Automated Machine Learning

Aaron Zhao, Imperial College London

Introduction

Introduction

In the last lecture, we explored optimizing neural networks through informed manual adjustments. Today, we'll delve into **Automated Machine Learning** (AutoML) and the process of **Network Architecture Search** (NAS).

The complete AutoML pipeline consists of:

- Data preparation and automated data cleaning
- Feature engineering
- Model selection and Network Architecture Search
- Hyperparameter tuning
- Model evaluation and deployment

Network Architecture Search

Core idea: Can we design the best network architecture purely from observing the data?

We want to pick the optimal architecture $a \in \mathcal{A}$ from a set of architectures $\mathcal{A}$.

At the same time, we want to pick the optimal parameters $w^*(a)$ for the architecture a.

$$\min_{a \in \mathcal{A}} \mathcal{L}_{val}\big(w^*(a), a\big)$$
$$\text{s.t. } w^*(a) = \arg\min_{w} \mathcal{L}_{train}(w, a)$$

Search Space

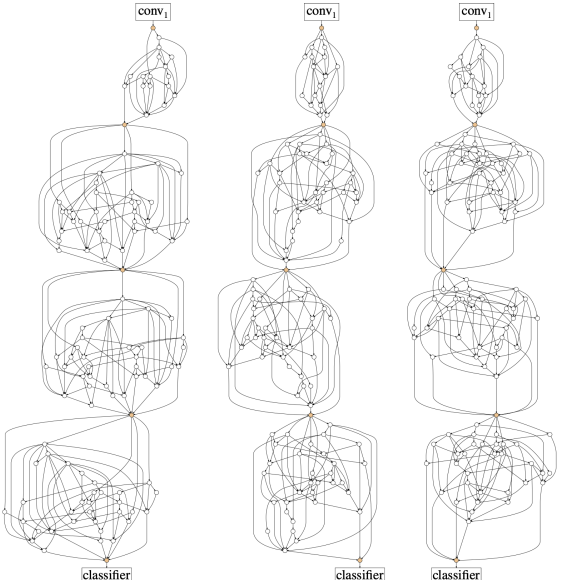
What is a search space?

We need to search through a given set of architectures $\mathcal{A}$, which is known as the **search space**.

Three types of search spaces:

- **Global search space** - Consider all possible elements in the DAG
- **Modular search space** - Search critical components and replicate them
- **Combined search space** - Combine micro and macro architecture design

Global search space



Global search space (ii)

A **global search space** considers all possible elements (search options) in the DAG (Directed Acyclic Graph).

- Extremely large search space - often intractable
- Explores all possible connections and operations
- Requires efficient search strategies

Modular search space

Key approaches:

- Search a critical component, then duplicate that component based on heuristics (e.g., DARTS)
- Use template and backbone with heuristics, search for design options in the backbone (e.g., MobileNet-V3)

The DARTS search space

Cell structure: 6-node DAG (Directed Acyclic Graph)

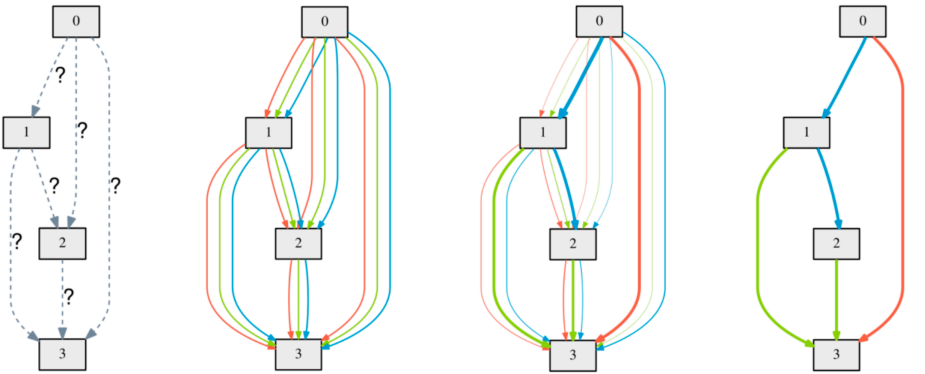
- Two input nodes
- One output node
- Four intermediate data nodes

Operations between nodes:

- conv3x3, conv1x1
- skip connections
- Activation functions (ReLU, sigmoid, tanh, etc.)

Search space size: Roughly 10^9 possible architectures

The DARTS search space (ii)



DARTS: Cell types and transferability

Two types of cells:

- **Normal Cell** - No resolution change
- **Reduction Cell** - Resolution changes with striding/pooling

DARTS: Cell types and transferability (ii)

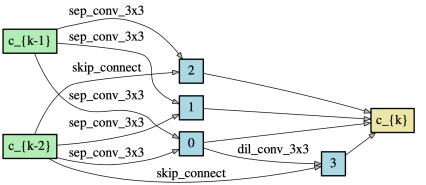


Figure 4: Normal cell learned on CIFAR-10.

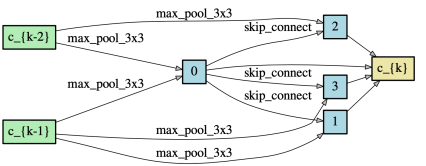


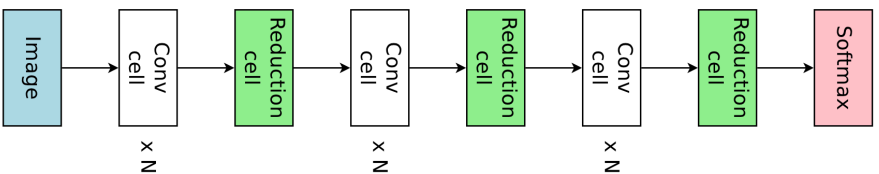
Figure 5: Reduction cell learned on CIFAR-10.

Transfer learning approach:

- Search performed on CIFAR-10 with one normal and one reduction cell

DARTS: Cell types and transferability (iii)

- Architectures learned on CIFAR-10 are generally transferable to other image tasks
- Stack the searched cells multiple times to build bigger networks on ImageNet
- Uses heuristics-defined stacking patterns



Combined search space

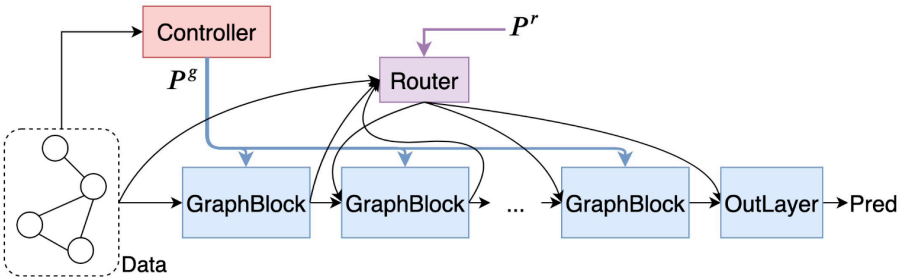
Two-level architecture design:

- **Micro-architecture design:** Design of a single cell using modular search space
- **Macro-architecture design:** How cells should be connected using global search space

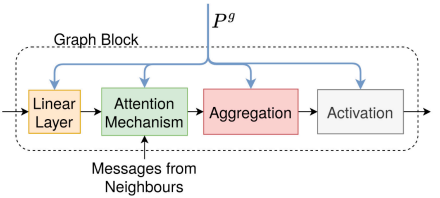
Benefits:

- Decomposition makes the total search space smaller and more tractable
- Balances flexibility with computational efficiency

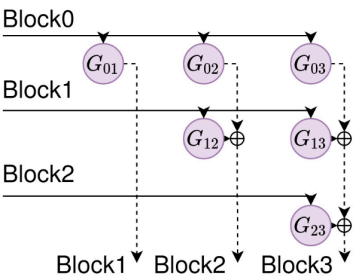
Combined search space (ii)



Combined search space (iii)



,



,

Search Strategies

What is a search strategy?

The core algorithm used to conduct the search within the search space.

Four main approaches:

- **Reinforcement learning** - Controller learns to generate architectures
- **Gradient-based optimization** - Differentiable architecture search
- **Evolutionary algorithm** - Population-based search with mutation and crossover
- **Performance estimators** - Predict performance without full training

Reinforcement Learning based NAS (NASNet)

Setup:

- A controller (parameterized by θ_c) performs actions $(a_0, a_1, ..., a_{T-1})$ to design a child network
- Child network achieves accuracy R on held-out validation dataset

Objective: The controller optimizes the expected reward:

$$J(\theta_c) = E_{P(\{a_0,...,a_{T-1}\} \mid \theta_c)}[R]$$

Optimization: Since reward R is non-differentiable, we use policy gradient:

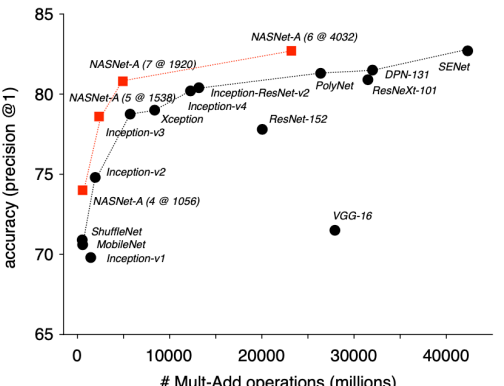
$$J(\theta_c) = \frac{1}{M} \sum_{k=0}^{M-1} \sum_{t=0}^{T-1} \log P(a_t \mid \{a_{t-1}, ..., a_0\}; \theta_c) R$$

where M is the number of architectures sampled per batch and T is the number of hyperparameters to predict.

Key characteristics:

Reinforcement Learning based NAS (NASNet) (ii)

- Differs from standard RL (only start and finish states)
- Requires full training run to obtain R - **very expensive!**



Gradient-based NAS (DARTS)

Problem with RL-based NAS: Hard for controller to receive gradients related to child network accuracy.

Solution: Make the operators differentiable!

Key idea: Associate each operator with a trainable parameter.

Let O be a set of candidate operators (e.g., convolution, max pool, etc.). To make a continuous search space, relax the discrete categorical distribution:

$$\overline{o}^{(i,j)}(x) = \sum_{o \in O} \frac{\exp\left(\alpha_o^{(i,j)}\right)}{\sum_{o' \in O} \exp\left(\alpha_{o'}^{(i,j)}\right)} o(x)$$

Interpretation:

- Operation mixing weights for nodes (i, j) parameterized by vector $\alpha^{(i,j)}$ of dimension $|O|$
- Weight each operator using trainable α values (softmax over operations)

Gradient-based NAS (DARTS): Optimization

Bi-level optimization procedure:

1. Update architecture α by descending:

$$\nabla_\alpha \mathcal{L}_{\text{val}}(w - \xi \nabla_w \mathcal{L}_{\text{train}}(w, \alpha), \alpha)$$

   (where $\xi = 0$ for first-order optimization)

2. Update weights w by descending:

$$\nabla_w \mathcal{L}_{\text{train}}(w, \alpha)$$

3. Derive final architecture based on learned α

Benefits:

- Dual-level optimization using SGD
- **Significant time reduction** - search on cell-based space only once
- No need to evaluate each child network separately

Gradient-based NAS (DARTS): Optimization (ii)

Architecture	Test Error (%)		Params	+×	Search Cost	Search
	top-1	top-5	(M)	(M)	(GPU days)	Method
Inception-v1 (Szegedy et al., 2015)	30.2	10.1	6.6	1448	–	manual
MobileNet (Howard et al., 2017)	29.4	10.5	4.2	569	–	manual
ShuffleNet 2× ($g = 3$) (Zhang et al., 2017)	26.3	–	∼5	524	–	manual
NASNet-A (Zoph et al., 2018)	26.0	8.4	5.3	564	2000	RL
NASNet-B (Zoph et al., 2018)	27.2	8.7	5.3	488	2000	RL
NASNet-C (Zoph et al., 2018)	27.5	9.0	4.9	558	2000	RL
AmoebaNet-A (Real et al., 2018)	25.5	8.0	5.1	555	3150	evolution
AmoebaNet-B (Real et al., 2018)	26.0	8.5	5.3	555	3150	evolution
AmoebaNet-C (Real et al., 2018)	24.3	7.6	6.4	570	3150	evolution
PNAS (Liu et al., 2018a)	25.8	8.1	5.1	588	∼225	SMBO
DARTS (searched on CIFAR-10)	26.7	8.7	4.7	574	4	gradient-based

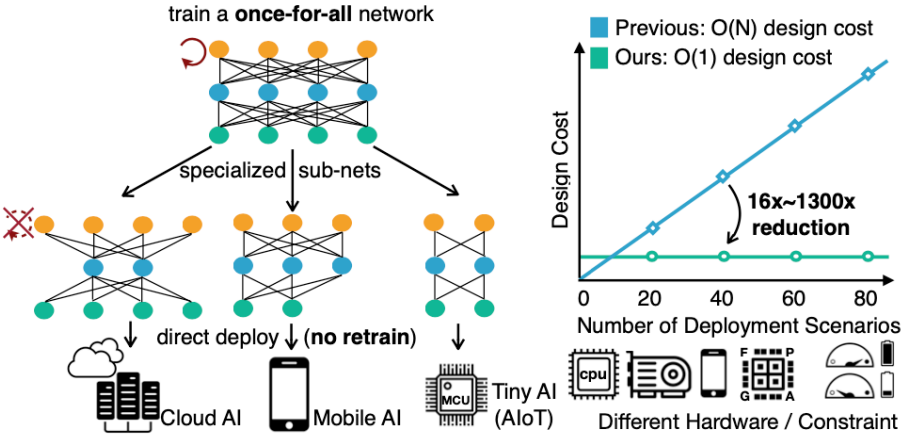
Evolutionary Algorithm (OFA)

Limitations of gradient-based methods:

- Large GPU VRAM usage (single-path method)
- Ranking reliability on proxy datasets
- Gradient interference issues (PC-DARTS)
- No awareness of deployment device and scenarios

Evolutionary Algorithm (OFA): Supernet

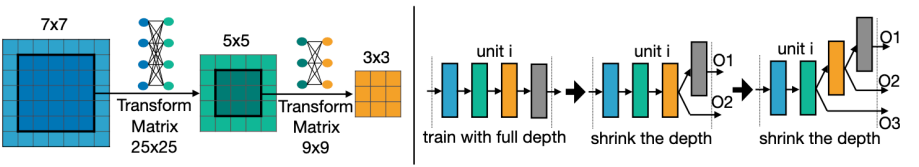
Key idea: Train a supernet and subsample it!



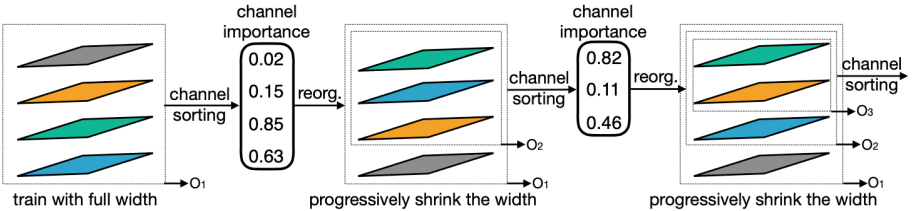
OFA: Supernet design

Design a large supernet that supports sub-sampling in various dimensions:

- **Kernel sizes** - Different receptive fields
- **Channel numbers** - Network width variations
- **Layer depths** - Network depth variations



OFA: Supernet design (ii)



OFA: Evolutionary search algorithm

Evolutionary algorithm steps:

1. **Initialization and Evaluation:** Generate random population and evaluate performance
2. **Selection:** Select individuals based on fitness scores
3. **Crossover:** Pair selected individuals and exchange structure parts to create offspring (recombination/mating)
4. **Mutation:** Randomly alter offspring to introduce variability and prevent local optima
5. **Replacement:** Update population with new offspring, replacing least fit individuals
6. Loop back to evaluation step

OFA: Hardware-aware search

Hardware-aware optimization:

- Perform search with different hardware targets
- Build fitness score using both latency and accuracy
- Optimize for deployment-specific constraints

OFA: Hardware-aware search (ii)

(a) 4.1ms latency on Xilinx ZU3EG (batch size = 1).

(b) 10.9ms latency on Intel Xeon CPU (batch size = 1).

(c) 14.9ms latency on NVIDIA 1080Ti (batch size = 64).

Speedy Performance Estimator

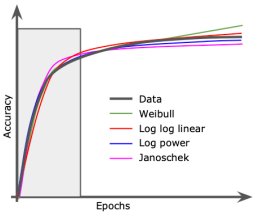
All previous methods require training one or more networks.

Can we predict performance without or with minimal training?

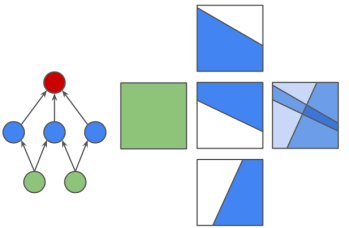
Three approaches:

1. **Learning curve extrapolation** - Extrapolate validation accuracy learning curve via parametric model
2. **Zero-cost proxies** - Assess generalizability with a single forward pass on one minibatch
3. **Subset selection** - Train architecture on a subset of the data

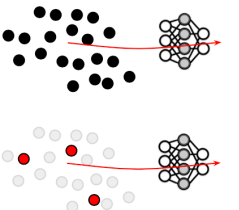
Speedy Performance Estimator (ii)



Learning Curve Extrapolation



Zero-Cost Proxies



Subset Selection

Performance Estimator: Training Speed Estimate

Example performance predictor:

- L = loss function
- $f_{\theta(x)}$ = neural network output with parameters θ
- $\theta_{t,i}$ = parameters after t epochs and i minibatches of SGD

Training Speed Estimate (TSE): After training for T epochs:

$$\text{TSE} = \sum_{t=1}^{T} \left[\frac{1}{B} \sum_{i=1}^{B} L\left(f_{\theta_{t,i}}(x_i), y_i \right) \right]$$

Benefits:

- Use TSE for evaluating sub-networks
- Better rank correlation performance than full training
- Significantly faster evaluation

Alternative approach: Evaluate multiple networks using low-fidelity proxies!

Performance Estimator: Training Speed Estimate (ii)

Method	Search (s)	CIFAR-10		CIFAR-100		ImageNet-16-120	
		validation	test	validation	test	validation	test
(a) NAS-Bench-201							
Non-weight sharing							
REA	12000	91.19±0.31	93.92±0.30	71.81±1.12	71.84±0.99	45.15±0.89	45.54±1.03
RS	12000	90.93±0.36	93.70±0.36	70.93±1.09	71.04±1.07	44.45±1.10	44.57±1.25
REINFORCE	12000	91.09±0.37	93.85±0.37	71.61±1.12	71.71±1.09	45.05±1.02	45.24±1.18
BOHB	12000	90.82±0.53	93.61±0.52	70.74±1.29	70.85±1.28	44.26±1.36	44.42±1.49
Weight sharing							
RSPS	7587	84.16±1.69	87.66±1.69	59.00±4.60	58.33±4.34	31.56±3.28	31.14±3.88
DARTS-V1	10890	39.77±0.00	54.30±0.00	15.03±0.00	15.61±0.00	16.43±0.00	16.32±0.00
DARTS-V2	29902	39.77±0.00	54.30±0.00	15.03±0.00	15.61±0.00	16.43±0.00	16.32±0.00
GDAS	28926	90.00±0.21	93.51±0.13	71.14±0.27	70.61±0.26	41.70±1.26	41.84±0.90
SETN	31010	82.25±5.17	86.19±4.63	56.86±7.59	56.87±7.77	32.54±3.63	31.90±4.07
ENAS	13315	39.77±0.00	54.30±0.00	15.03±0.00	15.61±0.00	16.43±0.00	16.32±0.00
Training-free							
NASWOT (N=10)	3.05	89.14 ± 1.14	92.44 ± 1.13	68.50 ± 2.03	68.62 ± 2.04	41.09 ± 3.97	41.31 ± 4.11
NASWOT (N=100)	30.01	89.55 ± 0.89	92.81 ± 0.99	69.35 ± 1.70	69.48 ± 1.70	42.81 ± 3.05	43.10 ± 3.16
NASWOT (N=1000)	306.19	89.69 ± 0.73	92.96 ± 0.81	69.86 ± 1.21	69.98 ± 1.22	43.95 ± 2.05	44.44 ± 2.10
Random	N/A	83.20 ± 13.28	86.61 ± 13.46	60.70 ± 12.55	60.83 ± 12.58	33.34 ± 9.39	33.13 ± 9.66
Optimal (N=10)	N/A	89.92 ± 0.75	93.06 ± 0.59	69.61 ± 1.21	69.76 ± 1.25	43.11 ± 1.85	43.30 ± 1.87
Optimal (N=100)	N/A	91.05 ± 0.28	93.84 ± 0.23	71.45 ± 0.79	71.56 ± 0.78	45.37 ± 0.61	45.67 ± 0.64
AREA	12000	91.20 ± 0.27	-	71.95 ± 0.99	-	45.70 ± 1.05	-
(b) NATS-Bench SSS							
Non-weight sharing							
REA	12000	90.37±0.20	93.22±0.16	70.23±0.50	70.11±0.61	45.30±0.69	45.54±0.92
RS	12000	90.10±0.26	93.03±0.25	69.57±0.57	69.72±0.61	45.01±0.74	45.42±0.86
REINFORCE	12000	90.25±0.23	93.16±0.21	69.84±0.59	69.96±0.57	45.06±0.77	45.24±1.18
BOHB	12000	90.07±0.28	93.01±0.24	69.75±0.60	69.90±0.60	45.11±0.69	45.56±0.81
NASWOT (N=10)	3.02	88.95 ± 0.88	88.66 ± 0.90	64.55 ± 4.57	64.54 ± 4.70	40.22 ± 3.73	40.48 ± 3.73
NASWOT (N=100)	32.36	89.68 ± 0.51	89.38 ± 0.54	66.71 ± 3.05	66.68 ± 3.25	42.68 ± 2.58	43.11 ± 2.42
NASWOT (N=1000)	248.23	90.14 ± 0.30	93.10 ± 0.31	68.96 ± 1.54	69.10 ± 1.61	44.57 ± 1.48	45.08 ± 1.55

Is NAS a solved problem?

Yes and No...

Remaining challenges:

- Search spaces are normally manually defined
- Low-fidelity methods typically work best at small search spaces
- Many search methods overfit to the search space
- Transferability across domains remains limited

Summary

Network Architecture Search (NAS):

Search Spaces:

- **Global** - All possible DAG elements (large, intractable)
- **Modular** - Search critical components and replicate (e.g., DARTS)
- **Combined** - Micro + macro architecture design

Search Strategies:

- **Reinforcement Learning** - Controller generates architectures (NASNet) - expensive
- **Gradient-based** - Differentiable architecture search (DARTS) - fast
- **Evolutionary** - Population-based with mutation/crossover (OFA) - hardware-aware
- **Performance Estimators** - Predict without full training (TSE, NASWOT) - efficient

Summary (ii)

Key Takeaway: NAS automates architecture design but challenges remain in search space definition, transferability, and avoiding overfitting.