Learned optimizers: why they're the future, why they’re hard, and what they can do now



Luke Metz



Niru Maheswaranathan



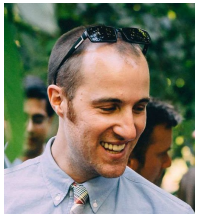
Daniel Freeman



Jeremy Nixon



Ruoxi Sun



Jascha Sohl-Dickstein

Outline

- The coming meta-learning revolution
- What is a learned optimizer?
- Open problems in learned optimizers (with a few solutions)
  - Generalization to new optimization tasks
  - Interpreting mechanisms of action
  - Engineering inductive biases
  - Chaotic outer-loss landscapes
  - Computational cost
  - Truncated unroll bias
- What learned optimizers can achieve now
  - Outperform hand-designed optimizers on narrow domain, or with limited hyperparameter tuning budget
- Novel capabilities
  - Unsupervised representation learning
- `learned_optimizers` library + colab demo

The deep learning revolution

- **Researchers used to build careers out of hand-designing image features.**
- **Then came the deep learning revolution.**



- **Given enough data and compute, learned functions outperform even well motivated hand-designed heuristic functions.**
- **… and now (almost) no-one has a job hand-designing image features.**

The meta-learning revolution

- **We still hand design our optimizers, loss functions, architectures, and regularizers.**
- **Even when well motivated, these are heuristics.**

Example: in classification, hand-designed optimizers minimize the **wrong loss** on the **wrong dataset**

- We care about **accuracy** on the **test** dataset
- We train on **cross entropy** on the **train** dataset

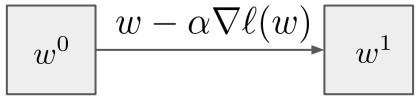
engineering to learn → learning to learn

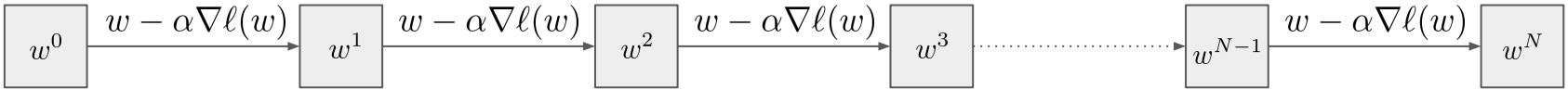
SGD (Robbins et. al. 1951, Bottou 2010)
Autoencoders (Hinton et. al. 2006)

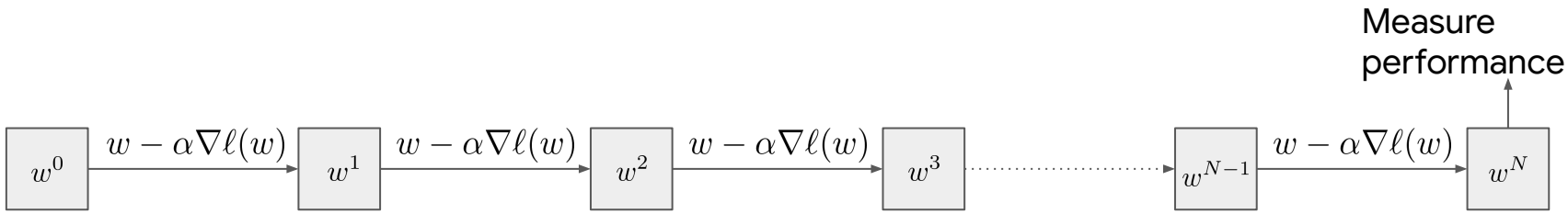
Learning To Learn (Hochreiter et. al. 2001)
Learned Optimizers (Bengio et. al. 1990,
Andrychowicz et. al. 2016, Li et. al. 2016,
Wichrowska et. al. 2017, Metz et al. 2019)
Unsupervised Learning (Metz, et. al. 2019)
Neural Architecture Search (Zoph, Le, 2016)

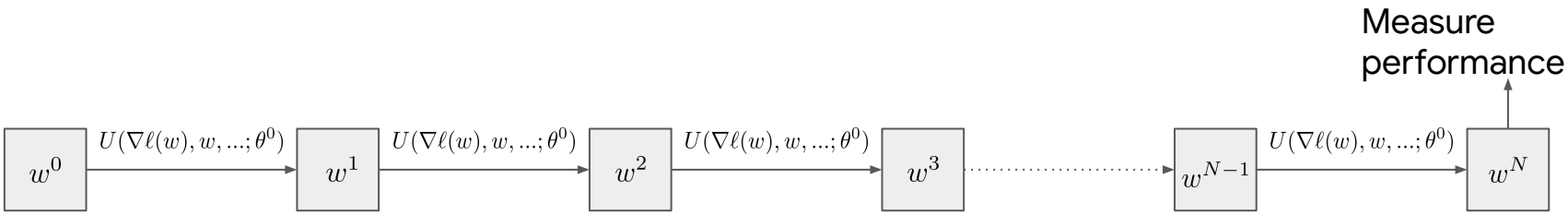
What is a learned optimizer?

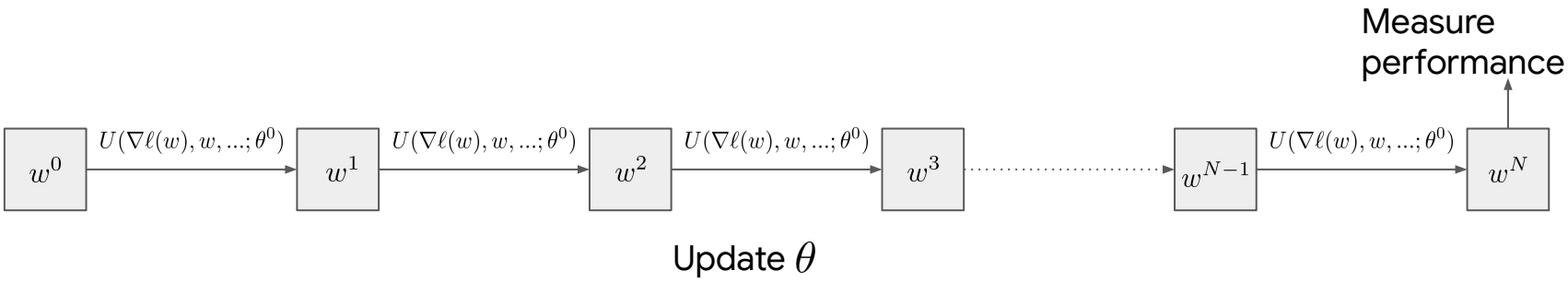

w^0

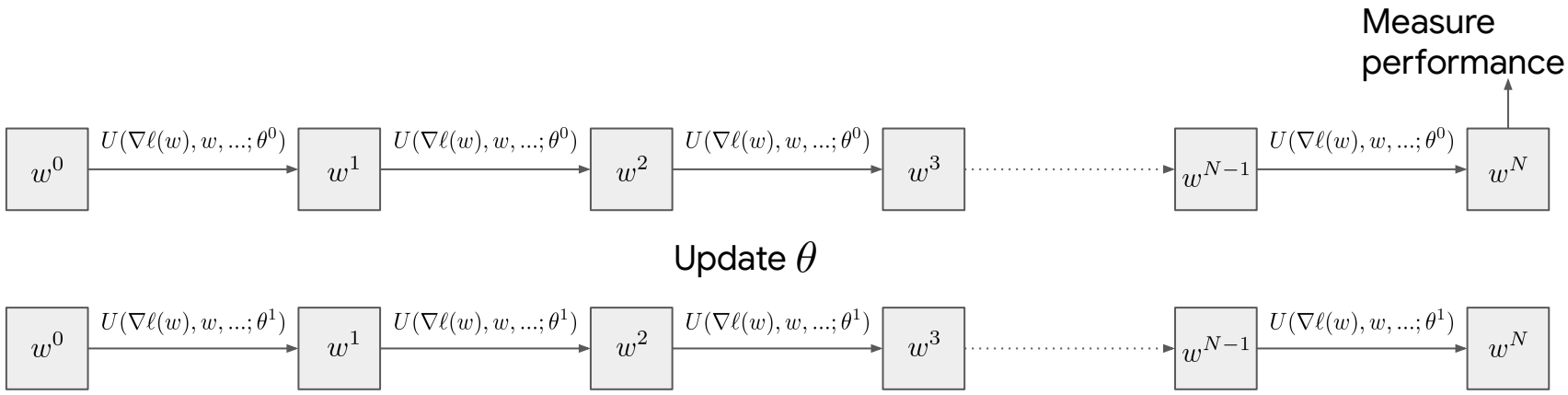

w^0
$w - \alpha\nabla\ell(w)$
w^1

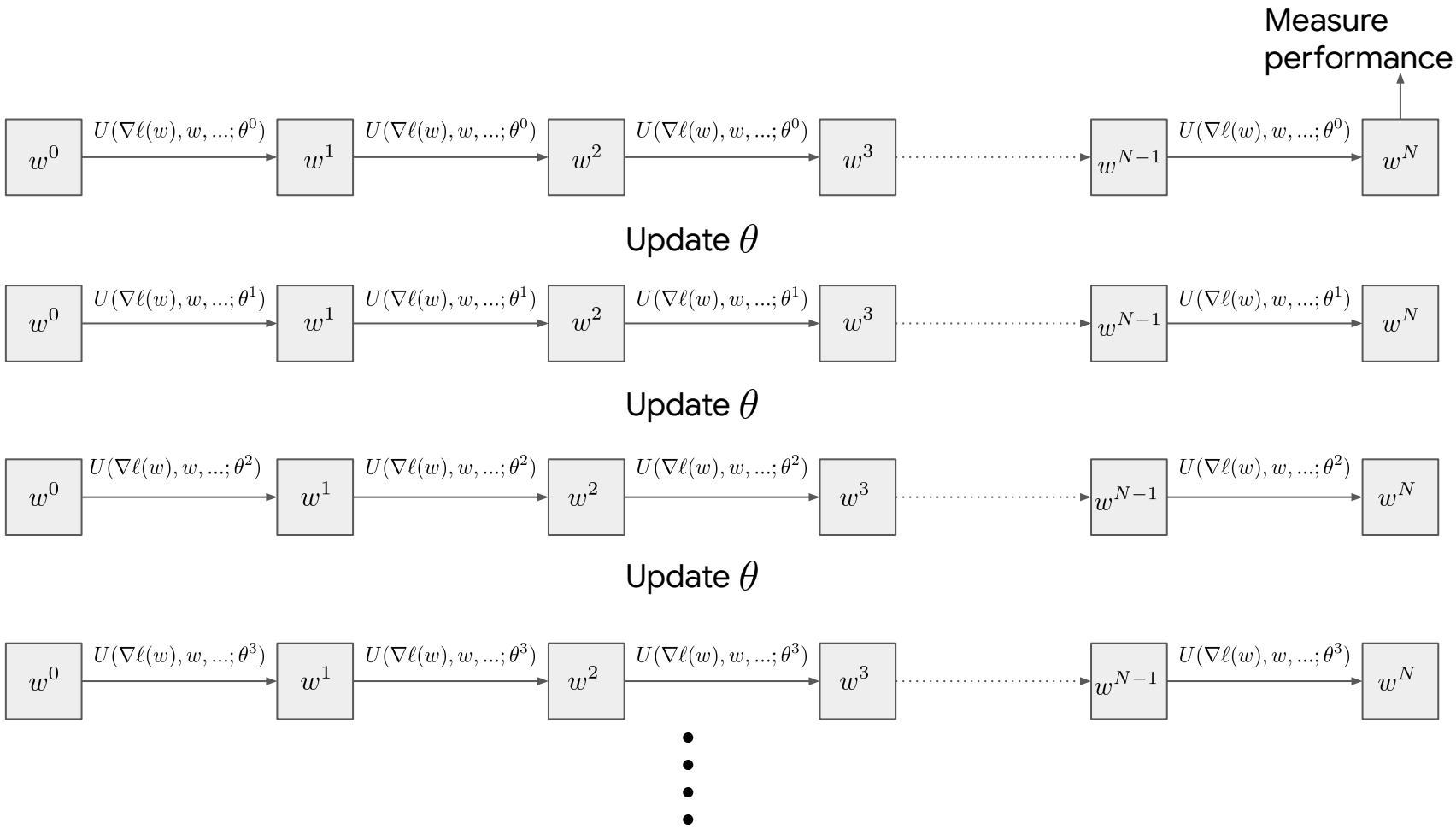

w^0
$w - \alpha\nabla\ell(w)$
w^1
$w - \alpha\nabla\ell(w)$
w^2
$w - \alpha\nabla\ell(w)$
w^3
w^{N-1}
$w - \alpha\nabla\ell(w)$
w^N

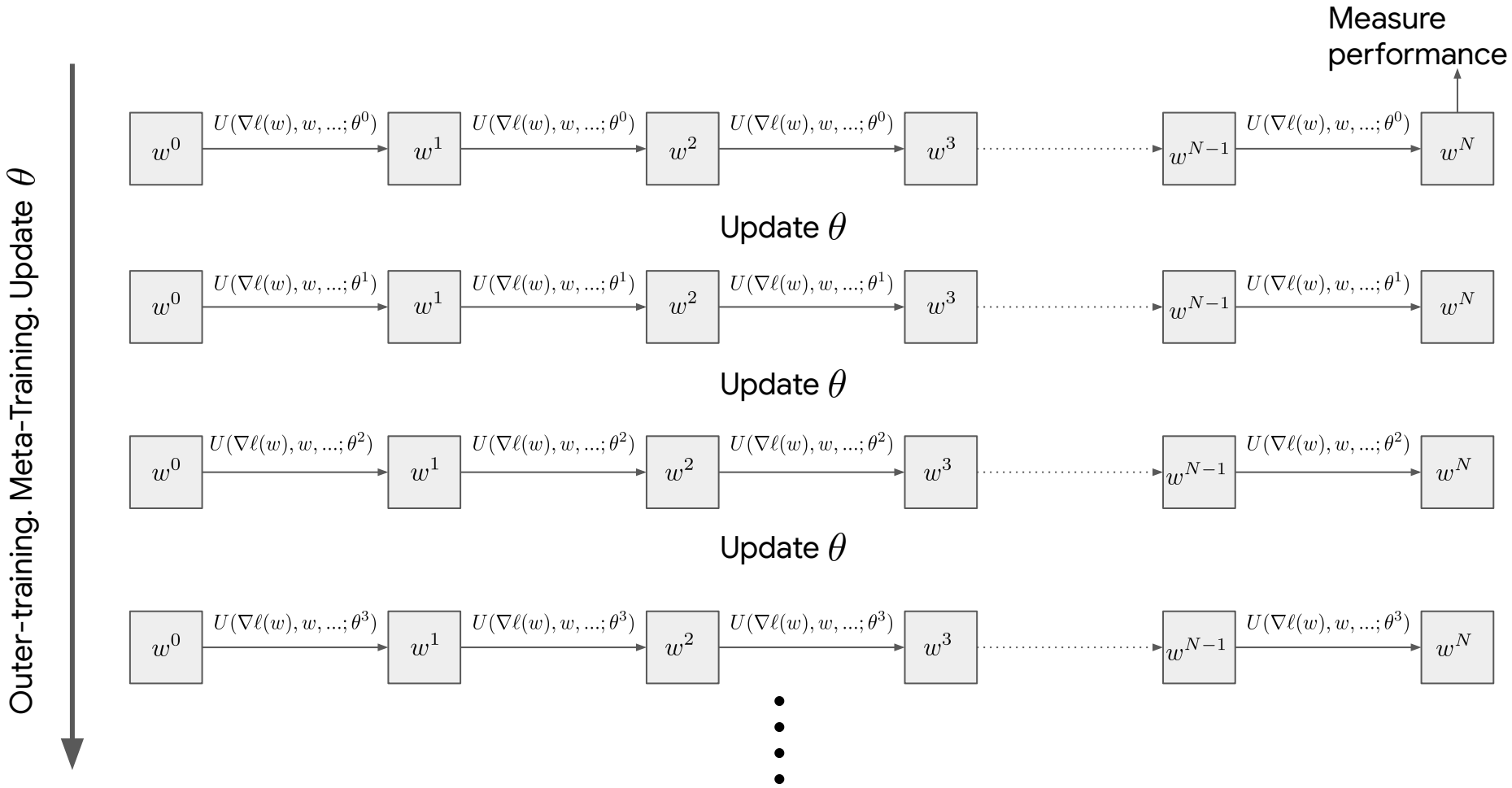

Measure performance
w^0
$w - \alpha \nabla \ell(w)$
w^1
$w - \alpha \nabla \ell(w)$
w^2
$w - \alpha \nabla \ell(w)$
w^3
w^{N-1}
$w - \alpha \nabla \ell(w)$
w^N

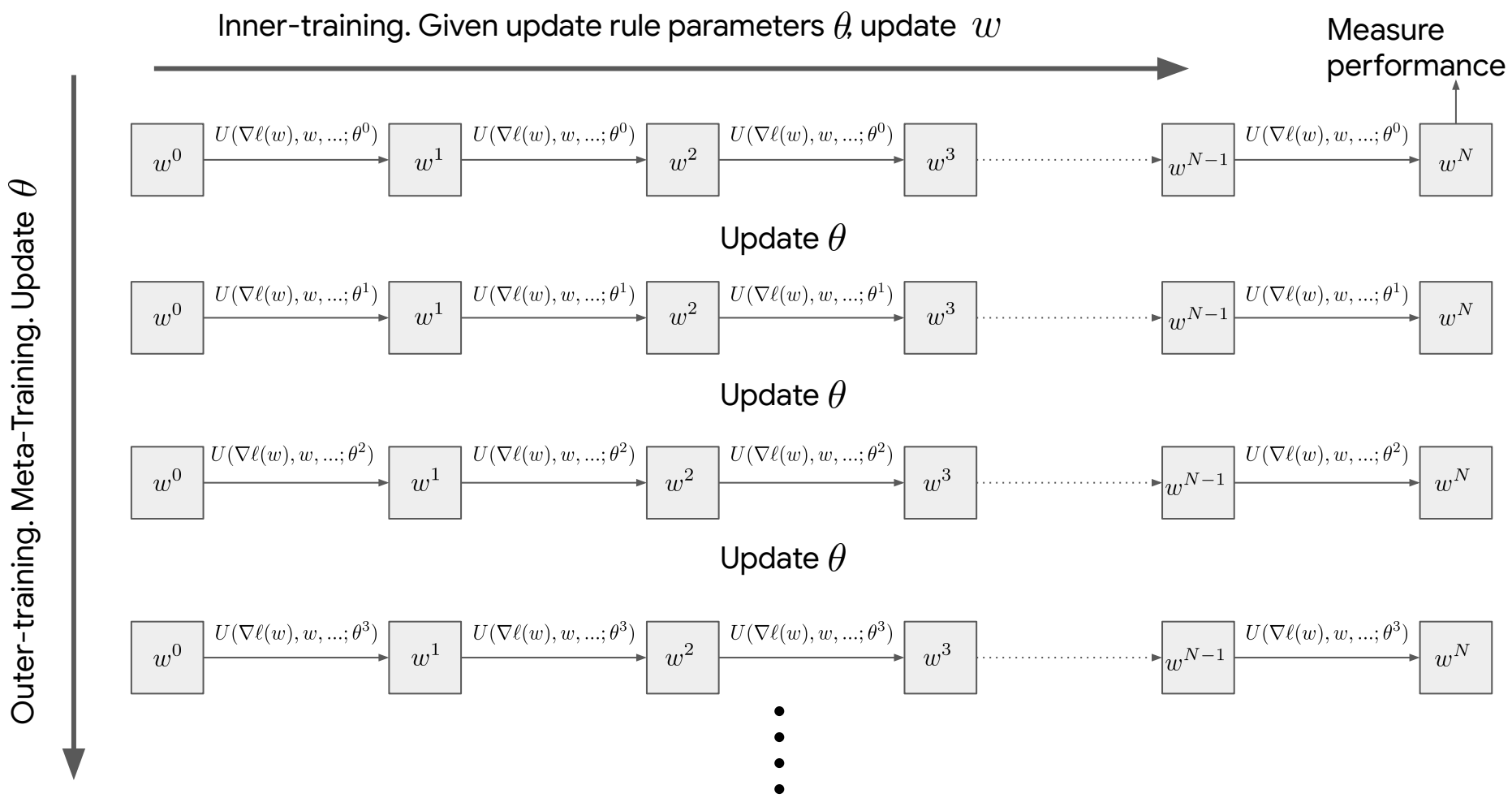

Measure performance
w^0
$U(\nabla\ell(w), w, ...; \theta^0)$
w^1
$U(\nabla\ell(w), w, ...; \theta^0)$
w^2
$U(\nabla\ell(w), w, ...; \theta^0)$
w^3
w^{N-1}
$U(\nabla\ell(w), w, ...; \theta^0)$
w^N


Measure performance
w^0
$U(\nabla\ell(w), w, ...; \theta^0)$
w^1
$U(\nabla\ell(w), w, ...; \theta^0)$
w^2
$U(\nabla\ell(w), w, ...; \theta^0)$
w^3
w^{N-1}
$U(\nabla\ell(w), w, ...; \theta^0)$
w^N
Update θ


Measure performance
w^0
$U(\nabla\ell(w), w, ...; \theta^0)$
w^1
$U(\nabla\ell(w), w, ...; \theta^0)$
w^2
$U(\nabla\ell(w), w, ...; \theta^0)$
w^3
w^{N-1}
$U(\nabla\ell(w), w, ...; \theta^0)$
w^N
Update θ
w^0
$U(\nabla\ell(w), w, ...; \theta^1)$
w^1
$U(\nabla\ell(w), w, ...; \theta^1)$
w^2
$U(\nabla\ell(w), w, ...; \theta^1)$
w^3
w^{N-1}
$U(\nabla\ell(w), w, ...; \theta^1)$
w^N

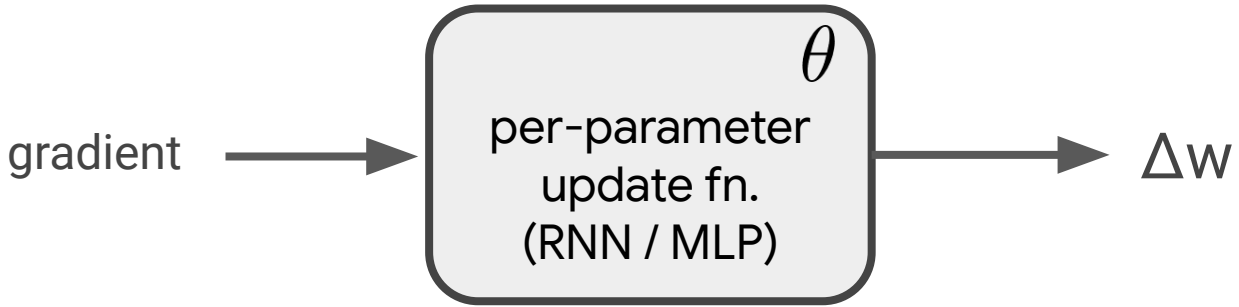

Measure performance
w^0
$U(\nabla\ell(w), w, ...; \theta^0)$
w^1
$U(\nabla\ell(w), w, ...; \theta^0)$
w^2
$U(\nabla\ell(w), w, ...; \theta^0)$
w^3
w^{N-1}
$U(\nabla\ell(w), w, ...; \theta^0)$
w^N
Update θ
w^0
$U(\nabla\ell(w), w, ...; \theta^1)$
w^1
$U(\nabla\ell(w), w, ...; \theta^1)$
w^2
$U(\nabla\ell(w), w, ...; \theta^1)$
w^3
w^{N-1}
$U(\nabla\ell(w), w, ...; \theta^1)$
w^N
Update θ
w^0
$U(\nabla\ell(w), w, ...; \theta^2)$
w^1
$U(\nabla\ell(w), w, ...; \theta^2)$
w^2
$U(\nabla\ell(w), w, ...; \theta^2)$
w^3
w^{N-1}
$U(\nabla\ell(w), w, ...; \theta^2)$
w^N
Update θ
w^0
$U(\nabla\ell(w), w, ...; \theta^3)$
w^1
$U(\nabla\ell(w), w, ...; \theta^3)$
w^2
$U(\nabla\ell(w), w, ...; \theta^3)$
w^3
w^{N-1}
$U(\nabla\ell(w), w, ...; \theta^3)$
w^N


Measure performance
Outer-training. Meta-Training. Update θ
w^0
$U(\nabla\ell(w), w, ...; \theta^0)$
w^1
$U(\nabla\ell(w), w, ...; \theta^0)$
w^2
$U(\nabla\ell(w), w, ...; \theta^0)$
w^3
w^{N-1}
$U(\nabla\ell(w), w, ...; \theta^0)$
w^N
Update θ
w^0
$U(\nabla\ell(w), w, ...; \theta^1)$
w^1
$U(\nabla\ell(w), w, ...; \theta^1)$
w^2
$U(\nabla\ell(w), w, ...; \theta^1)$
w^3
w^{N-1}
$U(\nabla\ell(w), w, ...; \theta^1)$
w^N
Update θ
w^0
$U(\nabla\ell(w), w, ...; \theta^2)$
w^1
$U(\nabla\ell(w), w, ...; \theta^2)$
w^2
$U(\nabla\ell(w), w, ...; \theta^2)$
w^3
w^{N-1}
$U(\nabla\ell(w), w, ...; \theta^2)$
w^N
Update θ
w^0
$U(\nabla\ell(w), w, ...; \theta^3)$
w^1
$U(\nabla\ell(w), w, ...; \theta^3)$
w^2
$U(\nabla\ell(w), w, ...; \theta^3)$
w^3
w^{N-1}
$U(\nabla\ell(w), w, ...; \theta^3)$
w^N


Inner-training. Given update rule parameters θ, update w
Measure performance
Outer-training. Meta-Training. Update θ
w^0
w^1
w^2
w^3
w^{N-1}
w^N
$U(\nabla\ell(w), w, ...; \theta^0)$
$U(\nabla\ell(w), w, ...; \theta^1)$
$U(\nabla\ell(w), w, ...; \theta^2)$
$U(\nabla\ell(w), w, ...; \theta^3)$
Update θ
Update θ
Update θ

Example learned optimizer architecture: per-parameter NN

Parametric update (U):



Same RNN/MLP applied to every parameter

Andrychowicz, Marcin, et al. "Learning to learn by gradient descent by gradient descent." *Neurips*. 2016.

Metz, Luke, et al. "Understanding and correcting pathologies in the training of learned optimizers." *ICML 2019*

Why is training learned optimizers hard?

Problem: Generalization to new optimization tasks. Generalization in deep learning typically requires a diverse dataset of thousands to millions of examples

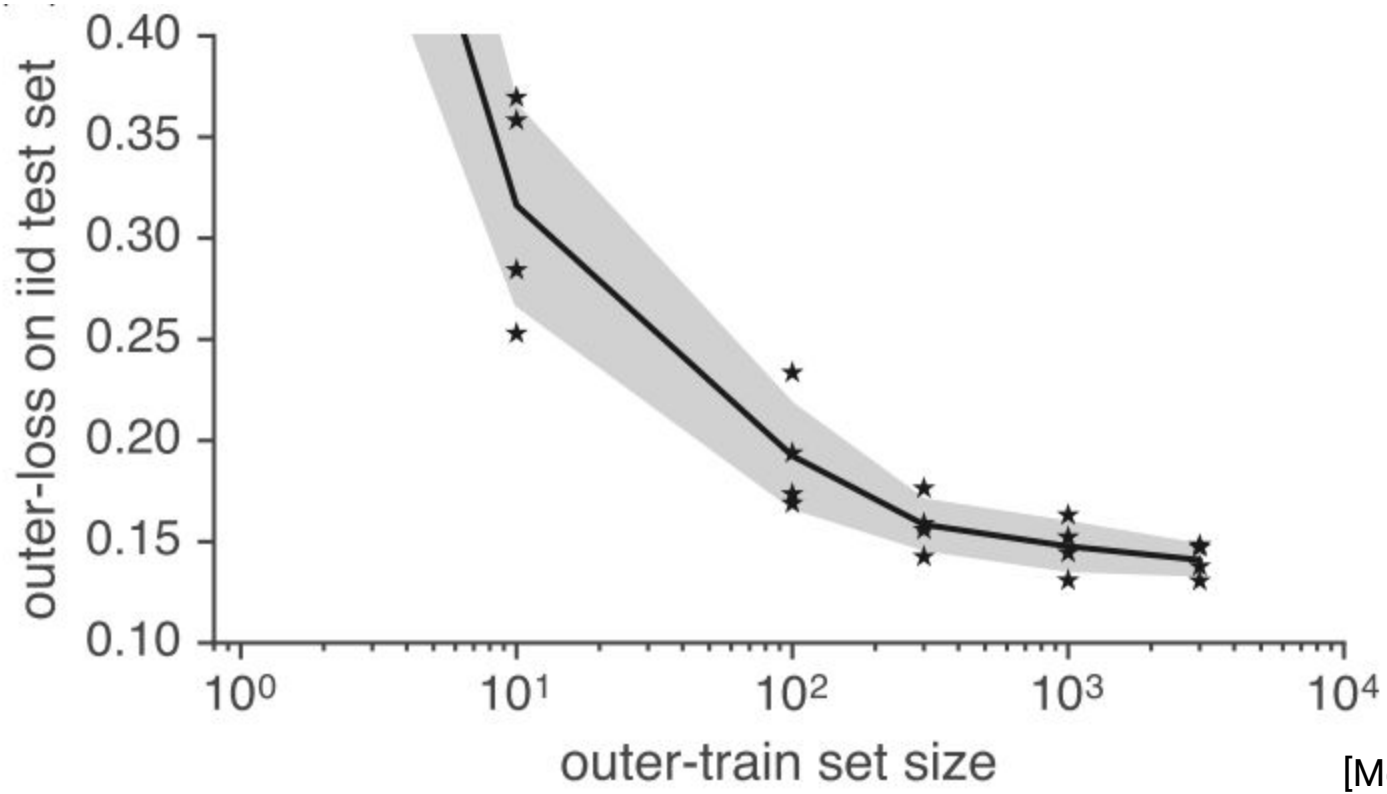
Solution: Construct a large “dataset” of optimization tasks

TSNE embedding of 1162 tasks for training learned optimizers



[Metz, et al., 2020]

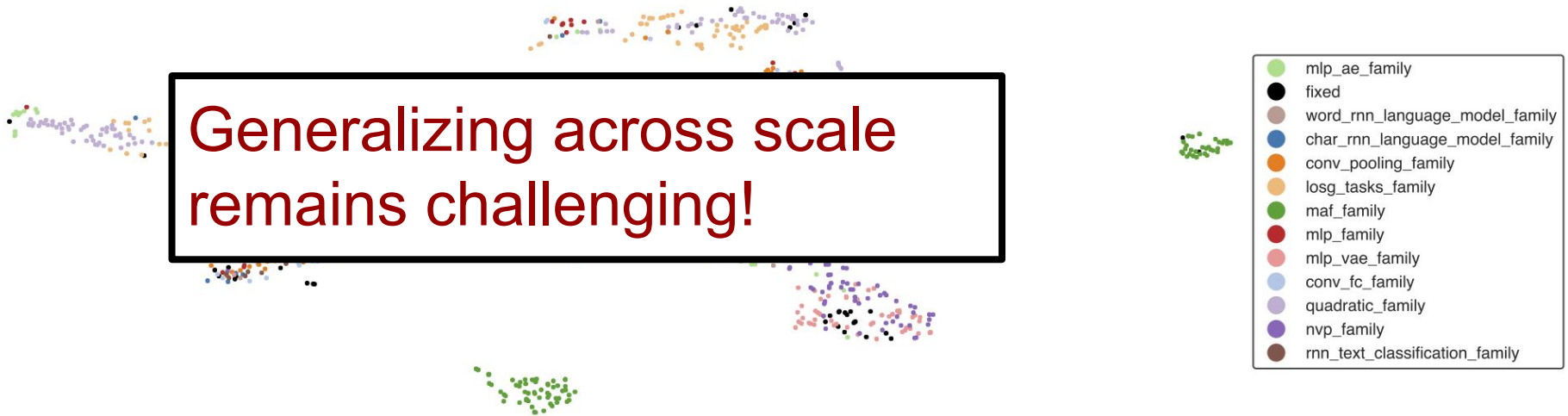
More tasks means better generalization



[Metz, et al., 2020]

Solution: Construct a large “dataset” of optimization tasks

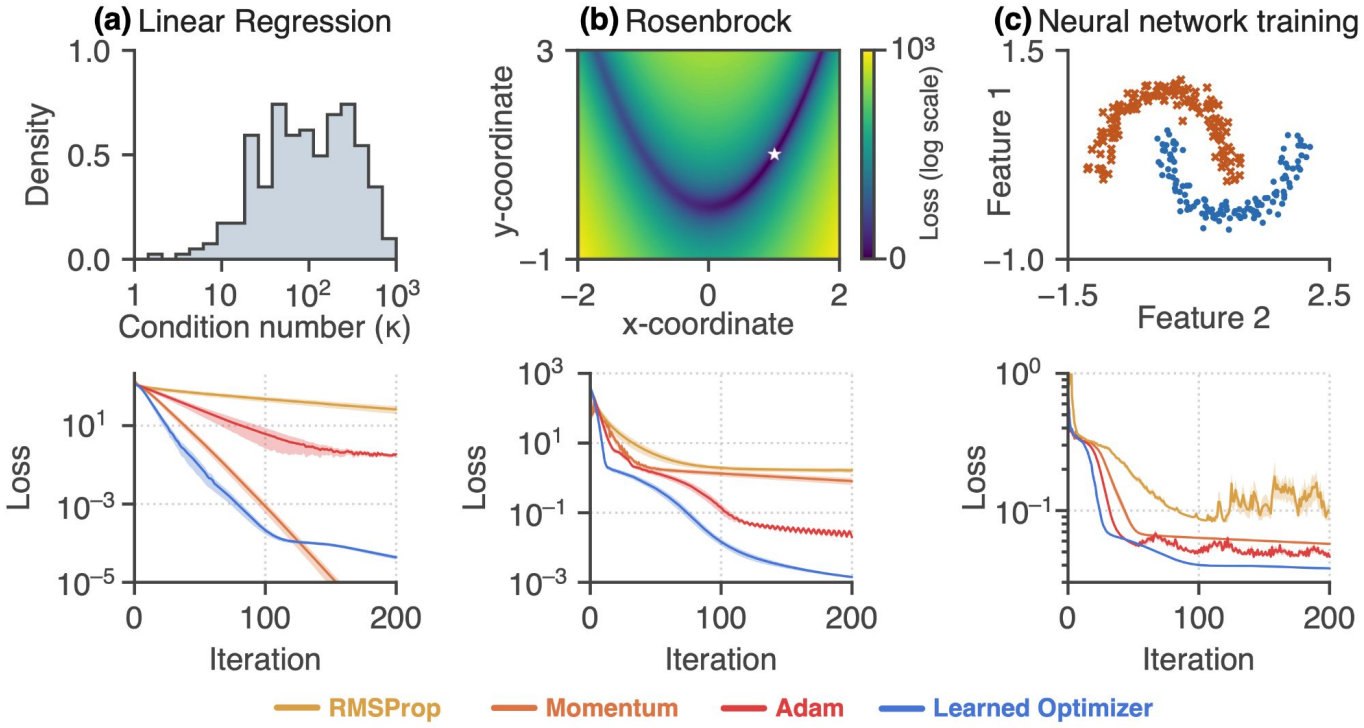
TSNE embedding of 1162 tasks for training learned optimizers



[Metz, et al., 2020]

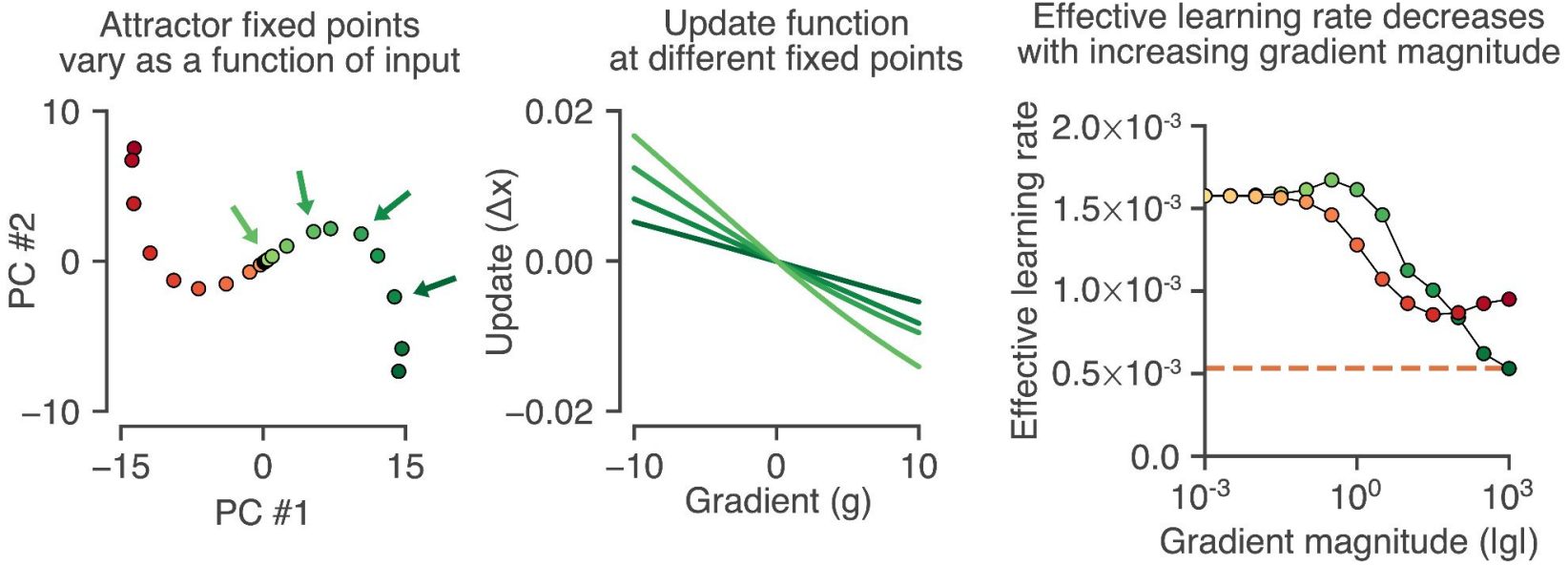
Problem: Interpretability -- understanding the mechanisms of action of learned optimizers

Careful analysis of GRU learned optimizer behavior on simple tasks



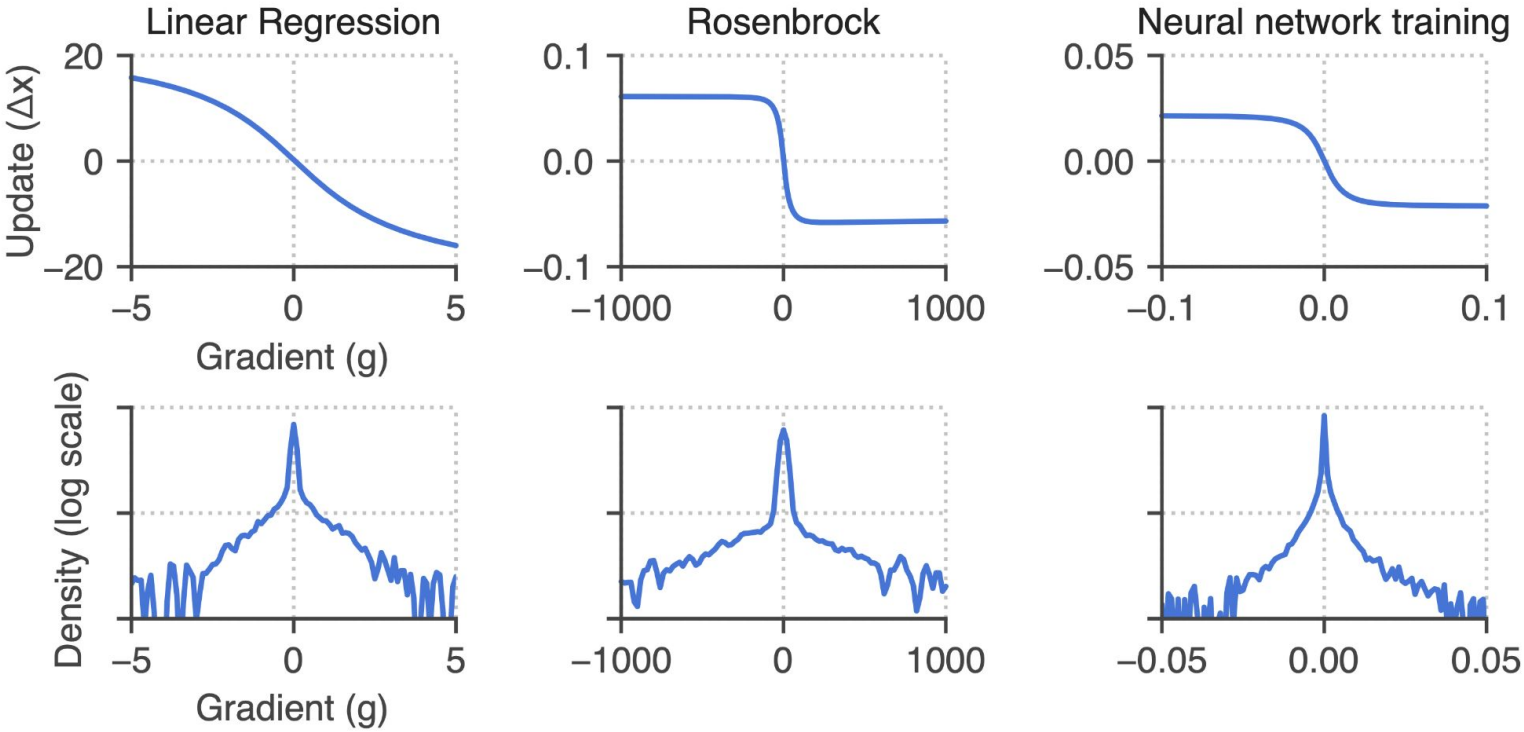
[Maheswaranathan, et al., 2021]

Example behavior: learning rate adaptation on Rosenbrock



[Maheswaranathan, et al., 2021]

Example behavior: gradient clipping



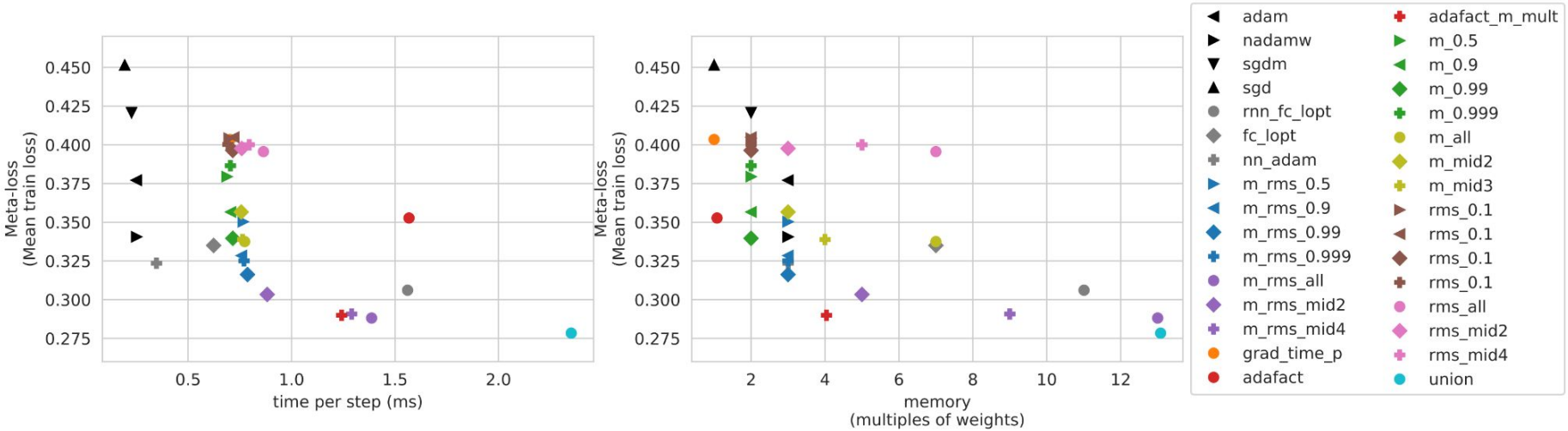
[Maheswaranathan, et al., 2021]

Problem: Designing learned optimizer architectures with the right inductive biases

Learned optimizer architectural themes

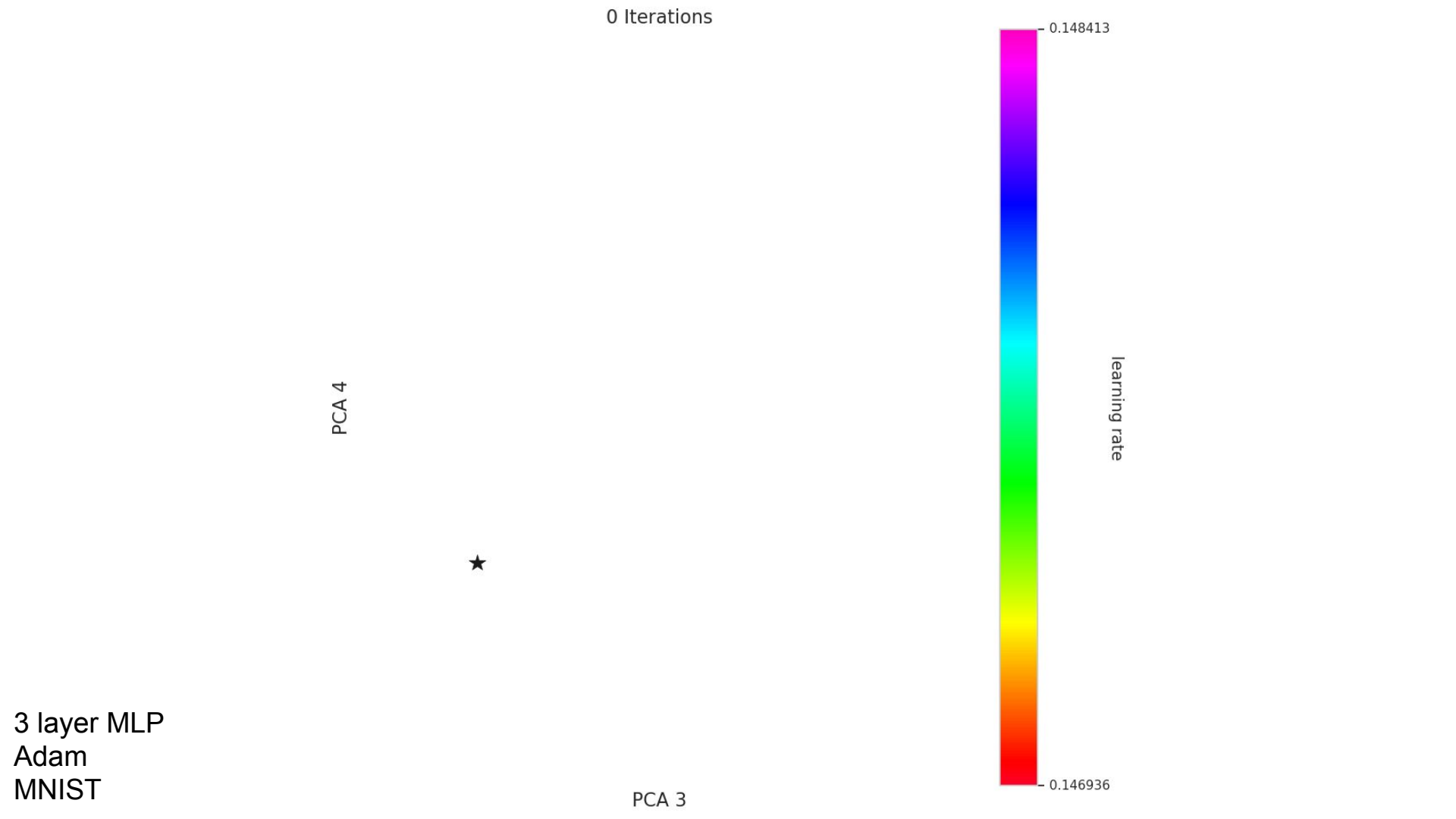
- Make input features invariant to gradient and parameter scale
  - e.g. RMSprop-style normalization by running average of gradient scale
- Make update steps proportional to parameter scale
  - scale updates by parameter norm
- Provide as much information as practical to the learned optimizer…
  - exponential weighted average of gradients on multiple timescales, validation loss, array shapes, training time, Adafactor features, …
- … while minimizing the compute overhead during training
  - use a hierarchical architecture, with very little optimizer compute per inner parameter

Pareto frontiers of performance, and compute and memory overhead

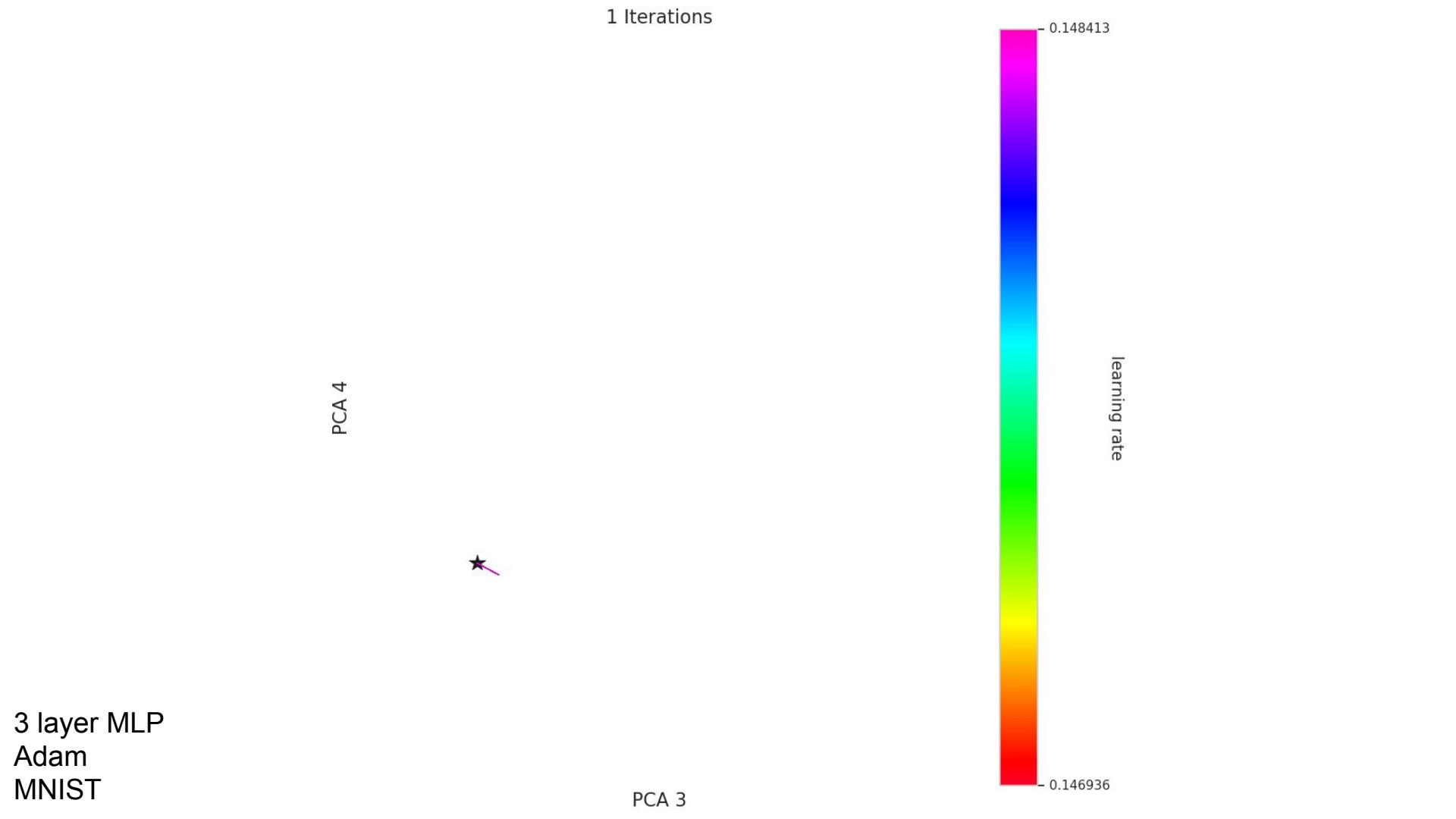


[Metz, et al, in preparation]

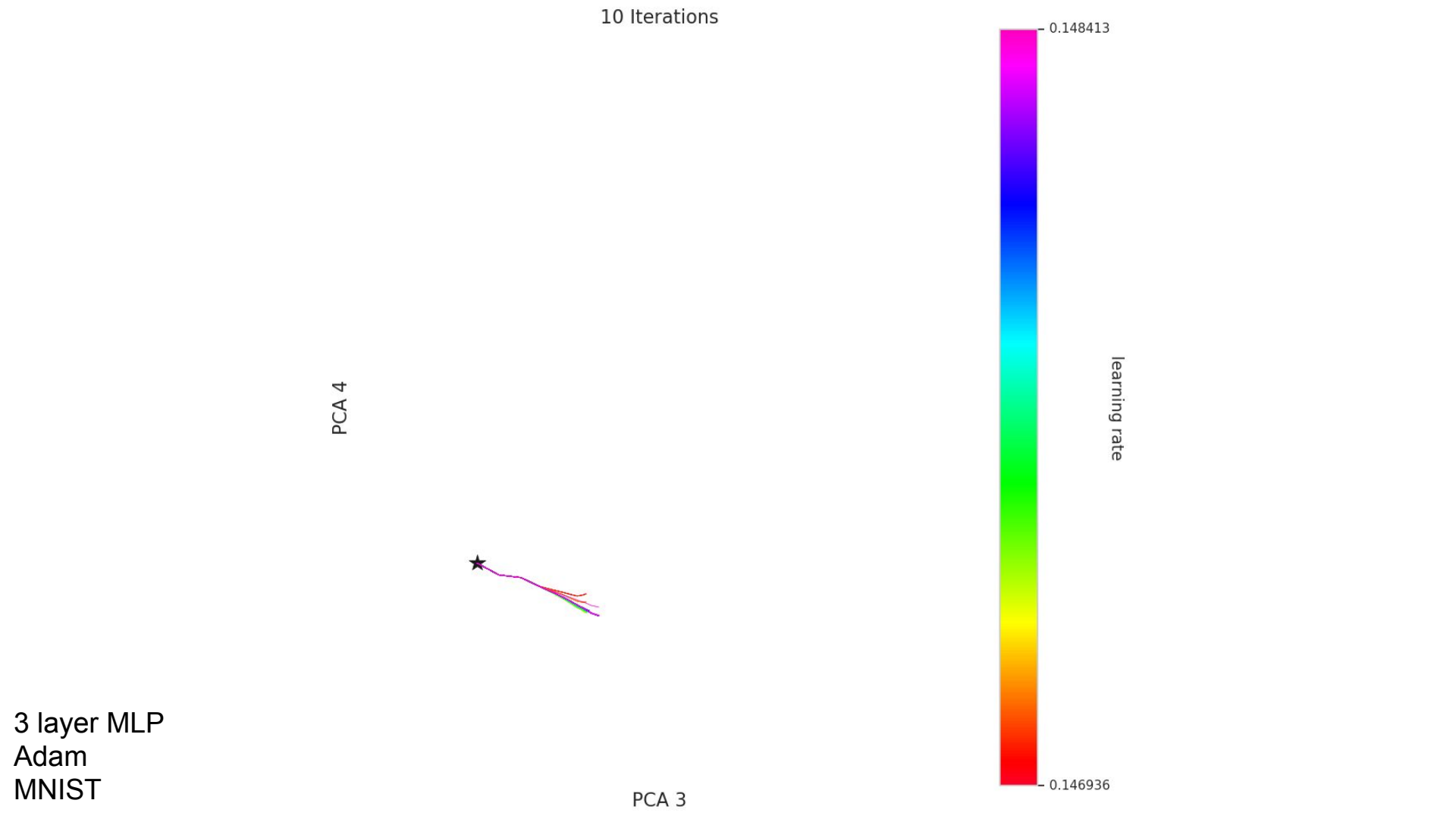
Problem: Chaotic inner-dynamics,
chaotic outer-loss,
and exploding outer-gradients



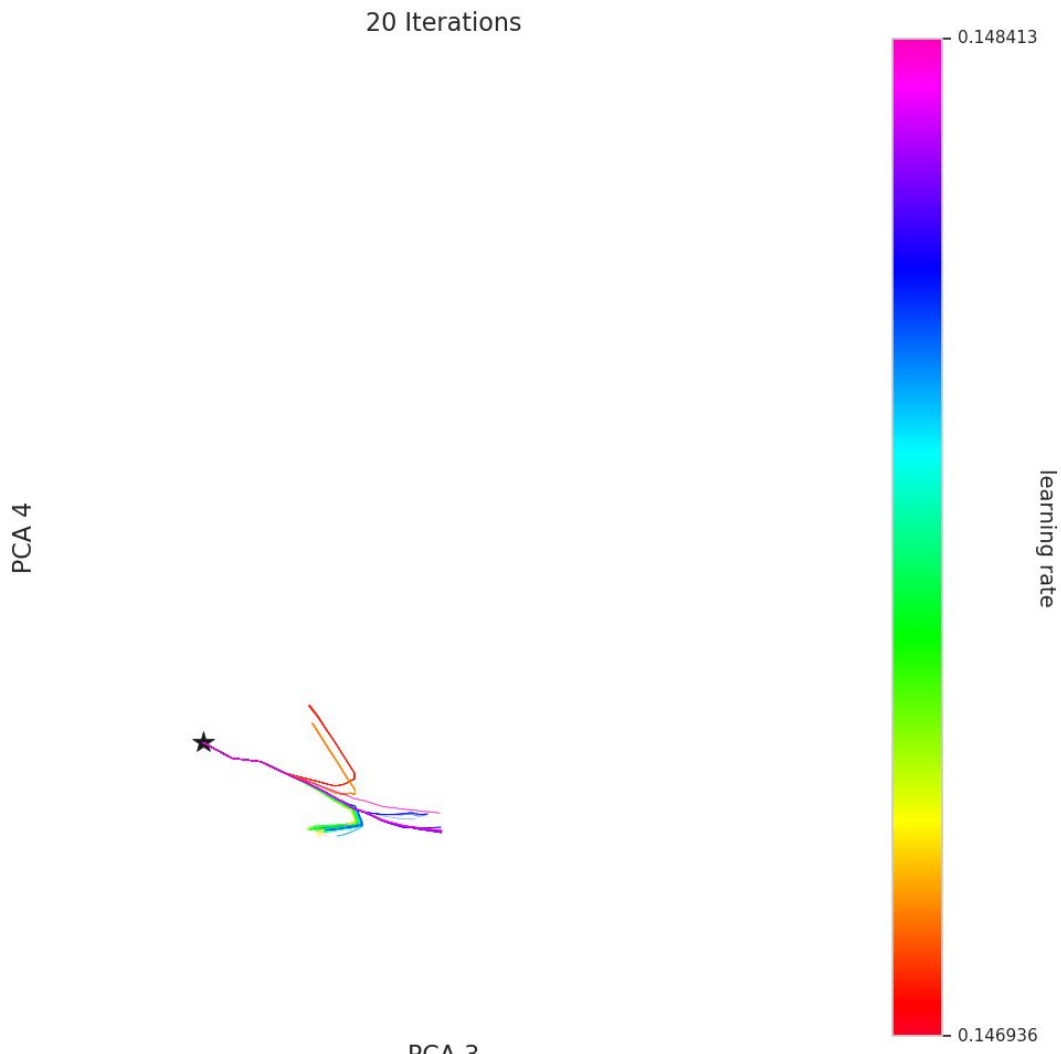
3 layer MLP
Adam
MNIST



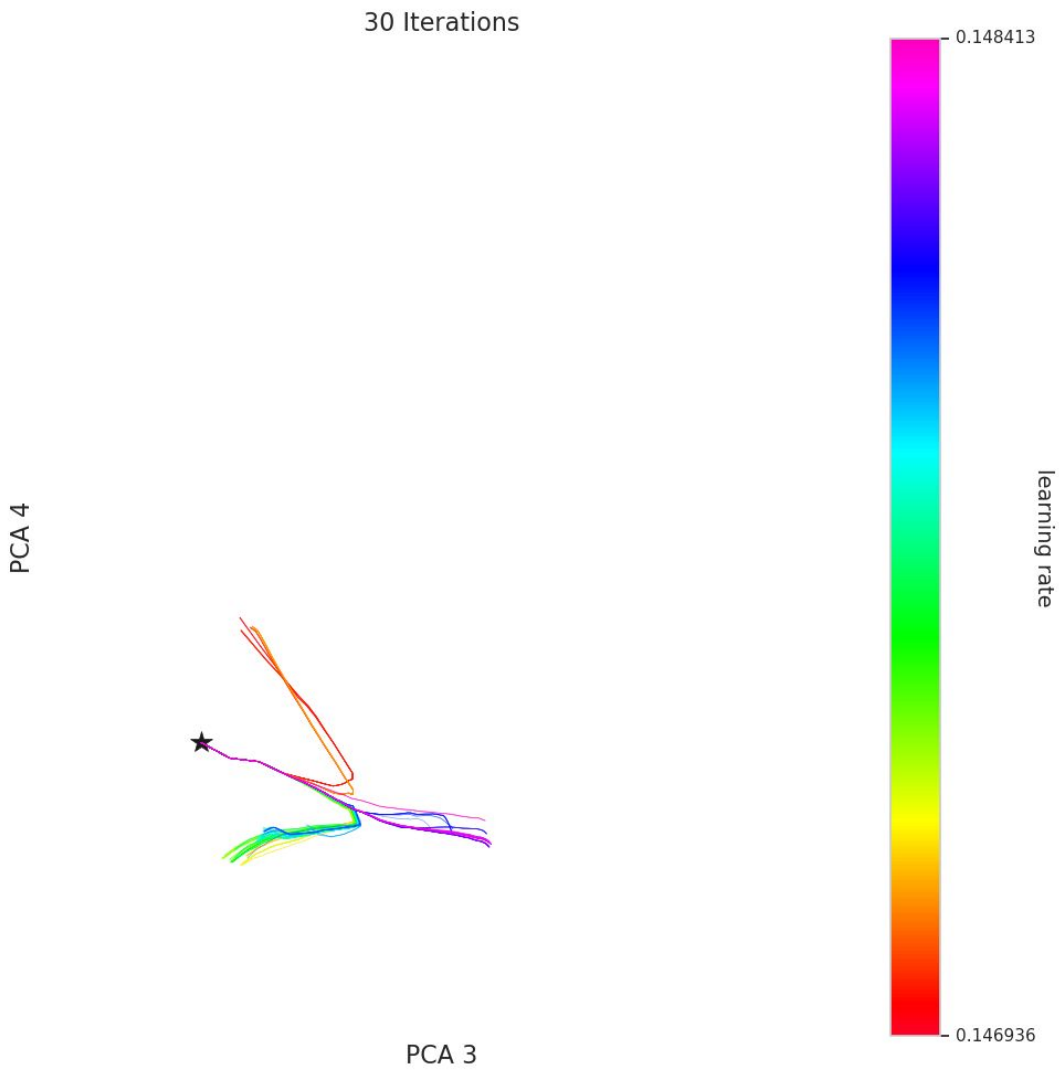
3 layer MLP
Adam
MNIST



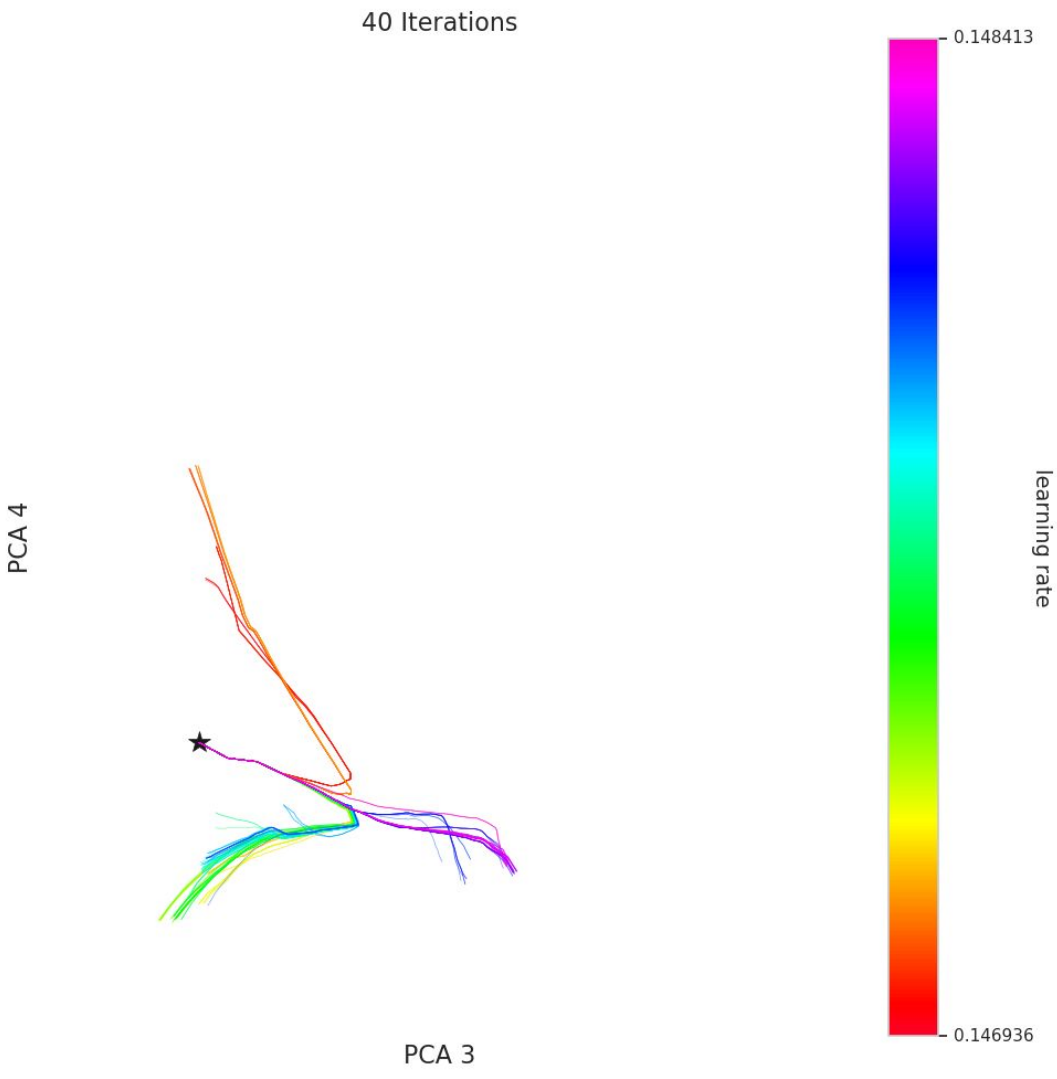
3 layer MLP
Adam
MNIST



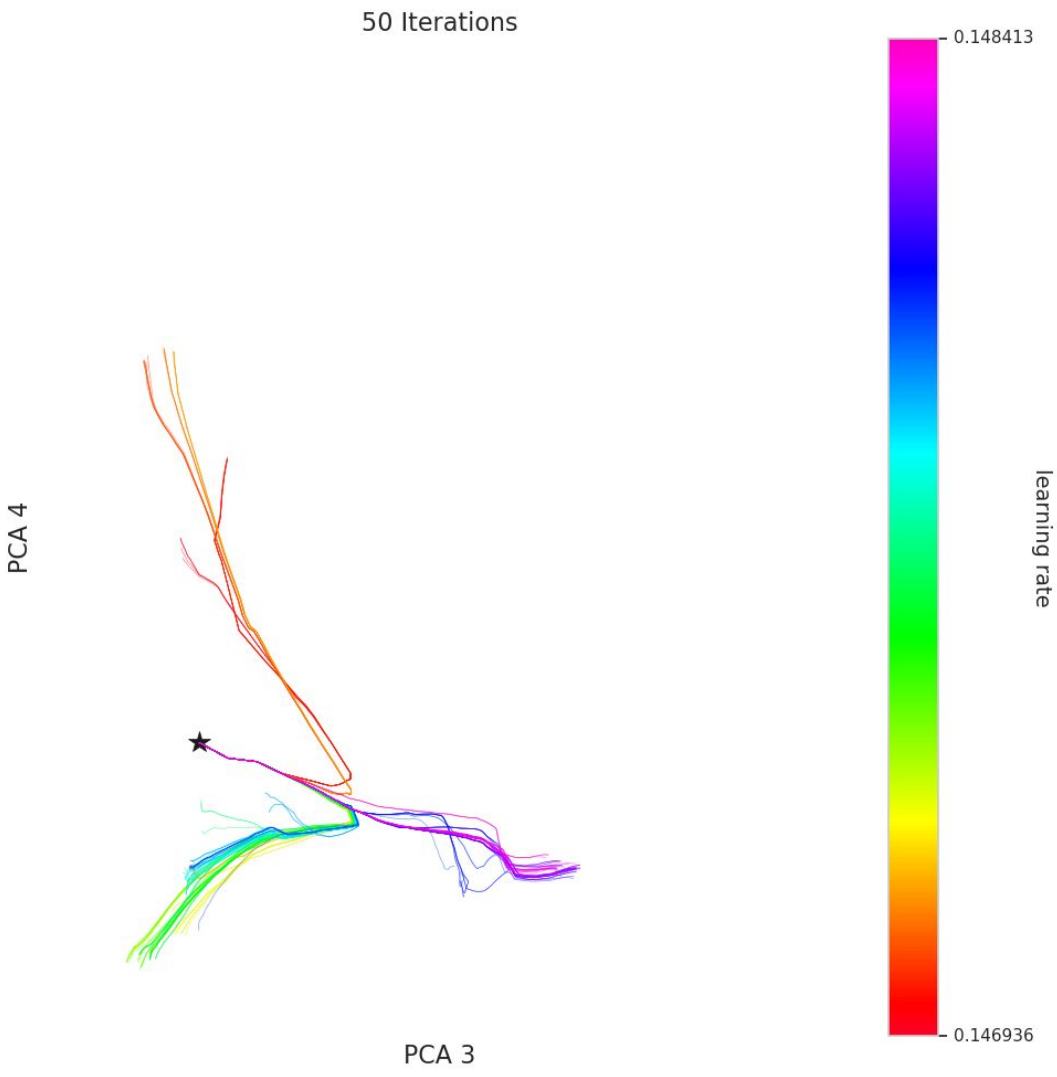
3 layer MLP
Adam
MNIST



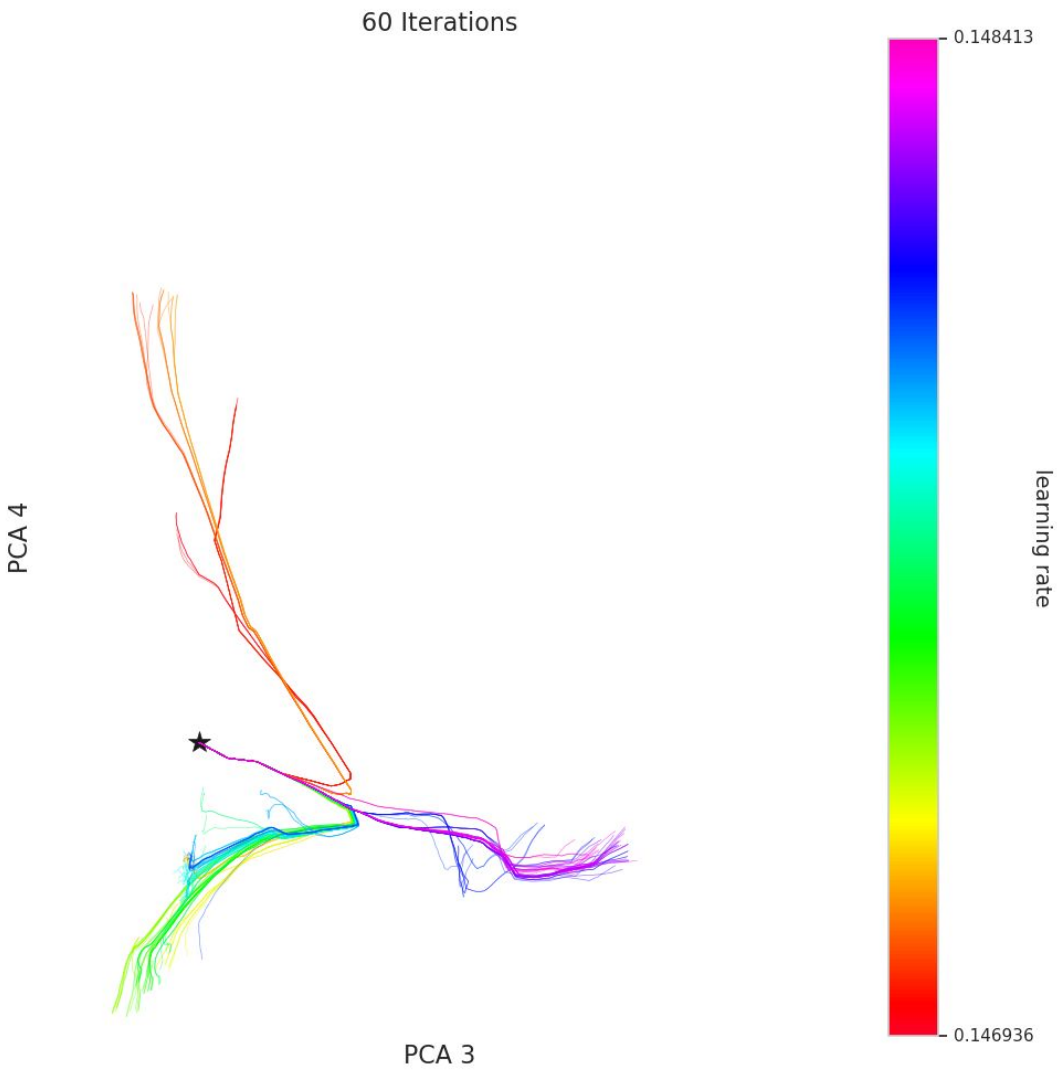
3 layer MLP
Adam
MNIST



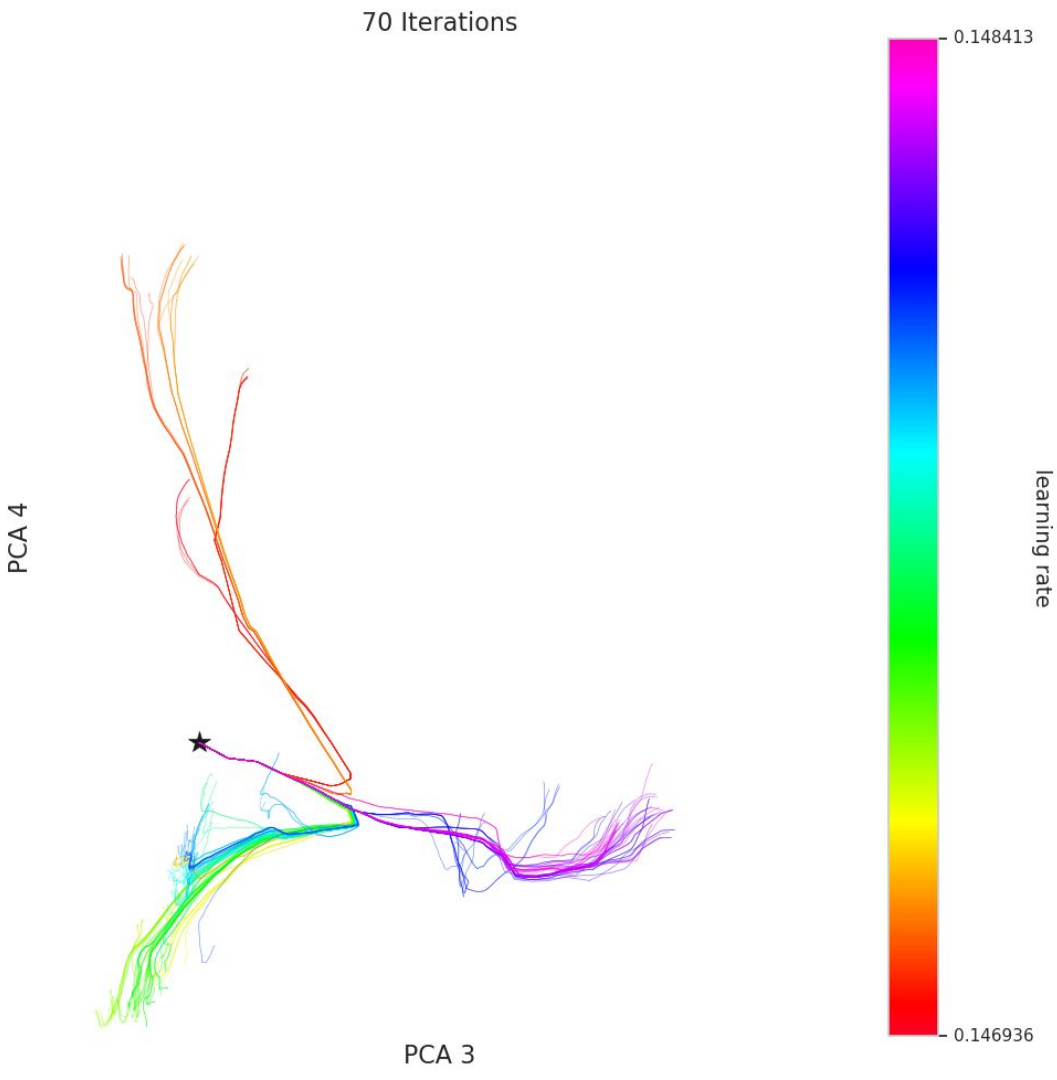
3 layer MLP
Adam
MNIST



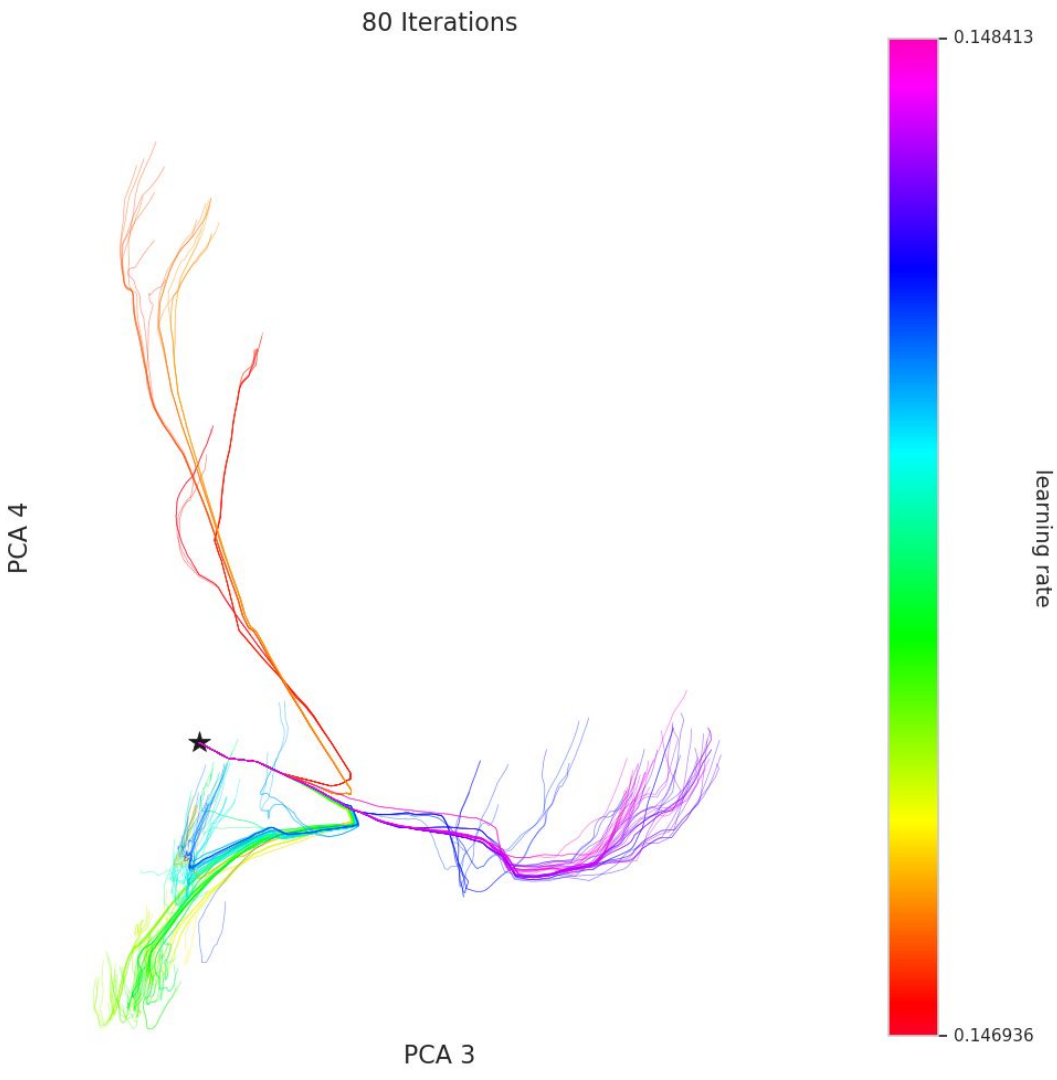
3 layer MLP
Adam
MNIST



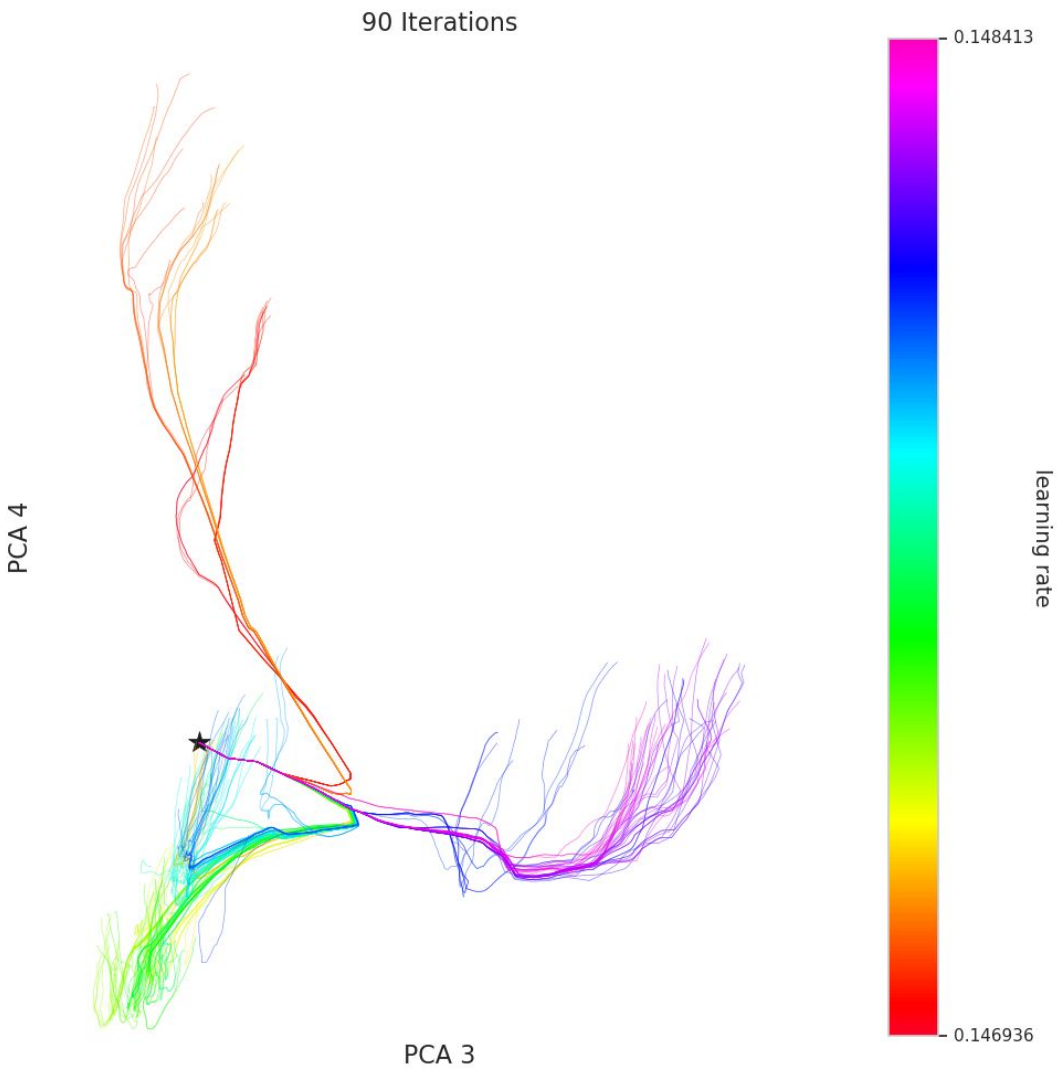
3 layer MLP
Adam
MNIST



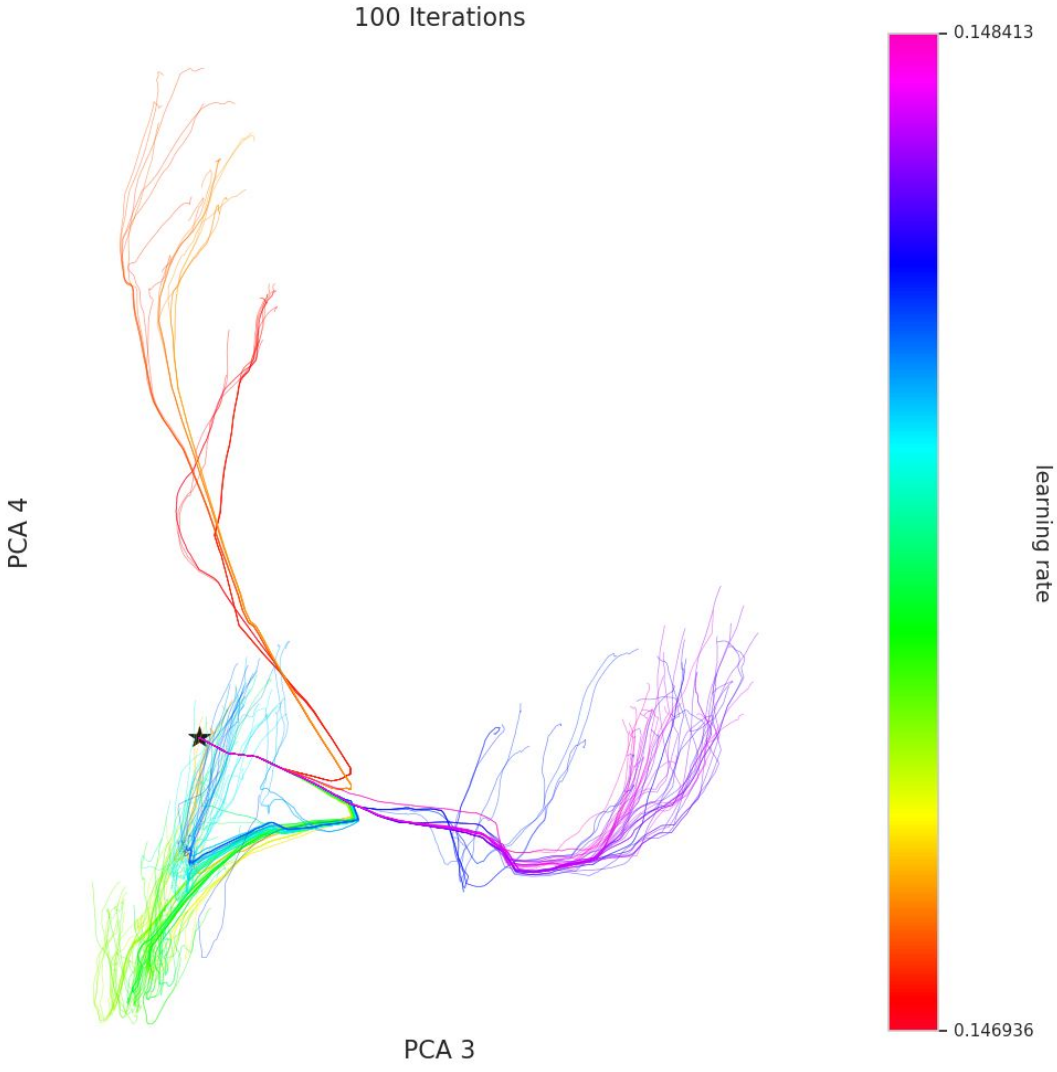
3 layer MLP
Adam
MNIST



3 layer MLP
Adam
MNIST



3 layer MLP
Adam
MNIST

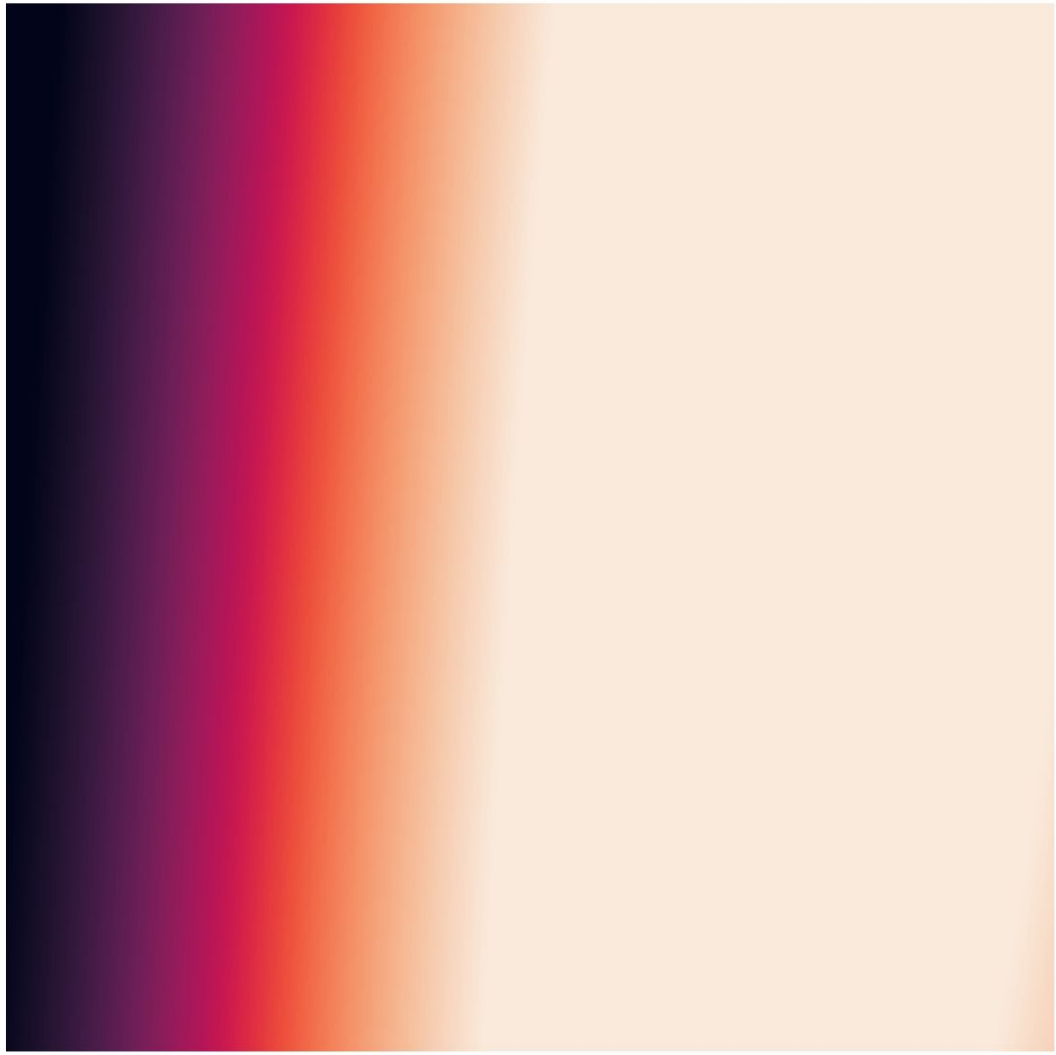


3 layer MLP
Adam
MNIST

The outer loss landscape is *chaotic* and difficult to meta-optimize

Outer loss surface for 2D slice through parameters of optimizer

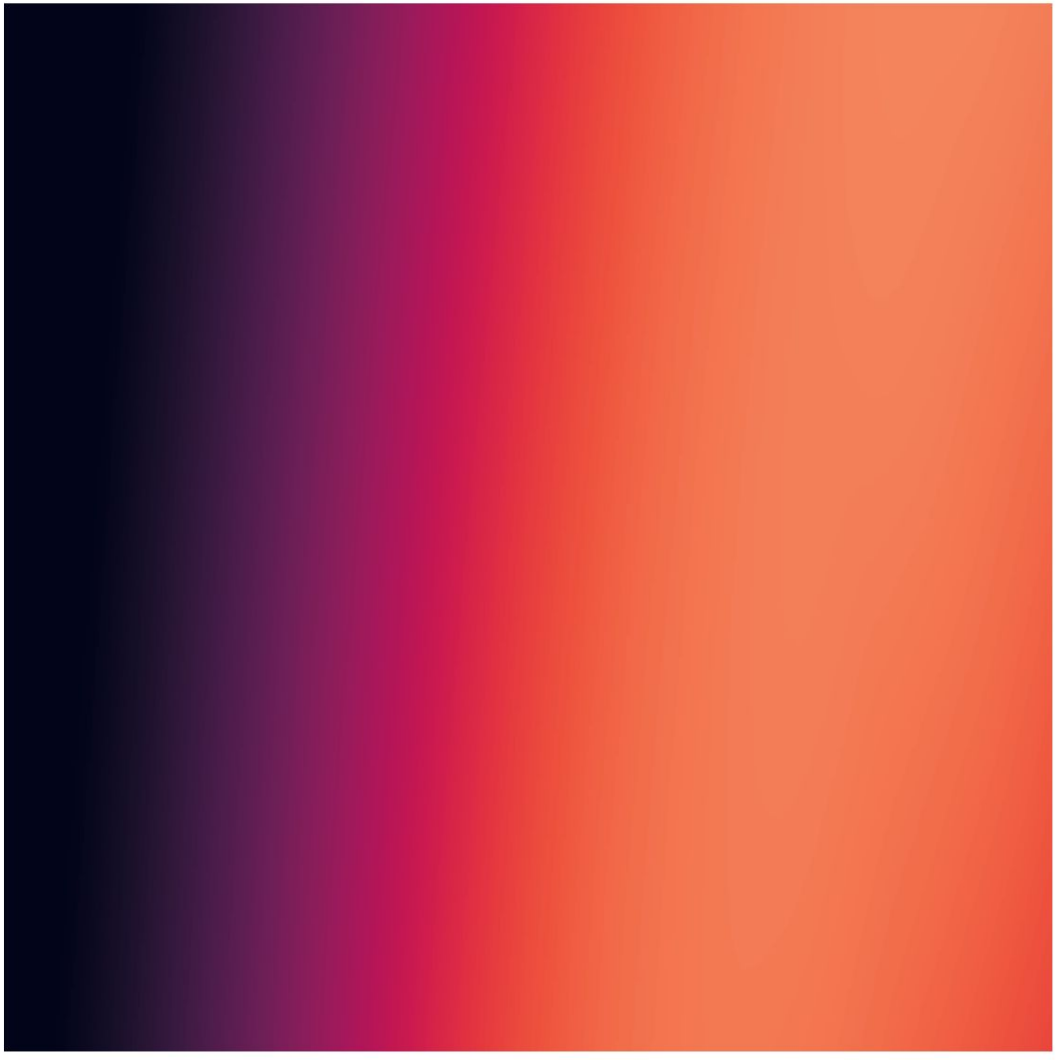
Number of unroll steps: 1



The outer loss landscape is *chaotic* and difficult to meta-optimize

Outer loss surface for 2D slice through parameters of optimizer

Number of unroll steps: 2



The outer loss landscape is *chaotic* and difficult to meta-optimize

Outer loss surface for 2D slice through parameters of optimizer

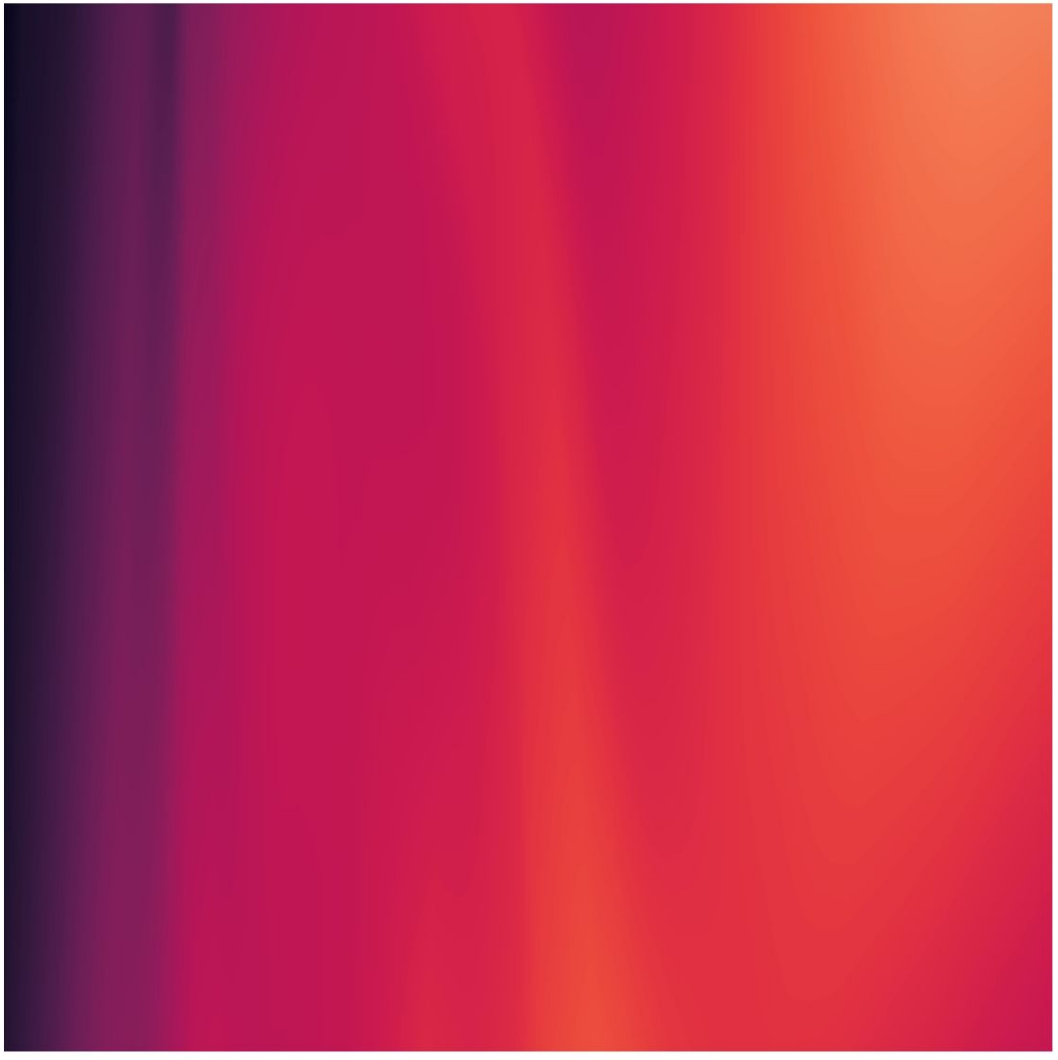
Number of unroll steps: 3



The outer loss landscape is *chaotic* and difficult to meta-optimize

Outer loss surface for 2D slice through parameters of optimizer

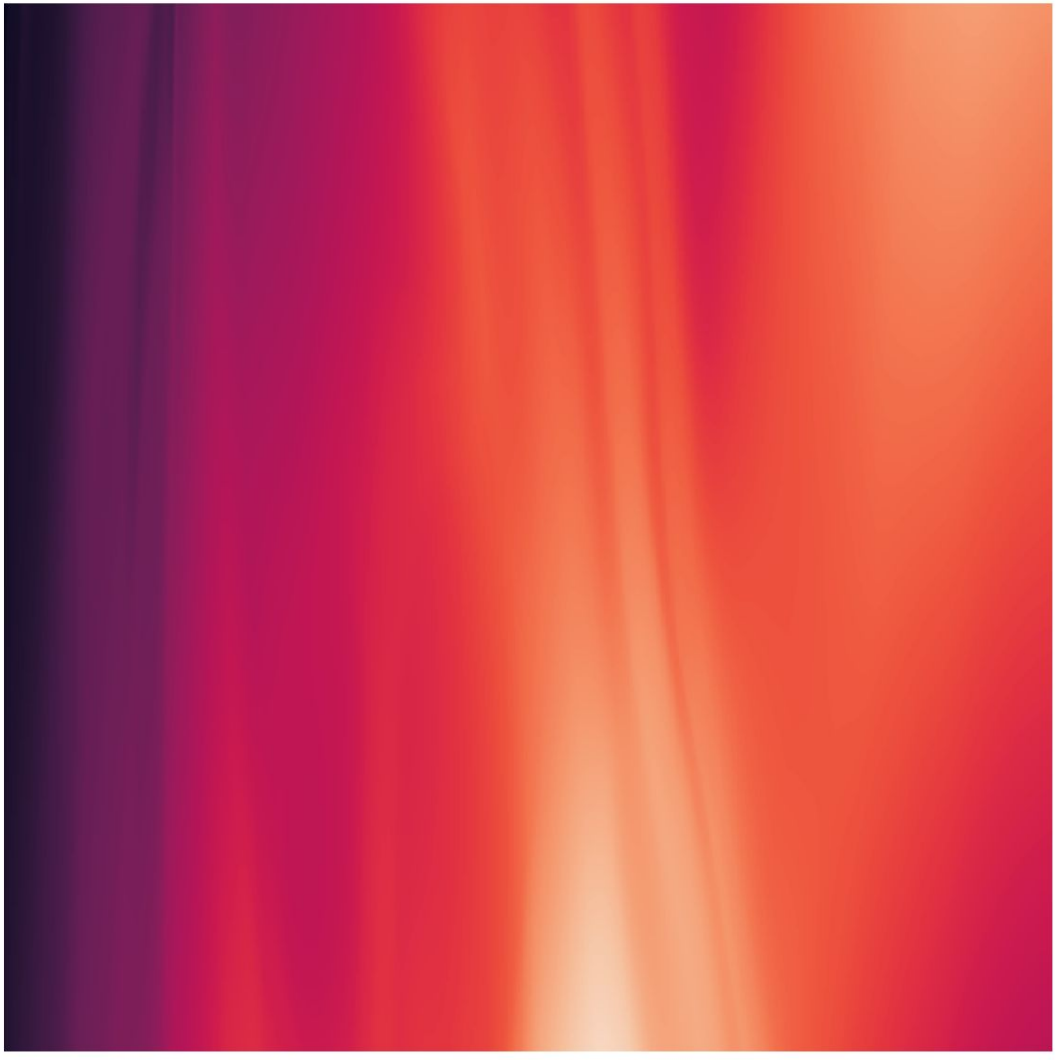
Number of unroll steps: 5



The outer loss landscape is *chaotic* and difficult to meta-optimize

Outer loss surface for 2D slice through parameters of optimizer

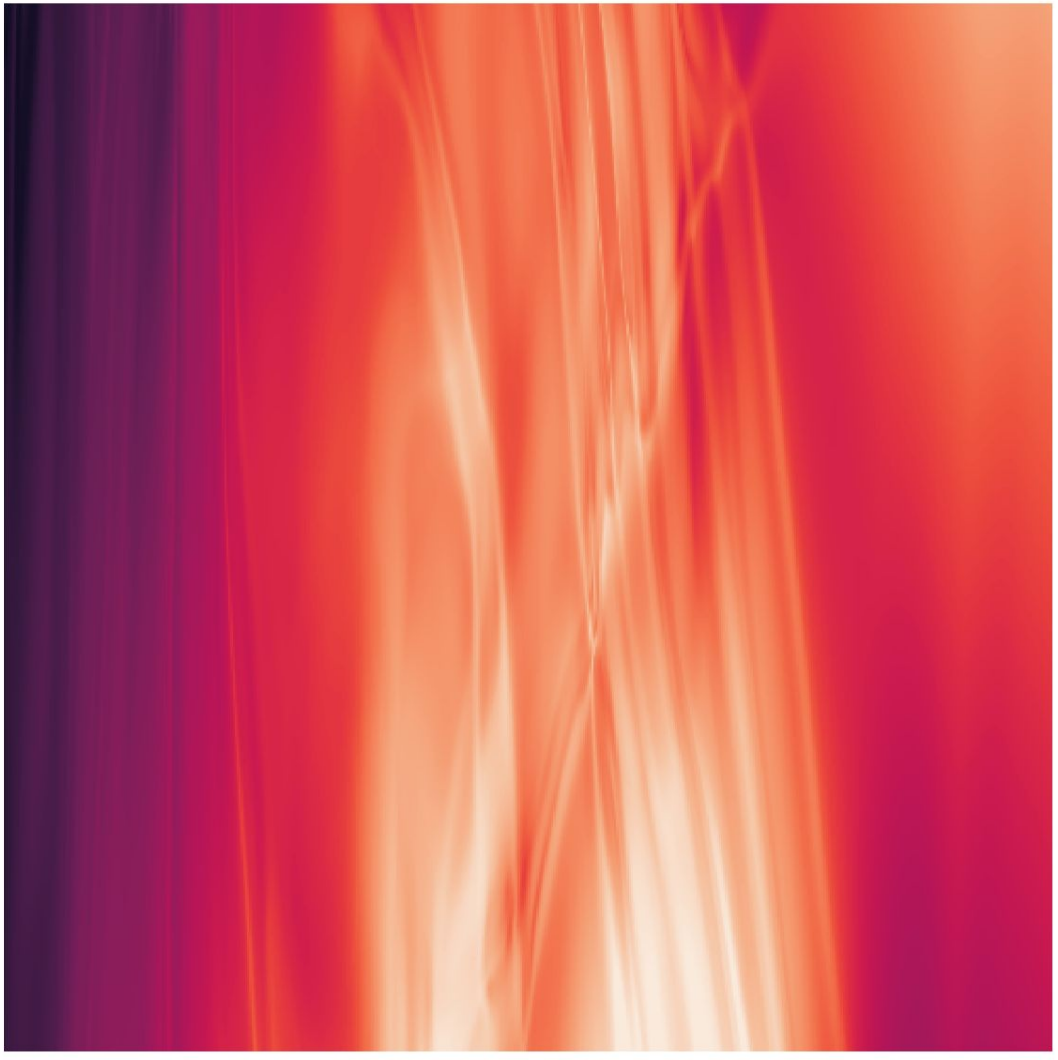
Number of unroll steps: 7



The outer loss landscape is *chaotic* and difficult to meta-optimize

Outer loss surface for 2D slice through parameters of optimizer

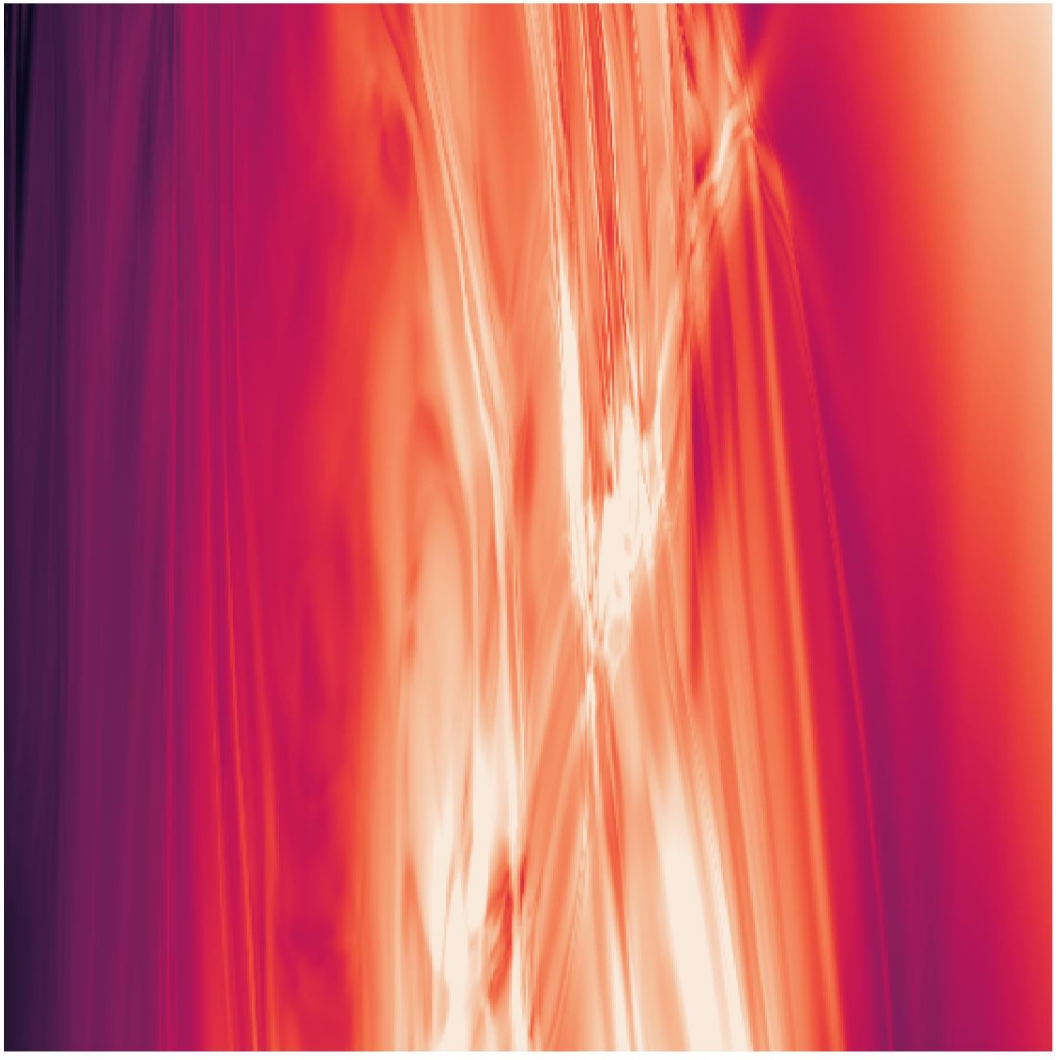
Number of unroll steps: 10



The outer loss landscape is *chaotic* and difficult to meta-optimize

Outer loss surface for 2D slice through parameters of optimizer

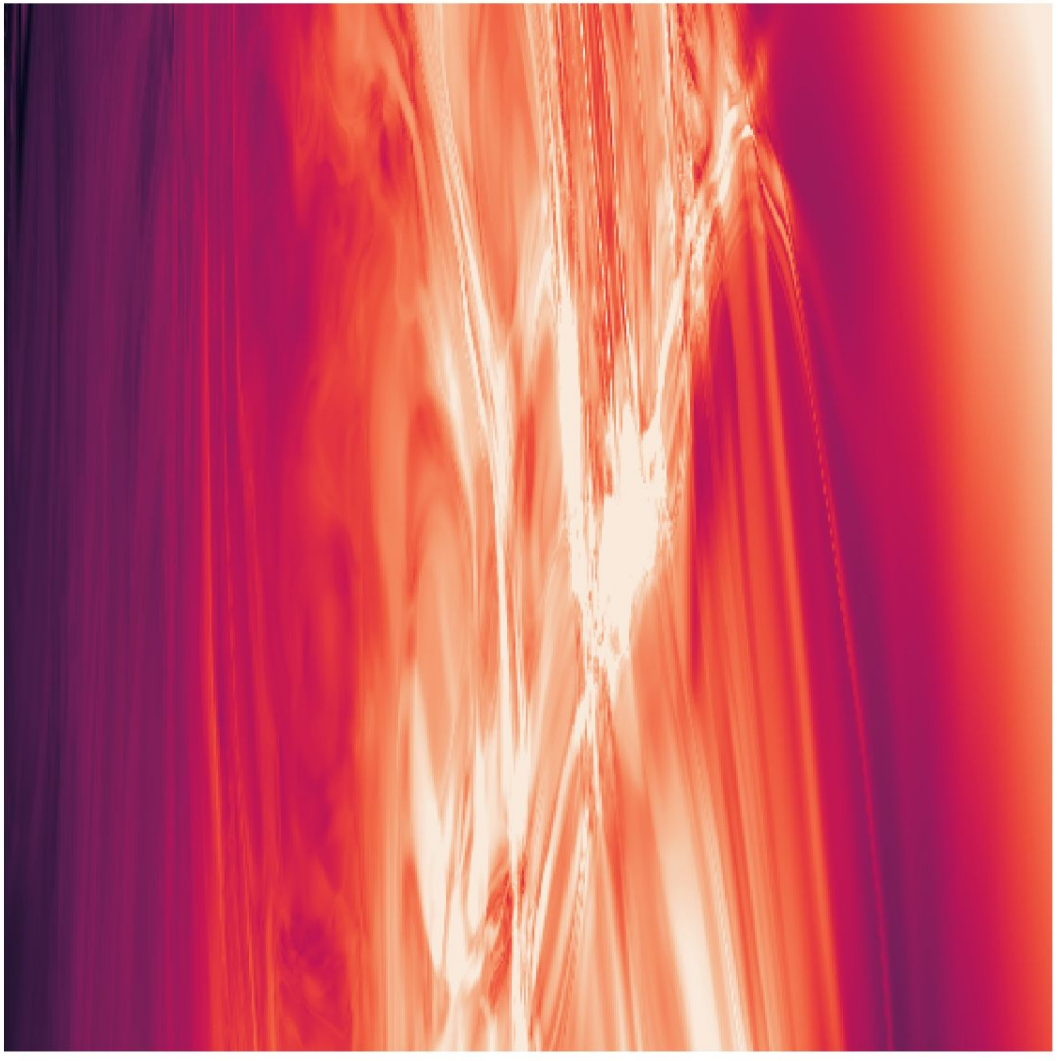
Number of unroll steps: 13



The outer loss landscape is *chaotic* and difficult to meta-optimize

Outer loss surface for 2D slice through parameters of optimizer

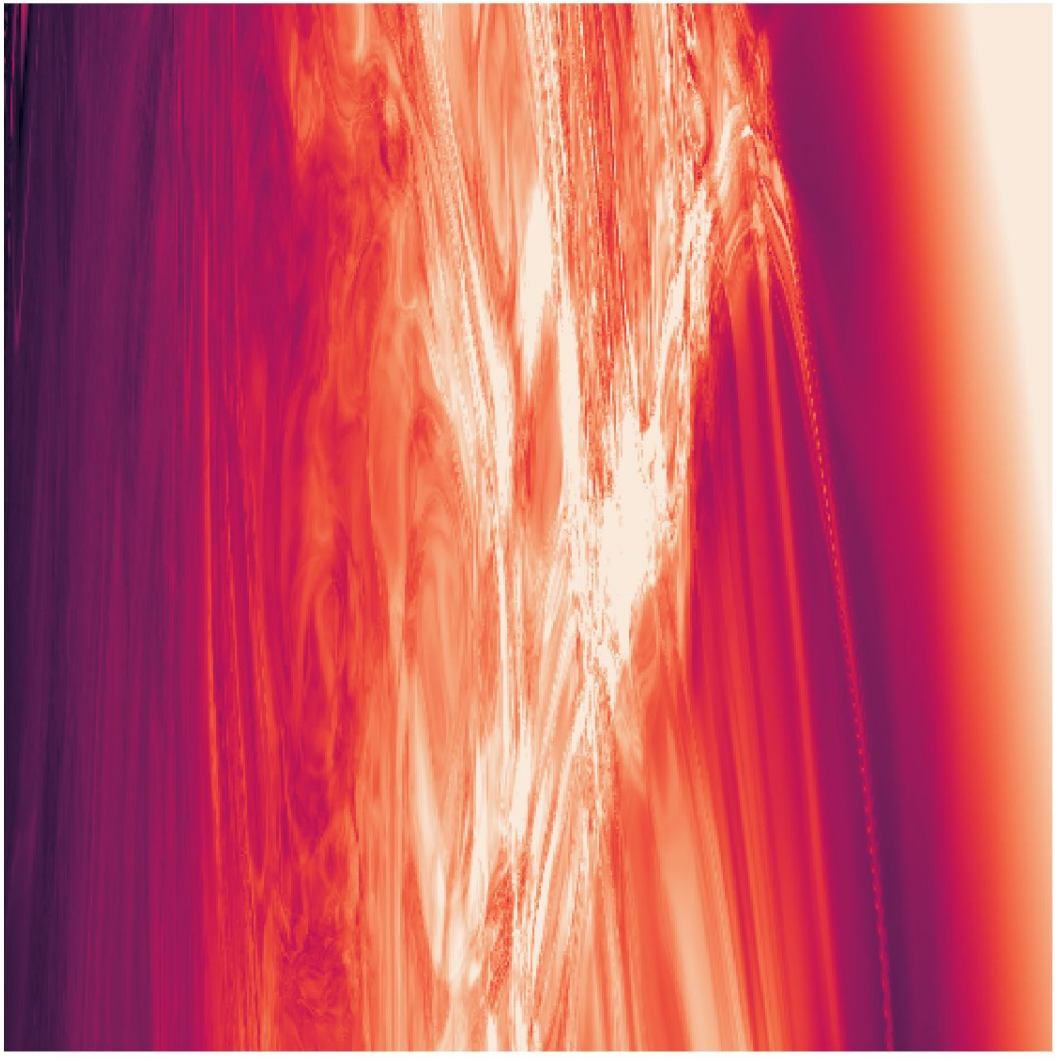
Number of unroll steps: 15



The outer loss landscape is *chaotic* and difficult to meta-optimize

Outer loss surface for 2D slice through parameters of optimizer

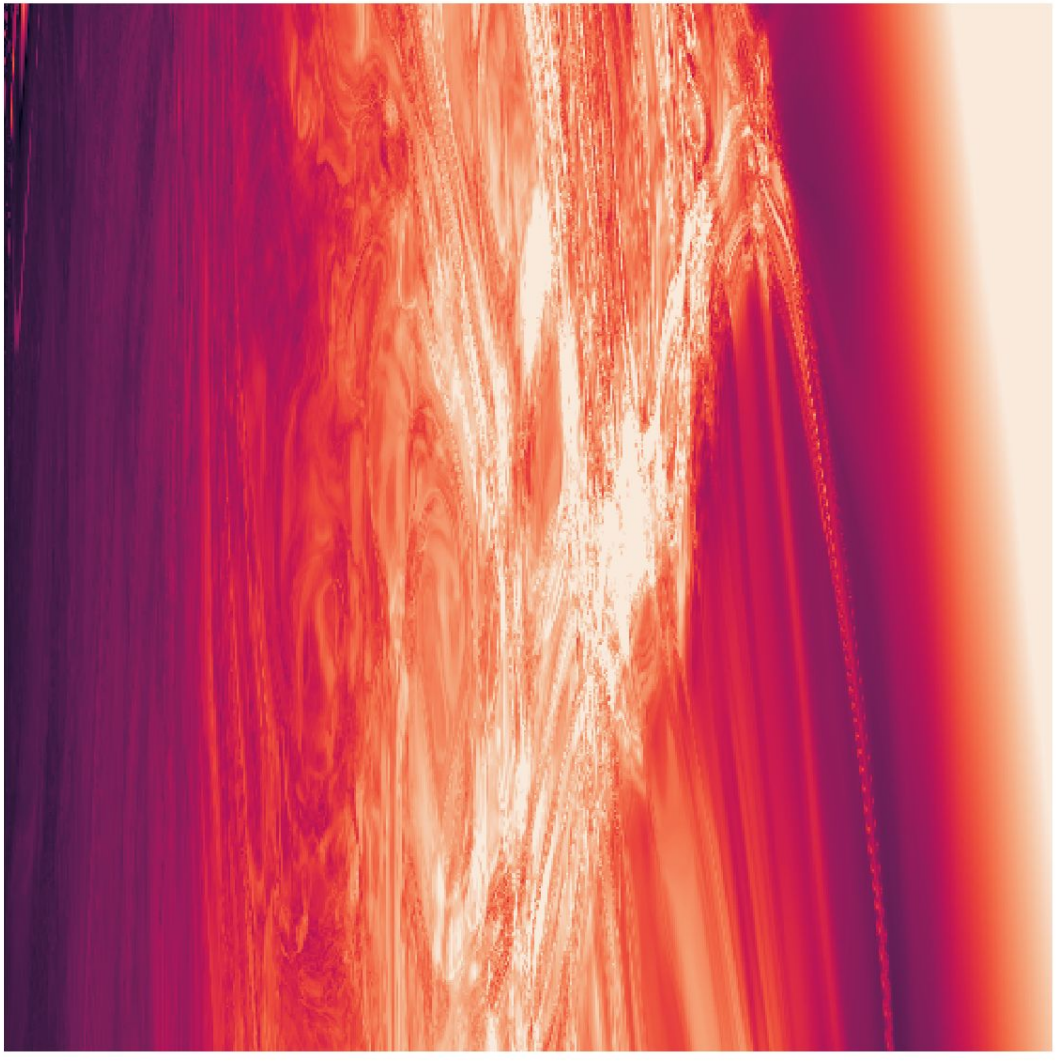
Number of unroll steps: 18



The outer loss landscape is *chaotic* and difficult to meta-optimize

Outer loss surface for 2D slice through parameters of optimizer

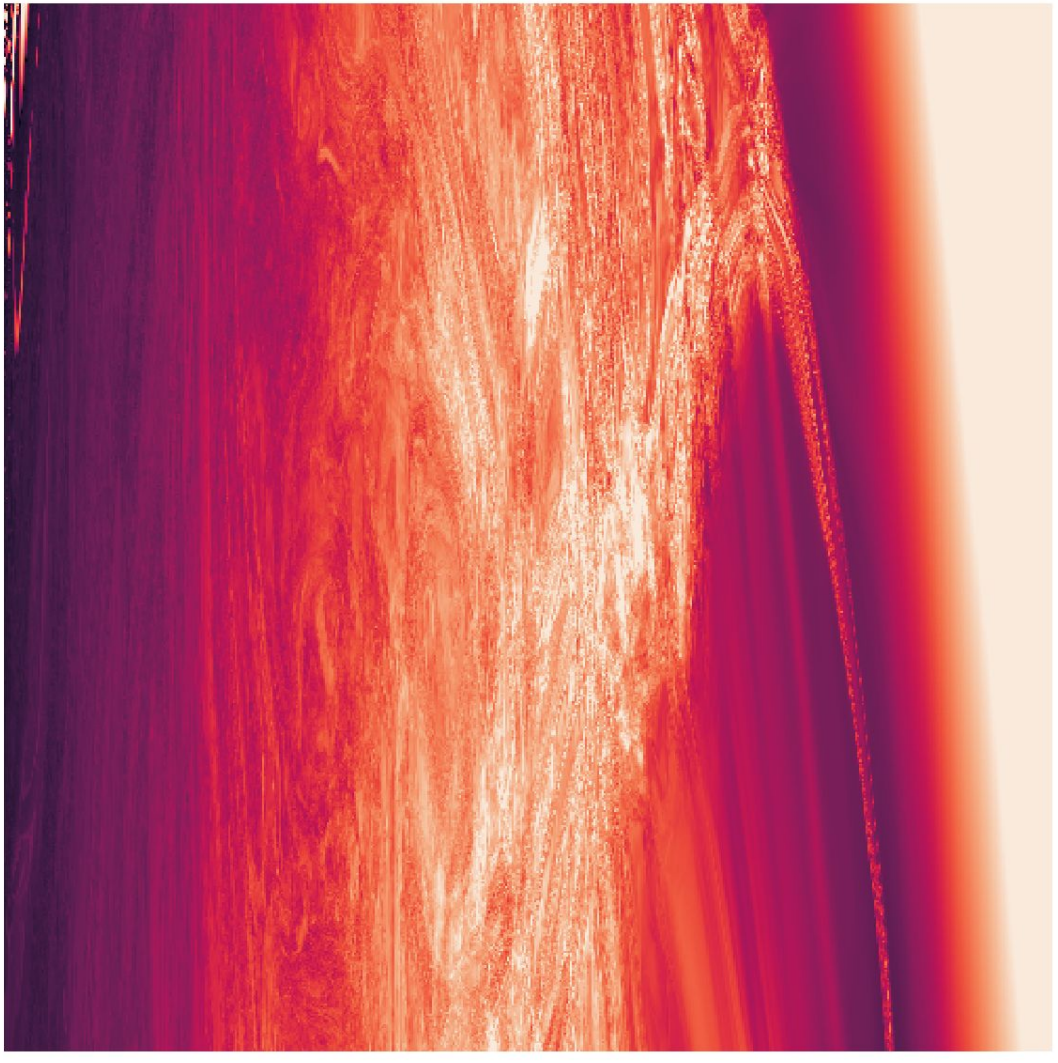
Number of unroll steps: 20



The outer loss landscape is *chaotic* and difficult to meta-optimize

Outer loss surface for 2D slice through parameters of optimizer

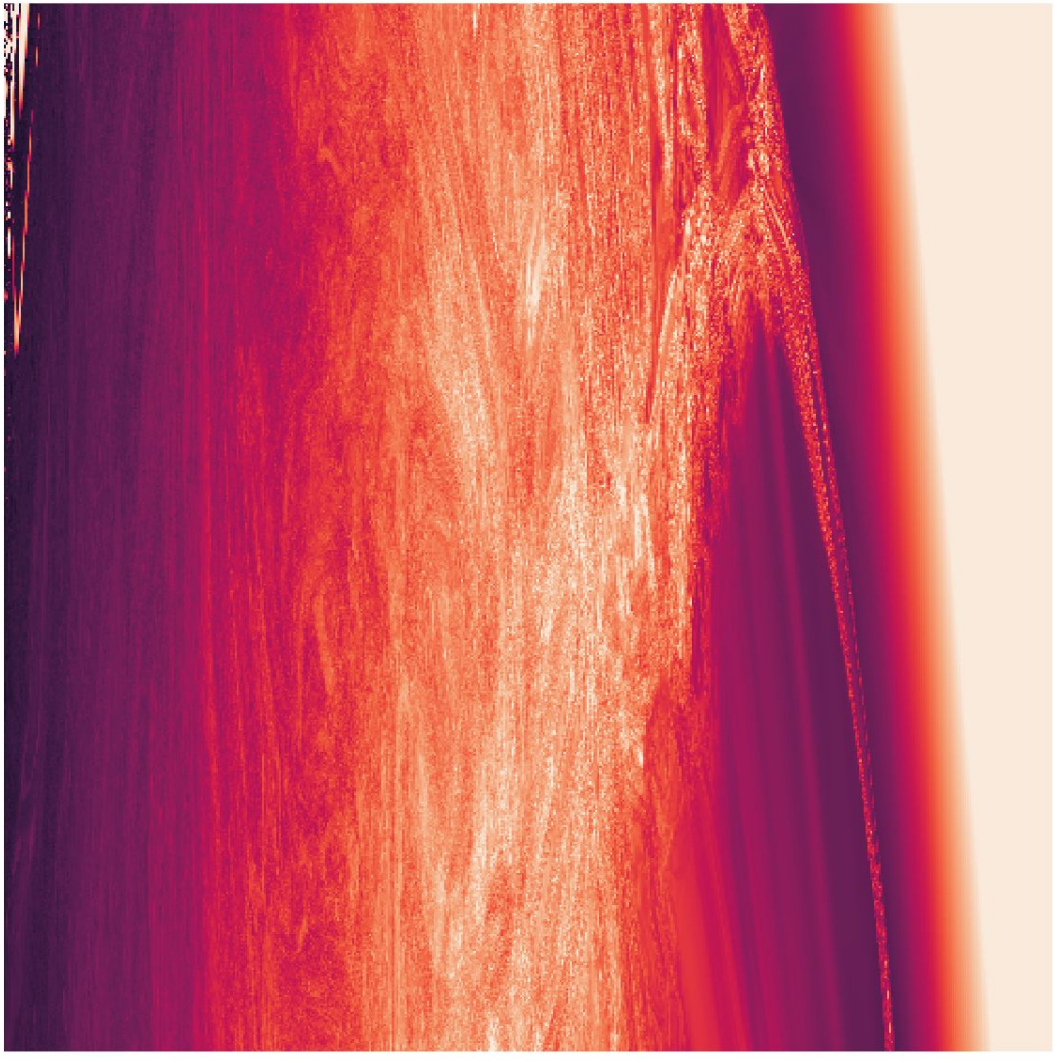
Number of unroll steps: 25



The outer loss landscape is *chaotic* and difficult to meta-optimize

Outer loss surface for 2D slice through parameters of optimizer

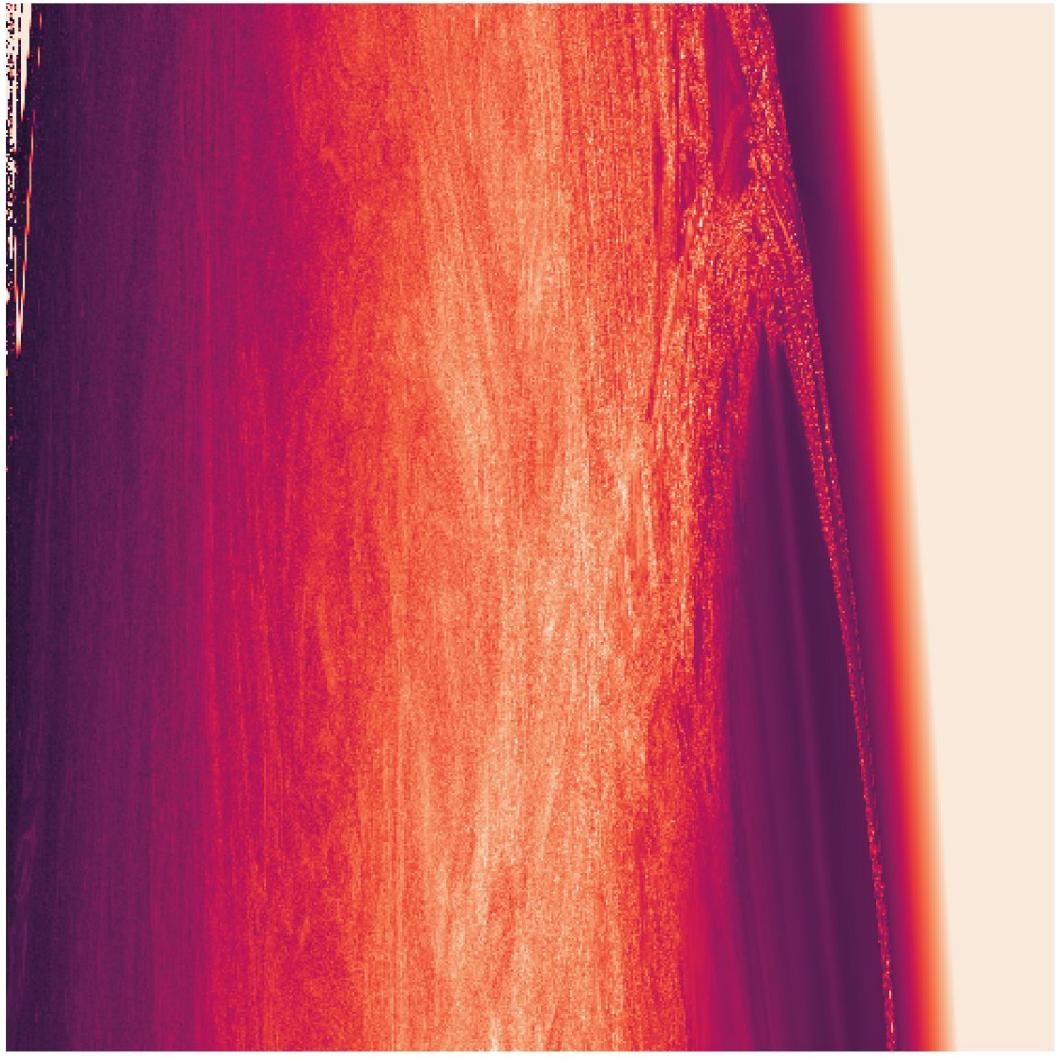
Number of unroll steps: 30



The outer loss landscape is *chaotic* and difficult to meta-optimize

Outer loss surface for 2D slice through parameters of optimizer

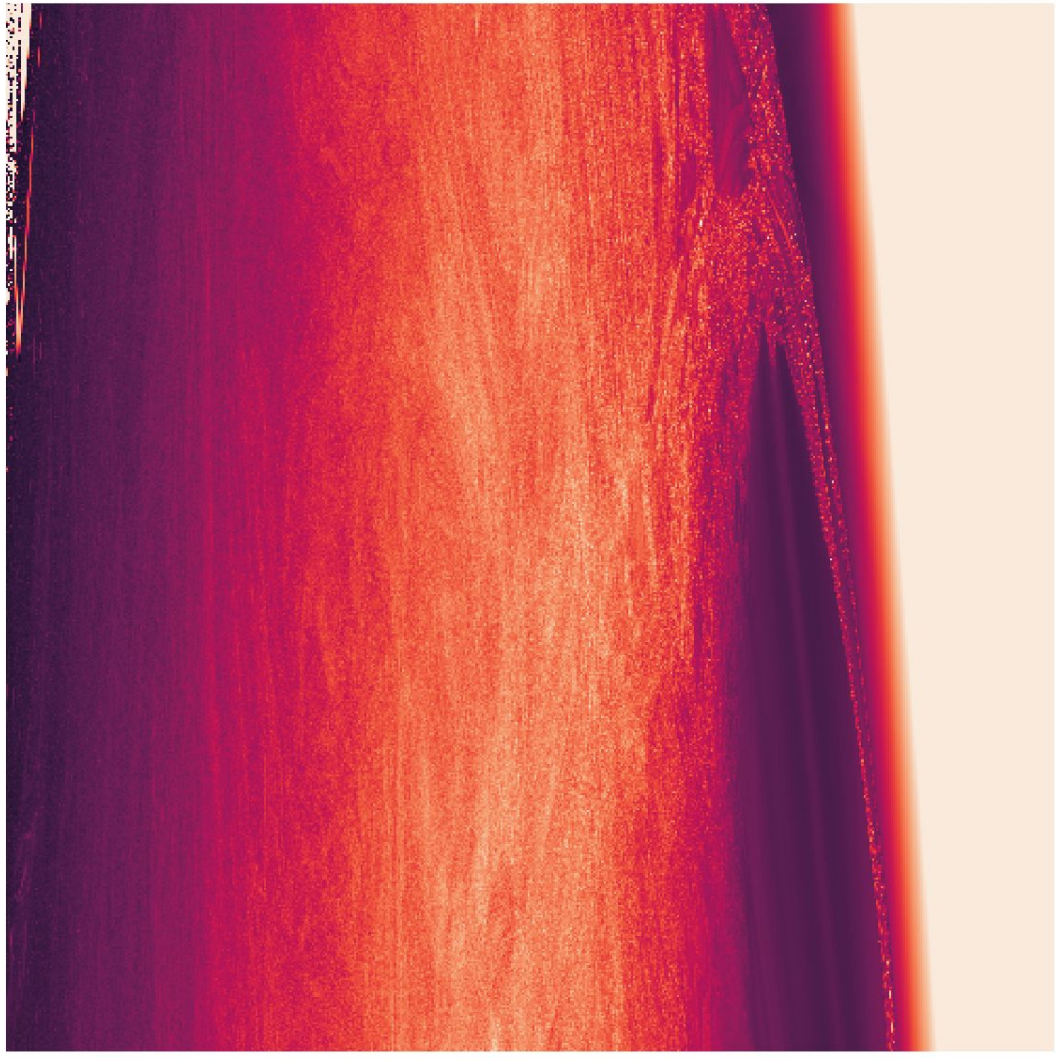
Number of unroll steps: 40



The outer loss landscape is *chaotic* and difficult to meta-optimize

Outer loss surface for 2D slice through parameters of optimizer

Number of unroll steps: 50



Problem: Chaotic inner-dynamics,
chaotic outer-loss,
and exploding outer-gradients

(Current best) solution: Smooth the outer-loss landscape

Variational Optimization

- Define a distribution over parameters
- Convolve outer-loss surface with this distribution
- High frequency structure suppressed
- Optimize modified outer-loss

$$\mathcal{L}(\theta) = \mathbb{E}_{\tilde{\theta} \sim \mathcal{N}(\theta, \sigma^2 I)} \left[L\left(\tilde{\theta}\right) \right]$$

Sample from distribution

Parameters of the distribution

Original loss

$$\mathcal{L}(\theta) = \mathbb{E}_{\tilde{\theta} \sim \mathcal{N}(\theta, \sigma^2 I)} \left[L\left(\tilde{\theta}\right) \right]$$

Two unbiased gradient estimators

$$\nabla_\theta \mathcal{L}(\theta) = ?$$

Evolution Strategies (ES)
Score Function

$$g_{\text{es}} = \frac{1}{S} \sum_s L\left(\tilde{\theta}_s\right) \nabla_\theta \left[\log\left(\mathcal{N}\left(\tilde{\theta}_s; \theta, \sigma^2 I\right) \right) \right] = \frac{1}{S\sigma^2} \sum_s L\left(\tilde{\theta}_s\right) \left(\tilde{\theta}_s - \theta\right)$$

$$\tilde{\theta}_s \sim \mathcal{N}\left(\theta, \sigma^2 I\right)$$

$$\mathcal{L}(\theta) = \mathbb{E}_{\tilde{\theta} \sim \mathcal{N}(\theta, \sigma^2 I)} \left[L\left(\tilde{\theta}\right) \right]$$

Two unbiased gradient estimators

$$\nabla_\theta \mathcal{L}(\theta) = ?$$

Evolution Strategies (ES)
Score Function

$$g_{\text{es}} = \frac{1}{S\sigma^2} \sum_s L(\theta + \epsilon_s)\, \epsilon_s$$

$$\epsilon_s \sim \mathcal{N}\left(0, \sigma^2 I\right)$$

$$\mathcal{L}(\theta) = \mathbb{E}_{\tilde{\theta} \sim \mathcal{N}(\theta, \sigma^2 I)} \left[L\left(\tilde{\theta}\right) \right]$$

Two unbiased gradient estimators

$$\nabla_\theta \mathcal{L}(\theta) = ?$$

Evolution Strategies (ES)
Score Function

$$g_{\text{es}} = \frac{1}{2S\sigma^2} \sum_s \left(L(\theta + \epsilon_s) - L(\theta - \epsilon_s) \right) \epsilon_s$$

Antithetic samples reduce variance

$$\epsilon_s \sim \mathcal{N}\left(0, \sigma^2 I\right)$$

$$\mathcal{L}(\theta) = \mathbb{E}_{\tilde{\theta} \sim \mathcal{N}(\theta, \sigma^2 I)} \left[L\left(\tilde{\theta}\right) \right]$$

Two unbiased gradient estimators

$$\nabla_\theta \mathcal{L}(\theta) = ?$$

Evolution Strategies (ES)
Score Function

$$g_{\text{es}} = \frac{1}{2S\sigma^2} \sum_s \left(L(\theta + \epsilon_s) - L(\theta - \epsilon_s) \right) \epsilon_s$$

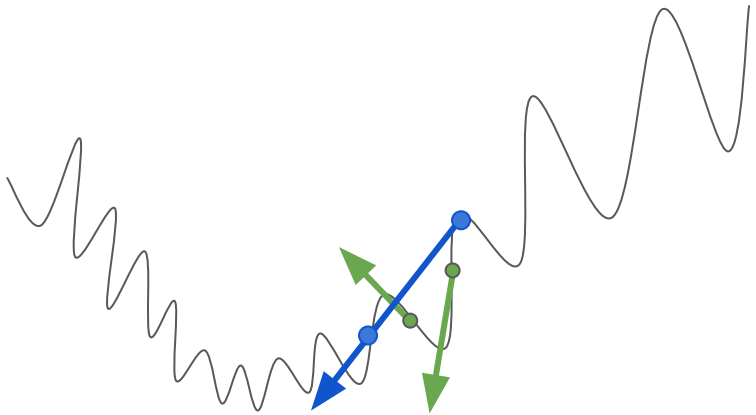
Reparameterization Gradient (RP)

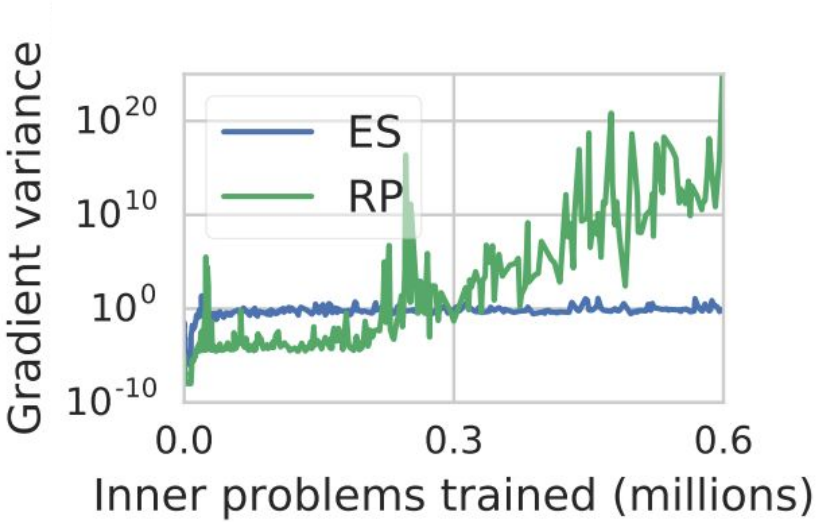
$$g_{\text{rp}} = \frac{1}{S} \sum_s \nabla_\theta L(\theta + \epsilon_s)$$

$$\epsilon_s \sim \mathcal{N}(0, \sigma^2 I)$$

Different variance properties

- If loss surface is smooth, reparameterization has lower variance.
- If loss surface has high curvature, ES has lower variance.





Combine them: inverse variance weighting

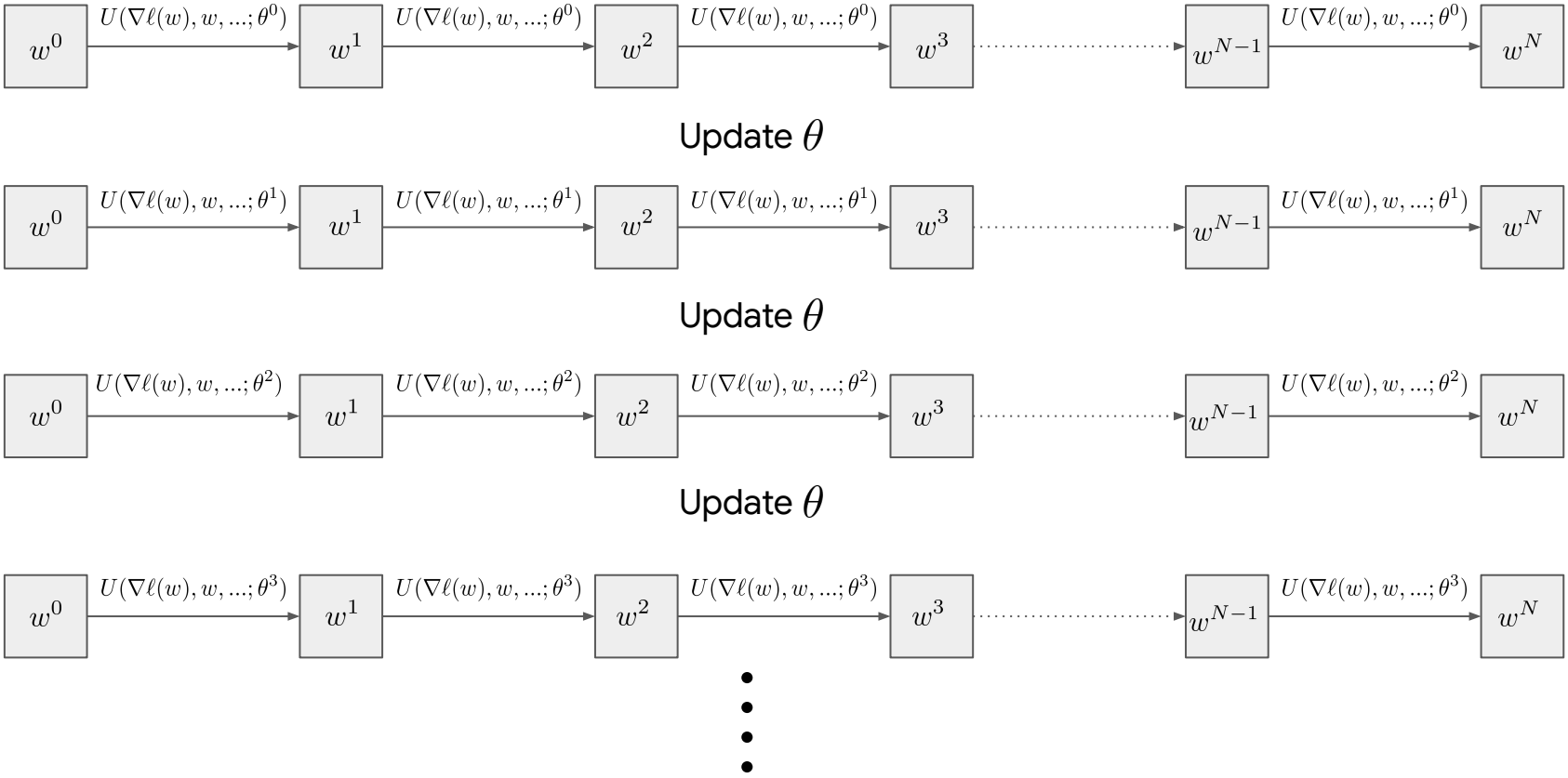
- Worst case, minimum of 2 variances
- Estimate variances ($\sigma^2_{\text{rp}}, \sigma^2_{\text{es}}$) empirically

$$g_{\text{merged}} = \frac{g_{\text{rp}}\sigma_{\text{rp}}^{-2} + g_{\text{es}}\sigma_{\text{es}}^{-2}}{\sigma_{\text{rp}}^{-2} + \sigma_{\text{es}}^{-2}}$$

See PIPPS (Parmas et. al. 2018) for more sophisticated methods for merging gradient estimators.

Problem: Outer-training is *extremely expensive*

Naive outer-training requires N^2 inner training steps



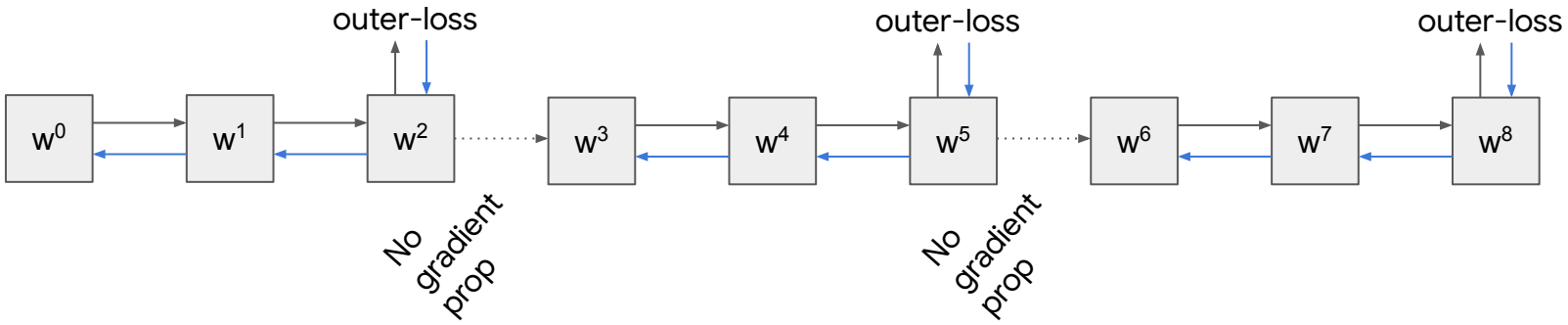
Problem: Outer-training is *extremely expensive*

Solutions:

- Massive parallelization
- Clever vectorization
- Partial unrolls
- Persistent Evolution Strategies

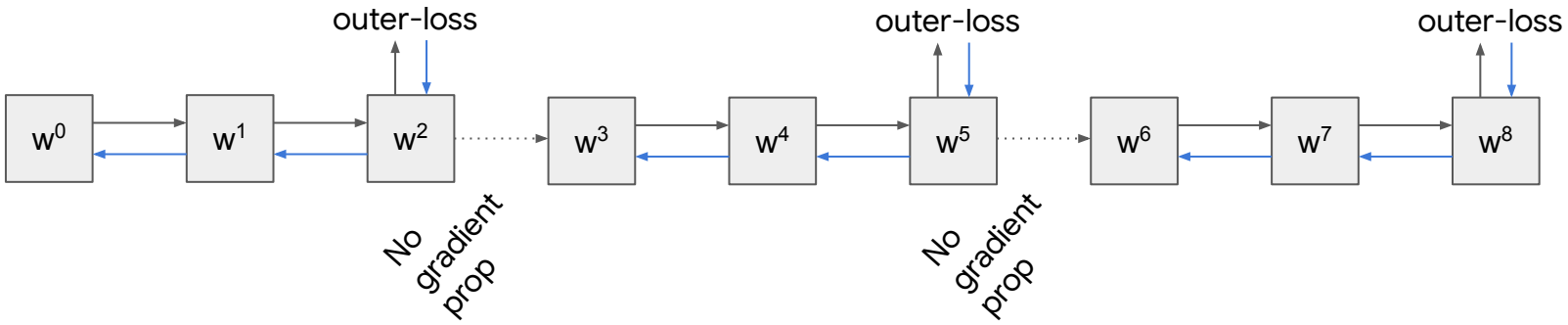
Partial Unrolls / Truncated Backprop Through Time

- Split sequence into multiple pieces



Partial Unrolls / Truncated Backprop Through Time

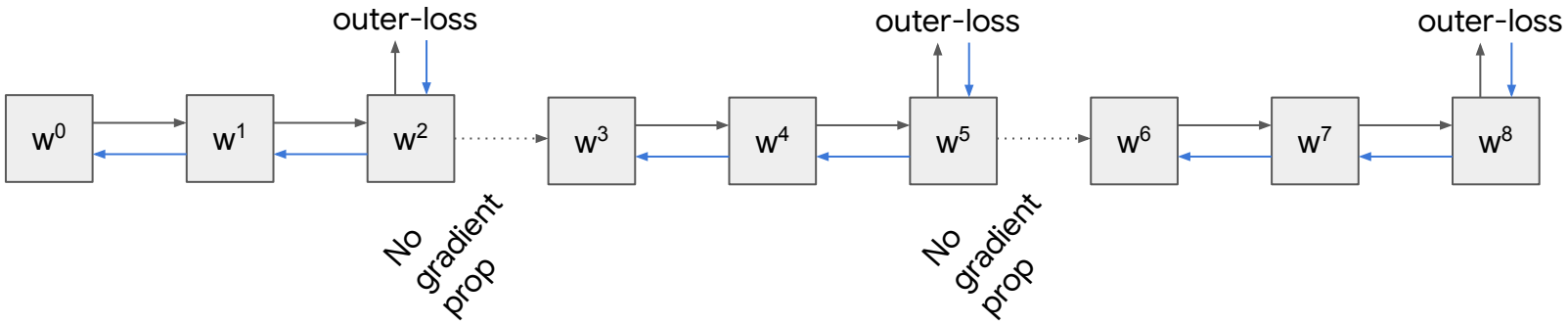
- Split sequence into multiple pieces



- More gradients (one computed per truncation)
- Better behaved loss surface

Partial Unrolls / Truncated Backprop Through Time

- Split sequence into multiple pieces



- More gradients (one computed per truncation)
- Better behaved loss surface
- Biased!

Short horizon bias often leads to overly conservative optimizers

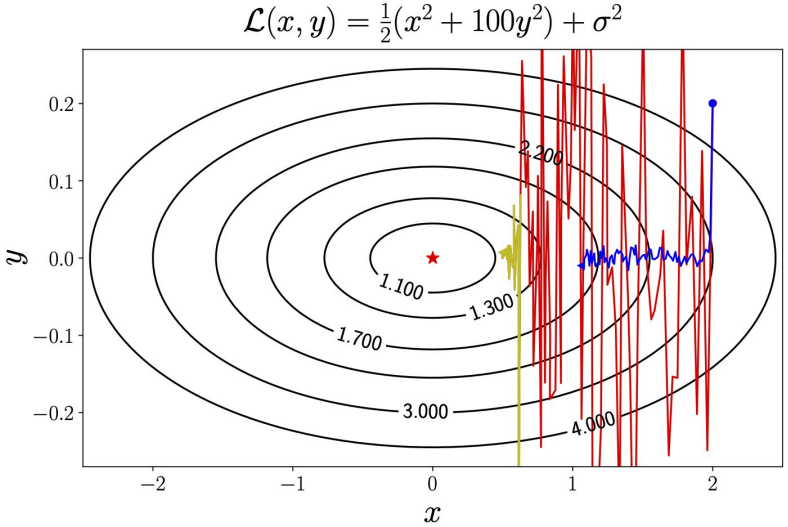
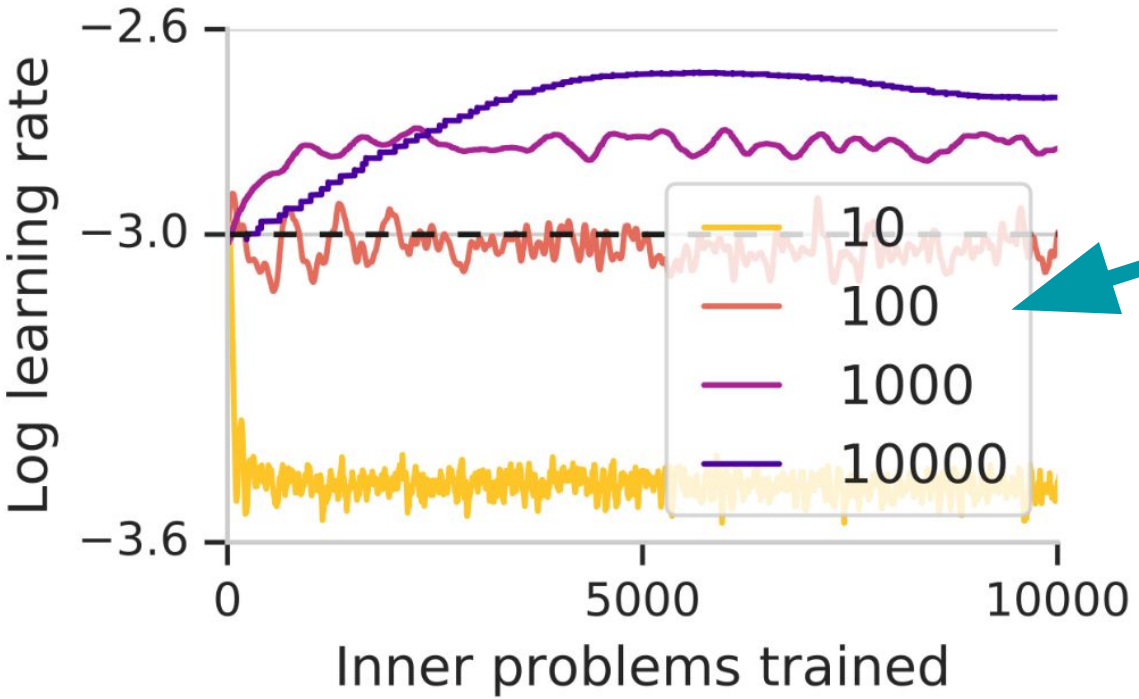


Figure 1: Aggressive learning rate (red) followed by a decay schedule (yellow) wins over conservative learning rate (blue) by making more progress along the low curvature direction (x direction).

[Wu, Ren, et al., 2018]

Short horizon bias often leads to overly conservative optimizers

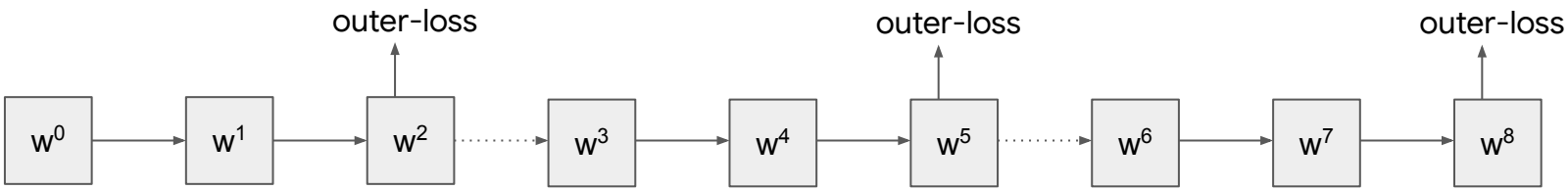


Truncated unroll lengths

[Metz, et al., 2018]

Persistent ES enables truncated unrolls without bias

PES splits the computation graph into a *series of truncated unrolls*
Performs an ES-style parameter *update after each unroll*



Eliminates bias from the truncations by accumulating correction terms
over the full sequence of unrolls

- Allows for *rapid, eventually unbiased, parameter updates*
- Inherits useful properties from ES:
  - Has low memory usage, does not require storing states for backprop
  - Unbiased gradient estimates for a *smoothed version of the loss surface*, which is useful for unrolled computations

[Vicol, et al., 2021]

ES & PES Algorithms

Algorithm 1 Truncated Evolution Strategies (ES) applied to partial unrolls of a computation graph.

Input: $\boldsymbol{w}_0$, initial state
K, truncation length for partial unrolls
N, number of particles
σ, standard deviation of perturbations
α, learning rate for ES optimization

Initialize $\boldsymbol{w} = \boldsymbol{w}_0$

repeat
  $\hat{\boldsymbol{g}}^{\text{ES}} \leftarrow \mathbf{0}$
  for $i = 1, \ldots, N$ **do**
    $\boldsymbol{\epsilon}^{(i)} = \begin{cases} \text{draw from } \mathcal{N}(0, \sigma^2 I) & i \text{ odd} \\ -\boldsymbol{\epsilon}^{(i-1)} & i \text{ even} \end{cases}$
    $\hat{L}_K^{(i)} \leftarrow \text{unroll}(\boldsymbol{w}, \boldsymbol{\theta} + \boldsymbol{\epsilon}^{(i)}, K)$
    $\hat{\boldsymbol{g}}^{\text{ES}} \leftarrow \hat{\boldsymbol{g}}^{\text{ES}} + \boldsymbol{\epsilon}^{(i)} \hat{L}_K^{(i)}$
  end for
  $\hat{\boldsymbol{g}}^{\text{ES}} \leftarrow \frac{1}{N\sigma^2} \hat{\boldsymbol{g}}^{\text{ES}}$
  $\boldsymbol{w} \leftarrow \text{unroll}(\boldsymbol{w}, \boldsymbol{\theta}, K)$
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \hat{\boldsymbol{g}}^{\text{ES}}$

Algorithm 2 Persistent evolution strategies (PES). Differences from ES are highlighted in purple.

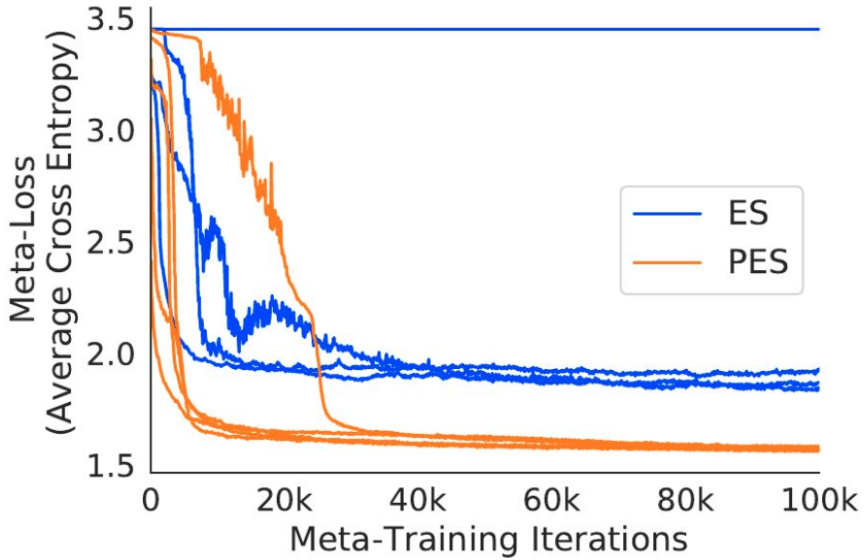
Input: $\boldsymbol{w}_0$, initial state
K, truncation length for partial unrolls
N, number of particles
σ, standard deviation of perturbations
α, learning rate for PES optimization

Initialize $\boldsymbol{w}^{(i)} = \boldsymbol{w}_0$ for $i \in \{1, \ldots, N\}$
Initialize $\boldsymbol{\xi}^{(i)} \leftarrow \mathbf{0}$ for $i \in \{1, \ldots, N\}$
repeat
  $\hat{\boldsymbol{g}}^{\text{PES}} \leftarrow \mathbf{0}$
  for $i = 1, \ldots, N$ **do**
    $\boldsymbol{\epsilon}^{(i)} = \begin{cases} \text{draw from } \mathcal{N}(0, \sigma^2 I) & i \text{ odd} \\ -\boldsymbol{\epsilon}^{(i-1)} & i \text{ even} \end{cases}$
    $\boldsymbol{w}^{(i)}, \hat{L}_K^{(i)} \leftarrow \text{unroll}(\boldsymbol{w}^{(i)}, \boldsymbol{\theta} + \boldsymbol{\epsilon}^{(i)}, K)$
    $\boldsymbol{\xi}^{(i)} \leftarrow \boldsymbol{\xi}^{(i)} + \boldsymbol{\epsilon}^{(i)}$
    $\hat{\boldsymbol{g}}^{\text{PES}} \leftarrow \hat{\boldsymbol{g}}^{\text{PES}} + \boldsymbol{\xi}^{(i)} \hat{L}_K^{(i)}$
  end for
  $\hat{\boldsymbol{g}}^{\text{PES}} \leftarrow \frac{1}{N\sigma^2} \hat{\boldsymbol{g}}^{\text{PES}}$
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \hat{\boldsymbol{g}}^{\text{PES}}$

[Vicol,et al., 2021]

Training Learned Optimizers with truncated ES and PES

- We *meta-train an MLP-based learned optimizer* as described in Metz et al. (2019)
- This optimizer is used to train a *two hidden-layer, 128 unit, MLP on CIFAR-10*
- Due to PES's unbiased nature, *PES achieves both lower losses, and is more consistent across random initializations of the learned optimizer*



[Vicol,et al., 2021]

What can learned optimizers do now?

When meta-trained on a specific task, learned optimizers outperform tuned hand-designed optimizers on that task

$$\text{argmin}_\theta \mathbb{E}_{t \sim \mathcal{T}} \text{OuterObjective}(\text{InnerLoop}(t; \theta))$$

Find the weights of the learned optimizer via variational optimization + truncated backprop

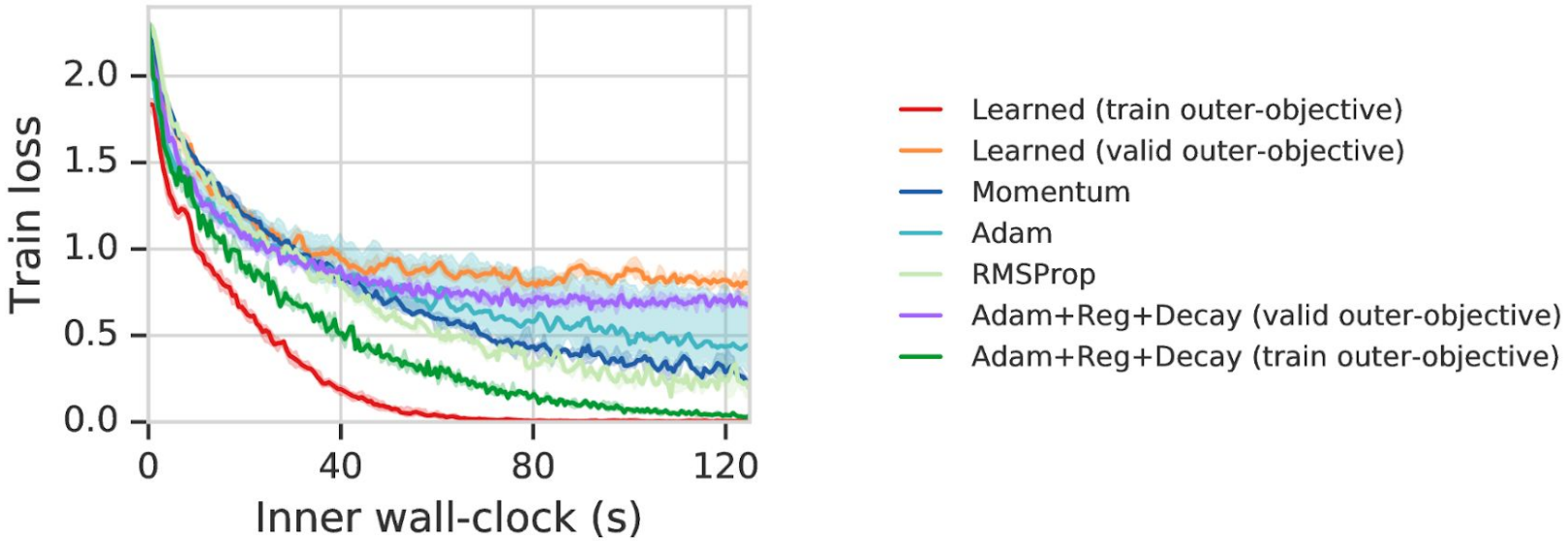
Tasks: 10 way classification problems sampled from imagenet

Training loss, or validation loss

Applying the learned optimizer to train a small ConvNet

When meta-trained on a specific task, learned optimizers outperform tuned hand-designed optimizers on that task

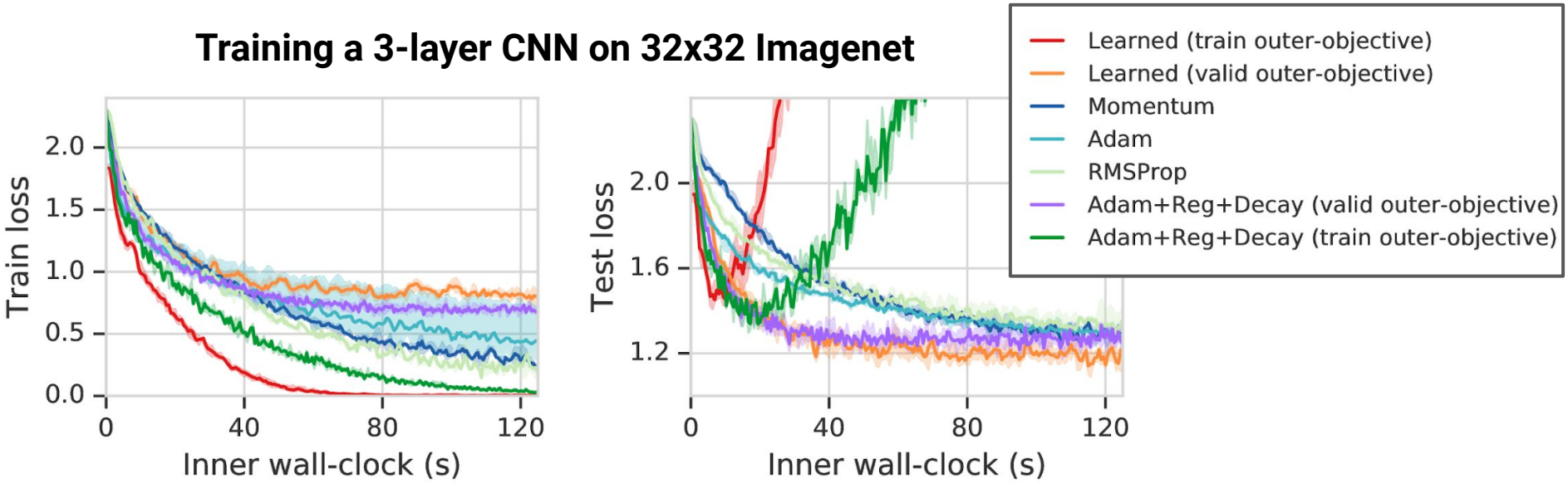
Training a 3-layer CNN on 32x32 Imagenet



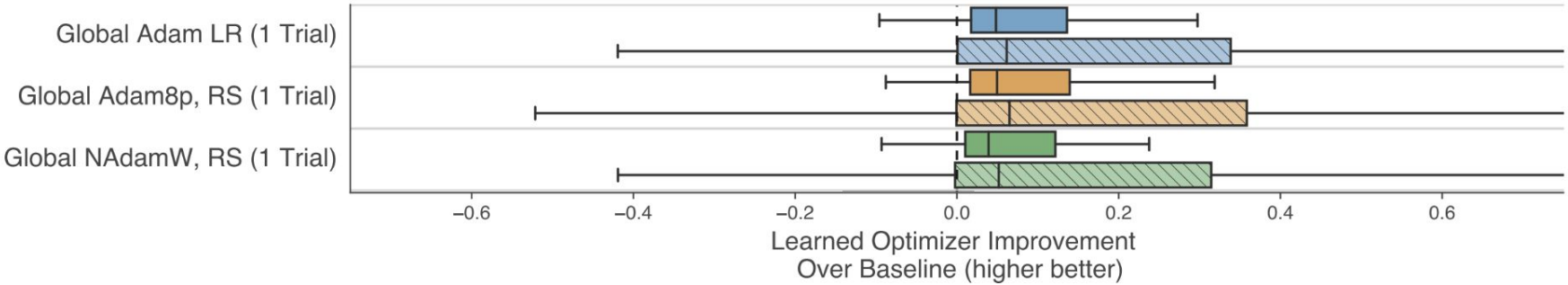
[Metz, et al, 2019]

When meta-trained on a specific task, learned optimizers outperform tuned hand-designed optimizers on that task

Learned optimizers can be outer-trained to target generalization

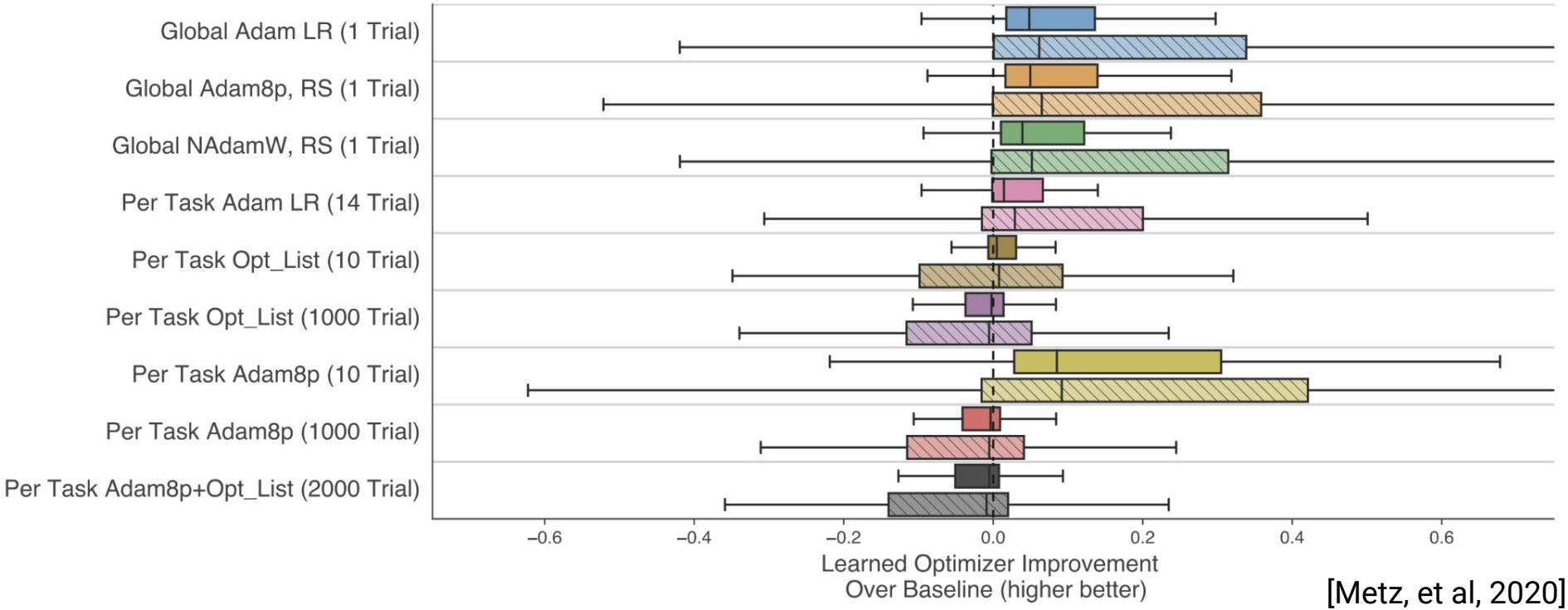


Learned optimizers outperform hand-designed optimizers on new tasks, when the hand-designed optimizers have limited hyperparameter tuning budget



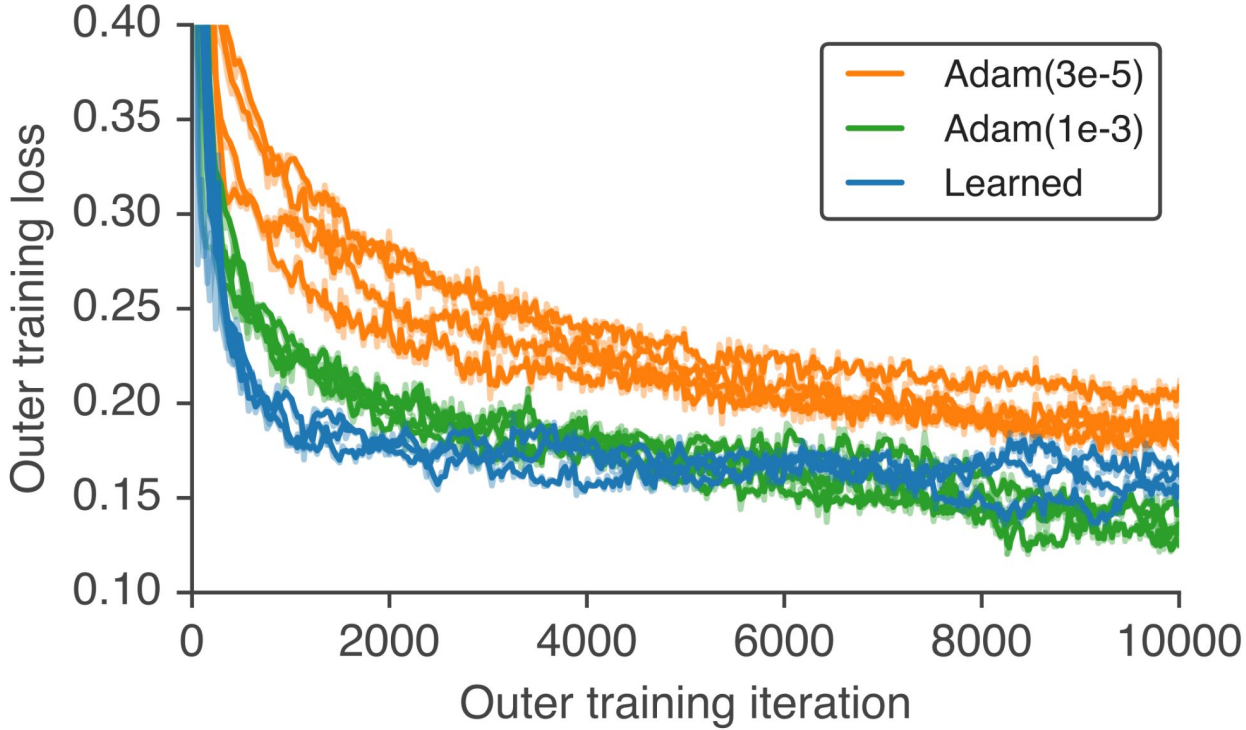
[Metz, et al, 2020]

Learned optimizers outperform hand-designed optimizers on new tasks, when the hand-designed optimizers have limited hyperparameter tuning budget



[Metz, et al, 2020]

Learned optimizers can train themselves from scratch



[Metz, et al, 2020]

Learned optimizers can target objectives that are impossible to specify for hand-designed optimizers

Unsupervised representation learning

Goal: Expose high level attributes of data (e.g. object identity) without labels.

Unsupervised representation learning

Goal: Expose high level attributes of data (e.g. object identity) without labels.

Challenge: Find a learning rule that optimizes for attributes that we *do not know* at training time.

Unsupervised representation learning

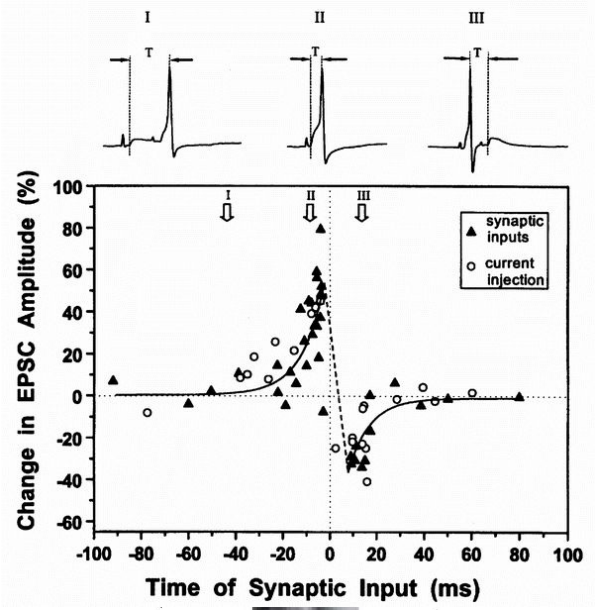
Goal: Expose high level attributes of data (e.g. object identity) without labels.

Current Approaches:

-*Hand design* a cost function that does not rely on labels

-*Hand design* a target statistic for the representation

-*Hand design* an update rule



Zhang et. al. 1998



Encoder

Hinton and Salakhutdinov 2006

Unsupervised representation learning

Goal: Expose high level attributes of data (e.g. object identity) without labels.

Challenge: Find a learning rule that optimizes for attributes that we *do not know* at training time.

Current approaches: Hand design a surrogate learning rule, and hope a useful representation arises as a side effect.

Unsupervised representation learning

Goal: Expose high level attributes of data (e.g. object identity) without labels.

Challenge: Find a learning rule that optimizes for attributes that we *do not know* at training time.

Current approaches: Hand design a surrogate learning rule, and hope a useful representation arises as a side effect.

- Mixed success (e.g. compare to supervised transfer learning)

Unsupervised representation learning

Goal: Expose high level attributes of data (e.g. object identity) without labels.

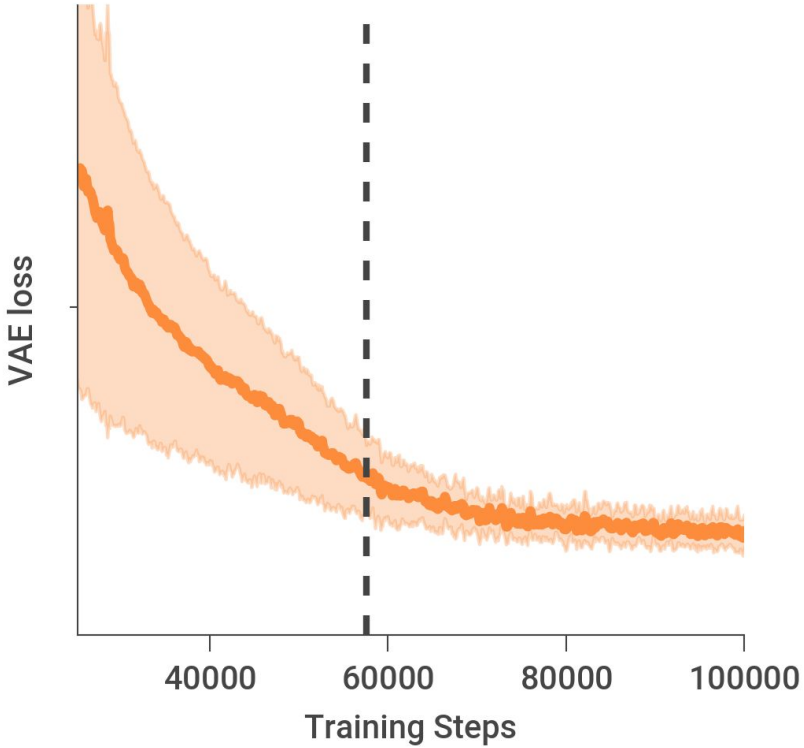
Challenge: Find a learning rule that optimizes for attributes that we *do not know* at training time.

Current approaches: Hand design a surrogate learning rule, and hope a useful representation arises as a side effect.

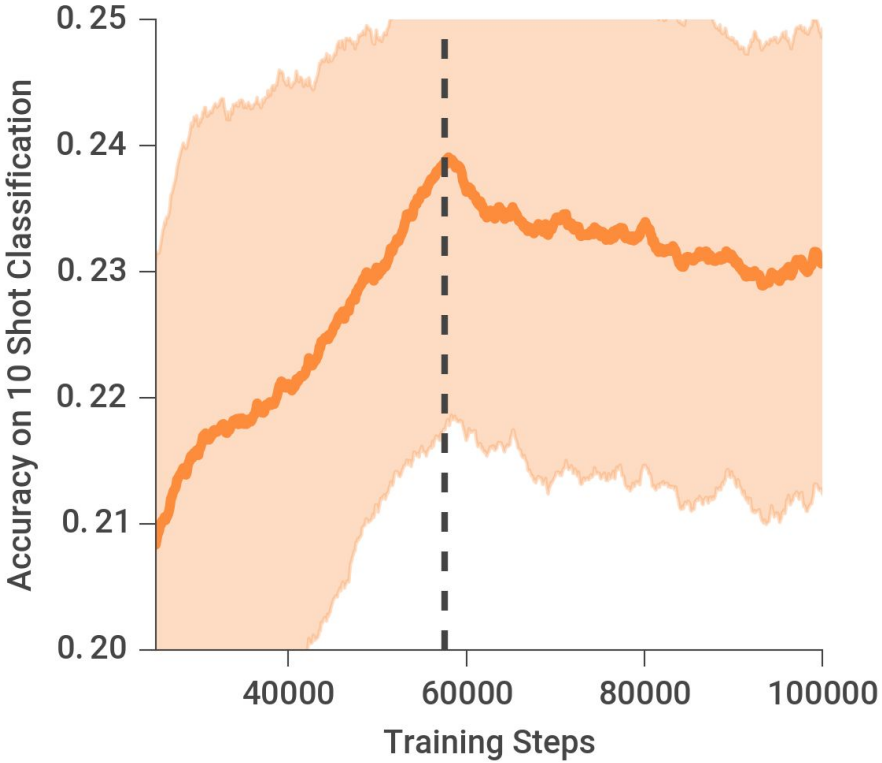
- Mixed success (e.g. compare to supervised transfer learning)
- *Objective mismatch!*

Hand designed surrogates optimize the wrong thing

Variational bound on log likelihood



Goal: few shot classification



Unsupervised representation learning

Goal: Expose high level attributes of data (e.g. object identity) without labels.

Challenge: Find a learning rule that optimizes for attributes that we *do not know* at training time.

Current approaches: Hand design a surrogate learning rule, and hope a useful representation arises as a side effect. *Objective mismatch!*

Unsupervised representation learning

Goal: Expose high level attributes of data (e.g. object identity) without labels.

Challenge: Find a learning rule that optimizes for attributes that we *do not know* at training time.

Current approaches: Hand design a surrogate learning rule, and hope a useful representation arises as a side effect. *Objective mismatch!*

Observation: There are many tasks for which we *do know* desired high level attributes.

Unsupervised representation learning

Goal: Expose high level attributes of data (e.g. object identity) without labels.

Challenge: Find a learning rule that optimizes for attributes that we *do not know* at training time.

Current approaches: Hand design a surrogate learning rule, and hope a useful representation arises as a side effect. *Objective mismatch!*

Observation: There are many tasks for which we *do know* desired high level attributes.

Proposed approach: Meta-learn an unsupervised learning rule that results in correct high level attributes for these tasks. Apply to new tasks.

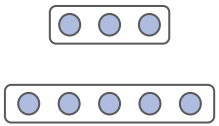
Meta-learn an unsupervised learning rule that generalizes

Unsupervised learning rule should generalize to

- New datasets, new data *modalities*
  - Meta-train on ensemble of tasks
    - Different datasets (e.g. mini-imagenet, CIFAR, glyph)
    - Different prediction targets (e.g. class labels, rotation angle)
  - Augment datasets to prevent memorization (e.g. randomly permute input dimensions)

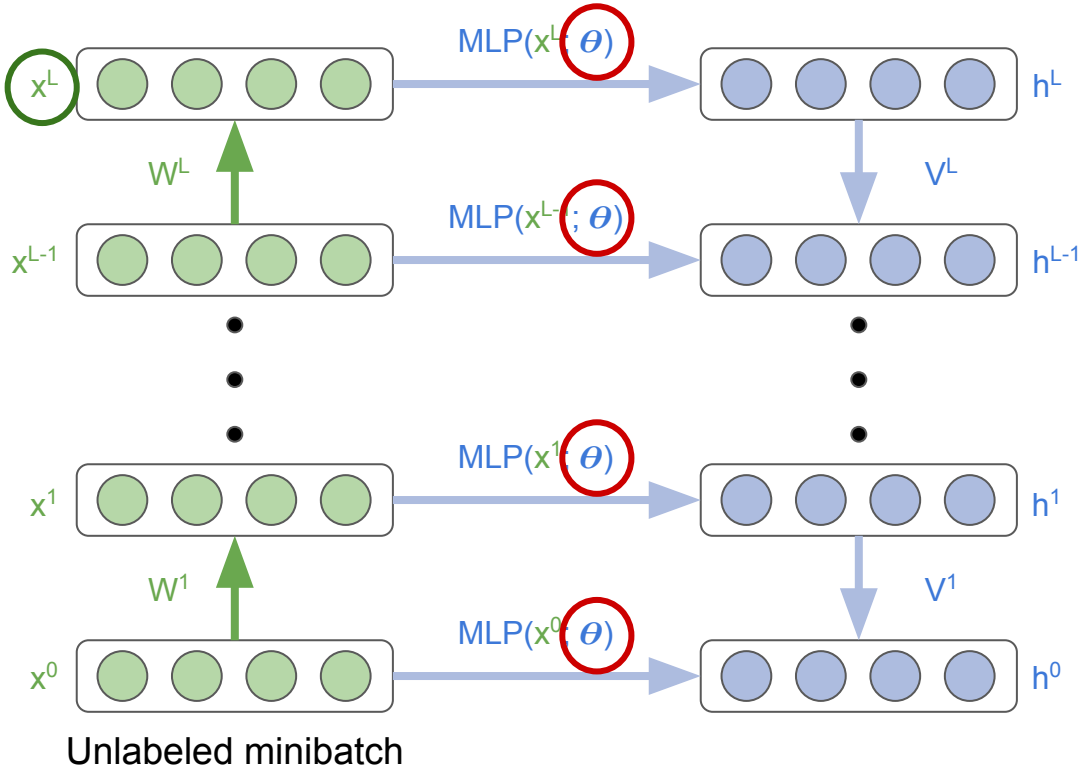


- New neural network architectures (depth, width, topology)
  - Meta-train on ensemble of architectures
  - Make learning rule *local*



Unsupervised meta-learning architecture (simplified)



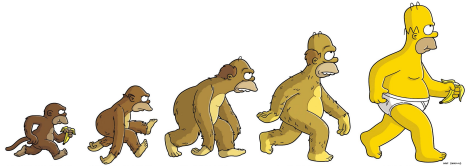


- Learned representation
- MLP is convolution over *minibatch* and *unit* dimensions
- Neuron-local updates

$$\Delta W^l_{mn} = h^l_m h^{l-1}_n$$

- Meta-train $\boldsymbol{\theta}$

Meta-objective: generalization from few shot training



- Infer regression weights on small labeled minibatch $\{x^0, y\}$

$$\hat{v} = \underset{v}{\operatorname{argmin}} \left(\left\| y - v^T x^L \right\|^2 + \lambda \left\| v \right\|^2 \right)$$

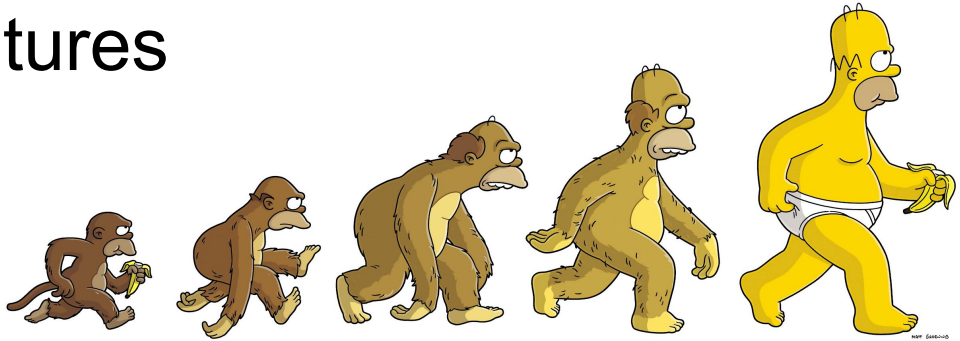
- Evaluate predictions on second small minibatch $\{\tilde{x}^0, \tilde{y}\}$ (for generalization)

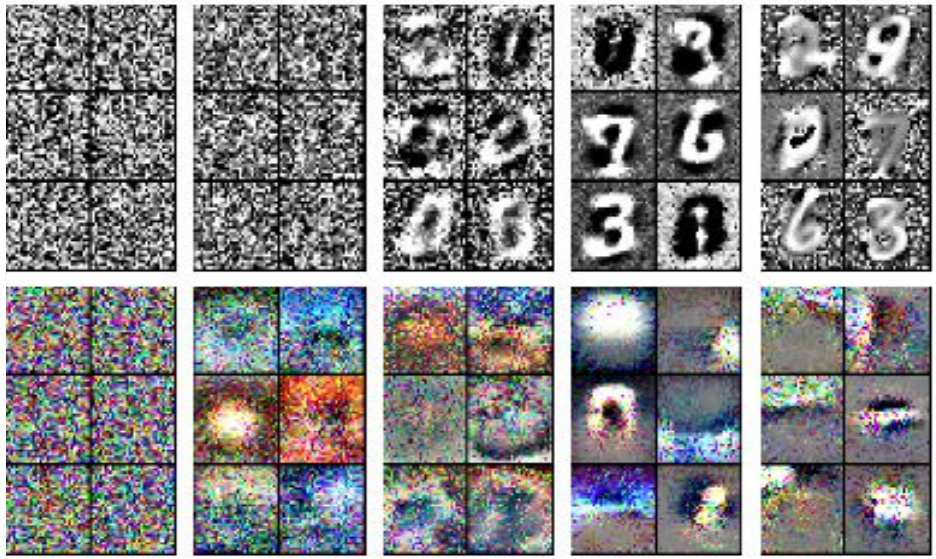
$$\text{MetaObjective}(\cdot; \theta) = \text{CosDist}\left(\tilde{y}, \hat{v}^T \tilde{x}^L\right)$$

- Meta-train by truncated backprop through unrolled unsupervised training steps

$$\Delta\theta \propto -\frac{\partial \left[\text{MetaObjective}(\textit{unrolled unsupervised training}; \theta)\right]}{\partial \theta}$$

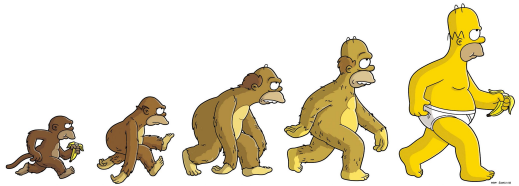
Learned features



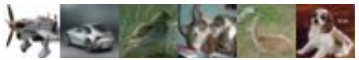


Evaluating the Learning Rule

MetaTraining



Train the Learning Rule

CIFAR-10 

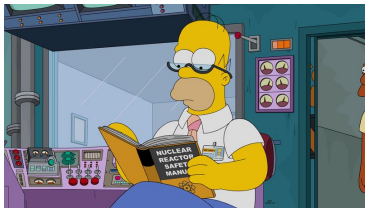
Glyph-Alphabet 

Glyph-Numbers 

ImageNet 

Transfer

Evaluation



Use the Learning Rule

Fashion-MNIST 

MNIST 

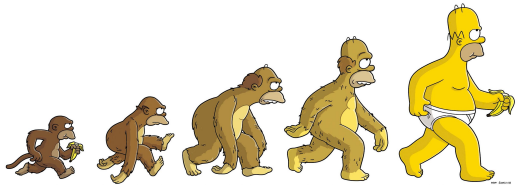
Learning rule generalizes to new datasets





Evaluating the Learning Rule

MetaTraining



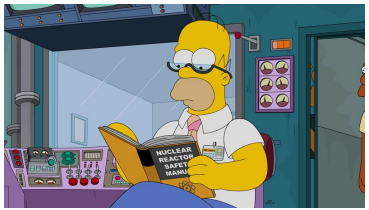
Train the Learning Rule

2-5 layers

64-512 units per layer

Transfer

Evaluation



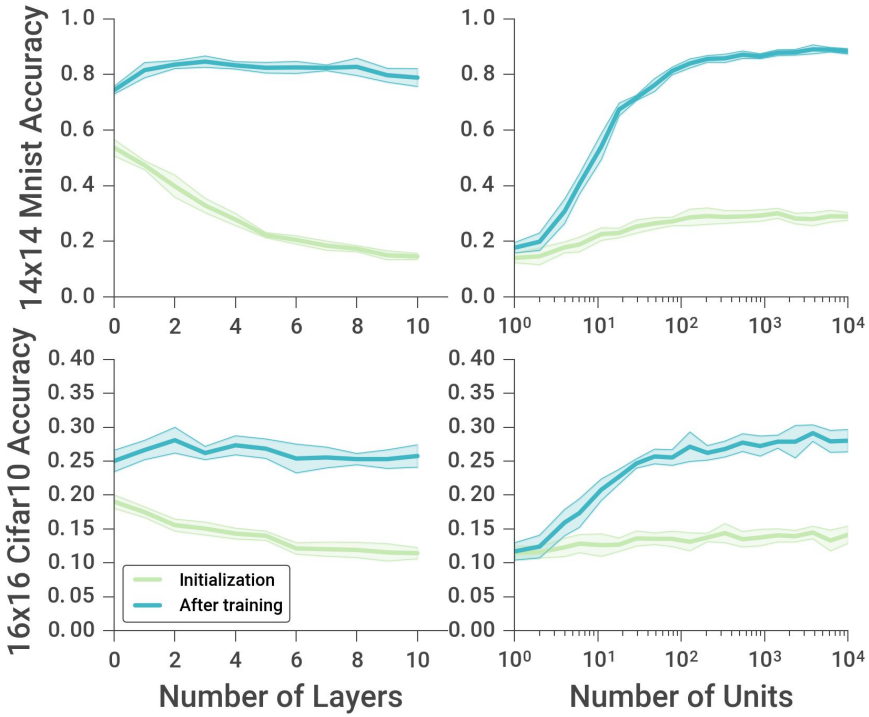
Use the Learning Rule

1-11 layers

8-10,000 units per layer

Learning rule generalizes to new architectures

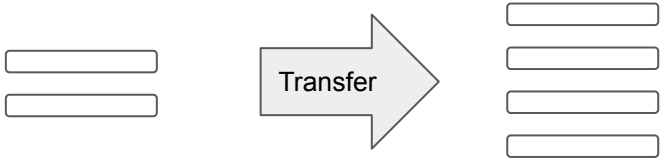




Number of Layers

1-4 layers

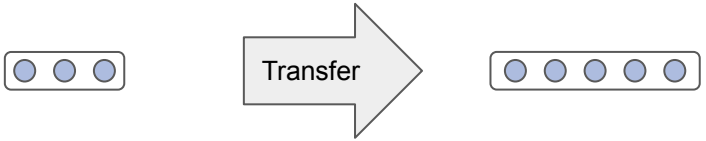
10 layers



Number of Units

64-512 units

10,000 units



New tools are making learned optimizer research *practical*

- `learned_optimizers` library for defining, outer-training, and applying learned optimizers
- Soon to be open sourced! (timescale = weeks)
- **colab demo**

Thank you!

- The coming meta-learning revolution
- What is a learned optimizer?
- Open problems in learned optimizers (with a few solutions)
  - Outer-training sets
  - Interpreting mechanisms of action
  - Engineering inductive biases
  - Chaotic outer-loss landscapes
  - Computational cost
  - Truncated unroll bias
- What learned optimizers can achieve now
  - Outperform hand-designed optimizers on narrow domain, or with limited hyperparameter tuning budget
- Novel capabilities
  - Unsupervised representation learning
- `learned_optimizers` library (soon to be open-sourced) + colab demo

END OF TALK