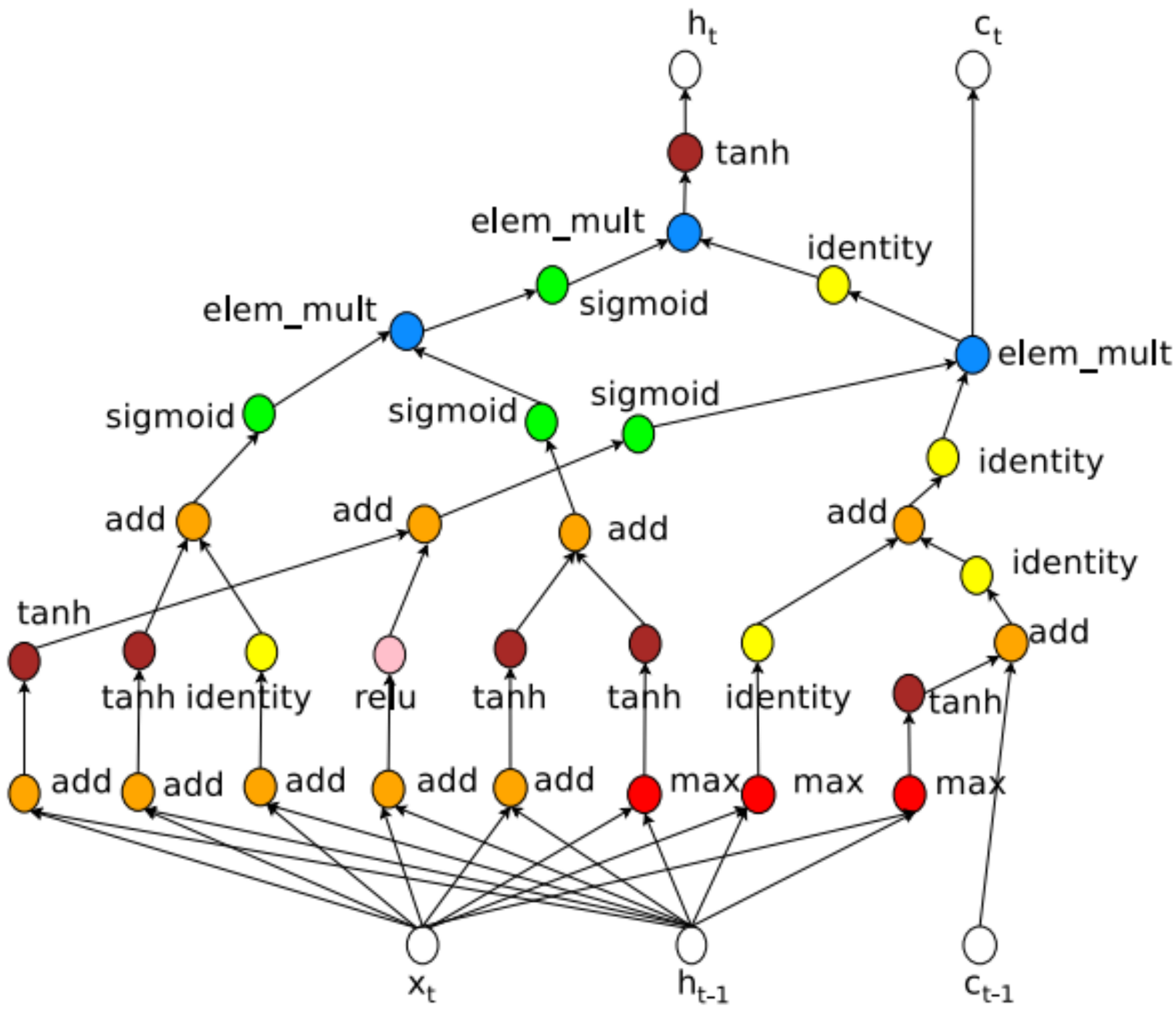


Presentation on “Neural Architecture Search with Reinforcement Learning”

Kiho Suh
Modulabs(모두의연구소), June 12th 2017

About Paper

- 2016년 11월 5일에 v1
- 현재는 v2
- Google Brain
- Barret Zoph, Quoc V. Le
- ICLR 2017 Oral Presentation으로 발표

Under review as a conference paper at ICLR 2017

NEURAL ARCHITECTURE SEARCH WITH REINFORCEMENT LEARNING

Barret Zoph, Quoc V. Le
Google Brain
{barretzoph,qvl}@google.com

ABSTRACT

Neural networks are powerful and flexible models that work well for many difficult learning tasks in image, speech and natural language understanding. Despite their success, neural networks are still hard to design. In this paper, we use a recurrent network to generate the model descriptions of neural networks and train this RNN with reinforcement learning to maximize the expected accuracy of the generated architectures on a validation set. On the CIFAR-10 dataset, our method, starting from scratch, can design a novel network architecture that rivals the best human-invented architecture in terms of test set accuracy. Our CIFAR-10 model achieves a test error rate of 3.65, which is 0.09 percent better and 1.05x faster than the previous state-of-the-art model that used a similar architectural scheme. On the Penn Treebank dataset, our model can compose a novel recurrent cell that outperforms the widely-used LSTM cell, and other state-of-the-art baselines. Our cell achieves a test set perplexity of 62.4 on the Penn Treebank, which is 3.6 perplexity better than the previous state-of-the-art model. The cell can also be transferred to the character language modeling task on PTB and achieves a state-of-the-art perplexity of 1.214.

1 INTRODUCTION

The last few years have seen much success of deep neural networks in many challenging applications, such as speech recognition (Hinton et al., 2012), image recognition (LeCun et al., 1998; Krizhevsky et al., 2012) and machine translation (Sutskever et al., 2014; Bahdanau et al., 2015; Wu et al., 2016). Along with this success is a paradigm shift from feature designing to architecture designing, i.e., from SIFT (Lowe, 1999), and HOG (Dalal & Triggs, 2005), to AlexNet (Krizhevsky et al., 2012), VGGNet (Simonyan & Zisserman, 2014), GoogleNet (Szegedy et al., 2015), and ResNet (He et al., 2016a). Although it has become easier, designing architectures still requires a lot of expert knowledge and takes ample time.

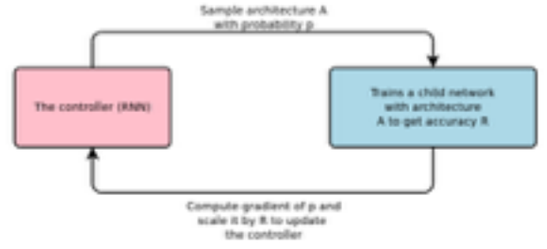
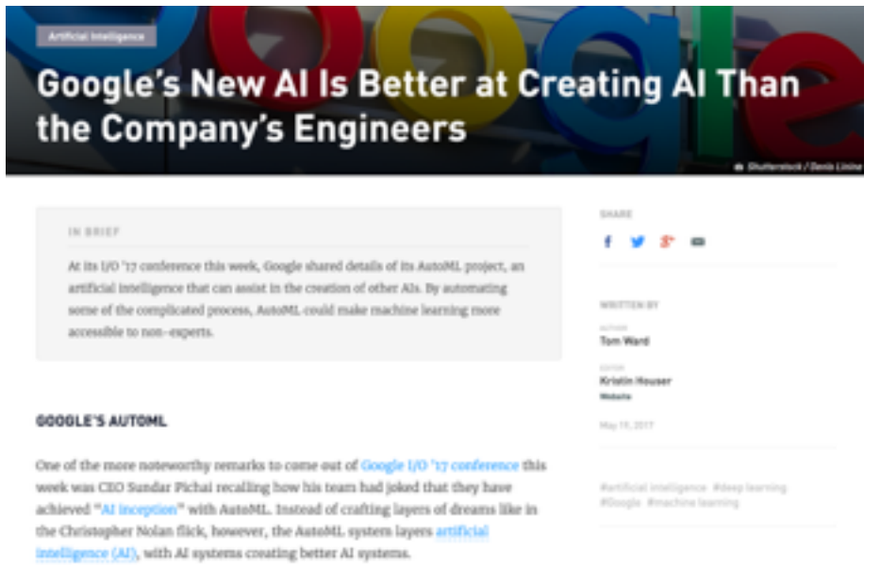


Figure 1: An overview of Neural Architecture Search.

Google’s AutoML



https://futurism.com/googles-new-ai-is-better-at-creating-ai-than-the-companys-engineers/

http://techm.kr/bbs/?t=154

- 머신러닝 개발 업무중 일부를 자동화하는 프로젝트
- AI 전문가의 미래는?…
- AutoML이 뭘하려는지 잘 보여주는 논문

Motivation for Architecture Search

- 뉴럴 네트워크를 디자인하는것은 힘들고 튜닝하는데 꽤 많은 노력이 필요하다. 잘 한다고 하시는 분들도 꽤 시간이 많이 걸린다.

- 디자인 하는데 “딱 이 방법이다” 하는게 없다.

- 이제는 Feature Engineering에서 Neural Network Engineering으로 페러다임이 변화되었다.

- 좋은 구조를 자동으로 학습할수 있을까?

- **딥러닝 구조를 만드는 딥러닝 구조**

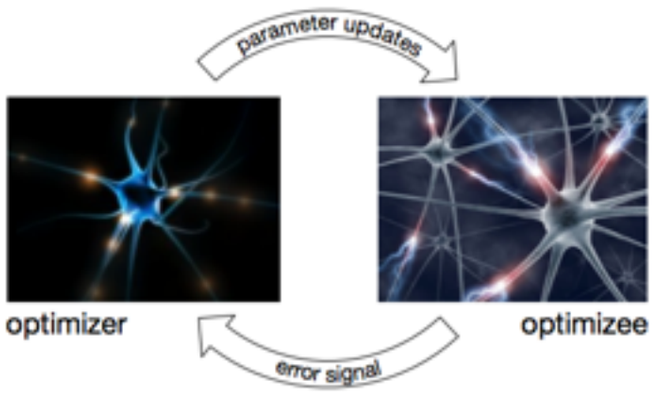
- 이 분야에서의 첫 시도

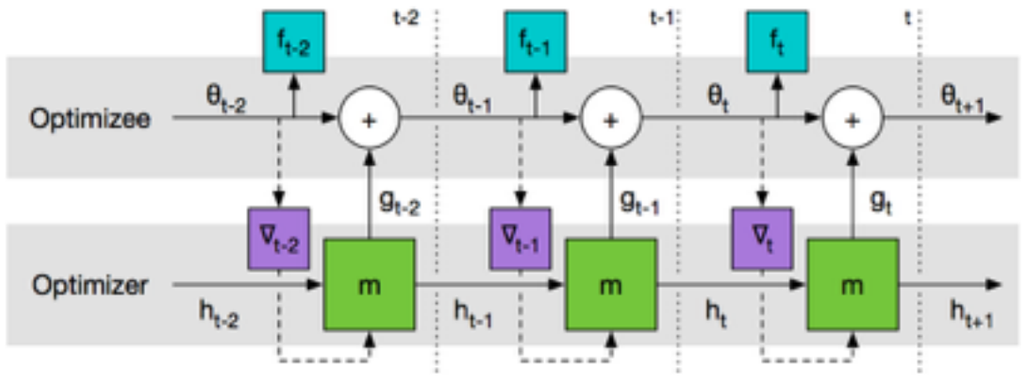
slide modified from Zoph, Le

Related Work

- Hyperparameter optimization
- Modern neuro-evolution algorithms
- Neural Architecture Search

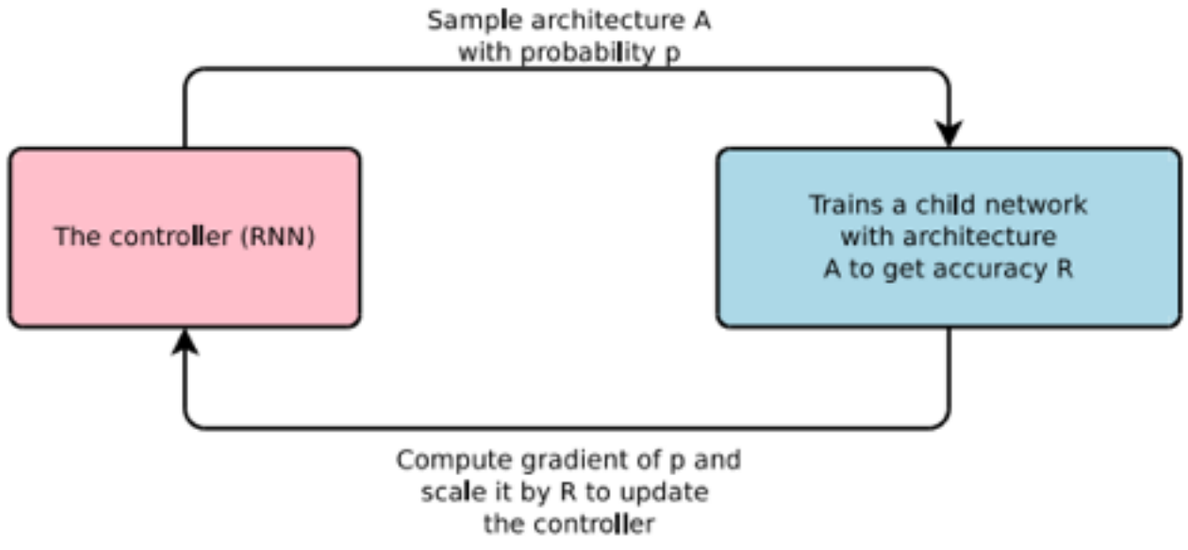
- Learning to learn or Meta-Learning





Learning to learn by gradient descent by gradient descent, Andrychowicz et al. 2016

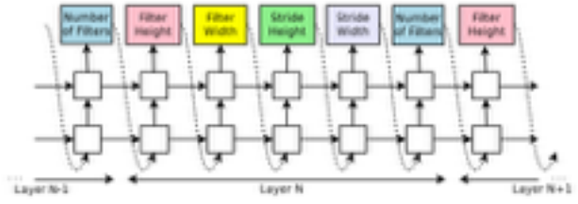
Neural Architecture Search



- 핵심은 Configuration string으로 뉴럴 네트워크의 구조와 연결을 명시할수 있다. (Caffe에서는 이렇게 함)
    - 한 layer의 Configuration: [“Filter Width: 5”, “Filter Height: 3”, “Num Filters: 24”]
- 여기에서 아이디어는 RNN (“Controller”)를 사용해서 Neural Network Architecture를 명시하는 이 string을 생성한다.
- RNN을 수렴하게 학습한다. 수렴할때 이 만들어진 string의 정확도를 알수 있다.
- Validation set에서 얼마나 잘하는지 알기위해서 만들어진 구조(“Child Network”)를 학습한다.
- Child model의 정확도를 기반으로한 Controller model의 파라미터들을 강화학습을 써서 업데이트한다.
- 강화학습이기때문에 정확도는 Reward signal로 쓰인다.

slide modified from Zoph, Le

Neural Architecture Search



1. Controller RNN은 뉴럴 네트워크의 구조적 hyperparameter들을 생성한다. Layer의 숫자가 어떤 값을 넘으면 구조를 생성하는것을 멈춘다. 이 값은 training이 진행될수록 높아지는 스케줄에 의해 정해진다.

2. Controller RNN에서 하나의 구조를 생성하면 이 구조를 가진 뉴럴 네트워크가 만들어지고 train한다.

3. 수렴할때 네트워크의 validation set에 대한 정확도가 기록된다.

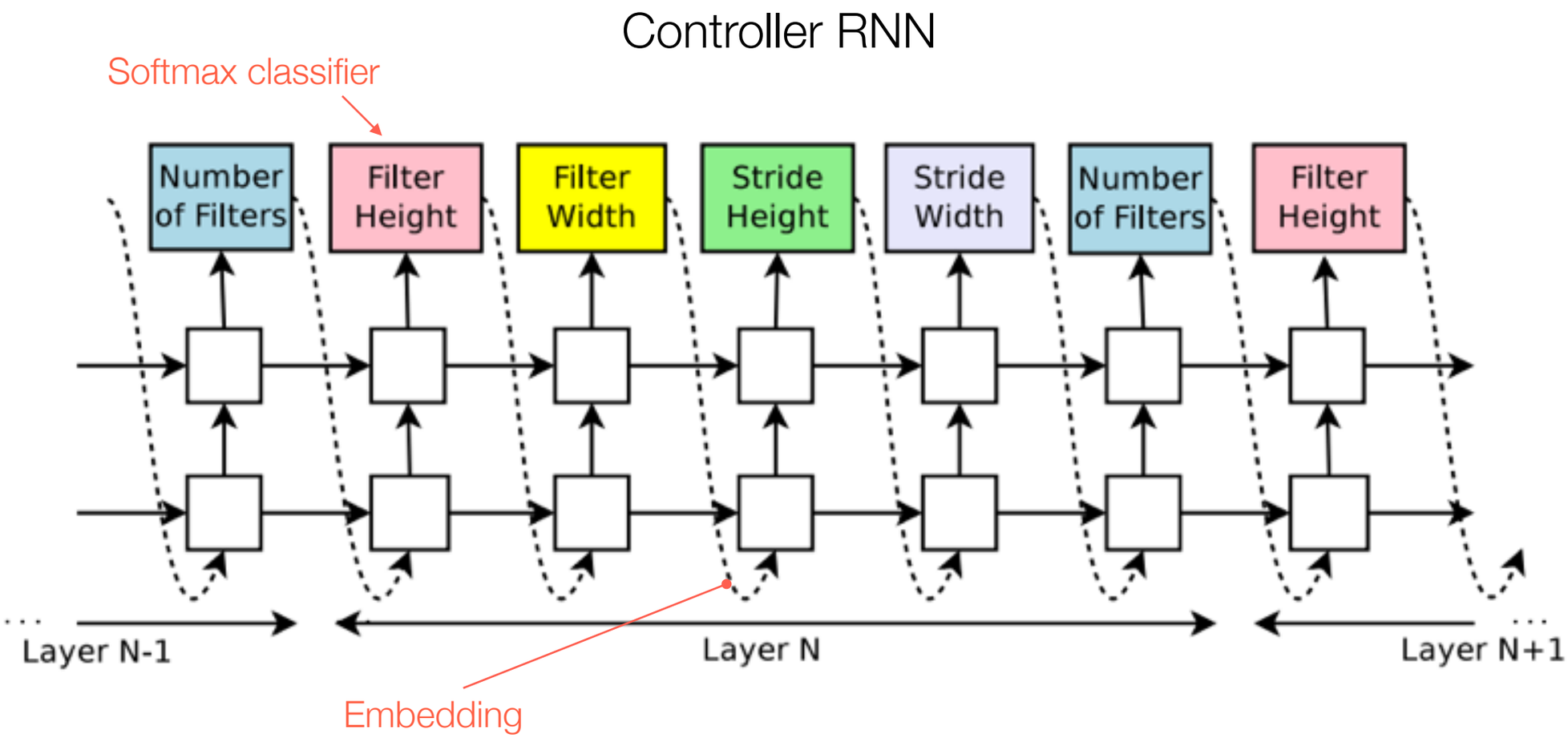
4. 제안된 구조의 Expected Validation Accuracy를 최대화하기 위해서 Controller RNN의 파라미터들인 θ_c 가 최적화된다.

$$\theta_c$$

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

5. Policy gradient를 써서 θ_c를 업데이트 한다.

Neural Architecture Search for Convolutional Networks



slide from Zoph, Le

Training with REINFORCE

Controller RNN의 파라미터들

생성된 구조의 정확도, Reward signal

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

Architecture predicted by the controller RNN viewed as a sequence of actions.

Controller가 새로운 뉴럴네트워크 구조를 디자인하기 위한 예상해야되는 hyperparameter들의 숫자

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

Standard REINFORCE Update Rule

- REINFORCE를 사용한 이유는 가장 간단하고 Q-learning을 포함한 다른 방법들에 비해서 튜닝하기 쉽다. 스케일 엄청 크기 때문에 튜닝을 많이 하려고 하면 힘들다.
- Layer 하나짜리 CNN에서 T는 3이다. a_1은 filter height, a_2는 filter width, a_3은 number of filters이다.
- Reward signal R이 non-differentiable해서 policy gradient를 써서 θ_c 업데이트해야 한다.

Training with REINFORCE

Controller가 새로운 뉴럴네트워크 구조를 디자인하기 위한 예상해야되는 hyperparameter들의 숫자

Controller RNN의 파라미터들

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

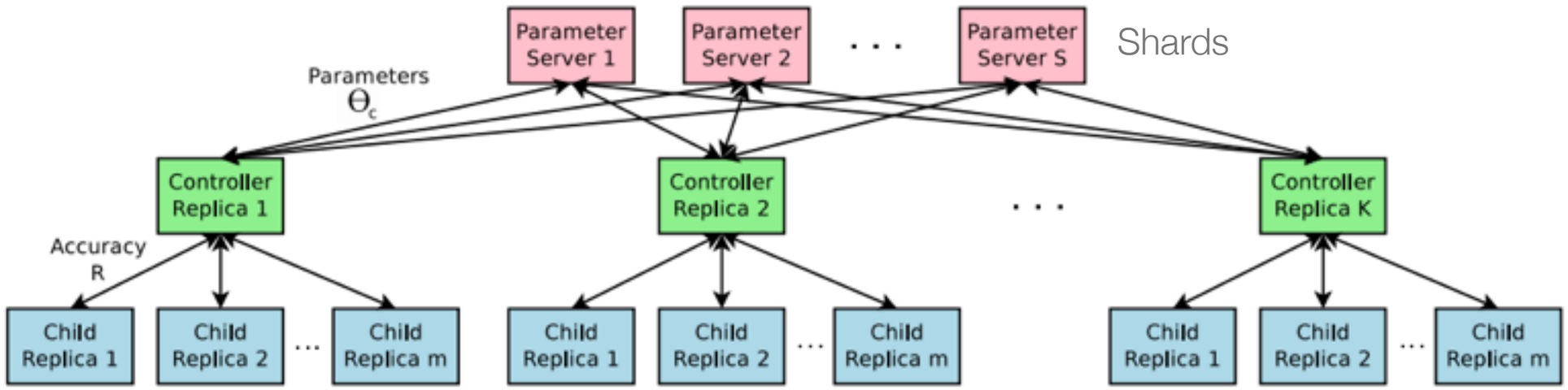
mini batch 안의 모델들의 갯수

Sample many architectures at one time and average them across mini batch

이 측정의 high variance를 줄이기 위한 baseline (정답값이라고도 볼수 있음)

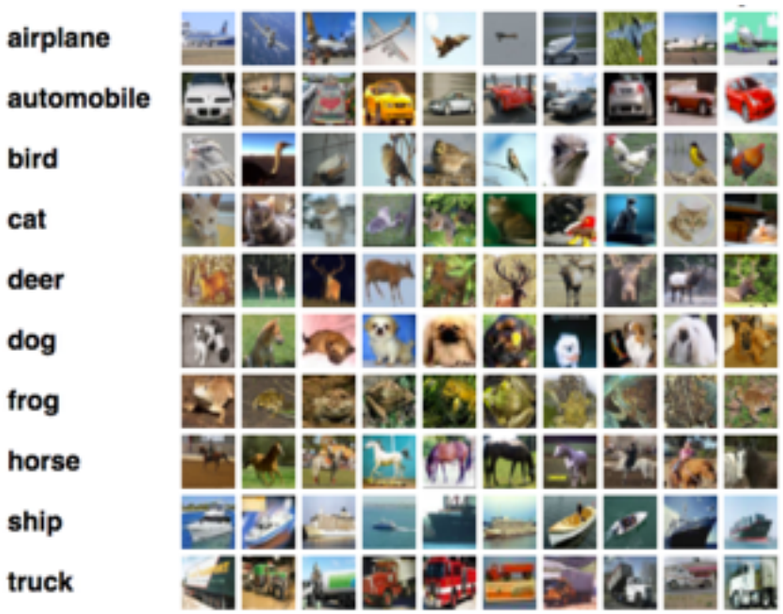
$$\frac{1}{m}\sum_{k=1}^{m}\sum_{t=1}^{T}\nabla_{\theta_c}\log P(a_t|a_{(t-1):1};\theta_c)(R_k-b)$$

Distributed Training



- Controller parameter들은 S parameter server들에 저장한다. Parameter server들에서 parameter들을 K controller replicas 보낸다.
- 각 controller replica는 m architecture들을 sample하고 여러개의 child model들을 병렬로 돌린다.
- 각 child model의 정확도는 parameter server들에 보낼 θ_c에 대한 gradient들을 계산하기 위해 기록한다.
- 10 parameter servers
- 800 GPUs. Accuracy를 train하는데 몇시간이 걸림.
- 13,000~15,000 모델들을 train함. Google에서 2~3주 걸렸다.

Overview of Experiments



https://www.cs.toronto.edu/~kriz/cifar.html

Alphabetical list of part-of-speech tags used in the Penn Treebank Project:

Number	Tag	Description
1.	CC	Coordinating conjunction
2.	CD	Cardinal number
3.	DT	Determiner
4.	EX	Existential *there*
5.	FW	Foreign word
6.	IN	Preposition or subordinating conjunction
7.	JJ	Adjective
8.	JJR	Adjective, comparative
9.	JJS	Adjective, superlative
10.	LS	List item marker
11.	MD	Modal
12.	NN	Noun, singular or mass
13.	NNS	Noun, plural
14.	NNP	Proper noun, singular
15.	NNPS	Proper noun, plural
16.	PDT	Predeterminer
17.	POS	Possessive ending
18.	PRP	Personal pronoun
19.	PRP$	Possessive pronoun
20.	RB	Adverb
21.	RBR	Adverb, comparative
22.	RBS	Adverb, superlative
23.	RP	Particle
24.	SYM	Symbol
25.	TO	*to*
26.	UH	Interjection
27.	VB	Verb, base form
28.	VBD	Verb, past tense
29.	VBG	Verb, gerund or present participle
30.	VBN	Verb, past participle
31.	VBP	Verb, non-3rd person singular present
32.	VBZ	Verb, 3rd person singular present
33.	WDT	Wh-determiner
34.	WP	Wh-pronoun
35.	WP$	Possessive wh-pronoun
36.	WRB	Wh-adverb

https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

- 딥러닝에서 가장 많이 쓰이는 데이터셋인 CIFAR-10와 Penn Treebank에 적용.
- CIFAR-10에 CNN을 생성하고 Penn Treebank에 RNN cell을 생성한다.
- Penn Treebank에서는 State of the Art이고 CIFAR-10에서는 거의 State of the Art 이면서 더 작고 더 빠른 네트워크
- Penn Treebank에서 나온 cell이 LSTM보다 다른 language modeling datasets과 번역에서 더 좋은 성과를 보였다.

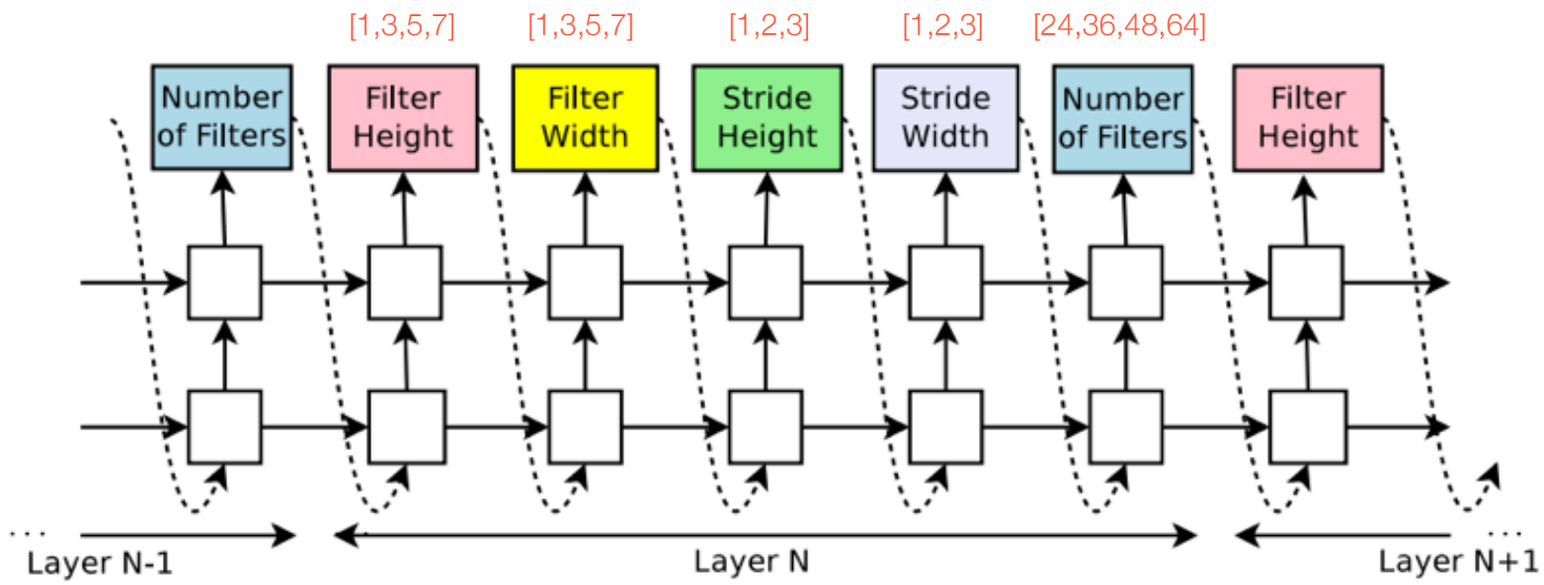
Neural Architecture Search for CIFAR-10

- CIFAR-10에서 convolutional network를 예측하기 위해서 Neural Architecture Search 적용한다.

- 고정된 layer들의 숫자 (15, 20, 13) 를 위해 다음을 예상한다:
  - Filter width/height
  - Stride width/height
  - Number of filters

slide modified from Zoph, Le

Neural Architecture Search for CIFAR-10



- Skip Connection이 없다 (그래서 한계가 있음)
- Filter Height나 Width에 정할수 있는 값들인 [1,3,5,7] 에서 softmax가 정한다. Continuous가 아니기 때문에, 리스트에서 선택한다.
- One layer = 128 unit RNN (pretty small)
- 현재 모델들에서 Skip Connection이 많이 사용되고 있음 (e.g. ResNet, DenseNet)

CIFAR-10 Prediction Method

- Branching과 residual connections를 포함하기 위해 탐색 범위를 넓힌다.

- 탐색 범위를 넓히기 위해 Skip Connection의 예측을 제안한다.

- Layer N에서 어떤 layer들이 layer N에 input으로 넣어야될지 N-1 sigmoid까지 모은다.

- 만약 layer가 안 모이면, 이미지들의 minibatch 넣는다.

- 마지막 layer에서 연결되지않은 모든 layer 출력들을 다 concatenate 한다.

Skip Connection in ResNet

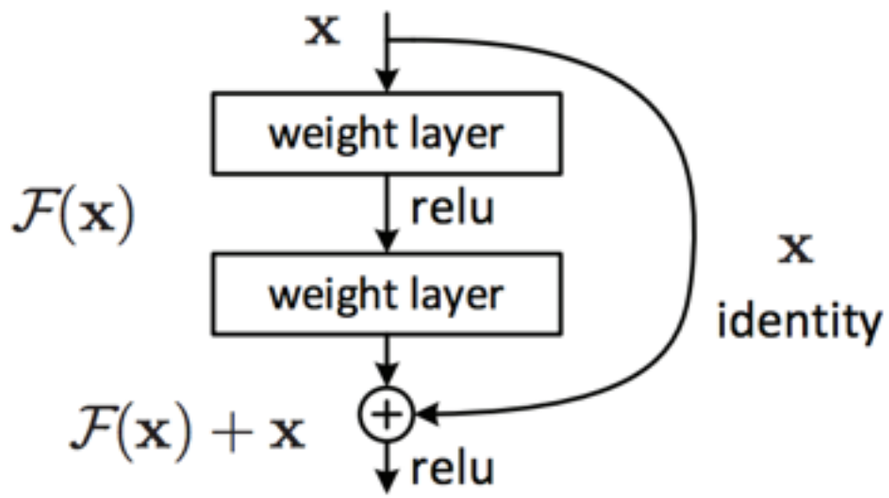


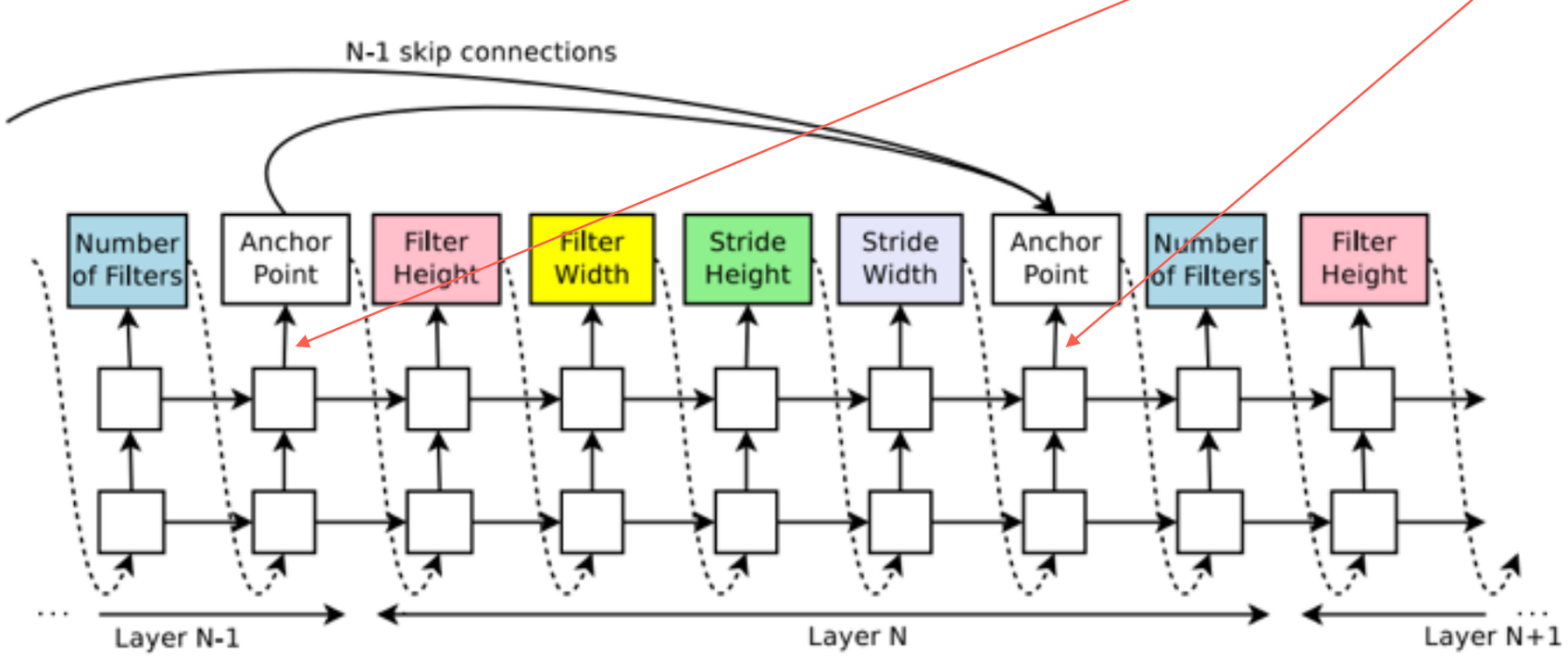
Figure 2. Residual learning: a building block.

The formulation of **F(x) +x** can be realized by feedforward neural networks with “shortcut connections” (Fig. 2). Shortcut connections [2, 34, 49] are those skipping one or more layers. In our case, **the shortcut connections simply perform identity mapping, and their outputs are added to the outputs of the stacked layers** (Fig. 2). Identity shortcut connections add neither extra parameter nor computational complexity.

Neural Architecture Search for CIFAR-10

Weight Matrices

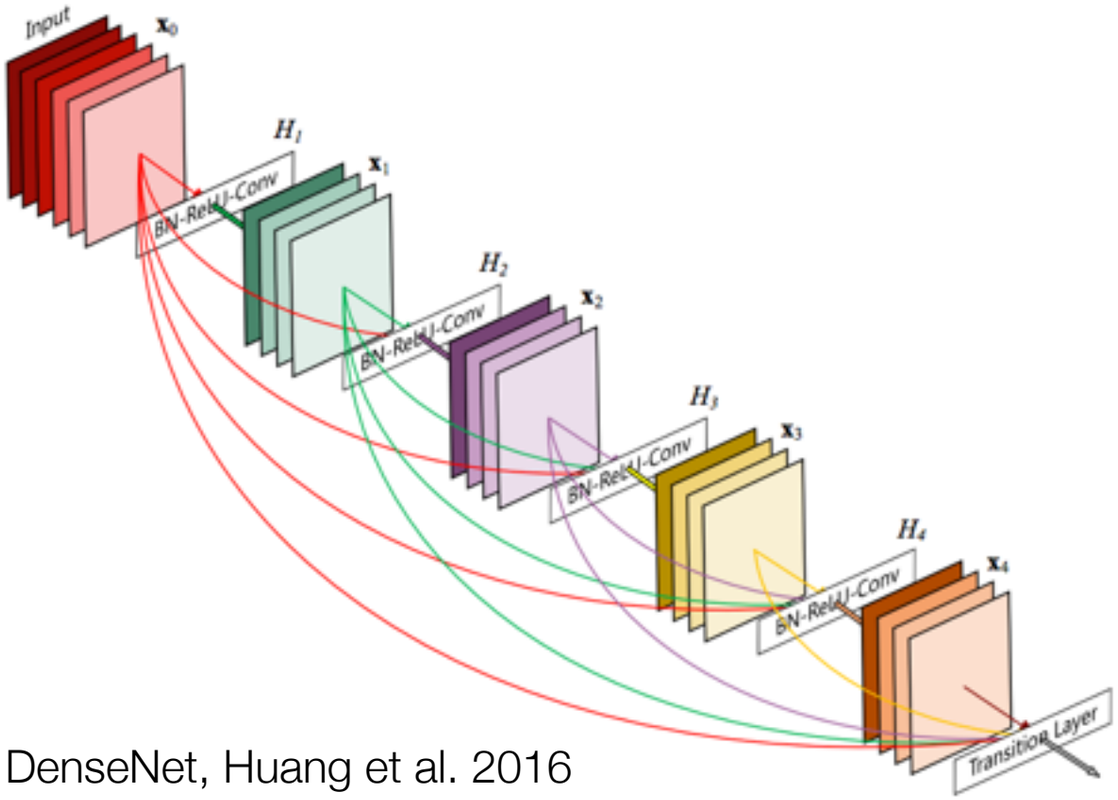
$$\mathrm{P}(\text{Layer j is an input to layer i}) = \mathrm{sigmoid}(v^{\mathrm{T}}\tanh(W_{prev} * h_j + W_{curr} * h_i))$$



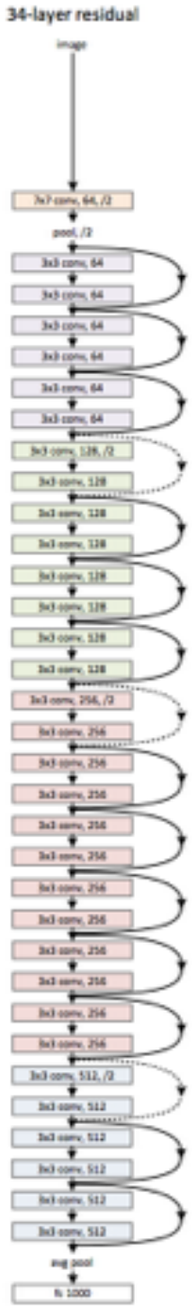
CIFAR-10 Experiment Details

- 동시에 100 Controller Replica들을 training 하는 8개의 child network를 썼다.
- 800 GPUs를 한번에 동시에 썼다.
- Reward given to the Controller is the maximum validation accuracy of the last 5 epochs squared
- 50,000개의 Training 예에서 45,000는 training으로 5,000는 validation으로 썼다.
- 각 child model은 50 epoch 동안 train했다. 반나절 걸림.
- 12,800개의 child model들을 돌렸다.
- Controller를 위해 curriculum training을 사용해서 layer 수를 늘려나갔다.
- 탐색 공간이 엄청 크다: 10^{30}~10^{40}

DenseNet and ResNet



DenseNet, Huang et al. 2016



ResNet, He et al. 2015

DenseNets connects each layer to every other layer in a feed-forward fashion. They alleviate the vanishing-gradient problem, strengthen feature propagation, encourage feature reuse, and substantially reduce the number of parameters.

Generated Convolutional Network from Neural Architecture Search

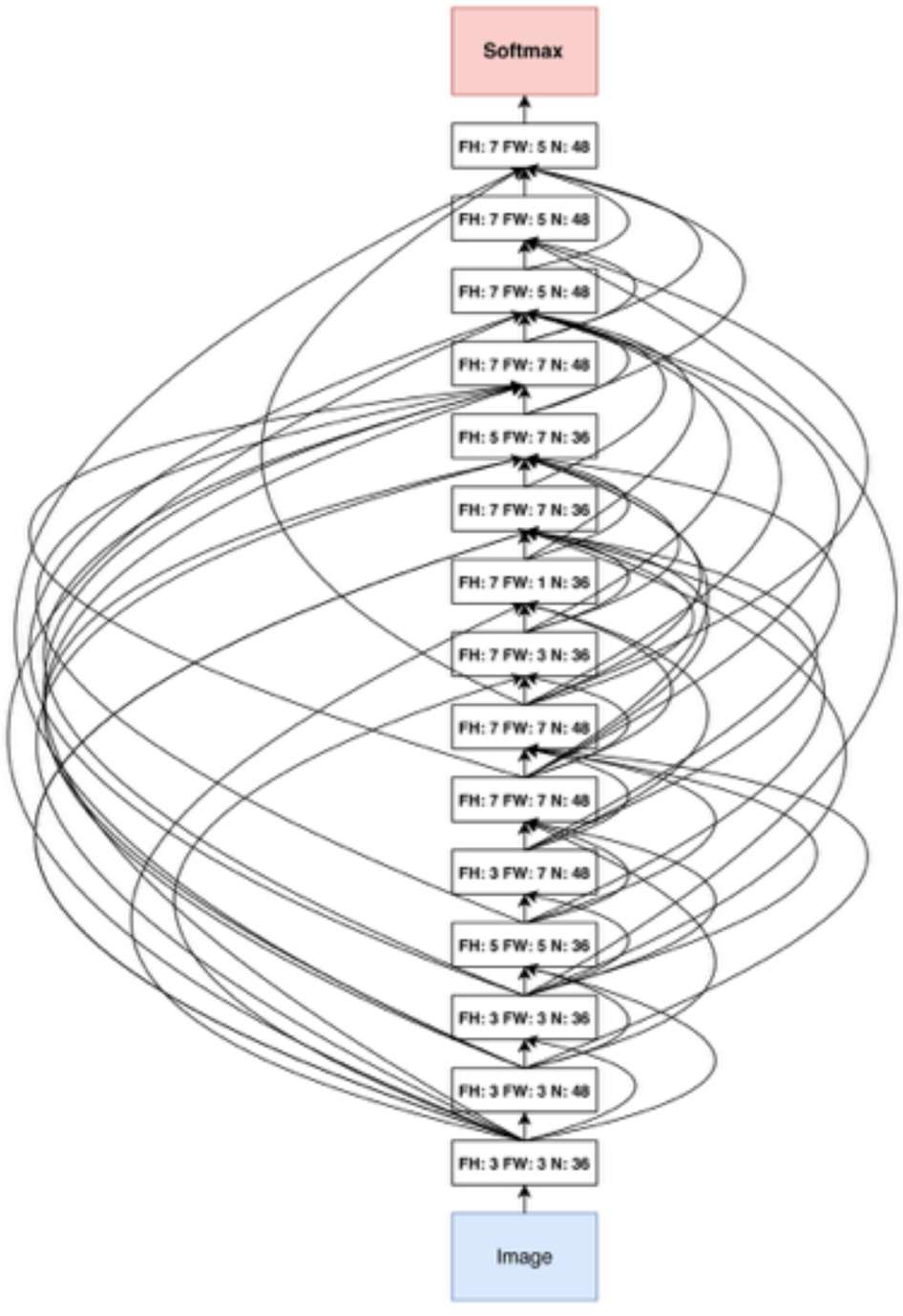
Model	Depth	Parameters	Error rate (%)
Network in Network (Lin et al., 2013)	-	-	8.81
All-CNN (Springenberg et al., 2014)	-	-	7.25
Deeply Supervised Net (Lee et al., 2015)	-	-	7.97
Highway Network (Srivastava et al., 2015)	-	-	7.72
Scalable Bayesian Optimization (Snoek et al., 2015)	-	-	6.37
FractalNet (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

5% faster

Best result of evolution (Real et al. 2017): 5.4%
Best result of Q-learning (Baker et al. 2017): 6.92%

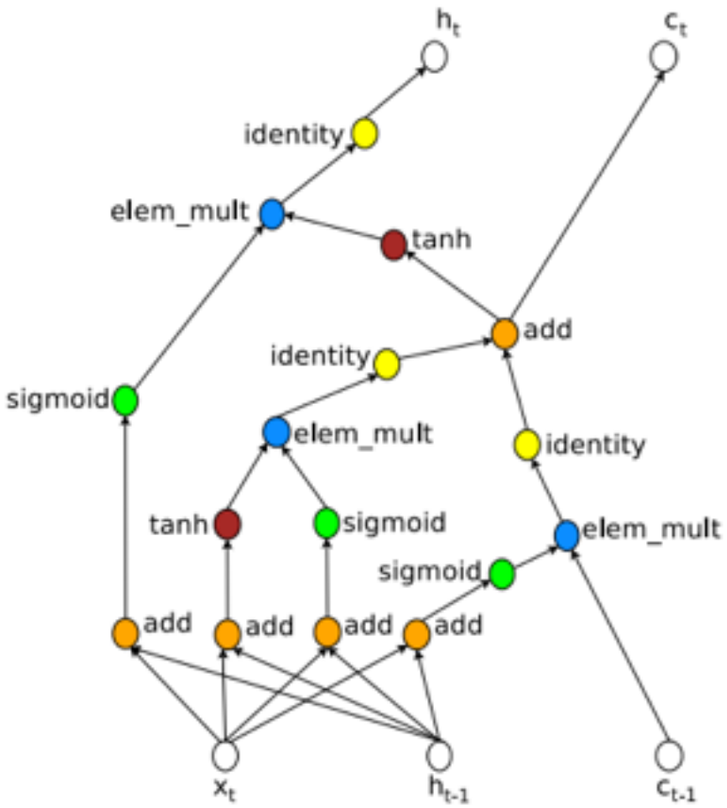
slide modified from Zoph, Le

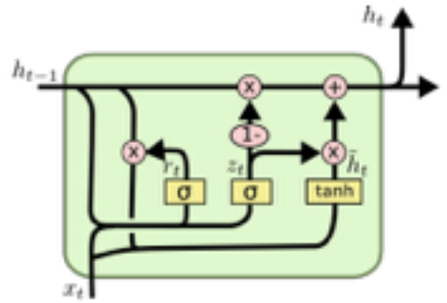
Generated Convolutional Network from Neural Architecture Search



- Skip Connection을 너무 좋아함
- 직사각형 필터를 좋아함 (e.g. 7 x 4 filter)
- 첫번째 convolution 이후에 모든 layer와 연결하는것을 좋아한다.

Recurrent Cell Prediction Method





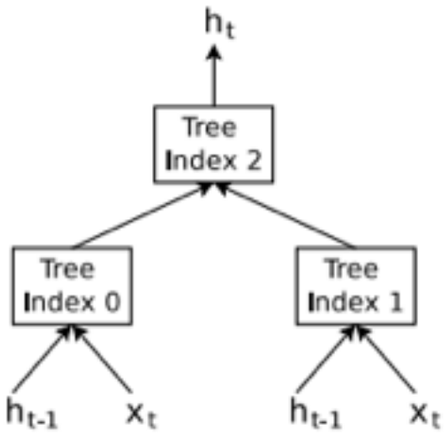
$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

- LSTM이나 GRU와 비슷한 RNN cell들을 찾기 위해서 search space를 만들었다.
- LSTM cell을 참고하고 Search space를 만들었다.

slide modified from Zoph, Le

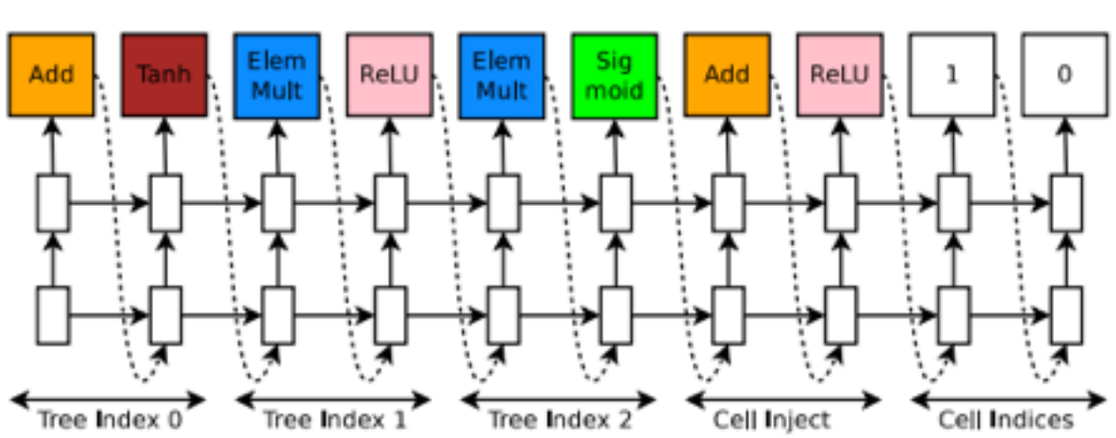
Recurrent Cell Prediction Method

Cell Search Space



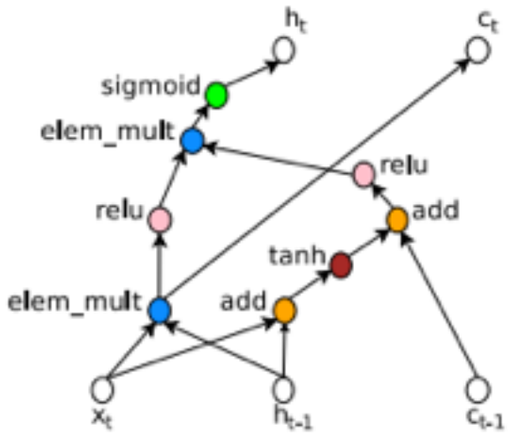
- 그래프의 계산은 트리로 표현함
- 어떤 activation function 이나 어떤 결합 방법을 쓸지 결정
- LSTM에서는 “더하기 함수”
- 트리의 leaf node는 8 (여기서는 2)

Controller RNN



- Controller RNN을 써서 어떤 함수를 결합할지 혹은 어떤 activation function 을 써서 트리를 label 결정
- Controller RNN의 역할은 트리를 보고 어떤 함수를 선택해서 생성할건지 결정

Created New Cell



- 트리를 생성하면 cell의 실제 구현을 한다.

Penn Treebank Experiment Details

- Run Neural Architecture Search with our cell prediction method on the Penn Treebank language modeling dataset

- 1 child network를 training하는 400개의 Controller Replica들을 사용했다.

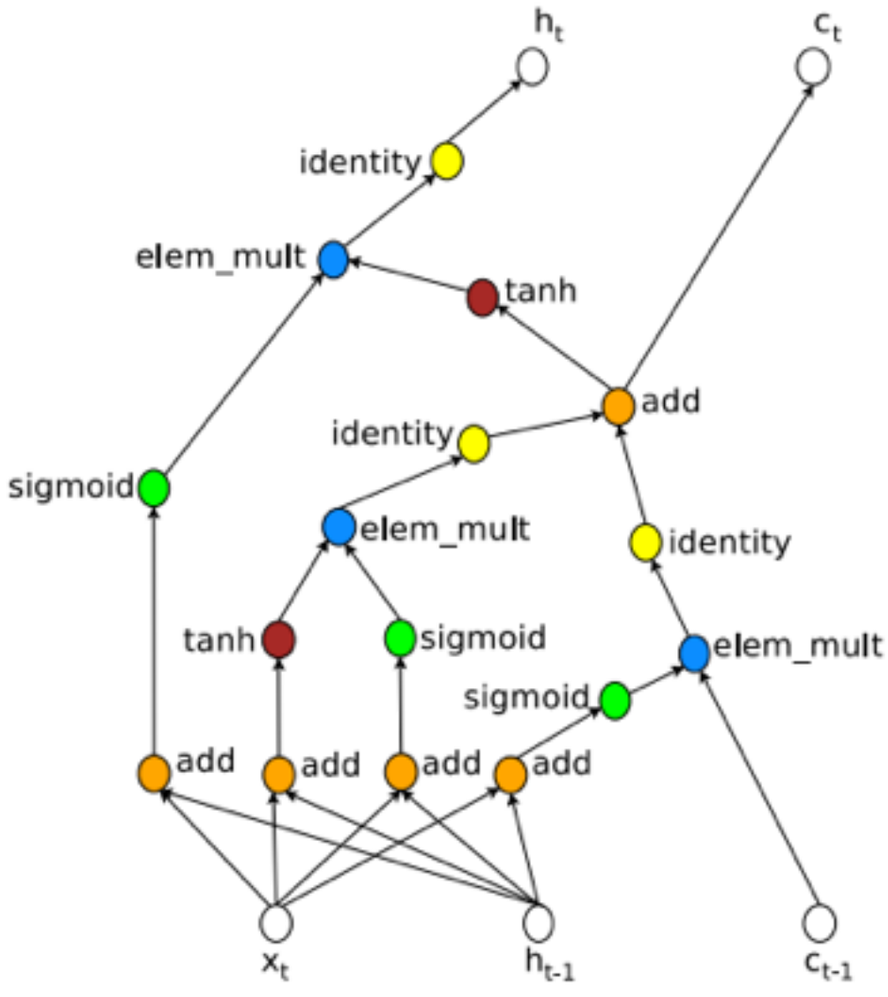
- 동시에 400 CPU들을 한번에 사용했다.

- 총 15,000 child model들을 사용했다. (실제 search space는 10^{18})

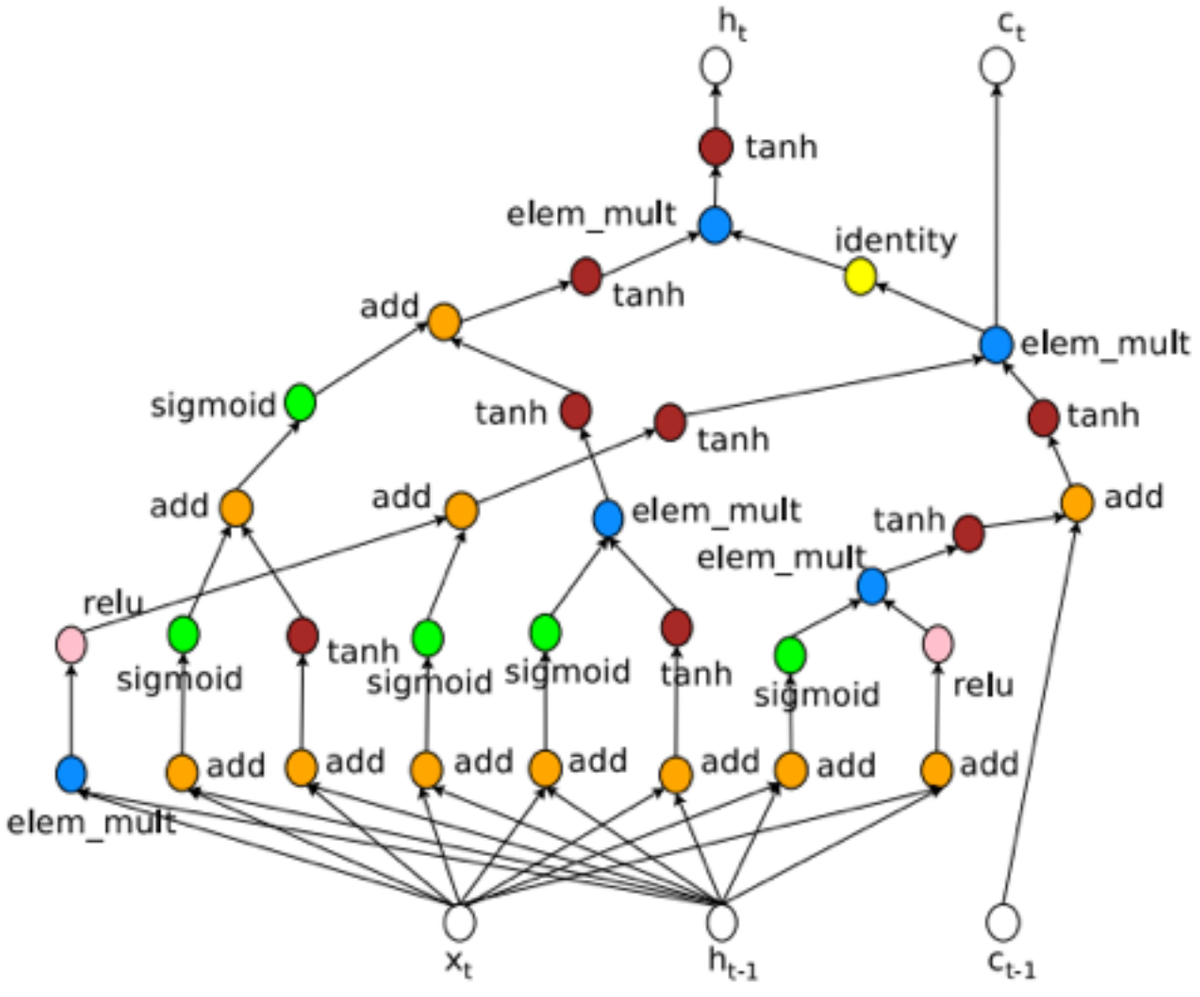
- Controller의 Reward는 c/(validation perplexity)^2

slide modified from Zoph, Le

Penn Treebank Results



LSTM Cell



Neural Architecture Search (NAS) Cell

Other Experiment Info.

- PennTree Bank: LSTM cell들과 비슷하다. Attention과 비슷한것을 한다.
- Google Brain에서 유전자 프로그래밍의 진화로 CIFAR-10을 위한 CNN을 만들어 봤는데 더 오래 걸리고 안 좋은 결과를 얻었다.
- 베이지안 최적화와 비교도 해봤다. NAS가 더 많은 기기를 써서 좋은 비교는 아니지만, NAS가 더 좋은 결과가 나왔고 더 확장가능하다. 베이지안 최적화에서 inverse matrix가 있고 엄청 큰 matrix (20,000 by 20,000)이 있어서 어렵다. NAS는 inversion이 없어서 학습하기 쉽다.
- 마지막에 끝나는 토큰을 추가했다. 실제로 학습하기 좋은 조그마한 네트워크를 좋아한다. 작은 string lists 에서 시작해서 더 늘려나간다.
- 처음에는 network가 무작위이기 때문에 다양했다. 하지만 수렴할때 좋은 architecture로 좁혀간다. 이게 문제가 될수 있어서 노이즈를 조금주면서 overfit을 막는다.

Penn Treebank Results

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

2x as fast

slide from Zoph, Le

RHN (Recurrent Highway Network)

Recurrent Highway Network, Zilly et al. 2016

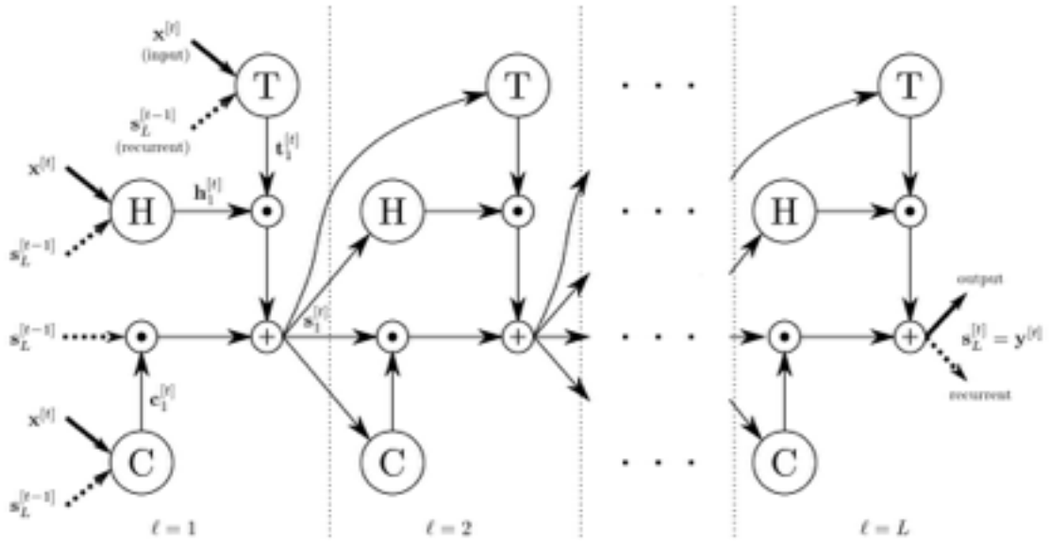


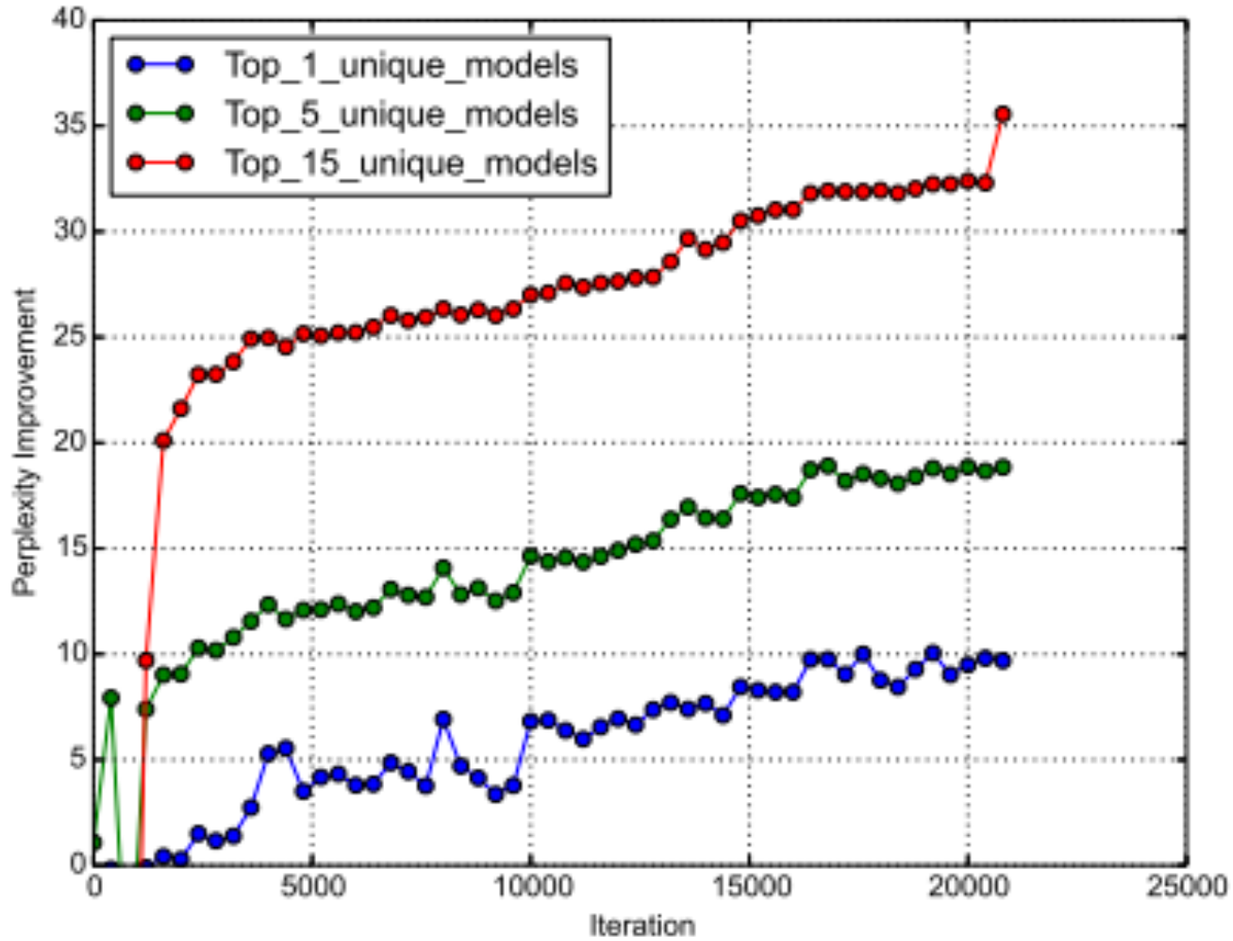
Figure 3: Schematic showing computation within an RHN layer inside the recurrent loop. Vertical dashed lines delimit stacked Highway layers. Horizontal dashed lines imply the extension of the recurrence depth by stacking further layers. H, T & C are the transformations described in equations 7, 8 and 9, respectively.

$$\mathbf{h}_\ell^{[t]} = tanh(\mathbf{W}_H \mathbf{x}^{[t]} \mathbb{I}_{\{\ell=1\}} + \mathbf{R}_{H\ell} \mathbf{s}_{\ell-1}^{[t]} + \mathbf{b}_{H\ell}), \quad (7)$$

$$\mathbf{t}_\ell^{[t]} = \sigma(\mathbf{W}_T \mathbf{x}^{[t]} \mathbb{I}_{\{\ell=1\}} + \mathbf{R}_{T\ell} \mathbf{s}_{\ell-1}^{[t]} + \mathbf{b}_{T\ell}), \quad (8)$$

$$\mathbf{c}_\ell^{[t]} = \sigma(\mathbf{W}_C \mathbf{x}^{[t]} \mathbb{I}_{\{\ell=1\}} + \mathbf{R}_{C\ell} \mathbf{s}_{\ell-1}^{[t]} + \mathbf{b}_{C\ell}), \quad (9)$$

Comparison to Random Search



- Policy gradient 대신에 random search를 사용해서 제일 좋은 네트워크를 찾을수 있다.
- 하지만 policy gradient가 더 좋다.

Transfer Learning on Character Level Language Modeling

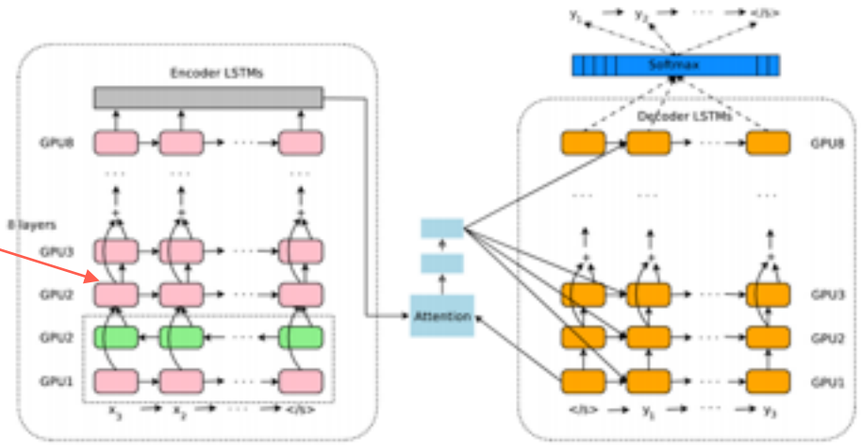
낮을수록 좋다

RNN Cell Type	Parameters	Test Bits Per Character
Ha et al. (2016) - Layer Norm HyperLSTM	4.92M	1.250
Ha et al. (2016) - Layer Norm HyperLSTM Large Embeddings	5.06M	1.233
Ha et al. (2016) - 2-Layer Norm HyperLSTM	14.41M	1.219
Two layer LSTM	6.57M	1.243
Two Layer with New Cell	6.57M	1.228
Two Layer with New Cell	16.28M	1.214

- Took the RNN cell that we evolved on Penn Treebank and tried it in on other datasets. Here, we tried on character level Penn Treebank datasets.

Transfer Learning on Neural Machine Translation

LSTM Cell



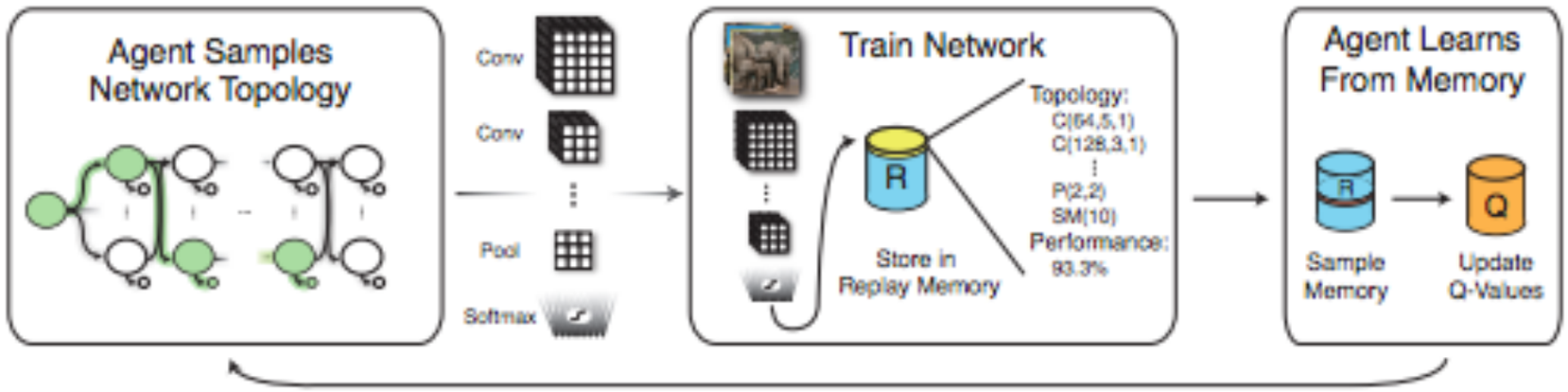
Google Neural Machine Translation, Wu et al. 2016

Model	WMT'14 en->de Test Set BLEU
GNMT LSTM	24.1
GNMT NAS Cell	24.6

- GNMT에서 LSTM빼고 NAS를 통해서 만든 cell을 넣었다.
- LSTM에 특화된 Hyperparameter(learning rate, weight initialization)들을 튜닝을 안했다.
- 0.5 BLEU 점수가 높아졌다. 꽤 의미있는 결과이다.
- 96 GPU들을 써서 1주일동안 train했다.

slide modified from Zoph, Le

Designing Neural Network Architectures Using RL



- 2016년 11월 30일에 발표
- MetaQNN, a meta-modeling algorithm based on RL to automatically generate high-performing CNN architectures for a given learning.
- The learning agent is trained to sequentially choose CNN layers using Q-Learning with an epsilon-greedy exploration strategy and experience replay.
- The agent explores a large but finite space of possible architectures and iteratively discovers designs with improved performance on the learning task.

UNDERSTANDING DEEP LEARNING REQUIRES RETHINKING GENERALIZATION (Zhang et al. 2016)

---

1. The effective capacity of neural networks is large enough for a brute-force memorization of the entire data set.

2. Even optimization on random labels remains easy. In fact, training time increases only by a small constant factor compared with training on the true labels.

3. Randomizing labels is solely a data transformation, leaving all other properties of the learning problem unchanged.

우리는 뉴럴 네트워크 모델의 '일반화'에 대해서 거의 이해하지 못하고 있다.

Reference

- https://arxiv.org/pdf/1611.01578.pdf
- https://arxiv.org/pdf/1608.06993.pdf
- https://arxiv.org/pdf/1606.04474.pdf
- https://arxiv.org/pdf/1512.03385.pdf
- https://arxiv.org/pdf/1611.02167.pdf
- https://arxiv.org/pdf/1607.03474.pdf
- https://arxiv.org/abs/1609.08144.pdf
- https://arxiv.org/abs/1602.07261.pdf
- https://openreview.net/pdf?id=Sy8gdB9xx
- https://futurism.com/googles-new-ai-is-better-at-creating-ai-than-the-companys-engineers/
- http://www.iclr.cc/doku.php?id=iclr2017:schedule
- http://techm.kr/bbs/?t=154
- https://medium.com/intuitionmachine/deep-learning-the-unreasonable-effectiveness-of-randomness-14d5aef13f87
- http://rll.berkeley.edu/deeprlcourse/docs/quoc_barret.pdf
- http://www.1-4-5.net/~dmm/ml/nnrnn.pdf
- https://blog.acolyer.org/2017/05/10/neural-architecture-search-with-reinforcement-learning/

Appendix 1: REINFORCE in depth

$$y = p(\mathbf{x};\theta) \qquad \#\ \text{definition; likelihood parametrized by } \theta \qquad (1)$$

$$z = \log y = \log p(\mathbf{x};\theta) \qquad \#\ \text{definition; z is the log likelihood} \qquad (2)$$

$$\frac{dz}{d\theta} = \frac{dz}{dy} \cdot \frac{dy}{d\theta} \qquad \#\ \text{chain rule definition} \qquad (3)$$

$$\frac{dz}{dy} = \frac{1}{p(\mathbf{x};\theta)} \qquad \#\ \frac{\log(X)}{dX} \approx \frac{1}{X} \qquad (4)$$

$$\frac{dy}{d\theta} = \frac{d\ p(\mathbf{x};\theta)}{d\theta} = \nabla_\theta p(\mathbf{x};\theta) \qquad \#\ \text{definition (chain rule, again)} \qquad (5)$$

$$\frac{dz}{d\theta} = \frac{dz}{dy} \cdot \frac{dy}{d\theta} = \frac{\nabla_\theta\ p(\mathbf{x};\theta)}{p(\mathbf{x};\theta)} \qquad \#\ \text{chain rule} \qquad (6)$$

$$= \nabla_\theta \log p(\mathbf{x};\theta) \qquad \#\ \text{using the identity } \nabla_\theta \log(w) = \frac{1}{w}\nabla_\theta w \qquad (7)$$

note by David Meyer

Appendix 1: REINFORCE in depth

Setting $\pi_\theta(s,a) = p(\mathbf{x};\theta)$ we see that

$$\nabla_\theta \pi_\theta(s,a) = \pi_\theta(s,a)\frac{\nabla_\theta\, \pi_\theta(s,a)}{\pi_\theta(s,a)} \tag{8}$$

$$= \pi_\theta(s,a)\nabla_\theta \log \pi_\theta(s,a) \qquad \#\ \text{log derivative trick} \tag{9}$$

and the score function is $\nabla_\theta \log \pi_\theta(s,a)$.

Now, since here $\pi_\theta(s,a) = P(a_{1:T};\theta_c)$, we have

$$\nabla_{\theta_c} J(\theta_c) = \sum_{s\in S} d(s) \sum_{a\in A} \nabla_{\theta_c} \pi_{\theta_c}(s,a)\mathcal{R}_{s,a} \qquad \#\ \text{defn policy gradient} \tag{10}$$

$$= \sum_{s\in S} d(s) \sum_{a\in A} \pi_{\theta_c}(s,a)\nabla \log \pi_{\theta_c}(s,a)R_{s,a} \qquad \#\ \text{log derivative trick (Eqn 9)} \tag{11}$$

$$= \mathbb{E}_{\pi_{\theta_c}}\left[\nabla_{\theta_c} \log \pi_{\theta_c}(s,a)R\right] \qquad \#\ \text{defn expectation} \tag{12}$$

$$= \mathbb{E}_{a_i\sim P}\left[\nabla_{\theta_c} \log P(a_t|a_{(t-1):1};\theta_c)R\right] \qquad \#\ \pi_\theta(s,a) = P(a_{1:T};\theta_c) \tag{13}$$

$$= \sum_{t=1}^{T} P_{(a_1:T;\theta_c)}\left[\nabla_{\theta_c} \log P(a_t|a_{(t-1):1};\theta_c)R\right] \qquad \#\ \text{REINFORCE pg} \tag{14}$$

note by David Meyer

Appendix 2: Proof of the Policy Gradient Theorem

Proof of the Policy Gradient Theorem (episodic case)

With just elementary calculus and re-arranging terms we can prove the policy gradient theorem from first principles. To keep the notation simple, we leave it implicit in all cases that π is a function of $\boldsymbol{\theta}$, and all gradients are also implicitly with respect to $\boldsymbol{\theta}$. First note that the gradient of the state-value function can be written in terms of the action-value function as

$$
\begin{aligned}
\nabla v_\pi(s) &= \nabla \left[\sum_a \pi(a|s) q_\pi(s,a) \right], \quad \forall s \in \mathcal{S} && \text{(Exercise 3.11)} \\
&= \sum_a \Big[\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \nabla q_\pi(s,a) \Big] && \text{(product rule)} \\
&= \sum_a \Big[\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \nabla \sum_{s',r} p(s',r|s,a)\big(r + \gamma v_\pi(s')\big) \Big] \\
& && \text{(Exercise 3.12 and Equation 3.6)} \\
&= \sum_a \Big[\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \sum_{s'} \gamma p(s'|s,a) \nabla v_\pi(s') \Big] && \text{(Eq. 3.8)} \\
&= \sum_a \Big[\nabla \pi(a|s) q_\pi(s,a) + \pi(a|s) \sum_{s'} \gamma p(s'|s,a) && \text{(unrolling)} \\
&\qquad \sum_{a'} \big[\nabla \pi(a'|s') q_\pi(s',a') + \pi(a'|s') \sum_{s''} \gamma p(s''|s',a') \nabla v_\pi(s'') \big] \Big] \\
&= \sum_{x \in \mathcal{S}} \sum_{k=0}^{\infty} \gamma^k \Pr(s \to x, k, \pi) \sum_a \nabla \pi(a|x) q_\pi(x,a),
\end{aligned}
$$

after repeated unrolling, where $\Pr(s \to x, k, \pi)$ is the probability of transitioning from state s to state x in k steps under policy π. It is then immediate that

$$
\begin{aligned}
\nabla \eta(\boldsymbol{\theta}) &= \nabla v_\pi(s_0) \\
&= \sum_s \sum_{k=0}^{\infty} \gamma^k \Pr(s_0 \to s, k, \pi) \sum_a \nabla \pi(a|s) q_\pi(s,a) \\
&= \sum_s d_\pi(s) \sum_a \nabla \pi(a|s) q_\pi(s,a). && \text{Q.E.D.}
\end{aligned}
$$

Page 269 in Reinforcement Learning: An Introduction by Richard S. Sutton and Andrew G. Barto

http://ufal.mff.cuni.cz/~straka/courses/npfl114/2016/sutton-bookdraft2016sep.pdf

Appendix 2: Large-Scale Evolution of Image Classifiers

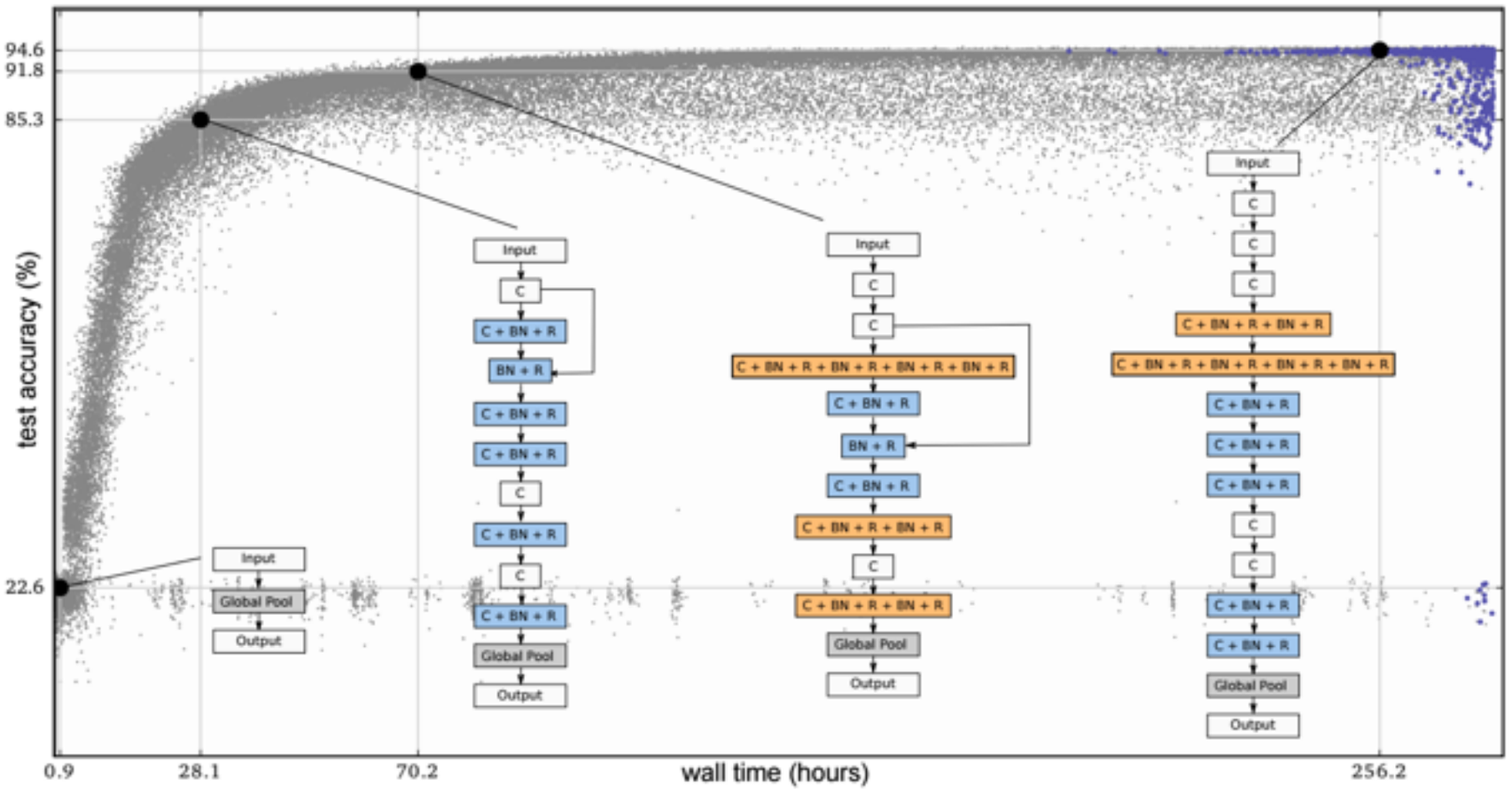
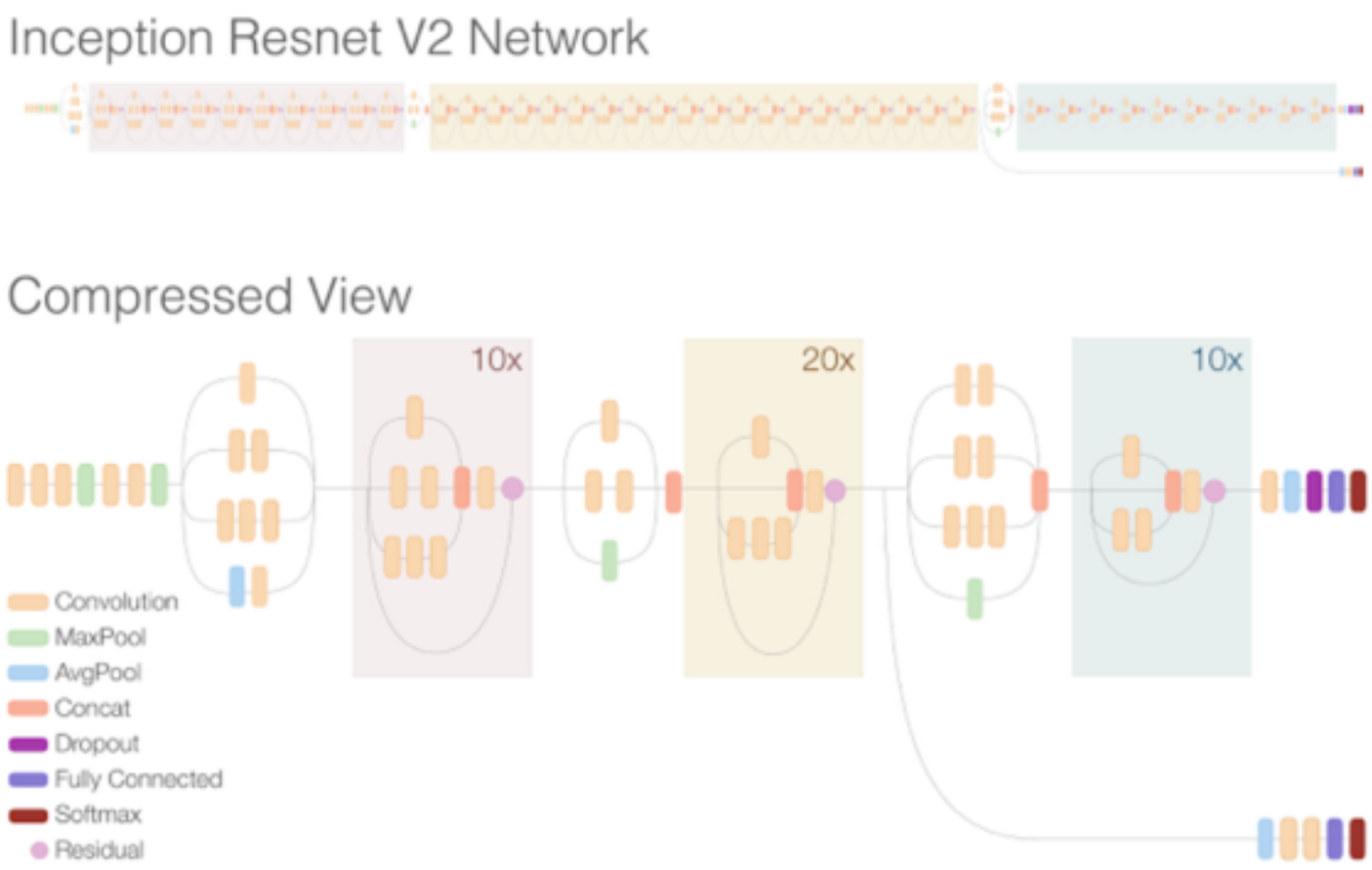


Figure 1. Progress of an evolution experiment. Each dot represents an individual in the population. Blue dots (darker, top-right) are alive. The rest have been killed. The four diagrams show examples of discovered architectures. These correspond to the best individual (rightmost) and three of its ancestors. The best individual was selected by its validation accuracy. Evolution sometimes stacks convolutions without any non-linearity in between ("C", white background), which are mathematically equivalent to a single linear operation. Unlike typical hand-designed architectures, some convolutions are followed by more than one nonlinear function ("$C+BN+R+BN+R+...$", orange background).

Large-Scale Evolution of Image Classifiers, Real et al. 2016

Appendix 3: Inception V4



Schematic diagram of Inception-ResNet-v2

Inception V4, Szegedy et al. 2016