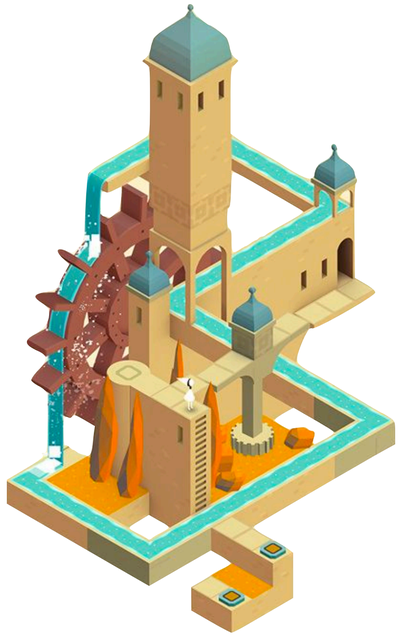


Meta-Learning

ACDL 2021, part 1

Joaquin Vanschoren (TU Eindhoven)

Cover art: MC Escher, Ustwo Games

Intro: humans can easily learn from a single example

Intro: humans can easily learn from a single example

Intro: humans can easily learn from a single example

Canna Indica ‘Picasso’

Intro: humans can easily learn from a single example

Canna Indica ‘Picasso’

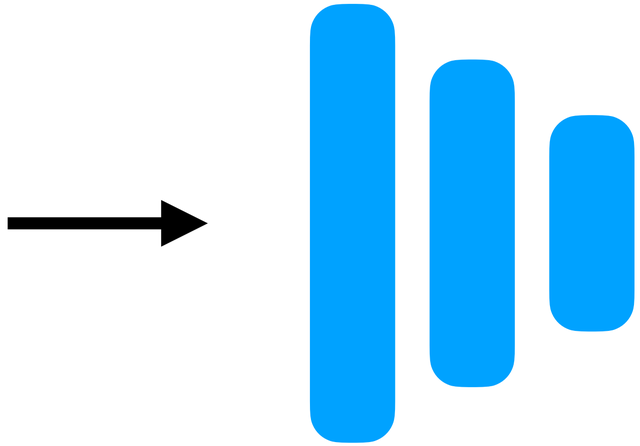
?

later

train

test

Can a computer learn from a single example?

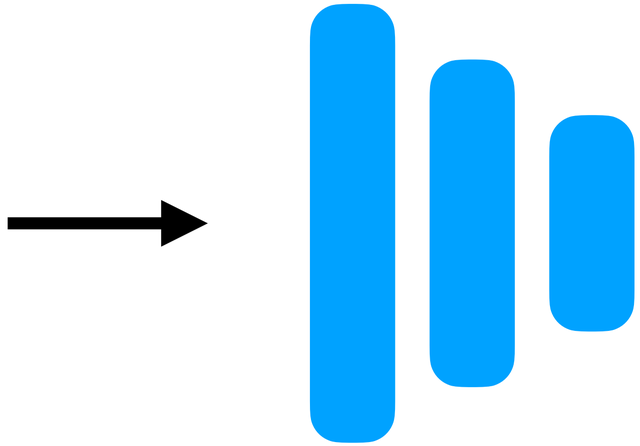


e.g. ResNet50
(23 million trainable parameters)

Canna
Indica
`Picasso'

That won't work :) Humans also don't start from scratch.

Can a computer learn from a single example?



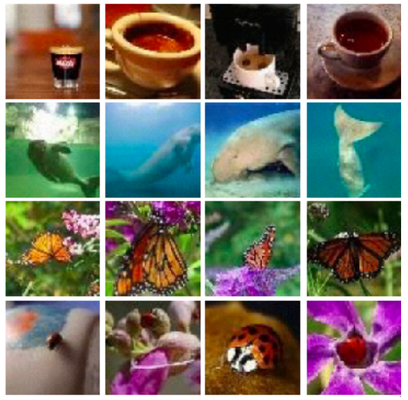
e.g. ResNet50
(23 million trainable parameters)

Canna
Indica
`Picasso'

That won't work :) Humans also don't start from scratch.

Transfer learning?

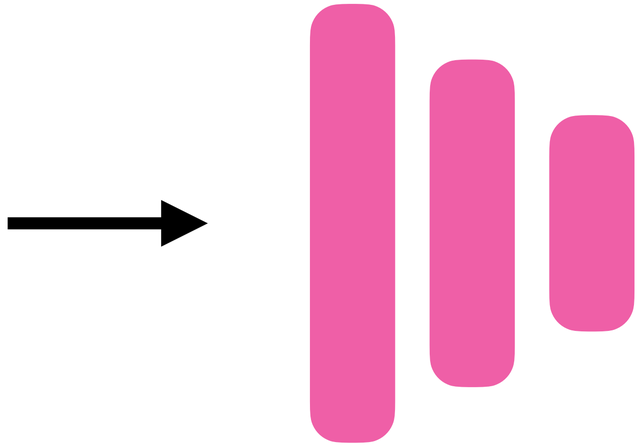
Source task



ImageNet
(14 million images)

Pretrain

Target task

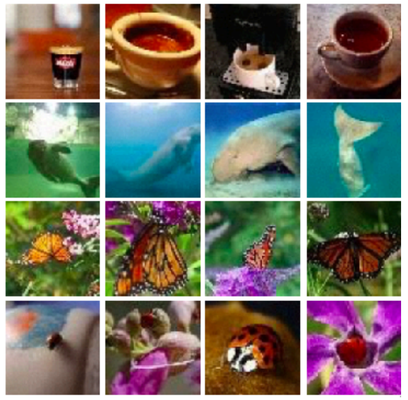


Canna Indica `Picasso'

e.g. ResNet50
(23 million trainable parameters)

Transfer learning?

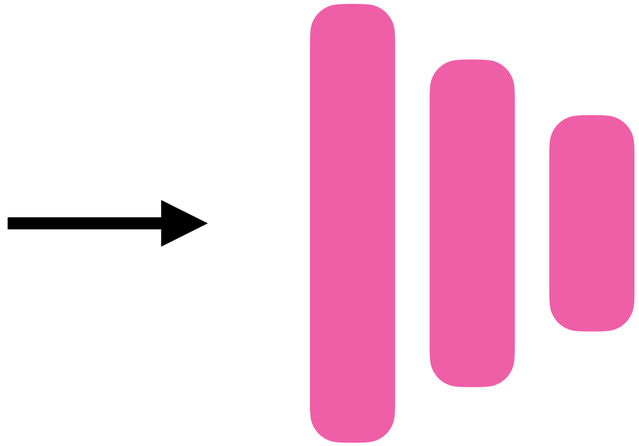
Source task



ImageNet
(14 million images)

Pretrain

Target task

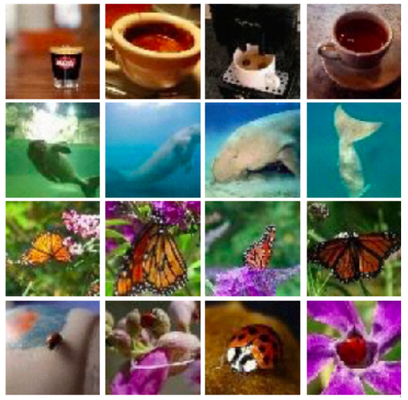


Canna Indica `Picasso'

e.g. ResNet50
(23 million trainable parameters)

Transfer learning?

Source task



ImageNet
(14 million images)

Pretrain

Target task

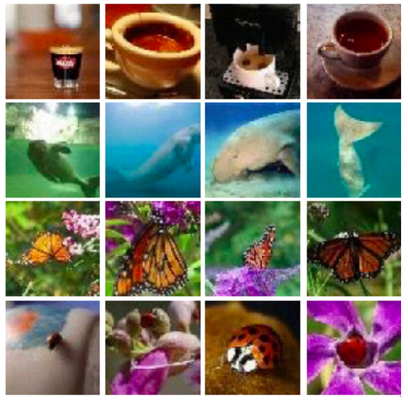
Finetune

Canna
Indica
`Picasso'

e.g. ResNet50
(23 million trainable parameters)

Transfer learning?

Source task



ImageNet
(14 million images)

Pretrain

Target task

Finetune

e.g. ResNet50
(23 million trainable parameters)

Canna Indica `Picasso'

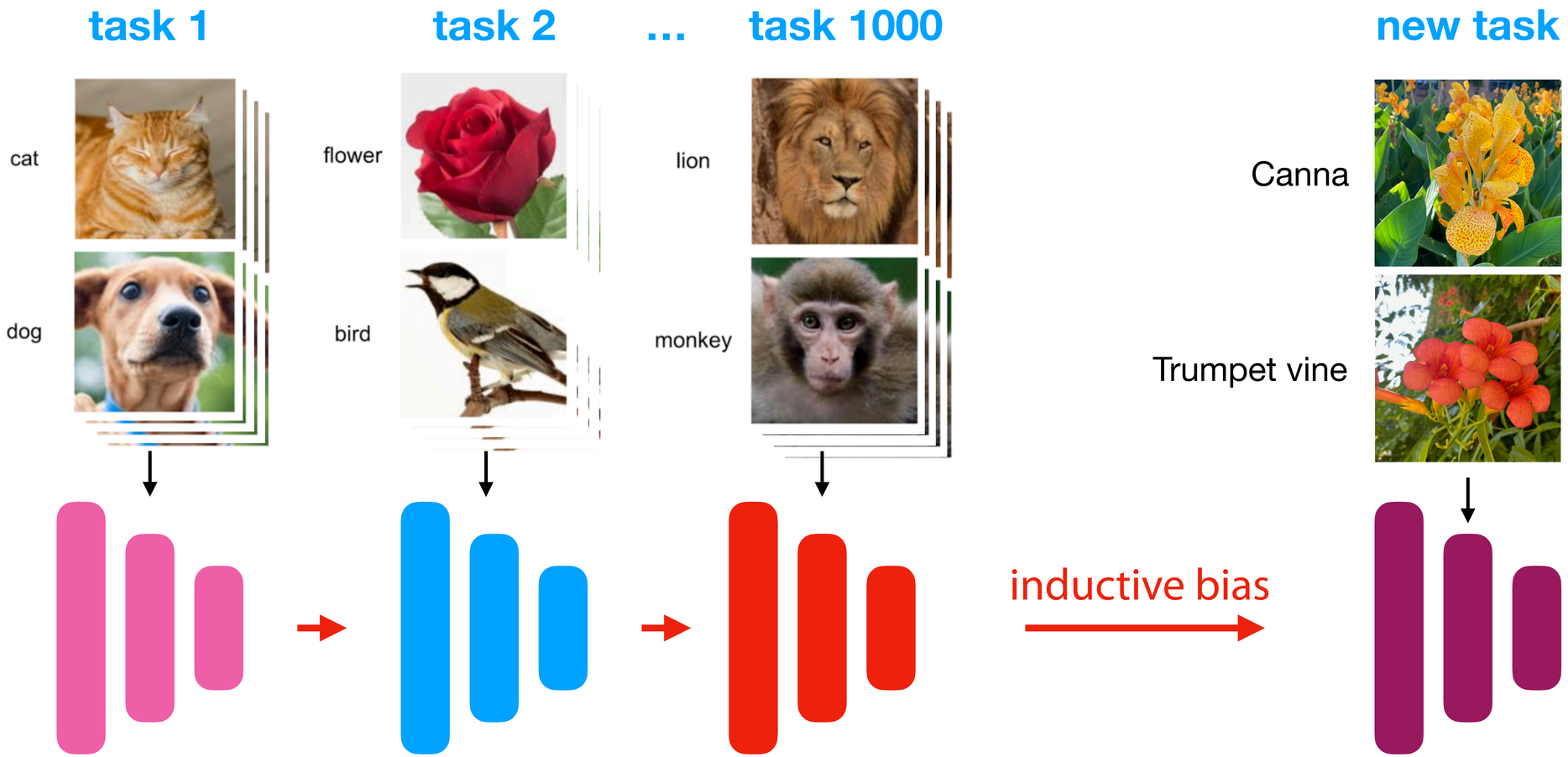
A single source task (e.g. ImageNet) may not generalize well to the test task

Meta-learning

Learn over a series (or distribution) of many different tasks/episodes

Learn what works well, and learn how to transfer that to new tasks

Prepare yourself to learn new things faster

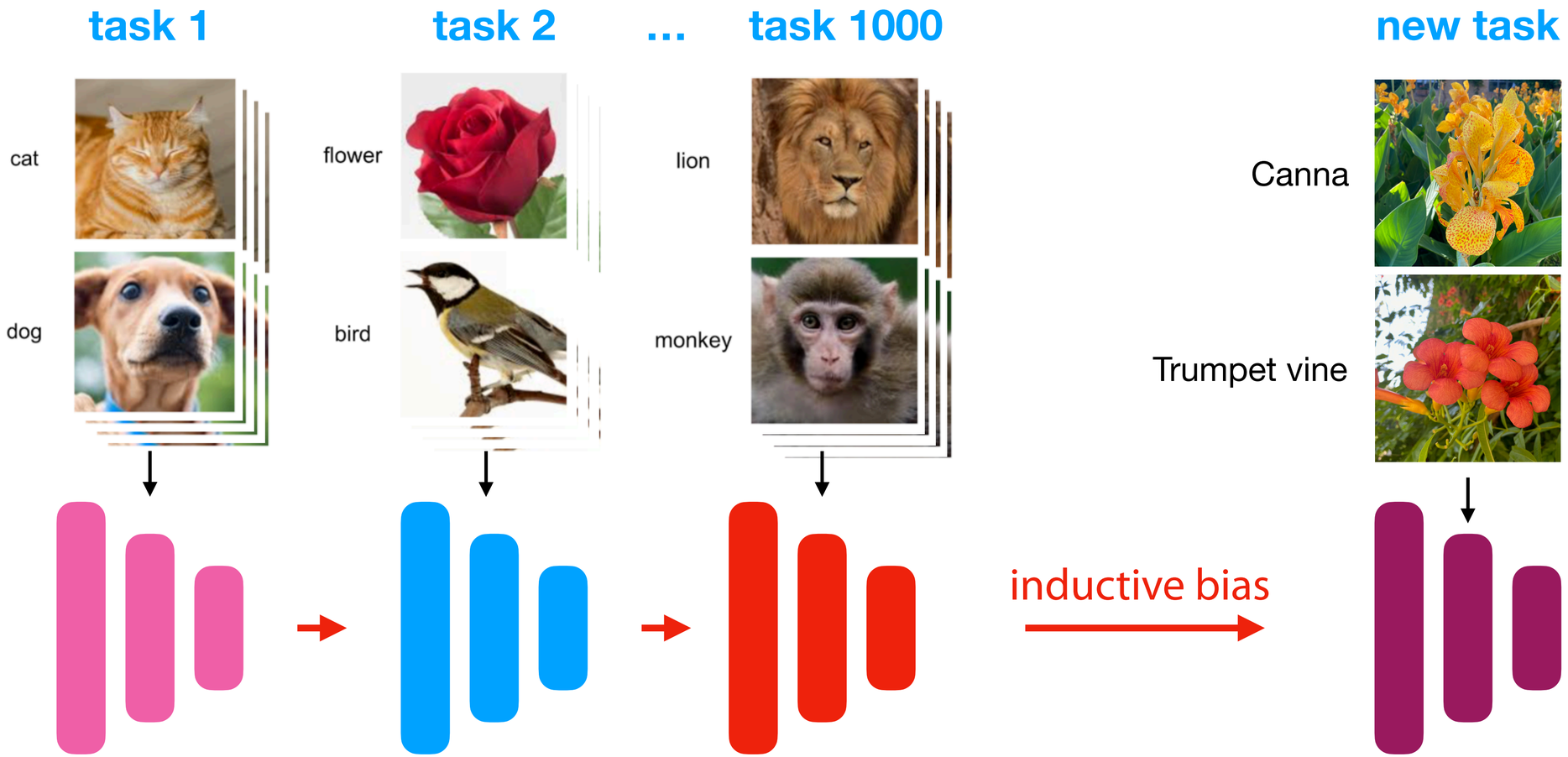


Meta-learning

Learn over a series (or distribution) of many different tasks/episodes

Learn what works well, and learn how to transfer that to new tasks

Prepare yourself to learn new things faster



Useful in many real-life situations: rare events, test-time constraints, data collection costs, privacy issues,…

Inspired by human learning

We don’t transfer from a single source task, we learn across many, many tasks

We have a ‘drive’ to explore new, challenging, but doable, fun tasks

year 1

year 2

year 3

year 4

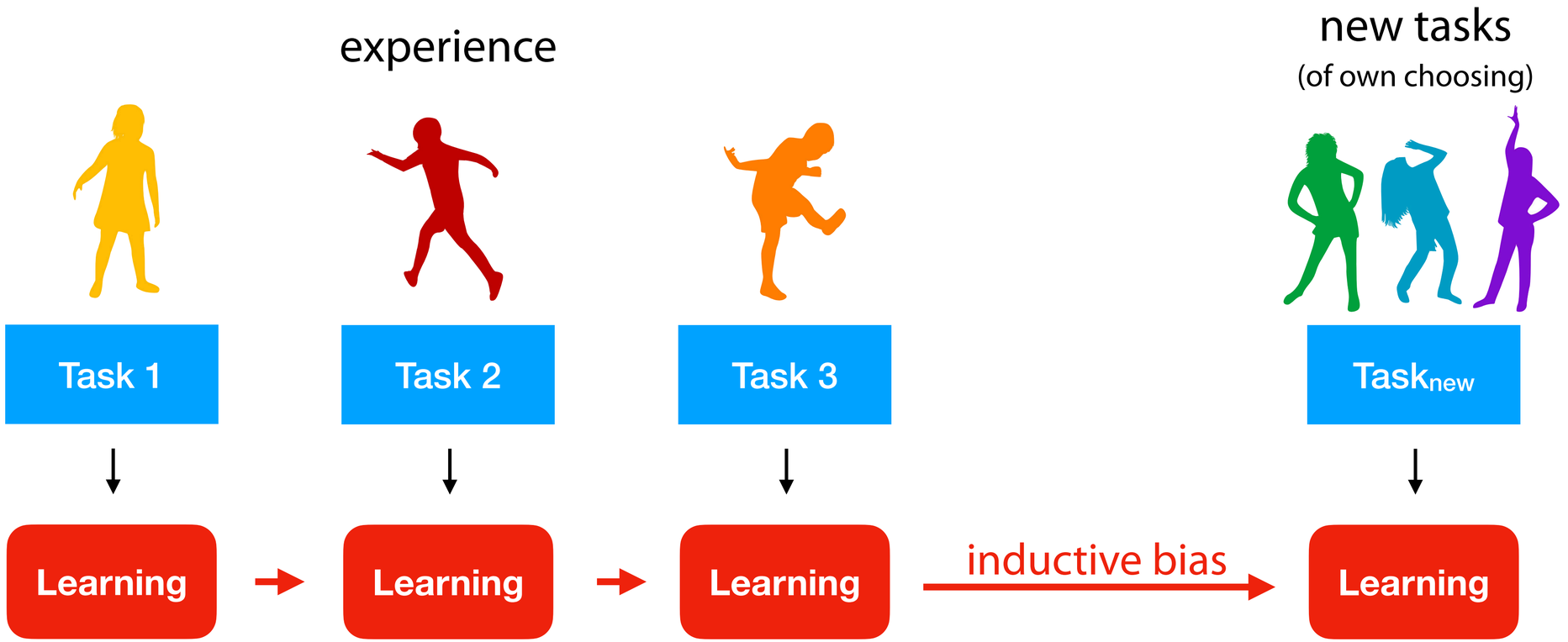
Lake et al. 2017

Kemp et al. 2007

Human-like Learning***

humans learn *across* tasks: less trial-and-error, less data, less compute

new tasks should be related to experience (doable, fun, interesting?)



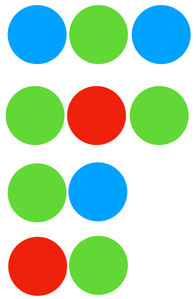
key aspects of fast learning: compositionality, causality, learning to learn

Lake et al. 2017

learning compositionality (e.g. language)

easy enough for humans, hard for computers

Training



lug blicket wif

wif blicket dax

lug kiki wif

wif kiki dax

Test

dax blicket zup?

wif blicket dax kiki lug?

Lake et al. 2017

learning compositionality (e.g. language)

easy enough for humans, hard for computers

Training

dax
zup
lug
wif

lug blicket wif

wif blicket dax

lug kiki wif

wif kiki dax

Test

dax blicket zup?

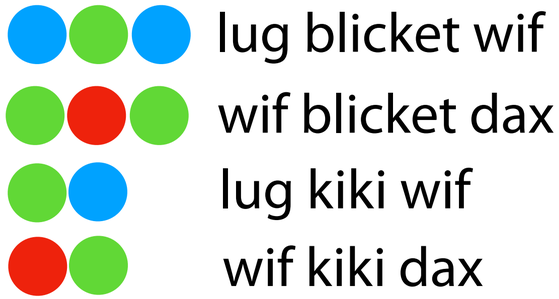
wif blicket dax kiki lug?

Lake et al. 2017

learning compositionality (e.g. language)

easy enough for humans, hard for computers

Training



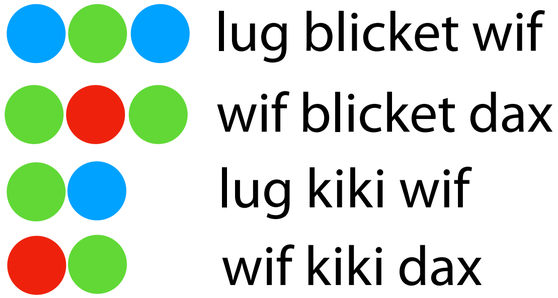
Test

Lake et al. 2017

learning compositionality (e.g. language)

easy enough for humans, hard for computers

Training



Test

Common mistakes

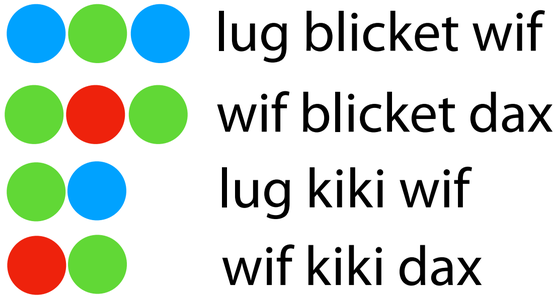
one-to-one bias:
assume that every word is one color

Lake et al. 2017

learning compositionality (e.g. language)

easy enough for humans, hard for computers

Training



Test

Common mistakes

one-to-one bias:
assume that every word is one color

concatenation bias:
assume that order is always left-to-right

What if there is *no* training data?

humans can still solve problems by making assumptions

[Lake et al. 2017]
[Lake et al. 2019]

Item pool 

Test

zup?

zup zup?

dax zup?

zup tufa?

zup wif zup?

zup wif blicket?

blicket wif zup?

What if there is *no* training data?

humans can still solve problems by making assumptions

[Lake et al. 2017]
[Lake et al. 2019]

Item pool 

Test 

zup?

zup zup?

dax zup?

zup tufa?

zup wif zup?

zup wif blicket?

blicket wif zup?

What if there is *no* training data?

humans can still solve problems by making assumptions

[Lake et al. 2017]
[Lake et al. 2019]

Item pool 

Test


zup?


zup zup?

dax zup?

zup tufa?

zup wif zup?

zup wif blicket?

blicket wif zup?

What if there is *no* training data?

humans can still solve problems by making assumptions

[Lake et al. 2017]
[Lake et al. 2019]

Item pool 

Test

 zup?

 zup zup?

 dax zup?

zup tufa?

zup wif zup?

zup wif blicket?

blicket wif zup?

What if there is *no* training data?

humans can still solve problems by making assumptions

[Lake et al. 2017]
[Lake et al. 2019]

Item pool 

Test

 zup?

 zup zup?

 dax zup?

 zup tufa?

zup wif zup?

zup wif blicket?

blicket wif zup?

What if there is *no* training data?

humans can still solve problems by making assumptions

[Lake et al. 2017]
[Lake et al. 2019]

Item pool 

Test

 zup?

 zup zup?

 dax zup?

 zup tufa?

 zup wif zup?

zup wif blicket?

blicket wif zup?

What if there is *no* training data?

humans can still solve problems by making assumptions

[Lake et al. 2017]
[Lake et al. 2019]

Item pool 

Test

 zup?

 zup zup?

 dax zup?

 zup tufa?

 zup wif zup?

 zup wif blicket?

blicket wif zup?

What if there is *no* training data?

humans can still solve problems by making assumptions

[Lake et al. 2017]
[Lake et al. 2019]

Item pool 

Test

 zup?

 zup zup?

 dax zup?

 zup tufa?

 zup wif zup?

 zup wif blicket?

 blicket wif zup?

What if there is *no* training data?

humans can still solve problems by making assumptions

[Lake et al. 2017]
[Lake et al. 2019]

Item pool 

Test

 zup?

 zup zup?

 dax zup?

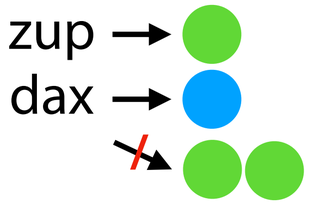
 zup tufa?

 zup wif zup?

 zup wif blicket?

 blicket wif zup?

Commonly used assumptions:



one-to-one bias:
assume that every word is one color
(and not a function or something else)

What if there is *no* training data?

humans can still solve problems by making assumptions

Lake et al. 2017
Lake et al. 2019

Item pool 

Test

 zup?

 zup zup?

 dax zup?

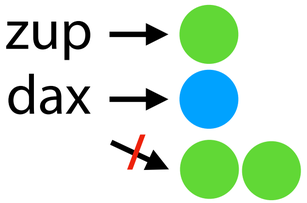
 zup tufa?

 zup wif zup?

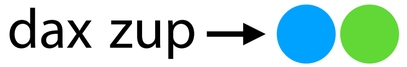
 zup wif blicket?

 blicket wif zup?

Commonly used assumptions:



one-to-one bias:
assume that every word is one color
(and not a function or something else)



concatenation bias:
assume that order is always left-to-right

What if there is *no* training data?

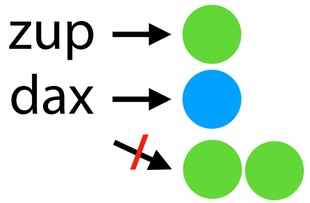
humans can still solve problems by making assumptions

[Lake et al. 2017]
[Lake et al. 2019]

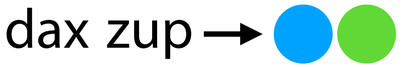
Item pool 

Test

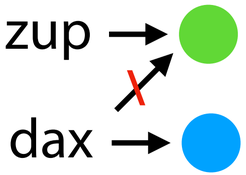
Commonly used assumptions:



one-to-one bias:
assume that every word is one color
(and not a function or something else)



concatenation bias:
assume that order is always left-to-right



mutual exclusivity:
if object has a name, it doesn't need another name

What if there is *no* training data?

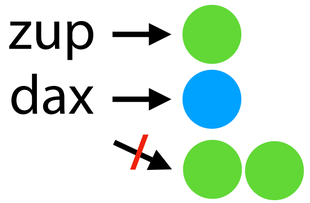
humans can still solve problems by making assumptions

Lake et al. 2017
Lake et al. 2019

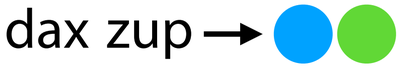
Item pool

Test

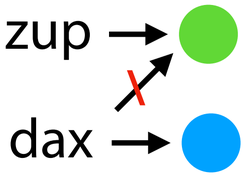
Commonly used assumptions:



one-to-one bias:
assume that every word is one color
(and not a function or something else)



concatenation bias:
assume that order is always left-to-right



mutual exclusivity:
if object has a name, it doesn't need another name

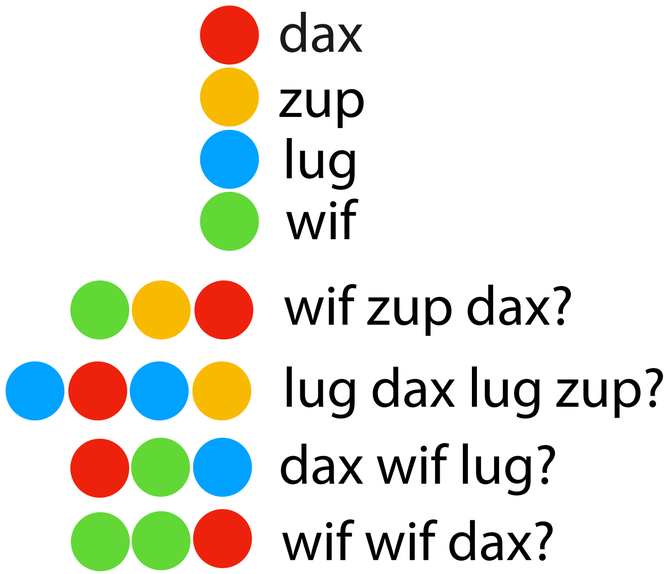
Humans *assume* that words have consistent meanings and follow input/output constraints

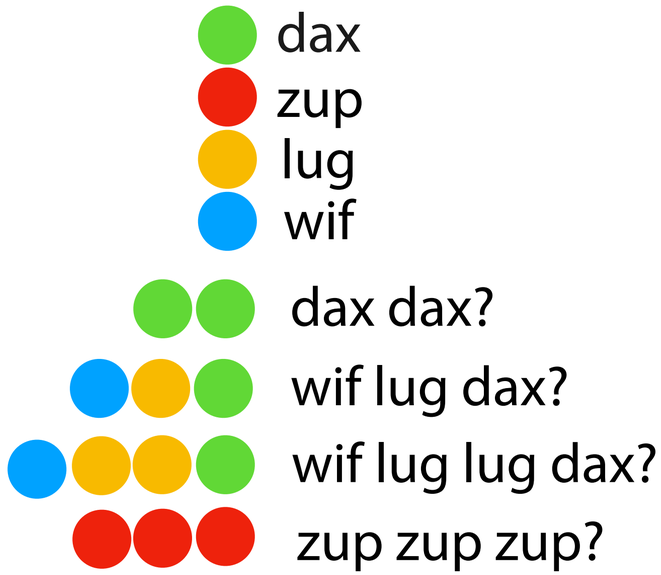
These assumptions (inductive biases) are necessary for learning

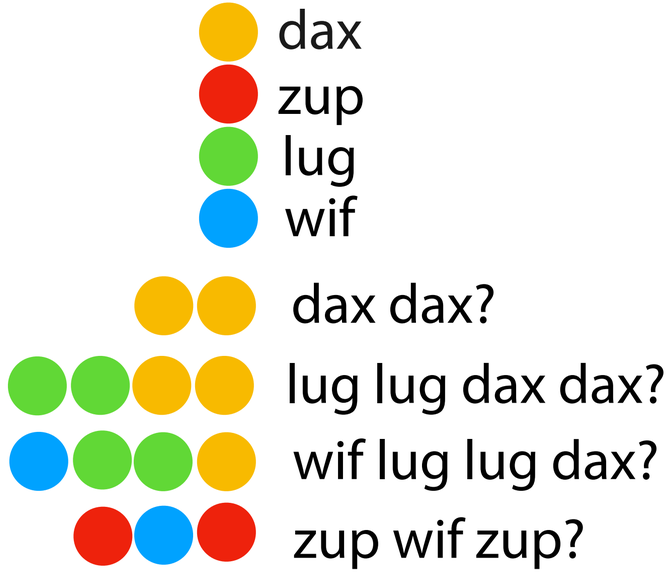
Meta-learning inductive biases

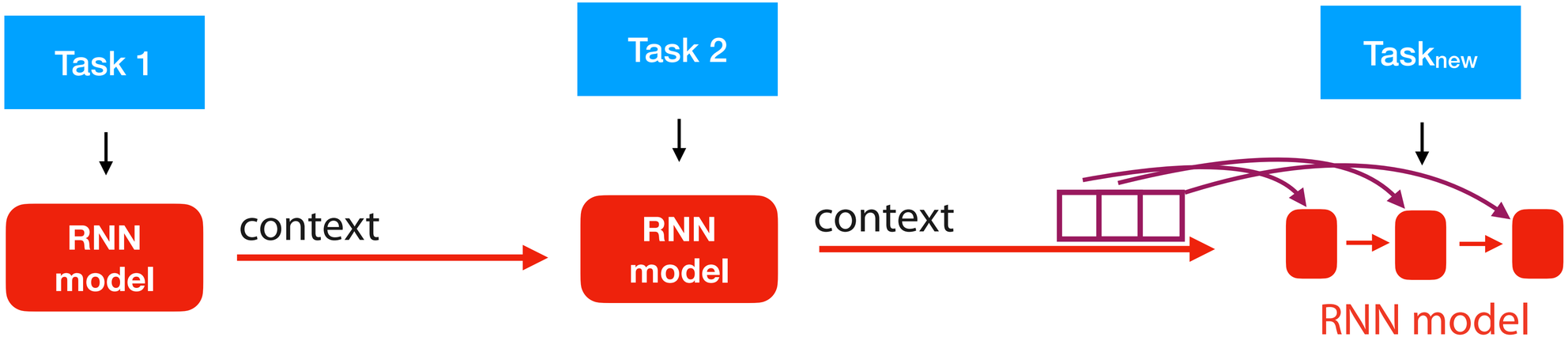
[Lake et al. 2017]
[Lake et al. 2019]

Capture useful assumptions from the data - that can often not be easily expressed



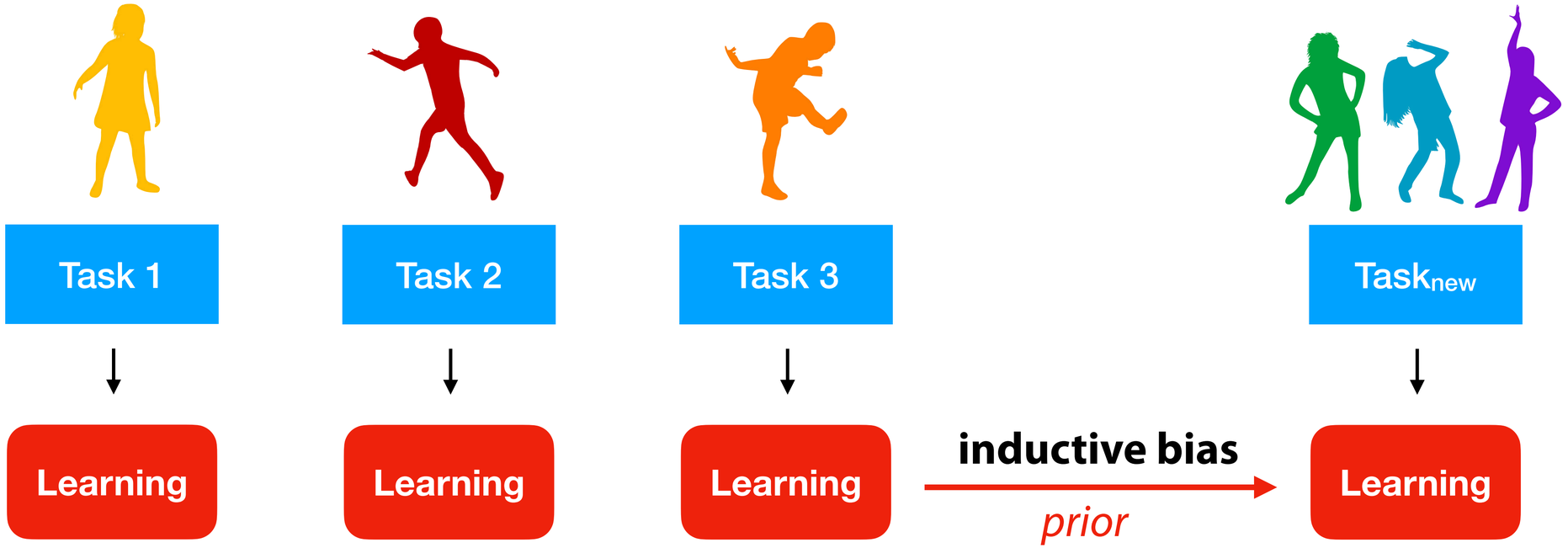






Meta-learning goal

learn *minimal* inductive biases from prior tasks instead of constructing *manual* ones
should still generalize well (otherwise you meta-overfit)

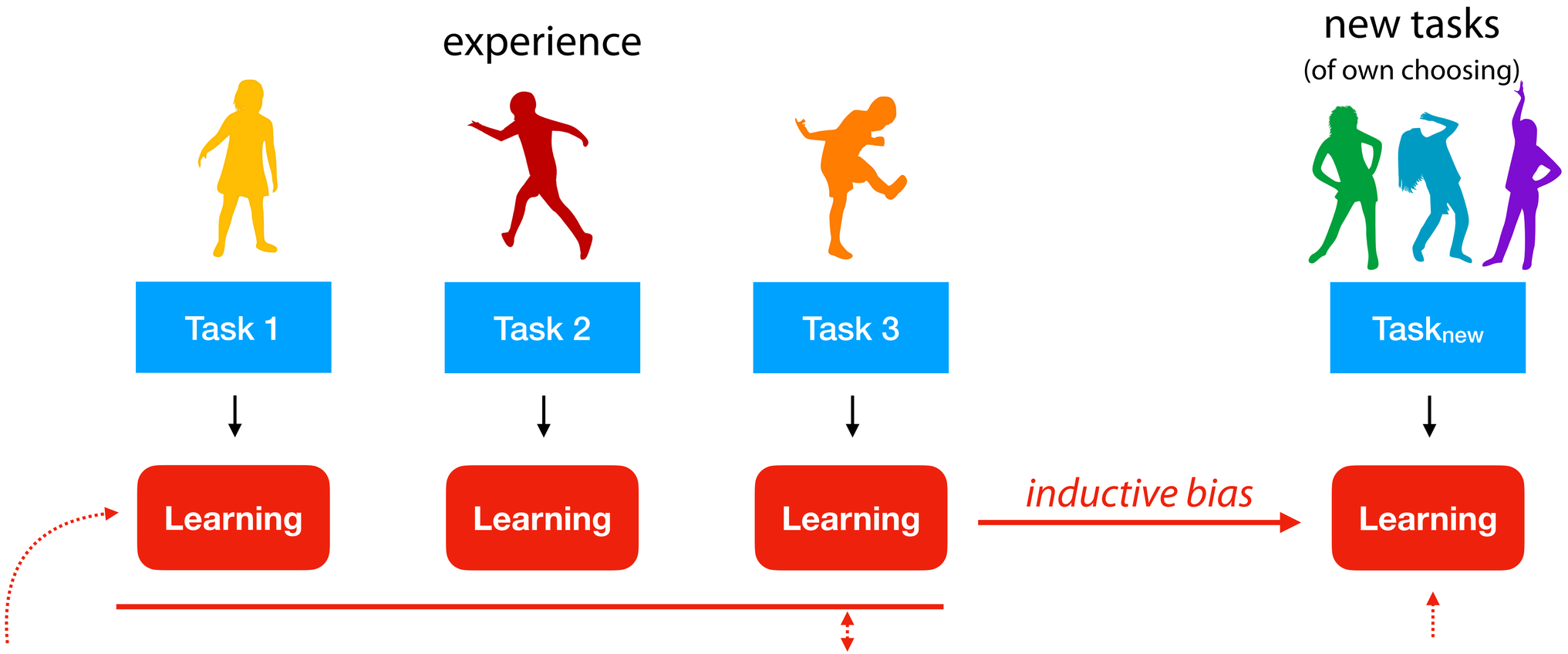


Inductive bias: any *assumptions* added to training data to learn more effectively. E.g:

- Instead of starting from random weights, learn good initial weights
- Instead of normal backpropagation, learn how to better update the weights
- Instead of standard loss/reward function, learn a better loss/reward function
- Instead of off-the-shelf model architectures, learn better architectures (and hyperparameter)

Clune 2019

What can we learn to learn?



1. architectures / pipelines (hyperparameters, structures)

2. learning algorithms (priors, weights, task embeddings,…)

3. learning environments (curricula, self-exploration)

part 2 (next)

part 1 (now)

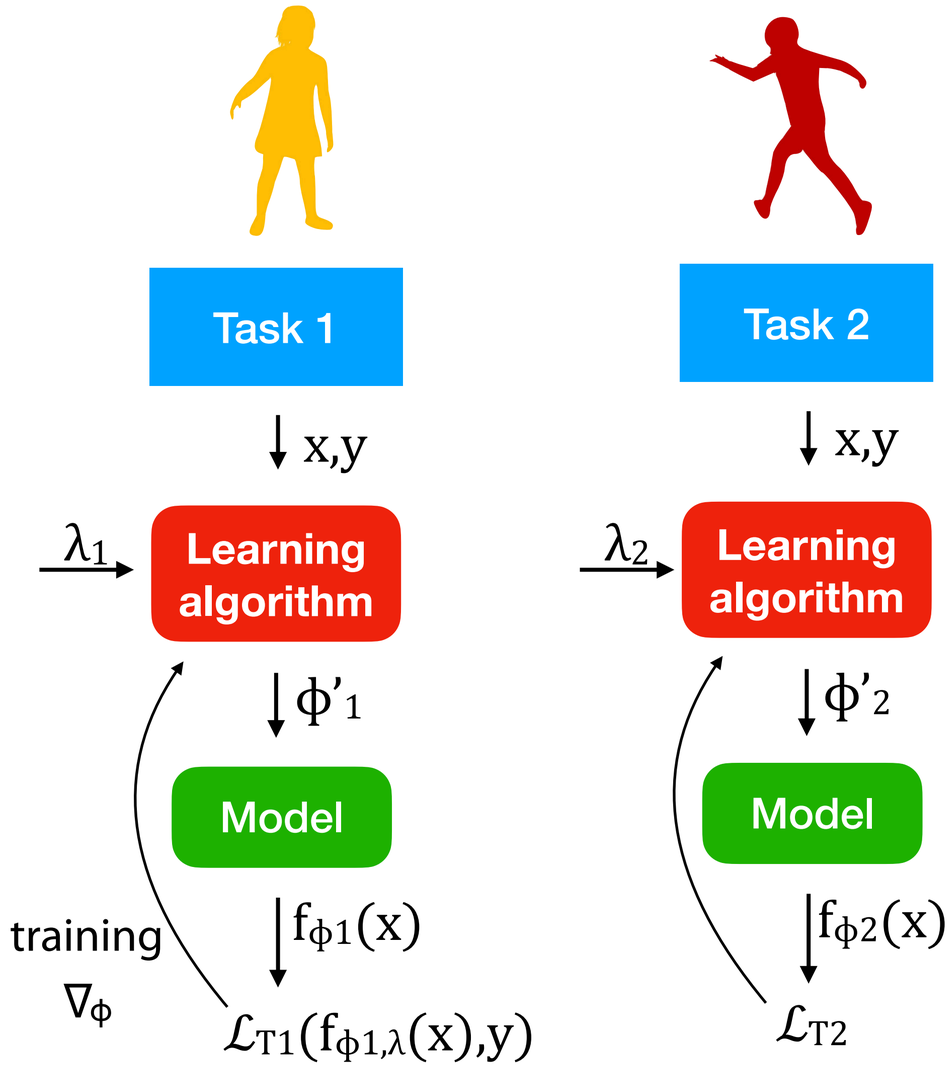
Terminology

Task: *distribution of samples* $q(x)$
outputs y, *loss* $\mathcal{L}(x,y)$

Learner: *model parameters* ϕ,
hyper-parameters λ
optimizer

$$\phi^* = \text{argmax}_\phi \log p(\phi \mid T)$$
$$= \text{argmin}_\phi \mathcal{L}(f_{\phi,\lambda}(x),y)$$

Model: $f_\phi(x) = y'$

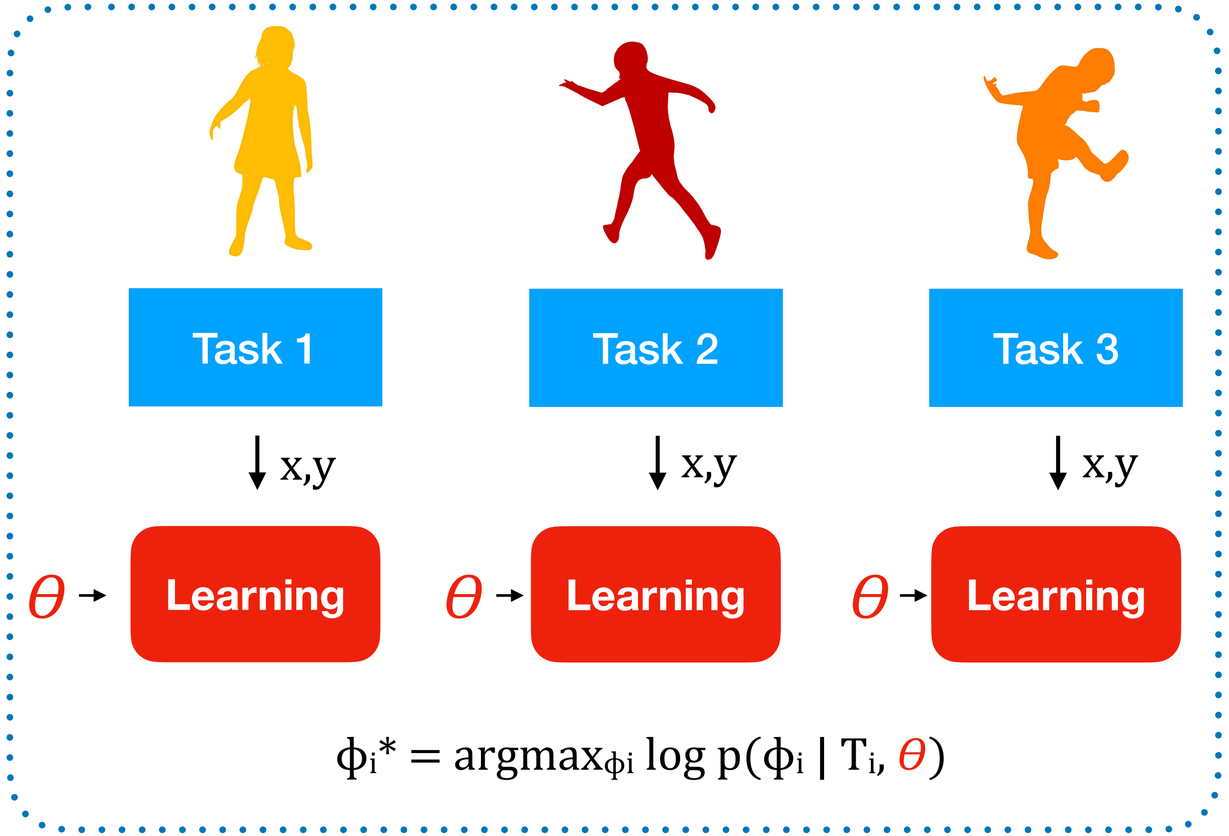


for reinforcement learning:
$q(x_1)$ + transition $q(x_{t+1} \mid x_t, y_t)$
$\mathcal{L}$ = (neg) reward

Strategy 1: bilevel optimization

parameterize some aspect of the learner that we want to learn as meta-parameters Θ

meta-learn Θ across tasks



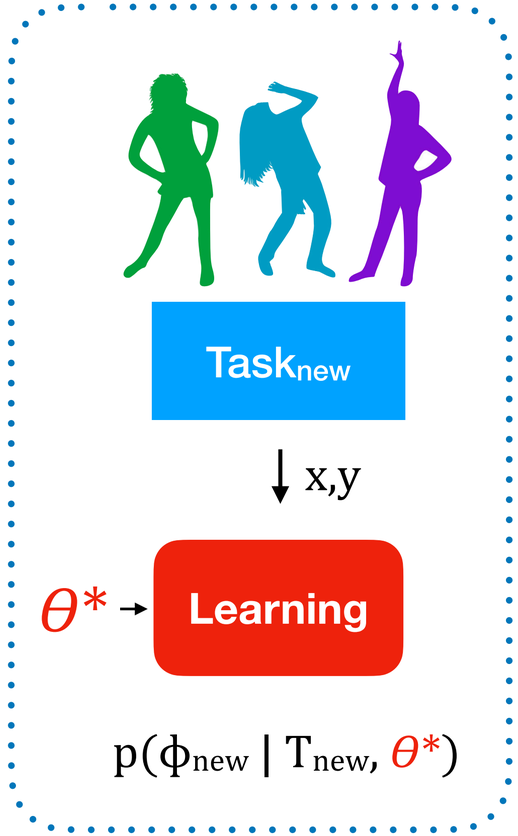
Bilevel optimization:

$$\theta^* = \text{argmax}_\theta \log p(\theta \mid T_i)$$

so that

$$\phi_i^* = \text{argmax}_\phi \log p(\phi \mid T_i, \theta^*)$$

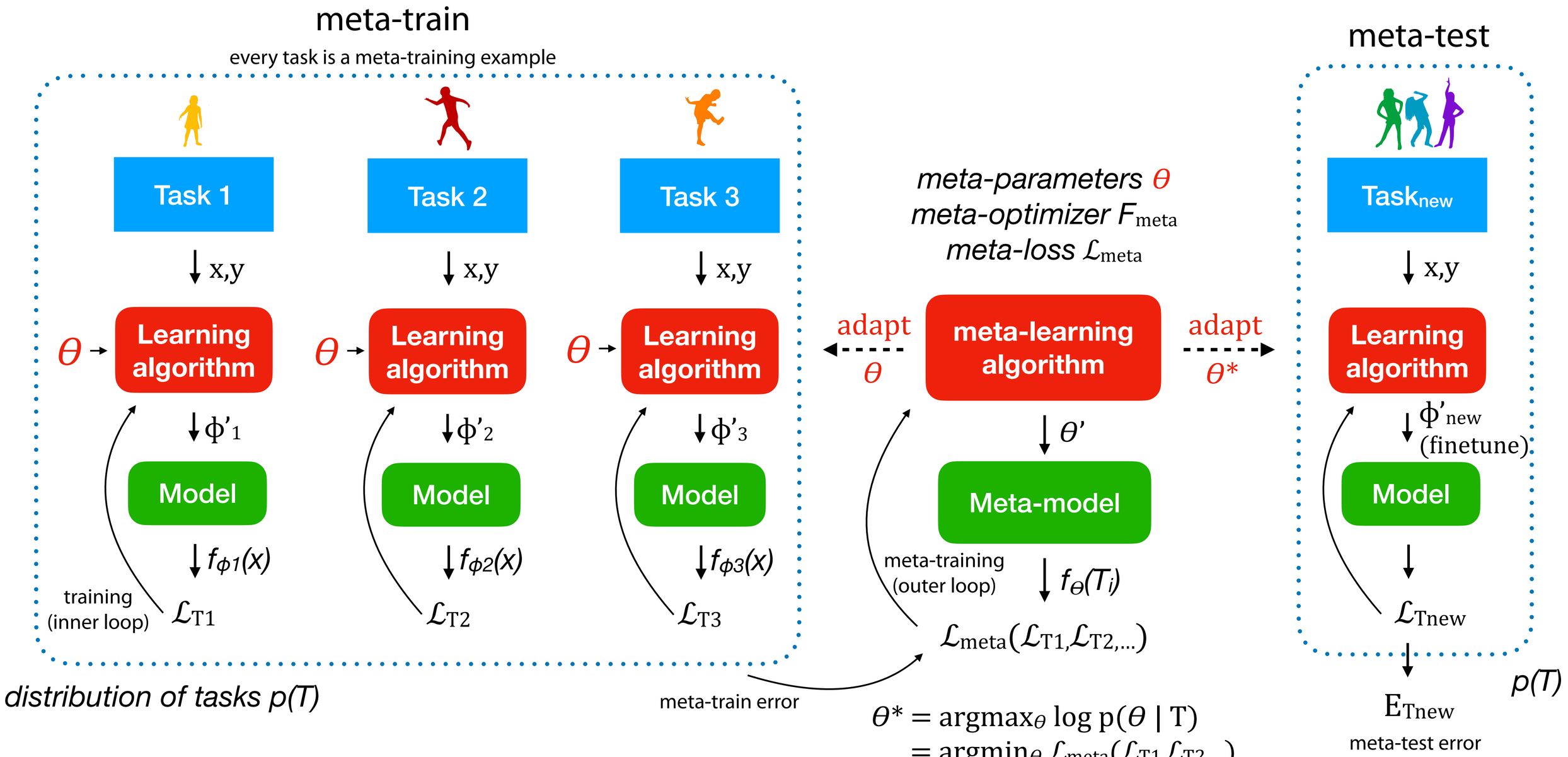
allows fine-tuning on T_{new}



Θ (prior), could encode an initialization ϕ, the hyperparameters λ, the optimizer,…

Learned Θ^* should learn T_{new} from small amount of data, yet generalize to a large number of tasks

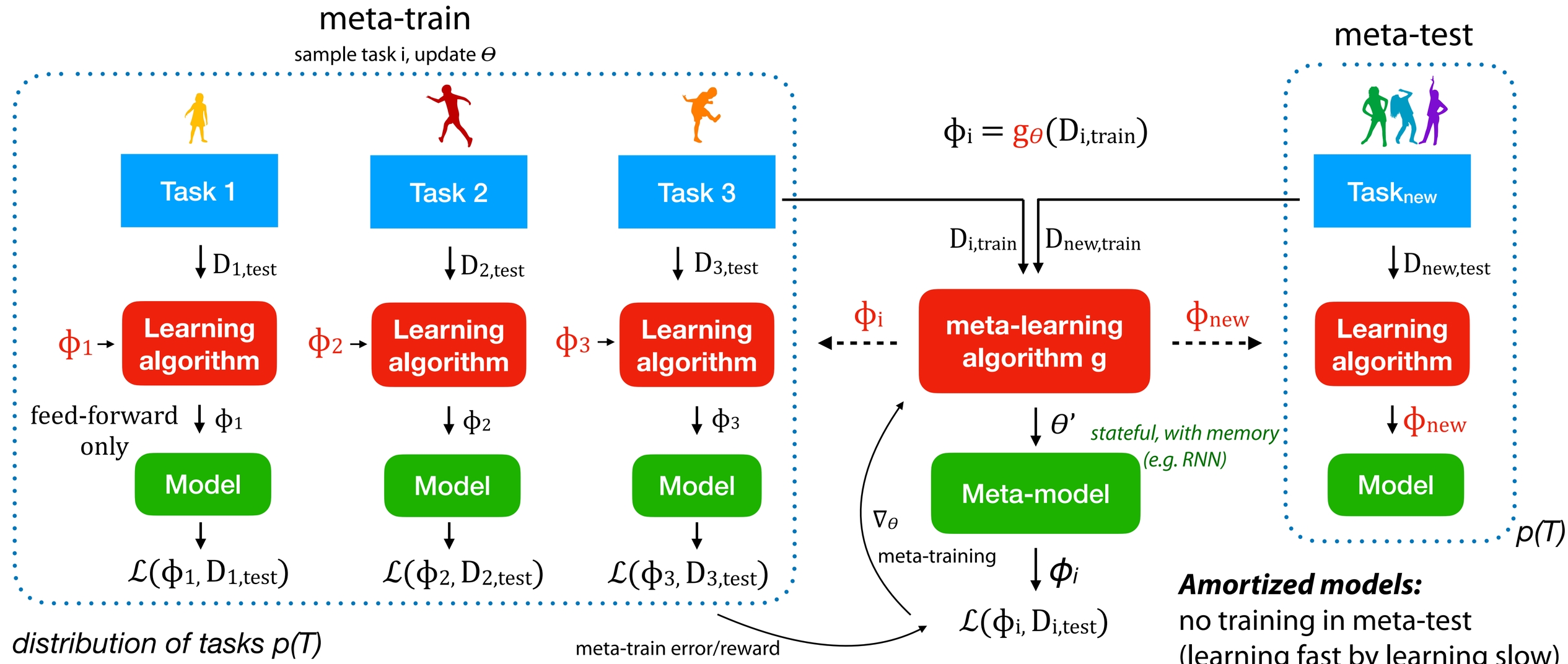
Meta-learning with bilevel optimization



Strategy 2: black-box models

black box meta-model g_θ*: predicts* ϕ *given* D_{train} (θ is hidden)

hypernetwork where input embedding learned across tasks



Amortized models:
no training in meta-test
(learning fast by learning slow)

Example: few-shot classification

[Vinyals et al. 2016](#)
[Triantafillou et al. 2019](#)

2-way, 3-shot

meta-train | **meta-test**

train task 1

D_{train} (support)

D_{test} (query)

train task n

D_{train}

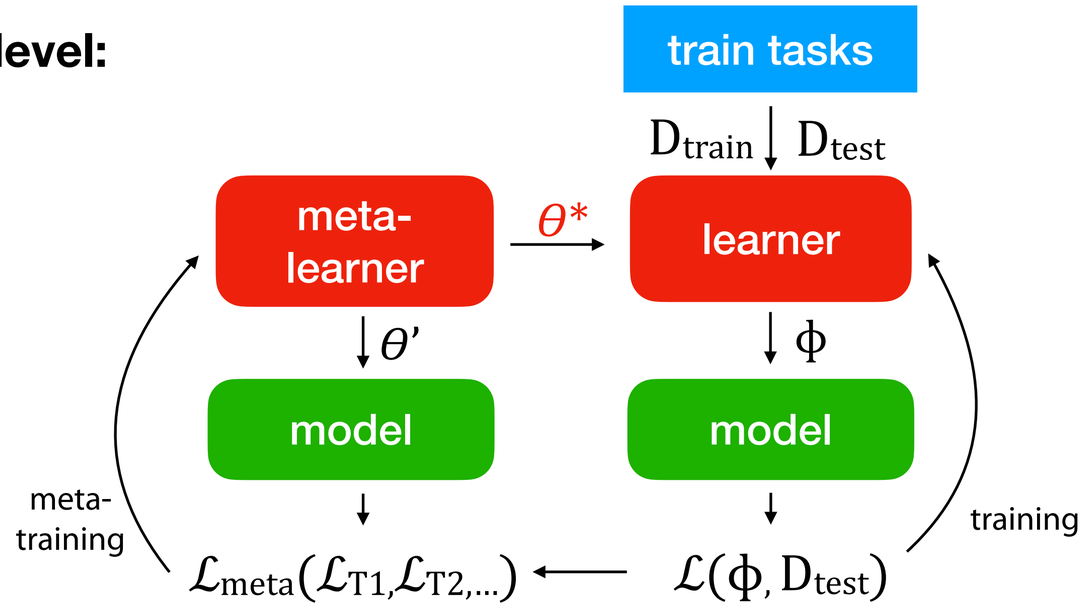
D_{test}

test task

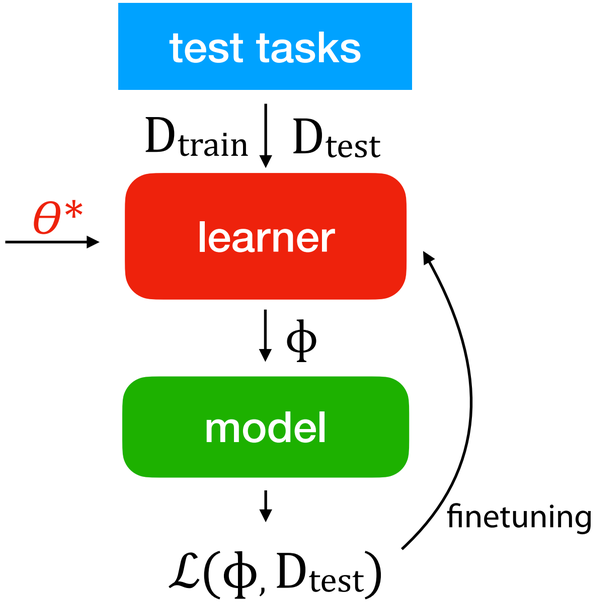
D_{train}

D_{test}

Bilevel:

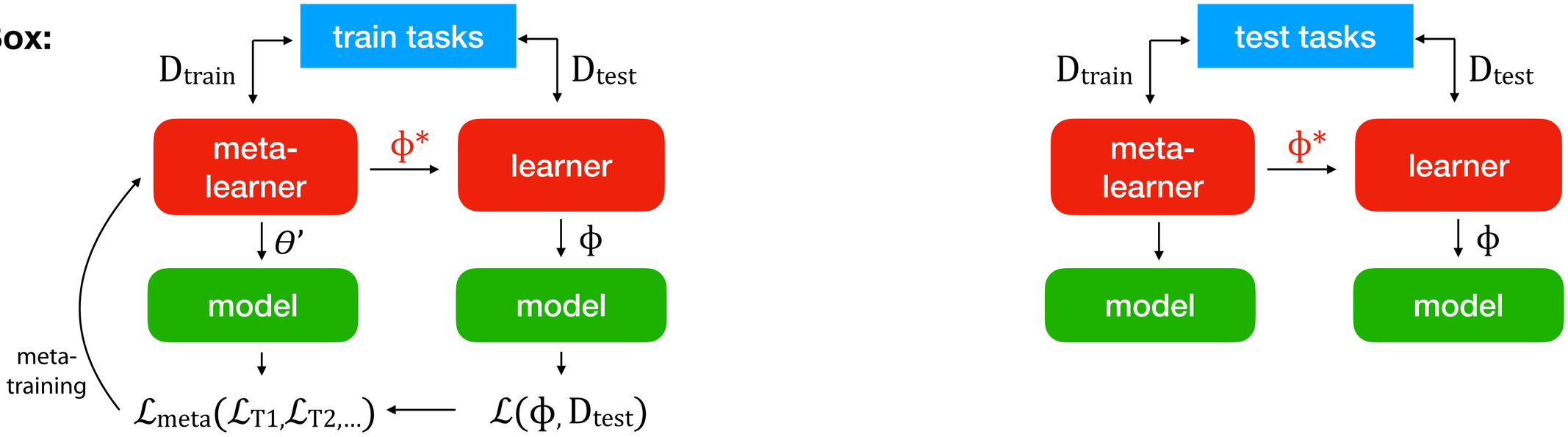


θ^*: initial ϕ
optimizer
metric
...



Example: few-shot classification

Black Box:

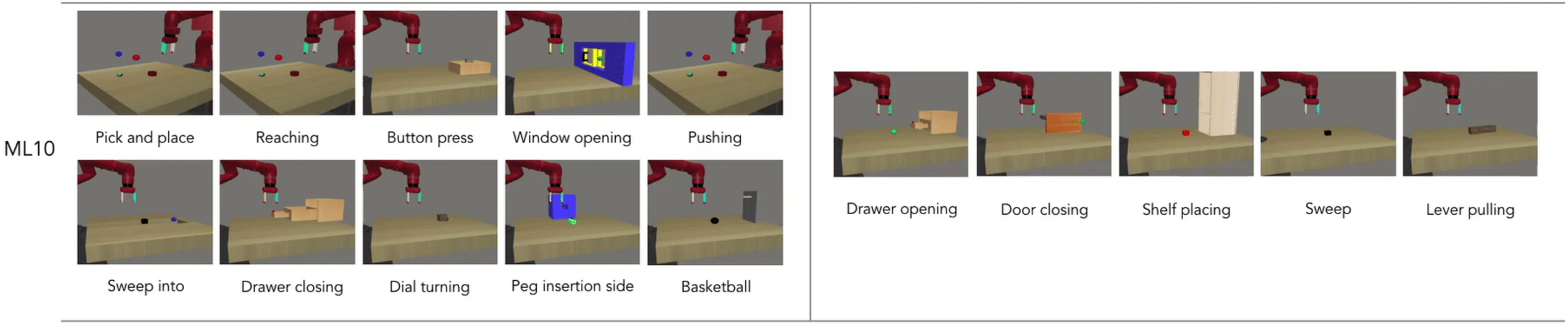


Example: meta-reinforcement learning

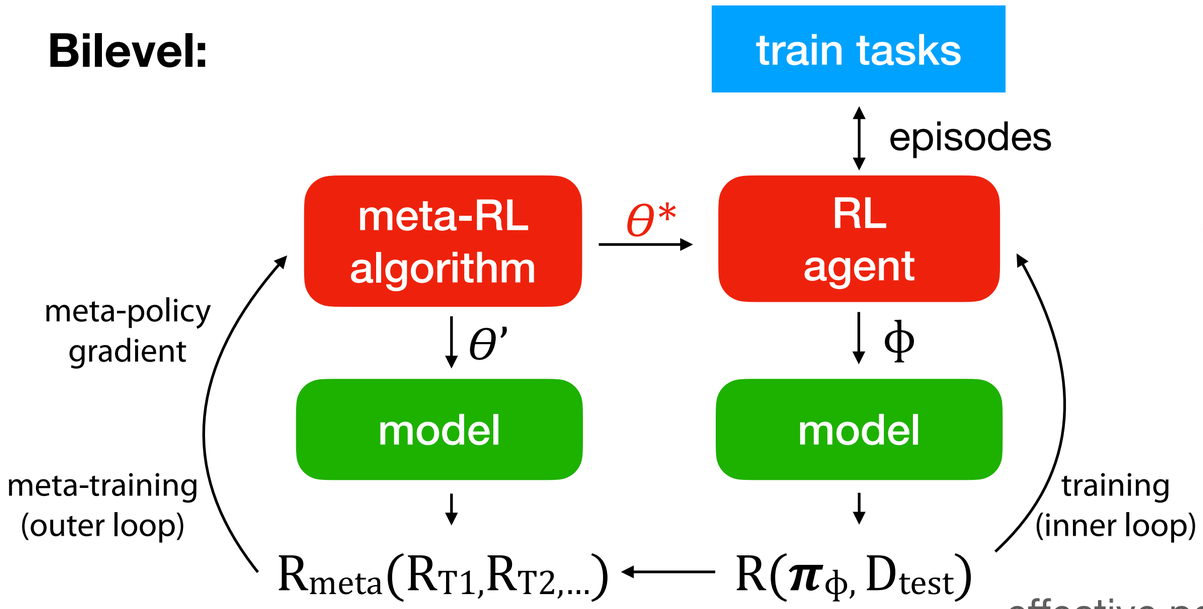
Yu et al. 2019

initial state:
randomized object
and goals positions

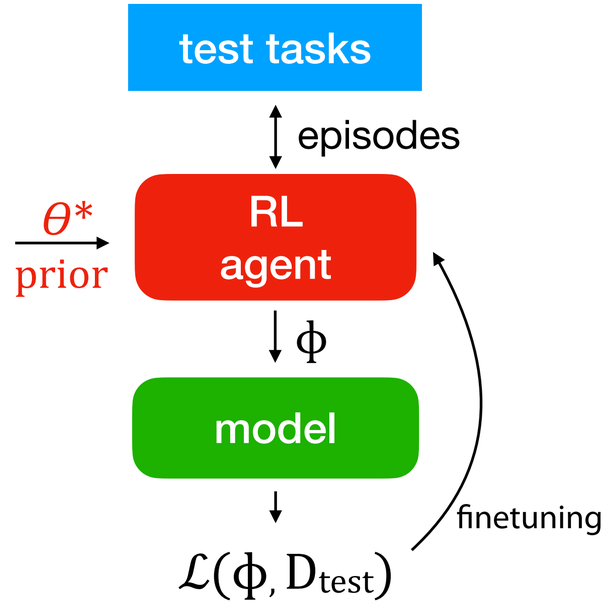
other initial states
or related tasks



Bilevel:



θ^*: fixed ϕ objective, e.g. which ϕ to learn



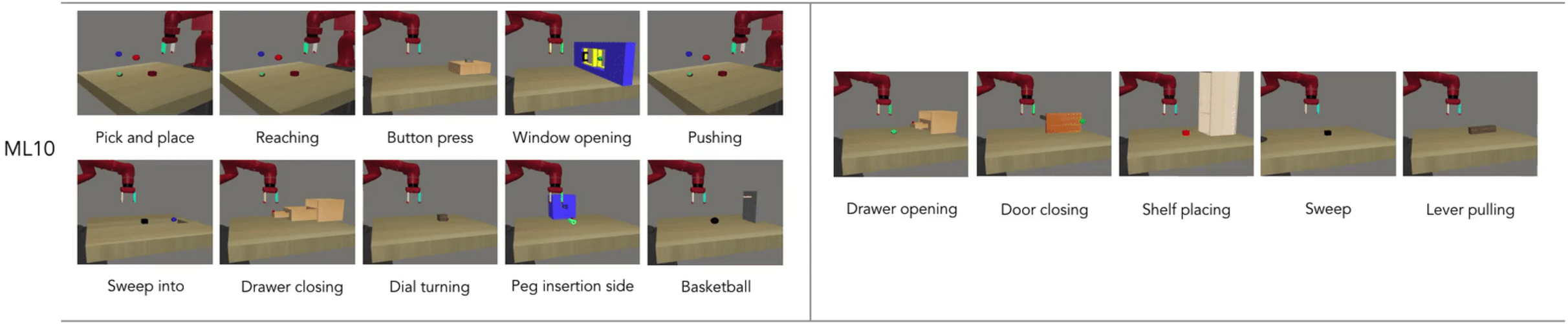
effective policy π_ϕ modulated by ϕ

Example: meta-reinforcement learning

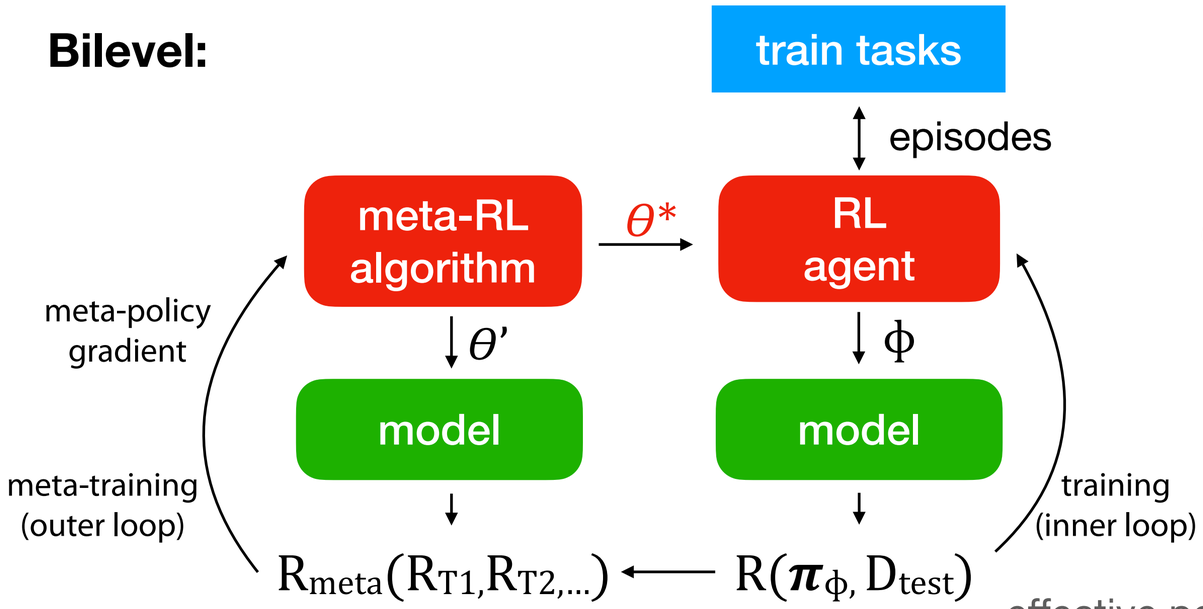
Yu et al. 2019

initial state:
randomized object
and goals positions

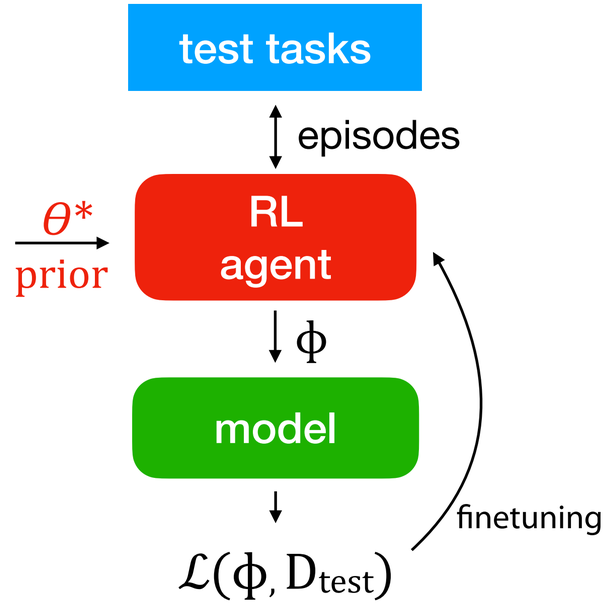
other initial states
or related tasks



Bilevel:



θ^*: fixed ϕ objective, e.g. which ϕ to learn



effective policy π_ϕ modulated by ϕ

Example: meta-reinforcement learning

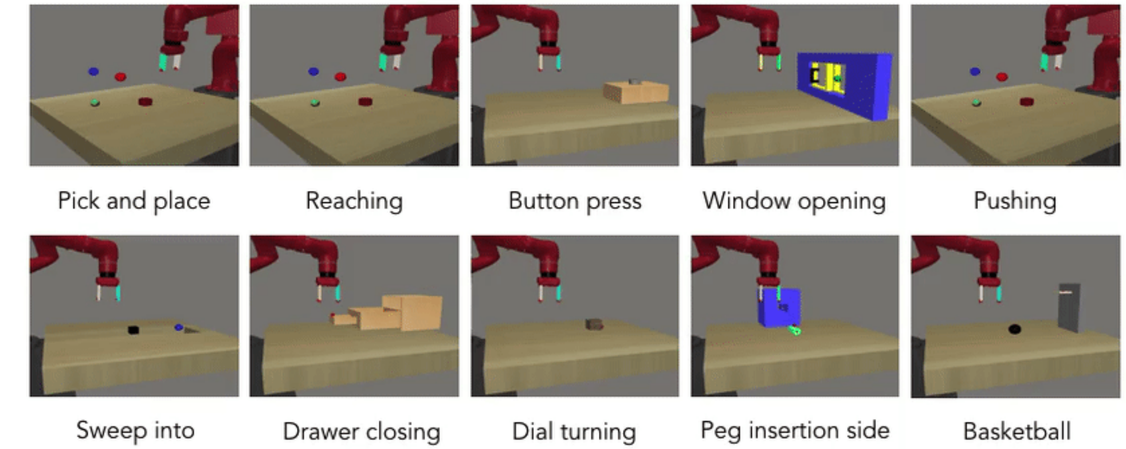
initial state:
randomized object
and goals positions

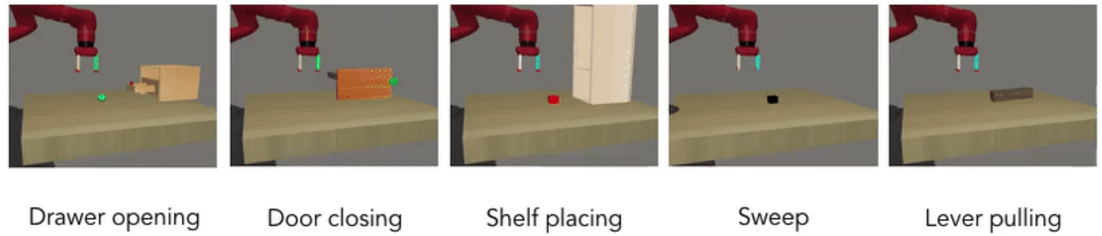
Train tasks

Test tasks

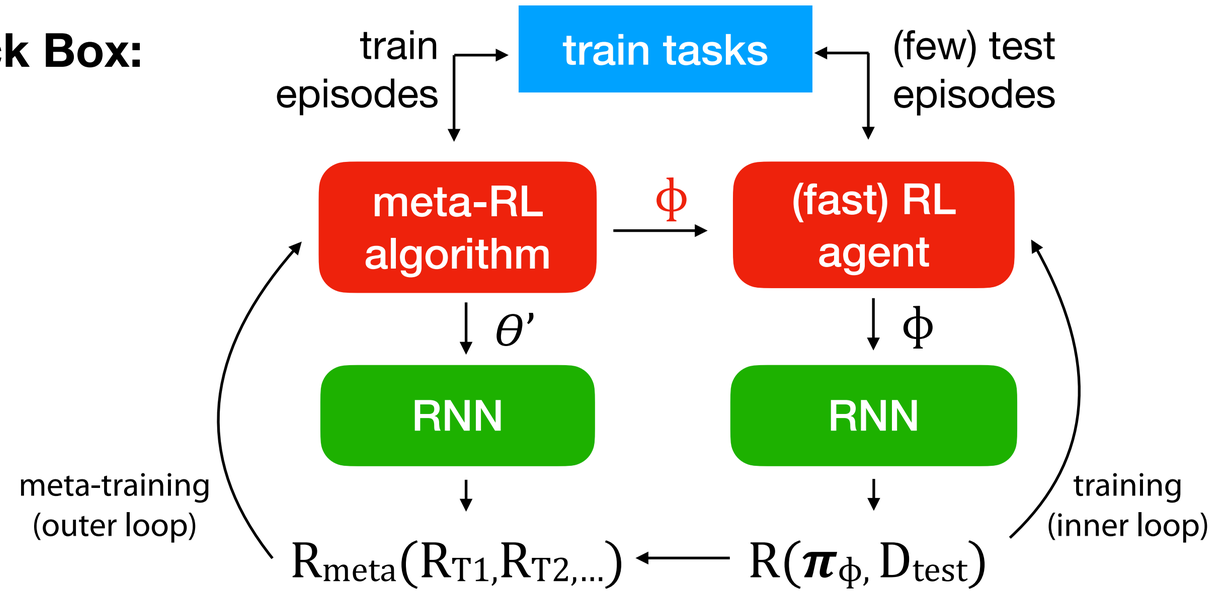
other initial states
or related tasks

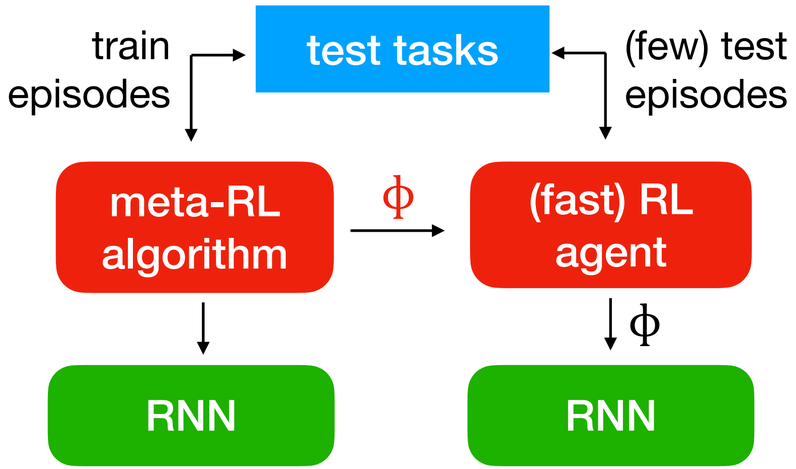
ML10





Black Box:





Example: meta-reinforcement learning

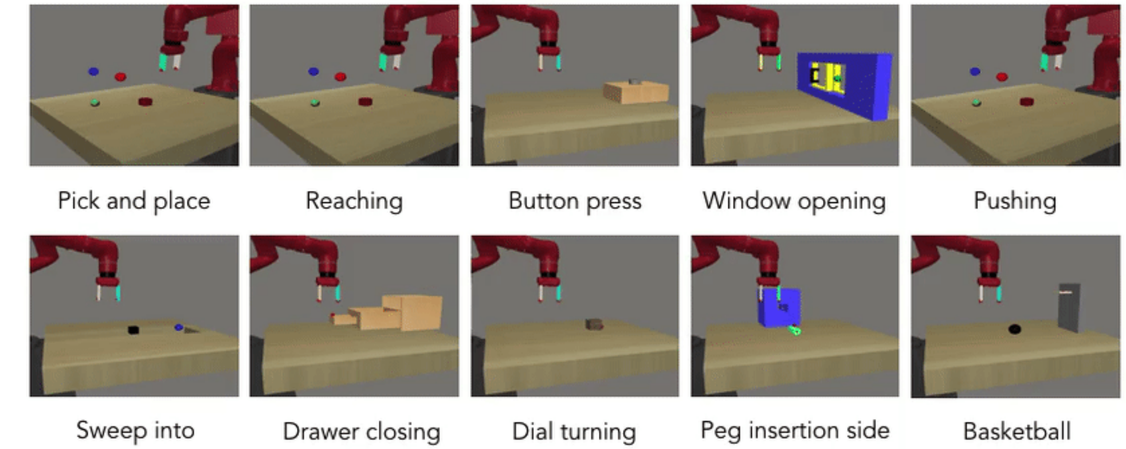
initial state:
randomized object
and goals positions

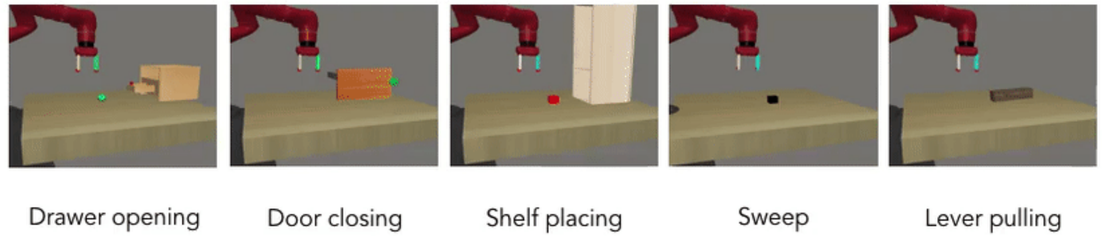
Train tasks

Test tasks

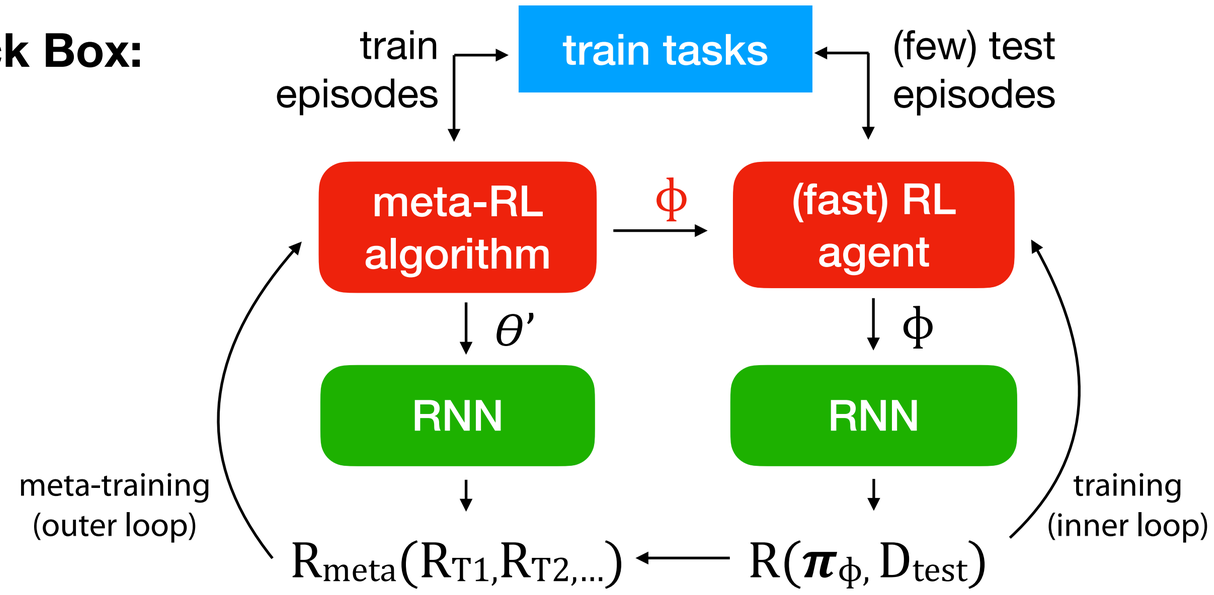
other initial states
or related tasks

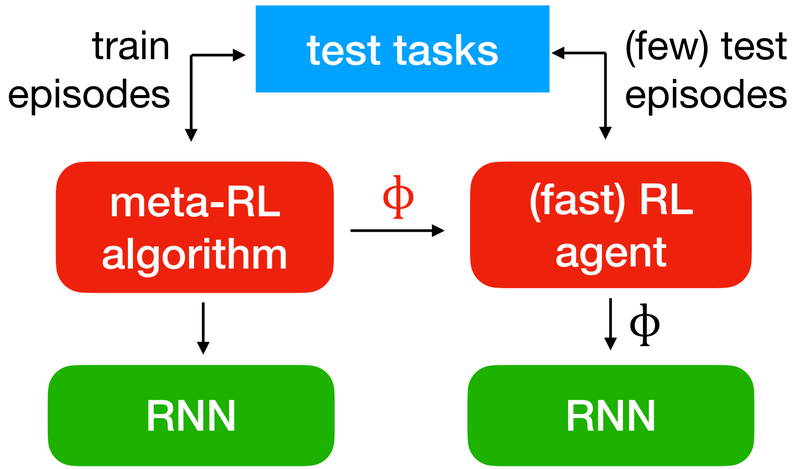
ML10





Black Box:





Hospedales et al. 2020

Taxonomy of meta-learning methods

like base-learners, meta-learners consist of a representation, an objective, and an optimizer

meta-learning algorithm

meta-parameters θ

meta-optimizer F_{meta}

meta-objective $\mathcal{L}_{\text{meta}}$

parameter initialization ϕ_0

gradient descent

multi vs single task

random ϕ_0

learned ϕ_0

Hospedales et al. 2020

Taxonomy of meta-learning methods

like base-learners, meta-learners consist of a representation, an objective, and an optimizer

meta-learning algorithm

meta-parameters θ	*meta-optimizer* F_{meta}	*meta-objective* $\mathcal{L}_{\text{meta}}$
optimizer	gradient descent	multi vs single task

normal SGD
(back-propagation)

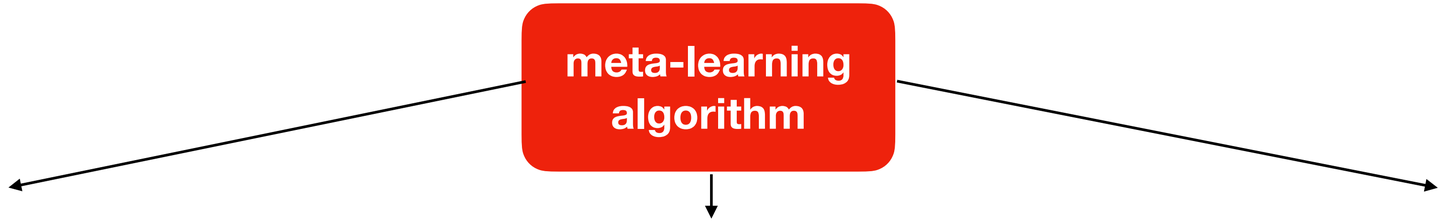
learned optimizer
(update rule)

Taxonomy of meta-learning methods

Hospedales et al. 2020

Huisman et al. 2020

like base-learners, meta-learners consist of a representation, an objective, and an optimizer



meta-parameters θ

- parameter initialization ϕ_0
- optimizer
- black-box model
- embedding (metric learning)
- hyperparameters λ
- architecture
- loss $\mathcal{L}$ / reward R
- ...

meta-optimizer F_{meta}

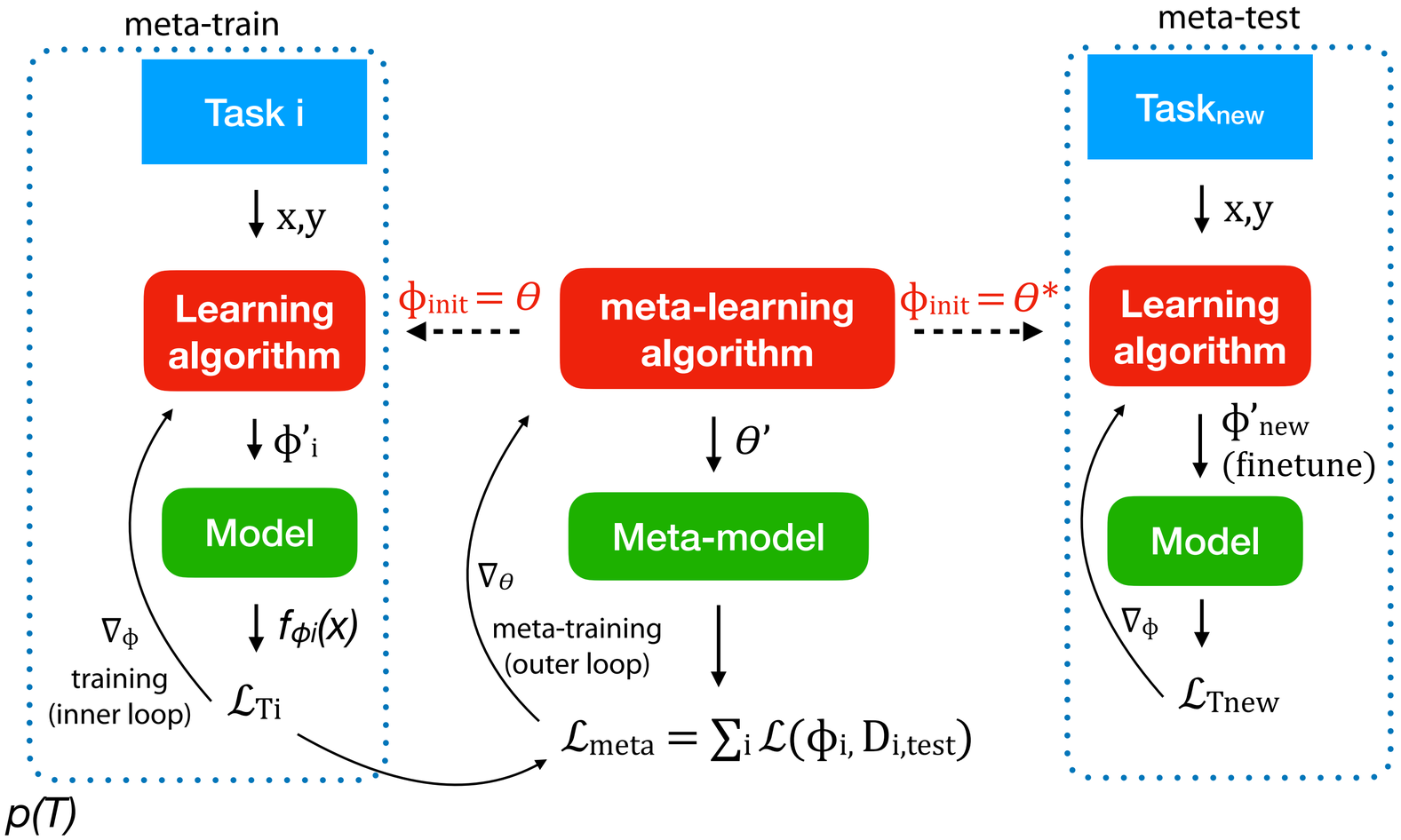
- gradient descent
- reinforcement learning
- evolution

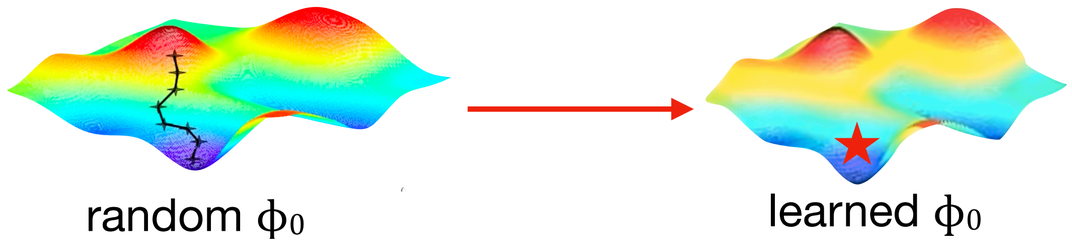
meta-objective $\mathcal{L}_{\text{meta}}$

- multi vs single task
- many vs few shots
- fast adaptation vs final perf.
- offline vs online

We won't exhaustively discuss all existing combinations, let's focus on understanding a few key strategies

Gradient-based methods: learning ϕ_{init}





- θ (prior): model initialization ϕ_{init}
  - learn representation suitable for many tasks (e.g. pretrained CNN)
  - maximize rapid learning
- Each task i yields task-adapted ϕ_i
  - Update algorithm u

$$\phi_i = u(\theta, D_{i,train})$$

- Finetune θ^* on T_{new} (in few steps)

$$\phi'_{new} = u(\theta^*, D_{new,train})$$

Can be seen as bilevel optimization:

$$\theta^* = \text{argmin}_\theta \sum_i \mathcal{L}_i(\phi_i, D_{i,test})$$

$$\phi_i = u(\theta, D_{i,train})$$

Finn et al. 2017

Model agnostic meta-learning (MAML)

Meta-training

- Current initialization θ, model f_θ
- On i tasks, perform k SGD steps to find ϕ_i^*, then evaluate $\nabla_\theta \mathcal{L}_i(f_{\phi_i^*})$

derivative of test-set loss

- Update task-specific parameters: $\phi_i = \theta - \alpha \nabla_\theta \mathcal{L}_i(f_{\phi_i^*})$
- Update θ to minimize sum of per-task losses, repeat

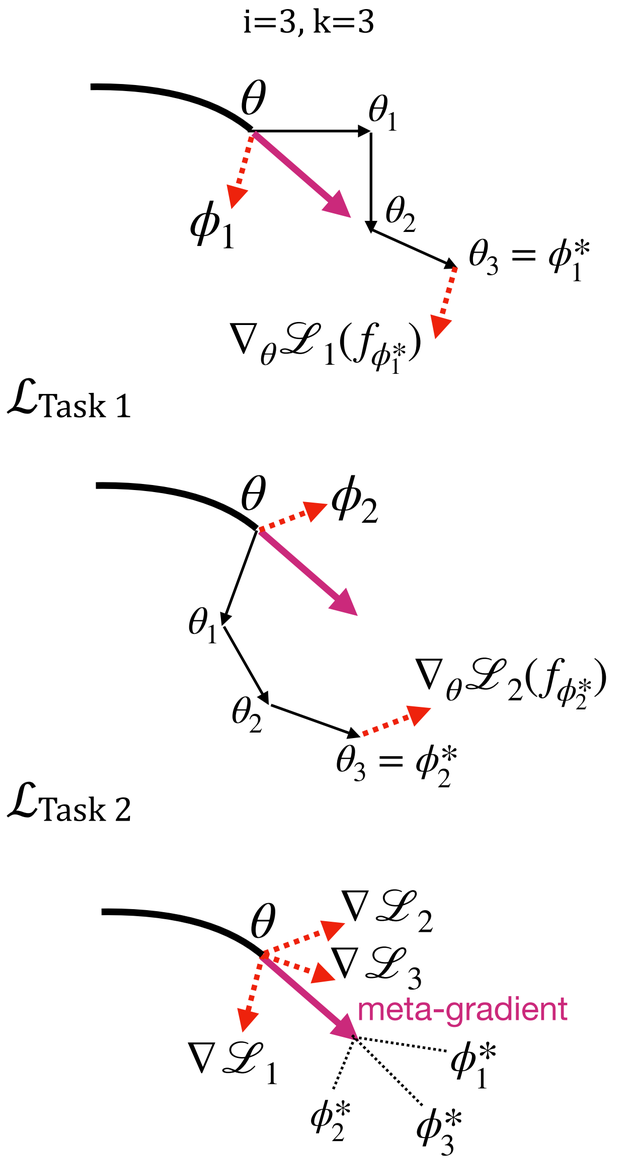
$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_i \mathcal{L}_i(f_{\phi_i})$$

$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_i \mathcal{L}_i(f(\theta - \alpha \nabla_\theta \mathcal{L}_i(f_{\phi_i^*})))$$

α, β: learning rates

meta-gradient: *second-order gradient* + backpropagate

compute how changes in Θ affect the gradient at new Θ



Meta-testing

- Training data of new task D_{train}
- θ^*: pre-trained parameters
- Finetune: $\phi = \theta^* - \alpha \nabla_\theta \mathcal{L}(f_\theta)$

Finn et al. 2017

Model agnostic meta-learning (MAML)

Meta-training

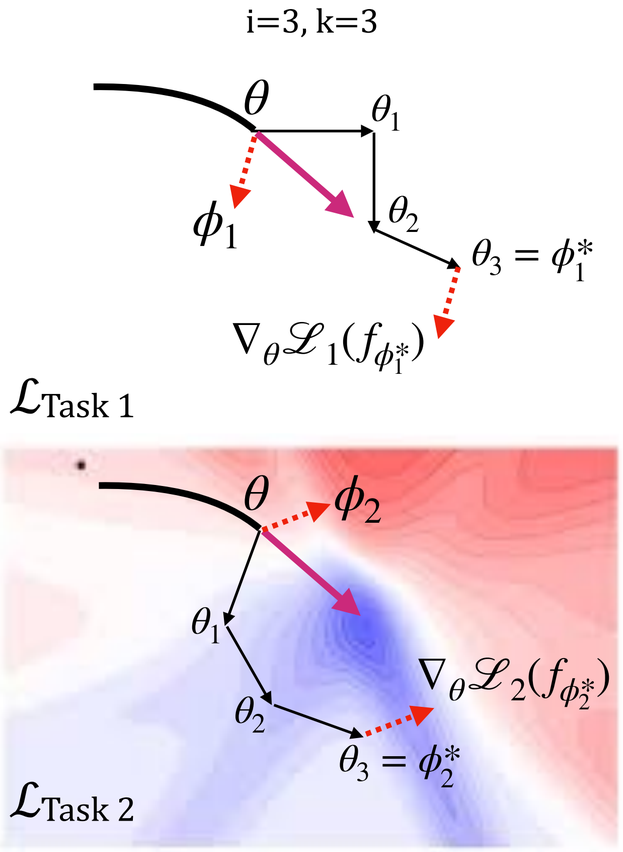
- Current initialization θ, model f_θ
- On i tasks, perform k SGD steps to find ϕ_i^*, then evaluate $\nabla_\theta \mathcal{L}_i(f_{\phi_i^*})$ ← derivative of test-set loss
- Update task-specific parameters: $\phi_i = \theta - \alpha \nabla_\theta \mathcal{L}_i(f_{\phi_i^*})$
- Update θ to minimize sum of per-task losses, repeat

$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_i \mathcal{L}_i(f_{\phi_i})$$

$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_i \mathcal{L}_i(f(\theta - \alpha \nabla_\theta \mathcal{L}_i(f_{\phi_i^*})))$$

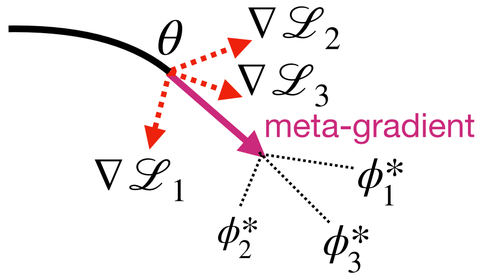
α, β: learning rates

→ meta-gradient: *second-order gradient* + backpropagate
compute how changes in Θ affect the gradient at new Θ



Meta-testing

- Training data of new task D_{train}
- θ^*: pre-trained parameters
- Finetune: $\phi = \theta^* - \alpha \nabla_\theta \mathcal{L}(f_\theta)$



Finn et al. 2017

Model agnostic meta-learning (MAML)

Meta-training

- Current initialization θ, model f_θ
- On i tasks, perform k SGD steps to find ϕ_i^*, then evaluate $\nabla_\theta \mathcal{L}_i(f_{\phi_i^*})$

derivative of test-set loss

- Update task-specific parameters: $\phi_i = \theta - \alpha \nabla_\theta \mathcal{L}_i(f_{\phi_i^*})$
- Update θ to minimize sum of per-task losses, repeat

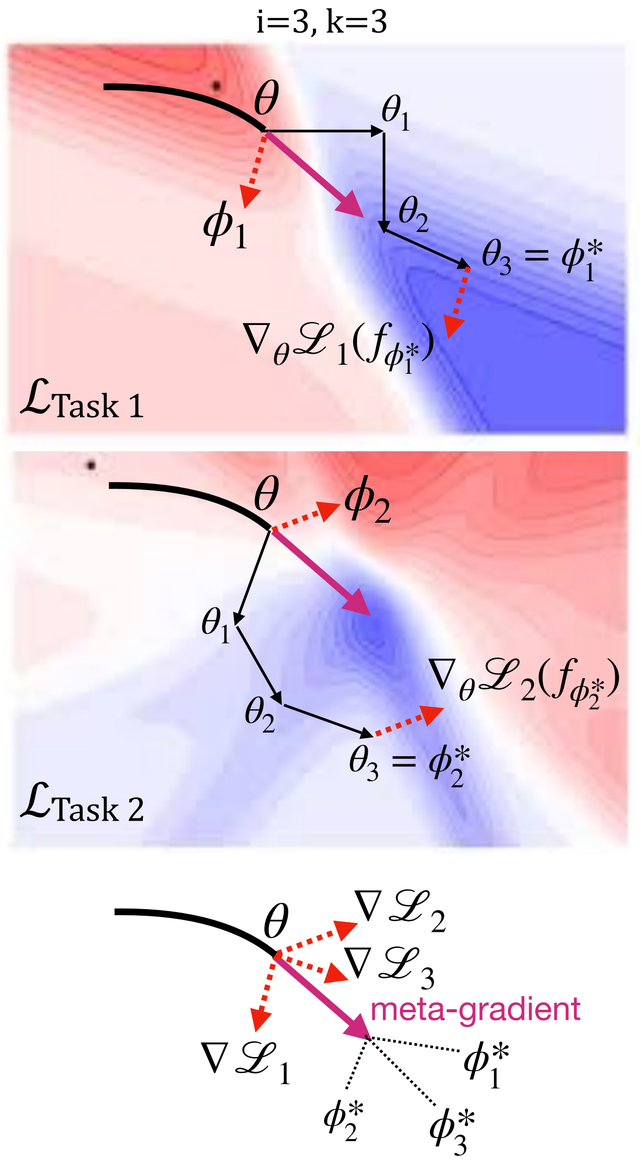
$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_i \mathcal{L}_i(f_{\phi_i})$$

$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_i \mathcal{L}_i(f(\theta - \alpha \nabla_\theta \mathcal{L}_i(f_{\phi_i^*})))$$

α, β: learning rates

meta-gradient: second-order gradient + backpropagate

compute how changes in Θ affect the gradient at new Θ

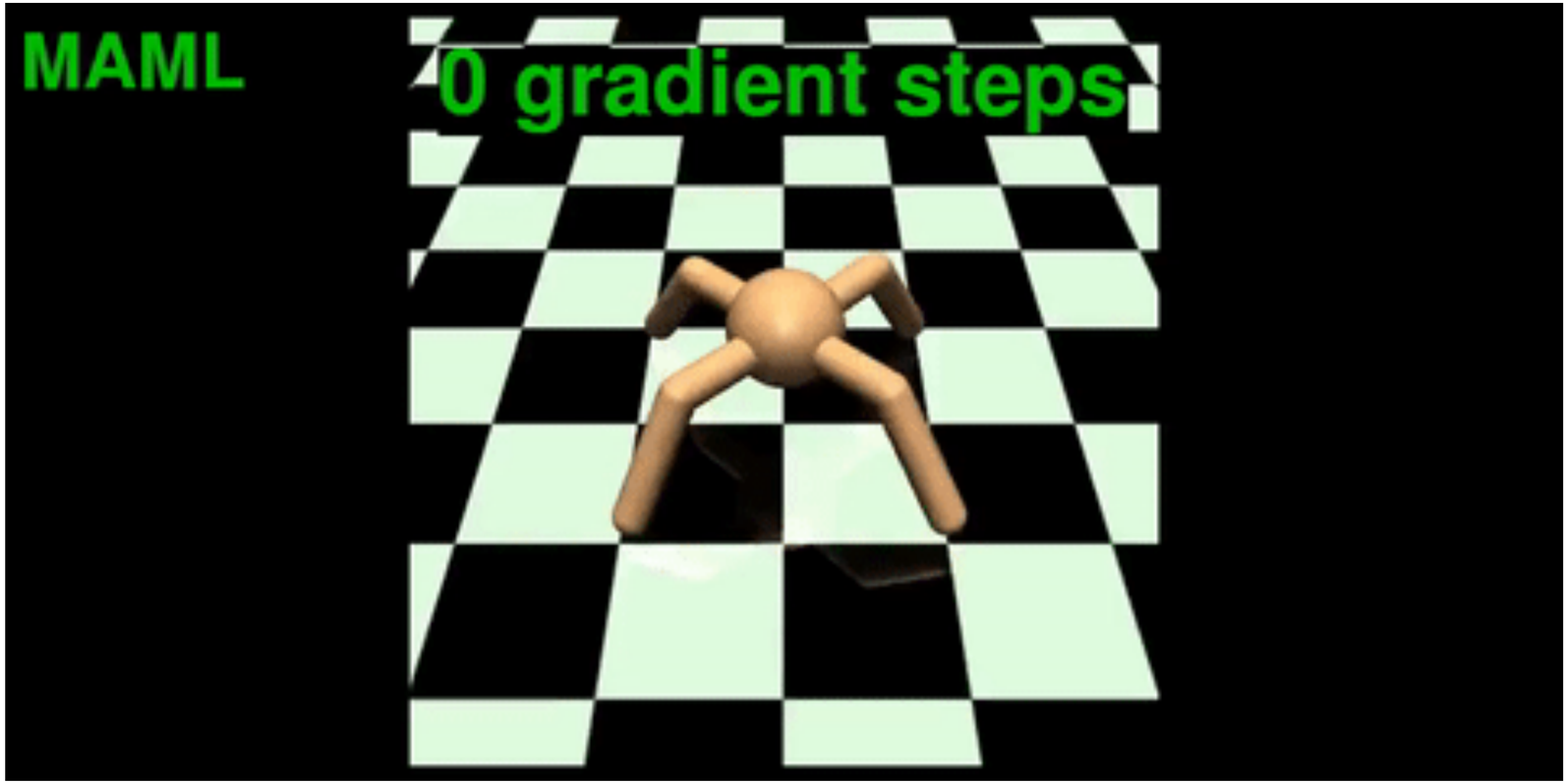


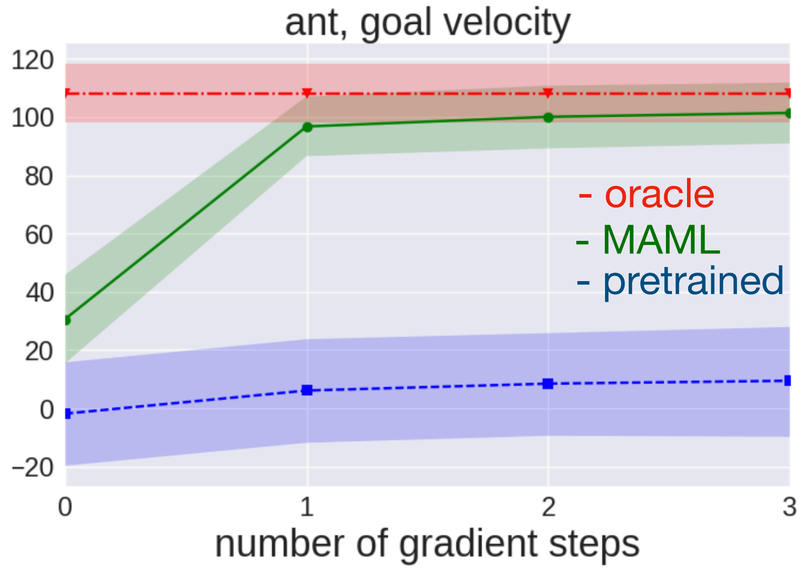
Meta-testing

- Training data of new task D_{train}
- θ^*: pre-trained parameters
- Finetune: $\phi = \theta^* - \alpha \nabla_\theta \mathcal{L}(f_\theta)$

Model agnostic meta-learning (MAML)

- Example for reinforcement learning:
  - Goal: reach certain velocity in certain direction

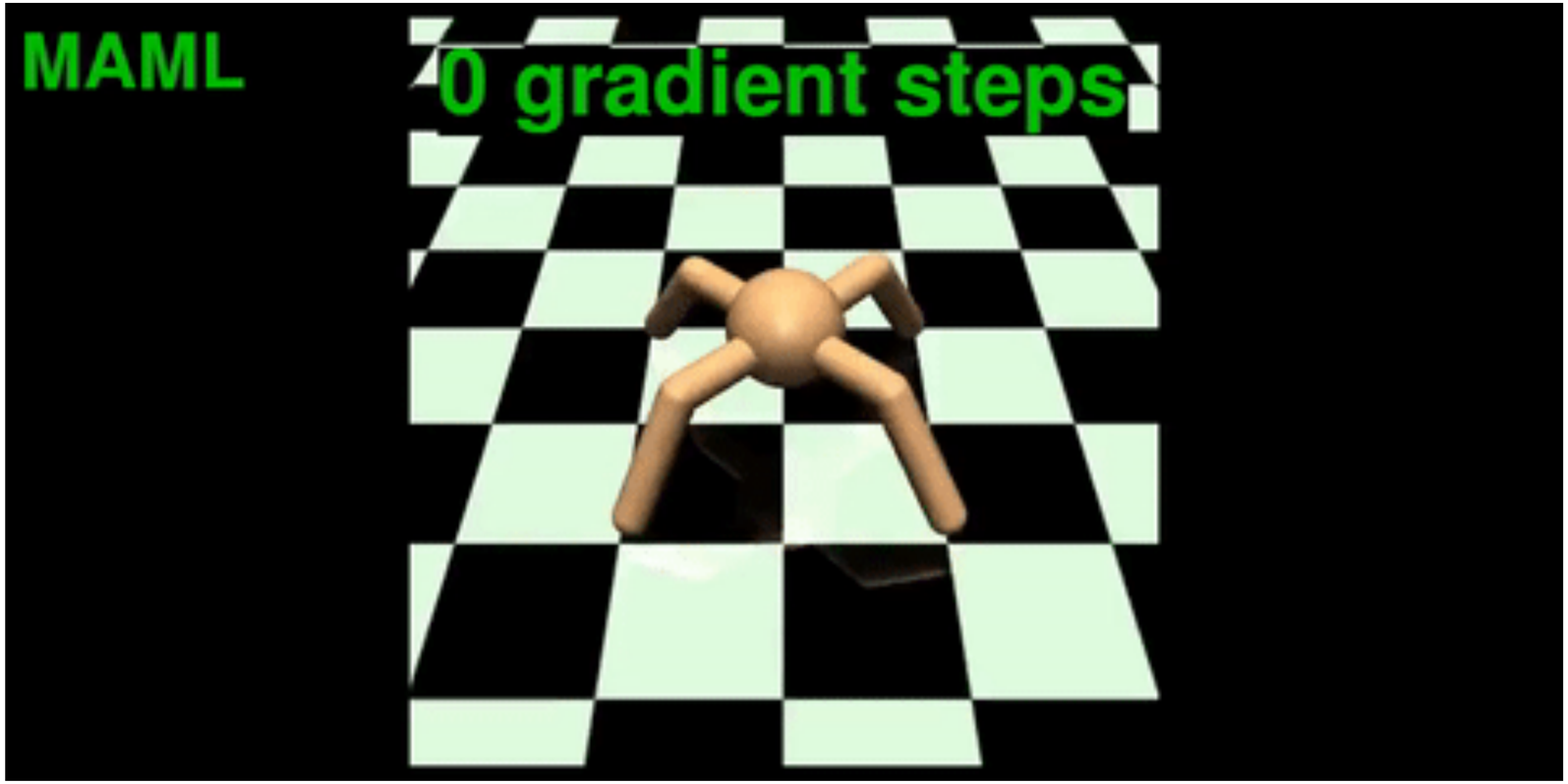


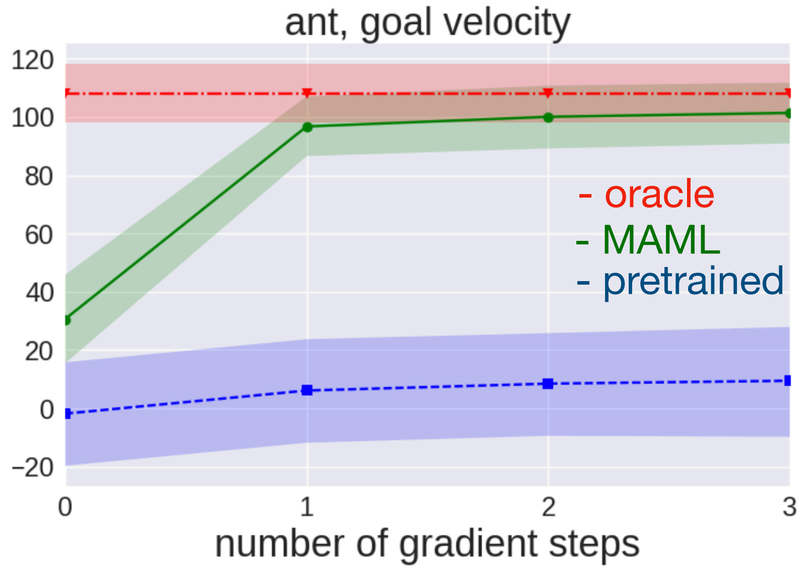


Source: Finn et al. 2017

Model agnostic meta-learning (MAML)

- Example for reinforcement learning:
  - Goal: reach certain velocity in certain direction





Source: Finn et al. 2017

Other gradient-based techniques

[1] Finn et al. 2017
[2] Li et al. 2017
[3] Lee et al. 2018
[4] Park et al. 2019
[5] Flennerhag et al. 2019
[6] Antoniou et al. 2019
[7] Finn et al. 2019

- Changing update rule yield different variants:
  - MAML [1,6] $\theta \leftarrow \theta - \beta \nabla_\theta \sum_i \mathcal{L}_i(f_{\phi_i})$ where $\phi_i = \theta - \alpha \nabla_\theta \mathcal{L}_i(f_{\phi_i^*})$
  - MetaSGD [2] $\phi_i = \theta - \alpha \; diag(w) \nabla_\theta \mathcal{L}_i(f_{\phi_i^*})$
  - Tnet [3] $\phi_i = \theta - \alpha \nabla_\theta \mathcal{L}_i(f_{\phi_i^*}, w)$
  - Meta curvature [4] $\phi_i = \theta - \alpha \; B(\theta, w) \nabla_\theta \mathcal{L}_i(f_{\phi_i^*})$
  - WarpGrad [5] $\phi_i = \theta - \alpha \; P(\theta, \phi_i) \nabla_\theta \mathcal{L}_i(f_{\phi_i^*})$

w: *weight per parameter*

All use second-order gradients, Meta-learn a transformation of the gradient for better adaptation

- Online MAML (Follow the Meta Leader) [7]
  - Minimizes regret
  - Robust, but computation costs grow over time

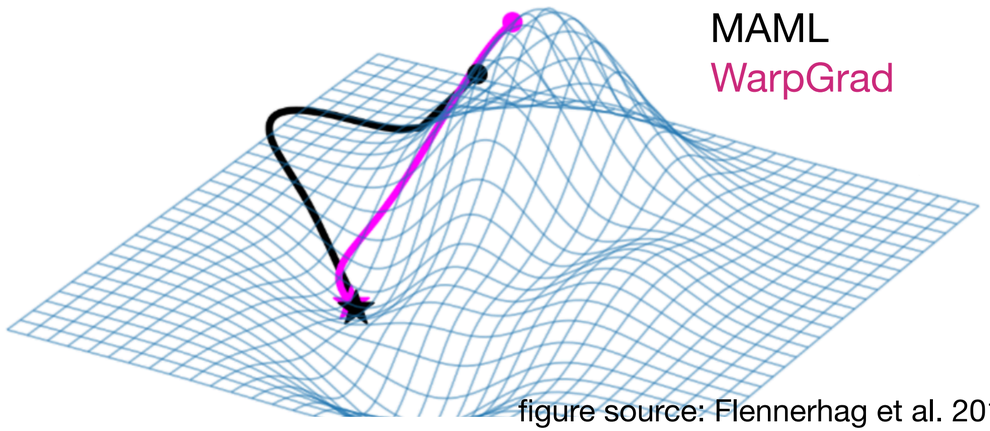


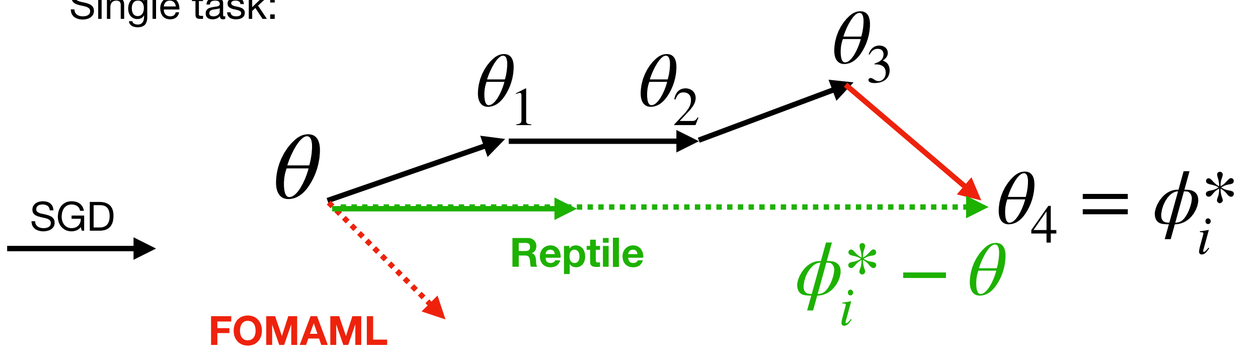
figure source: Flennerhag et al. 2019

Scalability

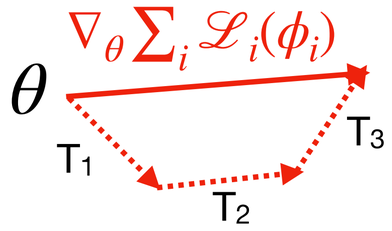
[1] Finn et al. 2017
[2] Nichol et al. 2018
[3] Rajeswaran et al. 2019

- Backpropagating derivates of ϕ_i wrt θ is compute + memory intensive (for large models)
- First order approximations of MAML:
  - First order MAML[1] uses only the last inner gradient update: $\theta \leftarrow \theta - \beta \sum_i \mathcal{L}_i(f_{\phi_i^*})$
  - Reptile [2]: iterate over tasks, update θ in direction of ϕ_i^*: $\theta \leftarrow \theta - \beta\,(\phi_i^* - \theta)$

Single task:



FOMAML meta-gradient:



Scalability (2)

[1] Rajeswaran et al. 2019

- Implicit MAML[1]: *uses* an approximate gradient
  - Compute derivative of ϕ_i^* wrt θ
  - ϕ_i^* could be anywhere. Hence, add penalty: $\|\phi_i^* - \theta\|^2$
  - Accurate if we stay close to θ
  - $\mathcal{L}_i(\phi_i) + \lambda\|\phi_i^* - \theta\|^2$ has closed form solution

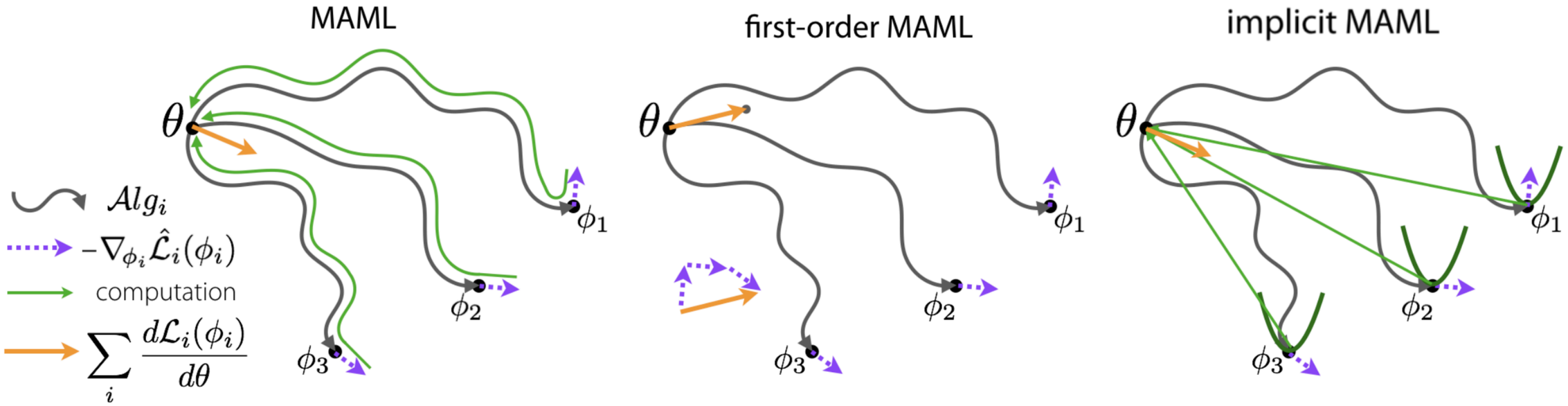
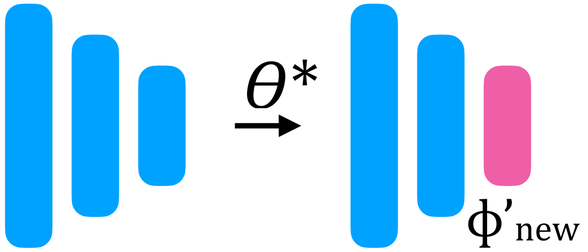


figure source: Rajeswaran et al. 2019

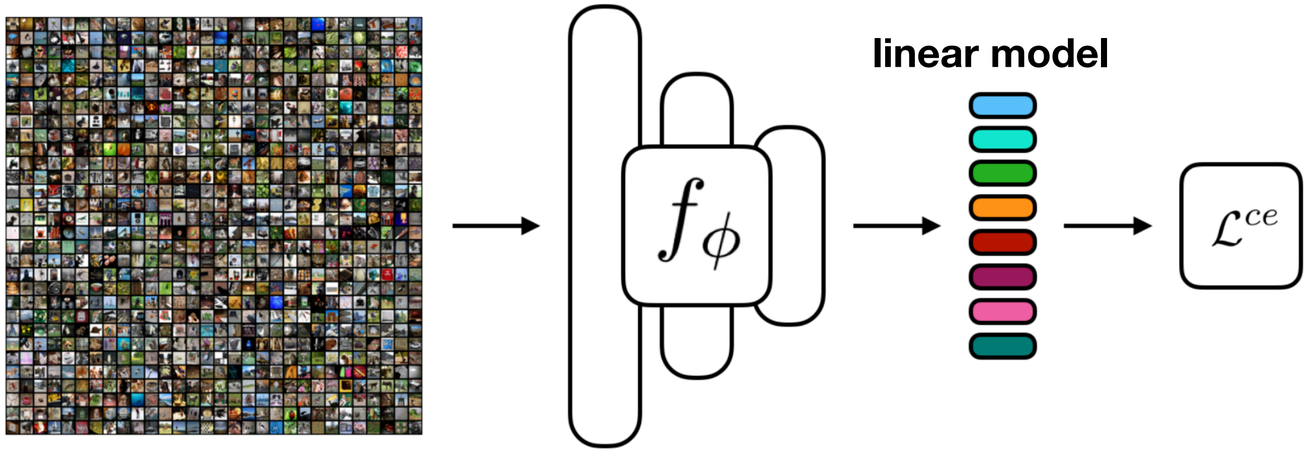
Generalizability

[1] Finn et al. 2018
[2] Raghu et al. 2020
[3] Tian et al. 2020
[4] Stadie et al. 2019

- MAML is more resilient to overfitting than many other meta-learning techniques [1]
- Effectiveness seems mainly due to feature reuse (finding Θ) [2]
  - Fine-tuning *only the last layer* equally good



- On few-shot learning benchmarks, a good embedding outperforms most meta-learning [3]
  - Learn representation on entire meta-dataset (merged into single task)
  - Train a linear classifier on embedded few-shot D_{train} , predict D_{test}



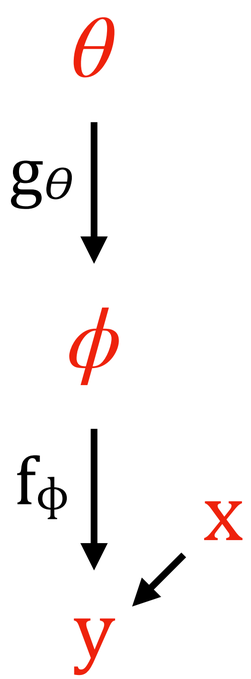
- For meta-RL, also learn how to *explore* (how to sample new environments)
  - E-MAML: add exploration to meta-objective (allows longer term goals) [4]

figure source: Tian et al. 2020

Bayesian meta-learning

Grant et al. 2018

Can meta-learning reason about uncertainty in the task distribution?



$$T_i = \{D_{train}, D_{test}\} \leftarrow \mathcal{T}_{meta-train}$$

$$\hat{\phi} \leftarrow g_\theta((x,y)_{train})$$

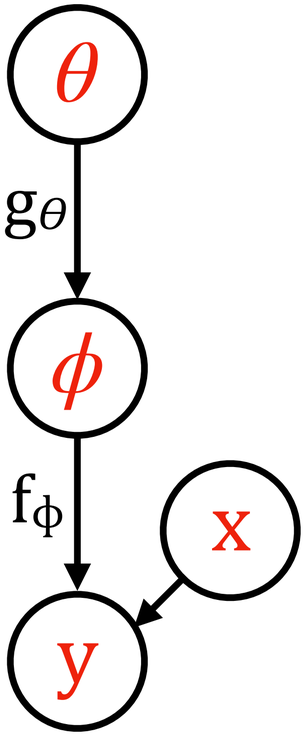
$$y_{test} \leftarrow f_{\hat{\phi}}(x_{test})$$

$$\theta \leftarrow \theta - \nabla_\theta \mathcal{L}(y_{test})$$

$$T_j = \{D_{train}, D_{test}\} \leftarrow \mathcal{T}_{meta-test}$$

$$\phi_j \leftarrow g_\theta((x,y)_{train})$$ *finetuning*

$$y_{test} \leftarrow f_{\phi_j}(x_{test})$$



$$T_i = \{D_{train}, D_{test}\} \leftarrow \mathcal{T}_{meta-train}$$

$$p(\phi_i \mid (x,y)_{train})$$

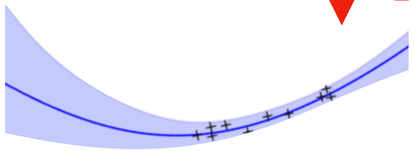
$$p(y_{test} \mid x_{test})$$

$$p(\theta) \leftarrow p(\theta \mid (x,y)_{train})$$

$$T_j = \{D_{train}, D_{test}\} \leftarrow \mathcal{T}_{meta-test}$$

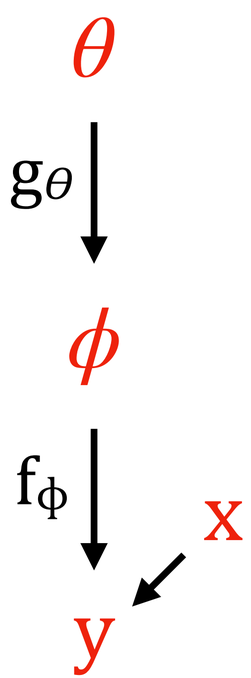
$$p(\phi_j \mid (x,y)_{train}, \theta)$$ *finetuning*

$$p(y_{test} \mid x_{test})$$



Bayesian meta-learning

Grant et al. 2018



$$T_i = \{D_{train}, D_{test}\} \leftarrow \mathcal{T}_{meta-train}$$

$$\hat{\phi} \leftarrow g_\theta((x,y)_{train})$$

$$y_{test} \leftarrow f_{\hat{\phi}}(x_{test})$$

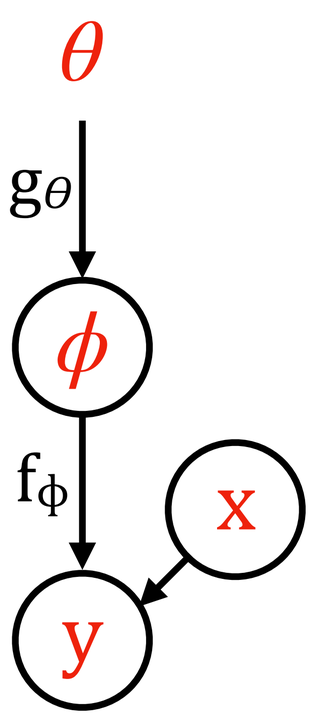
$$\theta \leftarrow \theta - \nabla_\theta \mathcal{L}(y_{test})$$

$$T_j = \{D_{train}, D_{test}\} \leftarrow \mathcal{T}_{meta-test}$$

$$\phi_j \leftarrow g_\theta((x,y)_{train})$$ *finetuning*

$$y_{test} \leftarrow f_{\phi_j}(x_{test})$$

Empirical Bayes: fit point estimate on Θ



$$T_i = \{D_{train}, D_{test}\} \leftarrow \mathcal{T}_{meta-train}$$

$$\hat{\phi} = argmax_\phi p(y \mid x, \phi) p(\phi \mid \theta)$$

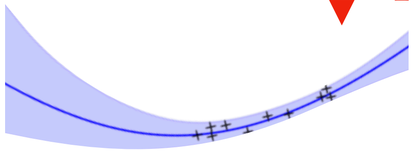
$$p(y_{test} \mid x_{test})$$

$$\theta \leftarrow argmax_\theta p(y_{train} \mid x_{train}, \theta)$$

$$T_j = \{D_{train}, D_{test}\} \leftarrow \mathcal{T}_{meta-test}$$

$$p(\phi_j \mid (x,y)_{train}, \theta)$$ *finetuning*

$$p(y_{test} \mid x_{test})$$



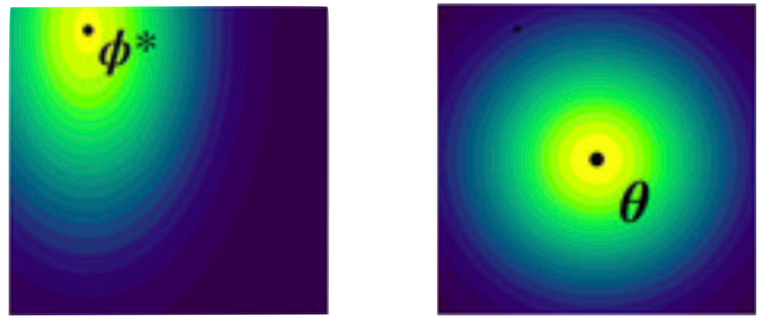
Bayesian meta-learning

[1] Grant et al. 2018

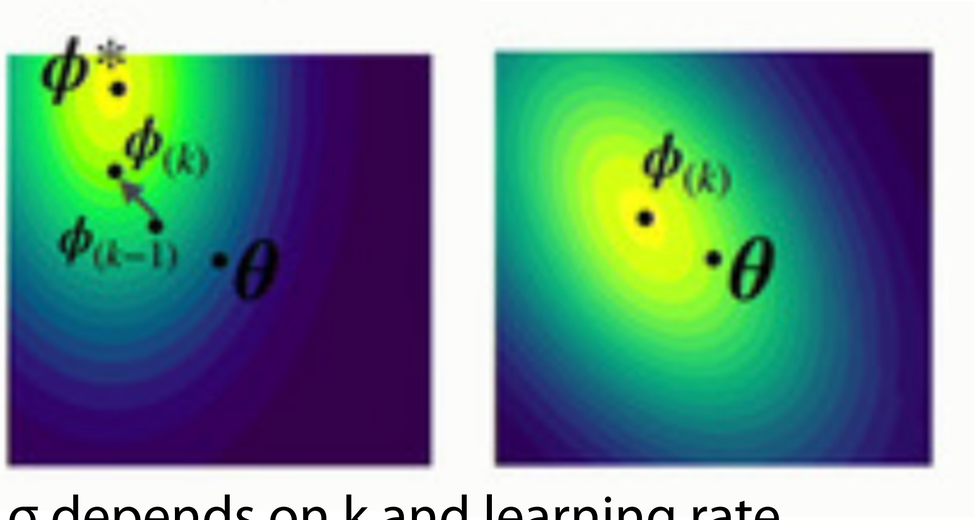
ϕ can be factored as the product of a likelihood and a prior

$$\hat{\phi} = argmax_{\phi}\ p(y \mid x, \phi)\ p(\phi \mid \theta)$$

$p(y \mid x, \phi)$ $p(\phi \mid \theta)$



When using θ as the initialization of gradient descent, early stopping is approximately equivalent to placing a prior over the task specific parameters [1]



σ depends on k and learning rate

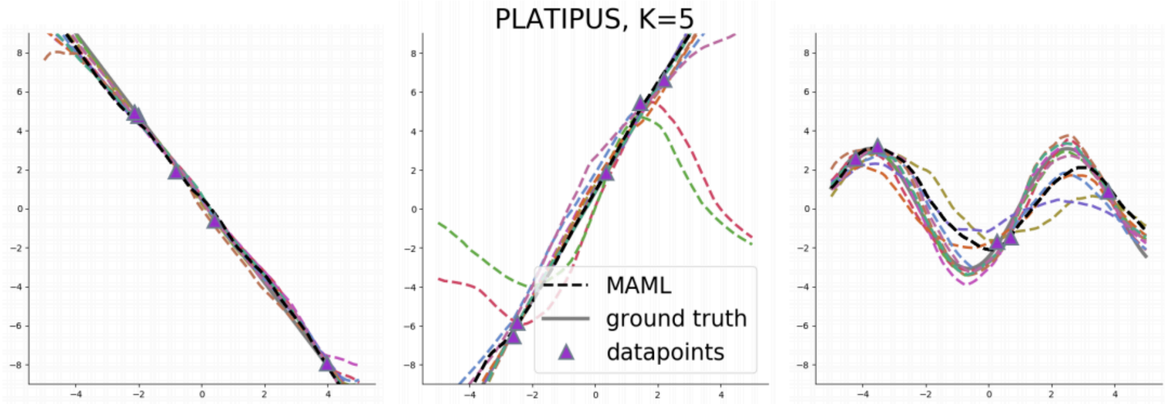
MAML is doing the same

- estimates ϕ_i^* with k steps of gradient descent
- update prior θ to makes it easier to optimize ϕ_i

Image source: Erin Grant. 2019

Fully Bayesian meta-learning

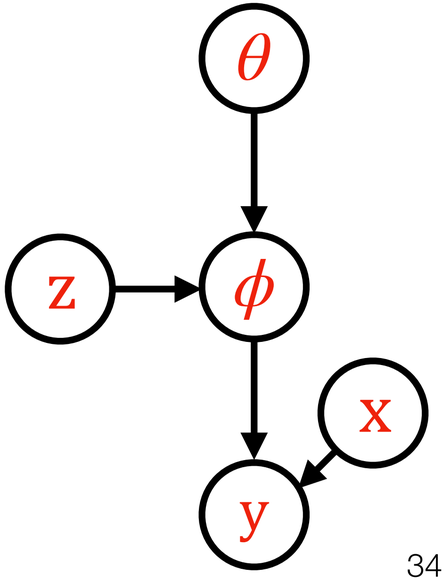
- Alternatively, use approximation methods to represent uncertainty over ϕ_i
  - Sampling technique + variational inference: PLATIPUS [1], BMAML [2], ABML [3]
  - Laplace approximation: LLAMA [4]
  - Variational approximation of posterior:
    - Neural Statistician [5], Neural Processes [6]
- What if our tasks are not IID?
  - Impose additional structure, e.g. with task-specific variable z [7]



mini-ImageNet with filters for non-homogeneous tasks:

(a) plain **(b)** blur **(c)** night **(d)** pencil

Figure source: Jerfel et al. 2019



[1] Finn et al. 2019
[2] Kim et al. 2018
[3] Ravi et al. 2019
[4] Grant et al. 2018
[5] Edwards et al. 2017
[6] Garnelo et al. 2018
[7] Jerfel et al. 2019

Meta-learning optimizers

[1] Bengio et al. 1995

[2] Schmidhuber 1992

[3] Runarsson and Jonsson 2000

[4] Hochreiter 2001

- Our brains *probably* don't do backprop, instead:
  - Simple bio-inspired rules to update weights [1]

$$\phi_i' = \phi_i - \eta \; y_{pre(i)} \; k$$

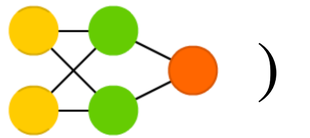
presynaptic activity

learning rate

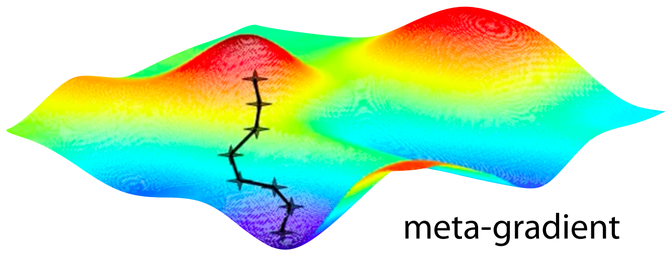
reinforcing signal

  - Fast weights [2]: networks that continuously modify the weights of another network
  - Gradient based [3, 4]: parameterize the update rule using a neural network
    - Learn meta-parameters across tasks, by gradient descent

$$\phi_i' = \phi_i - \eta(\quad)$$



θ



meta-gradient

Meta-learning optimizers

[1] Hochreiter 2001
[2] Andrychowicz et al. 2016
[3] Ravi and Larochelle 2017
[4] Wichrowska et al. 2017

- Represent update rule as an LSTM [1,2,3], hierarchical RNN [4]
  - Optimizee: receives weight update g_t from meta-learned optimizer
  - Optimizer: receives gradient estimate ∇_t from optimizee
- Meta-learner learns optimizer parameters with gradient descent across tasks
  - e.g. Image classification [2,4], few-shot learning [3]

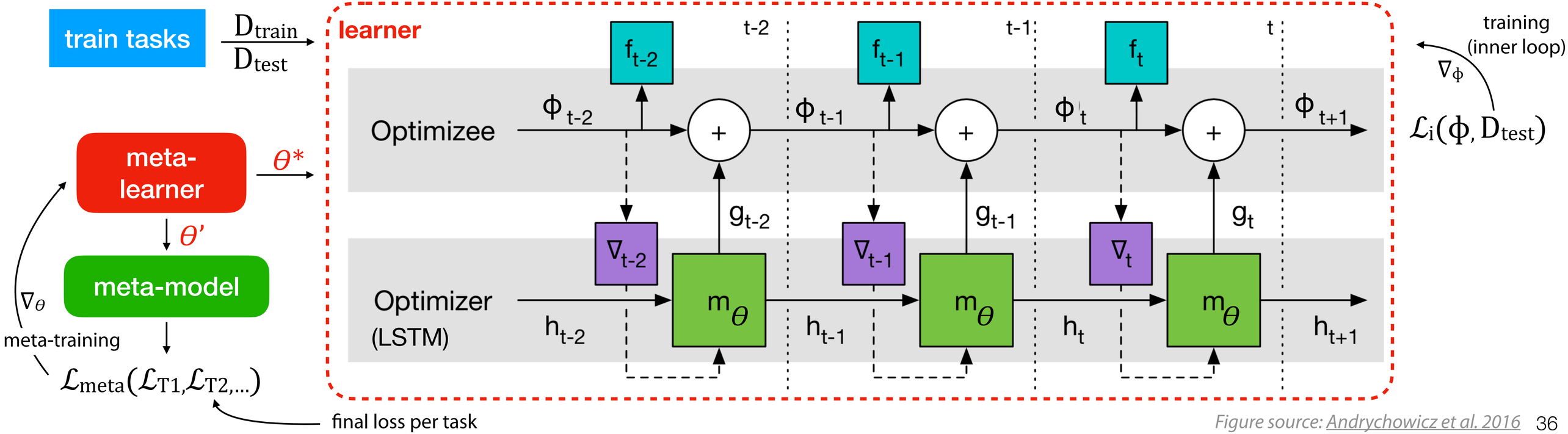


Figure source: Andrychowicz et al. 2016

Meta-learning optimizers

- Meta-learned (RNN) optimizers 'rediscover' momentum, gradient clipping, learning rate schedules, learning rate adaptation,… [1]

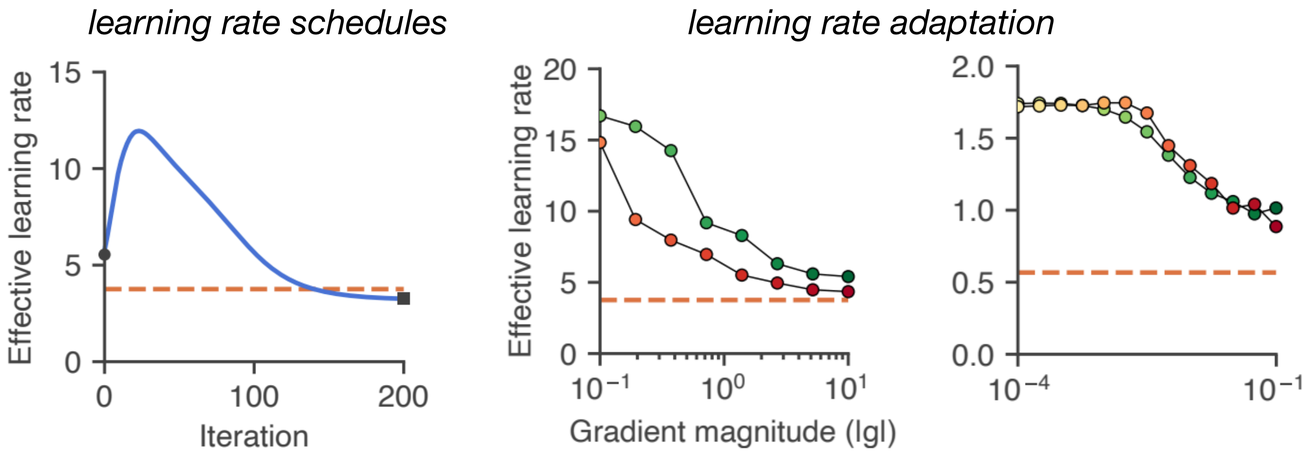


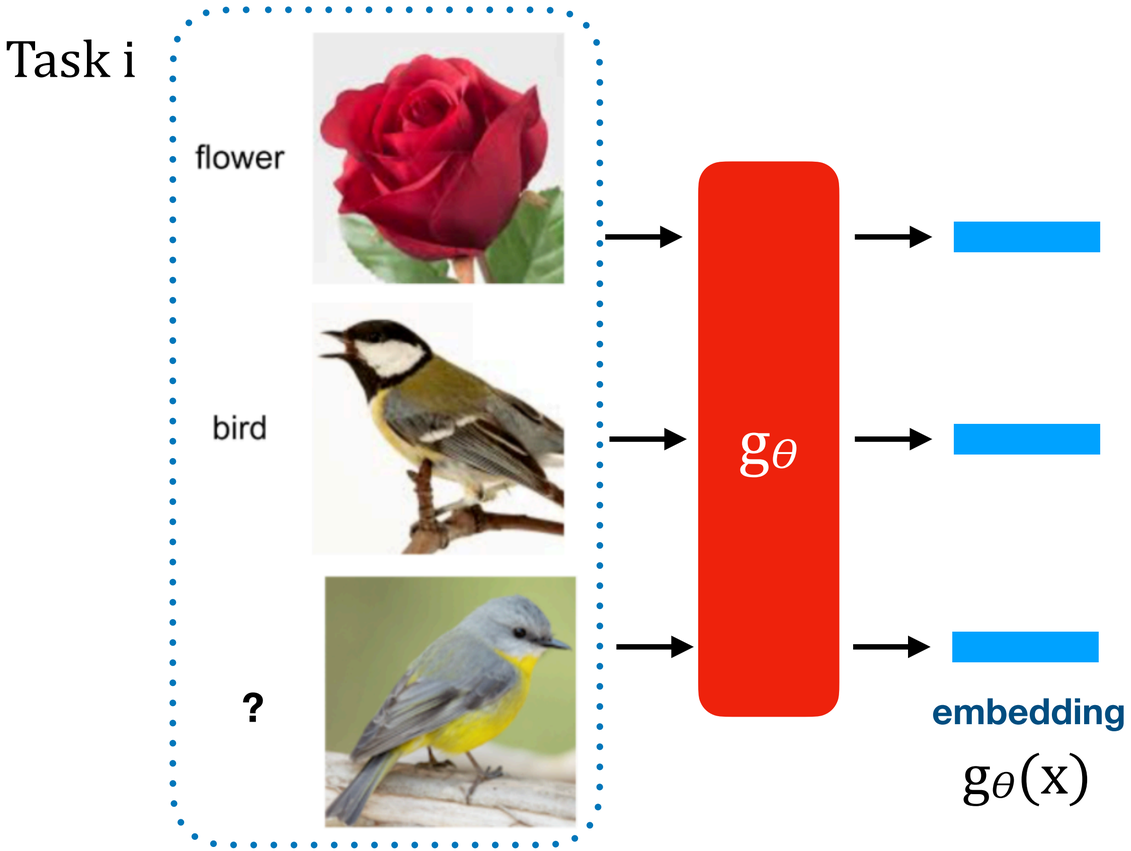
Figure source: Maheswaranathan et al. 2020

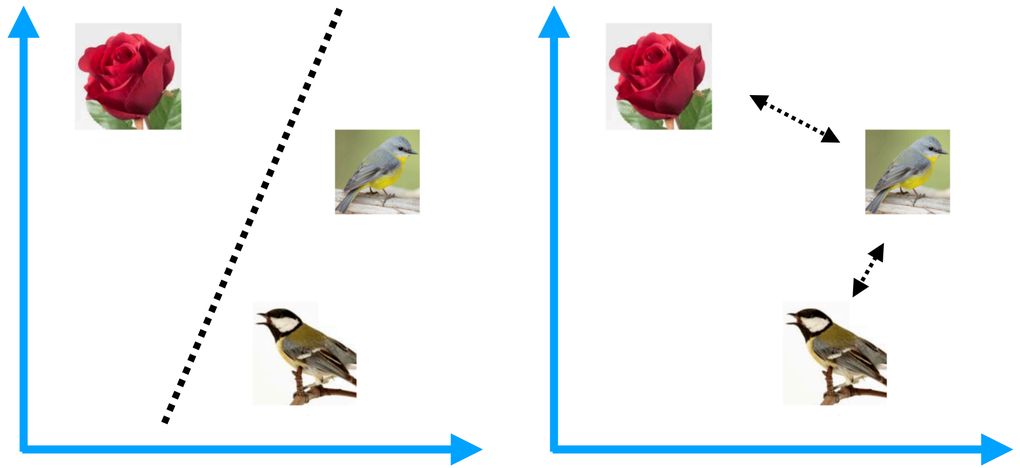
- RL-based optimizers: represent updates as a policy, learn using guided policy search [2]
- Combined with MAML:
  - learn per-parameter learning rates [3,4]
  - learn precondition matrix (to `warp' the loss surface) [5,6]
- Black-box optimizers: meta-learned with an RNN [7], or with user-defined priors [8]
- Speed up backpropagation by meta-learning sparsity and weight sharing [9]

[1] Maheswaranathan et al. 2020
[2] Ke and Malik 2016
[3] Li et al. 2017
[4] Antoniou et al. 2019
[5] Park et al. 2019
[6] Flennerhag et al. 2019
[7] Chen et al. 2017
[8] Souza et al. 2020
[9] Kirsch et al. 2020

Metric learning

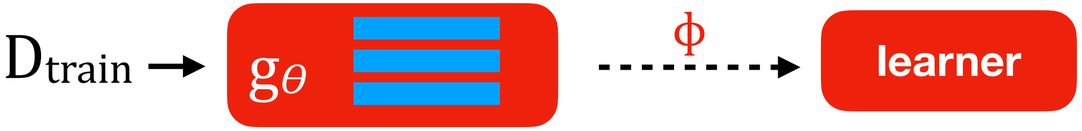
Learn an embedding network θ that transforms data $\{D_{train}, D_{test}\}$ across all tasks to a representation that allows easy similarity comparison





Can be seen as a simple black box meta-model

- Often non-parametric (independent of ϕ)



Matching networks

- Classifier: probability distribution based on similarity between x_{test} and $x_i \in D_{train}$

$$p(y \mid x_{test}, D_{train}) = \sum_{i=1}^{k} a(x_{test}, x_i)\ y_i$$

- Similarity: attention kernel based on cosine distance between embedded data points

$$a(x, x_i) = softmax(cos(f(x), g(x_i)))$$

- Learn two embeddings: $\theta \leftarrow \theta - \alpha \nabla_\theta \mathcal{L}(y, \hat{y})$
  - g_θ for D_{train} samples
  - $f_{\theta'}$ for D_{test} samples
- Two options:
  - Simple embedding: $\theta = \theta'$, so $g_\theta = f_{\theta'}$
  - Contextual embedding: g is a bidirectional LSTM and f is an attention LSTM

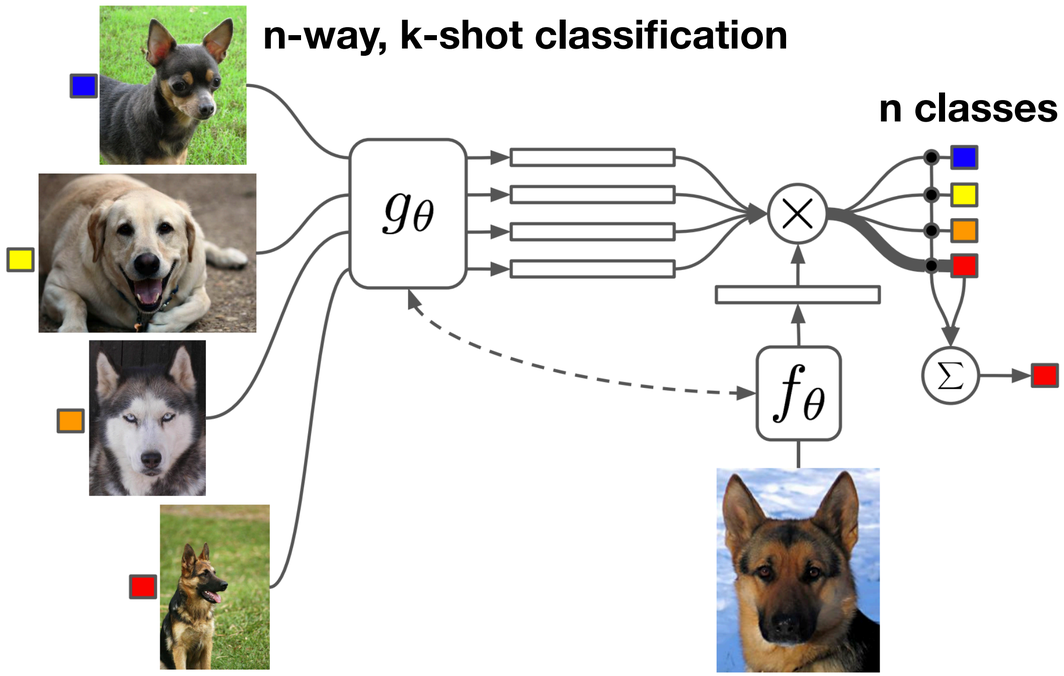


Figure source: Vinyals et al. 2016

Prototypical networks

- Use an embedding function f_θ to encode each data point
- Define a prototype v_c for every class c based on the examples of that class $D_{train}^{(y)}$

$$\mathbf{v}_c = \frac{1}{|D_{train}^{(y)}|} \sum_{(x_i,y_i) \in D_{train}^{(y)}} f_\theta(x_i)$$

- Class distribution for input x is based on inverse distance between x and prototypes

$$p(y = c \,|\, x_{test}) = softmax(-d_\phi(f_\theta(x), v_c))$$

- Distance function can be any differentiable distance
  - E.g. squared Euclidean
- Loss function to learn the embedding:

$$\mathcal{L}(\theta) = -\, log\; p_\theta(y = c \,|\, x)$$

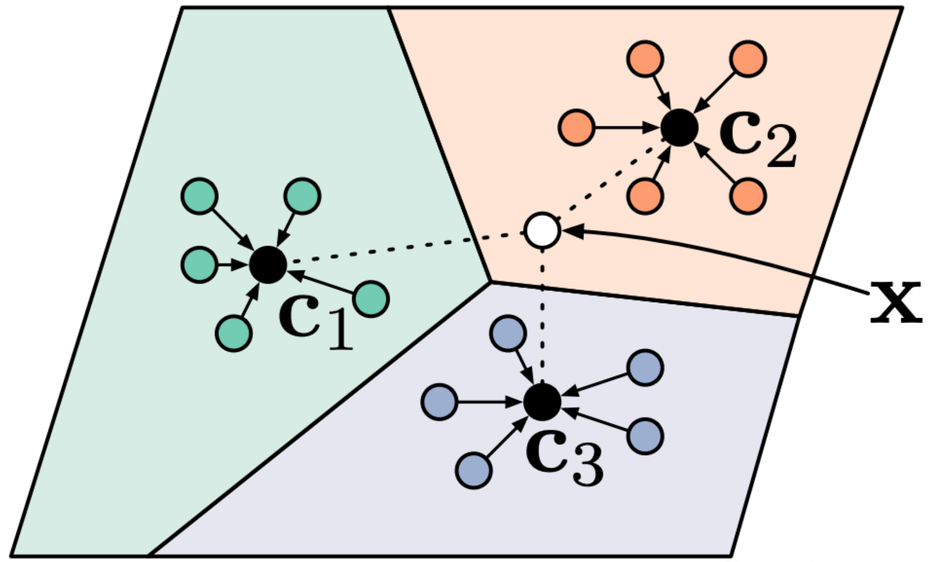


Figure source: Snell et al. 2017

Sung et al. 2018

Relation networks

- Similar to matching networks, but with a trainable similarity metric
  - Learns a non-linear relationship between the data points
- Learn an embedding network f_θ and a relation network $g_{\theta'}$
- The relationship/similarity between a pair of inputs:

$$r_{ij} = g_{\theta'}(concat(f_\theta(x_i), f_\theta(x_j)))$$

- Predictions based on the examples most related to x_{test}
- More expressive power, often beats other metric learners

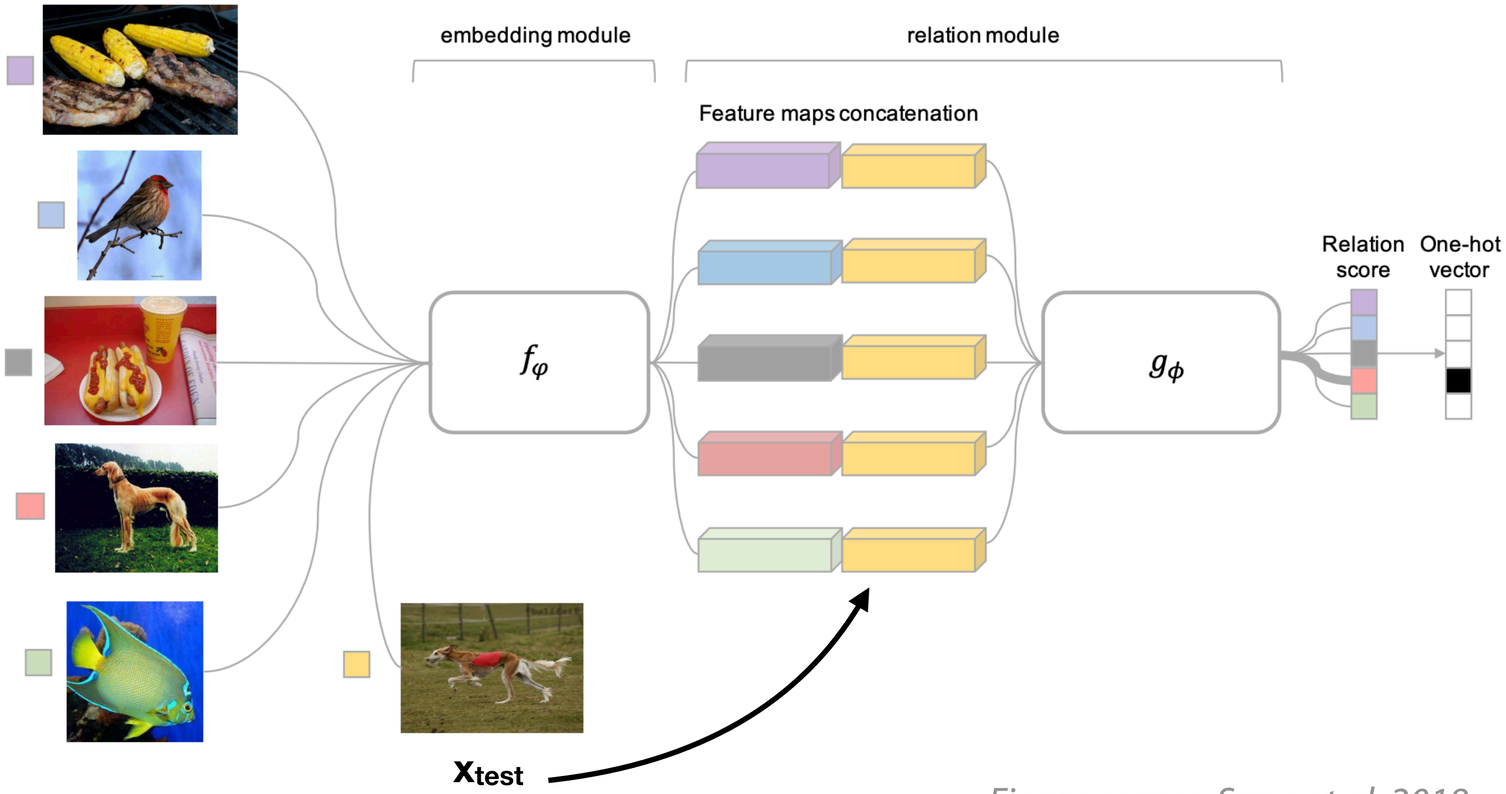


Figure source: Sung et al. 2018

Metric learning

- Quite a few other techniques exist
  - Siamese neural networks [1]
  - Graph Neural Networks [2]
    - Also applicable for semi-supervised and active learning
  - Attentive Recurrent Comparators [3]
    - Compares inputs not as a whole but by parts (e.g. image patches)
  - MetaOptNet [4]
    - Learns embeddings so that linear models can distinguish between classes
- Overall:
  - Fast at test time, although pair-wise comparisons limit task size
  - Mostly limited to few-shot supervised tasks
  - Fails when test tasks are more distant: no way to adapt

Koch et al. 2015

Garcia et al. 2017

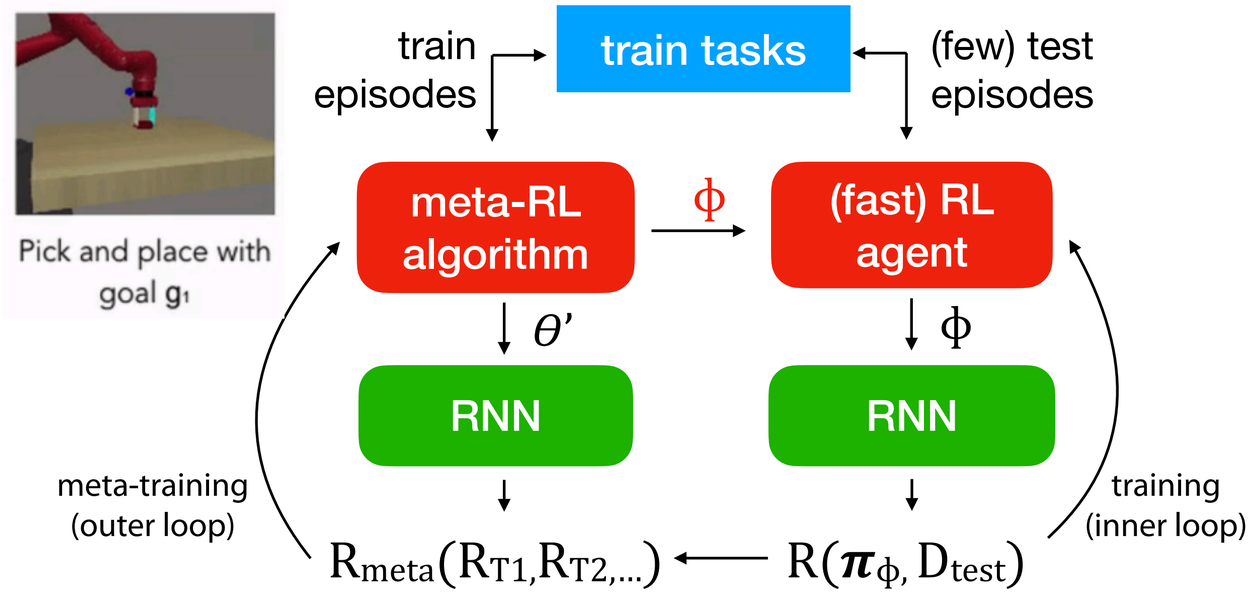
Shyam et al. 2017

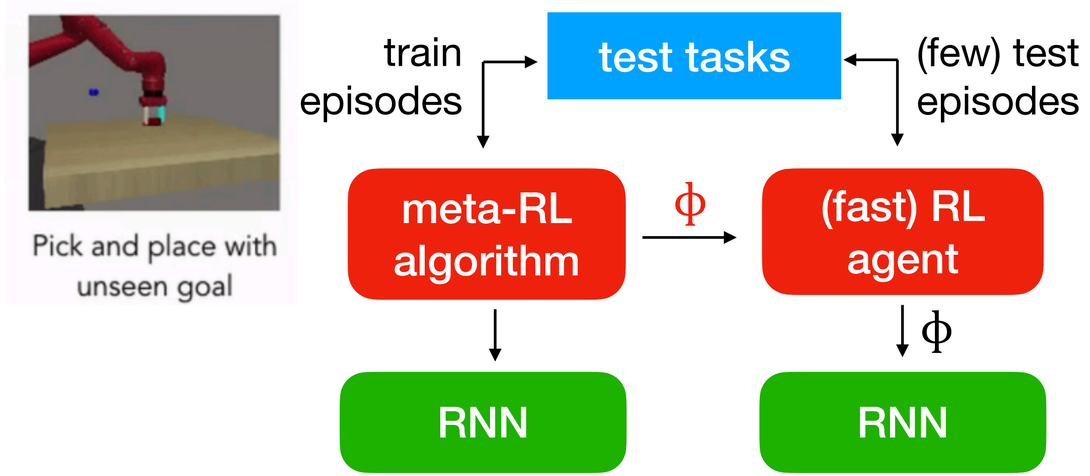
Lee et al. 2019

Black-box model for meta-RL

Wang et al. 2016

Duan et al. 2018





- RNNs serve as dynamic task embedding storage
- Maximize expected reward in each trial
- Very expressive, perform very well on short tasks
- Longer horizons are an open challenge

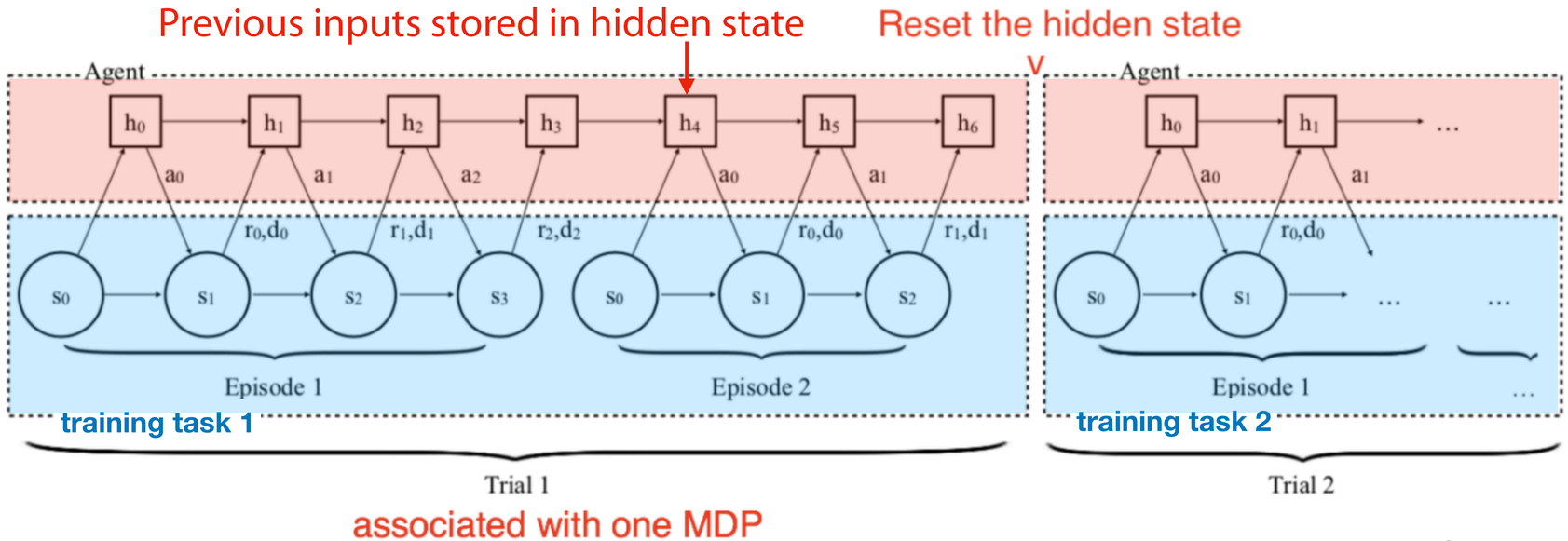
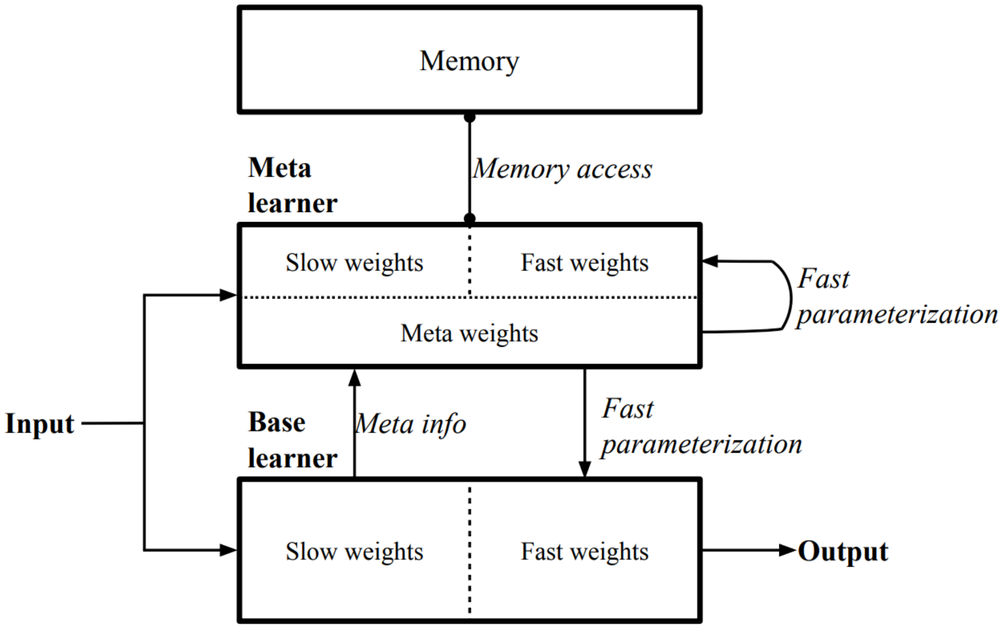


Image source: Duan et al. 2018

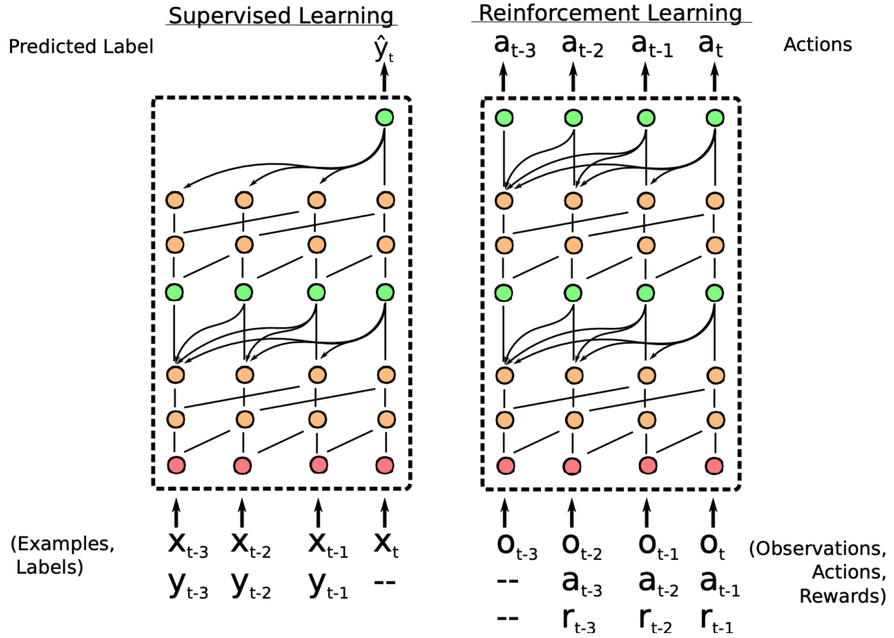
Other black-box models

Santoro et al. 2016
Munkhdalai et al. 2017
Mishra et al. 2018

- Memory-augmented NNs [1]
  - Uses neural Turing machines: short term + long term memory
- Meta Networks [2]
  - Meta-learner that returns 'fast weights' for itself and the base network solving the task
- Simple Neural attentive meta- learner (SNAIL)
  - Aims to overcome memory limitations of RNNs with series of 1D convolutions



Meta Networks. *Image source: Munkhdalai et al. 2017*



SNAIL. *Image source: Mishra et al. 2018*

Training Task Acquisition

Wang et al. 2019

Wang et al. 2020

- Ultimately, meta-learning translates constraints on the learner to constraints on the data
  - The biases we don't put in manually have to be learnable from data
- Can we automatically create new tasks to *inform and challenge* our meta-learners?
- Paired open-ended trailblazer (POET): evolves a parameterized environment Θ_E for agent Θ_A
  - Select agents that can solve challenges AND evolve environments so they are solvable

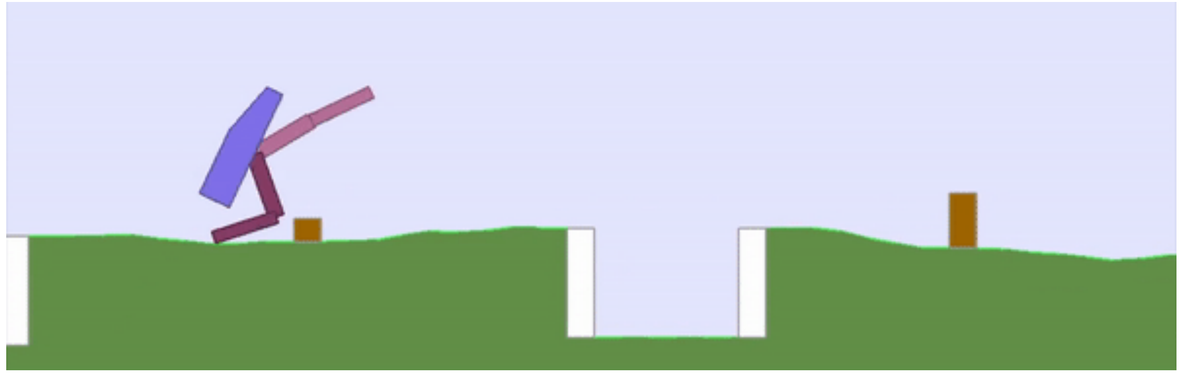
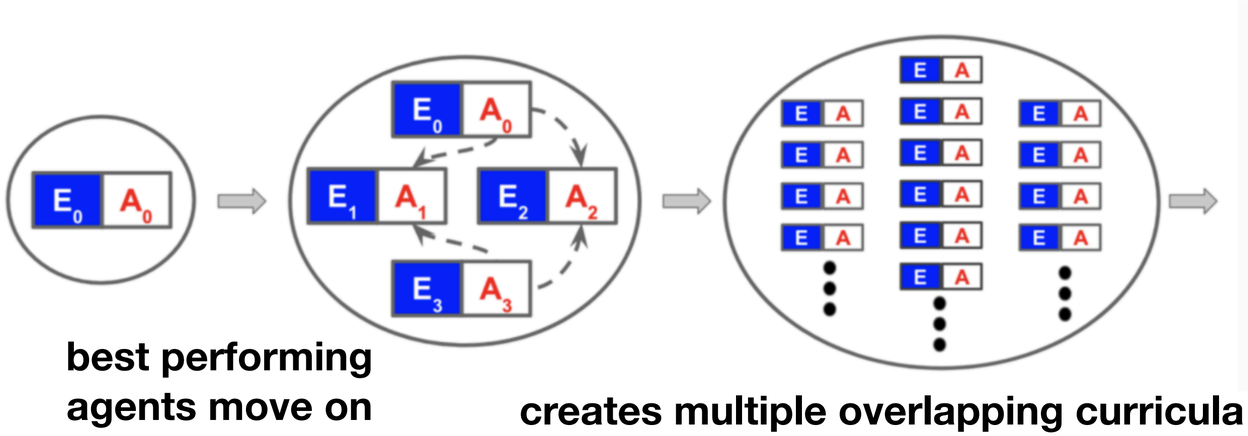


Figure source: Wang et al. 2019



Training Task Acquisition

Wang et al. 2019

Wang et al. 2020

- Ultimately, meta-learning translates constraints on the learner to constraints on the data
  - The biases we don't put in manually have to be learnable from data
- Can we automatically create new tasks to *inform and challenge* our meta-learners?
- Paired open-ended trailblazer (POET): evolves a parameterized environment Θ_E for agent Θ_A
  - Select agents that can solve challenges AND evolve environments so they are solvable

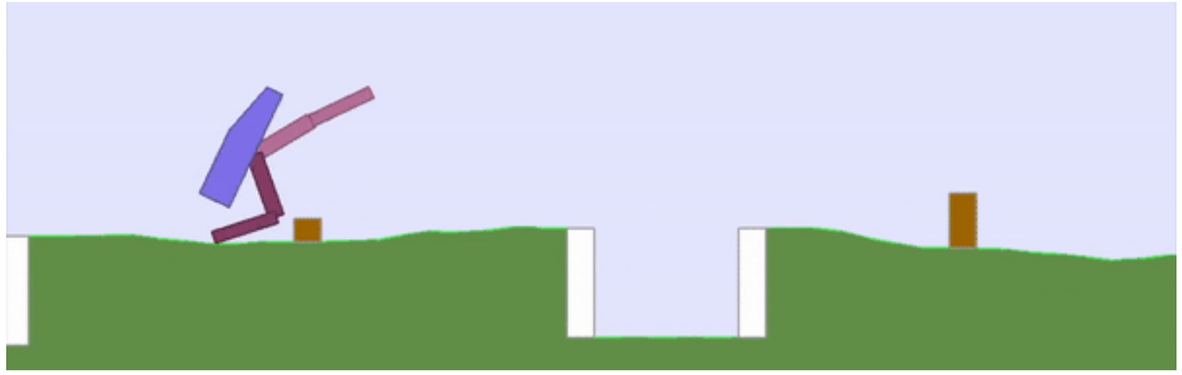
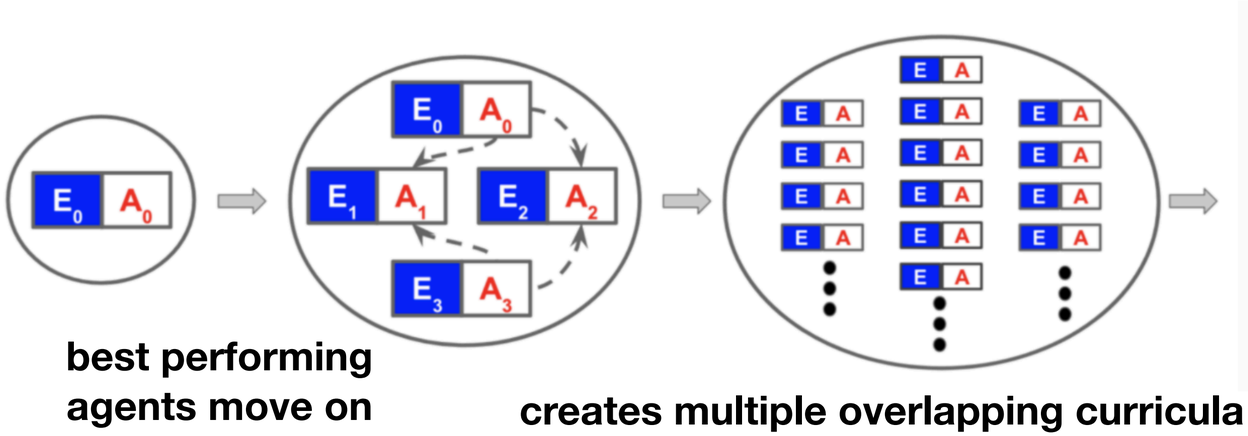


Figure source: Wang et al. 2019



Does POET scale? Increasingly difficult 3D terrain, 18 degrees of freedom.

Zhou & Vanschoren. 2021

Does POET scale? Increasingly difficult 3D terrain, 18 degrees of freedom.

Zhou & Vanschoren. 2021

Thank you

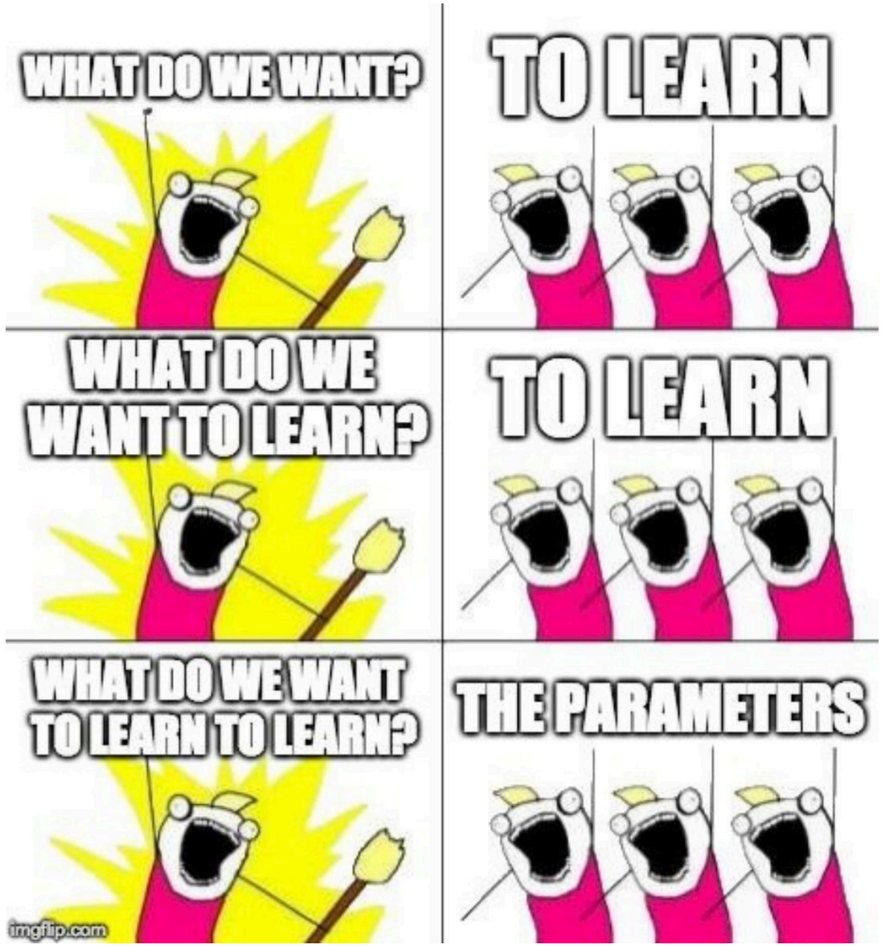


Image source: Pesah et al. 2018