

Advanced Meta-Learning 2: Large-Scale Meta-Optimization

CS 330

Reminder

HW3 out this Monday, due **next Wednesday**.

Do meta-learning methods scale?

Hand-designed priors

Modeling image formation

SIFT features

Fine-tuning from ImageNet

Meta-learning

↳ End-to-end learning

Data-driven priors



More data-driven approaches are supposed to be more scalable

Do meta-learning methods work at scale?

↳ Do meta learners like MAML and Prototypical Networks work better w/ more data?
⇒ Not really.

↳ more data → better at getting real priors

Plan for Today

Why consider *large-scale meta-optimization?*

Applications

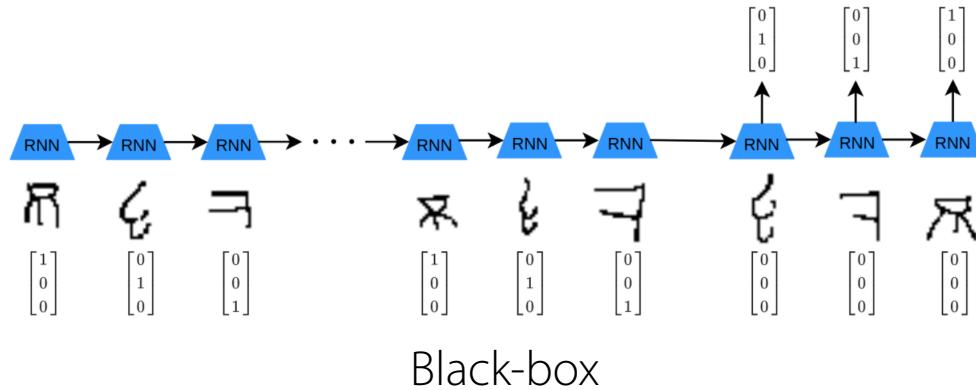
Approaches

- Truncated backpropagation
- Gradient-free optimization

Goals for by the end of lecture:

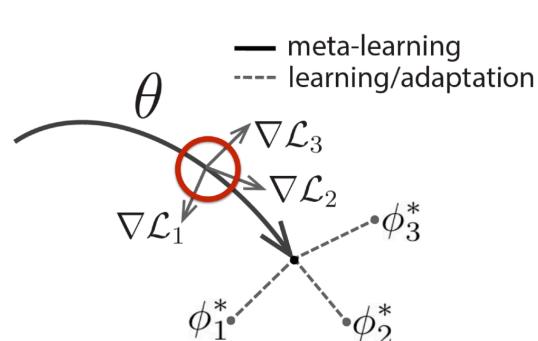
- Know scenarios where **existing meta-learning approaches fail** due to scale
- Understand techniques for **large-scale meta-optimization**

Direct Backpropagation

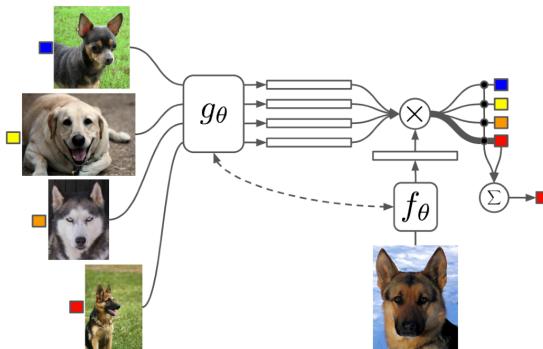


Black-box

General recipe: Build an inner computation graph, then backpropagate.



Optimization-based



Nonparametric

+ Automatically works with any differentiable computation graph
- **Memory cost** scales with **computation graph size!**

↳ Main challenge in large-scale meta optimization

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

4-layer CNN

Parameters: <1e5

WRN-28-10

Parameters: <4e6

ResNet-12

Parameters: <1e7

How Big are Computation Graphs?

```
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

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 = $\sim 4e3$

\hookrightarrow No. of steps

Total floats: $\sim 2e10 (> 100\text{GB}!)$

\hookrightarrow Too large to apply direct backpropagation.

?? Is this the case? ↵

Black-box

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

```
graph LR; x1["(x1, y1)"] --> L1[ ]; x2["(x2, y2)"] --> L1; x3["(x3, y3)"] --> L1; L1 --> L2[ ]; L2 --> L3[ ]; L3 --> yts["y^ts"];
```

Optimization-based

$$\begin{aligned} y^{\text{ts}} &= f_{\text{MAML}}(\mathcal{D}_i^{\text{tr}}, x^{\text{ts}}) \\ &= f_{\phi_i}(x^{\text{ts}}) \end{aligned}$$

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

Non-parametric

$$\begin{aligned} y^{\text{ts}} &= f_{\text{PN}}(\mathcal{D}_i^{\text{tr}}, x^{\text{ts}}) \\ &= \text{softmax}(-d(f_{\theta}(x^{\text{ts}}), \mathbf{c}_n)) \end{aligned}$$

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

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

Question: when might f_{LEARN} be too big to apply direct backpropagation?

- Model too big for GPU

Settings With Bigger Computation Graphs

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

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?*)

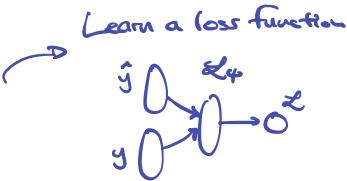
Because the whole computation graph
is (size) \times (no. of gradient steps)

↳ 이해 잘 안갈

↳ If the inner loop itself is meta-learning.

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

- HW2 |
- Initial parameters
 - Learning rate
 - Optimizer
 - Loss function
 - Dataset
 - Network architecture



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

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

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

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

Q: When would you want to learn a loss function?

A: We need a set loss func. in the outer loop.

Usually the loss func. we use is a proxy to sth we actually want to calc. cross-entropy instead of accuracy b/c. acc. is not back-propagatable.

But it may not be the case that CE is the best thing for us to optimize over to get high acc.

→ another ex) → learning an appropriate regularizer.

Plan for Today

Why consider large-scale meta-optimization?

Applications

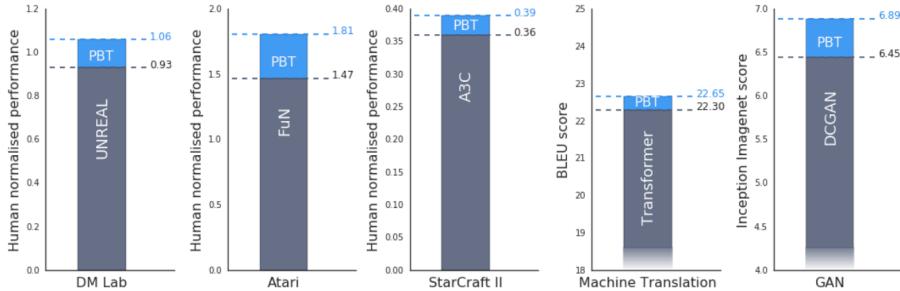
Approaches

- Truncated backpropagation
- Gradient-free optimization

Application: Hyperparameter Optimization

→ People usually don't view this as meta-learning, but it's possible.

Goal: Optimize *hyperparameters* for validation set performance



Benefits 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

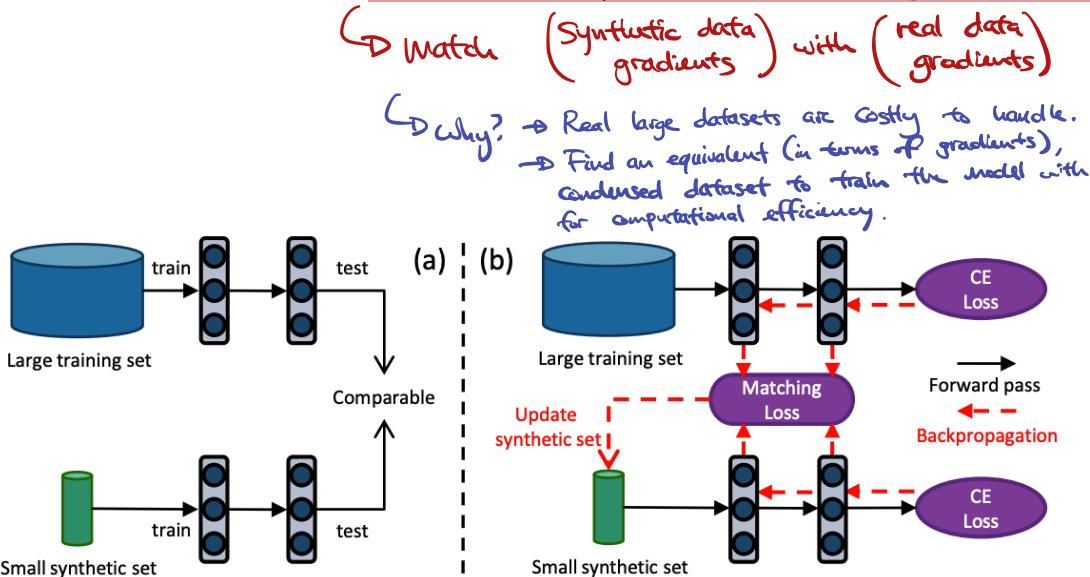
LSTM Hyperparameters

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

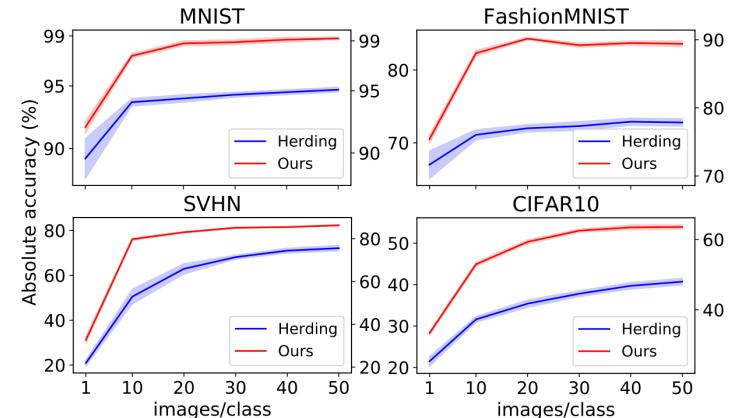
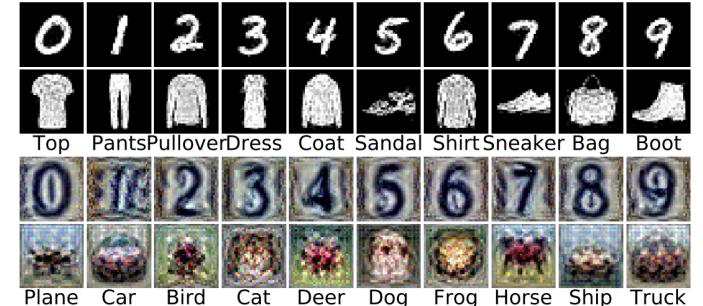
"Hyper"parameters of a data augmentation network

Application: Dataset Distillation

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



Method: Match training data gradients at each timestep



Q: How does it know how to
optimize in a new loss
landscape (that it's never
seen before)?

Application: Optimizer Learning

→ Parameterize an optimizer as a neural network.

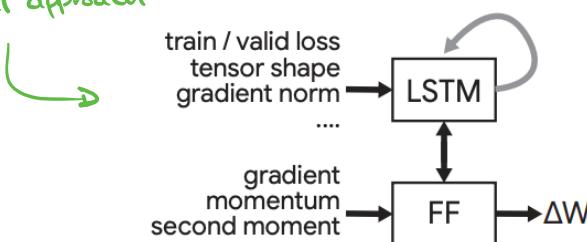
A: Once the inner loop is updated,
the optimization (of the target model)
is reset, ~ it begins from scratch again.

→ verify this; TA is unclear in his response.

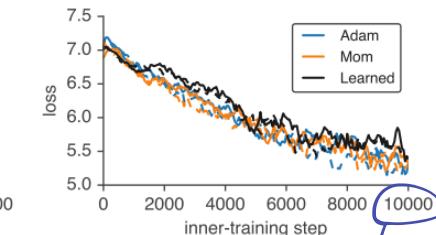
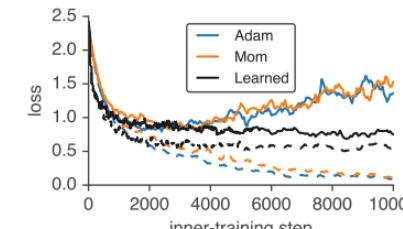


Simple architecture

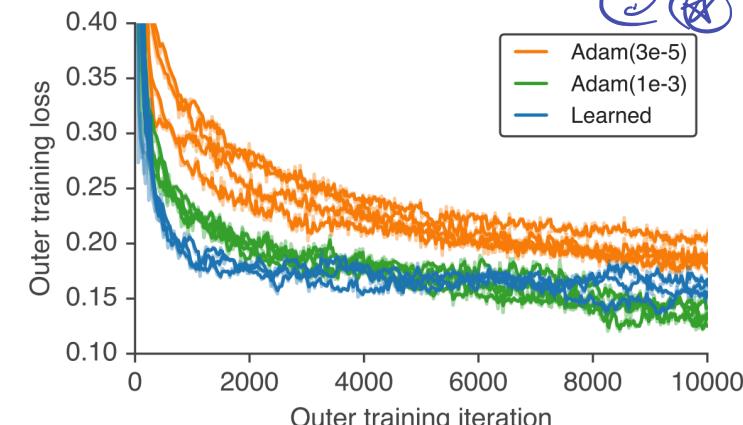
The optimizer takes in
a lot more info than
traditional approach



More complex architecture
with per-tensor LSTM



Works at ResNet scale



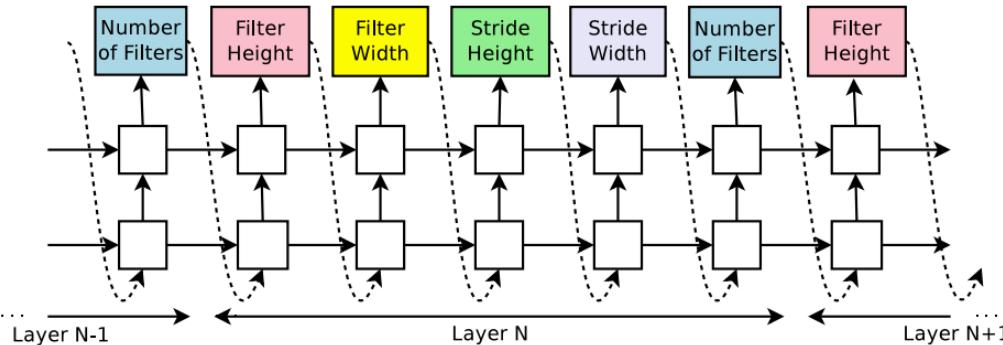
Can even train itself

Too big to apply direct backpropagation.
⇒ ??? → But they just did...

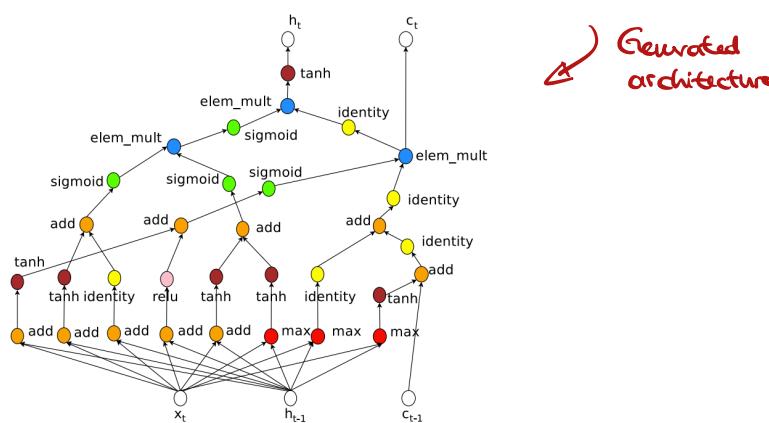


Application: Neural Architecture Search

Goal: Optimize an *architecture* for validation set performance



An RNN parameterizes a neural network



A generated cell for an RNN

↳ Example of a large-scale meta optimization.

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) with Dropout/Drop-path	21 21	38.6M 38.6M	5.22 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 1202	1.7M 10.2M	5.23 4.91
Wide ResNet (Zagoruyko & Komodakis, 2016)	16 28	11.0M 36.5M	4.81 4.17
ResNet (pre-activation) (He et al., 2016b)	164 1001	1.7M 10.2M	5.46 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

↳ Pretty old, but seminal

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

Plan for Today

Why consider large-scale meta-optimization?

Applications

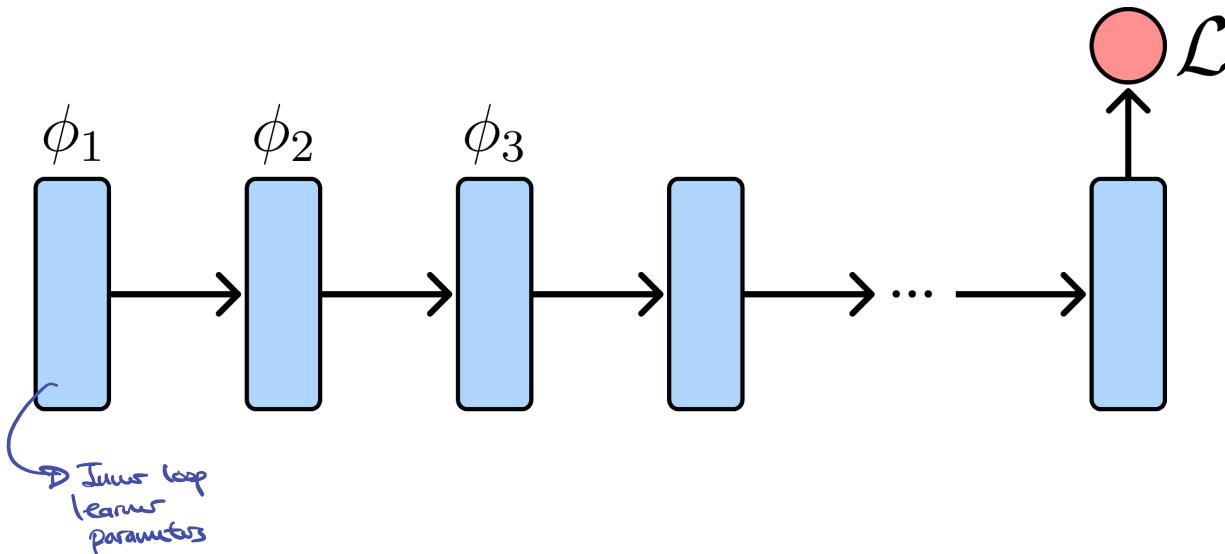
Approaches

- Truncated backpropagation
- Gradient-free optimization

Unrolled Computation Graphs

When you optimize some set of params continuously, we can unroll that computation graph into a sequence of parameters.

Goal: Modify ϕ s about this inner-loop learning system so that the red node's value becomes small. \rightarrow the first?

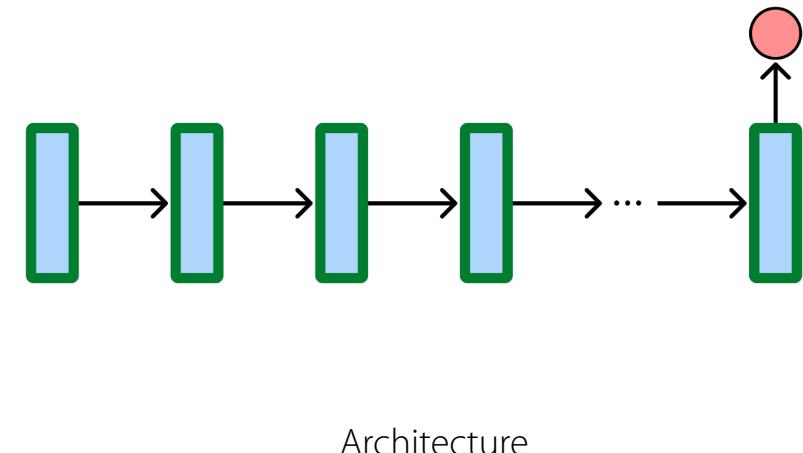
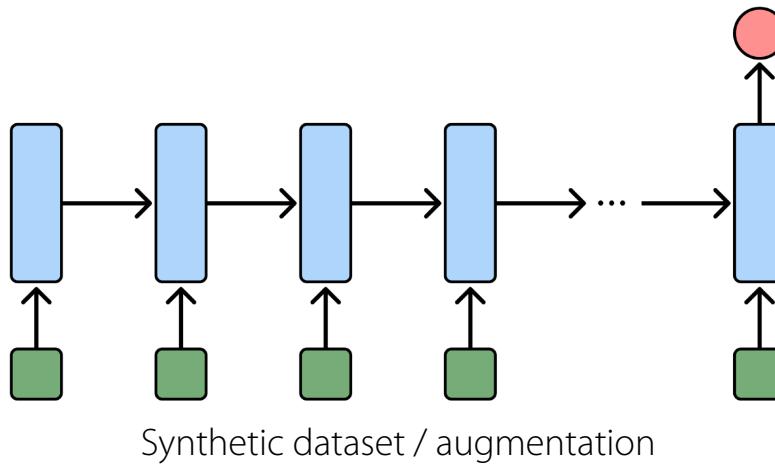
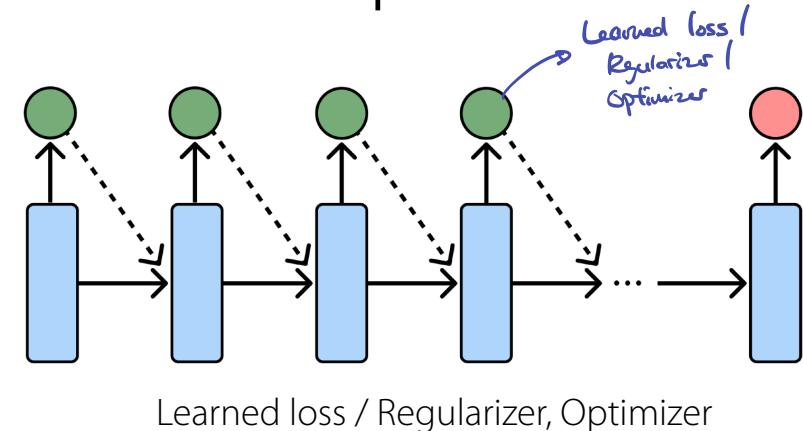
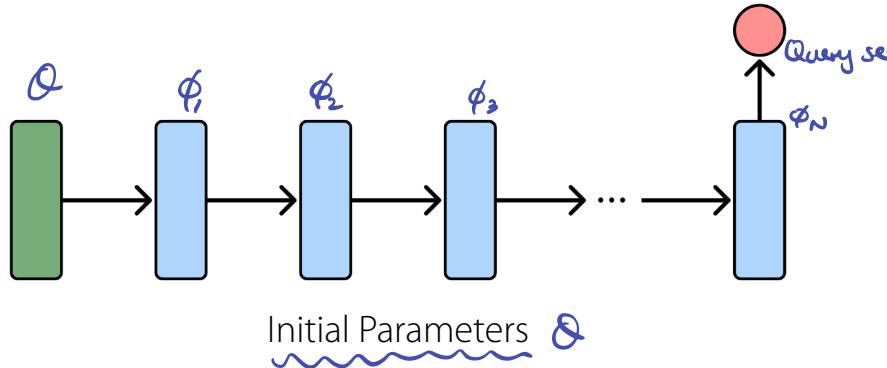


Problem for all 4 : You have to backprop through v. long chains to get from red to green.
→ Can't apply direct backprop.

What the fuck is this??

↙
What???

Unrolled Computation Graphs



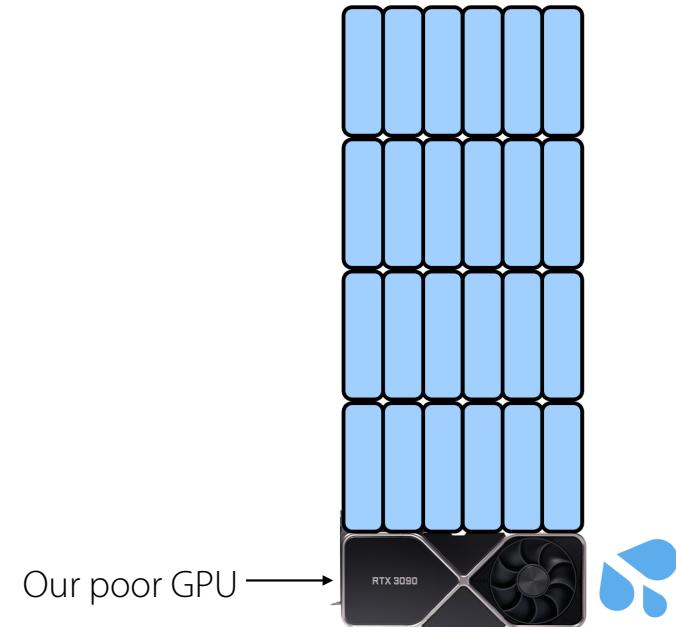
Plan for Today

Why consider large-scale meta-optimization?

Applications

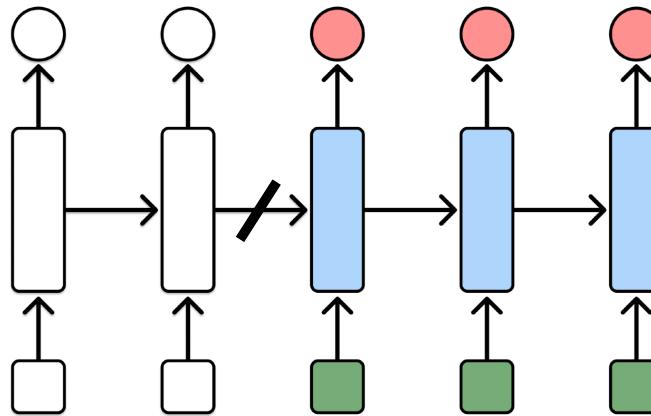
Approaches

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



Q: Is this specific to RNNs? Or does it also apply to things like MAML?

A: This works for anything with an inner sequential optimization loop.
Moving to the right (btw blue boxes below) would be a step of gradient descent.



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

Question: what could happen if we use short T?

↳ Backprop is faster, but don't learn from things earlier than the truncation.

Truncated Backpropagation

↳ Read paper: "Truncated Back-propagation for Bilevel Optimization"

```
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 = []
```

→ detach things when they became too old.

- + Simple: autograd handles everything
 - Biased estimator
 - Cannot take long-range dependencies into account
 - Sequence length introduces a tradeoff between correctness and memory cost
- ↳ depends on the value of T.

Plan for Today

Why consider large-scale meta-optimization?

Applications

Approaches

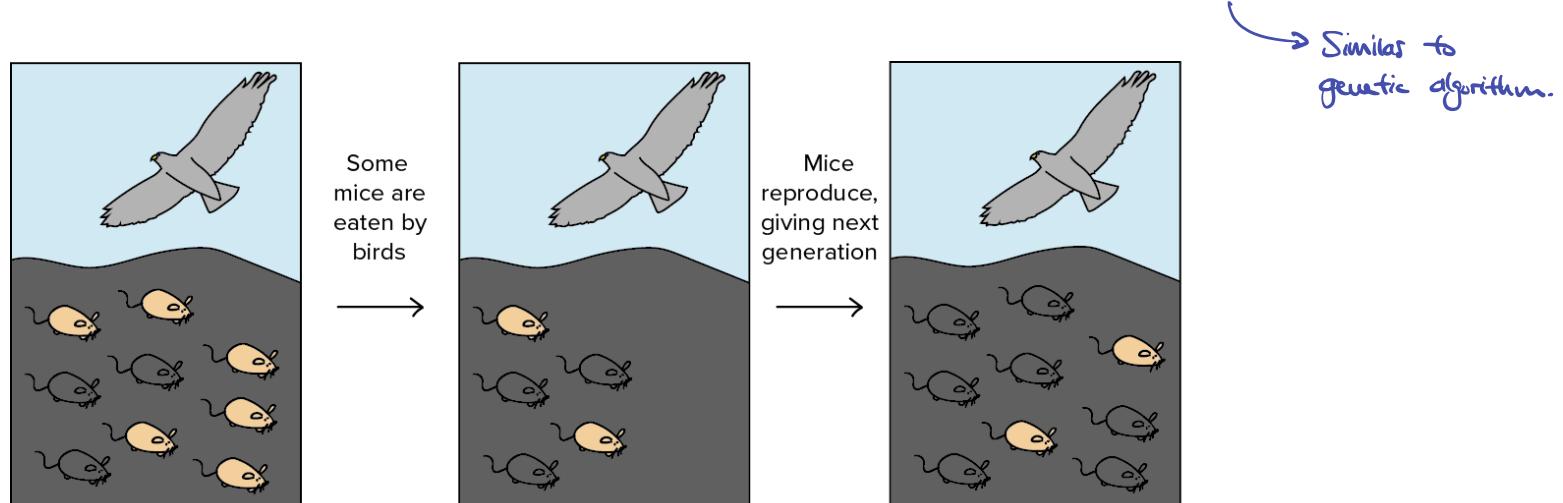
- Truncated backpropagation
- Gradient-free optimization

Gradient-free Optimization

Backpropagation is costly for large computation graphs...

Optimization **does not necessarily require gradients!** *→ Just avoid the issue of long-range gradient updating*

Evolution Strategies: Estimates gradients using stochastic finite differences.



Evolution Strategies

→ Applying this for the outer loop!

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

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

→ Find the top n -performing points, where $n < N$, and find the μ, σ^2 of that distribution.

→ Approximate gradient calculated from multiple full pass samplings.

Example: optimizing learning rate

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

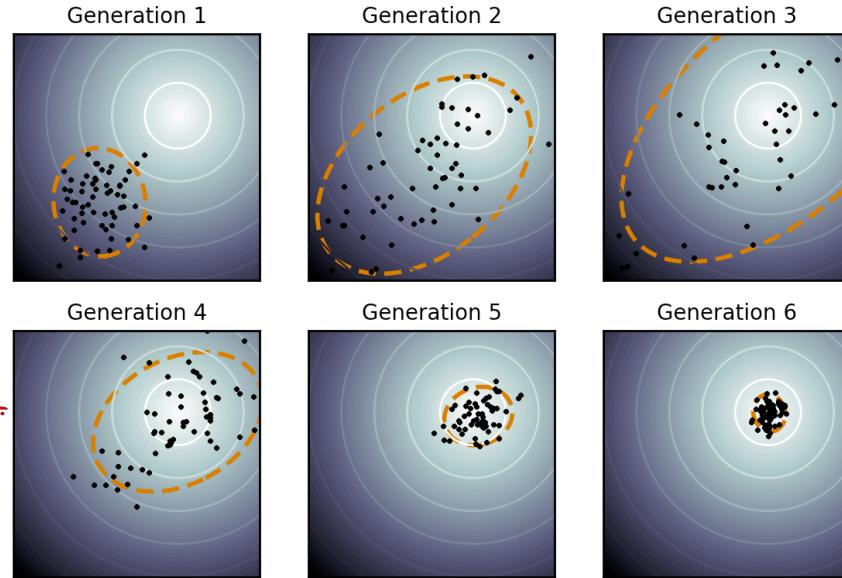
1. Sample lr: $\alpha^1, \alpha^2, \dots, \alpha^N \sim \mathcal{N}(\alpha, \sigma^2)$
2. Run SGD, get runs with best val accuracy: $\{e^1, \dots, e^n\} \subset \{\alpha^1, \dots, \alpha^N\}$
3. $\alpha, \sigma^2 \leftarrow \text{Avg}(e^1, \dots, e^n), \text{Var}(e^1, \dots, e^n)$

Here we run multiple instances of model param. optimizations. (using SGD, for instance).

Question: what could happen if we try to optimize **initial parameters** with ES?

Unlikely to observe good initial parameters, because parameter space is high-dimensional.

→ Evolution Strategies not v. appropriate

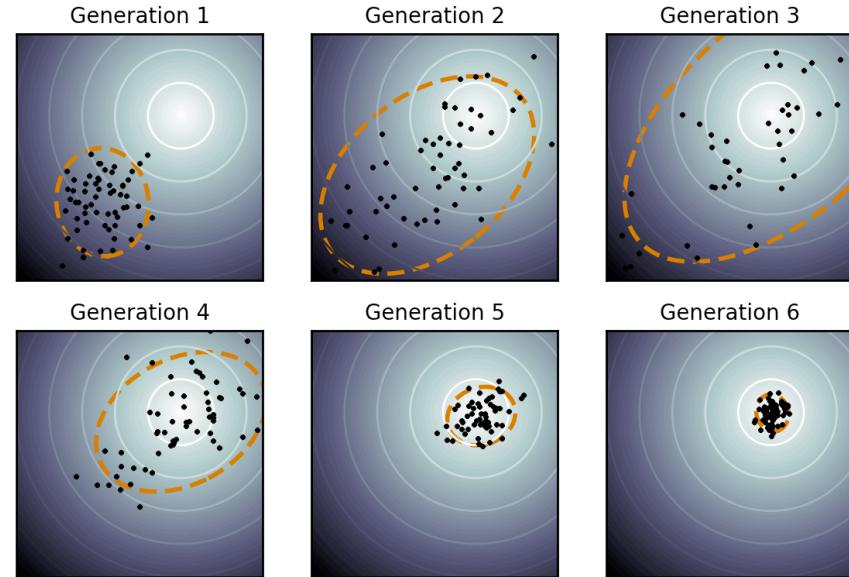


From: [Wikipedia CMA-ES page](#)

Evolution Strategies

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

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



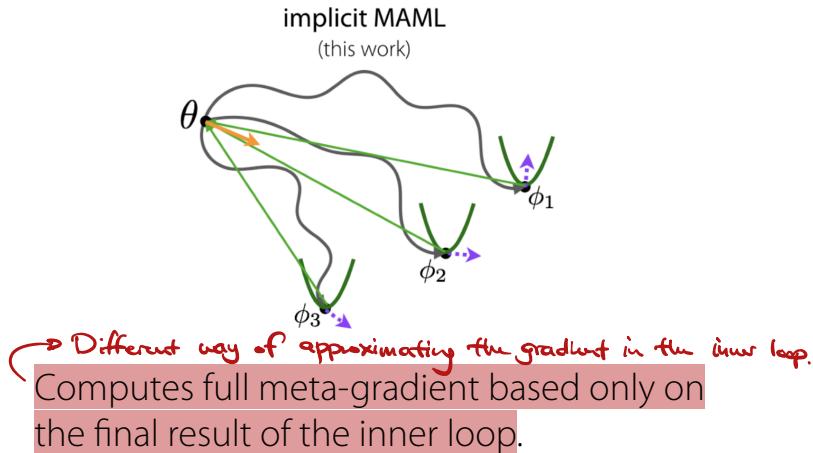
From: [Wikipedia CMA-ES page](#)

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

→ *bec. we're not calculating gradients*

Other Approaches

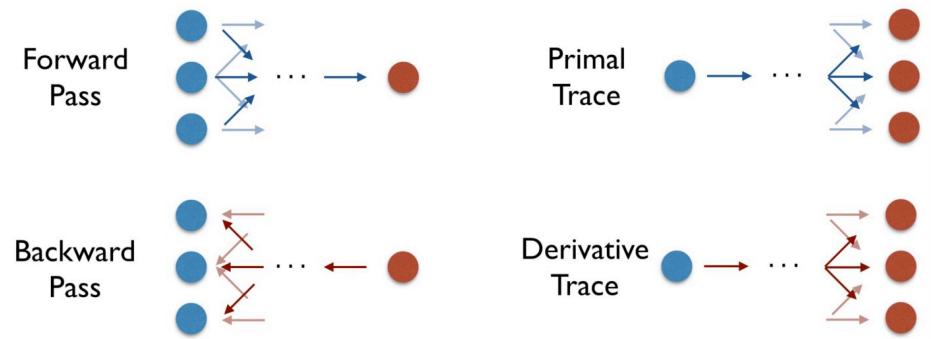
Implicit Differentiation



$$\theta \leftarrow \theta - \eta \frac{1}{M} \sum_{i=1}^M \frac{d\text{Alg}_i^*(\theta)}{d\theta} \nabla_\phi \mathcal{L}_i(\text{Alg}_i^*(\theta)).$$

$$\frac{d\text{Alg}_i^*(\theta)}{d\theta} = \left(\mathbf{I} + \frac{1}{\lambda} \nabla_\phi^2 \hat{\mathcal{L}}_i(\phi_i) \right)^{-1}$$

Forward-mode Differentiation



Uses the chain rule in the opposite direction from backprop, accumulating derivatives from start to finish.

Not sure why this is advantageous.

Plan for Today

Why consider large-scale meta-optimization?

Applications

Approaches

- 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**