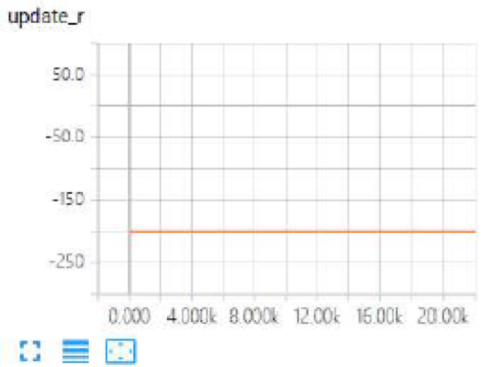
Meta (Reinforcement) Learning

王维埙 Weixun Wang

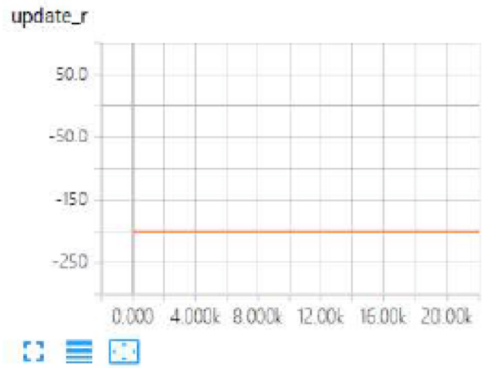
2019-08-06

https://github.com/wwxFromTju/wwxFromTju.github.io/blob/master/slide/Meta(Reinforcement)Learning-0805.pdf

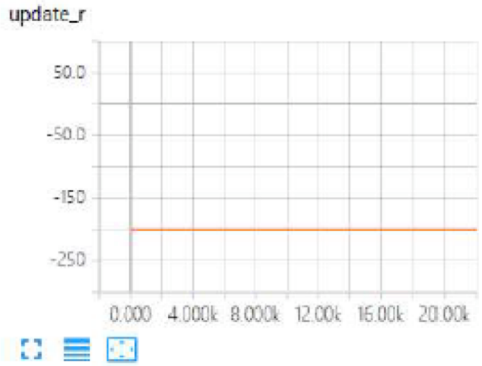
When you train DRL



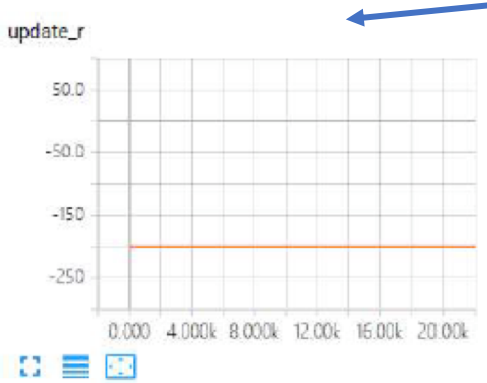
10 minutes



1 hour



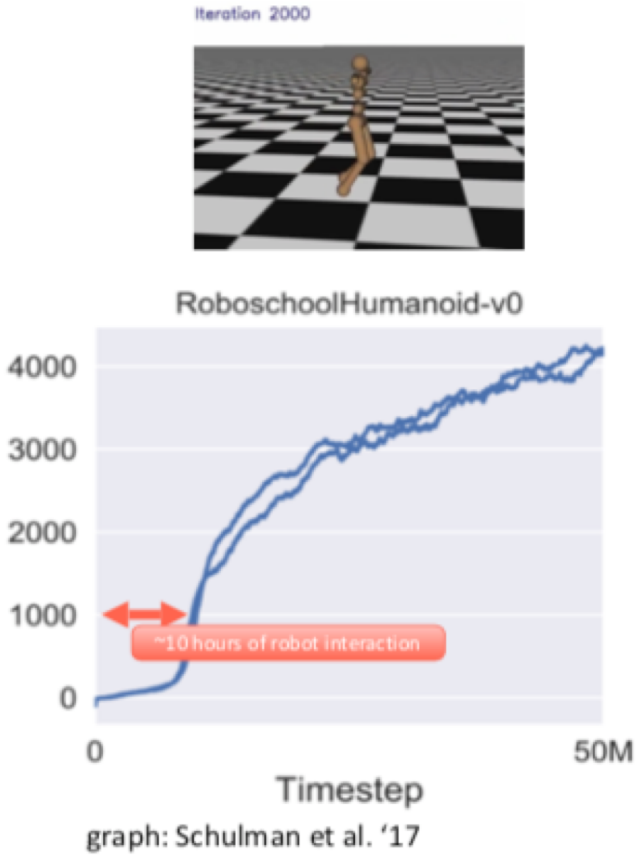
5 hours



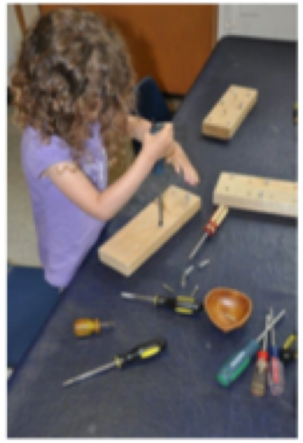
1 day

1 week

Why is my artificial mental retardation so slow?



graph: Schulman et al. '17



people can learn new skills **extremely** quickly

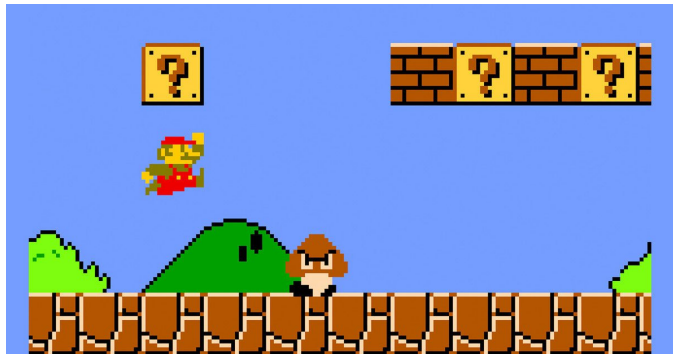
how?

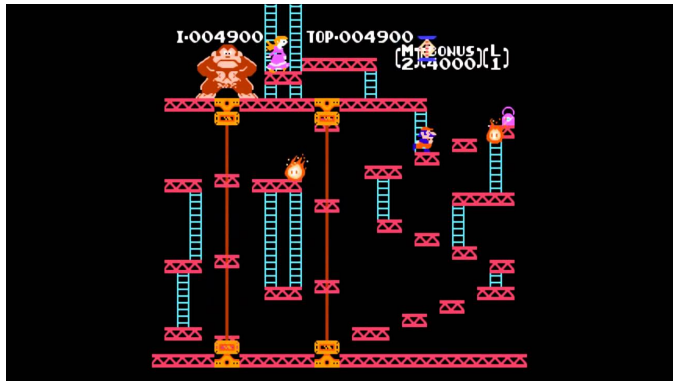
we never learn from scratch!

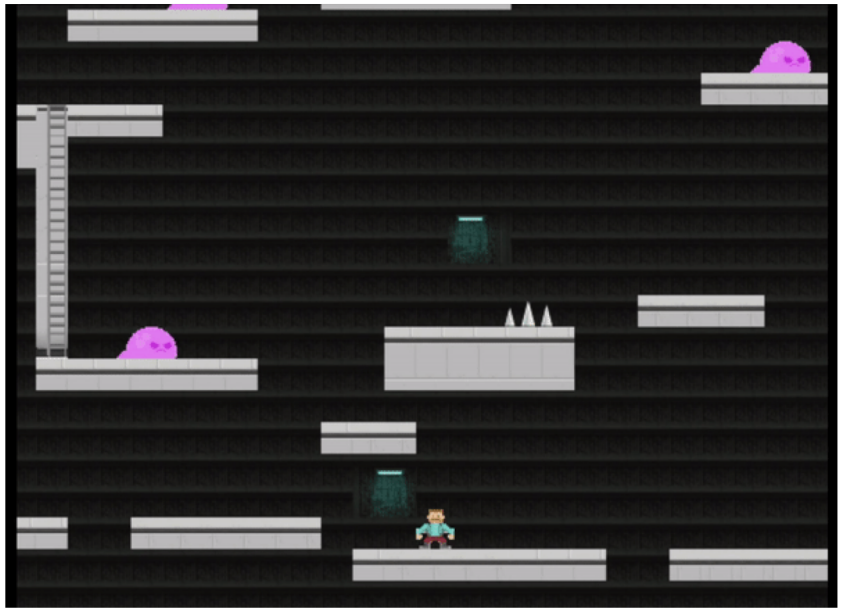
artificial mental retardation == 人工智障

How to play this game?

Seems to have played a similar game

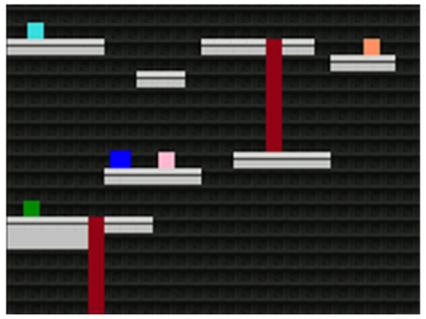




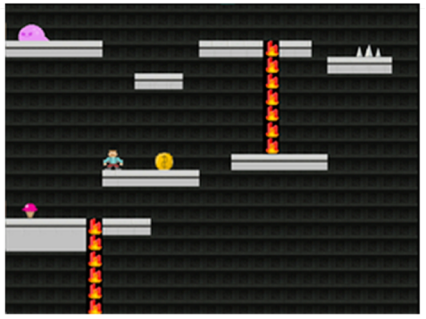


But some things are not the same

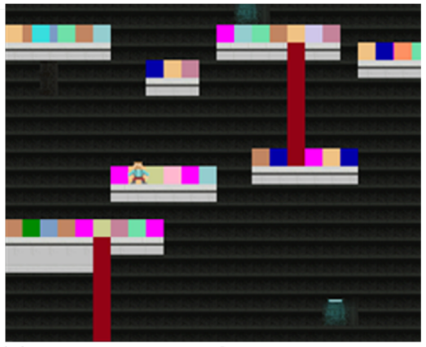
Masked
semantics



Reverse
Semantics

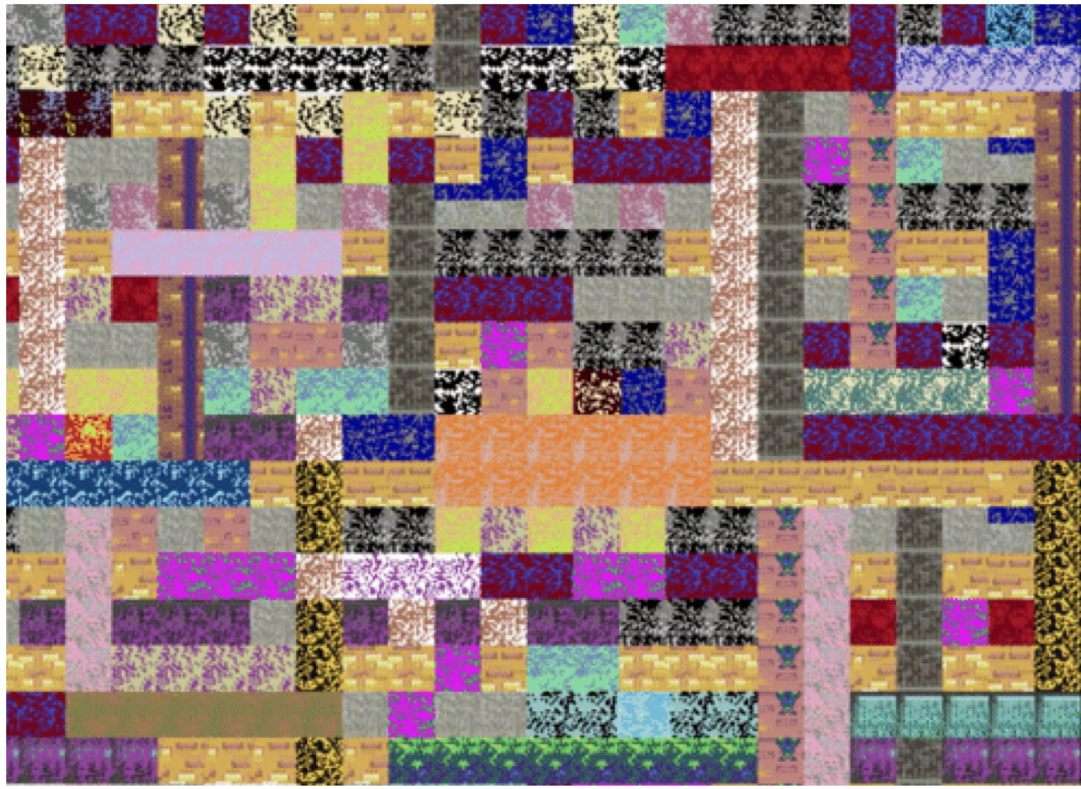


Masked identity
of objects



WTF...

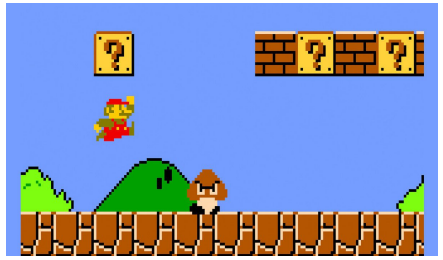
Game with all object priors removed

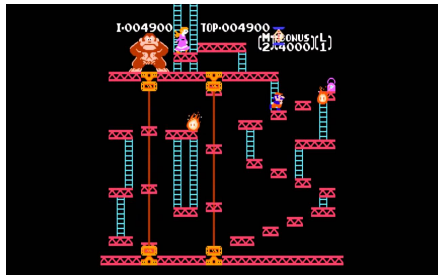


Meta-Learning

Can we explicitly learn priors from previous experience that learn faster?

Can we learn to learn?





To learn intelligent, **general-purpose** behavior, learning each new skill from scratch isn't going to cut it.

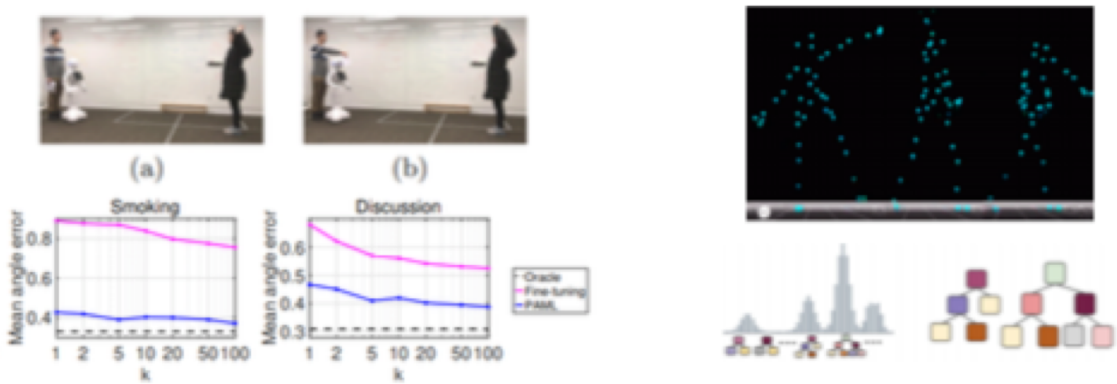
meta-learning ⟷ learning priors, structure such that learning new tasks is fast

Applications in computer vision

few-shot image recognition

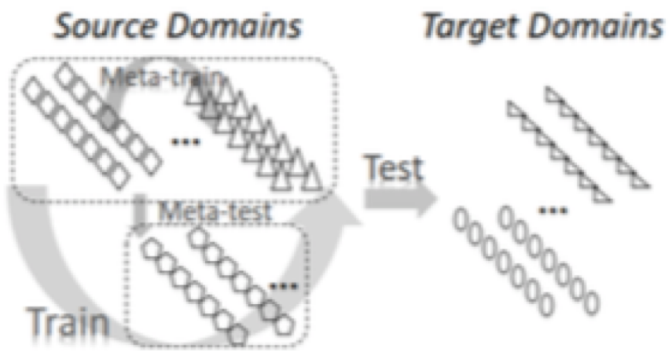
see, e.g.: Vinyals et al. **Matching Networks for One Shot Learning**, and many many others

human motion and pose prediction



see, e.g.: Gui et al. **Few-Shot Human Motion Prediction via Meta-Learning.**
Alet et al. **Modular Meta-Learning.**

domain adaptation



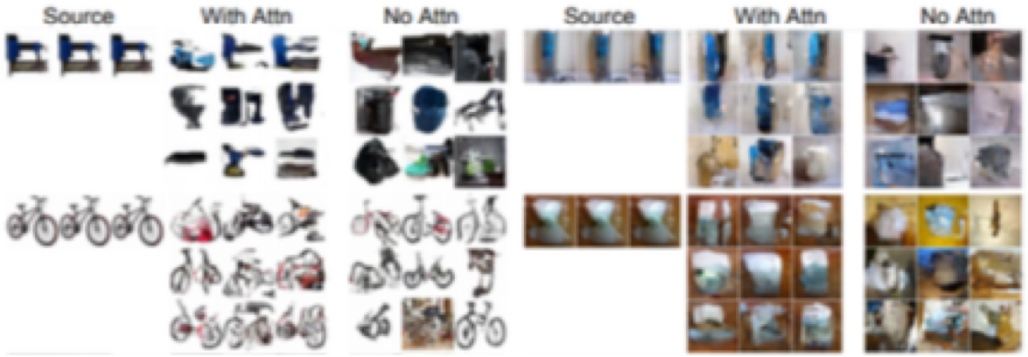
see, e.g.: Li, Yang, Song, Hospedales. **Learning to Generalize: Meta-Learning for Domain Adaptation.**

few-shot segmentation

see, e.g.: Shaban, Bansal, Liu, Essa, Boots. **One-Shot Learning for Semantic Segmentation.**
Rakelly, Shelhamer, Darrell, Efros, Levine. **Few-Shot Segmentation Propagation with Guided Networks.**
Dong, Xing. **Few-Shot Semantic Segmentation with Prototype Learning.**

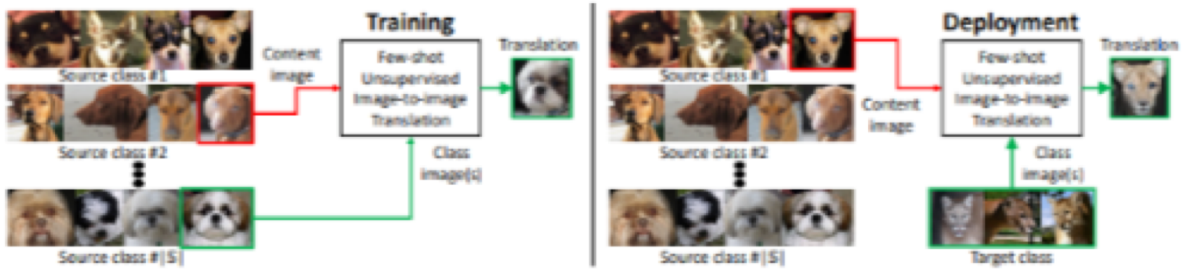
Applications in image & video generation

few-shot image generation



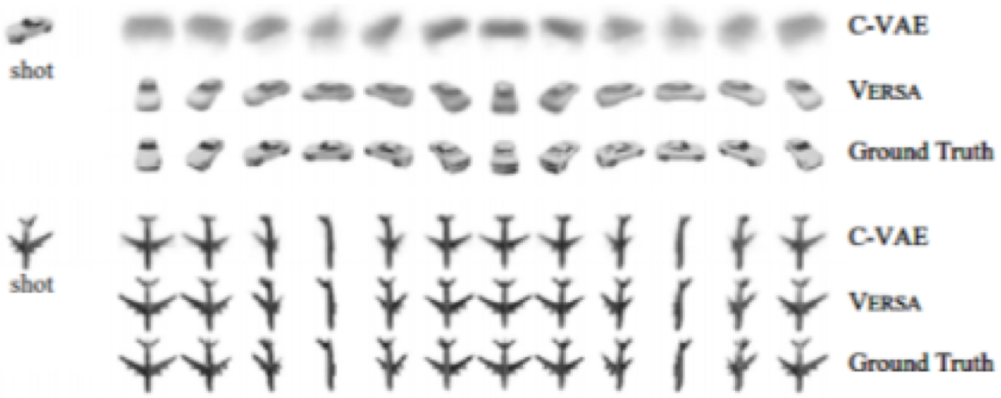
see, e.g.: Reed, Chen, Paine, van den Oord, Eslami, Rezende, Vinyals, de Freitas. **Few-Shot Autoregressive Density Estimation**. and many many others.

few-shot image-to-image translation



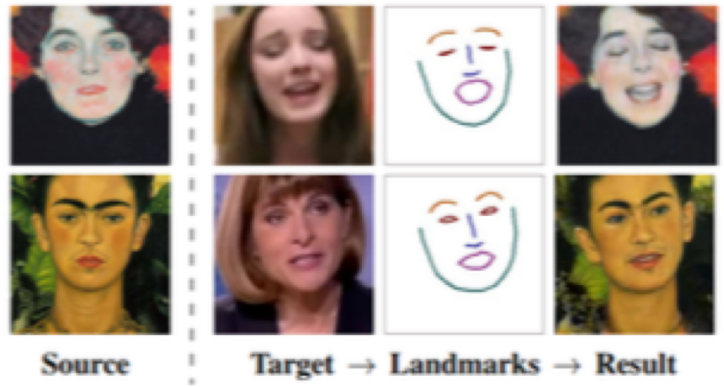
see, e.g.: Liu, Huang, Mallya, Karras, Aila, Lehtinen, Kautz. **Few-Shot Unsupervised Image-to-Image Translation.**

generation of novel viewpoints



see, e.g.: Gordon, Bronskill, Bauer, Nowozin, Turner. **VERSA: Versatile and Efficient Few-Shot Learning.**

generating talking heads from images



see, e.g.: Zakharov, Shysheya, Burkov, Lempitsky. **Few-Shot Adversarial Learning of Realistic Neural Talking Head Models**

Meta-Learning for ***Language***

Adapting to *new programs*

Meta Program Induction

Learn new program from a few I/O examples.

Devlin, Bunel* et al. NeurIPS '17*

Program Synthesis

Question:

How many CFL teams are from York College?

SQL:

```
SELECT COUNT CFL Team FROM
CFLDraft WHERE College = "York"
```

Result:

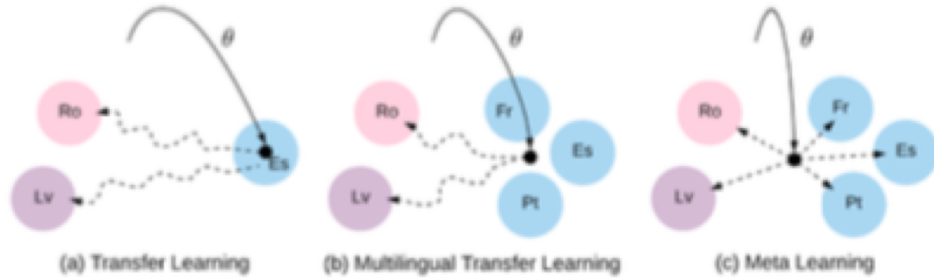
2

Construct pseudo-tasks with relevance function

Huang et al. NAACL '18

Adapting to *new languages*

Low-Resource Neural Machine Translation



Learn to translate new language pair w/o a lot of paired data?

Gu et al. EMNLP '18

Learning *new words*

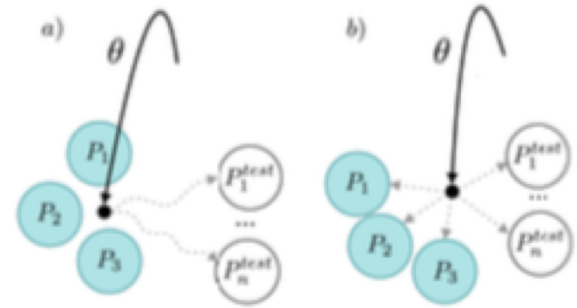
One-Shot Language Modeling

Learn how to use a new word from one example usage.

Vinyals et al. Matching Networks, '16

Adapting to *new personas*

Personalizing Dialogue Agents

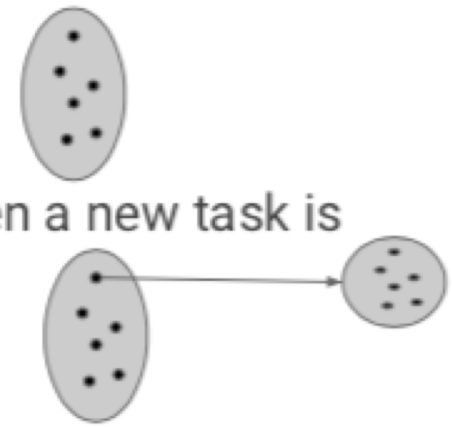


Adapt dialogue to a persona with a few examples

Lin, Madotto* et al. ACL '19*

Definition of Meta Learning

- What is Meta Learning / Learning to Learn?
  - Go beyond train from samples from a single distribution.
  - Distribution over tasks, so model has to “learn to learn” when a new task is presented



“... a system that improves or discovers a learning algorithm”

Hochreiter et al, ‘01

Meta-Learning

Two ways to view meta-learning

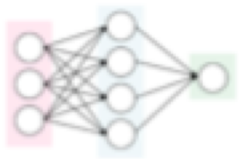
Mechanistic view

- Deep neural network model that can read in an entire dataset and make predictions for new datapoints
- Training this network uses a meta-dataset, which itself consists of many datasets, each for a different task
- This view makes it easier to implement meta-learning algorithms

Probabilistic view

- Extract prior information from a set of (meta-training) tasks that allows efficient learning of new tasks
- Learning a new task uses this prior and (small) training set to infer most likely posterior parameters
- This view makes it easier to understand meta-learning algorithms

Supervised Learning



supervised learning:

$$\arg\max_{\phi} \log p(\phi|\mathcal{D})$$

model parameters

training data

$$\mathcal{D} = \{(x_1, y_1), \ldots, (x_k, y_k)\}$$

input (e.g., image)

label

$$= \arg\max_{\phi} \log p(\mathcal{D}|\phi) + \log p(\phi)$$

data likelihood

regularizer (e.g., weight decay)

$$= \arg\max_{\phi} \sum_i \log p(y_i|x_i, \phi) + \log p(\phi)$$

What is wrong with this?

- The most powerful models typically require large amounts of labeled data
- Labeled data for some tasks may be very limited

Supervised Learning

supervised learning:

$$\arg\max_{\phi} \log p(\phi|\mathcal{D}) \qquad \mathcal{D} = \{(x_1, y_1), \ldots, (x_k, y_k)\}$$

can we incorporate *additional* data?

$$\mathcal{D}_{\text{meta-train}} = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$$

$$\arg\max_{\phi} \log p(\phi|\mathcal{D}, \mathcal{D}_{\text{meta-train}}) \qquad \mathcal{D}_i = \{(x_1^i, y_1^i), \ldots, (x_k^i, y_k^i)\}$$

The meta-learning problem

meta-learning:

$$\arg\max_{\phi} \log p(\phi|\mathcal{D}, \mathcal{D}_{\text{meta-train}})$$

$$\mathcal{D} = \{(x_1, y_1), \ldots, (x_k, y_k)\}$$

$$\mathcal{D}_{\text{meta-train}} = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$$

$$\mathcal{D}_i = \{(x_1^i, y_1^i), \ldots, (x_k^i, y_k^i)\}$$

what if we don't want to keep $\mathcal{D}_{\text{meta-train}}$ around forever?

The meta-learning problem

learn *meta-parameters* θ: $p(\theta|\mathcal{D}_{\text{meta-train}})$

whatever we need to know about $\mathcal{D}_{\text{meta-train}}$ to solve new tasks

assume $\phi \perp\!\!\!\perp \mathcal{D}_{\text{meta-train}} | \theta$

$$\log p(\phi|\mathcal{D}, \mathcal{D}_{\text{meta-train}}) = \log \int_{\Theta} p(\phi|\mathcal{D}, \theta) p(\theta|\mathcal{D}_{\text{meta-train}}) d\theta$$

$$\approx \log p(\phi|\mathcal{D}, \theta^\star) + \log p(\theta^\star|\mathcal{D}_{\text{meta-train}})$$

$$\arg\max_{\phi} \log p(\phi|\mathcal{D}, \mathcal{D}_{\text{meta-train}}) \approx \arg\max_{\phi} \log p(\phi|\mathcal{D}, \theta^\star)$$

this is the meta-learning problem

$$\theta^\star = \arg\max_{\theta} \log p(\theta|\mathcal{D}_{\text{meta-train}})$$

Closely related problem settings

meta-learning:

$$\theta^\star = \max_\theta \sum_{i=1}^{n} \log p(\phi_i | \mathcal{D}_i^{\text{ts}})$$

where $\phi_i = f_\theta(\mathcal{D}_i^{\text{tr}})$

$$\mathcal{D}_{\text{meta-train}} = \{(\mathcal{D}_1^{\text{tr}}, \mathcal{D}_1^{\text{ts}}), \ldots, (\mathcal{D}_n^{\text{tr}}, \mathcal{D}_n^{\text{ts}})\}$$

$$\mathcal{D}_i^{\text{tr}} = \{(x_1^i, y_1^i), \ldots, (x_k^i, y_k^i)\}$$

$$\mathcal{D}_i^{\text{ts}} = \{(x_1^i, y_1^i), \ldots, (x_l^i, y_l^i)\}$$

multi-task learning: learn model with parameters $\theta^\star$ that solves multiple tasks $\theta^\star = \arg\max_\theta \sum_{i=1}^{n} \log p(\theta | \mathcal{D}_i)$

can be seen as special case where $\phi_i = \theta$ (i.e., $f_\theta(\mathcal{D}_i) = \theta$)

hyperparameter optimization & auto-ML: can be cast as meta-learning

hyperparameter optimization: θ = hyperparameters, ϕ = network weights

architecture search: θ = architecture, ϕ = network weights

very active area of research! but outside the scope of this tutorial

Meta-learning so far…

learn θ such that $\phi_i = f_\theta(\mathcal{D}_i^{\text{tr}})$ is good for $\mathcal{D}_i^{\text{ts}}$

Probabilistic view:

$$\theta^\star = \arg\max_\theta \sum_{i=1}^{n} \log p(\phi_i | \mathcal{D}_i^{\text{ts}})$$

where $\phi_i = f_\theta(\mathcal{D}_i^{\text{tr}})$

Deterministic view:

$$\theta^\star = \arg\min_\theta \sum_{i=1}^{n} \mathcal{L}(\phi_i, \mathcal{D}_i^{\text{ts}})$$

where $\phi_i = f_\theta(\mathcal{D}_i^{\text{tr}})$

$$\mathcal{D}_{\text{meta-train}} = \{(\mathcal{D}_1^{\text{tr}}, \mathcal{D}_1^{\text{ts}}), \ldots, (\mathcal{D}_n^{\text{tr}}, \mathcal{D}_n^{\text{ts}})\}$$

$$\mathcal{D}_i^{\text{tr}} = \{(x_1^i, y_1^i), \ldots, (x_k^i, y_k^i)\}$$

$$\mathcal{D}_i^{\text{ts}} = \{(x_1^i, y_1^i), \ldots, (x_l^i, y_l^i)\}$$

The meta reinforcement learning problem

"Generic" learning (deterministic view):

$$\theta^\star = \arg\min_\theta \mathcal{L}(\theta, \mathcal{D}^{\text{tr}})$$

$$= f_{\text{learn}}(\mathcal{D}^{\text{tr}})$$

"Generic" meta-learning (deterministic view):

$$\theta^\star = \arg\min_\theta \sum_{i=1}^{n} \mathcal{L}(\phi_i, \mathcal{D}_i^{\text{ts}})$$

$$\text{where } \phi_i = f_\theta(\mathcal{D}_i^{\text{tr}})$$

Reinforcement learning:

$$\theta^\star = \arg\max_\theta E_{\pi_\theta(\tau)}[R(\tau)]$$

$$= f_{\text{RL}}(\mathcal{M}) \qquad \mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{P}, r\}$$

MDP

Meta-reinforcement learning:

$$\theta^\star = \arg\max_\theta \sum_{i=1}^{n} E_{\pi_{\phi_i}(\tau)}[R(\tau)]$$

$$\text{where } \phi_i = f_\theta(\mathcal{M}_i)$$

MDP for task i

The meta reinforcement learning problem

$$\theta^\star = \arg\max_\theta \sum_{i=1}^{n} E_{\pi_{\phi_i}(\tau)}[R(\tau)]$$

where $\phi_i = f_\theta(\mathcal{M}_i)$

$\{\mathcal{M}_1, \ldots, \mathcal{M}_n\}$

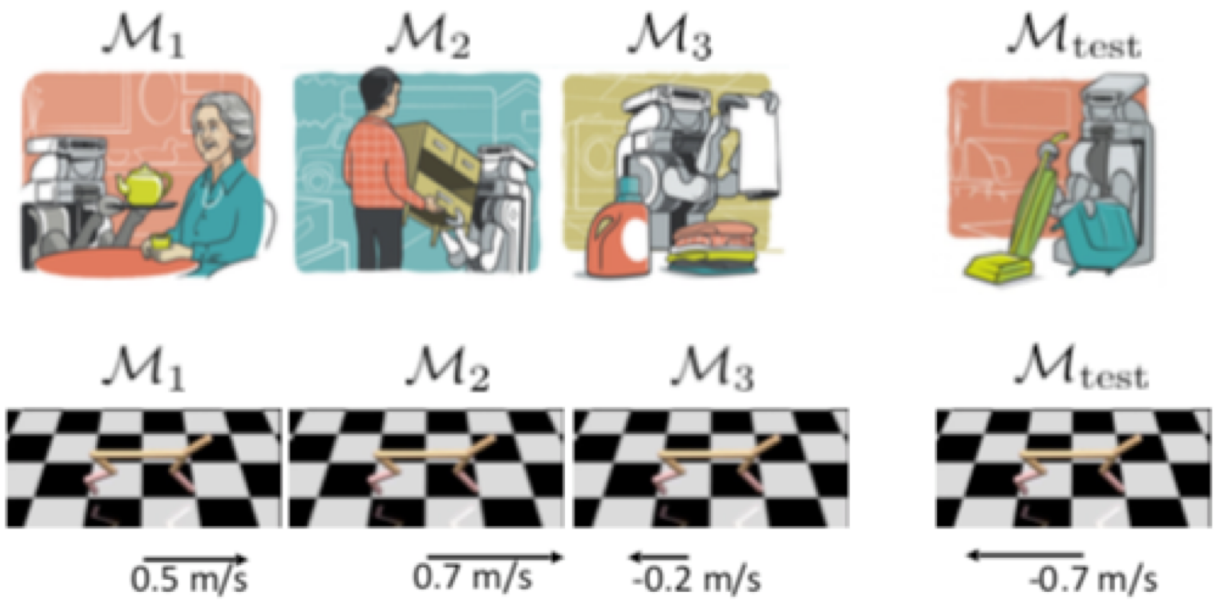
meta-training MDPs

assumption: $\mathcal{M}_i \sim p(\mathcal{M})$

meta test-time:

sample $\mathcal{M}_{\text{test}} \sim p(\mathcal{M})$, get $\phi_i = f_\theta(\mathcal{M}_{\text{test}})$

Some examples:

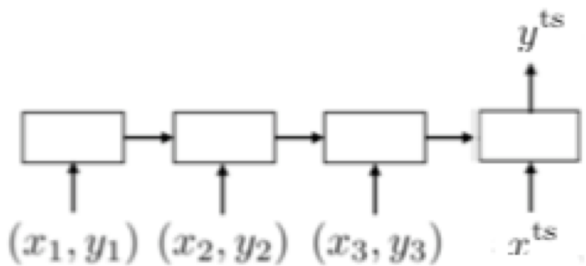


Amortized vs. Optimization vs. Non-Parametric

Computation graph perspective

Black-box amortized

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



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

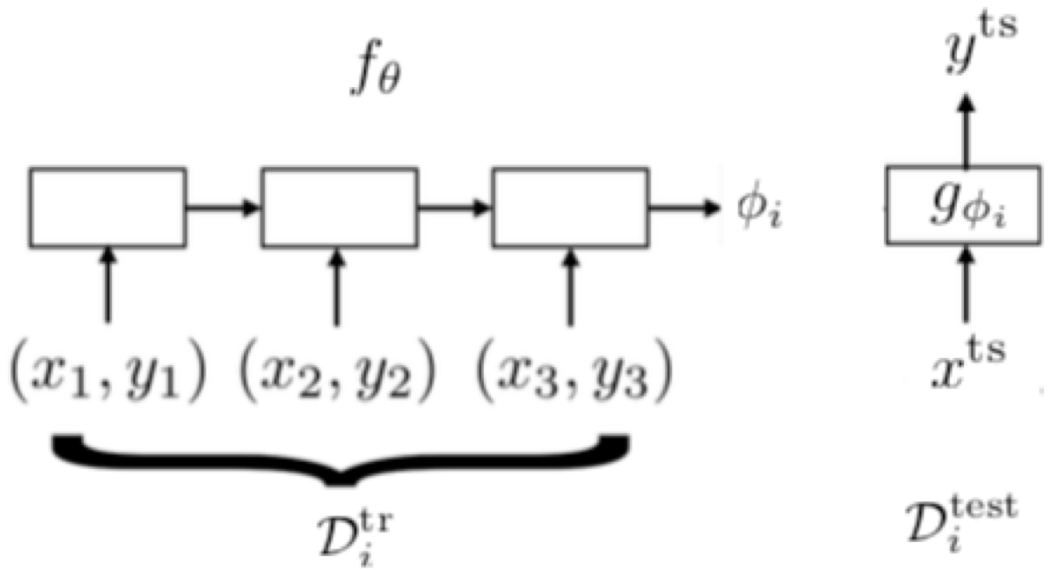
$$y^{\text{ts}} = f_{\text{PN}}(\mathcal{D}_i^{\text{tr}}, x^{\text{ts}}) = \text{softmax}(-d(f_\theta(x), c_k))$$

$$\text{where } c_k = \frac{1}{|\mathcal{D}_i^{\text{tr}}|} \sum_{(x,y) \in \mathcal{D}_i^{\text{tr}}} f_\theta(x)$$

Black-Box Adaptation

Key idea: Train a neural network to represent $p(\phi_i|\mathcal{D}_i^{\text{tr}}, \theta)$

For now: Use **deterministic** (point estimate) $\phi_i = f_\theta(\mathcal{D}_i^{\text{tr}})$



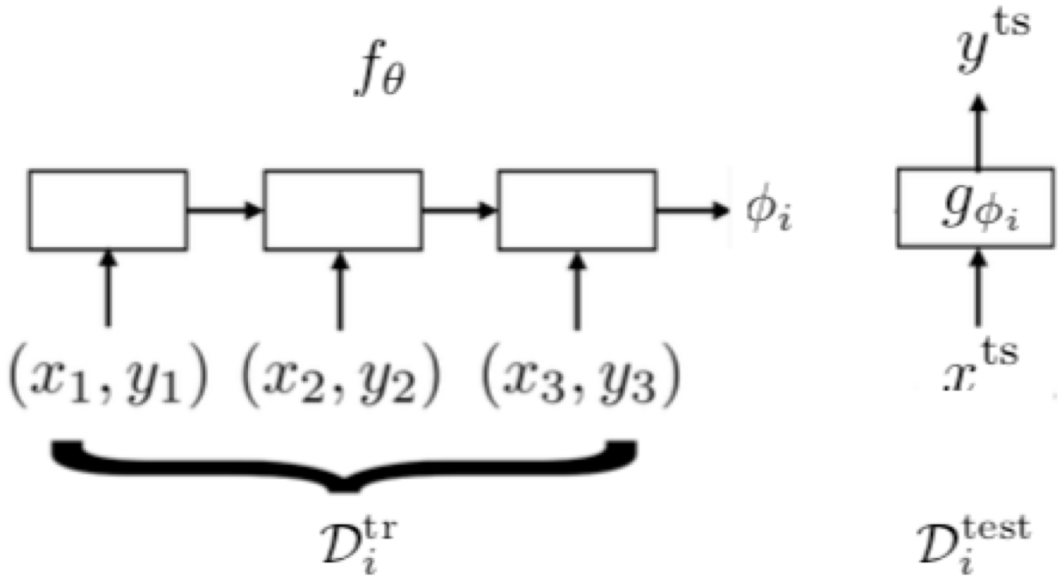
Train with standard supervised learning!

$$\max_\theta \sum_{\mathcal{T}_i} \underbrace{\sum_{(x,y)\sim\mathcal{D}_i^{\text{test}}} \log g_{\phi_i}(y|x)}_{\mathcal{L}(\phi_i, \mathcal{D}_i^{\text{test}})}$$

$$\max_\theta \sum_{\mathcal{T}_i} \mathcal{L}(f_\theta(\mathcal{D}_i^{\text{tr}}), \mathcal{D}_i^{\text{test}})$$

Black-Box Adaptation

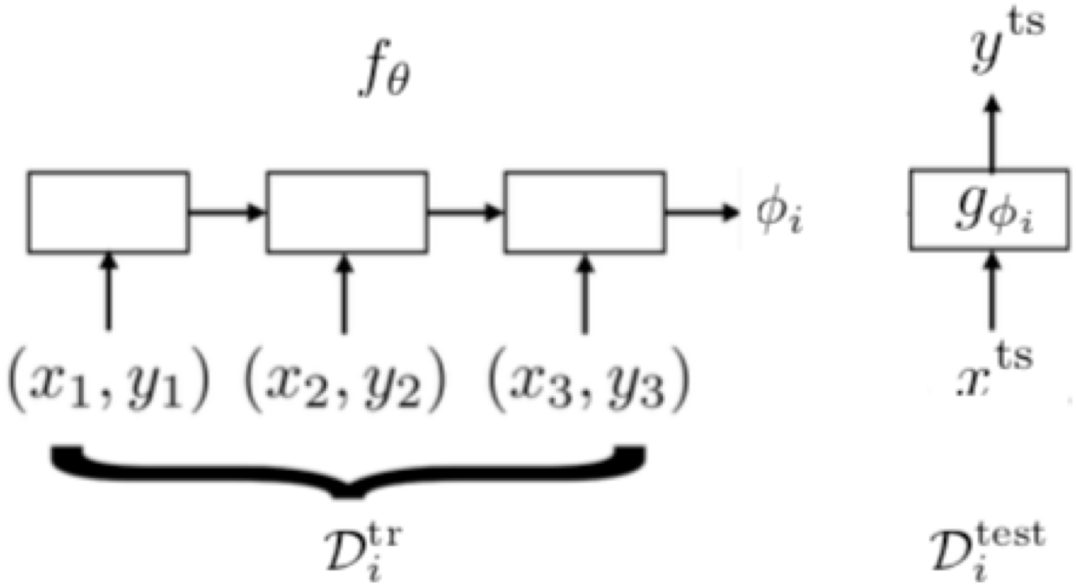
Key idea: Train a neural network to represent $p(\phi_i | \mathcal{D}_i^{\text{tr}}, \theta)$



1. Sample task $\mathcal{T}_i$ *(or mini batch of tasks)*
2. Sample disjoint datasets $\mathcal{D}_i^{\text{tr}}, \mathcal{D}_i^{\text{test}}$ from $\mathcal{D}_i$
3. Compute $\phi_i \leftarrow f_\theta(\mathcal{D}_i^{\text{tr}})$
4. Update θ using $\nabla_\theta \mathcal{L}(\phi_i, \mathcal{D}_i^{\text{test}})$

Black-Box Adaptation

Key idea: Train a neural network to represent $p(\phi_i | \mathcal{D}_i^{\text{tr}}, \theta)$



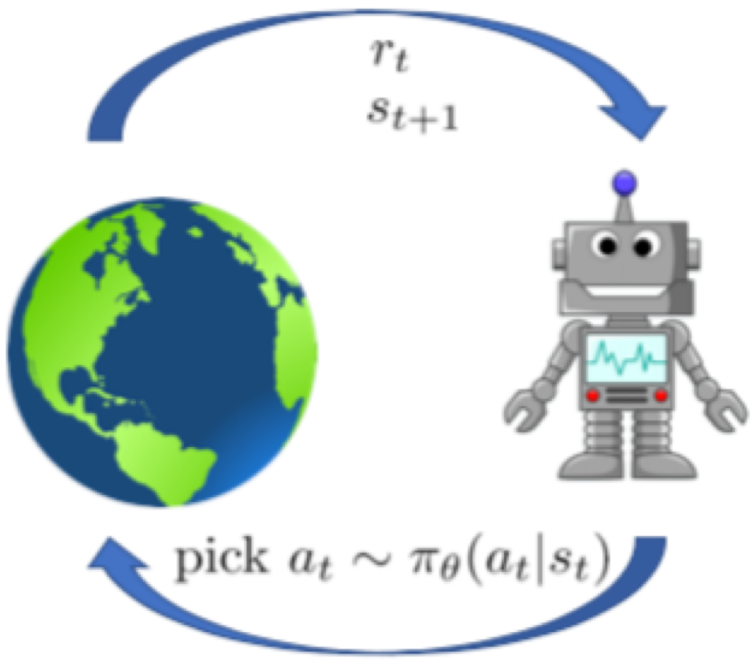
Form of f_θ ?

- LSTM
- Neural turing machine (NTM)
- Self-attention
- 1D convolutions
- feedforward + average

Meta-RL with recurrent policies

$$\theta^\star = \arg\max_\theta \sum_{i=1}^{n} E_{\pi_{\phi_i}(\tau)}[R(\tau)]$$

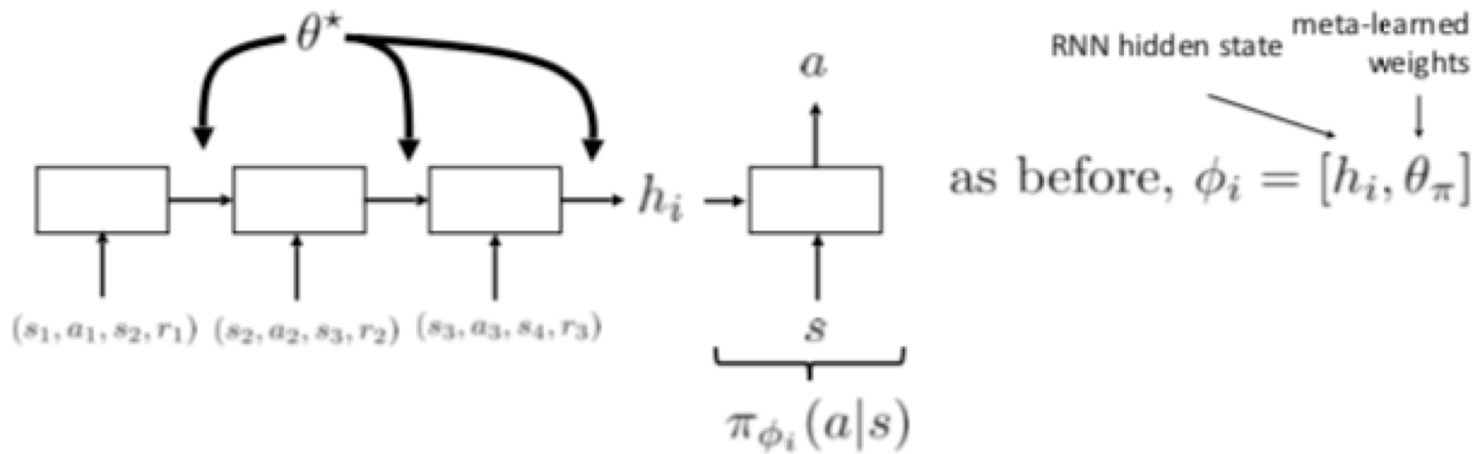
where $\phi_i = f_\theta(\mathcal{M}_i)$



main question: how to implement $f_\theta(\mathcal{M}_i)$?

what should $f_\theta(\mathcal{M}_i)$ *do*?

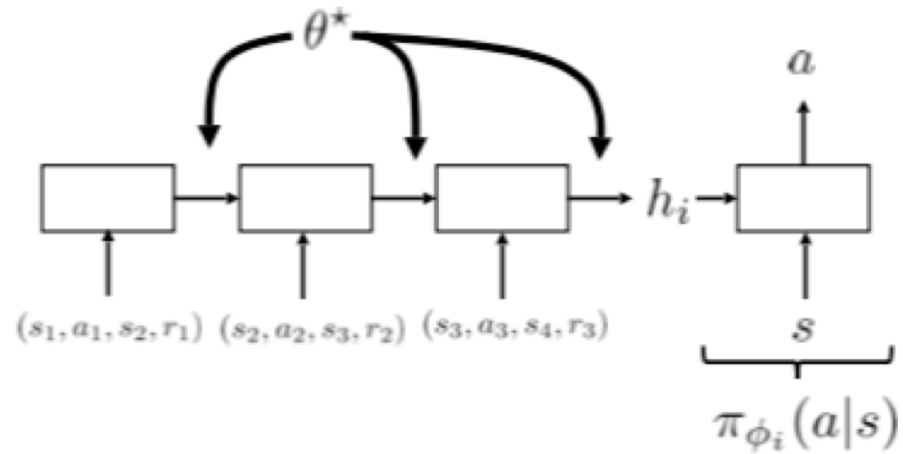
1. improve policy with experience from $\mathcal{M}_i$
   $\{(s_1, a_1, s_2, r_1), \ldots, (s_T, a_T, s_{T+1}, r_T)\}$
2. (new in RL): choose how to interact, i.e. choose a_t
   meta-RL must also *choose* how to *explore*!



Meta-RL with recurrent policies

$$\theta^\star = \arg\max_\theta \sum_{i=1}^{n} E_{\pi_{\phi_i}(\tau)}[R(\tau)]$$

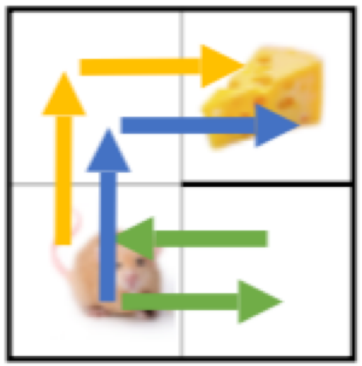
where $\phi_i = f_\theta(\mathcal{M}_i)$

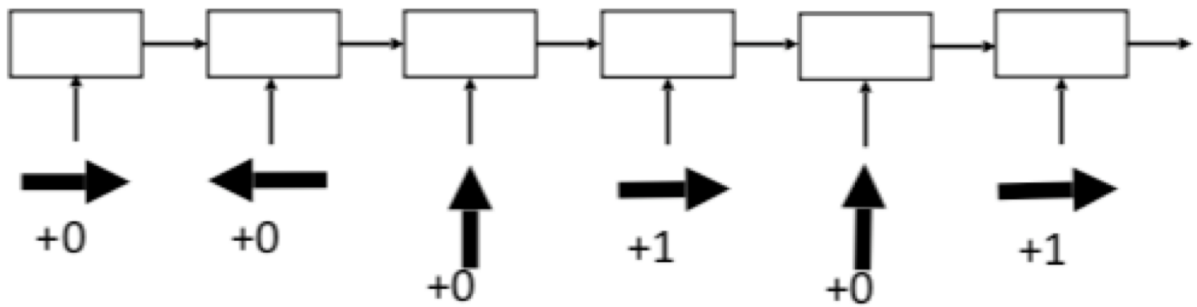


so... we just train an RNN policy?

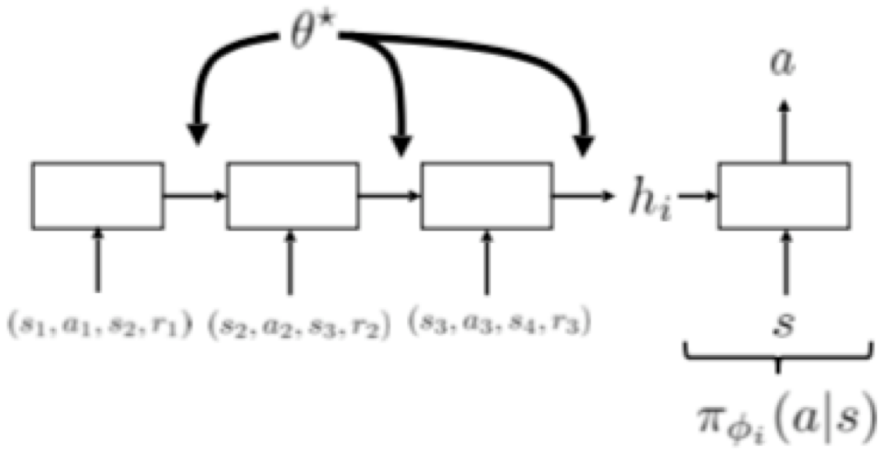
yes!

crucially, RNN hidden state is **not** reset between episodes!





Why recurrent policies learn to explore



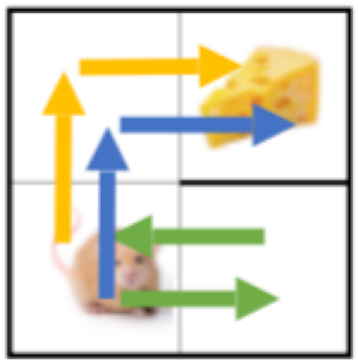
1. improve policy with experience from $\mathcal{M}_i$

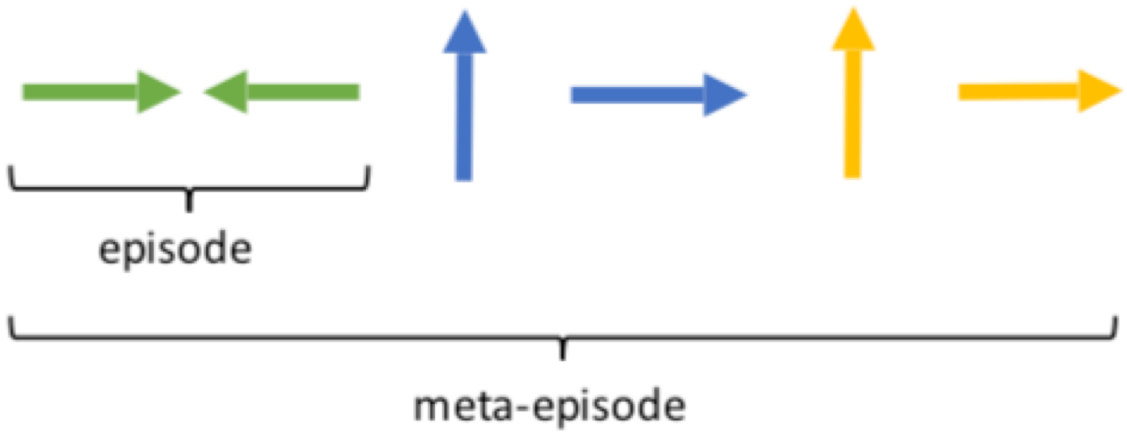
   $\{(s_1, a_1, s_2, r_1), \ldots, (s_T, a_T, s_{T+1}, r_T)\}$

2. (new in RL): choose how to interact, i.e. choose a_t

   meta-RL must also *choose* how to *explore*!

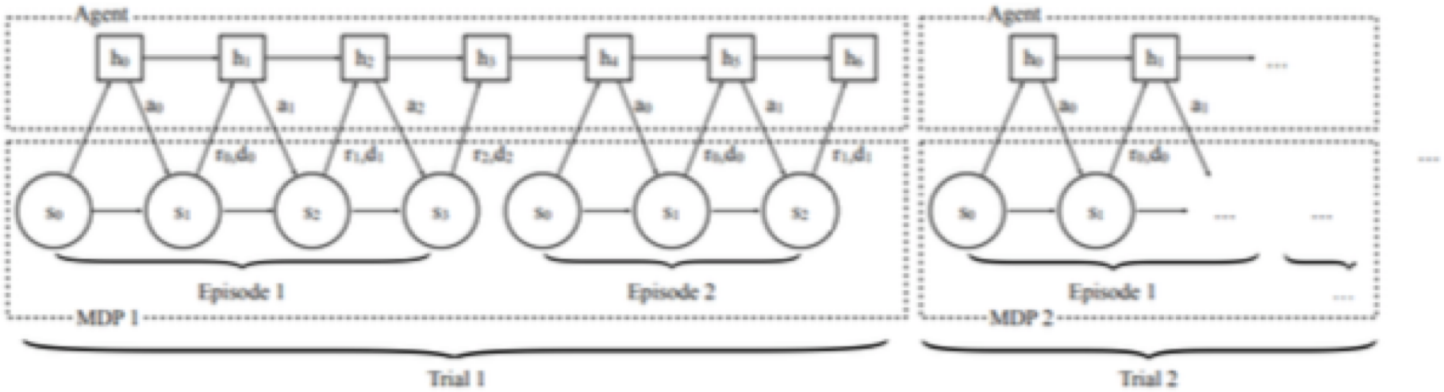
$$\theta^\star = \arg\max_\theta E_{\pi_\theta}\left[\sum_{t=0}^{T} r(s_t, a_t)\right]$$





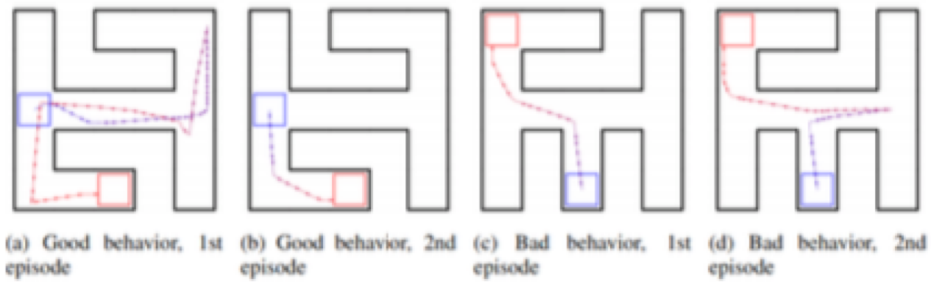
optimizing total reward over the entire **meta**-episode with RNN policy **automatically** learns to explore!

Architectures for meta-RL

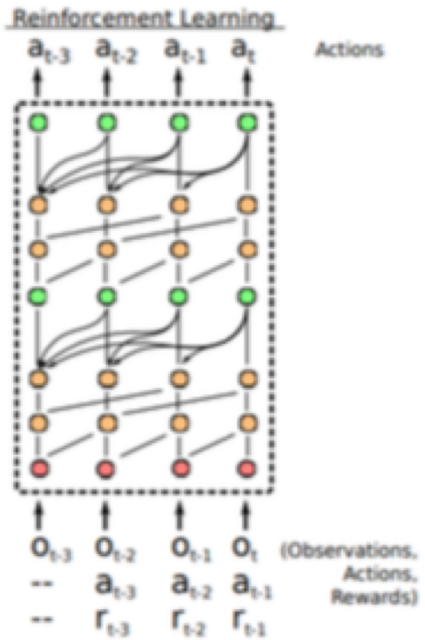


standard RNN (LSTM) architecture

Duan, Schulman, Chen, Bartlett, Sutskever, Abbeel. **RL2: Fast Reinforcement Learning via Slow Reinforcement Learning.** 2016.



(a) Good behavior, 1st episode (b) Good behavior, 2nd episode (c) Bad behavior, 1st episode (d) Bad behavior, 2nd episode



attention + temporal convolution

Mishra, Rohaninejad, Chen, Abbeel. **A Simple Neural Attentive Meta-Learner.**

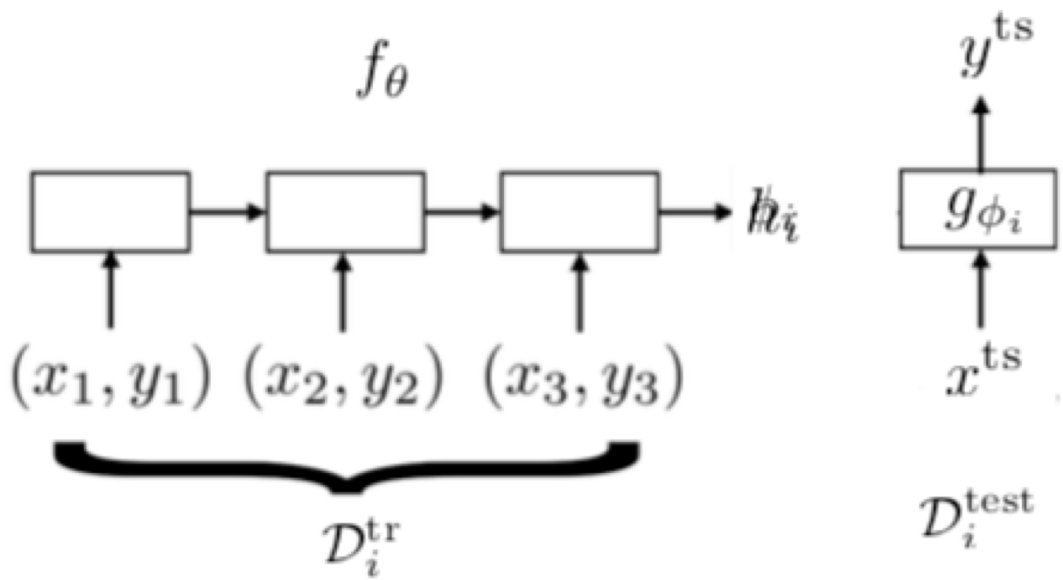
Black-Box Adaptation

Key idea: Train a neural network to represent $p(\phi_i | \mathcal{D}_i^{\text{tr}}, \theta)$

Challenges

Outputting all neural net parameters does not seem scalable?

Idea: Do not need to output **all** parameters of neural net, only sufficient statistics (Santoro et al. MANN, Mishra et al. SNAIL)



low-dimensional vector h_i
represents contextual task information

$$\phi_i = \{h_i, \theta_g\}$$

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

Is there a way to infer **all parameters** in a scalable way?

Optimization-Based Inference

Key idea: Acquire ϕ_i through optimization.

$$\max_{\phi_i} \log p(\mathcal{D}_i^{\text{tr}}|\phi_i) + \log p(\phi_i|\theta)$$

Meta-parameters θ serve as a prior. What form of prior?

One successful form of prior knowledge: **initialization** for **fine-tuning**

Optimization-Based Inference

Fine-tuning [test-time]

$$\phi \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\theta, \mathcal{D}^{\text{tr}})$$

pre-trained parameters

training data for new task

Meta-learning

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

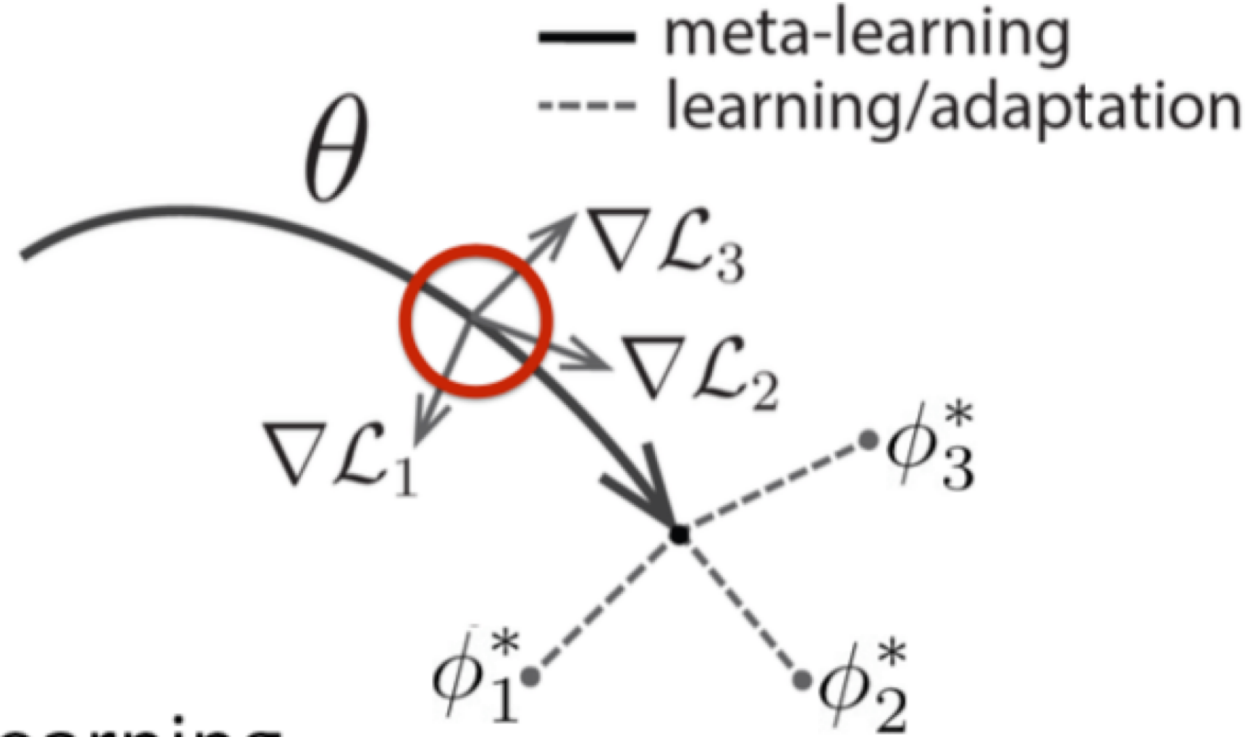
Key idea: Over many tasks, learn parameter vector θ that transfers via fine-tuning

Optimization-Based Inference

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

θ parameter vector being meta-learned

ϕ_i^* optimal parameter vector for task i



Model-**A**gnostic **M**eta-**L**earning

Optimization-Based Inference

Key idea: Acquire ϕ_i through optimization.

General Algorithm:

~~Amortized approach~~ Optimization-based approach

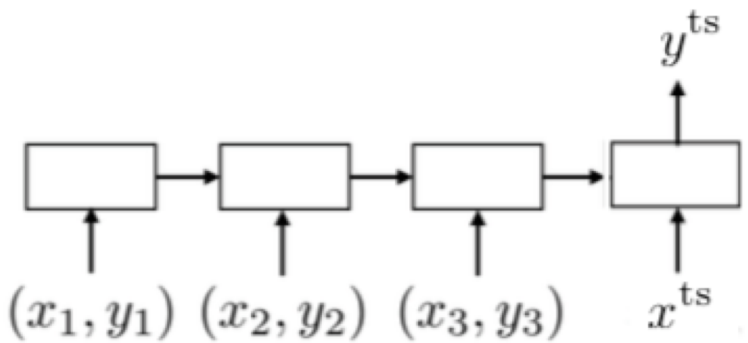
1. Sample task $\mathcal{T}_i$ *(or mini batch of tasks)*
2. Sample disjoint datasets $\mathcal{D}_i^{\text{tr}}, \mathcal{D}_i^{\text{test}}$ from $\mathcal{D}_i$
3. ~~Compute $\phi_i \leftarrow f_\theta(\mathcal{D}_i^{\text{tr}})$~~ Optimize $\phi_i \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(\theta, \mathcal{D}_i^{\text{tr}})$
4. Update θ using $\nabla_\theta \mathcal{L}(\phi_i, \mathcal{D}_i^{\text{test}})$

—> brings up **second-order** derivatives (more on this later)

Optimization vs. Black-Box Adaptation

Black-box adaptation

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



Model-agnostic meta-learning

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

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

MAML can be viewed as **computation graph**, with embedded gradient operator

Note: Can mix & match components of computation graph

Learn initialization but replace gradient update with learned network

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

Meta-RL as an optimization problem

$$\theta^\star = \arg\max_\theta \sum_{i=1}^{n} E_{\pi_{\phi_i}(\tau)}[R(\tau)]$$

where $\phi_i = f_\theta(\mathcal{M}_i)$

1. improve policy with experience from $\mathcal{M}_i$

$\{(s_1, a_1, s_2, r_1), \ldots, (s_T, a_T, s_{T+1}, r_T)\}$

what if $f_\theta(\mathcal{M}_i)$ is *itself* an RL algorithm?

$$f_\theta(\mathcal{M}_i) = \theta + \alpha \nabla_\theta \underbrace{J_i(\theta)}$$

requires interacting with $\mathcal{M}_i$ to estimate $\nabla_\theta E_{\pi_\theta}[R(\tau)]$

standard RL:

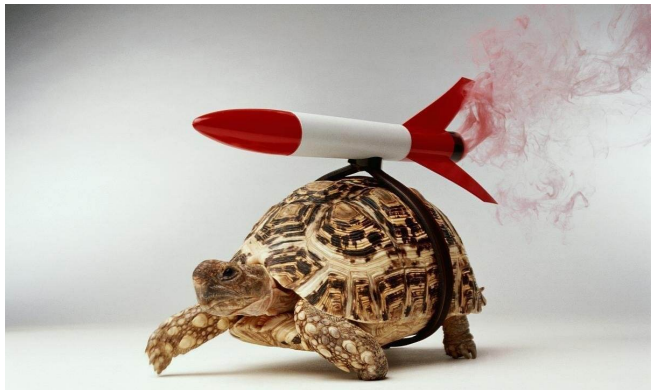
$$\theta^\star = \arg\max_\theta \underbrace{E_{\pi_\theta(\tau)}[R(\tau)]}_{J(\theta)}$$

$$\theta^{k+1} \leftarrow \theta_k + \alpha \nabla_{\theta^k} J(\theta^k)$$

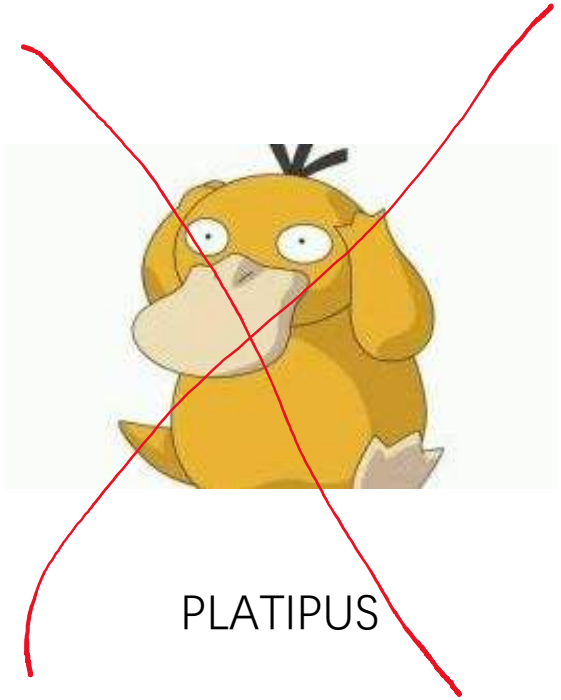
this is model-agnostic meta-learning (MAML) for RL!

Meta(-RL) Zoo

MAML



Reptile



PLATIPUS

CAVIA

First-Order MAML

The meta-optimization step above relies on second derivatives. To make the computation less expensive, a modified version of MAML omits second derivatives, resulting in a simplified and cheaper implementation, known as **First-Order MAML (FOMAML)**.

Let's consider the case of performing k inner gradient steps, $k \geq 1$. Starting with the initial model parameter θ_{meta}:

$$
\begin{aligned}
\theta_0 &= \theta_{\text{meta}} \\
\theta_1 &= \theta_0 - \alpha \nabla_\theta \mathcal{L}^{(0)}(\theta_0) \\
\theta_2 &= \theta_1 - \alpha \nabla_\theta \mathcal{L}^{(0)}(\theta_1) \\
&\dots \\
\theta_k &= \theta_{k-1} - \alpha \nabla_\theta \mathcal{L}^{(0)}(\theta_{k-1})
\end{aligned}
$$

Then in the outer loop, we sample a new data batch for updating the meta-objective.

First-Order MAML

$$
\begin{aligned}
\theta_0 &= \theta_{\text{meta}} \\
\theta_1 &= \theta_0 - \alpha \nabla_\theta \mathcal{L}^{(0)}(\theta_0) \\
\theta_2 &= \theta_1 - \alpha \nabla_\theta \mathcal{L}^{(0)}(\theta_1) \\
&\dots \\
\theta_k &= \theta_{k-1} - \alpha \nabla_\theta \mathcal{L}^{(0)}(\theta_{k-1})
\end{aligned}
$$

$$
\begin{aligned}
g_{\text{MAML}} &= \nabla_\theta \mathcal{L}^{(1)}(\theta_k) \\
&= \nabla_{\theta_k} \mathcal{L}^{(1)}(\theta_k) \cdot (\nabla_{\theta_{k-1}} \theta_k) \dots (\nabla_{\theta_0} \theta_1) \cdot (\nabla_\theta \theta_0) && \text{; following the chain rule} \\
&= \nabla_{\theta_k} \mathcal{L}^{(1)}(\theta_k) \cdot \prod_{i=1}^{k} \nabla_{\theta_{i-1}} \theta_i \\
&= \nabla_{\theta_k} \mathcal{L}^{(1)}(\theta_k) \cdot \prod_{i=1}^{k} \nabla_{\theta_{i-1}} (\theta_{i-1} - \alpha \nabla_\theta \mathcal{L}^{(0)}(\theta_{i-1})) \\
&= \nabla_{\theta_k} \mathcal{L}^{(1)}(\theta_k) \cdot \prod_{i=1}^{k} (I - \alpha \nabla_{\theta_{i-1}} (\nabla_\theta \mathcal{L}^{(0)}(\theta_{i-1})))
\end{aligned}
$$

The First-Order MAML ignores the second derivative part in red. It is simplified as follows, equivalent to the derivative of the last inner gradient update result.

$$
g_{\text{FOMAML}} = \nabla_{\theta_k} \mathcal{L}^{(1)}(\theta_k)
$$

Reptile

The Reptile works by repeatedly:

- 1) sampling a task,
- 2) training on it by multiple gradient descent steps,
- 3) and then moving the model weights towards the new parameters.

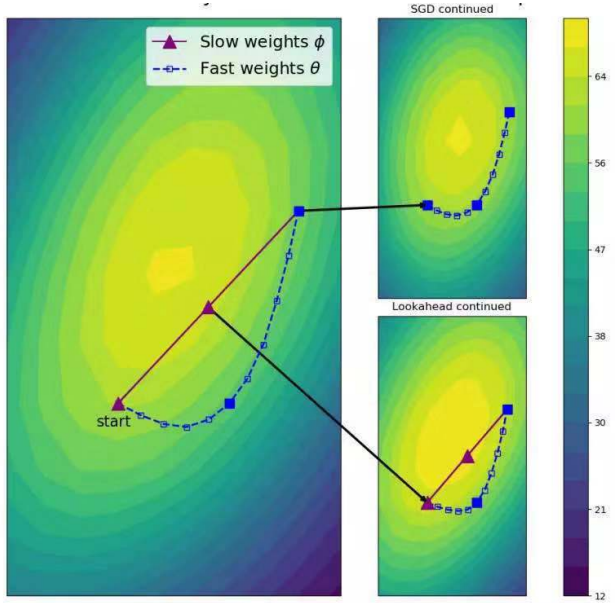
Algorithm 2 Reptile, batched version

Initialize θ
for iteration $= 1, 2, \ldots$ **do**
  Sample tasks $\tau_1, \tau_2, \ldots, \tau_n$
  for $i = 1, 2, \ldots, n$ **do**
    Compute $W_i = \mathrm{SGD}(L_{\tau_i}, \theta, k)$
  end for
  Update $\theta \leftarrow \theta + \beta \frac{1}{n} \sum_{i=1}^{n} (W_i - \theta)$
end for

有大佬真是好！

Algorithm 2 Reptile, batched version

Initialize θ
for iteration $= 1, 2, \ldots$ **do**
  Sample tasks $\tau_1, \tau_2, \ldots, \tau_n$
  for $i = 1, 2, \ldots, n$ **do**
    Compute $W_i = \mathrm{SGD}(L_{\tau_i}, \theta, k)$
  end for
  Update $\theta \leftarrow \theta + \beta \frac{1}{n} \sum_{i=1}^{n} (W_i - \theta)$
end for



Lookahead Optimizer: k steps forward, 1 step back

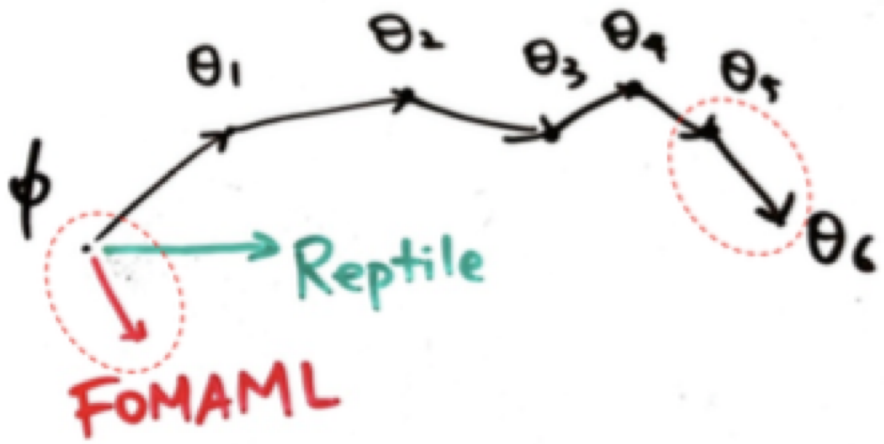
Michael R. Zhang **James Lucas** **Geoffrey Hinton**

Jimmy Ba
Department of Computer Science, University of Toronto, Vector Institute
{michael, jlucas, hinton,jba}@cs.toronto.edu

Algorithm 1 Lookahead Optimizer:

Require: Initial parameters ϕ_0, objective function L
Require: Synchronization period k, slow weights step size α, optimizer A
for $t = 1, 2, \ldots$ **do**
  Synchronize parameters $\theta_{t,0} \leftarrow \phi_{t-1}$
  for $i = 1, 2, \ldots, k$ **do**
    sample minibatch of data $d \sim \mathcal{D}$
    $\theta_{t,i} \leftarrow \theta_{t,i-1} + A(L, \theta_{t,i-1}, d)$
  end for
  Perform outer update $\phi_t \leftarrow \phi_{t-1} + \alpha(\theta_{t,k} - \phi_{t-1})$
end for
return parameters ϕ

Reptile vs FOMAML



simplified annotations: $g_j^{(i)} = \nabla_\theta \mathcal{L}^{(i)}(\theta_j)$ and $H_j^{(i)} = \nabla_\theta^2 \mathcal{L}^{(i)}(\theta_j)$.

$$\theta_0 = \theta_{\text{meta}}$$
$$\theta_1 = \theta_0 - \alpha \nabla_\theta \mathcal{L}^{(0)}(\theta_0) = \theta_0 - \alpha g_0^{(0)}$$
$$\theta_2 = \theta_1 - \alpha \nabla_\theta \mathcal{L}^{(1)}(\theta_1) = \theta_0 - \alpha g_0^{(0)} - \alpha g_1^{(1)}$$

According to the early section, the gradient of FOMAML is the last inner gradient update result.
Therefore, when k=1:

$$g_{\text{FOMAML}} = \nabla_{\theta_1} \mathcal{L}^{(1)}(\theta_1) = g_1^{(1)}$$
$$g_{\text{MAML}} = \nabla_{\theta_1} \mathcal{L}^{(1)}(\theta_1) \cdot (I - \alpha \nabla_\theta^2 \mathcal{L}^{(0)}(\theta_0)) = g_1^{(1)} - \alpha H_0^{(0)} g_1^{(1)}$$

The Reptile gradient is defined as:

$$g_{\text{Reptile}} = (\theta_0 - \theta_2)/\alpha = g_0^{(0)} + g_1^{(1)}$$

Reptile vs FOMAML

$$g_{\text{FOMAML}} = g_1^{(1)}$$
$$g_{\text{MAML}} = g_1^{(1)} - \alpha H_0^{(0)} g_1^{(1)}$$
$$g_{\text{Reptile}} = g_0^{(0)} + g_1^{(1)}$$

Next let's try further expand $g_1^{(1)}$ using Taylor expansion. Recall that Taylor expansion of a function $f(x)$ that is differentiable at a number a is:

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots = \sum_{i=0}^{\infty} \frac{f^{(i)}(a)}{i!}(x-a)^i$$

We can consider $\nabla_\theta \mathcal{L}^{(1)}(\,.\,)$ as a function and θ_0 as a value point. The Taylor expansion of $g_1^{(1)}$ at the value point θ_0 is:

$$\begin{aligned}
g_1^{(1)} &= \nabla_\theta \mathcal{L}^{(1)}(\theta_1) \\
&= \nabla_\theta \mathcal{L}^{(1)}(\theta_0) + \nabla_\theta^2 \mathcal{L}^{(1)}(\theta_0)(\theta_1 - \theta_0) + \frac{1}{2}\nabla_\theta^3 \mathcal{L}^{(1)}(\theta_0)(\theta_1 - \theta_0)^2 + \ldots \\
&= g_0^{(1)} - \alpha H_0^{(1)} g_0^{(0)} + \frac{\alpha^2}{2}\nabla_\theta^3 \mathcal{L}^{(1)}(\theta_0)(g_0^{(0)})^2 + \ldots \qquad ; \text{because } \theta_1 - \theta_0 = -\alpha g_0^{(0)} \\
&= g_0^{(1)} - \alpha H_0^{(1)} g_0^{(0)} + O(\alpha^2)
\end{aligned}$$

Reptile vs FOMAML

Plug in the expanded form of $g_1^{(1)}$ into the MAML gradients with one step inner gradient update:

$$
\begin{aligned}
g_{\text{FOMAML}} &= g_1^{(1)} = g_0^{(1)} - \alpha H_0^{(1)} g_0^{(0)} + O(\alpha^2) \\
g_{\text{MAML}} &= g_1^{(1)} - \alpha H_0^{(0)} g_1^{(1)} \\
&= g_0^{(1)} - \alpha H_0^{(1)} g_0^{(0)} + O(\alpha^2) - \alpha H_0^{(0)} (g_0^{(1)} - \alpha H_0^{(1)} g_0^{(0)} + O(\alpha^2)) \\
&= g_0^{(1)} - \alpha H_0^{(1)} g_0^{(0)} - \alpha H_0^{(0)} g_0^{(1)} + \alpha^2 \alpha H_0^{(0)} H_0^{(1)} g_0^{(0)} + O(\alpha^2) \\
&= g_0^{(1)} - \alpha H_0^{(1)} g_0^{(0)} - \alpha H_0^{(0)} g_0^{(1)} + O(\alpha^2)
\end{aligned}
$$

The Reptile gradient becomes:

$$
\begin{aligned}
g_{\text{Reptile}} &= g_0^{(0)} + g_1^{(1)} \\
&= g_0^{(0)} + g_0^{(1)} - \alpha H_0^{(1)} g_0^{(0)} + O(\alpha^2)
\end{aligned}
$$

So far we have the formula of three types of gradients:

$$
\begin{aligned}
g_{\text{FOMAML}} &= g_0^{(1)} - \alpha H_0^{(1)} g_0^{(0)} + O(\alpha^2) \\
g_{\text{MAML}} &= g_0^{(1)} - \alpha H_0^{(1)} g_0^{(0)} - \alpha H_0^{(0)} g_0^{(1)} + O(\alpha^2) \\
g_{\text{Reptile}} &= g_0^{(0)} + g_0^{(1)} - \alpha H_0^{(1)} g_0^{(0)} + O(\alpha^2)
\end{aligned}
$$

CAVIA: Fast Context Adaptation via Meta-Learning

In the inner update loop, we update context parameters ϕ.

$$\phi_i = \phi_0 - \alpha \nabla_\phi \frac{1}{M_{train}^i} \sum_{(x,y)\in\mathcal{D}_i^{train}} \mathcal{L}_{\mathcal{T}_i}(f_{\phi_0,\theta}(x), y)$$

In the outer update loop, we update the global parameters θ.

$$\theta = \theta - \beta \nabla_\theta \frac{1}{N} \sum_{\mathcal{T}_i\in\mathbf{T}} \frac{1}{M_{test}^i} \sum_{(x,y)\in\mathcal{D}_i^{test}} \mathcal{L}_{\mathcal{T}_i}(f_{\phi,\theta_i}(x), y)$$

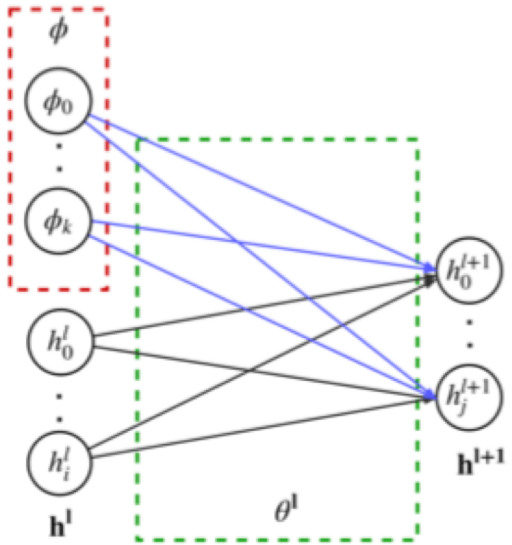


Figure 1. **Context adaptation.** A network layer h^l is augmented with additional context parameters ϕ (red) initialised to 0 before each adaptation step and updated by gradient descent during each inner loop and at test time. Network parameters θ (green) are only updated in the outer loop and shared across tasks. Hence, they stay fixed at test time. By initialising ϕ to 0, the network parameters associated with the context parameters (blue) do not affect the output of the layer before adaptation. After the first adaptation step they modulate the rest of the network to solve the new task.

Meta-RL as… partially observed RL?

$$\pi_\theta(a|\overbrace{s, z}^{\tilde{s}})$$

encapsulates information policy needs to solve current task

learning a task = inferring z

from *context* $(s_1, a_1, s_2, r_1), (s_2, a_2, s_3, r_2), \ldots$

this is just a POMDP!

before: $\mathcal{M} = \{\mathcal{S}, \mathcal{A}, \mathcal{P}, r\}$

now: $\tilde{\mathcal{M}} = \{\tilde{\mathcal{S}}, \mathcal{A}, \tilde{\mathcal{O}}, \tilde{\mathcal{P}}, \mathcal{E}, r\}$

$\tilde{\mathcal{S}} = \mathcal{S} \times \mathcal{Z}$ $\quad$ $\tilde{s} = (s, z)$

$\tilde{\mathcal{O}} = \mathcal{S}$ $\quad$ $\tilde{o} = s$

key idea: solving the POMDP $\tilde{\mathcal{M}}$ is equivalent to meta-learning!

Meta-RL as… partially observed RL?

$\pi_\theta(a|s,z)$

encapsulates information policy needs to solve current task

learning a task = inferring z

from *context* $(s_1, a_1, s_2, r_1), (s_2, a_2, s_3, r_2), \ldots$

exploring via posterior sampling with latent context

1. sample $z \sim \hat{p}(z_t|s_{1:t}, a_{1:t}, r_{1:t})$ ← some approximate posterior (e.g., variational)
2. act according to $\pi_\theta(a|s,z)$ to collect more data ← act as though z was correct!

this is just a POMDP!

typically requires *either*:

explicit state estimation, i.e. to estimate $p(s_t|o_{1:t})$

policies with memory

need to estimate $p(z_t|s_{1:t}, a_{1:t}, r_{1:t})$

this is not optimal!
why?

but it's pretty good, both in theory and in practice!

Variational inference for meta-RL

policy: $\pi_\theta(a_t|s_t, z_t)$

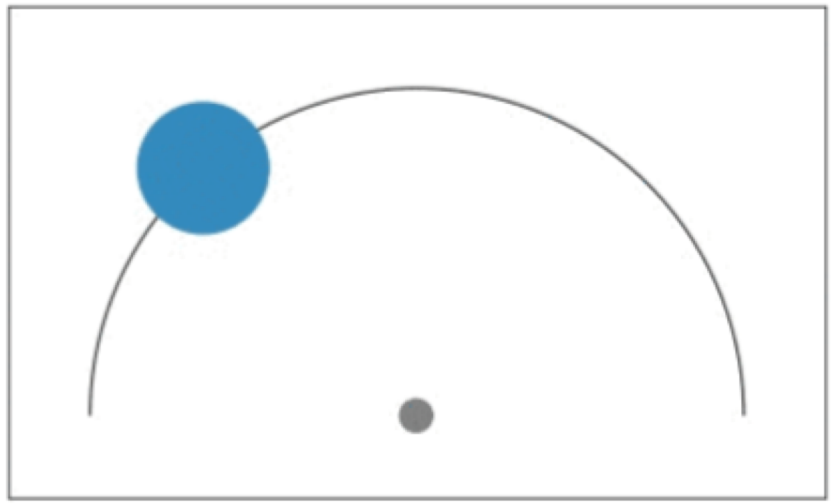
inference network: $q_\phi(z_t|s_1, a_1, r_1, \ldots, s_t, a_t, r_t)$

$$(\theta, \phi) = \arg\max_{\theta,\phi} \frac{1}{N} \sum_{i=1}^{n} E_{z\sim q_\phi, \tau\sim\pi_\theta}[R_i(\tau) - D_{\mathrm{KL}}(q(z|\ldots)\|p(z))]$$

maximize post-update reward
(same as standard meta-RL)

stay close to prior

$$z_t \sim q_\phi(z_t|s_1, a_1, r_1, \ldots, s_t, a_t, r_t)$$



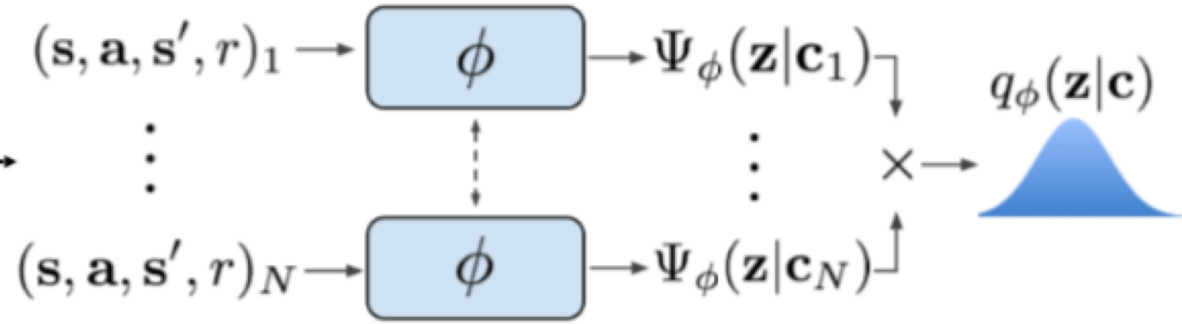
conceptually *very* similar to RNN meta-RL, but with stochastic z

stochastic z enables exploration via *posterior sampling*

Specific instantiation: PEARL

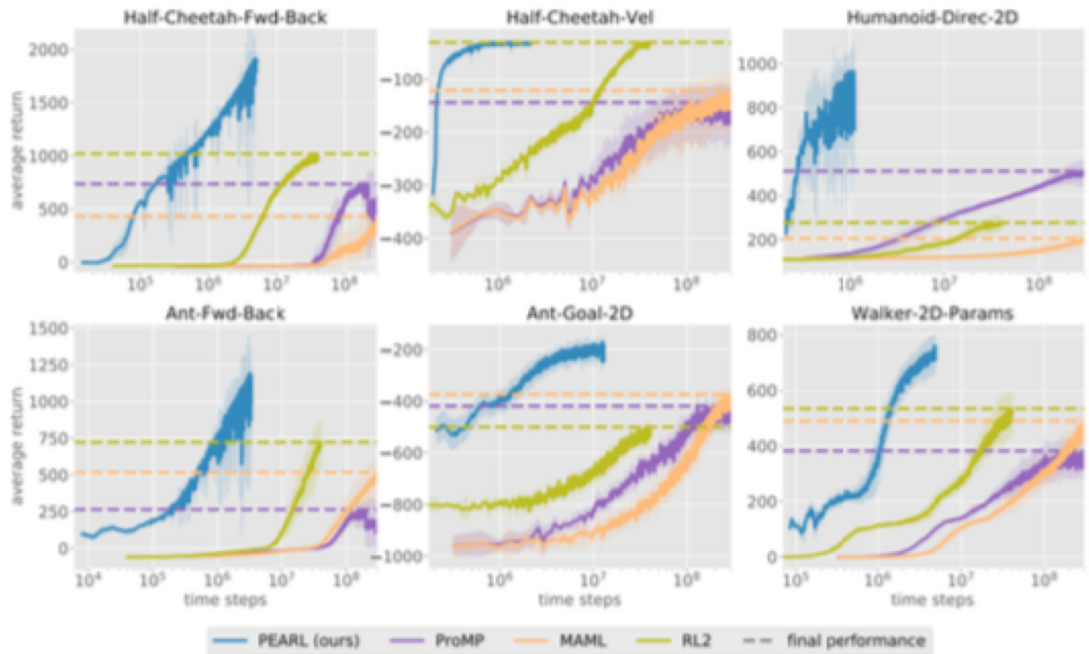
policy: $\pi_\theta(a_t|s_t, z_t)$

inference network: $q_\phi(z_t|s_1, a_1, r_1, \ldots, s_t, a_t, r_t)$



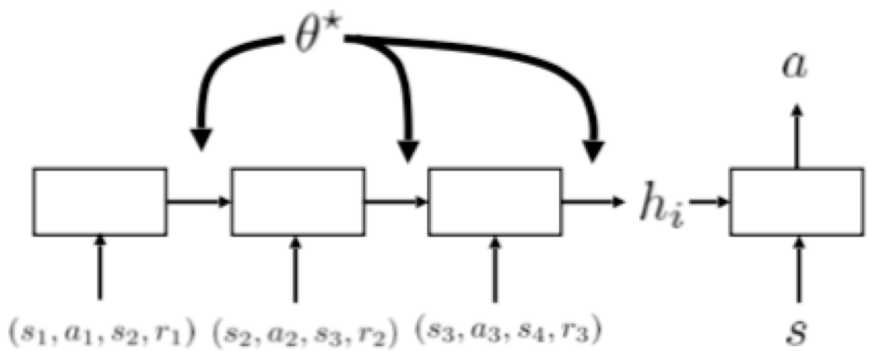
$$(\theta, \phi) = \arg\max_{\theta,\phi} \frac{1}{N} \sum_{i=1}^{n} E_{z\sim q_\phi, \tau\sim\pi_\theta}[R_i(\tau) - D_{\mathrm{KL}}(q(z|\ldots)\|p(z))]$$

perform maximization using soft actor-critic (SAC), state-of-the-art off-policy RL algorithm



The three perspectives on meta-RL

Perspective 1: just RNN it



Perspective 2: bi-level optimization

$$f_\theta(\mathcal{M}_i) = \theta + \alpha \nabla_\theta J_i(\theta)$$

MAML for RL

Perspective 3: it's an inference problem!

$\pi_\theta(a|s, z)$ $\qquad$ $z_t \sim p(z_t|s_{1:t}, a_{1:t}, r_{1:t})$

everything needed to solve task

$$\theta^\star = \arg\max_\theta \sum_{i=1}^{n} E_{\pi_{\phi_i}(\tau)}[R(\tau)]$$

where $\phi_i = f_\theta(\mathcal{M}_i)$

what should $f_\theta(\mathcal{M}_i)$ *do*?

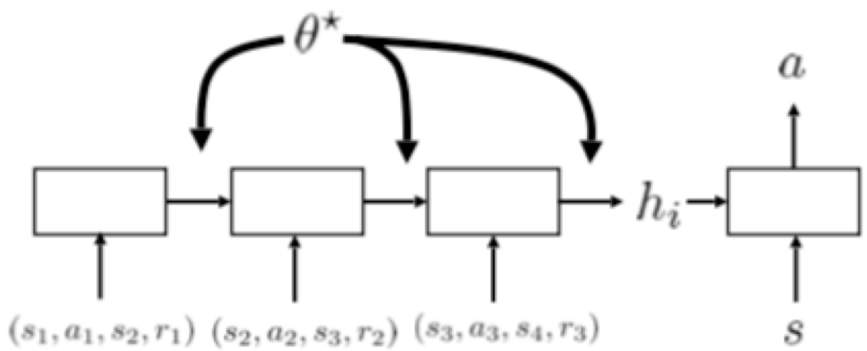
1. improve policy with experience from $\mathcal{M}_i$
   $\{(s_1, a_1, s_2, r_1), \ldots, (s_T, a_T, s_{T+1}, r_T)\}$
2. (new in RL): choose how to interact, i.e. choose a_t
   meta-RL must also *choose* how to *explore*!

But they're not that different!

just perspective 1, but with stochastic hidden variables!

i.e., $\phi = \mathbf{z}$

Perspective 1: just RNN it



Perspective 2: bi-level optimization

$$f_\theta(\mathcal{M}_i) = \theta + \alpha \nabla_\theta J_i(\theta)$$

MAML for RL

Perspective 3: it's an inference problem!

$$\pi_\theta(a|s, z) \qquad z_t \sim p(z_t|s_{1:t}, a_{1:t}, r_{1:t})$$

everything needed to solve task

just a particular architecture choice for these

Meta-learning Hyperparameters

$$G_\eta^{(n)}(\tau_t) = R_{t+1} + \gamma R_{t+2} + \cdots + \gamma^{n-1} R_{t+n} + \gamma^n v_\theta(s_{t+n})$$

$$G_\eta^\lambda(\tau_t) = (1-\lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_\eta^{(n)}$$

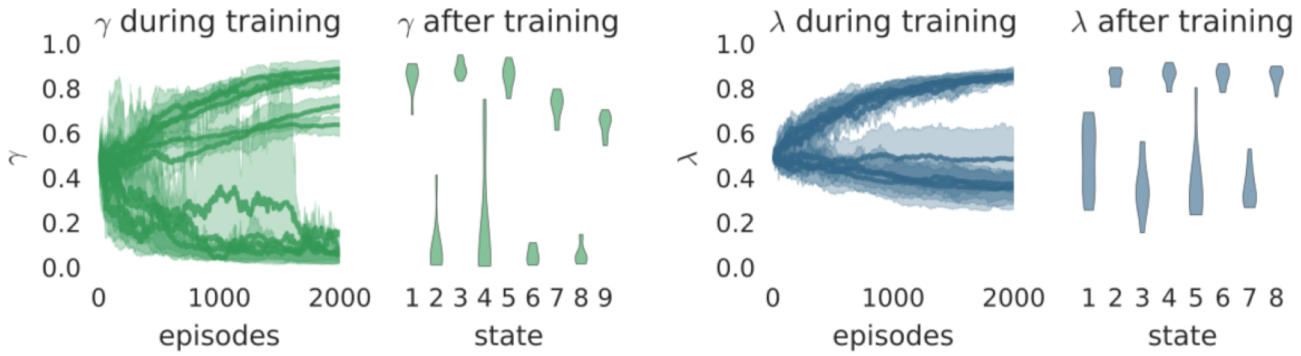
$$\begin{aligned}
\Delta\eta &= -\beta \frac{\partial J(\tau', \theta', \bar{\eta})}{\partial \eta} \\
&= -\beta \frac{\partial J(\tau', \theta', \bar{\eta})}{\partial \theta'} \frac{d\theta'}{d\eta} \\
&= -\beta \frac{\partial J(\tau', \theta', \bar{\eta})}{\partial \theta'} \frac{\partial(\theta + f(\tau, \theta, \eta))}{\partial \eta} \\
&= -\beta \frac{\partial J(\tau', \theta', \bar{\eta})}{\partial \theta'} \left(\frac{d\theta}{d\eta} + \frac{\partial f(\tau, \theta, \eta)}{\partial \theta} \frac{d\theta}{d\eta} + \frac{\partial f(\tau, \theta, \eta)}{\partial \eta} \frac{d\eta}{d\eta} \right) \\
&= -\beta \frac{\partial J(\tau', \theta', \bar{\eta})}{\partial \theta'} \left(\left(\mathbf{I} + \frac{\partial f(\tau, \theta, \eta)}{\partial \theta} \right) \frac{d\theta}{d\eta} + \frac{\partial f(\tau, \theta, \eta)}{\partial \eta} \right)
\end{aligned}$$

Meta-Gradient Reinforcement Learning

Zhongwen Xu
DeepMind
zhongwen@google.com

Hado van Hasselt
DeepMind
hado@google.com

David Silver
DeepMind
davidsilver@google.com



Meta-learning the Exploration Strategies

Meta-Reinforcement Learning of Structured Exploration Strategies

Abhishek Gupta, Russell Mendonca, YuXuan Liu, Pieter Abbeel, Sergey Levine
Department of Electrical Engineering and Computer Science
University of California, Berkeley
{abhigupta, pabbeel, svlevine}@eecs.berkeley.edu
{russellm, yuxuanliu}@berkeley.edu

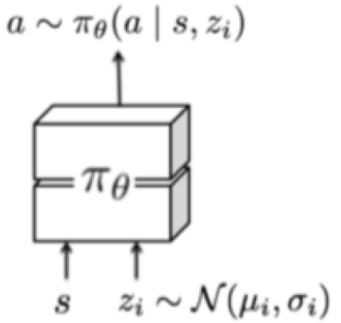
Algorithm 1 MAESN meta-RL algorithm

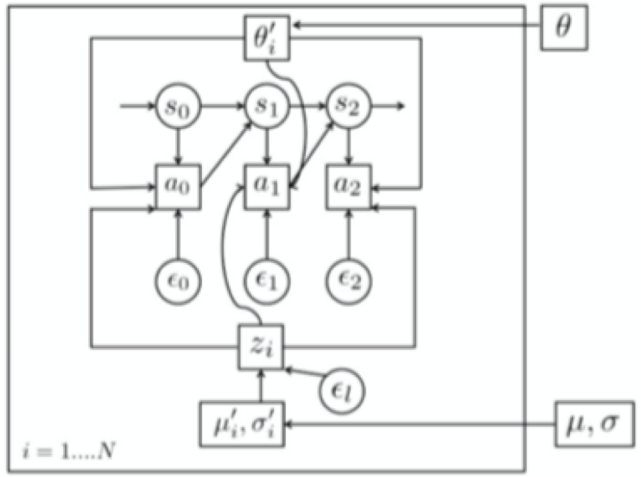
1: Initialize variational parameters ω_i for each training task τ_i
2: **for** iteration $k \in \{1, \ldots, K\}$ **do**
3: Sample a batch of N training tasks from $p(\tau)$
4: **for** task $\tau_i \in \{1, \ldots, N\}$ **do**
5: Gather data using the latent conditioned policy θ, (ω_i)
6: Compute inner policy gradient on variational parameters via Equation (4) (optionally (5))
7: **end for**
8: Compute meta update on both latents and policy parameters by optimizing (3) with TRPO
9: **end for**

$$\max_{\theta,\omega_i} \sum_{i \in tasks} E_{\substack{a_t \sim \pi(a_t|s_t;\theta_i',z_i') \\ z_i' \sim q_{\omega_i'}(.)}} \left[\sum_t R_i(s_t) \right] - \sum_{i \in tasks} D_{KL}(q_{\omega_i}(.) \| p(z)) \quad (2)$$

$$\omega_i' = \omega_i + \alpha_\omega \circ \nabla_{\omega_i} E_{\substack{a_t \sim \pi(a_t|s_t;\theta,z_i) \\ z_i \sim q_{\omega_i}(.)}} \left[\sum_t R_i(s_t) \right] \quad (3)$$

$$\theta_i' = \theta + \alpha_\theta \circ \nabla_\theta E_{\substack{a_t \sim \pi(a_t|s_t;\theta,z_i) \\ z_i \sim q_{\omega_i}(.)}} \left[\sum_t R_i(s_t) \right] \quad (4)$$





Unsupervised Meta-Learning

Unsupervised Meta-Learning for Reinforcement Learning

Abhishek Gupta
University of California, Berkeley
abhigupta@eecs.berkeley.edu

Benjamin Eysenbach
Google
eysenbach@google.com

Chelsea Finn
University of California, Berkeley
cbfinn@eecs.berkeley.edu

Sergey Levine
University of California, Berkeley
svlevine@eecs.berkeley.edu

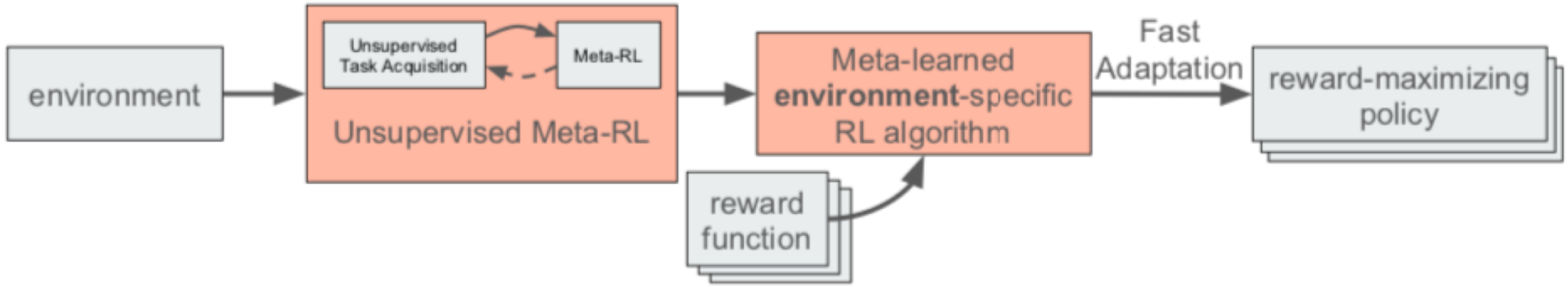
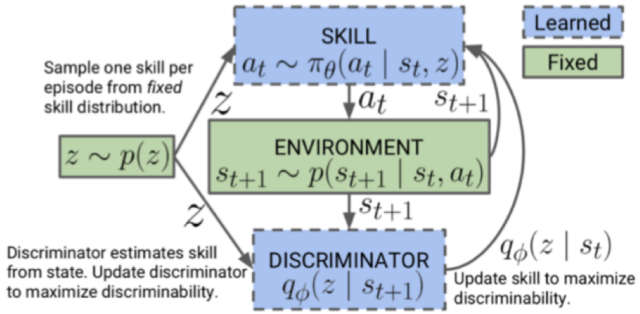


Figure 1: **Unsupervised meta-reinforcement learning**: Given an environment, unsupervised meta-reinforcement learning produces an environment-specific learning algorithm that quickly acquire new policies that maximizes any task reward function.

Unsupervised Meta-Learning

Algorithm 1: Unsupervised Meta-Reinforcement Learning Pseudocode

Data: $\mathcal{M} \setminus R$, an MDP without a reward function
Result: a learning algorithm $f : \mathcal{D} \rightarrow \pi$
Initialize $\mathcal{D} = \emptyset$
$D_\phi \leftarrow$ DIAYN() or $D_\phi \leftarrow random$
while *not converged* **do**
  Sample latent task variables $z \sim p(z)$
  Extract corresponding task reward functions $r_z(s)$ using $\mathcal{D}_\phi(z|s)$
  update f using MAML with reward $r_z(s)$



Algorithm 1: DIAYN

while *not converged* **do**
  Sample skill $z \sim p(z)$ and initial state $s_0 \sim p_0(s)$
  for $t \leftarrow 1$ **to** *steps_per_episode* **do**
    Sample action $a_t \sim \pi_\theta(a_t \mid s_t, z)$ from skill.
    Step environment: $s_{t+1} \sim p(s_{t+1} \mid s_t, a_t)$.
    Compute $q_\phi(z \mid s_{t+1})$ with discriminator.
    Set skill reward $r_t = \log q_\phi(z \mid s_{t+1}) - \log p(z)$
    Update policy (θ) to maximize r_t with SAC.
    Update discriminator (ϕ) with SGD.