AutoML

AutoML-Zero: Evolving Machine Learning Algorithms From Scratch Esteban Real, Chen Liang, David R. So, Quoc V. Le
Google Brain
2020.03 Preprint

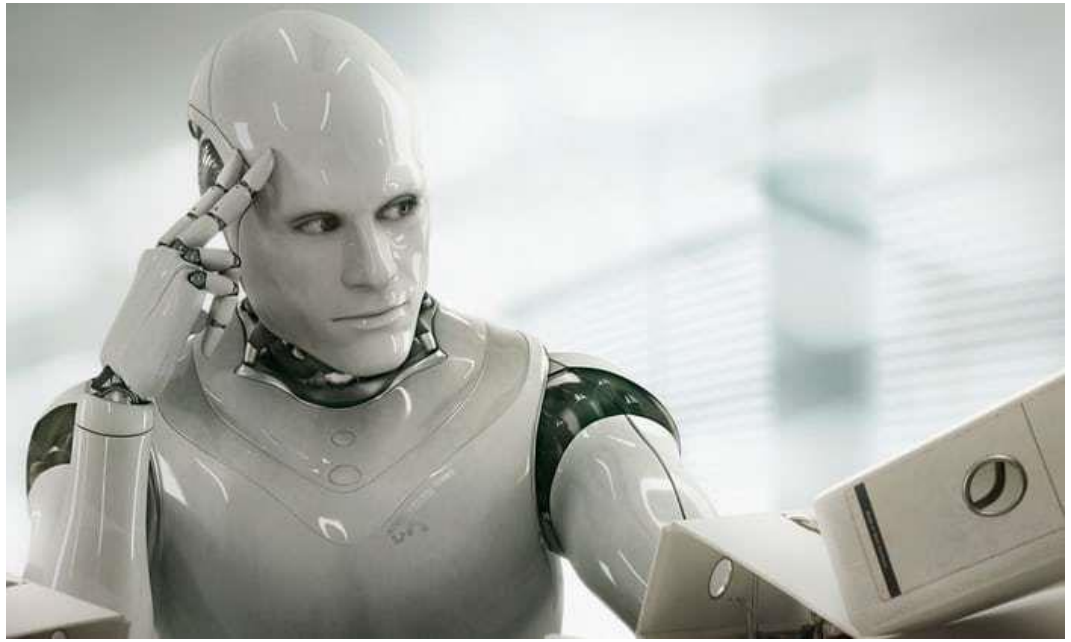
Learning Transferable Architectures for Scalable Image Recognition
Barret Zoph, Vijay Vasudevan, Jonathon Shlens, Quoc V. Le
Google Brain
2018 CVPR

석사과정 김 진용

Background

Background

AutoML?



머신러닝으로 머신러닝을 설계하는 일

AI Lab. ARTIFICIAL INTELLIGENCE LAB.

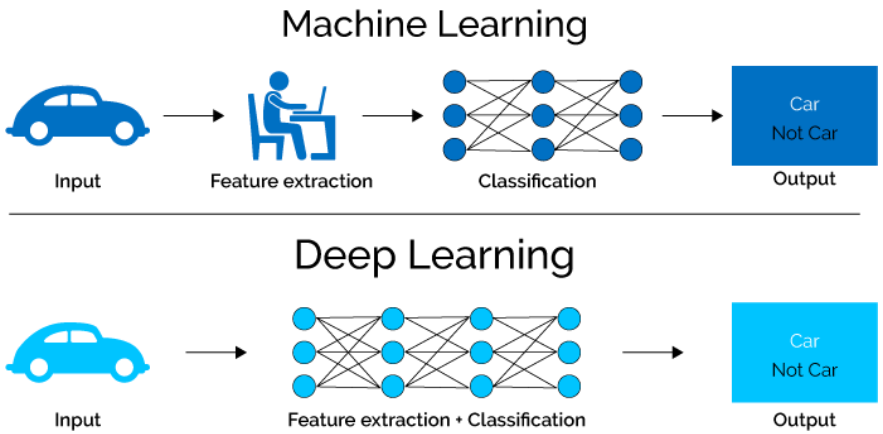
AutoML?

3가지의 방향으로 연구들이 진행되고 있음

1. Automated Feature Learning

- 유의미한 feature를 추출해 입력으로 사용하는 머신러닝에선 중요한 부분

- 하지만 딥러닝에서는 이 또한 자동으로 진행되어 중요성이 상대적으로 떨어짐



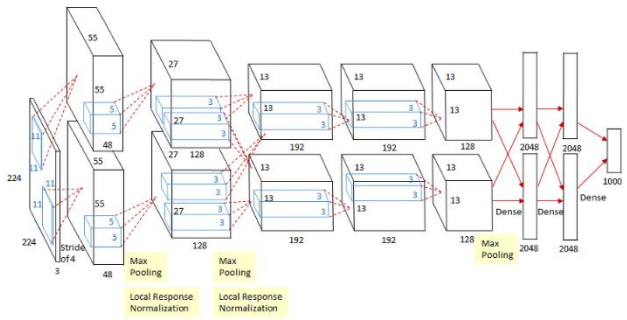
Background

AutoML?

3가지의 방향으로 연구들이 진행되고 있음

2. Architecture Search

- 기존 연구되었던 CNN (Alex,VGG,Resnet)이나 LSTM,GRU 같은 RNN 등을 구성하는 우리가 아는 그 구조
- AutoML 연구 중 가장 많이 연구되고 있고, 효과적이라고(개인피셜) 생각되는 분야
- 주로 강화학습, 유전 알고리즘 등으로 설계 된다고 함



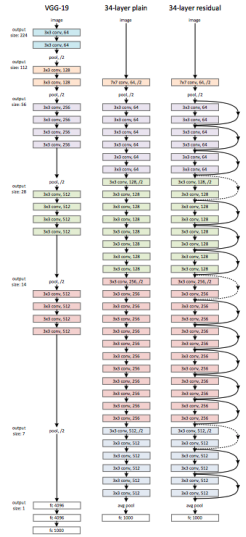
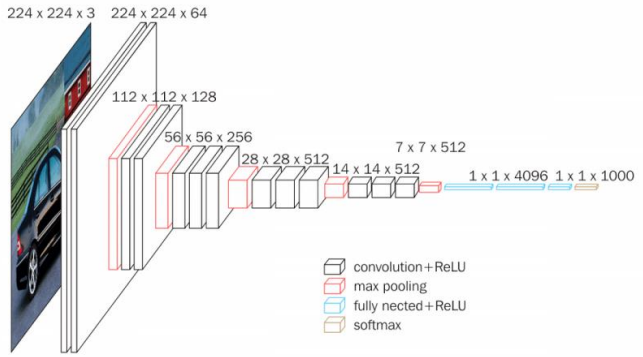


Figure 3. Example network architectures for ImageNet. **Left**: the VGG-19 model [41] (19.6 billion FLOPs) as a reference. **Middle**: a plain network with 34 parameter layers (3.6 billion FLOPs). **Right**: a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase dimensions. **Table 1** shows more details and other variants.



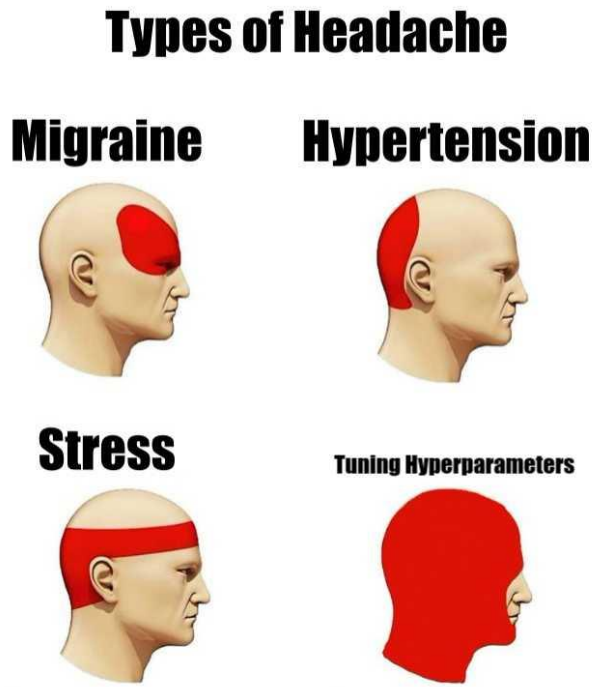
AI Lab. ARTIFICIAL INTELLIGENCE LAB.

AutoML?

3가지의 방향으로 연구들이 진행되고 있음

3. Hyperparameter Optimization

- 하이퍼 파라미터(lr, batch-size 등)들이 학습에 큰 영향을 주게되기 때문에 보통 모델을 만들더라도 튜닝(Tuning)에 굉장히 많은 시간이 듦
- 이를 효과적으로 추정하거나 선택하는데 머신러닝을 사용



NASNet

Learning Transferable Architectures for Scalable Image Recognition

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, Quoc V. Le
Google Brain

2018 CVPR

기존 방법

Neural Architectual Search(NAS) 방법

B. Zoph, Quoc V.Le "NEURAL ARCHITECTURE SEARCH WITH REINFORCEMENT LEARNING", 2017

- NAS는 두 개의 구성으로 이루어져있음
  1. RNN Controller(좌측)
  2. RNN에서 출력된 Architecture로 accuracy를 측정하고 그를 기반으로 Controller를 학습시키는 강화학습 모델

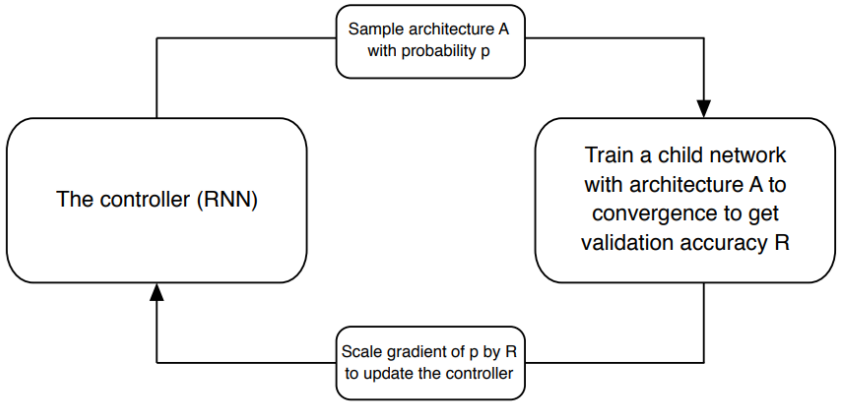


Figure 1. Overview of Neural Architecture Search [71]. A controller RNN predicts architecture A from a search space with probability p. A child network with architecture A is trained to convergence achieving accuracy R. Scale the gradients of p by R to update the RNN controller.

기존 방법

Neural Architectual Search(NAS) 방법

B. Zoph, Quoc V.Le "NEURAL ARCHITECTURE SEARCH WITH REINFORCEMENT LEARNING", 2017

- 1번의 시퀀스에 생성된 아키텍쳐를 학습해야하기 때문에 굉장히 오래걸림

800개의 GPU로 한달동안 했다고...

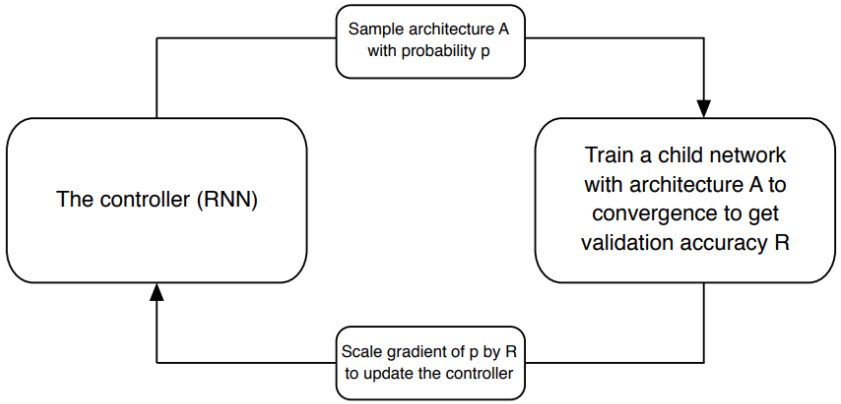


Figure 1. Overview of Neural Architecture Search [71]. A controller RNN predicts architecture A from a search space with probability p. A child network with architecture A is trained to convergence achieving accuracy R. Scale the gradients of p by R to update the RNN controller.

Proposed Method

- - 기존 방법에서는 학습 된 네트워크(Controller RNN)에서 학습된 내용을 기반으로 전체 아키텍쳐의 부분 부분(layer)을 매 시퀀스마다 다시짰다.
- - 본 논문에서는 이를 Block단위(여기서는 Cell이라 칭함)로 구분하여 Search Space를 줄였다.
- 구체적인 네트워크 정의 vs 시간
- 큰 네트워크를 생성할 수록 유리해짐

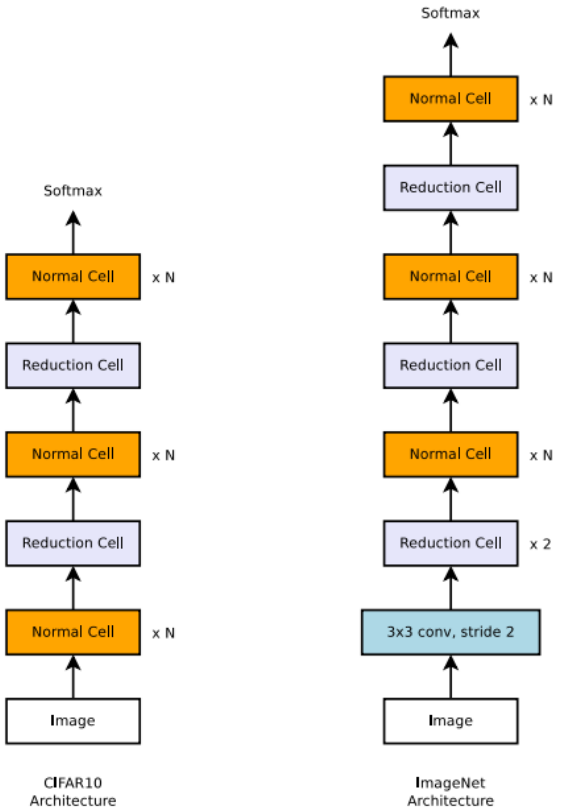
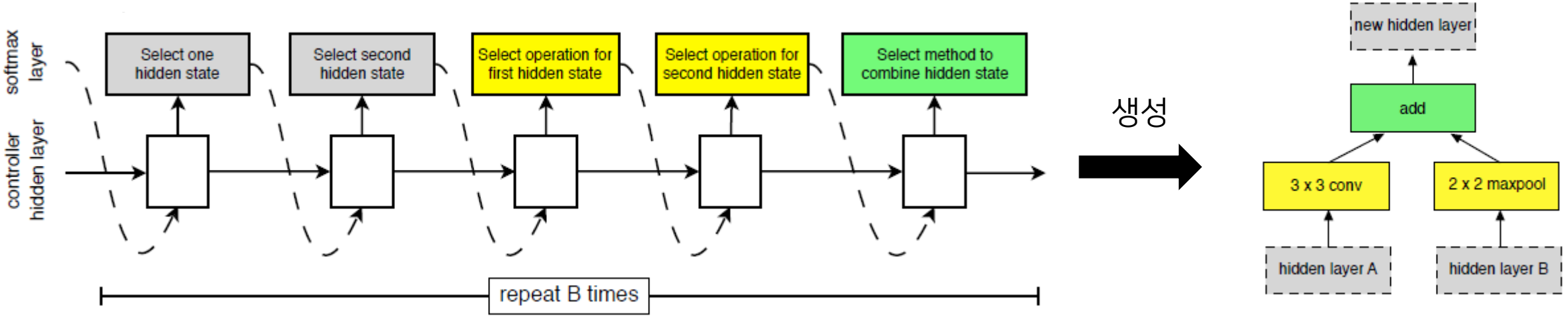


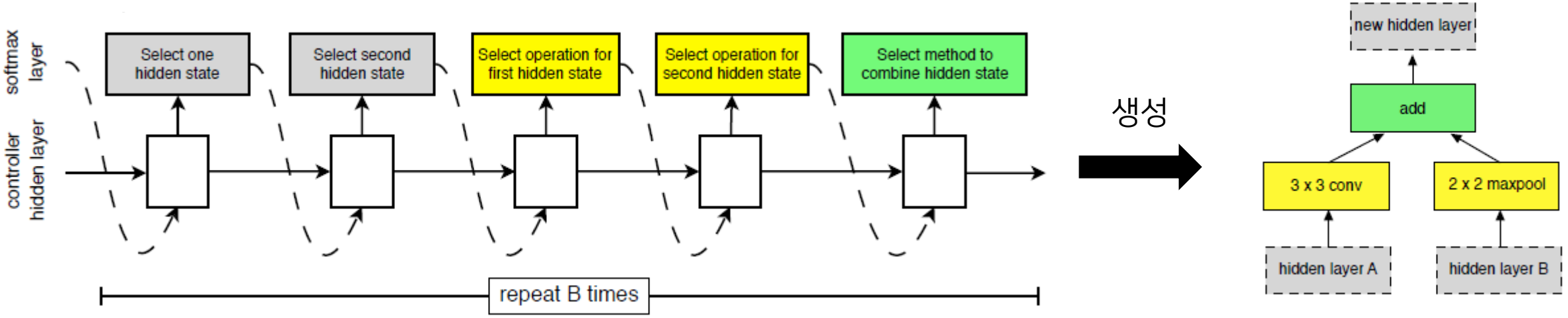
Figure 2. Scalable architectures for image classification consist of two repeated motifs termed *Normal Cell* and *Reduction Cell*. This diagram highlights the model architecture for CIFAR-10 and ImageNet. The choice for the number of times the Normal Cells that gets stacked between reduction cells, N, can vary in our experiments.

Proposed Method



- 가장 작은 단위인 Block의 생성 과정
- Block은 2개의 연산을 수행해 1개의 feature map을 출력하는 역할

Proposed Method

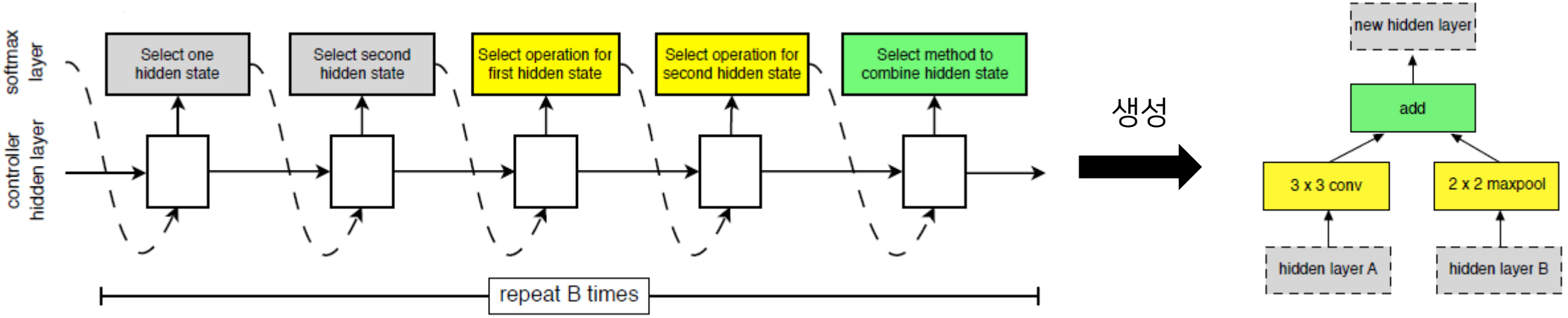


1. hidden state input 2개

input에 대한 제약을 두면서 Search Space를 줄임

- h_i
- h_{i-1}
- $h_{i,0}$
- $h_{i,1}$
- $h_{i,2}$
- $h_{i,3}$

Proposed Method

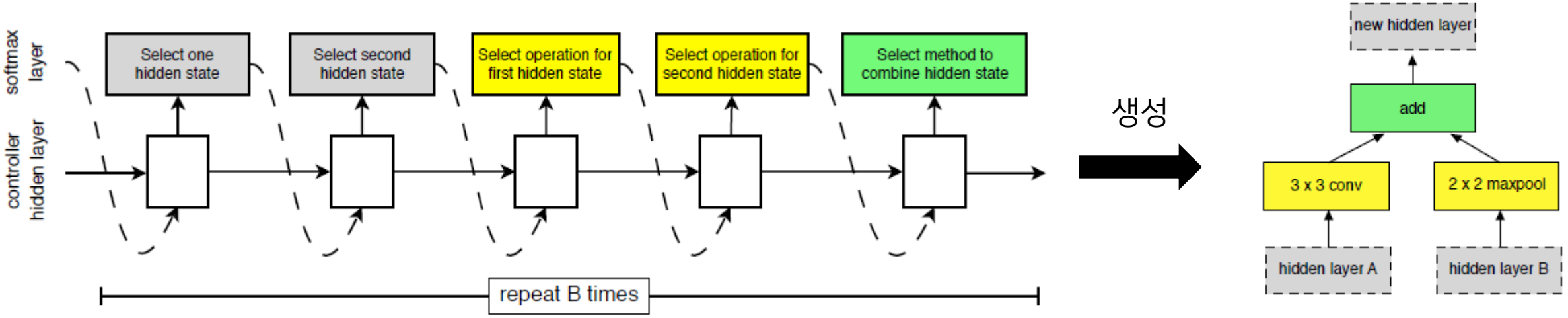


2. Operation 2개

마찬가지로 선택되는 항목을 추려놓아 Search space 줄임
기준은 성능에 따른 혹은 많이 선택되는 오퍼레이션 위주

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

Proposed Method



3. Combine Operation 1개

- **add**
- **concat**

Proposed Method

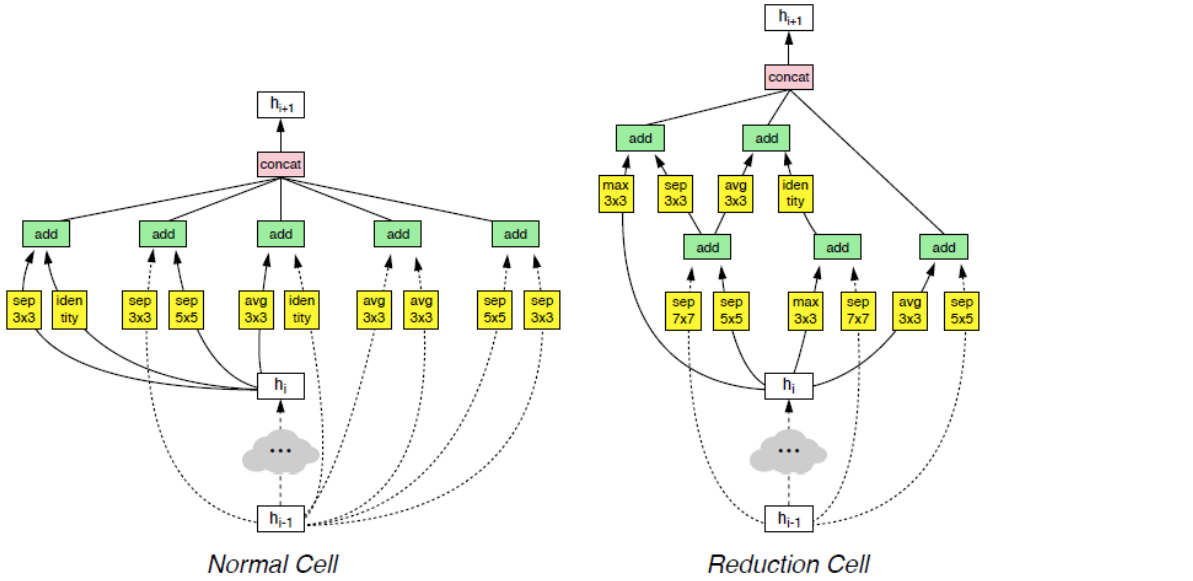


Figure 4. Architecture of the best convolutional cells (NASNet-A) with $B = 5$ blocks identified with CIFAR-10 . The input (white) is the hidden state from previous activations (or input image). The output (pink) is the result of a concatenation operation across all resulting branches. Each convolutional cell is the result of B blocks. A single block is corresponds to two primitive operations (yellow) and a combination operation (green). Note that colors correspond to operations in Figure 3.

- Block이 모여서 만들어진 Convolution Cell 단위는 2개의 종류가 있음
  - Normal Cell
  - Reduction Cell

Proposed Method

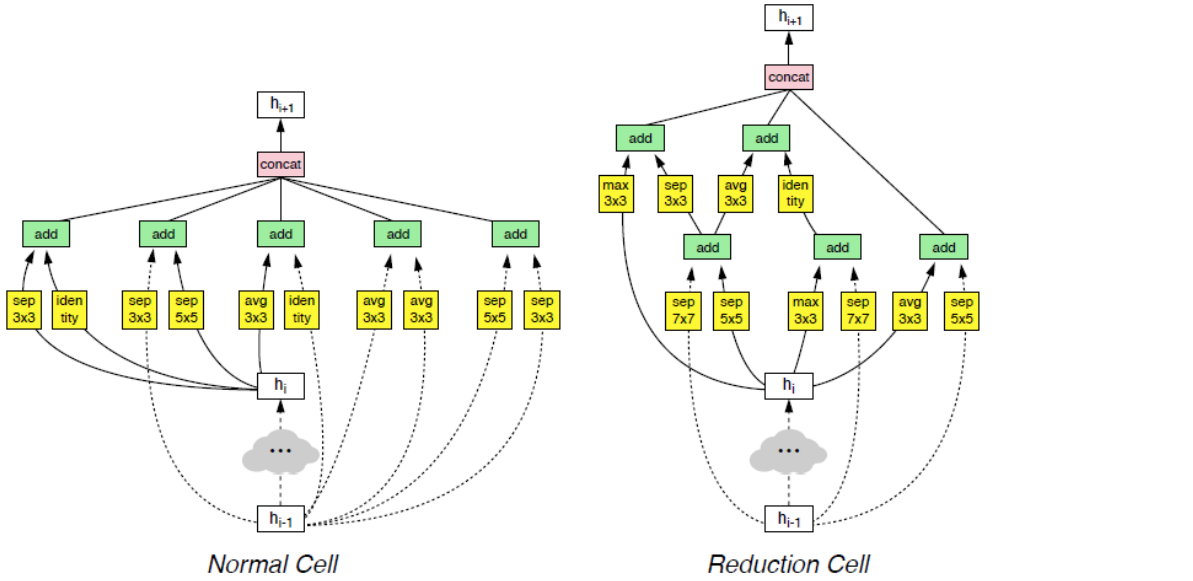


Figure 4. Architecture of the best convolutional cells (NASNet-A) with $B = 5$ blocks identified with CIFAR-10 . The input (white) is the hidden state from previous activations (or input image). The output (pink) is the result of a concatenation operation across all resulting branches. Each convolutional cell is the result of B blocks. A single block is corresponds to two primitive operations (yellow) and a combination operation (green). Note that colors correspond to operations in Figure 3.

- Normal Cell은 Feature map의 width, height가 같은 Cell
- Reduction Cell은 Feature map의 width,height가 **절반**이 되는 Cell

- 결론적으로 Block들의 Stride 차이이다.

Proposed Method

- 결론적으로는 Cell들을 합쳐 아키텍쳐를 만든다

순서 :
1. RNN Controller로 Normal Cell block, Reduction Cell block각각 B개씩 만듬
2. 이 Cell들로 Normal Cell과 Redcution Cell을 만듬
3. 순서에 맞게 배치하여 Network 구성 (오른쪽 그림)
4. Accuracy를 측정하여 reward를 통해 강화학습
5. update 후 반복

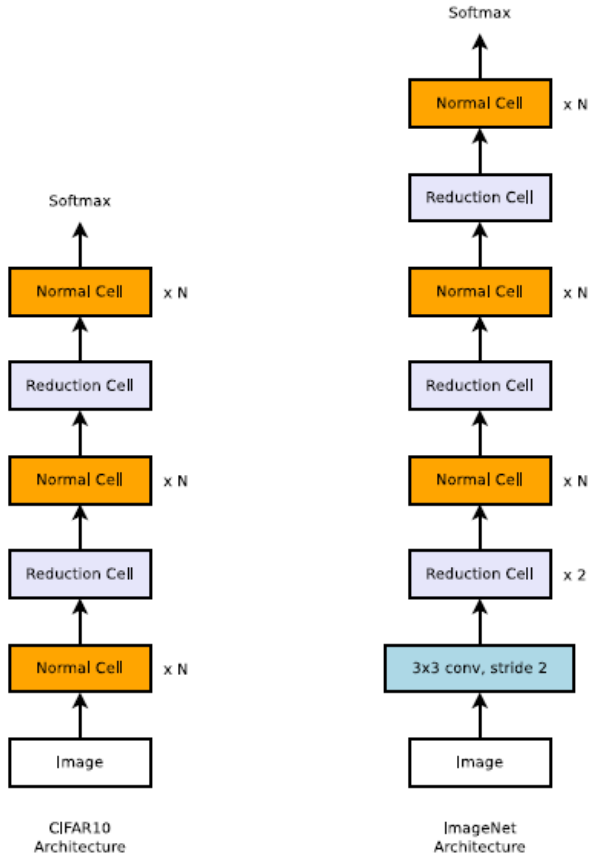
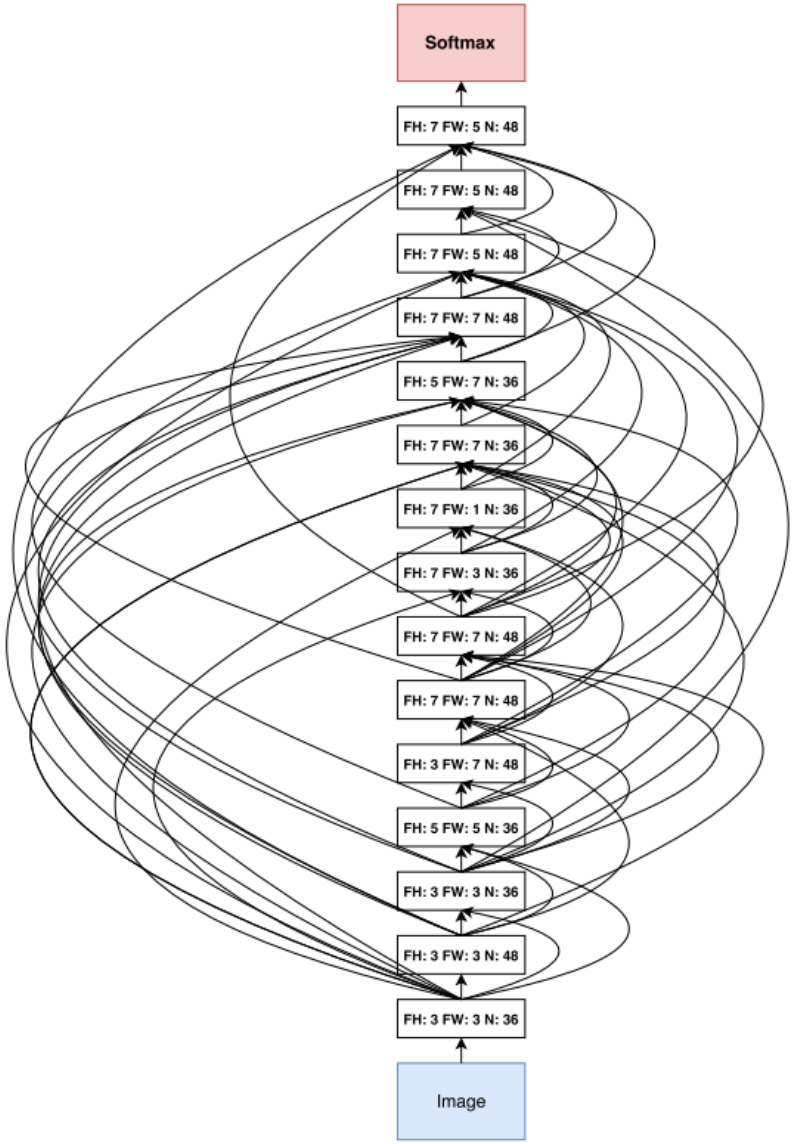
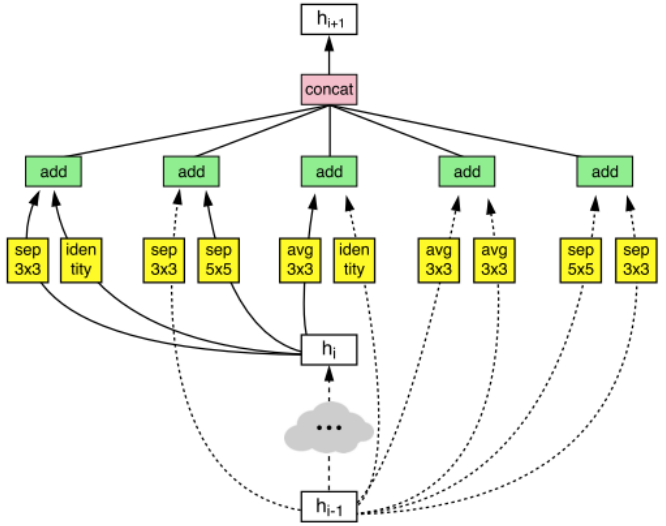


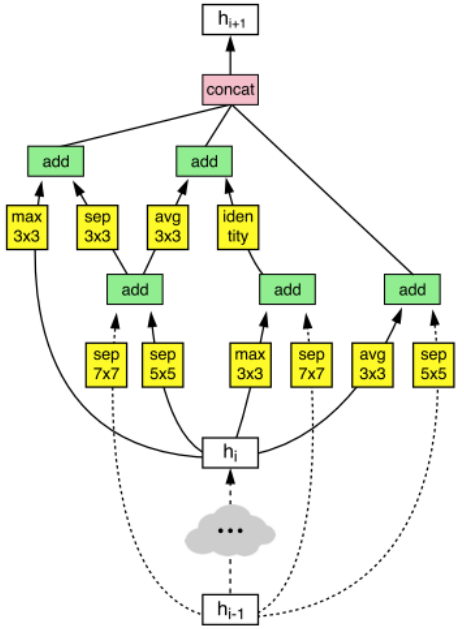
Figure 2. Scalable architectures for image classification consist of two repeated motifs termed *Normal Cell* and *Reduction Cell*. This diagram highlights the model architecture for CIFAR-10 and ImageNet. The choice for the number of times the Normal Cells that gets stacked between reduction cells, N, can vary in our experiments.

NAS vs NASNet





Normal Cell



Reduction Cell

AutoML-Zero

AutoML-Zero: Evolving Machine Learning Algorithms From Scratch Esteban Real, Chen Liang, David R. So, Quoc V. Le
Google Brain
2020.03 Preprint

Intro

본 논문의 목적은 "AutoML을 수행할 때, 사람의 간섭을 최소화 하겠다."

- 기존 Architecture search에 대한 Auto ML방식은 사람이 개입했다.
  예를들어 1x1 Conv, 3x3 Conv, max pooling등 사람이 정해둔 기준에 맞추어 만듬

- 최소한의 수학적인 연산만 정해주고 그를 통해 설계하도록 "사람의 개입이 최소화"된 모델을 만들도록

Search Space

Op ID	Code Example	Input Args: Addresses / types	Input Args: Consts.	Output Args: Address / type	Output Args: Index	Description (see caption)	
OP0	`no_op`	–	–	–	–	–	
OP1	`s2=s3+s0`	a,b / scalars	–	c / scalar	–	$s_c = s_a + s_b$	Arithmetic
OP2	`s4=s0-s1`	a,b / scalars	–	c / scalar	–	$s_c = s_a - s_b$	Arithmetic
OP3	`s8=s5*s5`	a,b / scalars	–	c / scalar	–	$s_c = s_a\, s_b$	Arithmetic
OP4	`s7=s5/s2`	a,b / scalars	–	c / scalar	–	$s_c = s_a / s_b$	Arithmetic
OP5	`s8=abs(s0)`	a / scalar	–	b / scalar	–	$s_b = \lvert s_a \rvert$	Arithmetic
OP6	`s4=1/s8`	a / scalar	–	b / scalar	–	$s_b = 1/s_a$	Arithmetic
OP7	`s5=sin(s4)`	a / scalar	–	b / scalar	–	$s_b = \sin(s_a)$	Trigonometry
OP8	`s1=cos(s4)`	a / scalar	–	b / scalar	–	$s_b = \cos(s_a)$	Trigonometry
OP9	`s3=tan(s3)`	a / scalar	–	b / scalar	–	$s_b = \tan(s_a)$	Trigonometry
OP10	`s0=arcsin(s4)`	a / scalar	–	b / scalar	–	$s_b = \arcsin(s_a)$	Trigonometry
OP11	`s2=arccos(s0)`	a / scalar	–	b / scalar	–	$s_b = \arccos(s_a)$	Trigonometry
OP12	`s4=arctan(s0)`	a / scalar	–	b / scalar	–	$s_b = \arctan(s_a)$	Trigonometry
OP13	`s1=exp(s2)`	a / scalar	–	b / scalar	–	$s_b = e^{s_a}$	Pre-Calculus
OP14	`s0=log(s3)`	a / scalar	–	b / scalar	–	$s_b = \log s_a$	Pre-Calculus
OP15	`s3=heaviside(s0)`	a / scalar	–	b / scalar	–	$s_b = \mathbb{1}_{\mathbb{R}+}(s_a)$	Pre-Calculus
OP16	`v2=heaviside(v2)`	a / vector	–	b / vector	–	$\vec{v}_b^{(i)} = \mathbb{1}_{\mathbb{R}+}(\vec{v}_a^{(i)})\ \forall i$	Pre-Calculus
OP17	`m7=heaviside(m3)`	a / matrix	–	b / matrix	–	$M_b^{(i,j)} = \mathbb{1}_{\mathbb{R}+}(M_a^{(i,j)})\ \forall i,j$	Pre-Calculus
OP18	`v1=s7*v1`	a,b / sc,vec	–	c / vector	–	$\vec{v}_c = s_a\, \vec{v}_b$	Linear Algebra
OP19	`v1=bcast(s3)`	a / scalar	–	b / vector	–	$\vec{v}_b^{(i)} = s_a \quad \forall i$	Linear Algebra
OP20	`v5=1/v7`	a / vector	–	b / vector	–	$\vec{v}_b^{(i)} = 1/\vec{v}_a^{(i)} \quad \forall i$	Linear Algebra
OP21	`s0=norm(v3)`	a / scalar	–	b / vector	–	$s_b = \lvert \vec{v}_a \rvert$	Linear Algebra
OP22	`v3=abs(v3)`	a / vector	–	b / vector	–	$\vec{v}_b^{(i)} = \lvert \vec{v}_a^{(i)} \rvert \quad \forall i$	Linear Algebra

[Table continues on the next page.]

앞서 말한 Search Space 에 대한 부분은 그림과 고등학생 수준의
연산 65가지(본 논문에선 Instruction이라 표기) 만을 이용할 것

Search Space

```
def Setup():
  # Init weights
  v1 = gaussian(0.0, 0.01)
  s2 = -1.3
def Predict(): # v0=features
  s1 = dot(v0, v1) # Prediction
def Learn(): # s0=label
  s3 = s1 / s2 # Scale prediction
  s1 = s0 + s3 # Compute error
  v2 = s1 * v0 # Gradient
  v1 = v1 + v2 # Update weights
```

Instruction을 통해 ML알고리즘을 만들어 Setup(), Predict(), Learn()을 정의하고 이것 만을 이용해 학습

- Setup() : weight 초기화
- Predict() : forward 하는 부분 모든 것을 의미 (training, validation에 모두 포함 됨)
- Learn() : training 에 관련되는 모든 것 (training에서만 포함)

Search Space

```
# (Setup, Predict, Learn) is the input ML algorithm.
# Dtrain / Dvalid is the training / validation set.
# sX/vX/mX: scalar/vector/matrix var at address X.
def Evaluate(Setup, Predict, Learn, Dtrain, Dvalid):
  # Zero-initialize all the variables (sX/vX/mX).
  initialize_memory()
  Setup() # Execute setup instructions.
  for (x, y) in Dtrain:
    v0 = x # x will now be accessible to Predict.
    Predict() # Execute prediction instructions.
    # s1 will now be used as the prediction.
    s1 = Normalize(s1) # Normalize the prediction.
    s0 = y # y will now be accessible to Learn.
    Learn() # Execute learning instructions.
  sum_loss = 0.0
  for (x, y) in Dvalid:
    v0 = x
    Predict() # Only execute Predict(), not Learn().
    s1 = Normalize(s1)
    sum_loss += Loss(y, s1)
  mean_loss = sum_loss / len(Dvalid)
  # Use validation loss to evaluate the algorithm.
  return mean_loss
```

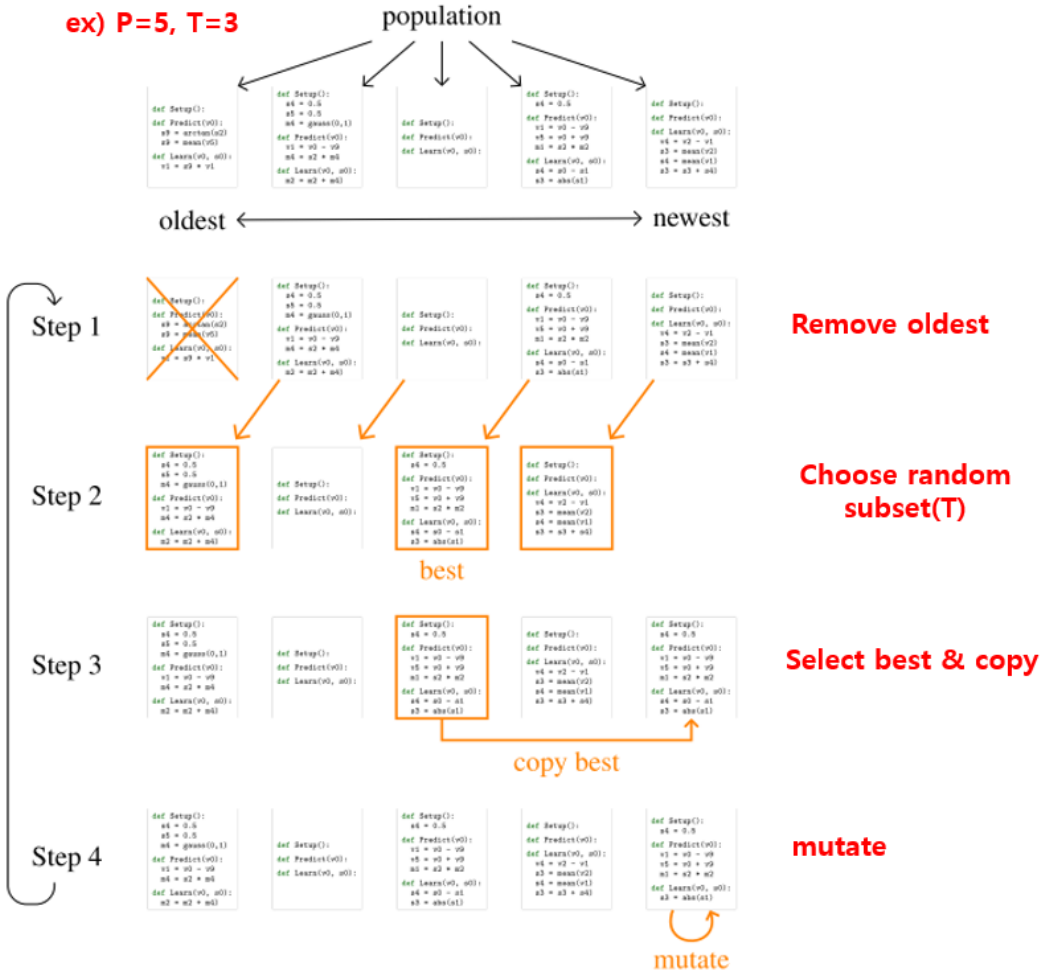
Figure 1: Algorithm evaluation on one task. We represent an algorithm as a program with three component functions (`Setup`, `Predict`, `Learn`). These are evaluated by the pseudo-code above, producing a mean loss for each task. The search method then uses the median across tasks as an indication of the algorithm's quality.

앞서 정의한 3개의 function이 완성되면 다음과 같은 코드에서 실행됨.

Normalize 같은 부분을 제외하면 개입이 최소화됨

AutoML-Zero

Search Method

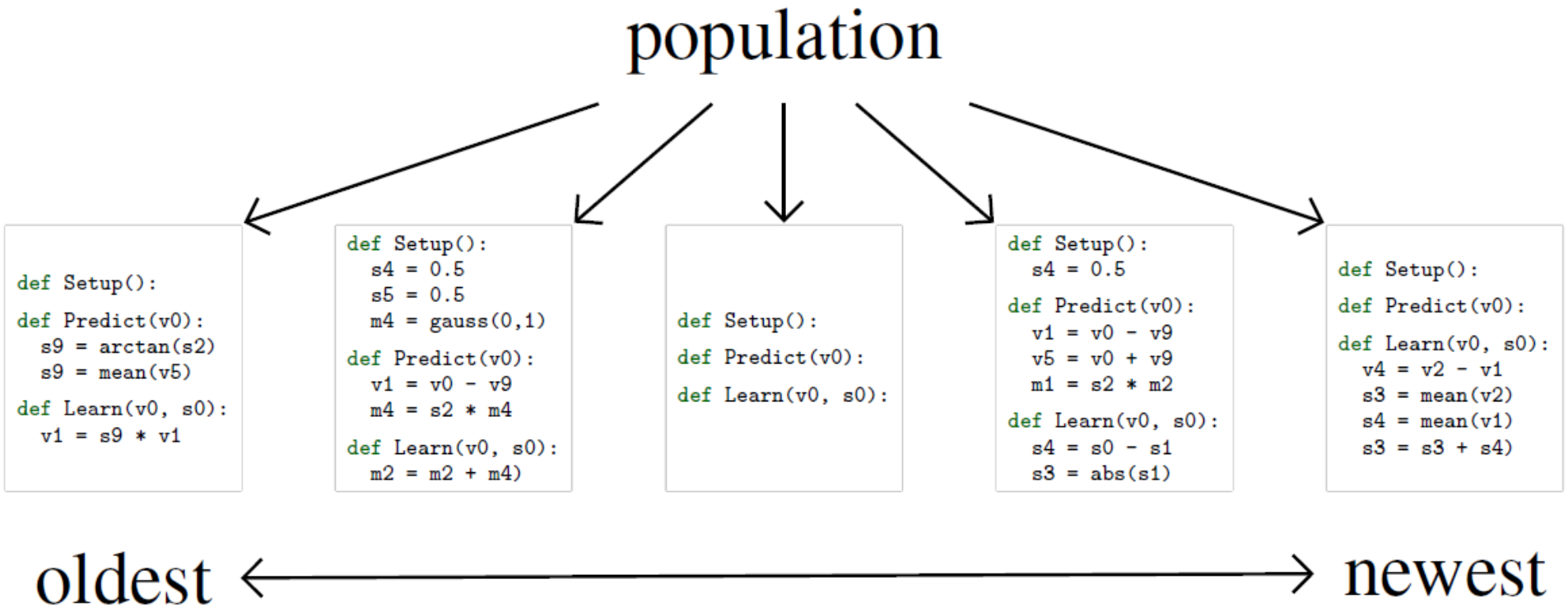


이런 function을 찾는 방법은 **진화알고리즘으로** 이루어져 있음
본 연구에 맞추어 정형화 시킨 진화알고리즘으로 4단계로 이루어짐

AI Lab. ARTIFICIAL INTELLIGENCE LAB.

AutoML-Zero

Search Method

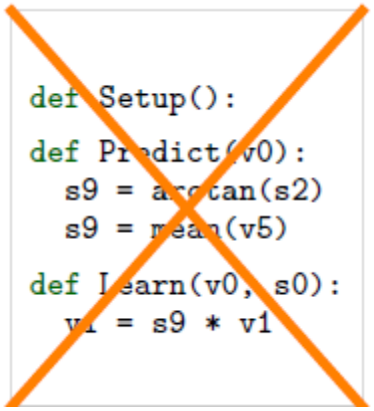


기존 방법을 오래된 순서대로 소팅

AutoML-Zero

Search Method

Step 1



```
def Setup():
def Predict(v0):
  s9 = arctan(s2)
  s9 = mean(v5)
def Learn(v0, s0):
  v1 = s9 * v1
```

```
def Setup():
  s4 = 0.5
  s5 = 0.5
  m4 = gauss(0,1)
def Predict(v0):
  v1 = v0 - v9
  m4 = s2 * m4
def Learn(v0, s0):
  m2 = m2 + m4)
```

```
def Setup():
def Predict(v0):
def Learn(v0, s0):
```

```
def Setup():
  s4 = 0.5
def Predict(v0):
  v1 = v0 - v9
  v5 = v0 + v9
  m1 = s2 * m2
def Learn(v0, s0):
  s4 = s0 - s1
  s3 = abs(s1)
```

```
def Setup():
def Predict(v0):
def Learn(v0, s0):
  v4 = v2 - v1
  s3 = mean(v2)
  s4 = mean(v1)
  s3 = s3 + s4)
```

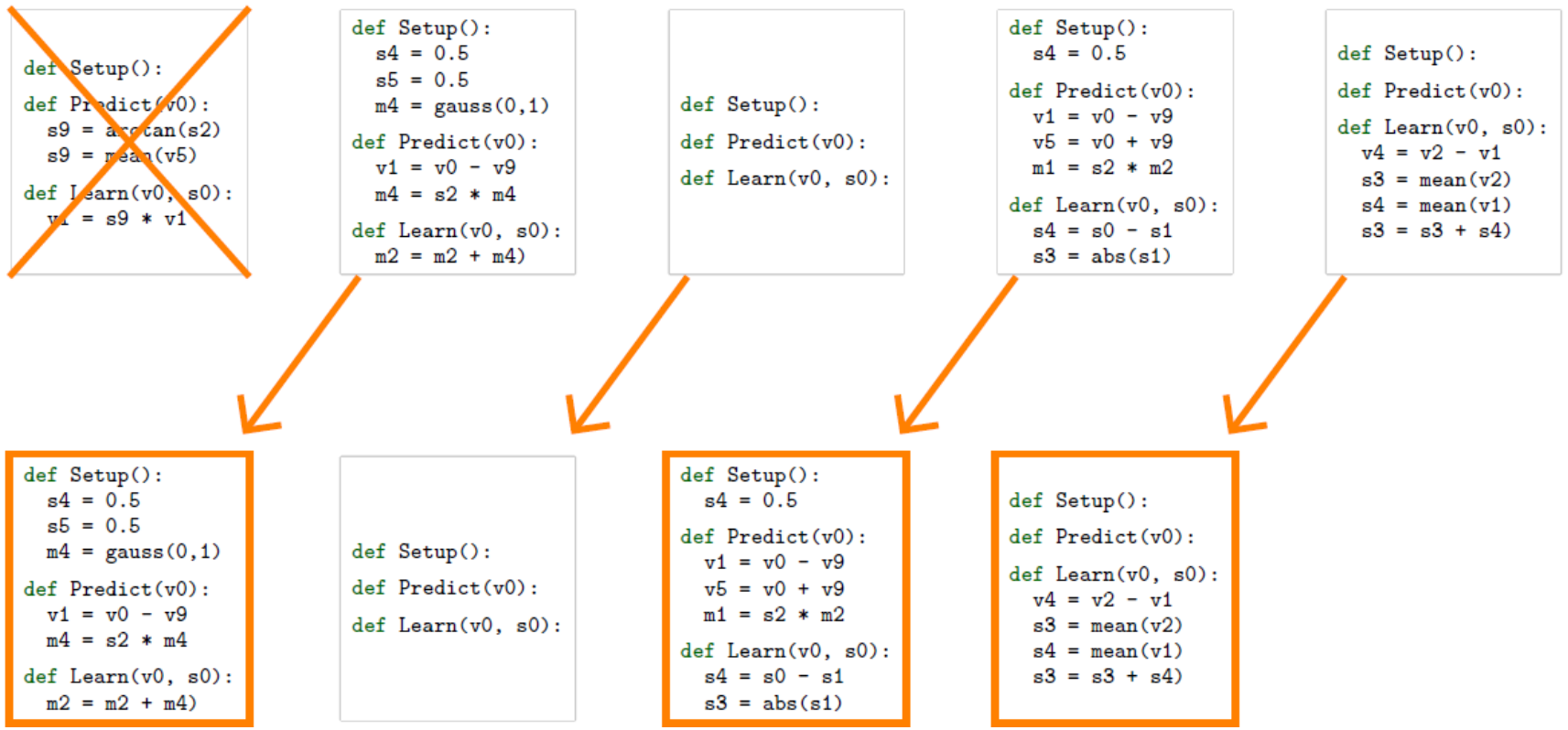
가장 오래된 것을 삭제

AI Lab. ARTIFICIAL INTELLIGENCE LAB.

AutoML-Zero

Search Method

Step 2



그 중 Random하게 T개를 샘플링 (T<P)

Search Method

Step 3

```
def Setup():
  s4 = 0.5
  s5 = 0.5
  m4 = gauss(0,1)
def Predict(v0):
  v1 = v0 - v9
  m4 = s2 * m4
def Learn(v0, s0):
  m2 = m2 + m4)
```

```
def Setup():
def Predict(v0):
def Learn(v0, s0):
```

```
def Setup():
  s4 = 0.5
def Predict(v0):
  v1 = v0 - v9
  v5 = v0 + v9
  m1 = s2 * m2
def Learn(v0, s0):
  s4 = s0 - s1
  s3 = abs(s1)
```

```
def Setup():
def Predict(v0):
def Learn(v0, s0):
  v4 = v2 - v1
  s3 = mean(v2)
  s4 = mean(v1)
  s3 = s3 + s4)
```

```
def Setup():
  s4 = 0.5
def Predict(v0):
  v1 = v0 - v9
  v5 = v0 + v9
  m1 = s2 * m2
def Learn(v0, s0):
  s4 = s0 - s1
  s3 = abs(s1)
```

copy best

그 중 가장 Best performance인 것을 선택해 복사 (새로 만듬)

AutoML-Zero

Search Method

Step 4

```
def Setup():
  s4 = 0.5
  s5 = 0.5
  m4 = gauss(0,1)
def Predict(v0):
  v1 = v0 - v9
  m4 = s2 * m4
def Learn(v0, s0):
  m2 = m2 + m4)
```

```
def Setup():
def Predict(v0):
def Learn(v0, s0):
```

```
def Setup():
  s4 = 0.5
def Predict(v0):
  v1 = v0 - v9
  v5 = v0 + v9
  m1 = s2 * m2
def Learn(v0, s0):
  s4 = s0 - s1
  s3 = abs(s1)
```

```
def Setup():
def Predict(v0):
def Learn(v0, s0):
  v4 = v2 - v1
  s3 = mean(v2)
  s4 = mean(v1)
  s3 = s3 + s4)
```

```
def Setup():
  s4 = 0.5
def Predict(v0):
  v1 = v0 - v9
  v5 = v0 + v9
  m1 = s2 * m2
def Learn(v0, s0):
  s3 = abs(s1)
```

mutate

돌연변이(Mutation)화 시켜 function 내용을 Random하게 바꿈

Search Method

Step 4

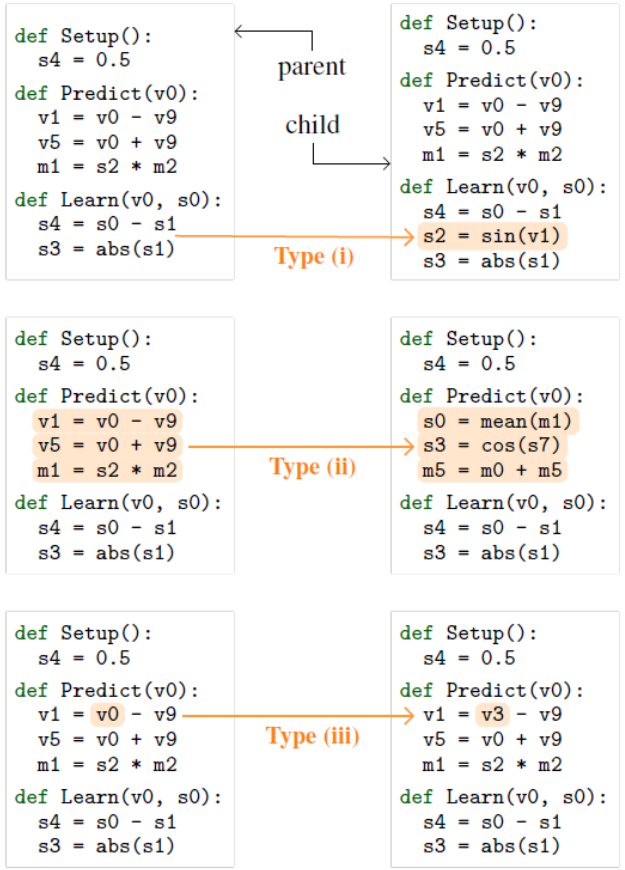


Figure 3: Mutation examples. Parent algorithm is on the left; child on the right. (i) Insert a random instruction (removal also possible). (ii) Randomize a component function. (iii) Modify an argument.

이 돌연변이 단계에선 3개의 타입 중 하나를 Random하게 실행

Type(i) : Insert or Remove random Instruction // 추가,삭제
Type(ii) : Randomize Component Function // 통으로 바꾸기
Type(iii) : Modify an argument // 소심하게 부분만 바꾸기

Overall

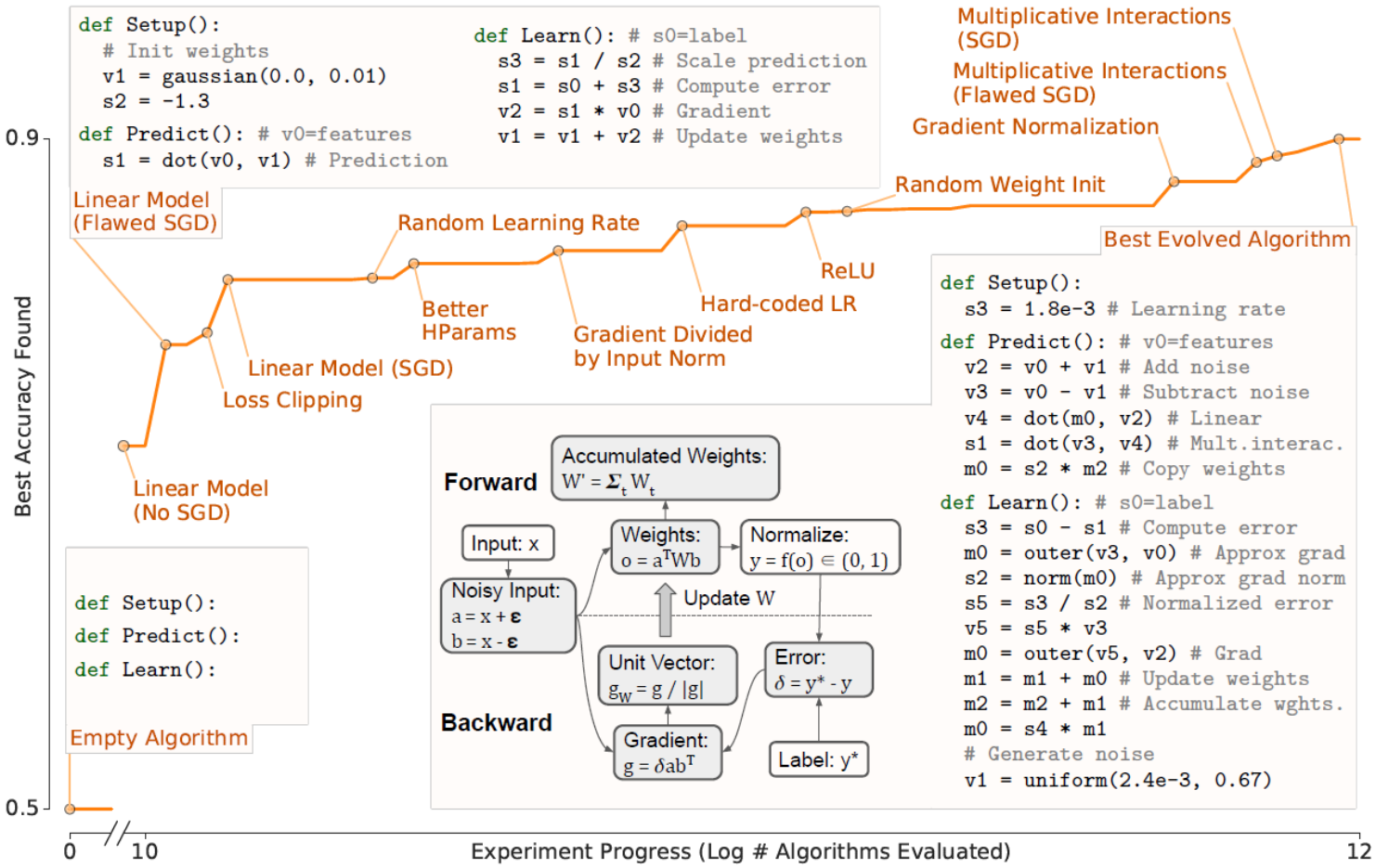


Figure 6: Progress of one evolution experiment on projected binary CIFAR-10. Callouts indicate some beneficial discoveries. We also print the code for the initial, an intermediate, and the final algorithm. The last is explained through the flow diagram. It outperforms a simple fully connected neural network on held-out test data and transfers to features 10x its size. Code notation is the same as in Figure 5.

Experiments

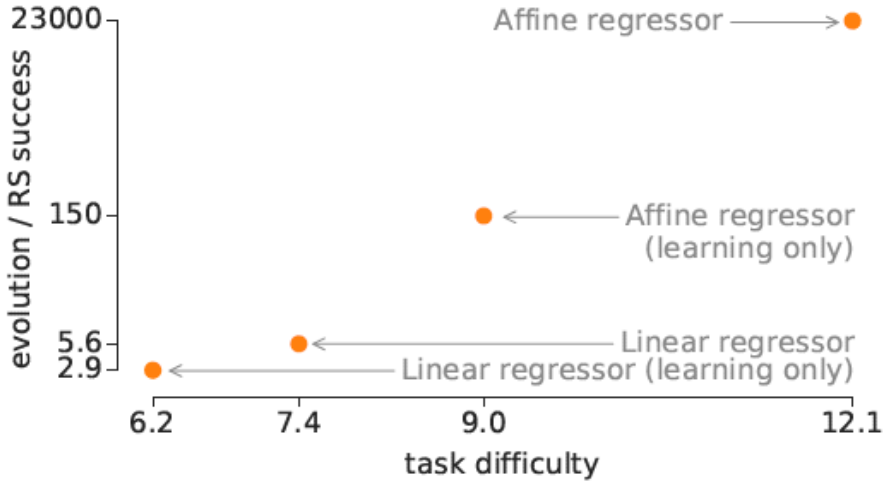


Figure 4: Relative success rate of evolution and random search (RS). Each point represents a different task type and the x-axis measures its difficulty (defined in main text). As the task type becomes more difficult, evolution vastly outperforms RS, illustrating the complexity of AutoML-Zero when compared to more traditional AutoML spaces.

Random Search 방식의 수행능력과의 비교
어려울 수록 성공의 비율(본 논문/RS)은 높아져만 간다

AutoML-Zero

Experiments

```
# sX/vX/mX = scalar/vector/matrix at address X.
# The C_ (eg C1) are constants tuned by search.
# "gaussian" produces Gaussian IID random numbers.
def Setup():
  # Initialize variables.
  m1 = gaussian(-1e-10, 9e-09) # 1st layer weights
  s3 = 4.1 # Set learning rate
  v4 = gaussian(-0.033, 0.01) # 2nd layer weights
def Predict():  # v0=features
  v6 = dot(m1, v0) # Apply 1st layer weights
  v7 = maximum(0, v6) # Apply ReLU
  s1 = dot(v7, v4) # Compute prediction
def Learn():  # s0=label
  v3 = heaviside(v6, 1.0) # ReLU gradient
  s1 = s0 - s1 # Compute error
  s2 = s1 * s3 # Scale by learning rate
  v2 = s2 * v3 # Approx. 2nd layer weight delta
  v3 = v2 * v4 # Gradient w.r.t. activations
  m0 = outer(v3, v0) # 1st layer weight delta
  m1 = m1 + m0 # Update 1st layer weights
  v4 = v2 + v4 # Update 2nd layer weights
```

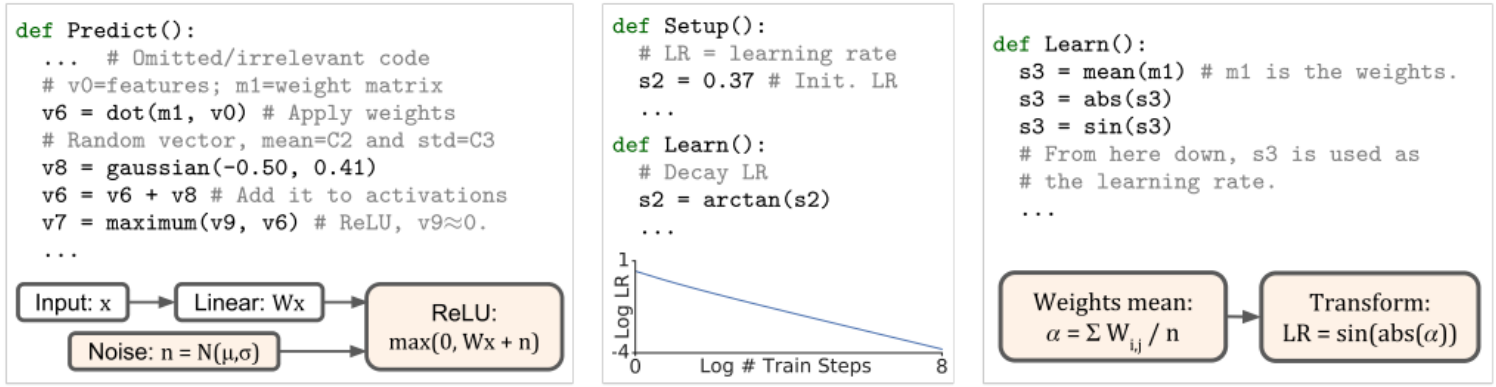
Figure 5: Rediscovered neural network algorithm. It implements backpropagation by gradient descent. Comments added manually.

- Regression 문제를 주고 확실시 하기위해 teacher network를 정해두고 실험을 실행

>> 정확히 재현하는데 성공함

- 이 과정에서 foward pass만 해서 만들지 않았고 back propagation까지 완벽하게 발명(Invent)하게 됨
- 이미지는 2-layer MLP with Gradient Descent

Experiments



(a) Adaptation to few examples.

(b) Adaptation to fast training.

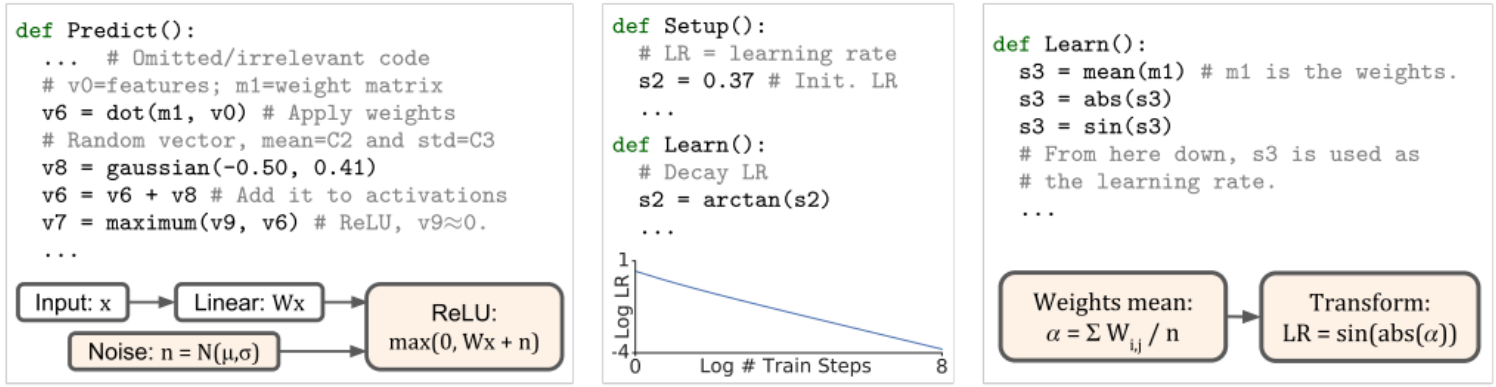
(c) Adaptation to multiple classes.

Figure 7: Adaptations to different task types. (a) When few examples are available, evolution creates a noisy ReLU. (b) When fast training is needed, we get a learning rate decay schedule implemented as an iterated arctan map (top) that is nearly exponential (bottom). (c) With multiple classes, the mean of the weight matrix is transformed and then used as the learning rate. Same notation as in Figure 5; full algorithms in Suppl. Section S5.

- 또한 상황이 바뀌었을때
  - training data가 부족
  - 빠르게 학습 해야 할 때
  - Class의 개수가 늘어났을 때

  유연하게 대처하는지에 대한 실험

Experiments



(a) Adaptation to few examples.

(b) Adaptation to fast training.

(c) Adaptation to multiple classes.

Figure 7: Adaptations to different task types. (a) When few examples are available, evolution creates a noisy ReLU. (b) When fast training is needed, we get a learning rate decay schedule implemented as an iterated arctan map (top) that is nearly exponential (bottom). (c) With multiple classes, the mean of the weight matrix is transformed and then used as the learning rate. Same notation as in Figure 5; full algorithms in Suppl. Section S5.

- Training data가 부족한 경우 Noise를 직접 주입해서 dataset을 augmentation 함
- 빠른 학습을 위해 Learning rate decay (learning rate 크게잡고 줄여가는 방식)을 배워 스스로 바꿈
- Class가 늘어난 경우 알 수 없는 방법을 썼지만 performance는 높아짐