Training Neural Networks

How we Train

Lecture 17

Pierfrancesco Beneventano
pierb@mit.edu

Massachusetts Institute of Technology

Part 0

Recap from last class: that was preliminaries.

Set the Plan

- We have data coming in real life: $X \sim P$
- We need to find the neural network $f_\theta(X)$ that has the right output $\varphi(X)$.
- A measure of the Error $distance(\phi(\,.\,), f_\theta(\,.\,))$.

They exist.

Can we find them?

~~**Idea 1:** Set by hand the parameters such that it works.~~

We call this *Training*

Our new *Goal*:

1. Sample some dataset $\mathscr{D} = \{(X_i, \varphi(X_i))\}_{i=1}^{n}$
2. See if we can adapt $f_\theta(\,.\,)$ to $\varphi(\,.\,)$ on $\mathscr{D}$.

How can we do it?

- Solving a non-linear system:

$$\begin{cases} f_\theta(X_1) = \varphi(X_1) \\ f_\theta(X_2) = \varphi(X_2) \\ \quad\vdots \\ f_\theta(X_n) = \varphi(X_n) \end{cases}$$

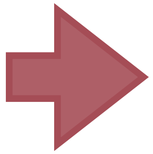
The good algorithms to solve (non-linear) systems are minimizations.

- How do you find it computationally?

Message 1

We have:

- A dataset.
- Some compute.



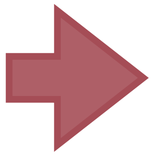
Training means using those to get a model

Learnability *vs* Optimization

We have:

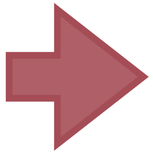
- A dataset.
- Some compute

  (Memory + number of computations constraint).



Do we have enough samples?

Learnability



In case, does the training routine find it within the compute?

Optimization

Compute / Memory

- The dimensionality N of the problem $\left(\theta \in \mathbb{R}^N\right)$ is very high, e
- We have many n loss is an

$$\frac{1}{n}\sum_{x \in D} L(\theta, x)$$

We work with:
(for both learnability and compute)

Mini-batch Gradient Methods

Gradient-based

$$\theta_{k+1} = \theta_k - \eta \nabla L(\theta_k)$$

Mini-batch methods

Zeroth Order Methods

- The cost of a gradient (with back-prop) is $\approx 2 \times$ forward pass
  $\Longrightarrow$ a gradient step is $\approx 3 \times$ forward pass.

- Zeroth order methods have $\approx 2m$ forward passes.

$$g = \frac{1}{m} \sum_{i=1}^{m} \frac{L(\theta + \sigma u_i) - L(\theta - \sigma u_i)}{2\sigma} u_i \,.$$

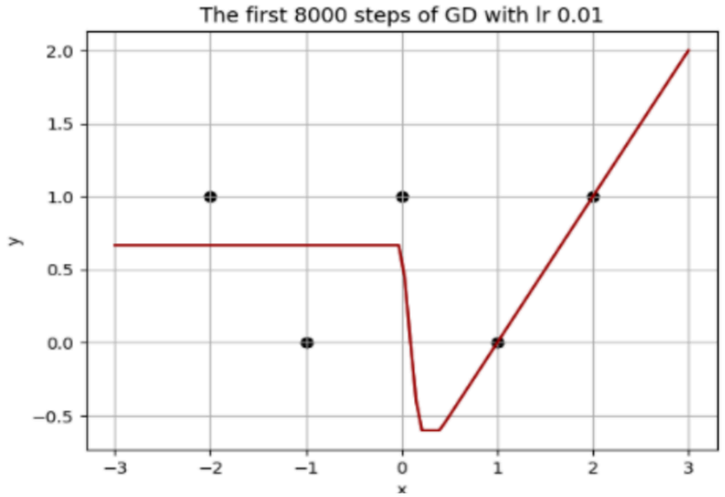
- The accuracy after $T \times$ forward pass compute is:

$$\frac{dimension}{T} \quad \textit{vs} \quad \frac{1}{T}.$$

We will do gradients!

REGIME 2

- ***Not enough exact*** data!
- Enough compute



We want a model that:

- Can fit them all.
- It's the simplest such model.

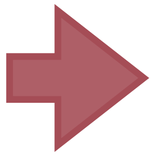
A training pipeline which goes to the best global minima.

Regime 3

- Enough exact data (all of them!)
- ***Not enough*** compute!

We want a model that:

- (Just) can fit them all.



A training pipeline that allows it within my compute!

Contents

This Class

Infinitesimally, to go down *optimally* we follow the gradient:

$$\begin{cases} \theta_0 = \text{initialization} \\ \dfrac{d\theta_t}{dt} = -\nabla L(\theta_t) \end{cases}$$

How to pick initialization?

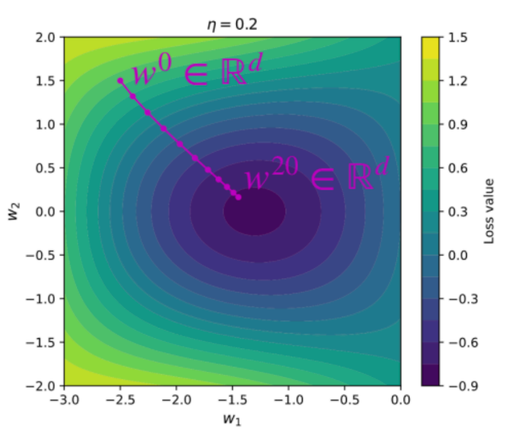
What is the optimal discrete step size version of this?

Gradient-Based Methods

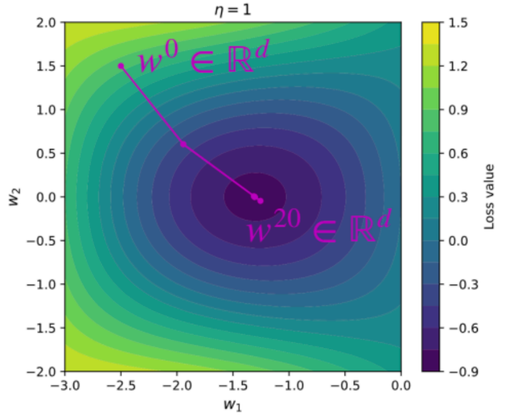
- **Gradient-Based Methods**
  - **Choosing Step Size**
  - In practice
- Initialization
- Scaling up

Pictures partly taken by Orvieto (2025)

Connecting Finite to Infinitesimal



Low η



High η

- Gradient is an infinitesimal thing.
- Gradient descent is a local but not infinitesimal.
- There is a gap there. How should we treat it, what should the size of the step be and depend on?
- Is there a more general framework for gradient-based methods?
- All is about deciding how to pick the step size!

Gradient Descent

- Infinitesimally, the optimal way to go down is by following the gradient:

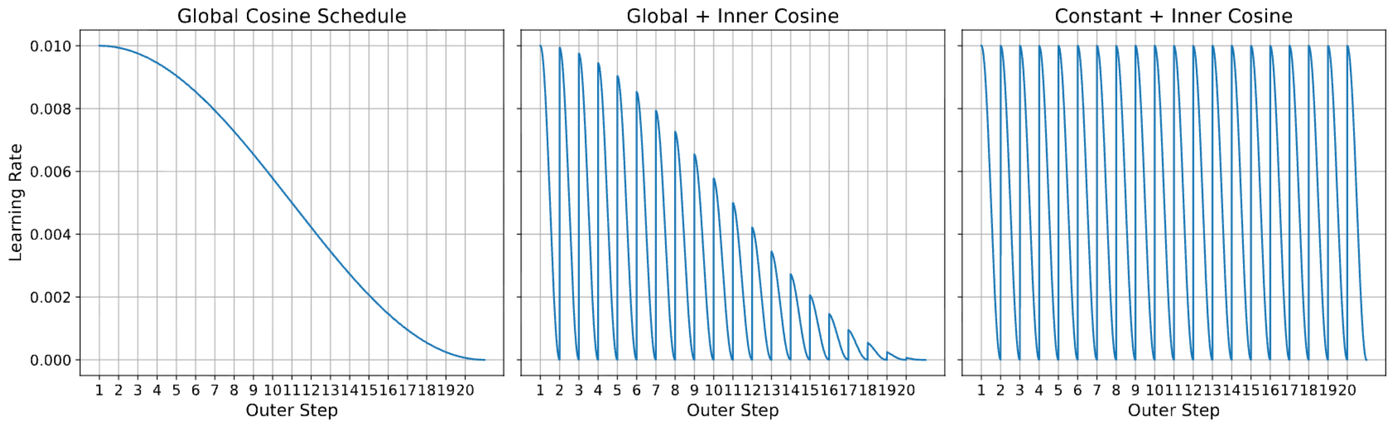
$$\begin{cases} \theta_0 = \text{initialization} \\ \dfrac{d\theta_t}{dt} = -\nabla L(\theta_t) \end{cases}$$

How do you do it *optimally* with *finite* step?

- We want to find a good (possibly high dimensional) step size $\eta : \theta_t \mapsto \mathbb{R}^{d\times d}$:

$$\theta_0 = \text{initialization}, \qquad \theta_{t+1} = \theta_t - \langle \eta_t, \nabla L(\theta_t) \rangle \qquad \text{is optimal.}$$

STEP-SIZE SCHEDULES



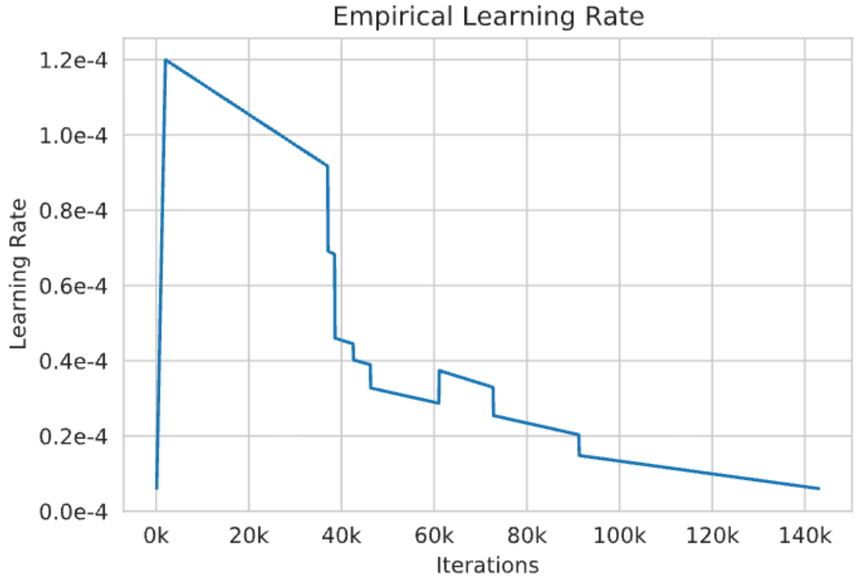
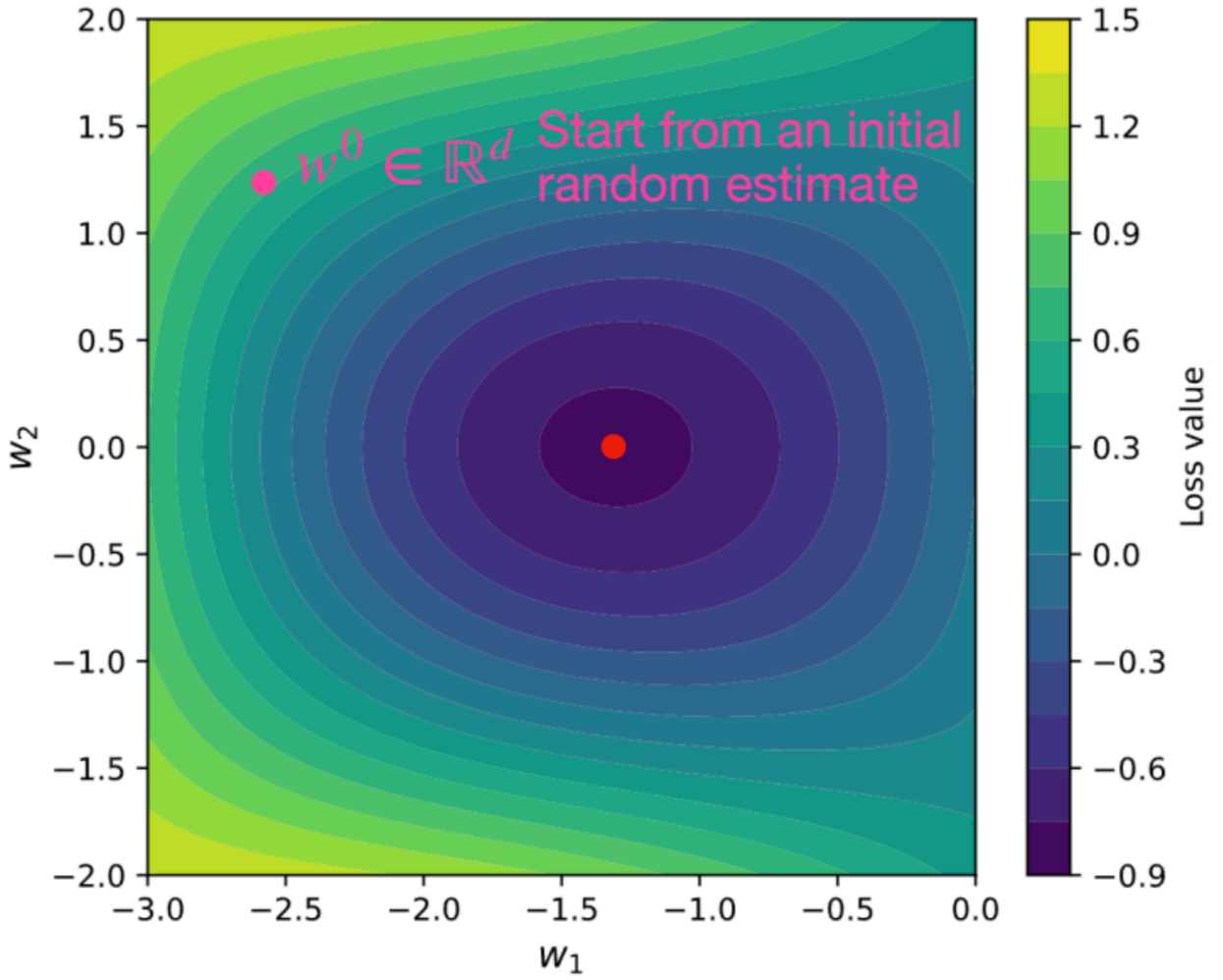


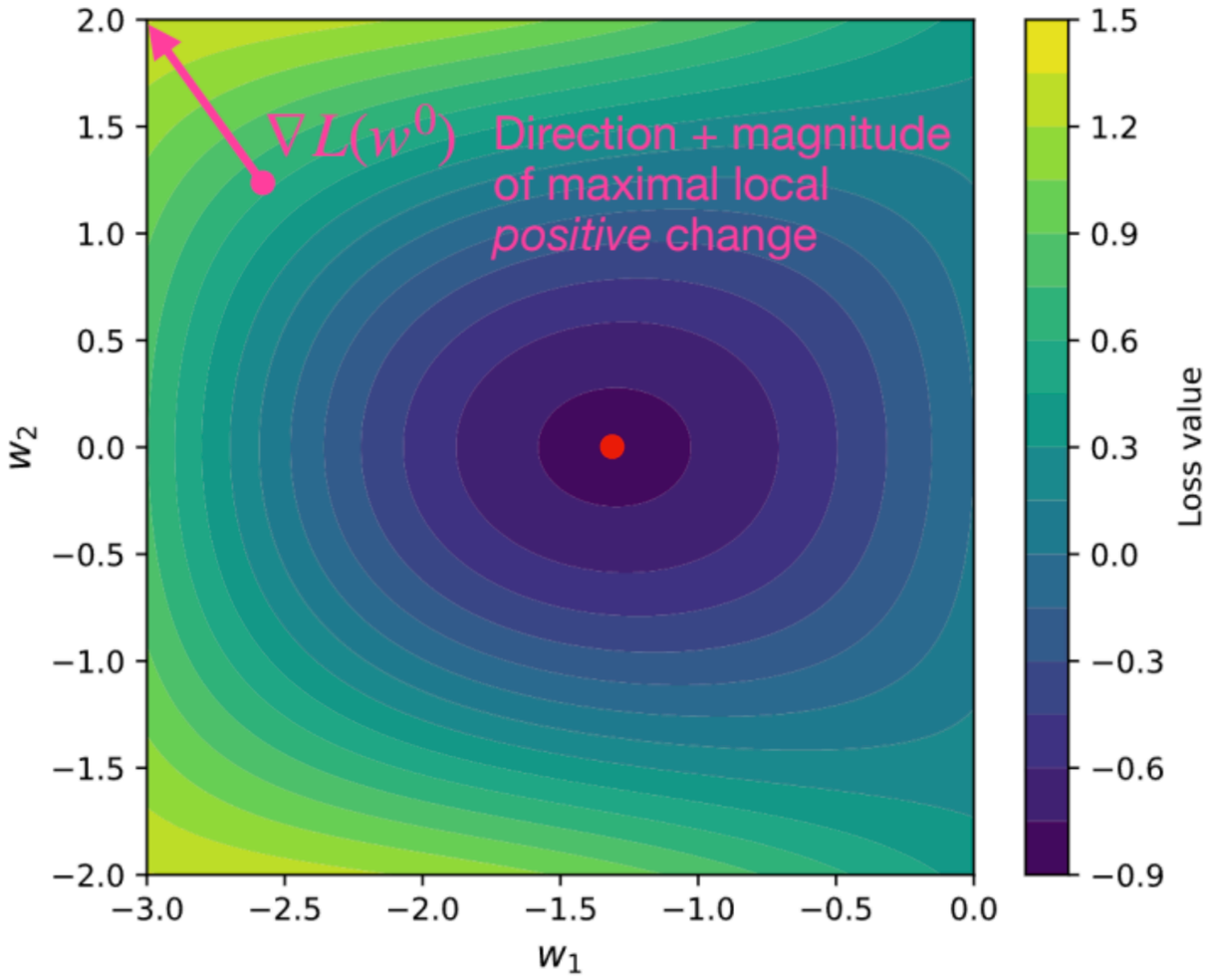
Figure 1: **Empirical LR schedule.** We found that lowering learning rate was helpful for avoiding instabilities.

How do you do it *well* with *finite* step?

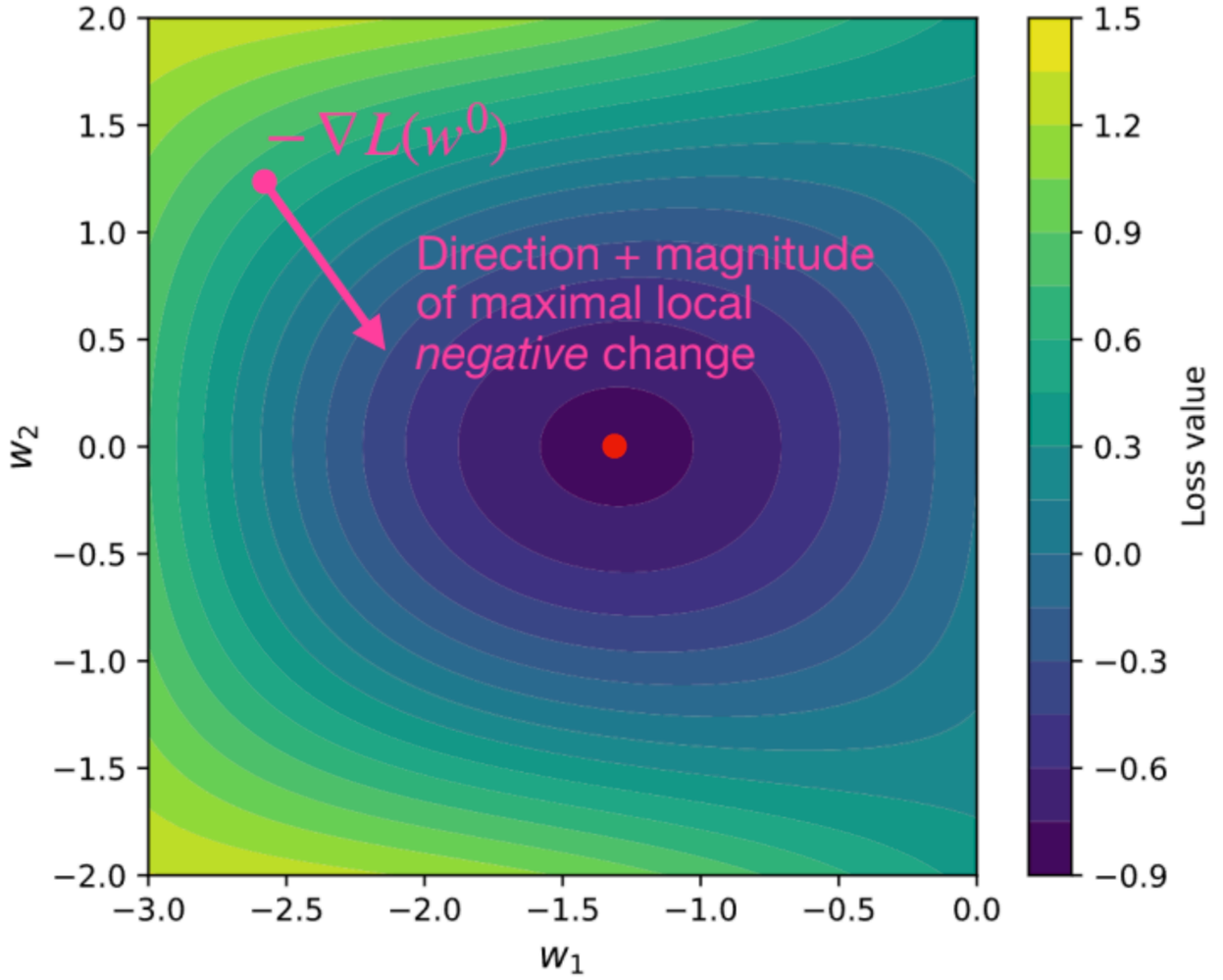
Gradient Descent



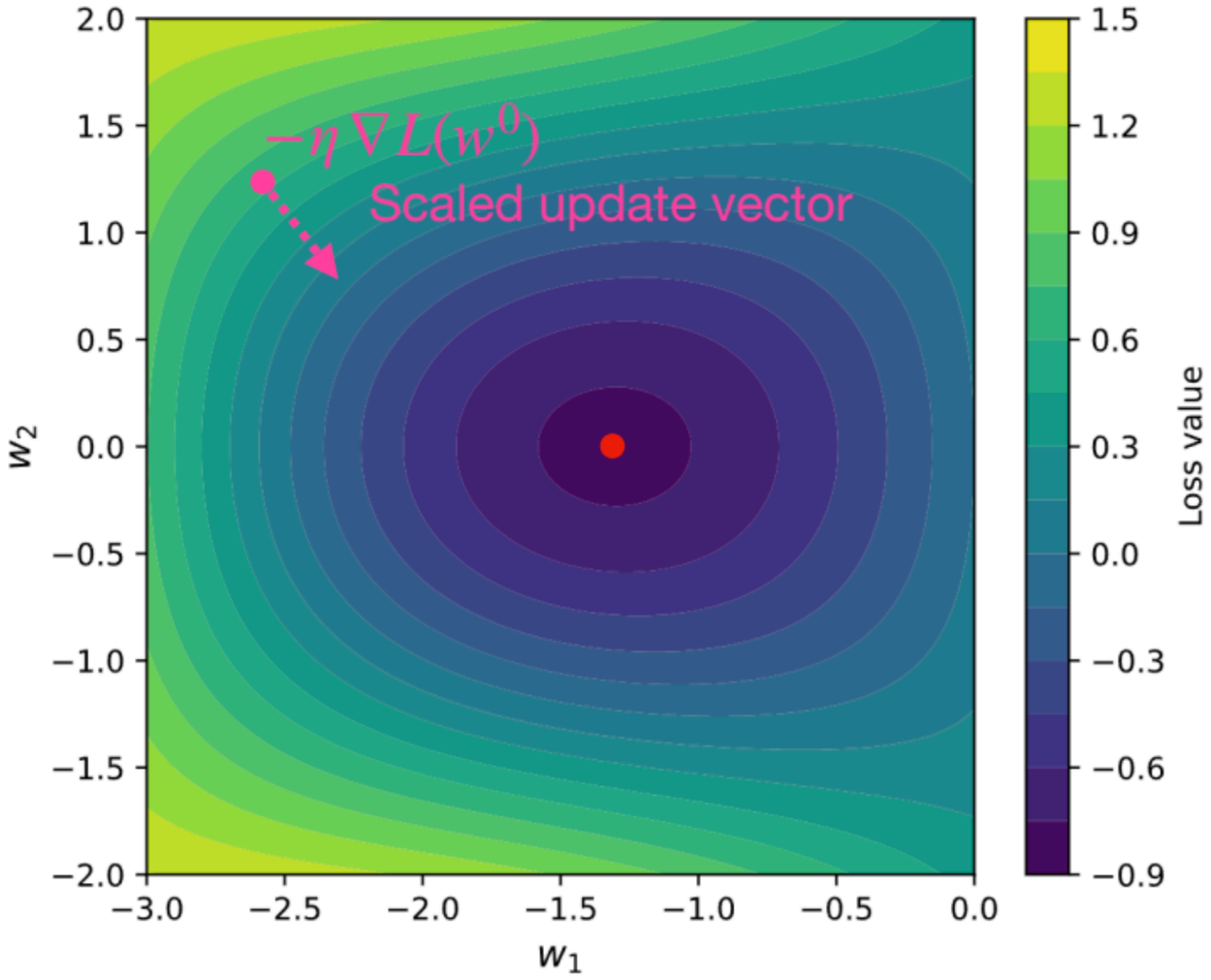
Gradient Descent



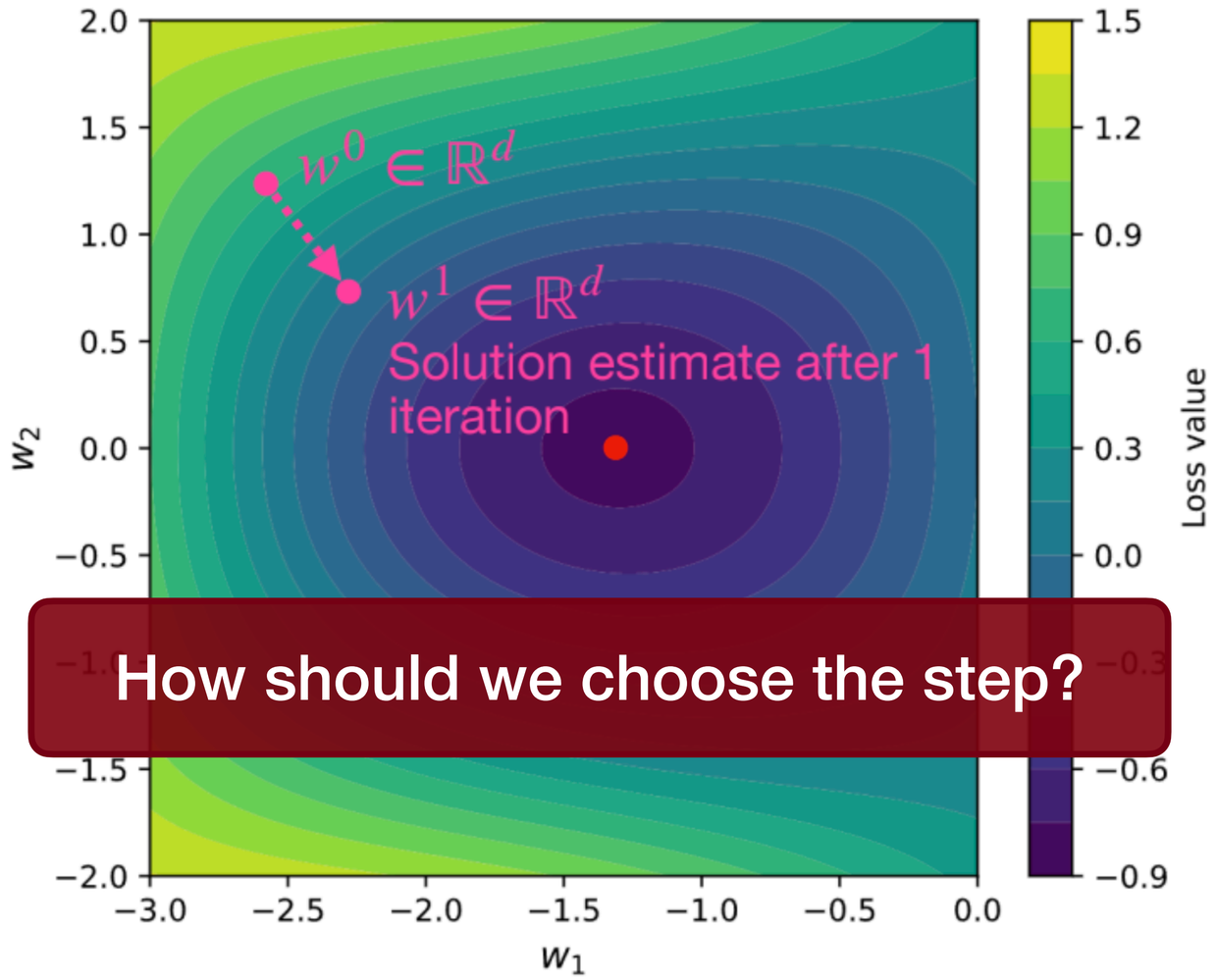
Gradient Descent

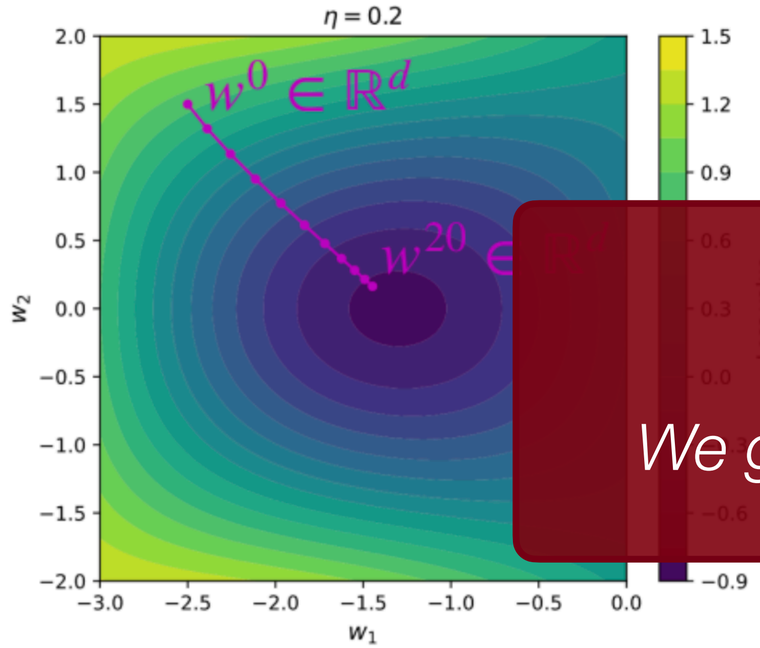


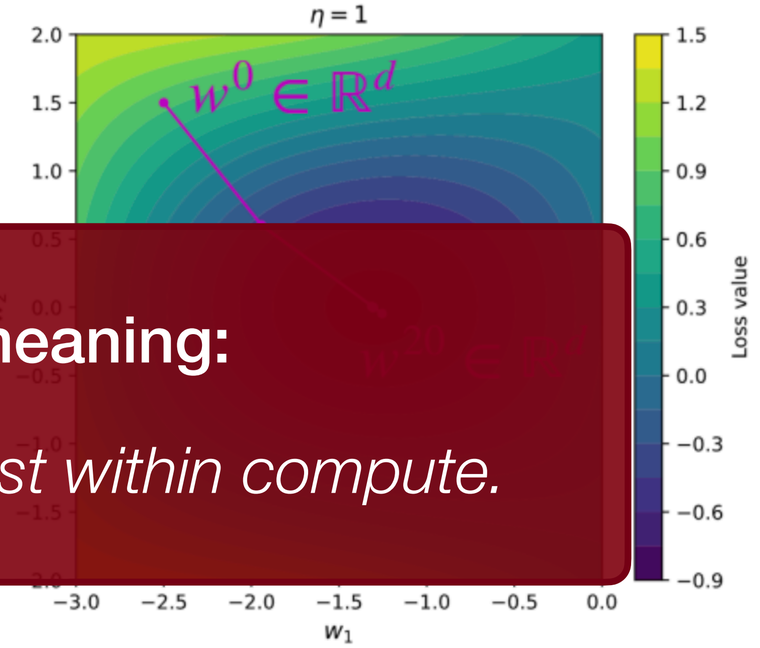
Gradient Descent



Gradient Descent



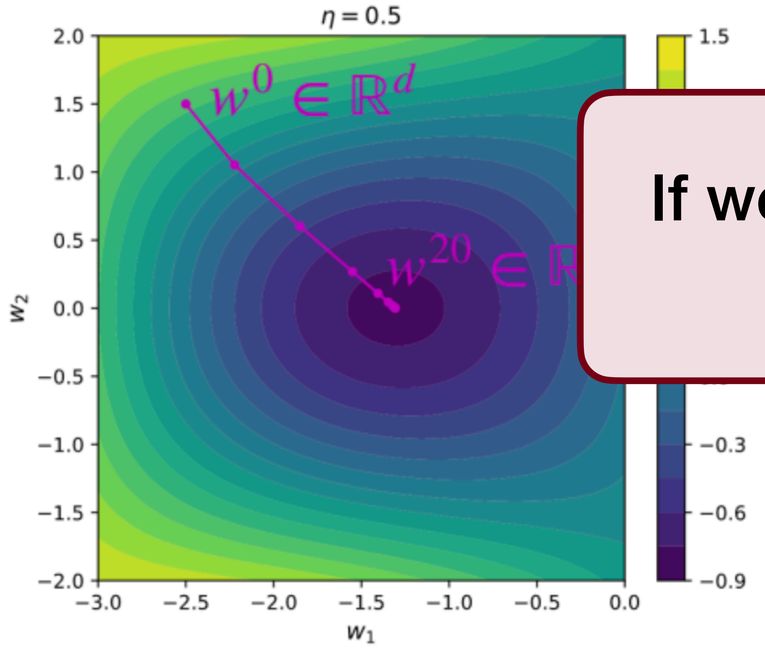


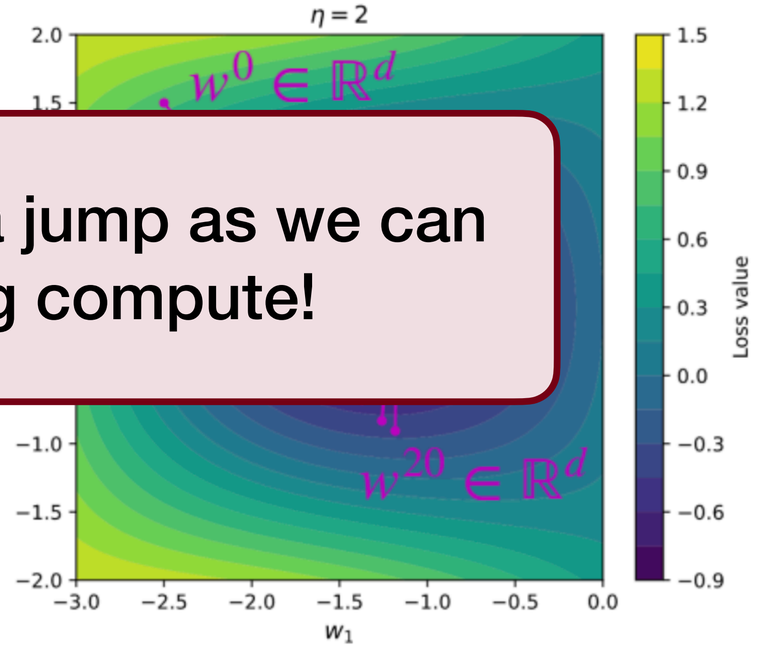


Optimal meaning:

We go down the most within compute.

High η





If we do as big of a jump as we can we are saving compute!

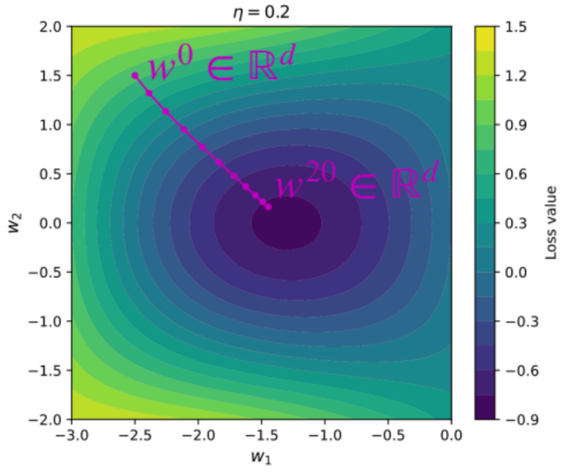
Too high η

INFINITESIMAL *VS* LOCAL

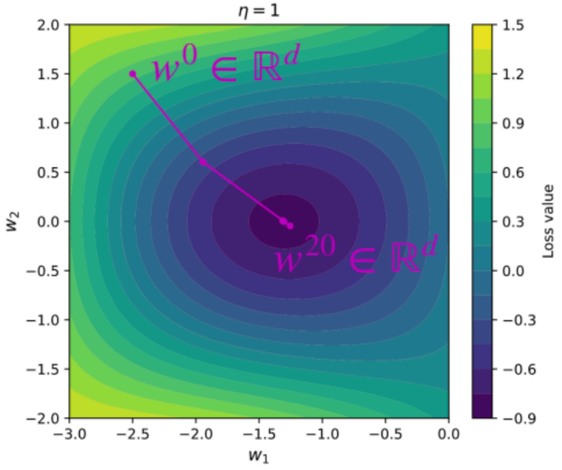
- Gradients are a descriptions of what your function looks like *infinitesimally close.*

Since our compute constraint, we want
the widest neighborhood possible

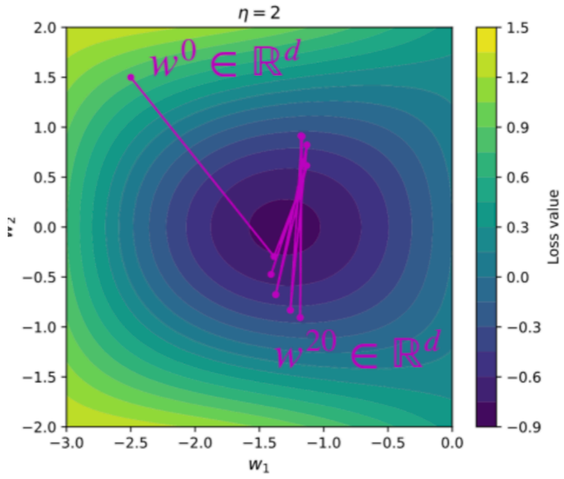
- What can allow us to turn something infinitesimal into something local?



Low η



High η



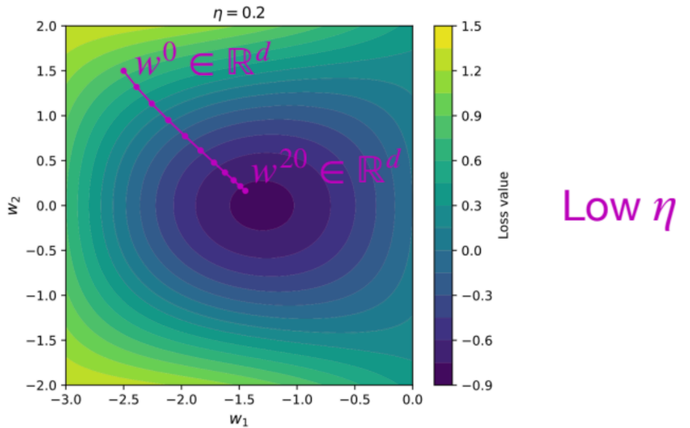
Too high η

INFINITESIMAL *VS* LOCAL

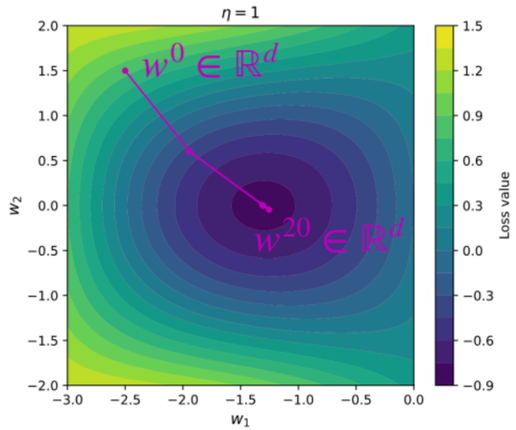
- It's about how much it changes!

Step size is:
How *trustworthy* the *direction* of the gradient is

- Smaller step size = less trust that the direction is accurate.
- Smaller step size = you think the direction is changing quickly!



Low η



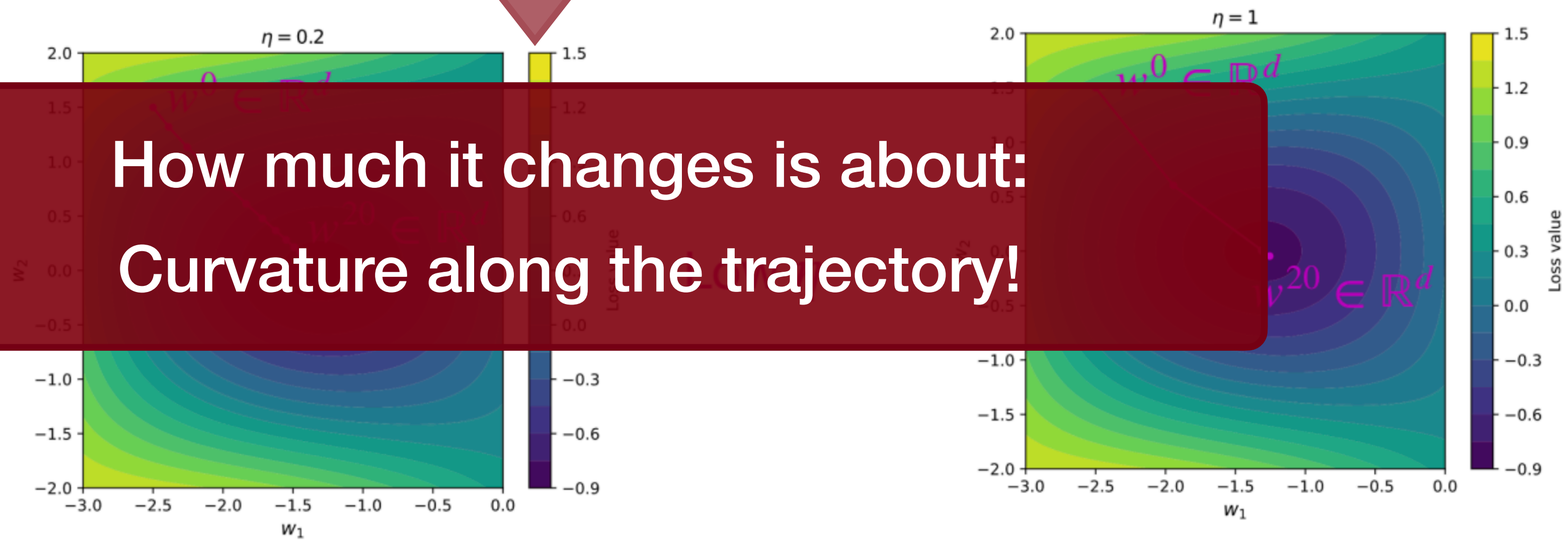
High η

INFINITESIMAL *VS* LOCAL

- It's about how much it changes!

Step size is:
How *trustworthy* the *direction* of the gradient is

How much it changes is about:
Curvature along the trajectory!



High η

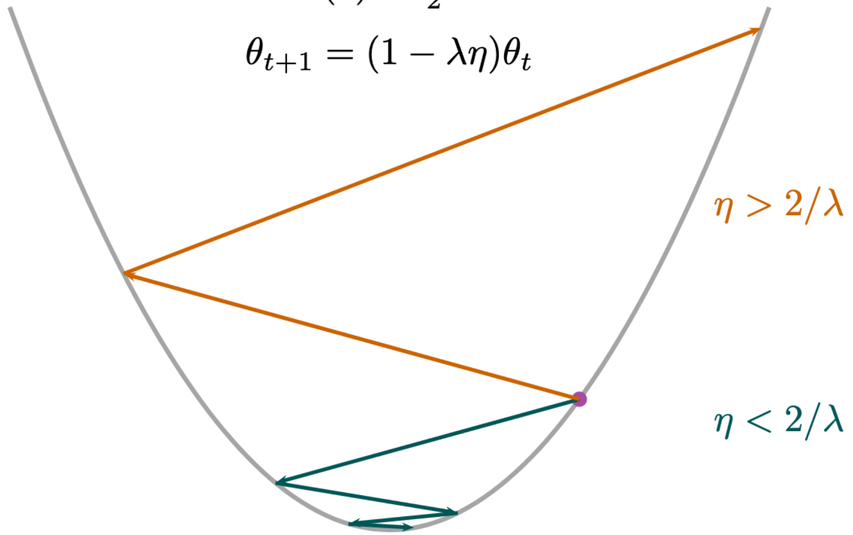
INFINITESIMAL *VS* LOCAL

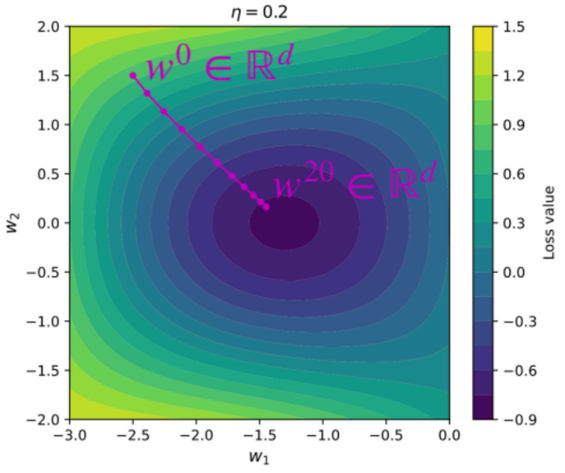
- How much the *direction* of the gradient *changes*.
- How big it needs to be before it *overshoots*.

2^{nd} **Derivative** *(in the neighborhood)*

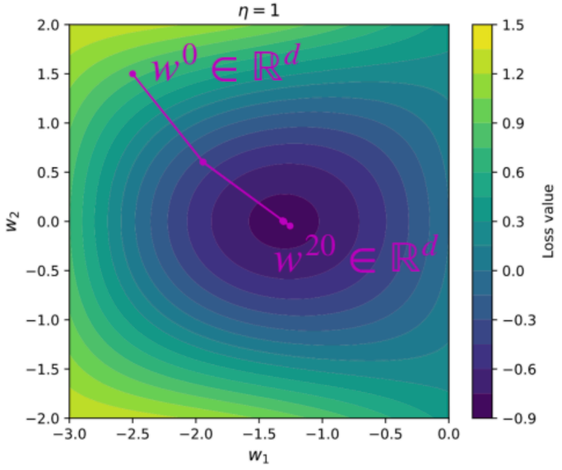
$$L(\theta) = \frac{\lambda}{2}\theta^2$$

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

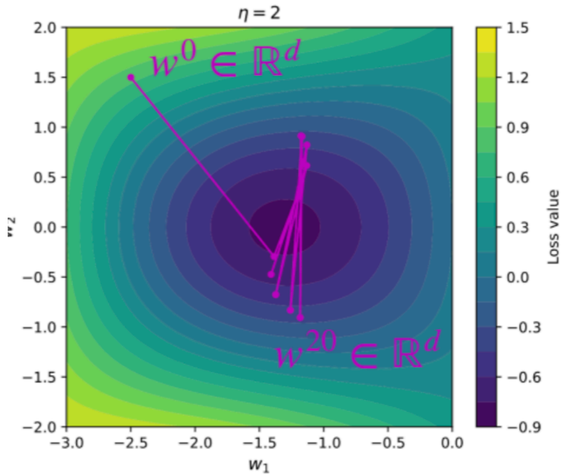




Low η



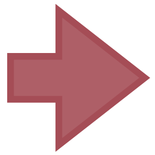
High η



Too high η

Message 1

Designing Gradient Based Optimizers



How to go:

from *infinitesimal* **to** *local*

(to global)

On Gradient Based Methods

What is a gradient-based method?

$$\theta_{t+1} = \theta_t + \text{function}\{\nabla_\theta L(z, \theta),\ z \in B\}$$

How can we choose a better one?

1) Changing step size:
Use local info optimally.

2) Using history:
Momentum, Adaptive

CHANGING STEP SIZE

1) Changing step size:
Use local info optimally.

Questions:

- Can we tweak the optimizer to use in the best possible way the local information?
- We can frame this question as:

We want to find the optimal step size (possibly matrix) $\eta : \theta_t \mapsto \mathbb{R}^{d \times d}$:

$\theta_0 =$ initialization, $\quad \theta_{t+1} = \theta_t - \eta_t \cdot \nabla L(\theta_t) \quad$ is "optimal".

What is GD

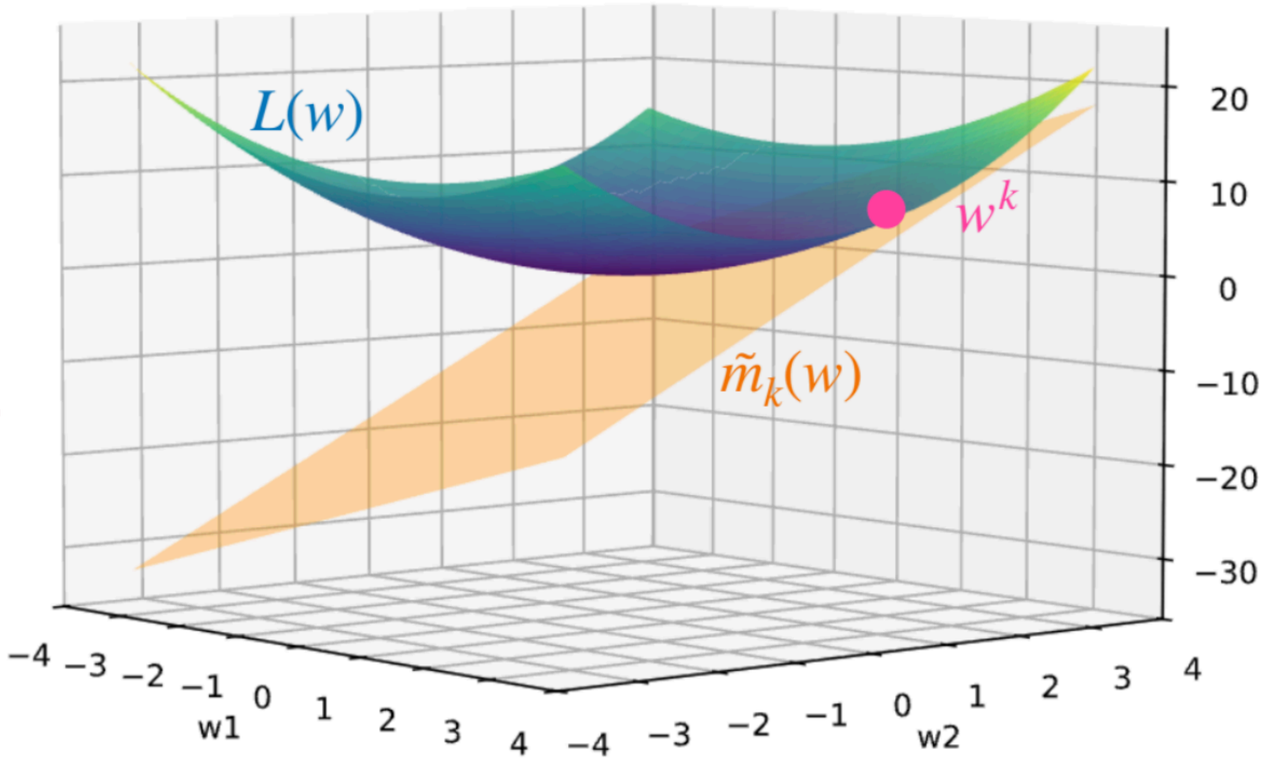
- Locally around θ_t we have the following approx:

$$\tilde{L}(\vartheta) \;:=\; L(\theta_t) + \nabla L(\theta_t)^\top\,(\vartheta - \theta_t).$$

- What is the min in ϑ of $\tilde{L}(\vartheta)$?

Minus infinity.

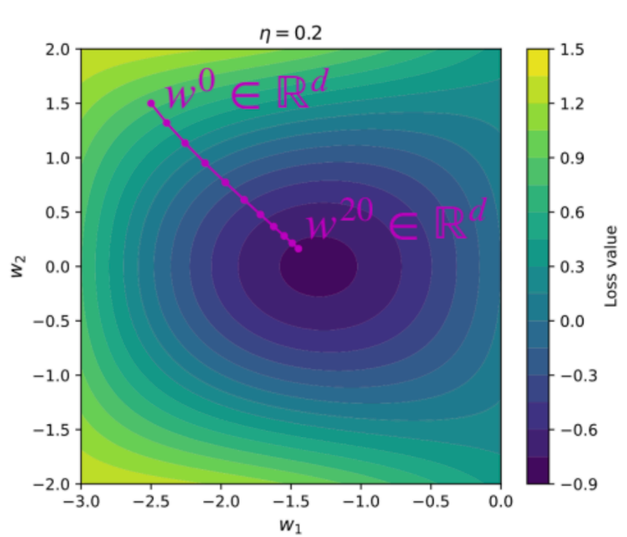
This is not an accurate approximation.



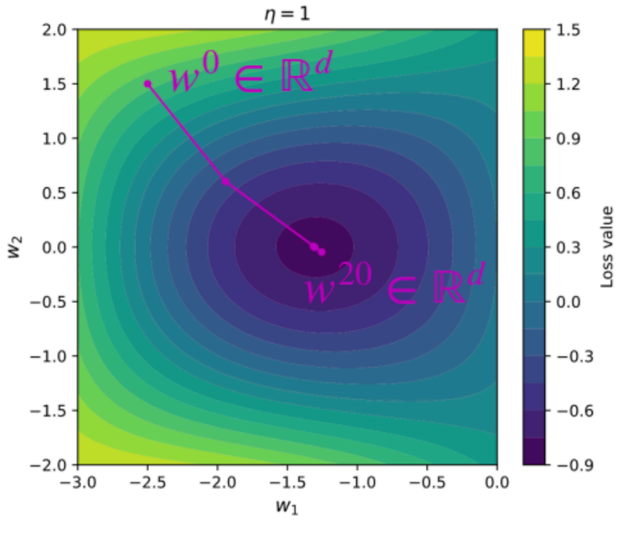
INFINITESIMAL *VS* LOCAL

- It's about how much it changes!

Step size is:
How *trustworthy* the *direction* of the gradient is



Low η



High η

GD *IS A* PROXIMAL METHOD

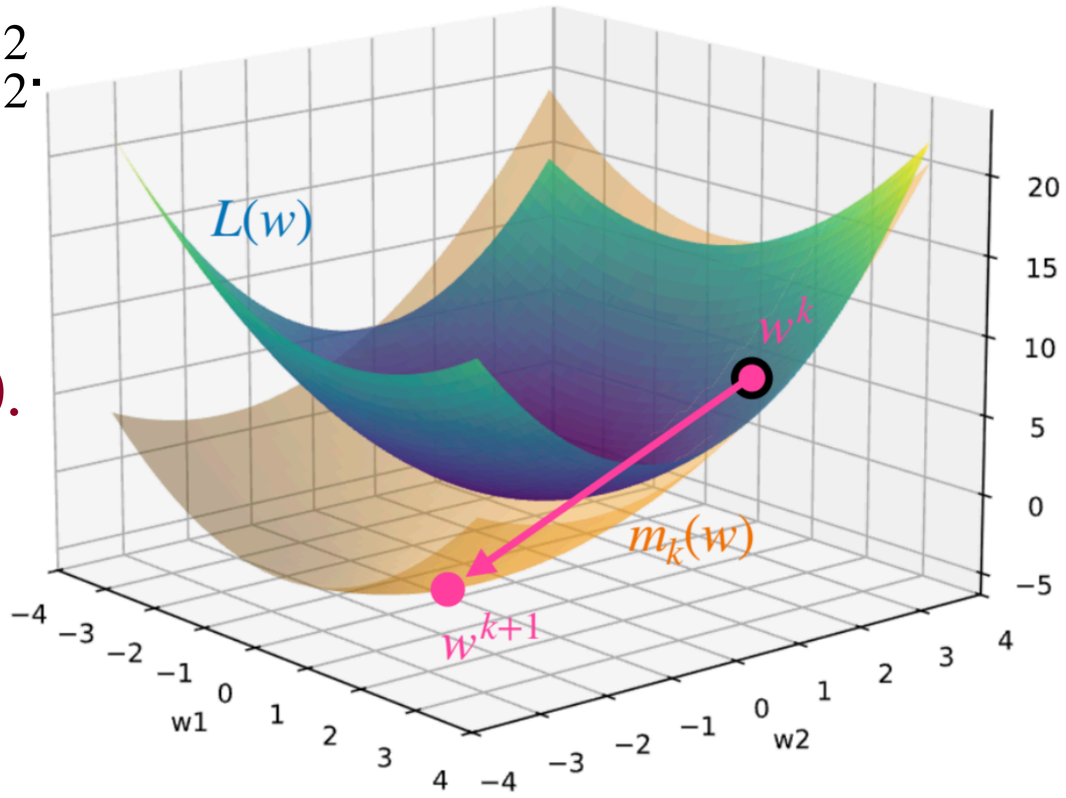
- Let say we trust it "$1/\lambda$", we want to stay closer:

$$\tilde{L}(\vartheta) \;:=\; L(\theta_t) \;+\; \nabla L(\theta_t)^\top (\vartheta - \theta_t) \;+\; \lambda \|\vartheta - \theta_t\|_2^2.$$

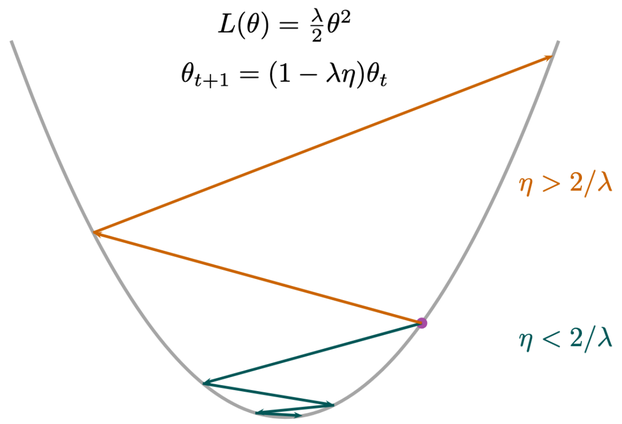
- What is the minimum in ϑ of that?

$$\text{argmin}_\vartheta \left(\tilde{L}(\vartheta) \right) \iff \nabla L(\theta_t) + 2\lambda \cdot (\vartheta - \theta_t) = 0.$$

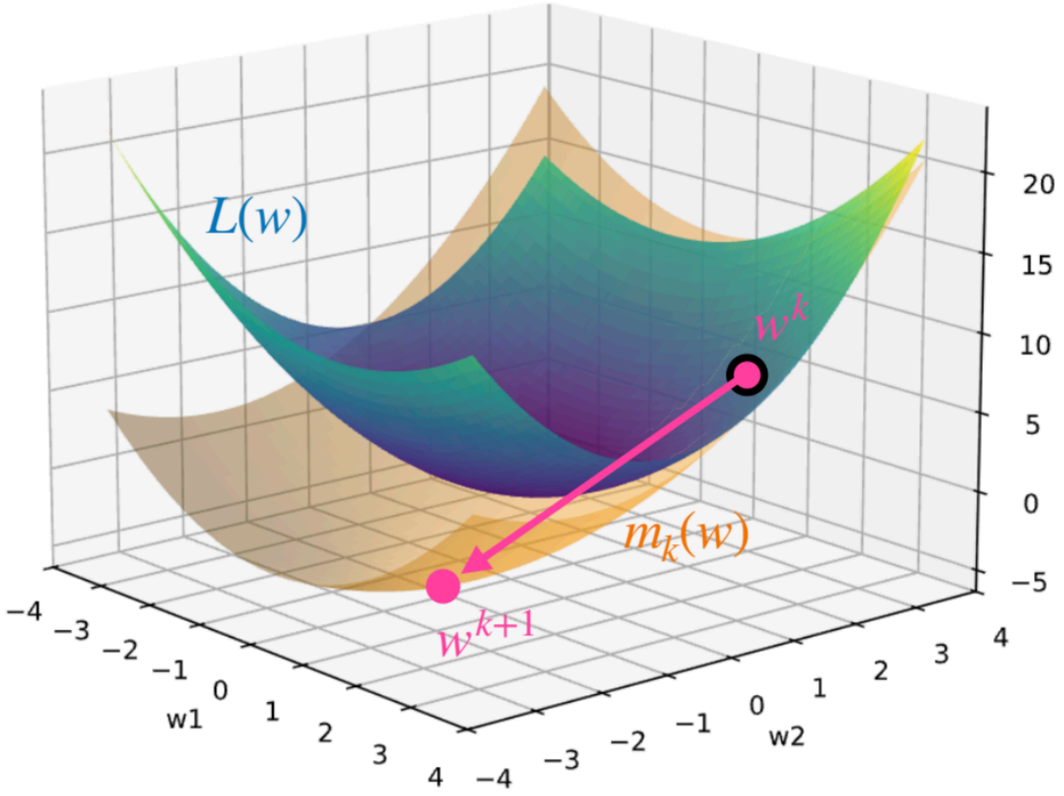
$$\vartheta - \theta_t = -\frac{1}{2\lambda} \nabla L(\theta_t)\,.$$

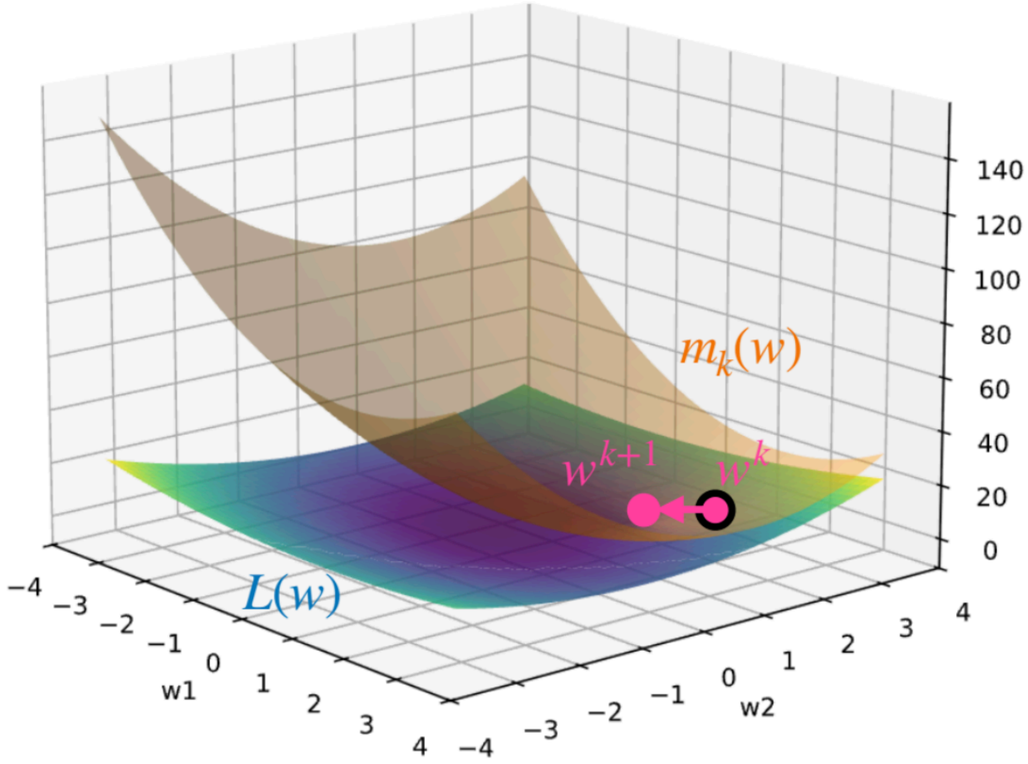


What is GD



Increasing my *trust* $1/\lambda$:



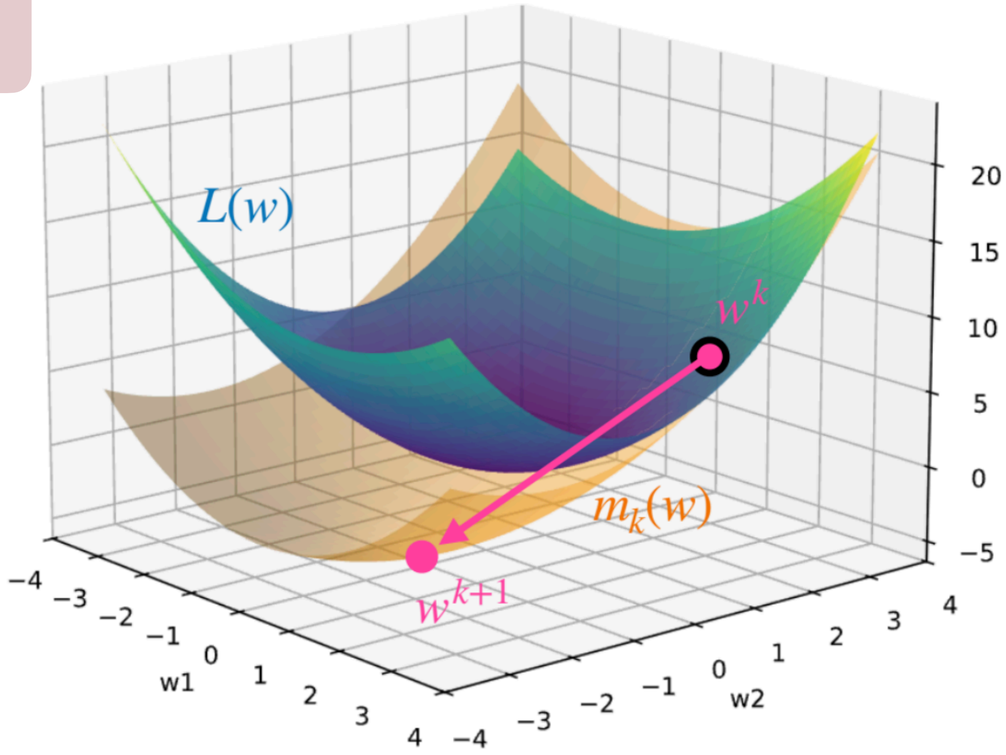


Proximal Methods

- Let say we trust it "$1/\lambda$", we want to stay closer:

$$\tilde{L}(\vartheta) \;:=\; L(\theta_t) \;+\; \nabla L(\theta_t)^\top (\vartheta - \theta_t) \;+\; \lambda \|\vartheta - \theta_t\|_2^2.$$

We can also change the norm and see what goes on!



Other Methods

- Let say we trust it "$1/\lambda$", in Euclidean norm:

$$\tilde{L}(\vartheta) \;:=\; L(\theta_t) \;+\; \nabla L(\theta_t)^\top\,(\vartheta-\theta_t) \;+\; \lambda\|\vartheta-\theta_t\|_2^2.$$

- We can also trust it "$1/\lambda$" is a different L^p-norm:

$$\tilde{L}(\vartheta) \;:=\; L(\theta_t) \;+\; \nabla L(\theta_t)^\top\,(\vartheta-\theta_t) \;+\; \lambda\|\vartheta-\theta_t\|_p^p.$$

- Or in a matrix-P norm (which leads to preconditioning):

$$\tilde{L}(\vartheta) \;:=\; L(\theta_t) \;+\; \nabla L(\theta_t)^\top\,(\vartheta-\theta_t) \;+\; \lambda\|\vartheta-\theta_t\|_P^2, \quad P \in Symm(d\times d).$$

Newton Method

- Locally around θ_t we also have the following Taylor approximation:

$$L(\vartheta) \approx L(\theta_t) + \nabla L(\theta_t)^\top (\vartheta - \theta_t) + \frac{1}{2}(\vartheta - \theta_t)^\top \nabla^2 L(\theta_t) (\vartheta - \theta_t).$$

- After one step of Newton method (the optimal GD?) this can be rewritten as:

$$L(\vartheta) \approx L(\theta_t) + \nabla L(\theta_t)^\top (\vartheta - \theta_t) + \frac{1}{2}\|\vartheta - \theta_t\|^2_{\nabla^2 L(\theta_t)}.$$

- It's saying that you take a matrix step size, and the optimal is the one of the Hessian matrix!

Other Methods

- Let say we trust it "$1/\lambda$", in Euclidean norm:

$$\tilde{L}(\vartheta) \;:=\; L(\theta_t) \;+\; \nabla L(\theta_t)^\top (\vartheta - \theta_t) \;+\; B\|\vartheta - \theta_t\|_2^2.$$

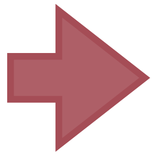
- Another norm we can pick is:

$$\tilde{L}(\vartheta) \;:=\; L(\theta_t) \;+\; \nabla L(\theta_t)^\top (\vartheta - \theta_t) \;+\; B\|\vartheta - \theta_t\|_\infty^2.$$

$$\vartheta - \theta_t = -\frac{\|\nabla L\|_1}{2B}\text{sign}\Big(\nabla L(\theta_t)\Big).$$

Message 1

Designing Gradient Based Optimizers



How to go:

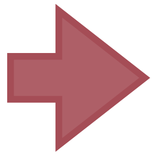
from *infinitesimal* **to** *local*

(to global)

Message 2

Designing Gradient

Based Optimizers



- Step size = Norm + Trust.
- About second derivative.

ADAM

Dissecting Adam: The Sign, Magnitude and Variance of Stochastic Gradients

Lukas Balles [1] Philipp Hennig [1]

2017

Abstract

The ADAM optimizer is exceedingly popular in the deep learning community. Often it works very well, sometimes it doesn't. Why? We interpret ADAM as a combination of two aspects: for each weight, the update direction is determined by the *sign* of stochastic gradients, whereas the update magnitude is determined by an estimate of their *relative variance*. We disentangle these two aspects and analyze them in isolation, gaining insight into the mechanisms underlying ADAM. This analysis also extends recent results on adverse effects of ADAM on generalization, isolating the sign aspect as the problematic one. Transferring the variance adaptation to SGD gives rise to a novel method, completing the practitioner's toolbox for problems where ADAM fails.

NOISE IS NOT THE MAIN FACTOR BEHIND THE GAP BETWEEN SGD AND ADAM ON TRANSFORMERS, BUT SIGN DESCENT MIGHT BE

2023

Frederik Kunstner, Jacques Chen, J. Wilder Lavington & Mark Schmidt[†]
University of British Columbia, Canada CIFAR AI Chair (Amii)[†]
{kunstner,jola2372,schmidtm}@cs.ubc.ca
jacquesc@students.cs.ubc.ca

In Search of Adam's Secret Sauce

Antonio Orvieto *
ELLIS Institute Tübingen, MPI-IS
Tübingen AI Center, Germany

Robert Gower
Flatiron Institute
New York, US

Gradient-Based Methods

- **Gradient-Based Methods**
  - Choosing Step Size
  - **In practice**
- Initialization
- Scaling up

Slides partly taken by Orvieto (2025)

METHOD 2: HISTORY

What is a gradient-based method?

$$\theta_{t+1} = \theta_t + \text{function}\{\nabla_\theta L(z, \theta),\ z \in \mathcal{D}\}$$

2) Using history:

Momentum, Adaptive

$$\theta_{t+1} = \theta_t + \text{function}\{\nabla_\theta L(B_t, \theta_t),\ \text{previous } t\}.$$

Using the Previous Directions

- Momentum — Heavy ball (Polyak, 1964):

$$v_{t+1} = \beta_1 v_t - \eta \nabla L(\theta_t), \qquad \theta_{t+1} = \theta_t + v_{t+1}\,.$$

- Nesterov Acceleration (Nesterov, 1983):

$$v_{t+1} = \beta_1 v_t - \eta \nabla L(\theta_t + \beta_1 v_t), \qquad \theta_{t+1} = \theta_t + v_{t+1}$$

Adams Family

Also the history of sizes

- Adam = momentum + per-coordinate adaptive steps (Kingma and Ba, 2014):

Let $g_t[i] = \partial L(\theta_t)/\partial\theta[i]$:

$$\begin{cases} m_t[i] = \beta_1, m_{t-1}[i] + (1-\beta_1) g_t[i], \\ v_t[i] = \beta_2, v_{t-1}[i] + (1-\beta_2), g_t[i]^2, \\ \hat{m}_t[i] = \dfrac{m_t[i]}{1-\beta_1^t}, \qquad \hat{v}_t[i] = \dfrac{v_t[i]}{1-\beta_2^t}, \\ \theta_{t+1}[i] = \theta_t[i] - \eta, \dfrac{\hat{m}_t[i]}{\sqrt{\hat{v}_t[i]} + \epsilon}. \end{cases}$$

- AdamW (Loshchilov & Hutter, 2019):

Adam on $(1 - \eta\lambda)\ \theta_t$.

MUON

- Muon (Jordan 2024) and:

SGD Momentum, only $U, V^\top$

2024

Old Optimizer, New Norm: An Anthology

Jeremy Bernstein JBERNSTEIN@MIT.EDU
Laker Newhouse LAKERN@MIT.EDU
MIT CSAIL, United States

Abstract

Deep learning optimizers are often motivated through a mix of convex and approximate second-order theory. We select three such methods—Adam, Shampoo and Prodigy—and argue that each method can instead be understood as a squarely first-order method without convexity assumptions. In fact, after switching off exponential moving averages, each method is equivalent to *steepest descent* under a particular *norm*. By generalizing this observation, we chart a new design space for training algorithms. Different operator norms should be assigned to different tensors based on the role that the tensor plays within the network. For example, while linear and embedding layers may have the same weight space of $\mathbb{R}^{m\times n}$, these layers play different roles and should be assigned different norms. We hope that this idea of carefully metrizing the neural architecture might lead to more stable, scalable and indeed faster training.

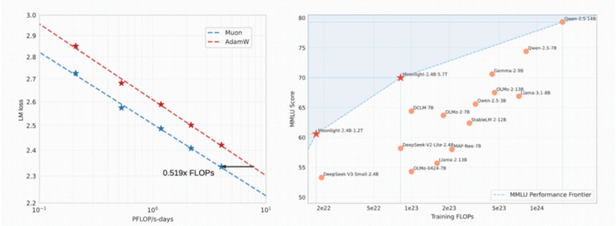
2025

MUON IS SCALABLE FOR LLM TRAINING

TECHNICAL REPORT

Jingyuan Liu[1] **Jianlin Su**[1] **Xingcheng Yao**[2] **Zhejun Jiang**[1] **Guokun Lai**[1] **Yulun Du**[1] **Yidao Qin**[1] **Weixin Xu**[1] **Enzhe Lu**[1] **Junjie Yan**[1] **Yanru Chen**[1] **Huabin Zheng**[1] **Yibo Liu**[1] **Shaowei Liu**[1] **Bohong Yin**[1] **Weiran He**[1] **Han Zhu**[1] **Yuzhi Wang**[1] **Jianzhou Wang**[1] **Mengnan Dong**[1] **Zheng Zhang**[1] **Yongsheng Kang**[1] **Hao Zhang**[1] **Xinran Xu**[1] **Yutao Zhang**[1] **Yuxin Wu**[1] **Xinyu Zhou**[1] * **Zhilin Yang**[1]

[1] Moonshot AI [2] UCLA

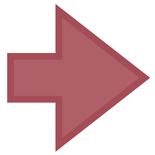


(a) (b)

Figure 1: Scaling up with Muon. **(a)** Scaling law experiments comparing Muon and Adam. Muon is $\sim 2\times$ more computational efficient than Adam with compute optimal training. **(b)** The MMLU performance of our Moonlight model optimized with Muon and other comparable models. Moonlight advances the Pareto frontier of performance vs training FLOPs.

Message 1

Designing Gradient Based Optimizers



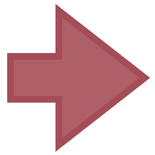
How to go:

from *infinitesimal* **to** *local*

(to global)

Message 3

Designing Gradient Based Optimizers



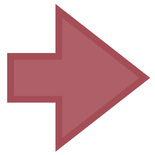
It’s about:

- **How you choose the step size**
- **What you make of the history**

Message 2

Designing Gradient

Based Optimizers



- Step size = Norm + Trust.
- About second derivative.

Initialization

- Gradient-Based Methods
  - Choosing Step Size
  - In practice
- **Initialization**
- Scaling up

This Class

Infinitesimally, to go down *optimally* we follow the gradient:

$$\begin{cases} \theta_0 = \text{initialization} \\ \dfrac{d\theta_t}{dt} = -\nabla L(\theta_t) \end{cases}$$

How to pick initialization?

What is the optimal discrete step size version of this?

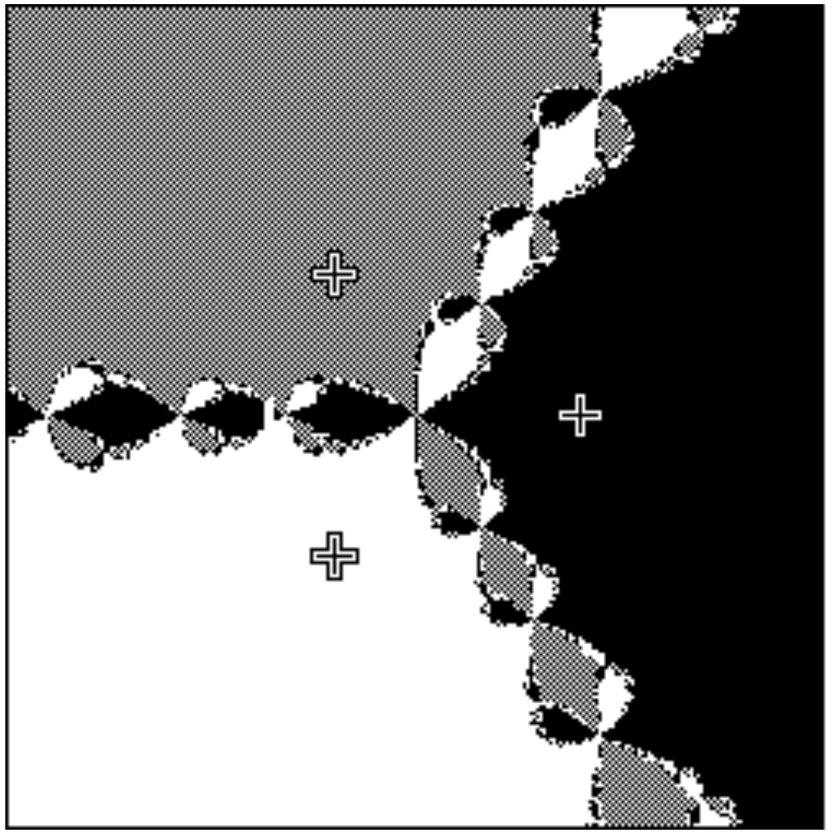
Classical PoV

Classically, initialization is about:

$$\begin{cases} \theta_0 = \text{initialization} \\ \dfrac{d\theta_t}{dt} = -\nabla L(\theta_t) \end{cases}$$

In what basin of attraction you are:

It is about:
which minimum



Basins for Newton on $L(\theta) = \theta^3 - 1$

Exploding Vanishing

- Imagine you are taking the product of a list of numbers (or matrices):

$$f_\theta(x) = \theta_L \cdot \theta_{L+1} \cdot \ldots \cdot \theta_2 \cdot \theta_1 \cdot x$$

$$\frac{\partial \mathscr{L}}{\partial h^{(\ell)}} = \Big(\prod_{k=\ell+1}^{L} J^{(k)} \Big) \frac{\partial \mathscr{L}}{\partial h^{(L)}}$$

- Imagine now the depth $L \gg 1$ is big:
  - If the $\|\theta_\ell\|$ are bigger than $1 + \mathcal{O}(1/L)$ this product explodes!
  - If the $\|\theta_\ell\|$ are smaller than $1 - \mathcal{O}(1/L)$ this product vanishes!

Exploding Vanishing

- Imagine you are taking the product of a list of numbers (or matrices):

$$f_\theta(x) = \theta_L \cdot \theta_{L+1} \cdot \ldots \cdot \theta_2 \cdot \theta_1 \cdot x$$

$$\frac{\partial \ell}{\partial \theta_L} \quad \frac{\partial \ell}{\partial \theta_1}$$

For this reason:

Historically it was difficult to train deep models

- Imagine now the de

  - If the $\|\theta_\ell\|$ are bigger than $1 + \mathcal{O}(1/L)$ this product explodes!

  - If the $\|\theta_\ell\|$ are smaller than $1 - \mathcal{O}(1/L)$ this product vanishes!

Initialization = Trainability

- Imagine now the depth $L \gg 1$ is big:
  - If the $\|\theta_\ell\|$ are bigger than $1 + \mathcal{O}(1/L)$ this product explodes!
  - If the $\|\theta_\ell\|$ are smaller than $1 - \mathcal{O}(1/L)$ this product vanishes!
- All the sublevel sets are connected ($width > 8d$) *(Nguyen, 2019)*

Maybe we can think of initialization as:

About stability/trainability, not location.

How do we Initialize then

- Fixing activation/gradient scale per layer:

Variance-preserving initialization.

- Xavier/Glorot *(tanh)*:

$$\mathrm{Var}\big(W_\ell[i,j]\big) = \frac{2}{\textbf{fan\_in} + \textbf{fan\_out}}.$$

- Kaiming *(ReLU)*:

$$\mathrm{Var}\big(W_\ell[i,j]\big) = \frac{2}{\textbf{fan\_in}}.$$

THROUGHOUT THE TRAINING?

- The issue is:

  We said something about initialization, not through training.

- The solution is:

  Changing the architecture

- Non saturating activations ~~*(tanh)*~~
- Forcing layer normalization!

Message 3

- We want to pick the architecture and the initialization to enhance trainability.
- Not really for purposes of approximation or location of convergence!

Scaling Up

- Gradient-Based Methods
  - Choosing Step Size
  - In practice
- Initialization
- **Scaling up**

Achtung: **No theory**

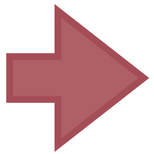
Cool references: https://howtoscalenn.github.io/ and therein

Regime 3

- Enough exact data (all of them!)
- ***Not enough*** compute!

We want a model that:

- (Just) can fit them all.



A training pipeline that allows it within my compute!

BIG MODELS

- We want the models to be big, cause error scales down!

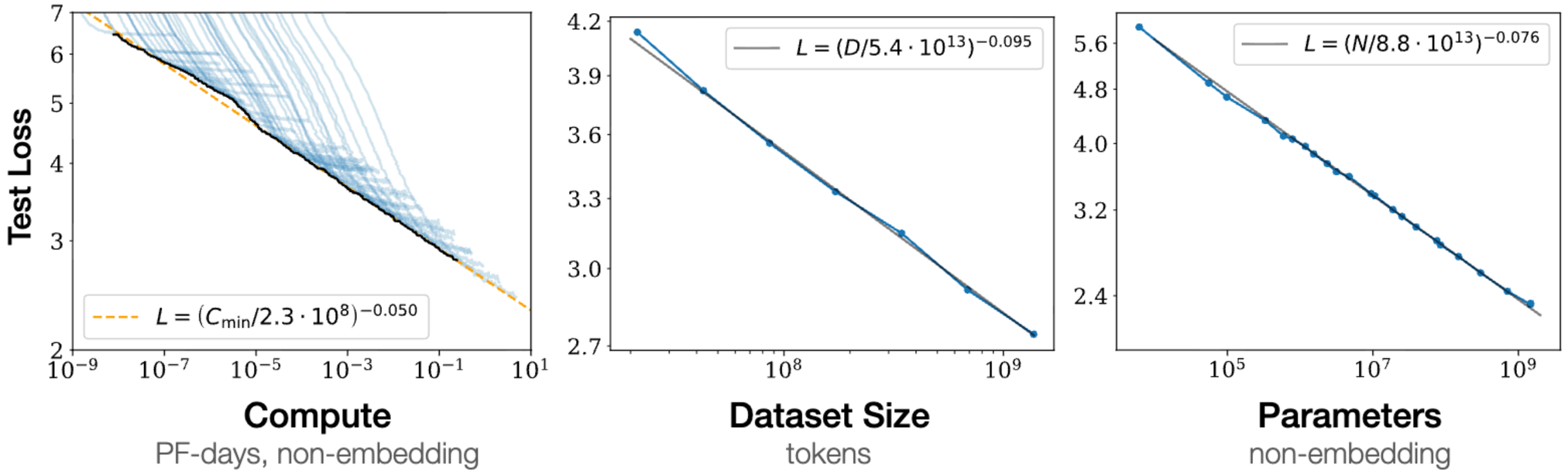


Figure 1 Language modeling performance improves smoothly as we increase the model size, datasetset size, and amount of compute[2] used for training. For optimal performance all three factors must be scaled up in tandem. Empirical performance has a power-law relationship with each individual factor when not bottlenecked by the other two.

Figure 1 from Kaplan et al. (2020).

The Issue with Big Models

- Whenever we train a model we want to find the *right* hyperparameters.
- Grid searching them is a long and painful process:

That we can not afford under compute constraints!

- The strategies are two:

- **Reducing the number of them.**
- **Finding a way to grid search in a more computable way.**

Reducing Hyperparameters

- Designing hyperparameter free optimizers.
- Finding the gold standards.
- Coupling a few of them:

Linear scaling rule.

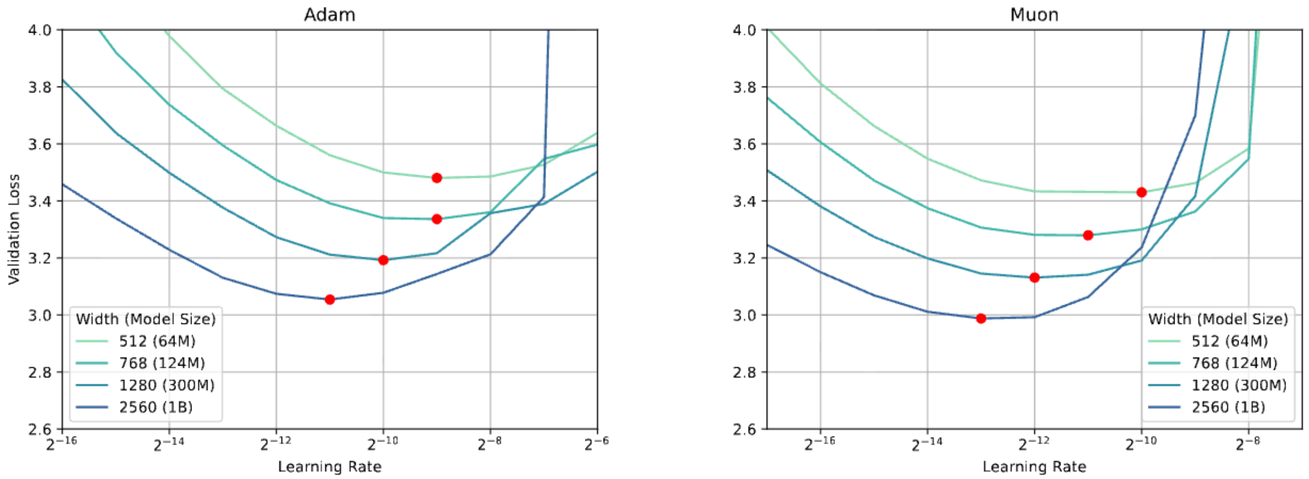
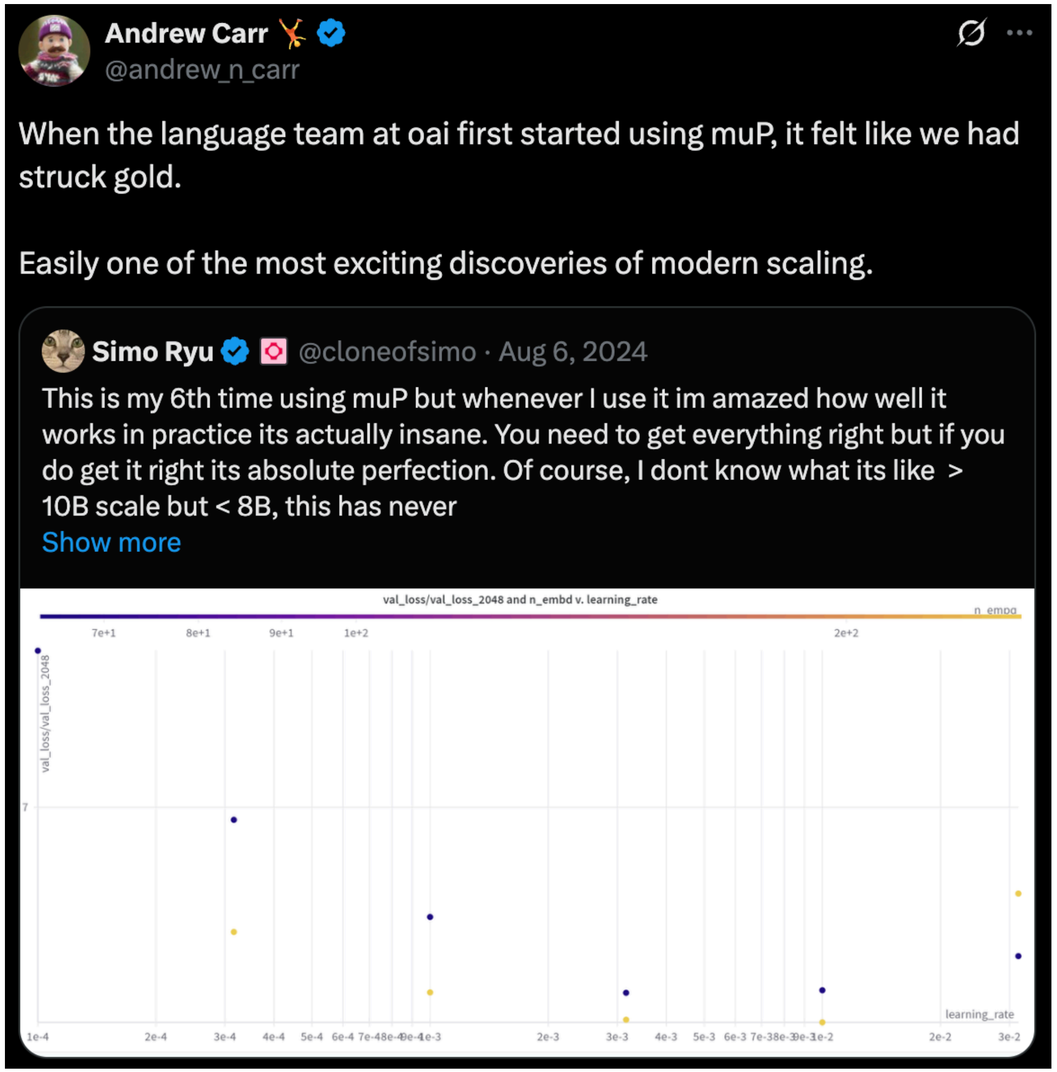


Figure from Pethick et al. (2025)

Searching, then Scaling Up



Source tweet.

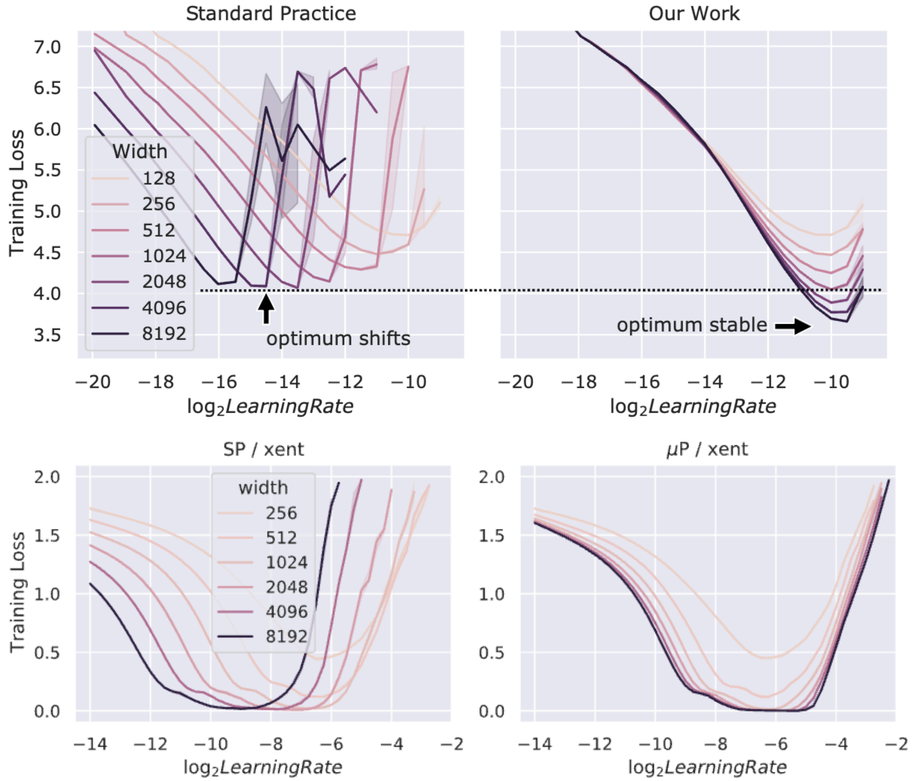


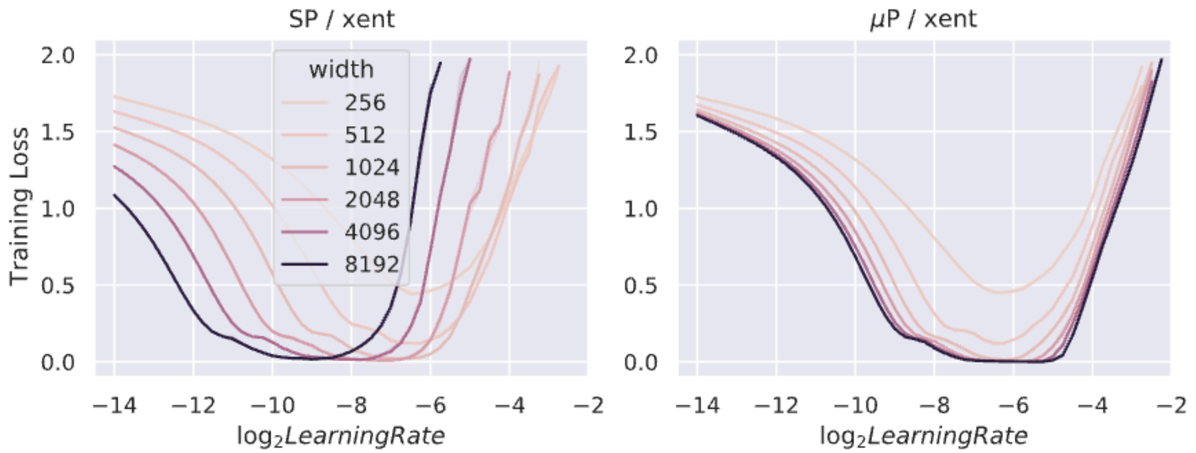
Figure 3: MLP width different hidden sizes trained for 20 epoch on CIFAR-10 using SGD. **Left** uses standard parametrization (SP); **right** uses maximal update parametrization (μP). μP networks exhibit better learning rate stability than their SP counterparts.

Extracts from Yang et al. (2022).

SCALING UP THE WIDTH

Algorithm 1 Tuning a Large Target Model via μTransfer

1: Parametrize target model in Maximal Update Parametrization (μP)
2: Tune a smaller version (in width and/or depth) of target model
3: Copy tuned hyperparameters to target model



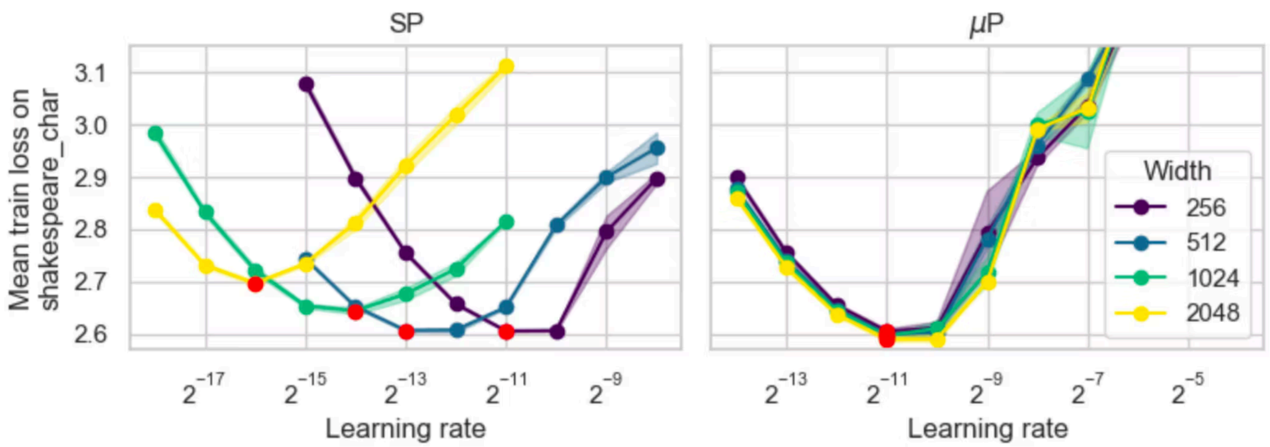


Figure 2 from Cerebras (2024), Figure 1 and tables from Yang et al. (2022).

µP: *NOW WE CAN SCALE*

Law of Large Numbers (LLN) If $x_1, \ldots, x_n, \ldots$ "look like" random independent samples of a random variable X, then

$$\frac{1}{n}\sum_{i=1}^{n} x_i \to \mathbb{E}[X], \quad \text{as } n \to \infty.$$

Central Limit Theorem (CLT) In the same scenario as above,

$$\frac{1}{\sqrt{n}}\sum_{i=1}^{n}(x_i - \mathbb{E}[X]) \to \mathcal{N}(0, \sigma(X)), \quad \text{as } n \to \infty,$$

where $\sigma(X)$ is the standard deviation of the random variable X.

Of course, there are many subtleties one must resolve to make the statements above truly rigorous (e.g., what is the meaning of "look like"?), but as rules of thumb, they typically give the correct prediction.

In particular, here we want to note the following basic intuition regarding the size of a sum of x_i:

$$\text{when } n \text{ is large, } \sum_{i=1}^{n} x_i \text{ has typical size } \begin{cases} \Theta(n) & \text{if } \mathbb{E}[X] \neq 0 \\ \Theta(\sqrt{n}) & \text{otherwise} \end{cases}$$

Desiderata J.1. At any time during training

1. Every (pre)activation vector in a network should have $\Theta(1)$-sized coordinates[32]
2. Neural network output should be $O(1)$.
3. All parameters should be updated as much as possible (in terms of scaling in width) without leading to divergence.

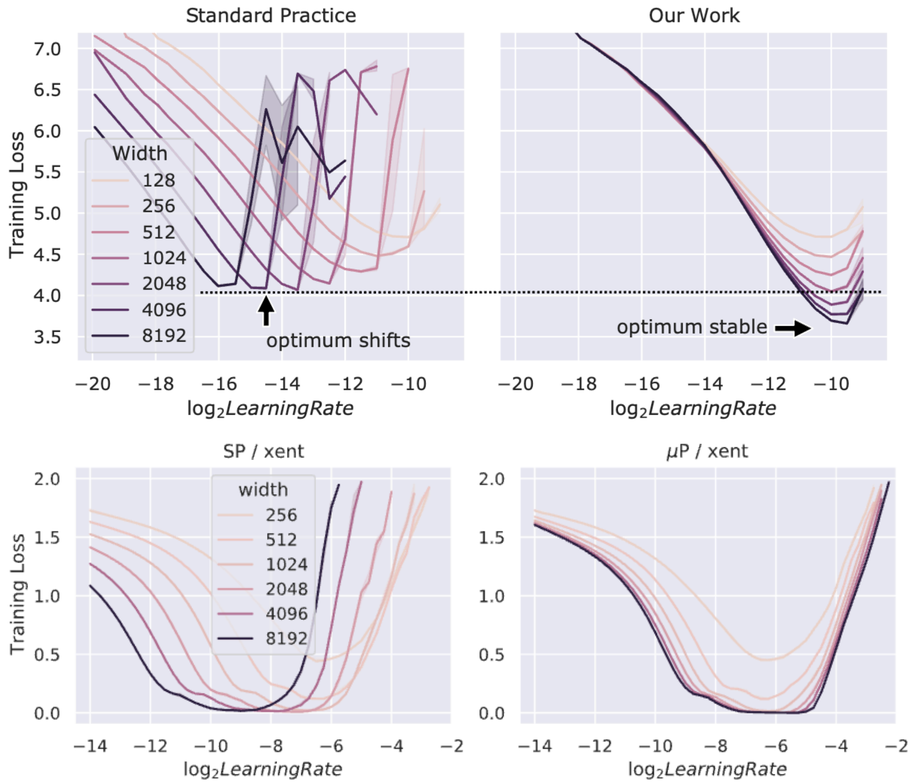


Figure 3: MLP width different hidden sizes trained for 20 epoch on CIFAR-10 using SGD. **Left** uses standard parametrization (SP); **right** uses maximal update parametrization (μP). μP networks exhibit better learning rate stability than their SP counterparts.

Exctracts from Yang et al. (2022).

muP: *NOW WE CAN SCALE*

Desiderata J.1. At any time during training

1. Every (pre)activation vector in a network should have $\Theta(1)$-sized coordinates[32]
2. Neural network output should be $O(1)$.
3. All parameters should be updated as much as possible (in terms of scaling in width) without leading to divergence.

Algorithm 1 Tuning a Large Target Model via μTransfer

1: Parametrize target model in Maximal Update Parametrization (μP)
2: Tune a smaller version (in width and/or depth) of target model
3: Copy tuned hyperparameters to target model

	Input weights & all biases	Output weights	Hidden weights
Init. Var.	$1/\text{fan\_in}$	$1/\text{fan\_in}^2$ $(1/\text{fan\_in})$	$1/\text{fan\_in}$
SGD LR	fan_out (1)	$1/\text{fan\_in}$ (1)	1
Adam LR	1	$1/\text{fan\_in}$ (1)	$1/\text{fan\_in}$ (1)

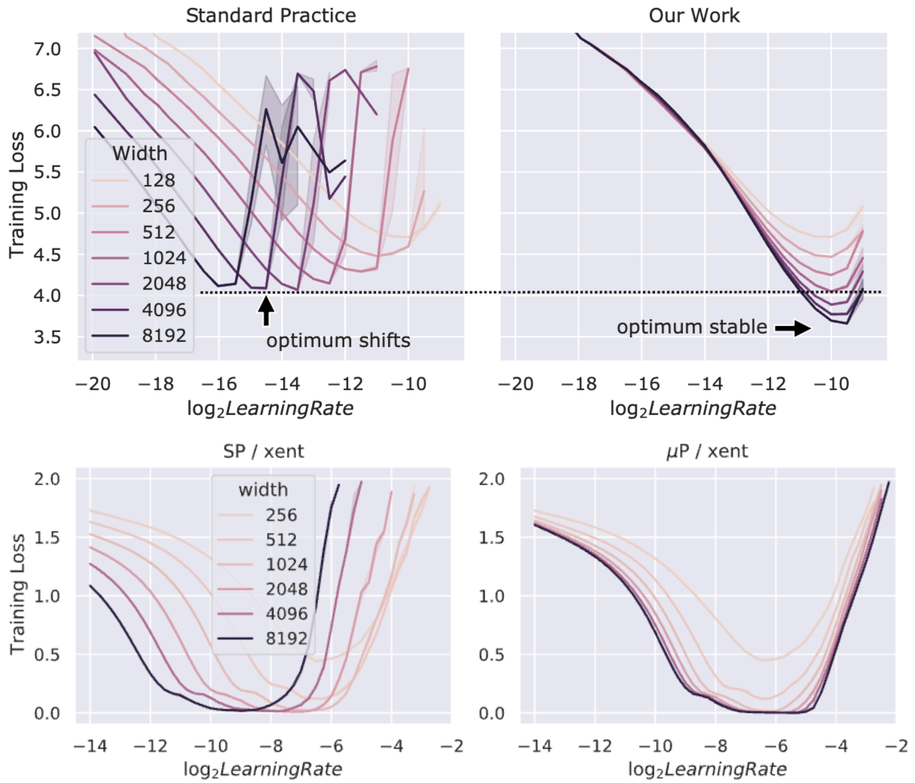


Figure 3: MLP width different hidden sizes trained for 20 epoch on CIFAR-10 using SGD. **Left** uses standard parametrization (SP); **right** uses maximal update parametrization (μP). μP networks exhibit better learning rate stability than their SP counterparts.

Extracts from Yang et al. (2022).

SCALING UP

muP figures out width scaling

- How about *depth*?
- How about *batch size* and *dataset size*?

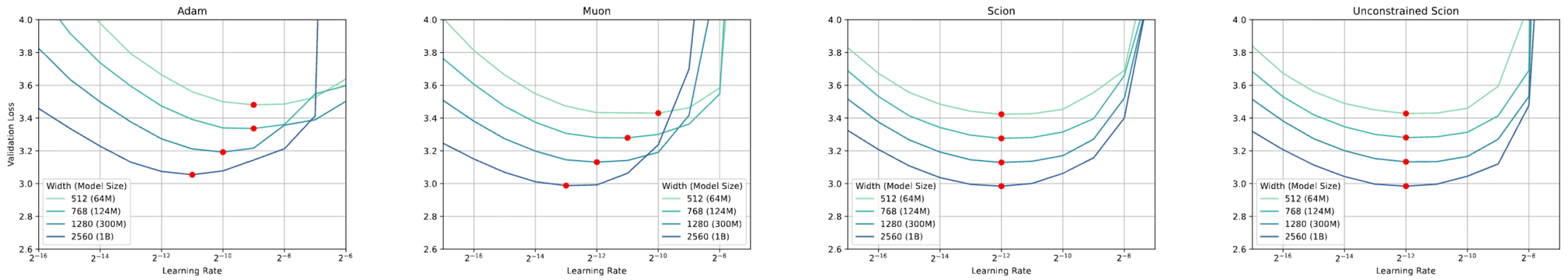


Figure 1. Performance on NanoGPT with between 64M and 1B parameters. The optimal learning rate of SCION is invariant to width.

Figure from Pethick et al. (2025)

Scaling Up

muP figures out width scaling

- How about *depth*?
- How about *batch size* and *dataset size*?

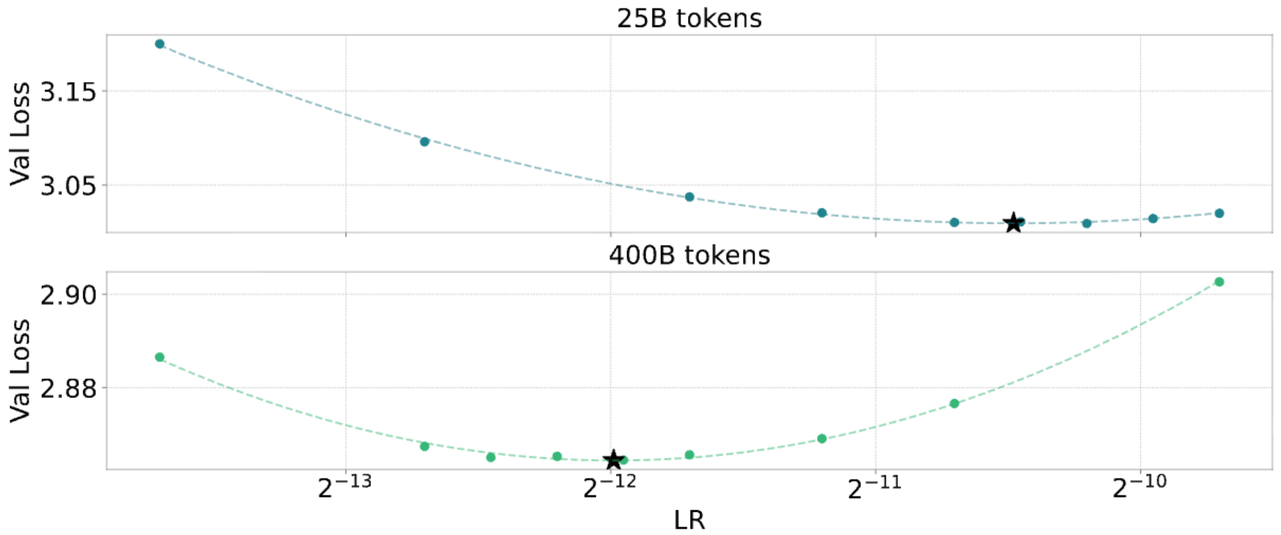


Figure from Bjorck et al. (2024)

Scaling Up

muP figures out width scaling

- How about *depth*?
- How about *batch size* and *dataset size*?

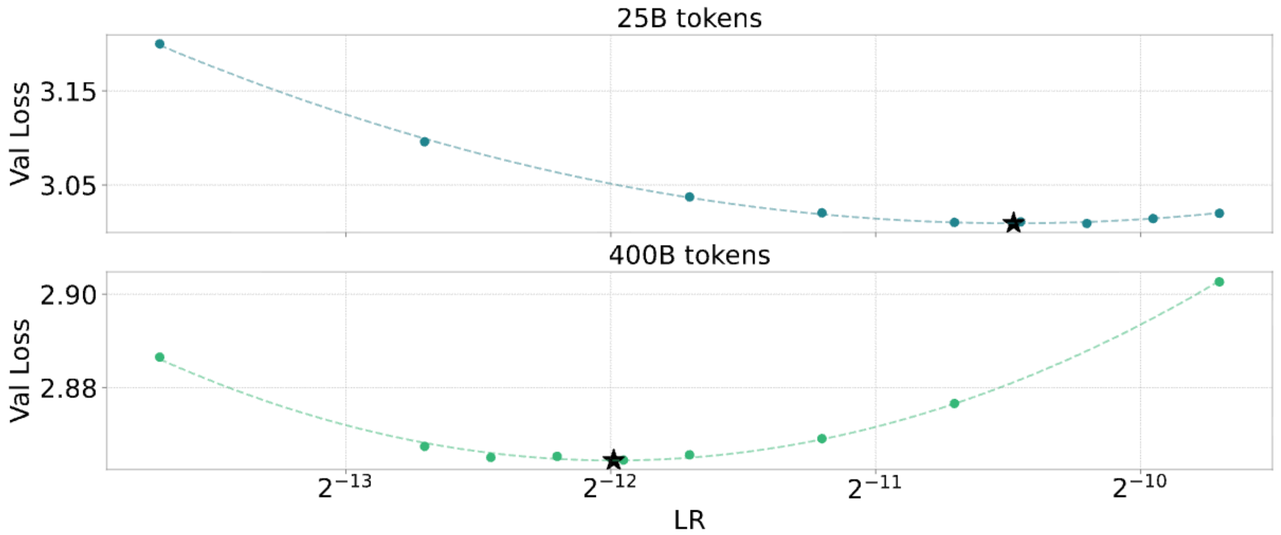
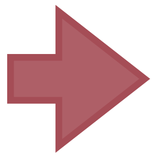


Figure from Bjorck et al. (2024)

Message 5

In Regime 3 it is difficult to find the hyperparams.



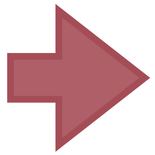
You can search small and scale up for some!

Conclusion

How do we train networks?

Message 1

Designing Gradient Based Optimizers



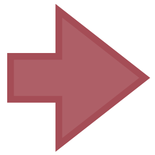
How to go:

from *infinitesimal* **to** *local*

(to global)

Message 3

Designing Gradient Based Optimizers



It’s about:

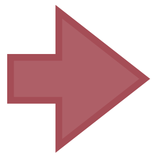
- **How you choose the step size**
- **What you make of the history**

Message 4

- We want to pick the architecture and the initialization to enhance trainability.
- Not really for purposes of approximation or location of convergence!

Message 5

In Regime 3 it is difficult to find the hyperparams.



You can search small and scale up for some!

Principles

- We want to pick as we said:

  The architecture and the initialization to enhance trainability.

- We can pick the right gradient-based method choosing step size and history:

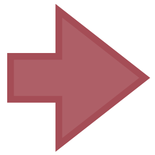
  Hessentially following 1/Hessian… (maybe not).

- We want to search hyperparameters in a small space and scale up.

Next Lecture: Message 1

Designing Gradient

Based Optimizers

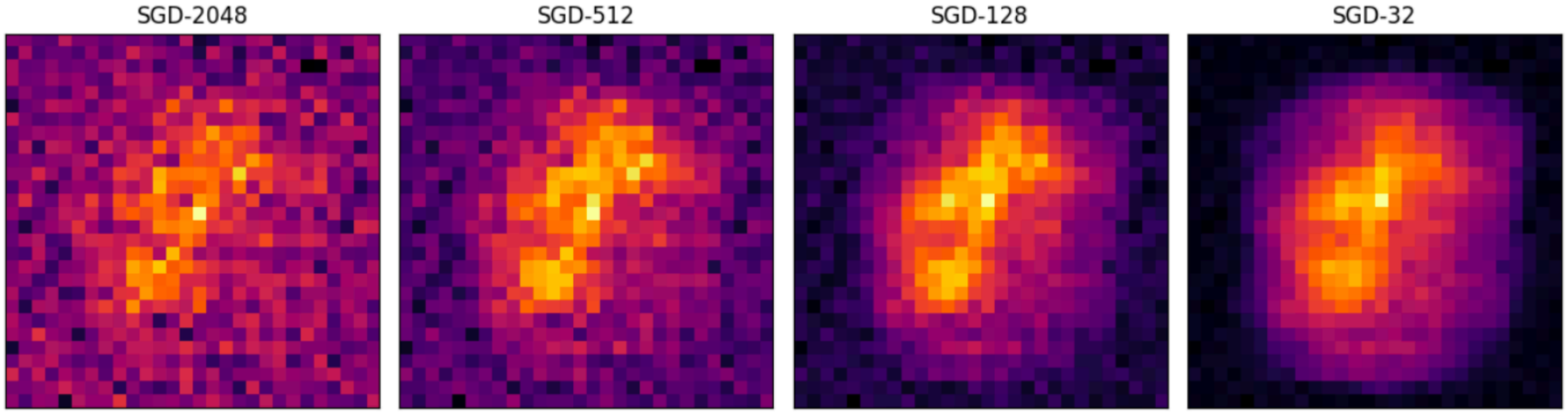


How to go:

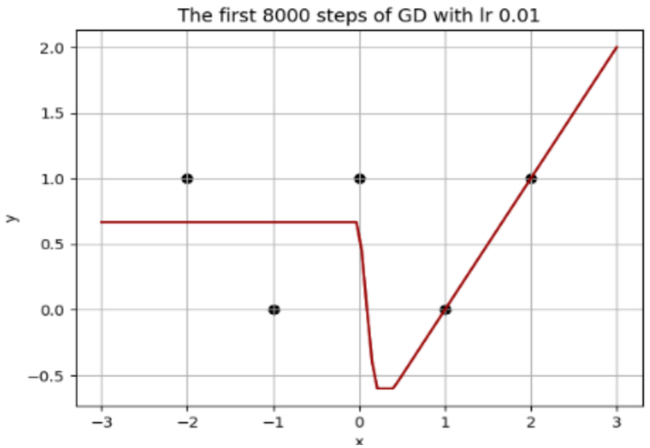
from *infinitesimal* **to** *local*

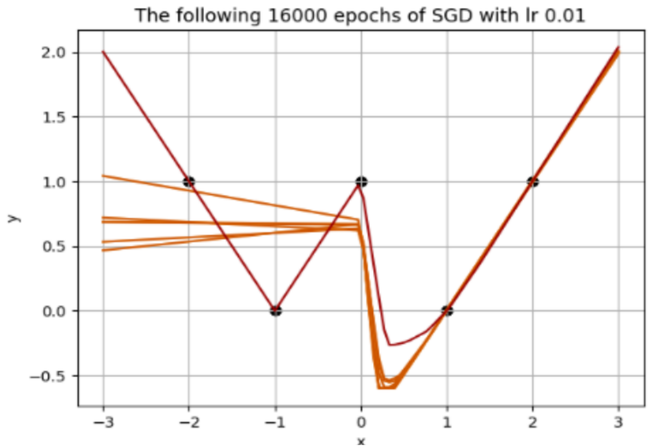
(to global)

What is this Picture Missing?



SGD smaller batch size





Location of Convergence

It is not just about if you go down within compute:

- It's also about where you go within compute!
- We discussed how to do your best in the step:

Infinitesimal to *local*

- We did not say anything about:

Local to *Global*

Next class is about local to global:

On the location of convergence and the phases of training