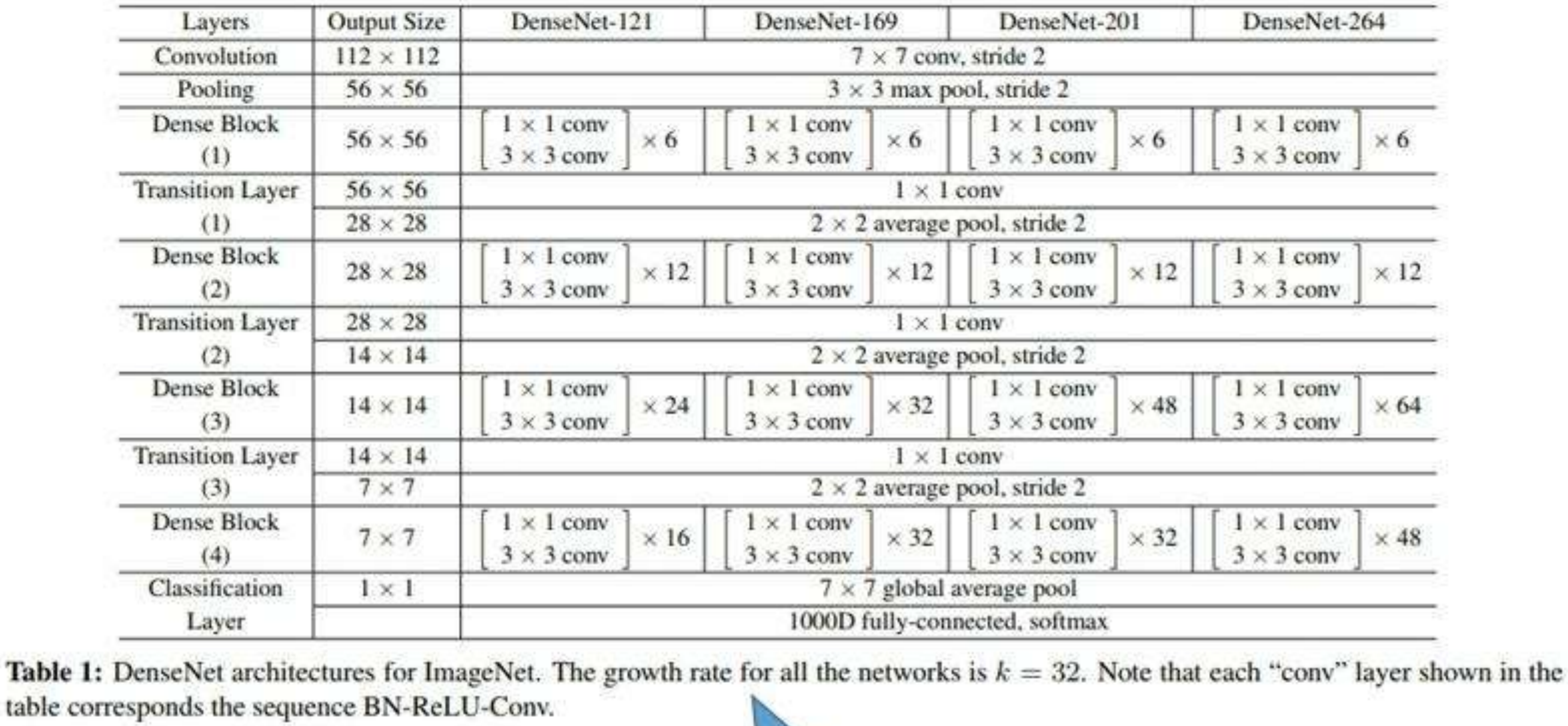
Project Petridish: Efficient Forward Architecture Search

Debadeepta Dey, Hanzhang Hu, John Langford,
Rich Caruana, Saurajit Mukherjee, Eric Horvitz



Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 64$
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

Table 1: DenseNet architectures for ImageNet. The growth rate for all the networks is $k = 32$. Note that each "conv" layer shown in the table corresponds the sequence BN-ReLU-Conv.

"I have no idea how to come up with this!" –
John Langford, April 2018

Motivation for Neural Architecture Search

- Model structure design is difficult.
    - Human experts have designed most networks till now.
    - Many data-sets without good priors on network design.
- Many models are similar to each other.
    - Hyper-parameter tuning is difficult.
    - Trying different architectures manually is difficult.
- Massive talent shortage!

15-minute literature overview

Neural Architecture Search

Search space?

- What architectures can be represented?
- Macro vs. Micro?
- How much human bias goes in search space design?

Search strategy?

- How to explore the search space?
- Classical explore-exploit problem
  - Want to find good architectures quickly.
  - Want to avoid premature convergence to suboptimal ones.

Performance estimation strategy?

- Find architectures that achieve high predictive performance on unseen data
- How do we estimate this performance?
- Can't simply perform standard training and validation (too expensive).
- Lots of research goes here.

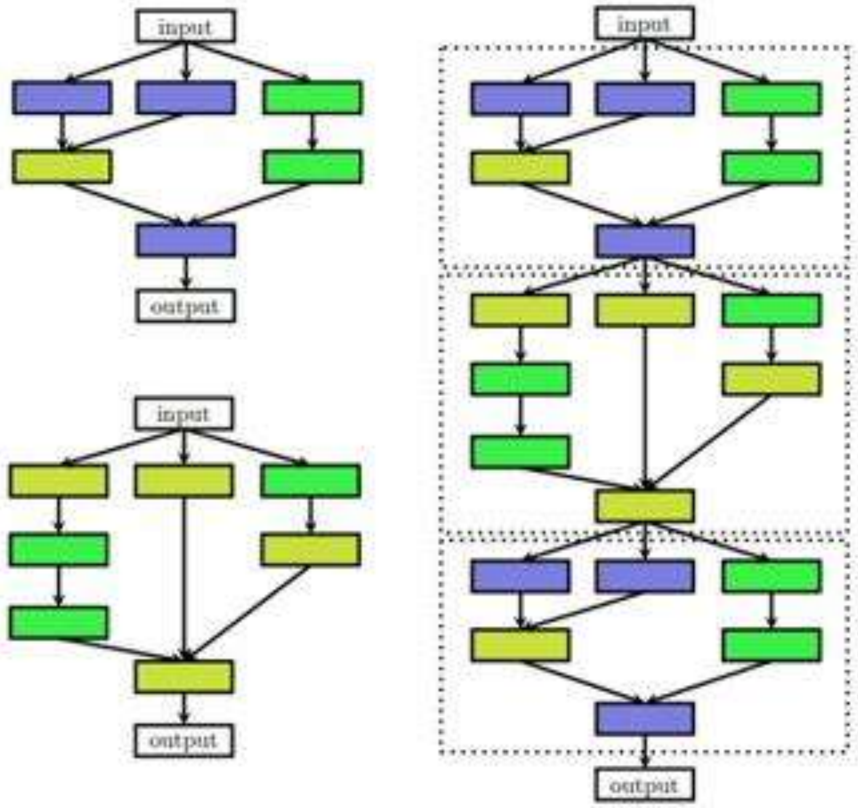


Figure 3: Illustration of the cell search space. Left: Two different cells, e.g., a normal cell (top) and a reduction cell (bottom) (Zoph et al., 2018). Right: an architecture built by stacking the cells sequentially. Note that cells can also be combined in a more complex manner, such as in multi-branch spaces, by simply replacing layers with cells.

Stack cells with predetermined skeletons. (Zoph et al., 2018)
Domain knowledge injection for good skeletons needed.
Our work shows lots of performance left on the table.

Graphic credit: Neural Architecture Search: A Survey, Elsken et al., 2018

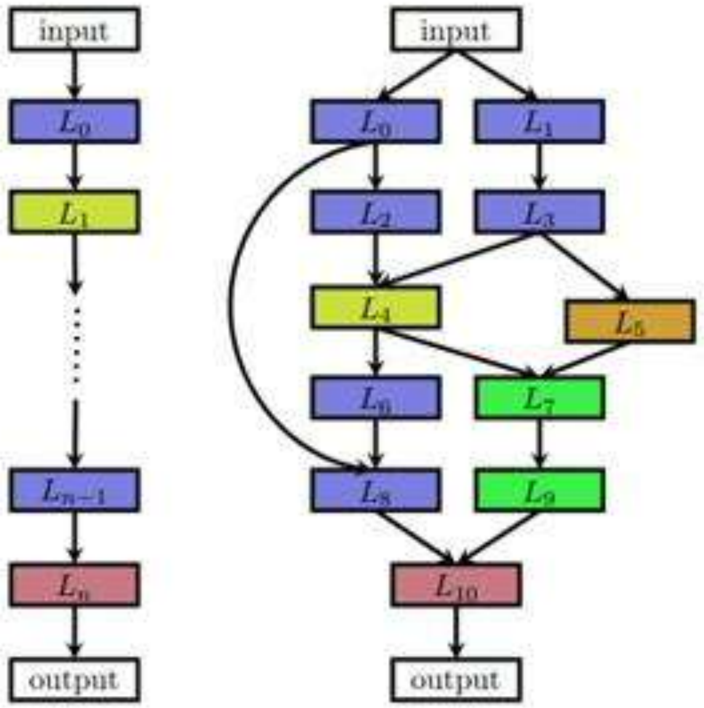


Figure 2: An illustration of different architecture spaces. Each node in the graphs corresponds to a layer in a neural network, e.g., a convolutional or pooling layer. Different layer types are visualized by different colors. An edge from layer L_i to layer L_j denotes that L_i receives the output of L_j as input. Left: an element of a chain-structured space. Right: an element of a more complex search space with additional layer types and multiple branches and skip connections.

General search space.

Little restriction on the kind of architectures that can be realized.

Can be hard to search due to size of space.

Graphic credit: Neural Architecture Search: A Survey, Elsken et al., 2018

Automatic Architecture Hunt (AutoML)

- Continuously updated list of NAS papers:
  - https://www.ml4aad.org/automl/literature-on-neural-architecture-search/

- Excellent survey article:
  - Neural Architecture Search: A Survey, Elsken, Metzen and Hutter, 2018

Neural Architecture Search with Reinforcement Learning (Zoph and Le, 2016)

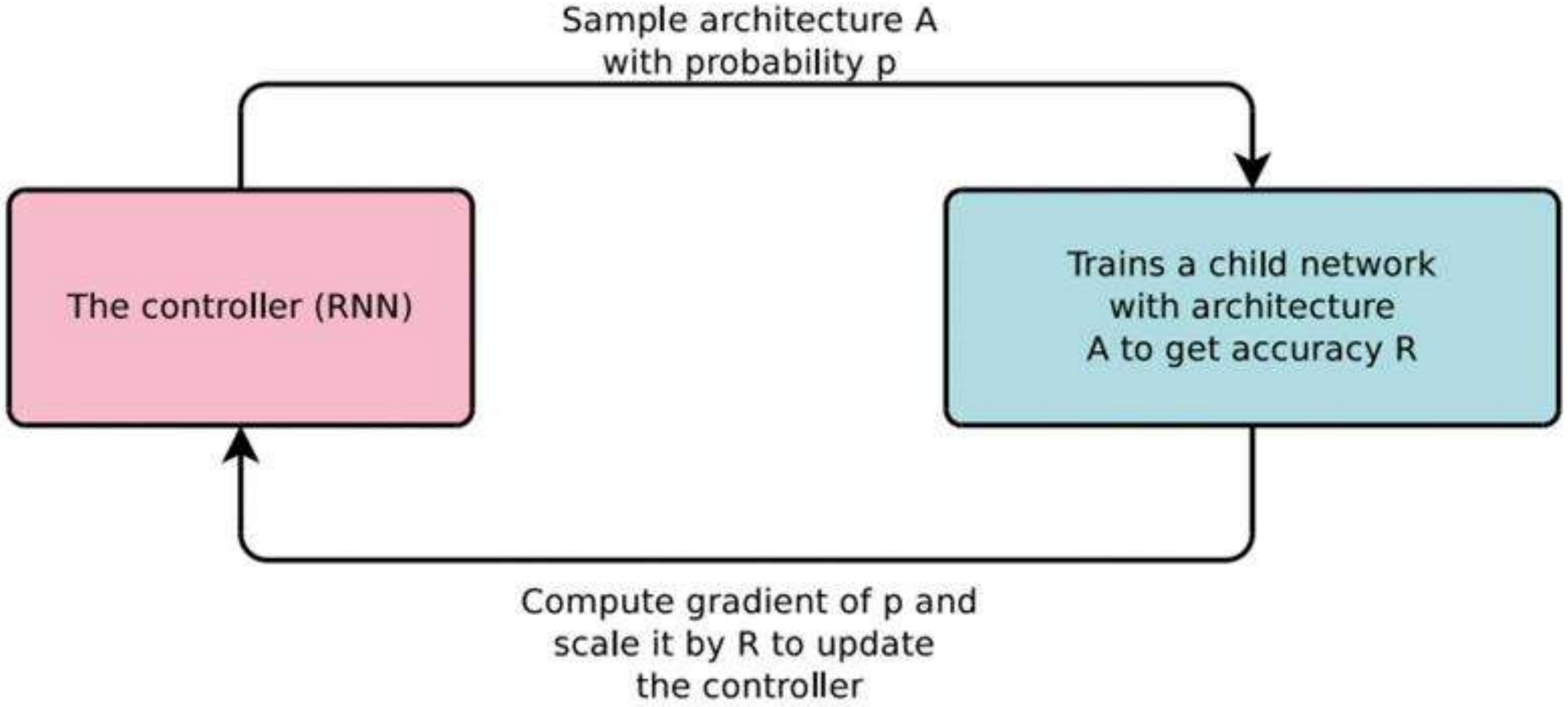


Figure 1: An overview of Neural Architecture Search.

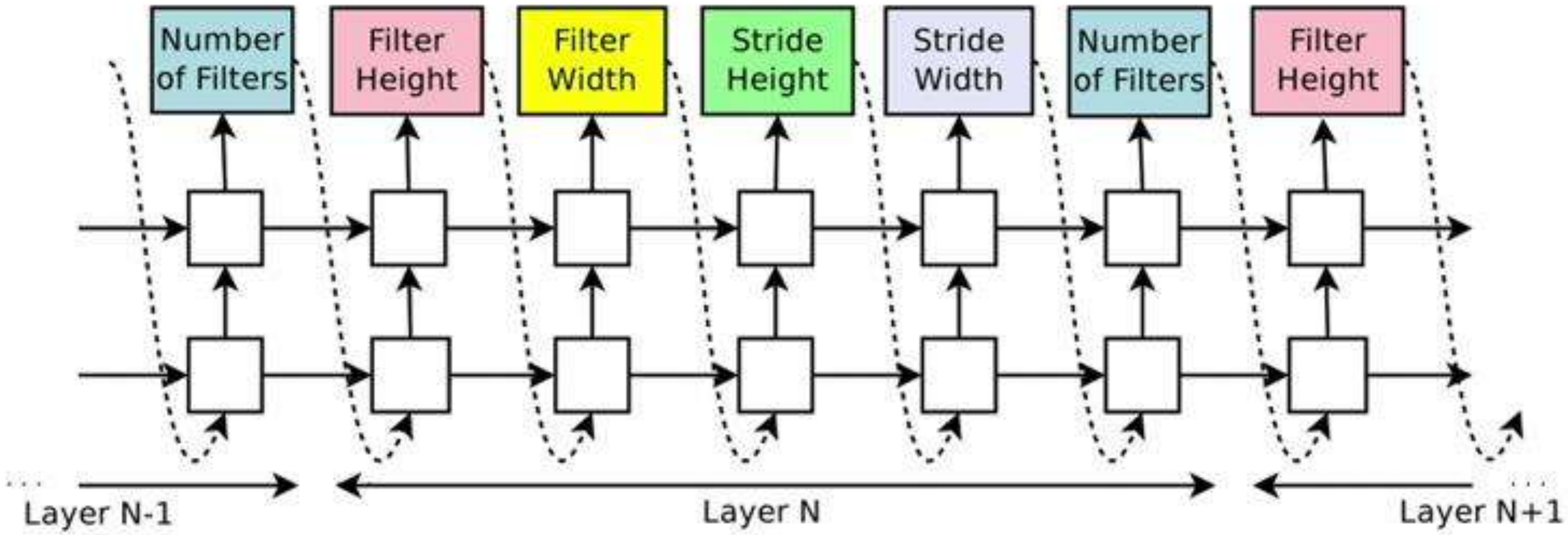


Figure 2: How our controller recurrent neural network samples a simple convolutional network. It predicts filter height, filter width, stride height, stride width, and number of filters for one layer and repeats. Every prediction is carried out by a softmax classifier and then fed into the next time step as input.

Efficient Neural Architecture Search via Parameter Sharing (Pham et al, 2018)

ENAS

On CIFAR-10 achieves a test error of 2.89%

On Penn Treebank achieves perplexity of 55.8 (NAS had 62.4)

In all experiments used a single Nvidia 1080Ti GPU

Search takes less than 16 hours

Compared to NAS >1000x reduction in search time

The main contribution of this work is to improve the efficiency of NAS by forcing all child models to share weights to eschew training each child model from scratch to convergence.

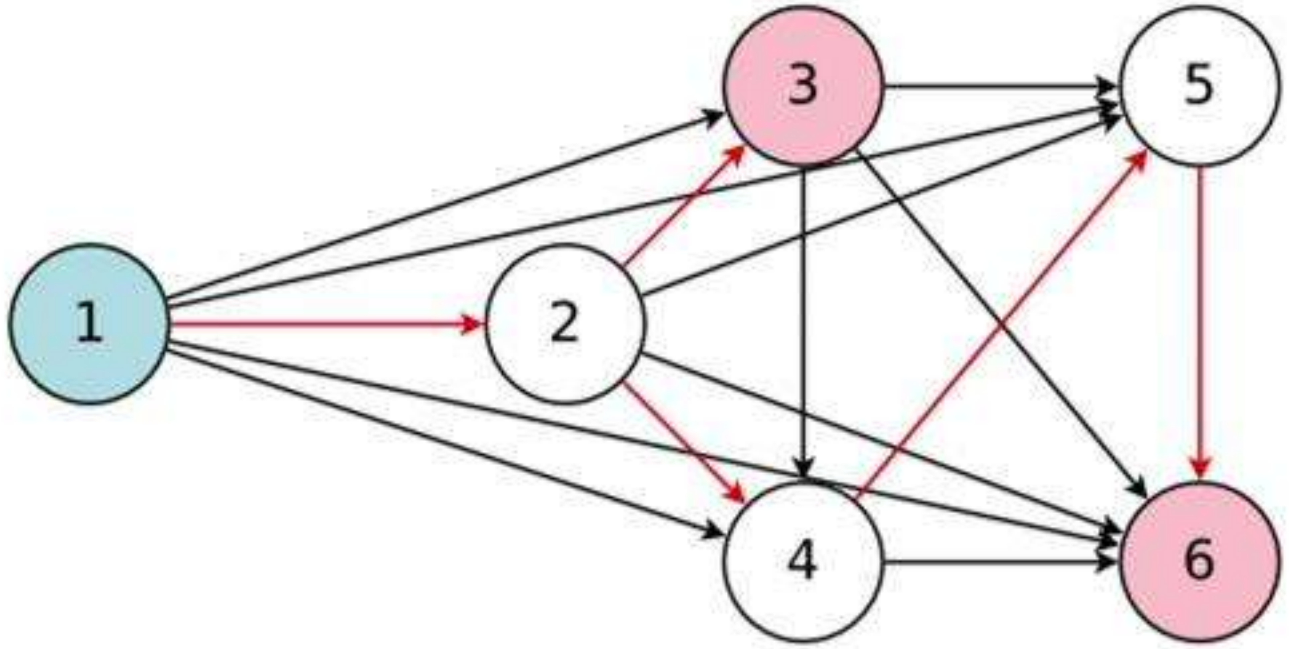


Figure 2. The graph represents the entire search space while the red arrows define a model in the search space, which is decided by a controller. Here, node 1 is the input to the model whereas nodes 3 and 6 are the model's outputs.

Recurrent cell sampling

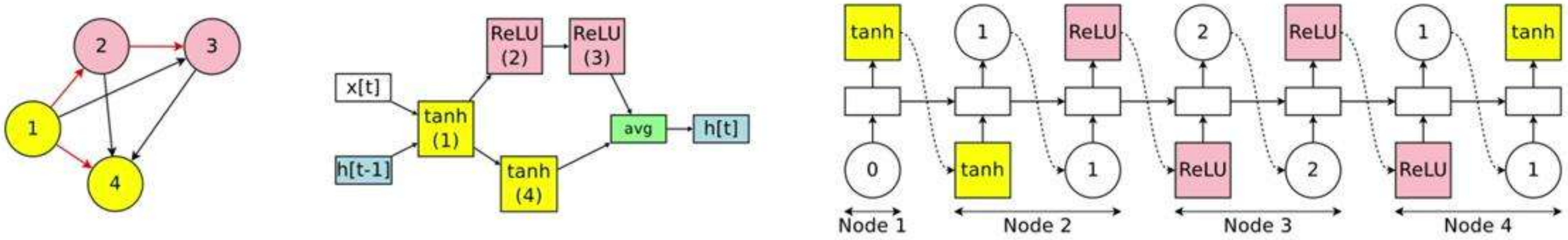


Figure 1. An example of a recurrent cell in our search space with 4 computational nodes. *Left:* The computational DAG that corresponds to the recurrent cell. The red edges represent the flow of information in the graph. *Middle:* The recurrent cell. *Right:* The outputs of the controller RNN that result in the cell in the middle and the DAG on the left. Note that nodes 3 and 4 are never sampled by the RNN, so their results are averaged and are treated as the cell's output.

Convolutional cell sampling

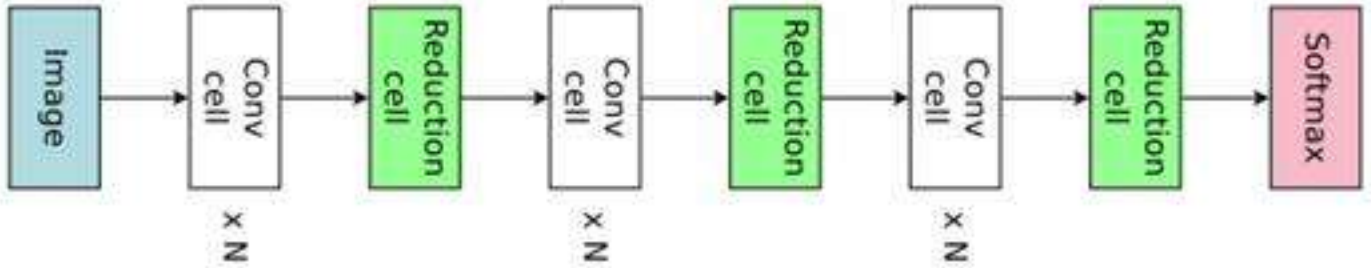


Figure 4. Connecting 3 blocks, each with N convolution cells and 1 reduction cell, to make the final network.

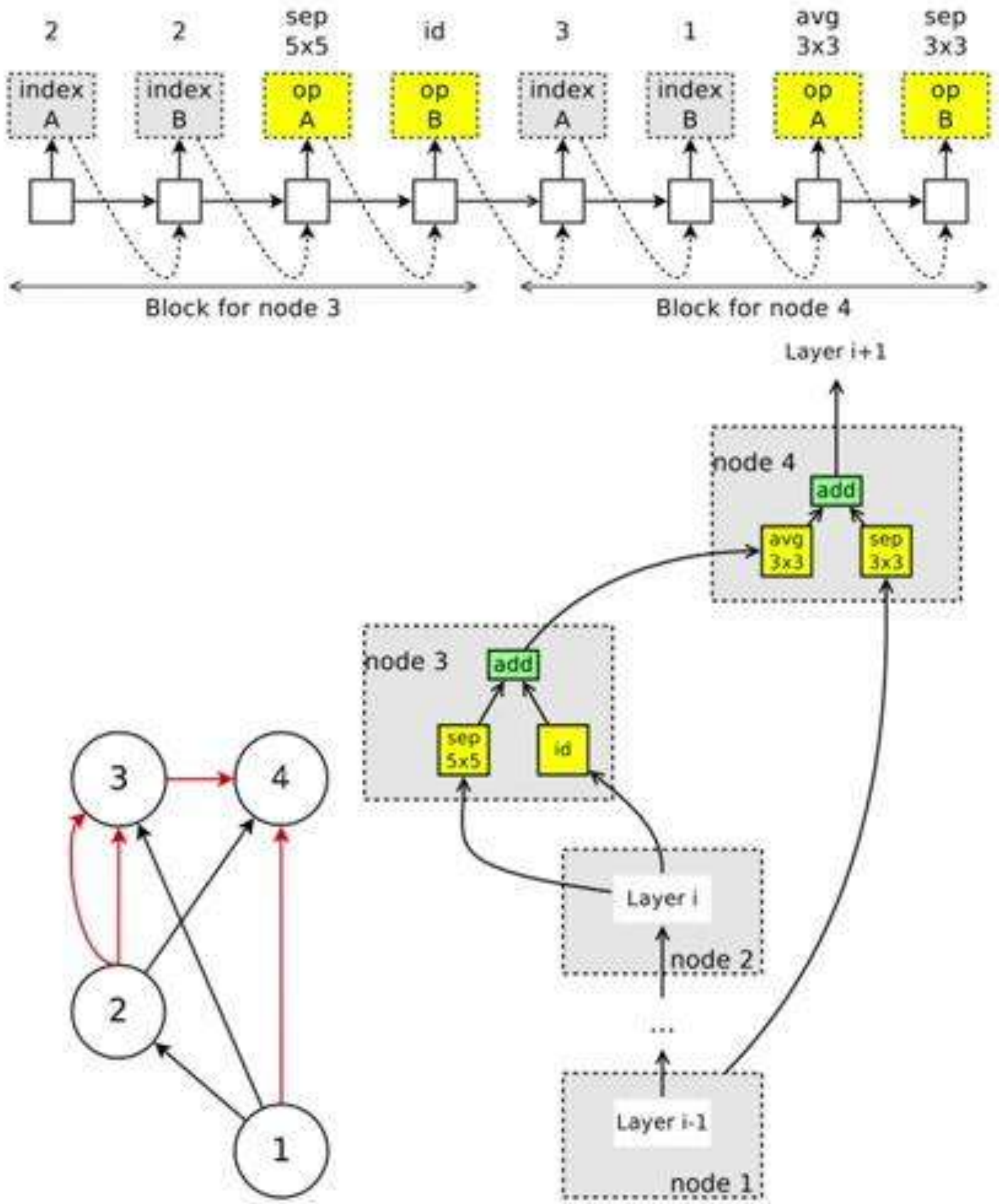
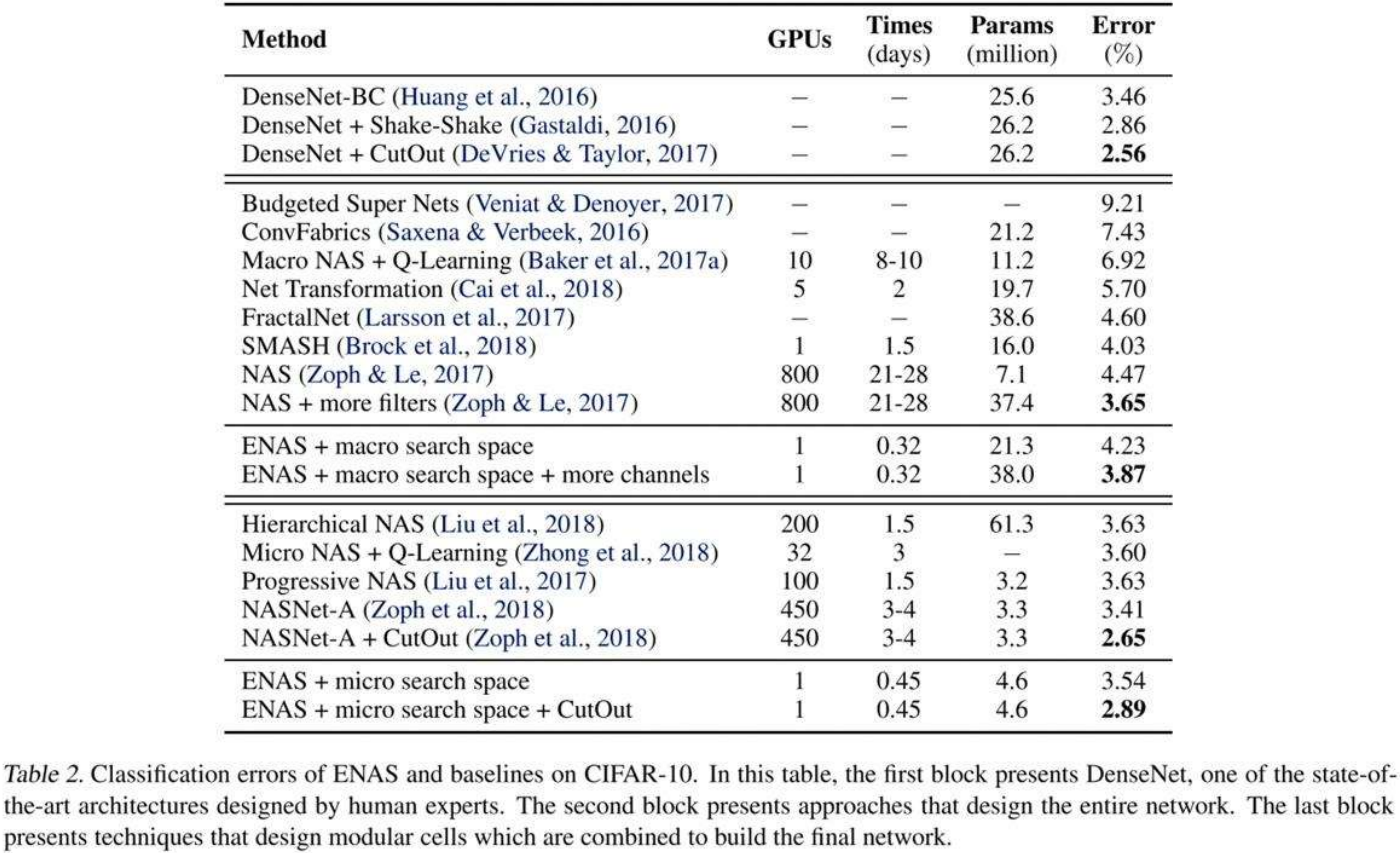


Figure 5. An example run of the controller for our search space over convolutional cells. *Top:* the controller's outputs. In our search space for convolutional cells, node 1 and node 2 are the cell's inputs, so the controller only has to design node 3 and node 4. *Bottom Left:* The corresponding DAG, where red edges represent the activated connections. *Bottom Right:* the convolutional cell according to the controller's sample.



Method	GPUs	Times (days)	Params (million)	Error (%)
DenseNet-BC (Huang et al., 2016)	–	–	25.6	3.46
DenseNet + Shake-Shake (Gastaldi, 2016)	–	–	26.2	2.86
DenseNet + CutOut (DeVries & Taylor, 2017)	–	–	26.2	**2.56**
Budgeted Super Nets (Veniat & Denoyer, 2017)	–	–	–	9.21
ConvFabrics (Saxena & Verbeek, 2016)	–	–	21.2	7.43
Macro NAS + Q-Learning (Baker et al., 2017a)	10	8-10	11.2	6.92
Net Transformation (Cai et al., 2018)	5	2	19.7	5.70
FractalNet (Larsson et al., 2017)	–	–	38.6	4.60
SMASH (Brock et al., 2018)	1	1.5	16.0	4.03
NAS (Zoph & Le, 2017)	800	21-28	7.1	4.47
NAS + more filters (Zoph & Le, 2017)	800	21-28	37.4	**3.65**
ENAS + macro search space	1	0.32	21.3	4.23
ENAS + macro search space + more channels	1	0.32	38.0	**3.87**
Hierarchical NAS (Liu et al., 2018)	200	1.5	61.3	3.63
Micro NAS + Q-Learning (Zhong et al., 2018)	32	3	–	3.60
Progressive NAS (Liu et al., 2017)	100	1.5	3.2	3.63
NASNet-A (Zoph et al., 2018)	450	3-4	3.3	3.41
NASNet-A + CutOut (Zoph et al., 2018)	450	3-4	3.3	**2.65**
ENAS + micro search space	1	0.45	4.6	3.54
ENAS + micro search space + CutOut	1	0.45	4.6	**2.89**

Table 2. Classification errors of ENAS and baselines on CIFAR-10. In this table, the first block presents DenseNet, one of the state-of-the-art architectures designed by human experts. The second block presents approaches that design the entire network. The last block presents techniques that design modular cells which are combined to build the final network.

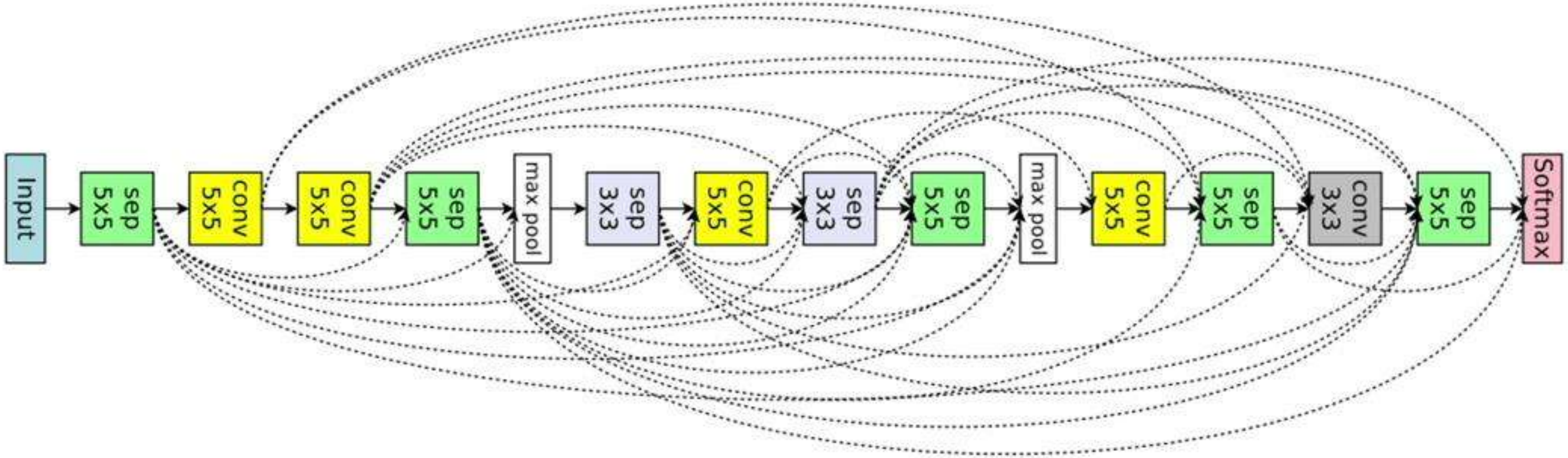
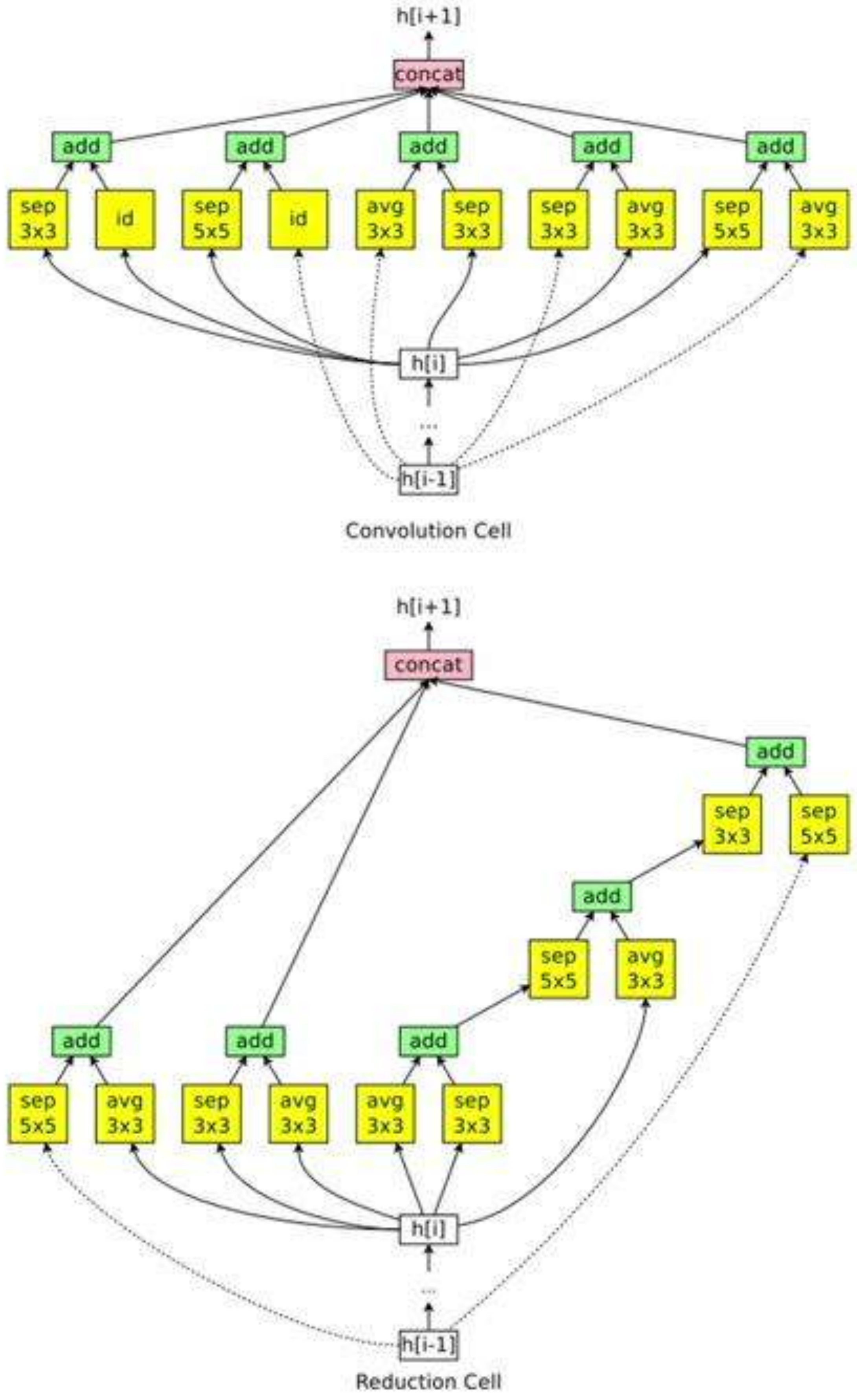


Figure 7. ENAS's discovered network from the macro search space for image classification.



Convolution Cell

Reduction Cell

Figure 8. ENAS cells discovered in the micro search space.

DARTS: Differentiable Architecture Search (Liu et al. 2018)

DARTS

- No controllers!
- No performance prediction!
- Outperforms ENAS, PNAS.
- Cell-based (micro).
- Achieves 2.83% error on CIFAR-10.
- Uses 1000x less computation than Regularized Evolution.

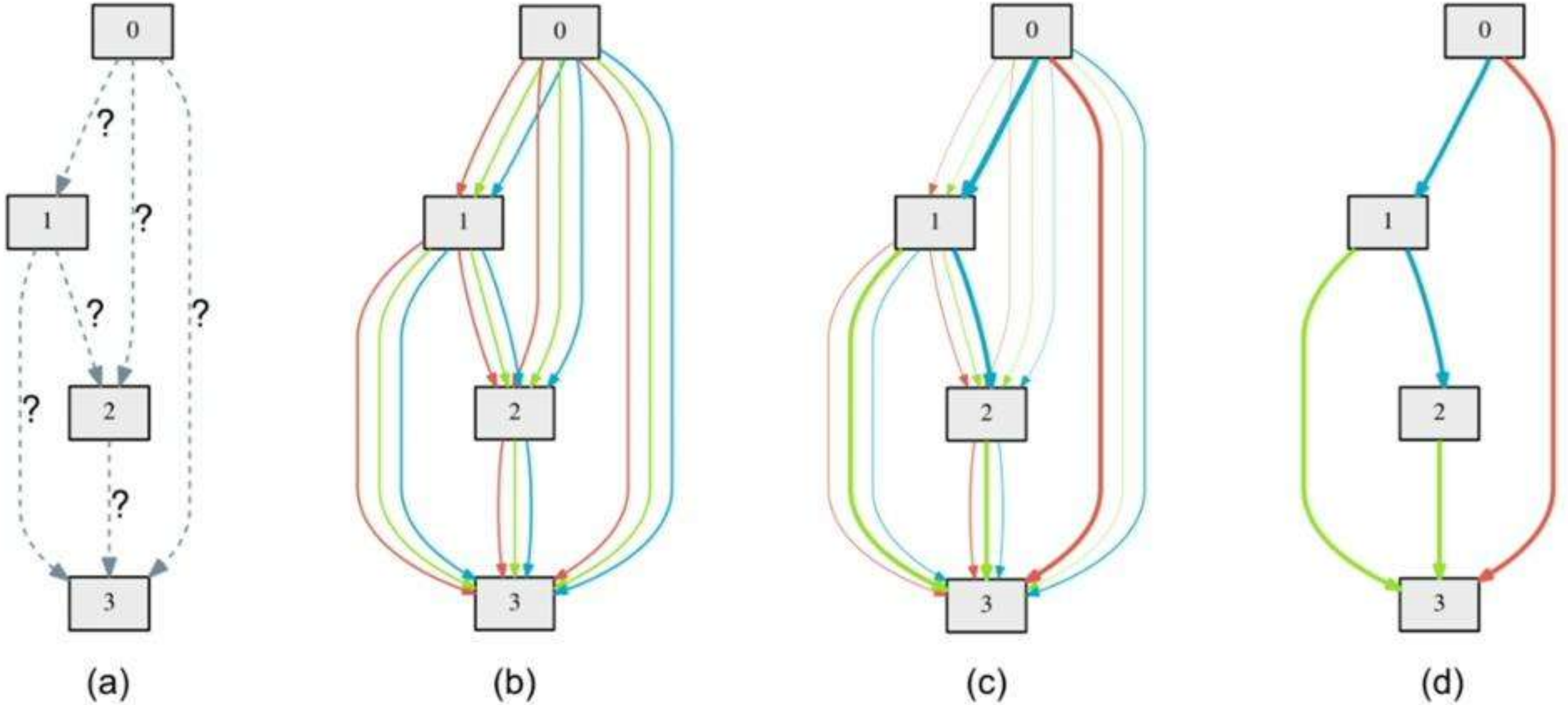
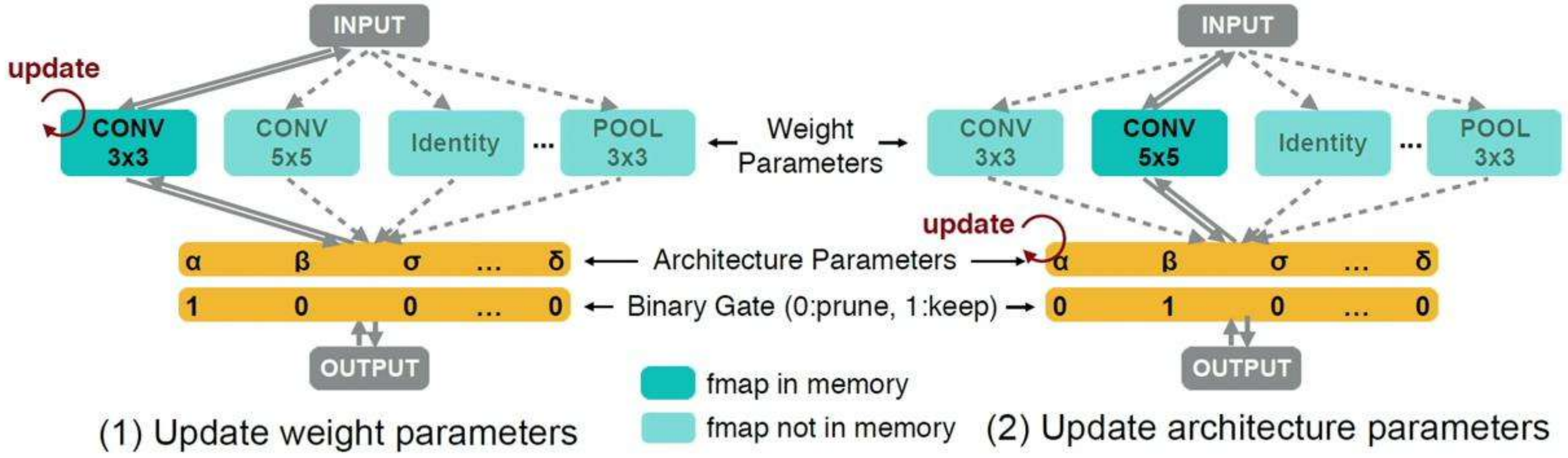
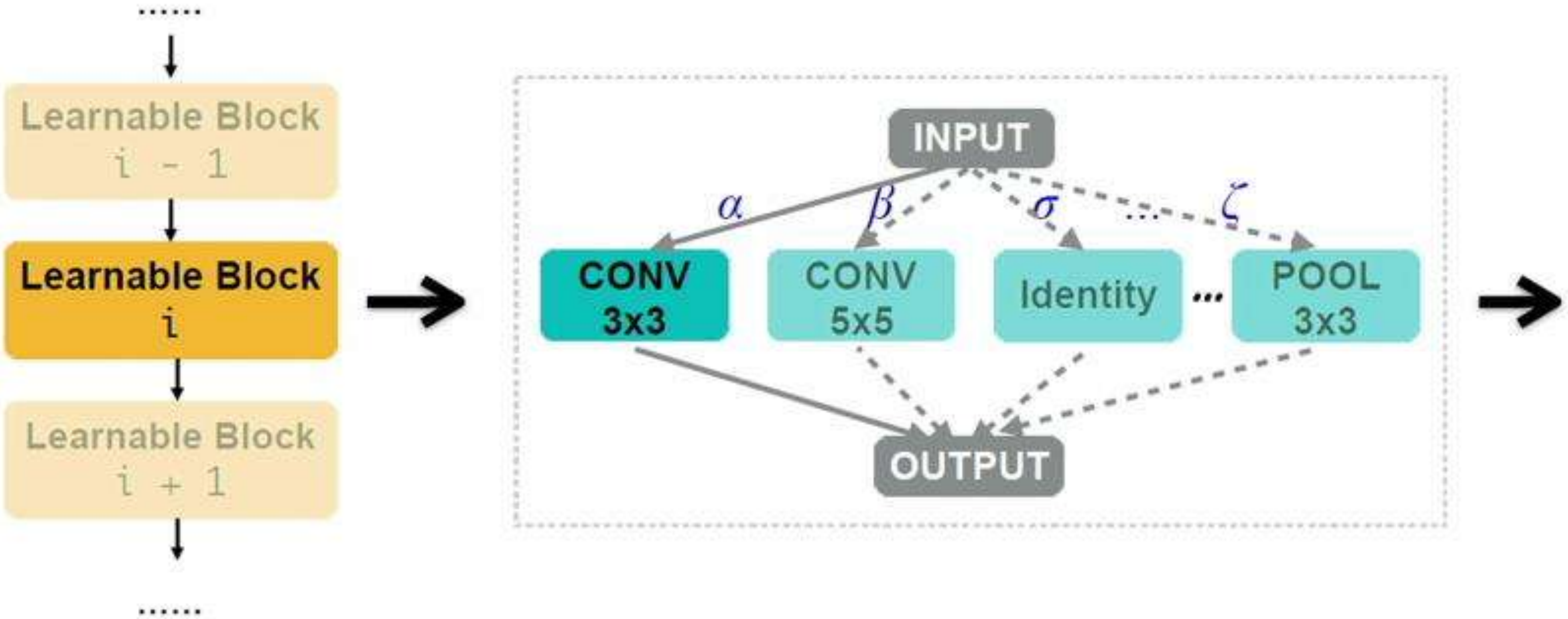


Figure 1: An overview of DARTS: (a) Operations on the edges are initially unknown. (b) Continuous relaxation of the search space by placing a mixture of candidate operations on each edge. (c) Joint optimization of the mixing probabilities and the network weights by solving a bilevel optimization problem. (d) Inducing the final architecture from the learned mixing probabilities.

ProxylessNas: Direct Neural Architecture Search on Target Task and Hardware (Cai et al. 2018)



Fixes the memory problems in DARTS



$$
\begin{aligned}
\mathbb{E}[\text{Latency}] = \; & \alpha \times F(\text{conv\_3x3}) + \\
& \beta \times F(\text{conv\_5x5}) + \\
& \sigma \times F(\text{identity}) + \\
& \ldots\ldots \\
& \zeta \times F(\text{pool\_3x3})
\end{aligned}
$$

$$\mathbb{E}[\text{latency}] = \sum_i \mathbb{E}[\text{latency}_\text{i}]$$

$$Loss = Loss_{CE} + \lambda_1 ||w||_2^2 + \lambda_2 \mathbb{E}[\text{latency}]$$

Figure 3: Making latency differentiable by introducing latency regularization loss.

Background: Neural Architecture Search

Tutorial on Neural Architecture Search at Microsoft Machine Learning Day (October 17th, 2018):
Neural Architecture Search: State-of-the-art Overview (Youtube), Debadeepta Dey

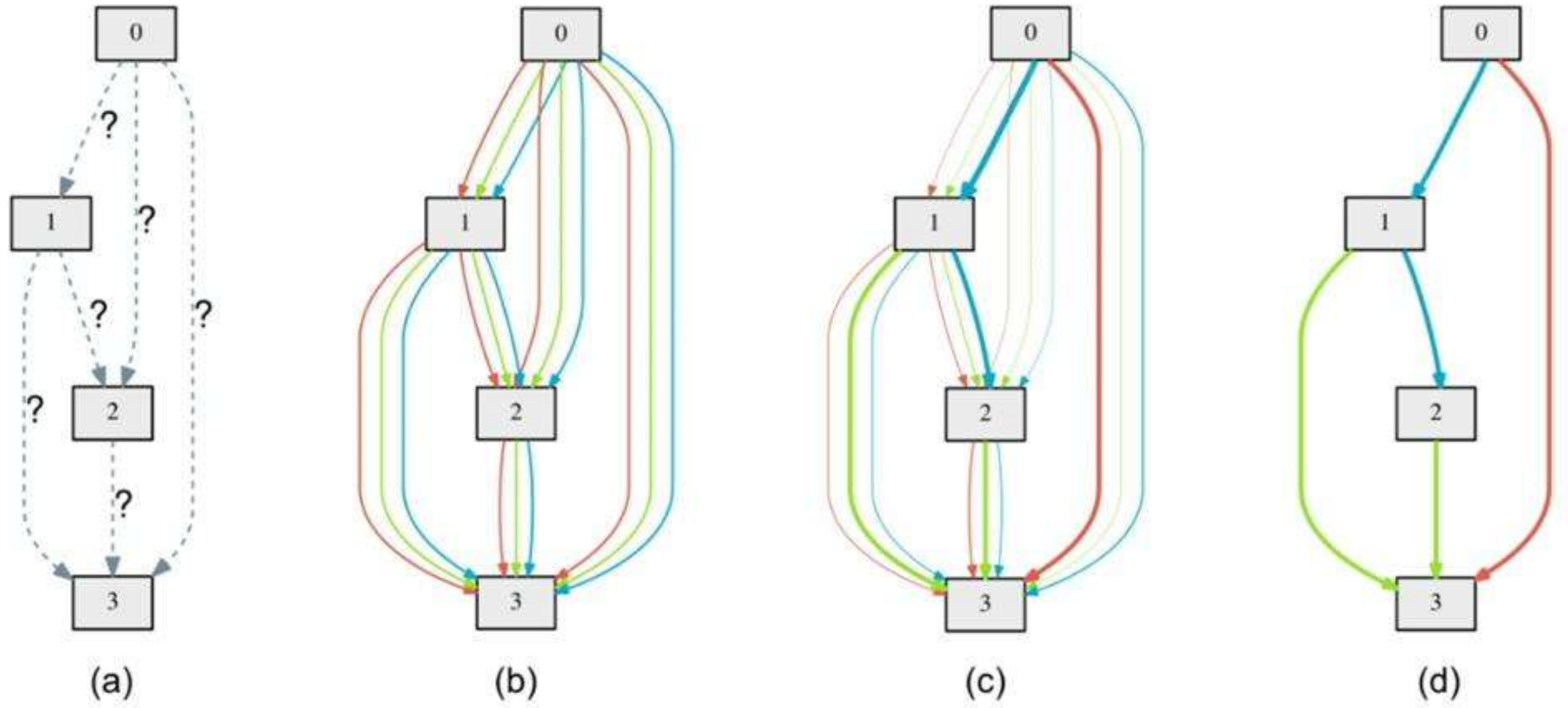


Figure 1: An overview of DARTS: (a) Operations on the edges are initially unknown. (b) Continuous relaxation of the search space by placing a mixture of candidate operations on each edge. (c) Joint optimization of the mixing probabilities and the network weights by solving a bilevel optimization problem. (d) Inducing the final architecture from the learned mixing probabilities.

Background: Neural Architecture Search

Tutorial on Neural Architecture Search at Microsoft Machine Learning Day (October 17th, 2018):
Neural Architecture Search: State-of-the-art Overview (Youtube), Debadeepta Dey

Project Petridish: Efficient Forward Architecture Search

Paper: https://arxiv.org/abs/1905.13360v1 (NeurIPS 2019)

Code: https://github.com/microsoft/petridishnn (TensorFlow currently, PyTorch coming soon)

Hanzhang Hu, John Langford, Rich Caruana, Saurajit Mukherjee, Eric Horvitz, Debadeepta Dey

Motivation for Growing Networks

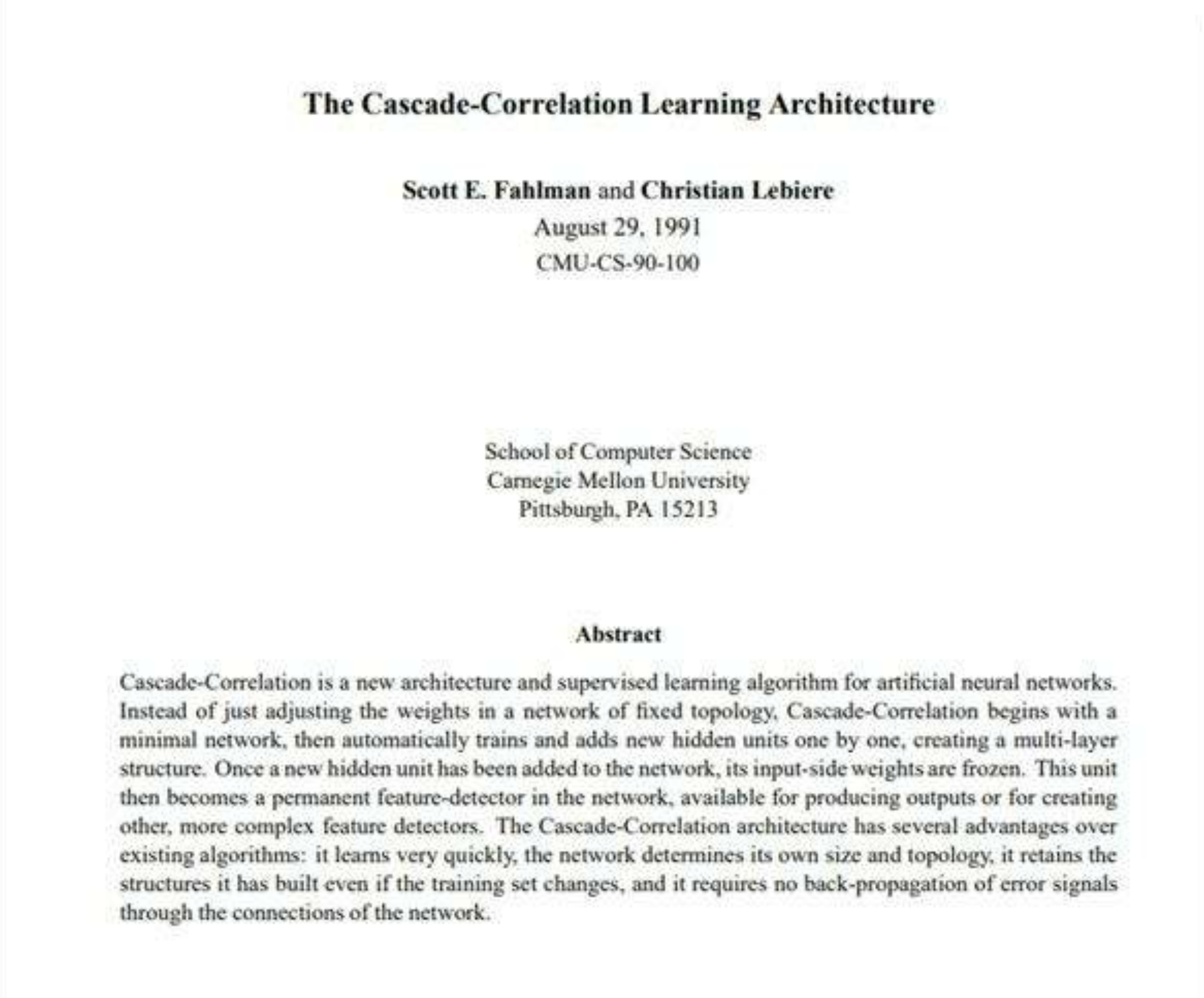
- Fully general method!
  - Cell-search not feasible when you don’t know good outer skeleton!
- Lifelong learning models.
  - New task/extra or changing data can be naturally incorporated.

Motivation for Growing Networks

- Fully general method!
  - Cell-search not feasible when you don’t know good outer skeleton!
- Lifelong learning models.
  - New task/extra or changing data can be naturally incorporated.
  - Can accommodate larger models.
- Exploit prior knowledge when available.
  - Explore a forest of models.
  - Warm start from existing models.
  - Universal post-processing for human-designed models!
  - Exploit information from similar models during search.

Project Petridish

- An overview of neural architecture search
- Method
  - Warm start
    - Inspired by gradient boosting.
  - Expand the search tree:
    - Focus on the most cost-effective ones.
    - Directly search the pareto-frontier.
  - Predict performance.
    - Utilizing model initialization to select children to train.



The Cascade-Correlation Learning Architecture

Scott E. Fahlman and **Christian Lebiere**

August 29, 1991

CMU-CS-90-100

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

Cascade-Correlation is a new architecture and supervised learning algorithm for artificial neural networks. Instead of just adjusting the weights in a network of fixed topology, Cascade-Correlation begins with a minimal network, then automatically trains and adds new hidden units one by one, creating a multi-layer structure. Once a new hidden unit has been added to the network, its input-side weights are frozen. This unit then becomes a permanent feature-detector in the network, available for producing outputs or for creating other, more complex feature detectors. The Cascade-Correlation architecture has several advantages over existing algorithms: it learns very quickly, the network determines its own size and topology, it retains the structures it has built even if the training set changes, and it requires no back-propagation of error signals through the connections of the network.

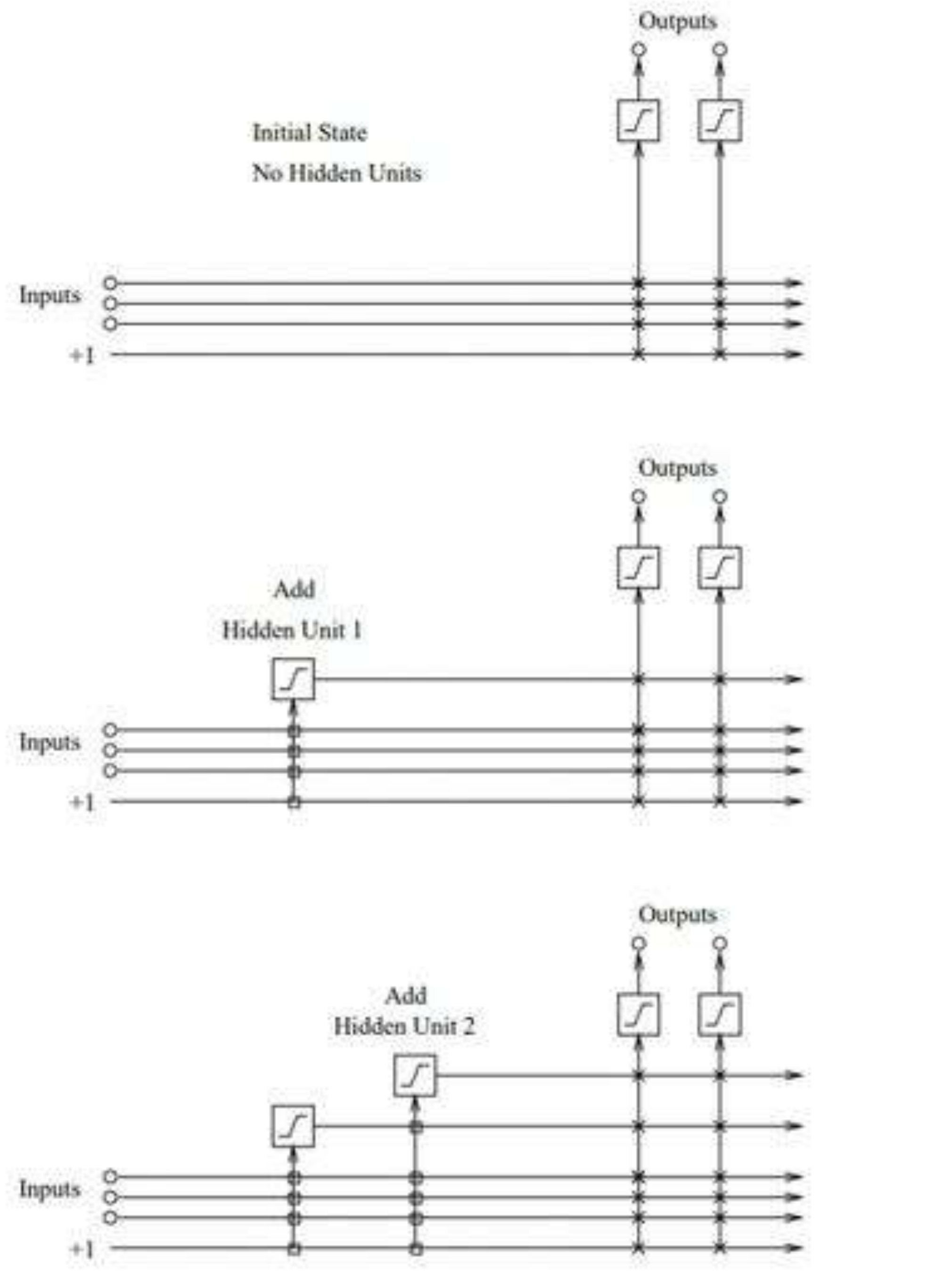


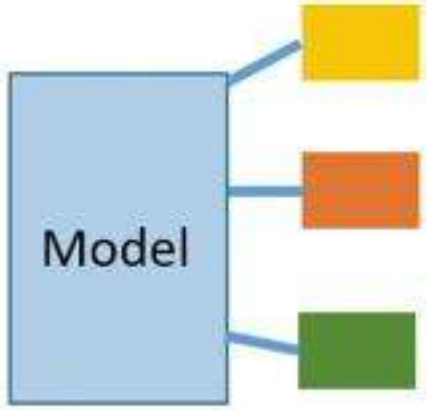
Figure 1: The Cascade architecture, initial state and after adding two hidden units. The vertical lines sum all incoming activation. Boxed connections are frozen, X connections are trained repeatedly.

Incremental Training

Phase 0
Original model

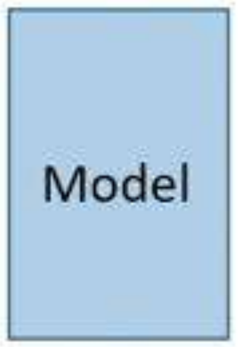
Incremental Training

Phase 0
Original model

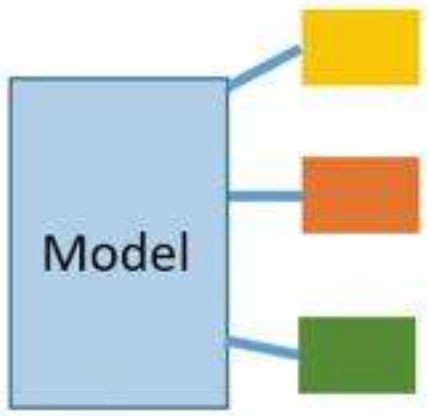


Phase 1
Initialize candidates,
but do not allow candidates
to affect the original model.

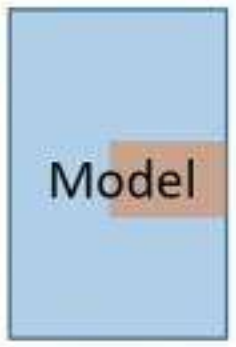
Incremental Training



Phase 0
Original model

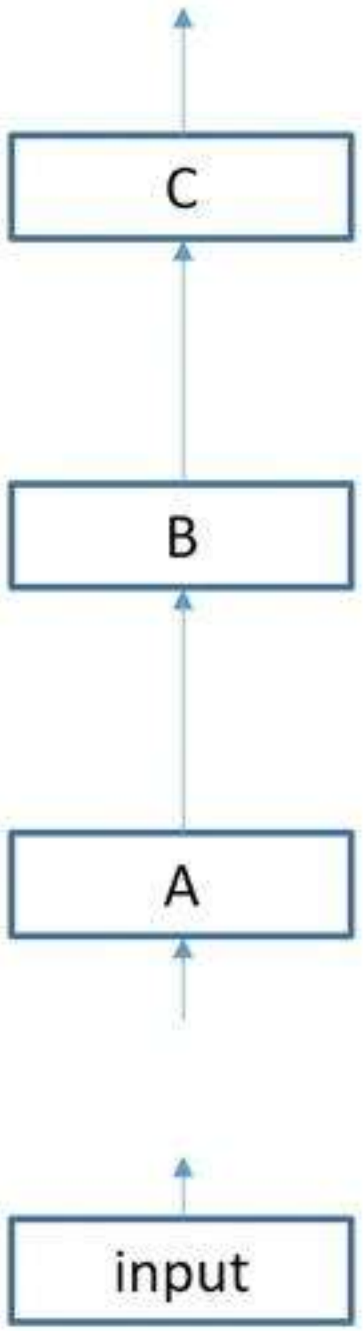


Phase 1
Initialize candidates,
but do not allow candidates
to affect the original model.



Phase 2
Officially add an
candidate to model.
Now the candidate can
affect the original.

Incremental Training



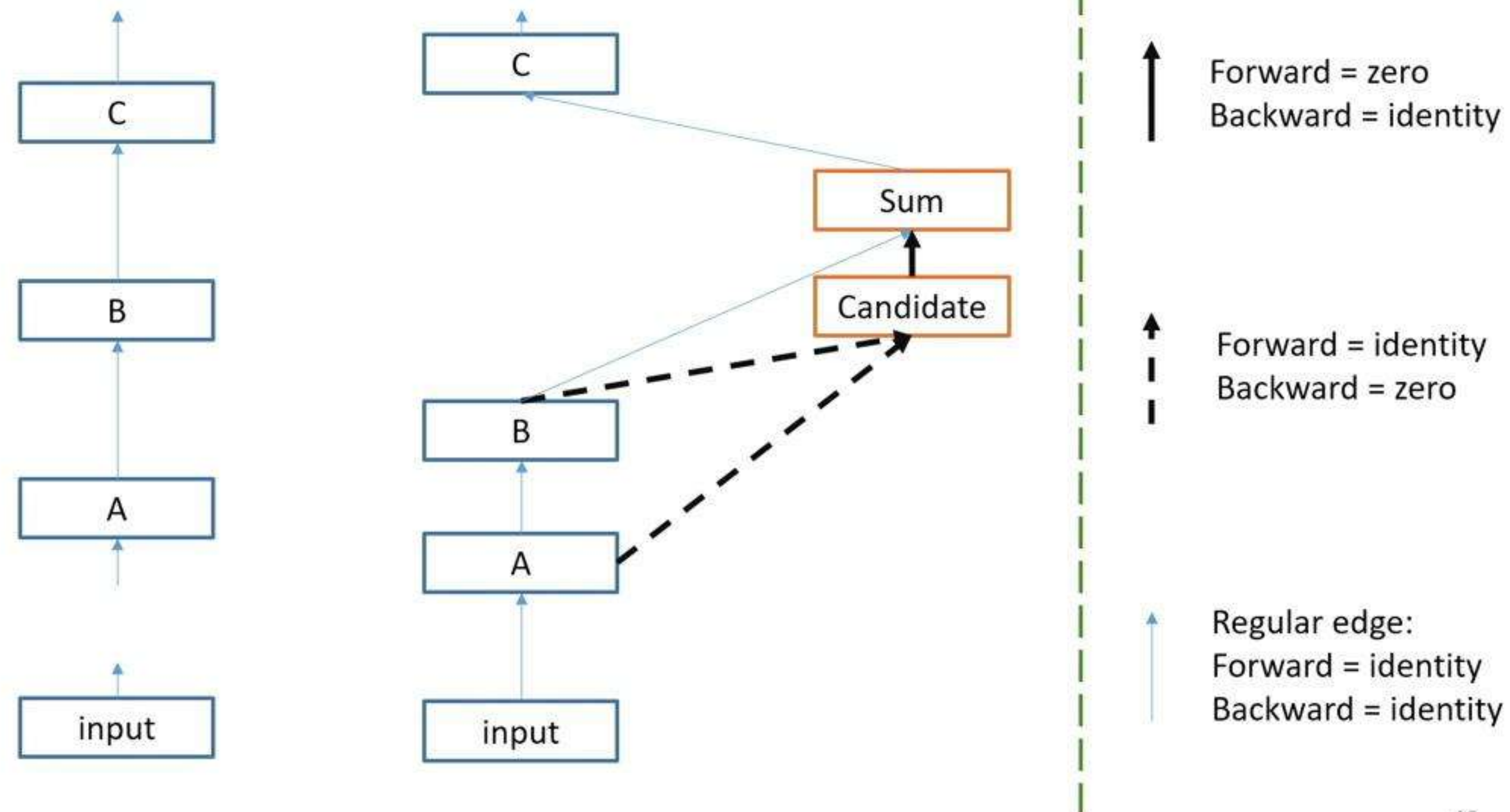
Original model

↑ Forward = zero
Backward = identity

⇡ Forward = identity
Backward = zero

↑ Regular edge:
Forward = identity
Backward = identity

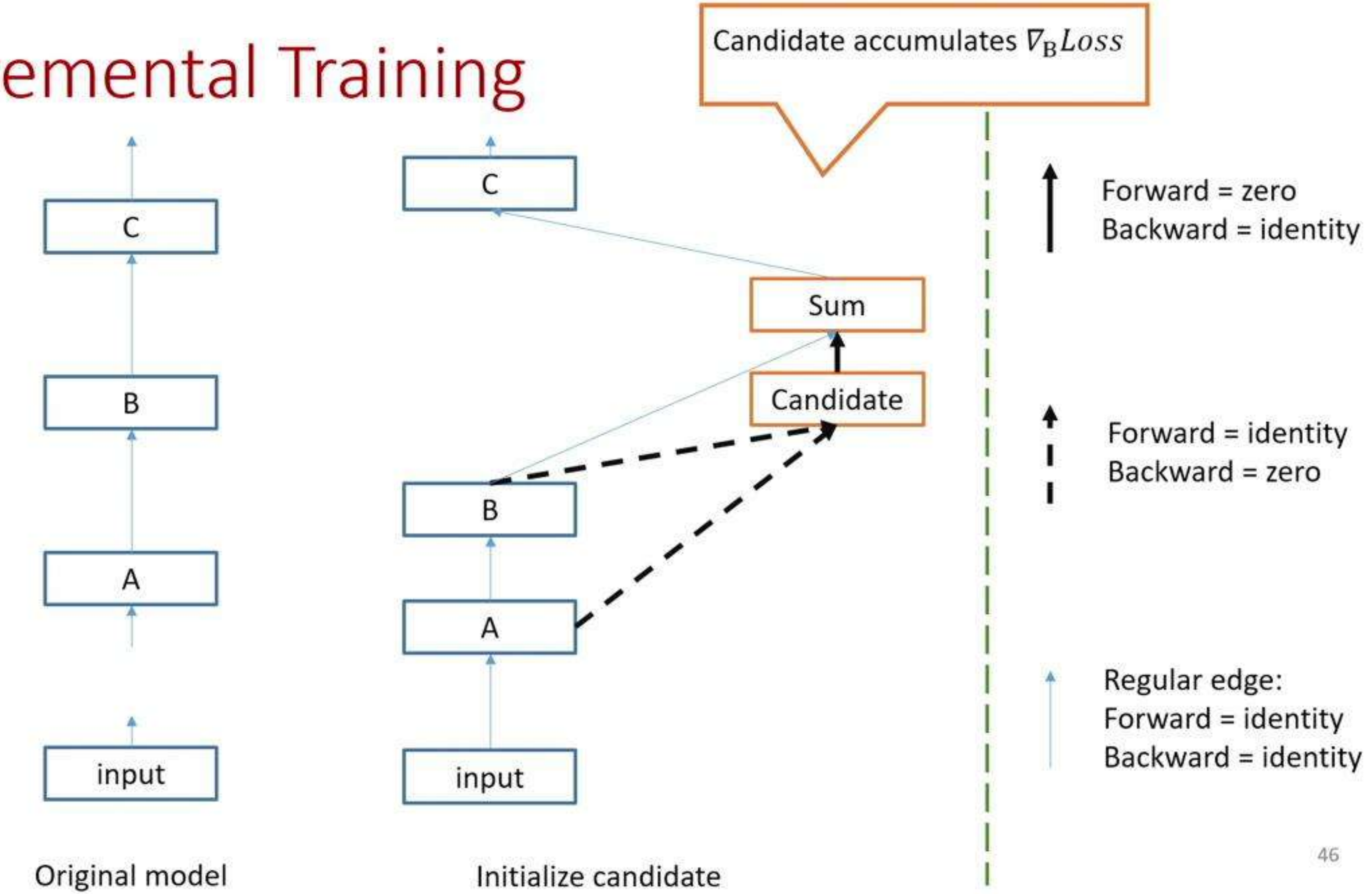
Incremental Training



Original model

Initialize candidate

Incremental Training



Candidate accumulates $\nabla_{\mathrm{B}} Loss$

Forward = zero
Backward = identity

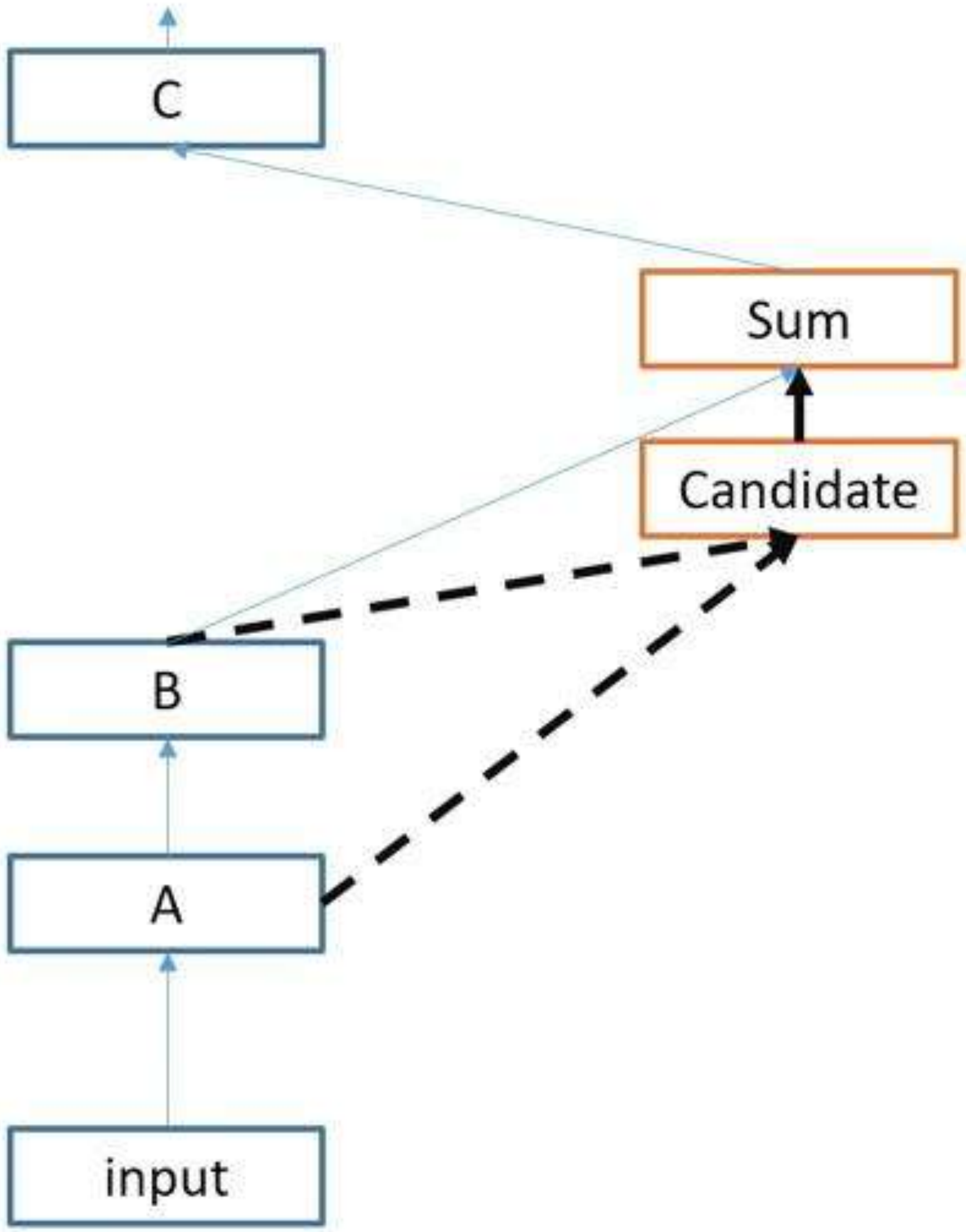
Forward = identity
Backward = zero

Regular edge:
Forward = identity
Backward = identity

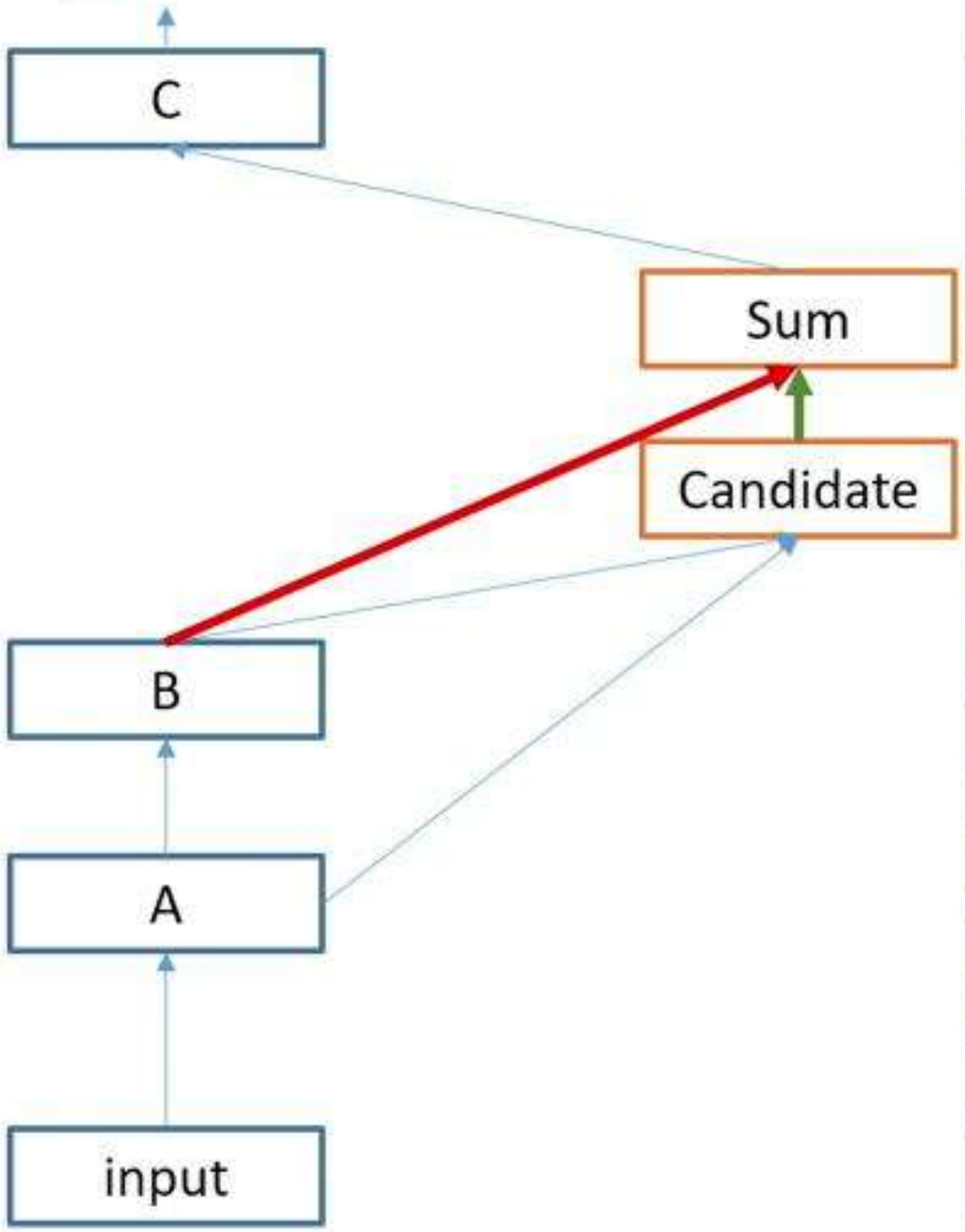
Original model

Initialize candidate

Incremental Training



Initialize candidate

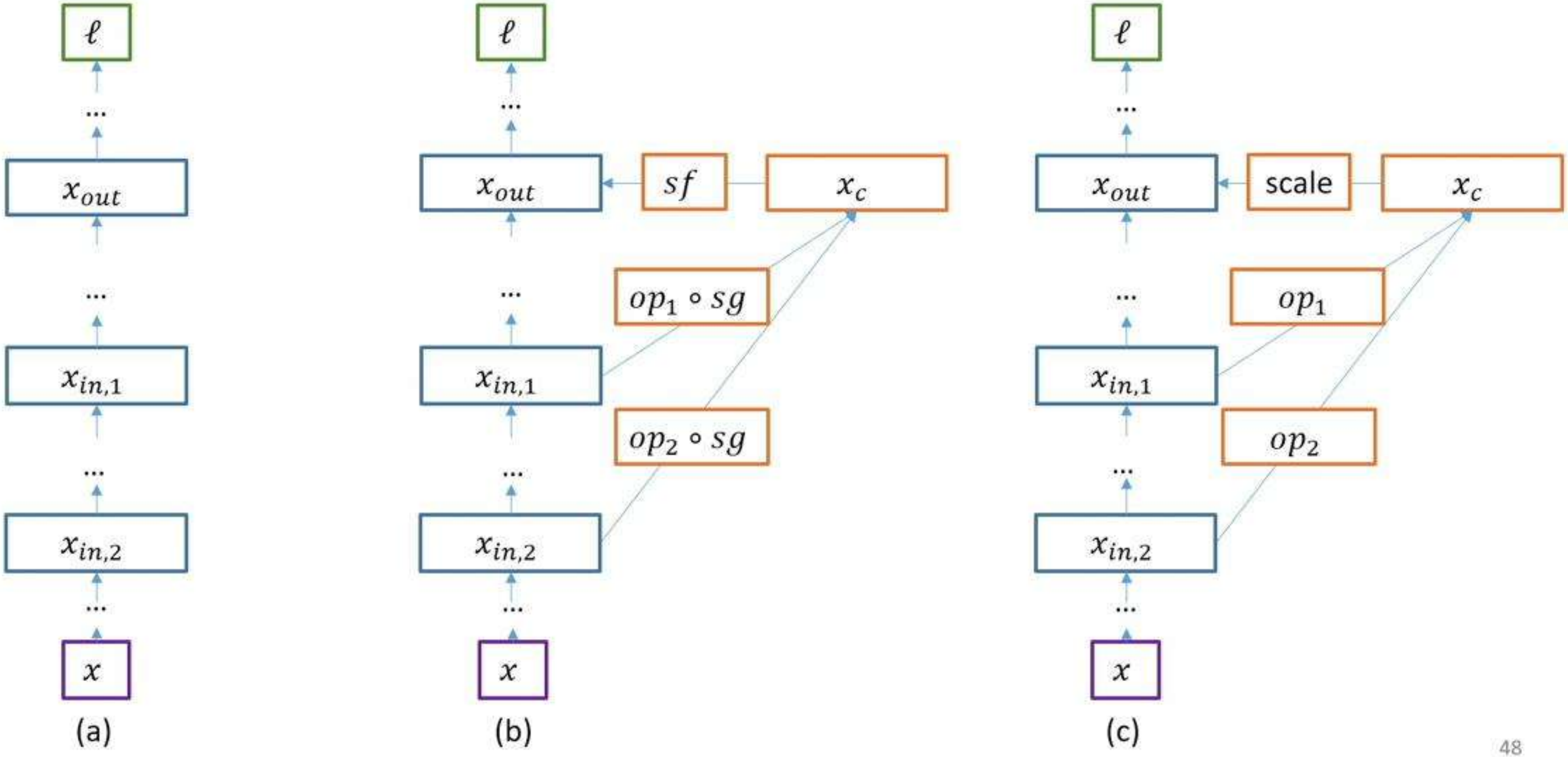


Officially add candidate to model

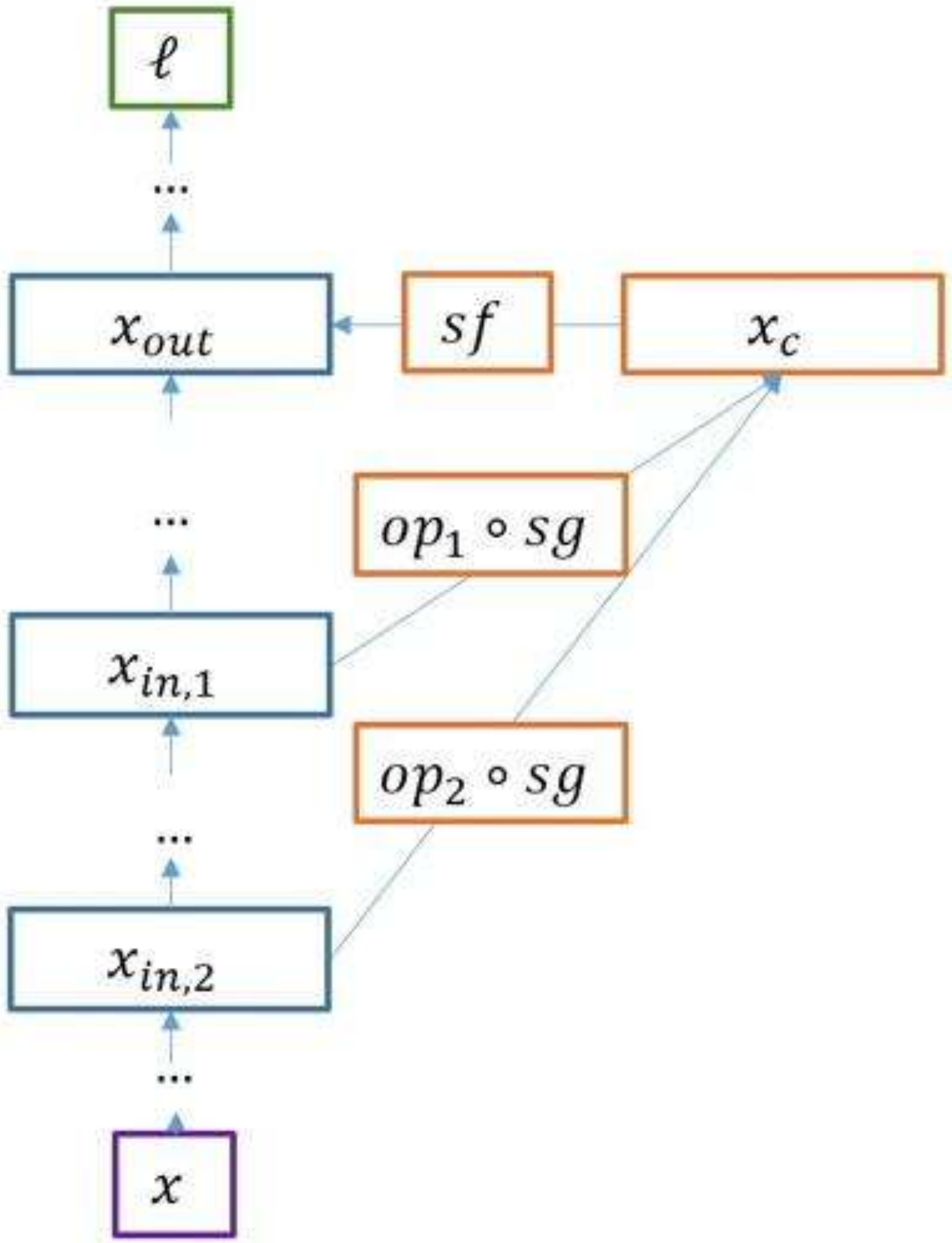
🟥 Scale the input.
Initial scale = 1

🟩 Scale the input.
Initial scale = 0

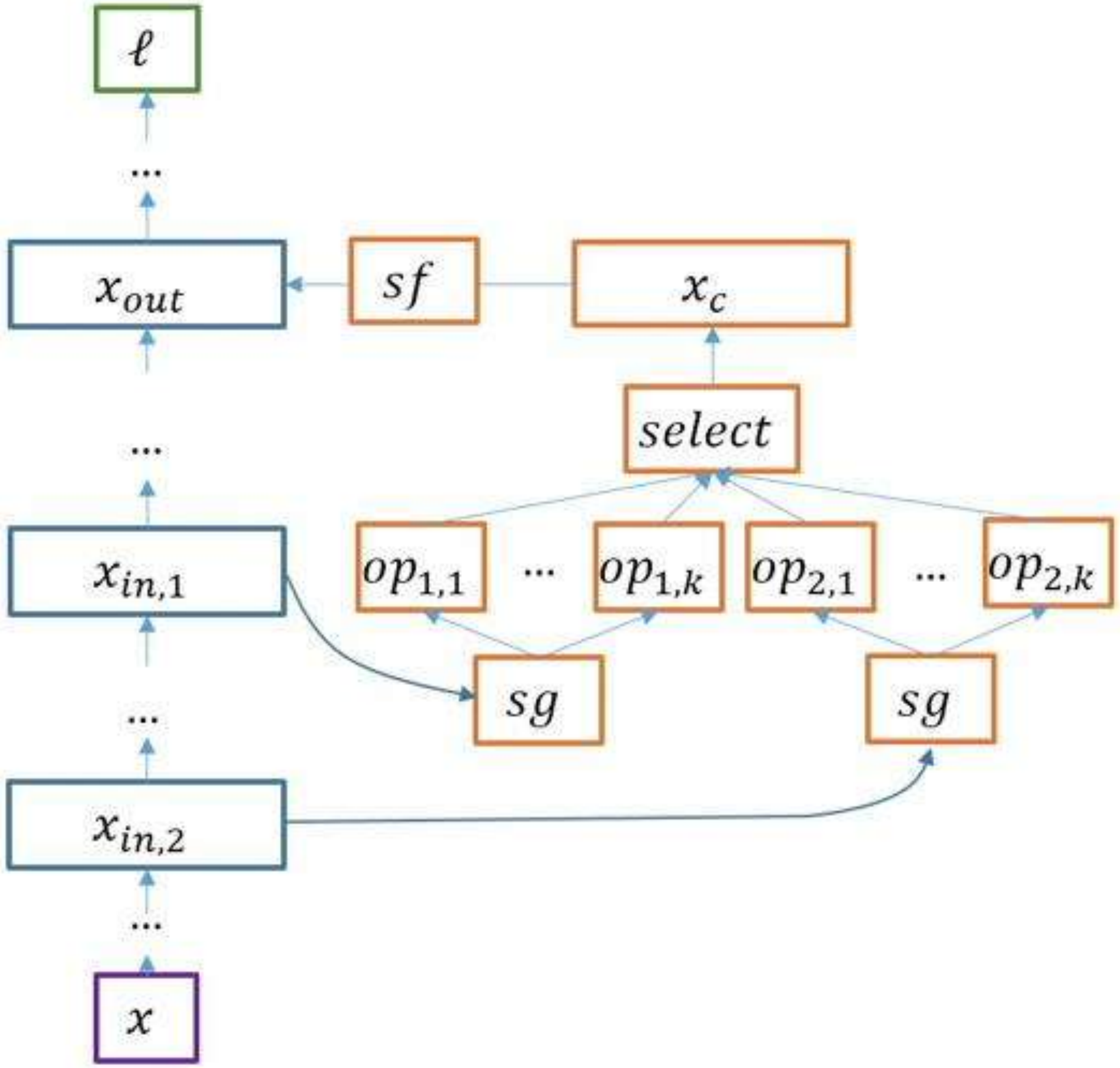
Incremental Training (Summary)



Incremental Training (Choice of Candidates)

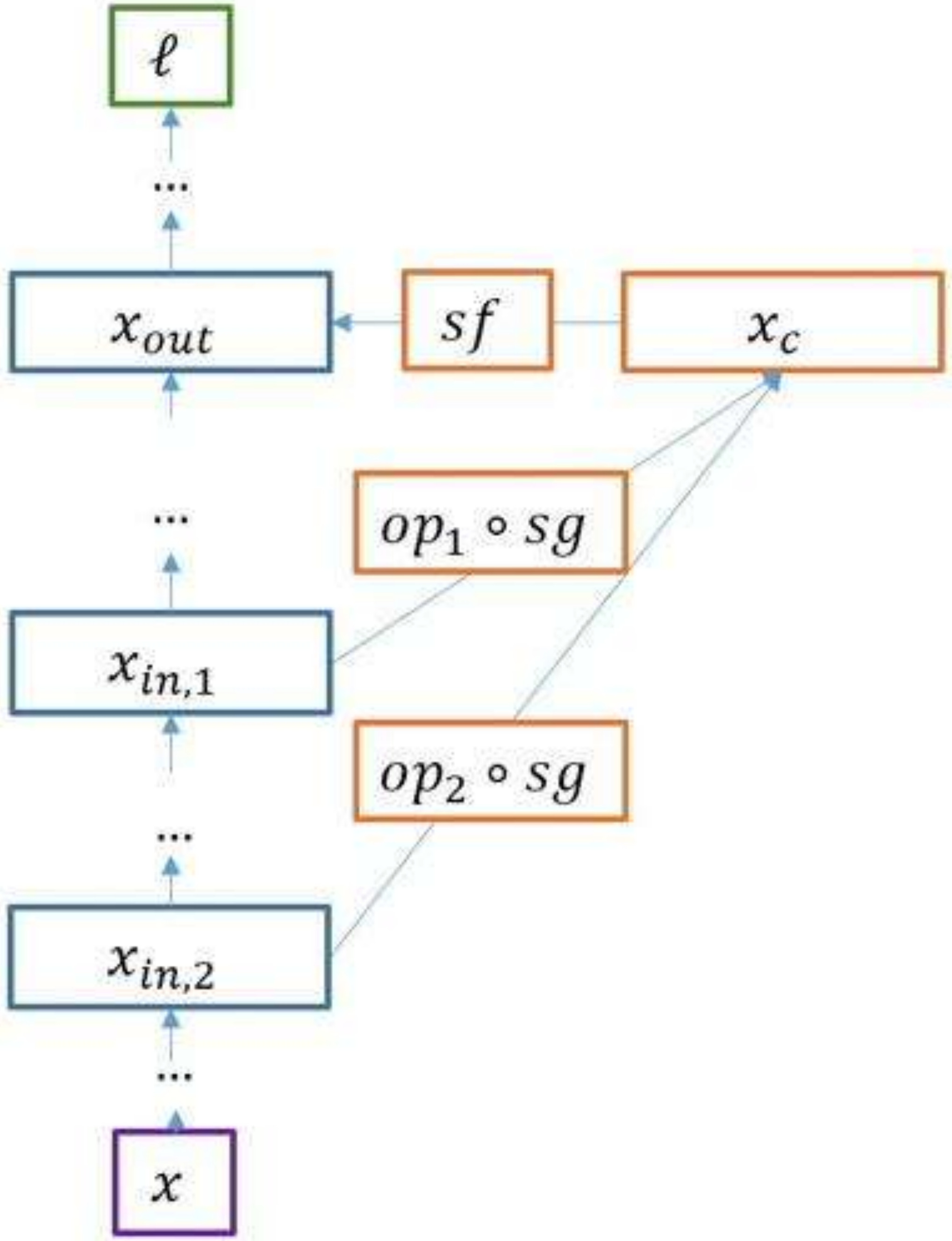


(b)

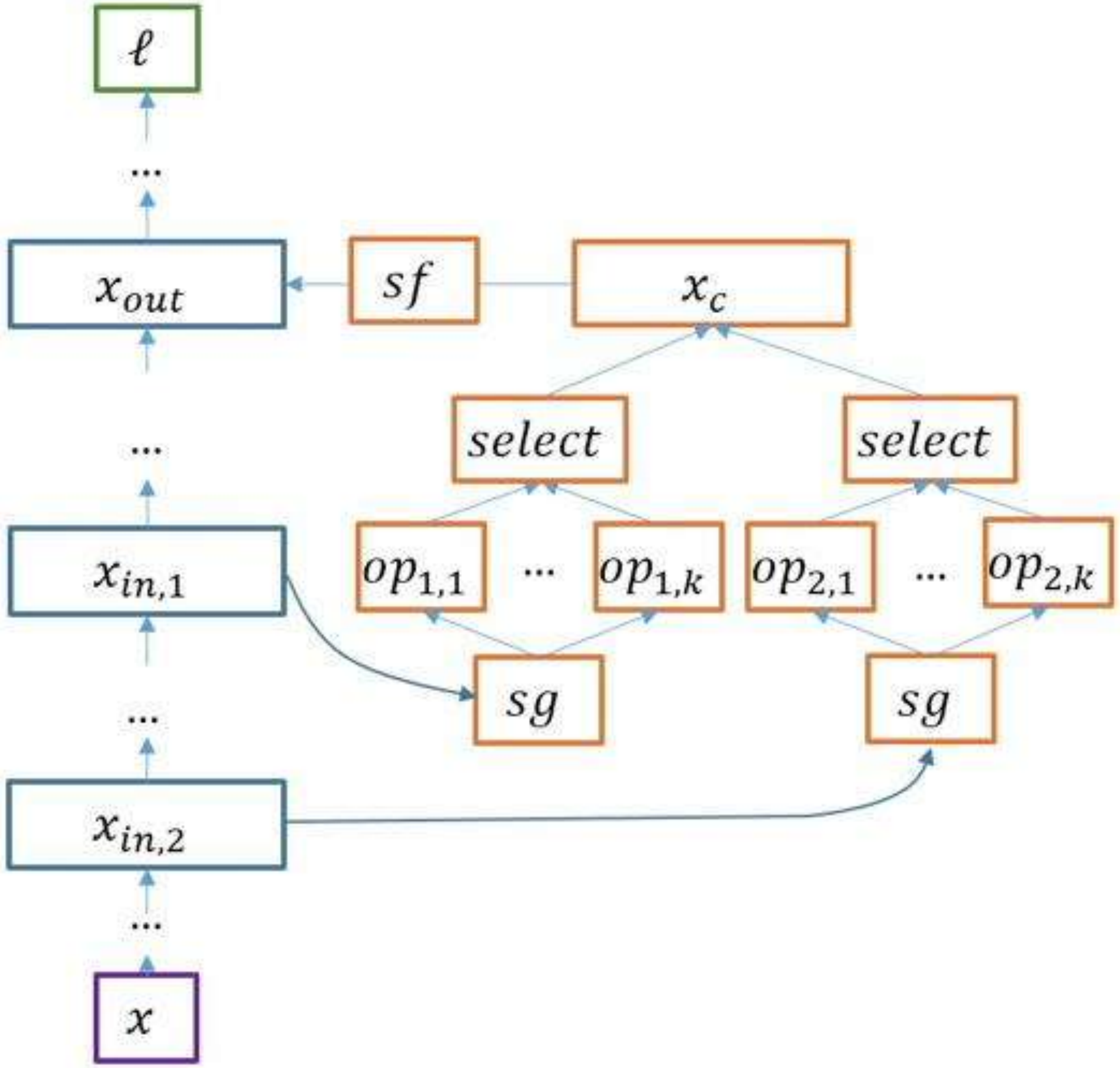


(b’)

Incremental Training (Choice of Candidates)



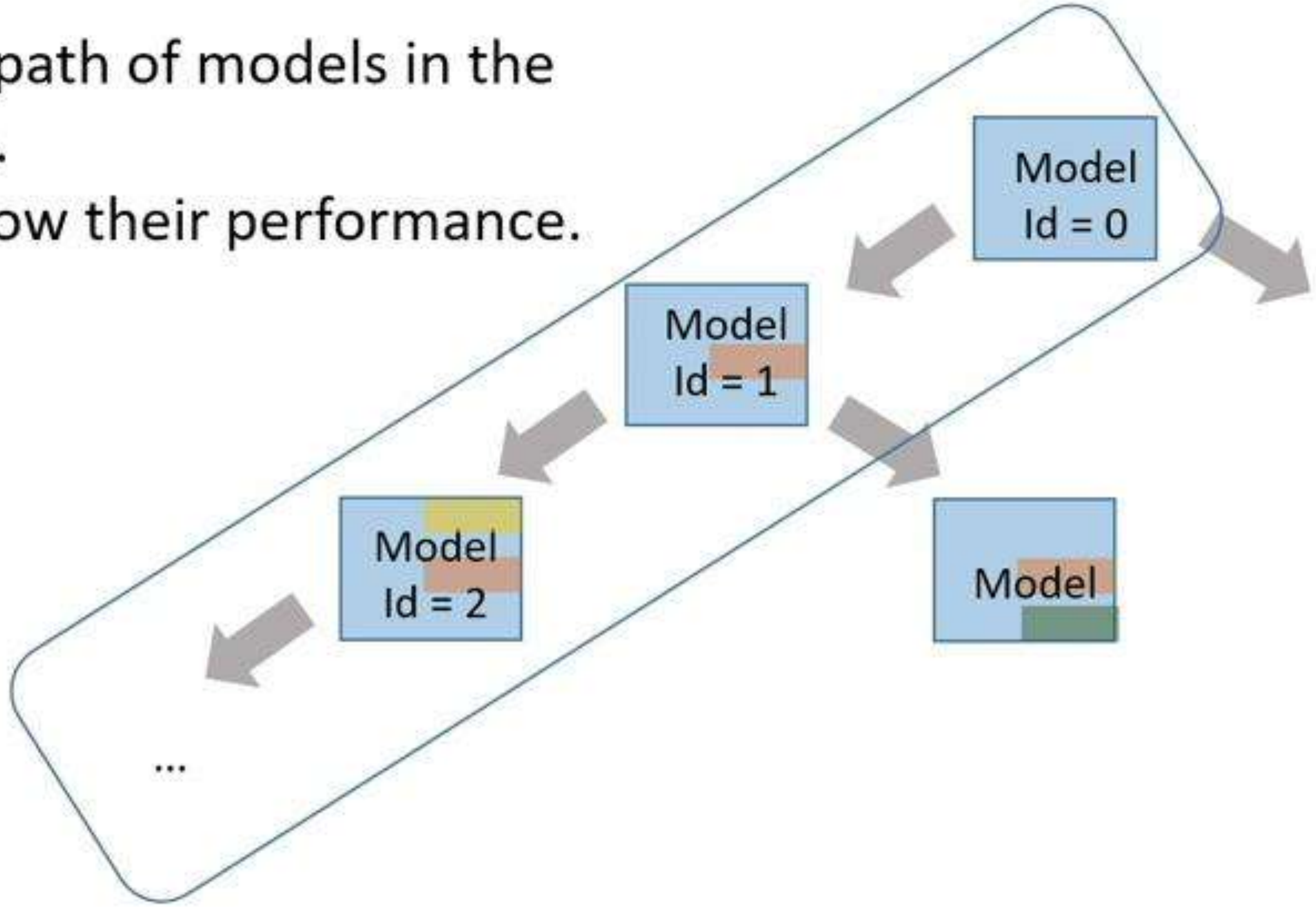
(b)



(b')

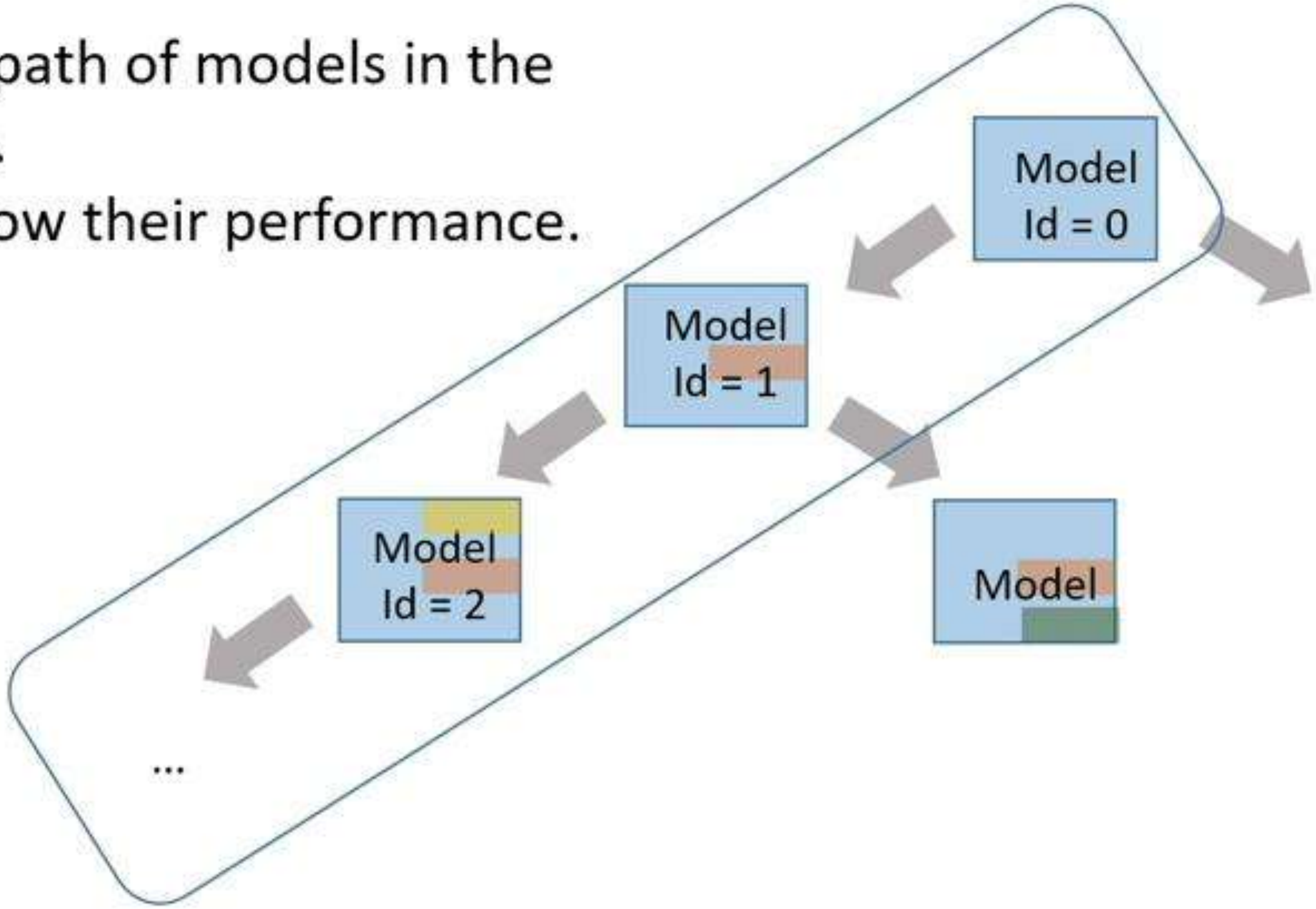
Incremental training during search

Consider a path of models in the search tree.
Want to know their performance.



Incremental training during search

Consider a path of models in the search tree.
Want to know their performance.



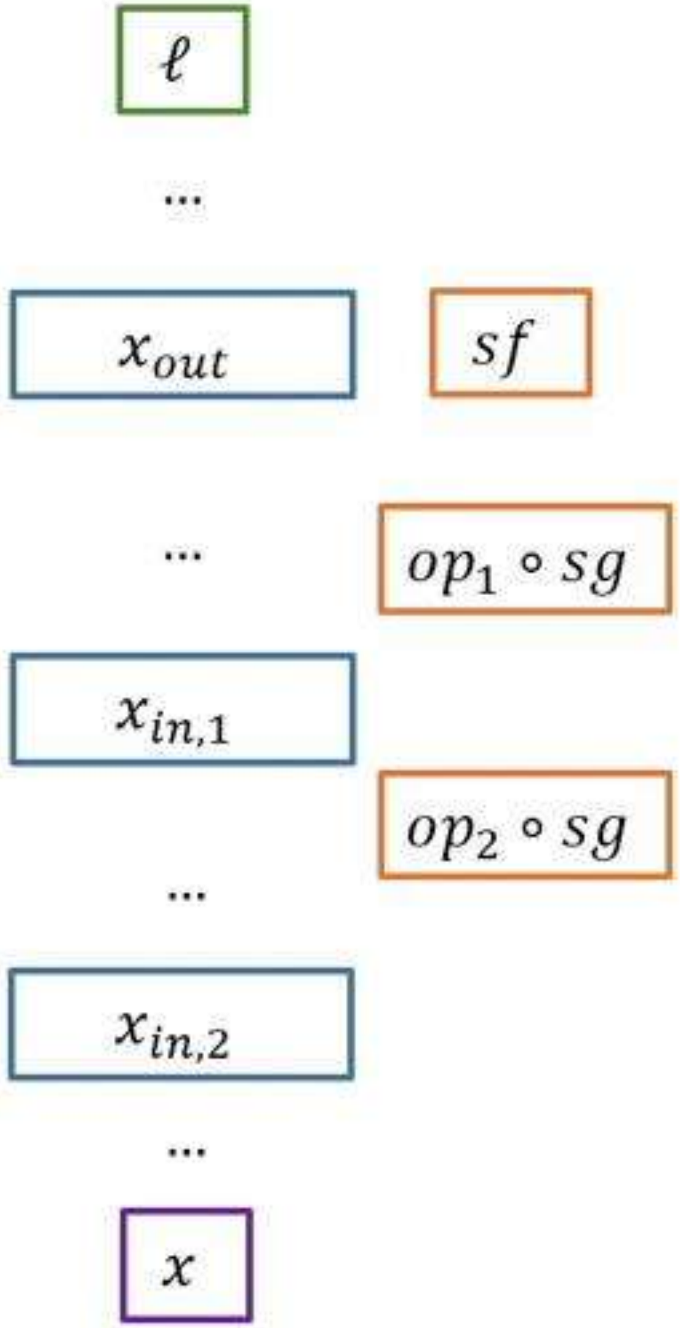
Option 1 (From-scratch) :

- Train models independently.
- 300 epochs per model

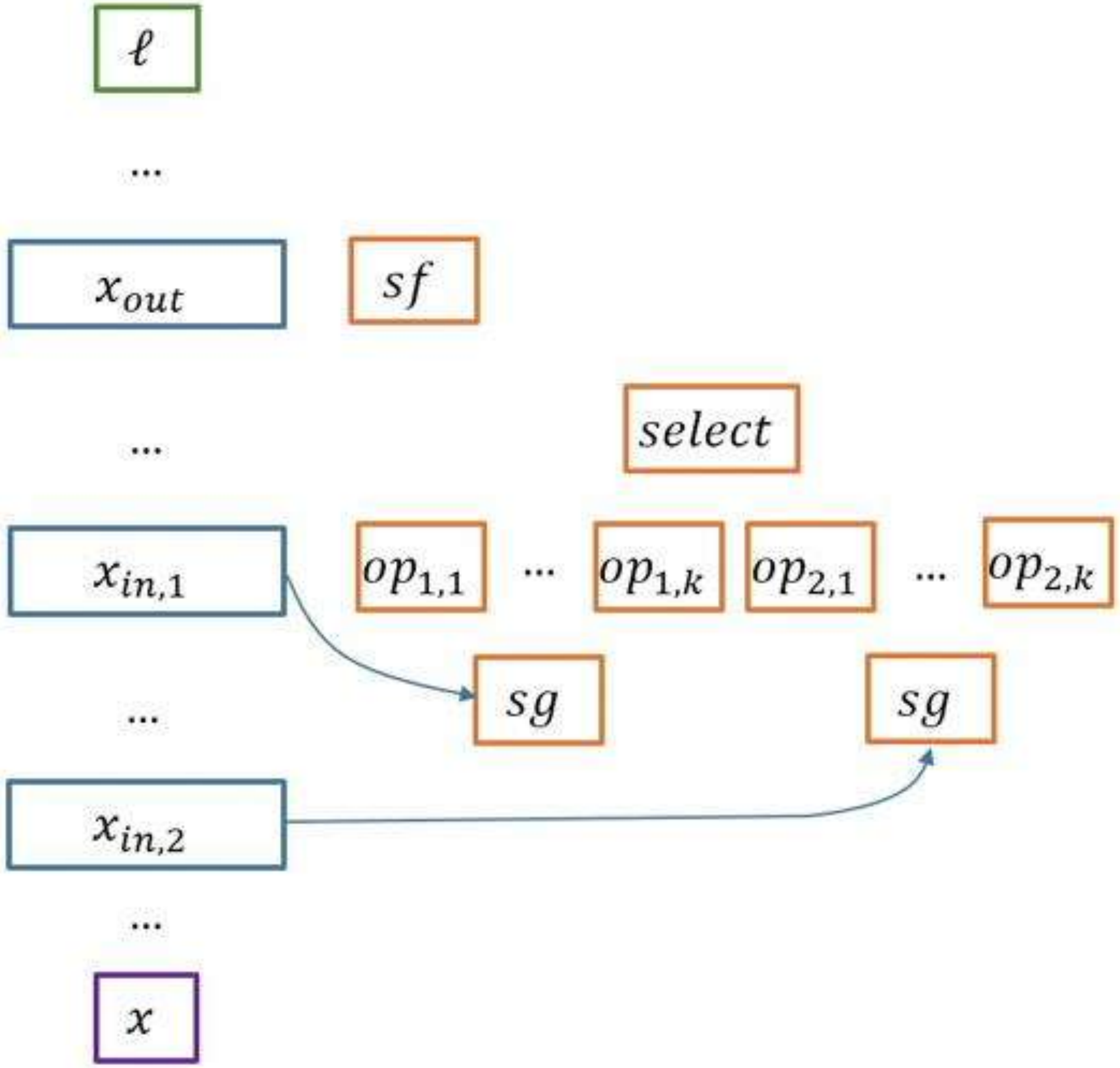
Option 2 (Incremental) :

- Start from parent; initialize children
- 40 epochs per model

Incremental Training (Choice of Candidates)

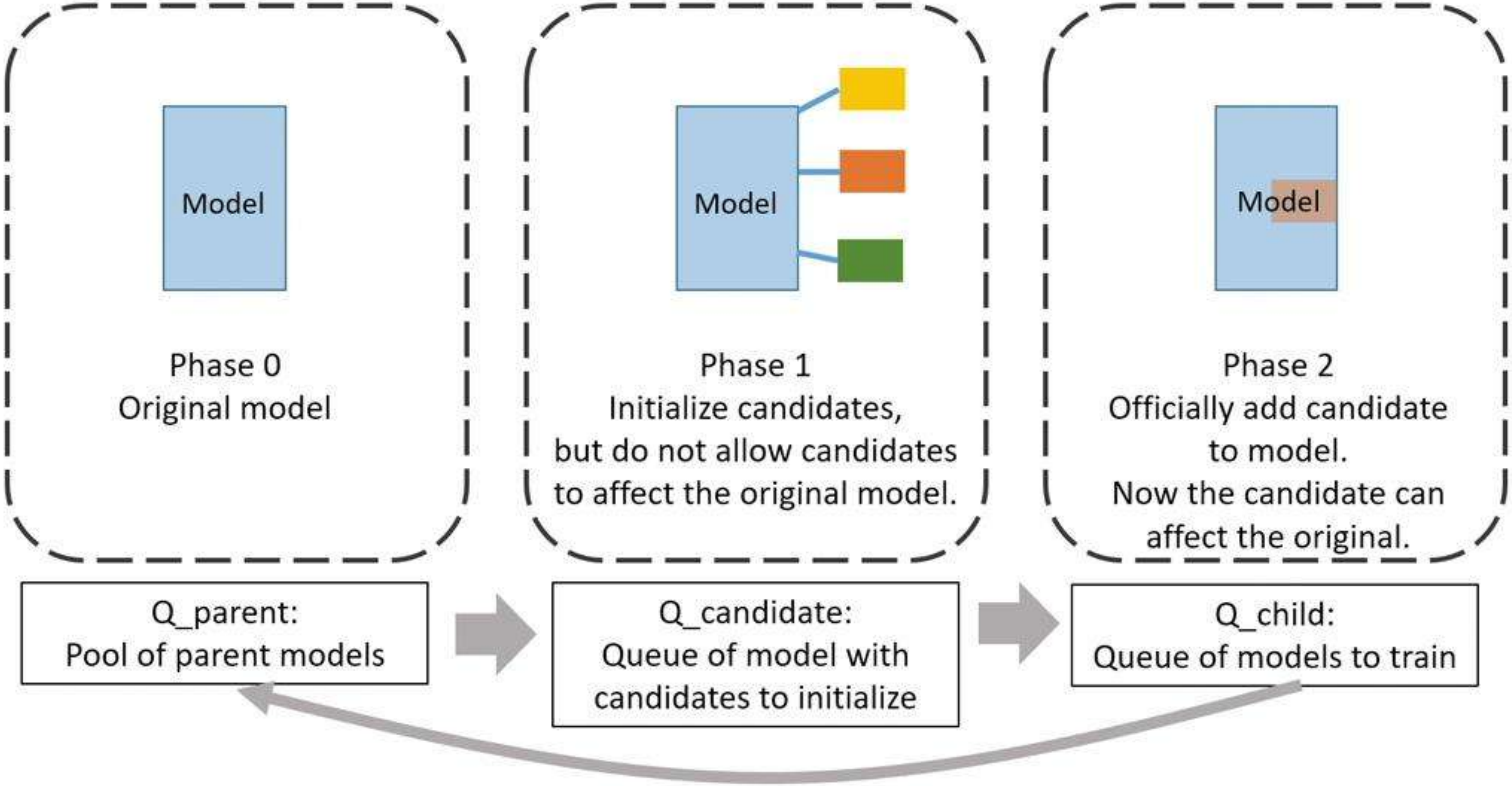


(b)



(b')

Search on distributed systems



Phase 0
Original model

Phase 1
Initialize candidates,
but do not allow candidates
to affect the original model.

Phase 2
Officially add candidate
to model.
Now the candidate can
affect the original.

Q_parent:
Pool of parent models

Q_candidate:
Queue of model with
candidates to initialize

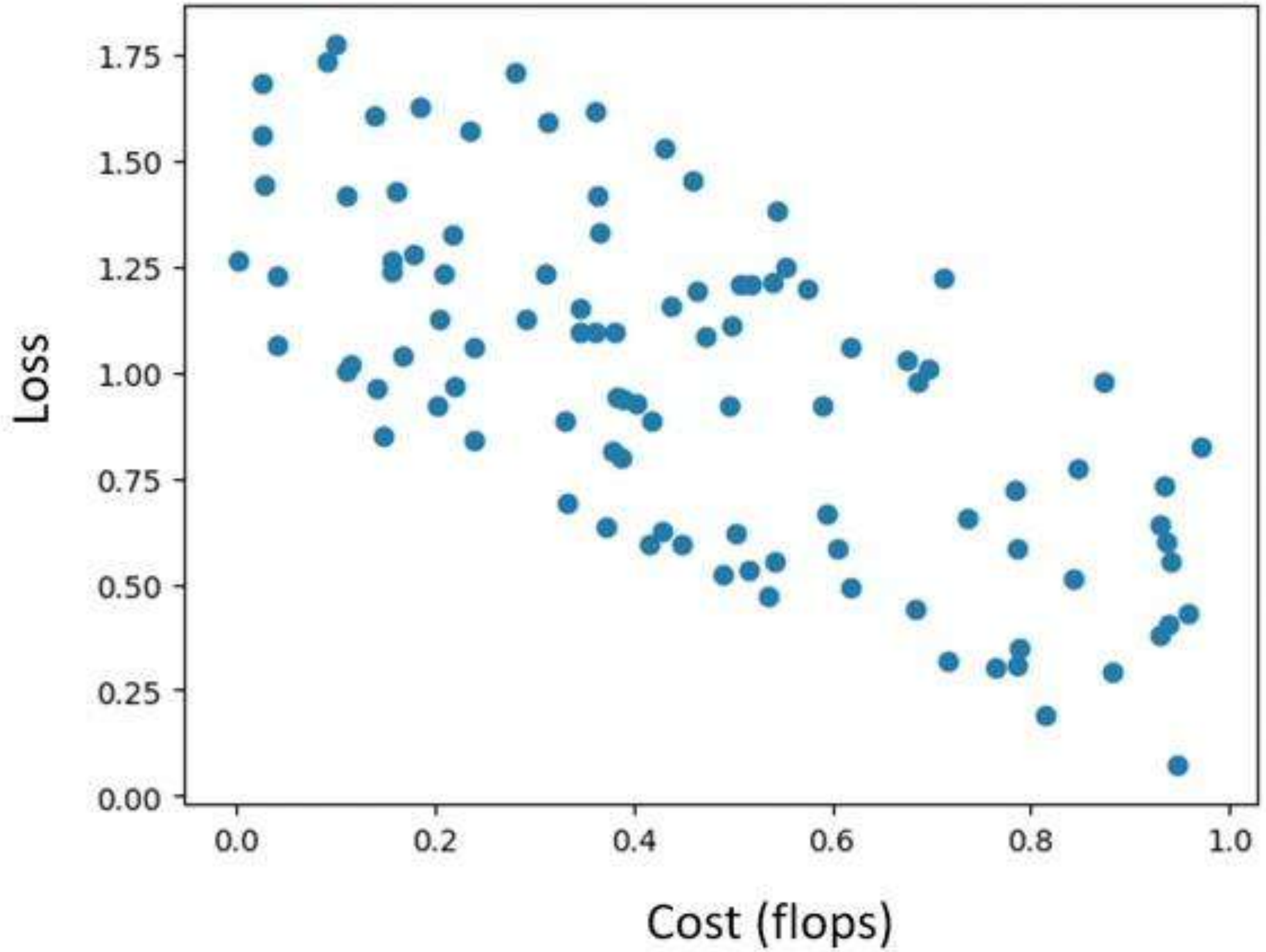
Q_child:
Queue of models to train

Search on distributed systems

- Q_parent: explore-exploit a diverse set of good models to extend.
- Q_candidate: initialize promising candidates
- Q_children: train promising children

- How do we know a model is good?

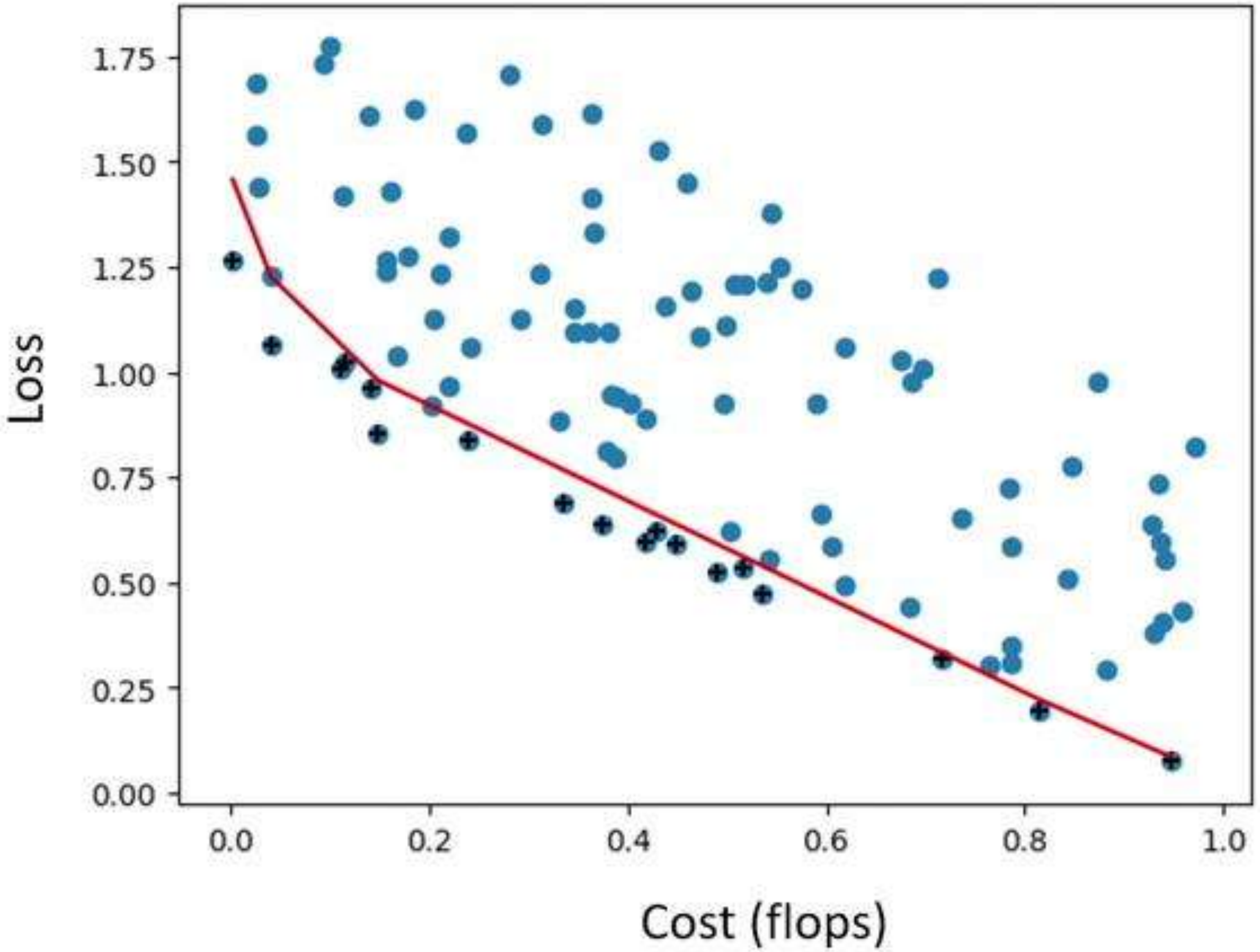
Expanding the Most Cost-efficient Models



This figure is for illustration only

Expanding the Most Cost-efficient Models

- Epsilon- convex hull



Key advantage:

Method naturally produces a 'gallery' of models which are nearly-optimal for every serving time budget need.

This is critical to production serving needs.

Results

Reproducibility and fair comparison crisis!

- Nearly impossible to compare algorithms due to differences in
  - Search spaces
  - Training routine used (does it have all the tips and tricks?)
  - Hardware used (TPU vs. GPU vs. driver version vs. cuda version vs.....)
  - Stochasticity in training on gpus.
  - Multi-objective optimization problem (flops, memory, latency, accuracy)
- Community working to establish standard benchmark
  - NASBench-101
    - Cannot evaluate weight-sharing, DARTS-like search spaces
    - More benchmarks are coming soon ;-)

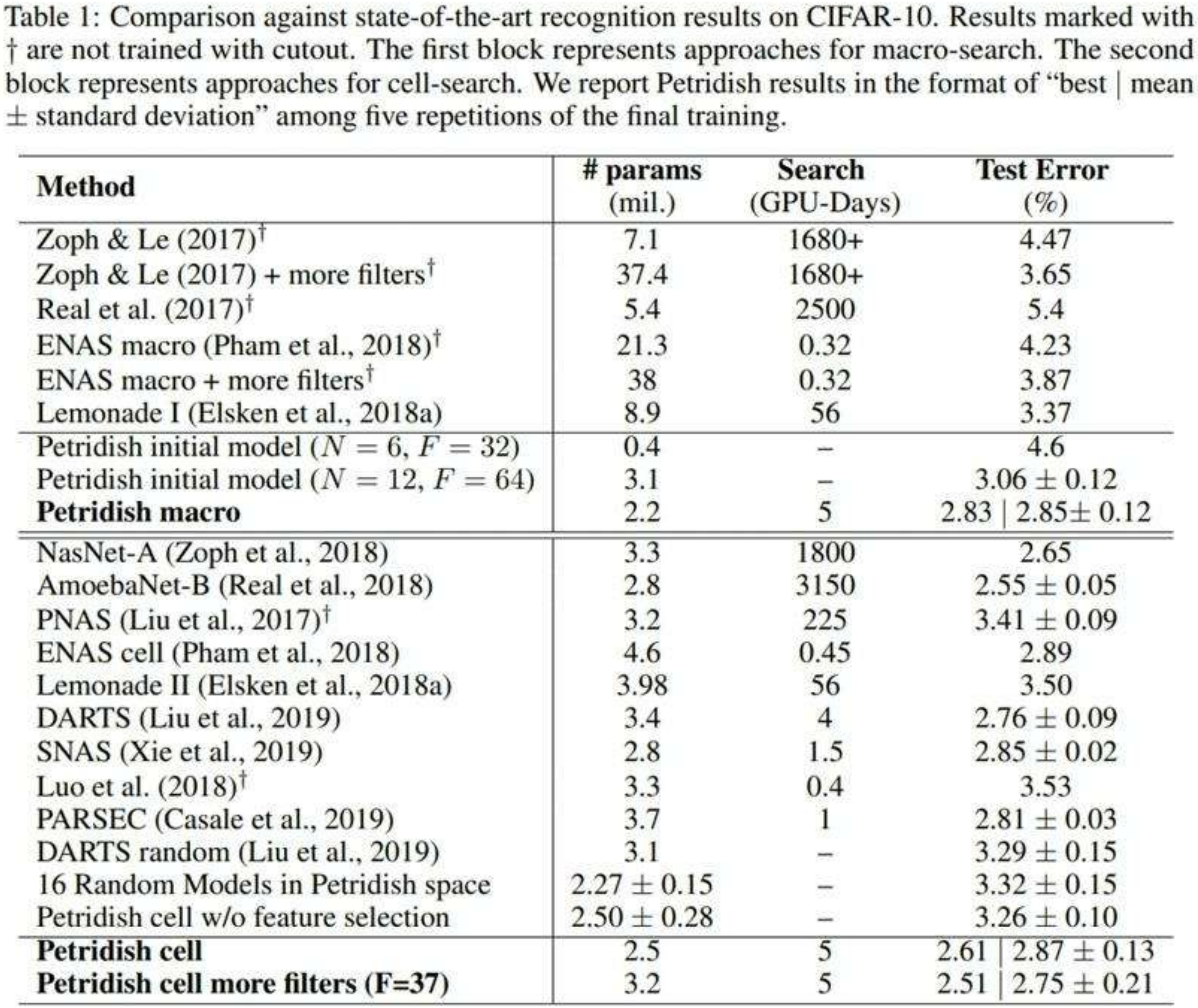


Table 1: Comparison against state-of-the-art recognition results on CIFAR-10. Results marked with † are not trained with cutout. The first block represents approaches for macro-search. The second block represents approaches for cell-search. We report Petridish results in the format of "best | mean ± standard deviation" among five repetitions of the final training.

Method	**# params** (mil.)	**Search** (GPU-Days)	**Test Error** (%)
Zoph & Le (2017)†	7.1	1680+	4.47
Zoph & Le (2017) + more filters†	37.4	1680+	3.65
Real et al. (2017)†	5.4	2500	5.4
ENAS macro (Pham et al., 2018)†	21.3	0.32	4.23
ENAS macro + more filters†	38	0.32	3.87
Lemonade I (Elsken et al., 2018a)	8.9	56	3.37
Petridish initial model ($N = 6$, $F = 32$)	0.4	–	4.6
Petridish initial model ($N = 12$, $F = 64$)	3.1	–	3.06 ± 0.12
Petridish macro	2.2	5	2.83 | 2.85± 0.12
NasNet-A (Zoph et al., 2018)	3.3	1800	2.65
AmoebaNet-B (Real et al., 2018)	2.8	3150	2.55 ± 0.05
PNAS (Liu et al., 2017)†	3.2	225	3.41 ± 0.09
ENAS cell (Pham et al., 2018)	4.6	0.45	2.89
Lemonade II (Elsken et al., 2018a)	3.98	56	3.50
DARTS (Liu et al., 2019)	3.4	4	2.76 ± 0.09
SNAS (Xie et al., 2019)	2.8	1.5	2.85 ± 0.02
Luo et al. (2018)†	3.3	0.4	3.53
PARSEC (Casale et al., 2019)	3.7	1	2.81 ± 0.03
DARTS random (Liu et al., 2019)	3.1	–	3.29 ± 0.15
16 Random Models in Petridish space	2.27 ± 0.15	–	3.32 ± 0.15
Petridish cell w/o feature selection	2.50 ± 0.28	–	3.26 ± 0.10
Petridish cell	2.5	5	2.61 | 2.87 ± 0.13
Petridish cell more filters (F=37)	3.2	5	2.51 | 2.75 ± 0.21

Transfer to ImageNet

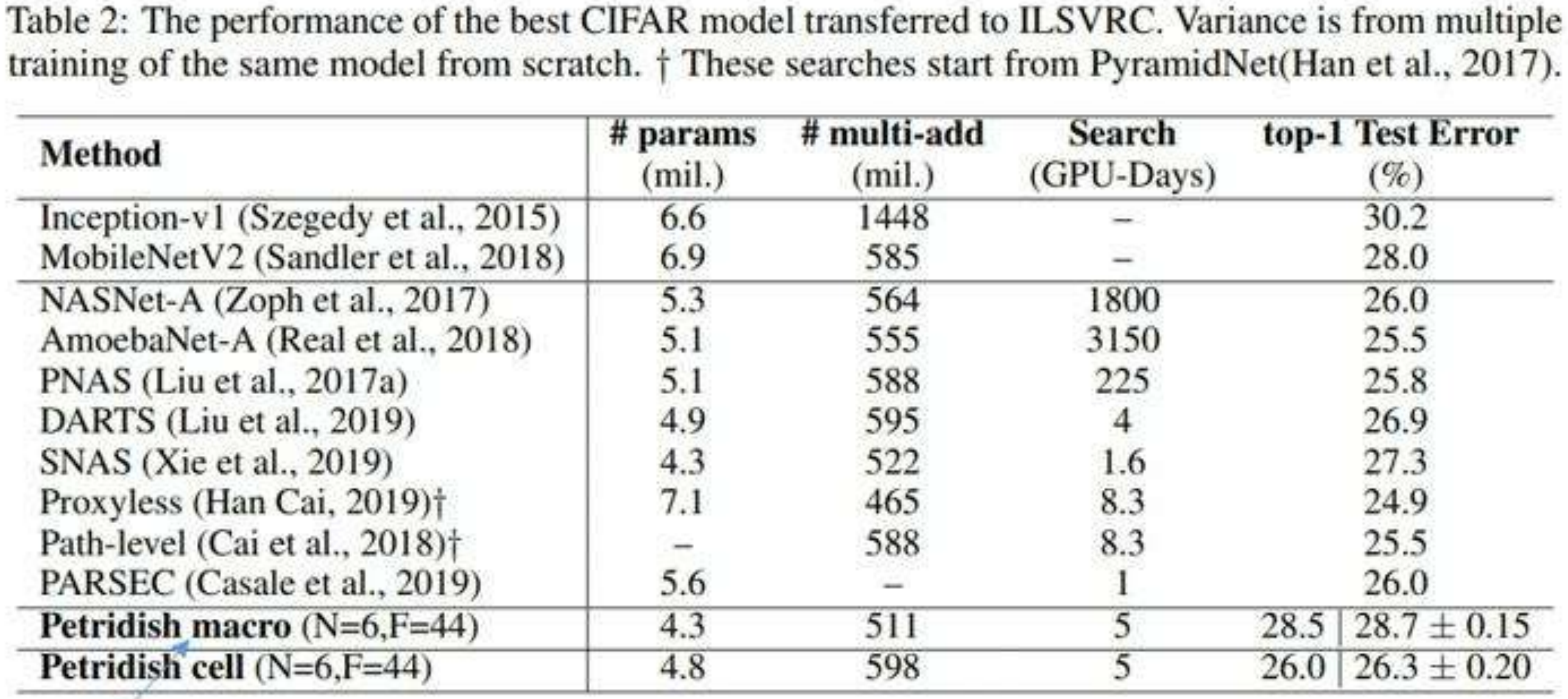


Table 2: The performance of the best CIFAR model transferred to ILSVRC. Variance is from multiple training of the same model from scratch. † These searches start from PyramidNet(Han et al., 2017).

Method	**# params** (mil.)	**# multi-add** (mil.)	**Search** (GPU-Days)	**top-1 Test Error** (%)
Inception-v1 (Szegedy et al., 2015)	6.6	1448	–	30.2
MobileNetV2 (Sandler et al., 2018)	6.9	585	–	28.0
NASNet-A (Zoph et al., 2017)	5.3	564	1800	26.0
AmoebaNet-A (Real et al., 2018)	5.1	555	3150	25.5
PNAS (Liu et al., 2017a)	5.1	588	225	25.8
DARTS (Liu et al., 2019)	4.9	595	4	26.9
SNAS (Xie et al., 2019)	4.3	522	1.6	27.3
Proxyless (Han Cai, 2019)†	7.1	465	8.3	24.9
Path-level (Cai et al., 2018)†	–	588	8.3	25.5
PARSEC (Casale et al., 2019)	5.6	–	1	26.0
Petridish macro (N=6,F=44)	4.3	511	5	28.5 | 28.7 ± 0.15
Petridish cell (N=6,F=44)	4.8	598	5	26.0 | 26.3 ± 0.20

No domain-knowledge injection in architecture design at all!

Language Modeling

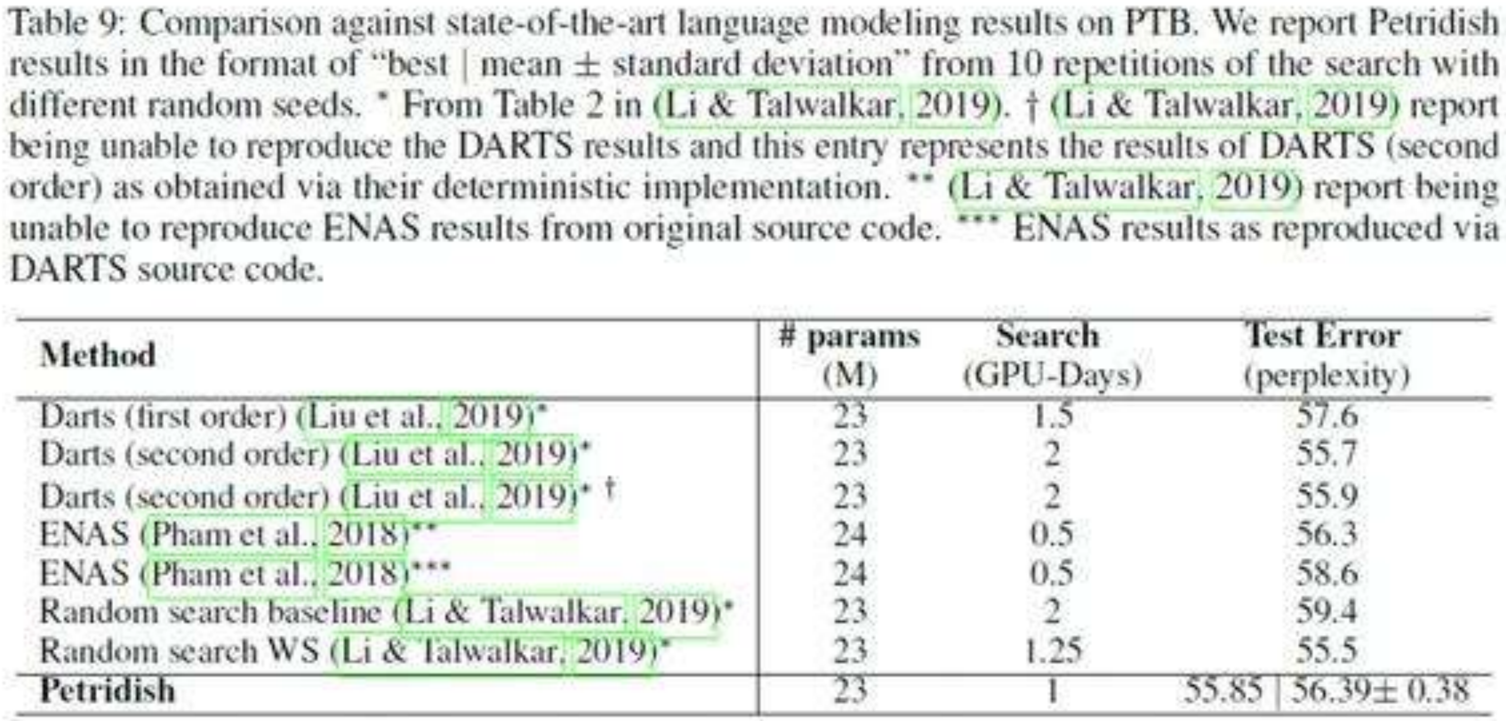


Table 9: Comparison against state-of-the-art language modeling results on PTB. We report Petridish results in the format of "best | mean ± standard deviation" from 10 repetitions of the search with different random seeds. * From Table 2 in (Li & Talwalkar, 2019). † (Li & Talwalkar, 2019) report being unable to reproduce the DARTS results and this entry represents the results of DARTS (second order) as obtained via their deterministic implementation. ** (Li & Talwalkar, 2019) report being unable to reproduce ENAS results from original source code. *** ENAS results as reproduced via DARTS source code.

Method	# params (M)	Search (GPU-Days)	Test Error (perplexity)
Darts (first order) (Liu et al., 2019)*	23	1.5	57.6
Darts (second order) (Liu et al., 2019)*	23	2	55.7
Darts (second order) (Liu et al., 2019)* †	23	2	55.9
ENAS (Pham et al., 2018)**	24	0.5	56.3
ENAS (Pham et al., 2018)***	24	0.5	58.6
Random search baseline (Li & Talwalkar, 2019)*	23	2	59.4
Random search WS (Li & Talwalkar, 2019)*	23	1.25	55.5
Petridish	23	1	55.85 | 56.39± 0.38

Note that since random search is essentially state-of-the-art search algorithm on PTB, we caution the community to not use PTB as a benchmark for comparing search algorithms for RNNs. The merits of any particular algorithm are difficult to compare at least on this particular dataset and task pairing. *More research along the lines of Ying et al. (2019) is needed on 1. whether the nature of the search space for RNNs specific to language modeling is particularly amenable to random search and or 2. whether it is the specific nature of RNNs by itself such that random search is competitive on any task which uses RNNs as the hypothesis space. We are presenting the results on PTB for the sake of completion since it has become one of the default benchmarks but ourselves don't derive any particular signal either way in spite of competitive performance.*

- Still lots of domain knowledge injection into the process.
- Tricks and tips needed for vision datasets are completely different from language or speech datasets (to be SOTA).
- Need better benchmarks and more rigorous reporting.
- All papers currently report on CIFAR10/100, ImageNet, PennTree Bank.
- Hyperparameters are set to magic constants.

No fully general solution yet but useful successes!

Moving Forward.

Patent filed Dec 7, 2018.

Compute and Platform for AzureML Service.

- Currently all development done on Philly.
- Migrating to Azure AML.

Datasets

- Can we get access to high business impact large scale 1P datasets?
- Larger vision and language datasets?