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
  ○ Trimmed 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.

![Flowchart diagram showing the transition from engineering features to learning features]

- SIFT (Lowe et al. 1999)
- HOG (Dalal et al. 2005)
- LeNet (LeCun et al. 1998)
- AlexNet (Krizhevsky et al. 2012)

- 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)
Unsupervised Learning (Metz, et. al. 2019)
Neural Architecture Search (Zoph, Le, 2016)
What is a learned optimizer?
$w^0$
$w^0 \quad w - \alpha \nabla \ell(w) \quad w^1$
$w^0 \rightarrow w - \alpha \nabla \ell(w) \rightarrow w^1 \rightarrow w - \alpha \nabla \ell(w) \rightarrow w^2 \rightarrow w - \alpha \nabla \ell(w) \rightarrow w^3 \rightarrow \vdots \rightarrow w^{N-1} \rightarrow w - \alpha \nabla \ell(w) \rightarrow w^N$

Measure performance
Measure performance
Measure performance

Update $\theta$

\[ w^0 \xrightarrow{U(\nabla \ell(w), w, ..., \theta^0)} w^1 \xrightarrow{U(\nabla \ell(w), w, ..., \theta^0)} w^2 \xrightarrow{U(\nabla \ell(w), w, ..., \theta^0)} w^3 \xrightarrow{\ldots} w^{N-1} \xrightarrow{U(\nabla \ell(w), w, ..., \theta^0)} w^N \]

\[ w^0 \xrightarrow{U(\nabla \ell(w), w, ..., \theta^1)} w^1 \xrightarrow{U(\nabla \ell(w), w, ..., \theta^1)} w^2 \xrightarrow{U(\nabla \ell(w), w, ..., \theta^1)} w^3 \xrightarrow{\ldots} w^{N-1} \xrightarrow{U(\nabla \ell(w), w, ..., \theta^1)} w^N \]
Outer-training. Meta-Training. Update $\theta$

Measure performance

Update $\theta$

Update $\theta$

Update $\theta$

...
Inner-training. Given update rule parameters $\theta$ update $w$

Update $\theta$

Measure performance
Example learned optimizer architecture: per-parameter NN

Parametric update (U):

\[
\text{gradient} \rightarrow \text{per-parameter update fn. (RNN / MLP)} \rightarrow \Delta w
\]

Same RNN/MLP applied to every parameter


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

Generalizing across scale remains challenging!

[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

[Attractor fixed points vary as a function of input]

[Update function at different fixed points]

[Effective learning rate decreases with increasing gradient magnitude]

[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
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
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.
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 \textit{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

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

- Sample from distribution
- Parameters of the distribution
- Original loss
Two unbiased gradient estimators

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

**Evolution Strategies (ES)**

**Score Function**

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

$$
\tilde{\theta}_s \sim \mathcal{N} (\theta, \sigma^2 I)
$$
Two unbiased gradient estimators

\[ \nabla_\theta \mathcal{L} (\theta) = ? \]

Evolution Strategies (ES)
Score Function

\[ g_{es} = \frac{1}{S\sigma^2} \sum_s L(\theta + \epsilon_s) \epsilon_s \]

\[ \epsilon_s \sim \mathcal{N}(0, \sigma^2 I) \]
Two unbiased gradient estimators

\[ \nabla_\theta \mathcal{L} (\theta) = \]?

Evolution Strategies (ES)
Score Function

\[ g_{es} = \frac{1}{2S\sigma^2} \sum_s (L(\theta + \epsilon_s) - L(\theta - \epsilon_s)) \epsilon_s \]

Antithetic samples reduce variance

\[ \epsilon_s \sim \mathcal{N}(0, \sigma^2 I) \]
Two unbiased gradient estimators

\[ \nabla_\theta \mathcal{L} (\theta) = ? \]

Evolution Strategies (ES) Score Function

\[ g_{es} = \frac{1}{2S\sigma^2} \sum_s (L(\theta + \epsilon_s) - L(\theta - \epsilon_s)) \epsilon_s \]

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

Reparameterization Gradient (RP)

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

\[ \mathcal{L}(\theta) = \mathbb{E}_{\tilde{\theta} \sim \mathcal{N}(\theta, \sigma^2 I)} [L(\tilde{\theta})] \]
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 ($\sigma_{rp}^2$, $\sigma_{es}^2$) empirically

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

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

![Diagram showing partial unrolls/truncated backprop through time]

- 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

\[ \mathcal{L}(x, y) = \frac{1}{2}(x^2 + 100y^2) + \sigma^2 \]

**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).
Short horizon bias often leads to overly conservative optimizers

[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:** \( w_0 \), initial state  
\( K \), truncation length for partial unrolls  
\( N \), number of particles  
\( \sigma \), standard deviation of perturbations  
\( \alpha \), learning rate for ES optimization

Initialize \( w = w_0 \)

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

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

**Input:** \( w_0 \), initial state  
\( K \), truncation length for partial unrolls  
\( N \), number of particles  
\( \sigma \), standard deviation of perturbations  
\( \alpha \), learning rate for PES optimization

Initialize \( w^{(i)} = w_0 \) for \( i \in \{1, \ldots, N\} \)
Initialize \( \xi^{(i)} = 0 \) for \( i \in \{1, \ldots, N\} \)

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

[Vicol, et al., 2021]
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.

\[
\argmin_{\theta} \mathbb{E}_{t \sim \mathcal{T}} \text{OuterObjective}(\text{InnerLoop}(t; \theta))
\]

- 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]
Learned optimizers can be outer-trained to target generalization when meta-trained on a specific task, they outperform tuned hand-designed optimizers on that task.

**Diagram:**
- **Training a 3-layer CNN on 32x32 Imagenet**

  - **Train loss** vs. **Inner wall-clock (s)**
  - **Test loss** vs. **Inner wall-clock (s)**

  - **Learned** (train outer-objective)
  - **Learned** (valid outer-objective)
  - Momentum
  - Adam
  - RMSProp
  - Adam+Reg+Decay (valid outer-objective)
  - Adam+Reg+Decay (train outer-objective)
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.
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
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

VAE loss

Training Steps

VAE loss

Accuracy on 10 Shot Classification

Training Steps
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)

Unlabeled minibatch

- 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 \( \theta \)
Meta-objective: generalization from few shot training

- Infer regression weights on small labeled minibatch $\{x^0, y\}$
  $$\hat{v} = \arg\min_v \left( \|y - v^T x^L\|^2 + \lambda \|v\|^2 \right)$$

- Evaluate predictions on second small minibatch $\{\tilde{x}^0, \tilde{y}\}$ (for generalization)
  $$\text{MetaObjective}(\cdot; \theta) = \text{CosDist}(\tilde{y}, \hat{v}^T \tilde{x}^L)$$

- Meta-train by truncated backprop through unrolled unsupervised training steps
  $$\Delta \theta \propto -\frac{\partial \left[ \text{MetaObjective}(\text{unrolled unsupervised training}; \theta) \right]}{\partial \theta}$$
Learned features

First Layer Weights $W^1$

Early Learning Rules

Later Learning Rules
Evaluating the Learning Rule

MetaTraining

- CIFAR-10
- Glyph-Alphabet
- Glyph-Numbers
- ImageNet

Train the Learning Rule

Evaluation

- Fashion-MNIST
- MNIST

Use the Learning Rule

Transfer
Learning rule generalizes to new datasets
Evaluating the Learning Rule

MetaTraining

Train the Learning Rule

- 2-5 layers
- 64-512 units per layer

Evaluation

Use the Learning Rule

- 1-11 layers
- 8-10,000 units per layer

Transfer
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