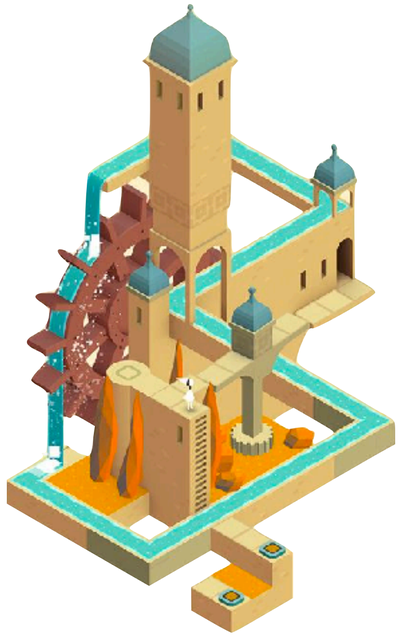


Meta-Learning

DSAA 2021 tutorial

Joaquin Vanschoren (TU Eindhoven)

Cover art: MC Escher, Ustwo Games

Intro: humans can easily learn from a single example

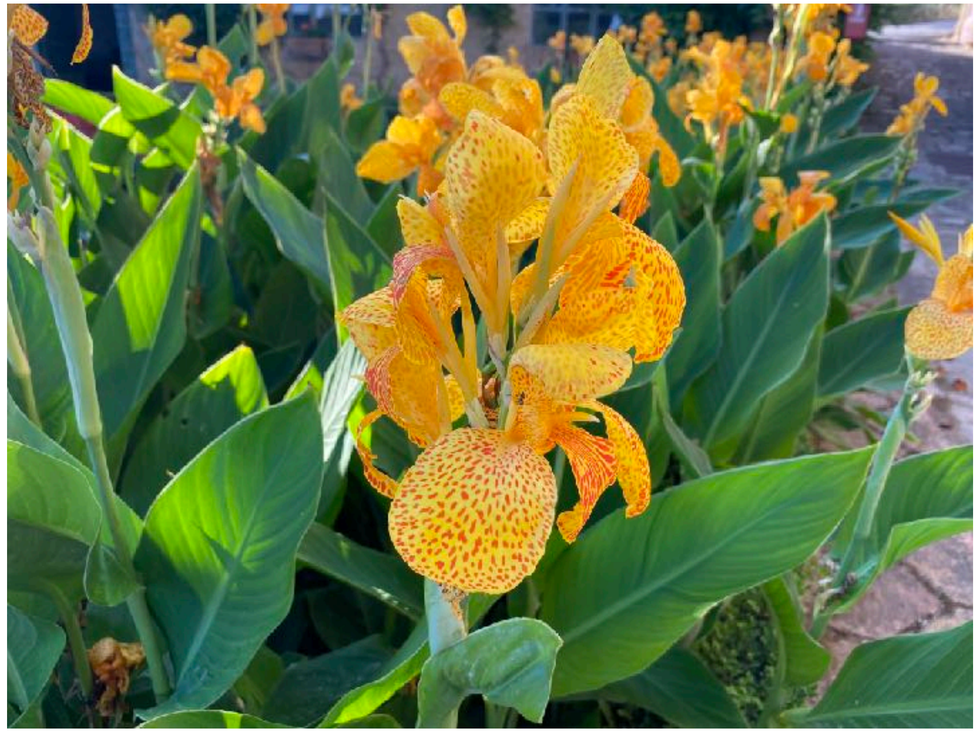
thanks to years of learning (and eons of evolution)

Canna Indica ‘Picasso’

?

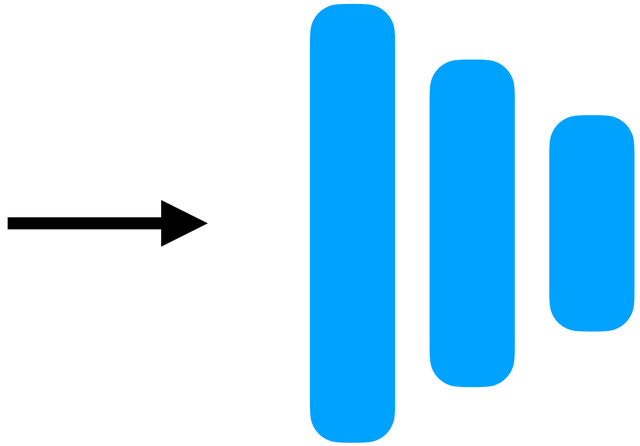
train

later



test

Can a computer learn from a single example?



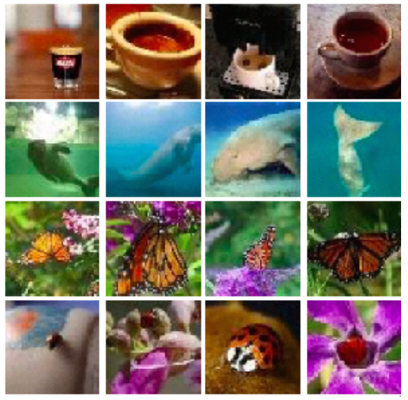
Canna
Indica
`Picasso'

e.g. ResNet50
(23 million trainable parameters)

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

Transfer learning?

Source task



ImageNet
(14 million images)

Pretrain

Target task

Finetune

e.g. ResNet50
(23 million trainable parameters)

Canna
Indica
`Picasso'

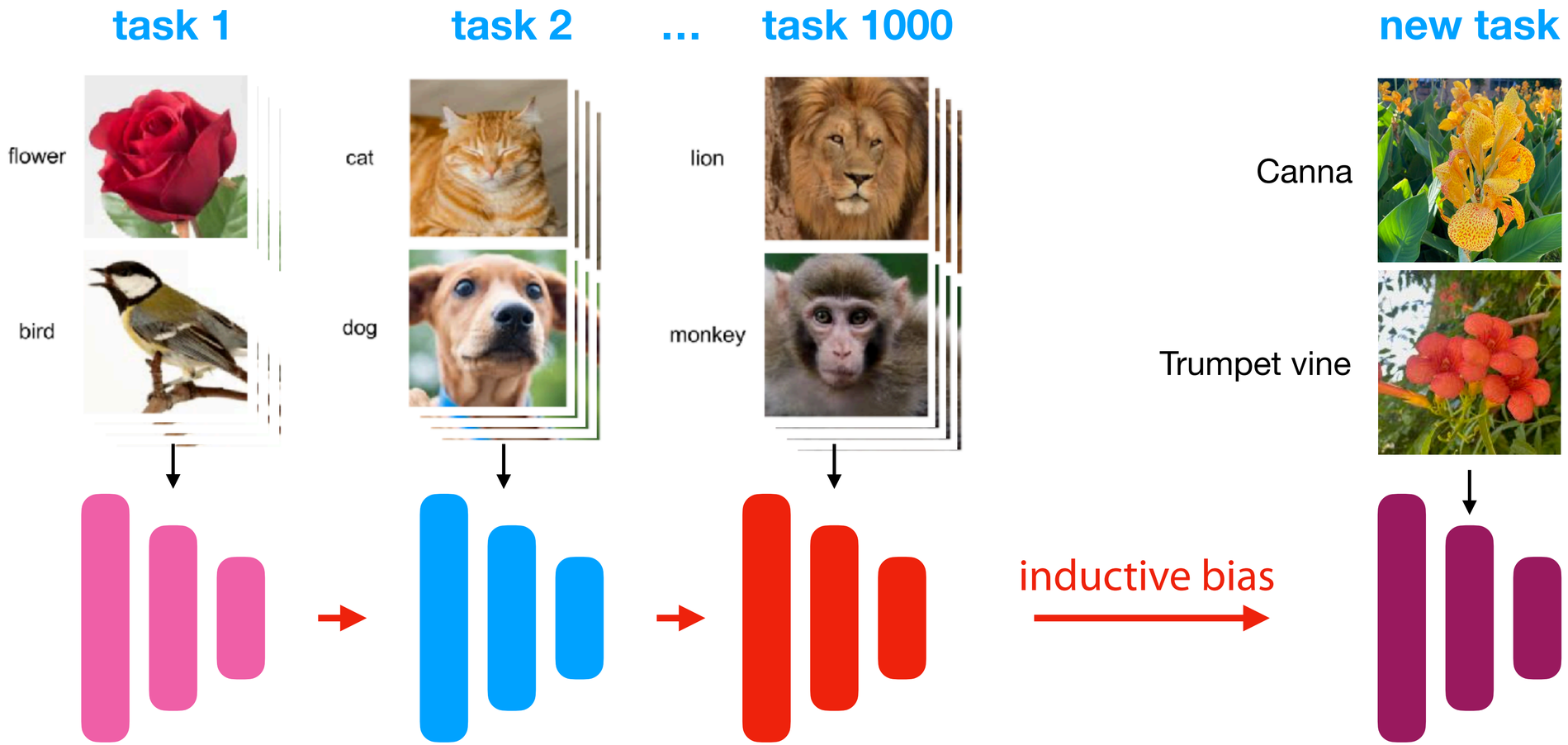
A single source task (e.g. ImageNet) may not generalize well to the test task

Meta-learning

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

***Inductive bias**: learn **assumptions** that you can to transfer 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

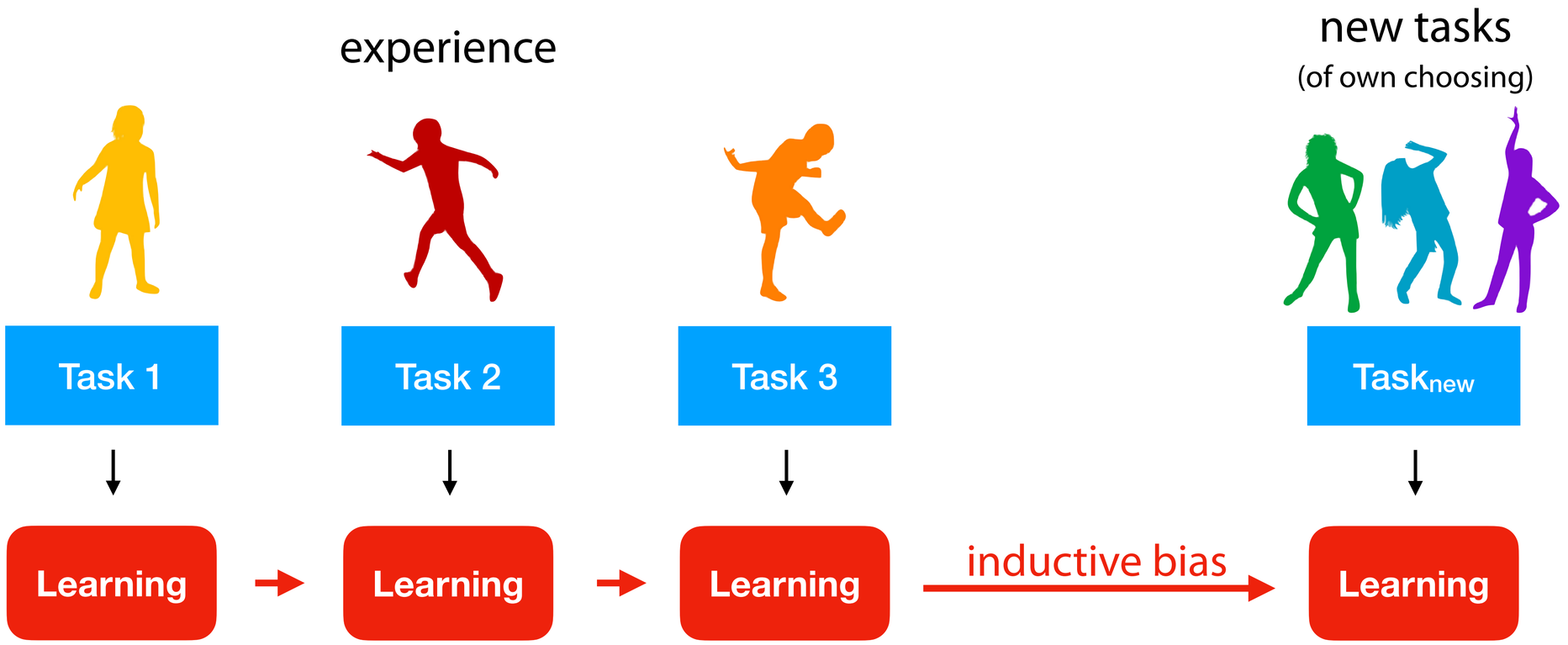
Lake et al. 2017

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?)



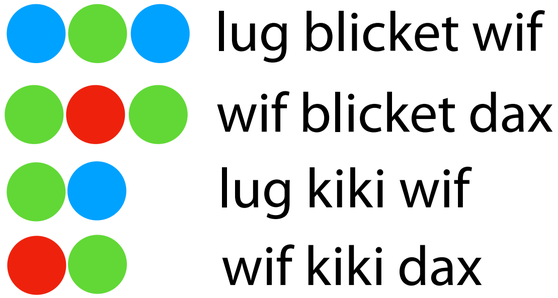
key aspects of fast learning: compositionality, causality, learning to learn

Lake et al. 2017

Inductive bias (in language)

which assumptions do we make?

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?

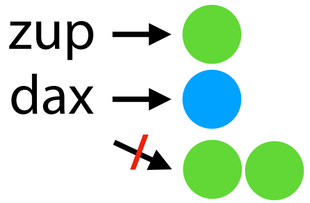
Lake et al. 2017
Lake et al. 2019

we can still solve problems by making assumptions

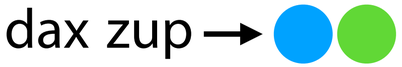
Item pool

Test

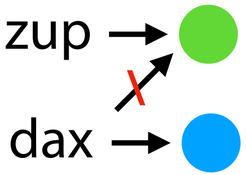
Commonly used assumptions:



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

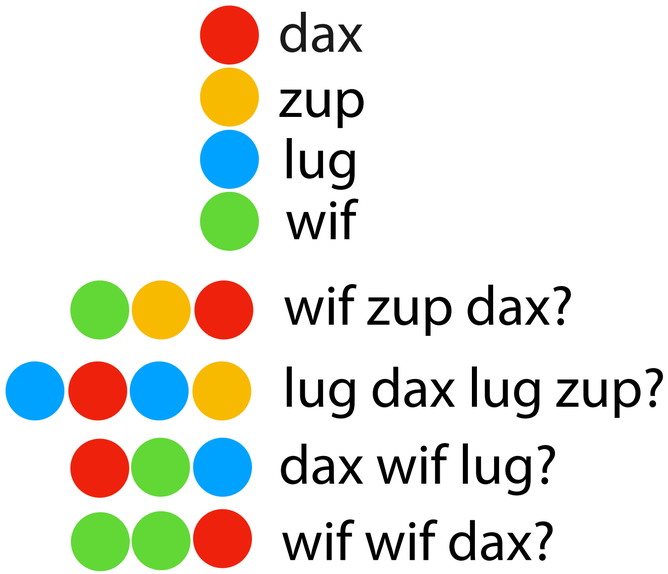
Humans assume that words have consistent meanings and follow input/output constraints

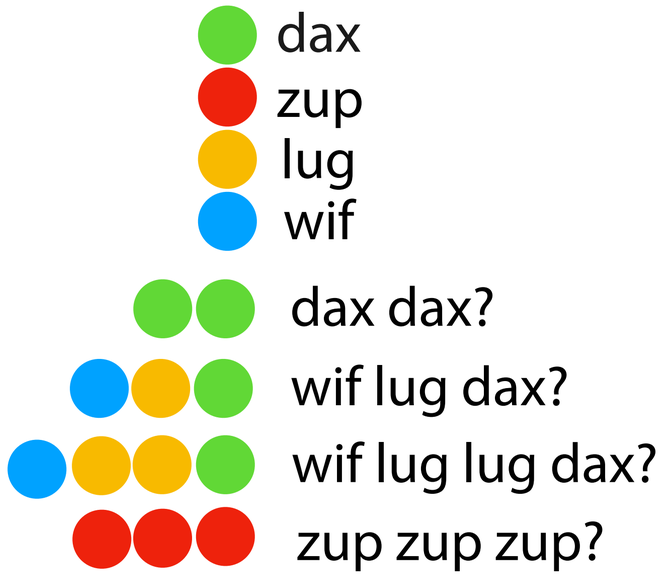
These assumptions (inductive biases) are necessary for learning quickly

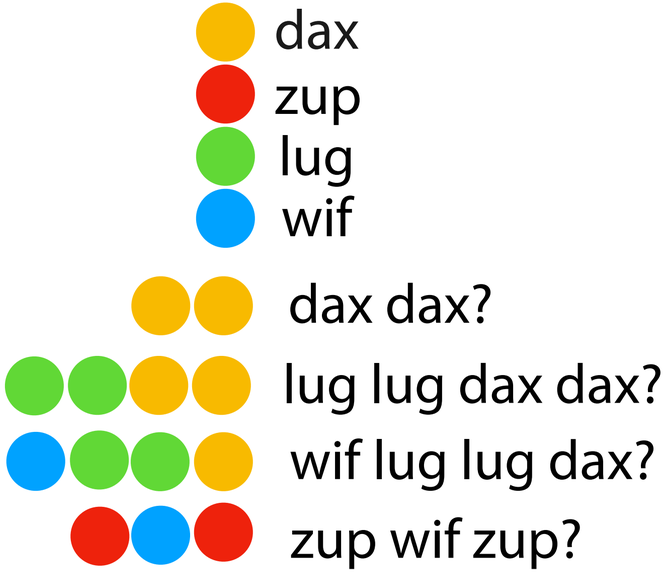
Meta-learning inductive biases

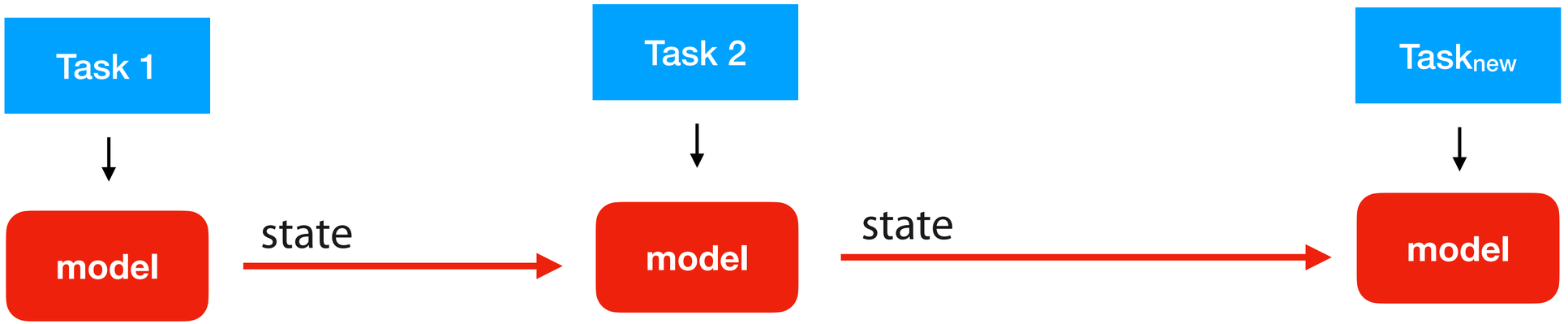
Lake et al. 2017
Lake et al. 2019

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



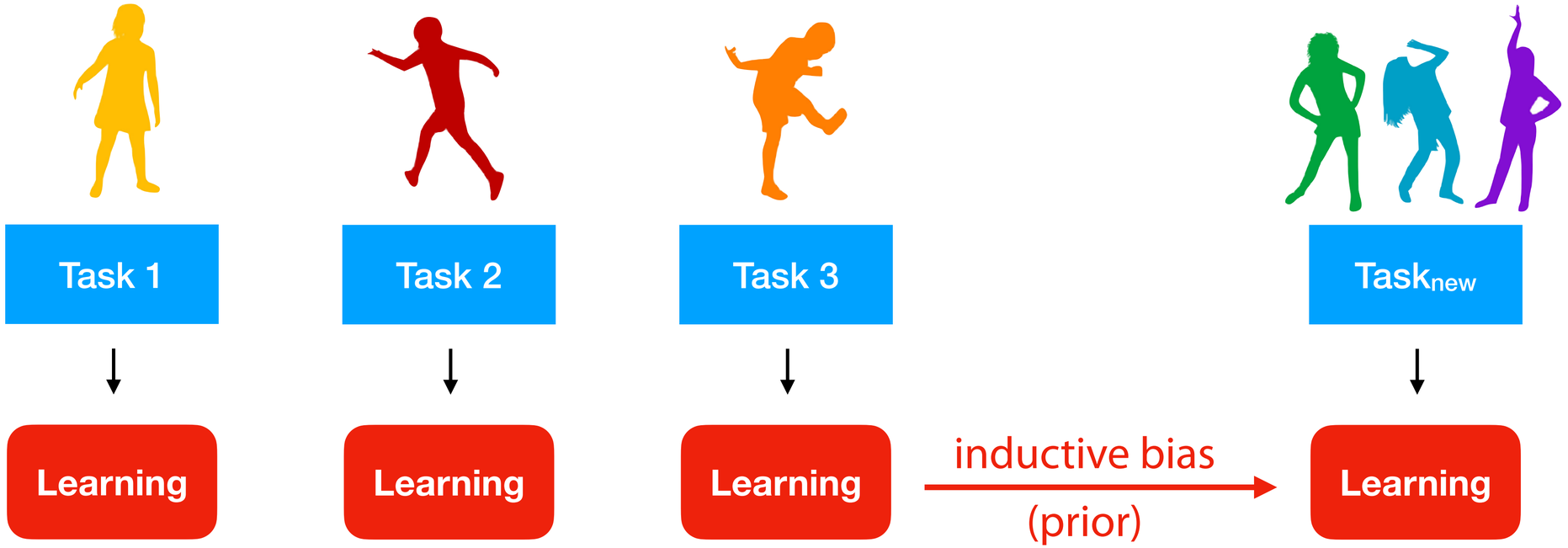






Meta-learning goal

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



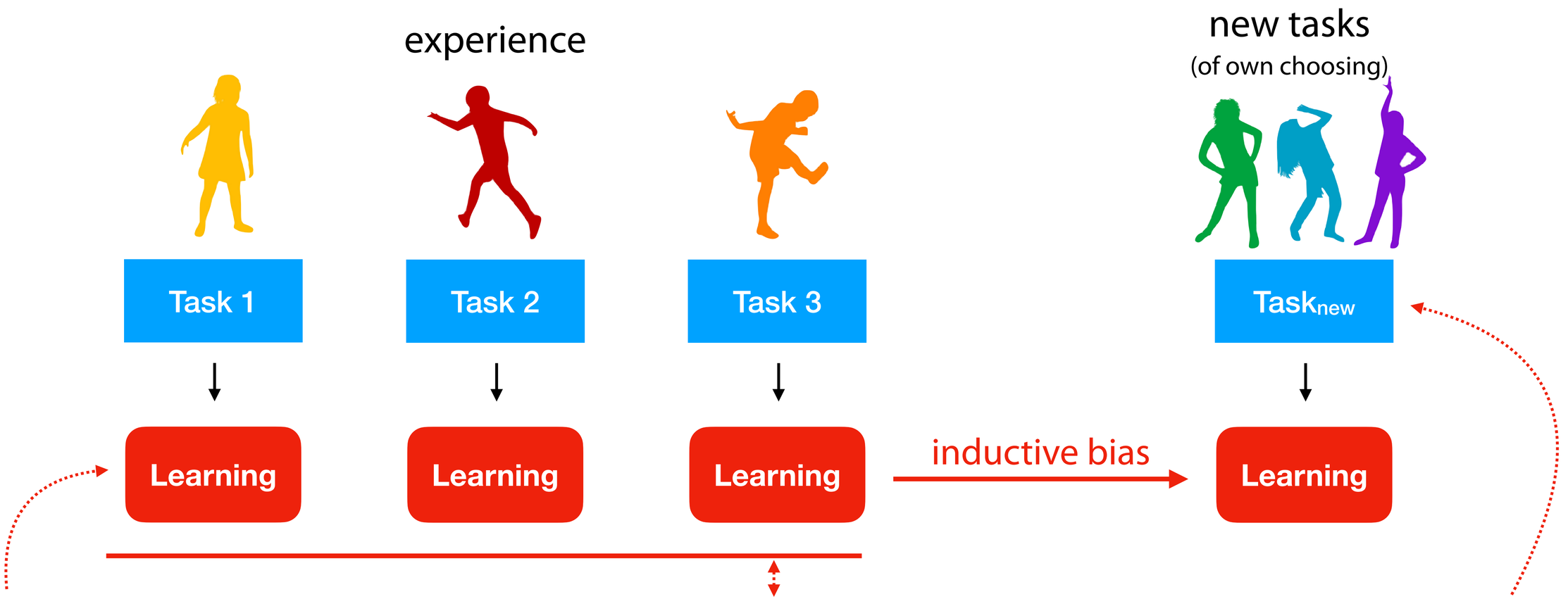
Inductive bias: any assumptions added to training data to learn more effectively. E.g:

- Instead of general model architectures, learn better architectures (and hyperparameters)
- Instead of starting from random weights, learn good initial weights
- Instead of standard loss/reward function, learn a better loss/reward function

Clune 2019

What can we learn to learn?

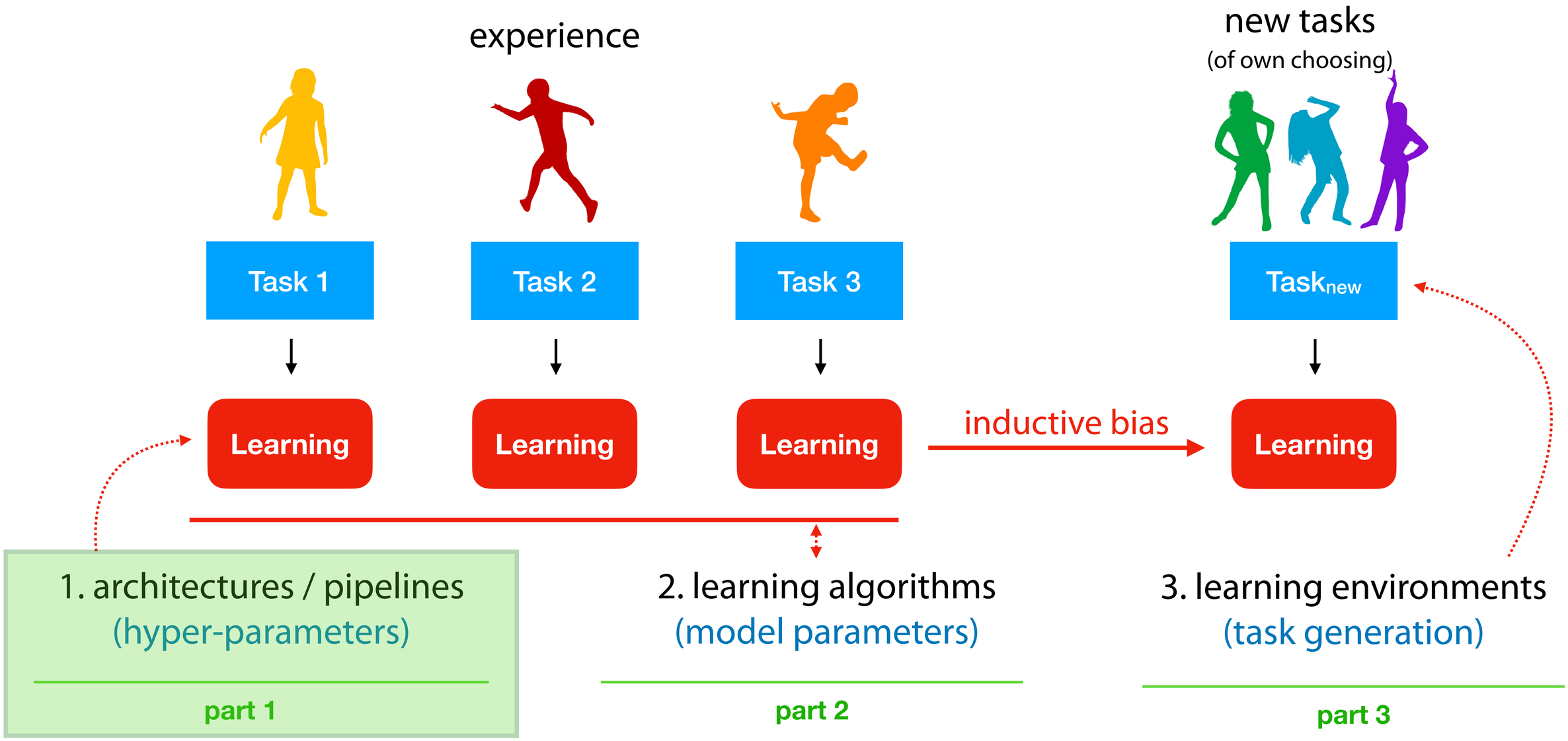
3 *pillars*



Clune 2019

What can we learn to learn?

3 pillars



Terminology

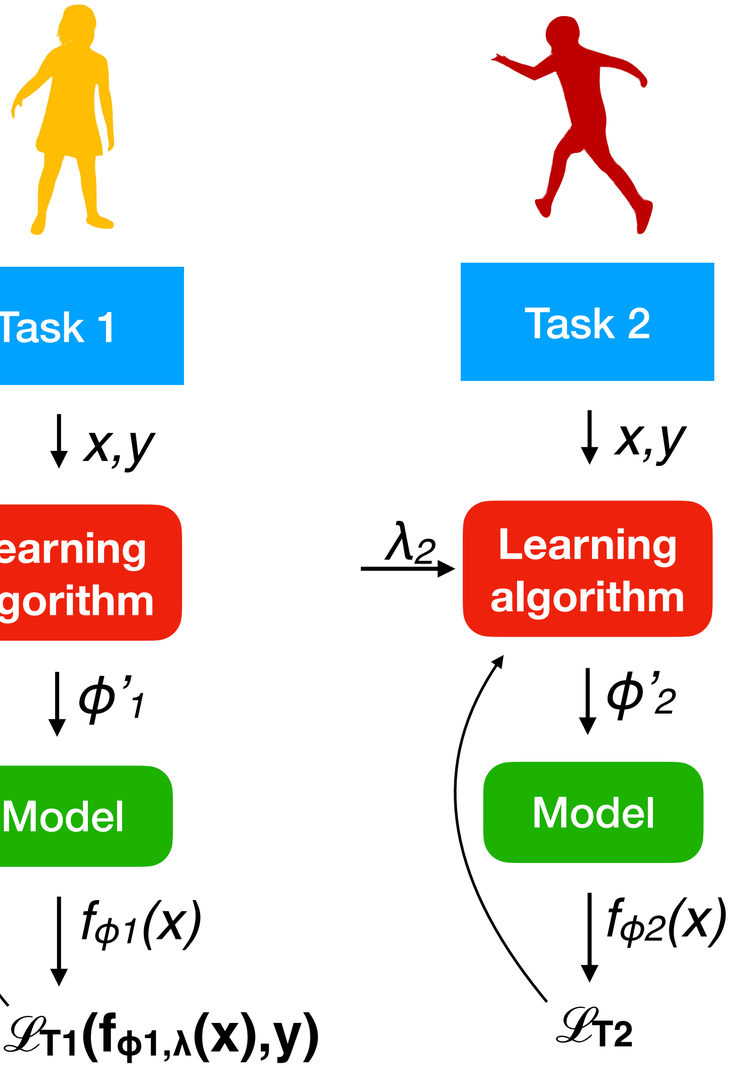
Task: *distribution of samples q(x)*
outputs y, loss $\mathscr{L}(x,y)$

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

$\phi^* = \text{argmax}_\phi \log p(\phi \mid T)$

$= \text{argmin}_\phi \mathscr{L}(f_{\phi,\lambda}(x),y)$

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



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

meta-parameters θ

(encode some inductive bias)

Hutter et al. 2019

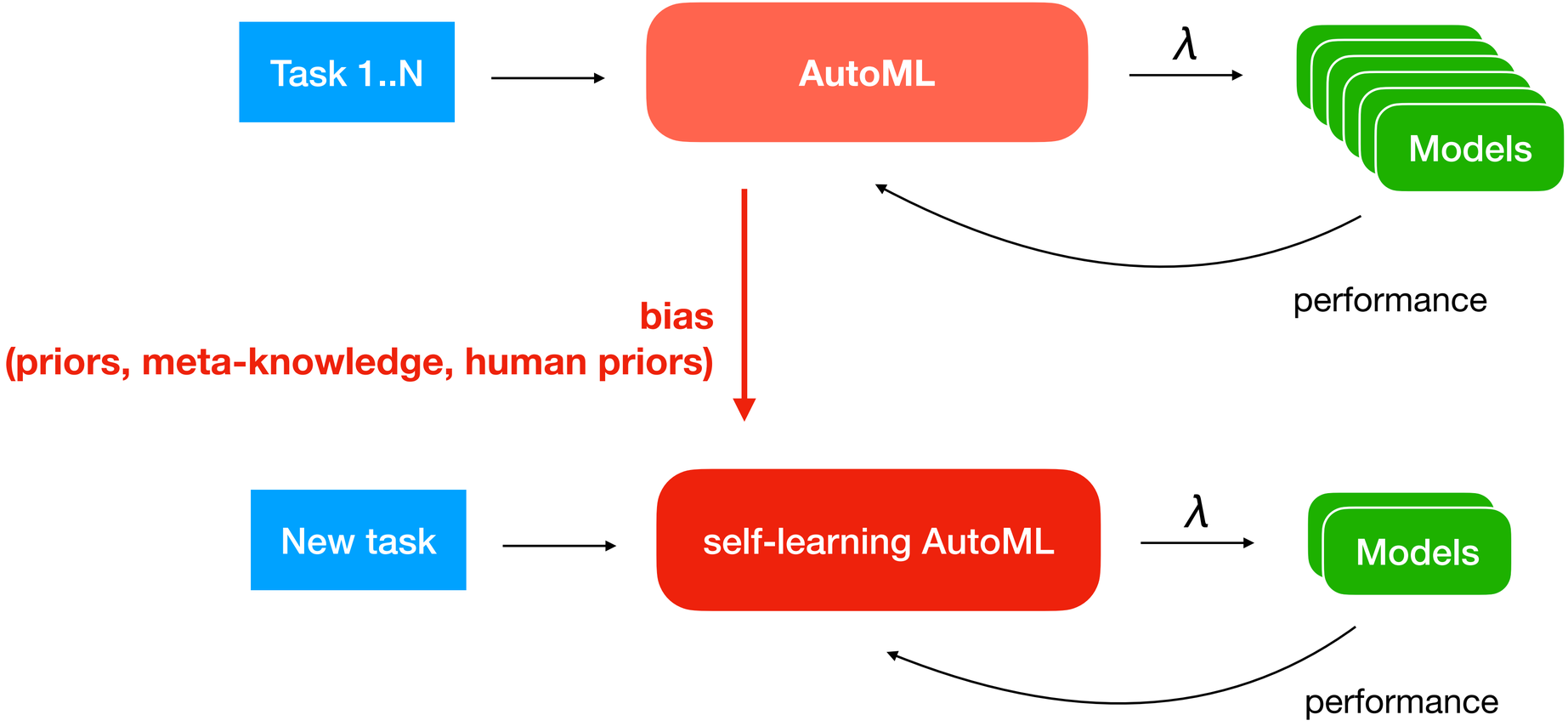
Learning hyperparameters

Closely related to Automated Machine Learning (AutoML)

But: meta-learn how to design architectures/pipelines and tune hyperparameters

Human data scientists also learn from experience

Search space can be huge!

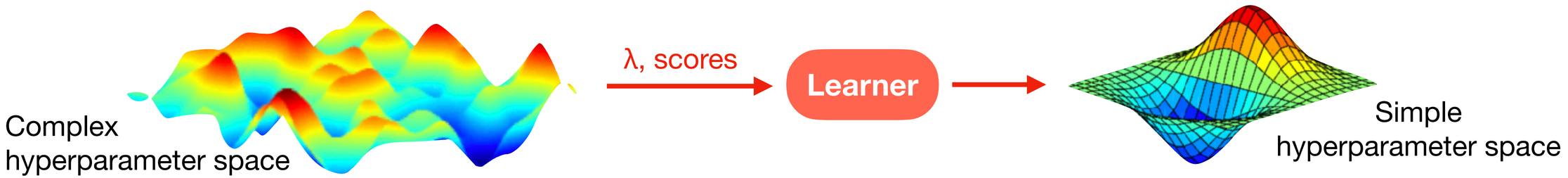


Vanschoren 2018

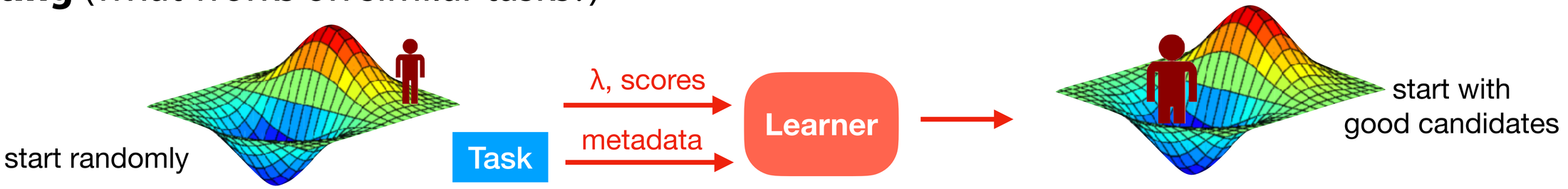
Meta-learning for AutoML: how?

hyperparameters = architecture + hyperparameters

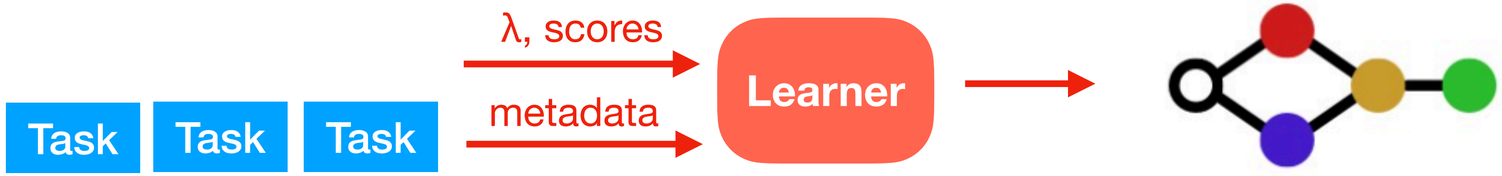
Learning hyperparameter priors



Warm starting (what works on similar tasks?)



Meta-models (learn how to build models/components)

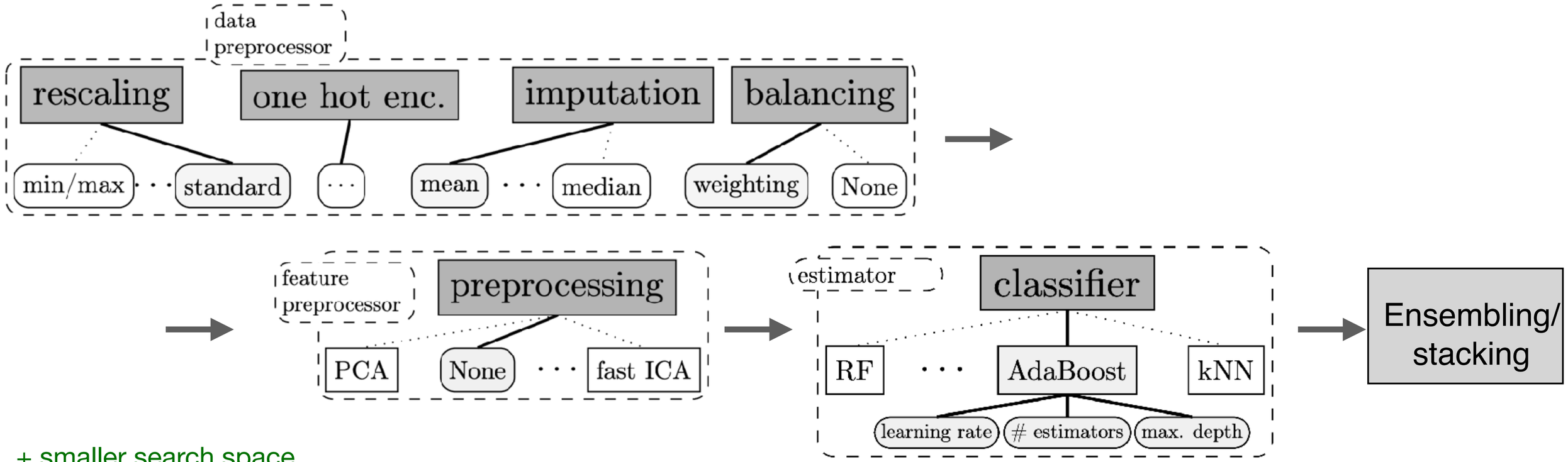


Observation:
current AutoML strongly depends on learned priors

Manual architecture priors

autosklearn Feurer et al. 2015
autoWEKA Thornton et al. 2013
hyperopt-sklearn Komer et al. 2014
AutoGluon-Tabular Erickson et al. 2020

- Most successful pipelines have a similar structure
- Can we meta-learn a prior over successful structures?



+ smaller search space
- you can't learn entirely new architectures

Figure source: Feurer et al. 2015

Manual architecture priors

Successful deep networks often have repeated motifs (cells)

e.g. Inception v4:

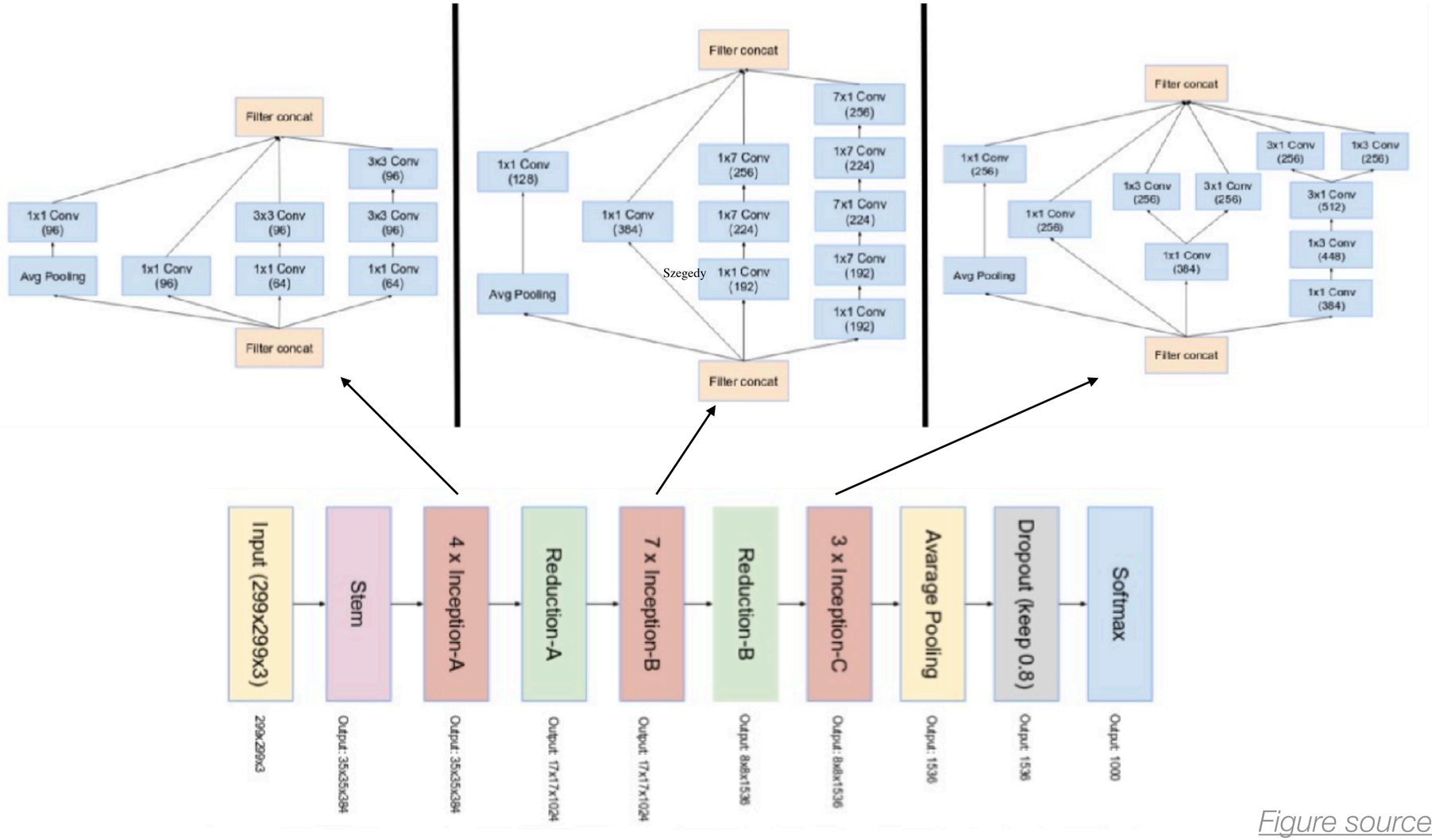


Figure source: Szegedy et al 2016

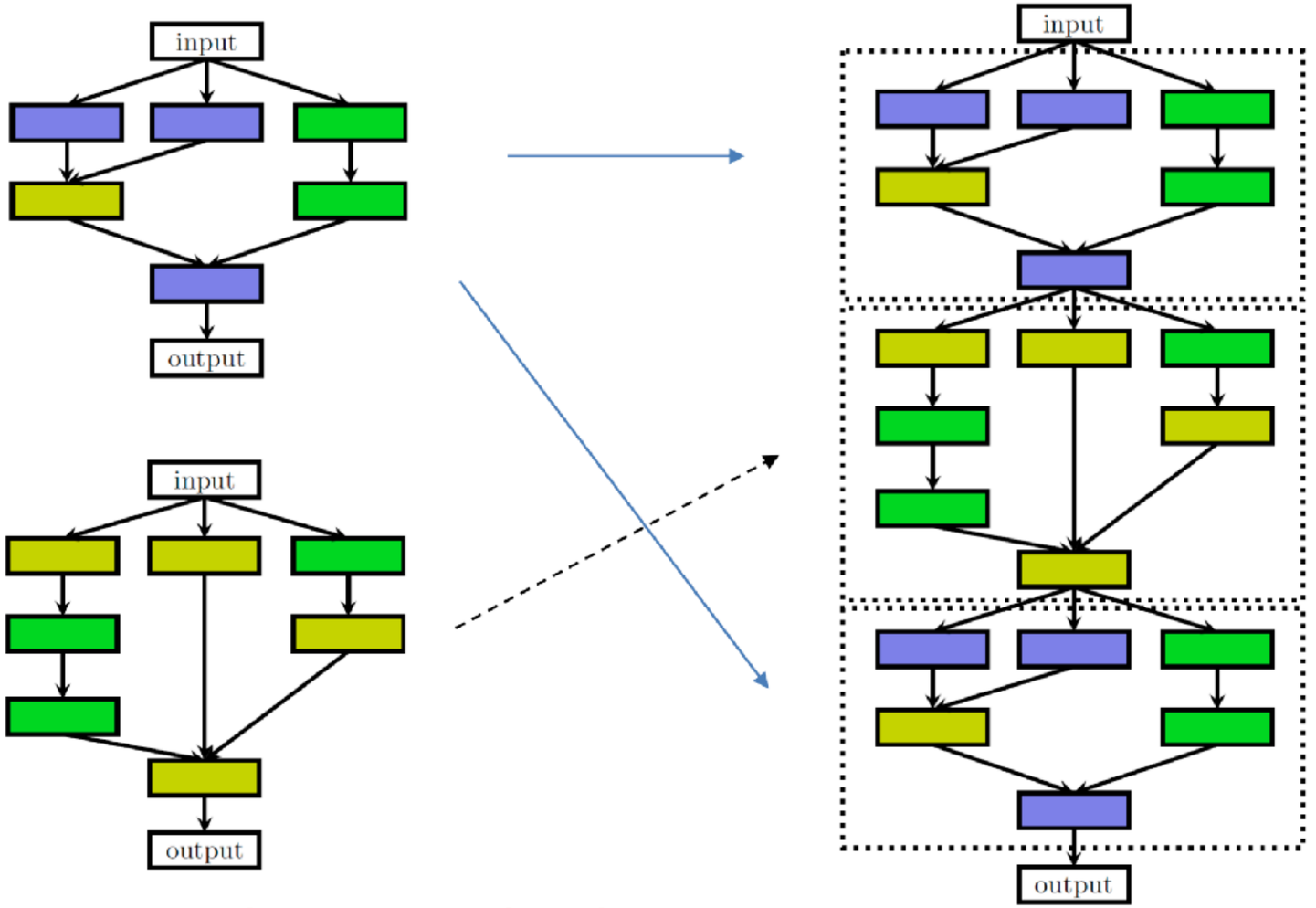
Google NASNet Zoph et al 2018

Cell search space prior

Compositionality: learn hierarchical building blocks to simplify the task

Cell search space

- learn parameterized building blocks *(cells)*
- stack cells together in macro-architecture



+ smaller search space
+ cells can be learned on a small dataset & transferred to a larger dataset
- strong domain priors, doesn't generalize well

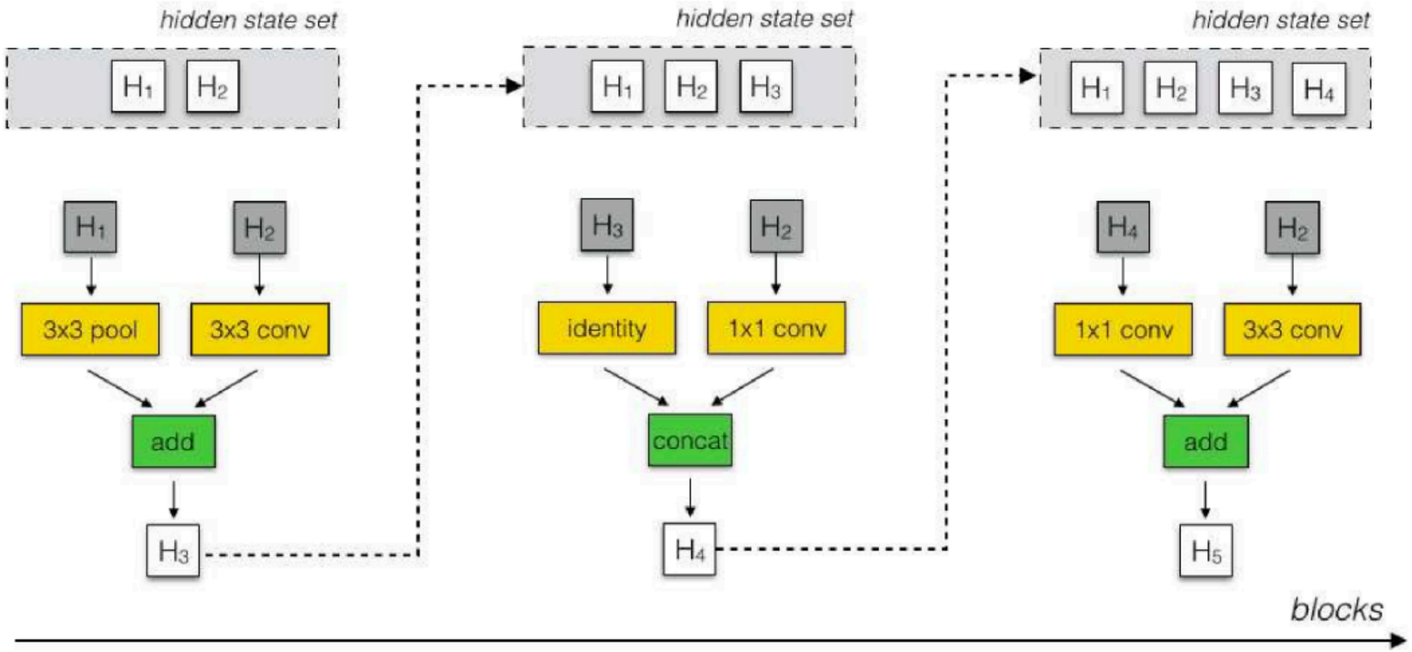
Can we meta-learn hierarchies / components that generalize better?

Figure source: Elsken et al., 2019

NASNet, Zoph et al 2018

Cell search space prior

- Cell construction with RL-based search (SOTA ImageNet, 450 GPUs, 3-4 days):
  - Select existing layers (hidden states, e.g. cell input) H_i to build on
  - Add operation (e.g. 3x3conv) on H_i
  - Combine into new hidden state (e.g. concat, add,…)
  - Iterate over *B* blocks



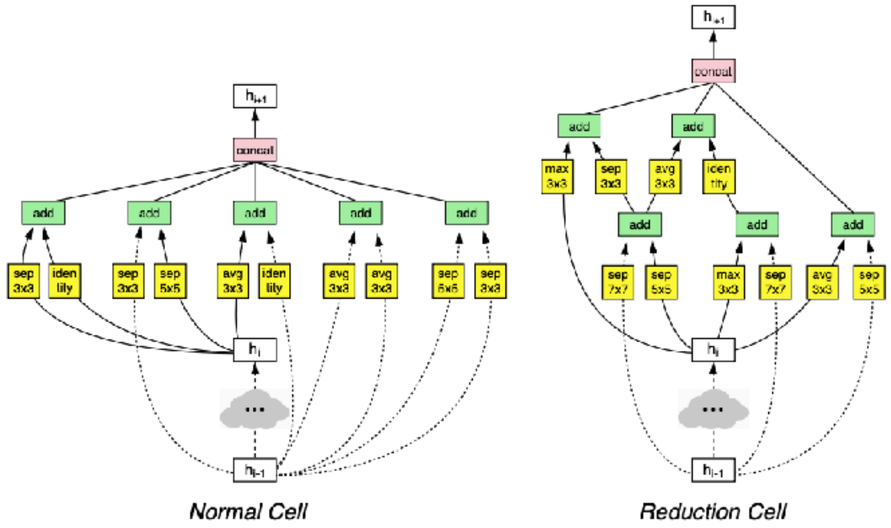
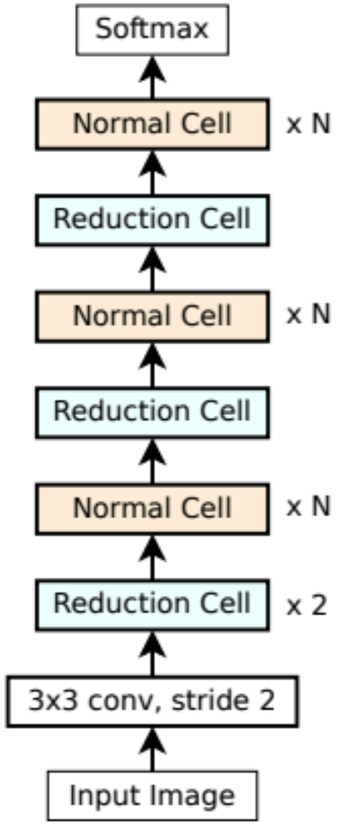


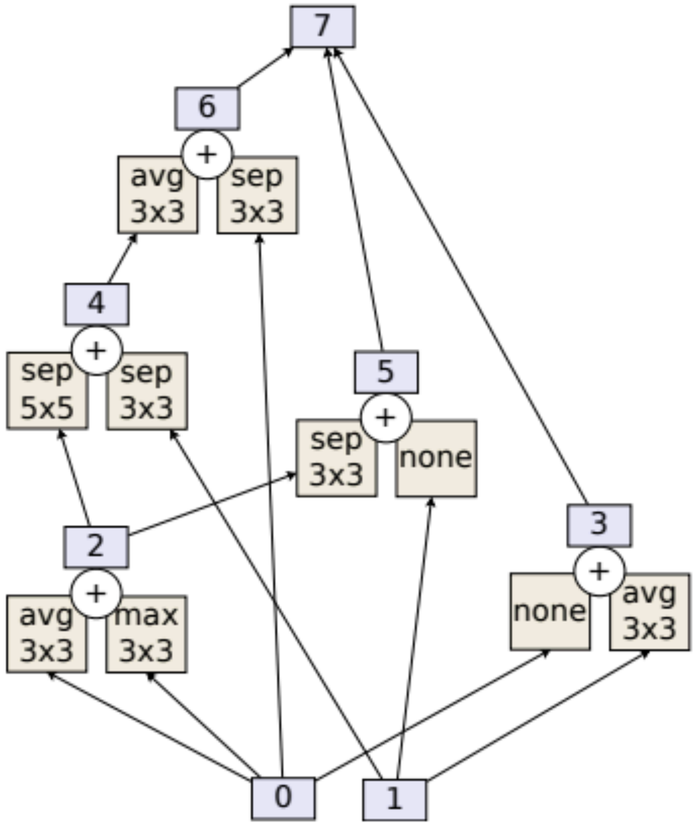
Figure source: Zoph et al., 2018

Cell search space prior

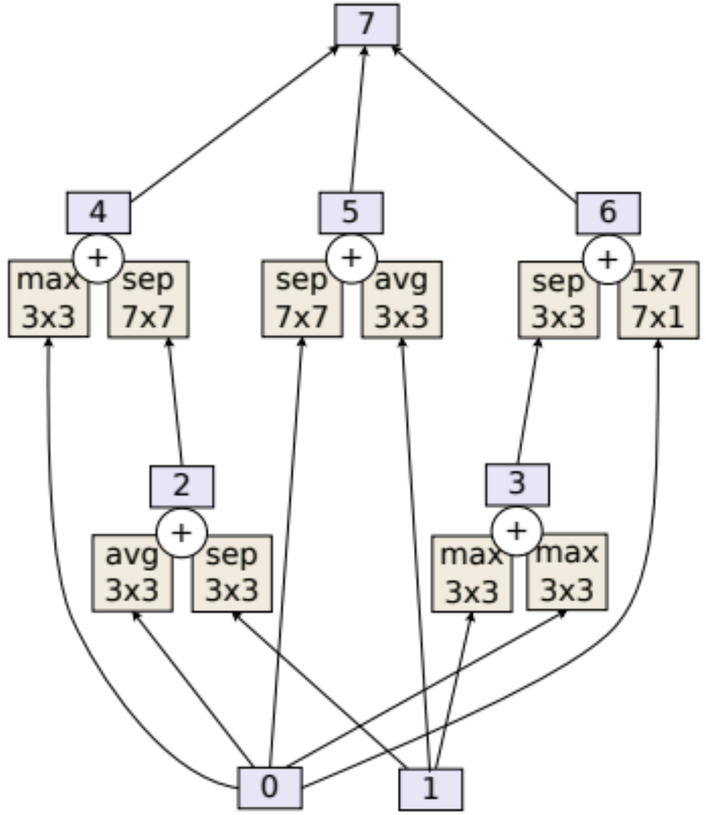
AmoebaNet, Real et al 2019

- Cell construction with neuro-evolution (SOTA ImageNet)





normal cell



reduction cell

Figure source: Real et al., 2019

Cell search space prior

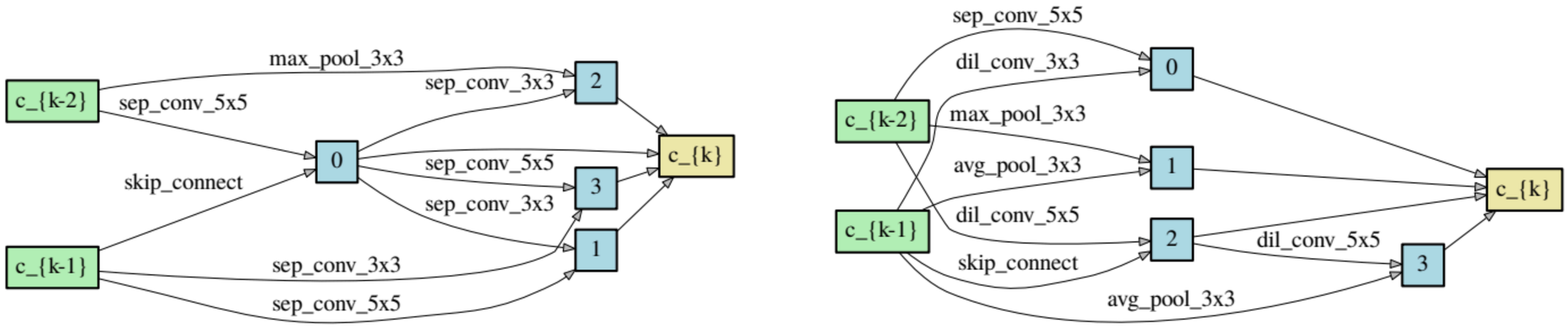
Li & Talwalkar 2019

Yu et al. 2019

Real et al. 2019

- Cell construction with multi-fidelity random search!

  If you constrain the search space enough, you can get SOTA results with random search!



(a) Normal Cell (b) Reduction Cell

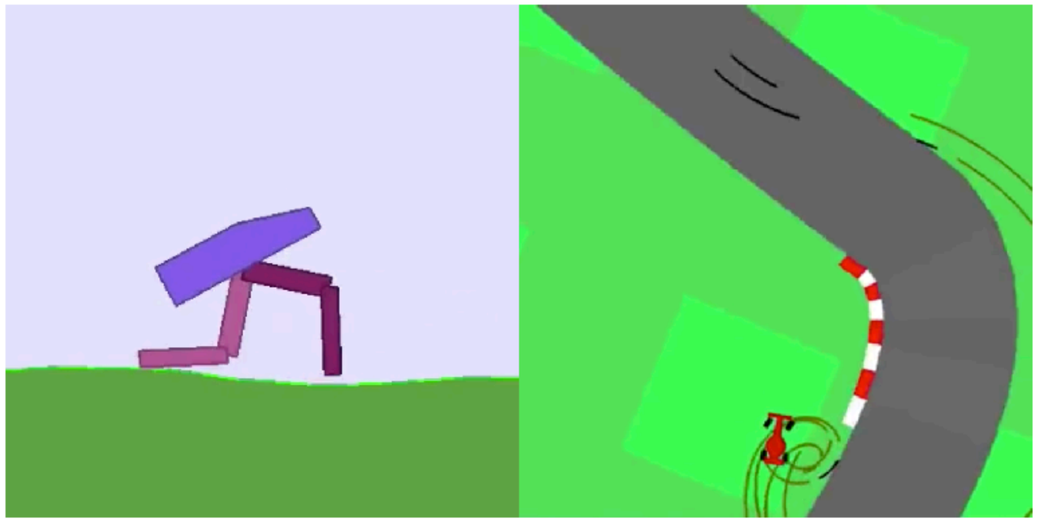
Convolutional Cells on CIFAR-10 Benchmark: Best architecture found by random search with weight-sharing.

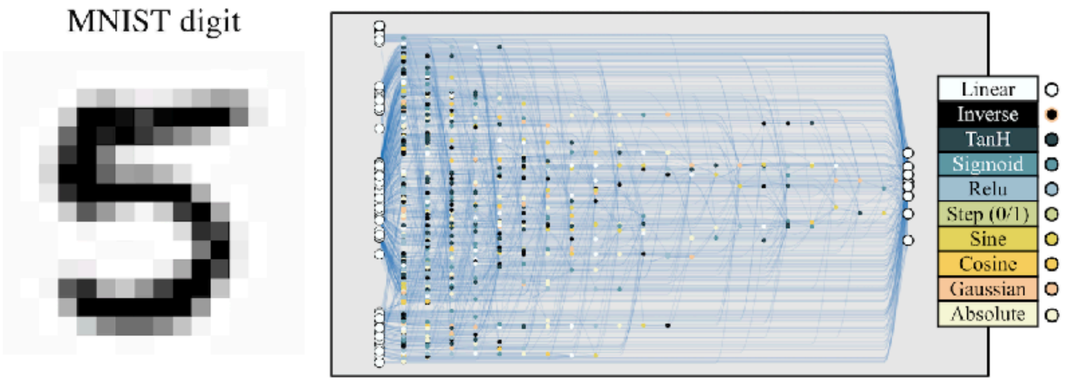
Figure source: Li & Talwalkar., 2019

Manual priors: Weight sharing

Weight-agnostic neural networks

- ALL weights are shared
- Only evolve the architecture?
  - Minimal description length
  - Baldwin effect?





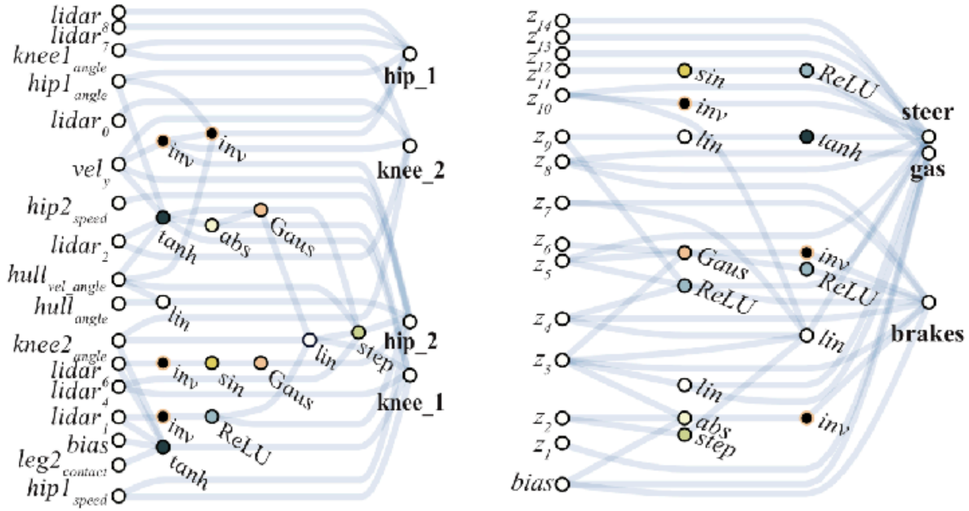
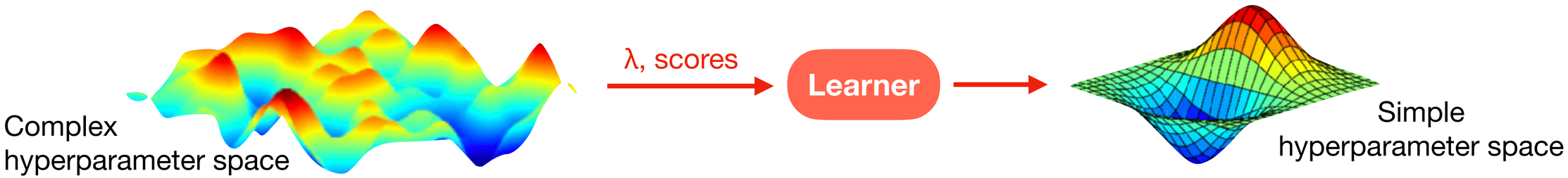


Figure source: Gaier & HA., 2019

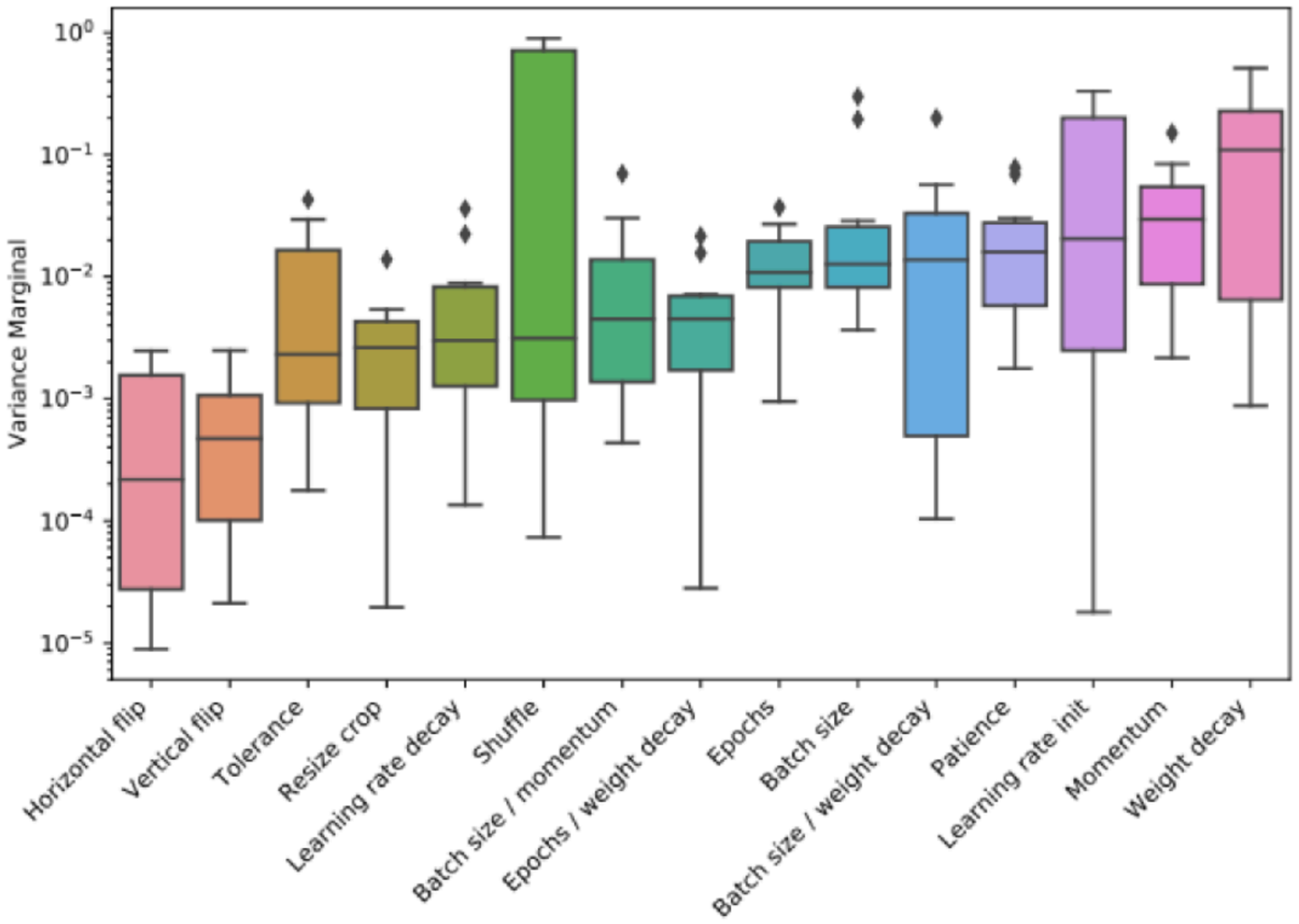
Learning hyperparameter priors



[1] van Rijn & Hutter 2018

Learn hyperparameter importance

- **Functional ANOVA** [1]
  - Select hyperparameters that cause variance in the evaluations.
  - Useful to speed up black-box optimization techniques



ResNets for image classification

Figure source: van Rijn & Hutter, 2018

Learn defaults + hyperparameter importance

[1] Probst et al. 2018

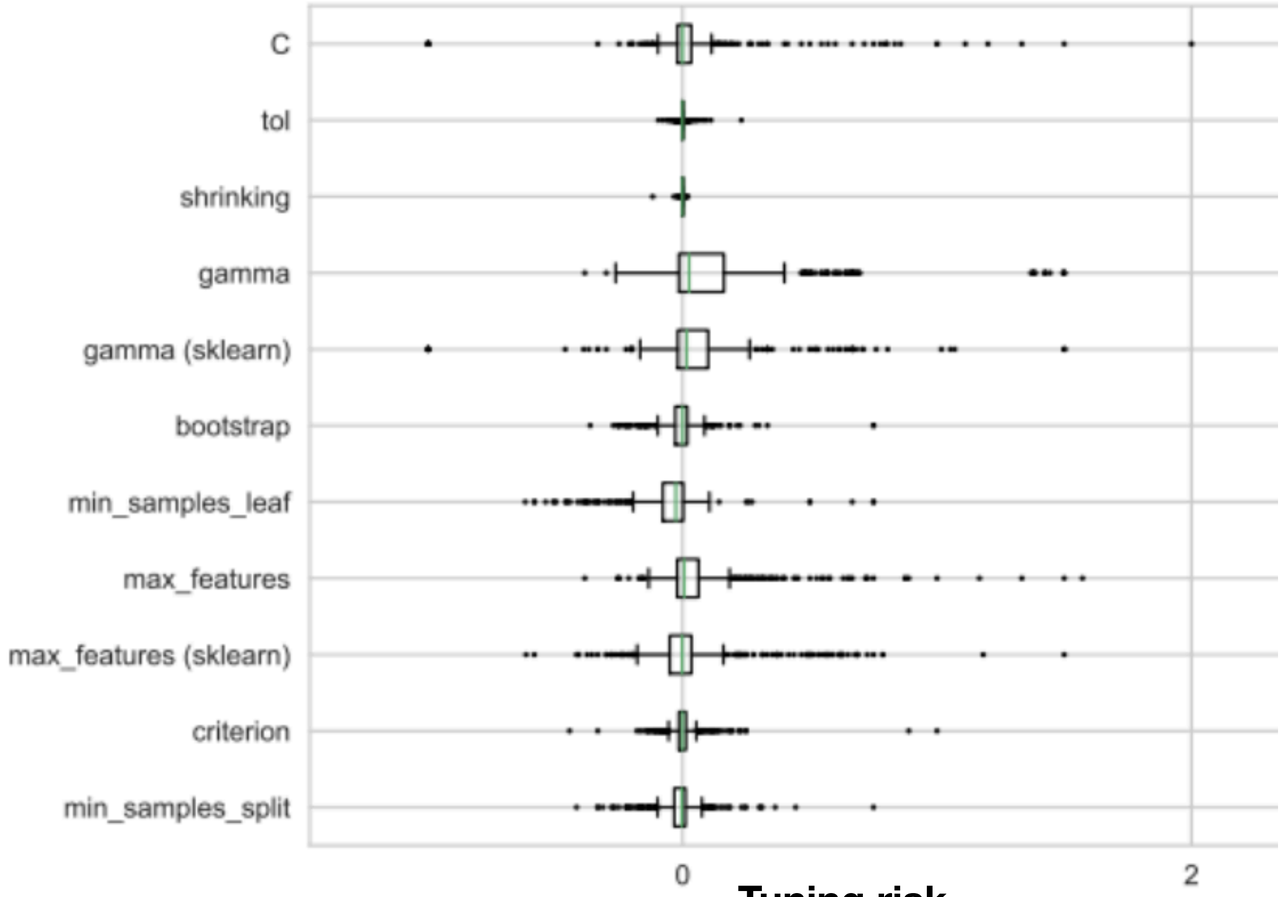
[2] Weerts et al. 2018

[3] van Rijn et al. 2018

- **Tunability** [1,2,3]
  Learn good defaults, measure importance as improvement via tuning

function
max_features
m = 0.16*p
m = p ^0.74
m = 1.15^sqrt(p)
m = sqrt(p)
gamma
m = 0.00574*p
m = 1/p
m = 0.006

Learned defaults

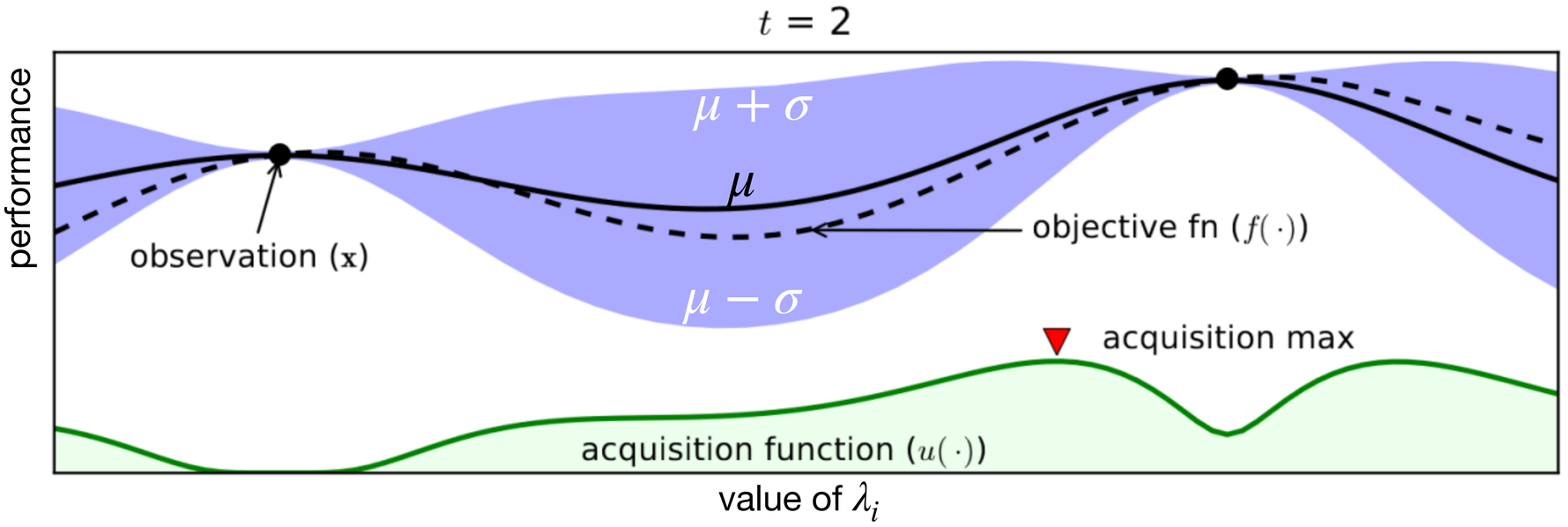


Tuning risk

Mockus, 1974

Bayesian Optimization (interlude)

- Start with a few (random) hyperparameter configurations
- Fit a ***surrogate model*** to predict other configurations
- Probabilistic regression: mean μ and standard deviation σ (blue band)
- Use an ***acquisition function*** to trade off exploration and exploitation, e.g. Expected Improvement (EI)
- Sample for the best configuration under that function



Bayesian Optimization

- Repeat until some stopping criterion:
  - Fixed budget
  - Convergence
  - EI threshold

- Theoretical guarantees

  Srinivas et al. 2010, Freitas et al. 2012, Kawaguchi et al. 2016

- Also works for non-convex, noisy data
- Used in AlphaGo

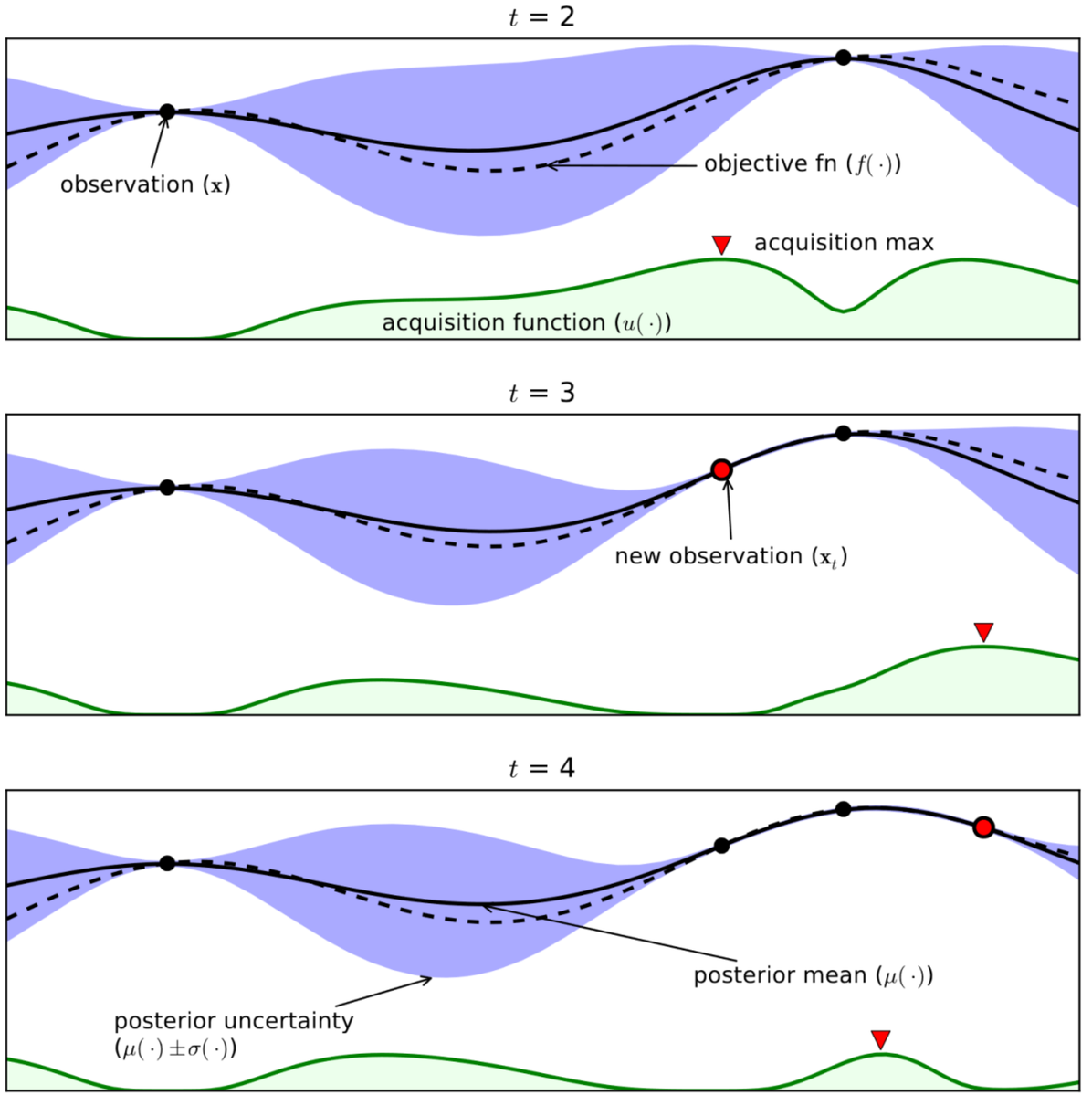


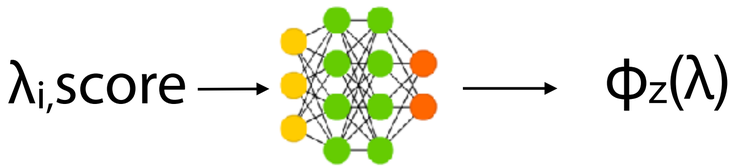
Figure source: Shahriari 2016

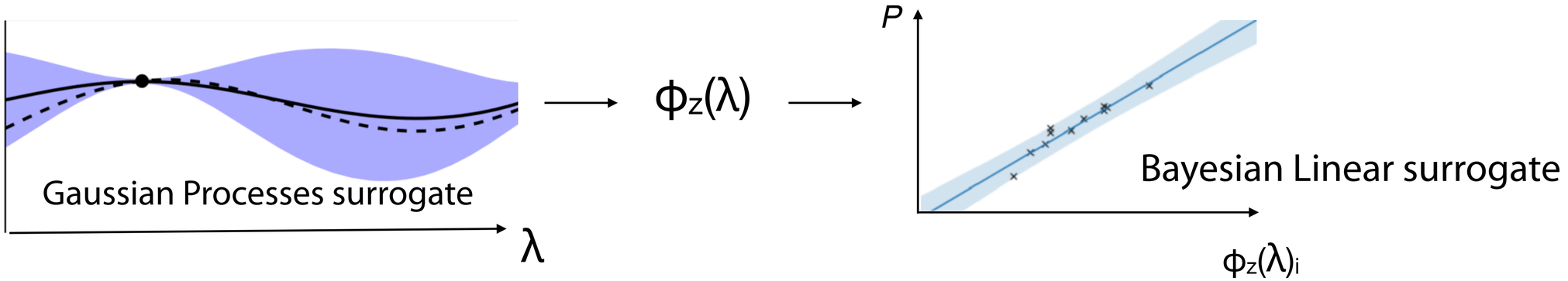
Perrone et al. 2018

Learn basis expansions for hyperparameters

- Hyperparameters can interact in very non-linear ways
- Use a neural net to learn a suitable basis expansion $\phi_z(\lambda)$ for all tasks
- You can use Bayesian linear models, transfers info on configuration space

Learn basis expansion on lots of data (e.g. OpenML)



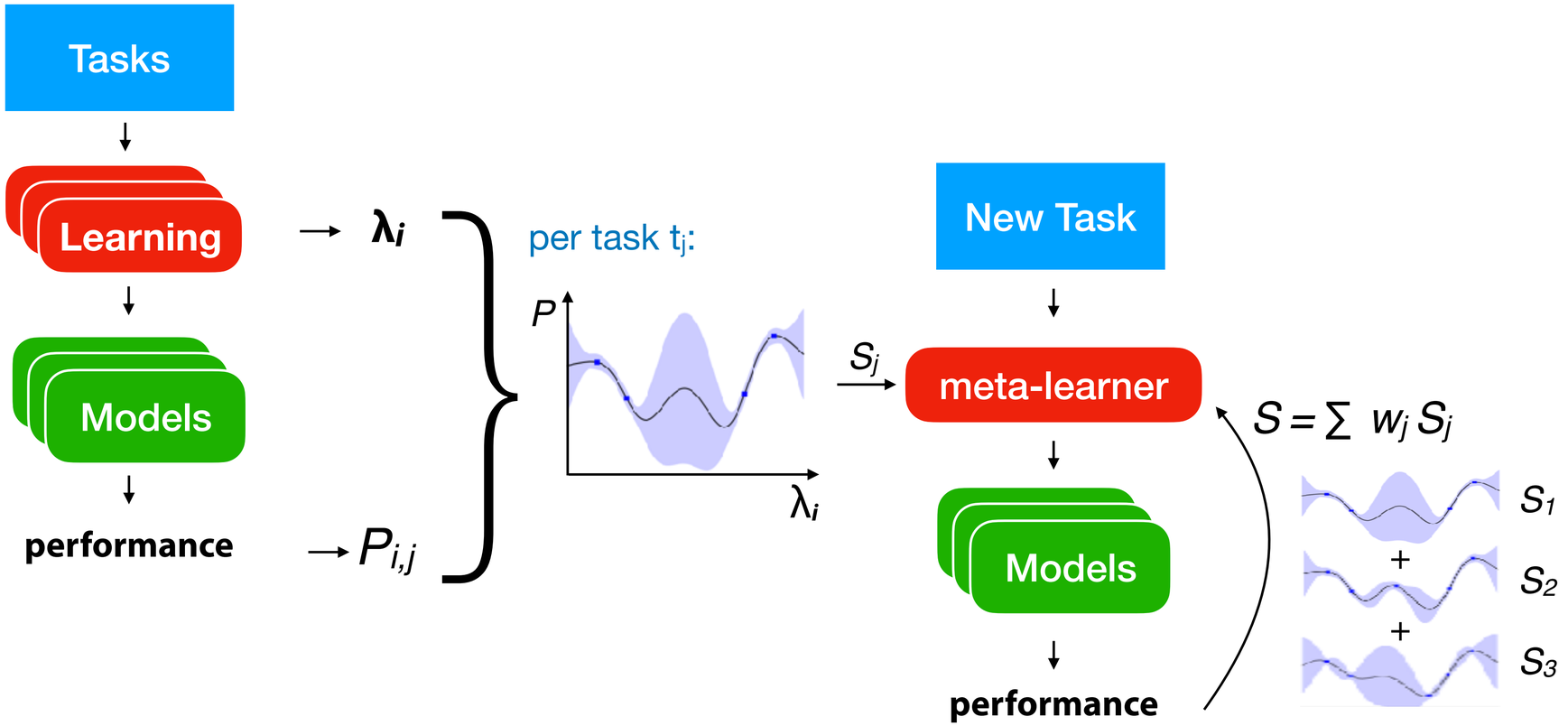


[1] Wistuba et al. 2018

[2] Feurer et al. 2018

Surrogate model transfer

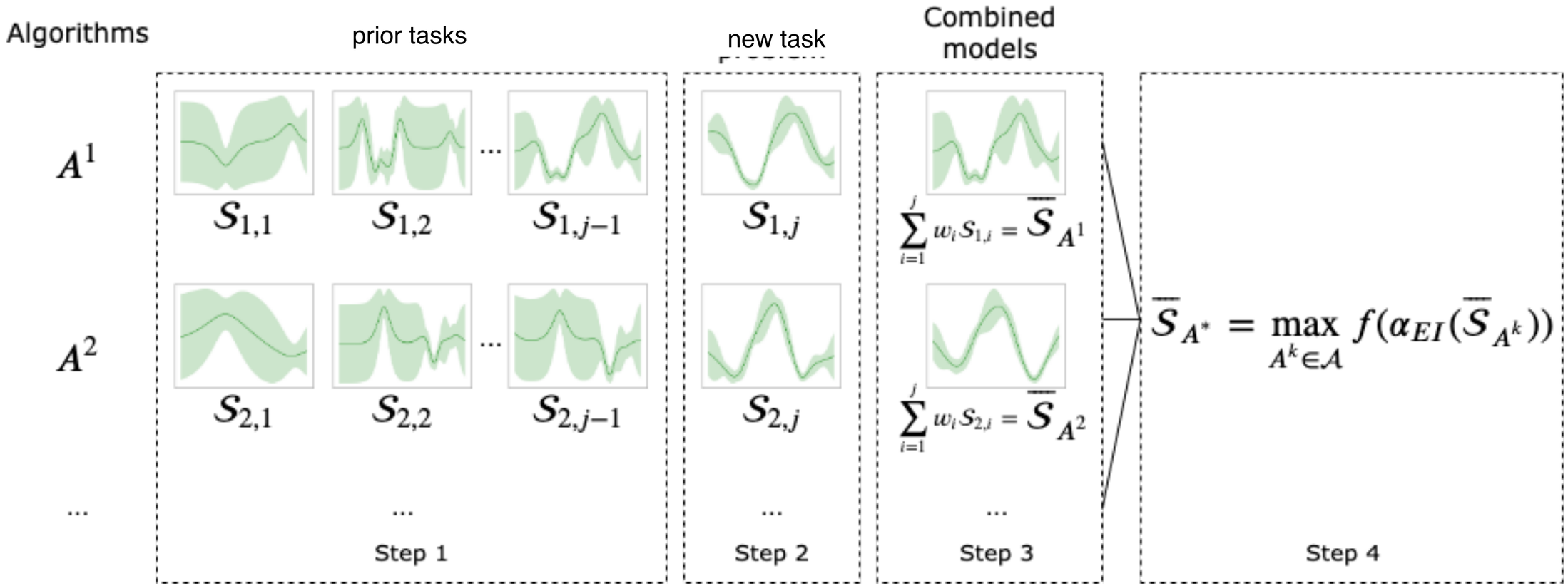
- If task j is similar to the new task, its surrogate model S_j will likely transfer well
- Sum up all S_j predictions, weighted by task similarity (as in active testing)[1]
- Build combined Gaussian process, weighted by current performance on new task[2]



Manolache & Vanschoren 2019

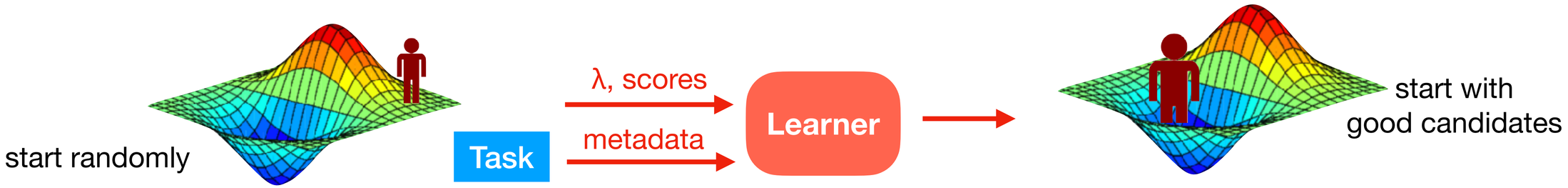
Surrogate model transfer

- Store surrogate model S_{ij} for every pair of task i and algorithm j
- Simpler surrogates, better transfer
- Learn weighted ensemble -> significant speed up in optimization



Warm starting

(what works on similar tasks?)



How to measure task similarity?

- Hand-designed (statistical) meta-features that describe (tabular) datasets [1]
- Task2Vec: task embedding for image data [2]
- Optimal transport: similarity measure based on comparing probability distributions [3]
- Metadata embedding based on textual dataset description [4]
- Dataset2Vec: compares batches of datasets [5]
- Distribution-based invariant deep networks [6]

Figure source: Alvarez-Melis et al. 2020

[1] Vanschoren 2018
[2] Achille et al. 2019
[3] Alvarez-Melis et al. 2020
[4] Drori et al. 2019
[5] Jooma et al. 2020
[6] de Bie et al. 2020

Feurer et al. 2015

Warm-starting with kNN

- Find k most similar tasks, warm-start search with best λ_i
  - Auto-sklearn: Bayesian optimization (SMAC)
    - Meta-learning yield better models, faster
    - Winner of AutoML Challenges

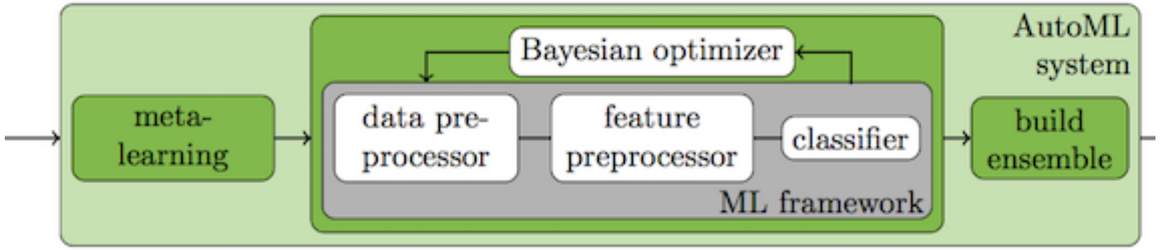
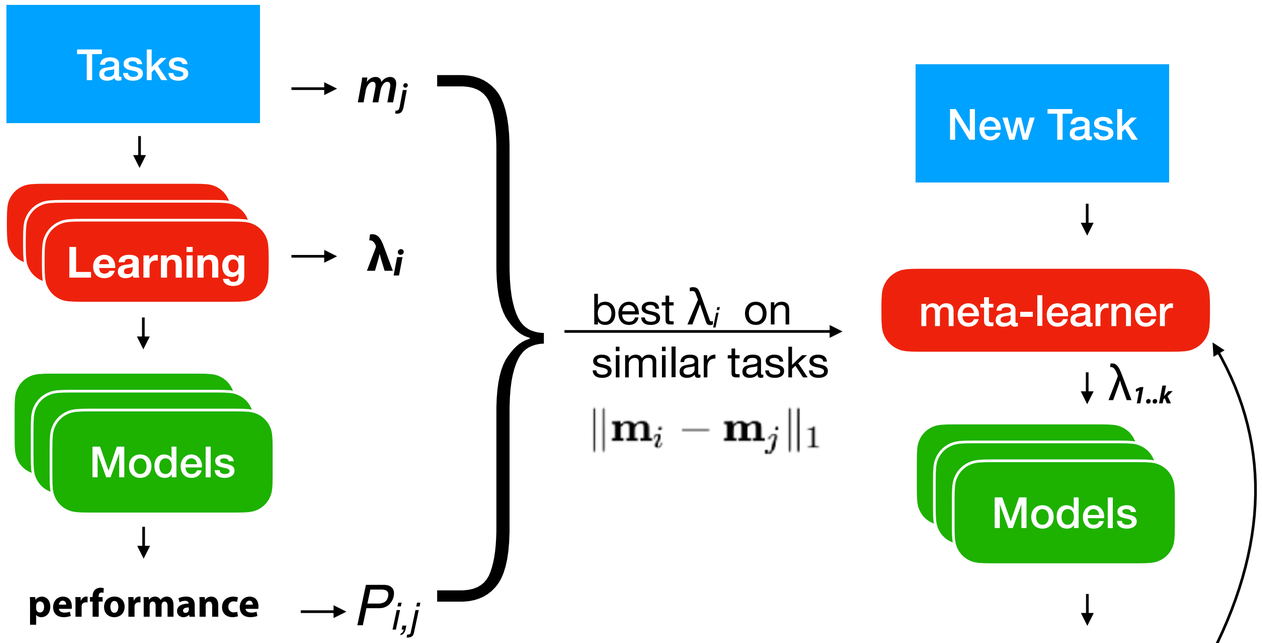
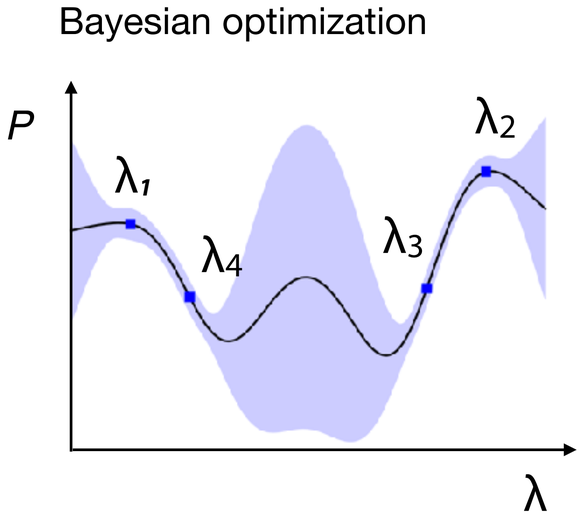


Figure source: Feurer et al., 2015



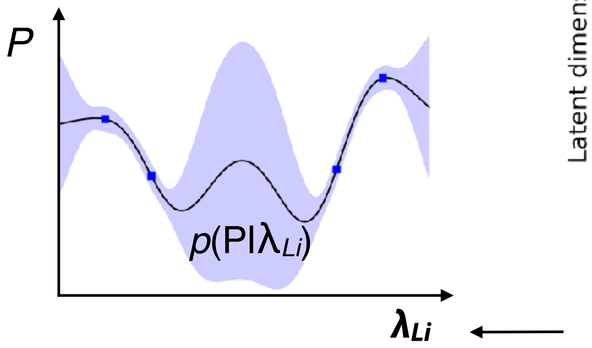


Fusi et al. 2017

Probabilistic Matrix Factorization

- Collaborative filtering: configurations λ_i are `rated' by tasks t_j

- Learn latent representation for tasks T and configurations λ
- Use meta-features to warm-start on new task
- Returns probabilistic predictions for Bayesian optitmization



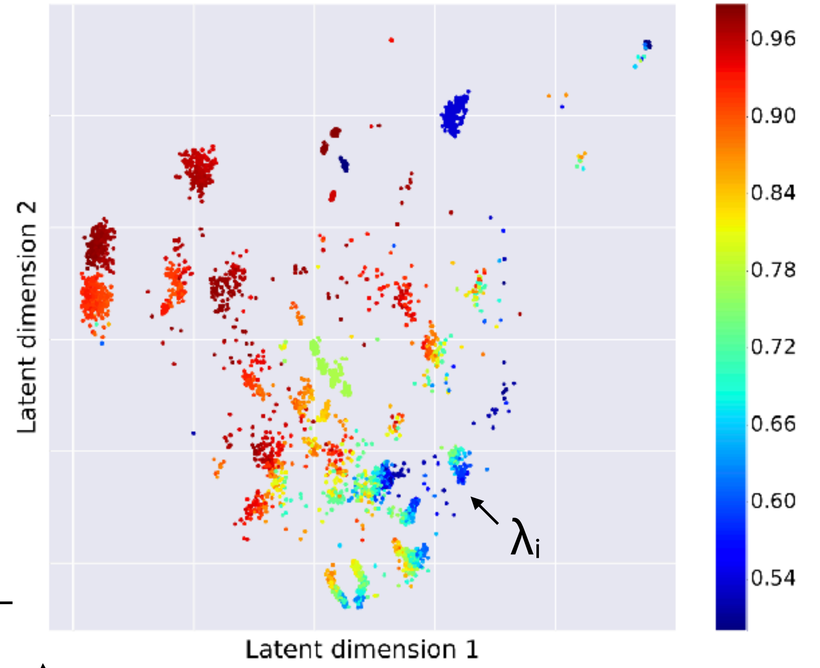
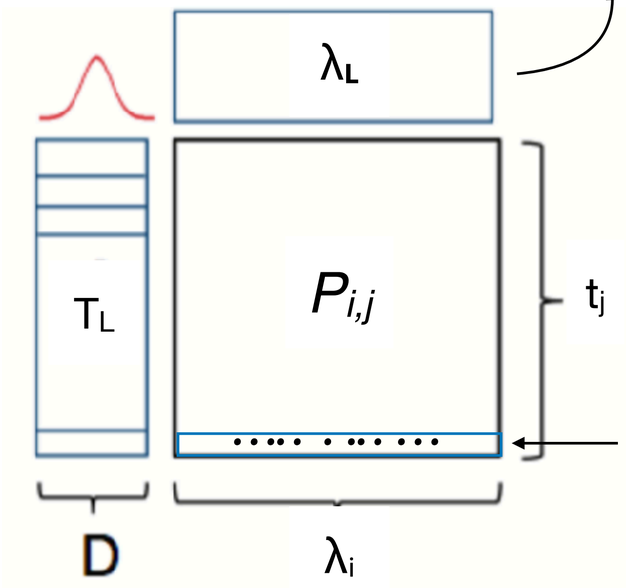


Figure source: Fusi et al., 2017



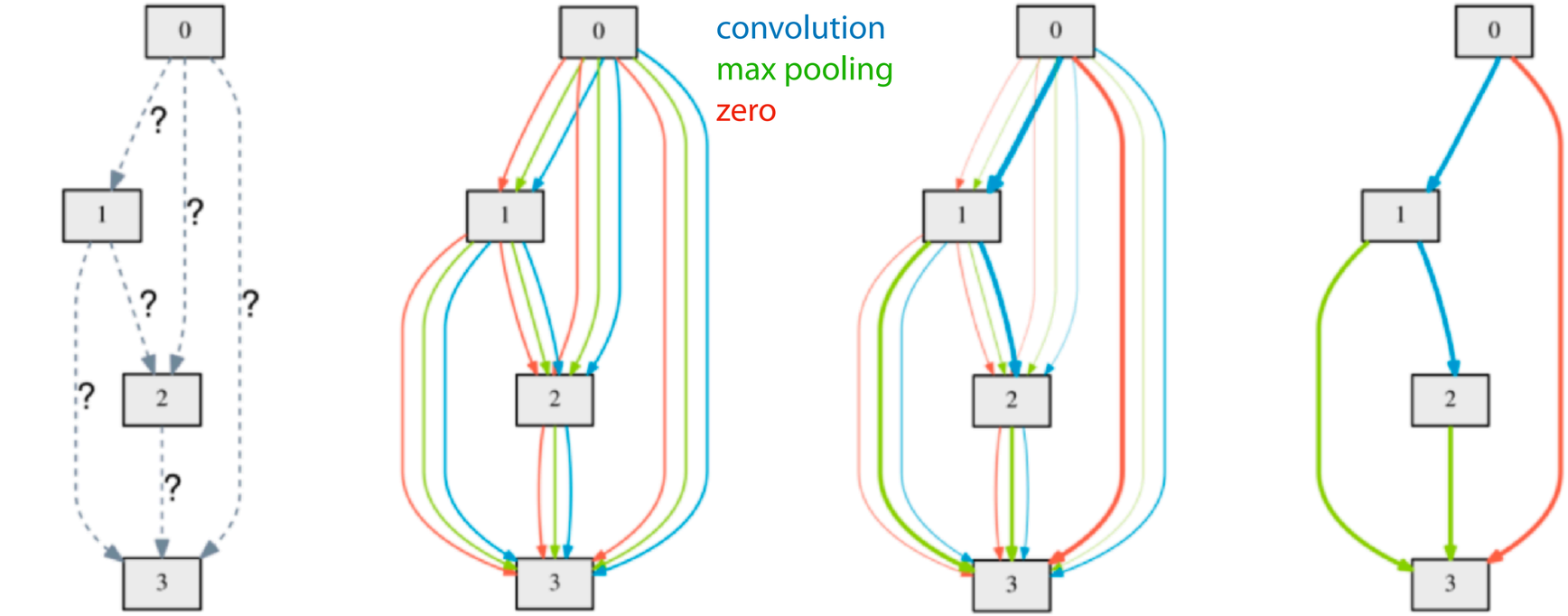
latent representation

t_{new} warm-started with $\lambda_{1..k}$

Liu et al. 2018

DARTS: Differentiable NAS

- Fixed (one-shot) structure, learn which operators to use
- Give all operators a weight α_i
- Optimize α_i and model weights ω_j using bilevel optimization
  - approximate $\omega_j^*(\alpha_i)$ by adapting ω_j after every training step



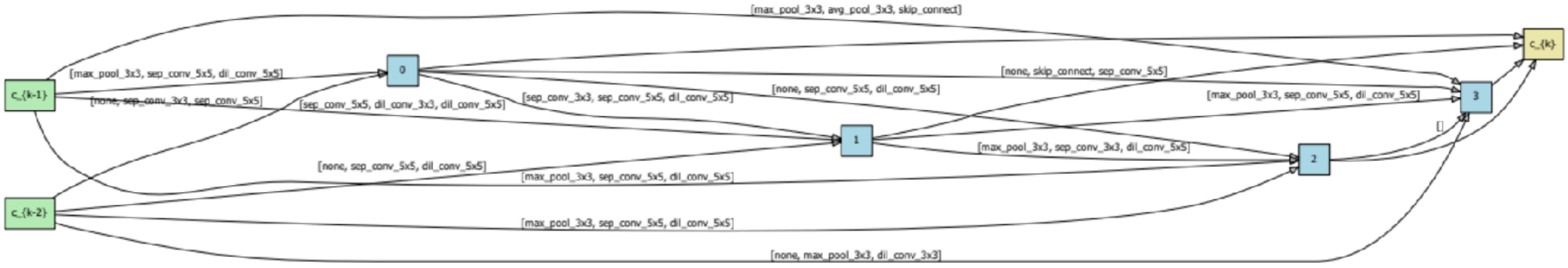
One-shot model | **operator weights α_i** | **interleaved optimization of α_i and ω_j with SGD** | **argmax α_i**

Figure source: Liu et al., 2018

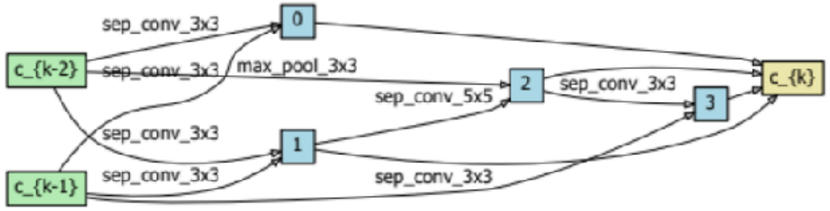
Grobelnik and Vanschoren, 2021

Warm-started DARTS

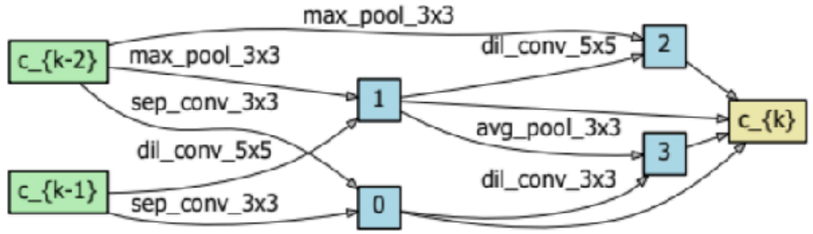
- Warm-start DARTS with architectures that worked well on similar problems
- Slightly better performance, but much faster (5x)



(a) FLOWER_V3 transfer architecture for normal cell



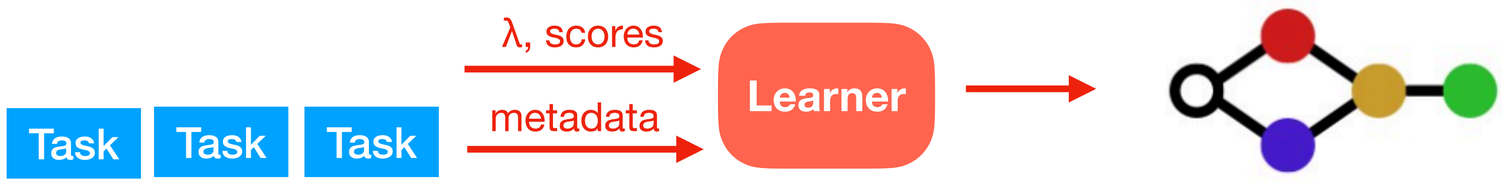
(b) Normal cell found on *aircraft* task



(c) Normal cell found on *birds* task

Meta-models

(learn how to build models/components)



Algorithm selection models

[1] Brazdil et al. 2009, Lemke et al. 2015

[2] Sun and Pfahringer 2013, Pinto et al. 2017

[3] Sanders and C. Giraud-Carrier 2017

[4] Yang et al. 2018

- Learn direct mapping between meta-features and $P_{i,j}$
  - Zero-shot meta-models: predict best λ_i given meta-features [1]

$$m_j \rightarrow \boxed{\text{meta-learner}} \rightarrow \lambda_{best}$$

  - Ranking models: return ranking $\lambda_{1..k}$ [2]

$$m_j \rightarrow \boxed{\text{meta-learner}} \rightarrow \lambda_{1..k}$$

  - Predict which algorithms / configurations to consider / tune [3]

$$m_j \rightarrow \boxed{\text{meta-learner}} \rightarrow \boldsymbol{\Lambda}$$

  - Predict performance / runtime for given Θ_i and task [4]

$$m_j, \lambda_i \rightarrow \boxed{\text{meta-learner}} \rightarrow P_{ij}$$

- Can be integrated in larger AutoML systems: warm start, guide search,…

[1] Ramachandran et al. 2017

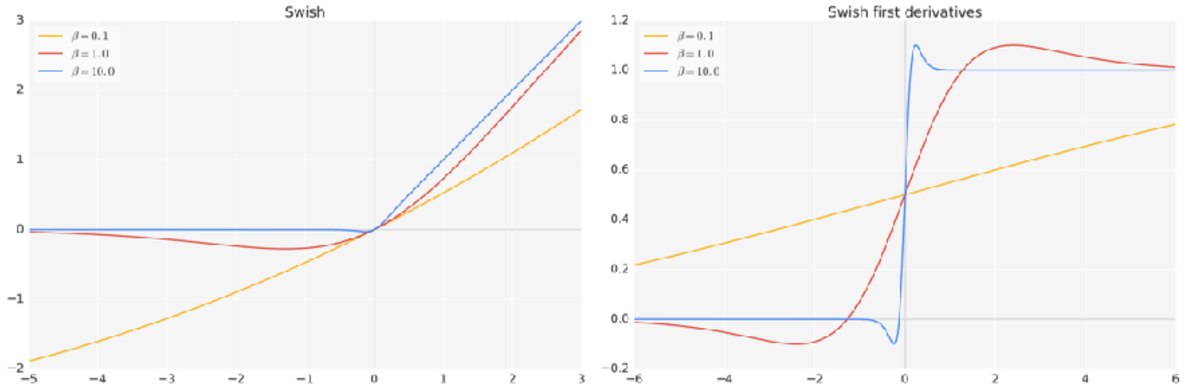
[2] Bello et al. 2017

[3] Volpp et al. 2020

Learning model components

- Learn nonlinearities: RL-based search of space of likely useful activation functions [1]
  - E.g. Swish can outperform ReLU

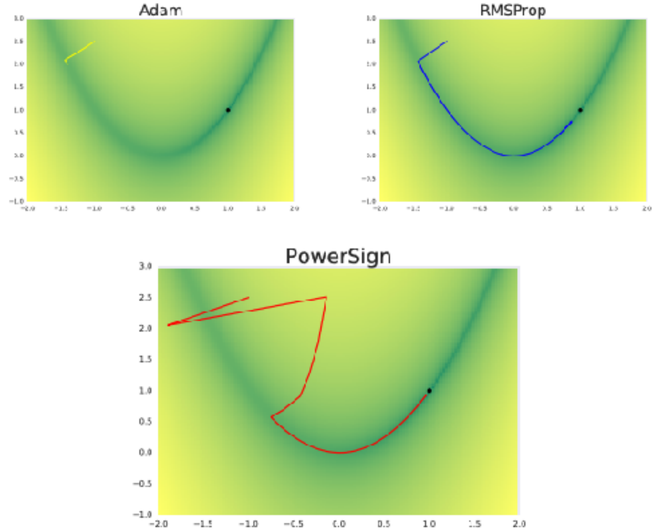
$$Swish : \frac{x}{1 + e^{-\beta x}}$$



- Learn optimizers: RL-based search of space of likely useful update rules [2]
  - E.g. PowerSign can outperform Adam, RMPprop

$$PowerSign : e^{sign(g)sign(m)}g$$

g: gradient, m:moving average



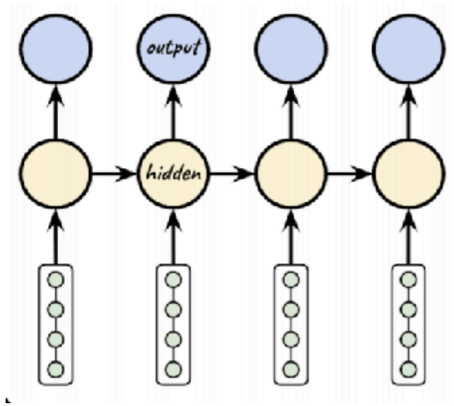
- Learn acquisition functions for Bayesian optimization [3]

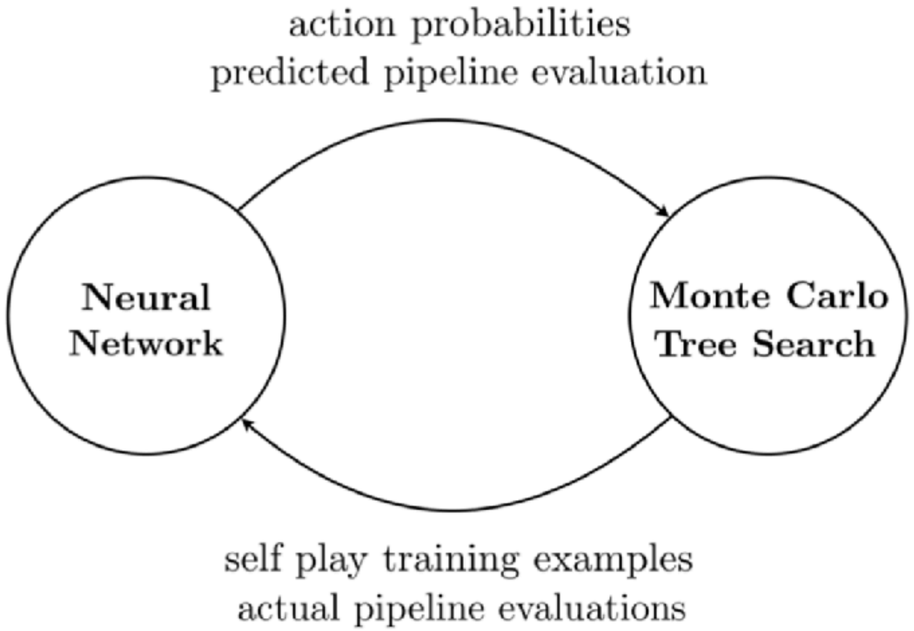
Figure source: Ramachandran et al., 2017 (top), Bello et al. 2017 (bottom)

MOSAIC [Rakotoarison et al. 2019]
AlphaD3M [Drori et al. 2019]

Monte Carlo Tree Search + reinforcement learning

- ***Self-play:***
  - Game actions: insert, delete, replace components in a pipeline
  - Monte Carlo Tree Search builds pipelines given action probabilities
    - With grammar to avoid invalid pipelines
  - Neural network (LSTM) Predicts pipeline performance (can be pre-trained on prior datasets)





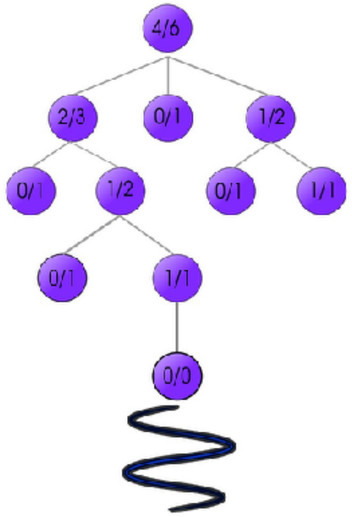


Figure source: Drori et al., 2019

Wong et al. 2018

Neural Architecture Transfer learning

- Warm-start a deep RL controller based on prior tasks
- Much faster than single-task equivalent

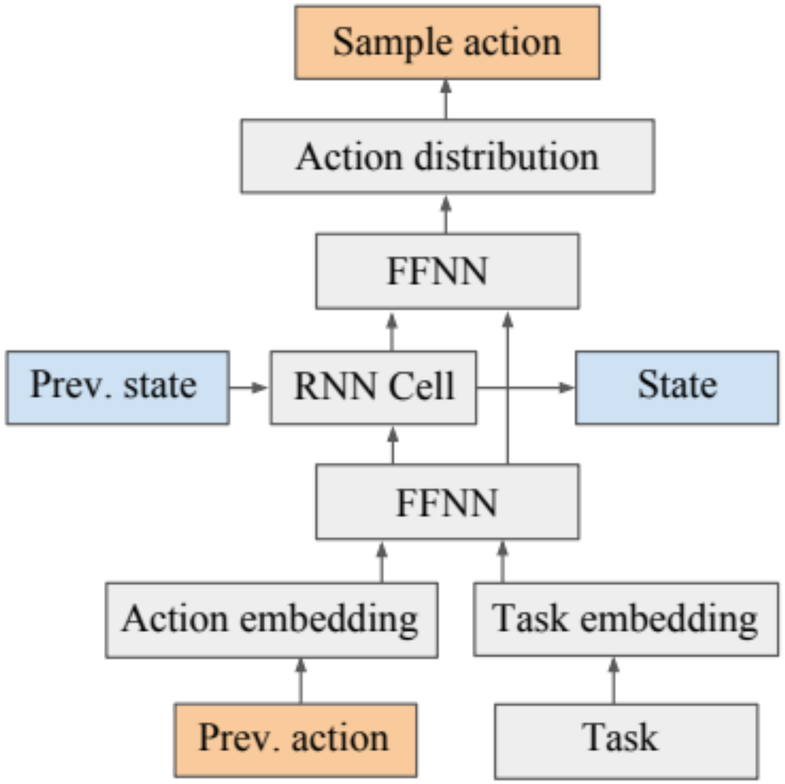


Figure source: Wong et al., 2018

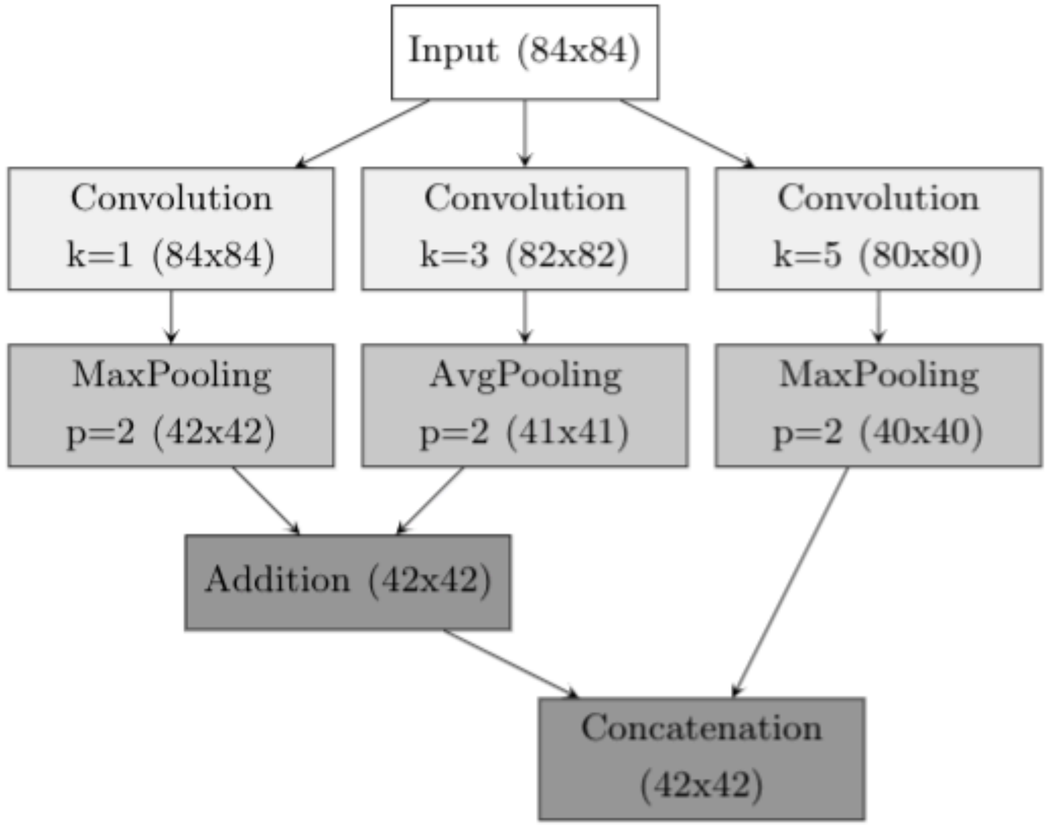
Meta-Reinforcement Learning for NAS

Gomez & Vanschoren, 2019

- Train an agent how to build a neural net, across tasks
- Should transfer but also adapt to new tasks

[0, 0, 0, 0, 0]
[0, 0, 0, 0, 0]
[1, 1, 1, 0, 0]
[2, 2, 2, 1, 0]
[3, 1, 3, 0, 0]
[4, 3, 2, 3, 0]
[5, 1, 5, 0, 0]
[6, 2, 2, 5, 0]
[7, 5, 0, 2, 4]
[8, 7, 0, 0, 0]

Actions: add/remove certain layers in certain locations

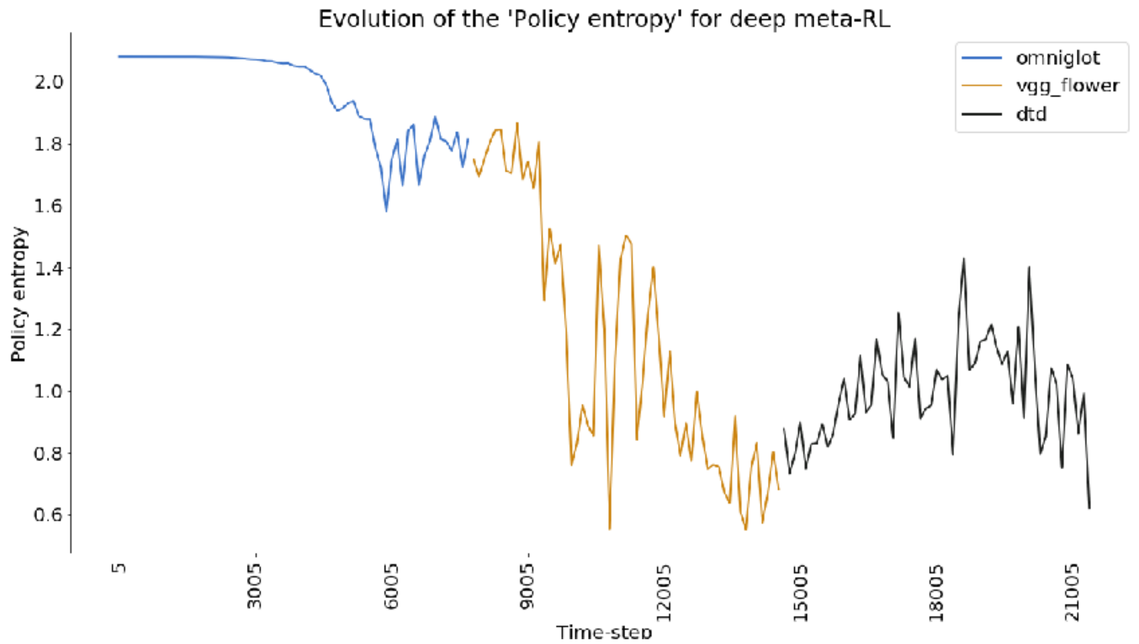


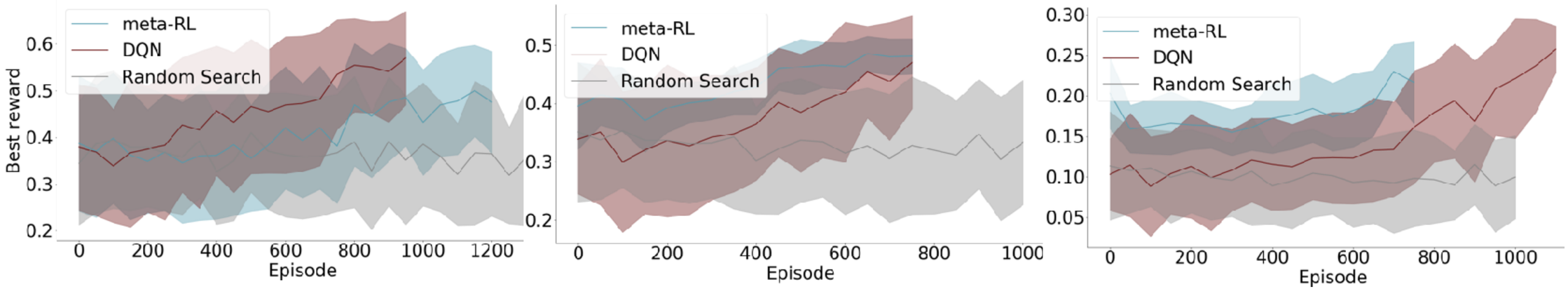
Gomez & Vanschoren, 2019

Meta-Reinforcement Learning for NAS

Results on increasingly difficult tasks:

- Initially slower than DQN, but faster after a few tasks
- Policy entropy shows learning/re-learning





omniglot vgg_flower dtd

Elsken et al., 2020

MetaNAS: MAML + Neural Architecture Search

- Combines gradient based meta-learning (REPTILE) with NAS
- During meta-train, it optimizes the meta-architecture (DARTS weights) along with the meta-parameters (initial weights) Θ
- During meta-test, the architecture can be adapted to the novel task through gradient descent

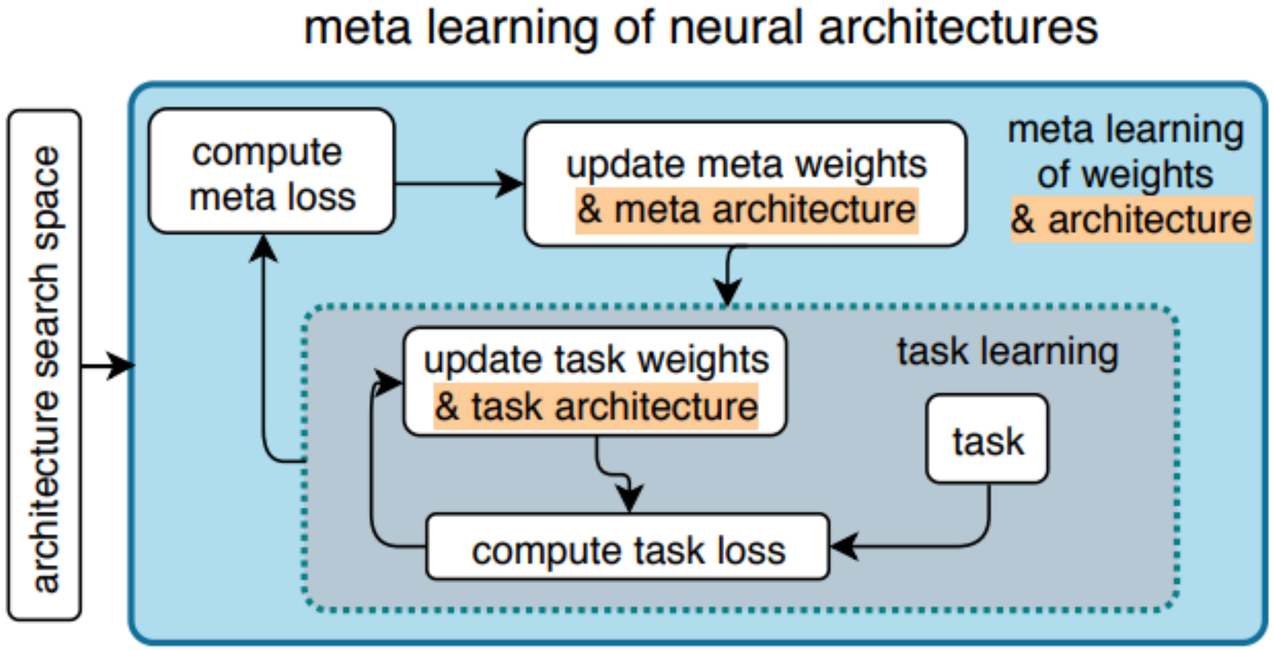
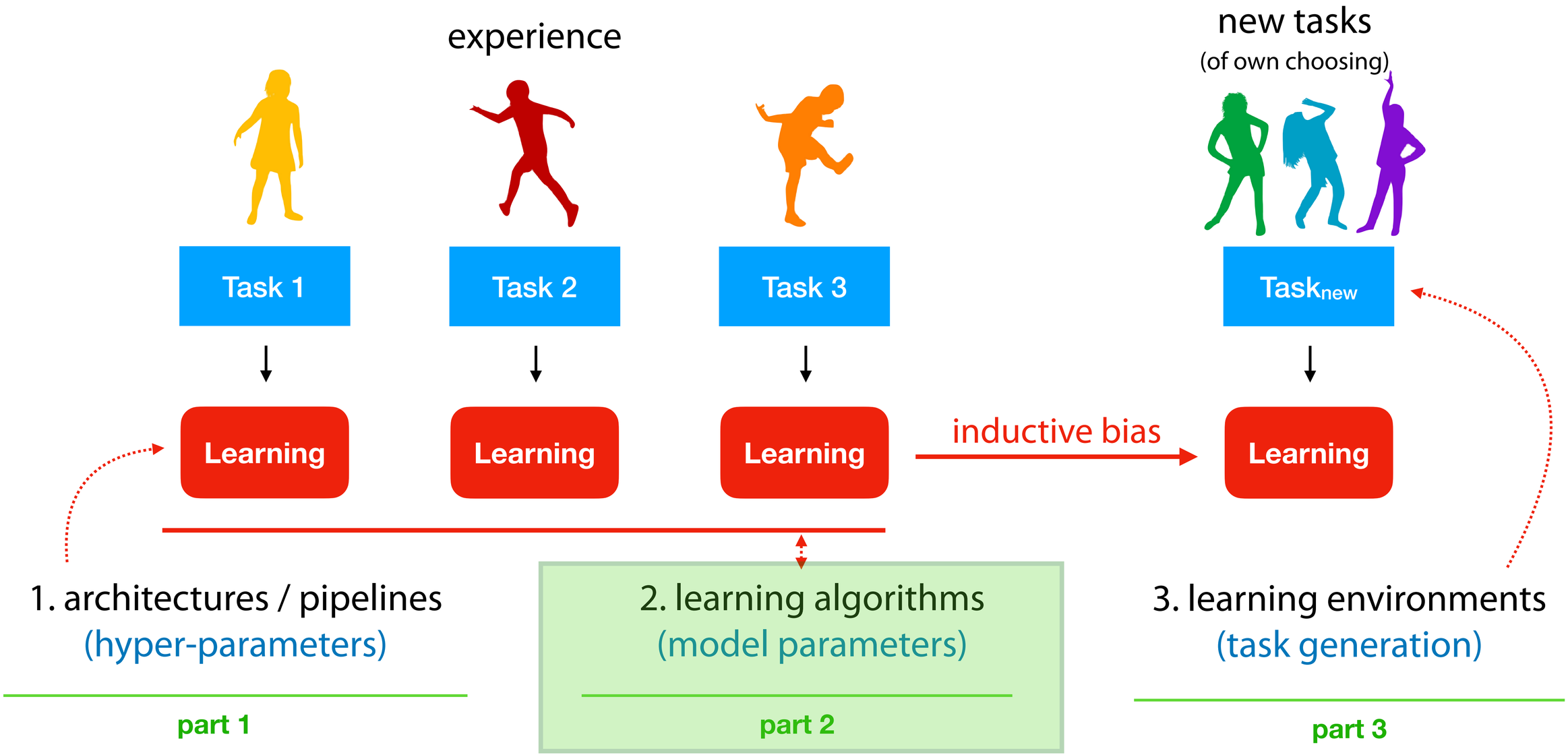


Figure source: Elsken et al., 2020

Clune 2019

What can we learn to learn?

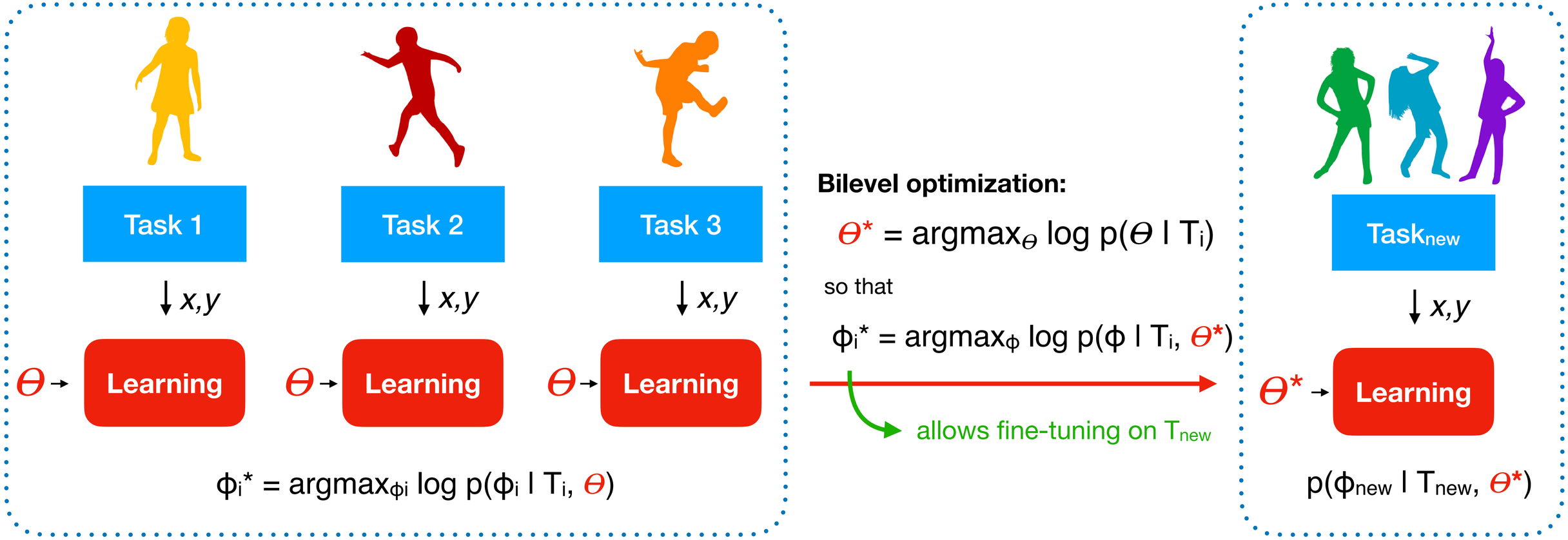
3 *pillars*



Strategy 1: bilevel optimization

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

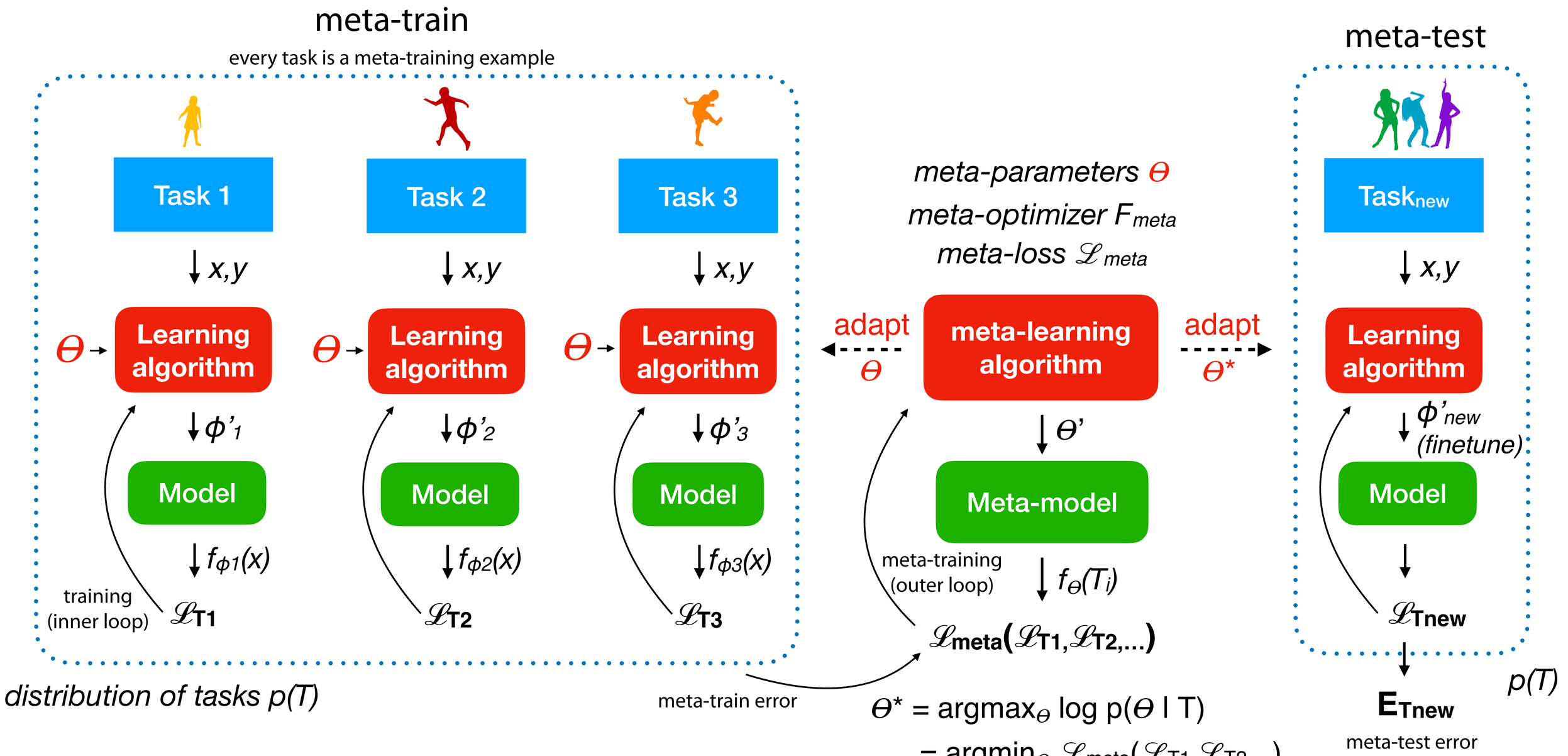
meta-learn Θ across tasks



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

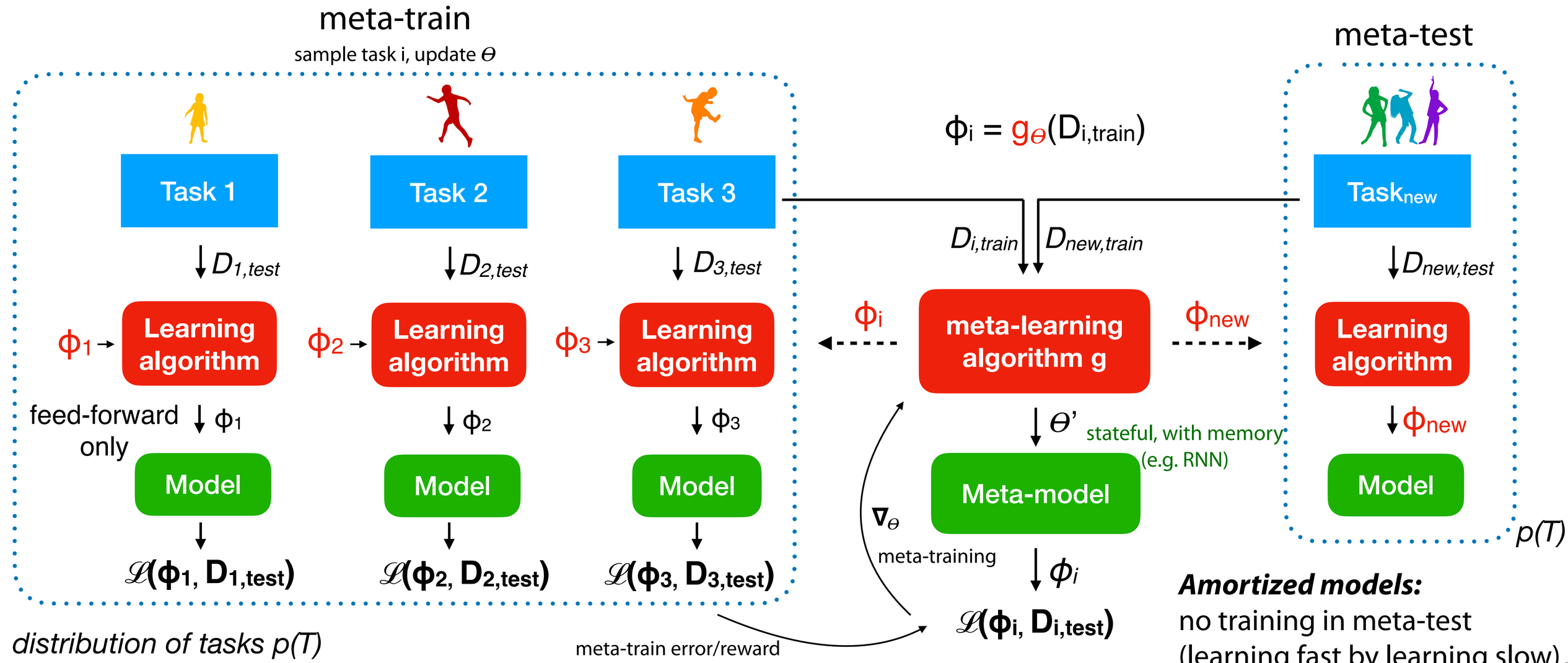
Meta-learning with bilevel optimization



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

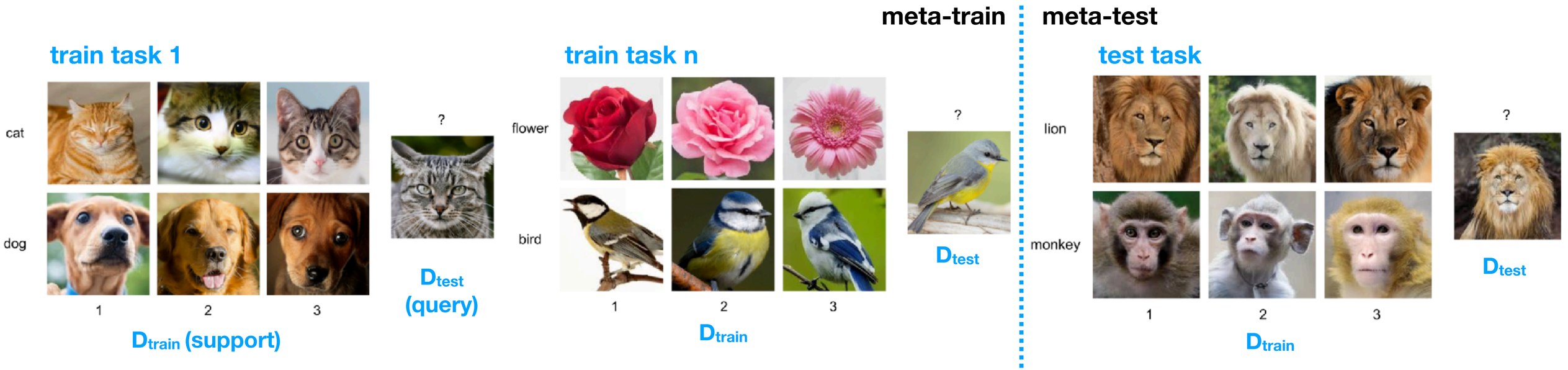


Amortized models:
no training in meta-test
(learning fast by learning slow)

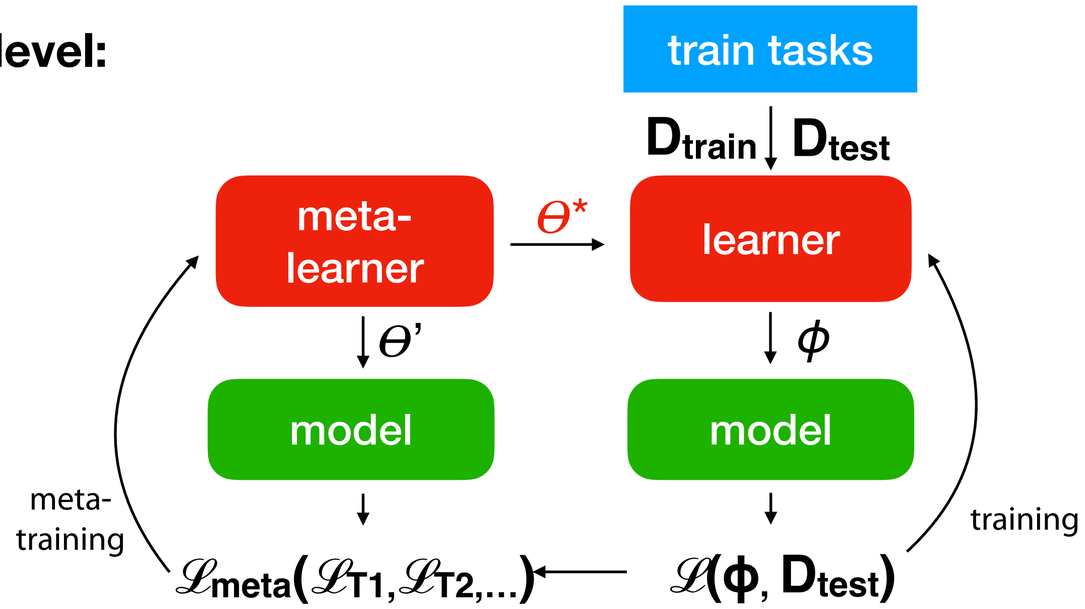
Example: few-shot classification

Vinyals et al. 2016
Triantafillou et al. 2019

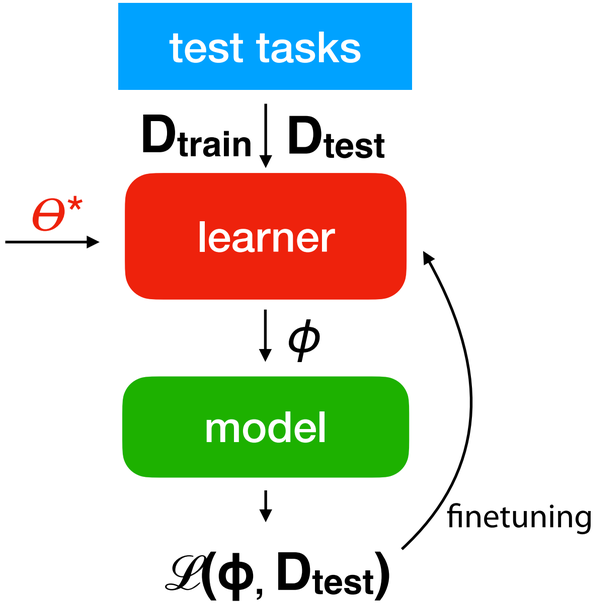
2-way, 3-shot



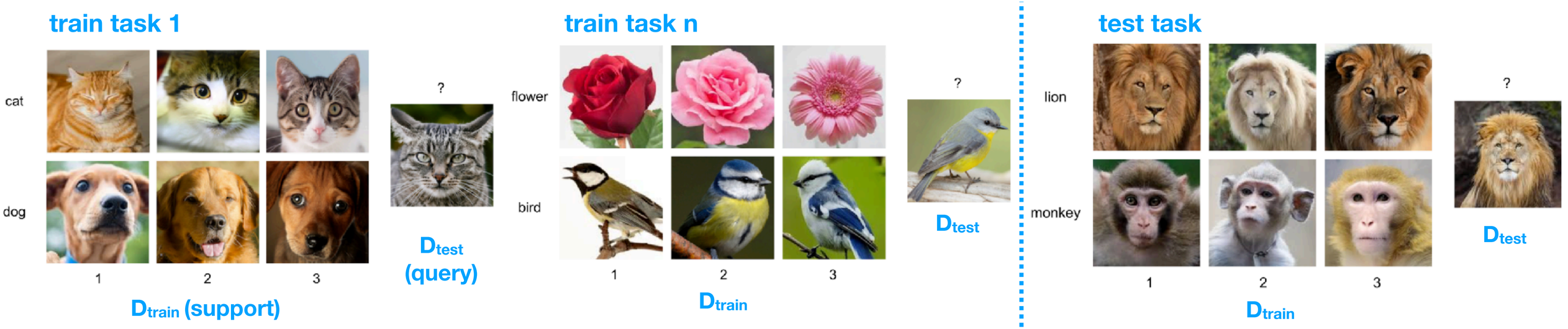
Bilevel:



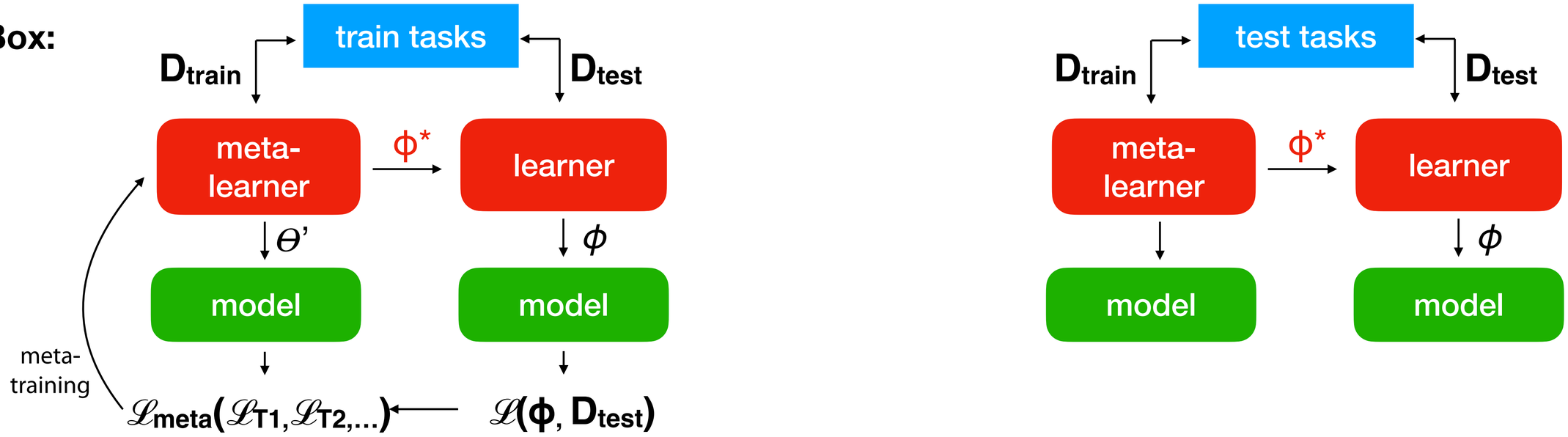
θ^*: initial ϕ
optimizer
metric
…



Example: few-shot classification



Black Box:

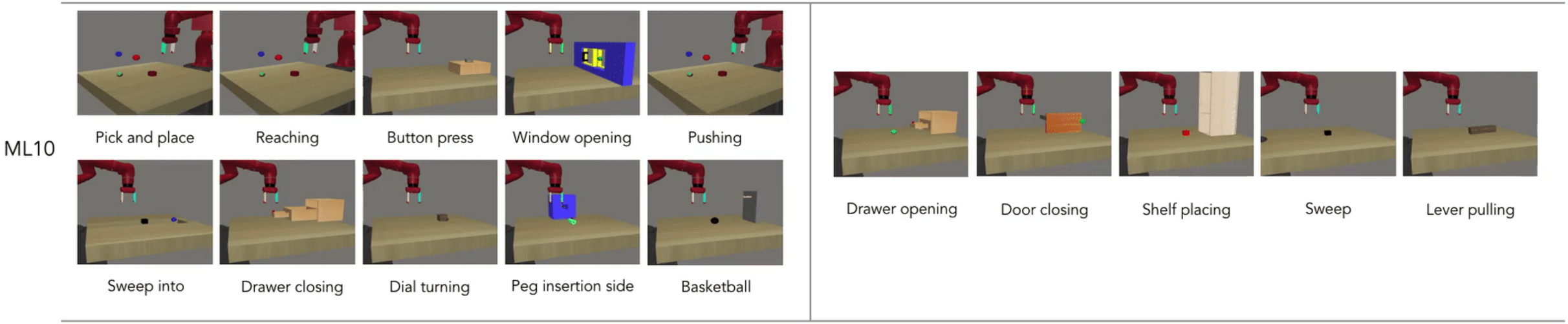


Example: meta-reinforcement learning

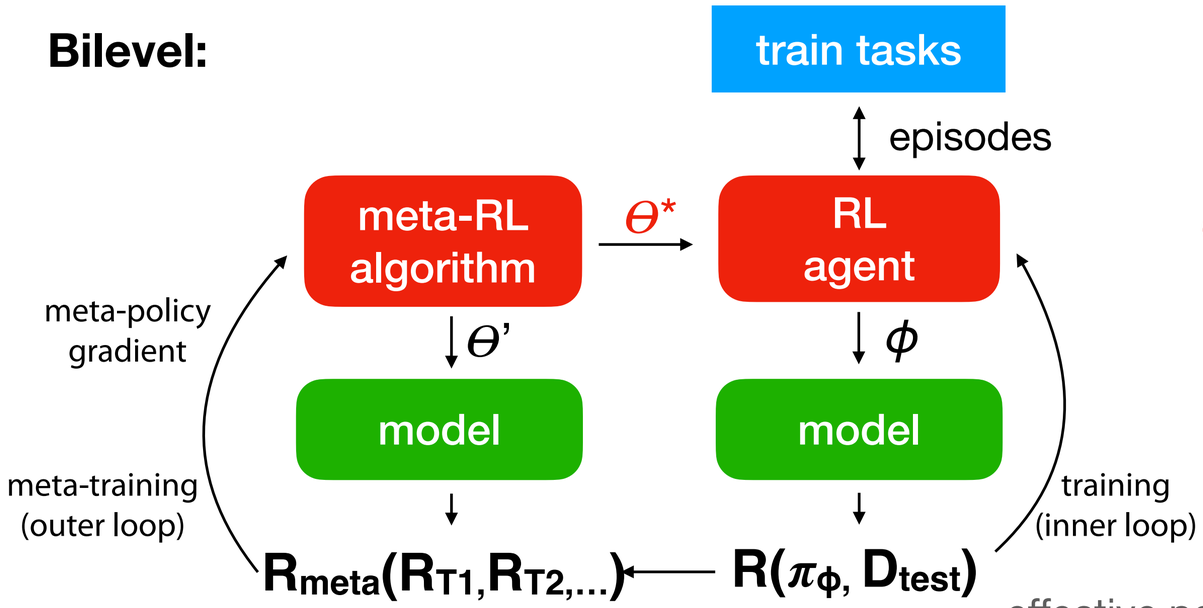
Yu et al. 2019

initial state:
randomized object
and goals positions

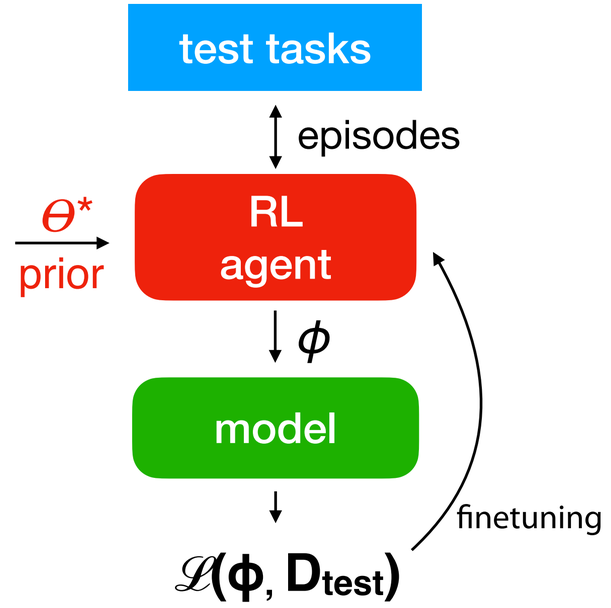
other initial states
or related tasks



Bilevel:



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



effective policy π_ϕ modulated by ϕ

Example: meta-reinforcement learning

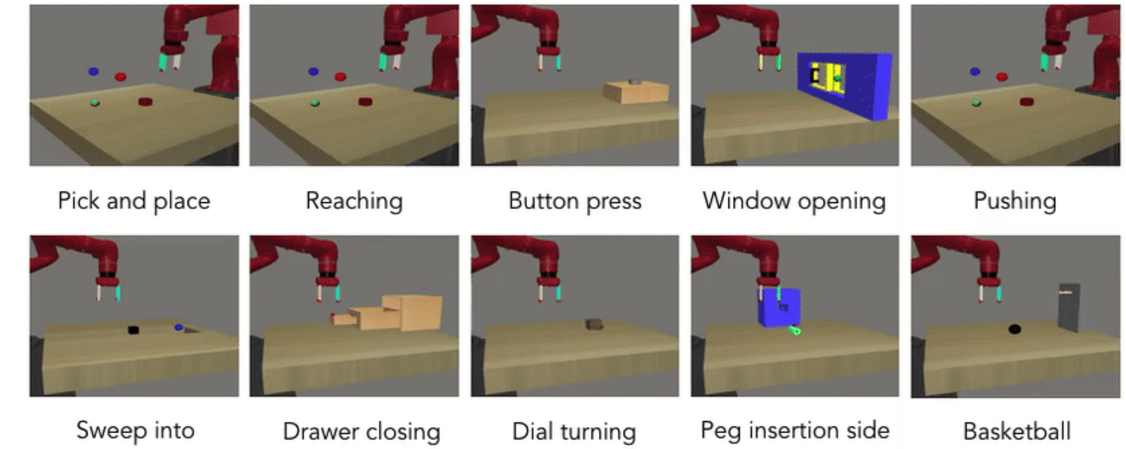
initial state:
randomized object
and goals positions

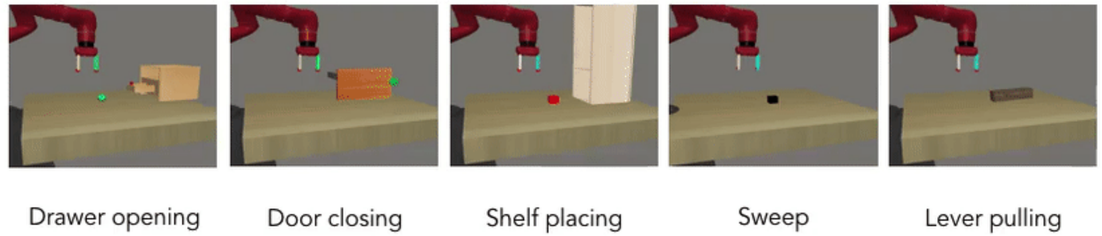
Train tasks

Test tasks

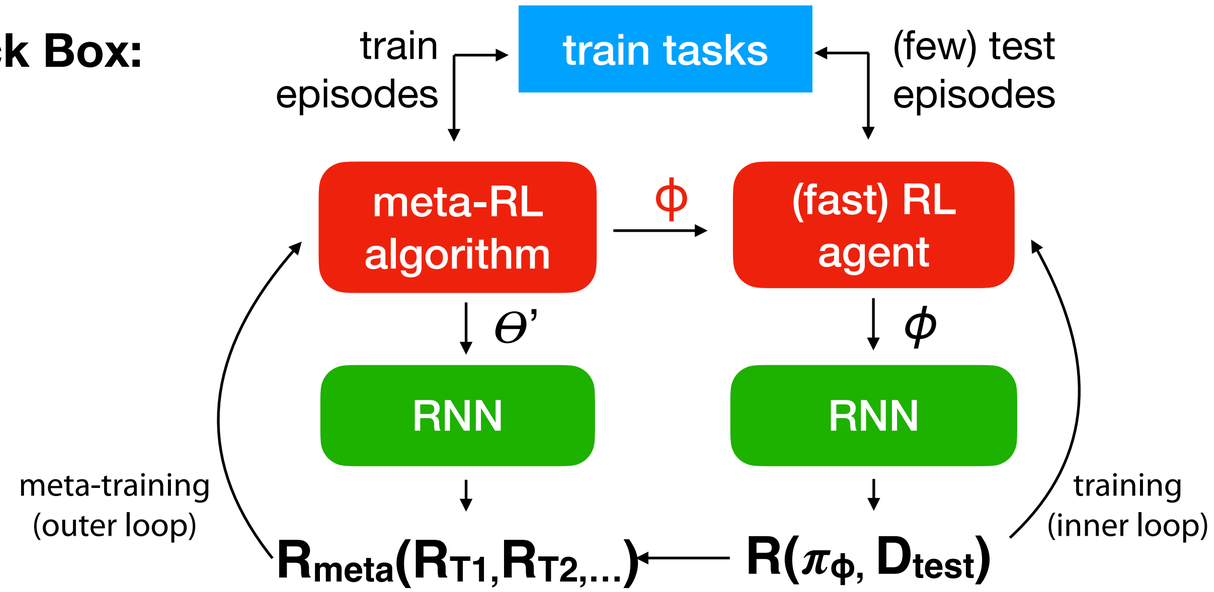
other initial states
or related tasks

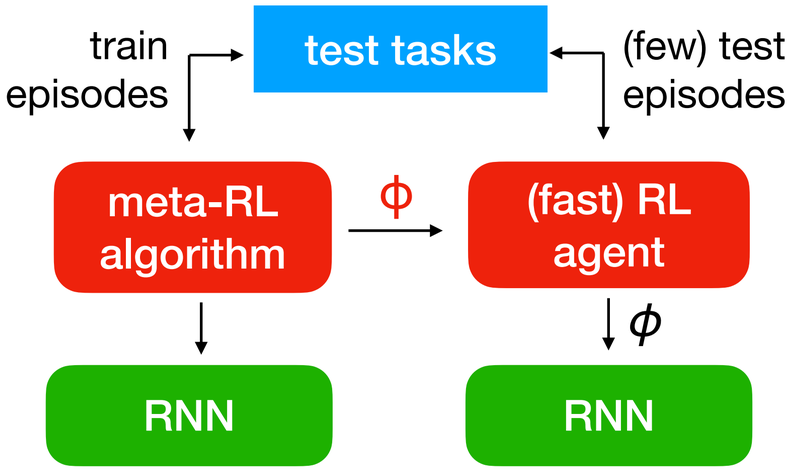
ML10





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

parameter initialization ϕ_0

meta-optimizer F_{meta}

gradient descent

meta-objective $\mathscr{L}_{meta}$

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

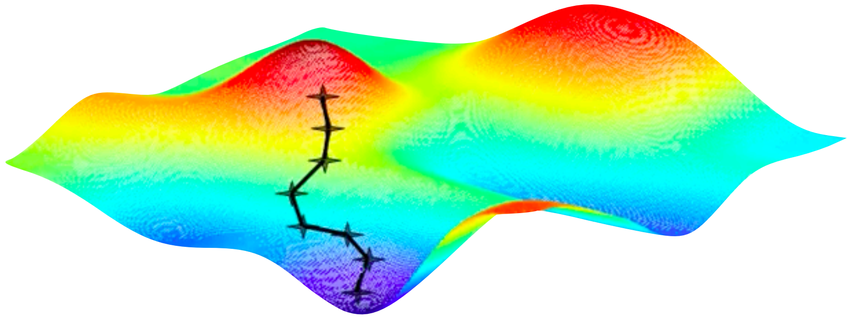
optimizer

meta-optimizer F_{meta}

gradient descent

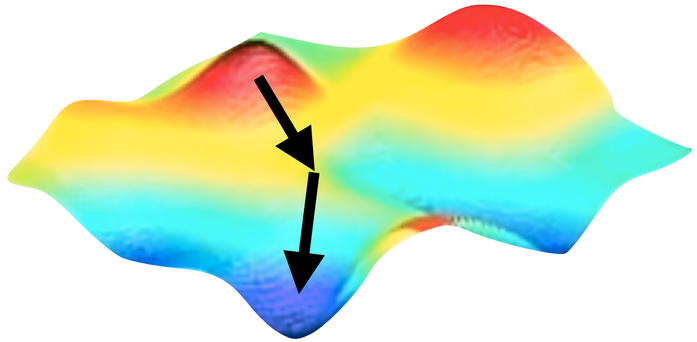
meta-objective $\mathcal{L}_{meta}$

multi vs single task



normal SGD
(back-propagation)

meta-learning

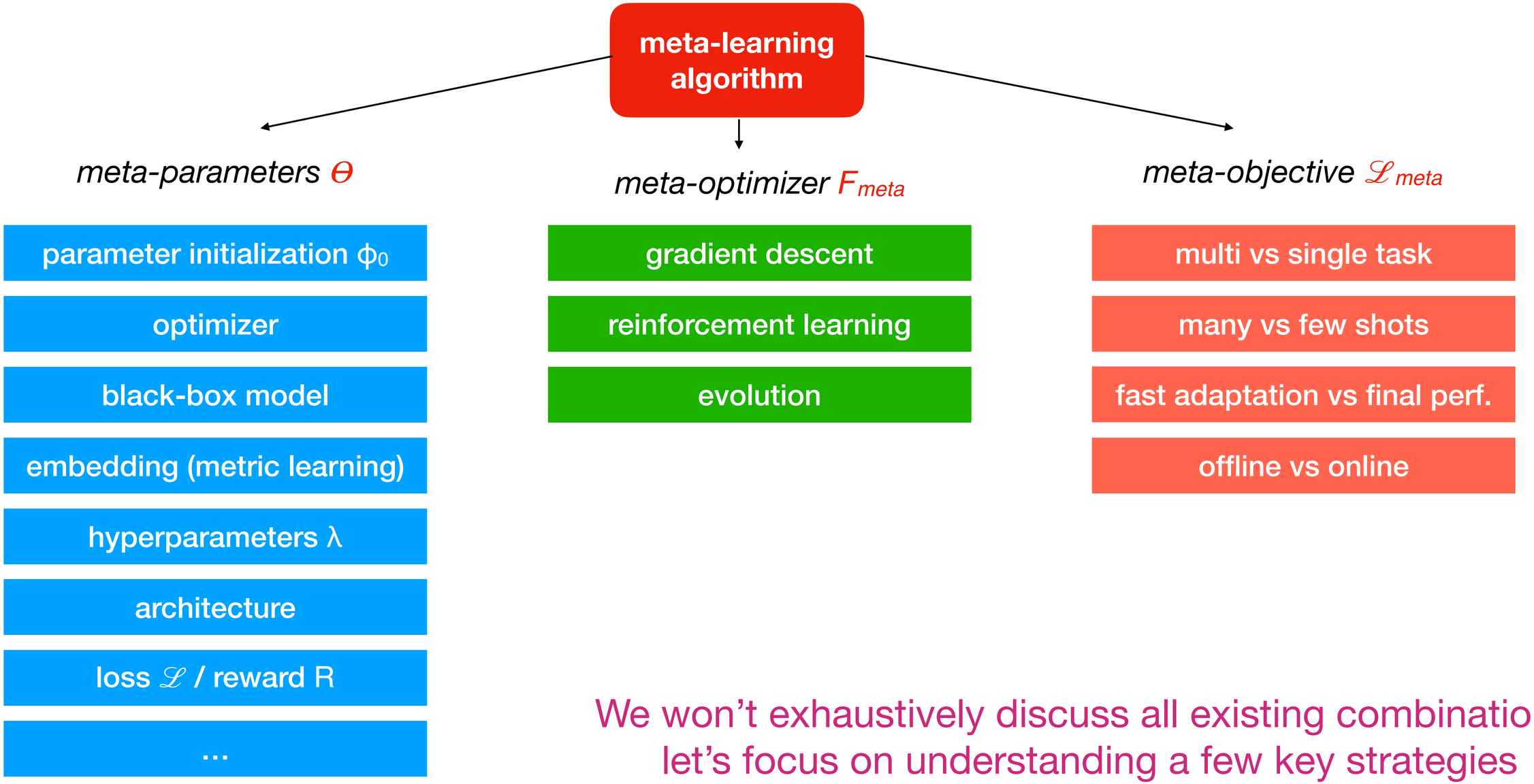


learned optimizer
(update rule)

Taxonomy of meta-learning methods

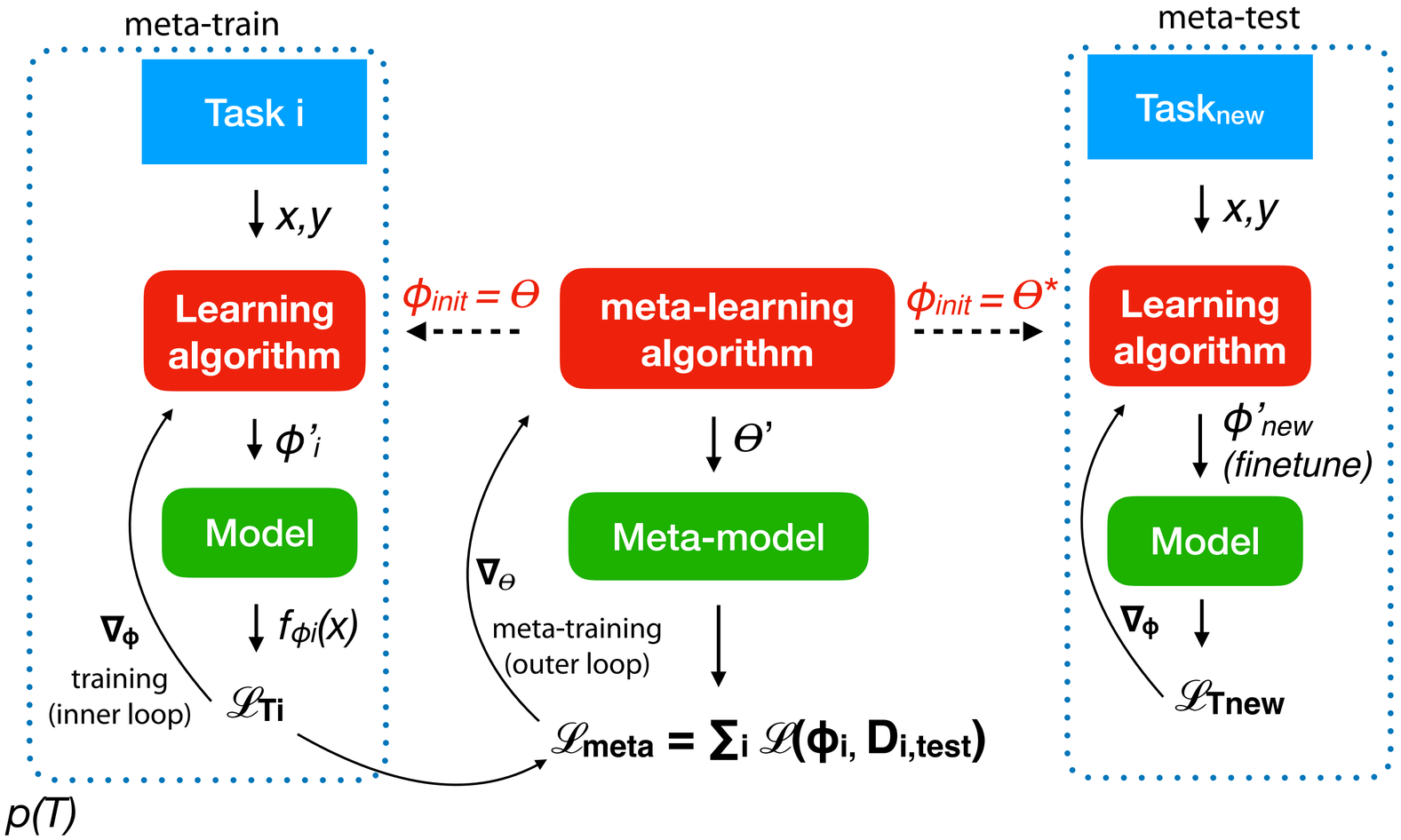
Hospedales et al. 2020
Huisman et al. 2020

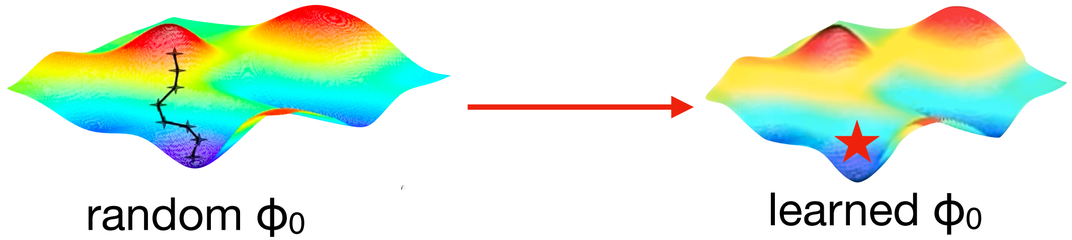
like base-learners, meta-learners consist of a representation, an objective, and an optimizer



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

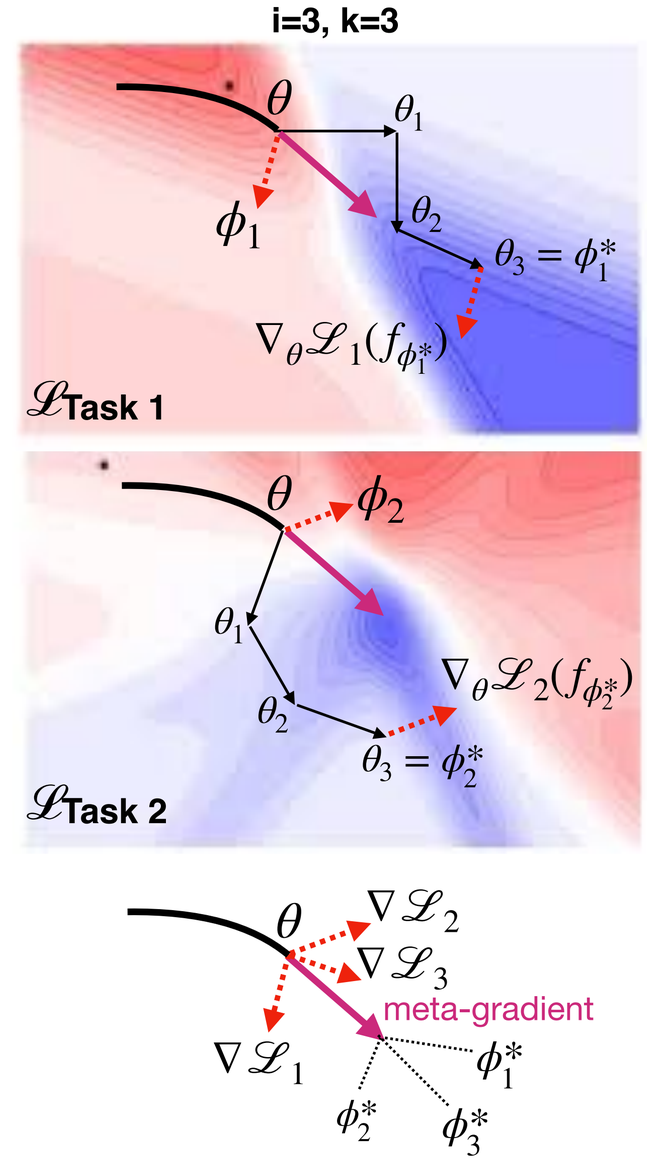
$$\theta \leftarrow \theta - \beta \nabla_\theta \sum_i \mathcal{L}_i(f_{\phi_i})$$

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

α, β: learning rates

meta-gradient: second-order gradient + backpropagate

compute how changes in Θ affect the gradient at new Θ

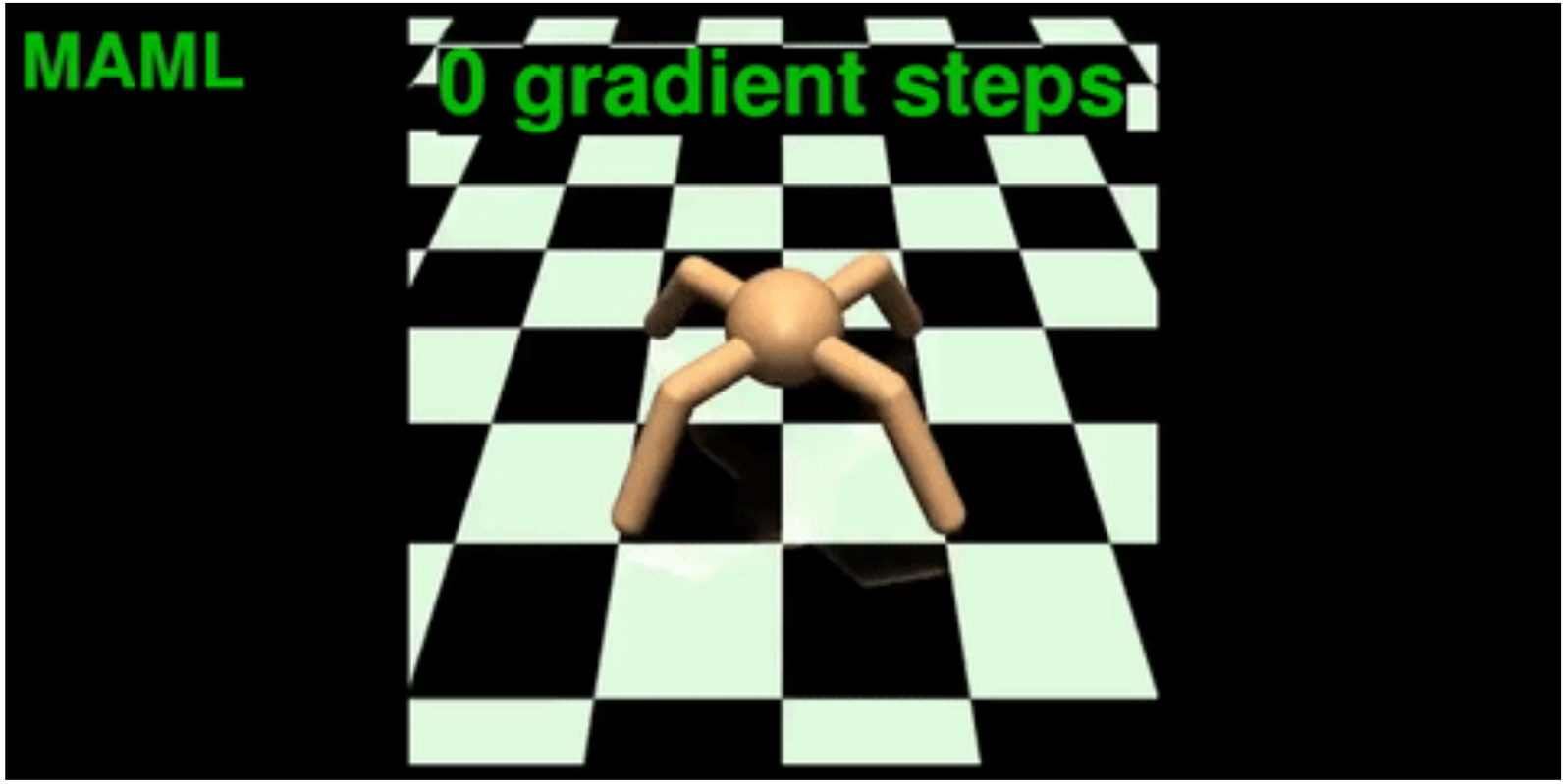


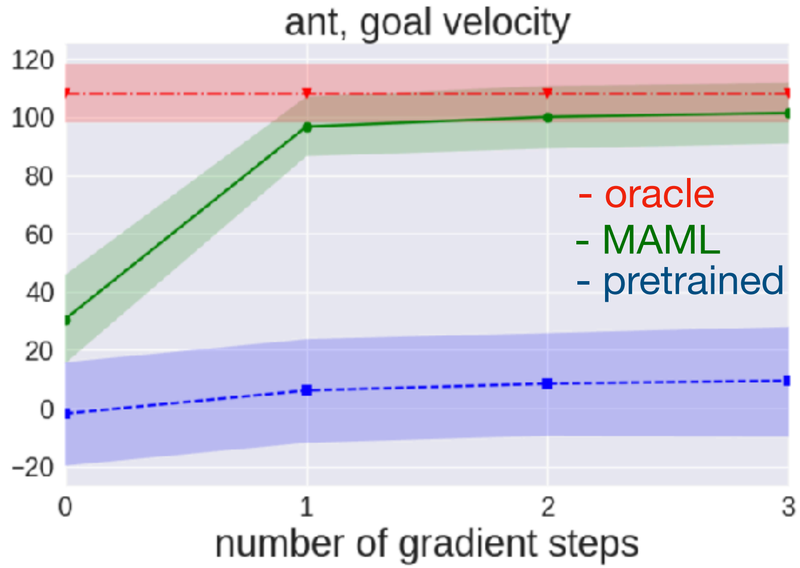
Meta-testing

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

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

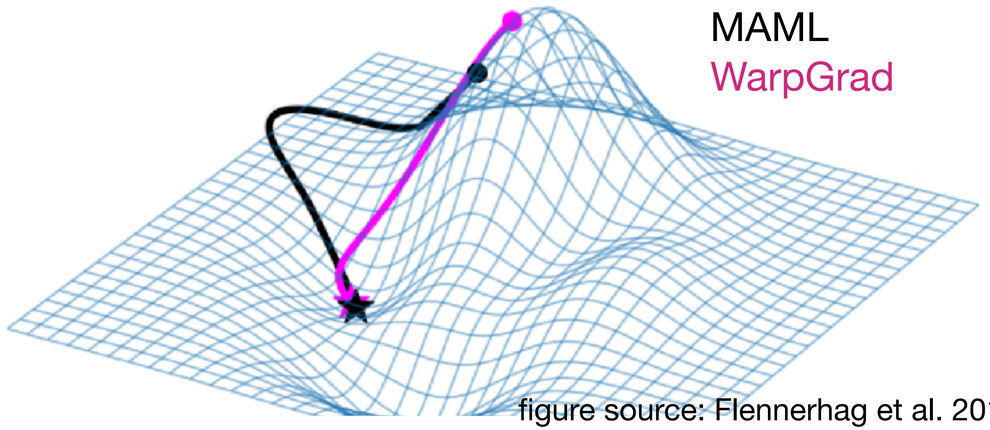


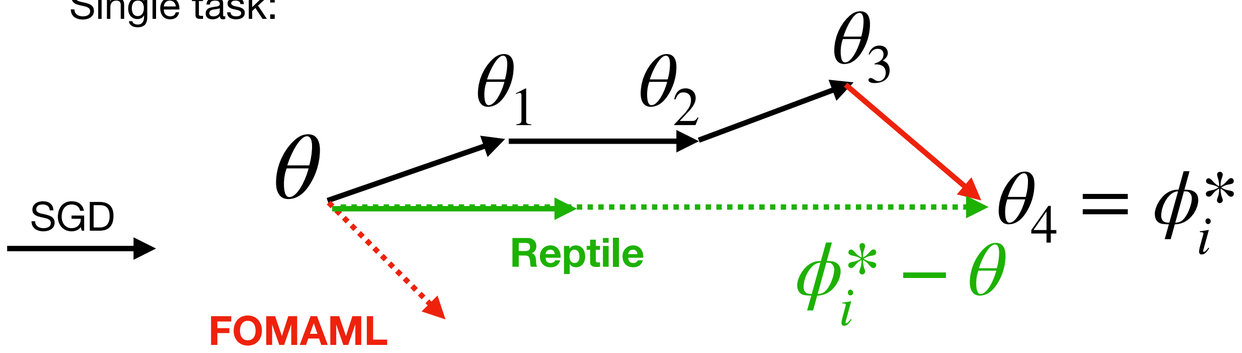
figure source: Flennerhag et al. 2019

Scalability

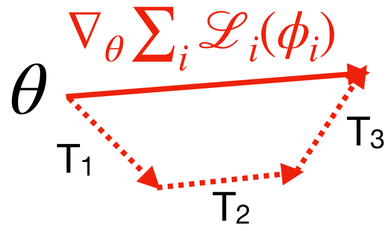
[1] Finn et al. 2017
[2] Nichol et al. 2018
[3] Rajeswaran et al. 2019

- 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

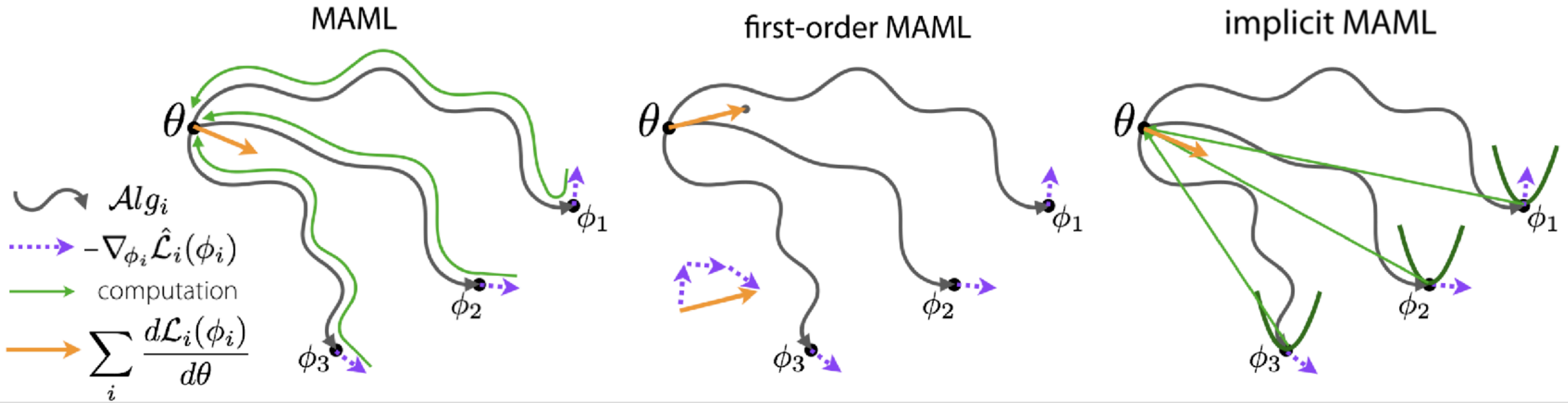
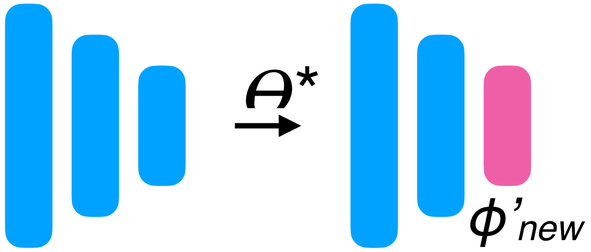


figure source: Rajeswaran et al. 2019

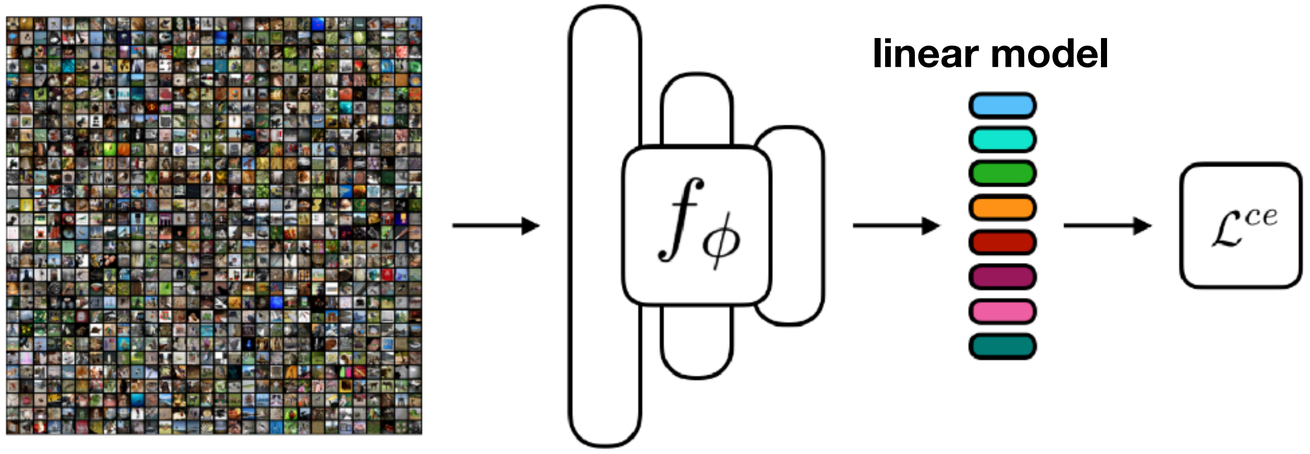
Generalizability

[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}



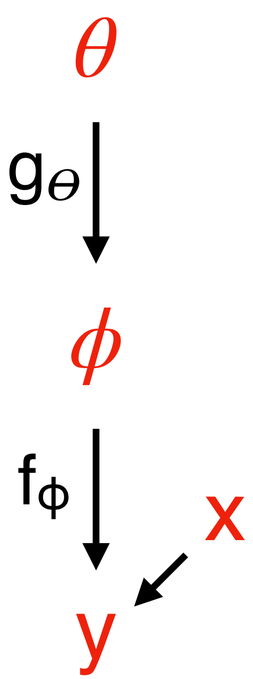
- For meta-RL, also learn how to explore (how to sample new environments)
  - E-MAML: add exploration to meta-objective (allows longer term goals) [4]

figure source: Tian et al. 2020

Bayesian meta-learning

Grant et al. 2018

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



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

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

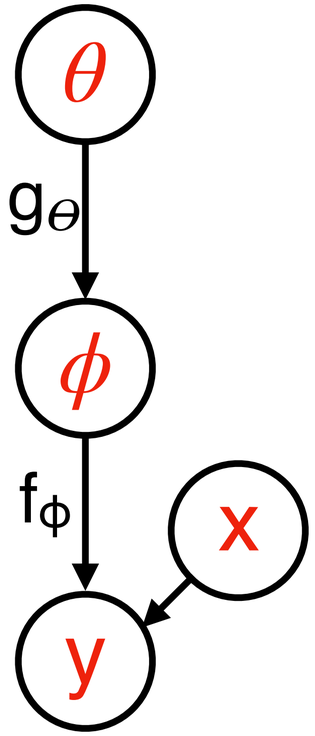
$$y_{test} \leftarrow f_{\hat{\phi}}(x_{test})$$

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

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

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

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



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

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

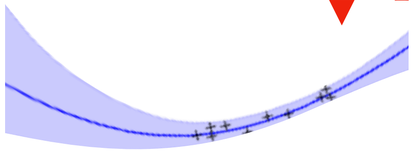
$$p(y_{test} \mid x_{test})$$

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

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

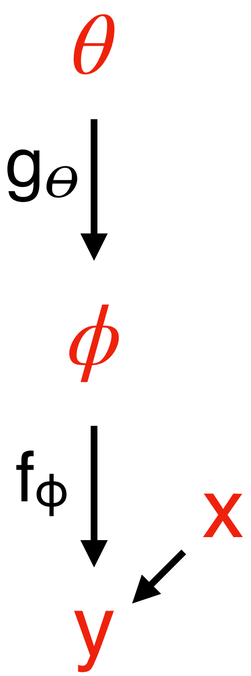
$$p(\phi_j \mid (x,y)_{train}, \theta)$$ *finetuning*

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



Bayesian meta-learning

Grant et al. 2018



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

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

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

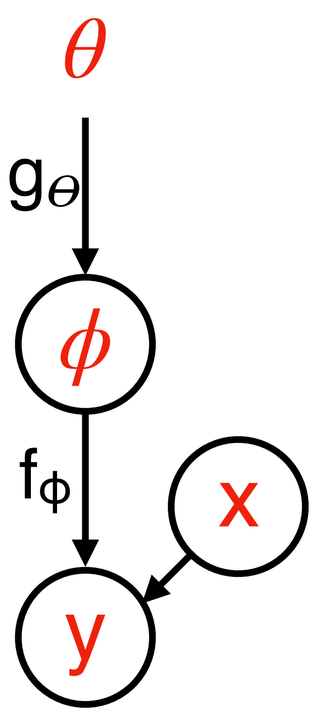
$$\theta \leftarrow \theta - \nabla_\theta \mathcal{L}(y_{test})$$

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

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

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

Empirical Bayes: fit point estimate on Θ



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

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

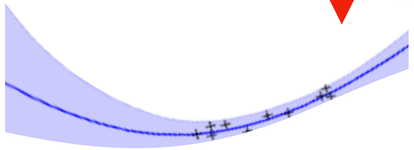
$$p(y_{test} \mid x_{test})$$

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

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

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

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



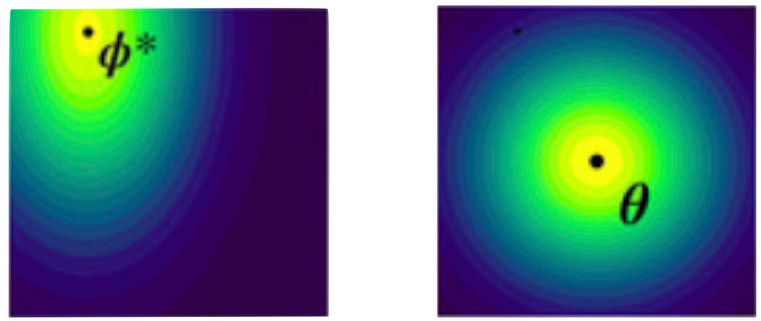
Bayesian meta-learning

[1] Grant et al. 2018

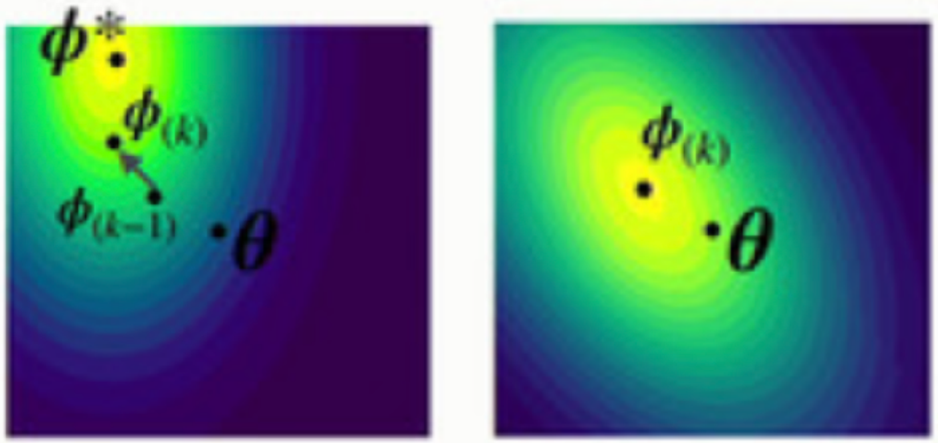
ϕ can be factored as the product of a likelihood and a prior

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

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



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

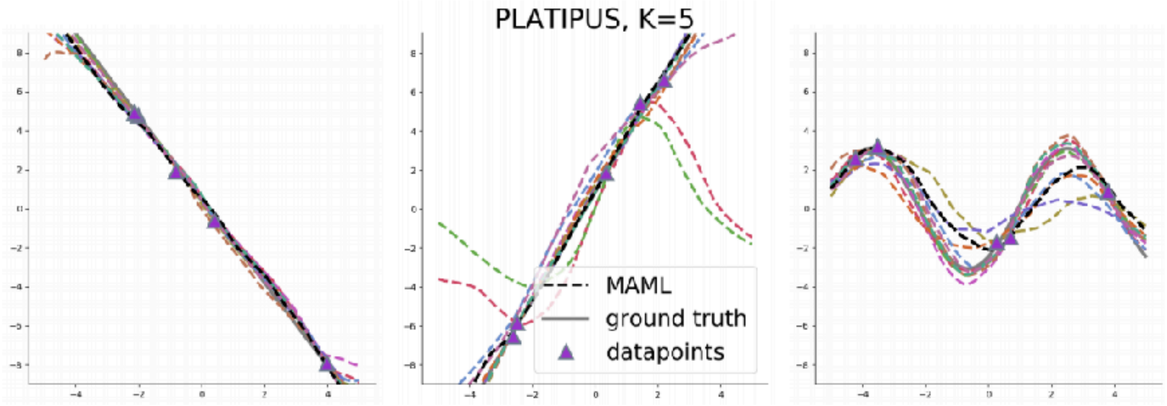
MAML is doing the same

- 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

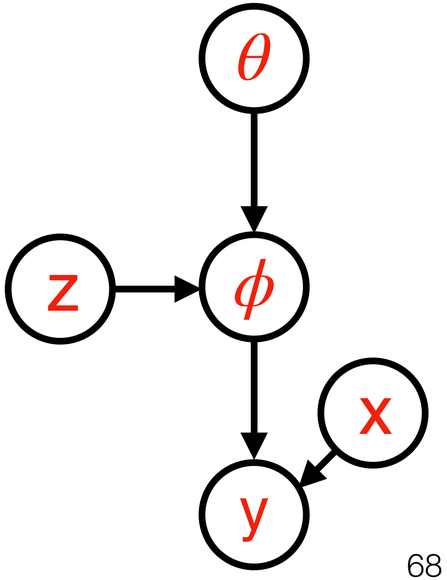
Fully Bayesian meta-learning

- 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:

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

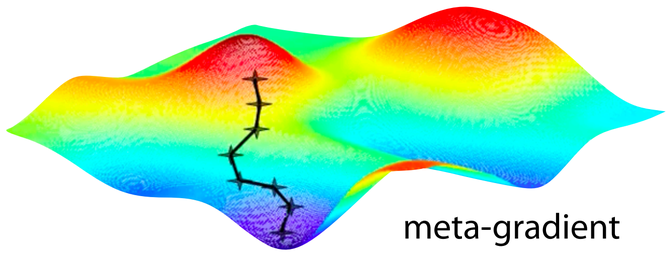
learning rate

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]

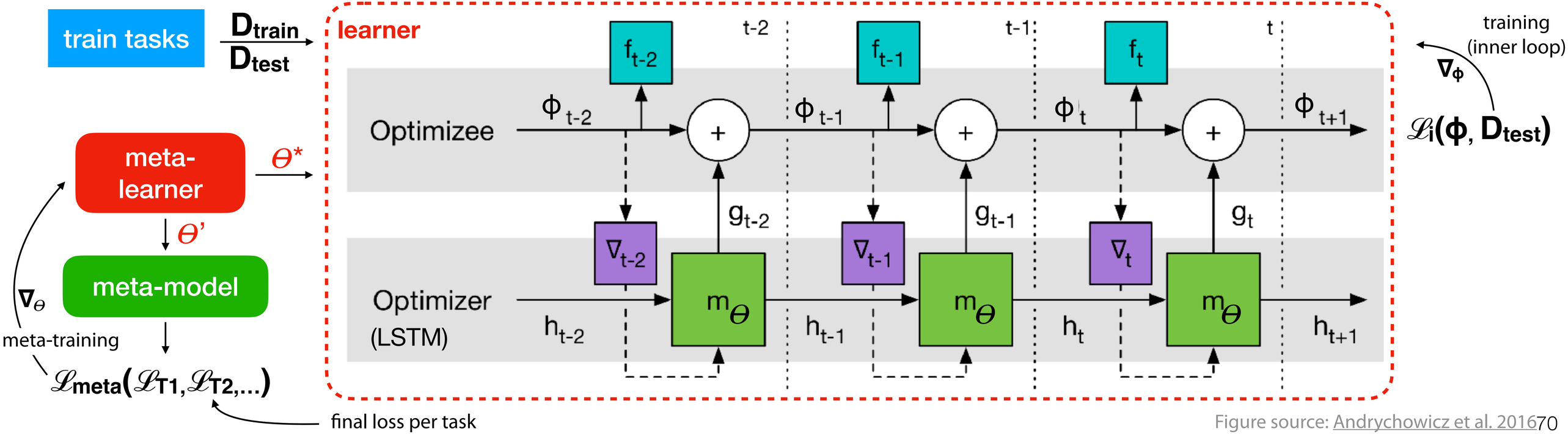


Figure source: Andrychowicz et al. 2016

Meta-learning optimizers

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

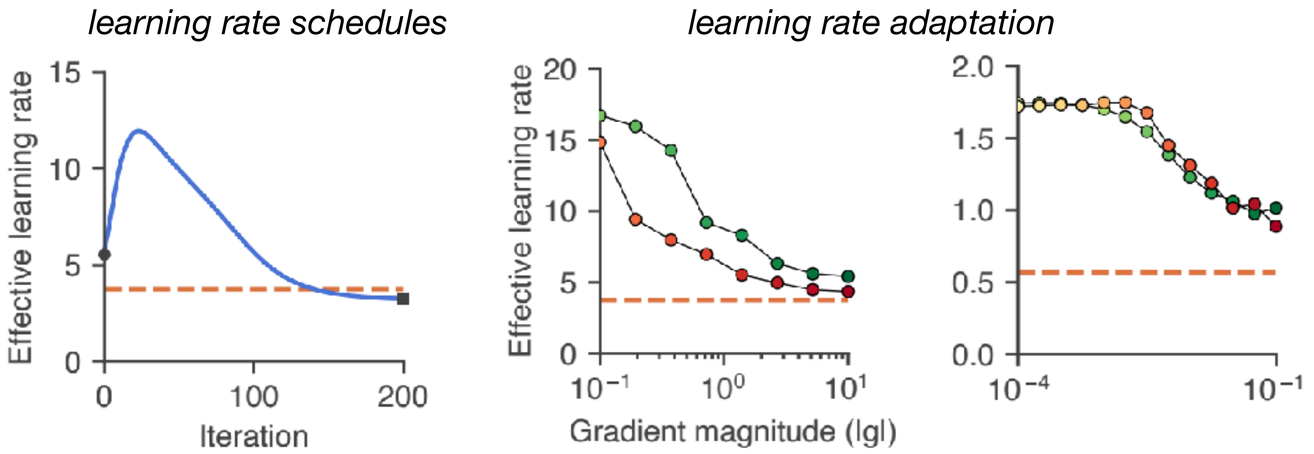


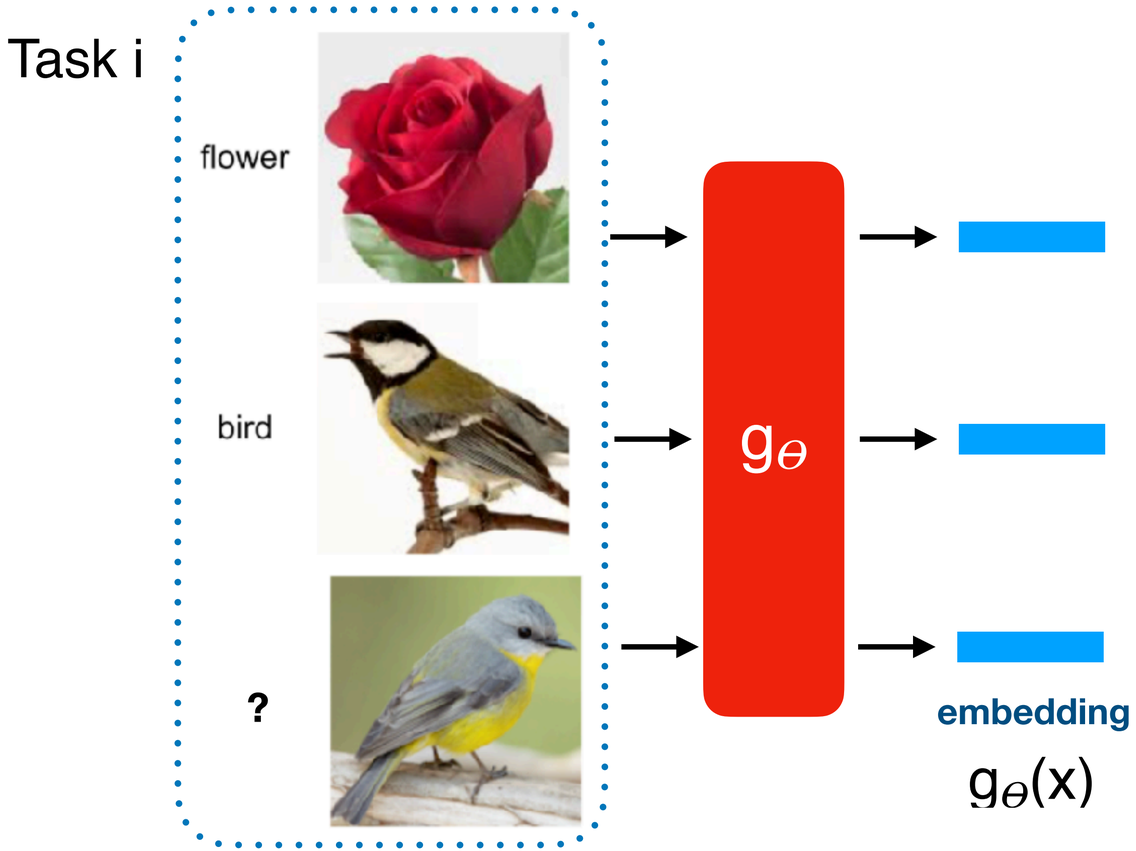
Figure source: Maheswaranathan et al. 2020

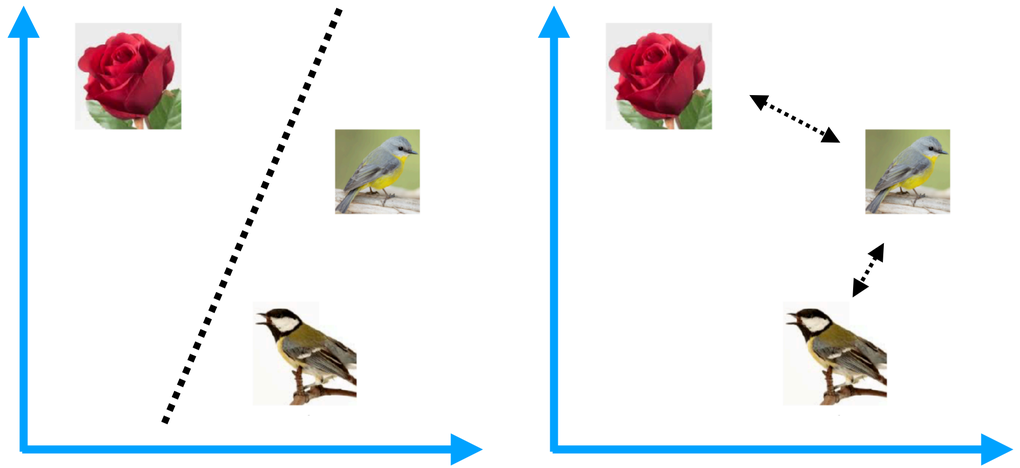
- RL-based optimizers: represent updates as a policy, learn using guided policy search [2]
- Combined with MAML:
  - learn per-parameter learning rates [3,4]
  - learn precondition matrix (to `warp' the loss surface) [5,6]
- Black-box optimizers: meta-learned with an RNN [7], or with user-defined priors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

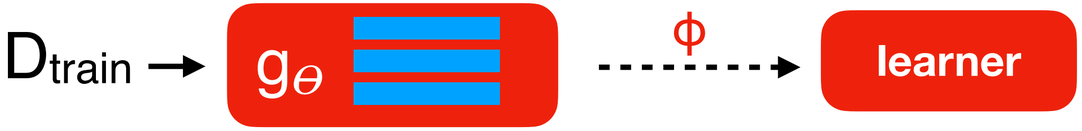
Learn an embedding network Θ that transforms data $\{D_{train}, D_{test}\}$ across all tasks to a representation that allows easy similarity comparison





Can be seen as a simple black box meta-model

- Often non-parametric (independent of ϕ)



Vinyals et al. 2016

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

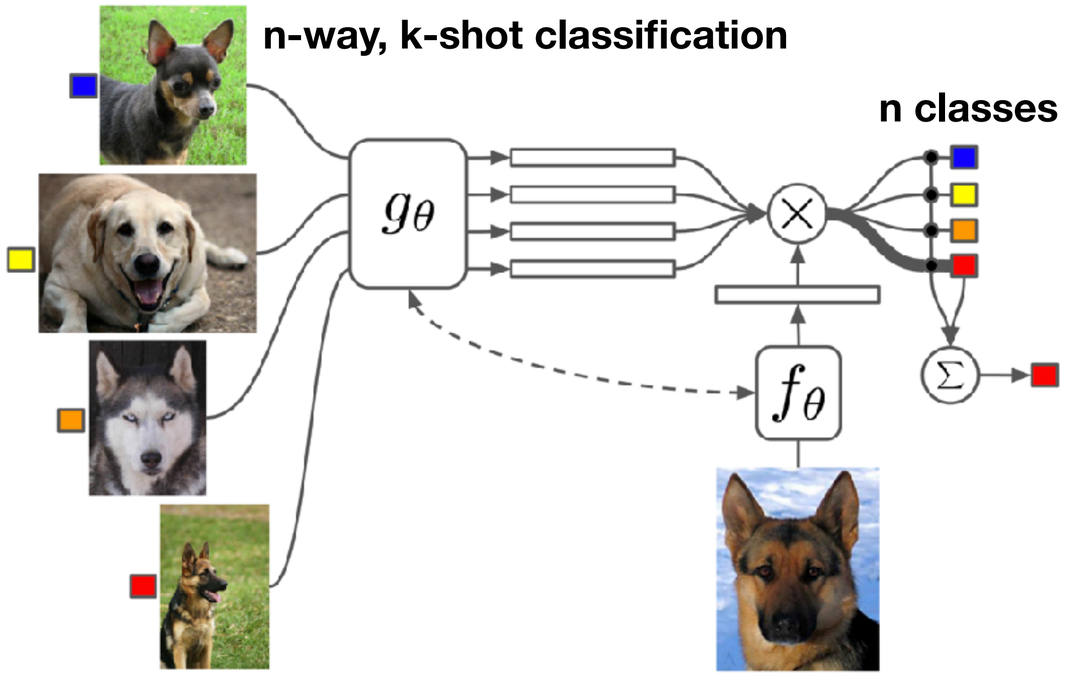


Figure source: Vinyals et al. 2016

Snell et al. 2017

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:

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

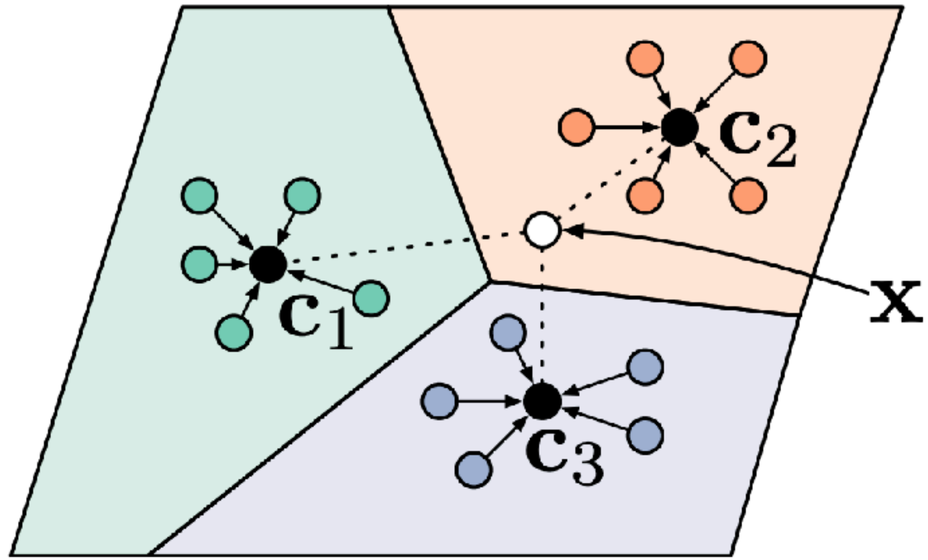


Figure source: Snell et al. 2017

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

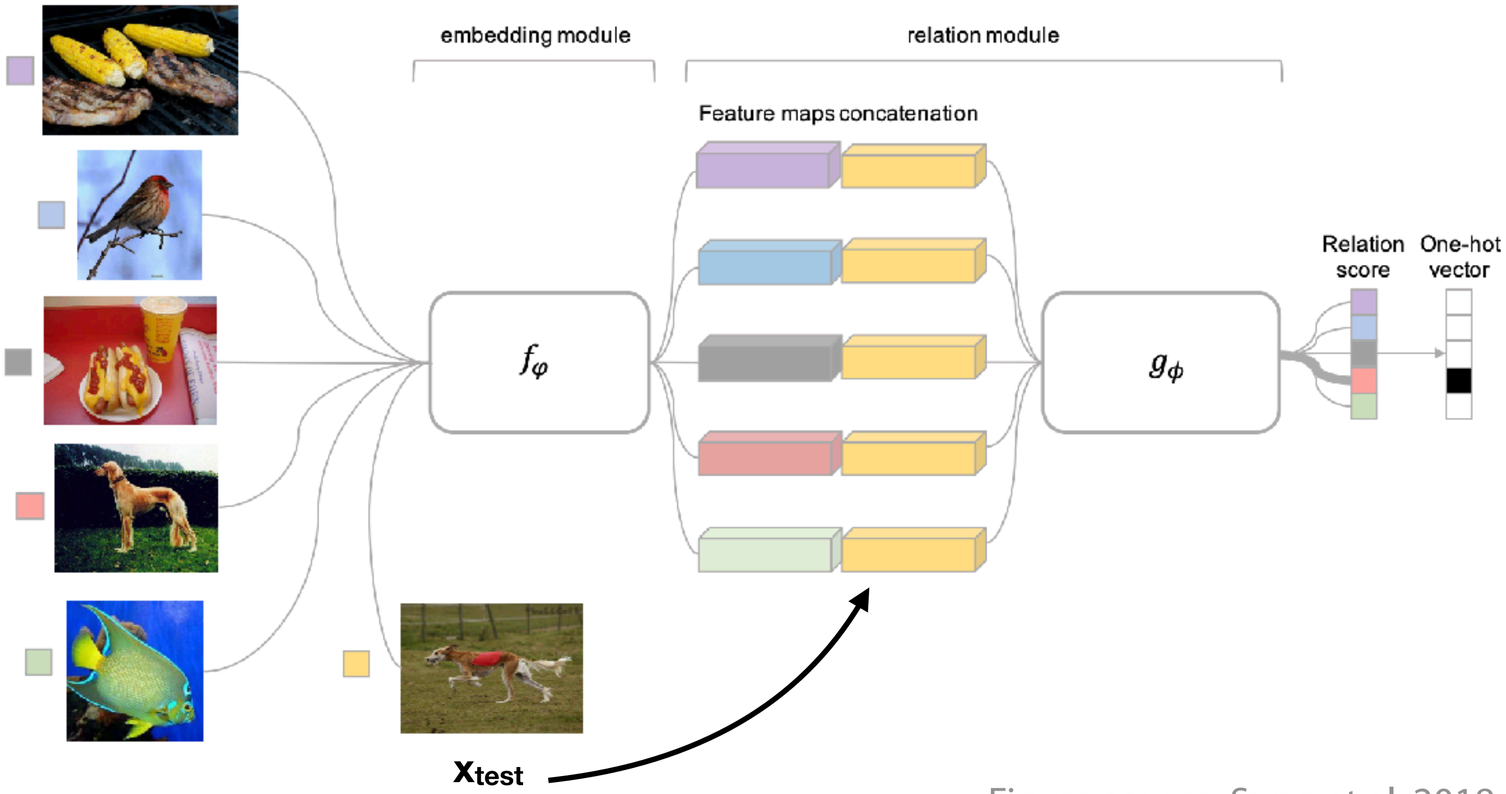


Figure source: Sung et al. 2018

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

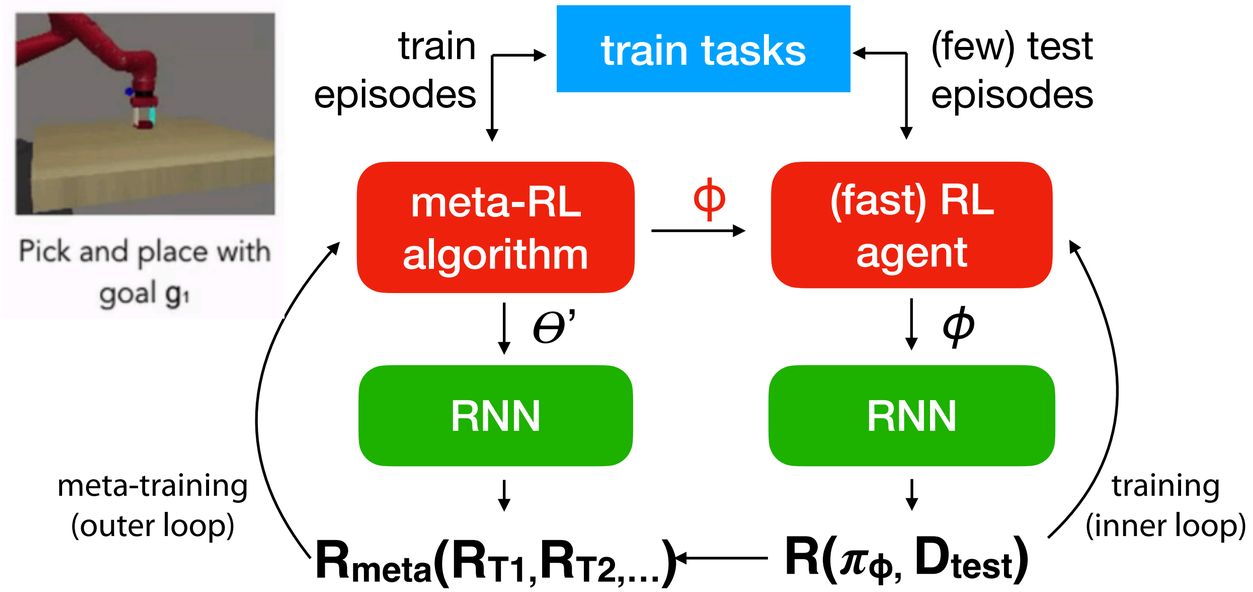
Shyam et al. 2017

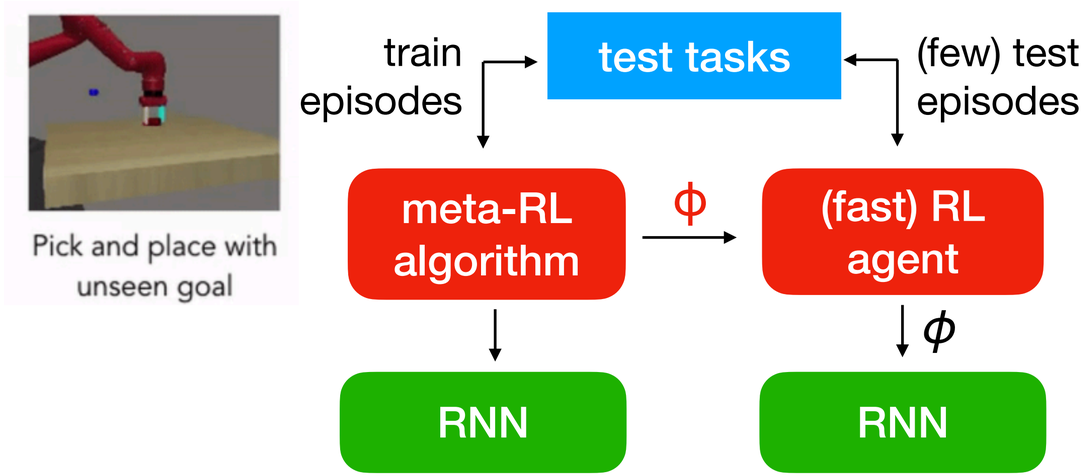
Lee et al. 2019

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

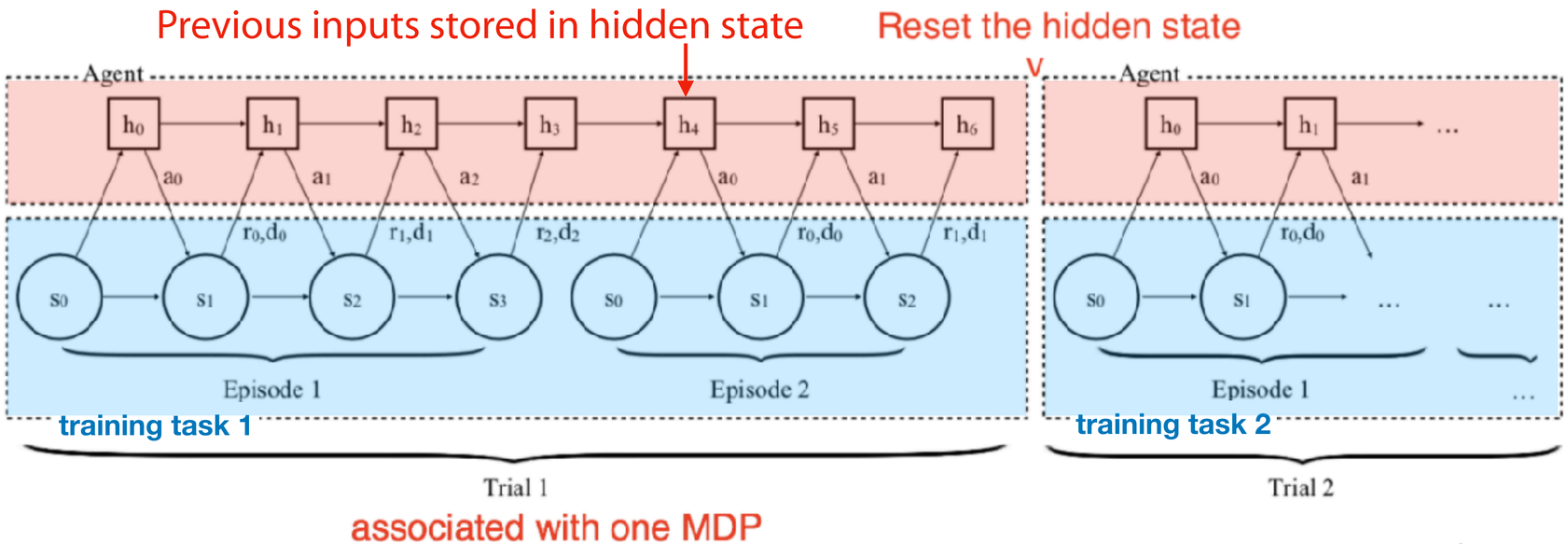
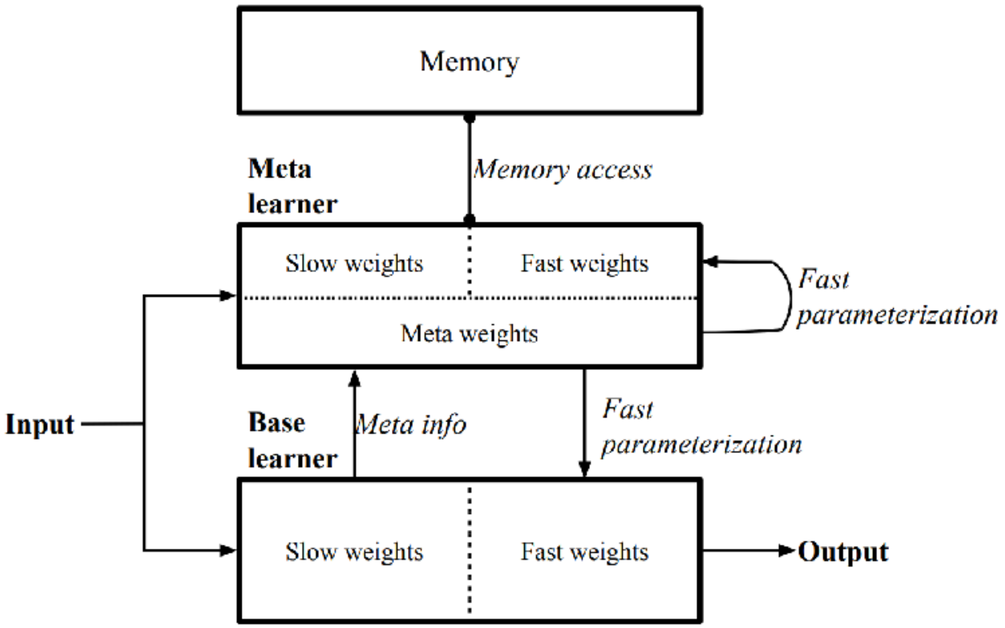


Image source: Duan et al. 2018

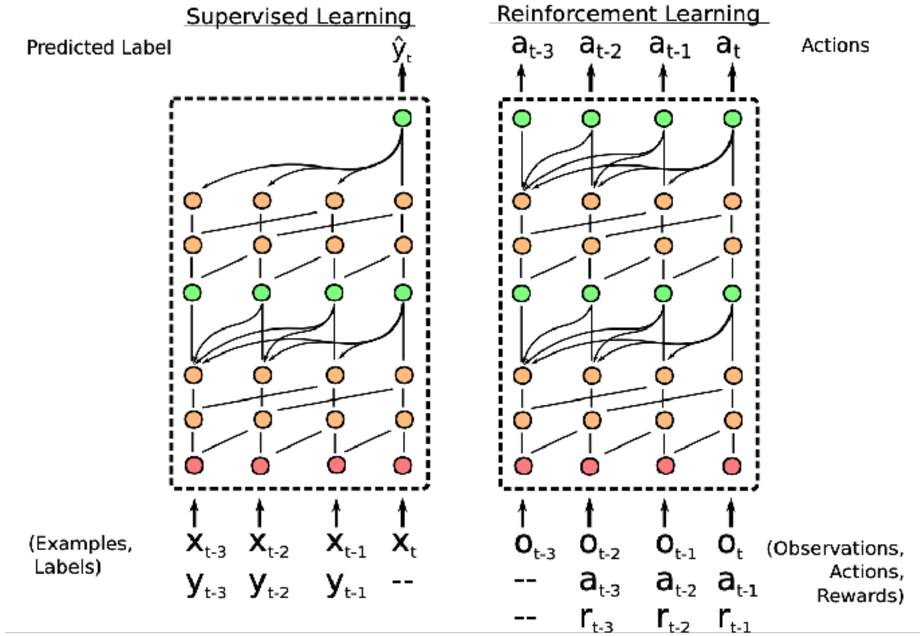
Other black-box models

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

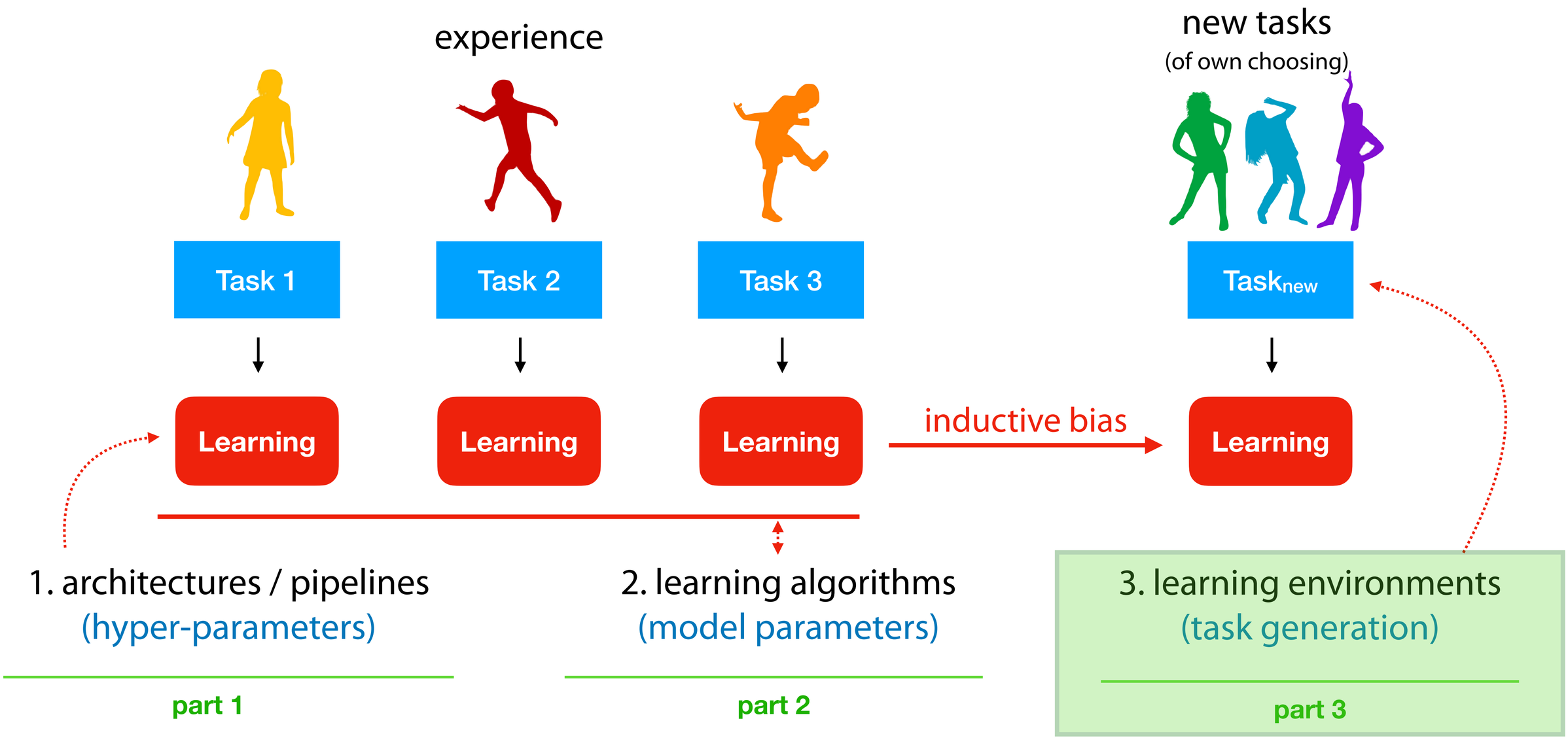


SNAIL. Image source: Mishra et al. 2018

Clune 2019

What can we learn to learn?

3 *pillars*



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

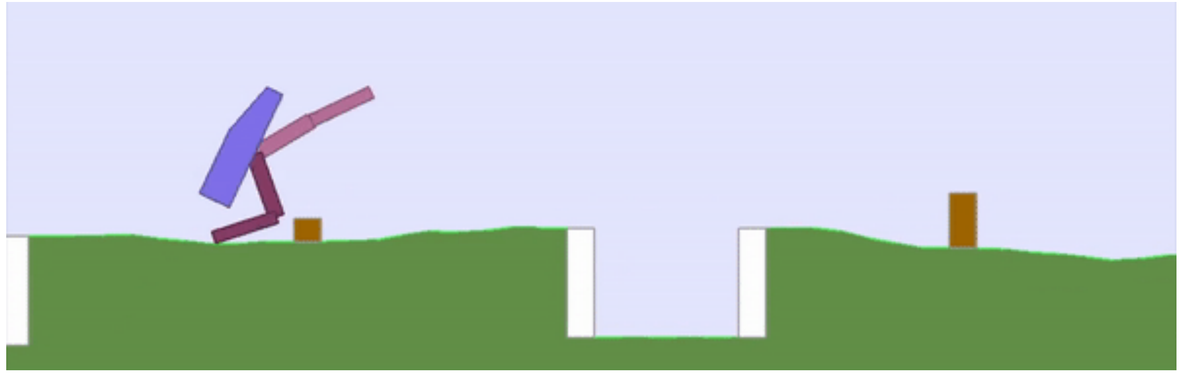
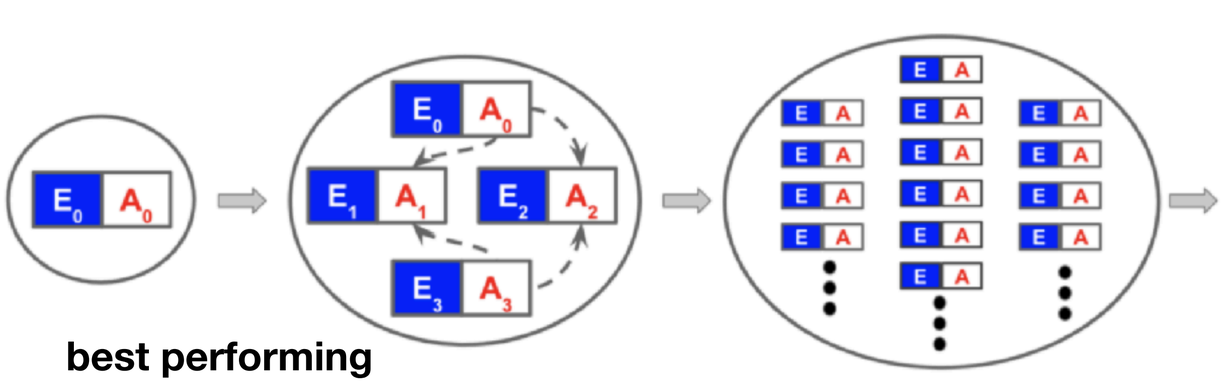


Figure source: Wang et al. 2019



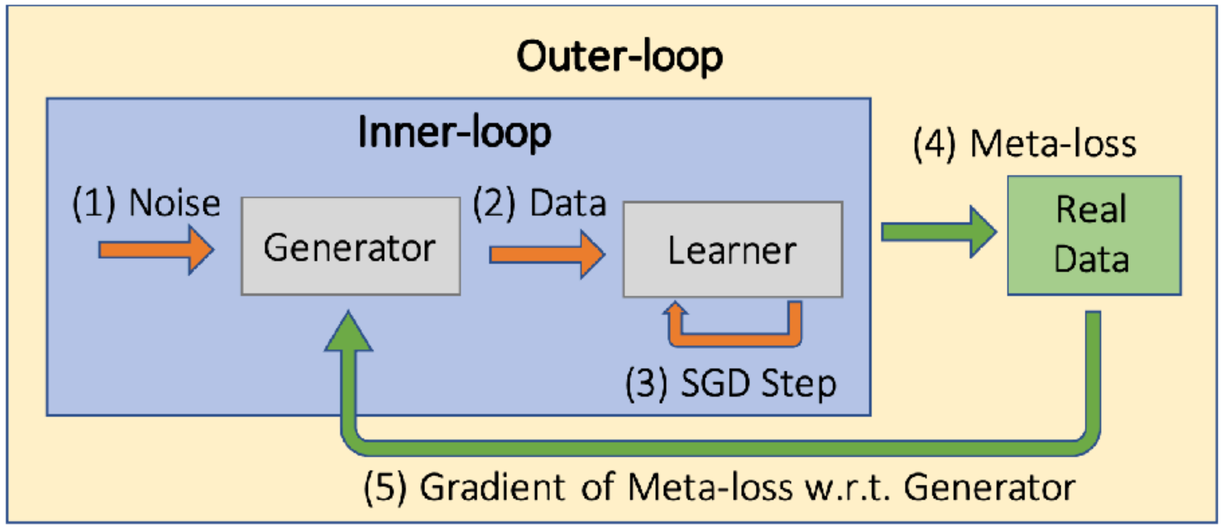
Does POET scale? Increasingly difficult 3D terrain, 18 degrees of freedom.

Zhou & Vanschoren. 2021

Generative Teaching Networks

Such et al. 2019

- Based on an existing dataset, generate synthetic training data for more efficient training
  - Like dataset distillation, but uses meta-learning to update the generator model
- While POET has limited expressivity (limited to θ_E), GTNs could produce all sorts of training datasets and environments
    - While being careful not to generate noise: needs some grounding in reality



Thank you

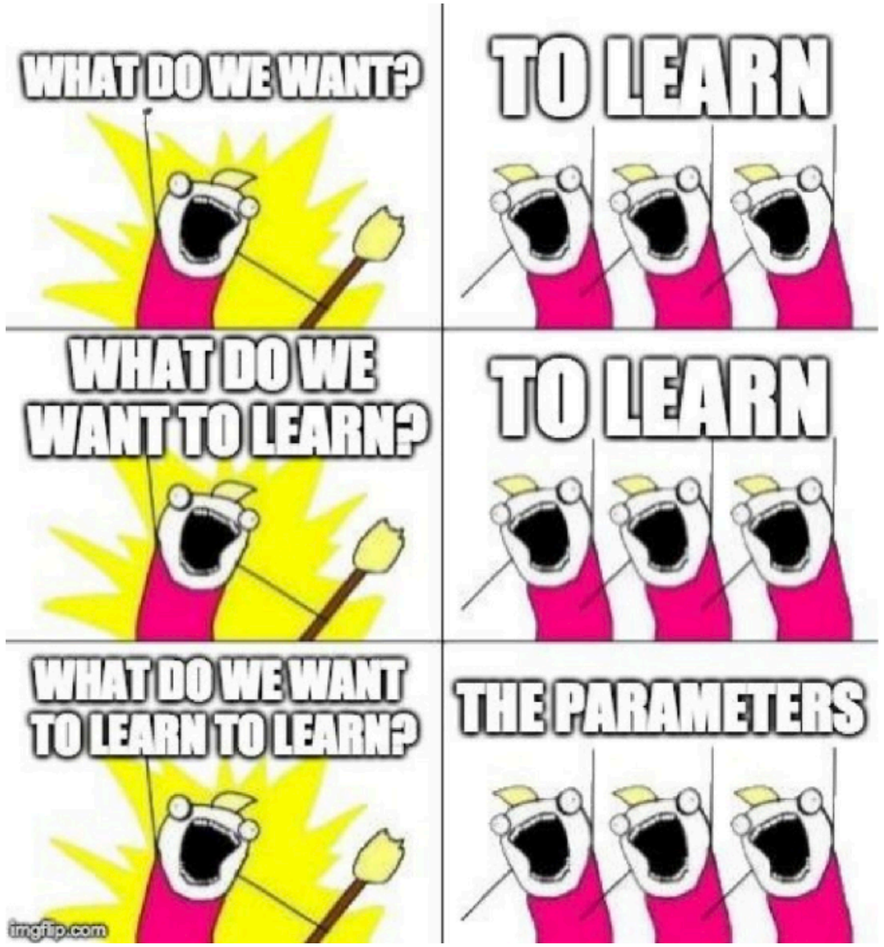


Image source: Pesah et al. 2018