Recent Advances in Neural Architecture Search

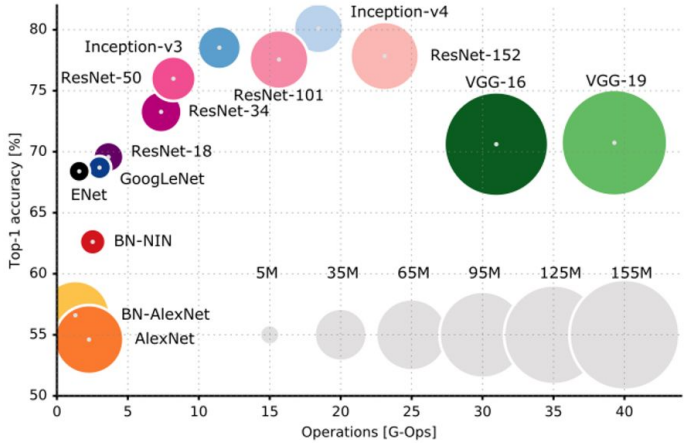
Cho-Jui Hsieh

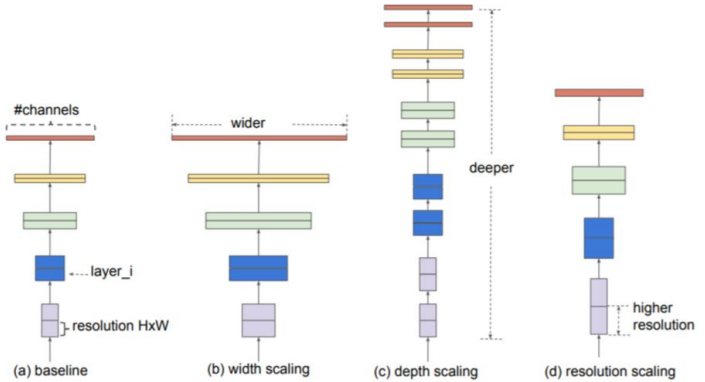
University of California, Los Angeles

Special thanks to Ruochen Wang who helped preparing the slides.

Architecture of Neural Network

- Neural network architecture is important for both accuracy and efficiency





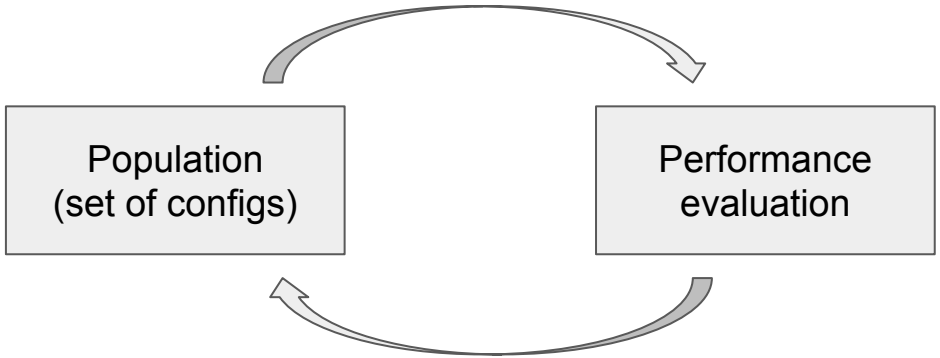
Can we ***automatically*** design architecture?

Outline

- Background (history, benchmarks)
- Significant progresses in the past 3 years:
  - Differentiable Neural Architecture Search
  - Predictor-based NAS with Graph Neural Networks
- Open problems

History of Neural Architecture Search (NAS)

- Early years: only on toy or small-scaled problems
  - Evolutionary algorithms (Miller et al., 89; Schaffer et al., 92; Verbancsics & Harguess, 13)
  - Bayesian optimization (Snoek et al, 12; Domhan et al., 15)



Evolutionary: recombination & mutation
Bayesian: resampling

An early example

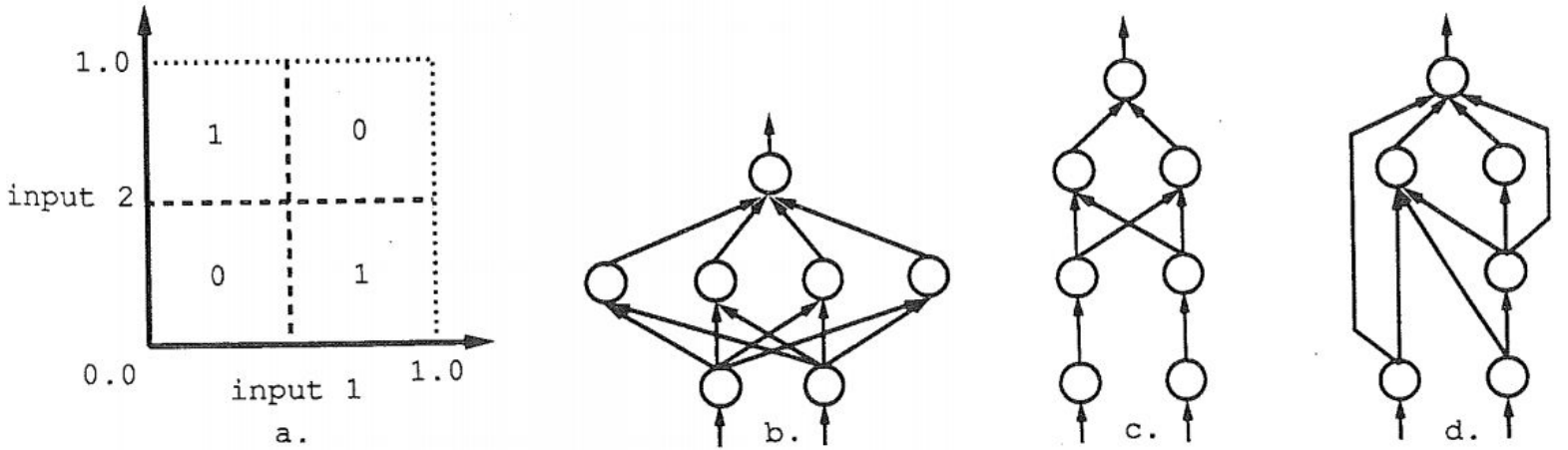


Figure 3. The four-quadrant problem. a. 2-d mapping to be learned. b. Standard 3-layer architectural solution. c. Standard 4-layer architectural solution. d. Typical discovered architectural solution.

From (Miller et al., 1989), NAS for the four-quadrant problem.

Breakthrough in 2016

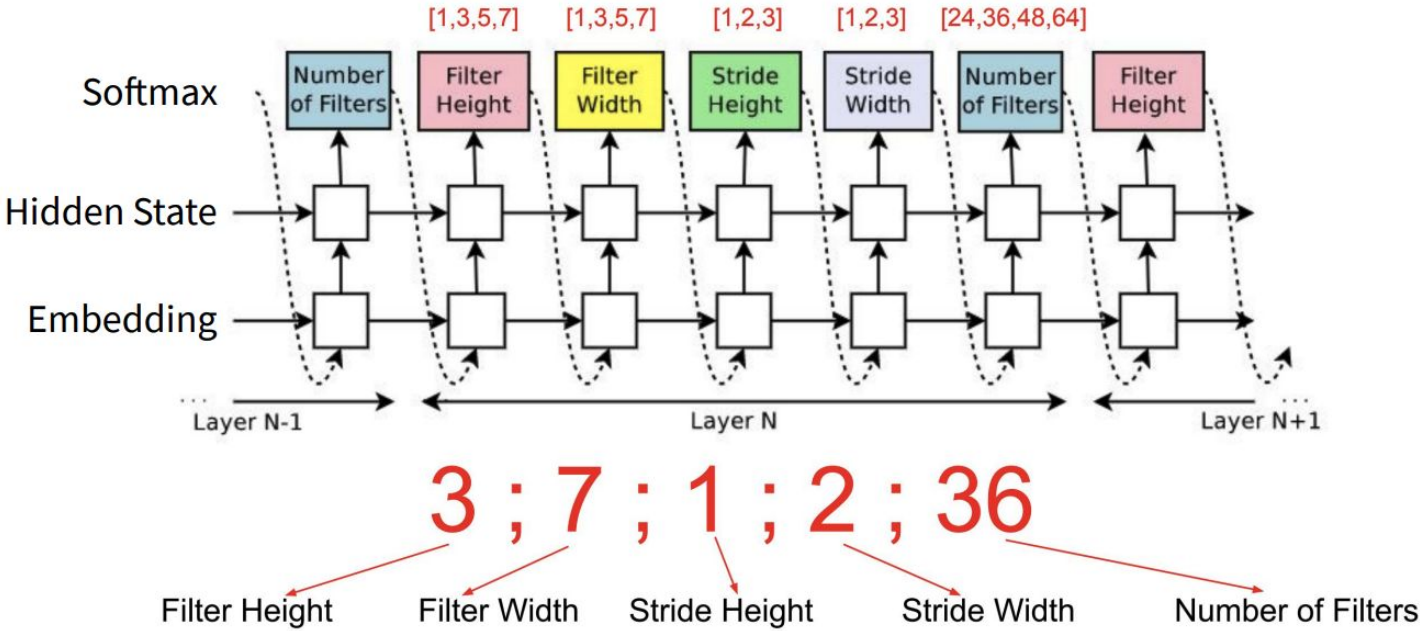
- In 2016, Reinforcement learning (RL) is proposed for NAS
  - A better (structured) representation of search space
  - Learning a controller to generate architectures

[Zoph and Quoc] Neural Architecture Search with Reinforcement Learning. ICLR, 2017.
[Baker, Gupta, Naik, Raskar] Designing Neural Network Architectures using Reinforcement Learning. ICLR, 2017.

- Successful results, but need hundreds of GPU days

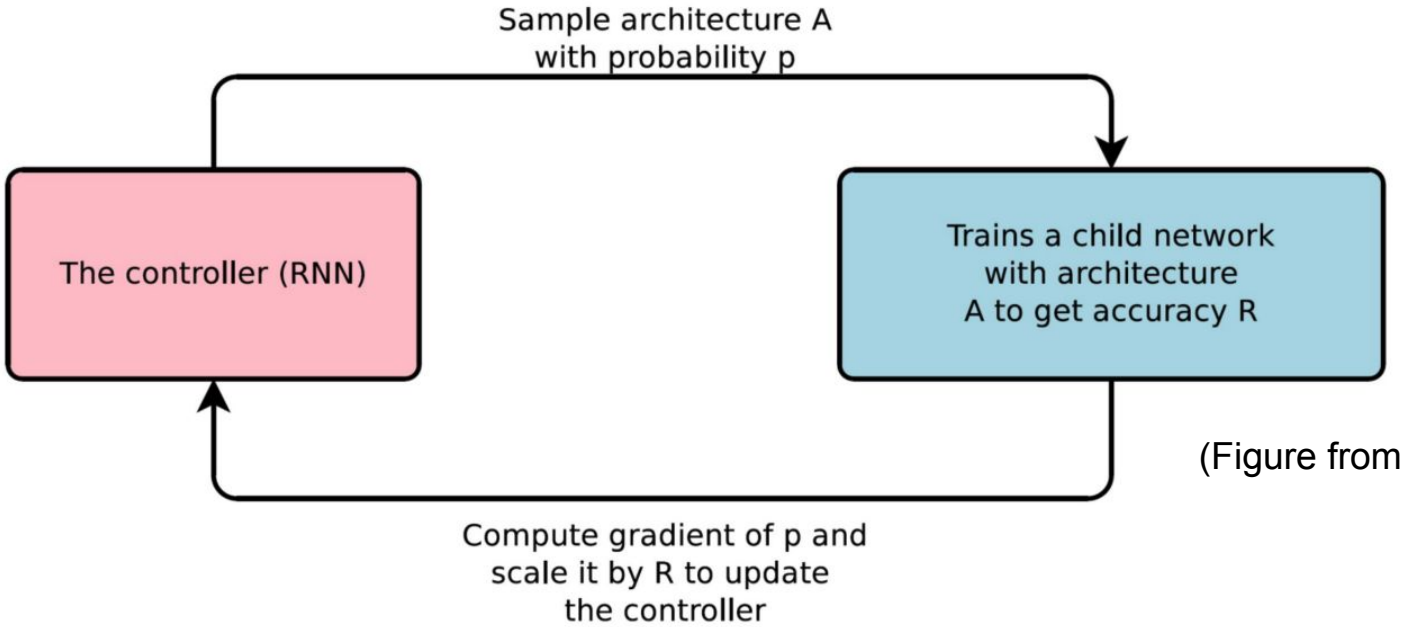
Architecture	Test Error (%)	Search Cost (GPU days)	Search Method
ResNet (He et al., 2016)	4.62	-	manual
DenseNet-BC (Huang et al., 2017)	3.46	-	manual
NAS-RL (Zoph & Le, 2017)	3.65	22,400	RL

NAS with Reinforcement Learning



(Figure from Quoc Le)

Training RNN controller by RL

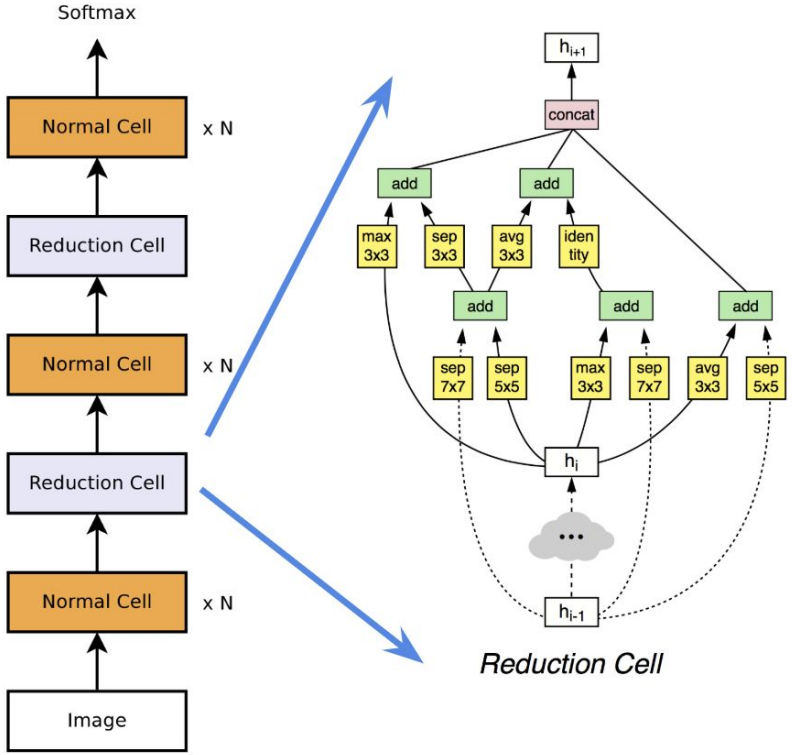


(Figure from Quoc Le)

Extremely slow (>20,000 GPU days)

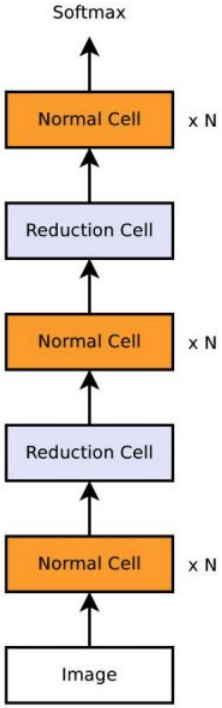
Cell-based Search Space (NASNet)

- Direct search on the global space:
  - Expensive; can't transfer to other datasets
- Cell-based search space:
  - Repeated cells (like ResNet)
  - Can use less blocks in searching
  - Can generalize to more complex datasets by stacking more blocks
- Compared with (Zoph & Le, 2017):
  - Error: 3.65 -> 2.65
  - Search cost: 22,400 -> 2000 GPU days

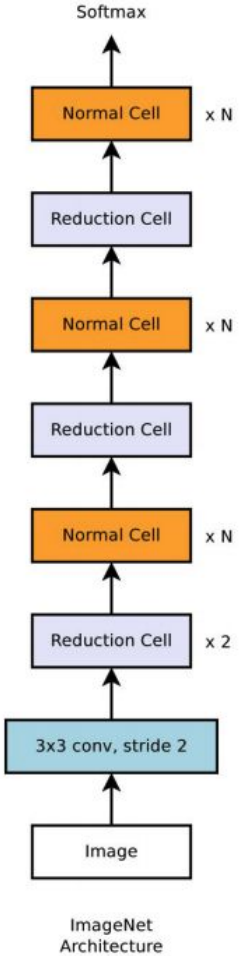


[Zoph, Vasudevan, Shlens, Le] Learning Transferable Architectures for Scalable Image Recognition. In CVPR, 2018.

Generalize from CIFAR-10 to ImageNet



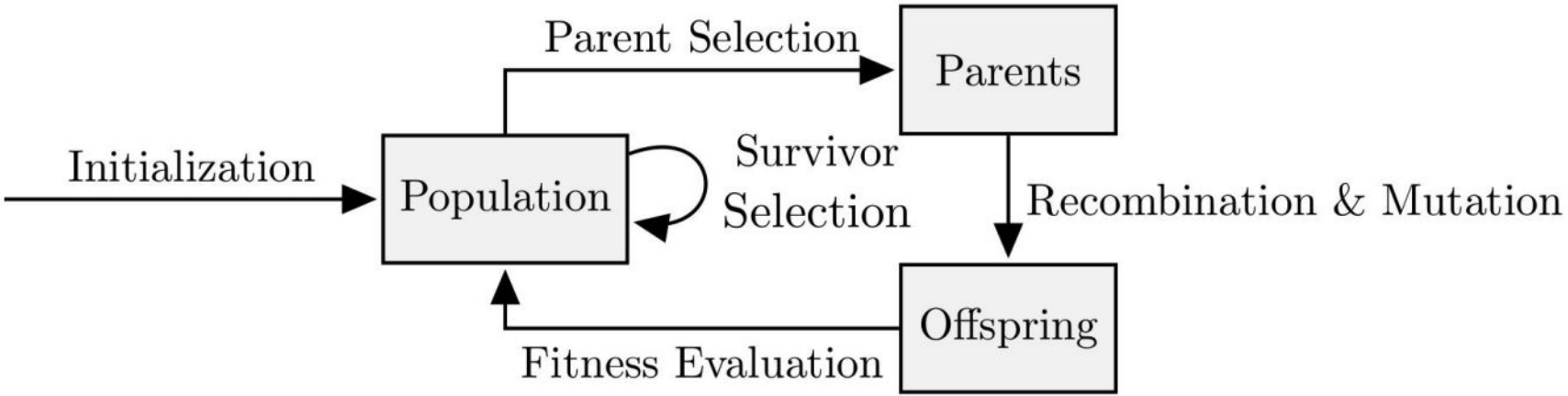
CIFAR10
Architecture



ImageNet
Architecture

Evolutionary Algorithm

Evolutionary algorithm also becomes possible with this search space



(Figure from Wituba et al., 2019)

[Real, Aggarwak, Huang, Le] Regularized Evolution for Image Classifier Architecture Search. AAAI, 2019.

Other RL or evolutionary algorithms proposed before 2018

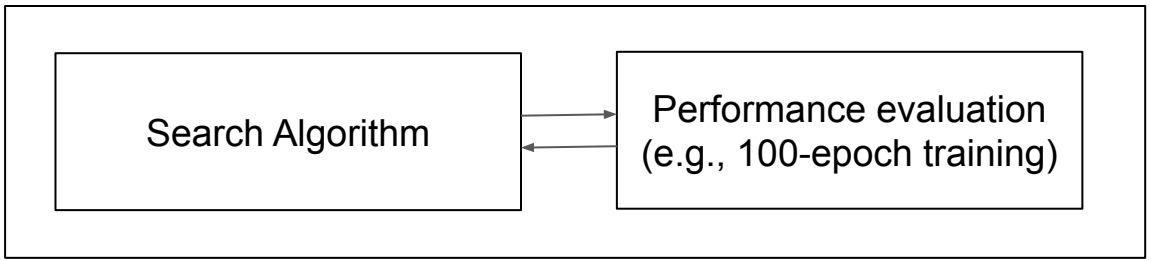
Reference	Error (%)	Params (Millions)	GPU Days	
Baker et al. (2017)	6.92	11.18	100	Reinforcement Learning
Zoph and Le (2017)	3.65	37.4	22,400	
Cai et al. (2018a)	4.23	23.4	10	
Zoph et al. (2018)	3.41	3.3	2,000	
Zoph et al. (2018) + Cutout	2.65	3.3	2,000	
Zhong et al. (2018)	3.54	39.8	96	
Cai et al. (2018b)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

Search typically takes **hundreds of GPU days**! Impractical for typical users.

(Figure from Nikhil Naik)

Running NAS is Hard

- Experiments for NAS are typically time consuming to run



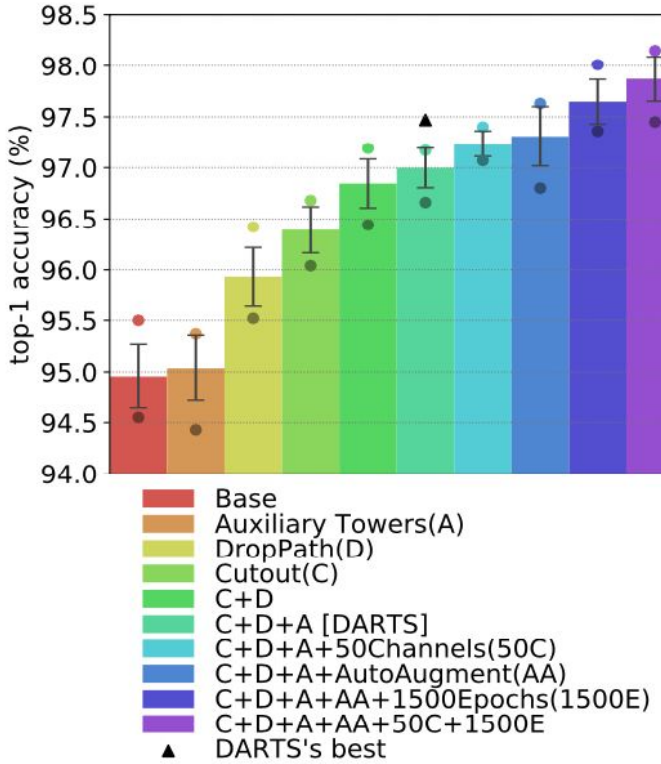
X #runs for hyperparameter tuning

X Multiple runs to compute mean

- RL or evolutionary algorithm often need to evaluate >10,000 configs in a single run

NAS Evaluation is Hard

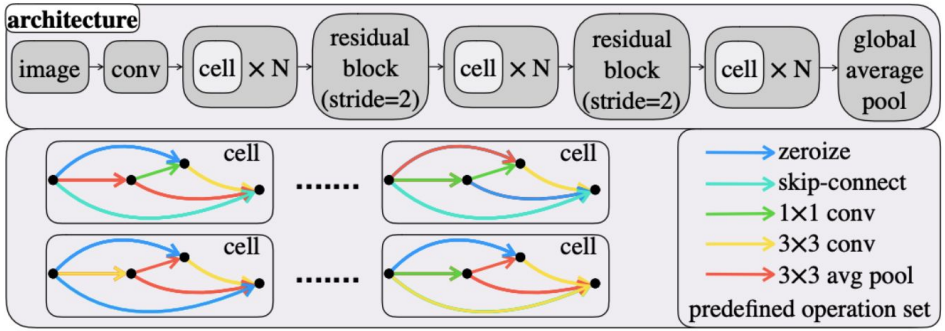
- Best vs Average Performance
- Different training recipes (#epochs, augmentation, hyperparameters)
- Different search space



[Yang, Esperanca, Carlucci] NAS Evaluation is Frustratingly Hard. In ICLR, 2020.

Benchmarks

- NAS-Bench 101 (Ying et al., ICML 2019)
  - 423K architectures evaluated
  - Only architectures with at most 9 edges (not suitable for 1-shot methods)
- NAS-Bench-1Shot1 (Zela et al., 2020), NAS-Bench-201(Dong&Yang, 2020)
  - Precomputed the performance for all the architectures in the space
    => easy for running experiments
  - NAS-Bench-201 Includes CIFAR-10, CIFAR-100, ImageNet 16-120



Main algorithmic Progresses after 2018

- Differentiable Neural Architecture Search

- Predictor-based NAS with Graph Neural Network

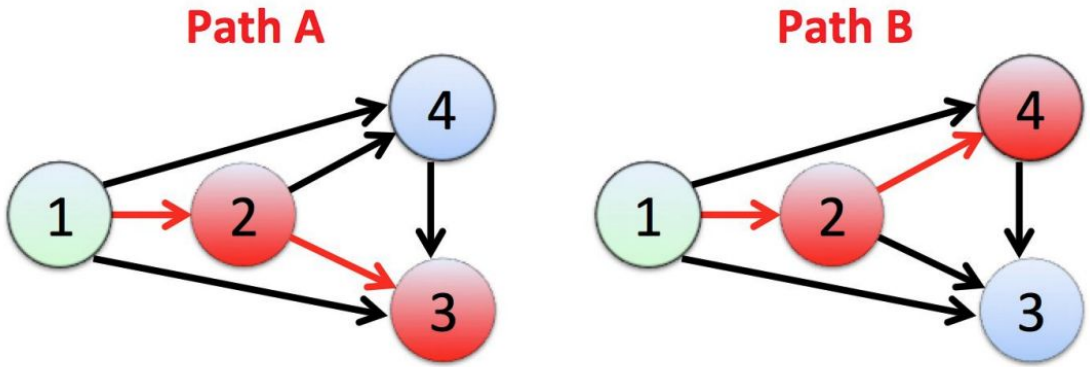
Differentiable Neural Architecture Search

Significantly reduced search time since 2018

Architecture	Test Error (%)	Search Cost (GPU days)	Search Method
DenseNet-BC (Huang et al., 2017)	3.46	-	manual
NAS-RL (Zoph & Le, 2017)	3.65	22,400	RL
NASNet-A (Zoph et al., 2018)	2.65	2000	RL
BlockQNN (Zhong et al., 2018)	3.54	96	RL
AmoebaNet (Real et al., 2019)	3.34 ± 0.06	3150	evolution
Hierarchical GA (Liu et al., 2018)	3.75	300	evolution
GCP (Suganuma et al., 2017)	5.98	15	evolution
DARTS (1st) (Liu et al., 2019)	3.00 ± 0.14	0.4	differentiable
DARTS (2nd) (Liu et al., 2019)	2.76 ± 0.09	1.0	differentiable
SNAS (moderate) (Xie et al., 2019)	2.85 ± 0.02	1.5	differentiable
GDAS (Dong & Yang, 2019)	2.93	0.3	differentiable
ProxylessNAS (Cai et al., 2019)†	2.08	4.0	differentiable
PC-DARTS (Xu et al., 2020)	2.57 ± 0.07	0.1	differentiable
NASP (Yao et al., 2019)	2.83 ± 0.09	0.1	differentiable
SDARTS-ADV (Chen & Hsieh, 2020)	2.61 ± 0.02	1.3	differentiable
DrNAS (Chen et al., 2019)	2.46 ± 0.03	0.6‡	differentiable
DARTS+PT (Wang et al., 2020)	2.61 ± 0.08	0.8	differentiable

Can run on a single GPU machine!

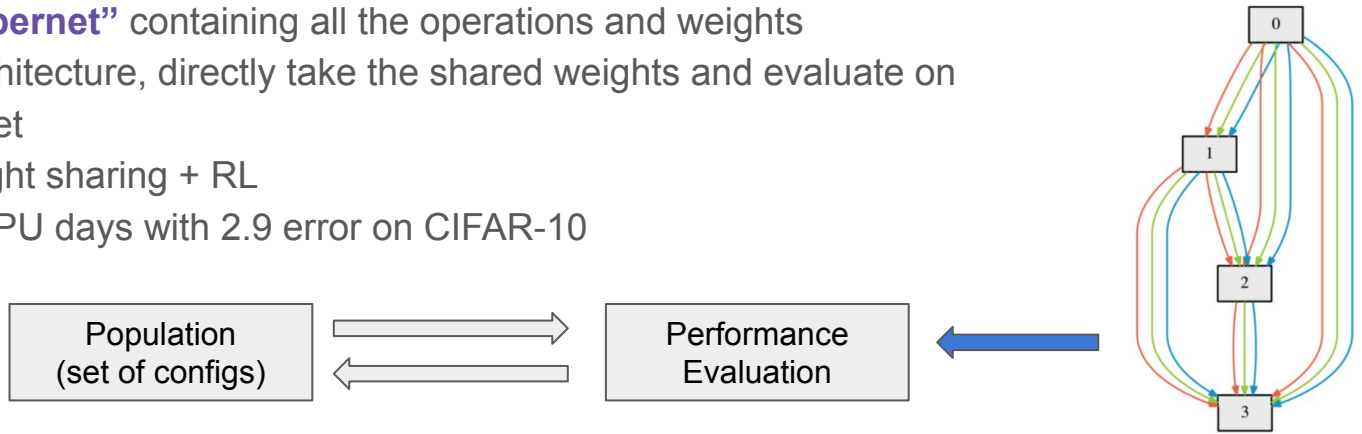
Concept of Weight Sharing



- Models defined by Path A and Path B should be trained separately
- Can we assume Path A and Path B share the same weight at 1->2?
  - **Weight Sharing!**
  - Avoid retraining for each new architecture

Concept of Weight Sharing

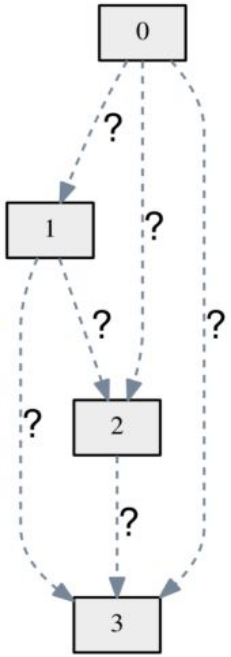
- Supernet: ensemble of many architectures
- All the architectures share the same w (**weight sharing**)
- **Weight sharing** can be directly used to speed up Performance Evaluation in other NAS methods
  - Train a "**supernet**" containing all the operations and weights
  - For any architecture, directly take the shared weights and evaluate on validation set
  - ENAS: weight sharing + RL
    - 0.5 GPU days with 2.9 error on CIFAR-10



Supernet

[Pham, Guan, Zoph, Le, Dean] Efficient Neural Architecture Search via Parameter Sharing. ICML, 2018.

Can we directly obtain the final architecture through supernet training?



Each edge is chosen from a pool of operations:

Conv3x3, Conv5x5, Conv7x7, skip_connect, max_pool, avg_pool, zero, noise, ...

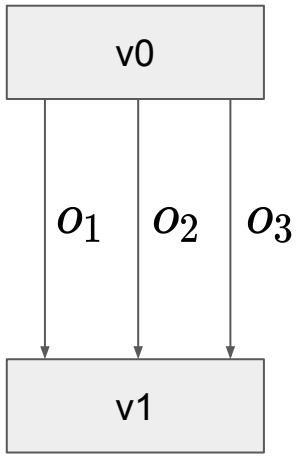
One operation per edge => a **discrete problem**

Continuous Relaxation

- For simplicity, assume 3 operations o_1 : Conv3x3, o_2 : skip_connect, o_3 : Zero
- Assume each edge is a mixed of three operations:

$$v_1 = \alpha_1 o_1(v_0) + \alpha_2 o_2(v_0) + \alpha_3 o_3(v_0)$$

Weight of each operation



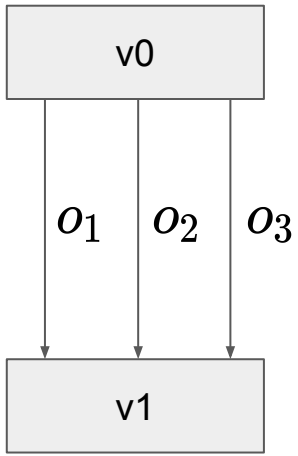
[Liu, Simonyan, Yang] DARTS: Differentiable Architecture Search. In ICLR 2019.

Continuous Relaxation

- For simplicity, assume 3 operations o_1 : Conv3x3, o_2 : skip_connect, o_3 : Zero
- Assume each edge is a mixed of three operations:

$$v_1 = \alpha_1 o_1(v_0) + \alpha_2 o_2(v_0) + \alpha_3 o_3(v_0)$$

Weight of each operation



- Can use softmax to ensure the weights form a prob. distribution

$$v_{\text{out}} = \sum_o \frac{\exp \alpha_o}{\sum_{o'} \exp \alpha_{o'}} o(v_{\text{in}})$$

[Liu, Simonyan, Yang] DARTS: Differentiable Architecture Search. In ICLR 2019.

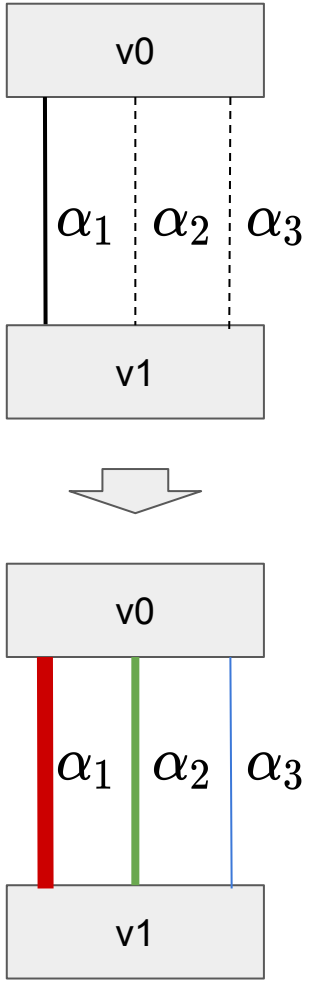
Continuous Relaxation

- Final architecture: $[\alpha_1, \alpha_2, \alpha_3]$ is a one-hot vector

- Relax to continuous values in the search phase
  => Bi-level optimization for finding α

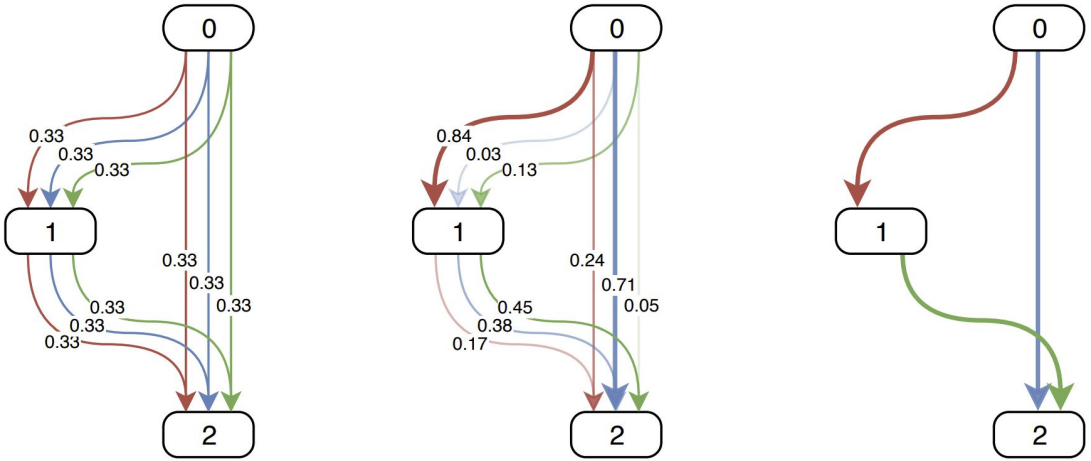
$$\min_{\alpha} L_{\text{val}}(w^*(\alpha), \alpha)$$

$$\text{s.t. } w^*(\alpha) = \arg\min_{w} L_{\text{train}}(w, \alpha)$$



Differentiable Neural Architecture Search (DARTS)

- Solve the bi-level optimization problem to obtain (α^*, w^*) **(supernet)**
- Use magnitude of α^* to choose the final architecture



(d) Search start (e) Search end (f) Final cell

(figure from Zela et al., 2020)

How to solve bi-level optimization?

$$\min_{\alpha} L_{\text{val}}(w^*(\alpha), \alpha)$$
$$\text{s.t. } w^*(\alpha) = \arg\min_{w} L_{\text{train}}(w, \alpha)$$

- Iteratively update w and α
- Update w :
  - Time consuming to compute w* exactly => approximate by one SGD step

$$w' \leftarrow w - \eta \nabla_w L_{\text{train}}(w, \alpha)$$

- Update α :
  - First order DARTS: assume w is constant w.r.t. α

$$\alpha \leftarrow \alpha - c \nabla_\alpha L_{\text{val}}(w', \alpha)$$

How to solve bi-level optimization?

- Second order DARTS: ***w'*** is dependent on ***α***, so

$$\nabla_\alpha L_{\text{val}}(w'(\alpha), w) = \nabla_\alpha L_{\text{val}}(w - \eta \nabla_w L_{\text{train}}(w, \alpha), \alpha)$$

$$= \nabla L_{\text{val}}(w', \alpha) - \eta \underbrace{\nabla^2_{\alpha,w} L_{\text{train}}(w, \alpha)}_{\text{Hessian}} \nabla_{w'} L_{\text{val}}(w', \alpha)$$

How to solve bi-level optimization?

- Second order DARTS: ***w'*** is dependent on ***α***, so

$$\nabla_\alpha L_{\text{val}}(w'(\alpha), w) = \nabla_\alpha L_{\text{val}}(w - \eta \nabla_w L_{\text{train}}(w, \alpha), \alpha)$$
$$= \nabla L_{\text{val}}(w', \alpha) - \eta \underbrace{\nabla^2_{\alpha,w} L_{\text{train}}(w, \alpha)}_{\text{Hessian}} \nabla_{w'} L_{\text{val}}(w', \alpha)$$

- Hessian-vector product can be computed with similar complexity as backprop:
  - Finite difference estimation:

$$\nabla^2_{\alpha,w} L_{\text{train}}(w, \alpha) v = \frac{\nabla_\alpha L_{\text{train}}(w+\epsilon v, \alpha) - \nabla_\alpha L_{\text{train}}(w-\epsilon v, \alpha)}{2\epsilon}$$

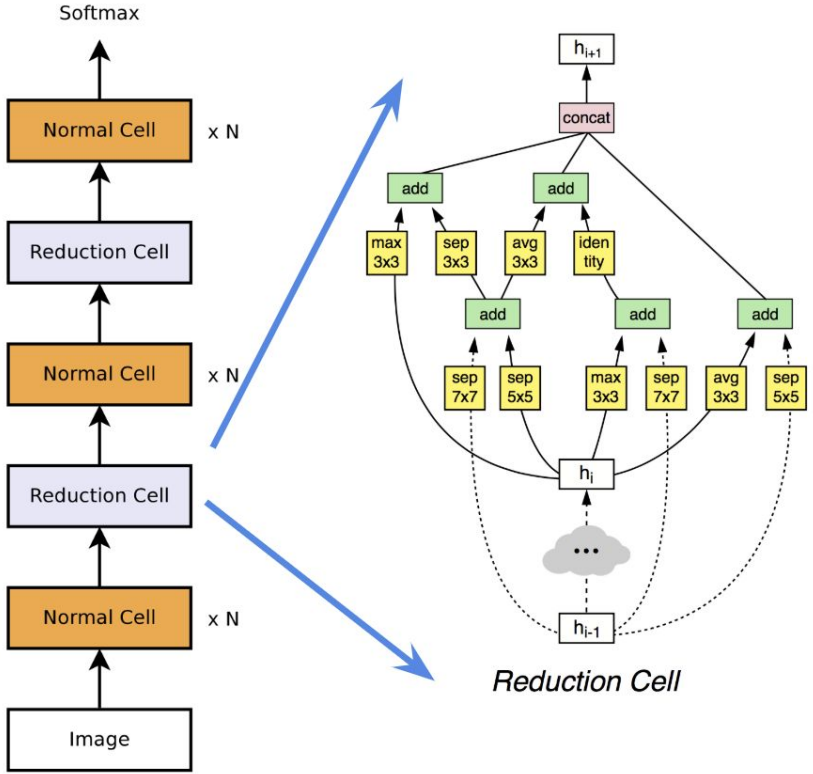
  - Compute with auto-differentiation

Complexity of DARTS

- Time complexity (both first and second order):
  training the supernet **only once**
  - Supernet is a network with K operations with each edge
    => only K times slower than standard training
  - Usually good enough
- Memory complexity (GPU memory):
  - Backprop on all the operations on each edge
    => K times memory consumption
  - Prohibits for many problems

Structured search space

- Using the cell-based search space as NASNet
- Further reduced search cost:
  - Searching on fewer of cells
  - Searching on fewer channels
  - 8 cells/16 channels for search
    ⇒ 20 cells/36 channels during evaluation



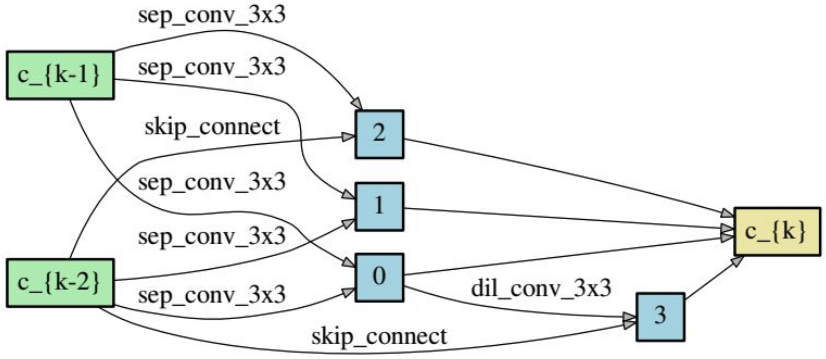
Performance on CIFAR-10

Architecture	Test Error (%)	Params (M)	Search Cost (GPU days)	#ops	Search Method
DenseNet-BC (Huang et al., 2017)	3.46	25.6	–	–	manual
NASNet-A + cutout (Zoph et al., 2018)	2.65	3.3	2000	13	RL
NASNet-A + cutout (Zoph et al., 2018)†	2.83	3.1	2000	13	RL
BlockQNN (Zhong et al., 2018)	3.54	39.8	96	8	RL
AmoebaNet-A (Real et al., 2018)	3.34 ± 0.06	3.2	3150	19	evolution
AmoebaNet-A + cutout (Real et al., 2018)†	3.12	3.1	3150	19	evolution
AmoebaNet-B + cutout (Real et al., 2018)	2.55 ± 0.05	2.8	3150	19	evolution
Hierarchical evolution (Liu et al., 2018b)	3.75 ± 0.12	15.7	300	6	evolution
PNAS (Liu et al., 2018a)	3.41 ± 0.09	3.2	225	8	SMBO
ENAS + cutout (Pham et al., 2018b)	2.89	4.6	0.5	6	RL
ENAS + cutout (Pham et al., 2018b)*	2.91	4.2	4	6	RL
Random search baseline‡ + cutout	3.29 ± 0.15	3.2	4	7	random
DARTS (first order) + cutout	3.00 ± 0.14	3.3	1.5	7	gradient-based
DARTS (second order) + cutout	2.76 ± 0.09	3.3	4	7	gradient-based

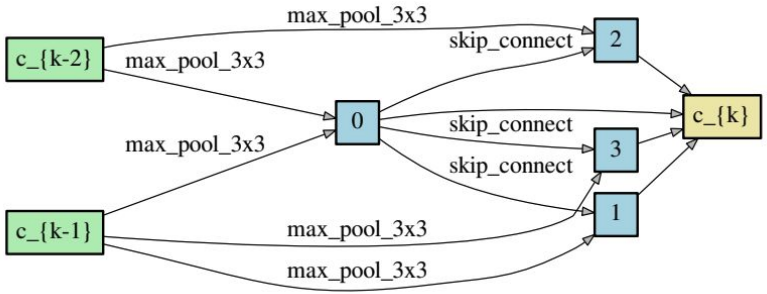
Transfer to ImageNet (mobile setting)

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

Architectures found by DARTS



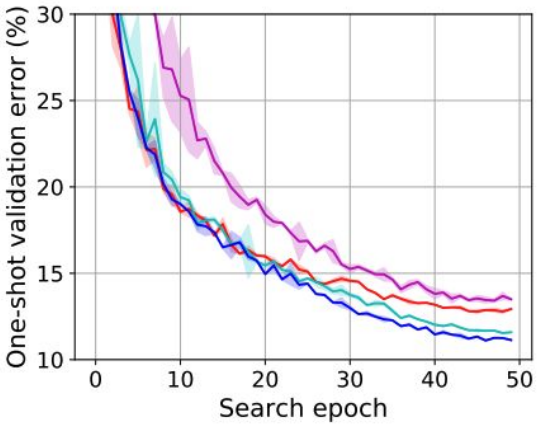
Normal cells on CIFAR-10



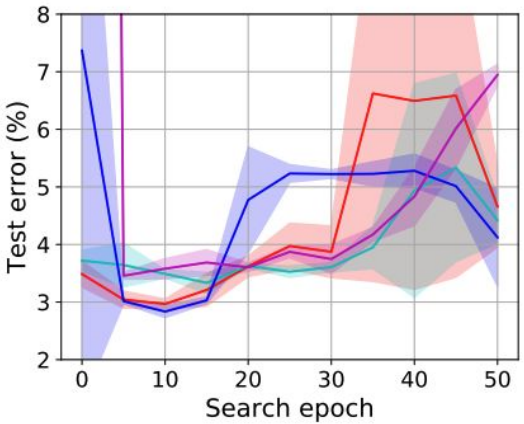
Reduction cells on CIFAR-10

However, DARTS fails in many simple cases

- Space 1: 2 operations per edge (selected from the original DARTS supernet)
- Space 2: 2 operations per edge {Conv3x3, skip_connect}
- Space 3: 3 operations per edge {Conv3x3, skip_connect, Zero}
- Space 4: 2 operations per edge {Conv3x3, Gaussian_noise}



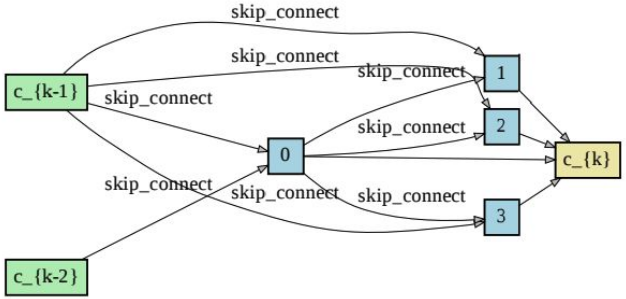
Validation error of supernet



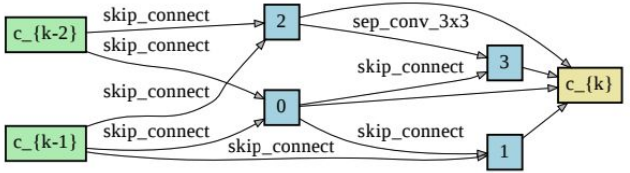
Test error of final architecture

[Zela, Elsken, Saikia, Marrakchi, Brox, Hutter] Understanding and Robustifying Differentiable Architecture Search. In ICLR, 2020.

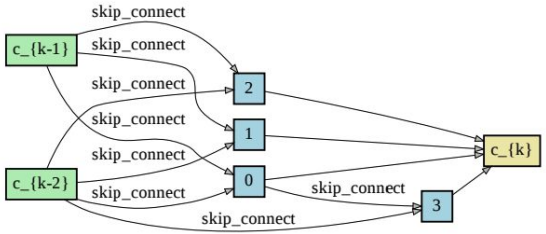
DARTS leads to degenerated solutions in S1-S4



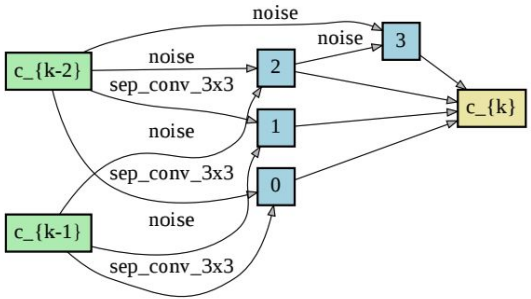
S1



S2



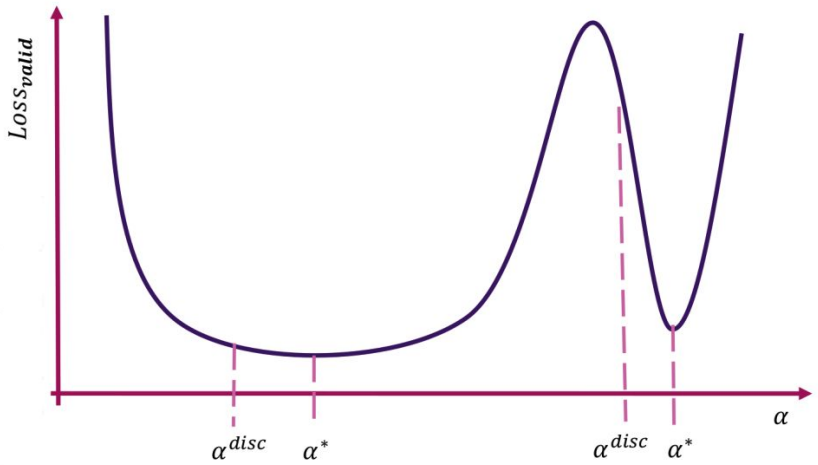
S3



S4

Reason 1: Sharpness of the solution

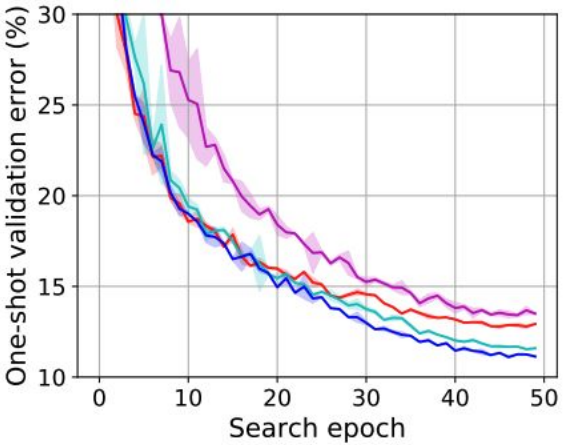
- A good continuous solution doesn't imply a good discrete solution
- Gap between continuous and discrete solutions can be estimated by sharpness
  - Assume α^* is the continuous solution and $\bar{\alpha}$ is the discrete solution
  - Based on Taylor expansion: $L_{\text{val}}(w^*, \bar{\alpha}) \approx L_{\text{val}}(w^*, \alpha^*) + \frac{1}{2}(\bar{\alpha} - \alpha^*)^T H(\bar{\alpha} - \alpha^*)$ where $H = \nabla^2_\alpha L_{\text{val}}(w^*, \alpha^*)$ is the Hessian w.r.t. α
  - Standard DARTS lead to "Sharp solutions" (large Hessian)



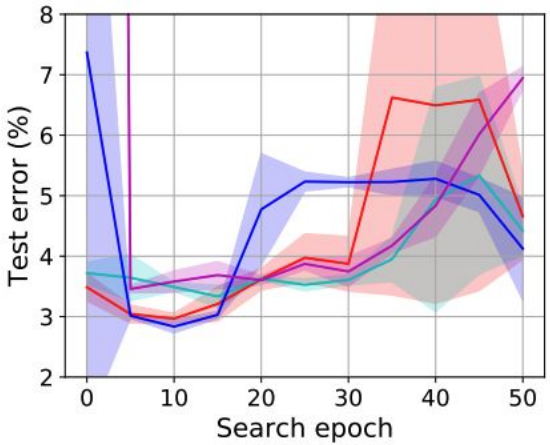
(figure from Zela et al., 2020)

Reason 1: Sharpness of the solution

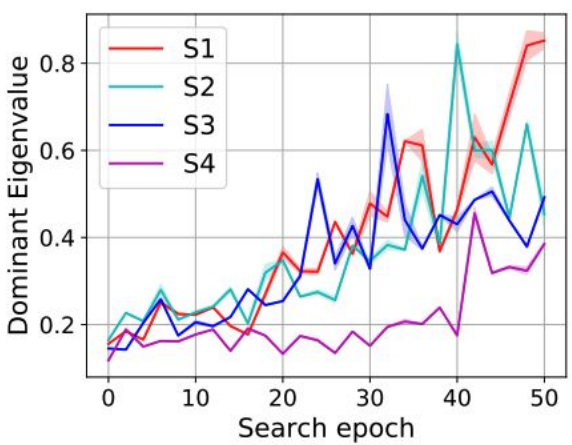
DARTS training leads to sharp local minimums



Validation error of supernet



Test error of final architecture

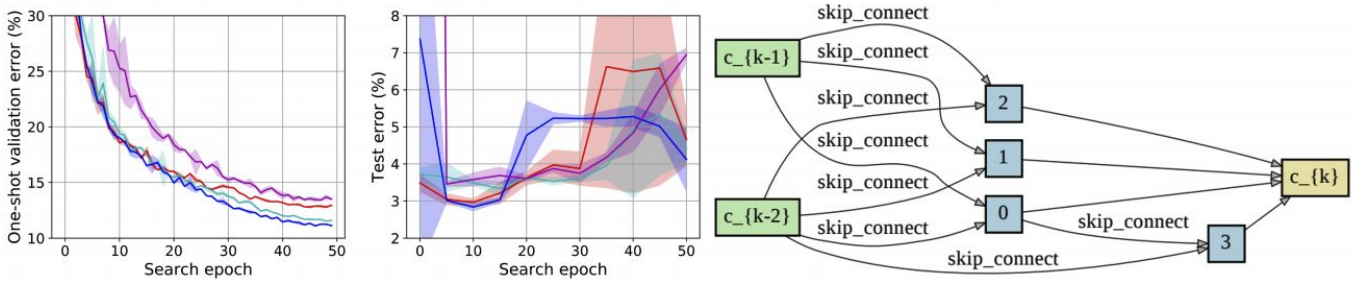


Dominant eigenvalue of Hessian

[Zela, Elsken, Saikia, Marrakchi, Brox, Hutter] Understanding and Robustifying Differentiable Architecture Search. In ICLR, 2020.

Reason 2: Skip connection domination

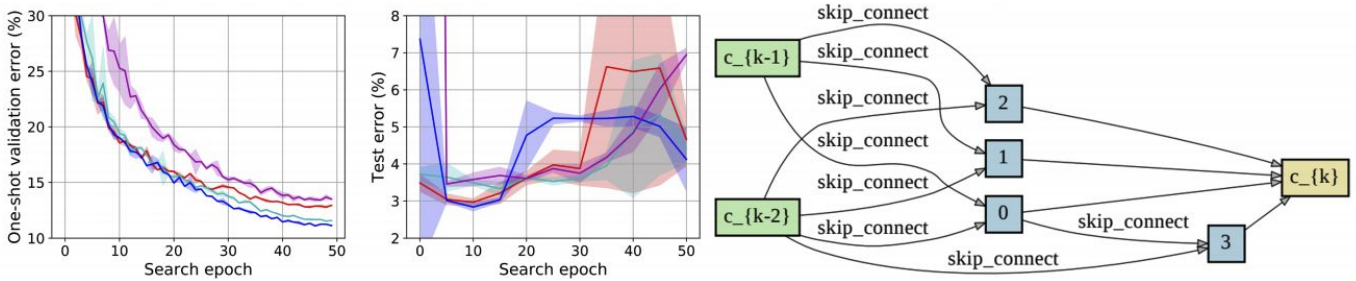
- Supernet accuracy ↑
- Weight for skip connection ↑
- Weight for convolution ↓



[Wang, Cheng, Chen, Tang, Hsieh] Rethinking Architecture Selection in Differentiable NAS. In ICLR, 2021.

Reason 2: Skip connection domination

- Supernet accuracy ↑
- Weight for skip connection ↑
- Weight for convolution ↓



- Formally, we proved that for the optimal supernet, as number of layers goes to infinity, $\alpha_{\text{skip}} \uparrow 1$ and $\alpha_{\text{conv}} \downarrow 0$

[Wang, Cheng, Chen, Tang, Hsieh] Rethinking Architecture Selection in Differentiable NAS. In ICLR, 2021.

Improvements over DARTS

Architecture	Test Error (%)	Search Cost (GPU days)	Search Method
DenseNet-BC (Huang et al., 2017)	3.46	-	manual
NAS-RL (Zoph & Le, 2017)	3.65	22,400	RL
NASNet-A (Zoph et al., 2018)	2.65	2000	RL
BlockQNN (Zhong et al., 2018)	3.54	96	RL
AmoebaNet (Real et al., 2019)	3.34 ± 0.06	3150	evolution
Hierarchical GA (Liu et al., 2018)	3.75	300	evolution
GCP (Suganuma et al., 2017)	5.98	15	evolution
DARTS (1st) (Liu et al., 2019)	3.00 ± 0.14	0.4	differentiable
DARTS (2nd) (Liu et al., 2019)	2.76 ± 0.09	1.0	differentiable
SNAS (moderate) (Xie et al., 2019)	2.85 ± 0.02	1.5	differentiable
GDAS (Dong & Yang, 2019)	2.93	0.3	differentiable
ProxylessNAS (Cai et al., 2019)†	2.08	4.0	differentiable
PC-DARTS (Xu et al., 2020)	2.57 ± 0.07	0.1	differentiable
NASP (Yao et al., 2019)	2.83 ± 0.09	0.1	differentiable
SDARTS-ADV (Chen & Hsieh, 2020)	2.61 ± 0.02	1.3	differentiable
DrNAS (Chen et al., 2019)	2.46 ± 0.03	$0.6^{\ddagger}$	differentiable
DARTS+PT (Wang et al., 2020)	2.61 ± 0.08	0.8	differentiable

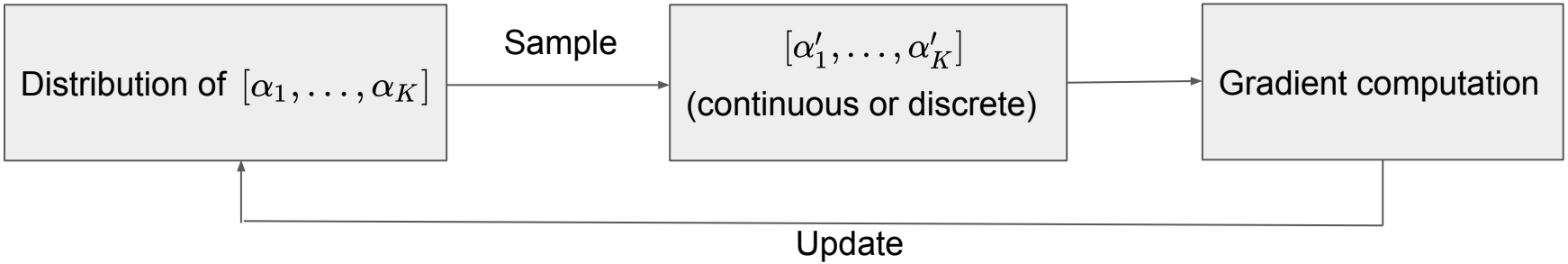
Improvements over DARTS

- Supernet Training
  - Usually aim to make supernet more “discreterizable”
  - Balance exploration and exploitation
- Scalability
  - How to use more blocks in searching?
  - Reduce memory overhead to directly search on larger problems
- Architecture Selection
  - Does architecture weights ($\boldsymbol{\alpha}$) really indicate their importance?

Improved Supernet Training for DARTS

Supernet training: Distribution Learning

- Rethink DARTS as a **distribution learning problem**
  - For each edge, $[\alpha_1, \ldots, \alpha_K]$ defines a distribution over operations
  - We eventually "sample" an architecture from this distribution
  - How to learn $[\alpha_1, \ldots, \alpha_K]$ based on gradient-based optimization?
- Benefits:
  - Performance will be preserved better after discretization
  - Reduced training time in some cases



Gumbel Softmax

- Sampling from a distribution $i \sim \alpha_i / \sum_{i'} \alpha_{i'}$ (can't backprop from i to $\boldsymbol{\alpha}$)

[Xie, Zheng, Liu, Lin] SNAS: Stochastic Neural Architecture Search. In ICLR, 2019.

Gumbel Softmax

- Sampling from a distribution $i \sim \alpha_i / \sum_{i'} \alpha_{i'}$ (can't backprop from i to $\boldsymbol{\alpha}$)
- Gumbel-max: this is equivalent to

$$i = \arg\max_{i'} \{G_{i'} + \log(\alpha_i)\}$$

where each $G_{i'} \sim \text{Gumbel}(0, 1)$

[Xie, Zheng, Liu, Lin] SNAS: Stochastic Neural Architecture Search. In ICLR, 2019.

Gumbel Softmax

- Sampling from a distribution $i \sim \alpha_i / \sum_{i'} \alpha_{i'}$ (can't backprop from i to $\boldsymbol{\alpha}$)
- Gumbel-max: this is equivalent to

$$i = \arg\max_{i'} \{G_{i'} + \log(\alpha_i)\}$$

where each $G_{i'} \sim \text{Gumbel}(0, 1)$

  - Gumbel-softmax: using softmax with temperature annealed to be close to zero

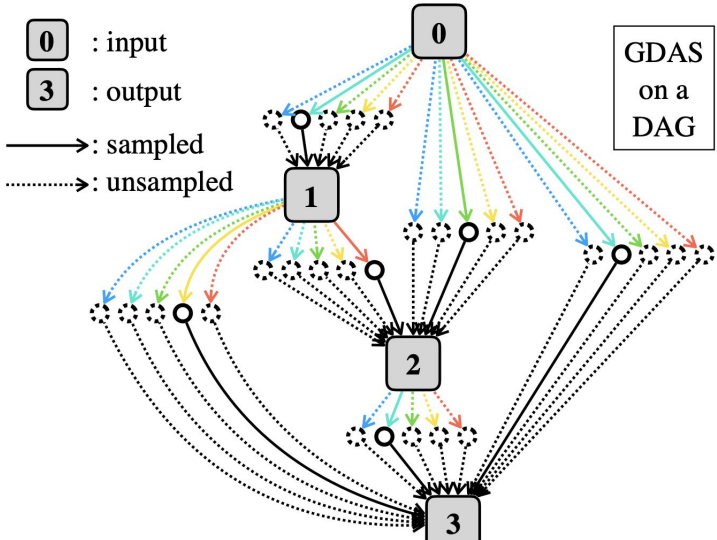
$$z_i = \frac{\exp(G_i + \log(\alpha_i))/\gamma}{\sum_{i'} \exp(G_{i'} + \log(\alpha_{i'}))/\gamma}$$

  - This enables back-propagation to $[\alpha_1, \ldots, \alpha_K]$ (reparameterization trick)
  - SNAS: use Gumbel softmax with annealed temperature in DARTS

[Xie, Zheng, Liu, Lin] SNAS: Stochastic Neural Architecture Search. In ICLR, 2019.

Gumbel Softmax with Discrete Samples

- z_i sampled by Gumbel softmax could be close to one-hot (when temperature is very slow), but not discrete
- Needs to forward-backward all the operations => same cost as DARTS
- GDAS: Straight-through trick to sample discrete variables:
  - Take $\bar{z}_i = \text{one\_hot}(\arg\max_{i'}\{G_{i'} + \log(\alpha_i)\})$
  - Let $\bar{z}_i = z_i + \text{const}$ and stop gradient for const (use discrete variable in forward pass and continuous z_i in the backward pass)



[Dong and Yang] Searching a Robust Neural Architecture in Four GPU Hours. In CVPR, 2019.

Performance of SNAS and GDAS

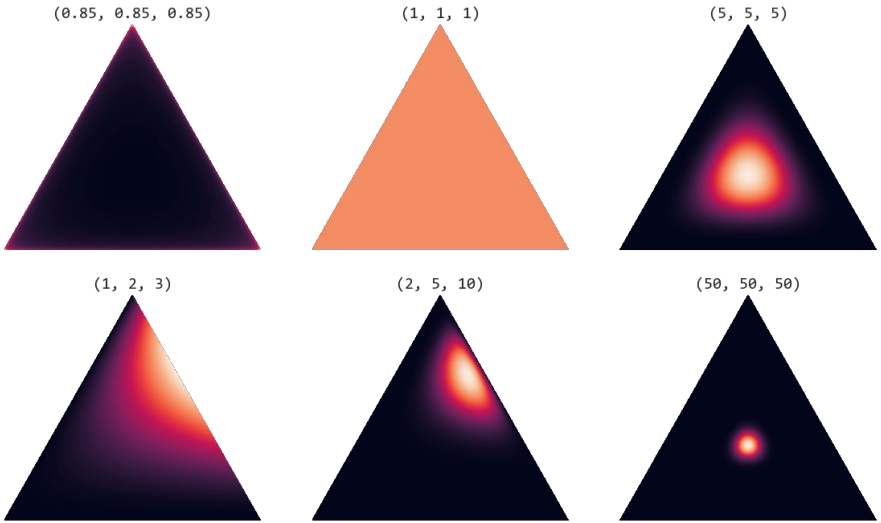
Architecture	Test Error (%)	Search Cost (GPU days)	Search Method
DenseNet-BC (Huang et al., 2017)	3.46	-	manual
NAS-RL (Zoph & Le, 2017)	3.65	22,400	RL
NASNet-A (Zoph et al., 2018)	2.65	2000	RL
BlockQNN (Zhong et al., 2018)	3.54	96	RL
AmoebaNet (Real et al., 2019)	3.34 ± 0.06	3150	evolution
Hierarchical GA (Liu et al., 2018)	3.75	300	evolution
GCP (Suganuma et al., 2017)	5.98	15	evolution
DARTS (1st) (Liu et al., 2019)	3.00 ± 0.14	0.4	differentiable
DARTS (2nd) (Liu et al., 2019)	2.76 ± 0.09	1.0	differentiable
SNAS (moderate) (Xie et al., 2019)	2.85 ± 0.02	1.5	differentiable
GDAS (Dong & Yang, 2019)	2.93	0.3	differentiable
ProxylessNAS (Cai et al., 2019)†	2.08	4.0	differentiable
PC-DARTS (Xu et al., 2020)	2.57 ± 0.07	0.1	differentiable
NASP (Yao et al., 2019)	2.83 ± 0.09	0.1	differentiable
SDARTS-ADV (Chen & Hsieh, 2020)	2.61 ± 0.02	1.3	differentiable
DrNAS (Chen et al., 2019)	2.46 ± 0.03	0.6‡	differentiable
DARTS+PT (Wang et al., 2020)	2.61 ± 0.08	0.8	differentiable

DrNAS: Dirichlet Neural Architecture Search

- Assume architecture parameters $[\alpha_1, \ldots, \alpha_K]$ are sampled from Dirichlet Distribution:

$$[\alpha_1, \ldots, \alpha_K] \sim \mathbf{Dir}([\beta_1, \ldots, \beta_K])$$

- Dirichlet distribution samples from the standard K-1 simplex
  - $\boldsymbol{\beta} \ll 1$ leads to sparse samples with high variance
  - $\boldsymbol{\beta} \gg 1$ leads to dense samples with low variance (for sufficient exploration)



DrNAS: Dirichlet Neural Architecture Search

- DrNAS objective:
  - Point estimation → distribution learning

$$\min_{\beta} E_{q(\alpha|\beta)}[L_{val}(w^*(\alpha), \alpha)] + \lambda d(\beta, \hat{\beta}), \quad s.t.$$

$$w^* = \arg\min_{w} L_{train}(w, \alpha), \quad q(\alpha|\beta) \sim Dir(\beta)$$

  - Gradient computation:

$$\frac{d\alpha_i}{d\beta_j} = -\frac{\frac{\partial F_{Beta}}{\partial \beta_j}(\alpha_j|\beta_j, \beta_{tot} - \beta_j)}{f_{Beta}(\alpha_j|\beta_j, \beta_{tot} - \beta_j)} \times (\frac{\delta_{ij} - \alpha_i}{1 - \alpha_j})$$

  - Architecture selection: magnitude of $\boldsymbol{\beta}$

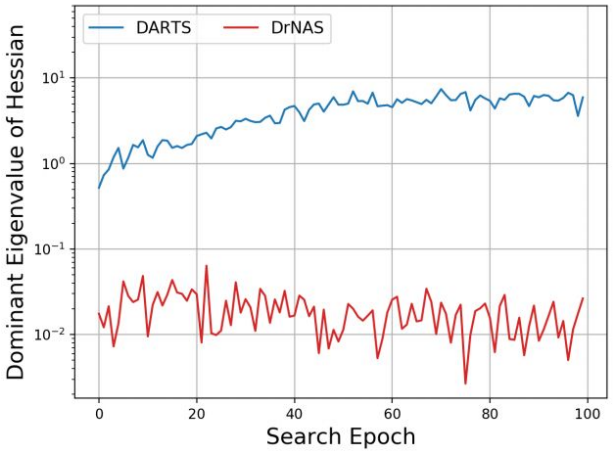
[Chen*, Wang*, Cheng*, Tang, Hsieh] DrNAS: Dirichlet Neural Architecture Search. In ICLR, 2020.

Regularization effect of distribution learning

- We show that minimizing the expected Lval controls the trace norm of Hessian:

$$E_{q(\alpha|\beta)}(L_{val}(w,\alpha)) \gtrsim \tilde{L}_{val}(w^*,\mu) + C \cdot tr\left(\nabla^2_{\mu}\tilde{L}_{val}(w^*,\mu)\right)$$

with $\tilde{L}_{val}(w^*,\mu) = L_{val}(w^*, Softmax(\mu))$



Performance of DrNAS

Architecture	Test Error (%)	Search Cost (GPU days)	Search Method
DARTS (1st) (Liu et al., 2019)	3.00 ± 0.14	0.4	differentiable
DARTS (2nd) (Liu et al., 2019)	2.76 ± 0.09	1.0	differentiable
SNAS (moderate) (Xie et al., 2019)	2.85 ± 0.02	1.5	differentiable
DrNAS (no progressive learning)	2.54 ± 0.03	0.4	differentiable

(progressive learning will be introduced later)

[Chen*, Wang*, Cheng*, Tang, Hsieh] DrNAS: Dirichlet Neural Architecture Search. In ICLR, 2020.

Performance of DrNAS

- On NAS-Bench-201
  - Achieve oracle when searching on CIFAR-100
    DrNAS (73.51) vs SNAS (69.34) vs DARTS (38.97)

Method	CIFAR-10		CIFAR-100		ImageNet-16-120	
	validation	test	validation	test	validation	test
ResNet	90.83	93.97	70.42	70.86	44.53	43.63
Random (baseline)	90.93 ± 0.36	93.70 ± 0.36	70.60 ± 1.37	70.65 ± 1.38	42.92 ± 2.00	42.96 ± 2.15
RSPS	84.16 ± 1.69	87.66 ± 1.69	45.78 ± 6.33	46.60 ± 6.57	31.09 ± 5.65	30.78 ± 6.12
Reinforce	91.09 ± 0.37	93.85 ± 0.37	70.05 ± 1.67	70.17 ± 1.61	43.04 ± 2.18	43.16 ± 2.28
ENAS	39.77 ± 0.00	54.30 ± 0.00	10.23 ± 0.12	10.62 ± 0.27	16.43 ± 0.00	16.32 ± 0.00
DARTS (1st)	39.77 ± 0.00	54.30 ± 0.00	38.57 ± 0.00	38.97 ± 0.00	18.87 ± 0.00	18.41 ± 0.00
DARTS (2nd)	39.77 ± 0.00	54.30 ± 0.00	38.57 ± 0.00	38.97 ± 0.00	18.87 ± 0.00	18.41 ± 0.00
GDAS	90.01 ± 0.46	93.23 ± 0.23	24.05 ± 8.12	24.20 ± 8.08	40.66 ± 0.00	41.02 ± 0.00
SNAS	90.10 ± 1.04	92.77 ± 0.83	69.69 ± 2.39	69.34 ± 1.98	42.84 ± 1.79	43.16 ± 2.64
DSNAS	89.66 ± 0.29	93.08 ± 0.13	30.87 ± 16.40	31.01 ± 16.38	40.61 ± 0.09	41.07 ± 0.09
PC-DARTS	89.96 ± 0.15	93.41 ± 0.30	67.12 ± 0.39	67.48 ± 0.89	40.83 ± 0.08	41.31 ± 0.22
DrNAS	91.55 ± 0.00	94.36 ± 0.00	73.49 ± 0.00	73.51 ± 0.00	46.37 ± 0.00	46.34 ± 0.00
optimal	91.61	94.37	73.49	73.51	46.77	47.31

Supernet training: Projection/proximal operation

- NASP: Adding constraints to enforce sparsity of operation weights:

  For each edge: $\alpha \in C,\ \text{where } C = \{\alpha \mid \|\alpha\|_0 = 1 \text{ and } 0 \leq \alpha_o \leq 1,\ \forall o\}$

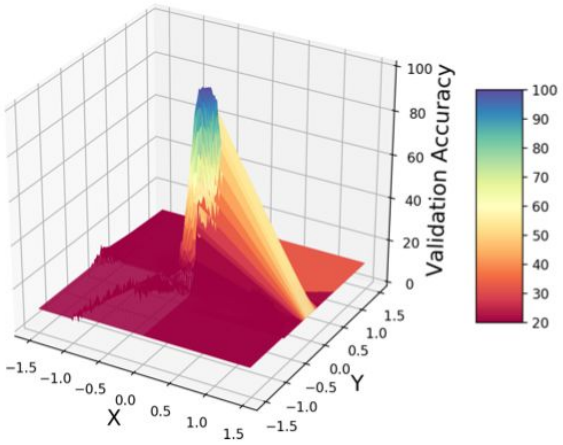
- At each step:
  - Update $\alpha' \leftarrow \alpha - \nabla_\alpha f(\alpha)$
  - Project α' to the discrete constraint set (roughly, select the most important operation)
  - NASP also has other regularization terms to promote sparsity

Architecture	Test Error (%)	Search Cost (GPU days)	Search Method
DARTS (1st) (Liu et al., 2019)	3.00 ± 0.14	0.4	differentiable
DARTS (2nd) (Liu et al., 2019)	2.76 ± 0.09	1.0	differentiable
SNAS (moderate) (Xie et al., 2019)	2.85 ± 0.02	1.5	differentiable
DrNAS (Chen et al., 2020)	2.54 ± 0.03	0.4	differentiable
NASP (Yao et al., 2019)	2.83 ± 0.09	0.1	differentiable

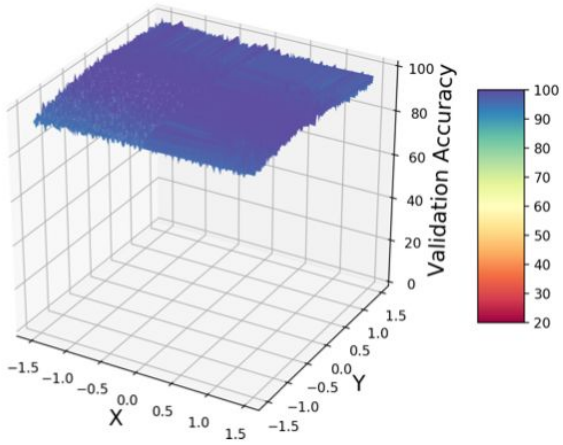
[Yao, Xu, Tu, Zhu] NASP: Efficient Neural Architecture Search via Proximal Iterations. In AAAI, 2020.

Supernet training: perturbation-based regularization

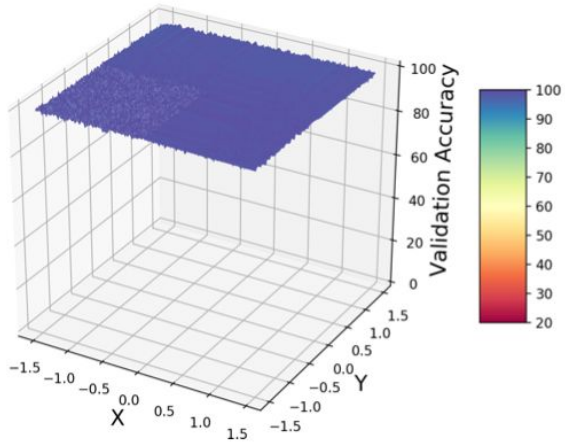
- A smoother landscape will make supernet robust to discreterization



(a) DARTS



(b) SDARTS-RS



(c) SDARTS-ADV

Supernet training: perturbation-based regularization

- Make supernet robust to $\boldsymbol{\alpha}$ perturbation
  - Since we need to perturb it to a discrete architecture in the final stage
- Mathematically, we hope the supernet robust to random or adversarial (worst-case) perturbation of $\boldsymbol{\alpha}$

[Chen and Hsieh] Stabilizing Differentiable Architecture Search via Perturbation-based Regularization. In ICML, 2020.

Supernet training: perturbation-based regularization

- Make supernet robust to $\boldsymbol{\alpha}$ perturbation
  - Since we need to perturb it to a discrete architecture in the final stage
- Mathematically, we hope the supernet robust to random or adversarial (worst-case) perturbation of $\boldsymbol{\alpha}$
- Smooth DARTS (SDARTS):

$$\min_\alpha L_{\text{val}}(\bar{w}(\alpha), \alpha), \quad \text{s.t.}$$

SDARTS-RS: $\bar{w}(\alpha) = \arg\min_w E_{\delta \sim U_{[-\epsilon,\epsilon]}} L_{\text{train}}(w, A + \delta)$

SDARTS-Adv: $\bar{w}(\alpha) = \arg\min_w \max_{\|\delta\| \le \epsilon} L_{\text{train}}(w, A + \delta)$

SDARTS: Each step

➔ Perturb $\boldsymbol{\alpha}$
- Random: $\alpha' \leftarrow \alpha + N(0, \sigma^2)$
- Adversarial: $\alpha' \leftarrow \alpha + \nabla_\alpha L_{\text{train}}(\alpha, w)$

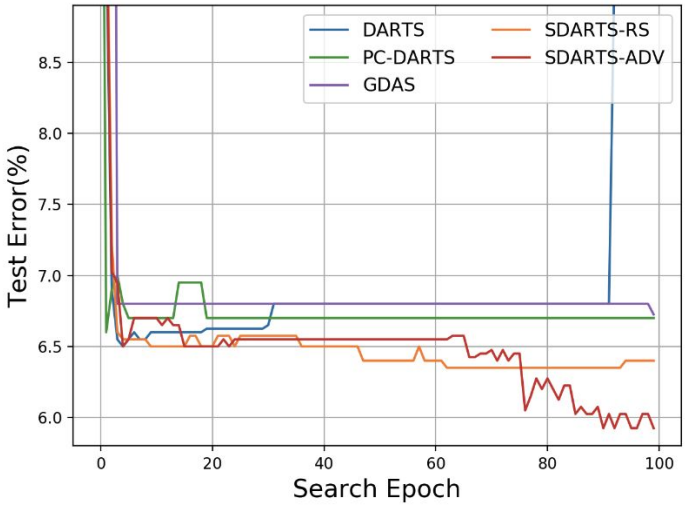
➔ Update w based on $\boldsymbol{\alpha}'$

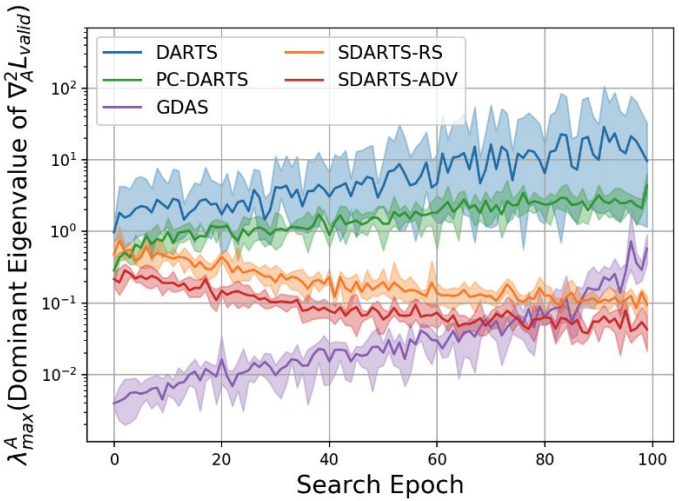
➔ Update $\boldsymbol{\alpha}$ based on w

[Chen and Hsieh] Stabilizing Differentiable Architecture Search via Perturbation-based Regularization. In ICML, 2020.

SmoothDARTS

- On NAS-Bench-1Shot1
  - Continues to discover better architectures
  - Anneal Hessian to a low level





SmoothDARTS

- Can be combined with other search methods:

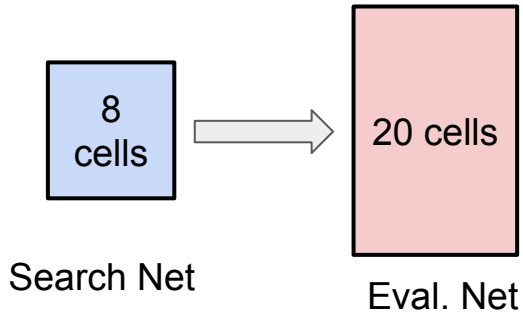
	–	**RS**	**ADV**
DARTS	3.00 ± 0.14	2.67 ± 0.03	2.61 ± 0.02
PC-DARTS	2.57 ± 0.07	2.54 ± 0.04	2.49 ± 0.04
P-DARTS	2.50 ± 0.10	2.50 ± 0.03	2.48 ± 0.02

[Chen and Hsieh] Stabilizing Differentiable Architecture Search via Perturbation-based Regularization. In ICML, 2020.

Improved Scalability for DARTS

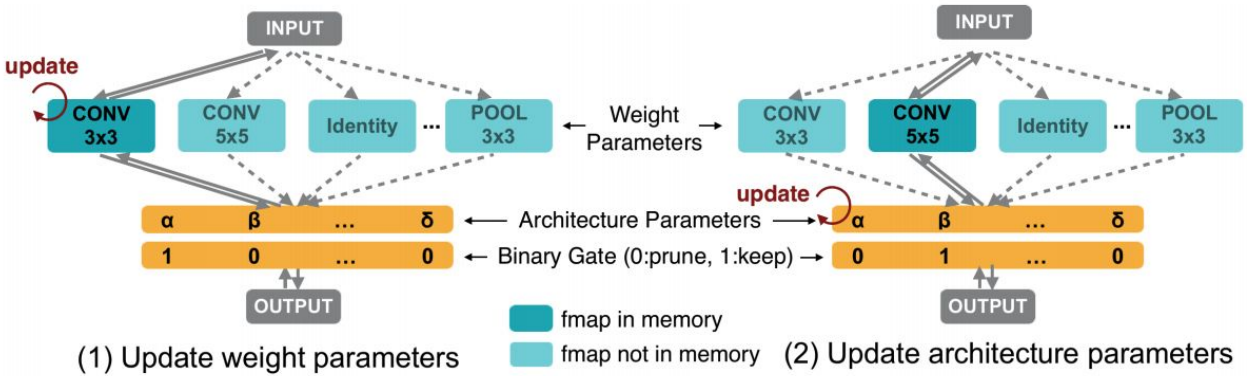
Scalability

- Supernet training cost: Assume each edge has K operations
  - Roughly K times slower than standard training
  - Roughly K times memory cost
- A standard DARTS pipeline:
  - Searching on a proxy task that with fewer layers/channels or smaller dataset
  - DARTS paper:
    8 cells/16 channels for search ⇒ 20 cells/36 channels during evaluation
    CIFAR-10 ⇒ ImageNet
- How to directly search on the target problem without a proxy task?



ProxylessNAS (operation-level reduction)

- Only sample **one path** in each update and masking all other operations
- Compute the gradient for the operations in this path
- All the operations will be adjusted accordingly due to the normalization effect
  - Selected operation weight ↑ => all other operation weights ↓



[Cai, Zhu, Han] ProxylessNAS: Direct Neural Architecture Search on Target Task and Hardware. In ICLR, 2019.

Channel-level Reduction

- PC-DARTS (Xu et al., 2019): Instead of dropping operations, randomly dropping **channels** in search time
  - Will not affect the performance too much
  - Sometimes can even improve generalization
- Progressive learning (Chen et al., 2020):
  - Progressively prune the channels while reducing the channel dropout rate

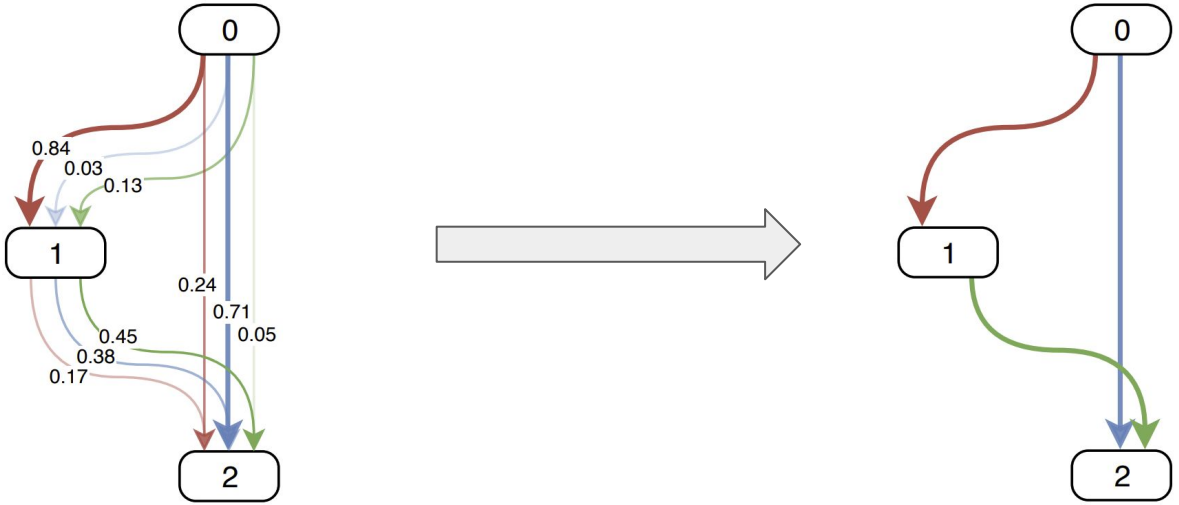
Performance summarization

Architecture	Test Error (%)	Search Cost (GPU days)	Search Method
DARTS (1st) (Liu et al., 2019)	3.00 ± 0.14	0.4	differentiable
DARTS (2nd) (Liu et al., 2019)	2.76 ± 0.09	1.0	differentiable
SNAS (moderate) (Xie et al., 2019)	2.85 ± 0.02	1.5	differentiable
DrNAS (Chen et al., 2020)	2.54 ± 0.03	0.4	differentiable
NASP (Yao et al., 2019)	2.83 ± 0.09	0.1	differentiable
SDARTS-ADV (Chen & Hsieh, 2020)	2.61 ± 0.02	1.3	differentiable
ProxylessNAS (Cai et al., 2019)†	2.08	4.0	differentiable
PC-DARTS (Xu et al., 2020)	2.57 ± 0.07	0.1	differentiable
DrNAS (with progressive learning)	2.46 ± 0.03	0.6	differentiable

† Obtained on a different space with PyramidNet as the backbone.

Architecture Selection

Architecture Selection in Differentiable NAS



(e) Search end

(f) Final cell

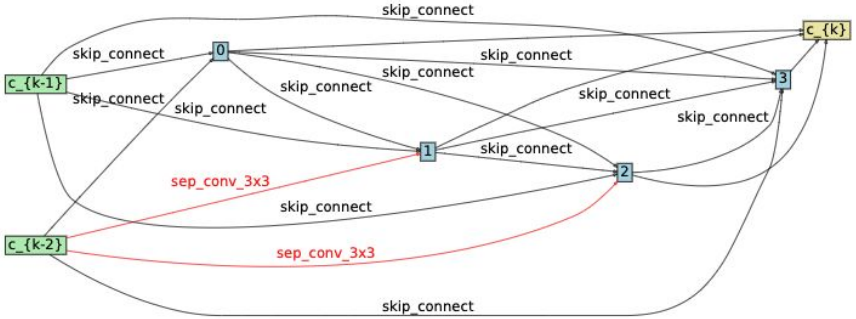
Architecture selection

- Recall the skip-domination problem:
  - For the optimal supernet with infinite number of layers: $\alpha_{\text{skip}} \uparrow 1 \quad \text{and} \quad \alpha_{\text{conv}} \downarrow 0$
  - $\boldsymbol{\alpha}$ values may not really represent the ***"importance"*** of each operation
- Skip connection stands out if we select the best operation based on $\boldsymbol{\alpha}$
- Does $\alpha_{\text{skip}} > \alpha_{\text{conv}}$ mean skip connection is better than convolution?

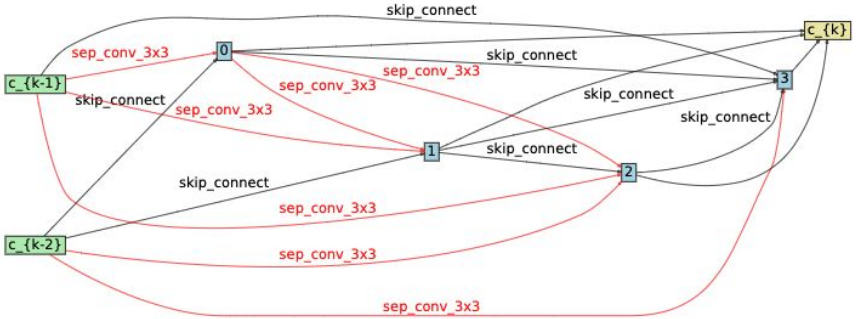
[Wang, Cheng, Chen, Tang, Hsieh] Rethinking Architecture Selection in Differentiable NAS. ICLR, 2020 (outstanding paper award)

Does α represent operation strength?

- Probably Not!
- S2: (Skip_connect, sep_conv_3x3)
  - Skip connections dominate according to α
  - But the accuracy of S2 supernet benefits from more convolutions



Magnitude-base selection



Progressive tuning selection

Does $\boldsymbol{\alpha}$ represent operation strength?

- Same observations on large space: DARTS space
  - Magnitude of $\boldsymbol{\alpha}$ deviates from accuracy of the supernet
  - Some operations with small $\boldsymbol{\alpha}$ are in fact more important for supernet

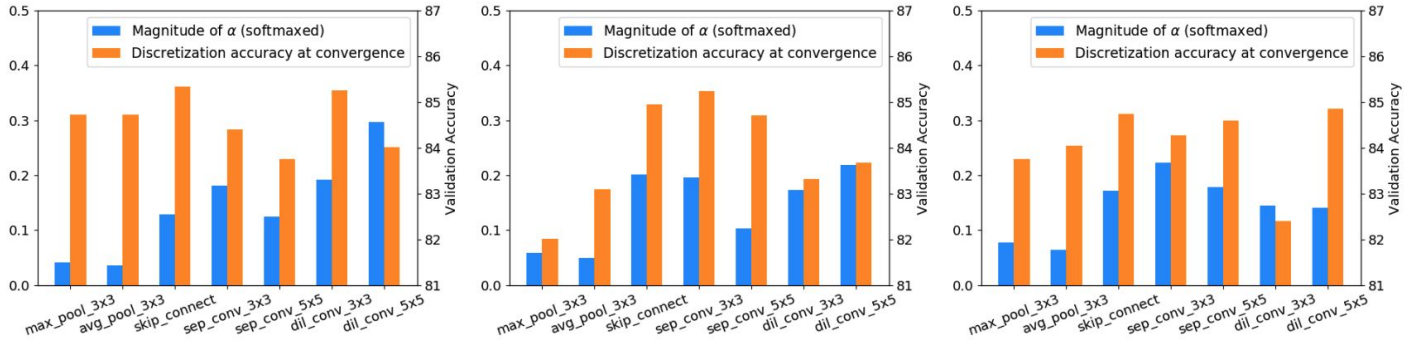


Figure: Magnitude of α vs Accuracy after choosing one operation

A New Architecture Selection Method

- Evaluate the importance of an operation ***o*** by:
  Compute the drop of validation accuracy when ***o*** is removed
  (no need for further training)
- Use this to choose the best ***o*** for an edge
- Fine-tune the solution, and move to the next edge
- “Perturbation-based Selection” (**PT** for short)

[Wang, Cheng, Chen, Tang, Hsieh] Rethinking Architecture Selection in Differentiable NAS. ICLR, 2020 (outstanding paper award)

Results in Cases Where DARTS Fails

- PT consistently improves over the original magnitude-based selection

Dataset	Space	DARTS	DARTS+PT (Ours)
C10	S1	3.84	3.50
	S2	4.85	2.79
	S3	3.34	2.49
	S4	7.20	2.64
C100	S1	29.46	24.48
	S2	26.05	23.16
	S3	28.90	22.03
	S4	22.85	20.80
SVHN	S1	4.58	2.62
	S2	3.53	2.53
	S3	3.41	2.42
	S4	3.05	2.42

Results in Cases Where DARTS Fails

- Performance improves with more searching epochs

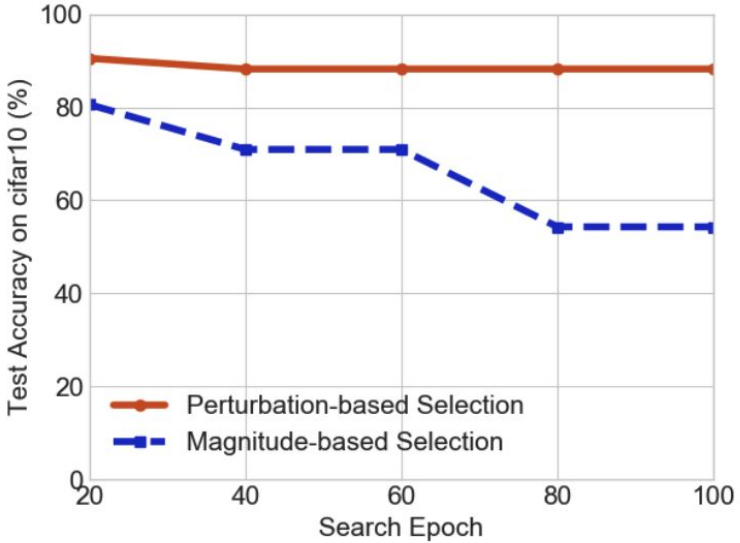


Figure: Test accuracy vs search epoch on NAS-Bench-201 space

Performance on DARTS space

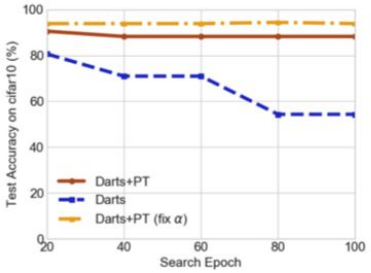
Architecture	Test Error (%)	Search Cost (GPU days)	Search Method
DARTS (1st) (Liu et al., 2019)	3.00 ± 0.14	0.4	differentiable
DARTS (2nd) (Liu et al., 2019)	2.76 ± 0.09	1.0	differentiable
SNAS (moderate) (Xie et al., 2019)	2.85 ± 0.02	1.5	differentiable
DrNAS (Chen et al., 2020)	2.54 ± 0.03	0.4	differentiable
NASP (Yao et al., 2019)	2.83 ± 0.09	0.1	differentiable
SDARTS-ADV (Chen & Hsieh, 2020)	2.61 ± 0.02	1.3	differentiable
ProxylessNAS (Cai et al., 2019)†	2.08	4.0	differentiable
PC-DARTS (Xu et al., 2020)	2.57 ± 0.07	0.1	differentiable
DrNAS (with progressive learning)	2.46 ± 0.03	0.6	differentiable
DARTS+PT (Wang et al., 2020)	2.61 ± 0.08	0.8	differentiable
SDARTS-ADV+PT	2.54 ± 0.01	0.8	differentiable

† Obtained on a different space with PyramidNet as the backbone.

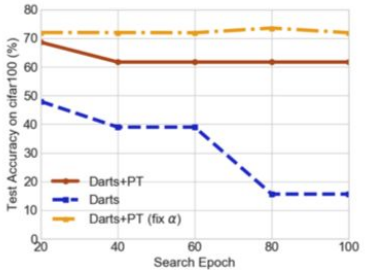
Possibility of removing α

Table 3: Darts+PT on S1-S4 (test error (%)).

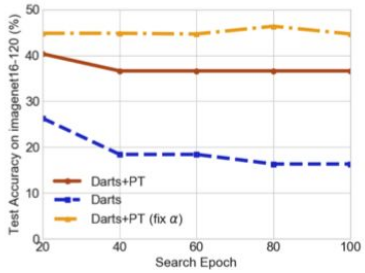
Dataset	Space	DARTS	Darts+PT (Ours)	Darts+PT (fix α)*
C10	S1	3.84	3.50	2.86
	S2	4.85	2.79	2.59
	S3	3.34	2.49	2.52
	S4	7.20	2.64	2.58
C100	S1	29.46	24.48	24.40
	S2	26.05	23.16	23.30
	S3	28.90	22.03	21.94
	S4	22.85	20.80	20.66
SVHN	S1	4.58	2.62	2.39
	S2	3.53	2.53	2.32
	S3	3.41	2.42	2.32
	S4	3.05	2.42	2.39



CIFAR10



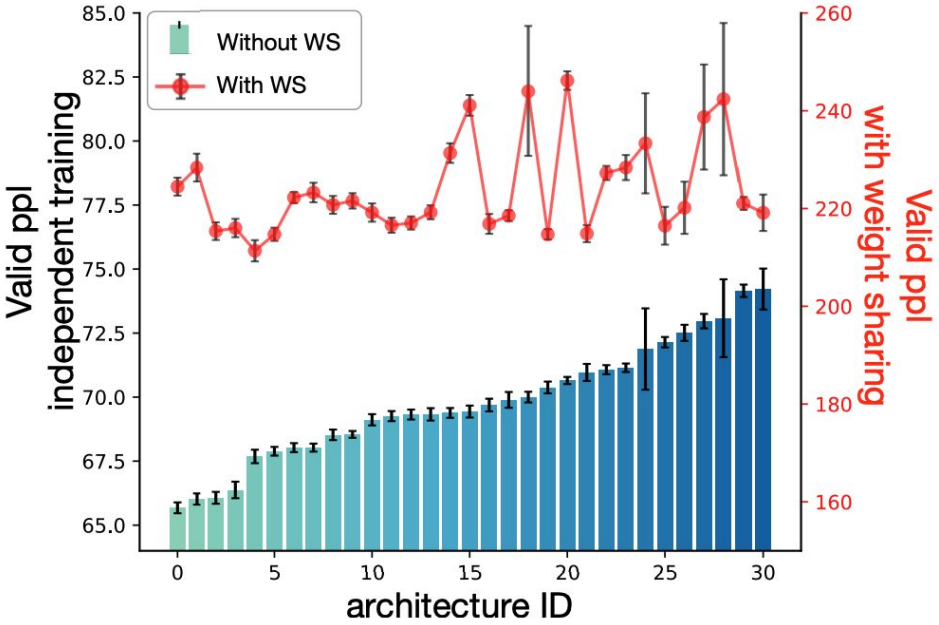
CIFAR100



SVHN

Challenges in Differentiable NAS

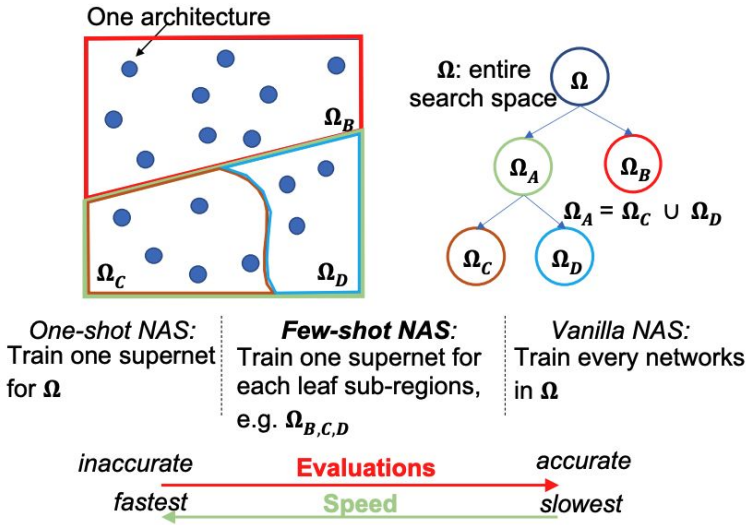
- Is the “weight sharing” assumption valid?



Weight sharing leads to significant different architecture ranking (Yu et al., 2020)

Partial weight sharing?

- (Zhao et al., 2021): Few-shot NAS
  - “Split” architectures based on the operations on an edge
  - Weight sharing only within each group
- Better ways to identify when to share weights?
- How to incorporate this idea in other NAS algorithms?



[Zhao, Wang, Tian, Fonseca, Guo] Few-shot Neural Architecture Search. ICML 2021.

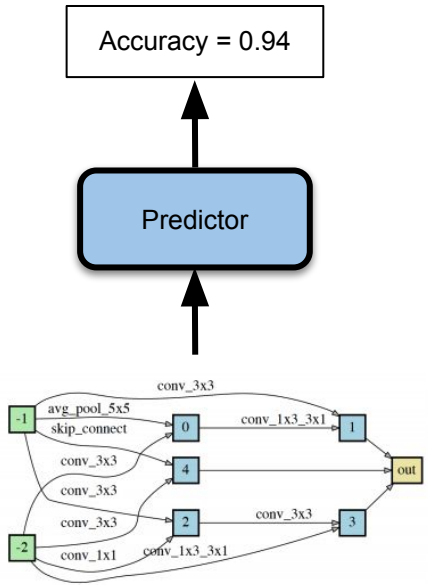
Predictor-based NAS with Graph Neural Network

Vanilla NAS

- $\alpha^* = \arg\max_{\alpha \in \mathcal{A}} Valid\_Acc(\alpha)$
- However, $Valid\_Acc(\alpha)$ is expensive to evaluate
  - Need to train an architecture fully
- Solution - Surrogate Predictor:
  - Learn a cheap surrogate model of $\tilde{f}(\alpha) \approx Valid\_Acc(\alpha)$
  - Use $\tilde{f}(\alpha)$ to evaluate architectures

Surrogate model (Predictor)

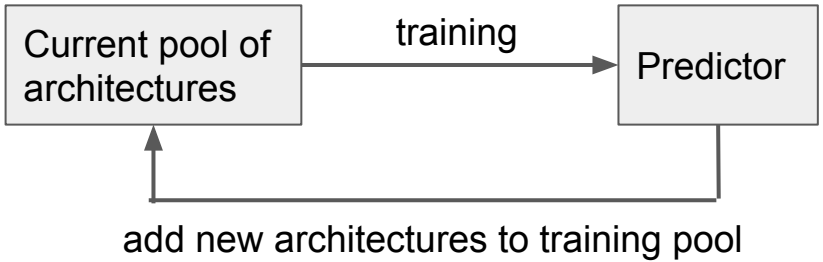
- $\tilde{f}(\alpha)$ is typically a regression model
- Train $\tilde{f}(\alpha)$ using a small set of architectures (hundreds)
- Use $\tilde{f}(\alpha)$ to evaluate other architectures and identify the best one



[Wen, Liu, Li, Chen, Bender, Kindermans] Neural Predictor for Neural Architecture Search. In ECCV, 2020.

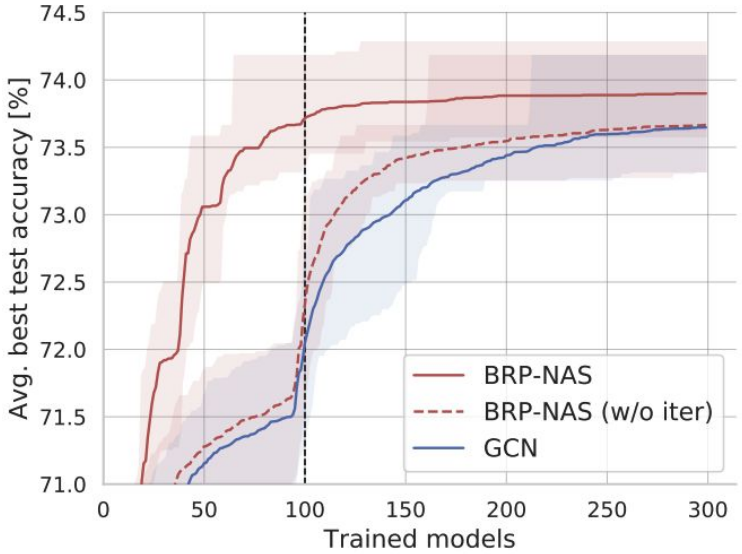
Iterative data selection

- Previously: generate training set at once, and use it to train predictor
- We can iteratively pick training data for the predictor:
  - Training predictor using the current training pool of architectures
  - Use the trained predictor to propose new architectures
  - Augment the current training pool with these new architectures
- Iterative data selection improves the predictor performance for top models



Iterative data selection

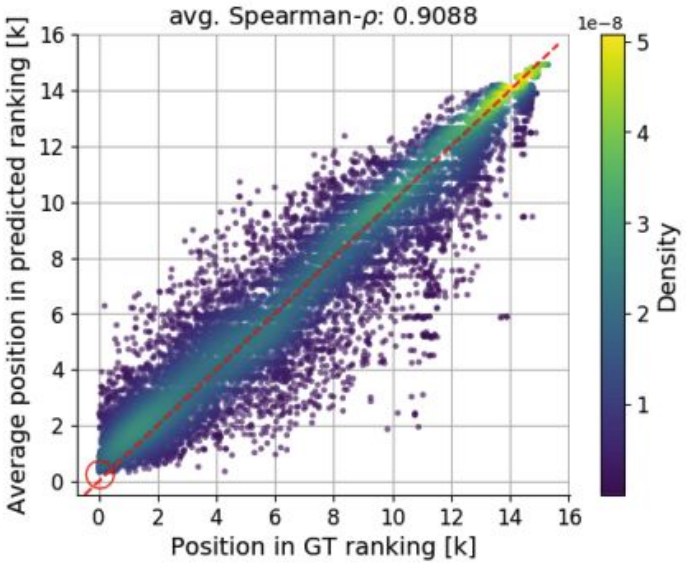
- Iterative data selection improves the predictor accuracy for top models



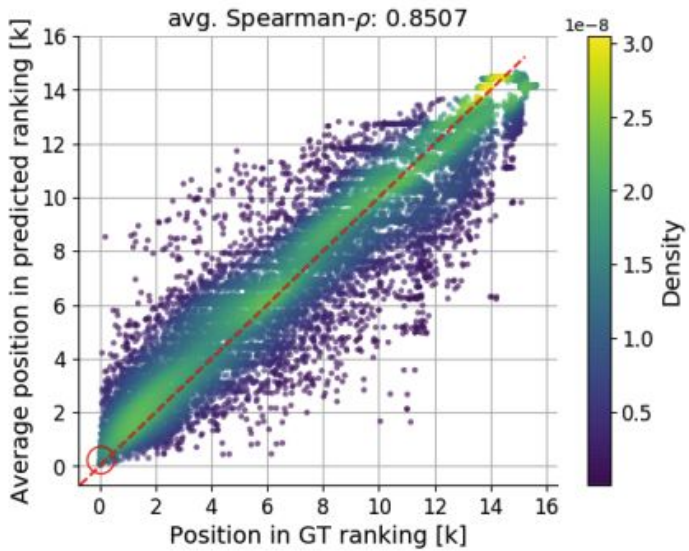
[Dudziak, Chau, Lee, Kim, Lane] Prediction-based NAS using GCNs. In NeurIPS, 2020.

Iterative data selection

- Iterative data selection improves the predictor accuracy for top models



Train Predictor At Once

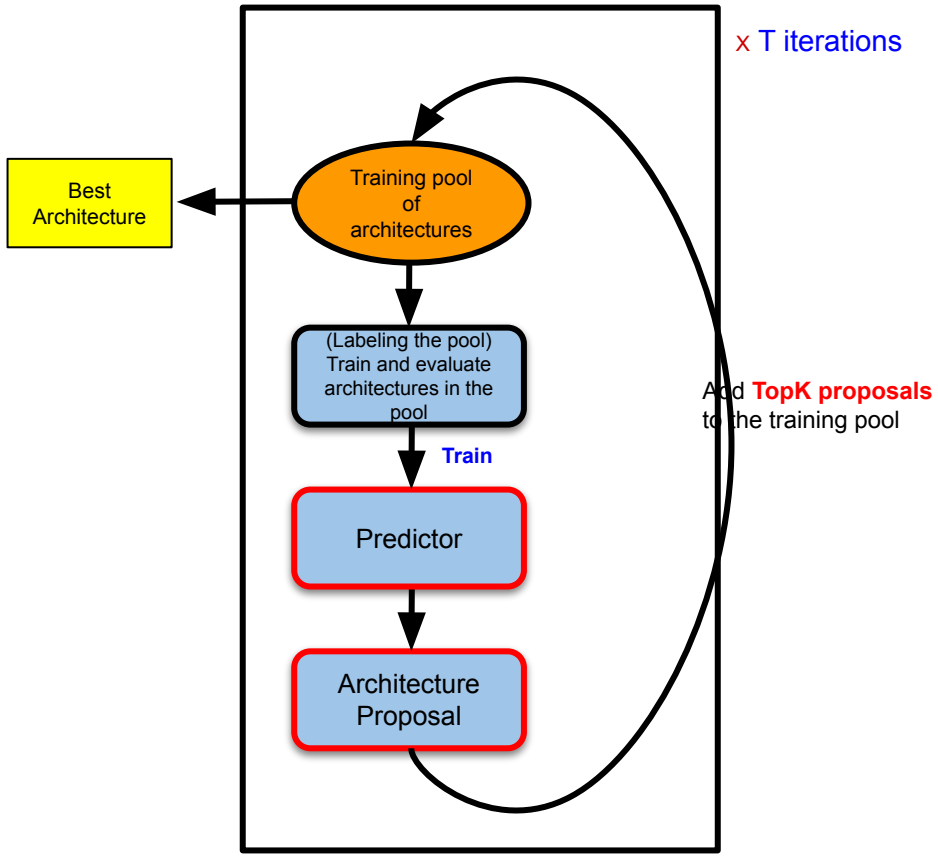


Training predictor iteratively

[Dudziak, Chau, Lee, Kim, Lane] Prediction-based NAS using GCNs. In NeurIPS, 2020.

Pipeline for predictor-based NAS

- Most of SOTA predictor-based methods follow this pipeline
- They mainly differ in:
  - Proposal method
  - Choice of the predictor



Architecture Proposal

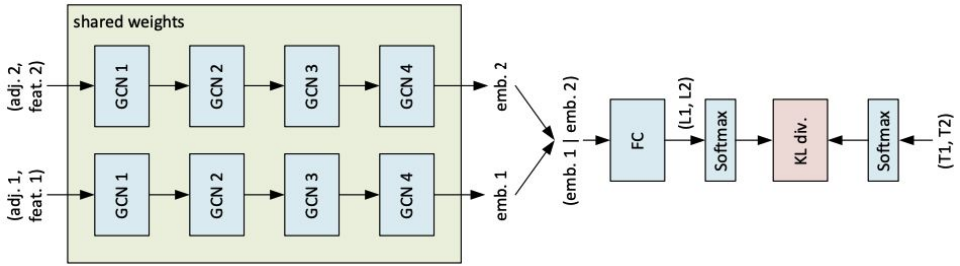
- Architectures with high predicted accuracy (exploitation)
  - Motivation: allow the predictor to focus on promising regions of the search space
- Architectures with high uncertainty under current predictor (exploration)
  - Motivation: encourage good coverage of the search space
- In practice, there are trade-offs between these two goals

Architecture Proposal

- Formally we define Acquisition function $\psi(\alpha; \tilde{f})$:
  - Used to score and rank architectures during Architecture Proposal
  - Architectures with highest acquisition scores will be proposed at every iteration
- Architecture proposal using $\psi(\alpha; \tilde{f})$:
  - Randomly sample a subset of architectures: $\mathcal{A}_s$
  - $\tilde{\mathcal{A}} = \underset{\alpha \in \mathcal{A}_s}{\mathrm{TopK}}(\psi(\alpha; \tilde{f}))$
- Two simple examples:
  - $\psi(\alpha; \tilde{f}) = \tilde{f}(\alpha)$: 100% exploitation
  - $\psi(\alpha; \tilde{f}) = random()$: 100% exploration

Predictors

- Common choice of predictors:
  - Gauss Process (Kandasamy et al., 2018; Ru et al., 2021)
  - MLP (White et al., 2019; Wen et al., 2019; Yan et al., 2020)
  - Siamese Ranker (Dudziak et al., 2020; Wang et al., 2021)
- A key component of the predictor is the Architecture Encoder
- Many of the recent improvements in predictor-based NAS come from better architecture encoding schema



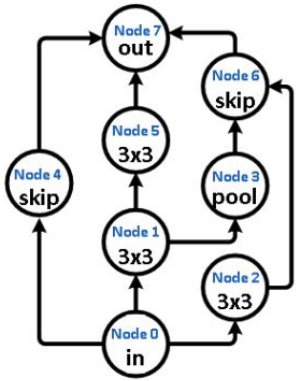
Siamese ranker with GCN encoder (Dudziak et al., 2020)

Architecture encoder is crucial to predictor performance

- A good encoding: architectures with similar accuracy stays close in the encoder space
- List of architecture encodings:
  - String inputs + LSTM [NAO] (Luo et al., 2018)
  - Adjacency matrix-based [BOHB NAS-variant] (Dong et al., 2020)
  - Path Encoding [BANANAS] (White et al., 2019)
  - GNN [NAT] (Guo et al., 2019)
  - GNN + pretraining [arch2vec] (Yan et al., 2020)
- GNN-variants are widely used as the encoder in recent SOTA algorithms

Graph Neural Network as architecture encoder

- Architectures as graphs
  - *Nodes*: operations (one hot) $X \in \mathcal{R}^{|\mathcal{N}||\mathcal{O}|}$
  - *Edges*: connections (Adjacency matrix): $A \in \mathcal{R}^{|\mathcal{N}||\mathcal{N}|}$
- Graph Neural Networks (Kipf., 2016):
  - $H^{(0)} = X$
  - $H^{(t)} = \sigma(AH^{(t-1)}W^{(t)}),\ t = 1, ..., T$
  - Architecture encoding: sum $H^{(T)}$ over all nodes: $h(\alpha) = row\_sum(H^{(T)})$



A

Node	0	1	2	3	4	5	6	7
0	0	1	1	0	1	0	0	0
1	0	0	0	1	0	1	0	0
2	0	0	0	0	0	0	1	0
3	0	0	0	0	0	0	1	0
4	0	0	0	0	0	0	0	1
5	0	0	0	0	0	0	0	1
6	0	0	0	0	0	0	0	1
7	0	0	0	0	0	0	0	0

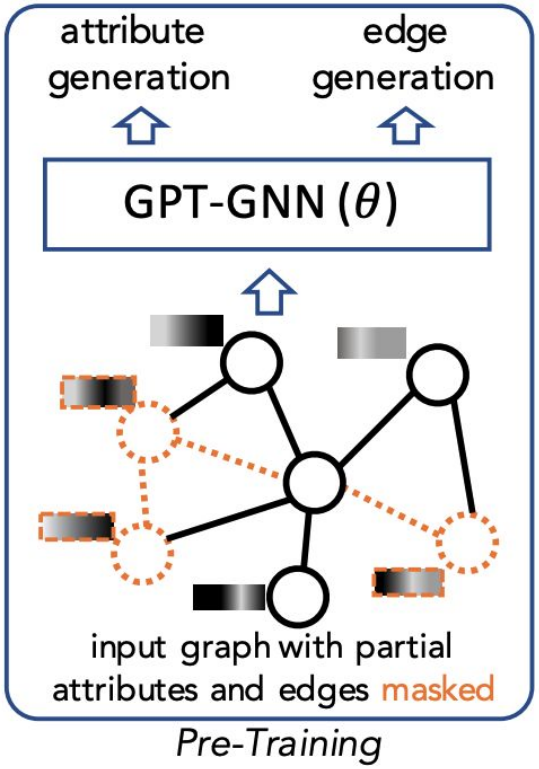
X

Node	in	1x1	3x3	pool	skip	none	out
0	1	0	0	0	0	0	0
1	0	0	1	0	0	0	0
2	0	0	1	0	0	0	0
3	0	0	0	1	0	0	0
4	0	0	0	0	1	0	0
5	0	0	1	0	0	0	0
6	0	0	0	0	1	0	0
7	0	0	0	0	0	0	1

Fig from Yan et al., 2020

Pretraining

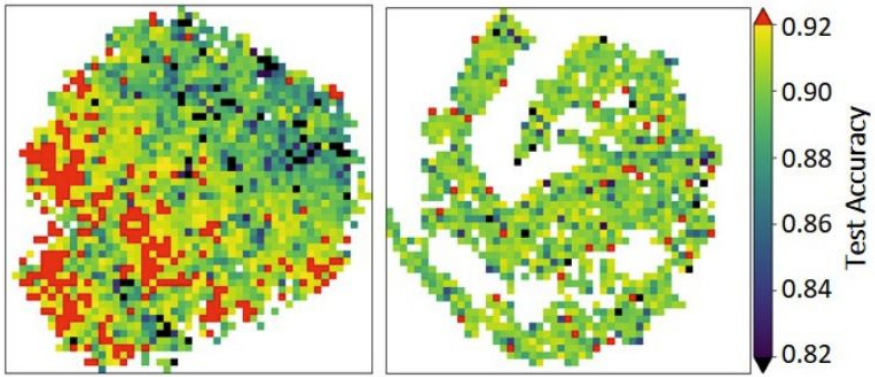
- Training a GNN encoder:
  - Standard supervised training (Arc -> performance):
    Training data is time-consuming to get
  - The training set for predictor is usually small: ~100 architectures
- Pretraining
  - Using (almost) unlimited unlabeled graphs (any architecture in the search space)
  - GNN pretraining:
    - Training GNN encoder to predict the graph itself



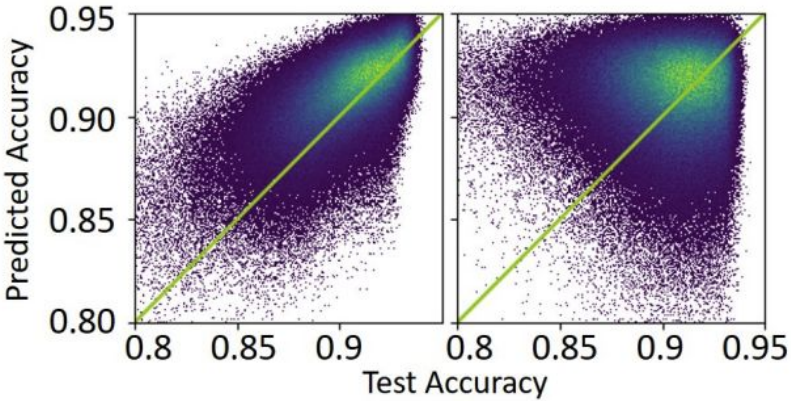
One of the GNN pretraining frameworks (Hu et al., 2020)

Pretraining GNN encoder improves predictor performance

- [Arch2vec] shows that pretraining encoder:
  - Leads to better architecture encodings
  - Improves the predictor performance



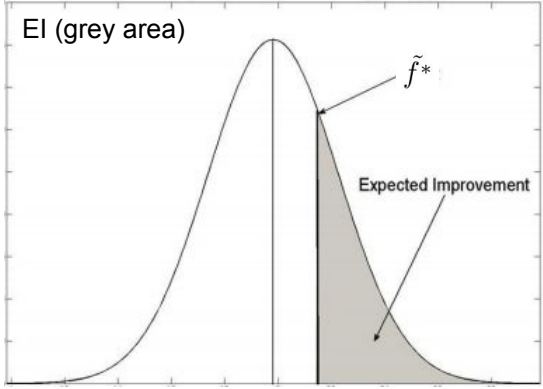
Left: pretraining | Right: w/o pretraining



Left: pretraining | Right: w/o pretraining

[Yan, Zheng, Ao, Zeng, Zhang] Does Unsupervised Architecture Representation Learning Help Neural Architecture Search? In. NeurIPS 2020.

Example: Bayesian Optimization for NAS



- Assume the predictor follows Gaussian Distribution
  - $\tilde{f}(\alpha) \sim N(\mu(\alpha), \sigma^2(\alpha))$
- $\psi(\alpha; \tilde{f})$ = Expected improvement score (EI)
  - Let $\tilde{f}^*$ denote the best accuracy in the current architecture pool
  - If we add an extra architecture α to the pool, the improvement over $\tilde{f}^*$:

$$I(\alpha) = max(\tilde{f}(\alpha) - \tilde{f}^*, 0)$$

If $\tilde{f}(\alpha) < \tilde{f}^*$, our best $\tilde{f}^*$ is unchanged, i.e. no improvement

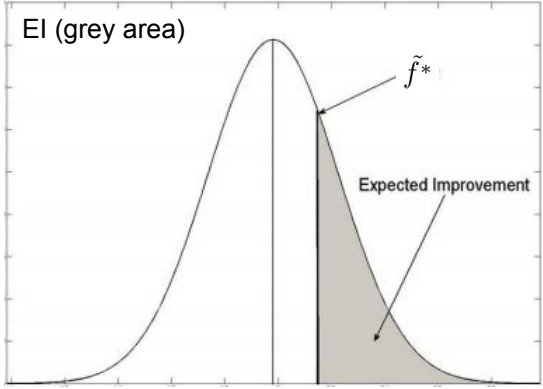
  - Expected Improvement:

$$E[I(\alpha)] = E_{\tilde{f}(\alpha) \sim N(\mu(\alpha), \sigma^2(\alpha))}[max(\tilde{f}(\alpha) - \tilde{f}^*, 0)]$$
$$= \sigma(\alpha)[\gamma(\alpha)\Phi(\gamma(\alpha)) + \phi(\gamma(\alpha))]$$

CDF of Gaussian PDF of Gaussian

$$\gamma(\alpha) = \frac{\mu(\alpha) - \tilde{f}^*}{\sigma(\alpha)}$$

Example: Bayesian Optimization for NAS



- Assume the predictor follows Gaussian Distribution
  - $\tilde{f}(\alpha) \sim N(\mu(\alpha), \sigma^2(\alpha))$
- $\psi(\alpha; \tilde{f})$ = Expected Improvement score (EI)

$$E[I(\alpha)] = E_{\tilde{f}(\alpha)\sim N(\mu(\alpha),\sigma^2(\alpha))}[max(\tilde{f}(\alpha) - \tilde{f}^*, 0)]$$
$$= \sigma(\alpha)[\gamma(\alpha)\Phi(\gamma(\alpha)) + \phi(\gamma(\alpha))]$$

$$\gamma(\alpha) = \frac{\mu(\alpha) - \tilde{f}^*}{\sigma(\alpha)}$$

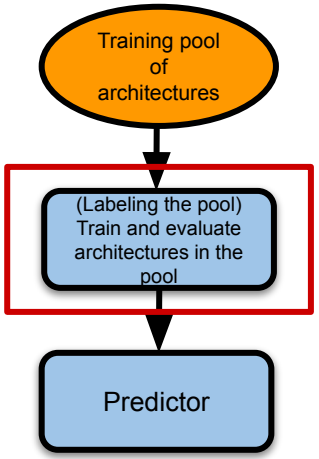
- EI score balances:
  - Exploitation: $\mu(\alpha)$
  - Exploration: $\sigma(\alpha)$
- Other common acquisition functions in BO:
  - Probability of Improvement: $PI(\alpha) = E_{\tilde{f}(\alpha)\sim N(\mu(\alpha),\sigma^2(\alpha))}[\mathbb{1}(\tilde{f}(\alpha) > \tilde{f}^*)]$
  - Upper Confidence Bound: $UCB(\alpha) = \mu(\alpha) + \sqrt{\beta}\sigma(\alpha)$

Example: Bayesian Optimization for NAS

- Computing Expected Improvement Score analytically requires
  - $\tilde{f}(\alpha) \sim N(\mu(\alpha), \sigma^2(\alpha))$
  - i.e. an uncertainty measure $\sigma(\alpha)$
- Popular Choices of predictors with uncertainty measure $\sigma(\alpha)$:
  - Gaussian Process [NASBOT] (Kandasamy et al., 2020)
  - Ensembles [BANANAS] (White et al., 2020)
  - GNN + MLP + Bayesian Linear Regression [arch2vec] (Yan et al., 2020)
  - …

Improve the efficiency of Predictor-based NAS

- Search cost of NAS methods (without weight-sharing):
  - Early **RL/EA**: >100 GPU days
  - Differentiable NAS: <1 GPU days
  - **Predictor-based**: ~10 GPU days
- Bottleneck of search efficiency - Architecture Training
  - Need to build a training set of architectures to train the predictor
  - Labeling: Train each architecture in the set fully to obtain the validation accuracy

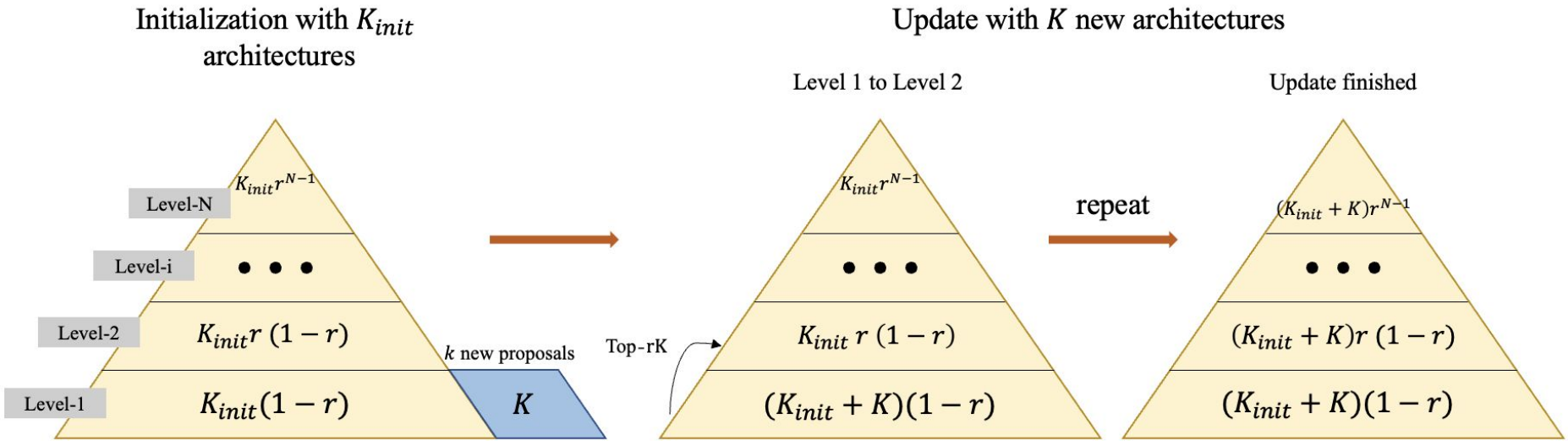


Improve the efficiency of Predictor-based NAS

- Naive early stopping:
  - Cut training length for all architectures
  - Use intermediate accuracy as supervision signals to train the predictor
  - Does not work well due to inaccurate signals
- NOn-uniform Successive Halving (NOSH):
  - Instead of training every architecture fully, we can identify poor performers at early stages
  - Pause/Resume poor architecture training based on their intermediate accuracy to save budget
- RANK-NOSH:
  - NOSH algorithm trains architectures for different lengths
  - Use a ranker model to learn the pairwise comparison between these architectures
  - Reduce the search cost by 5x

[Wang, Chen, Cheng, Tang, Hsieh] RANK-NOSH: Efficient Predictor-Based Architecture Search via Non-Uniform Successive Halving. To appear in ICCV 2021.

Rank-NOSH



All architectures at level i will be trained to epoch e_i;
only a certain ratio of architectures can advance to the next layer

After adding K new proposals, use the same strategy to populate the pyramid.

Improve the efficiency of Predictor-based NAS

-

Architecture	Test Error(%)		Param	Search Budget	Search
	Best	Avg	(M)	(#epochs)	Method
RSWS [20]	2.71	2.85 ± 0.08	4.3	-	Weight Sharing
DARTS [23]	$2.76 \pm 0.09^{\star}$	-	3.6	-	Weight Sharing
SNAS [40]	-	2.85 ± 0.02	2.8	-	Weight Sharing
BayesNAS [46]	$2.81 \pm 0.04^{\star}$	-	3.4	-	Weight Sharing
ProxylessNAS [4]	**2.08**†	-	4.0	-	Weight Sharing
ENAS [29]	$2.89^{\dagger}$	-	4.6	-	Weight Sharing
P-DARTS [8]	2.50	-	3.4	-	Weight Sharing
PC-DARTS [41]	$2.57 \pm 0.07^{\star}$	-	3.6	-	Weight Sharing
SDARTS-ADV [6]	-	2.61 ± 0.02	3.3	-	Weight Sharing
Random Search [23]	$3.29 \pm 0.15^{\star}$	-	3.2	2,400	Random
GATES [27]	$2.58^{\dagger}$	-	4.1	64,000	Predictor
BRP-NAS (high) [12]	-	2.59 ± 0.11	-	36,000	Predictor
BRP-NAS (med) [12]	-	2.66 ± 0.09	-	18,000	Predictor
BANANAS [37]	2.57	2.64	3.6	5,000	Predictor
arch2vec-BO [42]	**2.48**	2.56 ± 0.05	3.6	5,000	Predictor
RANK-NOSH	2.50	$\mathbf{2.53 \pm 0.02}$	3.5	**990**	Predictor

† Obtained on different search spaces than DARTS.

* Error bars are computed by retraining the best discovered architecture multiple times.

5x less budget

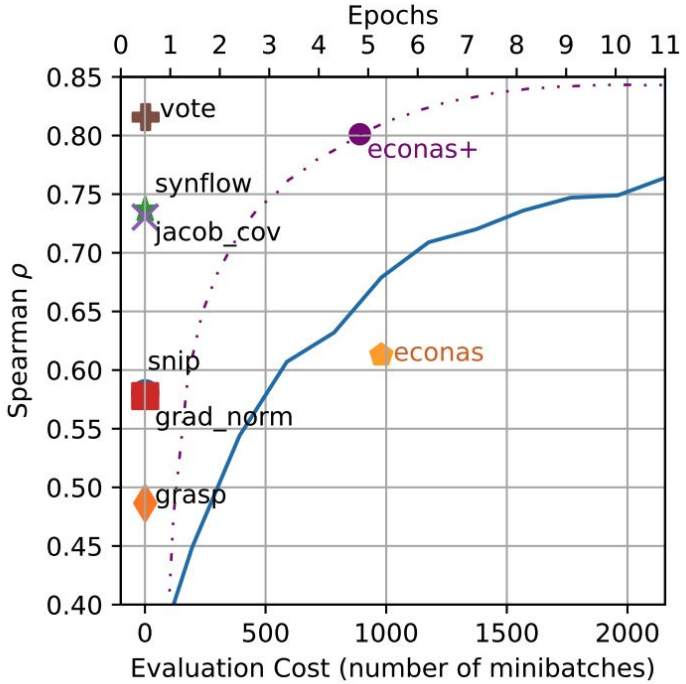
Conclusions and Open Problems

Conclusions

- Differentiable NAS:
  - Based on the idea of weight sharing
  - Very efficient, but less reliable
    - Partially solved by several recent work
- Predictor-based NAS:
  - No weight sharing (more reliable)
  - Slower than 1-shot methods
  - Need a good performance predictor (GNN)

NAS without Training

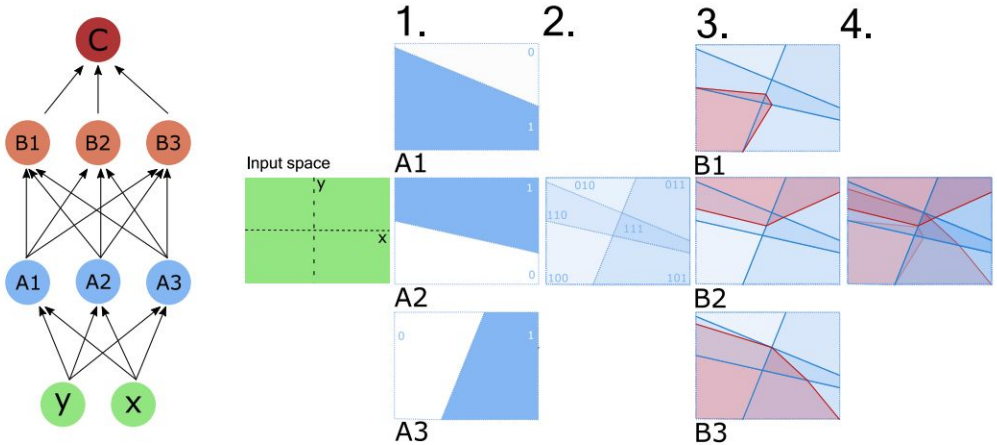
- Can we choose a good architecture without any training?
- Design some kind of "Zero-shot performance evaluator" without training
- These zero-shot performance evaluator demonstrates high correlation with the architecture's final performance



[Abdelfattah, Mehrotra, Dudziak, Lane] Zero-shot Proxies for Lightweight NAS. ICLR 2020.

NAS without Training

- Idea 1: Estimate how many different linear regions in the architecture (with random initialization)

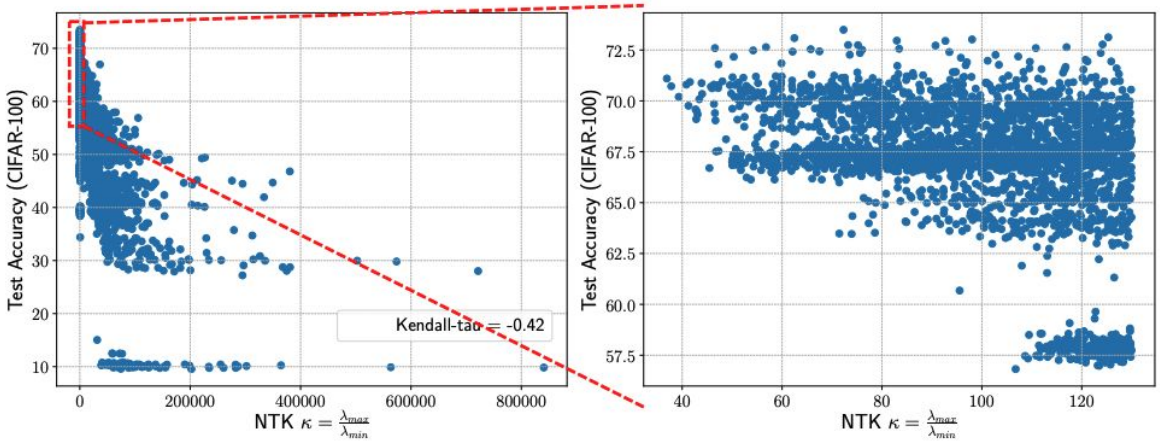


Can be estimated by number of activation patterns

[Mellor, Turner, Storkey, Crowley] Neural Architecture Search without Training. ICML, 2021.
[Chen, Gong, Wang] Neural Architecture Search on Imagenet in Four GPU Hours: A Theoretically Inspired Perspective. ICLR, 2021.

NAS without Training

- Idea 2: Using Neural Tangent Kernel
  - NTK is defined as $\Theta(x, x') = J(x)J(x')^T$
  - The condition number of NTK: measure the “trainability” of neural networks (under the linearization assumption)
  - Empirically has negative correlation with final performance



[Chen, Gong, Wang] Neural Architecture Search on Imagenet in Four GPU Hours: A Theoretically Inspired Perspective. ICLR, 2021.

NAS without Training

- However, using zero-shot predictor itself is limited
- Can we use zero-shot predictors to improve other methods?
  - Rank-NOSH (Predictor-based method):
    Use zero-shot predictors as performance evaluator at epoch 0
  - Combine with other methods (e.g., DARTS, Evolutionary Algorithms)?

Problems of the search space

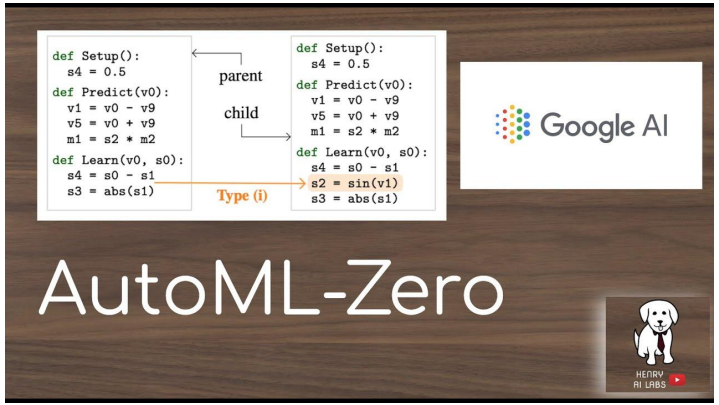
- Current search spaces are too “well-defined”
- Random search performs only marginally worse than NAS
- In practice:
  - Very difficult to define a search space:
  - Either too simple (no good solution) or too difficult (NAS algorithms fail)

Architecture	Source	Test Error Best	Test Error Average
Shake-Shake#	[9]	N/A	2.56
PyramidNet	[46]	2.31	N/A
NASNet-A#*	[50]	N/A	2.65
AmoebaNet-B*	[41]	N/A	2.55 ± 0.05
ProxylessNAS†	[7]	2.08	N/A
GHN#†	[48]	N/A	2.84 ± 0.07
SNAS†	[45]	N/A	2.85 ± 0.02
ENAS†	[39]	2.89	N/A
ENAS	[32]	2.91	N/A
Random search baseline	[32]	N/A	3.29 ± 0.15
DARTS (first order)	[32]	N/A	3.00 ± 0.14
DARTS (second order)	[32]	N/A	2.76 ± 0.09
DARTS (second order)‡	Ours	2.62	2.78 ± 0.12
ASHA baseline	Ours	2.85	3.03 ± 0.13
Random search WS‡	Ours	2.71	2.85 ± 0.08

(figure from Li et al., 2020)

Problems of the search space

- Is there a better search space to evaluate NAS methods?
- How to automatically design a search space?
- Fully automatic or partial automatic?



Thank you!