Neural Architecture Search

2018-03-23

곽민구

CONTENTS

Introduction

Neural Architecture Search

Experiments

Efficient NAS

Introduction

01

Neural Architecture Search

Experiments

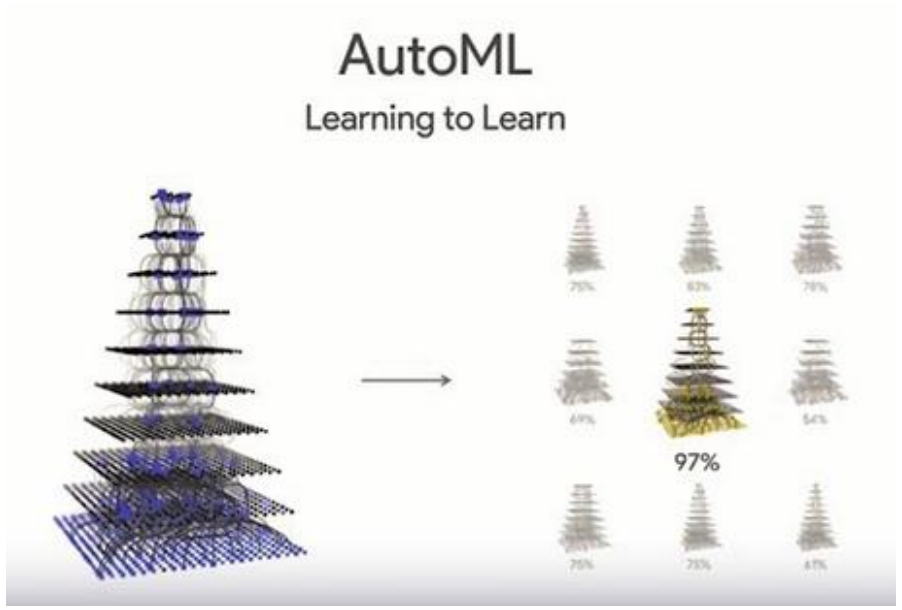
Efficient NAS

AutoML in Google
Motivation
Main Idea

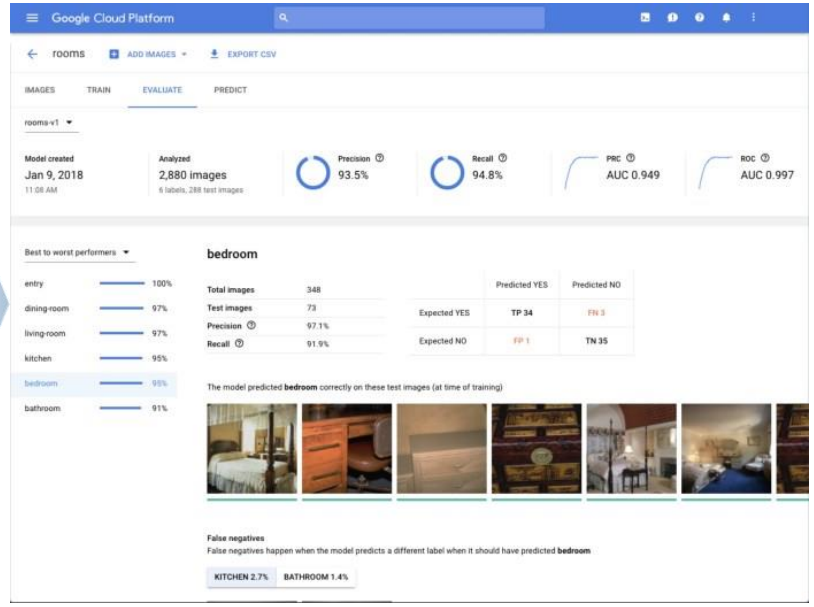
AutoML (Google)

자동화 머신러닝 프로젝트

- 기업간 자원 격차를 줄이고, 모든 비즈니스에서 AI를 손쉽게 사용할 수 있도록 돕는 클라우드 서비스
- AI를 구축하는 AI를 만드는 프로젝트



데이터에 알맞은 아키텍쳐 설계 및 선정

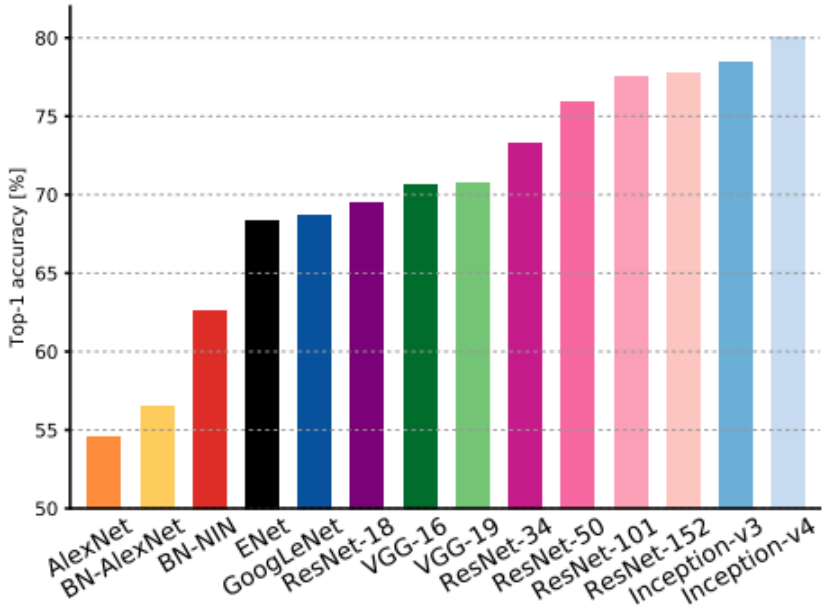


인공지능 시스템 구축

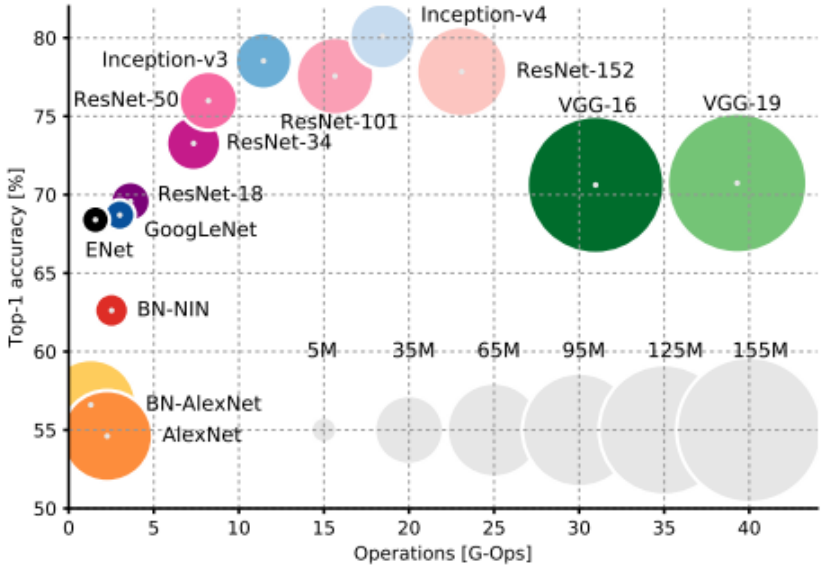
Motivation

» 성능이 좋은 네트워크 구조를 학습하자

- 전문가도 데이터에 알맞은 네트워크 구조를 디자인하는데 오랜 시간이 걸리며 튜닝에도 많은 노력이 필요함
- 또한, 학습 자체에도 오랜 시간이 걸리는 경우가 대부분
- 딥러닝 구조를 만들어주는 모델을 구성한다면?



Top1 Accuracy *vs* Network Type



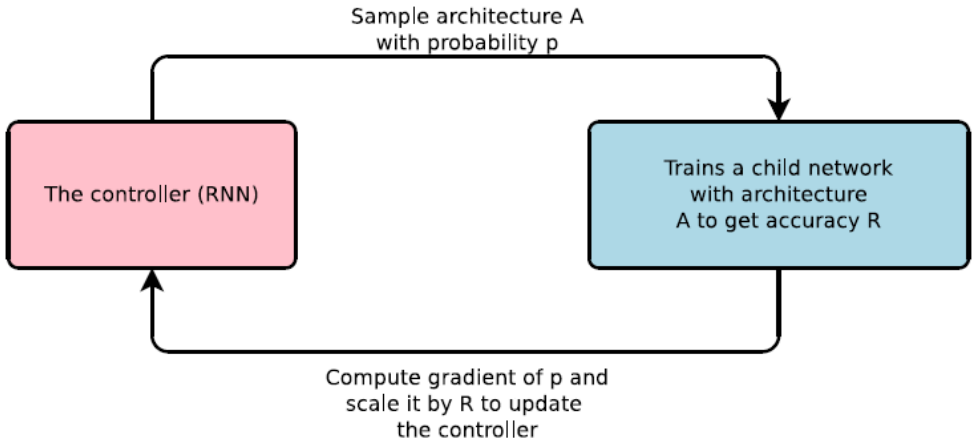
Top1 Accuracy *vs*

Operations, size, parameters

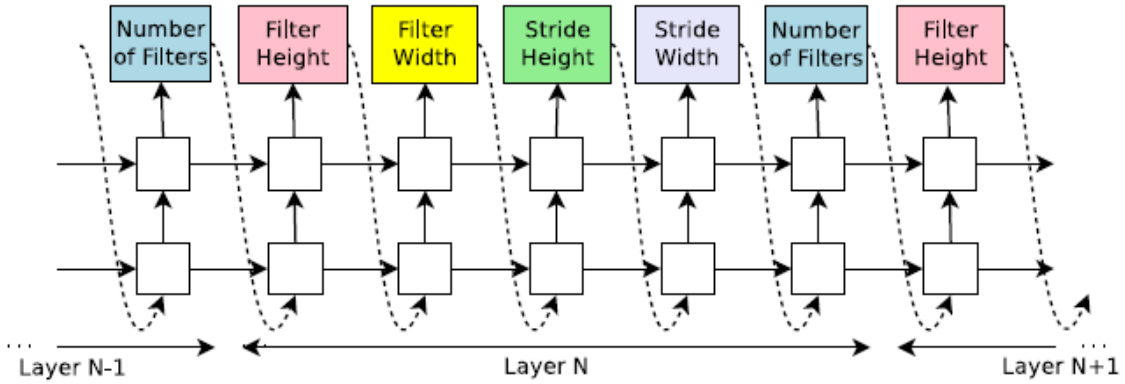
Main Idea

구조를 디자인하는 Controller를 학습

- RNN을 기반으로 하는 Controller는 뉴럴 네트워크의 구조를 결정짓는 파라미터들을 결정
- 결정된 모델 파라미터를 기반으로 여러 개의 네트워크를 동시에 학습
- Validation Accuracy를 Reward (R)로 사용하여 Controller의 파라미터를 업데이트



NAS의 학습 개념도



RNN Controller: CNN 예시

Neural Architecture Search

Introduction | 02 | Experiments | Efficient NAS

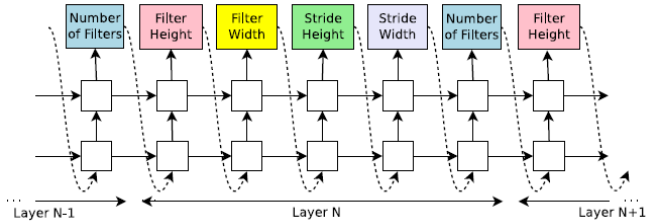
NAS Process
Design Convolutional Neural Network
Design Recurrent Neural Network Cell

NAS Process

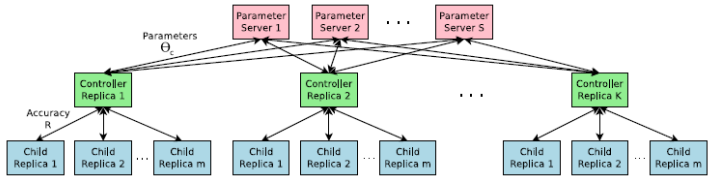
구조를 디자인하는 Controller를 학습

- NAS 모델이 학습되는 과정은 총 4가지 단계를 반복하게 됨

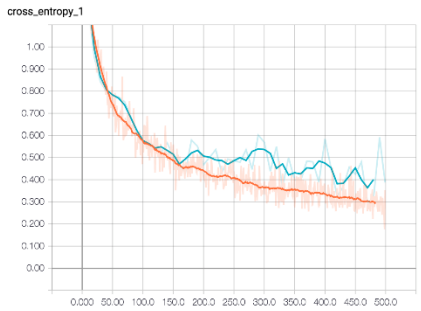
1. 네트워크 Hyperparameter 생성



2. 생성된 네트워크를 학습



3. Validation Accuracy (R)



4. Policy Gradient 기반 모델 업데이트

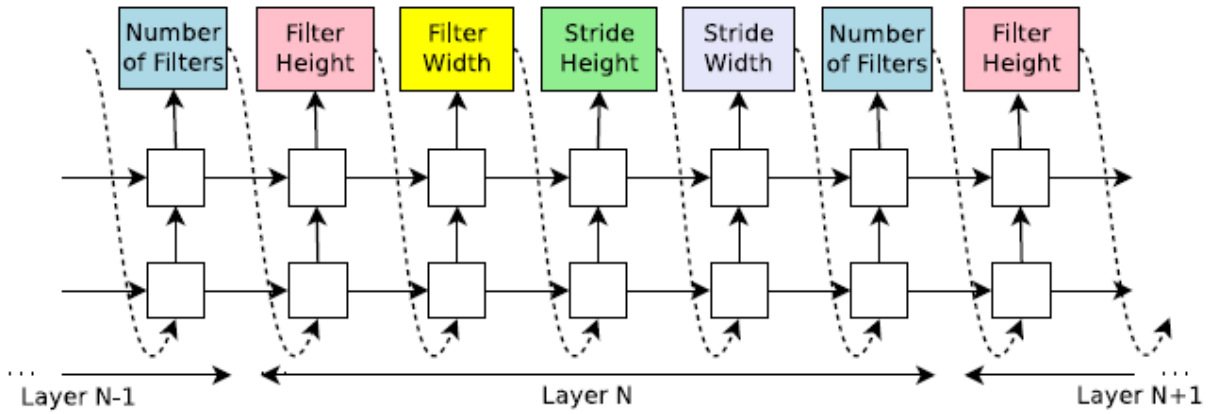
$$\frac{1}{m}\sum_{k=1}^{m}\sum_{t=1}^{T}\nabla_{\theta_c}\log P\left(a_t\middle|a_{(t-1):1};\theta_c\right)(R_k-b)$$

NAS Process

네트워크 Hyperparameter 생성

- Configuration String을 생성하여 모델의 구조를 명시
- 간단한 CNN 모델의 경우: Number of Filters, Filter Height and Width, Stride Height & Width
- Hyperparameter에 따른 Search Space를 사전에 정의하여 그 중 하나를 선택

  ex) Number of Filters=[24, 36, 48, 64] & Filter Height & Width = [1, 3, 5, 7] & Stride Height & Width = [1, 2, 3]



N번째 Layer의 구조를 생성

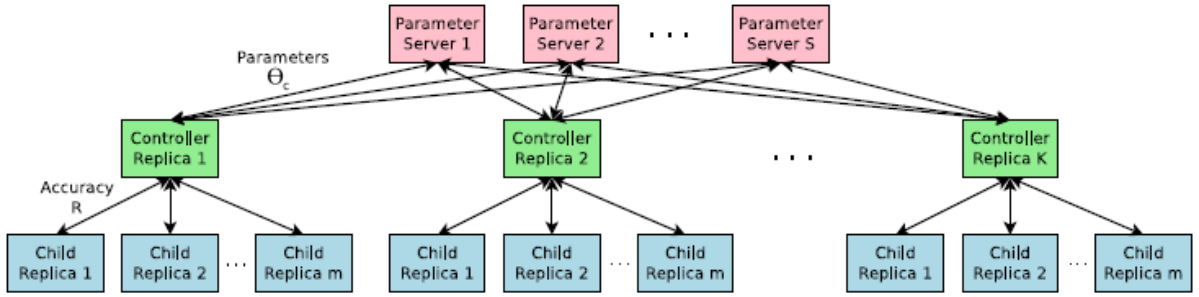
- Seq2Seq 형태를 사용하고 있음
- String을 생성하는 부분은 softmax classifier
- Layer 개수가 일정 이상 지나면 정지

  〉〉 학습 단계에 따라 증가하는 스케줄을 따름

NAS Process

병렬처리를 사용하여 다수의 모델을 동시에 학습

- Parameter Server: S개의 shards를 사용, 병렬처리를 위하여 사용하는 기능
- Controller Replica: RNN Controller 모델 파라미터 θ_c를 공유하는 K개의 replica를 설정
- Child Replica: 1개의 controller replica는 m개의 child replica를 병렬적으로 학습
  (child network: 모델 파라미터로부터 생성된 네트워크)



- 한번에 학습되는 네트워크의 개수는 $m \times K$개
- K개의 controller는 동일한 weight를 공유하지만, asynchronous하게 업데이트
- 각 controller는 m개의 구조를 제안에 각각을 학습시켜 reward를 받게 됨

NAS Process

Policy Gradient를 사용한 RNN Controller 업데이트

- Child Networks로부터 얻은 validation accuracy R은 θ_c에 대하여 미분 불가능: 모델 업데이트를 할 수 없다

➢ Policy Gradient를 기반으로 expected reward $J(\theta_c)$를 최대화

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

- $a_{1:T}$: RNN Controller에서 나온 actions (네트워크 구조 파라미터)
  - 〉〉 모델의 구조를 리스트로 표현한 것
- R: reward. CNN의 경우 마지막 5 epochs 정확도의 최대값

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

- REINFORCE 알고리즘을 사용하여 expected reward를 미분
- Q-learning 등 다른 알고리즘도 사용이 가능하지만,
  REINFORCE가 튜닝이 편한 장점이 있음 (large scale problem)

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

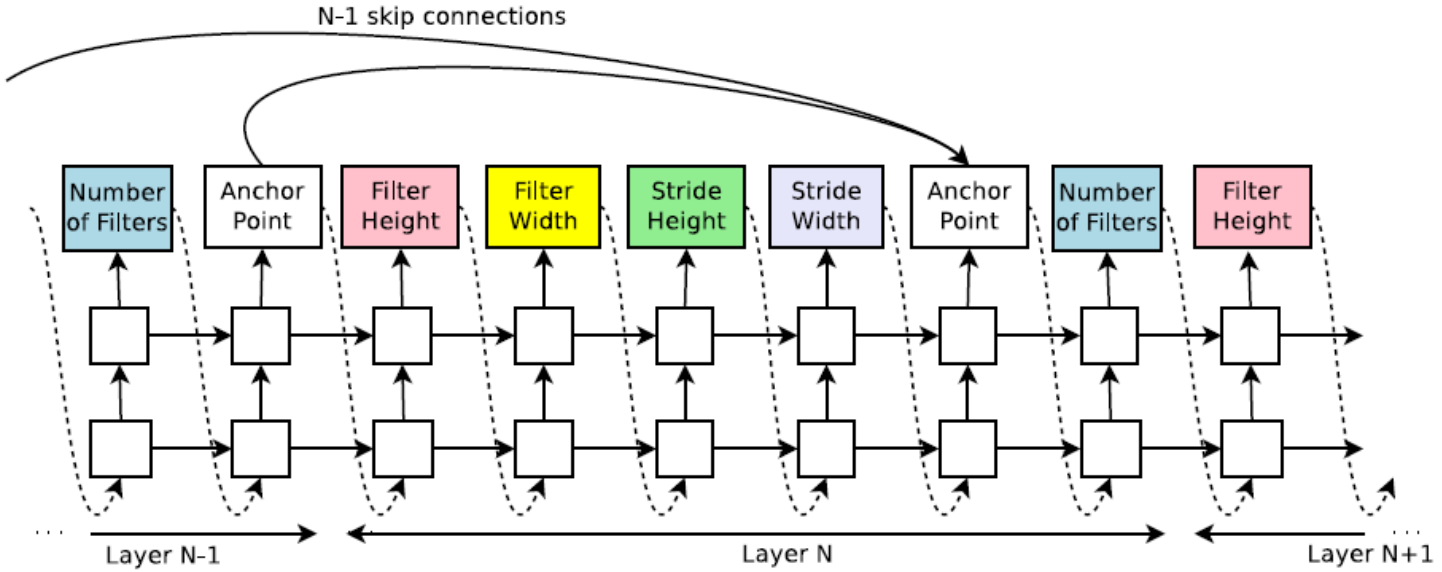
- Empirical Approximation
- m: 1개의 controller가 샘플링하는 모델의 개수
- b: variance를 줄여주기 위한 baseline function
  - 〉〉 이전 정확도의 exponential moving average

Convolutional Neural Network

Skip Connections: Increase Architecture Complexity

- GoogleNet, Residual Net에서 사용되는 skip connection을 생성하도록 모델링
- Anchor Point라는 개념을 사용하여 이전 레이어들의 정보를 받아들이도록 설계

  〉〉 N번째 레이어의 anchor는 $N-1$개의 content-based sigmoids와 연결됨
- $P(\text{layer } j \text{ is an input to layer } i) = \text{sigmoid}(v^T \tanh(W_{prev} * h_j + W_{curr} * h_i))$

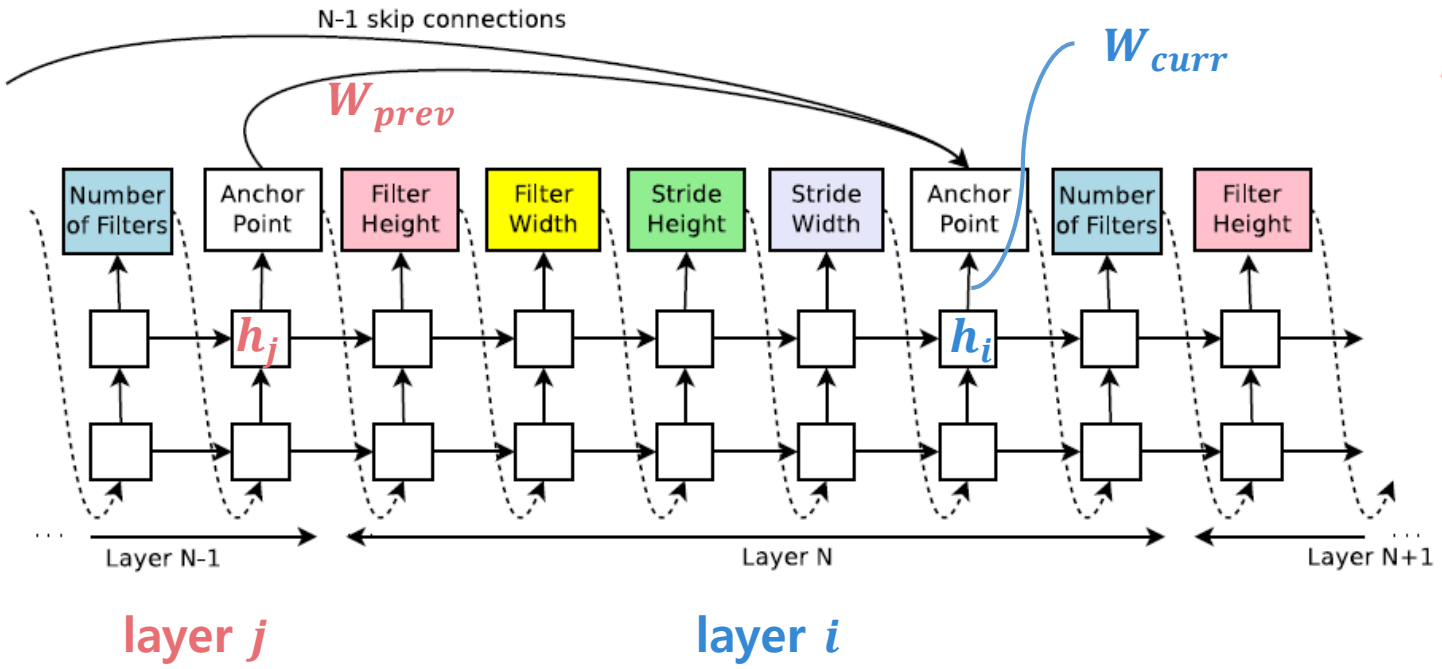


Convolutional Neural Network

Skip Connections: Increase Architecture Complexity

- GoogleNet, Residual Net에서 사용되는 skip connection을 생성하도록 모델링
- Anchor Point라는 개념을 사용하여 이전 레이어들의 정보를 받아들이도록 설계

  〉〉 N번째 레이어의 anchor는 $N-1$개의 content-based sigmoids와 연결됨
- $P(\text{layer } j \text{ is an input to layer } i) = \text{sigmoid}(v^T \tanh(W_{prev} * h_j + W_{curr} * h_i))$



- Skip Connections는 확률분포로 정의된다

  〉〉 REINFORCE로 업데이트가 가능
- v, W_{prev}, W_{curr}: 학습 파라미터
- Residual Connection과 다르다!

Convolutional Neural Network

NAS에서 Skip Connections를 사용하기 위한 Rules

- 여러 개의 input layer가 연결되는 경우 depth dimension을 기준으로 합침
- 이때 차원이 맞지 않아 오류가 발생할 수 있어 다음과 같은 규칙을 정함

① 이전 레이어들과 하나도 연결되지 않는 경우

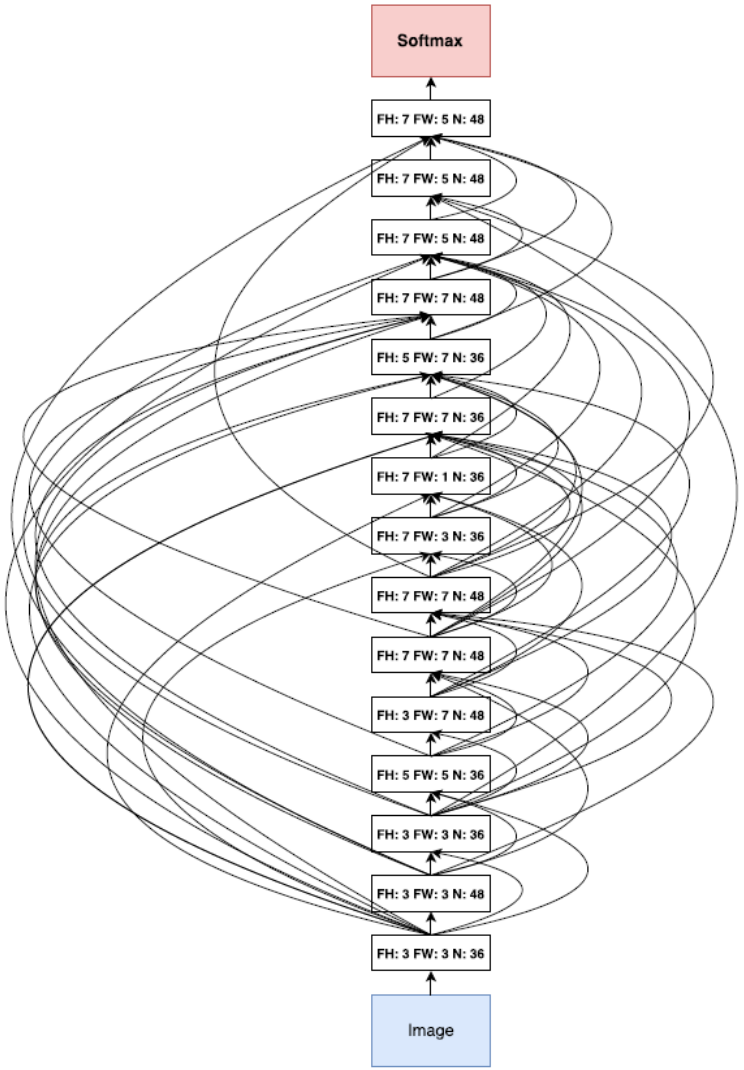
〉〉 Image를 input으로 받는다

② 네트워크의 마지막 레이어

〉〉 '지금까지 어떤 레이어의 input으로 할당되지 않은' 레이어들을 모두 input으로 받는다

③ 서로 다른 크기의 레이어들을 합칠 때

〉〉 작은 크기의 레이어에 zero padding을 한다



* Stride, Pooling은 하지 않은 NAS 예제: 5.50% error rate

Recurrent Neural Network

LSTM, GRU와 유사한 Cell을 설계하기 위한 Controller

- Cell의 구조를 tree 형태로 표현: x_t, h_{t-1}을 input으로 받아 h_t를 output으로 반환하는 형태
- 단계에 따라 Search Space에서 Operation을 선택하여 Cell을 구성
- LSTM의 구조를 사용하기 위해 cell variables c_{t-1}, c_t를 추가: cell 연결은 RNN Controller의 마지막 2개 block에서 예측되도록 설계

Operation 종류

① Combination Method

〉〉 2개의 input을 합치는 역할

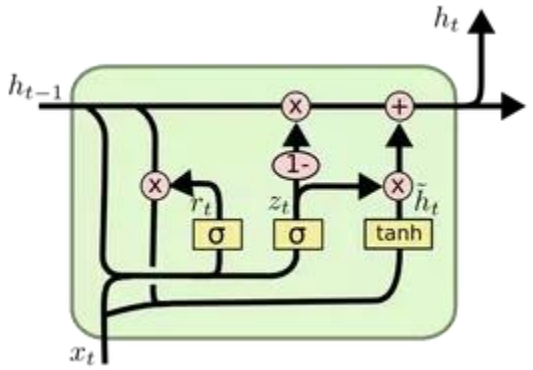
〉〉 Addition, Elementwise Multiplication, etc

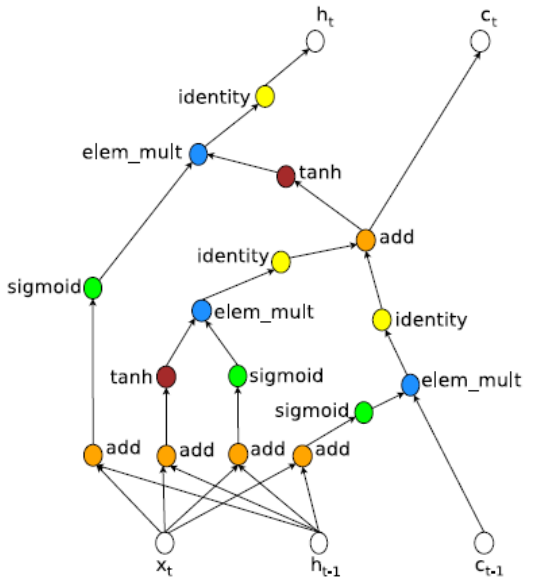
② Activation Function

〉〉 1개의 outpu을 반환하는 역할

〉〉 tanh, sigmoid, relu, etc

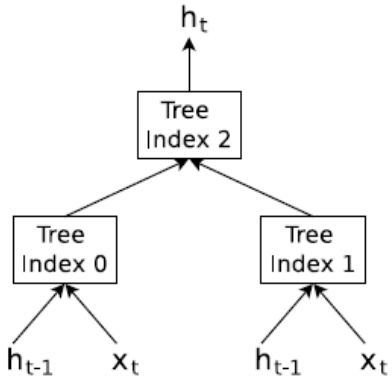
LSTM Cell



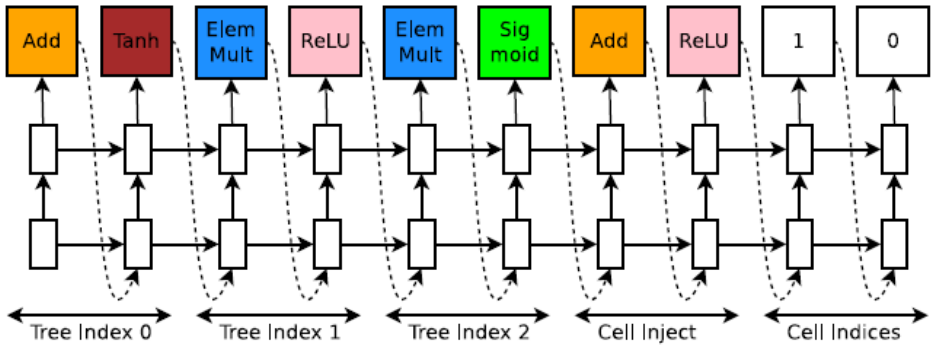


Recurrent Neural Network

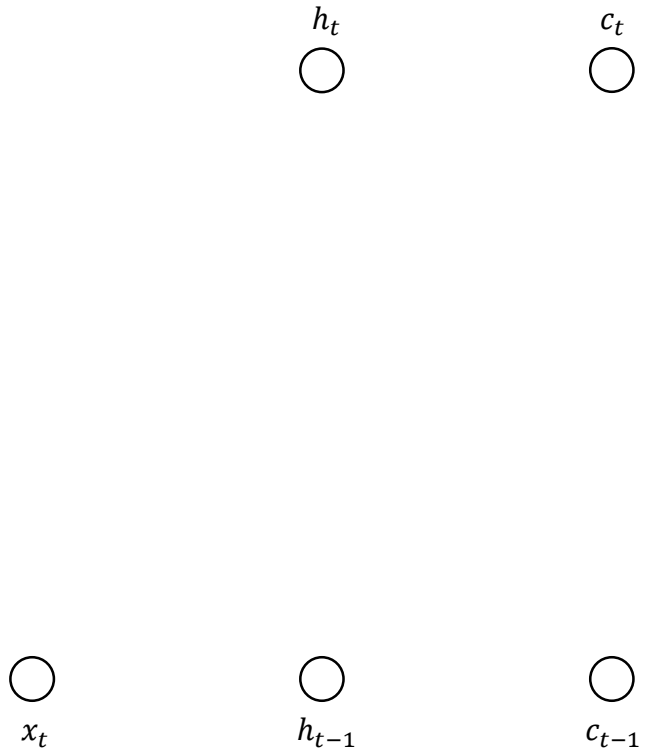
» 2개의 leaf node를 가진 RNN Cell 예시 (base 2)



Tree

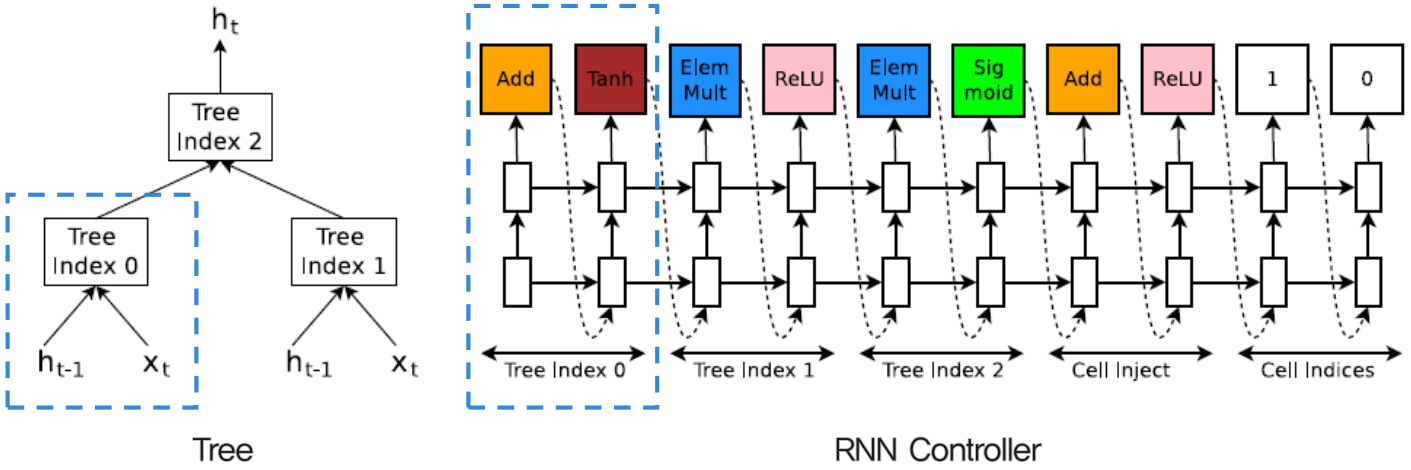


RNN Controller



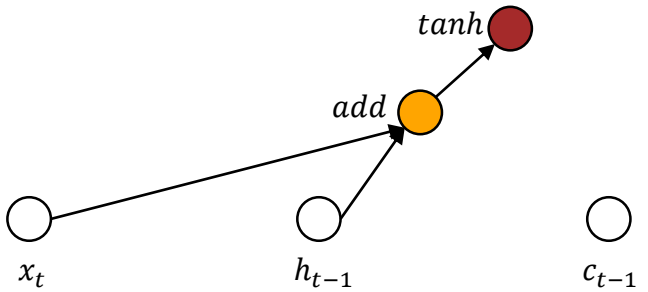
Recurrent Neural Network

» 2개의 leaf node를 가진 RNN Cell 예시 (base 2)



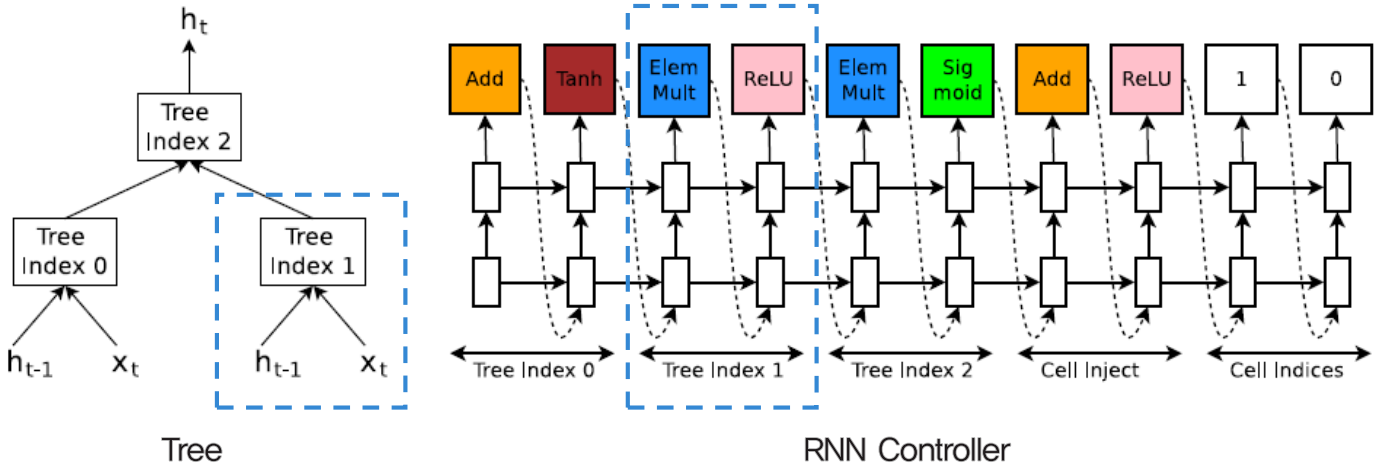
❖ Tree 0에 대하여 Add, Tanh의 연산을 수행하도록 예측

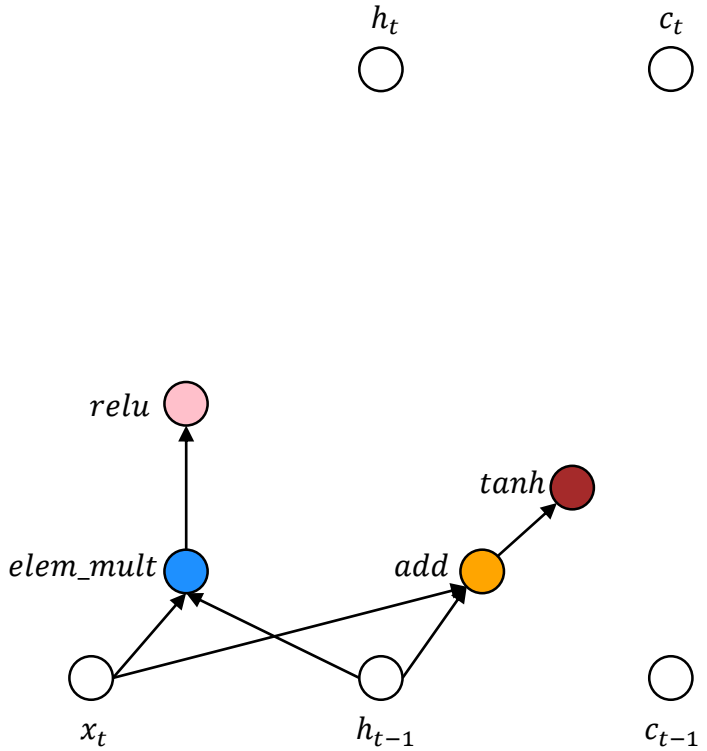
❖ $a_0 = \tanh(W_1 * x_t + W_2 * h_{t-1})$



Recurrent Neural Network

» 2개의 leaf node를 가진 RNN Cell 예시 (base 2)



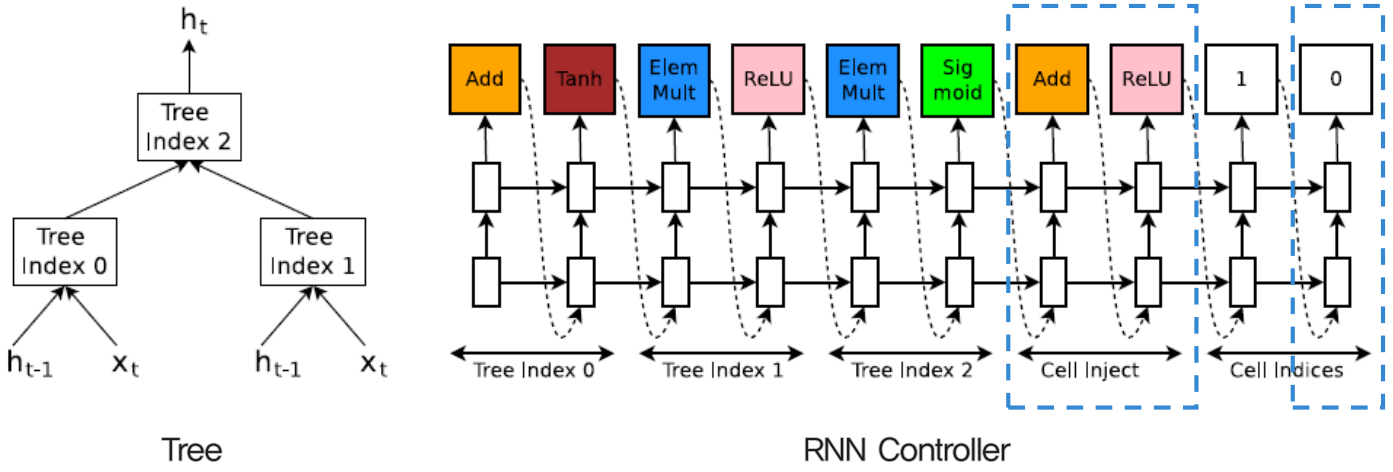


❖ Tree 1에 대하여 elementwise multiplication, relu 연산을 수행하도록 예측

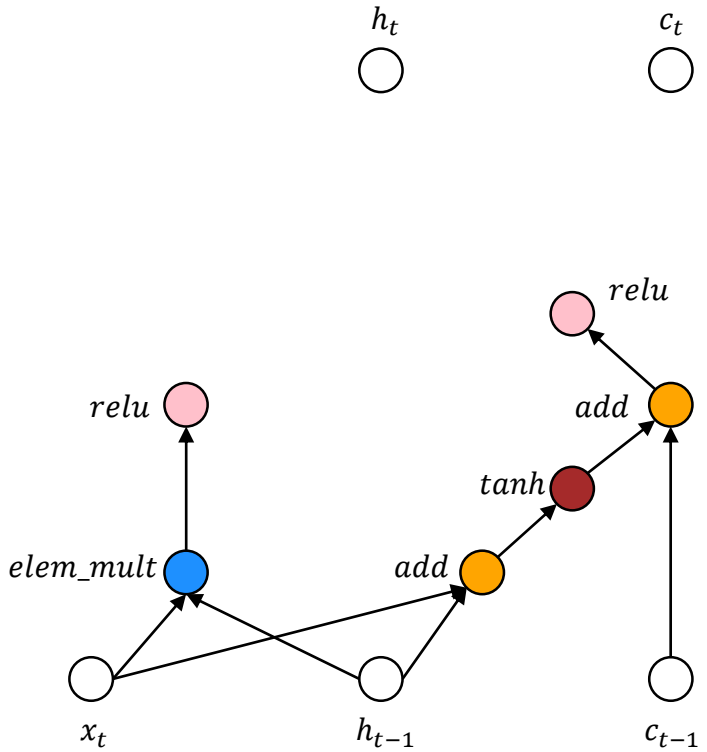
❖ $a_1 = \mathrm{ReLU}((W_3 * x_t) \odot (W_4 * h_{t-1}))$

Recurrent Neural Network

» 2개의 leaf node를 가진 RNN Cell 예시 (base 2)



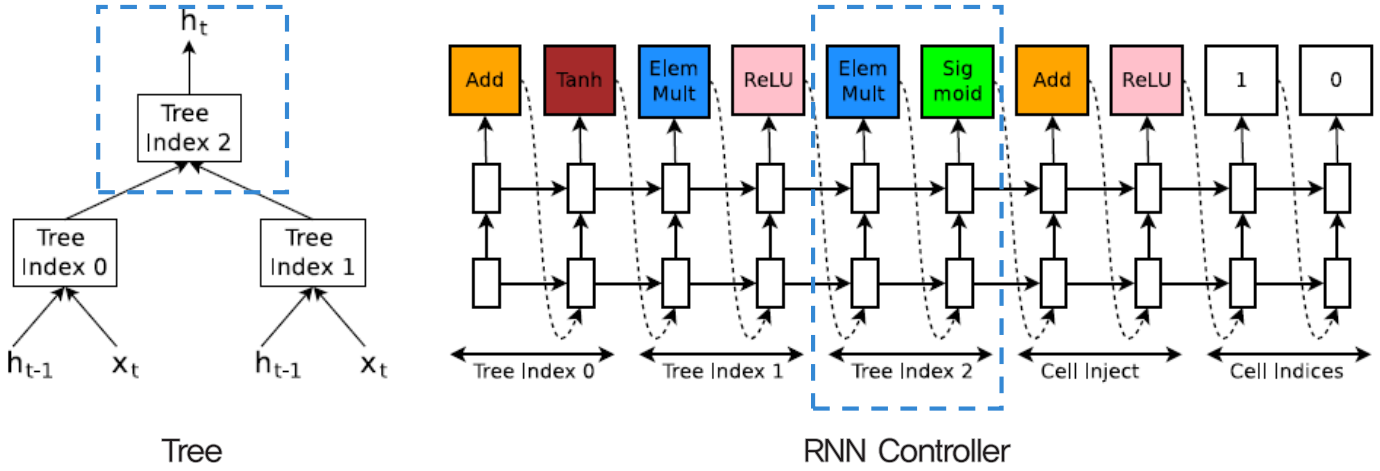
Tree RNN Controller

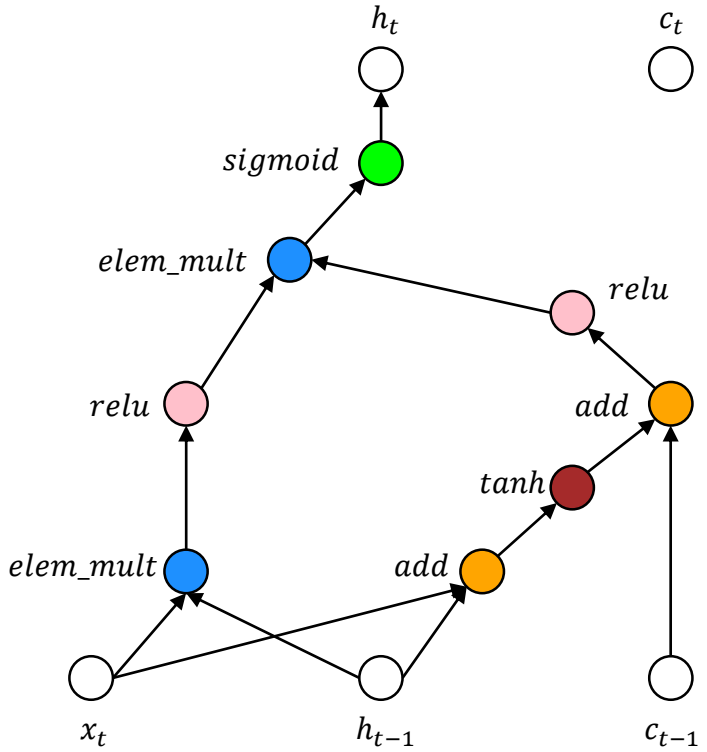


- RNN Controller의 마지막 block: Cell Inject를 지정하는 인덱싱 역할
- $a_0^{new} = \mathrm{ReLU}(a_0 + c_{t-1})$
- Controller에서의 config string 순서는 cell의 연산 순서와 일치하는 것이 아님

Recurrent Neural Network

» 2개의 leaf node를 가진 RNN Cell 예시 (base 2)

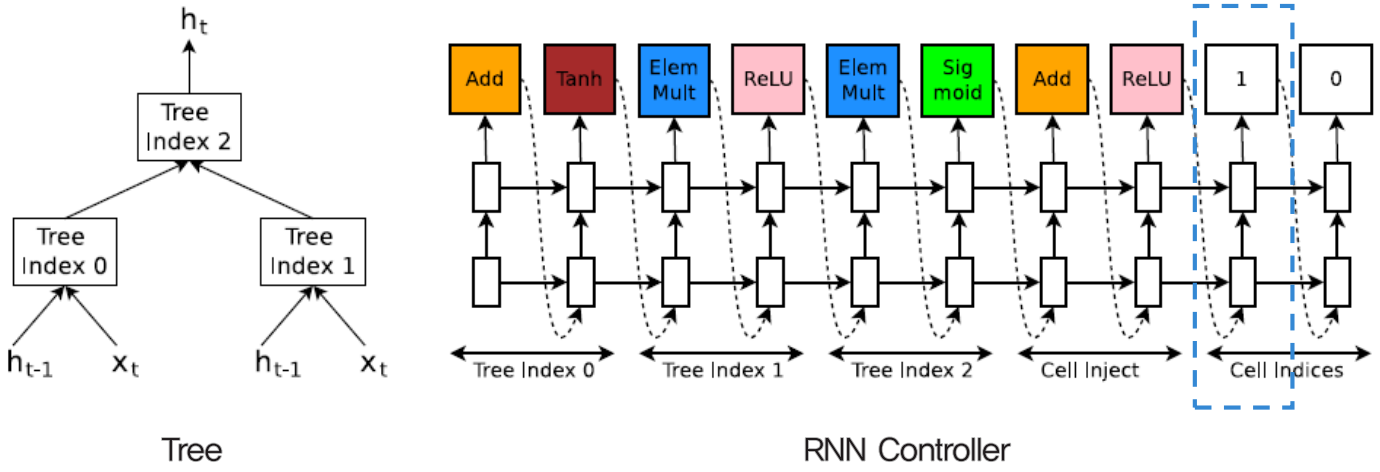


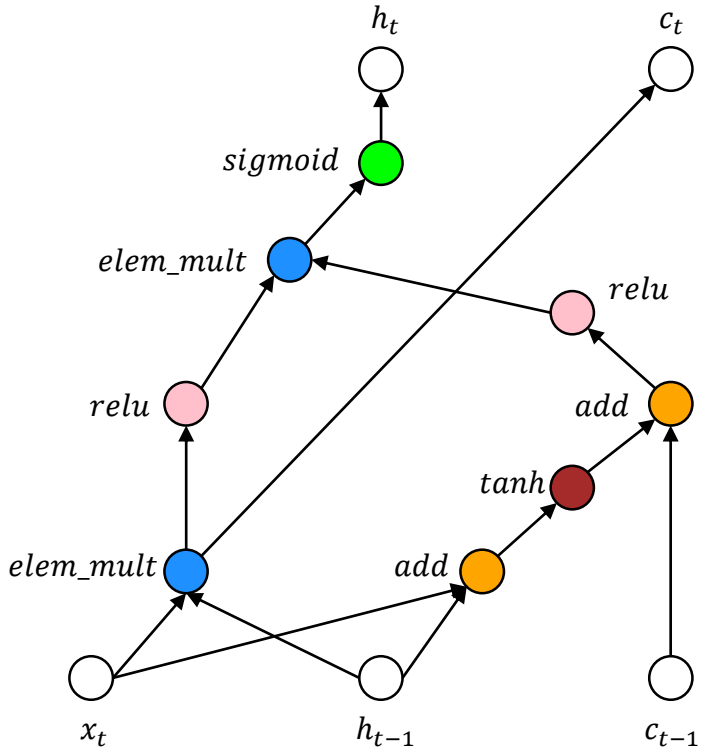


- Tree 2에 대하여 elementwise multiplication, sigmoid 연산을 수행하도록 예측
- $a_2 = \text{sigmoid}(a_0^{new} \odot a_1)$
- 가장 큰 index가 2이므로, $h_t = a_2$

Recurrent Neural Network

» 2개의 leaf node를 가진 RNN Cell 예시 (base 2)

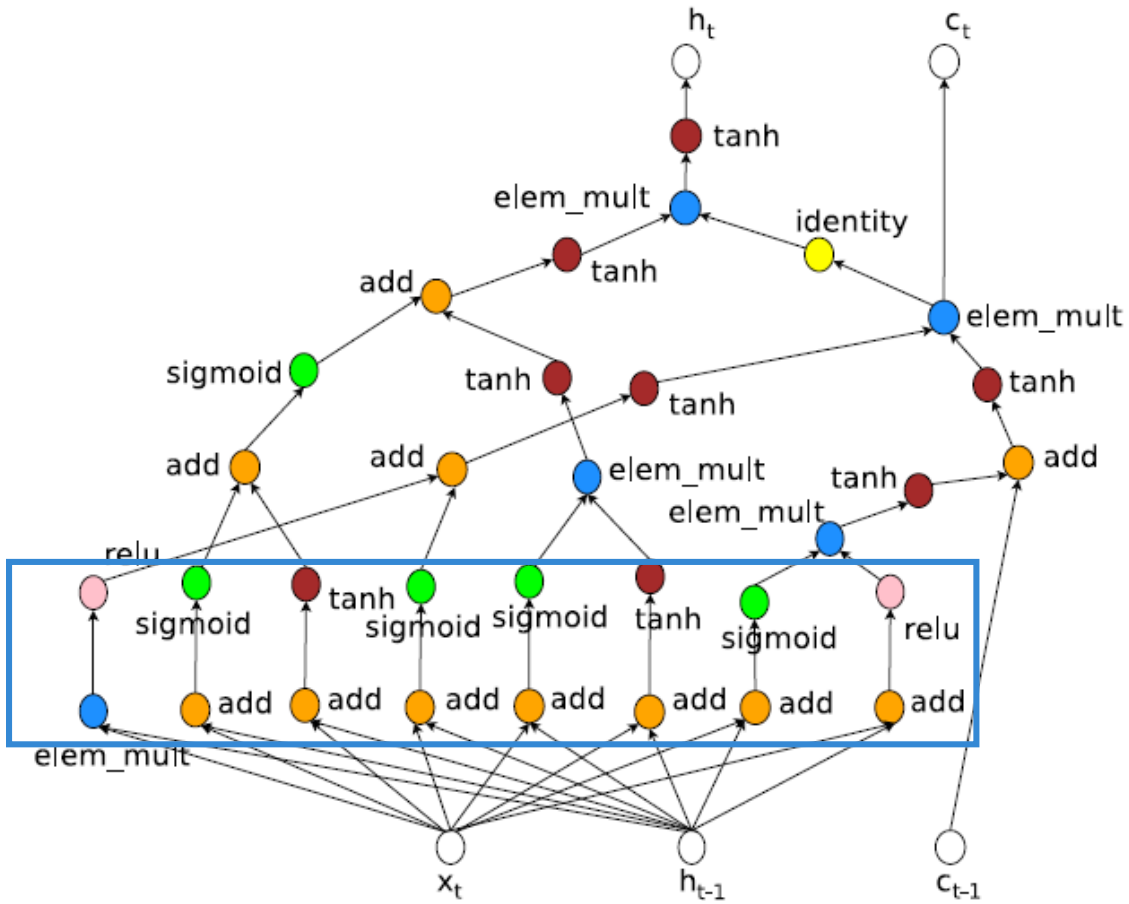




- RNN Controller의 2번째 마지막 block: c_t를 지정할 output of tree를 결정
- Activation Function 이전의 연산 결과를 c_t로 지정
- $a_1 = \mathrm{ReLU}((W_3 * x_t) \odot (W_4 * h_{t-1}))$ → $c_t = (W_3 * x_t) \odot (W_4 * h_{t-1})$

Recurrent Neural Network

» 논문 실험에서 만들어진 NAS Cell 구조 (base 8)

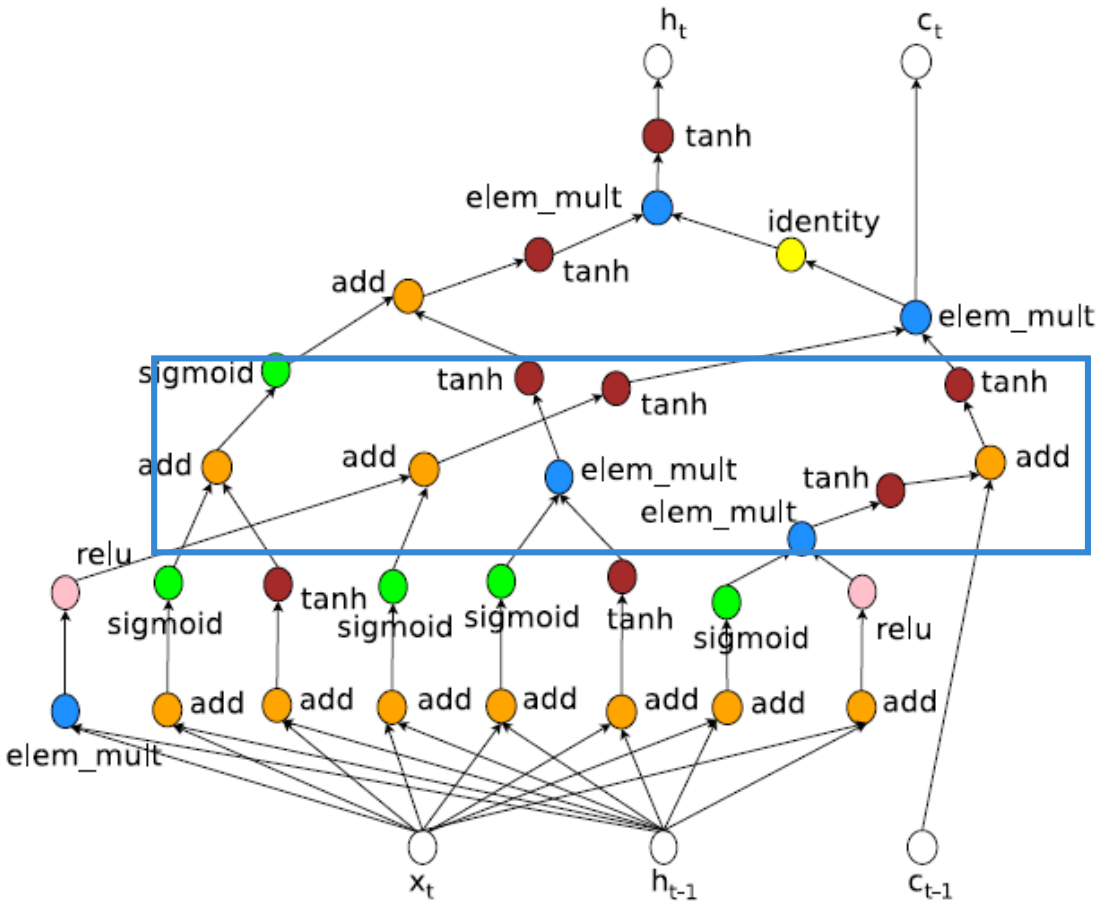


❖ Leaf Node (8개) 연산:

elementwise multiplication 1번, add 7번 〉〉 activation

Recurrent Neural Network

» 논문 실험에서 만들어진 NAS Cell 구조 (base 8)



- Leaf Node (8개) 연산:

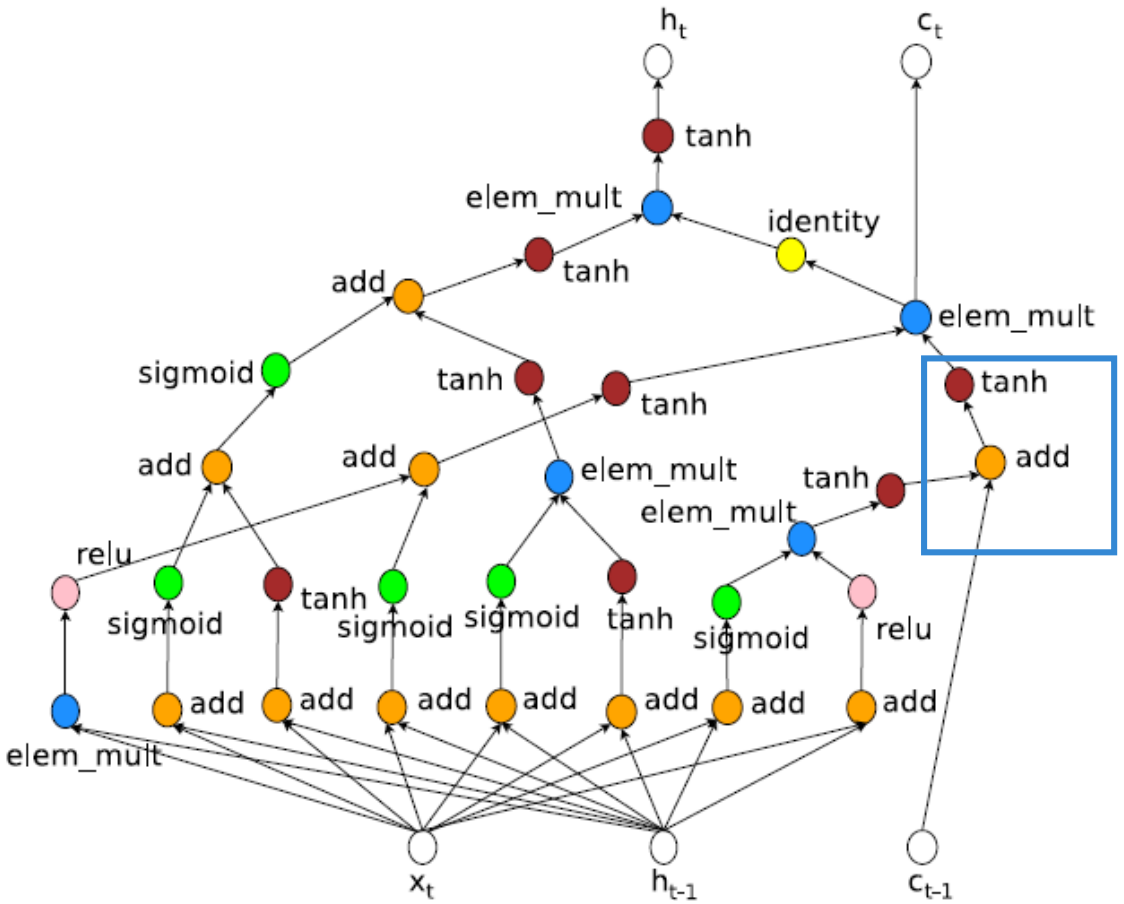
  elementwise multiplication 1번, add 7번 〉〉 activation

- Internal Node (4개) 연산:

  add 2번, elementwise multiplication 2번 〉〉 activation

Recurrent Neural Network

» 논문 실험에서 만들어진 NAS Cell 구조 (base 8)



- ❖ Leaf Node (8개) 연산:

  elementwise multiplication 1번, add 7번 〉〉 activation

- ❖ Internal Node (4개) 연산:

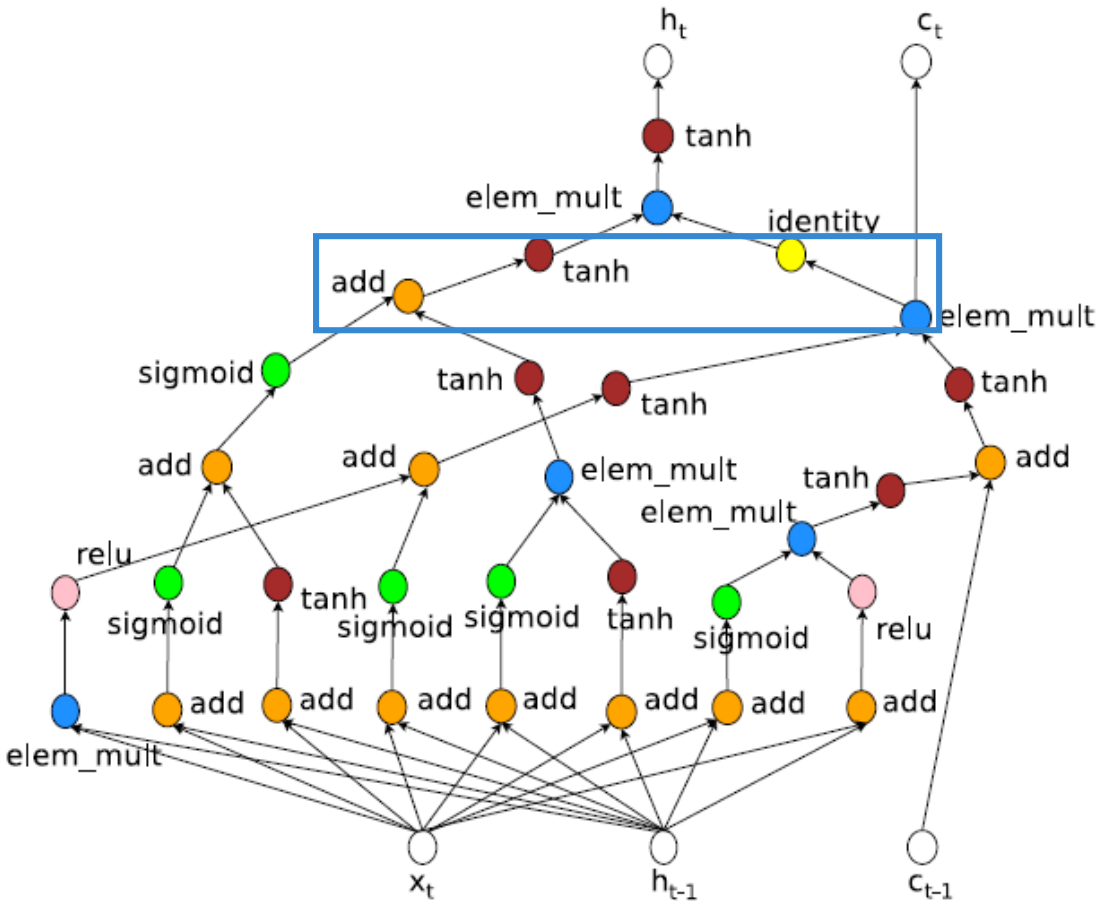
  add 2번, elementwise multiplication 2번 〉〉 activation

- ❖ Cell Injection 연산: 아직까지 tree node는 4개 남음

  〉〉 4번째 internal node에 대하여 add, tanh 연산 시행

Recurrent Neural Network

» 논문 실험에서 만들어진 NAS Cell 구조 (base 8)



- ❖ Leaf Node (8개) 연산:

  elementwise multiplication 1번, add 7번 〉〉 activation

- ❖ Internal Node (4개) 연산:

  add 2번, elementwise multiplication 2번 〉〉 activation

- ❖ Cell Injection 연산: 아직까지 node는 4개 남음

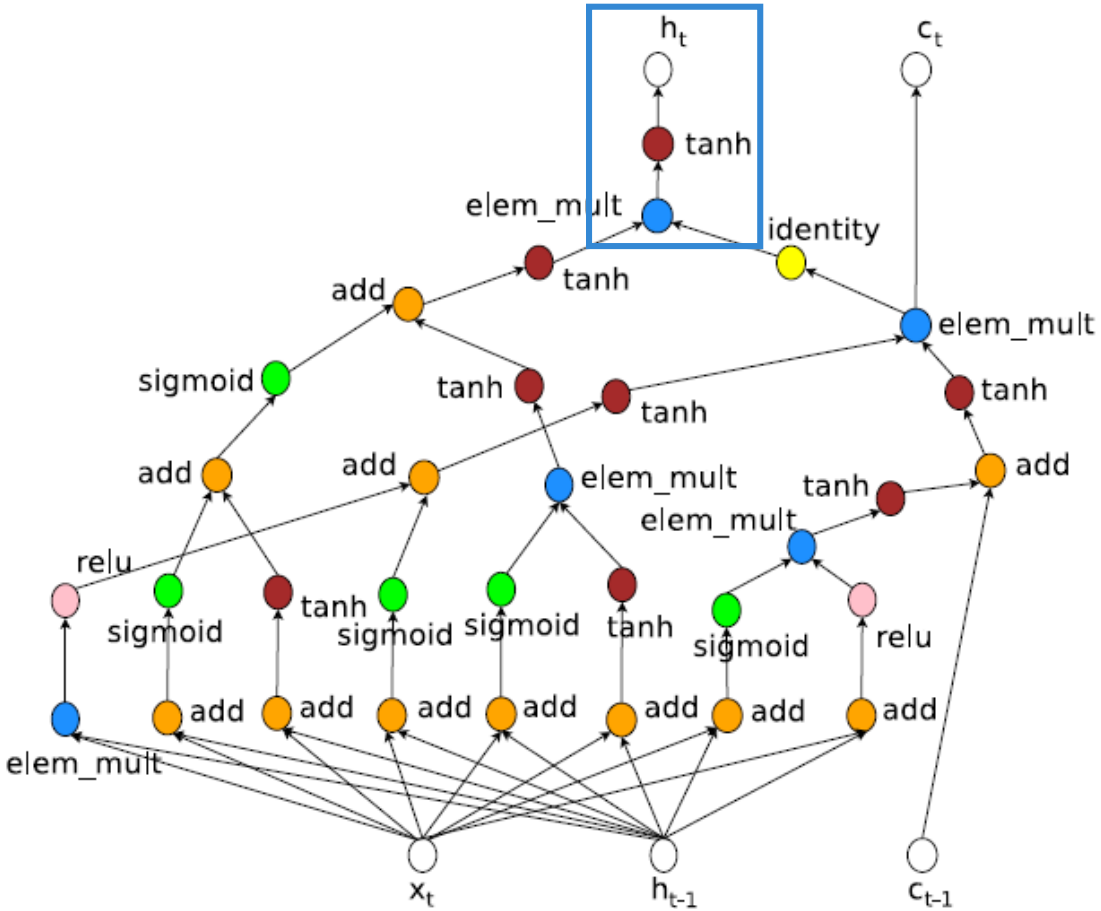
  〉〉 4번째 internal node에 대하여 add, tanh 연산 시행

- ❖ Internal Node (2개) 연산:

  add&tanh, elementwise multiplication&identity

Recurrent Neural Network

» 논문 실험에서 만들어진 NAS Cell 구조 (base 8)



- ❖ Leaf Node (8개) 연산:

  elementwise multiplication 1번, add 7번 〉〉 activation

- ❖ Internal Node (4개) 연산:

  add 2번, elementwise multiplication 2번 〉〉 activation

- ❖ Cell Injection 연산: 아직까지 node는 4개 남음

  〉〉 4번째 internal node에 대하여 add, tanh 연산 시행

- ❖ Internal Node (2개) 연산:

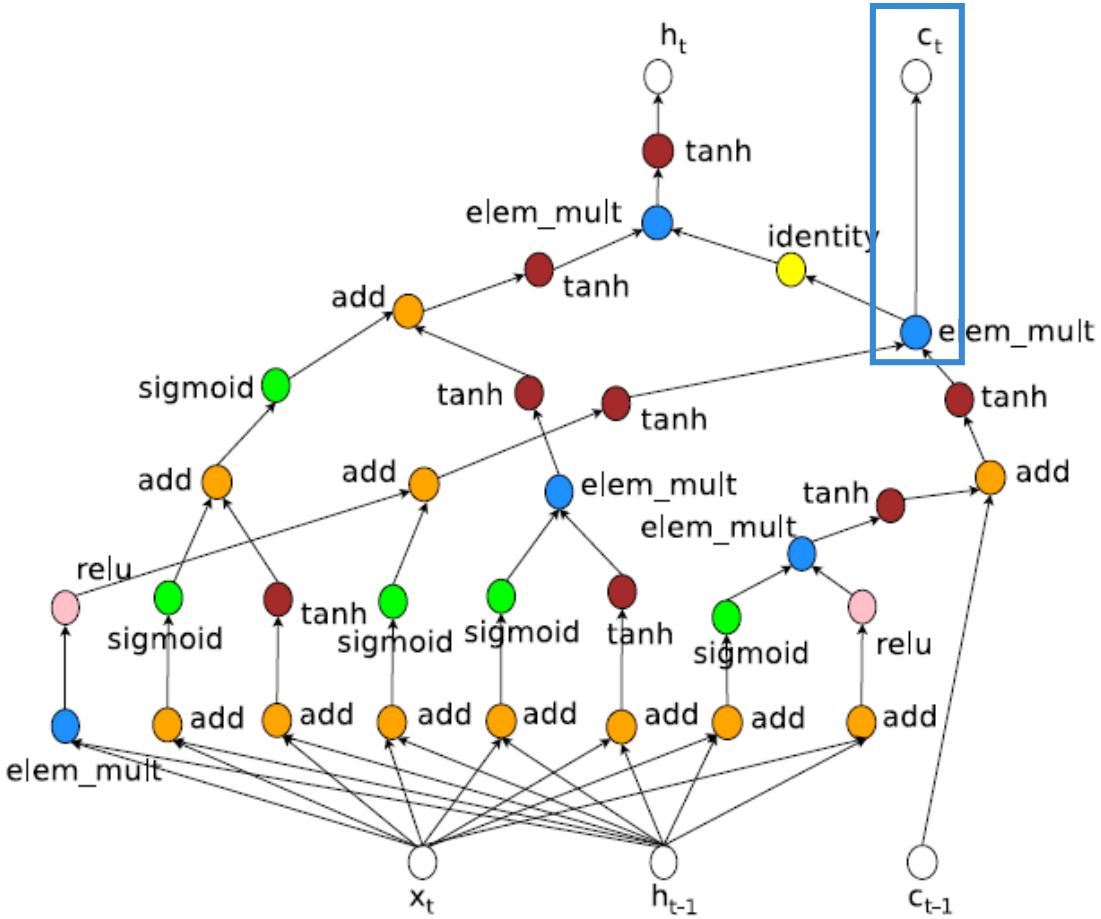
  add&tanh, elementwise multiplication&identity

- ❖ Terminal Node (1개) 연산:

  elementwise multiplication&tanh

Recurrent Neural Network

» 논문 실험에서 만들어진 NAS Cell 구조 (base 8)



- ❖ Leaf Node (8개) 연산:

  elementwise multiplication 1번, add 7번 〉〉 activation

- ❖ Internal Node (4개) 연산:

  add 2번, elementwise multiplication 2번 〉〉 activation

- ❖ Cell Injection 연산: 아직까지 node는 4개 남음

  〉〉 4번째 internal node에 대하여 add, tanh 연산 시행

- ❖ Internal Node (2개) 연산:

  add&tanh, elementwise multiplication&identity

- ❖ Terminal Node (1개) 연산:

  elementwise multiplication&tanh

- ❖ 다음 cell 정보 저장: identity function 이전의 정보를 c_t로 저장

Introduction | Neural Architecture Search | Efficient NAS

Experiments

03

CIFAR-10
Penn Treebank

CNN

실험 세팅: CIFAR-10

- 32x32 cropped image
- $K = 100,\ m = 8$: 800개의 모델을 한번에 학습
- Reward: 마지막 5 epoch 중 가장 높은 validation accuracy
- Search Space를 바꾸거나, max pooling layer 등을 추가하기도 함
- RELU, BN은 기본적으로 포함하는 형태

실험 결과

- 가장 좋은 아키텍쳐의 성능: 3.65%

  〉〉 3.74% 성능을 보이는 DenseNet에 비해 성능이 좋으며,

  1.05배 빠른 속도를 보임
- Local Optimum: 조금 구조를 바꾸면 성능이 하락
- Strides: Search Space 증가로 성능 하락

✓ DenseNet의 경우 1x1 Conv를 사용해 파라미터의 수를 줄임

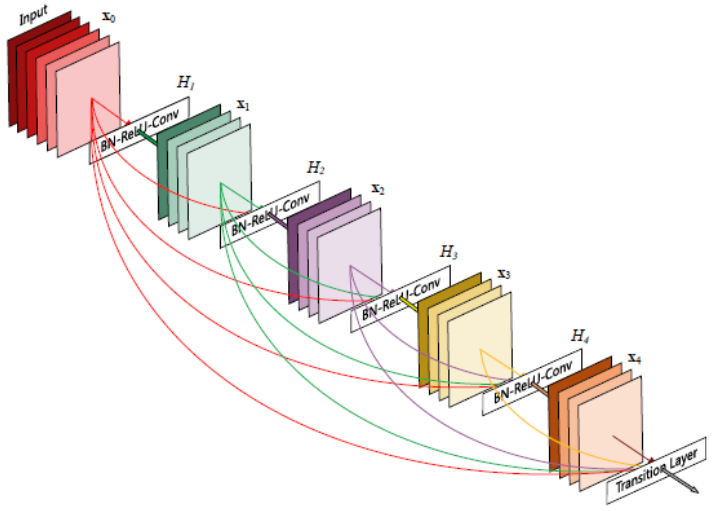
〉〉 Bottleneck Layers & Compression (BC)

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

Appendix. DenseNet

Densely Connected Convolution Networks

- 레이어 l은 이전 모든 레이어 $(0,1,2,\cdots,l-1)$의 레이어와 연결
- $L(L+1)/2$ 개의 연결이 존재하는 블록 구조
- 이전 feature map을 합침으로써 정보를 충분히 유지하고,
  모든 레이어의 정보를 이용해 최종 ouput을 도출하려는 시도



ResNet *vs* DenseNet

① ResNet

$$x_l = H_l(x_{l-1}) + x_{l-1}$$

② DenseNet

$$x_l = H_l([x_0, x_1, \dots, x_{l-1}])$$

H_l: BN + ReLU + 3x3 conv

〉〉 ResNet, Inception과 같이 1x1 conv를 사용하여 채널 차원 축소

〉〉BN + ReLU + 1x1 Conv + BN + ReLU + 3x3 Conv

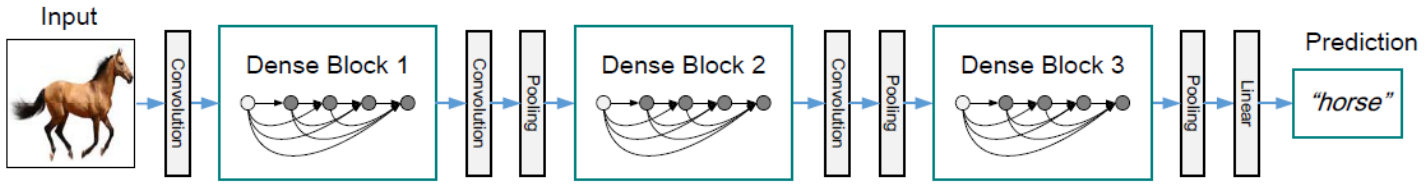


Figure 2: A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.

RNN

실험 세팅: Penn Treebank

- Base Number: 8
- $K = 400,\ m = 1$: 400개의 모델을 한번에 학습 (CPU)
- Reward: $c/(\text{validation perplexity})^2$
- Controller RNN 학습 후 learning rate, weight initialization, dropout rates, decay에 대한 grid search를 진행

실험 결과

- 기존의 가장 좋았던 알고리즘보다 perplexity 성능이 3.60이 향상됨
- Perplexity 64.0을 얻은 모델은 이전 알고리즘보다 성능도 우수하며, 2배 빠름

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

Best Model

RNN

Transfer Learning 실험

- Google Neural Machine Translation 모델에서 LSTM을 제거하고 NAS를 통해 찾은 Cell을 넣음
- 영어→독일어 번역을 기준으로 BLEU Score를 계산
- GNMT LSTM 24.1 BLEU

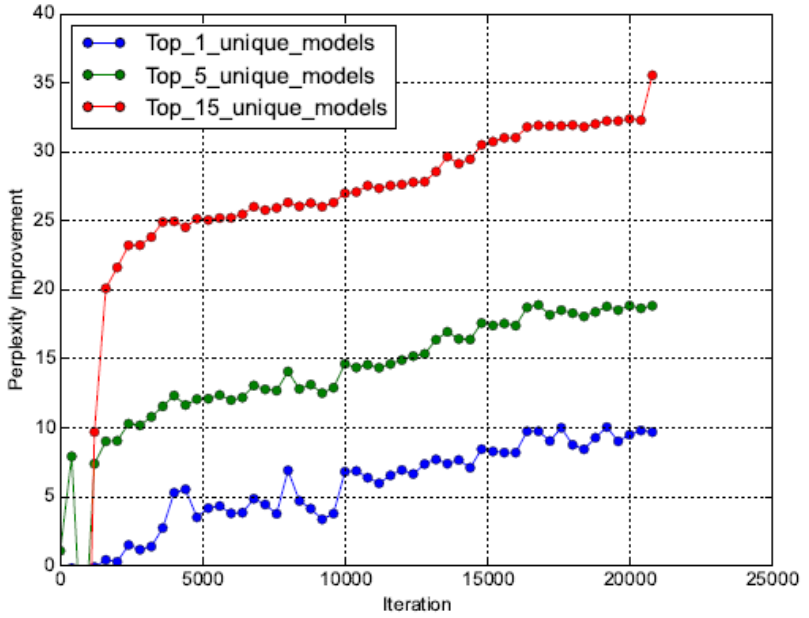
  → GNMT NAS 24.6 BLEU (0.5 증가)

Policy Gradient *vs* Random Search

- Algorithms for hyper-parameter optimization (Bengio, 2011)

  〉〉 Random Search가 생각보다 좋은 결과를 도출해내며,

  생각보다 실험적으로 이 결과보다 좋게 나오기가 어려움
- 상위 k 개의 모델의 평균 perplexity를 계산
- Policy Gradient가 Random Search보다 매우 좋은 성능을 보임



Introduction | Neural Architecture Search | Experiments

Efficient NAS

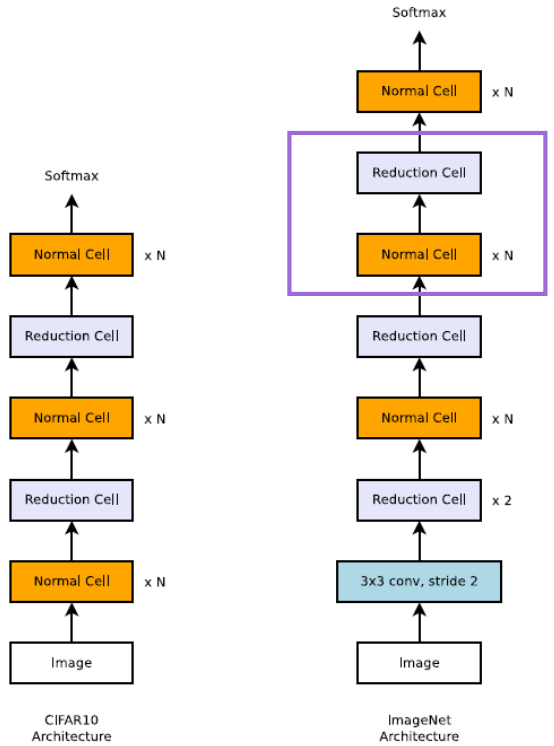
04

NASNet
Efficient NAS

NASNet

Learning Transferable Architectures for Scalable Image Recognition

- CNN의 경우에 한정하여 NAS를 효과적으로 디자인하는 것을 제안
- Search Space를 새롭게 디자인하여 모델 학습 효율이 높아지고, 다른 데이터에 사용하기에 쉬워짐 (Transferability)



Normal & Reduction Cell을
디자인하는 것이 주된 작업

❖ Normal Cell

같은 차원의 feature map을 생성하는 convolution cell

❖ Reduction Cell

높이, 너비가 1/2인 feature map을 생성하는

convolution cell: 처음 operation에 2 strides를 추가

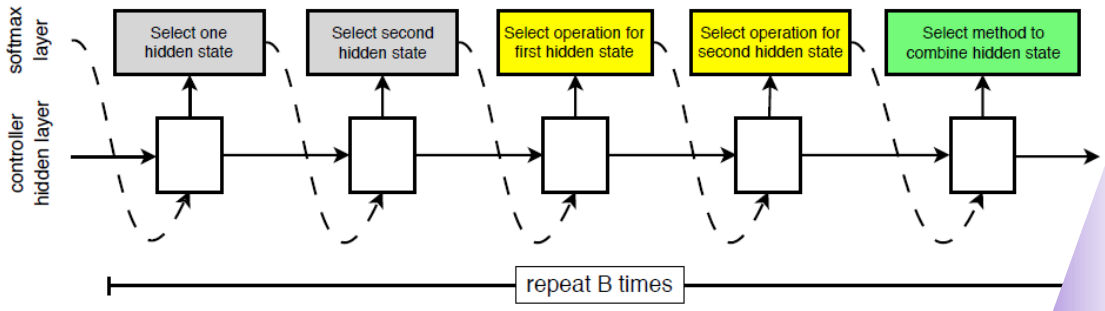
직접 네트워크 아키텍쳐를 설계

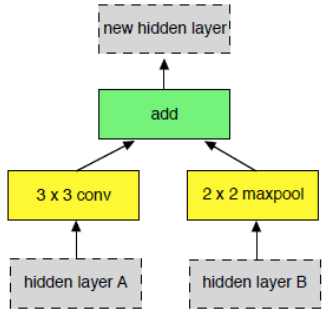
NASNet

Convolution Cell 디자인 프로세스

- Search Space: 기존에는 연산의 hyperparameter를 선택 → operation 자체를 선택
- 반복적으로 convolution cell의 블록을 설계하게 된다

RNN Controller





① 2개의 input을 선택

② Input마다 연산을 1개씩 선택

③ 연산 결과를 add or concatenate

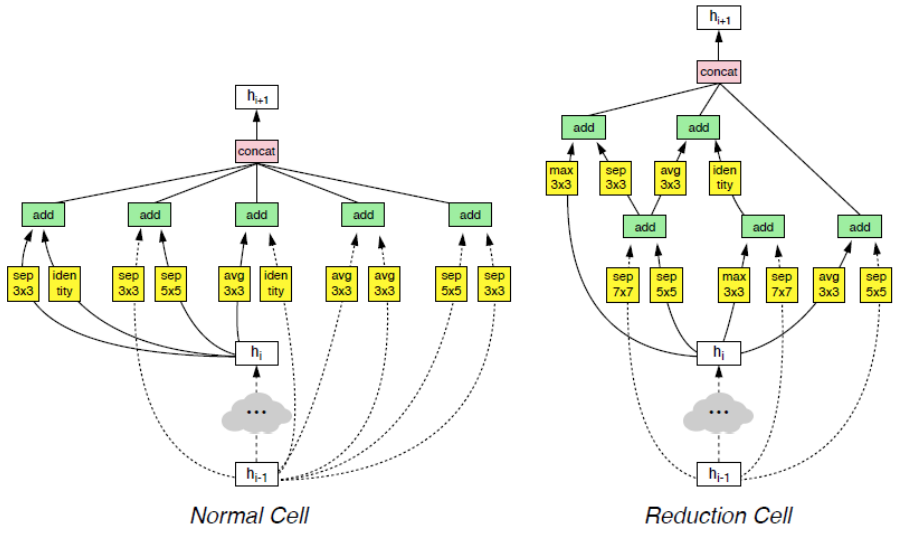
Set of Operations

- identitiy
- 1x3 then 3x1 conv
- 1x7 then 7x1 conv
- 3x3 dilated conv
- 3x3 avg.pool
- 3x3 max.pool
- 5x5 max.pool
- 7x7 max.pool
- 1x1 conv
- 3x3 conv
- 3x3 depth-sep conv
- 5x5 depth-sep conv
- 7x7 depth-sep conv

NASNet

» 주요 실험 결과

- NAS 대비 학습 시간을 7배 빠른 학습시간을 보임 (4일)
- ImageNet 실험 결과 이전 모델들에 비해 좋은 성능을 보임 (Top 1 Acc. 82.7%)



NASNet을 활용해 설계한 Cell

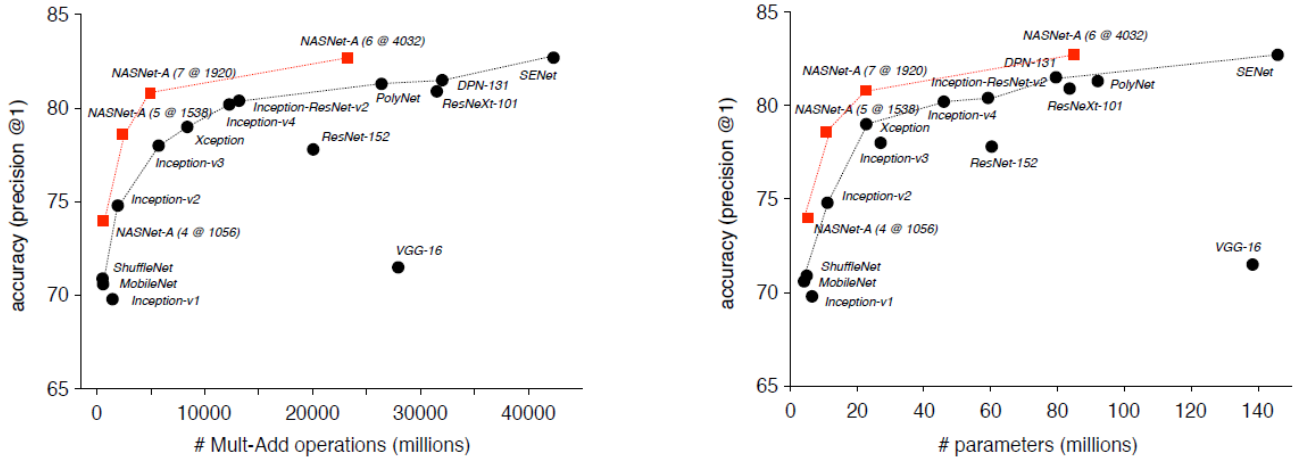


Figure 5. Accuracy versus computational demand (left) and number of parameters (right) across top performing published CNN architectures on ImageNet 2012 ILSVRC challenge prediction task. Computational demand is measured in the number of floating-point multiply-add operations to process a single image. Black circles indicate previously published results and red squares highlight our proposed models.

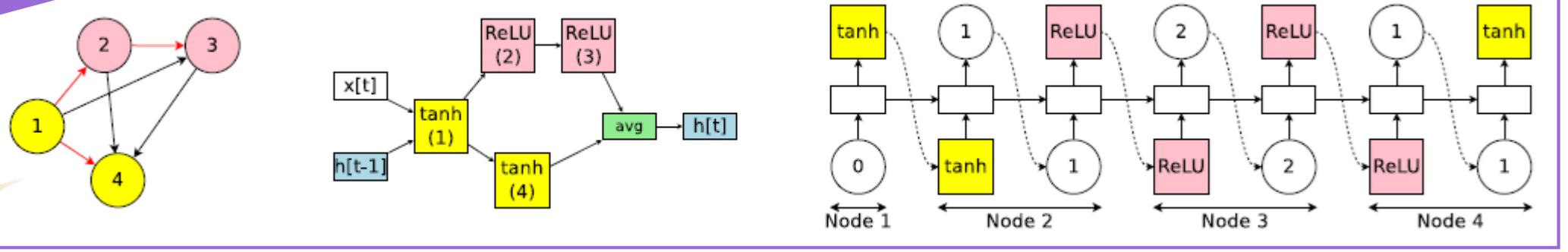
ImageNet 실험 결과

ENAS

Efficient Neural Architecture Search via Parameter Sharing

- Directed Acyclic Graph을 이용하여 네트워크를 만드는 과정을 표현
- 학습 대상인 여러 child network 구조에서 weight를 공유하게 함으로써 학습시간을 단축
- NASNet과 유사하게 Search Space를 정의하여 RNN Cell, Convolutional Architecture & Cell을 디자인

RNN



1. 노드 1의 연산을 수행
2. 노드 2와 노드 3의 연산을 수행
3. 노드 4의 연산을 수행
4. 마지막 단계에서 이전 연산 결과의 평균을 반환

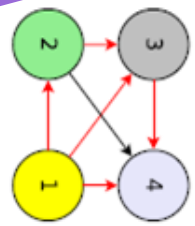
1. Node 1: input (0번째 노드)에 대하여 activation (tanh)
2. Node 2~4: 이전 노드의 index 중 하나를 선택하여 activation을 취하는 과정을 반복

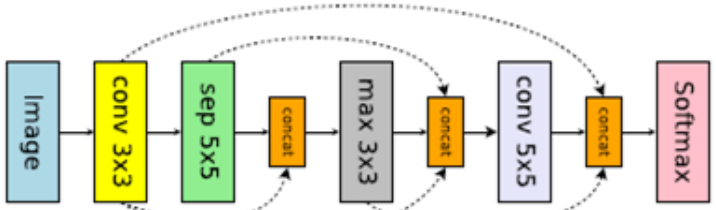
   → Node 4에서는 이전 인덱스를 1로 지정하여 (1, 4) edge가 생성됨
3. Output: 마지막 연산 결과의 단순 평균을 계산

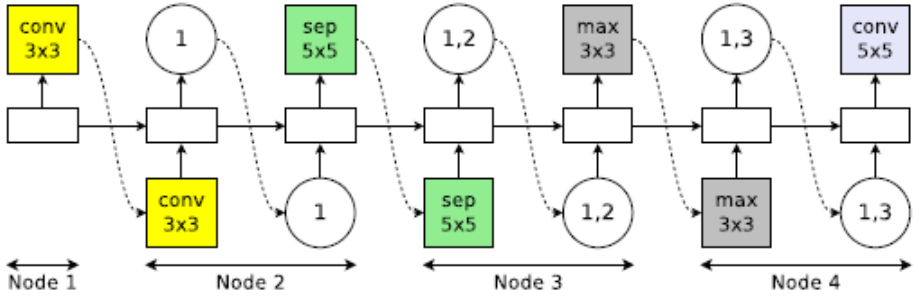
ENAS

» Efficient Neural Architecture Search via Parameter Sharing

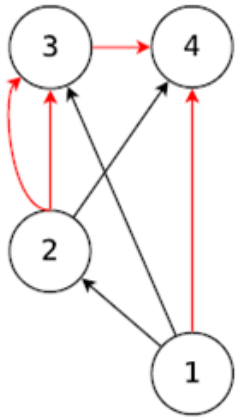
CNN – Architecture

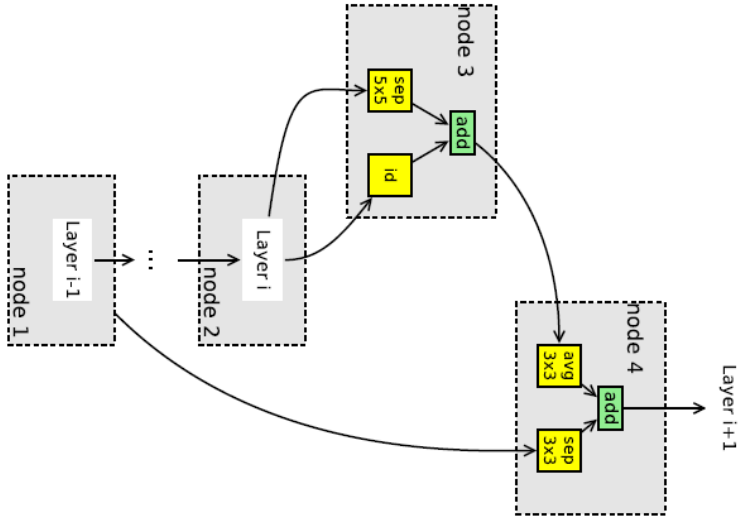


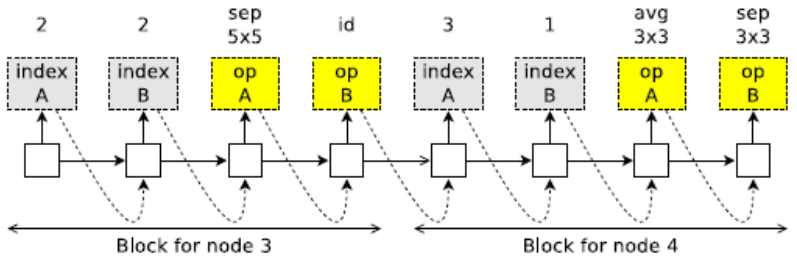




CNN – Cell







2개의 input 마다 연산을 적용하고, 이를

합치는 방법을 반복: NASNet과 유사

ENAS

Parameter Sharing

- DAG에서 Node 사이의 Weight는 공유됨
- 만약, 노드 3에서 노드 4로 연결되는 경우를 학습하면, 이는 이후에도 반복적으로 사용됨

  〉〉 학습 시간 단축에 가장 큰 영향을 미침
- Shared Parameters: child models를 minibatch로 학습하여 Stochastic Gradient Descent 방식으로 학습

  ** 실험적으로 minibatch size를 1로 하더라도 모델이 효과적으로 학습되었다

주요 실험 결과 (CIFAR-10)

- GPU 1개로 약 16시간 동안 실험을 진행

  (NAS 대비 1000배 향상)

1. Macro Search Space: 네트워크 전체를 디자인
2. Micro Search Space: Cell을 디자인하여 사람이 직접 제안한 네트워크에 삽입

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**

THANK YOU