Deep Learning + Reinforcement Learning
SUMMER SCHOOL

ÉCOLE D'ÉTÉ SUR
l'apprentissage profond + l'apprentissage par renforcement

Introduction to Optimization - Theory & Practice

Mark Schmidt

Canada CIFAR AI Chair
Fellow | Amii
Associate Professor | University of British Columbia

CIFAR | amii | Mila | VECTOR INSTITUTE | INSTITUT VECTEUR

Introduction

- Optimization is a big field.
  - Impossible to cover in detail in 90 minutes.

- I am assuming you have already trained a deep learning model.
  - Probably using stochastic gradient descent (SGD) or a variant like Adam.
- And it might useful if I could answer questions like these:
  1. How do I set the step size?
  2. How I pick the batch size?
  3. In what order should we process data?
  4. Are there faster algorithms than SGD?
  5. Can I just use Adam with default parameters?

Disclaimer

- These questions are complicated, depending on many factors.

- I will overview some important trade-offs to consider in this setting.
  - Which may help improve your understanding of these questions.
  - I will necessarily need to skip over a lot of precise details.

- I will just focus on differentiable and unconstrained functions.
  - And I will assume the basic setting of IID data (no reinforcement learning).
  - Basic ideas still apply if we relax these assumptions, but extra details needed.

- If you want to see the long/detailed version, some 2022 notes are here:
  - https://www.cs.ubc.ca/~schmidtm/Courses/5XX-S22
    - This lecture series got interrupted by baby, but I plan to finish them next summer.

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'.
      - Could be squared error or cross entropy of network's predictions.

- Most popular DL algorithm is stochastic gradient descent (SGD).

Deterministic and Stochastic Gradient Descent

- For minimizing functions of the form:

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

- Deterministic gradient descent (GD) repeats the iteration:

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

  - For a "step size" α_k.

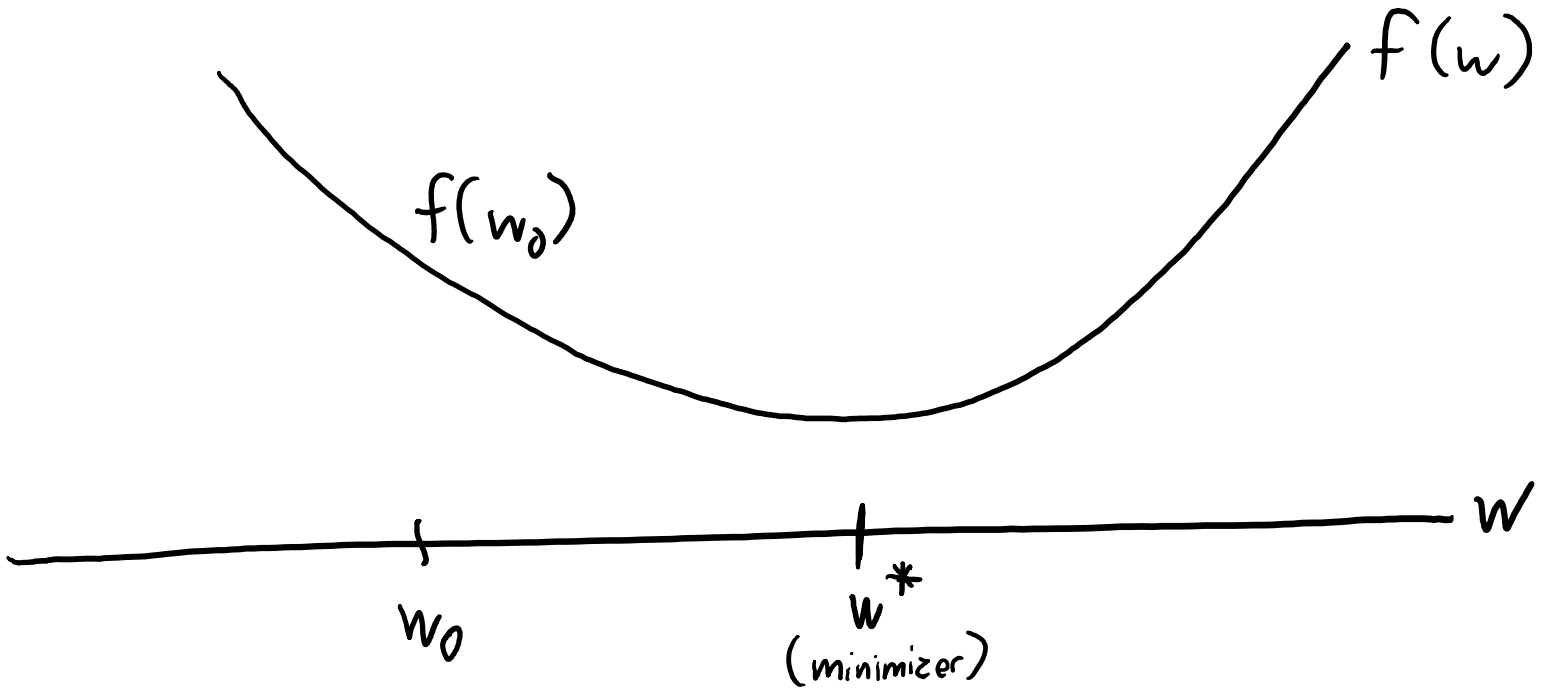
- Stochastic gradient descent (SGD) repeats the iteration:

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

  - 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.

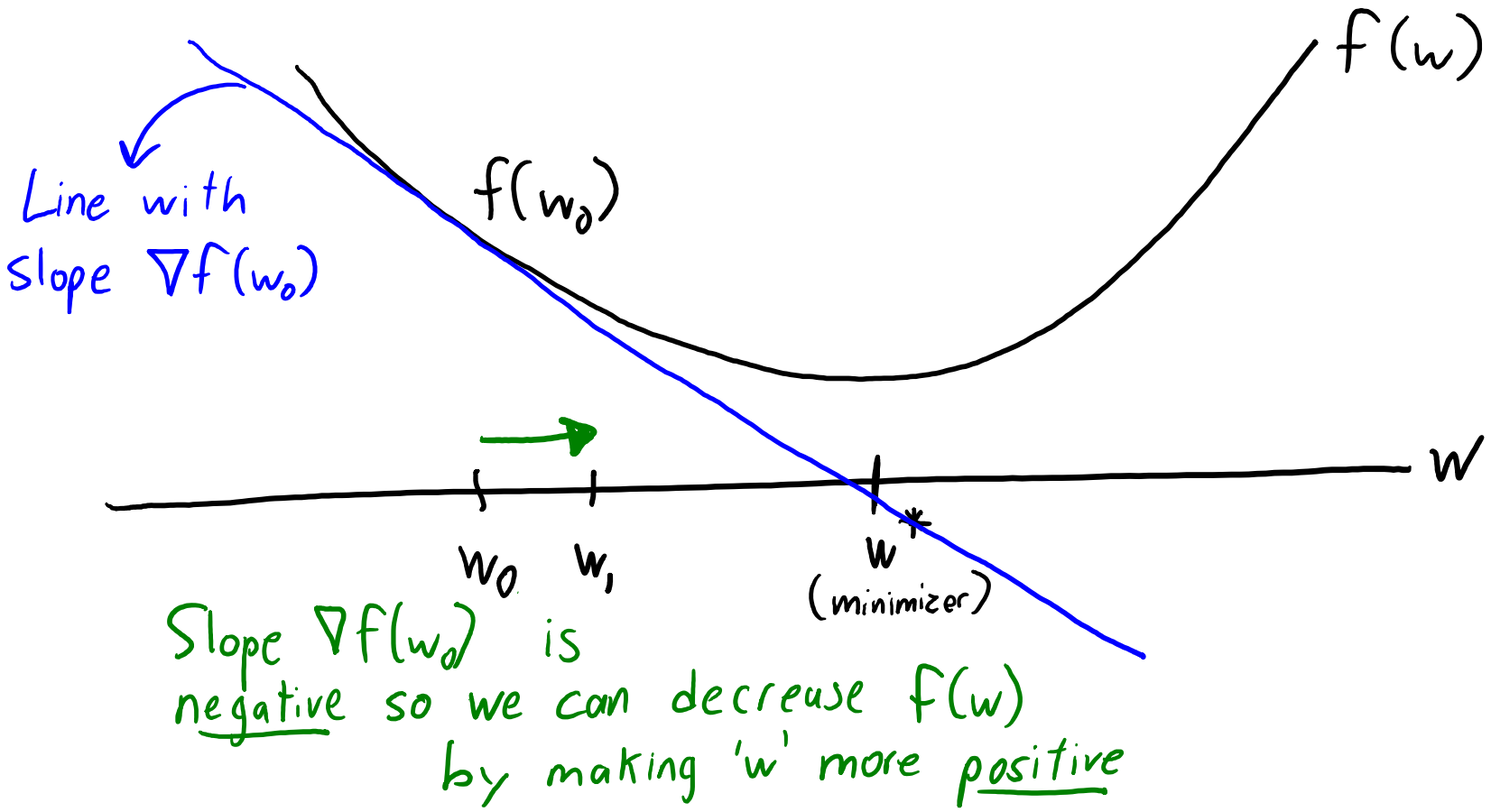
Deterministic Gradient Descent in 1 Dimension

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



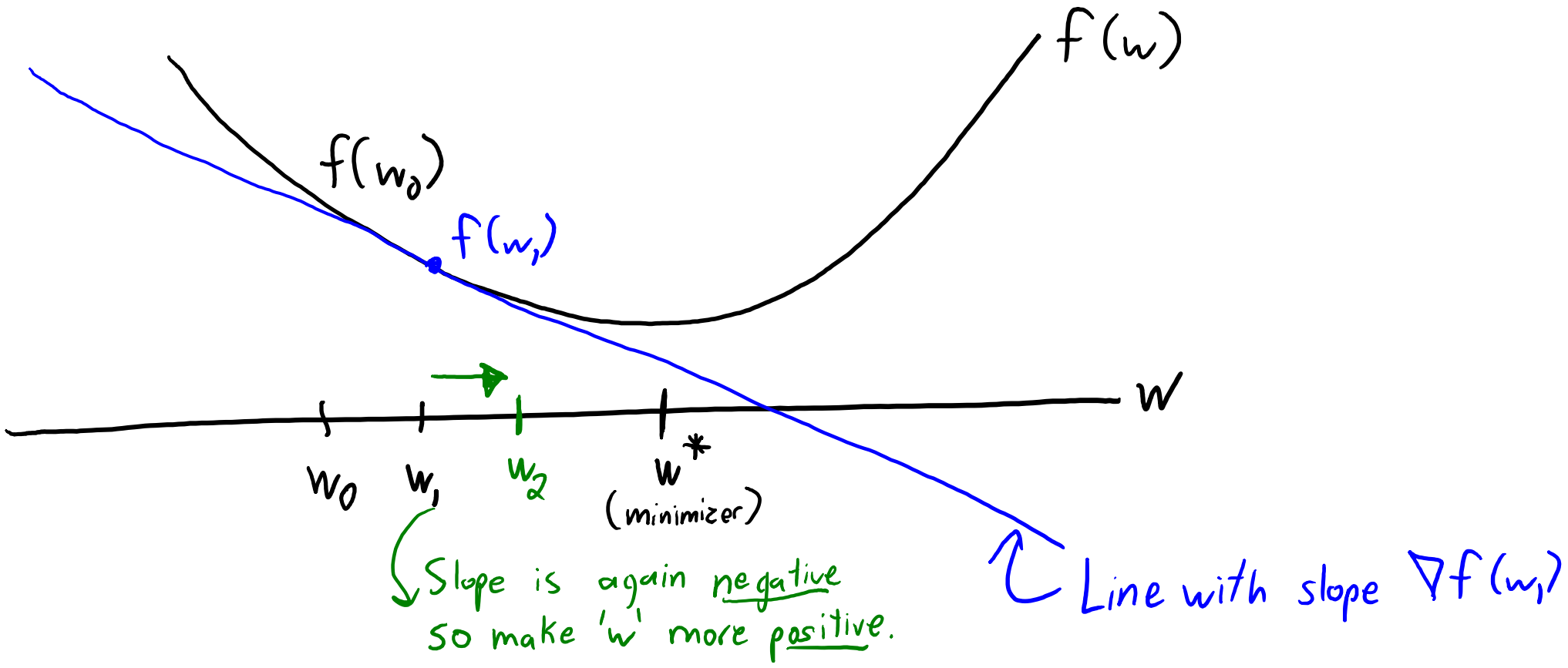
Deterministic Gradient Descent in 1 Dimension

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



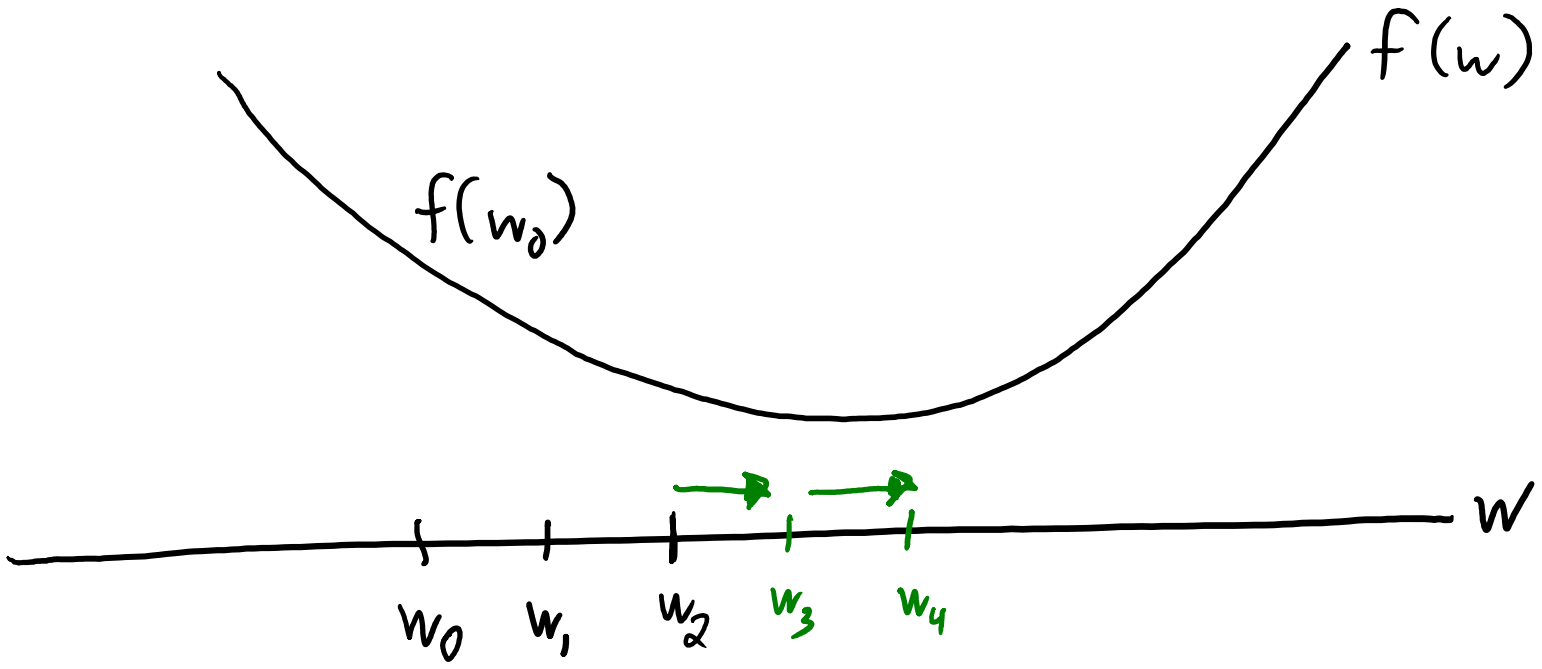
Deterministic Gradient Descent in 1 Dimension

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



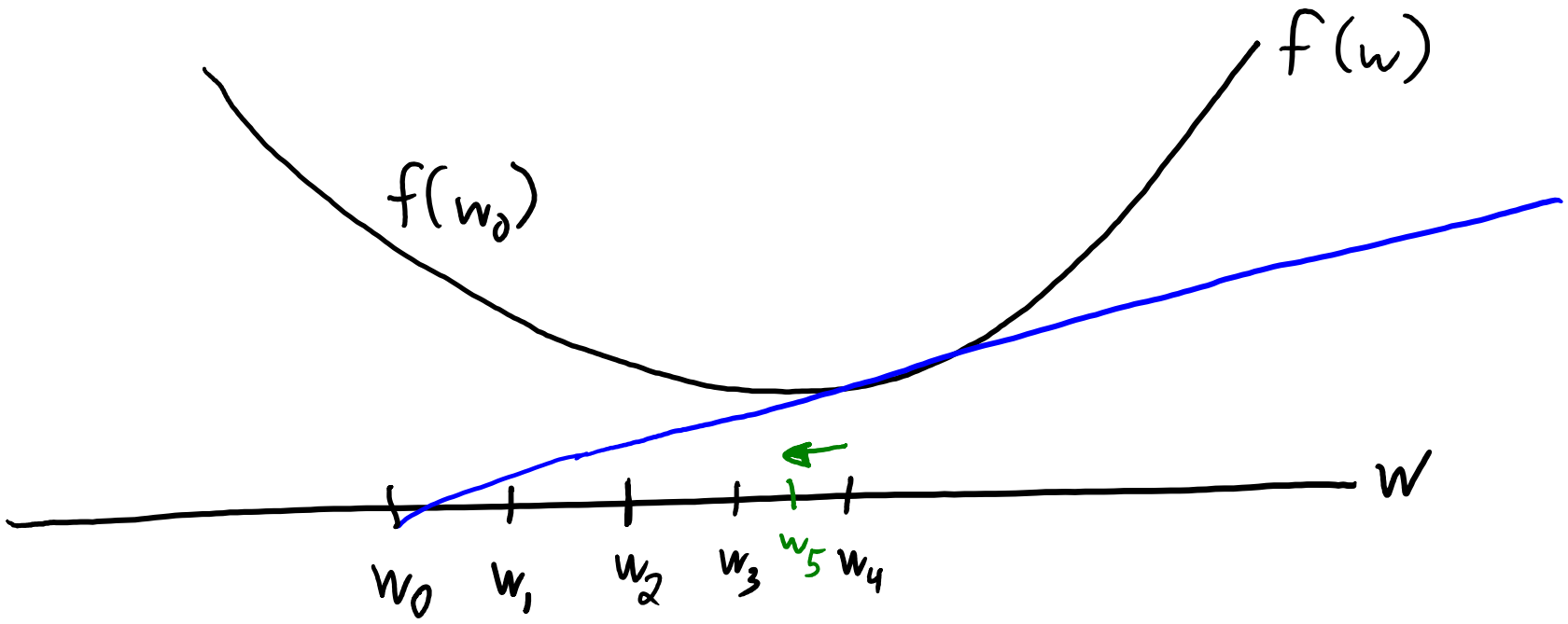
Deterministic Gradient Descent in 1 Dimension

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

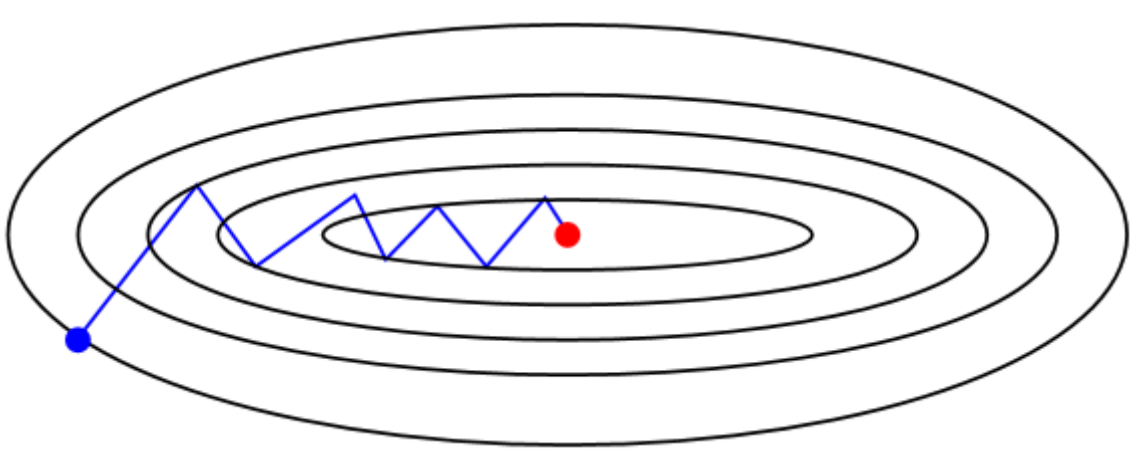


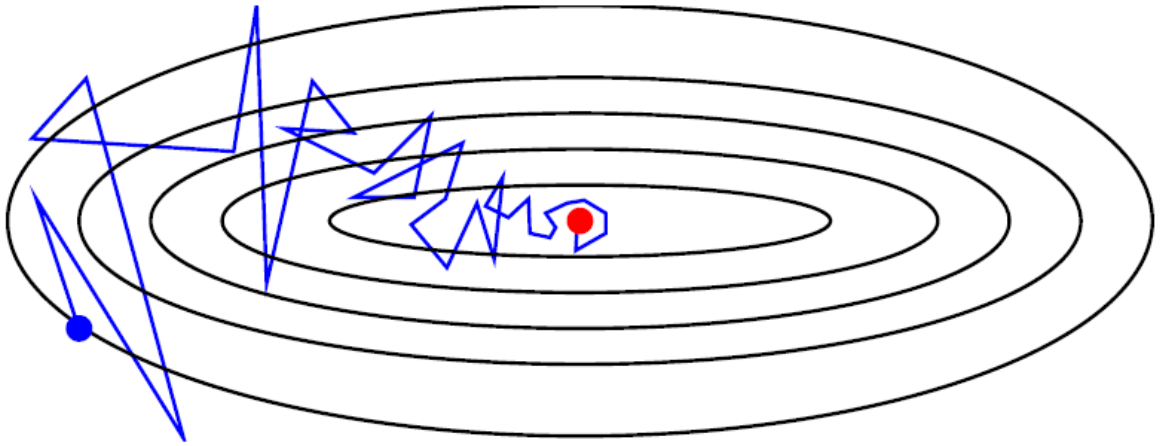
Deterministic Gradient Descent in 1 Dimension

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



GD and SGD in 2 Dimensions

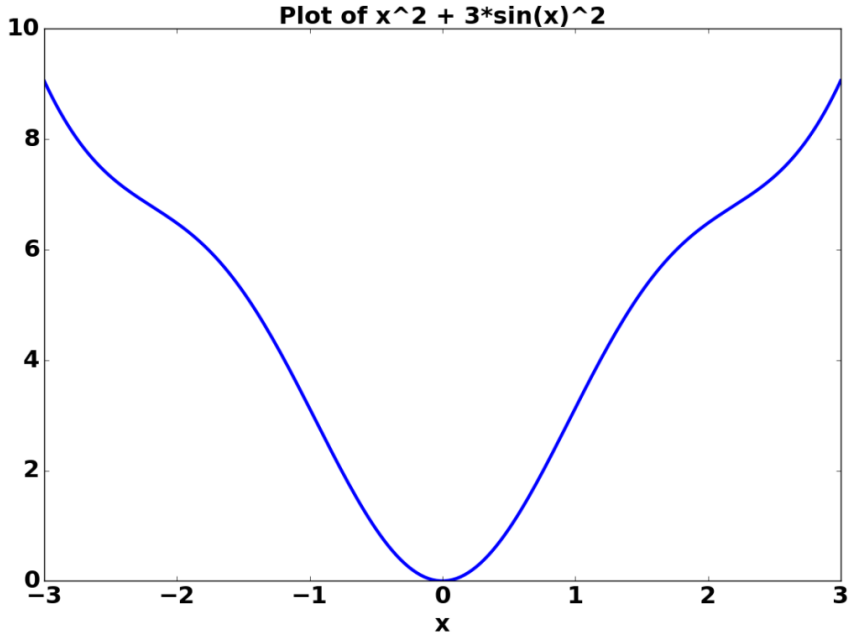




- Deterministic gradient descent (GD) converges with constant α_k.
- Stochastic gradient descent (SGD) converges with decreasing α_k.
- Both methods are local optimization methods:
  - May not find the global mimimum.
  - But with appropriate step sizes, we have that $\nabla f(w_k)$ converges to 0.

Polyak-Lojasiewicz Inequality and Invexity

- For some functions, we can show that GD/SGD find a global minimum.



- Recent works argue that some deep learning problems satisfy PL inequality.
  - Polyak-Lojasiewicz inequality.
    - "Gradient increases at least quadratically as we increase f(w) above global minima".
  - PL condition implies invexity.
    - Invexity is a generalization of convexity, where $\nabla f(w_k) = 0$ implies that w_k is a global min.

Theoretical vs. Empirical Optimization 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 quickly under PL inequality.
  - 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 compared to Adam across 9 datasets.
  - Advantage: reflects real-world performance (sometimes hard to prove).
  - Disadvantage: mislead ourselves as experiments may not reflect new situations.

- It can be useful to know a bit of both styles.

Question 1: How do I set the step size?

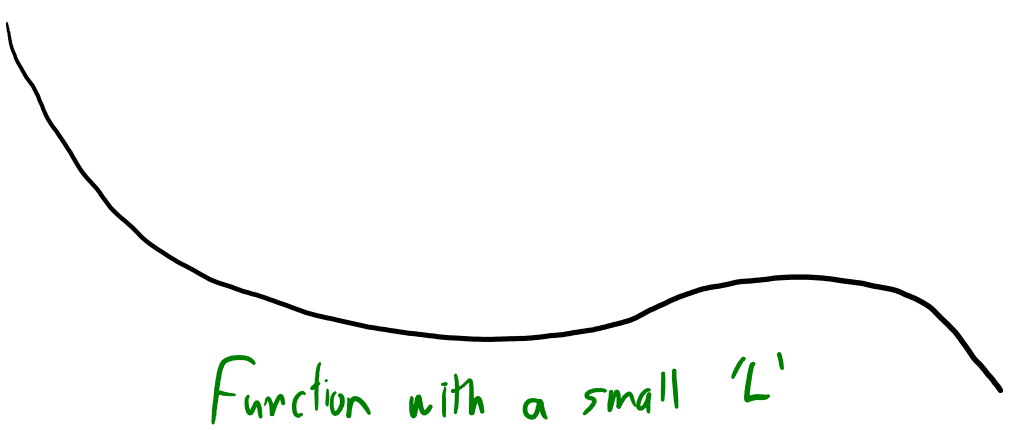
(we will spend the most time on this question)

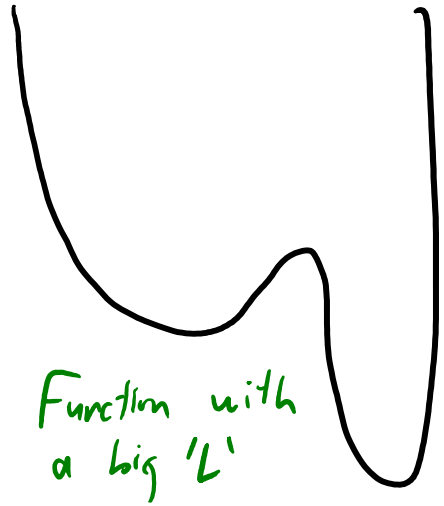
How do I set the GD step size?

- 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:

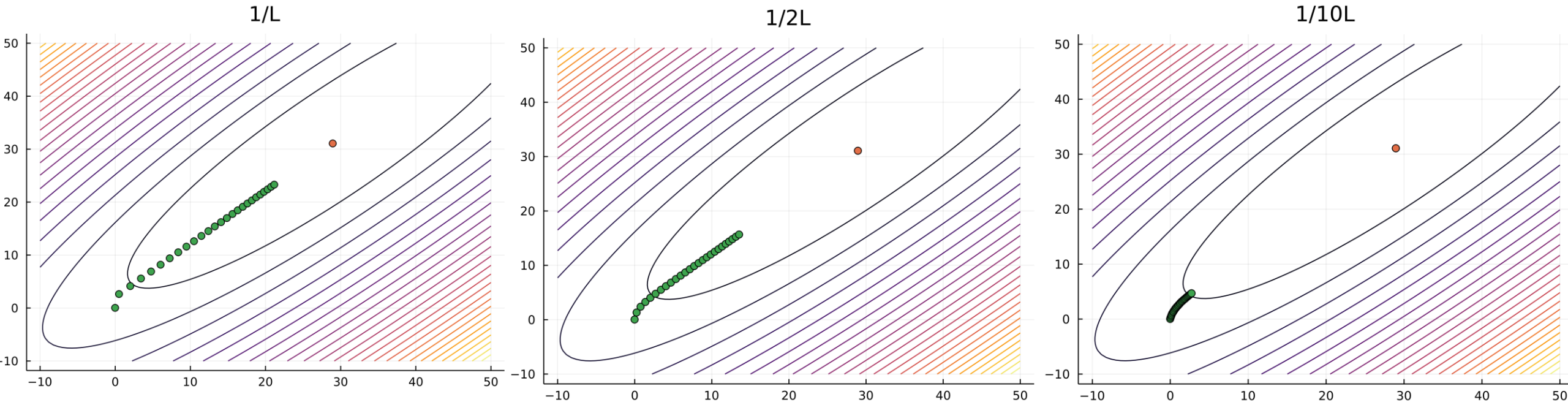
α_k vs. $\frac{2}{L}$

"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.
    - In practice, people use clever step sizes or line searches to make GD work well.

Comparison of Fixed Step Sizes for GD

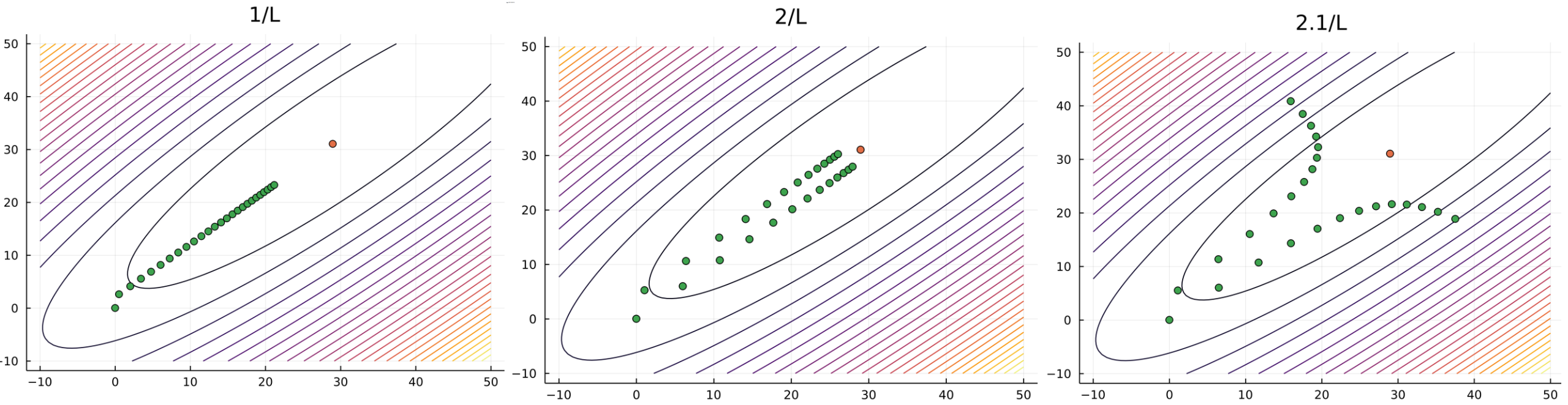
- 25 steps of deterministic gradient descent with different α_k.



- Progress is very slow if α_k is too small.

Comparison of Fixed Step Sizes for GD

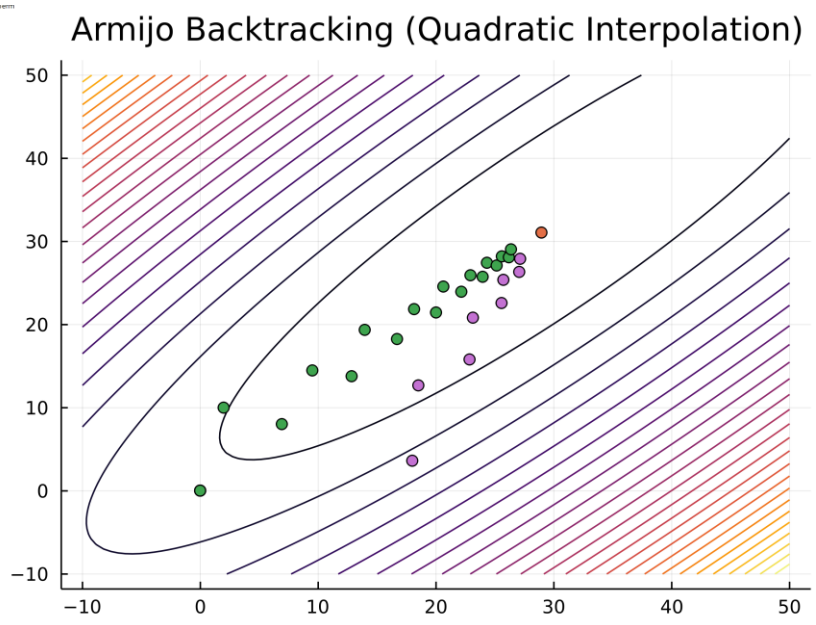
- Increasing the step size:

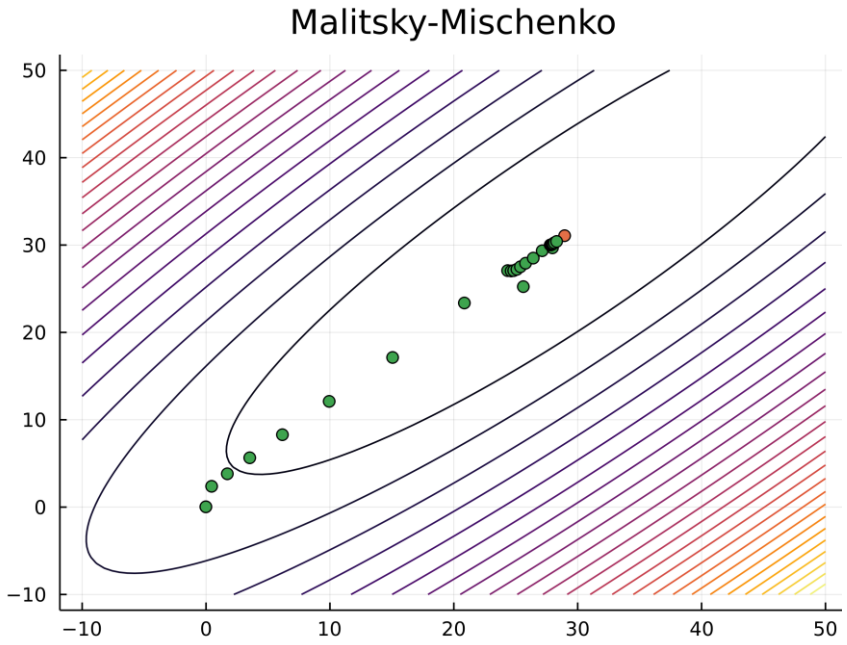


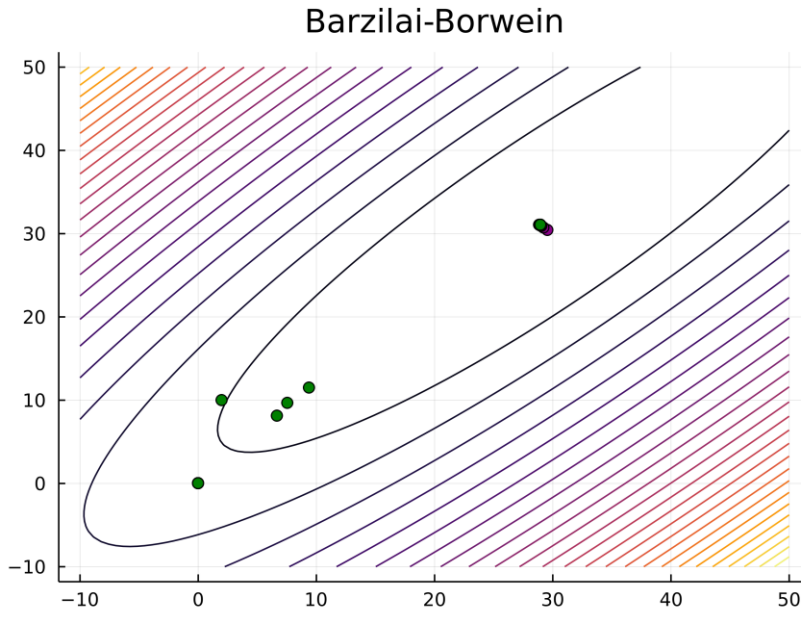
- Progress can be faster if $\alpha_k \geq 2/L$ or it may may increase function.

Comparison of Adaptive Step Sizes for GD

- For GD you can also use clever step sizes or line-searches.







- Better than fixed steps in practice, do not work for SGD in general.

How do I set the SGD step size?

- For stochastic gradient descent (SGD), there is a second trade-off:

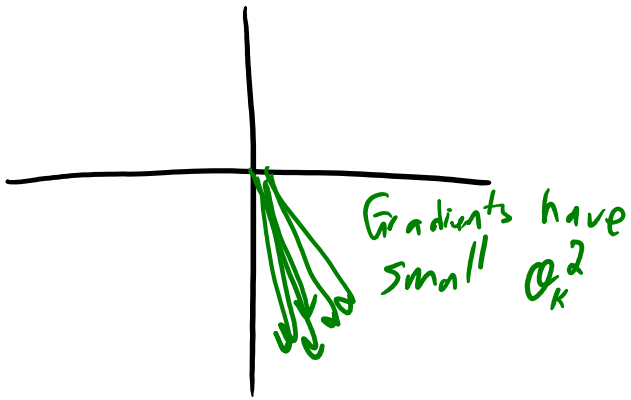
$||\nabla f(w_k)||^2$ vs. $\alpha_k \sigma_k^2$

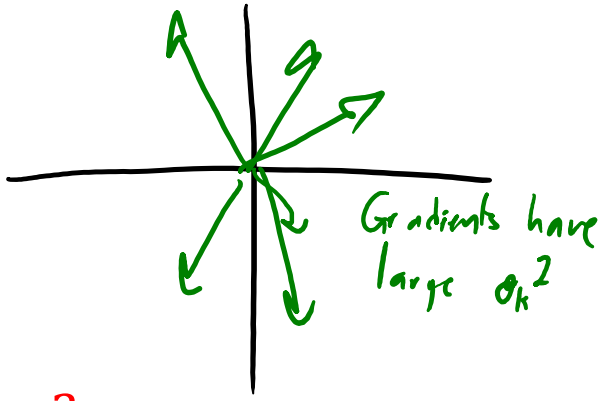
"size of gradient of overall function"

step size

"variation in the stochastic gradients"

- The number σ_k^2 measures the variation in the gradients.
  - We have $\sigma_k^2 = \frac{1}{n}\sum_{i=1}^{n} ||\nabla f_i(w^k) - \nabla f(w^k)||^2$.





  - Need smaller step size α_k for bigger σ_k^2 (but and smaller $||\nabla f(w_k)||$).

How do I set the SGD step size?

- For stochastic gradient descent (SGD), there is a second trade-off:

$||\nabla f(w_k)||^2$ vs. $\alpha_k \sigma_k^2$

"size of gradient of overall function"

step size

"variation in the stochastic gradients"

- If $||\nabla f(w_k)||^2 \gg \alpha_k \sigma_k^2$, SGD is guaranteed to decrease expected 'f'.
  - As long as we still have $\alpha_k < 2/L$.
  - For GD, we did not need to worry about this trade-off because $\sigma_k = 0$.

- If $\alpha_k \sigma_k^2 \gg \left||\nabla f(w_k)|\right|^2$, SGD is not guaranteed expected decrease.
  - Step size might be too big, causing the noise to interfere with progress.

How do I set the SGD step size?

- For stochastic gradient descent (SGD), there is a second trade-off:

$\|\nabla f(w_k)\|^2$ vs. $\alpha_k \sigma_k^2$

"size of gradient of overall function"

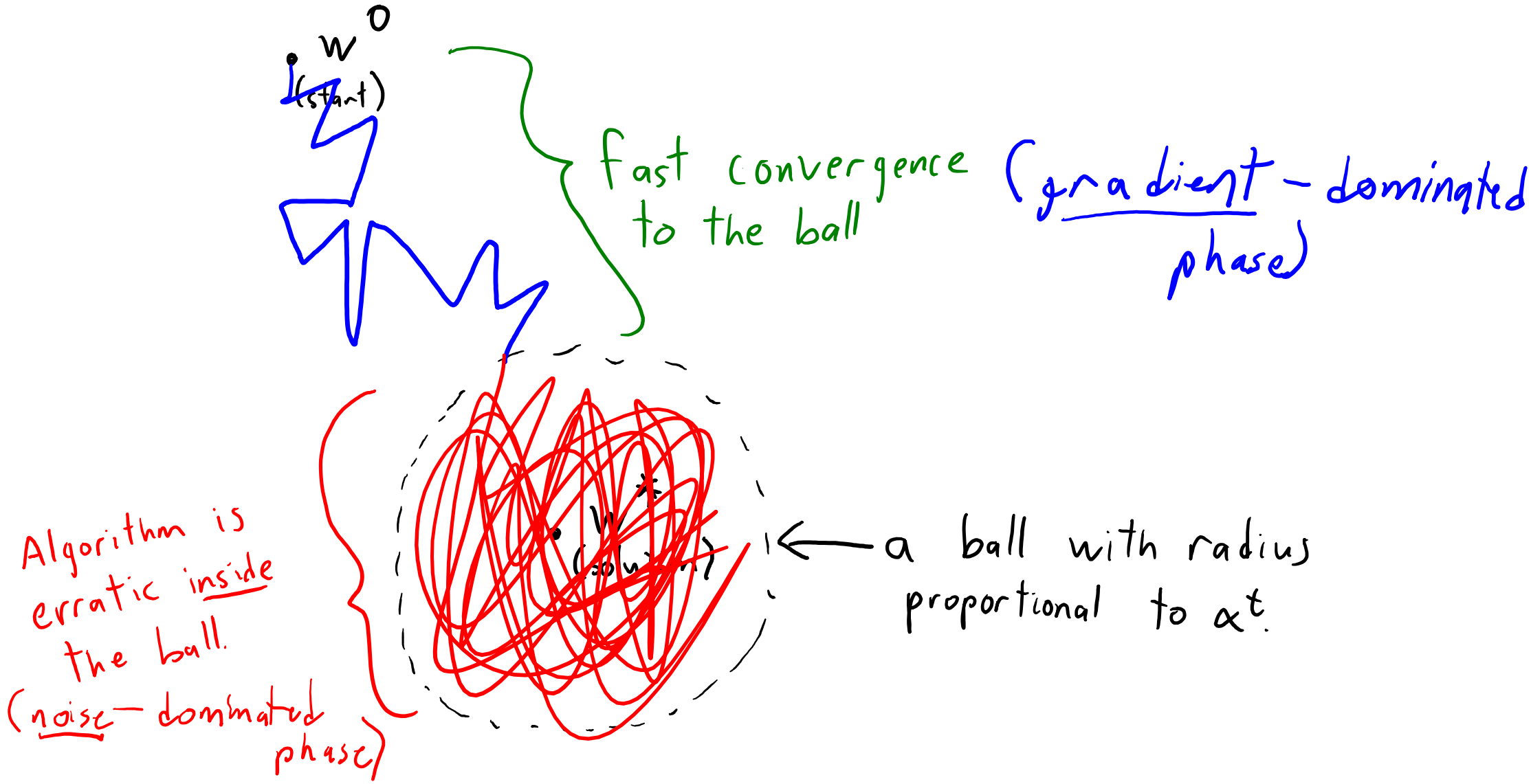
step size

"variation in the stochastic gradients"

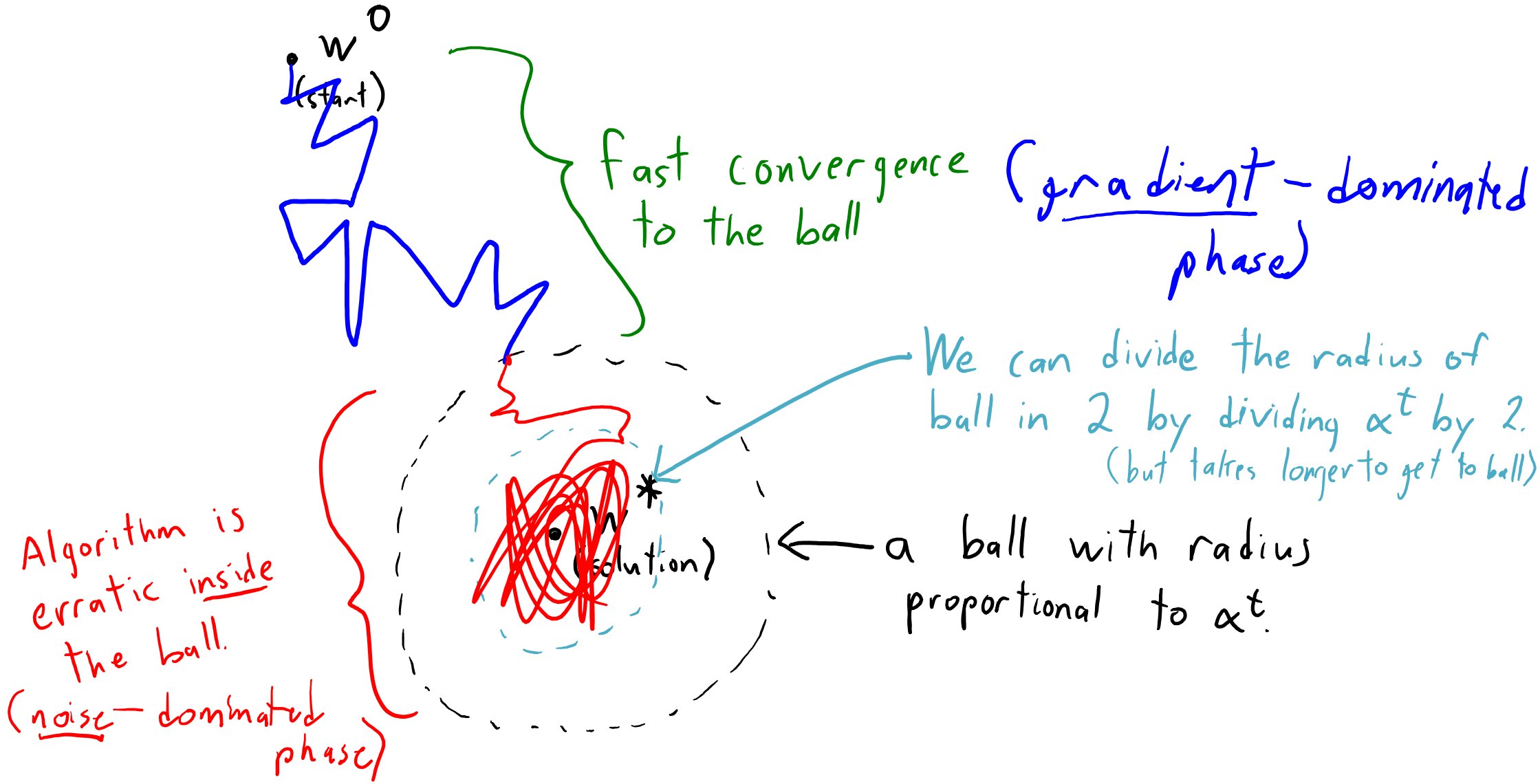
- As algorithm, $||\nabla f||^2$ goes to zero but σ^2 may not.
  - So for a fixed step-size α_k, SGD makes progress until σ^2 is too big.

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

Behaviour of SGD with Constant Step Size

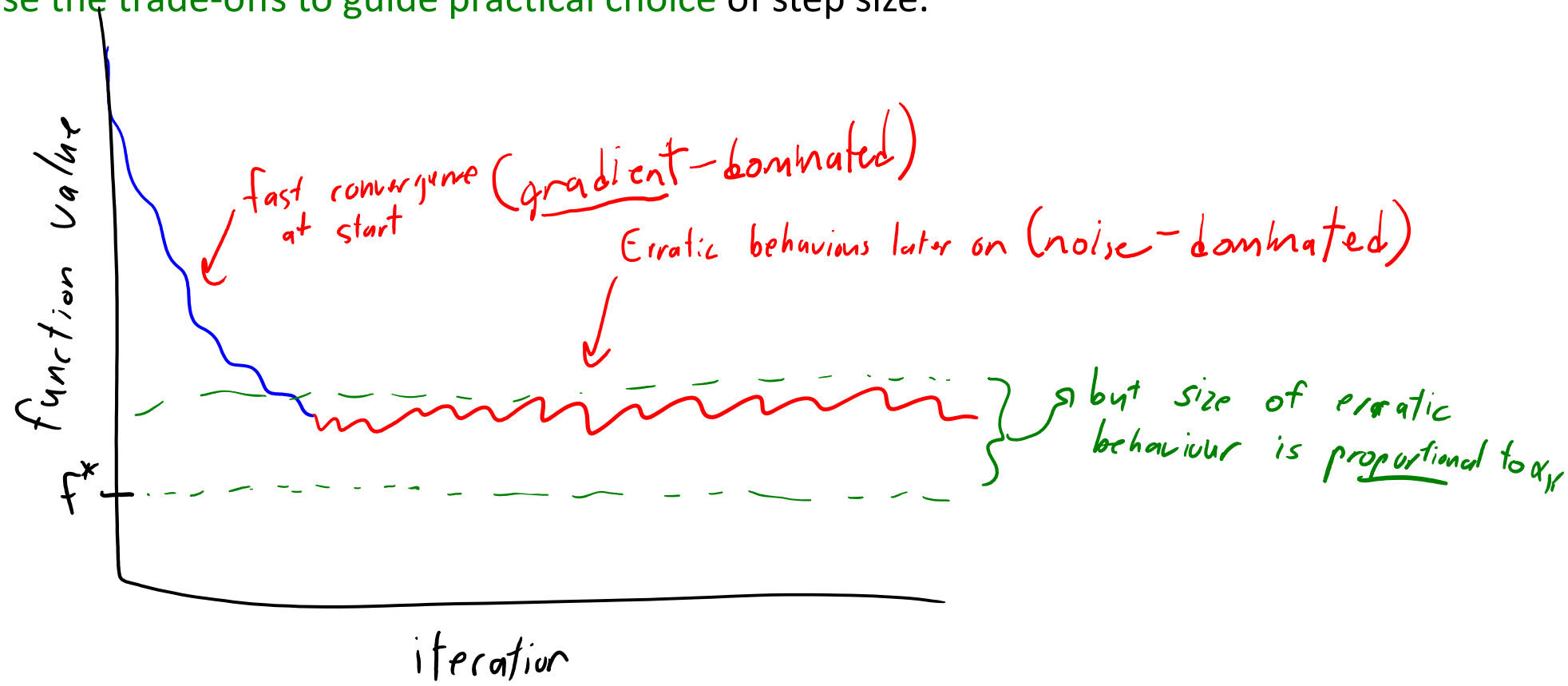


Behaviour of SGD with Constant Step Size



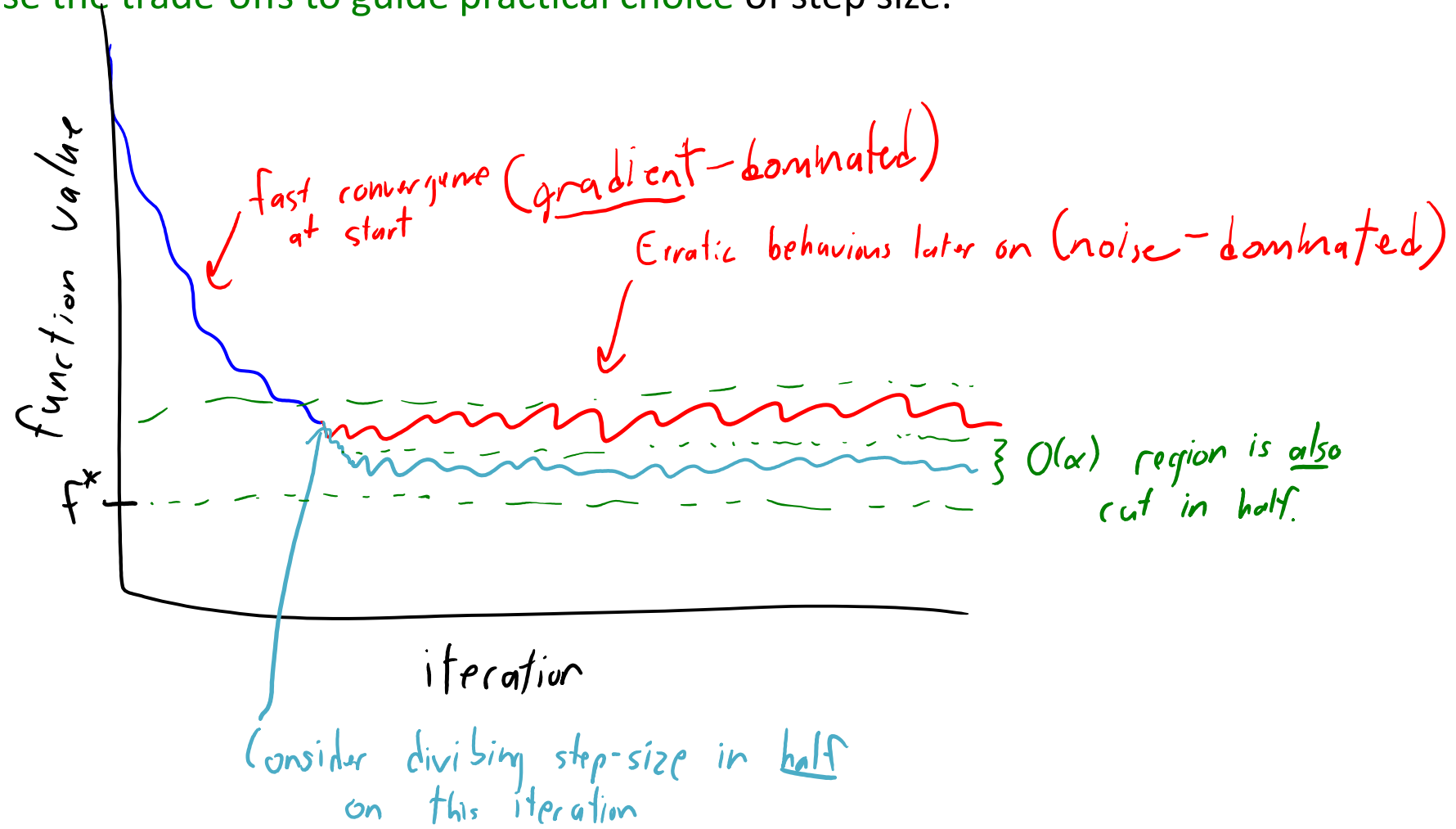
How do I set the SGD step size?

- In practice, SGD algorithm does not have access to $\nabla f(w_k)$ 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 $\nabla f(w_k)$ 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?

- 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 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.

- In some cases SGD will converge under a constant step size…

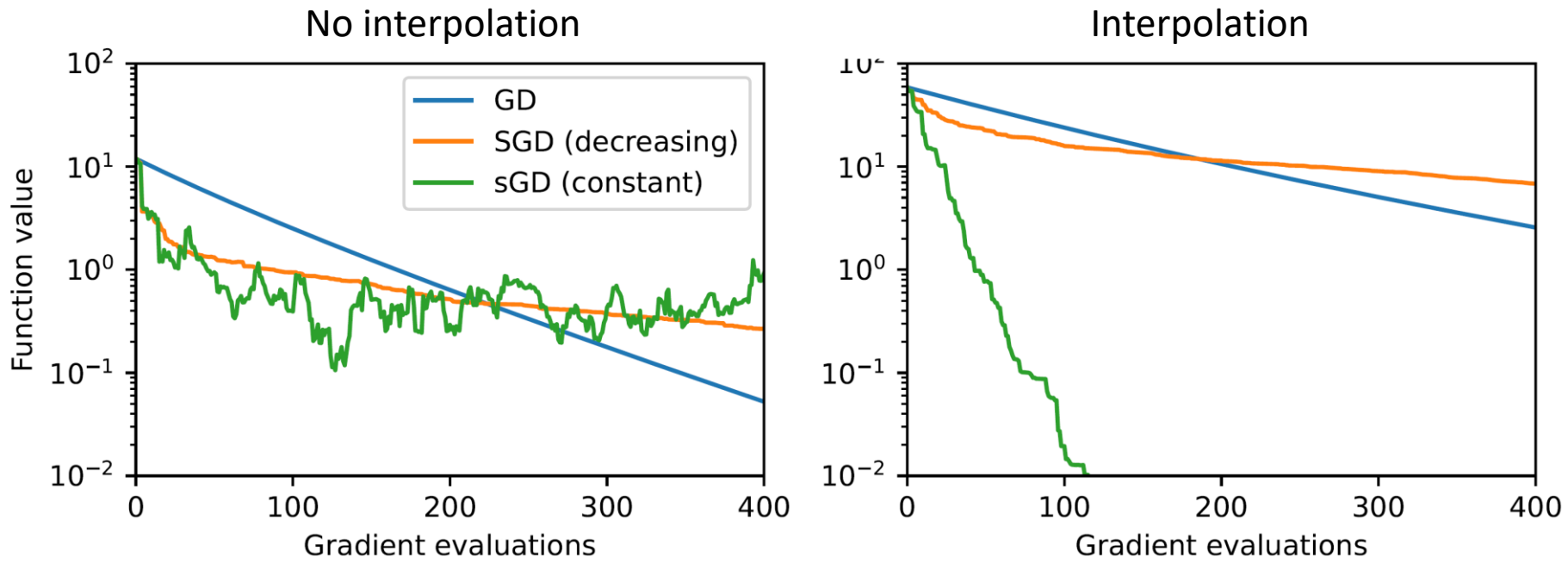
SGD under Interpolation

- A model interpolates data if it can fit data with training error 0.
  - Found to be true for many computer vision models.

- How does this affect optimization?
  - In this setting, when we minimize 'f' we also minimize each 'f_i'.
    - So at a global minimum w_* we have $\nabla f_i(w_*) = 0$ for all 'i'.
  - And this means that $\sigma_*^2 = 0$ (there is no noise at the minimum).
    - So σ_k goes to 0 as $\nabla f(w_k)$ goes to 0.

- SGD under interpolation is less affected by the second trade-off.
  - Since σ_k goes to 0 it converges like GD with the iteration cost of SGD!

Setting the SGD Step Size under Interpolation



- Under interpolation, no need to decrease step sizes.
- Still expect good performance if you are close to interpolating data.
  - Will not converge but will get close to solution, since σ_*^2 will be small.

How do I adjust the SGD step size?

- Could we somehow adjust the SGD step size as we go?
  - Accounting for both trade-offs and hoping to get "lucky"?

- Lots of methods have been proposed to do this.
  - "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: almost all of these methods are bad.

- 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 σ_k goes to 0.

How do I adjust the SGD step size?

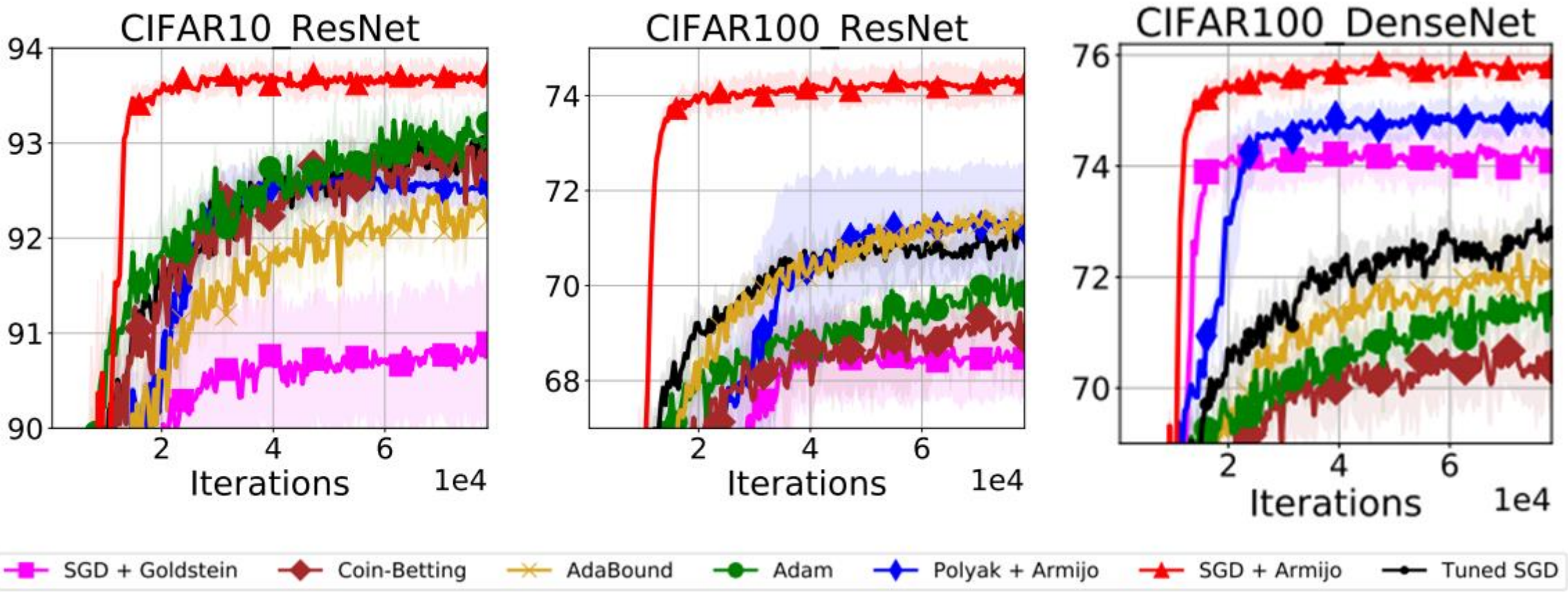
- Without interpolation:
  - I have not had good luck getting anything working well across problems.
    - And I have tried a lot of things over the years.
  - My students say that the recent "coin betting" method works well.
    - And is justified theoretically.

- For problems with interpolation (where σ_k goes to 0):
  - Can often adapt clever step sizes or line searches designed for GD.

- Example: with interpolation you can use backtracking line search.
  - "Decrease step size if it makes less progress than a step size like 1/L would."
  - Theory: "Performs as well or better as best fixed step size" (without knowing 'L').
    - And with careful implementation, cost is less in practice than trying out 2 fixed step sizes.

SGD with Armijo Line-Search with Interpolation

- SGD + Armijo outperforms other methods on many benchmarks.



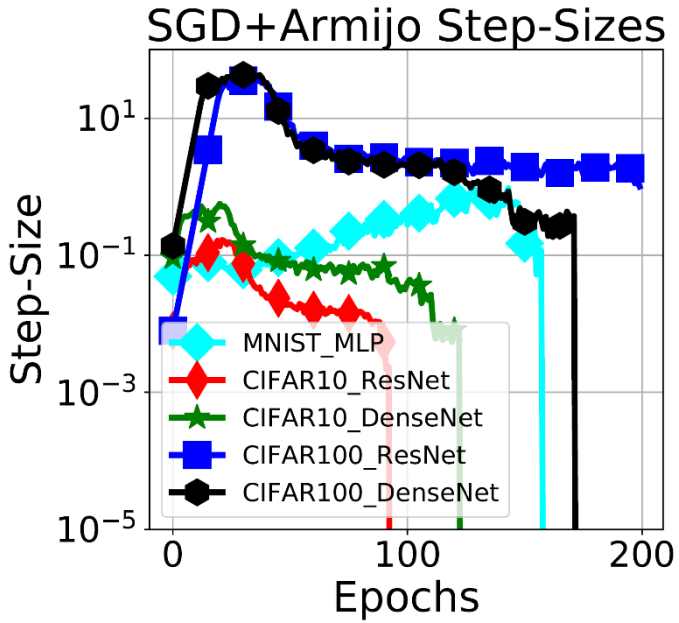
Weird Neural Network Dynamics

- General deep networks training dynamics are not well understood.

- Example: "edge of stability".
  - Classic setting: we adapt α_k based on the local value of L.
    - For GD, we decrease α_k until $\alpha_k \leq 2/L$.
  - Neural networks: local L seems to adapt to the value of α_k.
    - For GD, L increases until we have $\alpha_k \approx 2/L$.
- Example: "initialization and warm-up".
  - Optimization theory: reach stationary point from any initialization w_0.
  - Neural networks: careful normalization of w_0 is needed for good performance.
    - And we need to "warm up" the learning rate to reach a good solution.
- Example: "catapults".
  - More progress overall with occasional large increases in objective function.

SGD with Armijo Line-Search with Interpolation

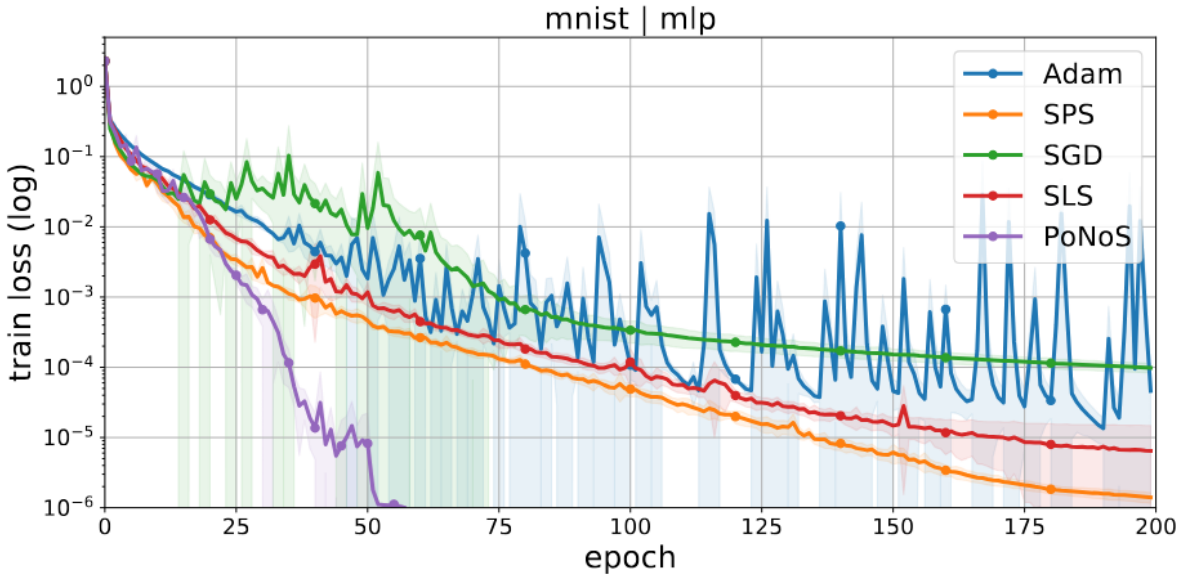
- Step sizes found using Armijo backtracking:

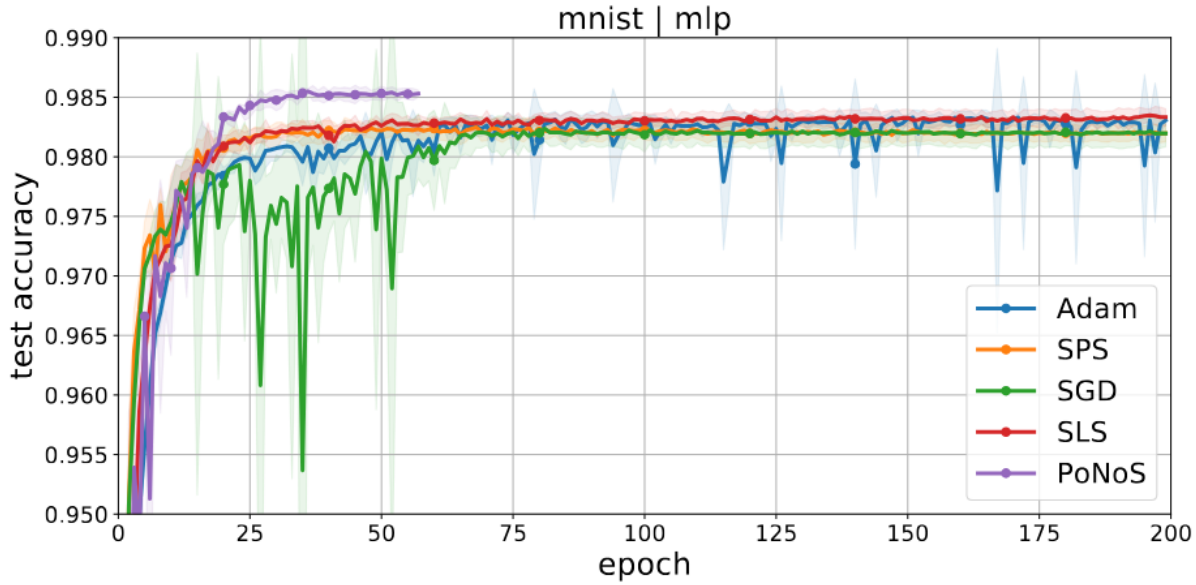


  - Uses different step sizes on different datasets at different times.
  - Seems to have a "warm-up" phase.
  - But eventually step sizes can become too small.
    - No "catapults" and eventual numerical collapse due to edge of stability.

SGD with Non-Monotonic Armijo Line Search

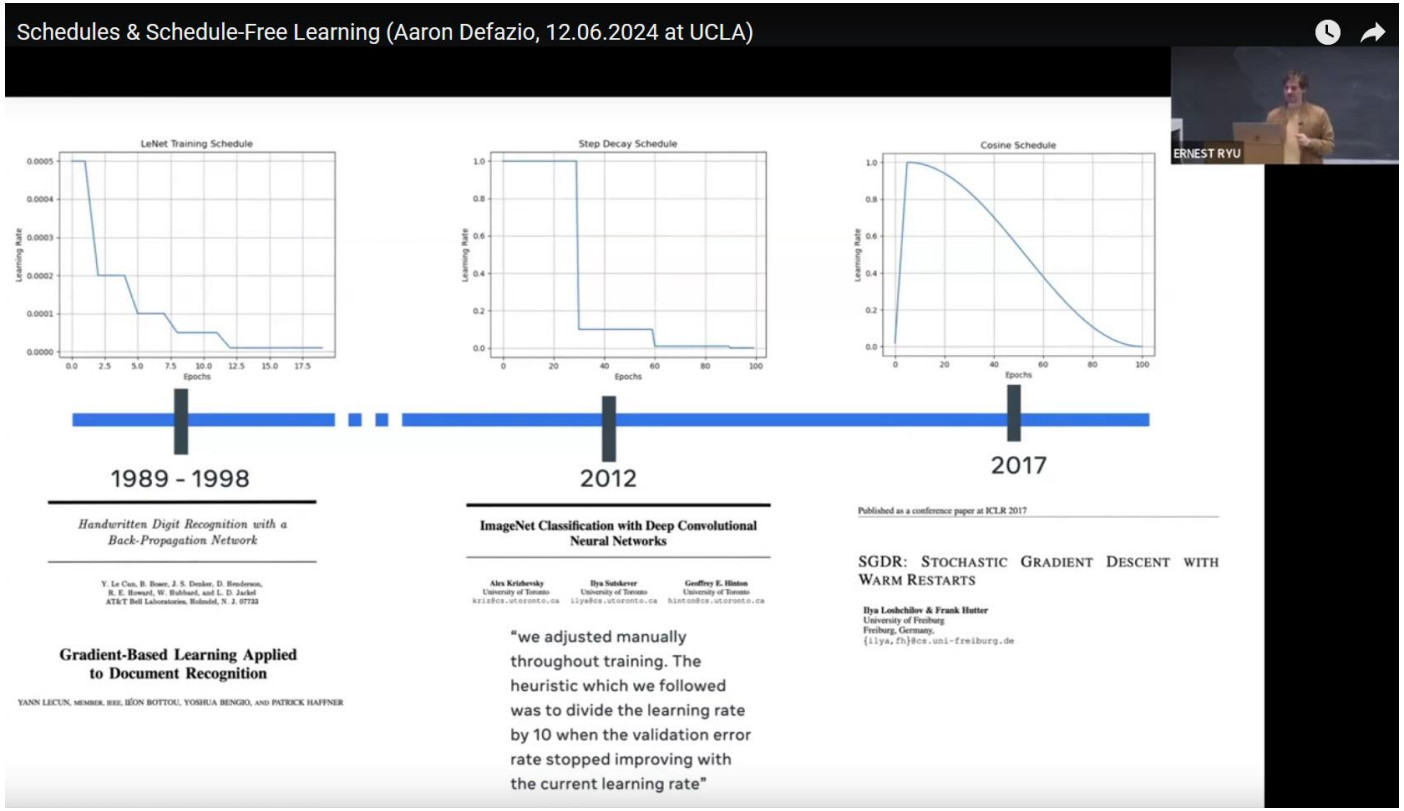
- SGD with non-monotonic backtracking:
  - Allows controlled increases in function.





  - Allows “catapults” and seems to jump to “edge of stability” immediately.
    - No warm-up or numerical collapse.

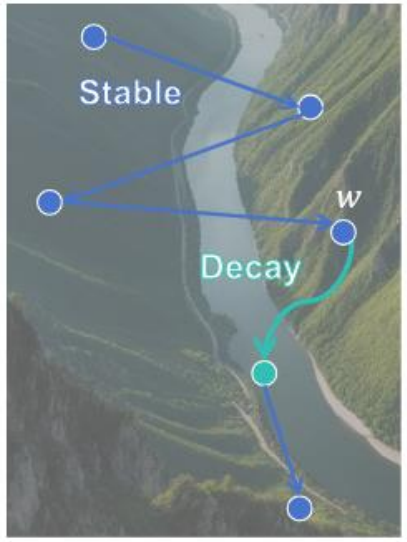
Setting the Step Size without Interpolation (Practice)



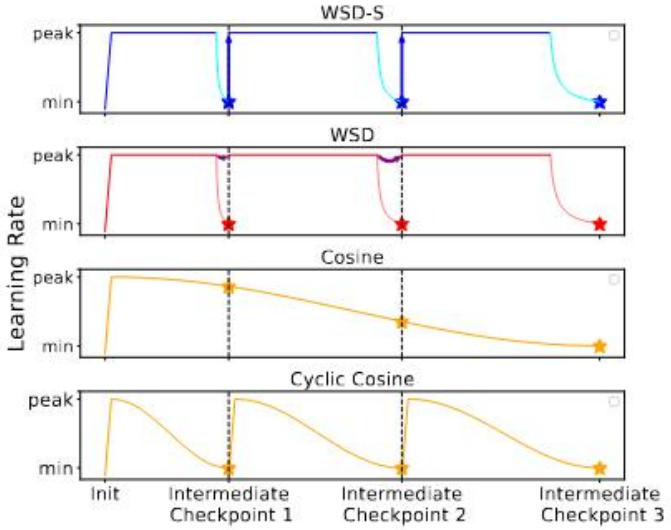
- Most popular strategies to set step size in practice:
  - "Constant for x passes through the data" and "Cosine annealing".
- Algoperf 2025: "black box" neural network training competition.
  - "Schedule-free" method used by winner to set step size.
    - Somewhat similar to cosine decay (still requires some tuning).

Step Sizes for Neural Nets: State of Affairs

- A recently-trendy strategy: "warmup stable decay".
  - Warm-up then constant step size.
  - Occasionally record "checkpoint" and solution found with smaller step sizes.
    - Then return to "checkpoint" and keep going.
    - "Assume gradient-dominated but occasionally explore noise-dominated".



(a) River Valley Landscape



(b) Learning rate schedules

https://arxiv.org/pdf/2410.05192

- Current work: "scaling laws".
  - Trying to predict step size for one problem based on smaller instances.

Question 2: How I pick 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 $\nabla f(w_k)$.

- Bigger batches lead to a longer gradient-dominated phase.
  - Trend towards larger batch sizes: GPT3 used $|B_k|$ = 3 million.
    - More recent LLM used batches of 64 million.

Mini-Batch SGD Trade-Off

- With random mini-batches, second part of trade-off changes to:

$\|\nabla f(w_k)\|^2$ vs. $\frac{\alpha_k \sigma_k^2}{|B_k|}$ ← size of mini-batch

  - "With a mini-batch size of 100, effect of noise is divided by 100."
- If $\left\|\|\nabla f(w_k)\|\right\|^2 \gg \alpha_k \sigma_k^2 / |\mathrm{B_k}|$, mini-batch SGD guarantees progress.
  - "With a mini-batch size of 100, you can use a step size that is 100-times larger."
    - As long as the step size is still less than 2/L.
  - Instead of using smaller α_k could use larger batch size to guarantee progress.

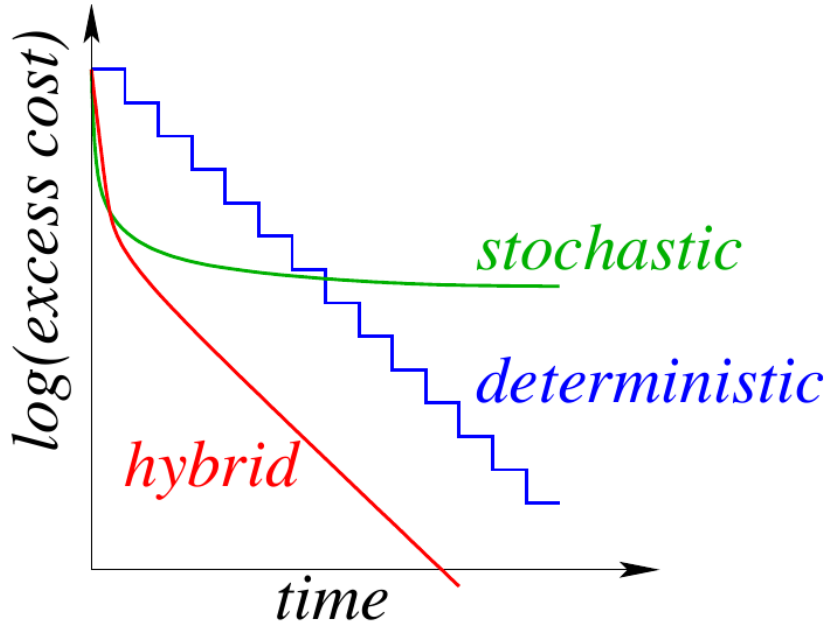
- The situation is better if you have a finite data set.
  - If you sample mini-batch without replacement from 'n' examples:

$\|\nabla f(w_k)\|^2$ vs. $\frac{\alpha_k \sigma_k^2}{|B_k|} \frac{(n - |B_k|)}{n}$ → "finite sample correction"

  - Drives the effect of σ to 0 as $|B_k|$ approaches 'n' (instead of ∞).

Gradually Growing Batch Sizes

- 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.
    - Allow using clever step sizes and line-searches.

Question 3: In what order should we process data?

Random Shuffling?

- How should we choose the random example?
  - Classic SGD theory: must sample i_k uniformly from {1,2,…,n}.
    - Or with importance sampling from a fixed distribution.

- In practice, people often use random shuffling:
  - Compute a random permutation of the training examples.
  - Go through this random permutation in order.
    - Guarantees each example is chosen at least once every 'n' iterations.

- Empirical evidence that random shuffling converges faster.
  - Theory is catching up.
    - Shuffling has proved to be faster if we do many epochs, or have high noise in gradients.

Importance Sampling?

- What about sampling i_k from a non-uniform distribution?

- Suppose you know Lipschitz constant L_i of each training example.
  - Consider biasing random sampling by the L_i values.
    - "If the gradient of example 'i' can change quickly, sample it more often."
  - This speeds convergence in "gradient dominated" region.
    - Normally SGD needs a step size less $2/\max(L_i)$, this allows step sizes less $2/\text{mean}(L_i)$.

- So importance sampling can helps until σ_* gets too large.
  - We do not expect important sampling to help when "noise dominated".

Curriculum Learning?

- Intuitive idea behind curriculum learing:
  - "Train on the easy examples first, then slowly learn the hard examples".
    - Unfortunately, not always easy to defined "easy" and often does not help much.

- Adaptive L_i importance sampling looks like curriculum learning:
  - The "local" L_i is small if the example is correctly classified.

- Curriculum used in some moderns LLMs:
  - First train on trillions on tokens.
  - Near the end of training, use higher-quality or task-specific data.
    - Liked vetted data or instruction data (to bias model towards specific behavior).

Question 4: Are there faster algorithms than SGD?

(ignoring Adam and friends for the moment)

Faster Deterministic Algorithms than GD

- We have faster algorithms than GD in specific settings:
  - Heavy-ball and conjugate gradient a momentum term (for some $\beta_k < 1$).

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

  - Nesterov's accelerated gradient method adds a second-order-ish correction:

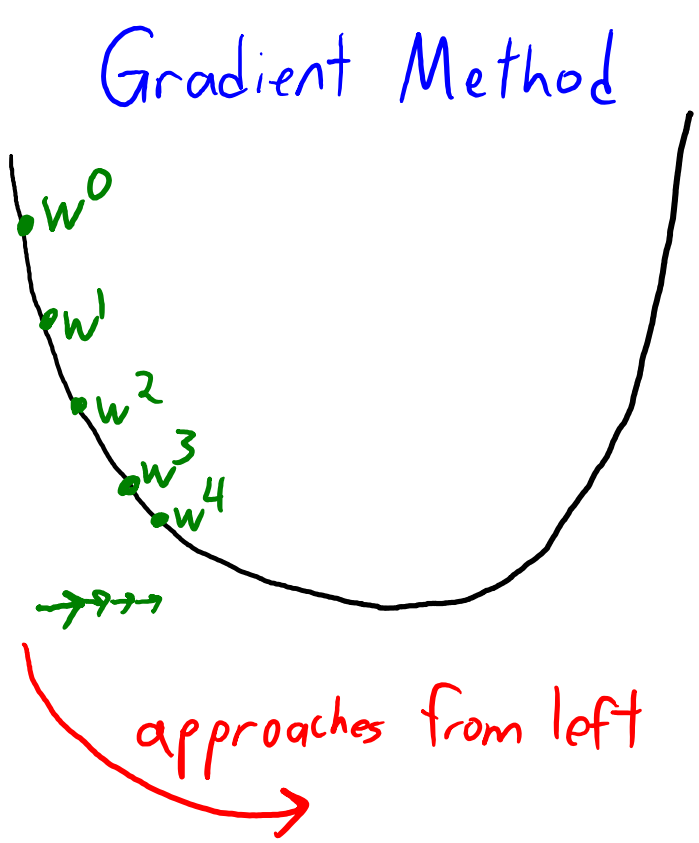
$$w_{k+1} = w_k - \alpha_k \nabla f(w_k) + \beta_k (w_k - w_{k-1}) + \alpha_k \beta_k (\nabla f(w_k) - \nabla f(w_{k-1}))$$

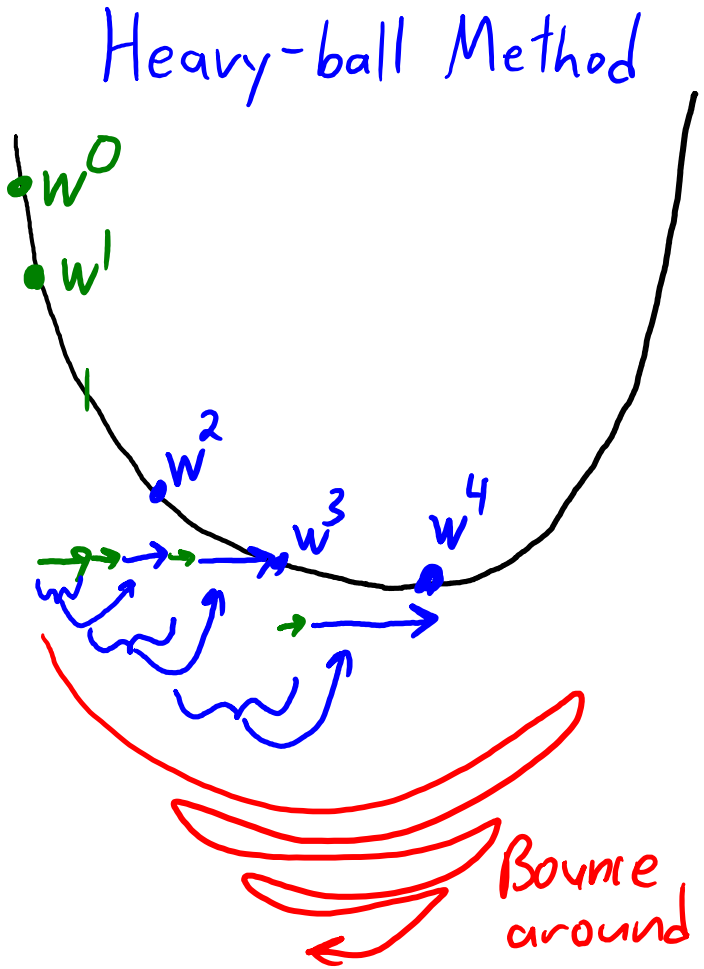
  - Newton's method scales gradient based on second derivatives.

$$w_{k+1} = w_k - \alpha_k [\nabla^2 f(w_k)]^{-1} \nabla f(w_k)$$

    - Many approximations that avoid Hessian operations (quasi-Newton, Hessian-free Newton).

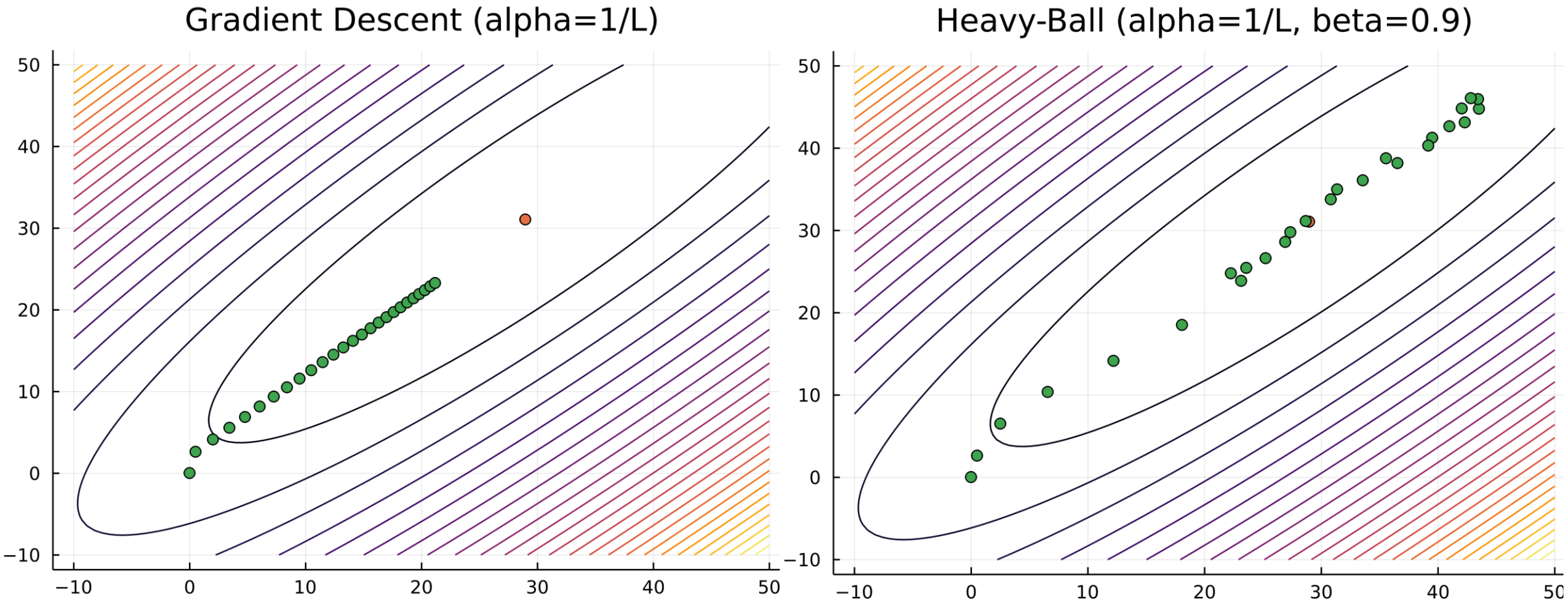
Gradient Descent vs. Heavy-Ball Method





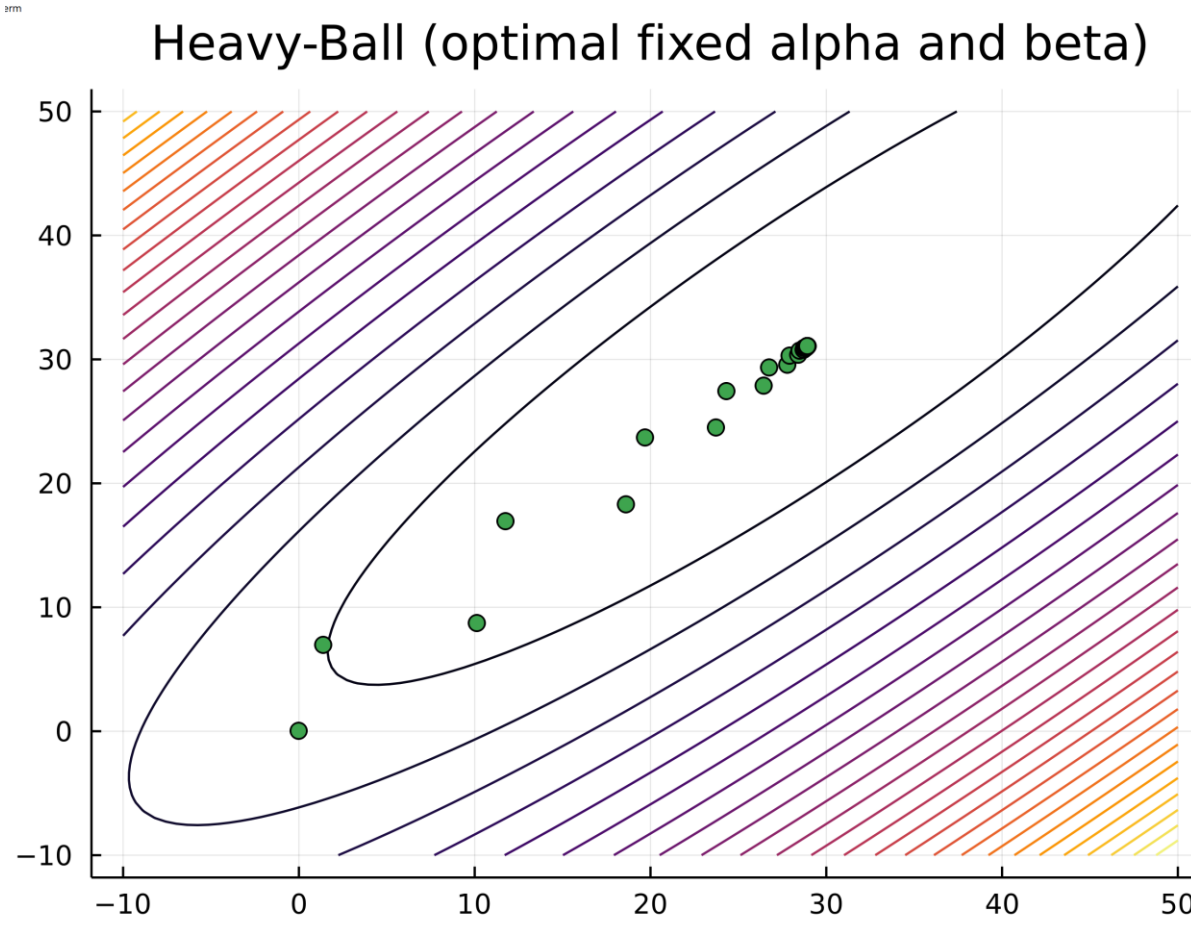
Gradient Descent vs. Heavy-Ball Method

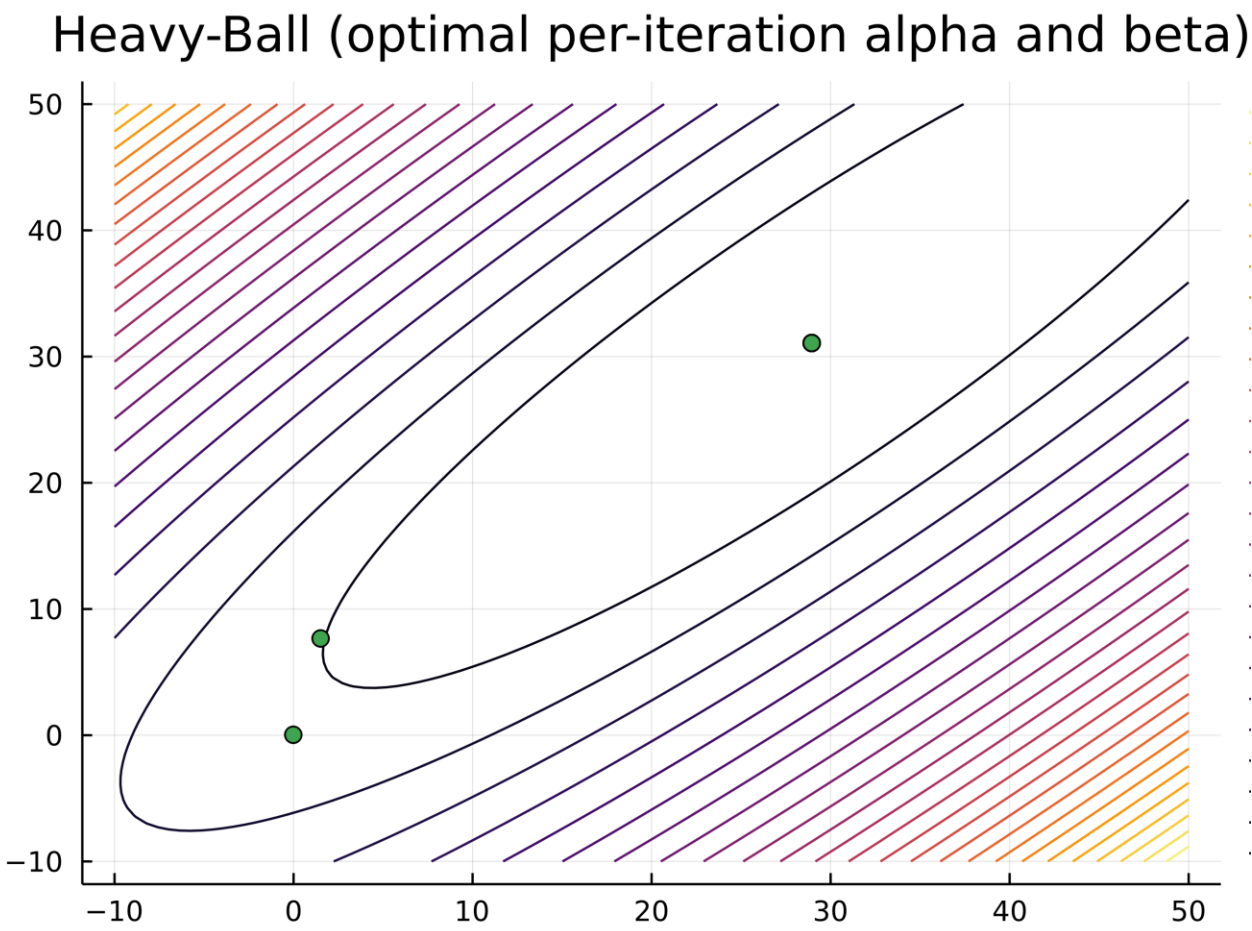
- Adding momentum to GD with fixed step size and momentum:



Gradient Descent vs. Heavy-Ball Method

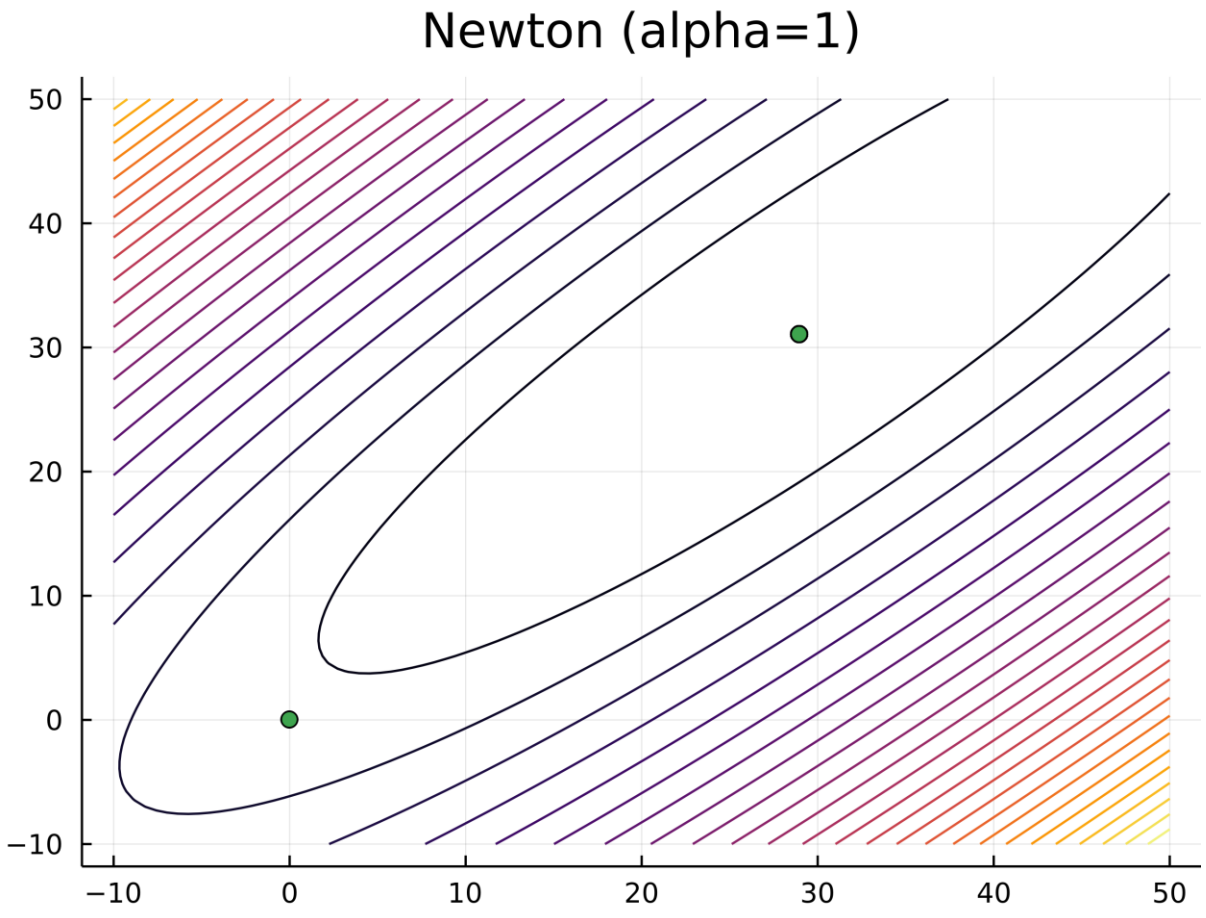
- For deterministic quadratics, exist clever ways to set α_k and β_k.

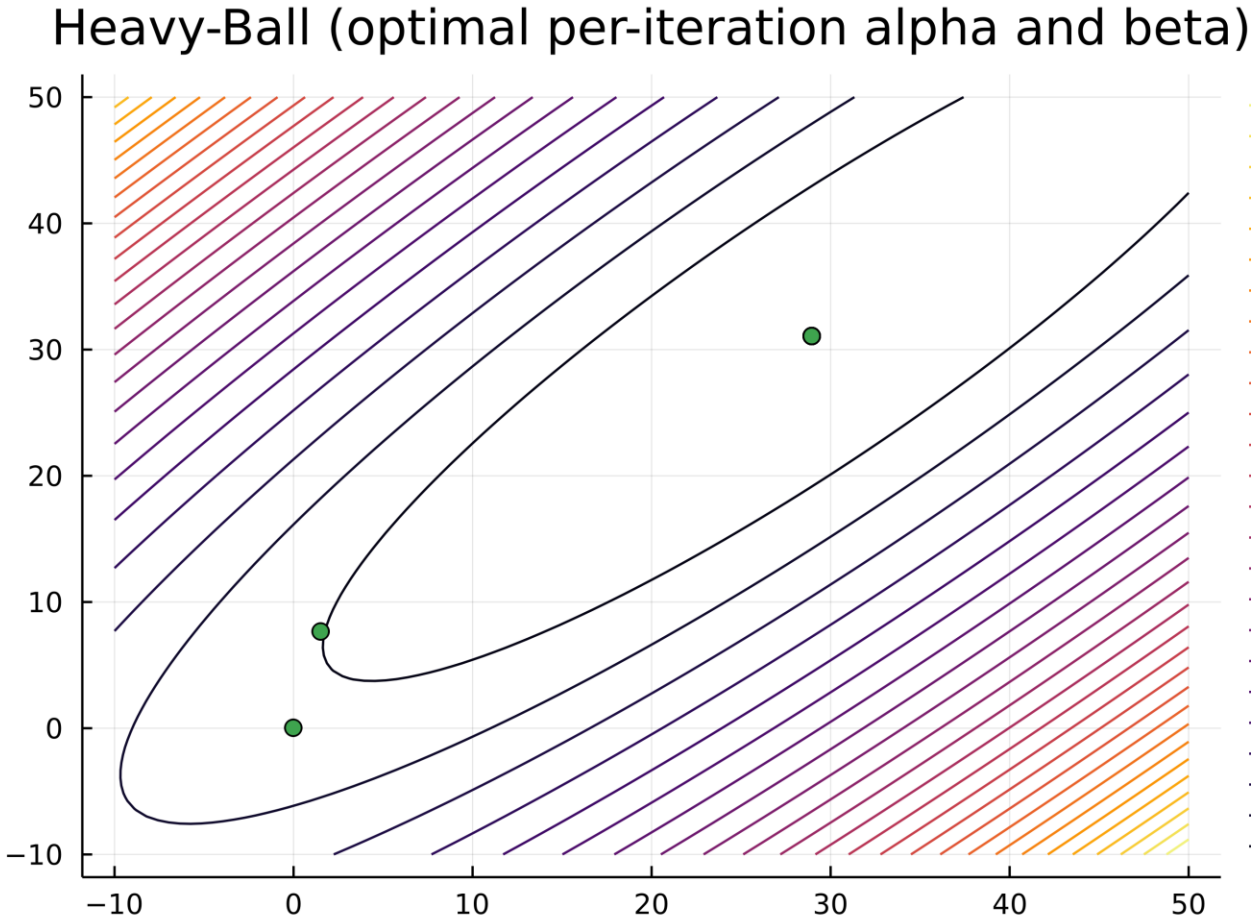




Gradient Descent vs. Heavy-Ball Method

- For deterministic quadratics, Newton converges in 1 step.





Faster Stochastic Algorithms than GD (Gradient Dominated)

- Are there faster algorithms than SGD? It depends!

- In gradient-dominated phase (low noise):
  - Ideas like momentum/Nesterov/Newton can make SGD converge faster.
    - Improve dependence on 'L'.
  - Useful on early iterations, with interpolation, or with large/growing batches.

- In noise-dominated phase:
  - These tricks do not lead to faster convergence.
    - Momentum/Nesterov/Newton do not improve the dependence on σ_k.
  - These methods can even amplify the effect of the noise.
    - You might need momentum parameter β_k to converge to 0.
      - And probably should not trust local second-order approximations.

Faster Stochastic Algorithms than GD (Noise Dominated)

- Interpolation or large batches avoid noise-dominated phase.
  - But interpolation does not make sense for LLMs.
  - And large batches can increase the iteration cost.

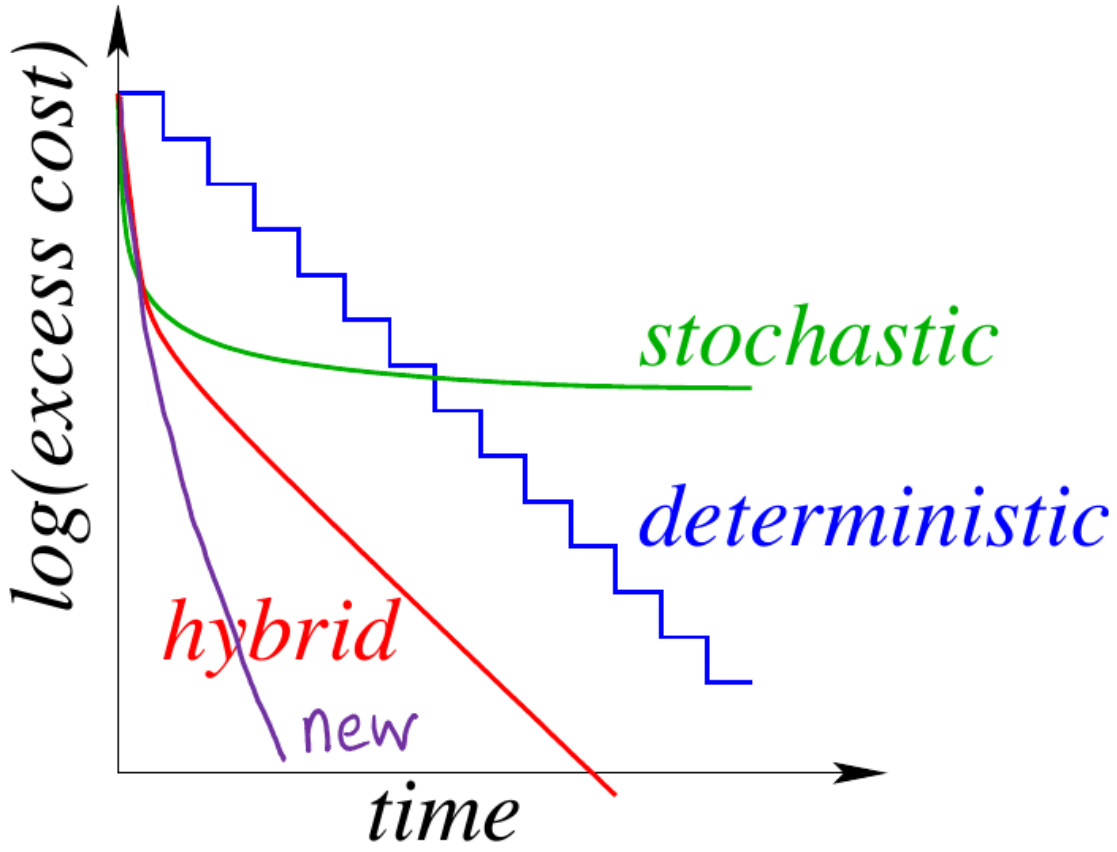
- Alternately, there are faster algorithms for the noise-dominated phase.

- Variance-reduced stochastic gradient methods (like SAG and SVRG):
  - Make SGD converge faster in noise-dominated settings.
    - And only need 1 random example per iteration.
  - Key idea: design an estimate of $\nabla f(w_k)$ whose variance converges to 0.

- Variance-reduced methods remove second part of trade-off.
  - This improves convergence speed a lot for noise-dominated settings.

Faster Stochastic Algorithms than GD (Noise Dominated)

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



Faster Stochastic Algorithms than GD (Noise Dominated)

- Variance reduction is not needed for gradient-dominated settings.
  - Here, variance-reduced SGD might be slower than SGD with fixed step size.

- Empirically, variance-reduction does not help for deep learning.
  - Probably because most of training is in gradient-dominated setting.

- Example for variance-reduction helps: GANs.
  - GAN loss cannot be driven to zero.
  - So variance-reduced SGD can lead to faster convergence.

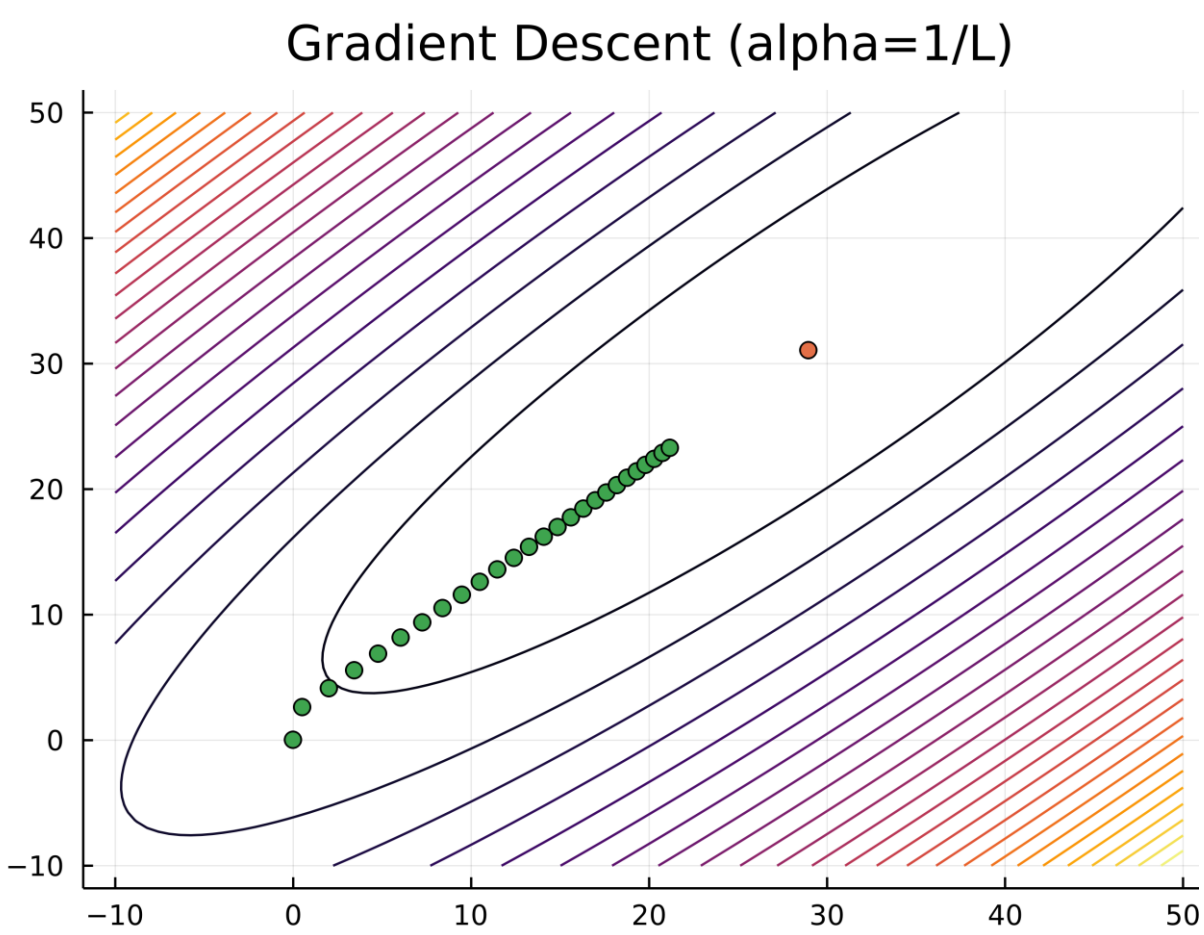
Question 7: Should I just use Adam with default parameters?

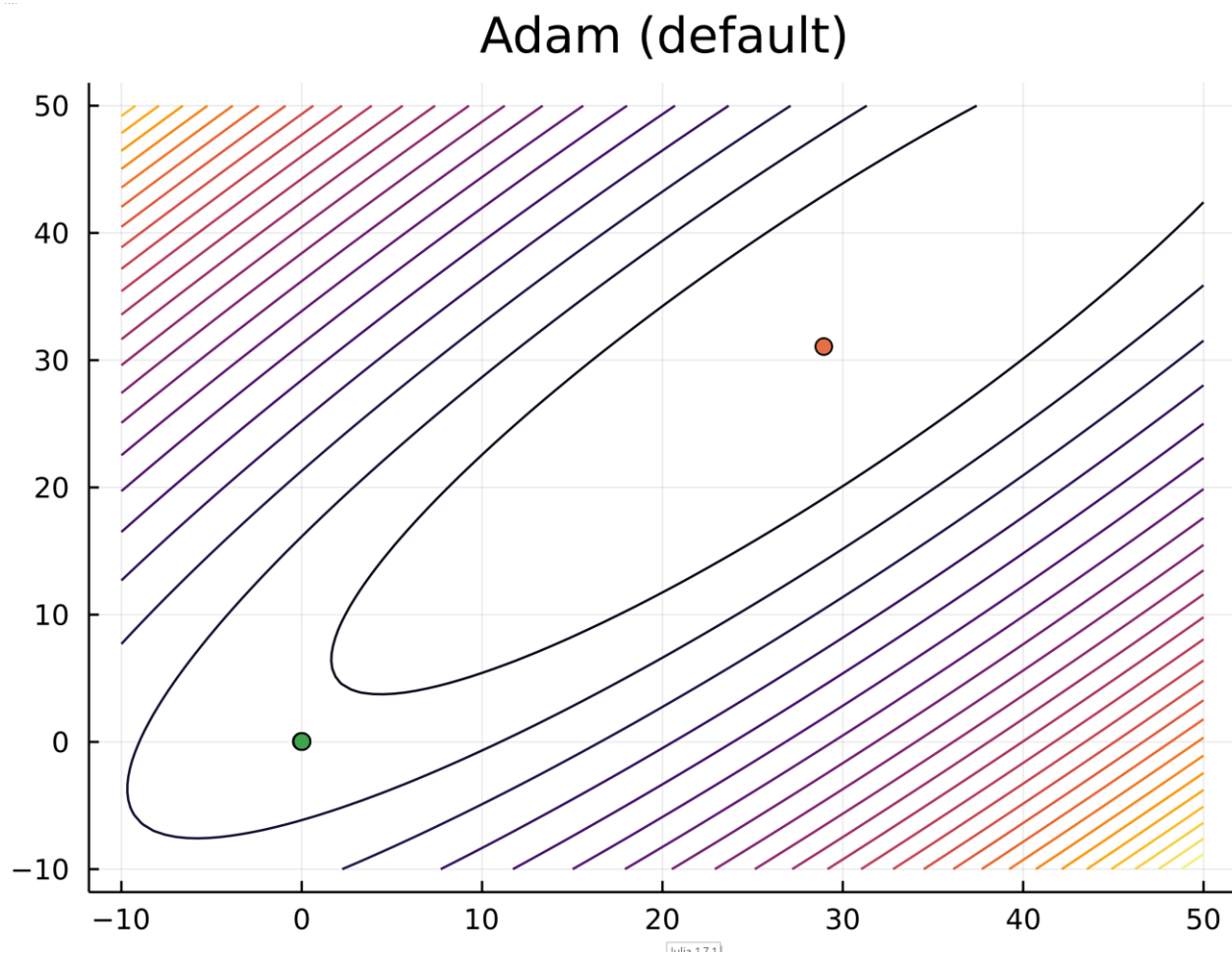
The Perplexity of the Adam Optimizer

- 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.

Gradient Descent vs. Adam

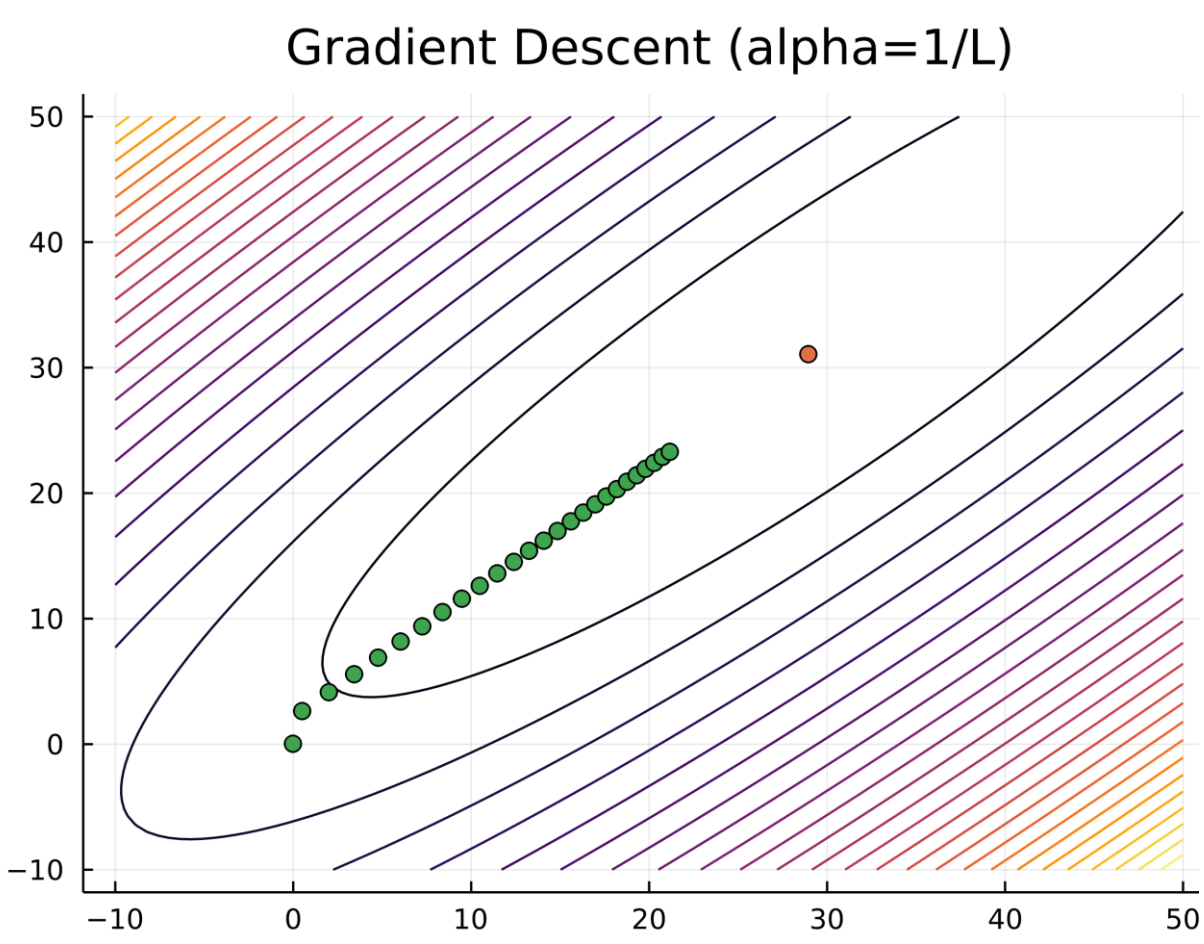
- GD vs. Adam:

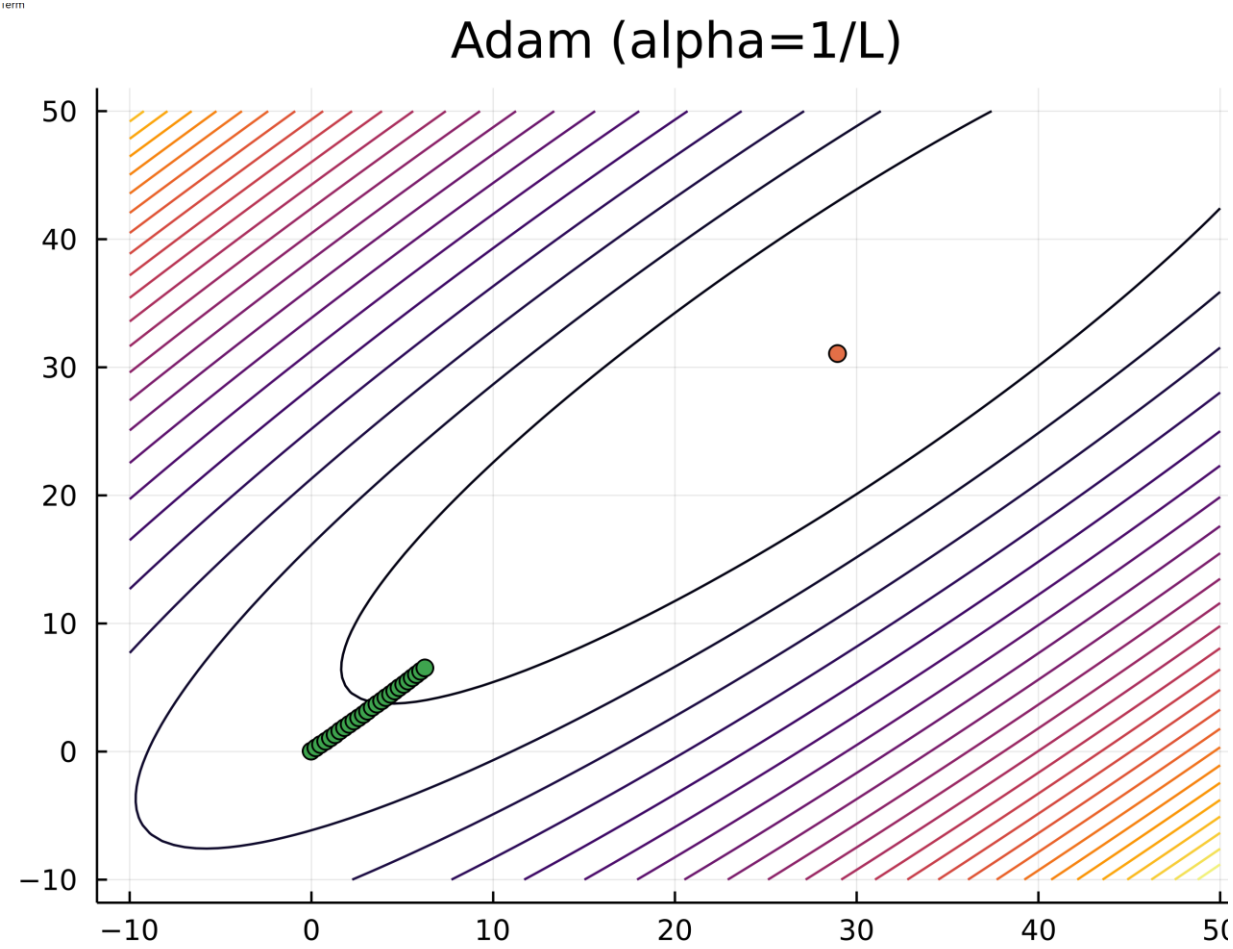




Gradient Descent vs. Adam

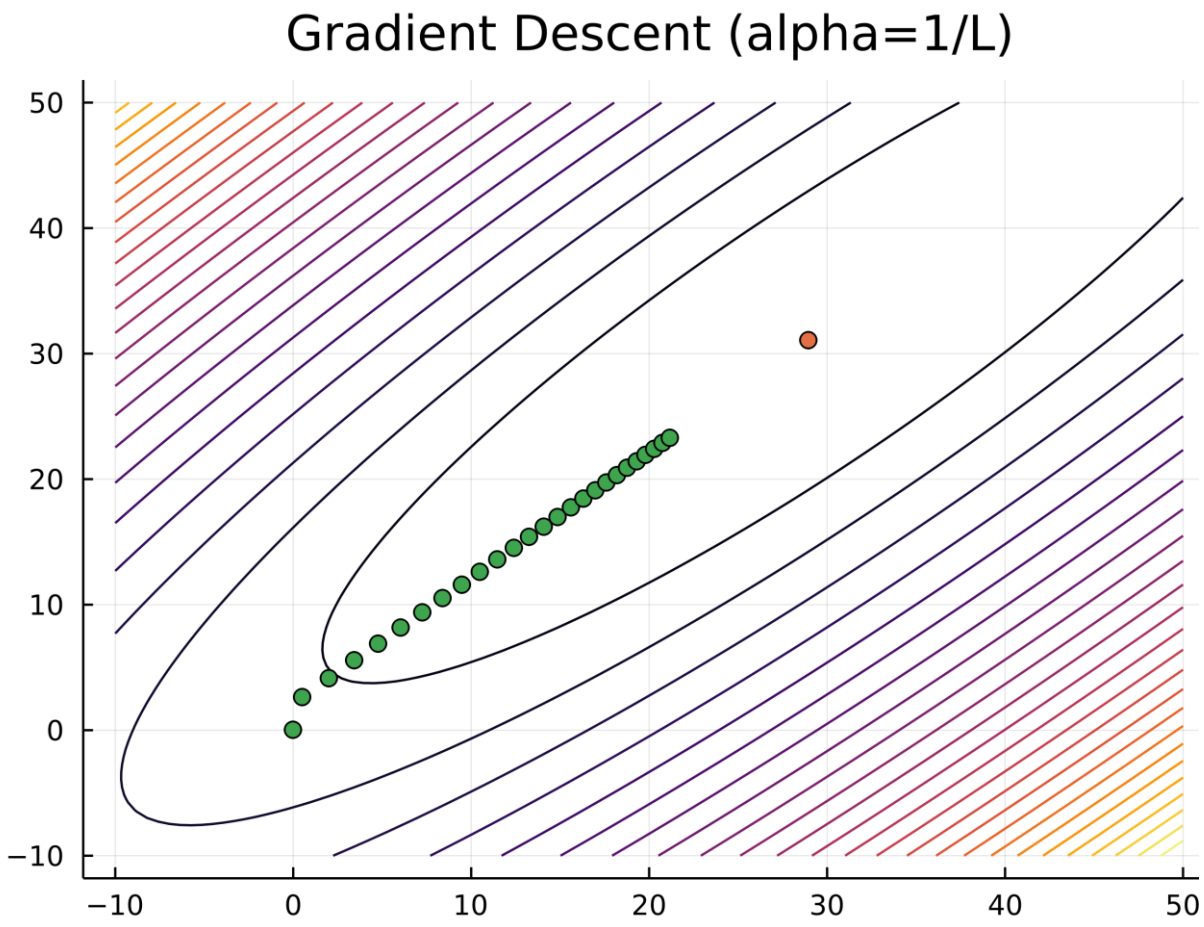
- GD vs. Adam:

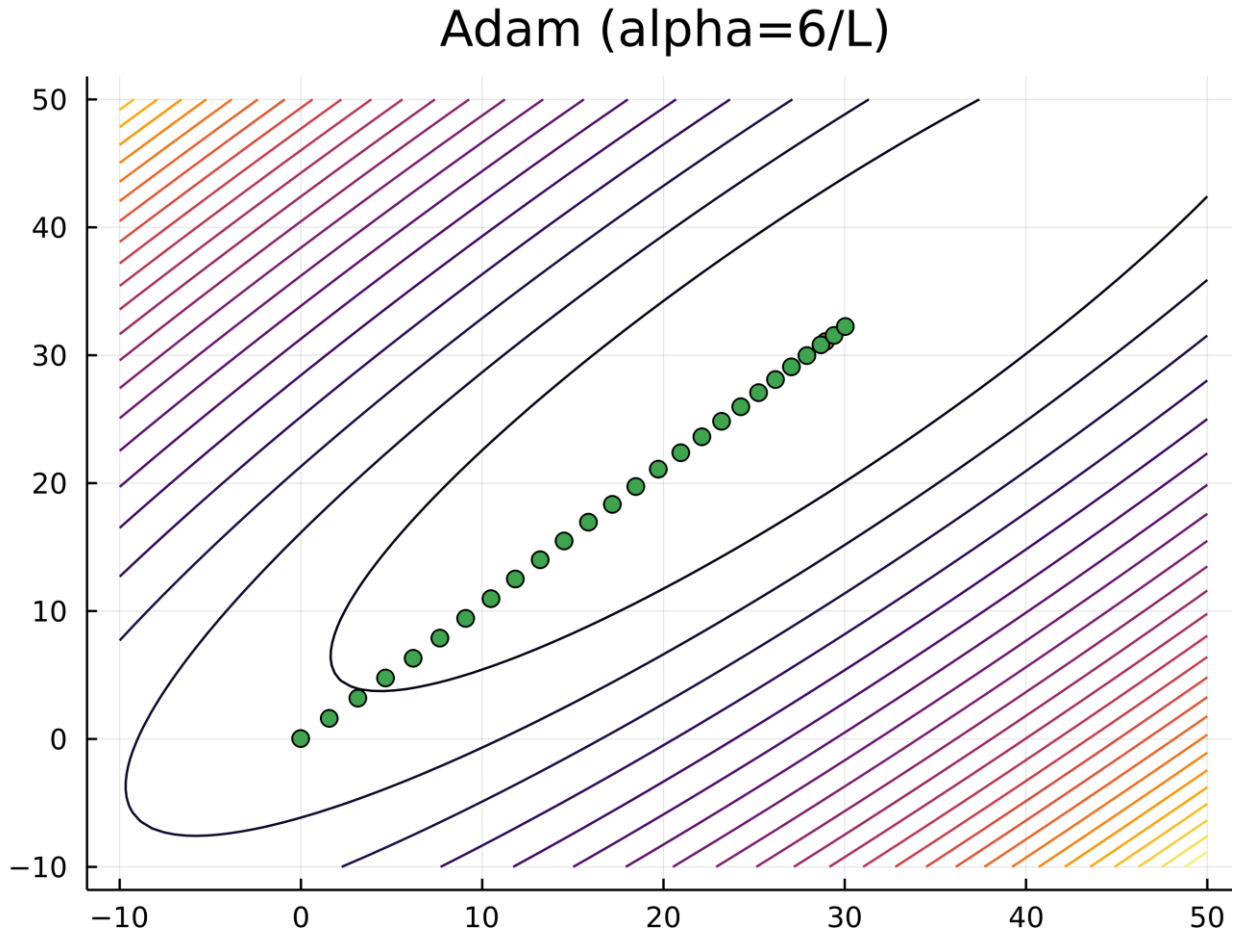




Gradient Descent vs. Adam

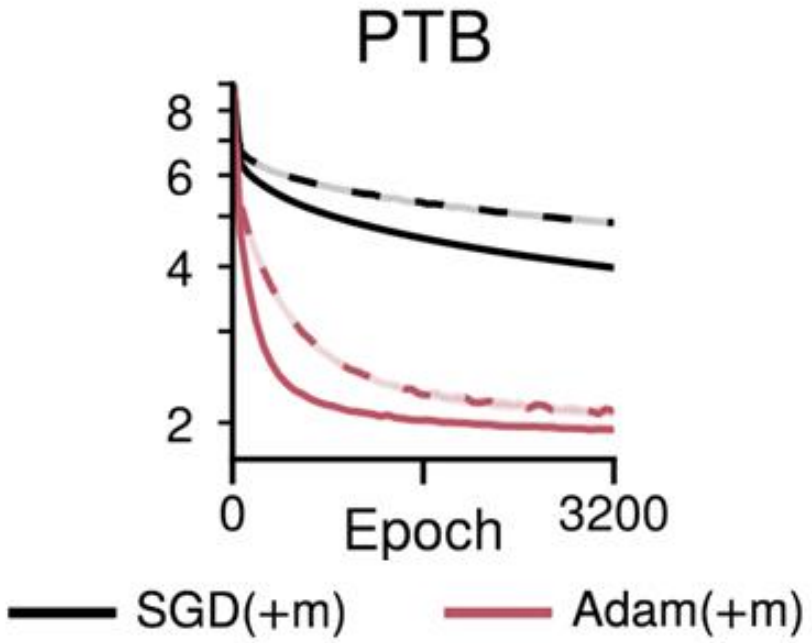
- GD vs. Adam:





Which Phase does Adam help with?

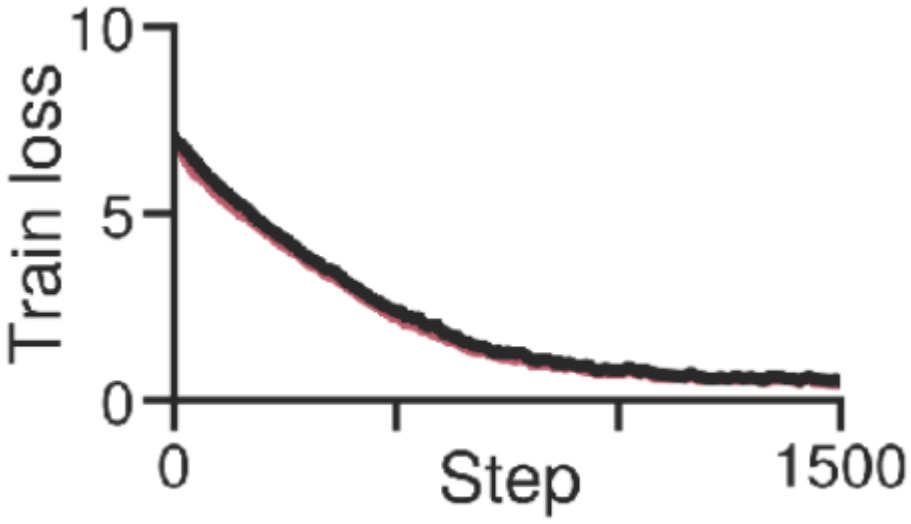
- Is Adam helping in gradient-dominated or noise-dominated phase?
  - We can change the batch size to vary the noise:



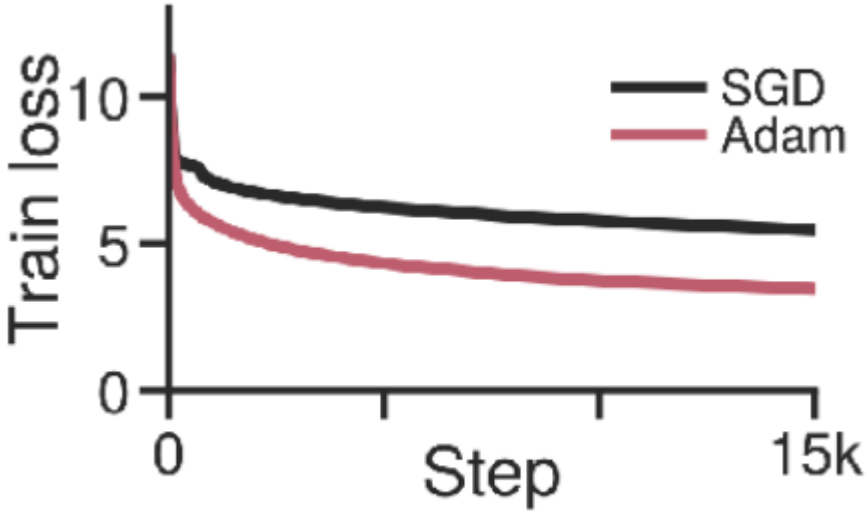
- For small batch sizes, Adam does not help.
- For large batch sizes, gap between Adam and SGD grows (when Adam helps).
  - Adam is helping in gradient-dominated phase, not noise-dominated phase.

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.



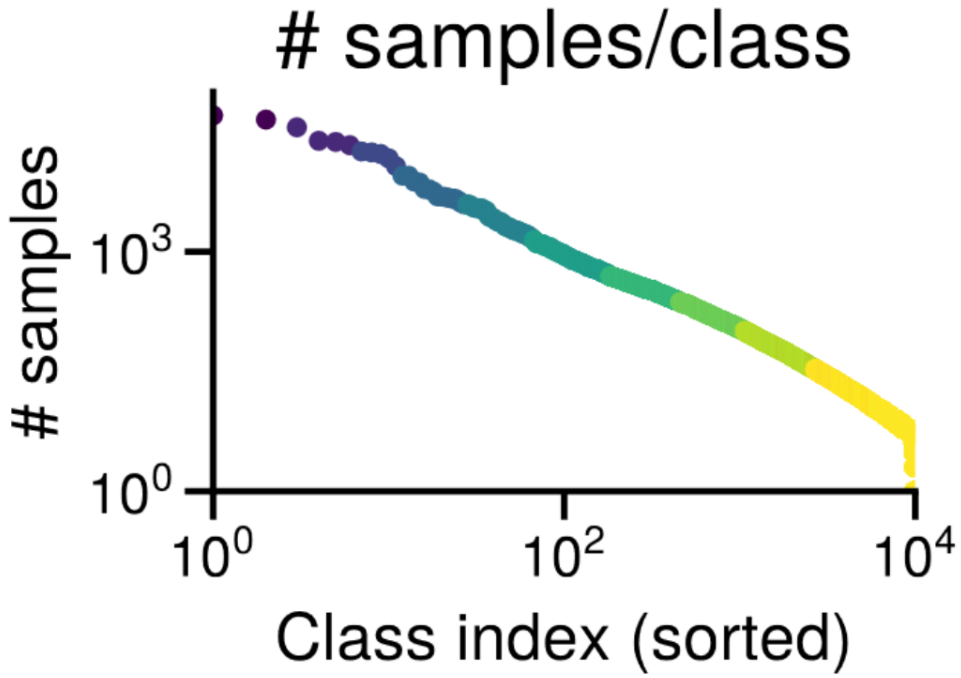
ResNet/ImageNet



GPT2-small/WikiText-103

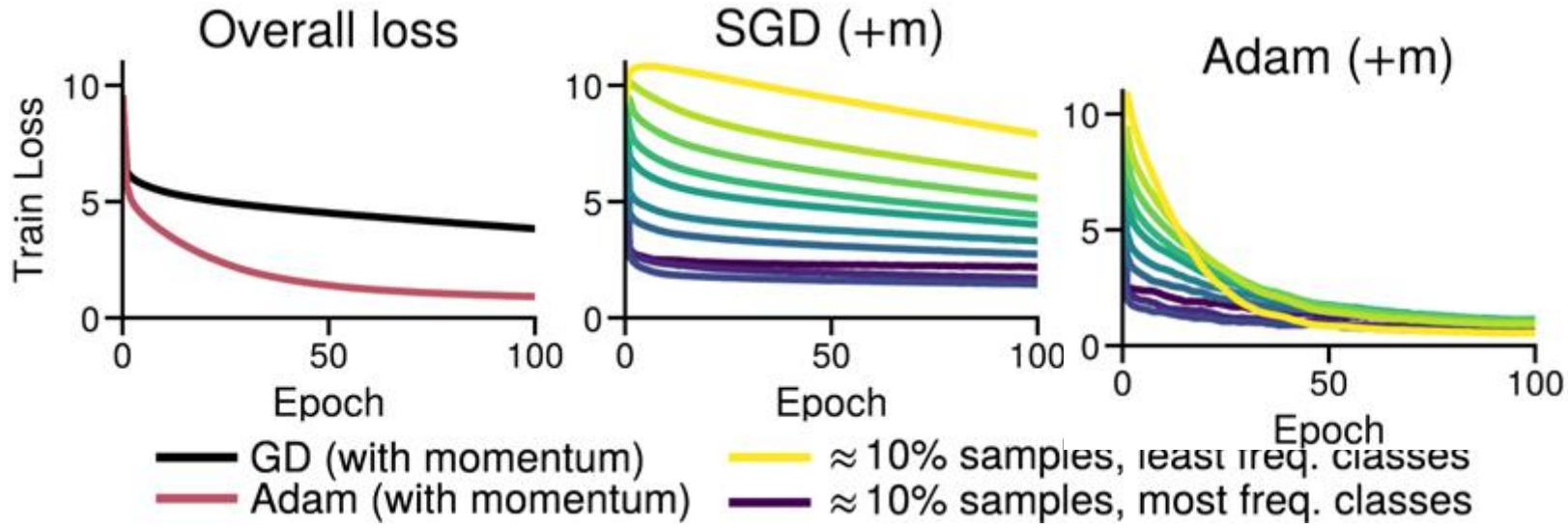
Adam on Natural Language vs. Computer Vision

- Classic explanation for effectiveness of 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.



Adam vs. SGD with Heavy-Tailed Labels

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

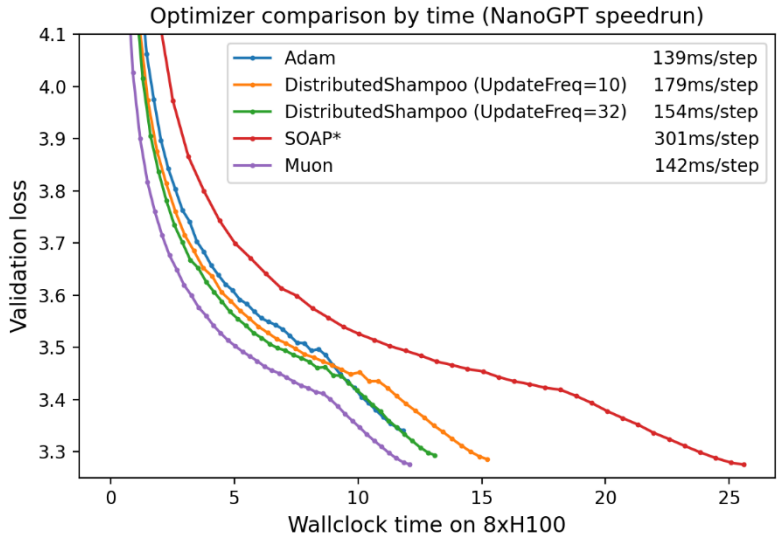


- Adam makes similar progress on all labels.
  - So it most examples have rare labels, Adam still makes progress.
- Adam outperforms SGD on vision datasets with heavy-tailed labels.

(short digression before we wrap up)

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.
  - Treats intermediate layers as matrices, trying to orthgonalize update.
  - Related to earlier work on spectral descent.

Gradient of the Step Sizes?

- An old and intuitive idea:
  - Set individual step sizes by taking gradient with respect to step sizes.
- Has appeared under various names:
  - Hypergradients, meta-gradient, IDBD, SMD, ASA, and so on.
- Some exciting recent developments:
  - We gave the first provably-faster version (multi-dimensional backtracking).
    - Madeleine Udell's group at Stanford has now proposed others.
  - Several groups at the U of A have proposed empirically-effective versions.
    - Like MetaOptimize and Stream-x.

- I am not sure if these are yet robust and practical, but am excited.

Take-Home Message

- Trade-off between gradient norm, and step-size*variance/batch size.

$$\|\nabla f(w_k)\|^2 \quad \text{vs.} \quad \frac{\alpha_k \sigma_k^2}{|\mathcal{B}_k|}$$

- Often leads a gradient-dominated phase then a noise-dominated phase of training.

- Many ways to improve performance during gradient-dominated phase:
  - Big step sizes, line search, importance sampling, momentum, second-order, Adam.

- Performance often worse and more erratic in the noise-dominated phase.
  - You can avoid noise-dominated phase by using large batch sizes or using growing batch sizes.
  - Under interpolation, there is no noise-dominated phase.

- If you cannot avoid noise-dominated phase, a few tricks are still possible:
  - Decreasing step sizes, random reshuffling, variance reduction.