Is numerical optimization theory irrelevant to machine learning practice in 2026?

Mark Schmidt
Department of Computer Science
University of British Columbia
201 2366 Main Mall
Vancouver BC V6T 1Z4
Canada
schmidtm@cs.ubc.ca

A Great Loss for Numerical Optimization Theory

- Last month we lost Dimitri Bertsekas.

- Wrote influential books in several different fields.
- Pioneer in numerical optimization (memories here).

Bertsekas' Two Cultures of Numerical Optimization

- Bertsekas contrasted two views on optimization theory:
  - Theoretical: "convergence analysis is considered primarily a mathematical subject ... General and broadly applicable statements, and simple and elegant proofs are at premium here."
  - Practical "properties of an algorithm for a given class of problems must be verified through practical experimentation, so if an algorithm looks promising on intuitive grounds, why bother with a convergence analysis?"

[Bertsekas, 2016]

My Previous Lifetimes

- I started on practical view:
  - ICML 2006 paper: “Maybe we should use SGD to train language models?”.
    - Trained ~10-million parameters on ~30k sentences (“large” model at time).
  - 2005 minFunc code: tuned to solve ML problems at time.

- But later worked on theoretical view:
  - NeurIPS 2011: “Accelerated gradient methods with inexact gradients”.
  - NeurIPS 2012: “SGD method with exponential convergence rate”.

- And in 2015 I thought these views were coming together!
  - SGD methods with nice theory that were achieving state of the art results.
  - Gave ICML 2015 tutorial mainly focusing theory-motivated fast algorithms.

- In 2015 I also wrote a review that started like this:

> This work proposes a new stochastic gradient method, in the spirit of AdaGrad and natural-gradient methods, that uses a per-variable scaling in the update which is based on the ratio of an estimate of the mean and the variance of the gradients.
>
> The approach is well-motivated, and I like the connection established to RPROP. It would be good to allow state the connection to the Le Roux and Fitzgibbon's NIPS 2010 paper 'A fast natural Newton method' (which also considers variance of hte gradients).
>
> Unfortunately the proofs of the regret bounds are missing. Further, they don't seem to actually be related to the stated Algorithm 1. In particular, \gamma_t seems to be a decreasing step-size in the analysis but there is \gamma_t and a constant step-size is used in Algorithm 1 and the subsequent analysis.

- "Adam" method worked dramatically better for neural networks.
  - As they were taking over the ML world.

Adam: A method for **stochastic** optimization
DP Kingma, J Ba - arXiv preprint arXiv:1412.6980, 2014 - arxiv.org
... functions with **stochastic** regularization. We compare the effectiveness of **Adam** to other **stochastic** first ... Figure 2 shows our results; **Adam** shows better convergence than other methods. ...
☆ Save 99 Cite Cited by 246918 Related articles All 14 versions »

- Since then gap between two cultures has become enormous!

- Have seen amazing results in "convergence analysis":
  - Optimal multi-step algorithms via PEP.
  - Parameter-free SGD via coin-betting.
  - Non-asymptotic analysis of BFGS.
- But these ideas have had little effect on practice.

- While the "proof is missing" works are changing the world:
  - Adam optimizer (course project at University of Toronto).
  - Cosine annealing (circumnutations in plant roots).
  - AdamW (trying to fix regularization empirically hurting late in training).
  - Muon (NanoGPT speedrunning and Twitter crowdsourcing).
- Though huge literature where it "works in my paper".

[Drori & Teboulle, 2014, King & Ba, 2014, Loschilov & Hutter, 2017, Orabona & Tommasi, 2017, Loschilov & Hutter, 2019, Jordan et al., 2024, Qin et al., 2025]

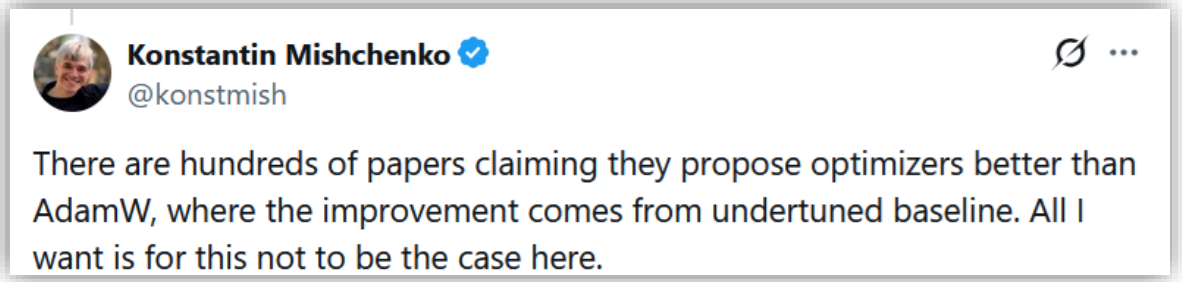
Mess of Attempts to Improve Adam (Mostly Failed)

A. List of optimizers and schedules considered

Table 2: List of optimizers considered for our benchmark. This is only a subset of all existing methods for deep learning.

Name	Ref.	Name	Ref.
AcceleGrad	(Levy et al., 2018)	HyperAdam	(Wang et al., 2019b)
ACClip	(Zhang et al., 2020)	K-BFGS/K-BFGS(L)	(Goldfarb et al., 2020)
AdaAlter	(Xie et al., 2019)	KF-QN-CNN	(Ren & Goldfarb, 2021)
AdaBatch	(Devarakonda et al., 2017)	KFAC	(Martens & Grosse, 2015)
AdaBayes/AdaBayes-SS	(Aitchison, 2020)	KFLR/KFRA	(Botev et al., 2017)
AdaBelief	(Zhuang et al., 2020)	L4Adam/L4Momentum	(Rolínek & Martius, 2018)
AdaBlock	(Yun et al., 2019)	LAMB	(You et al., 2020)
AdaBound	(Luo et al., 2019)	LaProp	(Ziyin et al., 2020)
AdaComp	(Chen et al., 2018)	LARS	(You et al., 2017)
Adadelta	(Zeiler, 2012)	LHOPT	(Almeida et al., 2021)
Adafactor	(Shazeer & Stern, 2018)	LookAhead	(Zhang et al., 2019)
AdaFix	(Bae et al., 2019)	M-SVAG	(Balles & Hennig, 2018)
AdaFom	(Chen et al., 2019a)	MADGRAD	(Defazio & Jelassi, 2021)
AdaFTRL	(Orabona & Pál, 2015)	MAS	(Landro et al., 2020)
Adagrad	(Duchi et al., 2011)	MEKA	(Chen et al., 2020b)
ADAHESSIAN	(Yao et al., 2020)	MTAdam	(Malkiel & Wolf, 2020)
Adai	(Xie et al., 2020)	MVRC-1/MVRC-2	(Chen & Zhou, 2020)
AdaLoss	(Teixeira et al., 2019)	Nadam	(Dozat, 2016)
Adam	(Kingma & Ba, 2015)	NAMSB/NAMSG	(Chen et al., 2019b)
Adam$^+$	(Liu et al., 2020b)	ND-Adam	(Zhang et al., 2017a)
AdamAL	(Tao et al., 2019)	Nero	(Liu et al., 2021b)
AdaMax	(Kingma & Ba, 2015)	Nesterov	(Nesterov, 1983)
AdamBS	(Liu et al., 2020c)	Noisy Adam/Noisy K-FAC	(Zhang et al., 2018)
AdamNC	(Reddi et al., 2018)	NosAdam	(Huang et al., 2019)
AdaMod	(Ding et al., 2019)	Novograd	(Ginsburg et al., 2019)
AdamP/SGDP	(Heo et al., 2021)	NT-SGD	(Zhou et al., 2021b)
AdamT	(Zhou et al., 2020)	Padam	(Chen et al., 2020a)
AdamW	(Loshchilov & Hutter, 2019)	PAGE	(Li et al., 2020b)
AdamX	(Tran & Phong, 2019)	PAL	(Mutschler & Zell, 2020)
ADAS	(Eliyahu, 2020)	PolyAdam	(Orvieto et al., 2019)
AdaS	(Hosseini & Plataniotis, 2020)	Polyak	(Polyak, 1964)
AdaScale	(Johnson et al., 2020)	PowerSGD/PowerSGDM	(Vogels et al., 2019)
AdaSGD	(Wang & Wiens, 2020)	Probabilistic Polyak	(de Roos et al., 2021)
AdaShift	(Zhou et al., 2019)	ProbLS	(Mahsereci & Hennig, 2017)
AdaSqrt	(Hu et al., 2019)	PStorm	(Xu, 2020)
Adathm	(Sun et al., 2019)	QHAdam/QHM	(Ma & Yarats, 2019)
AdaX/AdaX-W	(Li et al., 2020a)	RAdam	(Liu et al., 2020a)
AEGD	(Liu & Tian, 2020)	Ranger	(Wright, 2020b)
ALI-G	(Berrada et al., 2020)	RangerLars	(Grankin, 2020)
AMSBound	(Luo et al., 2019)	RMSProp	(Tieleman & Hinton, 2012)
AMSGrad	(Reddi et al., 2018)	RMSterov	(Choi et al., 2019)
AngularGrad	(Roy et al., 2021)	S-SGD	(Sung et al., 2020)
ArmijoLS	(Vaswani et al., 2019)	SAdam	(Wang et al., 2020b)
ARSG	(Chen et al., 2019b)	Sadam/SAMSGrad	(Tong et al., 2019)
ASAM	(Kwon et al., 2021)	SALR	(Yue et al., 2020)
AutoLRS	(Jin et al., 2021)	SAM	(Foret et al., 2021)
AvaGrad	(Savarese et al., 2019)	SC-Adagrad/SC-RMSProp	(Mukkamala & Hein, 2017)
BAdam	(Salas et al., 2018)	SDProp	(Ida et al., 2017)
BGAdam	(Bai & Zhang, 2019)	SGD	(Robbins & Monro, 1951)
BPGrad	(Zhang et al., 2017b)	SGD-BB	(Tan et al., 2016)
BRMSProp	(Aitchison, 2020)	SGD-G2	(Ayadi & Turinici, 2020)
BSGD	(Hu et al., 2020)	SGDEM	(Ramezani-Kebrya et al., 2021)
C-ADAM	(Tutunov et al., 2020)	SGDHess	(Tran & Cutkosky, 2021)
CADA	(Chen et al., 2021)	SGDM	(Liu & Luo, 2020)
Cool Momentum	(Borysenko & Byshkin, 2020)	SGDR	(Loshchilov & Hutter, 2017)
CProp	(Preechakul & Kijsirikul, 2019)	SHAdagrad	(Huang et al., 2020)
Curveball	(Henriques et al., 2019)	Shampoo	(Anil et al., 2020; Gupta et al., 2018)
Dadam	(Nazari et al., 2019)	SignAdam++	(Wang et al., 2019a)
DeepMemory	(Wright, 2020a)	SignSGD	(Bernstein et al., 2018)
DGNOpt	(Liu et al., 2021a)	SKQN/S4QN	(Yang et al., 2020)
DiffGrad	(Dubey et al., 2020)	SM3	(Anil et al., 2019)
EAdam	(Yuan & Gao, 2020)	SMG	(Tran et al., 2020)
EKFAC	(George et al., 2018)	SNGM	(Zhao et al., 2020)
Eve	(Hayashi et al., 2018)	SoftAdam	(Fetterman et al., 2019)
Expectigrad	(Daley & Amato, 2020)	SRSGD	(Wang et al., 2020a)
FastAdaBelief	(Zhou et al., 2021a)	Step-Tuned SGD	(Castera et al., 2021)
FRSGD	(Wang & Ye, 2020)	SWATS	(Keskar & Socher, 2017)
G-AdaGrad	(Chakrabarti & Chopra, 2021)	SWNTS	(Chen et al., 2019c)
GADAM	(Zhang & Gouza, 2018)	TAdam	(Ilboudo et al., 2020)
Gadam	(Granziol et al., 2020)	TEKFAC	(Gao et al., 2020)
GOALS	(Chae et al., 2021)	VAdam	(Khan et al., 2018)
GOLS-I	(Kafka & Wilke, 2019)	VR-SGD	(Shang et al., 2020)
Grad-Avg	(Purkayastha & Purkayastha, 2020)	vSGD-b/vSGD-g/vSGD-l	(Schaul et al., 2013)
GRAPES	(Dellaferrera et al., 2021)	vSGD-fd	(Schaul & LeCun, 2013)
Gravilon	(Kelterborn et al., 2020)	WNGrad	(Wu et al., 2018)
Gravity	(Bahrami & Zadeh, 2021)	YellowFin	(Zhang & Mitliagkas, 2019)
HAdam	(Jiang et al., 2019)	Yogi	(Zaheer et al., 2018)

AdaTnm
Eve
AAdam
BAdam
CAdam
DAdam
EAdam
FAdam
GAdam
HAdam
IAdam
KAdam
LAdam
MAdam
NAdam
OAdam
PAdam



AdamW paper was rejected 3 times.

Final meta-review: “Previous versions of the paper suffered from a lack of theoretical justification...AdamW has been validated by so many other groups on a range of domains that the improvement is well-established. And other researchers have offered possible explanations for the improvement.”

[Schmidt et al., 2021]

Empirical Progress Seems Slow/Stalled

- 2020 “Benchmarking Deep Learning Optimizers” [2020].
  - Newer methods fail significantly and consistently in outperforming Adam.
- 202x Major LLM release delayed because over trying new method.
- 2024 Algoperf self-tuning track:
  - Only 1 method beat “AdamW” baseline method.
    - It was AdamW with different step sizes (ScheduleFree).
      - Shampoo variant won other track (but heavy task-specific hyper-parameter tuning).
- 2025 “Fantastic Pretraining Optimizers and Where to Find Them”.
  - Speedup over baselines is lower than claimed, decreases with model size.
- 2025 “Smol Training Playbook”:
  - Tried newer methods but went back to AdamW.

Yaroslav Bulatov @yaroslavvb · Dec 11, 2025

20 years of reading and implementing optimization papers, yet after this year's NeurIPS, I'm still questioning the utility of the field. The few methods that work seem to do so by accident.

18 22 322 63K

Is optimization theory useless for practitioners?

- Ben Recht:
  - “I'm skeptical that optimization theory helps us do anything”
- Marta Skreta:
  - “I have so many other things to worry about from datasets, augmentation, architecture, hyperparameters before I think about the optimizer”
- Jason Lee:
  - “encourage people to understand algorithms and optimization in DL, rather than proposing new optimizers”
- James Martens:
  - “I think optimization theory is still a bit relevant, but not overly so. IMO, the most important thing is the ‘gradient-dominated vs. noise-dominated regime’ distinction”
- Tasos Kyrillidis:
  - “I like the gradient-dominated vs. noise-dominated axis … almost every modern trick … sits naturally there”
- Martin Jaggi:
  - “this year is kind of a pleasant surprise at least for me that many 'classic' optimization questions have moved front and center again”

My Goals

- Show that optimization theory helps understand common questions.
  - "How do I set the step size?"
    - Gradient-dominated vs. noise-dominated phases.
  - "How do I pick the batch size?"
  - "Why does Adam work?"
  - "Why isn't second-order information helping?"
  - "Why does [weird neural network trick] work?"

- For practitioners:
  - Show you simple concepts that explain a lot of what we see in practice.
  - Hopefully let you skip some generations of grad-student descent.

- For theoreticians:
  - Highlight where we are focusing on the wrong things.
  - Give ideas on how we can become more useful in practice.

Disclaimer Page #1

- Answer to these questions are complicated:
  - Depend on many factors and dynamics that are not fully understood.
  - But theory at least highlights important trade-offs.

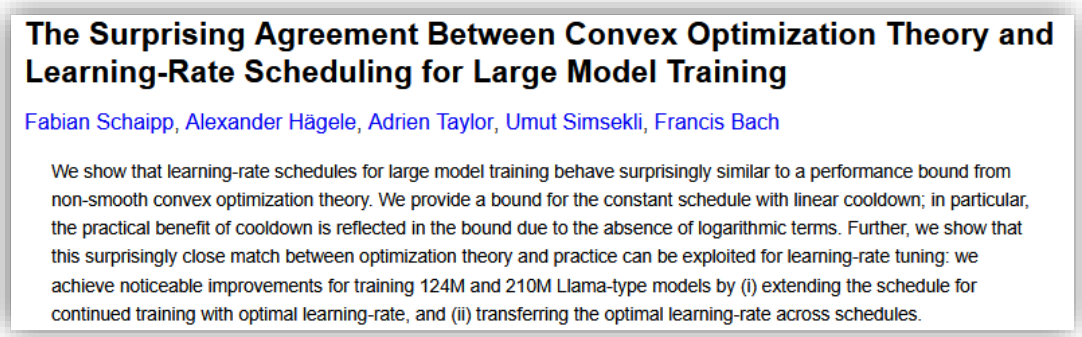
- Field is huge and changing and noisy.
  - I will inevitably miss big/important topics.
  - I will necessarily need to skip over a lot of precise details.
  - And long term I will be wrong about some things.

- Topics I will not cover to reduce scope:
  - Private, fair, federated, constrained, parallel, distributed, low-precision, continual, PEFT, RL, tuning hyper-parameters, scaling laws, DE and SE models.
    - Though fundamentals still give insights in these settings.

Disclaimer Page #2

- I am going to focus on smooth optimization.
  - Assume the function is differentiable everywhere (no ReLUs).

- Why?
  - Many conclusions are similar.
    - Especially for stochastic optimization.
  - Many analyses are simpler.
    - Especially for non-convex optimization.
  - There is more to gain performance-wise with smoothness.
  - Smooth case is already quite complicated.
  - There is some trend towards smooth functions.
    - LLMs have moved form ReLUs to GeLUs or SwiGLUs.
    - We train non-smooth mixture of experts via smooth objectives.



The Surprising Agreement Between Convex Optimization Theory and Learning-Rate Scheduling for Large Model Training

Fabian Schaipp, Alexander Hägele, Adrien Taylor, Umut Simsekli, Francis Bach

We show that learning-rate schedules for large model training behave surprisingly similar to a performance bound from non-smooth convex optimization theory. We provide a bound for the constant schedule with linear cooldown; in particular, the practical benefit of cooldown is reflected in the bound due to the absence of logarithmic terms. Further, we show that this surprisingly close match between optimization theory and practice can be exploited for learning-rate tuning: we achieve noticeable improvements for training 124M and 210M Llama-type models by (i) extending the schedule for continued training with optimal learning-rate, and (ii) transferring the optimal learning-rate across schedules.

Question: How do I set the GD Step Size?

(We are going to spend a lot of time on this)

(But often the most sensitive hyper-parameter!)

Introduction: Optimization and Gradient Descent

- We usually "train" models by solving an optimization problem.

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

  - We have d parameters w that we can modify the fit the data better.
    - For a deep neural network, this would contain the parameters of the various layers.
  - We have a function f that measures how well we fit the data.
    - Usually this function is the average over a set of n functions f_i.
      - Each f_i measures how well we fit training example i (squared error, cross entropy, and so on).

- Most popular DL algorithm is stochastic gradient descent (SGD).
  - And its descendants like Adam, AdamW, and so on.

Deterministic Optimization with Fixed Features

- But we will first focus on deterministic gradient descent (GD).
  - We can compute the function $f(x)$ and gradient $\nabla f(w)$ exactly.
    - "Full-batch" training (expensive for large datasets).
  - To make some things simpler we will assume f is twice-differentiable.
- We will also first focus on the case of fixed features.
  - Canonical examples are linear least squares and logistic regression:

$$f(w) = \frac{1}{2}\sum_{i=1}^{n}(w^T x_i - y_i)^2. \qquad f(w) = \sum_{i=1}^{n}\log(1+\exp(-y_i w^T x_i))$$

  - Or training the last layer of a neural network with other layers fixed.

Why talk about deterministic/fixed setting at all?

- Ideas form most ingredients for stochastic/neural optimization.
- Many concepts are already surprisingly complicated.
  - Such as choosing the step size.
- Many "stochastic/neural" challenges already appear in simpler setting.
  - Such as WarmUp/clipping, and oscillations that may or may not be good.
- We will see that deterministic tricks often help at the start of training.
  - In gradient-dominated phase, or with large batch sizes.
- Deterministic optimizers are robust.
  - Have hyper-parameters, but often work well with no hyper-parameter tuning.

Gradient Descent (GD)

- Classic gradient descent (GD) algorithm repeats the iteration:

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

  - For a step size α_k.
- Since $f(w)$ is an average over n training examples, takes the form

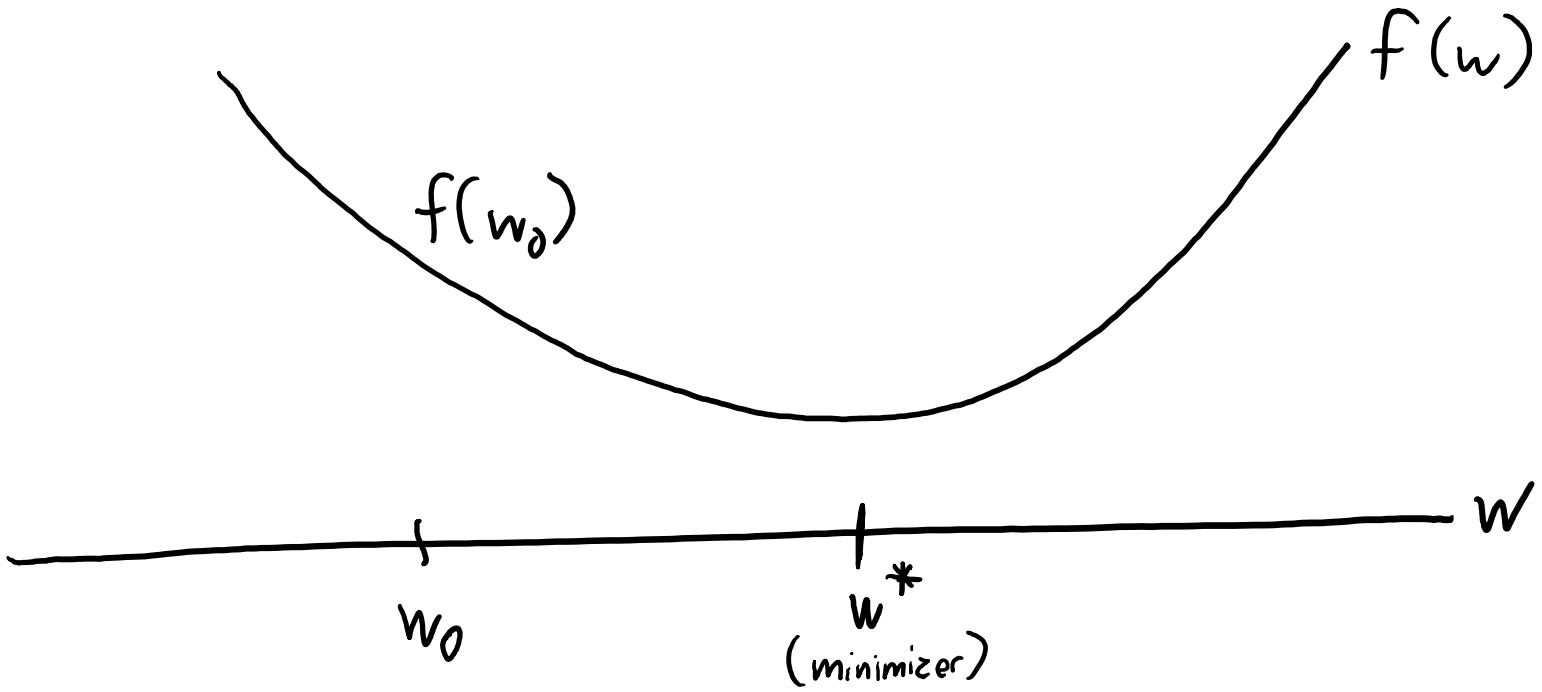
$$w_{k+1} = w_k - \alpha_k \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(w_k).$$

"full batch"

- First paper by Cauchy [1847], who gave two step size suggestions:
  - Choose a small step size.
  - Find the minimum along the direction (greedy).

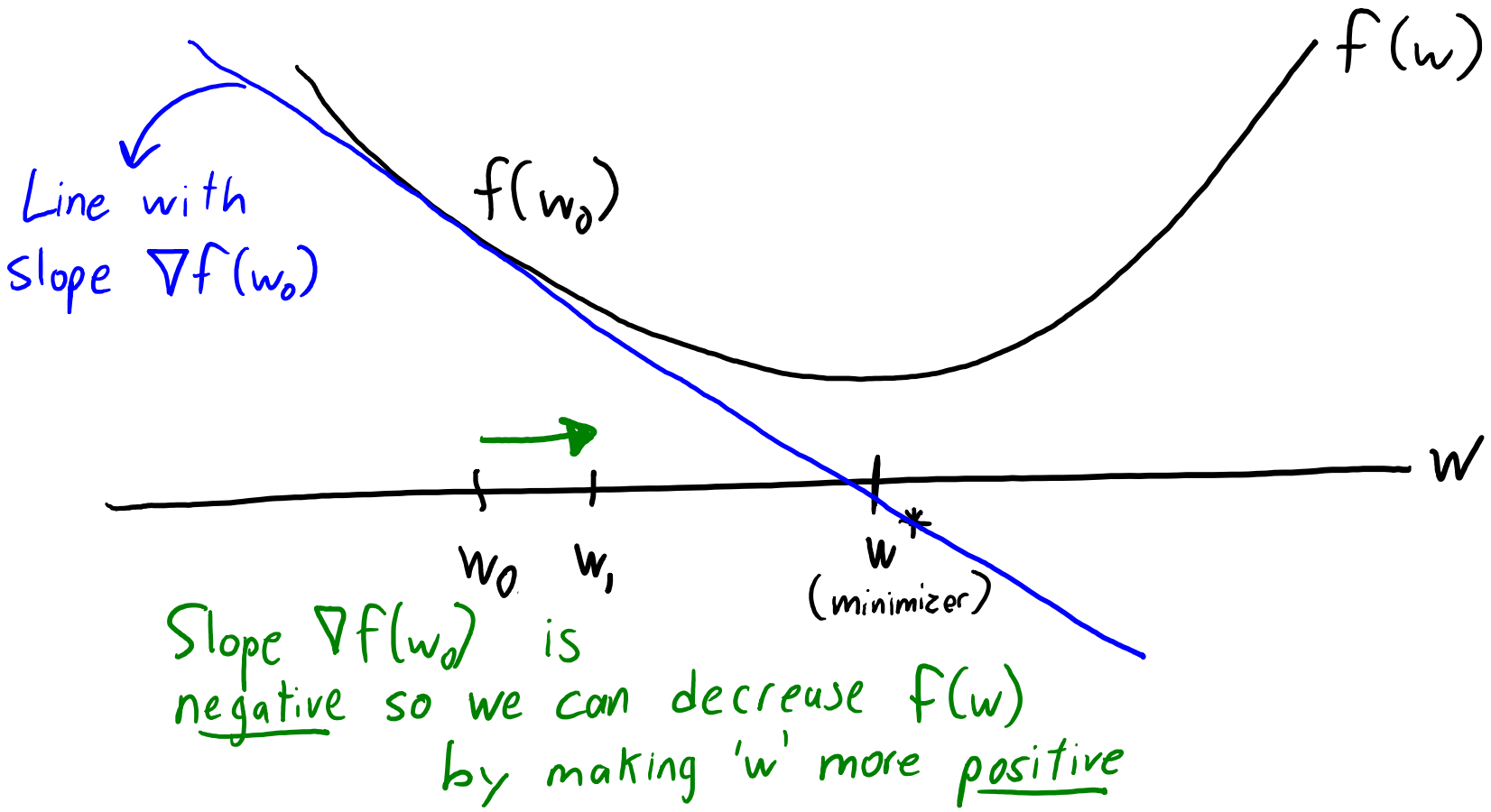
Deterministic Gradient Descent in 1 Dimension

- Deterministic gradient descent is based on a simple observation:
  - Give parameters w, the direction of largest decrease is $-\nabla f(w)$.



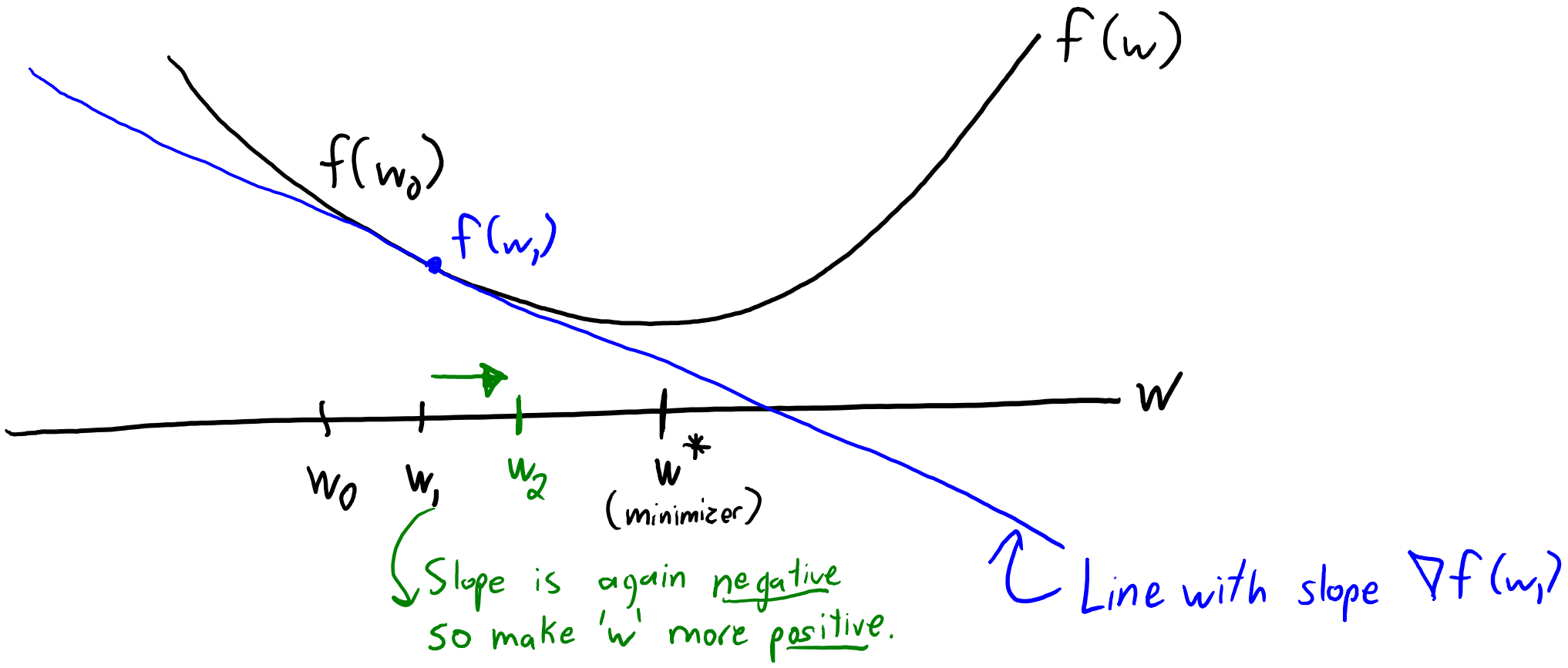
Deterministic Gradient Descent in 1 Dimension

- Deterministic gradient descent is based on a simple observation:
  - Give parameters w, the direction of largest decrease is $-\nabla f(w)$.



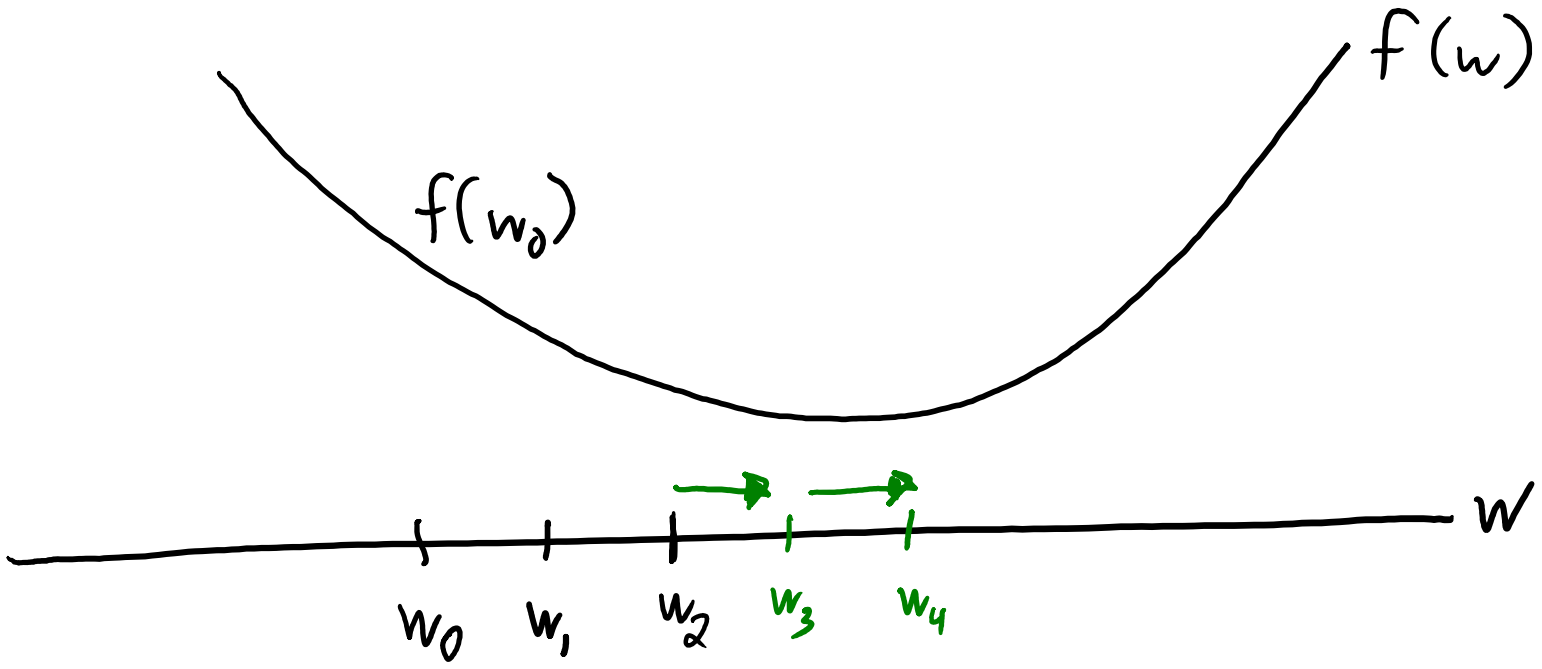
Deterministic Gradient Descent in 1 Dimension

- Deterministic gradient descent is based on a simple observation:
  - Give parameters w, the direction of largest decrease is $-\nabla f(w)$.



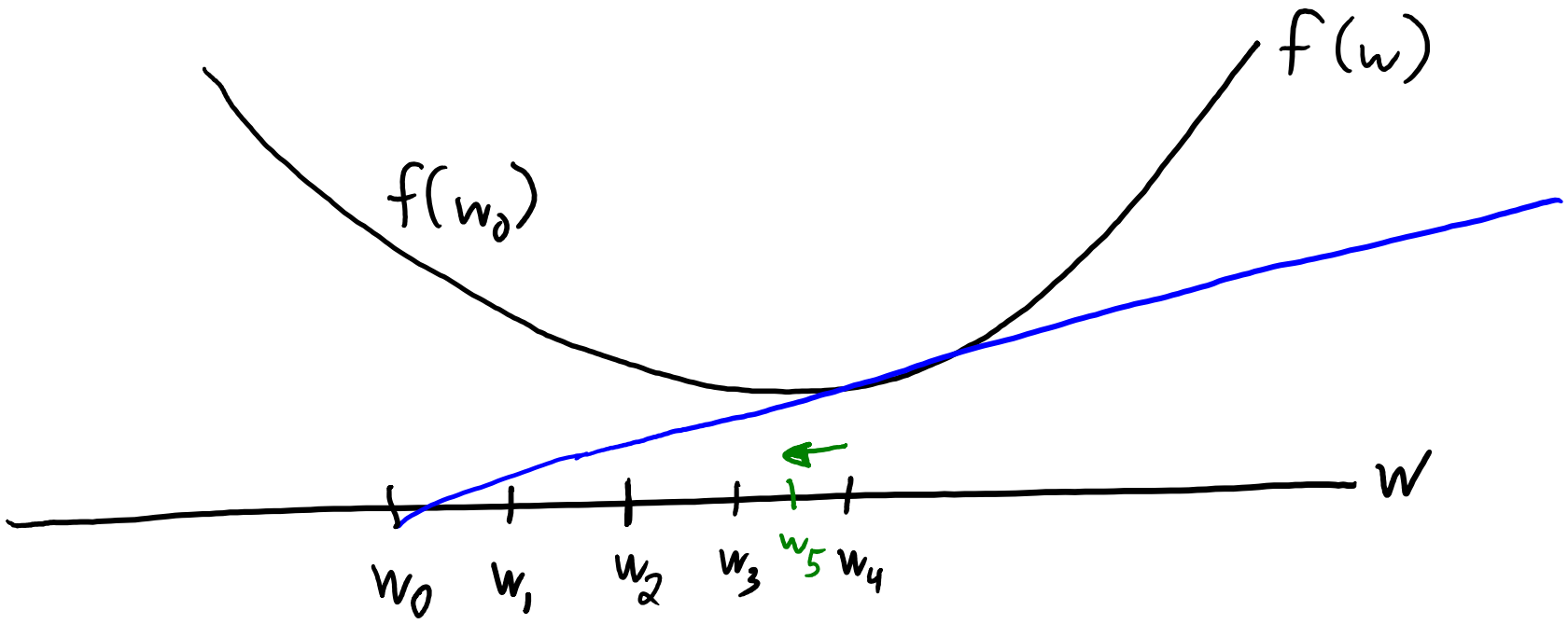
Deterministic Gradient Descent in 1 Dimension

- Deterministic gradient descent is based on a simple observation:
  - Give parameters w, the direction of largest decrease is $-\nabla f(w)$.



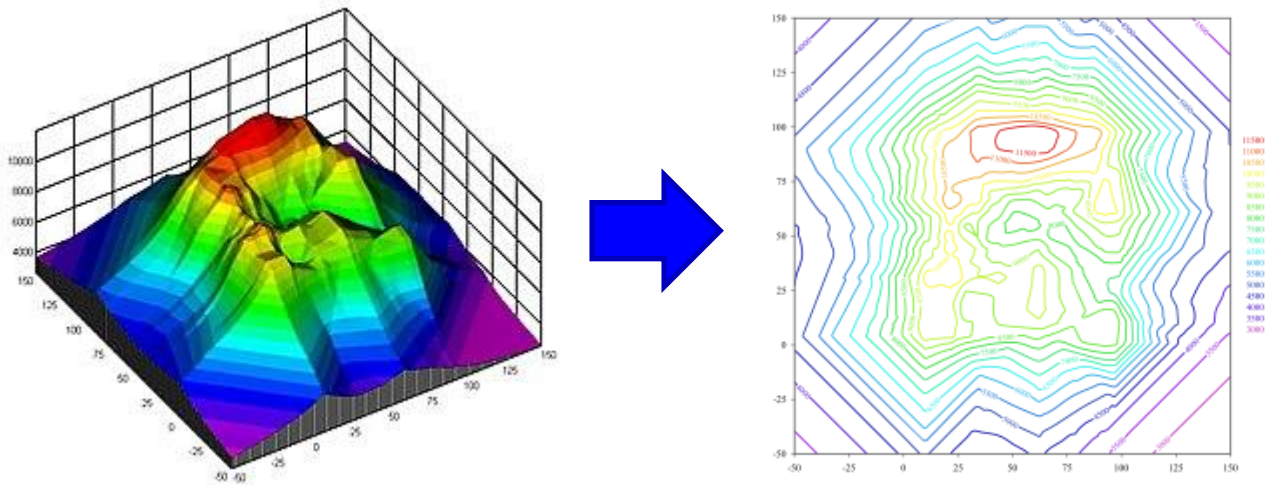
Deterministic Gradient Descent in 1 Dimension

- Deterministic gradient descent is based on a simple observation:
  - Give parameters w, the direction of largest decrease is $-\nabla f(w)$.



Digression: Contour Maps

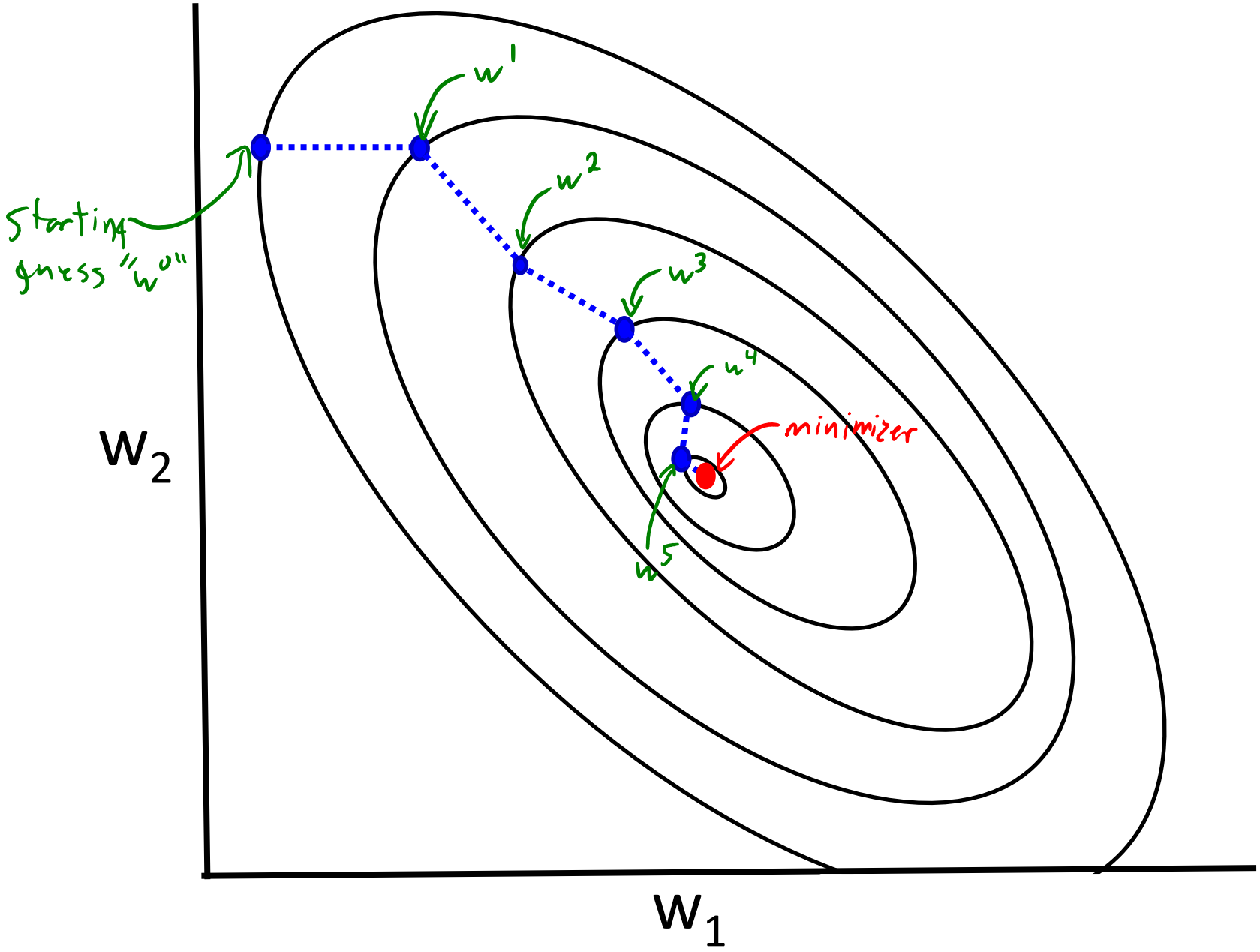
- To visualize gradient descent when d=2, we often use contour lines:
  - Figure where each curve has same function value.



  - Like a hiking elevation map:

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

Gradient Descent in 2D



How do I set the GD step size?

- For gradient descent, there is a trade-off between:

α_k "step size" vs. $\frac{2}{L}$ "Lipschitz constant of gradient"

- The number L is how fast the gradient can change.
  - The "Lipschitz constant": $\|\nabla f(w) - \nabla f(v)\| \leq L\|w - v\|$

  - Need smaller steps for bigger 'L'.

How do I set the GD step size?

- For gradient descent, there is a trade-off between:

α_k vs. $\frac{2}{L}$

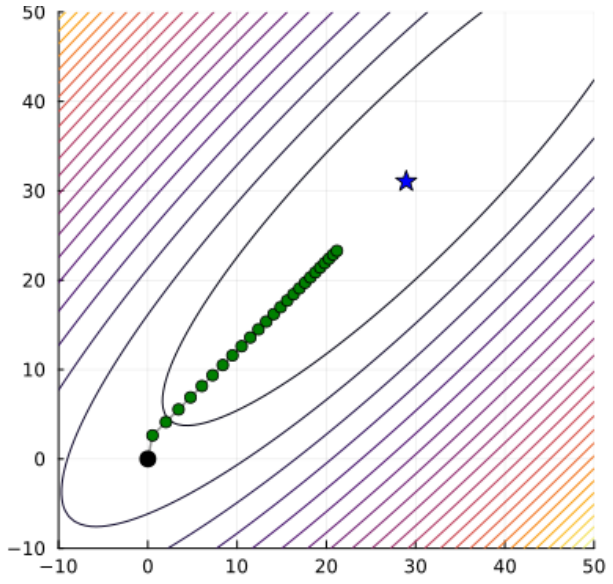
"step size" "Lipschitz constant of gradient"

- If $\alpha_k < 2/L$, GD is guaranteed to decrease the function 'f'.
- If α_k is much smaller than 2/L, GD will converge more slowly.
  - Step size is too small.
- If α_k is bigger than 2/L, you might increase the function.
  - Step could be too big and "overshoot" region where function decreases.
  - But you might get lucky and decrease the function a lot.

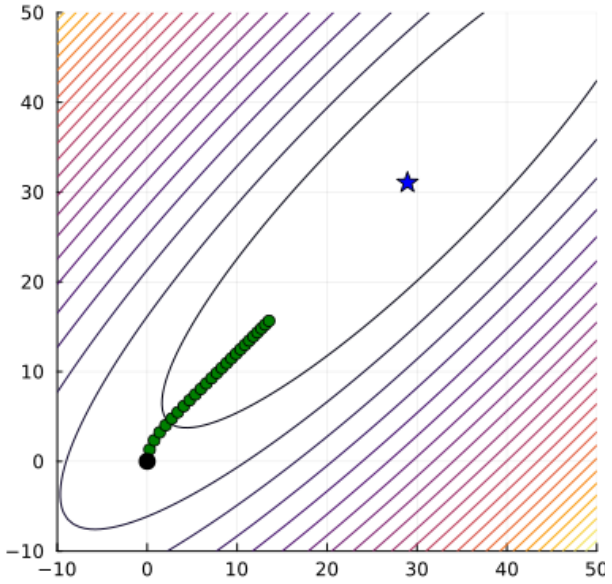
Comparison of Fixed Step Sizes for GD

- 25 steps of deterministic gradient descent with $\alpha_k < 2/L$:

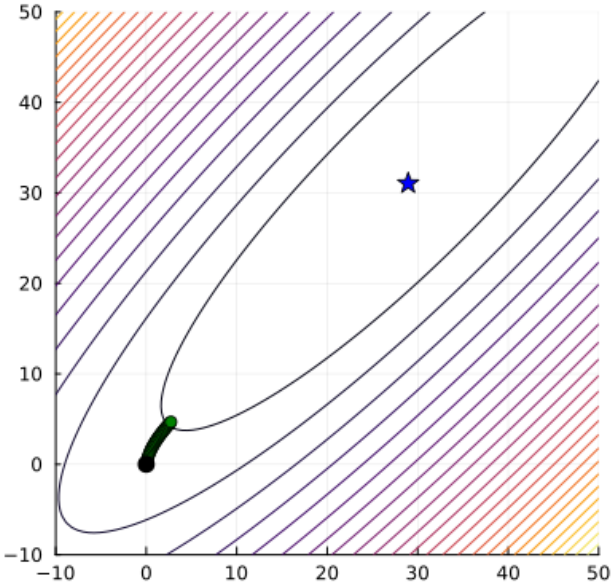
1/L



1/2L



1/10L

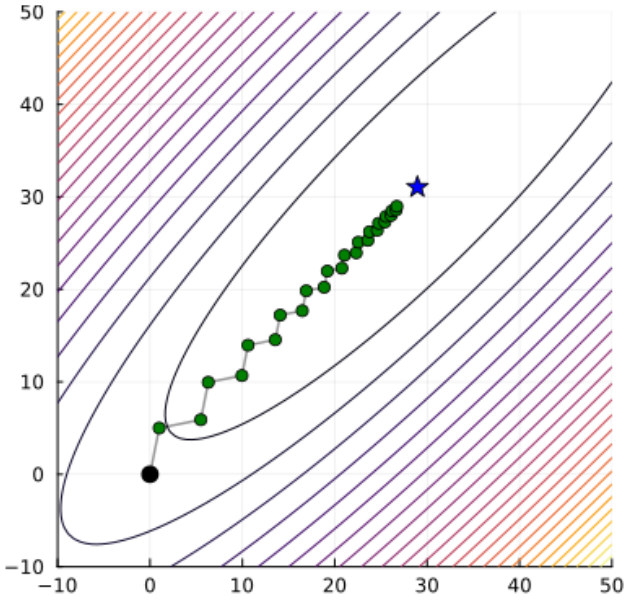


- Progress is very slow if α_k is too small.

Comparison of Fixed Step Sizes for GD

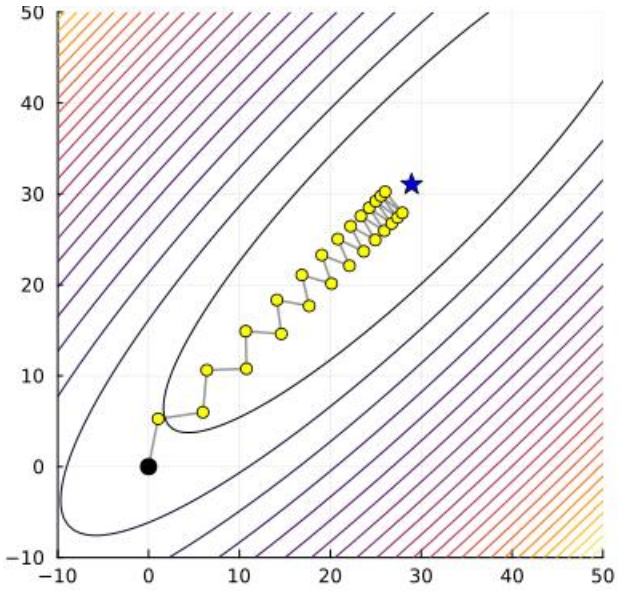
- 25 steps of deterministic gradient descent α_k close to $2/L$:

1.9/L



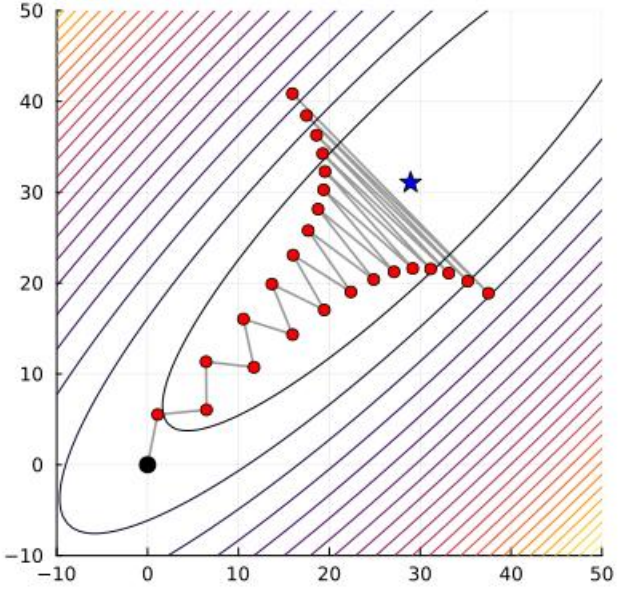
Decreases f on every step. Dampening oscillations.

2/L



Decreases f on every step. Oscillations stay constant (no convergence).

2.1/L



Increases f on later steps. Increasing oscillations (divergence).

GD and the Descent Lemma

- The α_k vs. $2/L$ trade-off comes from the descent lemma:

$$f(v) \leq f(w) + \nabla f(w)^T (v - w) + \frac{L}{2} \|v - w\|^2$$

  - Which uses that $\|\nabla^2 f(w)\| \leq L$ in second-order term of Taylor expansion.

- Set $w = w_k$ and set v to GD update to give:

$$f(w_{k+1}) \leq f(w_k) - \alpha_k \left(1 - \frac{\alpha_k L}{2}\right) \|\nabla f(w_k)\|^2$$

  - Guarantees $f(w_{k+1}) \leq f(w_k)$ if second term is negative.
    - Which happens if $\alpha_k < 2/L$.
  - The optimal step size α_k according to the bound is $\alpha_k = \frac{1}{L}$.

[Bertsekas, 1995]

Convergence and "Optimal" $1/L$ Step Size.

- Consider using a fixed step size, $\alpha_k = \alpha$, where $\alpha < \frac{2}{L}$.
  - Guaranteed to decrease f at every step.

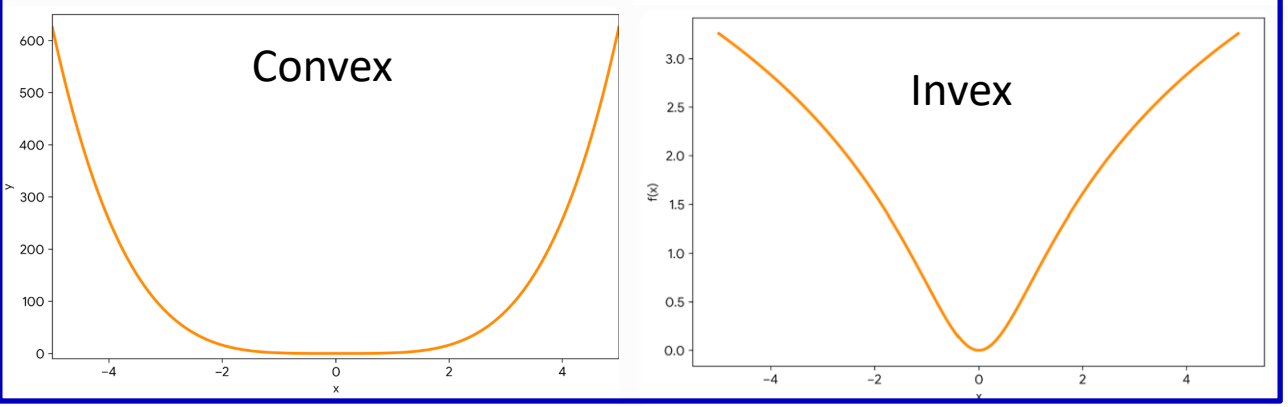
- Descent lemma progress guarantees standard convergence results.
  - If f is convex (or invex) $f(w_k)$ converges to f_* (if exists).
  - If f is strongly-convex (or PL), $f(w_k)$ converges "fast" to f_*.
  - If f is non-convex but bounded below, $\|\nabla f(w_k)\|$ still converges to 0.

- So "small constant step size works in many settings."

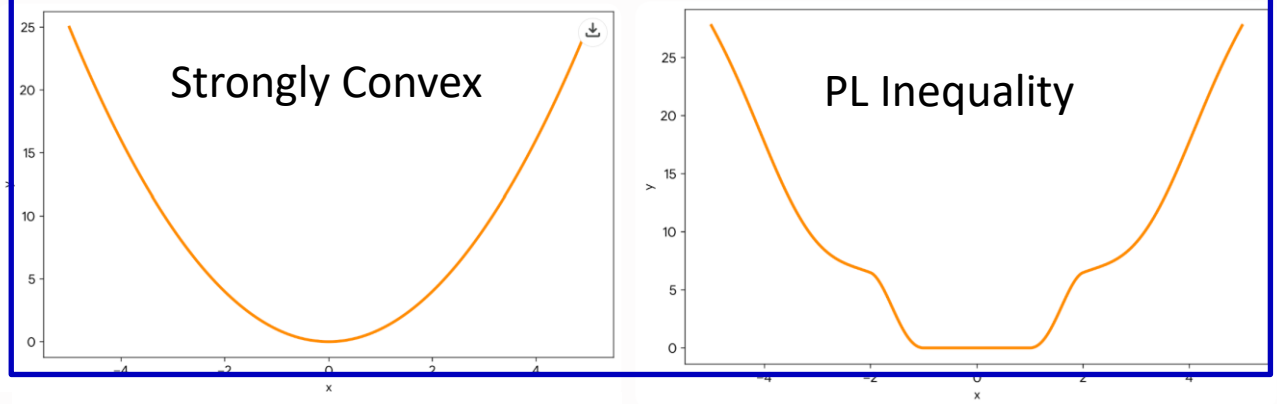
[Polyak, 1963, Lojasiewicz, 1963, Nesterov, 1998]

Convergence of Gradient Descent in Pictures

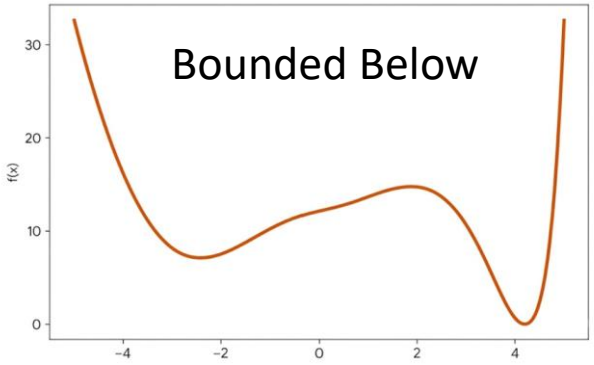
$f(w_k)$ converges to f_*



$f(w_k)$ converges "fast" to f_*



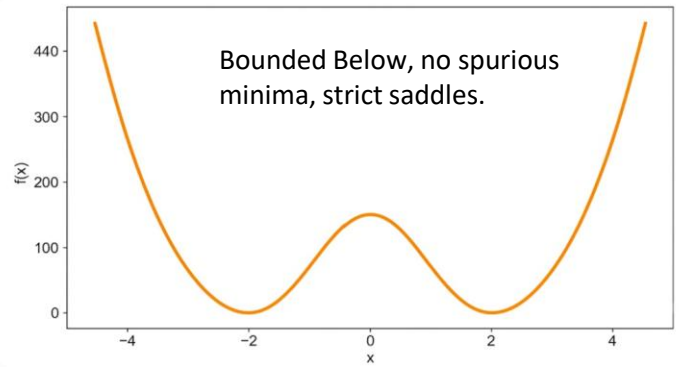
$\|\nabla f(w_k)\|$ converges to 0



[Nesterov, 1998, Karimi et al., 2016, Lee et al., 2016]

These are sufficient but not necessary:

$f(w_k)$ converges to f_*



“Safe” Parameter-Free Algorithms

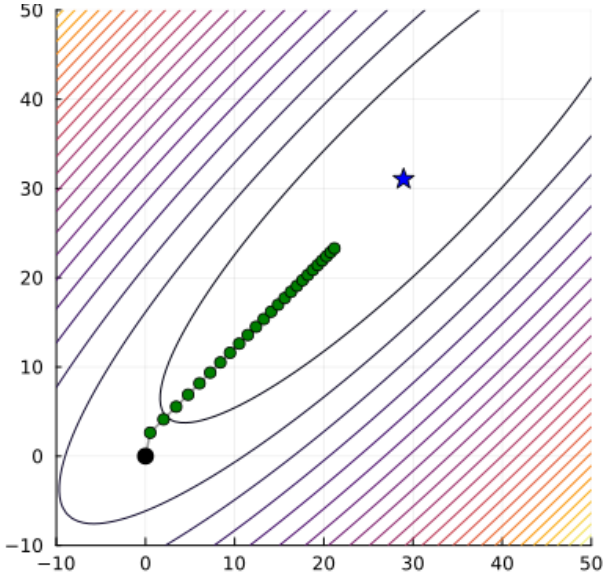
- We typically do not know L.
  - Many step sizes guarantee similar to progress to fixed $\alpha_k = \alpha < \frac{2}{L}$.
- Classic is AdaGrad-Norm:

$$\alpha_k = \frac{\alpha}{\sqrt{\delta + \sum_{t=0}^{k} \|\nabla f(w_t)\|^2}}$$

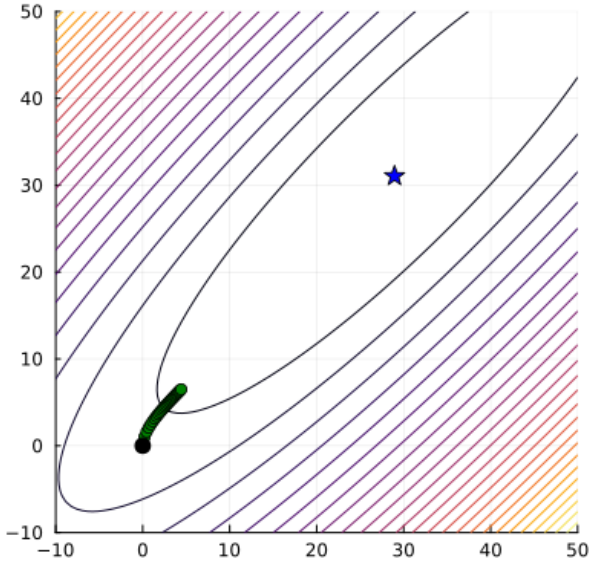
  - For any α, eventually gets similar progress to $\alpha_k = \frac{\gamma}{L}$ for some $\gamma < 2$.
    - But still pretty sensitive to choice of α.
- Recent variant is Prodigy:
  - Adapts numerator based on estimating distance to solution.
    - Practical variant that reduces effect of early gradients.
  - Among best “completely parameter free” method for neural networks.

[Ward et al., 2020, Mischenko & Defazio, 2023]

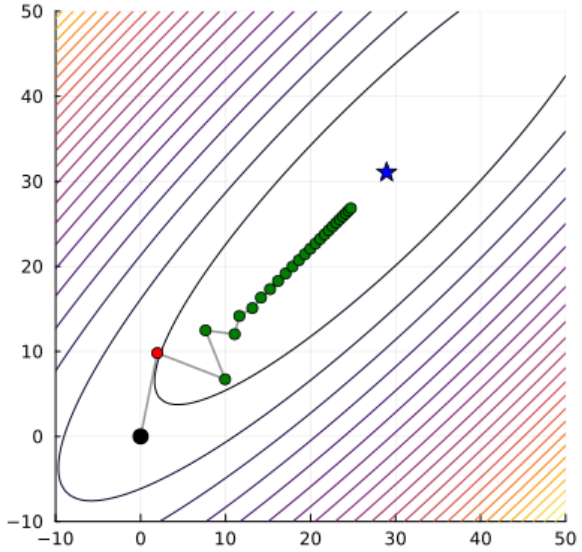
1/L



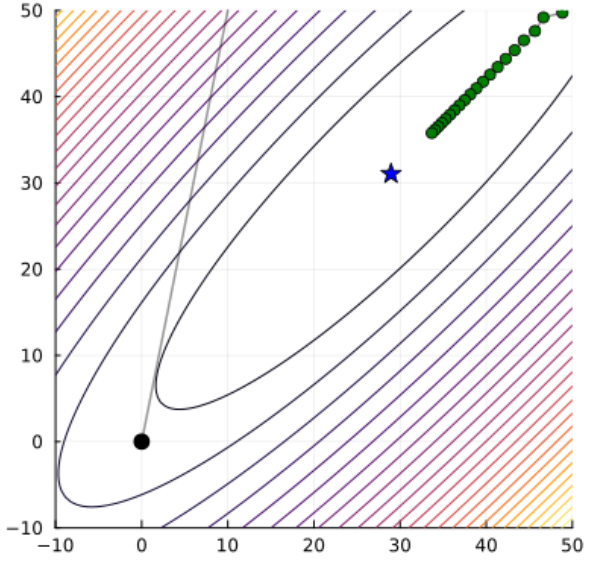
AdaGradNorm ($\alpha = 1, \delta = 10^{-8}$)



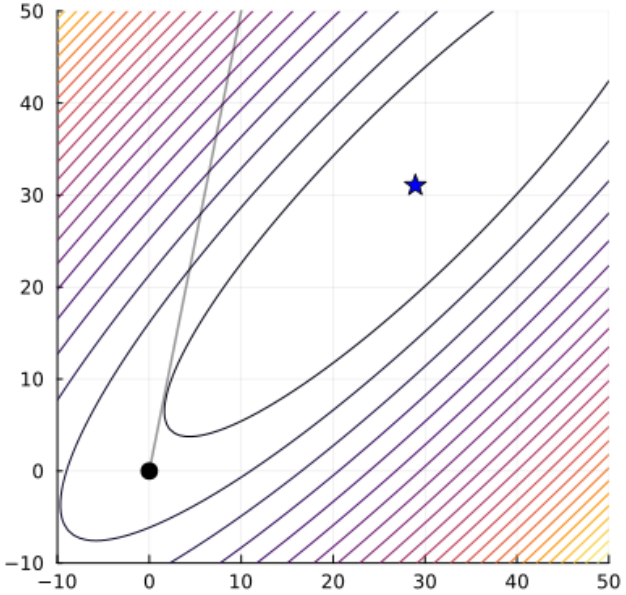
AdaGradNorm ($\alpha = 10$)



AdaGradNorm ($\alpha = 100$)



AdaGradNorm ($\alpha = 1000$)



Smooth-Until-Guilty Backtracking

- Also exist step sizes that are competitive with optimal $\alpha_k = 1/L$.

- Smooth-until-guilty (SUG) backtracking algorithm:
  - At each iteration initialize guess L_k to previous guess L_{k-1}.
  - Check whether descent lemma inequality is satisfied with $\alpha_k = 1/L_k$.
    - If yes, accept the step (makes similar progress to optimal step size).
    - If not, double L^k and test again (until it is satisfied).

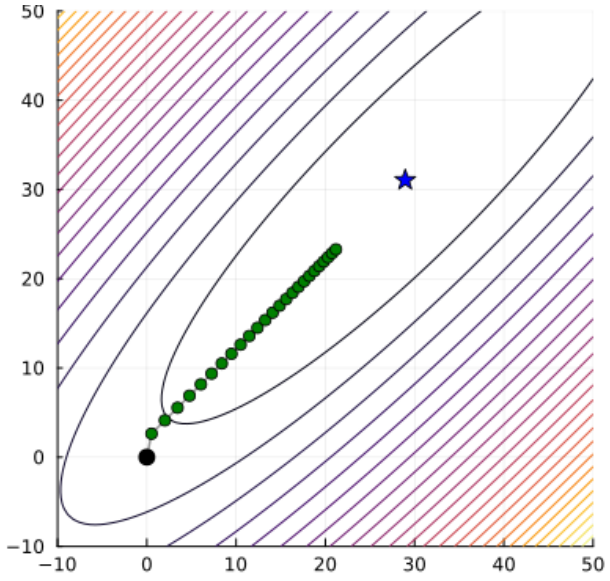
- SUG guarantees same progress bound as using $\alpha_k = \frac{1}{2L}$.
  - For any small enough L_0.

[Beck & Teboulle, 2009]

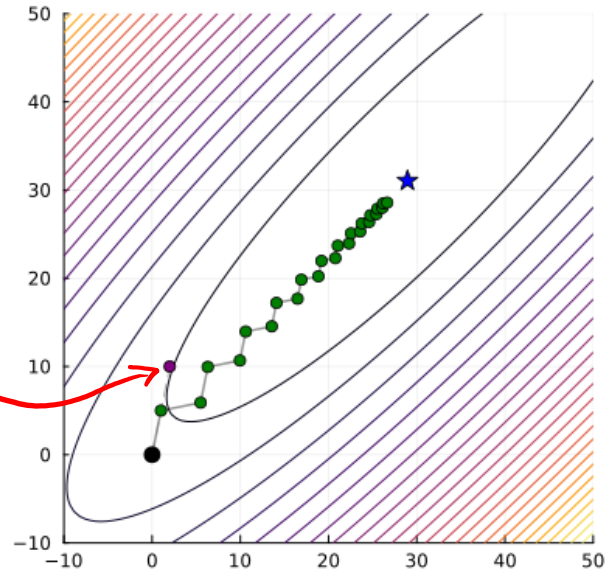
Comparing

- 25 gradient evals with $\alpha_k = \frac{1}{L}$ or smooth-until-guilty (SUG) algorithm:

1/L

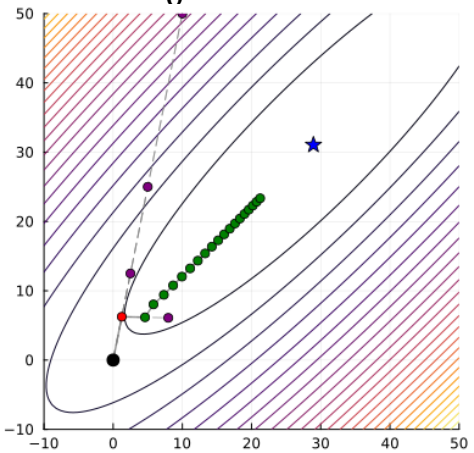


Smooth Until Guilty ($L_0 = 1$)

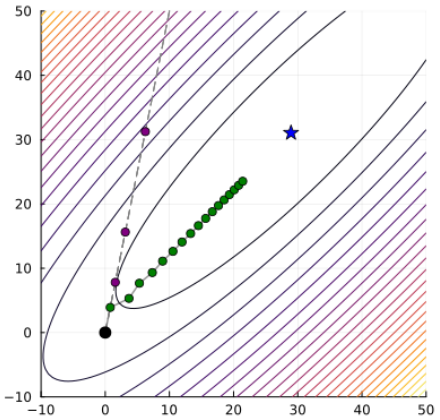


rejected: $\alpha_0 = 1$ did L_0 not satisfy descent lemma.

$L_0 = 0.1$



$L_0 = 0.01$



- Often works better than $\alpha_k = 1/L$.
- Total number of rejects is at most logarithmic.

Question: How do I really set the GD step size?

Beyond 2/L and 1/L

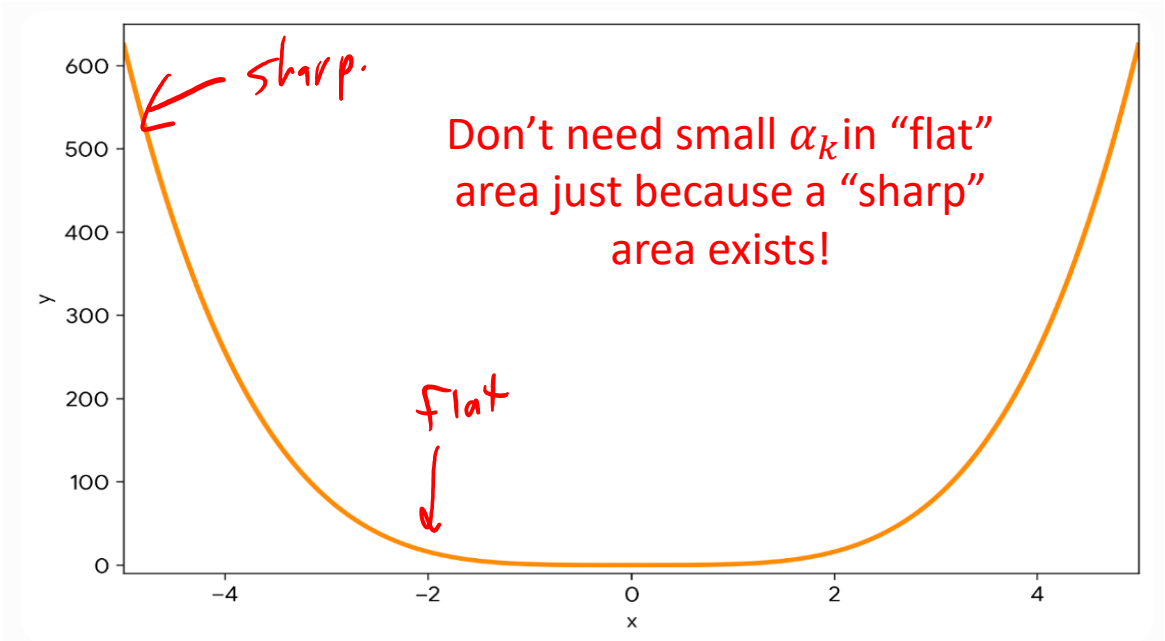
- Reality is more complicated than the "safe" $2/L$ and "optimal" $1/L$.
  - Using $\alpha^k = 1/L$ or SUG work terribly in practice.
  - And for neural networks L=∞ so GD needs $\alpha_k < 0$?

- To understand better, let's review how we got to $2/L$ and $1/L$:
  1. Define L as global maximum of sharpness ($\|\nabla^2 f(w)\|$, max singular value).
  2. Bound 2nd-order term in Taylor expansion ($d^{\mathrm{T}}\nabla^2 f(w)d \leq L\|d\|^2$).
  3. Look at the effect of taking 1 step of GD.

- This makes "apply descent lemma repeatedly" a very loose bound.
  - Let's relax each of the 3 bounds…

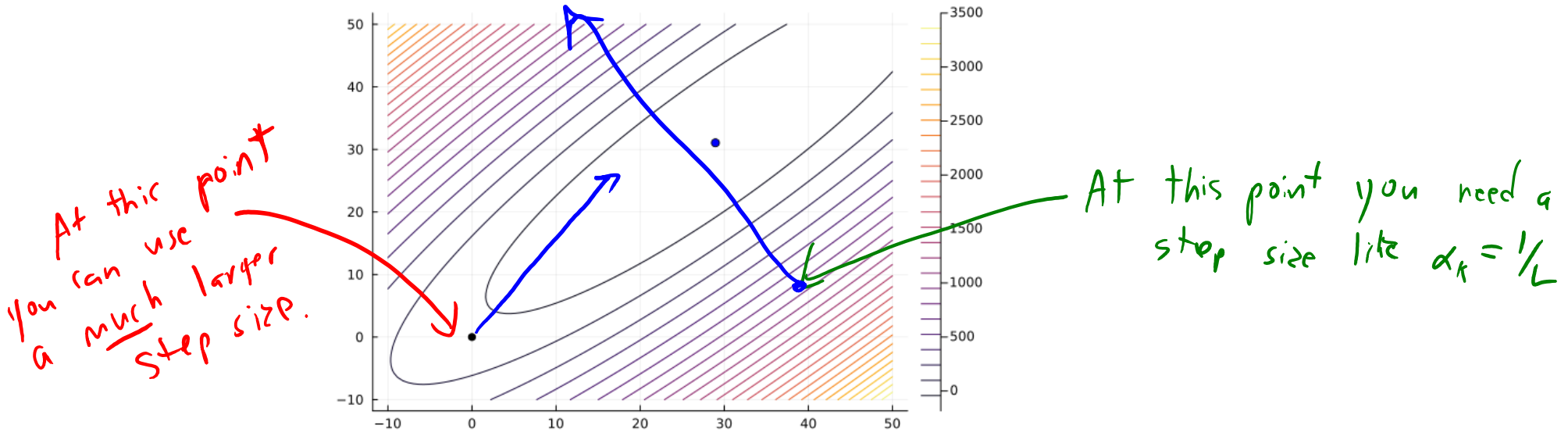
Beyond 2/L and 1/L

- Local sharpness bound $L(w_k)$ may be much smaller than global L.
  - "Not always in sharpest region".



Beyond 2/L and 1/L

- Local sharpness bound $L(w_k)$ may be much smaller than global L.
  - "Not always in sharpest region".
  - Our safe step size limit is now $2/L(w_k)$ and our optimal step size is $1/L(w_k)$.
    - "Can take bigger steps in flatter regions".
      - For neural networks we have finite $L(w_k)$ so $\alpha_k > 0$.
- Quadratic form $v^T \nabla^2 f(w^k) v$ may not be maximized by gradient.
  - "Gradient may not point in sharpest direction".

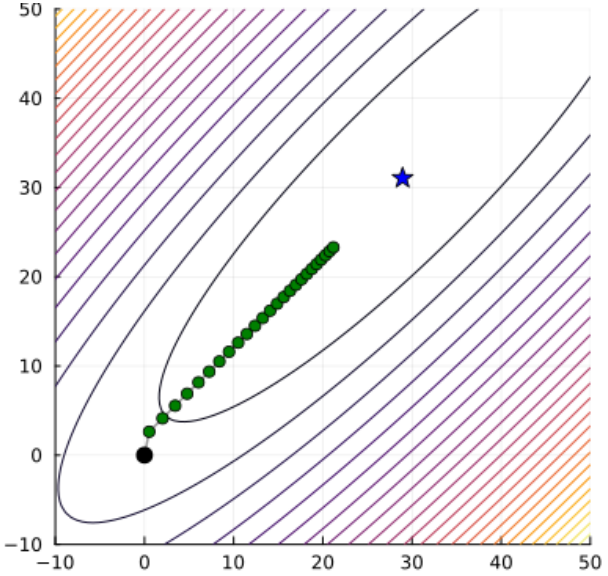


Beyond 2/L and 1/L

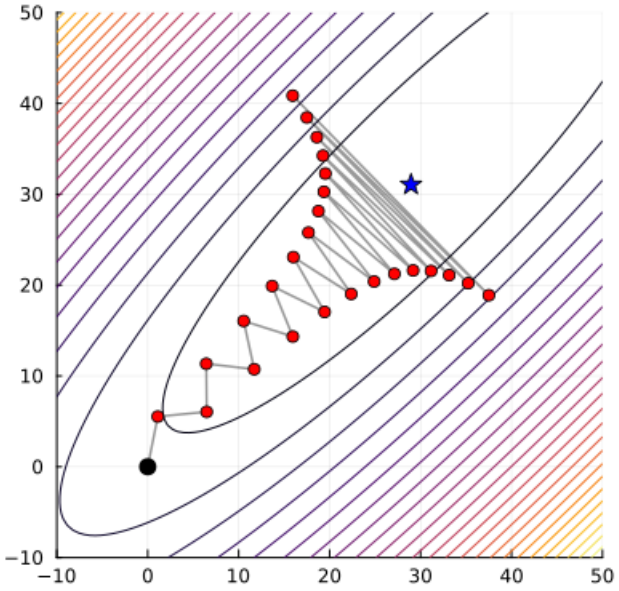
- Local sharpness bound $L(w_k)$ may be much smaller than global L.
  - "Not always in sharpest region".
  - Our safe step size limit is now $2/L(w_k)$ and our optimal step size is $1/L(w_k)$.
    - "Can take bigger steps in flatter regions".
      - For neural networks we have finite $L(w_k)$ so $\alpha_k > 0$.
- Quadratic form $v^{\mathrm{T}}\nabla^2 f(w^k)v$ may not be maximized by gradient.
  - "Gradient may not point in sharpest direction".
  - Bounding second-order Taylor expansion by $L(w_k)$ can be too conservative.
    - For least squares, optimal step size is $\|\nabla f(w_k)\|^2/\nabla f(w_k)^T\nabla^2 f(w_k)\nabla \mathrm{f}(\mathrm{w_k})$.
      - And the safe step limit can be as large as 2/(smallest non-zero eigenvalue of Hessian).
    - If gradient in direction of negative curvature, can take a huge step to escape a saddle point.
- Consider that you take more than 1 step of GD.
  - May want to take an "unsafe" step if it leads to more progress overall later.

Beyond 2/L and 1/L

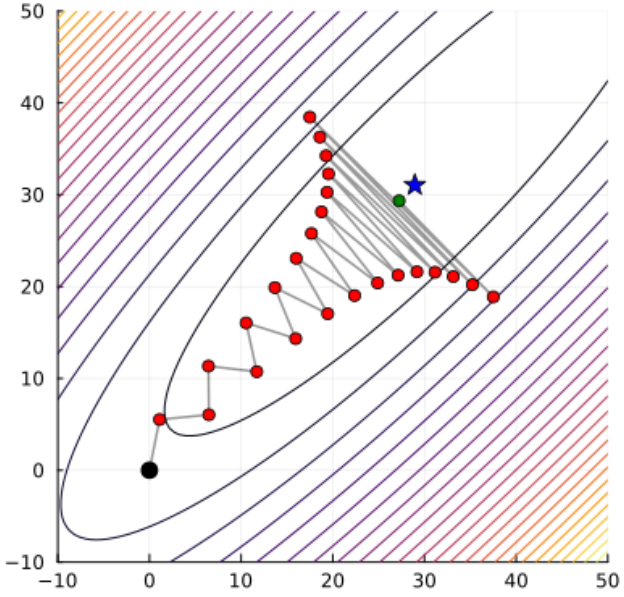
1/L for 25 steps



2.1/L for 25 steps



2.1/L for 24 steps, 1/L for 1 step

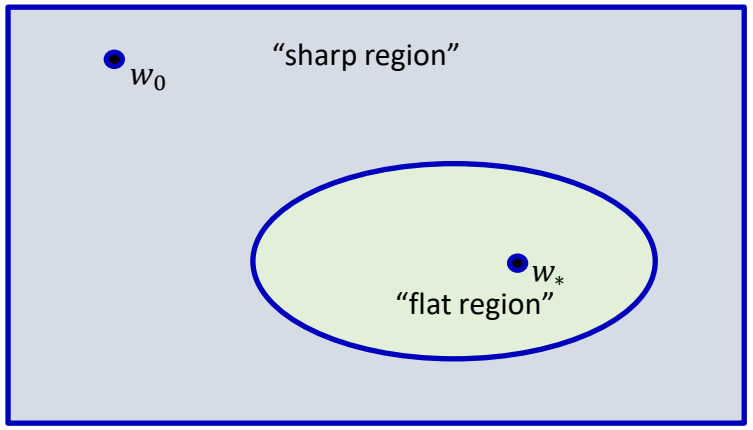


"diverged" for 24 steps then got closer to solution!

- Consider that you take more than 1 step of GD.
  - May want to take an "unsafe" step if it leads to more progress overall later.

Simple Model of $L(w)$: Glocal Smoothness

- Consider a function that satisfies glocal smoothness:
  - Globally L-smooth.
  - Locally L_*-smooth near solution, where $f(w) - f_* \leq \delta$.

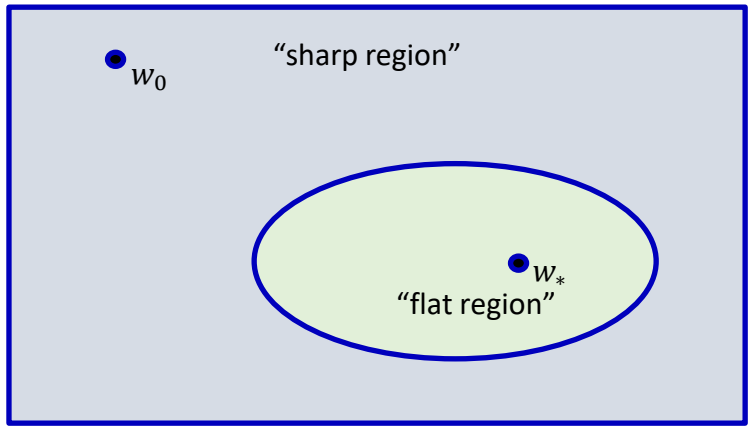


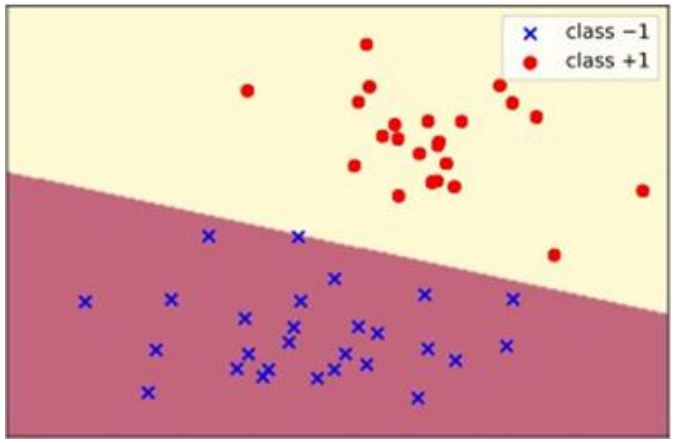
- Under this assumption, need to increase step size in flat region.
  - Using $\alpha_k = \frac{1}{L}$ or SUG can perform arbitrarily bad.

[Fox et al., 2026]

Can we really have $L_* \ll L$?

- Consider logistic regression with separable data:



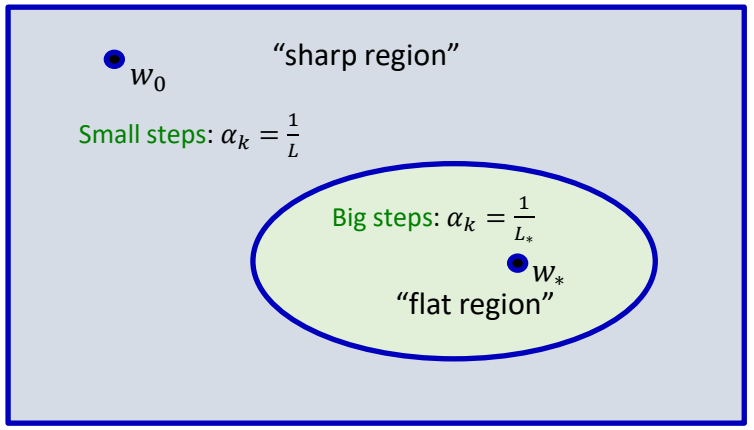


- For data matrix X, usual estimate of L is $\frac{1}{4}\|X\|^2$.
- Once we have $p(y_i) > 0.99$, $L \approx \frac{1}{100}\|X\|\text{^}2$.
  - GD can take steps that are 25-times larger.
- Glocal Lipshitz constant given δ is $L_* = \delta\|X\|\text{^}2$.
  - GD can eventually take arbitrarily-larger steps.

[Bohning, 1992, Fox et al., 2026]

Simple Model of $L(w)$: Glocal Smoothness

- Optimal sequence of step sizes under glocal smoothness:
  - Use $\alpha_k = \frac{1}{L}$ in "sharp" region
  - Switch to $\alpha_k = \frac{1}{L_*}$ in "flat" region.



- Problem: do not know L, L_*, or when we switch regions.

Simple Model of $L(w)$: Glocal Smoothness

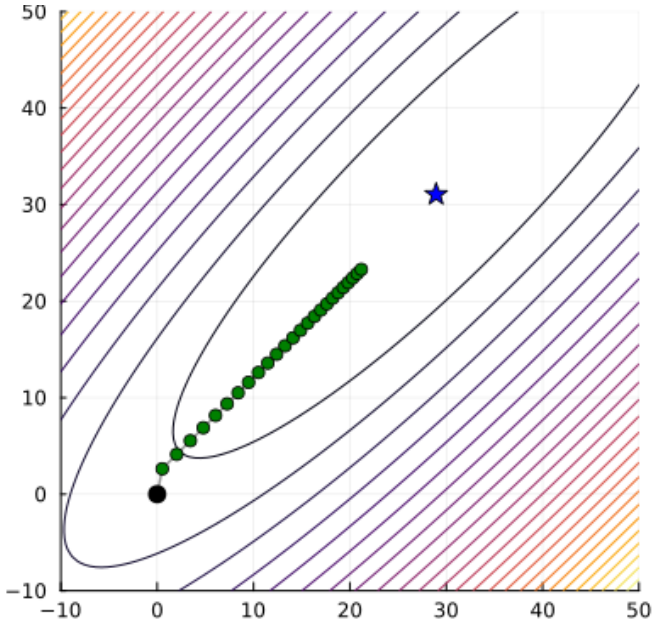
- But practical algorithms achieve rate of optimal sequence:
  - SUG with resets of L_k to small values.
    - Often relaxing decrease to allow larger step sizes (Armijo condition).
  - Greedy step size of minimizing function along gradient direction.
    - Efficient numerically for linear models and SO-friendly neural networks.
    - Approximate using line searches based on Wolfe conditions.
      - Annoying to implement but only small cost increase if implemented carefully.
  - Polyak step size for convex functions.
    - No extra $f(w)$ evals but requires f_*.

$$\alpha_k = \frac{f(w_k) - f_*}{\|\nabla f(w_k)\|^2}$$

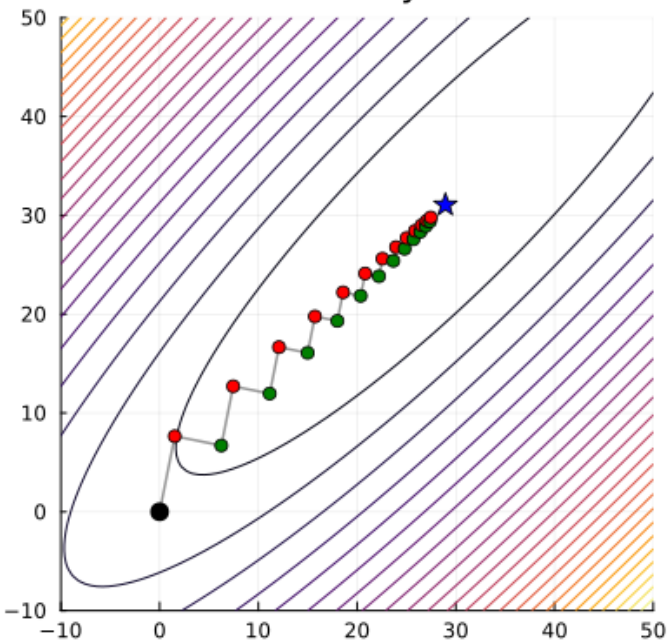
  - AdGd step sizes for convex functions.
    - Safeguarded approximation of $L(w_k)$ along most-recent algorithm step.
- Bonus for these practical methods:
  - Exploit that $\nabla f(w_k)$ may not be in sharpest directions to some extent.

[Polyak, 1987, Nocedal & Wright, 1999, Malitsky & Mischenko, 2019, Shea & Schmidt, 2024]

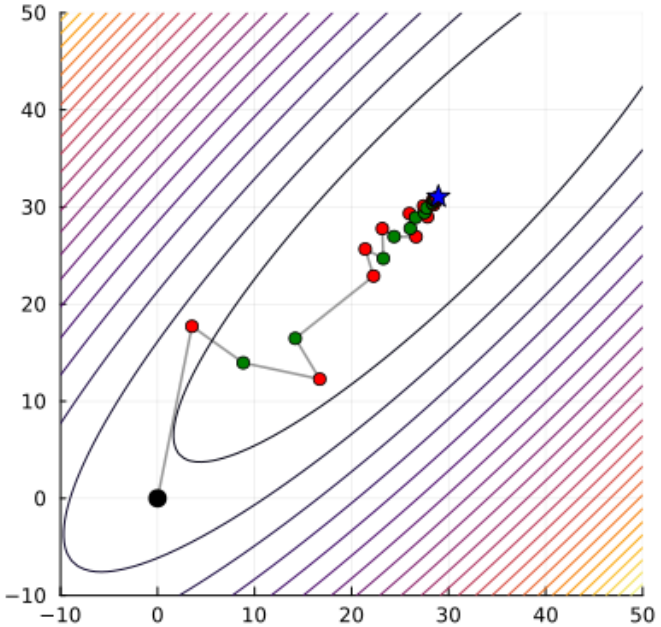
1/L



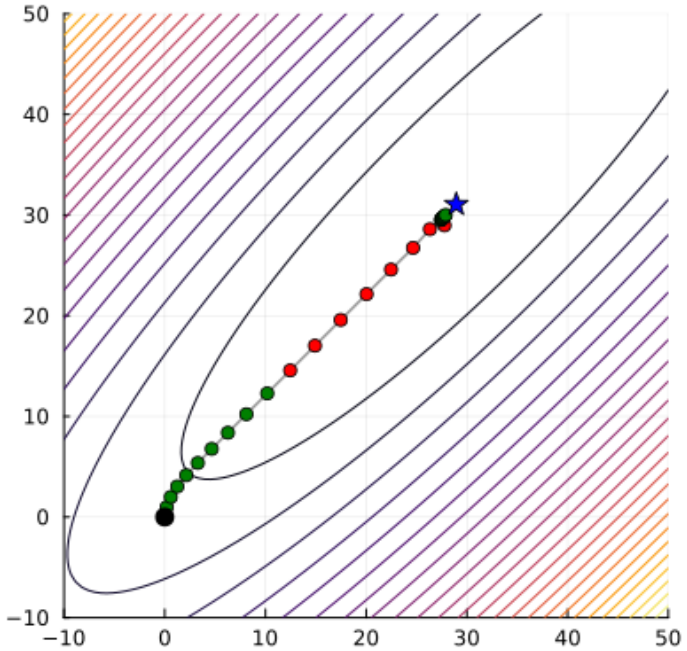
Greedy



Polyak



AdGD



Non-Uniform Smoothness ($L = \infty$)

- For neural networks global $L = \infty$.
  - So globally there is no safe constant step size.
- One model of this is non-uniform smoothness (NUS):

$$\|\nabla^2 f(w)\| \leq L_0 + L_1 \|\nabla f(w)\|$$

  - "Sharpness decreases as gradient decreases."
  - Allows $L = \infty$ but finite $L(w_k)$.
    - Always exists safe step sizes, with larger step sizes allowed as gradient shrinks.
- NUS has been used to justify constant step with gradient clipping:

$$w_{k+1} = w_k - \alpha \cdot \text{clip}\big(\nabla f(w_k)\big) \qquad \text{clip(v)} = \begin{cases} v \text{ if } \|v\| \leq \tau \\ \left(\frac{\tau}{\|v\|}\right) v \text{ if } \|v\| > \tau \end{cases}$$

[Zhang et al., 2020]

Non-Uniform Smoothness ($L = \infty$)

- GD in descent lemma under NUS:

$$f(w_{k+1}) \leq f(w_k) - \alpha_k \|\nabla f(w_k)\|^2 + \frac{\alpha_k^2 (L_0 + L_1 \|\nabla f(w_k)\|)}{2} \|\nabla f(w_k)\|^2.$$

  - Needs to be tiny to make up for cubic red term.

- Clipped(GD) shrinks effective α_k by $\tau / \|\nabla f(w_k)\|$ when clipped.
  - Descent lemma when clipping is active with `max_grad_norm = 1.0`:

$$f(w_{k+1}) \leq f(w_k) - \alpha_k \left(1 - \frac{\alpha_k L_1}{2}\right) \|\nabla f(w_k)\| + \frac{\alpha_k^2 L_0}{2}.$$

  - This is similar to usual GD progress even if $\|\nabla f(w_k)\|$ and α_k are large.
    - But without squared norm and limit on progress (goes away as gradient shrinks).

[Zhang et al., 2020]

Non-Uniform Smoothness ($L = \infty$)

- Another common variant of non-uniform smoothness:

$$\|\nabla^2 f(w)\| \leq L_0 + L_1(f(w) - f_*)$$

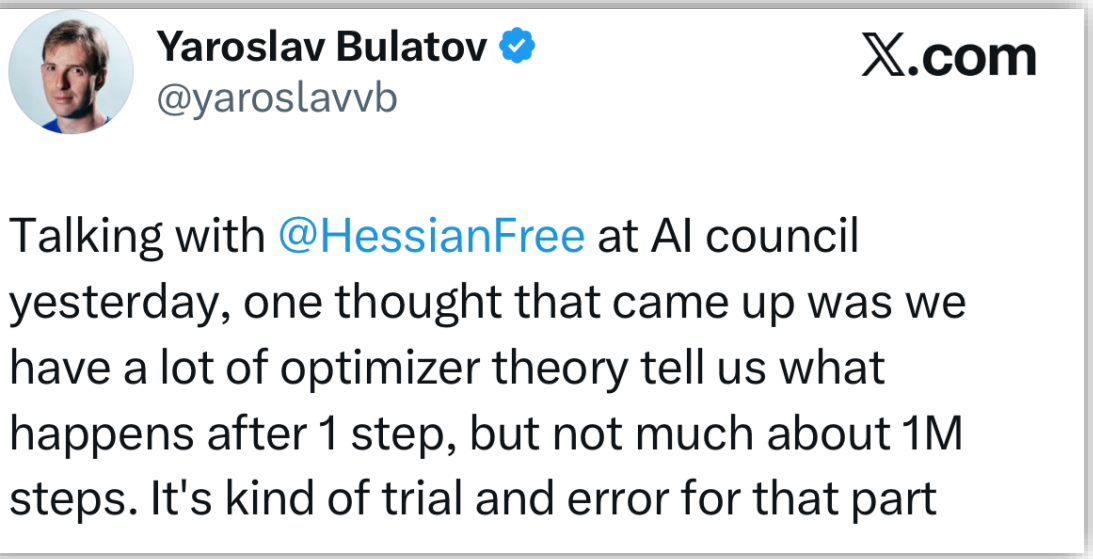
  - Under this assumption we need step size warm-up:
    - Need small but increasing step sizes at start to deal with large initial sharpness.
  - For separable logistic, adaptive α_k goes from "sub-linear" to "linear" rates.
- Optimization community has explored relative smoothness:

$$\|\nabla^2 f(w)\| \leq L\|\nabla^2 g(w)\|$$

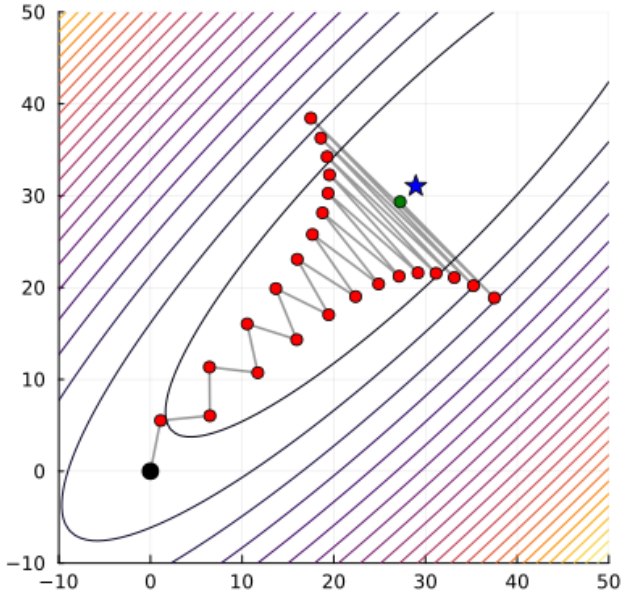
  - Smooth compared to a reference function g.
    - Leads to global progress bounds for functions like $|w^4|$, and tight EM analysis.

[Bauschke et al., 2017, Lu et al., 2018, Kunstner et al., 2021, Zhang et al., 2025, Alimisis et al. 2025, Vaswani et al., ICML 2026]

Multi-Step Analyses?



2.1/L for 24 steps, 1/L for 1 step

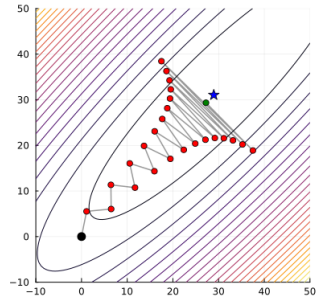


Quadratic Functions

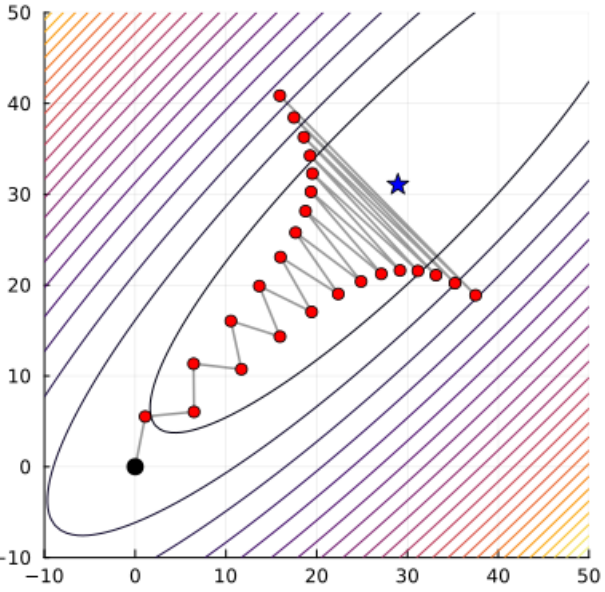
- For multi-step analyses, let's first look at quadratic functions:

$$f(w) = \frac{1}{2} w^T A w + b^T w$$

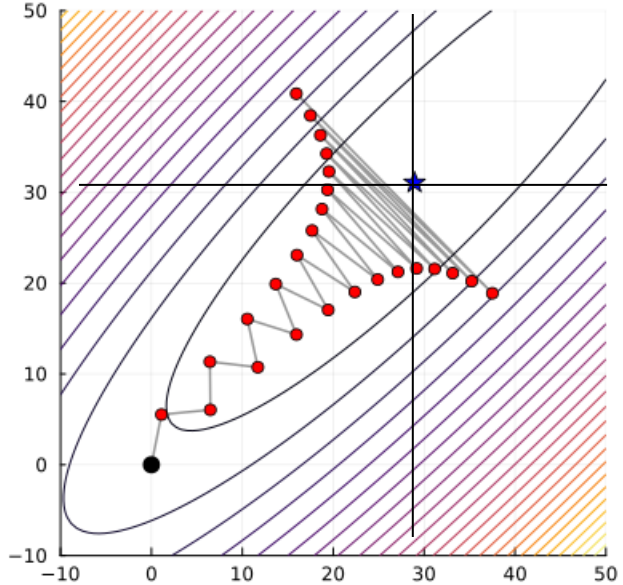
  - Includes linear least squares.
  - Can compute exact dynamics of GD (because gradient is linear).
- With 2 variables visualizing quadratics is easy.
  - But $d = 2$ is "easy" (special properties).



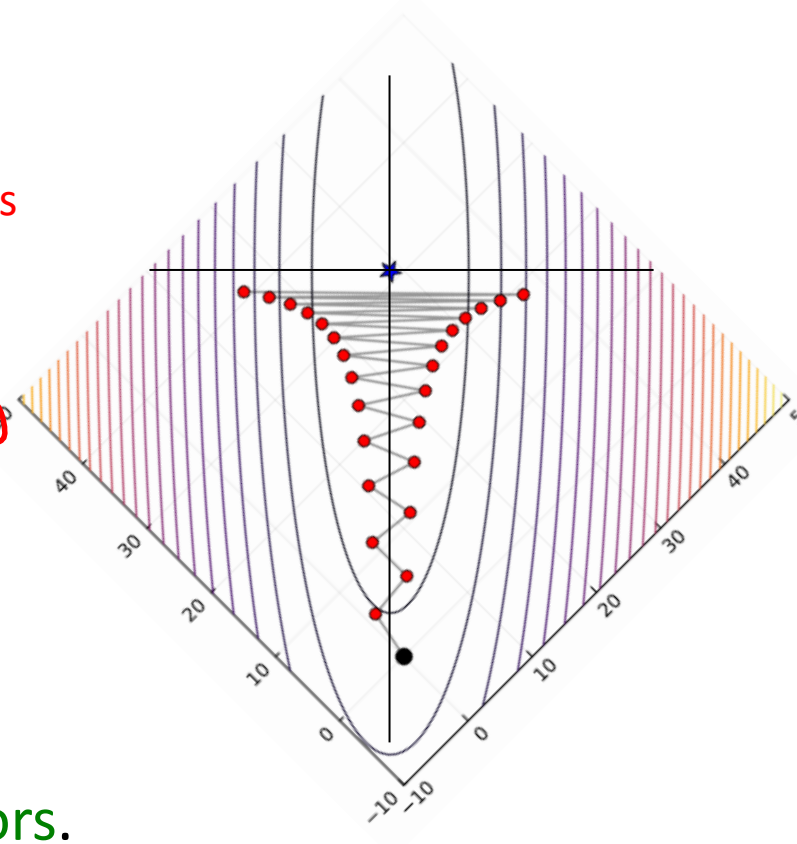
- Visualizing higher-dimensional quadratics is annoying.
  - But we can use that GD is translation and rotation invariant.



Translate so solution is at 0.



Rotate so eigenvectors align with axes

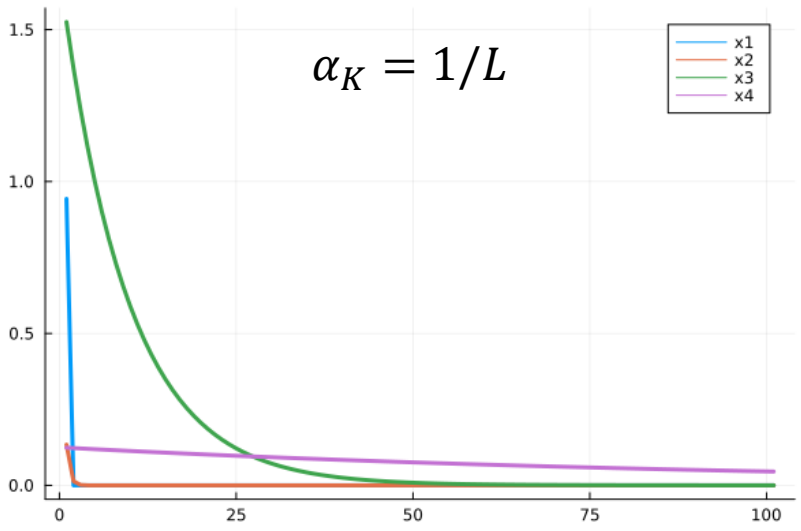


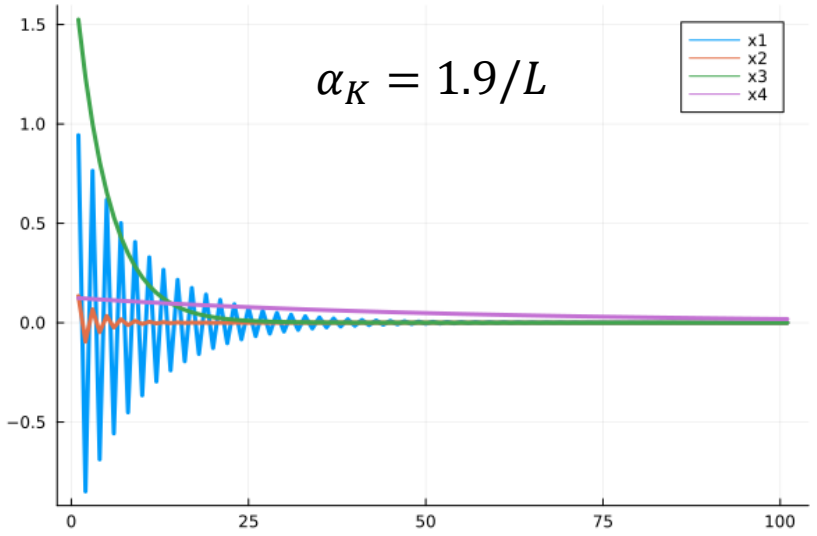
- We now have a diagonal quadratic:
  - Can plot variables to see what happens along eigenvectors.
- Classic valleys and ravines perspective:
  - X-axis eigenvector is a valley (steep).
    - Need small step sizes to avoid going up other side.
  - Y-axis eigenvector is a ravine (flat).
    - Need big step sizes to move along here.
  - $\alpha_k = \frac{2.1}{L}$: progress along ravine, oscillates along valley.

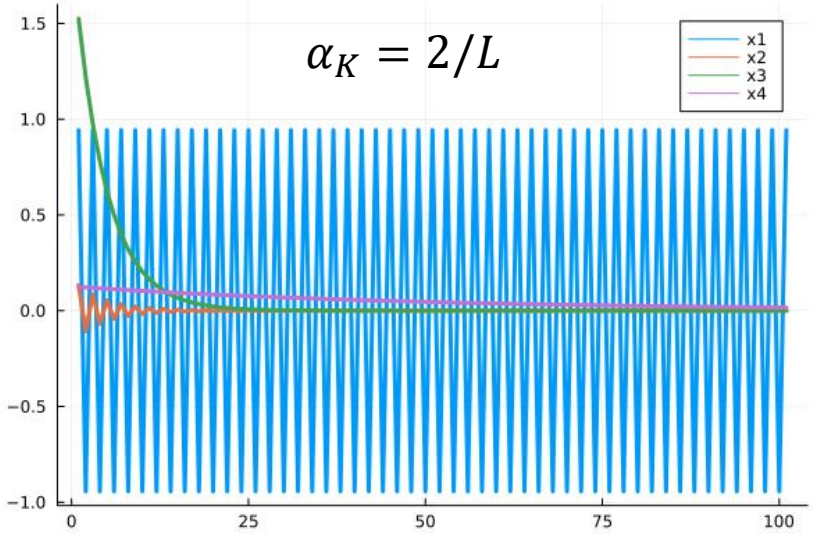
[Gelfand & Tsetlin, 1961]

GD Dynamics on Quadratics – Fixed Step Sizes

- Illustrate multi-step progress using 4-dimensional quadratic:
  - Two valleys with steepness 100 and 90 ("steepest valley" has $L = 100$).
  - Two ravines with steepness 10 and 1 ("flattest valley" has $\mu = 1$).

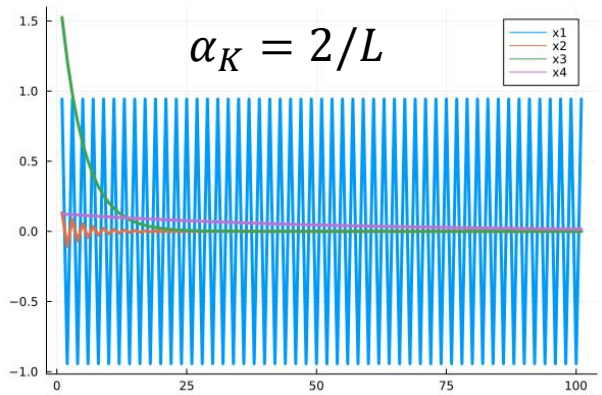






GD Dynamics on Quadratics – Fixed Step Sizes

- At each step, update of variable j is given by:



$$w_j^{k+1} = (1 - \alpha_k \lambda_j) w_j^k.$$

$\alpha_k \leq \frac{1}{\lambda_j}$ converge with same sign

$\alpha_k > \frac{1}{\lambda_j}$ it flips sign (oscillations)

Converge if $\alpha_k < \frac{2}{\lambda_j}$

Just flip sign if $\alpha_k = \frac{2}{\lambda_j}$

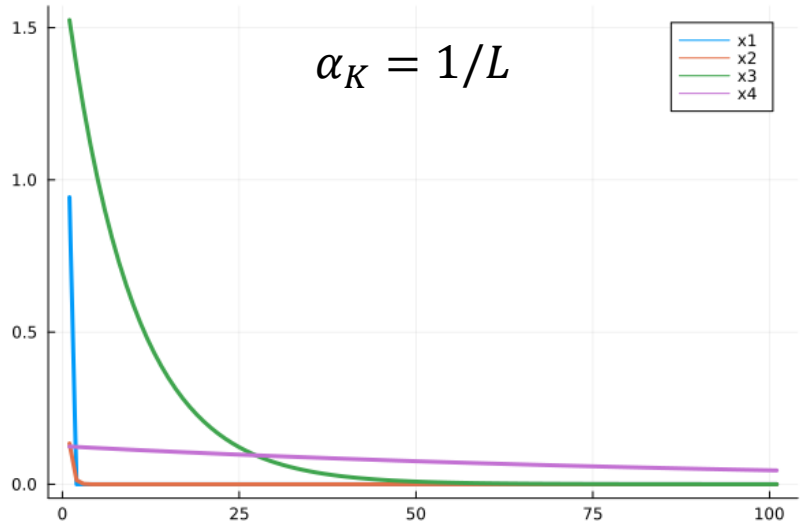
Diverge if $\alpha_k > \frac{2}{\lambda_j}$

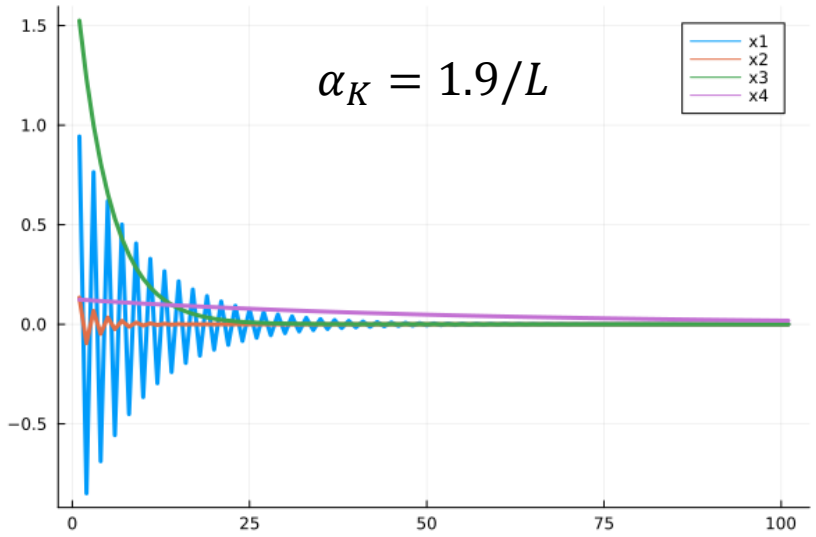
- The optimal fixed α_k for $\lambda_j \in [\mu, L]$ is:

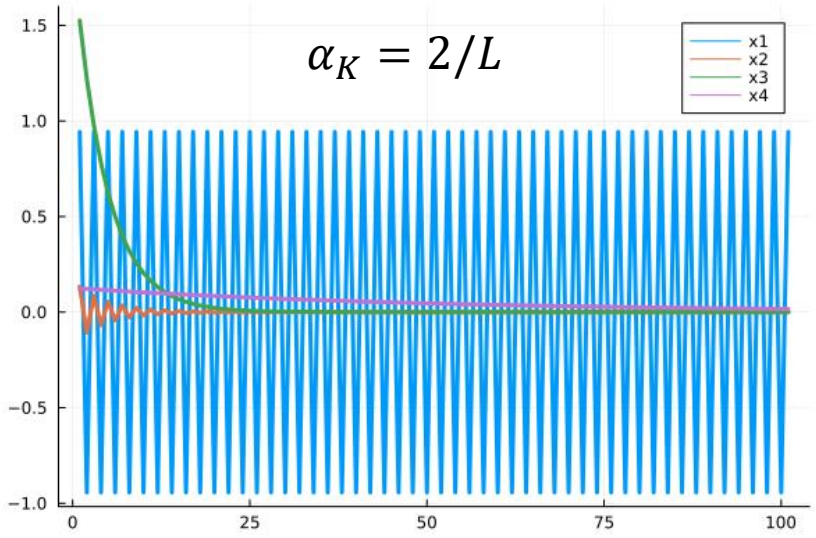
$$\alpha_k = \frac{2}{L + \mu}$$

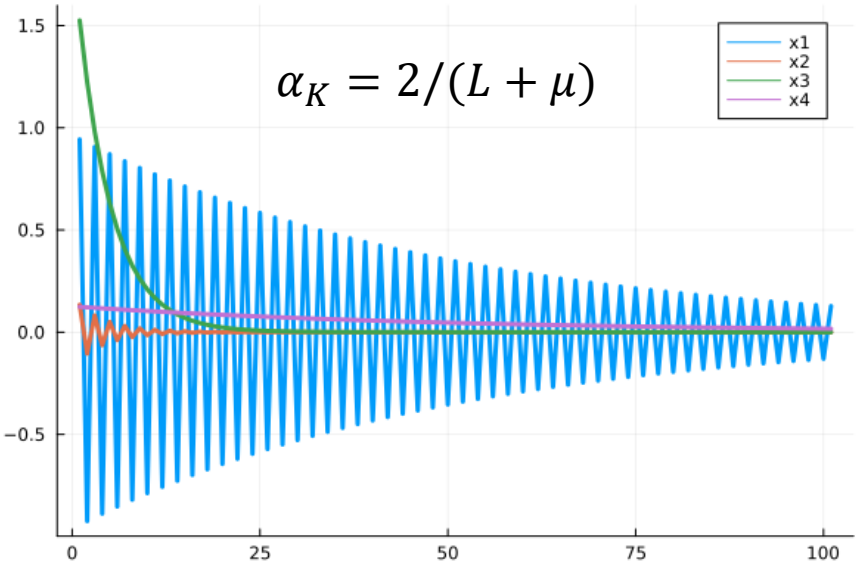
  - Converge at same speed along steepest valley and flattest ravine.
    - Slows down progress along valley to speed up progress along ravine.
  - For practical problems, really close to safe limit of $\alpha_k = 2/L$.

GD Dynamics on Quadratics – Fixed Step Sizes





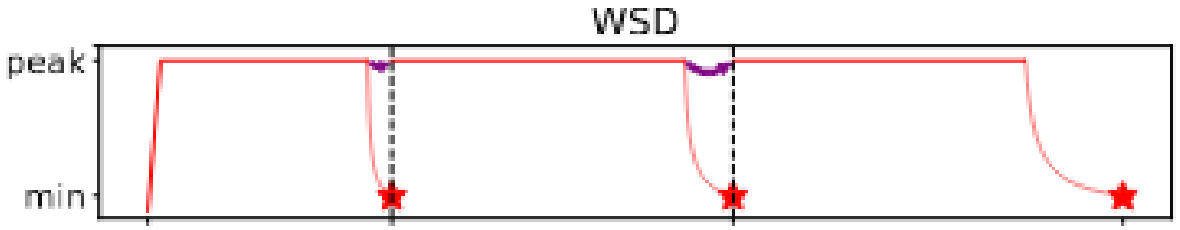


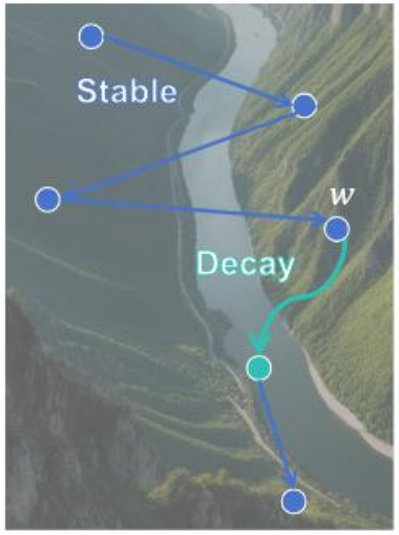


In this case $\alpha_k = 1.9/L$ is better over 100 steps because of initialization (starts close to bottom of flattest ravine)

Digression: “WarmUp Stable Decay”

- A recently-trendy strategy: “warmup stable decay”.





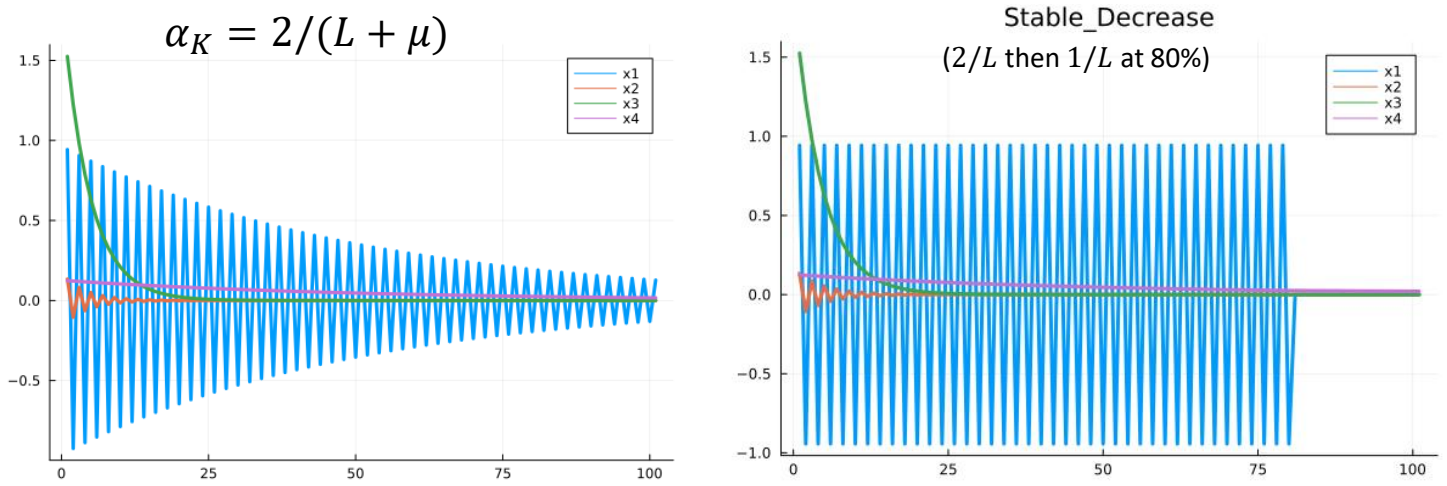
(a) River Valley Landscape

  - Start with warm-up (small but increasing step sizes for initial sharpness).
  - Switch to using large constant step size.
  - Occasionally record “checkpoint” and explore smaller step sizes.
    - Then return to “checkpoint” and keep going.

[Hu et al., 2024, Wen et al., 2024]

~~WarmUp Stable Decay~~ Step Decay

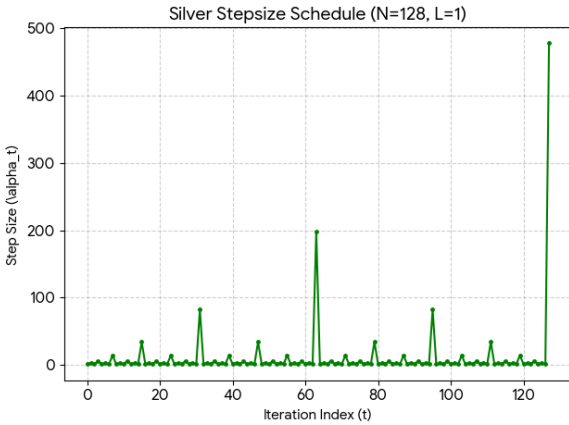
- For quadratics we do not need warmup (constant $L(w)$).
  - For deterministic optimization we do not need decay to 0.
- Let's compare optimal step size to step decay:
  - Use big step size for a long time, then switch to a small step size.



- Moved quickly down ravine with large step size.
  - Moved quickly down valley with small step size.
    - Function value was barely changing during first phase (looks like "slow progress").

Multi-Step Analyses – Convex Functions

- Multi-Step analyses for convex functions:
  - PEP framework can give optimal step size if running for T iterations.
    - If only doing 2 iterations, use $\alpha_0 = 2/3L$ then $\alpha_1 = 4/3L$.
    - For large k, starts with $\alpha_0 \approx 1/2L$ and increases to $\alpha_k \approx 2.2/L$.
  - Altschuler's silver step size and Grimmer's long-steps:
    - Chain together cycles of step sizes that provably improve speed.
      - 2-cycle: $1.5/L$ then almost $3/L$ then repeat.
      - 3-cycle: $1.5/L$ then $4.9/L$ then 1.5/L then repeat.
      - 7-cycle: $1.5/L$ then 2.2/L then 1.5/L then $12/L$ then then 1.5/L then 2.2/L then 1.5/L then repeat.

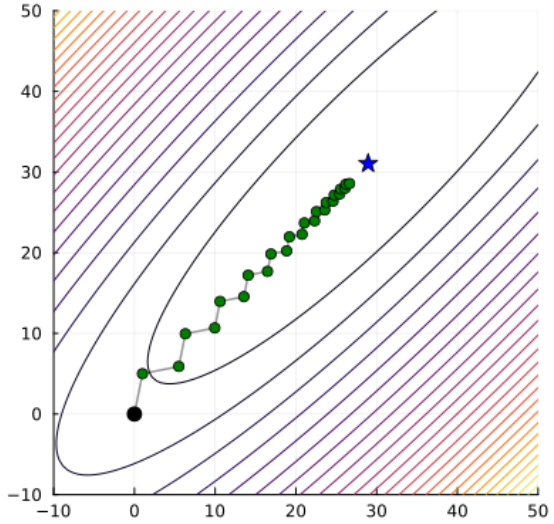


- Impressive theory but don't work great in practice.
  - Do not account for local $L(w_k)$ or specific directions $\nabla f(w_k)$.
- Recent work: use small constant with occasional Polyak step.
  - Fast under weaker conditions.

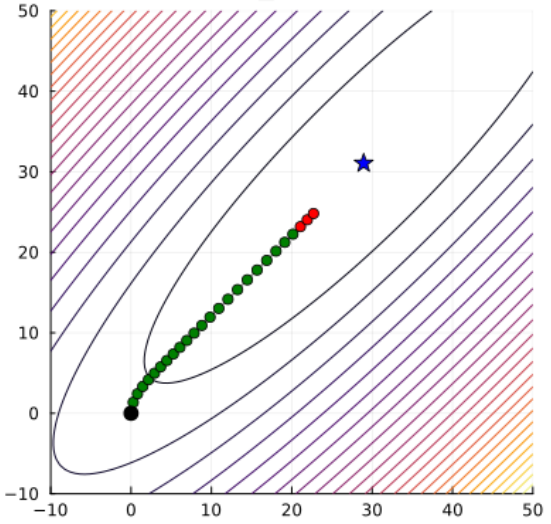
[Drori & Teboulle, 2014, Grimmer, 2023, Davis et al., 2024, Altschuler & Parrilo, 2025]

Multi-Step Analyses - Convex Functions

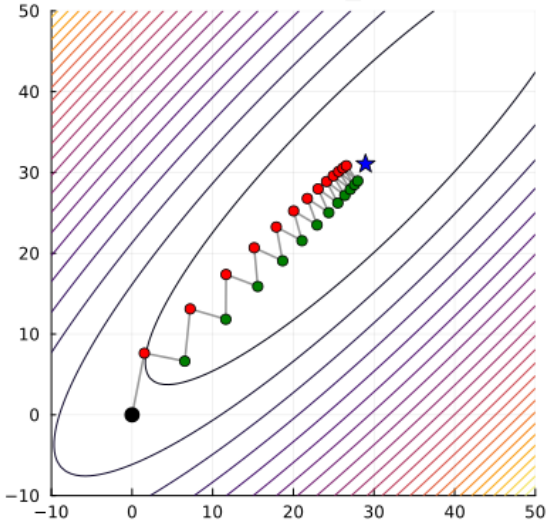
$2/(L+\mu)$



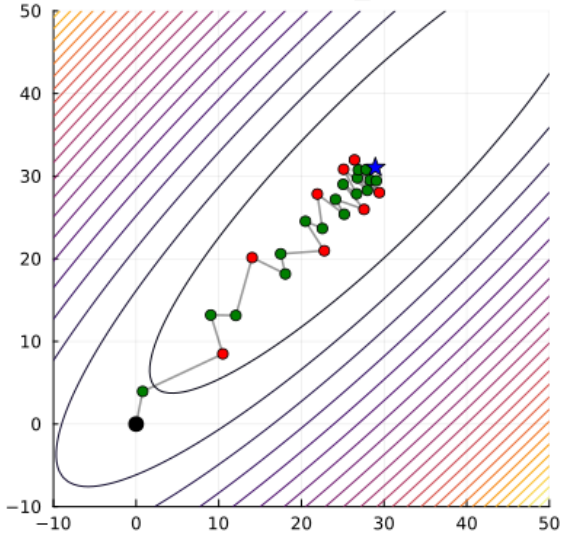
PEP_Approx



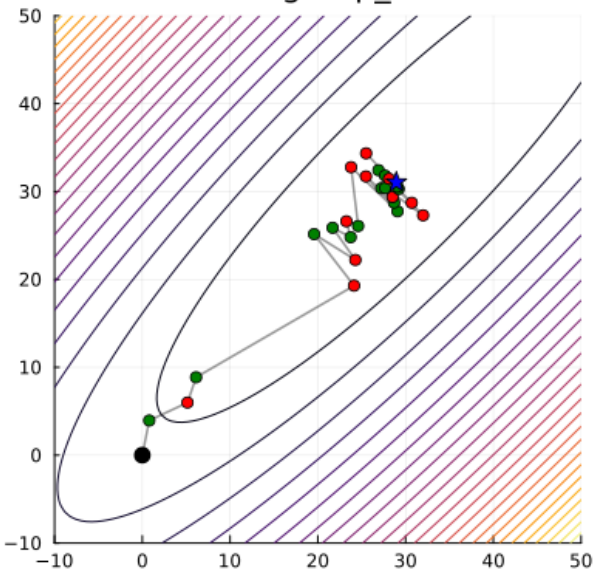
LongStep_2



LongStep_3

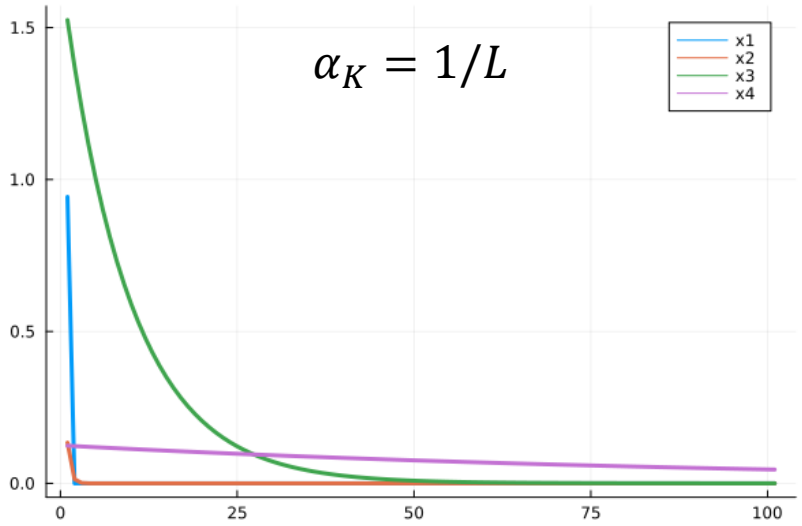


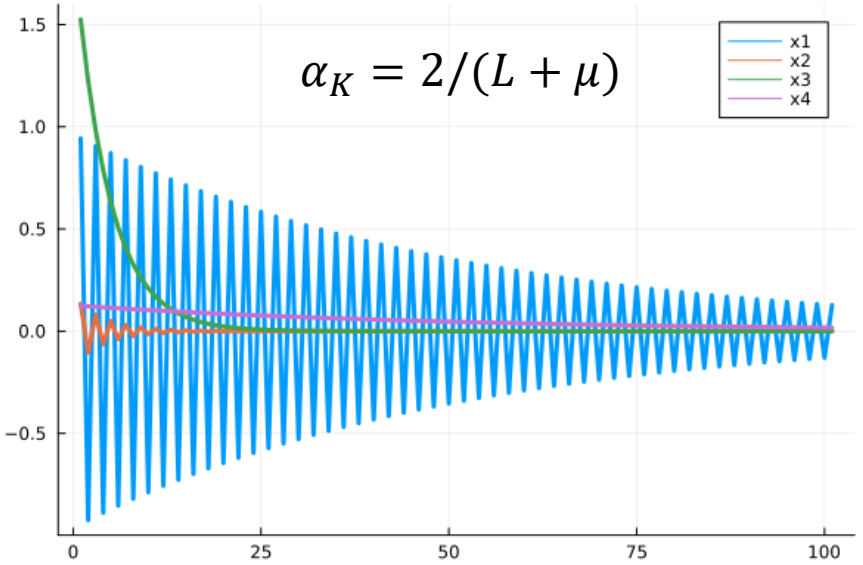
LongStep_7

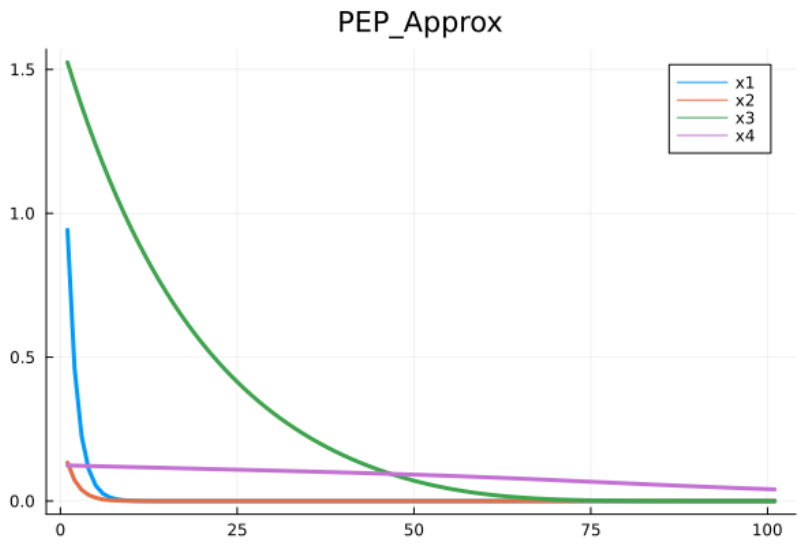


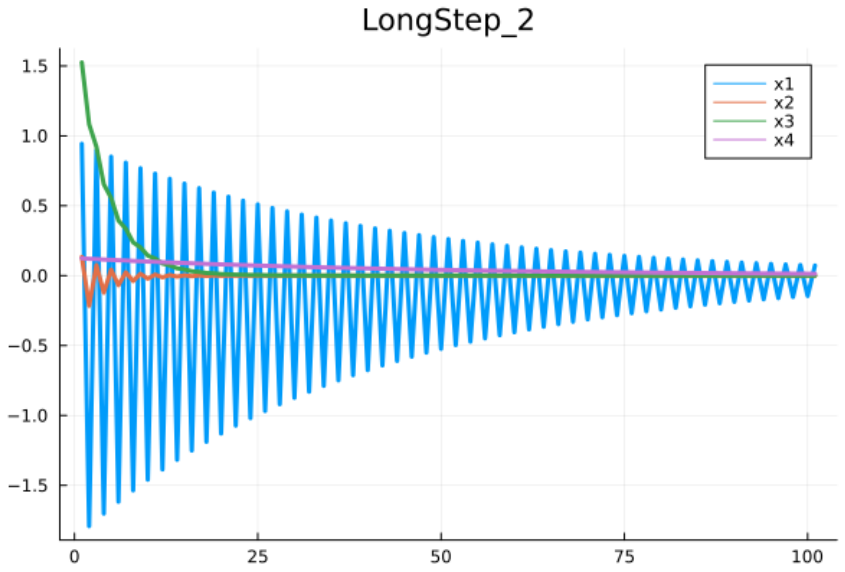
Caveat: $L(w_k)$ is constant and 2d quadratics are special. ("best case")

Multi-Step Analyses - Convex Functions







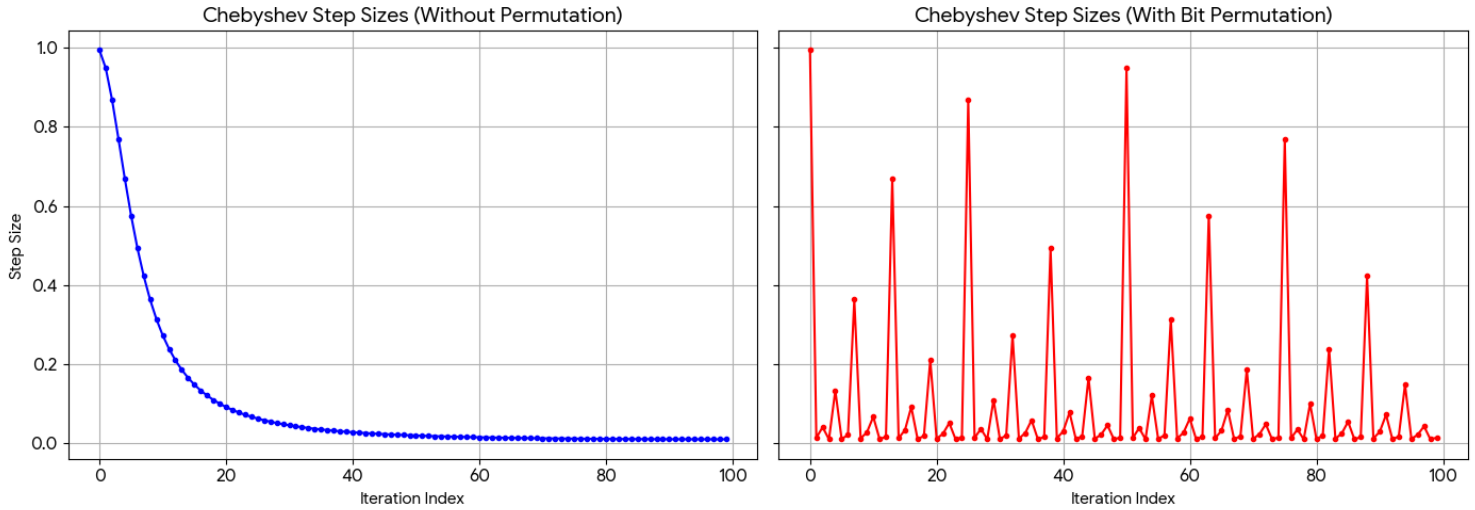


Multi-Step Analyses – Quadratic Functions

- Multi-Step analyses for quadratic functions:
  - Chebyshev step sizes if you know L, μ, and fixed horizon T:

$$\alpha_k = \frac{2}{(L+\mu) - (L-\mu)\cos\left(\frac{2k+1}{2T}\pi\right)}$$

    - Accelerated convergence rate (depends on $\sqrt{L/\mu}$ instead of L/μ).
    - Goes from $\approx 1/\mu$ to $\approx 1/L$.
      - Numerically unstable, need to re-order steps to keep under control.



[Young, 1953]

Multi-Step Analyses – Quadratic Functions

- Multi-Step analyses for quadratic functions:
  - Barzilai-Borwein (BB) step sizes:

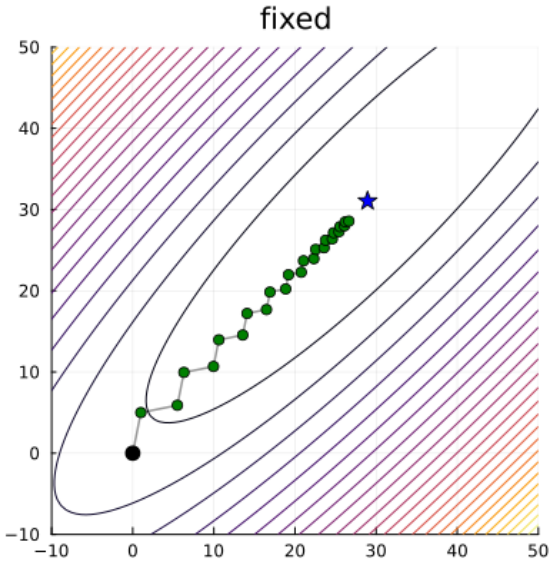
$$\alpha_k = \frac{\|w_k - w_{k-1}\|}{(w_k - w_{k-1})^T(\nabla f(w_k) - \nabla f(w_{k-1}))}$$

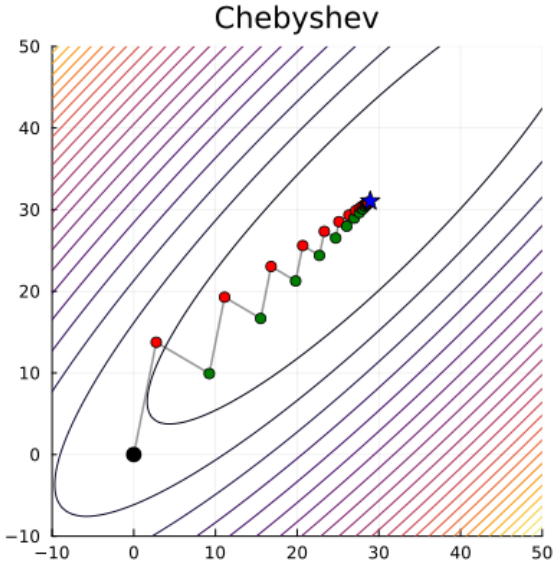
    - Approximates second derivative along last step.
      - Superlinear convergence rate (instead of linear) for two-dimensional quadratics.

  - "Spectral" step sizes if you know eigenvalues λ_j.
    - Set $\alpha_k = 1/\lambda_k$ where λ_k is the kth largest unique eigenvalue.
      - Solves problem after at most d steps (fully optimizes along one river/valley at each step).

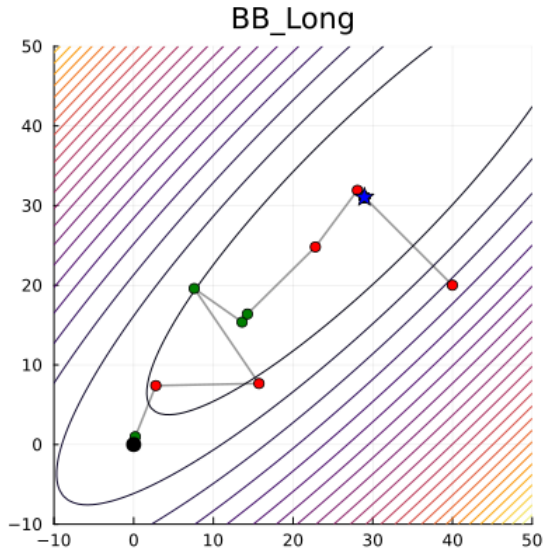
[Richardson, 1911, Barzilai & Borwein, 1988]

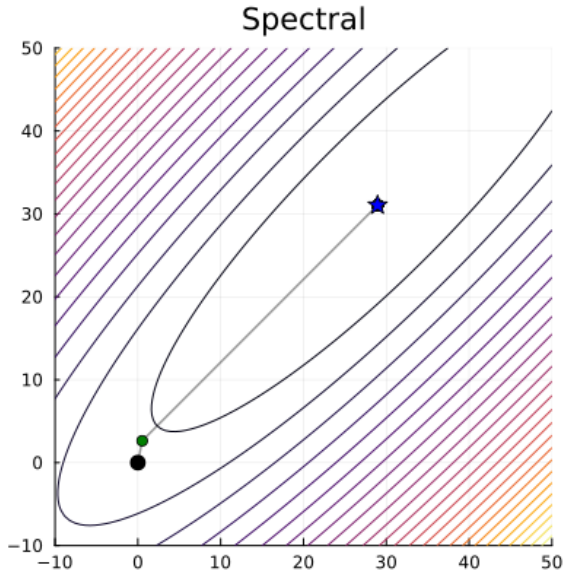
Multi-Step Analyses – Quadratic Functions





"Solved" after 14 steps.

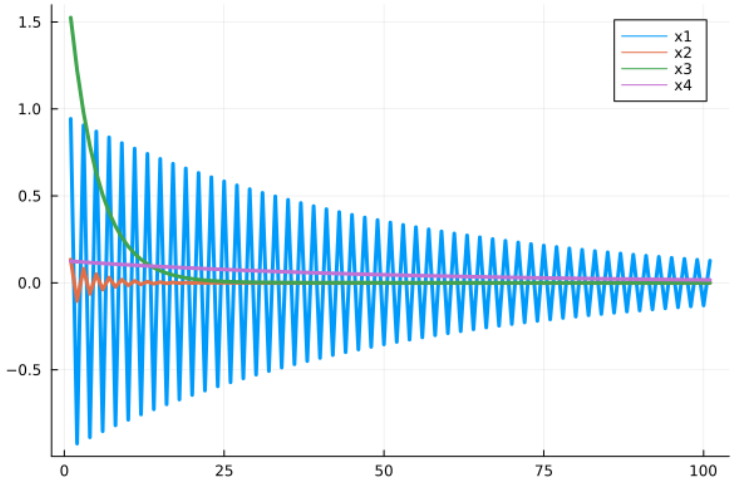


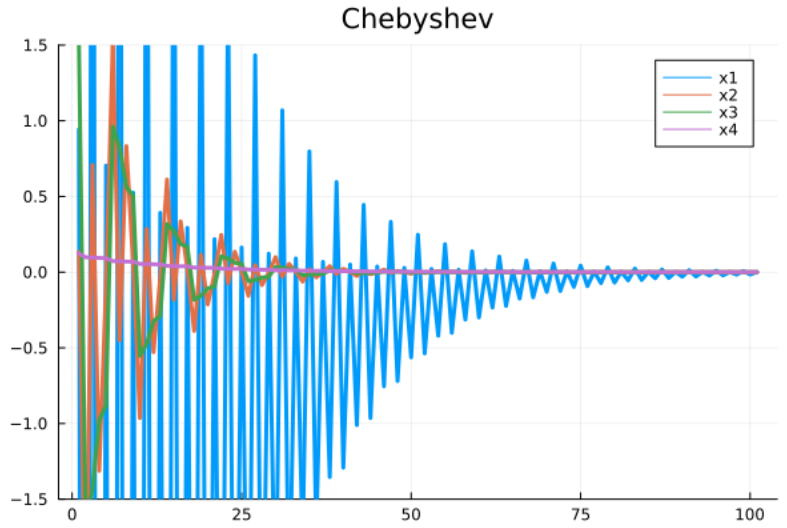


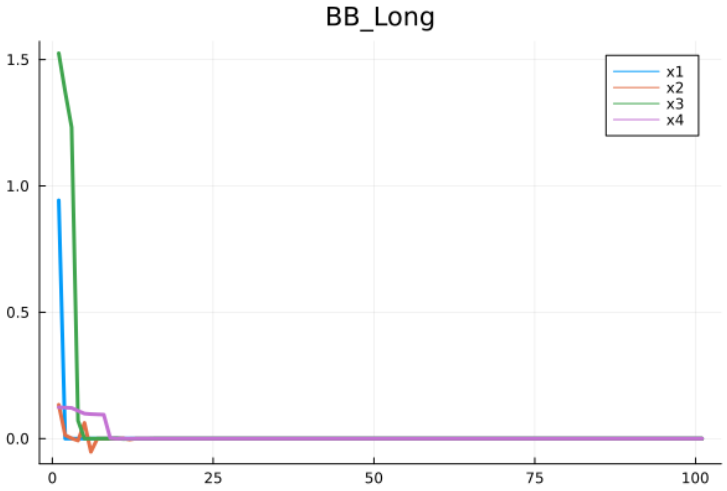
Solved after 2 steps.

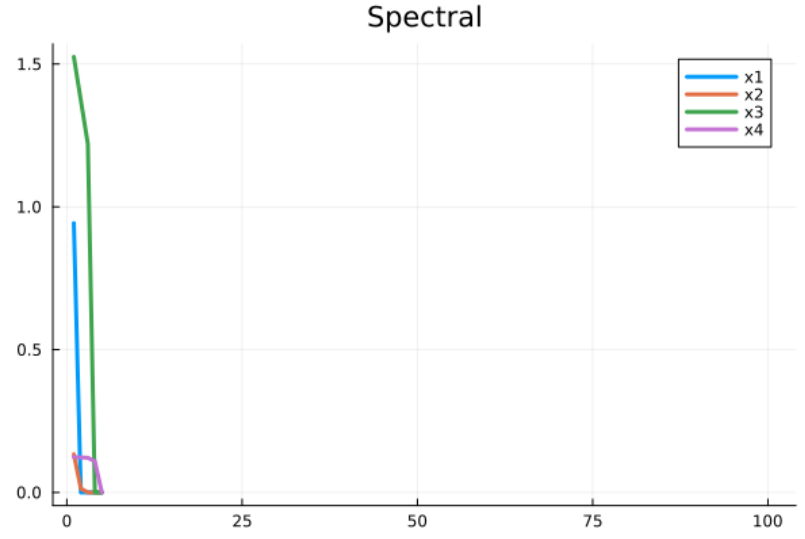
Multi-Step Analyses – Quadratic Functions

$\alpha_K = 2/(L+\mu)$









Multi-Step Practical Algorithms

- Step sizes for quadratics seem very specific to quadratics!
- But we have practical algorithms with related properties.
  - We will see that momentum is related to Chebyshev and Spectral sizes.
  - BB often works surprisingly well for non-quadratic functions.
    - Typically with non-monotonic line search (allows increases).
  - Original cosine annealing schedules have similarity to Silver/Chebyshev.

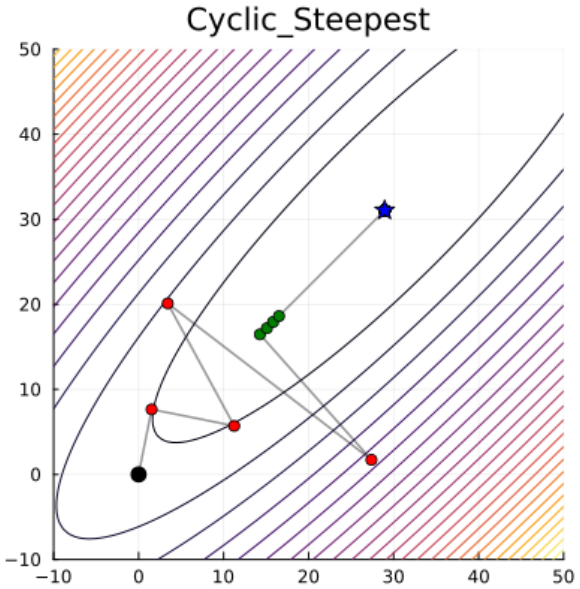
- But some practical approaches look very different:
  - "Try out every hyper-parameter and choose the one that works the best"

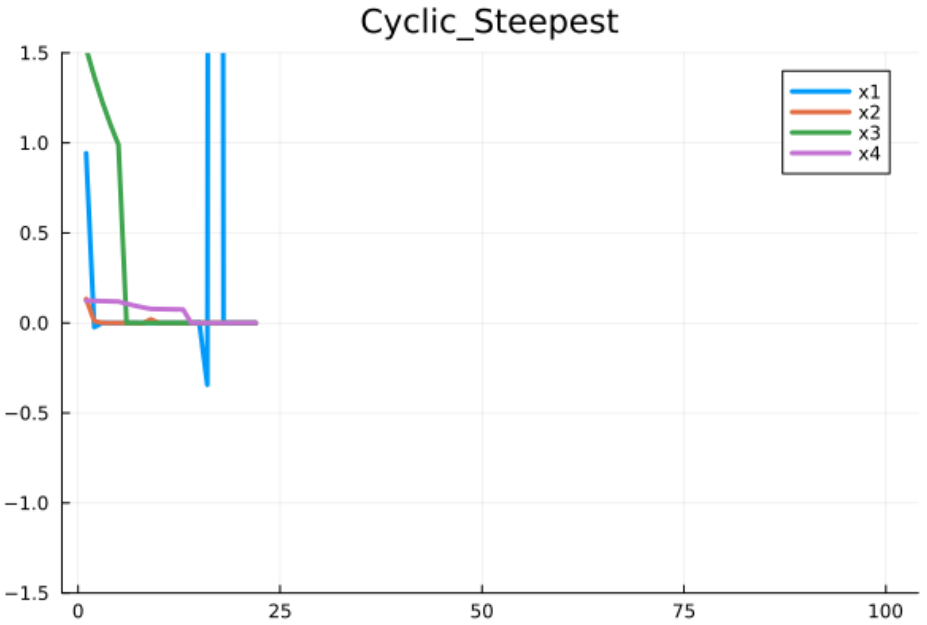
[Raydan, 1997]

Multi-Step Practical Algorithms

- Another practical multi-step algorithm is cyclic steepest descent:
  - Take [approximate] greedy step, then use it for t subsequent steps.
  - Often works surprisingly well and we do not know why.
    - Greedy adapts to local $L(w_k)$ and $\nabla f(w_k)$, cyclic part gives multi-step progress?

"Solved" after 14 steps.





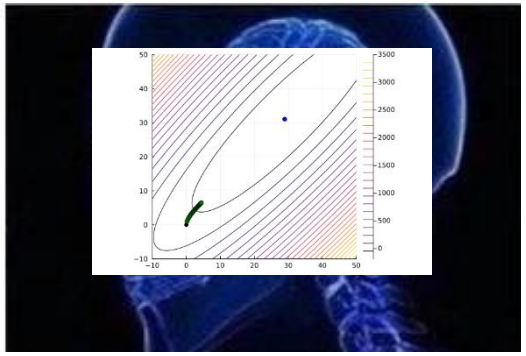
[Friedlander et al., 1995]

- Most theory lives on Level 1.
- Some theory on Level 2.
- Need more Level 3 and 4 theory!
- Practice lives on Level 4.

Level 1 Step sizes:

Need $\alpha_k < \frac{2}{L}$.

Optimal is $\alpha_k = \frac{1}{L}$.



AdaGrad-Norm
SUG

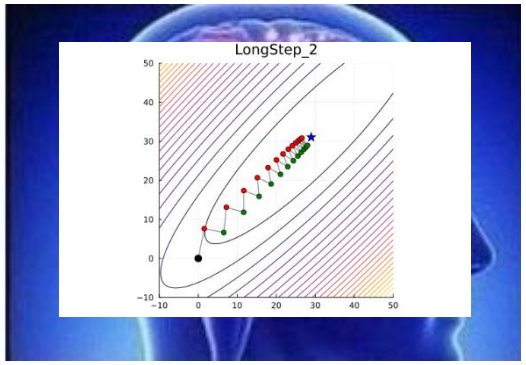
Level 2 Step sizes:

Optimal sequence is

$$\alpha_k = \frac{\gamma_k}{L}.$$

Optimal sequence is

$$\alpha_k = \frac{1}{L(w_k)}.$$

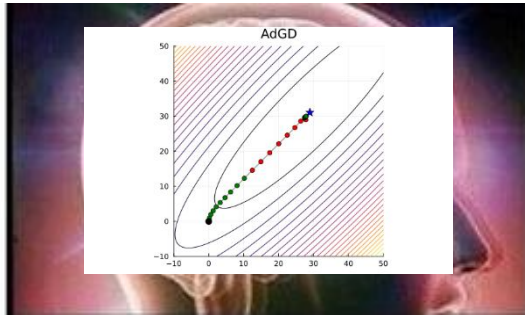


Silver step sizes
WarmUp

Level 3 Step sizes:

Optimal sequence is

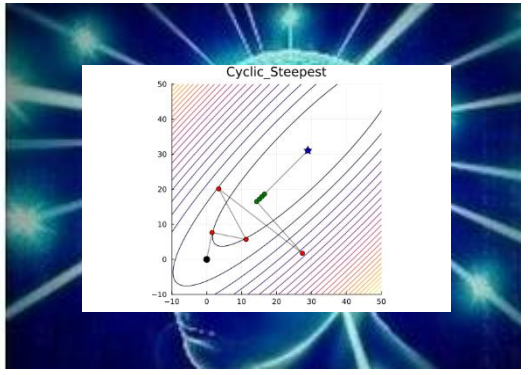
$$\alpha_k = \frac{1}{L(w_k, \nabla f(w_k), \nabla^2 f(w_k))}.$$



Greedy
AdGd

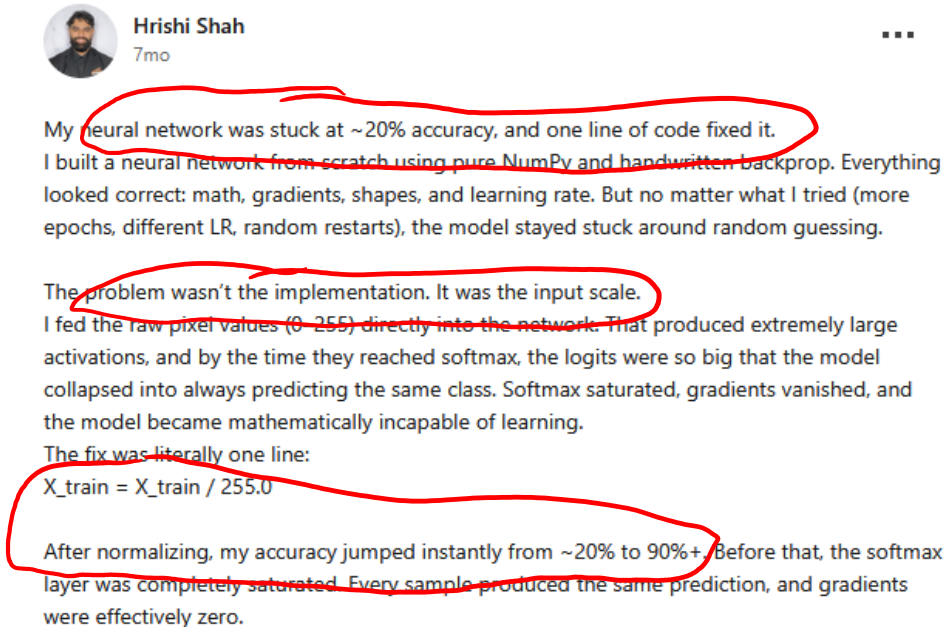
Level 4 Step sizes:

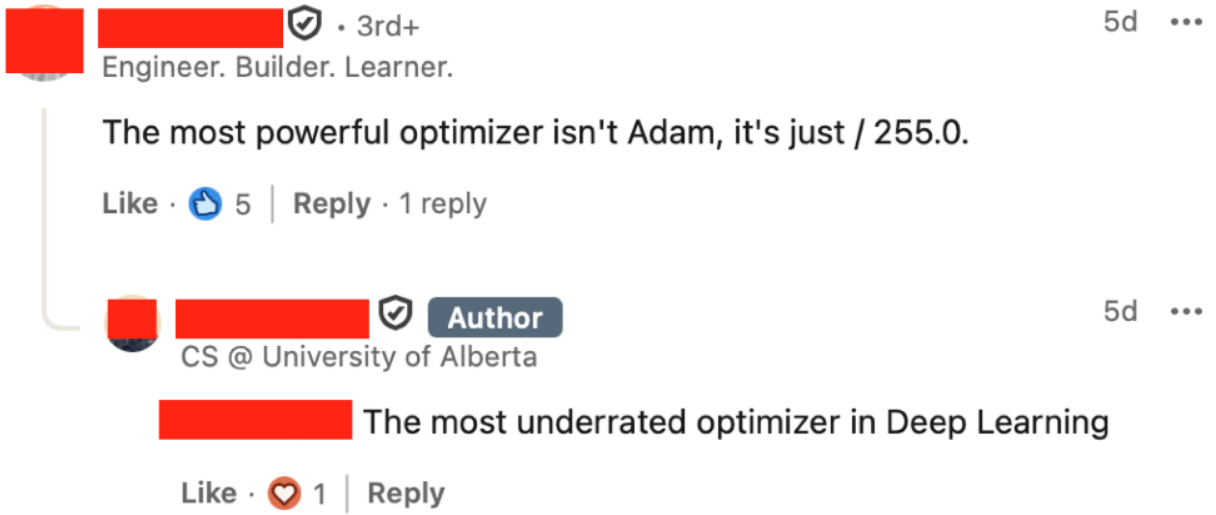
Optimal sequence optimizes $f(w_T)$.



WarmUp Stable Decay
Cyclic Steepest Descent
"Try out all hyper-parameters"

Before we get to SGD…





- Issues that have a huge influence but not viewed as "optimization":
  - Initialization.
  - Regularization.
  - Normalization.
- Especially important for neural networks.
  - Let's briefly discuss their influence in fixed feature case.

https://www.linkedin.com/posts/israfils_my-neural-network-was-stuck-at-20-accuracy-share-7395971530008006656-Tmw1

Non-Algorithm Trick #1: Initialization

- Ideal initialization is $w_0 = w_*$.
  - Simple view: "if $f(w_0)$ is smaller, we have less far to go."
- Other factors to consider:
  - For quadratics, ideally initialize in span of large eigenvectors.
    - "Start and then stay at the exact bottom of the valleys."
  - In general, more important to be further down ravines than valleys.
    - Not sure how to do this, but worth thinking about.
  - For non-convex, want to initialize near "good" solutions.
    - Example: k-means++ for initializing clustering algorithms.
- Could also consider potential progress:
  - Prefer $\|\nabla f(w_0)\|$ large and $L(w_0)$ small.
    - "Gradient should be large and slowly-changing " (see "level set teleportation").

[Arthur & Vassilvitskii, 2007, Mishkin et al., 2024]

Non-Algorithm Trick #2: Regularization

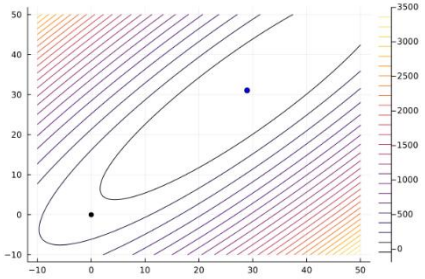
- Consider optimizing with L2-regularization:

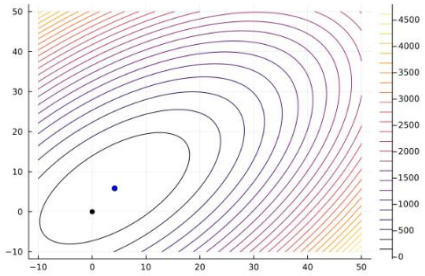
$$f(w) + \frac{\lambda}{2}\|w\|^2.$$

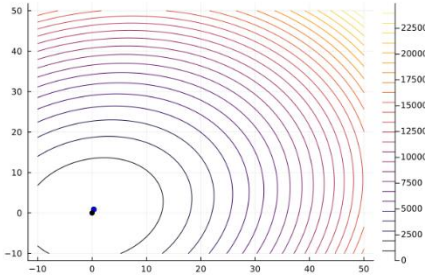
- Adds weight decay to the GD update:

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

- GD converges faster as we increase λ
  - More similar relative steepness between valleys and ravines ("all directions become good directions").

$\lambda = 0$ 

$\lambda = 1$ 

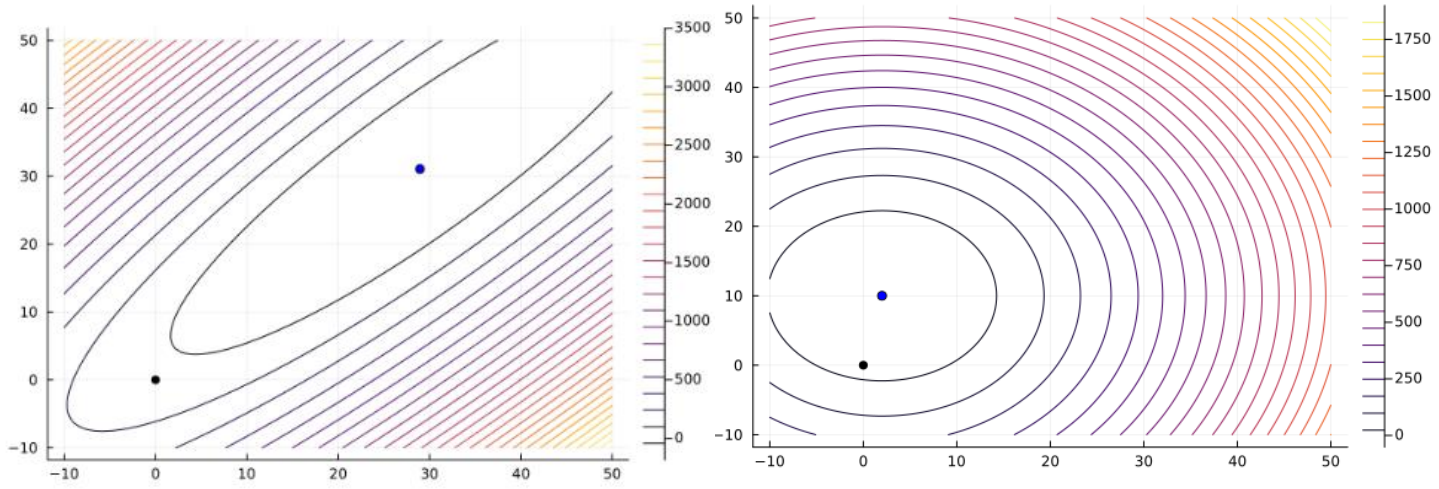
$\lambda = 10$ 

  - Can also decrease distance to solution and prevent overfitting.
    - For separable logistic regression, regularized solution is finite.
  - If f is convex then regularized version is PL (GD converges fast).

[Tikhonov, 1943]

Non-Algorithm Trick #3: Normalization

- Consider orthogonalizing feature matrix X in a linear model.
  - Compute QR decomposition, X=QR.
  - Use the orthogonal matrix Q as your features.
  - Generates identical predictions.
- Can dramatically improve optimization performance.
  - For least squares, GD with $\alpha_0 = 1$ solves problem in 1 step from any w_0.



Non-Algorithm Trick #3: Normalization

- Consider orthogonalizing feature matrix X in a linear model.
  - Compute QR decomposition, X=QR.
  - Use the orthogonal matrix Q as your features.
  - Generates identical predictions.
- Can dramatically improve optimization performance.
  - For least squares, GD with $\alpha_0 = 1$ solves problem in 1 step from any w_0.
- Other cheaper normalizations can improve optimization:
  - "Standardize features": subtract mean and divide by standard deviation.
  - Scaling raw pixel values from [0,255] down to [0,1].
- Alternately, non-linear feature transforms or kernels decrease f^*.
  - May make optimization problem harder, but solution better.

Question: How do I set the SGD Step Size?

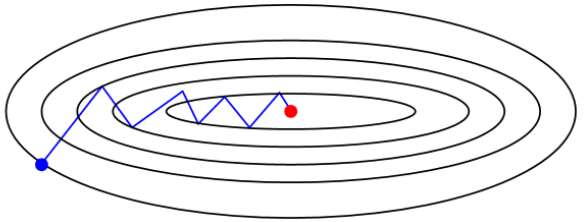
Deterministic and Stochastic Gradient Descent

- For minimizing functions of the form:

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

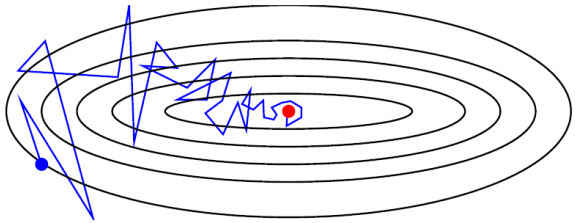
- Deterministic gradient descent repeats the iteration:

$$w_{k+1} = w_k - \alpha_k \frac{1}{n} \sum_{i=1}^{n} \nabla f_i(w_k)\,.$$



- Stochastic gradient descent (SGD) repeats the iteration:

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



  - For a random training example i_k (or average gradient on a batch of examples)
    - Key advantage is cost per iteration: 1 billion times faster for 1 billion examples.
      - If you budget is 2 billion tokens, would you rather do 2 GD steps or 2 billion SGD steps?

[Robbins & Monro, 1951]

Back to Level 1…

- We can analyze SGD progress using the (global) descent lemma:

$$f(v) \leq f(w_k) + \nabla f(w_k)^T (v - w_k) + \frac{L}{2} \|v - w_k\|^2$$

  - Plug in SGD update for v.
  - Take expectation with respect to i_k and assume α_k does not depend on i_k (for now).
  - Use that SGD is unbiased: $\mathbb{E}\left[\nabla f_{i_k}(w_k)\right] = \nabla f(w_k)$.
  - Define the "variance" of the gradients at w as:

$$\sigma^2(w) = \mathbb{E}[\|\nabla f_i(w) - \nabla f(w)\|^2]$$

- We get that expected function value satisfies:

$$\mathbb{E}[f(w_{k+1})] \leq f(w_k) - \alpha_k \left(1 - \frac{\alpha_k L}{2}\right) \|\nabla f(w_k)\|^2 + \frac{\alpha_k^2 L}{2} \sigma(w_k)^2 .$$

  - Decrease of gradient descent, but slowed down by variance of gradients.

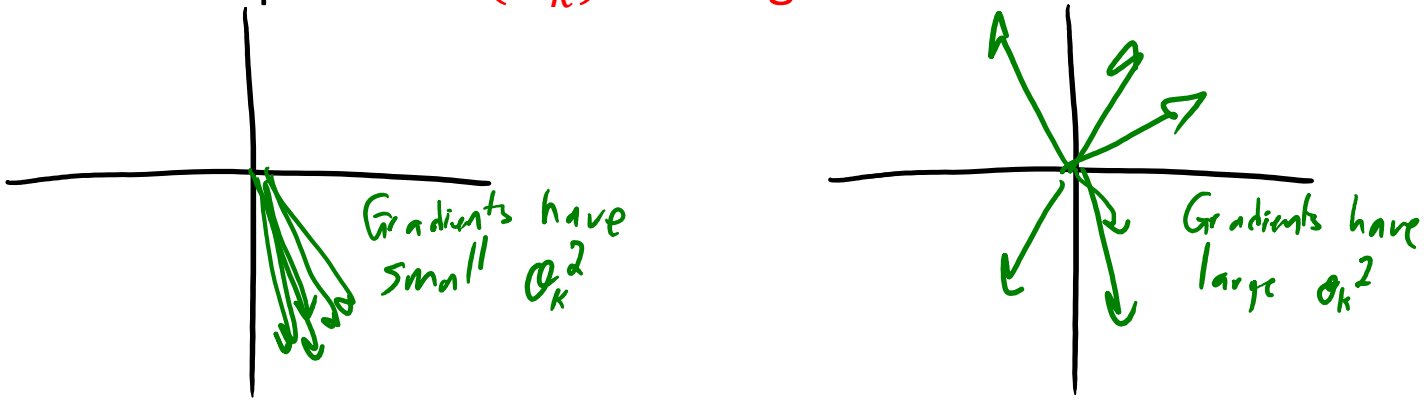
[Ghadimi & Lan, 2013]

How do I set the SGD step size?

- SGD still guarantees decrease in expectation if:

$$\alpha_k < \frac{2}{L}\left(\frac{1}{1+\sigma(w_k)^2/\|\nabla f(w_k)\|^2}\right)$$

  - We have our usual $2/L$ upper bound (red factor is ≤ 1).
    - Cannot go faster than GD and expect decrease.
  - But we now have $\sigma(w_k)^2$ measuring variation in gradients at w_k.
    - Need smaller step size if $\sigma(w_k)^2$ is large.



How do I set the SGD step size?

- SGD still guarantees decrease in expectation if:

$$\alpha_k < \frac{2}{L}\left(\frac{1}{1+\sigma(w_k)^2/\|\nabla f(w_k)\|^2}\right)$$

  - We have our usual $2/L$ upper bound (red factor is ≤ 1).
    - Cannot go faster than GD and expect decrease.
  - But we now have $\sigma(w_k)^2$ measuring variation in gradients at w_k.
    - Need smaller step size if $\sigma(w_k)^2$ is large.
  - We now also have overall gradient size $\|\nabla f(w_k)\|^2$ at w_k.
    - Big gradient reduces the effect of noise.

How do I set the SGD step size?

- SGD still guarantees decrease in expectation if:

$$\alpha_k < \frac{2}{L}\left(\frac{1}{1+\sigma(w_k)^2/\|\nabla f(w_k)\|^2}\right)$$

- How much we need to decrease α_k depends on ratio: $\frac{\sigma(w_k)^2}{\|\nabla f(w_k)\|^2}$.
  - If $\|\nabla f(w_k)\|^2 > \sigma(w_k)^2$, sufficient to have $\alpha_k \leq 1/L$.
    - If gradient is bigger than noise, acts like deterministic GD.
  - If $\sigma(w_k)^2$ >> $\|\nabla f(w_k)\|^2$, we roughly need $\alpha_k < 2\|\nabla f(w_k)\|^2/L\sigma(w_k)^2$.
    - If noise is huge compared to gradient, need to use much smaller step sizes.

Gradient-Dominated vs. Noise-Dominated Phase

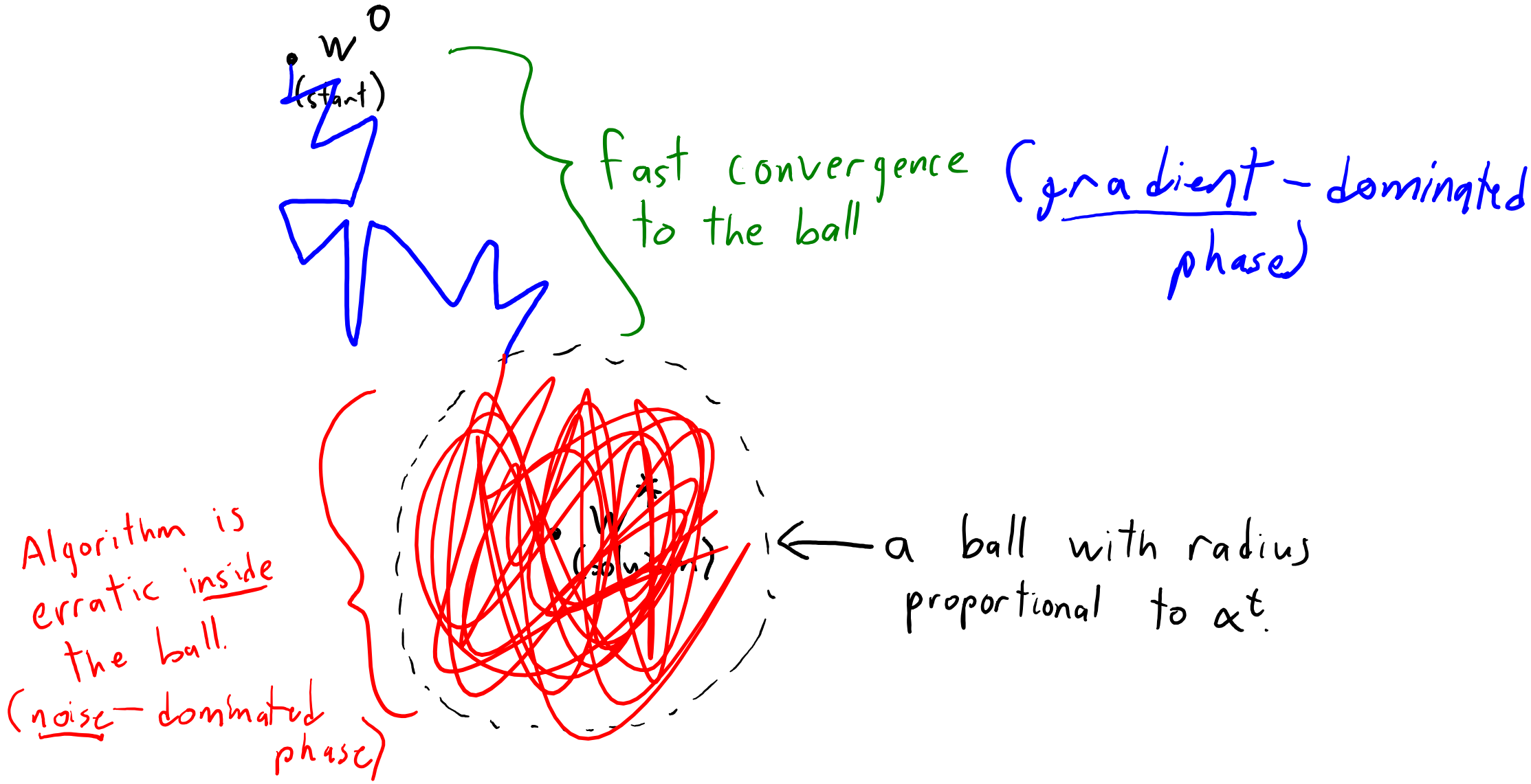
- For small α_k, we roughly make progress while:

$$\|\nabla f(w_k)\|^2 > O(\alpha_k \sigma(w_k)^2)$$

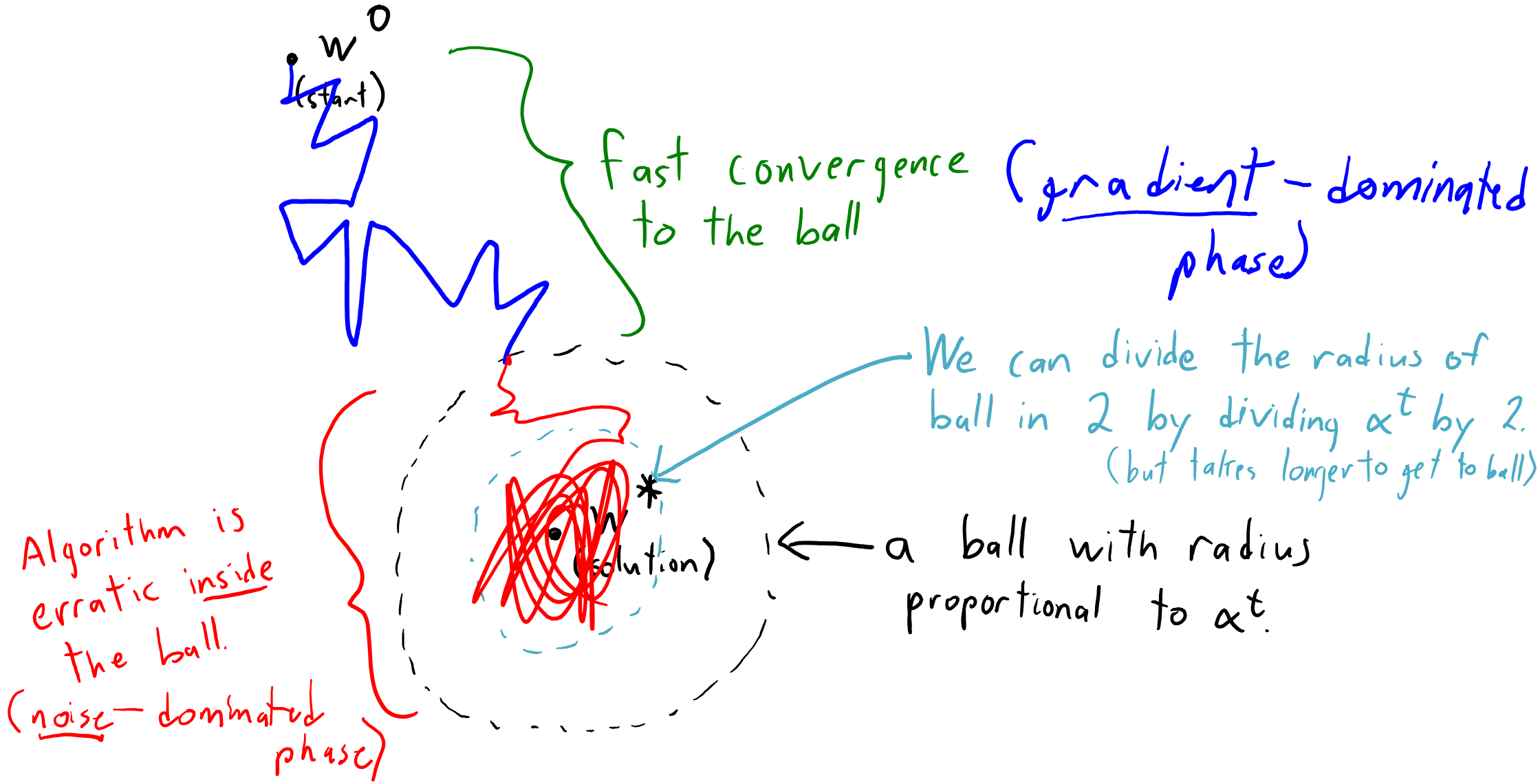
- Consider sequence w_k where $f(w_k)$ approaches f_*:
  - We have that $\|\nabla f(w_k)\|$ decreases to 0.
  - But $\sigma^2(w_k)$ may not decrease to 0.

- Often leads to two "phases" when we use a fixed step-size:
  1. Gradient dominated: consistent progress when $\|\nabla f(w_k)\|$ dominates.
  2. Noise dominated: erratic behaviour when $\alpha_k \sigma^2(w_k)$dominates.

Behaviour of SGD with Constant Step Size



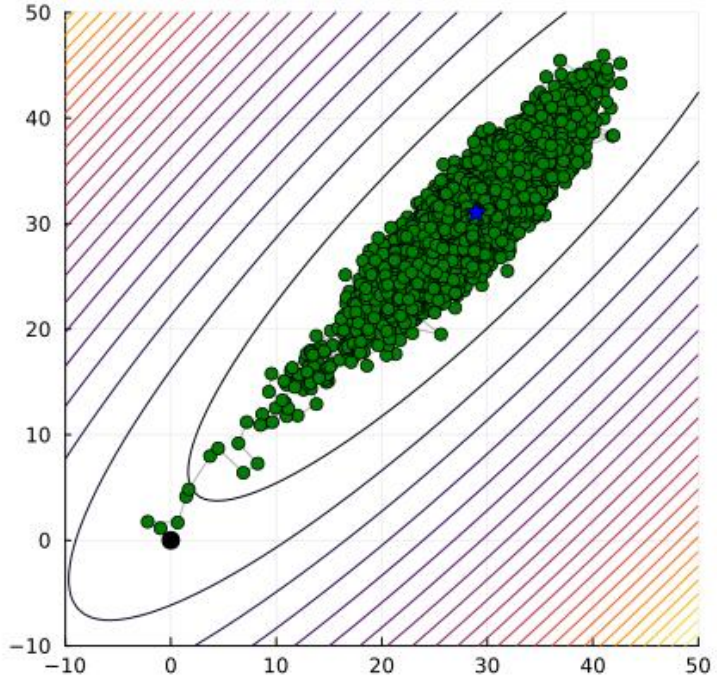
Behaviour of SGD with Constant Step Size



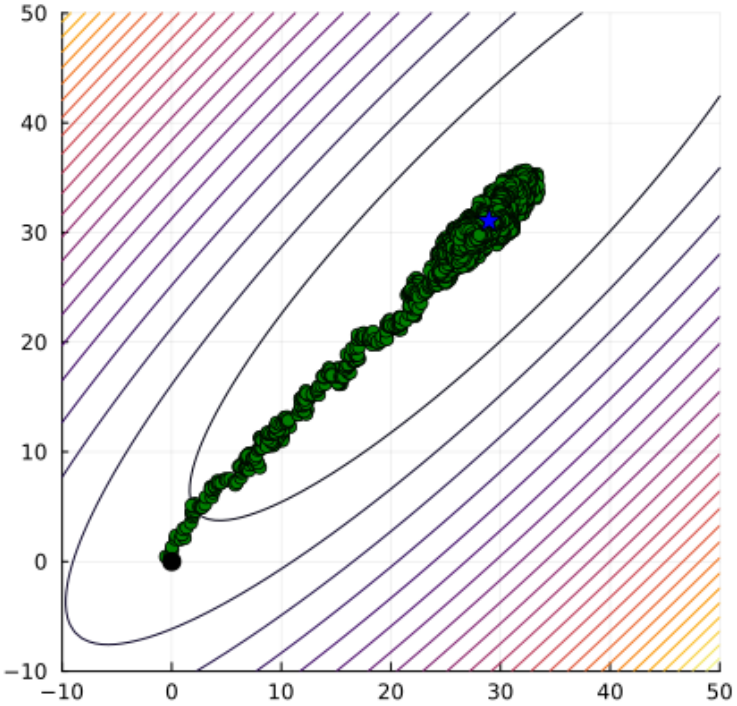
Behaviour of SGD with Constant Step Size

- 10,000 SGD steps with constant step size:

$\alpha_k = 1/2L$



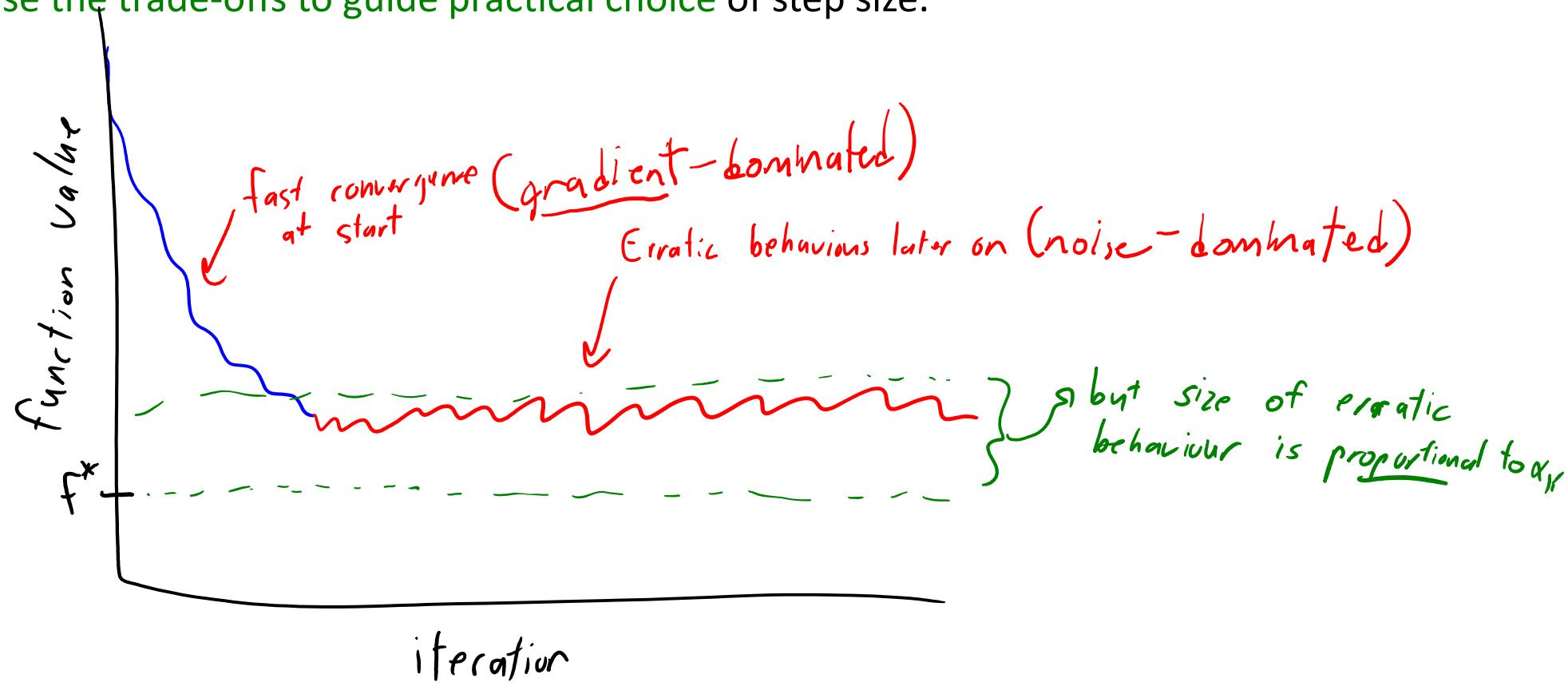
$\alpha_k = 1/10L$



– With smaller α_k: smaller erratic region but takes longer to get there.

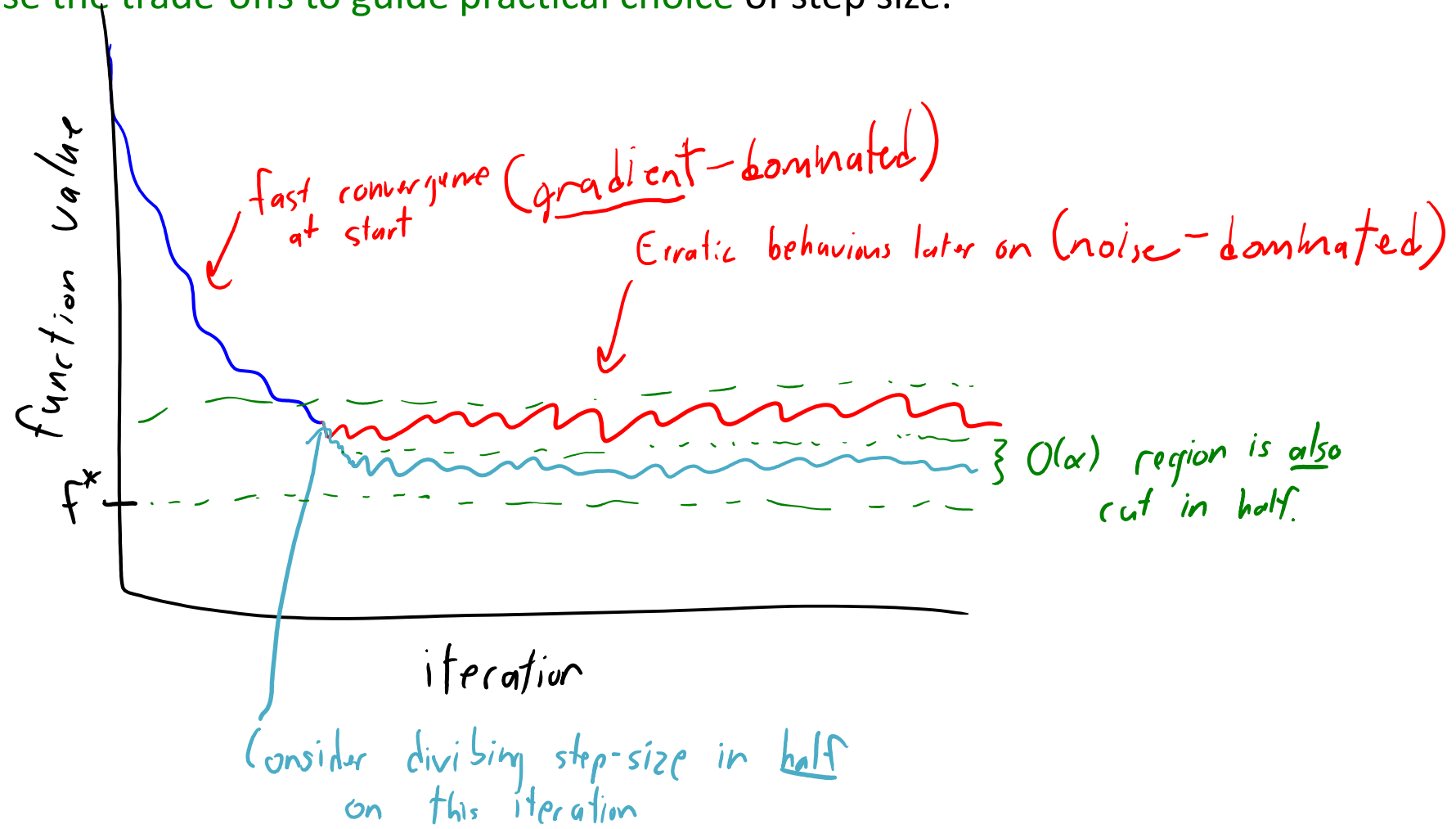
How do I set the SGD step size?

- In practice, SGD algorithm does not have access to L, $\nabla f(w_k)$ or σ_k.
  - But we can use the trade-offs to guide practical choice of step size.



How do I set the SGD step size?

- In practice, SGD algorithm does not have access to L, $\nabla f(w_k)$ or σ_k.
  - But we can use the trade-offs to guide practical choice of step size.



Textbook Step Sizes

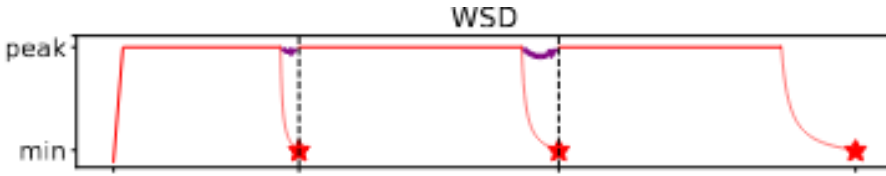
- Classic analyses of SGD assume strictly decreasing step sizes:

$$\alpha_k = \frac{\gamma}{k} \qquad\qquad \alpha_k = \frac{\gamma}{\sqrt{k}}$$

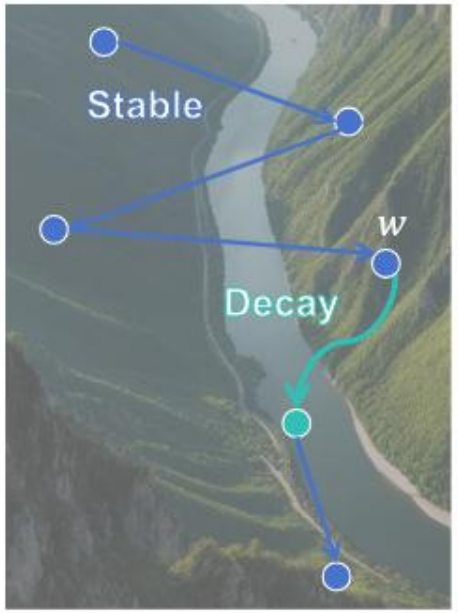
  - But this usually works terribly: ignores gradient-dominated phase.
- Better step strategy:
  - Use constant α_k while in gradient-dominated phase.
    - Example: can use $\alpha_k = \frac{1}{L}$ while $\|\nabla f(w_k)\| > \sigma(w_k)^2$.
  - Then switch to your decreasing α_k.
    - Or alternately switch to a smaller constant.
- Challenge: don't know when to switch.

Back to ~~Warmup~~ Stable Decay

- WarmUp Stable Decay (without warmup):
  - Use constant α_k while in gradient-dominated phase.
  - Occasionally try to switch to your decreasing α_k.
    - But go back to constant step size Checkpoint.



- Simple solution to complex problem:
  - Don't know when to stop going up valleys.
  - Don't know when to start reducing effect of noise.
  - Treat switching time as a hyper-parameter.
    - But don't re-run from scratch with different "switch" times.



(a) River Valley Landscape

- WSD new but it's my favourite SGD step size idea!
  - Practically-motivated but elegantly deals with unknown theoretical dynamics.

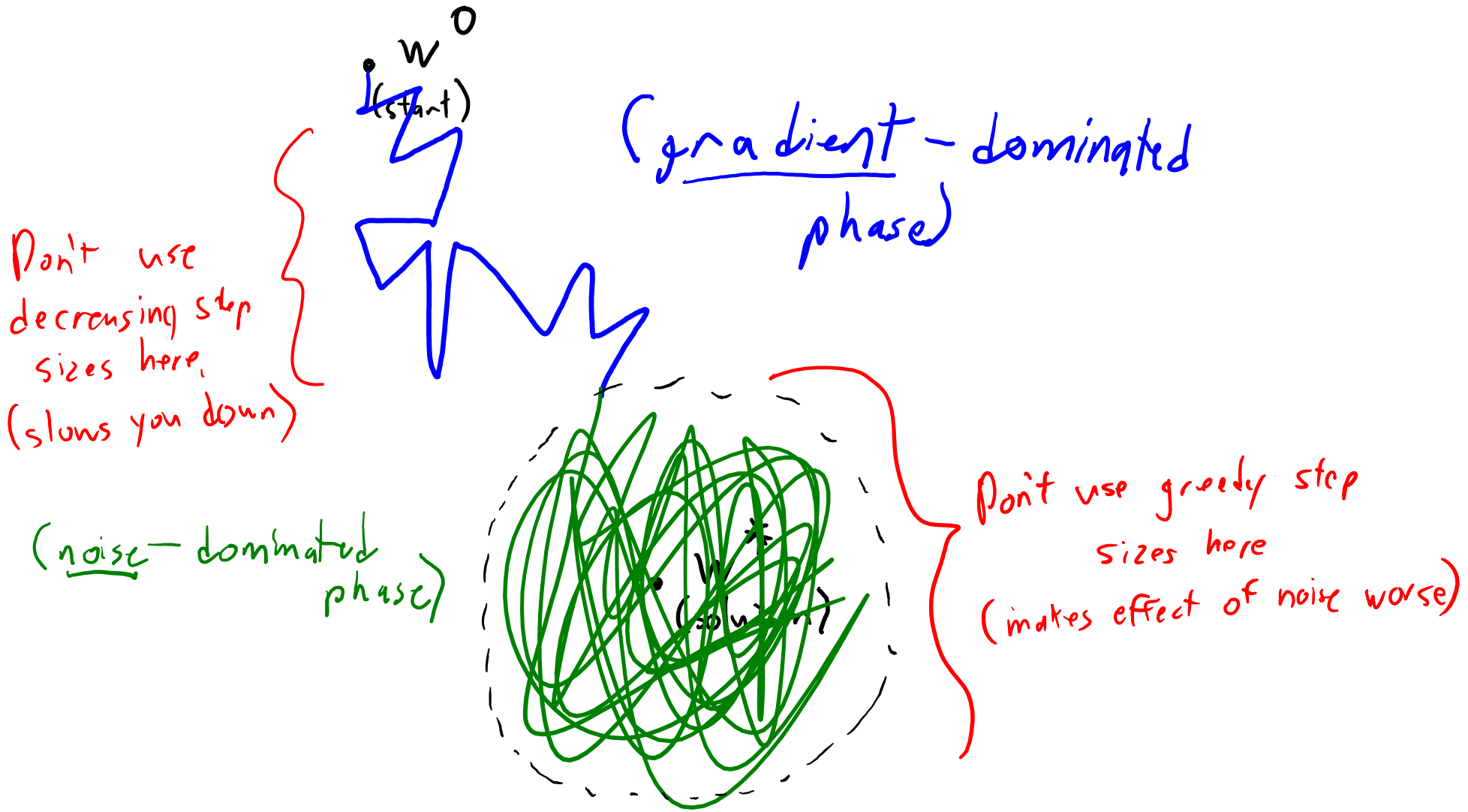
Question: how do we speed up SGD?

How to Speed Up SGD?

- Need different tricks in the two phases:
  - Speed up gradient-dominated phase by exploiting curvature:
    - Example: use a greedy step size.
  - Speed up noise-dominated phase by reducing effect of noise:
    - Example: using decreasing step sizes.

- Tricks that help in one setting may hurt the other!
  - Decreasing step sizes in gradient-dominated setting slows you down.
  - Greedy step sizes in noise-dominated can amplify noise.

Behaviour of SGD with Constant Step Size



Map of Speeding up SGD (Outline of Next Parts).

- Gradient-dominated:
  - Large step sizes.
  - Momentum.
  - Adam and friends.
  - Preconditioning.

- Noise-dominated:
  - Decreasing step sizes.
  - Interpolation.
  - Averaging.
  - Mini-batching.

Map of Speeding up SGD (Outline of Next Parts).

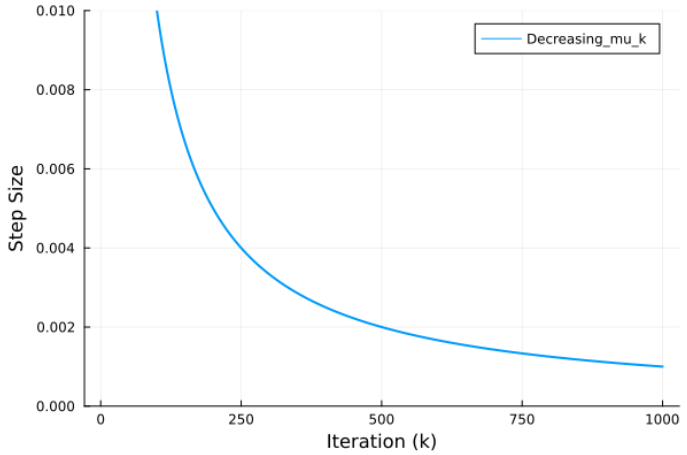
- Gradient-dominated:
  - Large step sizes.
  - Momentum.
  - Adam and friends.
  - Preconditioning.
- Noise-dominated:
  - Decreasing step sizes.
  - Interpolation.
  - Averaging.
  - Mini-batching.

Finite datasets tricks:

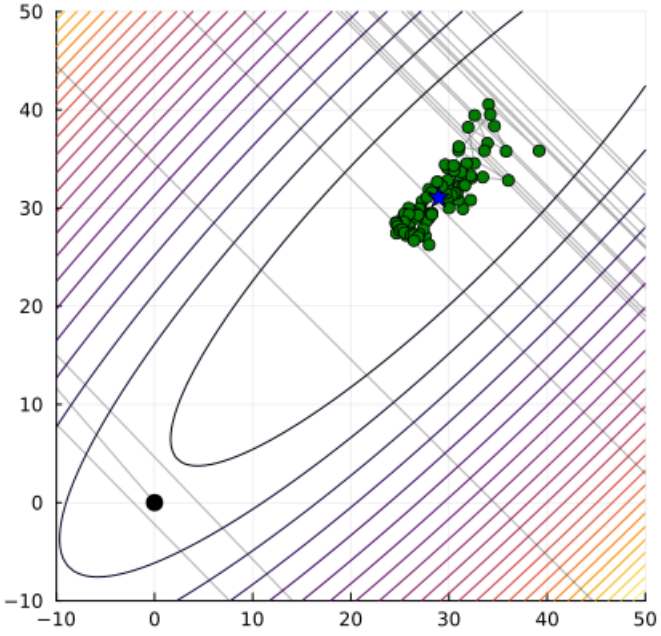
- Importance sampling.
- Random reshuffling.
- Variance reduction.

Step Size Schedules

- Textbook step sizes: $\alpha_k = \alpha/k$.
  - Optimal on PL functions (if you know μ).
    - Terrible for harder problems.
  - Terrible in practice (except binary SVMs).



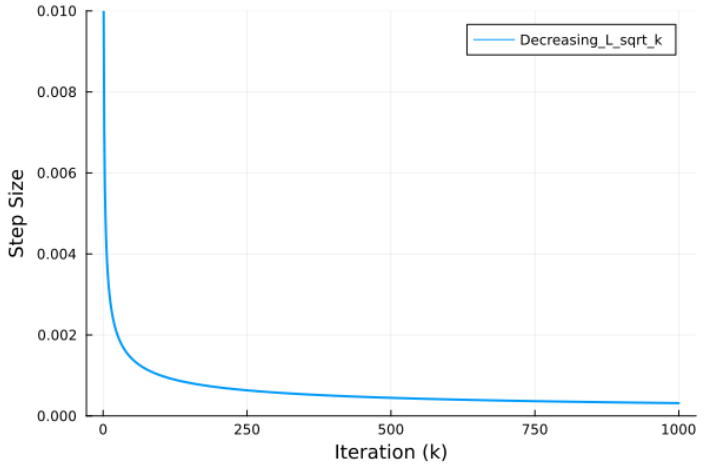
$\alpha_k = 1/\mu k$



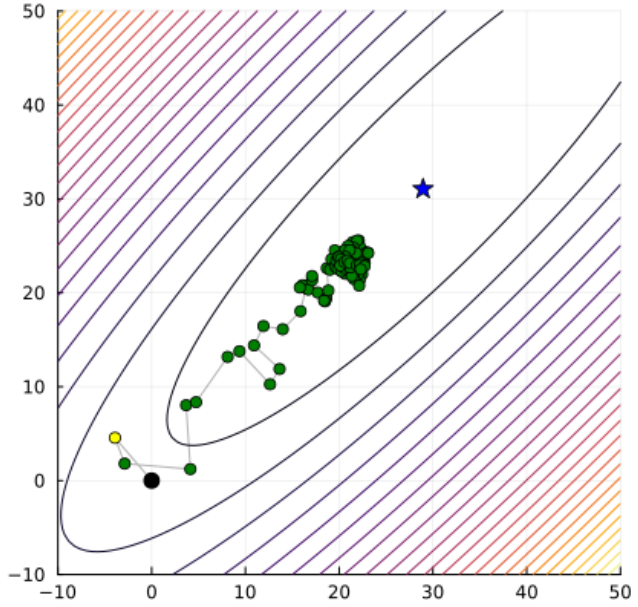
[Robbins & Monro, 1951, Chung, 1954, Shalev-Shwartz et al., 2007]

Step Size Schedules

- Textbook step sizes: $\alpha_k = \alpha/k$.
- Robust step sizes: $\alpha_k = \alpha/\sqrt{k}$.
  - Same speed across many settings.
  - But that speed is not fast (worst case).



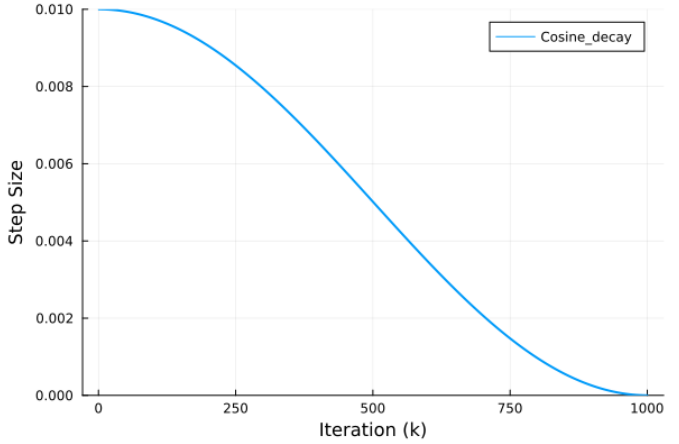
$\alpha_k = 2/L\sqrt{k}$



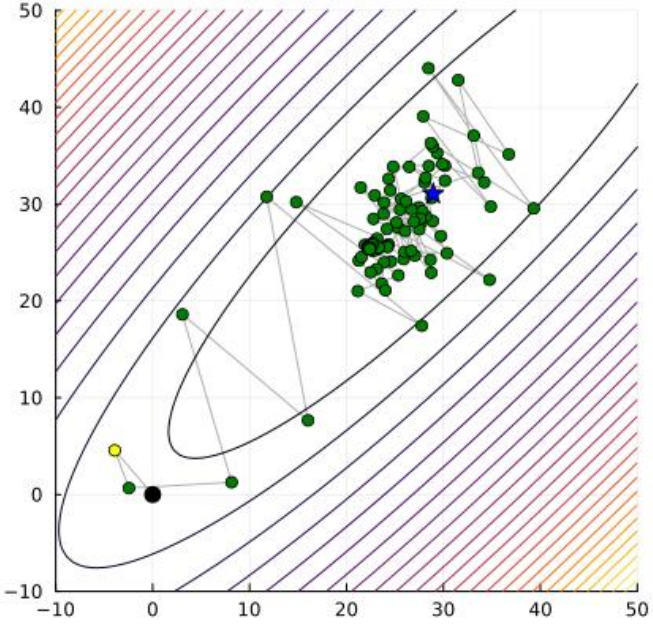
[Ruppert, 1988, Polyak & Juditsky, 1992, Nemirovski et al., 2009]

Step Size Schedules

- Textbook step sizes: $\alpha_k = \alpha/k$.
- Robust step sizes: $\alpha_k = \alpha/\sqrt{k}$.
- Cosine annealing: $\alpha_k = \frac{1}{2}\alpha\left(1 + \cos\left(\frac{k}{T}\right)\right)$.
  - Years of empirical success.
  - Needs horizon T (cannot “run longer”).



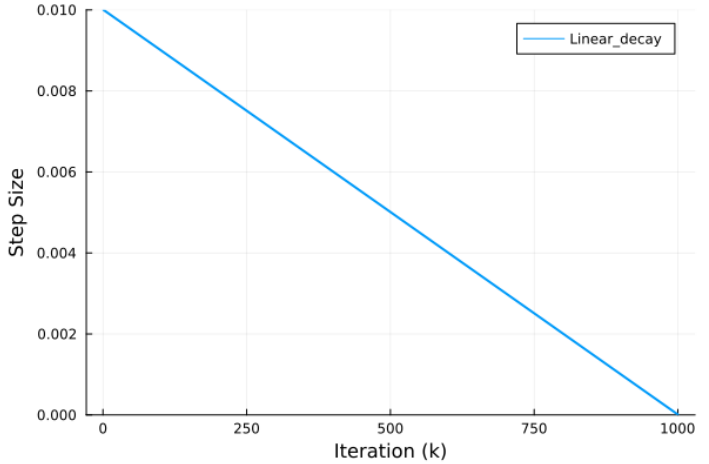
$\alpha_0 = 2/L$



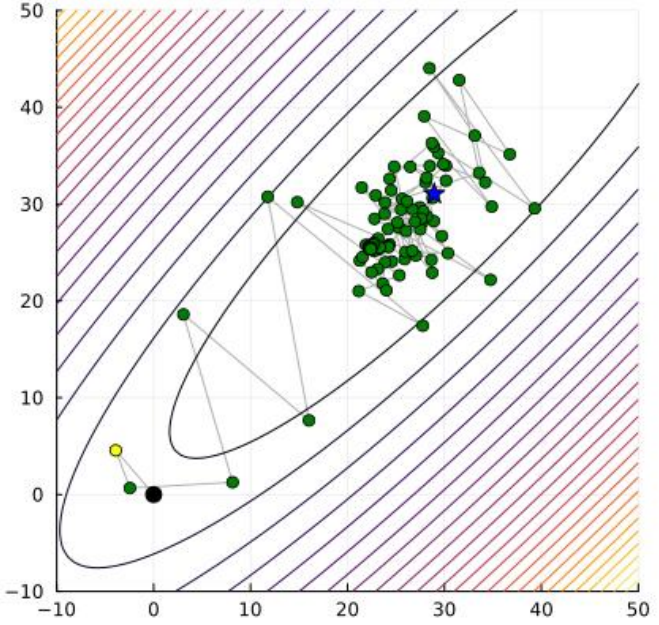
[Loshchilov & Hutter, 2016]

Step Size Schedules

- Textbook step sizes: $\alpha_k = \alpha/k$.
- Robust step sizes: $\alpha_k = \alpha/\sqrt{k}$.
- Cosine annealing: $\alpha_k = \frac{1}{2}\alpha\left(1 + \cos\left(\frac{k}{T}\right)\right)$.
- Linear decay: $\alpha_k = \alpha\left(1 - \frac{k}{T}\right)$.
  - Similar to cosine annealing.
  - Last iterate gets rate of robust step size.



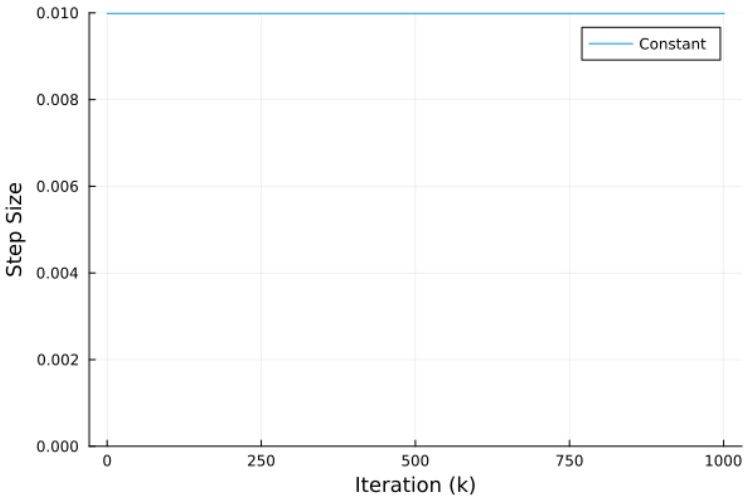
$\alpha_0 = 2/L$



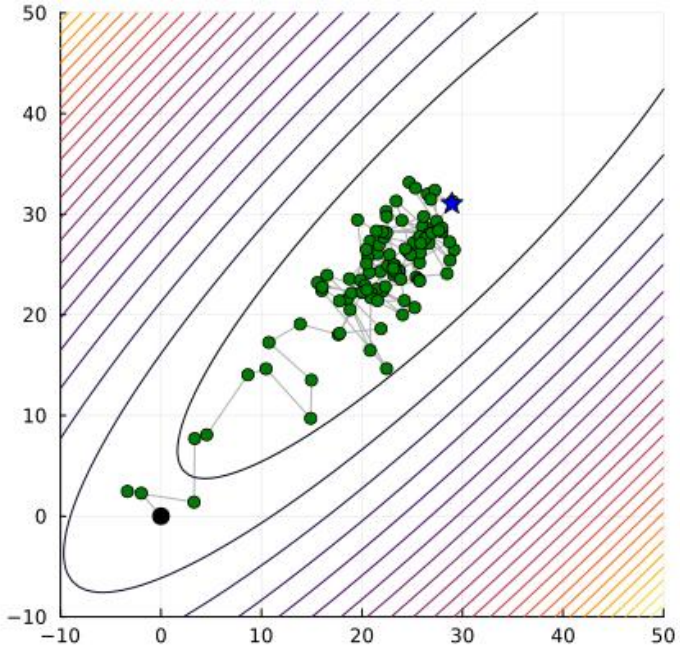
[Defazio et al., 2023]

Step Size Schedules

- Textbook step sizes: $\alpha_k = \alpha/k$.
- Robust step sizes: $\alpha_k = \alpha/\sqrt{k}$.
- Cosine annealing: $\alpha_k = \frac{1}{2}\alpha\left(1 + \cos\left(\frac{k}{T}\right)\right)$.
- Linear decay: $\alpha_k = \alpha\left(1 - \frac{k}{T}\right)$.
- Constant step sizes: $\alpha_k = \alpha$.
  - Fast GD speed at start.
  - Erratic when noise-dominated.
  - Using $\alpha = \alpha_0/\sqrt{T}$ gets robust rate.



$\alpha_k = 1/L$



[Widrow & Hoff, 1960, Nemirovski et al., 2009]

Digression: do we need "convergence"?

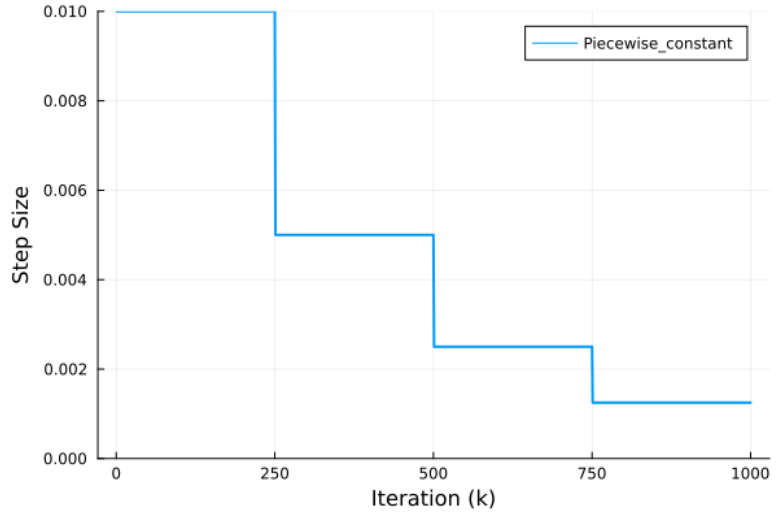
- Optimization: we want $\nabla f(w_k)$ to converge to 0 as k grows.
  - So we need to use decreasing step sizes to guarantee continued progress.
    - But as we decrease the step size SGD will converge slower.

- Machine learning: we only need $\nabla f(w_k)$ to close to 0.
  - We expect test error to be similar for all w_k "good enough".
    - If measuring test error to 2 decimals, may not need 10 decimal places of accuracy.
  - For any "good enough", there is a small-enough constant step size $\alpha_k = \alpha$.
    - Guarantees progress until $\nabla f(w_k)$ is "close enough" to zero.

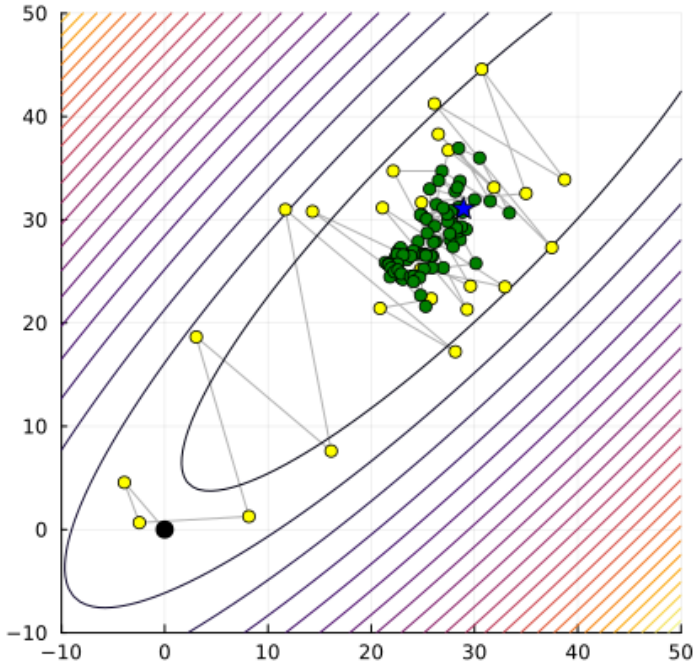
Step Size Schedules

- Textbook step sizes: $\alpha_k = \alpha/k$.
- Robust step sizes: $\alpha_k = \alpha/\sqrt{k}$.
- Cosine annealing: $\alpha_k = \frac{1}{2}\alpha\left(1 + \cos\left(\frac{k}{T}\right)\right)$.
- Linear decay: $\alpha_k = \alpha\left(1 - \frac{k}{T}\right)$.
- Constant step sizes: $\alpha_k = \alpha$.
- Piecewise-constant.
  - Constant with occasional decreases.
  - Repeatedly gets back to GD speed.
  - More hyper-parameters.

[Kesten, 1958, Ge et al., 2019]

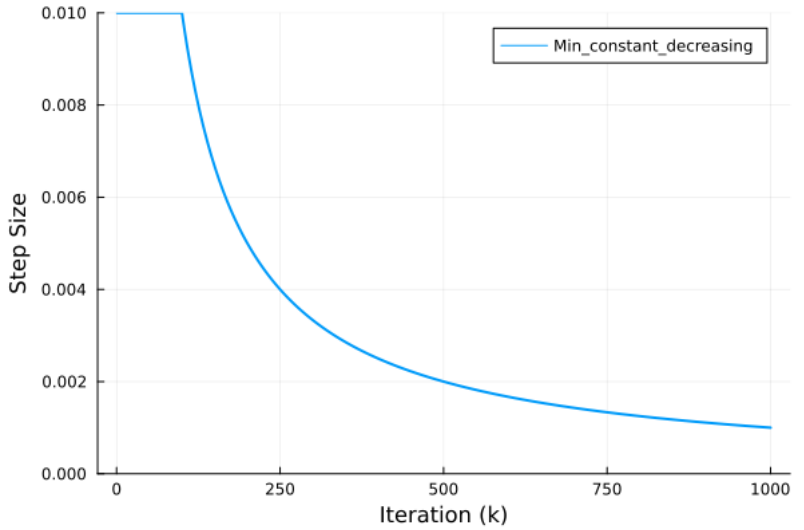


$\alpha_0 = 2/L$ (divide by 2 every 25)

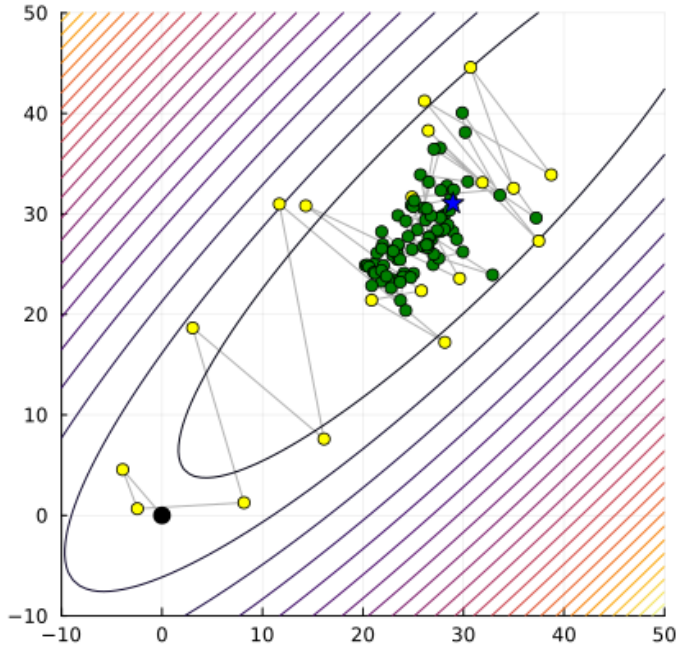


Step Size Schedules

- Textbook step sizes: $\alpha_k = \alpha/k$.
- Robust step sizes: $\alpha_k = \alpha/\sqrt{k}$.
- Cosine annealing: $\alpha_k = \frac{1}{2}\alpha\left(1 + \cos\left(\frac{k}{T}\right)\right)$.
- Linear decay: $\alpha_k = \alpha\left(1 - \frac{k}{T}\right)$.
- Constant step sizes: $\alpha_k = \alpha$.
- Piecewise-constant.
- Constant then decrease
  - Fast GD speed at start, hyper-parameters.
  - Converges (slow "schedule" speed at end).

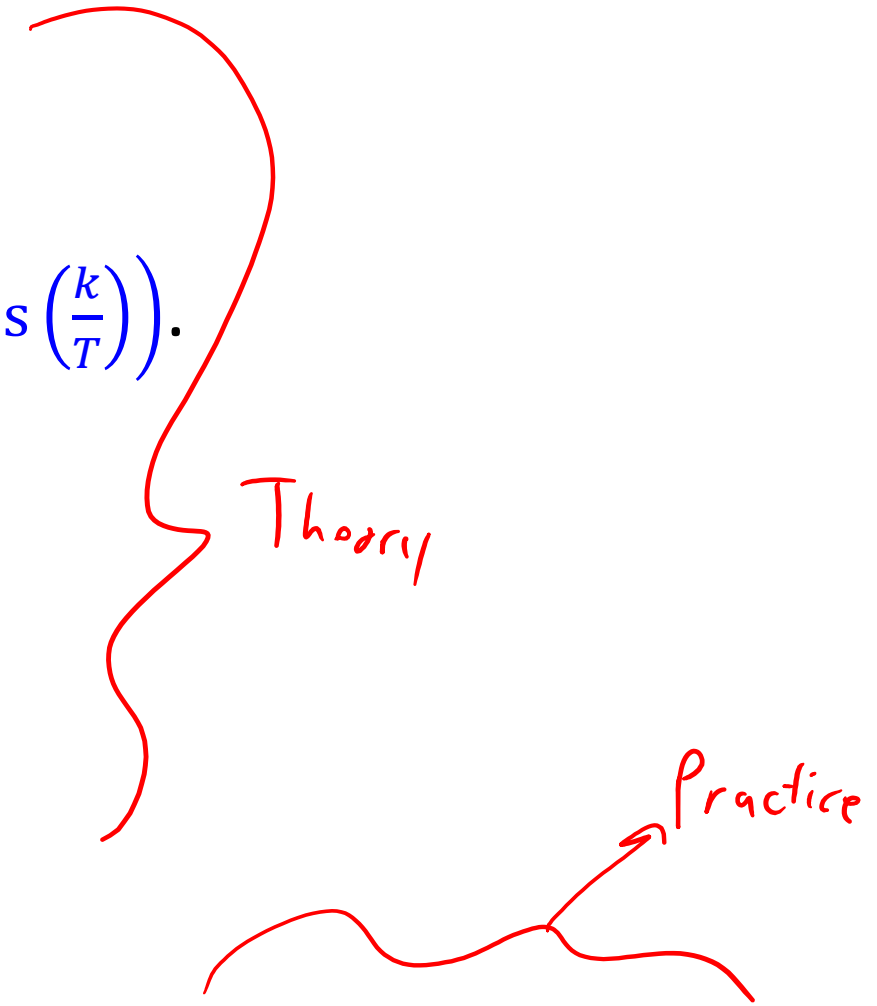


$\alpha_0 = \min\{2/L, 1/\mu k\}$



Step Size Schedules

- Textbook step sizes: $\alpha_k = \alpha/k$.
- Robust step sizes: $\alpha_k = \alpha/\sqrt{k}$.
- Cosine annealing: $\alpha_k = \frac{1}{2}\alpha\left(1 + \cos\left(\frac{k}{T}\right)\right)$.
- Linear decay: $\alpha_k = \alpha\left(1 - \frac{k}{T}\right)$.
- Constant step sizes: $\alpha_k = \alpha$.
- Piecewise-constant.
- Constant then decrease.
- None of these work in practice.
  - What works? Pick one of the above and tune hyper-parameter(s).



Level 2 Step Sizes?

- I would describe previous schedules as "Level 1".
  - Take a single iteration progress bound, recurse/sum over T iterations.
    - Do not use bound on progress across multiple iterations.
    - Do not take into account we could have "flat" local $L(w_k)$ or "less noisy" $\sigma(w_k)^2$.
    - Tuning hyper-parameters makes "Level 1" step sizes act like "Level 2+" (expensive).
- Are there "Level 2+" step sizes for SGD?
  - Yes, almost 70 years of attempts!
- Kesten [1958] for one-variable problems:
  - Check $\text{sign}(\nabla f(w_k))$:
    - If it stayed the same, keep the step size.
    - If it changed, decrease the step size.

Question: How do I adjust the SGD Step Size?

How do I adjust the SGD step size?

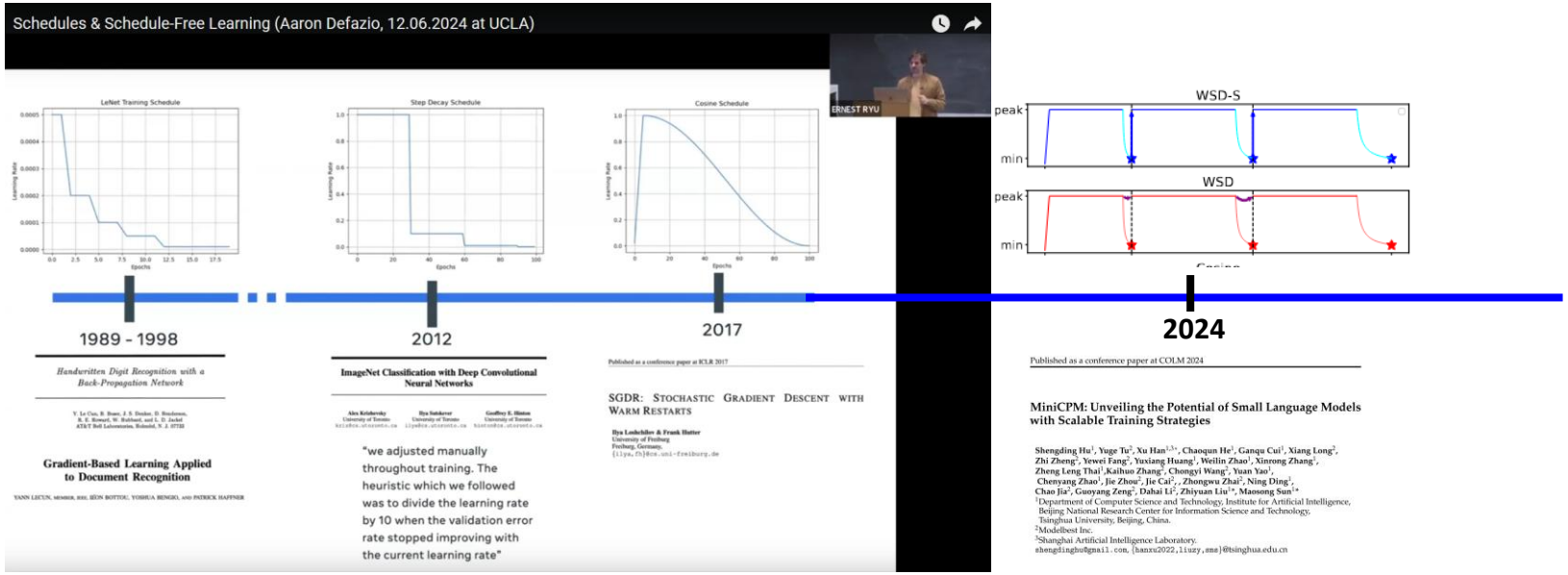
- Lots of methods have proposed to adjust the SGD step size as we go.
  - "Update step size based on some simple statistics".
  - "Do a line-search based on the mini-batch".
  - "Do gradient descent on the step size".
  - Mark being provocative: these methods usually work badly in practice.

- Most of these methods have at least one of these problems:
  - Introduce a new hyper-parameter that is just as hard to tune.
  - Do not converge theoretically (can catastrophically fail).
  - Converges theoretically but works badly in practice.
  - Need to assume that $\sigma(w_k)^2$ goes to 0.

- I have not had good luck getting anything working well across problems.
  - And I have tried a lot of things over the years.

How do I adjust the SGD step size?

- Practice: "Tune a bunch of hyper-parameters".



  - Piecewise-constant step size, with tuned drops and interval lengths.
    - Common ImageNet strategy is 60 passes through data, with drops at 20 and 40.
  - WarmUp and cosine annealing tuning α and parameters of schedule.
    - Or WarmUp Stable Decay with different α and annealing schedules.

How do I adjust the SGD step size?

- Theoretical:
  - Multi-Step progress bounds:
    - Exist analyses based on PEP, but tend to focus on verifying rates of existing methods.
  - Allowing $L = \infty$: gradient clipping and WarmUp.
  - Adapting to local $L(w_k)$:
    - First SGD method adapting to $L(w_k)$ was coin betting.
      - Increase/decrease step size based on $\langle \nabla f_{i_k}(w_k), w_k \rangle$ (descendent of Kesten's method).
        - » Normalized by AdaGrad-Norm denominator.
    - Recent example is Prodigy.
      - Practical variant is one of the best "completely parameter free" ways to train neural networks.
    - These methods tend to work worse than "tune hyper-parameters".
      - Optimizing over schedules allows adaptivity to complex hidden dynamics.

[Orabona & Pál, 2016, Mischenko & Defazio, 2024]

Adapting to Noise Level?

- What about adapting to $\sigma(w_k)^2$?

$$\mathbb{E}[f(w_{k+1})] \leq f(w_k) - \alpha_k \left(1 - \frac{\alpha_k L}{2}\right) \|\nabla f(w_k)\|^2 + \frac{\alpha_k^2 L}{2} \sigma(w_k)^2 .$$

  - If gradient has heavy-tails, then $\sigma(w_k)^2 = \infty$.
    - May be no safe step size (potential for arbitrarily-large catastrophic jumps).
  - Gradient clipping can also help here.
    - Clipped(GD) in $\mathbb{E}$[descent lemma] (`max_grad_norm` $= \tau$) when clipped:

$$\mathbb{E}[f(w_{k+1})] \leq f(w_k) - \alpha_k \nabla f(w_k)^T \mathbb{E}\left[\text{clip}[\nabla f(w_k)]\right] + \frac{\alpha_k^2 L}{2} \tau^2 .$$

      - Deterministically bounds the effect of noise, but introduces bias.
        - » Only converges to region around solution, can increase τ to decrease bias/region.

[Polyak & Tsypkin, 1979, Zhang et al., 2020]

Adapting to Noise Level?

- What about adapting to $\sigma(w_k)^2$?

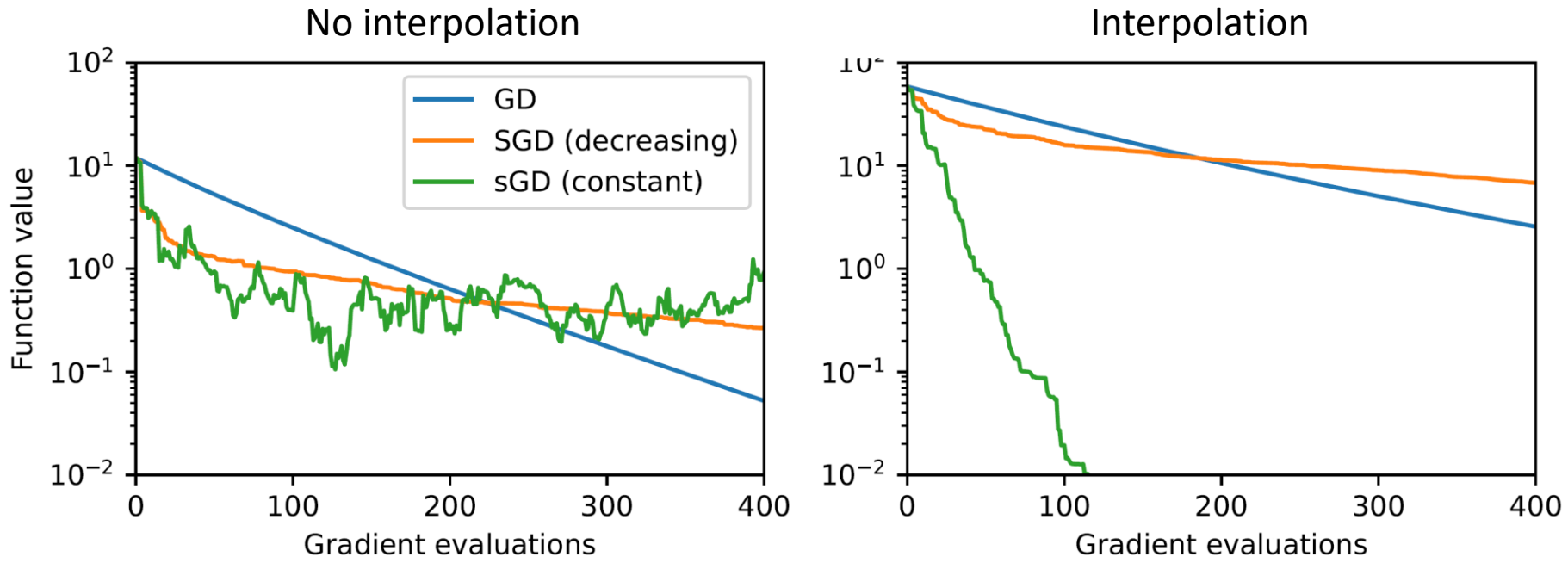
$$\mathbb{E}[f(w_{k+1})] \leq f(w_k) - \alpha_k \left(1 - \frac{\alpha_k L}{2}\right) \|\nabla f(w_k)\|^2 + \frac{\alpha_k^2 L}{2} \sigma(w_k)^2 \ .$$

  - At the other extreme, converge $\sigma(w_k)^2$ converging to 0.
    - In this setting SGD with constant α_k can converge almost as fast as GD!

  - This arises when model can interpolate the data.
    - We can achieve a training loss of 0 (fit every training example so $\sigma(w_*)^2 = 0$).
      - Found to be true in some computer vision models.
    - First studied in 1990s to justify use of constant α_k for training neural networks.

[Solodov, 1998, Tseng, 1998]

Setting the SGD Step Size under Interpolation



- Under interpolation, no need to decrease step sizes.
  - No noise-dominated phase.
- Still expect good performance if you are close to interpolating data.
  - Will not converge but will get close to solution, since $\sigma(w_*)^2$ will be small.

How do I adjust the SGD step size?

- Many ways to characterize interpolation:
  - Strong growth condition, weak growth condition, expected smoothness.

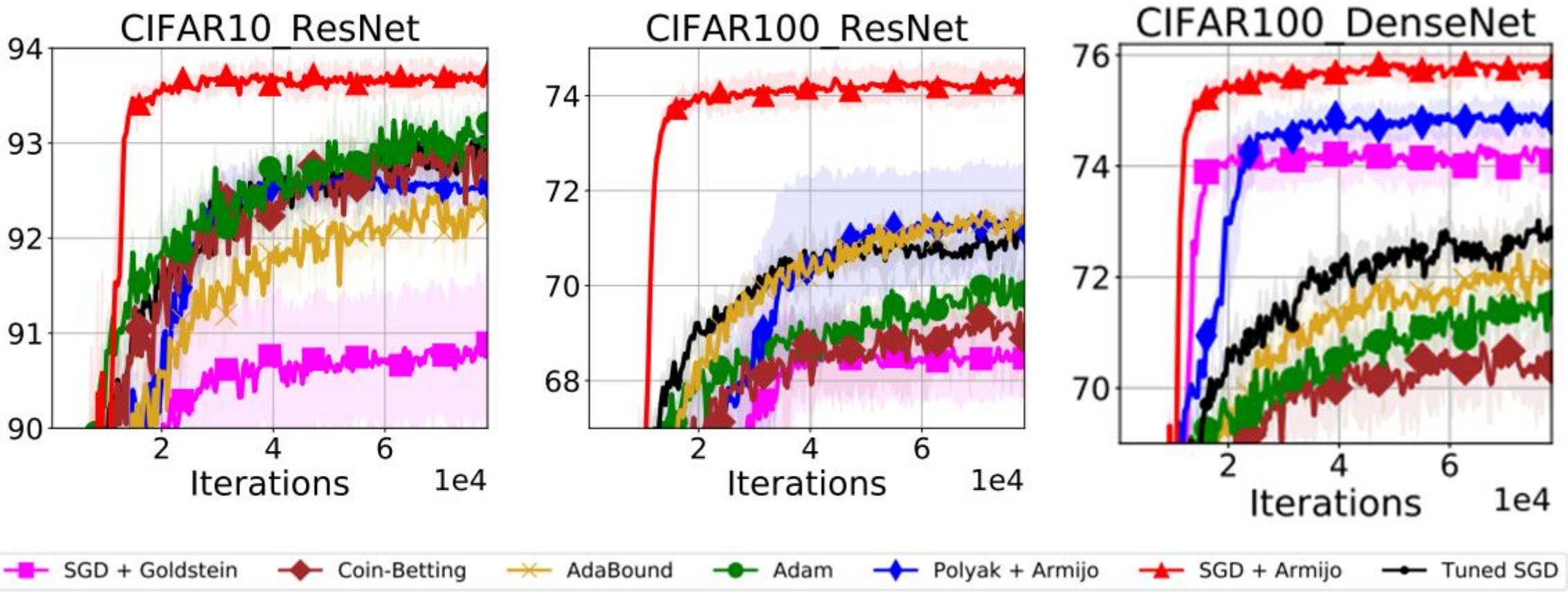
- But if just assume $\sigma(w_*)^2 = 0$, get convergence with $\alpha_k = 1/L_{\max}$.
  - Where $L_{\max}$ is smoothness of sharpest function.
    - "SGD converges like GD on worst training example."

- Under interpolation, can also use many Level 2+ GD step sizes:
  - Stochastic line search (SLS): SUG on mini-batch.
    - Use Armijo condition and increase step size between iterations.
      - With careful implementation, cost is less in practice than trying out 2 fixed step sizes.
  - Stochastic Polyak step-size (SPS): Polyak step size on mini-batch.

[Bach & Moulines, 2011, Schmidt & Le Roux, 2013, Needell et al., 2014, Gower et al., 2019, Vaswani et al., 2019, Loizou et al, 2021]

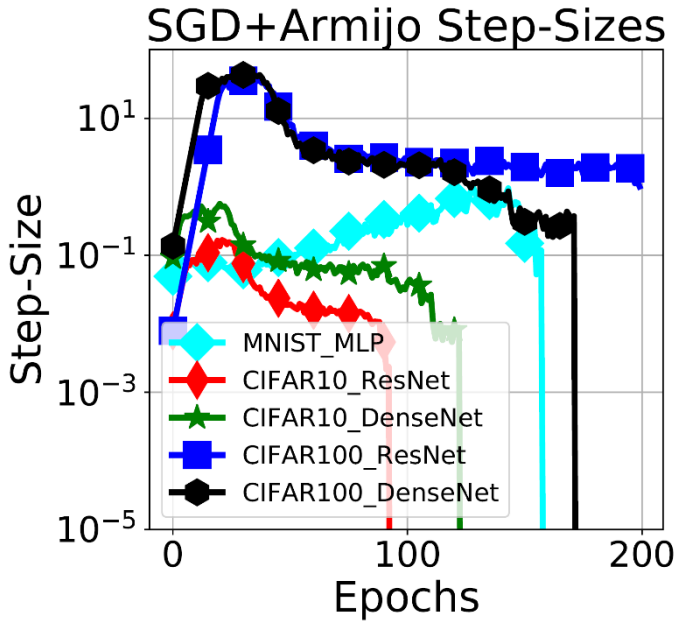
SGD with Armijo Line-Search with Interpolation

- SGD + Armijo outperforms other methods on many benchmarks.



SGD with Armijo Line-Search with Interpolation

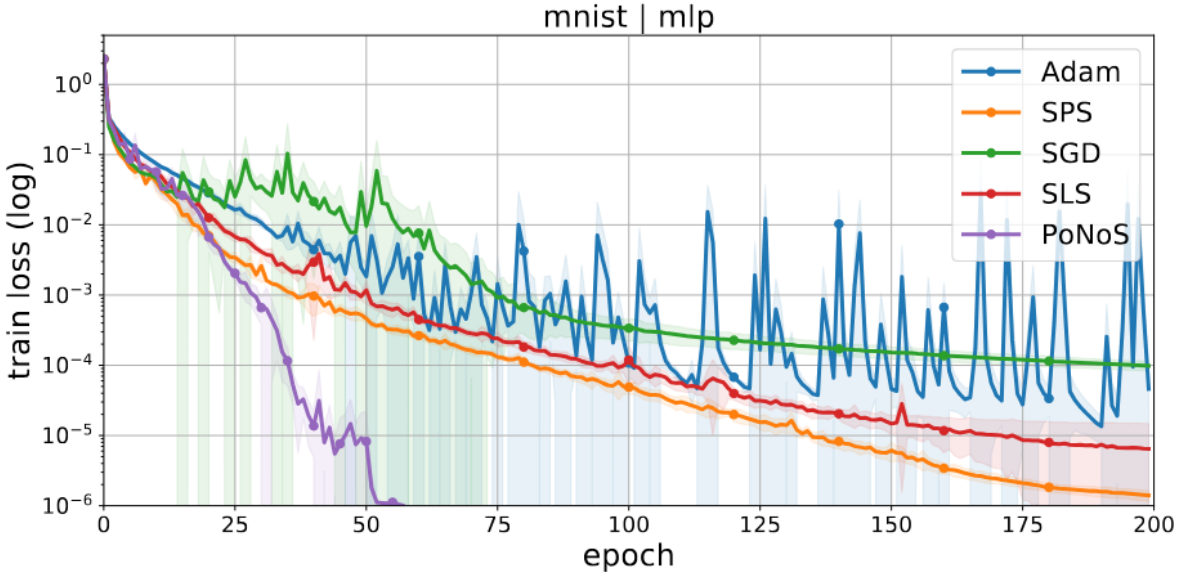
- Step sizes found using Armijo backtracking:

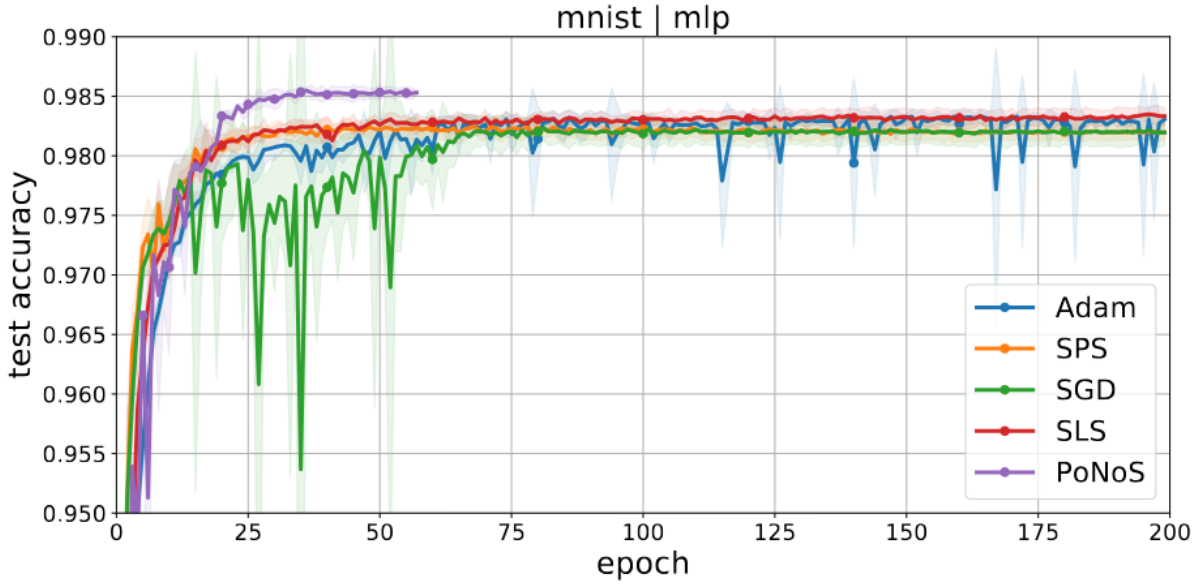


  - Uses different step sizes on different datasets at different times.
    - Seems to have a WarmUp phase.
  - But eventually step sizes can become too small.
    - Eventual numerical collapse on neural networks.

SGD with Non-Monotonic Armijo Line Search

- SGD with larger initializations and non-monotonic SUG:
  - Which allows function to increase to accept larger steps ("PoNoS").





  - No warm-up needed and no numerical collapse in practice.
    - Later: explored step size at limit of what could converge, and discovered "flatland".

[Galli et al., 2023, Galli et al., 2026]

Back to the Noise-Dominated Phase

- Interpolation assumption leads to fast convergence of SGD.
  - Step sizes that adapt to the problem instead of using schedules.
  - But assumption does not hold for language models.
    - Not always a unique “correct” next word/token.

- So we do need to worry about the noise-dominated phase.
  - But we do not want slow decreasing sizes.

- The trick allowing SGD to use larger step sizes is averaging.

Averaging Iterates over Time

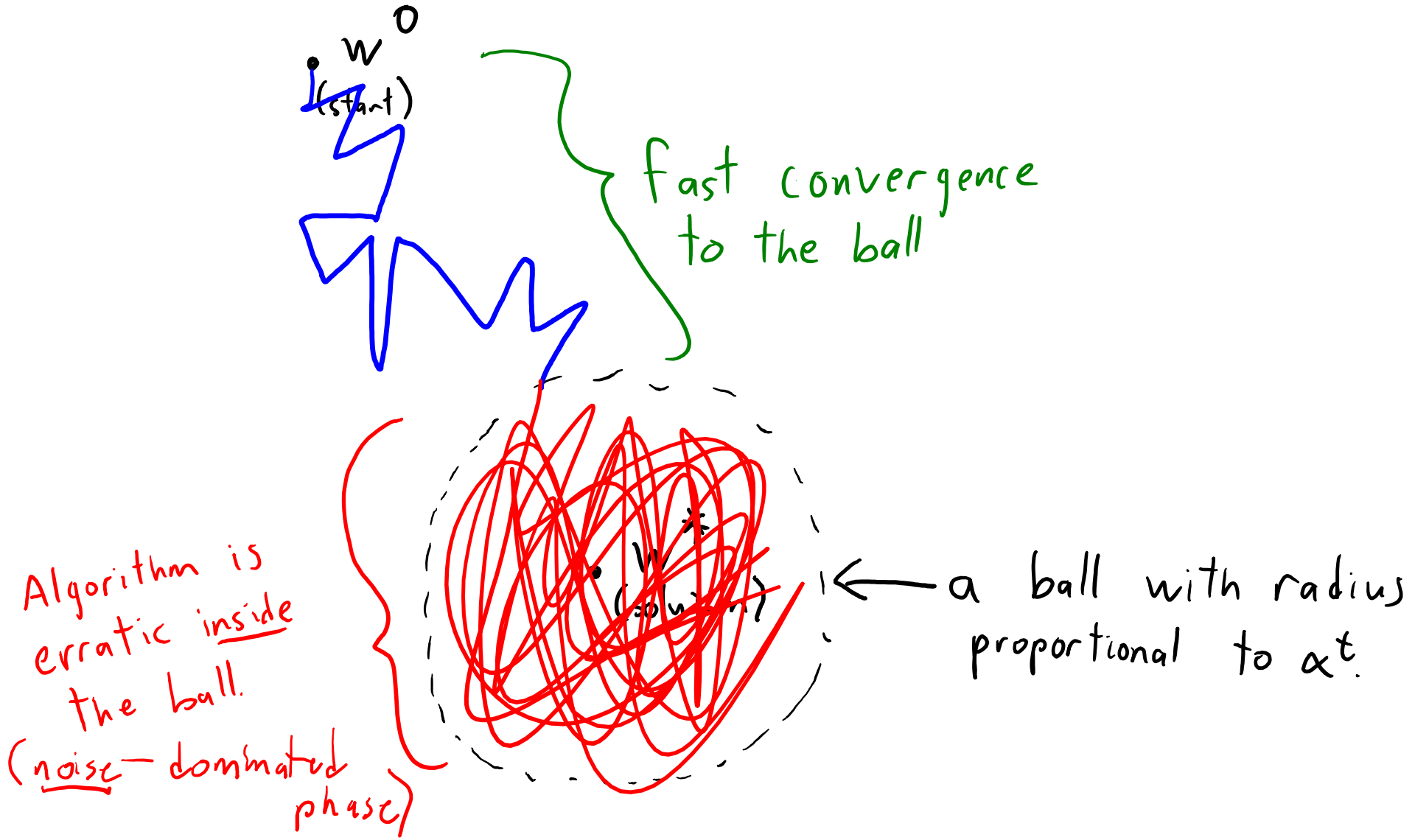
- A classic idea is to keep a running average of the iterates:

$$w_{k+1} = w_k - \alpha_k \nabla f_{i_k}(w_k) \qquad \bar{w}_{k+1} = \text{average}(w_1, w_2, \ldots, w_k, w_{k+1})$$

  - Average is offline ("watching" SGD but not influencing).
- Key idea: average $\bar{w}_k$ is less affected by noise than w_k.
  - Classic Polyak-Ruppert result: $\bar{w}_k$ converges at optimal asymptotic rate.
    - Even if you have second-order information.
      - "Averaging is the optimal way to handle noise variance near the solution."
  - Common variation is to exclude early iterations from average.
    - "Suffix averaging" or "stochastic weight averaging (SWA)".
      - Averaging slows you down in gradient-dominated phase (don't want old iterates if moving fast).

[Ruppert, 1988, Polyak & Juditsky, 1992, Shamir & Zhang, 2012, Izmailov et al., 2018]

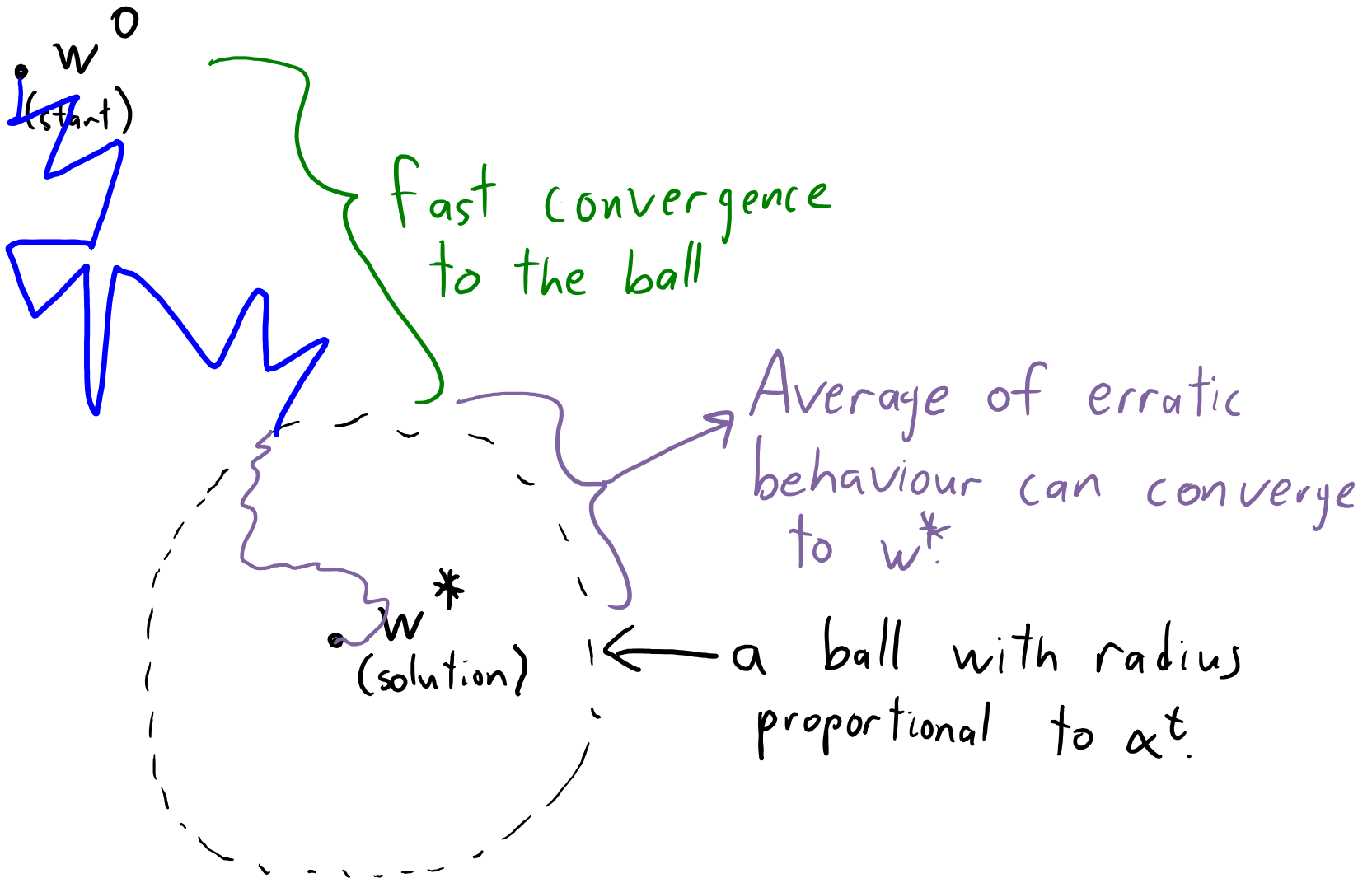
Stochastic Gradient with Iterate Averaging



Stochastic Gradient with Iterate Averaging

Often, you average the second half of the iterations.

Set $\bar{w}^t = \frac{1}{t/2} \sum_{k=t/2}^{t} w^k$



Stochastic Gradient with Iterate Averaging

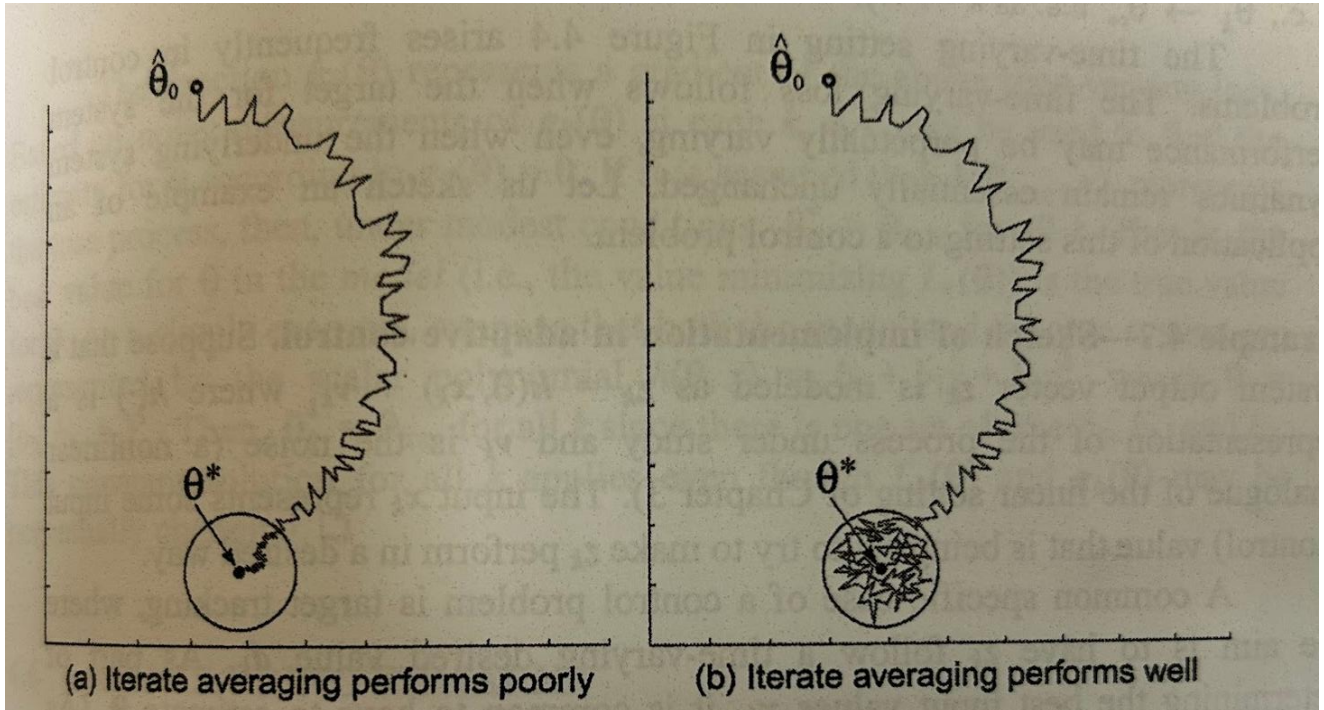


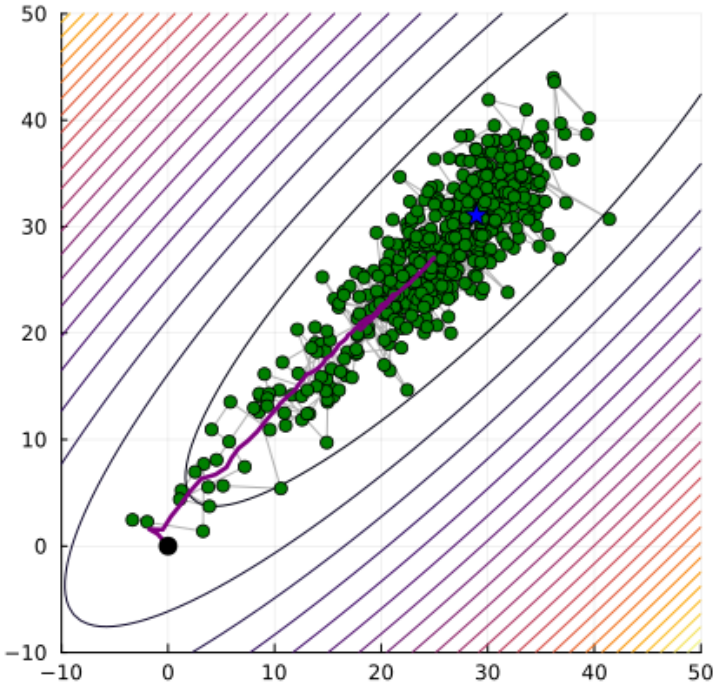
Figure 4.3. Two search paths for a $p = 2$ problem. Note *approximate* monotonic improvement in $\hat{\theta}_k$ until search reaches circled area. Iterate average $\bar{\theta}_k$ is usually poorer estimate than $\hat{\theta}_k$ before search process enters circled area. If process does not oscillate about θ^* (case (a)), terminal iterate average $\bar{\theta}_k$ will be poorer estimate than $\hat{\theta}_k$. If process oscillates around θ^* as in case (b), then terminal $\bar{\theta}_k$ is likely to provide better estimate than $\hat{\theta}_k$ if process is allowed to run long enough inside circled area. Case (a) is commonly associated with low noise, while case (b) may be seen with high noise.

[Spall, 2003]

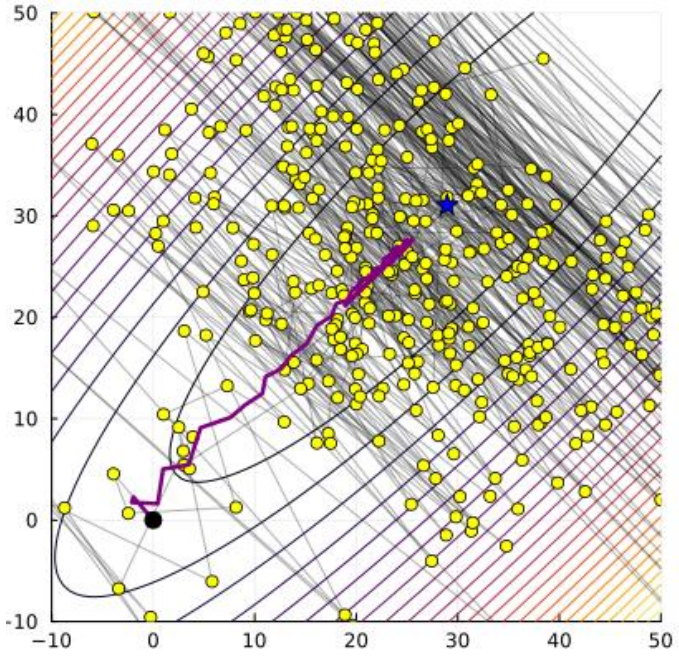
Stochastic Gradient with Iterate Averaging

- Constant step sizes: $\alpha_k = \alpha$ showing average iterate.

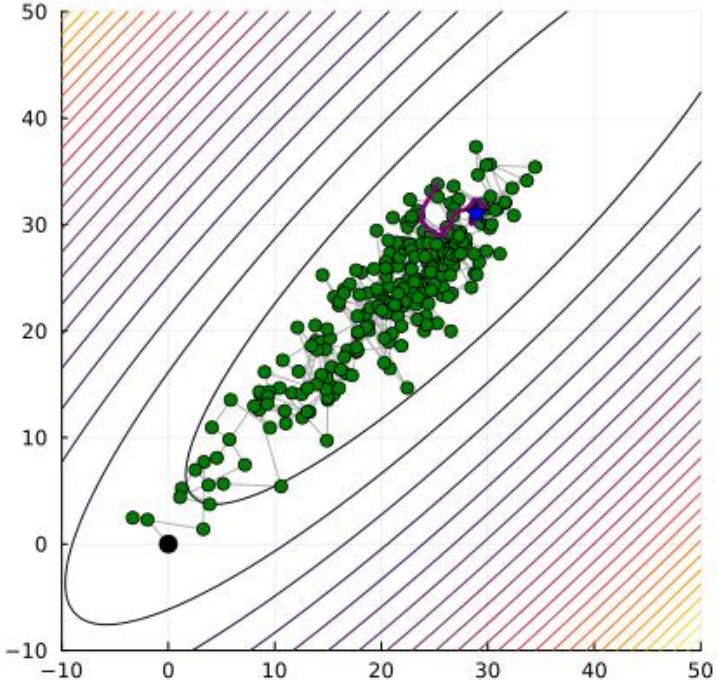
$\alpha_k = 1/L$



$\alpha_k = 2/L$



$\alpha_k = 1/L$, averaging second half



Averaging Iterates over Time

- Many variations on iterate averaging exist::
  - Weighted averages of the iterates (usually emphasizing recent iterates).
  - Online averaging which uses average in update ("primal averaging").

- Notable recent averaging variant is ScheduleFree:
  - Interpolates between Polyak-Ruppert and primal-averaging.
  - Winner of Algoperf 2024 "black box" neural network training competition.
    - Still needs hyper-parameter search for base learning rate and momentum parameter.
    - ScheduleFree+: many changes to get it working for LLMs

[Defazio et al., 2024, Defazio, 2026]

Averaging Gradients over Time

- Alternately some algorithms use arunning average of the gradients:

$$w_{k+1} = w_k - \alpha_k \bar{g}_k \qquad \bar{g}_k = \text{average}(\nabla f_{i_0}(w_0), \nabla f_{i_1}(w_1), \dots, \nabla f_{i_k}(w_k))$$

- Key idea: average of gradients has a lower variance than $\nabla f_{i_k}(w_k)$.
  - Many variations:
    - Weighted averages of the iterates (usually emphasizing recent iterates).
    - Replacing w_k with w_0 ("dual averaging").
    - Exclude early iterations from average.
      - Averaging slows you down in gradient-dominated phase (don't want old gradients if moving fast).
- You can also do both types of averaging across time:
  - Bather's method: online average iterate updated with average gradient.
    - Shares asymptotic optimality of Polyak-Rupert averaging.
- Or you can average gradients at the same time…

[Ruszczyński & Syski, 1984, Bather, 1989, Nesterov, 2009]

Question: How I set the batch size?

Mini-Batch SGD

- Instead 1 random example, we often use a mini-batch in SGD:

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

average gradient on a mini-batch of examples

  - We can often compute gradient of many examples in parallel.
  - The mini-batch gives a better approximation of true gradient $\nabla f(w_k)$.

- Drawback: increases cost of iterations.
  - But this cost is easy to parallelize ("embarrassingly parallel").
    - Often the best way to use parallel distributed computation.

- Selling point: bigger batches lead to a longer gradient-dominated phase.
  - "Instead of decreasing the step size, increase the batch size!"
  - Trend towards larger batch sizes: GPT3 used $|B_k|$ = 3 million.
    - More recent LLMs have used ~100 million.

[Smith et al., 2018, Brown et al., 2020]

How do I set the mini-batch SGD step size?

- Mini-batch SGD guarantees decrease in expectation if:

$$\alpha_k < \frac{2}{L}\left(\frac{1}{1+\sigma(w_k)^2/(|B_k|\cdot\|\nabla f(w_k)\|^2)}\right)$$

  - "With a mini-batch size of 100, effect of noise is divided by 100."
- Longer gradient-dominated phase:
  - Can use $\alpha_k = \frac{1}{L}$ until $\|\nabla f(w_k)\| < \frac{\sigma(w_k)^2}{|B_k|}$, can keep big step sizes longer.
- Linear scaling rule:
  - "With a mini-batch size of 100, step size can be 100-times larger."
    - Derived empirically, but comes from math: when $\frac{\sigma(w_k)^2}{\|\nabla f(w_k)\|^2}$ is big we can have $\alpha_k < \frac{2}{L}\frac{|B_k|}{\sigma(w_k)^2}$.
    - But there is a limit of $2/L$.

[Friedlander & Schmidt, 2012, Goyal et al., 2018]

How do I choose the batch size?

- If increasing batch is free due to hardware then do it.
  - Will not hurt, but gain is small if gradient-dominated.

- Practice: "critical batch size" that minimizes total computation (found empirically).

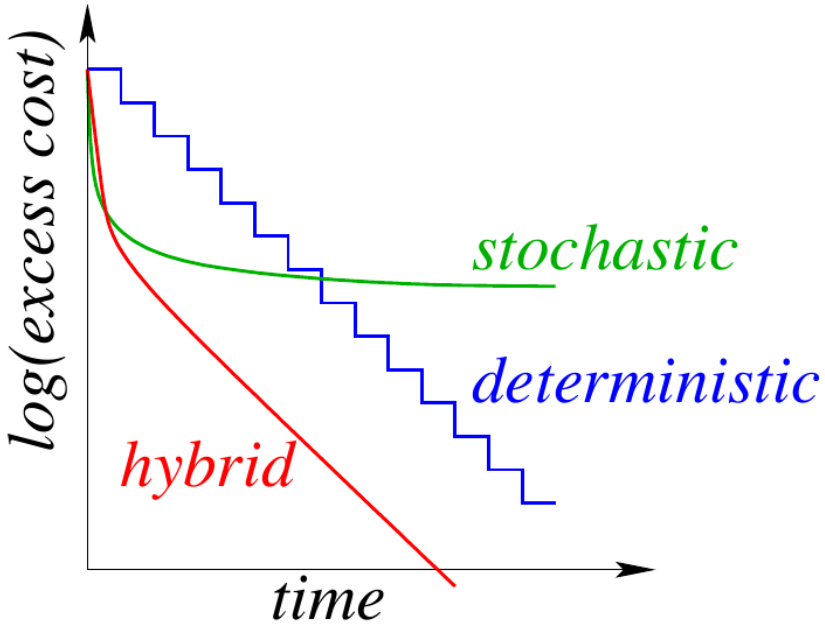
- Theory is surprisingly simple:
  - If you fix $\alpha_k = 1/L$ then mini-batch SGD converges quickly to an error of $O\left(\frac{\sigma^2}{|B_k|}\right)$.
  - Because GD reduces error "faster" than $|B_k|$ reduces variance,
    if we want an error of ϵ the compute-optimal batch size is $O\left(\frac{\sigma^2}{\epsilon}\right)$.

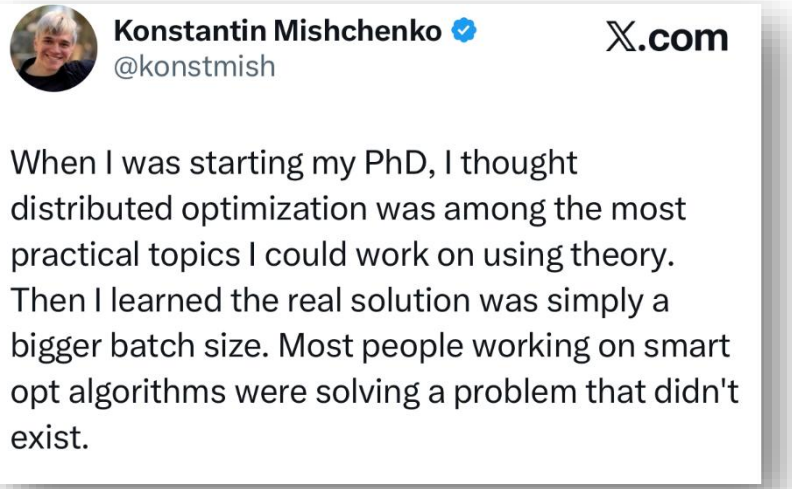
- So optimal $|B_k|$ depends on error ϵ you need (or equivalently how long you run).
  - Recent empirical critical batch size papers have re-discovered this.
  - And if you want to "converge", need a growing batch size (but not needed for any error).

[Friedlaner & Schmidt, 2012, McCandlish et al., 2018, Merrill et al., 2025]

Don’t Decrease the Step Size, Increase the Batch Size!

- The longer you run SGD, the more noise can slow it down:





- Consider the “hybrid” GD/SGD method of using an increasing batch size $|B_k|$.
  - Can stay in fast gradient-dominated phase throughout training.
    - Allows SGD to converge with constant step size, without interpolation.
    - Or using clever GD step sizes and other gradient-dominated tricks (next topic).
- Or just choose a really big batch size to guarantee a small error.

Map of Speeding up SGD

- Gradient-dominated:
  - ~~Large step sizes.~~
  - Momentum.
  - Adam and friends.
  - Preconditioning.
- Noise-dominated:
  - ~~Decreasing step sizes.~~
  - ~~Interpolation.~~
  - ~~Averaging.~~
  - ~~Mini-batching.~~

Finite datasets tricks:

- Importance sampling.
- Random reshuffling.
- Variance reduction.

Momentum and the Heavy-Ball Method

- A common variation on GD is Polyak's heavy-ball method:

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

  - Where we have added a momentum term.
- Though we often now use PyTorch form with a velocity vector:

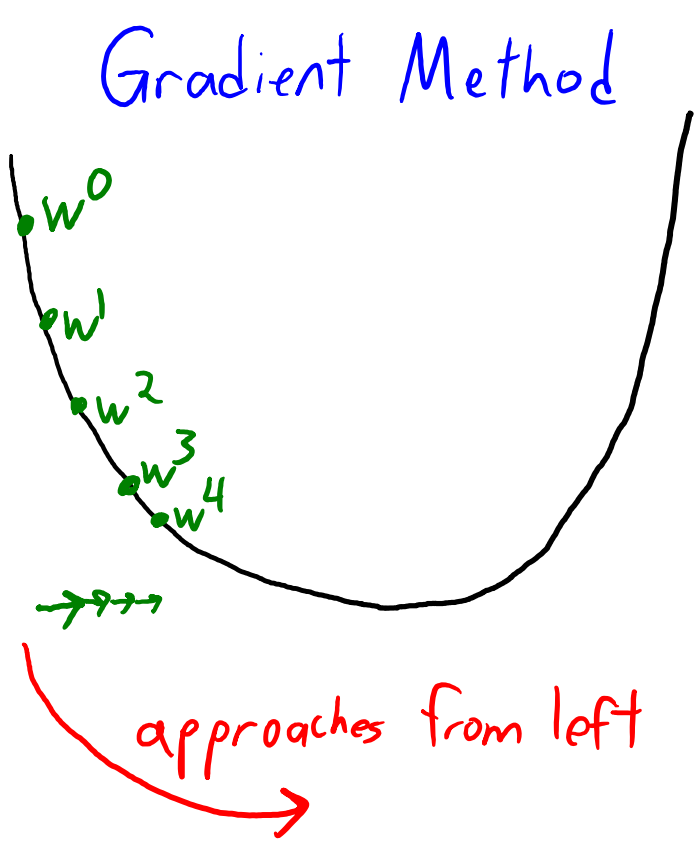
$$m_{k+1} = \beta_k m_k - \alpha_k \nabla f(w_k)$$

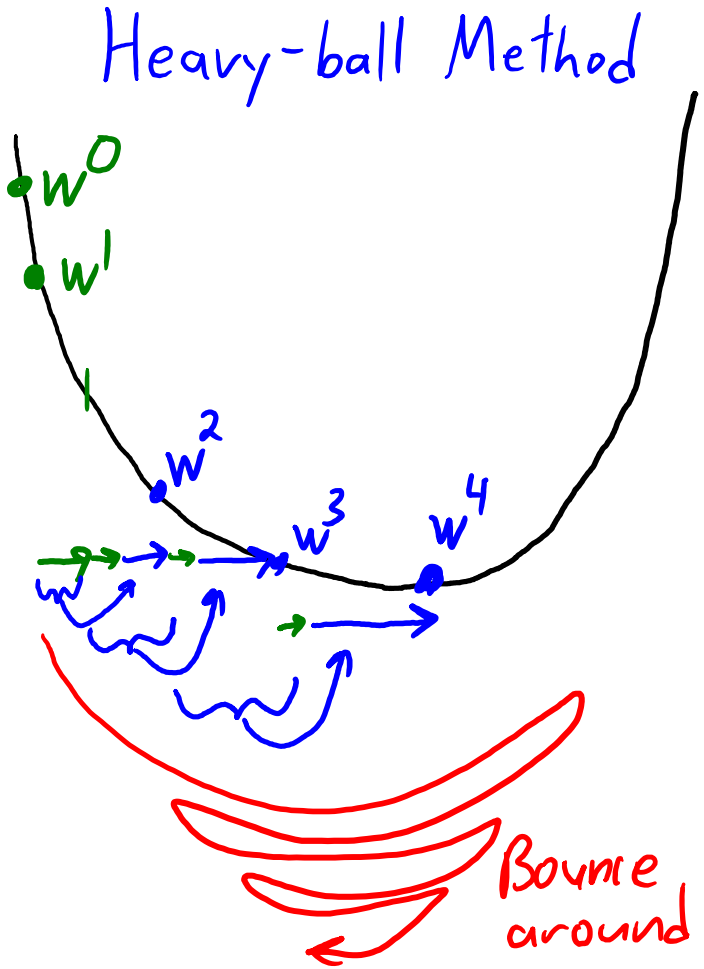
$$w_{k+1} = w_k + m_{k+1}$$

- Amazing properties for quadratic functions:,
  - Has a larger range of "good" step sizes.
  - Optimal fixed α_k and β_k get "accelerated" Chebyshev rate (more stable).
  - Optimal per-iteration α_k and β_k are faster than GD with spectral steps.

[Polyak, 1964]

Gradient Descent vs. Heavy-Ball Method

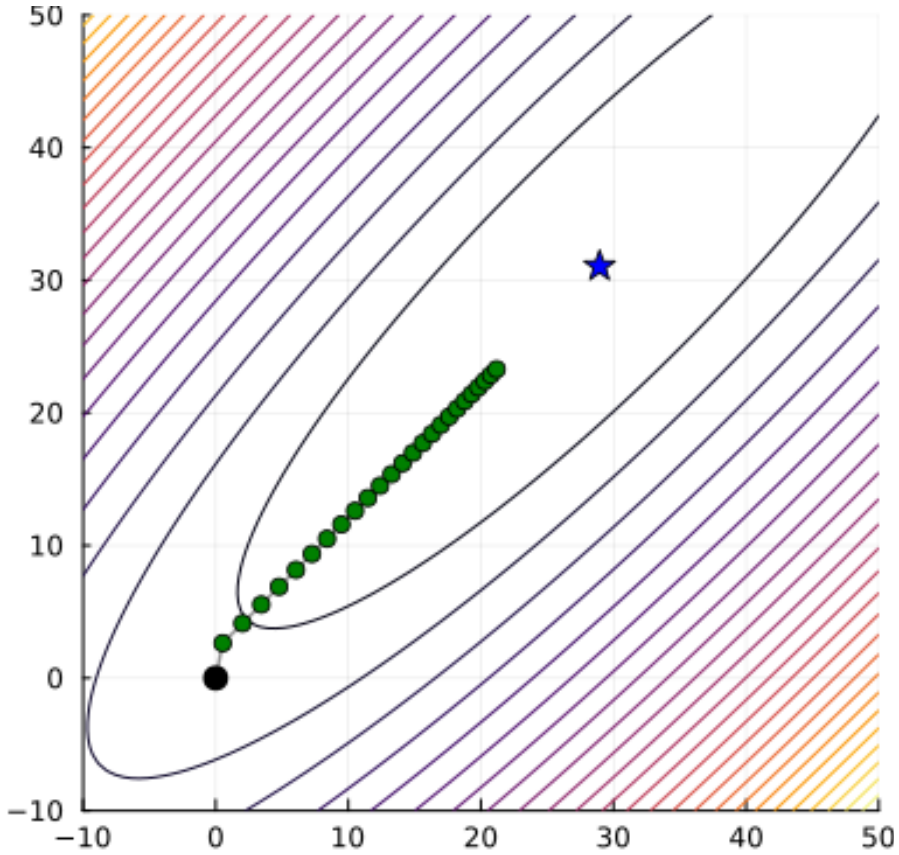




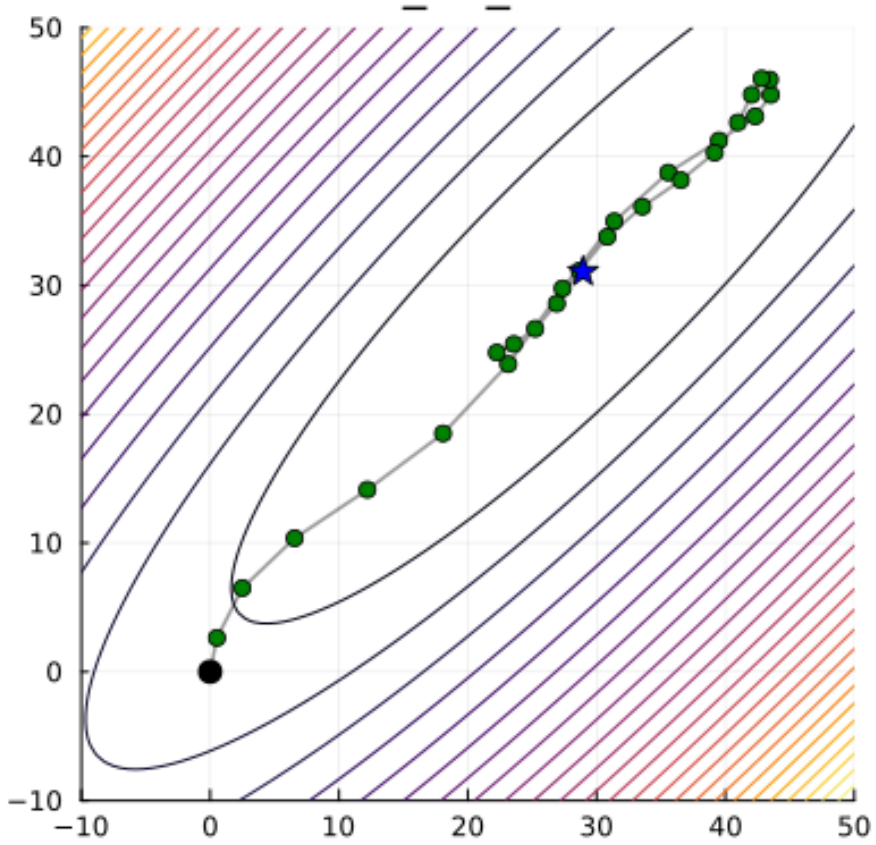
Heavy-Ball Method on Quadratic Functions

- Adding momentum to GD on quadratic function:

GD: $\alpha_k = 1/L$



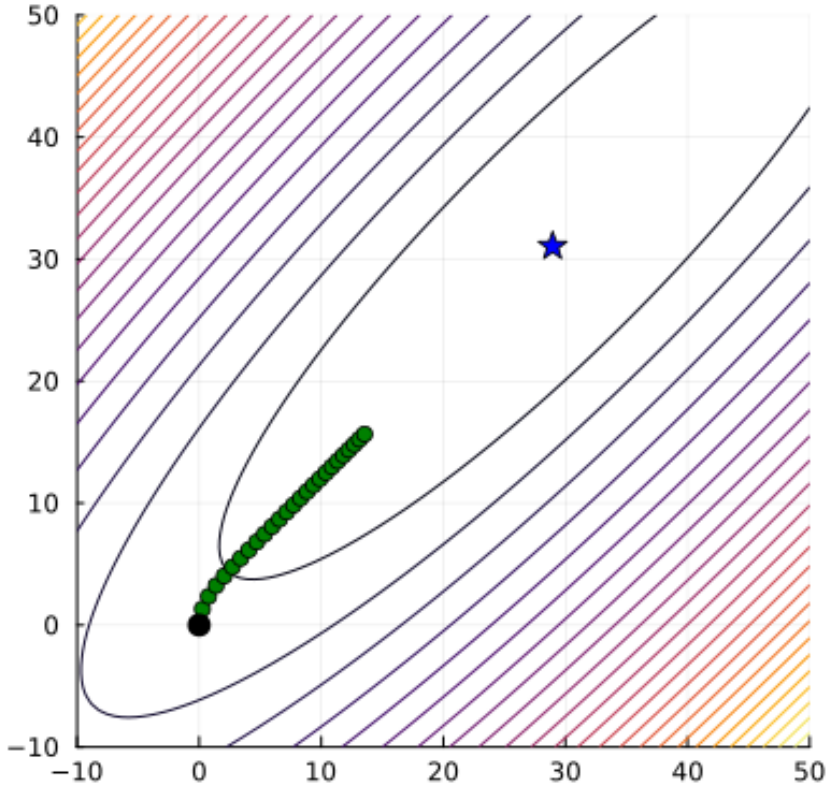
Heavy Ball: $\alpha_k = 1/L, \beta_k = 0.9$



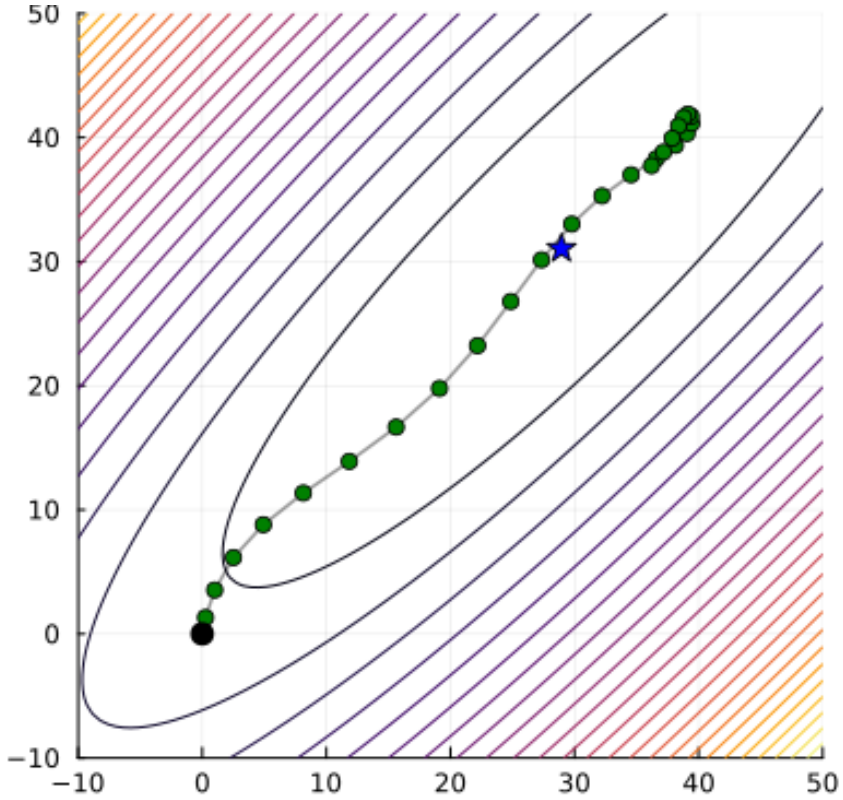
Heavy-Ball Method on Quadratic Functions

- Adding momentum to GD on quadratic function:

GD: $\alpha_k = 1/2L$



Heavy Ball: $\alpha_k = 1/2L$, $\beta_k = 0.9$

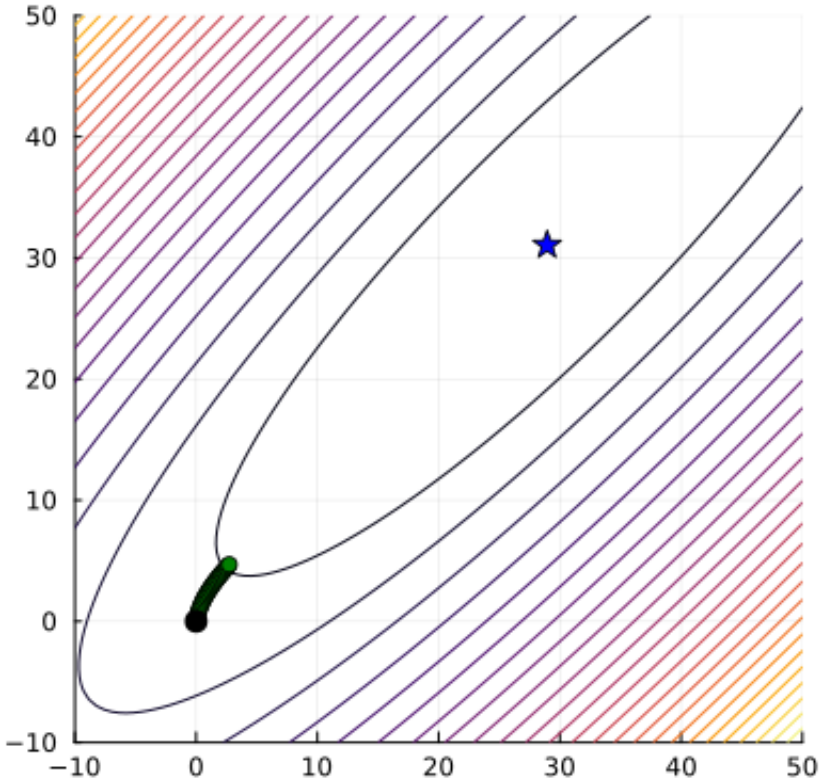


(faster for smaller step sizes)

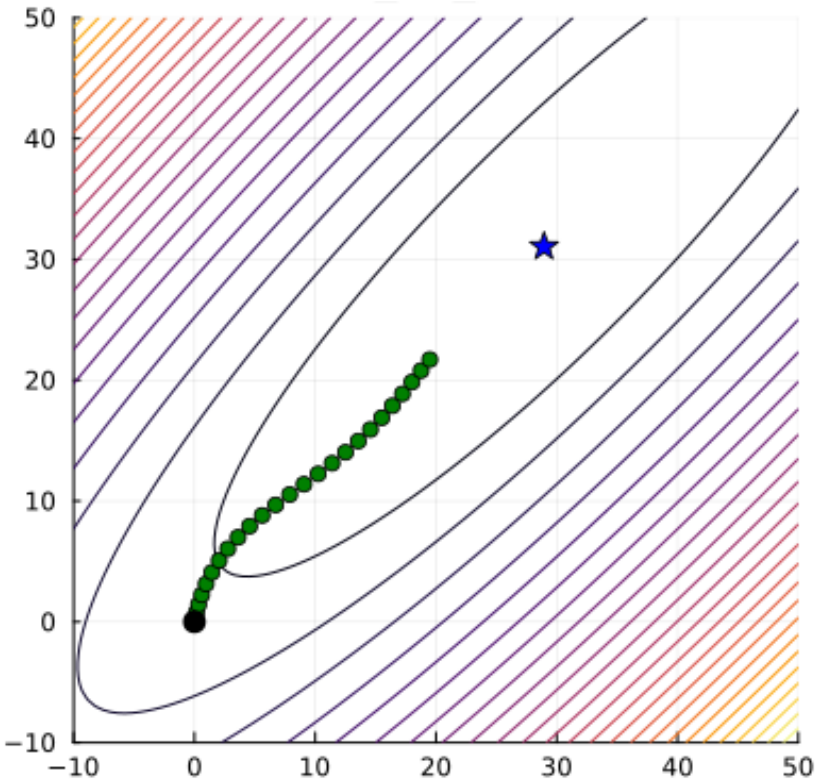
Heavy-Ball Method on Quadratic Functions

- Adding momentum to GD on quadratic function:

GD: $\alpha_k = 1/10L$



Heavy Ball: $\alpha_k = 1/10L$, $\beta_k = 0.9$

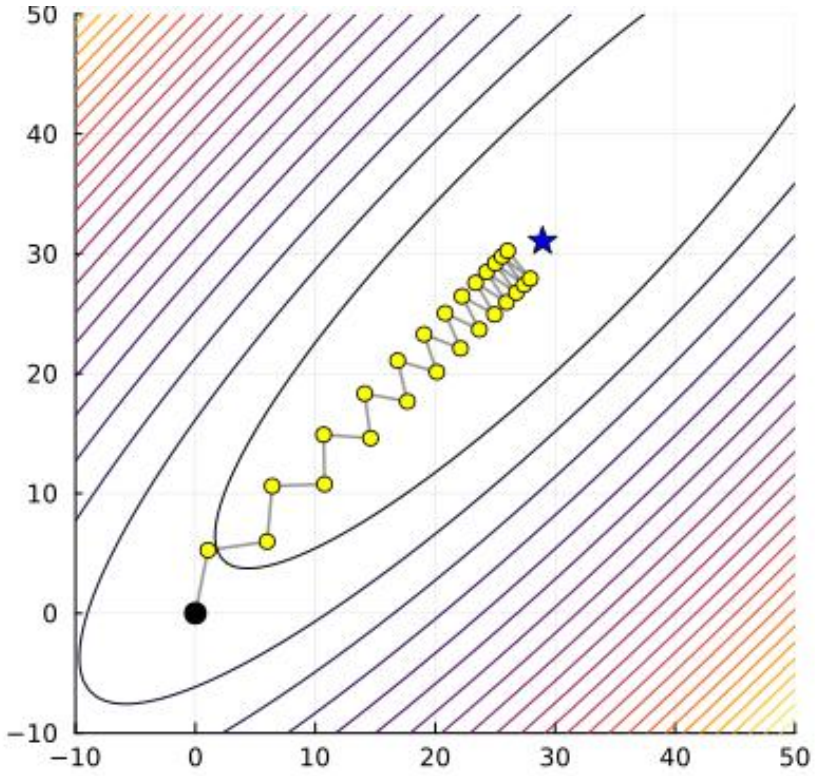


(faster for smaller step sizes)

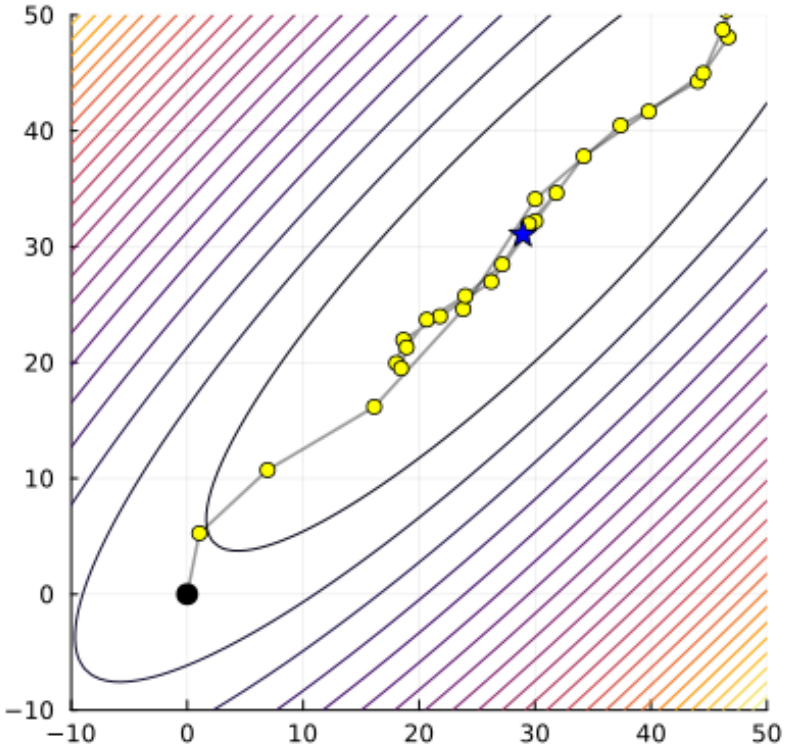
Heavy-Ball Method on Quadratic Functions

- Adding momentum to GD on quadratic function:

GD: $\alpha_k = 2/L$



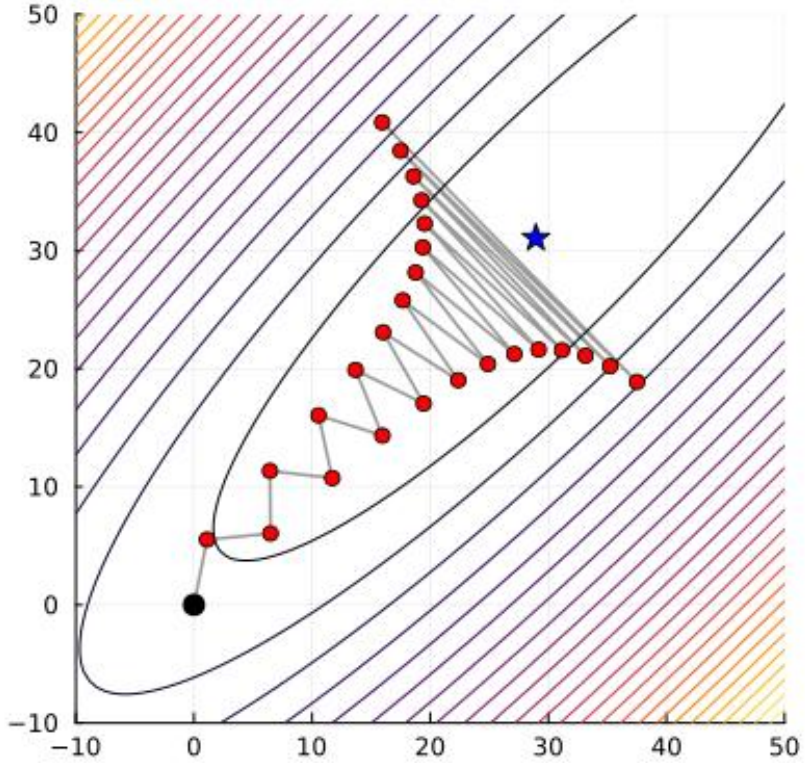
Heavy Ball: $\alpha_k = 2/L, \beta_k = 0.9$



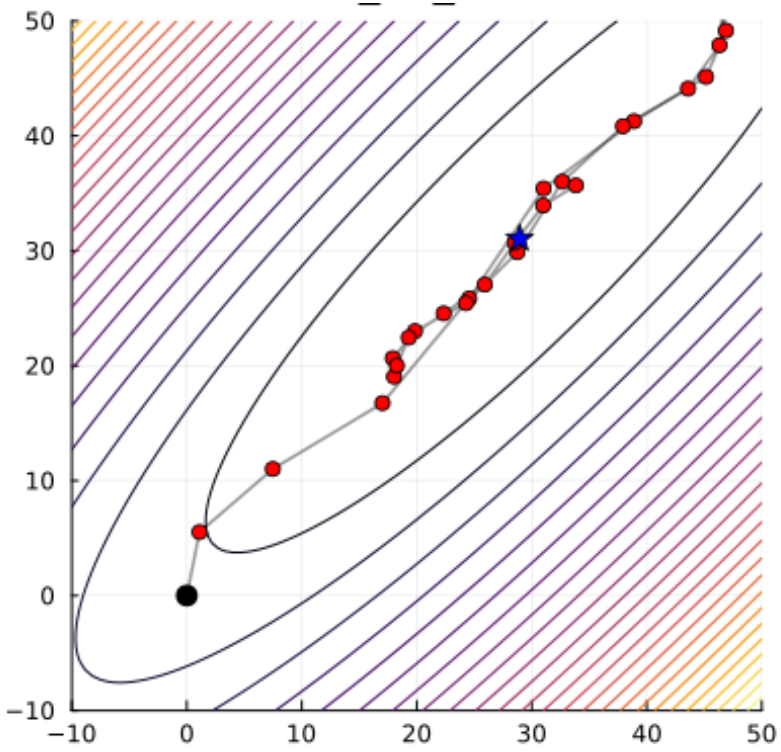
Heavy-Ball Method on Quadratic Functions

- Adding momentum to GD on quadratic function:

GD: $\alpha_k = 2.1/L$



Heavy Ball: $\alpha_k = 2.1/L, \beta_k = 0.9$

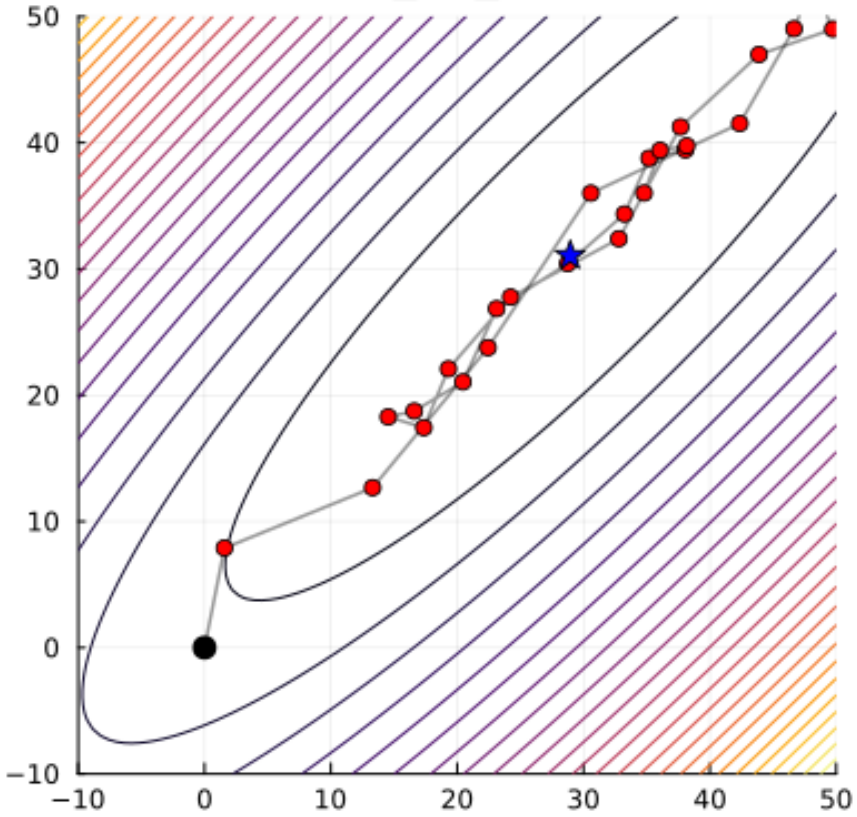


(more stable for larger step sizes)

Heavy-Ball Method on Quadratic Functions

- Adding momentum to GD on quadratic function:
  - On quadratics, safe step for convergence:
    - $\alpha_k < \frac{2}{L}(1+\beta)$.
  - So with $\beta = 0.9$, we can allow
    - $\alpha_k < 3.8/L$.

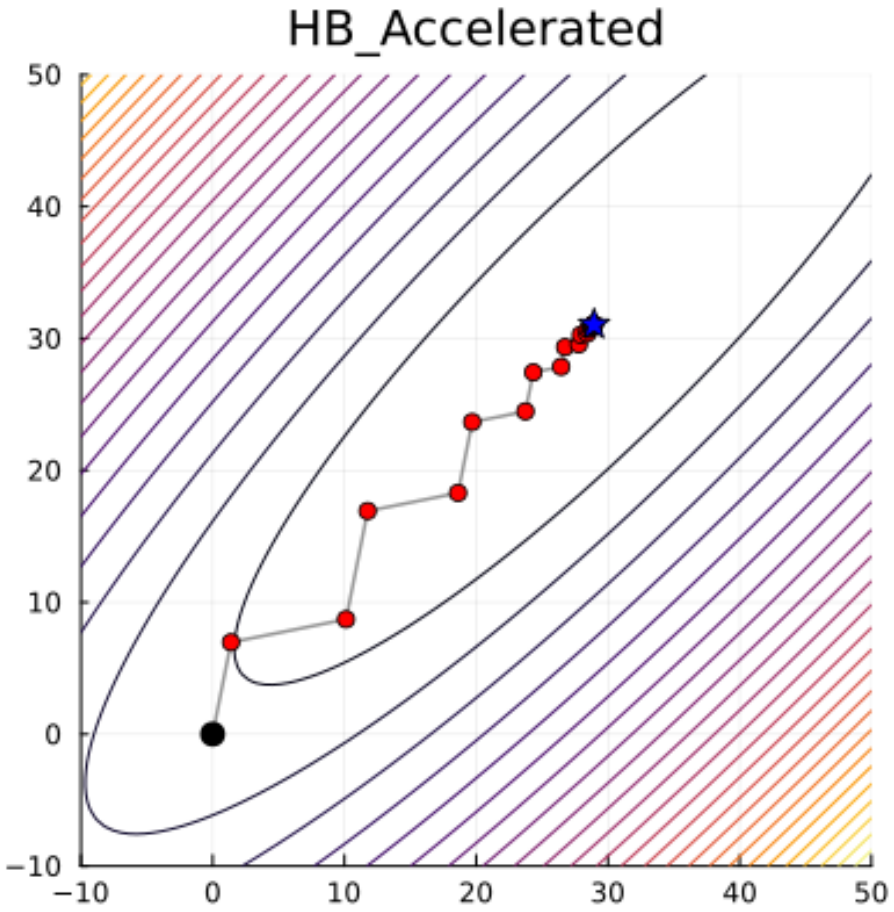
Heavy Ball: $\alpha_k = 3/L$, $\beta_k = 0.9$



(more stable for larger step sizes)

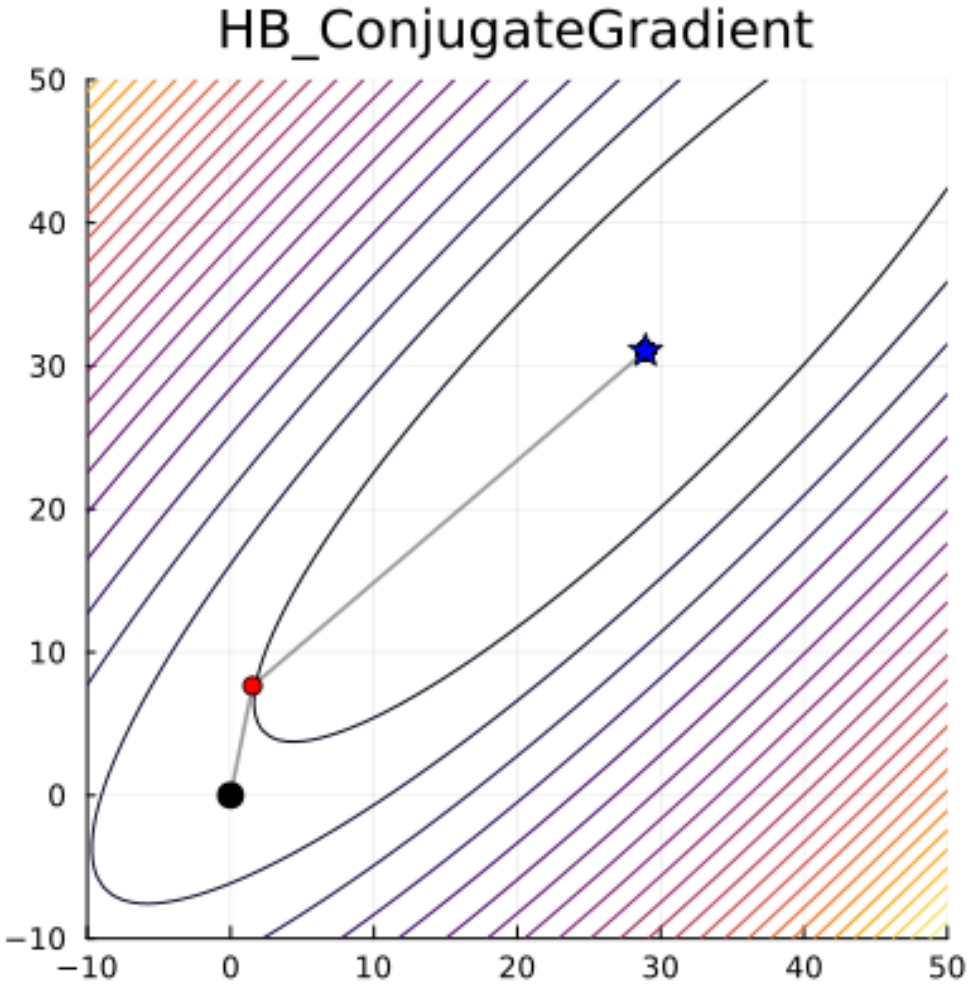
Heavy-Ball Method on Quadratic Functions

- Adding momentum to GD on quadratic function:
  - On quadratics, safe step for convergence:
    - $\alpha_k < \frac{2}{L}(1+\beta)$.
  - So with $\beta = 0.9$, we can allow
    - $\alpha_k < 3.8/L$.
  - Fastest fixed α_k and β_k given by:
    - $\alpha_k = \frac{4}{(\sqrt{L}+\sqrt{\mu})^2}$, $\beta_k = \left(\frac{\sqrt{L}-\sqrt{\mu}}{\sqrt{L}+\sqrt{\mu}}\right)^2$.
    - Often close to $\alpha_k = 4/L$ and $\beta_k = 1$.
    - Achieves accelerated rate.
      - Like GD with Chebyshev step size (but more stable).



Heavy-Ball Method on Quadratic Functions

- Adding momentum to GD on quadratic function:
  - Conjugate gradient (CG) algorithm:
    - Optimal per-iteration α_k and β_k:
    - Faster than GD with spectral step size.
      - Solves d-dimensional problem in d steps.
    - Closed-form: does not need λ_i or even L or μ.



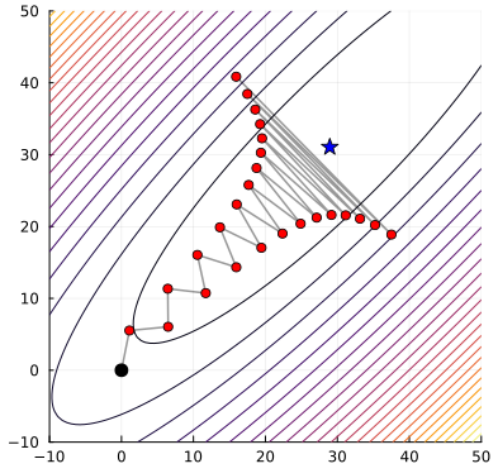
[Hestenes & Stiefel, 1952]

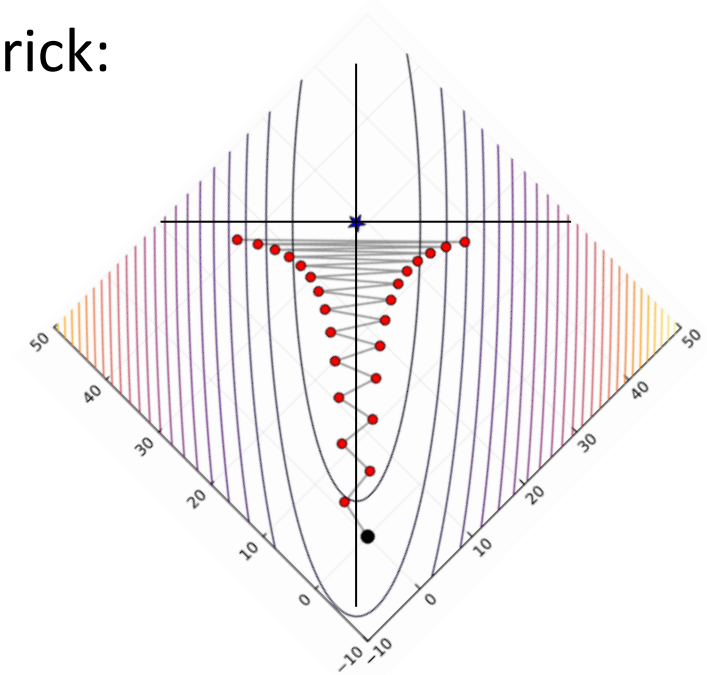
- Momentum is not interesting in 1d: could just use better step sizes.

  "Can we stop talking about balls rolling down hills already?"

  - Aaron Defazio

- Heavy-ball method is translation- and rotation-invariant.
  - So let's do quadratic trick:

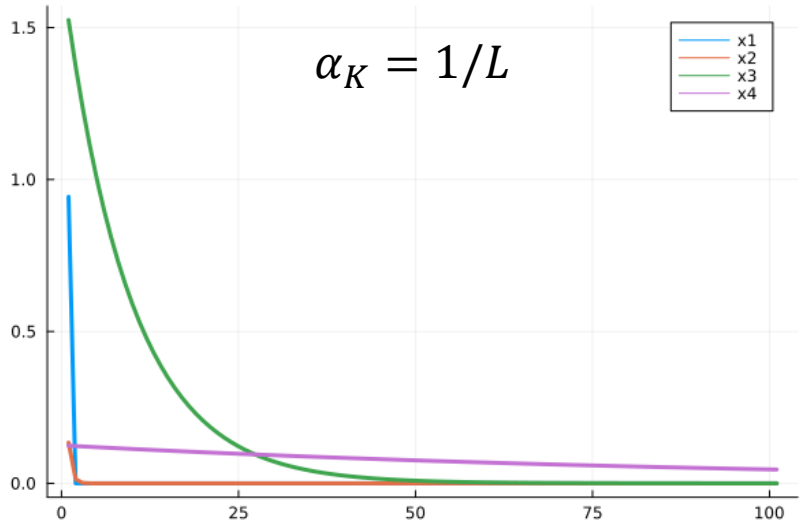


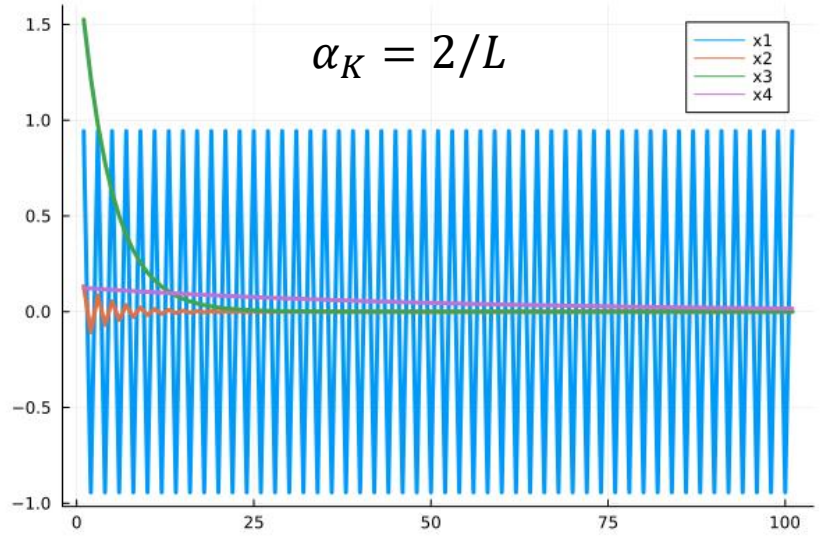


  - So we can talk about balls rolling down valleys and ravines.
    - "Momentum builds speed along ravines and dampens oscillations along valleys."

Gradient Descent vs. Heavy-Ball Dynamics

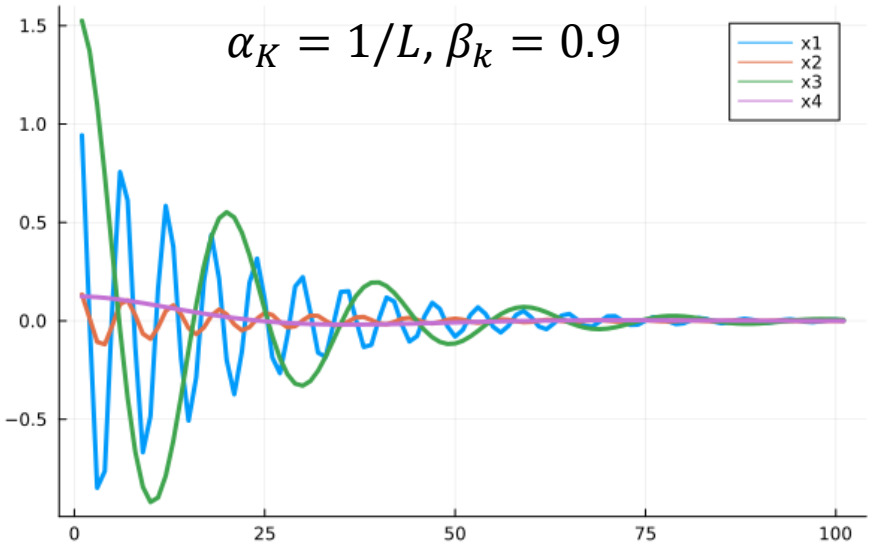
Gradient Descent:

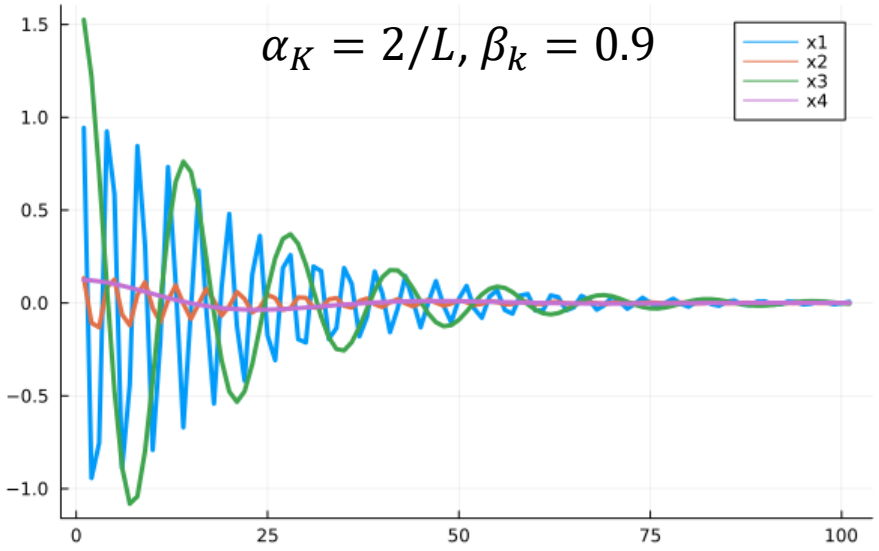




Heavy-ball:

"Discreteized damped harmonic oscillator along each eigenvector"

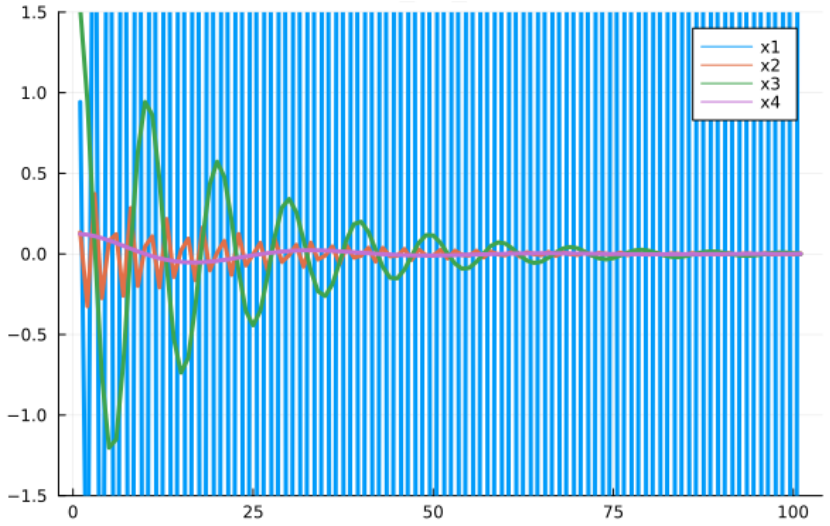




Limits of Heavy-Ball Dynamics

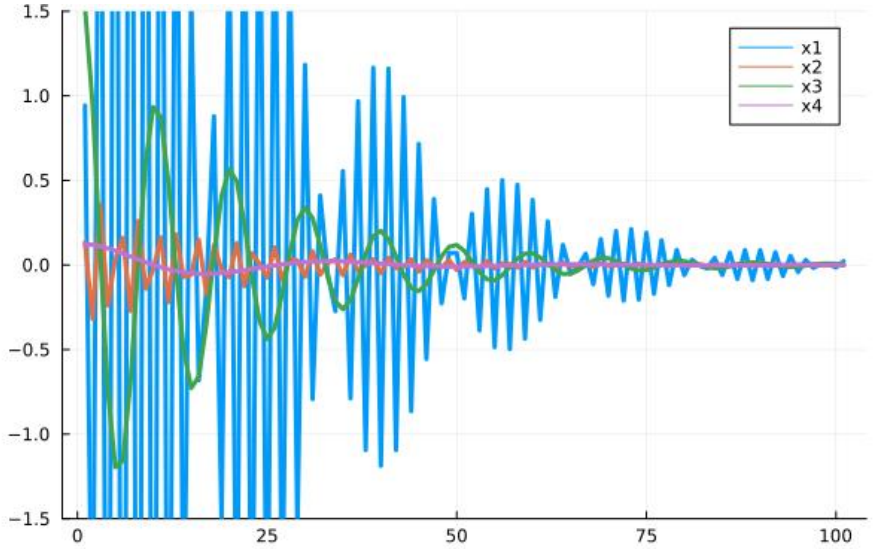
Max fixed α_k given β_k

$\alpha_K = 2(1+\beta)/L, \beta_k = 0.9$

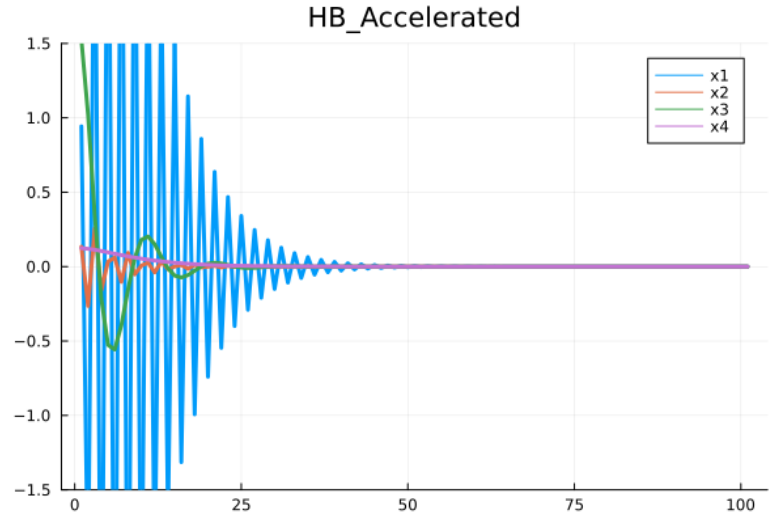


Optimal fixed α_k given β_k

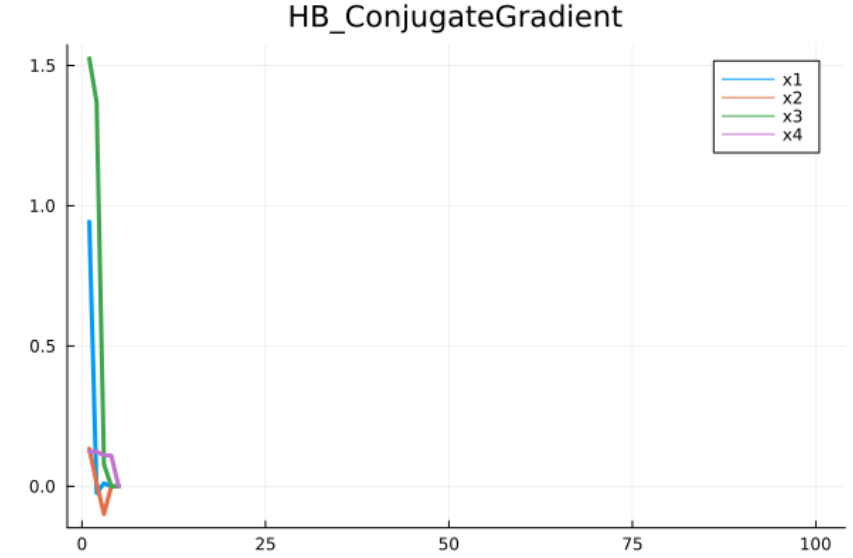
$\alpha_K = 2(1+\beta)/(L+\mu), \beta_k = 0.9$



Jointly-optimal fixed α_k and β_k:



Per-iteration optimal α_k and β_k:



Heavy-Ball: Beyond Quadratics

- Heavy-ball theory is less elegant for non-quadratics:
  - Requires smaller steps on general functions: $\alpha_k < \frac{2}{L}(1-\beta)$.
  - Does not achieve accelerated rate on usual problem classes.
    - Though some recent works show momentum helps in non-standard scenarios.

- On the practical side:
  - Tuned α_k and $\beta_k = 0.9$ typically helps neural network training.
    - Does make sense locally if Hessian is not changing fast (big range of good step sizes).
  - Non-linear CG among best off-the-shelf solvers.
    - Theory: globally slower than GD but locally faster.
    - Needs precise approximation of greedy step size, but usually net gain anyways.

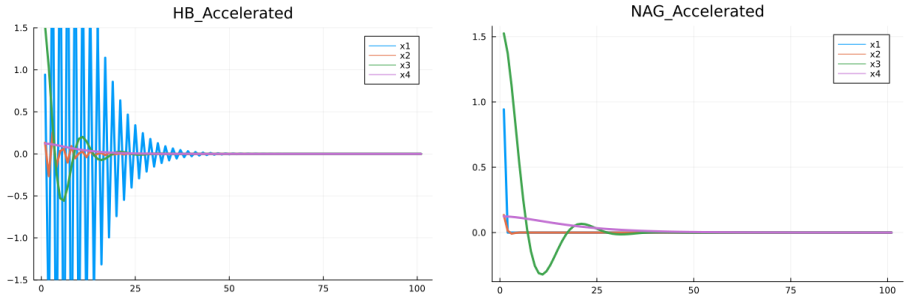
[Fletcher & Reeves, 1964, Polyak, 1987, Zavriev & Kostyuk, 1993, Zhang & Hager, 20025, Wang et al., 2022, Marumo et al., 2023, Das Gupta et al., 2024, Goujaud & Taylor, 2025]

Nesterov Acceleration

- Nesterov's accelerated gradient (NAG) method:

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

  - Adds a gradient difference which dampens oscillations.



- Original motivation:
  - Faster rate than GD for convex functions ($\alpha_k = 1/L$ and $\beta_k \to 1$).
- Theory: NAG faster than GD in many settings (including non-convex).
  - But unfortunately each setting requires different β sequence.
- Practice: NAG usually worse than non-linear CG.
  - But probably more suitable with noise.

[Nesterov, 1983, Nesterov, 2004, Li & Lin, 2023, Liao & Kyrillidis, 2024]

Gradient-Dominated vs. Noise-Dominated Phase

- Expect momentum to be helpful in gradient-dominated phase.
  - Where SGD acts like deterministic GD.
  - Some works argue that it even helps dampen noise in this phase.

- But momentum can hurt in noise-dominated phase.
  - Well known property of amplifying noise.
    - Noisy gradients get added to momentum and influences future iterations.
  - Classic stochastic momentum proofs required $\beta_k \to 0$ in SGD.

[d'Aspremont, 2008, Devolder et al., 2011, Schmidt et al., 2011, Defazio, 2021]

Question: Why does Adam work?

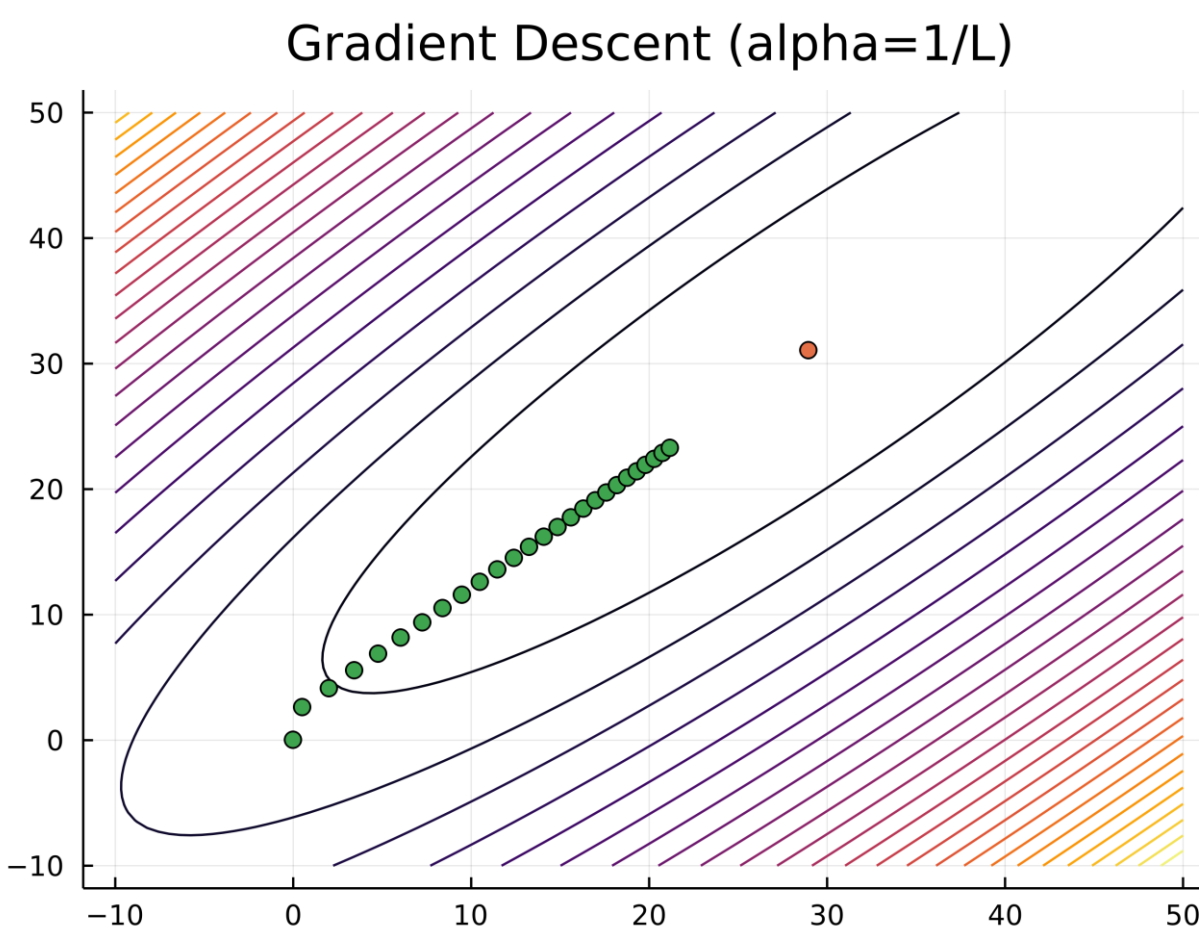
The Perplexity of the Adam Optimizer

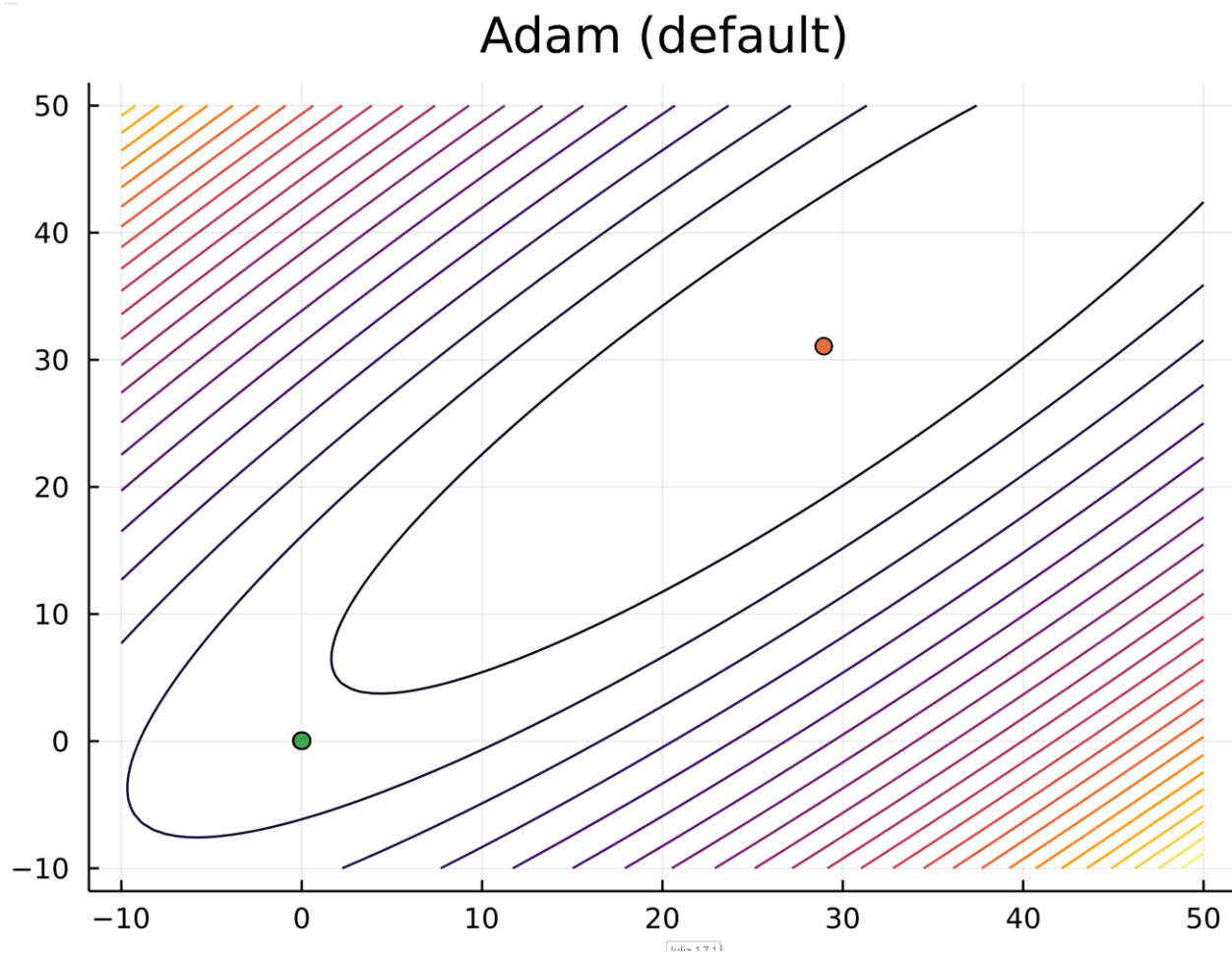
- An extremely popular SGD variant is Adam:
  - Combination of several ideas with good properties in specific settings:
    - AdaGrad which has appealing properties for online learning.
    - Momentum which is appealing for gradient-dominated phase.
    - Step sizes that are constant which is appealing for gradient-dominated phase.
- The properties of Adam are not well understood.
  - Analysis in original Adam paper was wrong.
  - There are simple functions where Adam performs terribly.
  - But it is difficult to beat Adam on many tasks.
    - Only 1 method in Algoperf 2025 competition beat AdamW baseline.
      - "Schedule-free" version of AdamW.

[Kingma & Ba, 2015]

Gradient Descent vs. Adam

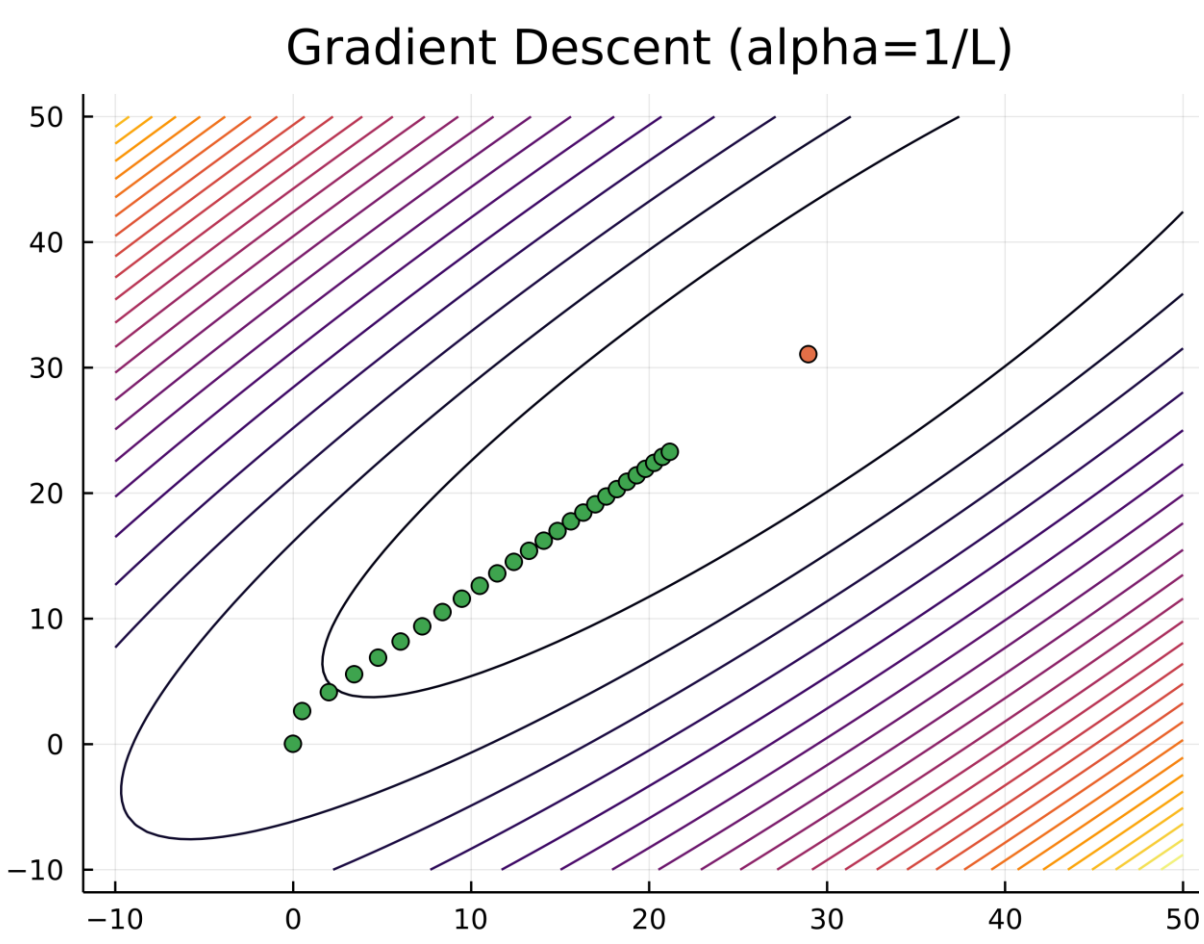
- GD vs. Adam:

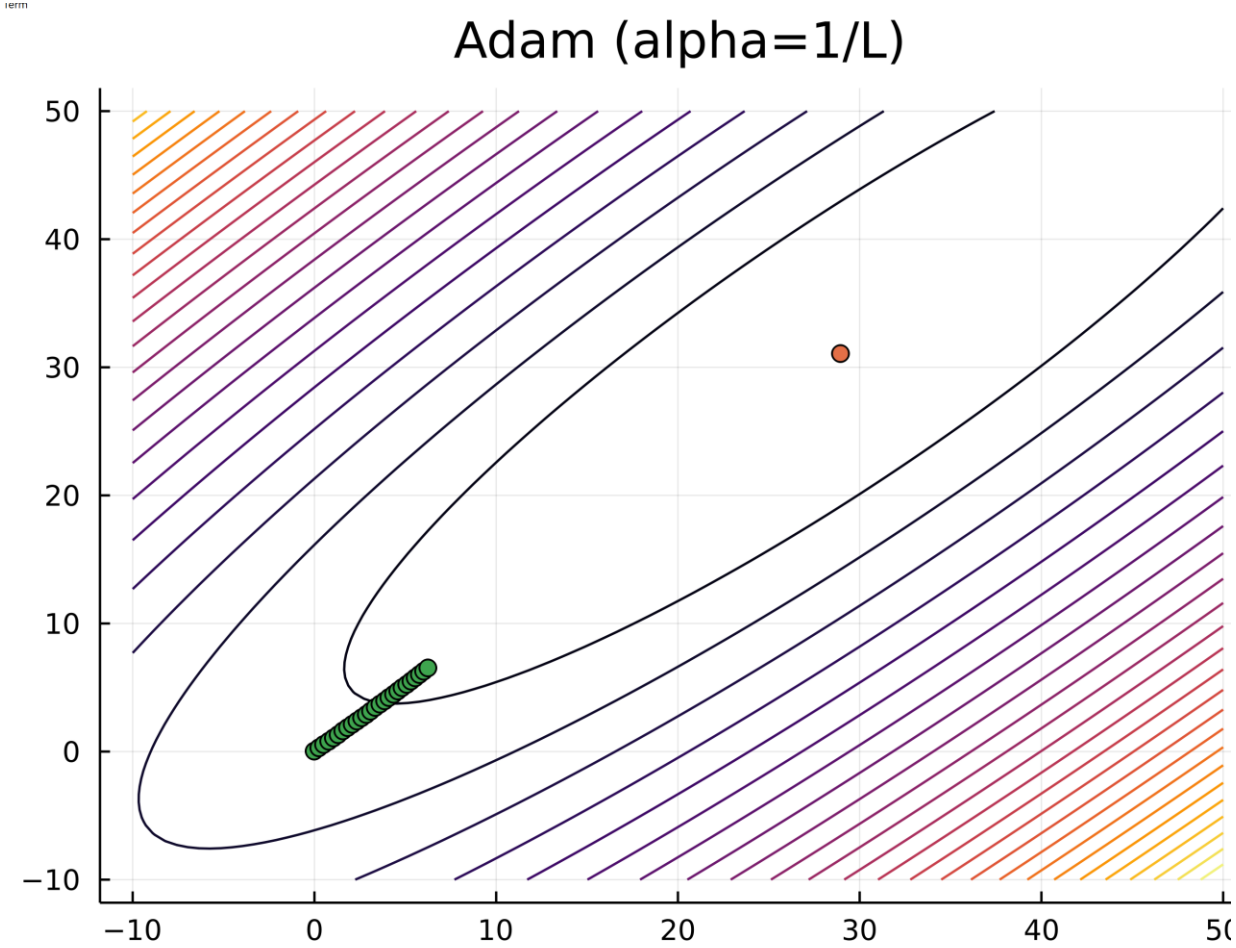




Gradient Descent vs. Adam

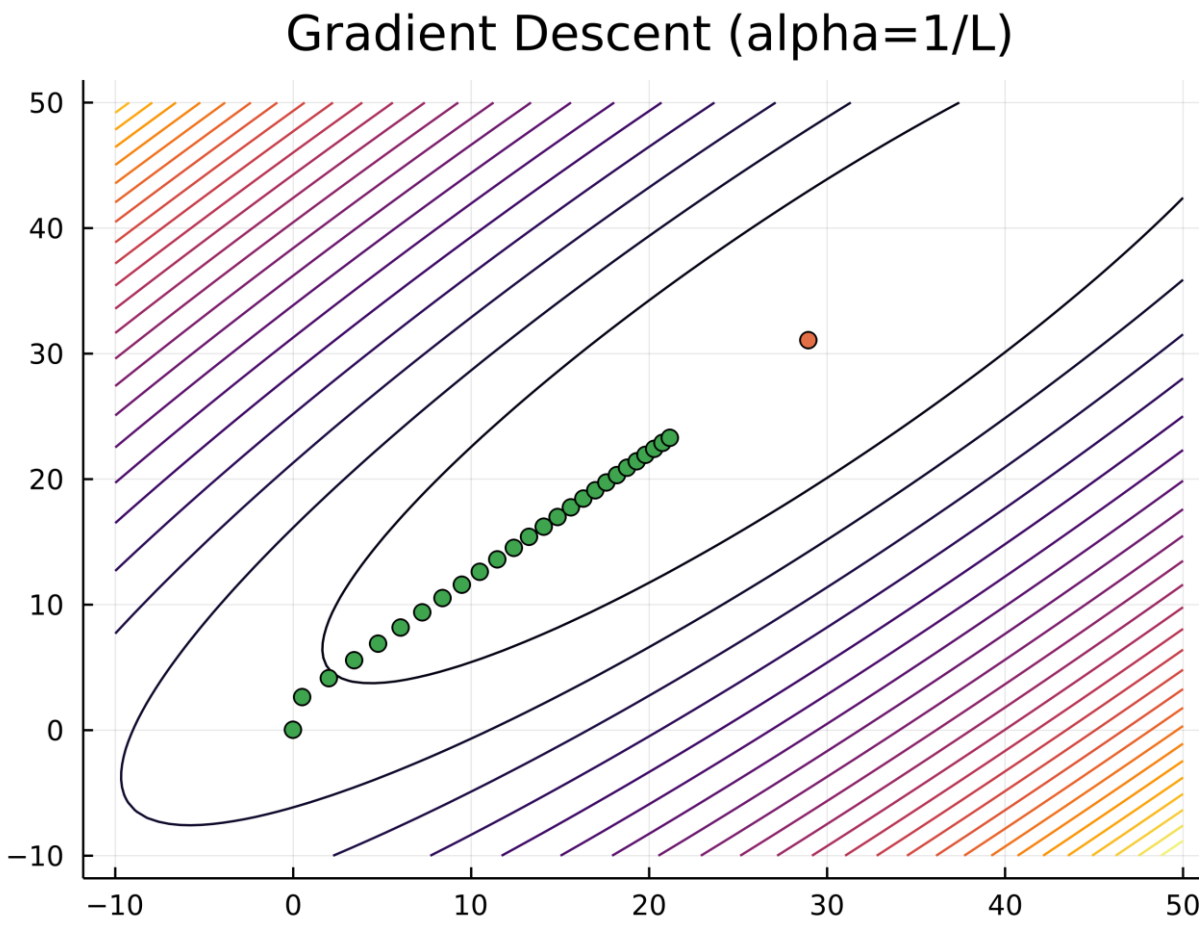
- GD vs. Adam:

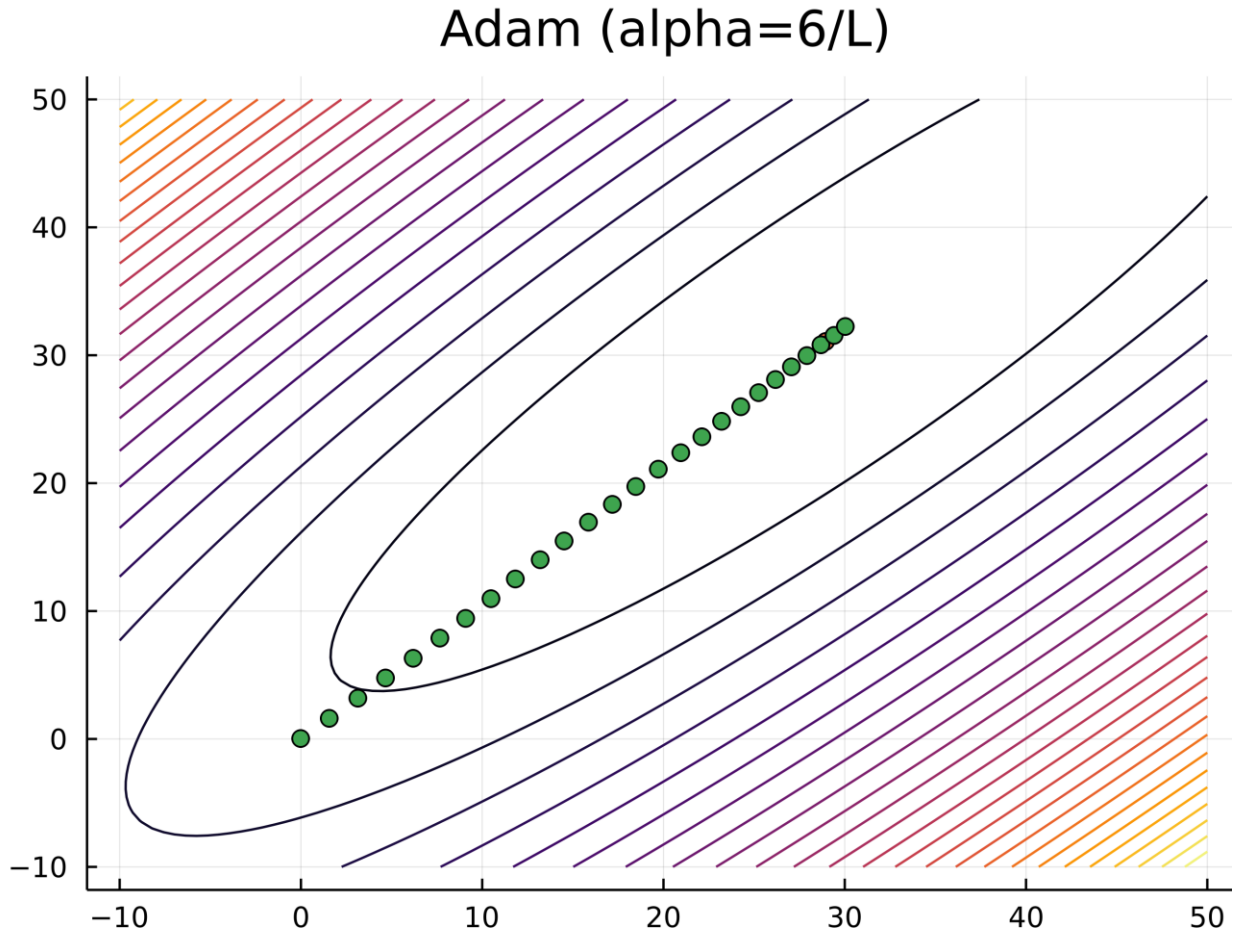




Gradient Descent vs. Adam

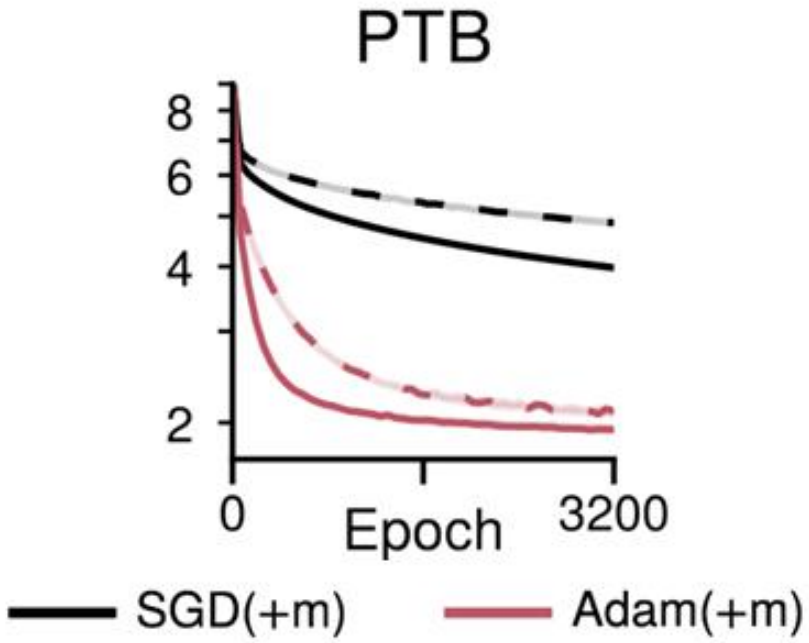
- GD vs. Adam:





Which Phase does Adam help with?

- Is Adam helping in gradient-dominated or noise-dominated phase?
  - We can change the batch size to vary the noise on a problem where Adam helps:

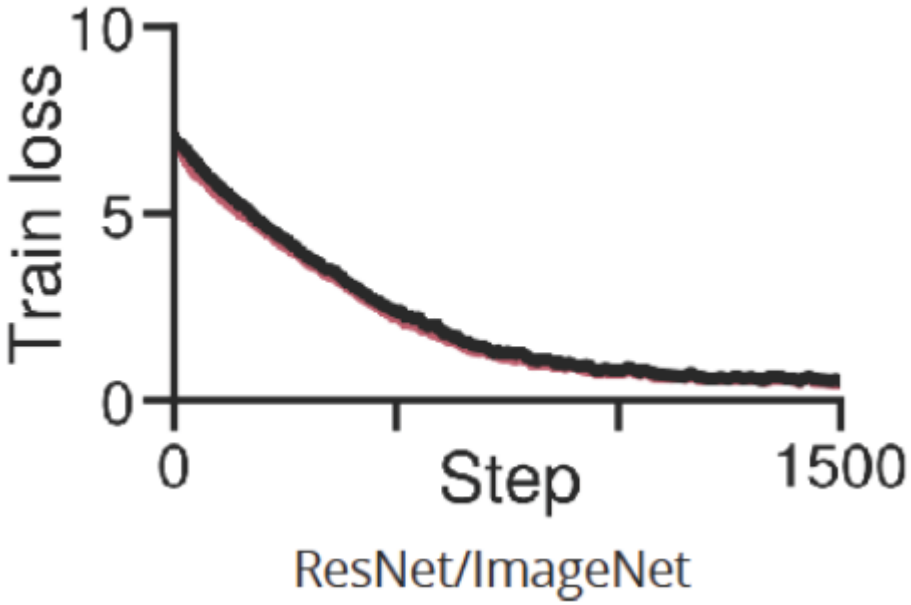


- For small batch sizes, Adam does not help.
- For large batch sizes, gap between Adam and SGD grows.
  - Adam is helping in gradient-dominated phase, not noise-dominated phase.

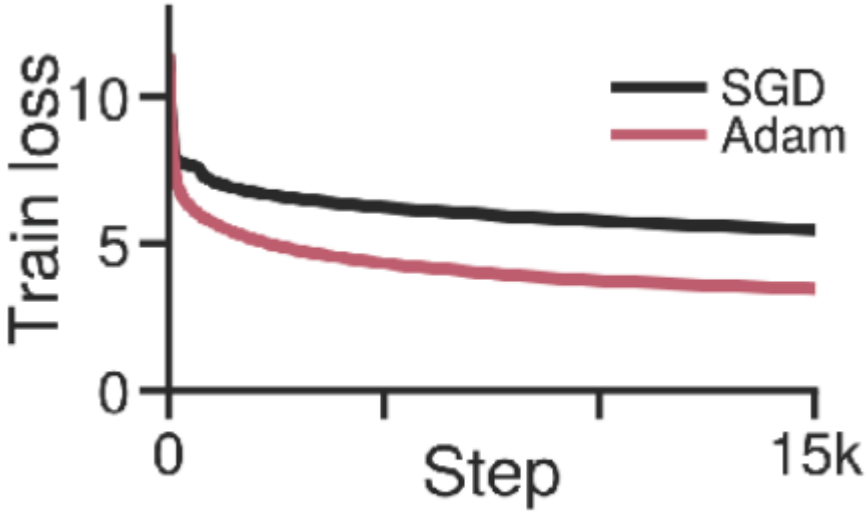
[Kunstner et al., 2023]

Adam on Natural Language vs. Computer Vision

- Adam versus SGD on vision compared to language data:
  - Adam does not tend to outperform SGD on computer vision benchmarks.
  - Adam tends to outperform SGD with a wide margin on language benchmarks.



ResNet/ImageNet



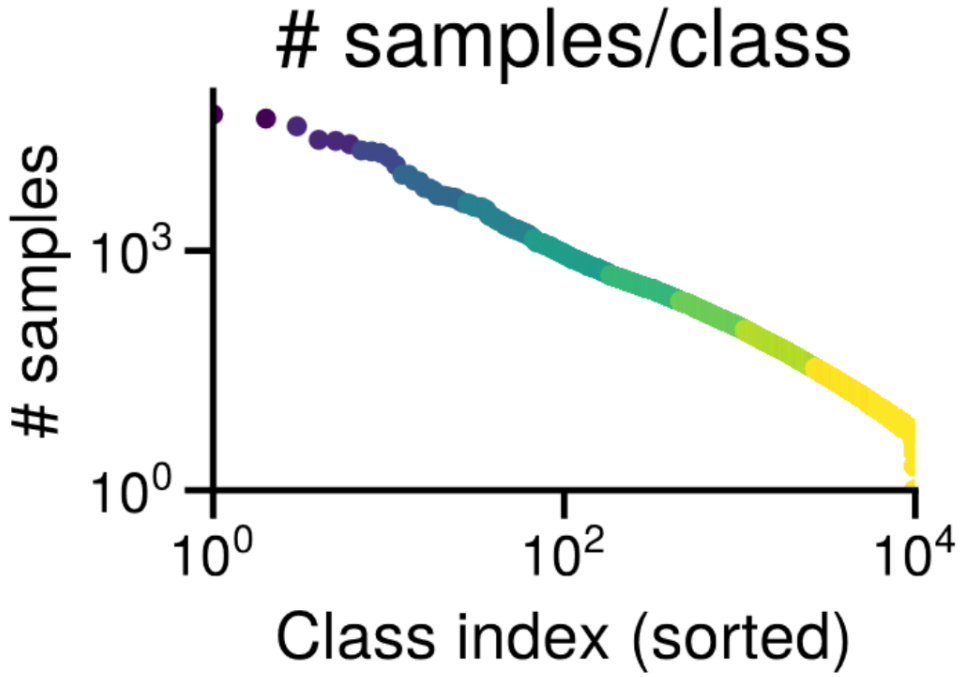
GPT2-small/WikiText-103

[Kunstner et al, 2024]

Adam on Natural Language vs. Computer Vision

- Classic explanation for effectiveness of AdaGrad/Adam:
  - Helps with sparse-but-informative features.
  - If a feature is important but usually 0, Adam takes big steps on the feature.

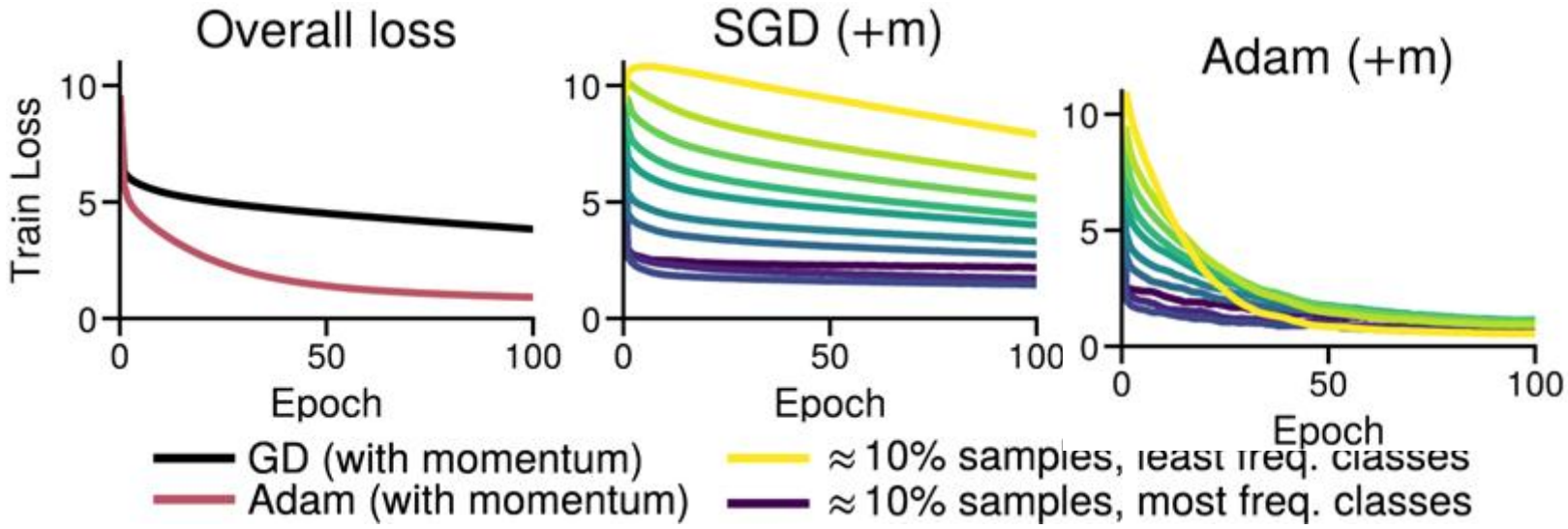
- Newer additional explanation:
  - Language models have heavy-tailed labels.
    - Word “the” appears frequently.
    - But most words are rare.



[Kunstner et al, 2024]

Adam vs. SGD with Heavy-Tailed Labels

- SGD makes slow progress on rare labels.
  - So if most examples have rare labels, SGD converges slowly.



- Adam makes similar progress on all labels.
  - So if most examples have rare labels, Adam still makes progress.
  - If you make vision datasets with heavy-tailed labels, Adam outperforms SGD.

[Kunstner et al, 2024]

Adam Surrogate: Sign Descent

- Adam update:

$$m_k = \beta_1 m_{k-1} + (1-\beta_1) g_k \qquad v_k = \beta_2 v_{k-1} + (1-\beta_2) g_k^2$$

$$\hat{m}_k = \frac{m_k}{1-\beta_1^k} \qquad \hat{v}_k = \frac{v_k}{1-\beta_2^k}$$

$$w_k = w_{k-1} - \frac{\alpha \hat{m}_k}{\sqrt{\hat{v}_k} + \epsilon}$$

  - Dynamics are hard to analyze.
- But if we set $\{\beta_1, \beta_2, \epsilon\}$ to 0 we get the simpler SignGD method:

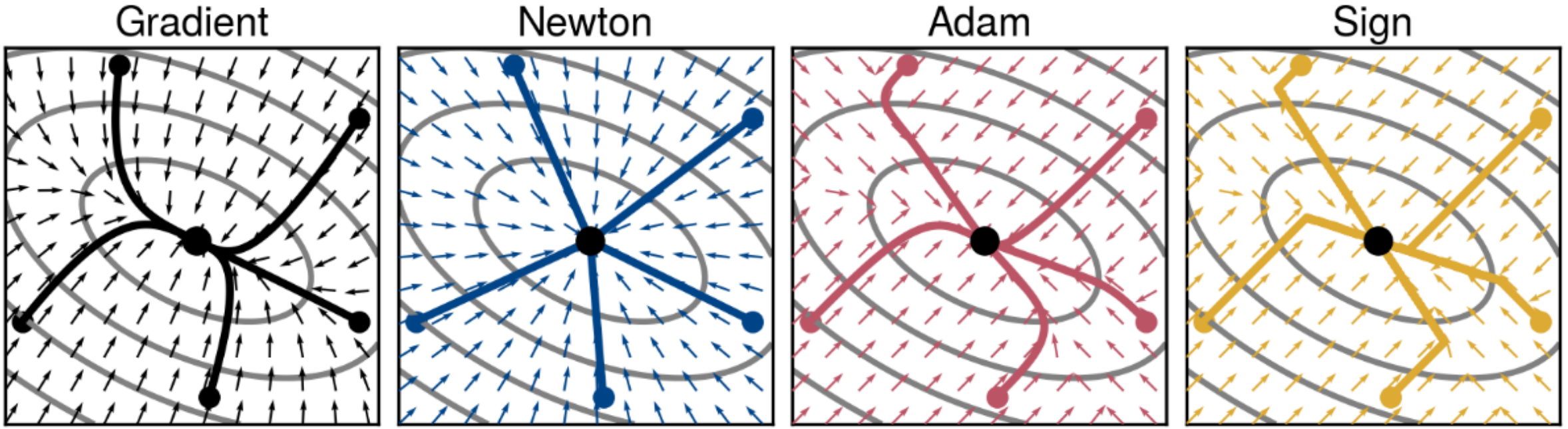
$$w_{k+1} = w_k - \alpha_k \text{sign}(\nabla f(w_k))$$

  - Where sign is +1 for positive values and -1 for negative values.
- Sign GD ignores magnitude of gradient.
  - Scales down large gradient values.
  - Scales up small gradient values.

[Kunstner et al, 2023]

Adam Surrogate: Sign Descent

- Comparing gradient flows (no momentum):



- SignGD is derived by optimizing descent lemma in different norm.

Plot from Fred Kunstner

Gradient Descent in Different Norms

- Recall the descent lemma:

$$f(v) \leq f(w) + \nabla f(w)^T (v - w) + \frac{L}{2} \|v - w\|^2$$

  - We used the descent lemma to derive step sizes for GD.
- But it can also be used derive gradient as direction to go:
  - With $w = w_k$, if you set w_{k+1} by optimizing v you get GD:

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

    - For a particular scalar α_k.
- Other norms for $\|v - w\|$ give different directions.

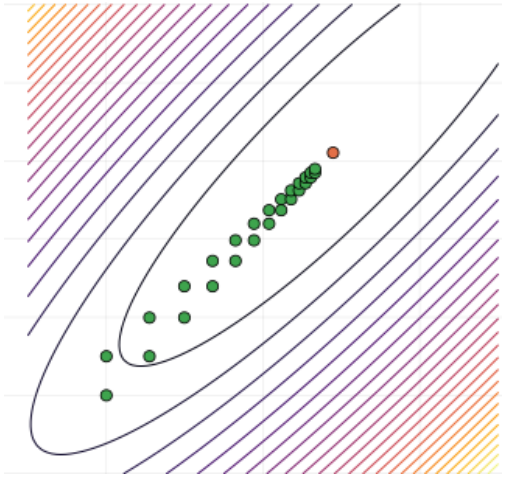
[Boyd & Vandenberghe, 2004]

Gradient Descent in 1-Norm

- Consider the descent lemma measuring distance in 1-norm:

$$f(v) \leq f(w) + \nabla f(w)^T (v - w) + \frac{L_1}{2} \|v - w\|_1^2$$

  - Where L_1 is smallest coefficient that makes inequality hold ($L_1 \leq L$).
- Setting w_{k+1} by optimizing v gives greedy coordinate descent.
  - GD step on variable with biggest derivative, with bigger step size than $1/L$.



  - Recent work on block version for training LLMs ("BlockLLM").

[Dhillon et al., 2011, Nutini et al., 2015, Varshini et al., 2024]

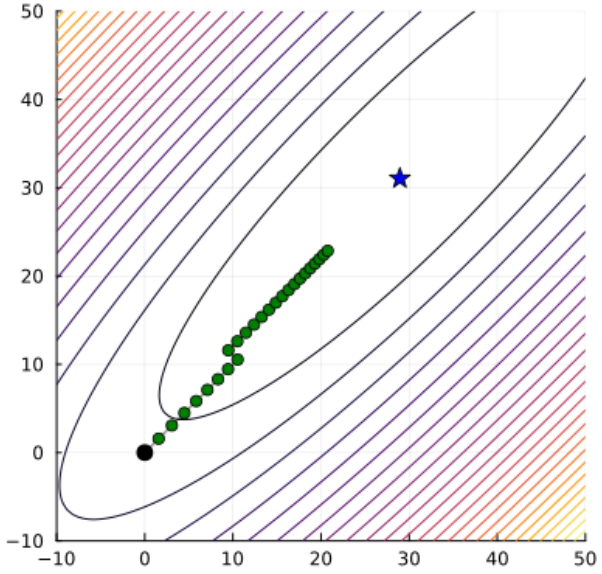
Gradient Descent in Inf-Norm

- Consider the descent lemma measuring distance in Inf-norm:

$$f(v) \leq f(w) + \nabla f(w)^T (v - w) + \frac{L_\infty}{2} \|v - w\|_\infty^2$$

- Setting w_{k+1} by optimizing v gives SignGD:

$$w_{k+1} = w_k - \alpha_k \text{sign}(\nabla f(w_k))$$

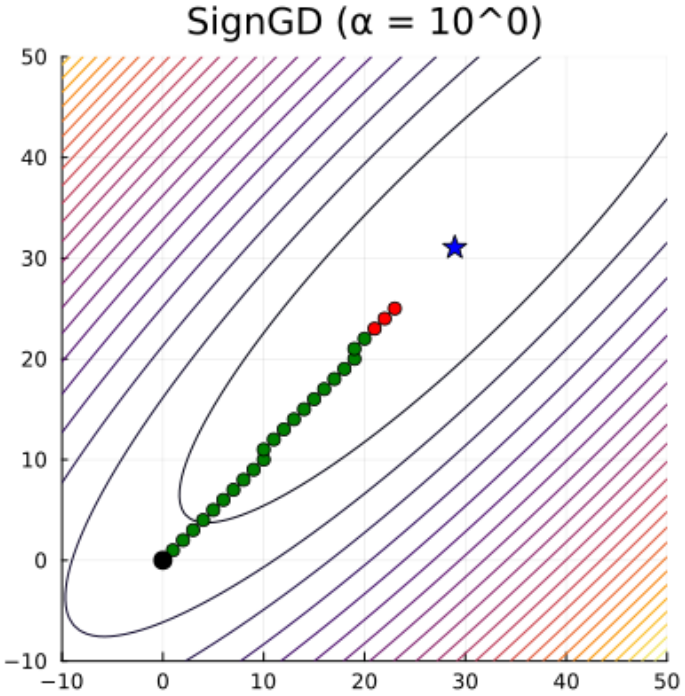


[Karimi et al., 2016]

Step Sizes for SignGD

- From the ∞-norm descent lemma, SignGD decreases f if:

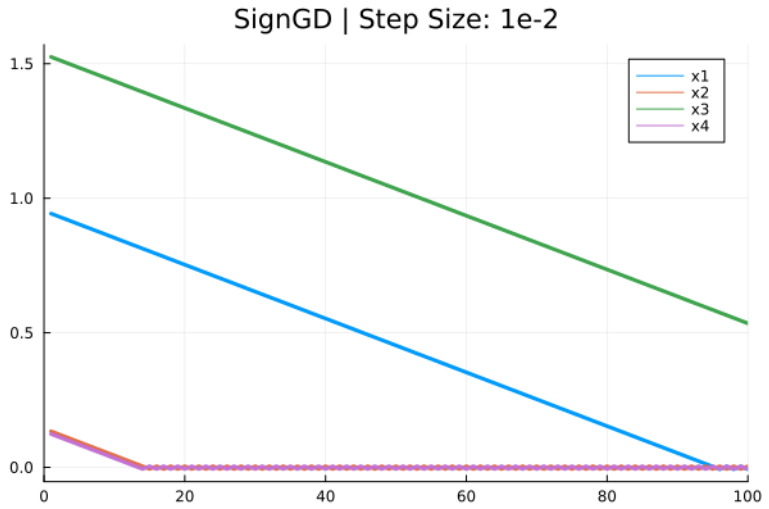
$$\alpha_k \leq \frac{2}{L_\infty}\|\nabla f(w_k)\|_1$$

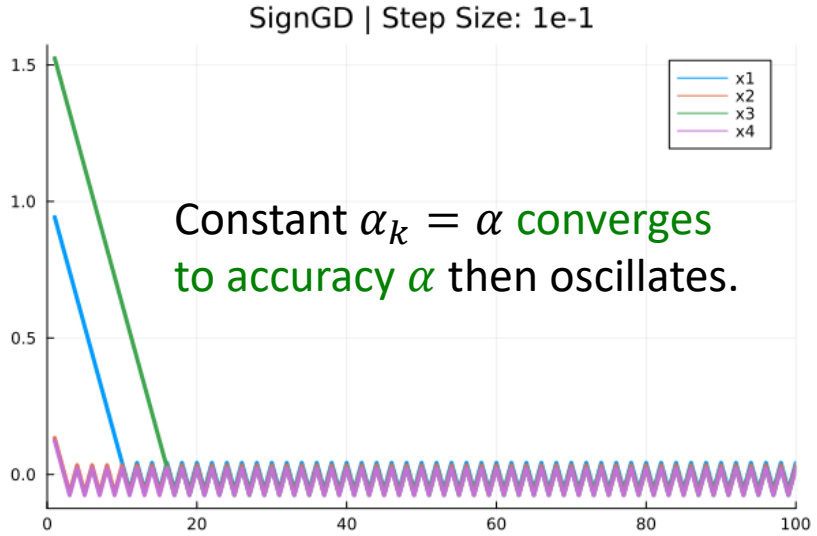


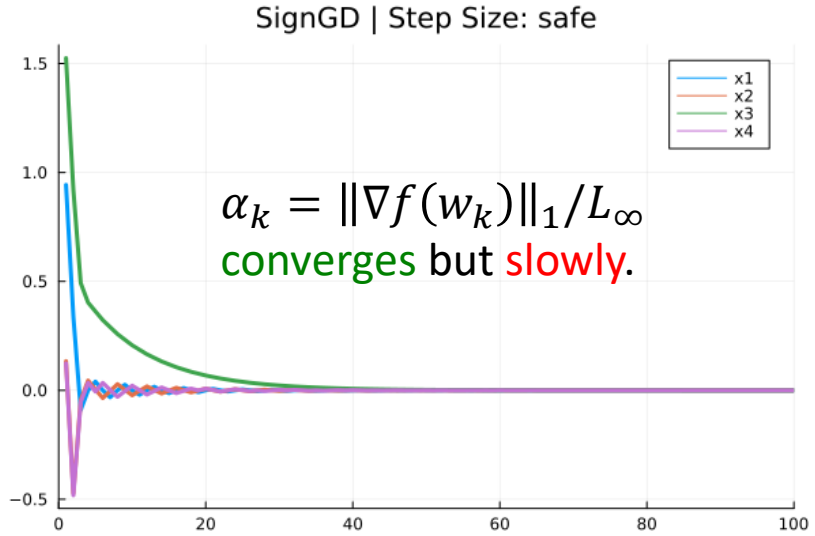
- Safe step sizes depend on gradient.
  - Large gradients allow large step sizes.
  - Small gradients require small step sizes.

- SignGD does not converge with fixed α_k.
  - We need scaling because we lost gradient magnitude.
  - But I do not recommend scaling by the 1-norm.
    - Gives absurd steps if 1-norm is large, not needed for any fixed accuracy.

SignGD on Diagonal Quadratic

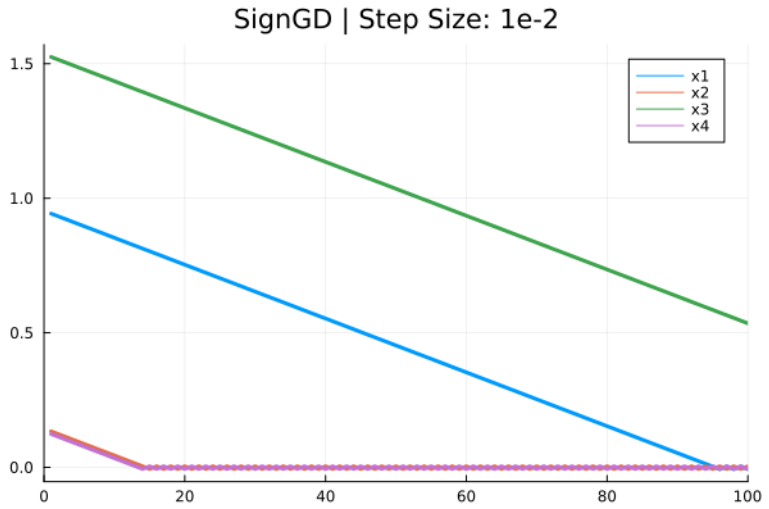


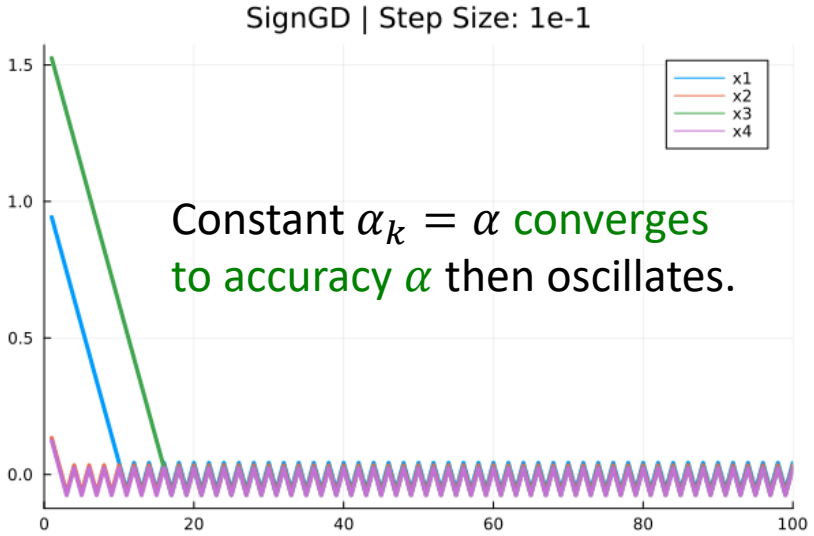




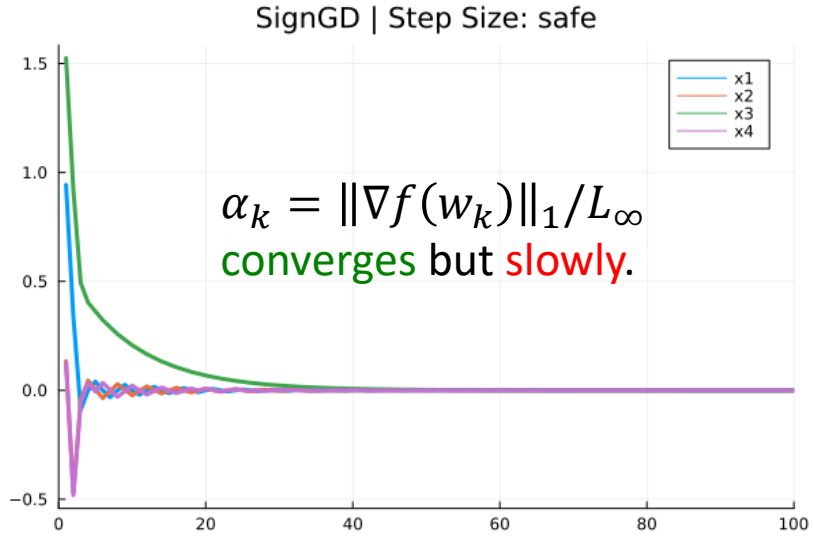
- Dynamics with constant $\alpha_k = \alpha$ are simple:
  - Until the sign changes we have: $w_j^k = w_j^0 - \alpha k \cdot \mathrm{sign}(w_j^0)$
  - After that point it oscillates in $[-\alpha, \alpha]$.
    - Steps do not converge, but can reach accuracy ϵ as long as $\alpha < \epsilon$.
      - Number of iterations only depends on α_k and distance to solution.

SignGD on Diagonal Quadratic





Constant $\alpha_k = \alpha$ converges to accuracy α then oscillates.



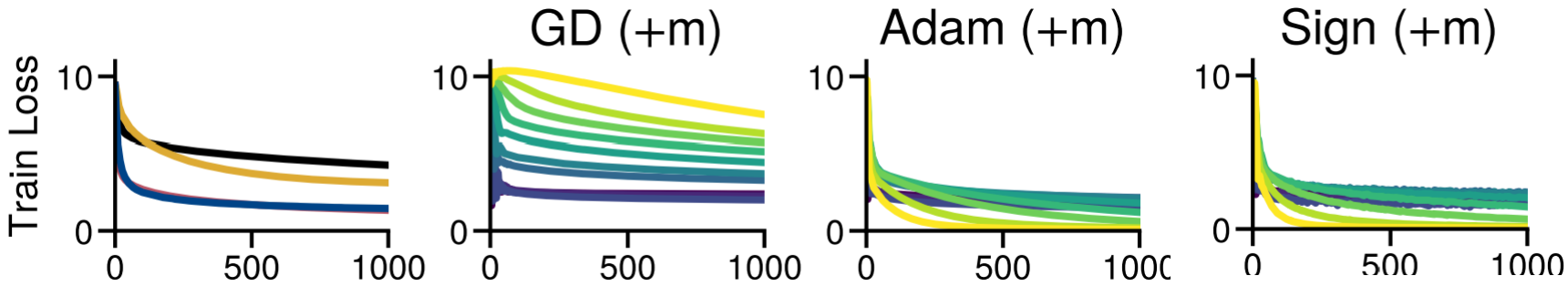
$\alpha_k = \|\nabla f(w_k)\|_1 / L_\infty$ converges but slowly.

- SignGD moves at same speed along valleys and ravines.
- Trade-off: faster than GD along ravines but slower on valleys.
  - Heavy-tailed assumption: big proportion of ravines that dominate f value.

Adam Surrogate: Sign Descent

- SignGD with momentum behaves similar to Adam (deterministic).

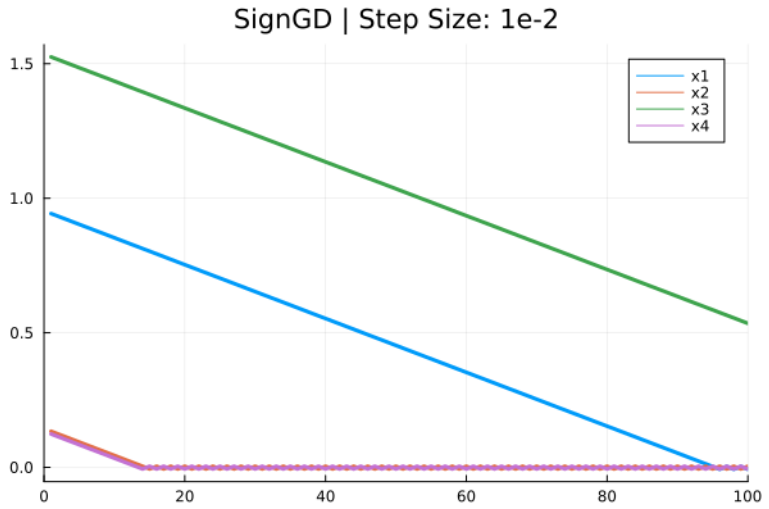
$$m_{k+1} = \beta_k m_k - \alpha_k \text{sign}\left(\nabla f(w_k)\right)$$
$$w_{k+1} = w_k + m_{k+1}$$

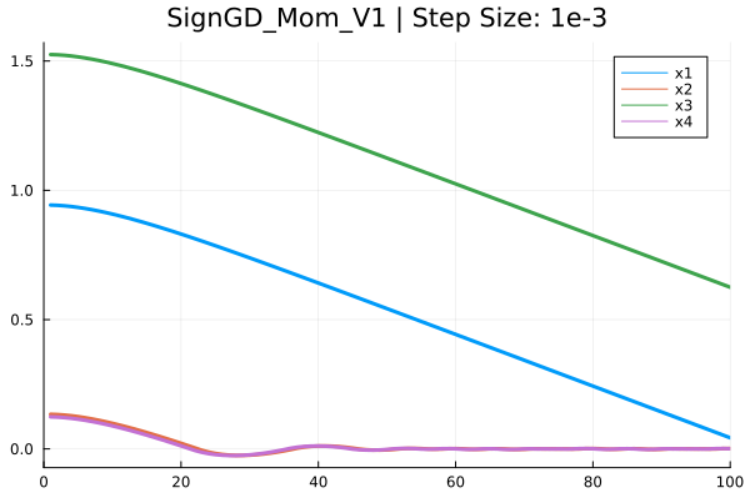


  - Variation is signum (take sign of momentum) and later Lion.

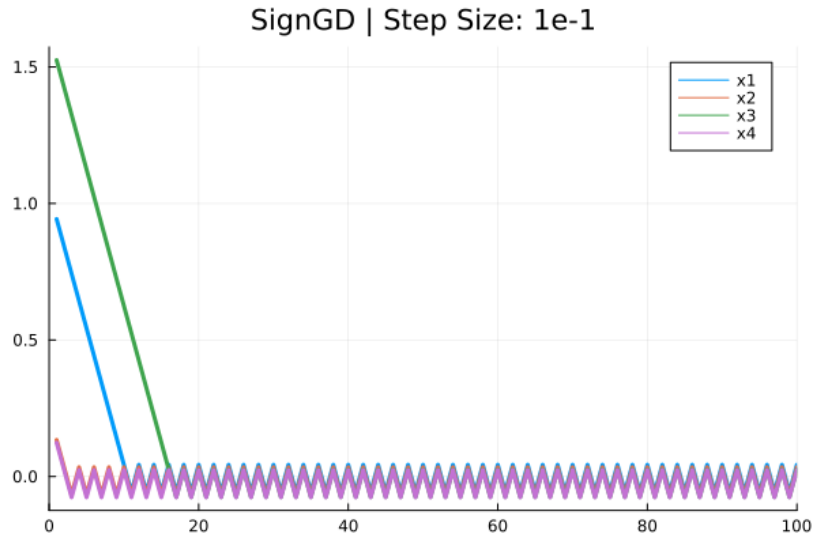
[Bernstein et al., 2018, Chen et al., 2023 Kunstner et al., 2023]

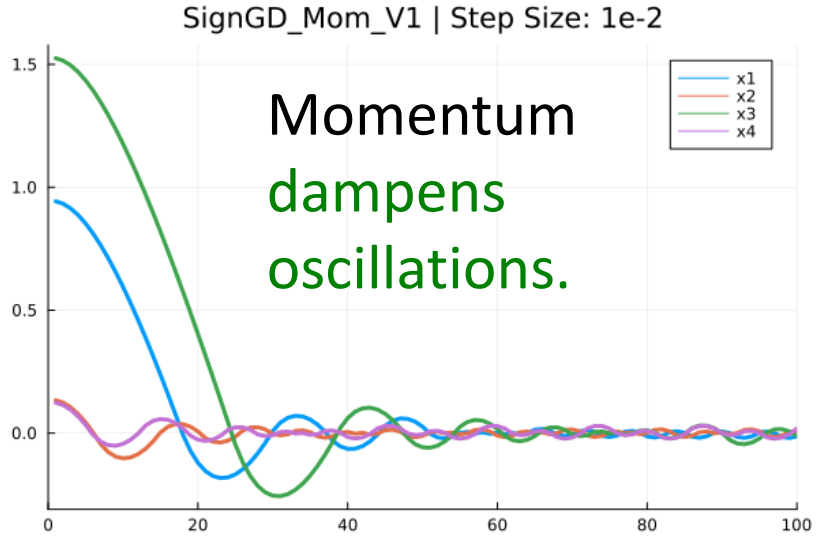
SignGD vs. SignGD + Momentum vs. Sign(Momentum)



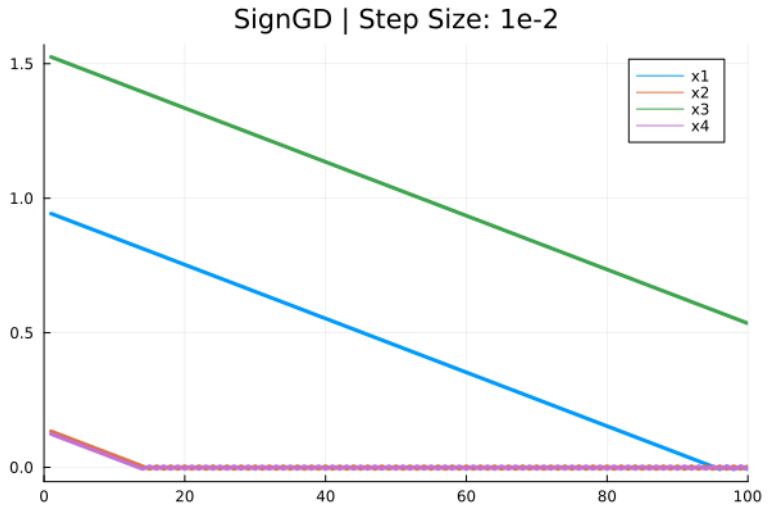


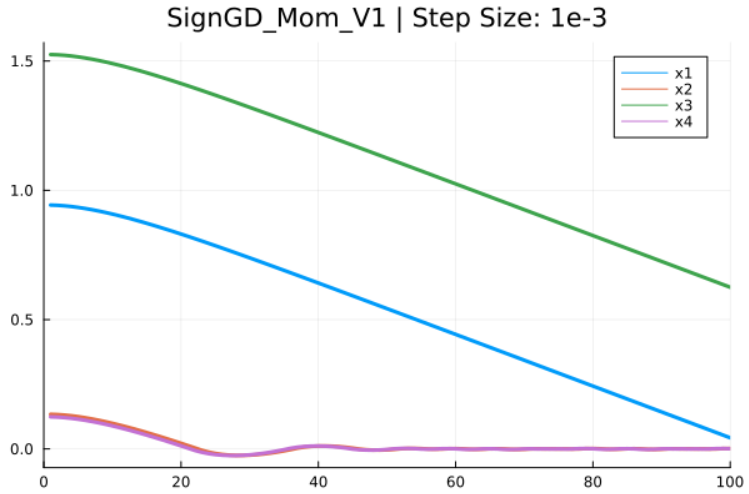
Momentum allows smaller α:
- Speeds up to "terminal velocity" of $\alpha/(1-\beta)$.

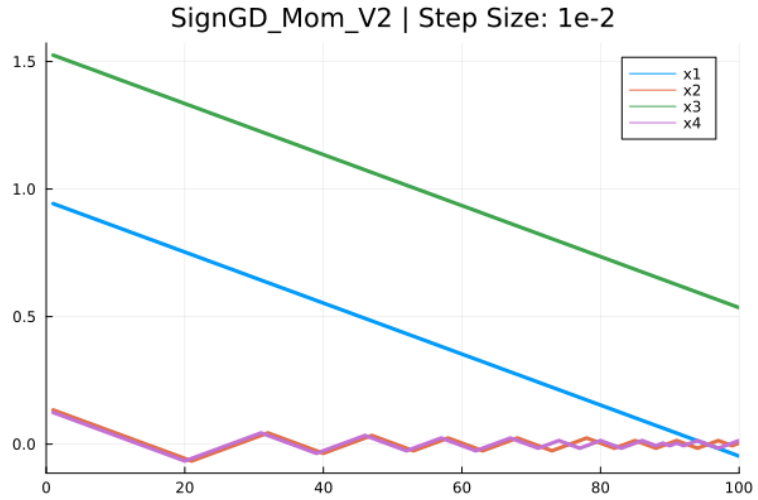


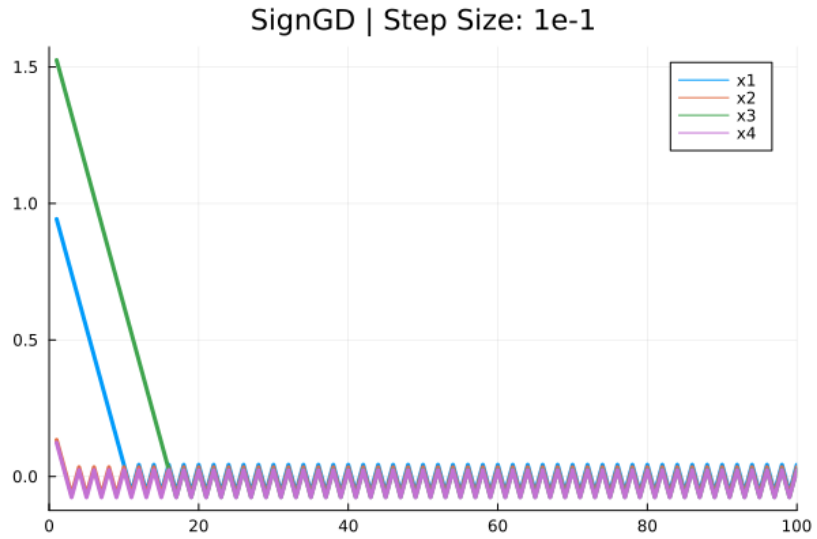


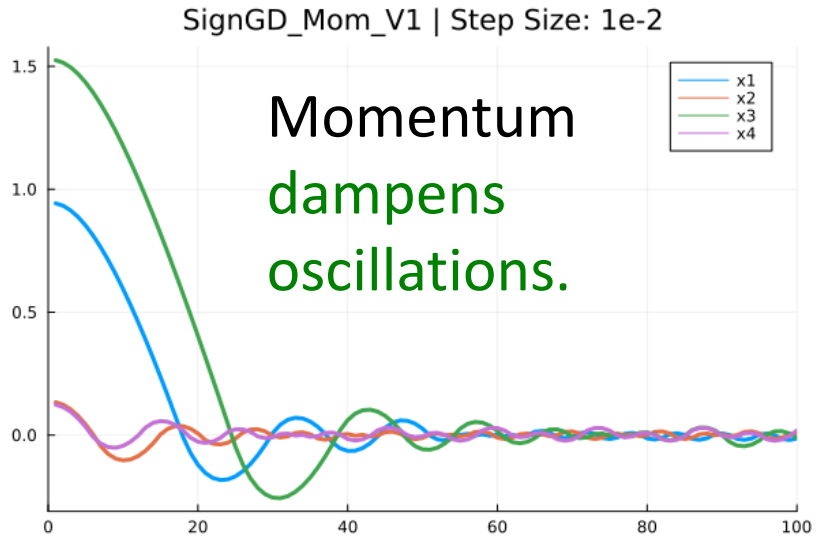
SignGD vs. SignGD + Momentum vs. Sign(Momentum)



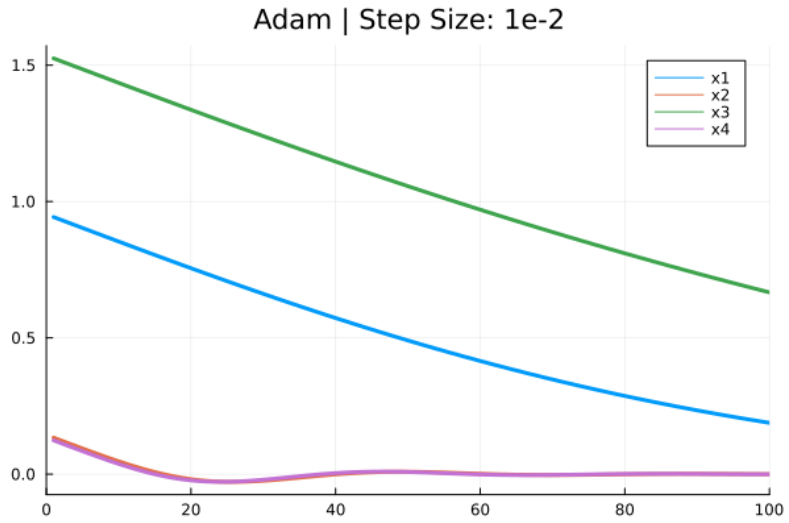


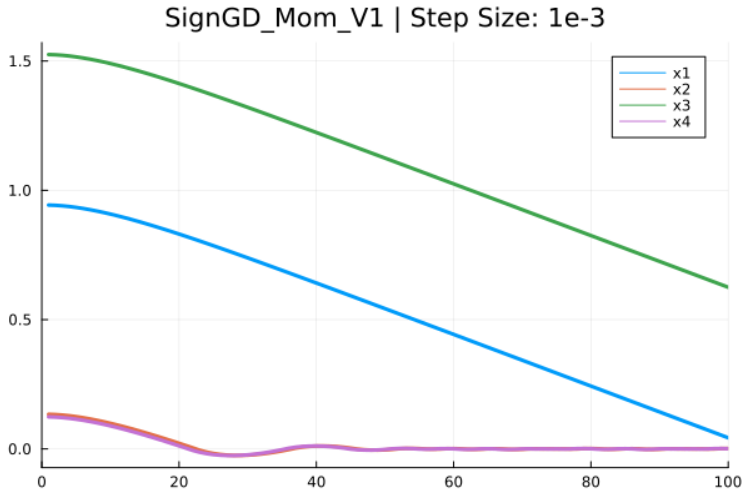


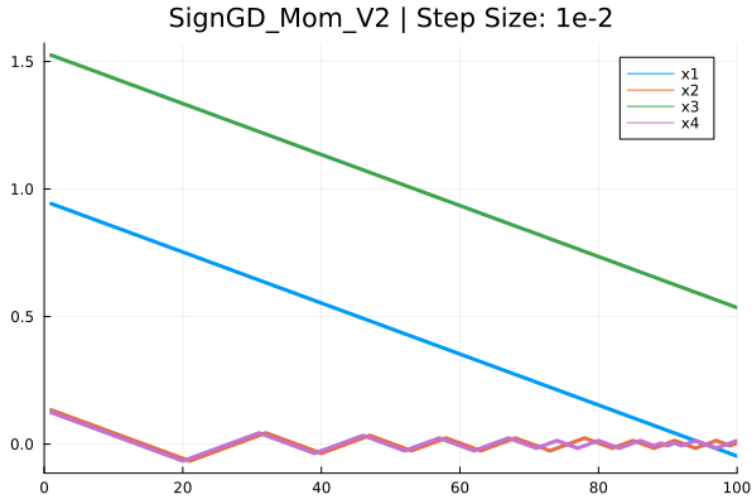


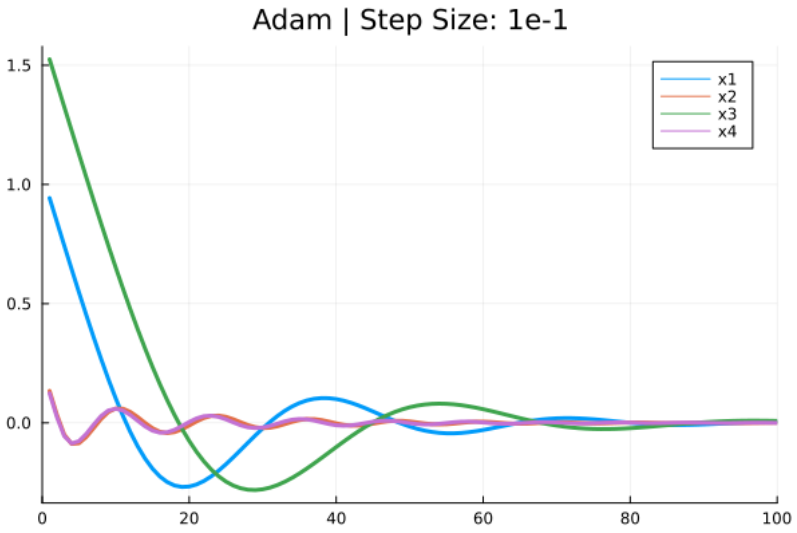


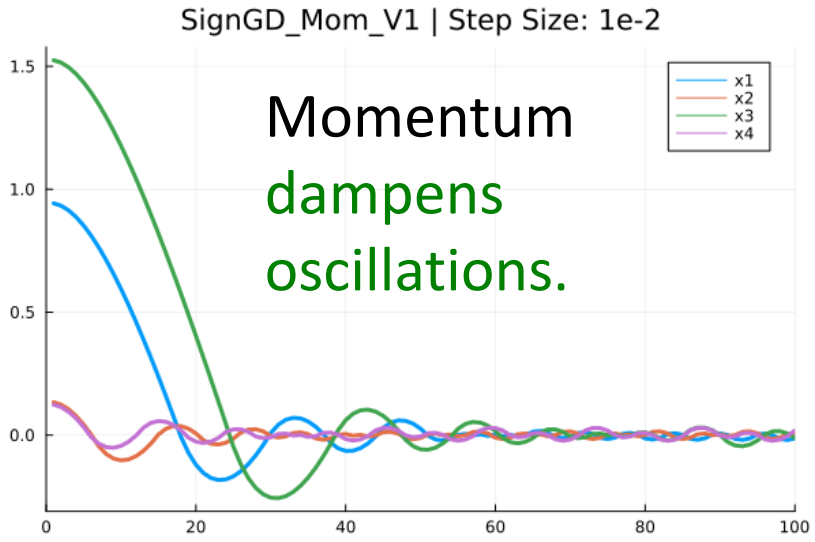
Adam vs. SignGD + Momentum vs. Sign(Momentum)

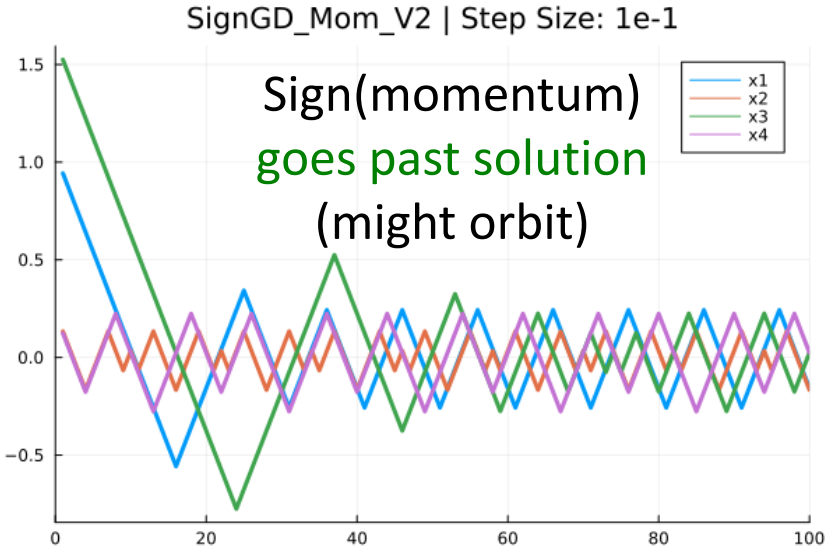






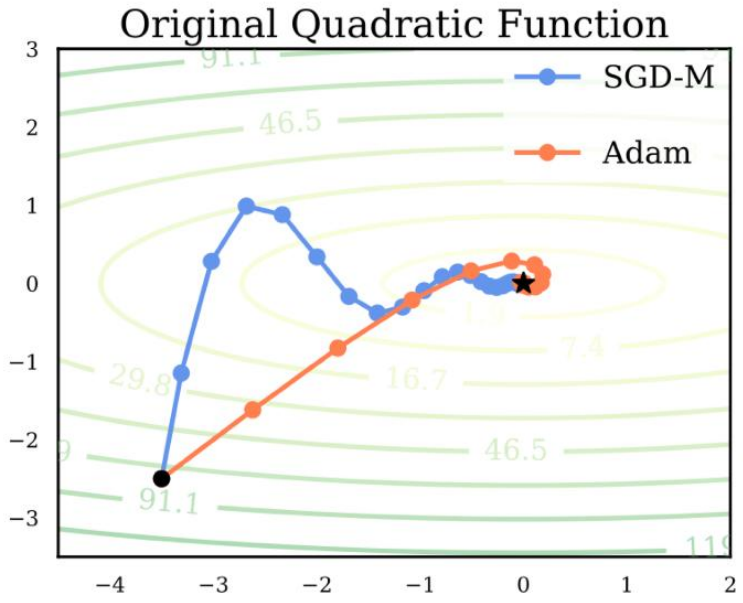


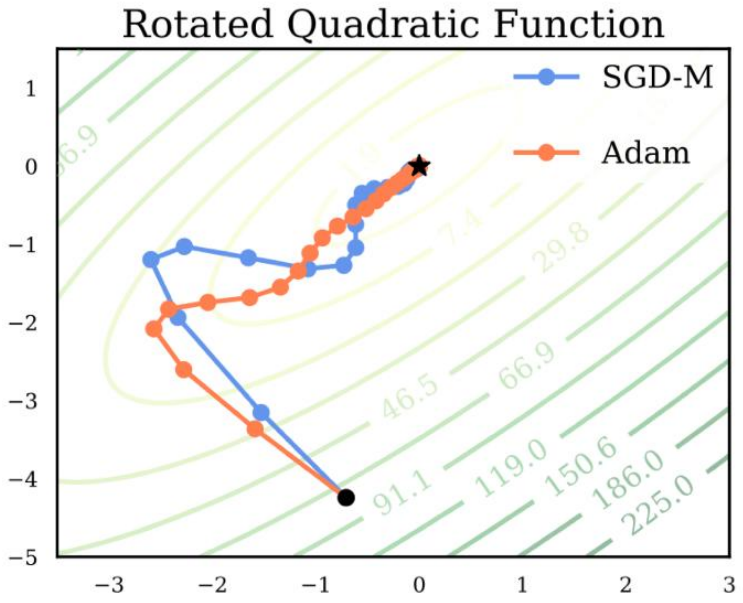




Rotation Dependence

- But unlike GD, SignGD and Adam are not rotation invariant.
  - If you rotate the function, algorithm changes.



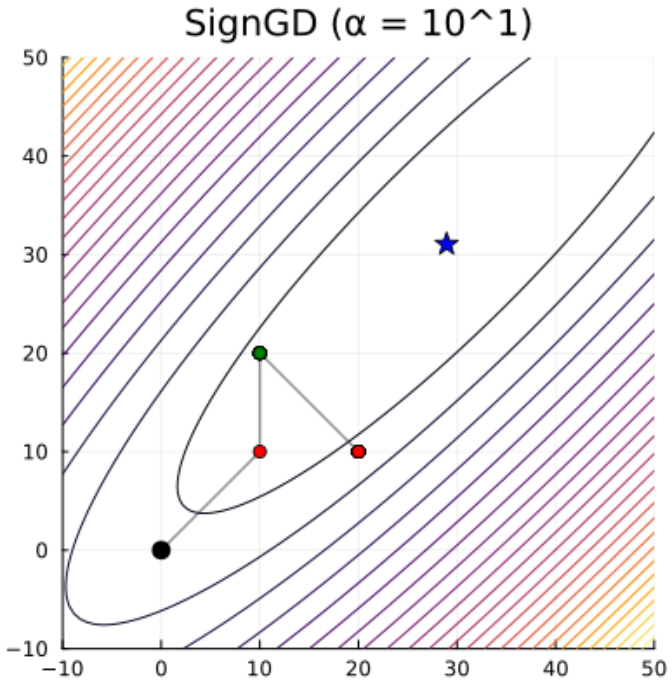


  - So diagonal case does not show full dynamics.

[Zhang et al., 2025]

Rotation Dependence

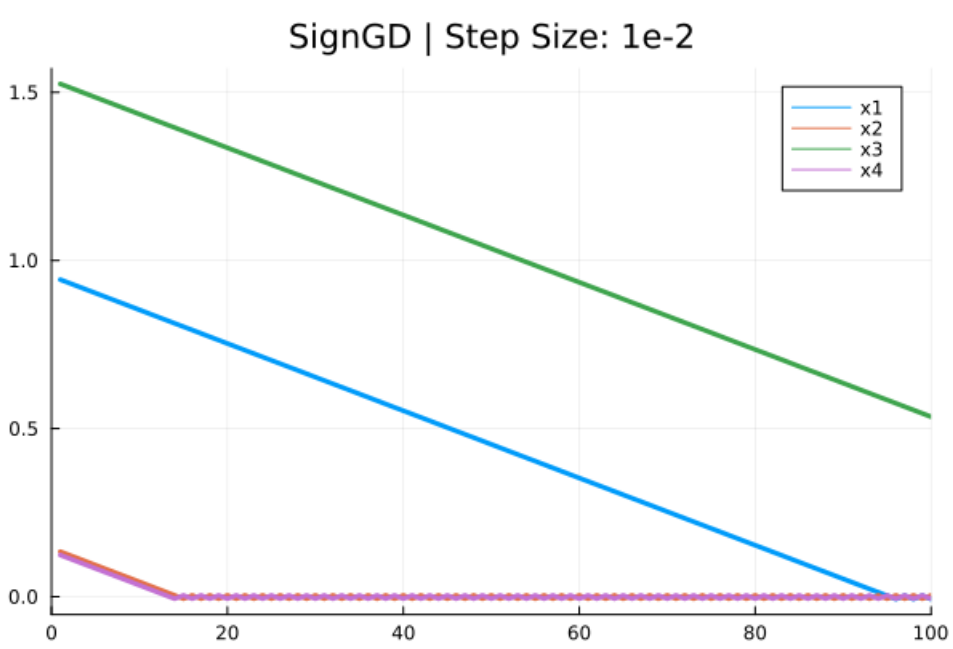
- But unlike GD, SignGD and Adam are not rotation invariant.
  - If you rotate the function, algorithm changes.

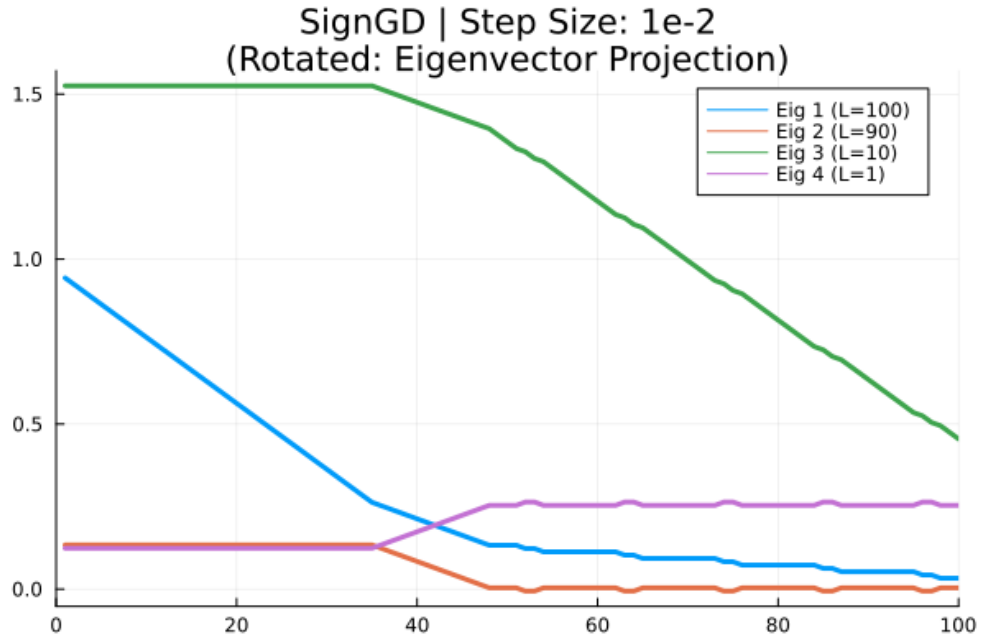


  - So diagonal case does not show full dynamics.
    - In non-diagonal case, oscillations are not necessarily around solution.

Rotation Dependence

- Performance along eigenvectors of diagonal vs rotated quadratic:





– L_∞ is much larger for rotated quadratic (201 vs. 400): need smaller α.

- In rotated quadratic, steps along different eigenvectors are cancelling with this α.

Progress for Non-Diagonal Case

- To look at progress in non-diagonal case, let's go back to Taylor:

$$f(v) \approx f(w) + \nabla f(w)^T (v - w) + \frac{1}{2}(v - w)^T \nabla^2 f(w)(v - w)$$

- With the GD update this gives:

$$f(w_{k+1}) \approx f(w_k) - \alpha_k \|\nabla f(w_k)\|^2 + \frac{1}{2}\alpha^2 \nabla f(w_k)^T \nabla^2 f(w_k) \nabla \mathrm{f}(\mathrm{w_k})$$

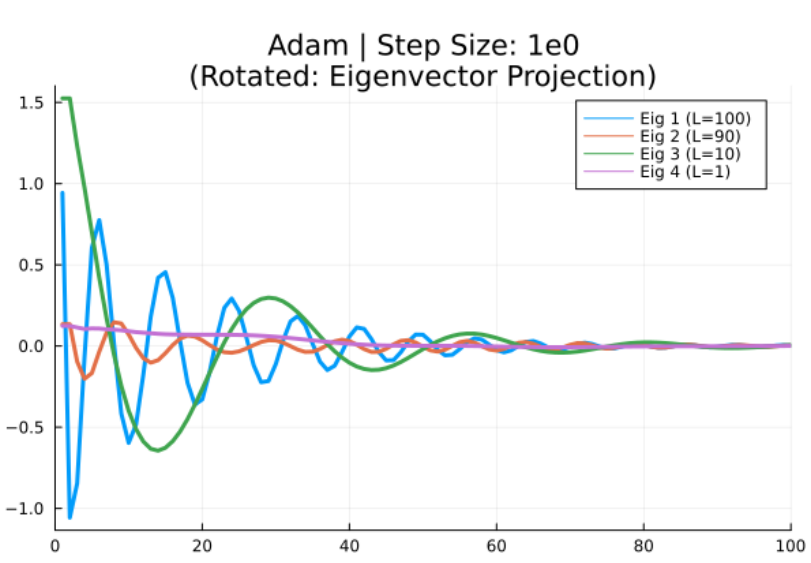
- With the SignGD update this gives:

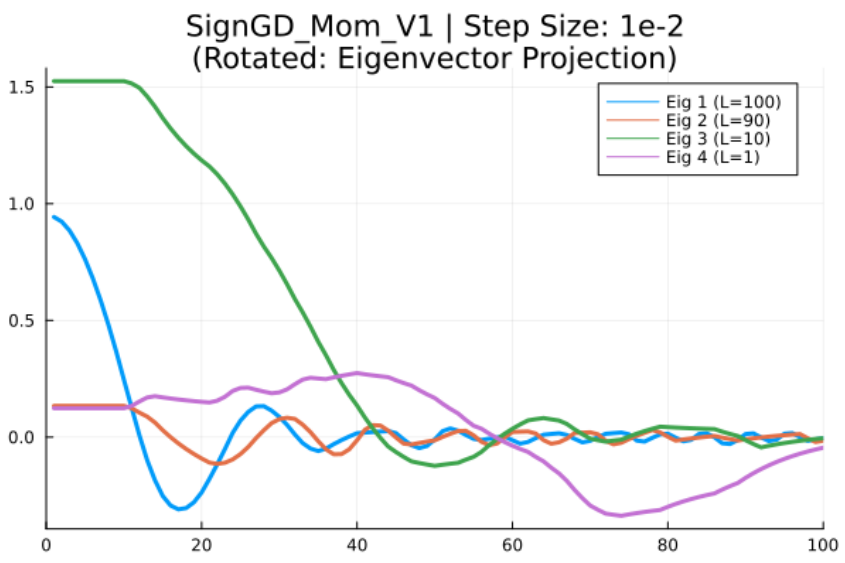
$$f(w_{k+1}) \approx f(w_k) - \alpha_k \|\nabla f(w_k)\|_1 + \frac{1}{2}\alpha^2 \mathrm{sign}(\nabla f(w_k))^T \nabla^2 f(w_k) \mathrm{sign}(\nabla \mathrm{f}(\mathrm{w_k}))$$

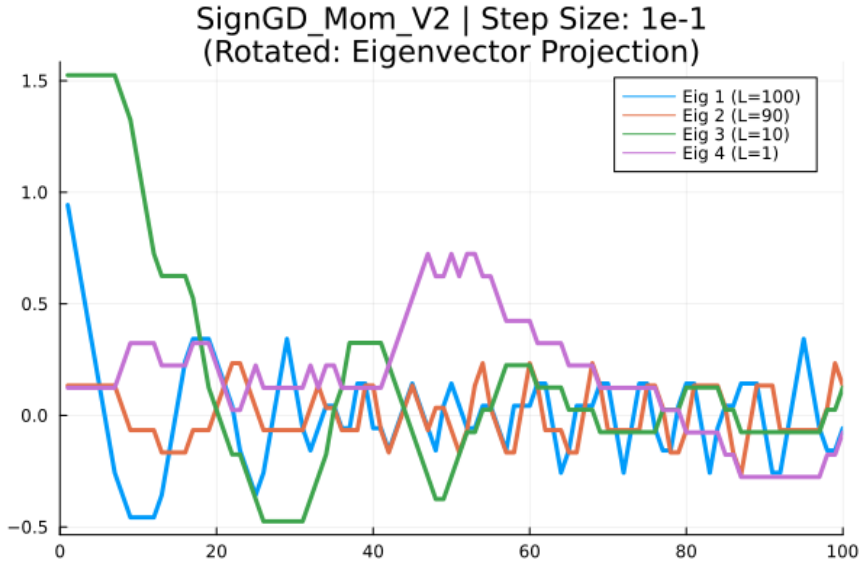
- For small α_k, SignGD makes more progress if 1-norm larger than 2-norm.
  - Happens when gradient has many small values.
- For large α_k, SignGD is helped out if Hessian is closer to diagonal.

SignGD with Momentum (Rotated Case)

- Adding momentum seems to help with rotations:







- No idea why! Adam also seems more stable at larger step sizes.

SignGD/Adam Scale Invariance

- While SignGD is not rotation invariant, it is scale invariant:
  - If we optimize $\gamma f(w)$ instead of $f(w)$, updates are the same.
    - Sign($\nabla f(w)$) = Sign($\gamma \nabla f(w)$) (for positive γ).
  - Get same performance under scalings of function.

- Adam with default parameters is not scale invariant.

- Adam with $\beta_1 = \beta_2$ is scale invariant ($\epsilon = 0$, no bias correct).
  - Gives natural online update for estimating mean and variance.
  - Recent works argue for using $\beta_1 = \beta_2$.

[Orvieto & Gower, 2025, Fernández-Hernández et al., 2026]

Stochastic SignGD

- Stochastic SignGD update would take the form:

$$w_{k+1} = w_k - \alpha_k \text{sign}(\nabla f_{i_k}(w_k))$$

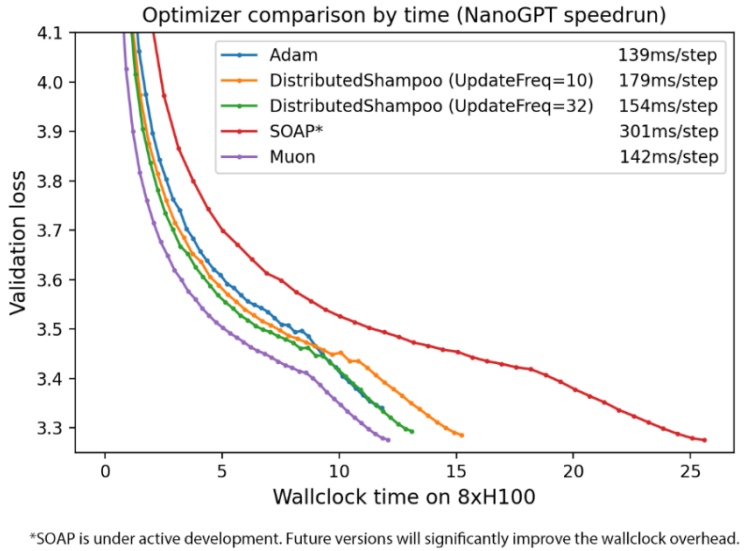
- Nice property is that it has clipping built in:
  - Helps with non-uniform smoothness and heavy-tailed noise.
  - Momentum variants can help with noise in early iterations.
- On the other hand, this is biased:
  - $\mathbb{E}[\text{sign}(\nabla f_i(w_k)] \neq \text{sign}(\nabla f(w_k))$.
    - Though bias will decrease with batch size.
- SignGD/Adam likely most useful in gradient-dominated phase.
  - "Folk" wisdom is that it "helps to switch to SGD late in training".

[Sun et al., 2023]

Question: What is Muon doing?

Speedrunning and Muon

- Recent trend towards speedrunning of benchmarks.
  - Trying to break speed record for training models.



*SOAP is under active development. Future versions will significantly improve the wallclock overhead.

- Promising approach coming out of this community is Muon.
  - Related to earlier work on spectral descent.

Steepest Descent in Matrix Norms

- For many problems parameters are naturally written as matrices.
  - In this setting we can consider gradient descent under matrix norms:

$$f(W_{k+1}) \leq f(W) + \nabla f(W)^T (W_{k+1} - W) + \frac{L_*}{2} \|W_{k+1} - W\|_*^2$$

  - Frobenius norm: gradient descent on matrix elements.
  - Nuclear norm: rank-1 update based on principal singular vector.
  - Operator norm: update in direction of polar decomposition.

$$W_{k+1} = W_k - \alpha_k UV^T$$

    - Which is Muon (without momentum).
      - Key practical ingredient is Newton-Schulz iterations to quickly compute polar decomposition.

[Carlsen et al., 2015, Jordan et al., 2024]

Muon Discussion

- Also get Muon if constrained to orthogonal update:
  - Enforcing $(W_{k+1} - W_k)^T(W_{k+1} - W_k) = I$ gives Muon (all 3 norms).
- Though search direction is orthogonal, weights W_k may grow.
  - Several works argue that Muon requires regularization.

- Can view Muon (no momentum) as SignGD on singular vectors
  - If we have a vector of parameters, Muon is SignGD.

- SignGD and Muon can also be derived as Frank-Wolfe methods.
  - But gives a slightly different update.

[Liu et al., 2025, Sfyraki & Wang, 2025]

Muon Discussion

- Much of our discussion of SignGD carries over to Muon:
  - Can get safe step sizes (depending on size of gradient).
  - Can compare progress (Muon depends on 1-norm of singular values).

- Recent works consider different norms for different layer.
  - Scion and Gluon.
  - Same steps can give insights on how these methods can help.

[Bernstein & Newhouse, 2025, Davis & Drusvyatskiy, 2025, Pethick et al., 2025, Riabinin et al., 2025]

Question: can second-order information help?

Preconditioning

- Preconditioned GD multiplies gradient by a matrix P_k:

$$w_{k+1} = w_k - \alpha_k P_k^{-1} \nabla f(w_k)$$

- Exist safe step sizes if P_k is positive-definite (from descent lemma).
  - But our goal is to choose P_k to improved progress.

- Optimal preconditioner for quadratics: $P_k = \nabla^2 f(w_k)$.
  - With inverse Hessian preconditioner GD finds optimal solution in 1 step.

- But in most cases not obvious how to choose preconditioner.

What about Newton's Method?

- Newton's method uses Hessian $P_k = \nabla^2 f(w_k)$.
  - Classic property: ultra-fast convergence with $\alpha_k = 1$ near minimizers.

- Immediate issue: Hessian may not be positive-definite.

- Common fix to this is to use $P_k = \nabla^2 f(w_k) + \lambda_k I$.
  - Where λ_k makes P_k positive definite.
  - Traditional methods: trust-region and Levenberg-Marquadt.
  - More recent: cubic regularization based on 2nd-order descent lemma.

[Nocedal & Wright, 1999]

2nd-Order Descent Lemma

- The 2nd-order descent lemma:

$$f(v) \leq f(w) + \nabla f(w)^T (v - w) + \frac{1}{2}(v - w)^T \nabla^2 f(w)(v - w) + \frac{M}{6}\|v - w\|^3$$

  - Where M is Lipschitz bound on the Hessian.
    - More accurate model of $f(v)$ if w and v are close (not bounding Hessian by sharpness).

- Using $w = w_k$ and setting w_{k+1} as minimal v gives regularized Newton:

$$w_{k+1} = w_k - (\nabla^2 f(w_k) + \lambda_k I)^{-1} \nabla f(w_k)$$

  - Where λ_k depends on M (and M is usually set by SUG).

- Neat property: escapes saddle points.
  - If $\|\nabla f(w_k)\| = 0$ (or small), escapes in negative curvature direction.

[Griewank, 1981, Nesterov & Polyak, 2006, Cartis et al., 2009]

First- vs. Second-Order Methods

- If we plug in GD with the optimal $\alpha_k = 1/L$ into the descent lemma we get:

$$f(w_{k+1}) \leq f(w_k) - \frac{1}{2L}\|\nabla f(w_k)\|^2$$

  - How much we are guaranteed to decrease f.
- We plug in regualarized Newton into 2nd-order descent lemma we get:

$$f(w_{k+1}) \leq f(w_k) - \frac{1}{6M^2}\|\nabla f(w_k)\|^{\frac{3}{2}}$$

- Notice that Newton is not necessarily better.
  - In order for Newton to guarantee more progress, need $\|\nabla f(w_k)\| < \frac{L^2}{9M^4}$.

- If gradient is large or Hessian is changing quickly, Newton can be worse than GD.

- Newton helps when:
  - Hessian is stable (small M like quadratics) or gradient is small (near minimizer or stationary).

[Nesterov & Polyak, 2006]

Stochastic Newton Methods?

- A decades-old idea is to use a stochastic Newton method.
  - But despite decades of trying it has never become popular.

- We have reasons to be skeptical:
  - Newton helps most when gradient is small.
  - But this is noise-dominated region (where curvature is not issue).
    - Indeed Polyak-Ruppert result is that Newton cannot beat average(SGD) here.

- Though stochastic Newton can make sense under interpolation.
  - Noise goes to 0 so curvature is still bottleneck.

[Meng et al., 2020]

Diagonal Preconditioner Per-Variable Step Sizes

- Diagonal preconditioners correspond to variable step sizes.
  - AdaGrad, RSMprop, Adam, and so on.
- An old idea:
  - Update step sizes based on gradient with respect to step sizes.
    - "Hyper-gradient" or "meta-gradient".
- Has been reinvented many times (IDBD) but hard to get working.
- Recent advances:
  - Multi-dimensional backtracking: extends SUG to multiple coordinates.
    - Proves that gets within factor of $\sqrt{d}$ of optimal fixed preconditioner.
  - Online scaling: use ideas from online learning in hyper-gradient update.
    - Behaves as well asymptotically as optimal fixed preconditioner.
  - Meta-optimize: practical hyper-gradient descent implementation.

[Sutton, 1992, Duchi et al., 2011, Hinton, 2012, Baydin et al., 2018, Kunstner et al., 2023, Gao et al., 2024, Sharifnassab et al., 2025]

Newton without the Hessian

- Hessian-free Newton and quasi-Newton avoid Hessian calculation.
  - Popular in optimization literature, less so for neural networks.
    - Found Hessian-free hard to get working, small gains over Adam from quasi-Newton.
- Gauss-Newton: method specifically for non-linear least squares.

$$f(w) = \frac{1}{2}\sum_{i=1}^{n} \|g(w, x_i) - y_i\|^2 \qquad \nabla^2 f(w) = \sum_{i=1}^{n} J_w(x_i)^T J_w(x_i) + \sum_{i=1}^{n} (g(w, x_i) - y_i)\nabla^2 \mathrm{g}(\mathrm{w}, \mathrm{x_i})$$

  - Where g could be a neural network.
  - Approximates Hessian by Jacobian^{T}Jacobian term.
    - "Hessian with current features" (same spectrum as "neural tangent kernel").
    - Always positive semi-definite (easy to make positive definite).
    - Becomes exact if you interpolate the data (reach an error of 0).
      - But removed Hessian term may be what drives feature learning.

[Nocedal, 1980, Dembo & Steihaug, 1983, Martens, 2010]

What about Natural Gradient?

- In machine learning, we often discuss using the natural gradient.
  - Set P_k to Fisher Information matrix (viewing loss as a negative LL).

$$f(w) = \sum_{i=1}^{n} \log p(y_i \mid x_i, w) \qquad F(w_k) = \mathbb{E}_{y\sim p}\left[\nabla \log p(y_i|x_i,w) \nabla p(y_i|x_i,w)^T\right]$$

    - For linear models this is the Hessian, for non-linear least squares it is Gauss-Newton.
- In practice people use the empirical Fisher:

$$F(w_k) \approx \frac{1}{n}\sum_{i=1}^{n} \nabla p(y_i|x_i,w) \nabla p(y_i|x_i,w)^T$$

  - Not the natural gradient unless we use squared error or are interpolating.
    - Closely related to AdaGrad (accumulates instead of averages and takes square root).

[Amari, 1998, Kunstner et al., 2019]

Empirical Fisher with Structured Preconditioners

- Empirical Fisher often used with structured preconditioners:
  - Block-diagonal (precondition each layer indepdendently).
  - Kronecker-factored approximations.
    - K-FAC (empirical Fisher approximation).
    - Shampoo (AdaGrad-style approximation, often with “grafting” for step size).
      - Used to win AlgoPerf 2025 (hyper-parameters tuned to datasets).
    - Soap (runs Adam in Shampoo eigenspaces).
    - KL-Shampoo (KL divergence regularization as in classic quasi-Newton updates)
      - Natural distance on positive-definite matrices (no grafting and outperforming Soap).

- I received various opinions on Muon vs. Shampoo/Soap debate.
  - In any case, both seem more useful in gradient-dominated phase.

[Martens & Grosse, 2015, Gupta et al., 2018, Vyas et al., 2024, Lin et al., 2026]

Question: Are there faster algorithms for finite datasets?

(Asked by James Spall during NeurIPS 2012 tutorial)

Map of Speeding up SGD.

- Gradient-dominated:
  - ~~Large step sizes.~~
  - ~~Momentum.~~
  - ~~Adam and friends.~~
  - ~~Preconditioning.~~
- Noise-dominated:
  - ~~Decreasing step sizes.~~
  - ~~Interpolation.~~
  - ~~Averaging.~~
  - ~~Mini-batching.~~

Finite datasets tricks:

- Importance sampling.
- Random reshuffling.
- Variance reduction.

Infinite vs. Finite Data Sets

- Our SGD discussion so far allows infinite data sets.
  - We have some way to generate a new random example.
  - Compute budget only allows us to go through the dataset once (or less).

- In practice, we can often do multiple passes over a finite dataset.

- Several things change in the finite-data setting:
  - Stronger effect of mini-batches.
  - Speed convergence with biased sampling.
  - Eliminate noise-dominated phase with variance reduction.

“With Replacement” vs. “Without Replacement”

- For mini-batch SGD, we assumed “with replacement” sampling:
  - Leads to an unbiased gradient approximation with updated variance of:

$$\sigma^2_{|B_k|} = \frac{\sigma(w_k)^2}{|B_k|}$$

  - But for a finite dataset, wasteful due to repeated examples in mini-batch.
- With finite data, we can use sampling “without replacement”:
  - Leads to an unbiased gradient approximation with updated variance of

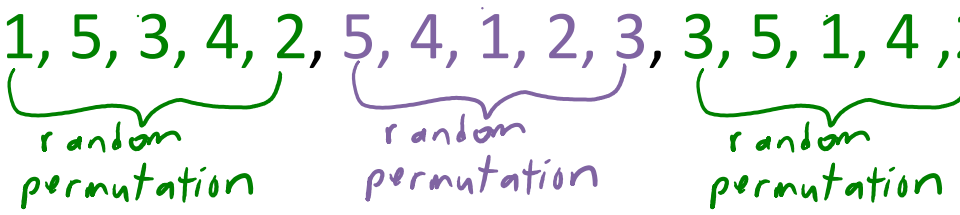
$$\sigma^2_{|B_k|} = \frac{\sigma(w_k)^2}{|B_k|}\frac{n - |B_k|}{n}$$

  - Finite sample correction reflects that we eventually sample every example.
    - So mini-batch SGD allows even larger step sizes, particularly for larger $|B_k|$.
    - Can sometimes re-use computation if examples appear in $|B_k|$ and $|B_{k-1}|$ (clever step sizes).

[Cochran, 1953]

Random Reshuffling (RR)

- Classic SGD theory: must sample i_k uniformly from {1,2,…,n}.
  - i_k sequence: 3, 1, 3, 3, 3, 3, 2, 2, 4, 1, 4, 4, 1, 2, 5.
    - First sequence I got with "rand", debated removing it due to "not looking random".
- For finite datasets, people often use random reshuffling (RR):
  - i_k sequence: 1, 5, 3, 4, 2, 5, 4, 1, 2, 3, 3, 5, 1, 4 ,2.



  - RR goes through all n examples every n steps.
    - Compute new order after every 'n' steps.
      - "Without replacement sampling across time".

[Bottou, 2009]

Random Reshuffling (RR)

- Practice: years of evidence that random shuffling converges faster.

- Theory: challenging because it gives biased gradient estimate.
  - First works showing improvement had unrealistic assumptions.
  - Showing that RR helps in various settings.
    - But not clean: depend on $L_{\max}$ ("worst sharpness") rather than smaller L.

- On the other hand, theory might just eventually verify the obvious.

[Haochen & Sra, 2019, Ahn et al., 2020, Mischenko et al., 2020, Gürbüzbalaban et al. 2021, Nguyen et al., 2021, Fan et al., 2023, Cai et al., 2024]

Variance-Reduced SGD

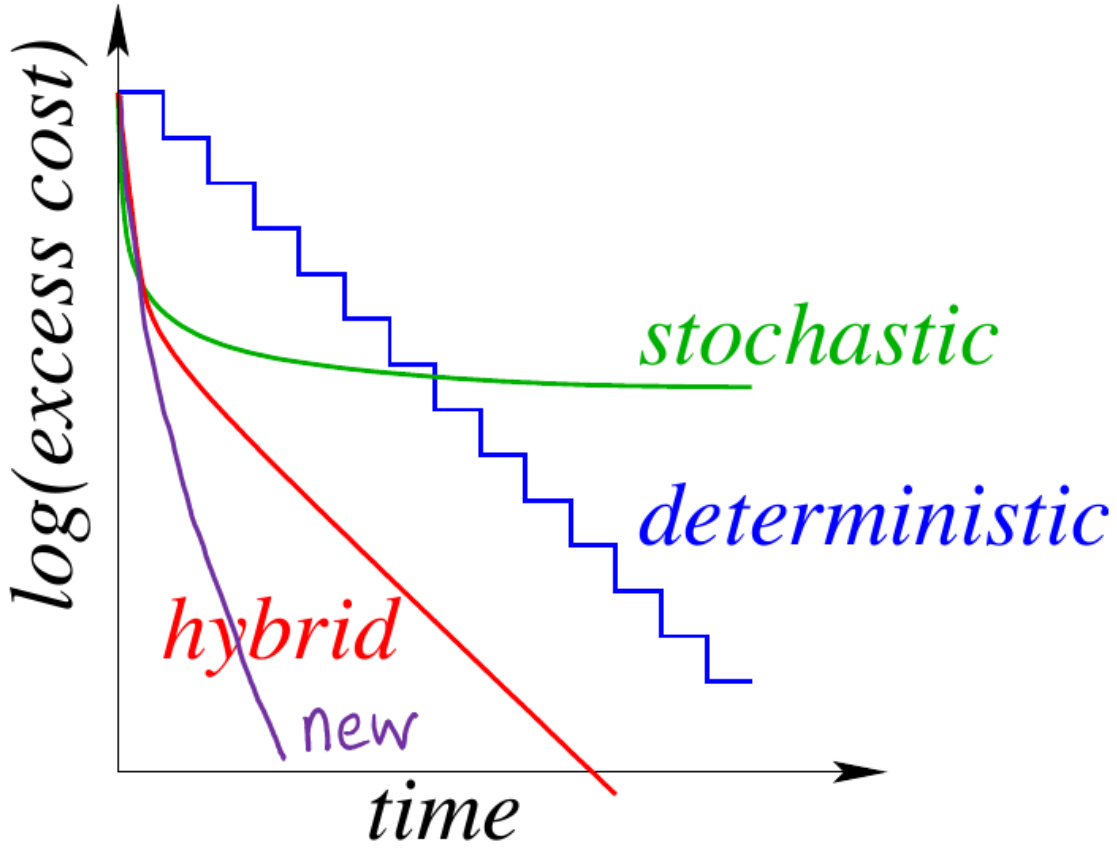
- Standard approaches to avoid noise-dominated phase::
  - Decaying step sizes slows down progress.
  - Interpolation does not make sense for LLMs.
  - Large batches can increase the iteration cost.

- Variance-reduced stochastic gradient methods:
  - Allow large step sizes with fixed batch sizes and non-vanishing noise.
    - "Convergence of GD with cost of SGD".
  - Key idea: design an estimate of $\nabla f(w_k)$ whose variance converges to 0.
    - Many methods including SAG, SVRG, SAGA, SARAH, SPIDER, STORM.
      - Accelerated versions: Catalyst, Point-SAGA, Katyusha.

[Le Roux et al., 2012, Johnson & Zhang, 2013, Defazio et al., 2014, Lin et al., 2015, Defazio, 2016,Allen-Zhu, 2017, Nguyen et al., 2017, Fang et al., 2018, Cutkosky & Orabona, 2019]

Variance-Reduced SGD

- Variance-reduced SGD keep low cost of SGD but speed of GD:



Variance-Reduced SGD

- Variance-reduction helps for many classic ML problems.
  - Theoretically and empirically faster methods for many problems.

- Empirically, variance-reduction does not help for deep learning.
  - It can slow you down in gradient-dominated phase.

- Recent work: Make vAriance Reduction Shine (MARS) method.
  - For large batches performed among best in recent comparison.

- Why would variance reduction be helping for larger batch sizes?
  - Variance reduction results rely on $L_{\max}$ ("sharpest batch").
    - As you increase the batch size, $L_{\max} \to L$ (larger step sizes and more progress).

[Schmidt et al., 2015, Defazio & Bottou, 2019, Yuan et al., 2025, Semenov et al., 2025]

Importance Sampling

- Classic SGD theory: must sample i_k uniformly from {1,2,…,n}.
  - i_k sequence: 3, 1, 3, 3, 3, 3, 2, 2, 4, 1, 4, 4, 1, 2, 5.
- What if we have "easy" examples and "hard" examples?
  - May want to sample i_k from a non-uniform distribution?
  - i_k sequence: 3, 1, 1, 3, 3, 2, 2, 2, 1, 1, 2, 1, 1, 2, 5.
- Suppose you know sharpness constant L_i of each training example.
  - Consider biasing random sampling by the L_i values.
    - "Sample the sharp examples more often."
  - With local $L_i(w_k)$ values, starts ignoring well-classified.
    - "Automatic curriculum learning".
- Can improve rates in gradient-dominated phase.
  - Depends on L_{mean} instead of $L_{\max}$ (both are still less than L).
  - Does not help SGD in noise-dominated phase.
    - But does help if we have interpolation or use variance-reduced SGD.
- Practice: does not seem to help much for training neural networks.

[Needell et al., 2014, Xiao & Tong, 2014, Schmidt et al., 2015]

Question: why is optimizing deep networks hard?

What is different about deep networks?

- Example 1:
  - Classic: initialization does not really matter.
  - Deep: careful initialization to avoid exploding/vanishing gradients.
- Example 2:
  - Classic: would like to adapt to $L(w_k)$, to move faster in flatter regions.
  - Deep: optimizer influences $L(w_k)$, and could move us to sharp regions.
    - But fixes that decrease sharpness can cause representation collapse.
      - And fixes for representation collapse can prevent us from learning features.
- Why are we "fixing one thing and breaking another"?
  - It might be that deep learning is hard because of trade-offs.
- Classic optimization has α_k vs. $2/L$, what does deep learning need?

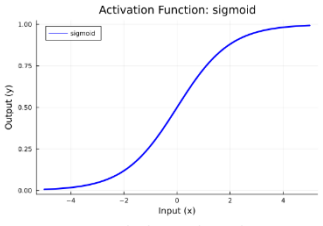
Trade-Offs in Training Deep Networks?

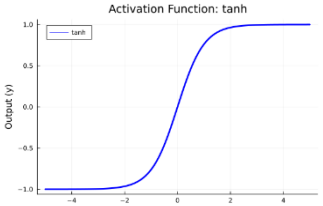
- An attempt at a short list of trade-offs in training deep networks:
  - Signal flow. $\prod|\lambda(J_m)| = O(1)$
    - Too high (exploding gradient) vs. too low (vanishing gradient).
  - Input sensitivity. $\|g(x_i, w) - g(x_j, w)\| = O(1)$
    - Too high (overfitting) vs. too low (representation collapse).
  - Parameter sensitivity. $\lambda(\nabla^2 f(w)) = O(1)$
    - Too high (need vanishing step sizes) vs. too low (need exploding step sizes).
  - Landscape Topology. $\int_w \delta(\nabla f(w)) \cdot |\det(\nabla^2 f(w))| dw = O(1)$
    - Too complicated (too spiky) vs. too simple (no flexibility).
  - Feature Plasticity. $\|w - w_*\| = O(1)$
    - Too high (chaotic dynamics) vs. too low (lazy learning).
- It seems like most practical tricks address one of these trade-offs.
  - But need to consider if tricks are too extreme and how they affect the others!
    - The "pipeline" is so rigid because we have an unstable equilibrium among trade-offs.

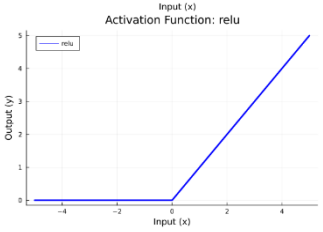
[Glorot & Bengio, 2010, Choromanksa et al., 2015, Goodfellow et al., 2016, Tishby & Schwartz-Ziv, 2017, Chizat et al., 2019]

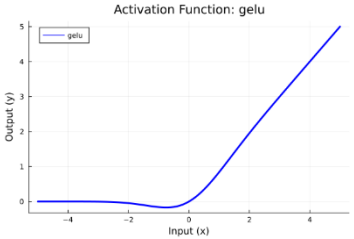
Trade-Offs in Activation Function History

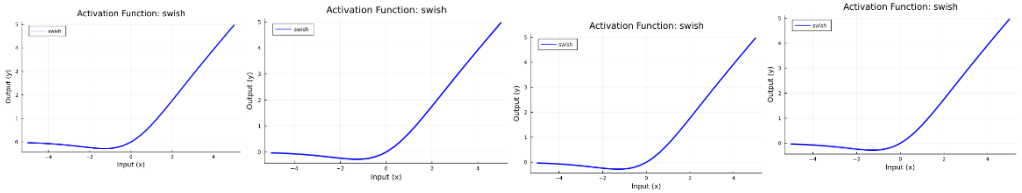
- Sigmoid – bounds input sensitivity.
  - But worsens signal flow.
- Tanh – improved signal flow over sigmoid.
- ReLU – improved signal flow over tanh.
  - But worsens parameter sensitivity.
- GELU – improved parameter sensitivity over ReLU.
- SwiGLU – improved input sensitivity over GELU.
  - But worsens feature plasticity (more chaotic).
    - So need tricks to control that.











[Nair & Hinton, 2010, Hendrycks & Gimpel, 2016, Shazeer, 2020]

Trade-Offs in Normalization and Skip Connections

- Operations like LayerNorm (and RMSNorm) affect many trade-offs:
  - Prevent signal flow from being too large.
  - Reduces input sensitivity (Lipschitz constant with respect to input).
  - Reduces parameter sensitivity (reduces $L(w_k)$).
- This is great, but often too aggressive so we add skip connections:
  - Prevent signal flow from being too small.
  - Increases input sensitivity (co-Lipschitz constant with respect to input).
  - Increases parameter sensitivity (increases $\mu(w_k)$).
- But this is too aggressive so need ReZero/FixUp.
- These are architecture changes, but exist algorithm changes too:
  - Decoupled weight decay, sharpness-aware minimization (SAM), muP step sizes.

[Srivastava et al, 2015, Ba et al., 2016, He et al., 2016, Loschilov & Hutter, 2019, Zhang & Senrich, 2019, Zhang et al., 2019, Bachlechner et al., 2021, Foret et al., 2021, Yang et al., 2022]

Weird Neural Network Dynamics

- General deep networks training dynamics are not well understood.
  - Example: "edge of stability".
    - Classic setting: we adapt α_k based on the local value $L(w_k)$.
      - For GD, we decrease α_k until $\alpha_k \leq 2/L(w_k)$.
    - Neural networks: local $L(w_k)$ can sometimes adapt to the value of α_k.
      - For GD, $L(w_k)$ increases until we have $\alpha_k \approx 2/L$.
  - Example: "catapults".
    - More progress overall with occasional large increases in objective function.
      - Possible on quadratics with variable step sizes.
  - Example: "grokking".
    - Huge progress after period of slow progress.
      - Believed to be related to feature alignment.
  - Big example: "feature learning" vs. "lazy learning".
    - How do we more precisely characterize the dynamics here?

[Jacot et al., 2018, Chizat et al., 2019, Lewkowycz et al., 2020, Cohen et al, 2021, Power et al., 2022]

Discussion

Summary: Theoretical vs. Practical Research

- Two main styles of optimization algorithms research:
  - Theoretical: use math to prove algorithm properties.
  - Empirical: use experiments to support algorithm properties.

- Theoretical:
  - Example: prove an algorithm converges.
  - Advantage: guaranteed worst-case performance of algorithms in new situations.
  - Disadvantage: unrealistic assumptions or too pessimistic about performance in practice.
- Empirical:
  - Example: evaluate algorithm performance.
  - Advantage: reflects real-world performance (sometimes hard to prove).
  - Disadvantage: mislead ourselves as experiments may not reflect new situations.

- I think it can be useful to know a bit of both styles.

Summary: How do I set the GD Step Size?

- Gradient descent (GD) is basis of modern methods like SGD and Adam.
- Level 1 (most theory is here):
  - If you are doing 1 GD step, and only know that curvature is at most L.
    - Any step size less than $2/L$ decreases functions.
    - A constant step size of $1/L$ is optimal.
    - But this is a terrible model of realistic functions.
- Level 2:
  - Can go faster using a local "L" or optimizing multi-step progress.
    - Suggests various forms of variable step sizes, or constant with gradient clipping if $L = \infty$.
- Level 3:
  - Can go faster by exploiting direction and size of each gradient.
- Level 4 (most practice is here):
  - Can go faster by planning for future progress (might want to increase function).

Summary: How do I set the SGD Step Size?

- Stochastic gradient (SGD): GD using gradient of random example.
- SGD has lower cost than GD but introduces noise.
  - Noise means we may need step sizes smaller than $2/L$.
  - But size of gradient reduces effect of noise.
- SGD often has two phases:
  - Gradient-dominated: where gradient signal outweighs noise.
  - Noise-dominated: where noise signal outweighs gradient.
- Different tricks needed in different phases.
  - But tricks that help in one phase may hurt the other.
- We do not know when we switch phases.
  - This is where hyper-parameters help.

Summary: faster algorithms than GD?

- Ways to speed up GD (or SGD in gradient-dominated phase):
  - Momentum and Nesterov:
    - Theory: provably faster than GD in various settings, but need to know setting.
    - Practice: fixed momentum often helps in DL, non-linear CG adapts momentum.
  - Adam:
    - Approximated by momentum combined with SignGD (GD in ∞-norm).
    - SignGD can help for quadratics but depends on rotation.
      - Heavy-tailed features/labels can lead to favourable structures in practice.
    - Muon can viewed as SignGD in a matrix norm.
  - Preconditioning:
    - Allows arbitrary speedups, but hard to get these speedups practically.
    - Newton's method only helps near solution, but that is often noise-dominated.
    - Per-variable step sizes is diagonal preconditioning: long history but recent theory/practice.
    - Natural gradient is Newton for linear models and Gauss-Newton under squared error.
      - Often use approximation (related to AdaGrad) that breaks connection (Shampoo).
  - Non-uniform sampling of random examples (might only SGD help in theory).

Summary: faster algorithms than SGD?

- Ways to speed up SGD in noise dominated phase:
  - Random reshuffling:
    - Trick with decades of empirical evidence, theory is catching up.
  - Variable step sizes:
    - Decreasing step size schedules give convergence based on global L and σ^2 value.
      - In practice need to use hyper-parameter tuning to adapt to problem.
    - Exist algorithms that adapt to $L(w_k)$ like coin betting and Prodigy.
      - Good "single run" performance but worse than tuning.
    - Exist algorithms that adapt to extreme values of $\sigma(w_k)^2$:
      - Gradient clipping for $\sigma^2 = \infty$, deterministic GD tricks for $\sigma(w_k)^2 \to 0$ (but weird on neural nets).
  - Averaging allows bigger step sizes:
    - Iterate averaging is asymptotically optimal way to handle noise.
    - Mini-batches directly reduce noise (optimal batch size is noise/error).
    - Variance reduction helps in theory, but has not had success for neural nets (changing?).

No Convergence Rates?

- I purposefully avoided convergence rates in this talk.
  - This was hard for me!

- But we get a lot of insights from some simple ideas:
  - "What happens if I plug this into some version of the descent lemma?"
    - Reveals under what conditions you expect progress, and how much progress.
      - And holds across many settings like PL, convex, and non-convex.
    - Gave insights into step sizes, SGD dynamics, batch sizes, first- vs. second-order.
  - "What happens if this quantity (like L or σ) changes with w_t?"
    - Reveals why some methods are bad in practice, and suggests ways to fix them.
  - "What does this look like on simple cases where I can visualize things?"
    - Reveals how algorithms are sometimes doing things that we do not understand.
      - Large step sizes move down ravines, momentum allows a big range of step sizes.

What do we need?

"There should be no YOLO training runs. Machine Learning should be stable, smooth and predictable" [Defazio, 2026]

- Years ago, I thought theory and practice were coming together.
  - But this was mis-leading due to theory focusing on fixed features.
- Need better models of how features evolve as we train.
  - Should have predicted that normalization would be needed before it was!
    - And MuP insights came surprisingly recently.
  - Until we understand neural network trade-offs, theory will be far from practice.
- Ultimate test of theory is whether it affects practice.