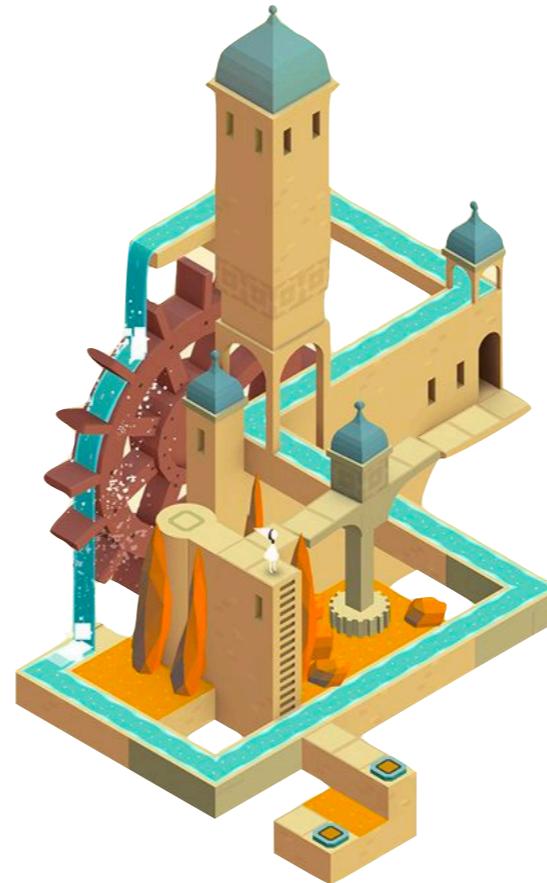


Advanced Course on Data Science & Machine Learning

Siena, 2019



Automatic Machine Learning & Meta-Learning

part 3

j.vanschoren@tue.nl

Joaquin Vanschoren
Eindhoven University of Technology
[@joavanschoren](https://twitter.com/joavanschoren)

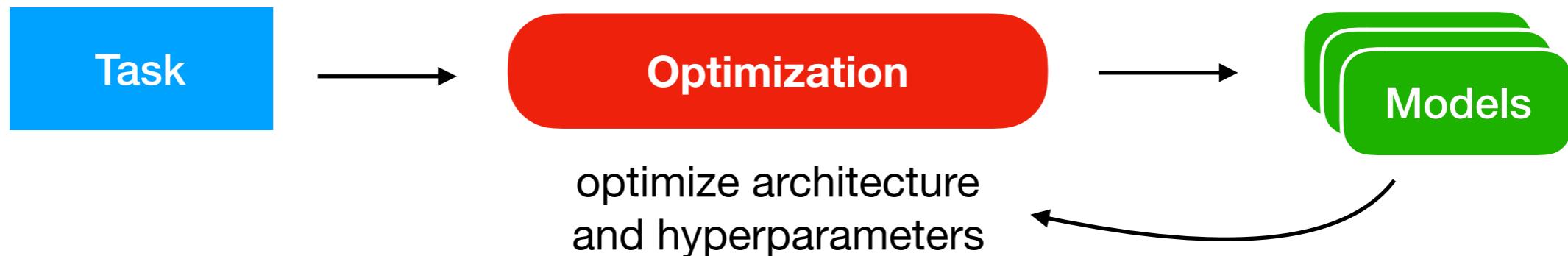


<http://bit.ly/openml>

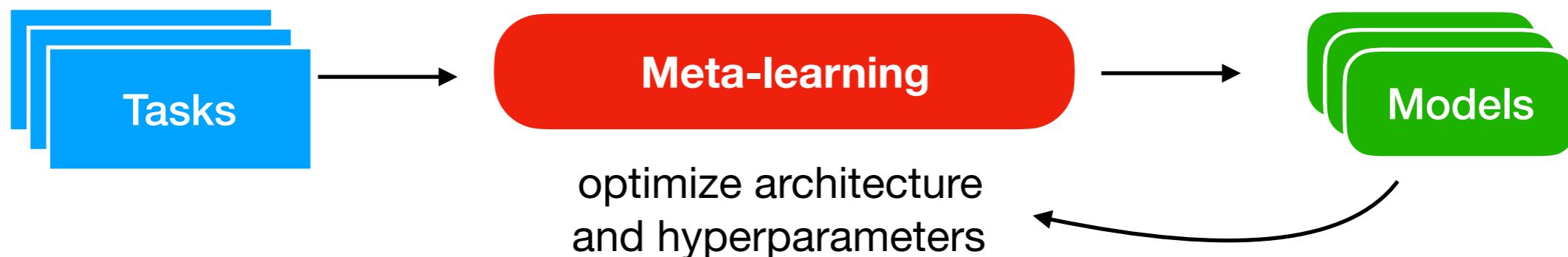
slides!

Overview

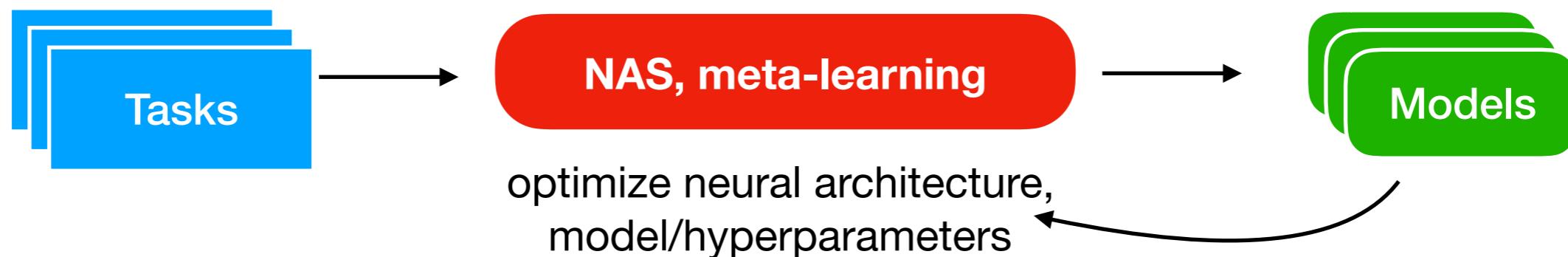
part I AutoML introduction, promising optimization techniques



part 2 Meta-learning

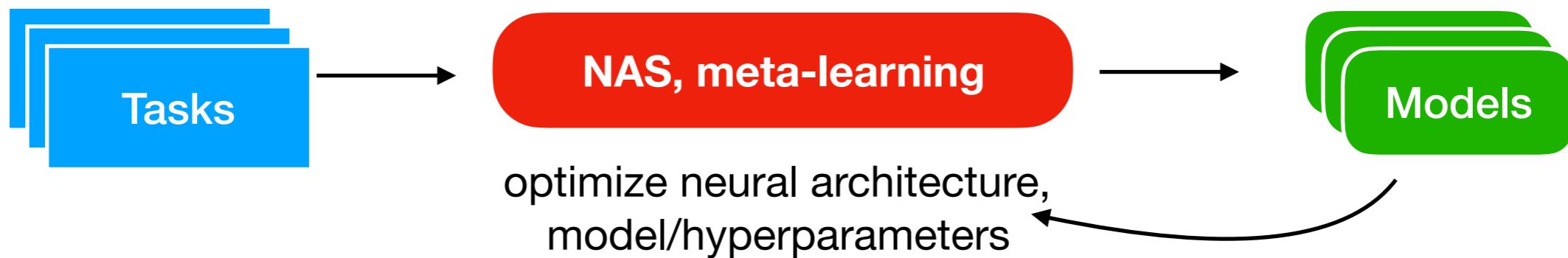


part 3 Neural architecture search, meta-learning on neural nets



Overview

part 3 Neural architecture search, meta-learning on neural nets



1. Neural Architecture Search space
2. Neural Architecture optimization
3. (Neural) Meta-Learning

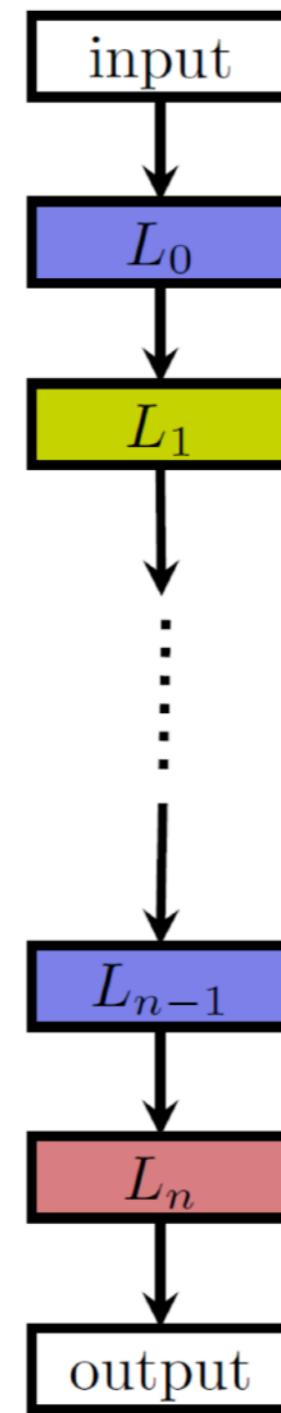
Neural Architecture Search space

Sequential (chain)

Choose:

- number of layers
- type of layers
 - dense
 - convolutional
 - max-pooling
 - ...
- hyperparameters of layers

+ easier to search
- sometimes too simple

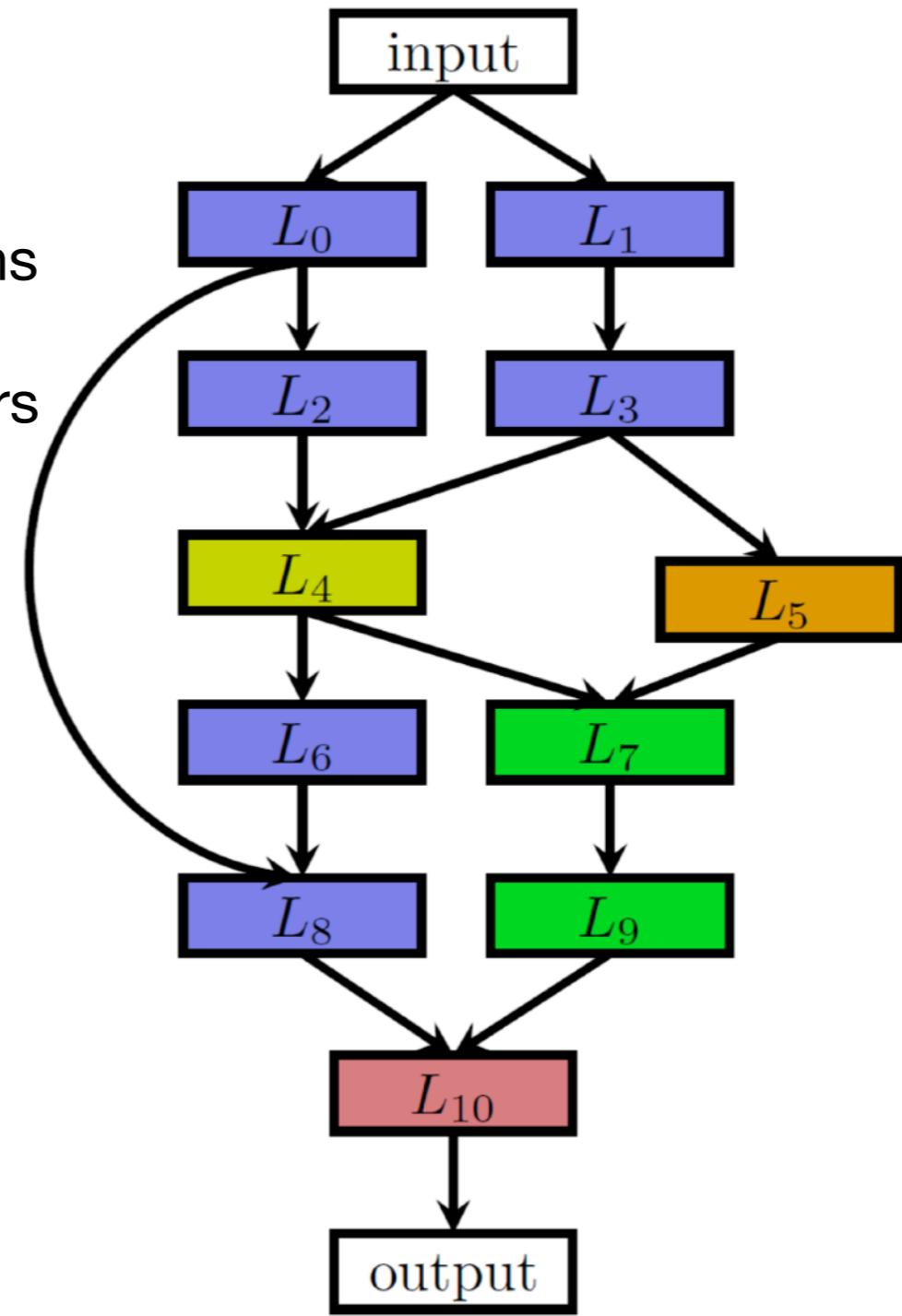


Arbitrary graph

Choose:

- branching
- joins
- skip connections
- types of layers
- hyperparameters of layers

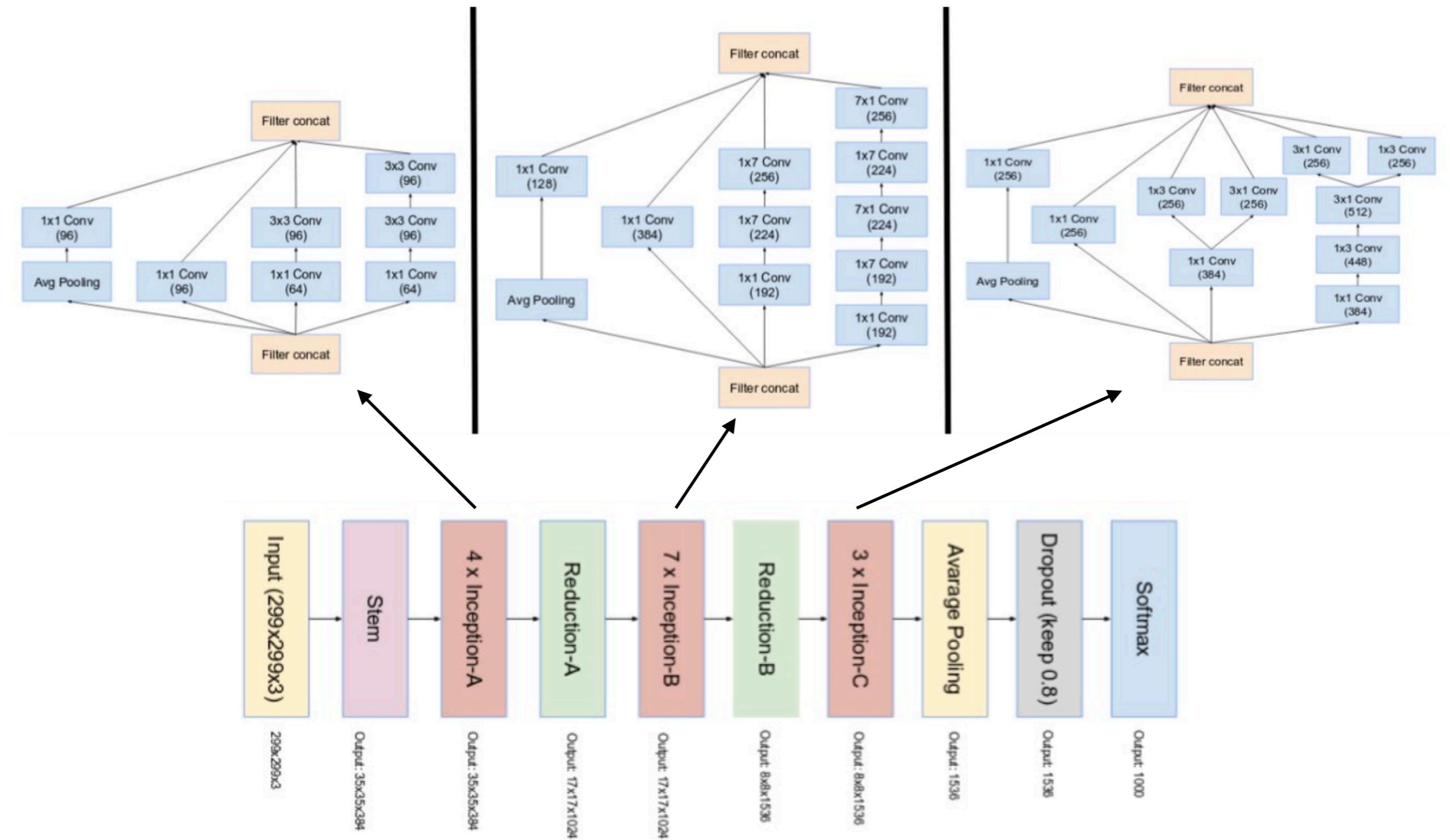
+ more flexible
- much harder to search



Neural Architecture Search space

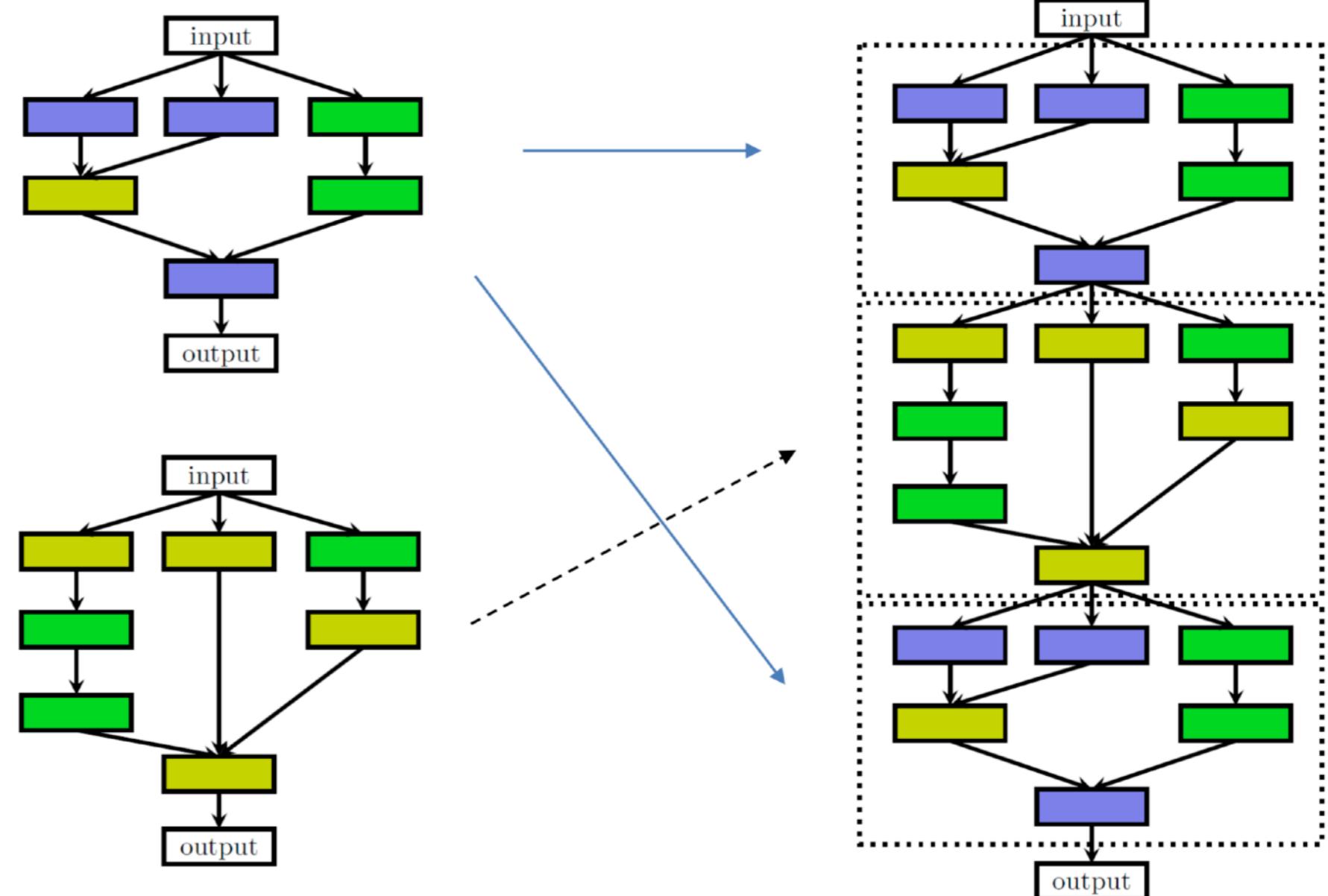
Observation: successful deep networks have repeated motifs (cells)

e.g. Inception v4:



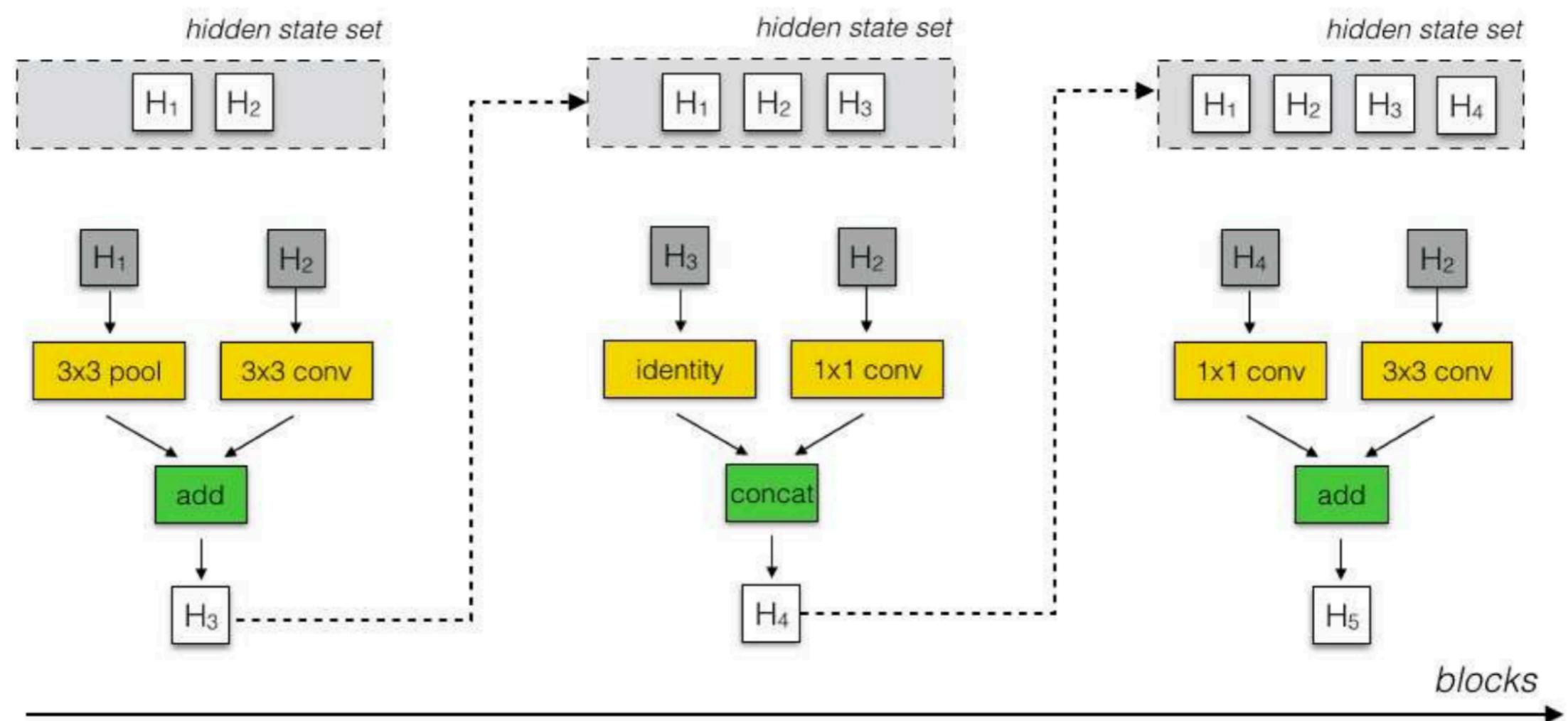
Cell search space

- parameterize building blocks (*cells*)
 - e.g. *regular cell* + *resolution reduction cell*
- stack cells together in macro-architecture
 - usually a chain
 - can be manual or learned
- + smaller search space
- + cells can be learned on a small dataset & transferred to a larger dataset
- you can't learn entirely new architectures



Within-cell search space

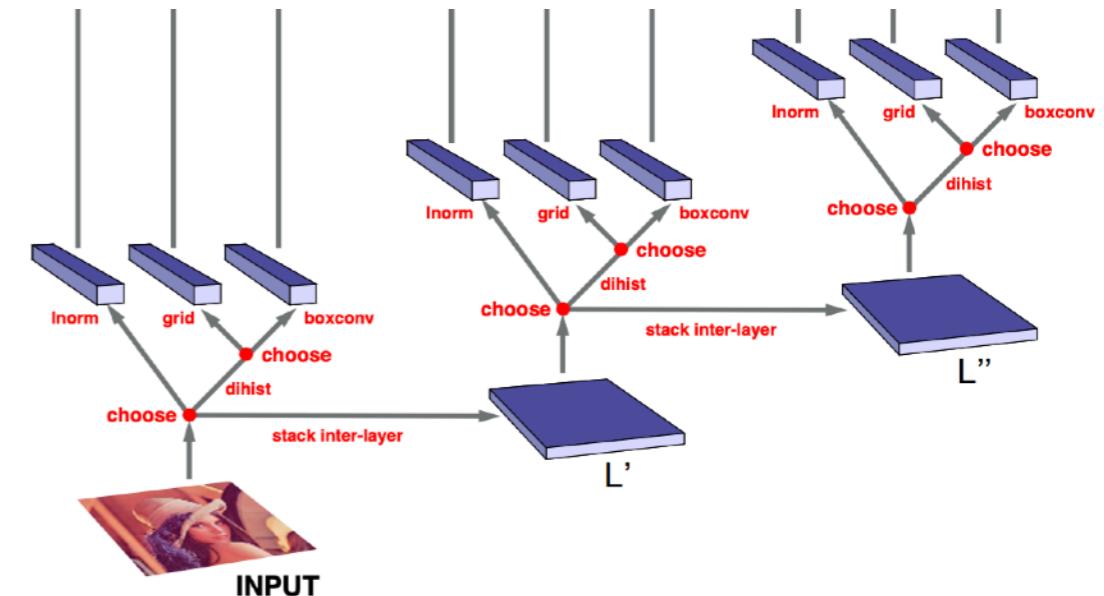
- Different *strategies* to construct cell, leading to different parameterizations
- Can be parameterized as set of categorical hyperparameters:
 - Which existing layer (hidden state, e.g. cell input) H_i to build on
 - Which operation (e.g. 3x3conv) to perform on H_i
 - How to combine into new hidden state
 - Iterate over B phases (blocks)



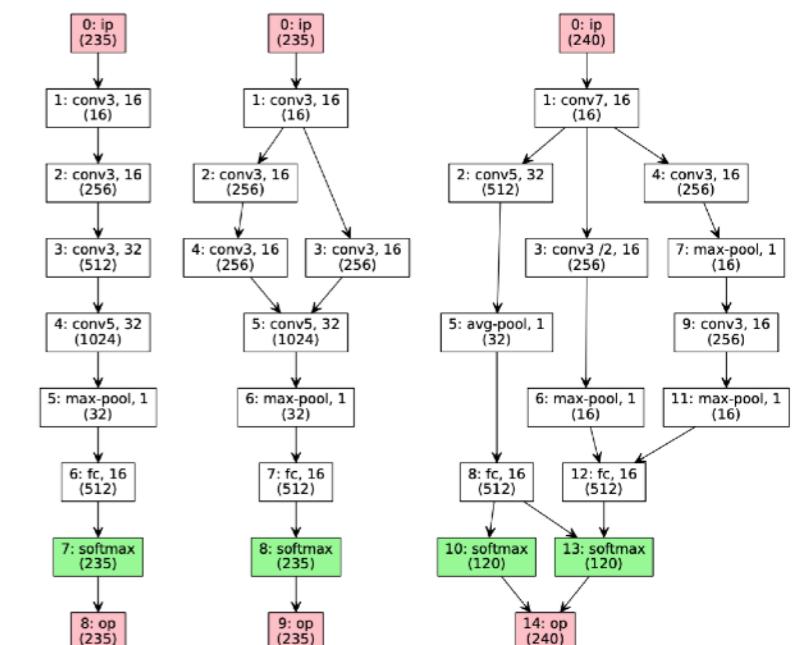
Neural architecture optimization

Standard hyperparameter optimization techniques

- Image classification pipelines¹
 - 238 hyperparameters, tuned with TPE



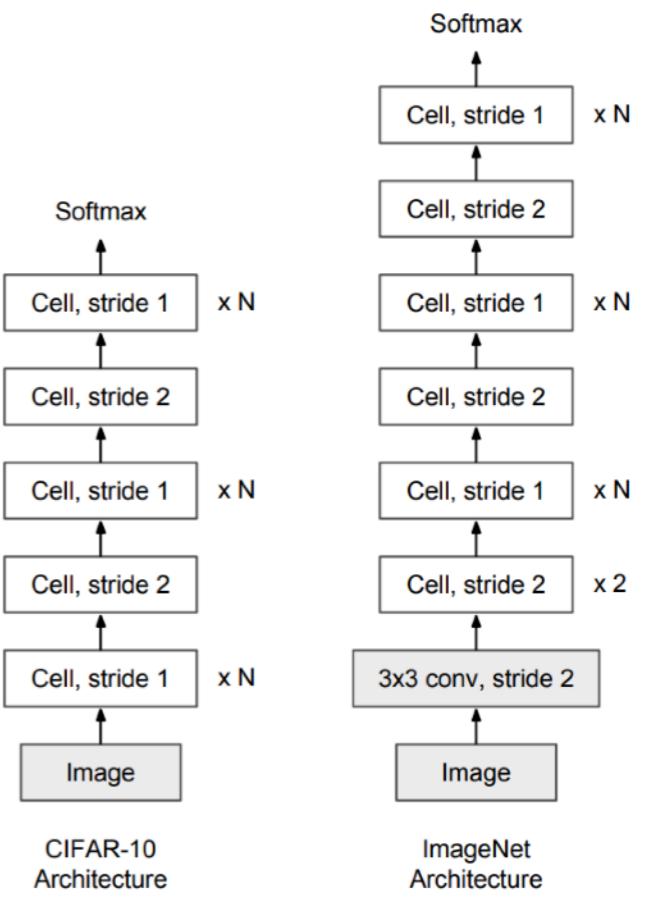
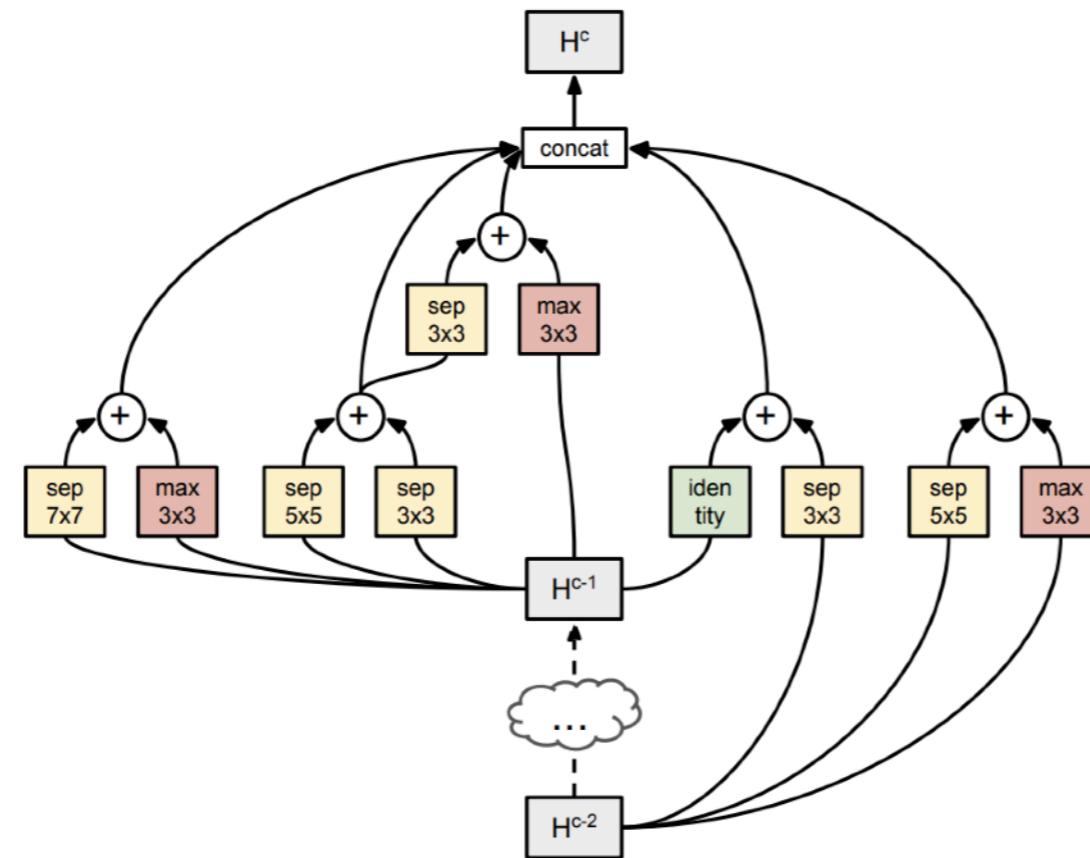
- Kernels to optimize neural nets with GP-based Bayesian optimization
 - ArcNet²: DNNs, 23 hyperparameters, 6 kernels
 - NASBOT³: DNNs and CNNs



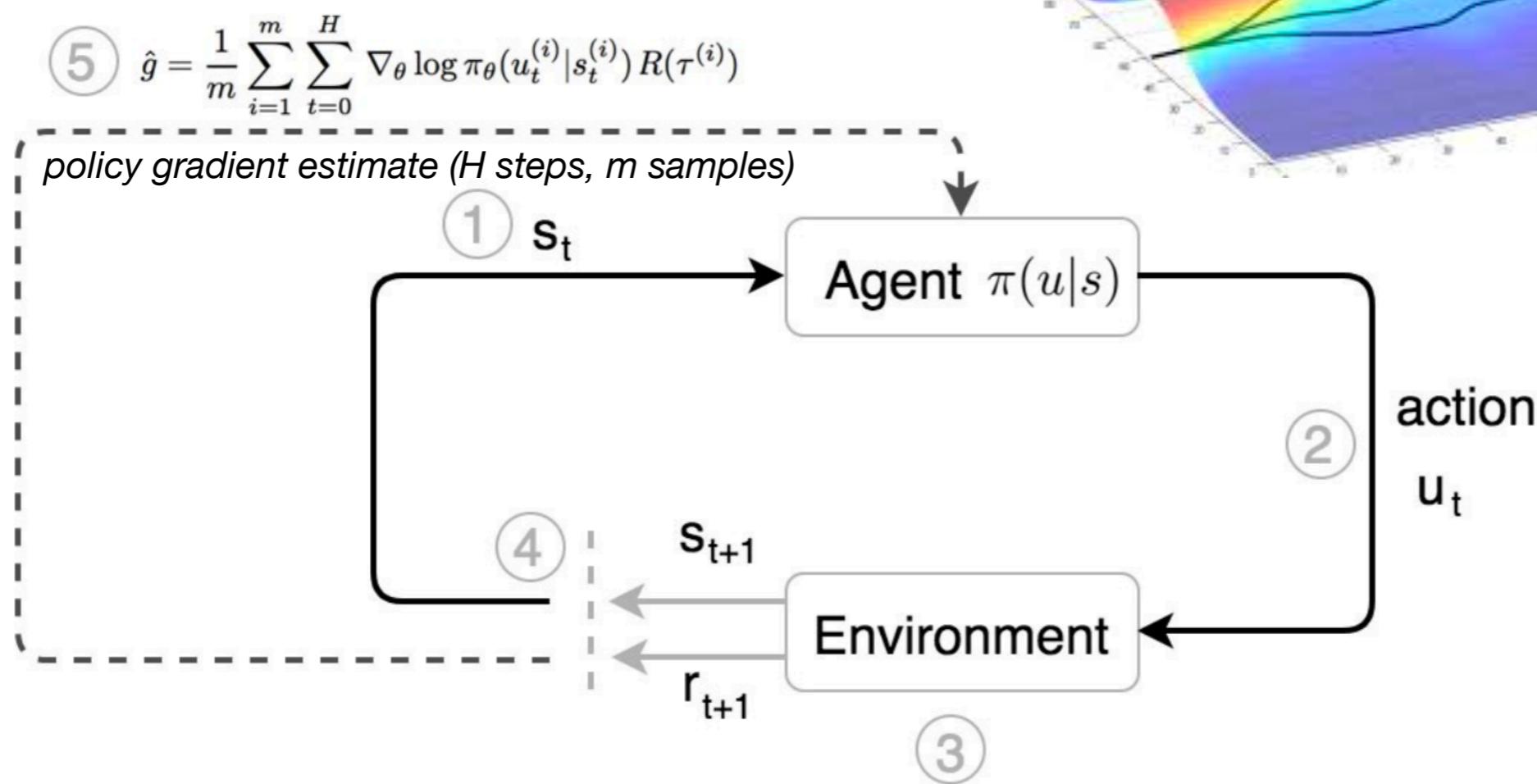
Neural architecture optimization

Standard hyperparameter optimization techniques

- AutoNet¹ : DNNs, 63 hyperparameters, tuned with SMAC
- Joint NAS + HPO²: ResNets, tuned with BO-HB
- PNAS (Progressive NAS)³
 - Cell search space, optimized with SMAC, HPO afterwards
 - SotA on ImageNet, CIFAR



Deep RL recap

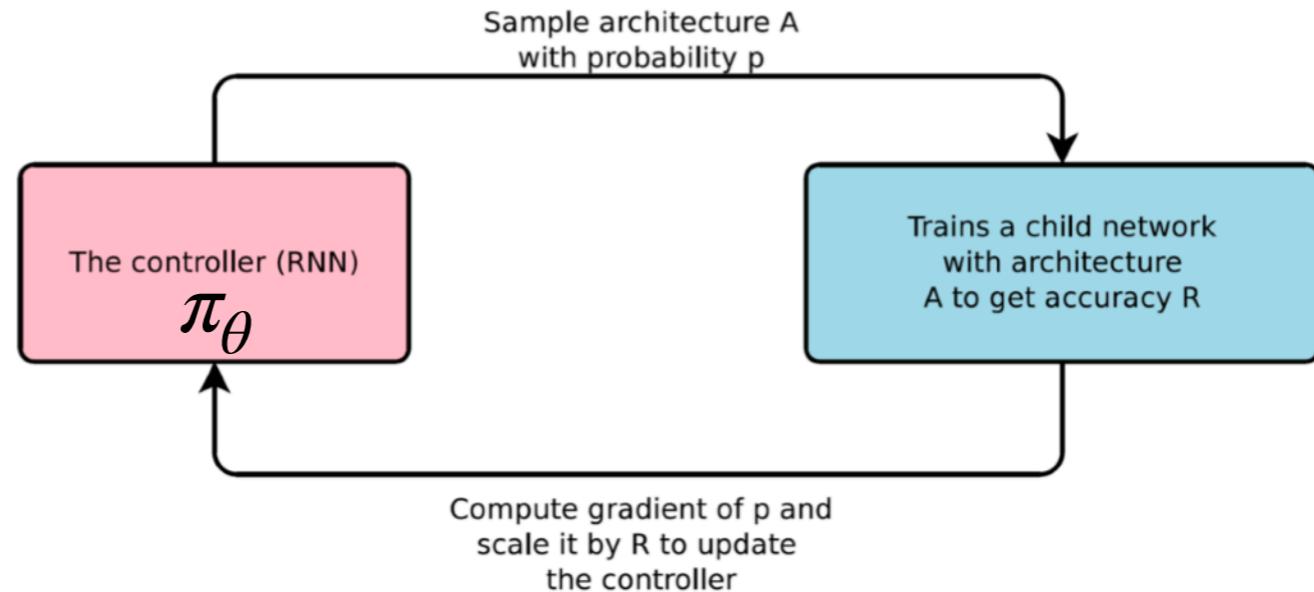


1. Observe state
2. Take action u based on policy π_θ (θ are weights in an RNN)
3. New state is formed
4. Take further actions based on new state
5. After trajectory τ of motions, adjust policy based on total reward $R(\tau)$

NAS with Reinforcement learning

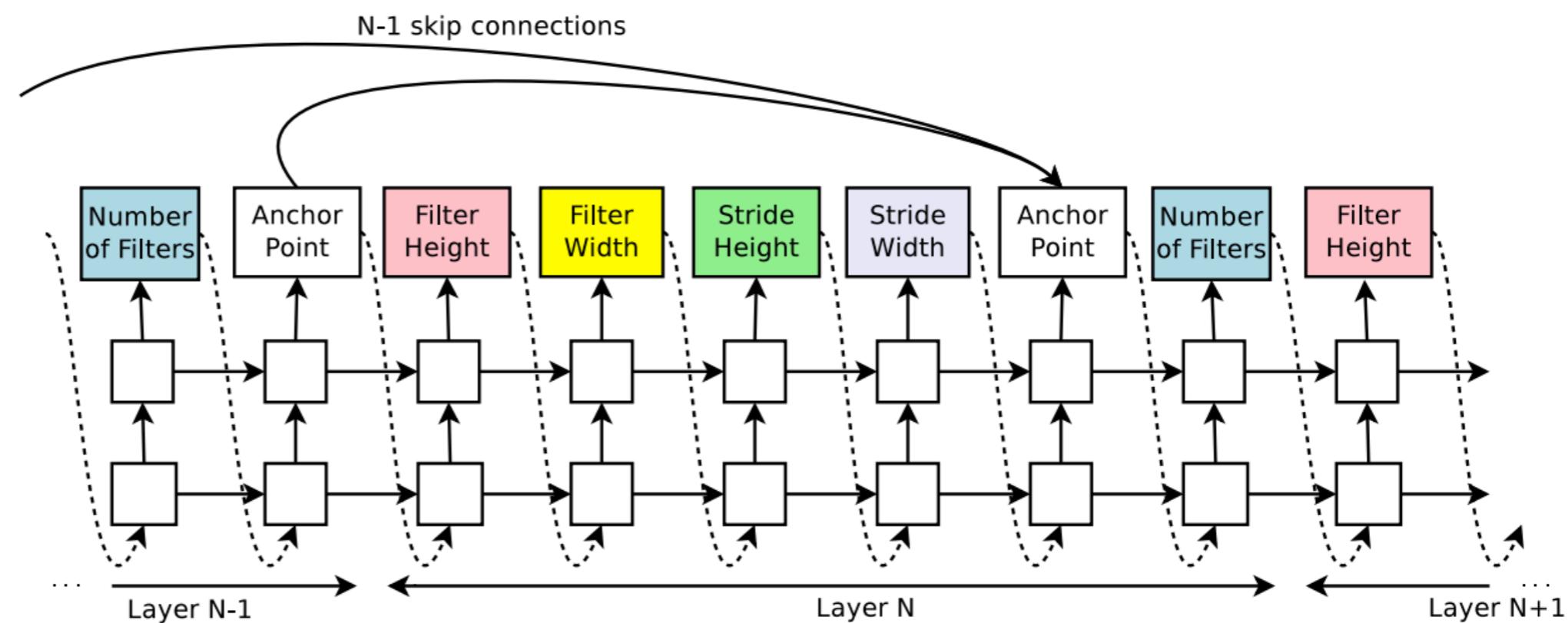
RNN policy network:

- generate architecture step by step
- evaluate to get reward
- update with policy gradient



2-layer LSTM (REINFORCE), chains of convolutional layers

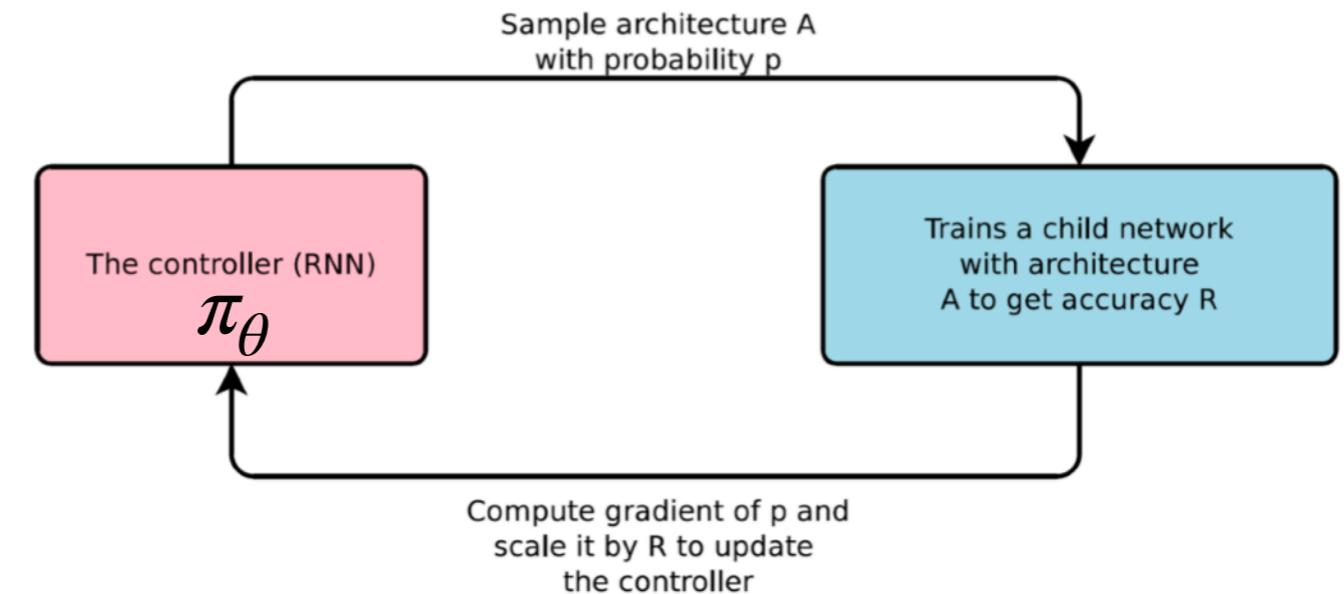
- State of the art on CIFAR-10, Penn Treebank
- 800 GPUs for 3-4 weeks, 12800 architectures



NAS with Reinforcement learning

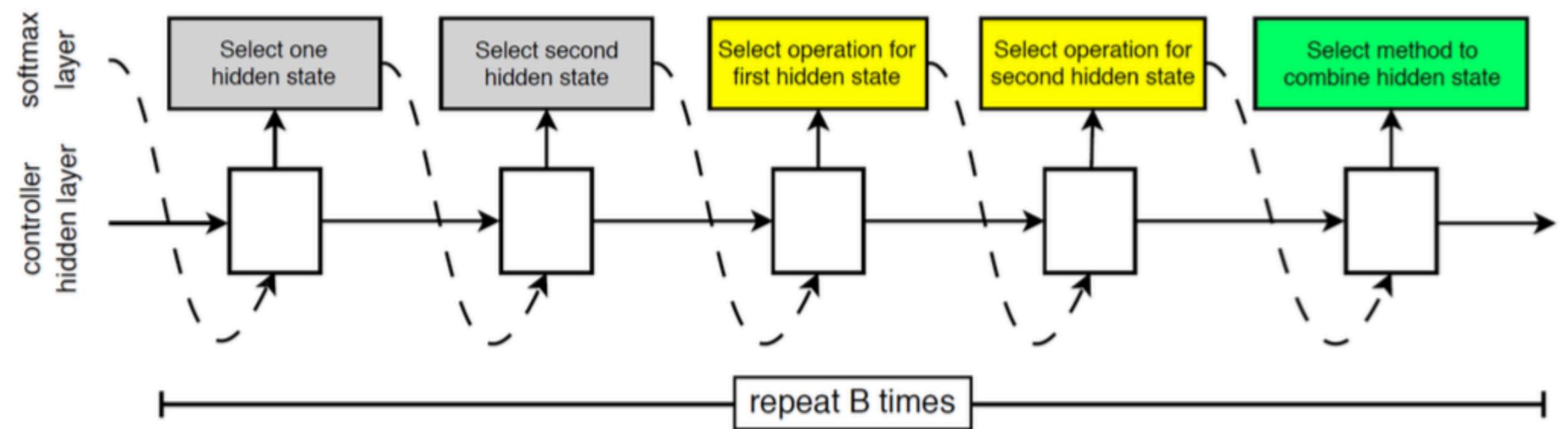
RNN policy network:

- generate architecture step by step
- evaluate to get reward
- update with policy gradient



1-layer LSTM (PPO), cell space search

- State of the art on ImageNet
- 450 GPUs, 20000 architectures

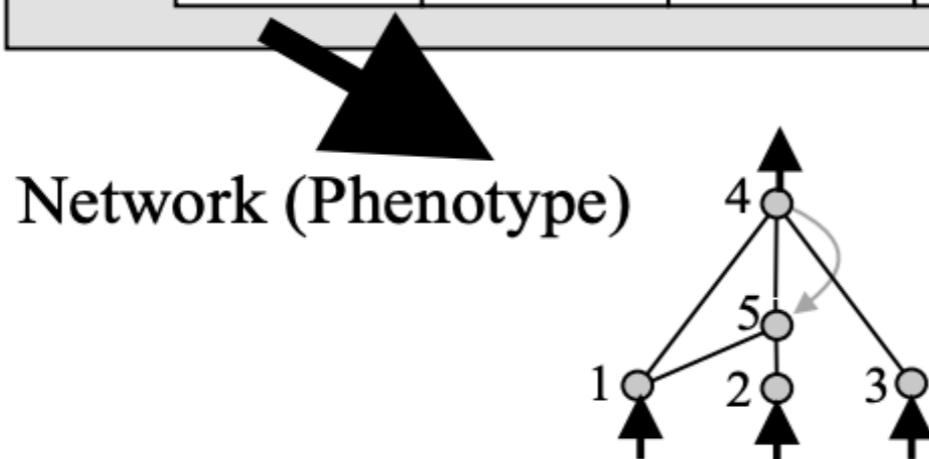


Neuroevolution

Earlier solutions tried to:

- Optimize both the configuration and the weights simultaneously with genetic algorithms
- Optimize individual nodes and connections
- Doesn't scale to deep networks

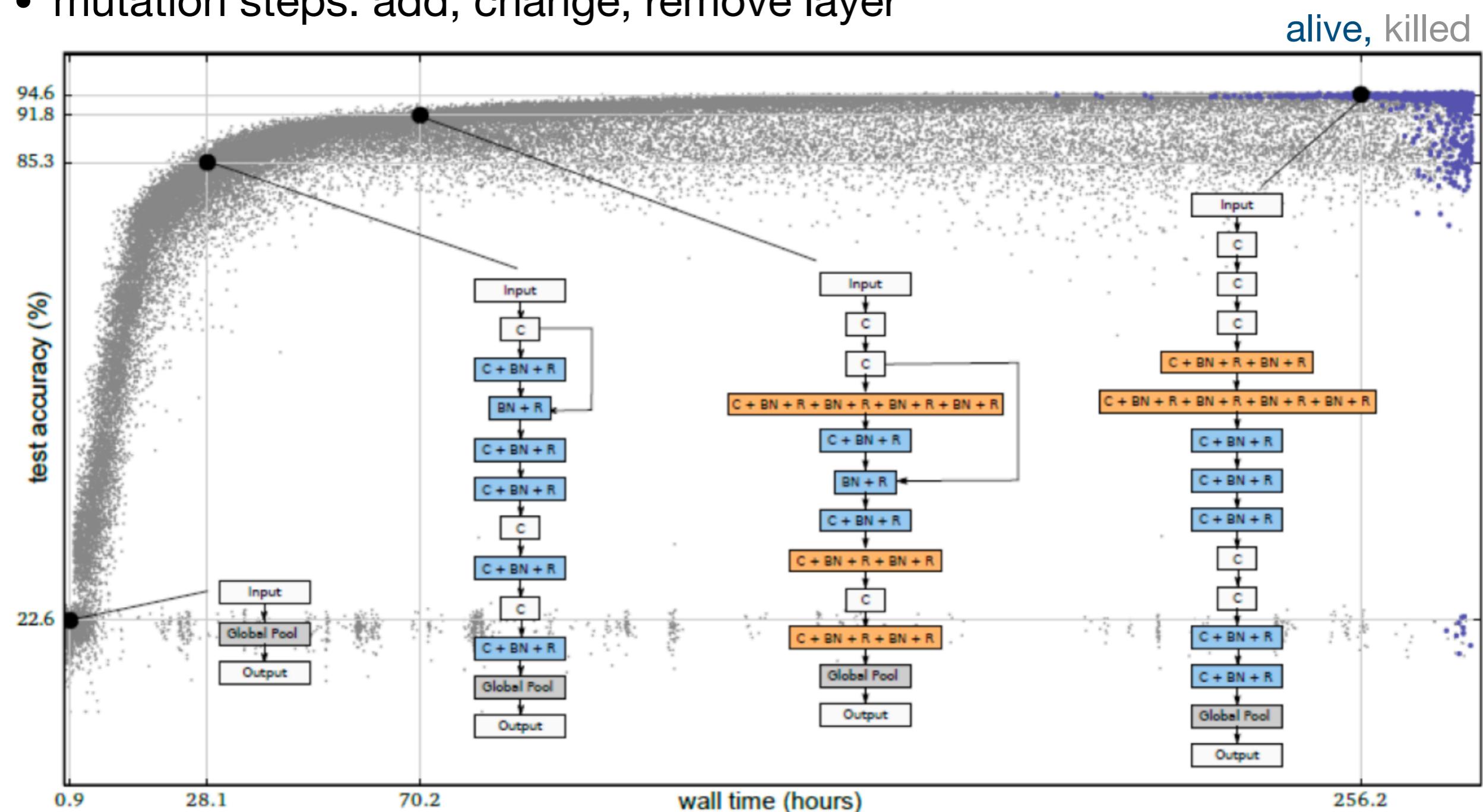
Genome (Genotype)							
Node Genes	Node 1 Sensor	Node 2 Sensor	Node 3 Sensor	Node 4 Output	Node 5 Hidden		
Connect. Genes	In 1 Out 4 Weight 0.7 Enabled Innov 1	In 2 Out 4 Weight -0.5 DISABLED Innov 2	In 3 Out 4 Weight 0.5 Enabled Innov 3	In 2 Out 5 Weight 0.2 Enabled Innov 4	In 5 Out 4 Weight 0.4 Enabled Innov 5	In 1 Out 5 Weight 0.6 Enabled Innov 6	In 4 Out 5 Weight 0.6 Enabled Innov 11



Neuroevolution

Better:

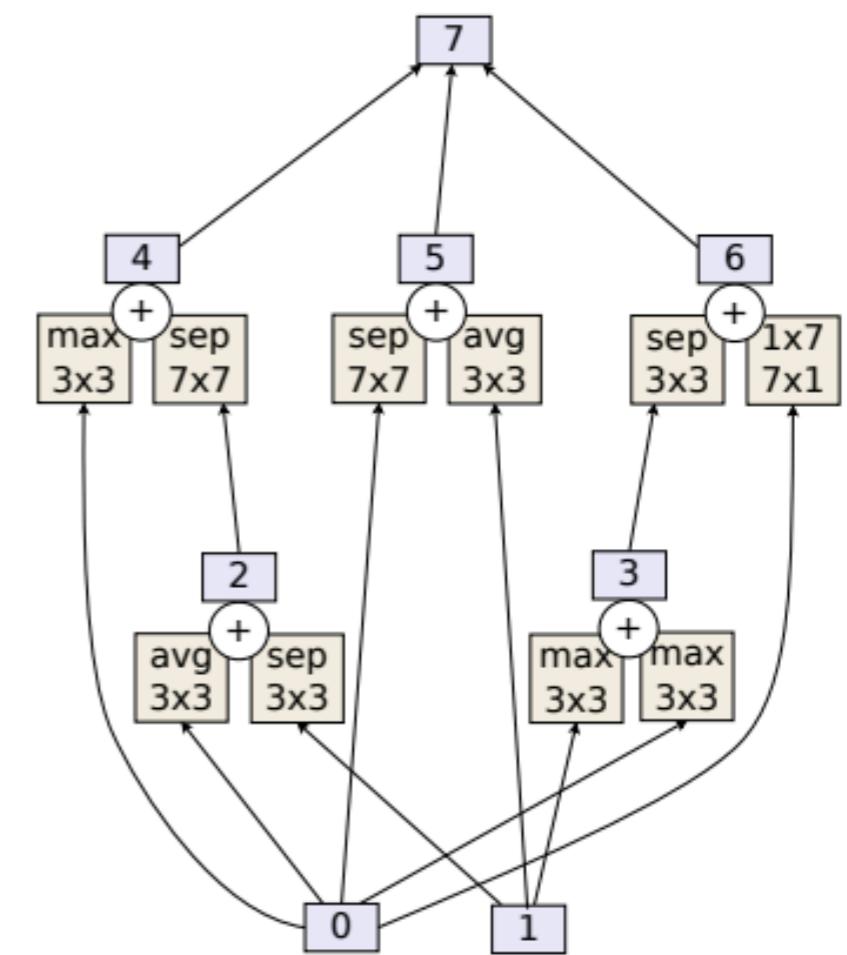
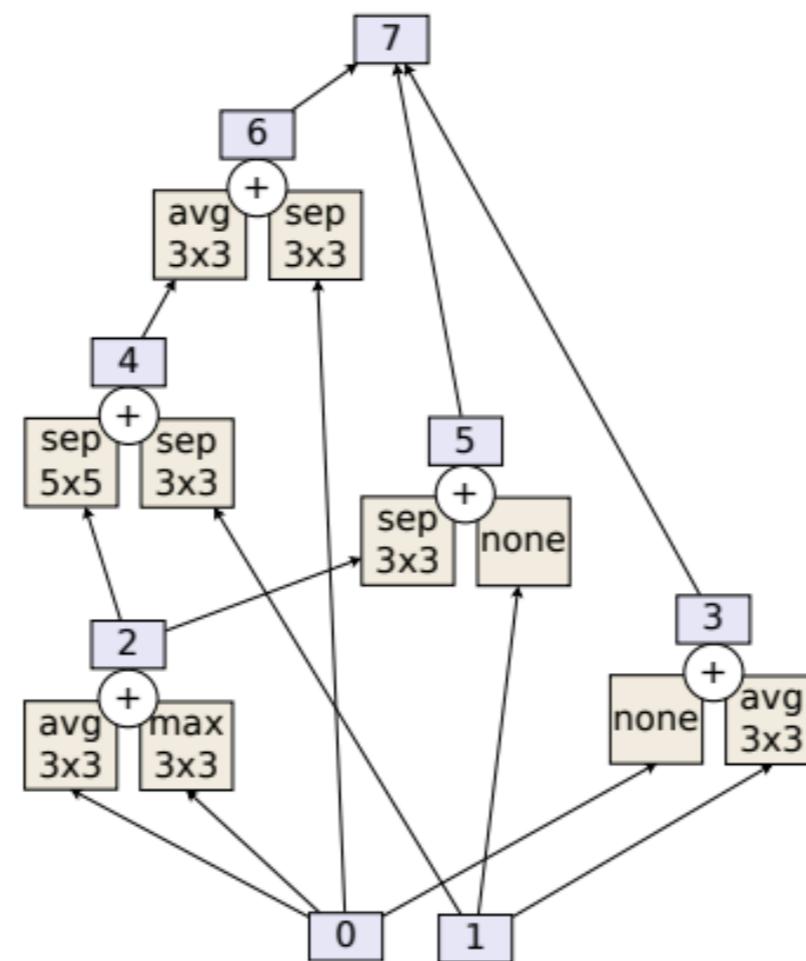
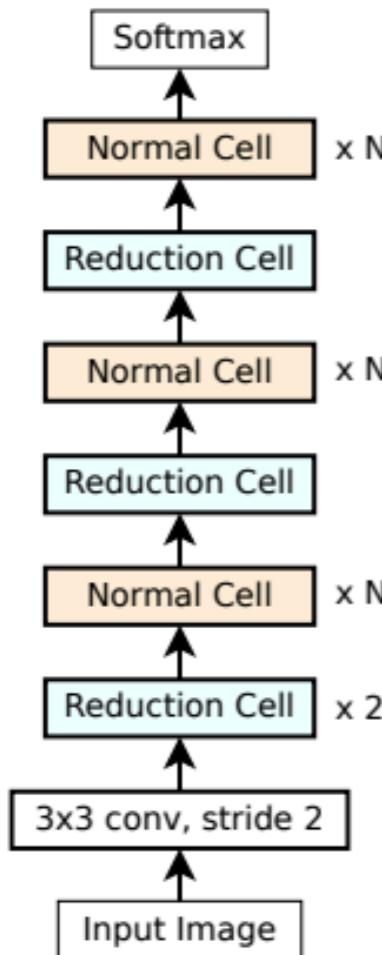
- learn the architecture and configuration with evolutionary techniques
- train the network with SGD
- mutation steps: add, change, remove layer



Neuroevolution

AmoebaNet: State of the art on ImageNet, CIFAR-10

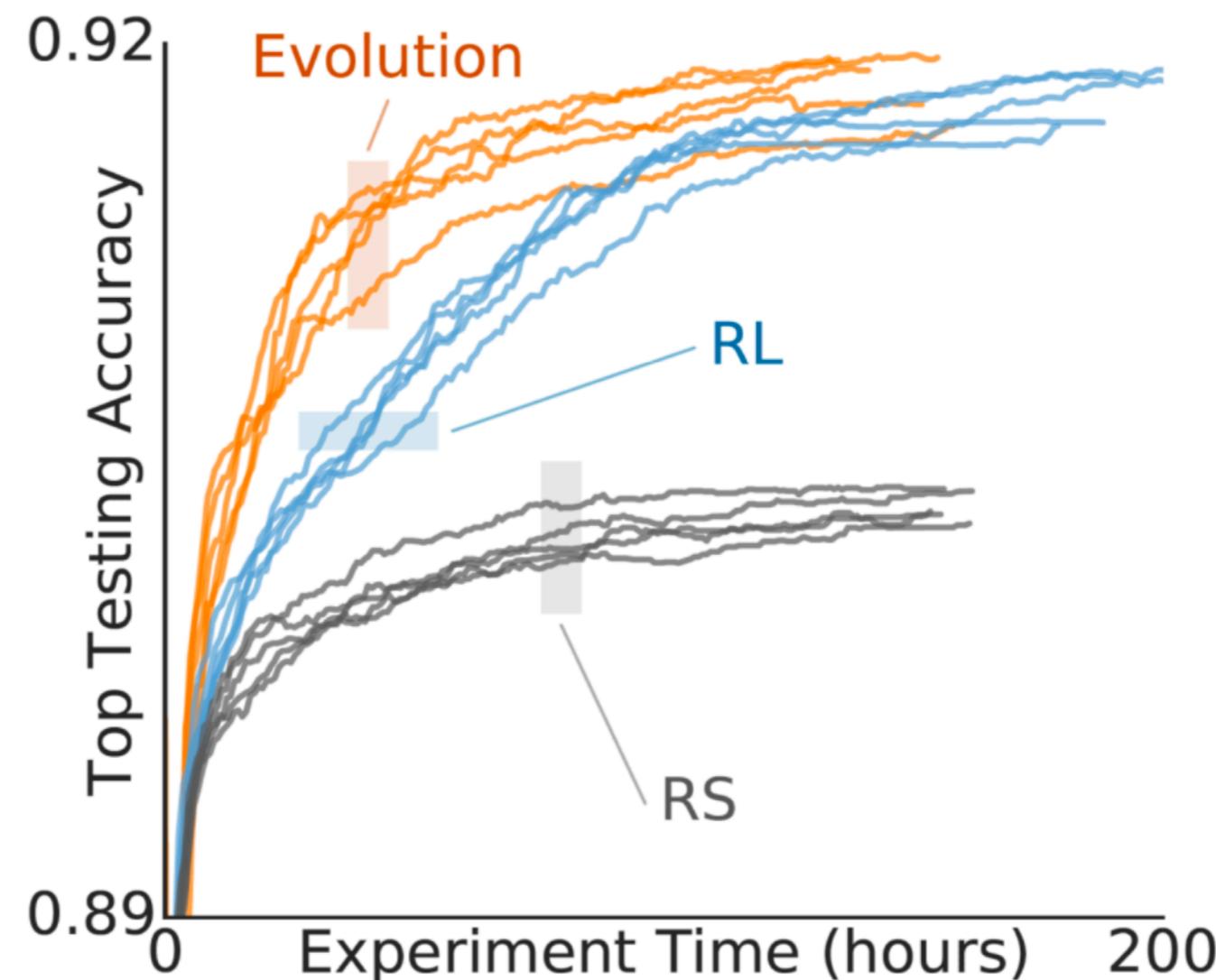
- Cell search space, aging evolution (kill oldest networks)



Neuroevolution

AmoebaNet: State of the art on ImageNet, CIFAR-10

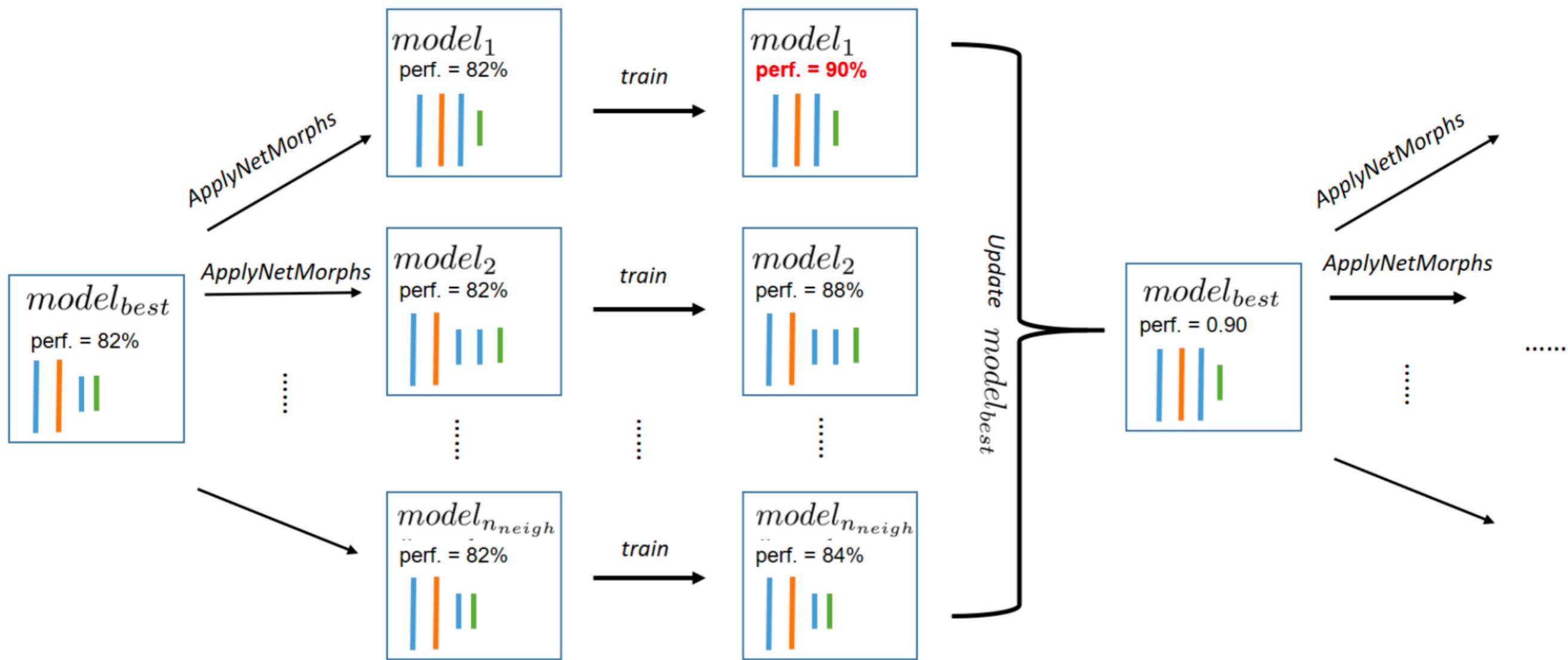
- Cell search space, aging evolution (kill oldest networks)
- More efficient than reinforcement learning



Network morphisms

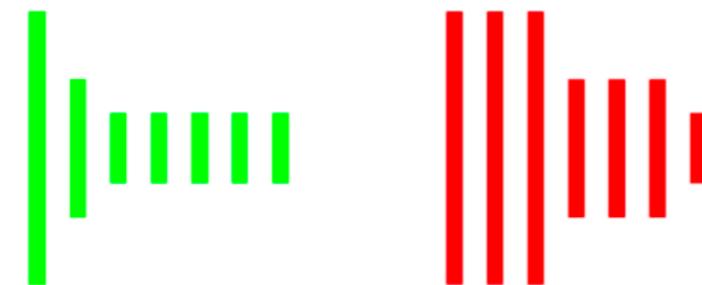


- Change architecture, but not modelled function
- Non-black box, allows much more efficient architecture search

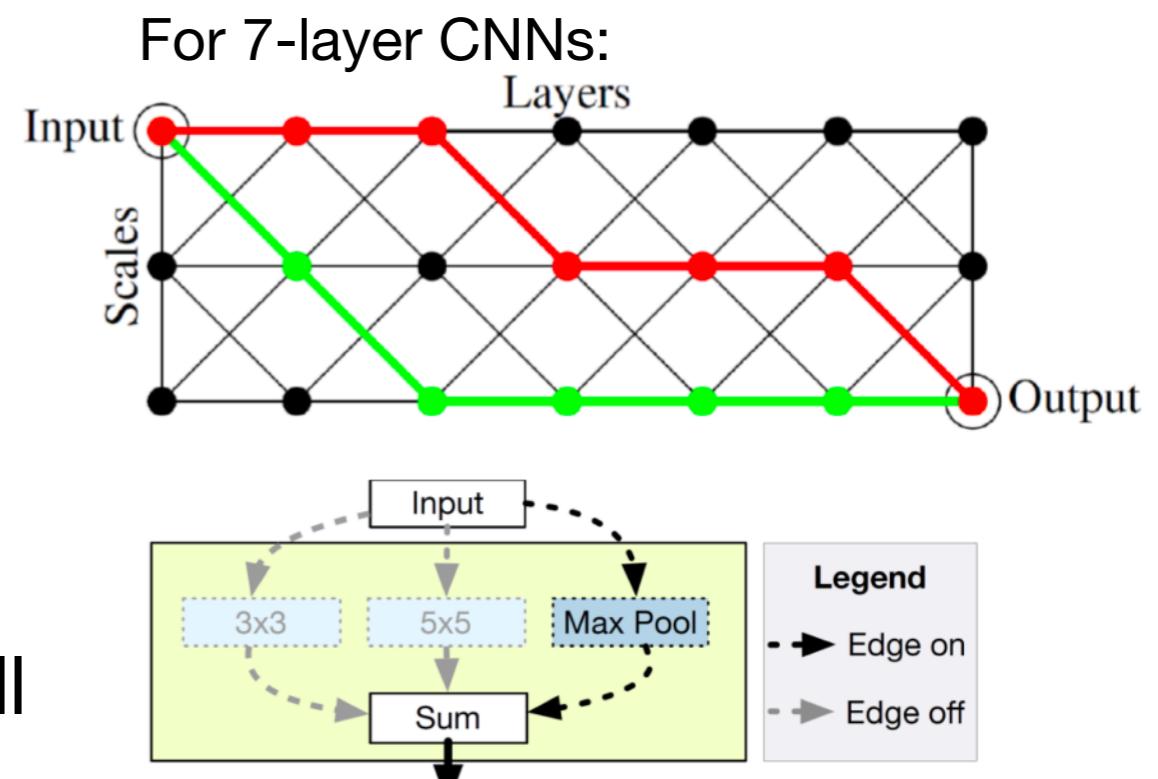


Weight sharing

- One-shot models (convolutional neural fabric)¹
 - Search space: ConvNet = path through *fabric* with shared weights
 - Train ensemble of all of them

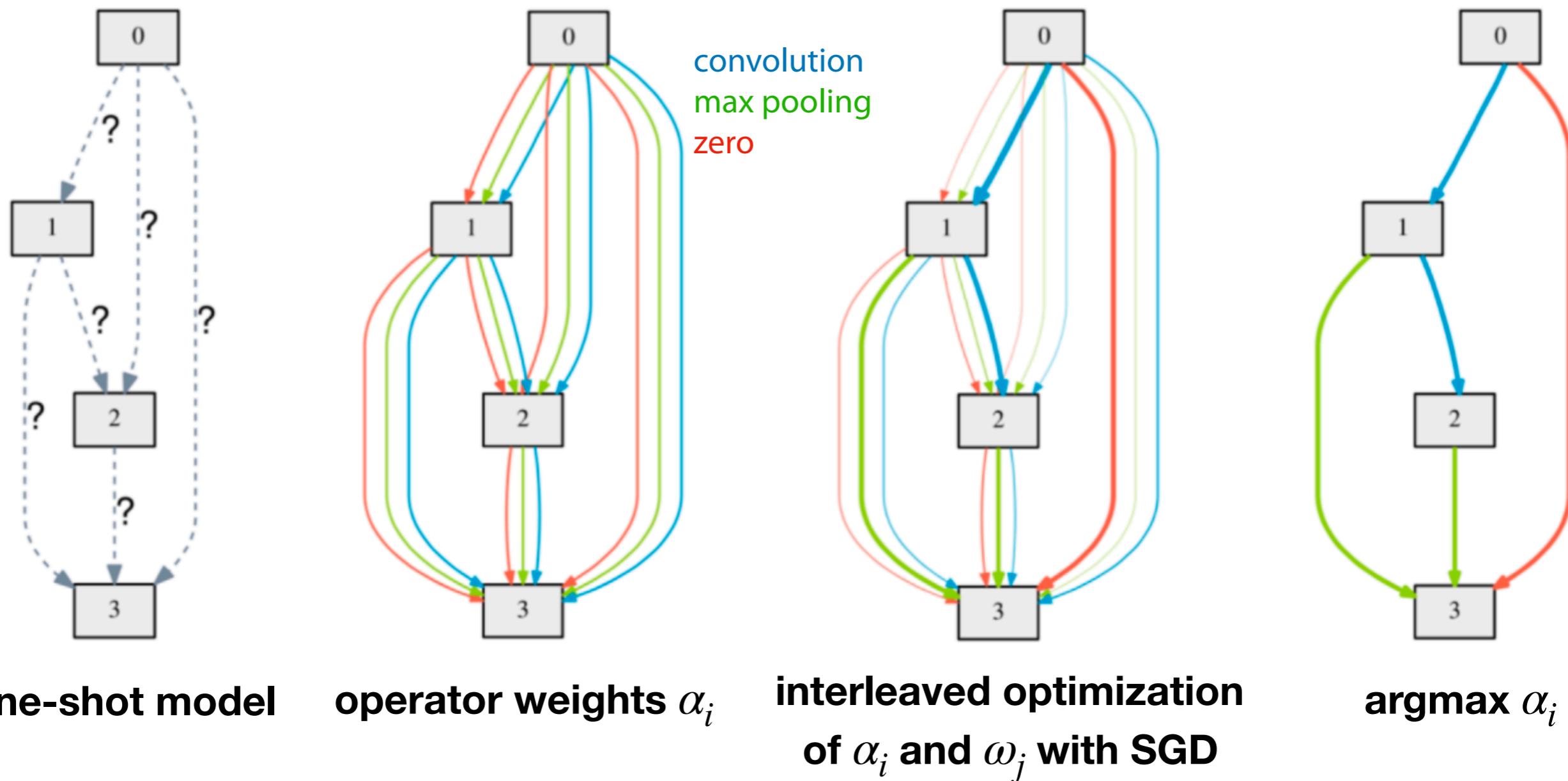


- *Path dropout*²
 - ensure that individual paths do well
- *ENAS*³
 - use RL to sample paths from one-shot model, train weights
 - other paths inherit these weights
- *SMASH*⁴
 - Shared hypernetwork that predicts weights given architecture



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 programming



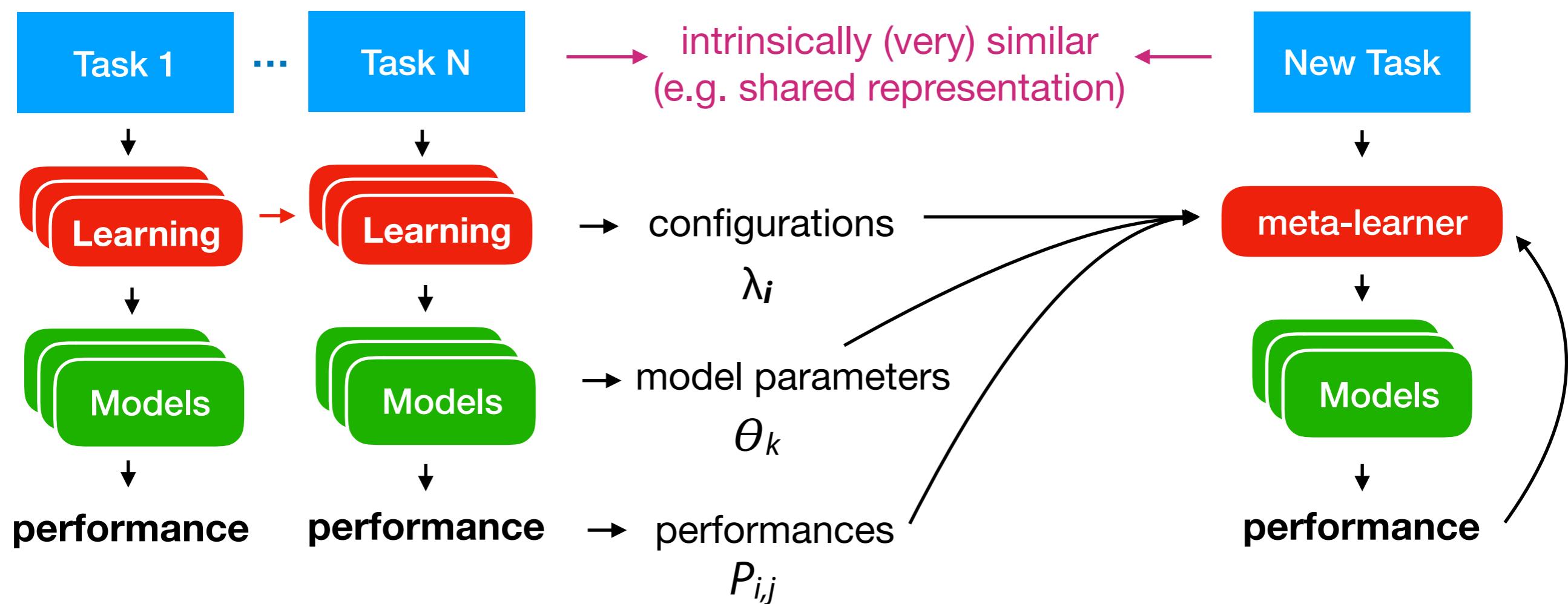
DARTS: Differentiable NAS

- Lots of further refinements
 - SNAS *[Xie et al 2019]*
 - Use Gumbel softmax (differentiable) on α_i
 - Proxyless NAS *[Cai et al 2019]*
 - Memory-efficient DARTS
 - trains sparse architectures only
 - ...

Learning to learn from previous *models*

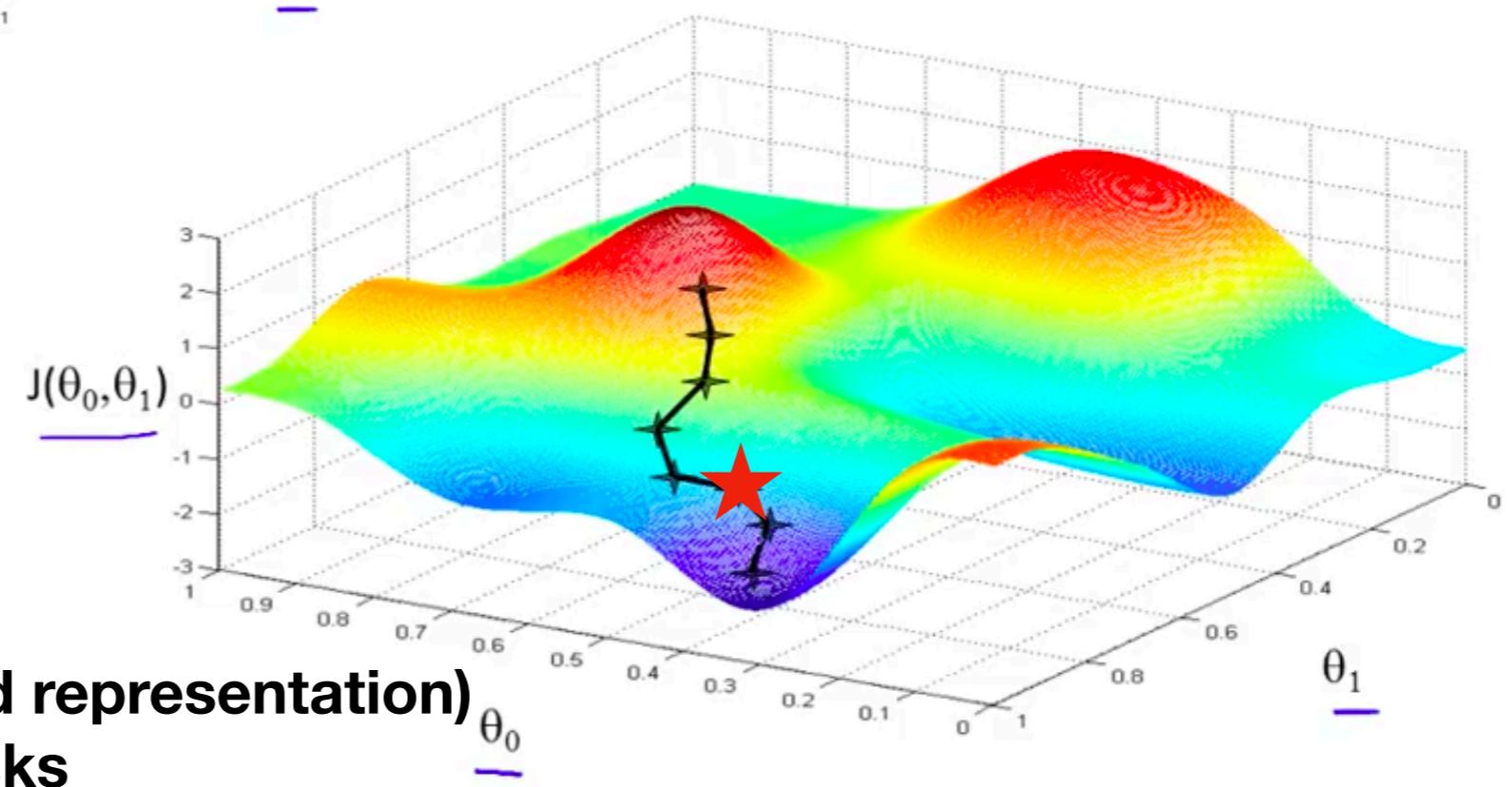
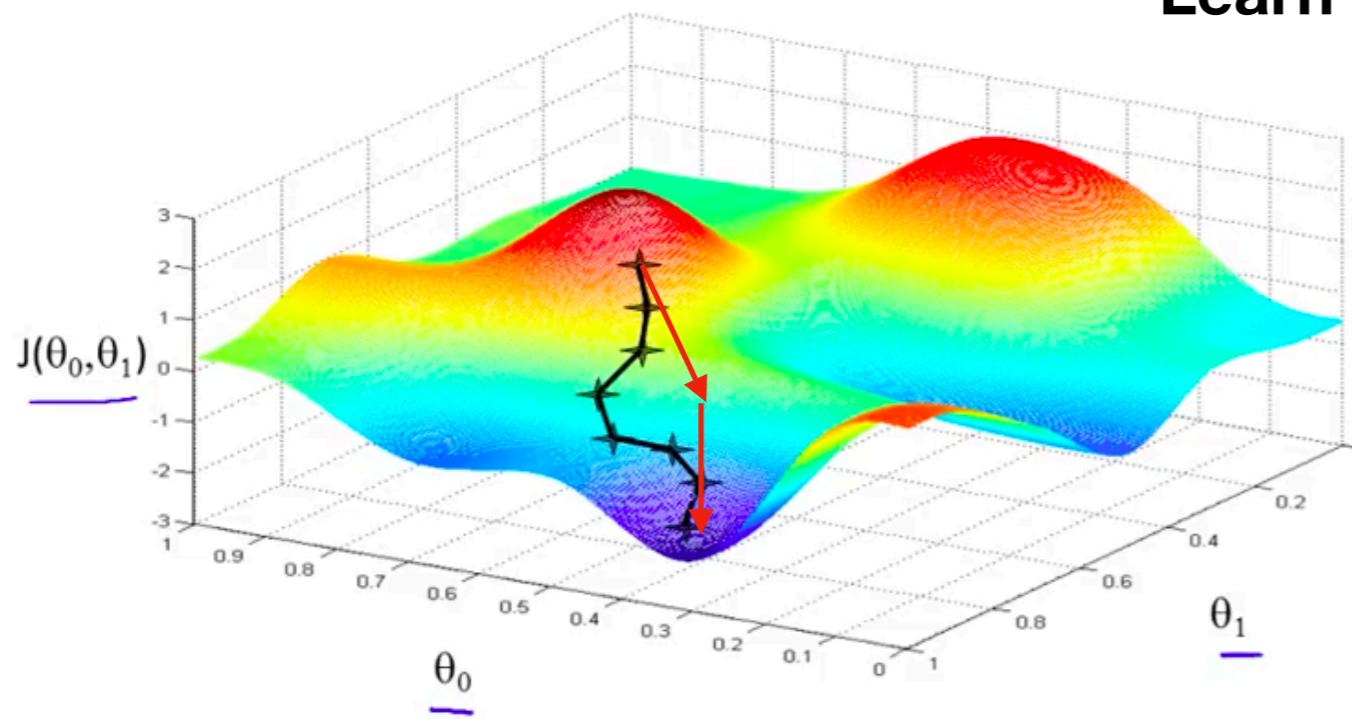
For models trained on *intrinsically similar* tasks

(model parameters, features,...)



Learning to learn

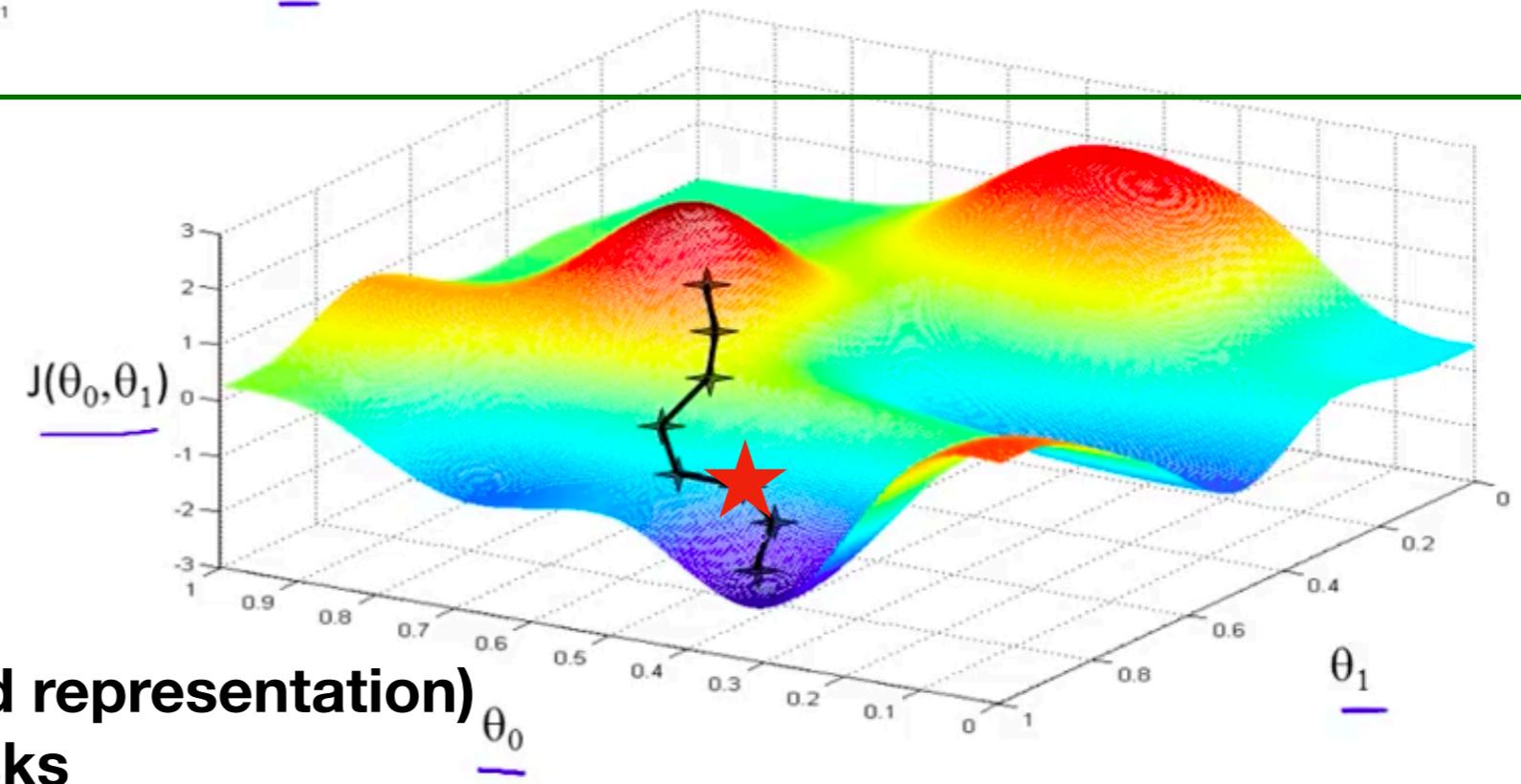
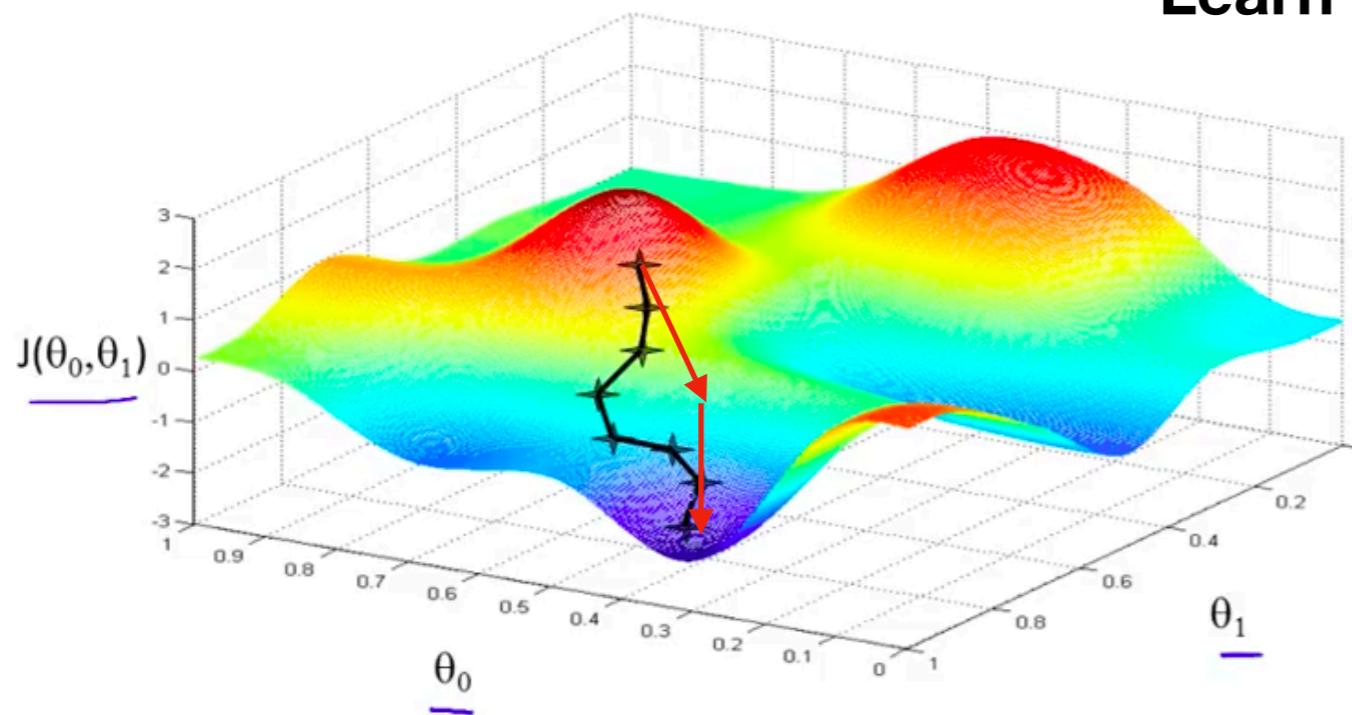
**Learn a better gradient update rule
for a group of tasks**



**Learn a better initialization (and representation)
for a group of tasks**

Learning to learn

Learn a better gradient update rule
for a group of tasks

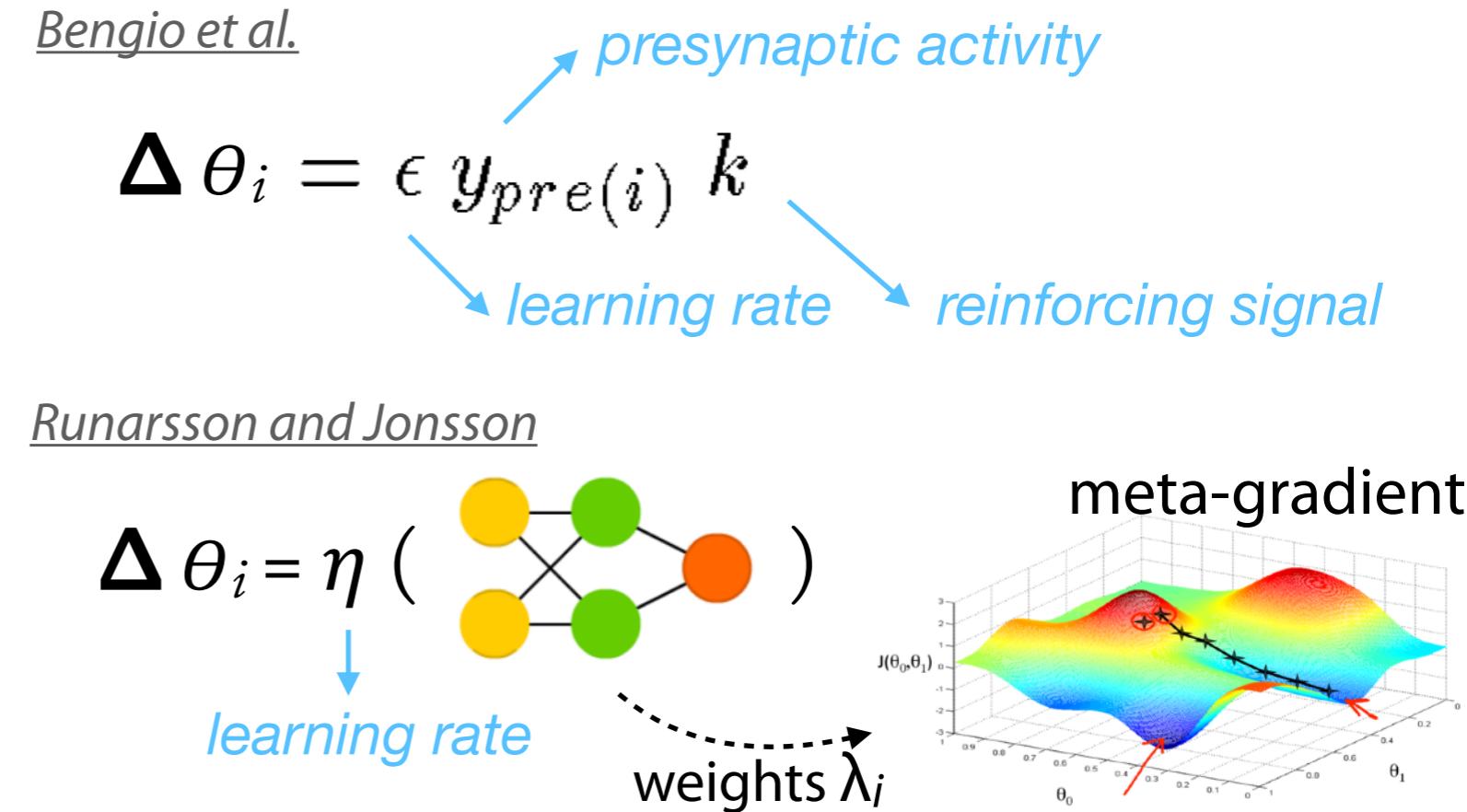
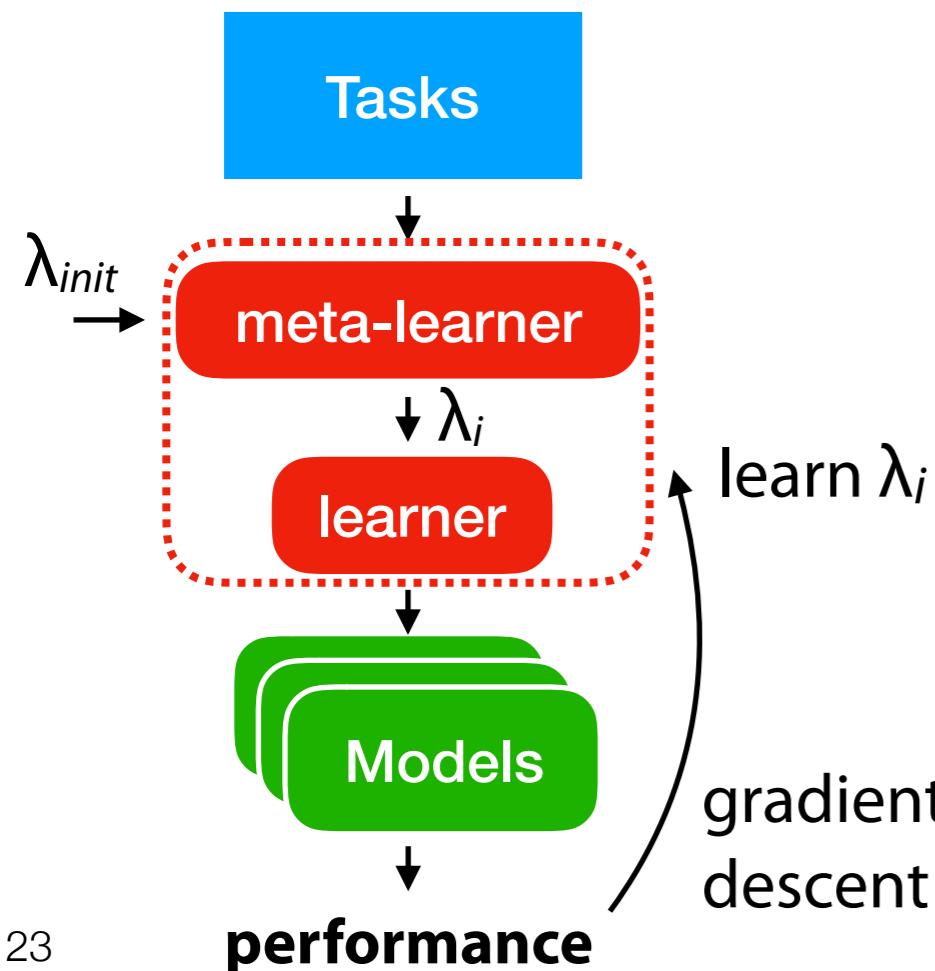


Learn a better initialization (and representation)
for a group of tasks

Learning to learn by gradient descent

- Our brains *probably* don't do backprop, replace it with:
 - Simple *parametric* (bio-inspired) rule to update weights ¹
 - Single-layer neural network to learn weight updates ²
 - Learn parameters across tasks, by gradient descent (meta-gradient)

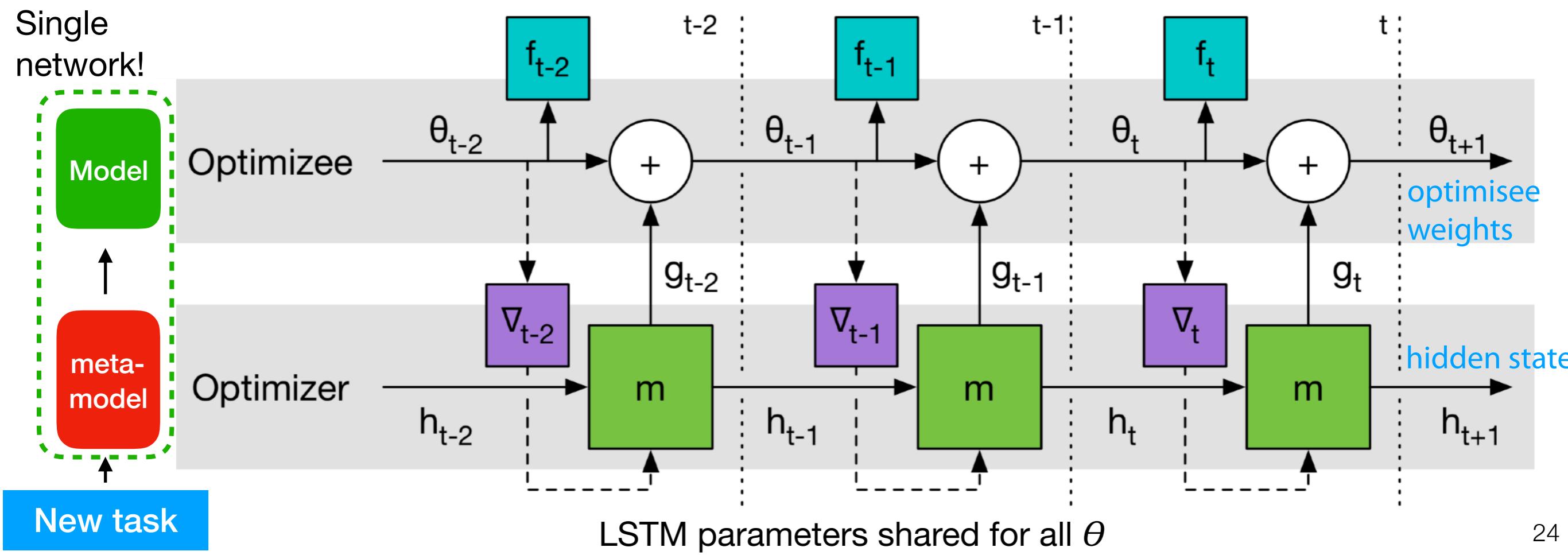
$$\Delta w_{ij} = -\eta \frac{\partial E_p}{\partial w_{ij}}$$



Learning to learn gradient descent

by gradient descent

- Replace backprop with a recurrent neural net (LSTM)¹, **not so scalable**
- Use a coordinatewise LSTM [m] for scalability/flexibility (cfr. ADAM, RMSprop) ²
 - Optimizee: receives weight update g_t from optimizer
 - Optimizer: receives gradient estimate ∇_t from optimizee
 - Learns how to do gradient descent across tasks



Learning to learn gradient descent by gradient descent

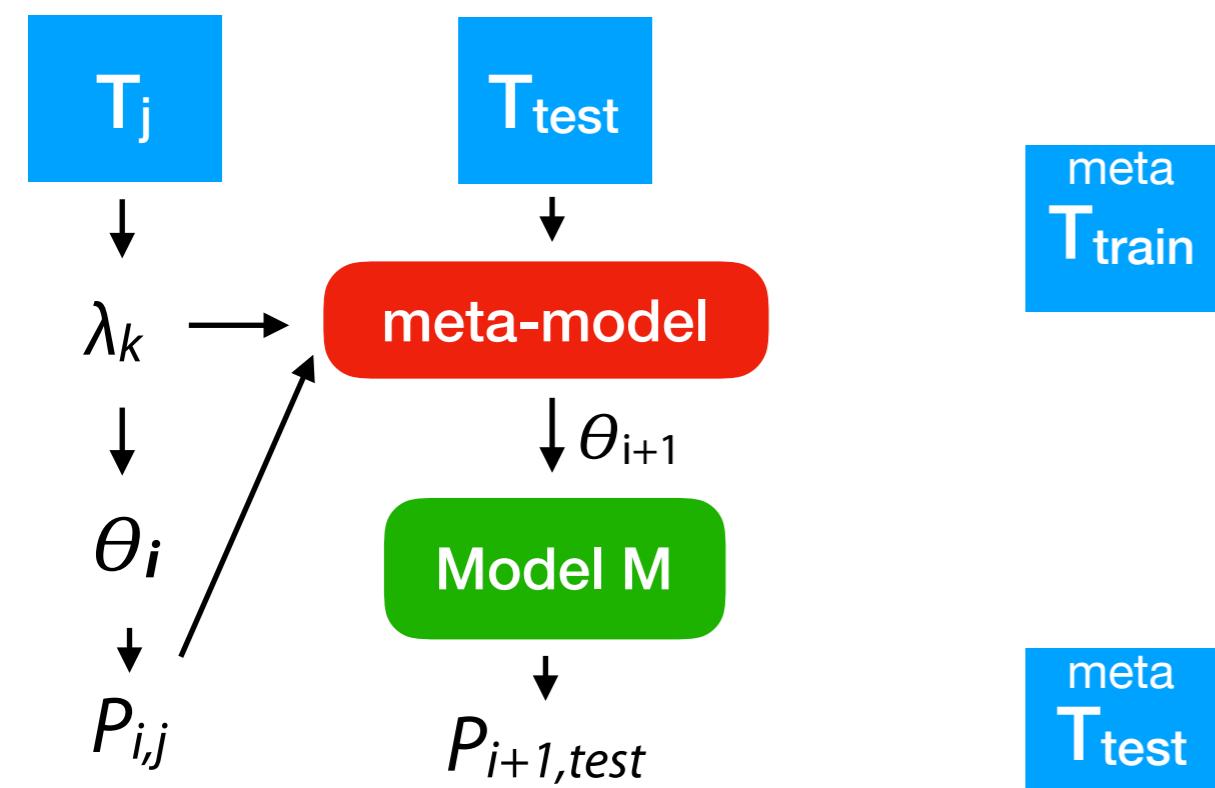


- Left: optimizer and optimizee trained to do style transfer
- Right: optimizer solves similar tasks (different style, content and resolution) without any more training data

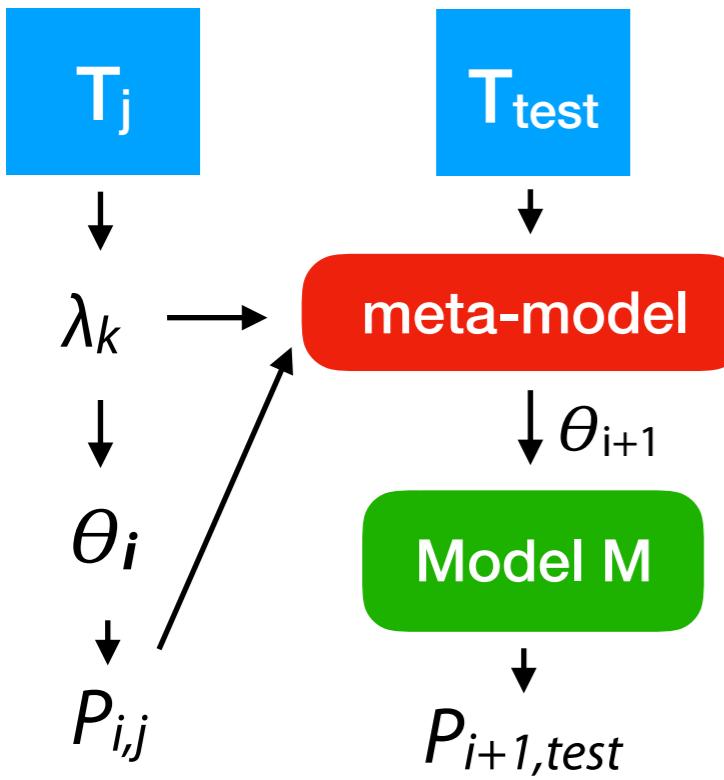
Application: Few-shot learning

- Learn how to learn from few examples (given similar tasks)
 - Meta-learner must learn how to train a base-learner based on prior experience
 - Parameterize base-learner model and learn the parameters θ_i

$$Cost(\theta_i) = \frac{1}{|T_{test}|} \sum_{t \in T_{test}} loss(\theta_i, t)$$



Few-shot learning: approaches



$$Cost(\theta_i) = \frac{1}{|T_{test}|} \sum_{t \in T_{test}} loss(\theta_i, t)$$

- Existing algorithm as meta-learner:
 - LSTM + gradient descent *Ravi and Larochelle 2017*
 - Learn θ_{init} + gradient descent *Finn et al. 2017*
 - kNN-like: Memory + similarity *Vinyals et al. 2016*
 - Learn embedding + classifier *Snell et al. 2017*
 - ...
- Black-box meta-learner
 - Neural Turing machine (with memory) *Santoro et al. 2016*
 - Neural attentive learner *Mishra et al. 2018*
 - ...

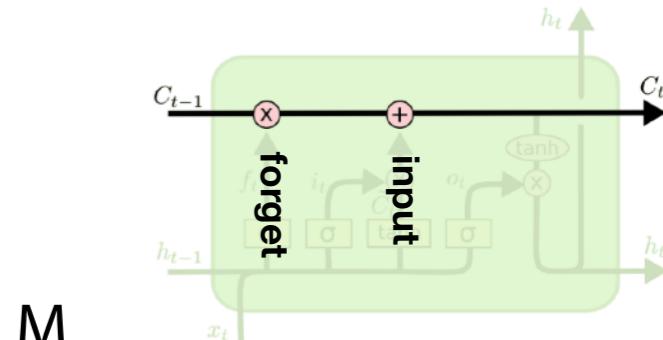
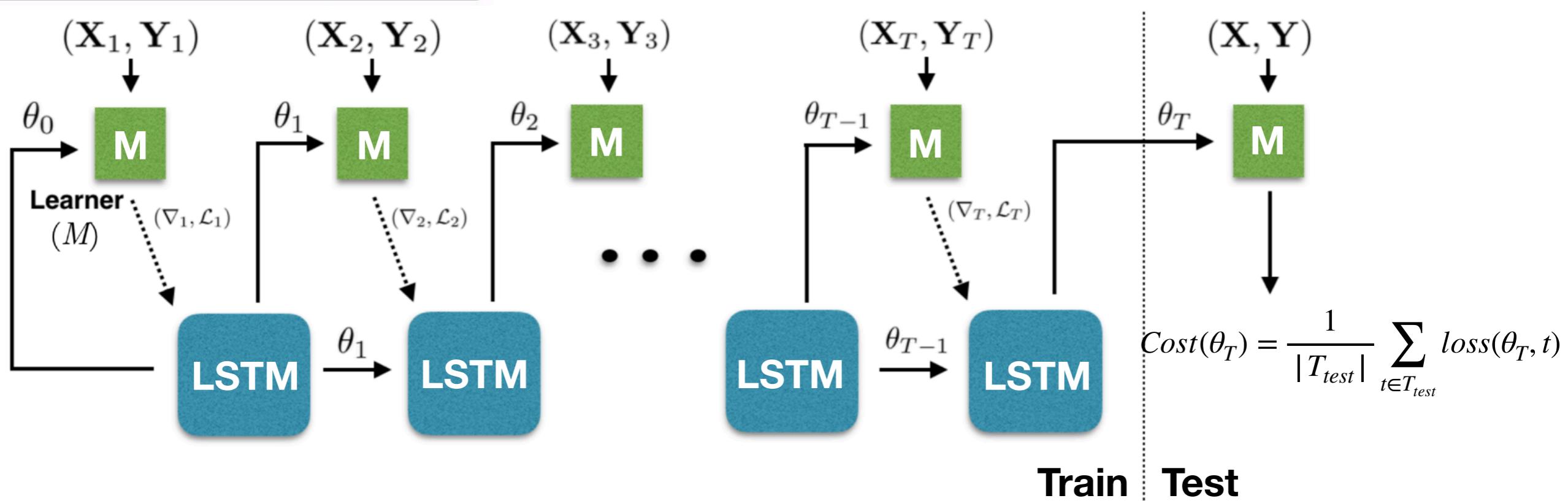
LSTM meta-learner + gradient descent

- Gradient descent update θ_t is similar to LSTM cell state update c_t

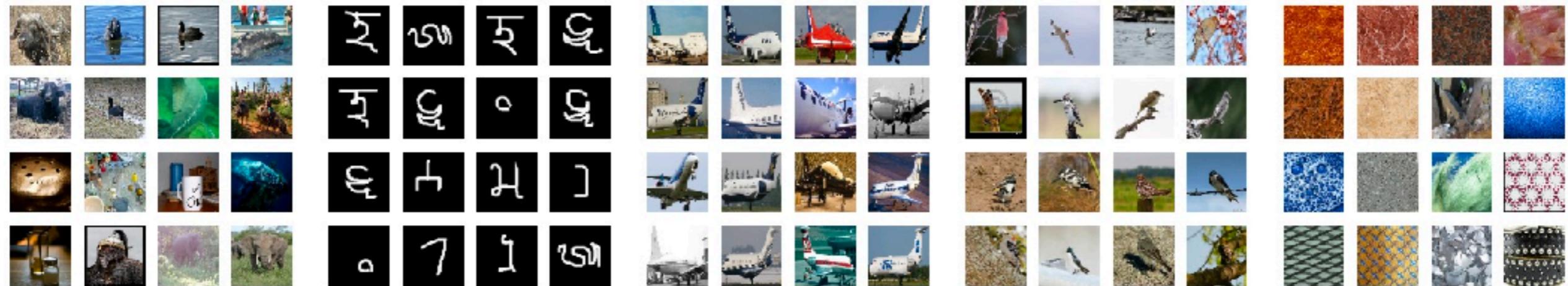
$$\theta_t = \theta_{t-1} - \alpha_t \nabla_{\theta_{t-1}} \mathcal{L}_t \quad c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

- Hence, training a meta-learner LSTM yields an update rule for training M

- Start from initial θ_0 , train model on first batch, get gradient and loss update
- Predict θ_{t+1} , continue to $t=T$, get cost, backpropagate to learn LSTM weights, optimal θ_0



Meta-dataset



(a) ImageNet

(b) Omniglot

(c) Aircraft

(d) Birds

(e) DTD



(f) Quick Draw

(g) Fungi

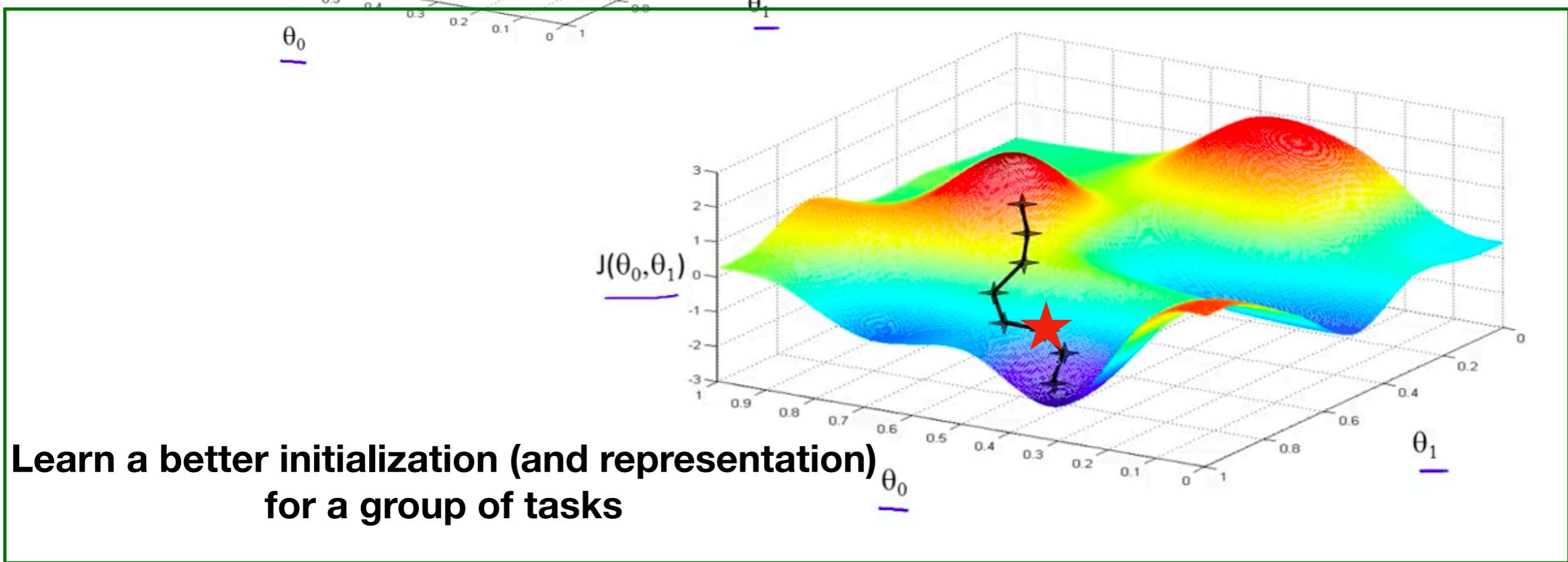
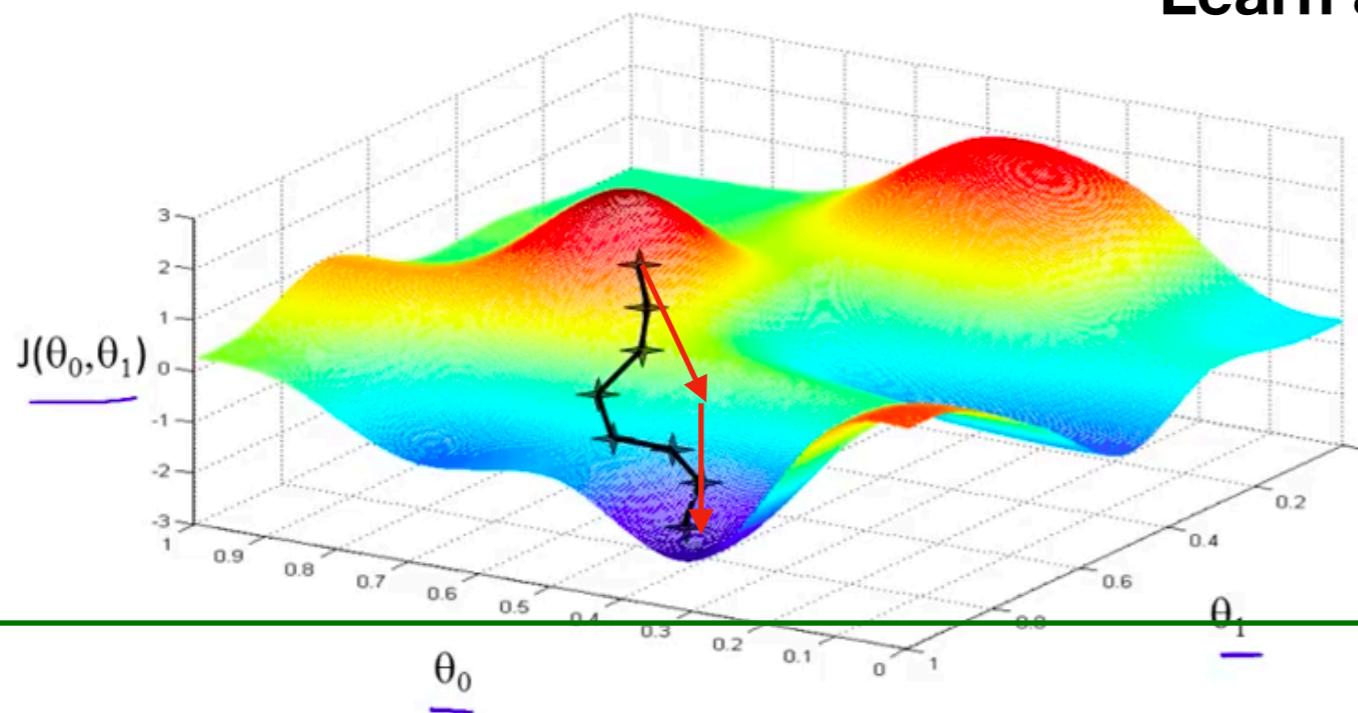
(h) VGG Flower

(i) Traffic Signs

(j) MSCOCO

Learning to learn

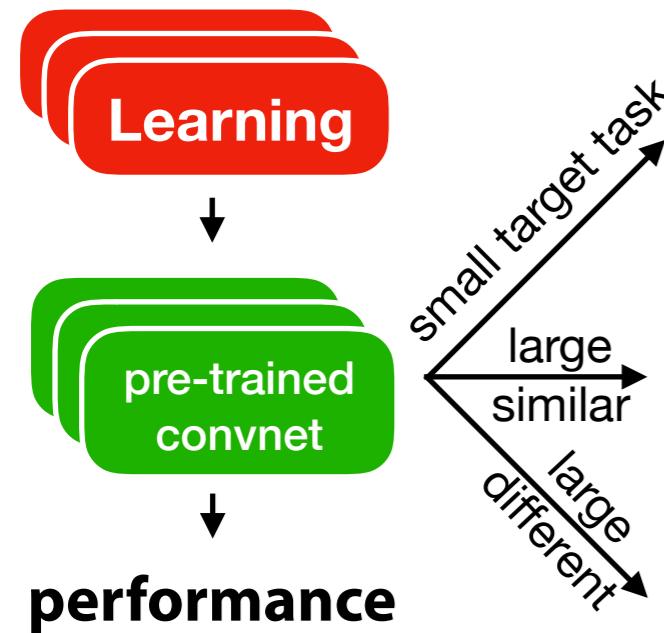
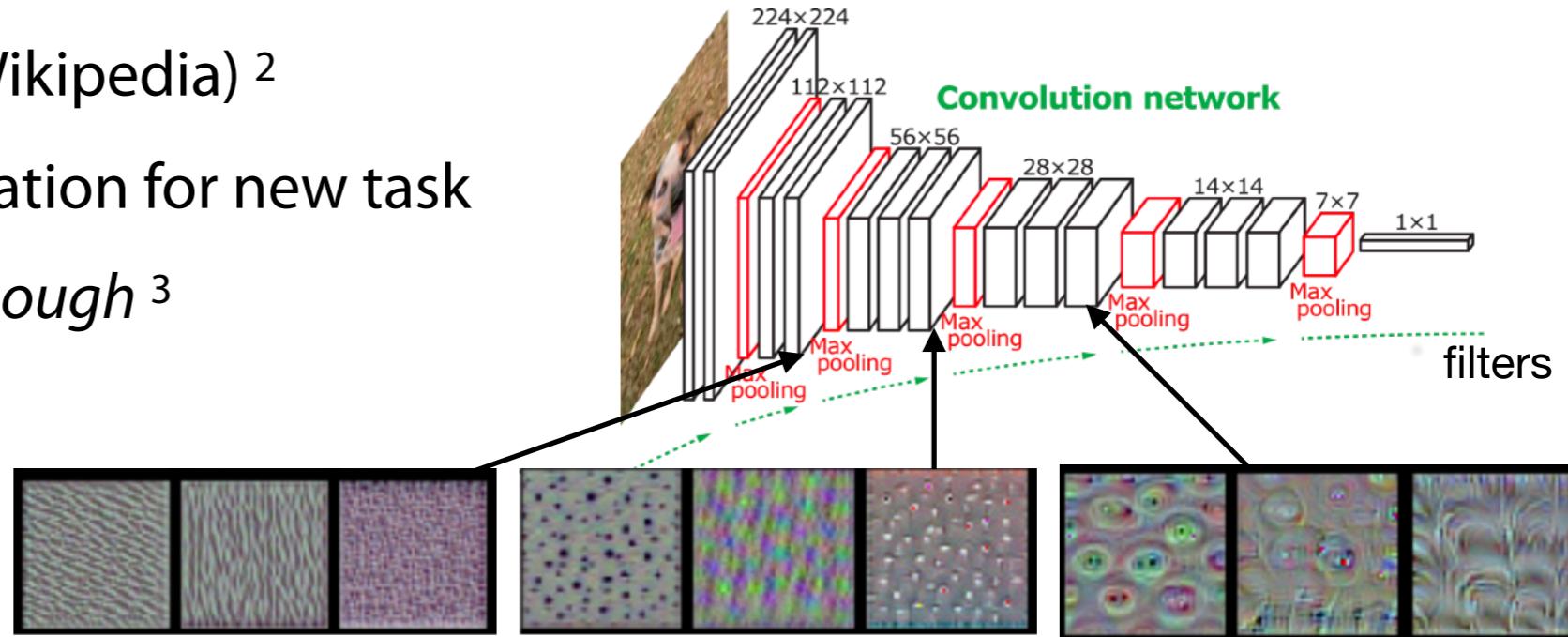
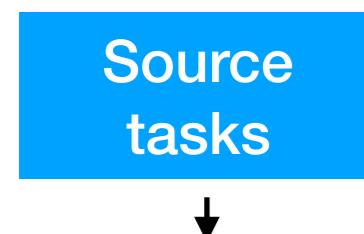
**Learn a better gradient update rule
for a group of tasks**



**Learn a better initialization (and representation)
for a group of tasks**

Transfer learning

- Pre-train weights from:
 - Large image datasets (e.g. ImageNet)¹
 - Large text corpora (e.g. Wikipedia)²
- Use these weights as initialization for new task
- Fails if tasks are *not similar enough*³



Feature extraction:
remove last layers, use output as features
if task is quite different, remove more layers

End-to-end tuning:
train from initialized weights

Fine-tuning:
unfreeze last layers, tune on new task

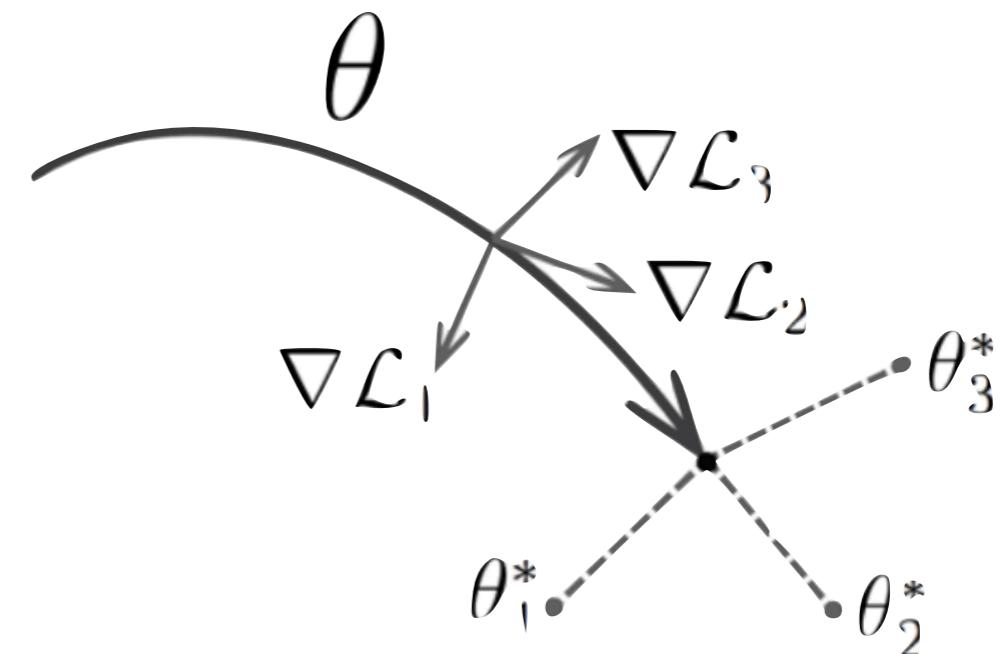
¹ Razavian et al. 2014

² Mikolov et al. 2013

³ Yosinski et al. 2014

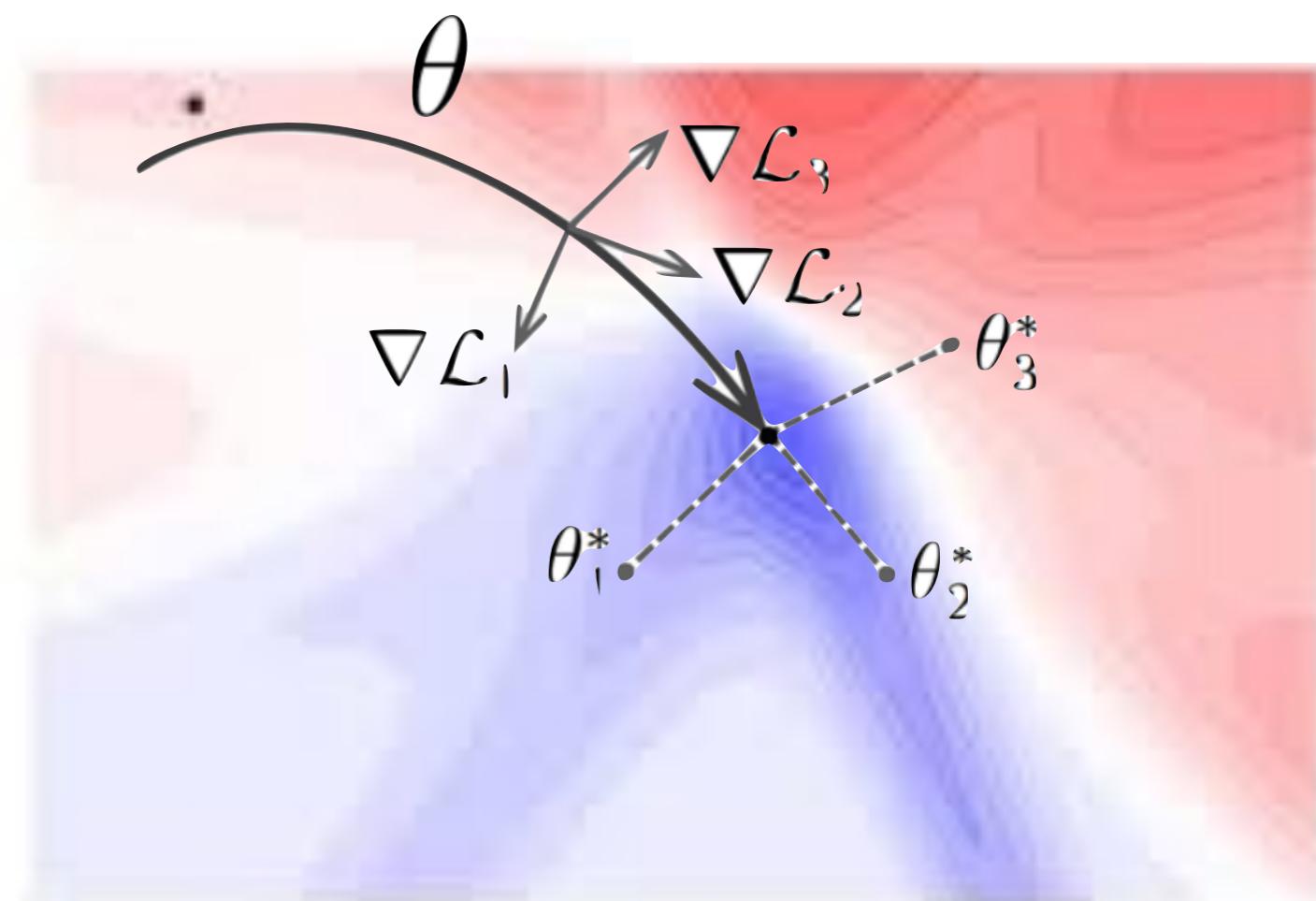
Model-agnostic meta-learning

- Solve new tasks faster by learning a model *initialization* from similar tasks
 - Current initialization θ
 - On K examples/task, evaluate $\nabla_{\theta} L_{T_i}(f_{\theta})$
 - Update weights for $\theta_1, \theta_2, \theta_3$
 - Update θ to minimize sum of per-task losses
$$\theta \leftarrow \theta - \beta \nabla_{\theta} \sum_{T_i \sim p(T)} \mathcal{L}_{T_i}(f_{\theta'_i})$$
- Repeat



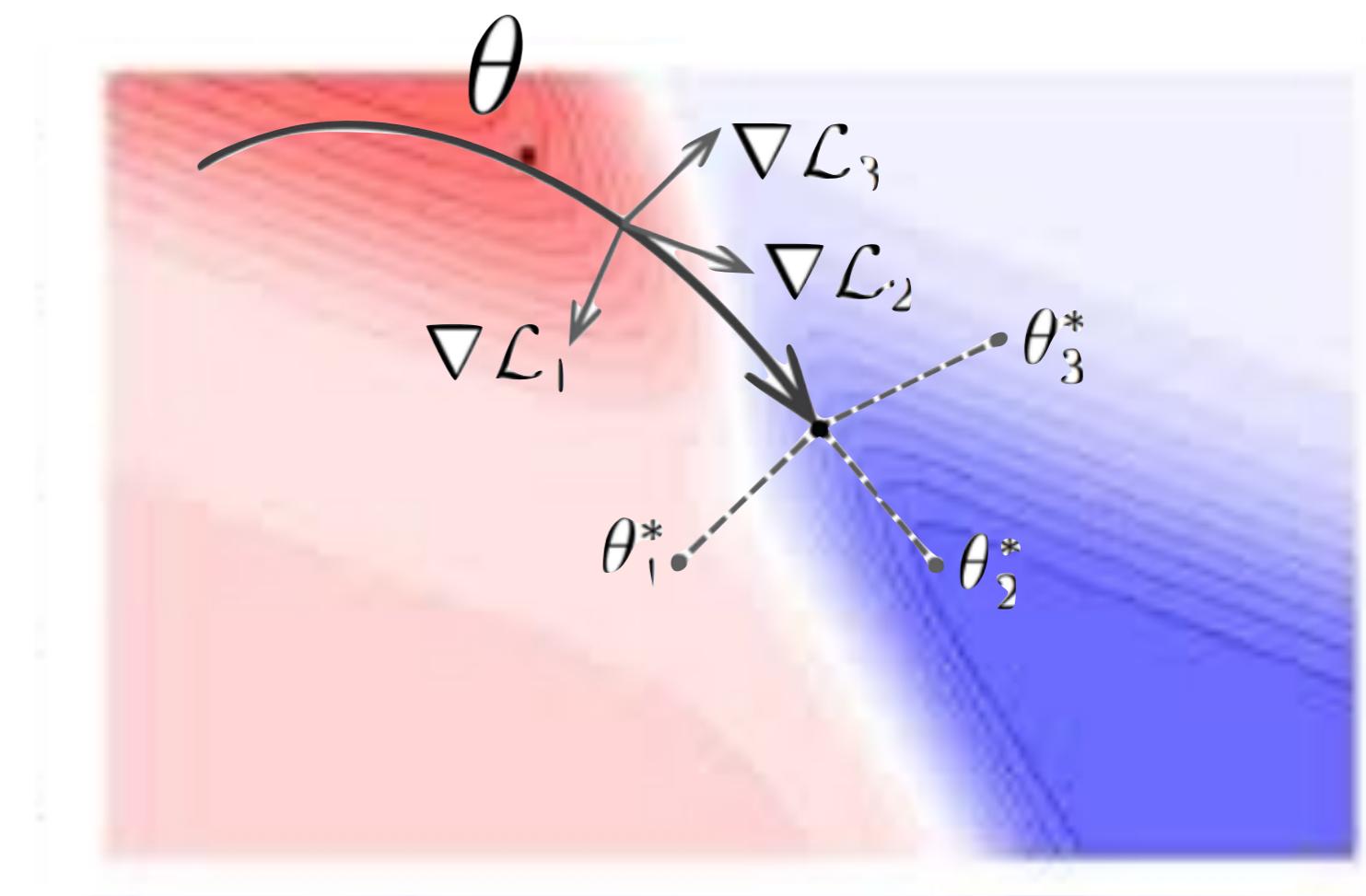
Model-agnostic meta-learning

- Solve new tasks faster by learning a model *initialization* from similar tasks
 - Current initialization θ
 - On K examples/task, evaluate $\nabla_{\theta} L_{T_i}(f_{\theta})$
 - Update weights for $\theta_1, \theta_2, \theta_3$
 - Update θ to minimize sum of per-task losses
 - Repeat
- $$\theta \leftarrow \theta - \beta \nabla_{\theta} \sum_{T_i \sim p(T)} \mathcal{L}_{T_i}(f_{\theta'_i})$$



Model-agnostic meta-learning

- Solve new tasks faster by learning a model *initialization* from similar tasks
 - Current initialization θ
 - On K examples/task, evaluate $\nabla_{\theta} L_{T_i}(f_{\theta})$
 - Update weights for $\theta_1, \theta_2, \theta_3$
 - Update θ to minimize sum of per-task losses
 - Repeat
- $$\theta \leftarrow \theta - \beta \nabla_{\theta} \sum_{T_i \sim p(T)} \mathcal{L}_{T_i}(f_{\theta'_i})$$

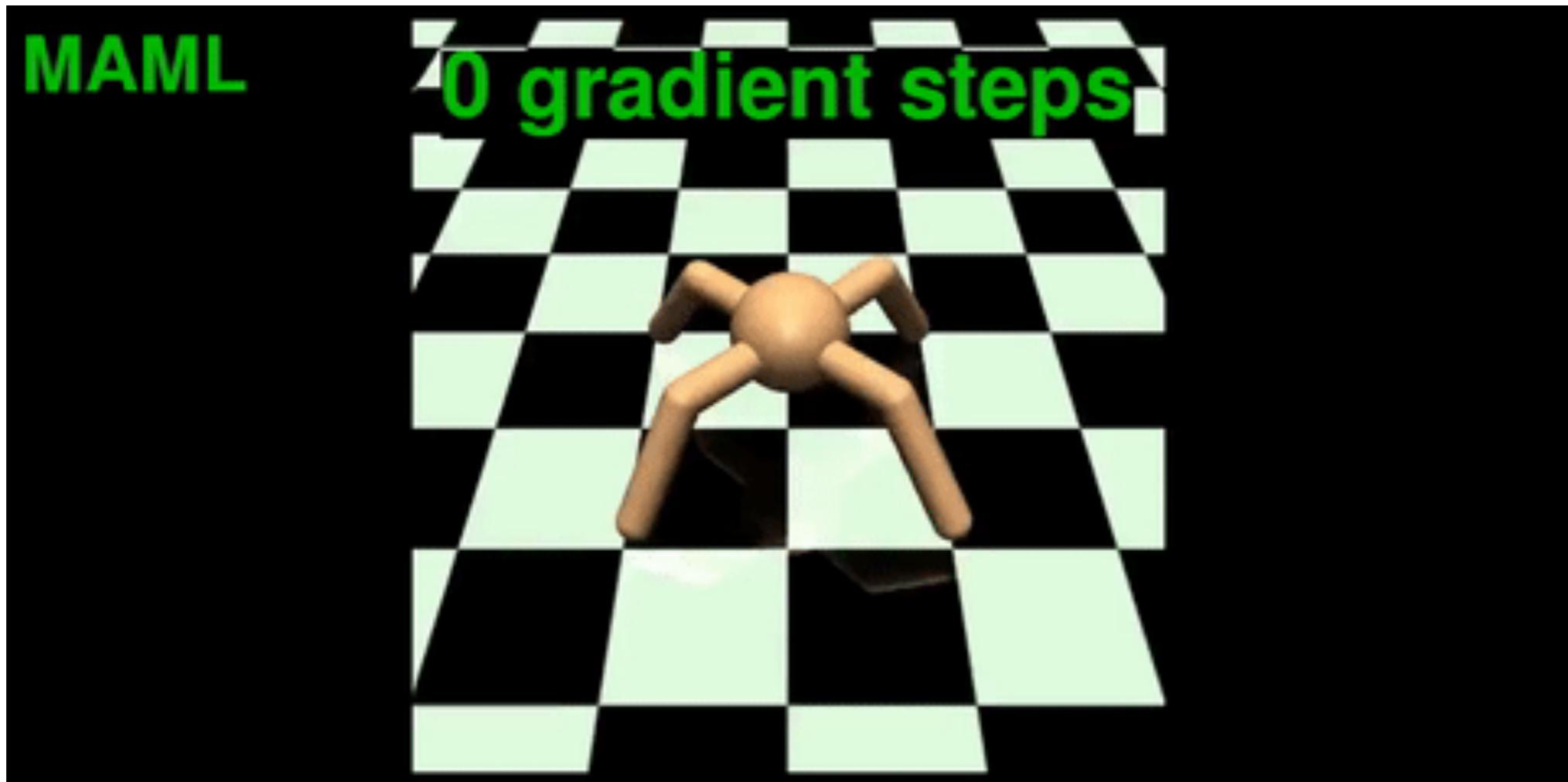


Model-agnostic meta-learning

- More resilient to overfitting
- Generalizes better than LSTM approaches for few shot learning
- *Universality*¹: no theoretical downsides in terms of expressivity when compared to alternative meta-learning models.
- Many variants and applications: *Finn & Levine 2019*
 - REPTILE: do SGD for k steps in one task, only then update init. weights²
 - PLATIPUS: probabilistic MAML
 - Bayesian MAML (Bayesian ensemble)
 - Online MAML
 - ...

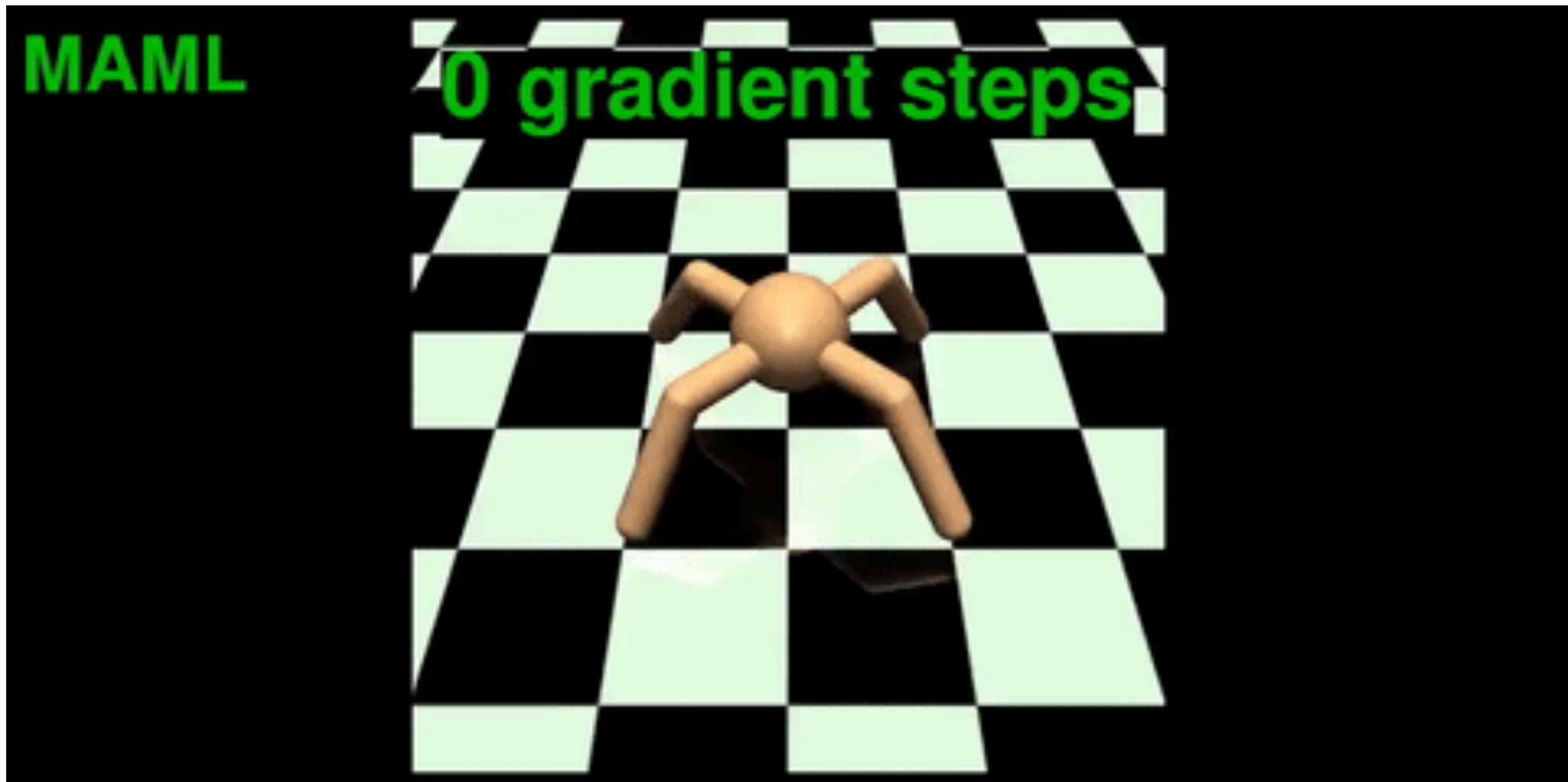
Model-agnostic meta-learning

- For reinforcement learning:



Model-agnostic meta-learning

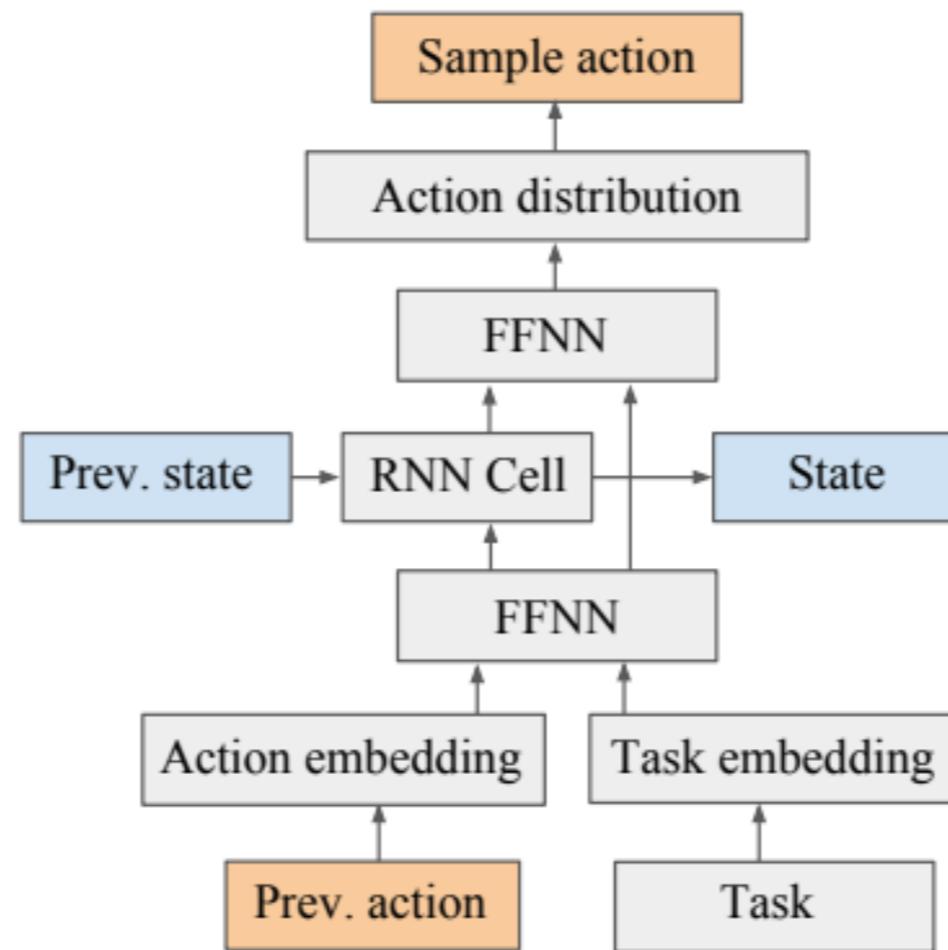
- For reinforcement learning:



Neural Architecture Meta-Learning

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

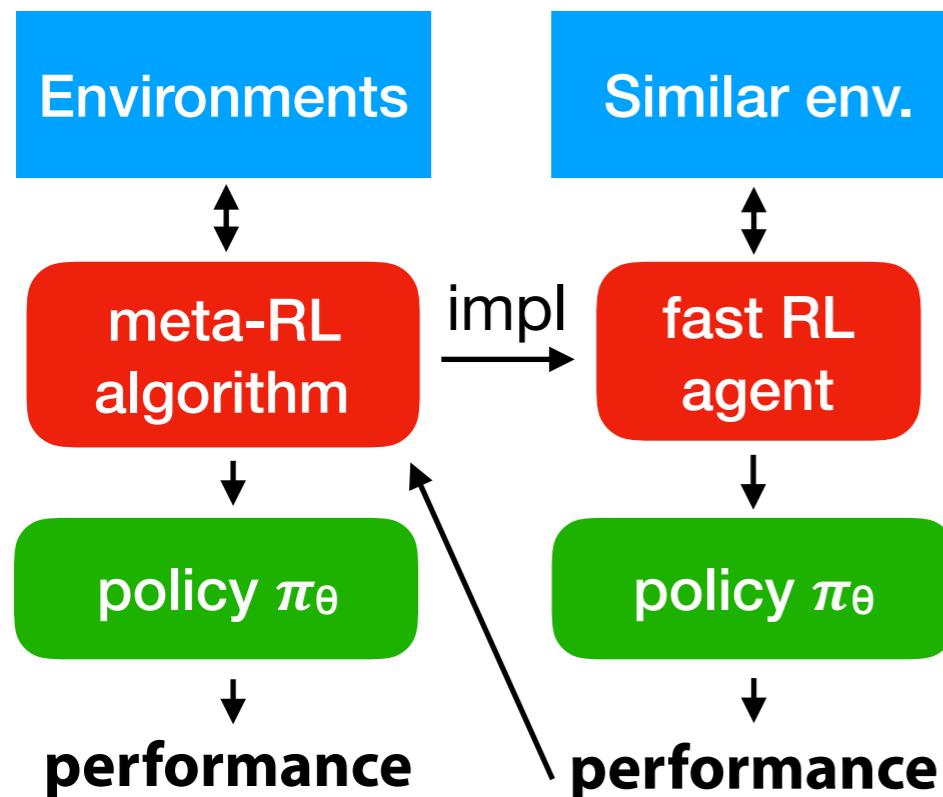
Wong et al. 2018



- Another idea: warm-start DARTS with task-specific one-shot model

Learning to reinforcement learn

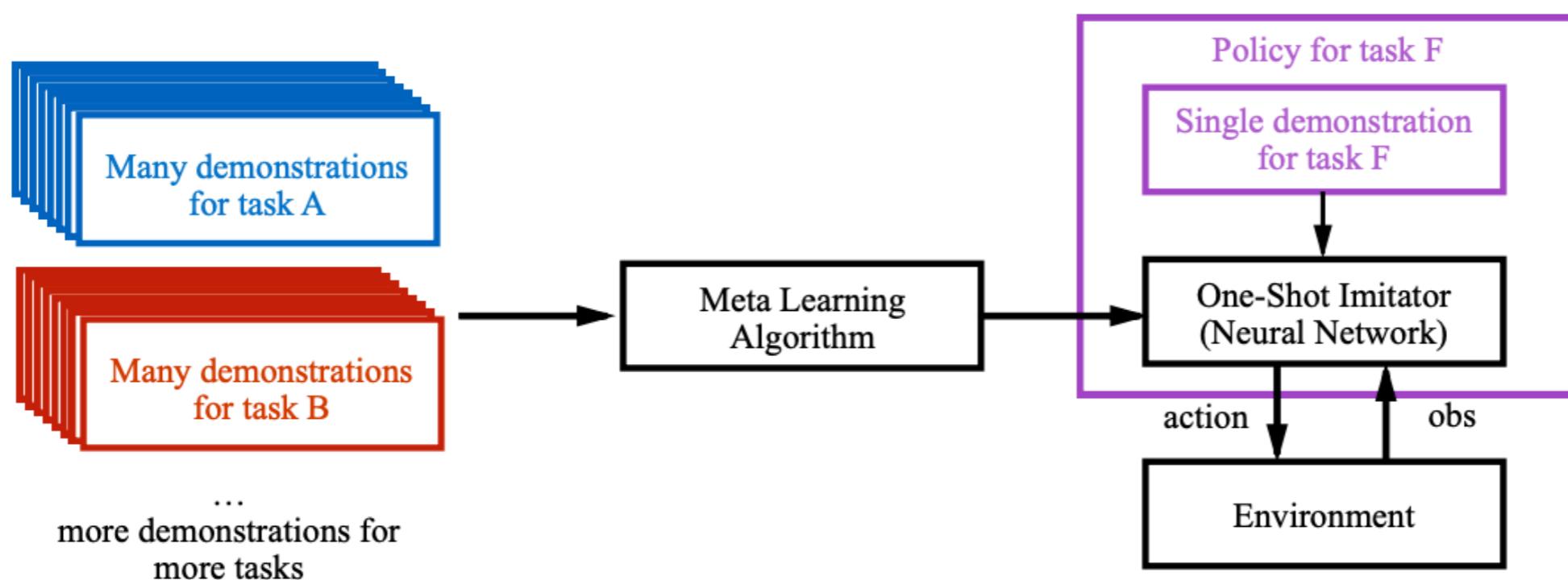
- Humans often learn to play new games much faster than RL techniques do
- Reinforcement learning is very suited for learning-to-learn:
 - Build a learner, then use performance as that learner as a reward



- Learning to reinforcement learn ^{1,2}
 - Use RNN-based deep RL to train a recurrent network on many tasks
 - Learns to implement a 'fast' RL agent, encoded in its weights

Learning to reinforcement learn

- Also works for few-shot learning ³
 - Condition on observation + upcoming demonstration
- You don't know what someone is trying to teach you, but you prepare for the lesson



Learning to learn more tasks

- Active learning *Pang et al. 2018*
 - Deep network (learns representation) + policy network
 - Receives state and reward, says which points to query next
- Density estimation *Reed et al. 2017*
 - Learn distribution over small set of images, can generate new ones
 - Uses a MAML-based few-shot learner
- Matrix factorization *Vartak et al. 2017*
 - Deep learning architecture that makes recommendations
 - Meta-learner learns how to adjust biases for each user (task)
- Replace hand-crafted algorithms by learned ones.
- Look at problems through a meta-learning lens!

Meta-data sharing building a shared memory

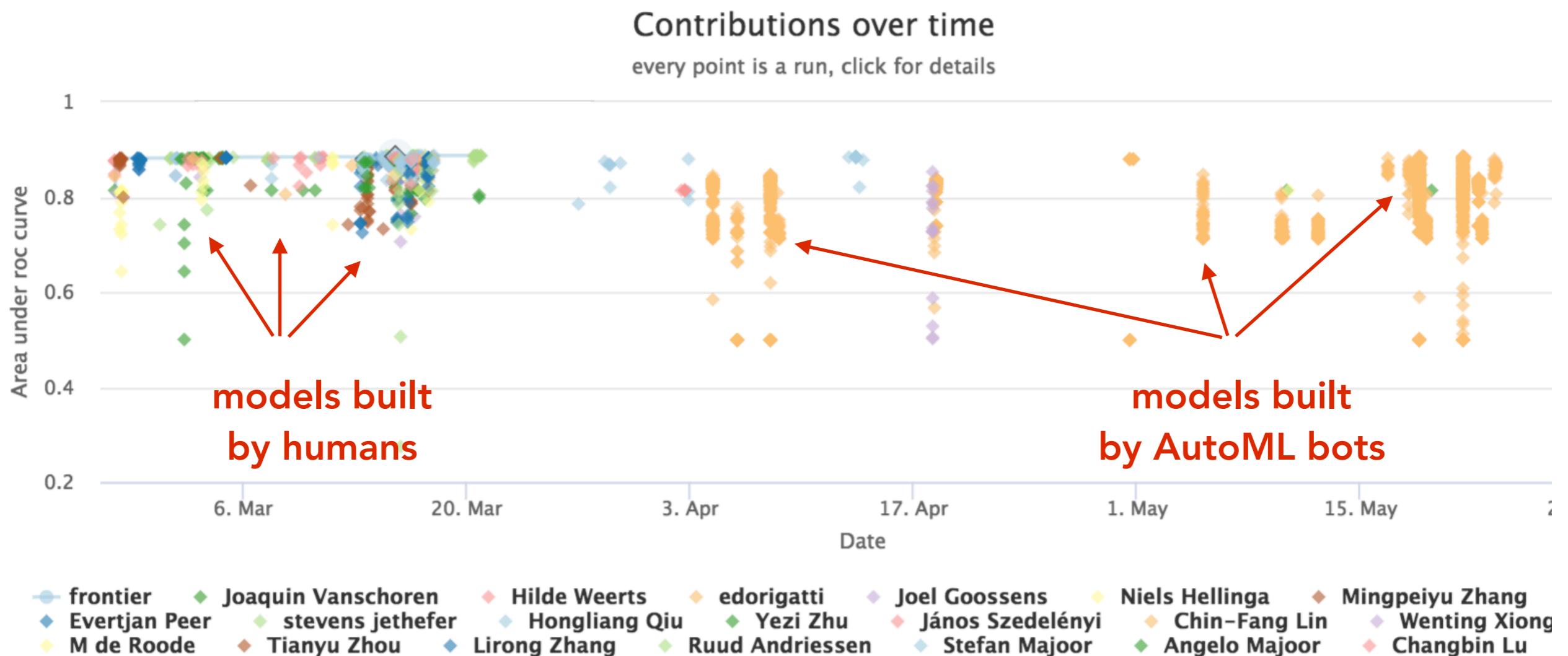
- What if we have a shared memory of all machine learning experiments?
- [OpenML.org](#)
 - Thousands of uniform datasets, 100+ meta-features
 - Millions of evaluated runs
 - Same splits, 30+ metrics
 - Traces, models (*opt*)
- APIs in Python, R, Java,... Sklearn, Keras, PyTorch, MXNet,...
- Publish your own runs
- Never ending learning
- Benchmarks

```
import openml as oml
from sklearn import tree

task = oml.tasks.get_task(14951)
clf = tree.ExtraTreeClassifier()
flow = oml.flows.sklearn_to_flow(clf)
run = oml.runs.run_flow_on_task(task, flow)
myrun = run.publish()
```

Meta-data sharing building a shared memory

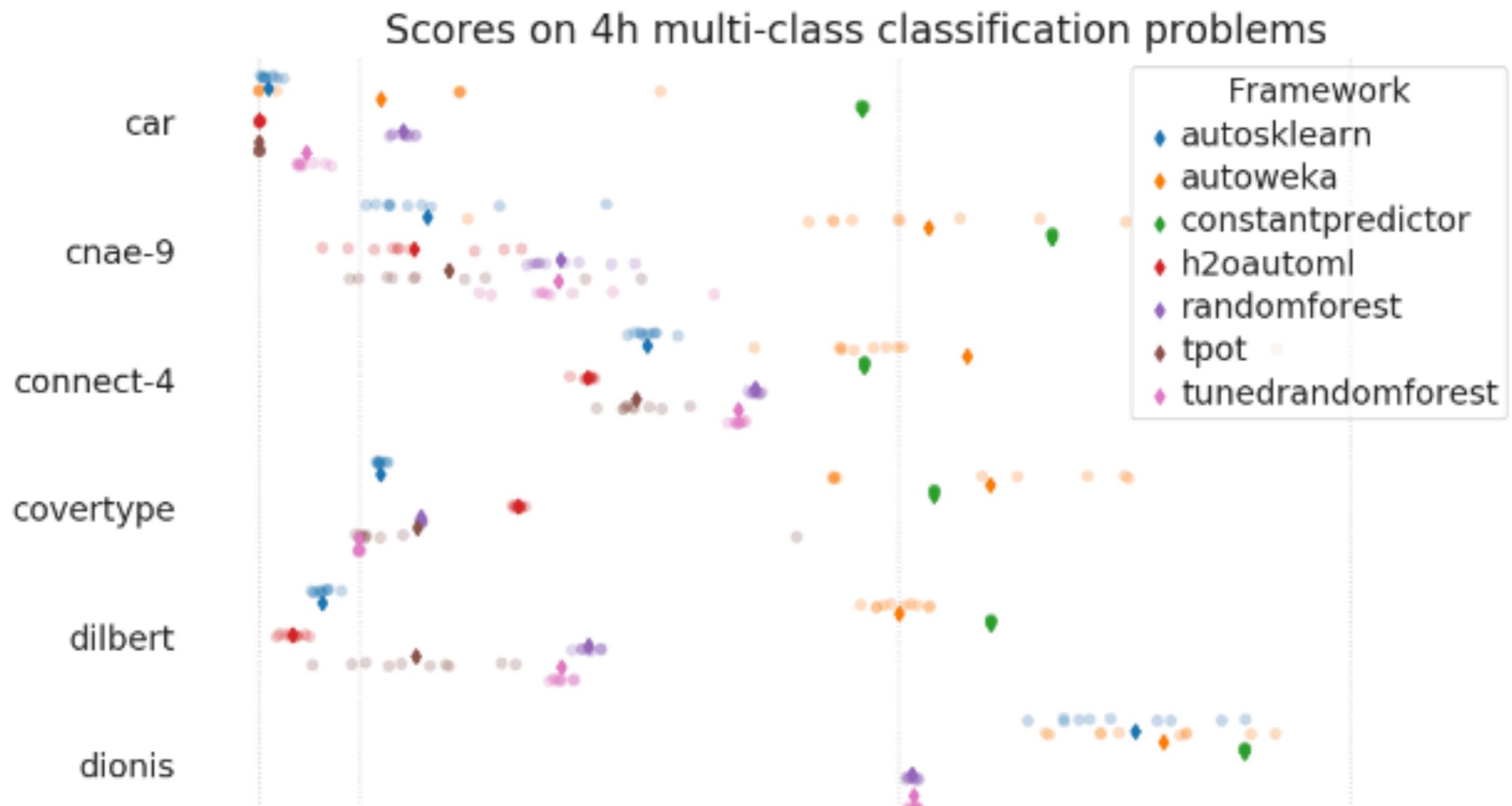
- Train and run AutoML bots on any OpenML dataset
- Never-ending learning (from robots and humans)



AutoML tools and benchmarks

Open source benchmark on <https://openml.github.io/automlbenchmark/>

- On OpenML datasets, will regularly be updated, anyone can add



Scores on 4h multi-class classification problems



AutoML open source tools

	Architecture search	Hyperparameter search	Improvements	Meta-learning
<u>Auto-WEKA</u>	Parameterized pipeline	Bayesian Optimization (RF)		
<u>auto-sklearn</u>	Parameterized pipeline	Bayesian Optimization (RF)	Ensembling	warm-start
BO-HB	Parameterized pipeline	Tree of Parzen Estimators	Ensembling, Hyperband	
<u>hyperopt-sklearn</u>	Parameterized pipeline	Tree of Parzen Estimators		
<u>TPOT</u>	Evolving pipelines (trees)	Genetic programming		
<u>GAMA</u>	Evolving pipelines (trees)	Asynchronous evolution	Ensembling, Hyperband	
<u>H2O AutoML</u>	Single algorithms	random search	Stacking	
<u>ML-Plan</u>	Planning-based pipeline	Hierarchical planner		
<u>OBOE</u>	Single algorithms	Low rank approximation	Ensembling	Runtime predictor

AutoML tools and benchmarks

Observations so far (for pipeline search):

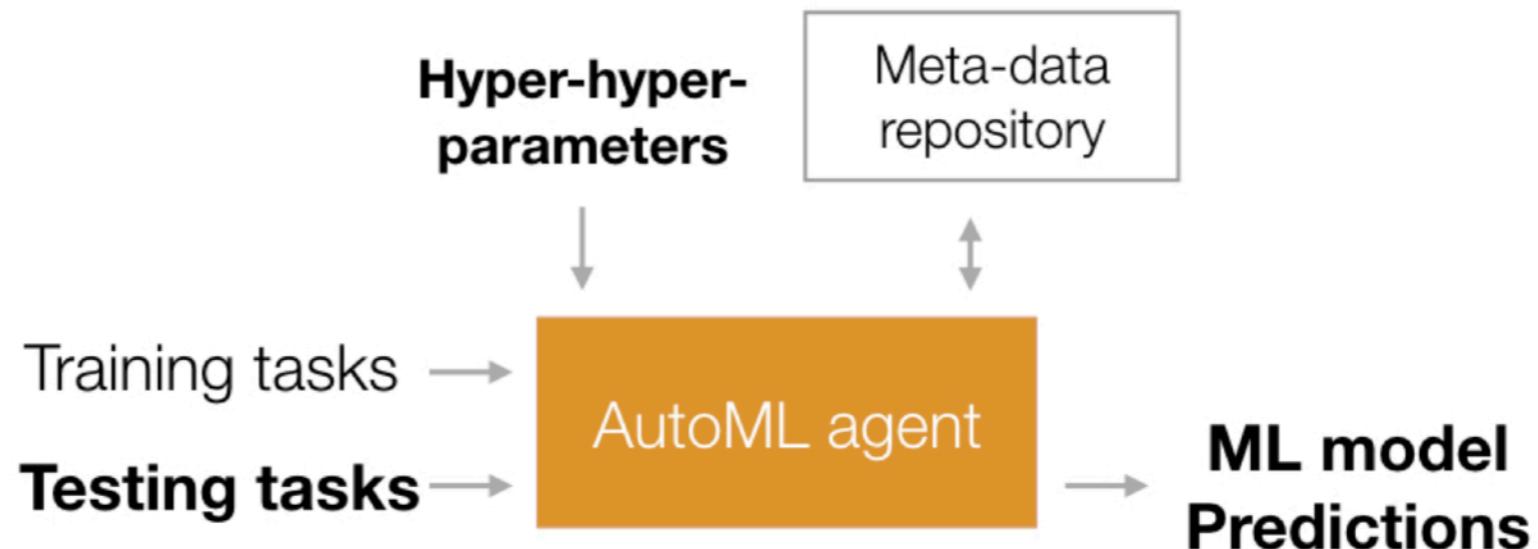
- No AutoML system outperforms all others (Auto-WEKA is worse)
- Auto-sklearn, H2O, TPOT can outperform each other on given datasets
- Tuned RandomForests are a strong baseline
- High numbers of classes, unbalanced classes, are a problem for most tools
- Little support for anytime prediction ...

We need good benchmarks for NAS as well...

AutoML Gym

Environment to train general-purpose AutoML algorithms

- Open benchmark, meta-learning, continual learning, efficient NAS



Open PostDoc position!
<http://bit.ly/automl-gym>

Towards human-like learning to learn

- Learning-to-learn gives humans a significant advantage
 - **Learning how to learn any task empowers us far beyond knowing how to learn specific tasks.**
 - It is a **universal** aspect of life, and how it evolves
 - Very exciting field with many unexplored possibilities
 - Many aspects not understood (e.g. task similarity), need more experiments.
- **Challenge:**
 - Build learners that **never stop learning**, that **learn from each other**
 - *Fast* learning by *slow* meta-learning
 - Build a ***global memory*** for learning systems to learn from
 - **Let them explore / experiment by themselves**

Thank you!

Never stop learning



more to learn

<http://www.automl.org/book/>
Chapter 2: Meta-Learning

special thanks to

Pavel Brazdil, Matthias Feurer, Frank Hutter, Erin Grant,
Hugo Larochelle, Raghu Rajan, Jan van Rijn, Jane Wang