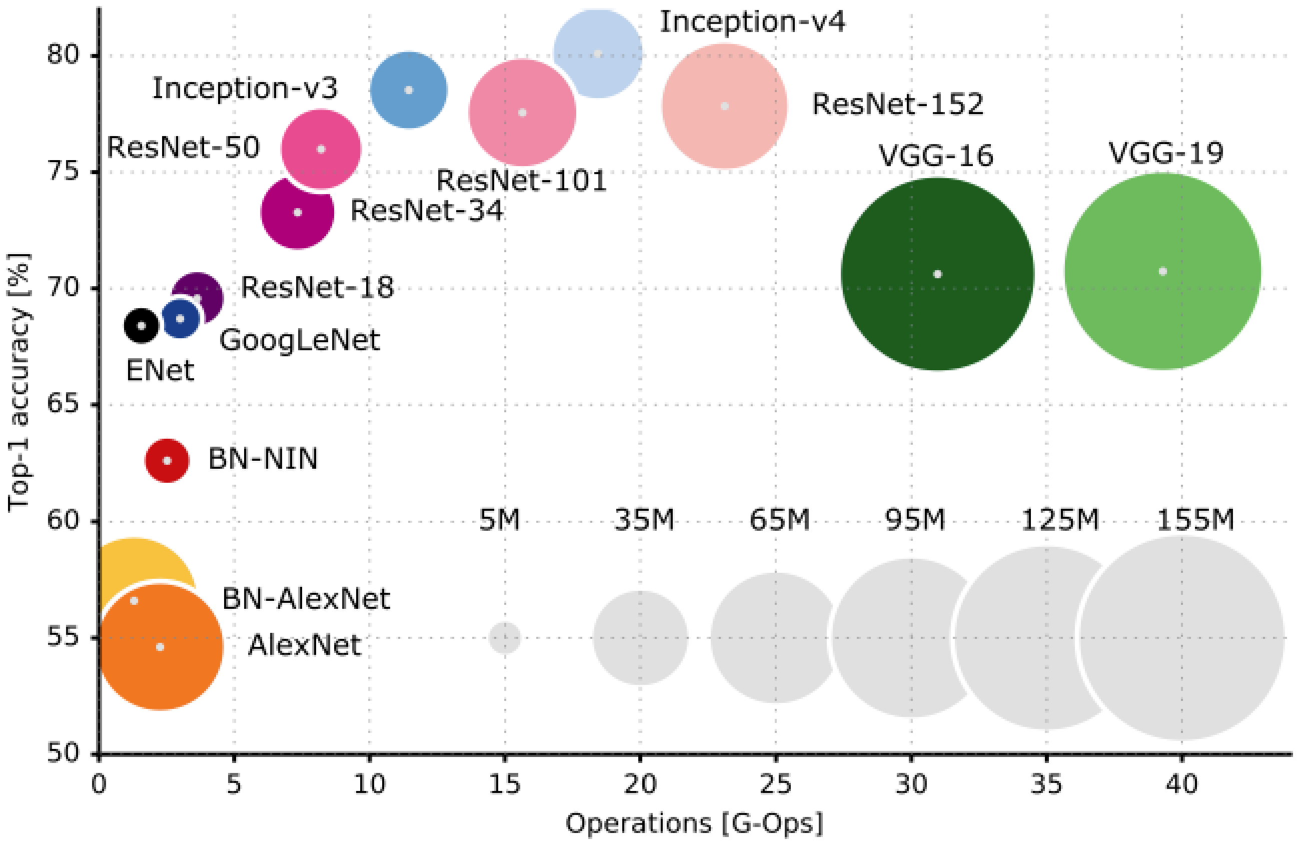
Neural Architecture Search

Changhoon Jeong

AI lab., Paper Seminar,
April 2018

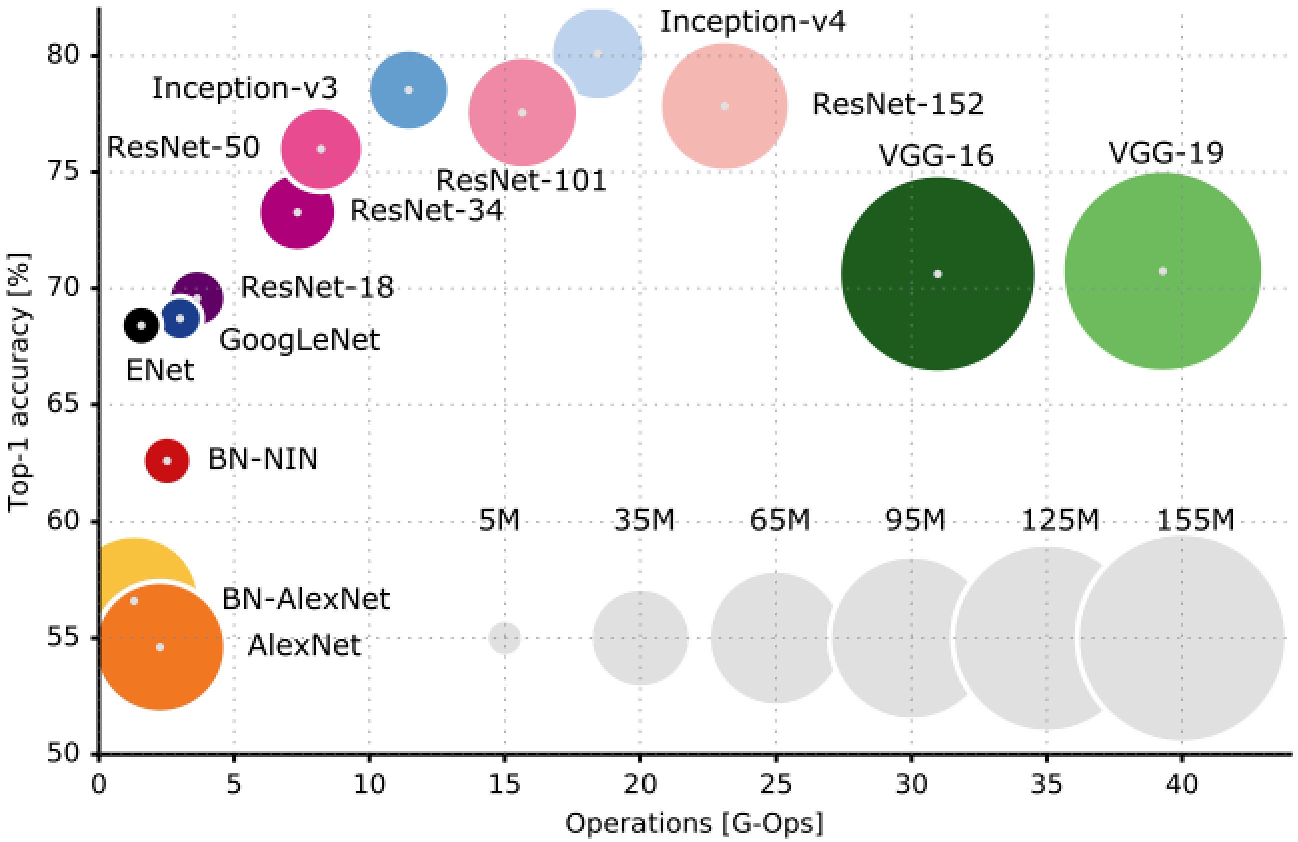
Introduction

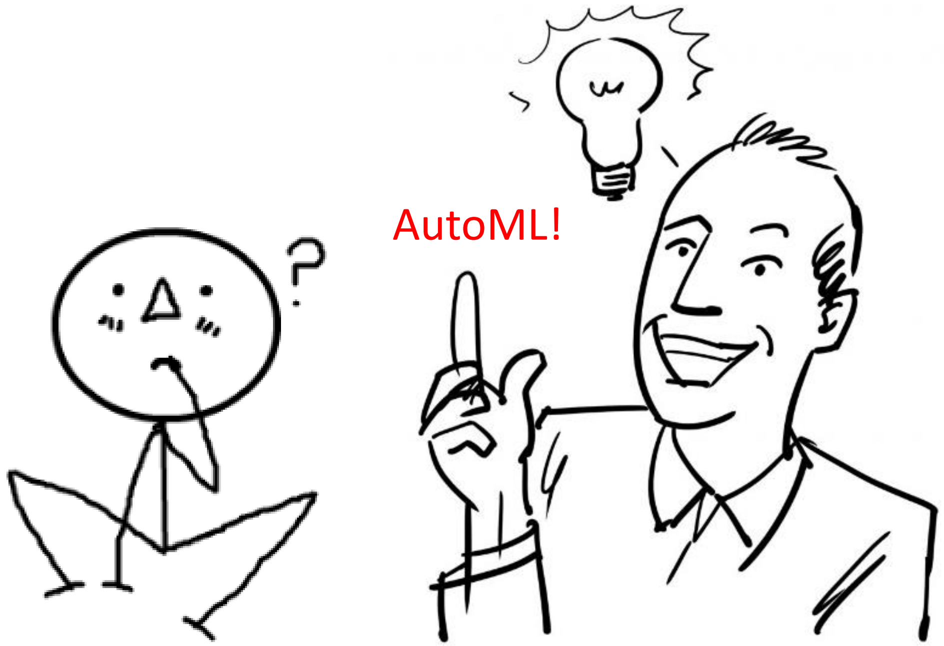
"We have experienced many state-of-the-art models, but we rarely understand the generalization of Neural Networks"



Introduction

"We have experienced many state-of-the-art models, but we rarely understand the generalization of Neural Networks"





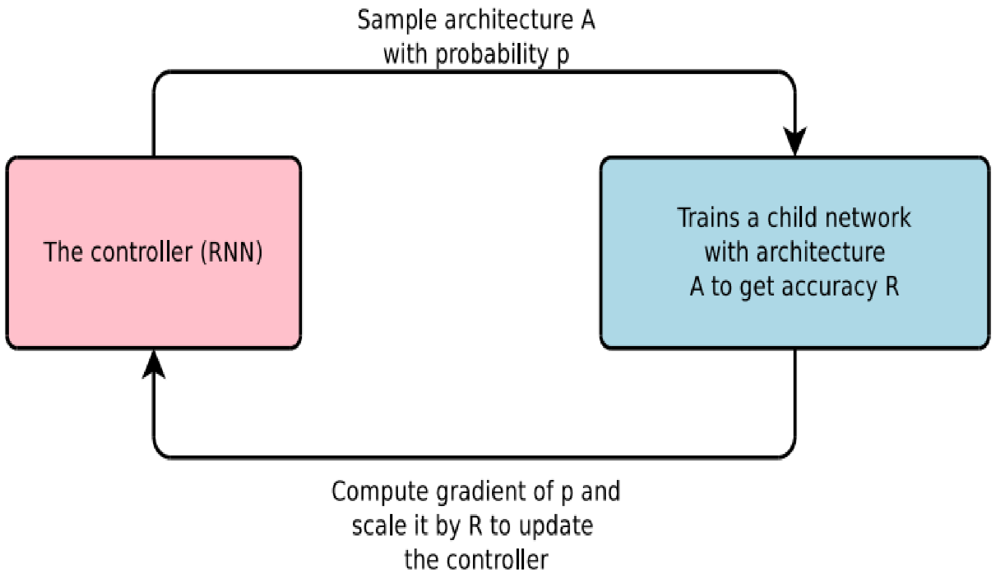
Introduction

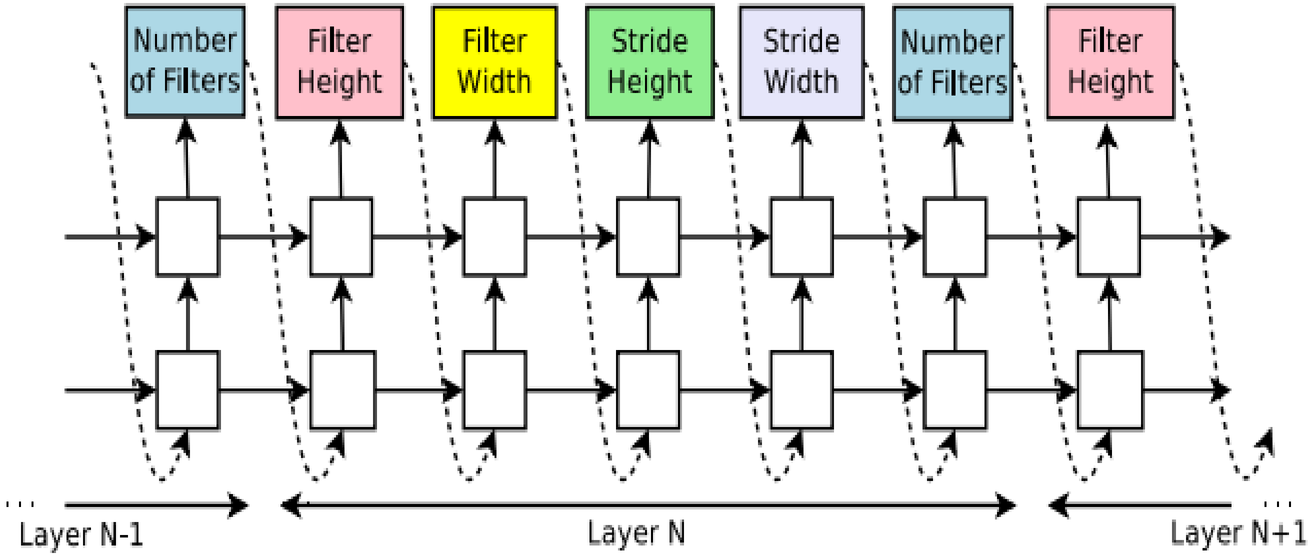
- Neural Architecture Search with Reinforcement Learning
  - Barret Zoph, Quoc V. Le (Google Brain)
  - arXiv, 15 Feb 2017
  - ICLR 2017, Oral Presentation

- Efficient Neural Architecture Search via Parameter Sharing
  - Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, Jeff Dean
  - arXiv, 12 Feb 2018

Neural Architecture Search with Reinforcement Learning

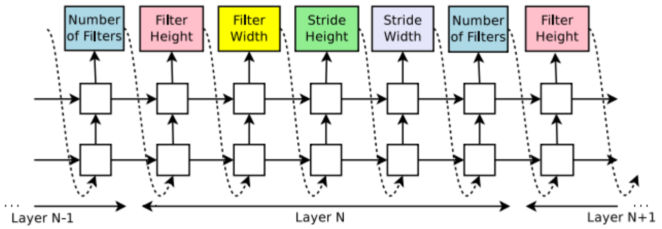
❖ An overview of Neural Architecture Search



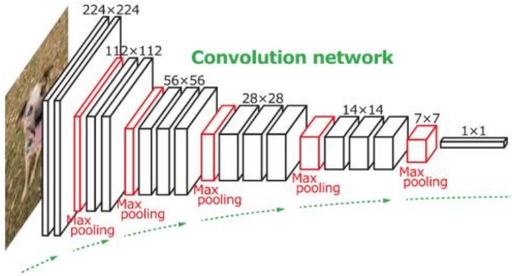


Neural Architecture Search with Reinforcement Learning

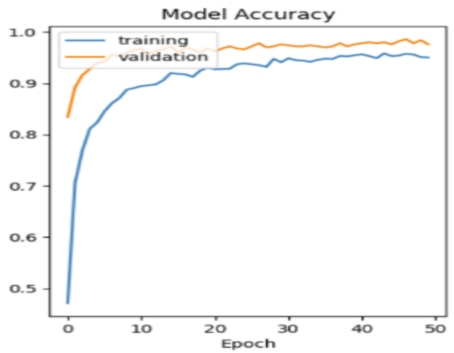
❖ The flow of Neural Architecture Search



RNN Controller

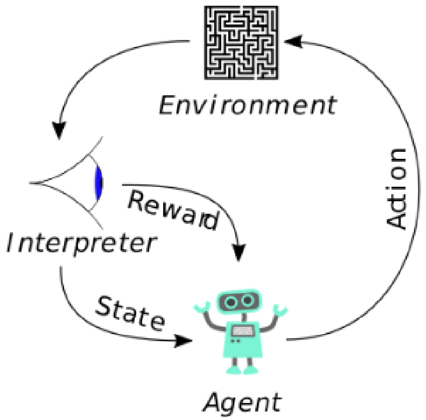


Child Network(Training)



Validation set Accuracy

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



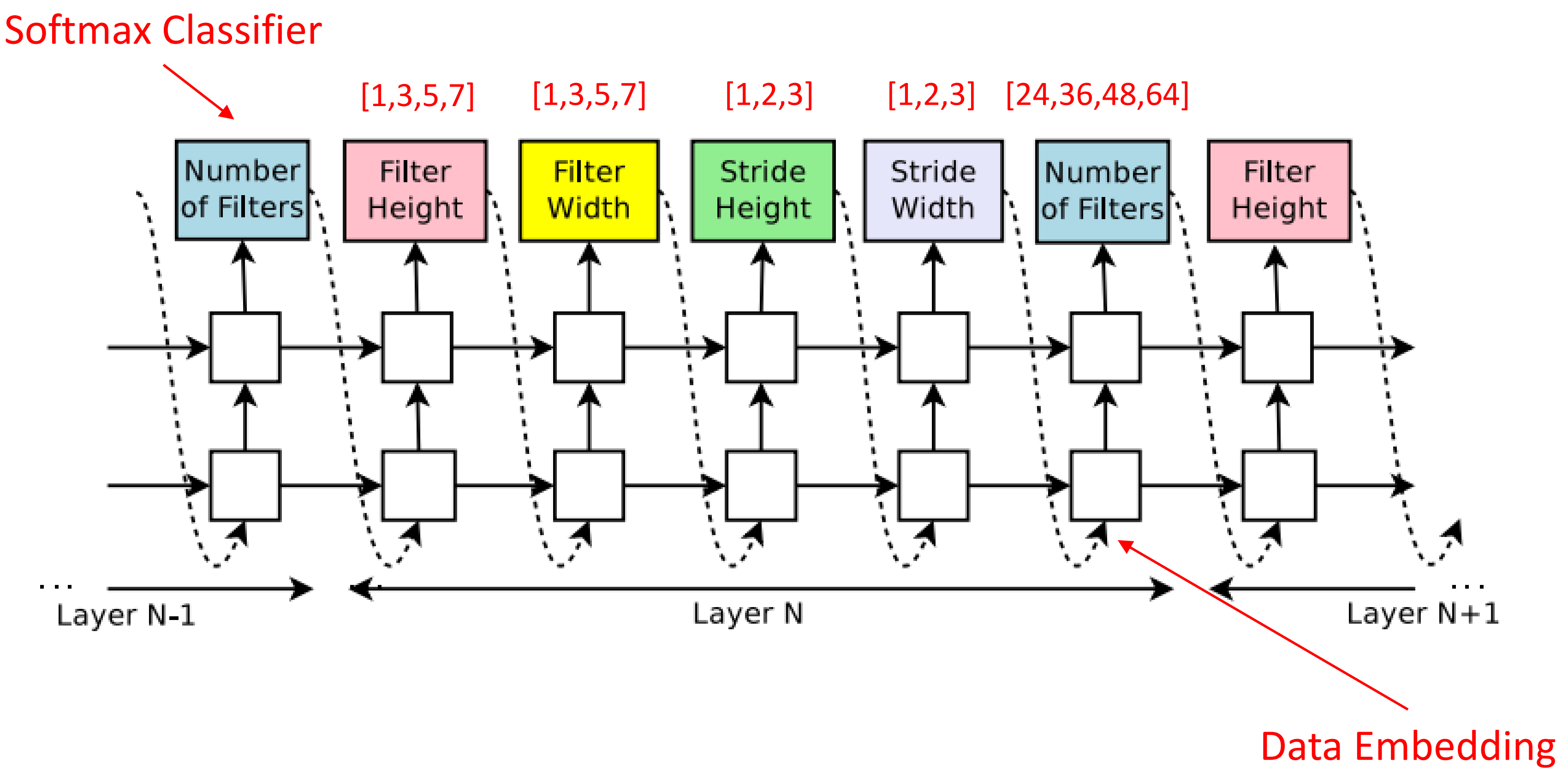
Reinforcement Learning for
Update Controller's Parameter

$$J(\theta_c) = E_{P(a_{1:T};\theta_c)}[R]$$

REINFORCE Algorithm

Neural Architecture Search with Reinforcement Learning

❖ Meta-Learner Model(Controller RNN)



Neural Architecture Search with Reinforcement Learning

❖ Monte-Carlo Policy Gradient(REINFORCE)

- Update parameters by stochastic gradient ascent
- Using policy gradient theorem
- Using return v_t as an unbiased sample of $Q^{\pi\theta}(s_t, a_t)$
- $\Delta\theta_t = \nabla_\theta log\pi_\theta(s_t, a_t)\, v_t$

function REINFORCE
    Initialise θ arbitrarily
    for each episode $\{s_1, a_1, r_2, ..., s_{T-1}, a_{T-1}, r_T\} \sim \pi_\theta$ **do**
        for $t = 1$ to $T - 1$ **do**
            $\theta \leftarrow \theta + \alpha\nabla_\theta \log \pi_\theta(s_t, a_t)v_t$
        end for
    end for
    return θ
end function

Neural Architecture Search with Reinforcement Learning

- REINFORCE Algorithm in Neural Architecture Search

$$J(\theta_c) = E_{P(a_{1:T};\theta_c)}[R]$$

Parameters of Controller RNN

Accuracy of architecture on held-out dataset

Architecture predicted by controller RNN viewed as a sequence of actions

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

Baseline : for reduce high variance

Number of models in minibatch

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

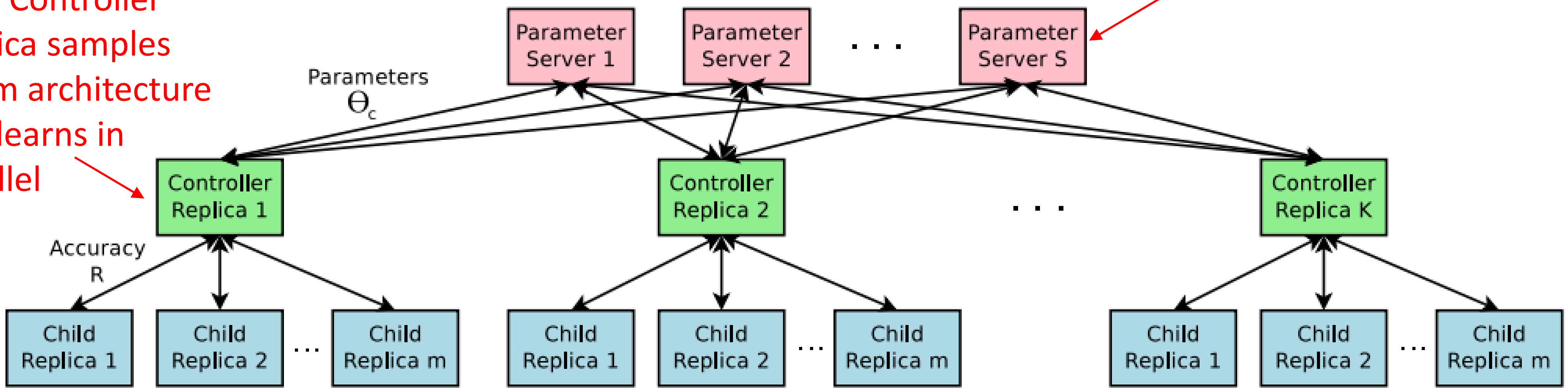
Neural Architecture Search with Reinforcement Learning

❖ Accelerate Training with Parallelism and Asynchronous Updates

Distributed Learning

Each Controller Replica samples the m architecture and learns in parallel

The Parameter Server (total 10) stores the parameters and sends(or receive) them to the Controller Replica



- Used 800 GPU
- It takes 2-3 weeks to learn 13,000-15,000 models

Neural Architecture Search with Reinforcement Learning

❖ Increase Architecture Complexity : Skip Connections

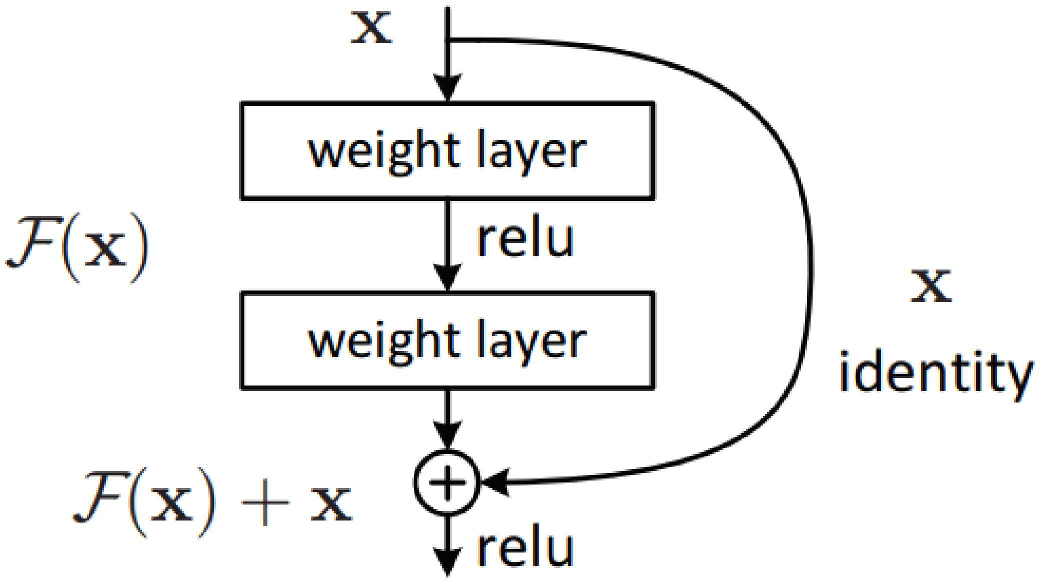
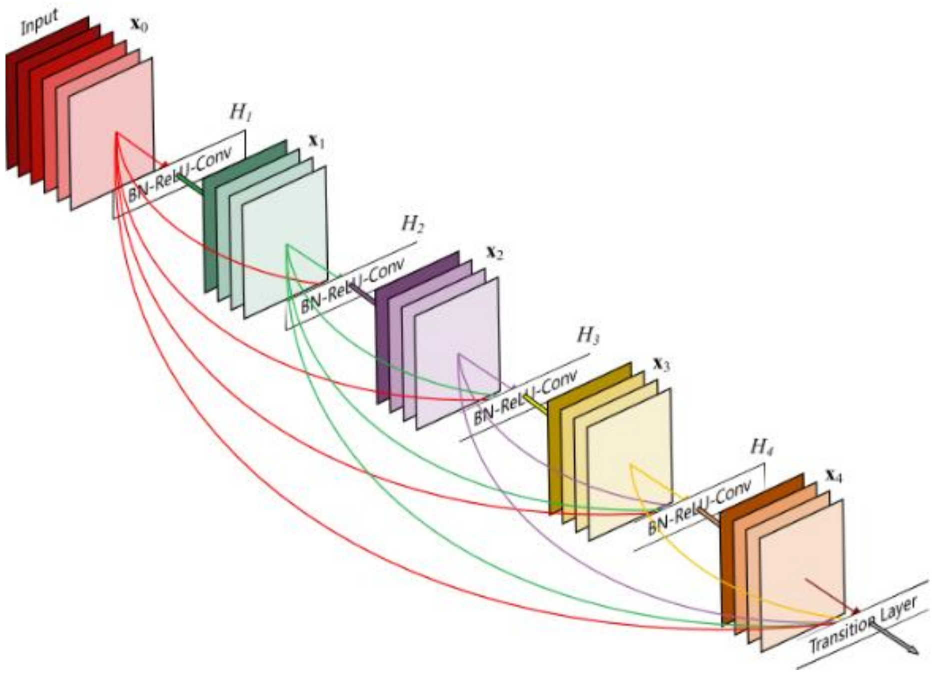


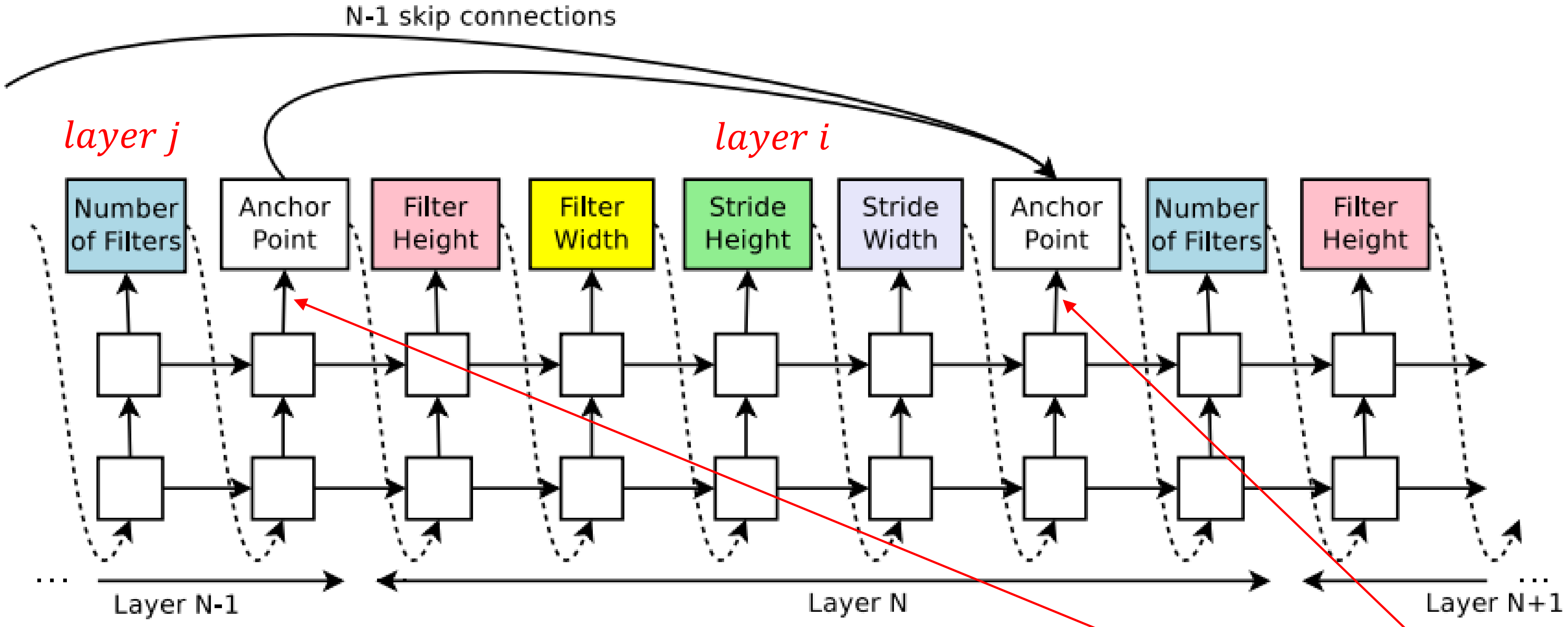
Figure 2. Residual learning: a building block.

ResNet or DenseNet has skip connections



Neural Architecture Search with Reinforcement Learning

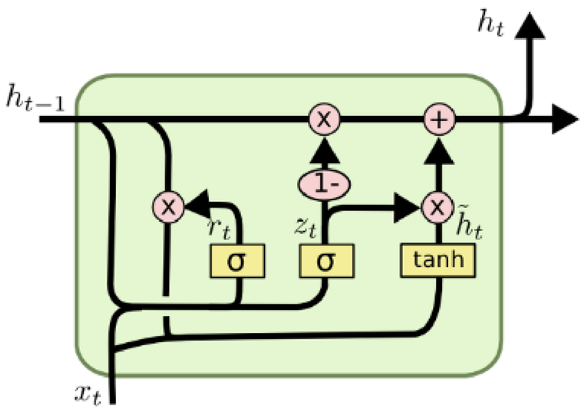
❖ Increase Architecture Complexity : Skip Connections



$$\mathrm{P(Layer\ j\ is\ an\ input\ to\ layer\ i)} = \mathrm{sigmoid}(v^{\mathrm{T}}\tanh(W_{prev} * h_j + W_{curr} * h_i))$$

Neural Architecture Search with Reinforcement Learning

❖ Generate Recurrent Cell Architectures

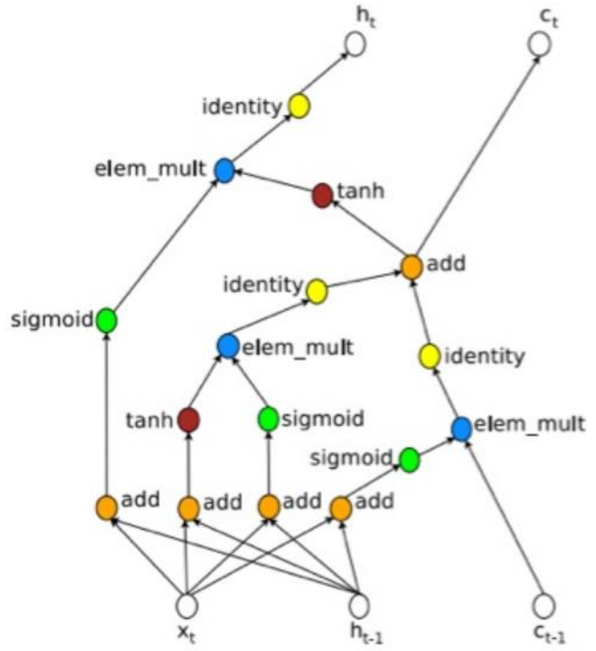


$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

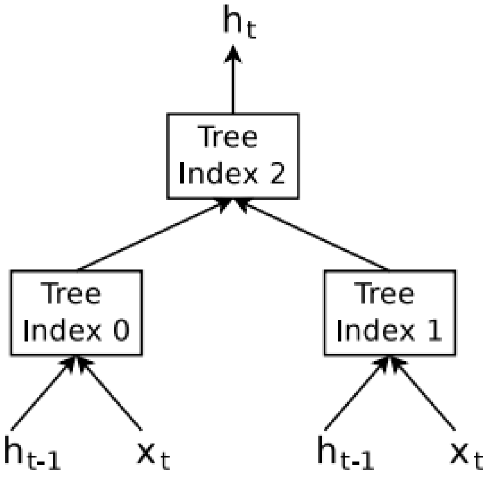
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

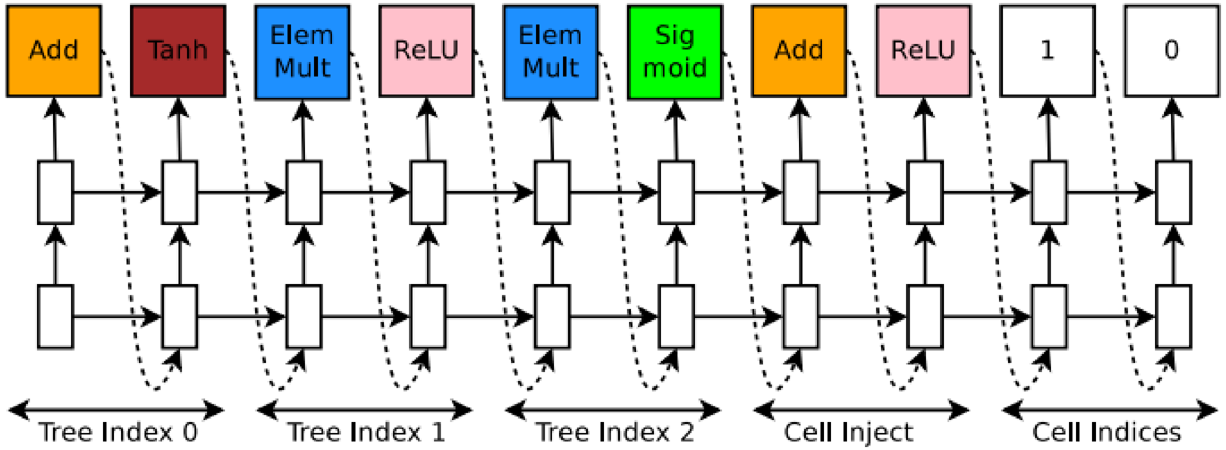


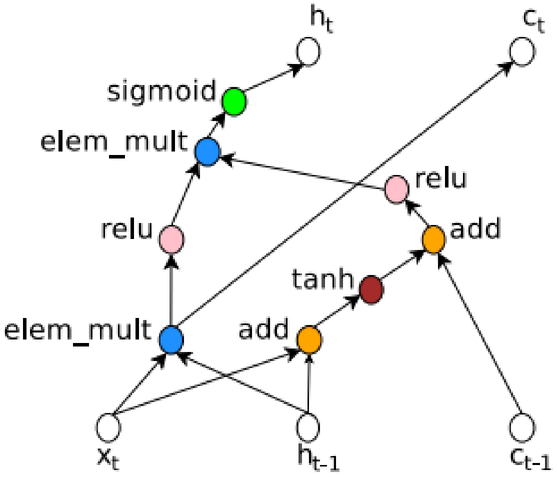
- In order to find an RNN cell similar to LSTM or GRU, a search space was created by referring to the LSTM cell
- Modeling a step of tree : take x_t and h_{t-1} as inputs and produce final output h_t

Neural Architecture Search with Reinforcement Learning

❖ Generate Recurrent Cell Architectures







- Express the graph's operation as a tree
- Decide which activation function or operation to use

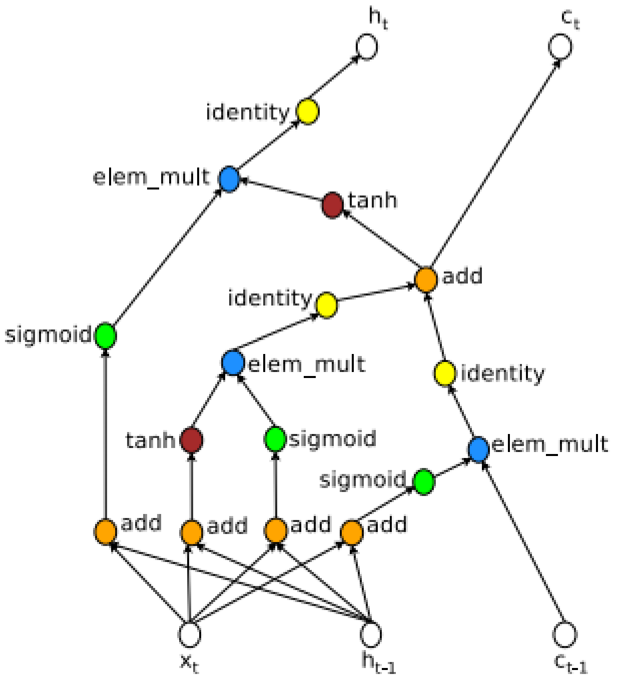
- Controller RNN predicts the combining method and determines the label of the tree
- Controller RNN refers to the tree to determine which function to select and generate

$$h_t = \tanh(W_1 * x_t + W_2 * h_{t-1})$$ <- Index 0

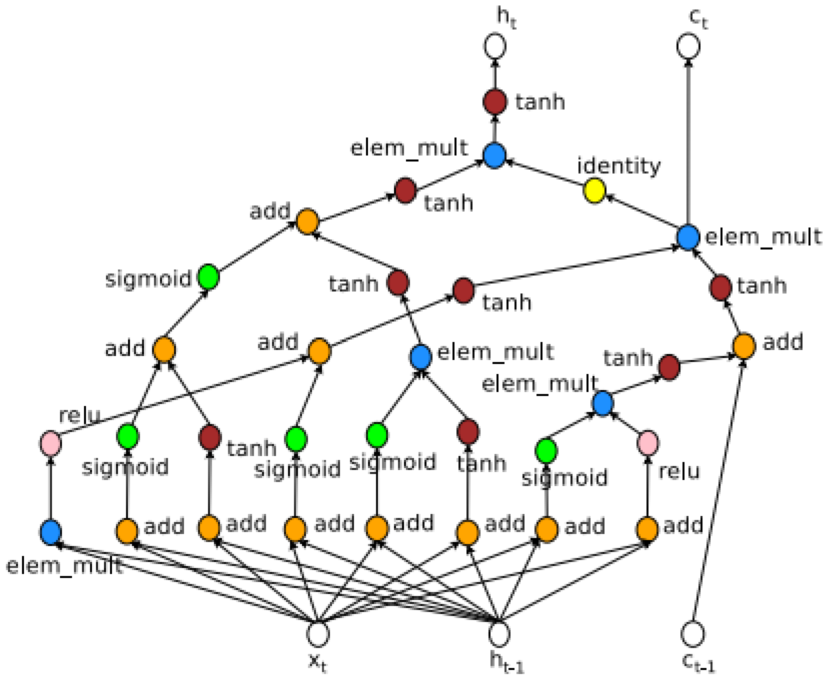
- Once the tree is created, the architecture of the RNN cell is implemented

Neural Architecture Search with Reinforcement Learning

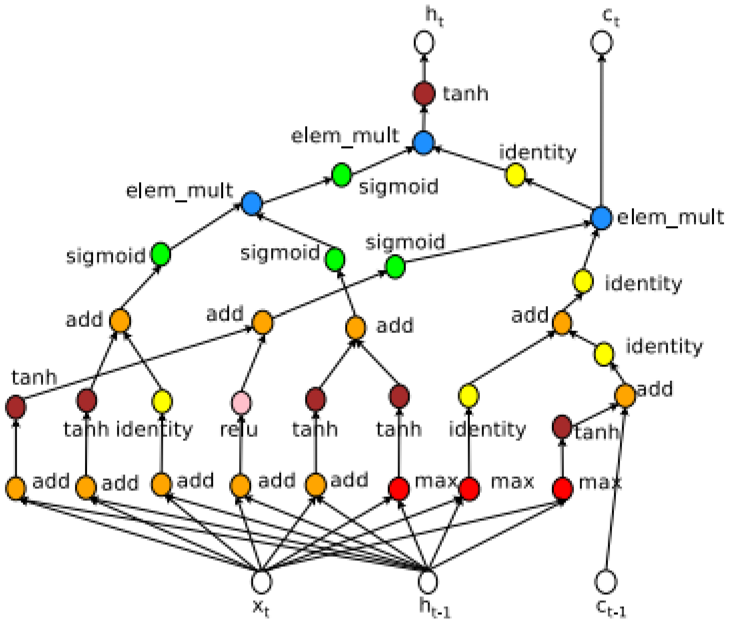
❖ Generate Recurrent Cell Architectures



LSTM Cell



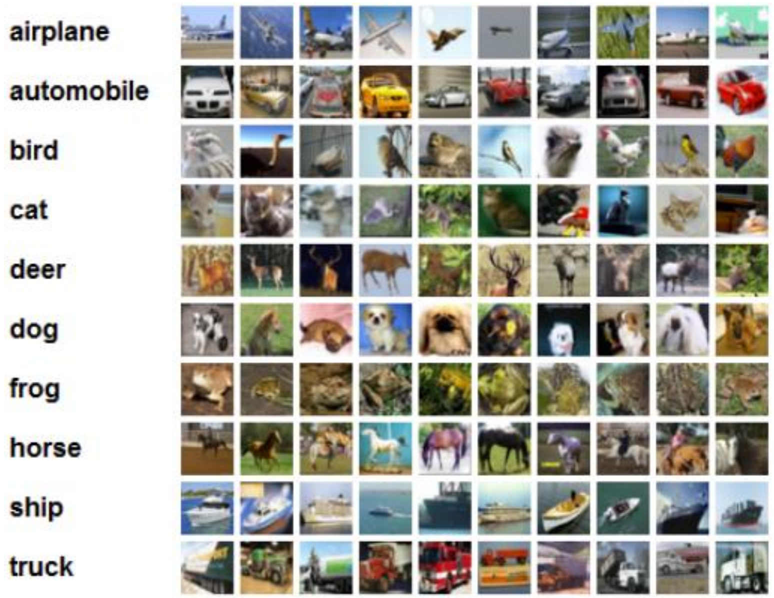
NAS Cell(Does not include max and sin operations)



NAS Cell(Include max and sin operations, but NAS does not use sin)

Neural Architecture Search with Reinforcement Learning

❖ Experiment Details & Results



Convolutional Neural Architecture Search for CIFAR-10 Dataset

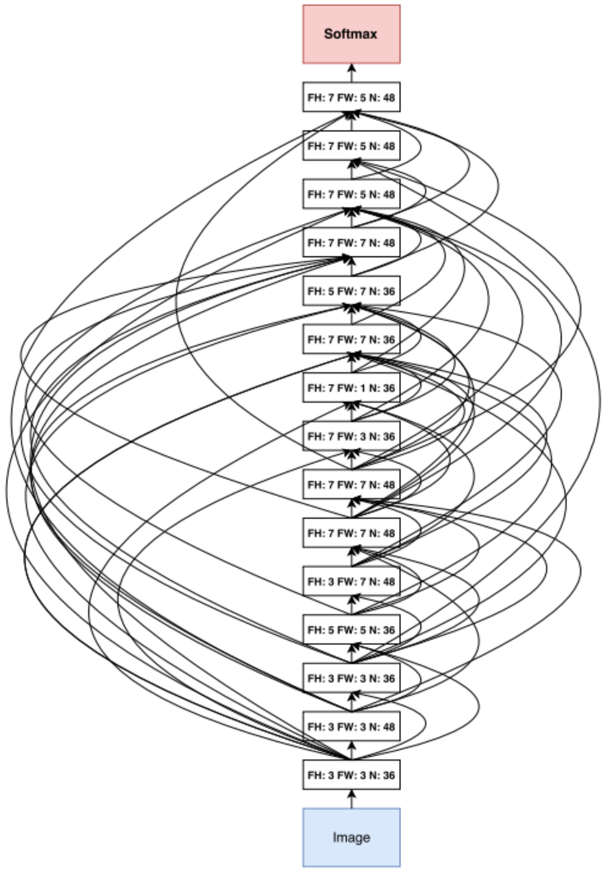
Brown/Penn Treebank tags

Tag	Description	Example	Tag	Description	Example
CC	Coordin. Conjunction	*and, but, or*	SYM	Symbol	+,%, &
CD	Cardinal number	*one, two, three*	TO	"to"	*to*
DT	Determiner	*a, the*	UH	Interjection	*ah, oops*
EX	Existential 'there'	*there*	VB	Verb, base form	*eat*
FW	Foreign word	*mea culpa*	VBD	Verb, past tense	*ate*
IN	Preposition/sub-conj	*of, in, by*	VBG	Verb, gerund	*eating*
JJ	Adjective	*yellow*	VBN	Verb, past participle	*eaten*
JJR	Adj., comparative	*bigger*	VBP	Verb, non-3sg pres	*eat*
JJS	Adj., superlative	*wildest*	VBZ	Verb, 3sg pres	*eats*
LS	List item marker	*1, 2, One*	WDT	Wh-determiner	*which, that*
MD	Modal	*can, should*	WP	Wh-pronoun	*what, who*
NN	Noun, sing. or mass	*llama*	WP$	Possessive wh-	*whose*
NNS	Noun, plural	*llamas*	WRB	Wh-adverb	*how, where*
NNP	Proper noun, singular	*IBM*	$	Dollar sign	$
NNPS	Proper noun, plural	*Carolinas*	#	Pound sign	#
PDT	Predeterminer	*all, both*	"	Left quote	(' or ")
POS	Possessive ending	*'s*	"	Right quote	(' or ")
PP	Personal pronoun	*I, you, he*	(	Left parenthesis	([, (, {, <)
PP$	Possessive pronoun	*your, one's*	)	Right parenthesis	( ],), }, >)
RB	Adverb	*quickly, never*	,	Comma	,
RBR	Adverb, comparative	*faster*	.	Sentence-final punc	(. ! ?)
RBS	Adverb, superlative	*fastest*	:	Mid-sentence punc	(: ; ... – -)
RP	Particle	*up, off*			

Recurrent Neural Architecture Search for Penn Treebank Dataset

Neural Architecture Search with Reinforcement Learning

❖ Experiment Details & Results : CIFAR-10



Model	Depth	Parameters	Error rate (%)
Network in Network (Lin et al., 2013)	-	-	8.81
All-CNN (Springenberg et al., 2014)	-	-	7.25
Deeply Supervised Net (Lee et al., 2015)	-	-	7.97
Highway Network (Srivastava et al., 2015)	-	-	7.72
Scalable Bayesian Optimization (Snoek et al., 2015)	-	-	6.37
FractalNet (Larsson et al., 2016)	21	38.6M	5.22
with Dropout/Drop-path	21	38.6M	4.60
ResNet (He et al., 2016a)	110	1.7M	6.61
ResNet (reported by Huang et al. (2016c))	110	1.7M	6.41
ResNet with Stochastic Depth (Huang et al., 2016c)	110	1.7M	5.23
	1202	10.2M	4.91
Wide ResNet (Zagoruyko & Komodakis, 2016)	16	11.0M	4.81
	28	36.5M	4.17
ResNet (pre-activation) (He et al., 2016b)	164	1.7M	5.46
	1001	10.2M	4.62
DenseNet ($L = 40, k = 12$) Huang et al. (2016a)	40	1.0M	5.24
DenseNet($L = 100, k = 12$) Huang et al. (2016a)	100	7.0M	4.10
DenseNet ($L = 100, k = 24$) Huang et al. (2016a)	100	27.2M	3.74
DenseNet-BC ($L = 100, k = 40$) Huang et al. (2016b)	190	25.6M	3.46
Neural Architecture Search v1 no stride or pooling	15	4.2M	5.50
Neural Architecture Search v2 predicting strides	20	2.5M	6.01
Neural Architecture Search v3 max pooling	39	7.1M	4.47
Neural Architecture Search v3 max pooling + more filters	39	37.4M	3.65

Almost State-of-the-art and smaller and 1.05x faster!

Neural Architecture Search with Reinforcement Learning

❖ Experiment Details & Results : Penn Treebank

Model	Parameters	Test Perplexity
Mikolov & Zweig (2012) - KN-5	2M‡	141.2
Mikolov & Zweig (2012) - KN5 + cache	2M‡	125.7
Mikolov & Zweig (2012) - RNN	6M‡	124.7
Mikolov & Zweig (2012) - RNN-LDA	7M‡	113.7
Mikolov & Zweig (2012) - RNN-LDA + KN-5 + cache	9M‡	92.0
Pascanu et al. (2013) - Deep RNN	6M	107.5
Cheng et al. (2014) - Sum-Prod Net	5M‡	100.0
Zaremba et al. (2014) - LSTM (medium)	20M	82.7
Zaremba et al. (2014) - LSTM (large)	66M	78.4
Gal (2015) - Variational LSTM (medium, untied)	20M	79.7
Gal (2015) - Variational LSTM (medium, untied, MC)	20M	78.6
Gal (2015) - Variational LSTM (large, untied)	66M	75.2
Gal (2015) - Variational LSTM (large, untied, MC)	66M	73.4
Kim et al. (2015) - CharCNN	19M	78.9
Press & Wolf (2016) - Variational LSTM, shared embeddings	51M	73.2
Merity et al. (2016) - Zoneout + Variational LSTM (medium)	20M	80.6
Merity et al. (2016) - Pointer Sentinel-LSTM (medium)	21M	70.9
Inan et al. (2016) - VD-LSTM + REAL (large)	51M	68.5
Zilly et al. (2016) - Variational RHN, shared embeddings	24M	66.0
Neural Architecture Search with base 8	32M	67.9
Neural Architecture Search with base 8 and shared embeddings	25M	64.0
Neural Architecture Search with base 8 and shared embeddings	54M	62.4

What is the perplexity?

- Measure how well language modeling works
- A measure of how well the probability model predicts sample words

$$e^{loss} = e^{-\frac{1}{N}\sum_{i=1}^{N} ln p_{target_i}}$$

$$loss = -\frac{1}{N}\sum_{i=1}^{N} ln p_{target_i}$$

State-of-the-art!

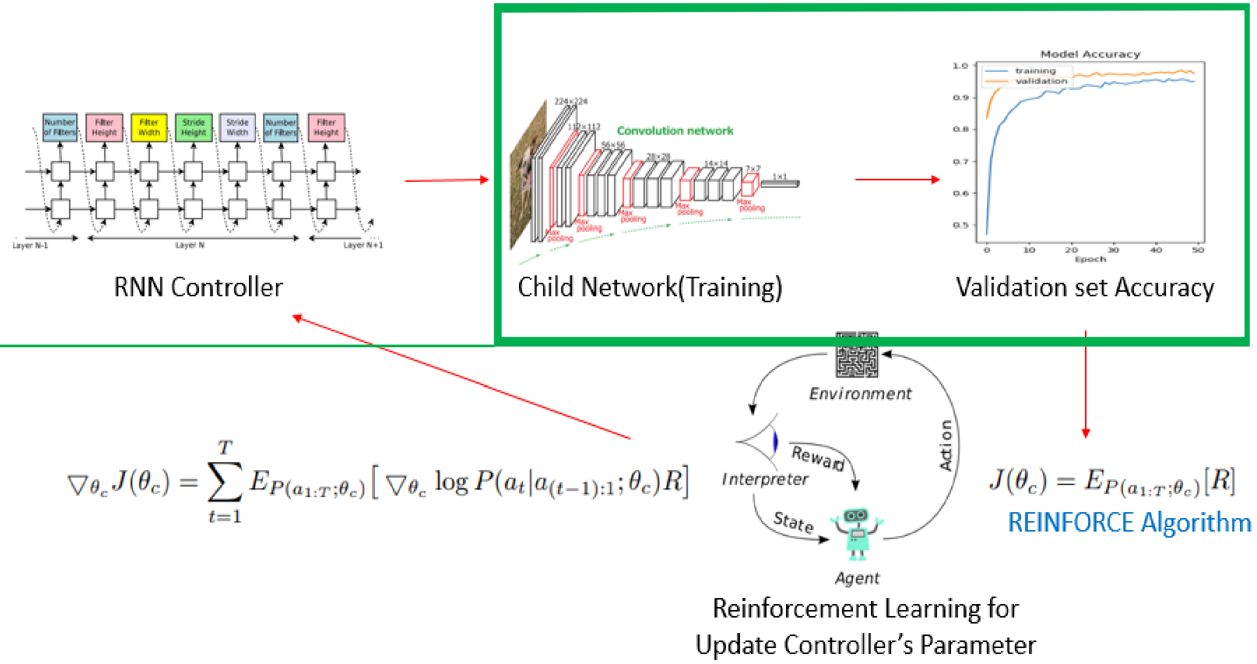
When the cell is transferred to the character language modeling, also achieved a state-of-the-art, perplexity of 1.214

Efficient Neural Architecture Search via Parameter Sharing

❖ What are the shortcomings of Neural Architecture Search?

- NAS used 800 GPUs for 28days and NAS-Net used 450 GPUs for 3-4days (i.e. 32,400-43,000 hours)

- Where is the most severe bottleneck?
  - The Child networks measure the accuracy and then all learned weights are discarded
  - So if the RNN Controller outputs the same hyper-parameters, there is a problem that needs to be learned again



Efficient Neural Architecture Search via Parameter Sharing

❖ ENAS Method : Directed Acyclic Graph(DAG)

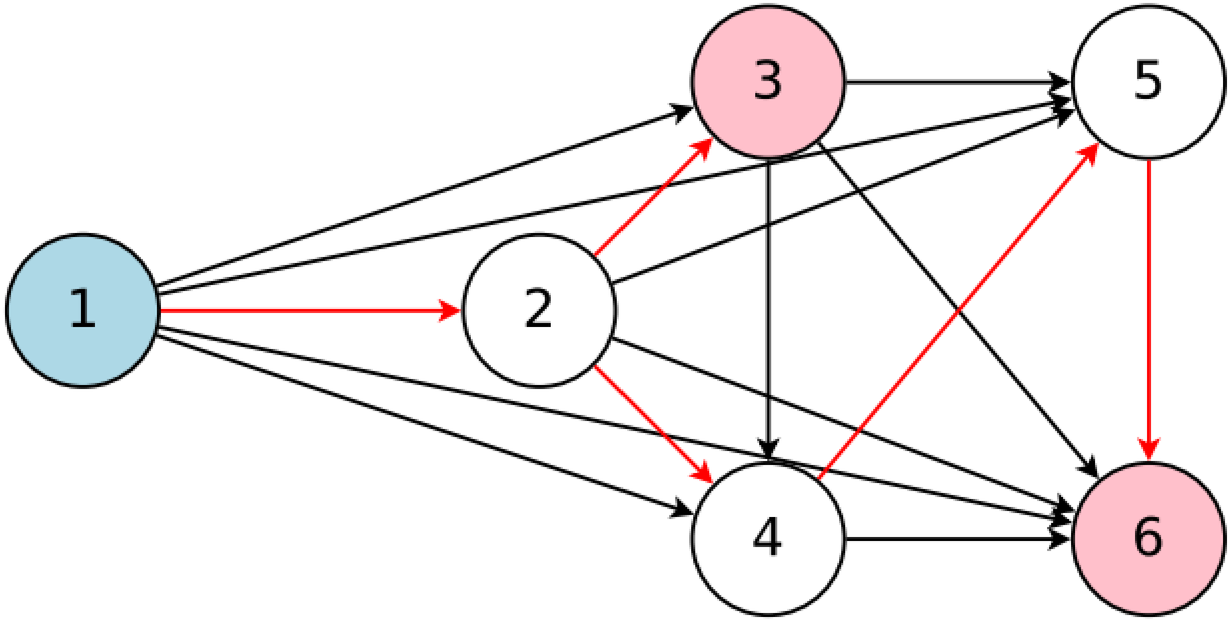


Figure 2. The graph represents the entire search space while the red arrows define a model in the search space, which is decided by a controller. Here, node 1 is the input to the model whereas nodes 3 and 6 are the model's outputs.

Node
Local computations
(Activation function etc.)

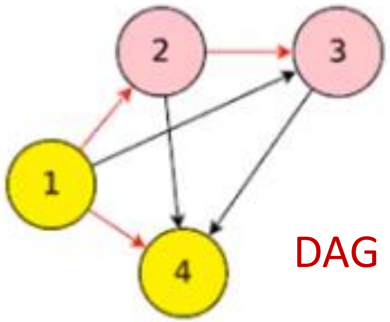
Edge
Flow of information between N nodes

The nodes share information about the weights and reuse the previously learned information when the same node is selected

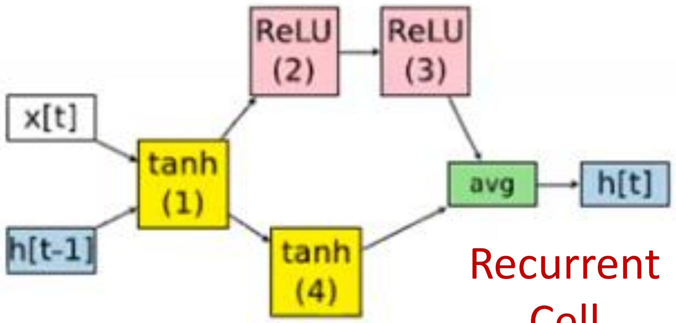
Efficient Neural Architecture Search via Parameter Sharing

Efficient Neural Architecture Search via Parameter Sharing

❖ ENAS Method : Designing Recurrent Cells

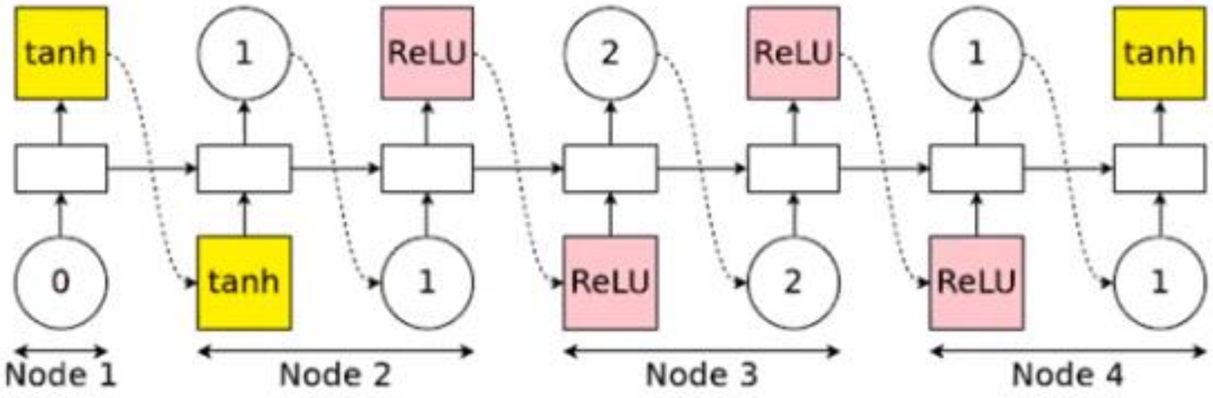


DAG



Recurrent Cell

Controller RNN : The outputs result in DAG and recurrent cell



Simple example : N = 4 (Activation function)

1. At node 1

   Controller samples $tanh$

   -> $h_1 = \tanh(x_t \cdot W^{(x)} + h_{t-1} \cdot {W_1}^{(h)})$
2. At node 2

   Choose previous index 1 and activation function is $ReLU$

   -> $h_2 = ReLU(h_1 \cdot {W_{2,1}}^{(h)})$
3. At node 3

   Choose previous index 2 and activation function is $ReLU$

   -> $h_3 = ReLU(h_2 \cdot {W_{3,2}}^{(h)})$
4. At node 4

   Choose previous index 1 and activation function is $tanh$

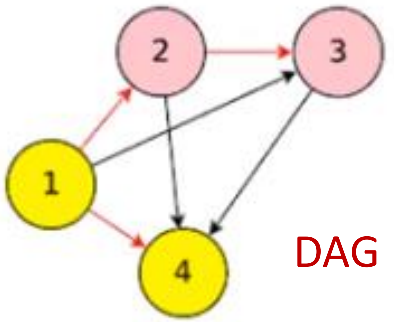
   -> $h_3 = tanh(h_1 \cdot {W_{4,1}}^{(h)})$
5. Output

   Since the indices 3 and 4 were not sampled, the recurrent uses their average

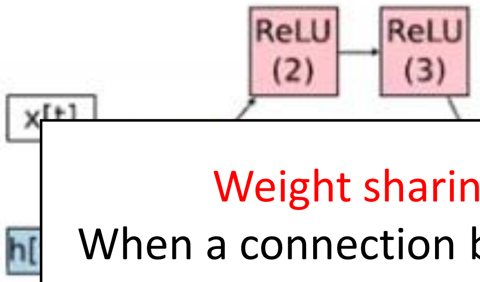
   -> $h_t = (h_3 + h_3)/2$

Efficient Neural Architecture Search via Parameter Sharing

❖ ENAS Method : Designing Recurrent Cells



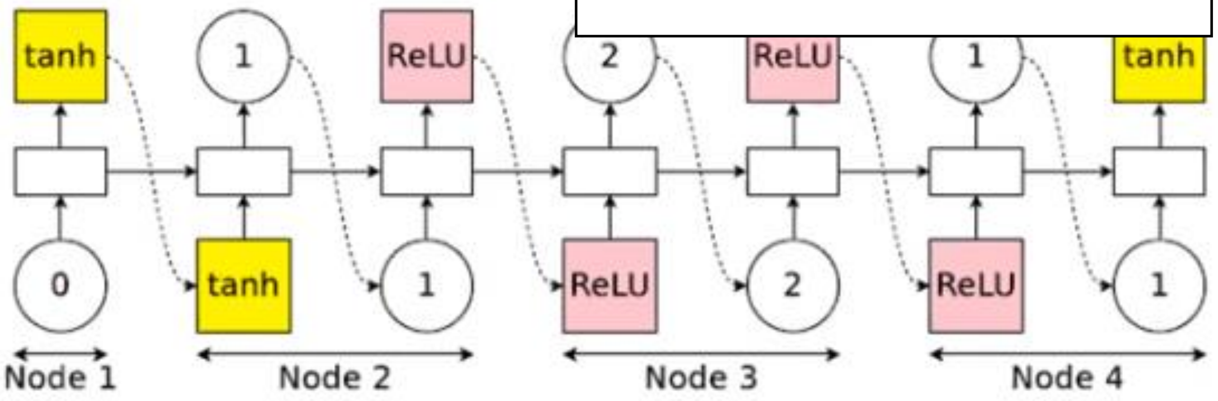
DAG



Weight sharing

When a connection between 2 and 1 occurs, that is, when the edge is activated, the weight will be reused

Controller RNN : The outputs



Simple example : N = 4 (Activation function)

1. At node 1

   Controller samples $tanh$

   -> $h_1 = \tanh(x_t \cdot W^{(x)} + h_{t-1} \cdot W_1{}^{(h)})$

2. At node 2

   Choose previous index 1 and activation function is $ReLU$

   -> $h_2 = ReLU(h_1 \cdot W_{2,1}{}^{(h)})$

3. At node 3

   Choose previous index 2 and activation function is $ReLU$

   -> $h_3 = ReLU(h_2 \cdot W_{3,2}{}^{(h)})$

4. At node 4

   Choose previous index 1 and activation function is $tanh$

   -> $h_3 = tanh(h_1 \cdot W_{4,1}{}^{(h)})$

5. Output

   Since the indices 3 and 4 were not sampled, the recurrent uses their average

   -> $h_t = (h_3 + h_3)/2$

Efficient Neural Architecture Search via Parameter Sharing

❖ ENAS Experiment Result

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**

CIFAR-10

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**

Penn Treebank

Using Single GTX 1080Ti GPU, the search for architectures takes less than 16hours (1000x less expensive than NAS)


THANK YOU