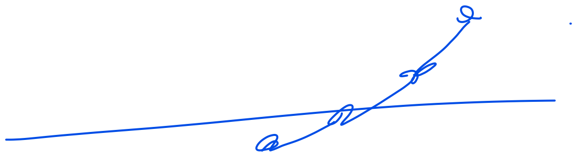
Gradient Descent - Step Size



Optimization in Machine Learning

First order methods
Gradient descent

Learning goals

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

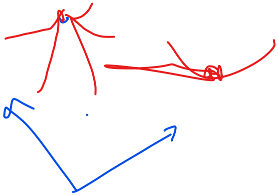
ITERATIVE DESCENT

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

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

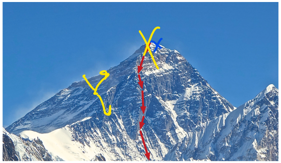
Goal: Reach the valley

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



Central idea: Repeat

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



"Walking down the hill, towards the valley."

ITERATIVE DESCENT

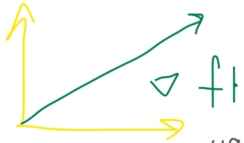
Let $f : \mathbb{R}^d \to \mathbb{R}$ continuously differentiable.

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

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

$-\nabla f(x)$

$\cos\alpha < 0$

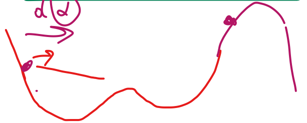


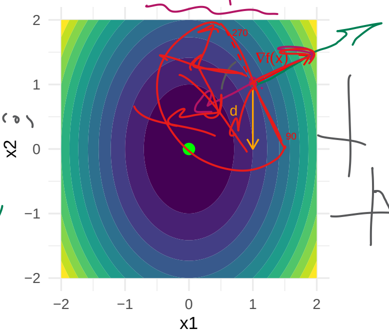
∇f

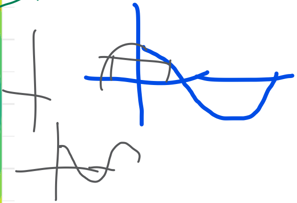
$\|f\| \cdot \cos$

$$\underbrace{\|\nabla f\| \cdot \|d\| \cos\alpha}$$

$d\alpha$







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

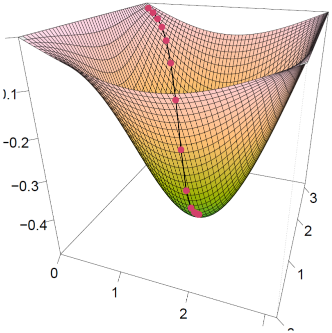
Properties of gradient:

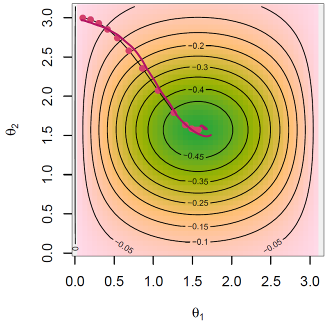
- $\underline{\nabla f(\mathbf{x})}$: direction of greatest increase
- $-\nabla f(\mathbf{x})$: direction of greatest decrease

$\nearrow$

$\swarrow$

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

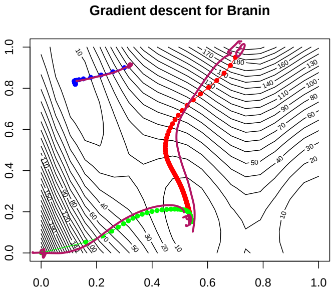


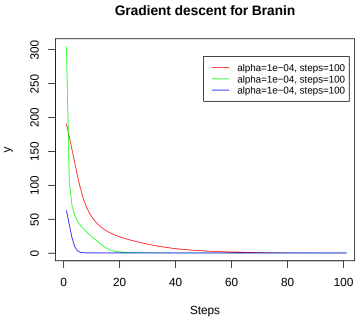


GD for $f(x_1, x_2) = -\sin(x_1) \cdot \frac{1}{2\pi} \exp\left((x_2 - \pi/2)^2\right)$ with sensibly chosen step size $\alpha^{[t]}$.

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

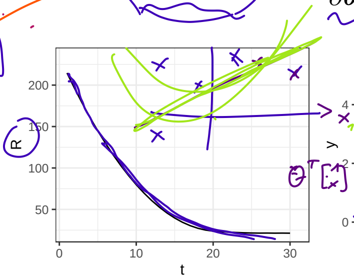
$\theta^T x - y$

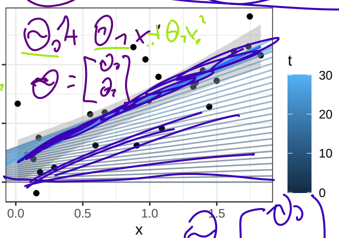
$$\mathcal{R}_{\text{emp}}(\boldsymbol{\theta}) = \sum_{i=1}^{n} \left(\boldsymbol{\theta}^\top \mathbf{x}^{(i)} - y^{(i)}\right)^2$$

x_1	x_1	x_2	y

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





$\theta_0 + \theta_1 x$

$\theta \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$

ERM FOR NN WITH GD

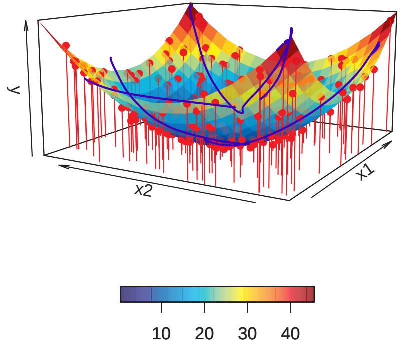
Let $\mathcal{D} = \left(\left(\mathbf{x}^{(1)}, y^{(1)}\right), \dots, \left(\mathbf{x}^{(n)}, y^{(n)}\right)\right)$, with $y = \boxed{x_1^2 + x_2^2}$ and minimize

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

DGD

$\theta \; \theta_0$

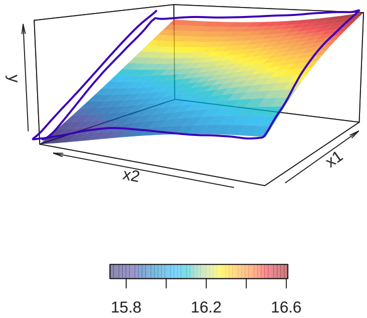
where $f(\mathbf{x} \mid \boldsymbol{\theta})$ is a neural network with 2 hidden layers (2 units each).

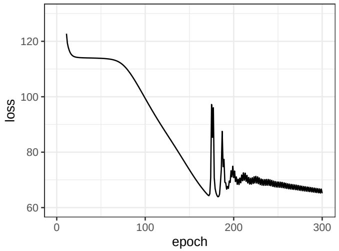


$\theta_0 \quad \theta_n$

ERM FOR NN WITH GD

After 10 iters of GD:

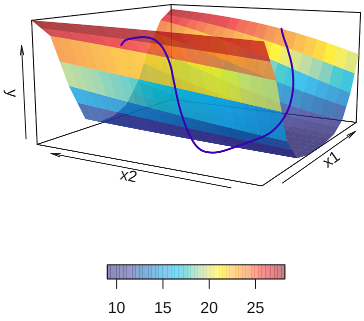


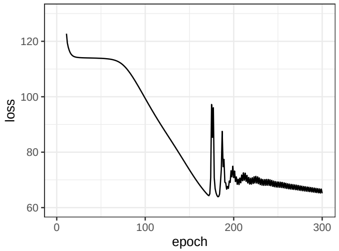


ERM FOR NN WITH GD

After 100 iters of GD:

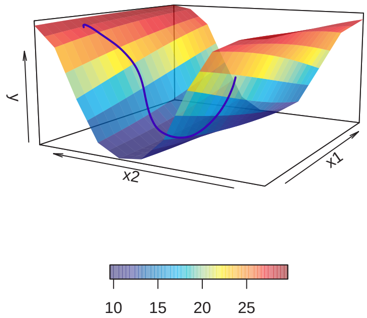
$\theta_0 \quad \theta_1$

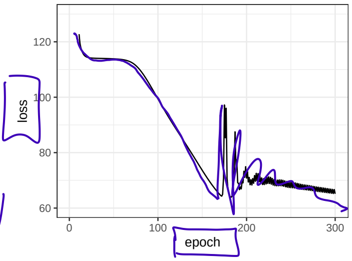




ERM FOR NN WITH GD

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





10.000

$$f : \mathbb{R}^{10000} \to \mathbb{R}^1$$

$x^{[0]}$ x^0

0.3

$f(x^0 + 0.3 \cdot \boxed{1})$

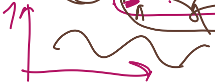
10.000

$g(\alpha) = f(x^{[0]} + \boxed{d}\,\underline{\alpha})$

$\boxed{1-\alpha}$

$f(x, y, z) = x + y + z$

$y, z = 0$

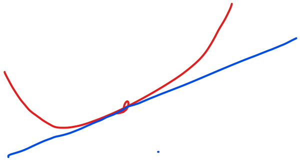


optimize α=0 α=10

Line Search Methods

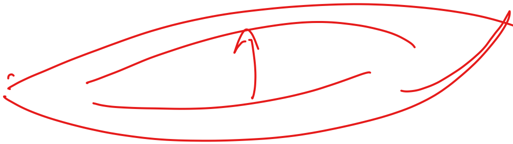
g.s.s

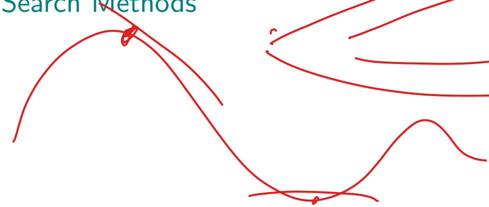
Videos



$$\mathcal{H}f \begin{bmatrix} \quad \end{bmatrix}$$

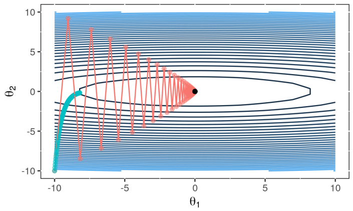
- Inexact Line Search
- Armijo Condition
- Wolfe 2nd Condition
- Line Search Methods





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

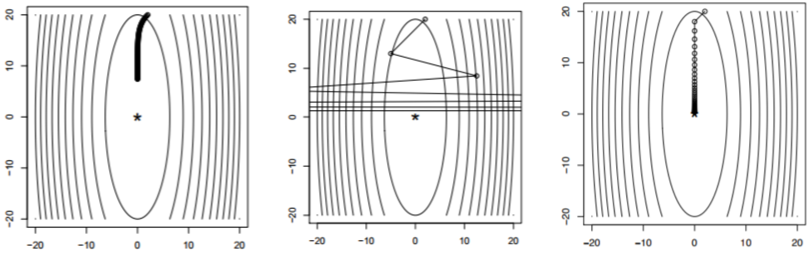
CONTROLLING STEP SIZE: FIXED & ADAPTIVE

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 $f(\boldsymbol{x}) = 10x_1^2 + x_2^2/2$

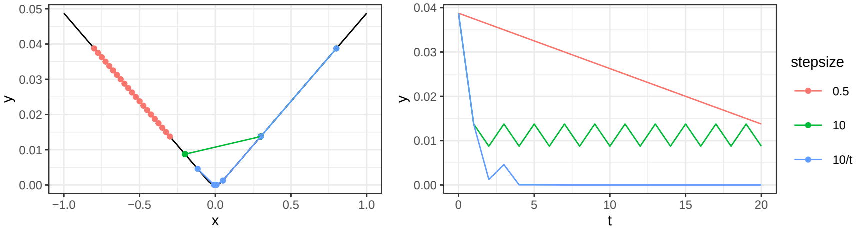
STEP SIZE CONTROL: DIMINISHING STEP SIZE

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

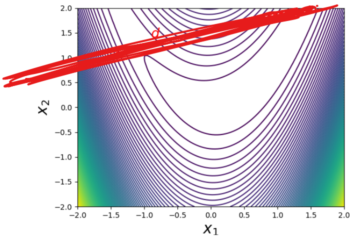
Use **optimal** step size in each iteration:

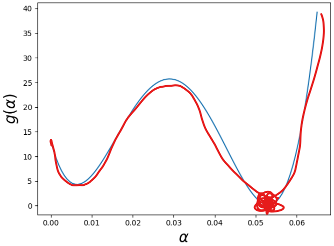
$$\alpha^{[t]} = \underset{\alpha \in \mathbb{R}_{\geq 0}}{\arg\min}\, g(\alpha) = \underset{\alpha \in \mathbb{R}_{\geq 0}}{\arg\min}\, f(\mathbf{x}^{[t]} + \alpha \mathbf{d}^{[t]})$$

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

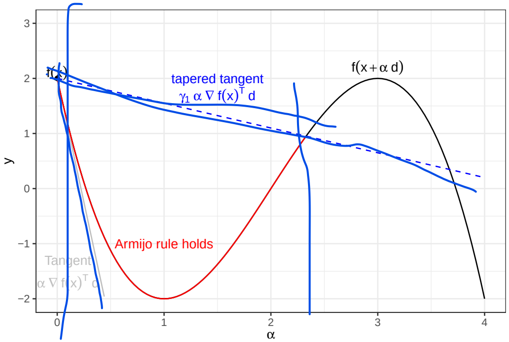
Problem: Expensive, prone to poorly conditioned problems

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.





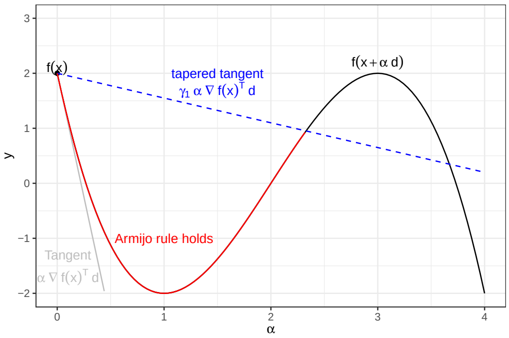
ARMIJO RULE



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

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

ARMIJO RULE

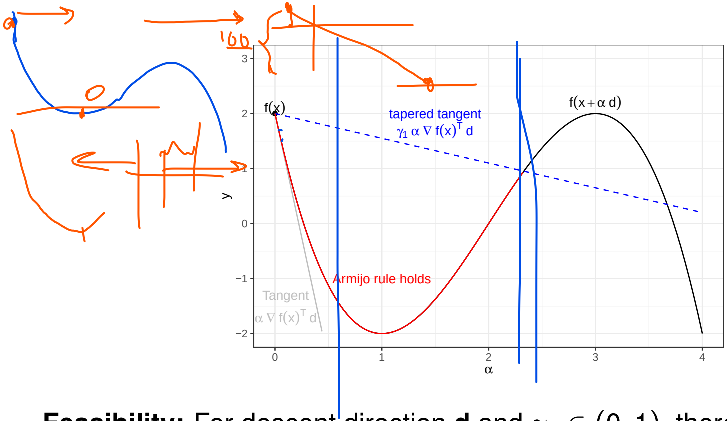


Fix $\gamma_1 \in (0,1)$. α satisfies **Armijo rule** in **x** for descent direction **d** if

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

Note: $\nabla f(\mathbf{x})^\top \mathbf{d} < 0$ (**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

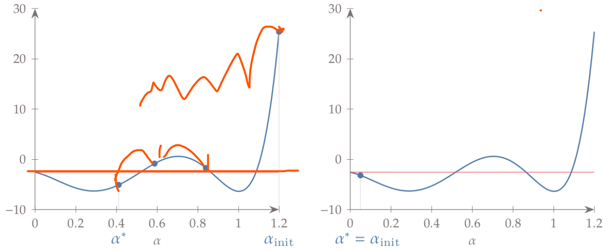
Procedure to meet the Armijo rule: **Backtracking** line search

Idea: Decrease α until Armijo rule is met

Algorithm Backtracking line search

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

0.8



(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 **x** for **d** if

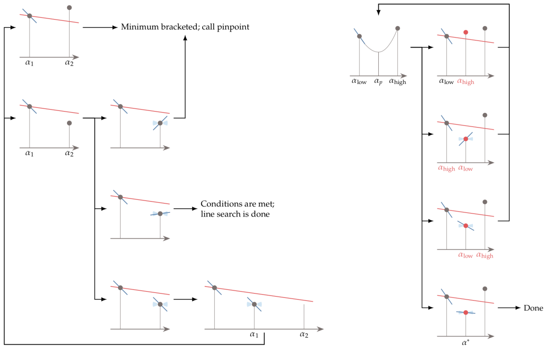
$$|\nabla f(\mathbf{x} + \alpha \mathbf{d})^\top \mathbf{d}| \leq \gamma_2 |\nabla f(\mathbf{x})^\top \mathbf{d}|.$$

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



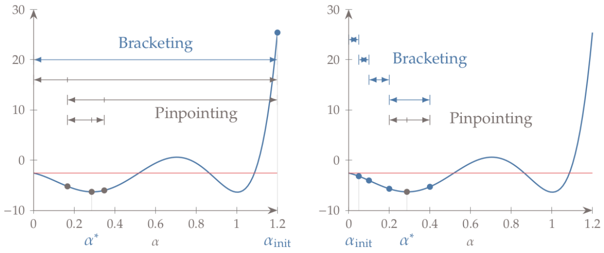
Left: Bracketing. **Right:** Pinpointing.

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

BRACKETING & PINPOINTING

Example:

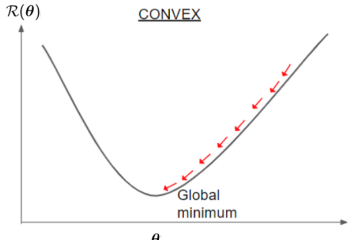
- Large initial step size results in quick bracketing but multiple pinpointing steps (**left**).
- Small initial step size results in multiple bracketing steps but quick pinpointing (**right**).



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

Optimization in Machine Learning

Deep dive
Gradient descent and optimality



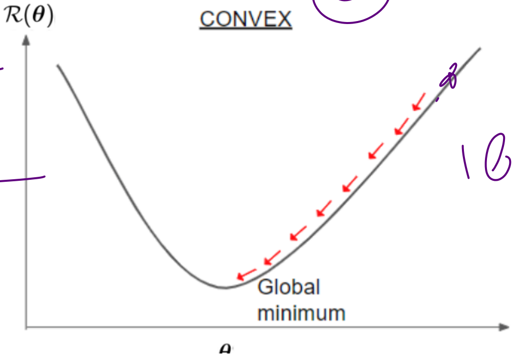
Learning goals

- 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.

α



L

$|\theta - \theta| L$

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

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

$1/k$

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

PROOF STRATEGY

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

Descent lemma:

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

2. Bound **error of one step**

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

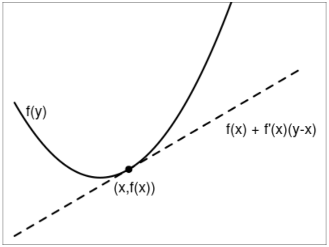
3. Finalize by **telescoping** argument

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

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

First order condition of convexity of g yields

$$\begin{aligned}
& g(\mathbf{x}) \geq g(\mathbf{x}^{[t]}) + \nabla g(\mathbf{x}^{[t]})^\top (\mathbf{x} - \mathbf{x}^{[t]}) \\
\Leftrightarrow \quad & \frac{L}{2}\|\mathbf{x}\|^2 - f(\mathbf{x}) \geq \frac{L}{2}\|\mathbf{x}^{[t]}\|^2 - f(\mathbf{x}^{[t]}) + (L\mathbf{x}^{[t]} - \nabla f(\mathbf{x}^{[t]}))^\top (\mathbf{x} - \mathbf{x}^{[t]}) \\
\Leftrightarrow \quad & \vdots \\
\Leftrightarrow \quad & f(\mathbf{x}) \leq f(\mathbf{x}^{[t]}) + \nabla f(\mathbf{x}^{[t]})^\top (\mathbf{x} - \mathbf{x}^{[t]}) + \frac{L}{2}\|\mathbf{x} - \mathbf{x}^{[t]}\|^2
\end{aligned}$$

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

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

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

FINALIZATION

Summing over iterations yields

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

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

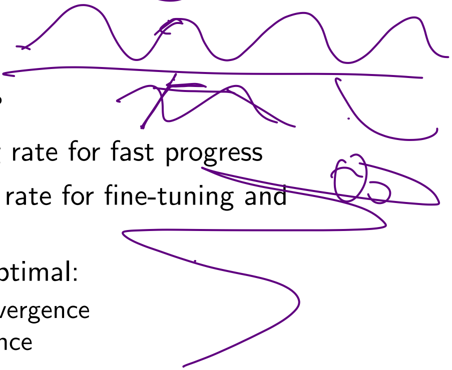
$$
\boxed{f(\mathbf{x}^{[t+1]}) - f(\mathbf{x}^*) \leq \frac{\|\mathbf{x}^{[0]} - \mathbf{x}^*\|^2}{2\alpha k}}
$$

Learning Rate Scheduling

Learning Rate Scheduling: Motivation

$\boxed{\alpha =}$ Wolfe

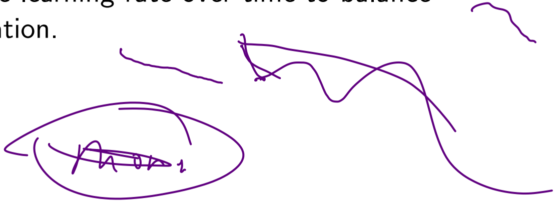
x_0 $\alpha := .$



Why schedule the learning rate?

- **Early training:** Large learning rate for fast progress
- **Late training:** Small learning rate for fine-tuning and convergence
- Fixed learning rate often suboptimal:
  - Too large $\Rightarrow$ oscillations, divergence
  - Too small $\Rightarrow$ slow convergence

Goal: Adaptively reduce learning rate over time to balance exploration and exploitation.

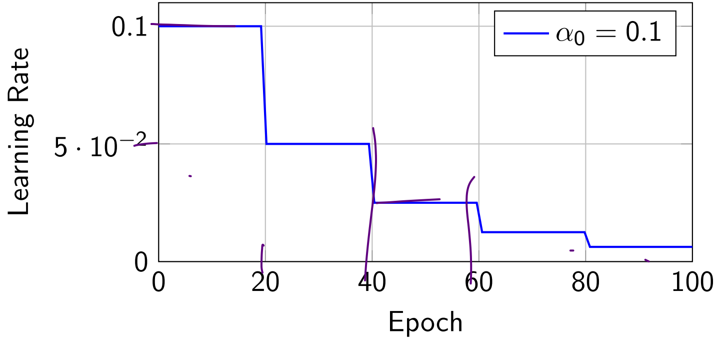


Step Decay Schedule

Formula:

$$\alpha(t) = \alpha_0 \cdot \gamma^{\lfloor t/s \rfloor}$$

where α_0 is initial learning rate, $\gamma \in (0, 1)$ is decay factor, s is step size.



Pros: Simple, widely used in practice (e.g., every 10 epochs multiply by 0.5)

Cons: Requires tuning γ and s; abrupt changes

Exponential Decay

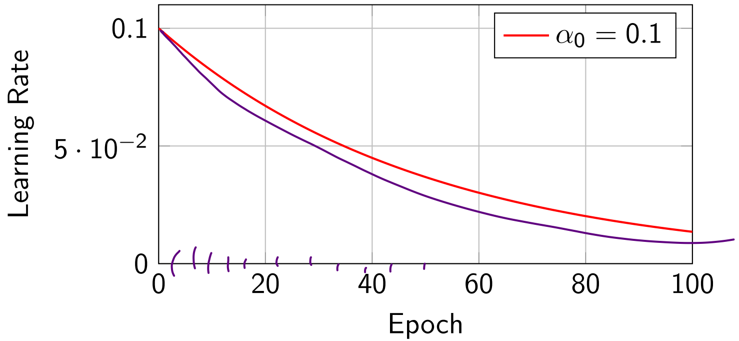
Formula:

$$\alpha(t) = \underline{\alpha_0} \cdot e^{-\lambda t}$$

$e^{-\lambda \cdot}$ $\frac{\alpha_0}{\cdot}$

$\rightarrow e^{30.3}$

where $\lambda > 0$ is the decay rate.



Pros: Smooth decay, theoretically motivated

Cons: Can decay too quickly or slowly depending on λ

1/t Decay (Inverse Time Decay)

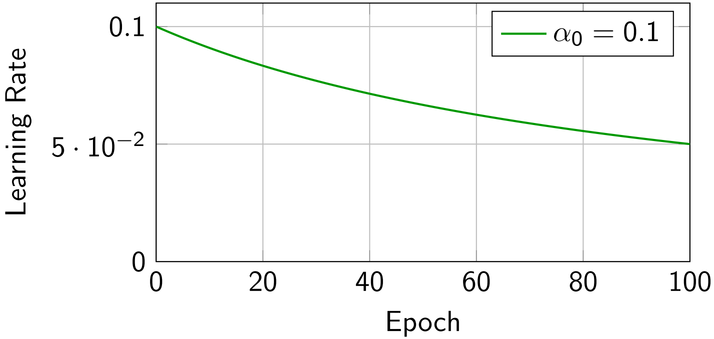
Formula:

$$\alpha(t) = \frac{\alpha_0}{\underline{1 + \lambda t}}$$

$$\frac{\alpha_0}{\underbrace{1}_{\sim} + \underbrace{\lambda}_{\sim} \cdot \underbrace{t}_{\sim}}$$

$t_0 = 0$

where $\lambda > 0$ controls decay speed.



Pros: Theoretical guarantees for convex optimization; never reaches zero

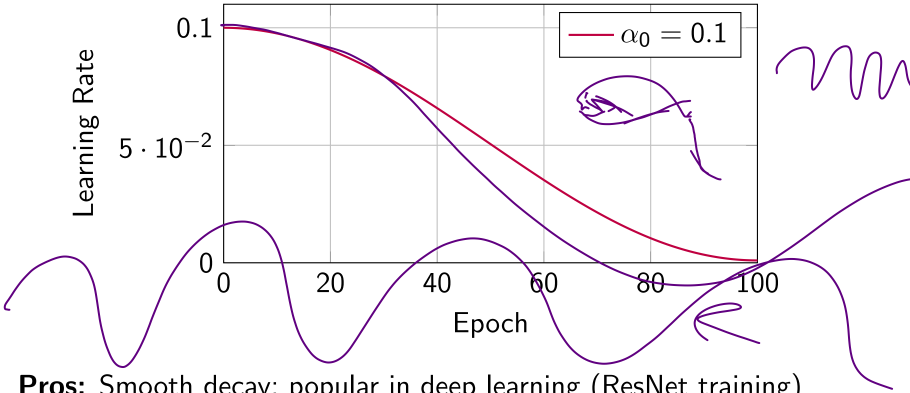
Cons: Decays slowly initially; may be too aggressive later

Cosine Annealing Schedule

Formula:

$$\alpha(t) = \alpha_{\text{min}} + \frac{1}{2}(\alpha_0 - \alpha_{\text{min}})\left(1 + \cos\left(\frac{t\pi}{T}\right)\right)$$

where T is the total number of epochs, α_{min} is minimum learning rate.



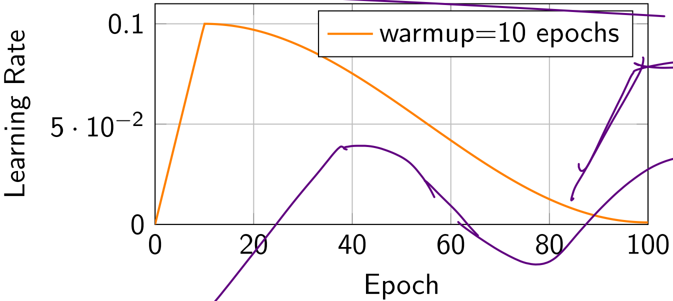
Pros: Smooth decay; popular in deep learning (ResNet training)
Cons: Needs to know total epochs in advance

Warmup Schedule

Idea: Gradually *increase* learning rate at the beginning, then decay.

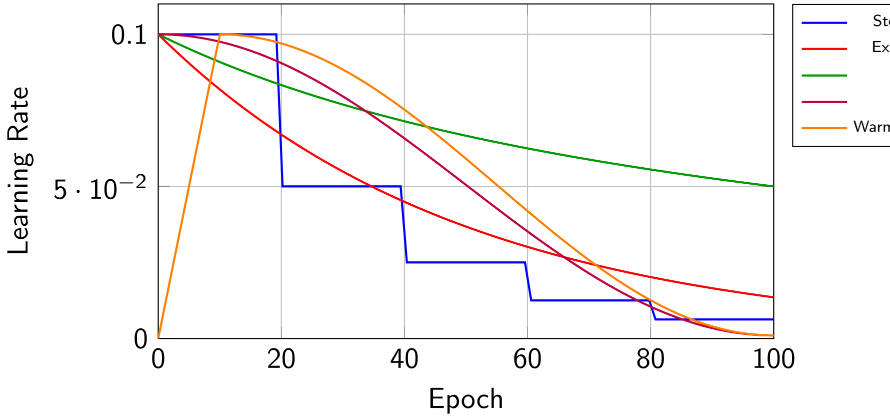
Linear Warmup + Cosine Decay:

$$\alpha(t) = \begin{cases} \alpha_0 \cdot \frac{t}{t_{\text{warmup}}} & \text{if } t \leq t_{\text{warmup}} \\ \alpha_{\min} + \frac{1}{2}(\alpha_0 - \alpha_{\min})\left(1 + \cos\left(\frac{(t - t_{\text{warmup}})\pi}{T - t_{\text{warmup}}}\right)\right) & \text{otherwise} \end{cases}$$



Motivation: Prevents instability with large batch sizes; used in Transformers (BERT, GPT)

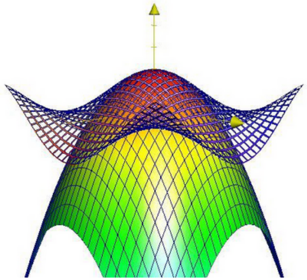
Comparison of Schedules



GD Weaknesses

Optimization in Machine Learning

First order methods
Weaknesses of GD – Curvature



Learning goals

- Effects of curvature
- Step size effect in GD

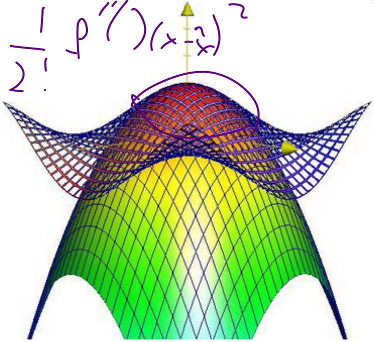
REMINDER: LOCAL QUADRATIC GEOMETRY

Locally approximate smooth function by quadratic Taylor polynomial:

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

$\tilde{x}$

$\mapsto$ $\frac{1}{2!} f''(\,)(x-\tilde{x})^2$



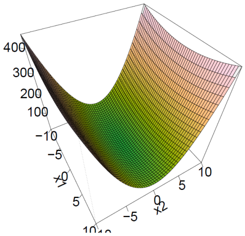
Source: `daniloroccatano.blog`.

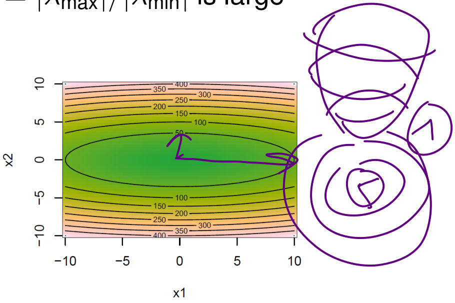
REMINDER: LOCAL QUADRATIC GEOMETRY

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

Recall for quadratic forms:

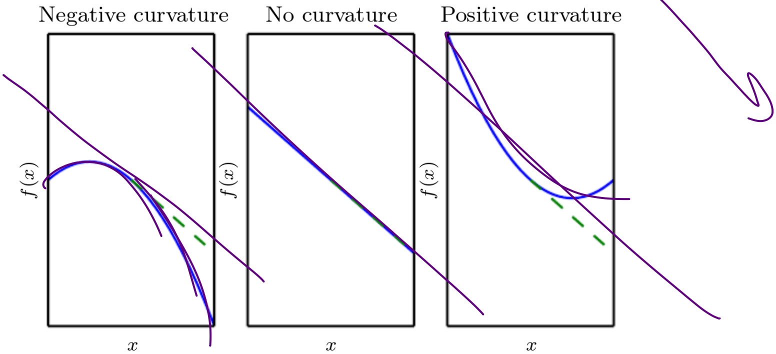
- Eigenvector $\mathbf{v}_{\text{max}}$ ($\mathbf{v}_{\text{min}}$) is direction of largest (smallest) curvature
- $\mathbf{H}$ called ill-conditioned if $\kappa(\mathbf{H}) = |\lambda_{\text{max}}|/|\lambda_{\text{min}}|$ is large





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 $\mathbf{g} = \nabla f$)

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

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

$\tilde{x} - \alpha g$

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

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

$\tilde{x} - x - \alpha y$

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

CURVATURE AND STEP SIZE IN GD

V_{max}

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

$\frac{1}{10}$ 10

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

$Av = \lambda v$

$vH = \lambda v$

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

0.1

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

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

$0.1 \quad \frac{1}{0} \rightarrow \alpha^* = 10$

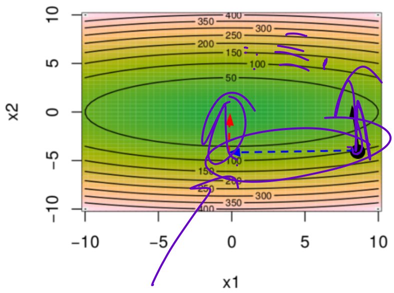
$\Rightarrow$ *Small* step sizes can be problematic.

0.1 , 10

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

CURVATURE AND STEP SIZE IN GD

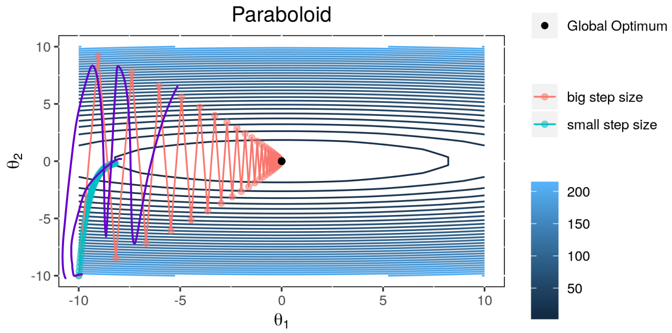
- What if **g** is not aligned with eigenvectors?
- Consider 2D case: Decompose **g** (black) into $\mathbf{v}_{\text{max}}$ and $\mathbf{v}_{\text{min}}$



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

CURVATURE AND STEP SIZE IN GD

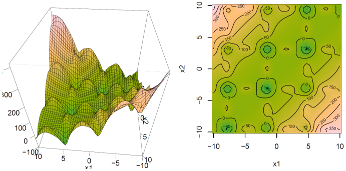
- GD is not aware of curvatures and can only walk along **g**
- Large step sizes result in “zig-zag” behaviour.
- Small step sizes result in weak performance.



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

Optimization in Machine Learning

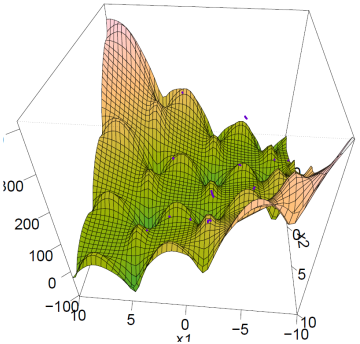
First order methods
GD – Multimodality and Saddle points

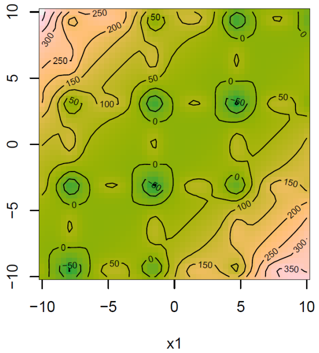


Learning goals

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

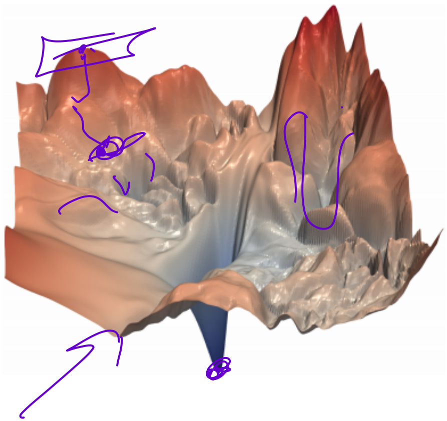
UNIMODAL VS. MULTIMODAL LOSS SURFACES

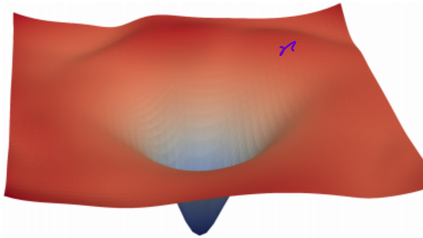




Snippet of a loss surface with many local optima

UNIMODAL VS. MULTIMODAL LOSS SURFACES

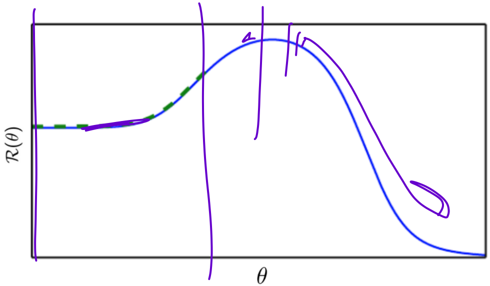




In deep learning, we often find multimodal loss surfaces.
Left: Multimodal loss surface. **Right:** (Nearly) unimodal loss surface.
(Source: Hao Li et al., 2017.

GD: ONLY LOCALLY OPTIMAL MOVES

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

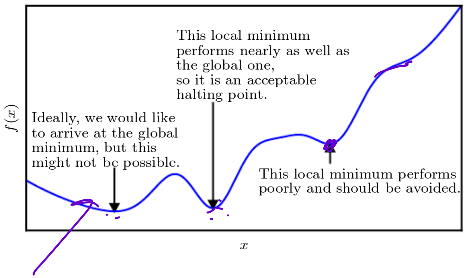


Source: Goodfellow et al., 2016

- 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

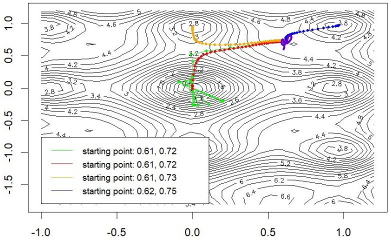
- **In practice:** Only local minima with high value compared to global minimium are problematic.

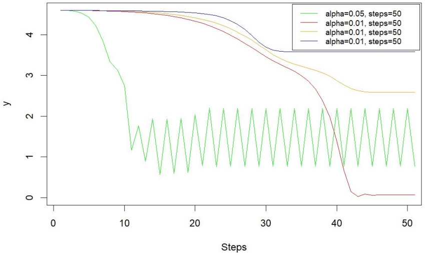


Source: Goodfellow et al., 2016

LOCAL MINIMA

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





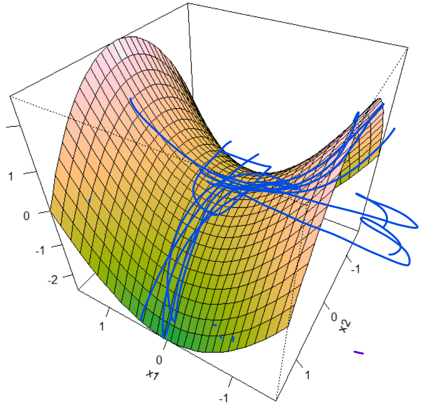
(Non-)Converging gradient descent for Ackley function

GD AT SADDLE POINTS

Example:

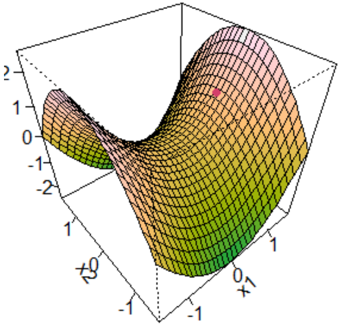
$$\begin{aligned} f(x_1, x_2) &= x_1^2 - x_2^2 \\ \nabla f(x_1, x_2) &= (2x_1, -2x_2)^\top \\ \boldsymbol{H} &= \begin{pmatrix} 2 & 0 \\ 0 & -2 \end{pmatrix} \end{aligned}$$

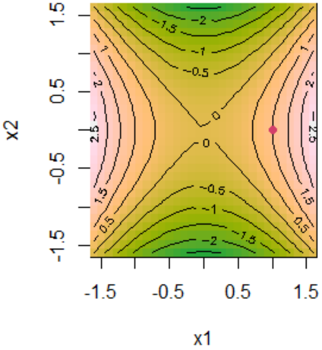
- Along x_1, curvature is positive ($\underbrace{\lambda_1 = 2 > 0}$).
- Along x_2, curvature is negative ($\lambda_2 = -2 < 0$).



EXAMPLE: SADDLE POINT WITH GD

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

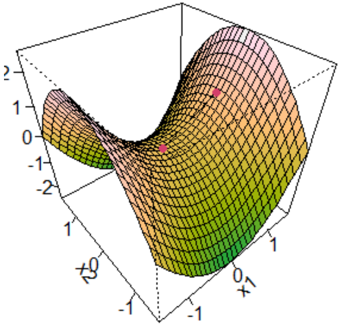


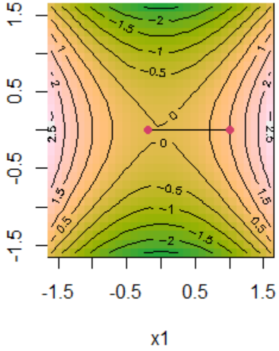


Red dot: Starting location

EXAMPLE: SADDLE POINT WITH GD

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

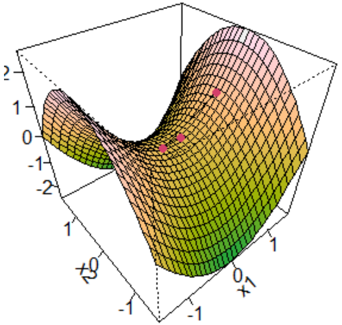


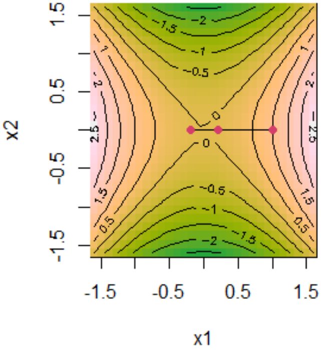


Step 1 ...

EXAMPLE: SADDLE POINT WITH GD

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

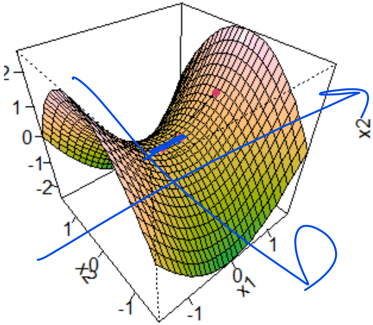


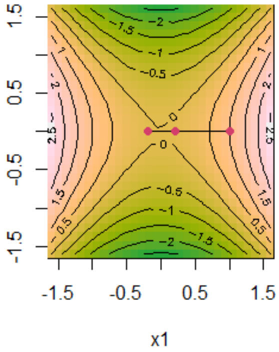


... Step 2 ...

EXAMPLE: SADDLE POINT WITH GD

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

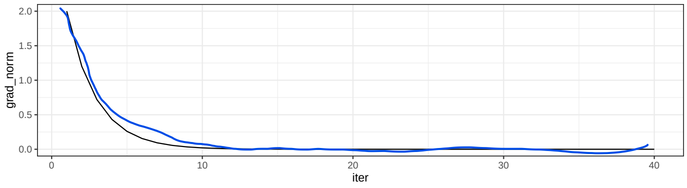




... Step 10 ... got stuck and cannot escape saddle point

EXAMPLE: SADDLE POINT WITH GD

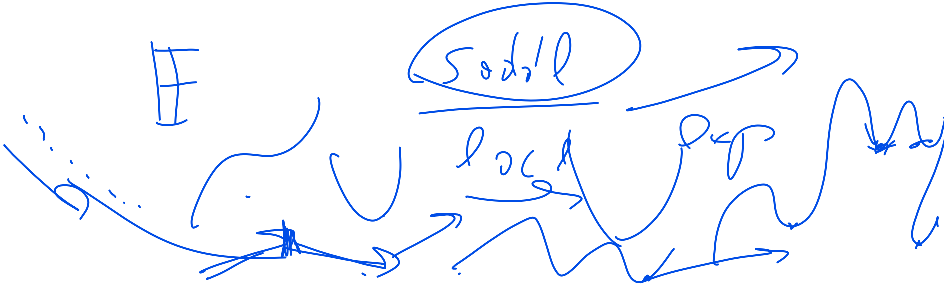
- How do saddle points impair optimization?
- Gradient-based algorithms **might** get stuck in saddle points



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

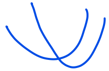
SADDLE POINTS IN NEURAL NETWORKS

- 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
- 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.



[illegible]

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.

H

- 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)).