清華大學
Tsinghua University

media and network lab

AutoML and Meta-learning for Multimedia

Wenwu Zhu Xin Wang Yue Liu Wenpeng Zhang
Tsinghua University

Why AutoML and Meta-learning ?

- AutoML: Automated Machine Learning

- Meta-learning: Learning to learn

Adapt to different tasks, different data!
Need meta-knowledge!

Important for advanced machine learning!
Useful for multimedia!

This Tutorial Will Include

- AutoML for Multimedia
- Meta-learning for Multimedia
- Practical Experiences in NeurIPS 2018 AutoML Challenge

清華大學
Tsinghua University

media and network lab

AutoML for Multimedia

Outline

- Introduction
- Neural Architecture Search (NAS)
- Hyperparameter Optimization (HPO)
- Application: Hyperparameter Optimization for Massive Network Embedding

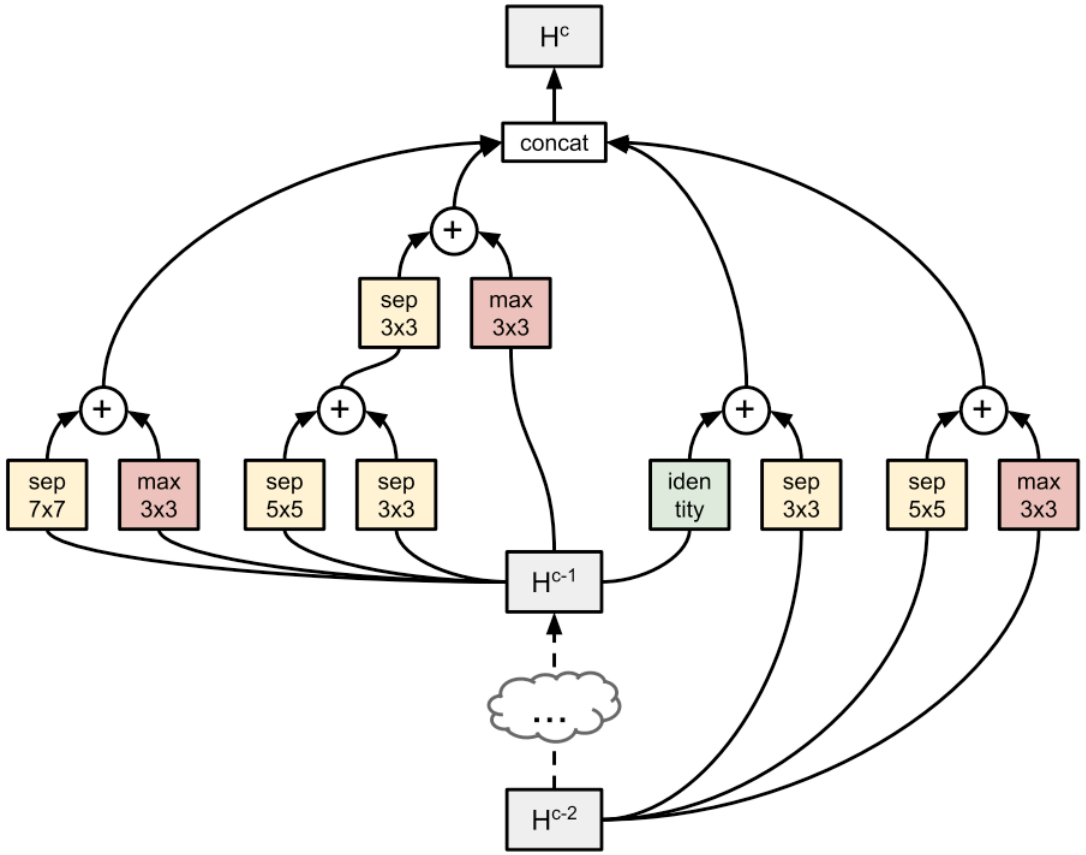
Outline

- Introduction
- Neural Architecture Search (NAS)
- Hyperparameter Optimization (HPO)
- Application: Hyperparameter Optimization for Massive Network Embedding

Introduction

- AutoML
  - A meta-approach to generate machine learning algorithms
  - Automatically search v.s. manually design (and search)

- AutoML for Deep Learning
  - Neural architecture search
  - Hyper-parameters turning
  - Loss function
  - Data augmentation
  - Activation function
  - Backpropagation



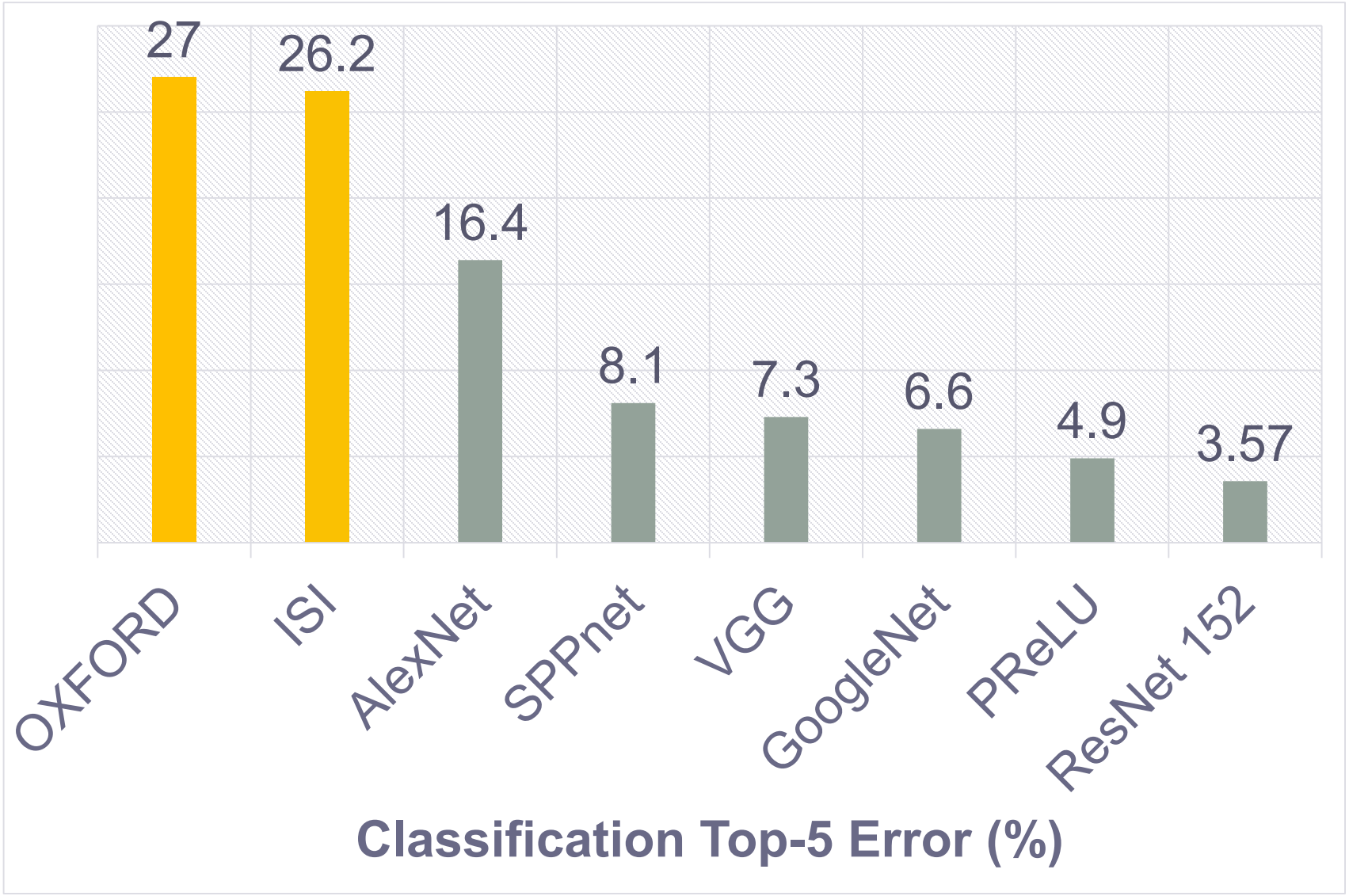
Revolution of Deep Learning

- ImageNet 2012 -
  - **Hand-craft feature**

    v.s.

  - **Deep representation**

- Age of **Deep Learning** begins!

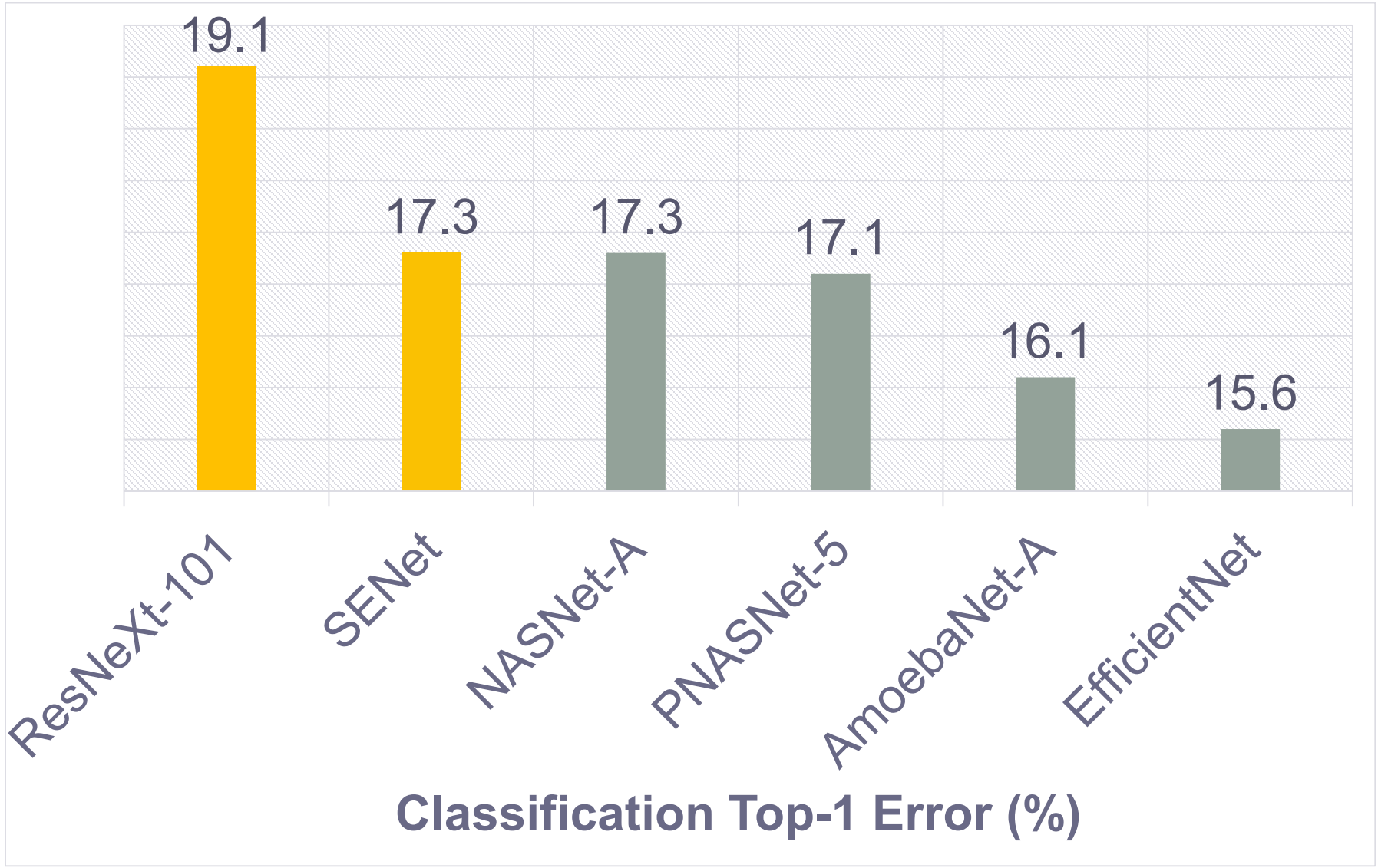


Revolution of AutoML

Manually designed structure

v.s.

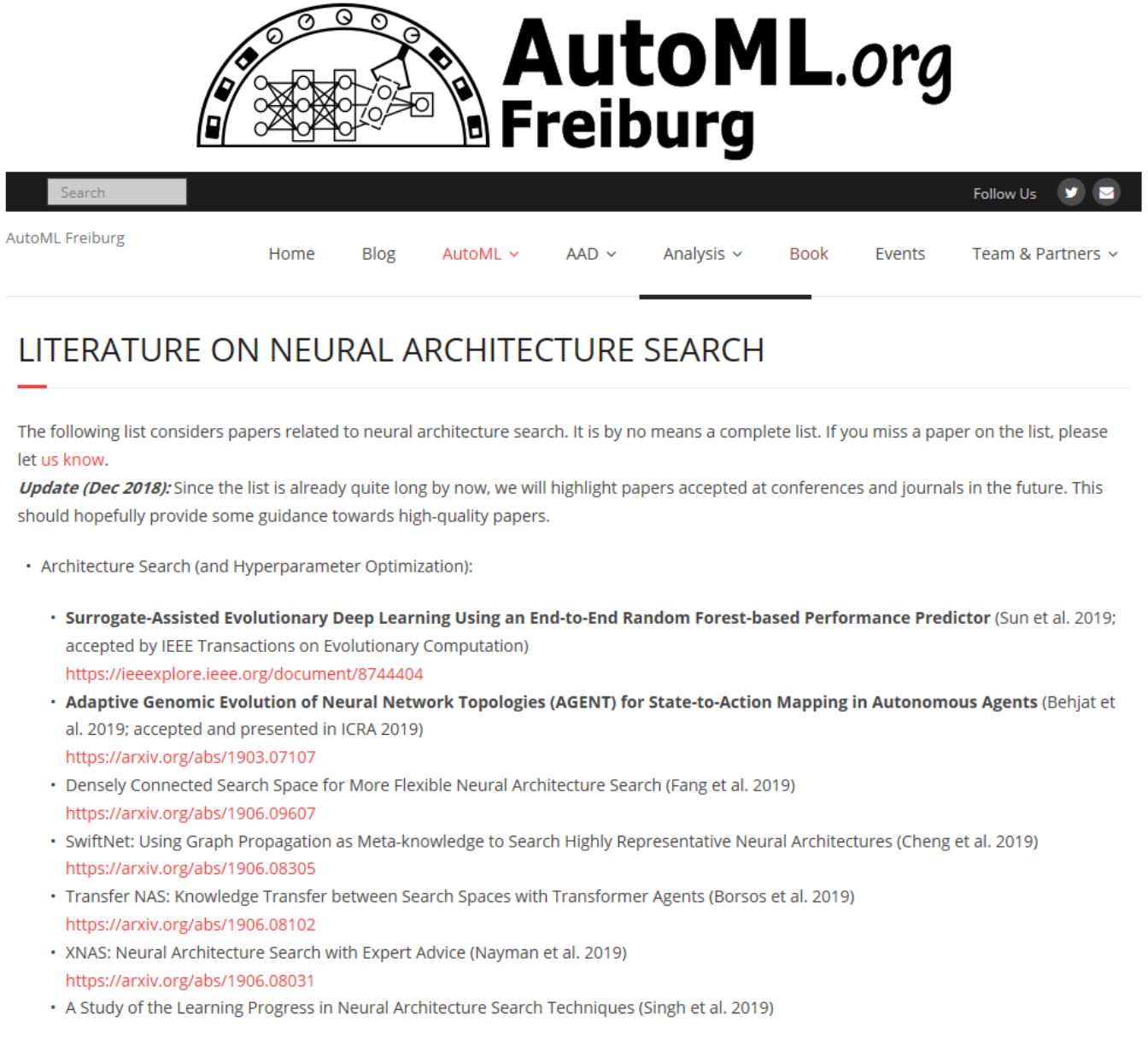
Automated searched structure



Revolution of AutoML

Literature

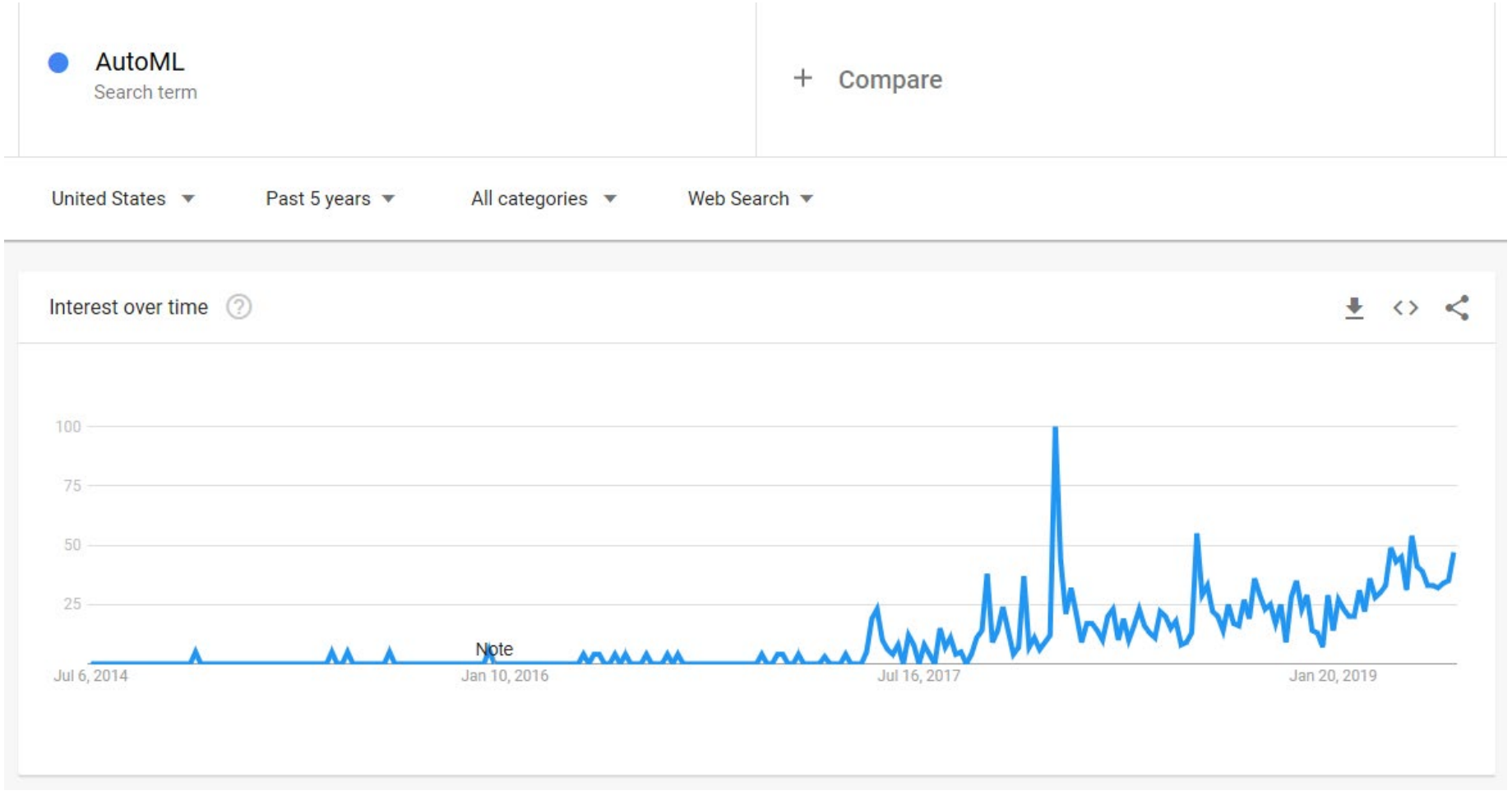
More than 200 papers since 2017.



Revolution of AutoML

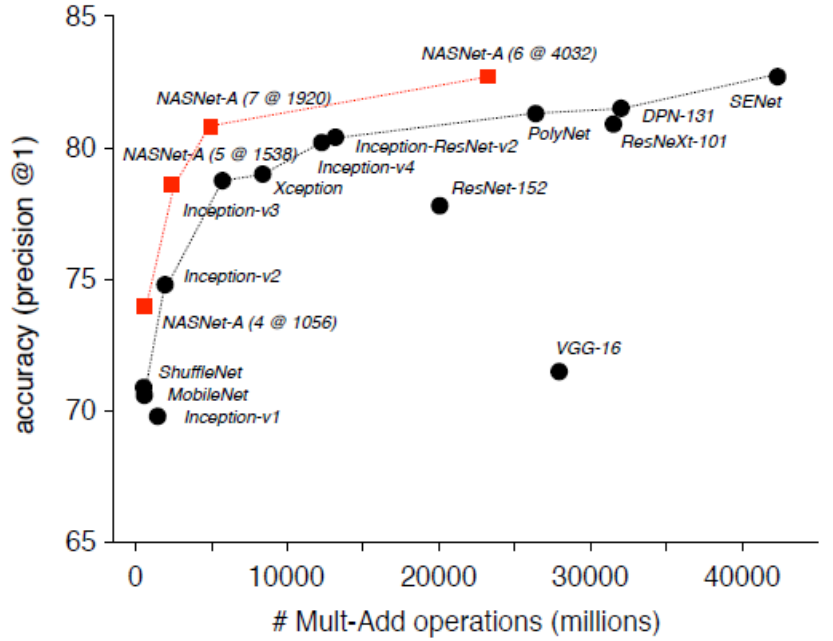
Google Trends

Increasing trend of the key word AutoML.

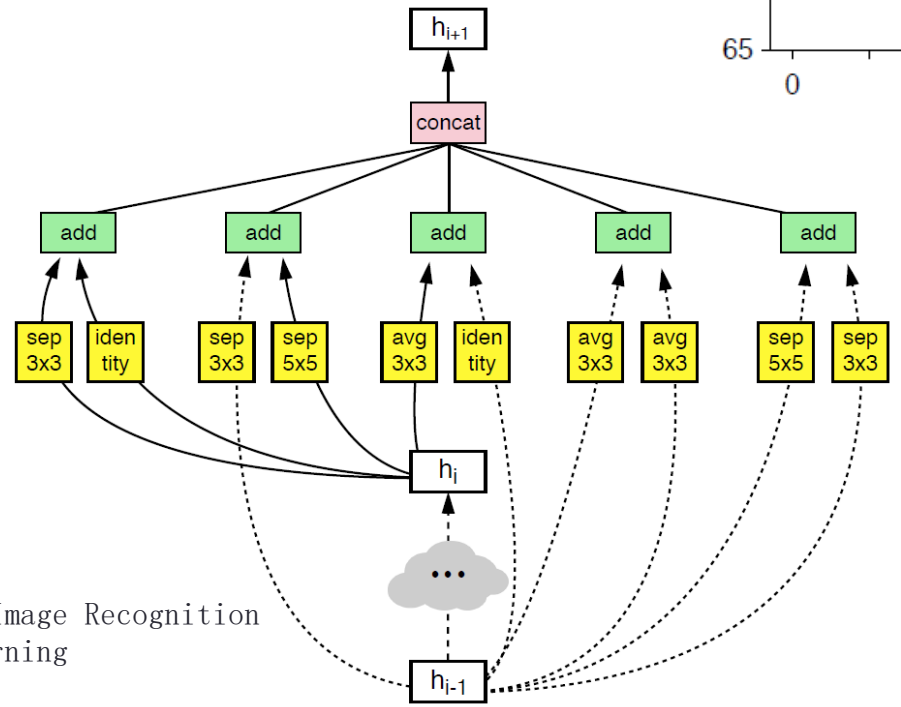


A General Overview for AutoML in Deep Learning

- Surpassing handcraft models
  - NASNet



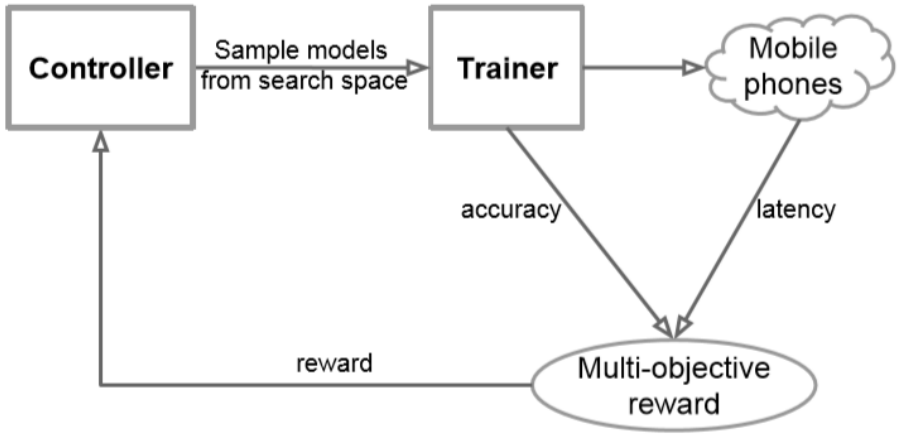
- Points
  - RNN controller + policy gradient
  - Flexible search space
  - Drawback: Proxy task needed



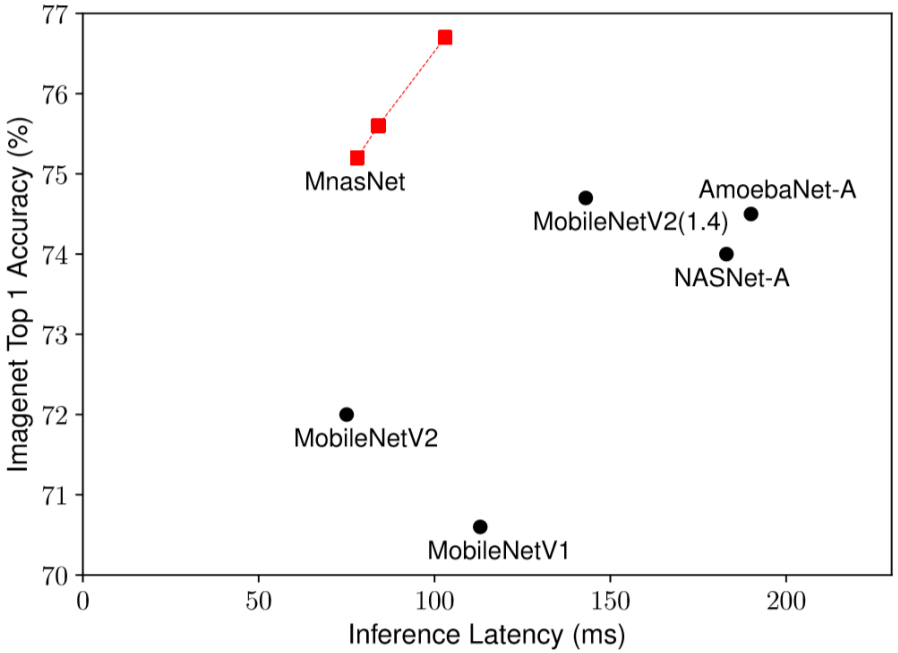
Zoph et al. Learning Transferable Architectures for Scalable Image Recognition
Zoph et al. Neural Architecture Search with Reinforcement Learning

A General Overview for AutoML in Deep Learning

- Search on the target task
  - MnasNet



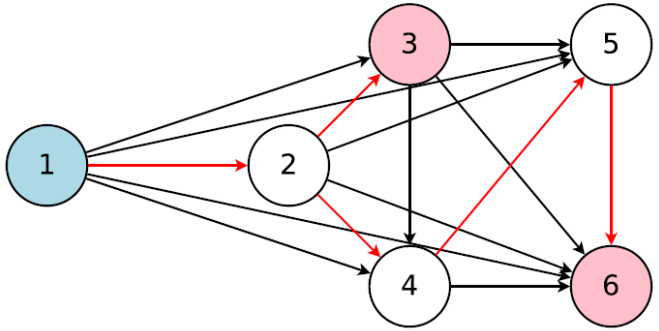
- Points
  - Search directly on ImageNet
  - Platform aware search
  - Drawback: Very costly (thousands of TPU-days)

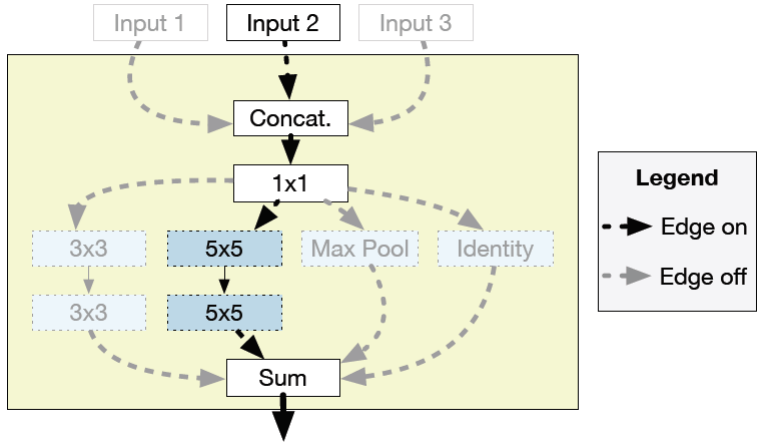


Tan et al. MnasNet: Platform-Aware Neural Architecture Search for Mobile

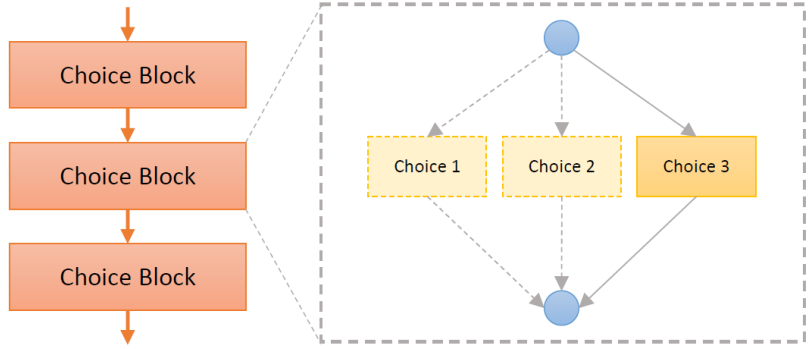
A General Overview for AutoML in Deep Learning

- Weight Sharing for Efficient Search & Evaluation
  - ENAS
  - One-shot methods





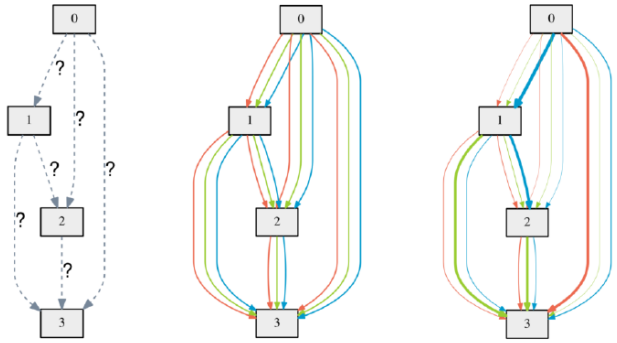
- Points
  - Super network
  - Finetuning & inference only instead of retraining
  - Drawback: Inconsistency in super net evaluation

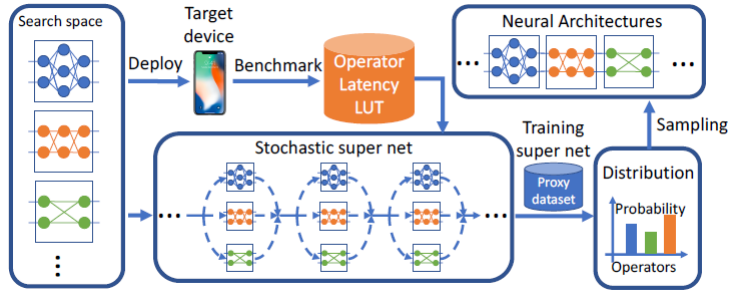


Pham et al. Efficient Neural Architecture Search via Parameter Sharing
Bender et al. Understanding and Simplifying One-Shot Architecture Search
Guo et al. Single Path One-Shot Neural Architecture Search with Uniform Sampling

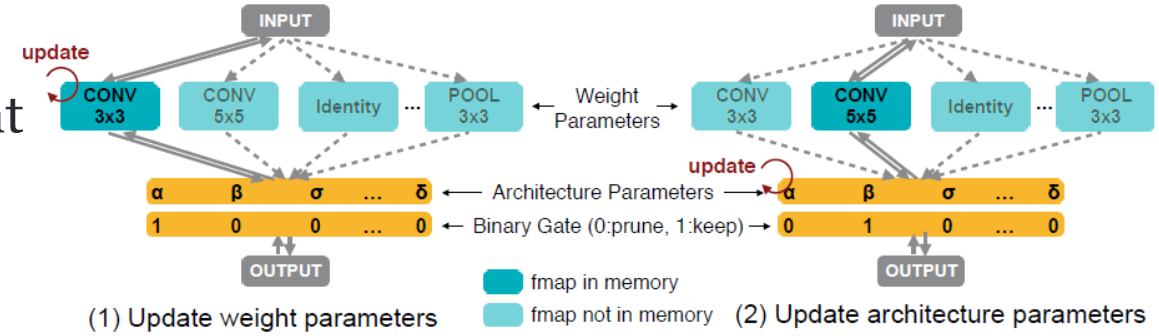
A General Overview for AutoML in Deep Learning

- Gradient-based methods
  - DARTS
  - SNAS, FBNet, ProxylessNAS, …





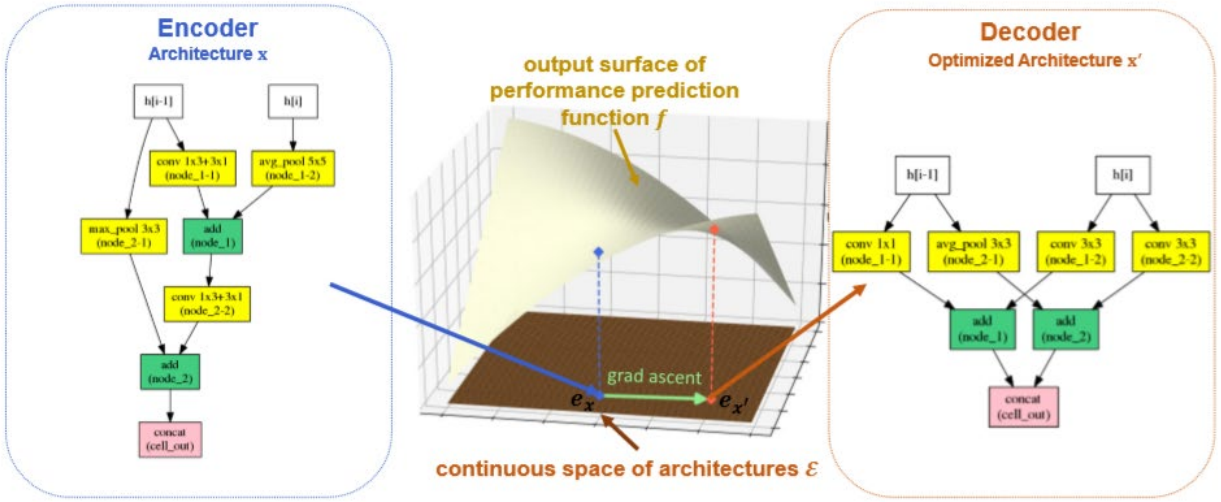
- Points
  - Joint optimization of architectures and weight
  - Weight sharing implied
  - Drawback: Sometimes less flexible

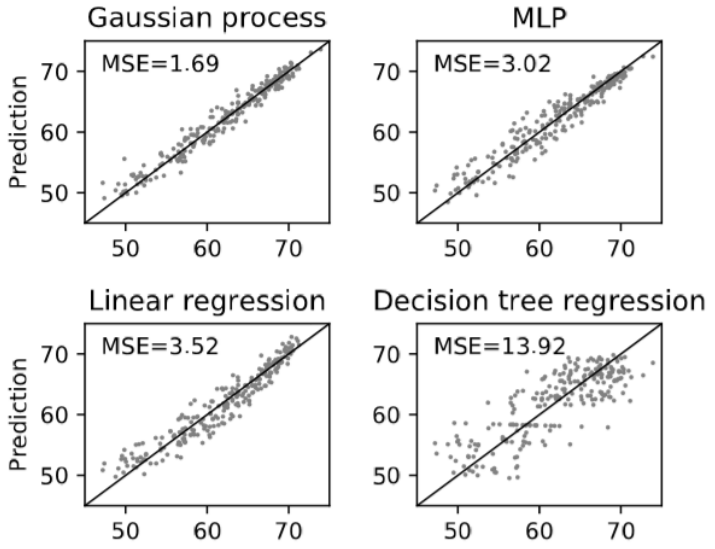


Liu et al. DARTS: Differentiable Architecture Search
Xie et al. SNAS: Stochastic Neural Architecture Search
Cai et al. ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware
Wu et al. FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search

A General Overview for AutoML in Deep Learning

- Performance Predictor
  - Neural Architecture Optimization
  - ChamNet
- Points
  - Architecture encoding
  - Performance prediction models
  - Drawback: Cold start problem



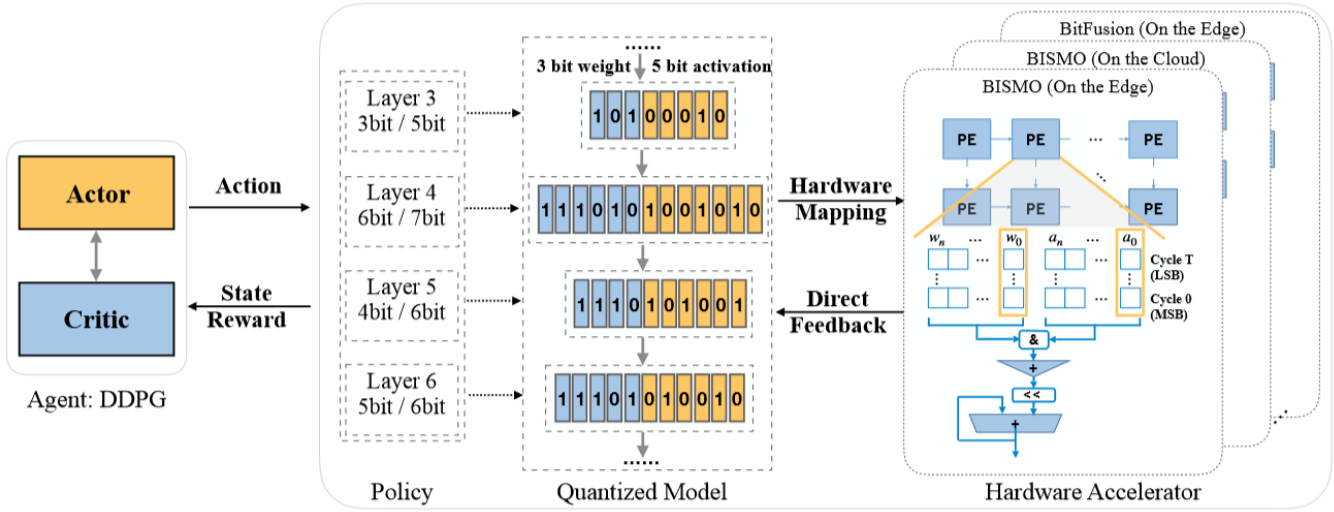


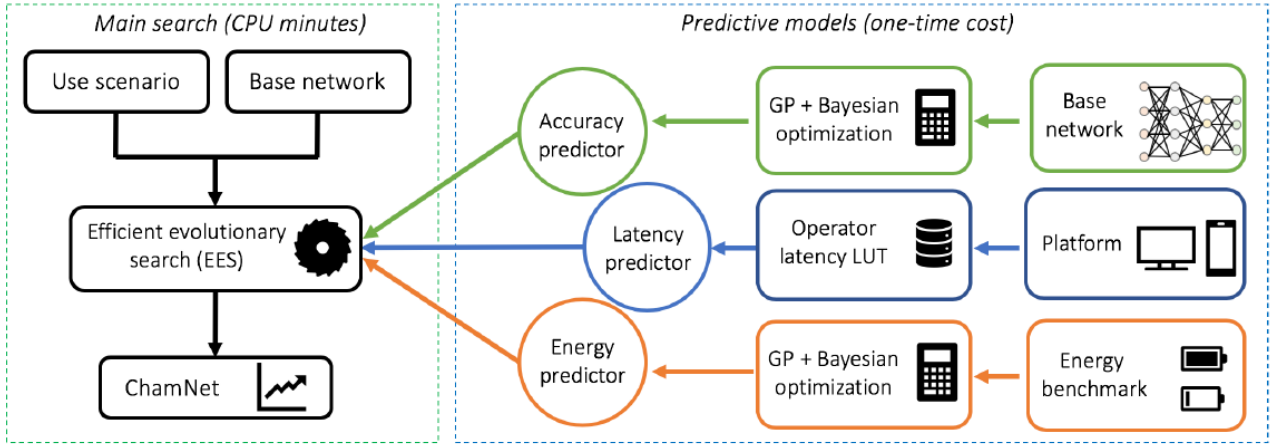
Luo et al. Neural Architecture Optimization
Dai et al. ChamNet: Towards Efficient Network Design through Platform-Aware Model Adaptation

A General Overview for AutoML in Deep Learning

- Hardware-aware Search
  - Search with complexity budget
  - Quantization friendly
  - Energy-aware search

  ...
- Points
  - Complexity-aware loss & reward
  - Multi-target search
  - Device personalized search

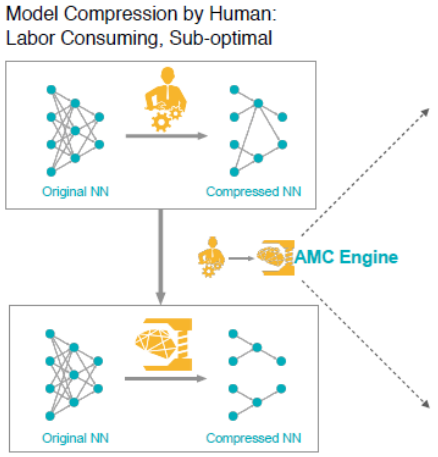


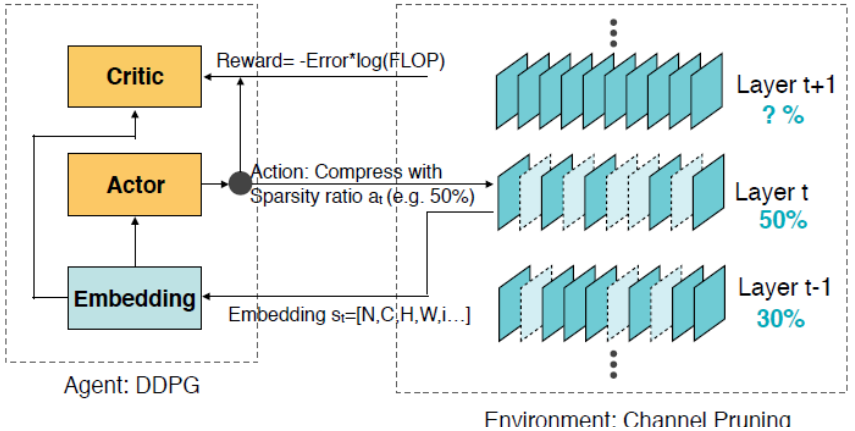


Wu et al. Mixed Precision Quantization of ConvNets via Differentiable Neural Architecture Search
V´eniat et al. Learning Time/Memory-Efficient Deep Architectures with Budgeted Super Networks
Wang et al. HAQ: Hardware-Aware Automated Quantization with Mixed Precision

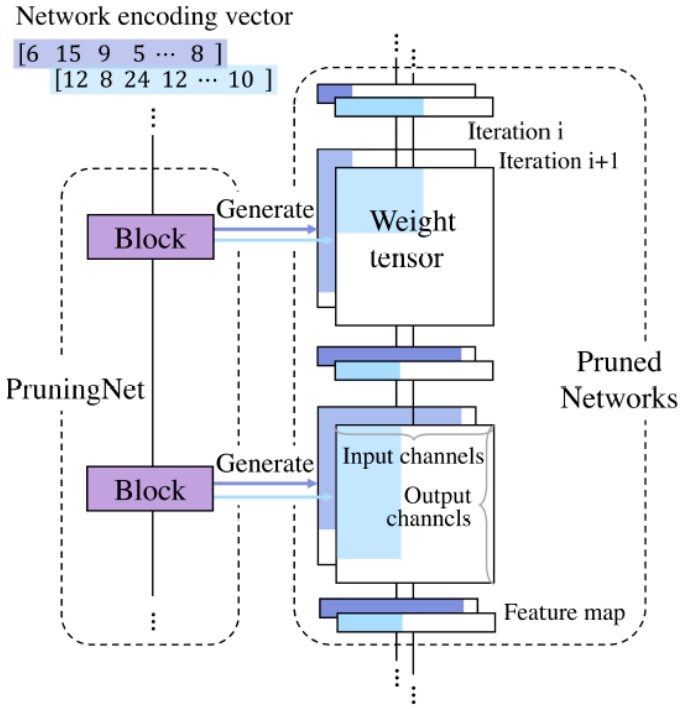
A General Overview for AutoML in Deep Learning

- AutoML in Model Pruning
  - NetAdapt
  - AMC
  - MetaPruning





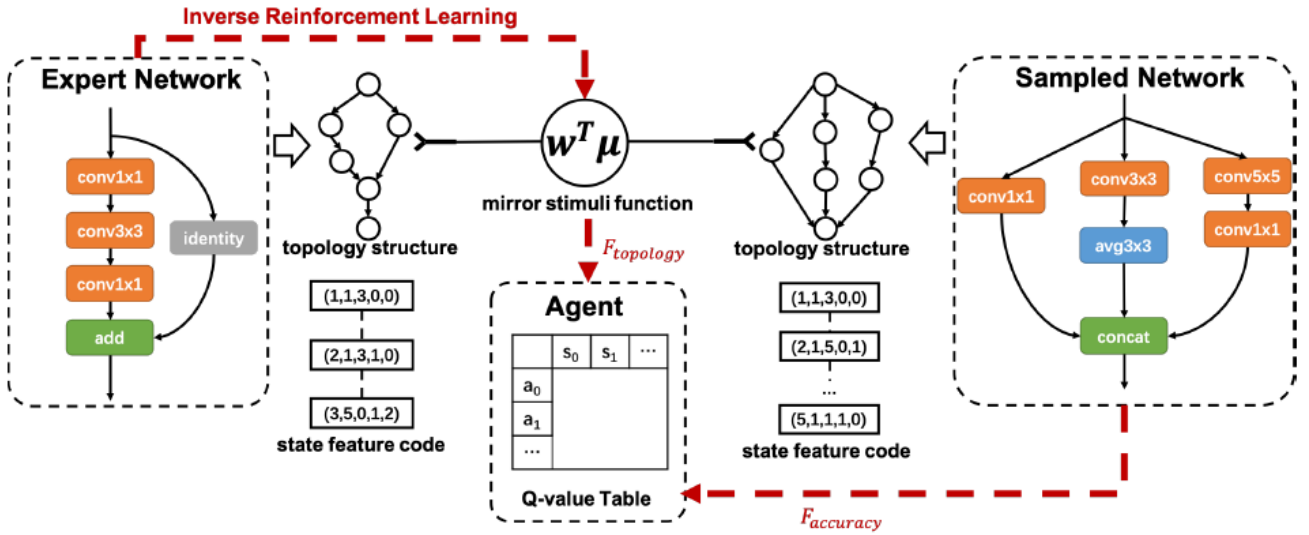
- Points
  - Search for the pruned architecture
  - Hyper-parameters like channels, spatial size, etc.



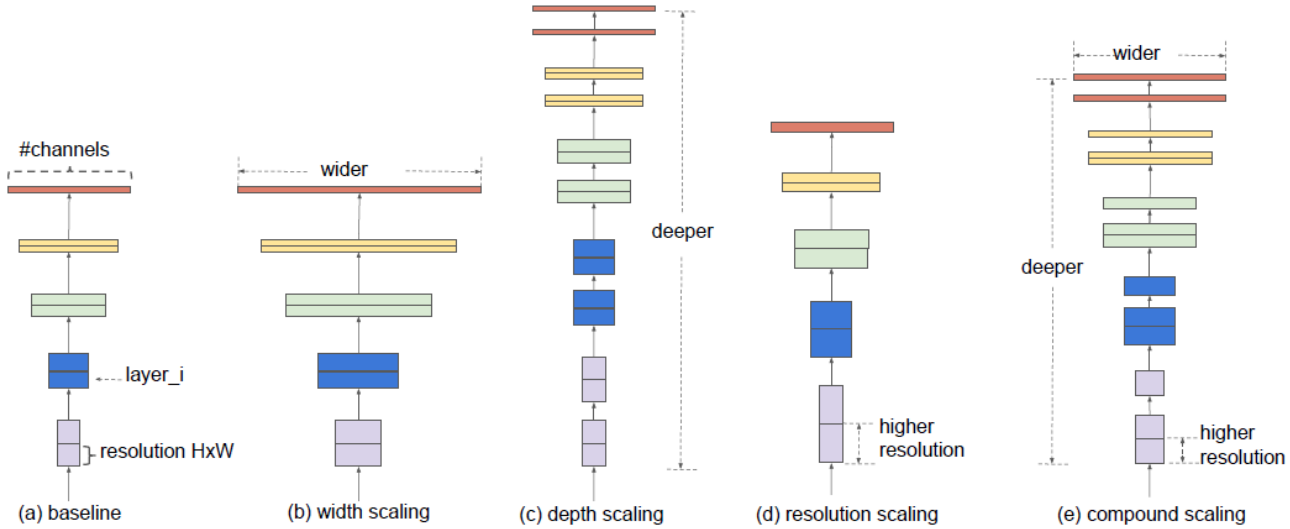
Yang et al. NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications
He et al. AMC: AutoML for Model Compression and Acceleration on Mobile Devices
Liu et al. MetaPruning: Meta Learning for Automatic Neural Network Channel Pruning

A General Overview for AutoML in Deep Learning

- Handcraft + NAS
  - Human-expert guided search (IRLAS)
  - Boosting existing handcraft models (EfficientNet, MobileNet v3)



- Points
  - Very competitive performance
  - Efficient
  - Drawback: Search space may be restricted

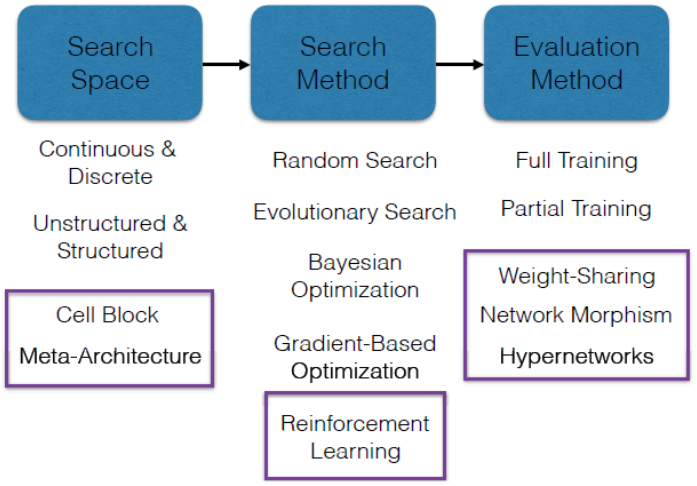


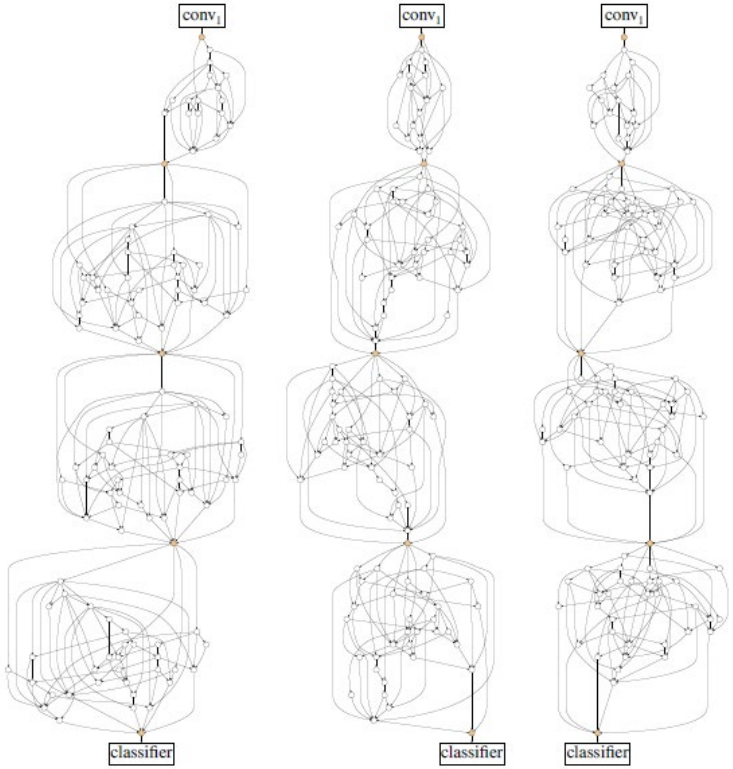
Howard et al. Searching for MobileNetV3
Tan et al. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks
Guo et al. IRLAS: Inverse Reinforcement Learning for Architecture Search

A General Overview for AutoML in Deep Learning

- Examing the Effectiveness of NAS
  - Random search
  - Random wire network

❖ Points
  - Reproducibility
  - Search algorithm or search space?
  - Baselines





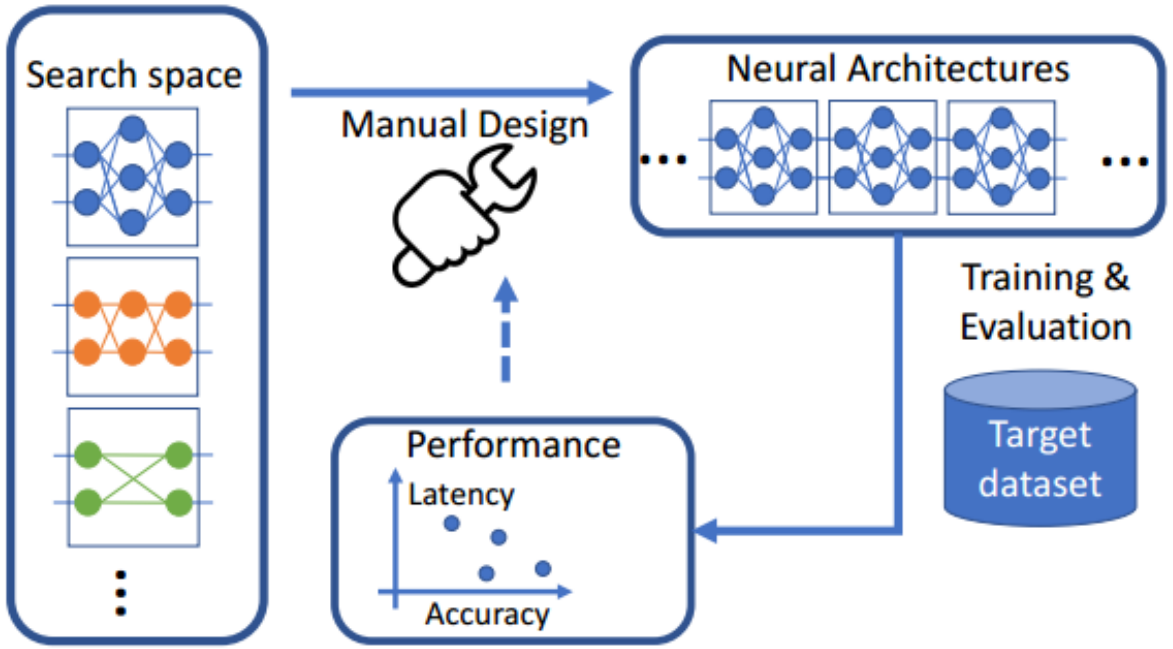
Li et al. Random Search and Reproducibility for Neural Architecture Search
Xie et al. Exploring Randomly Wired Neural Networks for Image Recognition

Outline

- Introduction
- Neural Architecture Search (NAS)
- Hyperparameter Optimization (HPO)
- Application: Hyperparameter Optimization for Massive Network Embedding

Motivation

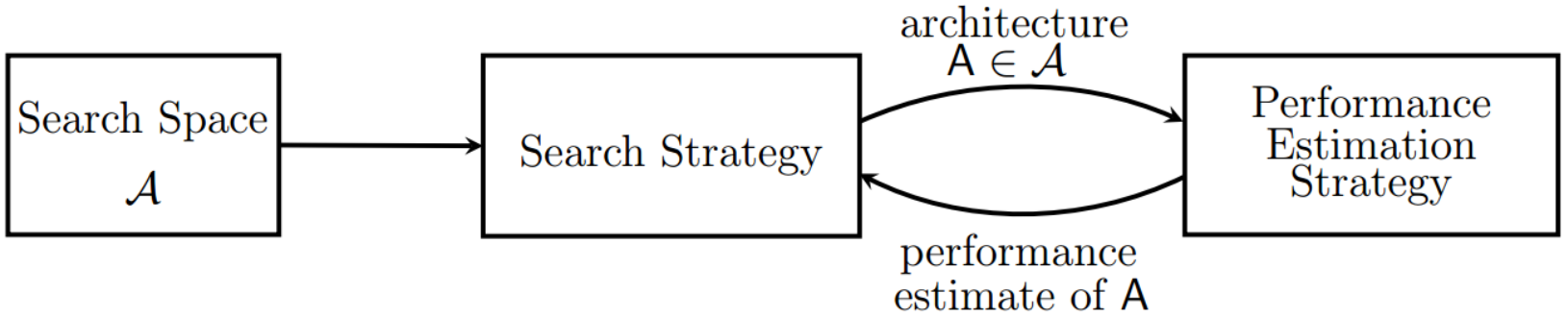
- Manual Network design
  - Intractable design space
  - Nontransferable optimality
    - Different input size, device…
- Neural Network Search
  - Fast iteration
  - Transferable optimality



Problem

Neural Architecture Search

- Search Space
  - Defines which architectures can be represented in principle
- Search Strategy
  - Details how to explore the search space
- Performance Estimation Strategy
  - Estimates the performance achieved on unseen data by a specific architecture



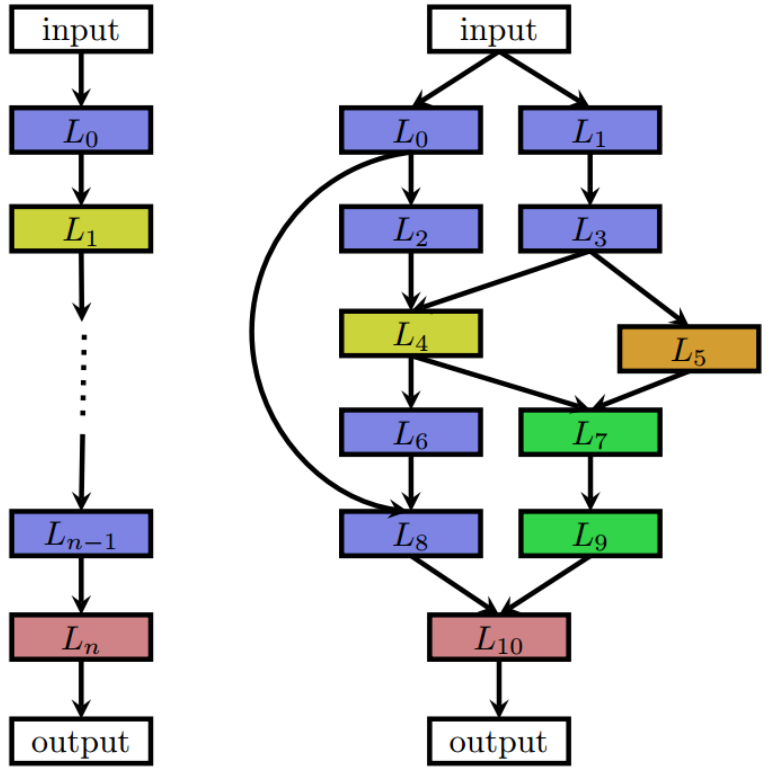
Survey on Automated Machine Learning, by Z¨oller and Huber

Search Space

Search Space

Chain-structured neural networks

- Maximum number of layers
- Type of operation in each layer
- Hyperparameters associated with the operation
  - Num of filters, kernel size and strides
- Baker et al., 2017a
  - Various operations with different hyperparameter setting



Search Space

Multi-branch networks

- Modern design elements (ResNet, DenseNet…)

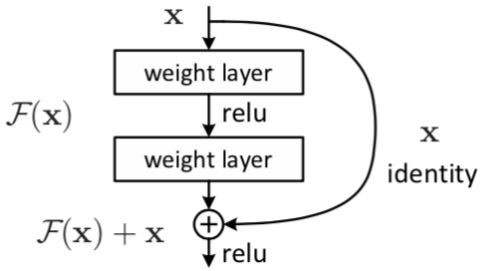
-- input of layer i is a function $g_i(L_{i-1}^{out}, \dots, L_0^{out})$ combining previous outputs

Special case:

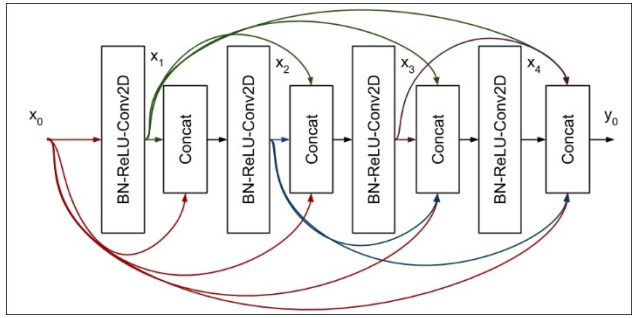
Chain-Structured Networks L_{i-1}^{out}

Residual Networks $L_{i-1}^{out} + L_j^{out}, j < i$

DenseNets $concat(L_{i-1}^{out}, \dots, L_0^{out})$



Residual



Dense

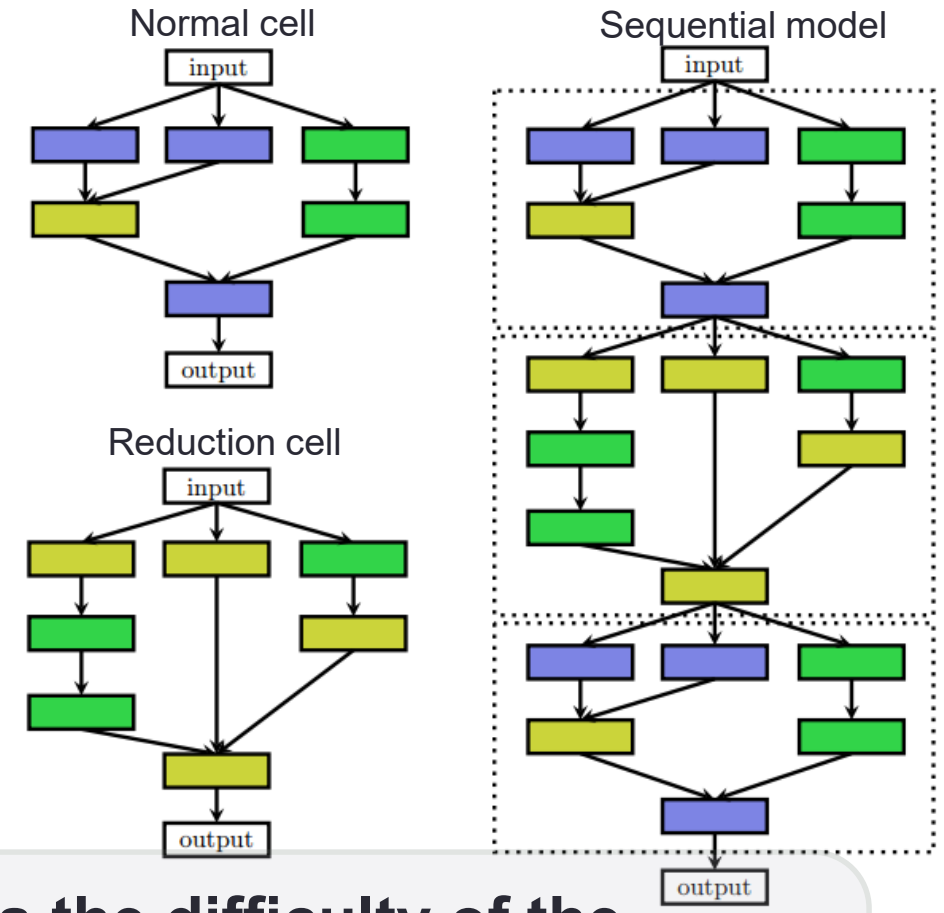
- Zoph et al., 2018
  - Incorporate modern design elements
  - Relax the search space by permitting arbitrary skip connections

Search Space

Cell-based search space

- Two major advantages
  - Smaller search space
  - Better transfer ability on different datasets

- Zoph et al. (2018) build a sequential model from cells
  - Normal cell: preserves the dimension of input
  - Reduction cell: reduces the dimension of input



The choice of the search space largely determines the difficulty of the optimization problem

---- non-continuous

---- relatively high-dimensional

Search Strategy

Search Strategy

- The search strategy details how to explore the search space

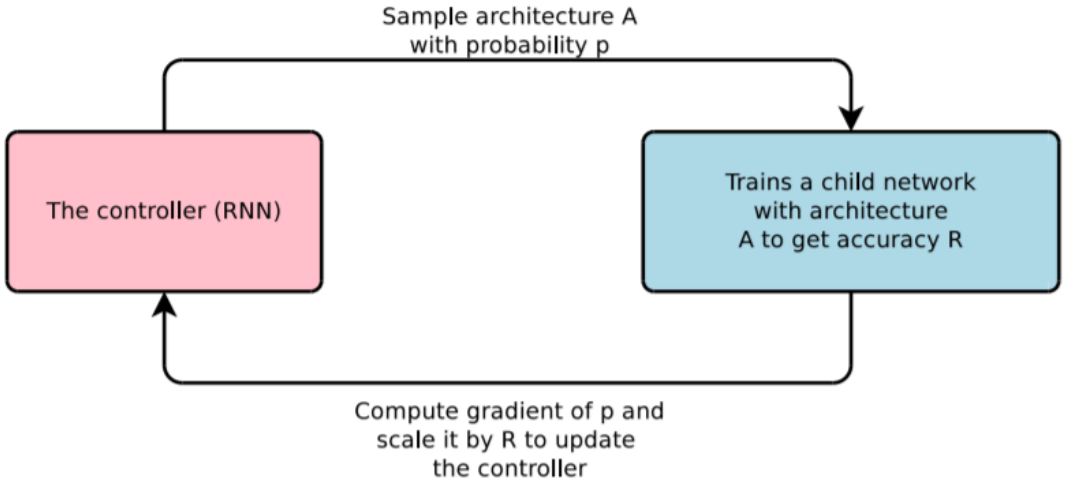
- For example, **Reinforcement learning:**
  - action: the generation of a neural architecture
  - reward: an estimate of the performance of the trained architecture on unseen data

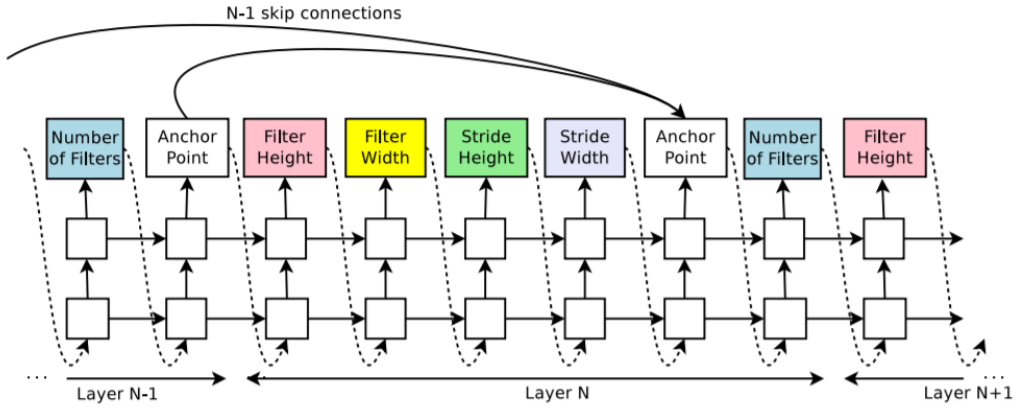
  - how they represent the agent's policy and how they optimize it

Search Strategy

- Zoph and Le (2017)
  - policy gradient
  - Proximal Policy Optimization(Zoph et al, 2018).
- Baker et al (2017a)
  - Q-learning

- Sequential decision processes
  - reward is obtained only after the final action
  - a stateless multi-armed bandit problem

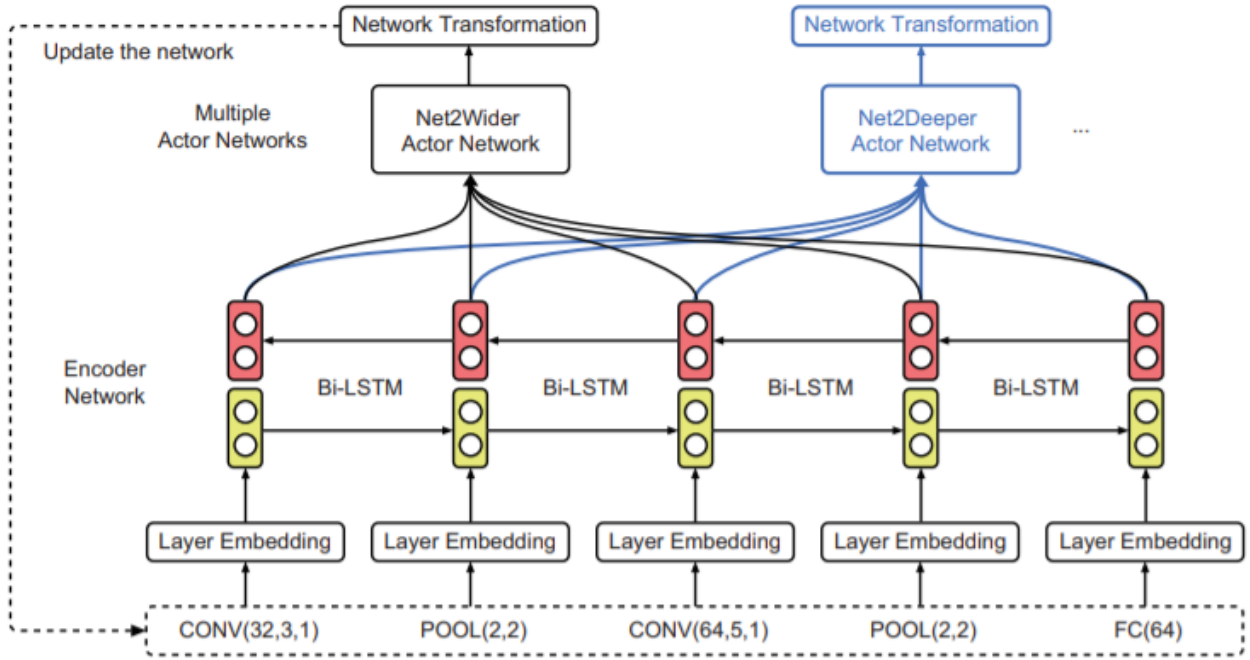




Zoph and Le (2017)

Search Strategy

- Cai et al (2018a)
  - Sequential decision process
  - Reward：an estimate of the architecture's performance
  - Action：an application of function-preserving mutations, dubbed network morphisms



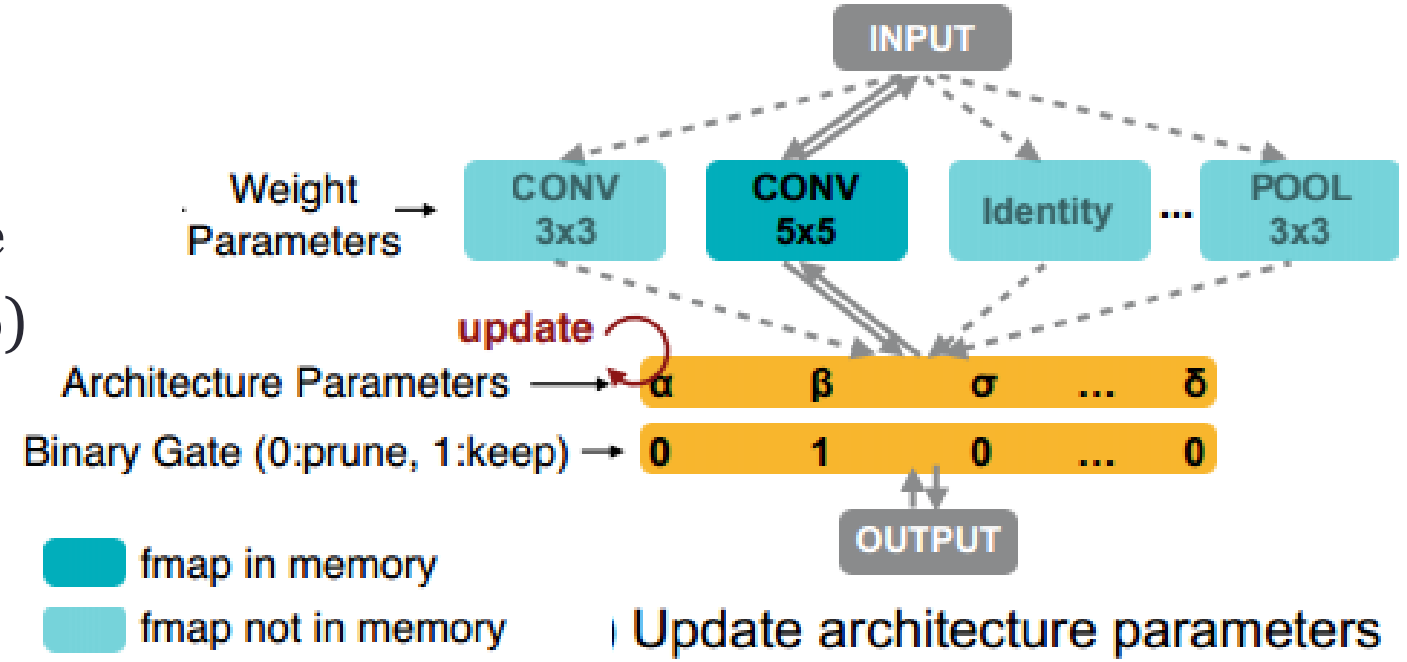
Search Strategy

- Reinforcement learning-- summary
  - Action=>Generation of neural architecture
    Action space=>Search space
    Reward=>Estimate of performance
  - Zoph and Le, 2017
    - Policy gradient
  - Zoph et al., 2018
    - Proximal Policy Optimization
  - Cai et al., 2018a
    - Sequential decision process
    - Bi-directional LSTM

Search Strategy

Gradient-based methods

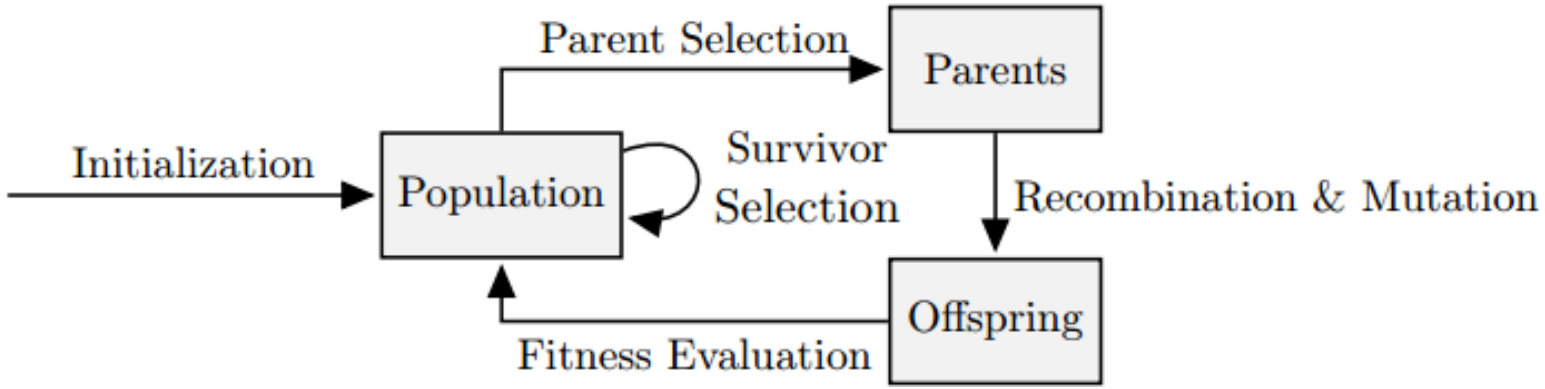
- Darts, Liu et al. (2018c)
  - Continuous relaxation of search space by combining operations
  - $y = \sum_{i=1}^{m} \lambda_i o_i(x), \lambda \geq 0, \sum_{i=1}^{m} \lambda_i = 1$
  - Optimize both weight and architecture
- Wu et al. (2018), Xie et al. (2019b)
  - Represent the architecture by using stochastic variable to model concrete distribution
- ProxylessNAS, Han et al. (2019)
  - Model the structural parameters with binary gates
  - Use learned probabililty to choose a part of the path



Search Strategy

Neural Evolutionary Strategy

- Sample parent
  - Real et al (2017), Real et al (2018), and Liu et al (2018a)
    - Tournament selection
- Update populations
  - Real et al (2017)
    - Remove the worst individual
  - Real et al (2018)
    - Remove the oldest individual
  - Liu et al (2018a)
    - Do not remove individuals



- Generate offspring
  - Most approaches initialize child networks randomly
  - Real et al (2017), Guo et al (2019)
    - Inherit all parameters of its parent

Search Strategy

Random search

- Xie et al. (2019)
  - Use a graph generator to generate random graph structures
- Sciuto et al. (2019)
  - Random search finds better RNN cells than any other optimizer
- Li and Talwalkar (2019)
  - Random search finds architectures that perform at least as well as the ones obtained from established optimizers for CNNs

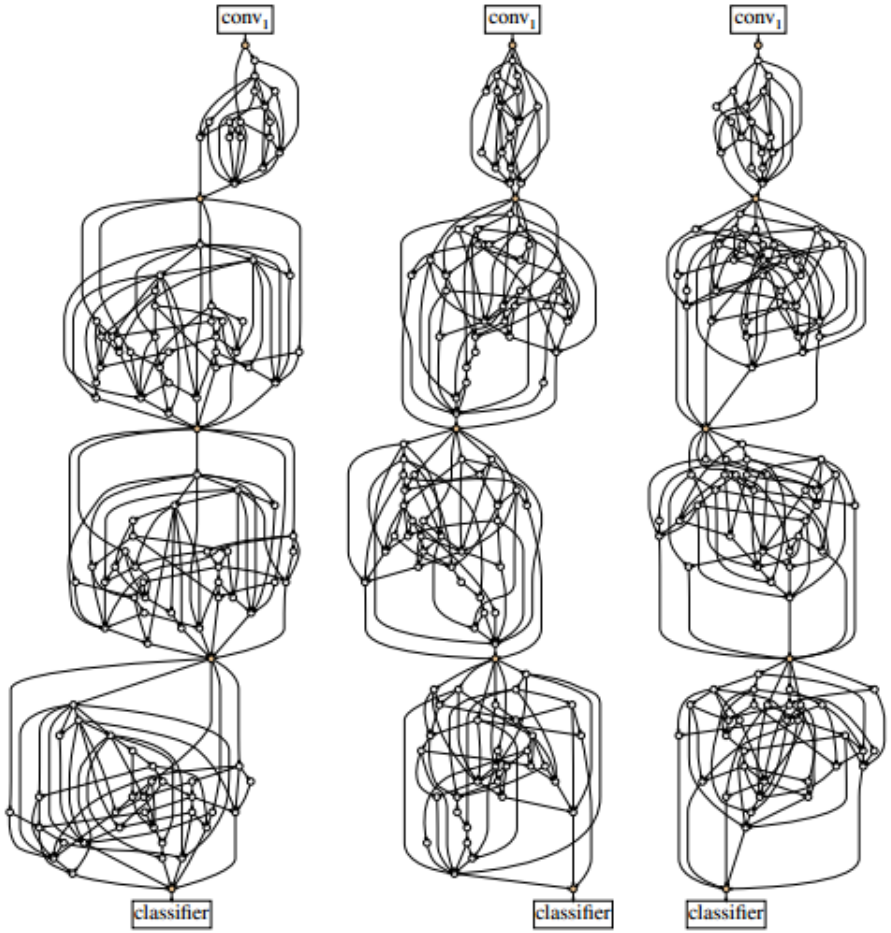


Figure 1. **Randomly wired neural networks** generated by the classical Watts-Strogatz (WS) [50] model: these three instances of random networks achieve (left-to-right) 79.1%, 79.1%, 79.0% classification accuracy on ImageNet under a similar computational budget to ResNet-50, which has 77.1% accuracy.

Search Strategy

Compare: RL, evolution, and random search (RS)

- Real et al (2018)
  - RL and evolution perform equally well, better than RS with a rather small margin
  - Evolution having better anytime performance and finding smaller models

Performance Estimation Strategy

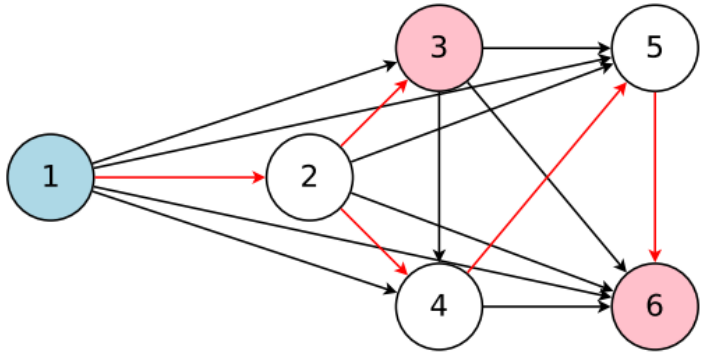
Performance Estimation Strategy

- Train on training data & Evaluate on validation data
  - Thousands of GPU days
  - Zoph and Le, 2017; Real et al., 2017; Real et al., 2018
- Lower fidelities with less computation cost
  - Zoph et al., 2018; Zela et al., 2018: Shorter training time
  - Klein et al., 2017b: Training on a subset data
  - Chrabaszcz et al., 2017: Lower-resolution images
- Challenge: good predictions in a relatively large search space need to be made based on relatively few evaluations.

Performance Estimation Strategy

One-Shot Architecture Search

- Only train the supernetwork which includes different subnetwork
- ENAS(Pham et al, 2018)
  - Learns an RNN controller that samples architectures
  - Trains the one-shot model based on approximate gradients obtained through REINFORCE



- DARTS (Liu et al, 2018c)
  - Optimizes all weights of the one-shot model jointly with a continuous relaxation of the search space
  - Placing a mixture of candidate operations on each edge of the one-shot model
- Bender et al (2018)
  - only train the one-shot model once
  - deactivating parts of this model stochastically during training using path dropout

Performance Estimation Strategy

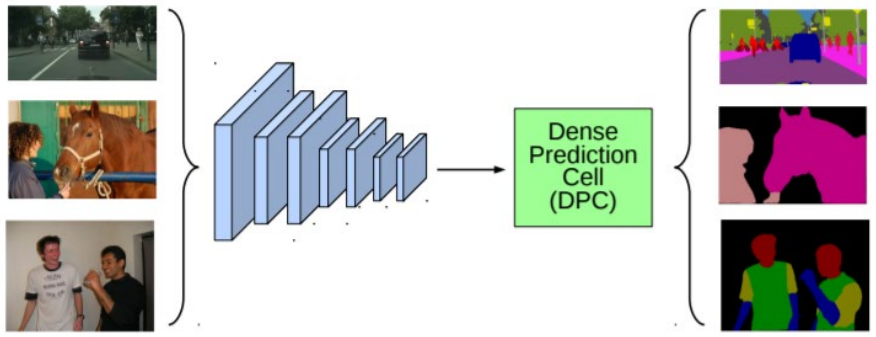
Limits of One-Shot Architecture Search

- The supergraph defined a priori restricts the search space to its subgraphs

- Relatively small supergraphs(GPU memory)
  - typically used in combination with cell-based search spaces

Neural Architecture Search on Multimedia

- Most NAS focus on Image Classification(CNN) and NLP(RNN).
- The multimedia problem may be more complicated
  - Different search space and different performance estimation strategy
- For example, scene parsing.
  - The key problem: the architecture search space is domain-related.



Scene Parsing

Two key components:

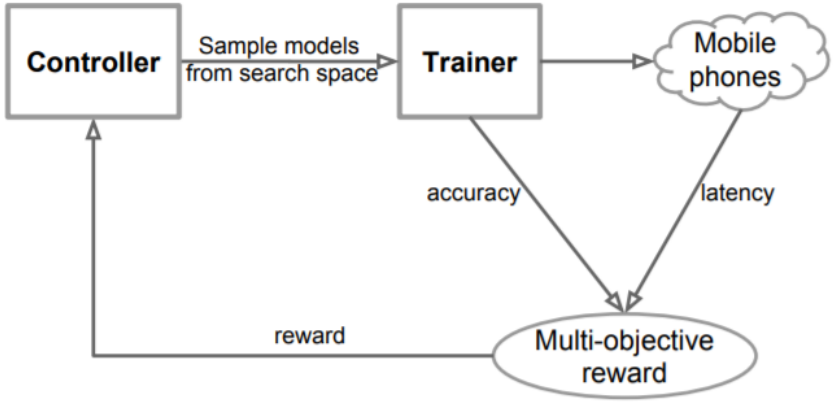
- Architecture search space： Dense Prediction Cell
  - recursive search space
- The design of the proxy task: high resolution imagery
  - employing a smaller network backbone
  - caching the feature maps and directly building a single DPC

Chen, Liang-Chieh, et al. "Searching for efficient multi-scale architectures for dense image prediction." *NIPS*. 2018.

Neural Architecture Search on Multimedia

Resource-constrained neural architecture search, such as mobile.

- Mnasnet:
  - Change into a multi-objective NAS problem
    - Different performance estimation strategy
    - Tarde off accuracy and latency
  - Factorized hierarchical search space
    - Different search space
    - Balance between flexibility and search space size

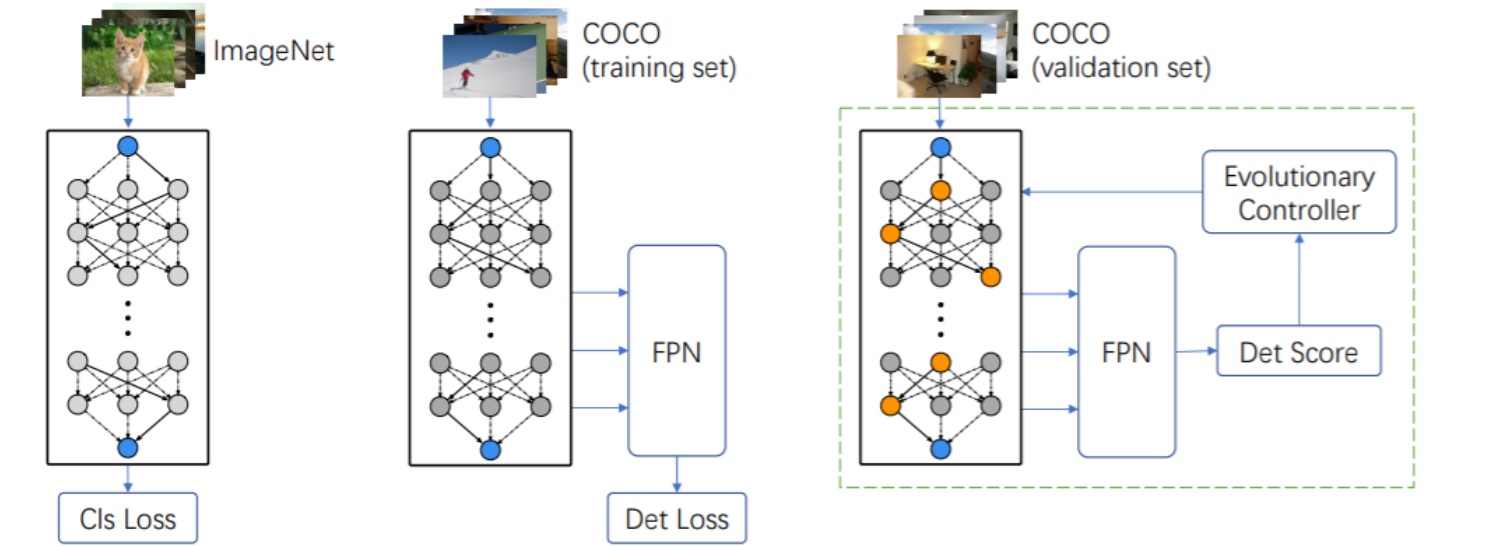


Tan, Mingxing, et al. "Mnasnet: Platform-aware neural architecture search for mobile." *CVPR*. 2019.

Neural Architecture Search on Multimedia

Object Detection

- Chen et al. 2019
  - Use NAS for the design of better backbones for object detection

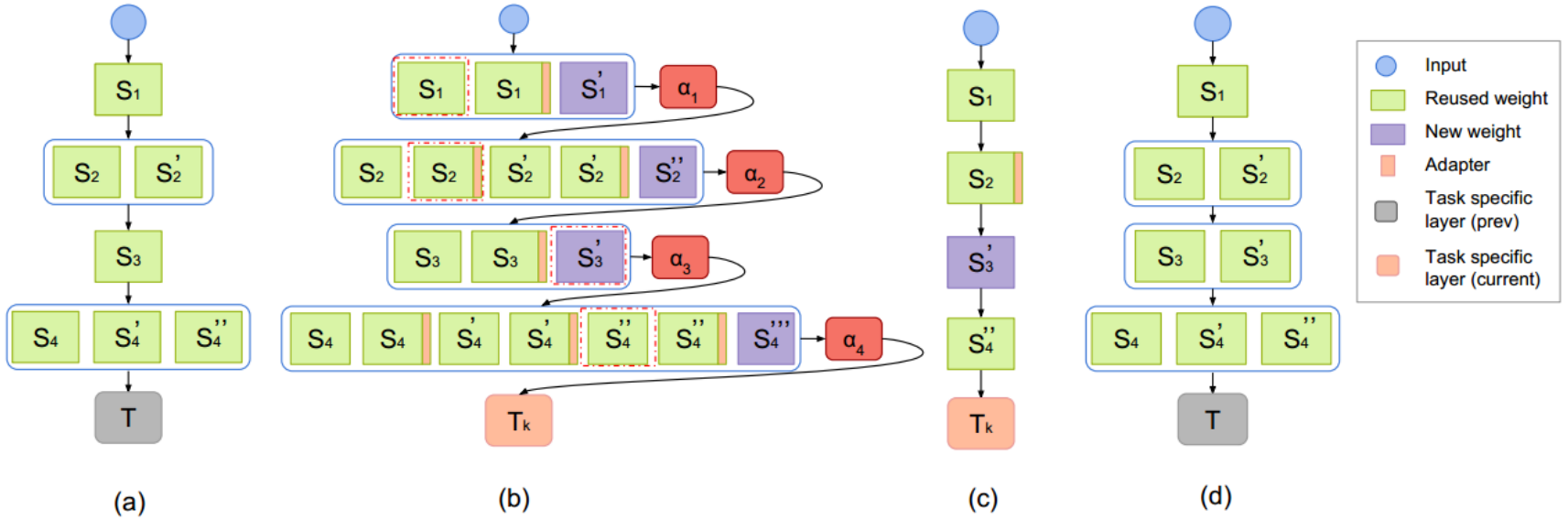


Step1: Supernet pre-training Step2: Supernet fine-tuning Step3: Evolutionary search on the trained supernet

Neural Architecture Search on Multimedia

Continual Learning

- Li et al. 2019
  - Use NAS to grow the network architecture with new tasks



Neural Architecture Search on Multimedia

Multimodal Fusion

- Pérez-Rúa et al. (2019)
  - Propose a generic search space that spans lots of fusion architectures

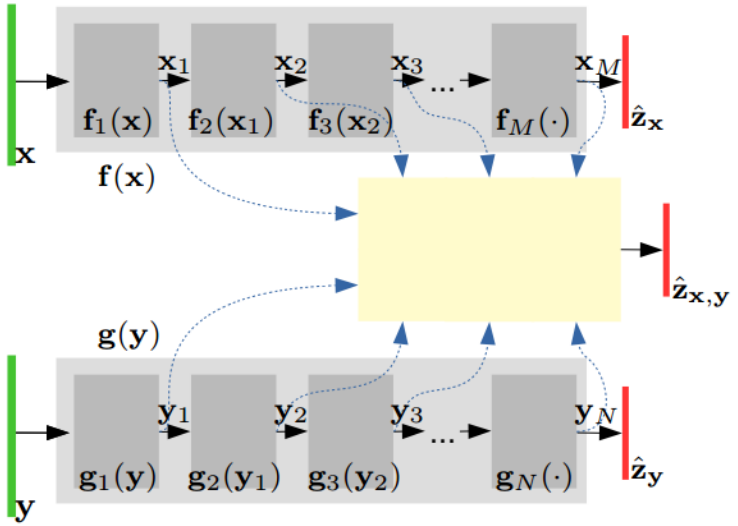


Figure 1. **General structure of a bi-modal fusion network**. Top: A neural network with several hidden layers (grey boxes) with input $\mathbf{x}$, and output $\hat{\mathbf{z}}_\mathbf{x}$. Bottom: A second network with input $\mathbf{y}$, and output $\hat{\mathbf{z}}_\mathbf{y}$. In this work we focus on finding efficient fusion schemes (yellow box and dotted lines).

Trends and Challenges

- Trends
  - Efficient & high-performance algorithm
  - Flexible search space
  - Device-aware optimization
  - Multi-task / Multi-target search
- Challenges
  - Trade-offs between efficiency, performance and flexibility
  - Search space matters
  - Fair benchmarks
  - Pipeline search

Summary

- Various Tasks
  - Object Detection
  - Semantic Segmentation
  - Super-resolution
  - Face Recognition

  …
- Not only Architecture, search for everything!
  - Activation function
  - Loss function
  - Data augmentation
  - Backpropagation

  …

Liu et al. Auto-DeepLab: Hierarchical Neural Architecture Search for Semantic Image Segmentation
Chu et al. Fast, Accurate and Lightweight Super-Resolution with Neural Architecture Search
Ramachandra et al. Searching for Activation Functions
Alber et al. Backprop Evolution

Outline

- Introduction
- Neural Architecture Search (NAS)
- Hyperparameter Optimization (HPO)
- Application: Hyperparameter Optimization for Massive Network Embedding

Background

- The choice of the hyperparameters significantly affects the effectiveness of the machine learning system.

- Hyperparameter optimization (HPO) aims to automatically select the optimal configurations of hyperparameters.

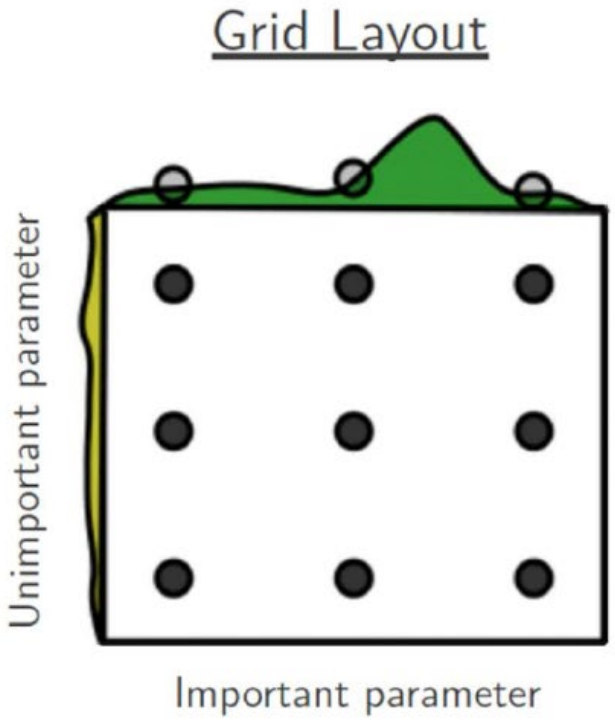
Definition

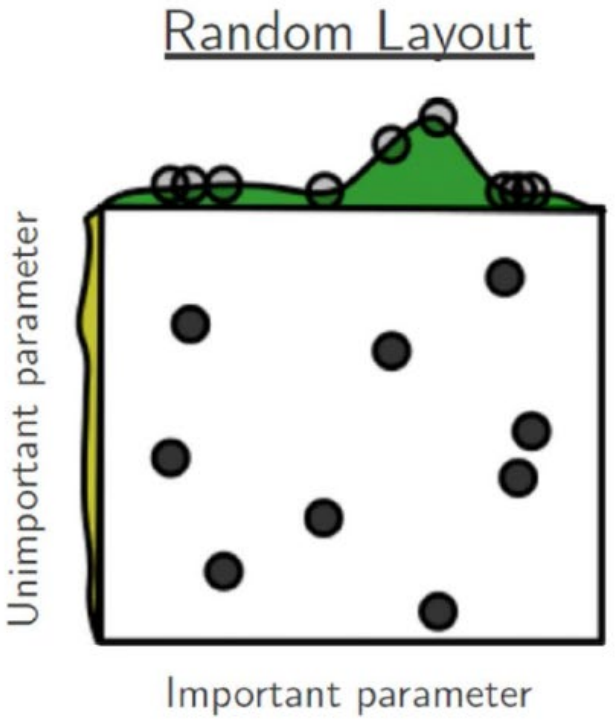
- Suppose an algorithm A is controlled by some hyperparameters chosen from a given domain Λ
- For any configuration $\lambda \in \Lambda$, its utility can be measured by the validation loss of A using λ, i.e. $\mathcal{L}_{val}(A, \lambda)$
- HPO searches for a configuration λ^* that minimizes the validation loss

$$\lambda^* \in \text{argmin}_{\lambda \in \Lambda} \mathcal{L}_{val}(A, \lambda)$$

Naïve Methods

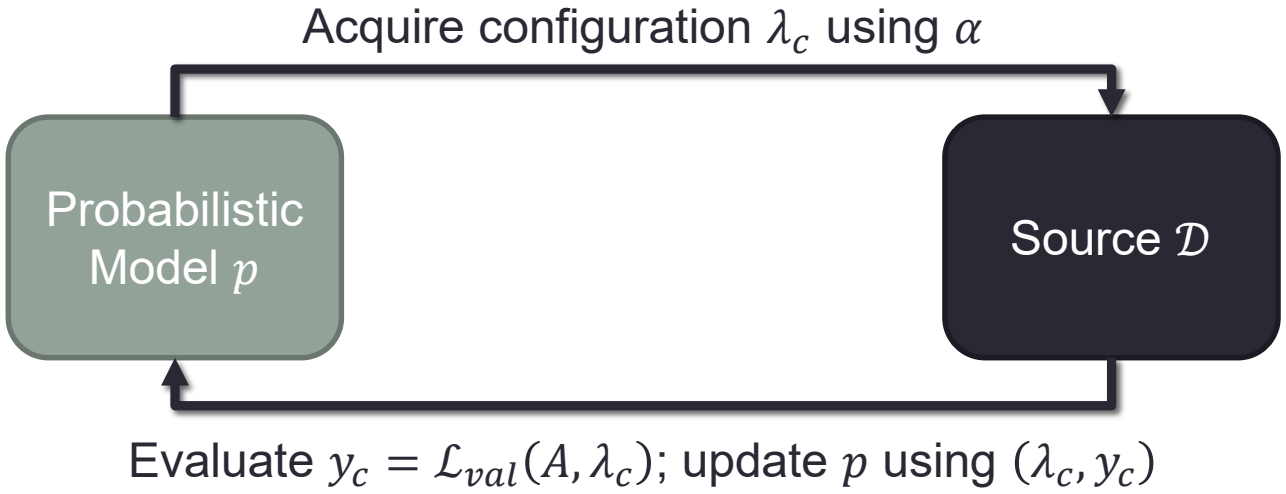
- Grid Search
- Random Search





Bayesian Optimization: Basics

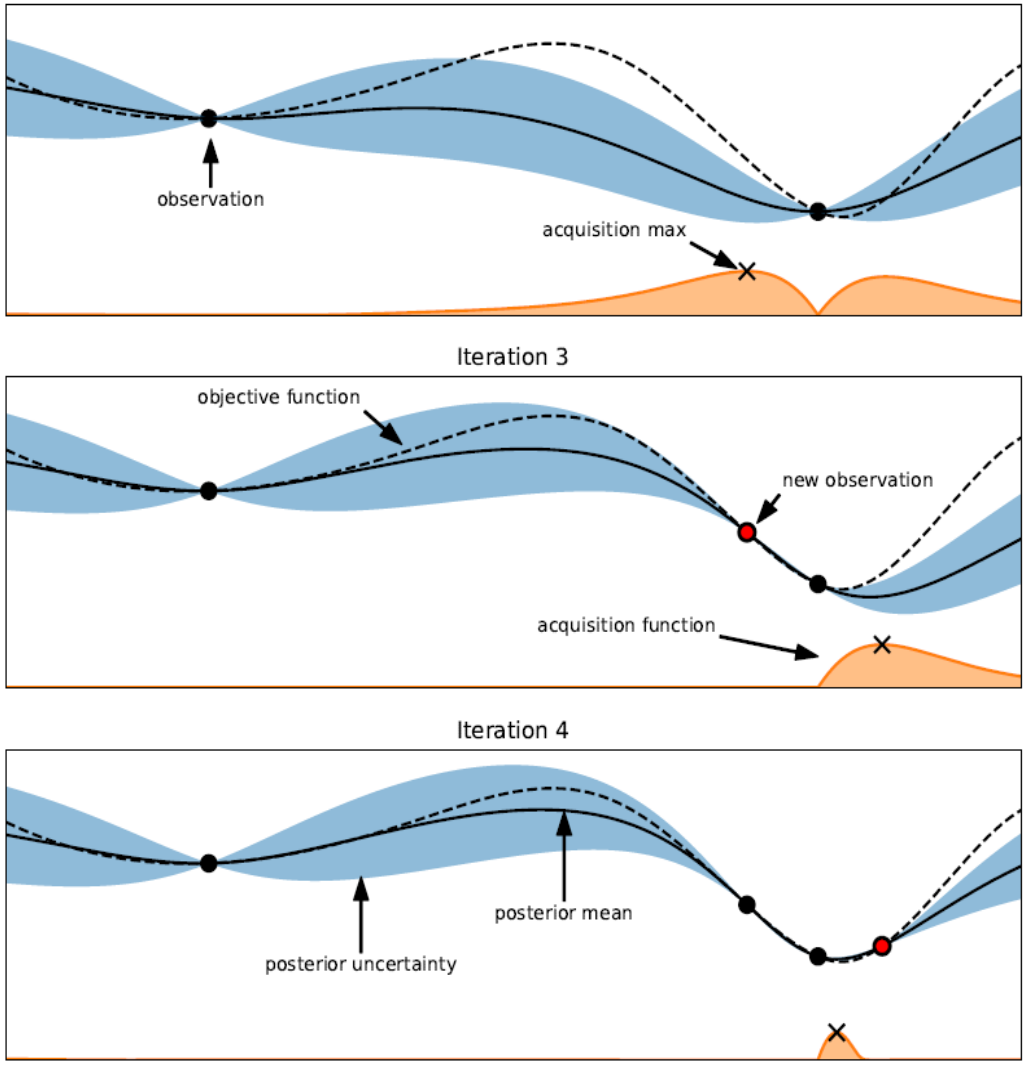
- A probabilistic model $p(y \mid \lambda)$ fits the evaluations $\left(\lambda, \mathcal{L}_{val}(A, \lambda)\right)$
- $p(y \mid \lambda)$ gets refined with more configurations being evaluated
- The candidate configurations are acquired by an acquisition function $\alpha(\lambda; p)$ that trades off exploration and exploitation



Bayesian Optimization: Procedure

- Prescribe a prior belief $p(y \mid \lambda)$

- Do sequentially:
  - Select a candidate λ_c using $\alpha(\lambda; p)$
  - Run A with λ_c, evaluate $y_c = \mathcal{L}_{val}(A, \lambda_c)$
  - Update posterior $p(y \mid \lambda)$ using (λ_c, y_c)



Bayesian Optimization: Probabilistic Model

- Gaussian process (GP)
  - Efficient posterior calculation using conjugate properties
  - Unsuitable for complex hyperparameter space
- Random forest
  - Compatible with hierarchical hyperparameters
- Tree Parzen estimator (TPE)
  - Efficient computation and robust
- Bayesian Neural Network
  - Strong performance in low-dimensional space
  - Ineffective in high-dimensional and hierarchical domain

Bayesian Optimization: Acquisition Function

- Trade off exploration and exploitation

- Upper confidence bound (UCB)

$$\alpha_{UCB}(\lambda; p) = \beta \cdot \text{std}[y \mid y \sim p(y \mid \lambda)] - \text{mean}[y \mid y \sim p(y \mid \lambda)]$$

- Thompson sampling

$$\alpha_{TS}(\lambda; p) = -f(\lambda), \qquad f(\lambda) \sim p(y \mid \lambda)$$

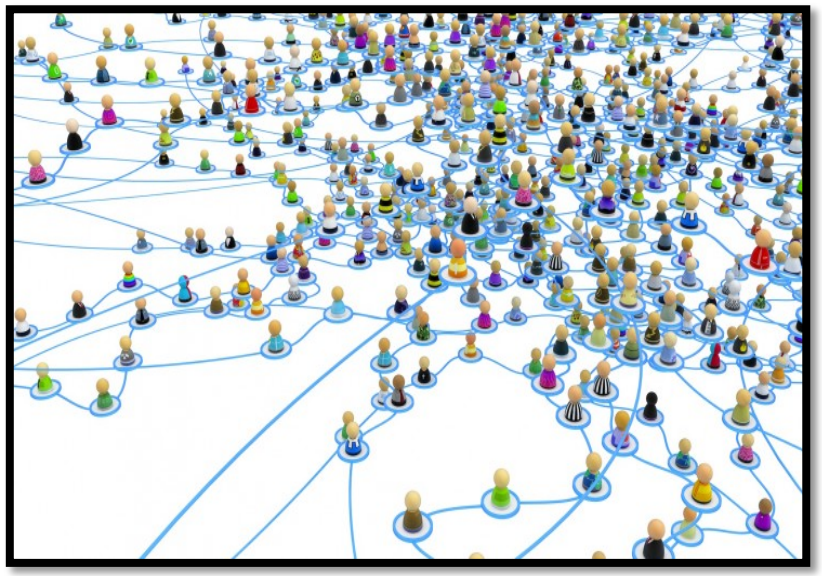
- Expected improvement (EI)

$$\alpha_{EI}(\lambda; p) = \mathbb{E}[\max(\tau - y, 0) \mid y \sim p(y \mid \lambda)]$$

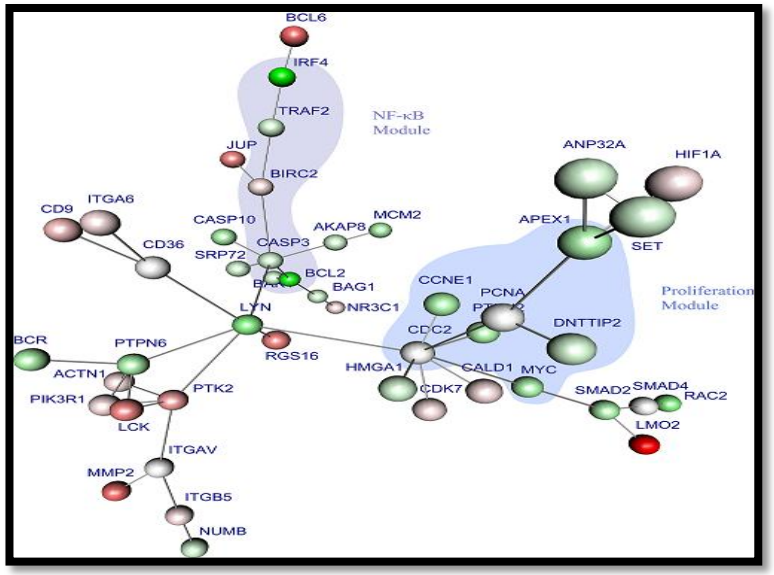
Outline

- Introduction
- Neural Architecture Search (NAS)
- Hyperparameter Optimization (HPO)
- Application: Hyperparameter Optimization for Massive Network Embedding

Network Analytics



Social Networks

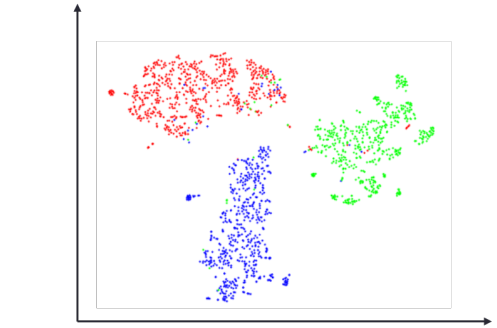


Biology Networks

Networks are widely used to represent the rich pairwise relationships of data objects

Network Embedding

Origin network

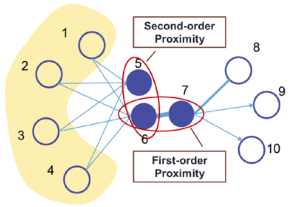


- in vector space

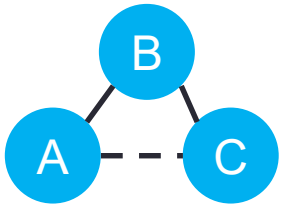
- Link Prediction
- Community Detection
- Node Classification
- Network Distance
- Node Importance
- …

Networks embedding aims to learn a low-dimensional representation for each node

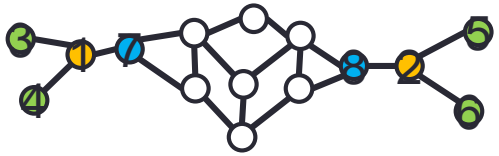
Existing Embedding Methods



High-order



Transitivity



Global position

Various network properties

- Link Prediction
- Community Detection
- Node Classification
- Network Distance
- Node Importance
- …

Various applications

- Leading to a large number of hyperparameters
  - E.g. Deepwalk: number of walks, walk length, window size
- Must be carefully tuned

AutoML

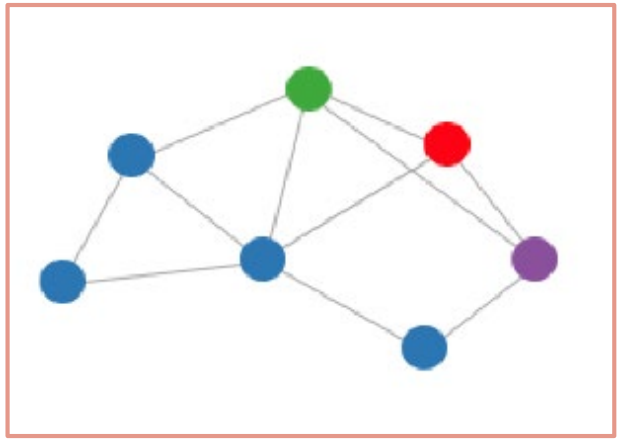
AutoML

- Ease the adoption of machine learning and reduce the reliance on human experts
  - e.g., hyperparameter optimization
- Network data remains largely unexplored
- Large scale issue:
  - Complexity of Network Embedding is usually at least $O(E)$
    - E is the number of edges (can be 10 billion)
  - Total complexity: $O(ET)$, T is the times searching for optimal hyperparameter

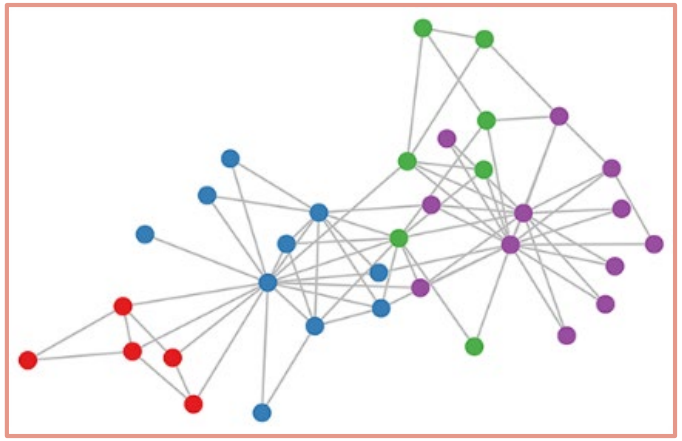
How to incorporate AutoML into massive network embedding efficiently? (reduce E and T)

Incorporating AutoML into NE

- A straightforward way: configuration selection on sampled sub-networks



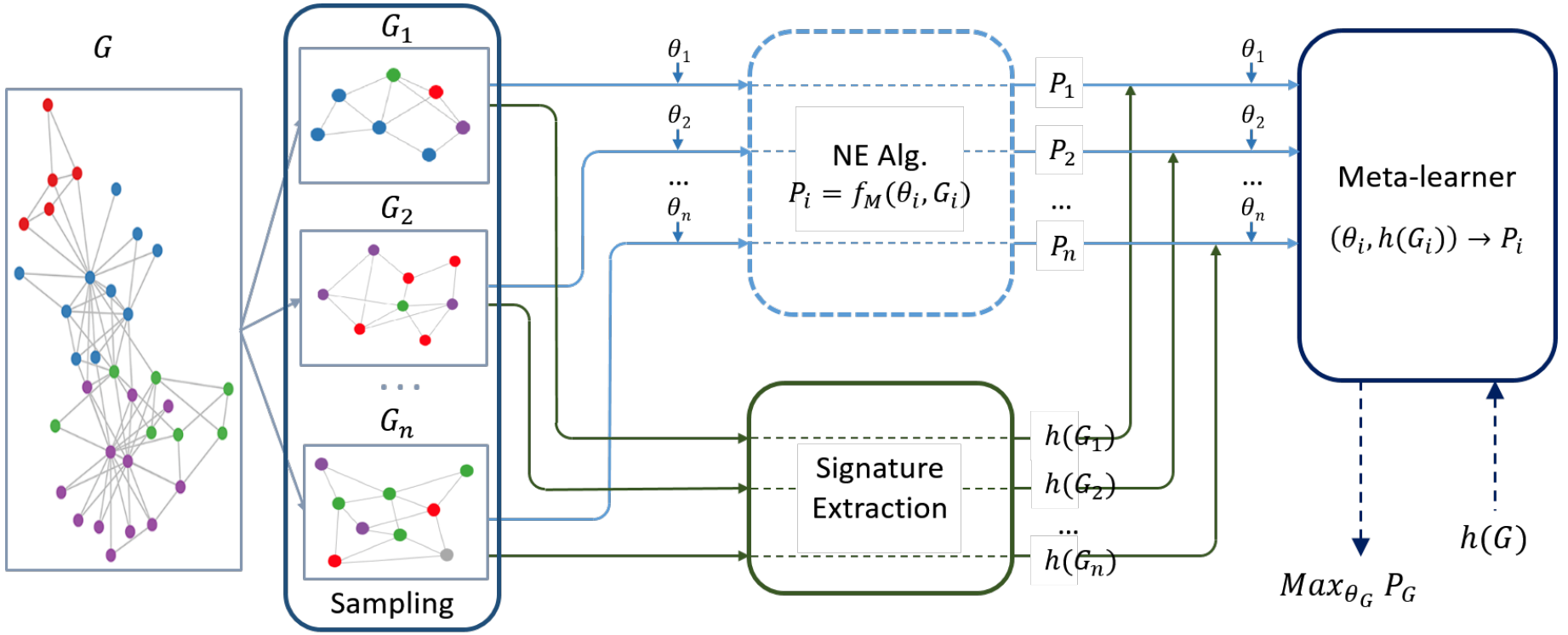
Sampled sub-network
Optimal configuration θ^*



Origin massive network
using θ^*

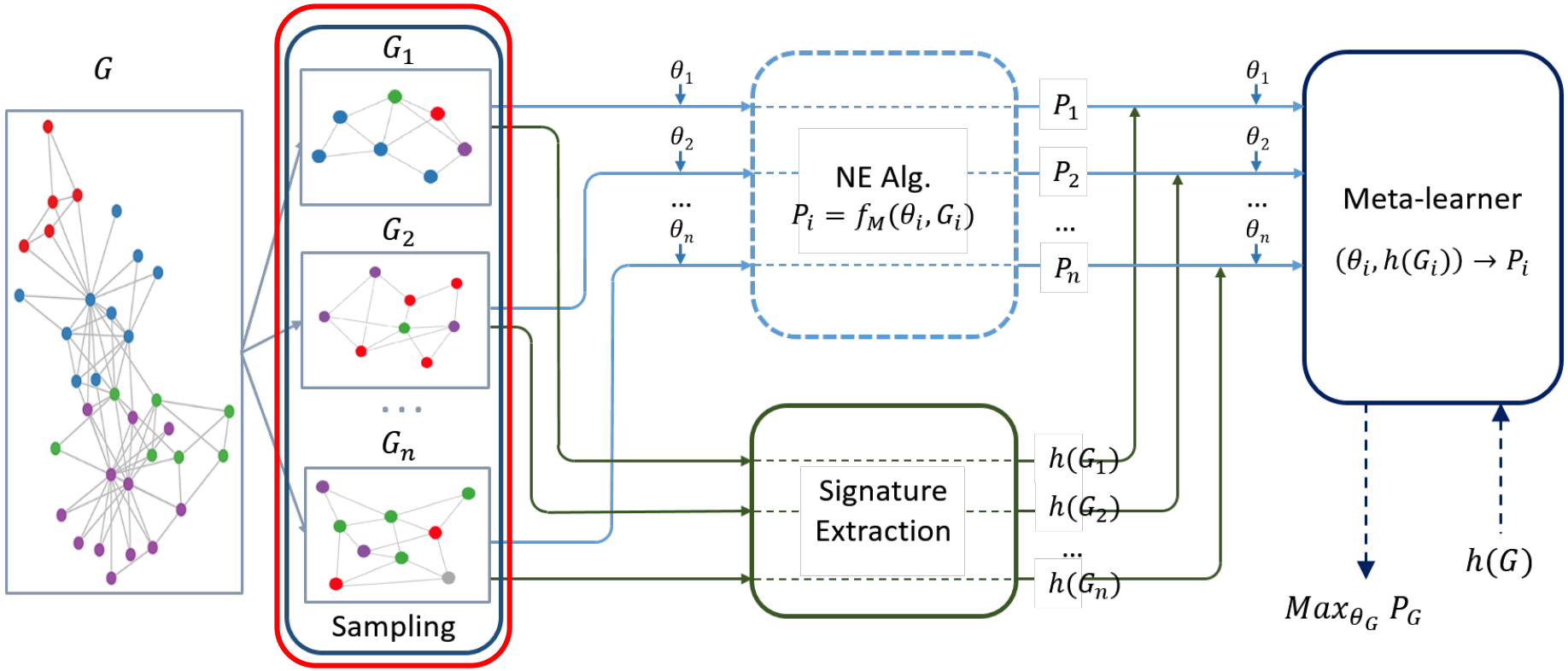
- **Transferability**
  - $\theta \neq$ optimal configuration on origin network
- **Heterogeneity**
  - several highly heterogeneous components => carefully designed sampling

AutoNE



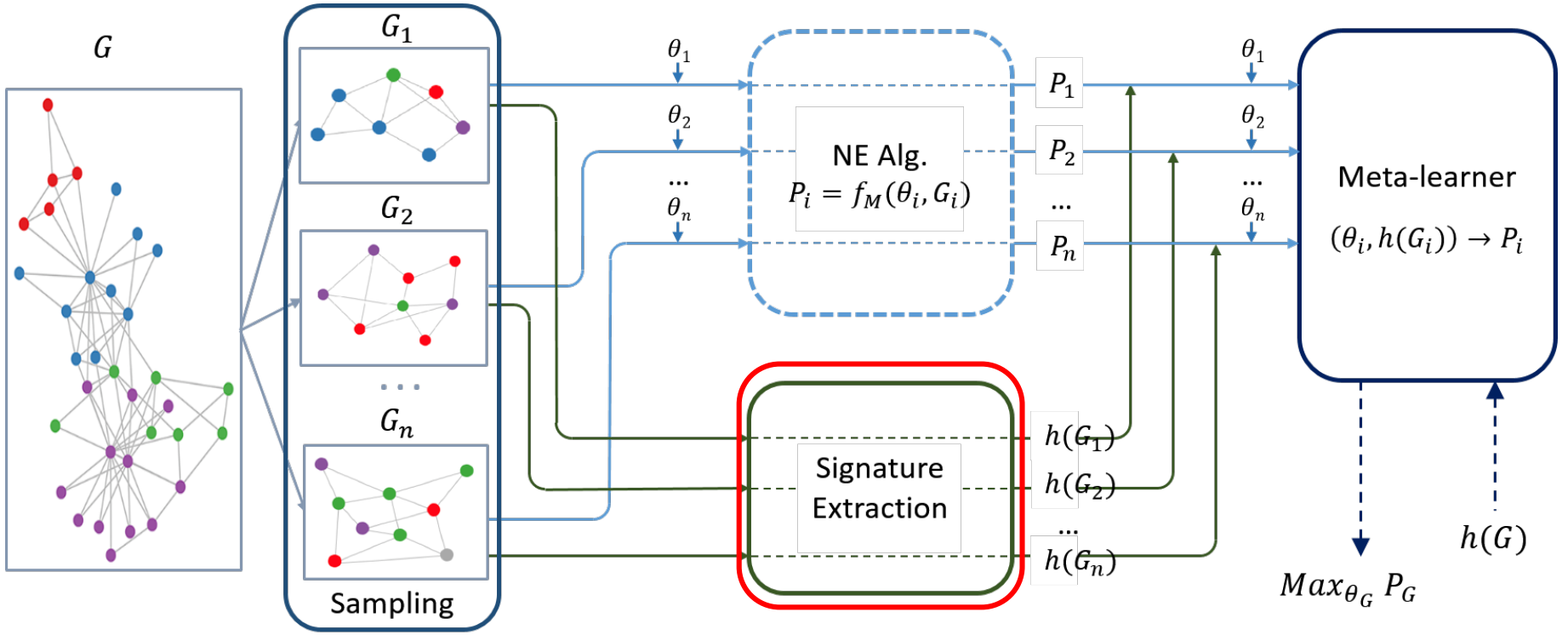
Transfer the knowledge about optimal hyperparameters from the sub-networks to the original massive network

AutoNE



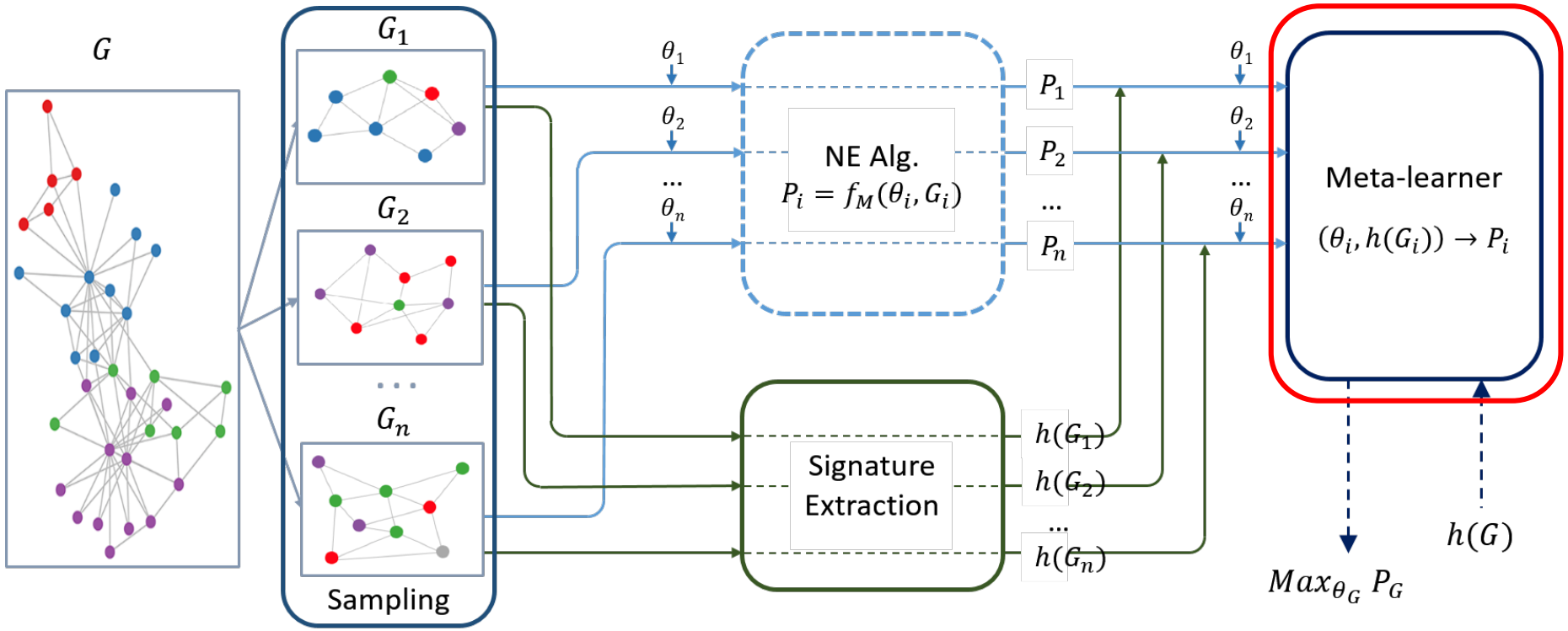
Transfer the knowledge about optimal hyperparameters from the sub-networks to the original massive network

AutoNE



Transfer the knowledge about optimal hyperparameters from the sub-networks to the original massive network

AutoNE



Transfer the knowledge about optimal hyperparameters from the sub-networks to the original massive network

Sampling Module

- **Goal**: Sample a series of representative sub-networks that share similar properties with the original large-scale network
- **Heterogeneity** issue: preserve diversity of the origin network

- Origin network $G = (V, E)$
- Random walk w $= (v_1 \rightarrow v_2 \rightarrow \ \ldots \rightarrow v_n)$
- Sub-graph G_s based random walk w:
  - Nodes: $\mathrm{V}' = \{\mathrm{v}_1, v_2, \ldots, v_n\}$
  - Edges: $\mathrm{E}' = \{(v_i, v_j) | \, i, j \in [1, n], \ (v_i, v_j) \in E\}$

- The starting points v_1
  - Supervised: several nodes with different labels
  - Unsupervised: from the different communities， e.g., a greedy algorithm that maximizes modularity

Signature Extraction Module

- The signature of a network is a vector descriptor that encodes the various properties of the whole network

- Graph signature $\mathrm{h}(G)$
  - $Similarity(G_1, G_2) = Distance(h(G_1), \mathrm{h}(G_2))$

- **Challenge**: The signatures should be comprehensive enough
  - Based on spectral graph theory, a large number of network properties are decided by the spectrum of a network, e.g. the normalized cuts used by spectral clustering
  - We choose NetLSD [Tsitsulin et al. KDD18]
    - heat diffusion process on a network
    - $h_t(G) = tr(H_t) = tr(e^{-tL}) = \sum_j e^{-t\lambda_j}$

Meta-Learning Module

- Performance function $f_M(\theta, G) \rightarrow P$

- **Transferability** issue:
  - Assumption: Two similar network will has similar optimal hyperparameter on the same network embedding method

- Learning Performance function on small sampling networks, and predict on origin massive network

- We choose f_M as **Gaussian Process** like Bayesian optimization
- The log likelihood: (K is the kernel function)

$$\ln p(\mathbf{f} \mid \mathbf{X}) = -\frac{1}{2}\mathbf{f}^\top K(\mathbf{X}, \mathbf{X})^{-1}\mathbf{f} - \frac{1}{2}\ln\det(K(\mathbf{X}, \mathbf{X})) + constant.$$

Meta-Learning Module

- Given a new test point $x_* = (\theta_*, h(G_*))$ and the observed values f
  - The predicted performance f_* and f follow a joint normal distribution
- The posterior distribution $p(f_M(\theta_*, h(G_*) \mid x_*, f, X)$ is a normal distribution:

$$f_M(\theta_*, G_*) \mid \mathbf{x}_*, \mathbf{f}, \mathbf{X} \sim \mathcal{N}(\mu_*, \sigma_*^2),$$
$$\mu_* = K(\mathbf{x}_*, \mathbf{X})K(\mathbf{X}, \mathbf{X})^{-1}\mathbf{f},$$
$$\sigma_*^2 = K(\mathbf{x}_*, \mathbf{x}_*) - K(\mathbf{x}_*, \mathbf{X})K(\mathbf{X}, \mathbf{X})^{-1}K(\mathbf{X}, \mathbf{x}_*).$$

- Optimize θ_*: Upper confidence bound: $\arg\max_{\theta_*} \mu_* + \kappa\sigma_*$.
- Kernel function K:
  - Measure the similarity between two sets of hyperparameters and the similarity between two networks

$$k\left((\theta_1, G_1), (\theta_2, G_2)\right) = k_\theta\left(\theta_1, \theta_2\right) \cdot k_g\left(h(G_1), h(G_2)\right).$$

Kernel for hyperparameter

Kernel for graph

Experiment Setting --- datasets

Datasets	Node	Edge	Label	Feature (For GCN)
BlogCatalog	10,312	333,983	39	-
Wikipedia	4,777	184,812	40	-
Pubmed	19,717	44,338	3	500
TopCat	1,791,489	28,511,807	-	-

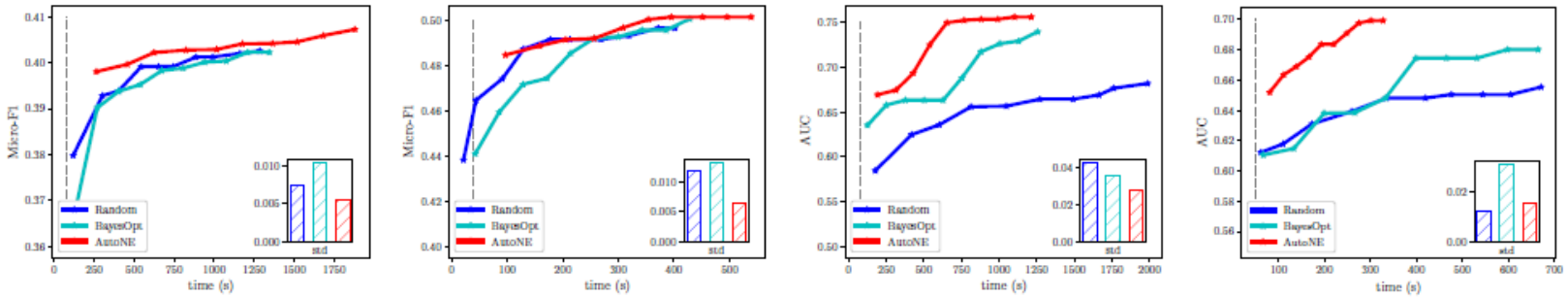
Experiment Setting --- NE

- Network Embedding method: Three classes
  - **Sampling-based NE: Deepwalk** [Perozzi, Bryan, et.al. KDD14]
    - Number of random walks
    - Length of each random walk
    - Windows size
  - **Factorization based NE: AROPE** [Zhang, Ziwei, et al. KDD18]
    - The weights of the different orders(Max 4)
  - **Deep neural network-based NE: GCN** [Thomas N. Kipf, et al. ICLR17]
    - Learning rate
    - Size of each hidden layer
    - Number of training epochs
    - The dropout rate
    - The weight decay

Experiment Setting --- Baselines and Tasks

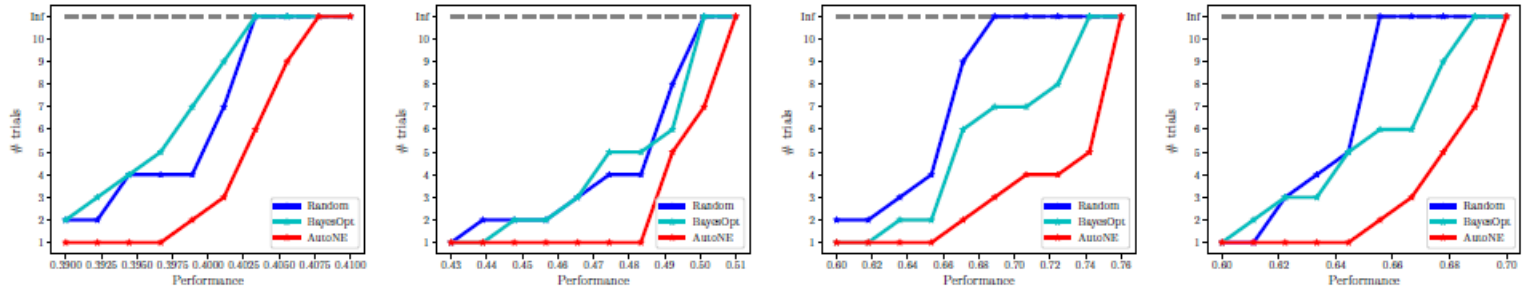
- Baselines
  - **Random search**
    - find the optimal solution as long as the time budget is large enough
    - Instead of grid search
      - explore larger configuration space more efficiently
      - find better solutions faster
  - **Bayesian optimization (BayesOpt)**
    - the most used method in AutoML framework
    - performs many trials on the original data
      - makes it inefficient at handling large-scale networks.
- Tasks
  - Link prediction and node classification

Experiment --- Sampling-Based NE



(a) Classification on BlogCatalog (b) Classification on Wikipedia (c) Link prediction on BlogCatalog (d) Link prediction on Wikipedia

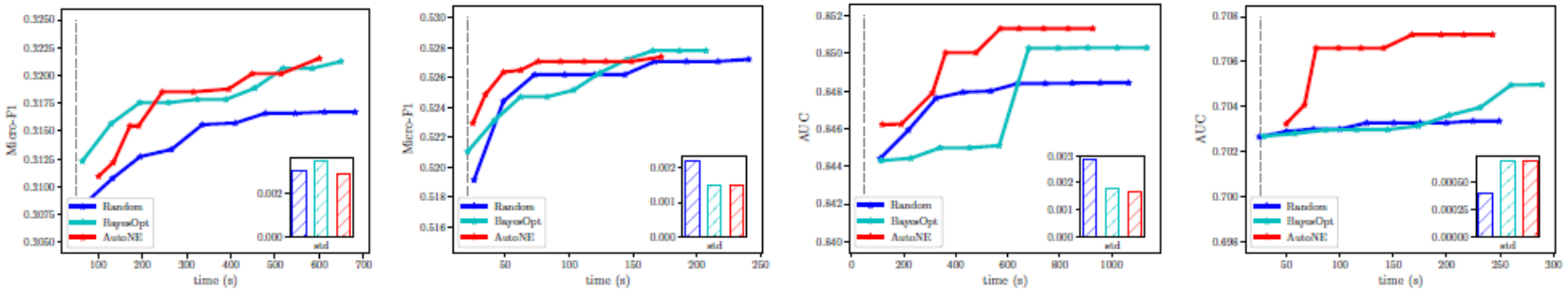
The performance achieved within various time thresholds.



(a) Classification on BlogCatalog (b) Classification on Wikipedia (c) Link prediction on BlogCatalog (d) Link prediction on Wikipedia

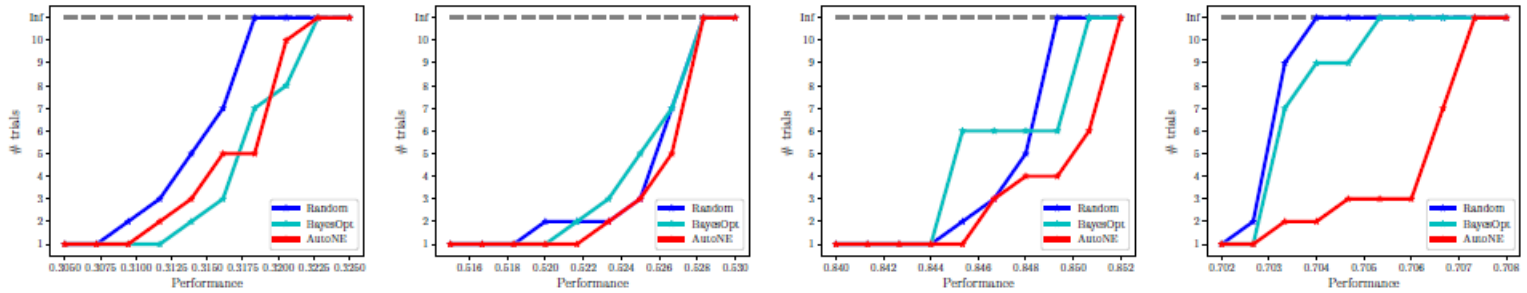
The number of trials to reach a certain performance threshold

Experiment---Factorization-Based NE



(a) Classification on BlogCatalog (b) Classification on Wikipedia (c) Link prediction on BlogCatalog (d) Link prediction on Wikipedia

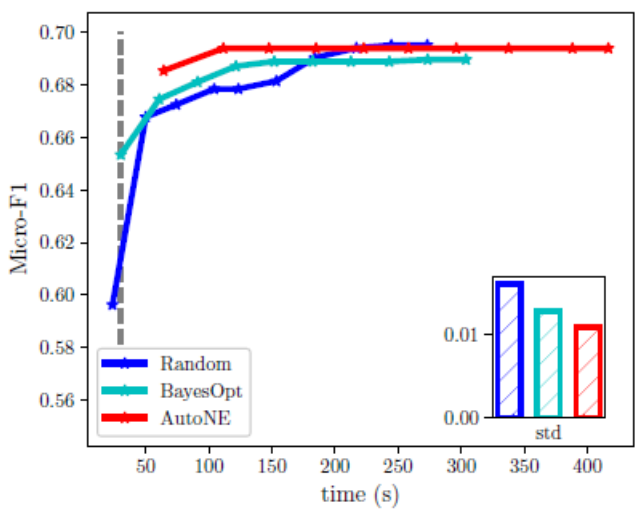
The performance achieved within various time thresholds.



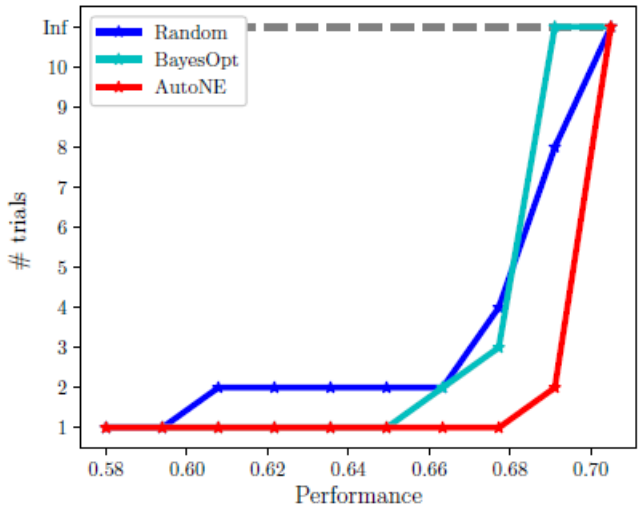
(a) Classification on BlogCatalog (b) Classification on Wikipedia (c) Link prediction on BlogCatalog (d) Link prediction on Wikipedia

The number of trials to reach a certain performance threshold

Experiment --- Deep NN-Based NE



(a) The performance achieved by each method within various time thresholds.



(b) The number of trials required to reach a certain performance threshold.

Node classification on Pubmed.

Experiment --- Large-Scale

Table 1: Results on a massive network with around thirty million edges, where we can only afford to run a NE algorithm on the whole network for a few times.

Method	Trial 1		Trial 2		Trial 3	
	AUC	Time(s)	AUC	Time(s)	AUC	Time(s)
AutoNE	0.717	1067.9	0.726	1856.2	**0.769**	2641.9
Random	0.714	698.3	0.727	1426.3	0.715	2088.6
BayesOpt	0.715	702.5	0.714	1405.1	0.727	2307.7

Conclusion

- Investigate the pressing problem of incorporating AutoML into NE
- Propose a novel framework AutoNE
  - Automate hyperparameter optimization for NE.
- Can scale up to massive real-world networks
  - Utilizing the meta-knowledge transferred from sampled sub-networks.
- Extensive experiment on real-world networks
  - Four real-world dataset
  - Three representative network embedding methods
  - Two baselines

清華大學
Tsinghua University

media and network lab

Meta-learning for multimedia

Outline

- Why meta-learning
- Problem definition
- Meta-learning applications
- Meta-learning algorithms

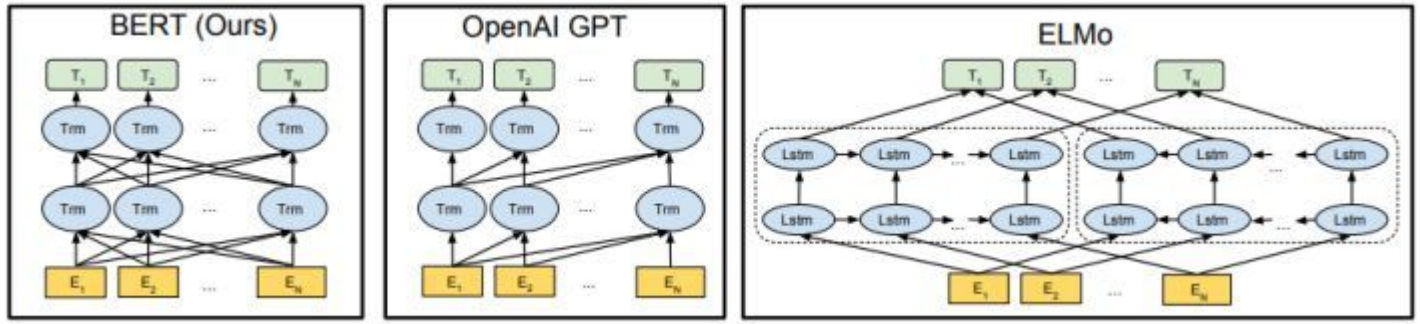
Outline

- Why meta-learning
- Problem definition
- Meta-learning applications
- Meta-learning algorithms

Why meta-learning?

- Computer vision
  - Large (labeled) dataset + Model Training → Classification
- Natural language processing
  - 40GB dataset
  - 1.5 billion parameters

Supervised learning



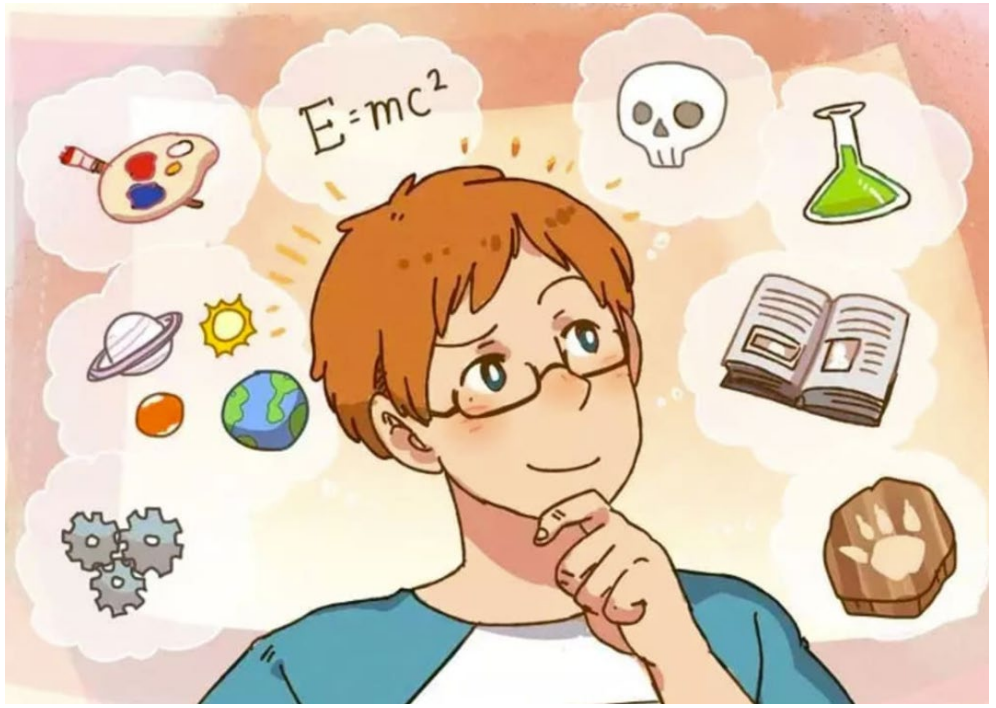
ImageNet, BERT

Why meta-learning?

- How do humans learn and behave intelligently?
  - Learn ability
  - Learn from few data
  - Learn fast
  - Lifelong learning

How to build a general-AI system?

Learning to learn
Learning to learn fast
Meta-learning



Outline

- Why meta-learning
- Problem definition
- Meta-learning applications
- Meta-learning algorithms

Problem definition

Need large labeled data

- Supervised learning:

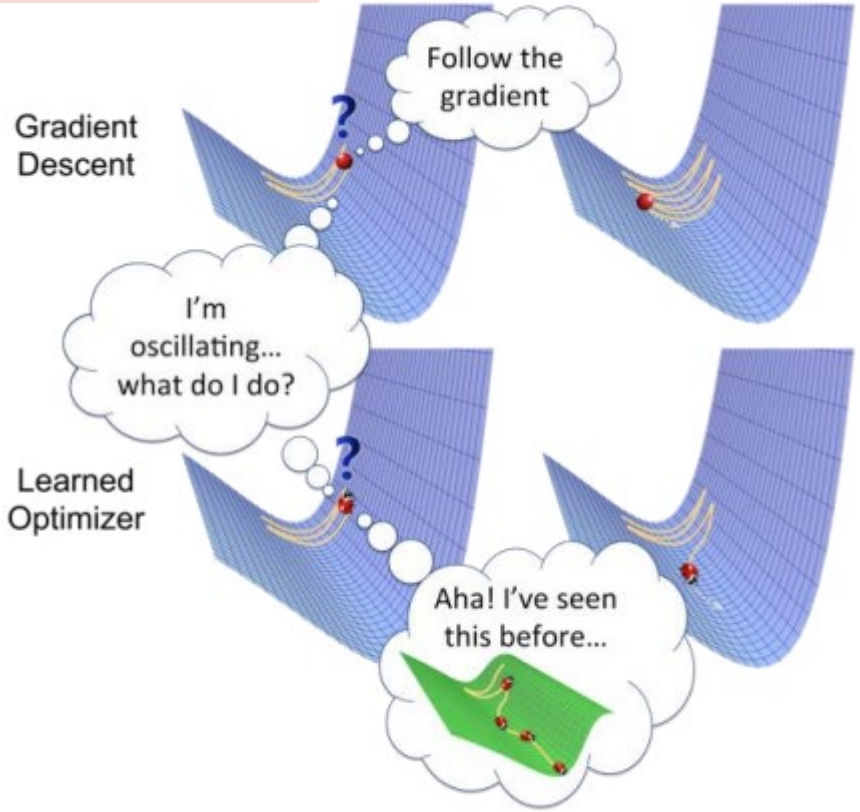
$$f_\theta(x) \to y,\ (x,y)\sim D$$

- Gradient descent

$$argmax_\theta \log p\,(\theta|D)$$
$$= argmax_\theta \log p(D|\theta) + \log p(\theta)$$

likelihood

prior



- Meta-learning:

$$f_\varphi(\theta, x) \to y, (x,y)\sim D_{meta-test}$$

$$D_{meta-train} \to \theta$$

image credit: Ke Li

Problem definition

- Learn from **prior experience** in a systematic, data-driven way

- Multi-task learning
- Ensemble learning

→

- NOT involve learning from prior experience on other tasks

Transfer learning ?

- Meta-learning:

$$f_\varphi(\theta, x) \rightarrow y, (x, y) \sim D_{meta-test} \qquad D_{meta-train} \rightarrow \theta$$

Meta learner

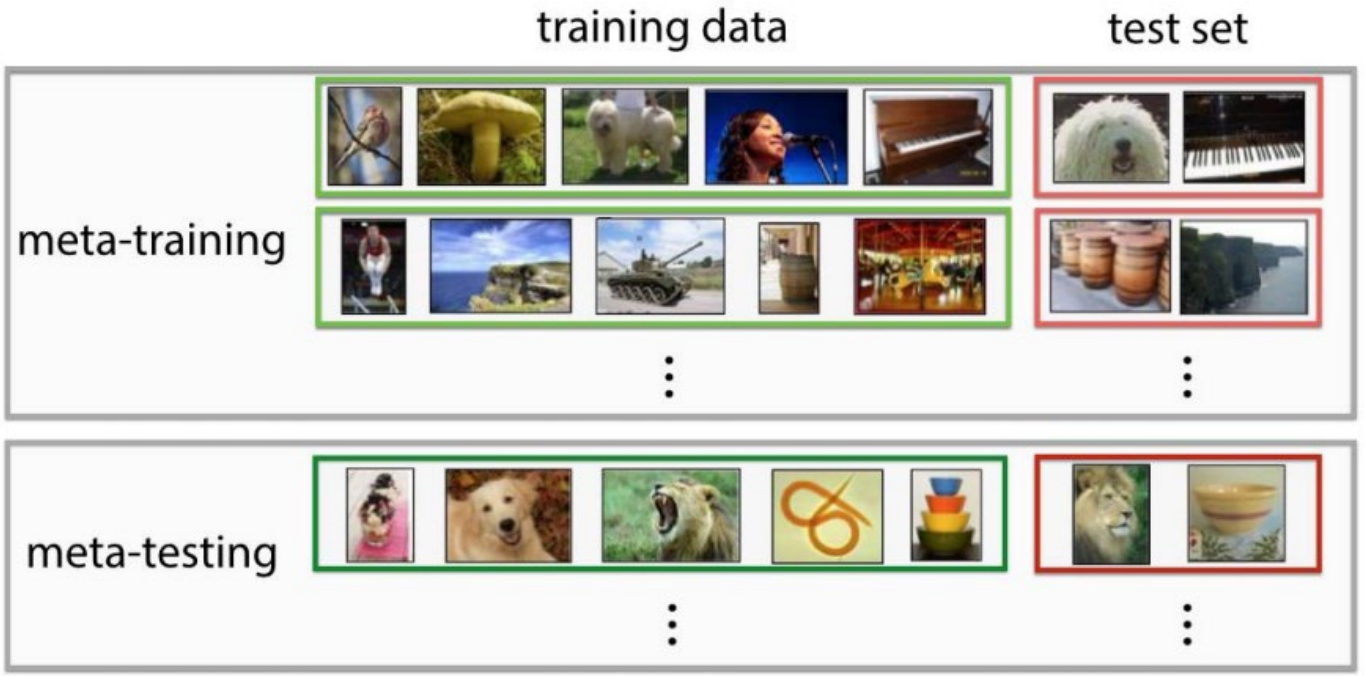
Meta-testing data

Meta-training data

Problem definition

- Meta learner: learn how to learn efficiently with new datasets
- Learner: normal learning paradigm

- $D_{meta-train} = \{D_1, D_2, \ldots, D_n\}$
- $D_{meta-test} = \{D_{n+1}, \ldots, D_m\}$
- $D_i = \{(x_k^i, y_k^i,), \ldots, (x_k^i, y_k^i)\}$



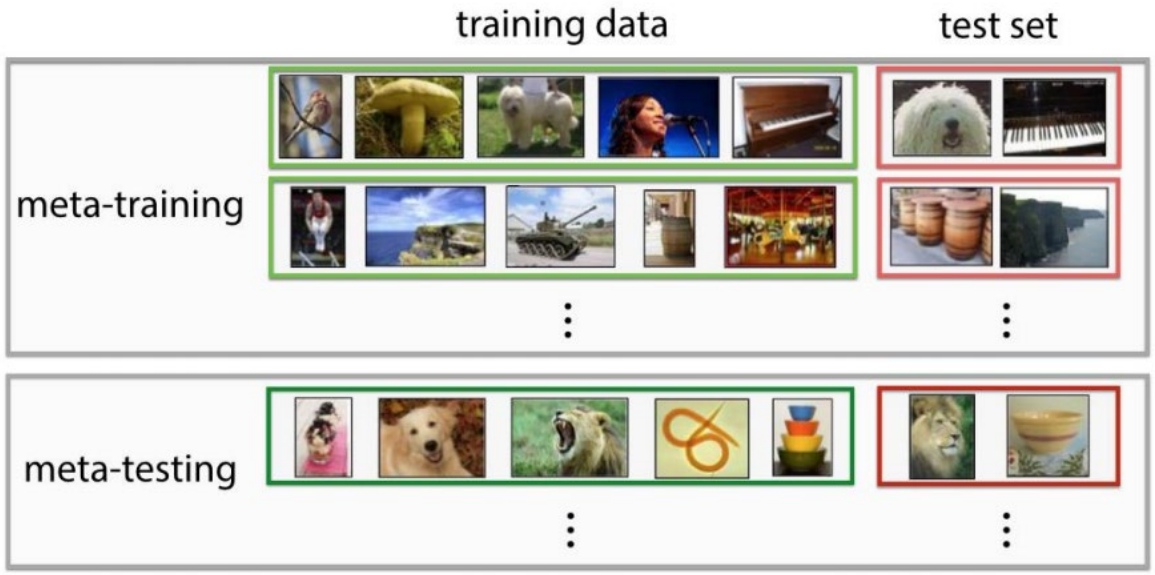
Ravi & Larochelle '17

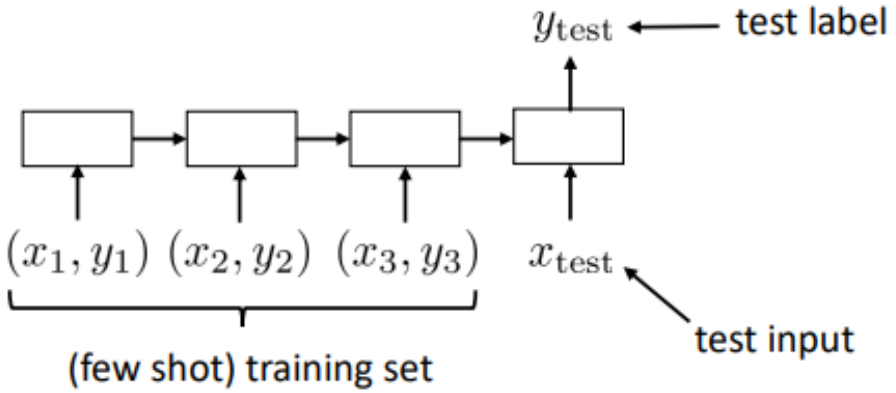
Outline

- Why meta-learning
- Problem definition
- Meta-learning applications
- Meta-learning algorithms

Few-shot learning

- Image Classification: k shot n way classification problem,
  - k is the number of labeled samples per class
  - n is the number of classes in one "data sample"

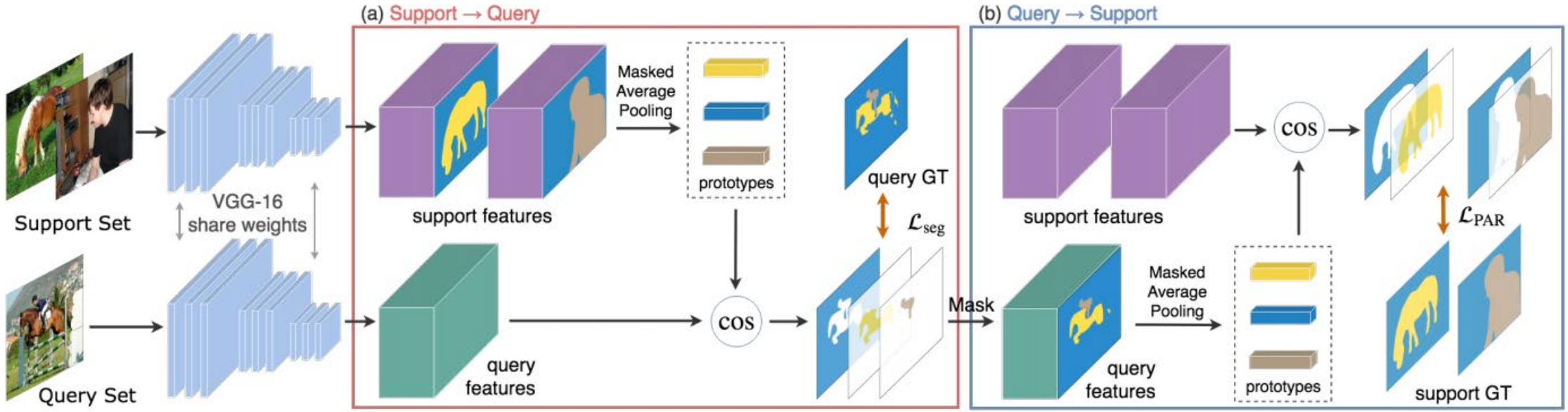




Ravi & Larochelle '17

Few-shot learning

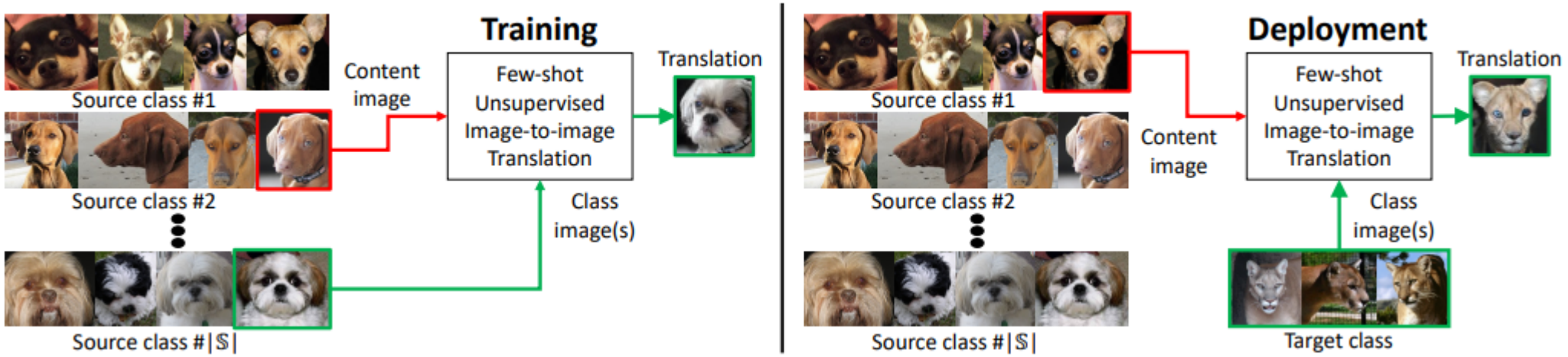
- Image Segmentation



PANet: Few-Shot Image Semantic Segmentation with Prototype Alignment

Few-shot learning

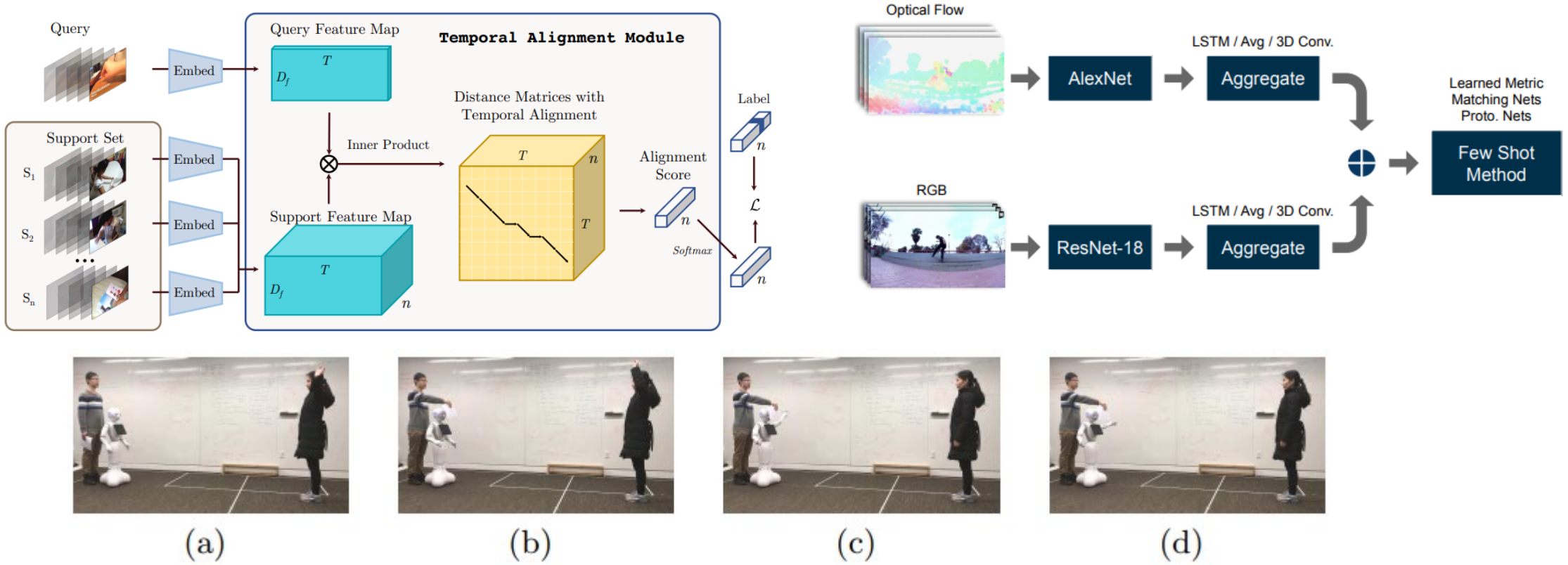
- Image Generation & Image-to-image Translation



FIGR: Few-shot Image Generation with Reptile.
Few-Shot Unsupervised Image-to-Image Translation.

Few-shot learning

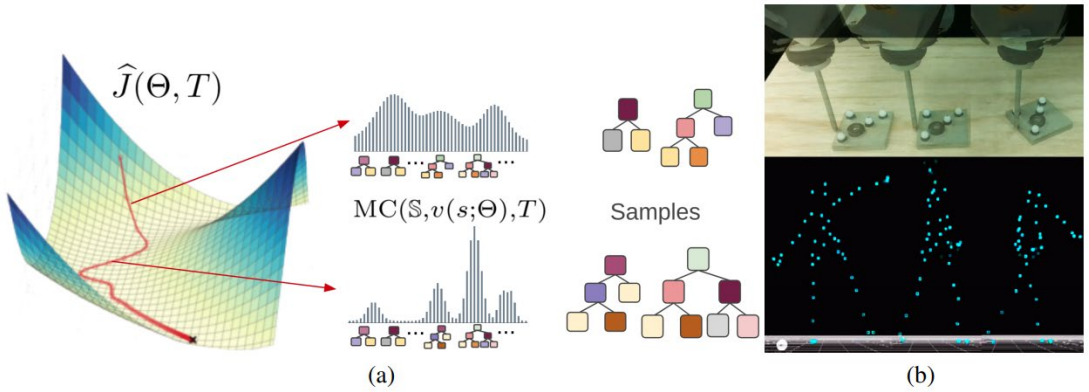
- Video Recognition & Prediction

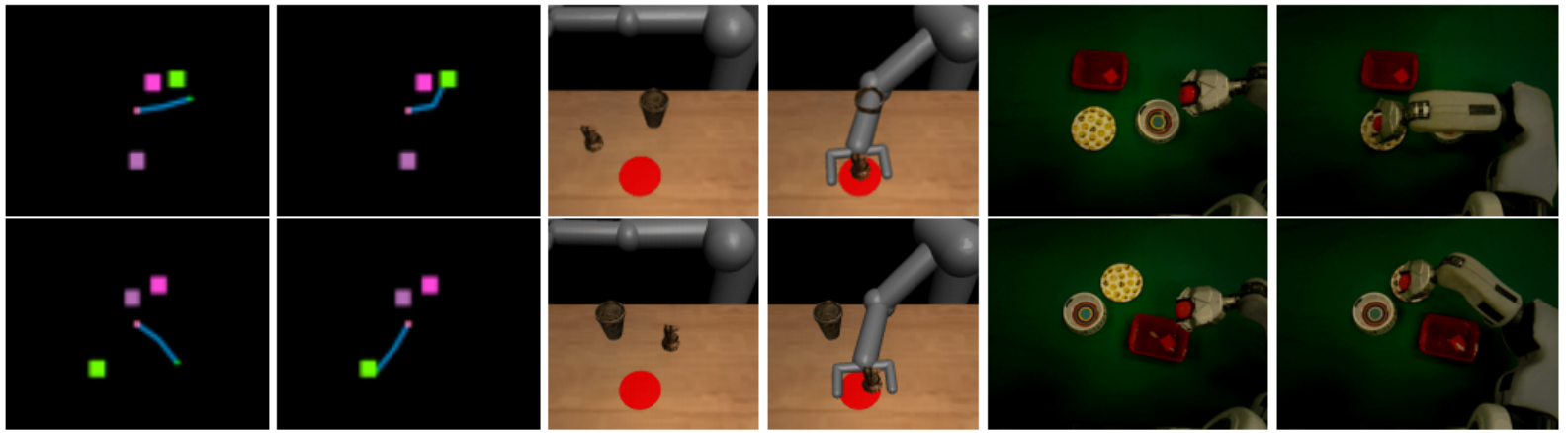


Few-Shot Video Classification via Temporal Alignment.
Metric-Based Few-Shot Learning for Video Action Recognition.
Few-Shot Human Motion Prediction via Meta-Learning.

Few-shot learning

- Robotic Imitation

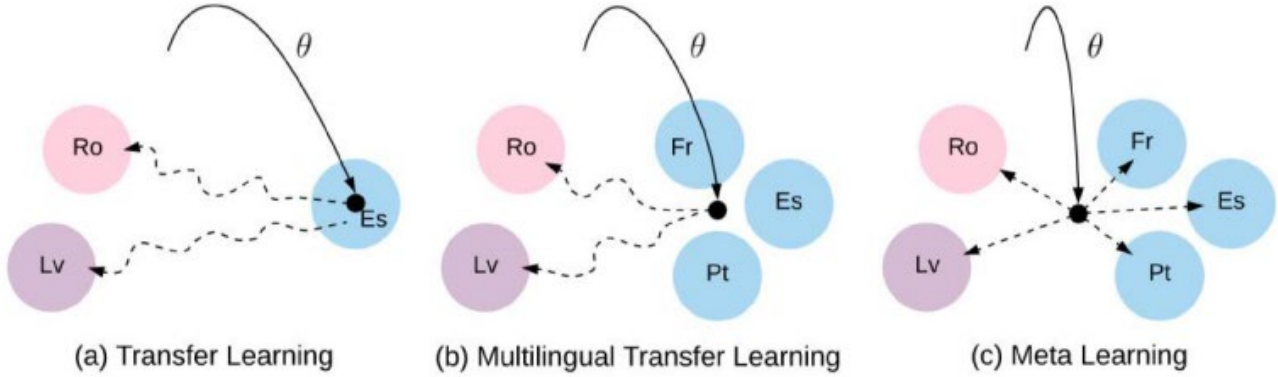




Modular Meta-Learning.
One-Shot Visual Imitation Learning via Meta-Learning.

Meta-learning in NLP

- Few-shot text classification
- Adapt to new languages
- Adapt to new words

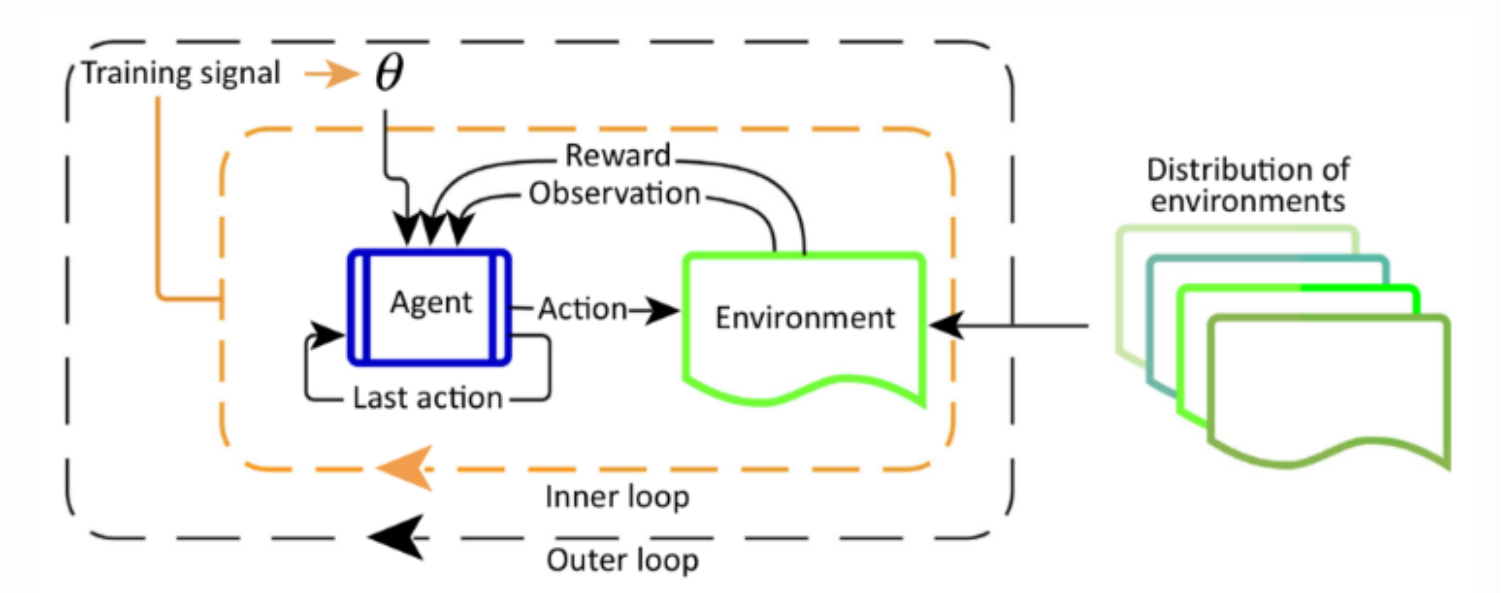


1. an experimental vaccine can alter the immune response of people infected with the aids virus a <blank_token> u.s. scientist said.	prominent
2. the show one of five new nbc <blank_token> is the second casualty of the three networks so far this fall.	series
3. however since eastern first filed for chapter N protection march N it has consistently promised to pay creditors N cents on the <blank_token>.	dollar
4. we had a lot of people who threw in the <blank_token> today said <unk> ellis a partner in benjamin jacobson & sons a specialist in trading ual stock on the big board.	towel
5. it's not easy to roll out something that <blank_token> and make it pay mr. jacob says.	comprehensive
Query: in late new york trading yesterday the <blank_token> was quoted at N marks down from N marks late friday and at N yen down from N yen late friday.	dollar

Induction Networks for Few-Shot Text Classification.
Meta-Learning for Low-Resource Neural Machine Translation.
Matching Networks for One Shot Learning.

Meta-learning in RL

- *Meta-learn* a faster reinforcement learner to learn new tasks efficiently
  - Explore more intelligently
  - Avoid trying actions that are know to be useless
  - Acquire the right features more quickly



Botvinick, et al. 2019

Outline

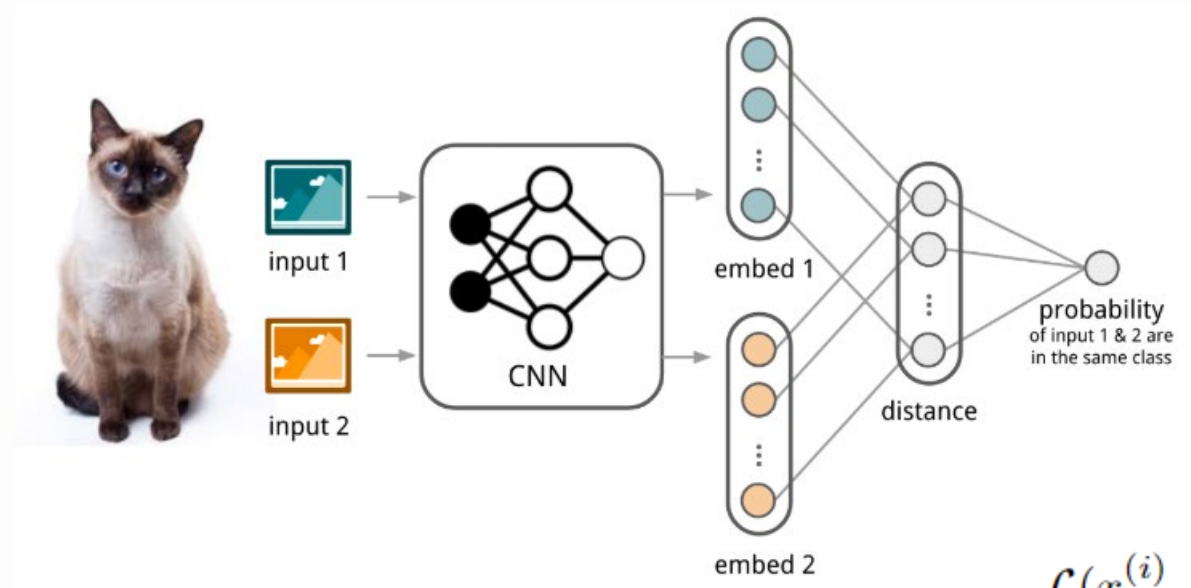
- Why meta-learning
- Problem definition
- Meta-learning applications
- Meta-learning algorithms

Meta-learning algorithms

- Learning from feature similarity
- Learning from task properties
- Model-based algorithm
- Optimization-based algorithm

Feature similarity

- Siamese Neural Networks

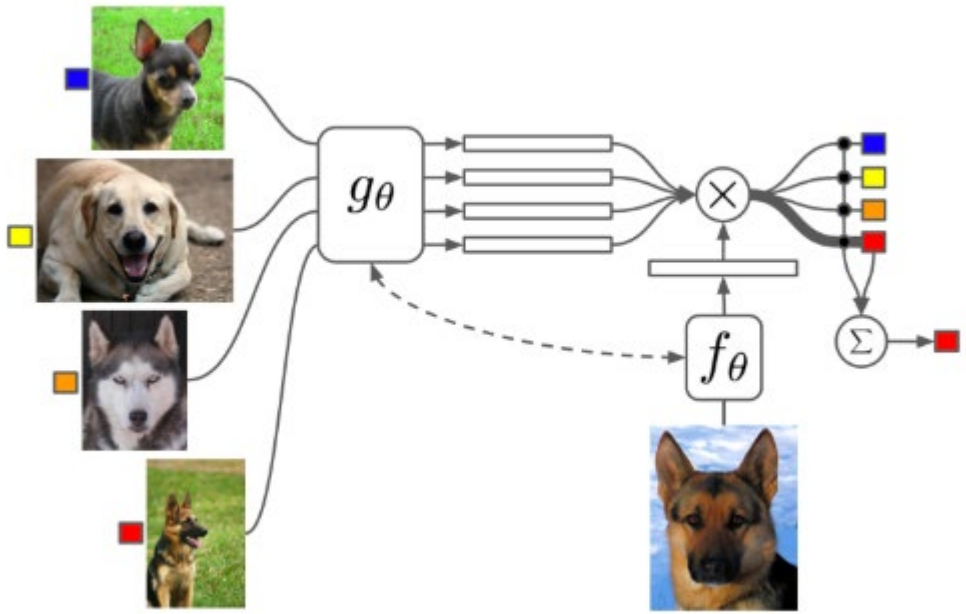


$$\mathcal{L}(x_1^{(i)}, x_2^{(i)}) = \mathbf{y}(x_1^{(i)}, x_2^{(i)}) \log \mathbf{p}(x_1^{(i)}, x_2^{(i)}) + (1 - \mathbf{y}(x_1^{(i)}, x_2^{(i)})) \log (1 - \mathbf{p}(x_1^{(i)}, x_2^{(i)})) + \boldsymbol{\lambda}^T |\mathbf{w}|^2$$

Siamese Neural Networks for One-shot Image Recognition.

Feature similarity

- Matching Networks: make meta-train & meta-test match



Full Context Embeddings

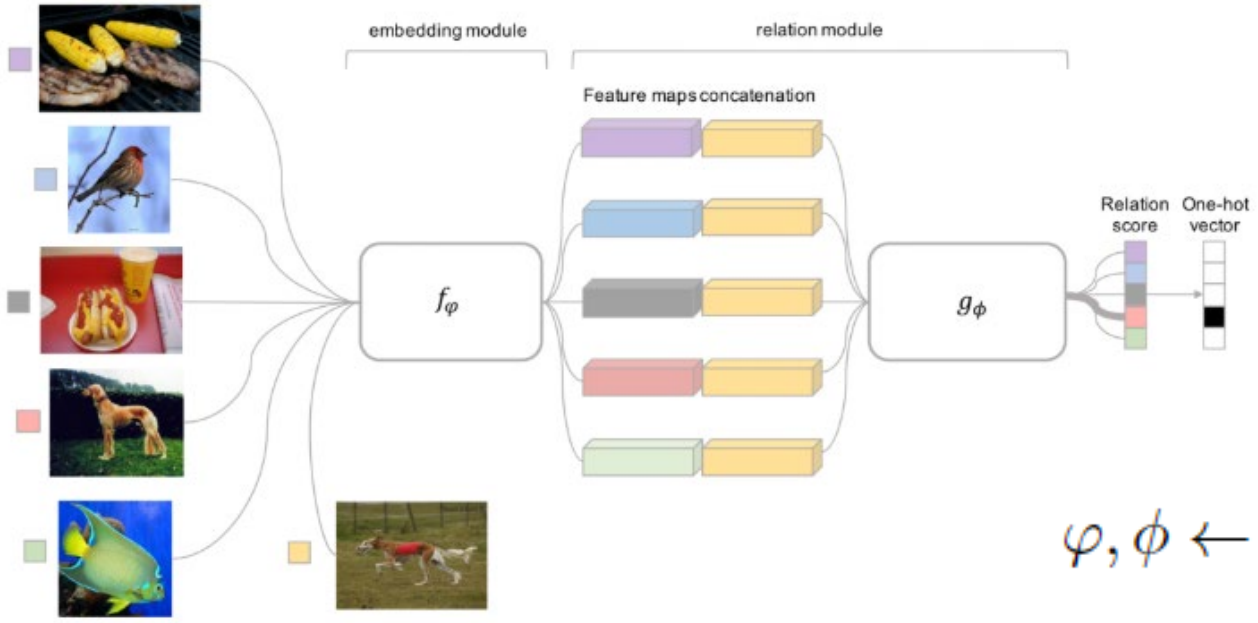
$$\hat{h}_k, c_k = \text{LSTM}(f'(\hat{x}), [h_{k-1}, r_{k-1}], c_{k-1})$$
$$h_k = \hat{h}_k + f'(\hat{x})$$
$$r_{k-1} = \sum_{i=1}^{|S|} a(h_{k-1}, g(x_i)) g(x_i)$$
$$a(h_{k-1}, g(x_i)) = e^{h_{k-1}^T g(x_i)} / \sum_{j=1}^{|S|} e^{h_{k-1}^T g(x_j)}$$

$$P(\hat{y}|\hat{x}, S) = \sum_{i=1}^{k} a(\hat{x}, x_i) y_i$$

Matching Networks for One Shot Learning.

Feature similarity

- Relation Networks



- Use a CNN model to capture relationships
- MSE Loss
  - Predict relation scores
  - More like a regression problem

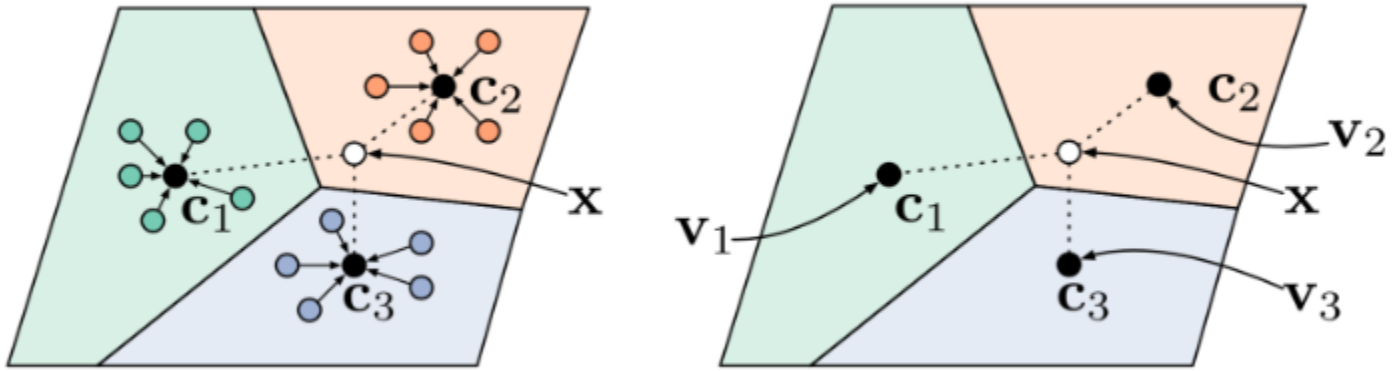
$$\varphi, \phi \leftarrow \underset{\varphi,\phi}{\operatorname{argmin}} \sum_{i=1}^{m}\sum_{j=1}^{n}(r_{i,j} - \mathbf{1}(y_i == y_j))^2$$

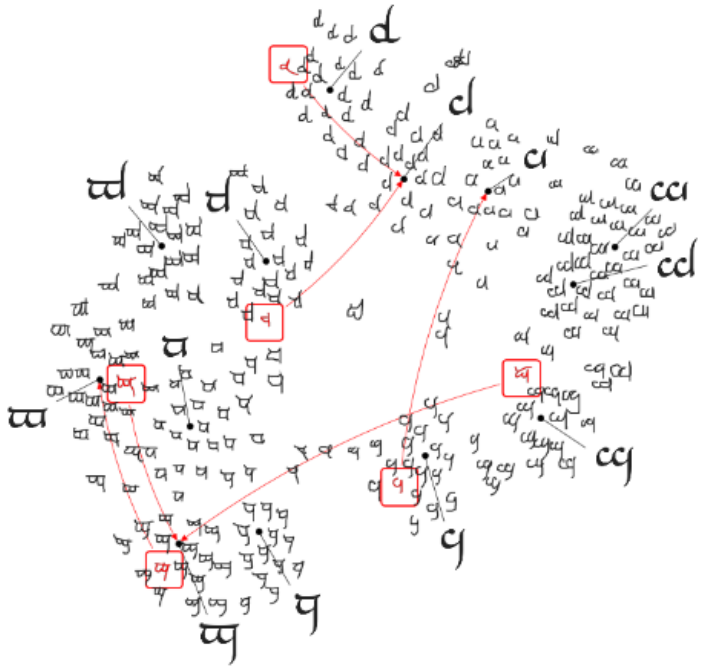
Learning to Compare: Relation Network for Few-Shot Learning.

Feature similarity

- Prototypical Networks

$$p_{\phi}(y = k \mid \mathbf{x}) = \frac{\exp(-d(f_{\phi}(\mathbf{x}), \mathbf{c}_k))}{\sum_{k'} \exp(-d(f_{\phi}(\mathbf{x}), \mathbf{c}_{k'}))}$$

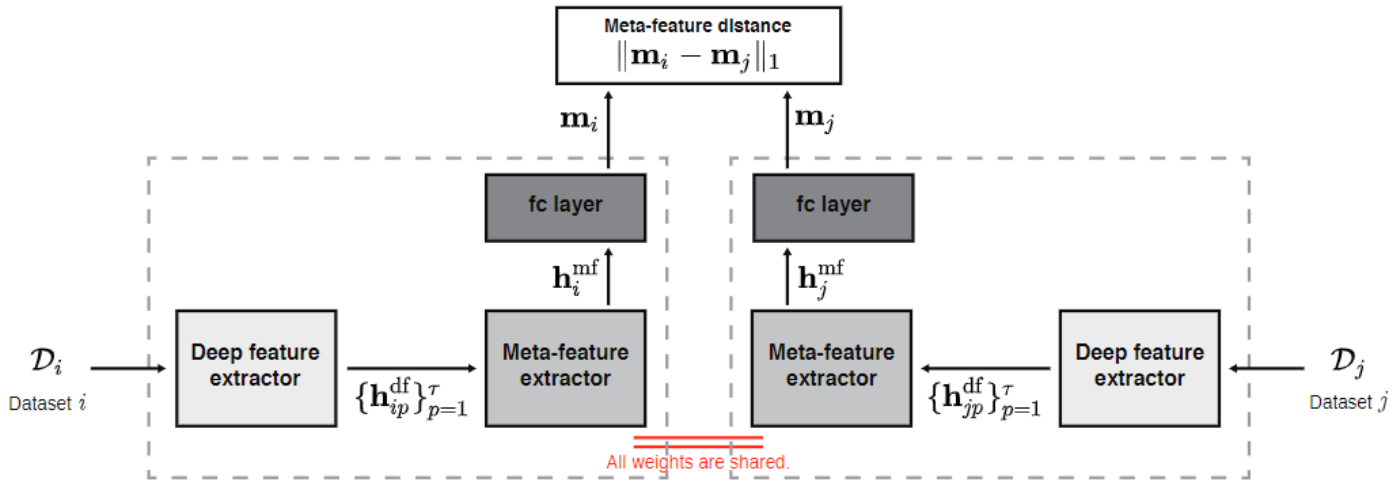




Prototypical Networks for Few-shot Learning.

Task properties

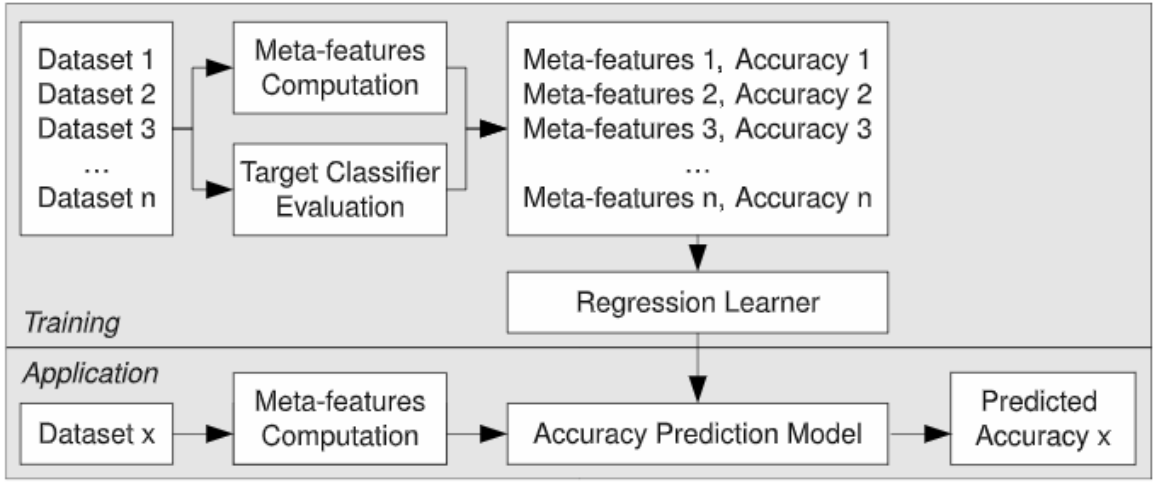
- Learn meta-features of tasks
  - Use meta-features to represent a task $m(t_j)$
  - Task similarity can be measured by the distance between $m(t_i), m(t_j)$
  - Transfer information from the most similar task to the new task



Learning to Warm-Start Bayesian Hyperparameter Optimization.

Task properties

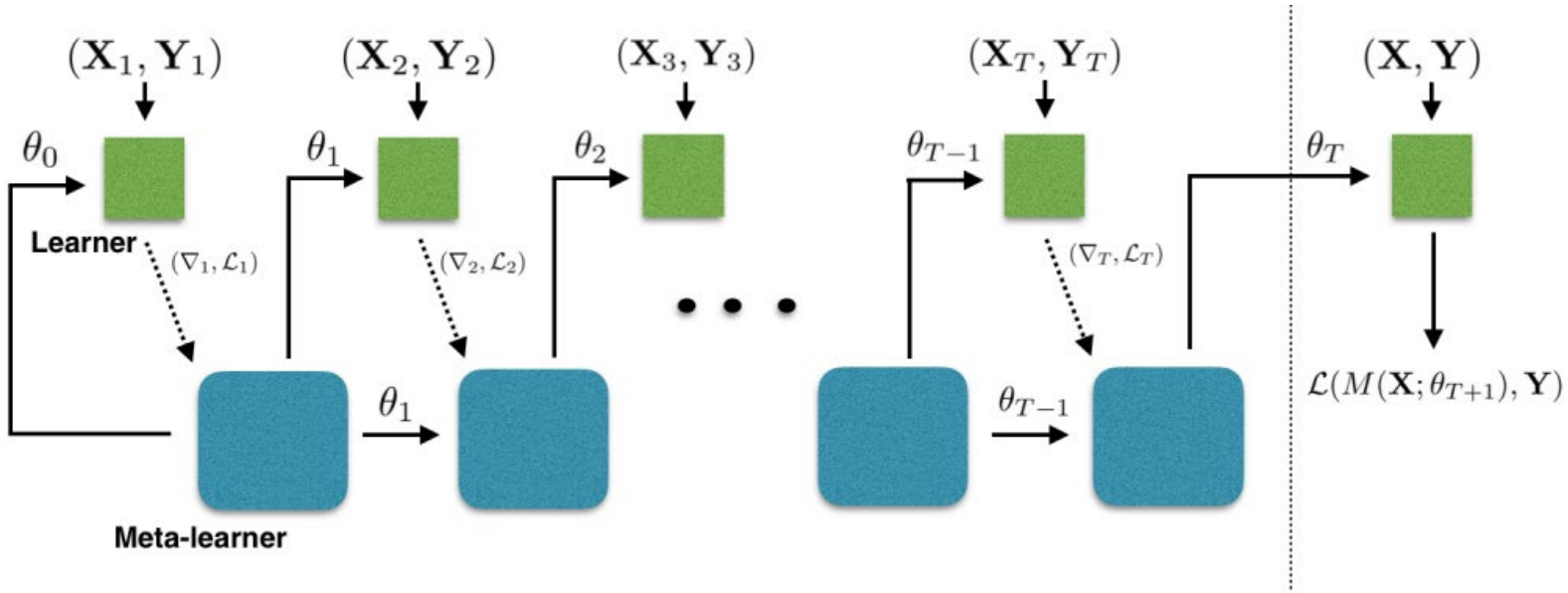
- Configurations transfer & selection
  - Build meta-model *L* to learn the complex relationship between meta-features specific configurations
  - Recommend the most useful configurations Θ for new task
  - e.g. ranking, performance prediction



Automatic Classifier Selection for Non-Experts.

Model-based algorithm

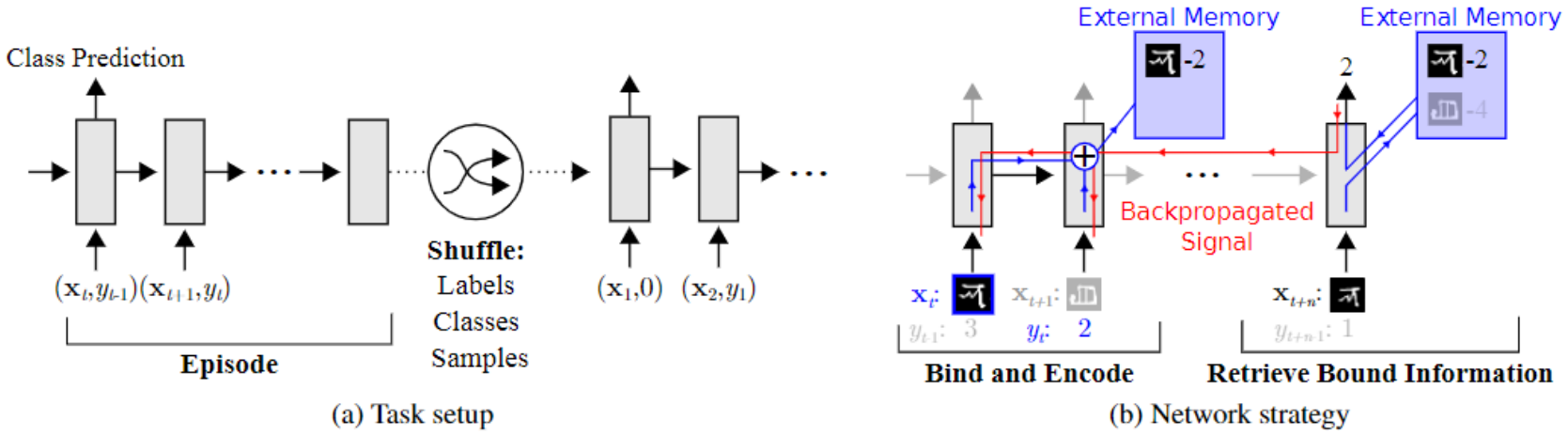
- Black-box optimization
  - No assumptions on task characteristics and configurations
  - Depends on a model designed specifically
  - e.g. RNN, memory network



Optimization as a model for few-shot learning.

Model-based algorithm

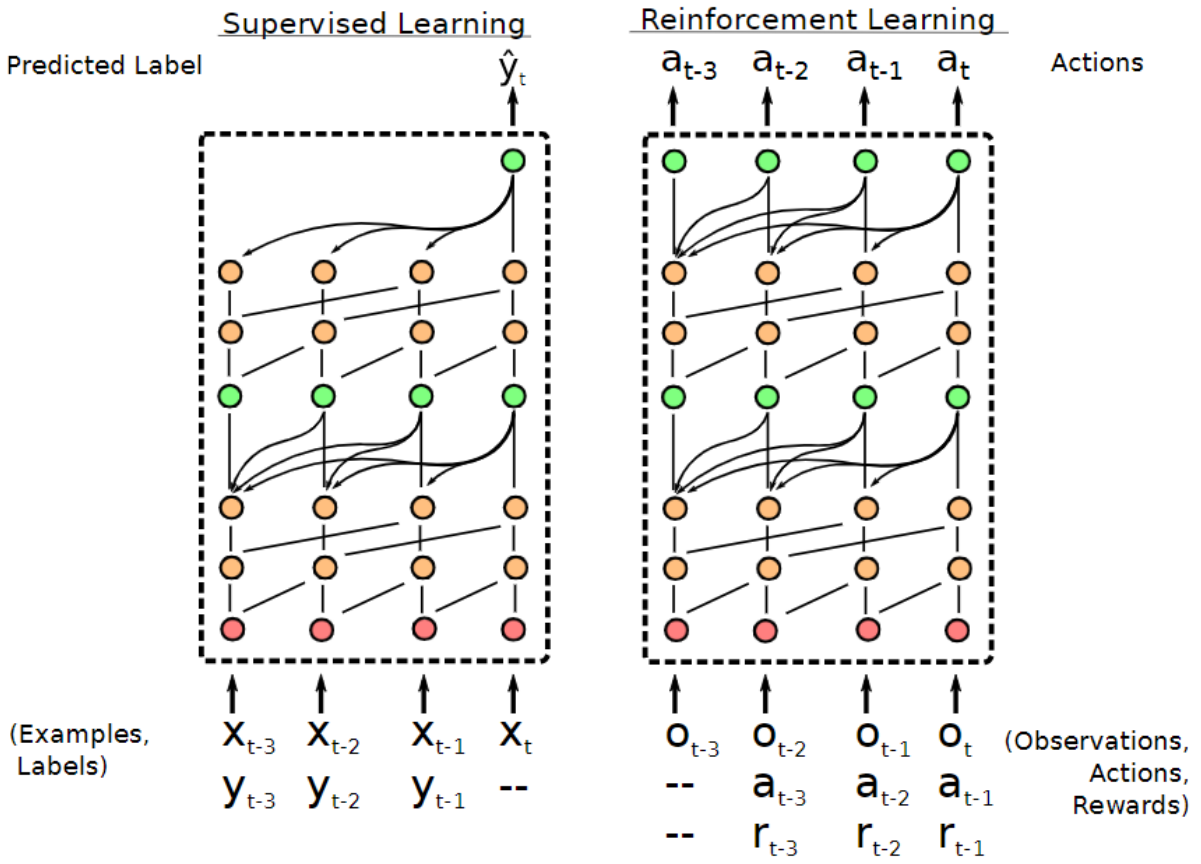
- Black-box optimization
  - No assumptions on task characteristics and configurations
  - Depends on a model designed specifically
  - e.g. RNN, memory network



Meta-Learning with Memory-Augmented Neural Networks.

Model-based algorithm

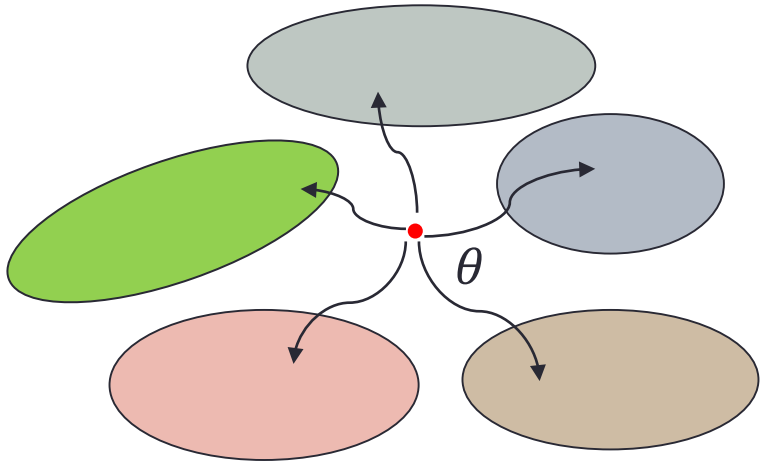
- Black-box optimization
  - Attentive network



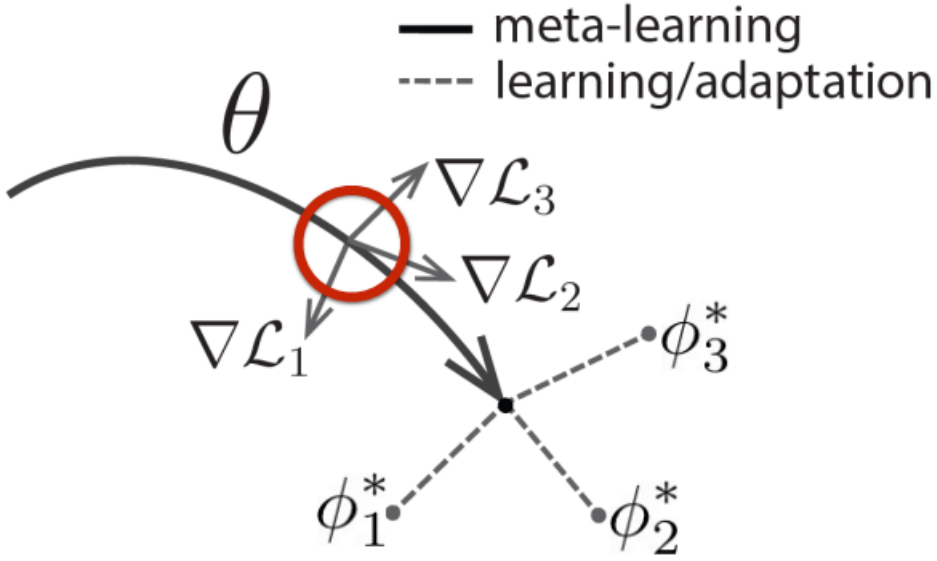
A simple neural attentive meta-learner.

Optimization-based algorithm

- Model-Agnostic Meta-learning (MAML)
  - Learn a model parameter initialization that generalizes better to similar tasks
  - Train θ that fits the tasks globally with meta-gradient computed from a batch of previous tasks $\emptyset$



Parameter space



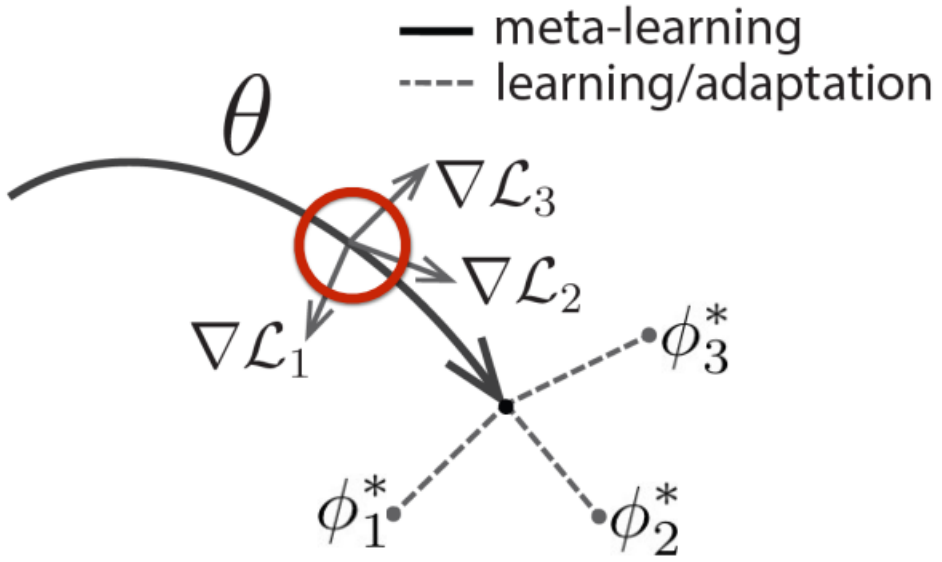
Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks.

Optimization-based algorithm

- Model-Agnostic Meta-learning (MAML)
  - At each iteration, select a batch of tasks $\{T_j\}$
  - Use current θ as initializations to learn from every T_j
  - Calculate the meta-gradient w.r.t. θ on these tasks
  - Use the meta-gradient to update θ

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

- After each iteration, θ becomes a better initialization to start fine-tuning any of the tasks



Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks.

First-order approximation of MAML

- Recall the gradient step on θ

$$\theta_i' = \theta - \alpha \nabla_\theta L_{T_i}(f_\theta), \theta \leftarrow \theta - \beta \nabla_\theta L_{T_i}(f_{\theta_i'})$$

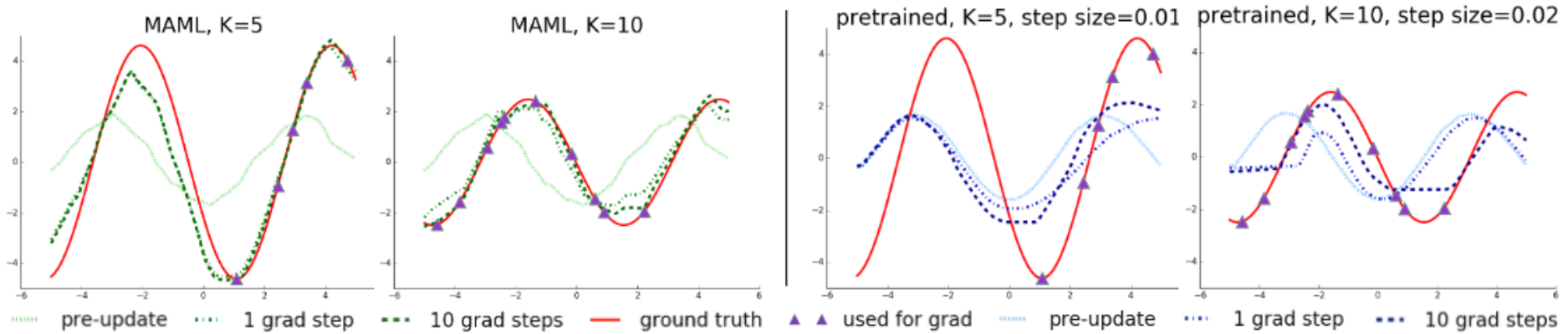
- It relates with the Hessian of $L_{T_i}(f_\theta)$

$$\nabla_\theta L_{T_i}\left(f_{\theta_i'}\right) = \nabla_{\theta_i'} L_{T_i}\left(f_{\theta_i'}\right) \left(I - \alpha \nabla_\theta^2 L_{T_i}(f_\theta)\right)$$

- Simply discard the Hessian in the formula for computational efficiency
- The performance is comparable

Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks.

Optimization-based algorithm



Omniglot (Lake et al., 2011)	5-way Accuracy 1-shot	5-way Accuracy 5-shot	20-way Accuracy 1-shot	20-way Accuracy 5-shot
MANN, no conv (Santoro et al., 2016)	82.8%	94.9%	–	–
MAML, no conv (ours)	**89.7 ± 1.1%**	**97.5 ± 0.6%**	–	–
Siamese nets (Koch, 2015)	97.3%	98.4%	88.2%	97.0%
matching nets (Vinyals et al., 2016)	98.1%	98.9%	93.8%	98.5%
neural statistician (Edwards & Storkey, 2017)	98.1%	99.5%	93.2%	98.1%
memory mod. (Kaiser et al., 2017)	98.4%	99.6%	95.0%	98.6%
MAML (ours)	**98.7 ± 0.4%**	**99.9 ± 0.1%**	**95.8 ± 0.3%**	**98.9 ± 0.2%**

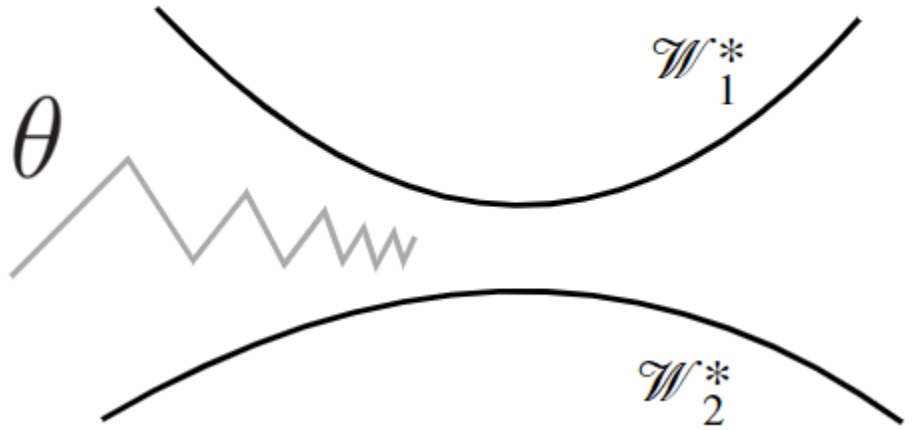
Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks.

Optimization-based algorithm

- On First-Order Meta-Learning Algorithms (Reptile)
  - No meta-gradient computation
  - Moving θ towards new parameters of tasks $\{T_j\}$

~~$\theta_i' = \theta - \alpha \nabla_\theta L_{\tau_i}(f_\theta), \theta \leftarrow \theta - \beta \nabla_\theta L_{\tau_i}(f_{\theta_i'})$~~

$$\theta_i' = \theta - \alpha \nabla_\theta L_{\tau_i}(f_\theta), \theta \leftarrow \theta + \epsilon \frac{1}{n} \sum_{i=1}^{n} (\theta_i' - \theta)$$



Reptile: a Scalable Meta learning Algorithm.

Pros and Cons

- Feature & task properties based
  - ☺ simple
  - ☺ easy to optimize
  - ☹ hard to generalize
  - ☹ hard to scale to large k
- Model-based
  - ☺ easy to apply to variety of problems
  - ☺ existing optimization methods
  - ☹ black-box
  - ☹ depending on model design
  - ☹ data inefficient
- Optimization-based
  - ☺ flexible to adapt to new tasks
  - ☺ model agnostic
  - ☹ second-order optimization

Auto-Lifelong: An AutoML System for Lifelong Machine Learning

---Second Place in **NeurIPS 2018 AutoML Challenge**

Outline

- Introduction
- NeurIPS 2018 AutoML Challenge
- Our System
- Results
- Conclusion

Introduction

- AutoML aims to automatically design machine learning pipelines
  - data preprocessing
  - feature engineering
  - model selection
  - hyperparameter optimization
  - ……

Auto-Sklearn

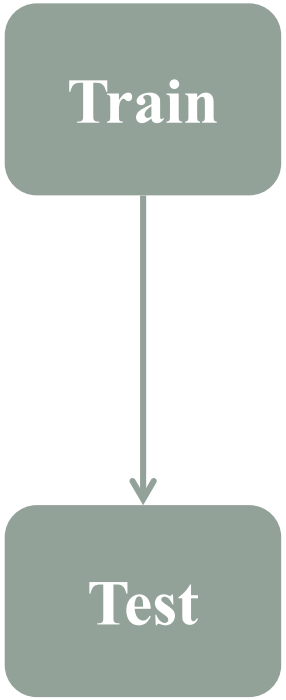
H2O.ai

Introduction

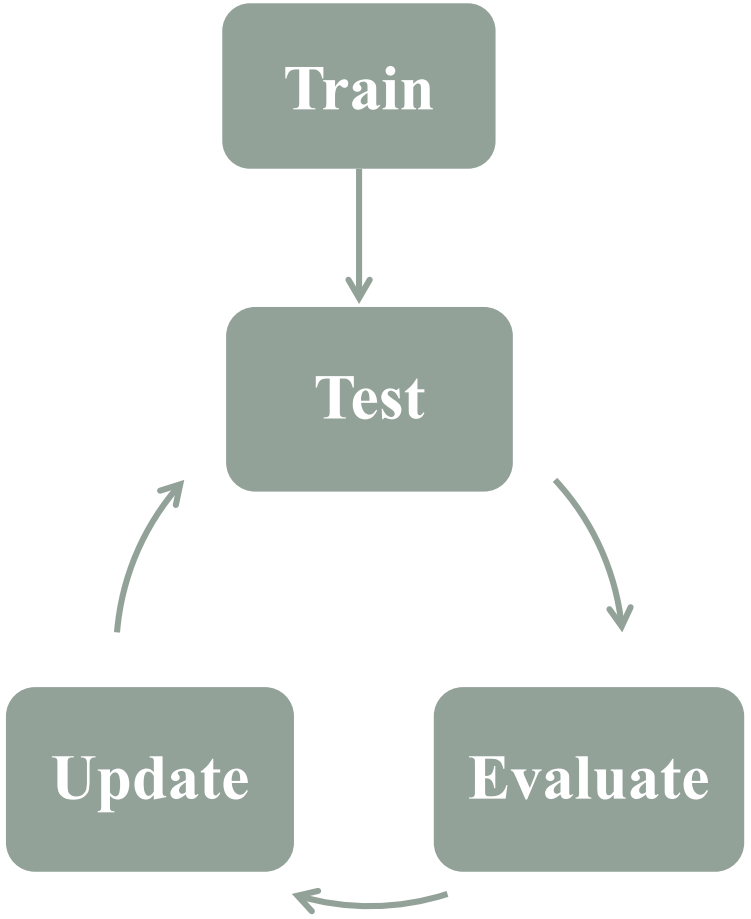
- Existing AutoML systems are mainly designed for isolated learning
- However, In many real-world problems, data arrives continuously in successive batches, which raises a lifelong learning challenge for AutoML
  - online advertising
  - Recommendation
  - fraud detection
  - ……

Introduction

Isolated Learning



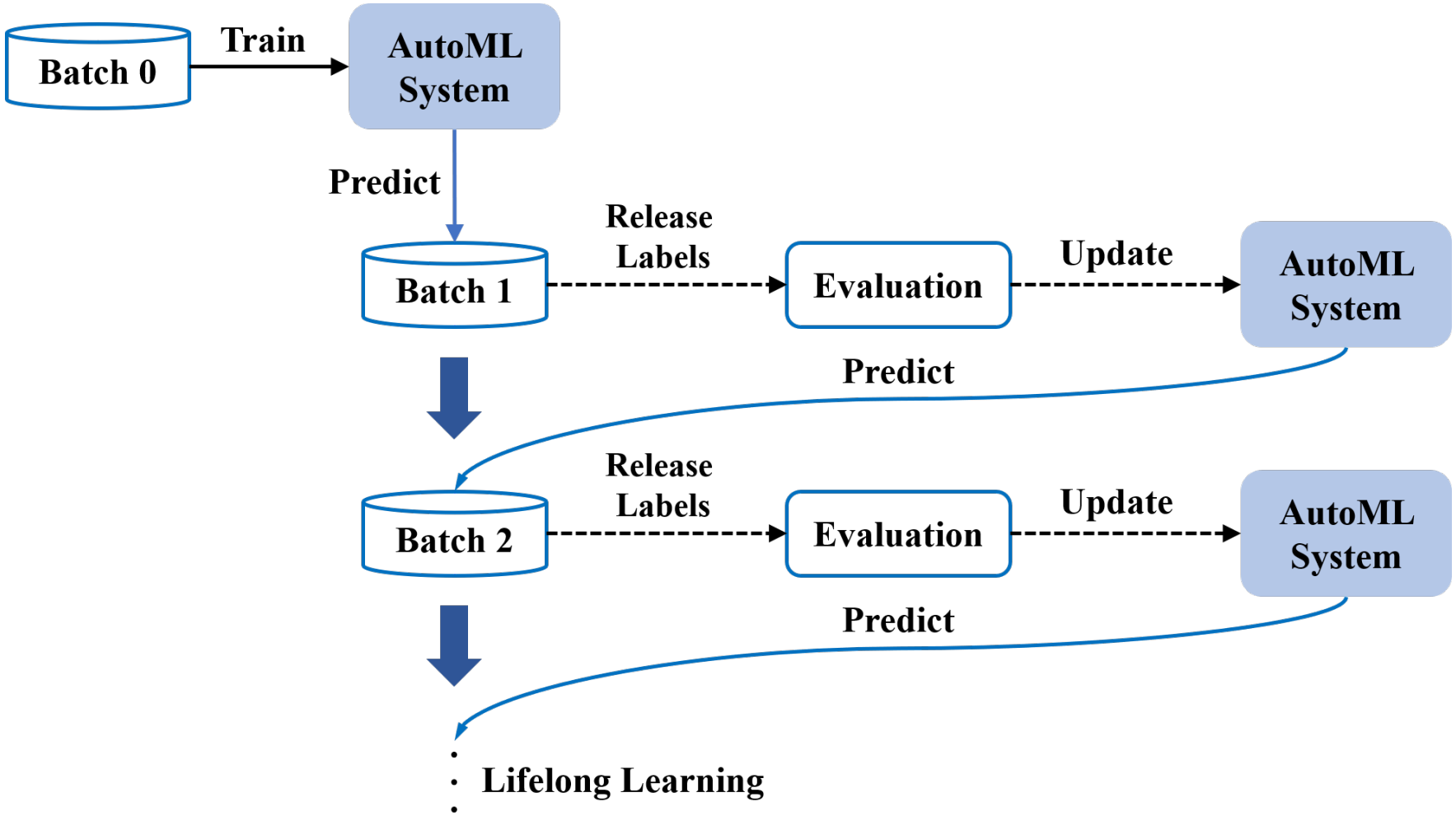
Lifelong Learning



Introduction

- Existing AutoML systems are mainly designed for isolated learning
- However, In many real-world problems, data arrives continuously in successive batches, which raises a lifelong learning challenge for AutoML
- **Our work: design an AutoML system for lifelong machine learning, which can learn continuously, accumulate the knowledge learned in the past, and utilize it for future learning**

NeurIPS 2018 AutoML Challenge



NeurIPS 2018 AutoML Challenge

Feedback Phase

- Develop the AutoML system on 5 public datasets

Blind Test Phase

- The last submission of the feedback phase is blindly tested on 5 new datasets without any human intervention

Evaluation Metric

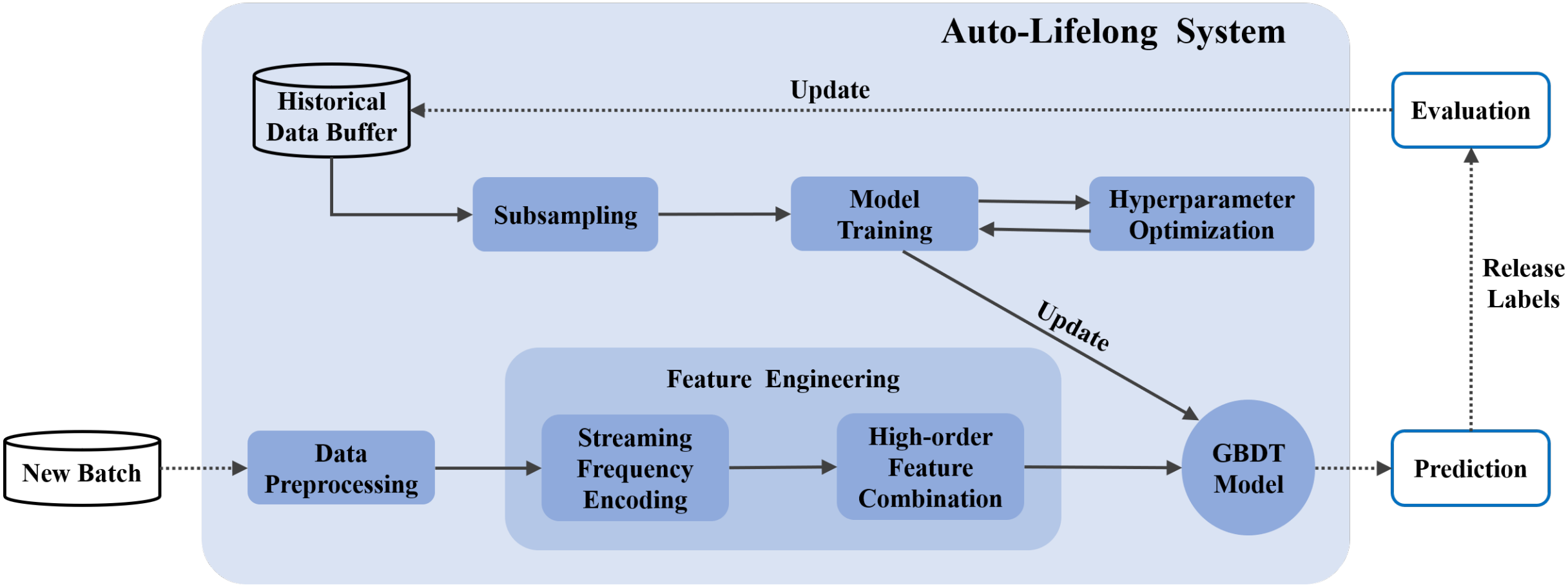
- Score on each dataset = average normalized AUC over all test batches
- Final rank = average rank over the 5 test datasets in the blind test phase

NeurIPS 2018 AutoML Challenge

Key Difficulties

- High-cardinality categorical features
- Varied feature types with complicated dependencies
- Concept drift: data distribution changing slowly over time

Our System


Auto-Lifelong System
Historical Data Buffer
Update
Evaluation
Subsampling
Model Training
Hyperparameter Optimization
Release Labels
Update
Feature Engineering
New Batch
Data Preprocessing
Streaming Frequency Encoding
High-order Feature Combination
GBDT Model
Prediction

Our System

Key Difficulties

- High-cardinality categorical features
- Varied types of features
- Concept drift

Corresponding Solutions

- Frequency encoding
- High-order feature combination
- Retraining & streaming encoding

Our System

Categorical Feature Encoding

- **Why encode?**
  - Transform nominal categories into numerical representation which can be handled by machine learning models

- **How to encode?**
  - Compact representation of high-cardinality categorical features
  - Ordinal representation with practical meaning

Our System

Categorical Feature Encoding

Frequency encoding: map each category X to its frequency in the batch

$$X \rightarrow P(X)$$

Sample #	Category	Frequency Encoding
1	A	0.5
2	B	0.25
3	A	0.5
4	C	0.25

Our System

High-order Feature Combination

- **Why?**
  - Extract useful information from the interaction between different features

- **Key challenges**
  - Adaptability: the optimal feature combination subset varies across datasets
  - Efficiency: select effective feature transformations in a huge combinatorial space within an affordable time budget

Our System

High-order Feature Combination

Search space: Predefine a set of binary transformations based on prior knowledge

Operand A	Operand B	Transformation
Numerical	Numerical	+, −, ×, ÷
Categorical	Numerical	num_mean_groupby_cat
Categorical	Categorical	cat_cat_combine cat_nunique_groupby_cat
Categorical	Temporal	time_difference_groupby_cat

Our System

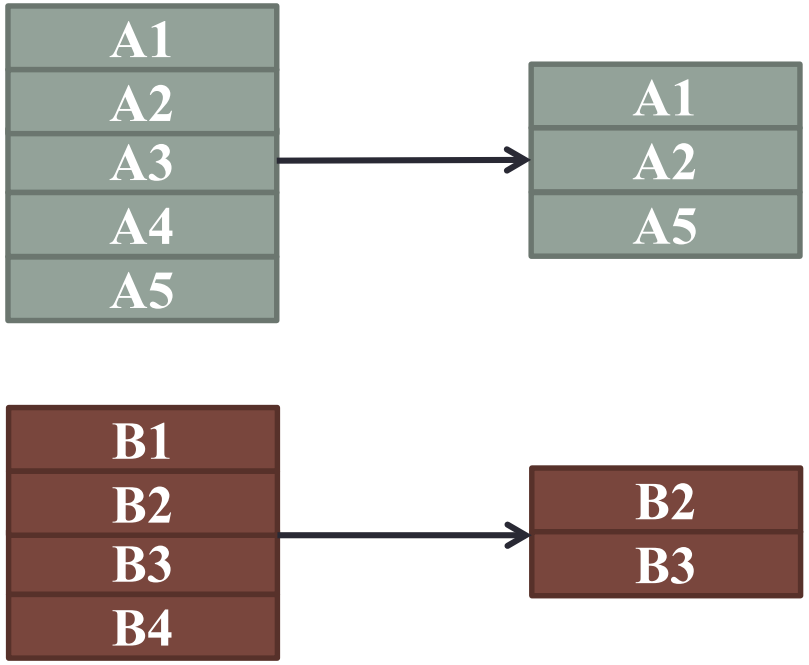
High-order Feature Combination

Search strategy: Apply each type of transformation on the original feature set to generate and select new features in an expansion-reduction fashion:

1. **Pre-selection**
2. **Feature generation**
3. **Post-selection**

Our System

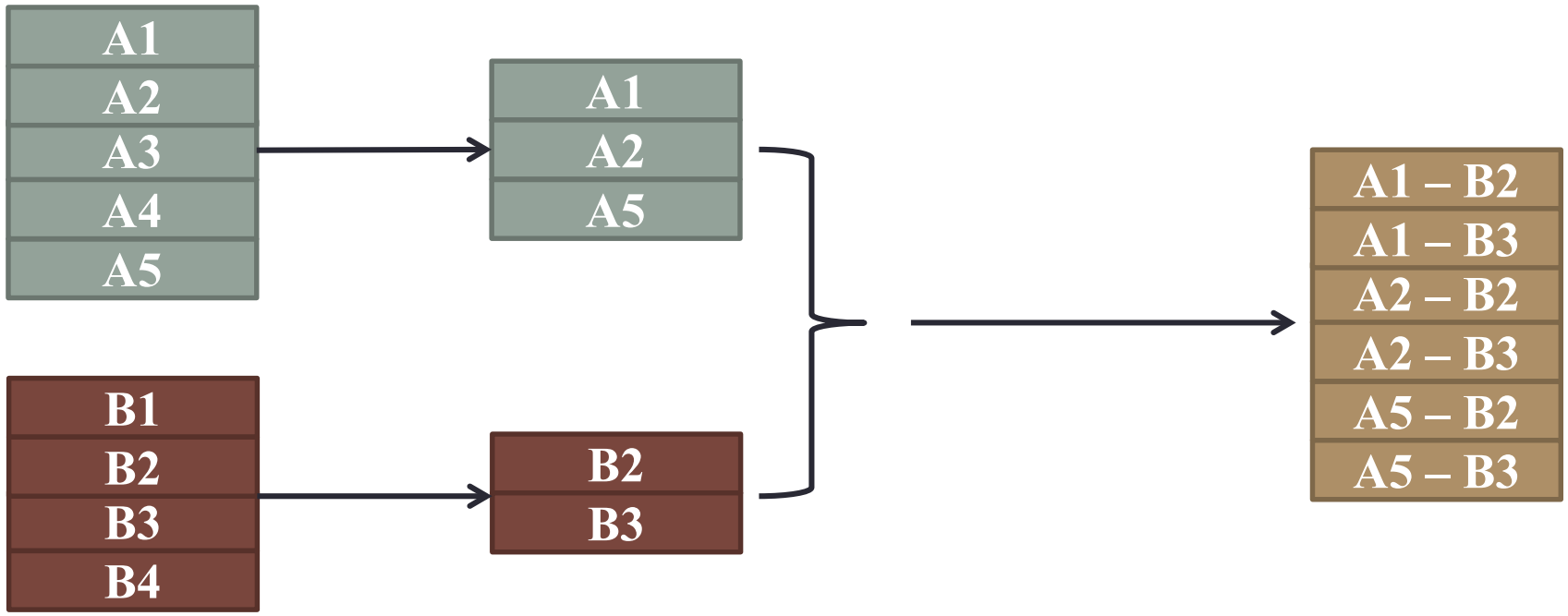
Pre-selection based on prior knowledge



Our System

Pre-selection based on prior knowledge

Feature generation with all feasible pairs

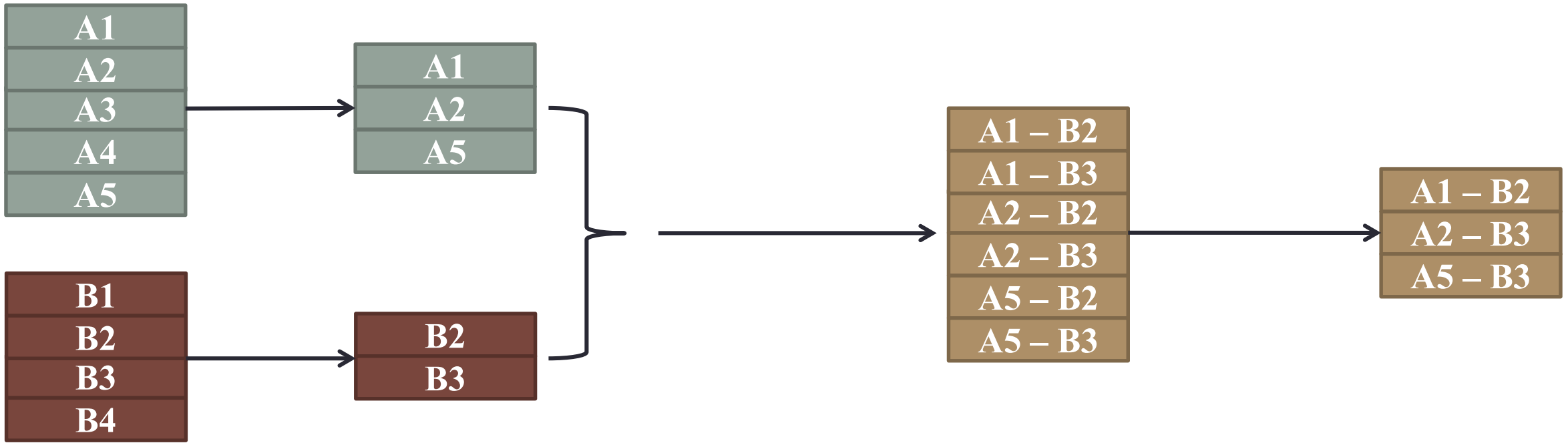


Our System

Pre-selection based on prior knowledge

Feature generation with all feasible pairs

Post-selection by feature importance

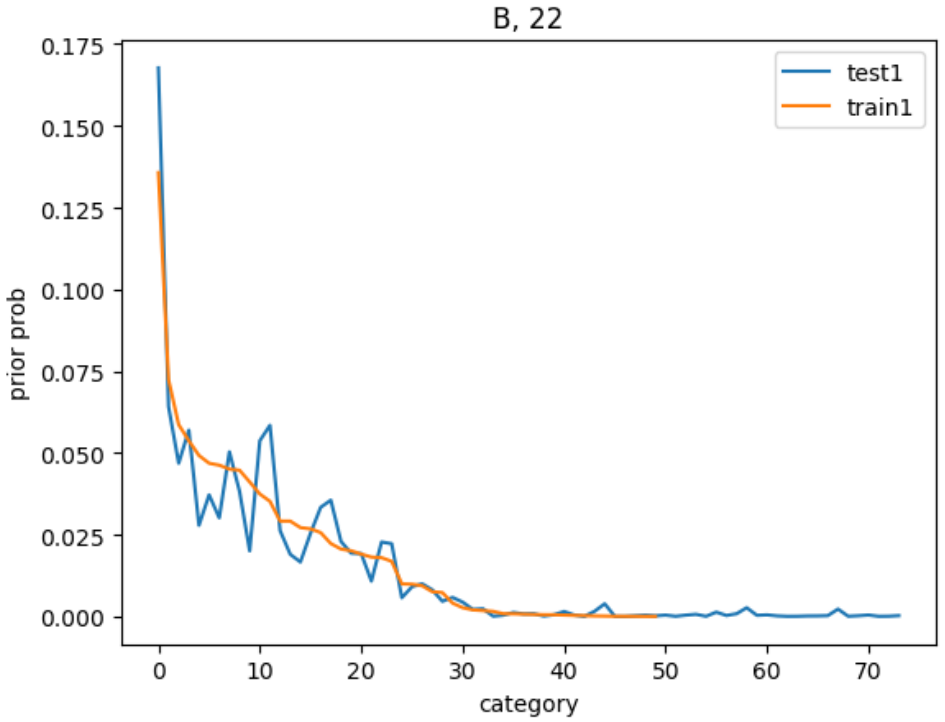


Our System

Concept Drift Adaptation

- **What is concept drift?**
  - $p_{t_0}(X, y) \neq p_{t_1}(X, y)$
  - Real drift: change of $p(\mathrm{y}|X)$
  - Virtual drift: change of $p(X)$
- **How to tackle concept drift?**
  - Real drift: apply a sliding window on the historical data buffer to select the most recent batches and retrain a new GBDT model with them for each test batch
  - Virtual drift: design a streaming encoding scheme for categorical features

Our System



Virtual drift in categorical features

- Many unseen new categories may appear in the new batch
- The frequency of existing categories may change significantly between batches

Our System

- **The problem with separate encoding**
  - Representation of the same category is inconsistent across batches

Batch 1

Sample #	Cat	Encoding
1	A	0.5
2	B	0.25
3	A	0.5
4	C	0.25

Batch 2

Sample #	Cat	Encoding
1	B	0.5
2	A	0.25
3	C	0.25
4	B	0.5

Batch 3

Sample #	Cat	Encoding
1	A	0.75
2	B	0.25
3	A	0.75
4	A	0.75

Our System

- **The problem with separate encoding**
  - Representation of the same category is inconsistent across batches

- **Solution: streaming encoding**
  - Maintain a streaming encoder and update it in an online fashion to capture the drifting trend of $p(X)$
  - Encode the training set and test set simultaneously to achieve a consistent representation of categorical features in each test stage

Our System

Batch 1

Sample #	Cat	Encoding
1	A	0.5
2	B	0.25
3	A	0.5
4	C	0.25

Mapping function of streaming encoder

Category	Batch 1
A	2 / 4 = 0.5
B	1 / 4 = 0.25
C	1 / 4 = 0.25

Our System

Batch 1

Sample #	Cat	Encoding
1	A	0.375
2	B	0.375
3	A	0.375
4	C	0.25

Batch 2

Sample #	Cat	Encoding
1	B	0.375
2	A	0.375
3	C	0.25
4	B	0.375

Mapping function of streaming encoder

Category	Batch 1	Batch 2
A	2 / 4 = 0.5	3 / 8 = 0.375
B	1 / 4 = 0.25	3 / 8 = 0.375
C	1 / 4 = 0.25	2 / 8 = 0.25

Our System

Batch 1

Sample #	Cat	Encoding
1	A	0.5
2	B	0.33
3	A	0.5
4	C	0.17

Batch 2

Sample #	Cat	Encoding
1	B	0.33
2	A	0.5
3	C	0.17
4	B	0.33

Batch 3

Sample #	Cat	Encoding
1	A	0.5
2	B	0.33
3	A	0.5
4	A	0.5

Mapping function of streaming encoder

Category	Batch 1	Batch 2	Batch 3
A	2 / 4 = 0.5	3 / 8 = 0.375	6 / 12 = 0.5
B	1 / 4 = 0.25	3 / 8 = 0.375	4 / 12 = 0.33
C	1 / 4 = 0.25	2 / 8 = 0.25	2 / 12 = 0.17

Our System

Other Components

- **Resource management**
  - Automatically tune some key configurations in the system to adaptively satisfy the time budget and memory constraints on different datasets
- **Model selection**
  - Use GBDT as the only candidate model
  - Improve efficiency without loss on performance
- **Hyperparameter optimization**
  - Portfolio search + Bayesian optimization in local search space
  - Improve efficiency

Results

Second place in the NeurIPS 2018 AutoML Challenge

Team Name	Set 1	Set 2	Set 3	Set 4	Set 5	Average
autodidact.ai	2	4	1	2	2	2.2
Meta_Learners	**3**	**1**	**2**	**1**	**5**	**2.4**
GrandMasters	4	6	4	3	4	4.2

Results

- Validate the effect of the novel components proposed in our system by gradually adding them to a vanilla model

Dataset	Vanilla model	Frequency encoding separately	Streaming encoding	High-order feature combination
A	0.4598	0.4641	0.5142	0.5488
B	0.1936	0.2995	0.3020	0.3284
C	0.4586	0.5278	0.5114	0.5794
D	0.3428	0.4704	0.4739	0.5968
E	0.6706	0.7275	0.7379	0.7672

Conclusion

- Propose a gradient boosting decision tree based AutoML system for lifelong machine learning
- Handle common difficulties under the lifelong learning scenario:
  - high-cardinality categorical features
  - varied types of features with complicated dependencies
  - concept drift
- Our system won the second place in the NeurIPS 2018 AutoML Challenge

Conclusion

Lessons learned for AutoML system design

- Tradeoff is a key issue in AutoML system design
  - Performance vs. generalization
  - Automation vs. prior knowledge
  - ……

Conclusion

Lessons learned for AutoML system design

- Tradeoff is a key issue in AutoML system design
  - Performance vs. generalization
  - Automation vs. prior knowledge
  - ……
- Importance of automated feature engineering
  - Feature representation determines the performance limit of an AutoML system
  - A very complicated problem with great value

Summary

- AutoML for Multimedia
  - --- NAS
  - --- HPO

- Meta-learning for Multimedia
  - --- Meta-learning
  - --- Continual Learning

- Practical Experiences in NeurIPS 2018 AutoML Challenge

Thank you !

Q&A

Contact:

xin_wang@tsinghua.edu.cn