The University of Texas at Austin
Electrical and Computer Engineering
Cockrell School of Engineering

Fall 2021

ADVANCED TOPICS IN COMPUTER VISION

Atlas Wang
Assistant Professor, The University of Texas at Austin

Visual Informatics Group@UT Austin
https://vita-group.github.io/

Progress in AI

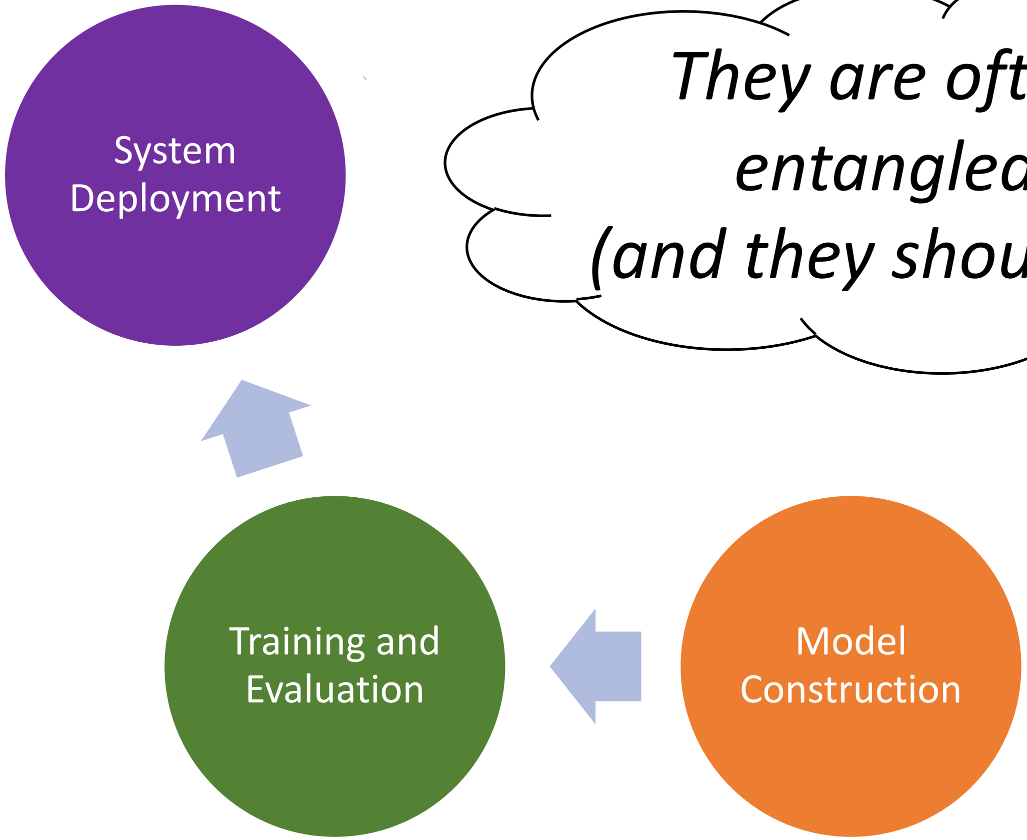
- Generation 1: Good Old Fashioned AI
  - Handcraft predictions
  - Learn nothing
- Generation 2: Shallow Learning
  - Handcraft features
  - Learn predictions
- Generation 3: Deep Learning
  - Handcraft algorithm (architectures, data processing, …)
  - Learn features and predictions end-to-end
- Generation 4: Learn2Learn (?)
  - Handcraft nothing
  - Learn algorithm, features and predictions end-to-end

The Rise of Automation for ML

Hyperparameter Tuning	*(algorithm optimization)*	*Auto-Weka, Auto-sklearn, H2O AutoML…*
Neural Architecture Search	*(model construction)*	*Auto-Keras, AutoGluon…*
Platform & Full Pipeline Automation		*Google Cloud AutoML, AWS Autopilot, H2O Driverless AI …*

Look at ML Lifecycle…

What Can be Automated?



They are often entangled (and they should be)

A Great Resource: https://www.ml4aad.org

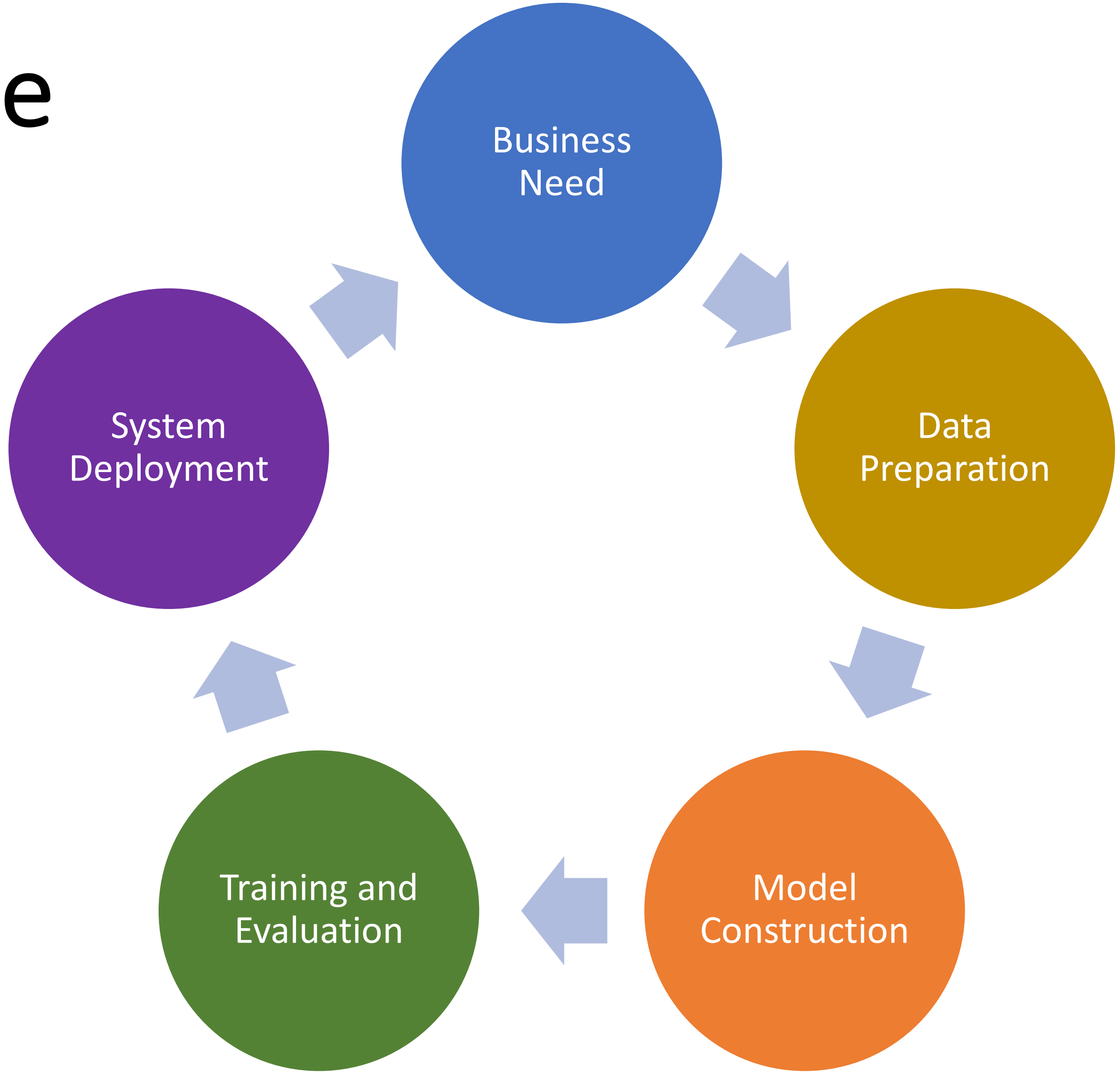
Traditional ML Lifecycle

- **Model:** What to Use
  - By cross-validation, trial-and-error …
  - Or experience (a.k.a. luck)
- **Algorithm:** How to Train
  - Analytical algorithms, or heuristics
  - Deep learning: SGD etc. for them all
- **Hardware:** How to Deploy
  - Manual design of hardware & system
  - … and separately from model/algorithm

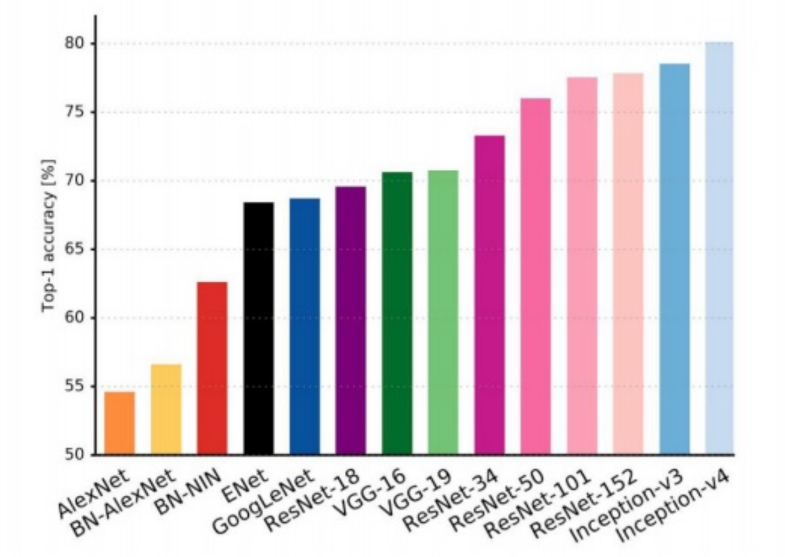
Automated ML Lifecycle

- **Model:** What to Use
  - Automatically discover the best model from the specific problem and data
- **Algorithm:** How to Train
  - Automatically customize an algorithm for the specific problem, data & model
- **Hardware:** How to Deploy
  - Automatically explore the co-design space for joint system-level optimization

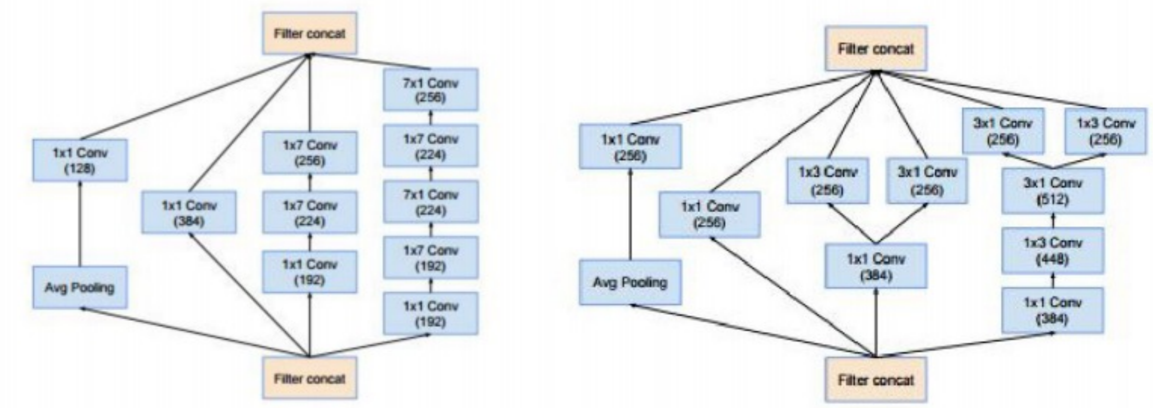
Automating ML Lifecycle



Architecture: the workhorse of Deep Learning



Canziani et al (2017)

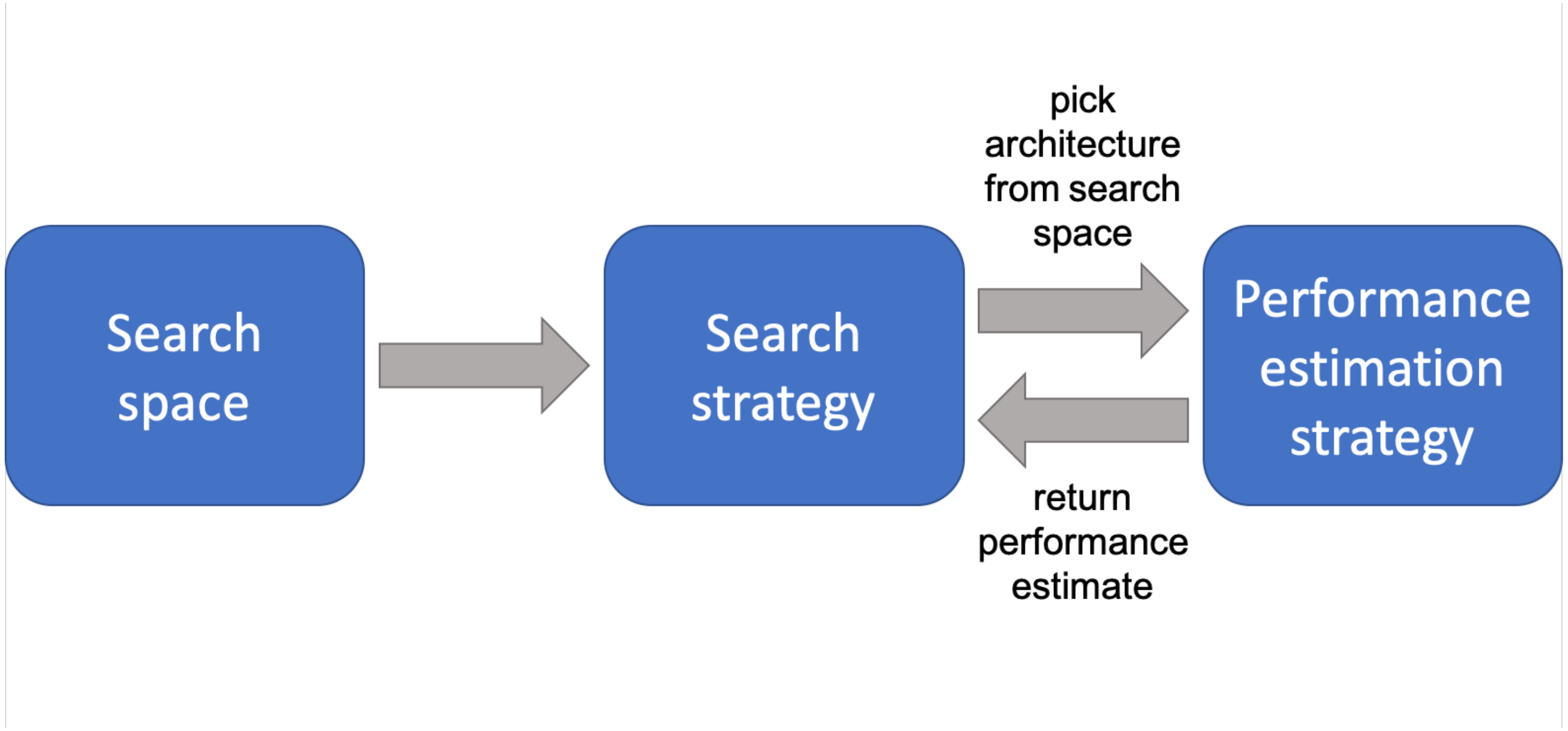


Complex hand-engineered layers from Inception-V4 (Szegedy et al., 2017)

Design Innovations (2012 - Present): Deeper networks, stacked modules, skip connections, squeeze-excitation block, ...

Can we try and learn good architectures automatically?

Neural Architecture Search (NAS)



- **View 1:** NAS as constrained optimization
- **View 2:** NAS as an MCMC process

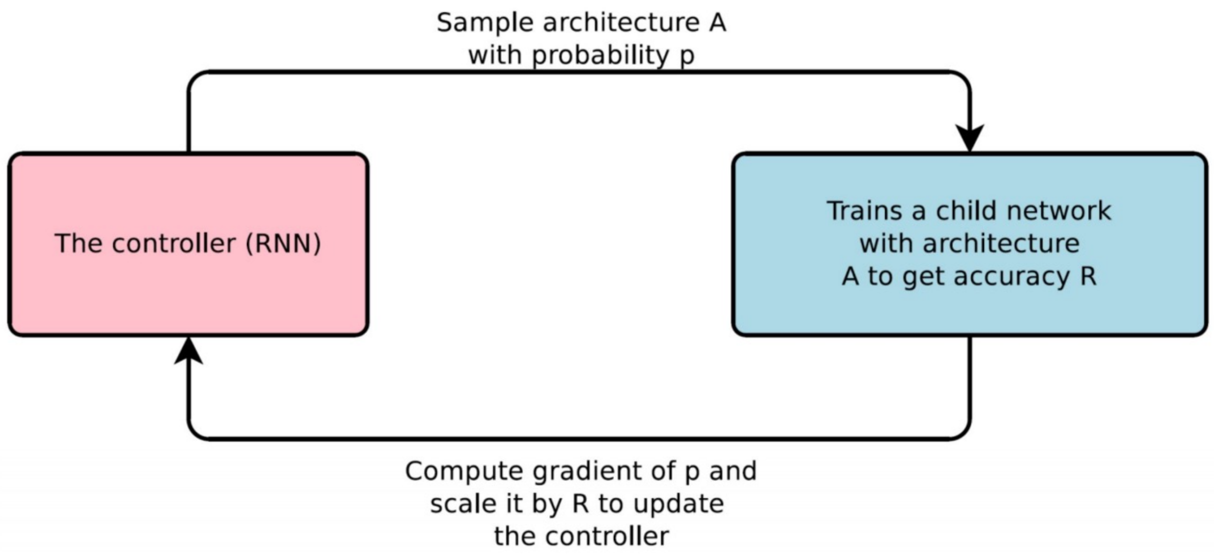
NEURAL ARCHITECTURE SEARCH WITH REINFORCEMENT LEARNING

Barret Zoph,* Quoc V. Le
Google Brain

DESIGNING NEURAL NETWORK ARCHITECTURES USING REINFORCEMENT LEARNING

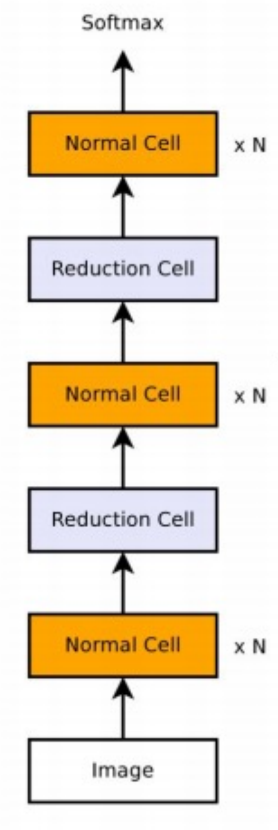
Bowen Baker, Otkrist Gupta, Nikhil Naik & Ramesh Raskar
Media Laboratory
Massachusetts Institute of Technology

Key Ideas: NAS via RL: Zoph and Le (2017)

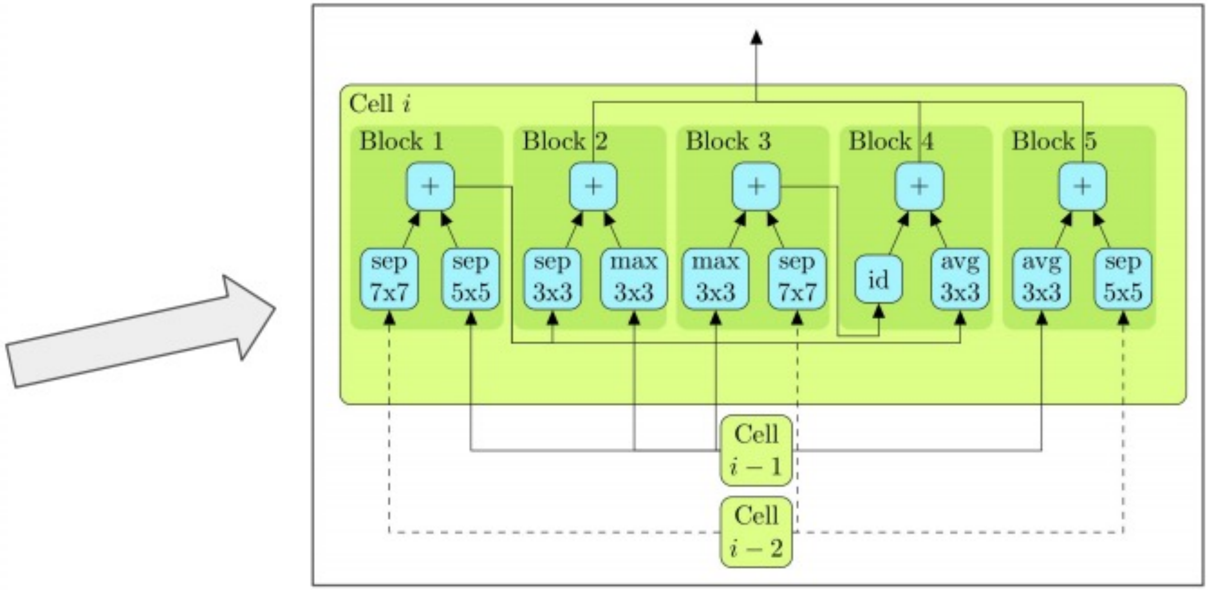


Represent	Specify the structure and connectivity of a NN by a configuration string (e.g., ["Filter Width: 5", "Filter Height: 3", "Num Filters: 24"])
Sample	Use a RNN ("Controller") to generate this string that specifies a neural network architecture
Evaluate	Train this architecture ("Child Network") to see its performance on a validation set
Update	Use RL to update the parameters of the Controller model based on the accuracy ("Reward") of the child model

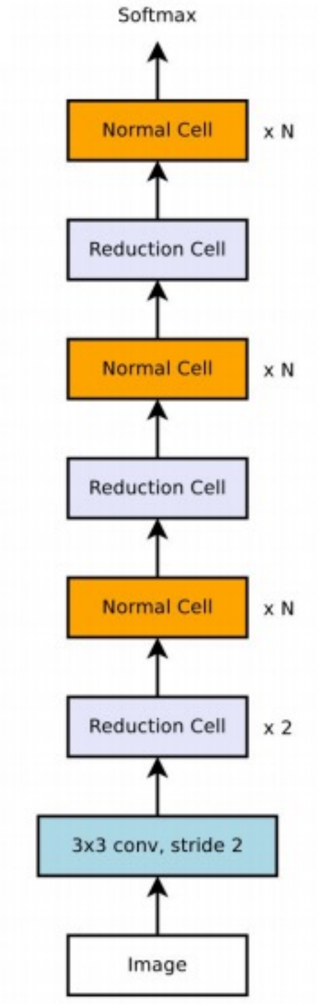
Cell-based Search Space



CIFAR10
Architecture



NASNet
Zoph et al. (2018)

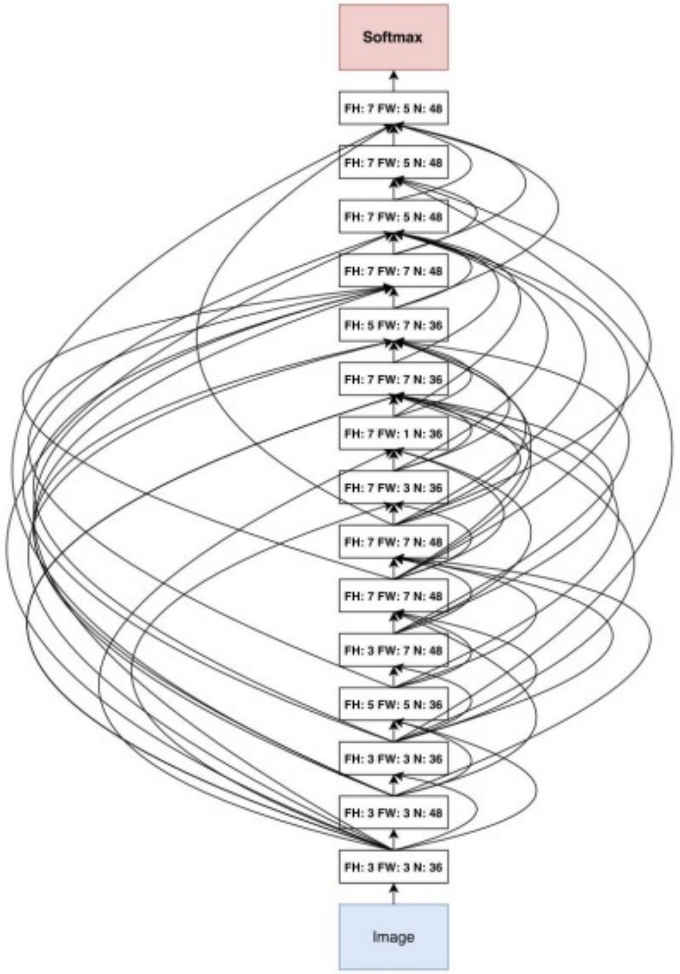


ImageNet
Architecture

Results on CIFAR-10

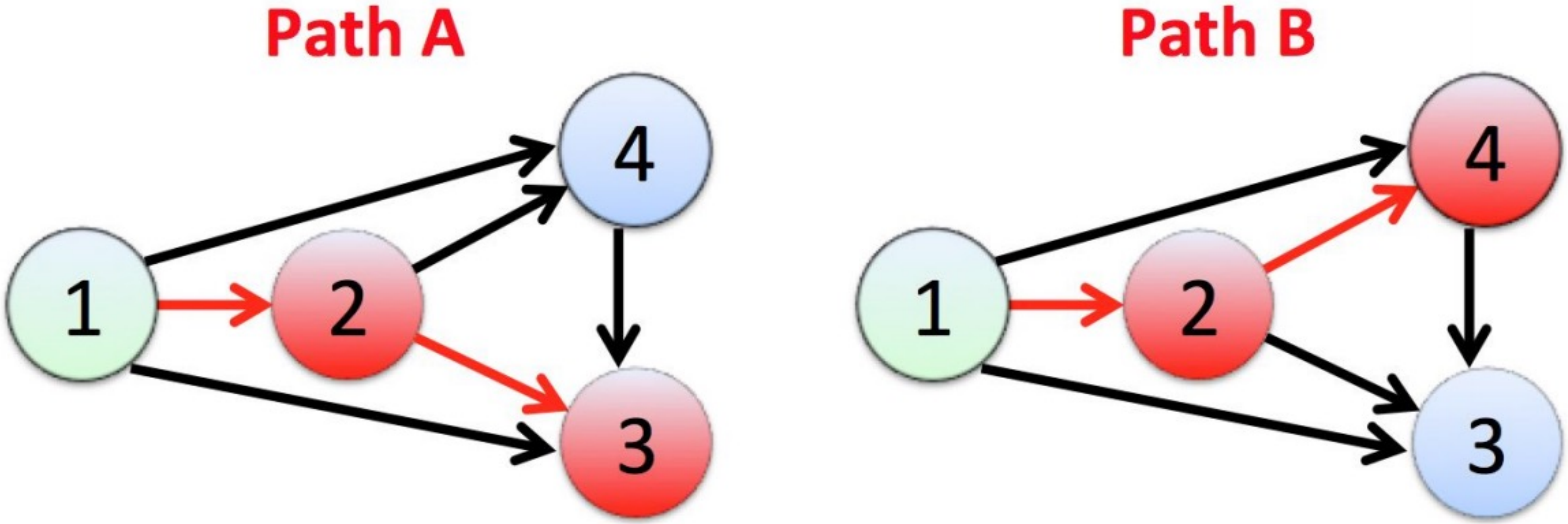
Model	Depth	Parameters	Error rate (%)
Wide ResNet (Zagoruyko & Komodakis, 2016)	16	11.0M	4.81
	28	36.5M	4.17
ResNet (pre-activation) (He et al., 2016b)	164	1.7M	5.46
	1001	10.2M	4.62
DenseNet ($L = 40, k = 12$) Huang et al. (2016a)	40	1.0M	5.24
DenseNet($L = 100, k = 12$) Huang et al. (2016a)	100	7.0M	4.10
DenseNet ($L = 100, k = 24$) Huang et al. (2016a)	100	27.2M	3.74
DenseNet-BC ($L = 100, k = 40$) Huang et al. (2016b)	190	25.6M	3.46
Neural Architecture Search v1 no stride or pooling	15	4.2M	5.50
Neural Architecture Search v2 predicting strides	20	2.5M	6.01
Neural Architecture Search v3 max pooling	39	7.1M	4.47
Neural Architecture Search v3 max pooling + more filters	39	37.4M	3.65
MetaQNN (Baker et al. 2017) no skip connections	12	11.8M	6.92

Comparable accuracy to best human-designed models ~2017



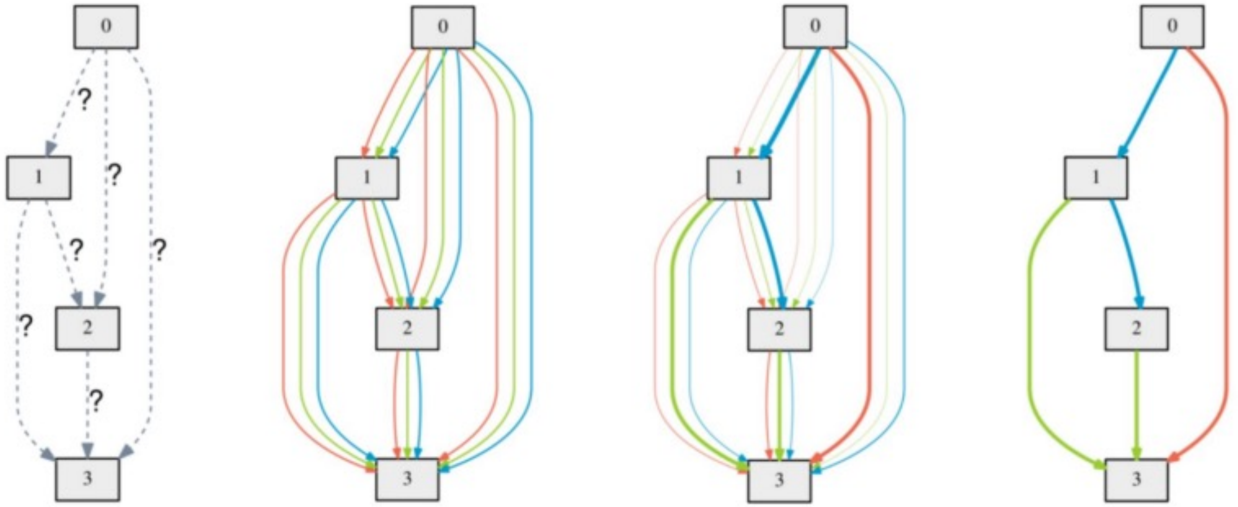
Best NAS Architecture on CIFAR-10
Zoph and Le (2017)

How To Make NAS More Efficient?



- Currently, models defined by **path A** and **path B** are trained independently
- Instead, treat all model trajectories as sub-graphs of a single directed acyclic graph
- Use a search strategy (e.g., RL, Evolution) to choose sub-graphs. Proposed in ENAS (Pham et al, 2018)

Gradient-based NAS with Weight Sharing



Find encoding weights α^* that

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

Learn design of normal and reduction cells

DARTS (Liu et al., 2018)

Create a mixed operation $\bar{o}^{(i,j)}$ parametrized by $\alpha^{(i,j)}$ for each edge (i, j)
while *not converged* **do**
1. Update architecture α by descending $\nabla_\alpha \mathcal{L}_{val}(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha)$ ($\xi = 0$ if using first-order approximation)
2. Update weights w by descending $\nabla_w \mathcal{L}_{train}(w, \alpha)$

Derive the final architecture based on the learned α.

Also see: SNAS (Xie et al. 2019)

Efficient NAS with Weight Sharing: Results on CIFAR-10

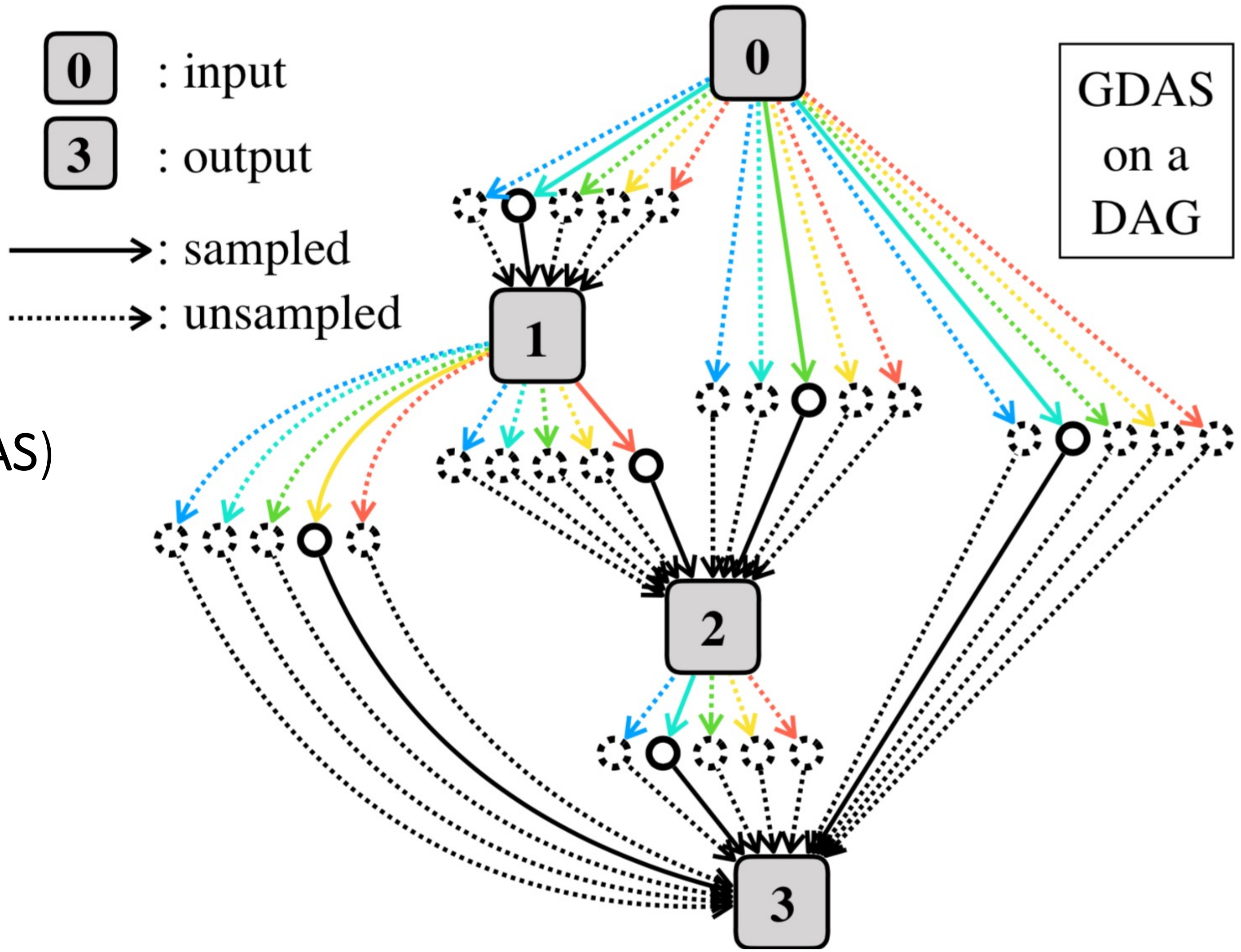
Reference	Error (%)	Params (Millions)	GPU Days
Pham et al. (2018)	3.54	4.6	0.5
Pham et al. (2018) + Cutout	2.89	4.6	0.5
Bender et al. (2018)	4.00	5.0	N/A
Casale et al. (2019) + Cutout	2.81	3.7	1
Liu et al. (2018c) + Cutout	2.76	3.3	4
Xie et al. (2019b) + Cutout	2.85	2.8	1.5
Cai et al. (2019) + Cutout	2.08	5.7	8.33
Brock et al. (2018)	4.03	16.0	3
Zhang et al. (2019)	4.30	5.1	0.4

Limitations:

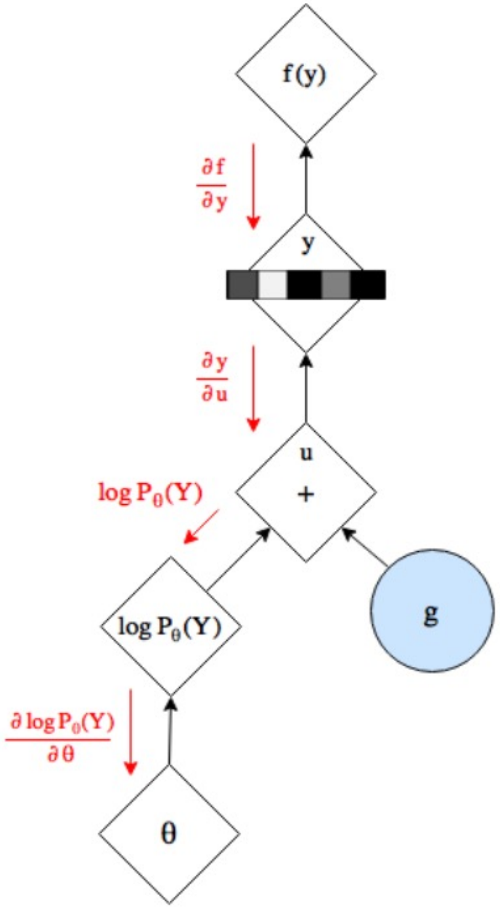
- Restrict the search space to the subgraphs of the supergraph
- Can bias the search towards certain regions of the search space
- In practice, the need to hold entire supergraph in GPU memory restricts search space size

Gradient-based NAS with Weight Sharing

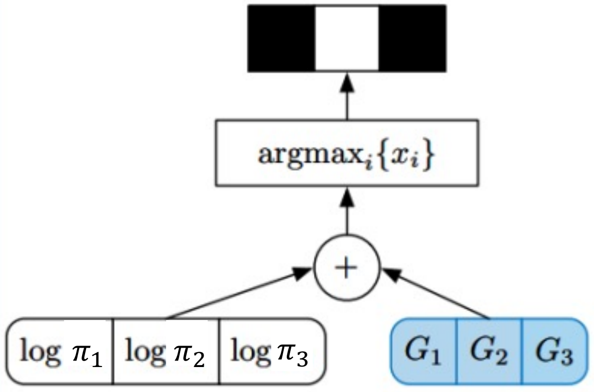
Gradient-based **search** using
Differentiable **Architecture** Sampler (**GDAS**)



Gradient-based NAS with Weight Sharing

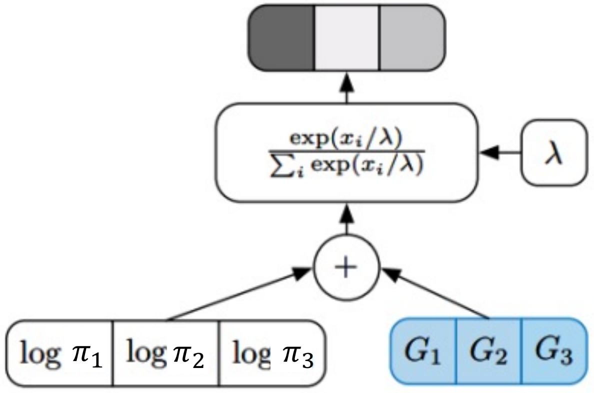


Gumbel-Max Trick



(a) Discrete(α)

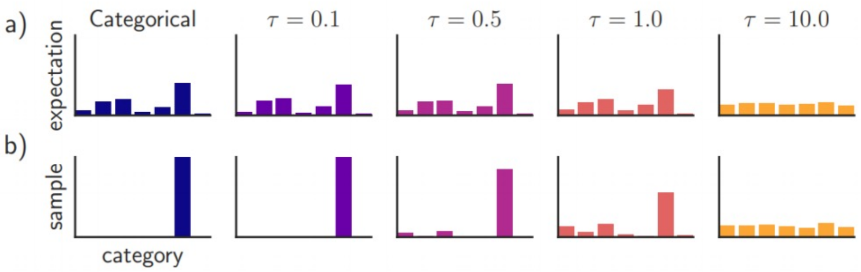
Gumbel-Softmax Trick



(b) Concrete(α, λ)

draw samples z from a categorical distribution with class probabilities π:

$$z = \mathtt{one\_hot}\left(\arg\max_i \left[g_i + \log \pi_i\right]\right)$$



To sample from a discrete categorical distribution we draw a sample of Gumbel noise, add it to $\boldsymbol{log(\pi_i)}$, and use $\boldsymbol{argmax}$ to find the value of $\boldsymbol{i}$ that produces the maximum.

Gradient-based NAS with Weight Sharing

```
def gumbel_max_sample(x):
    z = gumbel(loc=0, scale=1, size=x.shape)
    return (x + g).argmax(axis=1)
```

Inverse Transform Sampling

```
def sample_gumbel(shape, eps=1e-20):
  """Sample from Gumbel(0, 1)"""
  U = tf.random_uniform(shape,minval=0,maxval=1)
  return -tf.log(-tf.log(U + eps) + eps)
```

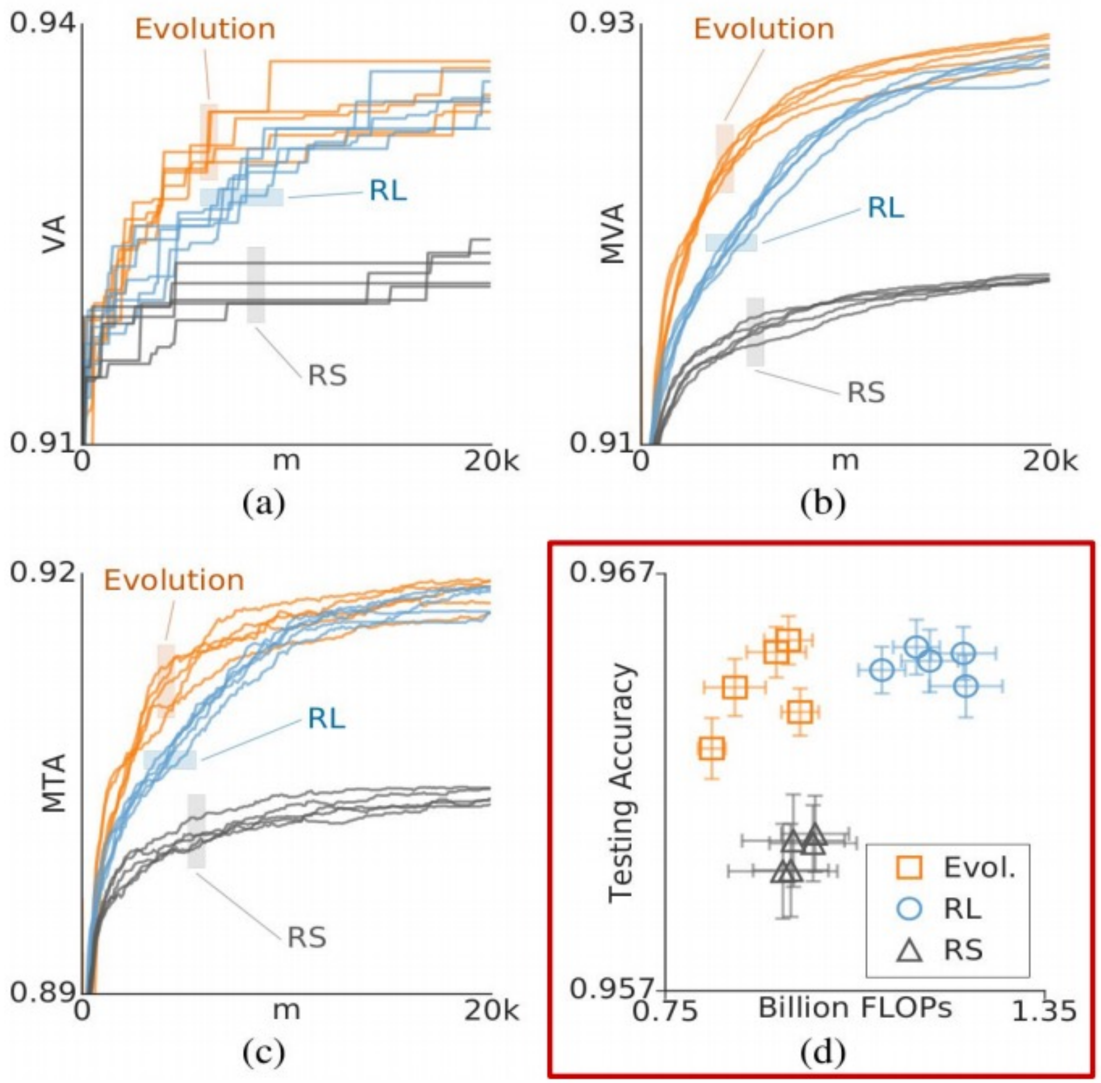
Smoothing relaxation

```
def gumbel_softmax_sample(logits, temperature):
  """ Draw a sample from the Gumbel-Softmax distribution"""
  y = logits + sample_gumbel(tf.shape(logits))
  return tf.nn.softmax( y / temperature)
```

- **Biased but low variance estimator**

  (Biased estimator w.r.t. original discrete objective but low variance & unbiased estimator w.r.t. continuous surrogate objective)
- **Plug & play (easy to code and implement)**
- **Computational efficiency**
- ~~**Better performance**~~

Are Intelligent Search Strategies Better Than Random Search?



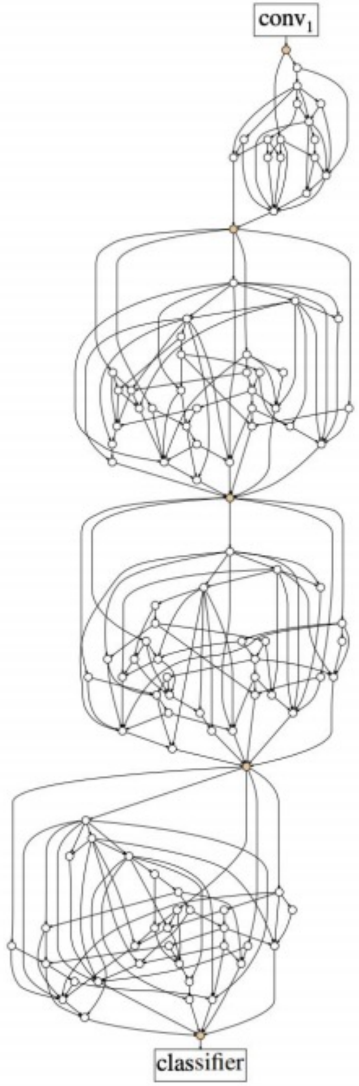
Real et al. (2018)
The difference in accuracy between best models found by random search, RL, and Evolution is **less than 1%** on CIFAR-10

Are Intelligent Search Strategies Better Than Random Search?

Architecture	Source	Test Error Best	Test Error Average	Params (M)
NASNet-A$^{\#*}$	[52]	N/A	2.65	3.3
AmoebaNet-B*	[43]	N/A	2.55 ± 0.05	2.8
ProxylessNAS†	[7]	2.08	N/A	5.7
GHN$^{\#\dagger}$	[50]	N/A	2.84 ± 0.07	5.7
SNAS†	[47]	N/A	2.85 ± 0.02	2.8
ENAS†	[41]	2.89	N/A	4.6
ENAS	[34]	2.91	N/A	4.2
Random search baseline	[34]	N/A	3.29 ± 0.15	3.2
DARTS (first order)	[34]	N/A	3.00 ± 0.14	3.3
DARTS (second order)	[34]	N/A	2.76 ± 0.09	3.3
DARTS (second order)‡	Ours	2.62	2.78 ± 0.12	3.3
ASHA baseline	Ours	2.85	3.03 ± 0.13	2.2
Random search WS‡	Ours	2.71	2.85 ± 0.08	4.3

Li and Talwalkar (2019)
Random search baseline finds a model with 2.71% error on CIFAR-10, comparable to best NAS methods based on RL, Evolution, Gradient Descent

Are Intelligent Search Strategies Better Than Random Search?

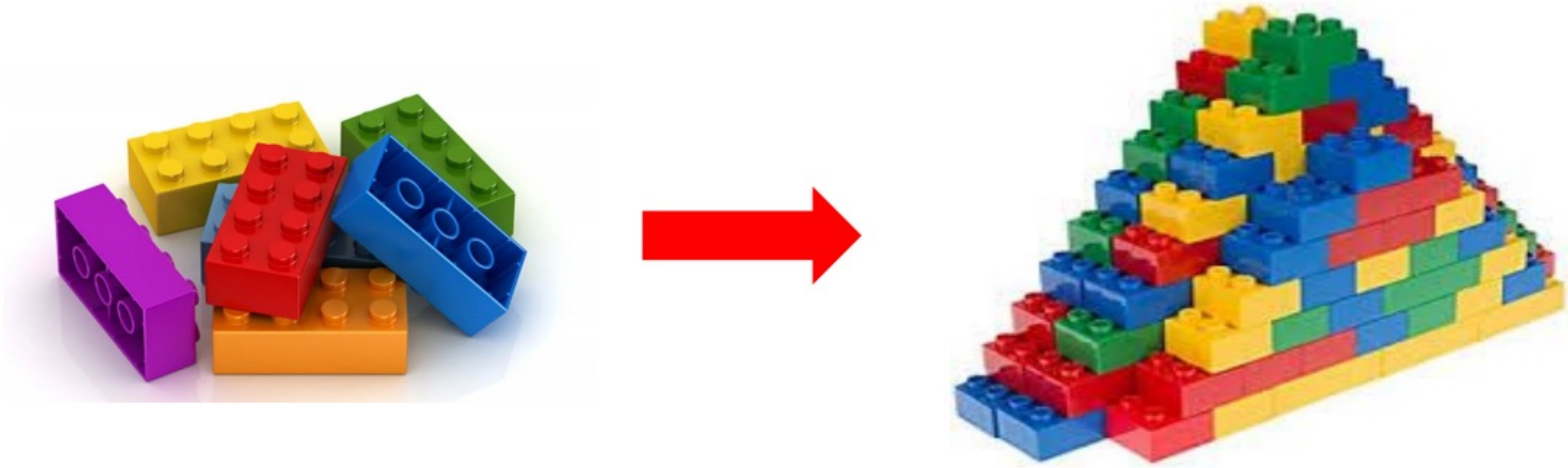


Xie et al. (2019)

A network consisting of multiple randomly wired "cells" is only **1.3% less accurate** than a similar capacity NAS models on **ImageNet**

network	test size	epochs	top-1 acc.	top-5 acc.	FLOPs (B)	params (M)
NASNet-A [56]	331^2	>250	82.7	96.2	23.8	88.9
Amoeba-B [34]	331^2	>250	82.3	96.1	22.3	84.0
Amoeba-A [34]	331^2	>250	82.8	96.1	23.1	86.7
PNASNet-5 [26]	331^2	>250	82.9	96.2	25.0	86.1
RandWire-WS	320^2	100	$81.6_{\pm 0.13}$	$95.6_{\pm 0.07}$	$16.0_{\pm 0.36}$	$61.5_{\pm 1.32}$

Are Intelligent Search Strategies Better Than Random Search?



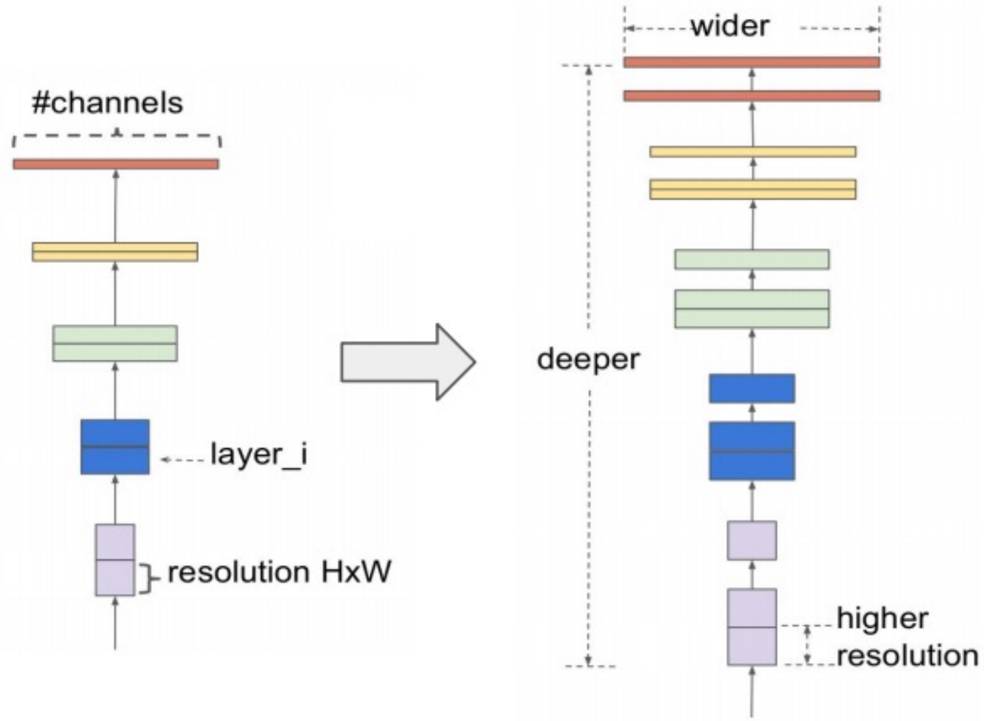
- Random search is a competitive baseline
- How do you design more flexible search spaces and sample-efficient search methods?
- Can intelligent search methods help discover new design motifs and basic building blocks?

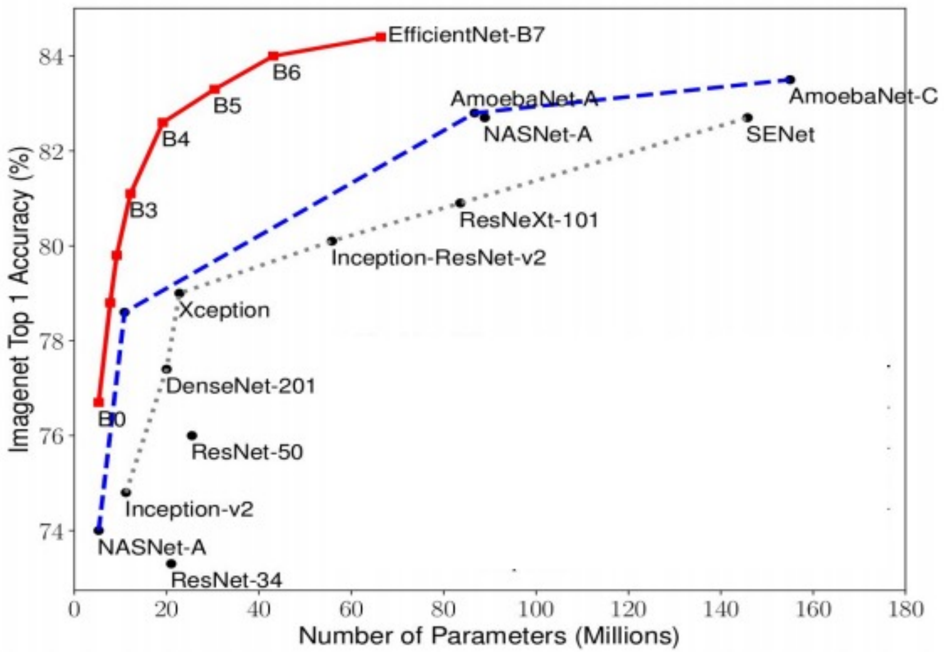
Designing Efficient Architectures

Introduce constraints like memory and inference time in addition to accuracy

NAS Method	References
Constrained Optimization	Tan et al. (2018), Cai et al. (2018), Hsu et al. (2018)
Multi-objective Optimization	Kim et al. (2017), Cai et al. (2019), Lu et al. (2018), Dong et al. (2018), Elsken et al. (2019), Tan and Le (2019)
Automated Pruning	He et al. (2019)

Designing Efficient Architectures from Scratch





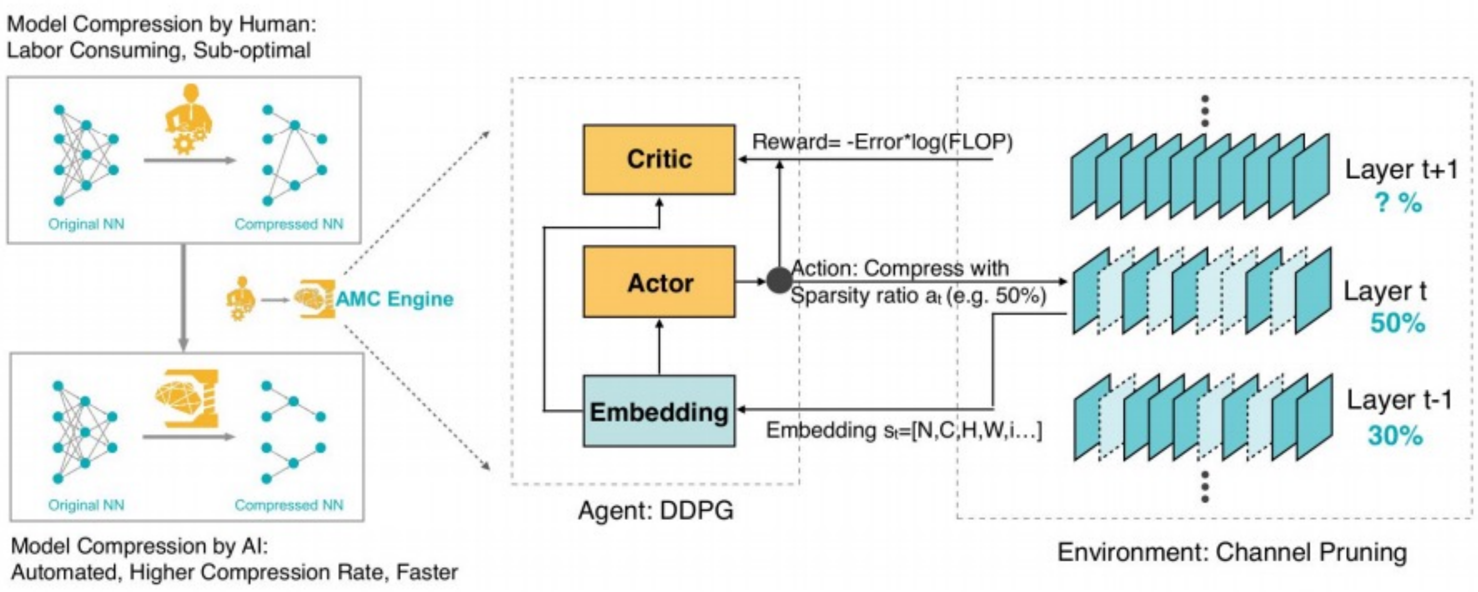
EfficientNet (Tan and Le, 2019)

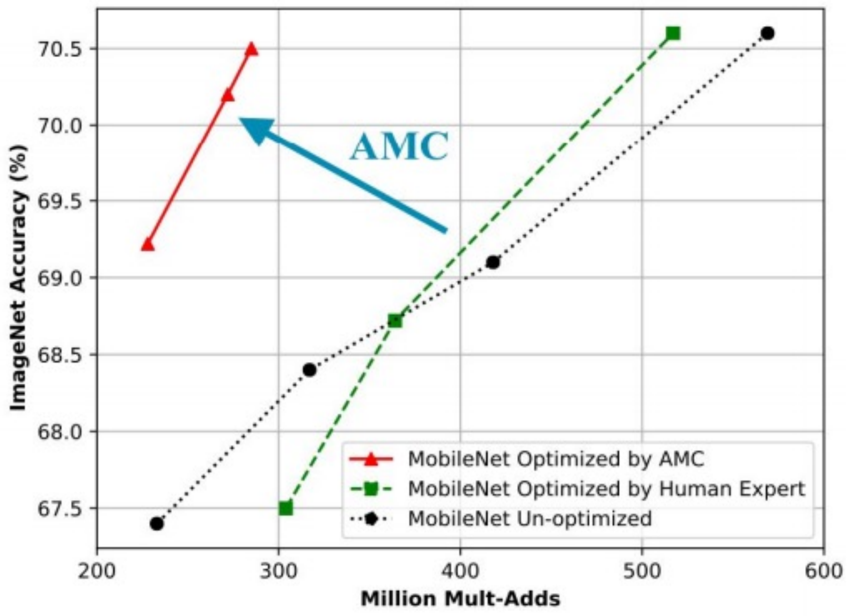
RL-based architecture search for a feed-forward block + scaling up using grid search

80% Top-1 Accuracy on ImageNet with 5x fewer parameters and 13x fewer FLOPS

than best human-designed model

Designing Efficient Architectures: Auto-Pruning

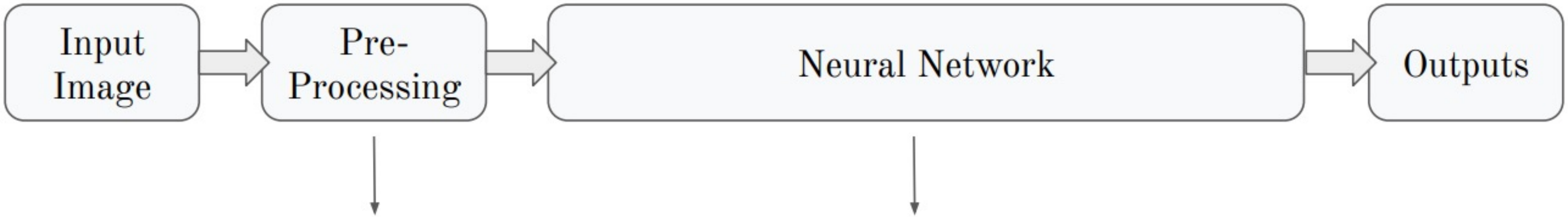




AMC: AutoML for Model Compression (He et al., 2019)

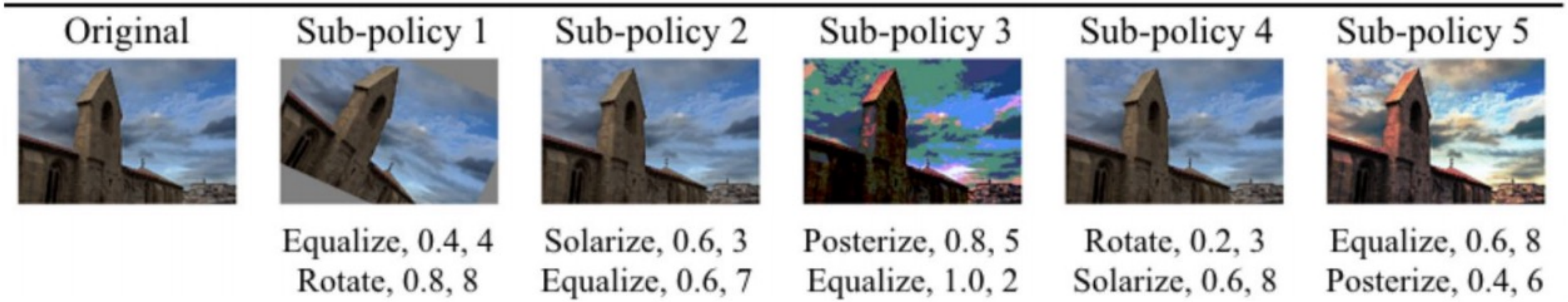
Using Deep Deterministic Policy Gradients to learn pruning ratio for each layer

Automating the Deep Learning Stack



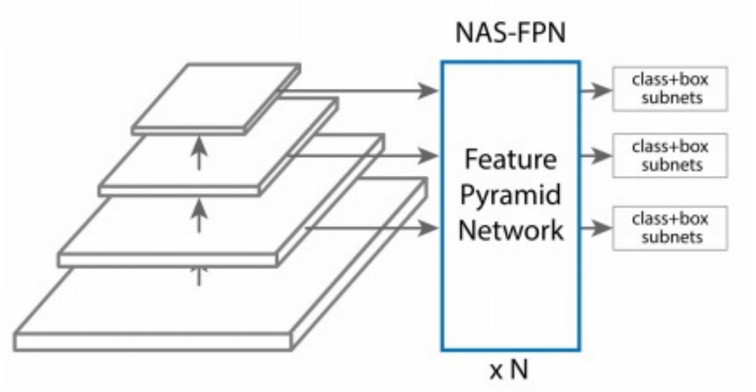
Data Augmentation (AutoAugment, Cubuk et al., 2018)

Activation Function (Ramachandran et al., 2018)
Optimizer (Bello et al., 2017)

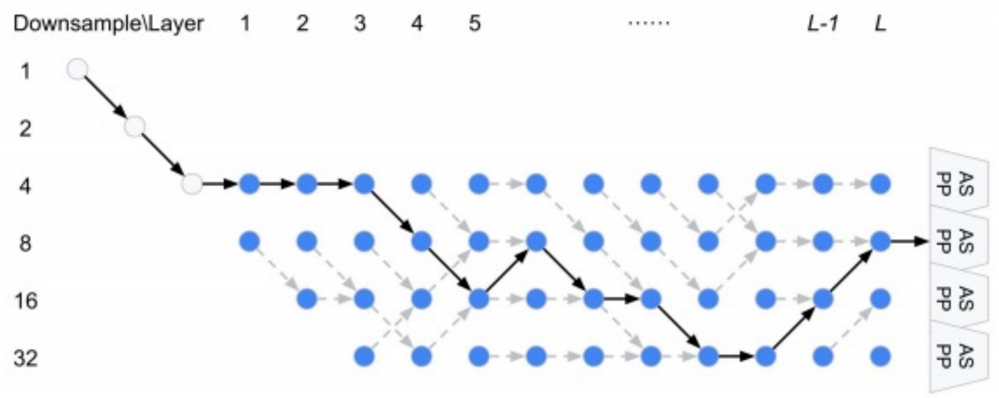


Model	Baseline	Inception Pre-processing [14]	AutoAugment
ResNet-50 [15]	24.70 / 7.80	23.69 / 6.92	22.37 / 6.18
ResNet-200 [15]	-	21.52 / 5.85	20.00 / 4.99

Neural Architecture Search: Beyond Image Classification



Object Detection
E.g. NAS-FPN (Ghaisi et al., 2019)



Semantic Segmentation
E.g. Auto-DeepLab (Liu et al., 2019)

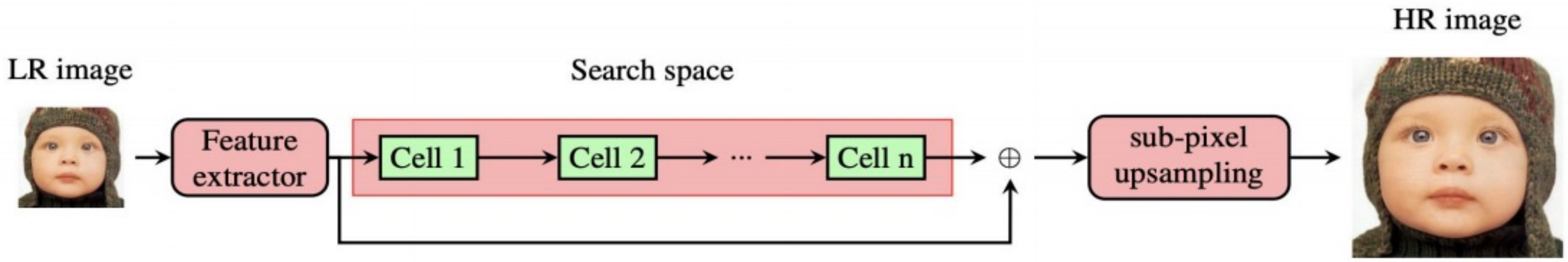


Image Super-Resolution e.g. MoreMNAS(Chu et al., 2019)

Neural Architecture Search (NAS): Summary

Status Quo:

- *Nascent area*: most work on image domain

What Remains to be Explored?

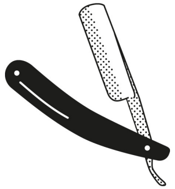
- *More tasks:* "real" complicated applications
- *More model types:* e.g., generative models
- *More data modalities:* beyond images
- Moving towards less constrained search spaces
- Theory Understanding, Faster Search, Tiny ML…

Key Challenges:

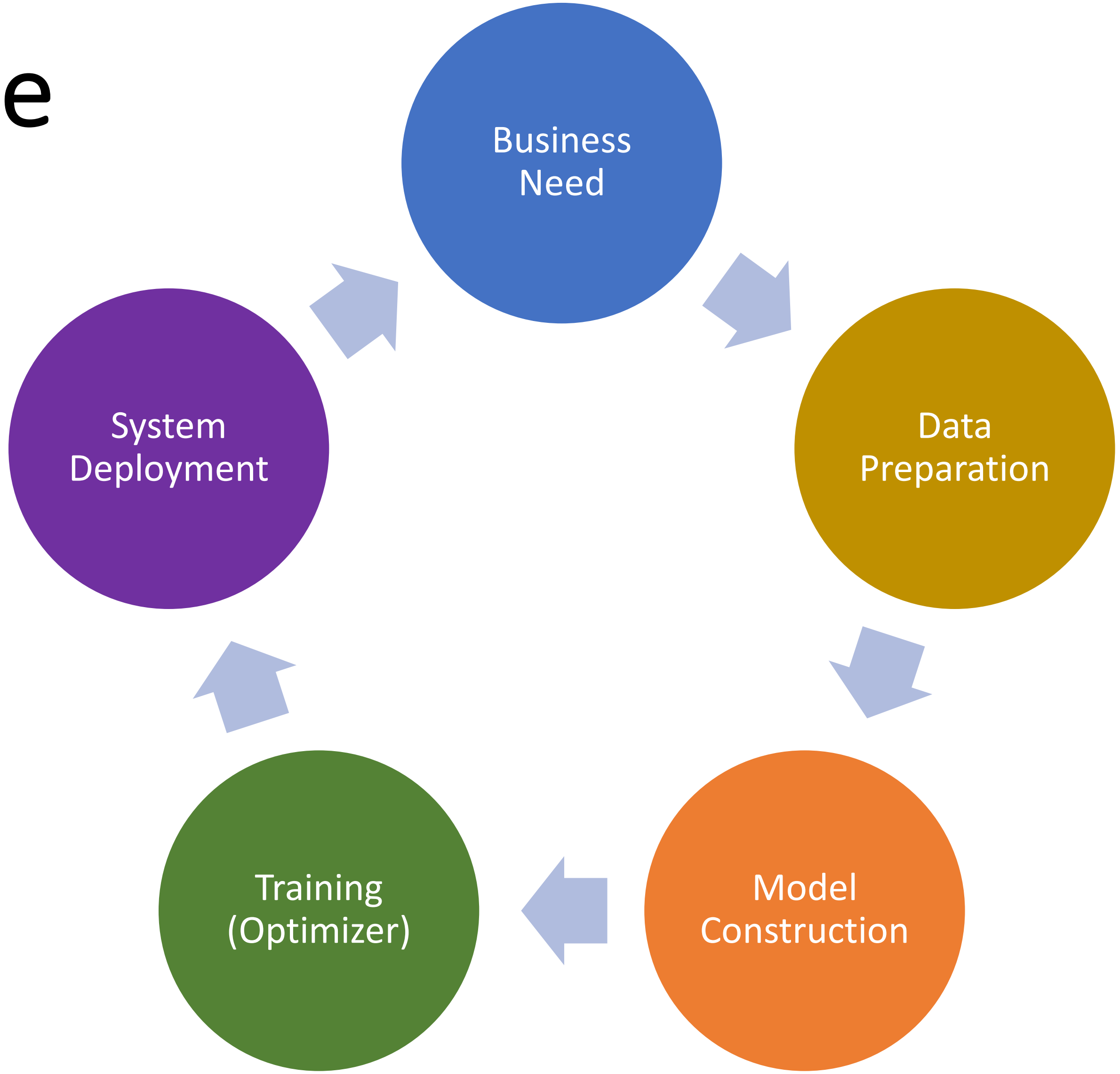
- Complicated applications → complex model → explosively larger search space
- Complex models → hard and unstable to train → affecting model selection

Mainstream Idea (for now): "Manual" Design as Priors

- Previous handcrafted models have identified consistent and successful design patterns
- Simplify search space and algorithm smartly, with those "manual design priors"
- But remember, it's not really "auto" yet …



Automating ML Lifecycle



Early Approaches to Meta-Learning

- Jürgen Schmidhuber
  - Genetic Programming. PhD thesis. 1987
  - Learning to control fast-weight memories: An alternative to dynamic recurrent networks. Neural Computation 1992
  - A neural network that embeds its own meta-levels. ICNN 1993.

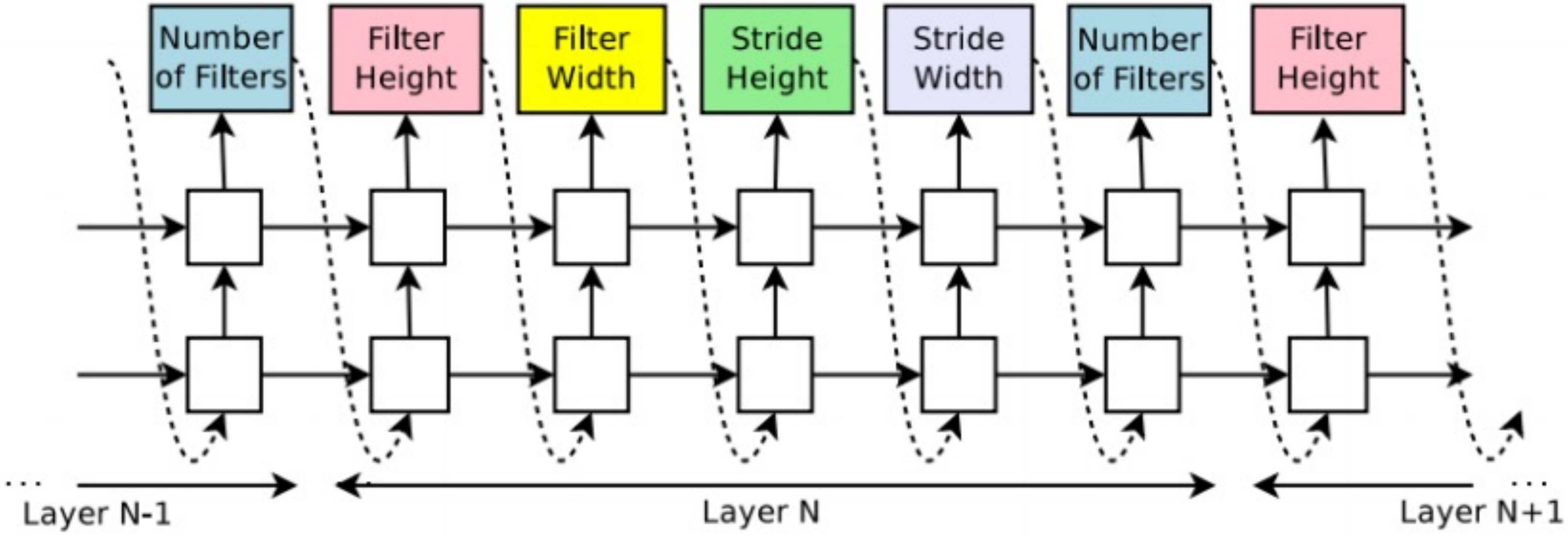
- Yoshua Bengio
  - Learning a synaptic learning rule. Univ. Montreal. 1990.
  - On the search for new learning rules for ANN. Neural Processing Letters 1992
    - Learning update rules with SGD

General Paradigm of Meta-Learning

Example meta-learning set-up for few-shot image classification

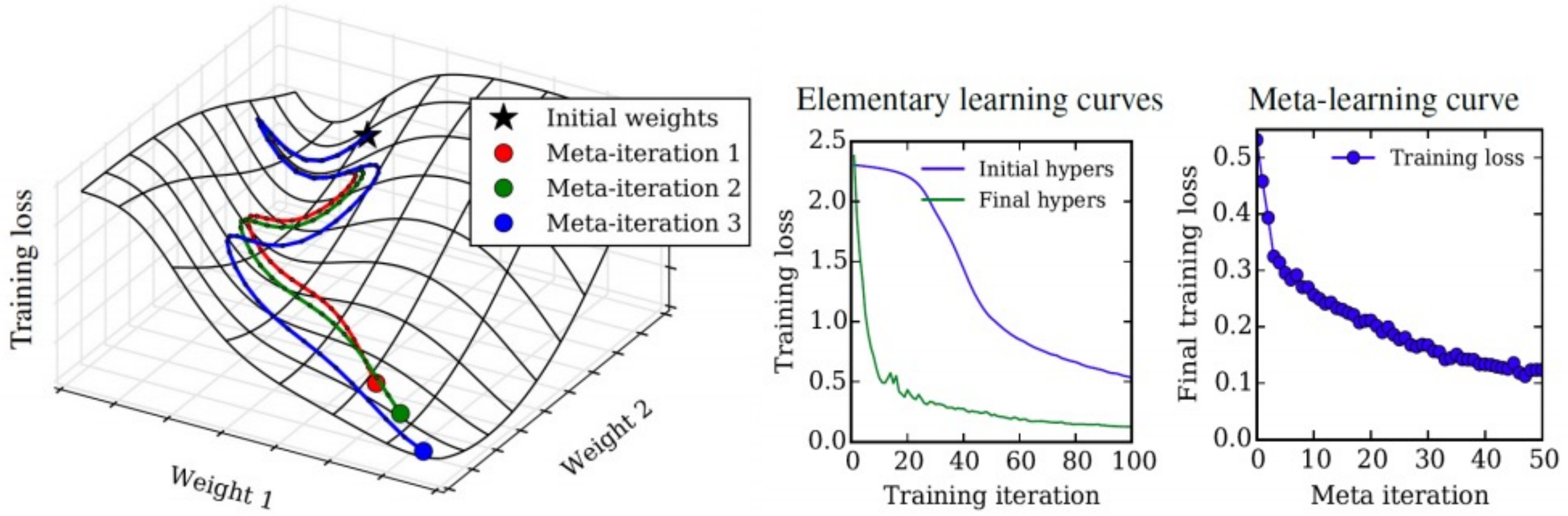
- During meta-learning, the model is trained to learn tasks in the meta-training set. Two optimizations:
  - The learner, which learns new tasks
  - The meta-learner, which trains the learner

Various Meta-Learning Tasks



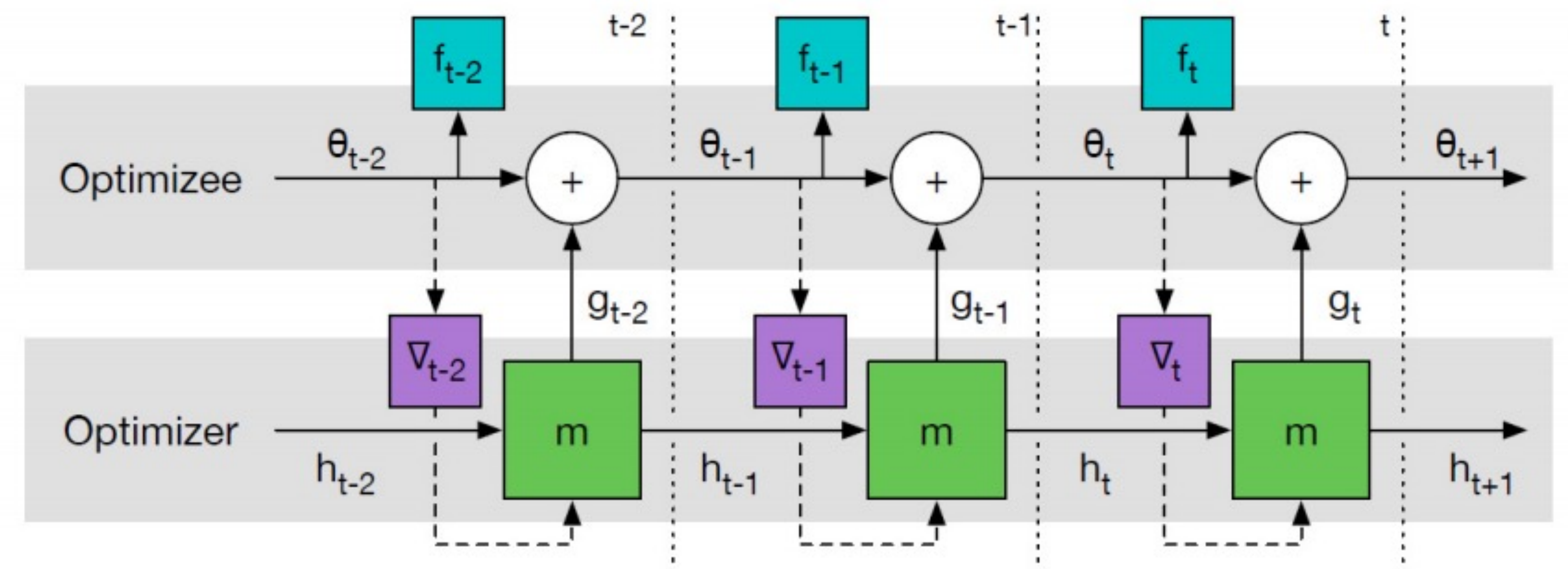
- Automatically search good network architectures
  - RL takes action of creating a network architecture and obtains reward by training & evaluating the network on a dataset
  - Barret Zoph and Quoc V. Le. Neural architecture search by reinforcement learning. ICLR 2017.

Various Meta-Learning Tasks



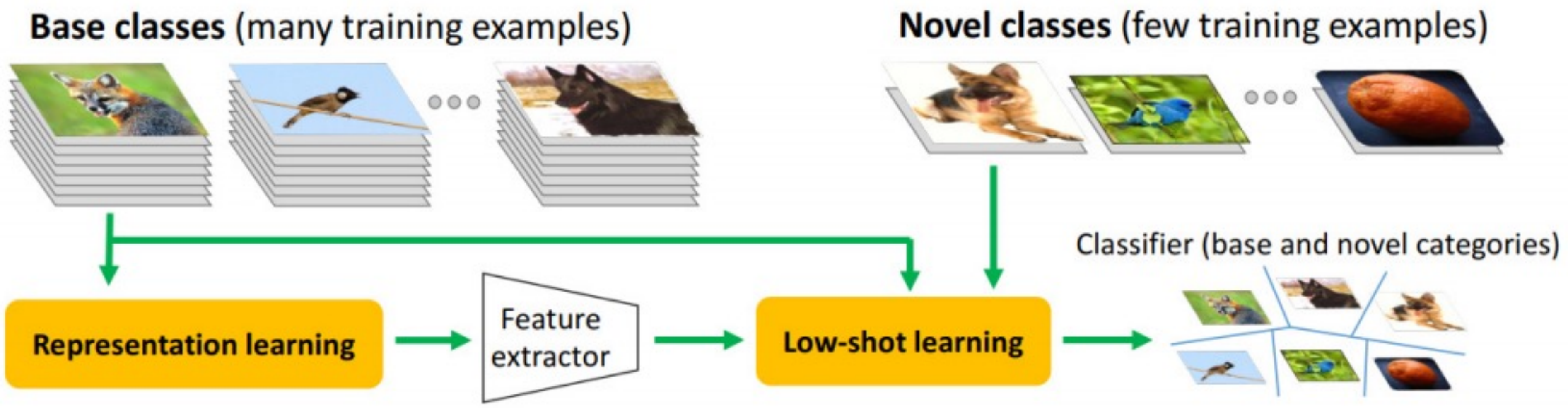
- Hyperparameter optimization
  - Compute exact gradients of cross-validation performance with respect to all hyperparameters by chaining derivatives backwards through the entire training procedure
  - Maclaurin et al. Gradient-based Hyperparameter Optimization through Reversible Learning. ICML 2015.

Various Meta-Learning Tasks



- Learn to produce good gradient
  - An RNN with hidden memory units takes in new raw gradient and outputs a tuned gradient so as to better train the model
  - Andrychowicz et al. Learning to learn by gradient descent by gradient descent. NIPS 2016.

Various Meta-Learning Tasks



- Few-shot learning
  - Learn a model from a large dataset that can be easily adapted to new classes with few instances
  - Hariharan and Girshick. Low-shot Visual Recognition by Shrinking and Hallucinating Features. ICCV 2017.

General Math Form

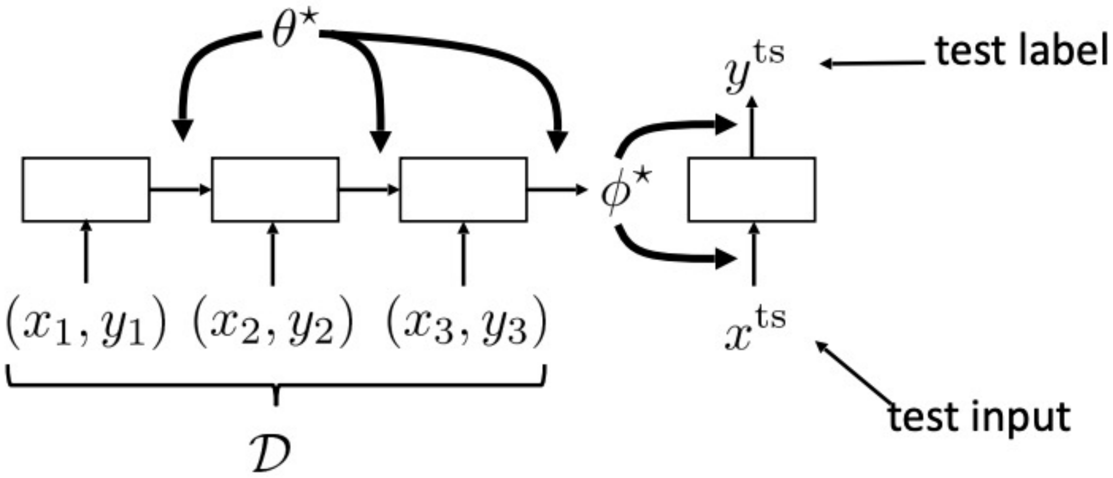
$$\text{meta-learning:} \quad \theta^\star = \arg\max_\theta \log p(\theta|\mathcal{D}_{\text{meta-train}})$$

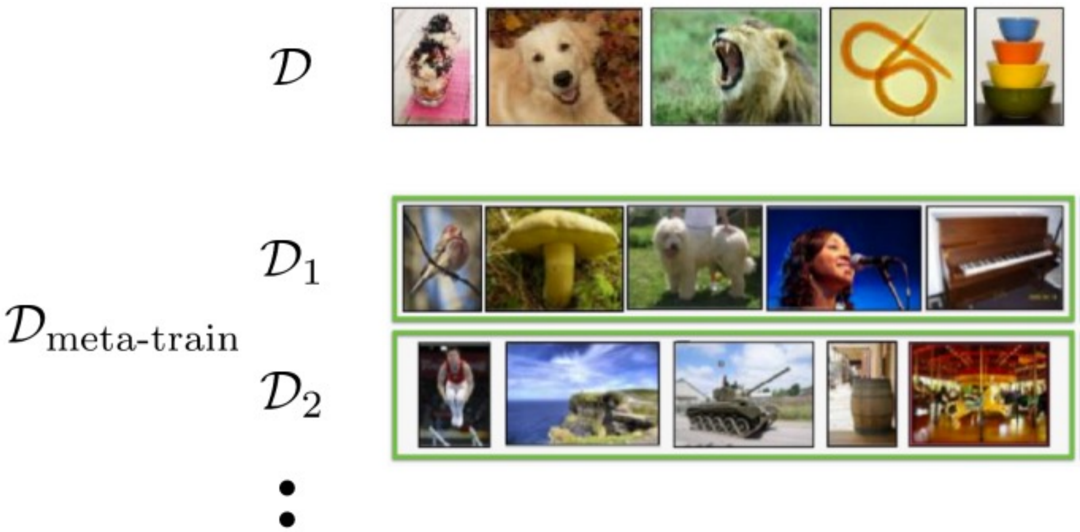
$$\boxed{\text{adaptation:} \quad \phi^\star = \arg\max_\phi \log p(\phi|\mathcal{D}, \theta^\star)}$$

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

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

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





Meta-learning Methods

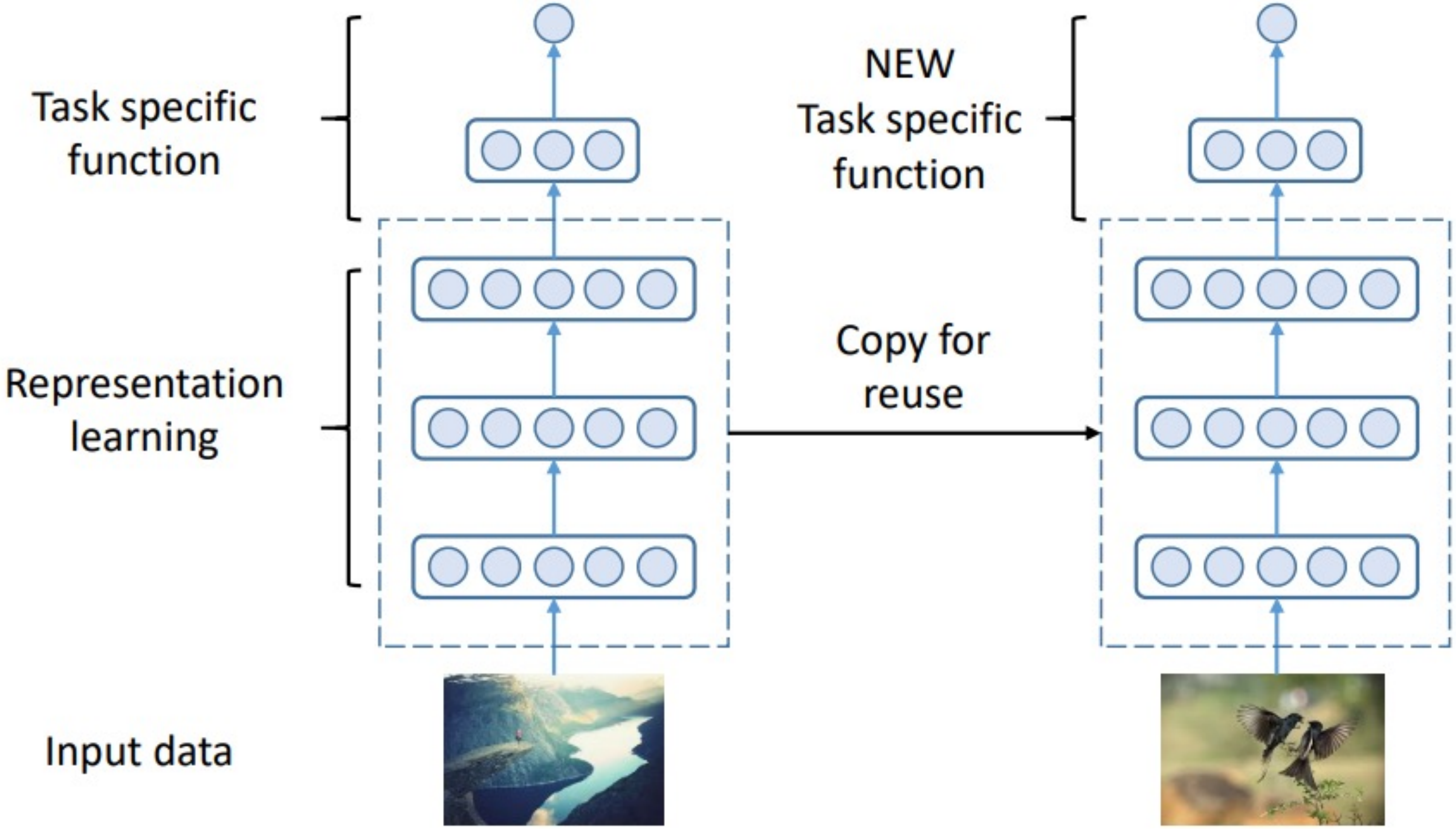
- Initialization based methods
  - Learning how to initialize the model for the new task

- Recurrent neural network methods
  - Learning how to produce good gradient in an auto-regressive manner

- Reinforcement learning methods
  - Learning how to produce good gradient in a reinforcement learning manner

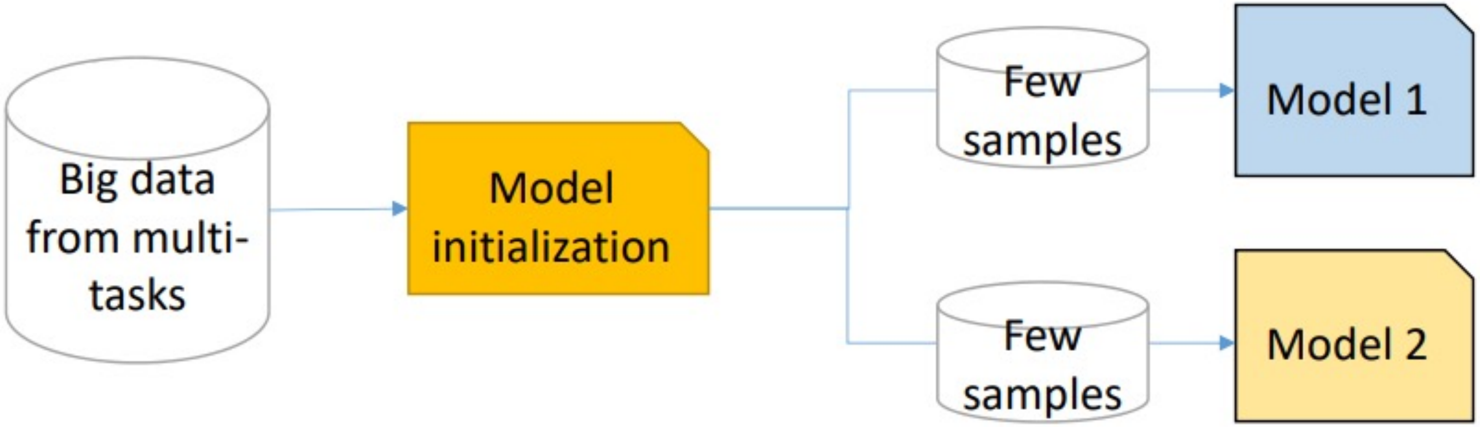
REVIEW
Network Parameter Reuse

- Treat lower layers as representation learning module and reuse them as good feature extractors



Model-Agnostic Meta Learning

- Goal: train a model that can be fast adapted to different tasks via few shots
- MAML idea: directly optimize for an initial representation that can be effectively fine-tuned from a small number of examples



Finn et al. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. ICML 2017.

Model-Agnostic Meta Learning

- Goal: train a model that can be fast adapted to different tasks via few shots

Traditional multi-task learning SGD

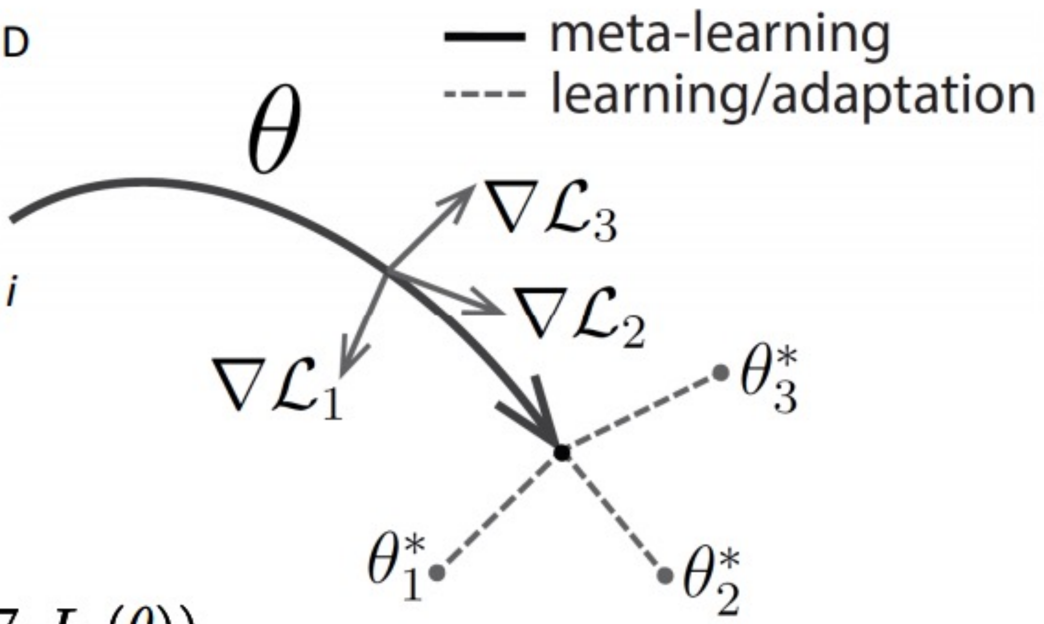
$$\theta \leftarrow \theta - \eta \sum_i \nabla_\theta L_i(\theta)$$

Imagined good parameter for task *i*

$$\theta^i \leftarrow \theta - \eta \nabla_\theta L_i(\theta)$$

MAML SGD

$$\theta \leftarrow \theta - \eta \sum_i \nabla_\theta L_i(\theta - \eta \nabla_\theta L_i(\theta))$$



Finn et al. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. ICML 2017.

Meta-learning Methods

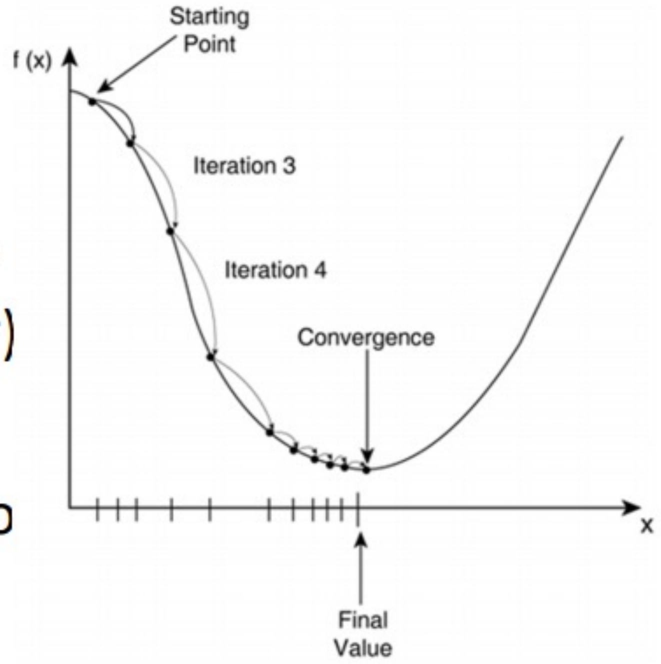
- Initialization based methods
  - Learning how to initialize the model for the new task
- Recurrent neural network methods
  - Learning how to produce good gradient in an auto-regressive manner
- Reinforcement learning methods
  - Learning how to produce good gradient in a reinforcement learning manner

Rethink About the Gradient Learning

- The traditional gradient in machine learning

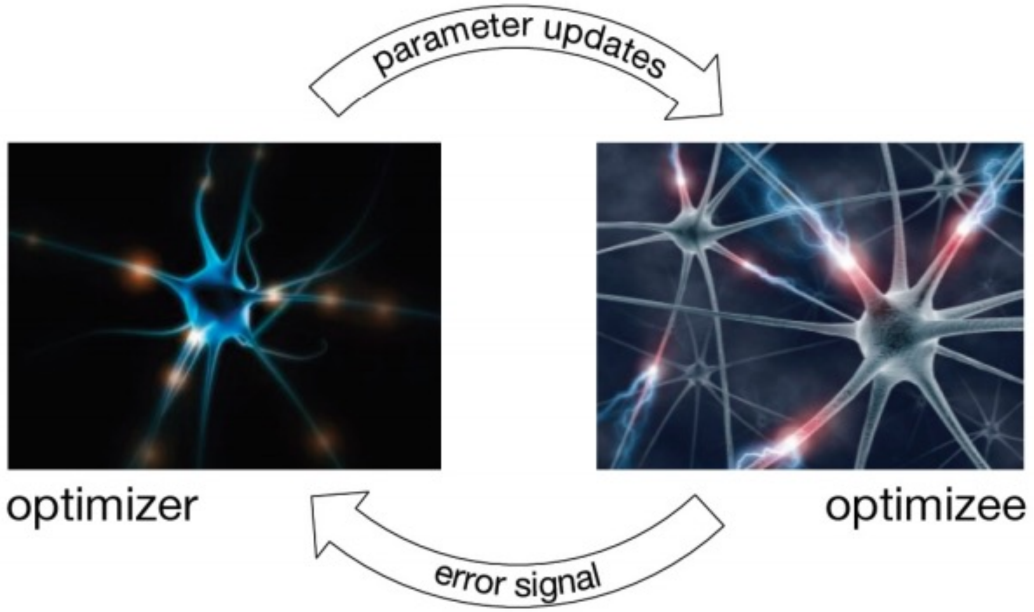
$$\theta_{t+1} = \theta_t - \eta_t \nabla_{\theta_t} L(\theta_t)$$

- Problems of it
  - Learning rate is fixed or changes with a heuristic rule
  - No consideration of second-order information (or even higher-order)
- Feasible idea
  - Memorize the historic gradients to better decide the next gradient



Andrychowicz, Marcin, et al. "Learning to learn by gradient descent by gradient descent." *NIPS* 2016.

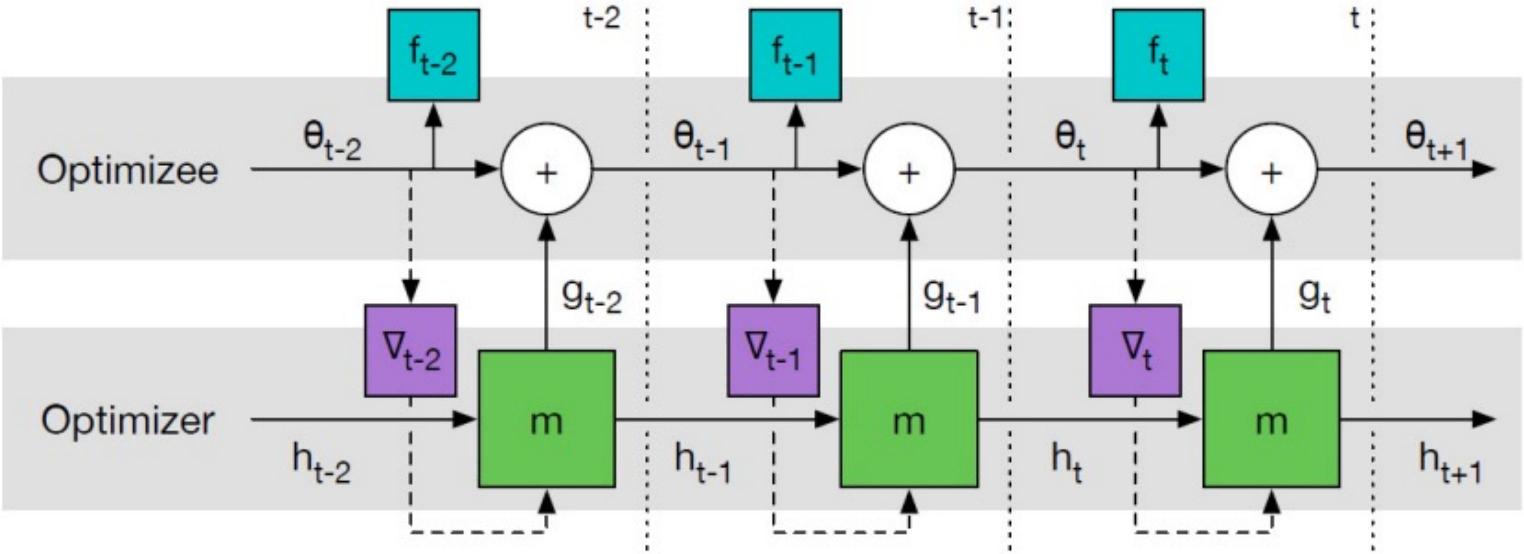
An Optimizer Module to Decide How to Optimize



- Two components: optimizer and optimize
- The optimizer (left) is provided with performance of the optimizee (right) and proposes updates to increase the optimizee's performance

Andrychowicz, Marcin, et al. "Learning to learn by gradient descent by gradient descent." *NIPS* 2016.

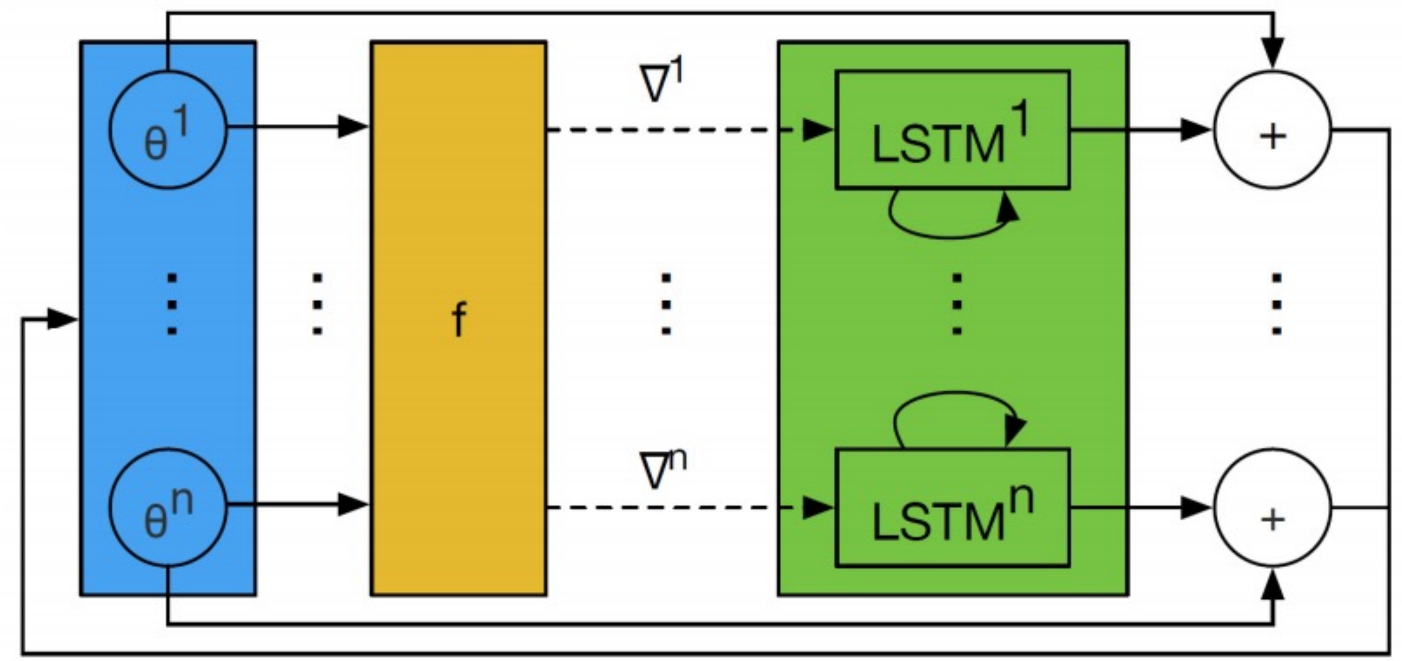
Recurrent Network for Meta-Learning



- With an RNN, the optimizer memorize the historic gradient information within its hidden layer
- The RNN can be directly updated with back-prop algorithms from the loss function

Andrychowicz, Marcin, et al. "Learning to learn by gradient descent by gradient descent." *NIPS* 2016.

Coordinatewise LSTM Optimizer



- Normally the parameter number *n* is large, thus a fully connected RNN is not feasible to train
- Above presents a coordinated LSRM, i.e., an $LSTM^i$ for each individual parameter ϑ^i with shared LSTM parameters

Andrychowicz, Marcin, et al. "Learning to learn by gradient descent by gradient descent." *NIPS* 2016.

Meta-learning Methods

- Initialization based methods
  - Learning how to initialize the model for the new task
- Recurrent neural network methods
  - Learning how to produce good gradient in an auto-regressive manner
- Reinforcement learning methods
  - Learning how to produce good gradient in a reinforcement learning manner

Review of Meta-Learning Methods

Algorithm 1 General structure of optimization algorithms

Require: Objective function f
  $x^{(0)} \leftarrow$ random point in the domain of f
  for $i = 1, 2, \ldots$ **do**
    $\boxed{\Delta x \leftarrow \phi(\{x^{(j)}, f(x^{(j)}), \nabla f(x^{(j)})\}_{j=0}^{i-1})}$
    if stopping condition is met **then**
      return $x^{(i-1)}$
    end if
    $x^{(i)} \leftarrow x^{(i-1)} + \Delta x$
  end for

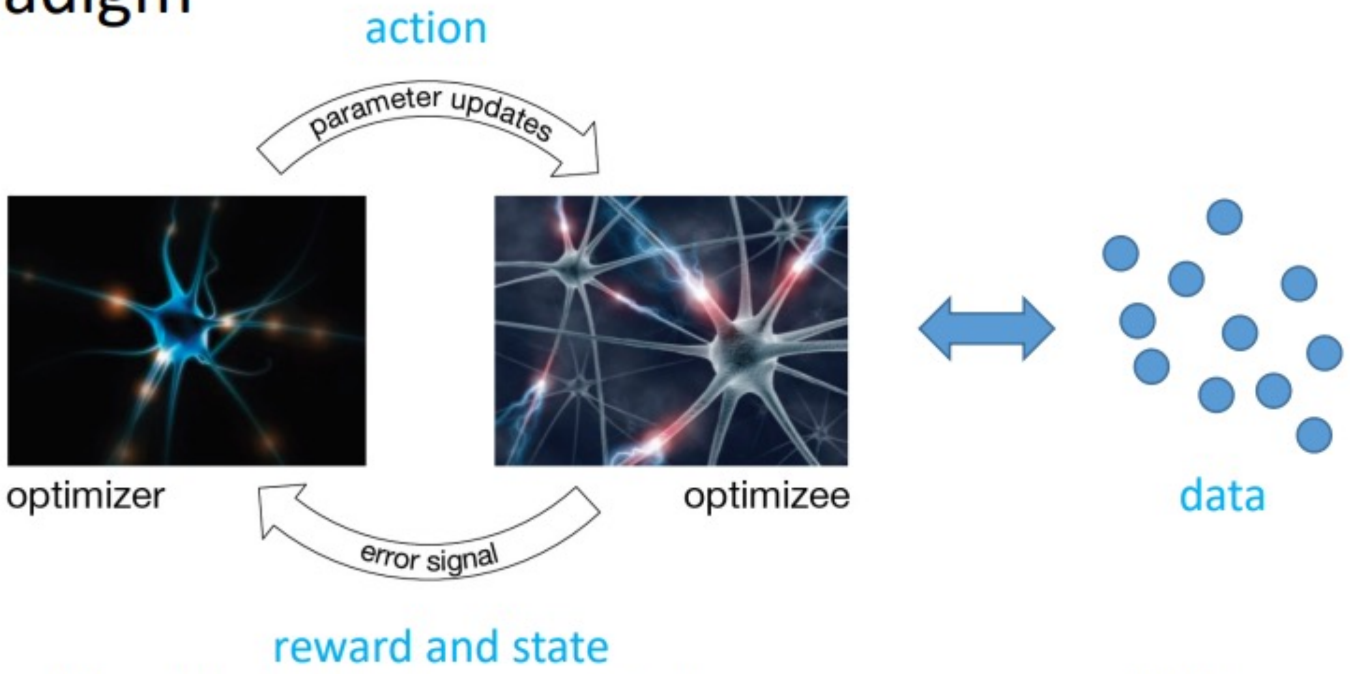
Gradient descent	$\phi(\cdot) = -\eta \nabla f(x^{(i-1)})$
Momentum	$\phi(\cdot) = -\eta \left(\sum_{j=0}^{i-1} \eta^{i-1-j} \nabla f(x^{(j)}) \right)$
Learned Algo.	$\phi(\cdot) =$ Neural Net

- The key of meta-learning (or learning to learn) is to design a good function that
  - takes previous observations and learning behaviors
  - outputs appropriate gradient for the ML model to update

Li, Ke, and Jitendra Malik. "Learning to optimize." *arXiv preprint arXiv:1606.01885* (2016).

Reinforcement Learning for Meta-Learning

- Idea: at each timestep, the meta-learner learns to deliver an optimization action to the learner and then observe the performance of the learner, which is very similar with reinforcement learning paradigm



Li, Ke, and Jitendra Malik. "Learning to optimize." *arXiv preprint arXiv:1606.01885* (2016).

Formulation as an RL Problem

Algorithm 1 General structure of optimization algorithms

Require: Objective function f
  $x^{(0)} \leftarrow$ random point in the domain of f
  for $i = 1, 2, \ldots$ **do**
    $\Delta x \leftarrow \boxed{\phi(\Phi(\{x^{(j)}, f(x^{(j)}), \nabla f(x^{(j)})\}_{j=0}^{i-1}))}$ Policy
    if stopping condition is met **then**
      return $x^{(i-1)}$
    end if Action
  $\boxed{\Phi(\cdot) \text{ and } x^{(i)}} \leftarrow x^{(i-1)} + \boxed{\Delta x}$ $L(x^{(i)})$
  end for

Space Reward

- State representation is generated from a function $\Phi(\cdot)$ mapping the observed data and learning behavior to a latent representation
- The policy outputs the gradient which is the action
- The reward is from the loss function w.r.t. the current model parameters

Li, Ke, and Jitendra Malik. "Learning to optimize." *arXiv preprint arXiv:1606.01885* (2016).

Learning to Optimize (L2O):

A Rising AutoML field

Research | Academics & Admissions | People | News & Events | Outreach

Academics & Admissions

Academic Information

MIT Professional Education

EECS curriculum - dynamic graphical display

Subject Updates Fall 2019

Subject UPDATES Spring 2019

6.S077

6.S081

6.S082/6.888

6.645

6.890 Learning-Augmented Algorithms

SHARE:

Graduate Level
Units: 3-0-9
Prerequisites: 6.036 or equivalent, 6.046 or equivalent

TOYOTA TECHNOLOGICAL INSTITUTE AT CHICAGO

Summer Workshop on Learning-Based Algorithms

This workshop will cover recent developments in using machine learning to improve the performance of "classical" algorithms, by adapting their behavior to the properties of the input distribution. This reduces their running time, space usage or improves their accuracy, while (often) retaining worst case guarantees.

The workshop will cover general approaches to designing such algorithms, as well as specific case studies. We plan to cover learning-augmented methods for designing data structures, streaming and sketching algorithms, on-line algorithms, compressive sensing and recovery, error-correcting codes, scheduling algorithms, and combinatorial optimization. The attendees span a diverse set of areas, including theoretical computer science, machine learning, algorithmic game theory, coding theory, databases and systems.

When: 12 - 14 August 2019
Where: Toyota Technological Institute at Chicago (TTIC)
6045 S Kenwood Ave,
Chicago, IL 60637
Organizers: Piotr Indyk (MIT), Yaron Singer (Harvard), Ali Vakilian (MIT) and Sergei Vassilvitskii (Google NYC)

Many lenses to "view" optimization as learning

- *Reinforcement Learning (current variable -> state; loss -> reward; update -> action)*
- *Markovian Chain/Recurrent NN*
- *Feedback-loop System (unrolling)*

... leading to many ways to make optimization "learnable"

Definition: What is learning to optimize (L2O)?

Using ML to improve classical optimization algorithms, by **adapting** their behaviors to *problem & data of interest*, via:

- **tuning** existing algorithms *(e.g., LISTA)* - *"white box"*
- **creating** new algorithms *(LSTM, RL …)* - *"black box"*
- and many **in between** *(PnP, RED, DIP, …)* - *"gray box"*

Learning to Optimize:

A field of our passions…

“Learning to Optimize: A Primer and A Benchmark”, arXiv 2021/3

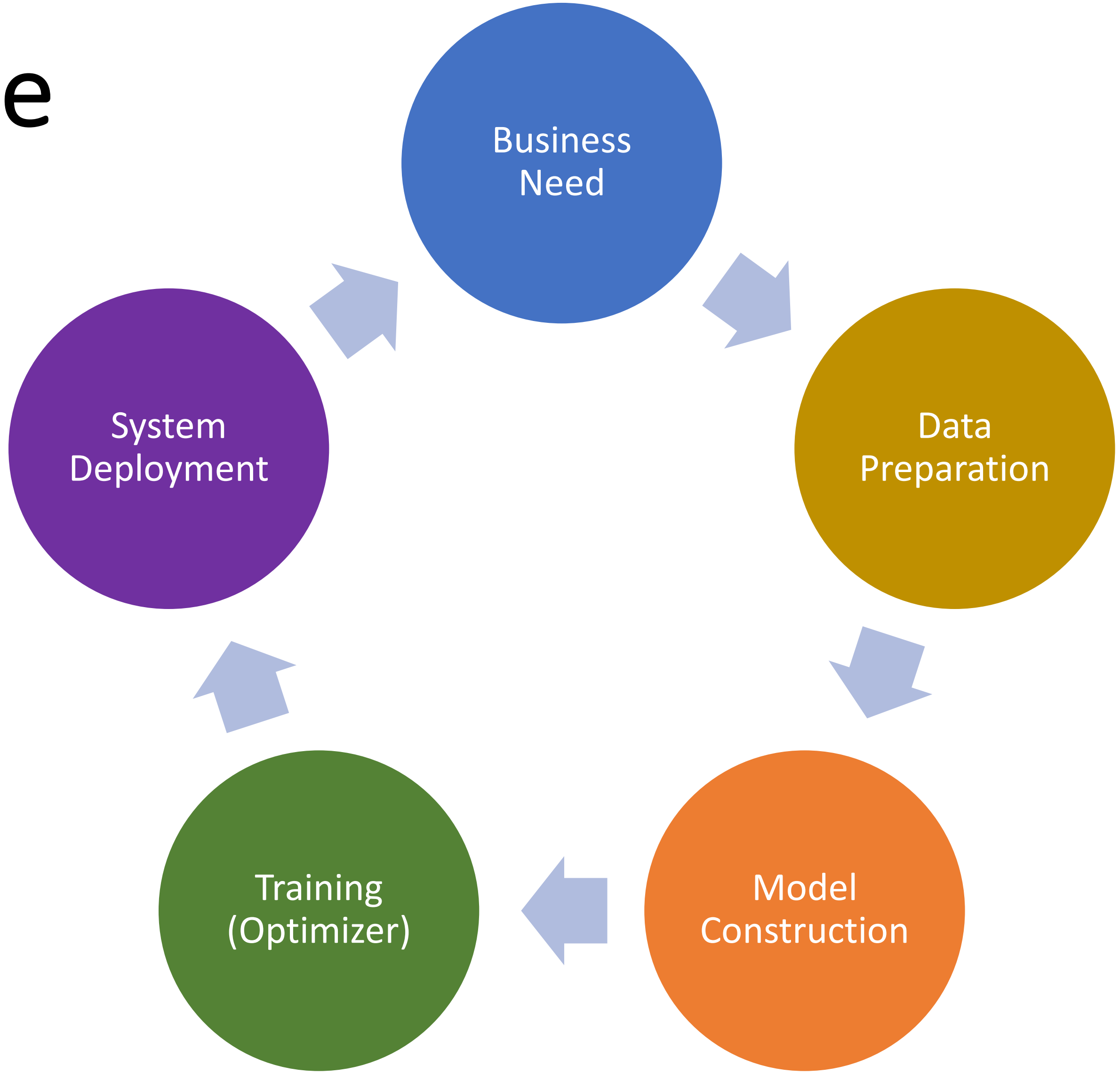
From convex to nonconvex optimization

- [NeurIPS’18, ICLR’19, NeurIPS’21] A full theory framework for L2O in sparse optimization, including convergence (rate) and minimal parameterization
- [ICML’19] Relaxed convergence proof, for learned plug-and-play solver to inverse problems, and how to practically ensure that convergence
- [NeurIPS’19] The first learned solver for swarm particle optimization
- [ICLR’21] Learned solver for minimax optimization

Going for deep networks: L2O has new and powerful applications!

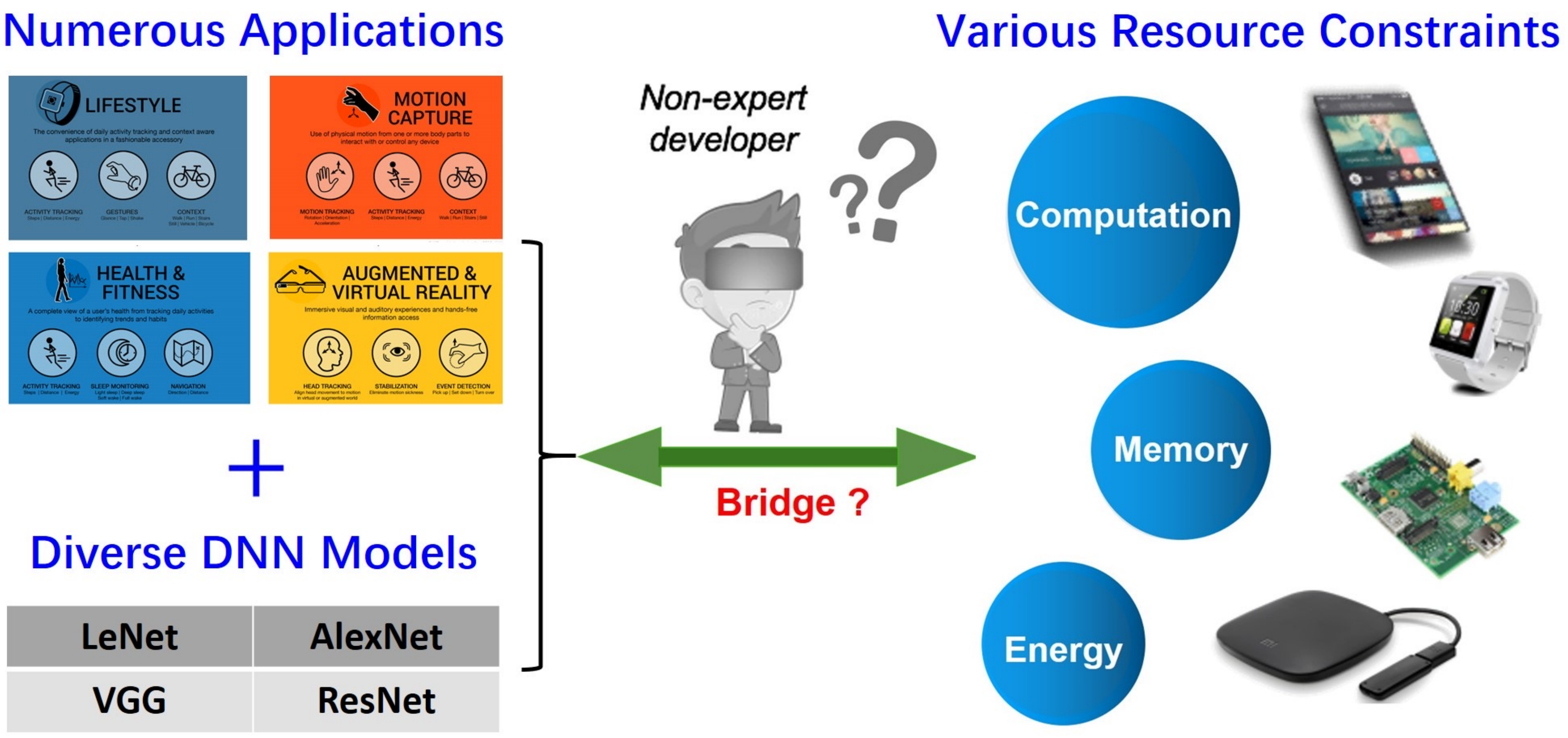
- [CVPR’20] Learned fast greedy optimizer for graph neural network
- [ICML’20] L2O for unsupervised domain generalization
- [ICML’20] L2O for noisy label training
- [ECCV’20] Learning to personalize/adapt deep networks on hardware
- [NeurIPS’20] How to meta-learn L2O better
- [AAAI’21] Learning to train differentially private deep networks
- More ……

Automating ML Lifecycle



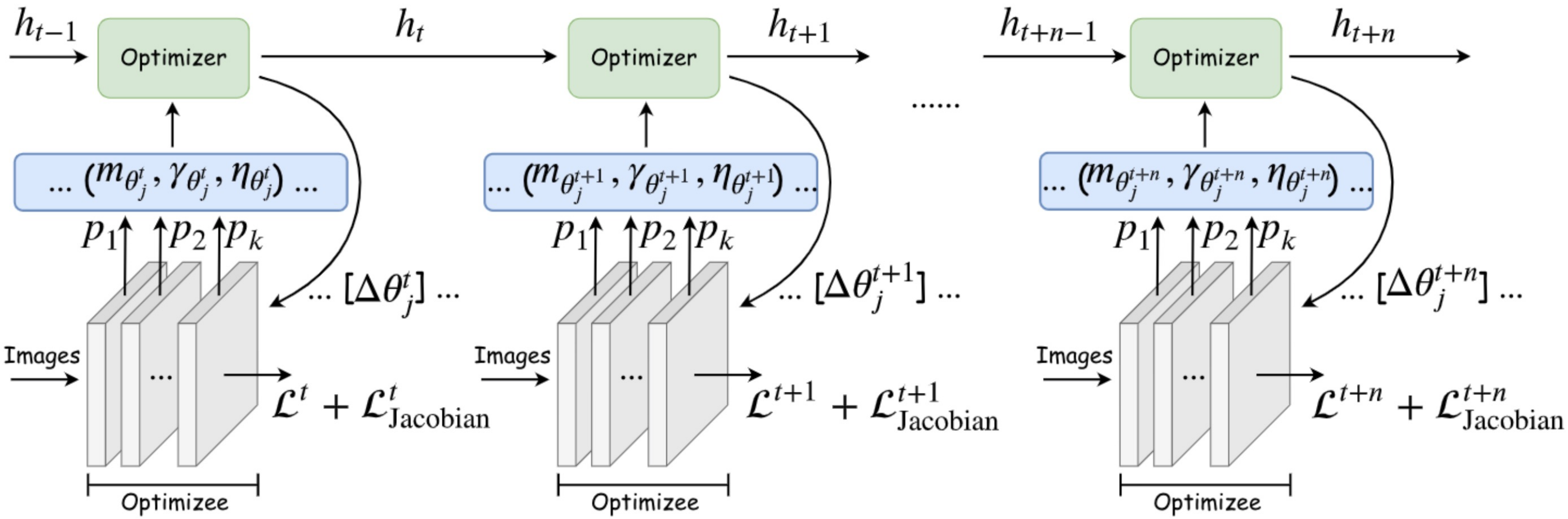
Automated System Deployment: Model-Hardware

- Automatically Discover ML Model with Hardware/Platform Awareness



Automated System Deployment: Algorithm-Hardware

- Automatically Discover ML Algorithm with Hardware/Platform Awareness



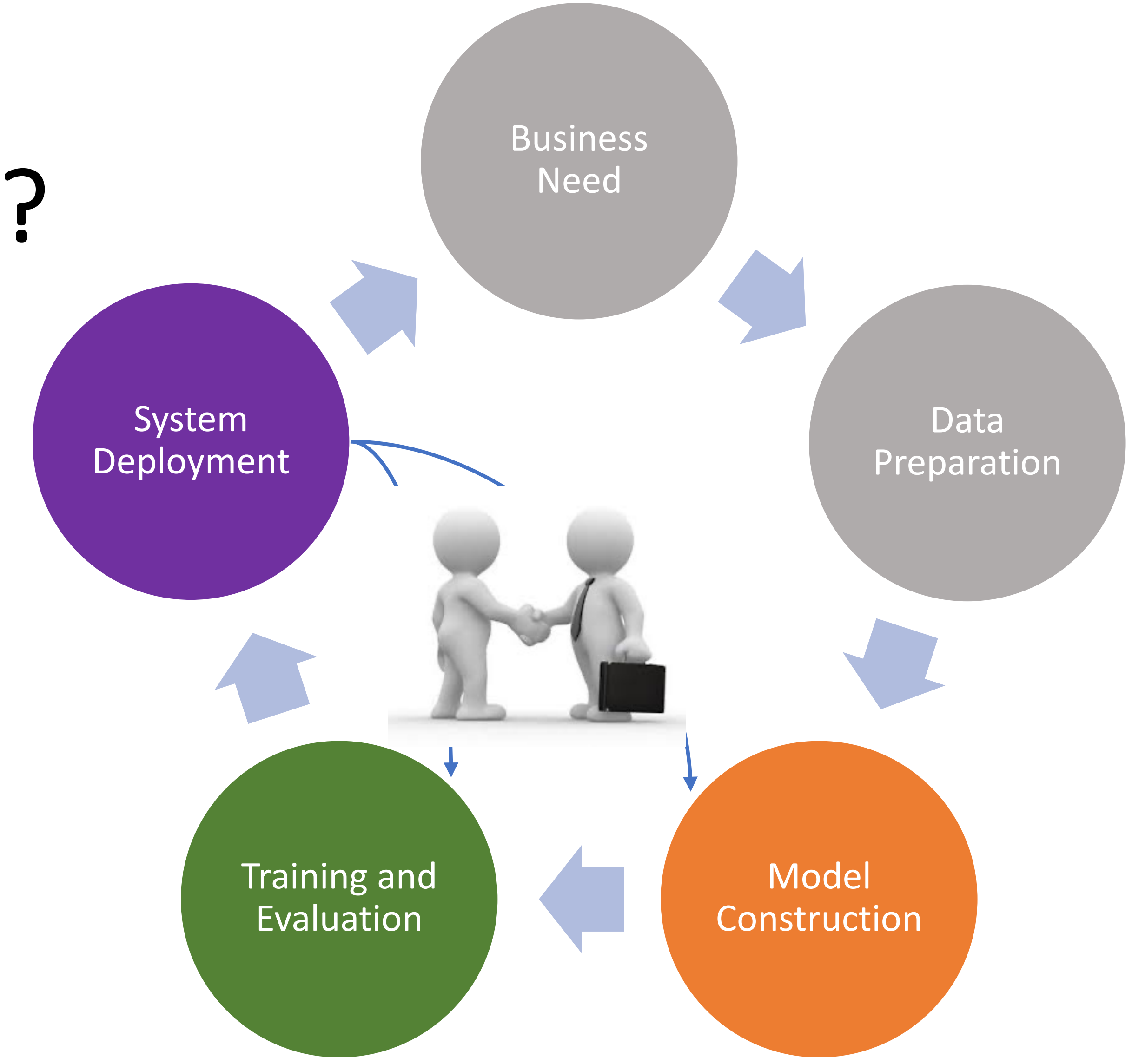
Hardware-Aware Learning-to-Optimize (HALO)

for Efficient On-Device ML Adaptation [ECCV'20]

Automated System Deployment: **Next Level?**

- Hardware/Platform Awareness:
  now as given, fixed constraint …

- Boost to Next Level Gain:
  Automated System-Level Co-Optimization!
  - *Isolated (auto)-design is never optimal!*



The University of Texas at Austin
Electrical and Computer Engineering
Cockrell School of Engineering