# Lec 2: Optimization

## CS 2281, Fall 2024

**Sham Kakade and Nikhil Vyas**

**(+Depen Morwani for contributions to the slides)**

# Today

HW2 is out

- Announcements/Recap++
- Whirlwind Tour of Optimization
- DL Optimization
- Training Dynamics/Edge of stability

# Recap++

# A Computational Graph (aka the "Evaluation Trace")

- Compute $f(w_1, w_2)$:

  input: $z_0 = (w_1, w_2)$

  $z_1 = w_1/w_2$

  $z_2 = \sin(2\pi z_1)$

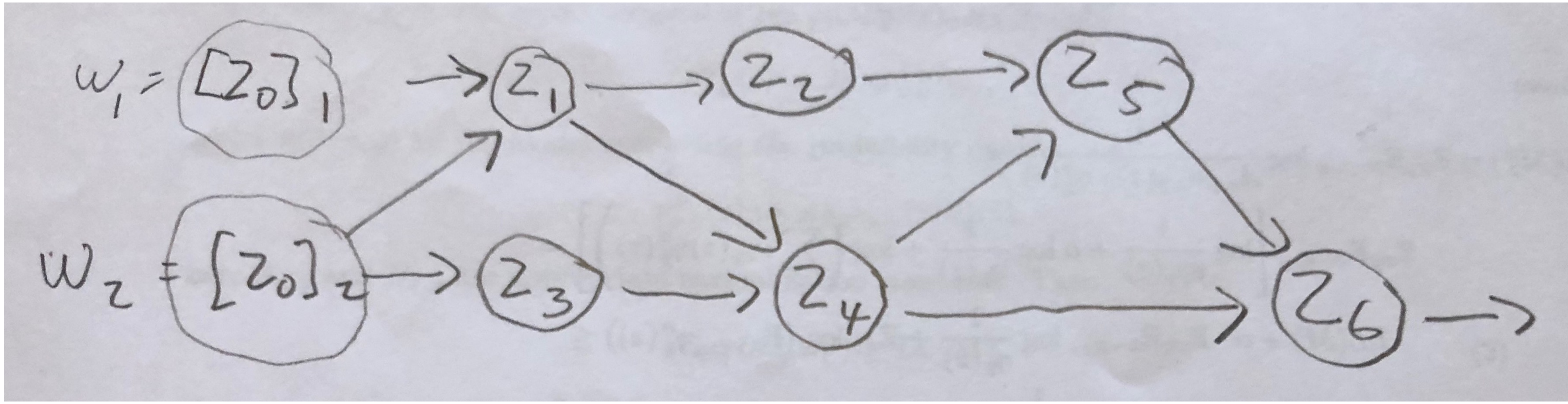
  $z_3 = \exp(2w_2)$

  $z_4 = 3z_1 - z_3$

  $z_5 = z_2 + z_4$

  $z_6 = z_4 z_5$

  return: $z_6$




- The computation graph is the flow of operations.
- We say that: $z_2$ and $z_4$ and children of $z_1$; $z_5$ is a child of $z_2$; etc.

# The Reverse Mode of AD

**Forward pass:**

1.Compute $f(w)$ and store in memory all the intermediate variables $z_{0:T}$.

**Backward pass:**

2. Initialize:

$$\frac{dz_T}{dz_T} = 1$$

3.Proceeding recursively, starting at $t = T - 1$ and going to $t = 0$

$$\frac{\partial z_T}{\partial z_t} = \sum_{c \text{ is a child of } t} \frac{\partial z_T}{\partial z_c} \frac{\partial z_c}{\partial z_t}$$

4. **Return:**

$$\frac{dz_T}{dz_0} = \frac{df}{dw}$$

(which is the desired answer as $z_T = f, z_0 = w$)

**Everything works if we allow $z_t$ to be vectors or matrices.**

# Time Complexity

- History of AD: Linnainmaa (Lin76), Werbos(82), …

Theorem: [BaurStrassen 83] Suppose that $h \in \mathscr{H}$ are of the form:

- Affine functions.
- A product of terms.
- Fixed functions, like $\cos(), \sin(), \exp(), \log()$, where computing $h'(x)$ is no more than $5$x the cost of computing $h(x)$

The Reverse Mode of AD computes $\nabla f(x)$ in time no more than a factor of $5$ than the program used to compute $f(x)$.

Proof sketch (basically a book keeping argument):
in the forward pass, we associate the computation along edges from parents to a child. In the backward pass, note $\frac{\partial z_c}{\partial z_t}$ only is computed once.

$$\frac{\partial z_T}{\partial z_t} = \sum_{c \text{ is a child of } t} \frac{\partial z_T}{\partial z_c} \frac{\partial z_c}{\partial z_t}$$

# The Reverse Mode of AD, with Checkpointing

Assume $z_{t+1}$ is only a function of the variables $z_t$ (here let the intermediate variables be vectors)

**Checkpoint** indexes: $C = \{\tau_1 \leq \tau_2 \ldots \leq \tau_k\}$, i.e. $C \subset \{1, \ldots T\}$.

**Forward pass:**

1.Compute $f(w)$ and store only the variables $\{z_\tau : \tau \in C\}$.

**Backward pass:**

2. Initialize: $\frac{dz_T}{dz_T} = 1$, set $\tau_{k+1} = T$

3.Proceeding recursively, for $i = k, \ldots 1$

- **Rematerialization:**

  Redo forward pass, computing/storing the graph in “block” k, from $t = \tau_i$ to $t = \tau_{i+1}$
- Backward pass in “block” k: Starting at $t = \tau_{i+1}$ and going to $t = \tau_i$

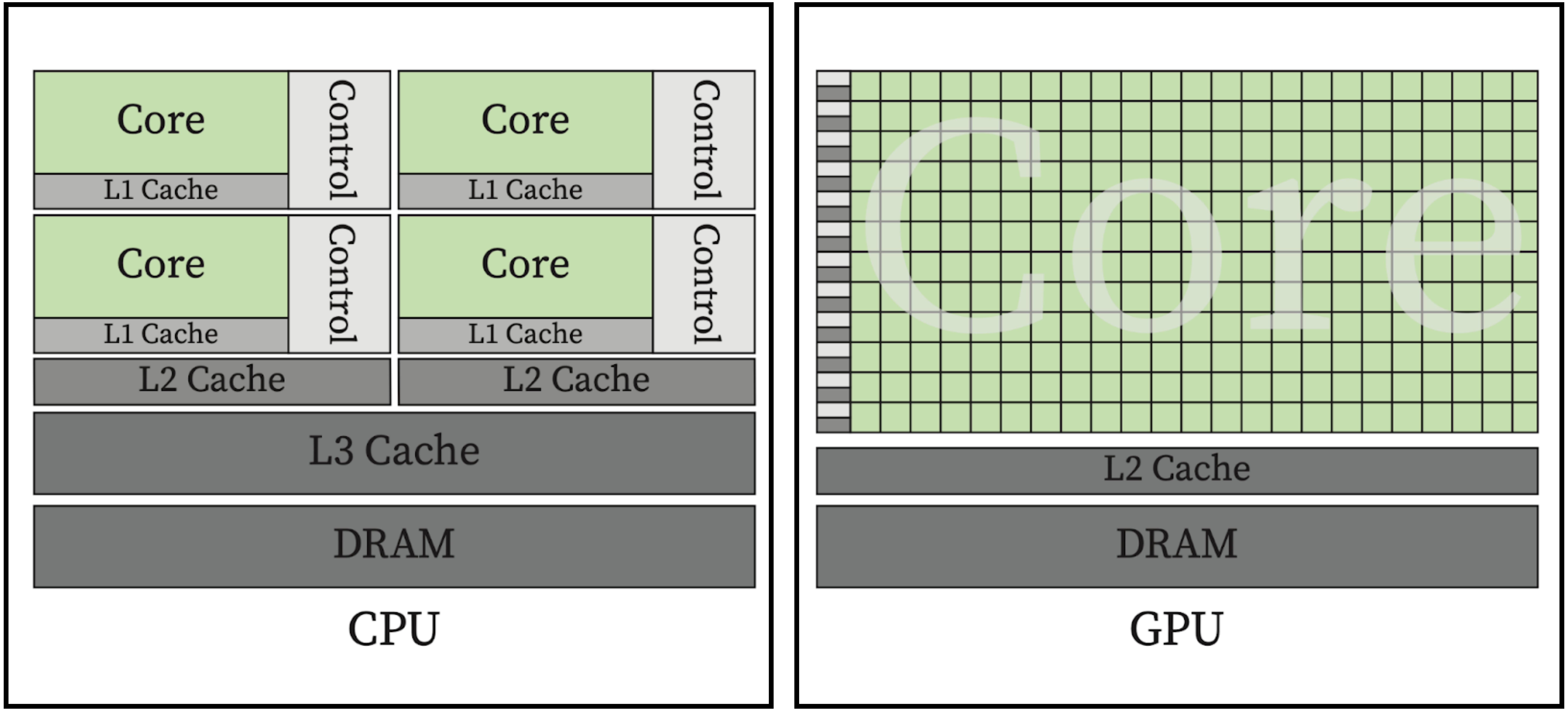
  $\frac{\partial z_T}{\partial z_t} = \sum_{c \text{ is a child of } t} \frac{\partial z_T}{\partial z_c} \frac{\partial z_c}{\partial z_t}$

4. **Return:** $\frac{dz_T}{dz_0}$

Memory required: store $\{z_\tau : \tau \in C\}$; store all variables in a “block” rematerialization pass

Compute overhead: need to recompute all the “blocks”, which is at most the cost to compute $f(x)$.

# GPUs vs CPUs




[Figure credit: Yasin Mazloumi]

# Transformer Memory: Forward & Computational Graph

Compute $Loss()$, (for $Bsz = 1, N_{heads} = 1$) :

**input:** parameters (embedding and MLP weights), data $X \in \{0,1\}^{TxV}$

$d$: hidden dim, $B$: batch size, $T$: context size, $L$: # layers, V: vocab size, $N_{heads}$: #heads

- Embed data: $X \leftarrow XW_{embed}$
- For $\ell = 0, \ldots L - 1$
  - Attention:
    - $Q = XW_Q^\ell, K = XW_K^\ell, V = XW_V^\ell$
    - $X \leftarrow$ MaskedRowSoftmax$(QK^\top)V$
  - MLP layers: ($d \rightarrow 4d \rightarrow d$)
    - $X \leftarrow \sigma(XW_1^\ell)$
    - $X \leftarrow \sigma(XW_2^\ell)$
    - $X \leftarrow XW_{proj}^\ell$
- Compute $X \leftarrow XW_{unemebed}$, Return LogLoss

With $N_{heads} \neq 1$

- Mem Transformer Params: $12d^2L + 2dV$
- Sufficient Memory for Forward pass: $4dT + N_{heads}T^2 + VT$
- Memory Created in the Graph: $12LdT + LN_{heads}T^2 + 2VT$

What about with $B \neq 1$?

# Checkpointing ($B = 1$ case)

$d$: hidden dim, $B$: batch size, $T$: context size, $L$: # layers, V: vocab size, $N_{heads}$: #heads

- Embed data: $X \leftarrow XW_{embed}$
- For $\ell = 0,\ldots L-1$
  - "block input" X (checkpoint here)
  - Attention:
    - $Q = XW_Q^\ell, K = XW_K^\ell, V = XW_V^\ell$
    - $X \leftarrow$ MaskedRowSoftmax$(QK^\top)V$
  - MLP layers: ($d \rightarrow 4d \rightarrow d$)
    - $X \leftarrow \sigma(XW_1^\ell)$
    - $X \leftarrow \sigma(XW_2^\ell)$
    - $X \leftarrow XW_{proj}^\ell$
- Compute $X \leftarrow XW_{unemebed}$, Return LogLoss

# Checkpointing ($B = 1$ case)

$d$: hidden dim, $B$: batch size, $T$: context size, $L$: # layers, V: vocab size, $N_{heads}$: #heads

- Embed data: $X \leftarrow XW_{embed}$
- For $\ell = 0,\ldots L-1$
  - "block input" X (checkpoint here)
  - Attention:
    - $Q = XW_Q^\ell, K = XW_K^\ell, V = XW_V^\ell$
    - $X \leftarrow \text{MaskedRowSoftmax}(QK^\top)V$
  - MLP layers: ($d \rightarrow 4d \rightarrow d$)
    - $X \leftarrow \sigma(XW_1^\ell)$
    - $X \leftarrow \sigma(XW_2^\ell)$
    - $X \leftarrow XW_{proj}^\ell$
- Compute $X \leftarrow XW_{unemebed}$, Return LogLoss

- Mem Transformer Params: $12d^2L + 2dV$

# Checkpointing ($B = 1$ case)

$d$: hidden dim, $B$: batch size, $T$: context size, $L$: # layers, V: vocab size, $N_{heads}$: #heads

- Embed data: $X \leftarrow XW_{embed}$
- For $\ell = 0, \ldots L-1$
  - "block input" X (checkpoint here)
  - Attention:
    - $Q = XW_Q^\ell, K = XW_K^\ell, V = XW_V^\ell$
    - $X \leftarrow$ MaskedRowSoftmax$(QK^\top)V$
  - MLP layers: ($d \to 4d \to d$)
    - $X \leftarrow \sigma(XW_1^\ell)$
    - $X \leftarrow \sigma(XW_2^\ell)$
    - $X \leftarrow XW_{proj}^\ell$
- Compute $X \leftarrow XW_{unemebed}$, Return LogLoss

- Mem Transformer Params: $12d^2L + 2dV$
- Option 1: Checkpointing (ignoring the $V$ part)
  - Memory for Checkpointing: $LdT$
  - Comp Graph Memory for Rematerialization: $12dT + N_{heads}T^2$
  - Computational overhead is basically a full factor of 2 (everything but $W_{proj}^\ell$ must be recomputed)

# Checkpointing ($B = 1$ case)

$d$: hidden dim, $B$: batch size, $T$: context size, $L$: # layers, V: vocab size, $N_{heads}$: #heads

- Embed data: $X \leftarrow XW_{embed}$
- For $\ell = 0, \ldots L-1$
  - “block input” X (checkpoint here)
  - Attention:
    - $Q = XW_Q^\ell, K = XW_K^\ell, V = XW_V^\ell$
    - $X \leftarrow$ MaskedRowSoftmax$(QK^\top)V$
  - MLP layers: ($d \rightarrow 4d \rightarrow d$)
    - $X \leftarrow \sigma(XW_1^\ell)$
    - $X \leftarrow \sigma(XW_2^\ell)$
    - $X \leftarrow XW_{proj}^\ell$
- Compute $X \leftarrow XW_{unemebed}$, Return LogLoss

- Mem Transformer Params: $12d^2L + 2dV$
- Option 1: Checkpointing (ignoring the $V$ part)
  - Memory for Checkpointing: $LdT$
  - Comp Graph Memory for Rematerialization: $12dT + N_{heads}T^2$
  - Computational overhead is basically a full factor of 2 (everything but $W_{proj}^\ell$ must be recomputed)
- Option 2: checkpoint everything but the $T \times T$ attention matrices
  - This saves on compute (the weight matrix multiples often costly) and needs $12LdT$ memory

# Flash Attention Simplified:

- Single Head Attention: $X \leftarrow \text{MaskedRowSoftmax}(QK^\top)V$
  - This requires having $T^2$ free memory, even though our output is of size $d \times T$
  - The computational cost is $O(dT^2)$
- Simpler case: Suppose we just wanted to do the following, where $\exp()$ is componentwise:

$$X \leftarrow \exp(QK^\top)V$$

  - Again, this require $T^2$ free memory and the same flops.

Can we do better on memory, using the same flops?

- Observe that that row $X[t, :]$ is equal to the t-th row of $\exp(QK^\top)$ times $V$.
  - What is the t-th row of $\exp(QK^\top)$?
  - This implies: $X[t, :] = \exp(Q[t, :] \cdot K^\top)V$
- So we can compute $X$ with a for loop over the $T$ rows.
  - The excess memory is now $O(T)$
  - The flops is still $O(dT^2)$

- Now do you see how compute $X \leftarrow \text{MaskedRowSoftmax}(QK^\top)V$ with less memory?
- FlashAttention: Now just checkpoint this approach.
  - But why do we do this on a gpu?
  - Fusing: the exp operations can be “fused” with vector multiplies to reduce memory movements on the GPU itself (this is why it is done on the GPU)

# Whirlwind Optimization Overview

# Today

- Announcements/Recap++
- Whirlwind Tour of Optimization
  - ✓ GD/Momentum/Newton's Method
  - SGD:
    - LR scheduling+averaging
    - batch size
- DL Optimization
- Training Dynamics/Edge of stability

# Gradient Descent

# Gradient Descent: warmup 1-dim

- Our general optimization problem: $\min_{w \in R^d} L(w)$

# Gradient Descent: warmup 1-dim

- Our general optimization problem: $\min_{w \in R^d} L(w)$
- Gradient descent:
  $w_{t+1} = w_t - \eta \nabla L(w_t)$

# Gradient Descent: warmup 1-dim

- Our general optimization problem: $\min_{w \in R^d} L(w)$
- Gradient descent:
  $w_{t+1} = w_t - \eta \nabla L(w_t)$

$\nabla L(w) = w - c$

- Consider the 1d convex quadratic case, $L(w) = \frac{1}{2}(w - c)^2$. GD is:

# Gradient Descent: warmup 1-dim

- Our general optimization problem: $\min_{w \in R^d} L(w)$
- Gradient descent:
  $w_{t+1} = w_t - \eta \nabla L(w_t)$
- Consider the 1d convex quadratic case, $L(w) = \frac{1}{2}(w - c)^2$. GD is:
- What is GD in terms of $w^\star$?
- What learning rates guarantee convergence?

$= w - w^*$

$\nabla L(w) = w - c$

$w^* = c$

$w - w^* \leftarrow w - w^* - \eta (w - w^*)$

$(w_{t+1} - w_*) = w_t - w_* - \eta [w_t - w^*]$

$= (1 - \eta)(w_t - w_*)$

$= (1 - \eta)^t (w_0 - w_*)$

$= -1 \cdot (w_t - w_*)$

$|(1 - \eta)| < 1$

$\eta < 2$ converge

$\eta = 2$ bounce around

$\eta > 2$ diverge

# Gradient Descent: warmup 1-dim

- Our general optimization problem: $\min_{w \in R^d} L(w)$
- Gradient descent:
  $w_{t+1} = w_t - \eta \nabla L(w_t)$
- Consider the 1d convex quadratic case, $L(w) = \frac{1}{2}(w - c)^2$. GD is:
- What is GD in terms of $w^\star$?
- What learning rates guarantee convergence?
- What is a good setting of $\eta$ for well behaved “smooth” functions?

# Gradient Descent: warmup 1-dim

- Our general optimization problem: $\min_{w \in R^d} L(w)$
- Gradient descent:
  $w_{t+1} = w_t - \eta \nabla L(w_t)$

- Consider the 1d convex quadratic case, $L(w) = \frac{1}{2}(w - c)^2$. GD is:
- What is GD in terms of $w^\star$?

- What learning rates guarantee convergence?

- What is a good setting of $\eta$ for well behaved "smooth" functions?

$$L(w + \delta) \approx L(w) + L'(w)\delta + \frac{1}{2}L''(w)\delta^2$$

$\delta = \left(-\frac{1}{L''(w)}\right) L'(w)$

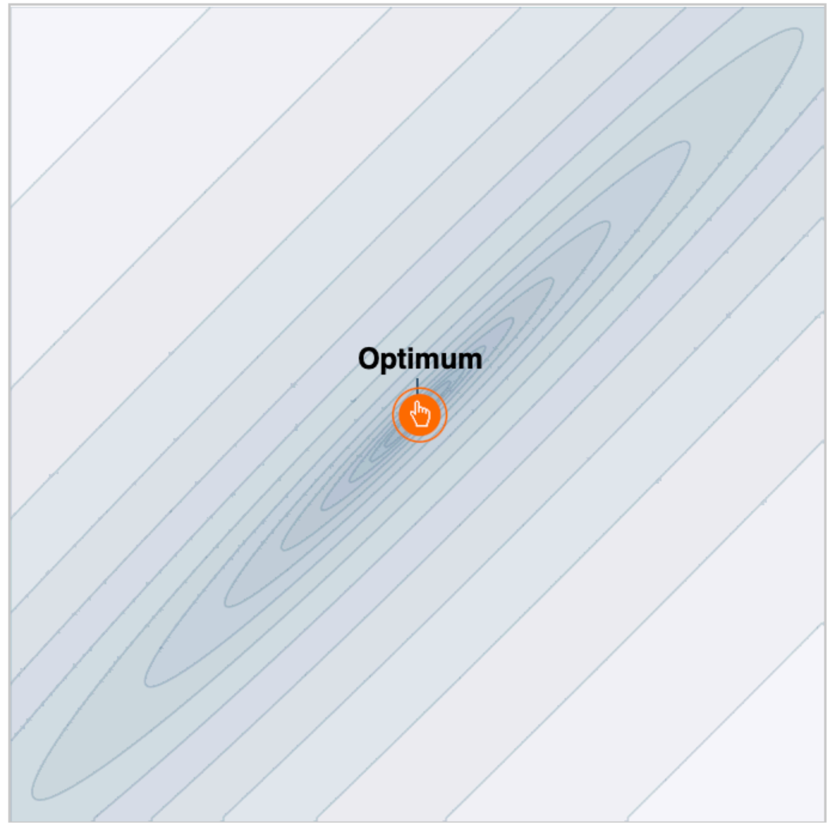
stepsize.

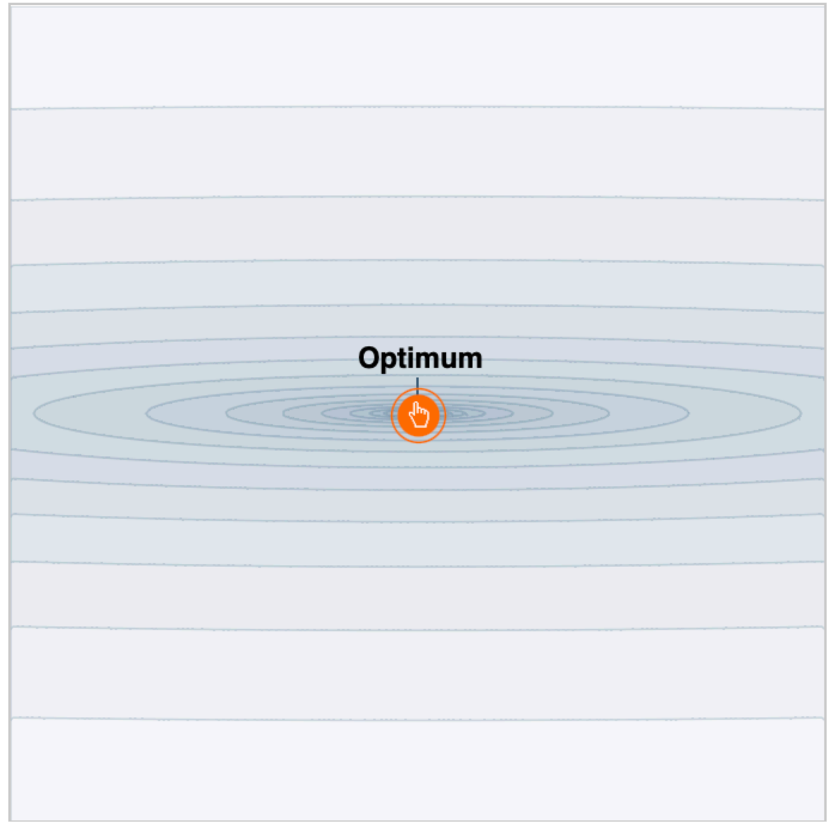
# Gradient Descent: convex quadratics

- Consider $L(w) = \frac{1}{2}w^\top Aw + bw + c$,

  for positive def symmetric matrix $A$, vector $b$, scalar $c$

$\nabla L(w) = Aw + b$



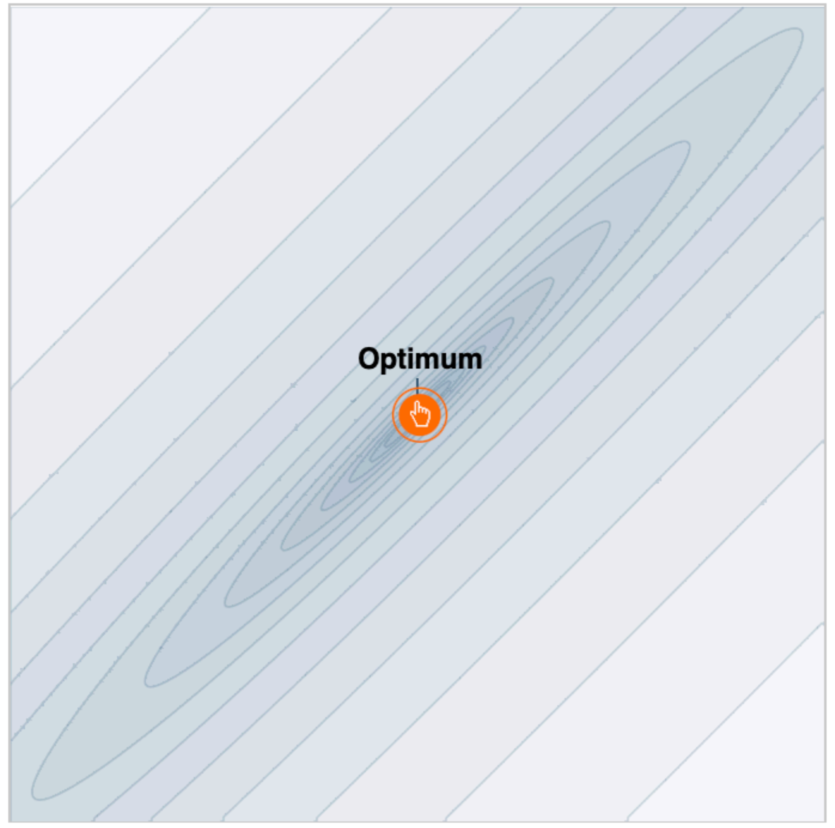



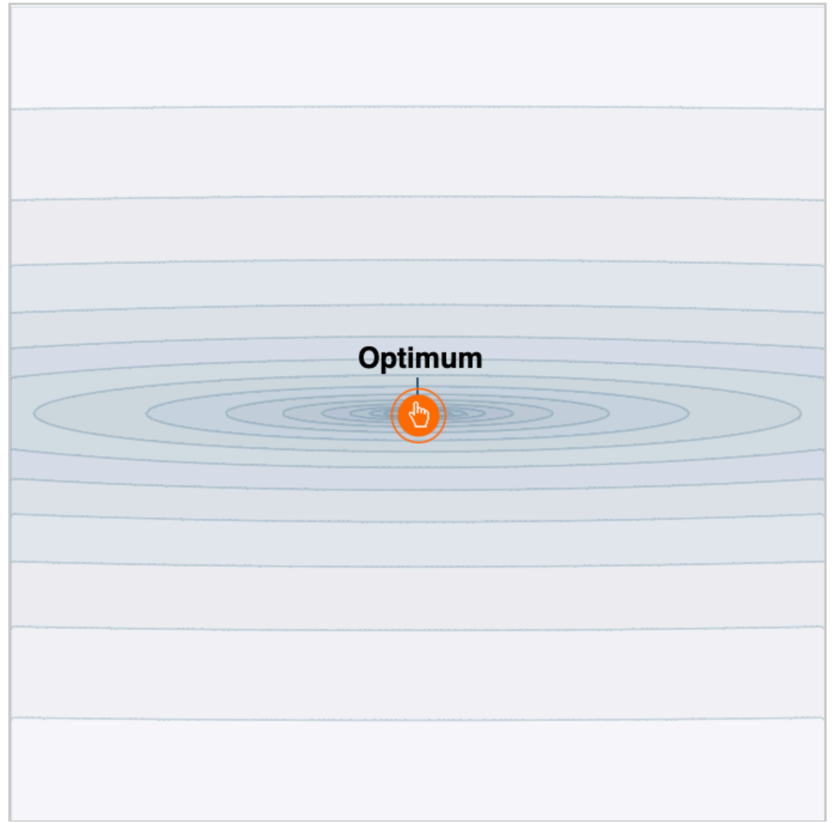
# Gradient Descent: convex quadratics

- Consider $L(w) = \dfrac{1}{2} w^\top A w + bw + c$,

  for positive def symmetric matrix $A$, vector $b$, scalar $c$
- Gradient descent:

  $w \leftarrow w - \eta(Aw - b)$



# Gradient Descent: convex quadratics

- Consider $L(w) = \frac{1}{2} w^\top A w + bw + c,$

  for positive def symmetric matrix $A$, vector $b$, scalar $c$
- Gradient descent:

  $w \leftarrow w - \eta(Aw + b)$
- Gradient descent in terms of $w^\star$:

$$w^\star = -A^{-1}b$$

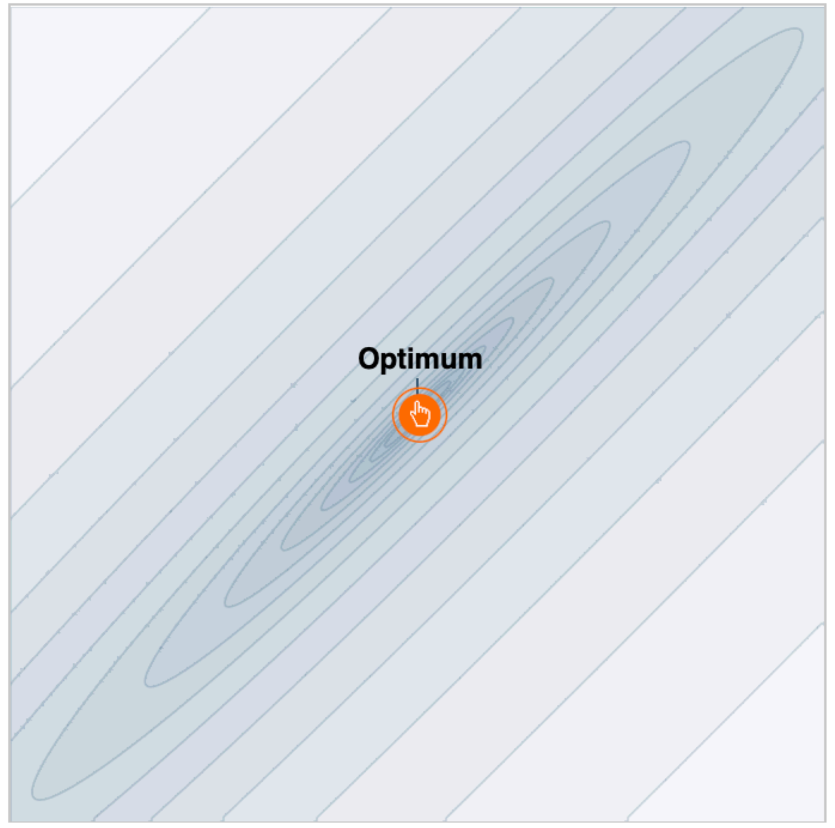
$$Aw^\star + b = 0$$

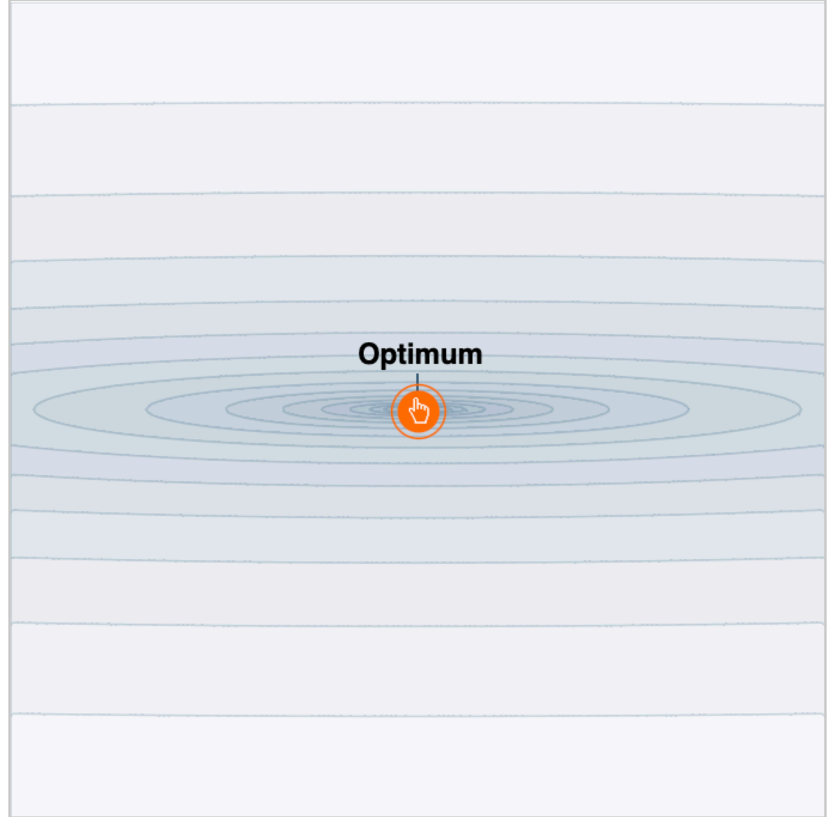
$$b = -Aw^\star$$

$$w \leftarrow w - \eta(Aw - Aw^\star)$$

$$(w - w^\star) \leftarrow w - w^\star - \eta A(w - w^\star)$$

$$(w - w^\star) \leftarrow (I - \eta A)(w - w^\star)$$



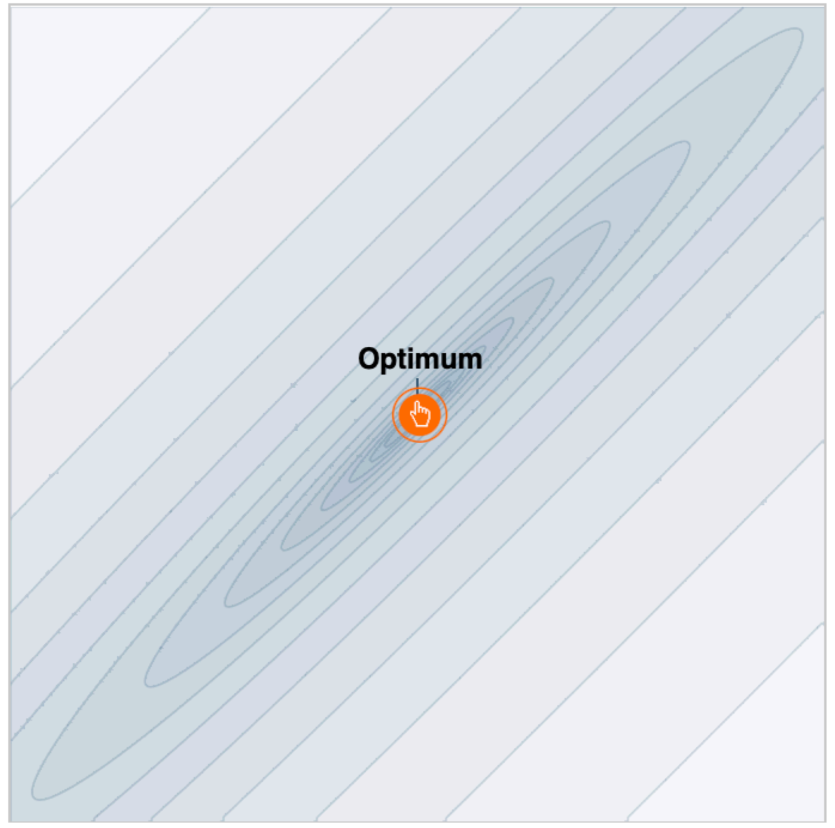
# Gradient Descent: convex quadratics

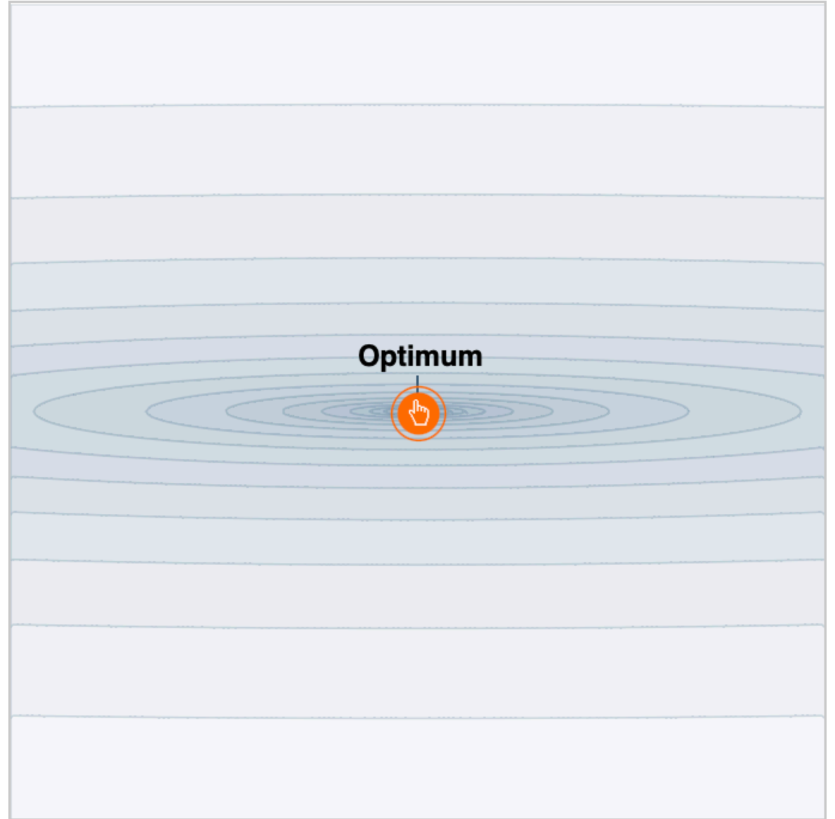
- Consider $L(w) = \frac{1}{2} w^\top A w + bw + c$,

  for positive def symmetric matrix $A$, vector $b$, scalar $c$
- Gradient descent:

  $w \leftarrow w - \eta(Aw - b)$
- Gradient descent in terms of $w^\star$:

- Let $A = UDU^\top$ be the SVD of A.



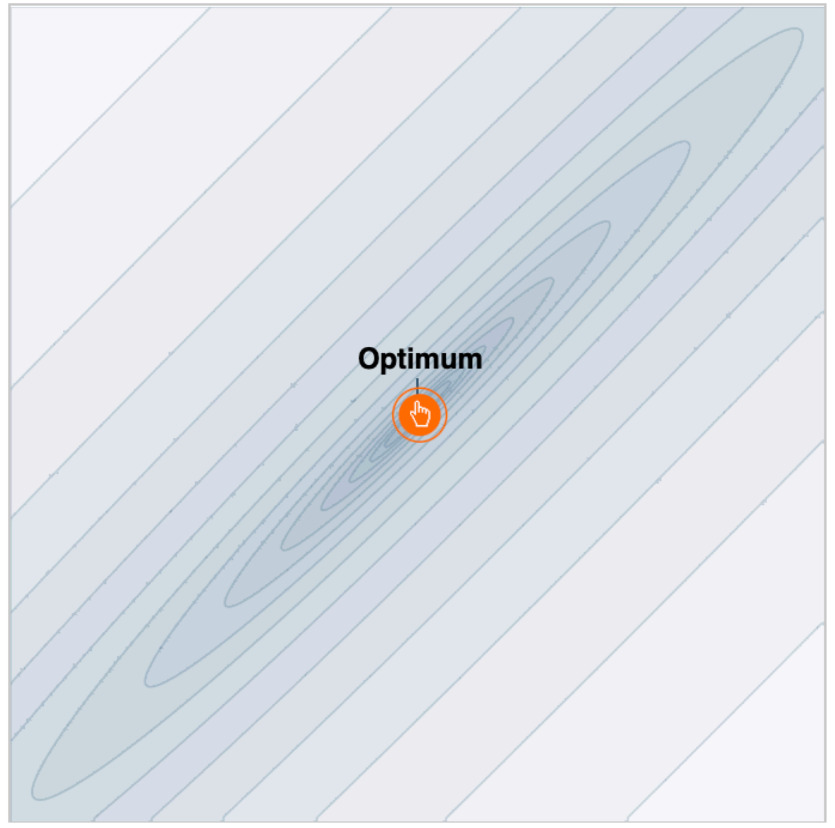



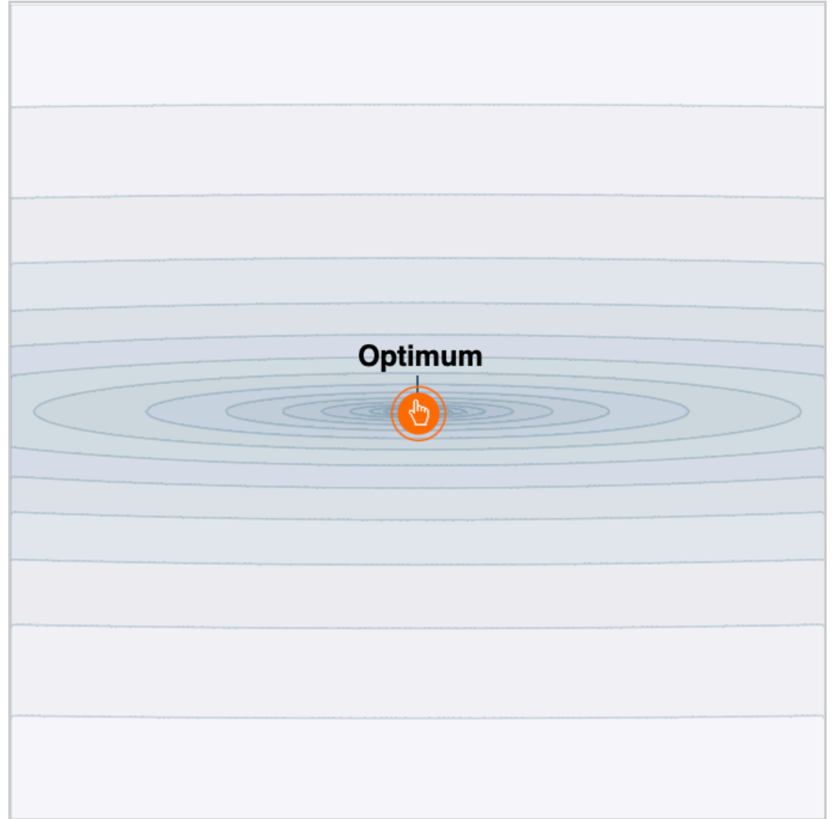

$D = \text{diag}(\lambda_1, \ldots \lambda_d)$

# Gradient Descent: convex quadratics

$\vec{b} \cdot \vec{w} = b^\top w$

- Consider $L(w) = \frac{1}{2} w^\top A w + bw + c$,
  for positive def symmetric matrix $A$, vector $b$, scalar $c$
- Gradient descent:
  $w \leftarrow w - \eta(Aw - b)$
- Gradient descent in terms of $w^\star$:




$\tilde{b} = U^\top b$

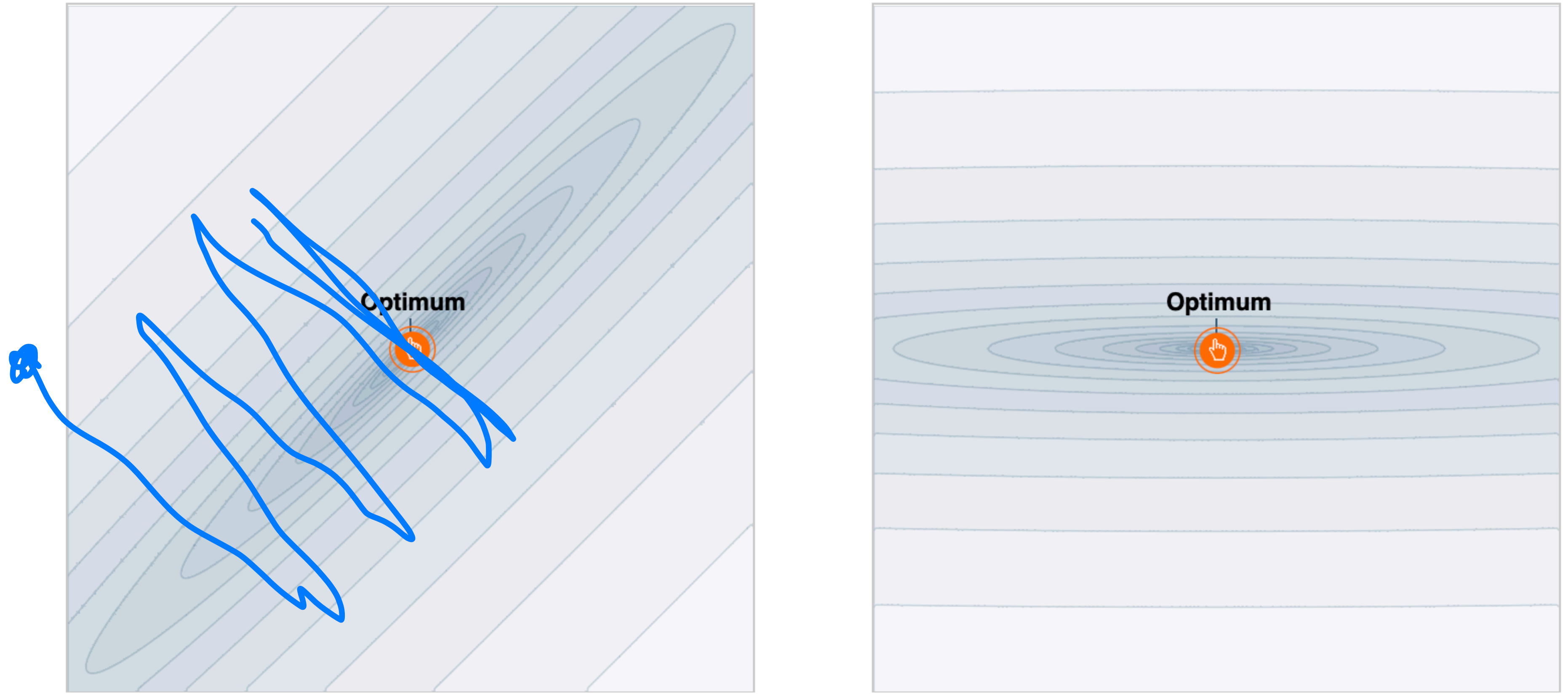
- Let $A = UDU^\top$ be the SVD of A.
- Now let us rotate coordinates (to the the eigenbasis):
  $\tilde{w} = U^\top w,$ and $L(\tilde{w}) =$ $\frac{1}{2} \tilde{w}^\top D \tilde{w} + \tilde{b} w + c$

$w = U \tilde{w}$

# Gradient Descent: convex quadratics

$\eta = \frac{2}{\lambda_{max}}$

- Consider $L(w) = \frac{1}{2} w^\top A w + bw + c,$
  for positive def symmetric matrix $A$, vector $b$, scalar $c$
- Gradient descent:
  $w \leftarrow w - \eta(Aw - b)$
- Gradient descent in terms of $w^\star$:
  $w - w^* \leftarrow (I - \eta A)(w - w^*)$
- Let $A = UDU^\top$ be the SVD of A.
- Now let us rotate coordinates (to the the eigenbasis):
  $\tilde{w} = U^\top w,$ and $L(\tilde{w}) =$
- What is the GD update rule in the new coordinate system?
  $\tilde{w} - \tilde{w}^\star \leftarrow (I - \eta D)(\tilde{w} - \tilde{w}^\star)$



# GD dynamics

- The GD update rule (in the eigenbasis):
  $\tilde{w} - \tilde{w}^\star \leftarrow (I - \eta D)(\tilde{w} - \tilde{w}^\star)$
- What is the update rule per coordinate and what is the iterate at time $t$?

# GD dynamics

- The GD update rule (in the eigenbasis):
  $\tilde{w} - \tilde{w}^{\star} \leftarrow (I - \eta D)(\tilde{w} - \tilde{w}^{\star})$

$D_{ii} = \lambda_i$

- What is the update rule per coordinate and what is the iterate at time $t$?

$$\left(\tilde{w}_{t+1}[i] - \tilde{w}^{*}_{t+1}[i]\right) = (1 - \eta\lambda_i)\left(\tilde{w}_t[i] - \tilde{w}^{*}_t[i]\right)$$

$$= (1 - \eta\lambda_i)^t \left(\tilde{w}_0[i] - \tilde{w}^{*}_0[i]\right)$$

- What learning rates guarantee convergence?

$\eta < 2/\lambda_{max}$ ← converge $\quad \forall i \; |1 - \eta\lambda_i| \le 1$

$\eta = 2/\lambda_{max}$ ← bounce in largest eig dir.

$\eta > 2/\lambda_{max}$ ← diverge in at least one direction

suppose $\eta = 1/\lambda_{max}$ for $\lambda_{min}$ dir

$\left(1 - \frac{\lambda_{min}}{\lambda_{max}}\right)$ ← contraction factor

# GD dynamics

- The GD update rule (in the eigenbasis):
  $\tilde{w} - \tilde{w}^\star \leftarrow (I - \eta D)(\tilde{w} - \tilde{w}^\star)$
- What is the update rule per coordinate and what is the iterate at time $t$?

- What learning rates guarantee convergence?

- Suppose $\lambda_1 > \lambda_2 \geq \ldots \lambda_d$. What are the dynamics “at the edge” (when $\eta = 2/\lambda_1$)?

# GD dynamics

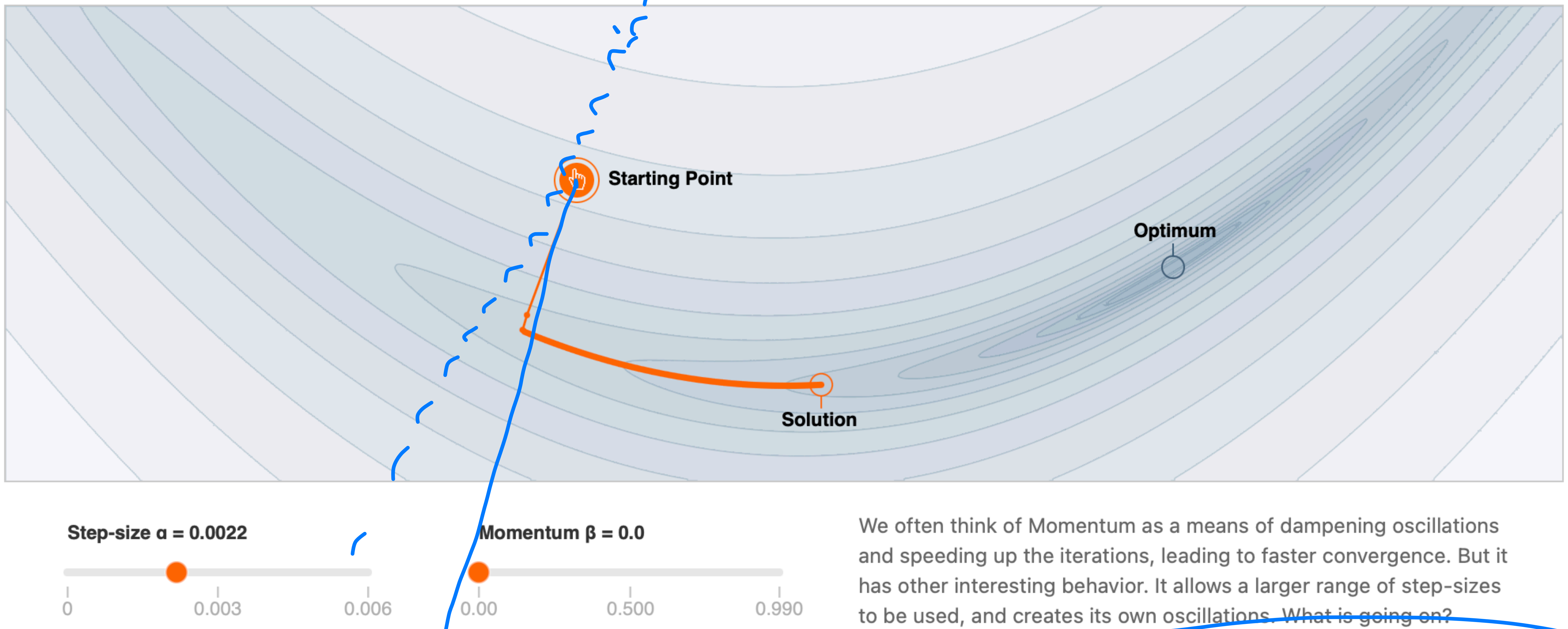
- The GD update rule (in the eigenbasis):
  $\tilde{w} - \tilde{w}^\star \leftarrow (I - \eta D)(\tilde{w} - \tilde{w}^\star)$
- What is the update rule per coordinate and what is the iterate at time $t$?

$$(1-\gamma)^t \approx e^{-\gamma t}$$

- What learning rates guarantee convergence?
- Suppose $\lambda_1 > \lambda_2 \geq \ldots \lambda_d$. What are the dynamics “at the edge” (when $\eta = 2/\lambda_1$)?
- What is the convergence rate for $\eta = 1/\lambda_{\max}$?

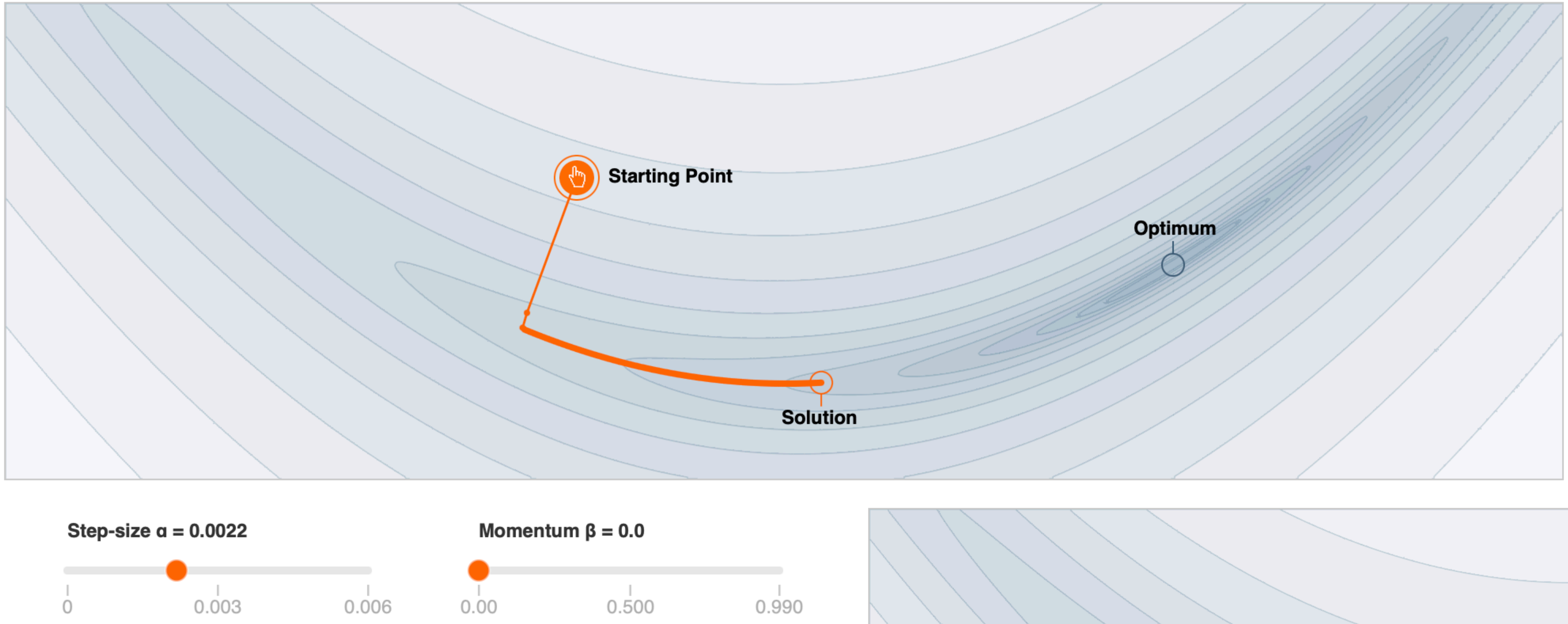
$$\|w_t - w^\star\|_2^2 \leq \exp(-t/\kappa)\|w_0 - w^\star\|_2^2, \text{ where } \kappa = \frac{\lambda_{\max}}{\lambda_{\min}}$$

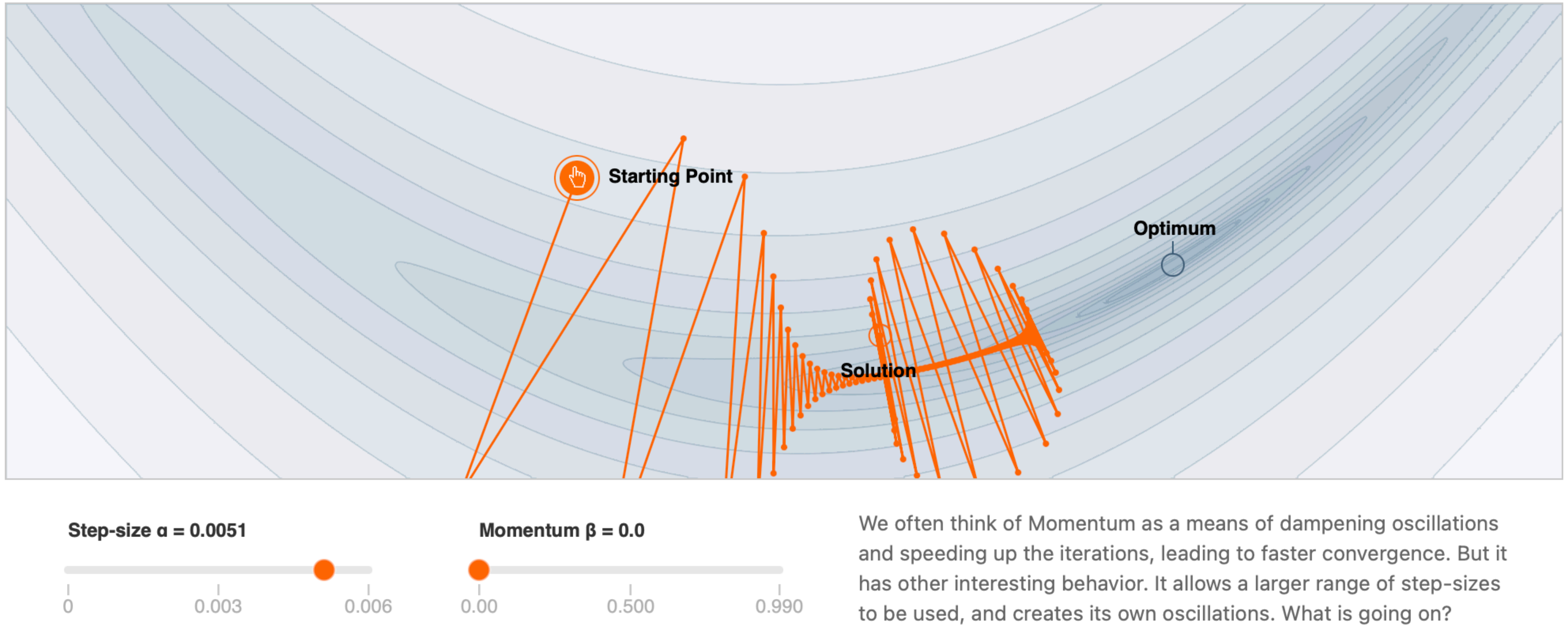
# GD dynamics




We often think of Momentum as a means of dampening oscillations and speeding up the iterations, leading to faster convergence. But it has other interesting behavior. It allows a larger range of step-sizes to be used, and creates its own oscillations. What is going on?

# GD dynamics




We often think of Momentum as a means of dampening oscillations and speeding up the iterations, leading to faster convergence. But it has other interesting behavior. It allows a larger range of step-sizes to be used, and creates its own oscillations. What is going on?

# GD + Momentum

# GD with Momentum

(aka the “heavy ball” method, Polyak ‘64)

# GD with Momentum

(aka the "heavy ball" method, Polyak '64)

- Gradient descent:
  $w \leftarrow w - \eta \nabla L(w)$

# GD with Momentum

(aka the “heavy ball” method, Polyak ‘64)

- Gradient descent:
  $w \leftarrow w - \eta \nabla L(w)$
- Gradient descent with momentum $0 \leq \gamma < 1$:
  $m \leftarrow \gamma m + \nabla L(w)$
  $w \leftarrow w - \eta m$

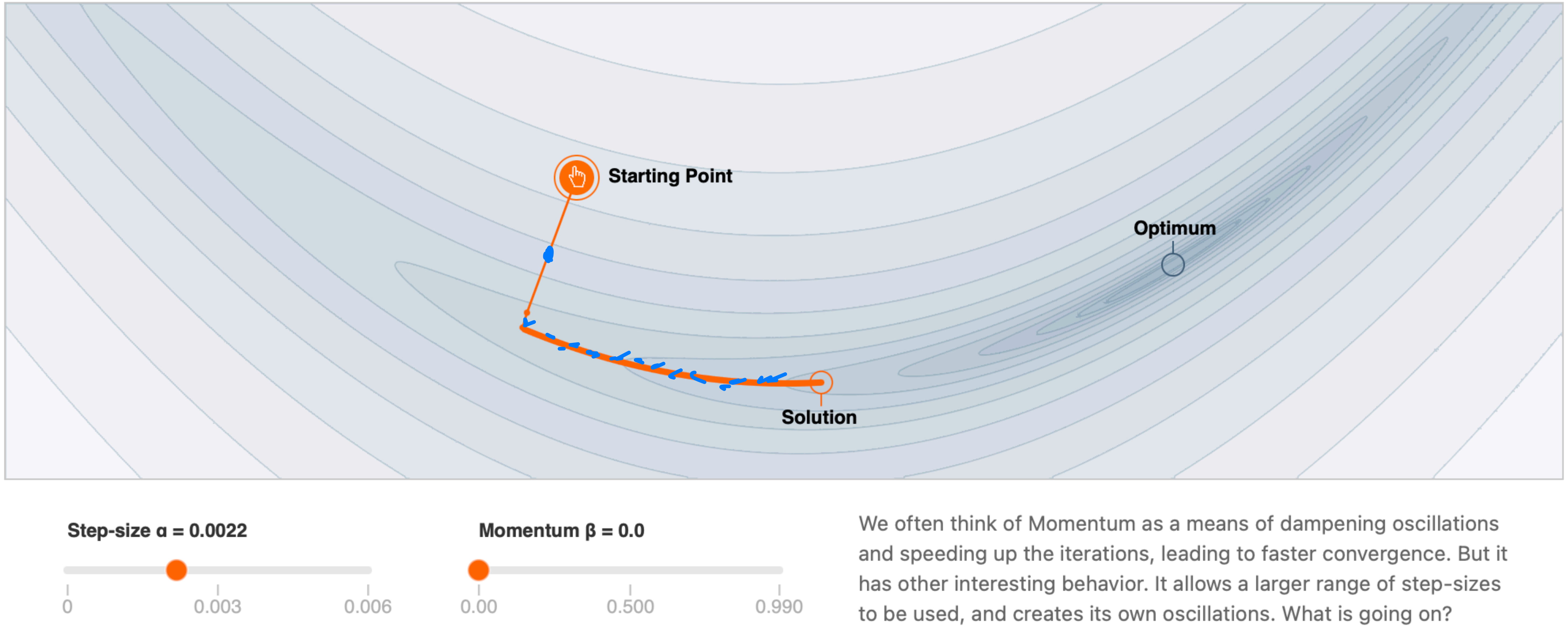
# GD with Momentum

(aka the “heavy ball” method, Polyak ‘64)

- Gradient descent:
  $w \leftarrow w - \eta \nabla L(w)$
- Gradient descent with momentum $0 \leq \gamma < 1$:
  $m \leftarrow \gamma m + \nabla L(w)$
  $w \leftarrow w - \eta m$

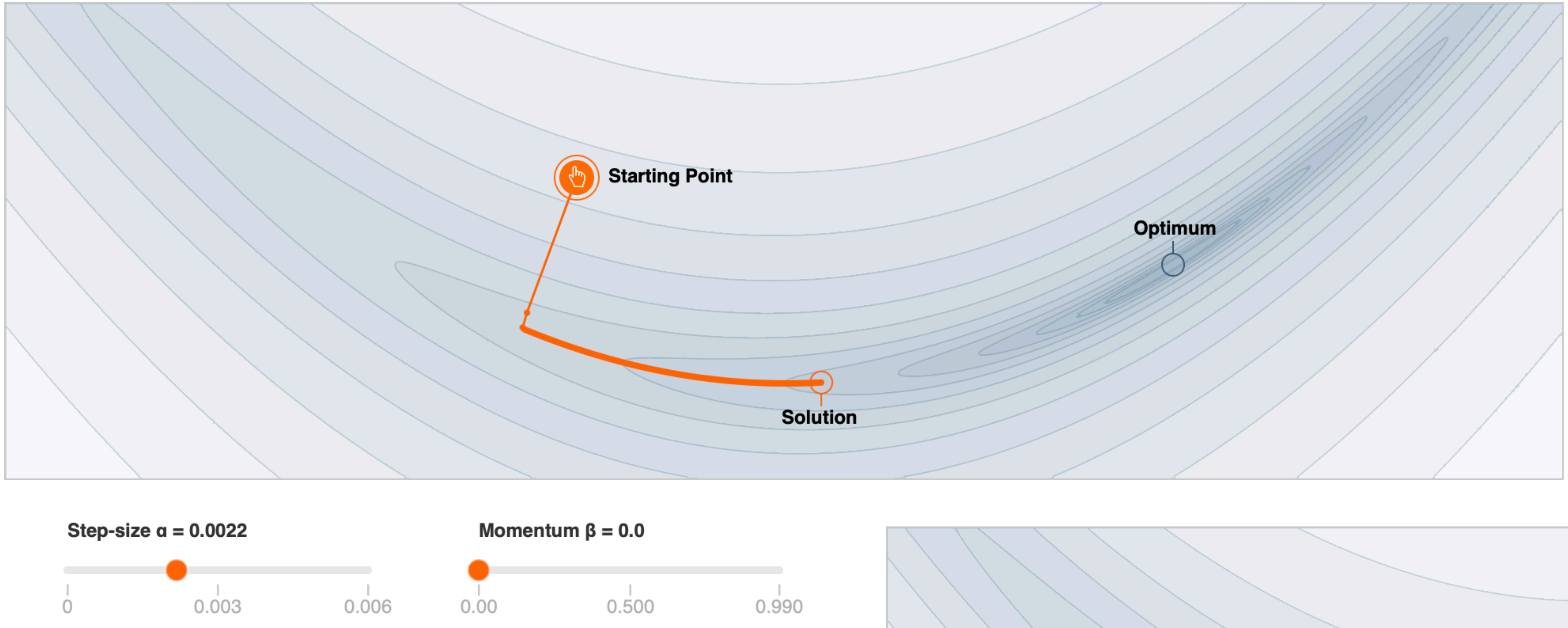
- GD+momentum (for quadratics) has convergence rate (with opt set params):
  $\|w_t - w^\star\|_2^2 \leq \exp\big(-t/\sqrt{\kappa}\big)\|w_0 - w^\star\|_2^2$, where $\kappa = \dfrac{\lambda_{\max}}{\lambda_{\min}}$

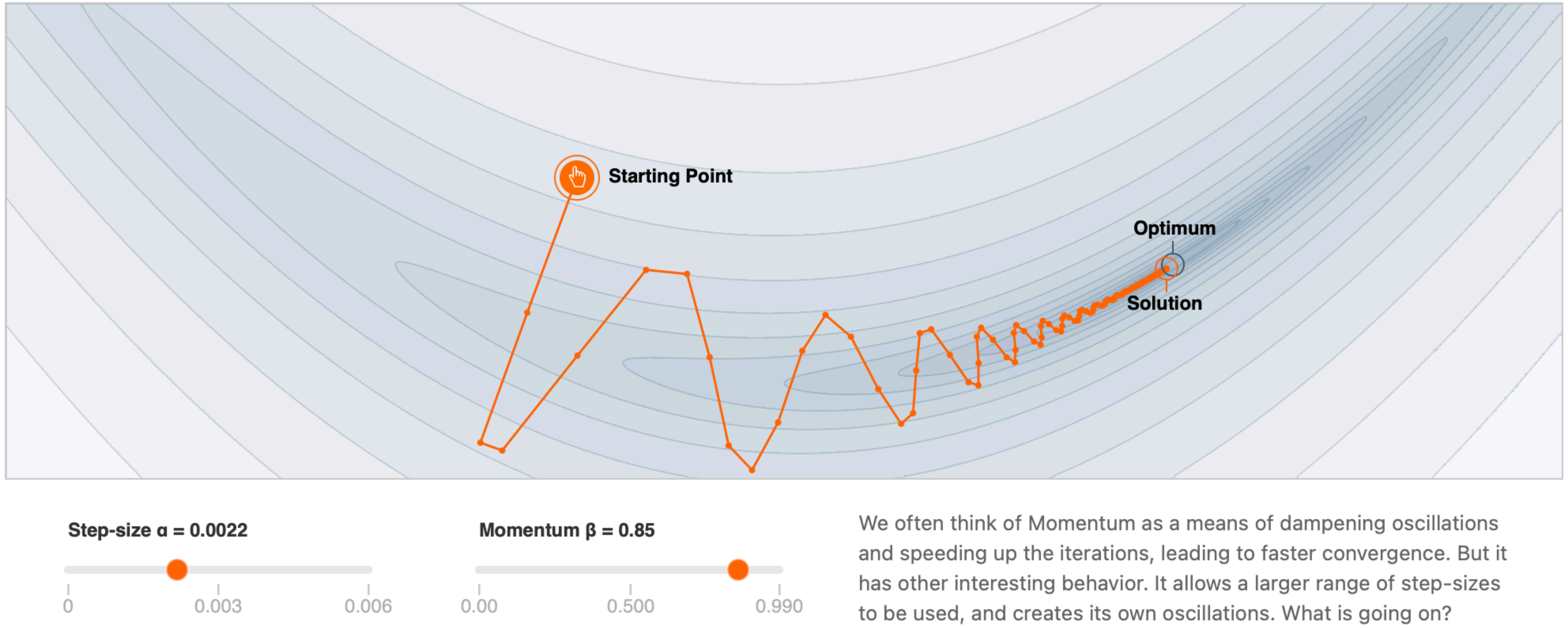
# GD + momentum dynamics




We often think of Momentum as a means of dampening oscillations and speeding up the iterations, leading to faster convergence. But it has other interesting behavior. It allows a larger range of step-sizes to be used, and creates its own oscillations. What is going on?

# GD + momentum dynamics




We often think of Momentum as a means of dampening oscillations and speeding up the iterations, leading to faster convergence. But it has other interesting behavior. It allows a larger range of step-sizes to be used, and creates its own oscillations. What is going on?

# Newtons Method

# Newton's Method

- Taylor's theorem around $w$

$$L(w + \Delta) \approx L(w) + \nabla L(w) \cdot \Delta + \frac{1}{2}\Delta^\top \left( \nabla^2 L(w^\star) \right) \Delta$$

# Newton's Method

- Taylor's theorem around $w$

$$L(w + \Delta) \approx L(w) + \nabla L(w) \cdot \Delta + \frac{1}{2}\Delta^\top \big(\nabla^2 L(w^\star)\big)\Delta$$

- First, how do set the stepwise for GD?

$$\eta \approx \lambda_{max}\left(\nabla^2 L(w^\star)\right)$$

# Newton's Method

- Taylor's theorem around $w$

$$L(w+\Delta) \approx L(w) + \nabla L(w)\cdot\Delta + \frac{1}{2}\Delta^\top\left(\nabla^2 L(w^\star)\right)\Delta$$

- First, how do set the stepwise for GD?
- Let's try to update $w$ so as to minimize the RHS:

$$w \leftarrow w - ??$$

$\Delta$

$$\Delta = -\underbrace{\left(\nabla^2 L(w^*)\right)^{-1}}_{\mathbb{R}^{d\times d}}\underbrace{\nabla L(w)}_{\in\mathbb{R}^d}$$

# Newton's Method

- Taylor's theorem around $w$

$$L(w + \Delta) \approx L(w) + \nabla L(w) \cdot \Delta + \frac{1}{2}\Delta^\top \Big( \nabla^2 L(w^\star) \Big) \Delta$$

- First, how do set the stepwise for GD?
- Let's try to update $w$ so as to minimize the RHS:

$$w \leftarrow w - ??$$

- For quadratics, what happens with one step of Newton's method?

# Newton's Method

- Taylor's theorem around $w$

$$L(w+\Delta) \approx L(w) + \nabla L(w) \cdot \Delta + \frac{1}{2}\Delta^{\top}\Big(\nabla^2 L(w^{\star})\Big)\Delta$$

- First, how do set the stepwise for GD?
- Let's try to update $w$ so as to minimize the RHS:

$$w \leftarrow w - ??$$

- For quadratics, what happens with one step of Newton's method?

- More generally, Newton's method and variants (like nonlinear conjugate gradient) are "very very good".

# SGD

# SGD

- Suppose we can get unbiased estimates $\widehat{\nabla L(w)}$ of $\nabla L(w)$. SGD: $w_{t+1} = w_t - \eta_t \widehat{\nabla L(w_t)}$

# SGD

- Suppose we can get unbiased estimates $\widehat{\nabla L(w)}$ of $\nabla L(w)$. SGD:
  $w_{t+1} = w_t - \eta_t \widehat{\nabla L(w_t)}$
- What conditions might we want our stepsizes to satisfy?

$\widehat{\nabla L(w)} = \nabla L(w) + \varepsilon$

# SGD

- Suppose we can get unbiased estimates $\widehat{\nabla L(w)}$ of $\nabla L(w)$. SGD: $w_{t+1} = w_t - \eta_t \widehat{\nabla L(w_t)}$
- What conditions might we want our stepsizes to satisfy?

- Robbins&Monro ('51) showed for convex functions, if $\sum_t \eta_t \to \infty$ and $\sum_t \eta_t^2$ is finite, then $w_t \to w^\star$ as $t \to \infty$.

# SGD

- Suppose we can get unbiased estimates $\widehat{\nabla L(w)}$ of $\nabla L(w)$. SGD:
  $w_{t+1} = w_t - \eta_t \widehat{\nabla L(w_t)}$
- What conditions might we want our stepsizes to satisfy?

- Robbins&Monro ('51) showed for convex functions,
  if $\sum_t \eta_t \to \infty$ and $\sum_t \eta_t^2$ is finite, then $w_t \to w^\star$ as $t \to \infty$.

- Example: suppose $L(w) = E_{(x,y)\sim D}[(y - w \cdot x)^2]$ and we can sample $(x, y) \sim D$.

# SGD: quadratic case

- Suppose $L(w) = E_{(x,y)\sim D}[(y - w \cdot x)^2]$ and we can sample $(x, y) \sim D$.
  $w_{t+1} = w_t + \eta_t(y_t - w_t \cdot x_t)x_t$

# SGD: quadratic case

- Suppose $L(w) = E_{(x,y)\sim D}[(y - w \cdot x)^2]$ and we can sample $(x, y) \sim D$.
  $w_{t+1} = w_t + \eta_t(y_t - w_t \cdot x_t)x_t$

- How do we optimally set $\eta_t$ for the quadratic case?

# SGD: quadratic case

- Suppose $L(w) = E_{(x,y)\sim D}[(y - w \cdot x)^2]$ and we can sample $(x, y) \sim D$.
  $w_{t+1} = w_t + \eta_t(y_t - w_t \cdot x_t)x_t$

- How do we optimally set $\eta_t$ for the quadratic case?
  - Suppose $y = w^\star \cdot x + \epsilon$, where $\epsilon \sim N(0,\sigma^2)$, and $x$ is Gaussian with covariance $E[xx^\top] = UDU^\top$

# SGD: quadratic case

- Suppose $L(w) = E_{(x,y)\sim D}[(y - w \cdot x)^2]$ and we can sample $(x, y) \sim D$.
  $w_{t+1} = w_t + \eta_t(y_t - w_t \cdot x_t)x_t$

- How do we optimally set $\eta_t$ for the quadratic case?
  - Suppose $y = w^\star \cdot x + \epsilon$, where $\epsilon \sim N(0,\sigma^2)$, and $x$ is Gaussian with covariance $E[xx^\top] = UDU^\top$
  - It is subtle, even for this case!

# SGD: quadratic case

- Suppose $L(w) = E_{(x,y)\sim D}[(y - w \cdot x)^2]$ and we can sample $(x, y) \sim D$.
  $w_{t+1} = w_t + \eta_t(y_t - w_t \cdot x_t)x_t$

- How do we optimally set $\eta_t$ for the quadratic case?
  - Suppose $y = w^\star \cdot x + \epsilon$, where $\epsilon \sim N(0,\sigma^2)$, and $x$ is Gaussian with covariance $E[xx^\top] = UDU^\top$
  - It is subtle, even for this case!

- The optimal achievable rate over all algorithms is: $E[L(w_t) - L(w^\star)] \geq \dfrac{d\sigma^2}{t}$

# SGD: quadratic case

- Suppose $L(w) = E_{(x,y)\sim D}[(y - w \cdot x)^2]$ and we can sample $(x, y) \sim D$.

  $w_{t+1} = w_t + \eta_t(y_t - w_t \cdot x_t)x_t$

- How do we optimally set $\eta_t$ for the quadratic case?
  - Suppose $y = w^\star \cdot x + \epsilon$, where $\epsilon \sim N(0,\sigma^2)$, and $x$ is Gaussian with covariance $E[xx^\top] = UDU^\top$
  - It is subtle, even for this case!

- The optimal achievable rate over all algorithms is: $E[L(w_t) - L(w^\star)] \geq \frac{d\sigma^2}{t}$

- "classical" lr decay schedule (like $\eta_t = 1/t^\alpha$), have converge rates like which are condition number worse.

  $E[L(w_t) - L(w^\star)] \lesssim \kappa \frac{d\sigma^2}{t}$, for large $t$

# SGD: quadratic case

- Suppose $L(w) = E_{(x,y)\sim D}[(y - w \cdot x)^2]$ and we can sample $(x, y) \sim D$.
  $w_{t+1} = w_t + \eta_t(y_t - w_t \cdot x_t)x_t$

- How do we optimally set $\eta_t$ for the quadratic case?
  - Suppose $y = w^\star \cdot x + \epsilon$, where $\epsilon \sim N(0,\sigma^2)$, and $x$ is Gaussian with covariance $E[xx^\top] = UDU^\top$
  - It is subtle, even for this case!

- The optimal achievable rate over all algorithms is: $E[L(w_t) - L(w^\star)] \geq \frac{d\sigma^2}{t}$

- "classical" lr decay schedule (like $\eta_t = 1/t^\alpha$), have converge rates like which are condition number worse.
  $E[L(w_t) - L(w^\star)] \lesssim \kappa\frac{d\sigma^2}{t}$, for large $t$

- "geometric+piecewise" decay is near opt, though $\log(\kappa)$ worse:
  $E[L(w_t) - L(w^\star)] \lesssim \log(\kappa)\frac{d\sigma^2}{t}$, for large $t$

# SGD: quadratic case

- Suppose $L(w) = E_{(x,y)\sim D}[(y - w \cdot x)^2]$ and we can sample $(x, y) \sim D$.

  $w_{t+1} = w_t + \eta_t(y_t - w_t \cdot x_t)x_t$

- How do we optimally set $\eta_t$ for the quadratic case?
  - Suppose $y = w^\star \cdot x + \epsilon$, where $\epsilon \sim N(0,\sigma^2)$, and $x$ is Gaussian with covariance $E[xx^\top] = UDU^\top$
  - It is subtle, even for this case!

- The optimal achievable rate over all algorithms is: $E[L(w_t) - L(w^\star)] \geq \dfrac{d\sigma^2}{t}$

- "classical" lr decay schedule (like $\eta_t = 1/t^\alpha$), have converge rates like which are condition number worse.

  $E[L(w_t) - L(w^\star)] \lesssim \kappa \dfrac{d\sigma^2}{t}$, for large $t$

- "geometric+piecewise" decay is near opt, though $\log(\kappa)$ worse:

  $E[L(w_t) - L(w^\star)] \lesssim \log(\kappa) \dfrac{d\sigma^2}{t}$, for large $t$

  - optimal (eigencurve) decay looks like cosine!
    (and no $\log(\kappa)$ under certain spectrum conditions)

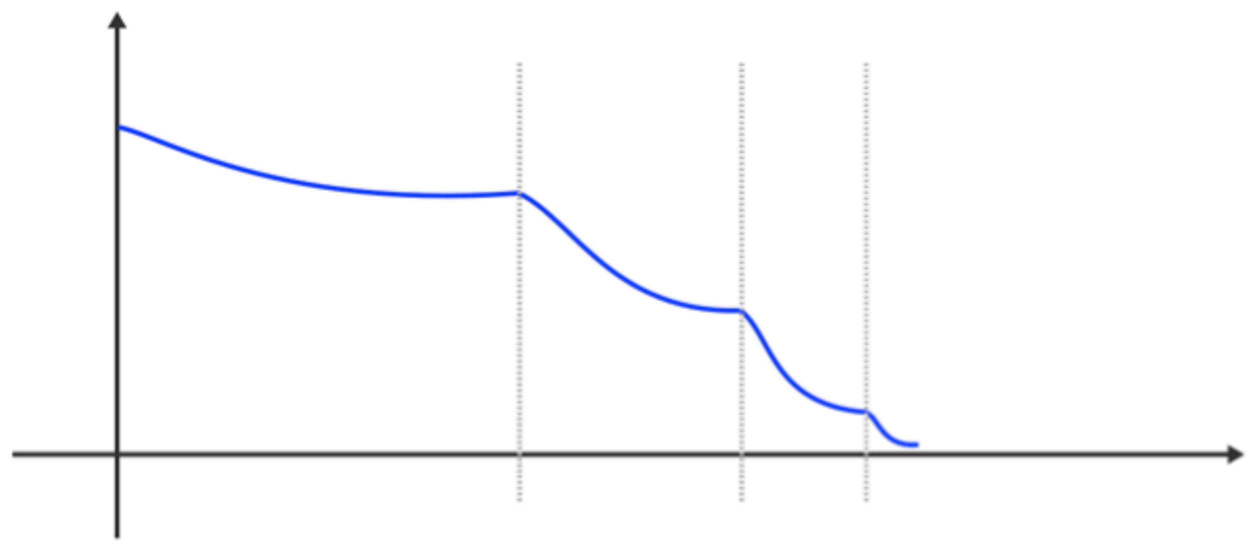


Figure 1: Eigencurve : piecewise inverse time decay scheduling.

# SGD: quadratic case

- Suppose $L(w) = E_{(x,y)\sim D}[(y - w \cdot x)^2]$ and we can sample $(x, y) \sim D$.
  $w_{t+1} = w_t + \eta_t(y_t - w_t \cdot x_t)x_t$

- How do we optimally set $\eta_t$ for the quadratic case?
  - Suppose $y = w^\star \cdot x + \epsilon$, where $\epsilon \sim N(0,\sigma^2)$, and $x$ is Gaussian with covariance $E[xx^\top] = UDU^\top$
  - It is subtle, even for this case!

- The optimal achievable rate over all algorithms is: $E[L(w_t) - L(w^\star)] \geq \frac{d\sigma^2}{t}$

- "classical" lr decay schedule (like $\eta_t = 1/t^\alpha$), have converge rates like which are condition number worse.
  $E[L(w_t) - L(w^\star)] \lesssim \kappa \frac{d\sigma^2}{t}$, for large $t$

- "geometric+piecewise" decay is near opt, though $\log(\kappa)$ worse:
  $E[L(w_t) - L(w^\star)] \lesssim \log(\kappa)\frac{d\sigma^2}{t}$, for large $t$
  - optimal (eigencurve) decay looks like cosine!
    (and no $\log(\kappa)$ under certain spectrum conditions)
  - note: need to know the end time $t$ for setting the the decay

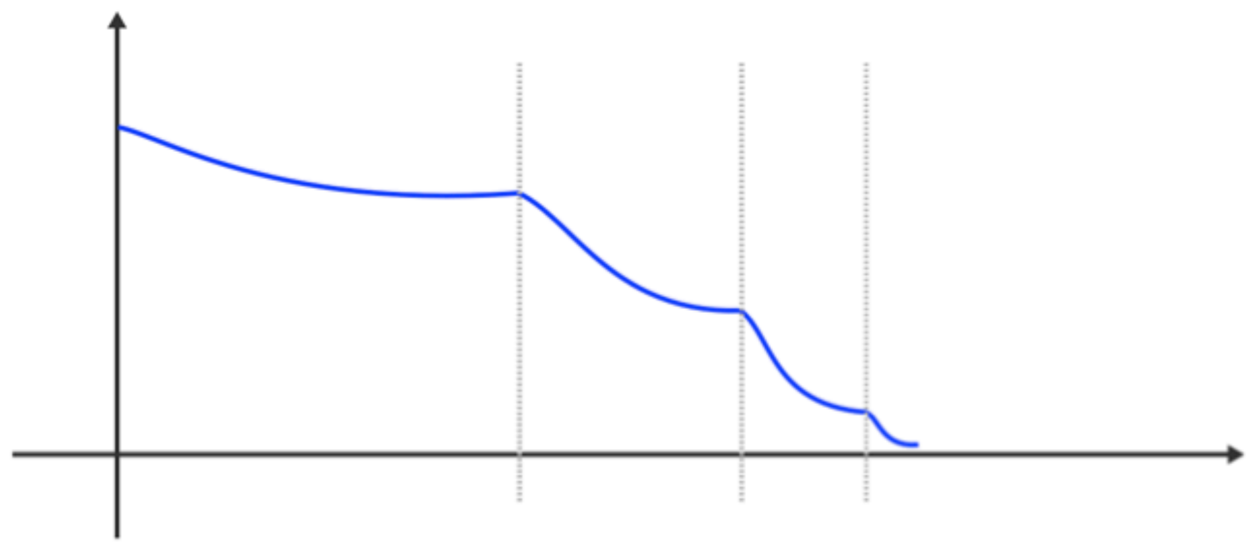


Figure 1: Eigencurve : piecewise inverse time decay scheduling.

# SGD: iterative averaging

- Suppose $y = w^\star \cdot x + \epsilon$, where $\epsilon \sim N(0,\sigma^2)$, and $x$ is Gaussian with covariance $E[xx^\top] = UDU^\top$

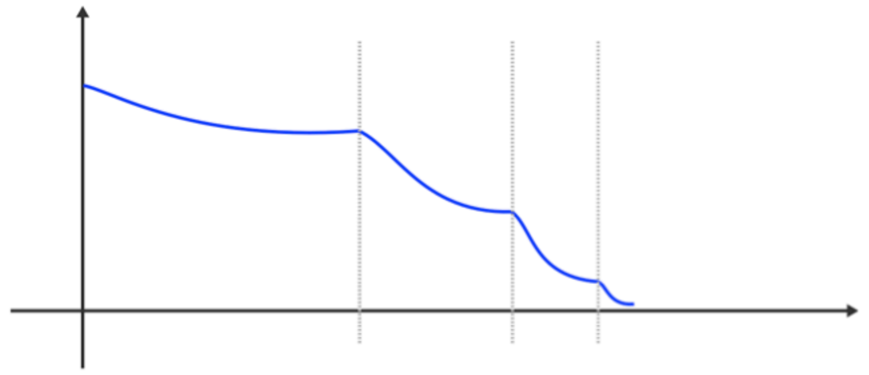


Figure 1: Eigencurve : piecewise inverse time decay scheduling.

# SGD: iterative averaging

- Suppose $y = w^\star \cdot x + \epsilon$, where $\epsilon \sim N(0, \sigma^2)$, and $x$ is Gaussian with covariance $E[xx^\top] = UDU^\top$
- The optimal achievable rate over all algorithms is: $E[L(w_t) - L(w^\star)] \geq \frac{d\sigma^2}{t}$

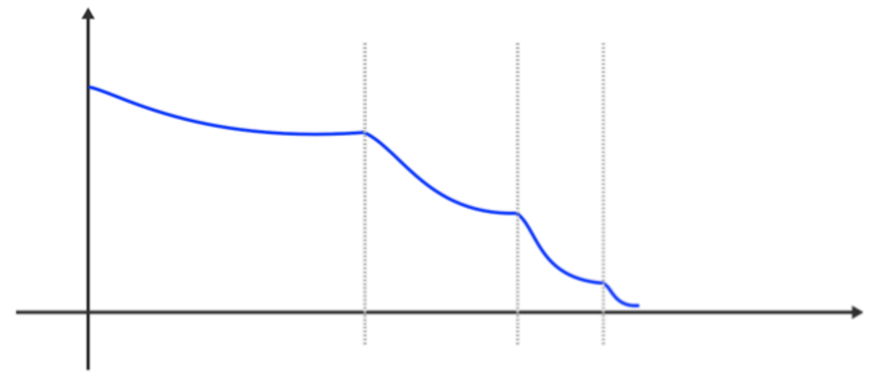


Figure 1: `Eigencurve` : piecewise inverse time decay scheduling.

# SGD: iterative averaging

- Suppose $y = w^\star \cdot x + \epsilon$, where $\epsilon \sim N(0,\sigma^2)$, and $x$ is Gaussian with covariance $E[xx^\top] = UDU^\top$
- The optimal achievable rate over all algorithms is: $E[L(w_t) - L(w^\star)] \geq \dfrac{d\sigma^2}{t}$
- Iterate (tail) averaging, with constant LR, obtains the optimal rate:
  Algo:

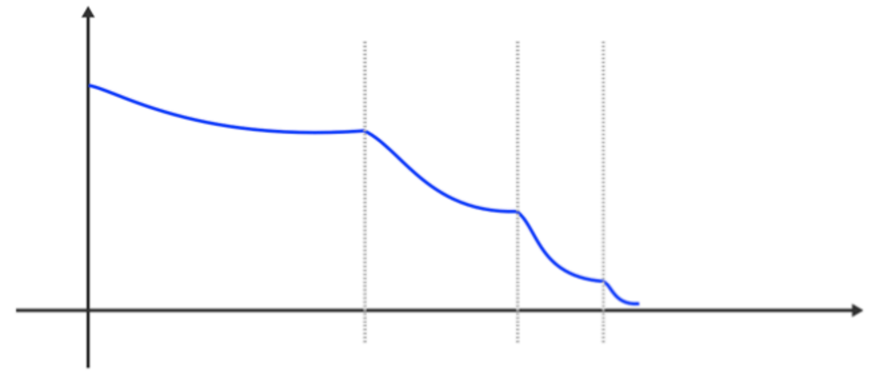


Figure 1: `Eigencurve` : piecewise inverse time decay scheduling.

# SGD: iterative averaging

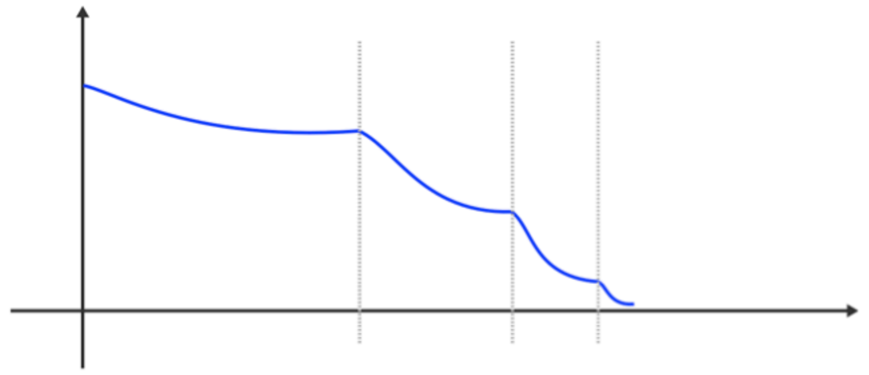


Figure 1: `Eigencurve` : piecewise inverse time decay scheduling.

- Suppose $y = w^\star \cdot x + \epsilon$, where $\epsilon \sim N(0,\sigma^2)$, and $x$ is Gaussian with covariance $E[xx^\top] = UDU^\top$
- The optimal achievable rate over all algorithms is: $E[L(w_t) - L(w^\star)] \geq \dfrac{d\sigma^2}{t}$
- Iterate (tail) averaging, with constant LR, obtains the optimal rate:
  Algo:
  - Run SGD with constant LR.

# SGD: iterative averaging

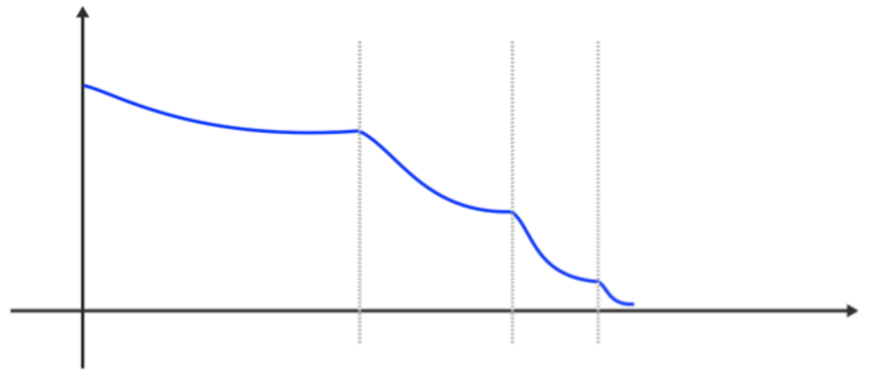


Figure 1: `Eigencurve` : piecewise inverse time decay scheduling.

- Suppose $y = w^\star \cdot x + \epsilon$, where $\epsilon \sim N(0, \sigma^2)$, and $x$ is Gaussian with covariance $E[xx^\top] = UDU^\top$
- The optimal achievable rate over all algorithms is: $E[L(w_t) - L(w^\star)] \geq \dfrac{d\sigma^2}{t}$
- Iterate (tail) averaging, with constant LR, obtains the optimal rate:
  Algo:
  - Run SGD with constant LR.
  - Return the average over the last half: $\widehat{w} = \dfrac{1}{T/2} \sum_{t=T/2}^{T} w_t$

# SGD: iterative averaging

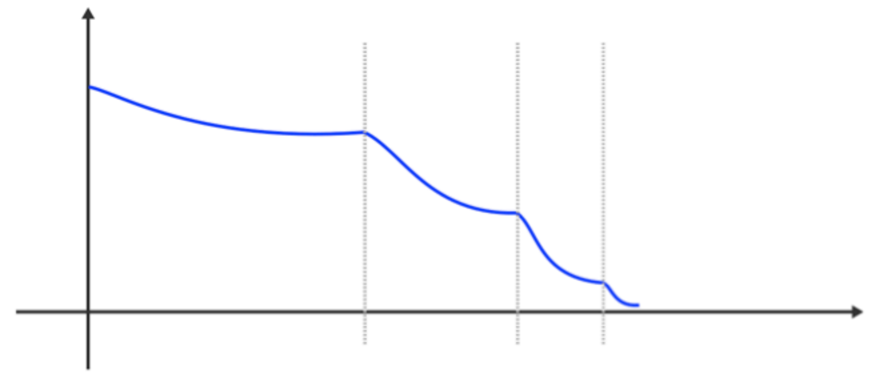


Figure 1: `Eigencurve` : piecewise inverse time decay scheduling.

- Suppose $y = w^\star \cdot x + \epsilon$, where $\epsilon \sim N(0,\sigma^2)$, and $x$ is Gaussian with covariance $E[xx^\top] = UDU^\top$
- The optimal achievable rate over all algorithms is: $E[L(w_t) - L(w^\star)] \geq \frac{d\sigma^2}{t}$
- Iterate (tail) averaging, with constant LR, obtains the optimal rate:
  Algo:
  - Run SGD with constant LR.
  - Return the average over the last half: $\widehat{w} = \frac{1}{T/2} \sum_{t=T/2}^{T} w_t$
- For more general convex case, the Polyak&Juditksy ('92) showed that both (a) decaying the LR and (b) integrate averaging obtains the optimal rate.

# SGD: iterative averaging

- Suppose $y = w^\star \cdot x + \epsilon$, where $\epsilon \sim N(0, \sigma^2)$, and $x$ is Gaussian with covariance $E[xx^\top] = UDU^\top$
- The optimal achievable rate over all algorithms is: $E[L(w_t) - L(w^\star)] \geq \dfrac{d\sigma^2}{t}$
- Iterate (tail) averaging, with constant LR, obtains the optimal rate:
  Algo:
  - Run SGD with constant LR.
  - Return the average over the last half: $\widehat{w} = \dfrac{1}{T/2} \sum_{t=T/2}^{T} w_t$
- For more general convex case, the Polyak&Juditksy ('92) showed that both (a) decaying the LR and (b) integrate averaging obtains the optimal rate.
- Also, exponential weight averaging (EWA) essentially the same:

$\widehat{w} =$

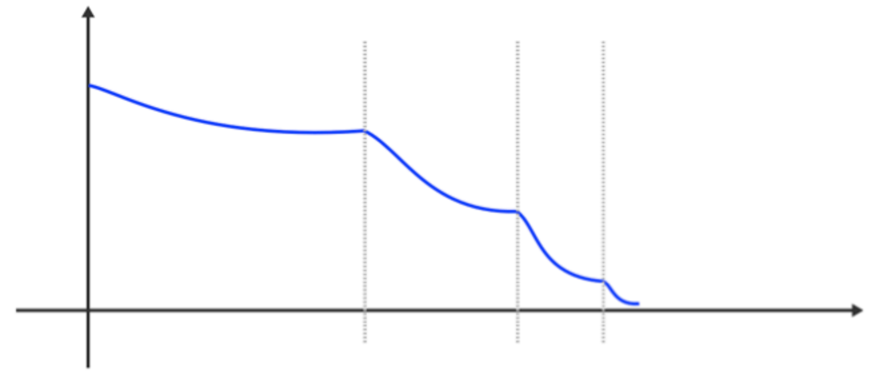


Figure 1: Eigencurve : piecewise inverse time decay scheduling.

# DL

# Today

- Announcements/Recap++
- Whirlwind Tour of Optimization
- ✓ DL Optimization Pipeline
  - Optimizers/Adam
  - Stability/Architecture Modifications
  - Learning rate scheduling/Batch size
  - Scaling Laws…
- Training Dynamics/Edge of stability

# Adam Algo

**Algorithm 1:** *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. $g_t^2$ indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With $\beta_1^t$ and $\beta_2^t$ we denote $\beta_1$ and $\beta_2$ to the power $t$.

**Require:** $\alpha$: Stepsize
**Require:** $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
**Require:** $f(\theta)$: Stochastic objective function with parameters $\theta$
**Require:** $\theta_0$: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  **while** $\theta_t$ not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  **end while**
  **return** $\theta_t$ (Resulting parameters)

Kingma and Ba 2014

# Adam Algo

Lots of works on “Why Adam works?”, but no consensus:

---

**Algorithm 1:** *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. $g_t^2$ indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With $\beta_1^t$ and $\beta_2^t$ we denote $\beta_1$ and $\beta_2$ to the power $t$.

---

**Require:** $\alpha$: Stepsize
**Require:** $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
**Require:** $f(\theta)$: Stochastic objective function with parameters $\theta$
**Require:** $\theta_0$: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  **while** $\theta_t$ not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  **end while**
  **return** $\theta_t$ (Resulting parameters)

---

Kingma and Ba 2014

# Adam Algo

**Algorithm 1:** *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. $g_t^2$ indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With $\beta_1^t$ and $\beta_2^t$ we denote $\beta_1$ and $\beta_2$ to the power $t$.

**Require:** $\alpha$: Stepsize
**Require:** $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
**Require:** $f(\theta)$: Stochastic objective function with parameters $\theta$
**Require:** $\theta_0$: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1$^{\text{st}}$ moment vector)
  $v_0 \leftarrow 0$ (Initialize 2$^{\text{nd}}$ moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  **while** $\theta_t$ not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  **end while**
  **return** $\theta_t$ (Resulting parameters)

Kingma and Ba 2014

Lots of works on “Why Adam works?”, but no consensus:

- Derived from Adagrad, an online learning method.

# Adam Algo

**Algorithm 1:** *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. $g_t^2$ indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With $\beta_1^t$ and $\beta_2^t$ we denote $\beta_1$ and $\beta_2$ to the power $t$.

**Require:** $\alpha$: Stepsize
**Require:** $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
**Require:** $f(\theta)$: Stochastic objective function with parameters $\theta$
**Require:** $\theta_0$: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize $1^{\text{st}}$ moment vector)
  $v_0 \leftarrow 0$ (Initialize $2^{\text{nd}}$ moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  **while** $\theta_t$ not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  **end while**
  **return** $\theta_t$ (Resulting parameters)

Kingma and Ba 2014

Lots of works on “Why Adam works?”, but no consensus:

- Derived from Adagrad, an online learning method.
- At $\beta_1 = \beta_2 = \epsilon = 0$, it becomes signed gradient descent, this connection is used in a lot of theoretical analysis of Adam.

# Adam Algo

**Algorithm 1:** *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. $g_t^2$ indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With $\beta_1^t$ and $\beta_2^t$ we denote $\beta_1$ and $\beta_2$ to the power $t$.

**Require:** $\alpha$: Stepsize
**Require:** $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
**Require:** $f(\theta)$: Stochastic objective function with parameters $\theta$
**Require:** $\theta_0$: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1$^{\text{st}}$ moment vector)
  $v_0 \leftarrow 0$ (Initialize 2$^{\text{nd}}$ moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  **while** $\theta_t$ not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t/(1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t/(1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t/(\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  **end while**
  **return** $\theta_t$ (Resulting parameters)

Kingma and Ba 2014

Lots of works on “Why Adam works?”, but no consensus:

- Derived from Adagrad, an online learning method.
- At $\beta_1 = \beta_2 = \epsilon = 0$, it becomes signed gradient descent, this connection is used in a lot of theoretical analysis of Adam.
- Also has connections to Newton’s method.

# Adam Algo

---

**Algorithm 1:** *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. $g_t^2$ indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With $\beta_1^t$ and $\beta_2^t$ we denote $\beta_1$ and $\beta_2$ to the power $t$.

---

**Require:** $\alpha$: Stepsize
**Require:** $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
**Require:** $f(\theta)$: Stochastic objective function with parameters $\theta$
**Require:** $\theta_0$: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1$^{\text{st}}$ moment vector)
  $v_0 \leftarrow 0$ (Initialize 2$^{\text{nd}}$ moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  **while** $\theta_t$ not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  **end while**
  **return** $\theta_t$ (Resulting parameters)

---

Kingma and Ba 2014

Lots of works on “Why Adam works?”, but no consensus:

- Derived from Adagrad, an online learning method.
- At $\beta_1 = \beta_2 = \epsilon = 0$, it becomes signed gradient descent, this connection is used in a lot of theoretical analysis of Adam.
- Also has connections to Newton’s method.

- Modern transformers need Adam, they cannot be trained with SGD.

# Adam Algo

**Algorithm 1:** *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. $g_t^2$ indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With $\beta_1^t$ and $\beta_2^t$ we denote $\beta_1$ and $\beta_2$ to the power $t$.

**Require:** $\alpha$: Stepsize
**Require:** $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
**Require:** $f(\theta)$: Stochastic objective function with parameters $\theta$
**Require:** $\theta_0$: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize $1^{\text{st}}$ moment vector)
  $v_0 \leftarrow 0$ (Initialize $2^{\text{nd}}$ moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  **while** $\theta_t$ not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  **end while**
  **return** $\theta_t$ (Resulting parameters)

Kingma and Ba 2014

Lots of works on “Why Adam works?”, but no consensus:

- Derived from Adagrad, an online learning method.
- At $\beta_1 = \beta_2 = \epsilon = 0$, it becomes signed gradient descent, this connection is used in a lot of theoretical analysis of Adam.
- Also has connections to Newton’s method.

- Modern transformers need Adam, they cannot be trained with SGD.

Changes:

# Adam Algo

**Algorithm 1:** *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. $g_t^2$ indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With $\beta_1^t$ and $\beta_2^t$ we denote $\beta_1$ and $\beta_2$ to the power $t$.

**Require:** $\alpha$: Stepsize
**Require:** $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
**Require:** $f(\theta)$: Stochastic objective function with parameters $\theta$
**Require:** $\theta_0$: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1$^{\text{st}}$ moment vector)
  $v_0 \leftarrow 0$ (Initialize 2$^{\text{nd}}$ moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  **while** $\theta_t$ not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  **end while**
  **return** $\theta_t$ (Resulting parameters)

Kingma and Ba 2014

Lots of works on “Why Adam works?”, but no consensus:

- Derived from Adagrad, an online learning method.
- At $\beta_1 = \beta_2 = \epsilon = 0$, it becomes signed gradient descent, this connection is used in a lot of theoretical analysis of Adam.
- Also has connections to Newton’s method.

- Modern transformers need Adam, they cannot be trained with SGD.

Changes:

- How to apply weight decay?: AdamW

# Adam Algo

**Algorithm 1:** *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. $g_t^2$ indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With $\beta_1^t$ and $\beta_2^t$ we denote $\beta_1$ and $\beta_2$ to the power $t$.

**Require:** $\alpha$: Stepsize
**Require:** $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
**Require:** $f(\theta)$: Stochastic objective function with parameters $\theta$
**Require:** $\theta_0$: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  **while** $\theta_t$ not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  **end while**
  **return** $\theta_t$ (Resulting parameters)

Kingma and Ba 2014

Lots of works on “Why Adam works?”, but no consensus:

- Derived from Adagrad, an online learning method.
- At $\beta_1 = \beta_2 = \epsilon = 0$, it becomes signed gradient descent, this connection is used in a lot of theoretical analysis of Adam.
- Also has connections to Newton’s method.

- Modern transformers need Adam, they cannot be trained with SGD.

Changes:

- How to apply weight decay?: AdamW
- $\beta_2 = .99$ would be a better default.

# Adam Algo

**Algorithm 1:** *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. $g_t^2$ indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With $\beta_1^t$ and $\beta_2^t$ we denote $\beta_1$ and $\beta_2$ to the power $t$.

**Require:** $\alpha$: Stepsize
**Require:** $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
**Require:** $f(\theta)$: Stochastic objective function with parameters $\theta$
**Require:** $\theta_0$: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1$^{\text{st}}$ moment vector)
  $v_0 \leftarrow 0$ (Initialize 2$^{\text{nd}}$ moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  **while** $\theta_t$ not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  **end while**
  **return** $\theta_t$ (Resulting parameters)

Kingma and Ba 2014

Lots of works on "Why Adam works?", but no consensus:

- Derived from Adagrad, an online learning method.
- At $\beta_1 = \beta_2 = \epsilon = 0$, it becomes signed gradient descent, this connection is used in a lot of theoretical analysis of Adam.
- Also has connections to Newton's method.

- Modern transformers need Adam, they cannot be trained with SGD.

Changes:

- How to apply weight decay?: AdamW
- $\beta_2 = .99$ would be a better default.
- For large models, smaller $\epsilon$ values are needed.

# Adam Algo

**Algorithm 1:** *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. $g_t^2$ indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With $\beta_1^t$ and $\beta_2^t$ we denote $\beta_1$ and $\beta_2$ to the power $t$.

**Require:** $\alpha$: Stepsize
**Require:** $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
**Require:** $f(\theta)$: Stochastic objective function with parameters $\theta$
**Require:** $\theta_0$: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  **while** $\theta_t$ not converged **do**
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  **end while**
  **return** $\theta_t$ (Resulting parameters)

Kingma and Ba 2014

Lots of works on "Why Adam works?", but no consensus:

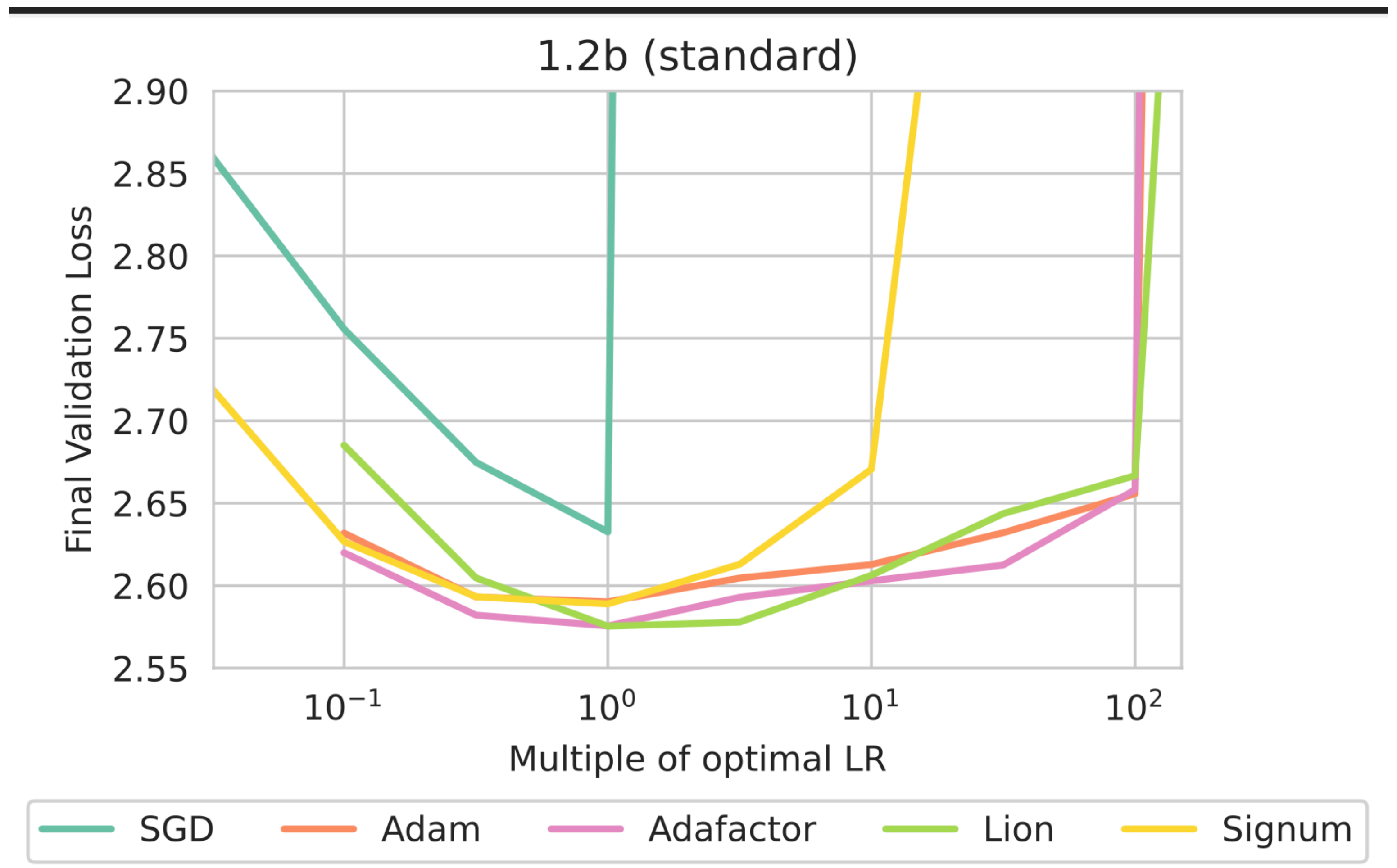
- Derived from Adagrad, an online learning method.
- At $\beta_1 = \beta_2 = \epsilon = 0$, it becomes signed gradient descent, this connection is used in a lot of theoretical analysis of Adam.
- Also has connections to Newton's method.

- Modern transformers need Adam, they cannot be trained with SGD.

Changes:

- How to apply weight decay?: AdamW
- $\beta_2 = .99$ would be a better default.
- For large models, smaller $\epsilon$ values are needed.
- Adafactor, 8bit Adam.

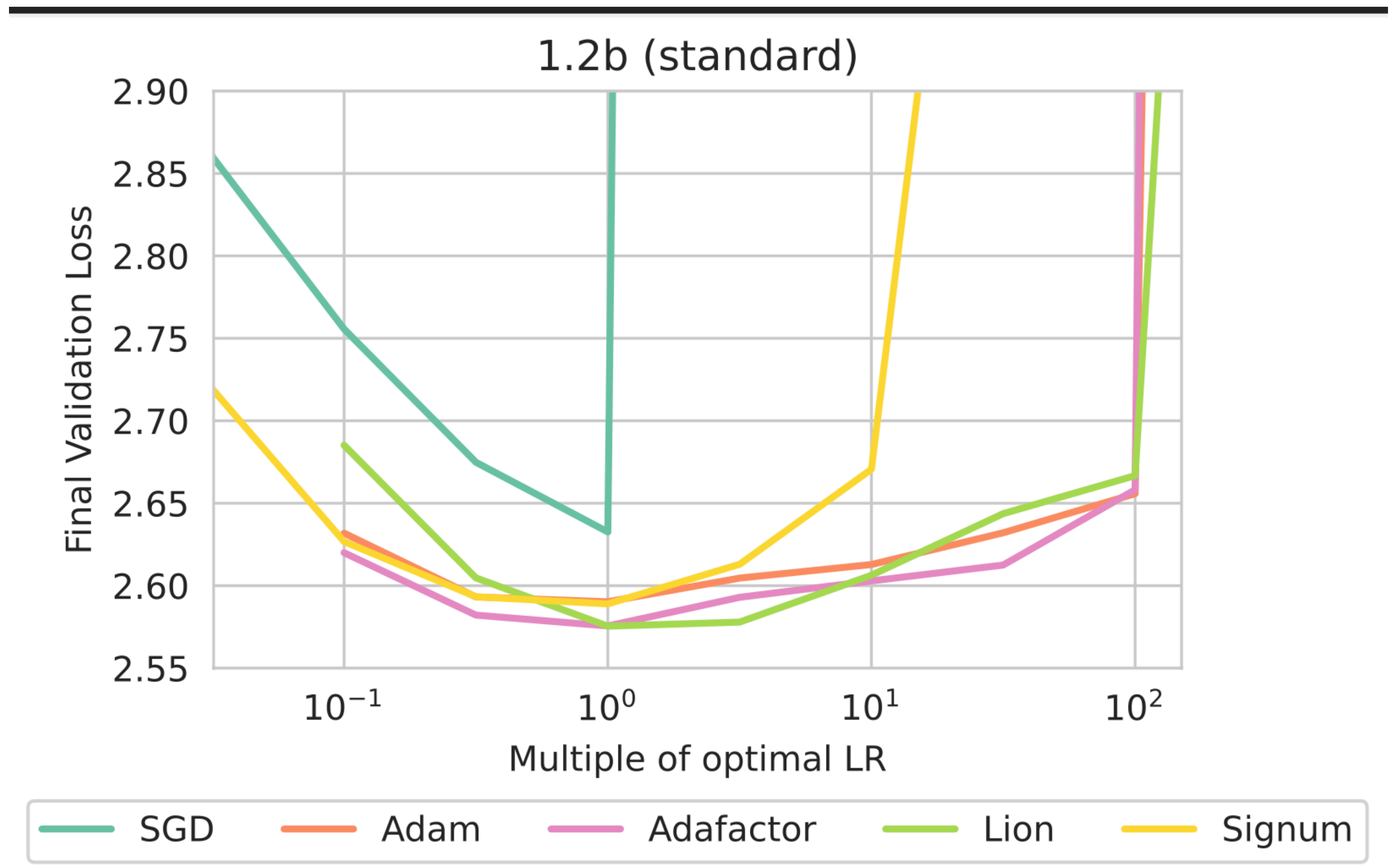
# Other Diagonal Preconditioner Optimizers



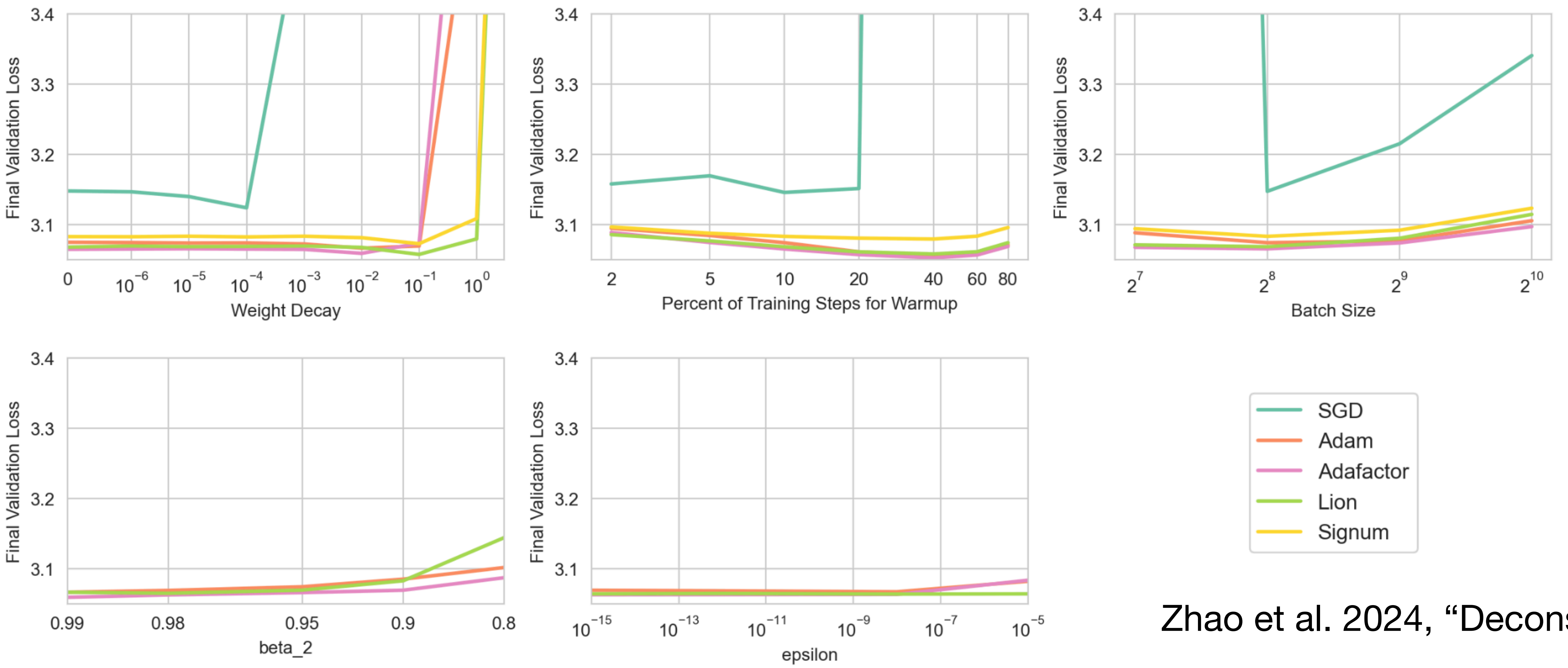

- Many methods such Adam, Adafactor, Lion perform very similarly, except SGD.

Zhao et al. 2024, “Deconstructing What Makes a Good Optimizer for Language Models”

# Other Diagonal Preconditioner Optimizers

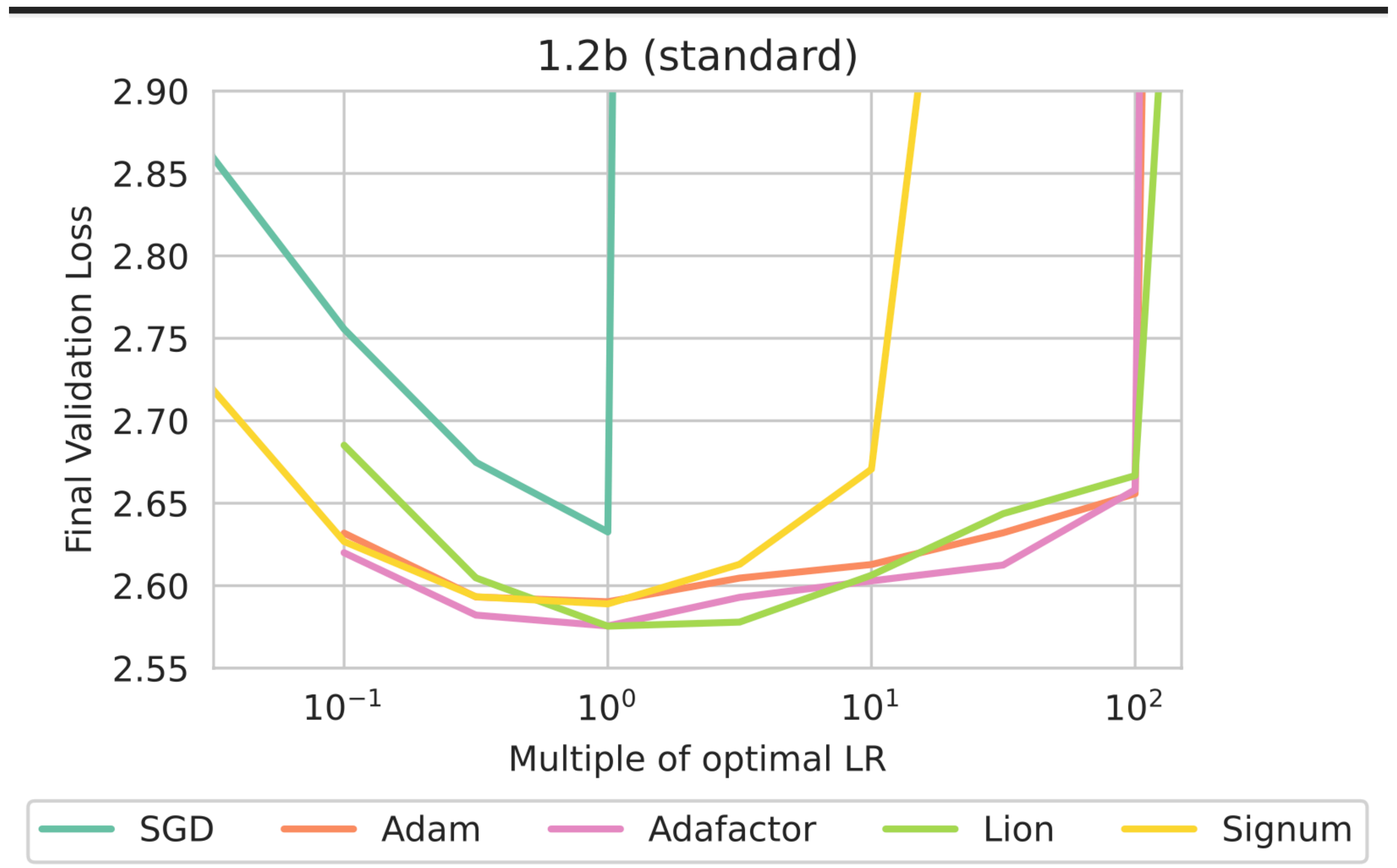



- Many methods such Adam, Adafactor, Lion perform very similarly, except SGD.

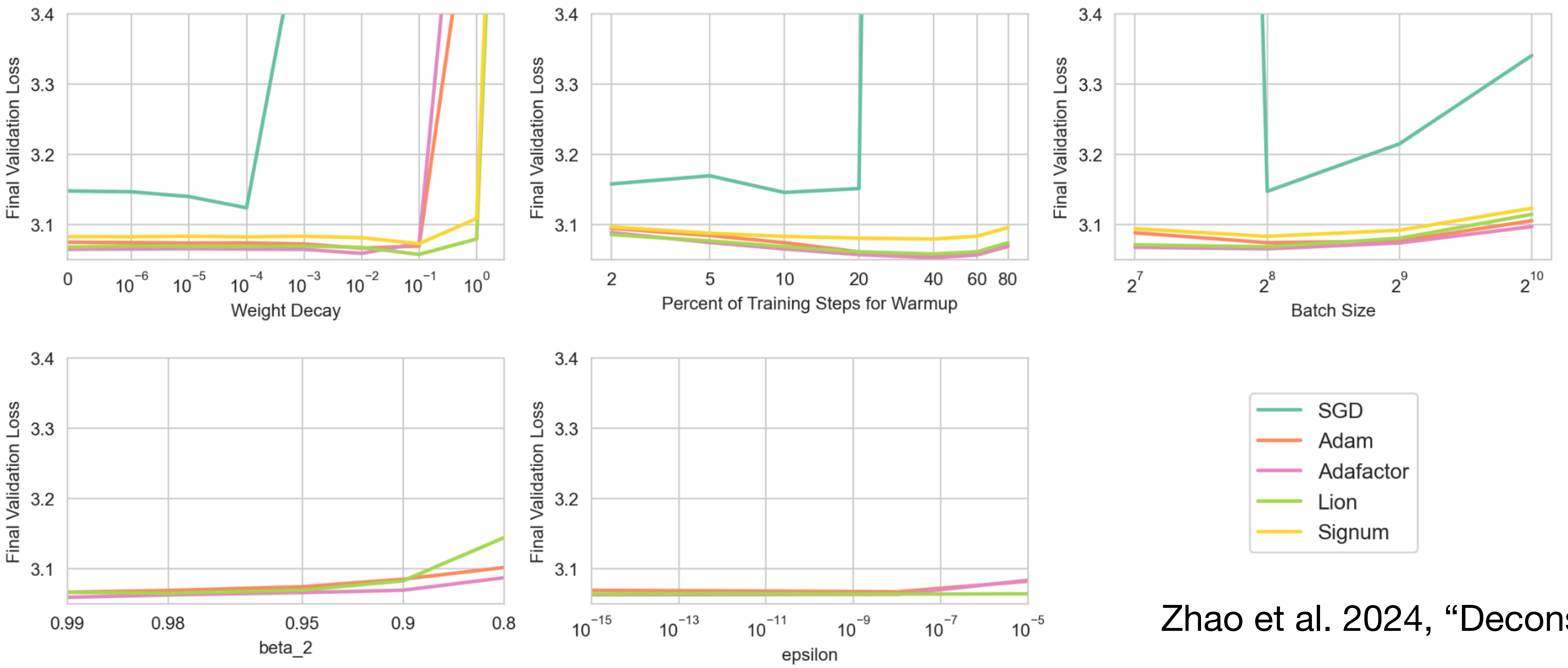



Zhao et al. 2024, “Deconstructing What Makes a Good Optimizer for Language Models”

# Other Diagonal Preconditioner Optimizers




- Many methods such Adam, Adafactor, Lion perform very similarly, except SGD.
- All of these performant optimizers are related to signed gradient descent.
- If optimizer space is a bottleneck, use Adafactor or Lion along with low precision training




Zhao et al. 2024, “Deconstructing What Makes a Good Optimizer for Language Models”

# Today

- Announcements/Recap++
- Whirlwind Tour of Optimization
- DL Optimization Pipeline
  - Optimizers/Adam
  - ✓ Stability/Architecture Modifications
  - Learning rate scheduling/Batch size
  - Scaling Laws…
- Training Dynamics/Edge of stability

# (Sometimes) Loss Spikes

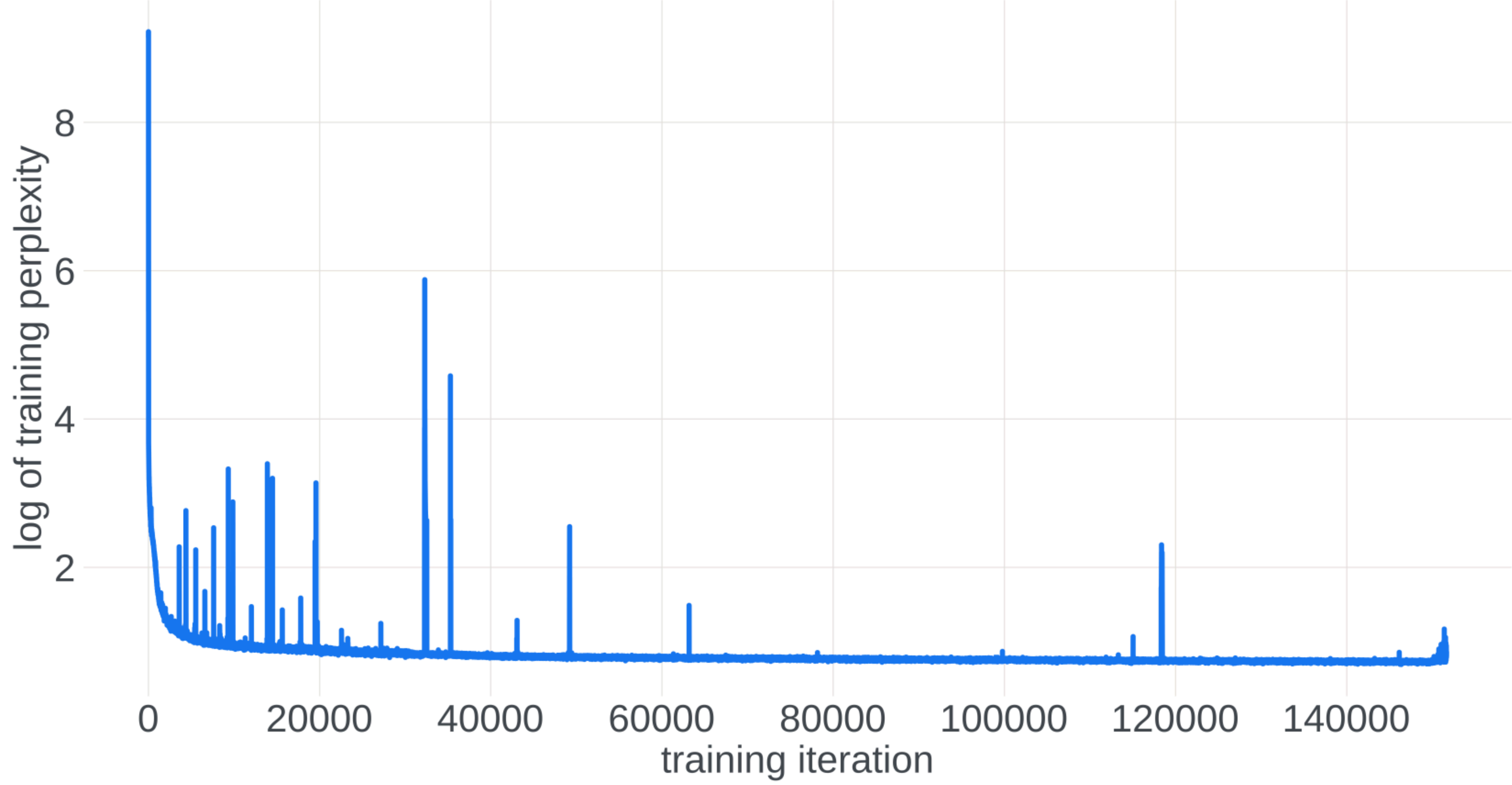



Figure 1: Training perplexity curve of 546b model with prominent spikes

Fig credit: A Theory on Adam Instability in Large-Scale Machine Learning

**PaLM: Scaling Language Modeling with Pathways**

## 5.1 Training Instability

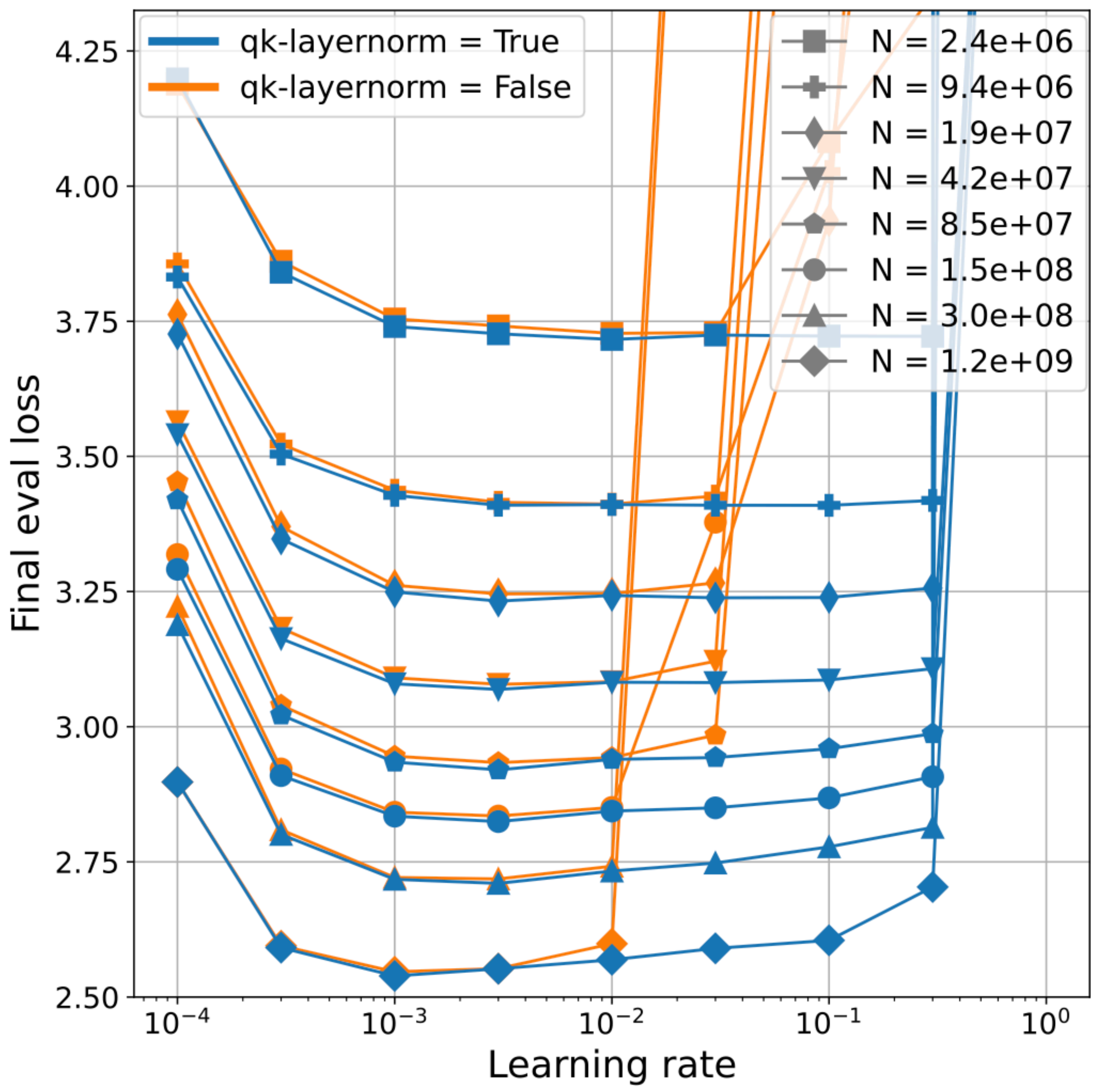
For the largest model, we observed spikes in the loss roughly 20 times during training, despite the fact that gradient clipping was enabled. These spikes occurred at highly irregular intervals, sometimes happening late into training, and were not observed when training the smaller models. Due to the cost of training the largest model, we were not able to determine a principled strategy to mitigate these spikes.

Instead, we found that a simple strategy to effectively mitigate the issue: We re-started training from a checkpoint roughly 100 steps before the spike started, and skipped roughly 200–500 data batches, which cover

# Stability Fixes:

Small-scale proxies for large-scale Transformer training instabilities

Mitchell Wortsman Peter J. Liu Lechao Xiao Katie Everett
Alex Alemi Ben Adlam John D. Co-Reyes Izzeddin Gur Abhishek Kumar
Roman Novak Jeffrey Pennington Jascha Sohl-dickstein Kelvin Xu
Jaehoon Lee* Justin Gilmer* Simon Kornblith*
Google DeepMind



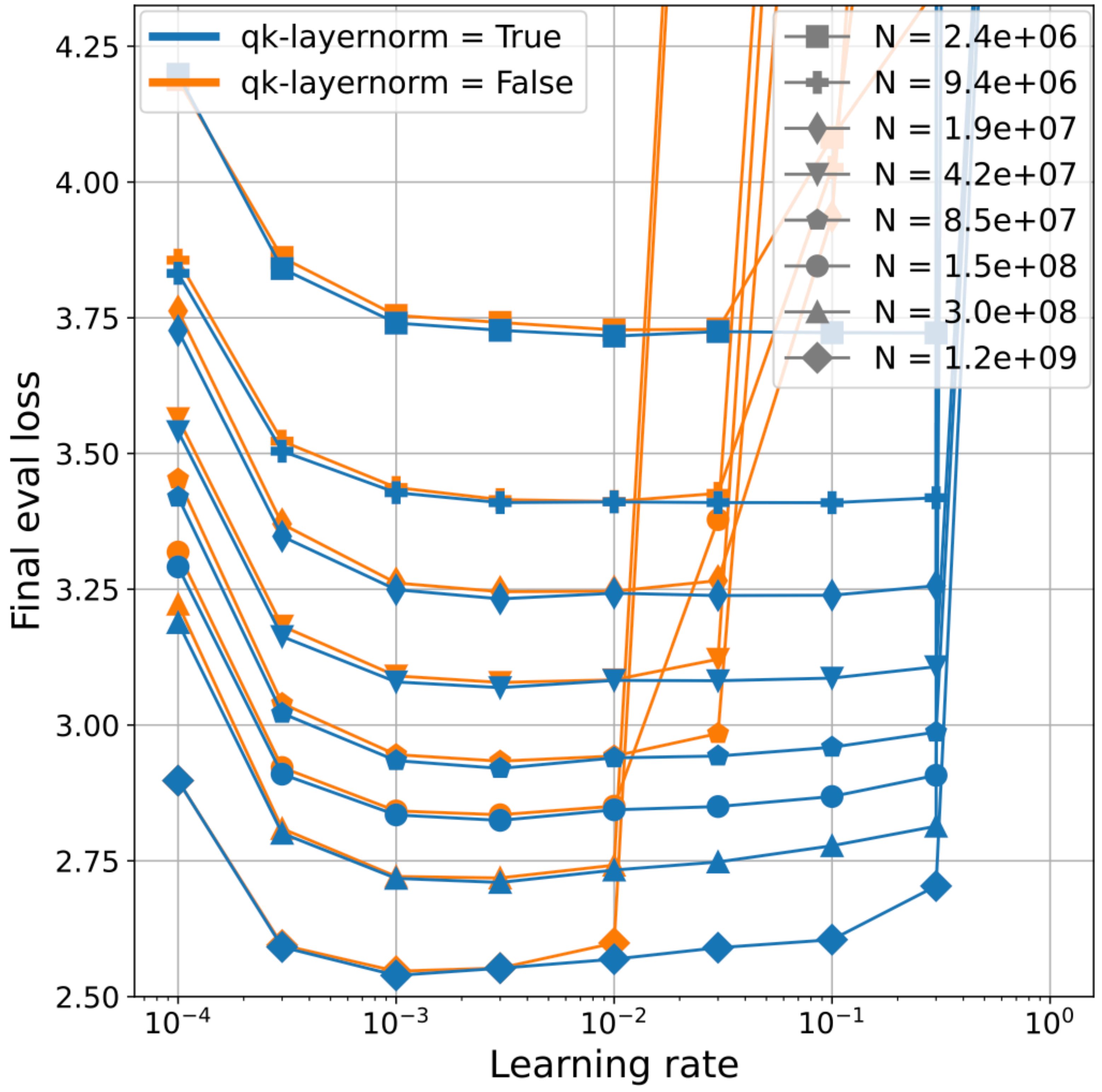

Two important fixes:

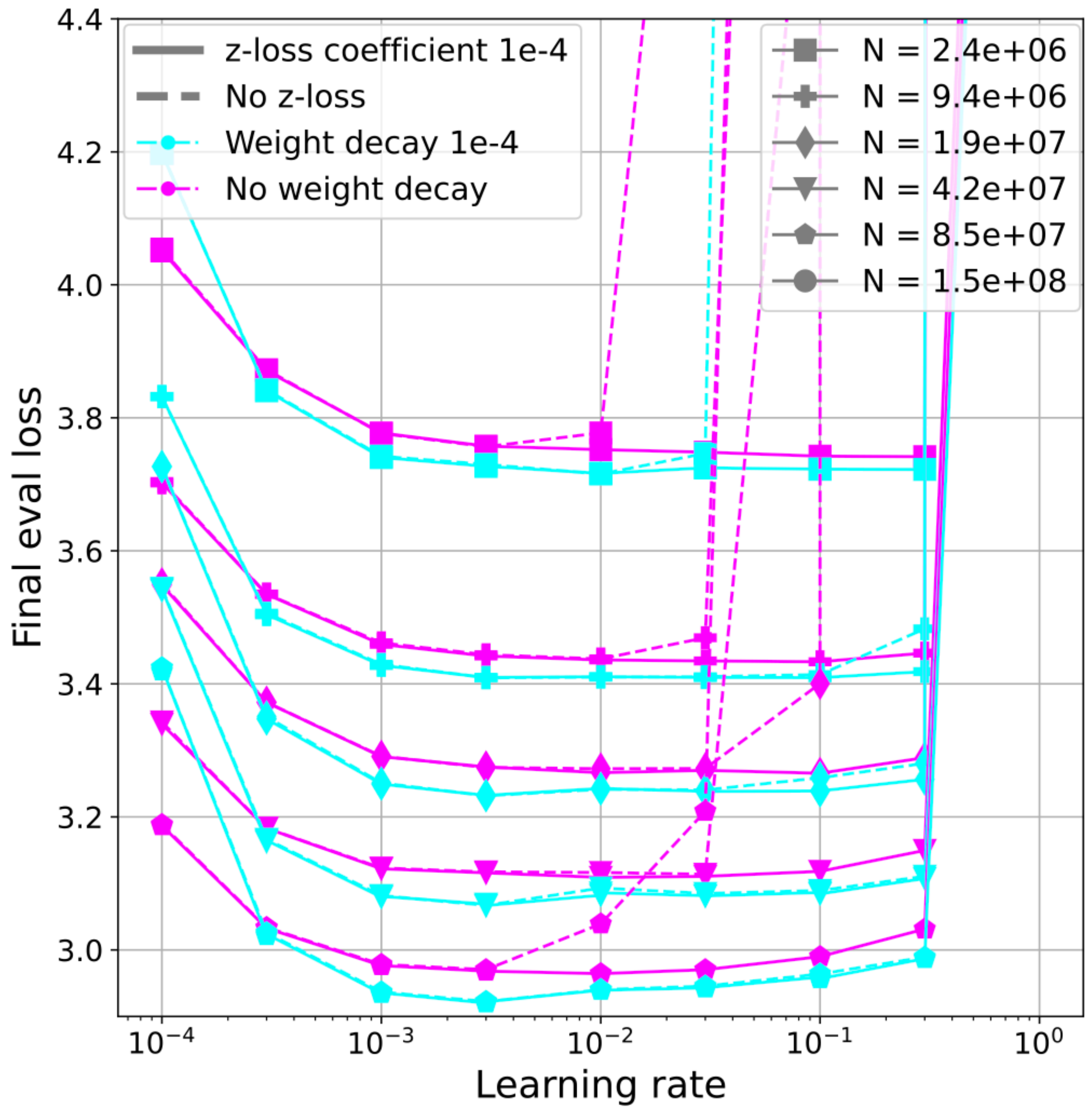
- (Left) Various “layer normalizations”

# Stability Fixes:

Small-scale proxies for large-scale Transformer training instabilities

Mitchell Wortsman Peter J. Liu Lechao Xiao Katie Everett
Alex Alemi Ben Adlam John D. Co-Reyes Izzeddin Gur Abhishek Kumar
Roman Novak Jeffrey Pennington Jascha Sohl-dickstein Kelvin Xu
Jaehoon Lee* Justin Gilmer* Simon Kornblith*
Google DeepMind



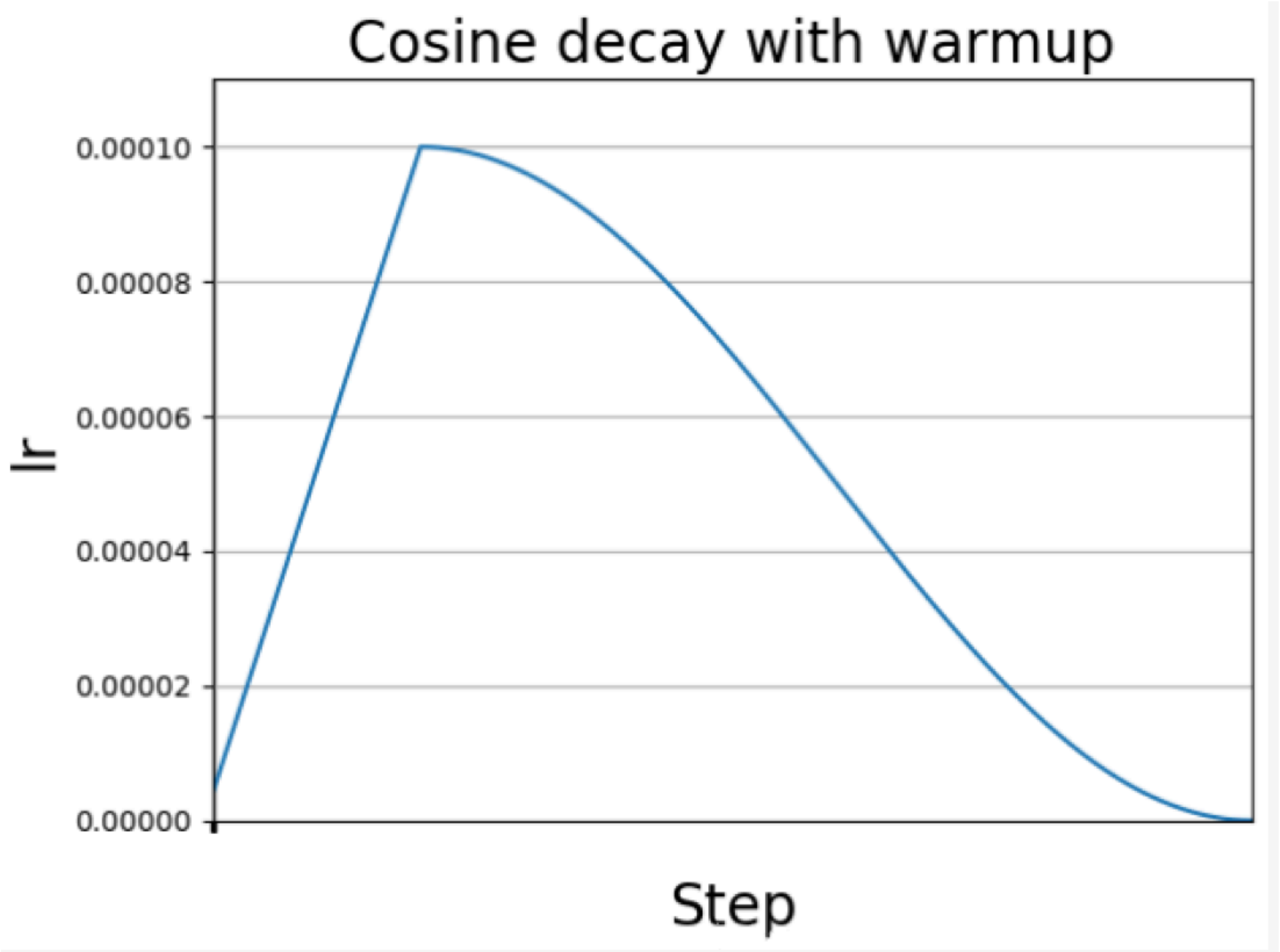

Two important fixes:

- (Left) Various “layer normalizations”
- (Right) Regularization/Weight decay

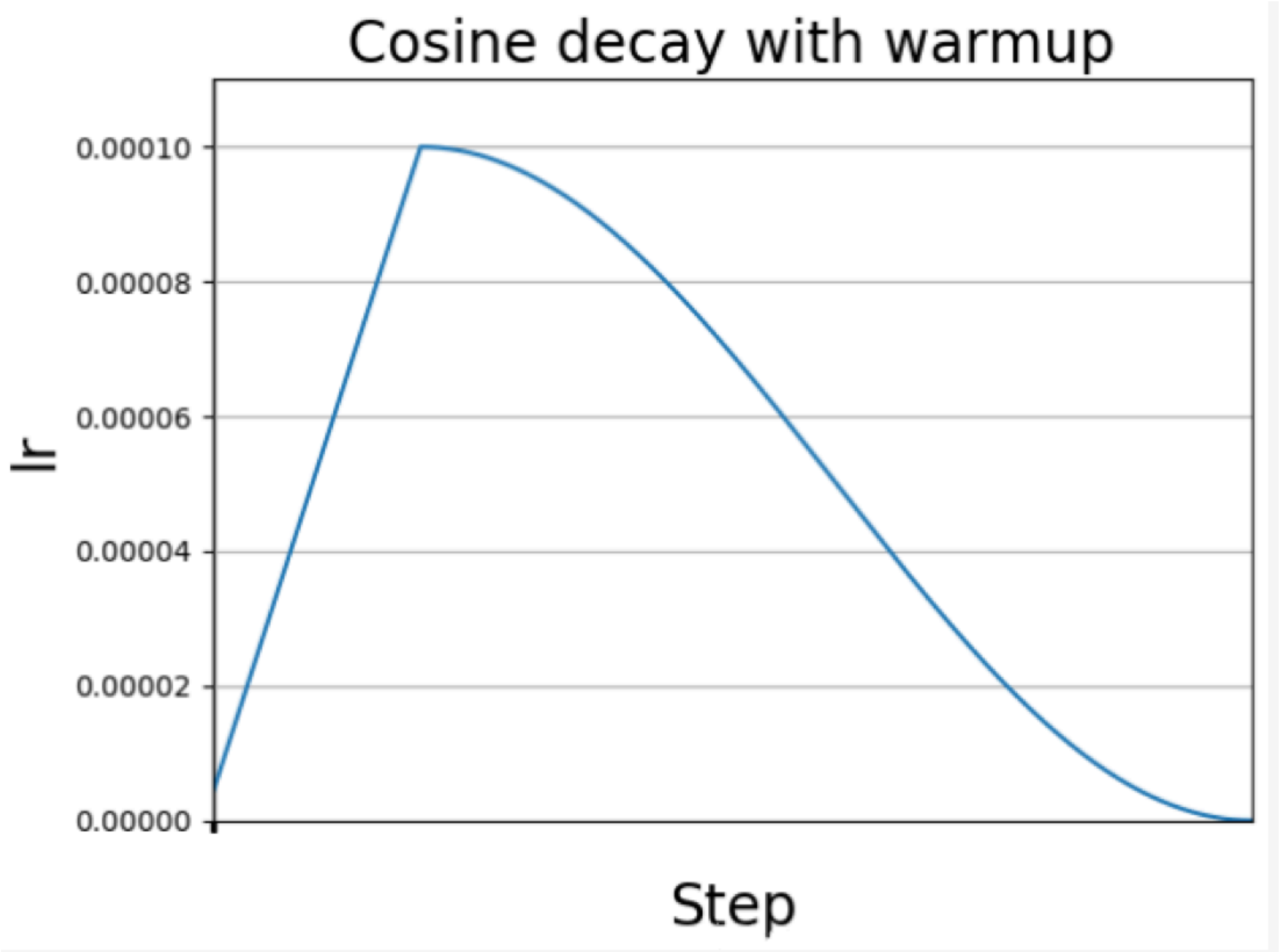
# Today

- Announcements/Recap++
- Whirlwind Tour of Optimization
- DL Optimization Pipeline
  - Optimizers/Adam
  - Stability/Architecture Modifications
  - ✓ Learning rate scheduling/Batch size
  - Scaling Laws…
- Training Dynamics/Edge of stability

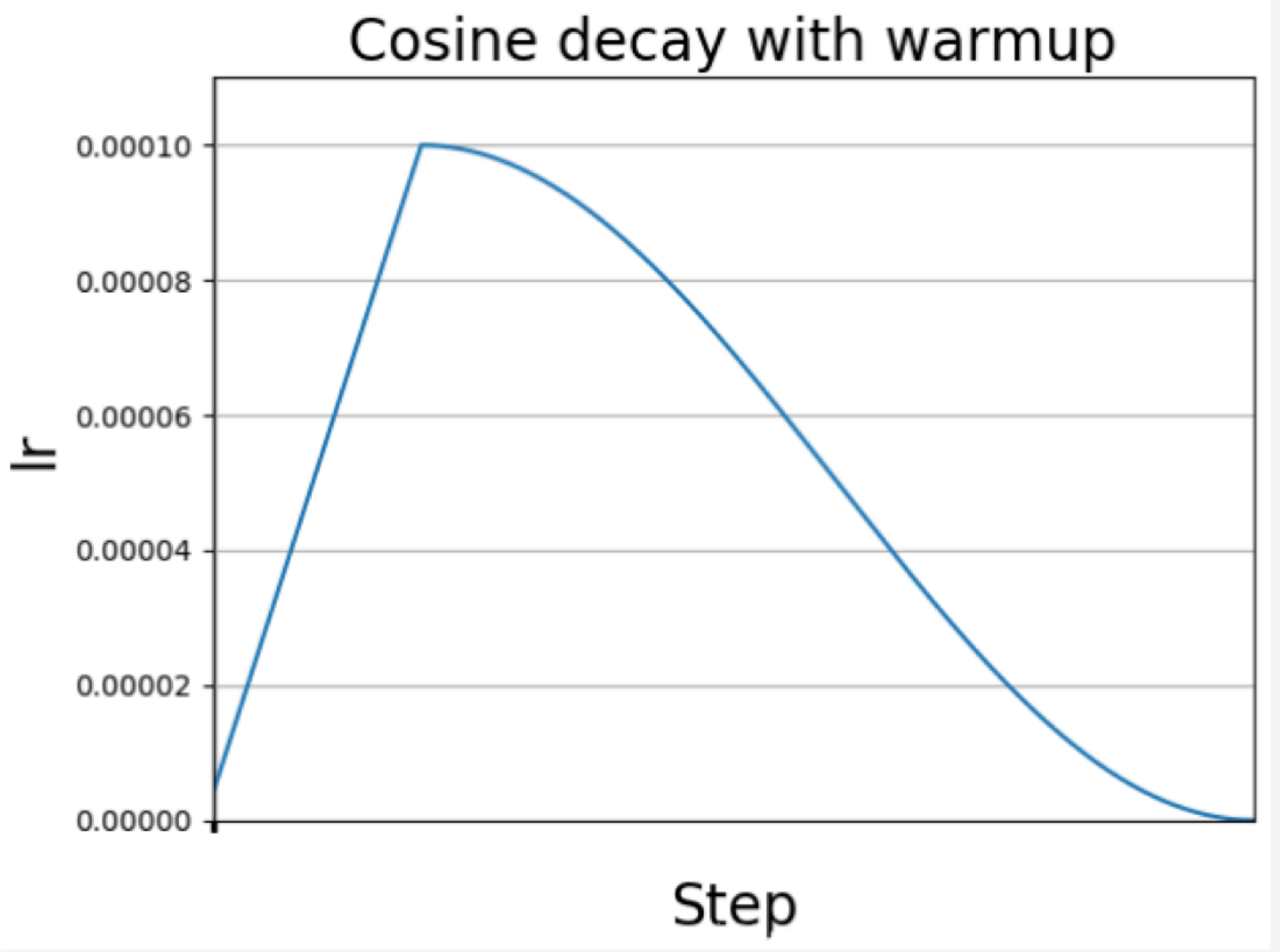
# Learning Rate Schedules



# Learning Rate Schedules

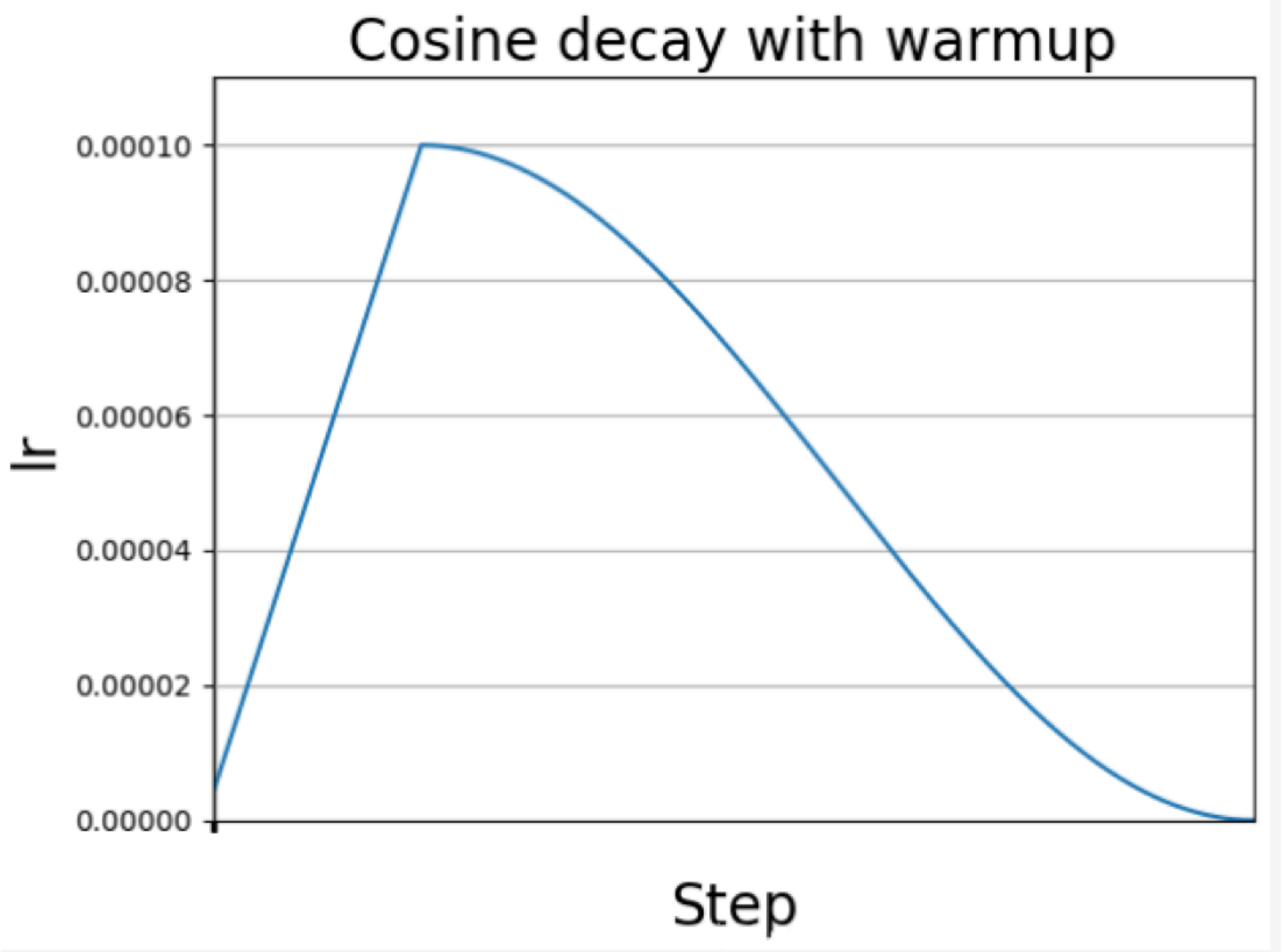


# Learning Rate Schedules



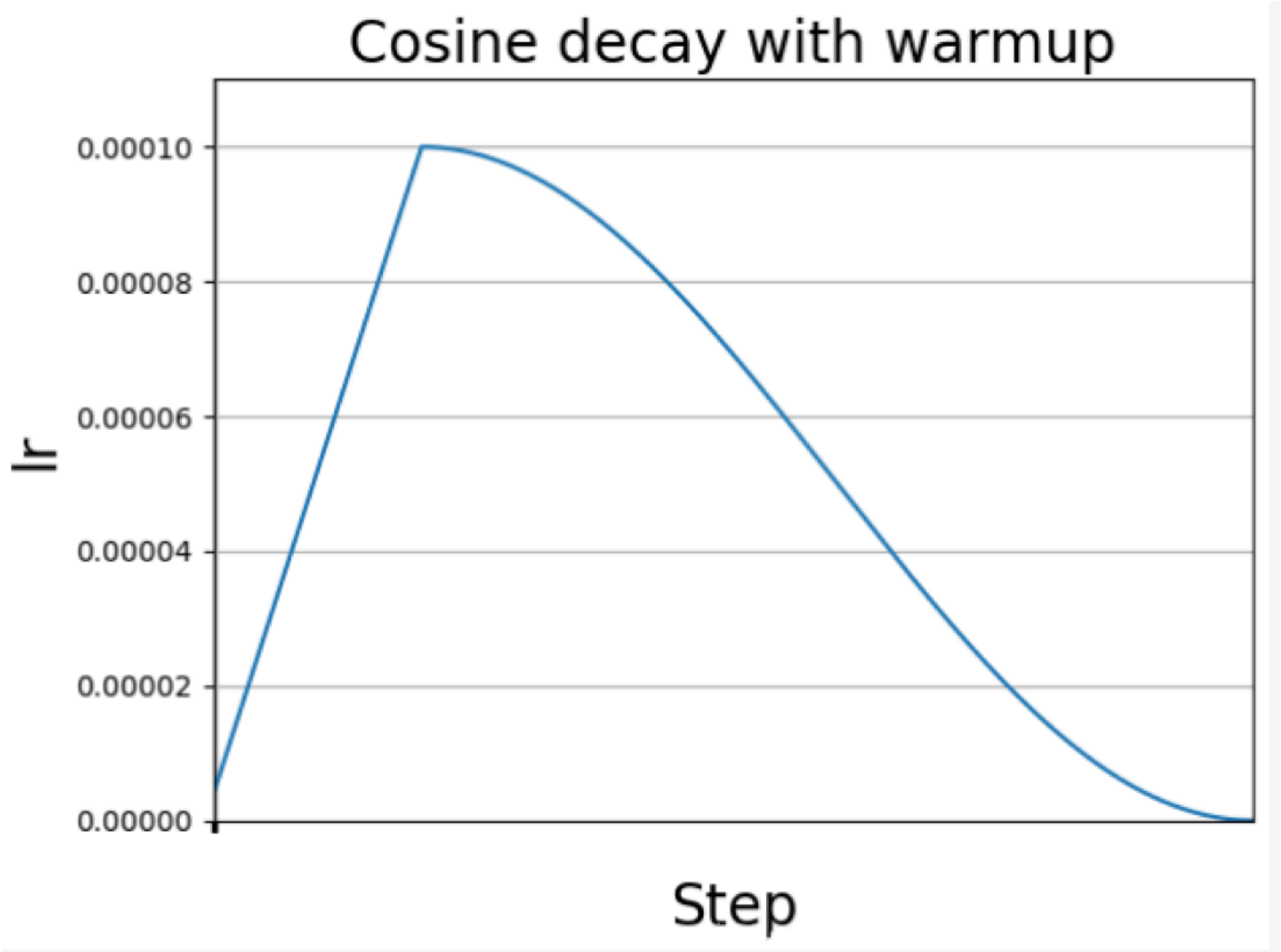

- We saw that learning rate decay is needed to reduce variance/noise.

# Learning Rate Schedules




- We saw that learning rate decay is needed to reduce variance/noise.
- Why do we need warmup?

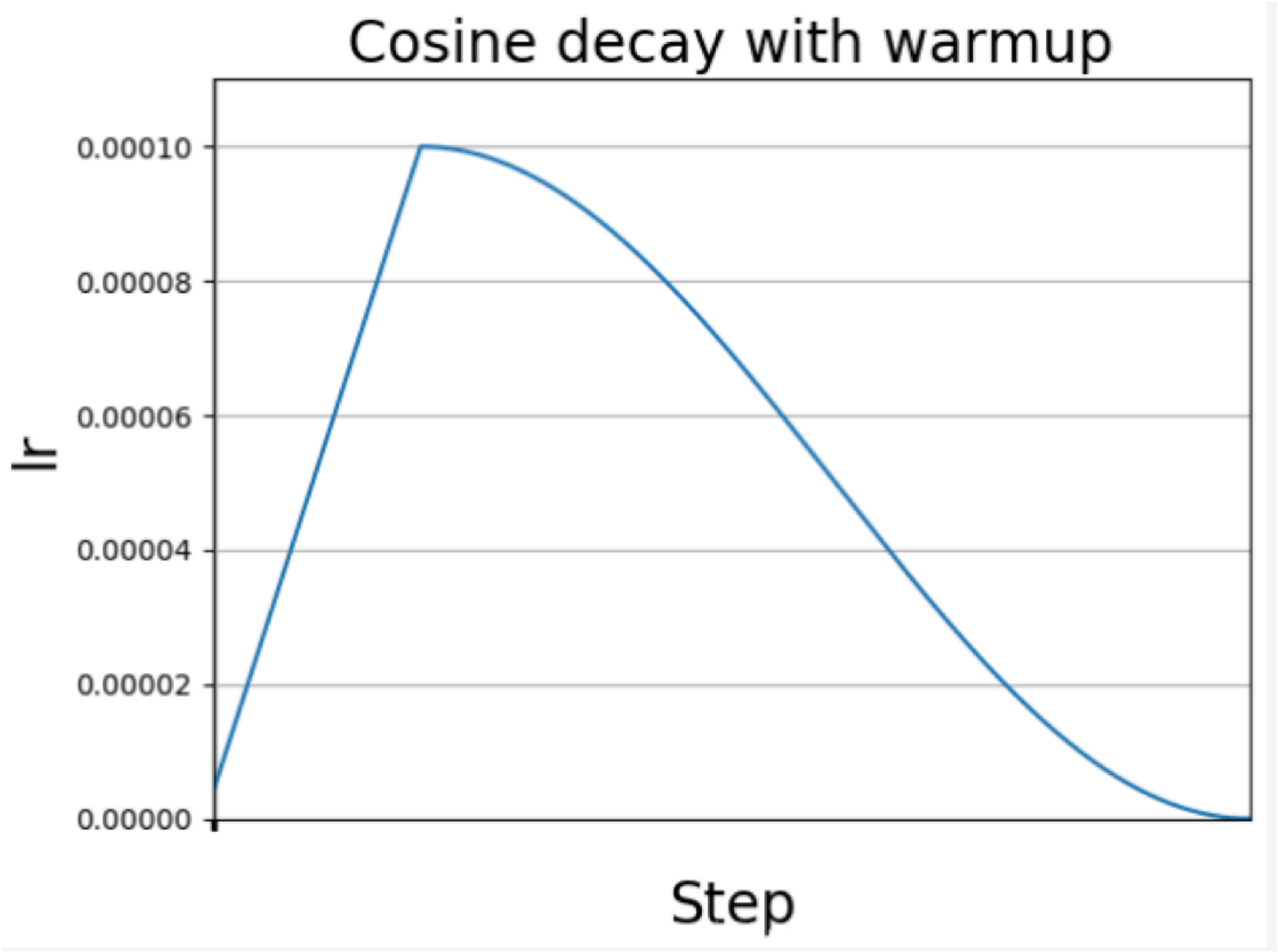
# Learning Rate Schedules




- We saw that learning rate decay is needed to reduce variance/noise.
- Why do we need warmup?

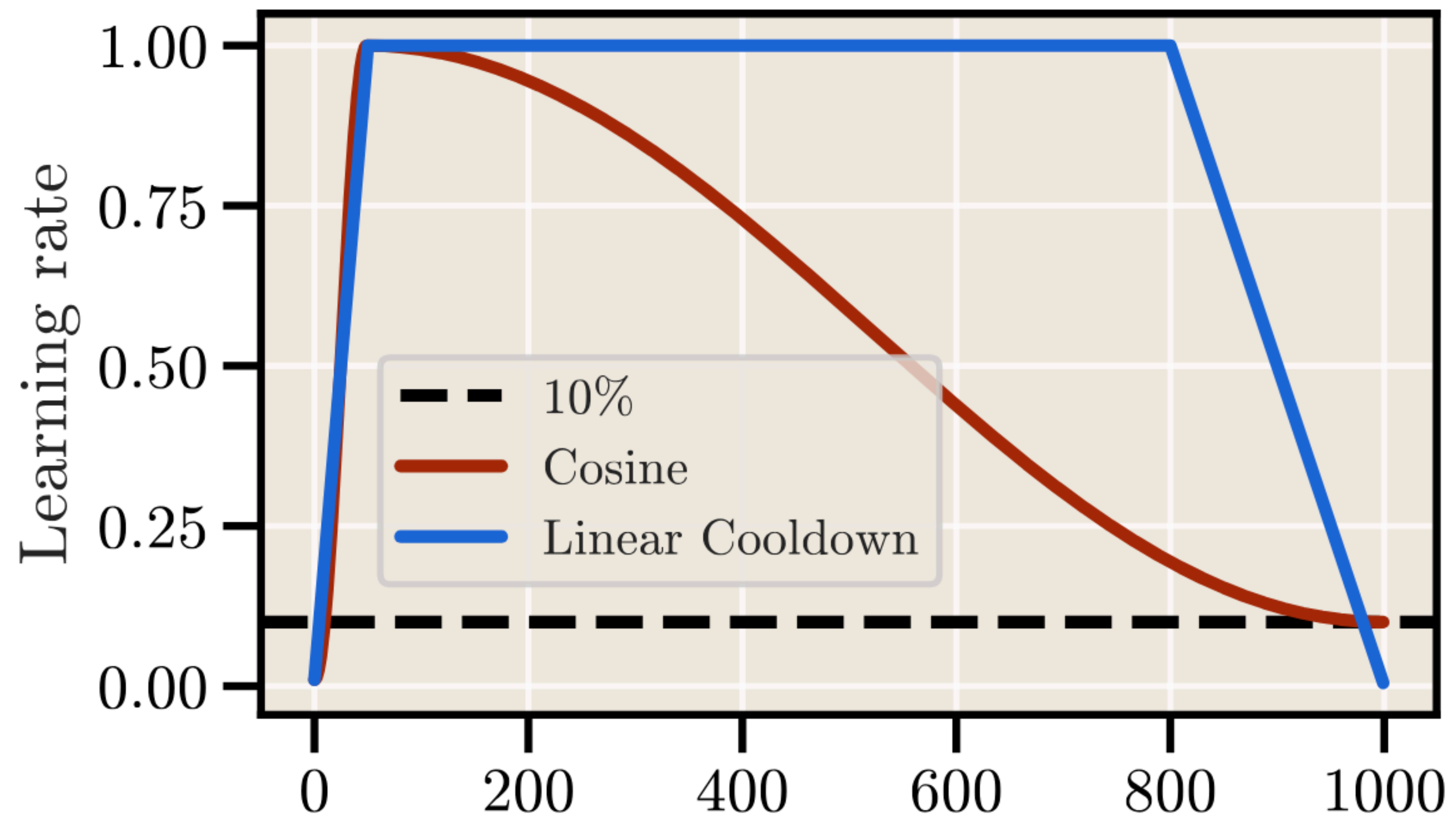
- Starting and ending at .1x max lr seems like a good default.

# Learning Rate Schedules

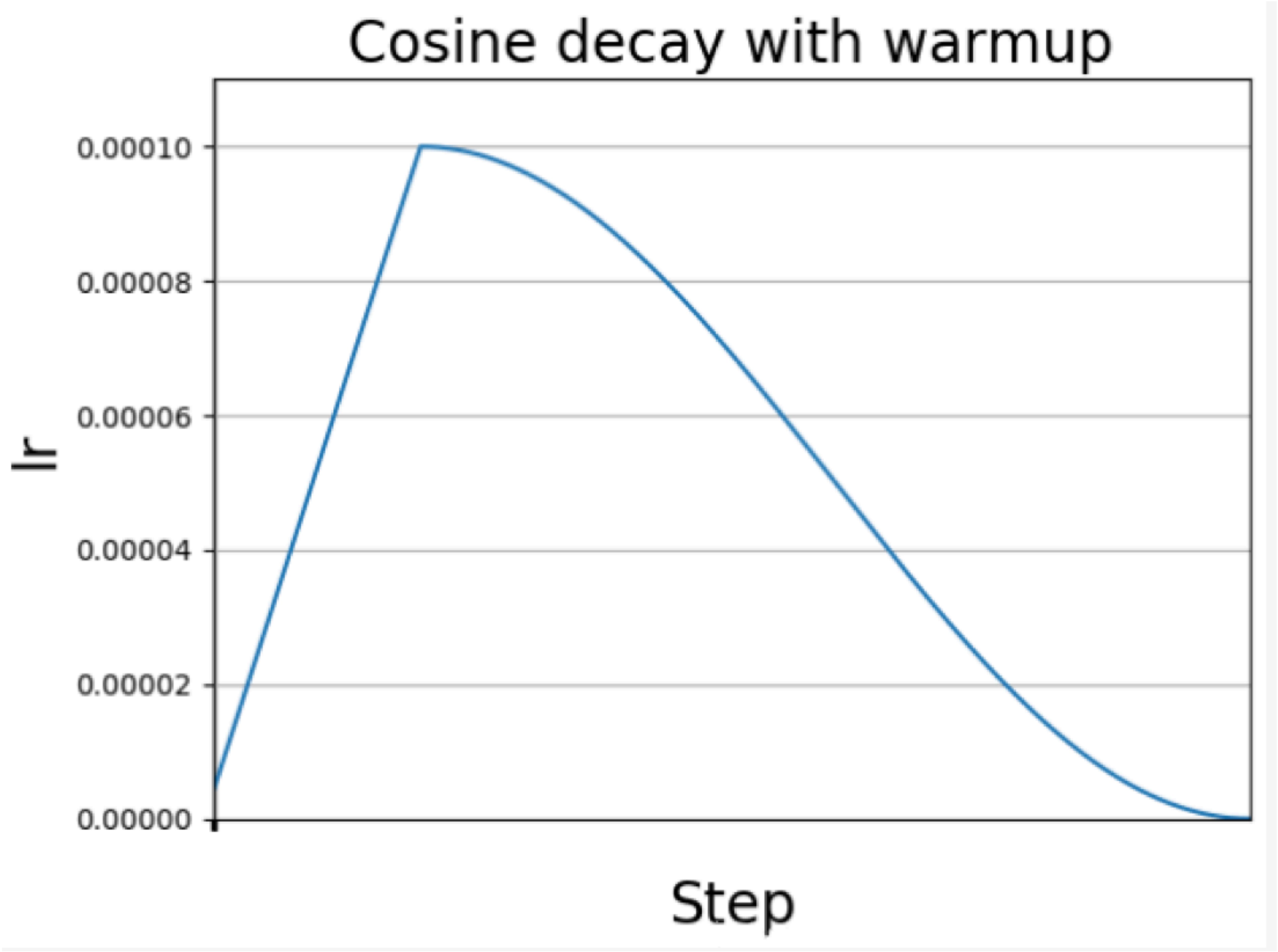



- We saw that learning rate decay is needed to reduce variance/noise.
- Why do we need warmup?

- Starting and ending at .1x max lr seems like a good default.
- People have recently been trying some modified schedules with marginal gains.

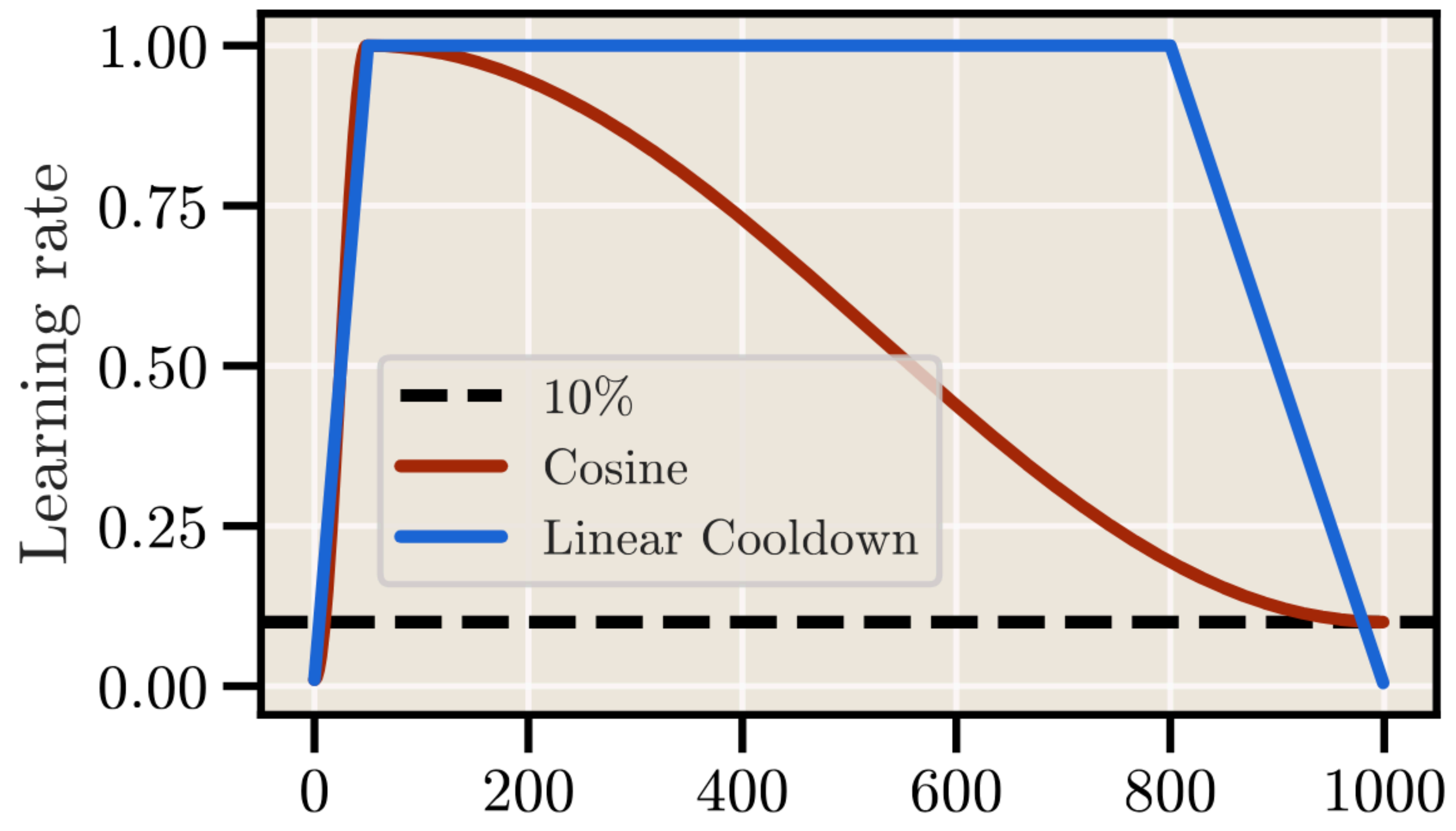


# Learning Rate Schedules




- We saw that learning rate decay is needed to reduce variance/noise.
- Why do we need warmup?

- Starting and ending at .1x max lr seems like a good default.
- People have recently been trying some modified schedules with marginal gains.
- Schedules can also be combined with weight averaging (important for diffusion models) Popularly known under terms “EWA/SWA”



# SGD: critical batch sizes

- Mini-batch SGD, with batchsize $m$:
  - Sample $m$ points iid, $(x_1, y_1)\ldots(x_m, y_m) \sim D$
  - $w_{t+1} = w_t + \eta_t \frac{1}{m} \sum_i (y_i - w_t \cdot x_i) x_i$
- The mini-batching benefits:
  - You can do the gradient computation updates in parallel.
  - The hope: large batch sizes can (substantially) reduce the serial time of optimization (at the same overall flops).
- (serial compute vs total compute) When we double the batch size, we hope that the loss drops twice as fast (in terms of number of iterations). This happens for small $m$ (for regression).
- The critical batch size is the batch size where this stops happening (i.e. where there is demising returns for doubling the batch size).
  - For regression, there is a sharp characterization of when this happens.
  - The same behavior happens for neural nets.

halving the l.r.
↕
double the batch
true.

# Critical Batch Size: (same as before)

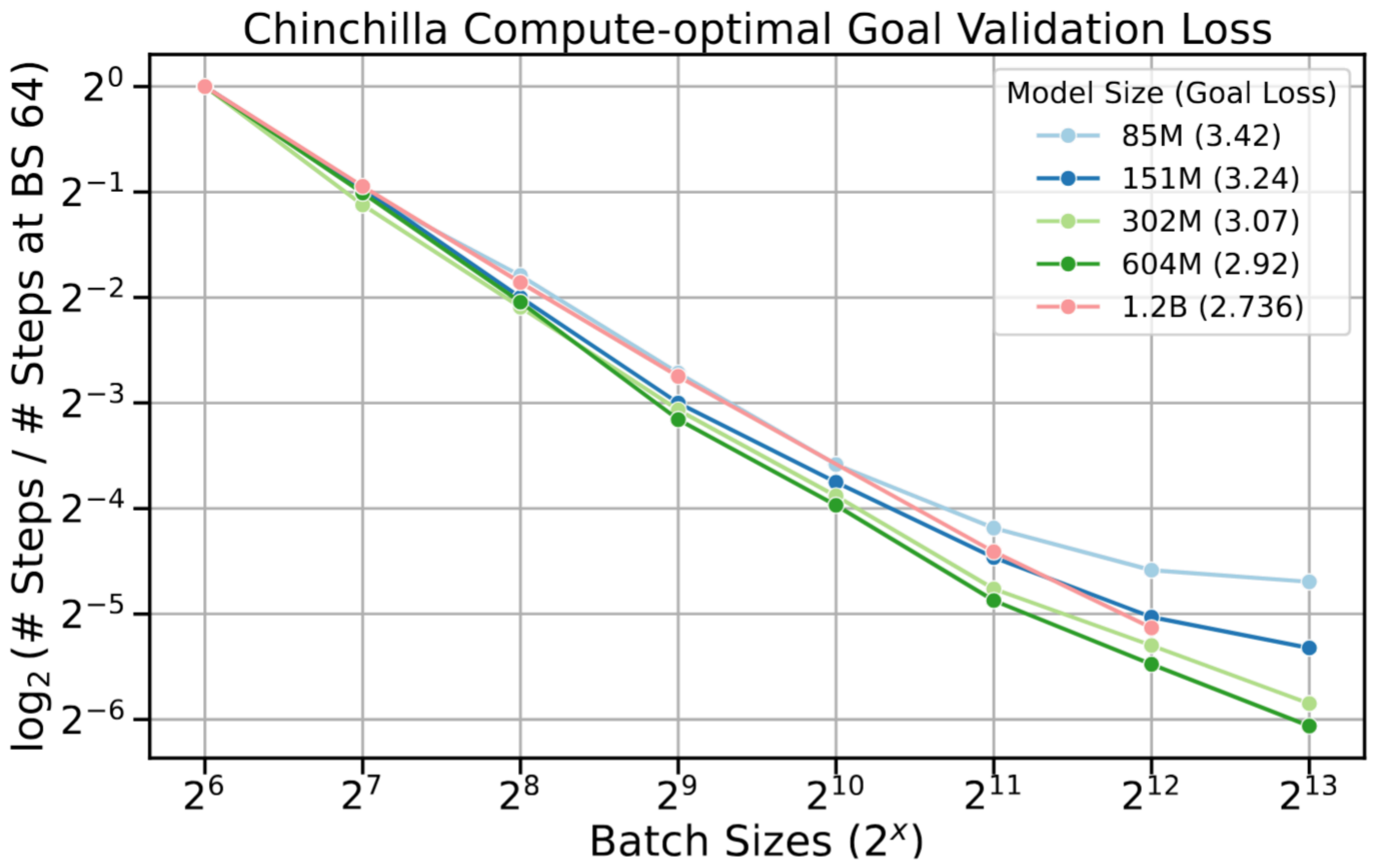
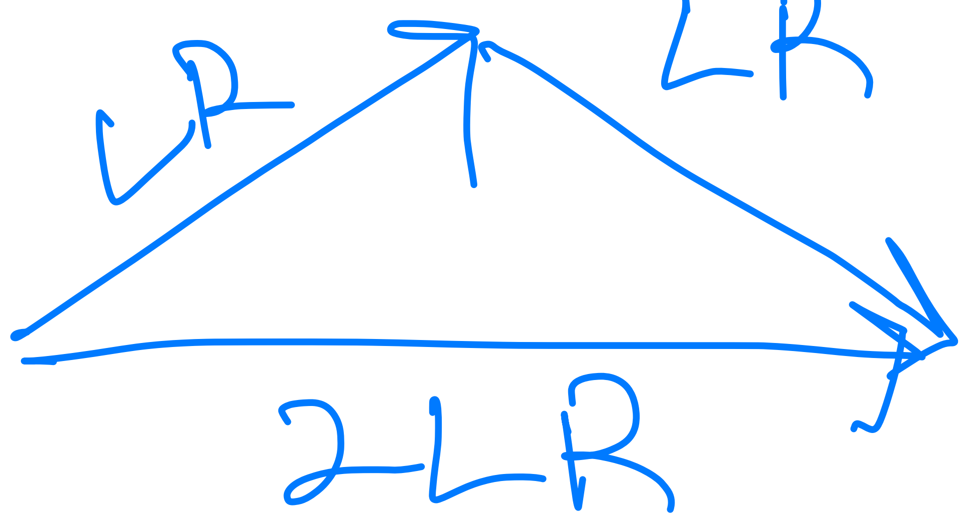



Figure 1: Model size ablation: y-axis - the number of steps to reach the Chinchilla-optimal validation loss of batch size 256.




Zhang et al. 2024, “Critical Batch Sizes in Language Model Training”, upcoming

# Today

- Announcements/Recap++
- Whirlwind Tour of Optimization
- DL Optimization Pipeline
  - Optimizers/Adam
  - Stability/Architecture Modifications
  - Learning rate scheduling/Batch size
  - ✓ Scaling Laws…
- Training Dynamics/Edge of stability

# How To Scale Up?:
# Scaling Laws For Everything

# Hyperparameter choices for scaling

We want to optimize for

# Hyperparameter choices for scaling

We want to optimize for

- Best loss for given compute (#params*#tokens),

# Hyperparameter choices for scaling

We want to optimize for

- Best loss for given compute (#params*#tokens),
- Best downstream evaluations for given compute.

# Hyperparameter choices for scaling

We want to optimize for

- Best loss for given compute (#params*#tokens),
- Best downstream evaluations for given compute.
- Best generalization for given data.

# Hyperparameter choices for scaling

We want to optimize for

- Best loss for given compute (#params*#tokens),
- Best downstream evaluations for given compute.
- Best generalization for given data.

# Hyperparameter choices for scaling

We want to optimize for

- Best loss for given compute (#params*#tokens),
- Best downstream evaluations for given compute.
- Best generalization for given data.

- Architecture:

# Hyperparameter choices for scaling

We want to optimize for

- Best loss for given compute (#params*#tokens),
- Best downstream evaluations for given compute.
- Best generalization for given data.

- Architecture:
  - Model choices: Attention type (Standard, Mamba), MLP type (Standard GeLU/ReLU, Swiglu) etc

# Hyperparameter choices for scaling

We want to optimize for

- Best loss for given compute (#params*#tokens),
- Best downstream evaluations for given compute.
- Best generalization for given data.

- Architecture:
  - Model choices: Attention type (Standard, Mamba), MLP type (Standard GeLU/ReLU, Swiglu) etc
  - Choices within architecture such as depth, width, vocabulary size.

# Hyperparameter choices for scaling

We want to optimize for

- Best loss for given compute (#params*#tokens),
- Best downstream evaluations for given compute.
- Best generalization for given data.

- Architecture:
  - Model choices: Attention type (Standard, Mamba), MLP type (Standard GeLU/ReLU, Swiglu) etc
  - Choices within architecture such as depth, width, vocabulary size.

- Optimization:

# Hyperparameter choices for scaling

We want to optimize for

- Best loss for given compute (#params*#tokens),
- Best downstream evaluations for given compute.
- Best generalization for given data.

- Architecture:
  - Model choices: Attention type (Standard, Mamba), MLP type (Standard GeLU/ReLU, Swiglu) etc
  - Choices within architecture such as depth, width, vocabulary size.

- Optimization:
  - Batch Size

# Hyperparameter choices for scaling

We want to optimize for

- Best loss for given compute (#params*#tokens),
- Best downstream evaluations for given compute.
- Best generalization for given data.

- Architecture:
  - Model choices: Attention type (Standard, Mamba), MLP type (Standard GeLU/ReLU, Swiglu) etc
  - Choices within architecture such as depth, width, vocabulary size.

- Optimization:
  - Batch Size
  - Warmup

# Hyperparameter choices for scaling

We want to optimize for

- Best loss for given compute (#params*#tokens),
- Best downstream evaluations for given compute.
- Best generalization for given data.

- Architecture:
  - Model choices: Attention type (Standard, Mamba), MLP type (Standard GeLU/ReLU, Swiglu) etc
  - Choices within architecture such as depth, width, vocabulary size.

- Optimization:
  - Batch Size
  - Warmup
  - Adam hyperparameters like $\beta_1, \beta_2, \epsilon$

# Hyperparameter choices for scaling

We want to optimize for
- Best loss for given compute (#params*#tokens),
- Best downstream evaluations for given compute.
- Best generalization for given data.

- Architecture:
  - Model choices: Attention type (Standard, Mamba), MLP type (Standard GeLU/ReLU, Swiglu) etc
  - Choices within architecture such as depth, width, vocabulary size.

- Optimization:
  - Batch Size
  - Warmup
  - Adam hyperparameters like $\beta_1, \beta_2, \epsilon$
  - Learning Rate

# Model Type

- Standard Approach: Show the benefits at a nontrivial scale and hope the benefits generalize to larger scales.

| Training Steps | 65,536 | 524,288 |
|---|---|---|
| $FFN_{ReLU}$ (*baseline*) | 1.997 (0.005) | 1.677 |
| $FFN_{GELU}$ | 1.983 (0.005) | 1.679 |
| $FFN_{Swish}$ | 1.994 (0.003) | 1.683 |
| $FFN_{GLU}$ | 1.982 (0.006) | 1.663 |
| $FFN_{Bilinear}$ | 1.960 (0.005) | 1.648 |
| $FFN_{GEGLU}$ | **1.942** (0.004) | **1.633** |
| $FFN_{SwiGLU}$ | **1.944** (0.010) | **1.636** |
| $FFN_{ReGLU}$ | 1.953 (0.003) | 1.645 |

Shazeer et al. 2020

# Model Type

- Standard Approach: Show the benefits at a nontrivial scale and hope the benefits generalize to larger scales.
- Scaling Laws approach: Show benefits with scale.

| Training Steps | 65,536 | 524,288 |
| --- | --- | --- |
| $\text{FFN}_{\text{ReLU}}$ (*baseline*) | 1.997 (0.005) | 1.677 |
| $\text{FFN}_{\text{GELU}}$ | 1.983 (0.005) | 1.679 |
| $\text{FFN}_{\text{Swish}}$ | 1.994 (0.003) | 1.683 |
| $\text{FFN}_{\text{GLU}}$ | 1.982 (0.006) | 1.663 |
| $\text{FFN}_{\text{Bilinear}}$ | 1.960 (0.005) | 1.648 |
| $\text{FFN}_{\text{GEGLU}}$ | **1.942** (0.004) | **1.633** |
| $\text{FFN}_{\text{SwiGLU}}$ | **1.944** (0.010) | **1.636** |
| $\text{FFN}_{\text{ReGLU}}$ | 1.953 (0.003) | 1.645 |

Shazeer et al. 2020

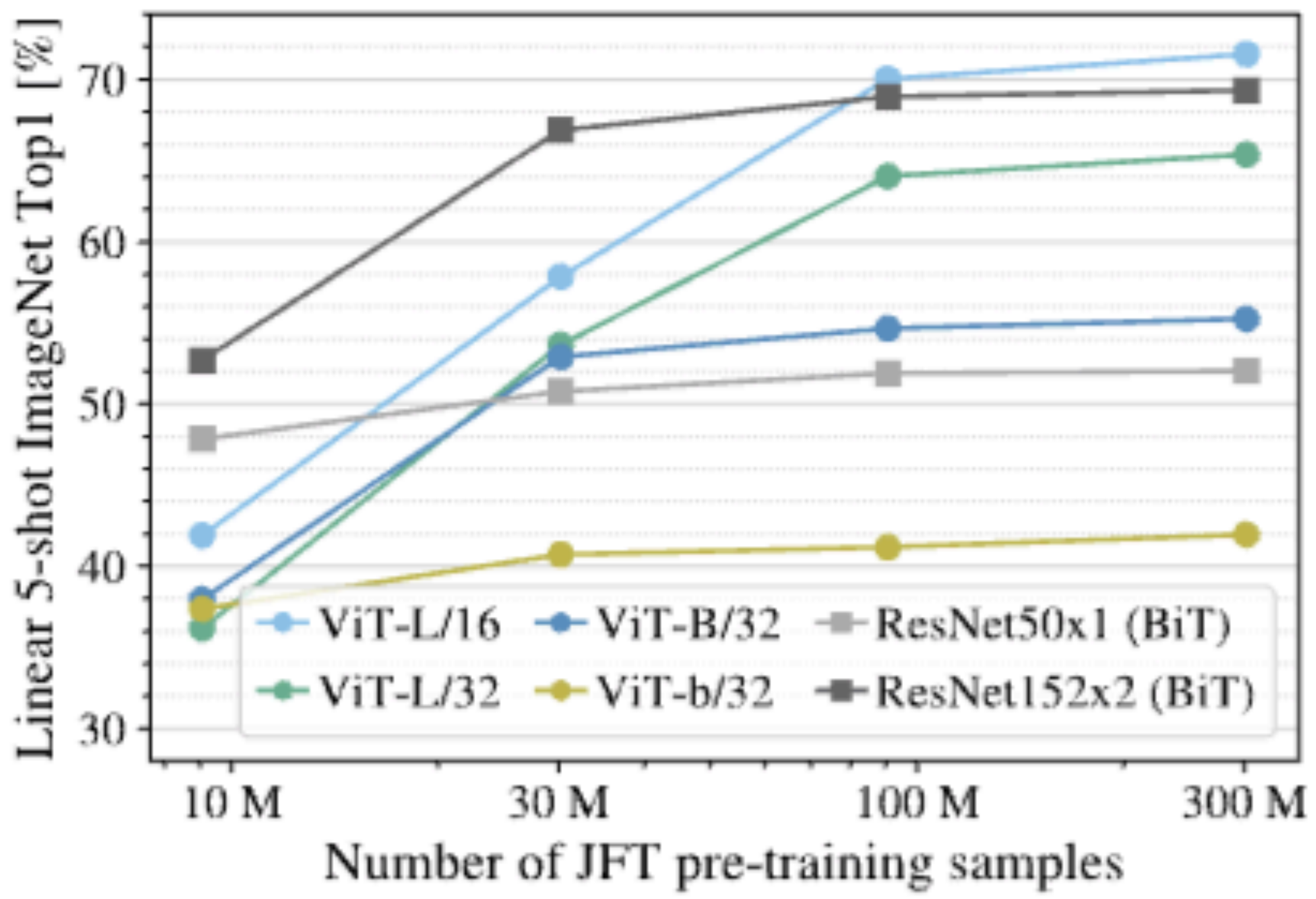



Figure 4: Linear few-shot evaluation on ImageNet versus pre-training size. ResNets perform better with smaller pre-training datasets but plateau sooner than ViT, which performs better with larger pre-training. ViT-b is ViT-B with all hidden dimensions halved.

Dosovitskiy et al. 2021
“ViT”

# Within Model Family Choices

Choices such as #tokens, depth, width, number of attention heads etc:

# Within Model Family Choices

Choices such as #tokens, depth, width, number of attention heads etc:
- Fit scaling laws.

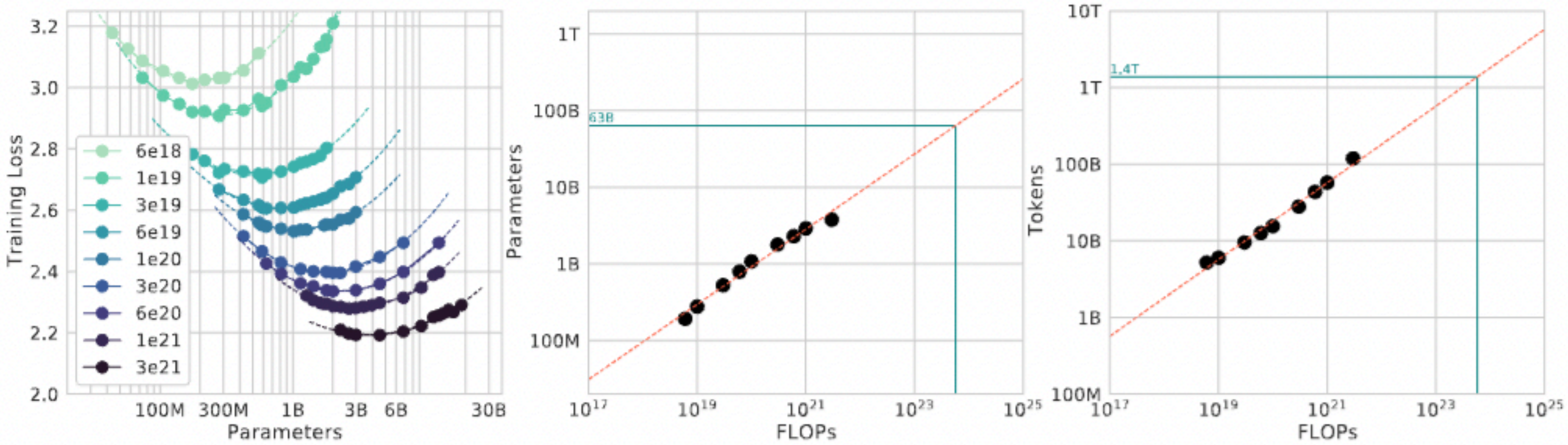



Figure 3 | **IsoFLOP curves.** For various model sizes, we choose the number of training tokens such that the final FLOPs is a constant. The cosine cycle length is set to match the target FLOP count. We find a clear valley in loss, meaning that for a given FLOP budget there is an optimal model to train (**left**). Using the location of these valleys, we project optimal model size and number of tokens for larger models (**center** and **right**). In green, we show the estimated number of parameters and tokens for an *optimal* model trained with the compute budget of *Gopher*.

# Optimizer

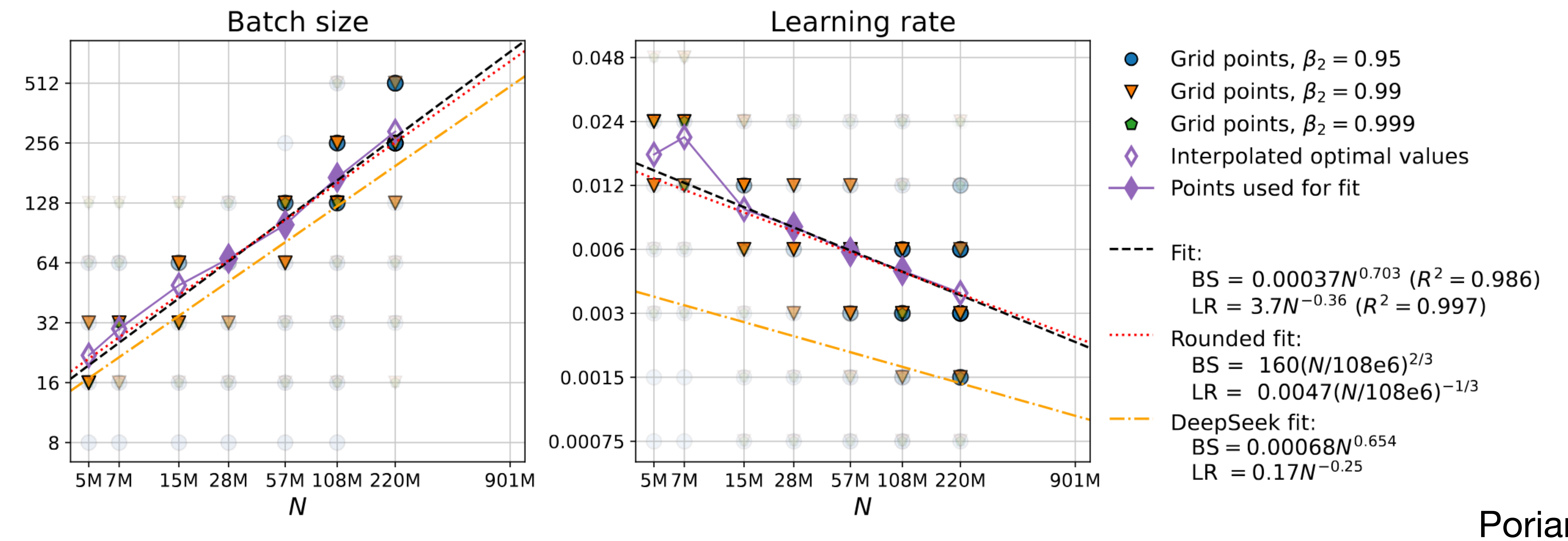
- $\beta_1, \beta_2$: Find values at medium scale and use them at large scale.

# Optimizer

- $\beta_1, \beta_2$: Find values at medium scale and use them at large scale.
- Batch Size:

# Optimizer

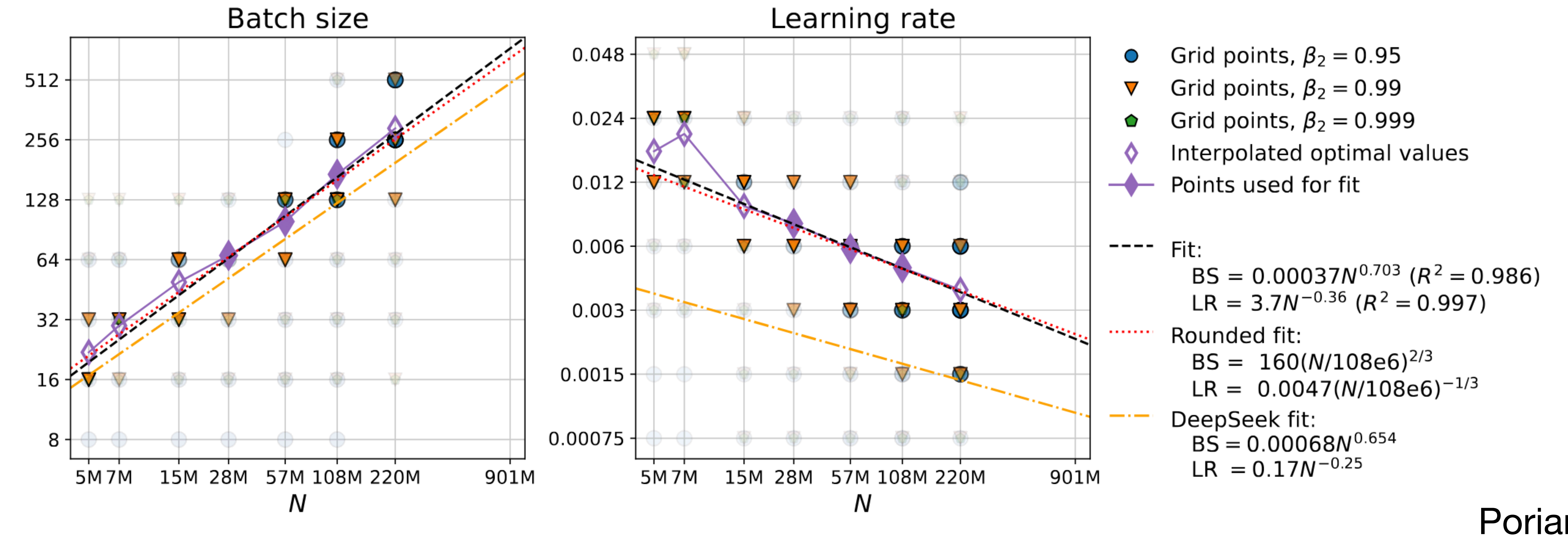
- $\beta_1, \beta_2$: Find values at medium scale and use them at large scale.
- Batch Size:
  - Fit scaling laws i.e. optimal batch size $= A +$ (model size)$^{\alpha}$.
    (Porian et al. 2024, Deepseek-AI 2024)




Porian et al. 2024
"Resolving Discrepancies in Compute-Optimal Scaling of Language Models"

# Optimizer

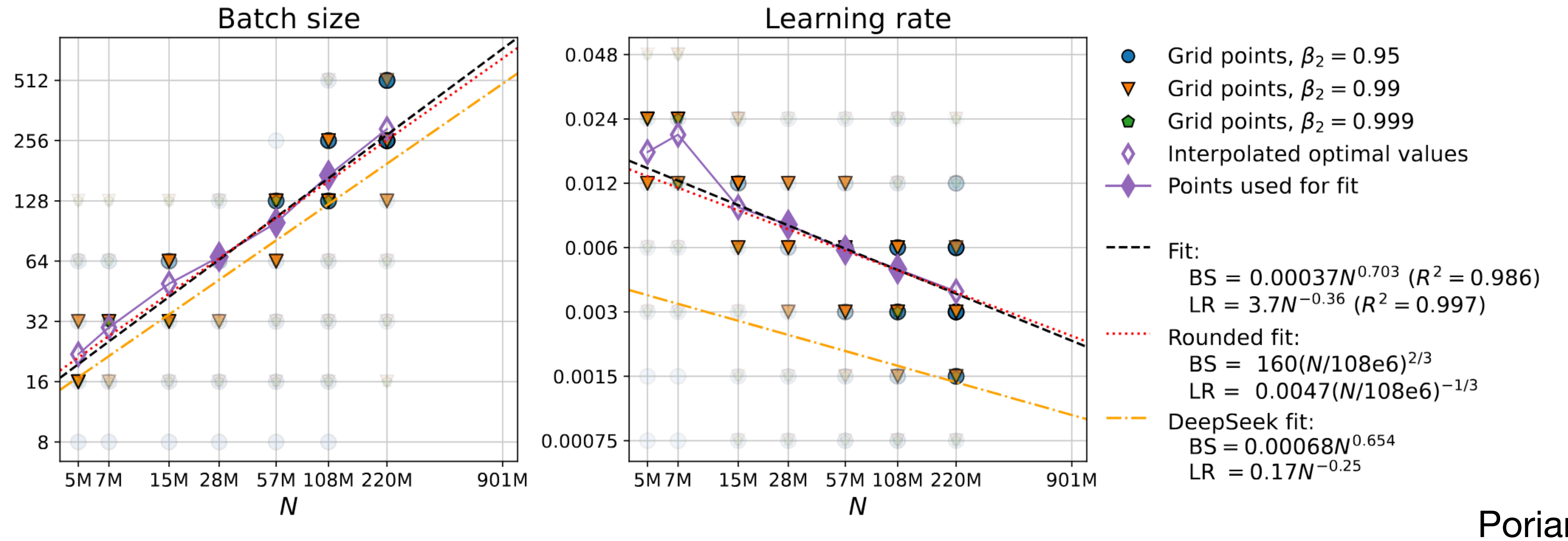
- $\beta_1, \beta_2$: Find values at medium scale and use them at large scale.
- Batch Size:
    - Fit scaling laws i.e. optimal batch size $= A +$ (model size)$^{\alpha}$. (Porian et al. 2024, Deepseek-AI 2024)
    - Is sometimes changed during training (Llama3, Chinchilla)




Porian et al. 2024
"Resolving Discrepancies in Compute-Optimal Scaling of Language Models"

# Optimizer

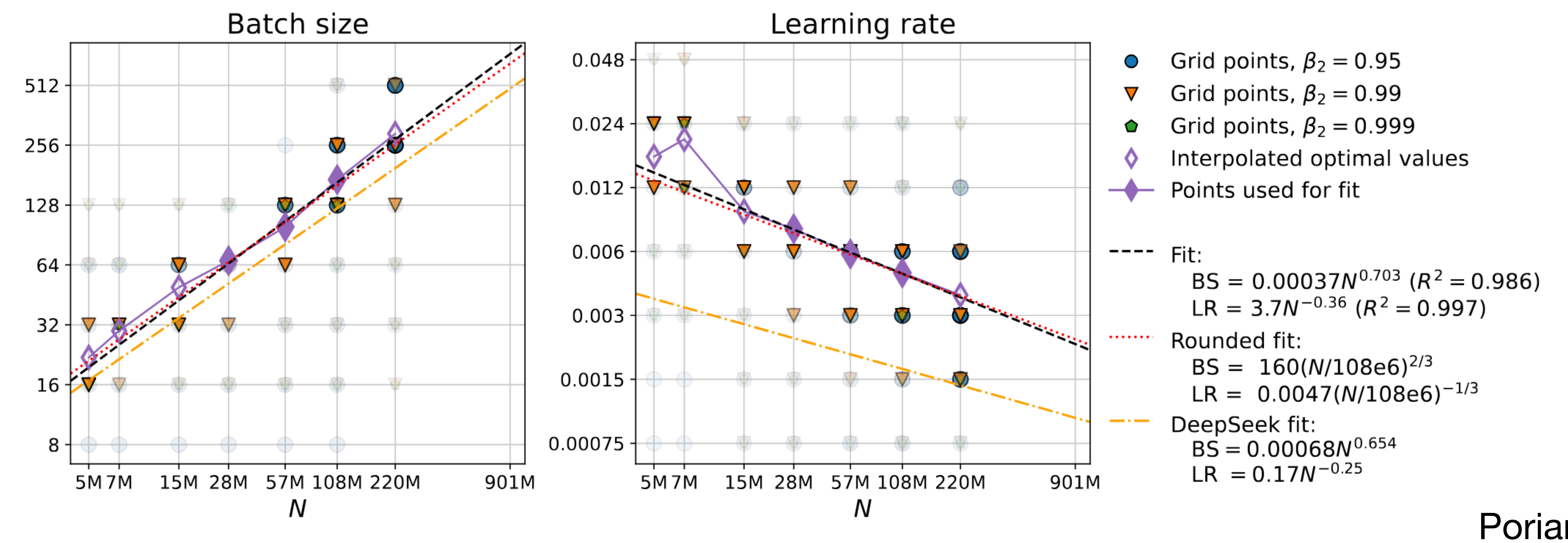
- $\beta_1, \beta_2$: Find values at medium scale and use them at large scale.
- Batch Size:
  - Fit scaling laws i.e. optimal batch size $= A +$ (model size)$^\alpha$. (Porian et al. 2024, Deepseek-AI 2024)
  - Is sometimes changed during training (Llama3, Chinchilla)
- Weight decay: Better to use decoupled weight decay (good defaults: 0,1e-4) i.e. $w_t = (1-\lambda)\cdot w_{t-1} - \eta \cdot g_t$ rather than $w_t = (1-\eta\lambda)\cdot w_{t-1} - \eta \cdot g_t$




Porian et al. 2024
"Resolving Discrepancies in Compute-Optimal Scaling of Language Models"

# Optimizer

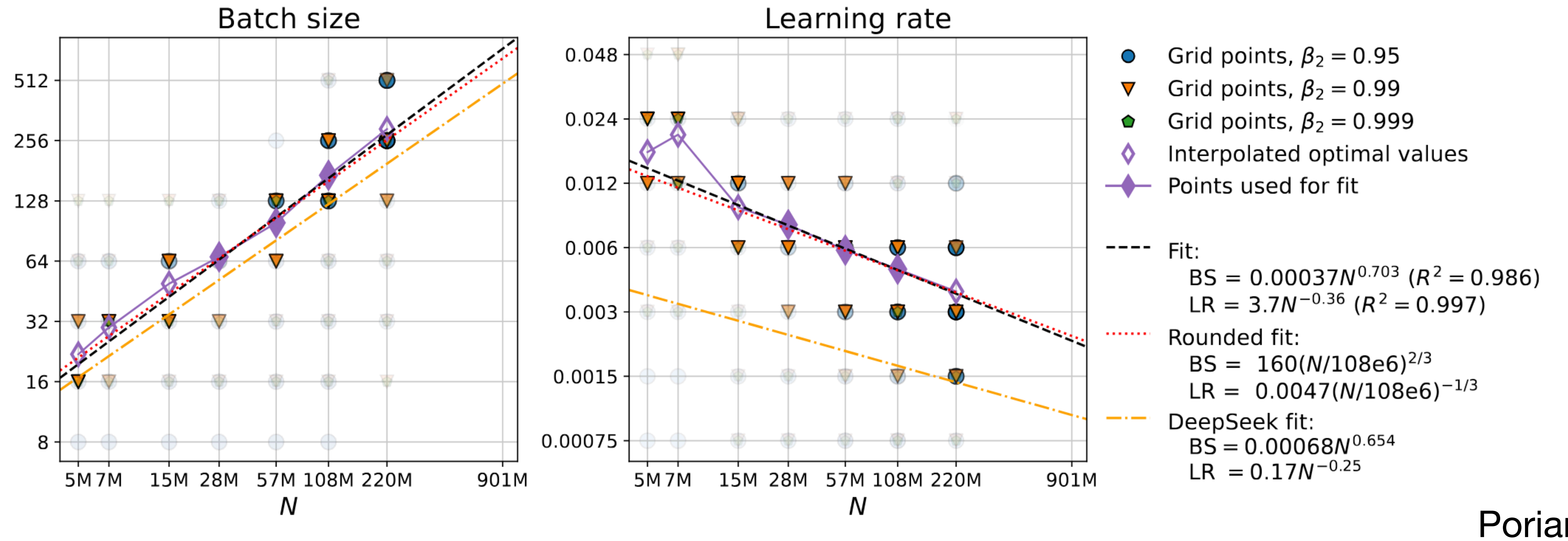
- $\beta_1, \beta_2$: Find values at medium scale and use them at large scale.
- Batch Size:
  - Fit scaling laws i.e. optimal batch size $= A +$ (model size)$^{\alpha}$. (Porian et al. 2024, Deepseek-AI 2024)
  - Is sometimes changed during training (Llama3, Chinchilla)
- Weight decay: Better to use decoupled weight decay (good defaults: 0,1e-4) i.e. $w_t = (1 - \lambda) \cdot w_{t-1} - \eta \cdot g_t$ rather than $w_t = (1 - \eta\lambda) \cdot w_{t-1} - \eta \cdot g_t$
- Warmup: No standard recommendations for LLMs, 20% might be a safe bet for medium sized models.




Porian et al. 2024
"Resolving Discrepancies in Compute-Optimal Scaling of Language Models"

# Optimizer

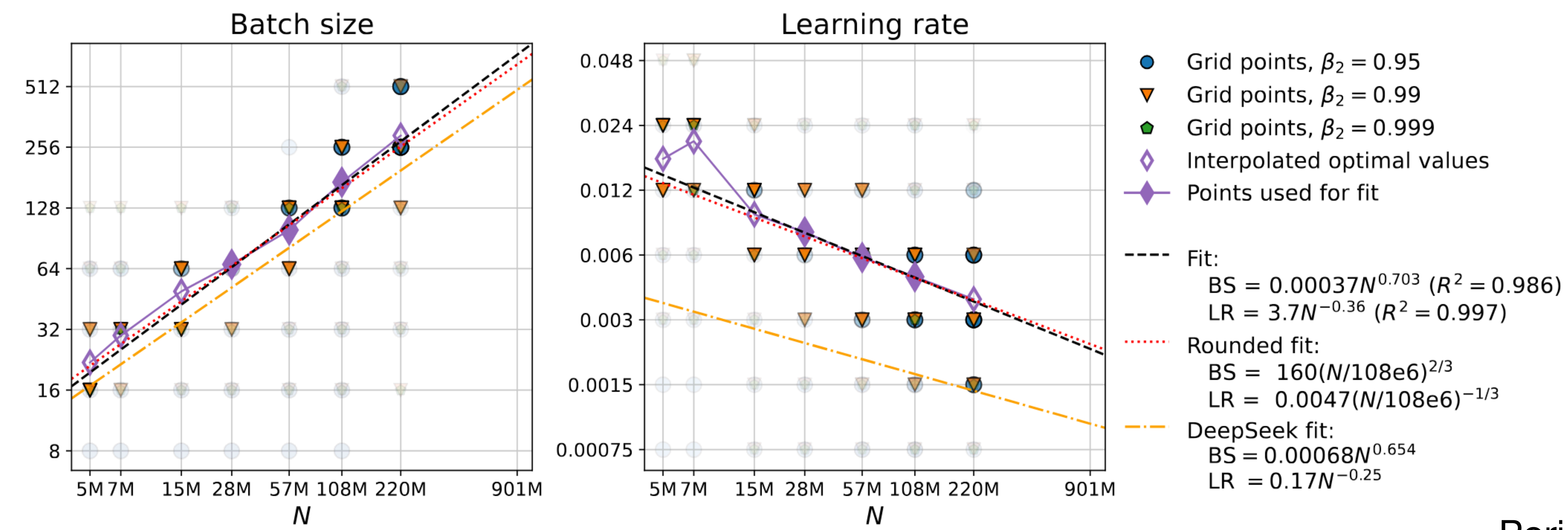
- $\beta_1, \beta_2$: Find values at medium scale and use them at large scale.
- Batch Size:
  - Fit scaling laws i.e. optimal batch size $= A +$ (model size)$^{\alpha}$. (Porian et al. 2024, Deepseek-AI 2024)
  - Is sometimes changed during training (Llama3, Chinchilla)
- Weight decay: Better to use decoupled weight decay (good defaults: 0,1e-4) i.e. $w_t = (1 - \lambda) \cdot w_{t-1} - \eta \cdot g_t$ rather than $w_t = (1 - \eta\lambda) \cdot w_{t-1} - \eta \cdot g_t$
- Warmup: No standard recommendations for LLMs, 20% might be a safe bet for medium sized models.




Porian et al. 2024
"Resolving Discrepancies in Compute-Optimal Scaling of Language Models"

# Optimizer

- $\beta_1, \beta_2$: Find values at medium scale and use them at large scale.
- Batch Size:
  - Fit scaling laws i.e. optimal batch size $= A + (\text{model size})^{\alpha}$. (Porian et al. 2024, Deepseek-AI 2024)
  - Is sometimes changed during training (Llama3, Chinchilla)
- Weight decay: Better to use decoupled weight decay (good defaults: 0,1e-4) i.e. $w_t = (1-\lambda)\cdot w_{t-1} - \eta \cdot g_t$ rather than $w_t = (1-\eta\lambda)\cdot w_{t-1} - \eta \cdot g_t$
- Warmup: No standard recommendations for LLMs, 20% might be a safe bet for medium sized models.




Porian et al. 2024
"Resolving Discrepancies in Compute-Optimal Scaling of Language Models"

# Learning Rate / μP

Learning rate (LR):

- Again fit scaling laws.
- Use learning rate transfer (Yang et al. 2022: Tensor Programs V/ μP):

# Learning Rate / µP

Learning rate (LR):
- Again fit scaling laws.
- Use learning rate transfer (Yang et al. 2022: Tensor Programs V/ µP):
  - To find optimal LR for a model of width $w_{\text{large}}$, we search for optimal LR of a model with smaller width $w_{\text{small}}$ (everything else like depth and batch size is held constant)

# Learning Rate / µP

Learning rate (LR):

- Again fit scaling laws.
- Use learning rate transfer (Yang et al. 2022: Tensor Programs V/ µP):
  - To find optimal LR for a model of width $w_{\text{large}}$, we search for optimal LR of a model with smaller width $w_{\text{small}}$ (everything else like depth and batch size is held constant)
  - Scale found optimal LR by $w_{\text{small}}/w_{\text{large}}$ .

# Learning Rate / µP

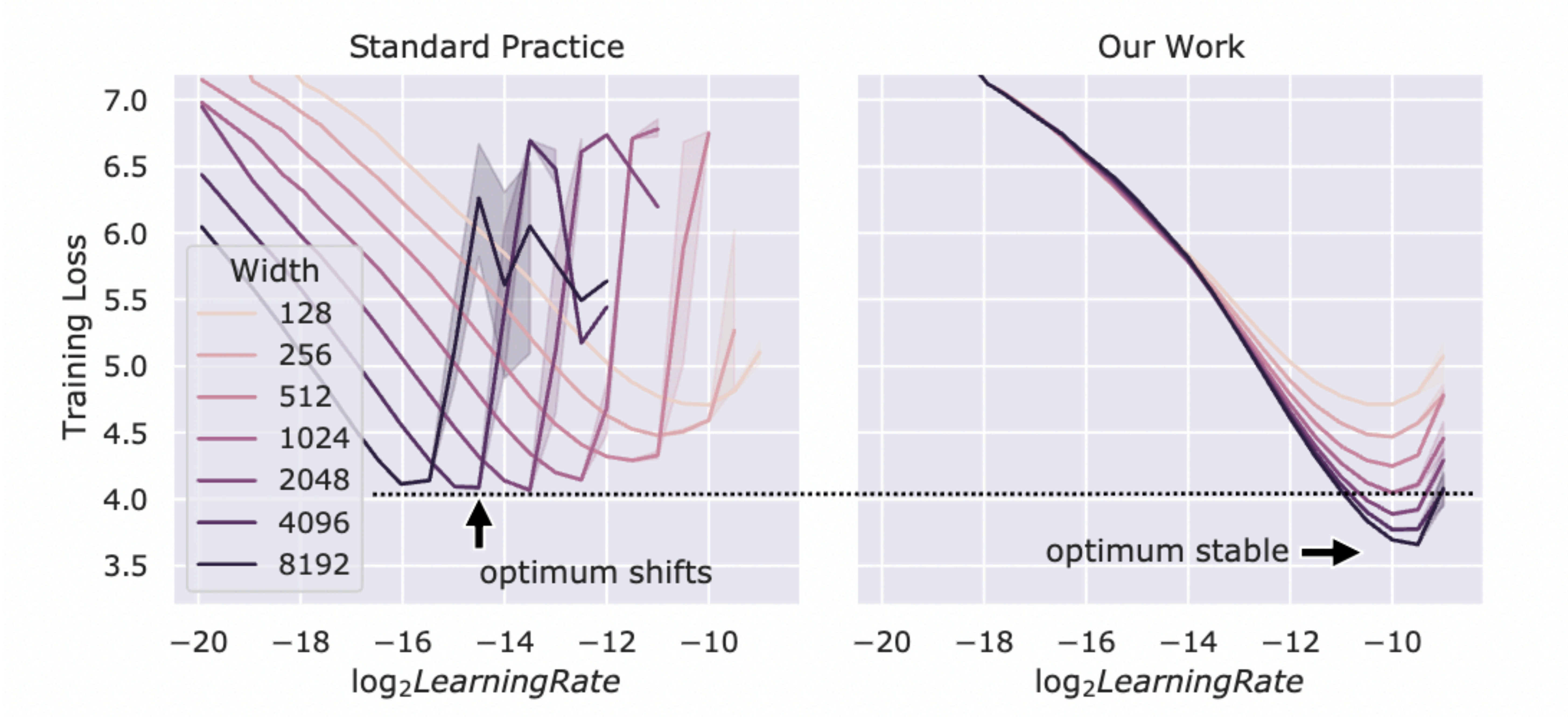
Learning rate (LR):

- Again fit scaling laws.
- Use learning rate transfer (Yang et al. 2022: Tensor Programs V/ µP):
  - To find optimal LR for a model of width $w_{\text{large}}$, we search for optimal LR of a model with smaller width $w_{\text{small}}$ (everything else like depth and batch size is held constant)
  - Scale found optimal LR by $w_{\text{small}}/w_{\text{large}}$ .

# Learning Rate / μP

Learning rate (LR):

- Again fit scaling laws.
- Use learning rate transfer (Yang et al. 2022: Tensor Programs V/ μP):
  - To find optimal LR for a model of width $w_{\text{large}}$, we search for optimal LR of a model with smaller width $w_{\text{small}}$ (everything else like depth and batch size is held constant)
  - Scale found optimal LR by $w_{\text{small}}/w_{\text{large}}$ .



# Today

- Announcements/Recap++
- Whirlwind Tour of Optimization
- DL Optimization Pipeline
- Training Dynamics
    - edge of stability
    - math

# Edge of Stability / Why Warmup?

# 'Optimal' Step size

- Reminder: In the quadratic model we need $\eta < 2/\lambda_{\text{max}} \iff \lambda_{\text{max}} < 2/\eta$ for stability.

# 'Optimal' Step size

- Reminder: In the quadratic model we need $\eta < 2/\lambda_{\text{max}} \iff \lambda_{\text{max}} < 2/\eta$ for stability.
- In deep learning, $\lambda_{\text{max}}$ can change with time: theory motivates using $\eta = 1/\lambda_{\text{max}}(\theta_t)$ for steepest descent.

# ‘Optimal’ Step size

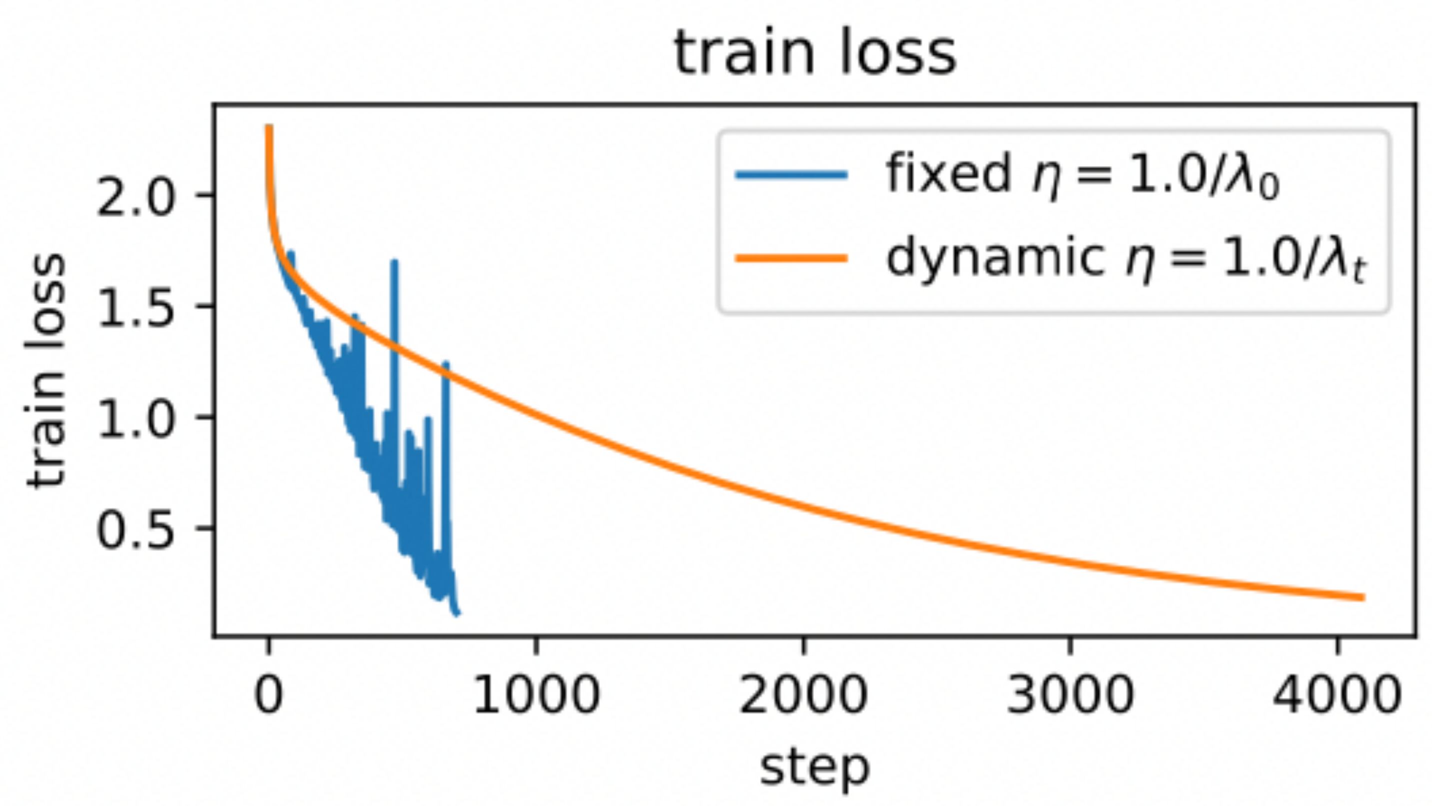
- Reminder: In the quadratic model we need $\eta < 2/\lambda_{\text{max}} \iff \lambda_{\text{max}} < 2/\eta$ for stability.
- In deep learning, $\lambda_{\text{max}}$ can change with time: theory motivates using $\eta = 1/\lambda_{\text{max}}(\theta_t)$ for steepest descent.
    - Let’s try using $\eta = 1/\lambda_{\text{max}}(\theta_t)$ and compare to a constant $\eta$!

# ‘Optimal’ Step size

- Reminder: In the quadratic model we need $\eta < 2/\lambda_{\text{max}} \iff \lambda_{\text{max}} < 2/\eta$ for stability.
- In deep learning, $\lambda_{\text{max}}$ can change with time: theory motivates using $\eta = 1/\lambda_{\text{max}}(\theta_t)$ for steepest descent.
  - Let’s try using $\eta = 1/\lambda_{\text{max}}(\theta_t)$ and compare to a constant $\eta$!
- We define the sharpness as $\lambda_{\text{max}}(\theta_t)$.

# ‘Optimal’ Step size

- Reminder: In the quadratic model we need $\eta < 2/\lambda_{\text{max}} \iff \lambda_{\text{max}} < 2/\eta$ for stability.
- In deep learning, $\lambda_{\text{max}}$ can change with time: theory motivates using $\eta = 1/\lambda_{\text{max}}(\theta_t)$ for steepest descent.
  - Let’s try using $\eta = 1/\lambda_{\text{max}}(\theta_t)$ and compare to a constant $\eta$!
- We define the sharpness as $\lambda_{\text{max}}(\theta_t)$.

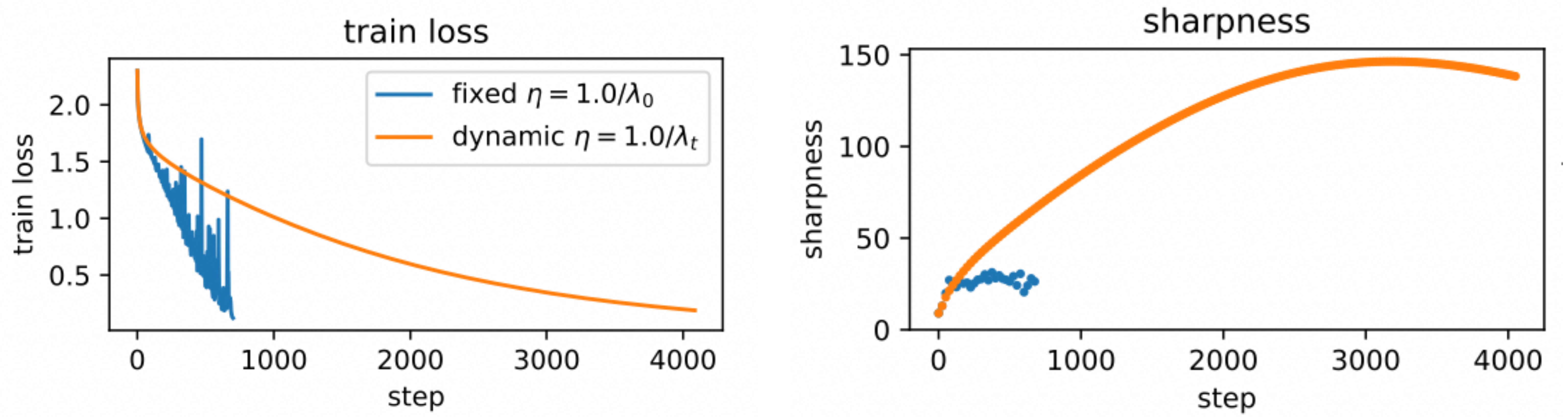





“Edge of Stability”
Cohen et al. 2021

# ‘Optimal’ Step size

- Reminder: In the quadratic model we need $\eta < 2/\lambda_{\text{max}} \iff \lambda_{\text{max}} < 2/\eta$ for stability.
- In deep learning, $\lambda_{\text{max}}$ can change with time: theory motivates using $\eta = 1/\lambda_{\text{max}}(\theta_t)$ for steepest descent.
  - Let’s try using $\eta = 1/\lambda_{\text{max}}(\theta_t)$ and compare to a constant $\eta$!
- We define the sharpness as $\lambda_{\text{max}}(\theta_t)$.

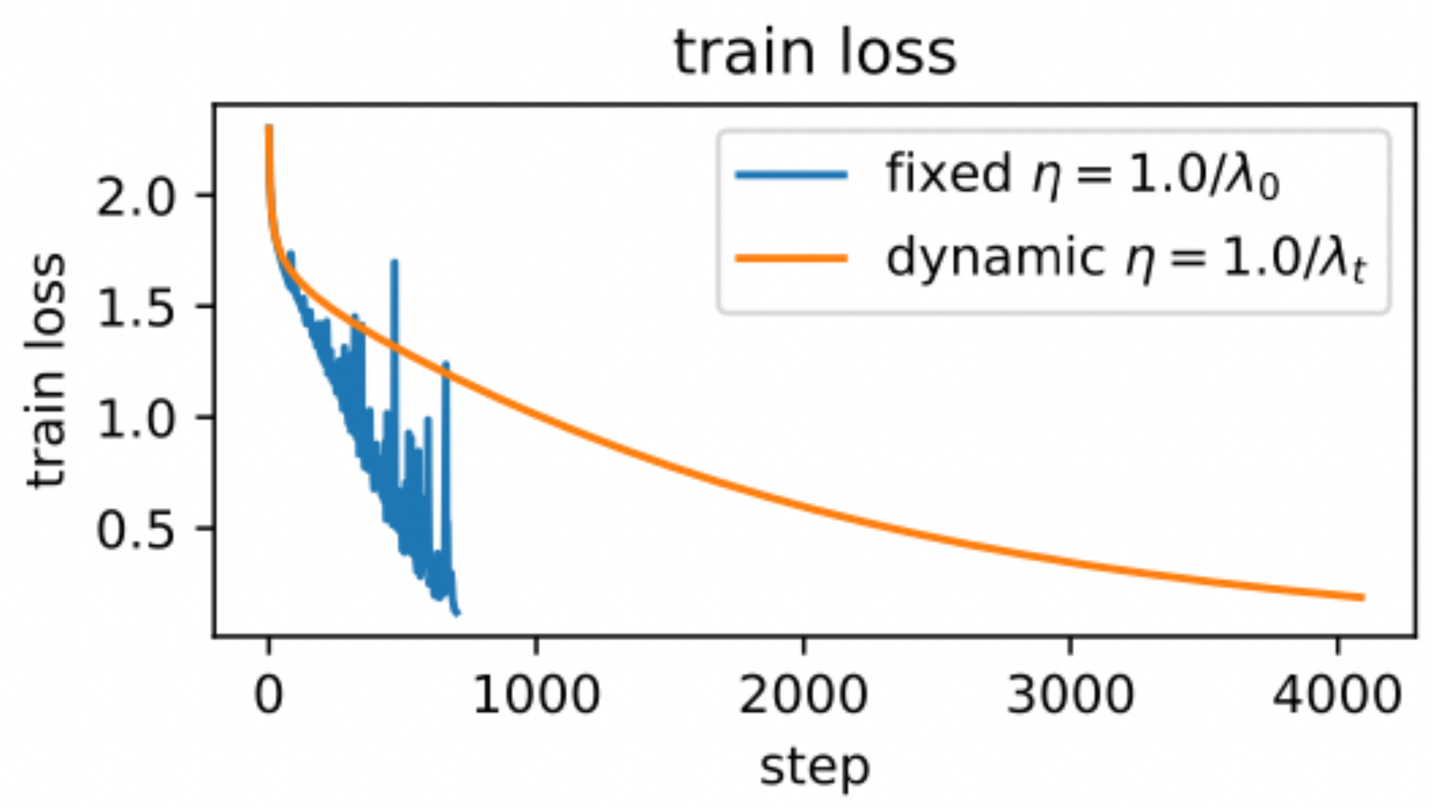


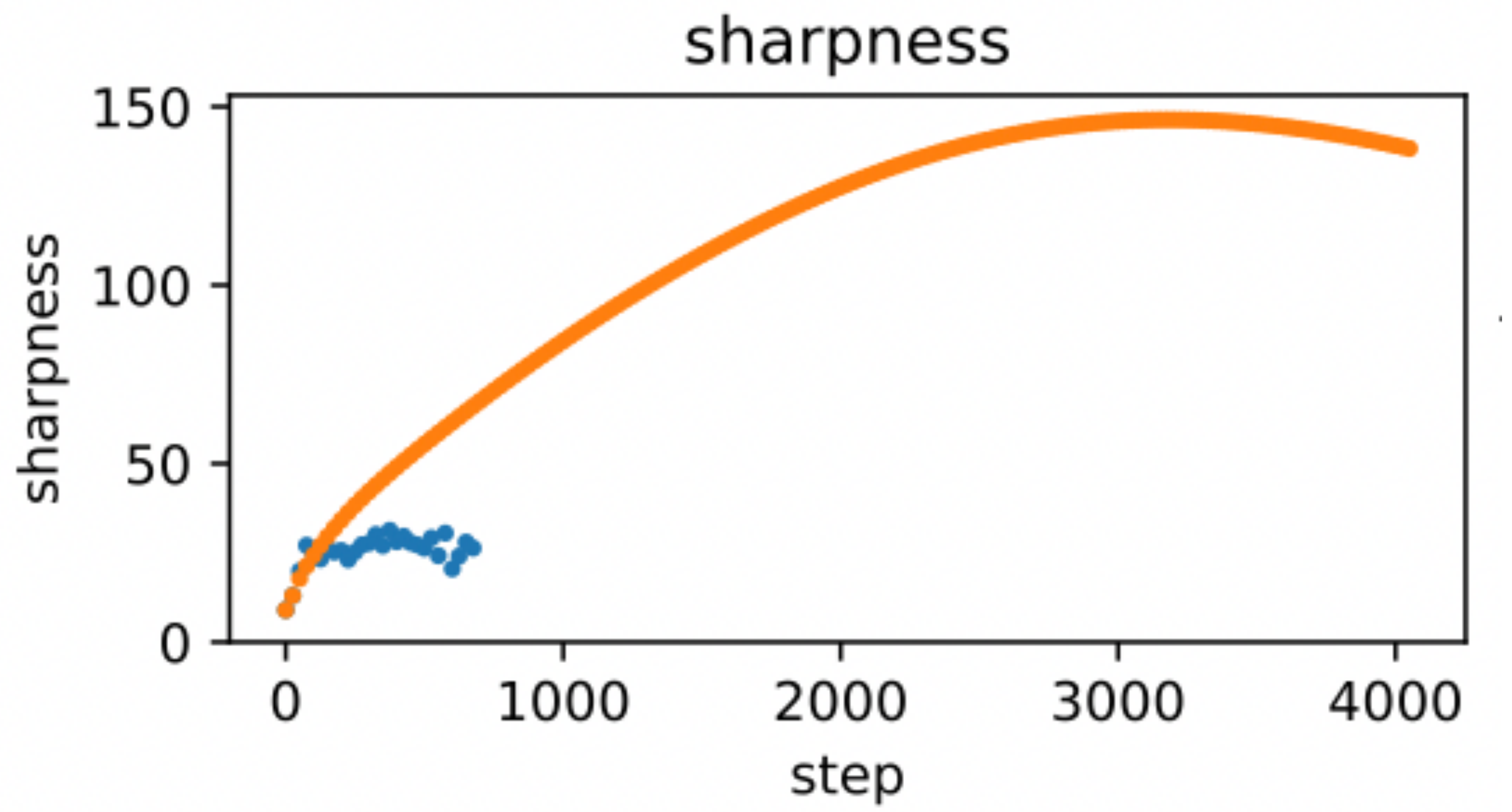



“Edge of Stability”
Cohen et al. 2021

# ‘Optimal’ Step size

- Reminder: In the quadratic model we need $\eta < 2/\lambda_{\max} \iff \lambda_{\max} < 2/\eta$ for stability.
- In deep learning, $\lambda_{\max}$ can change with time: theory motivates using $\eta = 1/\lambda_{\max}(\theta_t)$ for steepest descent.
  - Let’s try using $\eta = 1/\lambda_{\max}(\theta_t)$ and compare to a constant $\eta$!
- We define the sharpness as $\lambda_{\max}(\theta_t)$.



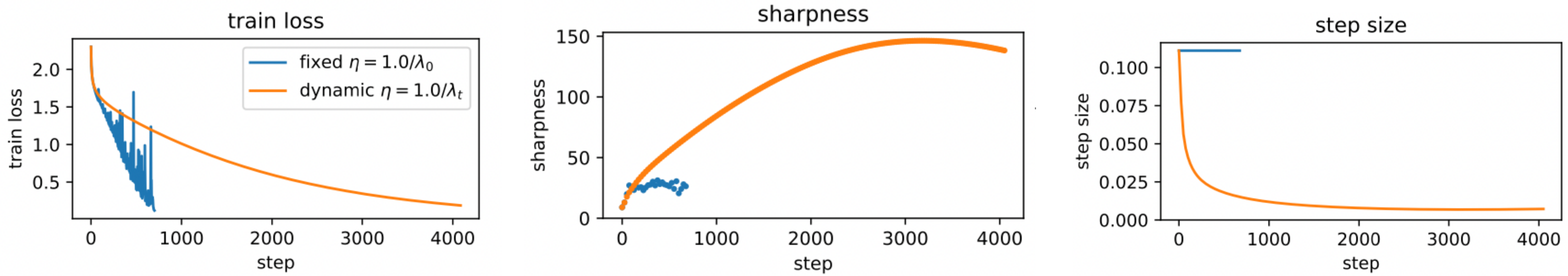




“Progressive sharpening”

“Edge of Stability”
Cohen et al. 2021

# 'Optimal' Step size

- Reminder: In the quadratic model we need $\eta < 2/\lambda_{\max} \iff \lambda_{\max} < 2/\eta$ for stability.
- In deep learning, $\lambda_{\max}$ can change with time: theory motivates using $\eta = 1/\lambda_{\max}(\theta_t)$ for steepest descent.
    - Let's try using $\eta = 1/\lambda_{\max}(\theta_t)$ and compare to a constant $\eta$!
- We define the sharpness as $\lambda_{\max}(\theta_t)$.




"Progressive sharpening"

"Edge of Stability"
Cohen et al. 2021

# Edge of Stability

- Define $\lambda^\star(\eta) = 2/\eta$, which is the maxima eigenvalue that leads to stable dynamics for lr $\eta$.
- If $\lambda(\theta) > \lambda^\star(\eta)$, then the dynamics should be unstable.

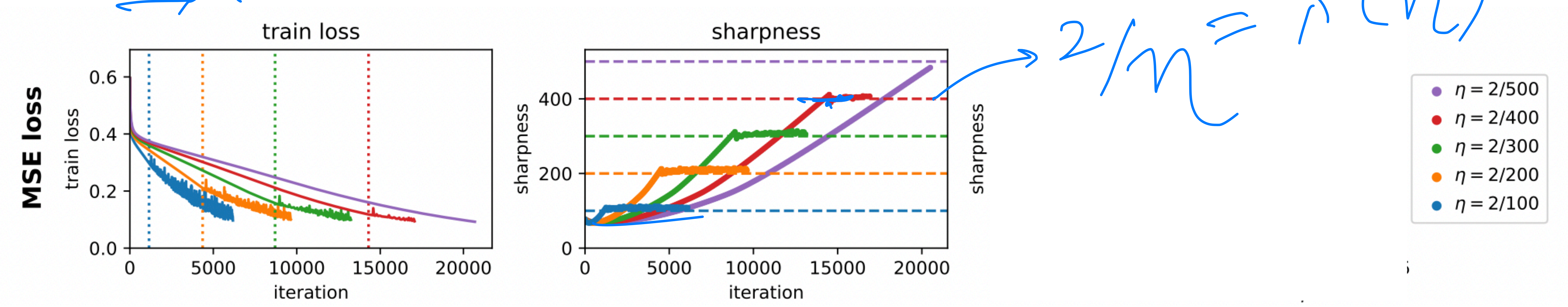
# Edge of Stability

- Define $\lambda^\star(\eta) = 2/\eta$, which is the maxima eigenvalue that leads to stable dynamics for lr $\eta$.
- If $\lambda(\theta) > \lambda^\star(\eta)$, then the dynamics should be unstable.

"Edge of Stability"
Cohen et al. 2021

# Edge of Stability

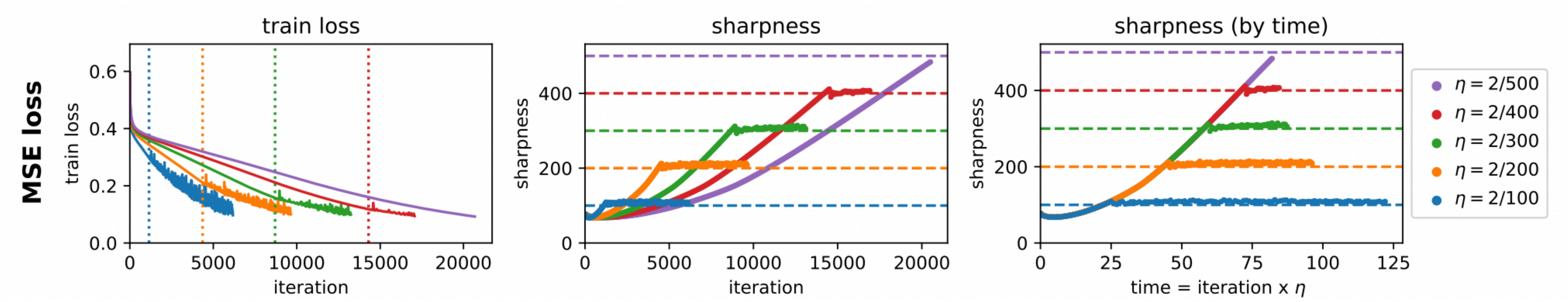
- Define $\lambda^{\star}(\eta) = 2/\eta$, which is the maxima eigenvalue that leads to stable dynamics for lr $\eta$.
- If $\lambda(\theta) > \lambda^{\star}(\eta)$, then the dynamics should be unstable.




"Edge of Stability"
Cohen et al. 2021

# Edge of Stability

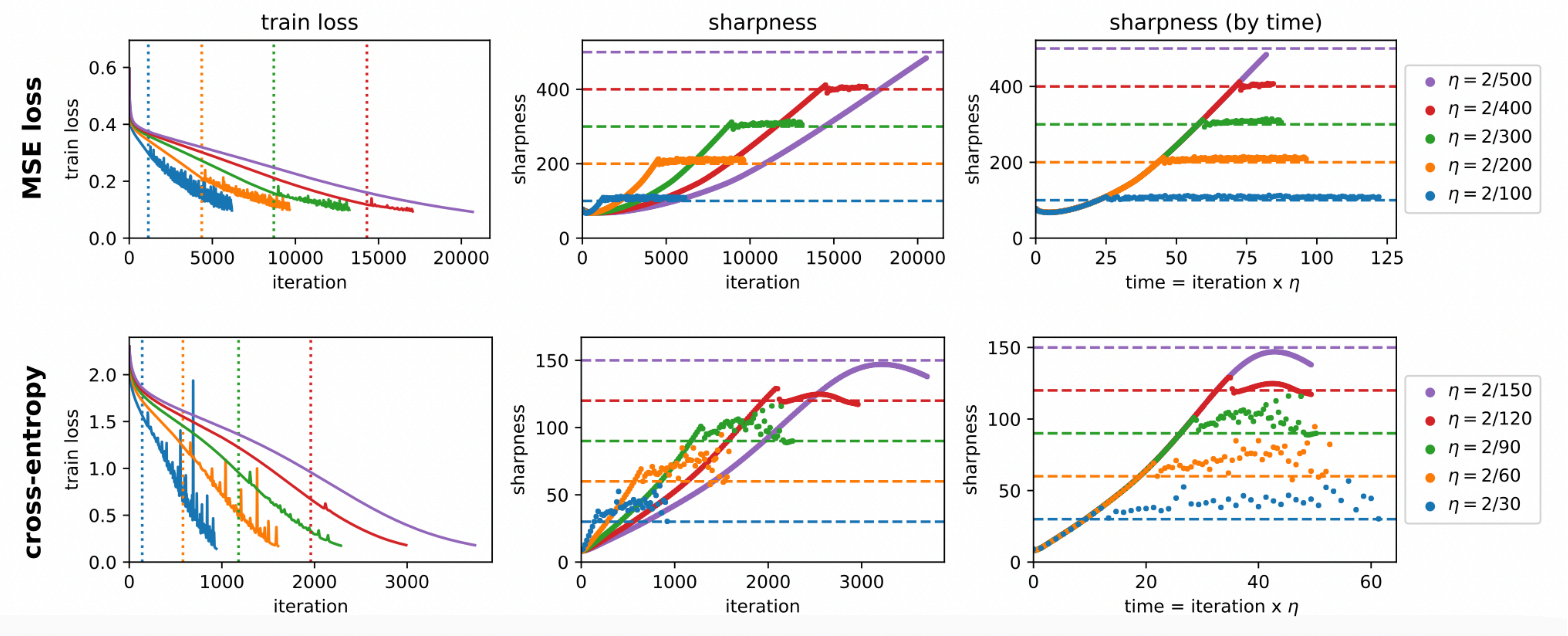
- Define $\lambda^\star(\eta) = 2/\eta$, which is the maxima eigenvalue that leads to stable dynamics for lr $\eta$.
- If $\lambda(\theta) > \lambda^\star(\eta)$, then the dynamics should be unstable.




"Edge of Stability"
Cohen et al. 2021

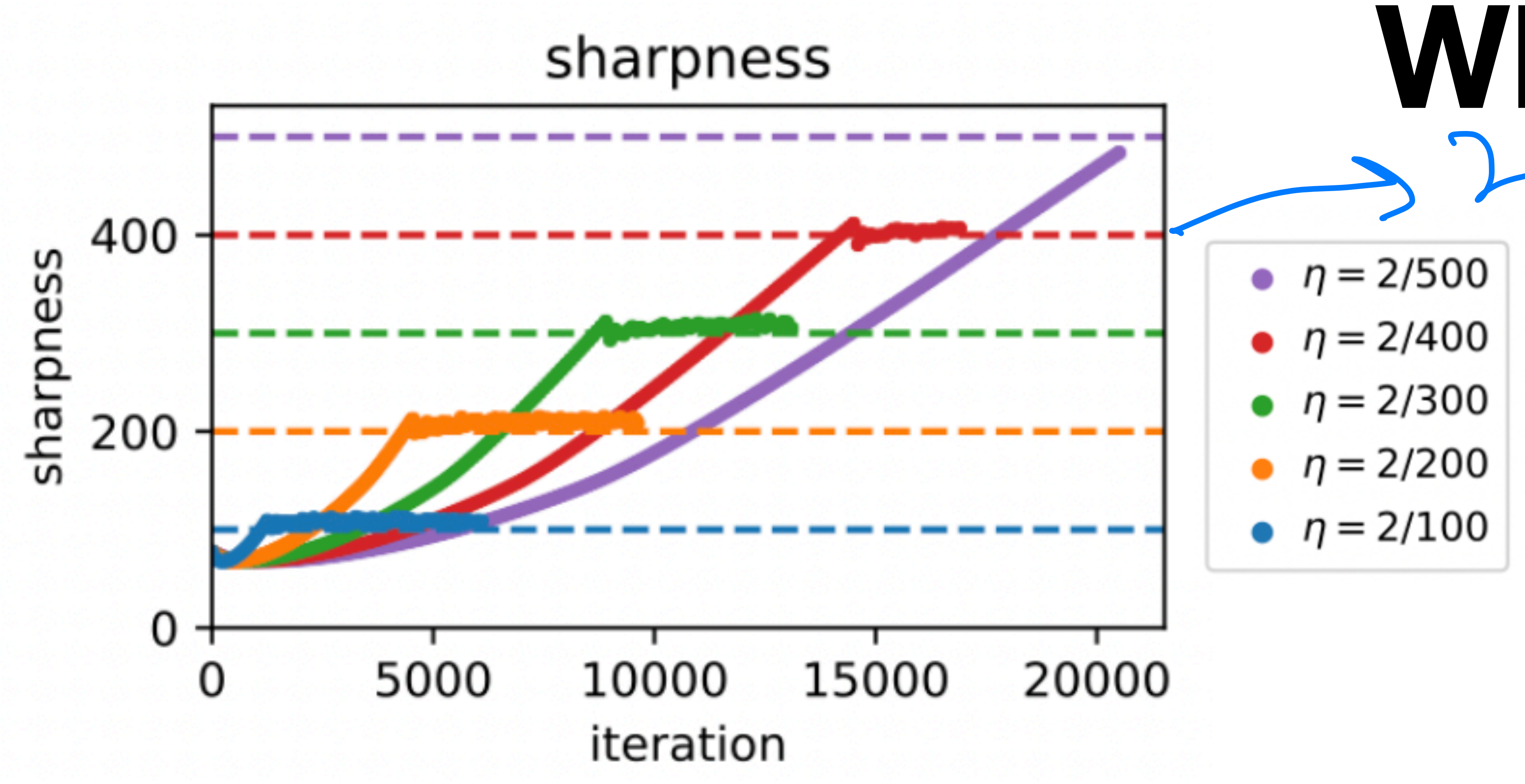
# Edge of Stability

- Define $\lambda^\star(\eta) = 2/\eta$, which is the maxima eigenvalue that leads to stable dynamics for lr $\eta$.
- If $\lambda(\theta) > \lambda^\star(\eta)$, then the dynamics should be unstable.



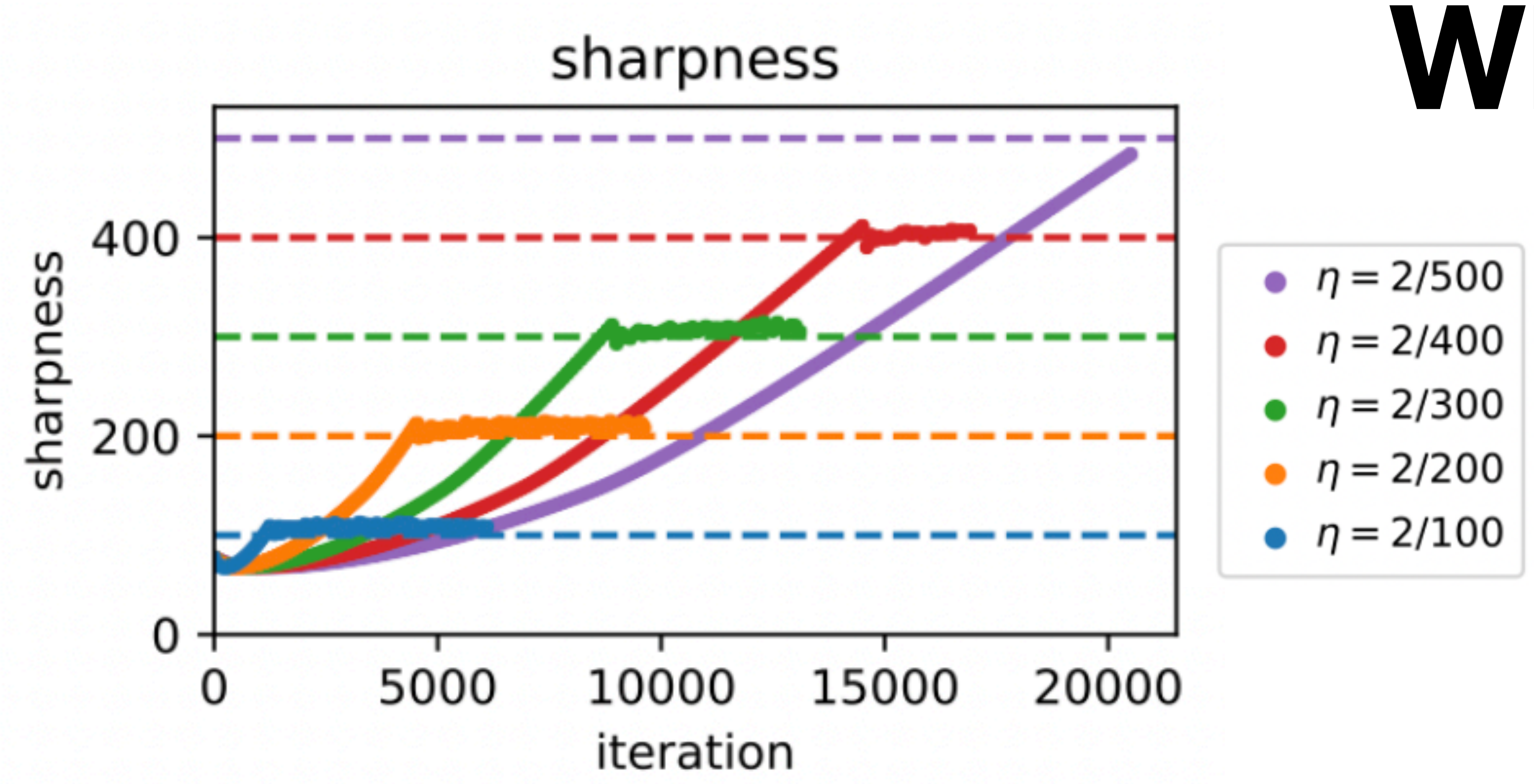

"Edge of Stability"
Cohen et al. 2021

# Why warmup?

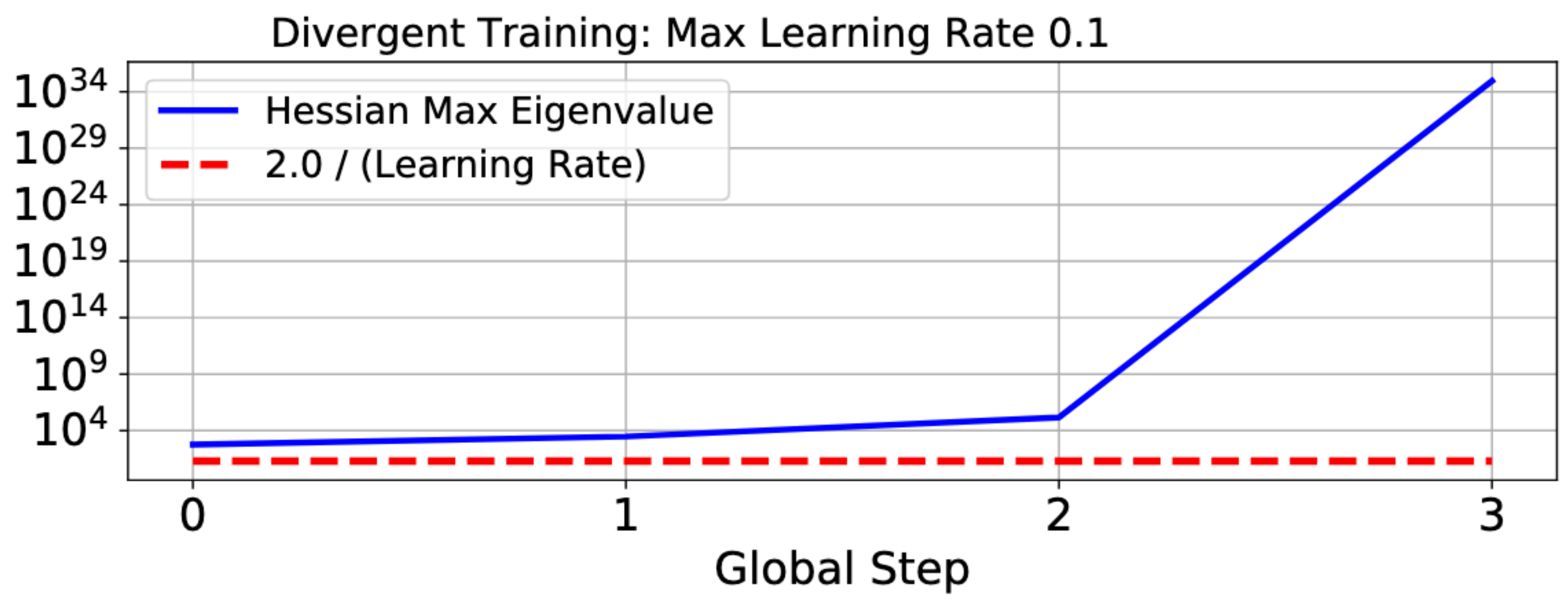



$2/\eta = \lambda^*(\eta)$

# Why warmup?

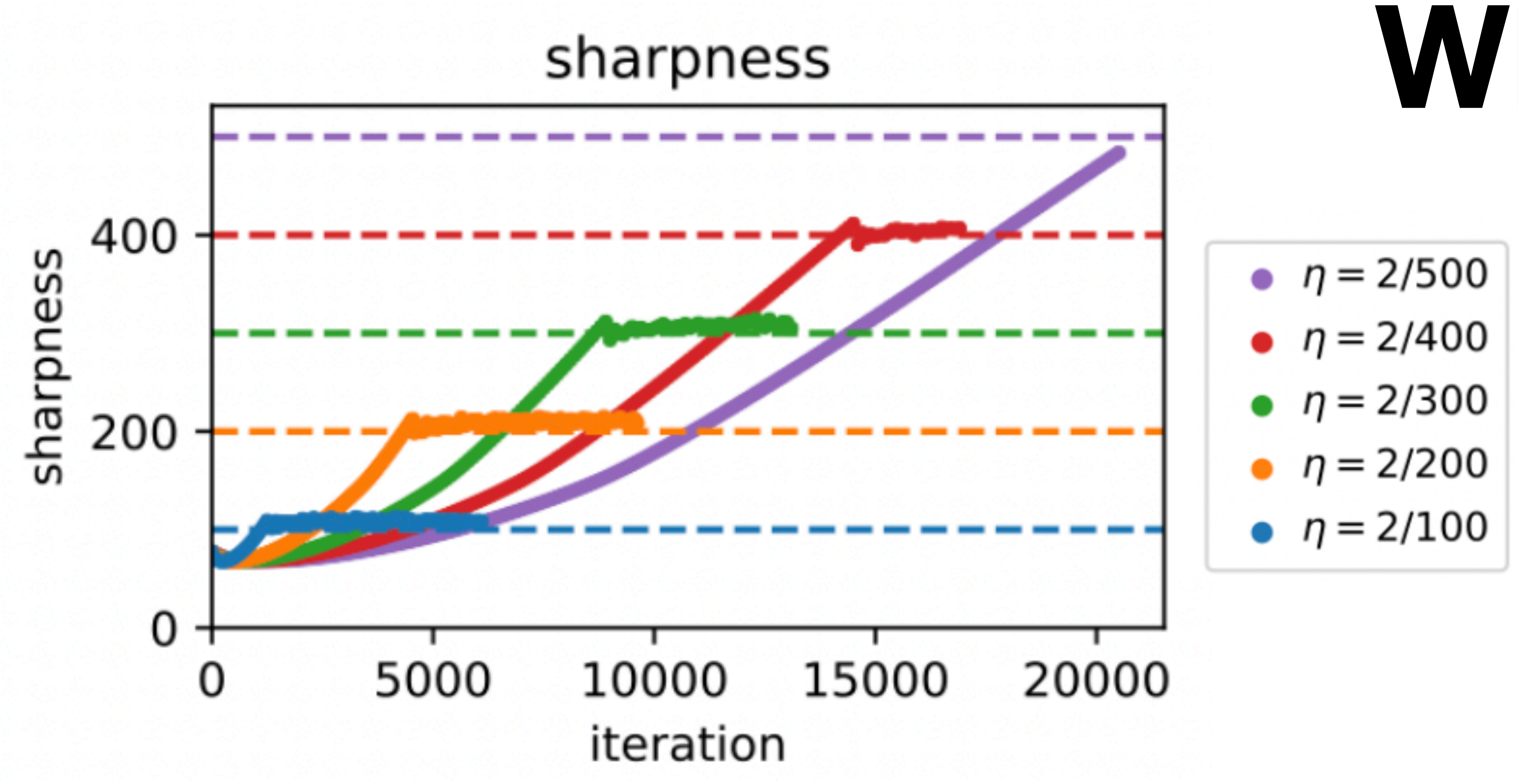


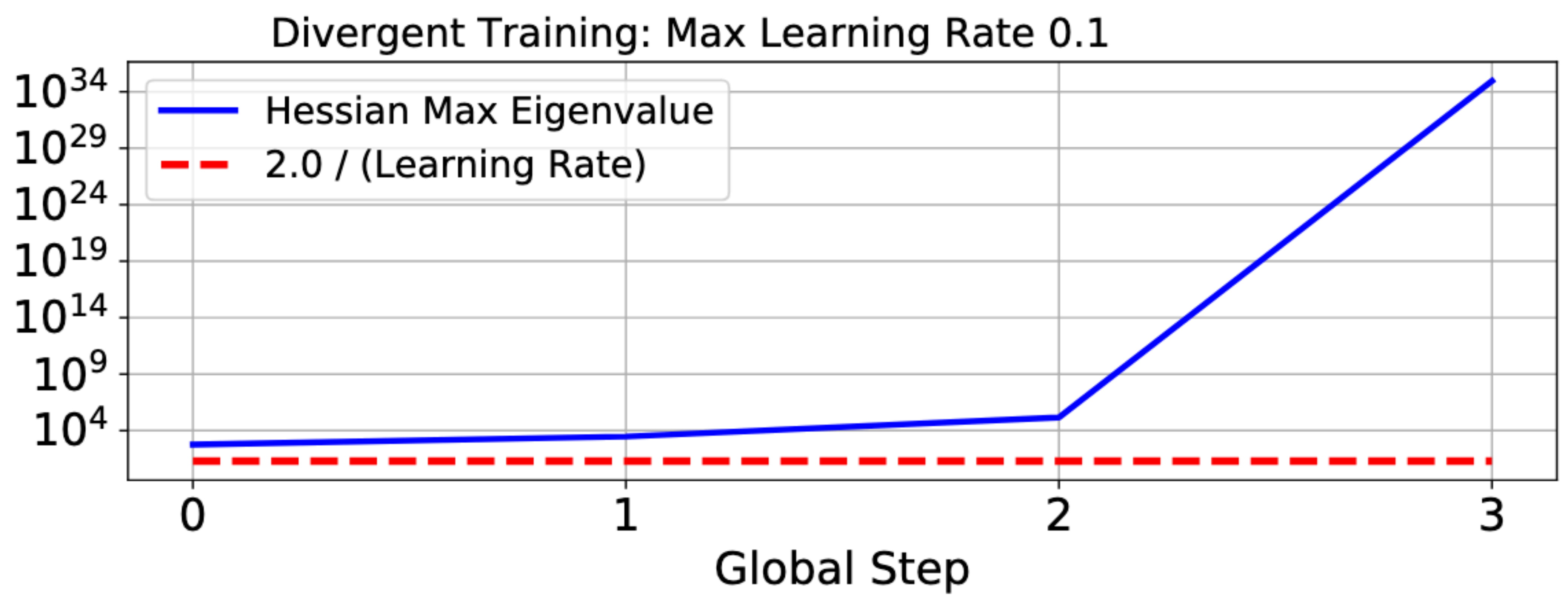
sharpness
sharpness
iteration
0
200
400
0
5000
10000
15000
20000
$\eta = 2/500$
$\eta = 2/400$
$\eta = 2/300$
$\eta = 2/200$
$\eta = 2/100$

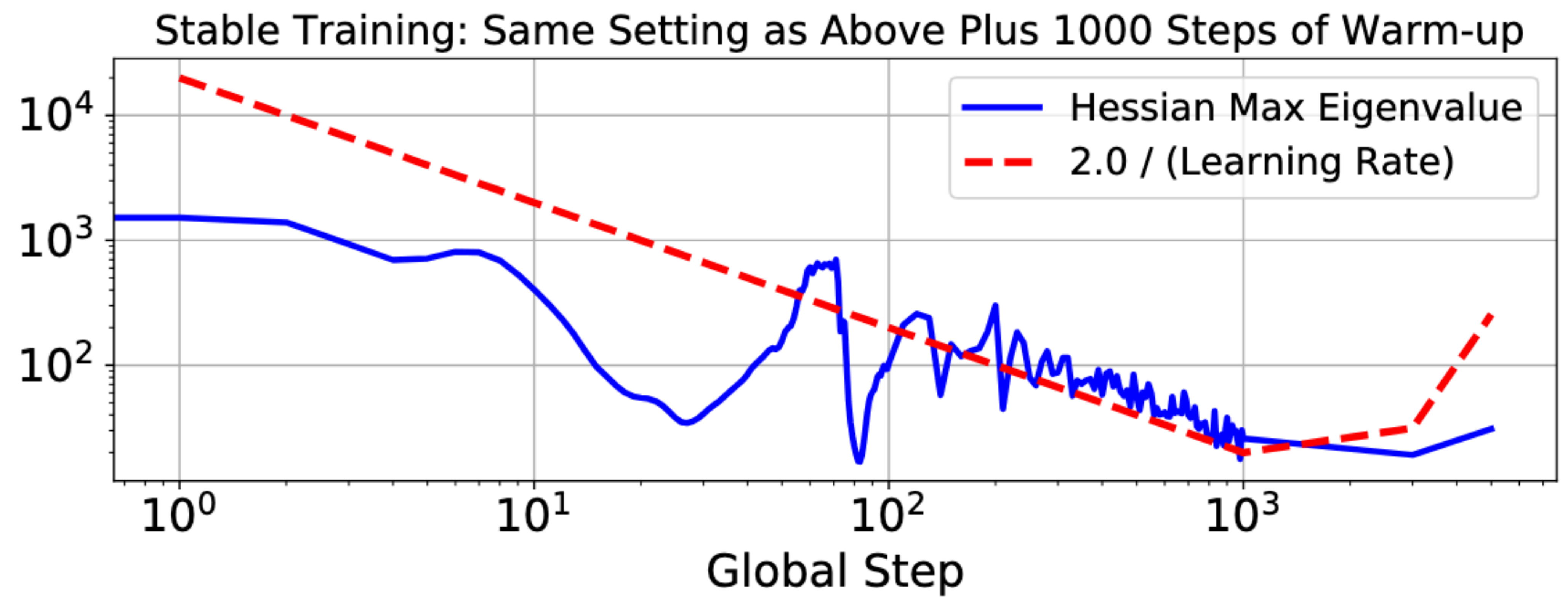



Divergent Training: Max Learning Rate 0.1
Hessian Max Eigenvalue
2.0 / (Learning Rate)
$10^{34}$
$10^{29}$
$10^{24}$
$10^{19}$
$10^{14}$
$10^{9}$
$10^{4}$
0
1
2
3
Global Step

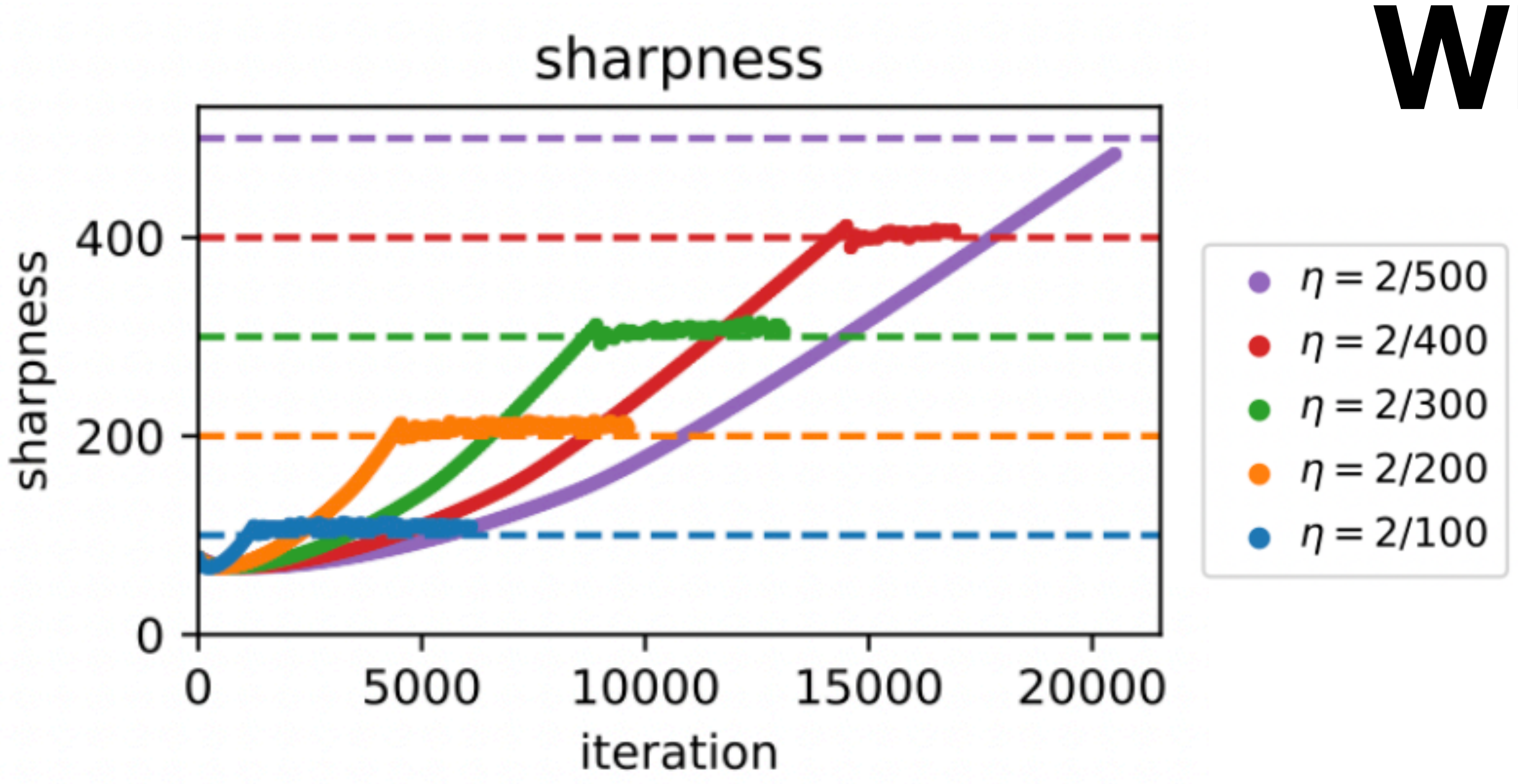
# Why warmup?

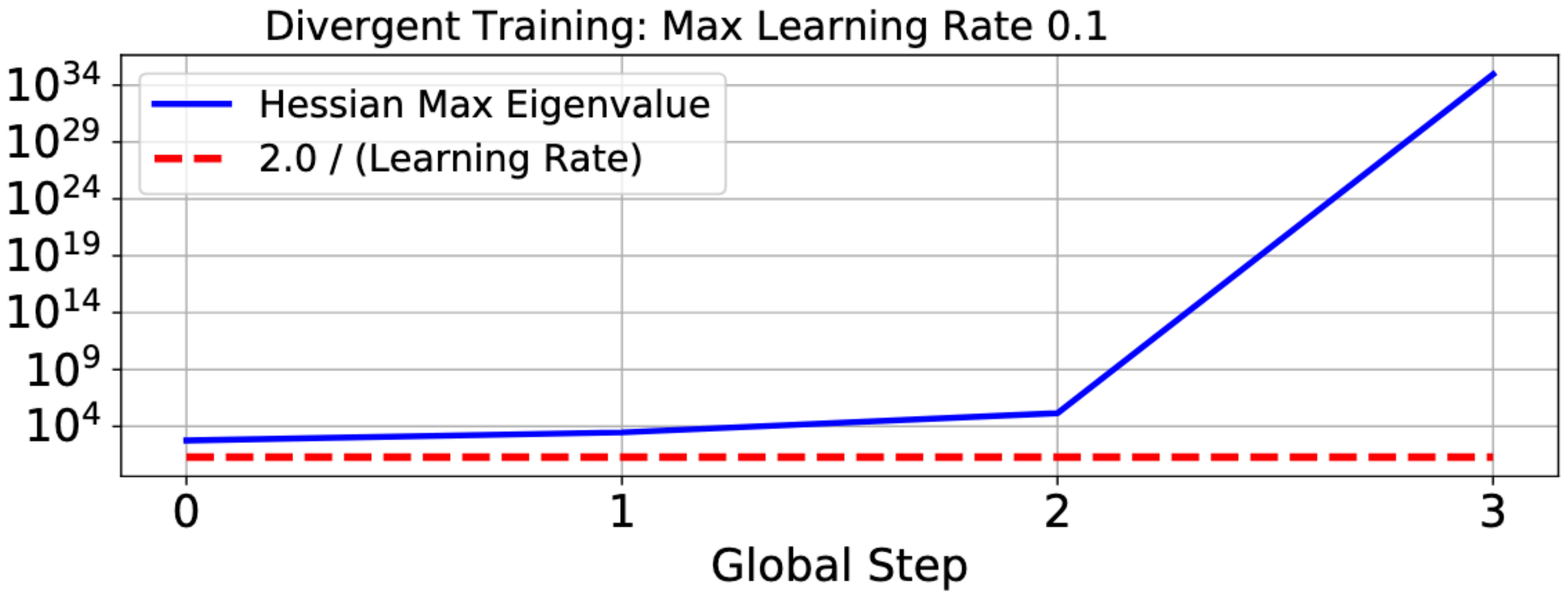

sharpness
sharpness
400
200
0
0
5000
10000
15000
20000
iteration
$\eta = 2/500$
$\eta = 2/400$
$\eta = 2/300$
$\eta = 2/200$
$\eta = 2/100$

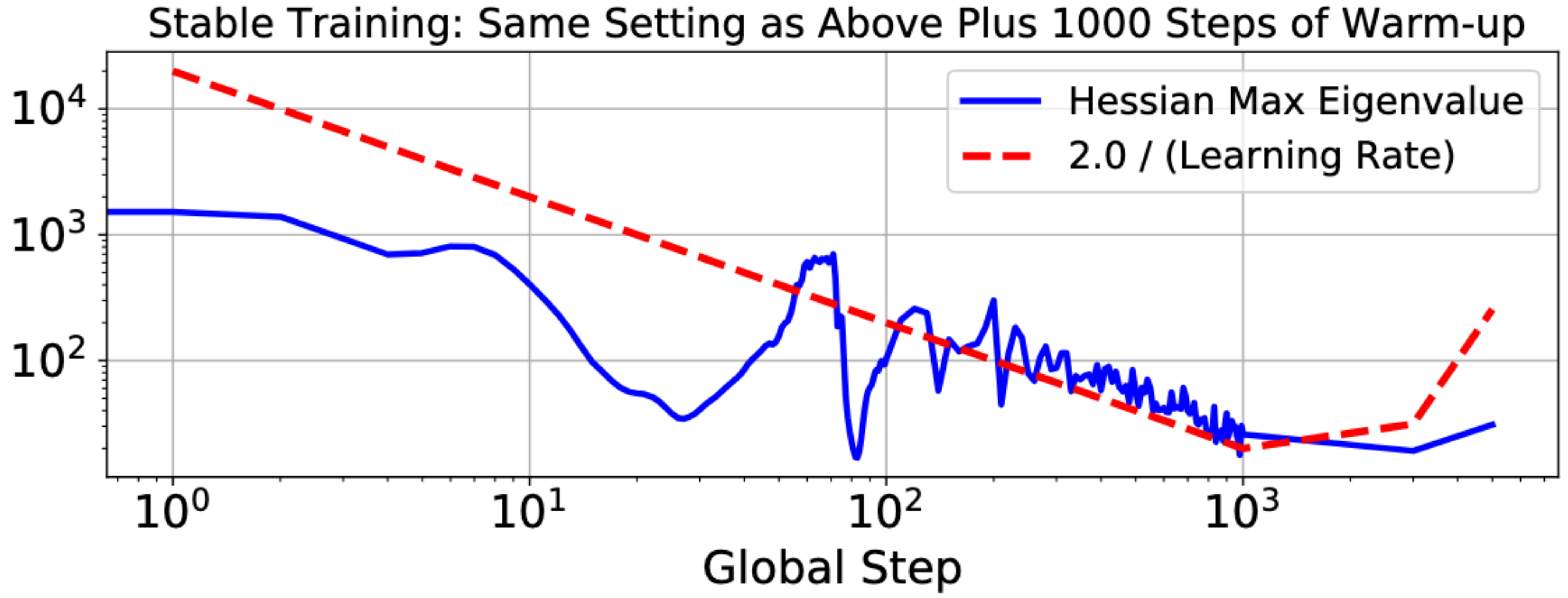

Divergent Training: Max Learning Rate 0.1
Hessian Max Eigenvalue
2.0 / (Learning Rate)
$10^{34}$
$10^{29}$
$10^{24}$
$10^{19}$
$10^{14}$
$10^{9}$
$10^{4}$
0
1
2
3
Global Step

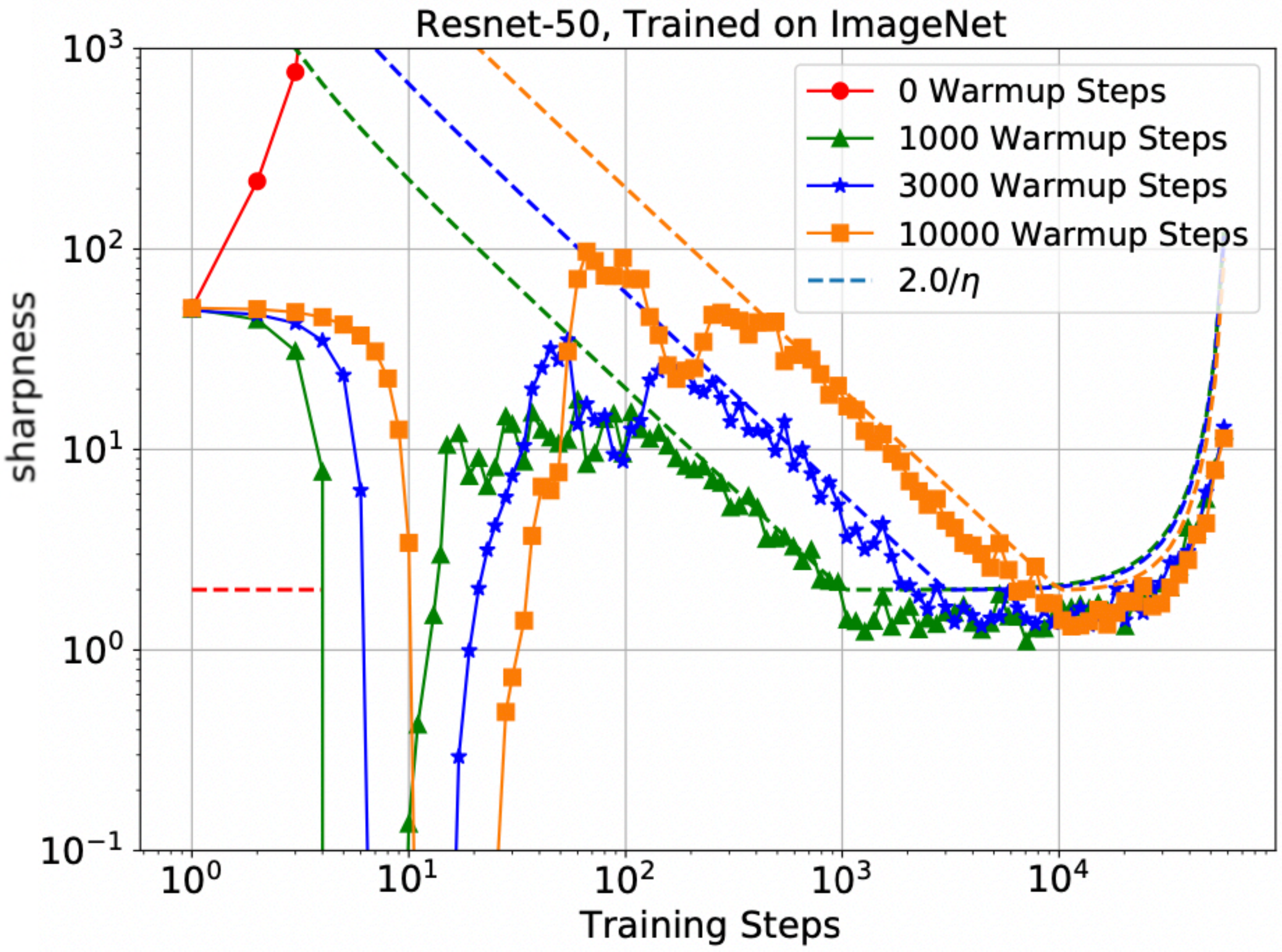

Stable Training: Same Setting as Above Plus 1000 Steps of Warm-up
Hessian Max Eigenvalue
2.0 / (Learning Rate)
$10^{4}$
$10^{3}$
$10^{2}$
$10^{0}$
$10^{1}$
$10^{2}$
$10^{3}$
Global Step

# Why warmup?


sharpness
400
200
0
sharpness
0
5000
10000
15000
20000
iteration
η = 2/500
η = 2/400
η = 2/300
η = 2/200
η = 2/100

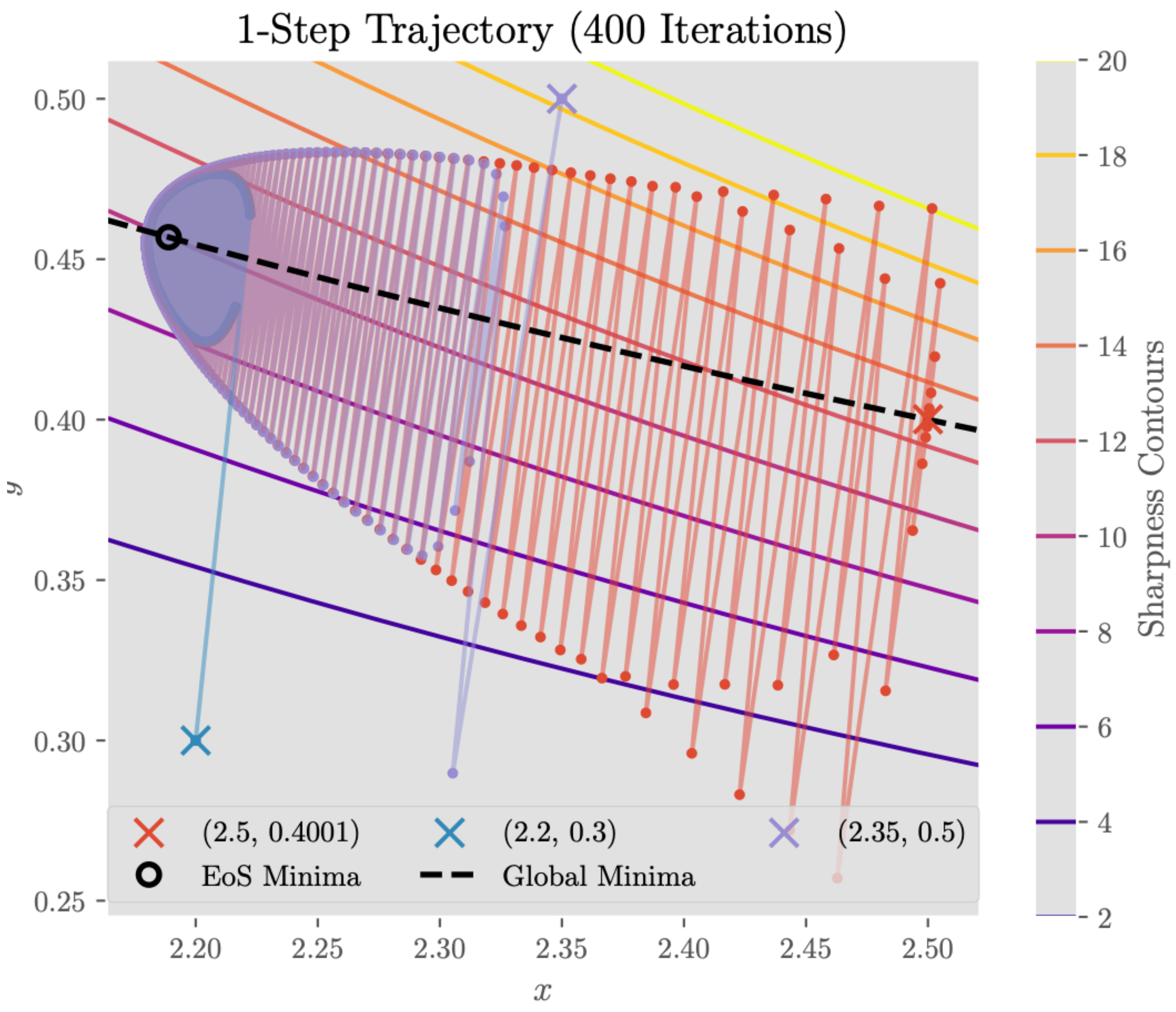

Divergent Training: Max Learning Rate 0.1
Hessian Max Eigenvalue
2.0 / (Learning Rate)
Global Step


Stable Training: Same Setting as Above Plus 1000 Steps of Warm-up
Hessian Max Eigenvalue
2.0 / (Learning Rate)
Global Step


Resnet-50, Trained on ImageNet
sharpness
0 Warmup Steps
1000 Warmup Steps
3000 Warmup Steps
10000 Warmup Steps
2.0/η
Training Steps

# Edge of Stability - Toy Model

$$\mathscr{L}(x,y) = (1 - x^2y^2)^2$$



# Edge of Stability - A generic explanation

- $\mathscr{L}(\theta)$ is the loss function. Let $S(\theta)$ represent the sharpness, i.e, $S(\theta) = \lambda_{max}[\nabla^2 \mathscr{L}(\theta)]$.

# Edge of Stability - A generic explanation

- $\mathscr{L}(\theta)$ is the loss function. Let $S(\theta)$ represent the sharpness, i.e, $S(\theta) = \lambda_{max}[\nabla^2 \mathscr{L}(\theta)]$.
- Stage 1: Suppose progressive sharpening has occurred and that we reach a point $\theta_t$ where $\eta = 2/S(\theta_t)$, i.e. we are at point where we should are unstable (and oscillating).
  Let $u$ be the largest eigenvector of $\nabla^2 \mathscr{L}(\theta_t)$.

# Edge of Stability - A generic explanation

- $\mathscr{L}(\theta)$ is the loss function. Let $S(\theta)$ represent the sharpness, i.e, $S(\theta) = \lambda_{max}[\nabla^2 \mathscr{L}(\theta)]$.
- Stage 1: Suppose progressive sharpening has occurred and that we reach a point $\theta_t$ where $\eta = 2/S(\theta_t)$, i.e. we are at point where we should are unstable (and oscillating).
  Let $u$ be the largest eigenvector of $\nabla^2 \mathscr{L}(\theta_t)$.
- Stage 2: Why don't we diverge?

# Edge of Stability - A generic explanation

- $\mathscr{L}(\theta)$ is the loss function. Let $S(\theta)$ represent the sharpness, i.e, $S(\theta) = \lambda_{max}[\nabla^2 \mathscr{L}(\theta)]$.
- Stage 1: Suppose progressive sharpening has occurred and that we reach a point $\theta_t$ where $\eta = 2/S(\theta_t)$, i.e. we are at point where we should are unstable (and oscillating).
  Let $u$ be the largest eigenvector of $\nabla^2 \mathscr{L}(\theta_t)$.
- Stage 2: Why don't we diverge?
  - Let us consider a perturbation of $\theta_t$ in the $u$ direction. For $\alpha > 0$,
    $$\nabla_\theta L(\theta_t + \alpha u) \approx \nabla_\theta L(\theta_t) + \alpha \nabla^2_\theta L(\theta_t) u + (\alpha^2/2) \nabla^3_\theta L(\theta_t) \cdot (u \otimes u)$$
    $$= \nabla_\theta L(\theta_t) + \alpha S(\theta_t) u + (\alpha^2/2) \nabla S(\theta_t)$$
    where the last step uses $\nabla^3_\theta L(\theta_t) \cdot (u \otimes u) = \nabla_\theta S(\theta_t)$. (Do you see why??)

# Edge of Stability - A generic explanation

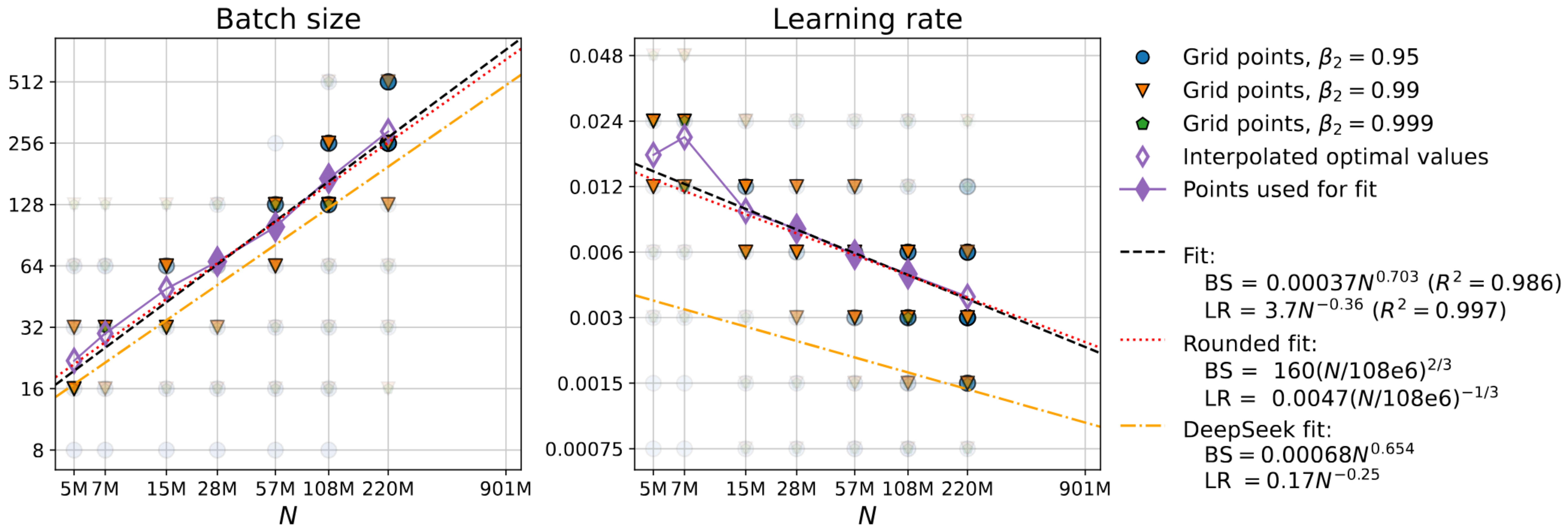
- $\mathscr{L}(\theta)$ is the loss function. Let $S(\theta)$ represent the sharpness, i.e, $S(\theta) = \lambda_{max}[\nabla^2 \mathscr{L}(\theta)]$.
- Stage 1: Suppose progressive sharpening has occurred and that we reach a point $\theta_t$ where $\eta = 2/S(\theta_t)$, i.e. we are at point where we should are unstable (and oscillating).
  Let $u$ be the largest eigenvector of $\nabla^2 \mathscr{L}(\theta_t)$.
- Stage 2: Why don't we diverge?
  - Let us consider a perturbation of $\theta_t$ in the $u$ direction. For $\alpha > 0$,
    $$\nabla_\theta L(\theta_t + \alpha u) \approx \nabla_\theta L(\theta_t) + \alpha \nabla^2_\theta L(\theta_t)u + (\alpha^2/2)\nabla^3_\theta L(\theta_t)\cdot(u \otimes u)$$
    $$= \nabla_\theta L(\theta_t) + \alpha S(\theta_t)u + (\alpha^2/2)\nabla S(\theta_t)$$
    where the last step uses $\nabla^3_\theta L(\theta_t)\cdot(u \otimes u) = \nabla_\theta S(\theta_t)$. (Do you see why??)
  - Therefore, for large $\alpha$, a gradient step after a large perturbation makes $S(\theta)$ smaller due to the $(\alpha^2/2)\nabla S(\theta_t)$ term.

# Edge of Stability - A generic explanation

- $\mathscr{L}(\theta)$ is the loss function. Let $S(\theta)$ represent the sharpness, i.e, $S(\theta) = \lambda_{max}[\nabla^2 \mathscr{L}(\theta)]$.
- Stage 1: Suppose progressive sharpening has occurred and that we reach a point $\theta_t$ where $\eta = 2/S(\theta_t)$, i.e. we are at point where we should are unstable (and oscillating).
  Let $u$ be the largest eigenvector of $\nabla^2 \mathscr{L}(\theta_t)$.
- Stage 2: Why don't we diverge?
  - Let us consider a perturbation of $\theta_t$ in the $u$ direction. For $\alpha > 0$,
    $$\begin{aligned}\nabla_\theta L(\theta_t + \alpha u) &\approx \nabla_\theta L(\theta_t) + \alpha \nabla^2_\theta L(\theta_t) u + (\alpha^2/2) \nabla^3_\theta L(\theta_t) \cdot (u \otimes u) \\ &= \nabla_\theta L(\theta_t) + \alpha S(\theta_t) u + (\alpha^2/2) \nabla S(\theta_t)\end{aligned}$$
    where the last step uses $\nabla^3_\theta L(\theta_t) \cdot (u \otimes u) = \nabla_\theta S(\theta_t)$. (Do you see why??)
  - Therefore, for large $\alpha$, a gradient step after a large perturbation makes $S(\theta)$ smaller due to the $(\alpha^2/2) \nabla S(\theta_t)$ term.
  - This makes the dynamics more stable because $2/S(\theta)$ increases after the perturbation.

# References

- Tay et al. 2022: https://arxiv.org/abs/2207.10551
- Clark et al. 2022: https://arxiv.org/abs/2202.01169
- Dosovitskiy et al. 2021: https://arxiv.org/pdf/2010.11929
- Other papers: “Getting ViT in Shape”, “SCALE EFFICIENTLY: INSIGHTS FROM PRE-TRAINING AND FINE-TUNING TRANSFORMERS”.




Porian et al. 2024