AutoML & NAS

# Automated machine learning Neural Architecture Search

**Current:**

Solution =

ML Expertise + Data + Computation

**But can we turn this into:**

Solution =

Data + 100X Computation

# Automated machine learning

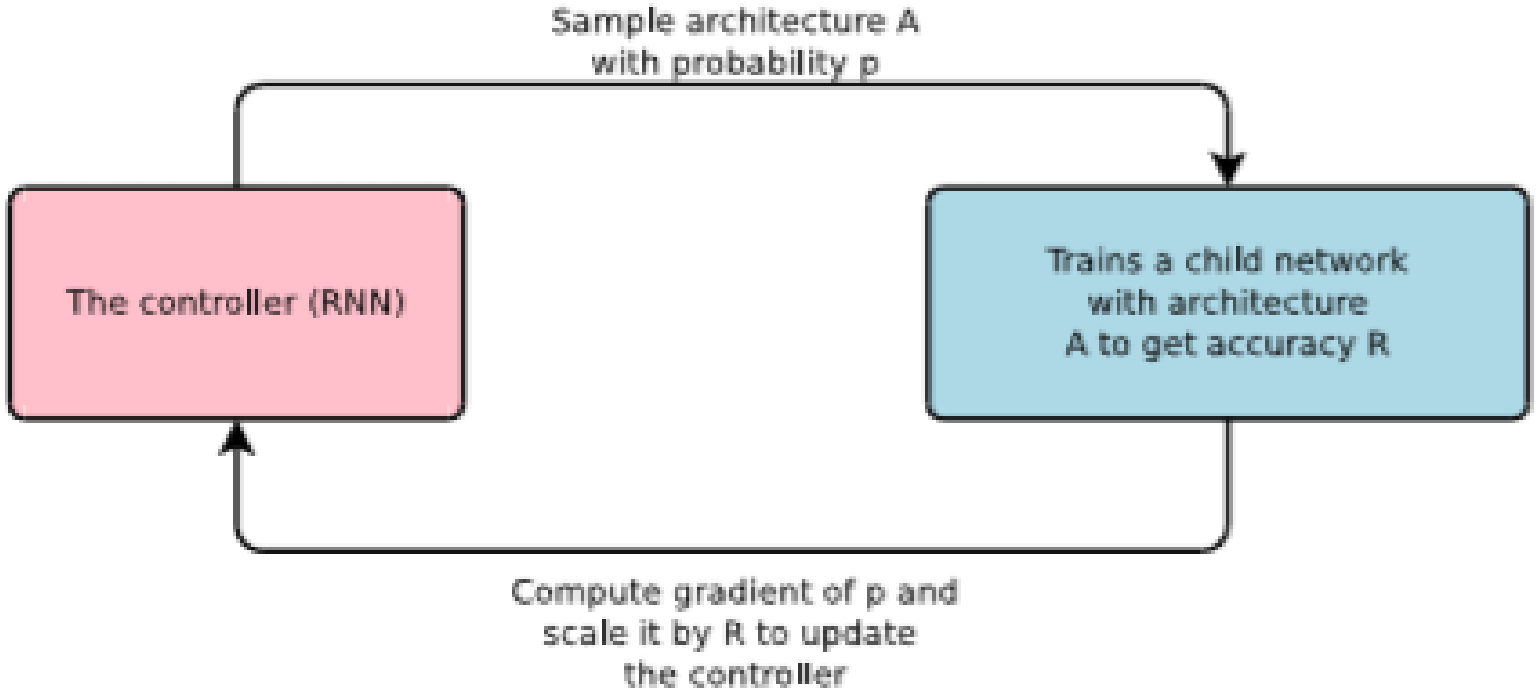
- Designing neural network architectures is hard
- Lots of human efforts go into tuning them
- Can we try and learn good architectures automatically?


# What is AutoML?

- **Traditionally**: automated methods for model selection and/or hyperparameter optimization
  - Method: random forests, gradient boosting machines, neural networks, and more
  - Filed:  open-source AutoML libraries, workshops, research, and competitions
  - Aims: test out different hyperparameters for a model; make ML pipeline easier; speed up
  - Libraries: AutoWEKA; auto-sklearn; H2O AutoML; TPOT
- **DeepLearning**: neural architecture search (NAS)
  - “it’s possible for neural nets to design neural nets”  ---------Google CEO Sundar Pichai
  - Category:
    - Reinforcement algorithms:  NASNet(ICLR-2017); ENAS(ICML-2018); PNAS(ECCV-2018)
    - Evolutionary learning: NASH(ICLR-2018) ; EAT-NAS(CVPR-2019)
    - Gradient-based approaches: DARTS(ICLR-2019)

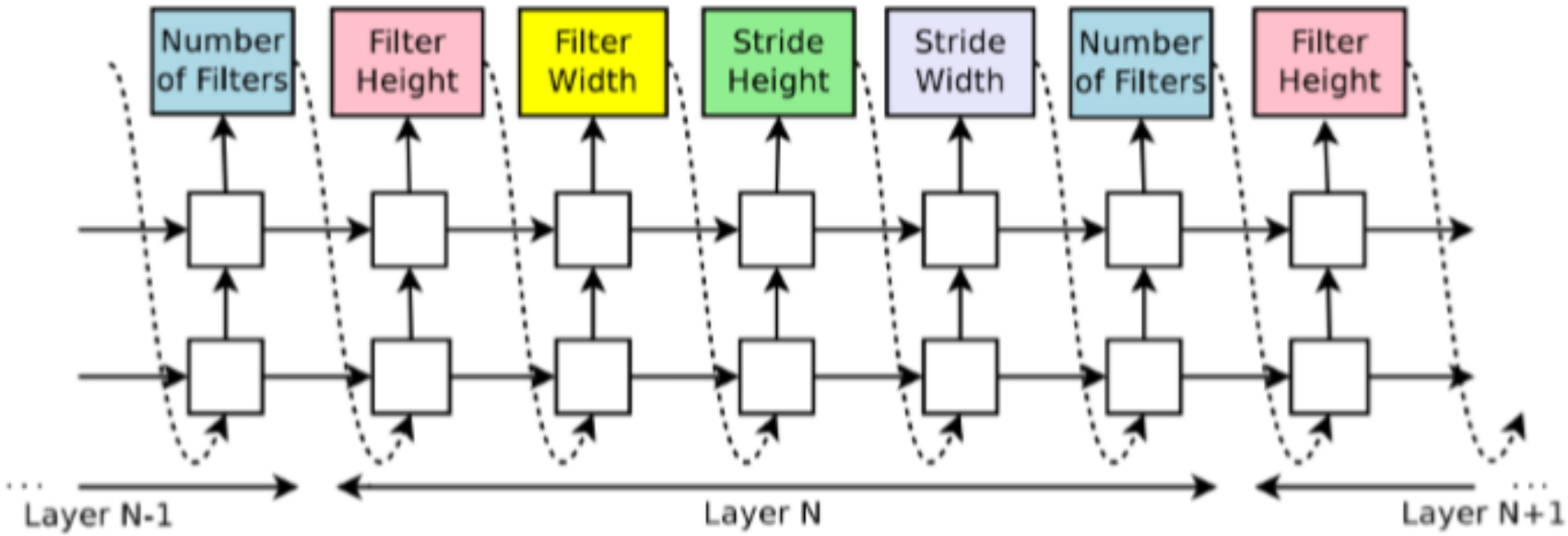
# Neural Architecture Search

- **search space**: which type of ANN can be designed and optimized in principle
- **search strategy**: which strategy is used to find optimal ANN's within the search space.
- **performance estimation strategy** : used obtain less costly estimates of a model's performance.



# Neural Architecture Search

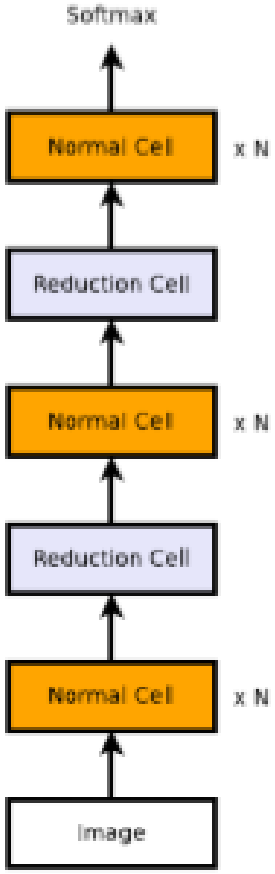
- How controller RNN samples a simple convolutional network



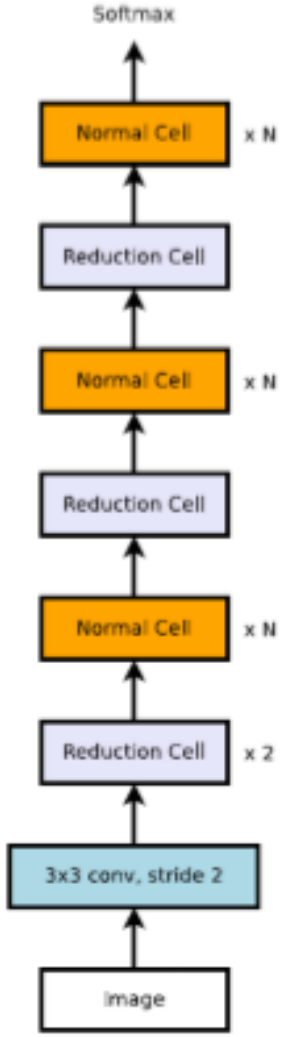
# Method

- Overall architectures of the convolutional nets are manually predetermined
  - Normal Cell – convolutional cells that return a feature map of the same dimension
  - Reduction Cell – convolutional cells that return a feature map where the feature map height and width is reduced by a factor of two
- Using common heuristic to double the number of filters in the output whenever the spatial activation size is reduced

# Scalable Architectures for Image Classification

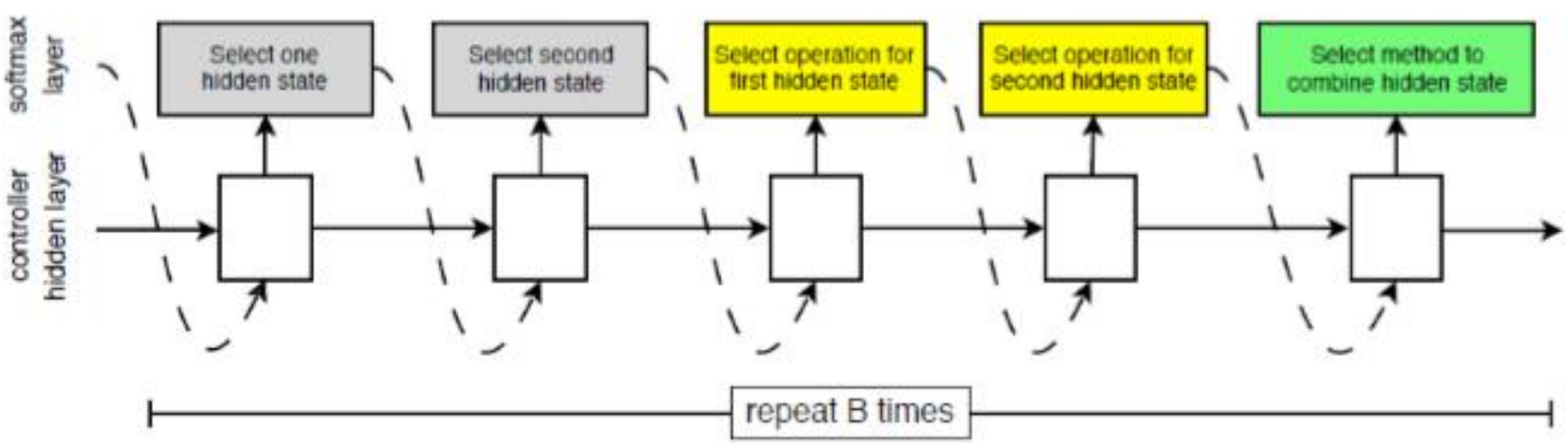


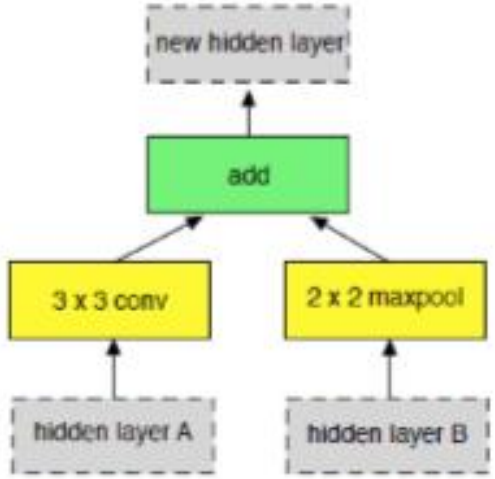

CIFAR10
Architecture




ImageNet
Architecture

# Models and Algorithms




**Step 1.** Select a hidden state from $h_i, h_{i-1}$ or from the set of hidden states created in previous blocks.

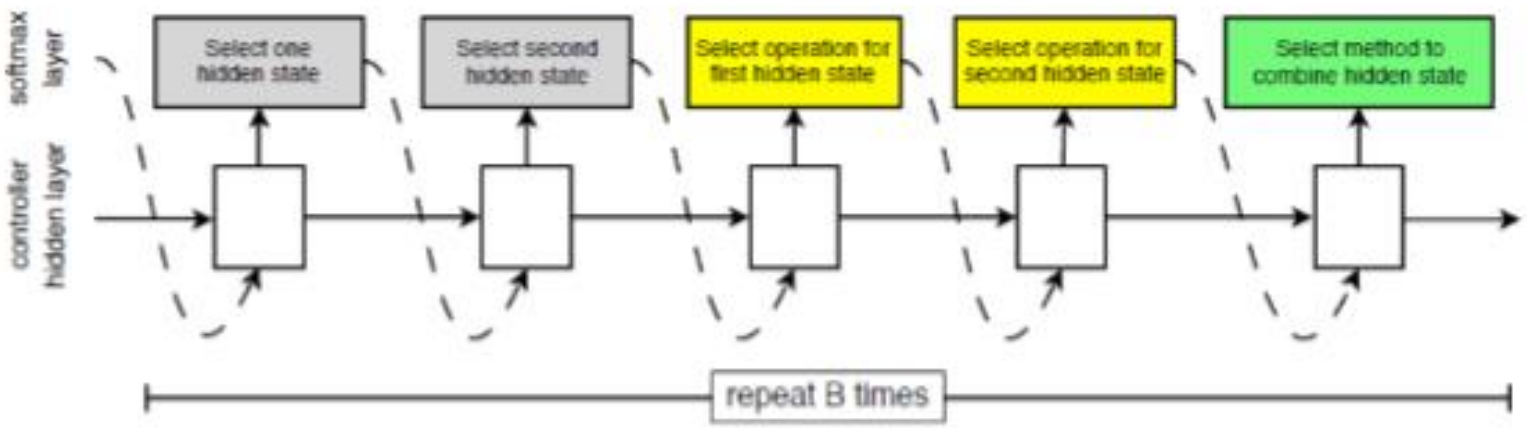
**Step 2.** Select a second hidden state from the same options as in Step 1.

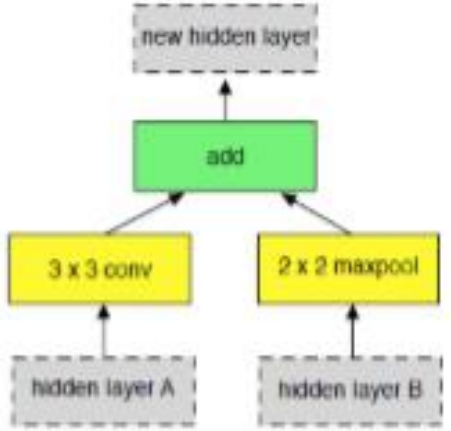
**Step 3.** Select an operation to apply to the hidden state selected in Step 1.

**Step 4.** Select an operation to apply to the hidden state selected in Step 2.

**Step 5.** Select a method to combine the outputs of Step 3 and 4 to create a new hidden state.

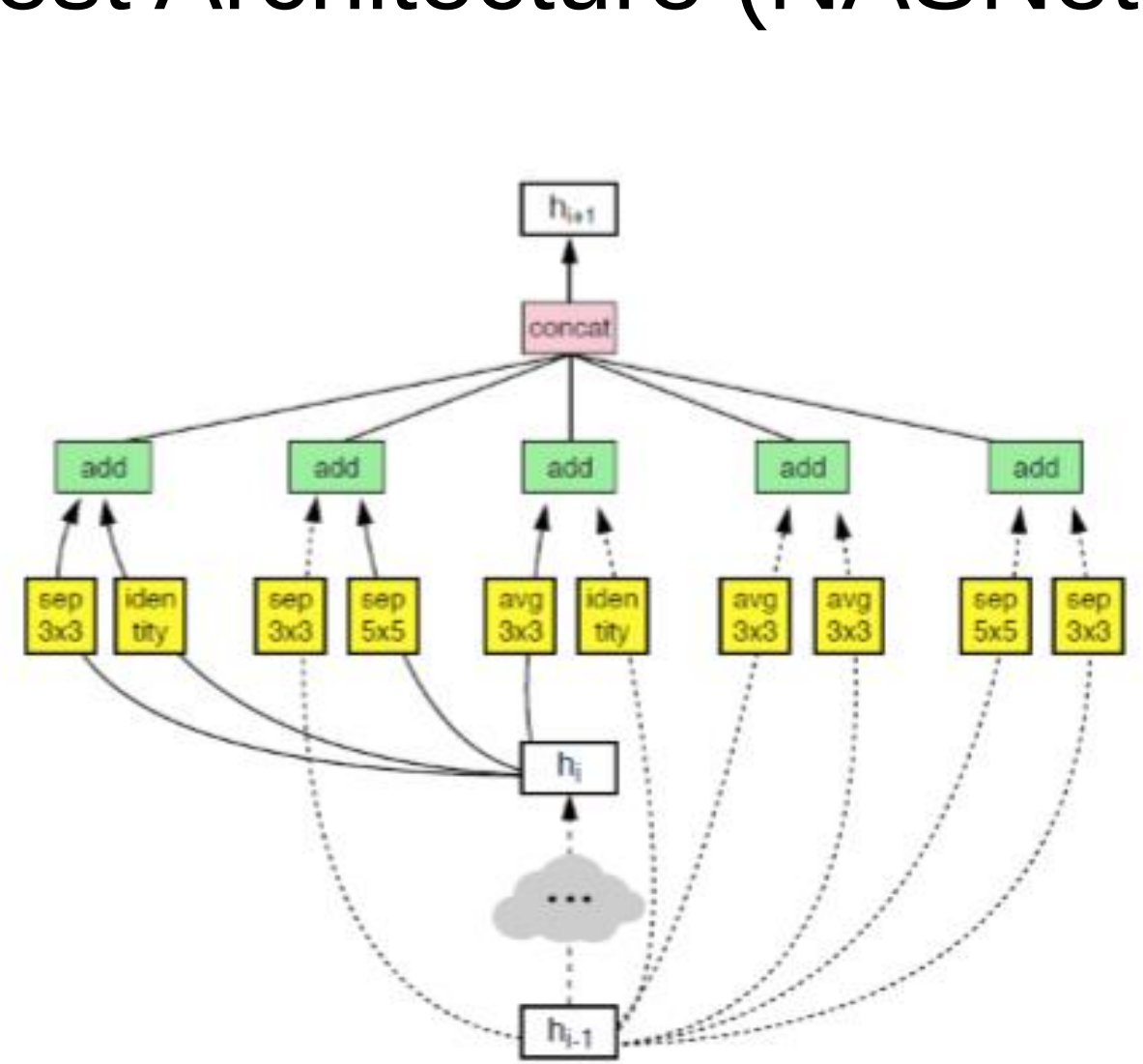
# Search Space in a Cell (Step 3 and 4)



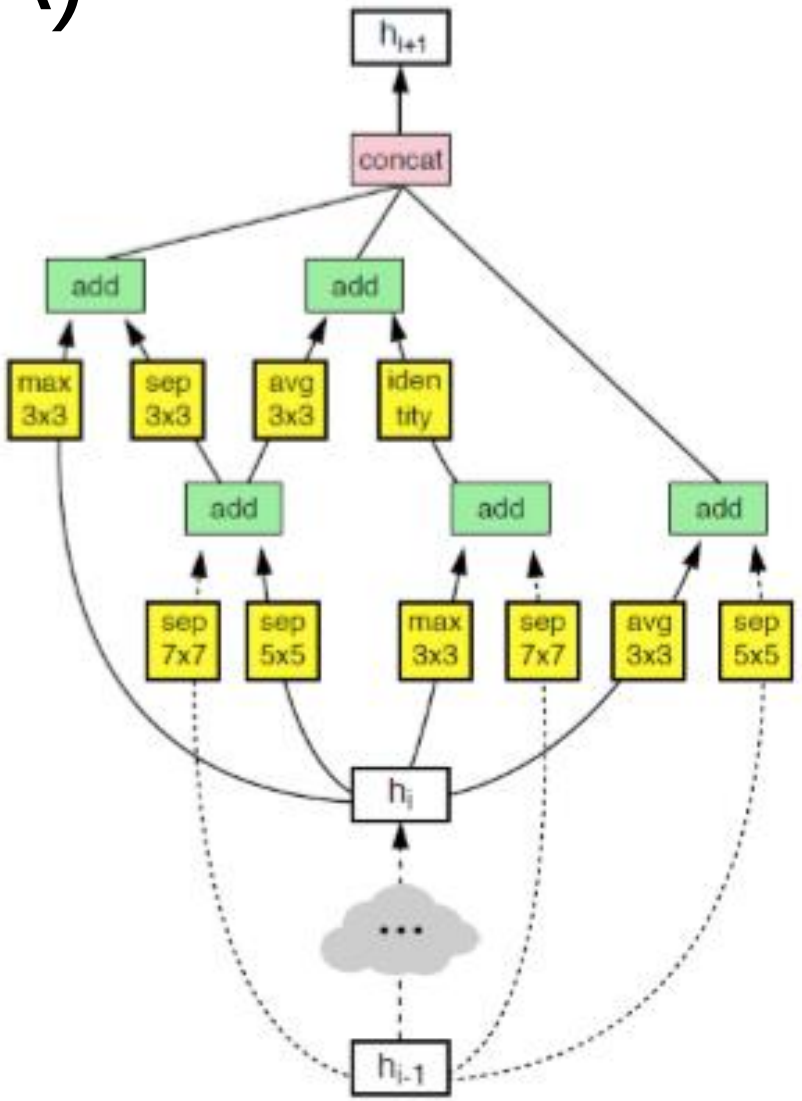




- identity
- 1x7 then 7x1 convolution
- 3x3 average pooling
- 5x5 max pooling
- 1x1 convolution
- 3x3 depthwise-separable conv
- 7x7 depthwise-separable conv
- 1x3 then 3x1 convolution
- 3x3 dilated convolution
- 3x3 max pooling
- 7x7 max pooling
- 3x3 convolution
- 5x5 depthwise-seperable conv

# Best Architecture (NASNet-A)

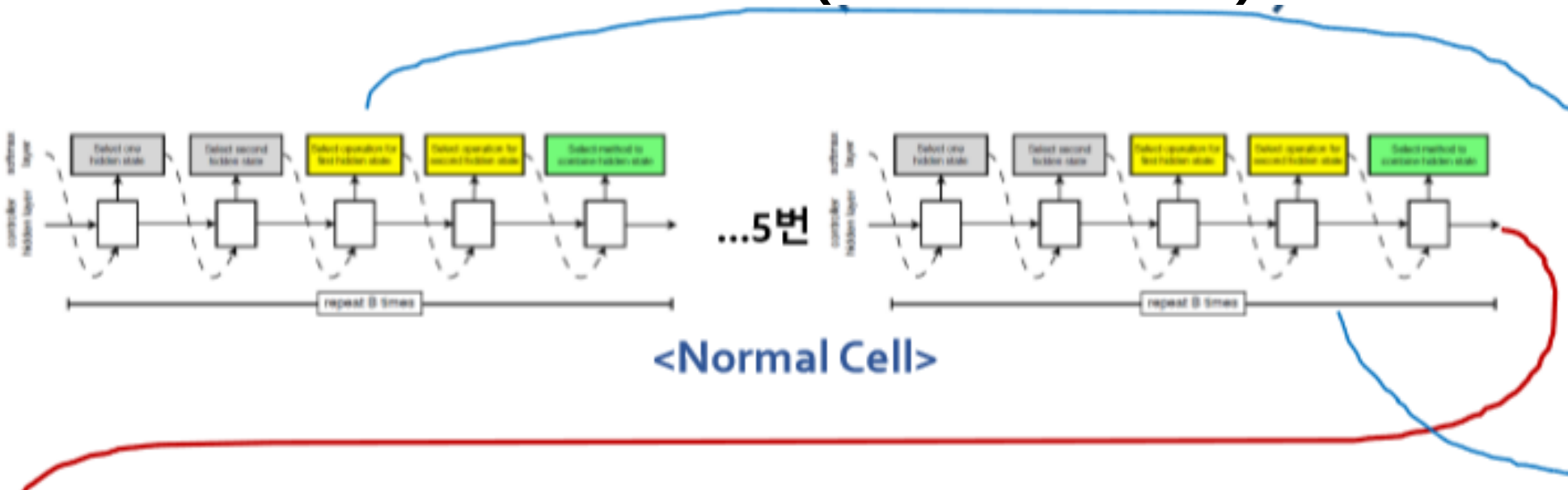



*Normal Cell*

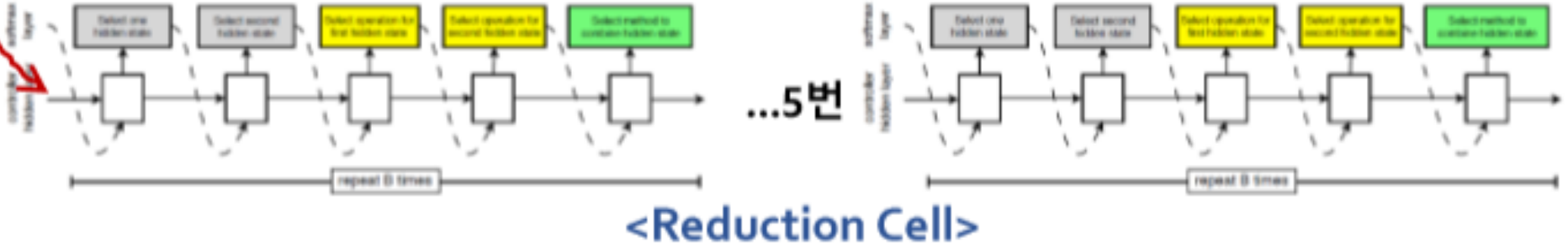



*Reduction Cell*

# Best Architecture (NASNet-A)

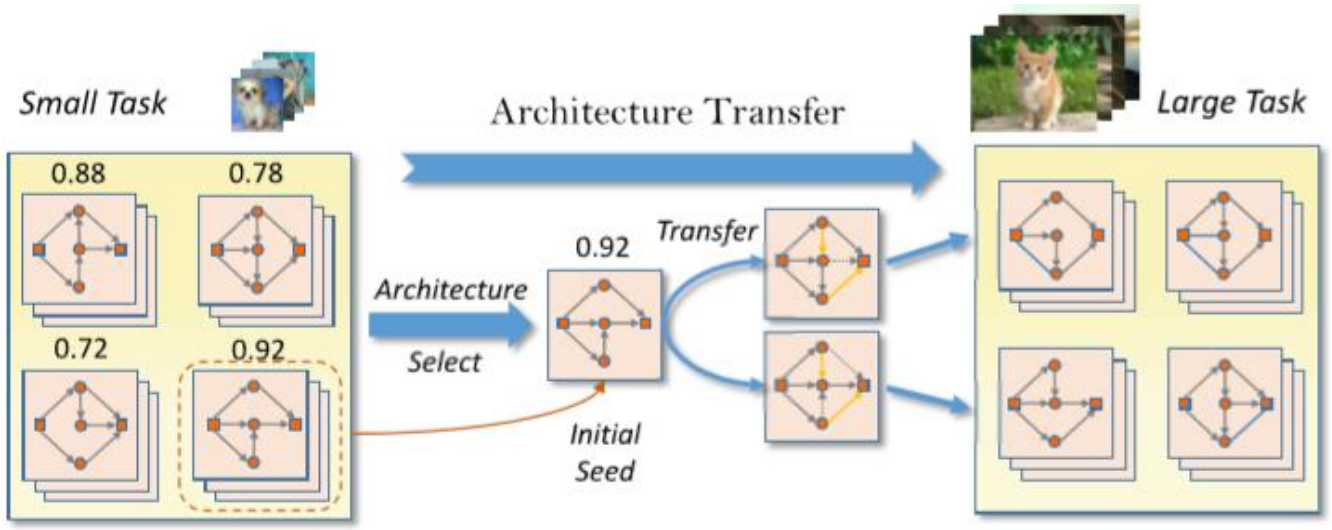







# EAT-NAS

- **Motivation:** computational resource
- **Contribution**: architectures are first searched on a small dataset, the best one is chosen as the basic architecture. Then the whole architecture is transferred with elasticity.




**Algorithm 1:** Evolutionary Algorithm

**input :** model population $\mathbb{P}$, population size $P$, population quality $Q$, sample size $S$, model $M$, dataset $D$

**output:** the best model $M_{best}$

```
P^(0) ← initialize(P)
while i < P do
    M_i.acc ← train-eval(M_i, D)
    M_i.score ← comp-score(M_i, M_i.acc)
end
Q^(0) ← comp-quality(P^(0))
while Q^(i) not converge do
    S^(i) ← sample(P^(i), S)
    M_best, M_worst ← pick(S^(i))
    M_mut ← mutate(M_best)
    M_mut.acc ← train-eval(M_mut, D)
    M_mut.score ← comp-score(M_mut, M_mut.acc)
    P^(i+1) ← remove(P^(i), M_worst)
    P^(i+1) ← add(P^(i+1), M_mut)
    Q^(i+1) ← comp-quality(P^(i+1))
end
M_best ← rerank-topk(P_best, k)
```

# References


- Barret Zoph, et al. “Neural Architecture Search with Reinforcement Learning”, In ICLR, 2017
- Pham, Hieu, et al. "Efficient neural architecture search via parameter sharing." *arXiv preprint arXiv:1802.03268* (2018).
- Liu, Chenxi, et al. "Progressive neural architecture search." *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018.
- Elsken, Thomas, Jan-Hendrik Metzen, and Frank Hutter. "Simple and efficient architecture search for convolutional neural networks." *arXiv preprint arXiv:1711.04528* (2017).
- Fang, Jiemin, et al. "EAT-NAS: Elastic Architecture Transfer for Accelerating Large-scale Neural Architecture Search." *arXiv preprint arXiv:1901.05884* (2019).
- Liu, Hanxiao, Karen Simonyan, and Yiming Yang. "Darts: Differentiable architecture search." *arXiv preprint arXiv:1806.09055* (2018).

# ENAS

## Efficient Neural Architecture Search via Parameter Sharing

# Authors


**Jeff Dean:**

- “Compilers don’t warn Jeff Dean. Jeff Dean warns compilers.”
- “When Jeff Dean designs software, he first codes the binary and then writes the source as documentation.”
- “I joined Google in mid-1999, and I'm currently a Google Senior Fellow in the Research Group, where I lead the Google Brain project.”

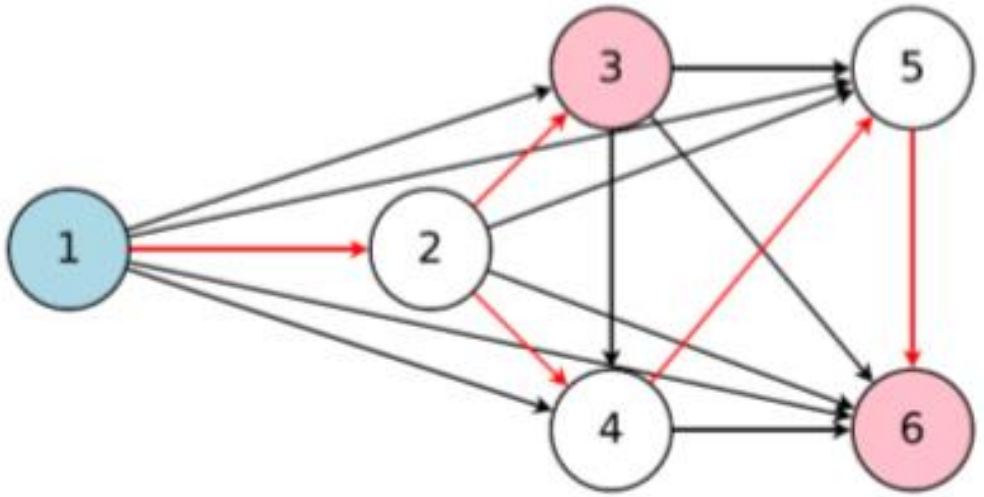
# Main idea

Forcing all child models to share weights

to eschew training each child model from scratch to convergence

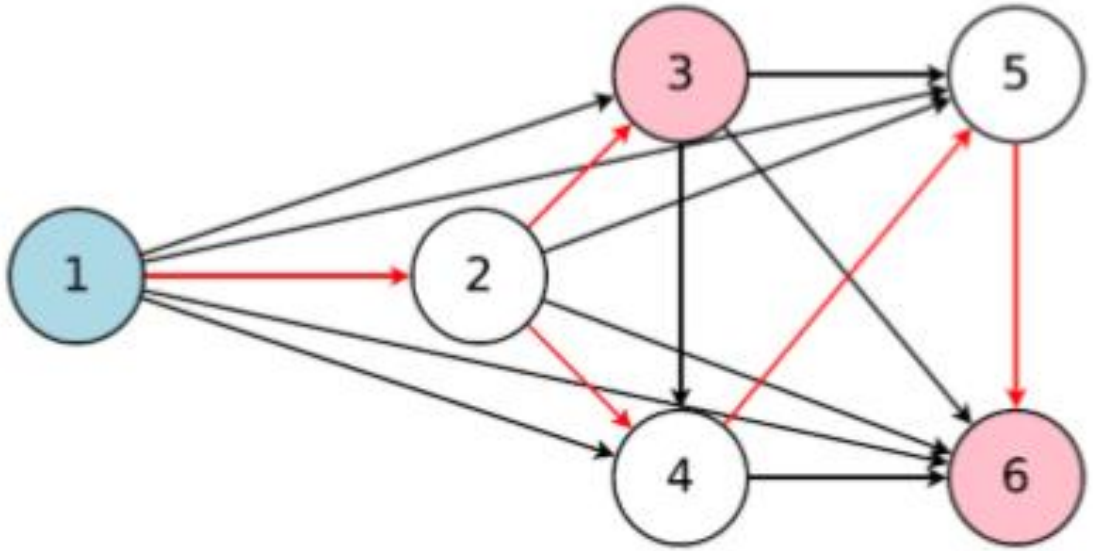
→ Using Single GTX 1080Ti GPU, the search for architectures takes less than 16 hours

# Directed Acyclic Graph(DAG)



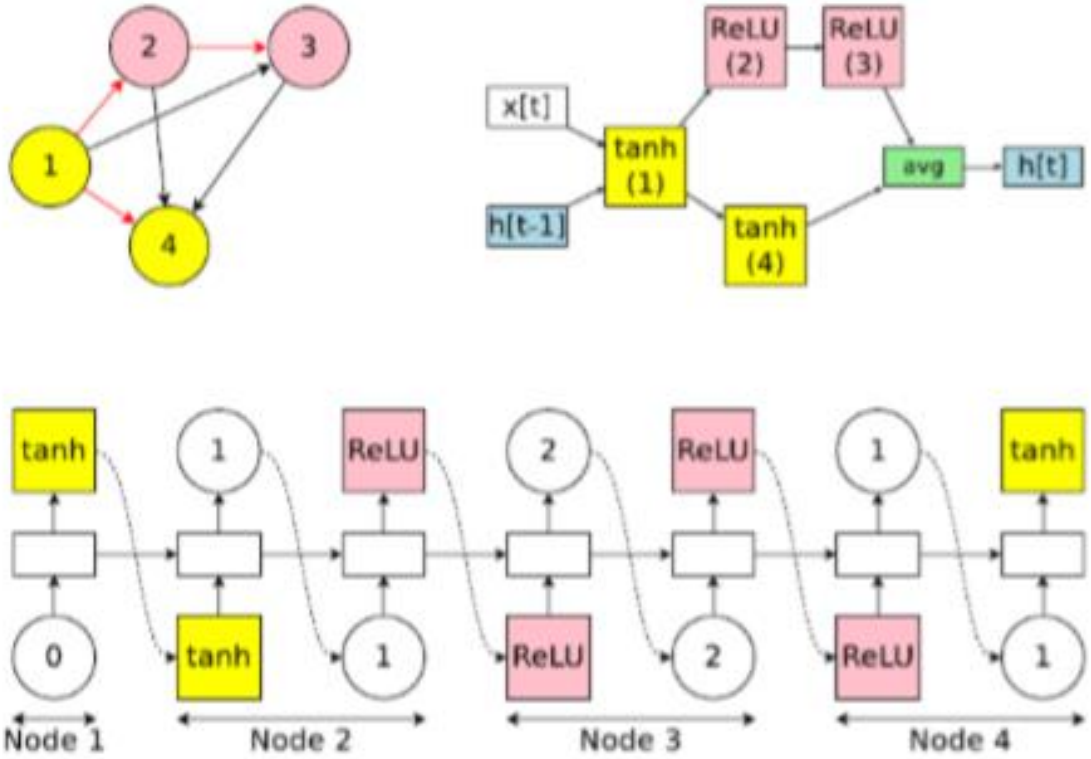

- ENAS's DAG is the superposition of all possible child models in a search space of NAS, where the nodes represent the local computations and the edges represent the flow of information.
- The local computations at each node have their own parameters, which are used only when the particular computation is activated.
- Therefore, ENAS's design allows parameters to be shared among all child models

# Directed Acyclic Graph(DAG)




- 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.

# Designing Recurrent Cells




1. At node 1: The controller first samples an activation function. In our example, the controller chooses the tanh activation function, which means that node 1 of the recurrent cell should compute $h_1 = \tanh(\mathbf{x}_t \cdot \mathbf{W}^{(x)} + \mathbf{h}_{t-1} \cdot \mathbf{W}_1^{(h)})$.

2. At node 2: The controller then samples a previous index and an activation function. In our example, it chooses the previous index 1 and the activation function ReLU. Thus, node 2 of the cell computes $h_2 = \text{ReLU}(h_1 \cdot \mathbf{W}_{2,1}^{(h)})$.

3. At node 3: The controller again samples a previous index and an activation function. In our example, it chooses the previous index 2 and the activation function ReLU. Therefore, $h_3 = \text{ReLU}(h_2 \cdot \mathbf{W}_{3,2}^{(h)})$.

4. At node 4: The controller again samples a previous index and an activation function. In our example, it chooses the previous index 1 and the activation function tanh, leading to $h_4 = \tanh(h_1 \cdot \mathbf{W}_{4,1}^{(h)})$.

5. For the output, we simply average all the loose ends, *i.e.* the nodes that are not selected as inputs to any other nodes. In our example, since the indices 3 and 4 were never sampled to be the input for any node, the recurrent cell uses their average $(h_3 + h_4)/2$ as its output. In other words, $\mathbf{h}_t = (h_3 + h_4)/2$.

# Search Space for Recurrent Cells

- 4 activation functions are allowed
  - tanh, ReLU, identity, sigmoid
- If the recurrent cell has N nodes,
  - The search space has $4^N \times N!$ configuration
  - When N = 12, there are approximately $10^{15}$ models in the search space

# Training ENAS

- The controller network is an LSTM with 100 hidden units
- This LSTM samples decisions via softmax classifier, in an autoregressive fashion
- In ENAS, there are two sets of learnable parameters
  - The parameters of the controller LSTM, denoted by θ
  - The shared parameters of child models, denoted by ω
- The first phase trains ω,

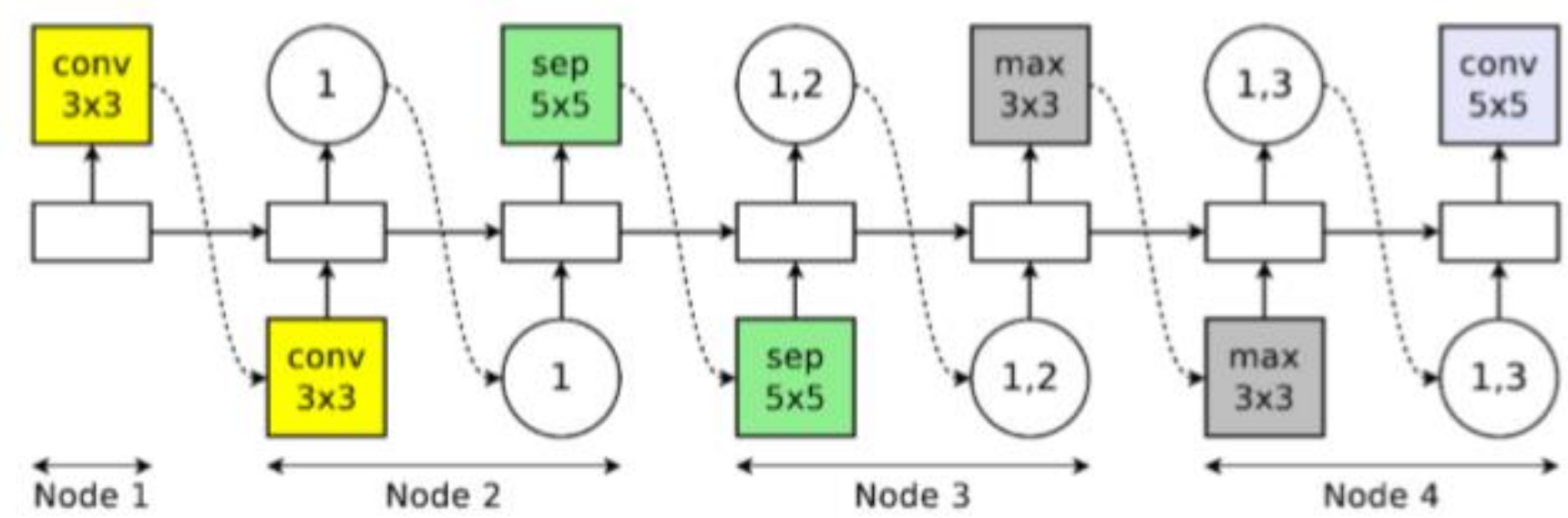
$$\nabla_\omega \mathbb{E}_{\mathbf{m}\sim\pi(\mathbf{m};\theta)}\left[\mathcal{L}(\mathbf{m};\omega)\right] \approx \frac{1}{M}\sum_{i=1}^{M}\nabla_\omega \mathcal{L}(\mathbf{m}_i,\omega),$$

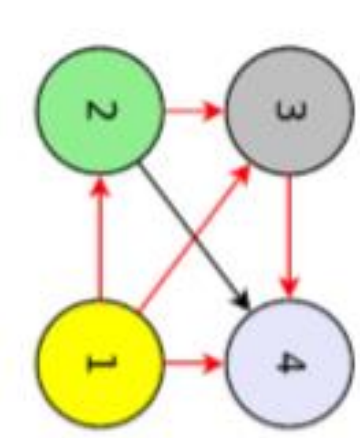
- The second phase trains θ

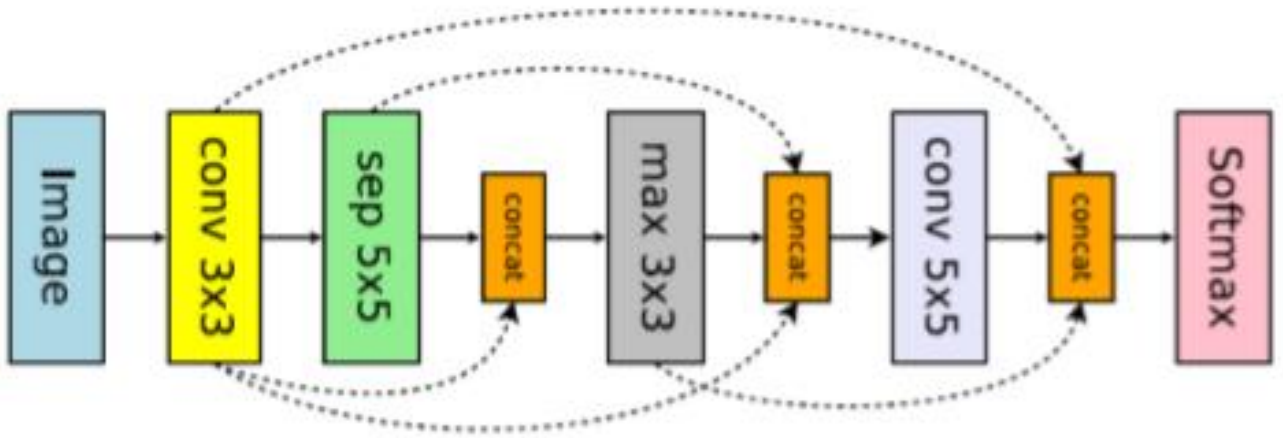
# Deriving Architectures

- Sampling several models from the trained policy $\pi(m, \theta)$
- Computing its reward on a single minibatch sampled from the validation set
- The model with the highest reward is taken and retrained from scratch
- training all the sampled models from scratch and selecting the model with the highest performance is possible but it is not economical

# Designing CNN



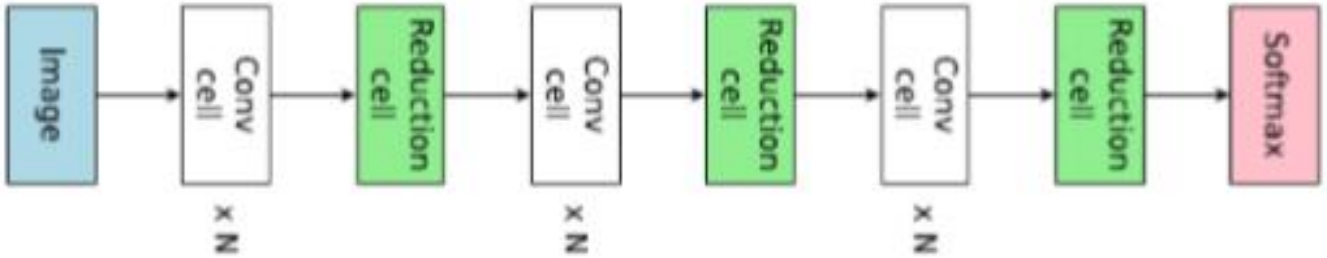



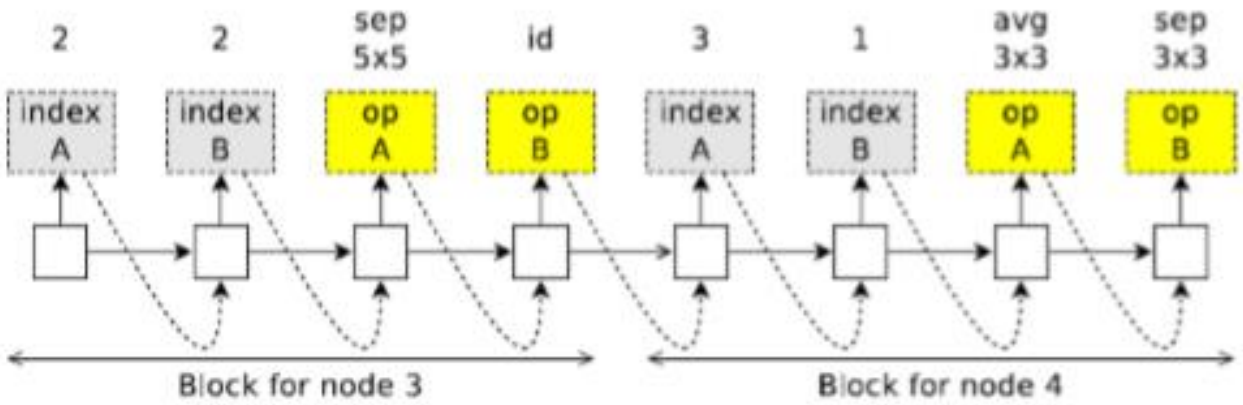



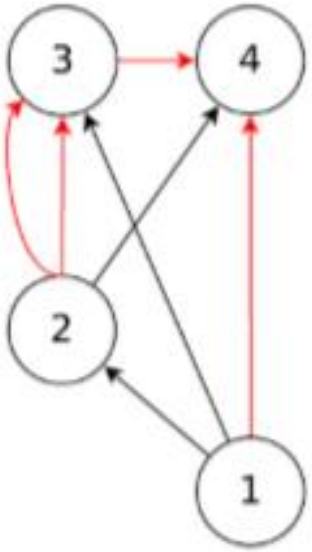
# Search Spaces for CNN

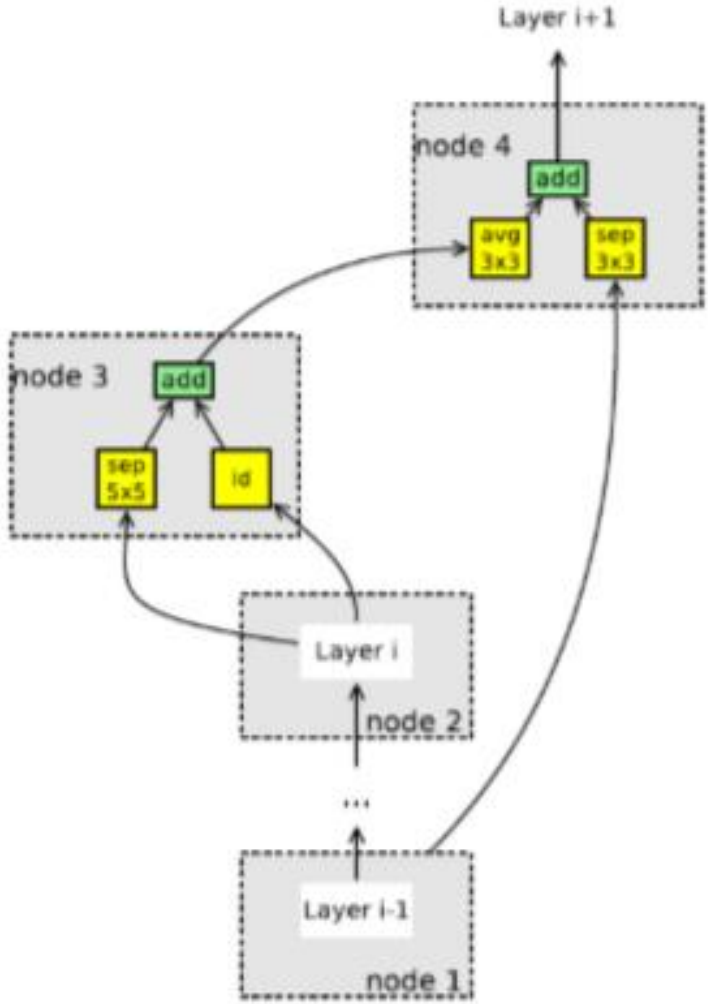
- The 6 operations available for controller are
  - Convolution with filter sizes 3x3 and 5x5
  - Depthwise-separable convolutions with filter sizes 3x3 and 5x5
  - Max pooling and average pooling of kernel size 3x3
- Making the described set of decisions for a total of $L$ times,we can sample a network of $L$ layers.
- Since all decisions are independent, there are $6^L \times 2^{L(L-1)/2}$
- When L=12, resulting in $1.6 \times 10^{29}$ possible networks

# Designing Convolutional Cells



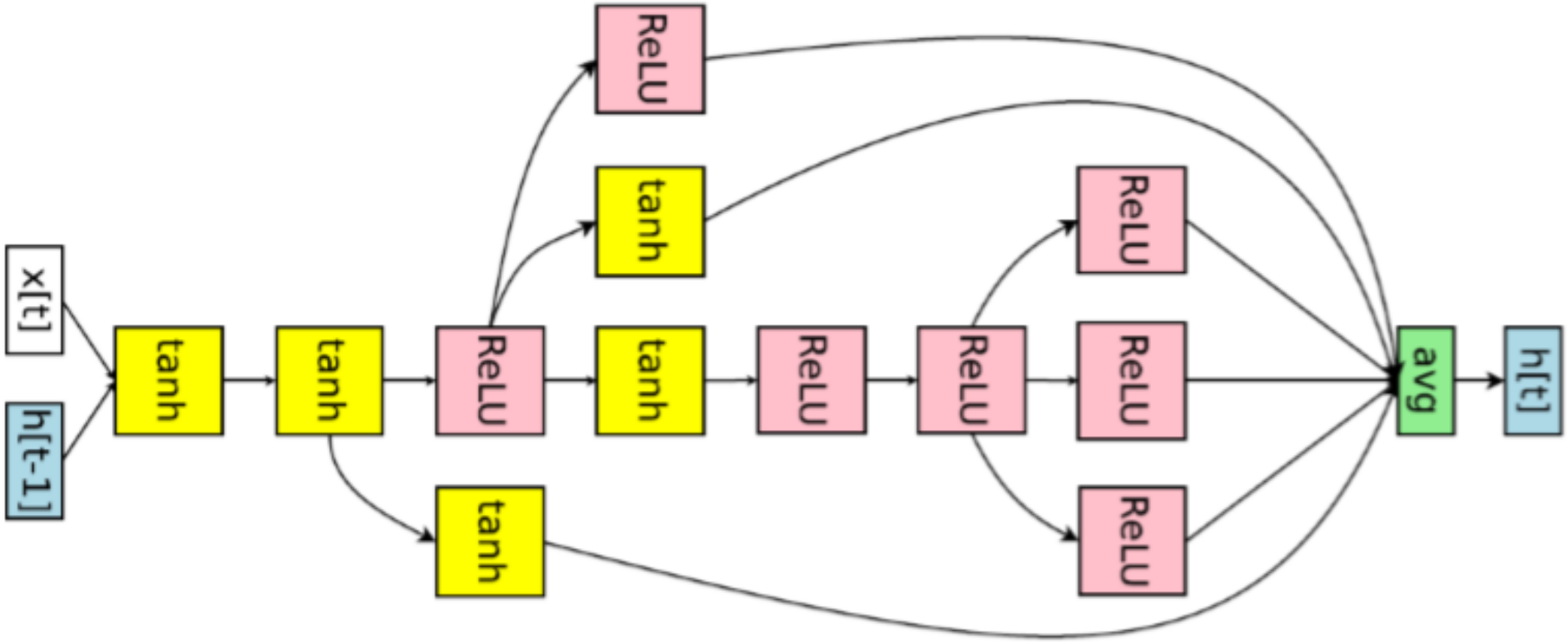



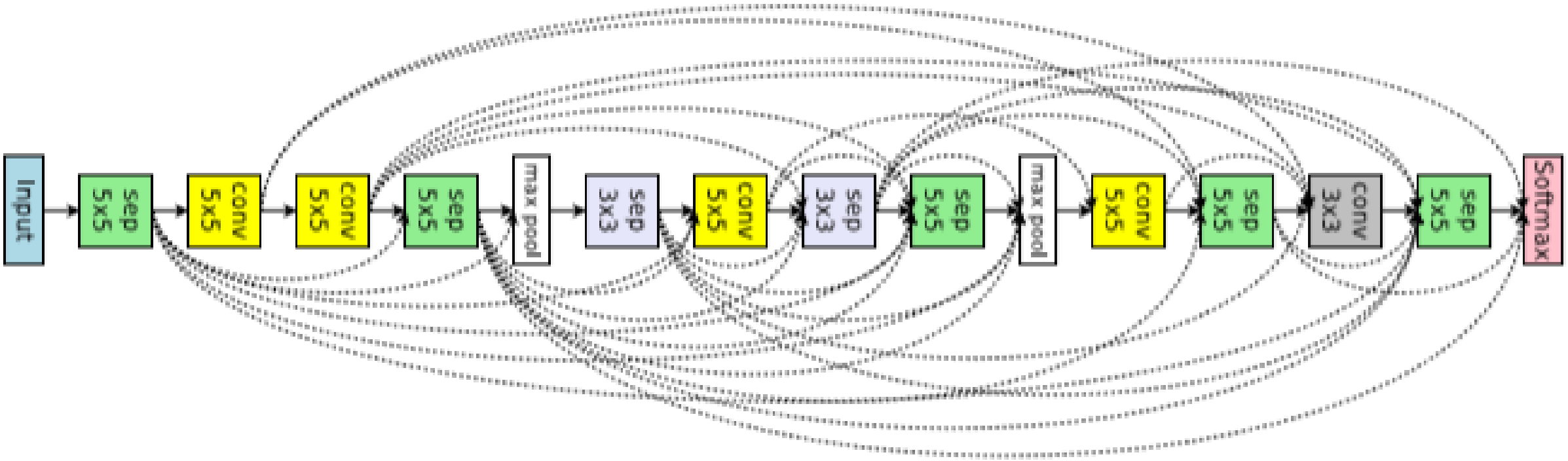
# Search Spaces for Convolutional Cells

- The 5 available operations are
  - Identity
  - Separable convolution with kernel size 3x3 and 5x5
  - Average pooling and max pooling with kernel size 3x3
- If there are B nodes, $(5 \times (B-2)!)^4$ cells are possible
- With B = 7, the search space can realize $1.3 \times 10^{11}$ final networks, making it significantly smaller than the search space for entire convolutional networks

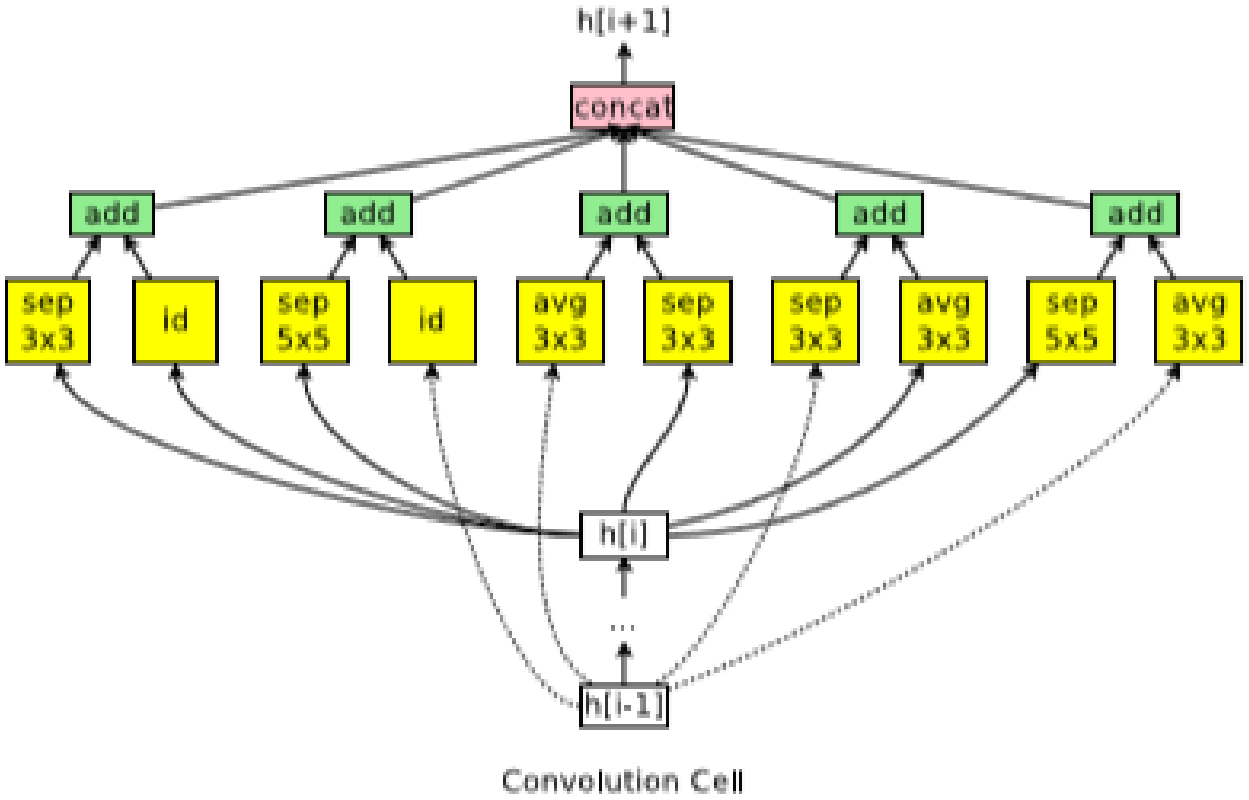
# Network Architecture for Penn Treebank



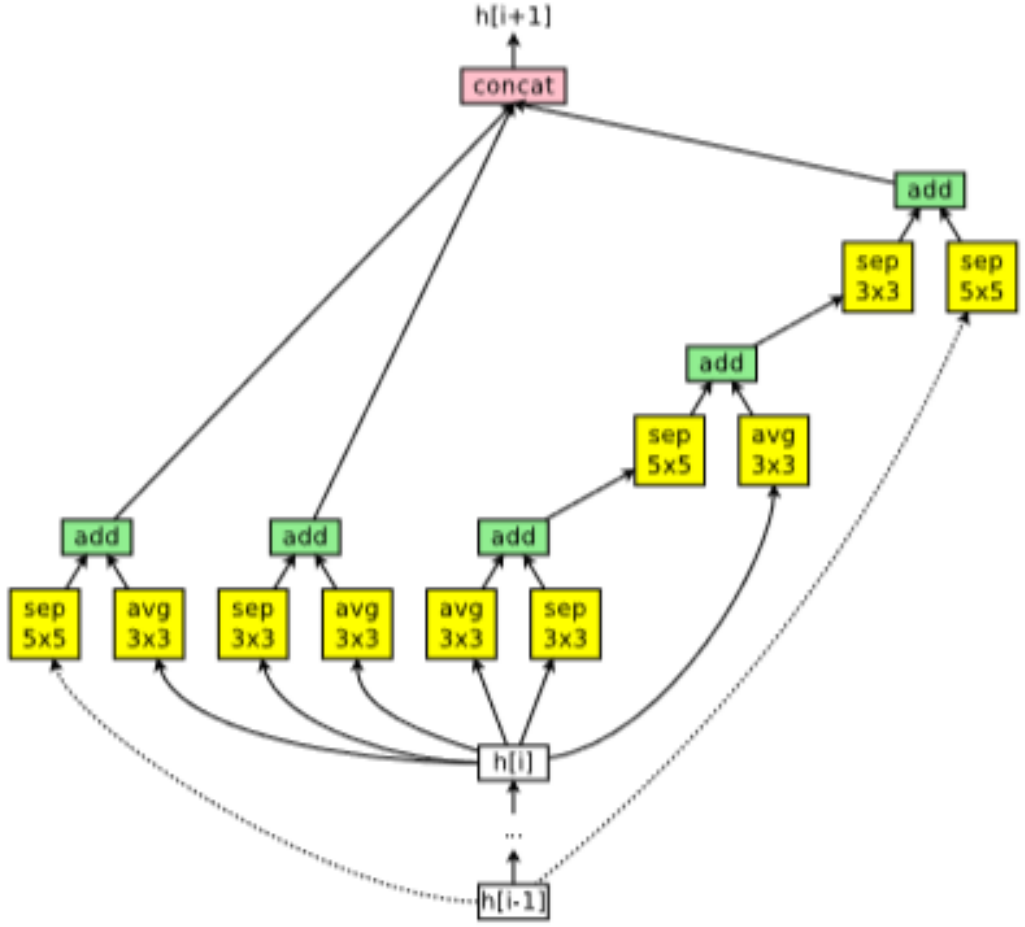
# Network Architecture for CIFAR-10(from macro search space)



# Network Architecture for CIFAR-10(from micro search space)




Convolution Cell




Reduction Cell

# Experimental Results

- Test perplexity on Penn Treebank of ENAS and other baselines

| Architecture | Additional Techniques | Params (million) | Test PPL |
|---|---|---|---|
| LSTM (Zaremba et al., 2014) | Vanilla Dropout | 66 | 78.4 |
| LSTM (Gal & Ghahramani, 2016) | VD | 66 | 75.2 |
| LSTM (Inan et al., 2017) | VD, WT | 51 | 68.5 |
| RHN (Zilly et al., 2017) | VD, WT | 24 | 66.0 |
| LSTM (Melis et al., 2017) | Hyper-parameters Search | 24 | 59.5 |
| LSTM (Yang et al., 2018) | VD, WT, $\ell_2$, AWD, MoC | 22 | 57.6 |
| LSTM (Merity et al., 2017) | VD, WT, $\ell_2$, AWD | 24 | 57.3 |
| LSTM (Yang et al., 2018) | VD, WT, $\ell_2$, AWD, MoS | **22** | **56.0** |
| NAS (Zoph & Le, 2017) | VD, WT | 54 | 62.4 |
| ENAS | VD, WT, $\ell_2$ | **24** | **56.3** |

# Experimental Results

- Classification errors of ENAS and baselines on CIFAR-10

| 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** |

# Conclusion

- NAS is an important advance that automatizes the designing process of neural networks. However, NAS's computational expense prevents it from being widely adopted
- ENAS, a novel method that speeds up NAS by more than 1000x
- ENAS's key contribution is the sharing of parameters across child models during the search for architectures.
    - implemented by searching for a subgraph within a larger graph that incorporates architectures in a search space.
- ENAS works well on both CIFAR-10 and Penn Treebank datasets.

# DARTS

## DARTS: DIFFERENTIABLE ARCHITECTURE SEARCH

# Authors

## DARTS: Differentiable Architecture Search

**Hanxiao Liu**
CMU
hanxiaol@cs.cmu.edu

**Karen Simonyan**
DeepMind
simonyan@google.com

**Yiming Yang**
CMU
yiming@cs.cmu.edu

- **Hanxiao Liu:** Tsinghua(2013)→CMU(2018)→Google brain
- Interests: meta-learning; language understanding; automatic design of neural architectures
- Papers:
  - Hierarchical Representations for Efficient Architecture Search (ICLR 2018)
  - Analogical Inference for Multi-Relational Embedding (ICML 2017)
  - Adaptive Smoothed Online Multi-Task Learning (NIPS 2016)
  - Cross-Graph Learning of Multi-Relational Associations (ICML 2016)

# Authors

## Yiming Yang

Professor of Language Technologies Institute and Machine Learning Department

School of Computer Science of Carnegie Mellon University

Curriculum Vitae

**Email:** *yiming AT cs DOT cmu DOT edu*
**Secretary:** Mary Jo Bensasi [maryjob AT cs DOT cmu DOT edu]

**Post:**
Yiming Yang
GHC 6717, LTI
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213-8213

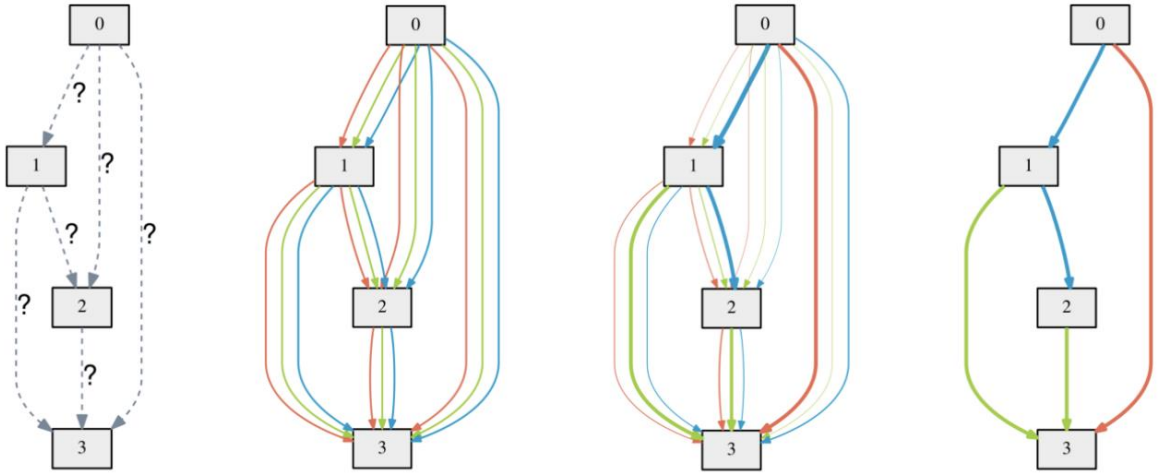
### Recent Research

Detailed descriptions are available here.

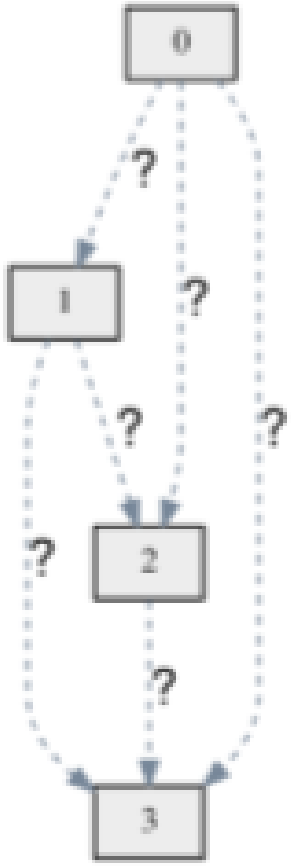
- Graph-based Transdutive Learning from Heterogeneous Data Sources
- Scalable Machine Learning for Time Series Analysis
- Extremely Large-scale Classification and Cross-language Transfer Learning for Classification
- Macro-Level Information Fusion for Events and Entities

# Main idea

- Problems: evolution or reinforcement methods use discrete and non-differentiable search space, always have randomness.
- DARTS: based on the continuous relaxation of the architecture representation, allowing efficient search of the architecture using gradient descent.
- Transfer: the architectures learned by DARTS on CIFAR-10 and PTB are transferable to ImageNet and WikiText-2
- Effective: being orders of magnitude faster than state-of-the-art non-differentiable techniques

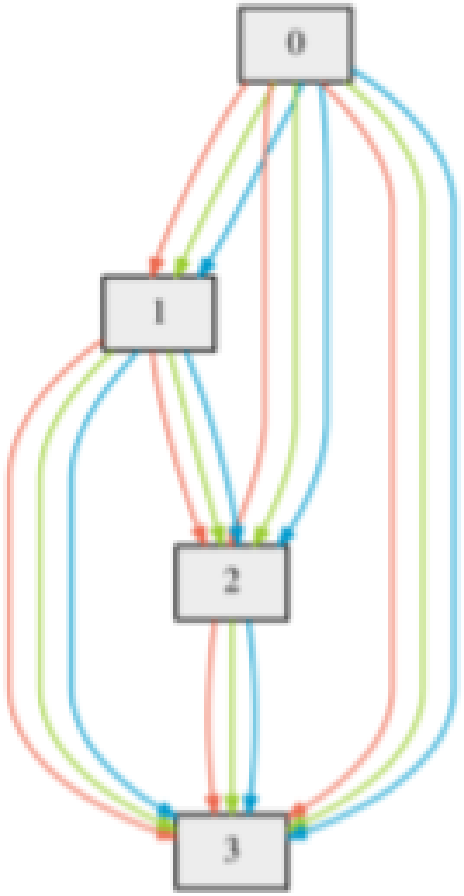


# Search space




- Building block: computation cell (DAG)
  - Each node x(i) is a latent representation (e.g. a feature map in convolutional networks)
  - each directed edge (i,j) is associated with some operation o(i,j) that transforms x(i).
  - the cell to have two input nodes and a single output node
    - Convolutional cells: previous two layers
    - Recurrent cells: the input at the current step and the state carried from the previous step
    - Output: applying a reduction operation (e.g. concatenation) to all the intermediate nodes
- The task of learning the cell therefore reduces to learning the operations on its edges.

# Continuous relaxation

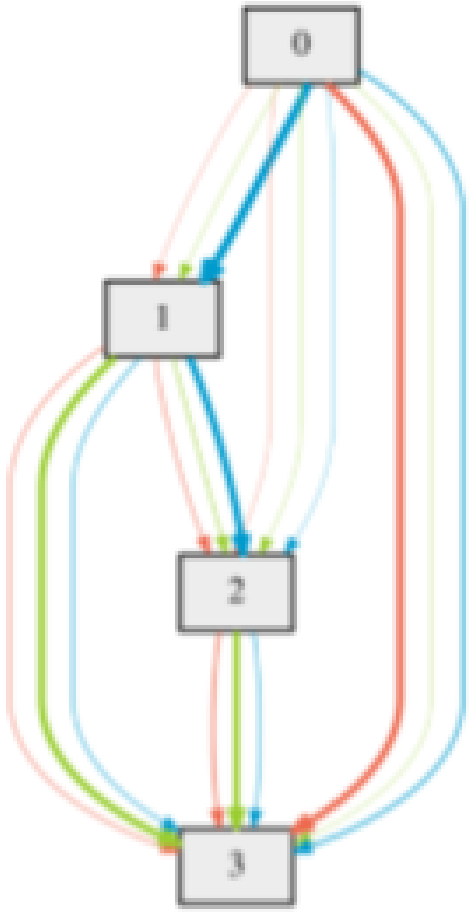



- To make the search space continuous, we relax the categorical choice of a particular operation to a softmax over all possible operations

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

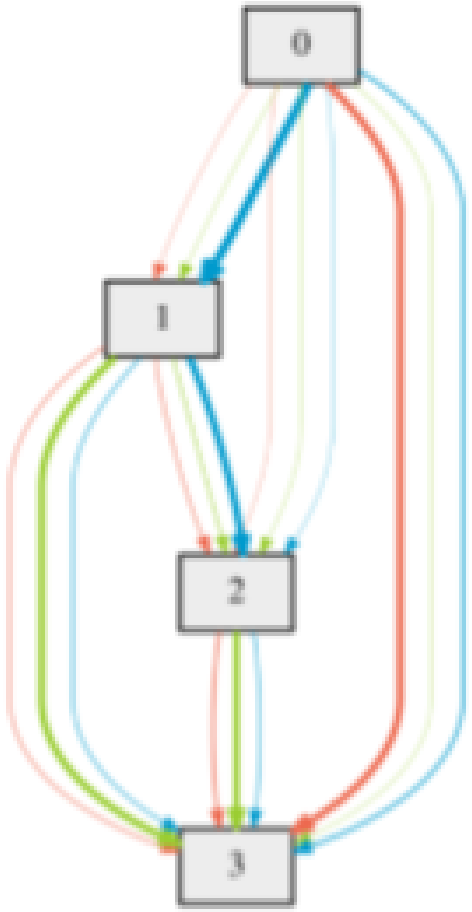
- the operation mixing weights for a pair of nodes (i,j) are parameterized by a vector α(i,j) of dimension |O|.
- task of architecture search: reduces to learning a set of continuous variables $\alpha = \{\alpha^{(i,j)}\}$
- Final discrete architecture can be obtained by replacing each mixed operation o(i,j) with the most likely operation:
  - i.e., $o^{(i,j)} = \text{argmax}_{o \in \mathcal{O}} \ \alpha_o^{(i,j)}$

# Optimize




- Goal: to jointly learn the architecture ***α*** and the weights ***w***
  - e.g. weights of the convolution filters
  - optimize the validation loss, but using gradient descent.
- Bilevel optimization problem: to find ***α∗*** that minimizes the validation loss ***Lval(w∗,α∗),*** where the weights ***w∗*** associated with the architecture are obtained by minimizing the training loss ***Ltrain(w∗,α∗)***
  - $$\min_{\alpha} \quad \mathcal{L}_{val}(w^*(\alpha), \alpha)$$
    $$\text{s.t.} \quad w^*(\alpha) = \text{argmin}_w \ \mathcal{L}_{train}(w, \alpha)$$

# Approximate architecture gradient




- Evaluating the architecture gradient exactly can be prohibitive due to the expensive inner optimization
- simple approximation scheme
  - $$\nabla_\alpha \mathcal{L}_{val}(w^*(\alpha), \alpha) \approx \nabla_\alpha \mathcal{L}_{val}(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha)$$
  - approximate w∗(α) by adapting w using only a single training step, without solving the inner optimization completely by training until convergence; $w^*(\alpha) = \text{argmin}_w \ \mathcal{L}_{train}(w, \alpha)$
  - Reduce $\nabla_\alpha \mathcal{L}_{val}(w, \alpha)$ if w is already a local optimum for the inner optimization
  - Eg: meta-learning; gradient based hyperparameter tuning; unrolled generative adversarial networks

# Approximate architecture gradient

**Algorithm 1:** DARTS – Differentiable Architecture Search

Create a mixed operation $\bar{o}^{(i,j)}$ parametrized by $\alpha^{(i,j)}$ for each edge $(i,j)$

**while** *not converged* **do**

1. Update architecture $\alpha$ by descending $\nabla_\alpha \mathcal{L}_{val}(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha)$ ($\xi = 0$ if using first-order approximation)
2. Update weights $w$ by descending $\nabla_w \mathcal{L}_{train}(w, \alpha)$

Derive the final architecture based on the learned $\alpha$.


- “While we are not currently aware of the convergence guarantees for our optimization algorithm, in practice it is able to reach a fixed point with a suitable choice of learning rate“
- Applying chain rule to the approximate architecture gradient yields
  - $\nabla_\alpha \mathcal{L}_{val}(w', \alpha) - \xi \nabla^2_{\alpha,w} \mathcal{L}_{train}(w, \alpha) \nabla_{w'} \mathcal{L}_{val}(w', \alpha)$
  - using the finite difference approximation
    - $\nabla^2_{\alpha,w} \mathcal{L}_{train}(w, \alpha) \nabla_{w'} \mathcal{L}_{val}(w', \alpha) \approx \frac{\nabla_\alpha \mathcal{L}_{train}(w^+, \alpha) - \nabla_\alpha \mathcal{L}_{train}(w^-, \alpha)}{2\epsilon}$

# Experiments

- Datasets:  CIFAR-10; PTB; ImageNet; WikiText-2 (WT2)
- Two stages:
  - Architecture search:  determine the best cells based on their validation performance
  - Architecture evaluation: use these cells to construct ***larger*** architectures, which we train from scratch and report their performance on the test set
- Transferability: use the best cells learned on CIFAR-10 and PTB, evaluating them on ImageNet and WikiText-2 (WT2) respectively

# Searching for convolutional cells on CIFAR-10

- Operations( both normal and reduction): 3×3 and 5×5 separable convolutions, 3×3 and 5×5 dilated separable convolutions, 3×3 max pooling, 3×3 average pooling, identity, and zero
- Stride: one for normal cells; two for reduction cells
- Use twice ReLU-Conv-BN order for convolutional operations
- Nodes: N=7
  - first and second nodes of cell k are set equal to the outputs of cell k−2 and cell k−1
  - output node is defined as the concatenation of all the intermediate nodes
- Cells located at the 1/3 and 2/3 of the total depth of the network are reduction cells

# Searching for convolutional cells on CIFAR-10

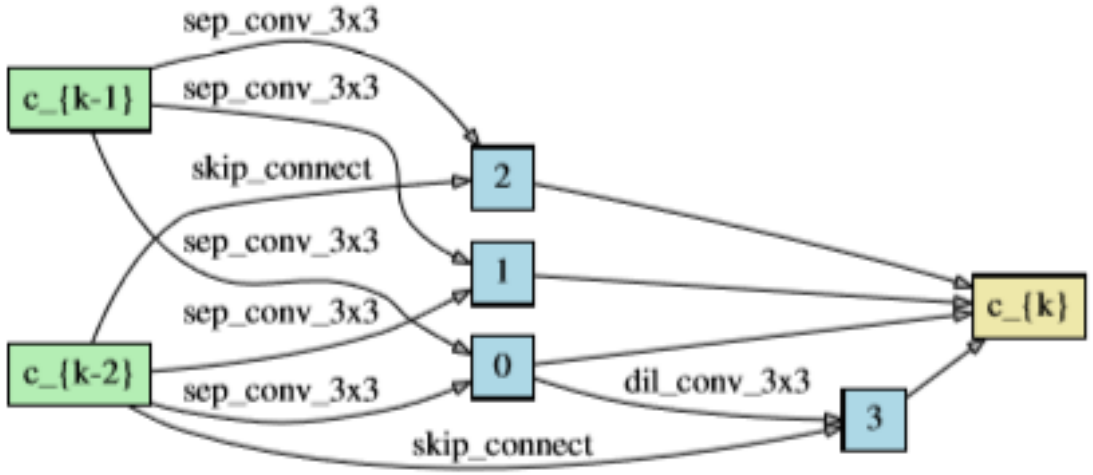



Figure 4: Normal cell learned on CIFAR-10.

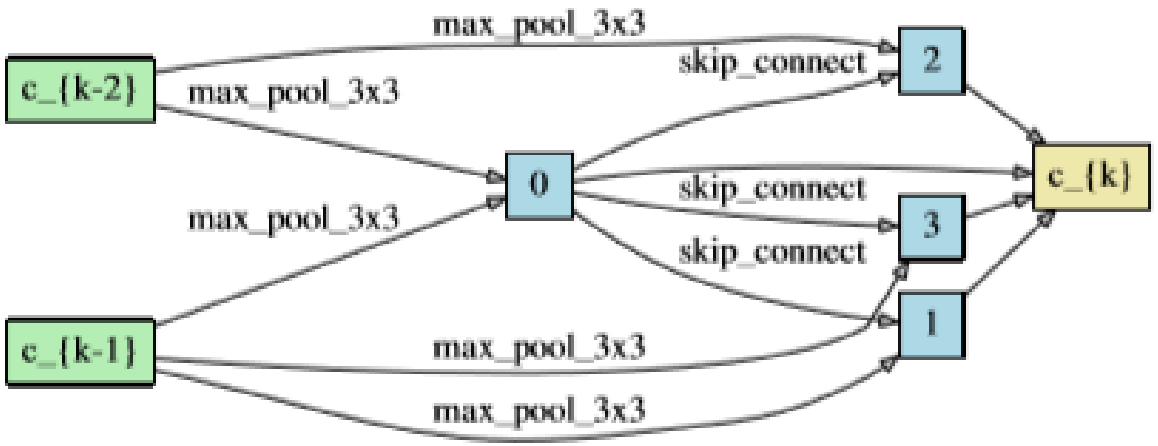



Figure 5: Reduction cell learned on CIFAR-10.

# Searching for recurrent cells on PTB

- Operations( both normal and reduction): linear transformations followed by one of tanh, relu, sigmoid activations, identity, zero
- Nodes: N=12
  - The very first intermediate node is obtained by linearly transforming the two input nodes, adding up the results and then passing through a tanh activation function (ENAS)
  - The rest of the cell is learned
  - Output is defined as the average of all the intermediate nodes
- Recurrent network consists of only a single cell
  - Do not assume any repetitive patterns within the recurrent architecture

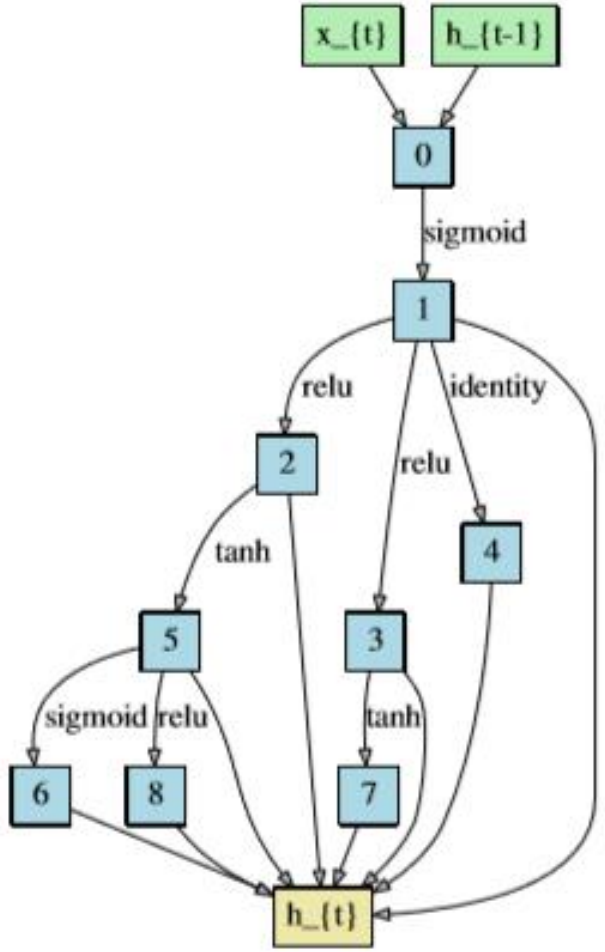



Figure 6: Recurrent cell learned on PTB.

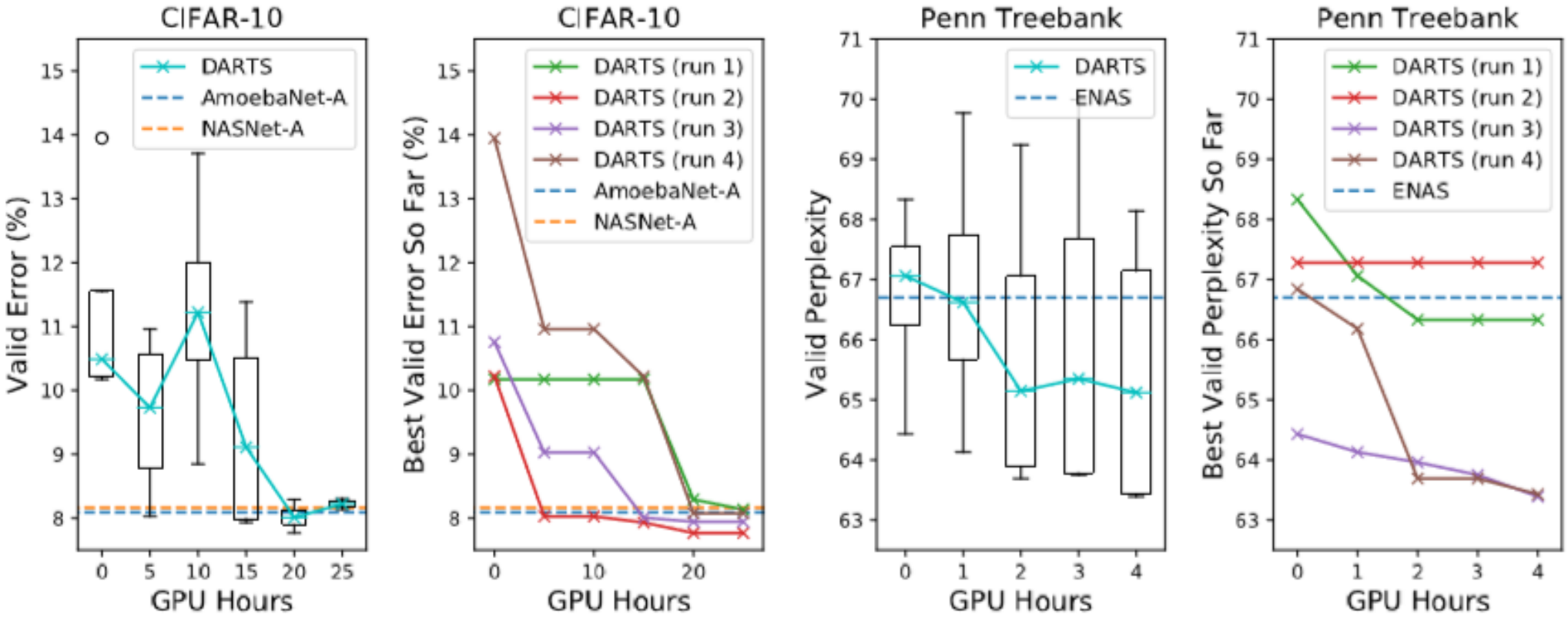



Figure 3: Search progress of DARTS for convolutional cells on CIFAR-10 and recurrent cells on Penn Treebank. We keep track of the most recent architectures over time. Each architecture snapshot is re-trained from scratch using the training set (for 100 epochs on CIFAR-10 and for 300 epochs on PTB) and then evaluated on the validation set. For each task, we repeat the experiments for 4 times with different random seeds, and report the median and the best (per run) validation performance of the architectures over time. As references, we also report the results (under the same evaluation setup; with comparable number of parameters) of the best existing cells discovered using RL or evolution, including NASNet-A (Zoph et al., 2018) (2000 GPU days), AmoebaNet-A (3150 GPU days) (Real et al., 2018) and ENAS (0.5 GPU day) (Pham et al., 2018b).

Table 1: Comparison with state-of-the-art image classifiers on CIFAR-10 (lower error rate is better). Note the search cost for DARTS does not include the selection cost (1 GPU day) or the final evaluation cost by training the selected architecture from scratch (1.5 GPU days).

| **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) | $3.34 \pm 0.06$ | 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) | $2.55 \pm 0.05$ | 2.8 | 3150 | 19 | evolution |
| Hierarchical evolution (Liu et al., 2018b) | $3.75 \pm 0.12$ | 15.7 | 300 | 6 | evolution |
| PNAS (Liu et al., 2018a) | $3.41 \pm 0.09$ | 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 | $3.29 \pm 0.15$ | 3.2 | 4 | 7 | random |
| DARTS (first order) + cutout | $3.00 \pm 0.14$ | 3.3 | 1.5 | 7 | gradient-based |
| DARTS (second order) + cutout | $2.76 \pm 0.09$ | 3.3 | 4 | 7 | gradient-based |

* Obtained by repeating ENAS for 8 times using the code publicly released by the authors. The cell for final evaluation is chosen according to the same selection protocol as for DARTS.

† Obtained by training the corresponding architectures using our setup.

‡ Best architecture among 24 samples according to the validation error after 100 training epochs.

# Results on ImageNet

Table 3: Comparison with state-of-the-art image classifiers on ImageNet in the mobile setting.

| Architecture | Test Error (%) | | Params | $+\times$ | 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 $2\times$ ($g = 3$) (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 |

# Results on WT2

Table 4: Comparison with state-of-the-art language models on WT2.

| Architecture | Perplexity valid | Perplexity test | Params (M) | Search Cost (GPU days) | Search Method |
|---|---|---|---|---|---|
| LSTM + augmented loss (Inan et al., 2017) | 91.5 | 87.0 | 28 | – | manual |
| LSTM + continuous cache pointer (Grave et al., 2016) | – | 68.9 | – | – | manual |
| LSTM (Merity et al., 2018) | 69.1 | 66.0 | 33 | – | manual |
| LSTM + skip connections (Melis et al., 2018) | 69.1 | 65.9 | 24 | – | manual |
| LSTM + 15 softmax experts (Yang et al., 2018) | 66.0 | 63.3 | 33 | – | manual |
| ENAS (Pham et al., 2018b)† (searched on PTB) | 72.4 | 70.4 | 33 | 0.5 | RL |
| DARTS (searched on PTB) | 71.2 | 69.6 | 33 | 1 | gradient-based |

† Obtained by training the corresponding architecture using our setup.

# Conclusion

- DARTS: a simple yet efficient architecture search algorithm for both convolutional and recurrent networks
- By searching in a continuous space, DARTS is able to match or outperform the state-of-the-art non-differentiable architecture search methods on image classification and language modeling tasks with remarkable efficiency improvement by several orders of magnitude.

# END

# THANK YOU