ARUBA: Efficient and Adaptive Meta-Learning with Provable Guarantees

Misha Khodak

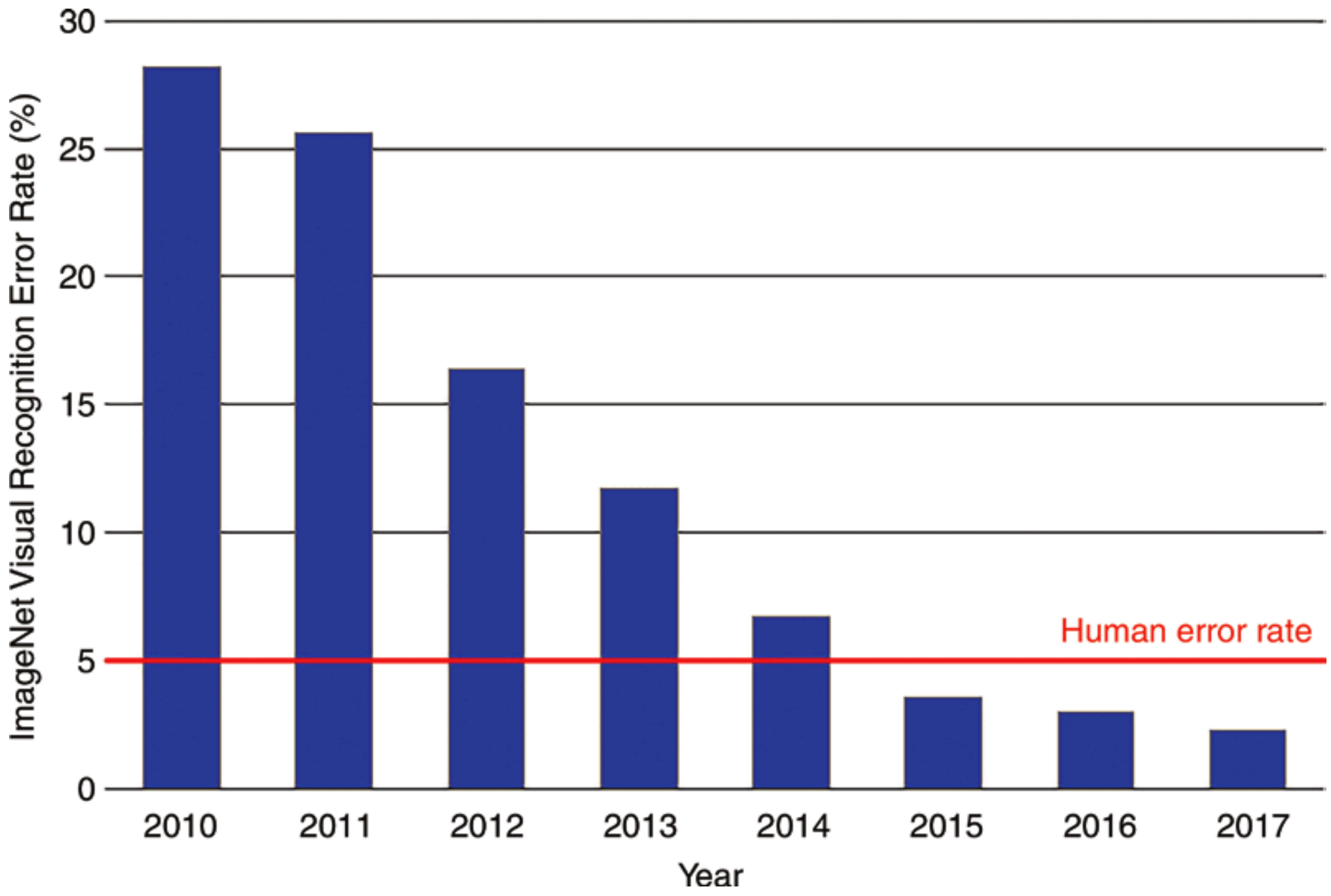
3 December 2019

Based on joint work with:

- **Nina Balcan, Ameet Talwalkar**
- **Jeff Li, Sebastian Caldas, Ameet Talwalkar**

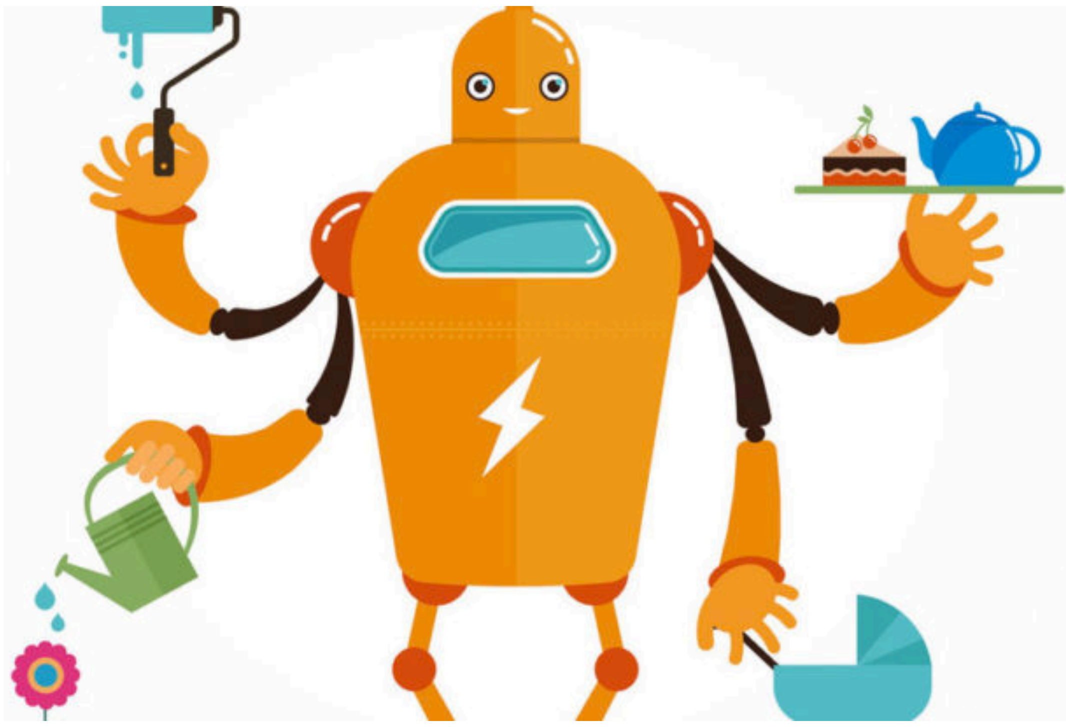
Carnegie Mellon University

Demanding more from machine learning pipelines



Demanding more from machine learning pipelines

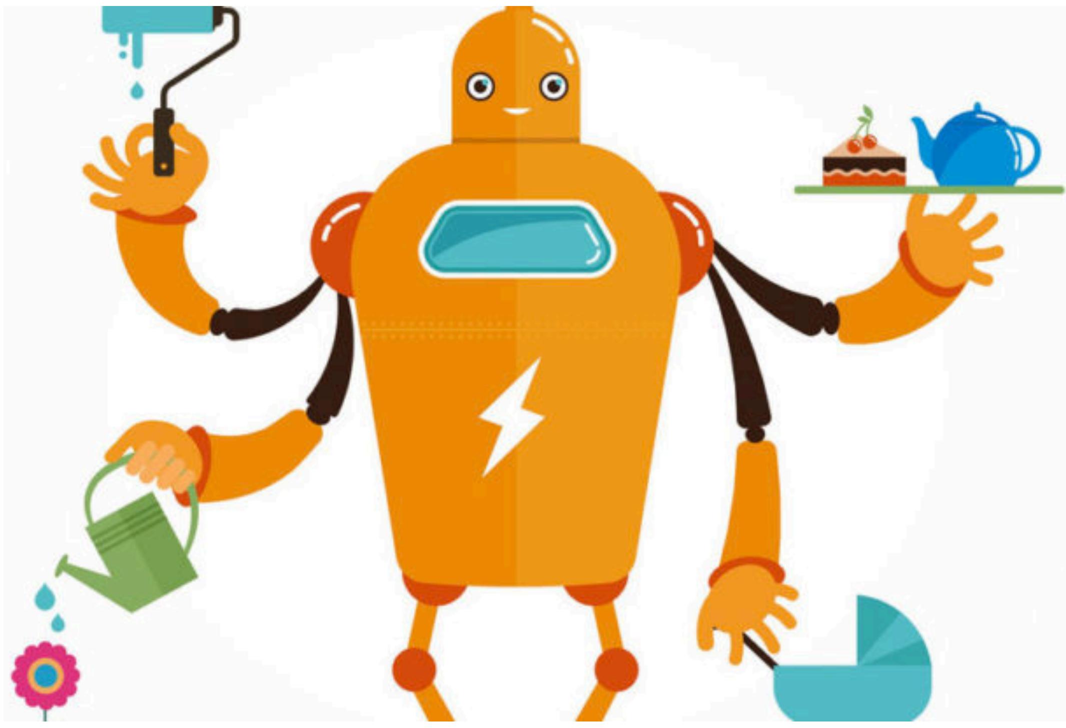
- learning deep models with limited data from many tasks



Source: Taboola Engineering

Demanding more from machine learning pipelines

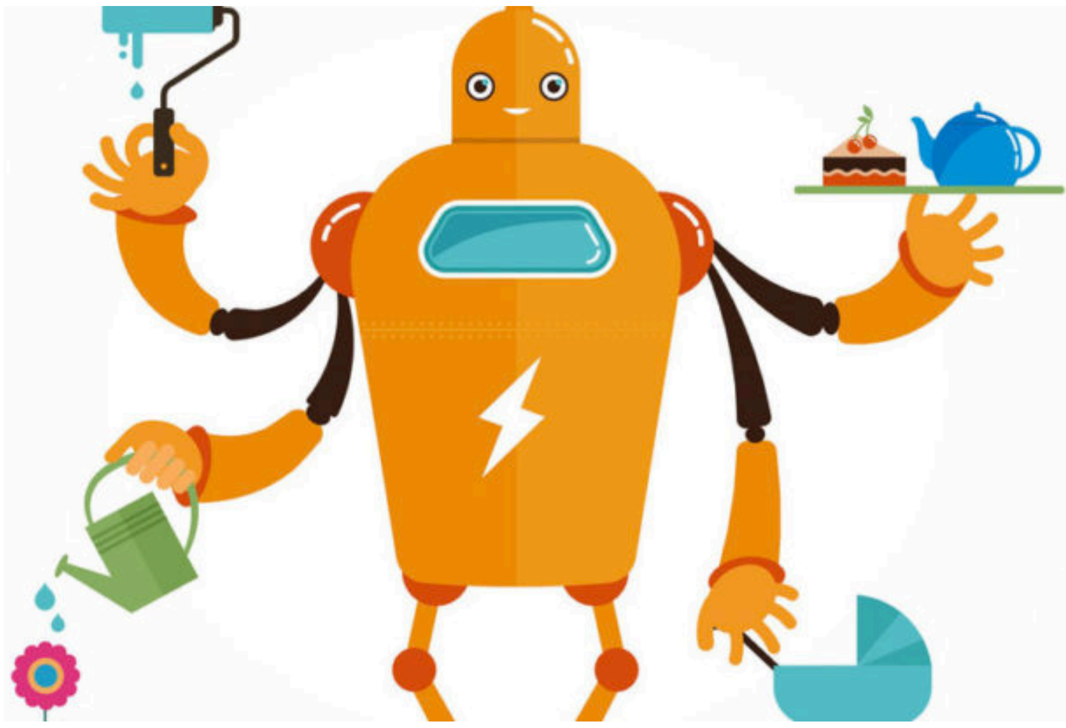
- learning deep models with limited data from many tasks
- sustained performance in changing environments



Source: Taboola Engineering

Demanding more from machine learning pipelines

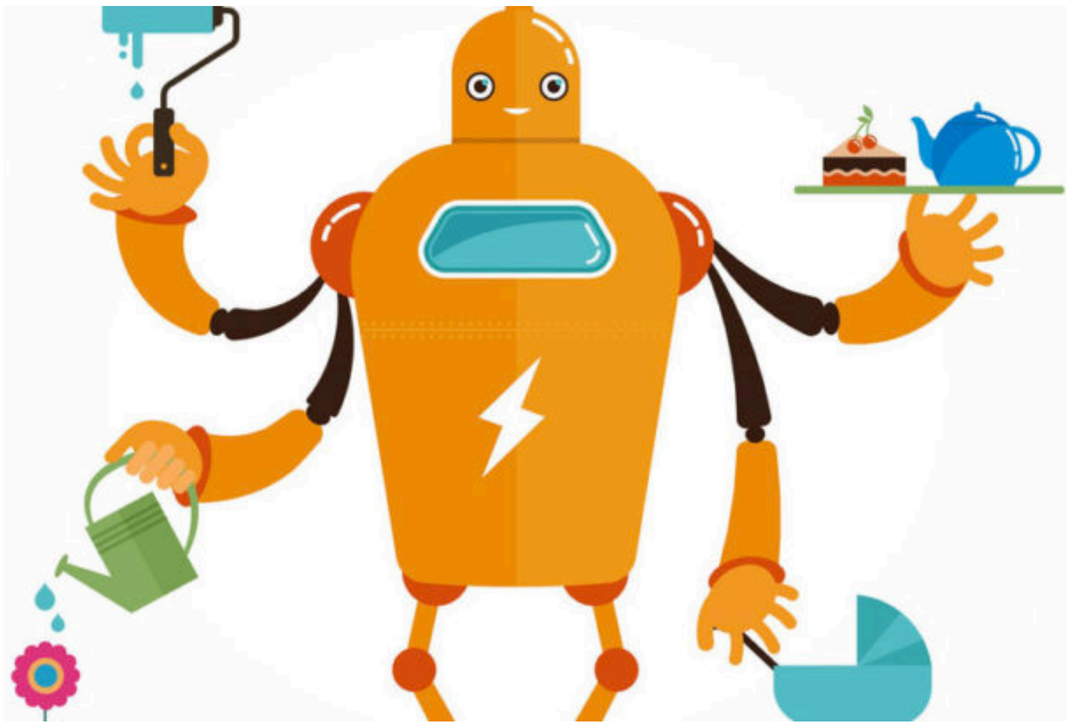
- learning deep models with limited data from many tasks
- sustained performance in changing environments
- fast adaptation to unseen distributions



Source: Taboola Engineering

Demanding more from machine learning pipelines

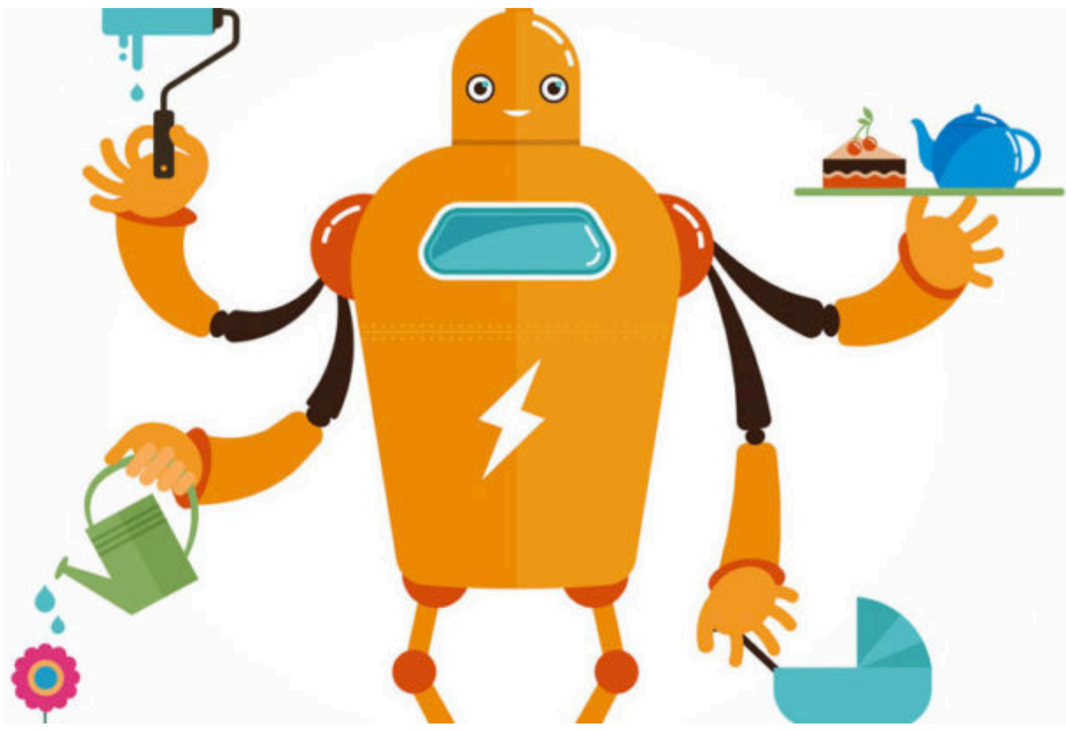
- learning deep models with limited data from many tasks
- sustained performance in changing environments
- fast adaptation to unseen distributions
- distributed training and inference



Source: Taboola Engineering

Demanding more from machine learning pipelines

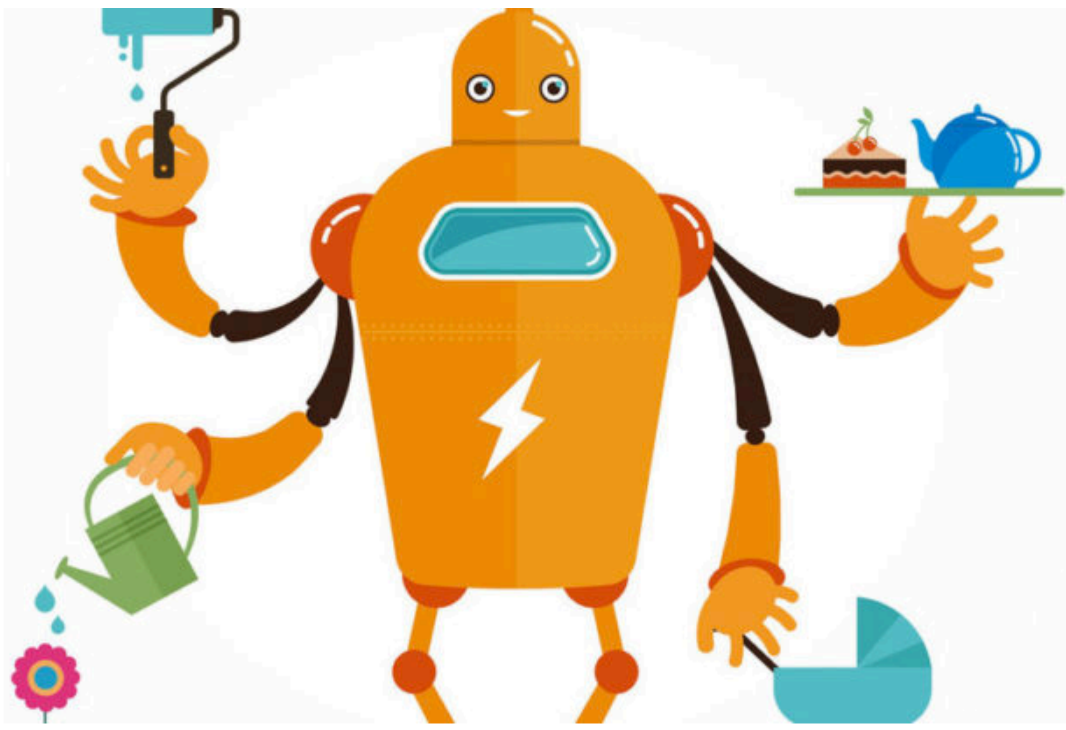
- learning deep models with limited data from many tasks [multi-task]
- sustained performance in changing environments [lifelong]
- fast adaptation to unseen distributions [transfer]
- distributed training and inference [federated]



Source: Taboola Engineering

Demanding more from machine learning pipelines

- learning deep models with limited data from many tasks [multi-task]
- sustained performance in changing environments [lifelong]
- fast adaptation to unseen distributions [transfer]
- distributed training and inference [federated]



Source: Taboola Engineering

Meta-learning:
a popular multi-task formulation of these objectives

Meta-learning:
a popular multi-task formulation of new objectives for ML

- improves ML by “learning-to-learn” across tasks
- promising performance in a variety of fields
- fast-evolving and poorly understood methodology

Meta-learning:
a popular multi-task formulation of new objectives for ML

- improves ML by “learning-to-learn” across tasks
- promising performance in a variety of fields
- fast-evolving and poorly understood methodology

This talk:
meta-learning algorithms with provable guarantees.

Standard ML: supervised prediction

Input $x \in \mathcal{X}$

Configurable Function

Output $y \in \mathcal{Y}$

Standard ML: supervised prediction

Input $x \in \mathcal{X}$

Configurable Function

Output $y \in \mathcal{Y}$

"meta-learning is"

$f_\theta : \mathcal{X} \mapsto \mathcal{Y}$

"interesting"

Standard ML: supervised prediction

Input $x \in \mathscr{X}$

Configurable Function

Output $y \in \mathscr{Y}$

“meta-learning is”

$$f_\theta : \mathscr{X} \mapsto \mathscr{Y}$$

“interesting”

Goal: find θ such that $f_\theta(x) = y$ for all $(x, y) \sim \mathscr{D}$

Standard ML: supervised prediction

Input $x \in \mathcal{X}$

Configurable Function

Output $y \in \mathcal{Y}$

"meta-learning is"

$$f_\theta : \mathcal{X} \mapsto \mathcal{Y}$$

"interesting"

Goal: find θ such that $f_\theta(x) = y$ for all $(x, y) \sim \mathcal{D}$

How: use training data $(x_1, y_1), \ldots, (x_m, y_m) \sim \mathcal{D}$

Standard ML: supervised prediction

“meta-learning is” $f_\theta : \mathcal{X} \mapsto \mathcal{Y}$ **“interesting”**

training data

Standard ML: stochastic gradient descent (SGD)

"meta-learning is" $f_{\theta_1} : \mathscr{X} \mapsto \mathscr{Y}$ **"banana"**

randomly initialize $\theta_1 \in \Theta$

training data

Standard ML: stochastic gradient descent (SGD)

“meta-learning is” $f_{\theta_1} : \mathcal{X} \mapsto \mathcal{Y}$ **“banana”**

randomly initialize $\theta_1 \in \Theta$

pick learning rate $\eta > 0$

training data

Standard ML: stochastic gradient descent (SGD)

"meta-learning is" $\quad f_{\theta_1} : \mathcal{X} \mapsto \mathcal{Y} \quad$ **"banana"**

randomly initialize $\theta_1 \in \Theta$

pick learning rate $\eta > 0$

for $i = 1, \ldots, m$

    sample (x_i, y_i)

training data

Project Gutenberg

Standard ML: stochastic gradient descent (SGD)

"meta-learning is" $f_{\theta_{100}} : \mathscr{X} \mapsto \mathscr{Y}$ **"wooden"**

randomly initialize $\theta_1 \in \Theta$

pick learning rate $\eta > 0$

for $i = 1, \ldots, m$

  sample (x_i, y_i)

  $\theta_{i+1} \leftarrow \theta_i - \eta \nabla L(f_{\theta_i}(x_i), y_i)$

differentiable
loss function

training data

Project
Gutenberg

Standard ML: stochastic gradient descent (SGD)

“meta-learning is” $f_{\theta_{1E4}} : \mathscr{X} \mapsto \mathscr{Y}$ **“boring”**

randomly initialize $\theta_1 \in \Theta$

pick learning rate $\eta > 0$

for $i = 1, \ldots, m$

  sample (x_i, y_i)

  $\theta_{i+1} \leftarrow \theta_i - \eta \nabla L(f_{\theta_i}(x_i), y_i)$

differentiable loss function

training data

Project Gutenberg

Standard ML: stochastic gradient descent (SGD)

“meta-learning is” $f_{\theta_{1E6}} : \mathcal{X} \mapsto \mathcal{Y}$ **“fine”**

randomly initialize $\theta_1 \in \Theta$

pick learning rate $\eta > 0$

for $i = 1, \ldots, m$

  sample (x_i, y_i)

  $\theta_{i+1} \leftarrow \theta_i - \eta \nabla L(f_{\theta_i}(x_i), y_i)$

differentiable
loss function

training data

Project
Gutenberg

Standard ML: stochastic gradient descent (SGD)

“meta-learning is” $\quad f_{\hat{\theta}} : \mathcal{X} \mapsto \mathcal{Y} \quad$ **“interesting”**

randomly initialize $\theta_1 \in \Theta$

pick learning rate $\eta > 0$

for $i = 1, \ldots, m$

  sample (x_i, y_i)

  $\theta_{i+1} \leftarrow \theta_i - \eta \nabla L(f_{\theta_i}(x_i), y_i)$

return $\hat{\theta} \leftarrow \theta_{m+1}$

training data

Meta-learning: many tasks, few examples

“meta-learning is”

$$f_\theta : \mathscr{X} \mapsto \mathscr{Y}$$

“interesting”

Meta-learning: many tasks, few examples

“meta-learning is” $f_\theta : \mathscr{X} \mapsto \mathscr{Y}$ **“interesting”**

“meta-learning is”  **“great!”**

Meta-learning: many tasks, few examples

"meta-learning is" $f_\theta : \mathscr{X} \mapsto \mathscr{Y}$ "interesting"

"meta-learning is"  "great!"

"meta-learning is"  "dope"

Meta-learning: many tasks, few examples

“meta-learning is” $f_\theta : \mathscr{X} \mapsto \mathscr{Y}$ **“interesting”**

“meta-learning is”  **“great!”**

“meta-learning is”  **“dope”**

Can we just use a single global model?

“meta-learning is” $f_{\hat{\theta}} : \mathscr{X} \mapsto \mathscr{Y}$

randomly initialize $\theta_1 \in \Theta$

pick learning rate $\eta > 0$

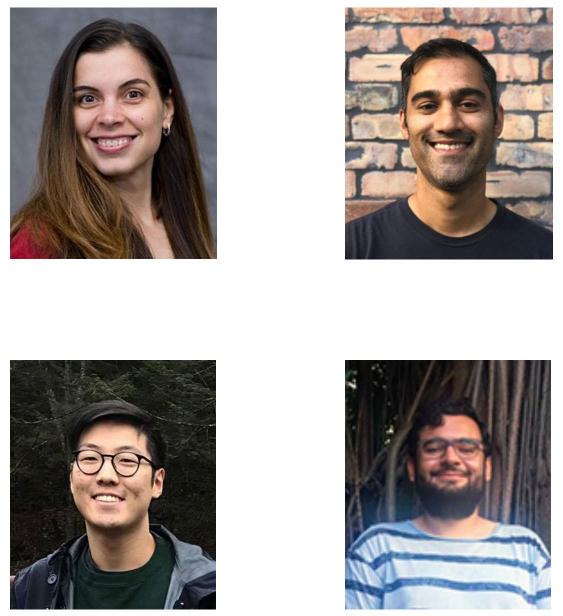
for $i = 1, \ldots, m$

    sample (x_i, y_i)

    $\theta_{i+1} \leftarrow \theta_i - \eta \nabla L(f_{\theta_i}(x_i), y_i)$

return $\hat{\theta} \leftarrow \theta_{m+1}$

training data



Can we just use a single global model?

no personalization

"meta-learning is" $f_{\hat{\theta}} : \mathcal{X} \mapsto \mathcal{Y}$ "interesting"

randomly initialize $\theta_1 \in \Theta$

pick learning rate $\eta > 0$

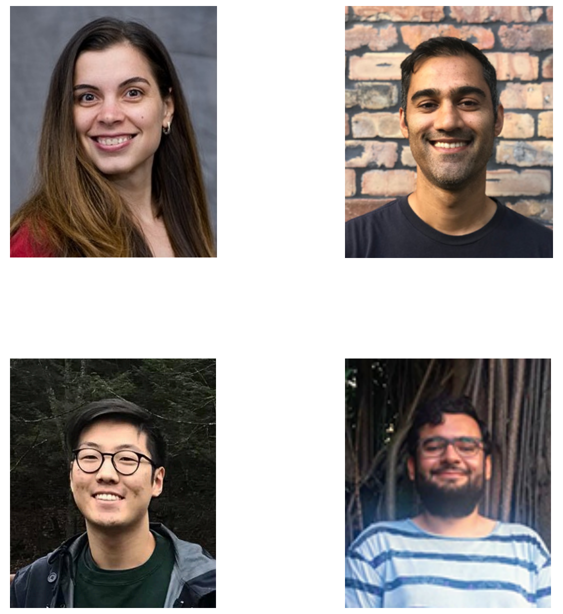
for $i = 1, \ldots, m$

  sample (x_i, y_i)

  $\theta_{i+1} \leftarrow \theta_i - \eta \nabla L(f_{\theta_i}(x_i), y_i)$

return $\hat{\theta} \leftarrow \theta_{m+1}$

training data



Can we train one model per person?

"meta-learning is"

$$f_{\hat{\theta}} : \mathcal{X} \mapsto \mathcal{Y}$$

randomly initialize $\theta_1 \in \Theta$

pick learning rate $\eta > 0$

for $i = 1, \ldots, m$

  sample (x_i, y_i)

  $\theta_{i+1} \leftarrow \theta_i - \eta \nabla L(f_{\theta_i}(x_i), y_i)$

return $\hat{\theta} \leftarrow \theta_{m+1}$

training data

Can we train one model per person?

☹ not enough data

"meta-learning is" $f_{\hat{\theta}} : \mathscr{X} \mapsto \mathscr{Y}$ "guarantees"

randomly initialize $\theta_1 \in \Theta$

pick learning rate $\eta > 0$

for $i = 1, \ldots, m$

  sample (x_i, y_i)

  $\theta_{i+1} \leftarrow \theta_i - \eta \nabla L(f_{\theta_i}(x_i), y_i)$

return $\hat{\theta} \leftarrow \theta_{m+1}$

training data

Can we learn an initialization for SGD?

"meta-learning is" $f_{\hat{\theta}} : \mathscr{X} \mapsto \mathscr{Y}$

use learned initialization $\theta_1 = \hat{\phi}$

pick learning rate $\eta > 0$

for $i = 1,\ldots,m$

  sample (x_i, y_i)

  $\theta_{i+1} \leftarrow \theta_i - \eta \nabla L(f_{\theta_i}(x_i), y_i)$

return $\hat{\theta} \leftarrow \theta_{m+1}$

training data

Can we learn an initialization for SGD?

“meta-learning is” $f_{\hat{\theta}} : \mathcal{X} \mapsto \mathcal{Y}$ **“great!”**

use learned initialization $\theta_1 = \hat{\phi}$

pick learning rate $\eta > 0$

for $i = 1, \ldots, m$

  sample (x_i, y_i)

  $\theta_{i+1} \leftarrow \theta_i - \eta \nabla L(f_{\theta_i}(x_i), y_i)$

return $\hat{\theta} \leftarrow \theta_{m+1}$

training data

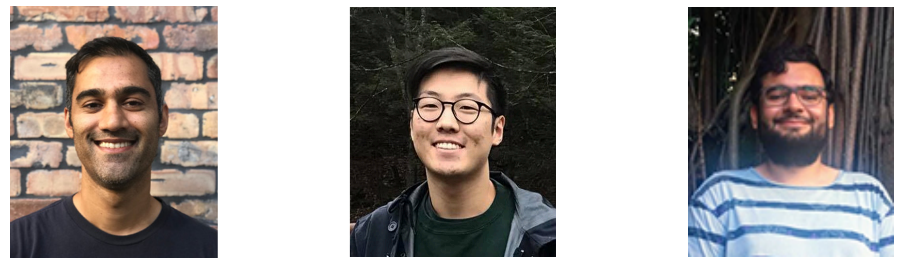
Gradient-Based Meta-Learning

randomly meta-initialize $\phi_1 \in \Theta$

pick meta-learning rate $\alpha > 0$

for task $t = 1, \ldots, T$

training tasks



Gradient-Based Meta-Learning

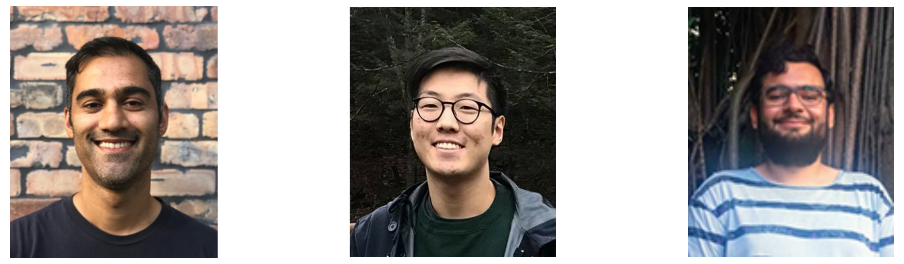
randomly meta-initialize $\phi_1 \in \Theta$

pick meta-learning rate $\alpha > 0$

for task $t = 1, \ldots, T$

sample task $\mathscr{D}_t$

training tasks



Gradient-Based Meta-Learning

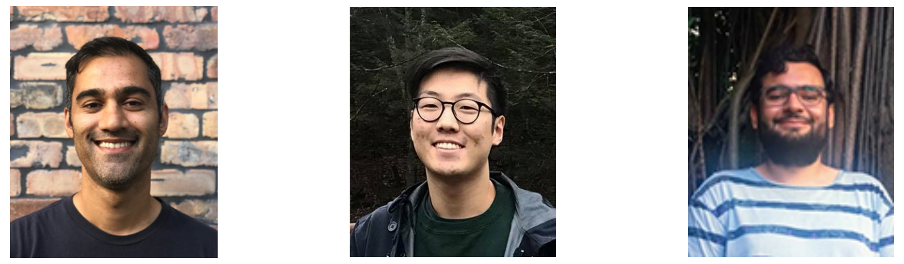
randomly meta-initialize $\phi_1 \in \Theta$

pick meta-learning rate $\alpha > 0$

for task $t = 1, \ldots, T$

sample task $\mathscr{D}_t$

training tasks



task data

Gradient-Based Meta-Learning

randomly meta-initialize $\phi_1 \in \Theta$

pick meta-learning rate $\alpha > 0$

for task $t = 1, \ldots, T$

sample task $\mathscr{D}_t$

initialize $\theta_{t,1} = \phi_t$

pick learning rate $\eta > 0$

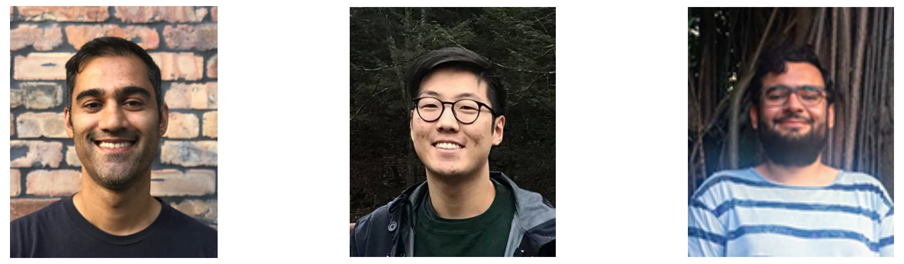
for $i = 1, \ldots, m$

sample $(x_{t,i}, y_{t,i})$

$\theta_{t,i+1} \leftarrow \theta_{t,i} - \eta \nabla L(f_{\theta_{t,i}}(x_{t,i}), y_{t,i})$

$\hat{\theta}_t \leftarrow \theta_{t,m+1}$

training tasks



task data

Gradient-Based Meta-Learning

randomly meta-initialize $\phi_1 \in \Theta$

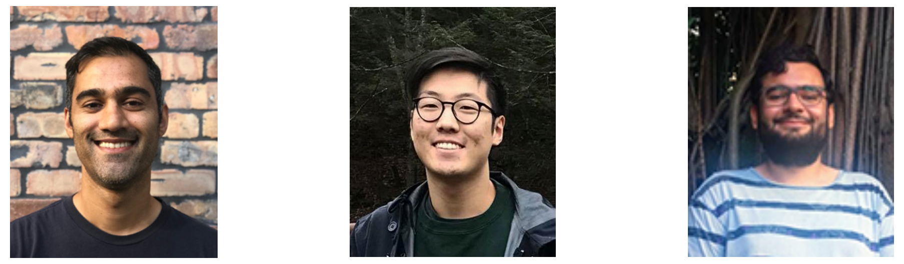
pick meta-learning rate $\alpha > 0$

for task $t = 1, \ldots, T$

sample task $\mathscr{D}_t$

$\hat{\theta}_t \leftarrow$ **within-task SGD**$(\mathscr{D}_t, \phi_t)$

training tasks



task data

Gradient-Based Meta-Learning

randomly meta-initialize $\phi_1 \in \Theta$

pick meta-learning rate $\alpha > 0$

for task $t = 1, \ldots, T$

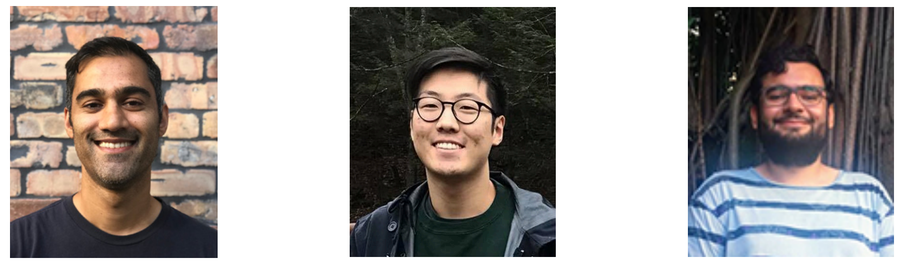
sample task $\mathscr{D}_t$

$\hat{\theta}_t \leftarrow$ **within-task SGD**$(\mathscr{D}_t, \phi_t)$

$$\phi_{t+1} \leftarrow (1 - \alpha)\phi_t + \alpha\hat{\theta}_t$$

“meta-update”

training tasks



task data

Gradient-Based Meta-Learning

randomly meta-initialize $\phi_1 \in \Theta$

pick meta-learning rate $\alpha > 0$

for task $t = 1, \ldots, T$

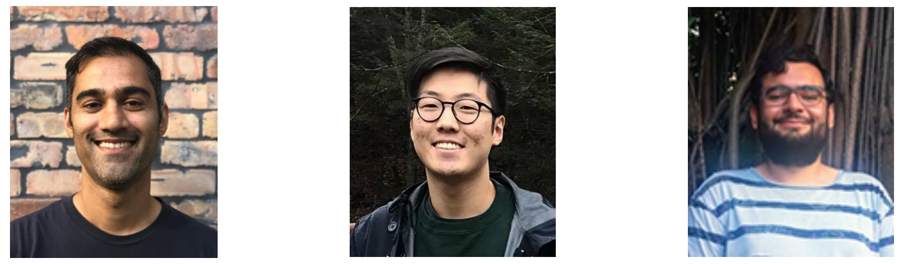
sample task $\mathscr{D}_t$

$\hat{\theta}_t \leftarrow$ **within-task SGD**$(\mathscr{D}_t, \phi_t)$

$\phi_{t+1} \leftarrow (1-\alpha)\phi_t + \alpha\hat{\theta}_t$

return ϕ_{T+1}

training tasks



task data

Gradient-Based Meta-Learning

randomly meta-initialize $\phi_1 \in \Theta$

pick meta-learning rate $\alpha > 0$

for task $t = 1, \ldots, T$

sample task $\mathscr{D}_t$

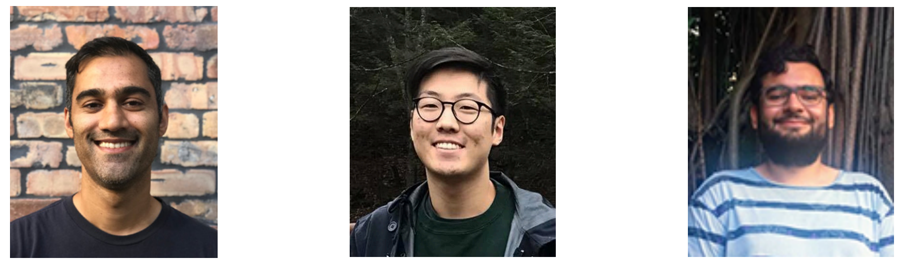
$$\boxed{\hat{\theta}_t \leftarrow \textbf{within-task SGD}(\mathscr{D}_t, \phi_t)}$$

$$\phi_{t+1} \leftarrow (1 - \alpha)\phi_t + \alpha\hat{\theta}_t$$

return ϕ_{T+1}

(**later called** $\hat{\phi}$)

training tasks



task data

Some successful gradient-based algorithms

randomly meta-initialize $\phi_1 \in \Theta$

pick meta-learning rate $\alpha > 0$

for task $t = 1, \ldots, T$

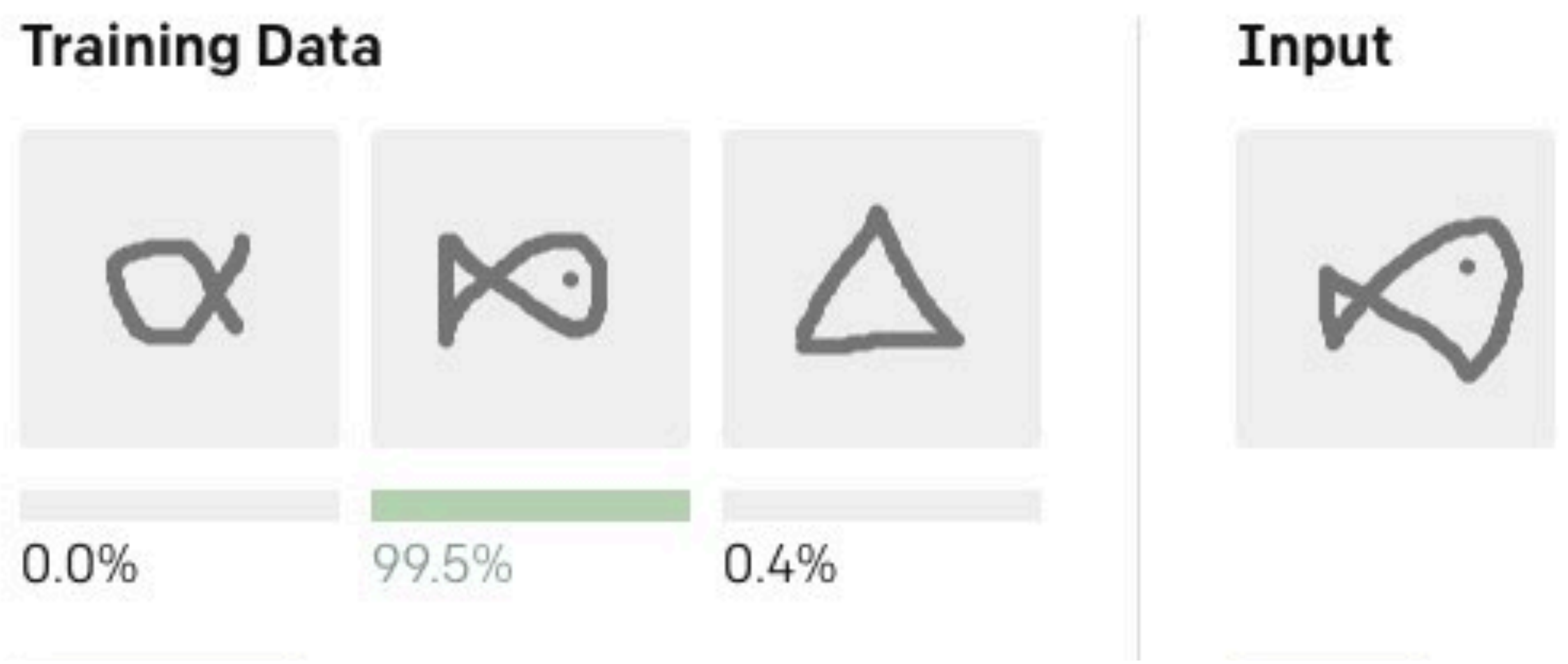
sample task $\mathscr{D}_t$

$\hat{\theta}_t \leftarrow$ **within-task SGD**$(\mathscr{D}_t, \phi_t)$

$$\phi_{t+1} \leftarrow (1 - \alpha)\phi_t + \alpha\hat{\theta}_t$$

return ϕ_{T+1}

Reptile [Nichol-Achiam-Schulman]



Few-Shot Learning

Some successful gradient-based algorithms

randomly meta-initialize $\phi_1 \in \Theta$
pick meta-learning rate $\alpha > 0$

for task $t = 1, \ldots, T$

sample task $\mathscr{D}_t$

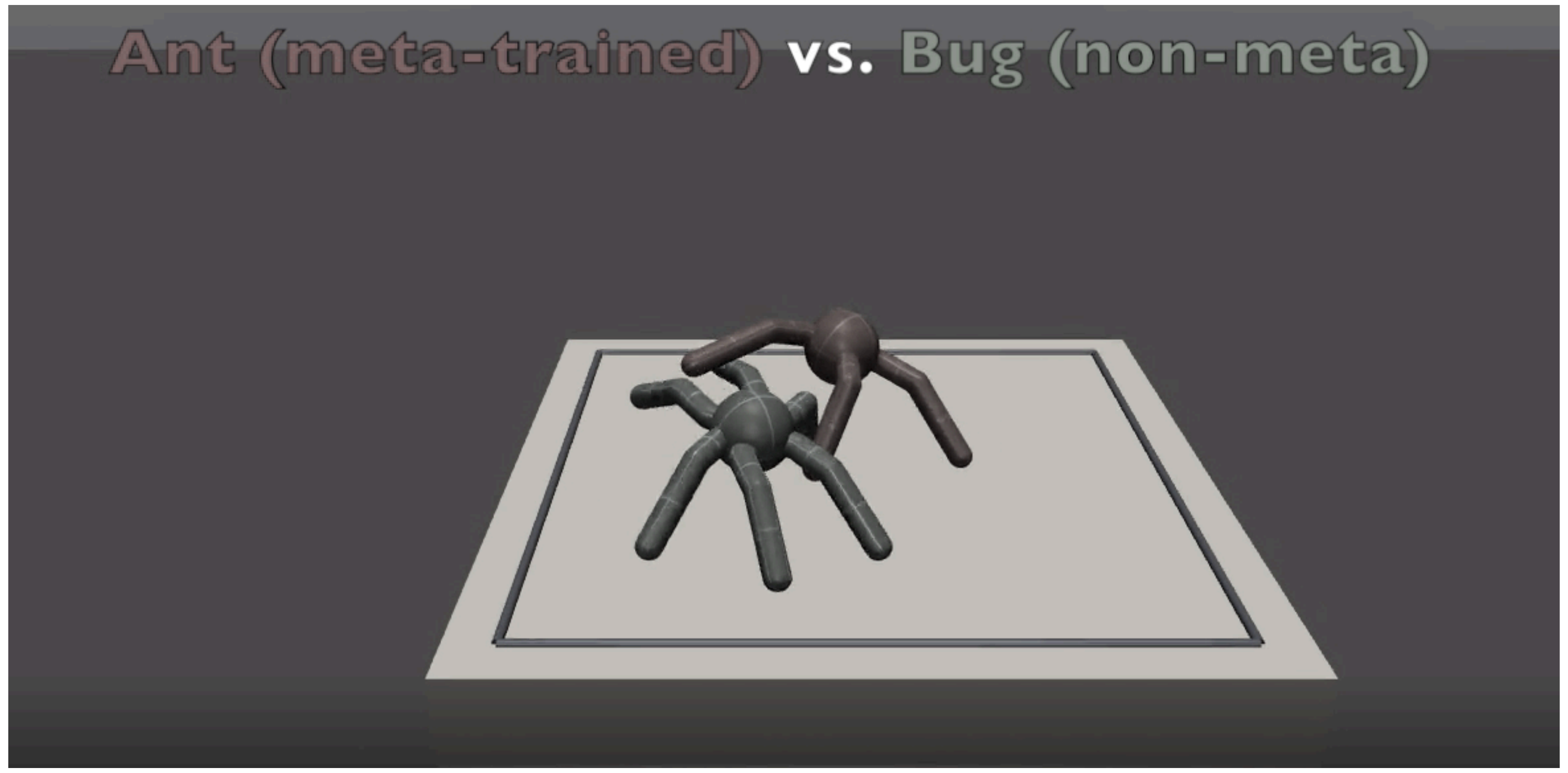
$\hat{\theta}_t \leftarrow$ **within-task SGD**$(\mathscr{D}_t, \phi_t)$

$\phi_{t+1} \leftarrow (1 - \alpha)\phi_t + \alpha\hat{\theta}_t$

return ϕ_{T+1}

replace by (non-stochastic) gradient descent

MAML [Finn-Abbeel-Levine]



Meta Reinforcement Learning

Some successful gradient-based algorithms

randomly meta-initialize $\phi_1 \in \Theta$
pick meta-learning rate $\alpha > 0$

for task $t = 1, \ldots, T$

sample task $\mathscr{D}_t$

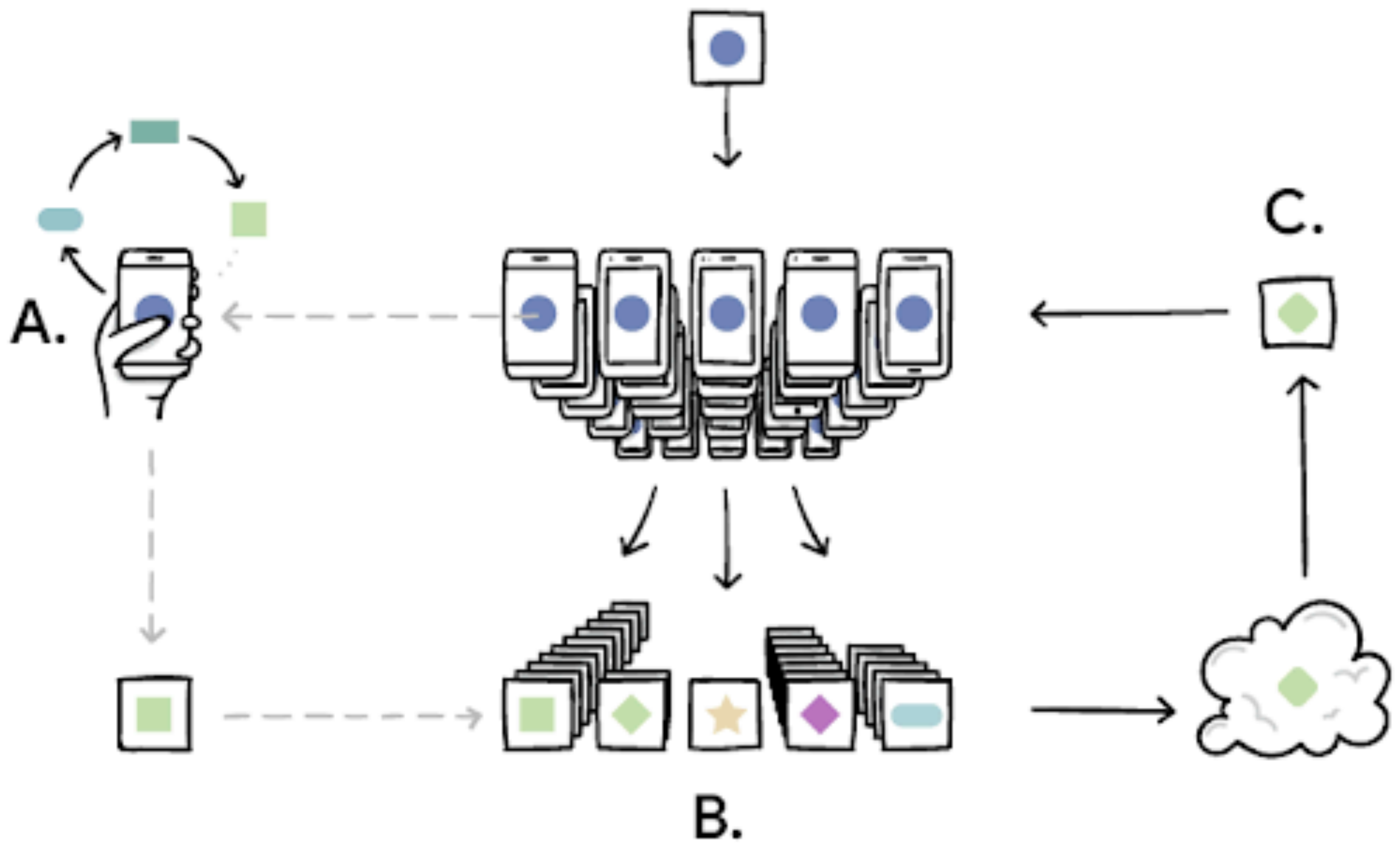
$\hat{\theta}_t \leftarrow$ **within-task SGD**$(\mathscr{D}_t, \phi_t)$

$\phi_{t+1} \leftarrow (1 - \alpha)\phi_t + \alpha\hat{\theta}_t$

return ϕ_{T+1}

run k tasks in parallel, update using their average last iterate

FedAvg [McMahan et al.]



Federated Learning with Personalization

Gradient-based meta-learning is simple & flexible…

Input: T few-shot training tasks $\{\mathscr{D}\}_1^T$

Algorithm: General; only assumes gradient updates

Output: Initialization $\hat{\phi}$ for few-shot test task

…what is it doing?

Input: T few-shot training tasks $\{\mathscr{D}\}_1^T$

Algorithm: General; only assumes gradient updates

Output: Initialization $\hat{\phi}$ for few-shot test task

Why/when do gradient-based methods work?

…what is it doing?

Input: T few-shot training tasks $\{\mathscr{D}\}_1^T$

Algorithm: General; only assumes gradient updates

Output: Initialization $\hat{\phi}$ for few-shot test task

Why/when do gradient-based methods work?

- What **provable guarantees** do these algorithms have?
- Can we **design new algorithms** for settings of interest?

ARUBA: Our new theoretical framework for meta-learning

Nina Balcan

Ameet Talwalkar

ARUBA: Our new theoretical framework for meta-learning

Use online learning to obtain the **first provable guarantees** for initialization-based meta-learning

ARUBA: Our new theoretical framework for meta-learning

Use online learning to obtain the **first provable guarantees** for initialization-based meta-learning

Guarantees improve with **natural notions of task-similarity**

ARUBA: Our new theoretical framework for meta-learning

Use online learning to obtain the **first provable guarantees** for initialization-based meta-learning

Guarantees improve with **natural notions of task-similarity**

Adapt to changing task-environments

ARUBA: Our new theoretical framework for meta-learning

Use online learning to obtain the **first provable guarantees** for initialization-based meta-learning

Guarantees improve with **natural notions of task-similarity**

Obtain faster statistical rates

ARUBA: Our new theoretical framework for meta-learning

Use online learning to obtain the **first provable guarantees** for initialization-based meta-learning

Guarantees improve with **natural notions of task-similarity**

Derive new methods for a broad variety of multi-task settings

ARUBA Framework

- Low-sample learning and gradient-based meta-learning
- An illustrative result for learning an initialization

Applications

Meta-learning through the lens of online learning

Online Learning

for $i = 1, \ldots, m$

pick action $\theta_i \in \Theta$

suffer loss $\ell_i(\theta_i)$

Meta-learning through the lens of online learning

Measure per-task performance via **regret**

$$R = \sum_{i=1}^{m} \ell_i(\theta_i) - \ell_i(\theta^*)$$

best fixed action
in hindsight

Online Learning

for $i = 1,\ldots,m$

pick action $\theta_i \in \Theta$

suffer loss $\ell_i(\theta_i)$

Meta-learning through the lens of online learning

Measure per-task performance via **regret**

$$R = \sum_{i=1}^{m} \ell_i(\theta_i) - \ell_i(\theta^*)$$

best fixed action
in hindsight

Online Learning

for $i = 1, \ldots, m$
pick action $\theta_i \in \Theta$
suffer loss $\ell_i(\theta_i)$

Non-IID Data / Tasks: Models realistic settings (e.g. mobile, RL data; lifelong learning)

Meta-learning through the lens of online learning

Measure per-task performance via **regret**

$$R = \sum_{i=1}^{m} \ell_i(\theta_i) - \ell_i(\theta^*)$$

best fixed action
in hindsight

Online Learning

for $i = 1, \ldots, m$
pick action $\theta_i \in \Theta$
suffer loss $\ell_i(\theta_i)$

Non-IID Data / Tasks: Models realistic settings (e.g. mobile, RL data; lifelong learning)

IID Implications: Online-to-batch conversion results

Meta-learning through the lens of online learning

Measure per-task performance via **regret**

$$R = \sum_{i=1}^{m} \ell_i(\theta_i) - \ell_i(\theta^*)$$

best fixed action
in hindsight

Online Learning

for $i = 1, \ldots, m$
pick action $\theta_i \in \Theta$
suffer loss $\ell_i(\theta_i)$

Non-IID Data / Tasks: Models realistic settings (e.g. mobile, RL data; lifelong learning)

IID Implications: Online-to-batch conversion results

Generality: Can adapt / generalize numerous online learning results to meta-learning

Single-Task Learning

Training Data	Hypothesis Class	Loss Function
$(x_1, y_1), \ldots, (x_m, y_m)$	$\{f_\theta : \mathcal{X} \mapsto \mathcal{Y} : \theta \in \Theta \subset \mathbb{R}^d\}$	$\ell_i(\theta) = L(f_\theta(x_i), y_i)$

$L : \mathcal{Y} \times \mathcal{Y} \mapsto \mathbb{R}$

best fixed action
in hindsight

$$R = \sum_{i=1}^{m} \ell_i(\theta_i) - \ell_i(\theta^*)$$

Single-Task Learning

Training Data	Hypothesis Class	Loss Function
$(x_1, y_1), \ldots, (x_m, y_m)$	$\{f_\theta : \mathcal{X} \mapsto \mathcal{Y} : \theta \in \Theta \subset \mathbb{R}^d\}$	$\ell_i(\theta) = L(f_\theta(x_i), y_i)$

Online Gradient Descent (OGD)

randomly initialize $\theta_1 \in \Theta, \quad \eta > 0$

for $i = 1, \ldots, m$

$\theta_{i+1} \leftarrow \theta_i - \eta \nabla \ell_i(\theta_i)$

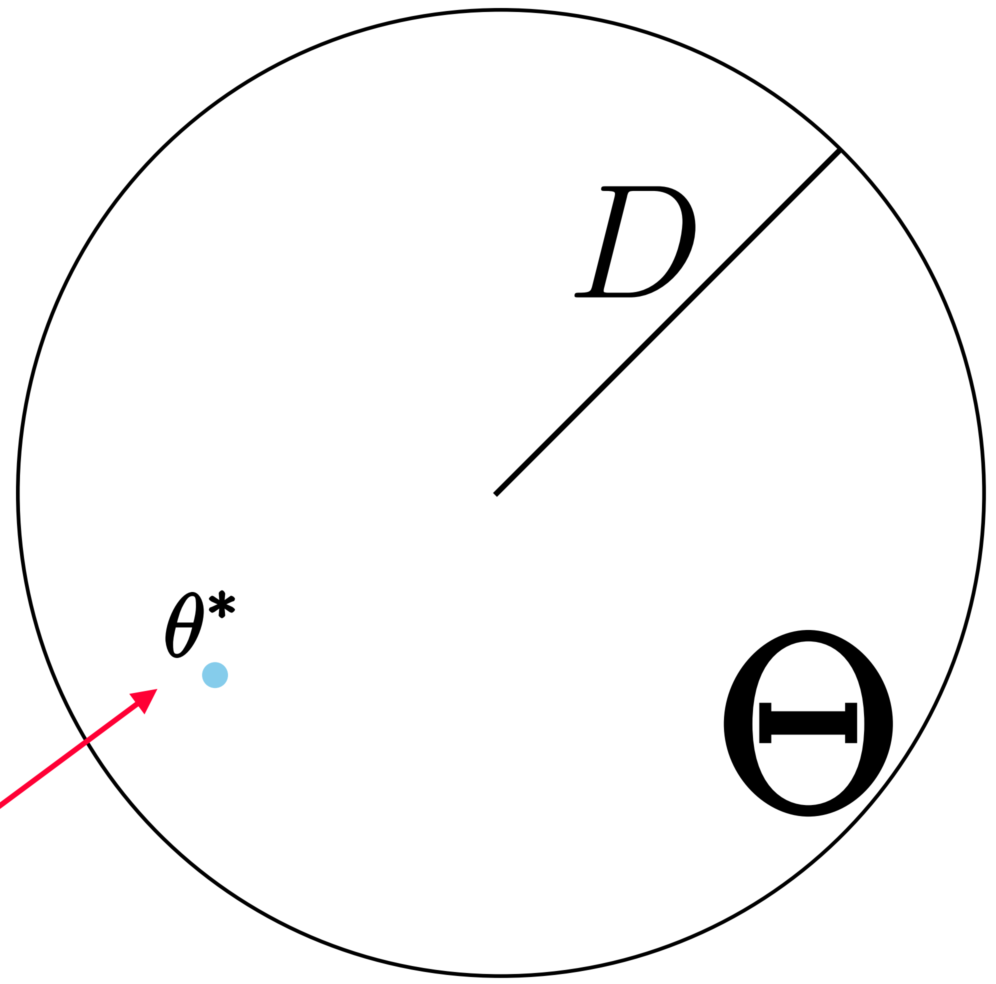
suffer $\ell_i(\theta_i)$

best fixed action
in hindsight

$$R = \sum_{i=1}^{m} \ell_i(\theta_i) - \ell_i(\theta^*)$$

Single-Task Regret

Training Data	Hypothesis Class	Loss Function
$(x_1, y_1), \ldots, (x_m, y_m)$	$\{f_\theta : \mathscr{X} \mapsto \mathscr{Y} : \theta \in \Theta \subset \mathbb{R}^d\}$	$\ell_i(\theta) = L(f_\theta(x_i), y_i)$



best fixed action
in hindsight

$$R = \sum_{i=1}^{m} \ell_i(\theta_i) - \ell_i(\theta^*)$$

Online Gradient Descent (OGD)

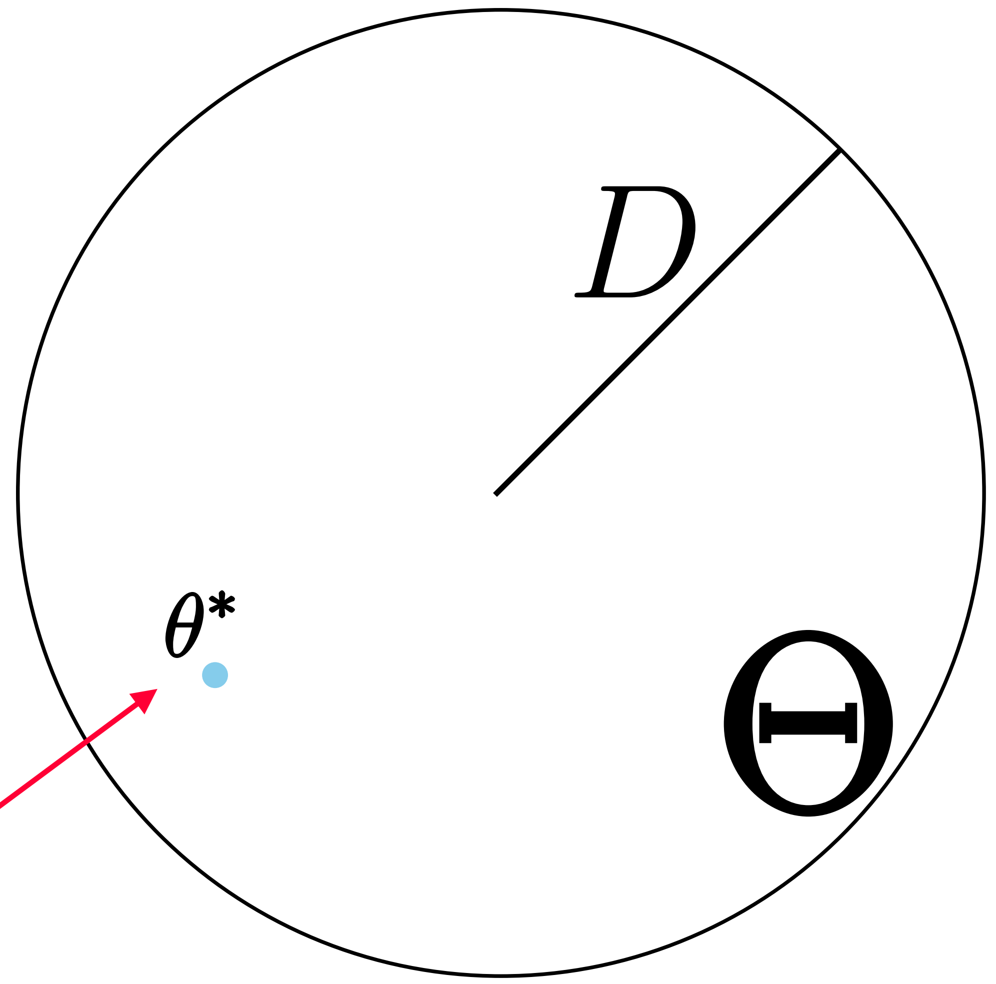
Size of Action Space: $D = \mathrm{radius}(\Theta)$

OGD upper-bound: $R = \mathcal{O}(D\sqrt{m})$

[Abernethy-Bartlett-Rakhlin-Tewari]

Single-Task Regret

Training Data	Hypothesis Class	Loss Function
$(x_1, y_1), \dots, (x_m, y_m)$	$\{f_\theta : \mathcal{X} \mapsto \mathcal{Y} : \theta \in \Theta \subset \mathbb{R}^d\}$	$\ell_i(\theta) = L(f_\theta(x_i), y_i)$



best fixed action
in hindsight

$$R = \sum_{i=1}^{m} \ell_i(\theta_i) - \ell_i(\theta^*)$$

Online Gradient Descent (OGD)

Size of Action Space: $D = \mathrm{radius}(\Theta)$

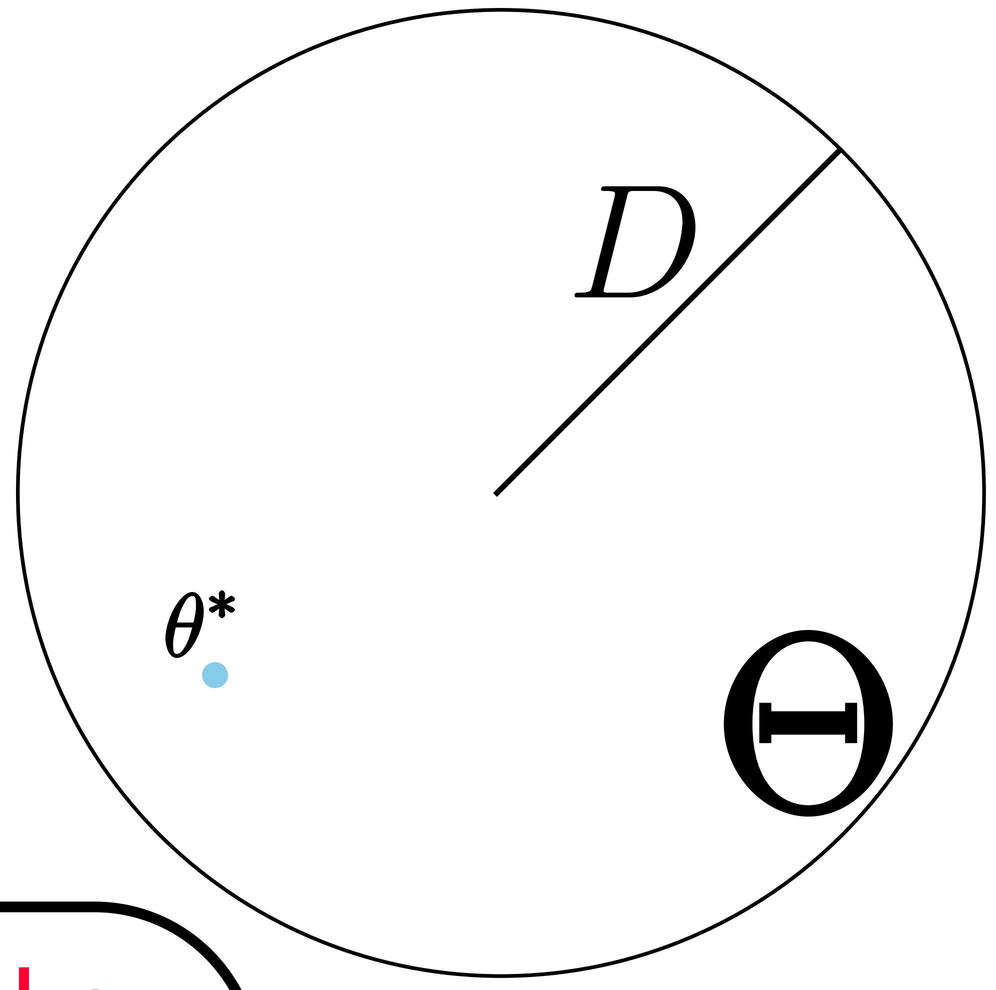
OGD upper-bound: $R = \mathcal{O}(D\sqrt{m})$

Matching lower-bound: $R = \Omega\left(D\sqrt{m}\right)$
(for any algorithm)

[Abernethy-Bartlett-Rakhlin-Tewari]

Single-Task Regret

Training Data	Hypothesis Class	Loss Function
$(x_1, y_1), \ldots, (x_m, y_m)$	$\{f_\theta : \mathscr{X} \mapsto \mathscr{Y} : \theta \in \Theta \subset \mathbb{R}^d\}$	$\ell_i(\theta) = L(f_\theta(x_i), y_i)$



Online Gradient Descent (OGD)

Size of Action Space: $D = \text{radius}(\Theta)$

OGD upper-bound: $R = \mathcal{O}(D\sqrt{m})$

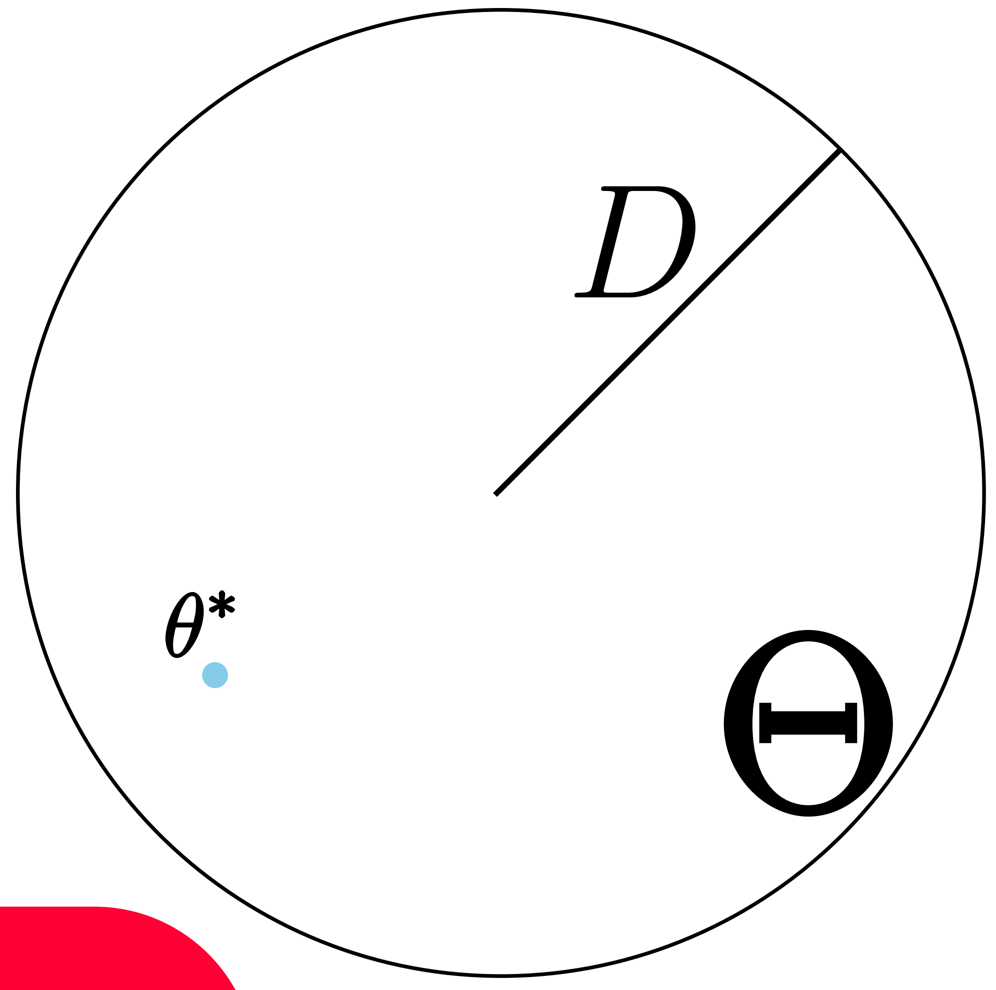
Matching lower-bound: $R = \Omega\left(D\sqrt{m}\right)$
(for any algorithm)

[Abernethy-Bartlett-Rakhlin-Tewari]

☹ Cannot hope to do well when ***m* is small**

Single-Task Regret

Training Data	Hypothesis Class	Loss Function
$(x_1, y_1), \ldots, (x_m, y_m)$	$\{f_\theta : \mathscr{X} \mapsto \mathscr{Y} : \theta \in \Theta \subset \mathbb{R}^d\}$	$\ell_i(\theta) = L(f_\theta(x_i), y_i)$



Online Gradient Descent (OGD)

Size of Action Space: $D = \text{radius}(\Theta)$

OGD upper-bound: $R = \mathcal{O}(D\sqrt{m})$

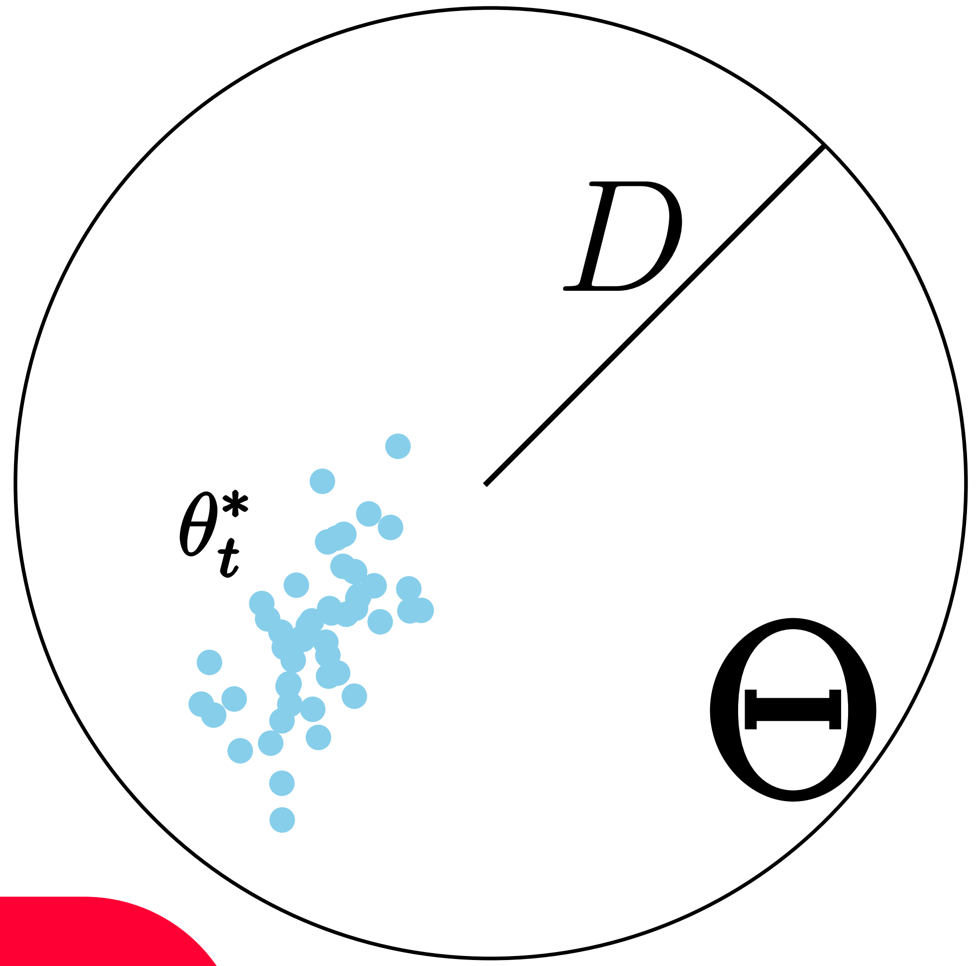
Matching lower-bound: $R = \Omega\left(D\sqrt{m}\right)$
(for any algorithm)

[Abernethy-Bartlett-Rakhlin-Tewari]

Key Question:
can we do better using
multi-task information?

Single-Task Regret

Training Data	Hypothesis Class	Loss Function
$(x_1, y_1), \ldots, (x_m, y_m)$	$\{f_\theta : \mathscr{X} \mapsto \mathscr{Y} : \theta \in \Theta \subset \mathbb{R}^d\}$	$\ell_i(\theta) = L(f_\theta(x_i), y_i)$



Key Question:
can we do better using
on-average across tasks?

Online Gradient Descent (OGD)

Size of Action Space: $D = \text{radius}(\Theta)$

OGD upper-bound: $R = \mathcal{O}(D\sqrt{m})$

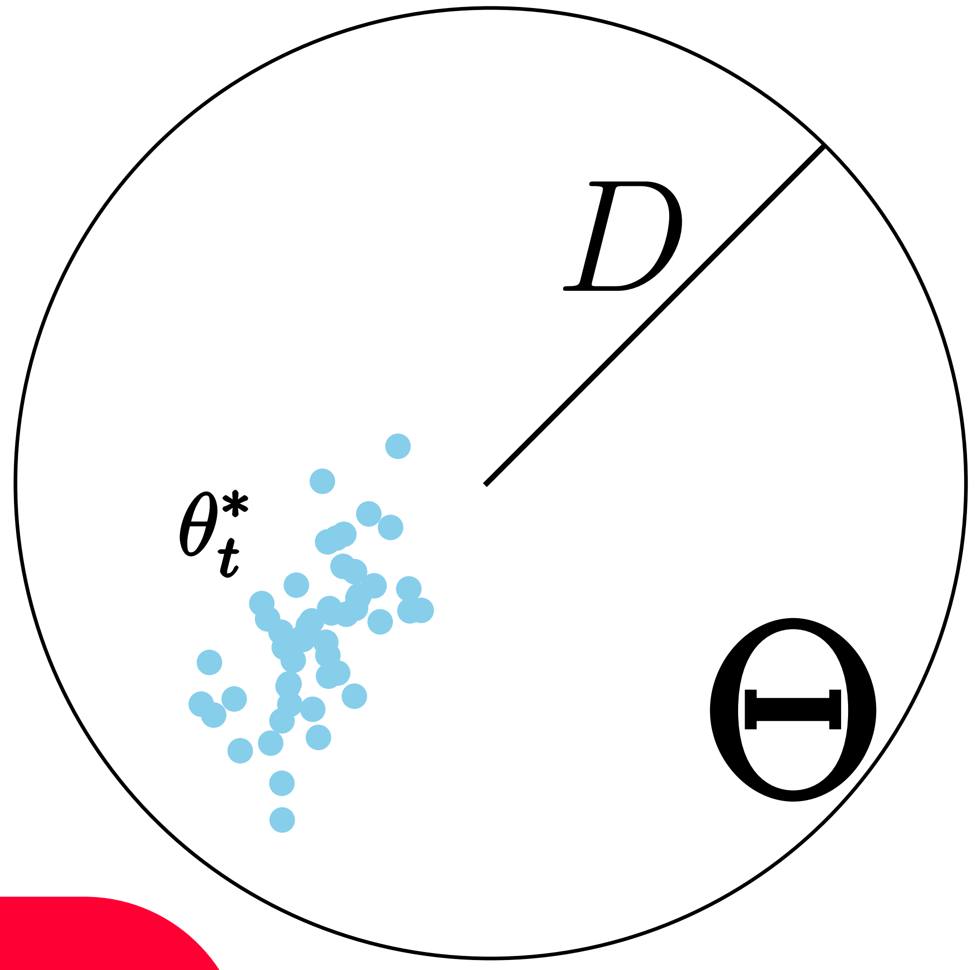
Matching lower-bound: $R = \Omega\left(D\sqrt{m}\right)$
(for any algorithm)

[Abernethy-Bartlett-Rakhlin-Tewari]

Single-Task Regret

Training Data	Hypothesis Class	Loss Function
$(x_1, y_1), \dots, (x_m, y_m)$	$\{f_\theta : \mathcal{X} \mapsto \mathcal{Y} : \theta \in \Theta \subset \mathbb{R}^d\}$	$\ell_i(\theta) = L(f_\theta(x_i), y_i)$

Learn an initialization sequentially from previous t tasks



Key Question:
can we do better using **on-average across tasks?**

Online Gradient Descent (OGD)

Size of Action Space: $D = \text{radius}(\Theta)$

OGD upper-bound: $R = \mathcal{O}(D\sqrt{m})$

Matching lower-bound: $R = \Omega\left(D\sqrt{m}\right)$
(for any algorithm)

[Abernethy-Bartlett-Rakhlin-Tewari]

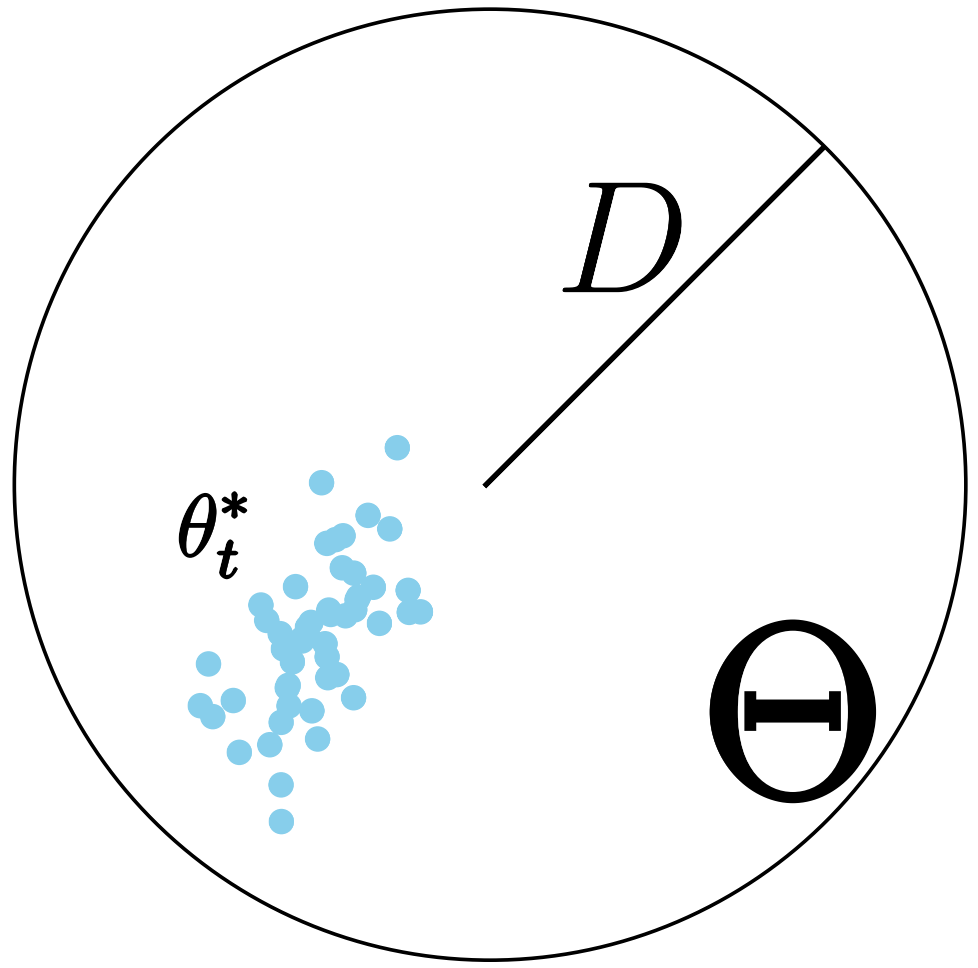
Average Regret and Task Similarity

Training Data	Hypothesis Class	Loss Function
$(x_{1,1}, y_{1,1}), \ldots, (x_{T,m}, y_{T,m})$	$\{f_\theta : \mathcal{X} \mapsto \mathcal{Y} : \theta \in \Theta \subset \mathbb{R}^d\}$	$\ell_{t,i}(\theta) = L(f_\theta(x_{t,i}), y_{t,i})$

Average Regret and Task Similarity

Training Data	Hypothesis Class	Loss Function
$(x_{1,1}, y_{1,1}), \ldots, (x_{T,m}, y_{T,m})$	$\{f_\theta : \mathscr{X} \mapsto \mathscr{Y} : \theta \in \Theta \subset \mathbb{R}^d\}$	$\ell_{t,i}(\theta) = L(f_\theta(x_{t,i}), y_{t,i})$

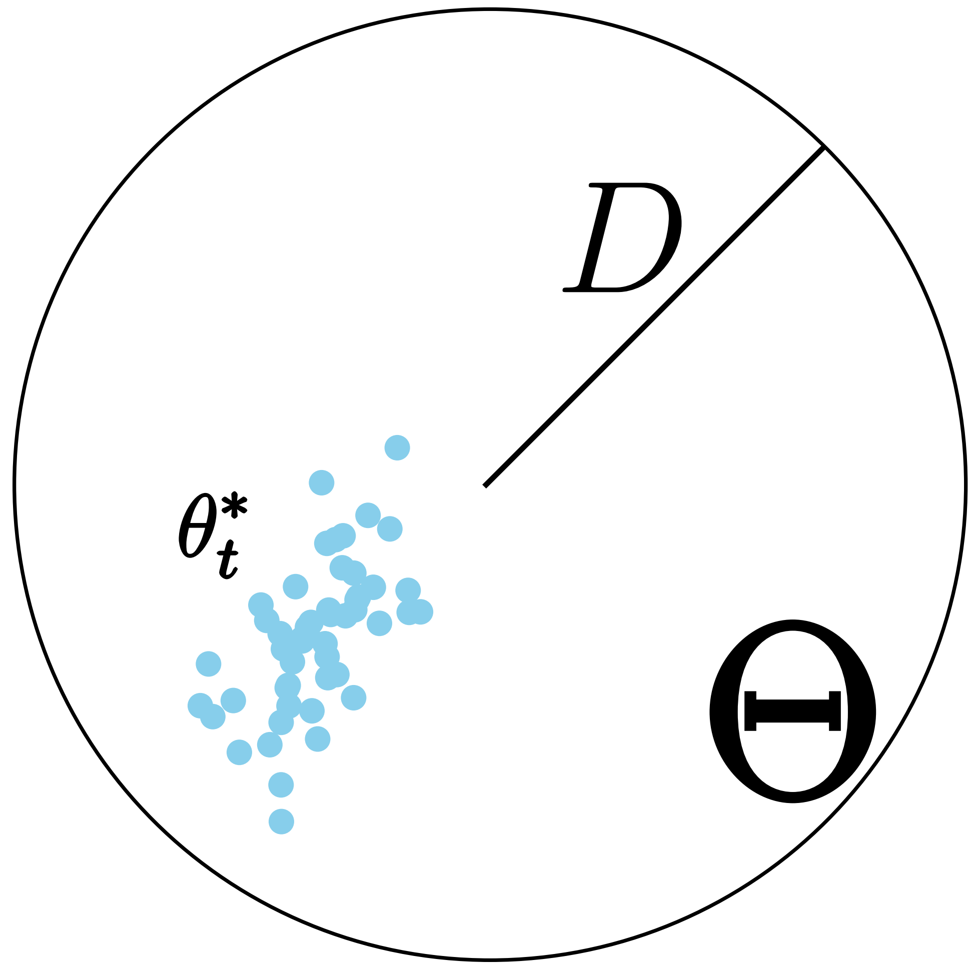
Average Regret: $\bar{R} = \frac{1}{T}\sum_{t=1}^{T} R_t = \frac{1}{T}\sum_{t=1}^{T}\sum_{i=1}^{m} \ell_{t,i}(\theta_{t,i}) - \ell_{t,i}(\theta_t^*)$



Average Regret and Task Similarity

Training Data	Hypothesis Class	Loss Function
$(x_{1,1}, y_{1,1}), \ldots, (x_{T,m}, y_{T,m})$	$\{f_\theta : \mathcal{X} \mapsto \mathcal{Y} : \theta \in \Theta \subset \mathbb{R}^d\}$	$\ell_{t,i}(\theta) = L(f_\theta(x_{t,i}), y_{t,i})$

Average Regret: $\bar{R} = \frac{1}{T}\sum_{t=1}^{T} R_t = \frac{1}{T}\sum_{t=1}^{T}\sum_{i=1}^{m} \ell_{t,i}(\theta_{t,i}) - \ell_{t,i}(\theta_t^*)$



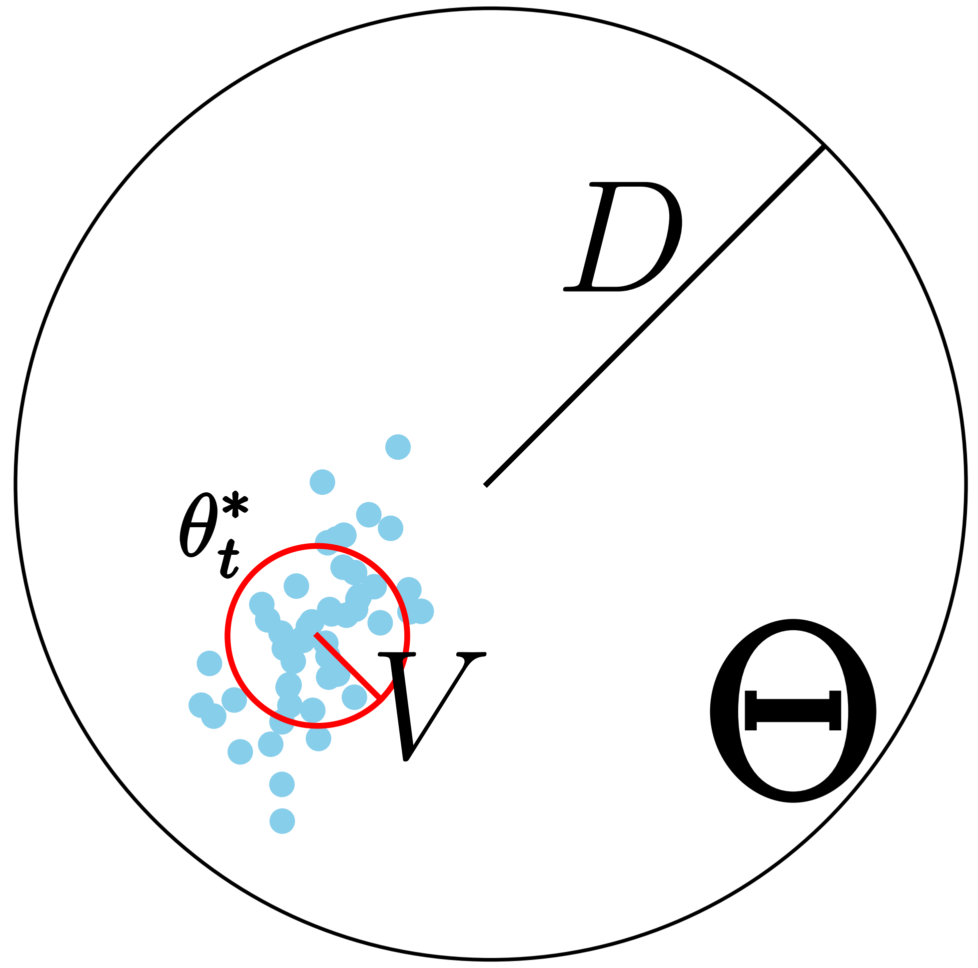
Task Similarity: $V^2 = \min_{\phi \in \Theta} \frac{1}{T}\sum_{t=1}^{T} \|\theta_t^* - \phi\|_2^2$

Average Regret and Task Similarity

Training Data	Hypothesis Class	Loss Function
$(x_{1,1}, y_{1,1}), \ldots, (x_{T,m}, y_{T,m})$	$\{f_\theta : \mathcal{X} \mapsto \mathcal{Y} : \theta \in \Theta \subset \mathbb{R}^d\}$	$\ell_{t,i}(\theta) = L(f_\theta(x_{t,i}), y_{t,i})$

Average Regret: $\bar{R} = \frac{1}{T}\sum_{t=1}^{T} R_t = \frac{1}{T}\sum_{t=1}^{T}\sum_{i=1}^{m} \ell_{t,i}(\theta_{t,i}) - \ell_{t,i}(\theta_t^*)$

Task Similarity: $V^2 = \min_{\phi\in\Theta} \frac{1}{T}\sum_{t=1}^{T} \|\theta_t^* - \phi\|_2^2$

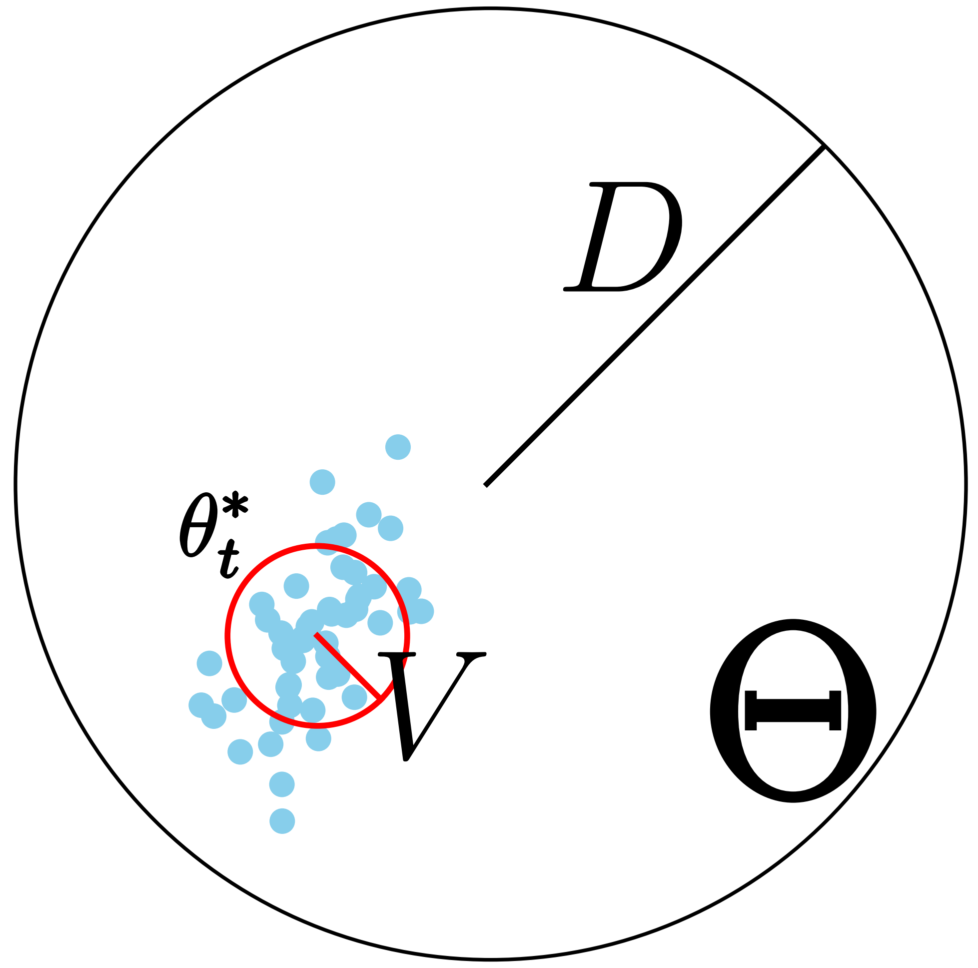


Average Regret and Task Similarity

Training Data	Hypothesis Class	Loss Function
$(x_{1,1}, y_{1,1}), \ldots, (x_{T,m}, y_{T,m})$	$\{f_\theta : \mathscr{X} \mapsto \mathscr{Y} : \theta \in \Theta \subset \mathbb{R}^d\}$	$\ell_{t,i}(\theta) = L(f_\theta(x_{t,i}), y_{t,i})$

Average Regret: $\bar{R} = \frac{1}{T}\sum_{t=1}^{T} R_t = \frac{1}{T}\sum_{t=1}^{T}\sum_{i=1}^{m} \ell_{t,i}(\theta_{t,i}) - \ell_{t,i}(\theta_t^*)$

Task Similarity: $V^2 = \min_{\phi\in\Theta} \frac{1}{T}\sum_{t=1}^{T} \|\theta_t^* - \phi\|_2^2$



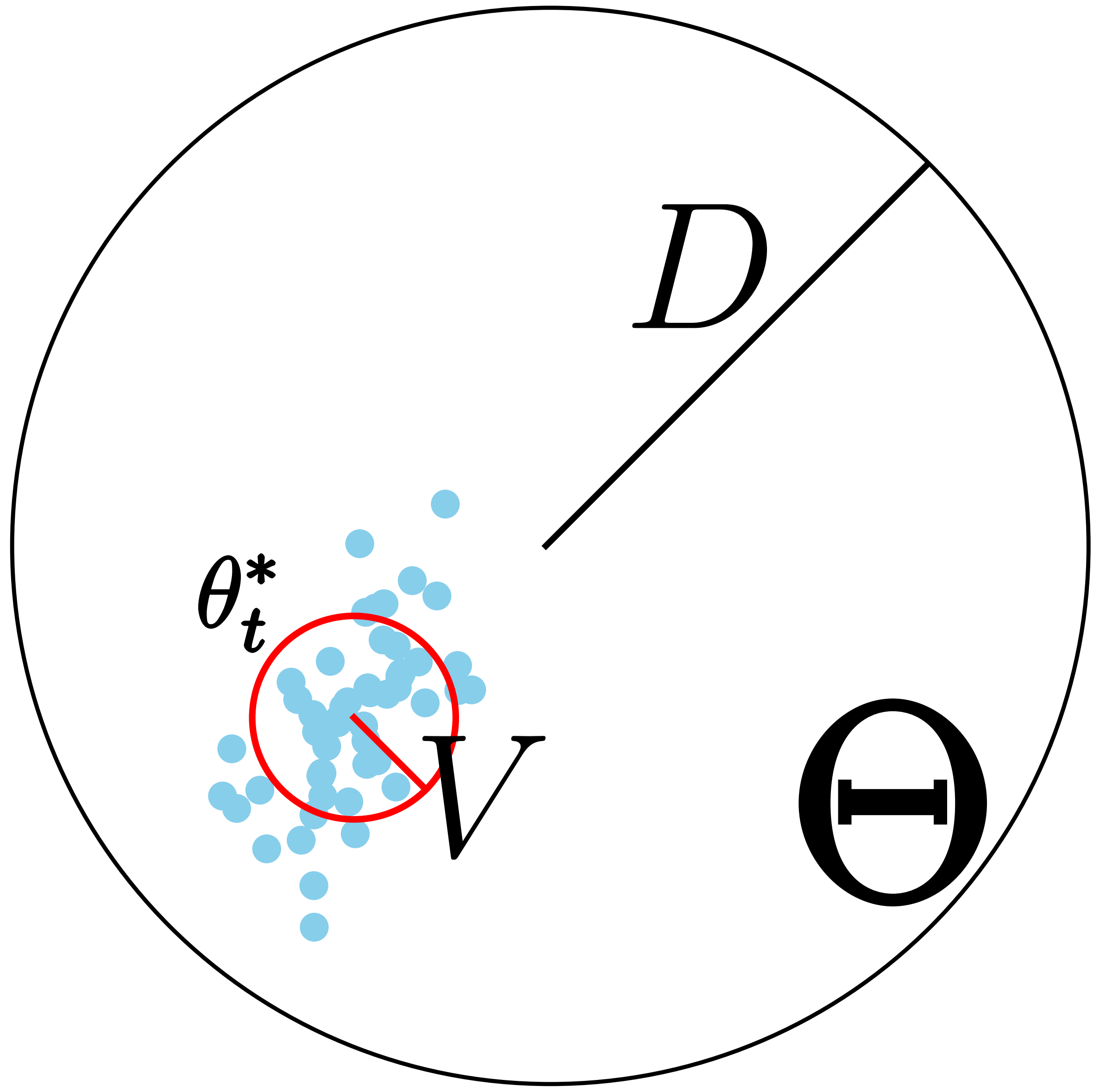
V is small when optimal parameters are close together

ARUBA: An Illustrative Result

Our Guarantee: $\bar{R} = \mathcal{O}\left(V + \frac{\log T}{VT}\right)\sqrt{m}$

Single-Task Lower Bound: $R_t = \Omega\left(D\sqrt{m}\right)$

Multi-Task Lower Bound: $\bar{R} = \Omega\left(V\sqrt{m}\right)$



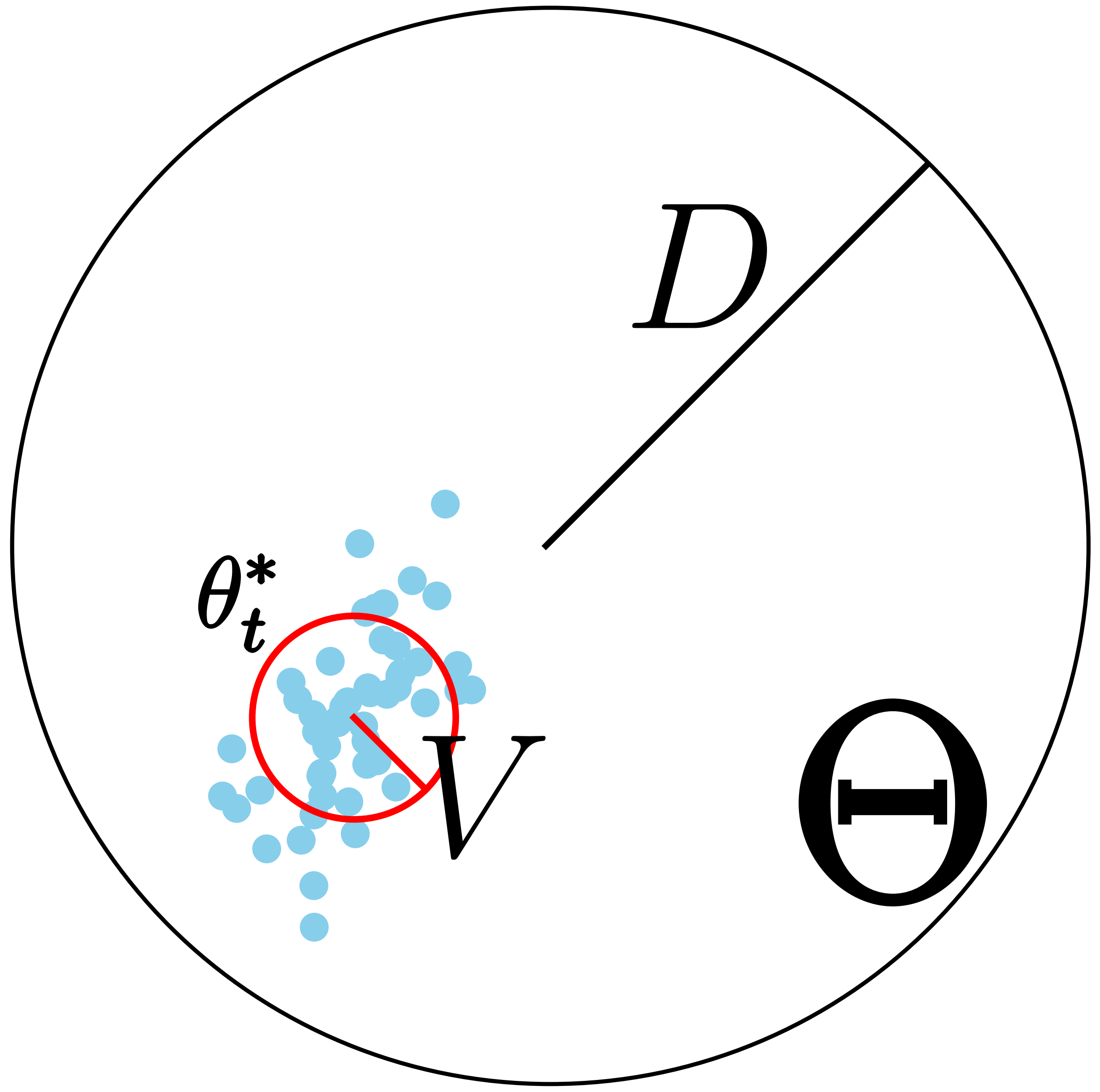
Task Similarity V

ARUBA: An Illustrative Result

Our Guarantee: $\bar{R} = \mathcal{O}\left(V + \frac{\log T}{VT}\right)\sqrt{m}$

Single-Task Lower Bound: $R_t = \Omega\left(D\sqrt{m}\right)$

Multi-Task Lower Bound: $\bar{R} = \Omega\left(V\sqrt{m}\right)$



Task Similarity V

When **optimal task parameters are close together**, meta-learning yields much **better average performance**

Recall: generic gradient-based algorithm (Reptile)

for task $t = 1,\ldots,T$

sample task $\mathscr{D}_t$

$\hat{\theta}_t \leftarrow$ **within-task SGD**$(\mathscr{D}_t, \phi_t)$

update ϕ_{t+1} **using** $\hat{\theta}_t$

Recall: generic gradient-based algorithm (Reptile)

for task $t = 1,\ldots,T$

sample task $\mathcal{D}_t$

$\hat{\theta}_t \leftarrow$ **within-task OGD**$(\mathcal{D}_t, \phi_t)$

update ϕ_{t+1} **using** $\hat{\theta}_t$

replace SGD by
online gradient descent (OGD)

Recall: generic gradient-based algorithm (Reptile)

for task $t = 1,\ldots,T$

sample task $\mathscr{D}_t$

within-task OGD$(\mathscr{D}_t, \phi_t)$

update ϕ_{t+1} **using** θ_t^*

replace last iterate by optimum-in-hindsight

Recall: generic gradient-based algorithm (Reptile)

for task $t = 1, \ldots, T$

sample task $\mathscr{D}_t$

within-task OGD$(\mathscr{D}_t, \phi_t)$

update ϕ_{t+1} **using** θ_t^*

replace last iterate by optimum-in-hindsight

(assumes oracle access to last iterate *after* task completion)

Recall: generic gradient-based algorithm (Reptile)

for task $t = 1,\ldots,T$

sample task $\mathscr{D}_t$

within-task OGD$(\mathscr{D}_t, \phi_t)$

update ϕ_{t+1} **using** θ_t^*

replace last iterate by optimum-in-hindsight

(assumes oracle access to last iterate *after* task completion)

(can be relaxed under a non-degeneracy assumption)

Recall: generic gradient-based algorithm (Reptile)

Goal: set ϕ_t to get low average regret across tasks.

for task $t = 1,\ldots,T$

sample task $\mathscr{D}_t$

within-task OGD$(\mathscr{D}_t, \phi_t)$

update ϕ_{t+1} **using** θ_t^*

$$\frac{1}{T}\sum_{t=1}^{T} R_t$$

Recall: generic gradient-based algorithm (Reptile)

Goal: set ϕ_t to get low average regret across tasks.

for task $t = 1, \ldots, T$

sample task $\mathscr{D}_t$

within-task OGD$(\mathscr{D}_t, \phi_t)$

update ϕ_{t+1} **using** θ_t^*

$$\frac{1}{T}\sum_{t=1}^{T} R_t \leq \frac{1}{T}\sum_{t=1}^{T} U_t(\phi_t)$$

regret-upper-bound

Key Idea:
Use online learning to optimize a sequence of OGD regret bounds.

ARUBA: Key Observation

Single-task regret guarantees are often **nice** and **data-dependent** functions of the algorithm parameters.

ARUBA: Key Observation

Single-task regret guarantees are often **nice** and **data-dependent** functions of the algorithm parameters.

for OGD$(\mathscr{D}_t, \phi)$:

$$R_t = \sum_{i=1}^{m} \ell_{t,i}(\theta) - \ell_{t,i}(\theta_t^*) = \mathcal{O}(D\sqrt{m})$$

ARUBA: Key Observation

Single-task regret guarantees are often **nice** and **data-dependent** functions of the algorithm parameters.

for OGD$(\mathscr{D}_t, \phi)$:

regret-upper-bound

$$R_t = \sum_{i=1}^{m} \ell_{t,i}(\theta) - \ell_{t,i}(\theta_t^*) \leq U_t(\phi) = \frac{\|\theta_t^* - \phi\|_2^2}{2\eta} + \eta m$$

set $\eta = \frac{D}{\sqrt{m}}$ **to get** $\mathcal{O}(D\sqrt{m})$

ARUBA: Key Observation

Single-task regret guarantees are often **nice** and **data-dependent** functions of the algorithm parameters.

for OGD$(\mathscr{D}_t, \phi)$:

regret-upper-bound

$$R_t = \sum_{i=1}^{m} \ell_{t,i}(\theta) - \ell_{t,i}(\theta_t^*) \leq U_t(\phi) = \frac{\|\theta_t^* - \phi\|_2^2}{2\eta} + \eta m$$

Average Regret-Upper-Bound Analysis:

reduce the analysis of meta-learning algorithms to online learning over within-task regret-upper-bounds

Updating the initialization using online learning

Goal: set ϕ_t to get low average regret across tasks.

for task $t = 1, \ldots, T$

sample task $\mathscr{D}_t$

within-task OGD$(\mathscr{D}_t, \phi_t)$

update ϕ_{t+1} **using** θ_t^*

$$\frac{1}{T}\sum_{t=1}^{T} R_t \le \frac{1}{T}\sum_{t=1}^{T} U_t(\phi_t)$$

regret-upper-bound

Key Idea:
Use online learning to optimize a sequence of OGD regret bounds.

Updating the initialization using online learning

Goal: set ϕ_t to get low average regret across tasks.

for task $t = 1,\ldots,T$

sample task $\mathscr{D}_t$

within-task OGD$(\mathscr{D}_t, \phi_t)$

update ϕ_{t+1} **using** θ_t^*

$$\frac{1}{T}\sum_{t=1}^{T} R_t \leq \frac{1}{T}\sum_{t=1}^{T} U_t(\phi_t)$$

$$U_t(\phi_t) = \frac{\|\theta_t^* - \phi_t\|_2^2}{2\eta} + \eta m$$

Key Idea:
Use online learning to optimize a sequence of OGD regret bounds.

Updating the initialization using online learning

Goal: set ϕ_t to get low average regret across tasks.

for task $t = 1, \ldots, T$

sample task $\mathscr{D}_t$

within-task OGD$(\mathscr{D}_t, \phi_t)$

update ϕ_{t+1} **using** θ_t^*

$$U_t(\phi_t) = \frac{\|\theta_t^* - \phi_t\|_2^2}{2\eta} + \eta m$$

Key Idea: apply OGD

Updating the initialization using online learning

Goal: set ϕ_t to get low average regret across tasks.

for task $t = 1, \ldots, T$

sample task $\mathscr{D}_t$

within-task OGD$(\mathscr{D}_t, \phi_t)$

$$\phi_{t+1} \leftarrow \phi_t - \alpha\eta \nabla U_t(\phi_t)$$

learning rate $\alpha\eta > 0$

$$U_t(\phi_t) = \frac{\|\theta_t^* - \phi_t\|_2^2}{2\eta} + \eta m$$

Key Idea: apply OGD

Updating the initialization using online learning

Goal: set ϕ_t to get low average regret across tasks.

for task $t = 1, \dots, T$

sample task $\mathscr{D}_t$

within-task OGD$(\mathscr{D}_t, \phi_t)$

$$\phi_{t+1} \leftarrow \phi_t - \alpha(\phi_t - \theta_t^*)$$

$$U_t(\phi_t) = \frac{\|\theta_t^* - \phi_t\|_2^2}{2\eta} + \eta m$$

Key Idea: apply OGD

Updating the initialization using online learning

Goal: set ϕ_t to get low average regret across tasks.

for task $t = 1, \ldots, T$

sample task $\mathscr{D}_t$

within-task OGD$(\mathscr{D}_t, \phi_t)$

$$\phi_{t+1} \leftarrow (1-\alpha)\phi_t + \alpha\theta_t^*$$

$$U_t(\phi_t) = \frac{\|\theta_t^* - \phi_t\|_2^2}{2\eta} + \eta m$$

Key Idea: apply OGD

Updating the initialization using online learning

Goal: set ϕ_t to get low average regret across tasks.

for task $t = 1, \ldots, T$

sample task $\mathscr{D}_t$

within-task OGD$(\mathscr{D}_t, \phi_t)$

$$\phi_{t+1} \leftarrow (1-\alpha)\phi_t + \alpha\theta_t^*$$

(almost) same update as Reptile!

$$U_t(\phi_t) = \frac{\|\theta_t^* - \phi_t\|_2^2}{2\eta} + \eta m$$

Key Idea: apply OGD

Updating the initialization using online learning

Goal: set ϕ_t to get low average regret across tasks.

for task $t = 1, \ldots, T$

sample task $\mathscr{D}_t$

within-task OGD$(\mathscr{D}_t, \phi_t)$

$$\phi_{t+1} \leftarrow (1-\alpha)\phi_t + \alpha\theta_t^*$$

(almost) same update as Reptile!

$\frac{1}{\eta}$-strongly-convex

$$U_t(\phi_t) = \frac{\|\theta_t^* - \phi_t\|_2^2}{2\eta} + \eta m$$

Key Idea: apply OGD

Regret guarantee:

$$\sum_{t=1}^{T} U_t(\phi_t) - \min_{\phi \in \Theta} \sum_{t=1}^{T} U_t(\phi) \le \mathcal{O}\left(\frac{\log T}{\eta}\right)$$

3 Key Steps of ARUBA Framework

1. Control average within-task performance using average regret-upper-bound

Step 1: Substitute Regret-Upper-Bound

$$\frac{1}{T}\sum_{t=1}^{T} R_t \leq \frac{1}{T}\sum_{t=1}^{T} U_t(\phi_t)$$

3 Key Steps of ARUBA Framework

1. Control average within-task performance using average regret-upper-bound

Step 1: Substitute Regret-Upper-Bound

Addition/Subtraction

$$\frac{1}{T}\sum_{t=1}^{T} R_t \leq \frac{1}{T}\sum_{t=1}^{T} U_t(\phi_t) = \frac{1}{T}\left(\sum_{t=1}^{T} U_t(\phi_t) - \min_{\phi\in\Theta}\sum_{t=1}^{T} U_t(\phi)\right) + \min_{\phi\in\Theta}\frac{1}{T}\sum_{t=1}^{T} U_t(\phi)$$

3 Key Steps of ARUBA Framework

1. Control average within-task performance using average regret-upper-bound
2. Use across-task OGD to set initialization so that average upper bound is small

Step 1: Substitute Regret-Upper-Bound

Addition/Subtraction

$$\frac{1}{T}\sum_{t=1}^{T} R_t \leq \frac{1}{T}\sum_{t=1}^{T} U_t(\phi_t) = \frac{1}{T}\left(\sum_{t=1}^{T} U_t(\phi_t) - \min_{\phi\in\Theta}\sum_{t=1}^{T} U_t(\phi)\right) + \min_{\phi\in\Theta}\frac{1}{T}\sum_{t=1}^{T} U_t(\phi)$$

Step 2: Across-task OGD

$$= \frac{1}{T}\left(\sum_{t=1}^{T}\frac{\|\theta_t^* - \phi_t\|_2^2}{2\eta} - \min_{\phi\in\Theta}\frac{\|\theta_t^* - \phi\|_2^2}{2\eta}\right) + \quad \ldots$$

3 Key Steps of ARUBA Framework

1. Control average within-task performance using average regret-upper-bound
2. Use across-task OGD to set initialization so that average upper bound is small

Step 1: Substitute Regret-Upper-Bound

Addition/Subtraction

$$\frac{1}{T}\sum_{t=1}^{T} R_t \leq \frac{1}{T}\sum_{t=1}^{T} U_t(\phi_t) = \frac{1}{T}\left(\sum_{t=1}^{T} U_t(\phi_t) - \min_{\phi\in\Theta}\sum_{t=1}^{T} U_t(\phi)\right) + \min_{\phi\in\Theta}\frac{1}{T}\sum_{t=1}^{T} U_t(\phi)$$

regret of OGD over T

$\frac{1}{\eta}$ – **strongly-convex functions** $+ \ldots$

3 Key Steps of ARUBA Framework

1. Control average within-task performance using average regret-upper-bound
2. Use across-task OGD to set initialization so that average upper bound is small

Step 1: Substitute Regret-Upper-Bound

Addition/Subtraction

$$\frac{1}{T}\sum_{t=1}^{T} R_t \leq \frac{1}{T}\sum_{t=1}^{T} U_t(\phi_t) = \frac{1}{T}\left(\sum_{t=1}^{T} U_t(\phi_t) - \min_{\phi\in\Theta}\sum_{t=1}^{T} U_t(\phi)\right) + \min_{\phi\in\Theta}\frac{1}{T}\sum_{t=1}^{T} U_t(\phi)$$

Step 2: Across-task OGD

$$= \frac{1}{T}\left(\sum_{t=1}^{T} \frac{\|\theta_t^* - \phi_t\|_2^2}{2\eta} - \min_{\phi\in\Theta}\frac{\|\theta_t^* - \phi\|_2^2}{2\eta}\right) + \dots$$

$$= \mathcal{O}\left(\frac{\log T}{\eta T}\right) + \dots$$

3 Key Steps of ARUBA Framework

1. Control average within-task performance using average regret-upper-bound
2. Use across-task OGD to set initialization so that average upper bound is small
3. Analyze impact of task-relatedness on resulting bound

Step 1: Substitute Regret-Upper-Bound

Addition/Subtraction

$$\frac{1}{T}\sum_{t=1}^{T} R_t \le \frac{1}{T}\sum_{t=1}^{T} U_t(\phi_t) = \frac{1}{T}\left(\sum_{t=1}^{T} U_t(\phi_t) - \min_{\phi\in\Theta}\sum_{t=1}^{T} U_t(\phi)\right) + \min_{\phi\in\Theta}\frac{1}{T}\sum_{t=1}^{T} U_t(\phi)$$

Step 2: Across-task OGD

Step 3: Impact of Task Relatedness

$$= \frac{1}{T}\left(\sum_{t=1}^{T}\frac{\|\theta_t^* - \phi_t\|_2^2}{2\eta} - \min_{\phi\in\Theta}\frac{\|\theta_t^* - \phi\|_2^2}{2\eta}\right) + \min_{\phi\in\Theta}\frac{1}{T}\sum_{t=1}^{T}\frac{\|\theta_t^* - \phi\|_2^2}{2\eta} + \eta m$$

$$= \mathcal{O}\left(\frac{\log T}{\eta T}\right) + \dots$$

3 Key Steps of ARUBA Framework

1. Control average within-task performance using average regret-upper-bound
2. Use across-task OGD to set initialization so that average upper bound is small
3. Analyze impact of task-relatedness on resulting bound

Step 1: Substitute Regret-Upper-Bound

Addition/Subtraction

$$\frac{1}{T}\sum_{t=1}^{T} R_t \leq \frac{1}{T}\sum_{t=1}^{T} U_t(\phi_t) = \frac{1}{T}\left(\sum_{t=1}^{T} U_t(\phi_t) - \min_{\phi\in\Theta}\sum_{t=1}^{T} U_t(\phi)\right) + \min_{\phi\in\Theta}\frac{1}{T}\sum_{t=1}^{T} U_t(\phi)$$

Step 2: Across-task OGD

Step 3: Impact of Task Relatedness

$$= \frac{1}{T}\left(\sum_{t=1}^{T} \frac{\|\theta_t^* - \phi_t\|_2^2}{2\eta} - \min_{\phi\in\Theta}\frac{\|\theta_t^* - \phi\|_2^2}{2\eta}\right) + \boxed{\min_{\phi\in\Theta}\frac{1}{T}\sum_{t=1}^{T}\frac{\|\theta_t^* - \phi\|_2^2}{2\eta}} + \eta m$$

$$= \mathcal{O}\left(\frac{\log T}{\eta T}\right) + \dots$$

3 Key Steps of ARUBA Framework

1. Control average within-task performance using average regret-upper-bound
2. Use across-task OGD to set initialization so that average upper bound is small
3. Analyze impact of task-relatedness on resulting bound

Step 1: Substitute Regret-Upper-Bound

Addition/Subtraction

$$\frac{1}{T}\sum_{t=1}^{T} R_t \leq \frac{1}{T}\sum_{t=1}^{T} U_t(\phi_t) = \frac{1}{T}\left(\sum_{t=1}^{T} U_t(\phi_t) - \min_{\phi\in\Theta}\sum_{t=1}^{T} U_t(\phi)\right) + \min_{\phi\in\Theta}\frac{1}{T}\sum_{t=1}^{T} U_t(\phi)$$

Step 2: Across-task OGD

$$= \frac{1}{T}\left(\sum_{t=1}^{T}\frac{\|\theta_t^* - \phi_t\|_2^2}{2\eta} - \min_{\phi\in\Theta}\frac{\|\theta_t^* - \phi\|_2^2}{2\eta}\right) +$$

definition of task-similarity

$$V^2 = \min_{\phi\in\Theta}\sum_{t=1}^{T}\|\theta_t^* - \phi\|_2^2$$

$$= \mathcal{O}\left(\frac{\log T}{\eta T}\right) + \dots$$

3 Key Steps of ARUBA Framework

1. Control average within-task performance using average regret-upper-bound
2. Use across-task OGD to set initialization so that average upper bound is small
3. Analyze impact of task-relatedness on resulting bound

Step 1: Substitute Regret-Upper-Bound

Addition/Subtraction

$$\frac{1}{T}\sum_{t=1}^{T} R_t \leq \frac{1}{T}\sum_{t=1}^{T} U_t(\phi_t) = \frac{1}{T}\left(\sum_{t=1}^{T} U_t(\phi_t) - \min_{\phi\in\Theta}\sum_{t=1}^{T} U_t(\phi)\right) + \min_{\phi\in\Theta}\frac{1}{T}\sum_{t=1}^{T} U_t(\phi)$$

Step 2: Across-task OGD

Step 3: Impact of Task Relatedness

$$= \frac{1}{T}\left(\sum_{t=1}^{T}\frac{\|\theta_t^* - \phi_t\|_2^2}{2\eta} - \min_{\phi\in\Theta}\frac{\|\theta_t^* - \phi\|_2^2}{2\eta}\right) + \min_{\phi\in\Theta}\frac{1}{T}\sum_{t=1}^{T}\frac{\|\theta_t^* - \phi\|_2^2}{2\eta} + \eta m$$

$$= \mathcal{O}\left(\frac{\log T}{\eta T}\right) + \mathcal{O}\left(\frac{V^2}{\eta} + \eta m\right)$$

3 Key Steps of ARUBA Framework

1. Control average within-task performance using average regret-upper-bound
2. Use across-task OGD to set initialization so that average upper bound is small
3. Analyze impact of task-relatedness on resulting bound

Step 1: Substitute Regret-Upper-Bound

Addition/Subtraction

$$\frac{1}{T}\sum_{t=1}^{T} R_t \le \frac{1}{T}\sum_{t=1}^{T} U_t(\phi_t) = \frac{1}{T}\left(\sum_{t=1}^{T} U_t(\phi_t) - \min_{\phi\in\Theta}\sum_{t=1}^{T} U_t(\phi)\right) + \min_{\phi\in\Theta}\frac{1}{T}\sum_{t=1}^{T} U_t(\phi)$$

Step 2: Across-task OGD

Step 3: Impact of Task Relatedness

$$= \frac{1}{T}\left(\sum_{t=1}^{T}\frac{\|\theta_t^* - \phi_t\|_2^2}{2\eta} - \min_{\phi\in\Theta}\frac{\|\theta_t^* - \phi\|_2^2}{2\eta}\right) + \min_{\phi\in\Theta}\frac{1}{T}\sum_{t=1}^{T}\frac{\|\theta_t^* - \phi\|_2^2}{2\eta} + \eta m$$

substitute $\eta = V/\sqrt{m}$

$$= \mathcal{O}\left(\frac{\log T}{\eta T}\right) + \mathcal{O}\left(\frac{V^2}{\eta} + \eta m\right)$$

3 Key Steps of ARUBA Framework

1. Control average within-task performance using average regret-upper-bound
2. Use across-task OGD to set initialization so that average upper bound is small
3. Analyze impact of task-relatedness on resulting bound

Step 1: Substitute Regret-Upper-Bound

Addition/Subtraction

$$\frac{1}{T}\sum_{t=1}^{T} R_t \le \frac{1}{T}\sum_{t=1}^{T} U_t(\phi_t) = \frac{1}{T}\left(\sum_{t=1}^{T} U_t(\phi_t) - \min_{\phi\in\Theta}\sum_{t=1}^{T} U_t(\phi)\right) + \min_{\phi\in\Theta}\frac{1}{T}\sum_{t=1}^{T} U_t(\phi)$$

Step 2: Across-task OGD

Step 3: Impact of Task Relatedness

$$= \frac{1}{T}\left(\sum_{t=1}^{T}\frac{\|\theta_t^* - \phi_t\|_2^2}{2\eta} - \min_{\phi\in\Theta}\frac{\|\theta_t^* - \phi\|_2^2}{2\eta}\right) + \min_{\phi\in\Theta}\frac{1}{T}\sum_{t=1}^{T}\frac{\|\theta_t^* - \phi\|_2^2}{2\eta} + \eta m$$

substitute $\eta = V/\sqrt{m}$

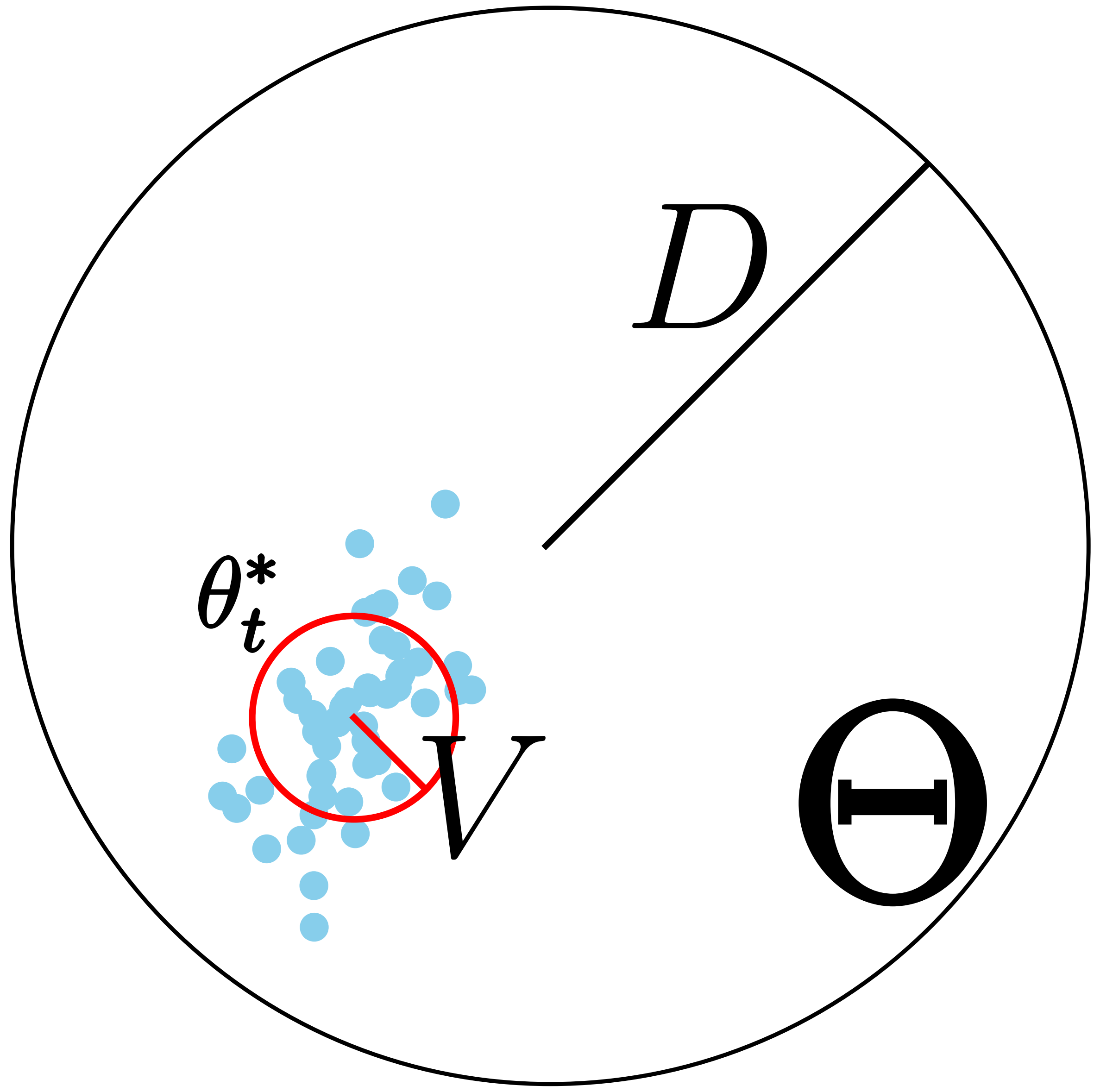
$$= \mathcal{O}\left(\frac{\log T}{VT}\right)\sqrt{m} + \mathcal{O}\left(V\sqrt{m}\right)$$

Recap: what have we achieved?

Our Guarantee: $\bar{R} = \mathcal{O}\left(V + \frac{\log T}{VT}\right)\sqrt{m}$

Single-Task Lower Bound: $R_t = \Omega\left(D\sqrt{m}\right)$

Multi-Task Lower Bound: $\bar{R} = \Omega\left(V\sqrt{m}\right)$



Task Similarity V

When **optimal task parameters are close together**, meta-learning yields much **better average performance**

Our results in context

Online learning of multi-task representations:

- Linear representations using online Frank-Wolfe or matrix multiplicative weights [Bullins-Hazan-Kalai-Livni, ALT 2019]

Our results in context

Online learning of multi-task representations:

- Linear representations using online Frank-Wolfe or matrix multiplicative weights [Bullins-Hazan-Kalai-Livni, ALT 2019]

Online meta-initialization learning:

- Average regret bounds for online gradient descent [**K**-Balcan-Talwalkar, ICML 2019]
- Excess transfer risk bounds on regularized SGD via online-to-batch conversion [Denevi-Ciliberto-Grazzi-Pontil, ICML 2019]
- Learnability of MAML via Follow-the-Leader [Finn-Rajeswaran-Kakade-Levine, ICML 2019]

Our results in context

Online learning of multi-task representations:

- Linear representations using online Frank-Wolfe or matrix multiplicative weights [Bullins-Hazan-Kalai-Livni, ALT 2019]

Online meta-initialization learning:

- Average regret bounds for online gradient descent [**K**-Balcan-Talwalkar, ICML 2019]
- Excess transfer risk bounds on regularized SGD via online-to-batch conversion [Denevi-Ciliberto-Grazzi-Pontil, ICML 2019]
- Learnability of MAML via Follow-the-Leader [Finn-Rajeswaran-Kakade-Levine, ICML 2019]

General online frameworks for learning parameterized algorithms:

- ARUBA [**K**-Balcan-Talwalkar, NeurIPS 2019]
- Primal-dual approach [Denevi-Ciliberto-Grazzi-Pontil, NeurIPS 2019]

Our results in context

Online learning of multi-task representations:

- Linear representations using online Frank-Wolfe or matrix multiplicative weights [Bullins-Hazan-Kalai-Livni, ALT 2019]

Online meta-initialization learning:

- Average regret bounds for online gradient descent [**K**-Balcan-Talwalkar, ICML 2019]
- Excess transfer risk bounds on regularized SGD via online-to-batch conversion [Denevi-Ciliberto-Grazzi-Pontil, ICML 2019]
- Learnability of MAML via Follow-the-Leader [Finn-Rajeswaran-Kakade-Levine, ICML 2019]

General online frameworks for learning parameterized algorithms:

- ARUBA [**K**-Balcan-Talwalkar, NeurIPS 2019]
- Primal-dual approach [Denevi-Ciliberto-Grazzi-Pontil, NeurIPS 2019]

Non-convex stochastic optimization for meta-initialization:

- Stationary-point convergence for MAML [Fallah-Mokhtari-Ozdaglar, 2019]
- Stationary-point convergence of proximal update [Zhou-Yuan-Xu-Yan-Feng, NeurIPS 2019]

What else can we get by applying ARUBA?

Adaptivity

- Learn any base-learner parameter from data
- e.g., **improved training algorithms** for learning ϕ and η simultaneously

What else can we get by applying ARUBA?

Adaptivity

- Learn any base-learner parameter from data
- e.g., **improved training algorithms** for learning ϕ and η simultaneously

Generality

- Low-dynamic-regret algorithms for **changing task-environments**
- Stronger online-to-batch conversions for **faster statistical rates**
- Specialized within-task algorithms, e.g., satisfying **privacy guarantees**

ARUBA Framework

Applications

- Adaptivity for improved few-shot learning
- Federated learning & private meta-learning

Is learning an initialization good enough?

per-coordinate
learning rate

gradient on sample i
from task t

$$\theta_{t,i+1} = \theta_{t,i} - \eta_{t,i} \odot \nabla_{t,i}$$

Is learning an initialization good enough?

per-coordinate learning rate

gradient on sample i from task t

$$\theta_{t,i+1} = \theta_{t,i} - \eta_{t,i} \odot \nabla_{t,i}$$

When is pre-conditioned online gradient descent useful?

Is learning an initialization good enough?

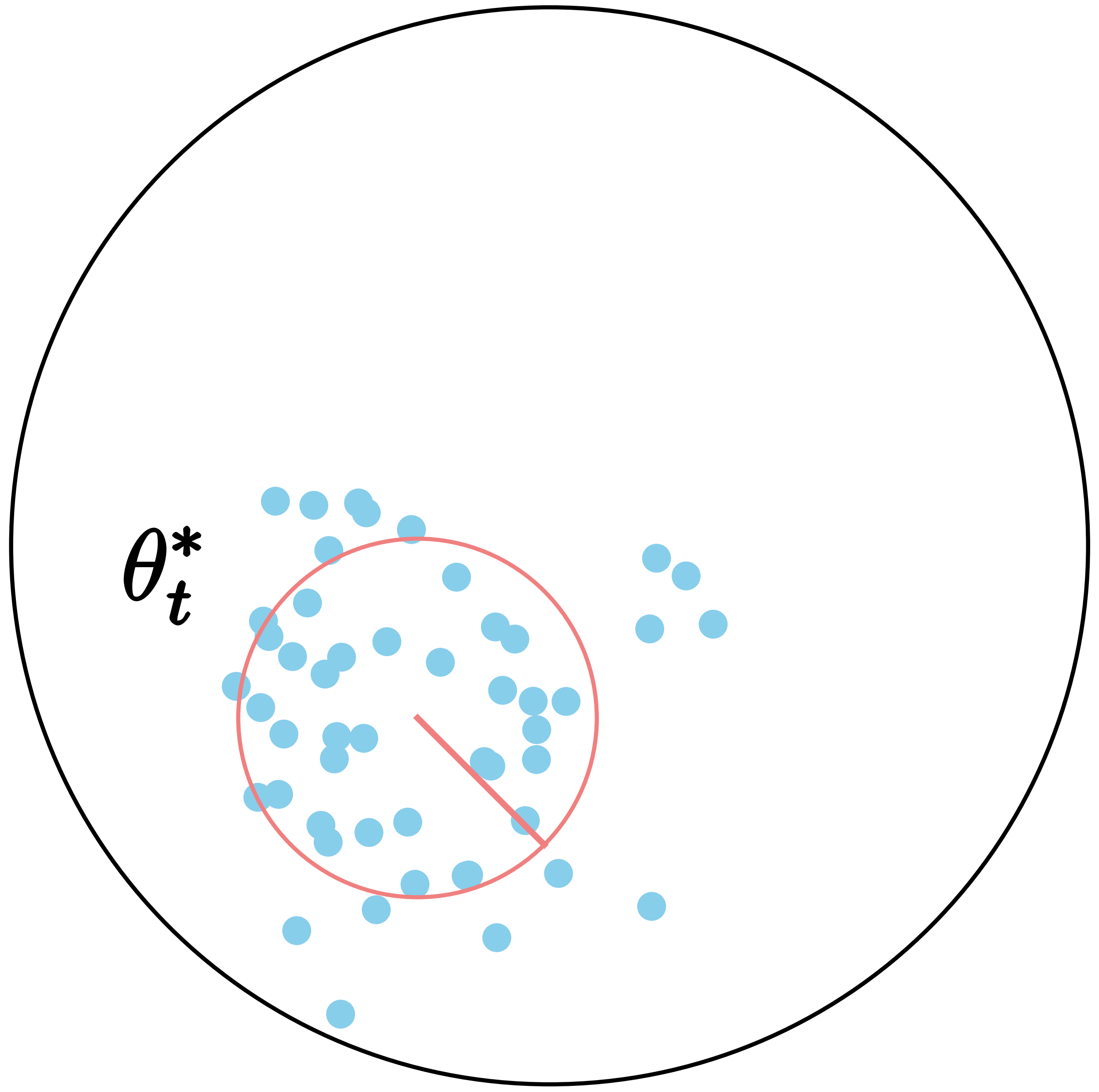
per-coordinate learning rate

gradient on sample i from task t

$$\theta_{t,i+1} = \theta_{t,i} - \eta_{t,i} \odot \nabla_{t,i}$$

When is pre-conditioned online gradient descent useful?

The convex case:



Is learning an initialization good enough?

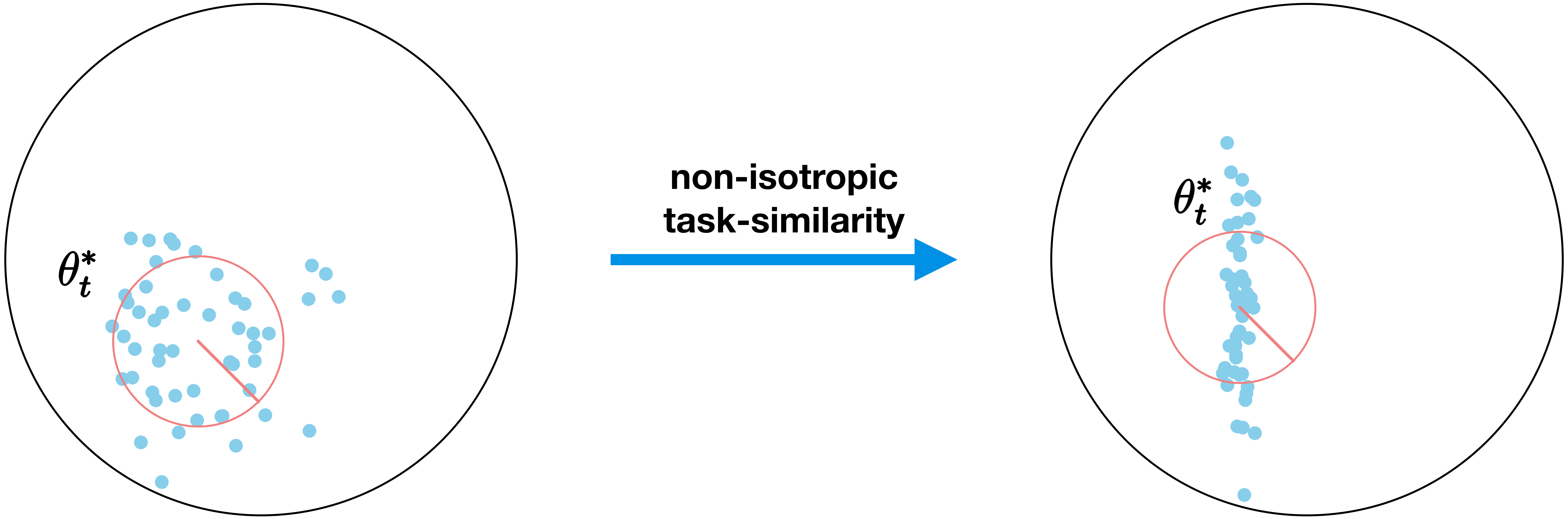
per-coordinate learning rate

gradient on sample i from task t

$$\theta_{t,i+1} = \theta_{t,i} - \eta_{t,i} \odot \nabla_{t,i}$$

When is pre-conditioned online gradient descent useful?

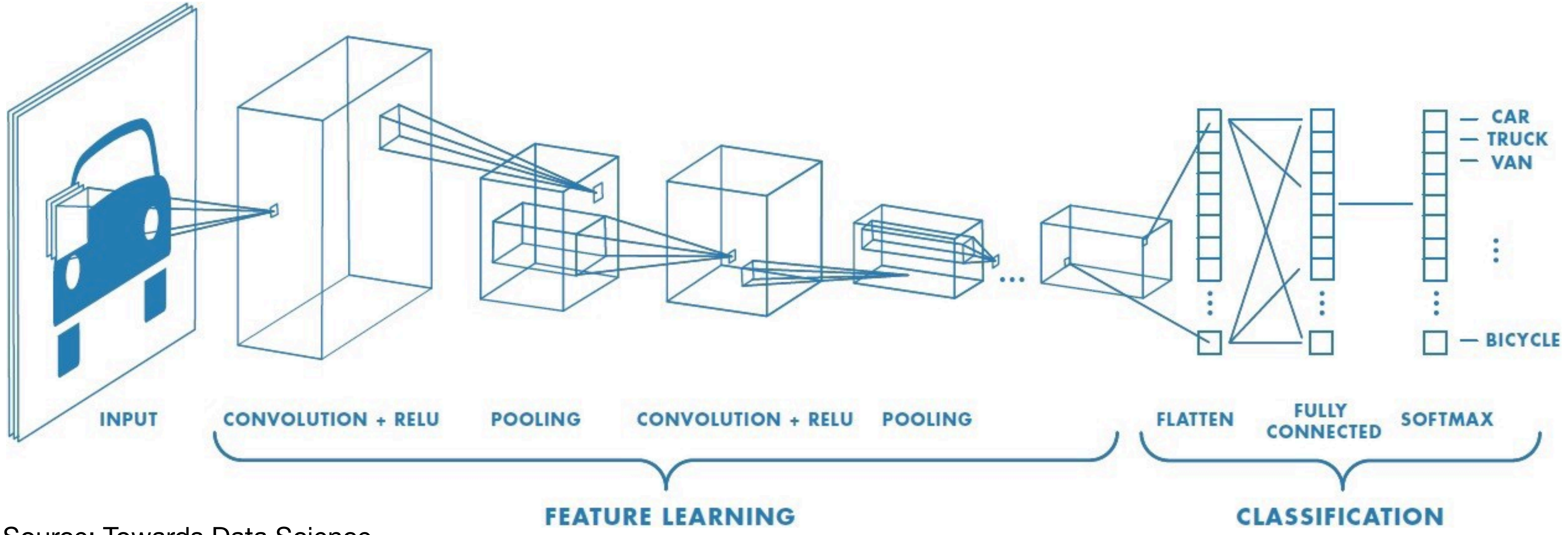
The convex case:



Is learning an initialization good enough?

$$\theta_{t,i+1} = \theta_{t,i} - \underbrace{\eta_{t,i}}_{\text{per-coordinate learning rate}} \odot \underbrace{\nabla_{t,i}}_{\text{gradient on sample i from task t}}$$

When is pre-conditioned online gradient descent useful?
The neural network case:



Source: Towards Data Science

Is learning an initialization good enough?

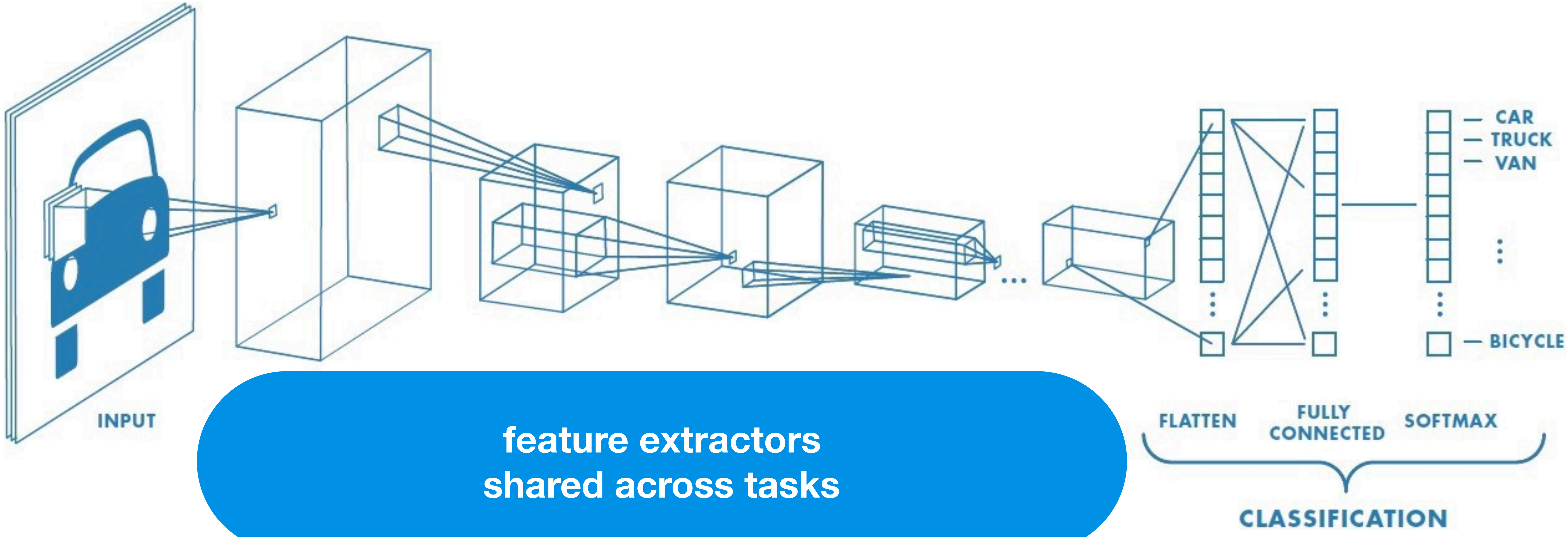
per-coordinate learning rate

gradient on sample i from task t

$$\theta_{t,i+1} = \theta_{t,i} - \eta_{t,i} \odot \nabla_{t,i}$$

When is pre-conditioned online gradient descent useful?

The neural network case:



feature extractors shared across tasks

Is learning an initialization good enough?

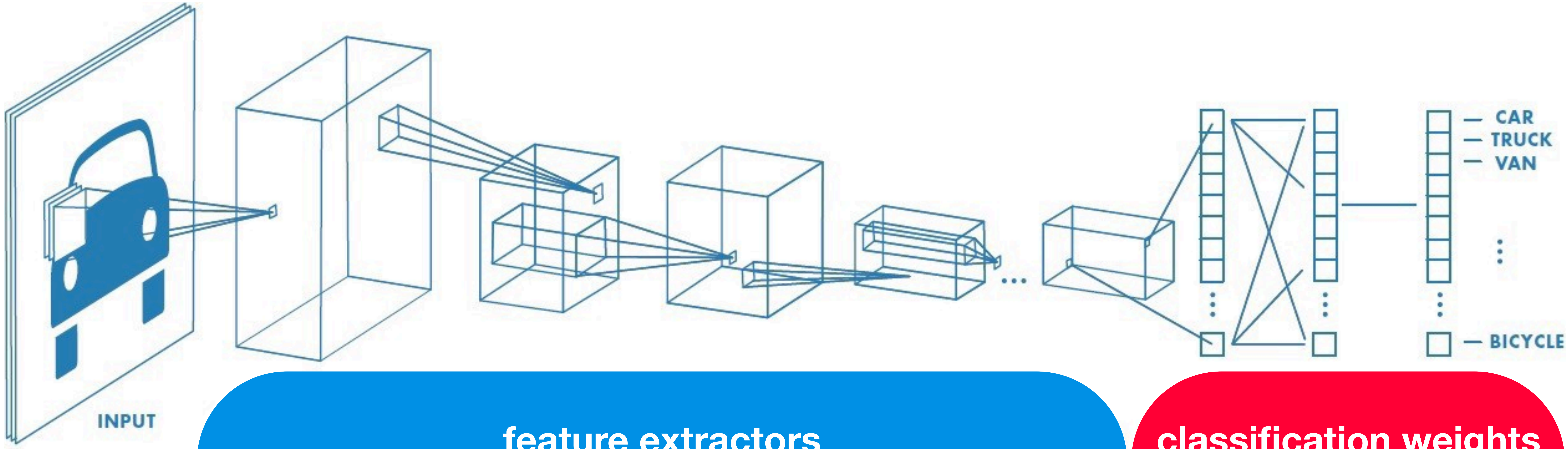
per-coordinate learning rate

gradient on sample i from task t

$$\theta_{t,i+1} = \theta_{t,i} - \eta_{t,i} \odot \nabla_{t,i}$$

When is pre-conditioned online gradient descent useful?

The neural network case:



feature extractors shared across tasks

classification weights adapted for each task

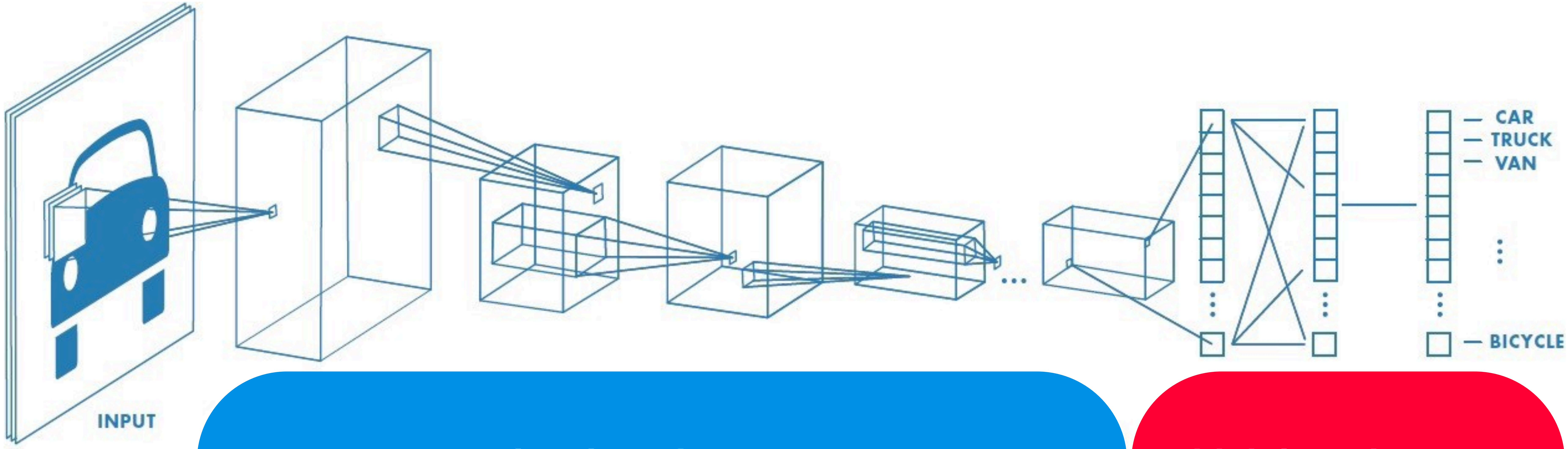
Is learning an initialization good enough?

per-coordinate learning rate

gradient on sample i from task t

$$\theta_{t,i+1} = \theta_{t,i} - \eta_{t,i} \odot \nabla_{t,i}$$

When is pre-conditioned online gradient descent useful?
The neural network case:



low learning rate

high learning rate

Updating the initialization using online learning

for task $t = 1, \ldots, T$

sample task $\mathscr{D}_t$

within-task OGD$(\mathscr{D}_t, \phi_t)$

update ϕ_{t+1} **using** θ_t^*

Updating the initialization and the learning rate using online learning

for task $t = 1,\ldots,T$

sample task $\mathscr{D}_t$

within-task OGD$(\mathscr{D}_t, \phi_t, \eta_t)$

update ϕ_{t+1} **using** θ_t^*

update η_{t+1}

Updating the initialization and the learning rate using online learning

Goal: set ϕ_t, η_t to get low average regret across tasks.

$$\frac{1}{T}\sum_{t=1}^{T} R_t$$

for task $t = 1,\ldots,T$

sample task $\mathscr{D}_t$

within-task OGD$(\mathscr{D}_t, \phi_t, \eta_t)$

update ϕ_{t+1} **using** θ_t^*

update η_{t+1}

Updating the initialization and the learning rate using online learning

Goal: set ϕ_t, η_t to get low average regret across tasks.

for task $t = 1, \ldots, T$

sample task $\mathscr{D}_t$

within-task OGD$(\mathscr{D}_t, \phi_t, \eta_t)$

update ϕ_{t+1} **using** θ_t^*

update η_{t+1}

$$\frac{1}{T}\sum_{t=1}^{T} R_t \le \frac{1}{T}\sum_{t=1}^{T} U_t(\phi_t, \eta_t)$$

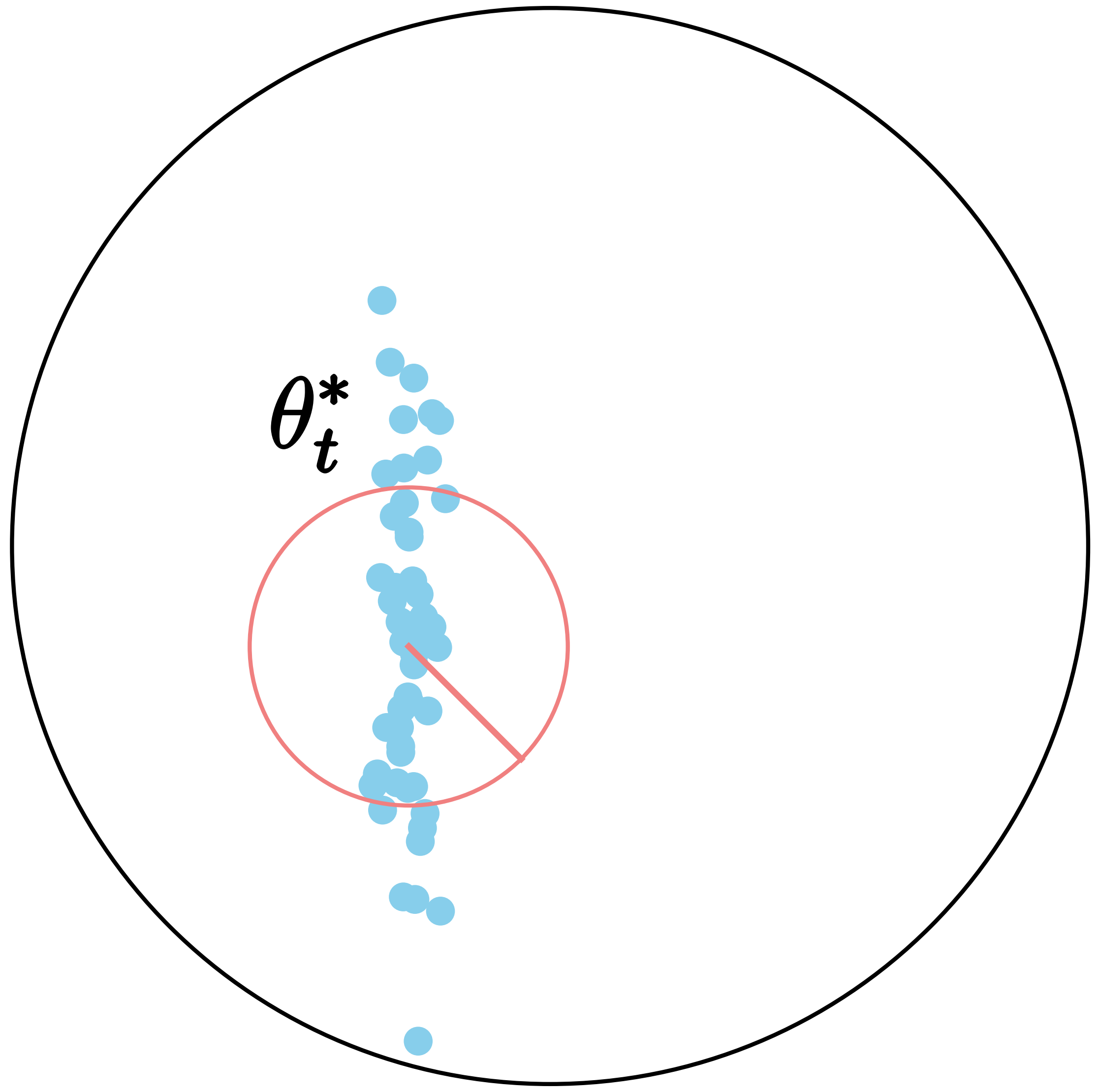
regret-upper-bound

Applying ARUBA: the regret-upper-bound

Single-task regret guarantees are often **nice** and **data-dependent** functions of the algorithm parameters.

$$U_t(\phi, \eta) \quad = \quad \frac{1}{2}\|\theta_t^* - \phi\|_{\frac{1}{\eta}}^2 \quad + \quad \sum_{i=1}^{m} \|\nabla_{t,i}\|_{\eta}^2$$

Mahalanobis norm



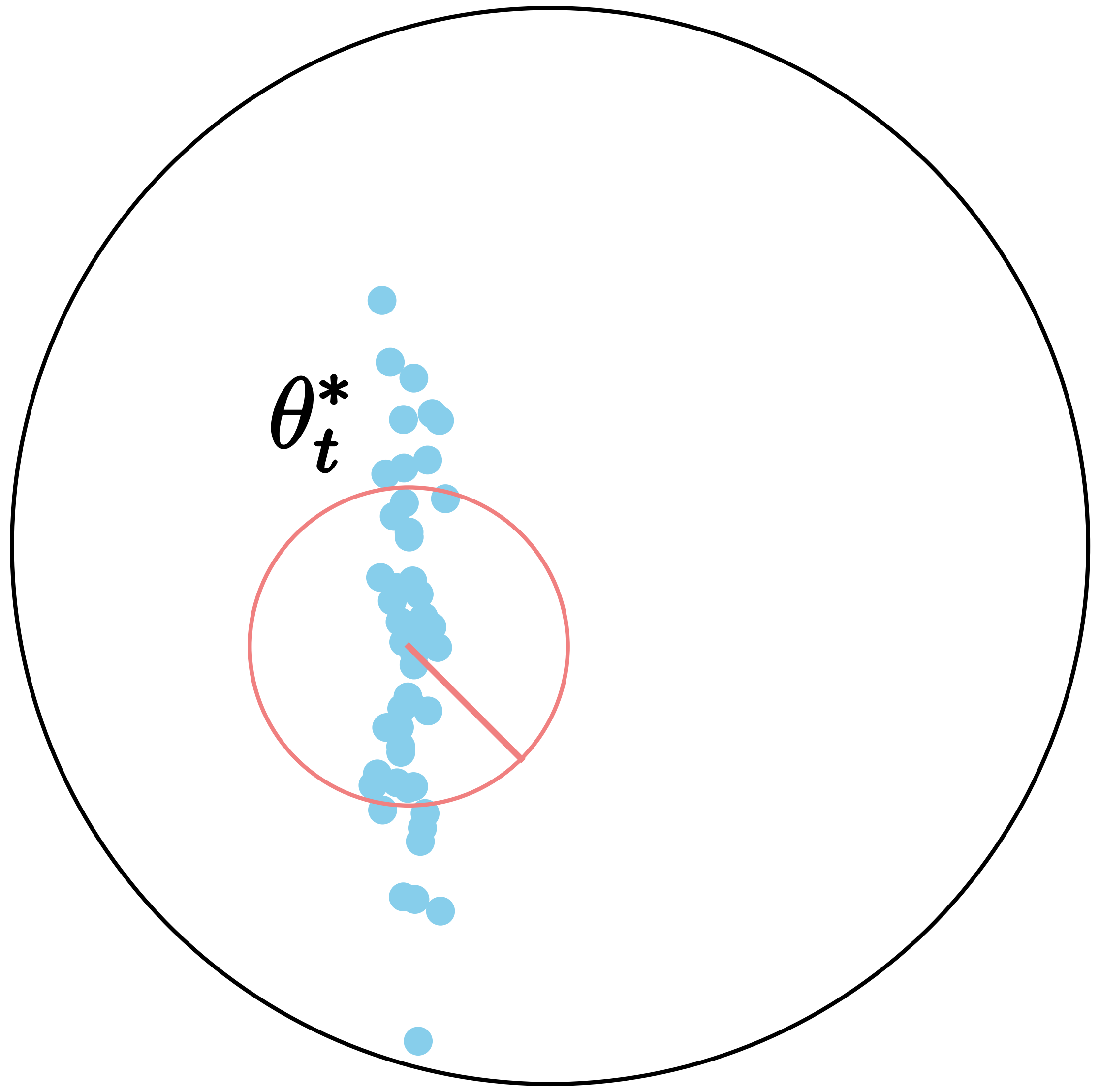
Applying ARUBA: the regret-upper-bound

Single-task regret guarantees are often **nice** and **data-dependent** functions of the algorithm parameters.

$$U_t(\phi, \eta) = \frac{1}{2}\|\theta_t^* - \phi\|_{\frac{1}{\eta}}^2 + \sum_{i=1}^{m} \|\nabla_{t,i}\|_\eta^2$$

$$= \sum_{j=1}^{d} \frac{(\theta_{t,j}^* - \phi_j)^2}{2\eta_j} + \sum_{j=1}^{d} \eta_j \sum_{i=1}^{m} \nabla_{t,i,j}^2$$

summation over coordinates



Setting the learning rate along coordinate j :

optimal learning rate

$$\eta_j = \sqrt{\frac{B_j}{G_j}}$$

$$B_j = \frac{1}{2} \sum_{t=1}^{T} (\theta^*_{t,j} - \phi_{s,j})^2$$

sum of squared distances from initialization

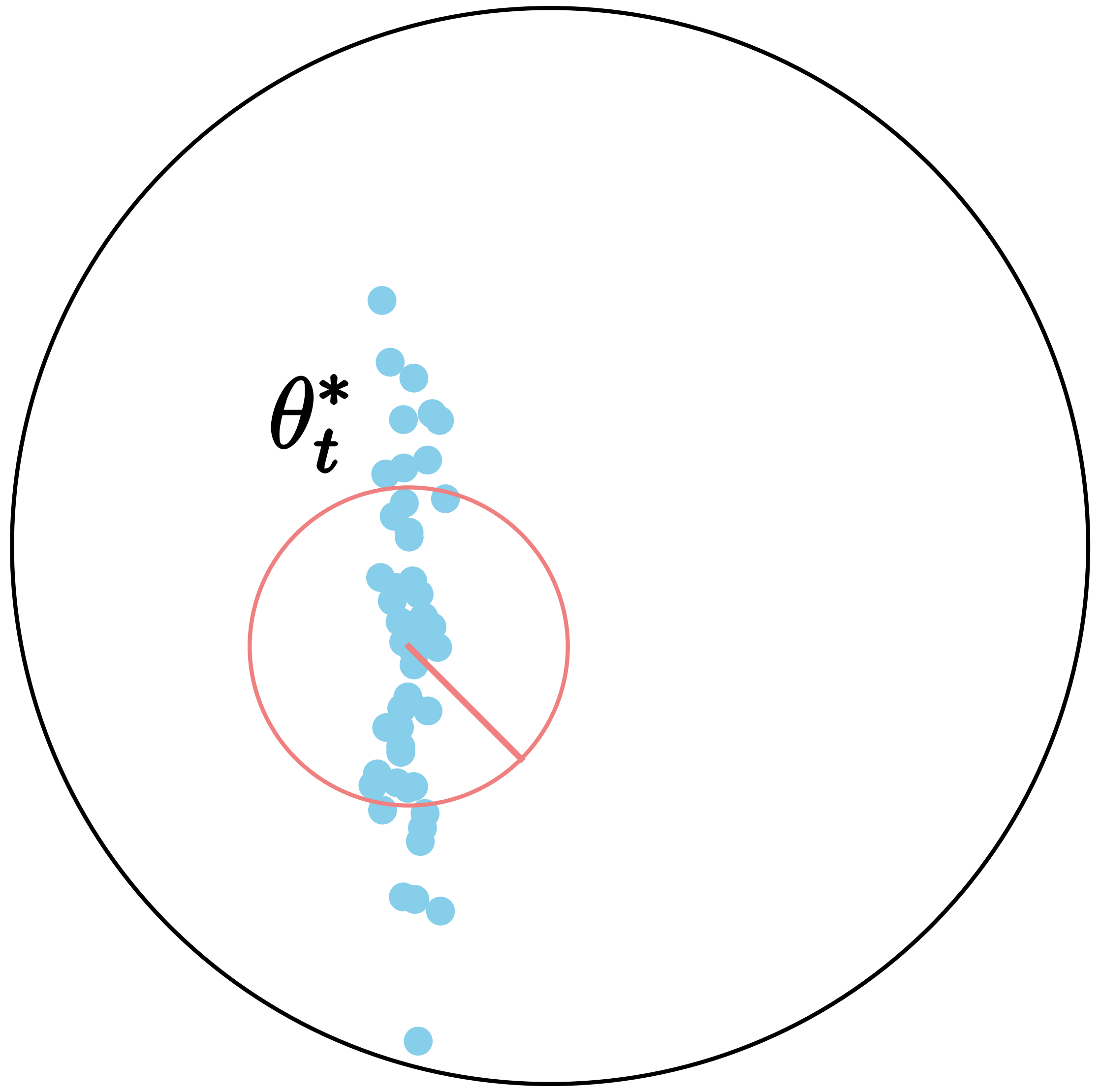
$$G_j = \sum_{t=1}^{T} \sum_{i=1}^{m} \nabla^2_{t,i,j}$$

sum of squared gradients

Setting the learning rate along coordinate j :

learned learning rate

$$\eta_{t,j} = \sqrt{\frac{B_{t,j} + \varepsilon_t}{G_{t,j} + \zeta_t}}$$



track quantities across tasks

$$B_{t,j} = \frac{1}{2} \sum_{s<t} (\theta^*_{s,j} - \phi_{s,j})^2$$

sum of squared distances from initialization

$$G_{t,j} = \sum_{s<t} \sum_{i=1}^{m} \nabla^2_{s,i,j}$$

sum of squared gradients

Setting the learning rate along coordinate j :

learned learning rate

$$\eta_{t,j} = \sqrt{\frac{B_{t,j} + \varepsilon_t}{G_{t,j} + \zeta_t}}$$

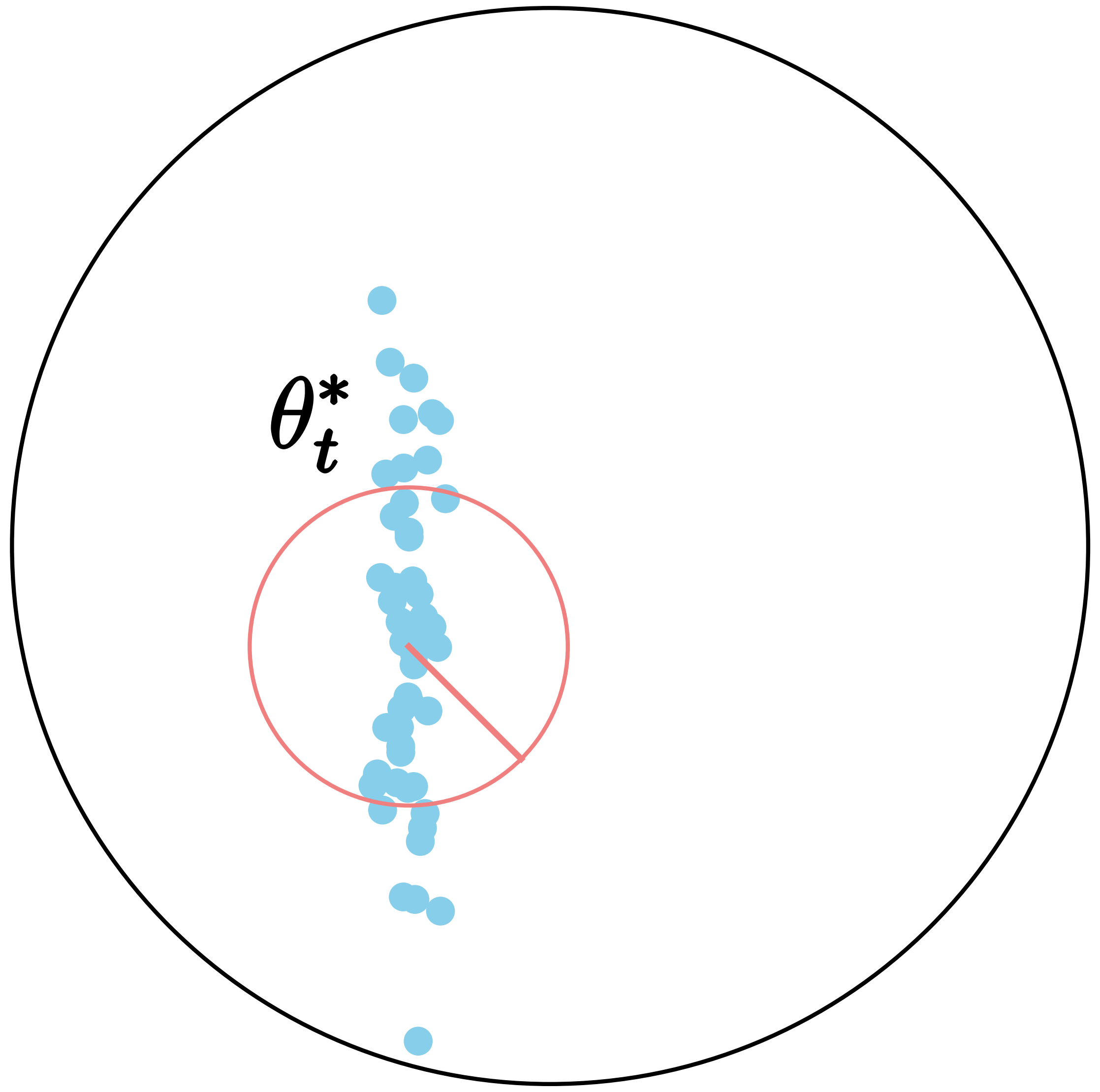
add smoothing terms

$$B_{t,j} = \frac{1}{2} \sum_{s<t} (\theta^*_{s,j} - \phi_{s,j})^2$$

sum of squared distances from initialization

$$G_{t,j} = \sum_{s<t} \sum_{i=1}^{m} \nabla^2_{s,i,j}$$

sum of squared gradients



Setting the learning rate along coordinate j :

learned learning rate

$$\eta_{t,j} = \sqrt{\frac{B_{t,j} + \varepsilon_t}{G_{t,j} + \zeta_t}}$$

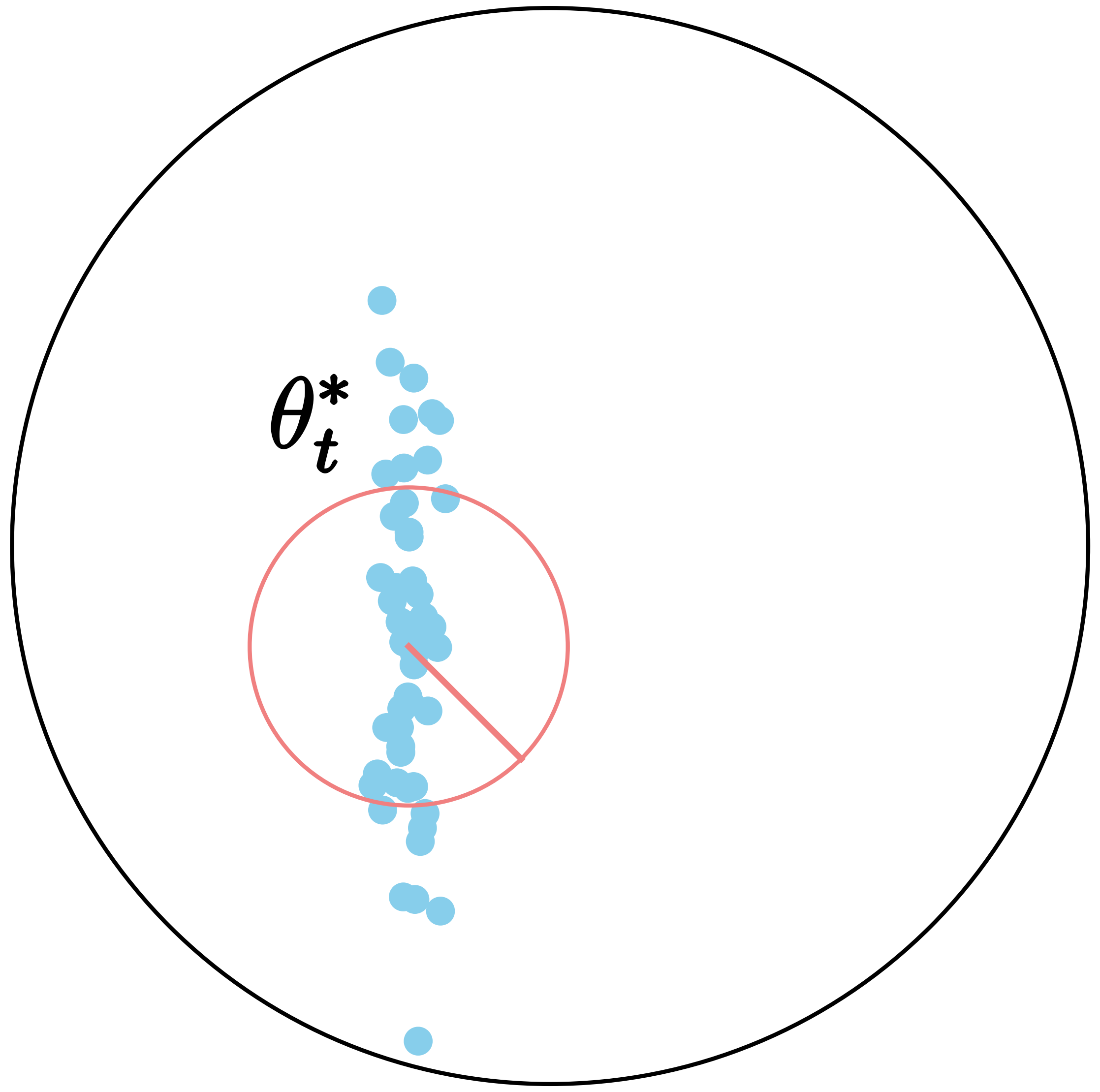
$$B_{t,j} = \frac{1}{2} \sum_{s<t} (\theta^*_{s,j} - \phi_{s,j})^2$$

sum of squared distances from initialization

Theorem: $\tilde{\mathcal{O}}(T^{-2/5})$ **convergence to optimal per-coordinate** η

$$G_{t,j} = \sum_{s<t} \sum_{i=1}^{m} \nabla^2_{s,i,j}$$

sum of squared gradients



Setting the learning rate along coordinate j :

learned learning rate

$$\eta_{t,j} = \sqrt{\frac{B_{t,j} + \varepsilon_t}{G_{t,j} + \zeta_t}}$$

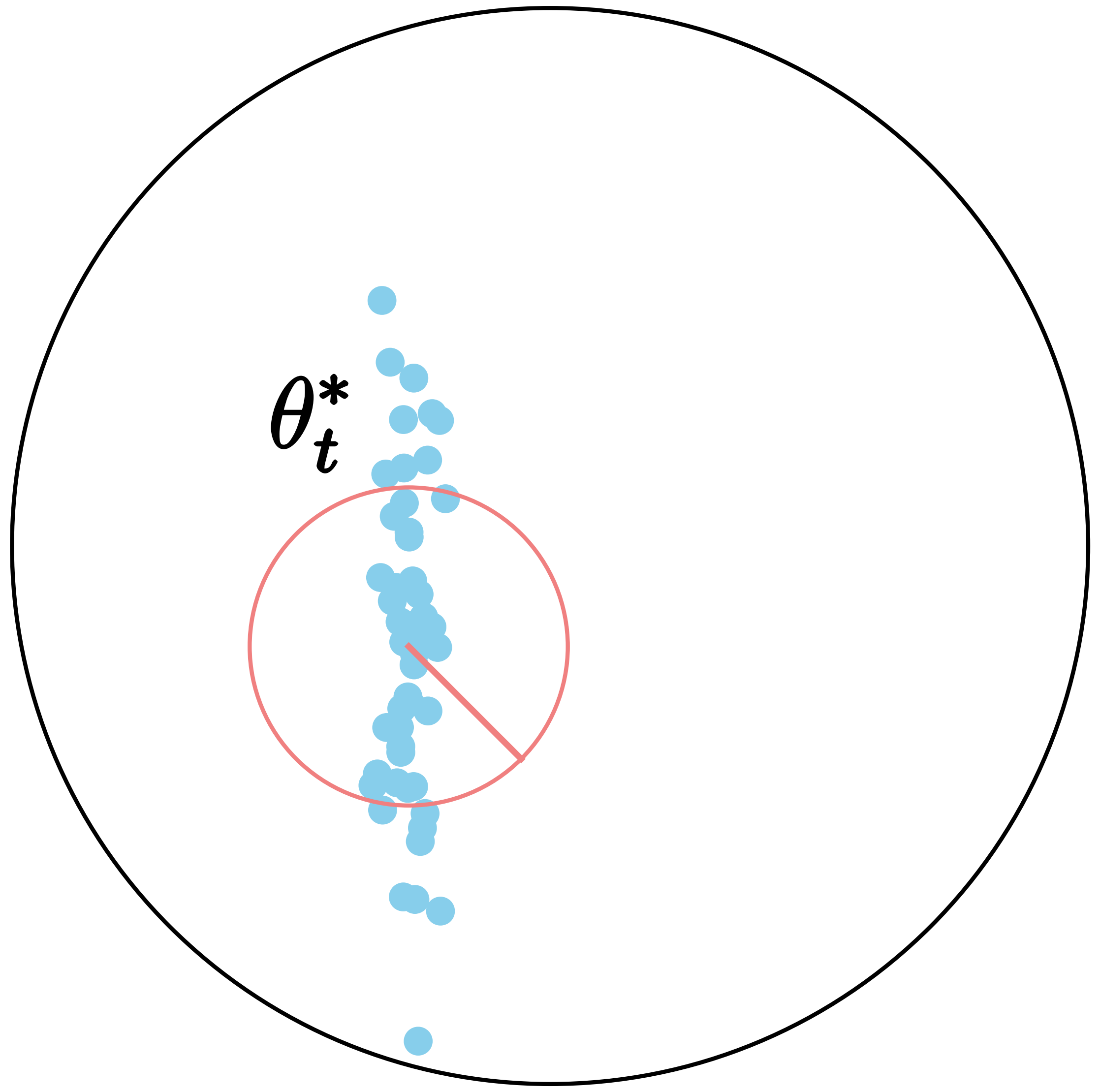
$$B_{t,j} = \frac{1}{2} \sum_{s<t} (\hat{\theta}_{s,j} - \phi_{s,j})^2$$

sum of squared distances from initialization

to obtain a practical algorithm, use last-iterate

$$G_{t,j} = \sum_{s<t} \sum_{i=1}^{m} \nabla^2_{s,i,j}$$

sum of squared gradients



Setting the learning rate along coordinate j :

learned learning rate

$$\eta_{t,j} = \sqrt{\frac{B_{t,j} + \varepsilon_t}{G_{t,j} + \zeta_t}}$$

$$B_{t,j} = \frac{1}{2} \sum_{s<t} (\hat{\theta}_{s,j} - \phi_{s,j})^2$$

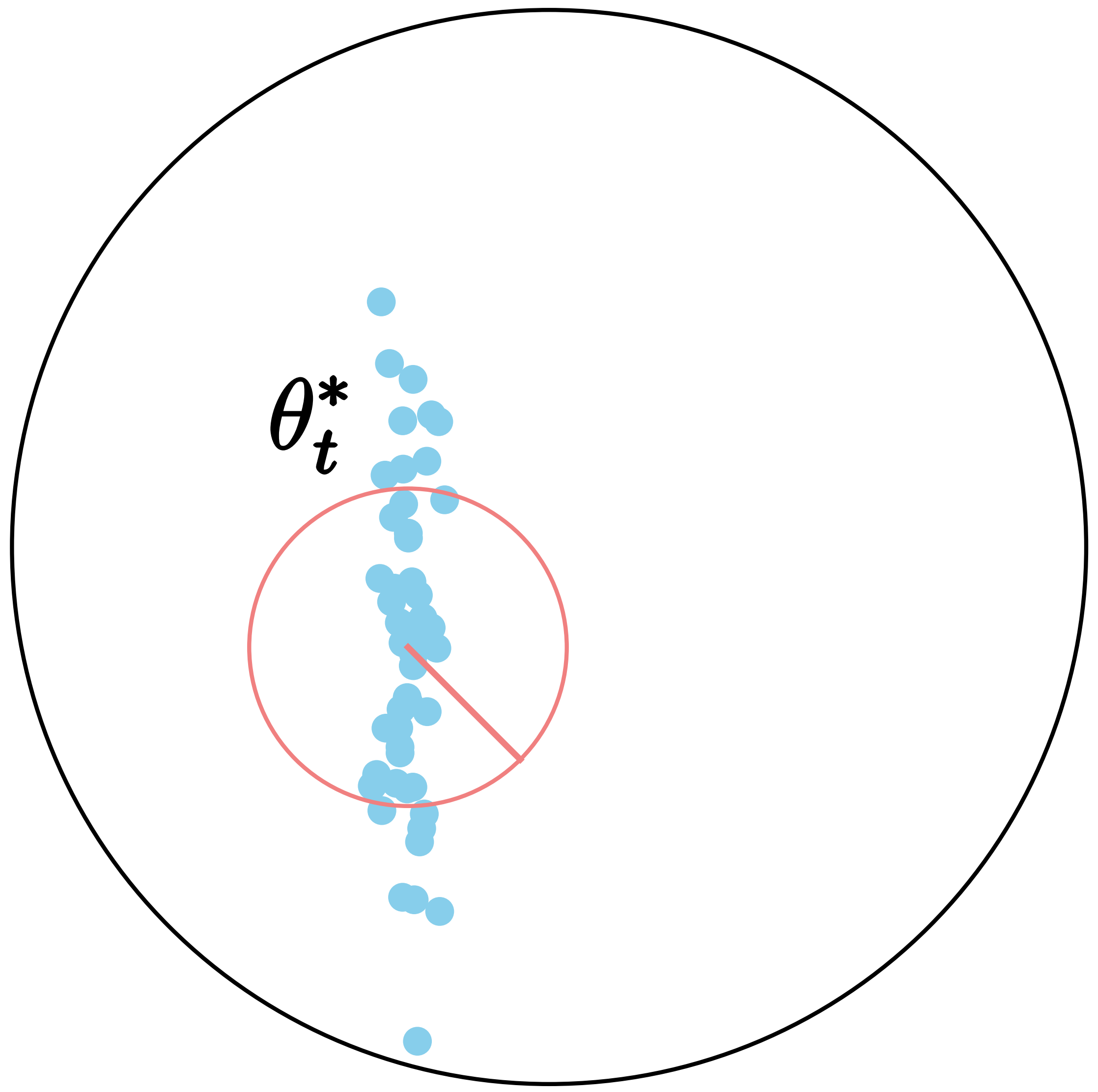
sum of squared distances from initialization

compare to AdaGrad [Duchi-Hazan-Singer]

$$\eta_{t,j} = \frac{\eta_0}{\sqrt{G_{t,j} + \delta}}$$

$$G_{t,j} = \sum_{s<t} \sum_{i=1}^{m} \nabla^2_{s,i,j}$$

sum of squared gradients



Applying this meta-learned learning rate on few-shot image classification

Applying this meta-learned learning rate on few-shot image classification

Mini-ImageNet dataset [Ravi-Larochelle]:

generate n-shot 5-way classification tasks by sampling n images from each of 5 classes

Applying this meta-learned learning rate on few-shot image classification

Mini-ImageNet dataset [Ravi-Larochelle]:

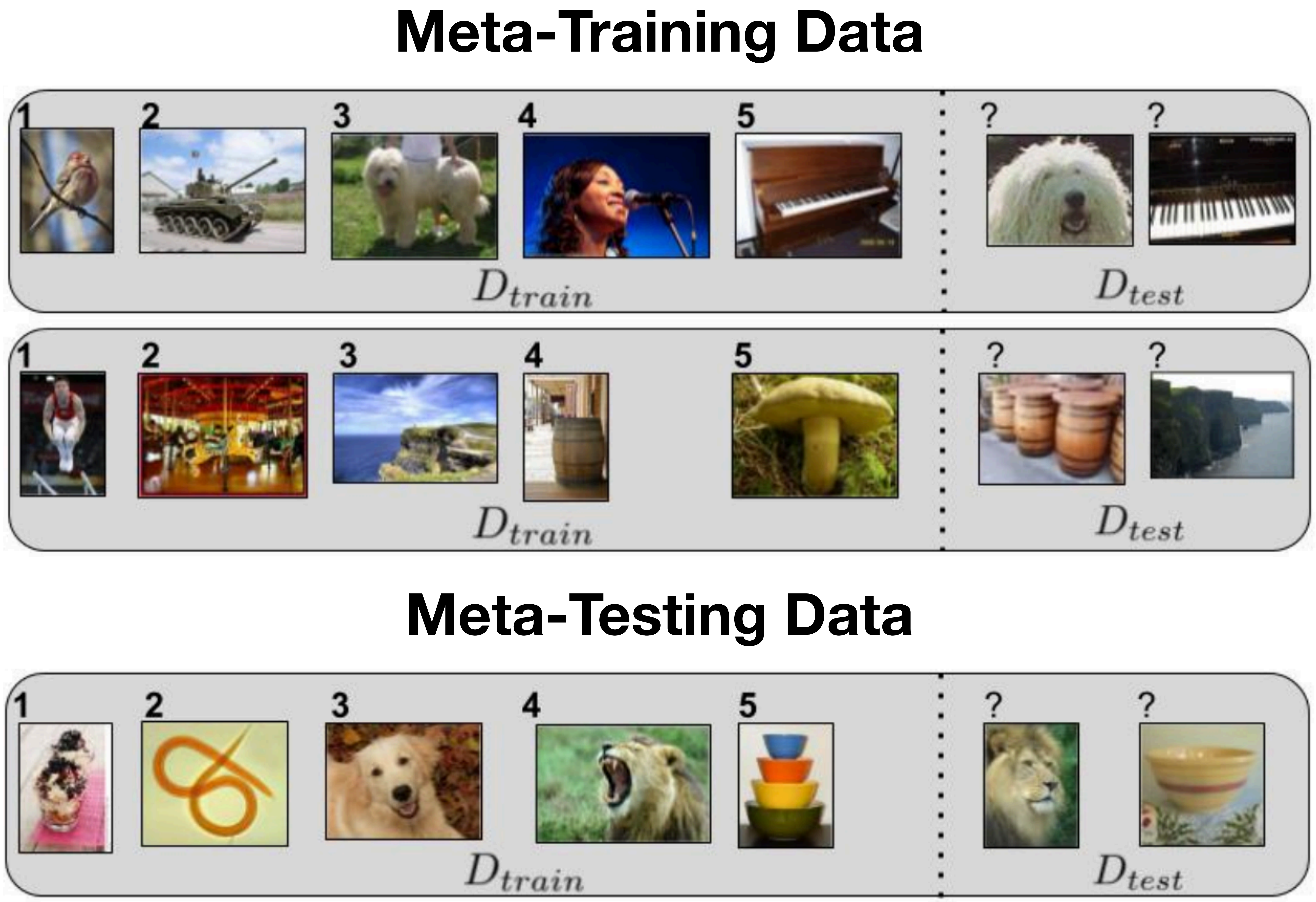
generate n-shot 5-way classification tasks by sampling n images from each of 5 classes

Meta-Training Data

Applying this meta-learned learning rate on few-shot image classification

Mini-ImageNet dataset [Ravi-Larochelle]:

generate n-shot 5-way classification tasks by sampling n images from each of 5 classes



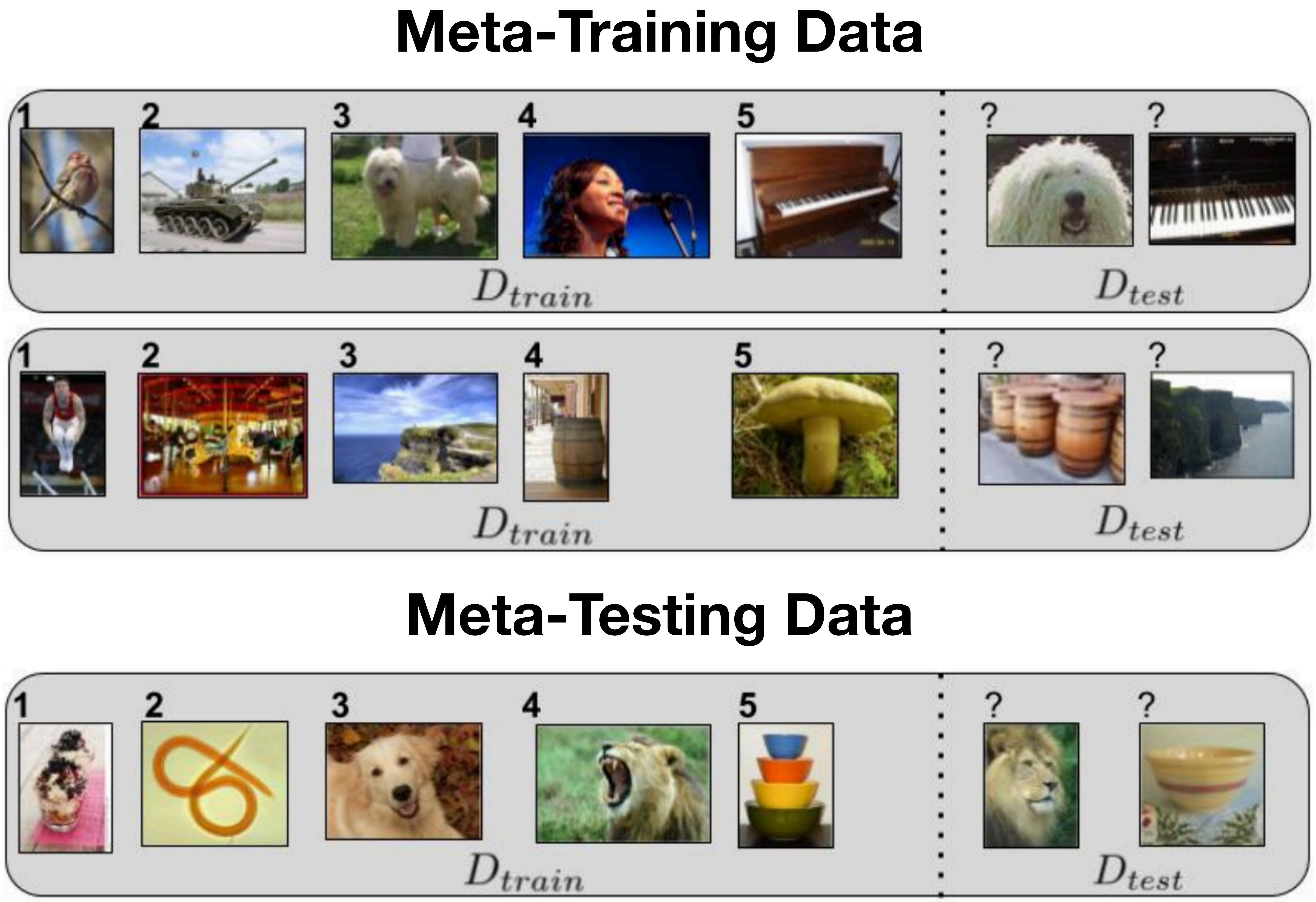
Applying this meta-learned learning rate on few-shot image classification

Mini-ImageNet dataset [Ravi-Larochelle]:

generate n-shot 5-way classification tasks by sampling n images from each of 5 classes

Goal:

learn to initialize ($\hat{\phi}$) and adapt ($\hat{\eta}$) a four-layer convolutional neural network



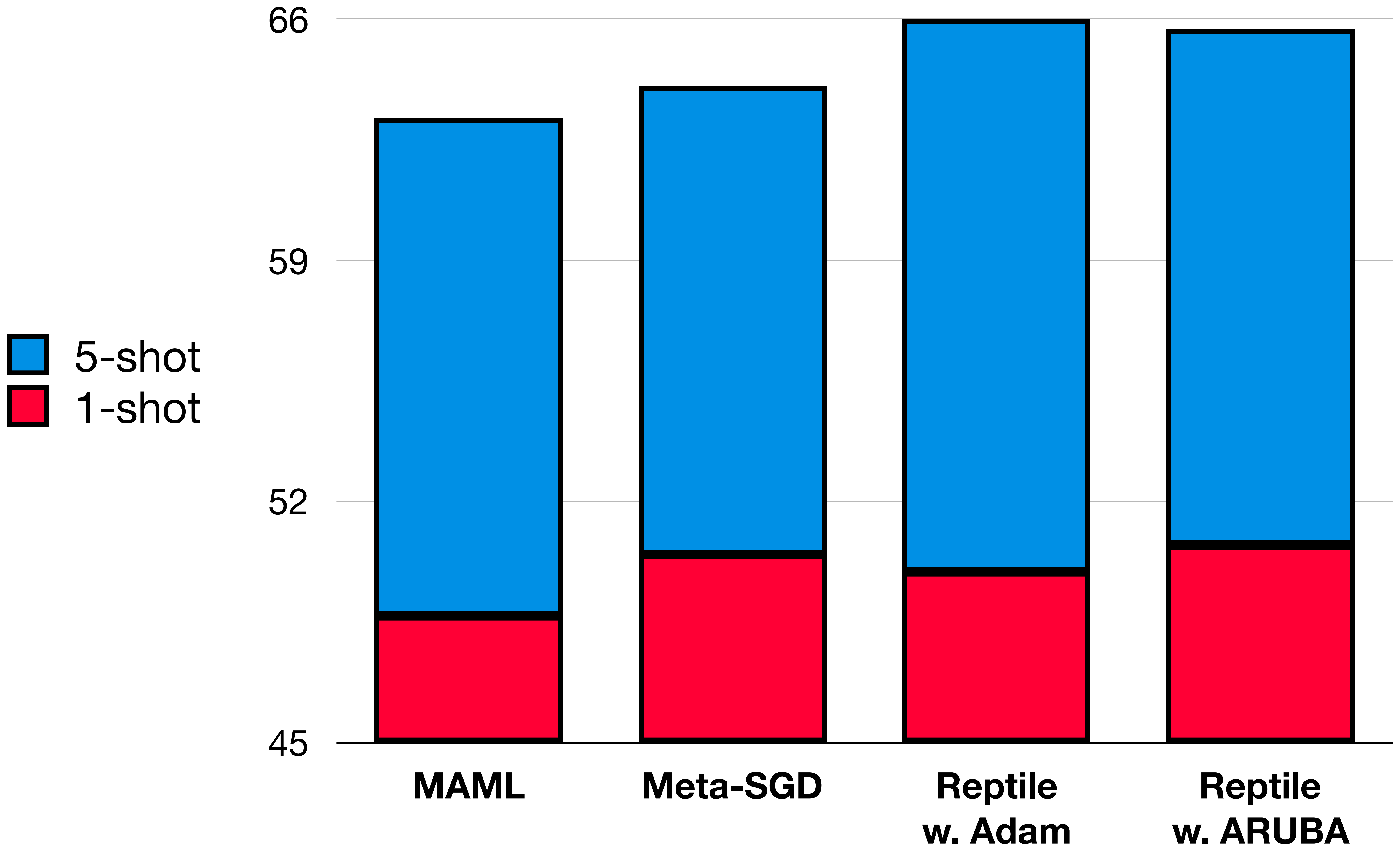
Applying this meta-learned learning rate on few-shot image classification

Mini-ImageNet dataset [Ravi-Larochelle]:

generate n-shot 5-way classification tasks by sampling n images from each of 5 classes

Goal:

learn to initialize ($\hat{\phi}$) and adapt ($\hat{\eta}$) a four-layer convolutional neural network



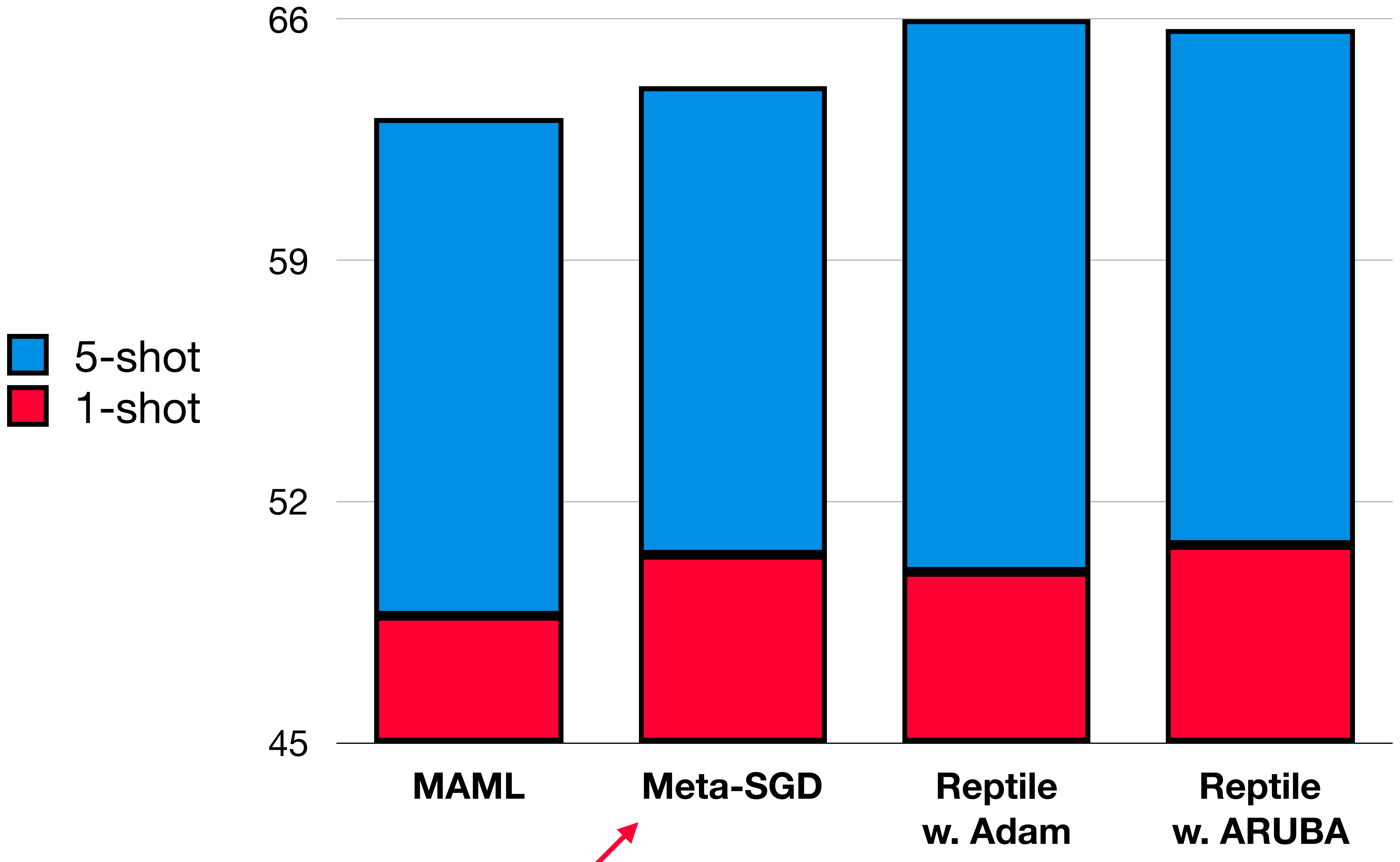
Applying this meta-learned learning rate on few-shot image classification

Mini-ImageNet dataset [Ravi-Larochelle]:

generate n-shot 5-way classification tasks by sampling n images from each of 5 classes

Goal:

learn to initialize ($\hat{\phi}$) and adapt ($\hat{\eta}$) a four-layer convolutional neural network



MAML with per-coordinate learning rate [Li et al.]

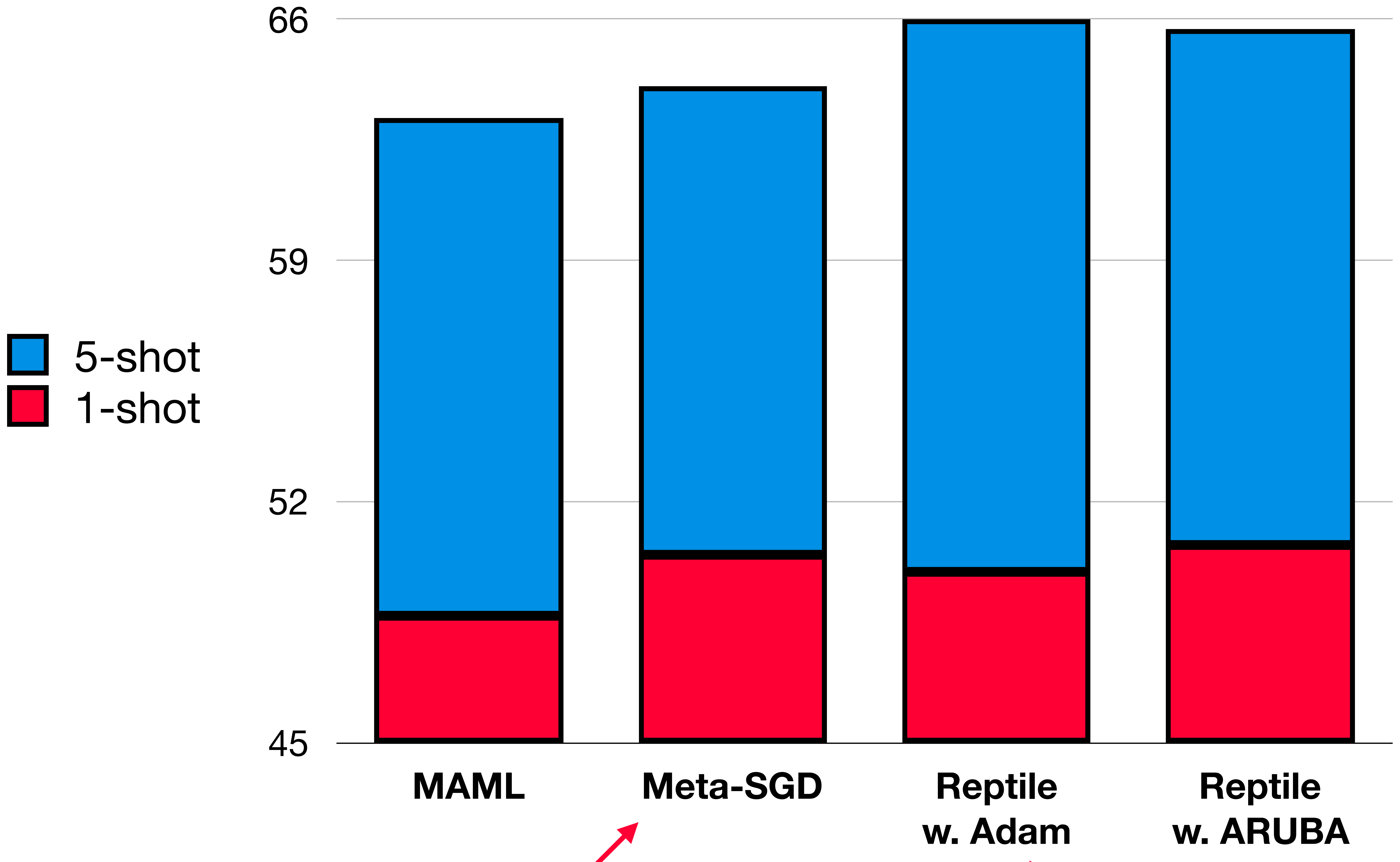
Applying this meta-learned learning rate on few-shot image classification

Mini-ImageNet dataset [Ravi-Larochelle]:

generate n-shot 5-way classification tasks by sampling n images from each of 5 classes

Goal:

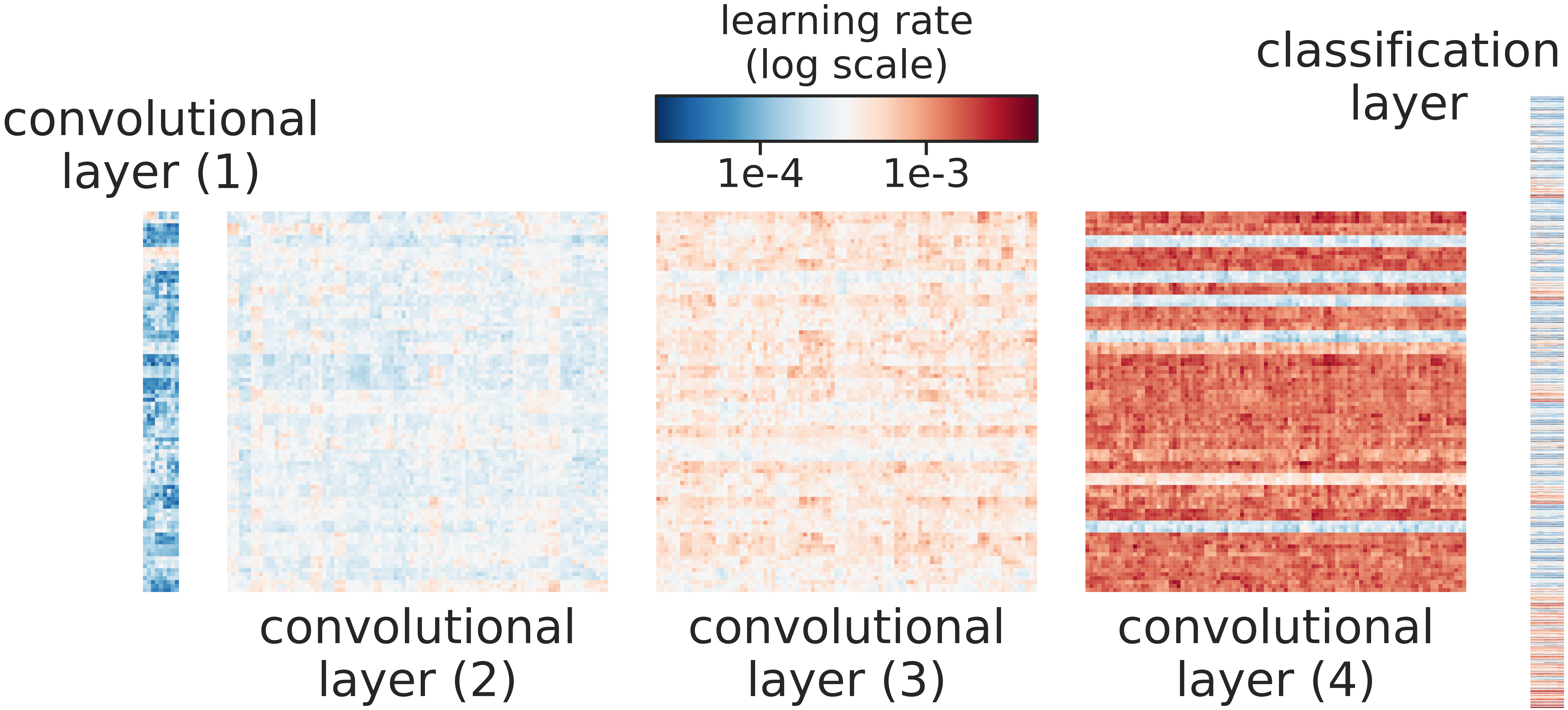
learn to initialize $(\hat{\phi})$ and adapt $(\hat{\eta})$ a four-layer convolutional neural network



MAML with per-coordinate learning rate [Li et al.]

popular per-coordinate learning rate [Kingma-Ba]

Our adaptive learning rate after meta-training



ARUBA Framework

Applications

- Adaptivity for improved few-shot learning
- Federated learning & private meta-learning

Personalized Federated Learning

- Massively distributed
- Small sample sizes
- Privacy concerns
- Non-IID data and tasks
- Underlying task similarity

FedAvg ≈ Reptile [with a batch-averaged meta-update]

FedAvg ≈ Reptile [with a batch-averaged meta-update]

- Most popular algorithm in federated learning
- Usually run without personalization - just use the meta-initialization within-task

Personalization in Federated Learning via Adaptive ARUBA

- Meta-training: run FedAvg with ARUBA optimizer within-task

Personalization in Federated Learning via Adaptive ARUBA

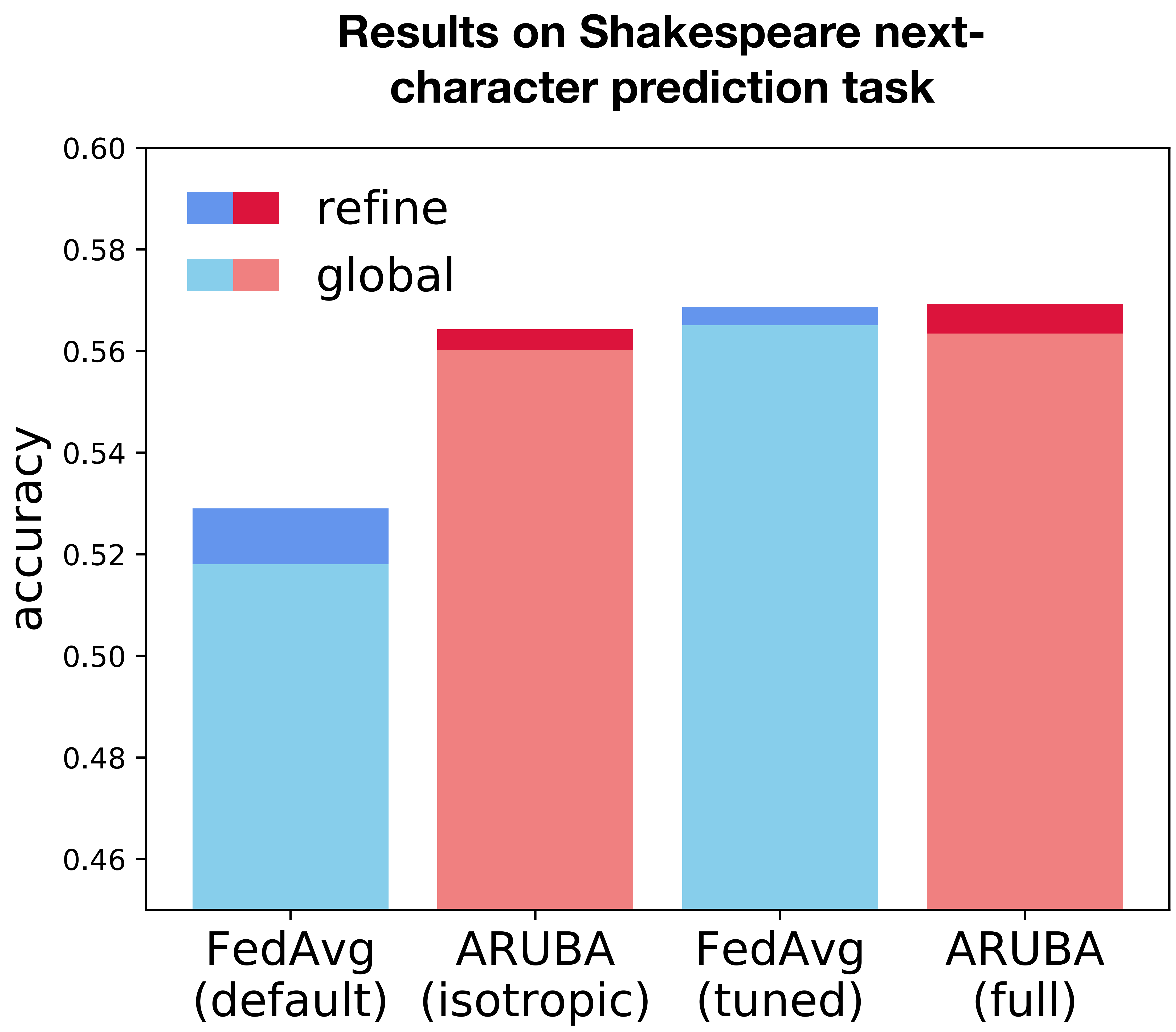
- Meta-training: run FedAvg with ARUBA optimizer within-task

- Meta-testing - use (preconditioned) OGD to learn a personalized model for each user

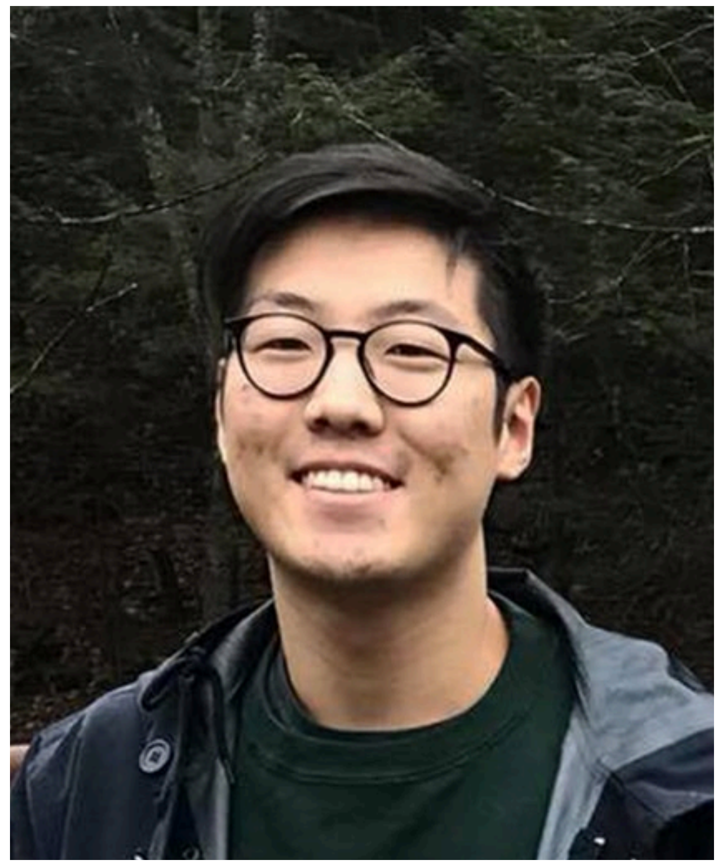
Personalization in Federated Learning via Adaptive ARUBA

- Meta-training: run FedAvg with ARUBA optimizer within-task

- Meta-testing - use (preconditioned) OGD to learn a personalized model for each user



Differentially Private Federated Learning via ARUBA



Jeff Li

Sebastian Caldas

Ameet Talwalkar

Differentially Private Federated Learning via ARUBA

Motivation:

- protect user data from untrusted central server - the meta-learner
- avoid utility loss associated with local differential privacy

Differentially Private Federated Learning via ARUBA

for task $t = 1,\ldots,T$

sample task $\mathscr{D}_t$

$\hat{\theta}_t \leftarrow$ **within-task SGD**$(\mathscr{D}_t, \phi_t)$

$\phi_{t+1} \leftarrow (1-\alpha)\phi_t + \alpha\hat{\theta}_t$

return $\hat{\phi} = \phi_{T+1}$

Motivation:

- protect user data from untrusted central server - the meta-learner
- avoid utility loss associated with local differential privacy

Differentially Private Federated Learning via ARUBA

Motivation:

- protect user data from untrusted central server - the meta-learner
- avoid utility loss associated with local differential privacy

for task $t = 1,\ldots,T$

sample task $\mathscr{D}_t$

$\hat{\theta}_t \leftarrow$ **within-task SGD**$(\mathscr{D}_t, \phi_t)$

$\phi_{t+1} \leftarrow (1-\alpha)\phi_t + \alpha\hat{\theta}_t$ ← **not private to central server**

return $\hat{\phi} = \phi_{T+1}$

Differentially Private Federated Learning via ARUBA

for task $t = 1,\ldots,T$

sample task $\mathscr{D}_t$

$\hat{\theta}_t \leftarrow$ **within-task noisy SGD**$(\mathscr{D}_t, \phi_t)$

$\phi_{t+1} \leftarrow (1-\alpha)\phi_t + \alpha\hat{\theta}_t$

return $\hat{\phi} = \phi_{T+1}$

Motivation:

- protect user data from untrusted central server - the meta-learner
- avoid utility loss associated with local differential privacy

Differentially Private Federated Learning via ARUBA

for task $t = 1,\ldots,T$

sample task $\mathscr{D}_t$

$\hat{\theta}_t \leftarrow$ **within-task noisy SGD**$(\mathscr{D}_t, \phi_t)$

$\phi_{t+1} \leftarrow (1-\alpha)\phi_t + \alpha\hat{\theta}_t$

return $\hat{\phi} = \phi_{T+1}$

Motivation:

- protect user data from untrusted central server - the meta-learner
- avoid utility loss associated with local differential privacy

Our results:

- immediate user-record-level privacy guarantee for any model

Differentially Private Federated Learning via ARUBA

for task $t = 1,\ldots,T$

sample task $\mathscr{D}_t$

$\hat{\theta}_t \leftarrow$ **within-task noisy SGD**$(\mathscr{D}_t, \phi_t)$

$\phi_{t+1} \leftarrow (1-\alpha)\phi_t + \alpha\hat{\theta}_t$

return $\hat{\phi} = \phi_{T+1}$

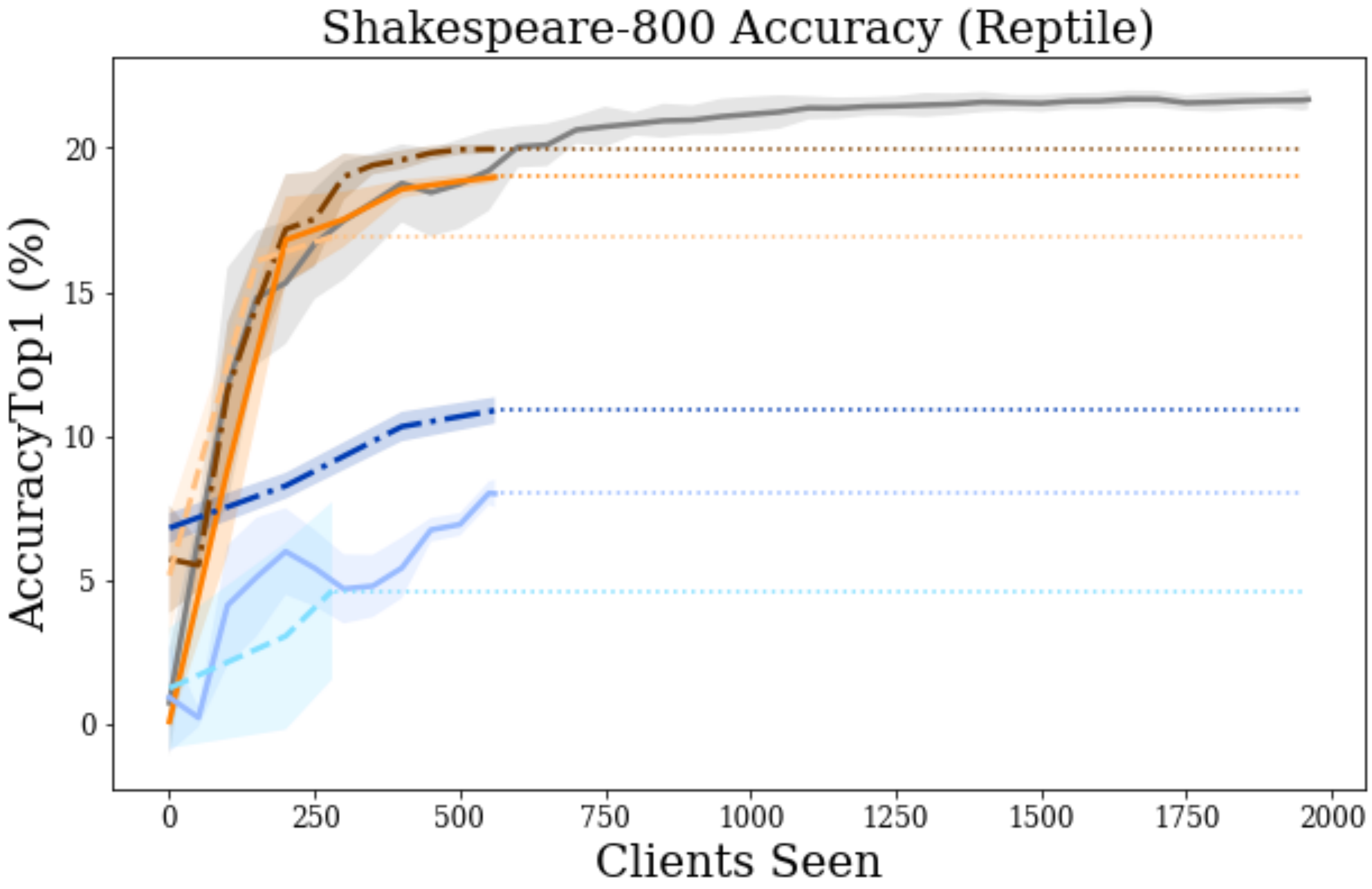
Motivation:

- protect user data from untrusted central server - the meta-learner
- avoid utility loss associated with local differential privacy

Our results:

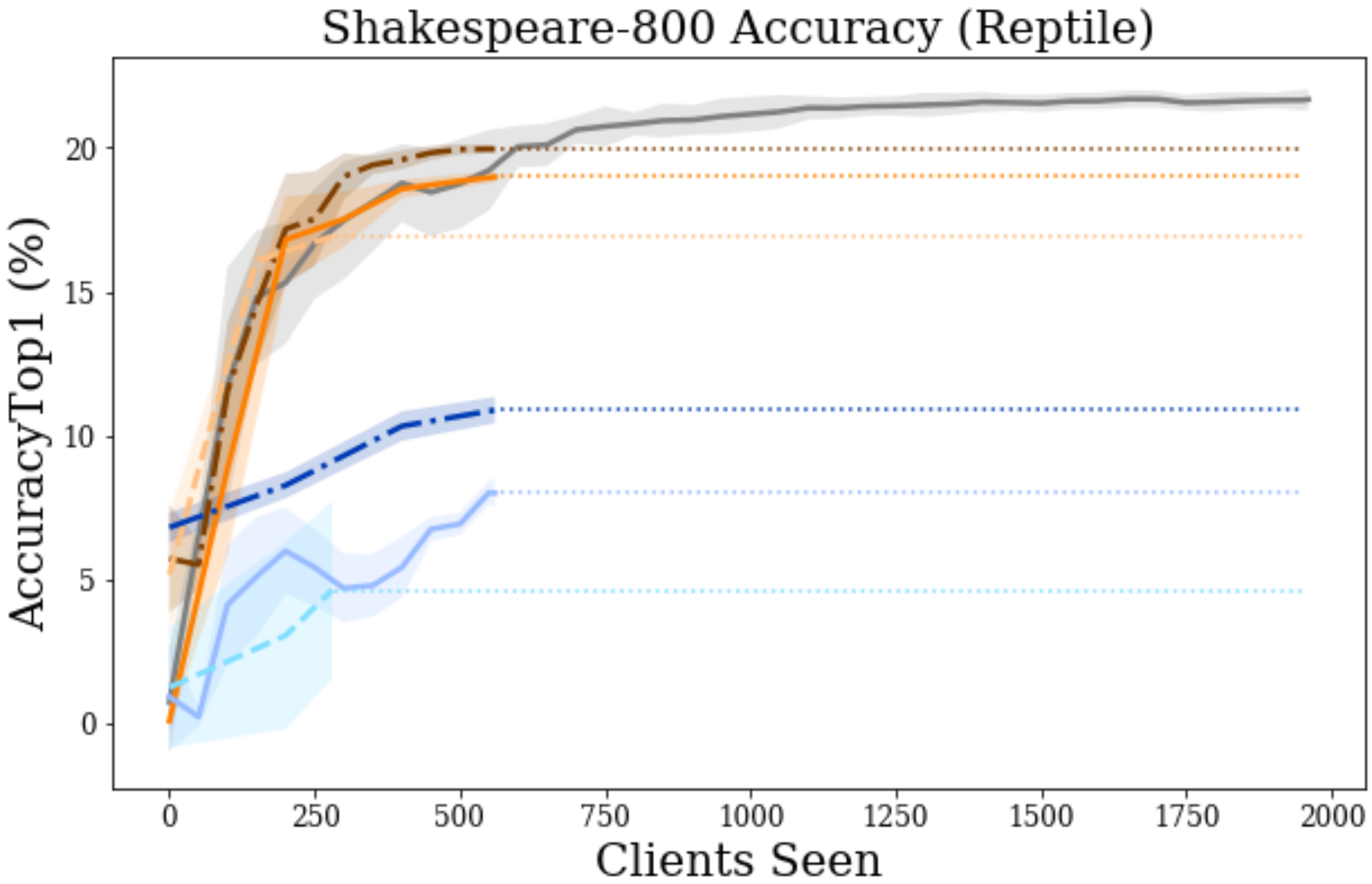
- immediate user-record-level privacy guarantee for any model
- in the convex case: bound on excess transfer risk that improves with task-similarity

Differentially Private Next-Character Prediction



local differential privacy
(for three different
privacy budgets)

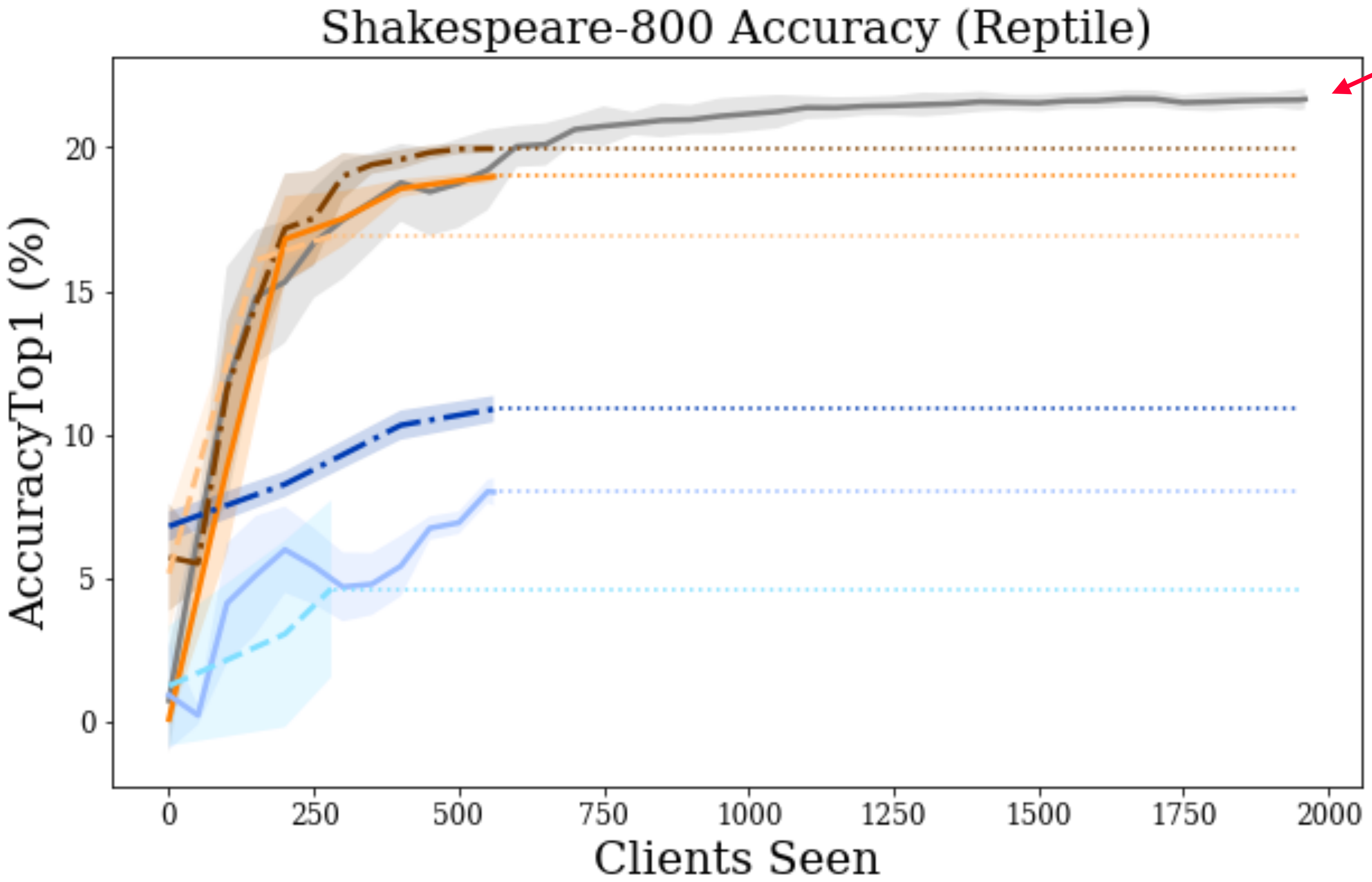
Differentially Private Next-Character Prediction



our approach (for three different privacy budgets)

local differential privacy (for three different privacy budgets)

Differentially Private Next-Character Prediction



non-private learning

our approach (for three different privacy budgets)

local differential privacy (for three different privacy budgets)

Takeaways

ARUBA: a theoretical framework for analyzing and designing meta-learning algorithms via reduction to online learning:

- First guarantees for initialization-based meta-learning methods showing provable improvement over single-task learning
- New principled algorithm for meta-learning the learning rate in addition to the initialization
- Novel practical algorithm for differentially private meta-learning

Next steps

Future directions:

- Beyond adversarial analysis within-task — can the base-learners be statistical or reinforcement learning algorithms?
- Better multi-task optimizers for regimes beyond few-shot learning.
- Non-convex losses and non-linear representations

Thank You!

ARUBA: https://arxiv.org/abs/1906.02717

More Info: http://www.cs.cmu.edu/~mkhodak/

Blog: https://blog.ml.cmu.edu/2019/11/22/aruba