Automated Machine Learning (AutoML): A Tutorial

Matthias Feurer

University of Freiburg

feurerm@cs.uni-freiburg.de

Thomas Elsken

Bosch Center for Artificial Intelligence
& University of Freiburg

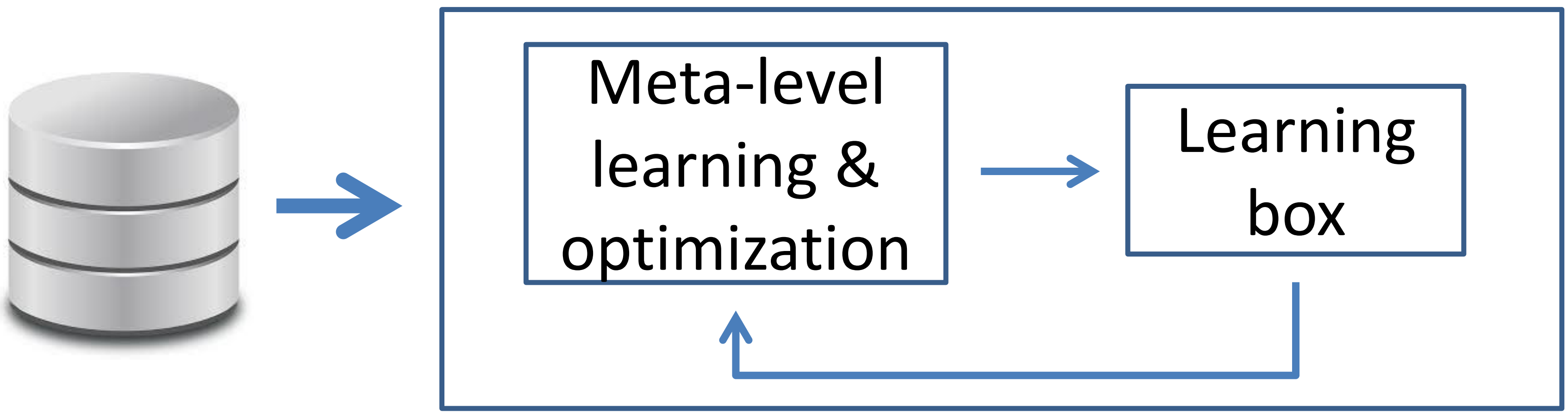
Thomas.Elsken@de.bosch.com

Outline

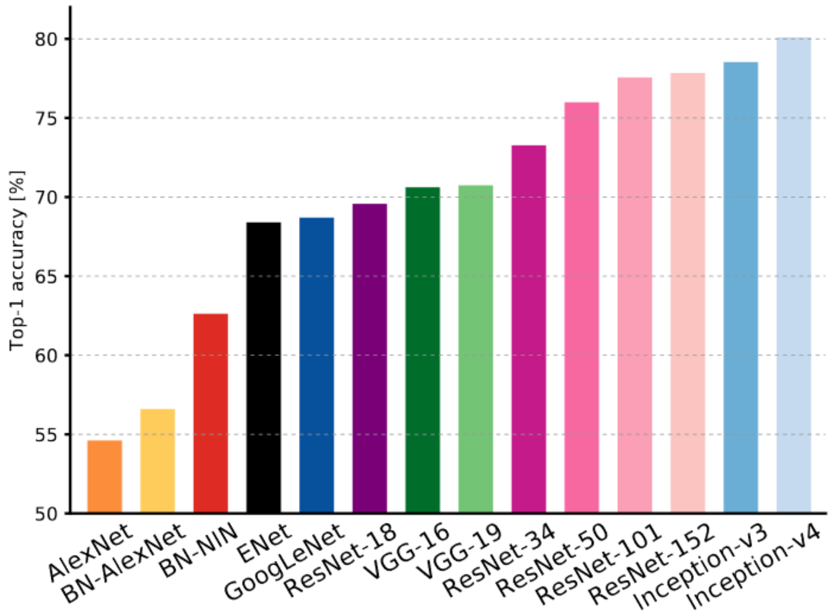
1. General AutoML
2. Neural Architecture Search
3. Meta Learning & Learning to Learn

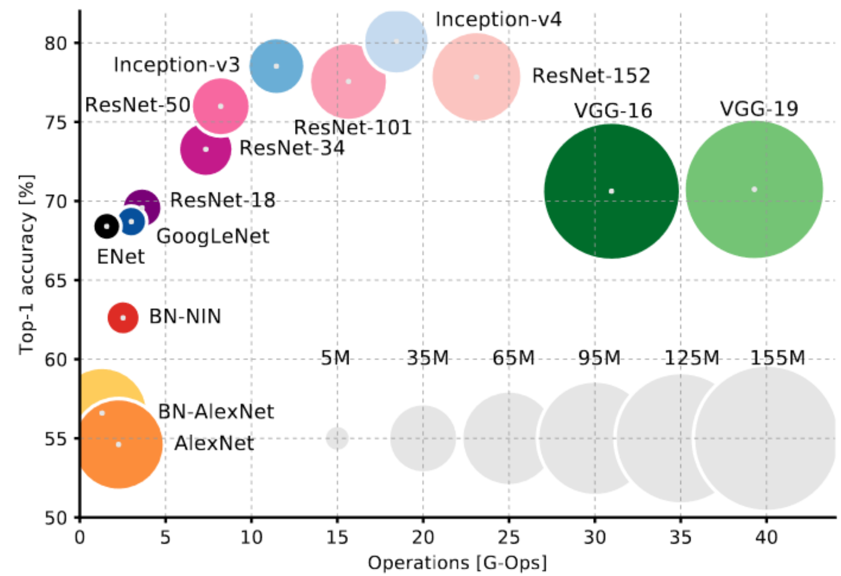
For more details, see: automl.org/book

AutoML: true end-to-end learning



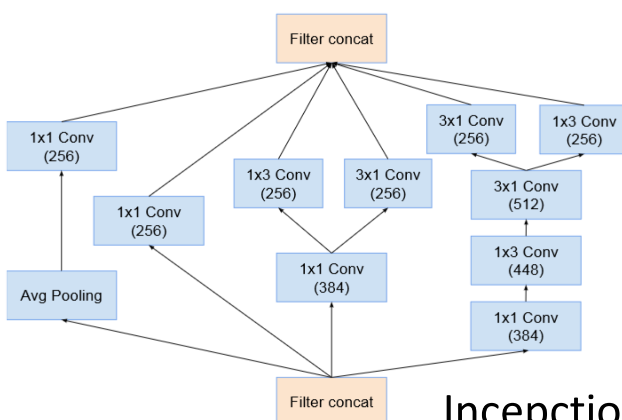
Neural Architecture Search - Motivation

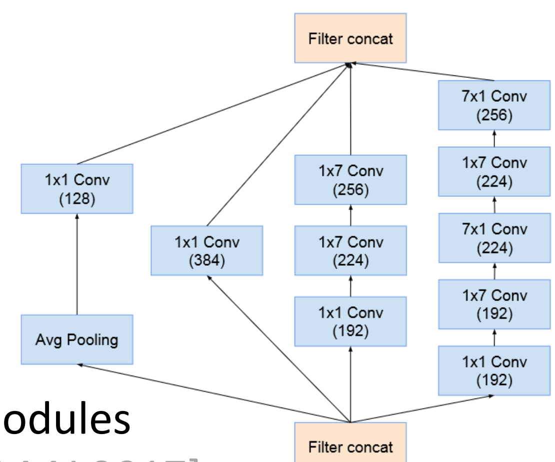




[Canziani et al., preprint 2017]

Bigger,
more complex
architectures…





Incepction-V4 modules

[Szegedy et al., AAAI 2017]

Slide courtesy Nikhil Naik

Neural Architecture Search - Motivation

Can we automatically design neural network architectures?

Outline

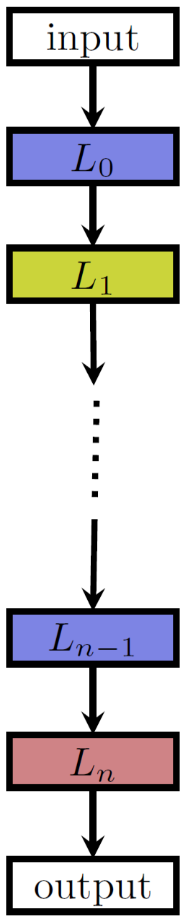
Neural Architecture Search

- Search Space Design
- Blackbox Optimization
- Beyond Blackbox Optimization

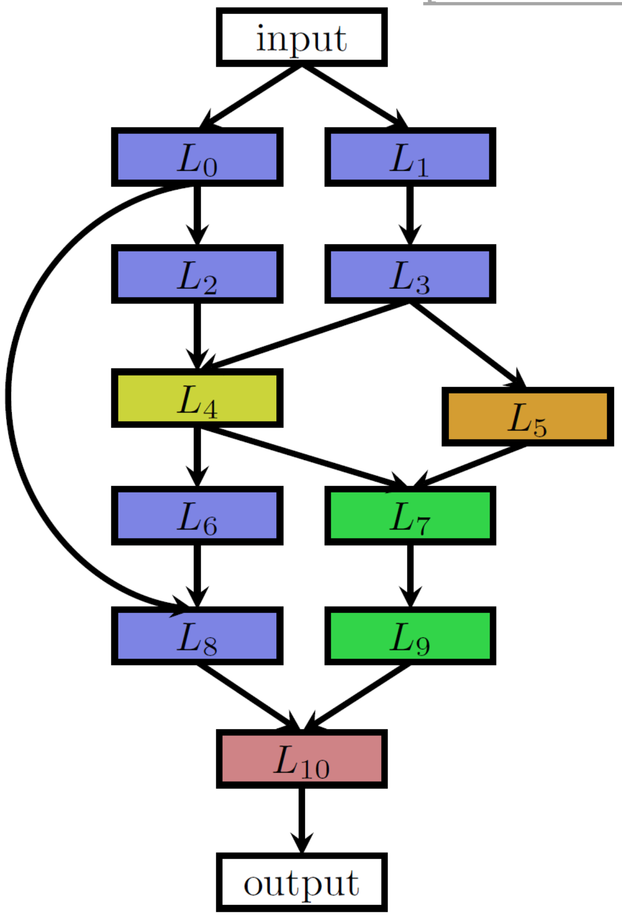
Based on: Elsken, Metzen and Hutter
[Neural Architecture Search: a Survey, JMLR 2019; also Chapter 3 of the AutoML book]

Basic Neural Architecture Search Spaces

[Elsken et al., JMLR 2019]



Chain-structured space
(different colours:
different layer types)



More complex space
with multiple branches
and skip connections

Cell Search Spaces

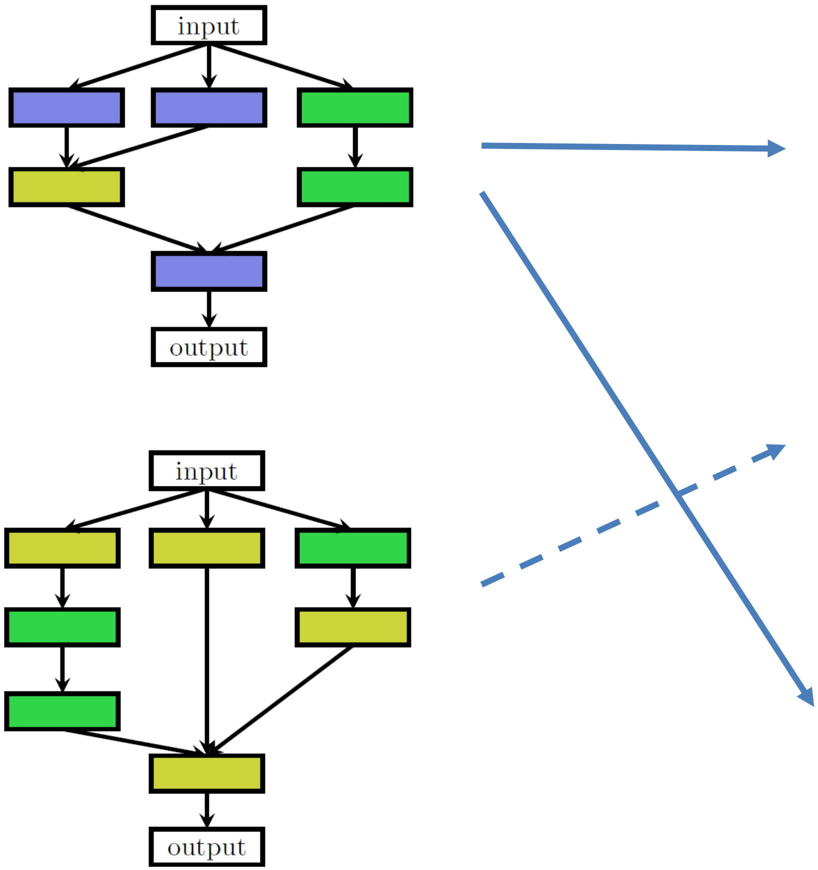
Introduced by Zoph et al. [CVPR 2018]

Two possible cells

Architecture composed of stacking together individual cells

normal cell:
preserves spatial resolution

reduction cell:
reduce spatial resolution



Outline

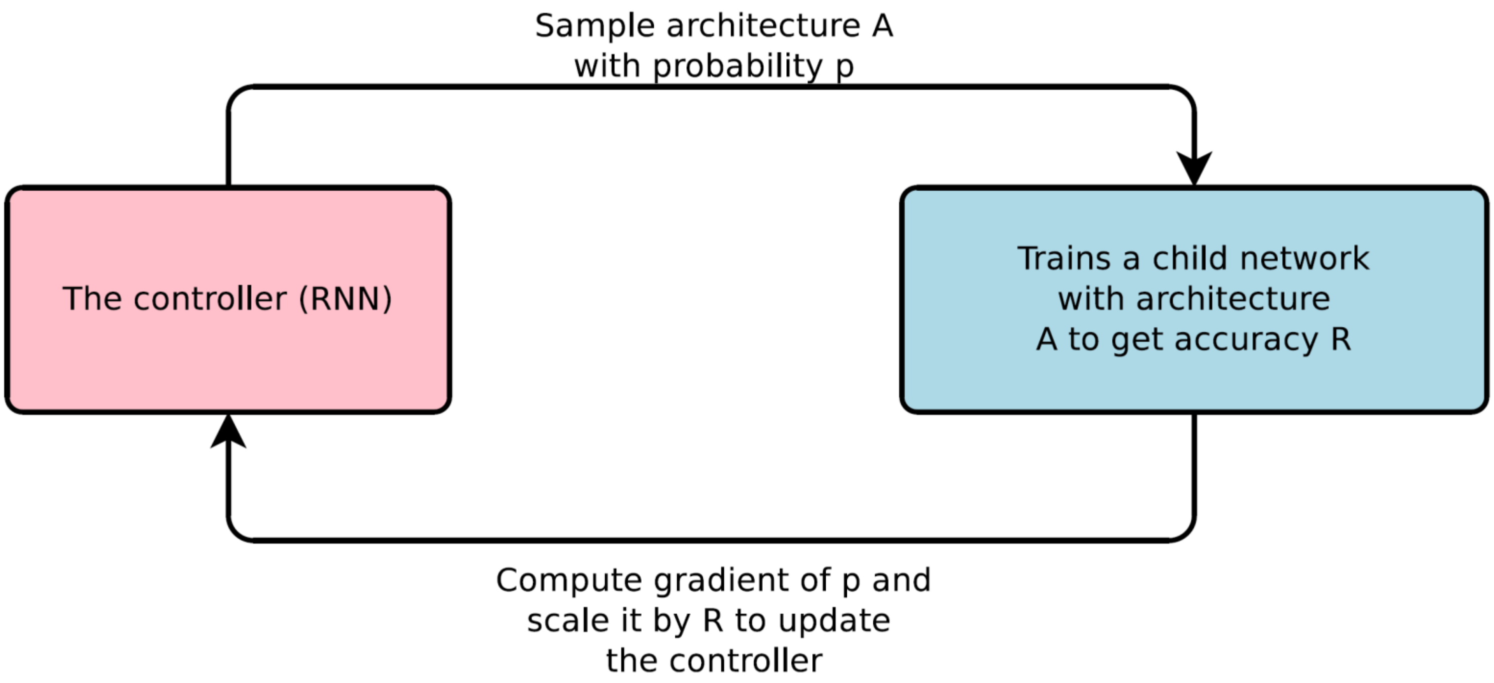
Neural Architecture Search

- Search Space Design
- Blackbox Optimization
- Beyond Blackbox Optimization

Based on: Elsken, Metzen and Hutter
[Neural Architecture Search: a Survey, JMLR 2019; also Chapter 3 of the AutoML book]

NAS with Reinforcement Learning

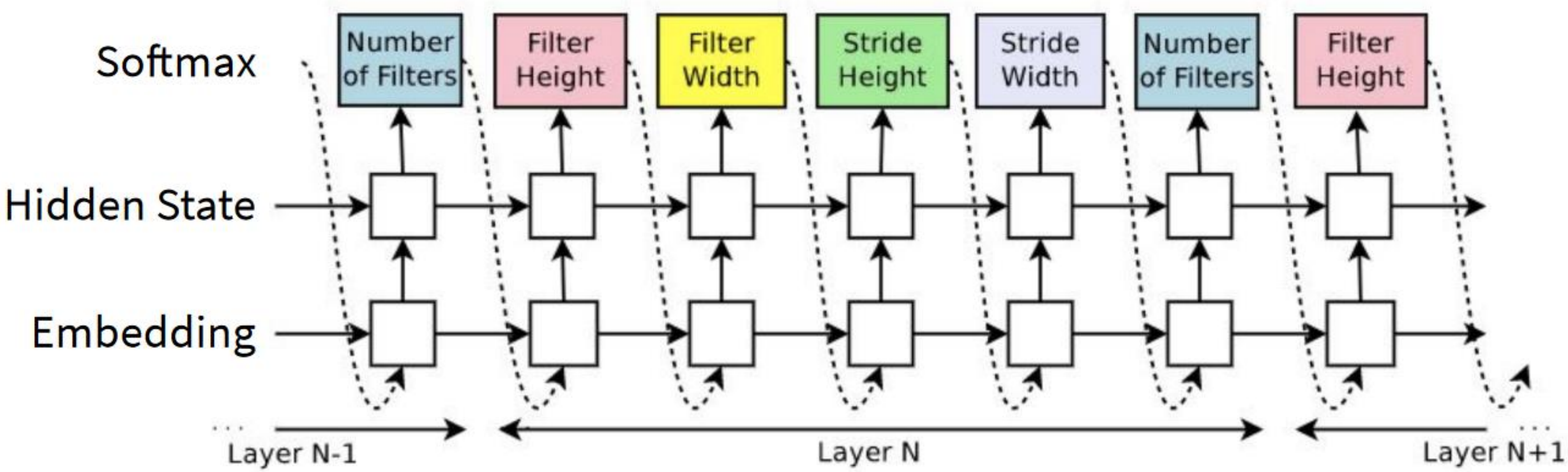
- NAS with Reinforcement Learning [Zoph & Le, ICLR 2017]
  - State-of-the-art results for CIFAR-10, Penn Treebank
  - Large computational demands:
    800 GPUs for 3-4 weeks, 12.800 architectures trained



NAS with Reinforcement Learning

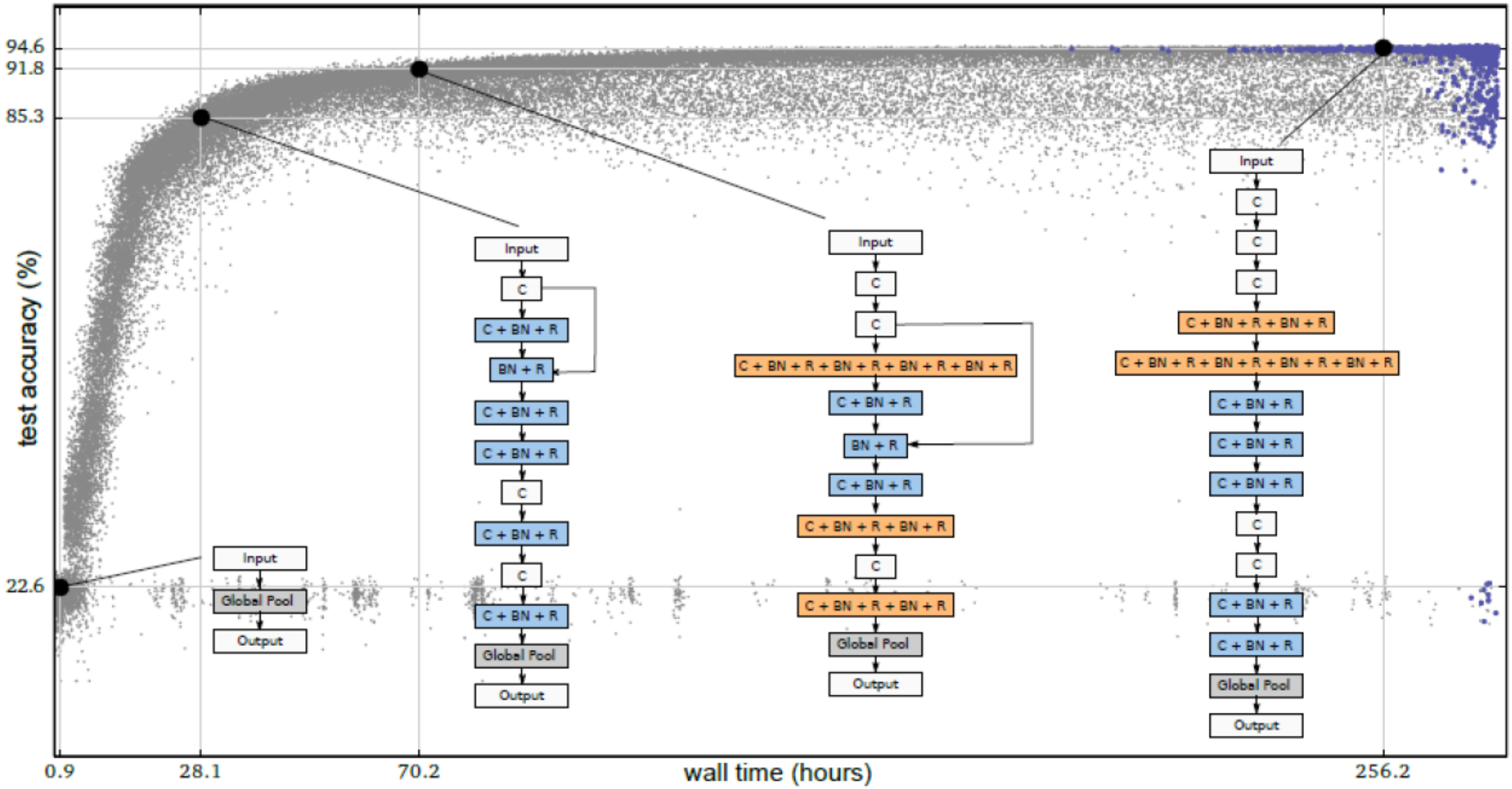
[Zoph & Le, ICLR 2017]

- architecture of neural network represented as string
  e.g., ["filter height: 5", "filter width: 3", "# of filters: 24"]
- controller (RNN) generates string that represents architecture



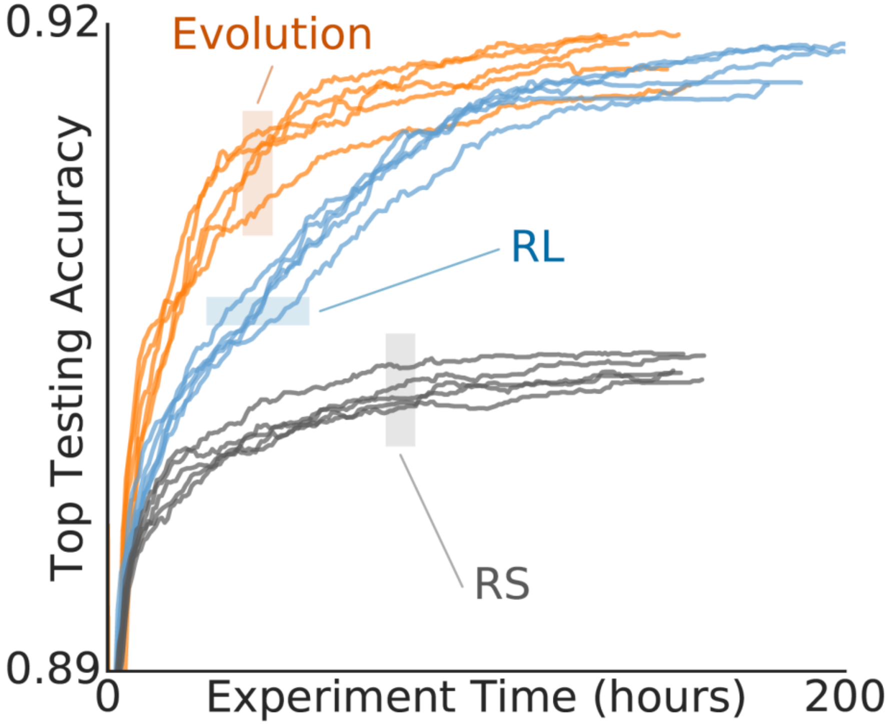
NAS with Evolution

- Neuroevolution (already since the 1990s [Angeline et al., 1994; Stanley and Miikkulainen, 2002])
  - Mutation steps, such as adding, changing or removing a layer [Real et al., ICML 2017; Miikkulainen et al., arXiv 2017]

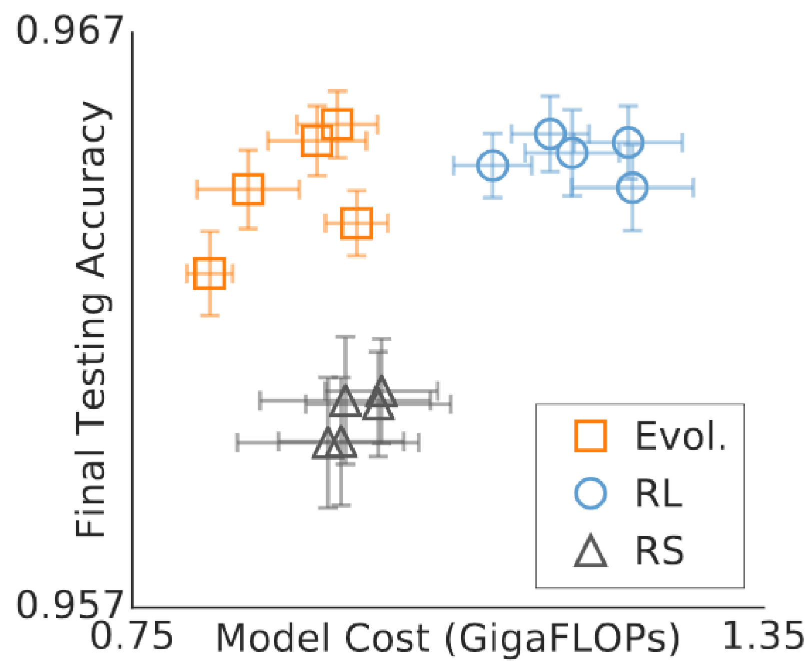


RL vs. Evolution vs. Random Search

during architecture search



final evaluation



[Real et al., AAAI 2019]

Bayesian Optimization

- Joint optimization of a vision architecture with 238 hyperparameters with TPE [Bergstra et al, ICML 2013]
- Auto-Net
  - Joint architecture and hyperparameter search with SMAC
  - First Auto-DL system to win a competition dataset against human experts [Mendoza et al, AutoML 2016]
- Kernels for GP-based NAS
  - Arc kernel [Swersky et al, BayesOpt 2013]
  - NASBOT [Kandasamy et al, NIPS 2018]
- Sequential model-based optimization
  - PNAS [Liu et al, ECCV 2018]

Some numbers (Cifar-10)

	Reference	Error (%)	Params (Millions)	GPU Days
RL	Zoph and Le (2017)	3.65	37.4	22,400
	Zoph et al. (2018)	3.41	3.3	2,000
EA	Real et al. (2017)	5.40	5.4	2,600
	Real et al. (2019)	3.34	3.2	3,150

Going to cell search space

[Wistuba et al., preprint 2019]

Motivation

Blackbox optimization is expensive!
Can we do better?

Outline

Neural Architecture Search

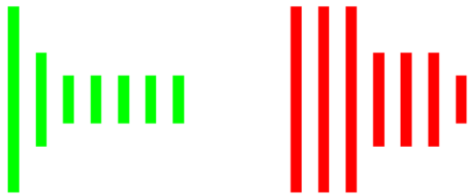
- Search Space Design
- Blackbox Optimization
- Beyond Blackbox Optimization

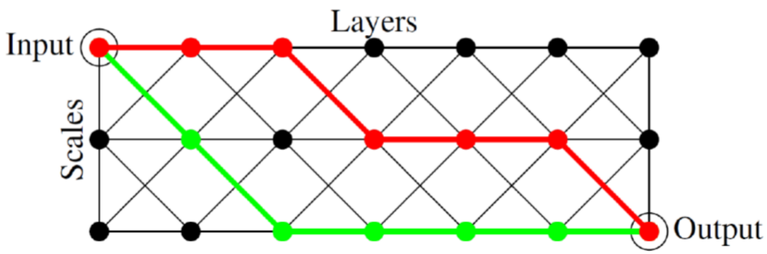
Based on: Elsken, Metzen and Hutter
[Neural Architecture Search: a Survey, JMLR 2019; also Chapter 3 of the AutoML book]

Main approaches for making NAS efficient

- Weight inheritance & network morphisms

- Weight sharing & one-shot models





- Multi-fidelity optimization
  - do search on smaller models, less training epochs, fewer data,…
- Meta-learning
  - Avoid running NAS from scratch for every task

Weight inheritance & network morphisms

- Network morphisms [Chen et al., 2016; Wei et al., 2016]
  - Change the network structure, but not the modelled function (i.e., for every input the network yields the same output as before applying the network morphism)

  - Can use this in NAS algorithms as operations to generate new networks
  - Avoids costly training from scratch

Network morphism example

we have trained network

$$N_1(x) = Softmax_{w_{1,1}} \circ ReLU \circ Conv_{w_{1,2}}(x)$$

More details in our blog post.

want to add another Relu-Conv block

$$N_2(x) = Softmax_{w_{2,1}} \boxed{\circ ReLU \circ Conv_{w_{2,2}}} \circ ReLU \circ Conv_{w_{2,3}}(x)$$

copy

$$w_{2,1} = w_{1,1}, \quad w_{2,3} = w_{1,2}$$

and set $w_{2,2}$ so that $Conv_{w_{2,2}}(x) = x$

Then:

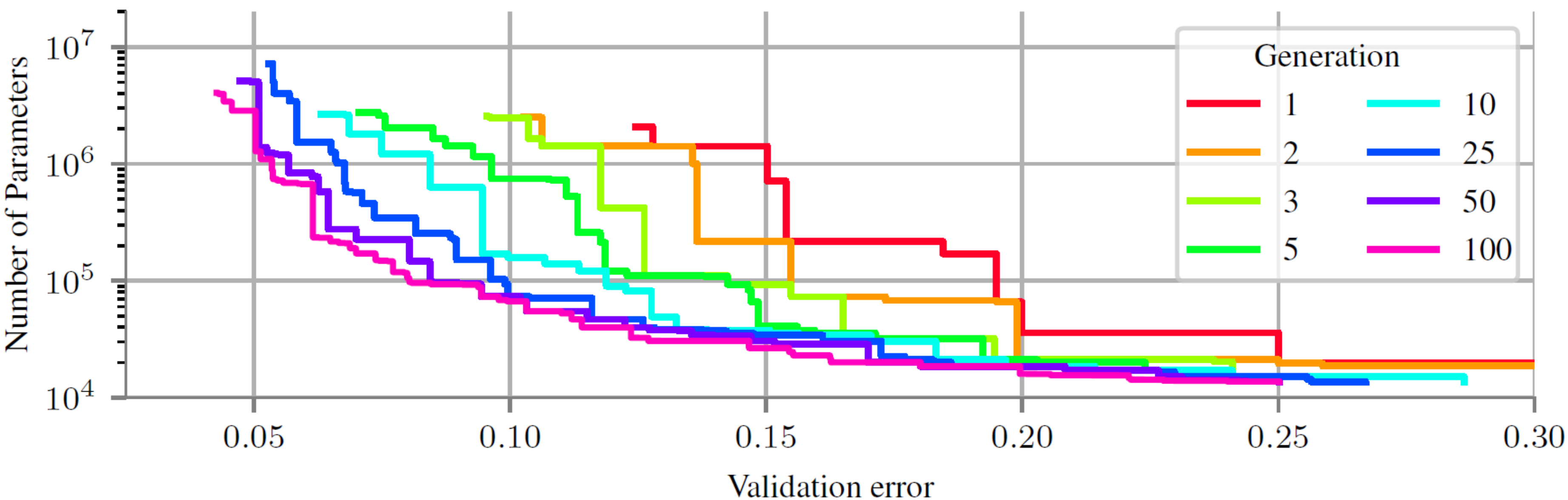
Weight inheritance & network morphisms

[Cai et al, AAAI 2018; Elsken et al, NeurIPS MetaLearn 2017; Cortes et al, ICML 2017; Cai et al, ICML 2018; Elsken et al, ICLR 2019]

$\rightarrow$ enables efficient architecture search

Designing Efficient Architectures

- Multi-objective NAS: LEMONADE [Elsken et al., ICLR 2019, blog post]
  - Multi-objective evolutionary method
  - Objectives such as accuracy, # parameters, # flops, latency
  - Outputs Pareto-front wrt. multiple objectives
  - No need to specify tradeoff between objectives a-priori



More work on designing efficient architectures, e.g.:
NAS for compression [Cao et al., ICLR 2019; He et al., ECCV 2018], NAS with hardware-constrained objectives [Tan et al., CVPR 2019, Tan et al., ICML 2019, Cai et al., ICLR 2019]

Some numbers (Cifar-10)

	Reference	Error (%)	Params (Millions)	GPU Days
RL	Baker et al. (2017)	6.92	11.18	100
	Zoph and Le (2017)	3.65	37.4	22,400
	Cai et al. (2018a)	4.23	23.4	10
	Zoph et al. (2018)	3.41	3.3	2,000
	Zoph et al. (2018) + Cutout	2.65	3.3	2,000
	Zhong et al. (2018)	3.54	39.8	96
	Cai et al. (2018b)	2.99	5.7	200
	Cai et al. (2018b) + Cutout	2.49	5.7	200
EA	Real et al. (2017)	5.40	5.4	2,600
	Xie and Yuille (2017)	5.39	N/A	17
	Suganuma et al. (2017)	5.98	1.7	14.9
	Liu et al. (2018b)	3.75	15.7	300
	Real et al. (2019)	3.34	3.2	3,150
	Elsken et al. (2018)	5.2	19.7	1
	Wistuba (2018a) + Cutout	3.57	5.8	0.5

NAS with weight inher. / network morphisms

[Wistuba et al., preprint 2019]

Weight Sharing & One-shot Models

- embed architectures from search space into single network, the „*one-shot model*“
- each path through the one-shot model is an architecture
- solely need a single training of the one-shot model
- weights are shared across architectures embedded in one-shot model

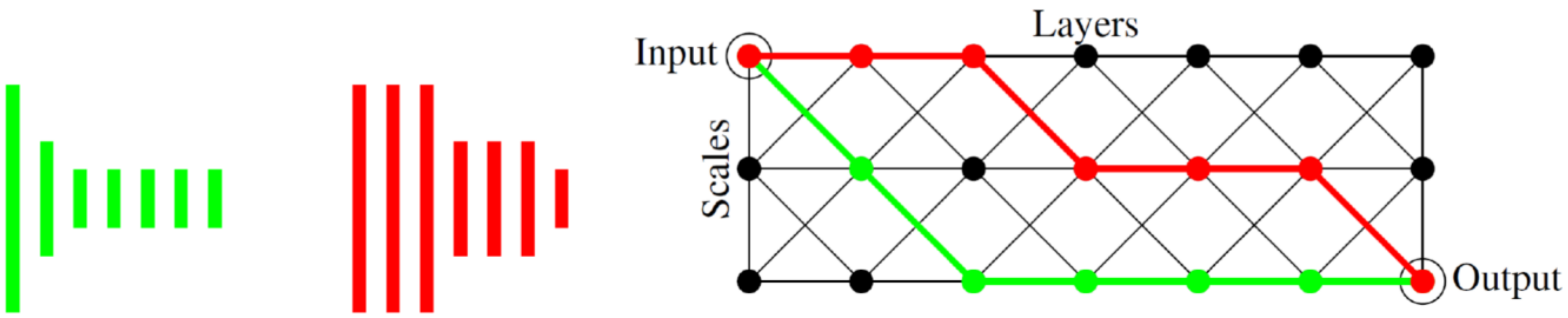
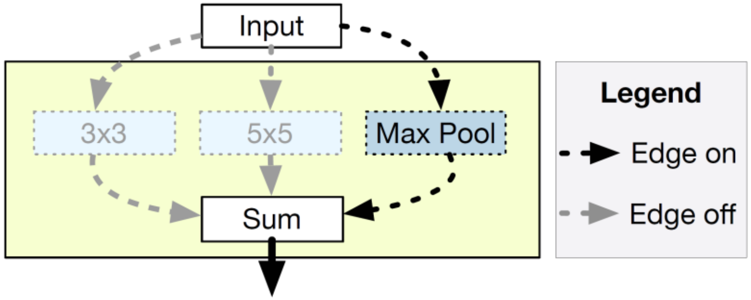


Figure: embeddings of two 7-layer CNNs (red, green) [Saxena & Verbeek, NeurIPS 2016]

- Problems/ limitations:
  - Search space restricted to one-shot model
  - One-shot model needs to be kept in GPU-memory
  - Search bias?

Weight Sharing & One-shot Models

- Simplifying One-Shot Architecture Search [Bender et al., ICML 2018]
  - Use path dropout to make sure the individual models perform well by themselves

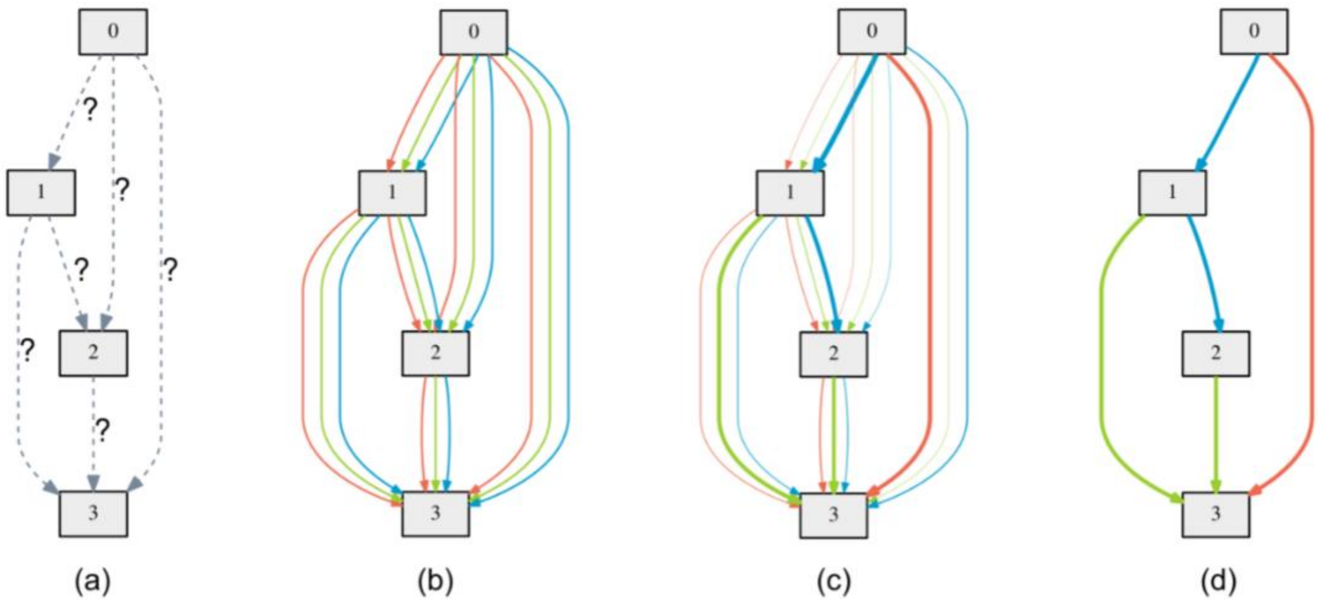


- ENAS [Pham et al., ICML 2018]
  - Use RL to sample paths (=architectures) from one-shot model

- SMASH [Brock et al., MetaLearn 2017]
  - Train hypernetwork that generates weights of models

DARTS: Differentiable Architecture Search

[Liu et al., ICLR 2019]



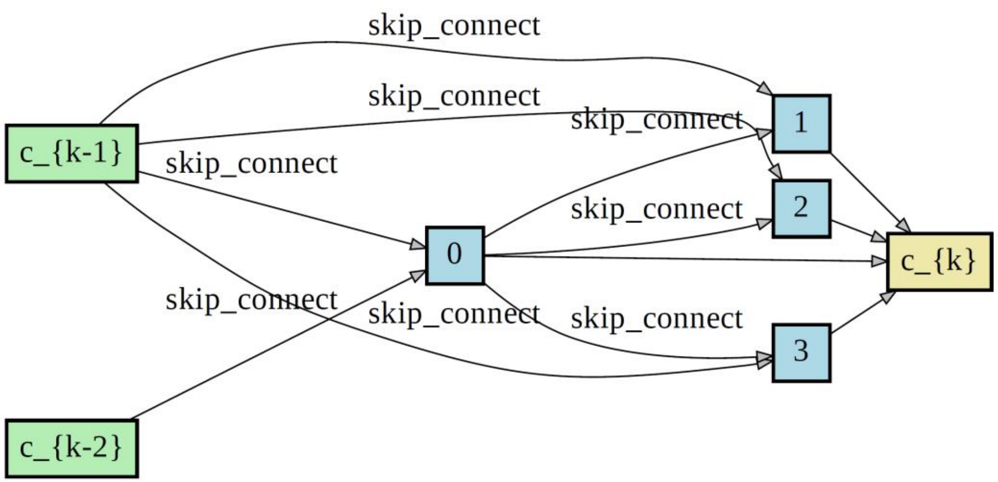
- Relax the discrete NAS problem (a->b)
  - One-shot model with continuous architecture weight α for each operator
  - mixed operator: $$\bar{o}^{(i,j)}(x) = \sum_{o \in \mathcal{O}} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o' \in \mathcal{O}} \exp(\alpha_{o'}^{(i,j)})} o(x)$$
- Solve a bi-level optimization problem (c)

$$\min_{\alpha} \quad \mathcal{L}_{val}(w^*(\alpha), \alpha)$$
$$\text{s.t.} \quad w^*(\alpha) = \text{argmin}_w \; \mathcal{L}_{train}(w, \alpha)$$

- In the end, discretize to obtain a single architecture (d)

DARTS: Differentiable Neural Architecture Search

- Very fast:
  - By alternating SGD steps for α and w runtime only a bit higher than SGD for w alone
- Very brittle optimization:
  - Requires hyperparameter tuning for new problems



- One-shot models needs to be kept in GPU memory
- Discretization at end of search degenerates performance; re-training necessary
- Already lots of follow-up work trying to solve these problems [Xie et al., ICLR 2019, Cai et al., ICLR 2019, Dong et al., CVPR 2019]

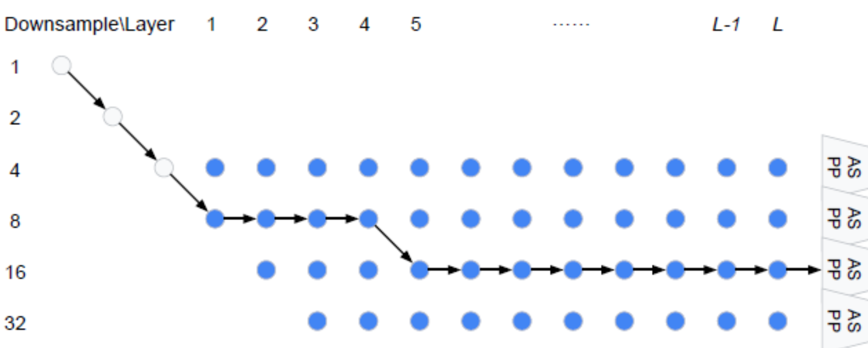
Some numbers (Cifar-10)

	Reference	Error (%)	Params (Millions)	GPU Days
RL	Zoph and Le (2017)	3.65	37.4	22,400
	Cai et al. (2018a)	4.23	23.4	10
	Zoph et al. (2018)	3.41	3.3	2,000
	Zoph et al. (2018) + Cutout	2.65	3.3	2,000
	Cai et al. (2018b)	2.99	5.7	200
	Cai et al. (2018b) + Cutout	2.49	5.7	200
EA	Real et al. (2017)	5.40	5.4	2,600
	Liu et al. (2018b)	3.75	15.7	300
	Real et al. (2019)	3.34	3.2	3,150
	Elsken et al. (2018)	5.2	19.7	1
	Wistuba (2018a) + Cutout	3.57	5.8	0.5
One-Shot	Pham et al. (2018)	3.54	4.6	0.5
	Pham et al. (2018) + Cutout	2.89	4.6	0.5
	Bender et al. (2018)	4.00	5.0	N/A
	Casale et al. (2019) + Cutout	2.81	3.7	1
	Liu et al. (2019b) + Cutout	2.76	3.3	4
	Xie et al. (2019b) + Cutout	2.85	2.8	1.5
	Cai et al. (2019) + Cutout	2.08	5.7	8.33
	Brock et al. (2018)	4.03	16.0	3
	Zhang et al. (2019)	2.84	5.7	0.84
Random	Liu et al. (2018b)	3.91	N/A	300
	Luo et al. (2018)	3.92	3.9	0.3
	Liu et al. (2019b) + Cutout	3.29	3.2	4
	Li and Talwalkar (2019) + Cutout	2.85	4.3	2.7

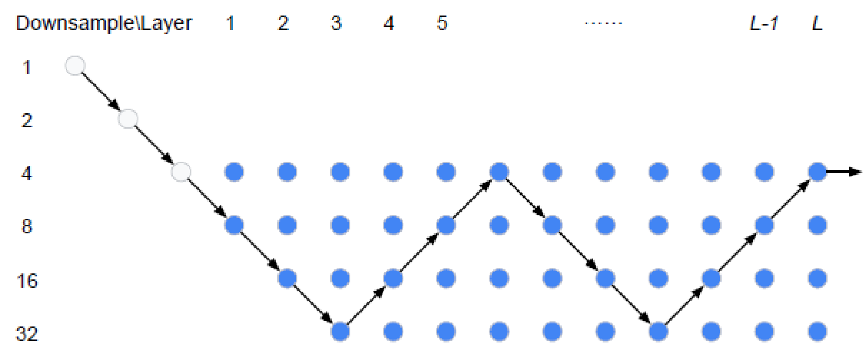
[Wistuba et al., preprint 2019]

NAS for dense prediction tasks

- Auto-DeepLab [Liu et al., CVPR 2019]
  - also optimize downsampling factor for each layer
  - 3 GPU days search on Cityscapes
  - Based on DARTS

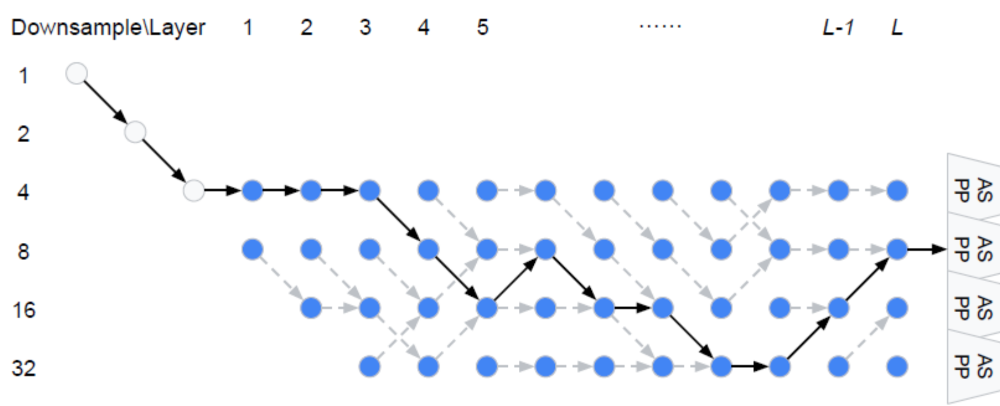


(a) Network level architecture used in DeepLabv3 [9].



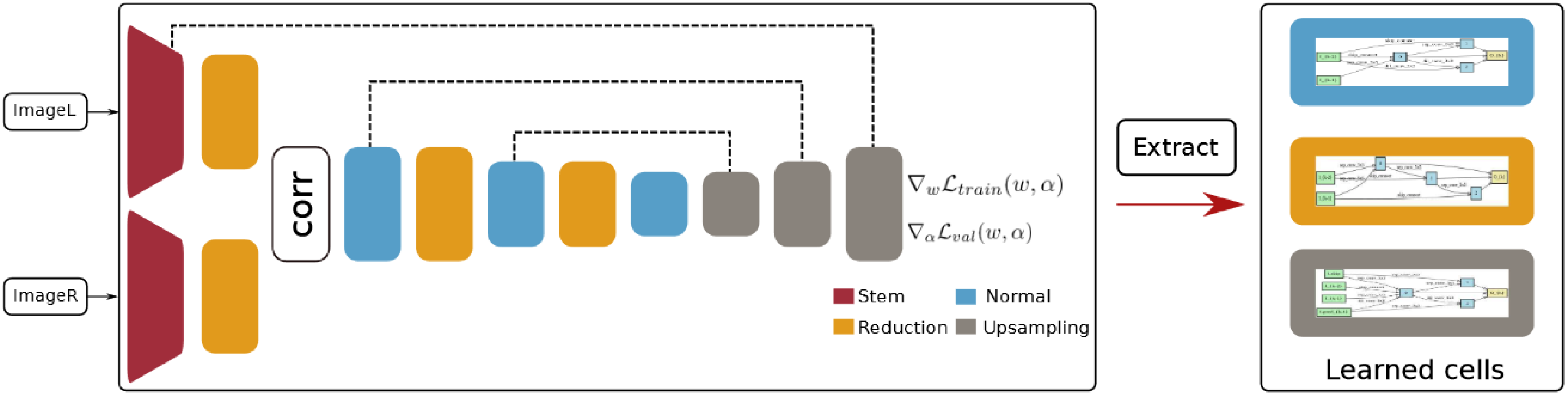
(c) Network level architecture used in Stacked Hourglass [55].

Optimized:
(3 GPU days search on Cityscapes)



NAS for dense prediction tasks

- AutoDispNet [Saikia et al., ICCV 2019]
  - Introduce upsampling cells in addition to normal and reduction cells to allow for encoder-decoder architectures
  - DARTS [Liu et al., ICLR 2019] for architecture optimization
  - BOHB [Falkner et al., ICML 2018] for hyperparameters



Remarks on Experimentation in NAS

- Final results are often *incomparable* due to
  - Different training pipelines without available source code
    - Releasing the final architecture does not help for comparisons
  - Different hyperparameter choices
    - Very different hyperparameters for training and final evaluation
  - Different search spaces / initial models
    - Starting from random or from state-of-the-art?

→ Need for looking beyond the error numbers on CIFAR

→ Need for benchmarks including training pipeline & hyperparams

- Experiments are often very expensive

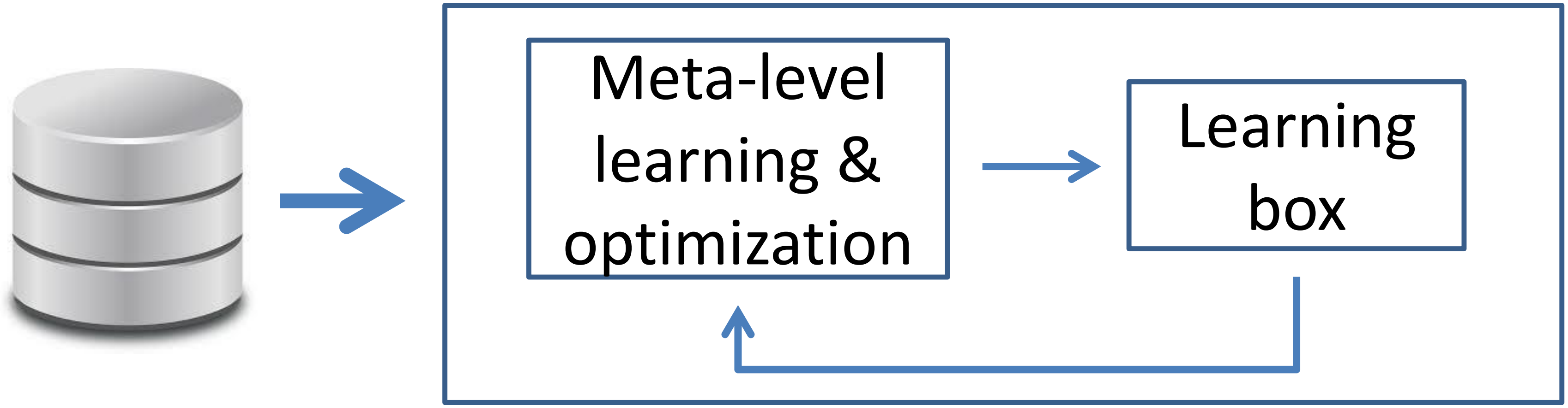
→ Need for cheap benchmarks that allow for many runs, e.g., Ying et al., ICML 2019

Outline

1. General AutoML
2. Neural Architecture Search
3. Meta Learning & Learning to Learn

For more details, see: automl.org/book

AutoML: true end-to-end learning



Meta Learning & Learning to Learn

	Deep Learning	
engineering features	→	learning features
	Meta Learning	
engineering to learn	→	learning to learn

inspired by Luke Metz

“Meta-learning, or learning to learn, is the science of systematically observing how different machine learning approaches perform on a wide range of learning tasks, and then learning from this experience, or meta-data, to learn new tasks much faster than otherwise possible.“

[Vanschoren, Chapter 2 of the AutoML book]

Outline

Meta Learning & Learning to Learn

- Meta Learning for Few Shot Learning
- Learning to Optimize

Outline

Meta Learning & Learning to Learn

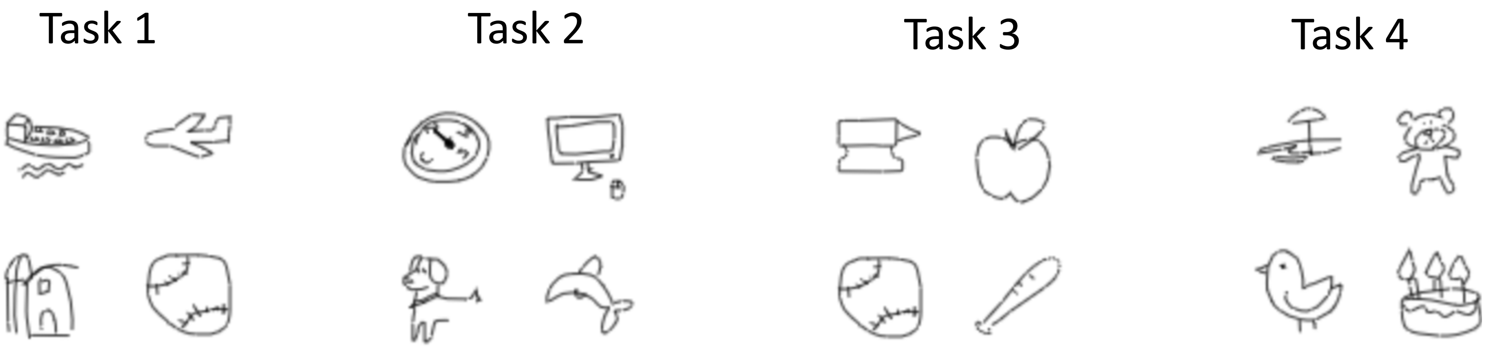
- Meta Learning for Few Shot Learning
- Learning to Optimize

Few-Shot Learning

- Classic deep learning setting: large, diverse data

source

- What if we don't have a large dataset?
  (medical data sets, personalized education, speech for rare languages,…)
- We can still learn from related tasks and experience.



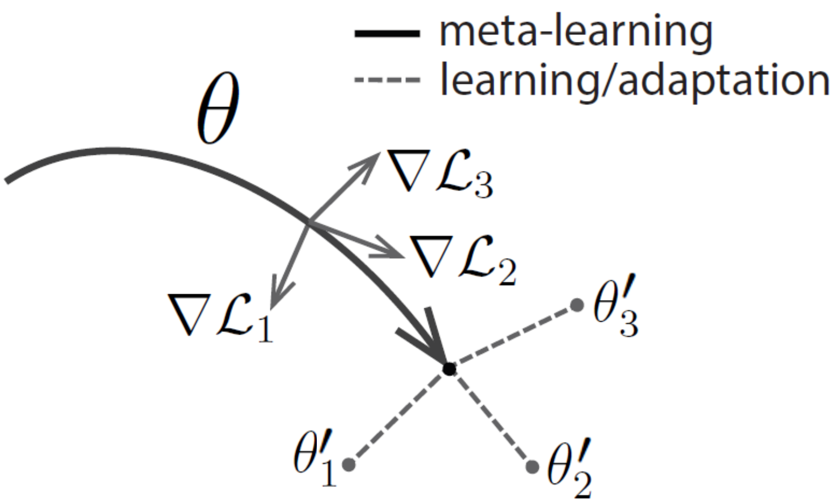
Quick, Draw! Dataset

- Few-Shot Learning setting: many small but related tasks

Slide inspired by Chelsea Finn

Learning Initializations for Few-Shot Learning

- Model-Agnostic Meta-Learning (MAML) [Finn et al., ICML 2017]
  - Learn initialization for weights θ of neural network that quickly adapts to weights θ_i' for new task T_i



Learning Initializations for Few-Shot Learning

- Model-Agnostic Meta-Learning (MAML) [Finn et al., ICML 2017]

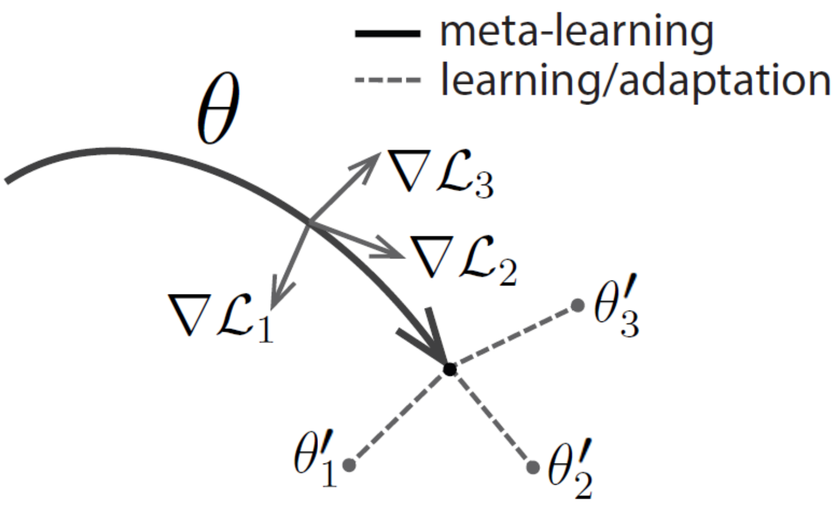
while not done:

1. sample tasks T_i
2. update task weights

   $$\theta_i' = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)$$

3. update meta weights θ by solving

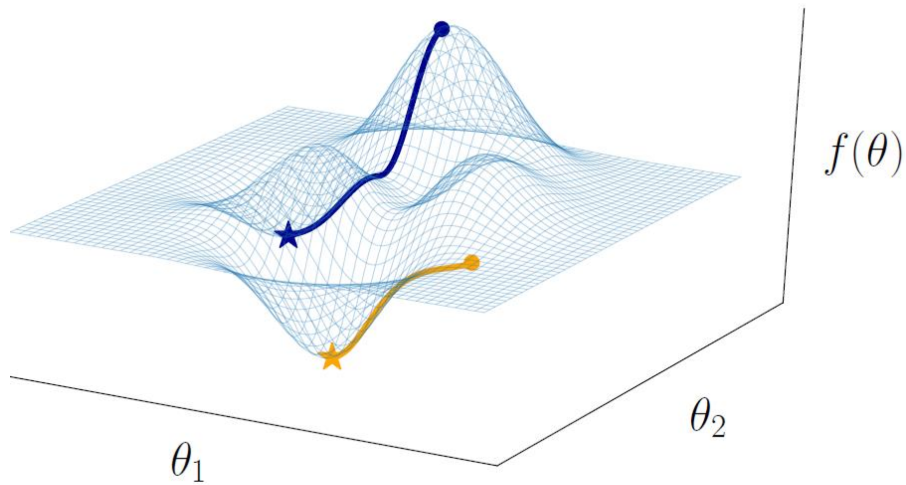
$$\min_\theta \sum_{\mathcal{T}_i} \mathcal{L}_{\mathcal{T}_i}(f_{\theta_i'}) = \sum_{\mathcal{T}_i} \mathcal{L}_{\mathcal{T}_i}(f_{\theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)})$$



See also
REPTILE [Nichol et al., arXiv preprint 2018], PLATIPUS [Finn et al., NeurIPS 2018]
Meta-Dataset: A Dataset of Datasets for Learning to Learn from Few Examples
[Triantafillou et al., NeurIPS MetaLearn 2018]

Learning Initializations for Few-Shot Learning

- Transferring Knowledge Across Learning (LEAP)
  [Flennerhag et al., ICLR 2019]
  - look at training process rather than final weights only
  - Iteratively learn initialization with
    a) shorter expected gradient path that
    b) also improves performance



gradient path length

meta objective:

$$\begin{aligned} \min_{\theta^0} \quad & F(\theta^0) = \mathbb{E}_{\tau \sim p(\tau)}\left[d(\theta^0; M_\tau)\right] \\ \text{s.t.} \quad & \theta_\tau^{i+1} = u_\tau(\theta_\tau^i), \quad \theta_\tau^0 = \theta^0, \\ & \theta^0 \in \Theta = \cap_\tau \left\{\theta^0 \mid f_\tau(\theta_\tau^{K_\tau}) \leq f_\tau(\psi_\tau^{K_\tau})\right\} \end{aligned}$$

improved performance

Outline

Meta Learning & Learning to Learn

- Meta Learning for Few Shot Learning
- Learning to Optimize

Learning to Optimize

- Many machine learning problems solved by applying some form of gradient descent

$$\theta_{t+1} = \theta_t - \alpha_t \nabla f(\theta_t)$$

- Much modern work in optimization based on designing algorithms for specific class of problems

- e.g., for deep learning: AdaGrad [Duchi et al., JMLR 2011], Adam [Kingma et al., ICLR 2015], RMSProp, ...

Can we learn better optimizers / update rules?

Learning to Optimize (with RNNs)

- Learning to Learn by Gradient Descent by Gradient Descent [Andrychowicz et al., NeurIPS 2016]
- Replace classic gradient-based updates such as

$$\theta_{t+1} = \theta_t - \alpha_t \nabla f(\theta_t)$$

- by learning model g_t providing updates

$$\theta_{t+1} = \theta_t + g_t(\nabla f(\theta_t), \phi)$$

RNN providing updates for function's parameters

function to be optimized

parameters of RNN

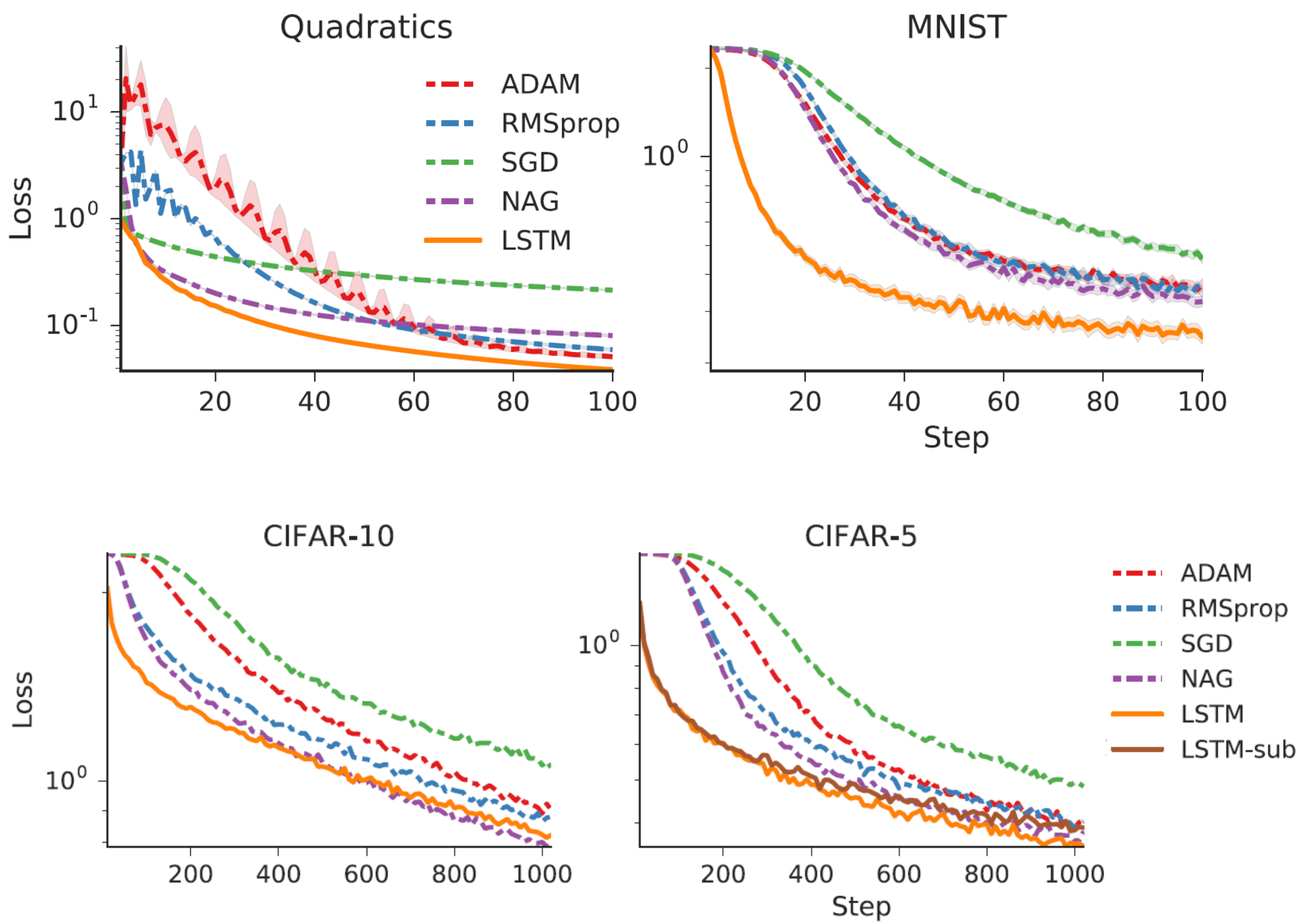
- g_t trained by optimizing meta objective

$$\mathcal{L}(\phi) = \mathbb{E}_f \left[\sum_{t=1}^{T} w_t f(\theta_t) \right]$$

Learning to Optimize (with RNNs)

[Andrychowicz et al., NeurIPS 2016]

Training MLP

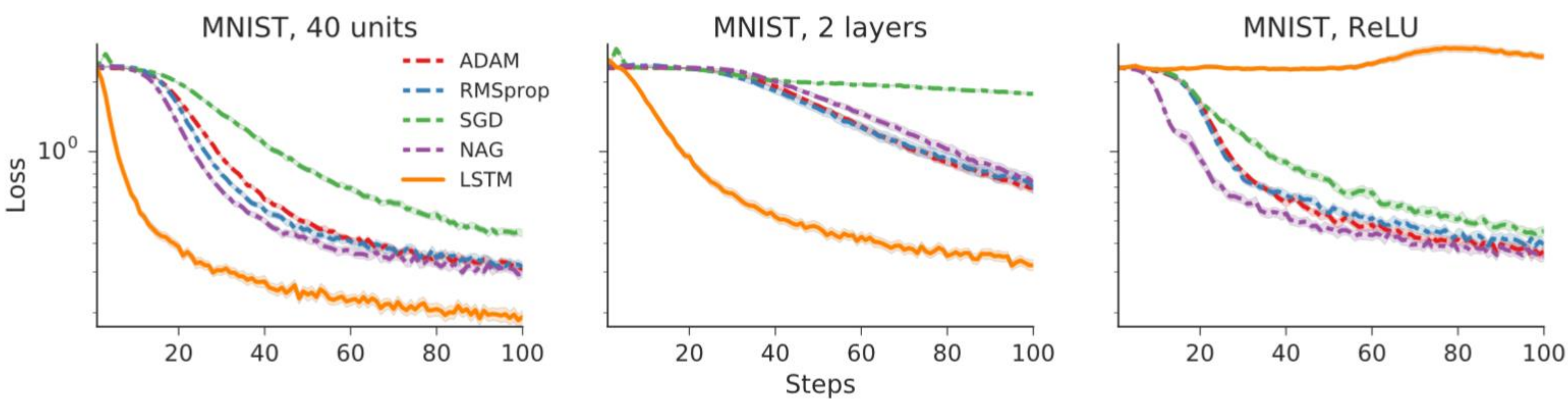


Training ConvNet

Learning to Optimize (with RNNs)

[Andrychowicz et al., NeurIPS 2016]

- Does the learned optimizer generalize to optimizing other neural networks architectures?
- Learned optimizer trained on optimizing MLP with 20 hidden units, 1 layer, sigmoid activation function



more hidden units

more layers

Different activation function

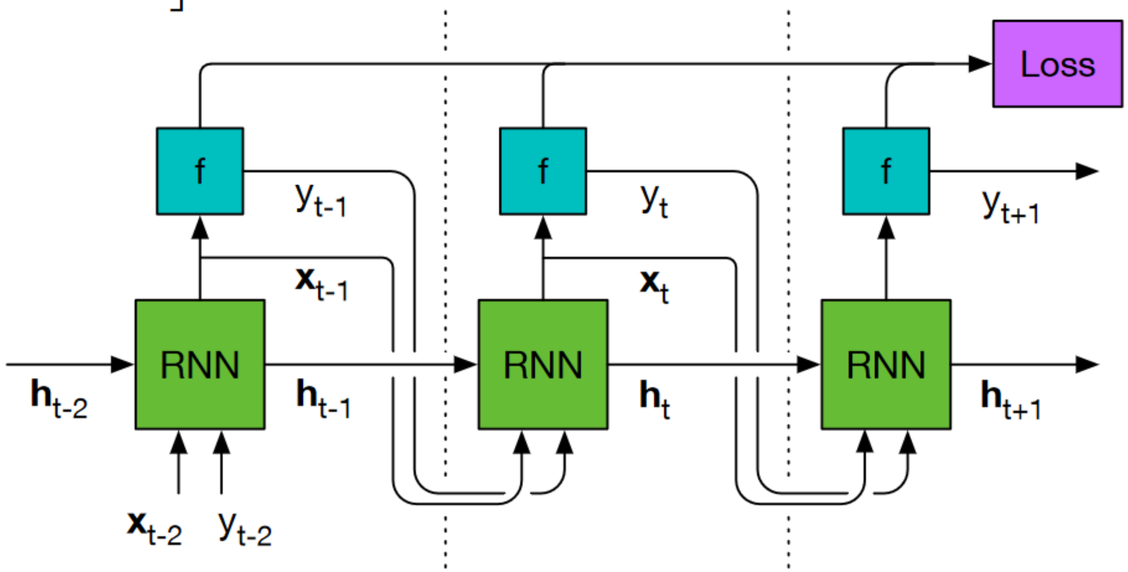
Learning to Optimize (with RNNs)

- Learning to Learn **without** Gradient Descent by Gradient Descent [Chen et al., ICML 2017]
  - Directly propose next query point x_t rather then update rule

$$\mathbf{h}_t, \mathbf{x}_t = \mathrm{RNN}_\theta(\mathbf{h}_{t-1}, \mathbf{x}_{t-1}, y_{t-1})$$

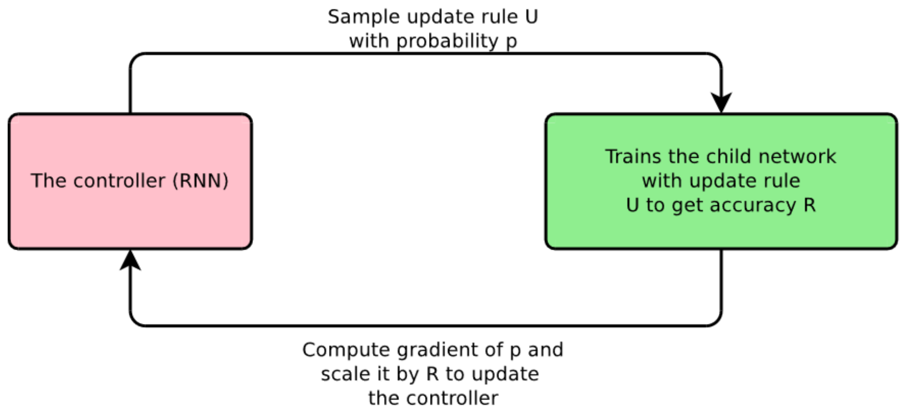
  - No need for gradients during meta-test time!
  - meta objective: e.g., maximize Expected Improvement

$$L_{\mathrm{EI}}(\theta) = -\mathbb{E}_{f, y_{1:T-1}}\left[\sum_{t=1}^{T} \mathrm{EI}(\mathbf{x}_t \mid y_{1:t-1})\right]$$



Learning to Optimize (with RL)

- Neural Optimizer Search with RL [Bello et al., ICML 2017]
- Use RL to train RNN controller to sample update rules for training neural networks



- Allow updates of the form

$$\Delta w = b(u_1(op_1), u_2(op_2))$$

binary function

- Binary functions which map (x, y) to $x + y$ (addition), $x - y$ (subtraction), $x * y$ (multiplication), $\frac{x}{y+\epsilon}$ (division) or x (keep left).

unary functions

- Unary functions which map input x to: x, $-x$, e^x, $\log|x|$, $clip(x, 10^{-5})$, $clip(x, 10^{-4})$, $clip(x, 10^{-3})$, $drop(x, 0.1)$, $drop(x, 0.3)$, $drop(x, 0.5)$ and $sign(x)$.

operands

- Operands: g, g^2, g^3, $\hat{m}$, $\hat{v}$, $\hat{\gamma}$, $sign(g)$, $sign(\hat{m})$, 1, small constant noise, $10^{-4}w$, $10^{-3}w$, $10^{-2}w$, $10^{-1}w$, ADAM and RMSProp.

+ update rules of above form

Learning to Optimize (with RL)

[Bello et al., ICML 2017]

Top 5 update rules

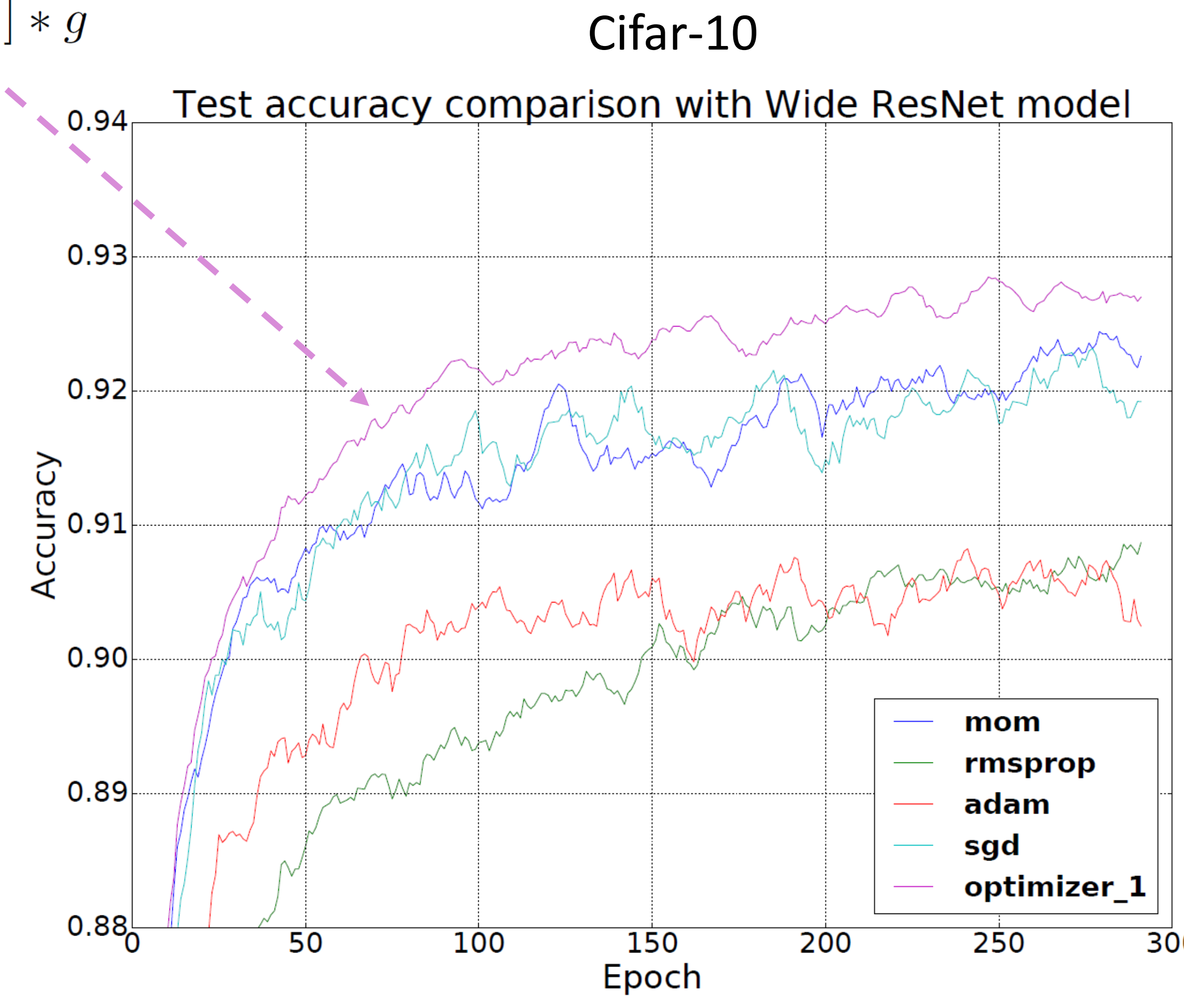
$[e^{\text{sign}(g)*\text{sign}(m)} + \text{clip}(g, 10^{-4})] * g$

$\text{clip}(\hat{m}, 10^{-4}) * e^{\hat{v}}$

$\hat{m} * e^{\hat{v}}$

$g * e^{\text{sign}(g)*\text{sign}(m)}$

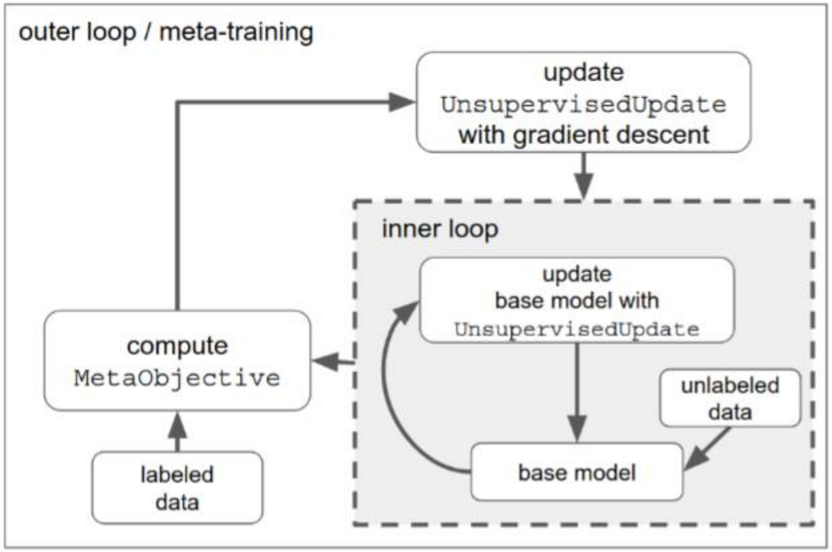
$\text{drop}(g, 0.3) * e^{\text{sign}(g)*\text{sign}(m)}$



- Method can also be used to search for activation functions [Ramachandran et al., ICLR WS 2018]
- See also [Li and Malik, ICLR 2017]

Learning to Optimize for Unsupervised Learning

- Meta-Learning Update Rules for Unsupervised Representation Learning [Metz et al., ICLR 2019]
- Unsupervised Learning: discover data representations without access to supervised labels



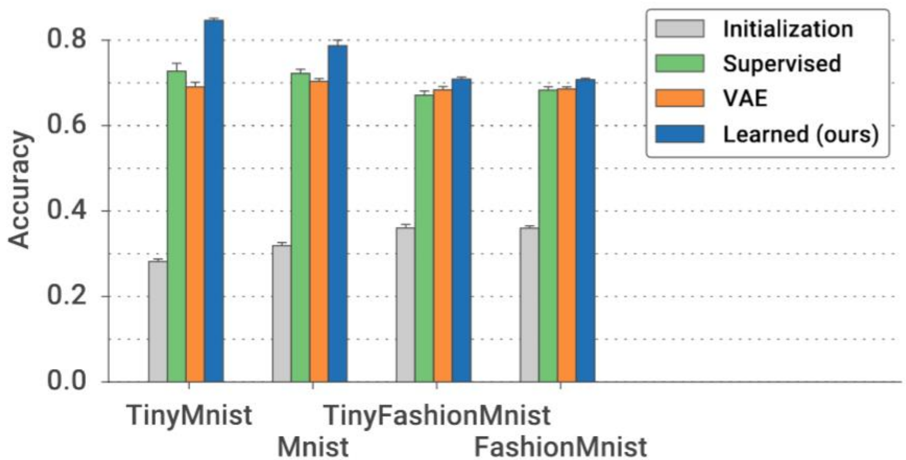
Learning to Optimize for Unsupervised Learning

Generalization of learned unsupervised learning rules

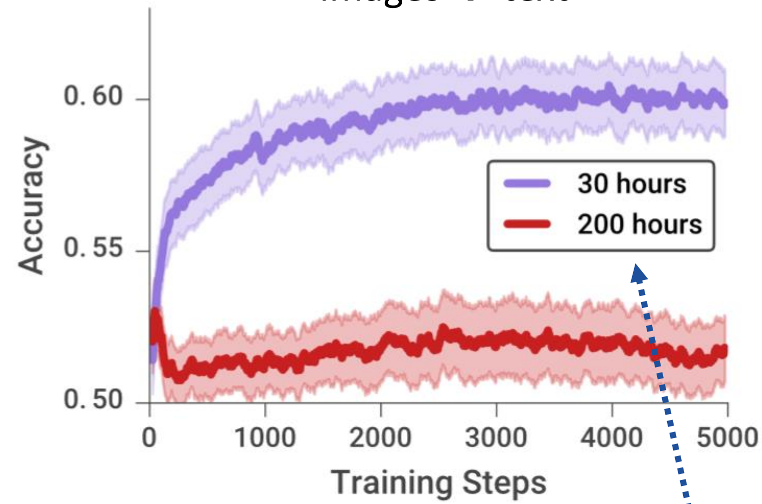
[Metz et al., ICLR 2019]

… to different data sets

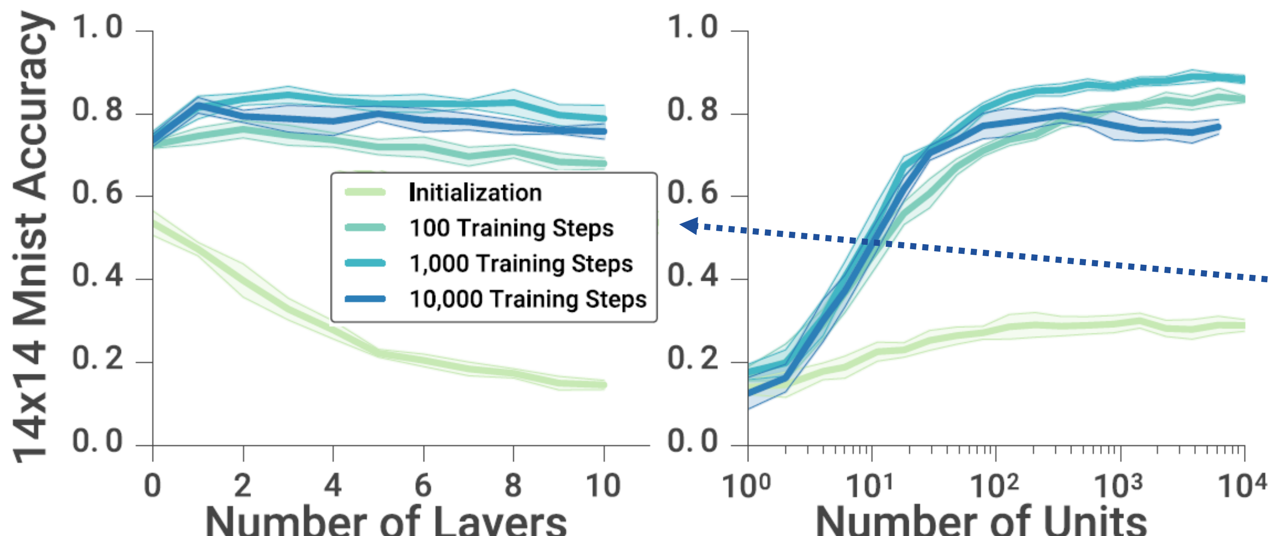
images → images



images → text



… to different architectures



meta train time

Learning to Simulate

- Learn the parameters ψ of a simulator via RL by solving

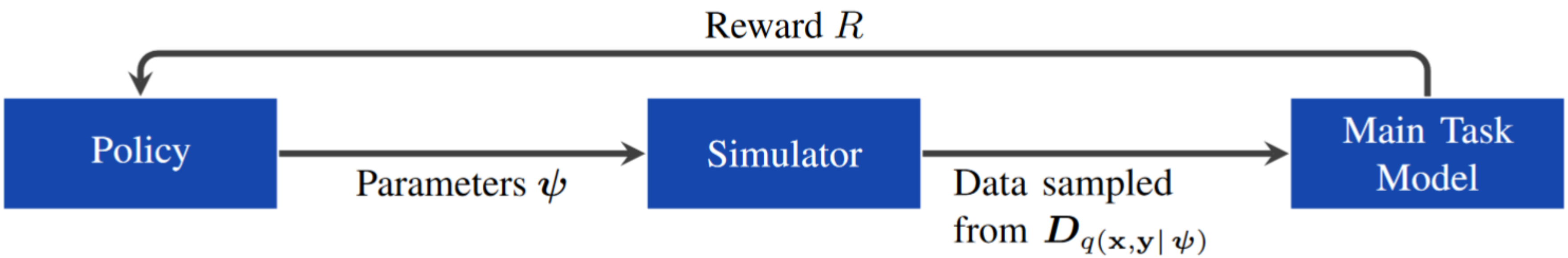
[Ruiz et al., ICLR 2019]

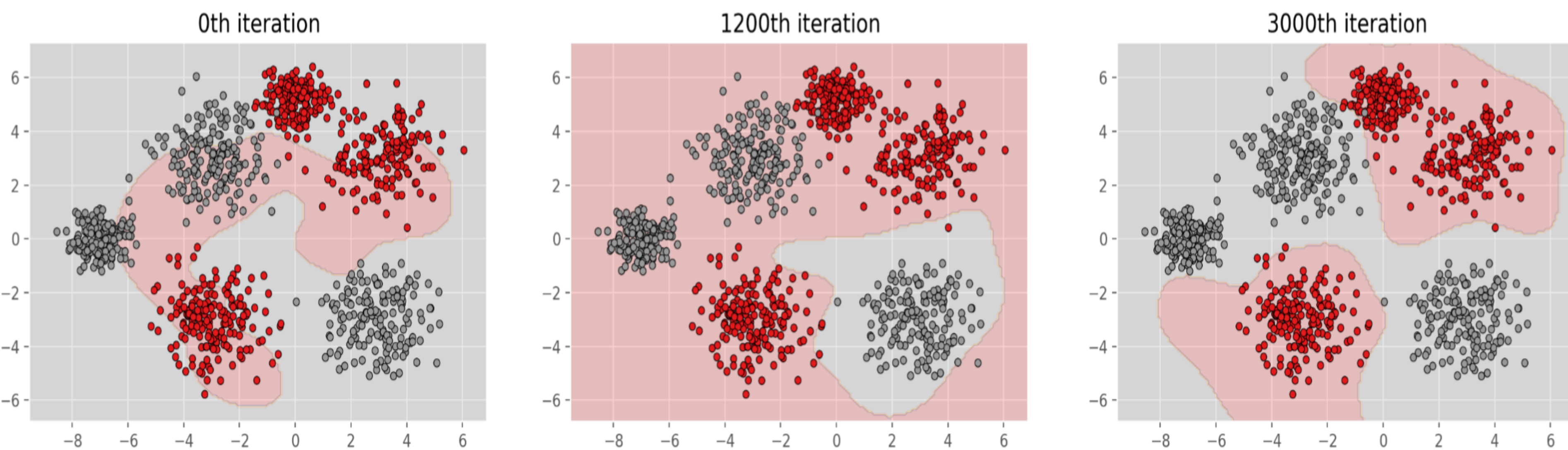
$$\psi^* = \arg\min_{\psi} \sum_{(\boldsymbol{x},\boldsymbol{y}) \in \boldsymbol{D}_{\text{val}}} \mathcal{L}\left(y, h_{\boldsymbol{\theta}}(\boldsymbol{x}; \boldsymbol{\theta}^*(\boldsymbol{\psi}))\right)$$

tune ψ on validation data given a model h_θ trained on simulated data

$$\text{s.t.} \quad \boldsymbol{\theta}^*(\boldsymbol{\psi}) = \arg\min_{\boldsymbol{\theta}} \sum_{(\boldsymbol{x},\boldsymbol{y}) \in \boldsymbol{D}_{q(\mathbf{x},\mathbf{y}|\,\boldsymbol{\psi})}} \mathcal{L}\left(\boldsymbol{y}, h_{\boldsymbol{\theta}}(\boldsymbol{x}, \boldsymbol{\theta})\right)$$

train model h_θ on data generated by simulator with parameter ψ





NAS & Meta Learning Wrap-up

- NAS is not insanely expensive anymore; several ways to speed up blackbox NAS
  - Weight inheritance
  - Weight sharing & one-shot models
  - Meta Learning (so far mostly unexplored)

  → by now: resources required for running NAS methods often in same order of magnitude as simply training a network

- Exciting research fields, lots of progress but also lots of open problems:
  - meaningful benchmarks missing
  - NAS beyond image classification
  - NAS for multi-task, multi-objective problems
  - Understanding why architectures found by NAS work well

- Meta Learning and Learning to Learn very natural concepts to look at
  - Humans almost never learn from scratch
  - Has the potential to significantly speed up learning of new tasks and improve performance when limited data is available

Thank you for your attention!

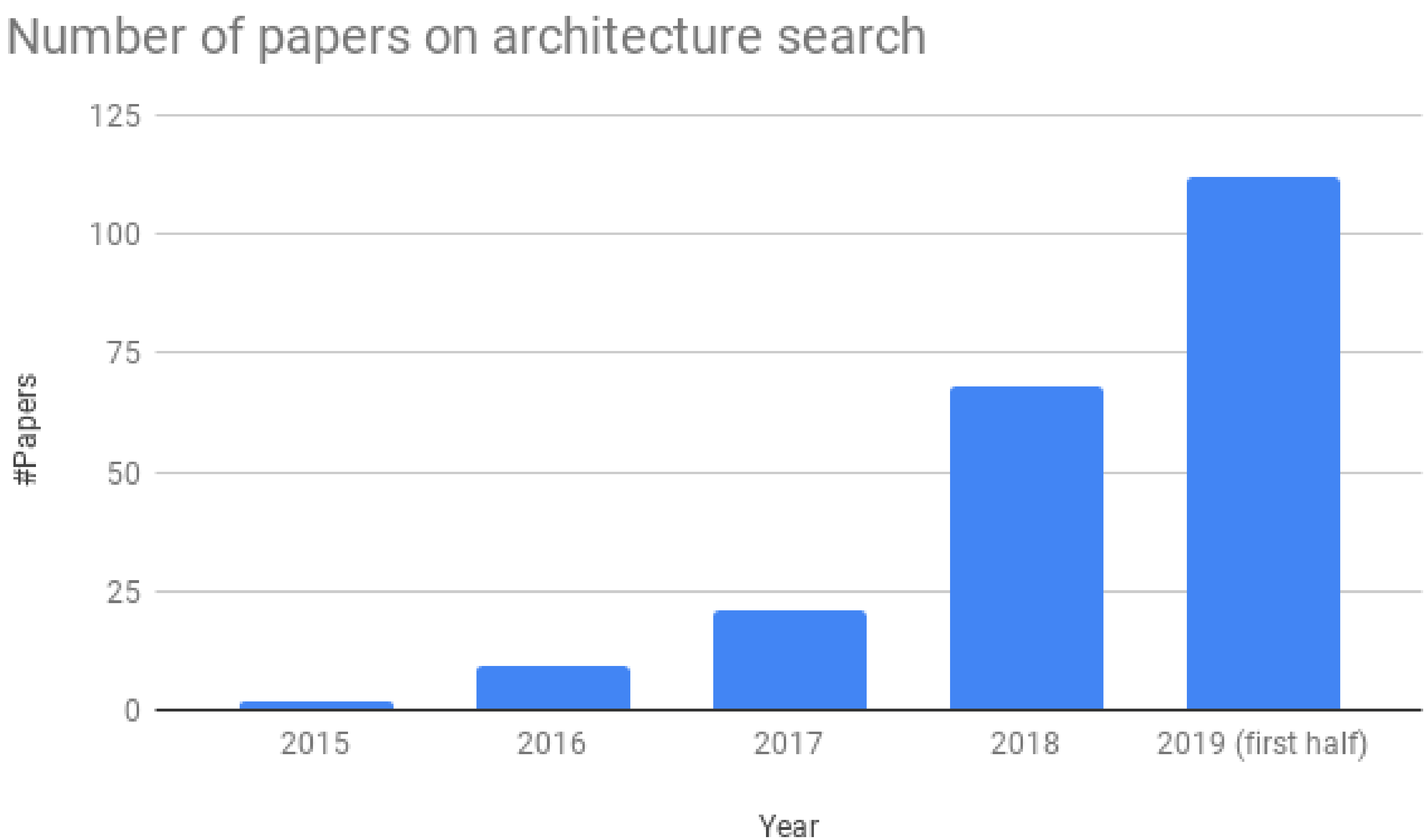
More details in AutoML book:
automl.org/book

Contact:

thomas.elsken@de.bosch.com

Backup Slides

New NAS papers over time



source

New NAS papers over time

	Reference	Error (%)	Params (Millions)	GPU Days
RL	Baker et al. (2017)	6.92	11.18	100
	Zoph and Le (2017)	3.65	37.4	22,400
	Cai et al. (2018a)	4.23	23.4	10
	Zoph et al. (2018)	3.41	3.3	2,000
	Zoph et al. (2018) + Cutout	2.65	3.3	2,000
	Zhong et al. (2018)	3.54	39.8	96
	Cai et al. (2018b)	2.99	5.7	200
	Cai et al. (2018b) + Cutout	2.49	5.7	200
EA	Real et al. (2017)	5.40	5.4	2,600
	Xie and Yuille (2017)	5.39	N/A	17
	Suganuma et al. (2017)	5.98	1.7	14.9
	Liu et al. (2018b)	3.75	15.7	300
	Real et al. (2019)	3.34	3.2	3,150
	Elsken et al. (2018)	5.2	19.7	1
	Wistuba (2018a) + Cutout	3.57	5.8	0.5
SMBO	Kandasamy et al. (2018)	8.69	N/A	1.7
	Liu et al. (2018a)	3.41	3.2	225
	Luo et al. (2018)	3.18	10.6	200
One-Shot	Pham et al. (2018)	3.54	4.6	0.5
	Pham et al. (2018) + Cutout	2.89	4.6	0.5
	Bender et al. (2018)	4.00	5.0	N/A
	Casale et al. (2019) + Cutout	2.81	3.7	1
	Liu et al. (2019b) + Cutout	2.76	3.3	4
	Xie et al. (2019b) + Cutout	2.85	2.8	1.5
	Cai et al. (2019) + Cutout	2.08	5.7	8.33
	Brock et al. (2018)	4.03	16.0	3
	Zhang et al. (2019)	2.84	5.7	0.84
Random	Liu et al. (2018b)	3.91	N/A	300
	Luo et al. (2018)	3.92	3.9	0.3
	Liu et al. (2019b) + Cutout	3.29	3.2	4
	Li and Talwalkar (2019) + Cutout	2.85	4.3	2.7
Human	Zagoruyko and Komodakis (2016)	3.87	36.2	-
	Gastaldi (2017) (26 2x32d)	3.55	2.9	-
	Gastaldi (2017) (26 2x96d)	2.86	26.2	-
	Gastaldi (2017) (26 2x112d)	2.82	35.6	-
	Yamada et al. (2016) + ShakeDrop	2.67	26.2	-

Designing Efficient Architectures

- NAS and compression:
  - Efficient Neural Architecture Compression (ESNAC) [Cao et al., ICLR 2019]
    - learn embedding space over architectures via bi-directional LSTM
    - Use Bayesian Optimization in embedded space to compress architectures
  - AutoML for Model Compression (AMC) [He et al., ECCV 2018]
    - Optimize per-layer compression rate via RL

Open-source Automated Deep Learning Systems

- Auto-Net [Mendoza et al, AutoML 2016]
  - First system AutoML 2016
  - Based on SMAC and Lasagne ($\rightarrow$ deprecated)
- Auto-Keras [Jin, Song & Hu, AutoML 2018]
  - Based on network morphisms and Keras; code
- Auto-PyTorch [unpublished]
  - Based on PyTorch and BOHB; code (pre-alpha)
- Neural Network Intelligence by Microsoft
  - Based on various techniques; code