Advanced Meta-Learning Topics: Large-Scale Meta-Optimization

CS 330

Course Reminders

Optional HW4 out today, due **next Wednesday.**

Please submit high-resolution feedback!

Don’t forget to turn off your Azure machines!

Meta-learning and Scalability

Hand-designed priors

Hand-designed rules

Hand-designed features

End-to-end learning

Learned priors

Meta-learning

Core lesson: given enough data and compute, learned components outperform even the best hand-designed heuristics.

The key advantage of data-driven approaches is scalability.

Do meta-learning methods work at scale?

Plan for Today

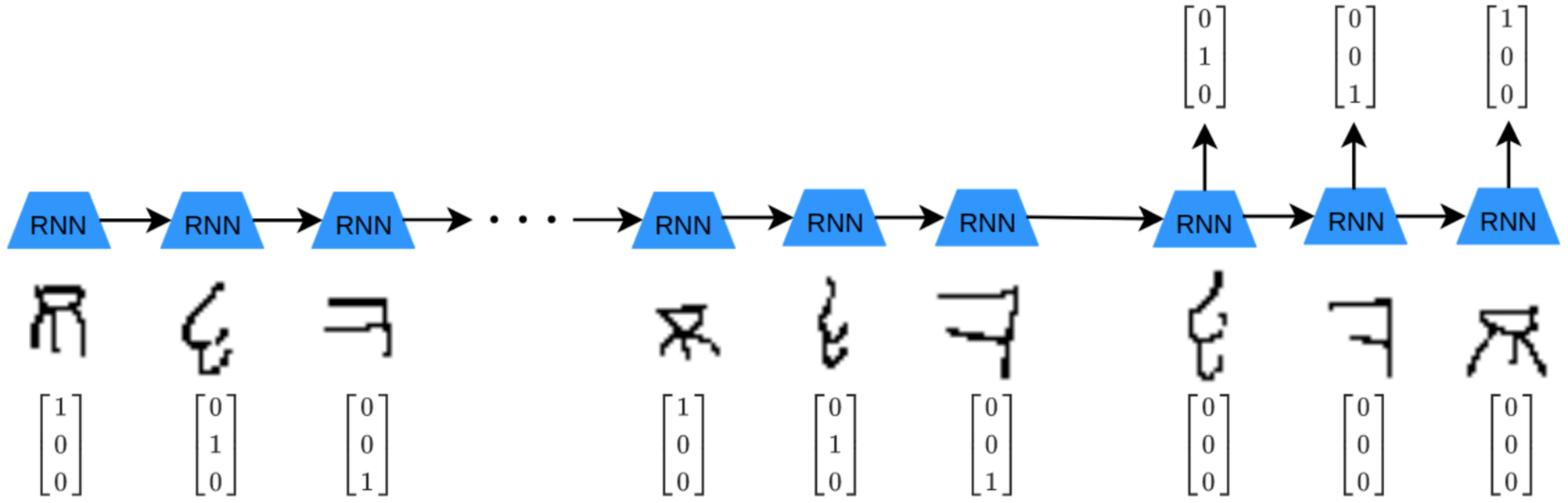
Why consider *large-scale meta-optimization*?

Applications

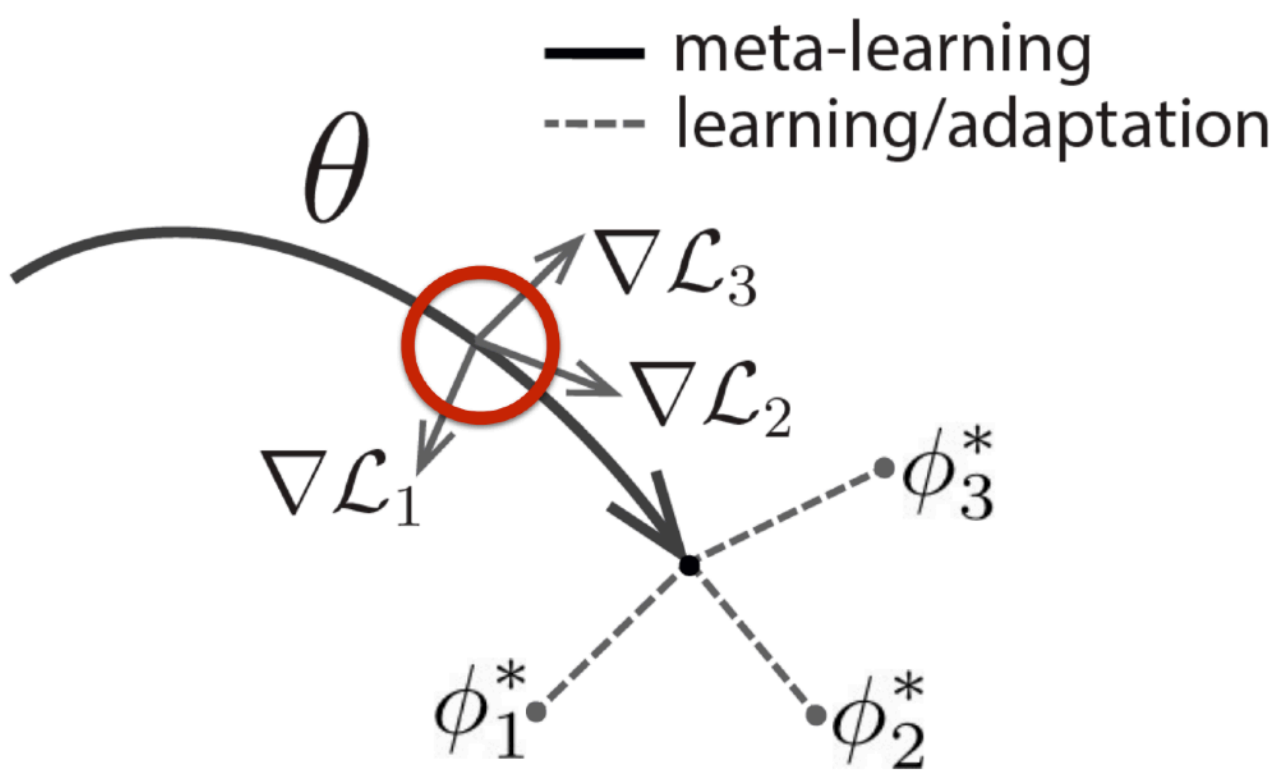
Methods
- Truncated backpropagation
- Gradient-free optimization

Goals for by the end of lecture:
- Know scenarios where existing meta-learning approaches fail to scale
- Understand techniques for large-scale meta-optimization

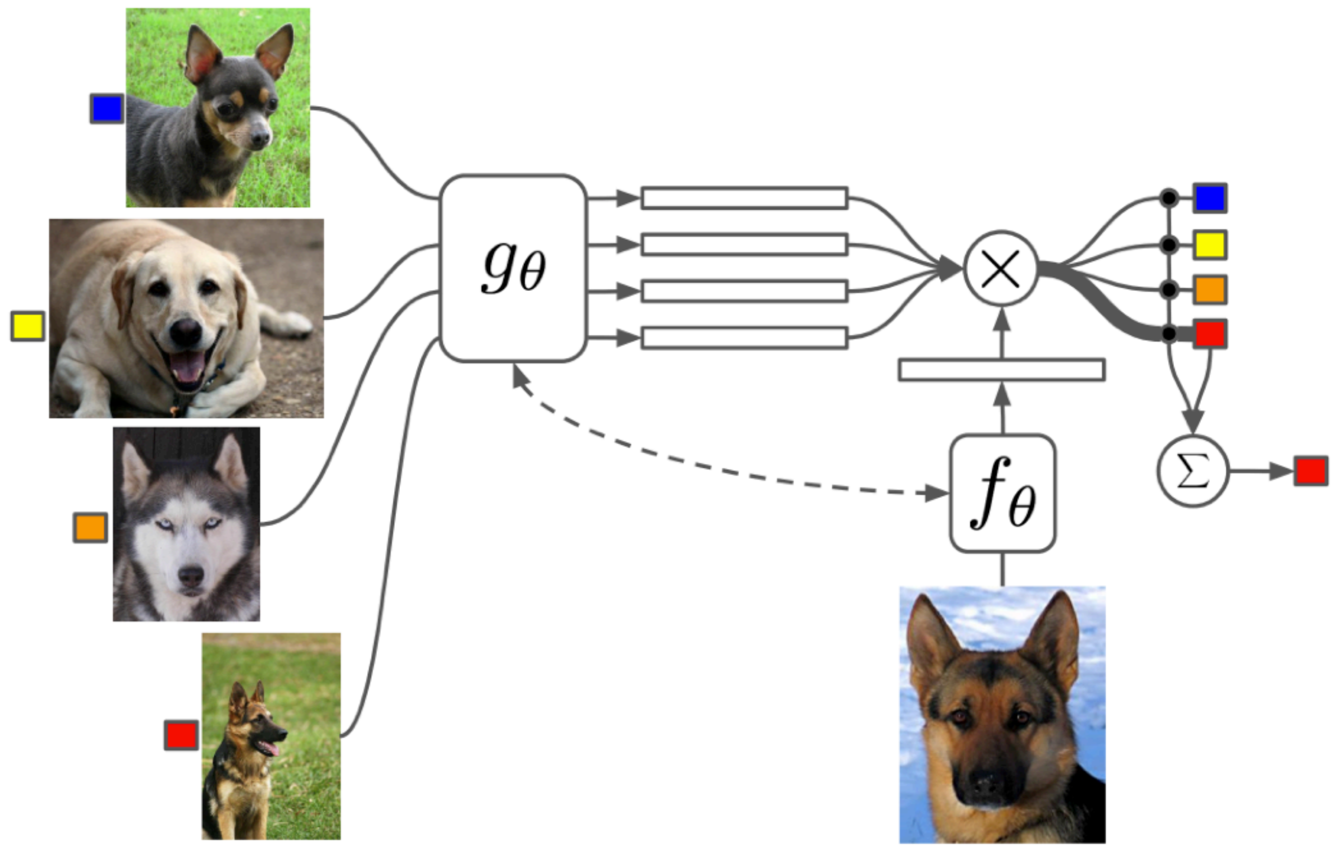
Direct Backpropagation



Black-box



Optimization-based



Nonparametric

General recipe: Consider the inner loop as one big computation graph, then backpropagate.

+ Automatically works with any differentiable computation graph

- **Memory cost** scales with **computation graph size!**

How Big are Computation Graphs?

model	backbone	miniImageNet 5-way		tieredImageNet 5-way	
		1-shot	5-shot	1-shot	5-shot
Meta-Learning LSTM* [22]	64-64-64-64	43.44 ± 0.77	60.60 ± 0.71	-	-
Matching Networks* [33]	64-64-64-64	43.56 ± 0.84	55.31 ± 0.73	-	-
MAML [8]	32-32-32-32	48.70 ± 1.84	63.11 ± 0.92	51.67 ± 1.81	70.30 ± 1.75
Prototypical Networks*† [28]	64-64-64-64	49.42 ± 0.78	68.20 ± 0.66	53.31 ± 0.89	72.69 ± 0.74
Relation Networks* [29]	64-96-128-256	50.44 ± 0.82	65.32 ± 0.70	54.48 ± 0.93	71.32 ± 0.78
R2D2 [3]	96-192-384-512	51.2 ± 0.6	68.8 ± 0.1	-	-
Transductive Prop Nets [14]	64-64-64-64	55.51 ± 0.86	69.86 ± 0.65	59.91 ± 0.94	73.30 ± 0.75
SNAIL [18]	ResNet-12	55.71 ± 0.99	68.88 ± 0.92	-	-
Dynamic Few-shot [10]	64-64-128-128	56.20 ± 0.86	73.00 ± 0.64	-	-
AdaResNet [19]	ResNet-12	56.88 ± 0.62	71.94 ± 0.57	-	-
TADAM [20]	ResNet-12	58.50 ± 0.30	76.70 ± 0.30	-	-
Activation to Parameter† [21]	WRN-28-10	59.60 ± 0.41	73.74 ± 0.19	-	-
LEO† [25]	WRN-28-10	61.76 ± 0.08	77.59 ± 0.12	**66.33 ± 0.05**	**81.44 ± 0.09**
MetaOptNet-RR (ours)	ResNet-12	61.41 ± 0.61	77.88 ± 0.46	**65.36 ± 0.71**	**81.34 ± 0.52**
MetaOptNet-SVM (ours)	ResNet-12	62.64 ± 0.61	78.63 ± 0.46	**65.99 ± 0.72**	**81.56 ± 0.53**
MetaOptNet-SVM-trainval (ours)†	ResNet-12	**64.09 ± 0.62**	**80.00 ± 0.45**	**65.81 ± 0.74**	**81.75 ± 0.53**

4-layer CNN
Parameters: <1e5

WRN-28-10
Parameters: <4e6

ResNet-12
Parameters: <1e7

From: "Meta-Learning with Differentiable Convex Optimization", Lee et al.

How Big are Computation Graphs?

```python
class NeuralNetwork(nn.Module):
    def __init__(self):
        super().__init__()
        self.flatten = nn.Flatten()
        self.linear_relu_stack = nn.Sequential(
            nn.Linear(28*28, 512),
            nn.ReLU(),
            nn.Linear(512, 512),
            nn.ReLU(),
            nn.Linear(512, 10)
        )

    def forward(self, x):
        x = self.flatten(x)
        logits = self.linear_relu_stack(x)
        return logits
```

```python
epochs = 5
for t in range(epochs):
    print(f"Epoch {t+1}\n-------------------------------")
    train(train_dataloader, model, loss_fn, optimizer)
    test(test_dataloader, model, loss_fn)
print("Done!")
```

Toy 2-layer MLP from official PyTorch tutorial
Parameters: <7e6
Gradient steps: 5 epochs = ~4e3

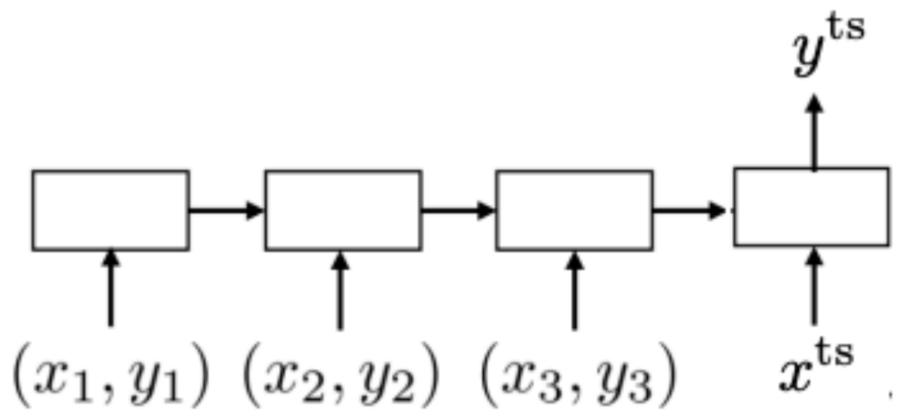
Total computation graph:
over 20 billion parameters!

From: https://pytorch.org/tutorials/beginner/basics/quickstart_tutorial.html

Direct Backpropagation

Black-box

$$y^{\text{ts}} = f_{\theta}(\mathcal{D}_i^{\text{tr}}, x^{\text{ts}})$$



Optimization-based

$$y^{\text{ts}} = f_{\text{MAML}}(\mathcal{D}_i^{\text{tr}}, x^{\text{ts}})$$
$$= f_{\phi_i}(x^{\text{ts}})$$
$$\text{where } \phi_i = \theta - \alpha \nabla_\theta \mathcal{L}(\theta, \mathcal{D}_i^{\text{tr}})$$

Non-parametric

$$y^{\text{ts}} = f_{\text{PN}}(\mathcal{D}_i^{\text{tr}}, x^{\text{ts}})$$
$$= \text{softmax}(-d\left(f_{\theta}(x^{\text{ts}}), \mathbf{c}_n\right))$$
$$\text{where } \mathbf{c}_n = \frac{1}{K} \sum_{(x,y)\in\mathcal{D}_i^{\text{tr}}} \mathbb{1}(y = n) f_\theta(x)$$

$$y^{\text{ts}} = f_{\text{LEARN}}\left(\mathcal{D}_i^{\text{tr}}, x^{\text{ts}}\ ; \theta\right)$$

Question: Describe a scenario where f_{LEARN} is too big to apply direct backpropagation.

(pollev.com/330)

Plan for Today

Why consider large-scale meta-optimization?

Applications

Methods
- Truncated backpropagation
- Gradient-free optimization

Settings With Bigger Computation Graphs

$$y^{\text{ts}} = f_{\text{LEARN}}\left(\mathcal{D}_i^{\text{tr}}, x^{\text{ts}}\ ; \theta\right)$$

Computation graph of f_{LEARN} is large when:

- It uses a big network and/or many gradient steps
- It includes second-order optimization (***meta-meta learning***?)

Meta-parameter θ can be any component of f_{LEARN}:

HW2
- Initial parameters
- Learning rate
- Optimizer
- Model architecture
- Loss function / regularizer
- Dataset / augmentation

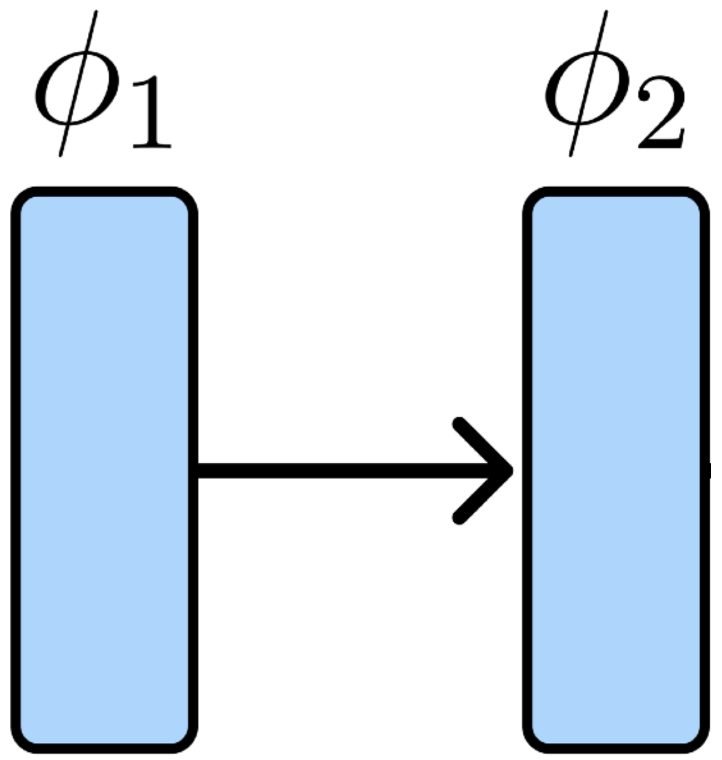
$$\min_{\theta} \sum_{\text{task}_i} \left(\theta - \alpha \nabla_\theta L\left(\theta, D_i^{\text{tr}}\right), D_i^{\text{ts}}\right)$$

$$\min_{\theta, \alpha} \sum_{\text{task}_i} \left(\theta - \alpha \nabla_\theta L\left(\theta, D_i^{\text{tr}}\right), D_i^{\text{ts}}\right)$$

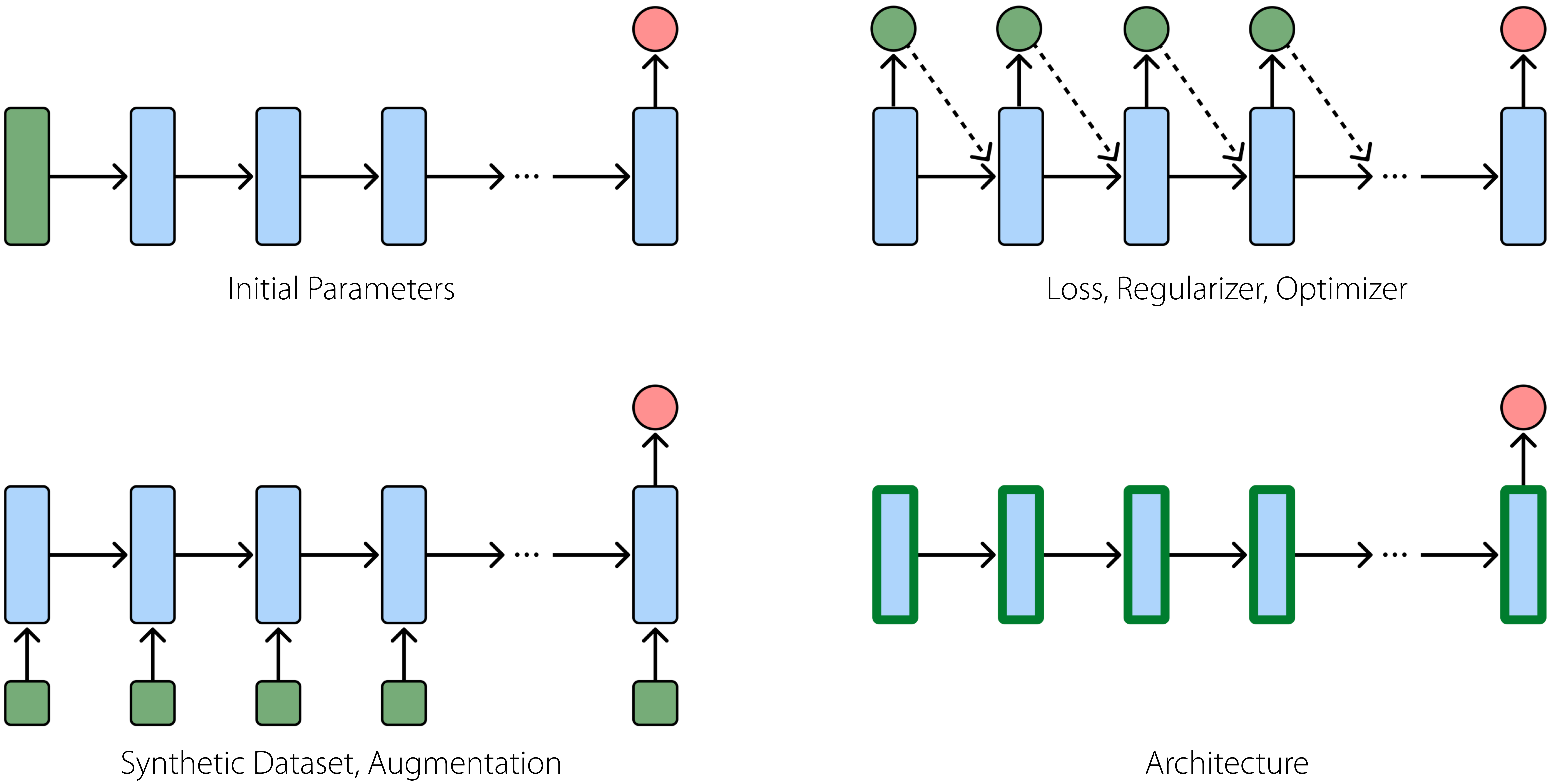
$$\min_{\theta, \psi} \sum_{\text{task}_i} \left(\theta - \alpha \nabla_\theta L_\psi\left(\theta, D_i^{\text{tr}}\right), D_i^{\text{ts}}\right)$$

$$\min_{\omega} \sum_{\theta \sim p(\theta_0)} \left(\theta - \alpha \nabla_\theta L\left(\theta, D_\omega\right), D_i^{\text{ts}}\right)$$

Unrolled Computation Graphs

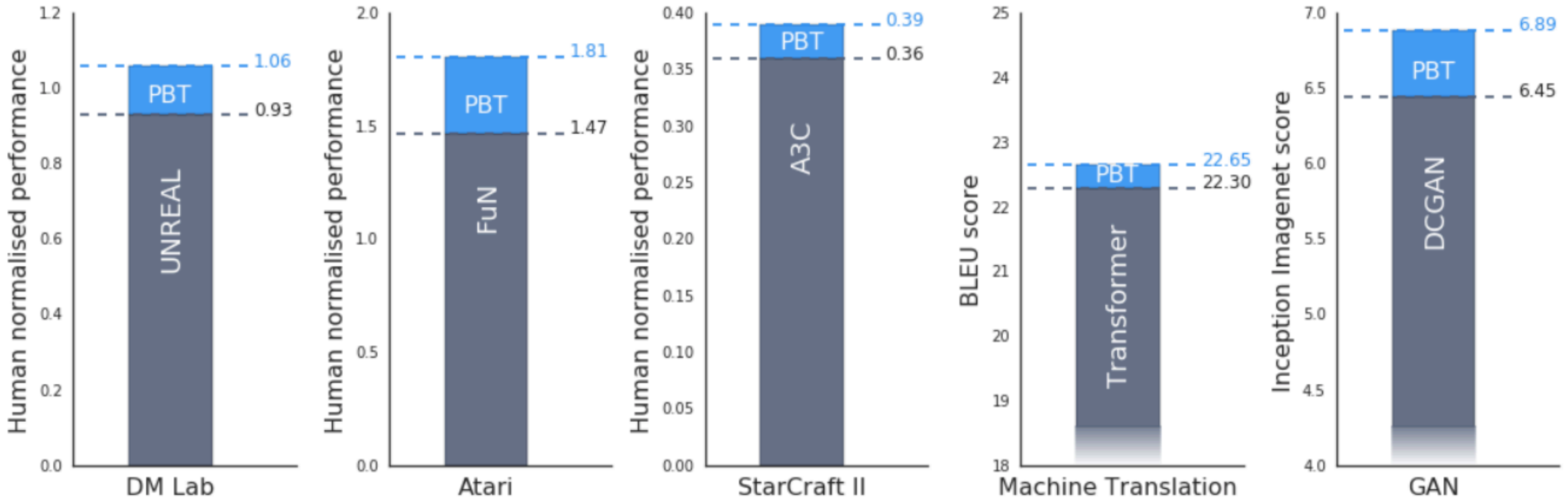


Unrolled Computation Graphs



Application: Hyperparameter Optimization

Goal: Optimize ***hyperparameters*** for validation set performance



Bene?ts over random search in many domains

Method	Validation	Test	Time(s)
Grid Search	97.32	94.58	100k
Random Search	84.81	81.46	100k
Bayesian Opt.	72.13	69.29	100k
STN	70.30	67.68	25k
No HO	75.72	71.91	18.5k
Ours	69.22	66.40	18.5k
Ours, Many	**68.18**	**66.14**	18.5k

LSTM Hyperparameters

Inverse Approx.	Validation	Test
0	92.5 ±0.021	92.6 ±0.017
3 Neumann	**95.1** ±0.002	94.6 ±0.001
3 Unrolled Diff.	95.0 ±0.002	**94.7** ±0.001
I	94.6 ±0.002	94.1 ±0.002

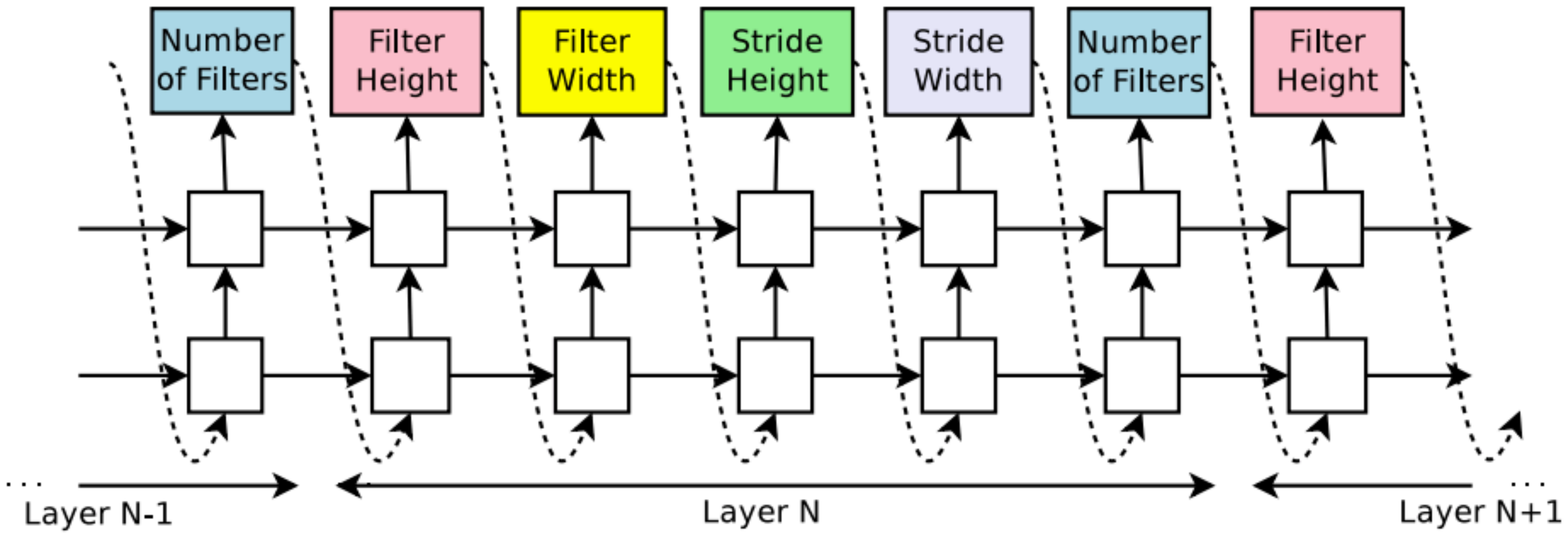
"Hyper"parameters of a data augmentation network

From: "Population Based Training of Neural Networks", Jaderberg et al. (2017)

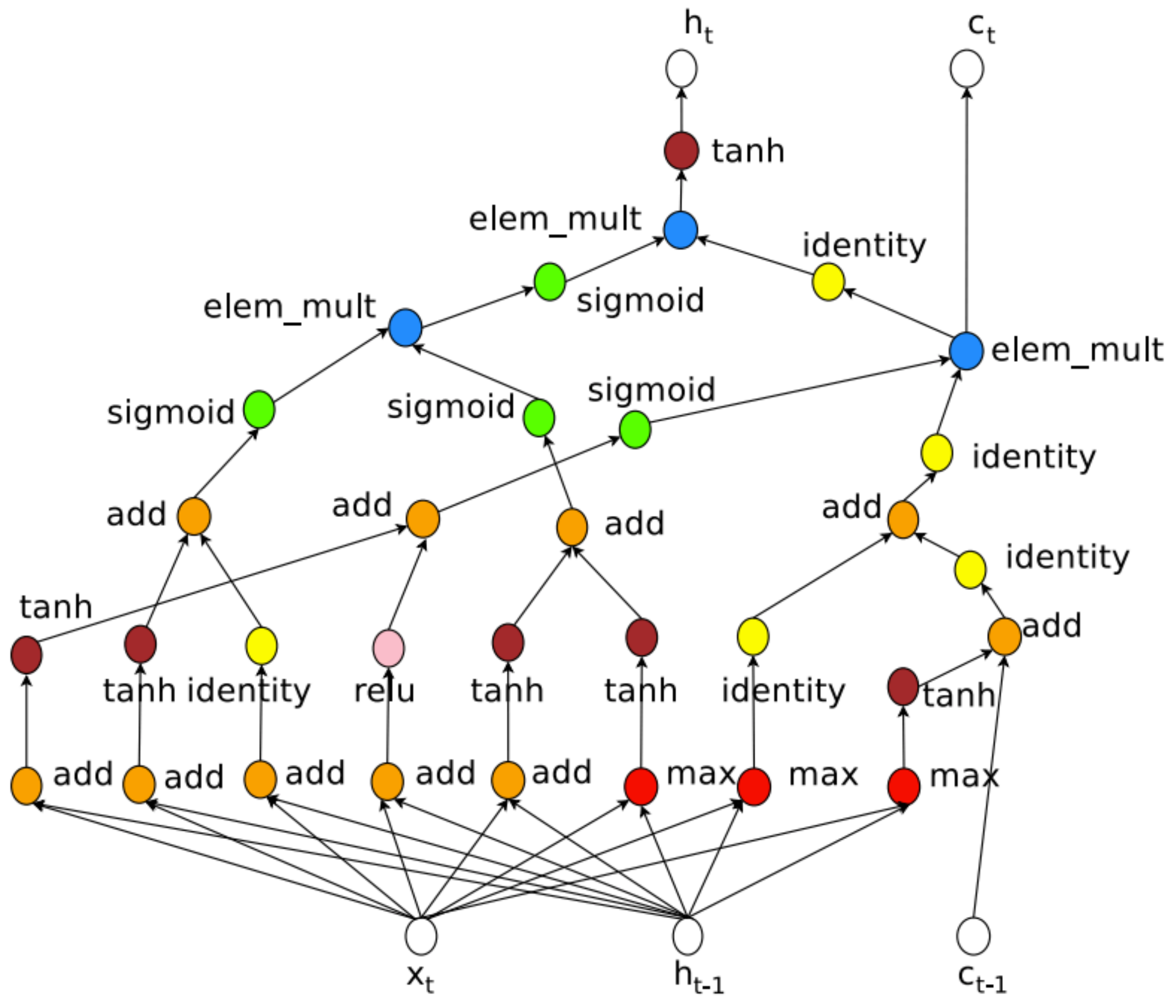
From: "Optimizing Millions of Hyperparameters by Implicit Differentiation", Lorraine et al. (2019)

Application: Architecture Search

Goal: Optimize an ***architecture*** for validation set performance



An RNN parameterizes a neural network



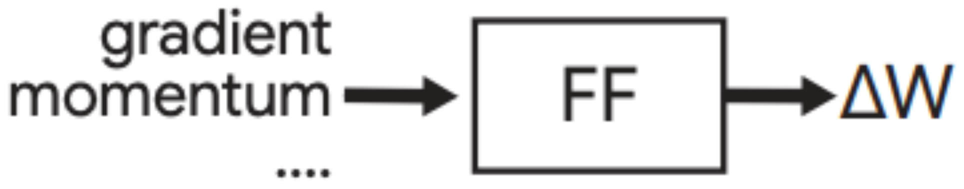
A generated cell for an RNN

Model	Depth	Parameters	Error rate (%)
Network in Network (Lin et al., 2013)	-	-	8.81
All-CNN (Springenberg et al., 2014)	-	-	7.25
Deeply Supervised Net (Lee et al., 2015)	-	-	7.97
Highway Network (Srivastava et al., 2015)	-	-	7.72
Scalable Bayesian Optimization (Snoek et al., 2015)	-	-	6.37
FractalNet (Larsson et al., 2016)	21	38.6M	5.22
with Dropout/Drop-path	21	38.6M	4.60
ResNet (He et al., 2016a)	110	1.7M	6.61
ResNet (reported by Huang et al. (2016c))	110	1.7M	6.41
ResNet with Stochastic Depth (Huang et al., 2016c)	110	1.7M	5.23
	1202	10.2M	4.91
Wide ResNet (Zagoruyko & Komodakis, 2016)	16	11.0M	4.81
	28	36.5M	4.17
ResNet (pre-activation) (He et al., 2016b)	164	1.7M	5.46
	1001	10.2M	4.62
DenseNet ($L = 40, k = 12$) Huang et al. (2016a)	40	1.0M	5.24
DenseNet($L = 100, k = 12$) Huang et al. (2016a)	100	7.0M	4.10
DenseNet ($L = 100, k = 24$) Huang et al. (2016a)	100	27.2M	3.74
DenseNet-BC ($L = 100, k = 40$) Huang et al. (2016b)	190	25.6M	3.46
Neural Architecture Search v1 no stride or pooling	15	4.2M	5.50
Neural Architecture Search v2 predicting strides	20	2.5M	6.01
Neural Architecture Search v3 max pooling	39	7.1M	4.47
Neural Architecture Search v3 max pooling + more filters	39	37.4M	3.65

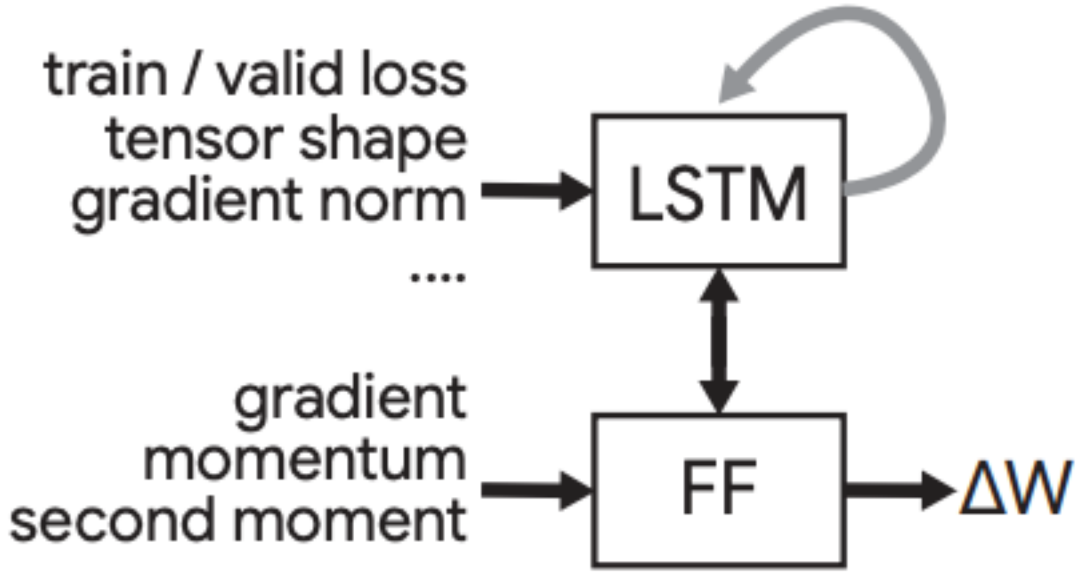
Zoph and Le, "Neural Architecture Search with Reinforcement Learning" (2017)

Application: Optimizer Learning

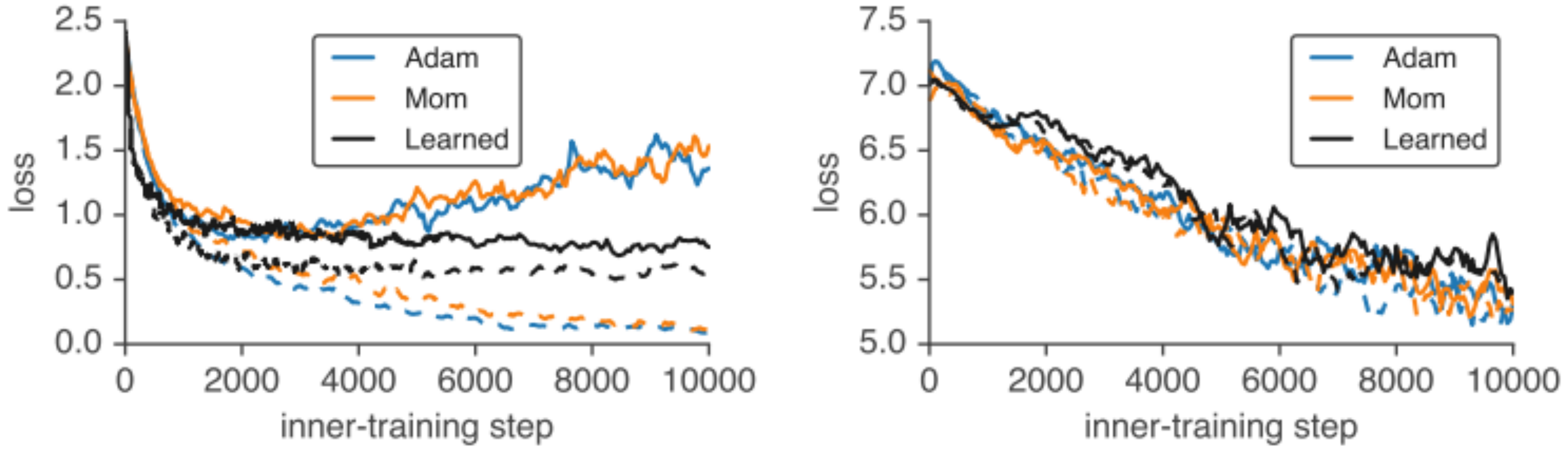
Goal: Optimize an ***optimizer*** for validation set performance



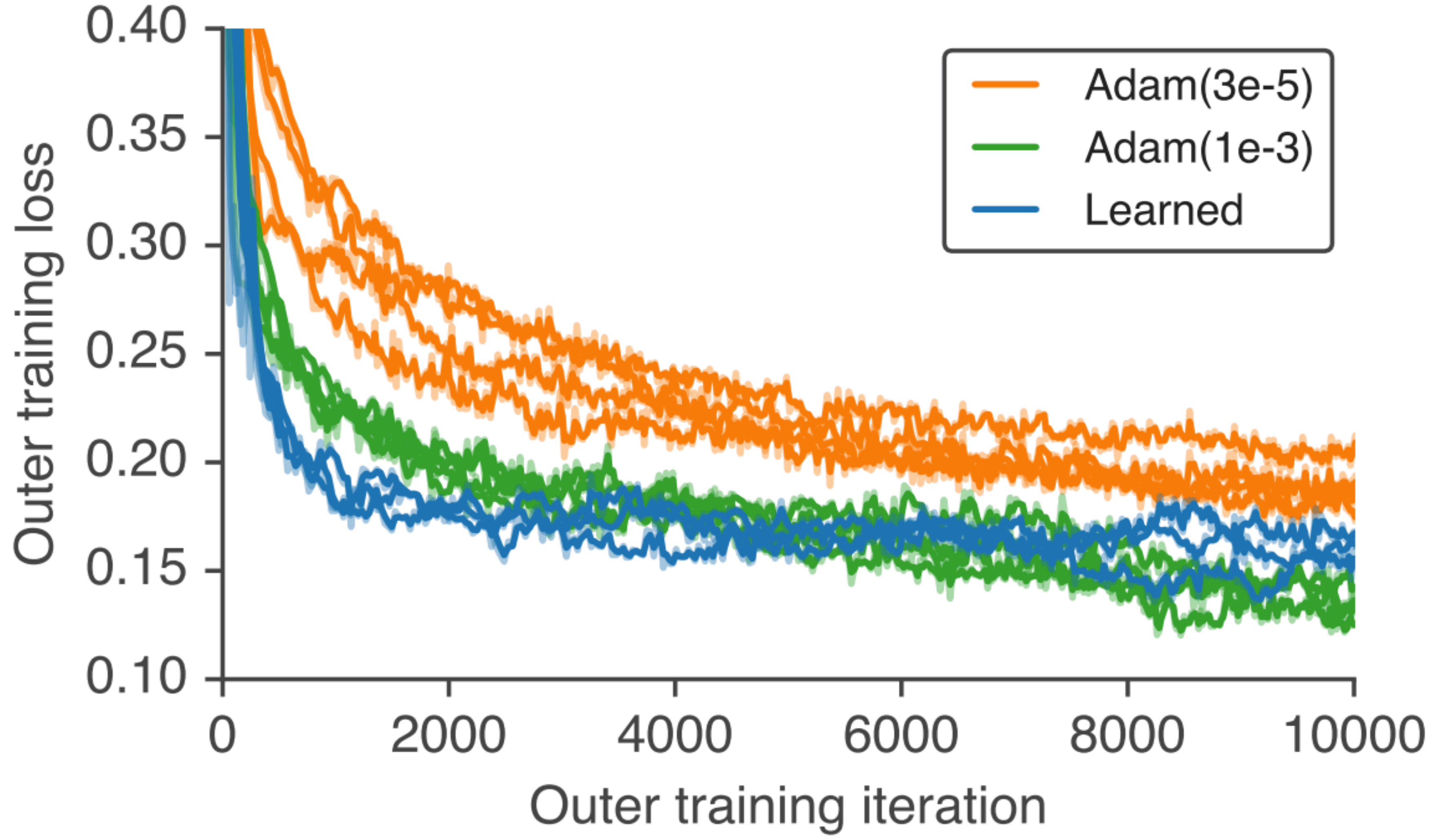
Simple learned optimizer



More complex optimizer



Works in long training loops

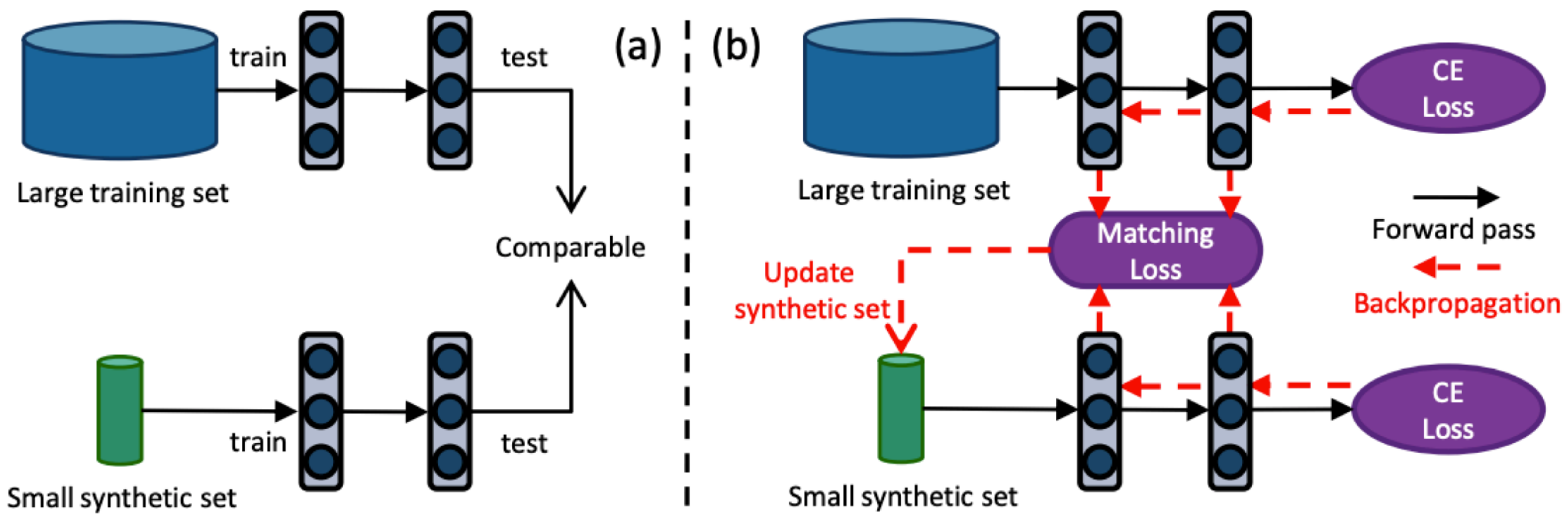


Can even train itself

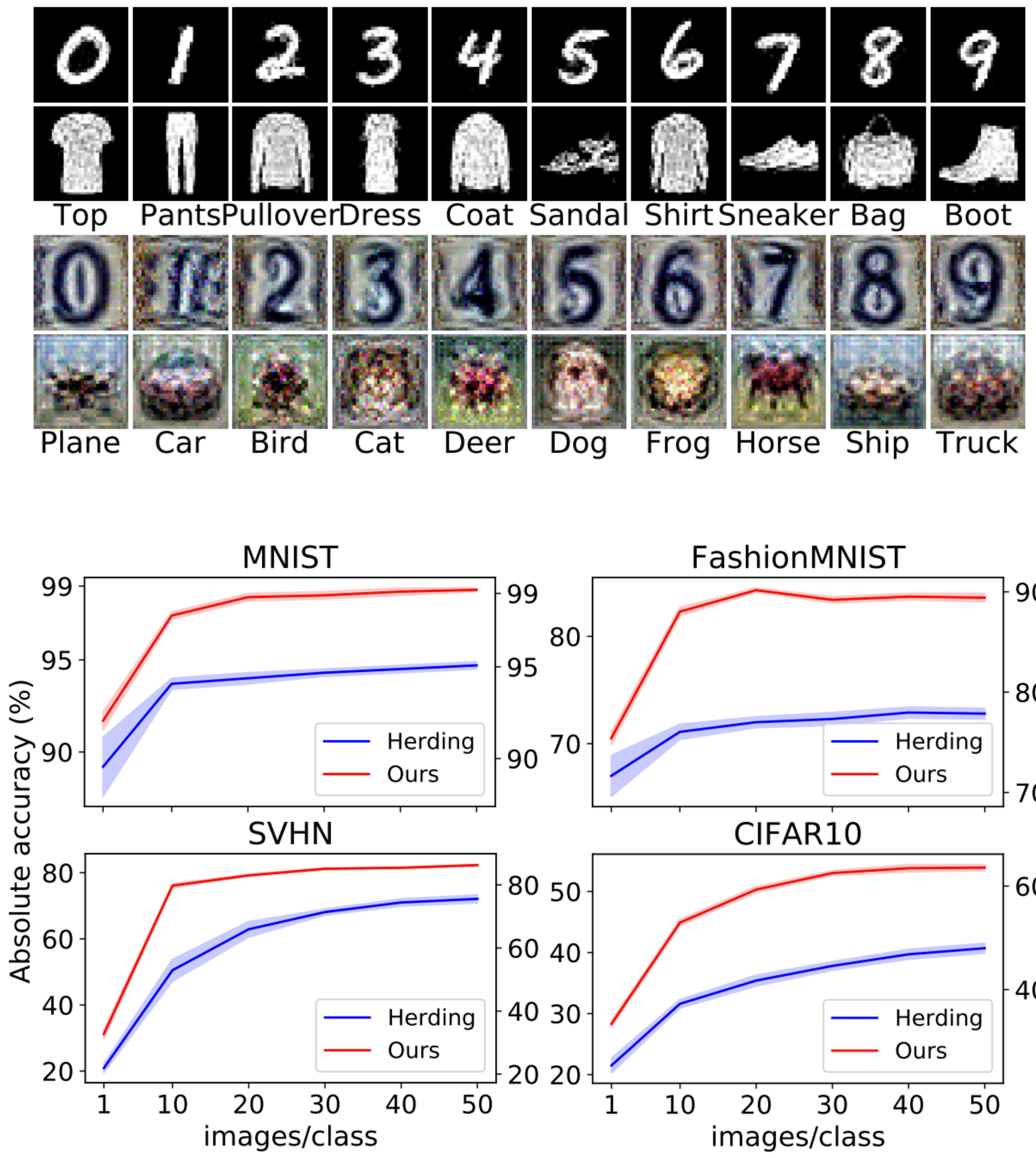
From: "Tasks, stability, architecture, and compute: Training more effective learned optimizers, and using them to train themselves", Metz et al. (2020)

Application: Dataset Distillation

Goal: optimize a ***synthetic training set*** for validation set performance



Method: Match training data gradients at each timestep



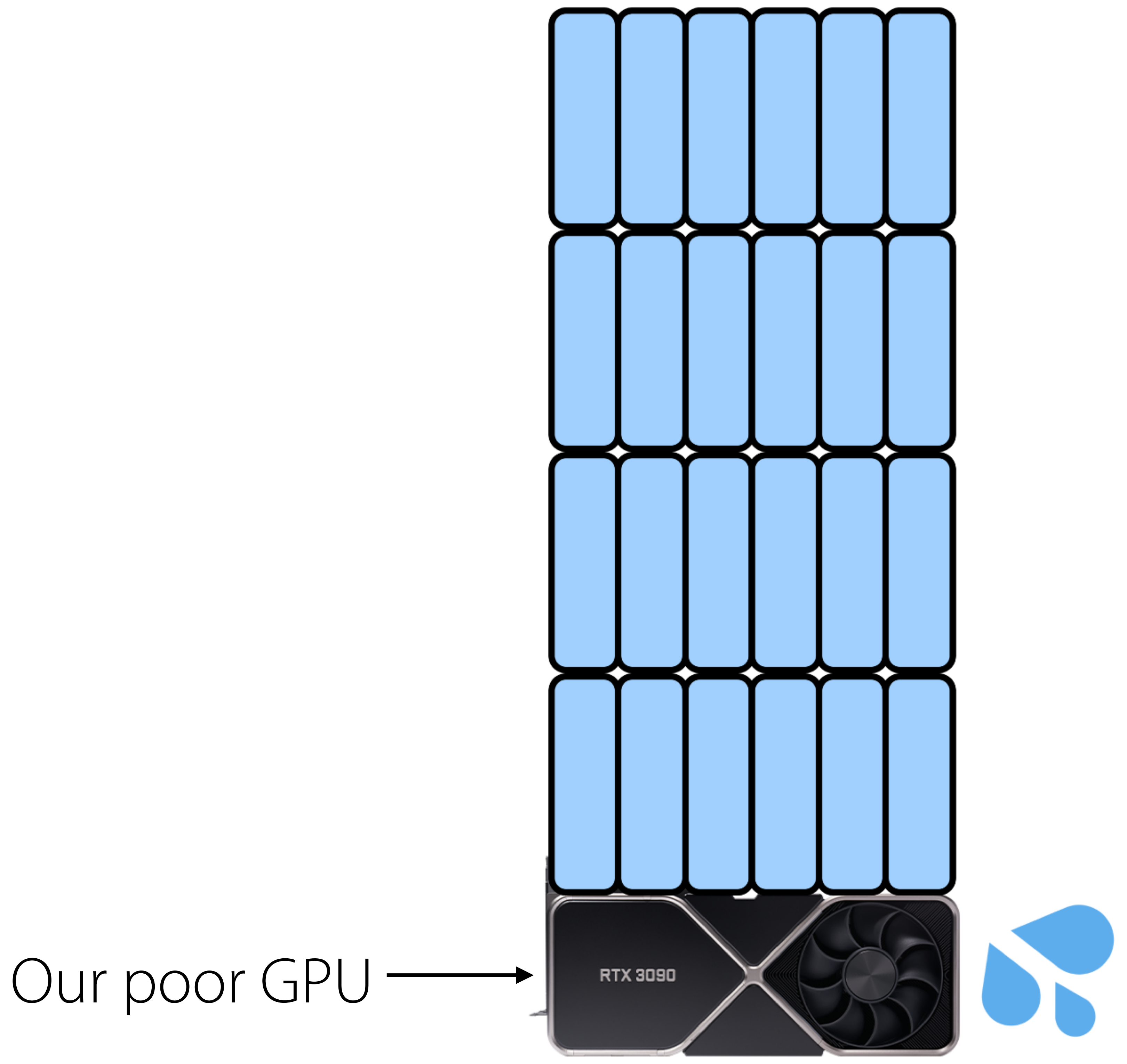
From: "Dataset Condensation with Gradient Matching", Zhao et al. (2020)

Plan for Today

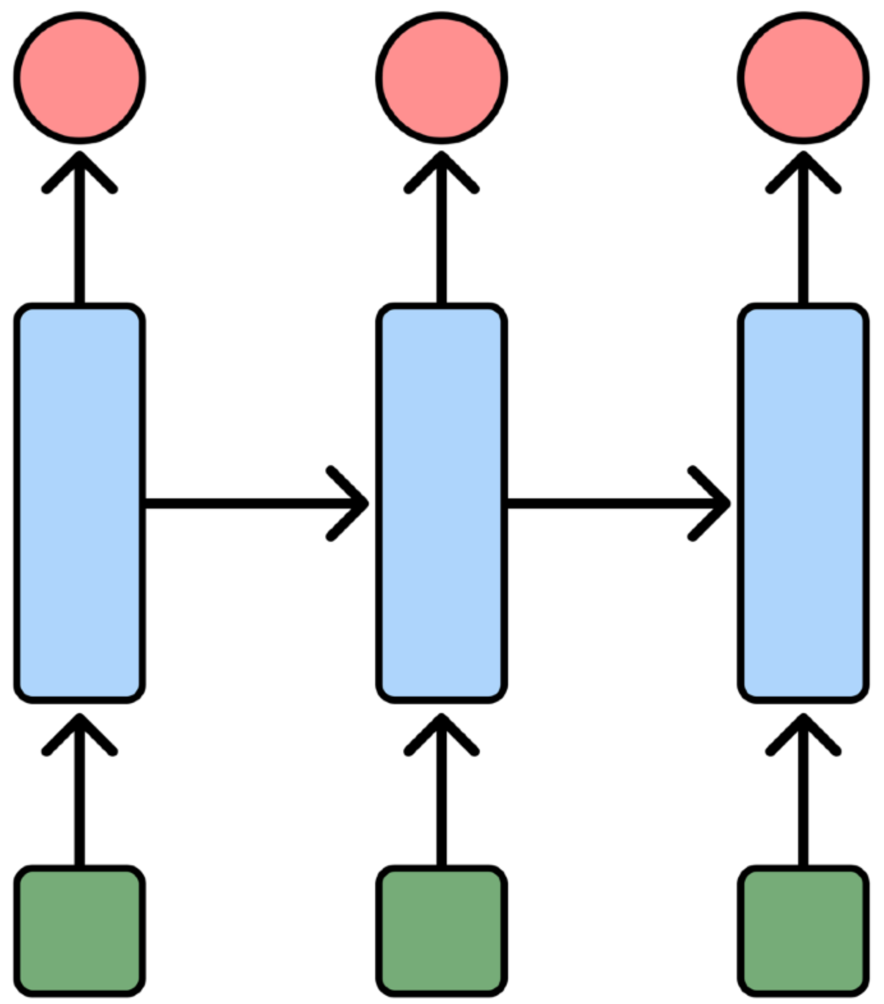
Why consider large-scale meta-optimization?

Applications

Approaches
- **Truncated backpropagation**
- Gradient-free optimization

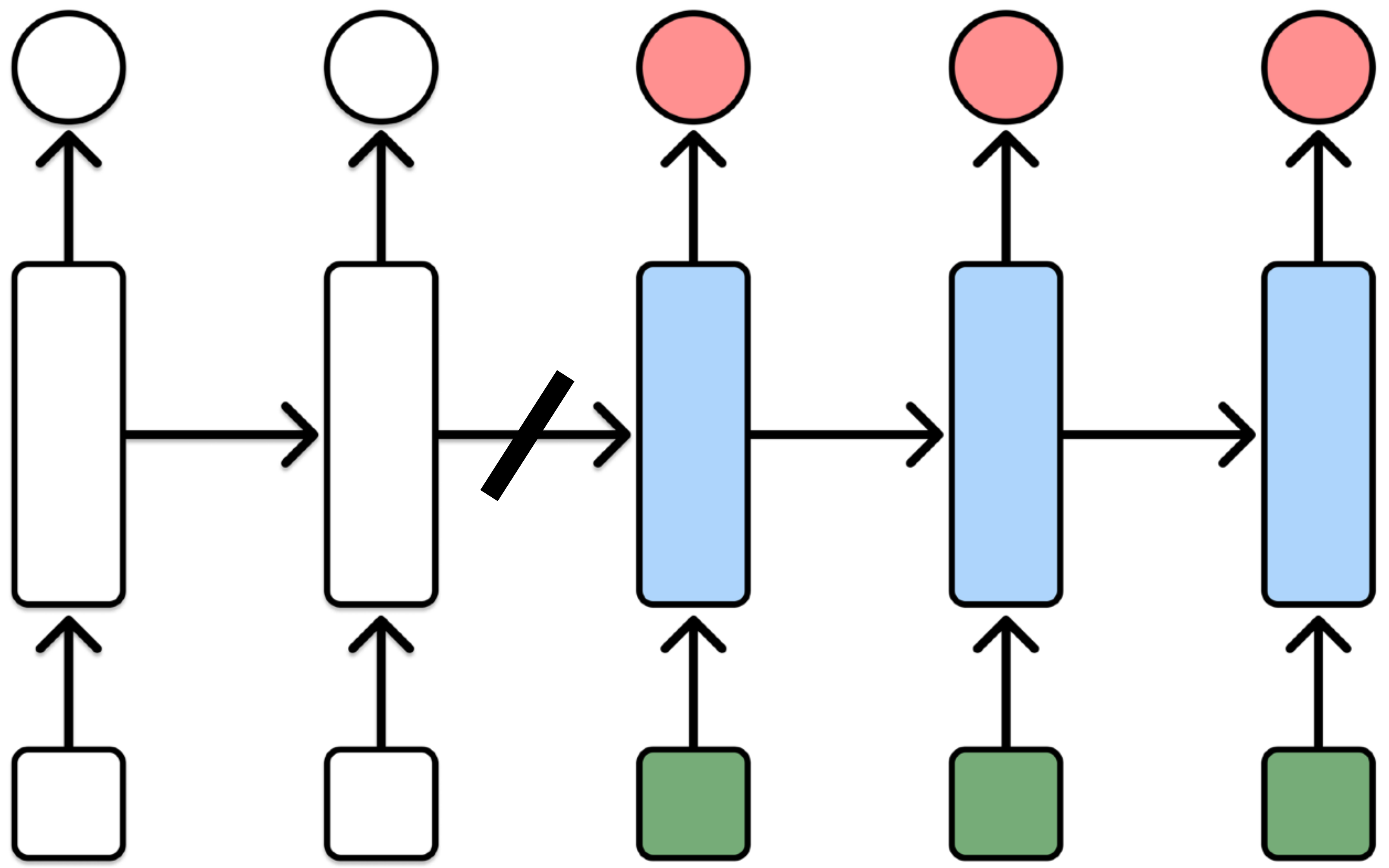


Truncated Backpropagation



T=3

Truncated Backpropagation



T=3

```python
losses = []
for x, y in train_loader:
    y_pred, state = rnn(state)
    losses.append(loss_fn(y, y_pred))
    if len(losses) == T:
        torch.sum(losses).backward()
        opt.step()
        opt.zero_grad()
        state.detach_()
        losses = []
```

Split the full sequence into shorter slices, and backpropagate after processing each slice.

Question: what could happen if we use short T?

+ **Simple**: autograd handles everything
- **Biased** estimator
- Cannot take **long-range dependencies** into account
- Sequence length introduces a **tradeoff between correctness and memory cost**

Plan for Today

Why consider large-scale meta-optimization?

Applications

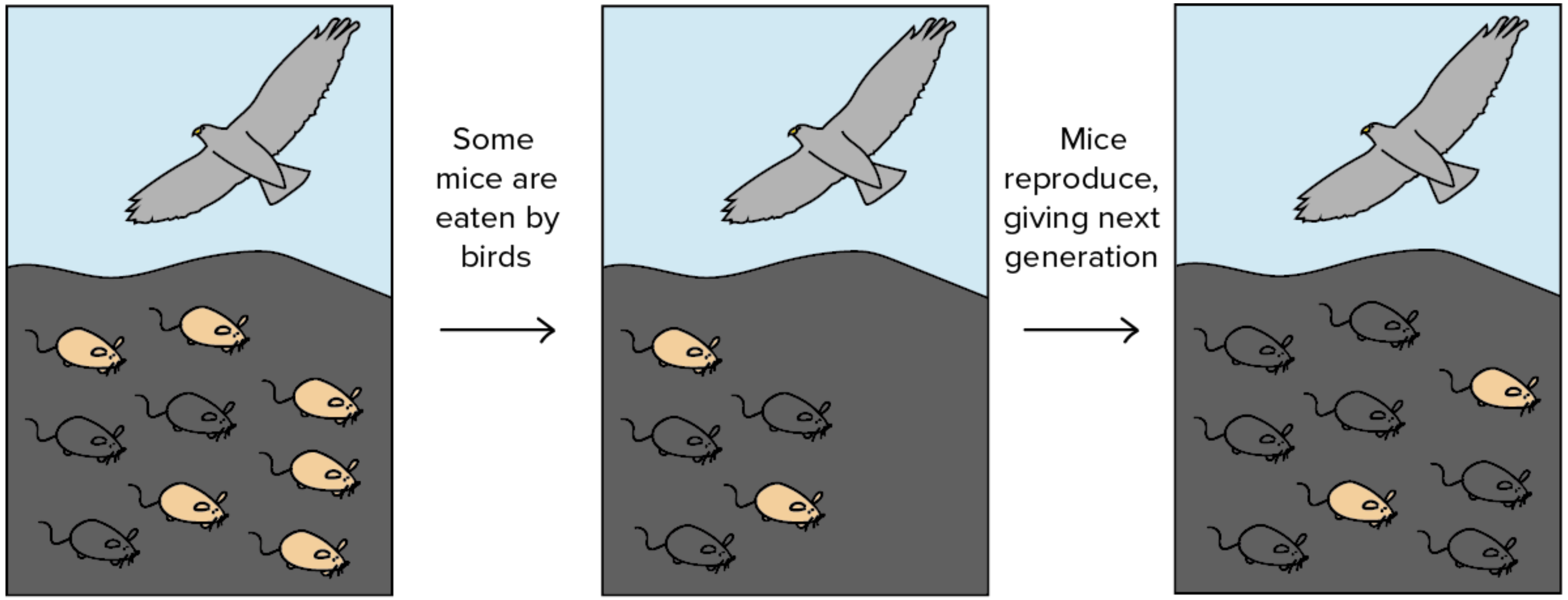
Methods
- Truncated backpropagation
- **Gradient-free optimization**

Gradient-free Optimization

Backpropagation is costly for large computation graphs…

Optimization **does not necessarily require gradients!**

Evolution Strategies: Estimates gradients using stochastic finite differences.

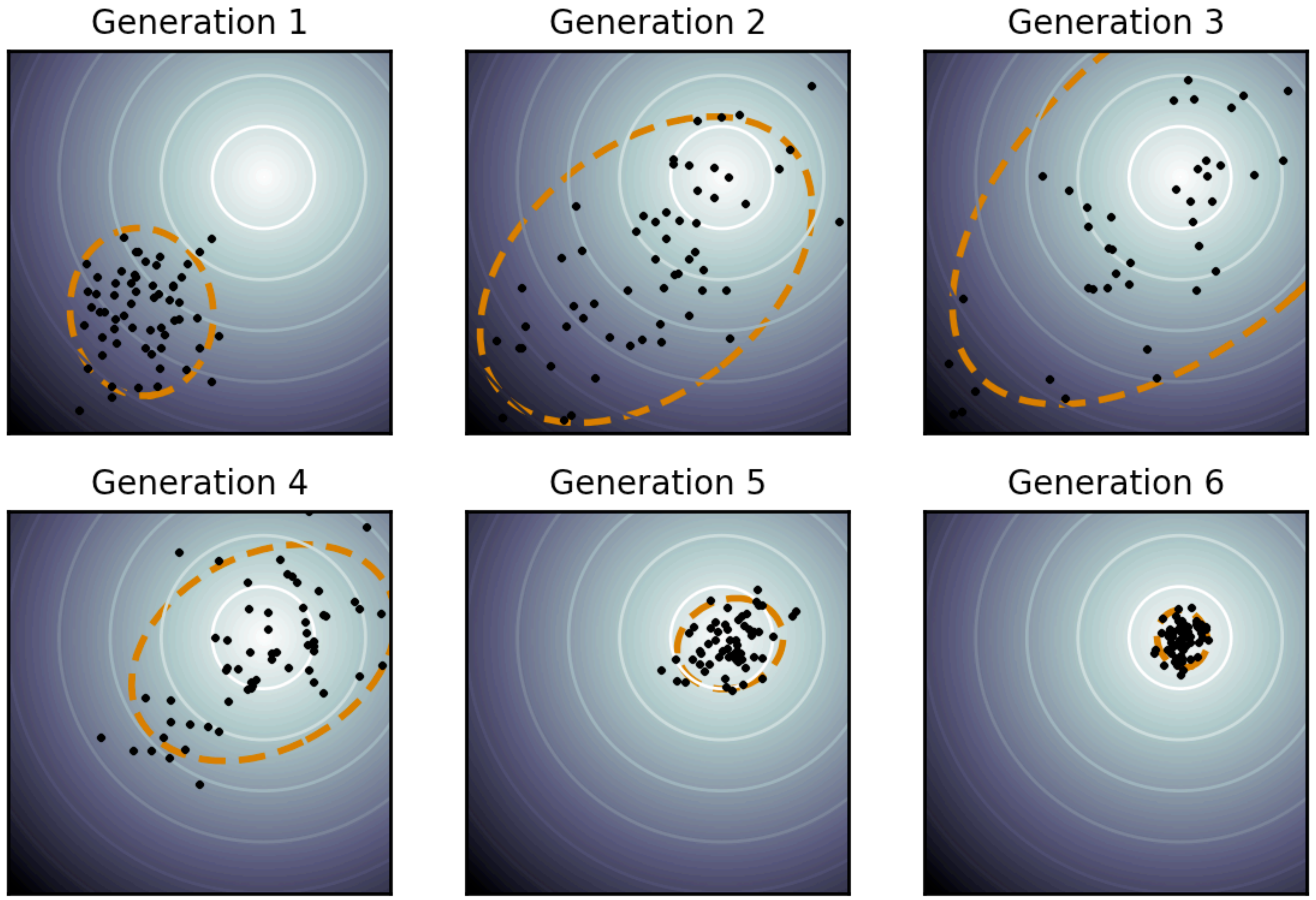


From: Khan Academy: Darwin, evolution, & natural selection

Evolution Strategies

Initialize parameters $(\mu, \sigma) \leftarrow (\mu_0, \sigma_0)$. Repeat:

1. Sample particles: $x^1, x^2, \ldots, x^N \sim \mathcal{N}(\mu, \sigma^2 I)$
2. Evaluate and get best: $\{e^1, \ldots, e^n\} \subset \{x^1, \ldots, x^N\}$
3. $\mu, \sigma^2 \leftarrow \mathrm{Avg}(e^1, \ldots, e^n), \mathrm{Var}(e^1, \ldots, e^n)$



From: Wikipedia CMA-ES page

Example: optimizing **learning rate**

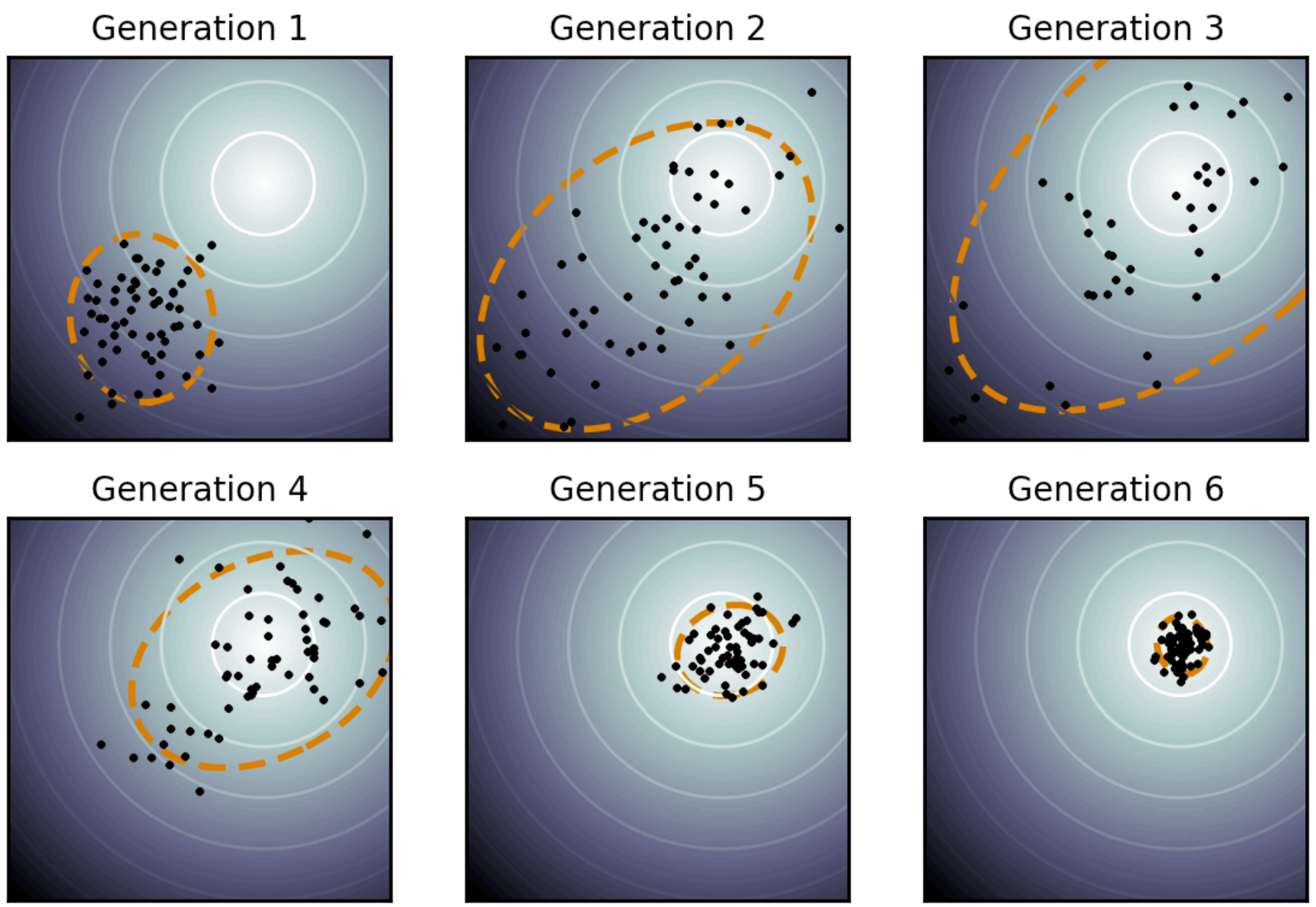
Initialize lr and noise $(\alpha, \sigma) \leftarrow (\alpha_0, \sigma_0)$

1. Sample lr: $\alpha^1, \alpha^2, \ldots, \alpha^N \sim \mathcal{N}(\alpha, \sigma^2)$
2. Run SGD, get runs with best val accuracy:
   $\{e^1, \ldots, e^n\} \subset \{\alpha^1, \ldots, \alpha^N\}$
3. $\alpha, \sigma^2 \leftarrow \mathrm{Avg}(e^1, \ldots, e^n), \mathrm{Var}(e^1, \ldots, e^n)$

Evolution Strategies

Initialize parameters $(\mu, \sigma) \leftarrow (\mu_0, \sigma_0)$. Repeat:

1. Sample particles: $x^1, x^2, \ldots, x^N \sim \mathcal{N}(\mu, \sigma^2 I)$
2. Evaluate and get best: $\{e^1, \ldots, e^n\} \subset \{x^1, \ldots, x^N\}$
3. $\mu, \sigma^2 \leftarrow \mathrm{Avg}(e^1, \ldots, e^n), \mathrm{Var}(e^1, \ldots, e^n)$

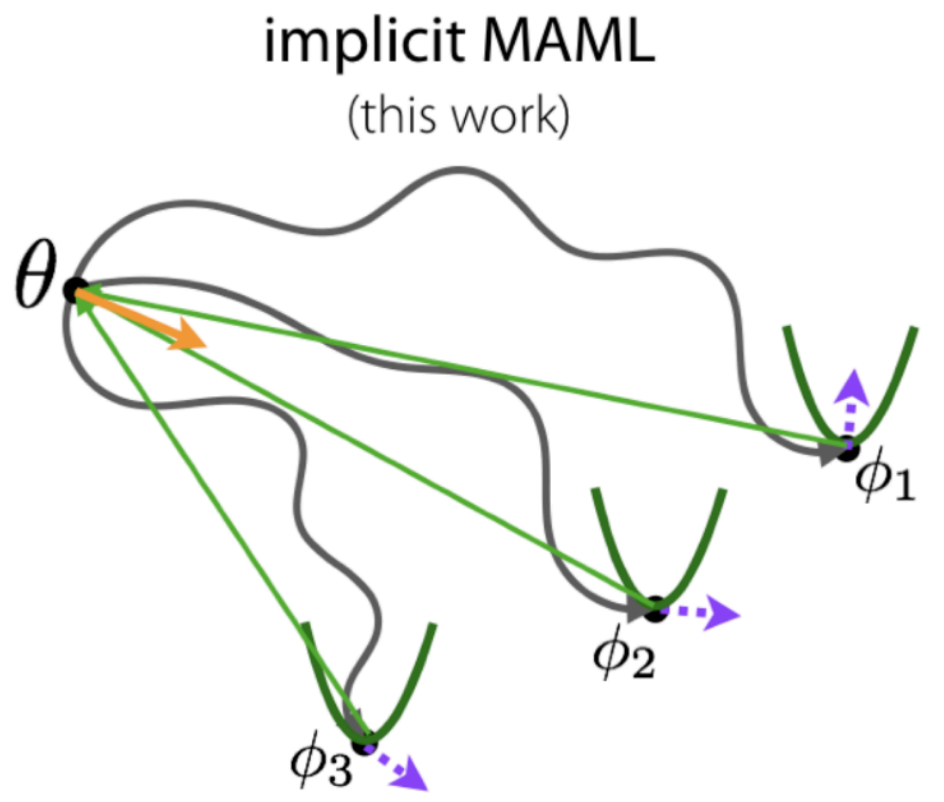


From: Wikipedia CMA-ES page

+ **Constant memory cost**
+ **Parallelizable** across particles
+ Inner steps can be non-differentiable
- Struggles with **high-dimensional covariates** and/or **complex loss surfaces**

Other Methods for Large-Scale Meta-Optimization

Implicit Differentiation



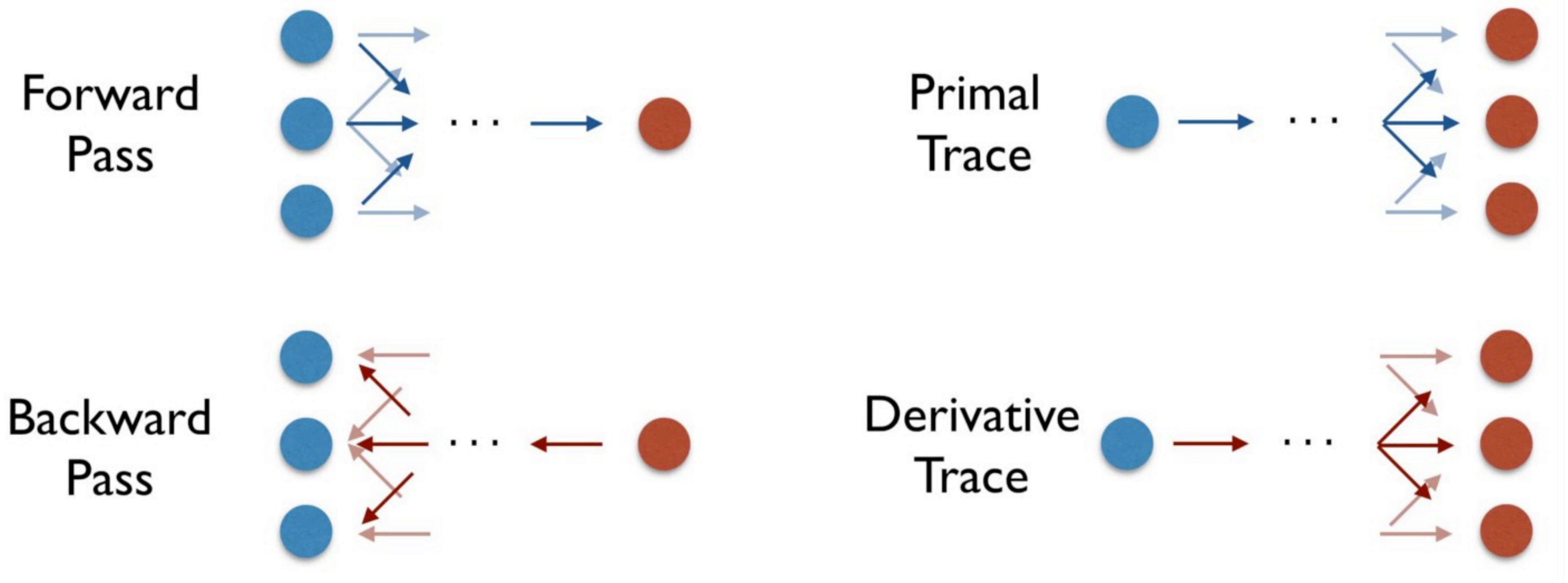
Finds the derivative of a function that is not explicitly defined.

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \, \frac{1}{M} \sum_{i=1}^{M} \frac{d\mathcal{A}lg_i^\star(\boldsymbol{\theta})}{d\boldsymbol{\theta}} \nabla_\phi \mathcal{L}_i(\mathcal{A}lg_i^\star(\boldsymbol{\theta})).$$

$$\frac{d\mathcal{A}lg_i^\star(\boldsymbol{\theta})}{d\boldsymbol{\theta}} = \left(\boldsymbol{I} + \frac{1}{\lambda} \nabla_{\boldsymbol{\phi}}^2 \hat{\mathcal{L}}_i(\boldsymbol{\phi}_i) \right)^{-1}$$

From: "Meta-Learning with Implicit Gradients", Rajeswaran et al. (2019)

Forward-mode Differentiation



Computes a derivative by propagating derivatives from the inputs to the outputs, following the chain rule.

From: "Forward Mode Automatic Differentiation & Dual Numbers", Lange

Plan for Today

Why consider large-scale meta-optimization?

Applications

Methods
- Truncated backpropagation
- Gradient-free optimization

Goals for by the end of lecture:
- Know scenarios where existing meta-learning approaches fail due to scale
- Understand techniques for large-scale meta-optimization

Next Time

Wed: Guest lecture on learned optimizers!

Next week: Domain adaptation & lifelong learning

Following week: 🦃

Course Reminders

HW4 is due **next Wednesday** (optional)