Introduction to Machine Learning (IML)

LECTURE #7: OPTIMIZATION

236756 – 2024 SPRING – TECHNION

LECTURER: NIR ROSENFELD

Today

- **Part II**: *the different aspects of learning*
  1. Statistics: generalization and PAC theory
  2. Modeling:
     - Error decomposition
     - Regularization
     - Model selection
  3. Optimization: convexity, gradient descent (today)
  4. Practical aspects and potential pitfalls

Optimization

Optimizing the learning objective

- **Regularized Risk Minimization** (RRM):

$$\underset{\theta \in \mathbb{R}^d}{\operatorname{argmin}} \boxed{\frac{1}{m} \sum_{i=1}^{m} \underbrace{\ell(y_i, \theta^\top x_i)}_{\text{loss}}} + \underbrace{\lambda R(\theta)}_{\text{regularization}}$$

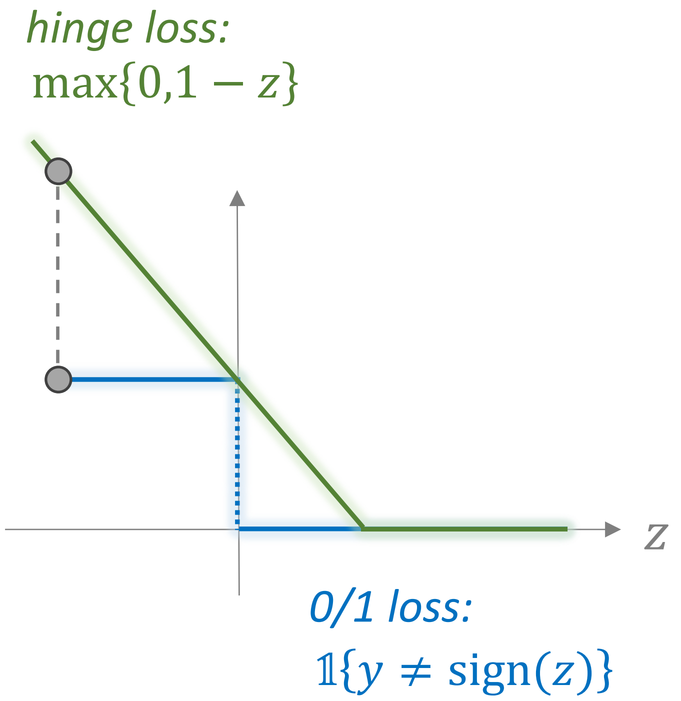
- **Recall**: ERM of 0/1 is hard (discrete)
- **Solution**: continuous proxy

Optimizing the learning objective

- **Regularized Risk Minimization** (RRM):

$$\underset{\theta \in \mathbb{R}^d}{\operatorname{argmin}} \boxed{\frac{1}{m}\sum_{i=1}^{m} \underbrace{\ell(y_i, \theta^\top x_i)}_{\textbf{loss}}} + \underbrace{\lambda R(\theta)}_{\textbf{regularization}}$$

- **Recall**: ERM of 0/1 is hard (discrete)
- **Solution**: continuous proxy
- Soft SVM's *hinge loss* is one option
- **Problem**: unnecessarily large penalties for far from margin

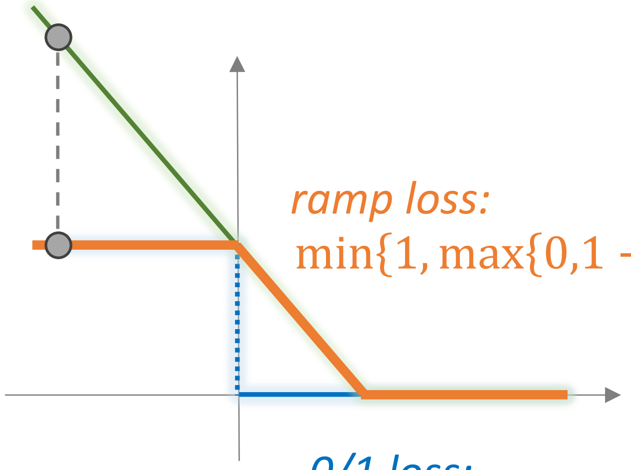


Optimizing the learning objective

- **Regularized Risk Minimization** (RRM):

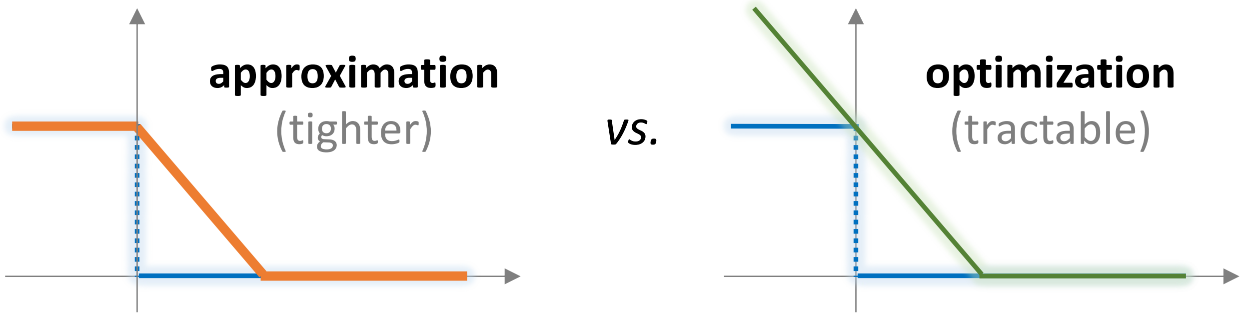
$$\underset{\theta\in\mathbb{R}^d}{\operatorname{argmin}} \boxed{\frac{1}{m}\sum_{i=1}^{m} \underbrace{\ell(y_i, \theta^\top x_i)}_{\text{loss}}} + \underbrace{\lambda R(\theta)}_{\text{regularization}}$$

- **Recall**: ERM of 0/1 is hard (discrete)
- **Solution**: continuous proxy
- Soft SVM's *hinge loss* is one option
- **Problem**: unnecessarily large penalties for far from margin
- **Alternative**: *ramp loss*
- **Problem**: optimization… (today we'll see why)



Optimizing the learning objective

- **Tradeoff**:



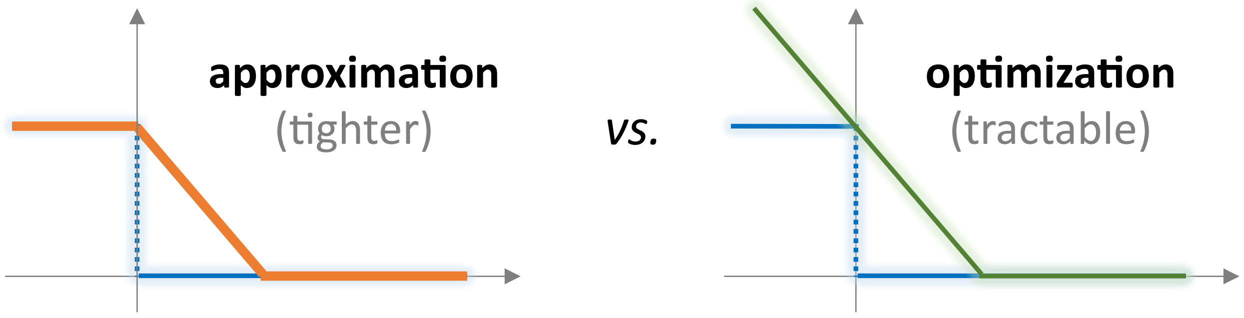
- Many proxies out there!
- [DESMOS]

Proxy losses

https://www.desmos.com/calculator/f2wlucevfq

Optimizing the learning objective

- **Tradeoff**:



- Many proxies out there – which is better?
- No definite answer
- But we can say some things sometimes
- **Today**: focus on optimization (but also generalization)
- Functional properties that give optimization guarantees:
  - (sub-)differentiable
  - convex
  - Lipschitz
  - smooth
  - bounded

1. $u(x) = \{x < 0{:}1, x \geq$
2. $\max(0, 1 - x)$
3. e^{-x}
4. $\ln(1 + e^{-x})$
5. $\frac{1}{1 + e^{x}}$
6. $\max(0, 1 - x)^2$
7. …

Gradient descent


https://en.wikipedia.org/wiki/Gradient_descent

One algorithm to rule them all

- **Goal**: solve $\min_{\theta \in \mathbb{R}^d} f(\theta)$
- (in learning: f is the learning objective, θ are model parameters)

- **Approach**: Gradient Descent (GD) [~Cauchy 1847]

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

- "First order iterative method"
- Applies to <u>any</u> differentiable learning objective
- Forms the basis for many other optimization algorithms
  (we'll see some today)

Gradient descent

- **Definition**:
  The *gradient* of f w.r.t. θ is a vector function of partial derivatives,

$$\nabla f = \begin{pmatrix} \frac{\partial f}{\partial \theta_1} \\ \vdots \\ \frac{\partial f}{\partial \theta_d} \end{pmatrix}$$

- *Gradient evaluation* at $\bar{\theta} \in \mathbb{R}^d$:

$$\nabla f(\bar{\theta}) \in \mathbb{R}^d$$

<u>Gradient Descent (GD)</u>

- Initialize θ_0 (e.g., $\theta_0 = \vec{0}$)
- Repeat:
  - $\theta_{t+1} = \theta_t - \eta \nabla f(\theta_t)$
- Until convergence
  (e.g., $\|\theta_{t+1} - \theta_t\| \leq \epsilon$)

Gradient descent

- **View I**: fog in mountains
- **View II**: local greedy descent
- **View III**: optimize simple approximations

$$@t: \; f(\theta) \approx f(\theta_t) + \nabla f(\theta_t)^\top (\theta - \theta_t)$$

- **View IV**: Taylor expansion + tradeoff
  - $f(\theta_t + \delta) \approx f(\theta_t) + \nabla f(\theta_t)^\top \delta, \;\; \delta \in \mathbb{R}^d$
  - $\theta_{t+1} = \theta_t + \delta$ close to θ_t, i.e., $\|\delta\|_2$ is small
  - $\theta_{t+1} = \underset{\theta}{\operatorname{argmin}} \frac{1}{2}\|\theta - \theta_t\|_2^2 + \eta\big(f(\theta_t) + \nabla f(\theta_t)^\top(\theta - \theta_t)\big)$
  - $\partial = 0 \Rightarrow \theta_{t+1} = \theta_t - \eta \nabla f(\theta_t)$

Gradient Descent (GD)

- Initialize θ_0 (e.g., $\theta_0 = \vec{0}$)
- Repeat:
  - $\theta_{t+1} = \theta_t - \eta \nabla f(\theta_t)$
- Until convergence (e.g., $\|\theta_{t+1} - \theta_t\| \leq \epsilon$)

Gradient descent

- **View I**: fog in mountains
- **View II**: local greedy descent
- **View III**: optimize simple approximations

  $$@t: \ \ f(\theta) \approx \nabla f(\theta_t)^\top \theta$$

- **View IV**: Taylor expansion + tradeoff
  - $f(\theta_t + \delta) \approx f(\theta_t) + \nabla f(\theta_t)^\top \delta, \ \ \delta \in \mathbb{R}^d$
  - $\theta_{t+1} = \theta_t + \delta$ close to θ_t, i.e., $||\delta||_2$ is small
  - $\theta_{t+1} = \underset{\theta}{\operatorname{argmin}} \frac{1}{2}||\theta_t - \theta||_2^2 + \eta\big(f(\theta_t) + \nabla f(\theta_t)^\top(\theta - \theta_t)\big)$
  - $\partial = 0 \Rightarrow \theta_{t+1} = \theta_t - \eta \nabla f(\theta_t)$

Gradient Descent (GD)

- Initialize θ_0 (e.g., $\theta_0 = \vec{0}$)
- Repeat:
  - $\theta_{t+1} = \theta_t - \eta \nabla f(\theta_t)$
- Until convergence (e.g., $||\theta_{t+1} - \theta_t|| \leq \epsilon$)

Need to determine:

1. how to initialize
2. step size or learning rate η
3. stopping criterion

Gradient descent

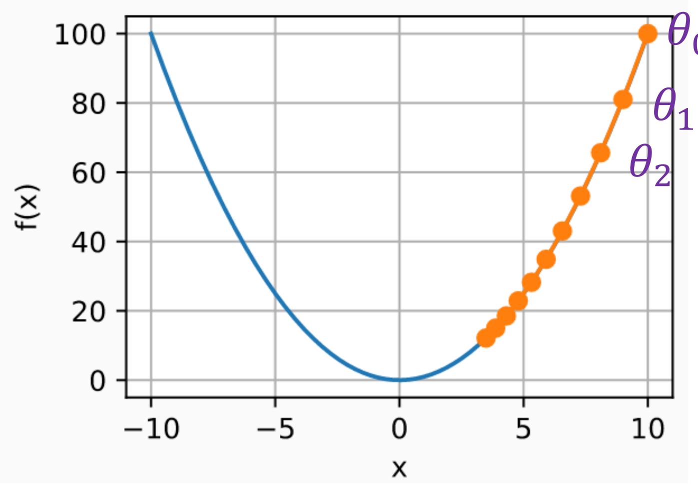
- **Definition**: Let $B_\epsilon(\theta) = \{v : \ ||\theta - v|| \leq \epsilon\}$ be a ball of radius ϵ around θ. Then θ is a ***local minimum*** of f if

$$\exists \epsilon > 0 \ \forall v \in B_\epsilon(\theta), \ f(\theta) \leq f(v)$$

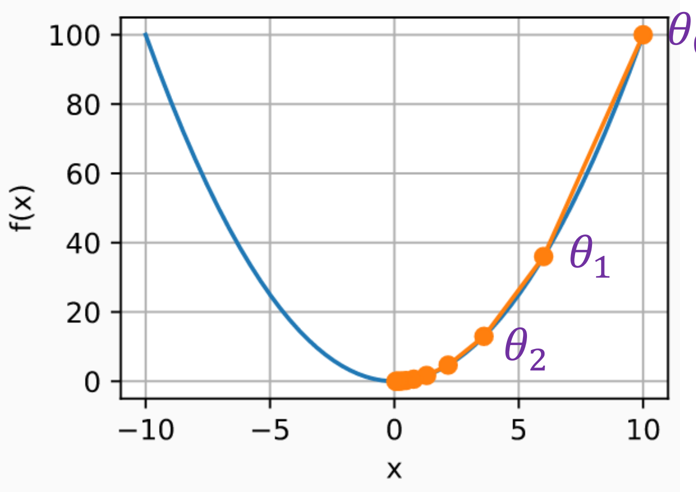
- **Claim**: if η is small enough, then $f(\theta_{t+1}) < f(\theta_t)$ for any non-minimum θ_t (this is the "descent" part of "gradient descent")
- **Proof**: consider non-minimum θ_t and negate
- **Corollary**: GD converges* to a local minimum, or stops (saddle point!)
  * for an appropriate choice of η
- **Trick**: use time-varying learning rate η_t, for example, $\eta_t = 1/t$ (wont' prove)
- **Note**: if θ_t is a local minimum then $\nabla f(\theta_t) = 0$ (but not iff!)

Learning rate

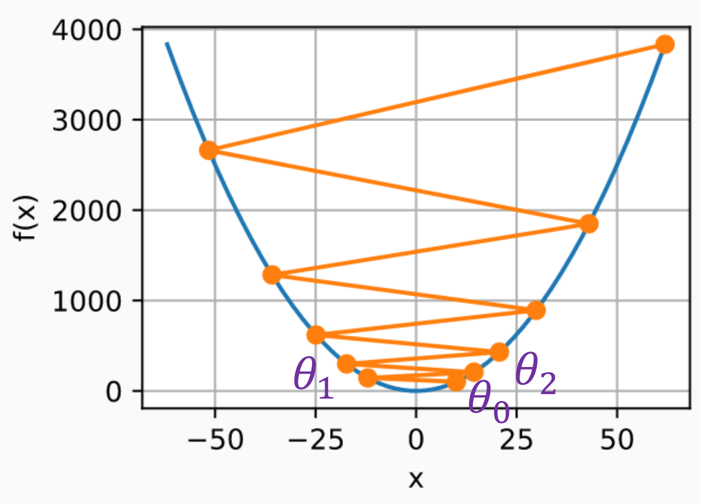
- **Q1**: So GD converges for learning rate $\eta_t = 1/t$. Are we done?
- **A1**: No!



$\eta_t = 1/t$
slow convergence
(too slow!)



$\eta_t = 1/\sqrt{t}$
"just right"
(how can we know?)



$\eta_t = 10$
divergence
(too fast!)

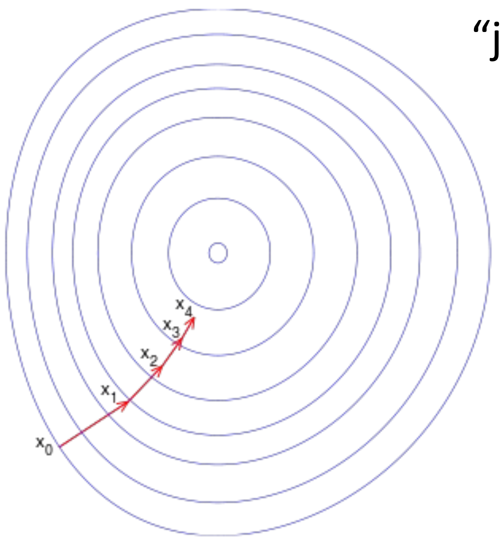
(illustrative)

https://d2l.ai/chapter_optimization/

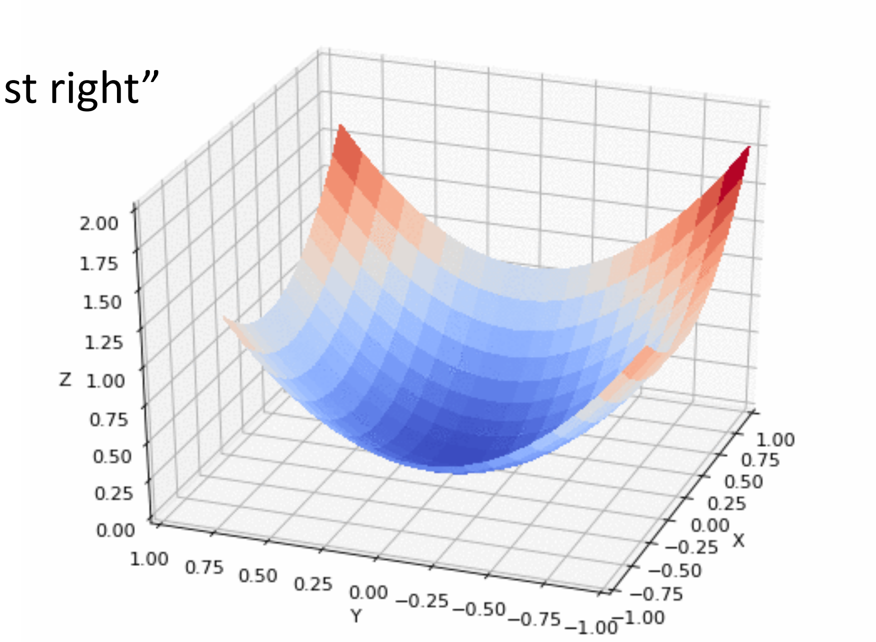
Learning rate

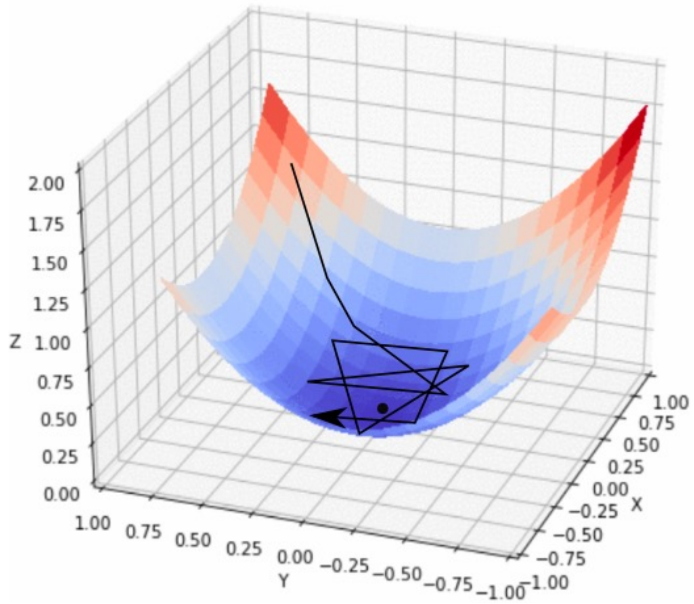
- **Q1**: So GD converges for learning rate $\eta_t = 1/t$. Are we done?
- **A1**: No!

divergence?
slow convergence?



“just right”





- Choosing the learning rate can be tricky – we will return to this.

https://blog.paperspace.com/intro-to-optimization-in-deep-learning-gradient-descent/

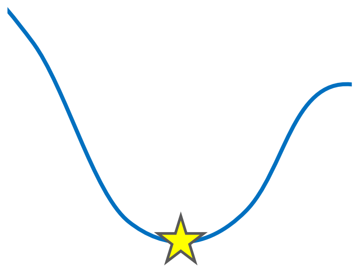
When to stop

- **Q2**: So GD converges in theory. But in practice, how do we know when (and if) it did?
- **A2**: Can't "know", but can estimate.

- Set criteria for stopping, for example:
  - stop when $||\theta_{t+1} - \theta_t|| \leq \epsilon$
  - or when $||f(\theta_{t+1}) - f(\theta_t)|| \leq \epsilon$
  - or when $||\nabla f(\theta_t)|| \leq \epsilon$
  - or …
- Actually, knowing when to stop is (also) tricky – and has statistical implications! (surprised?) We will return to this as well.

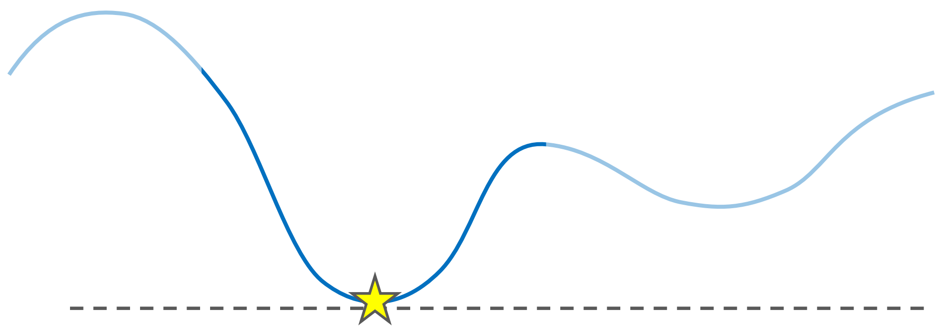
Local minima

- **Q3**: So GD can converge to a local minimum. Should we be happy?
- **A3**: It depends…



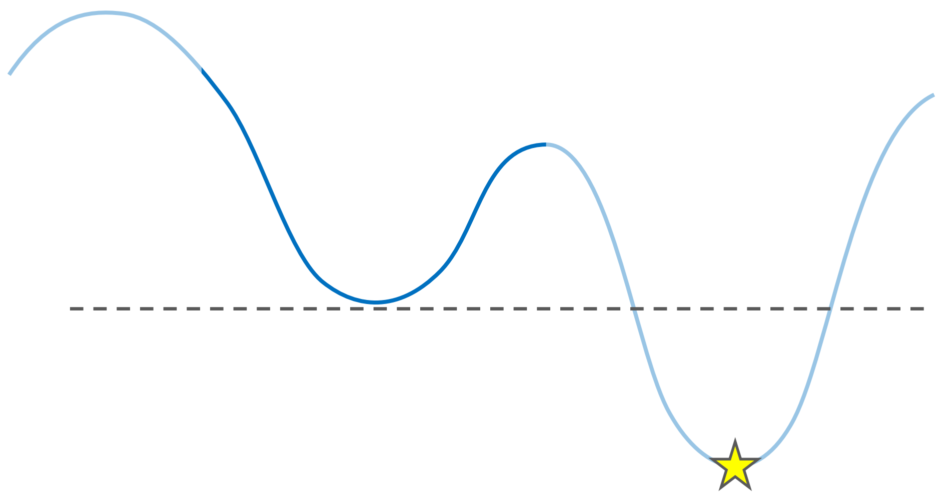
Local minima

- **Q3**: So GD can converge to a local minimum. Should we be happy?
- **A3**: It depends…



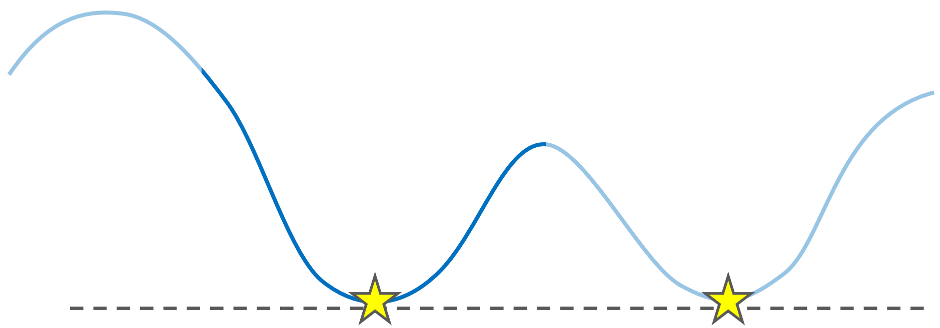
Local minima

- **Q3**: So GD can converge to a local minimum. Should we be happy?
- **A3**: It depends…



Local minima

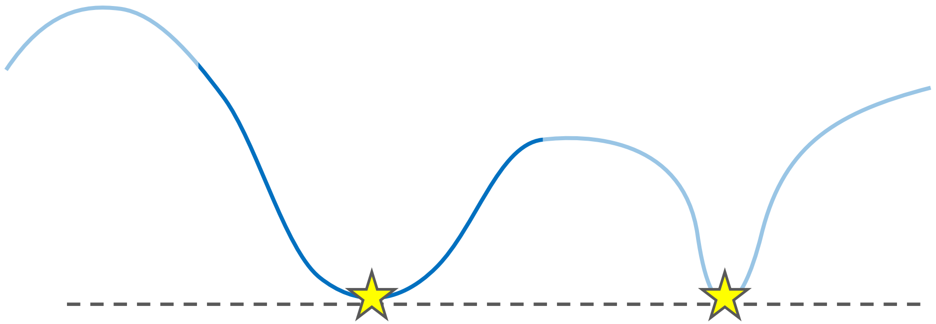
- **Q3**: So GD can converge to a local minimum. Should we be happy?
- **A3**: It depends…



- **Goal:** find conditions on f for which GD is guaranteed to work well

Local minima

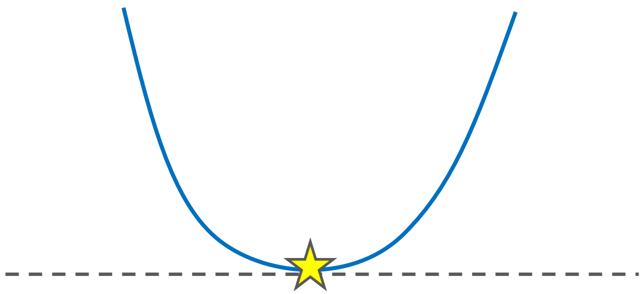
- **Q3**: So GD can converge to a local minimum. Should we be happy?
- **A3**: It depends…



- **Goal:** find conditions on f for which GD is guaranteed to work well

Local minima

- **Q3**: So GD can converge to a local minimum. Should we be happy?
- **A3**: It depends…



- **Goal:** find conditions on f for which GD is guaranteed to work well

Convexity

Descending with guarantees

- Recall foggy mountain story
- When will the "going down" approach help you reach your cabin?

- When mountain range is:
    1. continuous
    2. smooth
    3. **has a single valley**

- Enter **convexity.**
- [on board]

https://en.wikipedia.org/wiki/Gradient_descent

Convex sets

- **Definition:** *convex combination*:

$$\alpha u + (1-\alpha)v \quad \text{for} \quad u, v \in \mathbb{R}^d, \alpha \in [0,1]$$

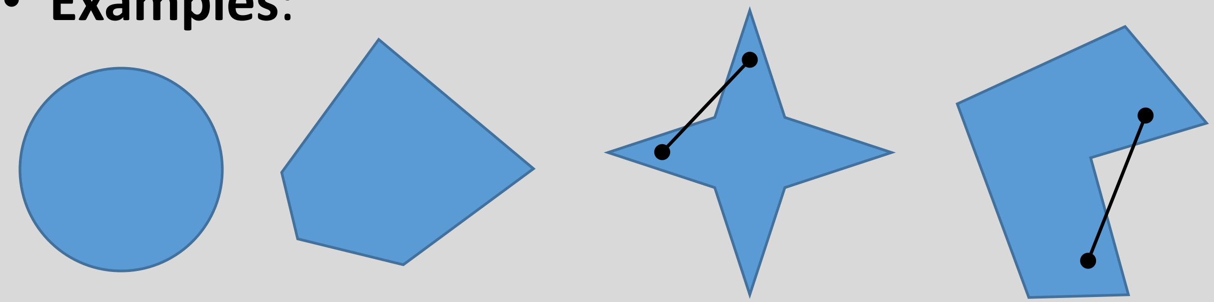
- As interpolation – point on line between u, v
- **Definition**: C is a *convex set* if

$$\forall u, v \in C, \text{ any convex combination of } u, v \text{ is also in } C$$

- **Examples**:

Union of Convex is not necessarily Convex

Intersection of Convex is also Convex



$$H_{\theta,b} = \{x : w^\top x \geq b\} \qquad \bigcap_\theta H_\theta \qquad \mathbb{R}^d$$

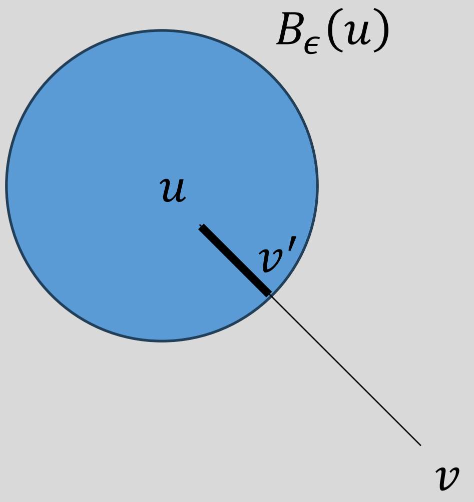
- **Definition:** For C convex set, $f: C \to \mathbb{R}$ is *convex function* if $\forall u, v \in C, \alpha \in [0,1]$:

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

x^2? $w^\top x$? $\sqrt{x}$? $\max\{0, x\}$?

Convex sets

- **Claim:** Let f be *convex*, then any local minima is also global
- **Proof**: Let u be a local minima, so exist ϵ s.t. for all $w \in B_\epsilon(u)$, we have $f(u) \le f(w)$.
- We will show that $f(u) \le f(v)$ for any v.
- Let v. Choose α s.t. $v' = u + \alpha(v - u) \in B_\epsilon(u)$. Note v' is on the line between u and v.
- Since v' is in the ball, then:
- $$\begin{aligned} f(u) \le f(v') &= f\big(u + \alpha(v-u)\big) \\ &= f(\alpha v + (1-\alpha)u) \\ &\le \alpha f(v) + (1-\alpha) f(u) \end{aligned}$$



- where the second inequality is from convexity.
- Rearranging, we get $f(u)$

GD convergence rates

If f is convex and:

- L-Lipschitz: $f(\theta_t) - f(\theta^*) = O\left(\frac{L}{\sqrt{t}}\right)$
- β-Smooth: $f(\theta_t) - f(\theta^*) \leq \frac{2\beta\|\theta_0 - \theta^*\|}{t+4} = O\left(\frac{\beta}{t}\right)$
  - Lower bound: $\Omega\left(\frac{\beta}{t^2}\right)$
- σ-Strongly convex: $f(\theta_t) - f(\theta^*) = O(e^{-2\sigma t})$

f is L-Lipschitz if :

$$\|f(u) - f(w)\| \leq L\|u - w\|$$

f is β-Smooth if ∇f is β-Lipschitz:

$$\|\nabla f(u) - \nabla f(w)\| \leq \beta\|u - w\|$$

f is σ-Strongly convex if

$$f(\alpha w + (1-\alpha)u) \leq \alpha f(w) + (1-\alpha)f(u) - \frac{\sigma}{2}\alpha(1-\alpha)\|u - w\|^2$$

- **Rate depends on:**
  - Type
  - Parameter
  - Initial guess!

GD convergence rates

- which is smooth?
- which is Lipschitz?
- which is convex?
- which is differentiable?

Optimization for learning

Convex learning problems

- **Recall ERM/RLM:**

$$\operatorname*{argmin}_{\theta} \frac{1}{m} \sum_{i=1}^{m} \ell\big(y_i, f_\theta(x_i)\big) + \lambda R(\theta)$$

- Before we asked if the loss function itself (i.e., ℓ) is convex
- But what matter is if the entire <u>objective</u> is convex or not (in θ)
- How can we tell?

Composition rules

- **Convexity-preserving operations:**
  1. **scaling**: f convex, $\alpha \geq 0 \Rightarrow \alpha f$ convex [easy]
  2. **sum**: f, g convex $\Rightarrow f + g$ convex [tirgul]
  3. **noneg. weighted sum**: f_i convex, $\alpha_i \geq 0 \Rightarrow \sum_i \alpha_i f_i$ convex
  4. **composition**: f convex, g linear $\Rightarrow f \circ g$ convex [on board]
  5. …
- (these are sufficient conditions, but not necessary)

Composition of convex and linear is convex

- **Claim**: if g is convex and h is linear, then $f = g \circ h$ is convex

- **Proof**: $f(\alpha u + (1 - \alpha)v) = g\big(h(\alpha u + (1 - \alpha)v)\big) =_{h\ linear}$

$$g\big((\alpha u + (1 - \alpha)v)^T x + b\big) = g(\alpha u^T x + (1 - \alpha)v^T x + b) =$$

$$g(\alpha u^T x + (1 - \alpha)v^T x + (\alpha + 1 - \alpha)b) =$$

$$g(\alpha(u^T x + b) + (1 - \alpha)(v^T x + b)) = g\big(\alpha h(u) + (1 - \alpha)h(v)\big)$$

$$\leq_{g\ convex} \alpha g\big(h(u)\big) + (1 - \alpha)g\big(h(v)\big) = \alpha f(u) + (1 - \alpha)f(v)$$

Convex learning problems

- **Claim**: The following conditions are sufficient for the learning objective to be convex (in θ):
  1. $\ell(y,\cdot)$ is convex (in it's second argument)
  2. $R(\cdot)$ is convex
  3. f_θ is linear (in θ, i.e., $f_\theta(x) = \theta^\top x$)

Convex learning problems

- **Claim**: The following conditions are sufficient for the learning objective to be convex (in θ):
  1. $\ell(y,\cdot)$ is convex (in it's second argument)
  2. $R(\cdot)$ is convex
  3. f_θ is linear (in θ, i.e., $f_\theta(x) = \theta^\top x$)
- **Let's try** →

$$\underbrace{\underset{\underset{\textit{conv}}{\uparrow}}{\underset{\theta\in\mathbb{R}^d}{\operatorname{argmin}}}\;\underbrace{\overset{\overset{\geq 0}{\downarrow}}{\frac{1}{m}}\sum_{i=1}^{m}\underbrace{\overset{\overset{\textit{conv}}{\downarrow}}{\ell}\big(y_i,\overset{\overset{\textit{lin}}{\downarrow}}{f_\theta}(x_i)\big)}_{\textit{conv}}}_{\textit{conv}} + \underbrace{\overset{\overset{\geq 0}{\downarrow}}{\lambda}\overset{\overset{\textit{conv}}{\downarrow}}{R}(\theta)}_{\textit{conv}}}_{\textit{conv}}$$

Convex learning problems

- **Claim**: The following conditions are sufficient for the learning objective to be convex (in θ):
  1. $\ell(y,\cdot)$ is convex (in it's second argument)
  2. $R(\cdot)$ is convex
  3. f_θ is linear (in θ, i.e., $f_\theta(x) = \theta^\top x$)
- **Let's try** $\longrightarrow$

$$\underbrace{\underbrace{\underset{\theta\in\mathbb{R}^d}{\operatorname{argmin}}}_{\text{conv}} \underbrace{\overset{\geq 0}{\frac{1}{m}}\sum_{i=1}^{m} \underbrace{\overset{\text{conv}}{\ell}\big(y_i, \overset{\text{lin}}{f_\theta}(x_i)\big)}_{\text{conv}}}_{\text{conv}} + \underbrace{\overset{\geq 0}{\lambda}\overset{\text{conv}}{R}(\theta)}_{\text{conv}}}_{\text{conv}}$$

- **Claim:** The Soft SVM objective is convex
- **Need to prove:**
  1. $\max\{0, 1 - z\}$ is convex [ex]
  2. ℓ_2-norm squared is convex [tirgul]
- **Corollary**: can solve with GD!

$$\underset{\theta\in\mathbb{R}^d}{\operatorname{argmin}} \frac{1}{m}\sum_{i=1}^{m} \max\{0, 1 - y_i \cdot \theta^\top x_i\} + \lambda\|\theta\|_2^2$$

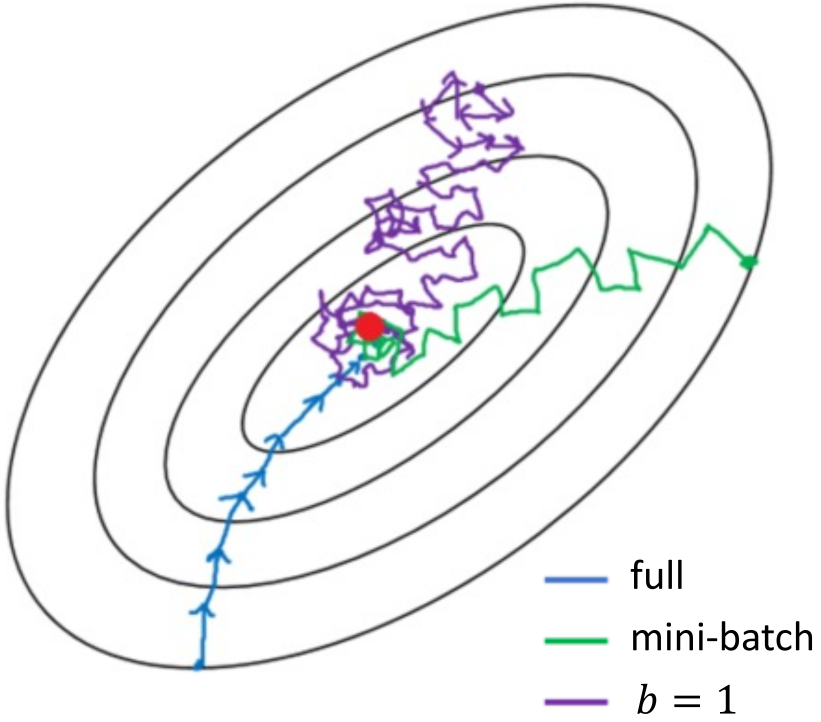
Variations, extensions, and beyond

The computational cost of GD

- For running GD we need to compute gradients of the learning objective.
- **Note**: $\nabla\left(\frac{1}{m}\sum_{i=1}^{m}\ell\left(y_i, f_\theta(x_i)\right)\right) = \frac{1}{m}\sum_{i=1}^{m}\underbrace{\nabla\ell\left(y_i, f_\theta(x_i)\right)}_{\nabla_i}$
- **For Soft SVM**:
  - For each example i, computing $\nabla \max\{0, 1 - y_i \cdot \theta^\top x_i\}$ costs: $O(d)$
  - For m examples, overall cost is: $O(dm)$
- GD can be costly!
- **Solution**: use *approximate* gradients

Taking approximate gradient steps

- **Idea**: gradient is average, can replace with average over smaller sub-sample!
- **Rationale**: smaller average has same expected value (=unbiased), but is noisy
- **Stochastic Gradient Descent (SGD):***
    1. Sample small random "mini-batch" $B \subset S$ of size b
    2. Compute average gradient $\bar{\nabla} = \frac{1}{b}\sum_{i \in B} \nabla_i$
    3. Apply *approximate* gradient step $\theta_{t+1} = \theta_t - \eta\bar{\nabla}$
- **Pros**: reduces compute time (significantly!)
- **Cons**: adds noise (often worth it; sometimes even helpful!)
- Hyper parameter b trades off compute time with noise



* The name "SGD" typically refers to the $b = 1$ case

Looking back

- Consider 2D quadratic with highly varying curvatures
  - On low-curvature dimension, GD crawls hesitantly
  - On high-curvature dimension, GD oscillates frantically
- Adding “momentum” sorts this out:

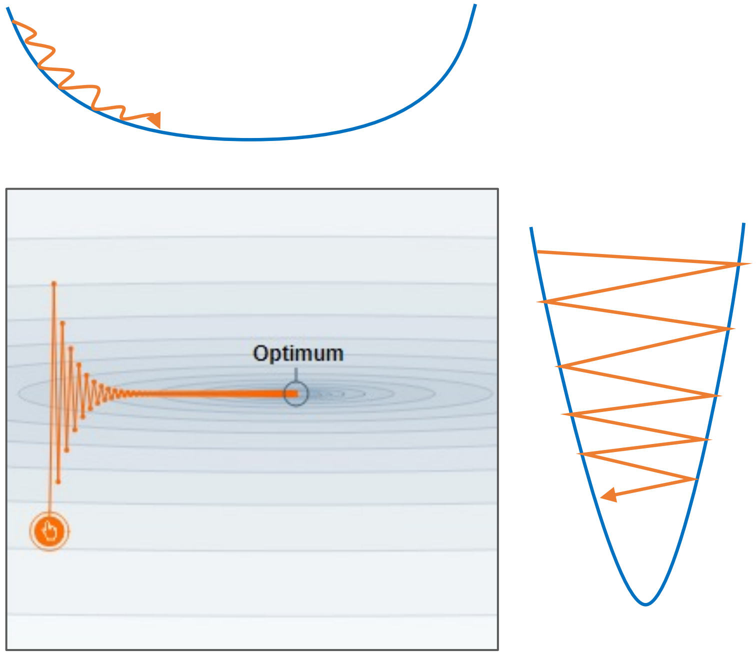
$$v_{t+1} = \gamma v_t - \eta \nabla f(\theta_t) \quad \rightarrow \text{velocity/“memory”}$$

$$\theta_{t+1} = \theta_t + v_{t+1} \quad \rightarrow \text{position}$$

$$= \theta_t - \eta \nabla f(\theta_t) + \gamma v_t$$

(imagine a rolling ball)

- Effects of momentum:
  - *increases* in dimensions where gradient preserves direction
  - *decreases* in dimensions where gradient direction varies
- Typical β values: 0.9, 0.95, 0.99



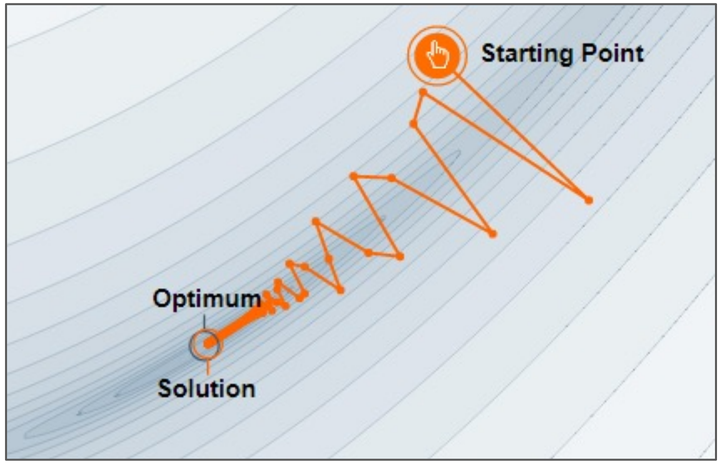
$$\theta_{t+1} = \theta_t - \eta \nabla$$

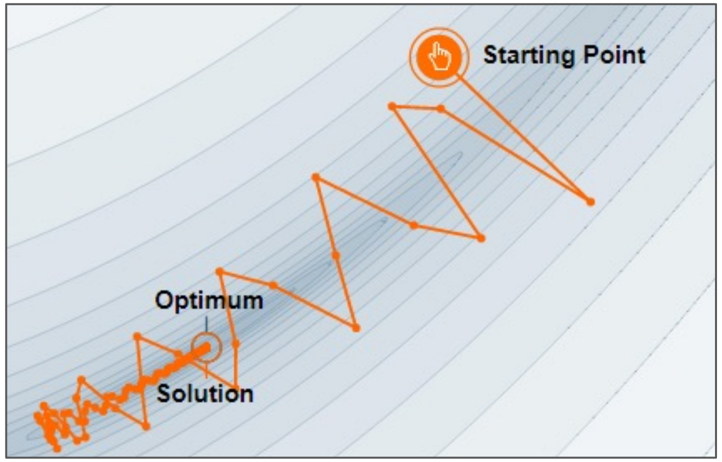
same rate η in all directions!

https://distill.pub/2017/momentum/

Looking forward

- Momentum is great, but can overshoot
- **Solution**: look into *future*
- **Idea:** *imagine* momentum has been applied,
  then compute gradient





https://distill.pub/2017/momentum/

Looking forward

- Momentum is great, but can overshoot
- **Solution**: look into *future*
- **Idea:** *imagine* momentum has been applied, then compute gradient

- **Nesterov's accelerated gradient:**

$$\tilde{\theta}_{t+1} = \theta_t + \beta v_t$$

$$v_{t+1} = \beta v_t - \eta \nabla f\left(\tilde{\theta}_{t+1}\right)$$

$$\theta_{t+1} = \theta_t + v_{t+1}$$

- **Equivalent**:

$$v_{t+1} = \beta v_t - \eta \nabla f(\theta_t)$$

$$\theta_{t+1} = \theta_t + (1+\beta) v_{t+1} - \beta v_t$$

<u>Vs. momentum:</u>

$$v_{t+1} = \beta v_t - \eta \nabla f(\theta_t)$$

$$\theta_{t+1} = \theta_t + v_{t+1}$$

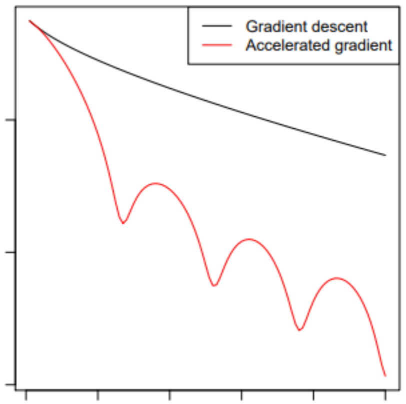
$$= \theta_t - \eta \nabla f(\theta_t) + \beta v_t$$

Looking forward

- **Nesterov's accelerated gradient:**

$$v_{t+1} = \beta v_t - \eta \nabla f(\theta_t)$$
$$\theta_{t+1} = \theta_t + (1+\beta) v_{t+1} - \beta v_t$$

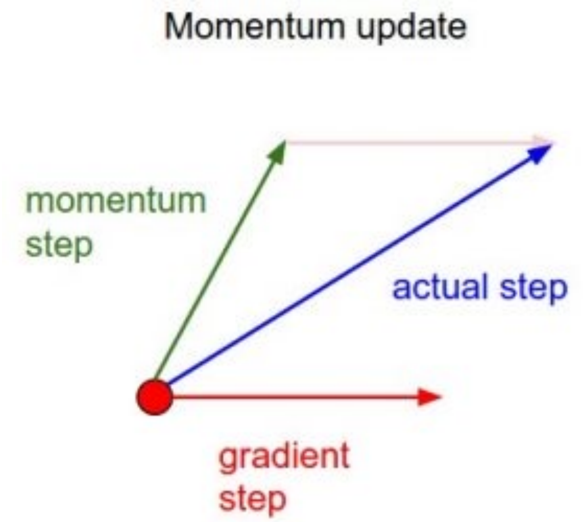
- This is not a descent method!



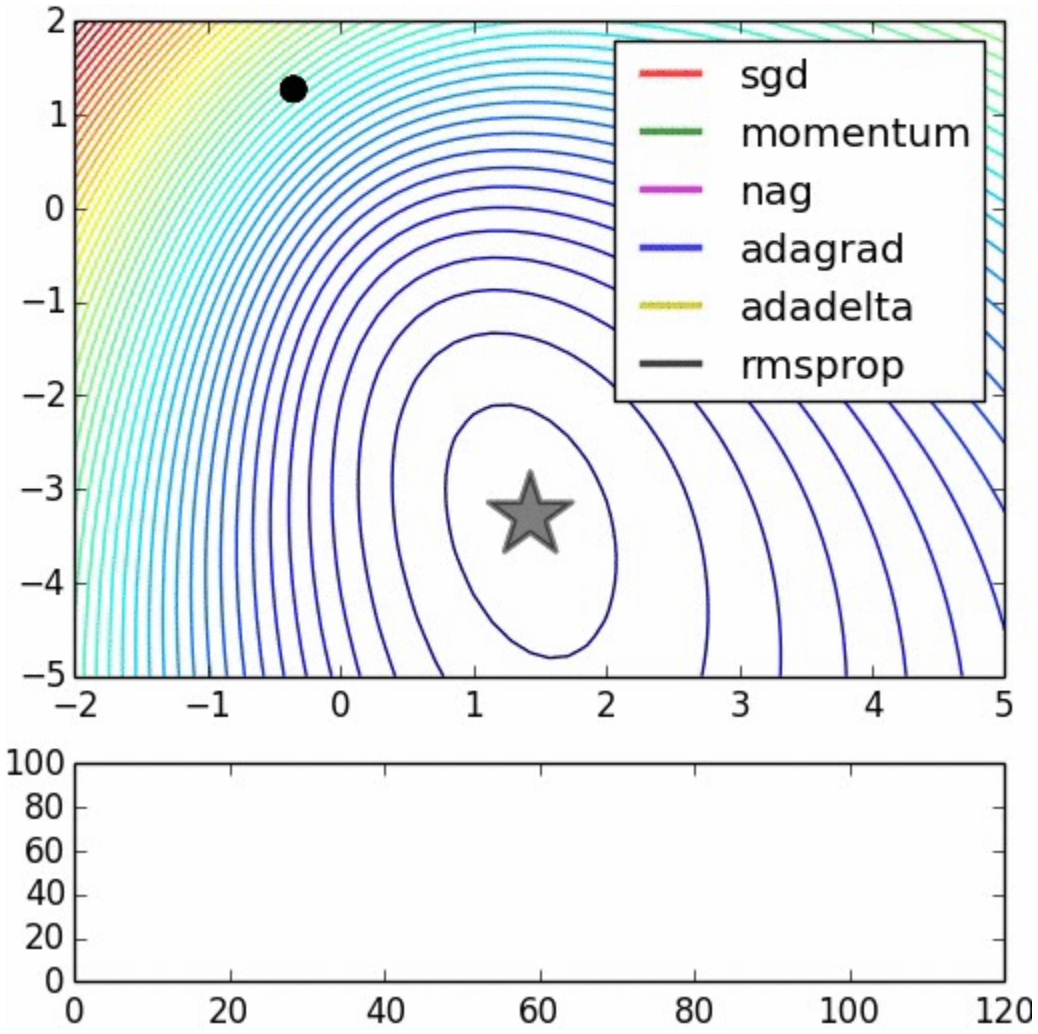
- Nonetheless, converges faster:

$$f(\theta_t) - f(\theta^*) \leq \frac{2\beta \|\theta_0 - \theta^*\|}{t^2} = O\left(\frac{\beta}{t^2}\right)$$

matches lower bound!



https://cs231n.github.io/neural-networks-3/



sgd
momentum
nag
adagrad
adadelta
rmsprop
2
1
0
−1
−2
−3
−4
−5
−2
−1
0
1
2
3
4
5
100
80
60
40
20
0
0
20
40
60
80
100
120

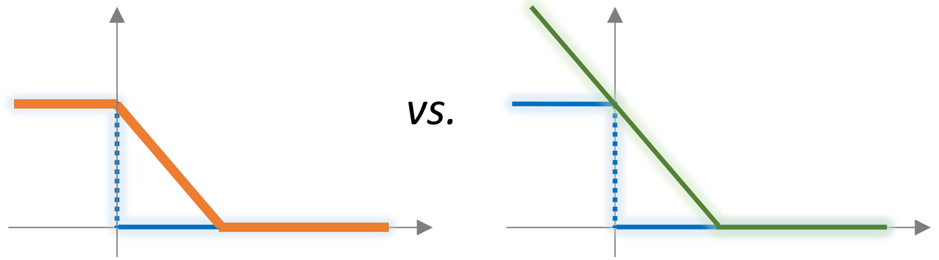
Beyond convexity

Back to modelling

- **Recall**:
  - Really want to optimize 0/1 loss
  - Instead optimize a continuous proxy
  - Proxies trade off in approximation vs. optimization
- GD provides strong guarantees for *convex* objectives
- But can still be applied to non-convex objectives!
  - ➢ non-convex losses
  - ➢ non-convex regularizers
  - ➢ non-convex predictive models

  (just need differentiability)
- Finds only <u>local</u> minimum, but many “tricks” for ending up at good local minimum

$$\underset{\theta \in \mathbb{R}^d}{\operatorname{argmin}} \frac{1}{m} \sum_{i=1}^{m} \overset{\textit{conv}}{\ell}\big(y_i, \overset{\textit{lin}}{f_\theta}(x_i)\big) + \overset{\textit{conv}}{\lambda R(\theta)}$$

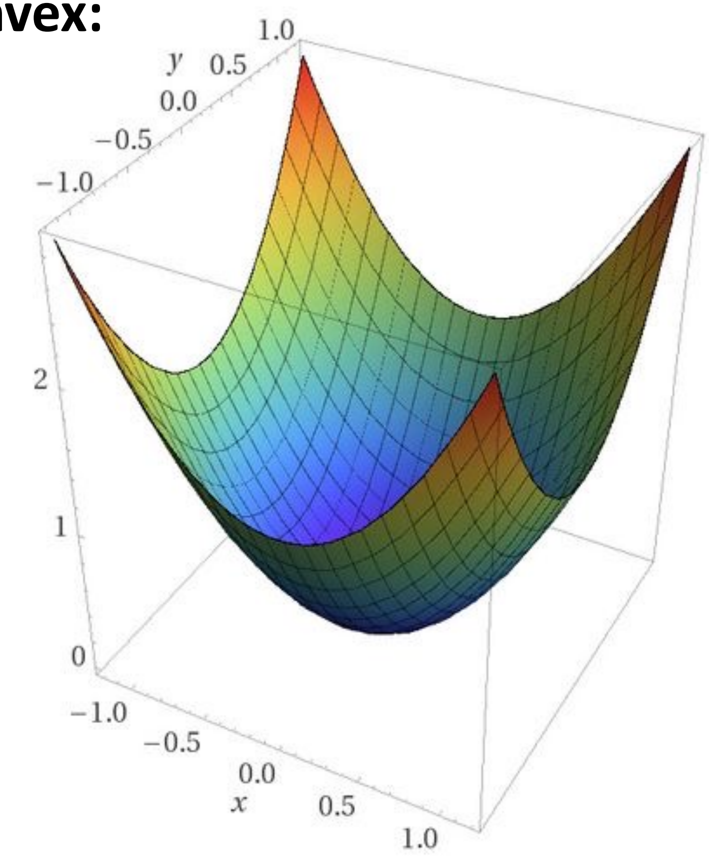


approximation (tighter)

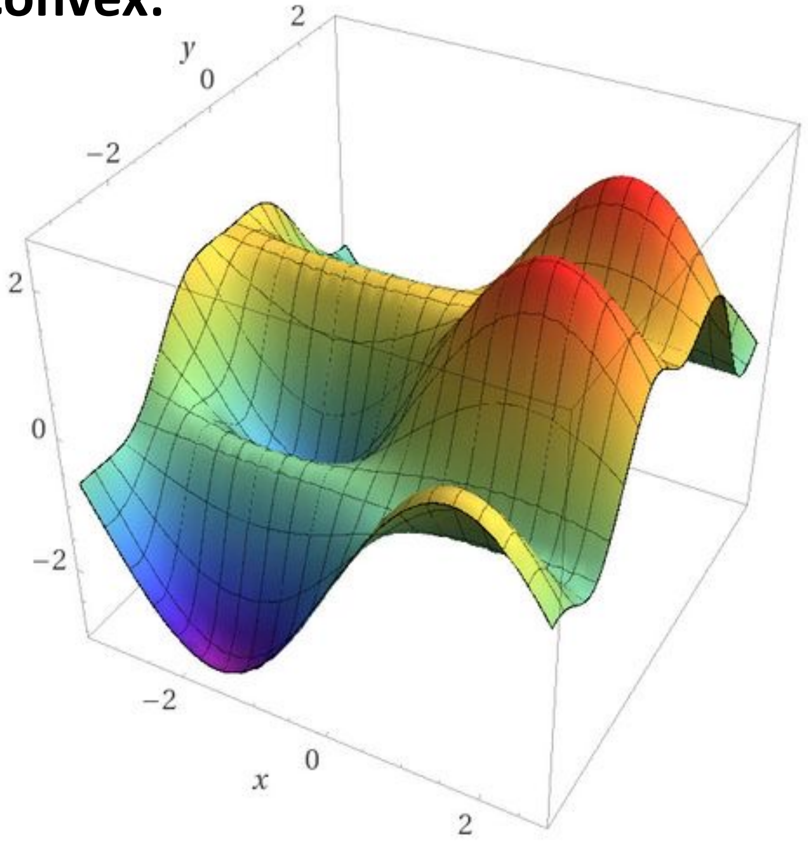
optimization (tractable)

Optimization landscape

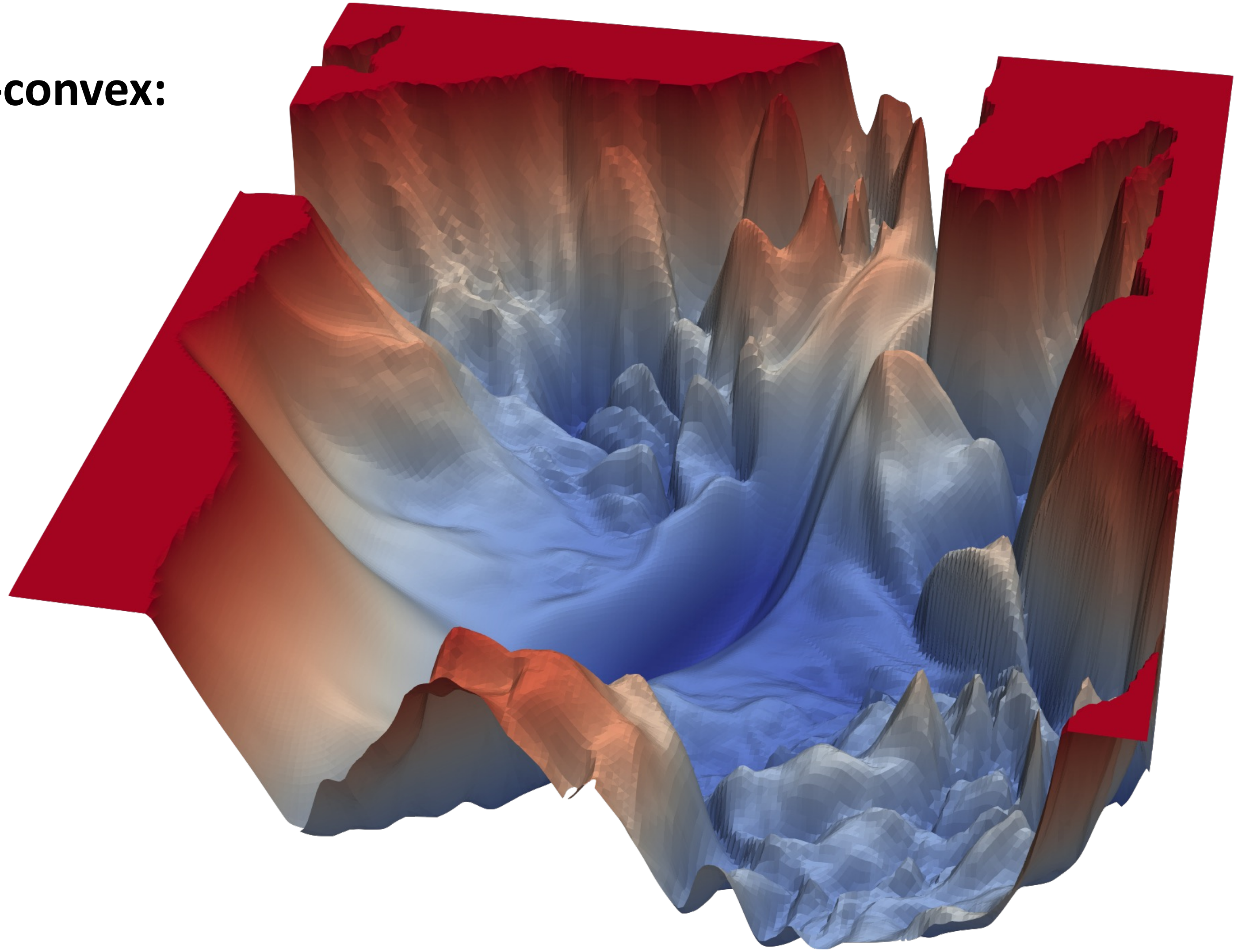
convex:

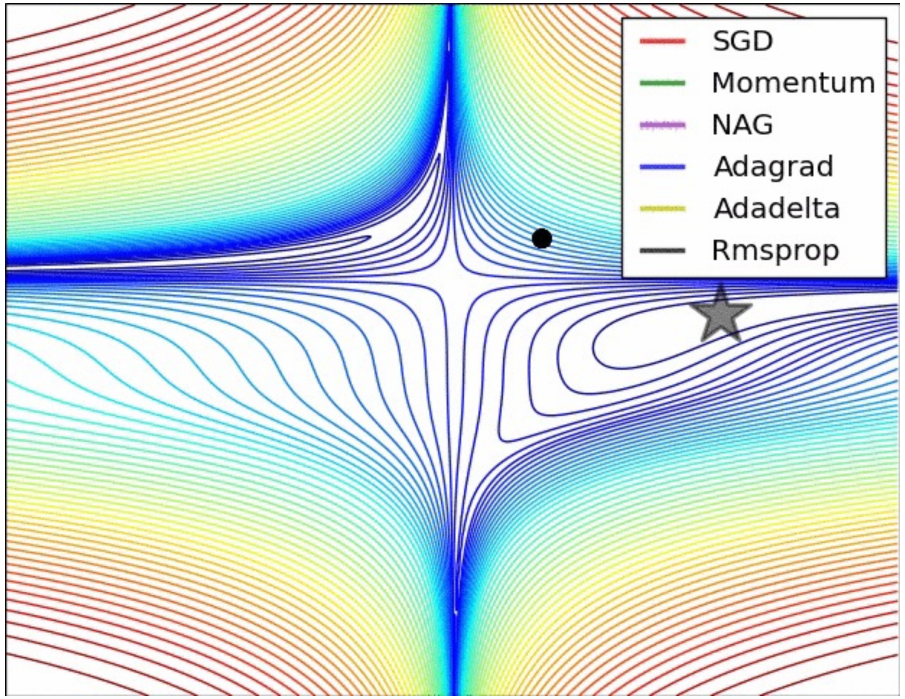


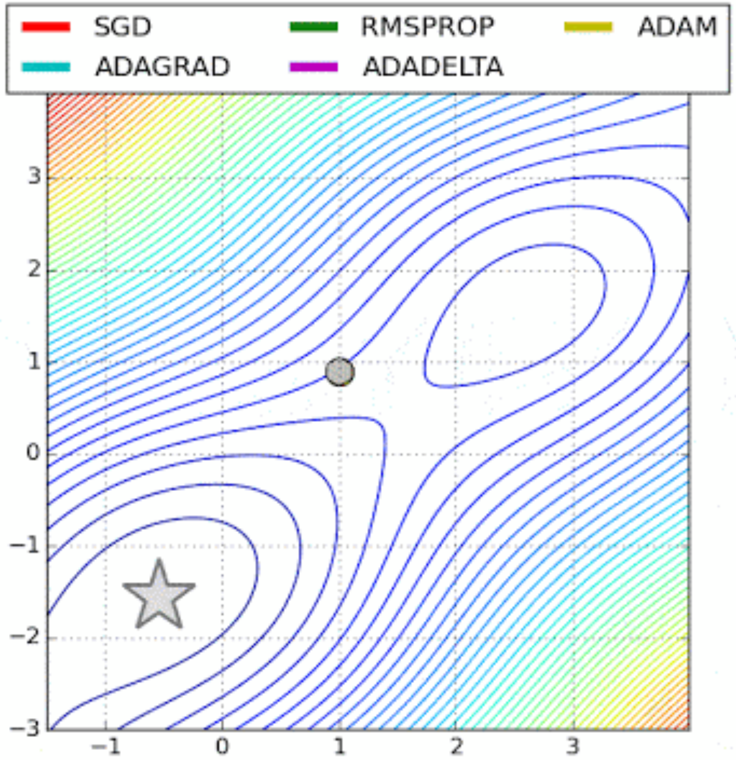
non-convex:



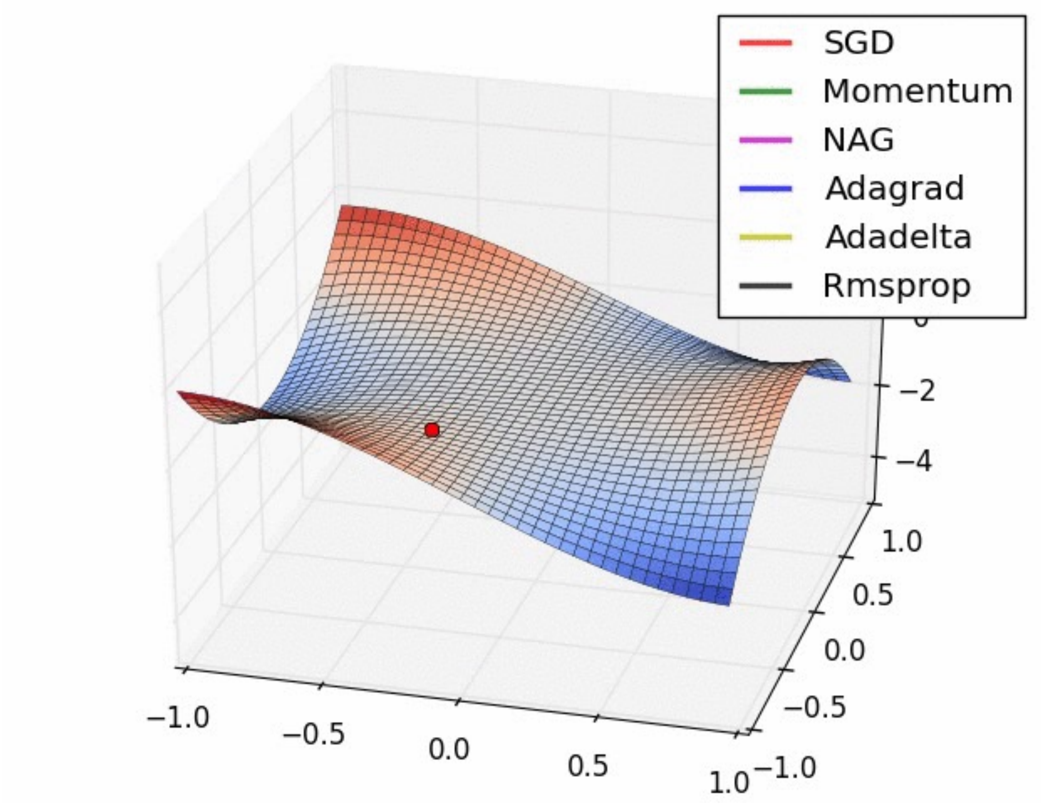
highly non-convex:

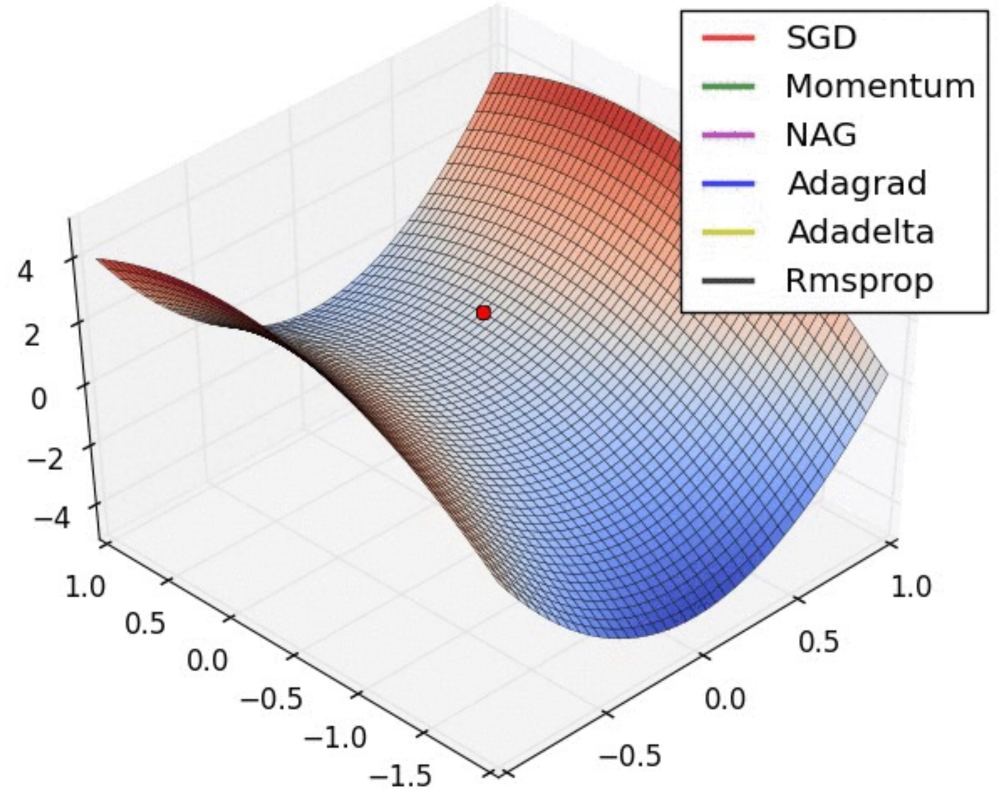






Can still apply gradient methods!





https://imgur.com/a/Hqolp

Automatic differentiation

- GD requires access to gradients
- **Stone age**: had to compute gradients by hand

$$P_k = \frac{e^{f_k}}{\sum_j e^{f_j}} \qquad L_i = -\sum_k p_{i,k} \log P_k \qquad f_m = (x_i W)_m$$

when $k = m$, $$\frac{\partial P_k}{\partial f_m} = \frac{e^{f_k}\sum_j e^{f_j} - e^{f_k}\cdot e^{f_k}}{(\sum_j e^{f_j})^2} = P_k(1-P_k)$$

when $k \neq m$, $$\frac{\partial P_k}{\partial f_m} = -\frac{e^{f_k}e^{f_m}}{(\sum_j e^{f_j})^2} = -P_k P_m$$

then:

$$\begin{aligned}
\frac{\partial L_i}{\partial f_m} &= -\sum_k p_{i,k}\frac{\partial \log P_k}{\partial f_m} \\
&= -\sum_k p_{i,k}\frac{1}{P_k}\frac{\partial P_k}{\partial f_m} \\
&= -\sum_{k=m} p_{i,k}\frac{1}{P_k}P_k(1-P_k) + \sum_{k\neq m} p_{i,k}\frac{1}{P_k}P_k P_m \\
&= \sum_{k\neq m} p_{i,k}P_m - \sum_{k=m} p_{i,k}(1-P_k) \\
&= \begin{cases} P_m & , \quad m \neq y_i \\ P_m - 1 & , \quad m = y_i \end{cases} \\
&= P_m - p_{i,m}
\end{aligned}$$

Last:

$$\frac{\partial L_i}{\partial W_k} = \frac{\partial L_i}{\partial f_m}\frac{\partial f_m}{\partial W_k} = x_i^T(P_m - p_{i,m})$$

$$\nabla_{W_k} L = -\frac{1}{N}\sum_i x_i^T(p_{i,m} - P_m) + 2\lambda W_k$$

Differentiating in practice

- GD requires access to gradients
- **Stone age**: had to compute gradients by hand
- **Modern age**: automatic differentiation (AutoDiff)

forward
```
(y, dy/dx) = foo(x)
```
backward

- Gradient computation completely abstracted away
- Building blocks + composition = **differentiable programs**
- We'll return to this when we discuss deep learning

$$P_k = \frac{e^{f_k}}{\sum_j e^{f_j}} \qquad L_i = -\sum_k p_{i,k} \log P_k \qquad f_m = (x_i W)_m$$

when $k = m$,
$$\frac{\partial P_k}{\partial f_m} = \frac{e^{f_k}\sum_j e^{f_j} - e^{f_k}\cdot e^{f_k}}{(\sum_j e^{f_j})^2} = P_k(1-P_k)$$

when $k \neq m$,
$$\frac{\partial P_k}{\partial f_m} = -\frac{e^{f_k} e^{f_m}}{(\sum_j e^{f_j})^2} = -P_k P_m$$

then:
$$\begin{aligned}\frac{\partial L_i}{\partial f_m} &= -\sum_k p_{i,k} \frac{\partial \log P_k}{\partial f_m} \\ &= -\sum_k p_{i,k} \frac{1}{P_k}\frac{\partial P_k}{\partial f_m} \\ &= -\sum_{k=m} p_{i,k}\frac{1}{P_k} P_k(1-P_k) + \sum_{k\neq m} p_{i,k}\frac{1}{P_k} P_k P_m \\ &= \sum_{k\neq m} p_{i,k} P_m - \sum_{k=m} p_{i,k}(1-P_k) \\ &= \begin{cases} P_m & , \quad m \neq y_i \\ P_m - 1 & , \quad m = y_i \end{cases} \\ &= P_m - p_{i,m}\end{aligned}$$

Last:
$$\frac{\partial L_i}{\partial W_k} = \frac{\partial L_i}{\partial f_m}\frac{\partial f_m}{\partial W_k} = x_i^T (P_m - p_{i,m})$$

$$\nabla_{W_k} L = -\frac{1}{N}\sum_i x_i^T (p_{i,m} - P_m) + 2\lambda W_k$$

Up next

- **Part II**: *the different aspects of learning*
  1. Statistics: generalization and PAC theory
  2. Modeling:
     - Error decomposition
     - Regularization
     - Model selection
  3. Optimization: convexity, gradient descent
  4. Practical aspects and potential pitfalls

Perceptron

input: `sample set` $S = \{(x_i, y_i)\}_{i=1}^{m}$

algorithm:

- `initialize` $w_0 = \vec{0}$
- `for` $t = 1,2, \dots$
  - `if` $\exists i \in [m]$ `s.t.` $y_i w_t^\top x_i \le 0$ `#wrong classification`
    - $w_{t+1} = w_t + y_i x_i$
  - `else`
    - `return` w_t