Efficient Neural Architecture Search via Parameter Sharing

Hieu Pham [*1 2] **Melody Y. Guan** [*3] **Barret Zoph** [1] **Quoc V. Le** [1] **Jeff Dean** [1]

[*]Equal contribution [1]Google Brain [2]Language Technology Institute, Carnegie Mellon University [3]Department of Computer Science, Stanford University. Correspondence to: Hieu Pham <hyhieu@cmu.edu>, Melody Y. Guan <mguan@stanford.edu>.

ICML 2018

Total cites : 1268

2021.05.12

최영제

Index

1 Introduction

Background – NAS

- Neural Architecture Search (NAS) is a gradient-based method for finding good architectures
- NAS use a recurrent network – the controller – to generate string which denoted hyperparameter of architectures
- Training the network specified by the string – the “child network” – on the real data will result in an accuracy on a validation set

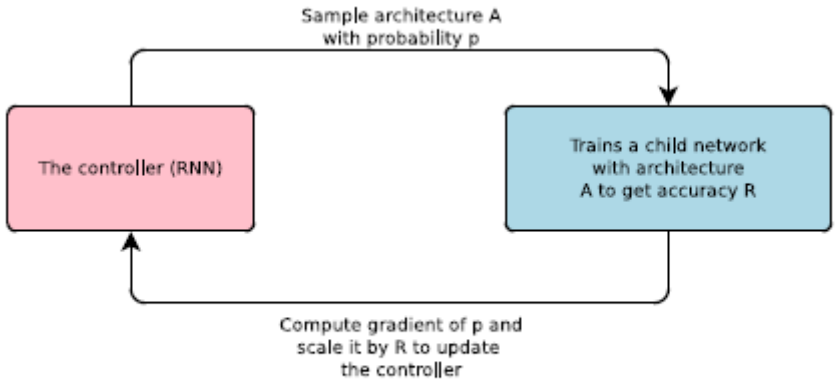


Figure 1: An overview of Neural Architecture Search.

Introduction

Background – NAS

- RNN structure
- Every prediction is carried out by a softmax classifier
- Every output is fed into the next time step as input
- If the number of layers exceeds a certain value, the process of generating an architecture stops

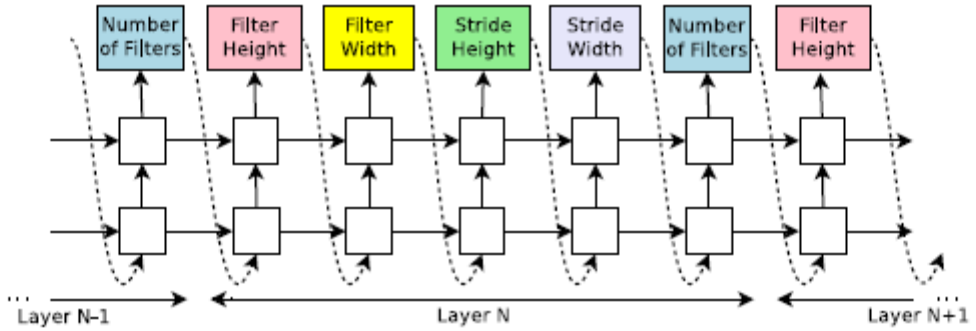


Figure 2: How our controller recurrent neural network samples a simple convolutional network. It predicts filter height, filter width, stride height, stride width, and number of filters for one layer and repeats. Every prediction is carried out by a softmax classifier and then fed into the next time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

1 Introduction

Background – NAS

- The parameters of the controller recurrent neural network (RNN) is noted θ_c

  → Have to optimized in order to maximize the validation accuracy of the proposed architectures

- For optimized the parameters, policy gradient methods is used

$$\nabla_{\theta_c} J(\theta_c) = \sum_{t=1}^{T} E_{P(a_{1:T};\theta_c)} \left[\nabla_{\theta_c} \log P(a_t | a_{(t-1):1}; \theta_c) R \right]$$

- An empirical approximation of the above quantity is follow

$$\frac{1}{m} \sum_{k=1}^{m} \sum_{t=1}^{T} \nabla_{\theta_c} \log P(a_t | a_{(t-1):1}; \theta_c) R_k$$

m is number of different architectures (episode)
T is number of hyperparameters our controller has to predict to design a neural network (length of RNN per episode)
R is the accuracy which achieved on a held-out dataset using child network (R_k : validation score of k-th neural network architecture)

Background – NAS

- Above gradient has high variance, thus using baseline for reduce variance

$$\frac{1}{m}\sum_{k=1}^{m}\sum_{t=1}^{T}\nabla_{\theta_c}\log P(a_t|a_{(t-1):1};\theta_c)(R_k-b)$$

- Using exponential moving average of the previous architecture accuracies for baseline function b
- As training a child network can take hours, we use distributed training and asynchronous parameter updates in order to speed up the learning process of the controller

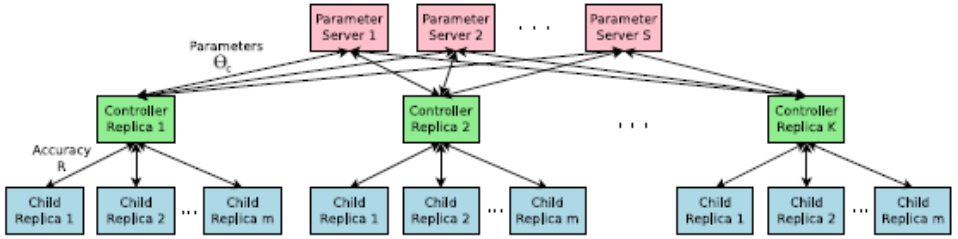


Figure 3: Distributed training for Neural Architecture Search. We use a set of S parameter servers to store and send parameters to K controller replicas. Each controller replica then samples m architectures and run the multiple child models in parallel. The accuracy of each child model is recorded to compute the gradients with respect to θ_c, which are then sent back to the parameter servers.

2 SIL Methods

ENAS

- The main contribution of efficient neural architecture search (ENAS) is 1,000x faster than NAS via parameter sharing
- We can represent NAS’s search space using a single directed acyclic graph (DAG)
- ENAS's search spaces are represented sub-graph of a larger graph

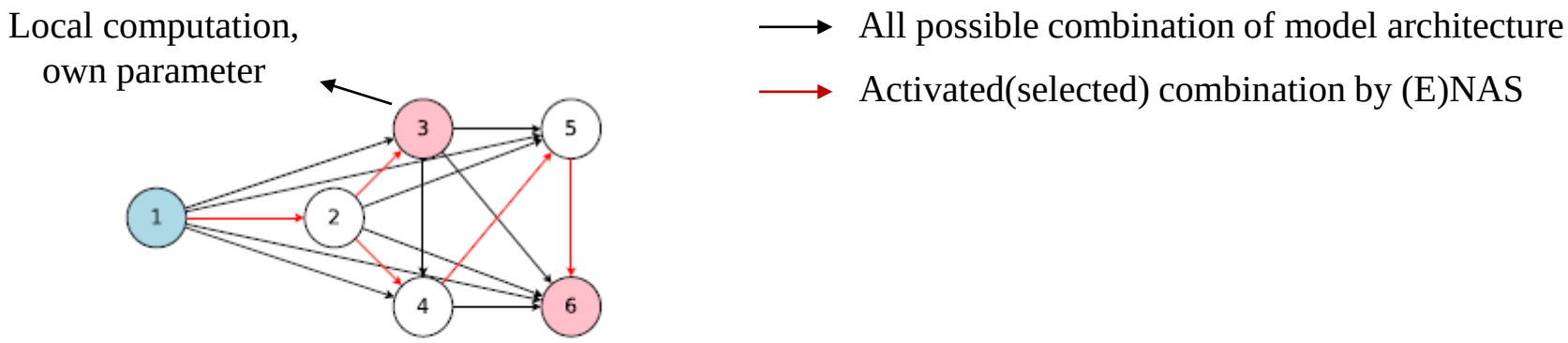


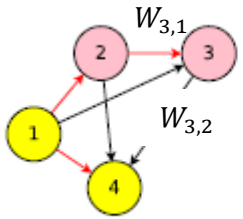
Figure 2. The graph represents the entire search space while the red arrows define a model in the search space, which is decided by a controller. Here, node 1 is the input to the model whereas nodes 3 and 6 are the model’s outputs.

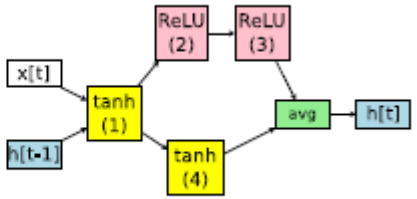
2 Methods

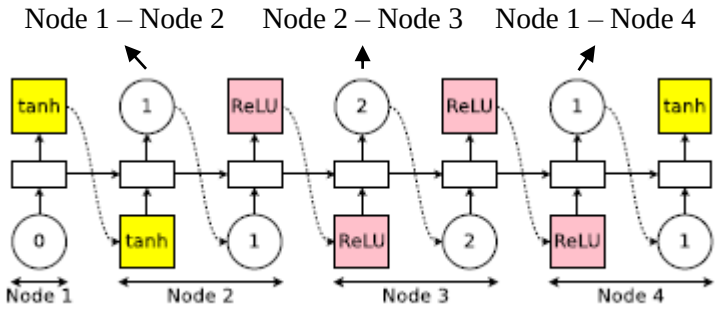
ENAS – designing recurrent cells

- ENAS's controller decides 1) what previous node to connect to and 2) what activation function to use at each node in the DAG

Construction example of single LSTM cell







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 $h_1 = \tanh\left(\mathbf{x}_t \cdot \mathbf{W}^{(\mathbf{x})} + \mathbf{h}_{t-1} \cdot \mathbf{W}_1^{(\mathbf{h})}\right)$.
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 $h_2 = \text{ReLU}(h_1 \cdot \mathbf{W}_{2,1}^{(\mathbf{h})})$.
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, $h_3 = \text{ReLU}(h_2 \cdot \mathbf{W}_{3,2}^{(\mathbf{h})})$.
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 $h_4 = \tanh\left(h_1 \cdot \mathbf{W}_{4,1}^{(\mathbf{h})}\right)$.
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, $\mathbf{h}_t = (h_3 + h_4)/2$.

$$h_i = activation(h_{i\text{-}1} \cdot W_{to,from})$$ i does not mean time-step t

Methods

Training procedure of ENAS

- Training procedure of ENAS has two phase, 1) training the shared parameters ω of the child models and 2) training the controller parameters θ
- When training ω, fix the controller's policy $\pi(\boldsymbol{m};\theta)$ and minimize the expected loss function $\mathbb{E}_{\boldsymbol{m}\sim\pi}[\mathcal{L}(\boldsymbol{m};\omega)]$

cross-entropy loss, computed on a minibatch of traning data

- The gradient is computed using the Monte Carlo estimate as follow

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

Update ω by sampling

- $\boldsymbol{m}_i$ is sampled model from fixed policy $\pi(\boldsymbol{m};\theta)$, surprisingly they find that $M = 1$ works just fine
- When training θ, fix the child model's parameter ω, aiming to maximize the expected reward $\mathbb{E}_{\boldsymbol{m}\sim\pi(\boldsymbol{m};\theta)}[\mathcal{R}(\boldsymbol{m};\omega)]$

$$\frac{1}{m}\sum_{k=1}^{m}\sum_{t=1}^{T}\nabla_{\theta_c}\log P(a_t|a_{(t-1):1};\theta_c)(R_k - b)$$

computed on a minibatch of the validation set

2 SIL Methods

Deriving architectures

- To derive novel architectures from a trained ENAS model,
  1) Sampling several models from the trained policy $\pi(\boldsymbol{m}; \theta)$
  2) For each sampled model, compute its reward on a single minibatch from the validation set
  3) Take only the model with the highest reward to re-train from scratch
- It is possible to improve our experimental results by training all the sampled models from scratch and selecting the model with the highest performance on a separated validation set
- However, our method yields similar performance whilst being much more economical

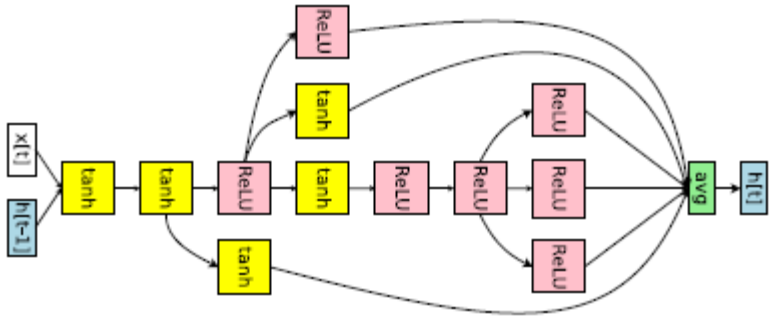
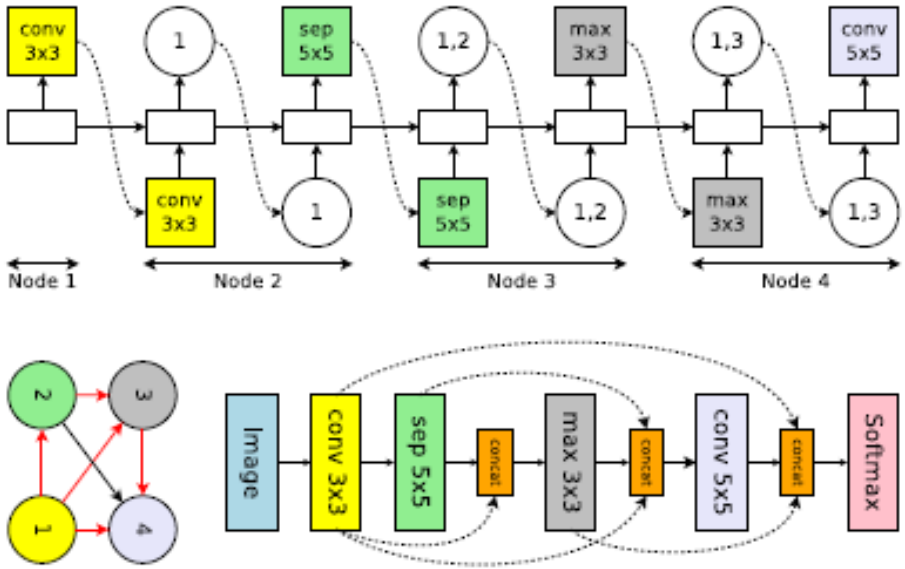


Figure 6. The RNN cell ENAS discovered for Penn Treebank.

ENAS – designing convolutional networks

- ENAS's controller decides 1) what previous nodes to connect to and 2) what computation operation to use at each node in the DAG
- They have six operation to design each layer of convolutional networks
  - 3 x 3 or 5 x 5 conv filters
  - 3 x 3 or 5 x 5 depthwise-separable conv filters
  - Average or max pooling with 3 x 3 kernal size



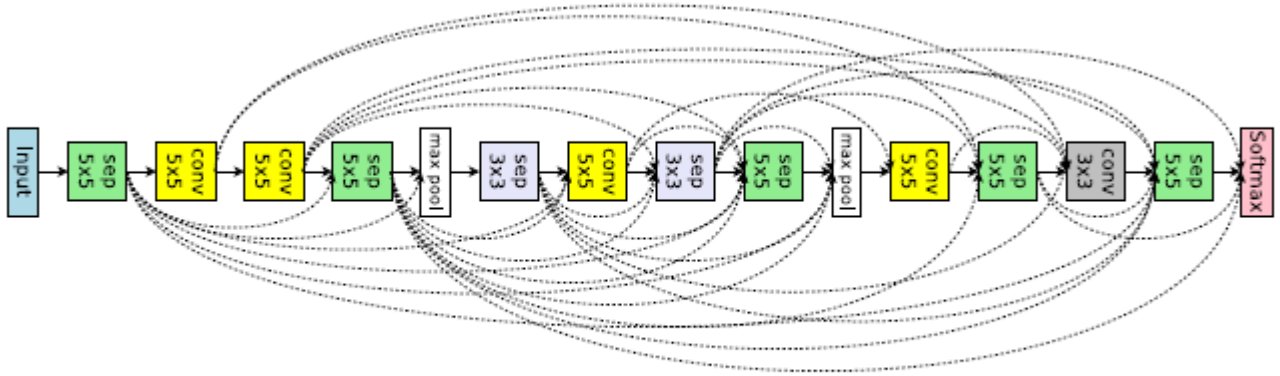


Figure 7. ENAS's discovered network from the macro search space for image classification.

2 Methods

ENAS – designing convolutional cells

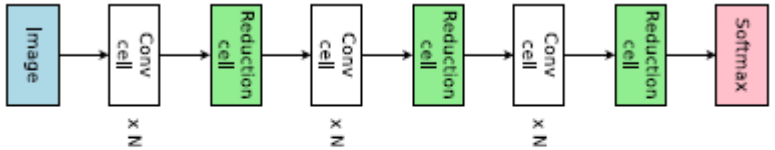
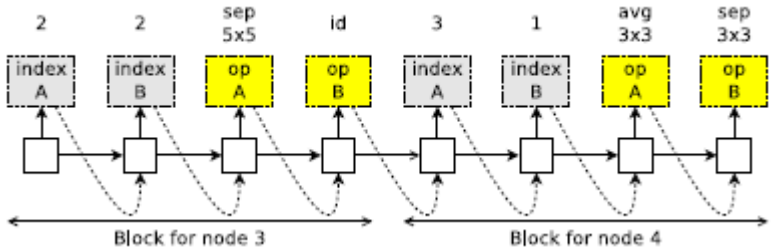
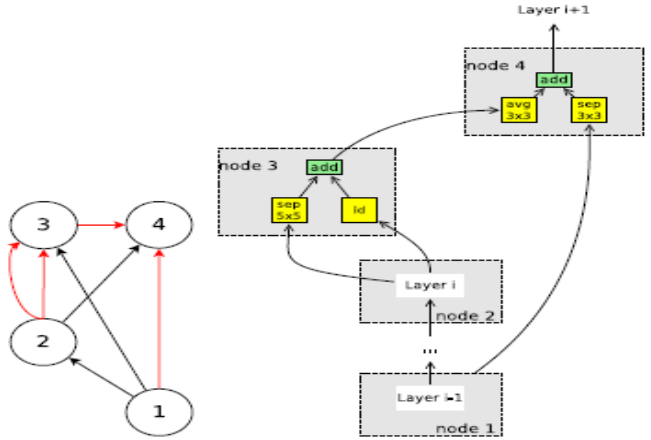


Figure 4. Connecting 3 blocks, each with N convolution cells and 1 reduction cell, to make the final network.

- Rather than designing the entire convolutional network, one can design smaller modules(3 conv cell, 1 reduction cell) and then connect them together to form a network (=micro search approach, TNAS)

Construction example of single conv cell





1. Nodes 1, 2 are input nodes, so no decisions are needed for them. Let h_1, h_2 be the outputs of these nodes.

2. At node 3: the controller samples two previous nodes and two operations. In Figure 5 *Top Left*, it samples *node 2*, *node 2*, *separable_conv_5x5*, and *identity*. This means that $h_3 = \text{sep\_conv\_5x5}(h_2) + \text{id}(h_2)$.

3. At node 4: the controller samples *node 3*, *node 1*, *avg_pool_3x3*, and *sep_conv_3x3*. This means that $h_4 = \text{avg\_pool\_3x3}(h_3) + \text{sep\_conv\_3x3}(h_1)$.

4. Since all nodes but h_4 were used as inputs to at least another node, the only loose end, h_4, is treated as the cell's output. If there are multiple loose ends, they will be concatenated along the depth dimension to form the cell's output.

ENAS – designing convolutional cells

- Each architecture of ENAS's cell

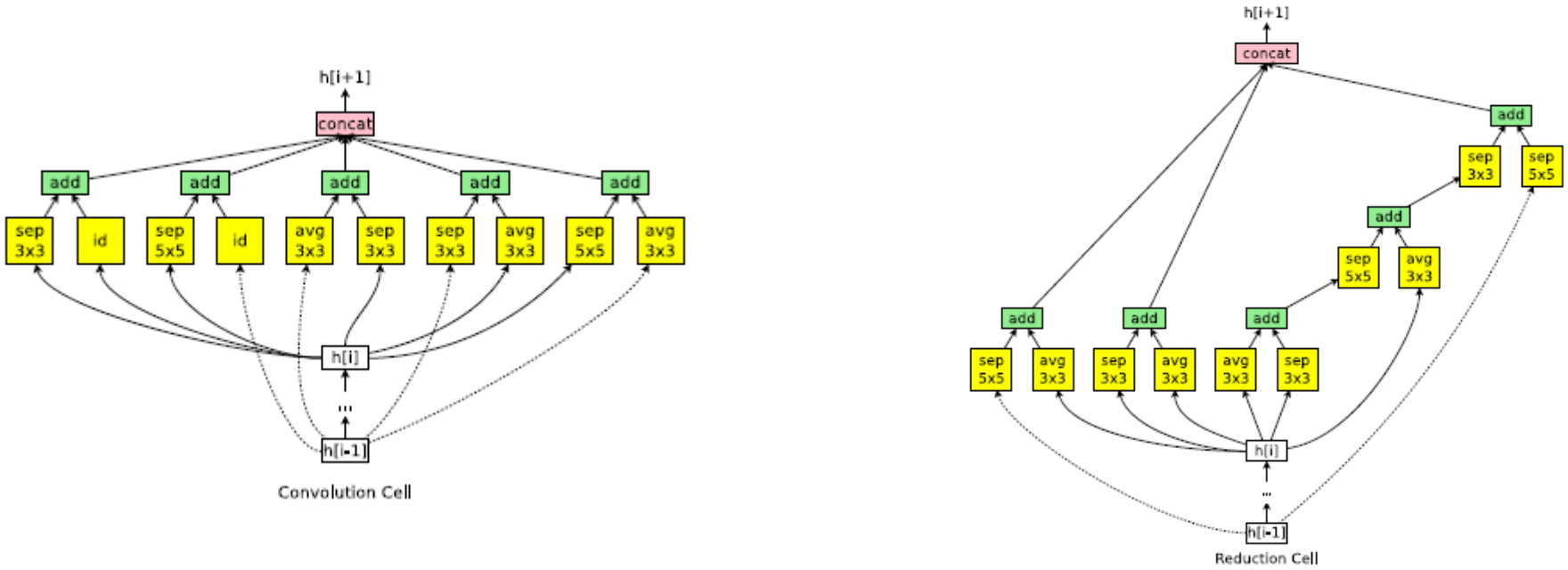


Figure 8. ENAS cells discovered in the micro search space.

Experiments and Results

Language model with Penn Treebank

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
LSTM (Melis et al., 2017)	Hyper-parameters Search	24	59.5
LSTM (Yang et al., 2018)	VD, WT, ℓ_2, AWD, MoC	22	57.6
LSTM (Merity et al., 2017)	VD, WT, ℓ_2, AWD	24	57.3
LSTM (Yang et al., 2018)	VD, WT, ℓ_2, AWD, MoS	**22**	**56.0**
RHN (Zilly et al., 2017)	VD, WT	24	66.0
NAS (Zoph & Le, 2017)	VD, WT	54	62.4
ENAS	VD, WT, ℓ_2	**24**	**55.8**

Table 1. Test perplexity on Penn Treebank of ENAS and other baselines. Abbreviations: RHN is *Recurrent Highway Network*, VD is *Variational Dropout*; WT is *Weight Tying*; ℓ_2 is *Weight Penalty*; AWD is *Averaged Weight Drop*; MoC is *Mixture of Contexts*; MoS is *Mixture of Softmaxes*.

3 Experiments and Results

Image classification 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**

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.

Q&A