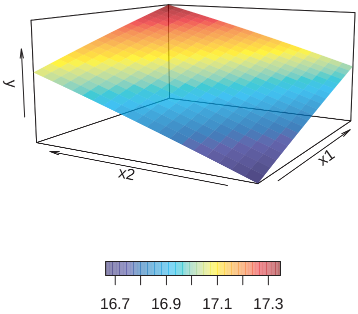
$$\mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) = \sum_{i=1}^{n} \left(f(\mathbf{x} \mid \boldsymbol{\theta}) - y^{(i)} \right)^2$$

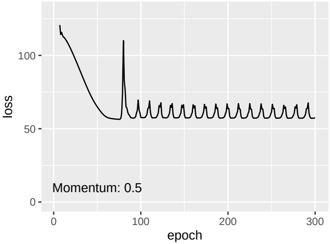
where $f(\mathbf{x} \mid \boldsymbol{\theta})$ is a neural network with 2 hidden layers (2 units each).

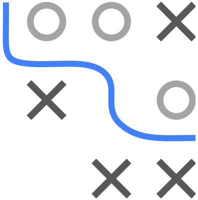


ERM FOR NN WITH GD

After 10 iters of GD:

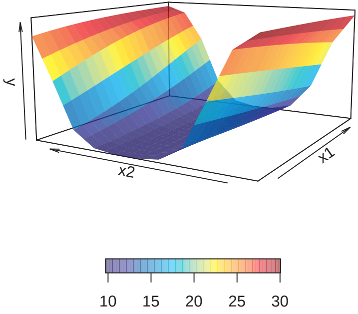


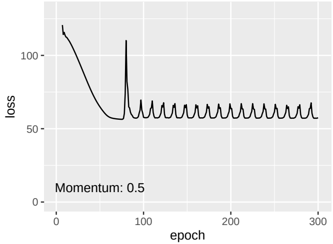


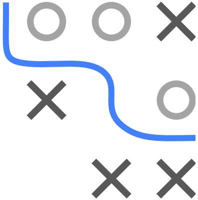


ERM FOR NN WITH GD

After 100 iters of GD:

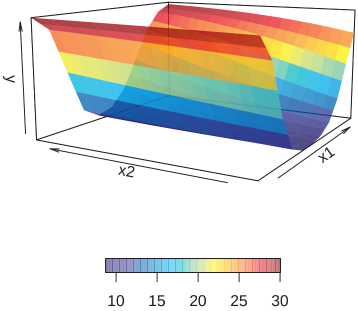


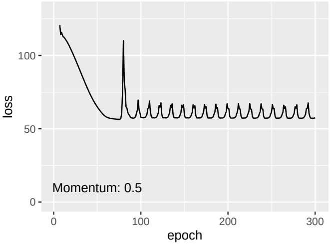


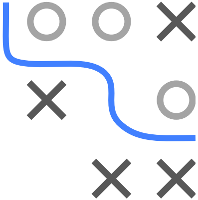


ERM FOR NN WITH GD

After 300 iters of GD:

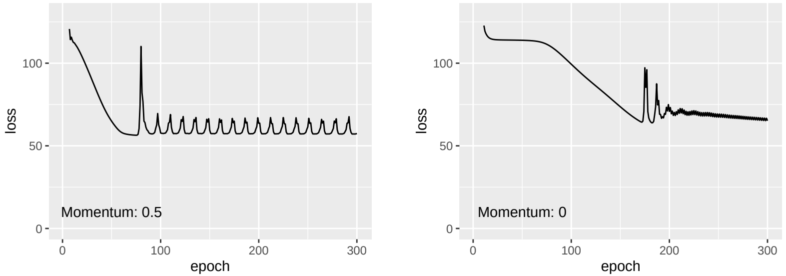






ERM FOR NN WITH GD

Gradient Descent with and without momentum

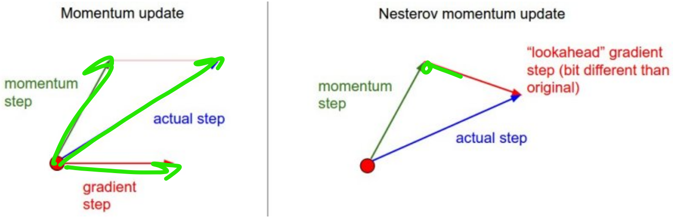


NESTEROV ACCELERATED GRADIENT

- Slightly modified version: **Nesterov accelerated gradient**
- Stronger theoretical convergence guarantees for convex functions
- Avoid moving back and forth near optima

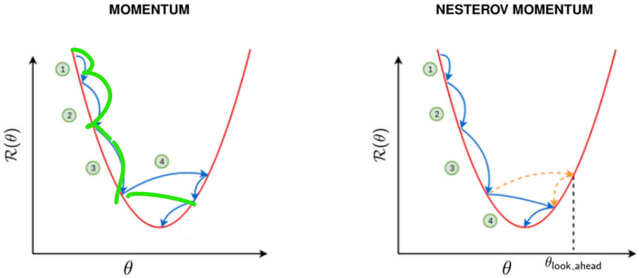
$$\boldsymbol{\nu}^{[t+1]} = \varphi\boldsymbol{\nu}^{[t]} - \alpha\nabla f(\mathbf{x}^{[t]} + \varphi\boldsymbol{\nu}^{[t]})$$
$$\mathbf{x}^{[t+1]} = \mathbf{x}^{[t]} + \boldsymbol{\nu}^{[t+1]}$$

ToDo



Nesterov momentum update evaluates gradient at the "look-ahead" position. (Source: `https://cs231n.github.io/neural-networks-3/`)

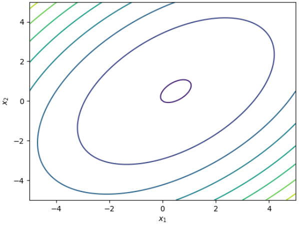
MOMENTUM VS. NESTEROV



GD with momentum (**left**) vs. GD with Nesterov momentum (**right**).
Near minima, momentum makes a large step due to gradient history.
Nesterov momentum “looks ahead” and reduces effect of gradient history.
(Source: Chandra, 2015)

Optimization in Machine Learning

First order methods
GD on quadratic forms



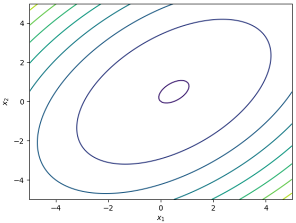
Learning goals

- Eigendecomposition of quadratic forms
- GD steps in eigenspace

QUADRATIC FORMS & GD

- We consider the quadratic function $q(\mathbf{x}) = \mathbf{x}^\top \mathbf{A}\mathbf{x} - \mathbf{b}^\top \mathbf{x}$.
- We assume that Hessian $\mathbf{H} = 2\mathbf{A}$ has full rank
- Optimal solution is $\mathbf{x}^* = \frac{1}{2}\mathbf{A}^{-1}\mathbf{b}$
- As $\nabla q(\mathbf{x}) = 2\mathbf{A}\mathbf{x} - \mathbf{b}$, iterations of gradient descent are

$$\mathbf{x}^{[t+1]} = \mathbf{x}^{[t]} - \alpha(2\mathbf{A}\mathbf{x}^{[t]} - \mathbf{b})$$

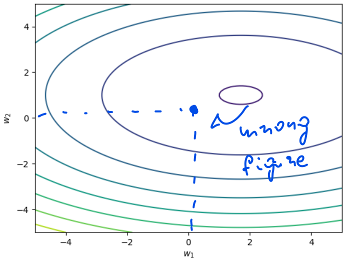


The following slides follow the blog post "Why Momentum Really Works", Distill, 2017.
http://doi.org/10.23915/distill.00006

EIGENDECOMPOSITION OF QUADRATIC FORMS

- We want to work in the coordinate system given by q
- **Recall:** Coordinate system is given by the eigenvectors of $\mathbf{H} = 2\mathbf{A}$
- Eigendecomposition of $\mathbf{A} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^\top$
- $\mathbf{V}$ contains eigenvectors $\mathbf{v}_i$ and $\boldsymbol{\Lambda} = \text{diag}(\lambda_1, ..., \lambda_n)$ eigenvalues
- Change of basis: $\mathbf{w}^{[t]} = \mathbf{V}^\top(\mathbf{x}^{[t]} - \mathbf{x}^*)$

sym $\Rightarrow$ vectors orthogonal



1) move optimum to origin

2) rotate (because V is orthogonal)

GD STEPS IN EIGENSPACE

With $\mathbf{w}^{[t]} = \mathbf{V}^\top(\mathbf{x}^{[t]} - \mathbf{x}^*)$, a single GD step

$$\mathbf{x}^{[t+1]} = \mathbf{x}^{[t]} - \alpha(2\mathbf{A}\mathbf{x}^{[t]} - \mathbf{b})$$

becomes

$$\mathbf{w}^{[t+1]} = \mathbf{w}^{[t]} - 2\alpha\mathbf{\Lambda}\mathbf{w}^{[t]}.$$

Therefore:

$$\begin{aligned} w_i^{[t+1]} &= w_i^{[t]} - 2\alpha\lambda_i w_i^{[t]} \\ &= (1 - 2\alpha\lambda_i) w_i^{[t]} \\ &= \cdots \\ &= (1 - 2\alpha\lambda_i)^{t+1} w_i^{[0]} \end{aligned}$$

GD STEPS IN EIGENSPACE / 2

Proof (for $\mathbf{w}^{[t+1]} = \mathbf{w}^{[t]} - 2\alpha\boldsymbol{\Lambda}\mathbf{w}^{[t]}$):

- A single GD step means

$$\mathbf{x}^{[t+1]} = \mathbf{x}^{[t]} - \alpha(2\mathbf{A}\mathbf{x}^{[t]} - \mathbf{b})$$

subtract and apply

- Then:

$$\mathbf{V}^\top(\mathbf{x}^{[t+1]} - \mathbf{x}^*) = \mathbf{V}^\top(\mathbf{x}^{[t]} - \mathbf{x}^*) - \alpha\mathbf{V}^\top(2\mathbf{A}\mathbf{x}^{[t]} - \mathbf{b})$$

definition

$$\mathbf{w}^{[t+1]} = \mathbf{w}^{[t]} - \alpha\mathbf{V}^\top(2\mathbf{A}\mathbf{x}^{[t]} - \mathbf{b})$$

add and subtract $2Ax^*$

$$\mathbf{w}^{[t+1]} = \mathbf{w}^{[t]} - \alpha\mathbf{V}^\top(2\mathbf{A}(\mathbf{x}^{[t]} - \mathbf{x}^*) + \underbrace{2\mathbf{A}\mathbf{x}^* - \mathbf{b}}_{=0})$$

because optimal solution also gradient at the optimum must be 0

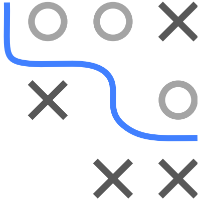
$$= \mathbf{w}^{[t]} - 2\alpha\boldsymbol{\Lambda}\mathbf{V}^\top(\mathbf{x}^{[t]} - \mathbf{x}^*)$$

$$= \mathbf{w}^{[t]} - 2\alpha\boldsymbol{\Lambda}\mathbf{w}^{[t]}$$

$A = V \Lambda V^T$

$\underbrace{V^T V}_{I} \Lambda V^T$

GD ERROR IN ORIGINAL SPACE



- Move back to **original space**:

recursion

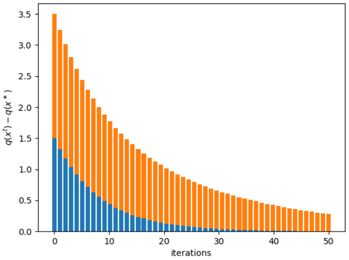
$$\mathbf{x}^{[t]} - \mathbf{x}^* = \mathbf{V}\mathbf{w}^{[t]} = \sum_{i=1}^{d} (1 - 2\alpha\lambda_i)^t w_i^{[0]} \mathbf{v}_i$$

- **Intuition:** Initial error components $w_i^{[0]}$ (in the eigenbasis) decay with rate $1 - 2\alpha\lambda_i$ need to be <1 so we don't diverge
- **Therefore:** For sufficiently small step sizes α, error components along eigenvectors with large eigenvalues decay quickly

GD ERROR IN ORIGINAL SPACE / 2

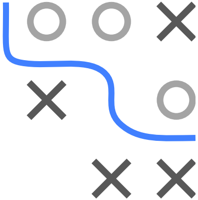
We now consider the contribution of each eigenvector to the total loss

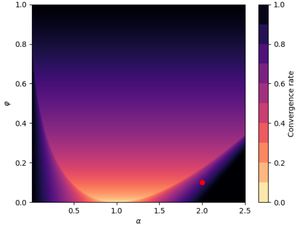
$$q(\mathbf{x}^{[t]}) - q(\mathbf{x}^*) = \frac{1}{2}\sum_{i}^{d}(1 - 2\alpha\lambda_i)^{2t}\lambda_i(w_i^{[0]})^2$$



Optimization in Machine Learning

First order methods
Momentum on quadratic forms





Learning goals

- Momentum update in Eigenspace
- Effect of φ

RECAP: MOMENTUM UPDATE

$$
\begin{aligned}
\boldsymbol{\nu}^{[t+1]} &= \varphi \boldsymbol{\nu}^{[t]} + \alpha \nabla f(\mathbf{x}^{[t]}) \\
\mathbf{x}^{[t+1]} &= \mathbf{x}^{[t]} - \boldsymbol{\nu}^{[t+1]},
\end{aligned}
$$

which simplifies to

$$
\begin{aligned}
\boldsymbol{\nu}^{[t+1]} &= \varphi \boldsymbol{\nu}^{[t]} + \alpha (\mathbf{A}\mathbf{x}^{[t]} - b) \\
\mathbf{x}^{[t+1]} &= \mathbf{x}^{[t]} - \boldsymbol{\nu}^{[t+1]},
\end{aligned}
$$

for the quadratic form.

RECAP: DYNAMICS OF MOMENTUM

Changing the basis as before with $\boldsymbol{w}^{[t]} = \boldsymbol{V}^\top(\mathbf{x}^{[t]} - \mathbf{x}^*)$ and $\boldsymbol{u}^{[t]} = \boldsymbol{V}\boldsymbol{\nu}^{[t]}$, we get the following set of equations, where each component acts independently, although $w_i^{[t]}$ and $u_i^{[t]}$ are coupled:

$$
\begin{aligned}
u_i^{[t+1]} &= \varphi u_i^{[t]} + \alpha\lambda_i w_i^{[t]}, \\
w_i^{[t+1]} &= w_i^{[t]} - u_i^{[t+1]}
\end{aligned}
$$

We rewrite this:

$$
\begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \begin{pmatrix} u_i^{[t+1]} \\ w_i^{[t+1]} \end{pmatrix} = \begin{pmatrix} \varphi & \alpha\lambda_i \\ 0 & 1 \end{pmatrix} \begin{pmatrix} u_i^{[t]} \\ w_i^{[t]} \end{pmatrix}
$$

and invert the matrix on the LHS:

$$
\begin{pmatrix} u_i^{[t+1]} \\ w_i^{[t+1]} \end{pmatrix} = \begin{pmatrix} \varphi & \alpha\lambda_i \\ -\varphi & 1 - \alpha\lambda_i \end{pmatrix} \begin{pmatrix} u_i^{[t]} \\ w_i^{[t]} \end{pmatrix} = R^{t+1} \begin{pmatrix} u_i^0 \\ w_i^0 \end{pmatrix}
$$

RECAP: DYNAMICS OF MOMENTUM / 2

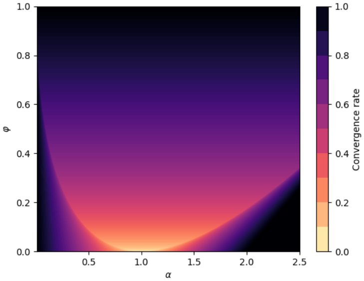
Taking a 2×2 matrix to the t^{th} power reduces to a formula involving the eigenvalues of R, σ_1 and σ_2:

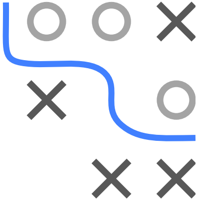
$$R^t = \begin{cases} \sigma_1^t R_1 - \sigma_2^t R_2, & \text{if } \sigma_1 \neq \sigma_2 \\ \sigma_1^t (tR/\sigma_1 - (t-1)I), & \text{if } \sigma_1 = \sigma_2 \end{cases}$$

where $R_j = \frac{R - \sigma_j I}{\sigma_1 - \sigma_2}$.

In contrast to GD, where we got one geometric series, we have two coupled series with real or complex values.

RECAP: DYNAMICS OF MOMENTUM / 3

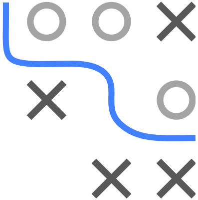


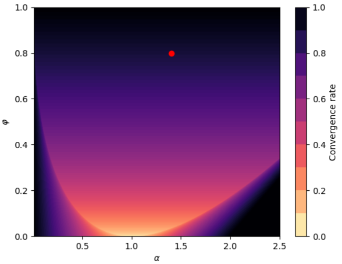


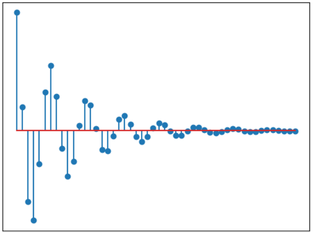
The achieved convergence rate is therefore the slowest of the two, $\max\{|\sigma_1|, |\sigma_2|\}$.

Each region shows a different convergence behavior.

RECAP: DYNAMICS OF MOMENTUM / 4

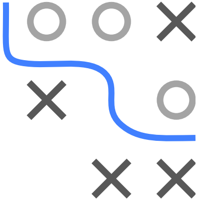


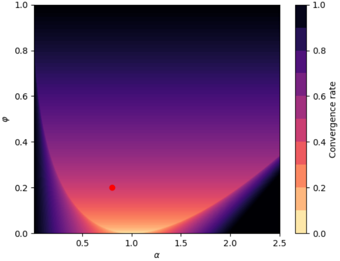


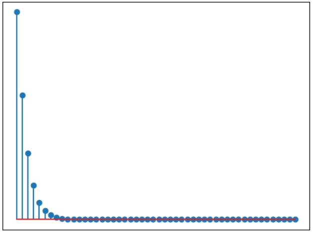


The eigenvalues of R are complex and we see low frequency ripples.

RECAP: DYNAMICS OF MOMENTUM / 5

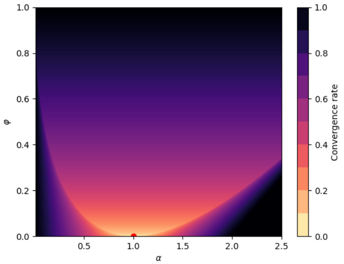


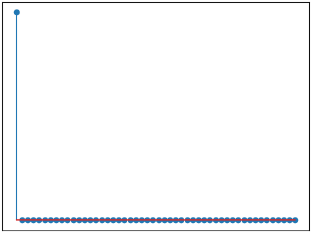




Here, both eigenvalues of R are positive with their norm being less than 1. This behavior resembles gradient descent.

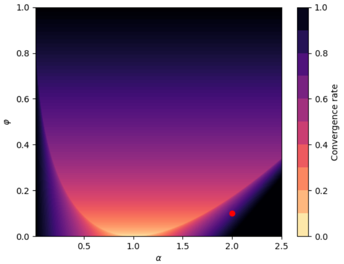
RECAP: DYNAMICS OF MOMENTUM / 6

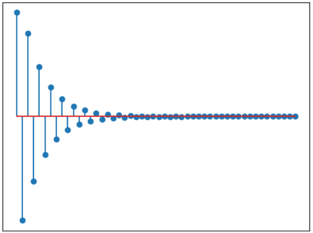




The step size is $\alpha = 1/\lambda_i$ and $\varphi = 0$ - we converge in one step.

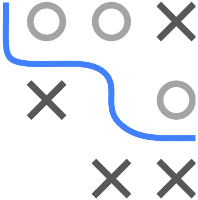
RECAP: DYNAMICS OF MOMENTUM / 7

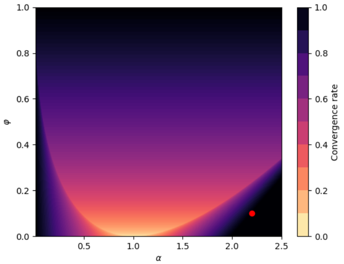


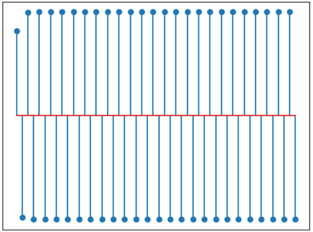


When $\alpha > 1/\lambda_i$, the iterates flip sign every iteration.

RECAP: DYNAMICS OF MOMENTUM / 8



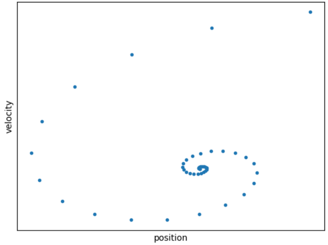




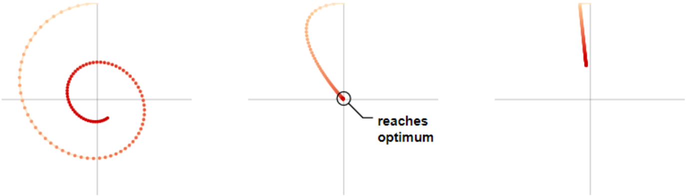
If $\max\{|\sigma_1|, |\sigma_2|\} > 1$, the iterates diverge.

RECAP: DYNAMICS OF MOMENTUM / 9

- Finally, we investigate the role of φ.
- We can think of gradient descent with momentum as a damped harmonic oscillator: a weight on a spring. We pull the weight down and study the path back to the equilibrium in phase space (looking at the position and the velocity).
- Depending on the choice of φ, the rate of return to the equilibrium position is affected.



RECAP: DYNAMICS OF MOMENTUM / 10



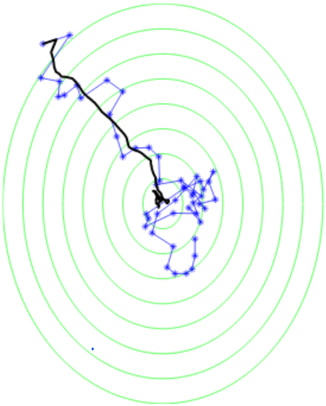
Left: If φ is too large, we are underdamping. The spring oscillates back and forth and misses the optimum.

Middle: The best value of φ lies in the middle.

Right: If φ is too small, we are overdamping, meaning that the spring experiences too much friction and stops before reaching the equilibrium.

Optimization in Machine Learning

First order methods
SGD

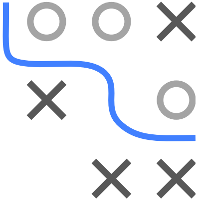


Learning goals

- SGD
- Stochasticity
- Convergence
- Batch size

STOCHASTIC GRADIENT DESCENT

NB: We use g instead of f as objective, bc. f is used as model in ML.



$g : \mathbb{R}^d \to \mathbb{R}$ objective, g **average over functions**:

loss at datapoint i

$$g(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^{n} g_i(\mathbf{x}), \qquad g \text{ and } g_i \text{ smooth}$$

Stochastic gradient descent (SGD) approximates the gradient

$$\begin{aligned} \nabla_{\mathbf{x}}\, g(\mathbf{x}) = \frac{1}{n} \sum_{i=1}^{n} \nabla_{\mathbf{x}}\, g_i(\mathbf{x}) \quad &:= \quad \mathbf{d} \quad \text{by} \\ \frac{1}{|J|} \sum_{i \in J} \nabla_{\mathbf{x}}\, g_i(\mathbf{x}) \quad &:= \quad \hat{\mathbf{d}}, \end{aligned}$$

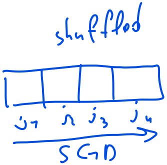
in the limit we'll have normally distrib grads.
CLT helps.
Didn't understand this but sounds cool

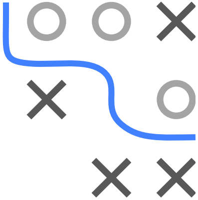
with random subset $J \subset \{1, 2, ..., n\}$ of gradients called **mini-batch**.
This is done e.g. when computing the true gradient is **expensive**.

STOCHASTIC GRADIENT DESCENT / 2

Algorithm Basic SGD pseudo code

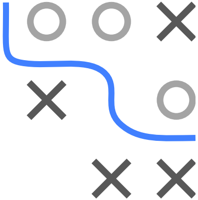
Initialize $\mathbf{x}^{[0]}$, $t = 0$
while stopping criterion not met **do**
    Randomly shuffle indices and partition into minibatches $J_1, \dots, J_K$ of size m
    for $k \in \{1, \dots, K\}$ **do**
        $t \leftarrow t + 1$
        Compute gradient estimate with J_k: $\hat{\mathbf{d}}^{[t]} \leftarrow \frac{1}{m} \sum_{i \in J_k} \nabla_{\mathbf{x}} g_i(\mathbf{x}^{[t-1]})$
        Apply update: $\mathbf{x}^{[t]} \leftarrow \mathbf{x}^{[t-1]} - \alpha \cdot \hat{\mathbf{d}}^{[t]}$
    end for
end while





- Instead of drawing batches randomly we might want to go through the g_i sequentially (unless g_i are sorted in any way)
- Updates are computed faster, but also more stochastic:
  - In the simplest case, batch-size $m := |J_k|$ is set to $m = 1$
  - If n is a billion, computation of update is a billion times faster
  - **But** (later): Convergence rates suffer from stochasticity!

SGD IN ML



In ML, we perform ERM:

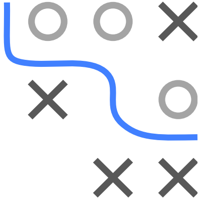
$$\mathcal{R}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \underbrace{L\left(y^{(i)}, f\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right)\right)}_{g_i(\boldsymbol{\theta})}$$

- for a data set

$$\mathcal{D} = \left(\left(\mathbf{x}^{(1)}, y^{(1)}\right), \ldots, \left(\mathbf{x}^{(n)}, y^{(n)}\right)\right)$$

- a loss function $L\left(y, f(\mathbf{x})\right)$, e.g., L2 loss $L\left(y, f(\mathbf{x})\right) = (y - f(\mathbf{x}))^2$,
- and a model class f, e.g., the linear model $f\left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta}\right) = \boldsymbol{\theta}^\top \mathbf{x}$.

SGD IN ML / 2



For large data sets, computing the exact gradient

$$\mathbf{d} = \frac{1}{n} \sum_{i=1}^{n} \nabla_{\boldsymbol{\theta}} L \left(y^{(i)}, f \left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta} \right) \right)$$

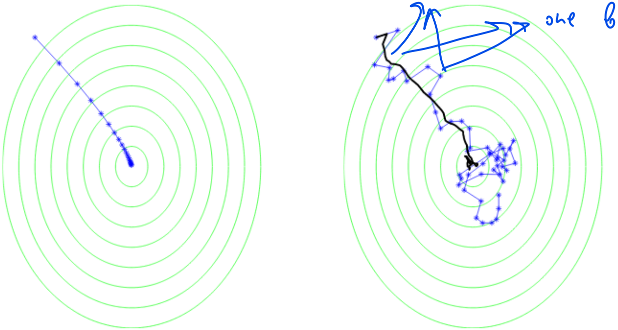
may be expensive or even infeasible to compute and is approximated by

$$\hat{\mathbf{d}} = \frac{1}{m} \sum_{i \in J} \nabla_{\boldsymbol{\theta}} L \left(y^{(i)}, f \left(\mathbf{x}^{(i)} \mid \boldsymbol{\theta} \right) \right),$$

for $J \subset 1, 2, ..., n$ random subset.

NB: Often, maximum size of J technically limited by memory size.

STOCHASTICITY OF SGD



one smaller batch

one bigger

in plots like this

batch size is often set to 1 (which is bad)

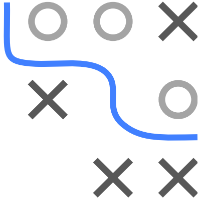
Minimize $g(x_1, x_2) = 1.25(x_1 + 6)^2 + (x_2 - 8)^2$.

Left: GD. **Right:** SGD. Black line shows average value across multiple runs.

(Source: Shalev-Shwartz et al., Understanding Machine Learning, 2014.)

STOCHASTICITY OF SGD / 2

Assume batch size $m = 1$ (statements also apply for larger batches).



- **(Possibly) suboptimal direction:** Approximate gradient $\hat{\mathbf{d}} = \nabla_{\mathbf{x}} g_i(\mathbf{x})$ might point in suboptimal (possibly not even a descent!) direction

Because $\mathbb{E}$ is a linear op

- **Unbiased estimate:** If J drawn i.i.d., approximate gradient $\hat{\mathbf{d}}$ is an unbiased estimate of gradient $\mathbf{d} = \nabla_{\mathbf{x}} g(\mathbf{x}) = \sum_{i=1}^{n} \nabla_{\mathbf{x}} g_i(\mathbf{x})$:

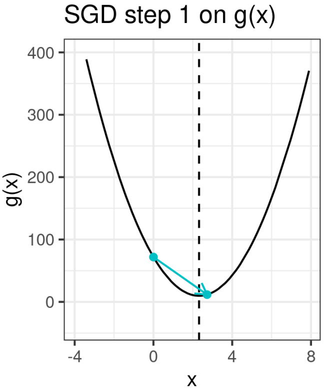
$$
\begin{aligned}
\mathbb{E}_i \left[\nabla_{\mathbf{x}} g_i(\mathbf{x})\right] &= \sum_{i=1}^{n} \nabla_{\mathbf{x}} g_i(\mathbf{x}) \cdot \mathbb{P}(i = i) \\
&= \sum_{i=1}^{n} \nabla_{\mathbf{x}} g_i(\mathbf{x}) \cdot \frac{1}{n} = \nabla_{\mathbf{x}} g(\mathbf{x}).
\end{aligned}
$$

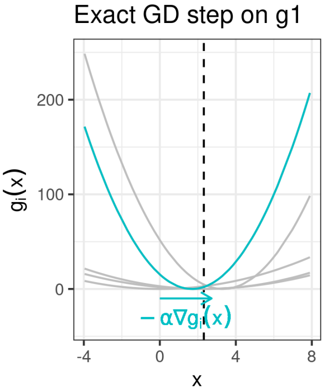
momentum helps a lot

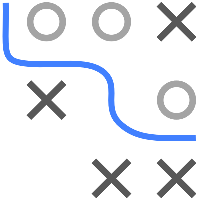
Conclusion: SGD might perform single suboptimal moves, but moves in “right direction” **on average**.

ERRATIC BEHAVIOR OF SGD

Example: $g(\mathbf{x}) = \sum_{i=1}^{5} g_i(\mathbf{x})$, g_i quadratic. Batch size $m = 1$.

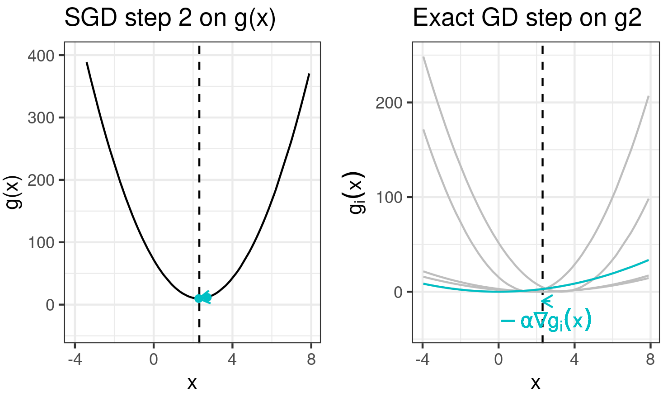






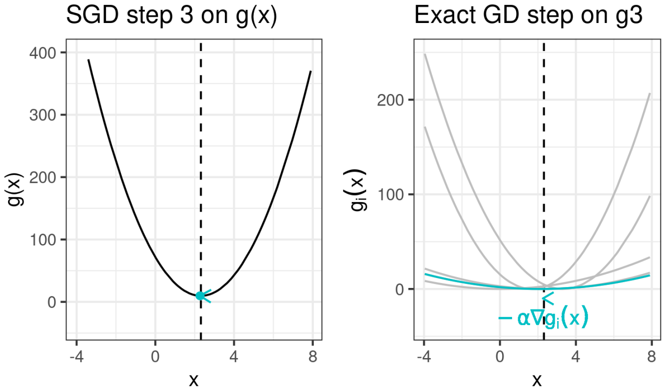
ERRATIC BEHAVIOR OF SGD

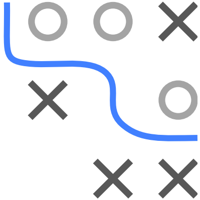
Example: $g(\mathbf{x}) = \sum_{i=1}^{5} g_i(\mathbf{x})$, g_i quadratic. Batch size $m = 1$.



ERRATIC BEHAVIOR OF SGD

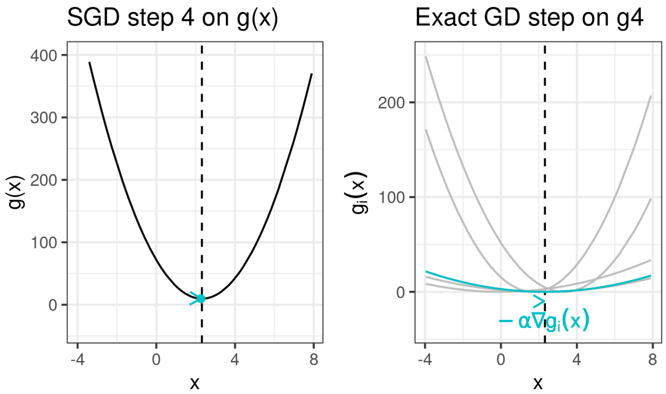
Example: $g(\mathbf{x}) = \sum_{i=1}^{5} g_i(\mathbf{x})$, g_i quadratic. Batch size $m = 1$.





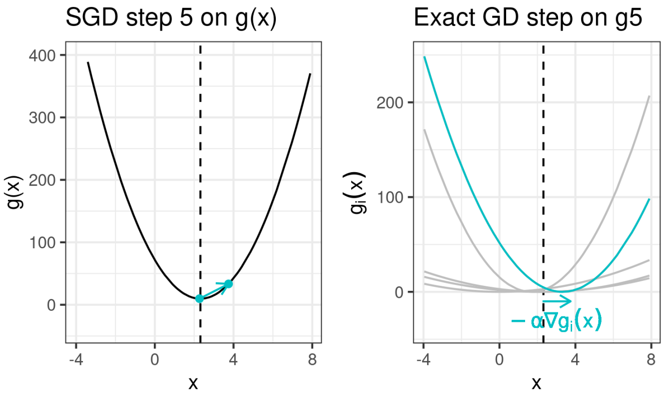
ERRATIC BEHAVIOR OF SGD

Example: $g(\mathbf{x}) = \sum_{i=1}^{5} g_i(\mathbf{x})$, g_i quadratic. Batch size $m = 1$.



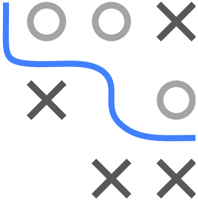
ERRATIC BEHAVIOR OF SGD

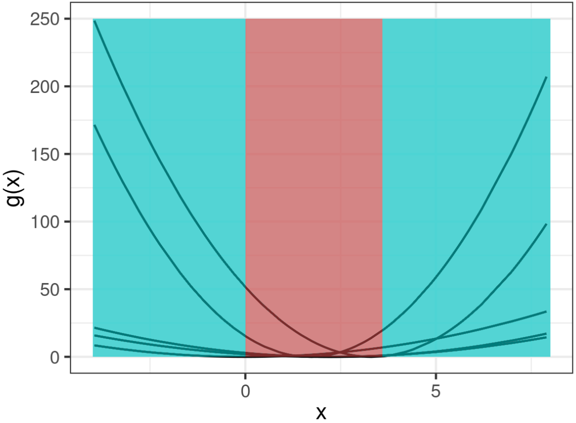
Example: $g(\mathbf{x}) = \sum_{i=1}^{5} g_i(\mathbf{x})$, g_i quadratic. Batch size $m = 1$.



In iteration 5, SGD performs a suboptimal move away from the minimum.

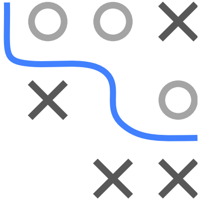
ERRATIC BEHAVIOR OF SGD





Blue area: Each $-\nabla g_i(\mathbf{x})$ points towards minimum.
Red area ("confusion area"): $-\nabla g_i(\mathbf{x})$ might point away from minimum and perform a suboptimal move.

ERRATIC BEHAVIOR OF SGD / 2

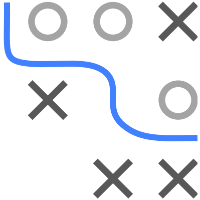


- At location $\mathbf{x}$, “confusion” is captured by variance of gradients

$$\frac{1}{n}\sum_{i=1}^{n}\|\nabla_{\mathbf{x}} g_i(\mathbf{x}) - \nabla_{\mathbf{x}} g(\mathbf{x})\|^2$$

- If term is 0, next step goes in gradient direction (for each i)
- If term is small, next step *likely* goes in gradient direction
- If term is large, next step likely goes in direction different than gradient

CONVERGENCE OF SGD



As a consequence, SGD has worse convergence properties than GD.

But: Can be controlled via **increasing batches** or **reducing step size**.

The larger the batch size m

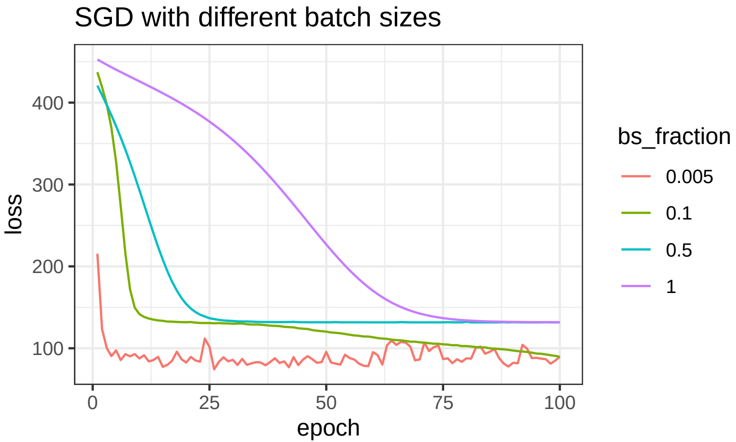
- the better the approximation to $\nabla_{\mathbf{x}} g(\mathbf{x})$
- the lower the variance
- the lower the risk of performing steps in the wrong direction

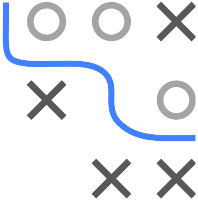
The smaller the step size α

- the smaller a step in a potentially wrong direction
- the lower the effect of high variance

As maximum batch size is usually limited by computational resources (memory), choosing the step size is crucial.

EFFECT OF BATCH SIZE

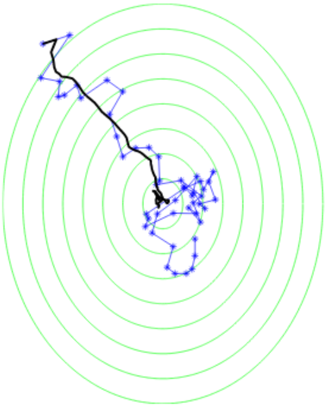




SGD for a NN with batch size $\in \{0.5\%, 10\%, 50\%\}$ of the training data.
The higher the batch size, the lower the variance.

Optimization in Machine Learning

First order methods
SGD Further Details

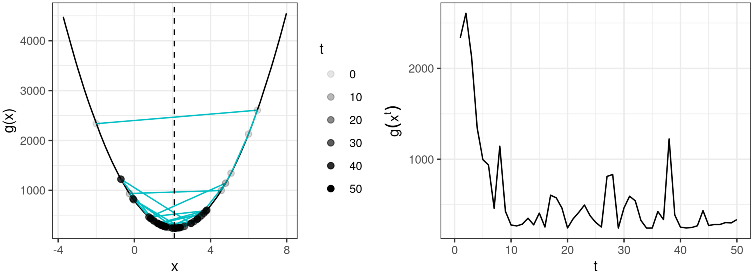


Learning goals

- Decreasing step size for SGD
- Stopping rules
- SGD with momentum

SGD WITH CONSTANT STEP SIZE

Example: SGD with constant step size.



Fast convergence of SGD initially. Erratic behavior later (variance too big).

SGD WITH DECREASING STEP SIZE

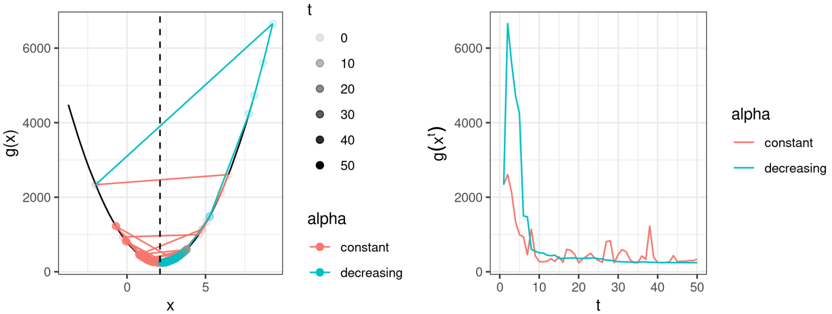
- **Idea:** Decrease step size to reduce magnitude of erratic steps.
- **Trade-off:**
  - if step size $\alpha^{[t]}$ decreases slowly, large erratic steps
  - if step size decreases too fast, performance is impaired
- SGD converges for sufficiently smooth functions if

$$\frac{\sum_{t=1}^{\infty} \left(\alpha^{[t]}\right)^2}{\sum_{t=1}^{\infty} \alpha^{[t]}} = 0$$

("how much noise affects you" by "how far you can get").

SGD WITH DECREASING STEP SIZE / 2

- Popular solution: step size fulfilling $\alpha^{[t]} \in \mathcal{O}(1/t)$.



Example continued. Step size $\alpha^{[t]} = 0.2/t$.

- Often not working well in practice: step size gets small quite fast.
- Alternative: $\alpha^{[t]} \in \mathcal{O}(1/\sqrt{t})$

ADVANCED STEP SIZE CONTROL

Why not Armijo-based step size control?

- Backtracking line search or other approaches based on Armijo rule usually not suitable: Armijo condition

$$g(\mathbf{x} + \alpha \mathbf{d}) \leq g(\mathbf{x}) + \gamma_1 \alpha \nabla g(\mathbf{x})^\top \mathbf{d}$$

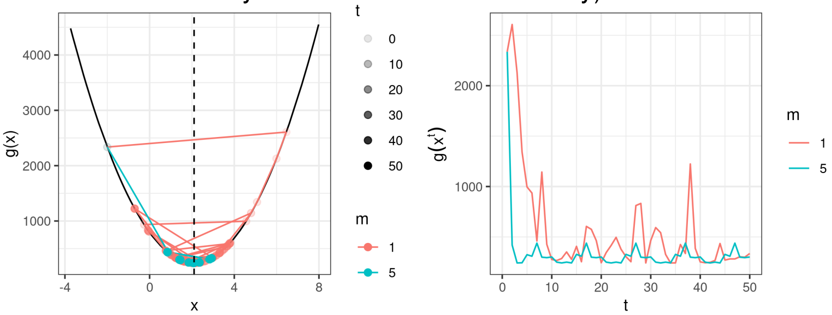
  requires evaluating full gradient.
- But SGD is used to *avoid* expensive gradient computations.
- Research aims at finding inexact line search methods that provide better convergence behaviour, e.g., Vaswani et al., *Painless Stochastic Gradient: Interpolation, Line-Search, and Convergence Rates*. NeurIPS, 2019.

MINI-BATCHES

- Reduce noise by increasing batch size m for better approximation

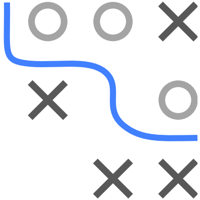
$$\hat{\mathbf{d}} = \frac{1}{m}\sum_{i\in J}\nabla_{\mathbf{x}} g_i(\mathbf{x}) \approx \frac{1}{n}\sum_{i=1}^{n}\nabla_{\mathbf{x}} g_i(\mathbf{x}) = \mathbf{d}$$

- Usually, the batch size is limited by computational resources (e.g., how much data you can load into the memory)



Example continued. Batch size $m = 1$ vs. $m = 5$.

STOPPING RULES FOR SGD



- **For GD**: We usually stop when gradient is close to 0 (i.e., we are close to a stationary point)
- **For SGD**: individual gradients do not necessarily go to zero, and we cannot access full gradient.
- Practicable solution for ML:
  - Measure the validation set error after T iterations
  - Stop if validation set error is not improving

SGD AND ML

General remarks:

- SGD is a variant of GD
- SGD particularly suitable for large-scale ML when evaluating gradient is too expensive / restricted by computational resources
- SGD and variants are the most commonly used methods in modern ML, for example:
  - Linear models

    Note that even for the linear model and quadratic loss, where a closed form solution is available, SGD might be used if the size n of the dataset is too large and the design matrix does not fit into memory.
  - Neural networks
  - Support vector machines
  - ...

SGD WITH MOMENTUM

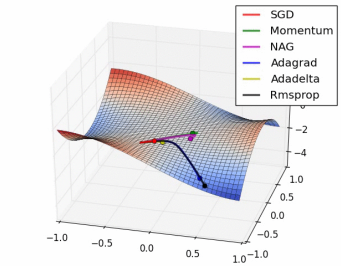
SGD is usually used with momentum due to reasons mentioned in previous chapters.

Algorithm Stochastic gradient descent with momentum

```
require step size α and momentum φ
require initial parameter x and initial velocity ν
while stopping criterion not met do
    Sample mini-batch of m examples
    Compute gradient estimate ∇ĝ(x) using mini-batch
    Compute velocity update: ν ← φν − α∇ĝ(x)
    Apply update: x ← x + ν
end while
```

Optimization in Machine Learning

First order methods
Adam and friends

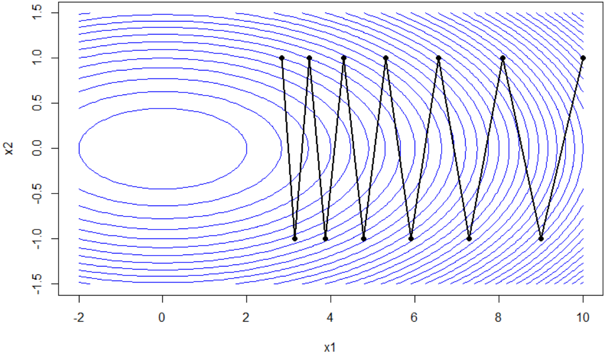


Learning goals

- Adaptive step sizes
- AdaGrad
- RMSProp
- Adam

ADAPTIVE STEP SIZES

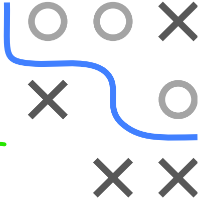
- Step size is probably the most important control parameter
- Has strong influence on performance
- Natural to use different step size for each input individually and automatically adapt them



ADAGRAD

- AdaGrad adapts step sizes by scaling them inversely proportional to square root of the sum of the past squared derivatives
  - Inputs with large derivatives get smaller step sizes
  - Inputs with small derivatives get larger step sizes
- Accumulation of squared gradients can result in premature small step sizes (Goodfellow et al., 2016)

$\frac{1}{\sqrt{\ }}$



sc. factor $= \frac{1}{\sqrt{\sum_{i=1}^{t-1} (\nabla f^{[i]})^2}}$

What if we do last n iter, not all?

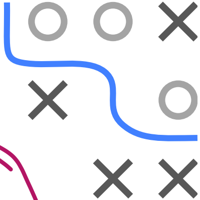
IDK, maybe there is reason that does it?

ADAGRAD / 2

Algorithm AdaGrad

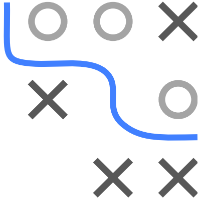
require Global step size α
require Initial parameter $\boldsymbol{\theta}$
require Small constant β, perhaps 10^{-7}, for numerical stability
Initialize gradient accumulation variable $\mathbf{r} = \mathbf{0}$
while stopping criterion not met **do**
    Sample minibatch of m examples from the training set $\{\tilde{\mathbf{x}}^{(1)}, \ldots, \tilde{\mathbf{x}}^{(m)}\}$
    Compute gradient estimate: $\hat{\mathbf{g}} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L\left(y^{(i)}, f\left(\tilde{\mathbf{x}}^{(i)} \mid \boldsymbol{\theta}\right)\right)$
    Accumulate squared gradient $\mathbf{r} \leftarrow \mathbf{r} + \hat{\mathbf{g}} \odot \hat{\mathbf{g}}$
    Compute update: $\nabla\boldsymbol{\theta} = -\frac{\alpha}{\beta + \sqrt{\mathbf{r}}} \odot \hat{\mathbf{g}}$ (operations element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \nabla\boldsymbol{\theta}$
end while

$\odot$: element-wise product (Hadamard)



RMSPROP

- Modification of AdaGrad
- Resolves AdaGrad's radically diminishing step sizes.
- Gradient accumulation is replaced by exponentially weighted moving average
- Theoretically, leads to performance gains in non-convex scenarios
- Empirically, RMSProp is a very effective optimization algorithm. Particularly, it is employed routinely by DL practitioners.



$\beta \cdot v_t + (1-\beta)\, v^T$

β^n

RMSPROP / 2

Algorithm RMSProp

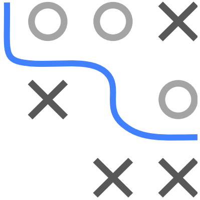
require Global step size α and decay rate $\rho \in [0, 1)$
require Initial parameter $\boldsymbol{\theta}$
require Small constant β, perhaps 10^{-6}, for numerical stability
Initialize gradient accumulation variable $\mathbf{r} = \mathbf{0}$
while stopping criterion not met **do**
    Sample minibatch of m examples from the training set $\{\tilde{\mathbf{x}}^{(1)}, \dots, \tilde{\mathbf{x}}^{(m)}\}$
    Compute gradient estimate: $\hat{\mathbf{g}} \leftarrow \frac{1}{m} \nabla_\theta \sum_i L\left(y^{(i)}, f\left(\tilde{\mathbf{x}}^{(i)} \mid \boldsymbol{\theta}\right)\right)$
    Accumulate squared gradient $\mathbf{r} \leftarrow \rho \mathbf{r} + (1 - \rho)\hat{\mathbf{g}} \odot \hat{\mathbf{g}}$
    Compute update: $\nabla \boldsymbol{\theta} = -\frac{\alpha}{\beta + \sqrt{\mathbf{r}}} \odot \hat{\mathbf{g}}$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \nabla \boldsymbol{\theta}$
end while

we accumulate the squared grad differently, by decaying the previous ones.

ρ trades of amount of emphasis on new and old gradients

ADAM = Mo + RMSP

ADAM



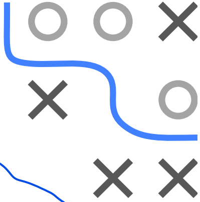
- Adaptive Moment Estimation also has adaptive step sizes
- Uses the 1st and 2nd moments of gradients
  - Keeps an exponentially decaying average of past gradients (1st moment)
  - Like RMSProp, stores an exp-decaying avg of past squared gradients (2nd moment)
  - Can be seen as combo of RMSProp + momentum.

$\beta_b \theta + v\theta$

ADAM / 2

Algorithm Adam

require Global step size α (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 β (suggested default 10^{-8})
require Initial parameters $\boldsymbol{\theta}$
Initialize time step $t = 0$
Initialize 1st and 2nd moment variables $\mathbf{s}^{[0]} = 0, \mathbf{r}^{[0]} = 0$
while stopping criterion not met **do**
  $t \leftarrow t + 1$
  Sample a minibatch of m examples from the training set $\{\tilde{x}^{(1)}, \dots, \tilde{x}^{(m)}\}$
  Compute gradient estimate: $\hat{\mathbf{g}}^{[t]} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L\left(y^{(i)}, f\left(\tilde{\mathbf{x}}^{(i)} \mid \boldsymbol{\theta}\right)\right)$
  Update biased first moment estimate: $\mathbf{s}^{[t]} \leftarrow \rho_1 \mathbf{s}^{[t-1]} + (1 - \rho_1)\hat{\mathbf{g}}^{[t]}$ → Momentum
  Update biased second moment estimate: $\mathbf{r}^{[t]} \leftarrow \rho_2 \mathbf{r}^{[t-1]} + (1 - \rho_2)\hat{\mathbf{g}}^{[t]} \odot \hat{\mathbf{g}}^{[t]}$ → RMS Prop
  Correct bias in first moment: $\hat{\mathbf{s}} \leftarrow \frac{\mathbf{s}^{[t]}}{1-\rho_1^t}$
  Correct bias in second moment: $\hat{\mathbf{r}} \leftarrow \frac{\mathbf{r}^{[t]}}{1-\rho_2^t}$ → this Part
  Compute update: $\nabla\boldsymbol{\theta} = -\alpha \frac{\hat{\mathbf{s}}}{\sqrt{\hat{\mathbf{r}}}+\beta}$
  Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \nabla\boldsymbol{\theta}$
end while



more expensive but not really, we just have extra 2 vectors. and we dont have second-order stuff.

$t:10$ 0.9^{10}

10 $\quad \frac{5}{1-\rho^t}$

ADAM / 3

- Initializes moment variables **s** and **r** with zero $\Rightarrow$ Bias towards zero

$$\mathbb{E}[\mathbf{s}^{[t]}] \neq \mathbb{E}[\hat{\mathbf{g}}^{[t]}] \quad \text{and} \quad \mathbb{E}[\mathbf{r}^{[t]}] \neq \mathbb{E}[\hat{\mathbf{g}}^{[t]} \odot \hat{\mathbf{g}}^{[t]}]$$

($\mathbb{E}$ calculated over minibatches) Own I, II moment estimates aren't unbiased estimates of gradient

- Indeed: Unrolling $\mathbf{s}^{[t]}$ yields

cancels out cancel

$$\mathbf{s}^{[0]} = 0$$

$$\mathbf{s}^{[1]} = \rho_1 \mathbf{s}^{[0]} + (1-\rho_1)\hat{\mathbf{g}}^{[1]} = (1-\rho_1)\hat{\mathbf{g}}^{[1]}$$

$$\mathbf{s}^{[2]} = \rho_1 \mathbf{s}^{[1]} + (1-\rho_1)\hat{\mathbf{g}}^{[2]} = \rho_1(1-\rho_1)\hat{\mathbf{g}}^{[1]} + (1-\rho_1)\hat{\mathbf{g}}^{[2]}$$

$$\mathbf{s}^{[3]} = \rho_1 \mathbf{s}^{[2]} + (1-\rho_1)\hat{\mathbf{g}}^{[3]} = \rho_1^2(1-\rho_1)\hat{\mathbf{g}}^{[1]} + \rho_1(1-\rho_1)\hat{\mathbf{g}}^{[2]} + (1-\rho_1)\hat{\mathbf{g}}^{[3]}$$

- Therefore: $\mathbf{s}^{[t]} = (1-\rho_1)\sum_{i=1}^{t} \rho_1^{t-i} \hat{\mathbf{g}}^{[i]}$. (can prove by induction)
- **Note:** Contributions of past $\hat{\mathbf{g}}^{[i]}$ decreases rapidly

ADAM / 4

- We continue with

$$\begin{aligned}\mathbb{E}[\mathbf{s}^{[t]}] &= \mathbb{E}[(1-\rho_1)\sum_{i=1}^{t}\rho_1^{t-i}\hat{\mathbf{g}}^{[i]}] \\ &= \mathbb{E}[\hat{\mathbf{g}}^{[t]}](1-\rho_1)\sum_{i=1}^{t}\rho_1^{t-i}+\zeta \\ &= \mathbb{E}[\hat{\mathbf{g}}^{[t]}](1-\rho_1^{t})+\zeta,\end{aligned}$$

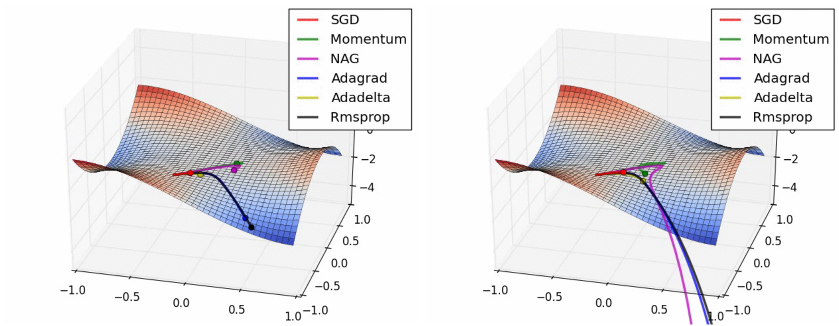
assume all grads are the same

where we approximated $\hat{\mathbf{g}}^{[i]}$ by $\hat{\mathbf{g}}^{[t]}$. The resulting error is put in ζ and be kept small due to the exponential weights of past gradients.

- Therefore: $\mathbf{s}^{[t]}$ is a biased estimator of $\hat{\mathbf{g}}^{[t]}$
- But bias vanishes for $t \to \infty$ ($\rho_1^t \to 0$)
- Ignoring ζ, we correct for the bias by $\hat{\mathbf{s}}^{[t]} = \frac{\mathbf{s}^{[t]}}{(1-\rho_1^t)}$
- Analogously: $\hat{\mathbf{r}}^{[t]} = \frac{\mathbf{r}^{[t]}}{(1-\rho_2^t)}$

why we want unbiasedness
Seems contra productive...

COMPARISON OF OPTIMIZERS: ANIMATION



Credits: Dettmers (2015) and Radford

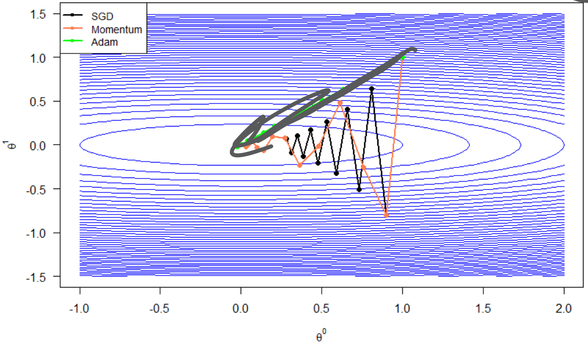
Comparison of SGD optimizers near saddle point.

Left: After start. **Right:** Later.

All methods accelerate compared to vanilla SGD.

Best is RMSProp, then AdaGrad. (Adam is missing here.)

COMPARISON ON QUADRATIC FORM



SGD vs. SGD with Momentum vs. Adam on a quadratic form.

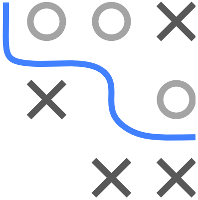
Optimization in Machine Learning

First order methods
Comparison of first order methods

Learning goals

- Gradient Descent
- Stochastic Gradient Descent
- Momentum
- Step size decay

COMPARISON OF FIRST ORDER METHODS

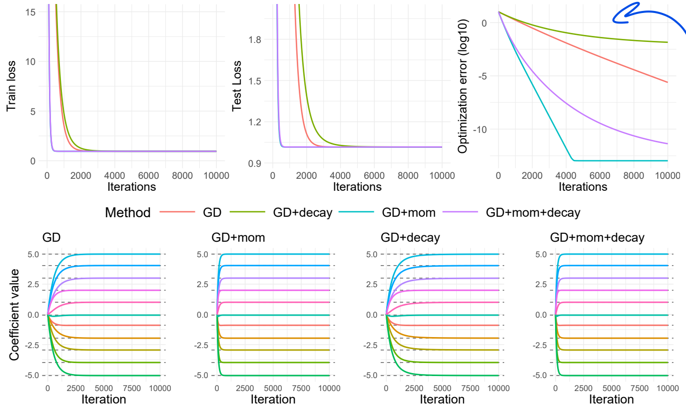


Comparison of (S)GD, (S)GD + momentum, and (S)GD + momentum + step size control on simulated data. We do not use analytical solution on purpose although one exists:

- Linear regression (squared loss) simulation $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\theta}^* + \varepsilon$ with $n = 500$ samples and $p = 11$ features, where $\boldsymbol{\theta}^* = (-5, -4, \dots 0, \dots, 4, 5)^\top$, $\varepsilon \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, and $\boldsymbol{X} \sim \mathcal{N}(\mathbf{0}, \Sigma)$ for $\Sigma = \boldsymbol{I}$ (i.i.d. features) or $\Sigma_{i,j} = 0.99^{|i-j|}$ (corr. features) ([illegible] toeplets structure)
- Indep. features result in a condition number of ≈ 2.9, whereas the corr. feature set-up produces a bad condition number of ≈ 600 (Is there a formula to estimate κ give a correlation) why not 1?
- We set the momentum parameter to 0.8 and the decay step size using schedule $\alpha^{[t]} = \alpha^{[0]} \cdot \texttt{decay}^{t/t_{\max}}$ for $\texttt{decay} = 0.1$
- For GD and SGD we use different step sizes to show that benefit of momentum/decay depends heavily on step size
- ERM has unique global minimizer given by $\hat{\boldsymbol{\theta}} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1}\boldsymbol{X}^\top \boldsymbol{y}$
- We also track the optimization error $\|\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}\|_2$

LIN-REG (GD + MED STEP SIZE, DEFAULT CASE)

GD with medium $\alpha = 2 \cdot 10^{-3}$ and indep. features:



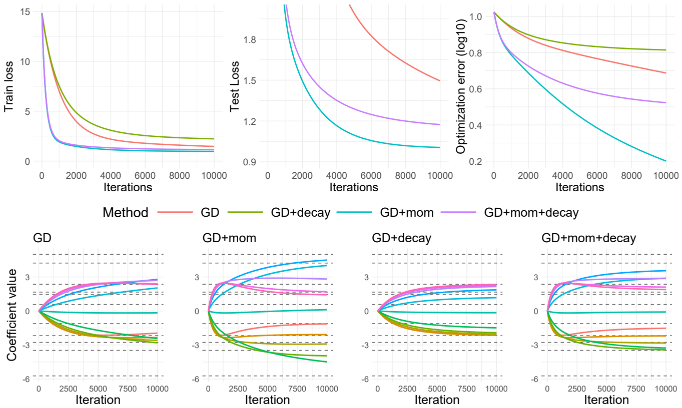
when we look at $\|\theta - \theta^*\|_2$ we take θ^* from DGP but it's not the same for a sample from DGP

Irreducible error due to additive noise is $\sigma = 1$. Dotted lines indicate global minimizers.

All variants converge to global min. **Momentum** accelerates optimization and **decay** slows down optimization under that step size.

LIN-REG (GD + CORR. FEATURES)

GD with medium $\alpha = 2 \cdot 10^{-3}$ and bad conditioning (corr. features):



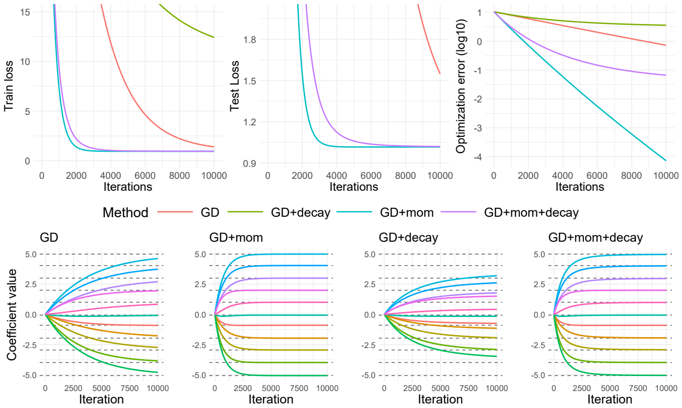
$\hat{\theta}$ quite far from θ^*

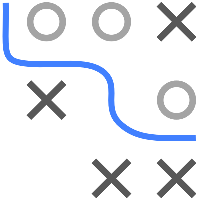
Irreducible error due to additive noise is $\sigma = 1$. Dotted lines indicate global minimizers.

Bad **conditioning slows down optim** severely! Only **momentum** w/o decay comes close to global min. **Momentum** causes “overshooting” of some coefs. Corr. features cause corr. global minimizers.

LIN-REG (GD + SMALL STEP SIZE)

GD with (too small) $\alpha = 3 \cdot 10^{-4}$ and indep. features:



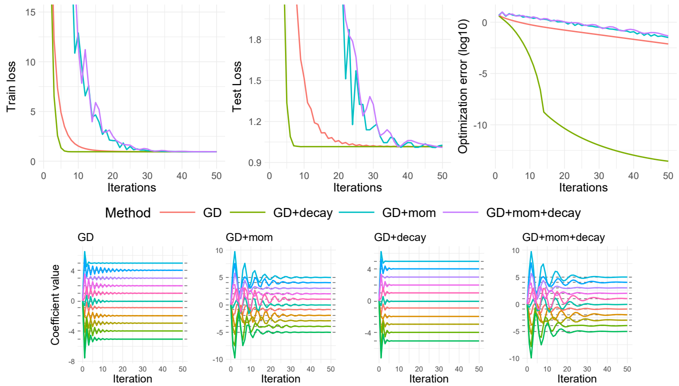


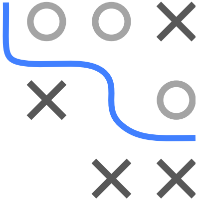
Irreducible error due to additive noise is $\sigma = 1$. Dotted lines indicate global minimizers.

Only two **momentum** variants come close to global min in $t_{\text{max}} = 10000$. **Decay** worsens performance as α was already too low.

LIN-REG (GD + LARGE STEP SIZE)

GD with large $\alpha = 1.5$ and indep. features:



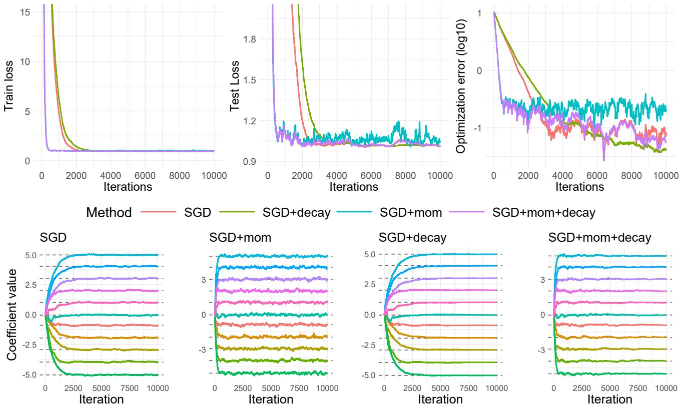


Irreducible error due to additive noise is $\sigma = 1$. Dotted lines indicate global minimizers.

Super fast convergence in < 20 steps. **Decay** here accelerates optim while **momentum** becomes slow and unstable. Coefficients oscillate at beginning which is reduced by decay.

LIN-REG (SGD + MED STEP SIZE)

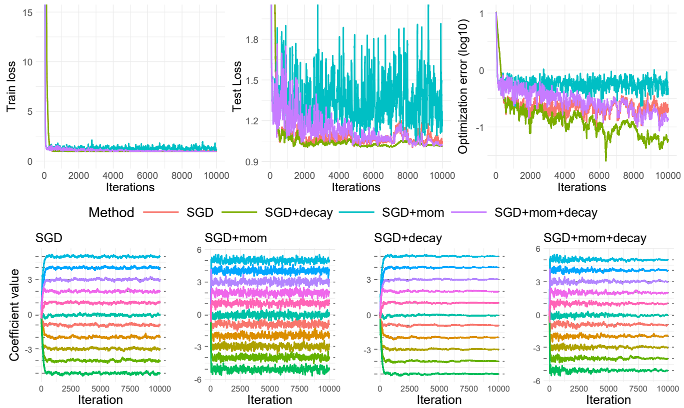
SGD with medium $\alpha = 2 \cdot 10^{-3}$ and indep. features:



Momentum accelerates optim initially but is eventually outperformed by other variants. **Momentum+decay** is both fast initially and has small final error. **Decay** performs best overall but is slowest initially.

LIN-REG (SGD + LARGE STEP SIZE)

SGD with large $\alpha = 1 \cdot 10^{-2}$ and indep. features:



Irreducible error due to additive noise is $\sigma = 1$

High variance in SGD dynamics. **Momentum** becomes unstable while **decay** (necessary to eliminate noise) performs best and is fastest.

DIGRESSION: SOLVING OLS WITH QR DECOMP.

Solving linear least squares via (S)GD is rarely done in practice. But inversion of $\boldsymbol{X}^\top\boldsymbol{X}$ is numerically unstable and to be avoided. A standard numerical approach applies **QR Decomposition**:

- Factorize $\boldsymbol{X} \in \mathbb{R}^{n\times p}$ as $\boldsymbol{X} = \boldsymbol{QR}$, where $\boldsymbol{Q} \in \mathbb{R}^{n\times p}$ (thin form) so that $\boldsymbol{Q}^\top\boldsymbol{Q} = \boldsymbol{I}$ and $\boldsymbol{R} \in \mathbb{R}^{p\times p}$ is upper triangular.
- Purpose: replace solving $\hat{\boldsymbol{\theta}} = (\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{y}$ with a more numerically stable method by avoiding direct inversion.
- The QR decomposition can be computed using Gram-Schmidt orthogonalization or Householder transformations

rewrite

$(X^T X)\theta = X^T y$

Lin. eq. system,

Because computing inverse of $X^T X$ is awful

Steps:

1. Decompose $\boldsymbol{X}$ into $\boldsymbol{Q}$ and $\boldsymbol{R}$.
2. Compute $\boldsymbol{Q}^\top\boldsymbol{y}$.
3. Solve triangular system $\boldsymbol{R}\hat{\boldsymbol{\theta}} = \boldsymbol{Q}^\top\boldsymbol{y}$ via **back substitution**.

→ maybe we want to avoid $X^T X$ mat. multip.

$X = QR$ Q-ort. to.

$X^T X = R^T \cancel{Q^T} Q R$

ToDo?

DIGRESSION: SOLVING OLS WITH QR DECOMP.

Why this system? Remember normal equation for least squares problem is $\mathbf{X}^\top \mathbf{X}\hat{\theta} = \mathbf{X}^\top \mathbf{y}$. Now replace $\mathbf{X} = \mathbf{Q}\mathbf{R}$:

$$\mathbf{X}^\top \mathbf{X}\hat{\theta} = \mathbf{X}^\top \mathbf{y} \iff \mathbf{R}^\top (\mathbf{Q}^\top \mathbf{Q})\mathbf{R}^\top \hat{\theta} = \mathbf{R}^\top \mathbf{Q}^\top \mathbf{y}$$

$$\mathbf{R}^\top \mathbf{R}\hat{\theta} = \mathbf{R}^\top \mathbf{Q}^\top \mathbf{y} \iff \mathbf{R}\hat{\theta} = \mathbf{Q}^\top \mathbf{y}$$

Be careful $R^\top$ isn't square

$\mathbf{R}\hat{\theta} = \mathbf{Q}^\top \mathbf{y}$ is a triangular system easily solvable by back substitution:

$$\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & \dots & r_{1p} \\ 0 & r_{22} & \dots & r_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & r_{pp} \end{bmatrix}, \quad \hat{\theta} = \begin{bmatrix} \hat{\theta}_1 \\ \hat{\theta}_2 \\ \vdots \\ \hat{\theta}_p \end{bmatrix}, \quad \mathbf{Q}^\top \mathbf{y} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_p \end{bmatrix}.$$

DIGRESSION: SOLVING OLS WITH QR DECOMP.

Steps for Back Substitution:

1. Start with the last equation (1 unknown):

$$\hat{\theta}_p = \frac{b_p}{r_{pp}}.$$

2. Move upwards to the $(p-1)$-th equation:

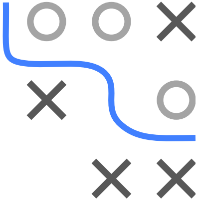
$$\hat{\theta}_{p-1} = \frac{b_{p-1} - r_{p-1,p}\hat{\theta}_p}{r_{p-1,p-1}}.$$

3. Continue this process up to the first row:

$$\hat{\theta}_i = \frac{b_i - \sum_{j=i+1}^{p} r_{ij}\hat{\theta}_j}{r_{ii}} \quad \text{for each } i = p-1, \ldots, 1.$$

Back substitution leverages triangular structure of $\boldsymbol{R}$, moving upward from the last row and inserting already known values of $\hat{\boldsymbol{\theta}}$ as we go.

(S)GD FOR LOGISTIC REGRESSION

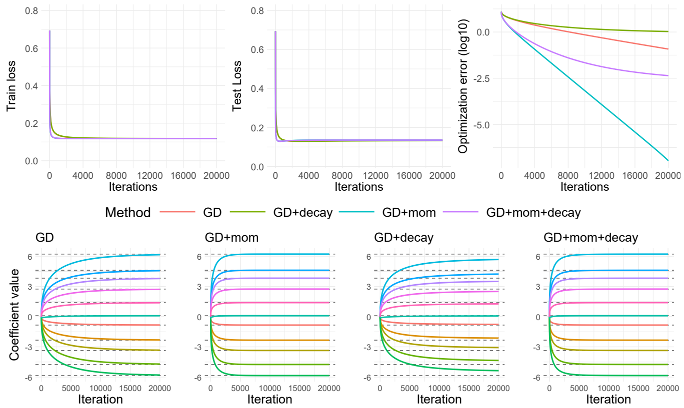


Comparison of (S)GD, (S)GD + momentum, and (S)GD + momentum + step size control on simulated data:

- Logistic regression (log loss) with same simulation setup as for linear regression
- To simulate response, we set $y^{(i)} \sim \mathcal{B}(\pi^{(i)}), \pi^{(i)} = \frac{1}{1+e^{-\left(\mathbf{x}^{(i)}\right)^\top \boldsymbol{\theta}^*}}$
- We set the momentum parameter to 0.8 and `decay` $= 0.1$
- We again use different step sizes to illustrate benefit of momentum and decay depends on step size
- ERM has unique global minimizer but no closed-form solution. We can approximate $\hat{\boldsymbol{\theta}}$ using the `glm` solution (second-order optim)
- We also track the optimization error $\|\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}\|_2$ ~ get from GLM (which uses Newton-Raphson)

LOG-REG (GD + MED STEP SIZE)

GD with medium $\alpha = 0.25$ and indep. features:

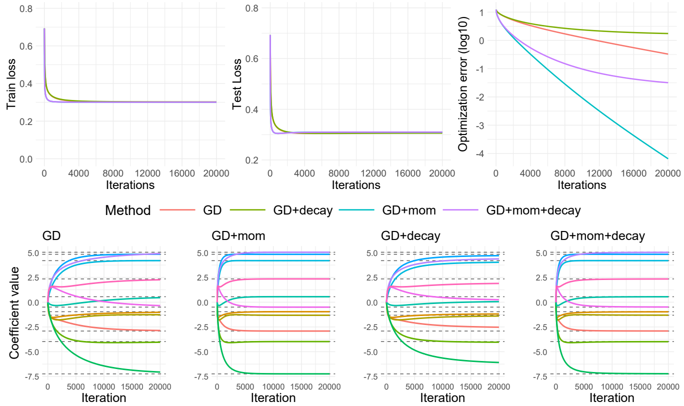


Dotted lines indicate global minimizers.

All variants converge to global min. **Momentum** accelerates and **decay** slows down optimization. **GD+mom** performs best.

LOG-REG (GD + CORR. FEATURES)

GD with medium $\alpha = 0.25$ and bad conditioning (corr. features):

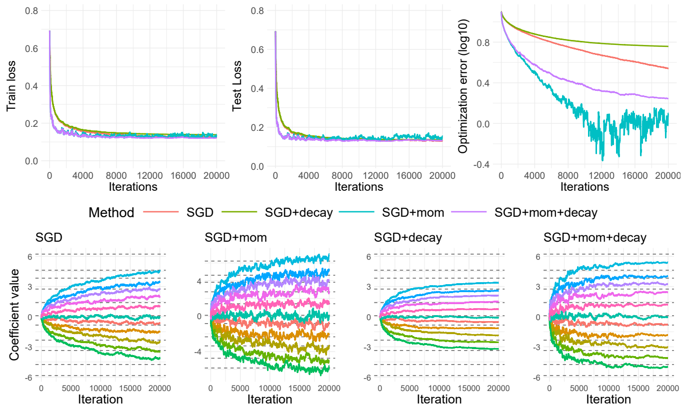


Dotted lines indicate global minimizers.

Bad conditioning slows down optim slightly. Only **momentum w/o decay** converges exactly to global min in t_{max}. Momentum causes "**overshooting**" of some coefs.

LOG-REG (SGD + MED STEP SIZE)

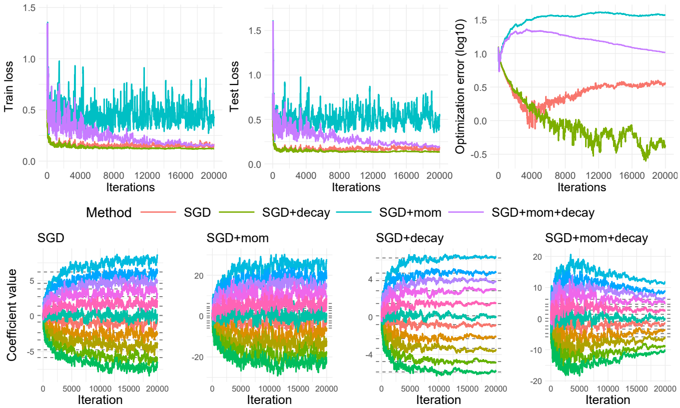
SGD with medium $\alpha = 0.03$ and indep. features:

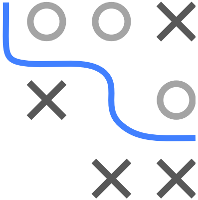


Under SGD dynamics become more noisy. **Momentum** accelerates while **decay** slows down optimization. Only the coefs of **momentum w/o decay** come close to global minimizers.

LOG-REG (SGD + LARGE STEP SIZE)

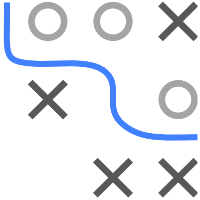
SGD with large $\alpha = 0.3$ and indep. features:





High variance in optim dynamics. Plain SGD and **Momentum** become unstable and oscillate at suboptimal loss values/diverge while **decay** (necessary to eliminate oscillations) performs best and is fastest. **Momentum+decay** initially diverges but recovers once step size reduces (would converge with many more steps).

CLASSIFICATION ON MNIST (SGD)



For a more realistic application, we compare optimizers on training an NN on subset of MNIST image classification task.

- DNN has two hidden layers with 128 and 64 ReLU-activated units and Kaiming normal init. The loss is cross-entropy

  Sample around zero from Normal

- Instead of SGD we use mini-batch SGD with 100 images per batch

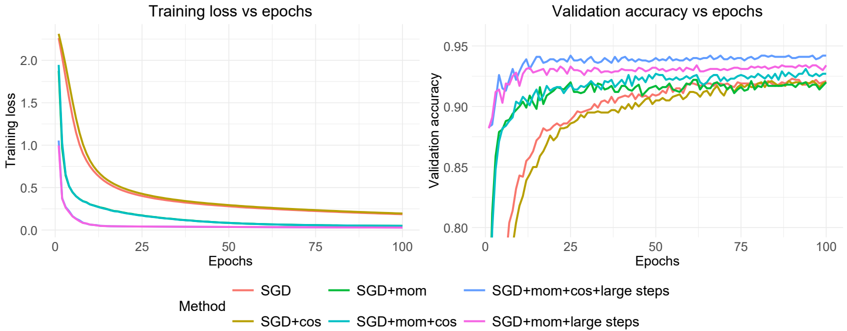
- For the step size schedule, we use cosine decay $\alpha^{[t]} = \alpha^{[0]} \left[(1 - r_{\min}) \cdot \frac{1}{2} \left(1 + \cos\left(\pi \cdot \frac{t}{t_{\max}} \right) \right) + r_{\min} \right]$ with final step size fraction $r_{\min} = 0.01$

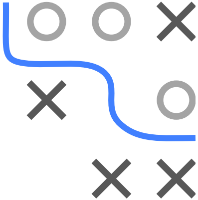
- In case of momentum we set the parameter to 0.8

- Regular initial step size is 0.01 and to 0.1 for large step size

CLASSIFICATION ON MNIST (SGD)

Mini-batch SGD with $\alpha \in \{0.01, 0.1\}$ for 100 epochs (5000 iterations):





Observations (**NB**: green/cyan + blue/purple train losses overlap but not val. acc.):

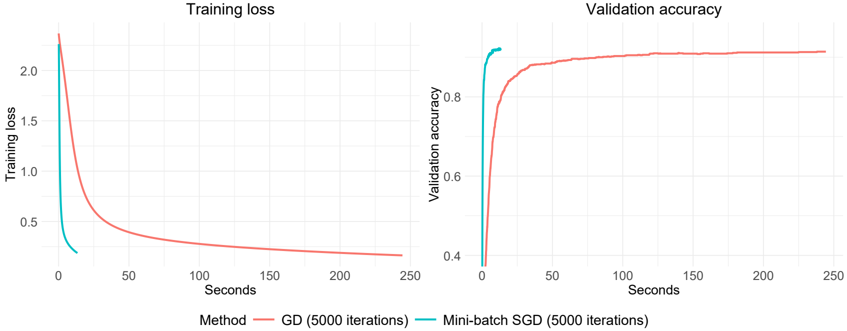
1. **Momentum** drastically speeds up optimization in all settings.
2. **Plain SGD**: step size decay slows optimization slightly.
3. SGD, SGD+cos and SGD+mom achieve **same** val. acc.

$\Rightarrow$ no generalization benefit for medium α

4. **Cosine decay+momentum** improves generalization for medium α slightly.
5. **Large step size** with momentum and decay performs best

CLASSIFICATION ON MNIST (GD VS. SGD)

Why is it not a good idea to use GD in most DL applications? SGD is much faster. Compare runtime of mini-batch SGD (batch size=100) with GD (constant $\alpha = 0.01$ without momentum for $t_{\text{max}} = 5000$ iterations):



Observations:

1. Mini-batch SGD is over an order of magnitude faster.
2. SGD generalizes better than GD despite being much faster.