Evolving Machine Learning Algorithms

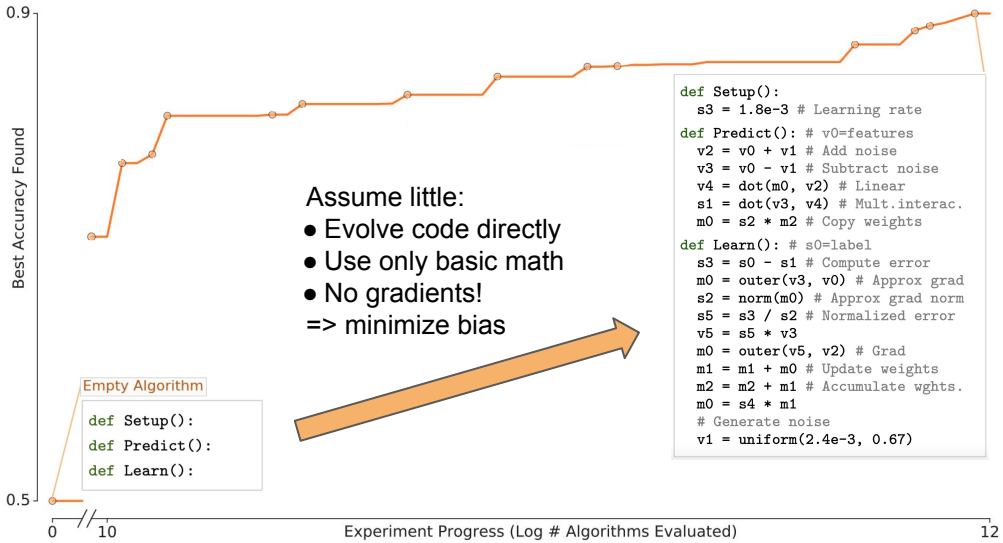
Esteban Real - ereal@google.com

Google Research / *Brain Team*

"An evolving population of algorithms"

This talk is about evolution in AutoML.
Background → SOTA → AutoML-Zero

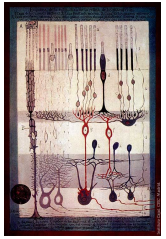
Automatically search for programs that learn (e.g. image classifiers), starting from empty code.



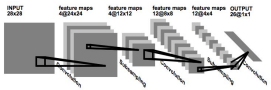
Discover full algorithms
- Simple feed-forward NNs
- Bilinear models
- Gradient descent
- LR decay
- Weight averaging
- Dropout-like regularization
- Etc.

Motivation

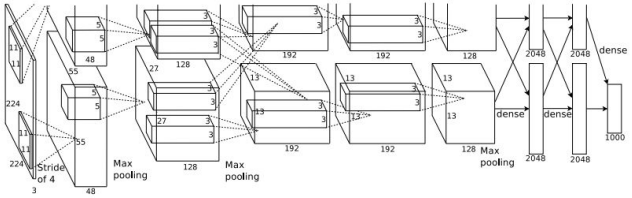
Drawing by Ramon y Cajal c. 1900, "Structure of the Mammalian Retina"



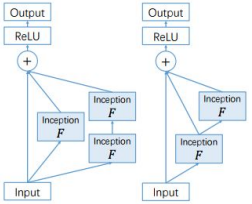
Evolution -541



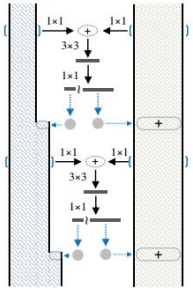
LeCun and Bengio 1995. Convolutional networks for images, speech, and time-series.



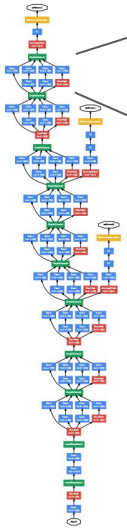
Krizhevsky et al 2012. Imagenet classification with deep convolutional neural networks.

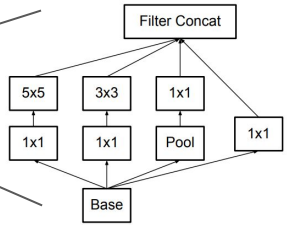


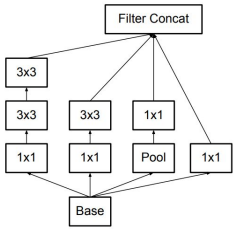
Zhang et al. 2017. PolyNet: the pursuit of structural diversity in very deep networks

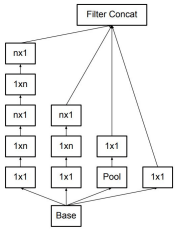


Chen et al. 2017. Dual path networks.



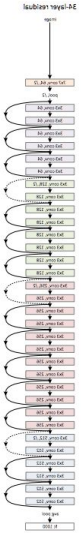


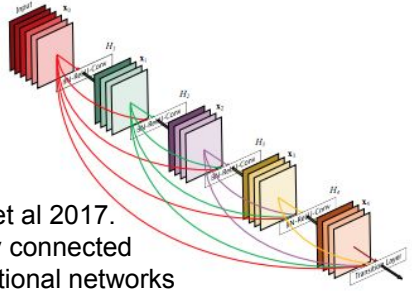




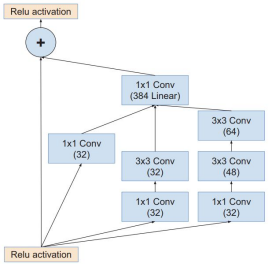
Szegedy et al 2016. Rethinking the inception architecture for computer vision

Szegedy et al. 2015. Going deeper with convolutions.

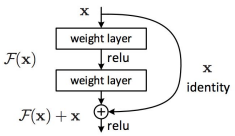




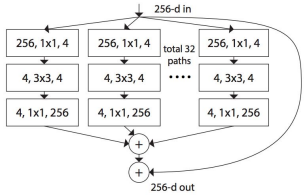
Huang et al 2017. Densely connected convolutional networks



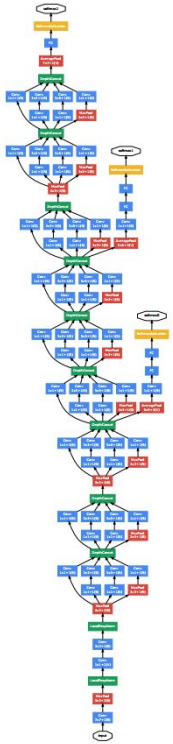
Szegedy et al 2017 Inception-v4, Inception-Resnet, and the Impact of Residual Connections on Learning.



He et al. 2016. Deep residual learning for image recognition.



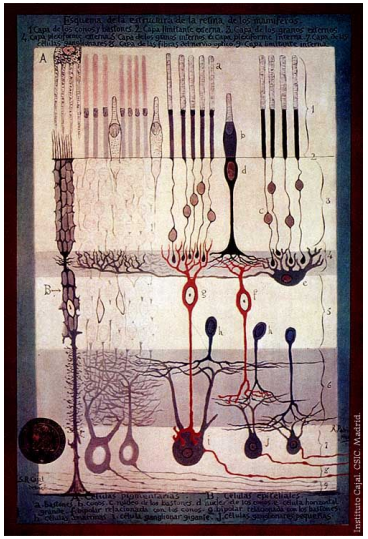
Xie et al 2017. Aggregated residual transformations for deep neural networks



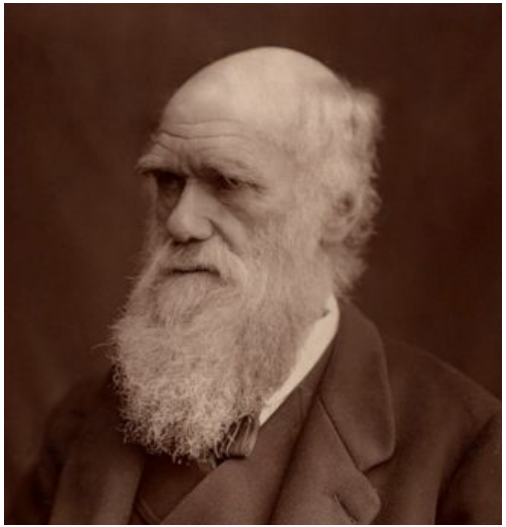
Szegedy et al. 2015.
Going deeper with convolutions.

Why AutoML?

- ML requires rare expertise
- Automate tedious tuning
- Find better models

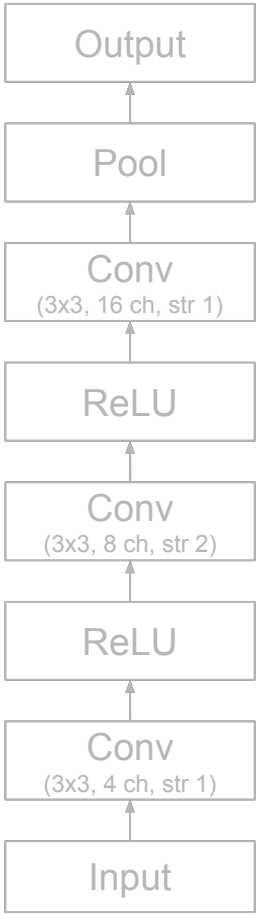


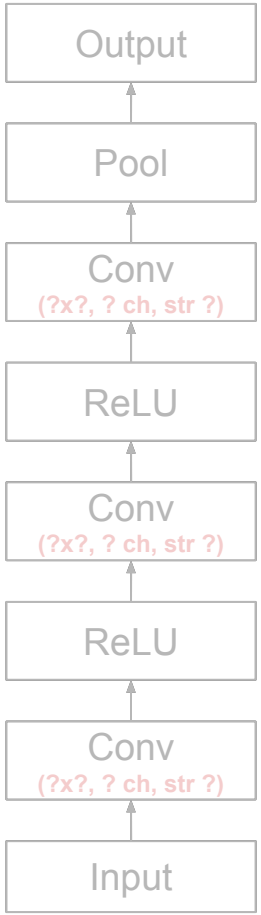
Drawing by Ramon y Cajal c. 1900,
"Structure of the Mammalian Retina"

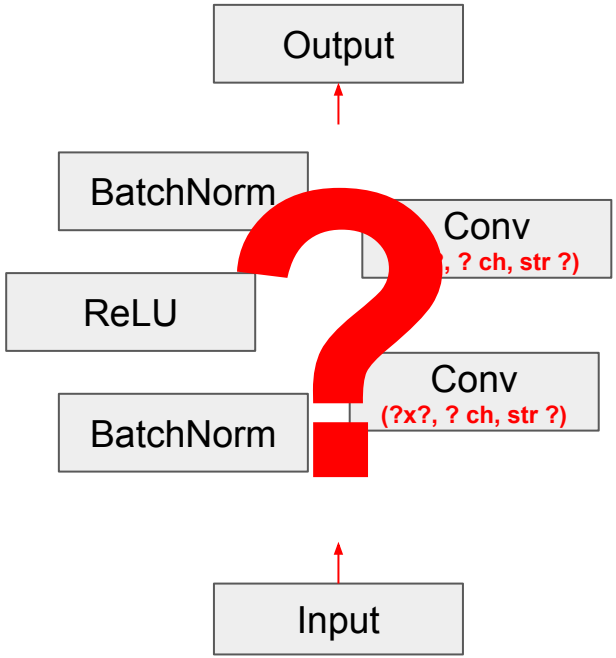


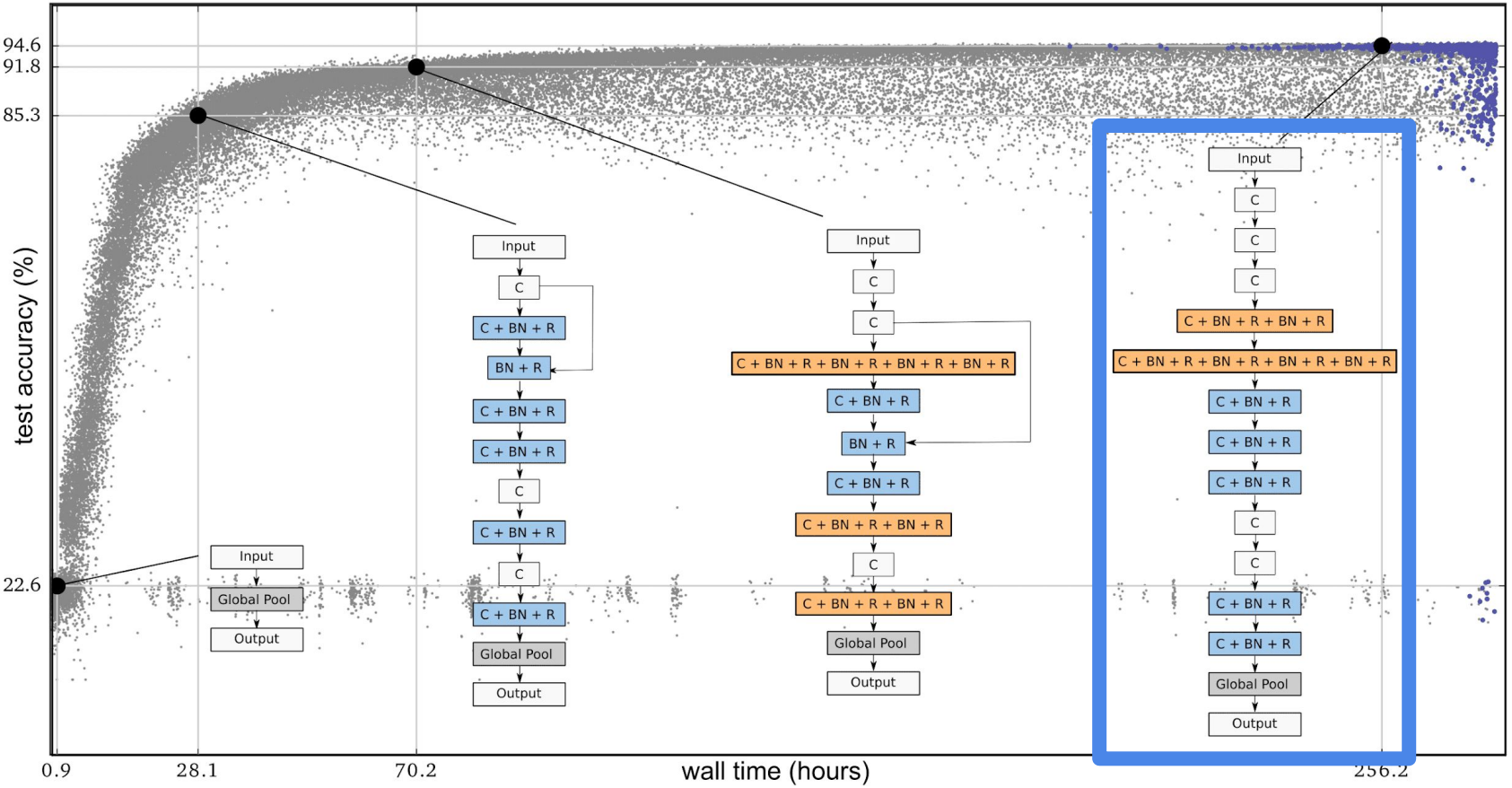
"[Mr W. D. Fry] has never seen or heard of a blueish-gray cat which was not deaf. May I quote you? [...] Might I say that you have seen or heard of as many as 6?"

-- C. Darwin to W. D. Fox, 1856


Output
Pool
Conv
(3x3, 16 ch, str 1)
ReLU
Conv
(3x3, 8 ch, str 2)
ReLU
Conv
(3x3, 4 ch, str 1)
Input


Output
Pool
Conv
(?x?, ? ch, str ?)
ReLU
Conv
(?x?, ? ch, str ?)
ReLU
Conv
(?x?, ? ch, str ?)
Input


Output
BatchNorm
Conv
, ? ch, str ?)
ReLU
Conv
(?x?, ? ch, str ?)
BatchNorm
Input



Large Scale Evolution of Image Classifiers
Real, Moore, Selle, Saxena, Suematsu, Tan, Le, and Kurakin (ICML 2017)

Related work: Evolving deep neural networks. Miikkulainen et al. 2019.

History

Neuro-Evolution

- Early work evolved the weights (Miller et al. 1989, for example).
- NEAT algorithm (Stanley & Miikkulainen 2002).
- Indirect Encodings
  - Gruau 1993
  - Stanley 2007, Stanley et al. 2009, Pugh & Stanley 2013
  - Kim & Rigazzio 2015, Fernando et al. 2016
- Backprop + Evolution (Stanley et al. 2009, Breuel & Shafait 2010)
- Weight inheritance (Fernando 2016)

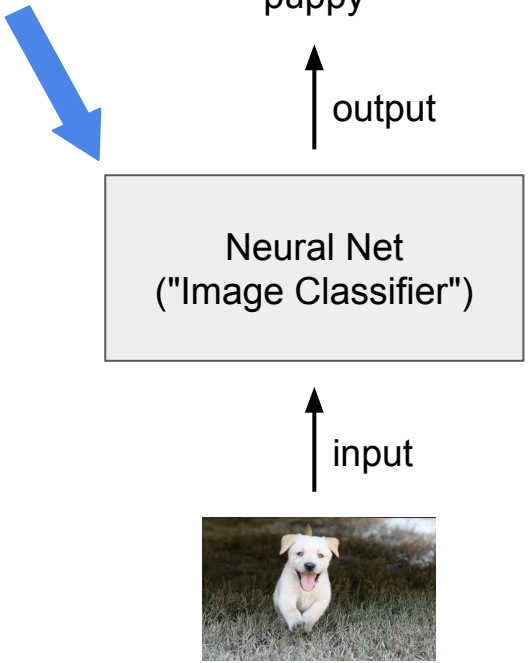
Alternative AutoML Approaches

- Grid search and random search
- Q-Learning (Baker et al. 2016)
- REINFORCE (Zoph and Le 2016)

Reaching the State of the Art

Image Classification

Discover this



IMAGENET

- most popular benchmark
- 1M images
- 1k classes
- tends to generalize


"upright piano"


"cow"


"kidney bean"


"Roman building"


"sun tea"

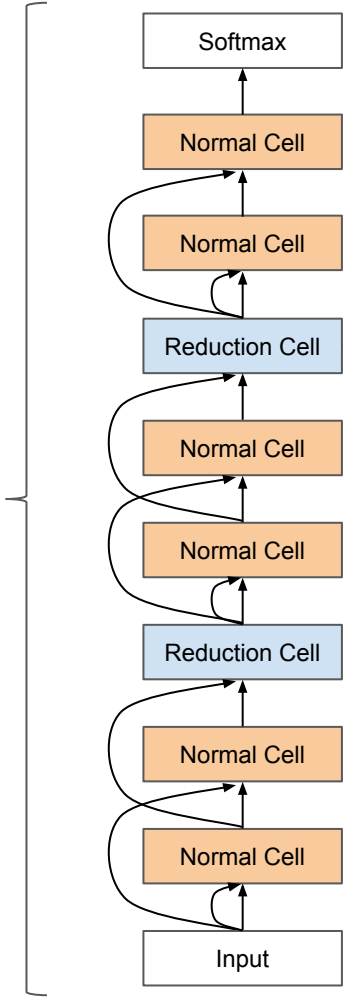

"car racing"

Model	image size	# parameters	Mult-Adds	Top 1 Acc. (%)	Top 5 Acc. (%)
Inception V2 [29]	224×224	11.2 M	1.94 B	74.8	92.2
NASNet-A (5 @ 1538)	**299×299**	**10.9 M**	**2.35 B**	**78.6**	**94.2**
Inception V3 [60]	299×299	23.8 M	5.72 B	78.8	94.4
Xception [9]	299×299	22.8 M	8.38 B	79.0	94.5
Inception ResNet V2 [58]	299×299	55.8 M	13.2 B	80.1	95.1
NASNet-A (7 @ 1920)	**299×299**	**22.6 M**	**4.93 B**	**80.8**	**95.3**
ResNeXt-101 (64 x 4d) [68]	320×320	83.6 M	31.5 B	80.9	95.6
PolyNet [69]	331×331	92 M	34.7 B	81.3	95.8
DPN-131 [8]	320×320	79.5 M	32.0 B	81.5	95.8
SENet [25]	**320×320**	**145.8 M**	**42.3 B**	**82.7**	**96.2**
NASNet-A (6 @ 4032)	**331×331**	**88.9 M**	**23.8 B**	**82.7**	**96.2**

Learning Transferable Architectures for Scalable Image Recognition.

Barret Zoph, Vijay Vasudevan, Jon Shlens, and Quoc V. Le (CVPR 2018)

Fixed outer skeleton

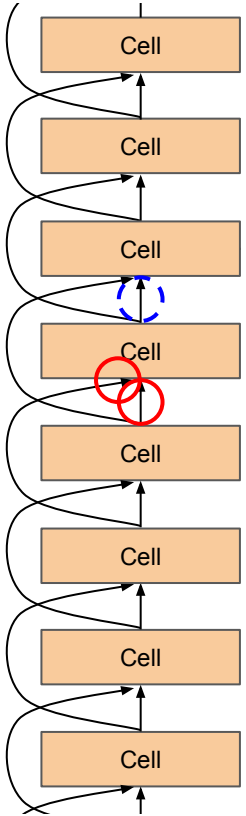


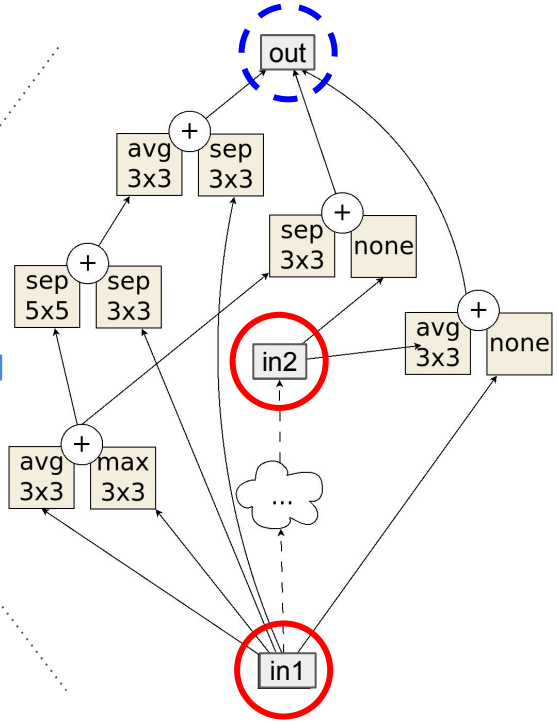
"NASNet search space"

Learning Transferable Architectures for Scalable Image Recognition.

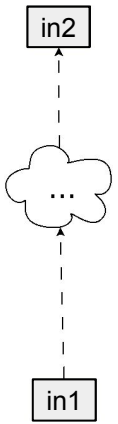
Barret Zoph, Vijay Vasudevan, Jon Shlens, and Quoc V. Le
(CVPR 2018)

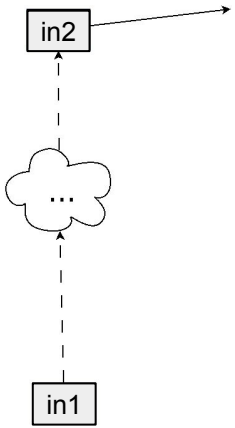
(Simplified)
Fixed
outer
skeleton

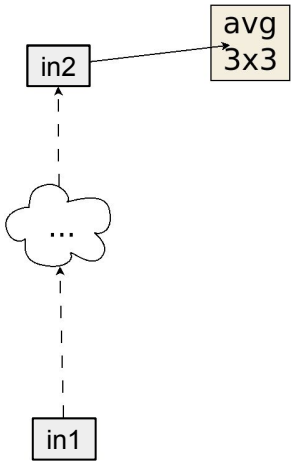


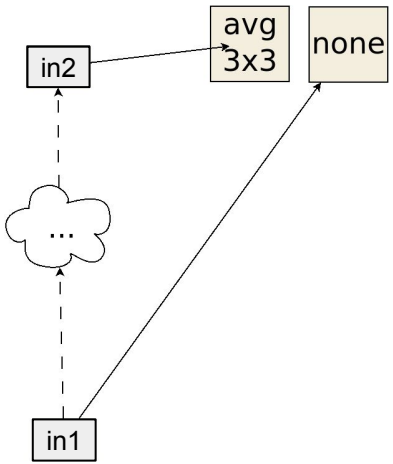


Searchable
inner
"cell"

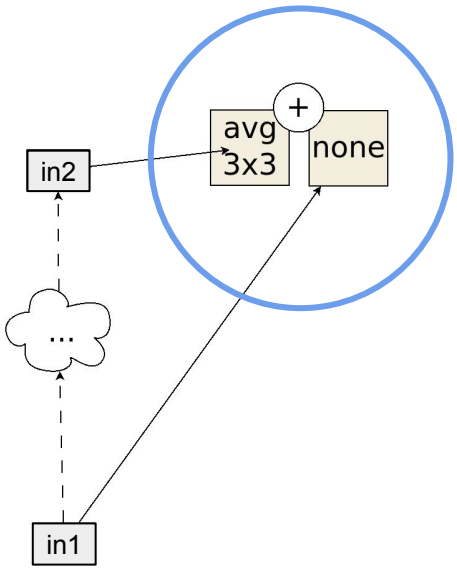

in2
...
in1

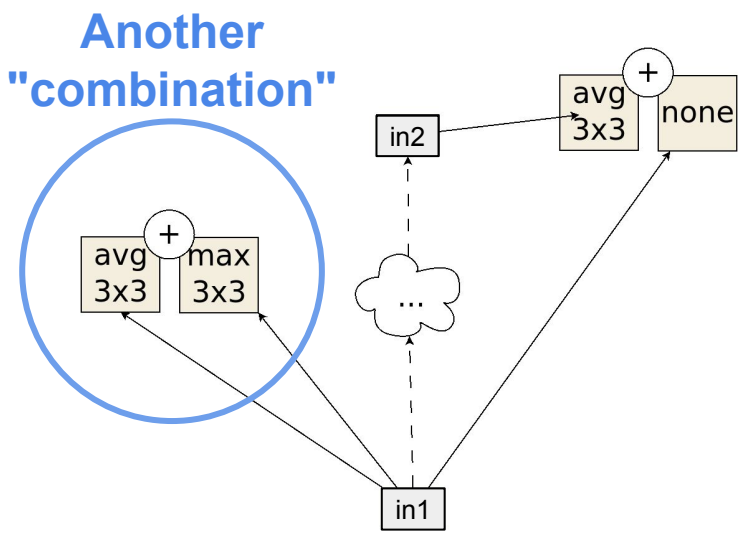

in2
...
in1

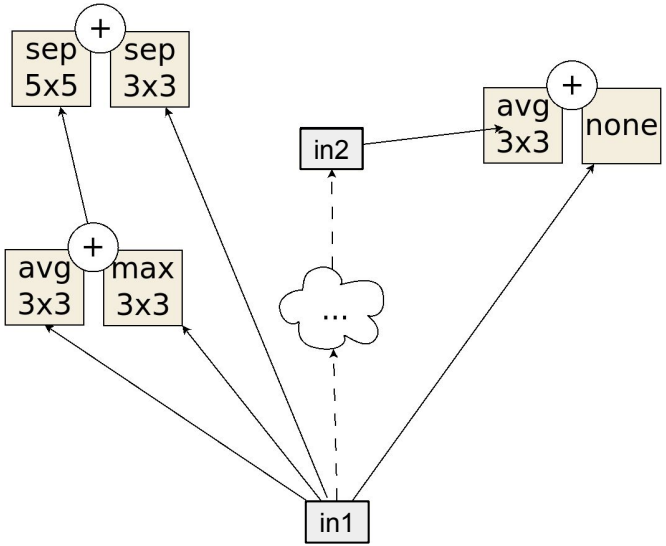

avg
3x3
in2
...
in1

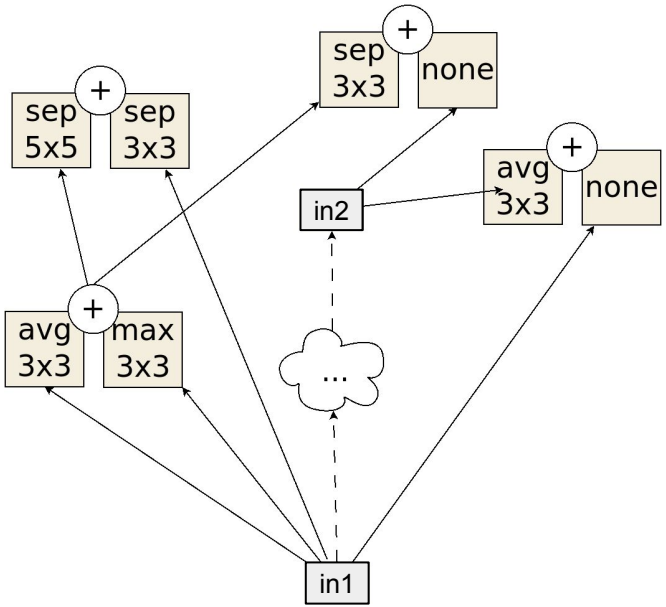

avg
3x3
none
in2
...
in1

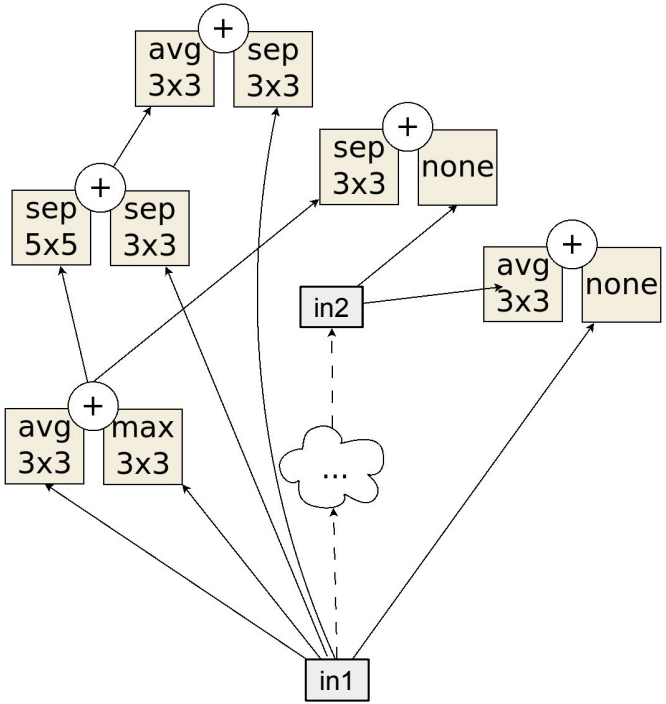
A "combination"

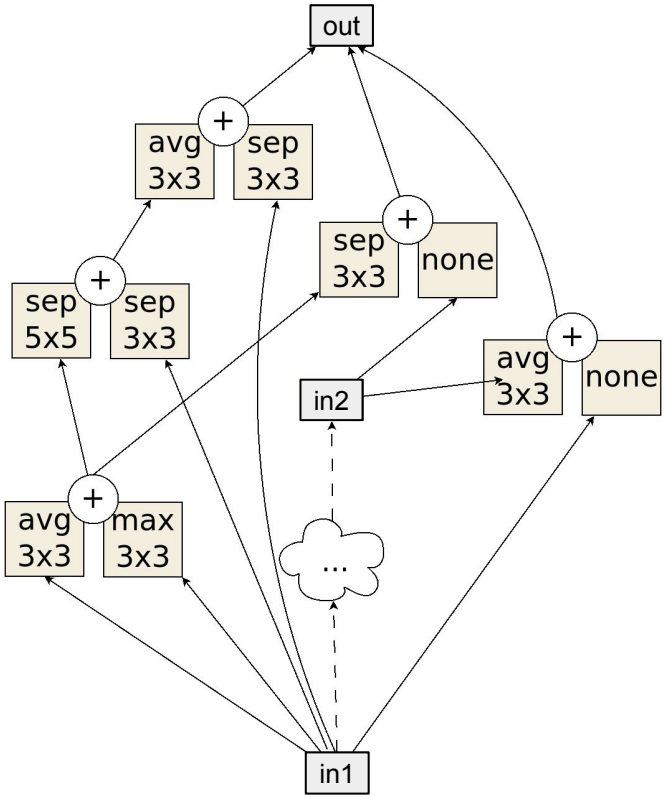


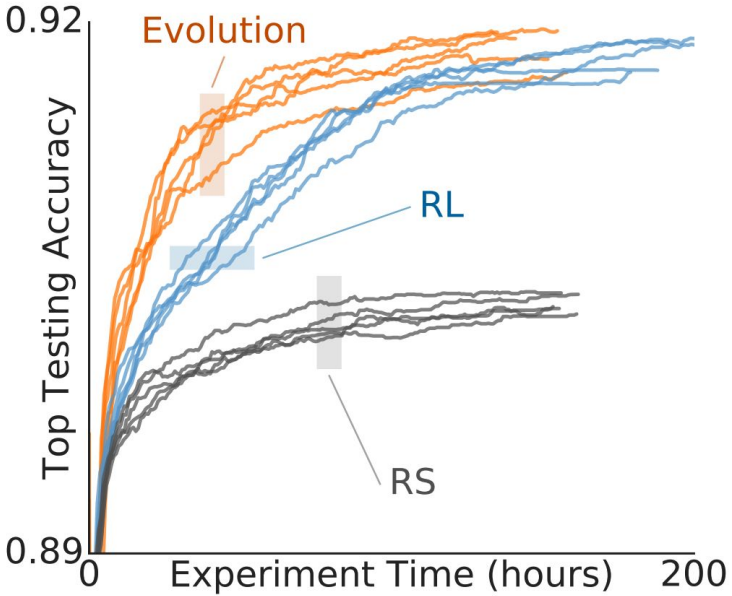

Another
"combination"
avg
3x3
max
3x3
+
in2
...
avg
3x3
none
+
in1


sep
5x5
sep
3x3
+
avg
3x3
max
3x3
+
in2
...
avg
3x3
none
+
in1


sep
5x5
+
sep
3x3
sep
3x3
+
none
avg
3x3
+
none
in2
avg
3x3
+
max
3x3
...
in1

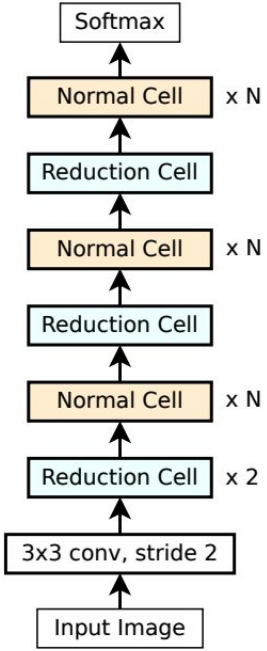

avg 3x3
sep 3x3
+
sep 5x5
sep 3x3
+
sep 3x3
none
+
in2
avg 3x3
none
+
avg 3x3
max 3x3
+
...
in1

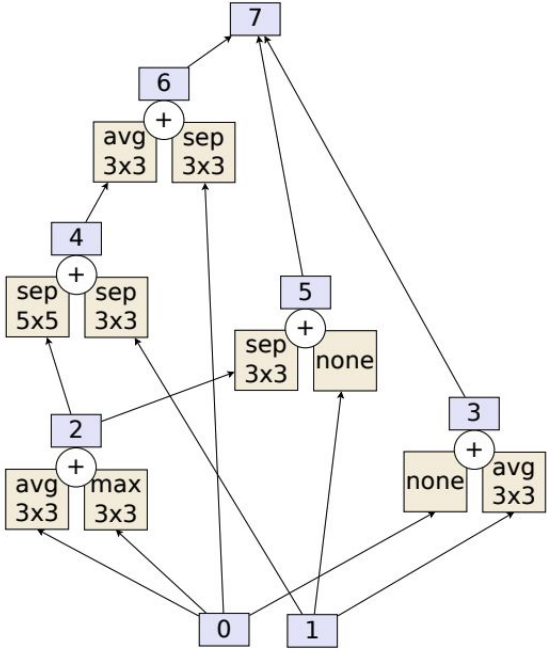

out
+
avg 3x3
sep 3x3
+
sep 3x3
none
+
sep 5x5
sep 3x3
+
avg 3x3
none
in2
+
avg 3x3
max 3x3
...
in1

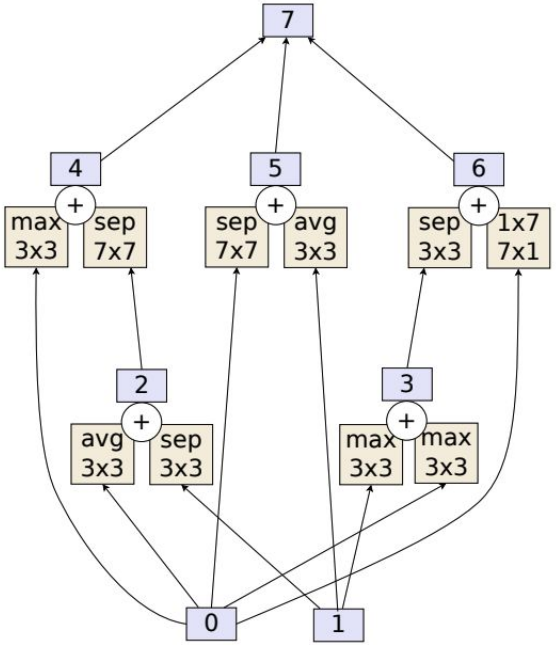


Regularized Evolution for Image Classifier Architecture Search
Real, Aggarwal, Huang, and Le (ICML 2018)

AmoebaNet-A



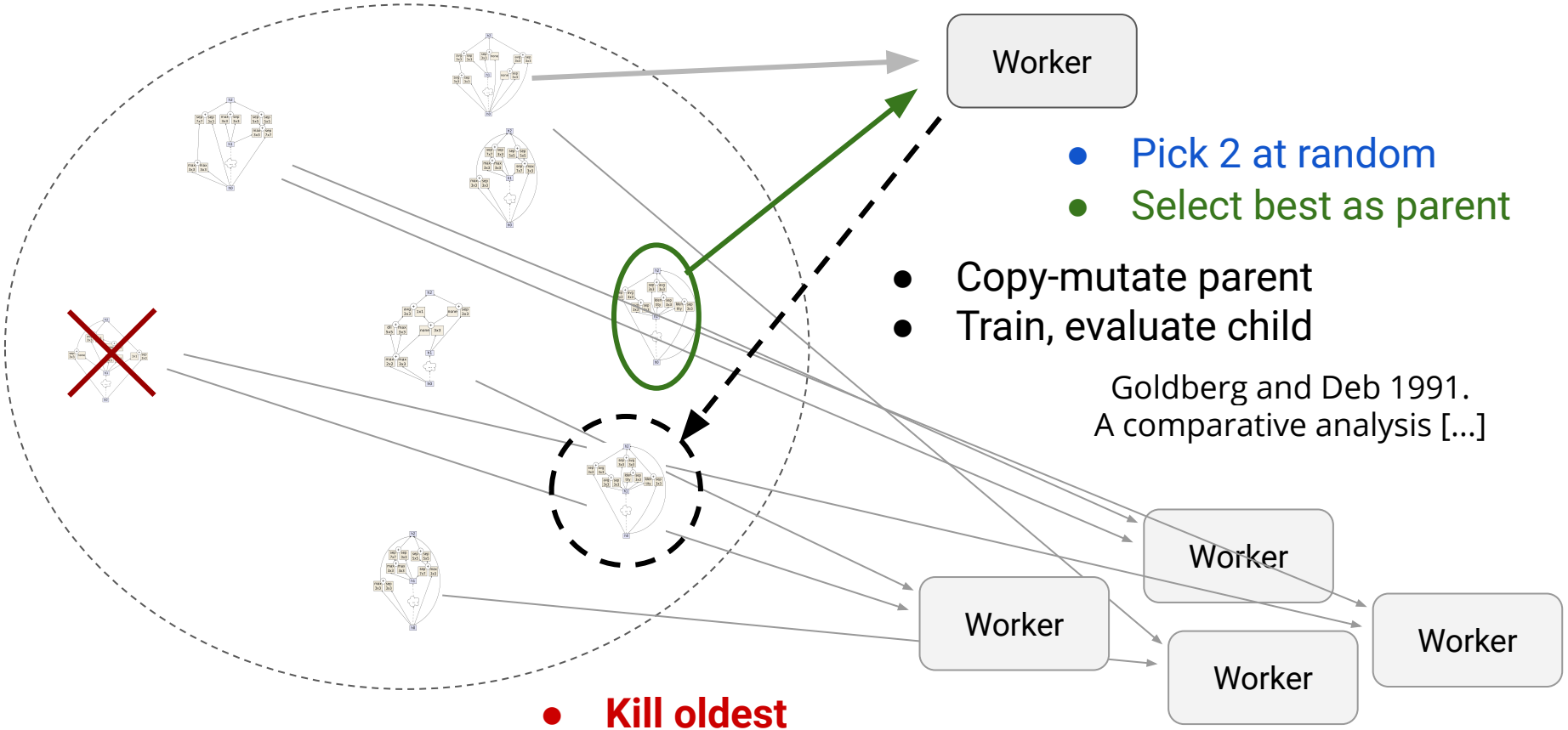




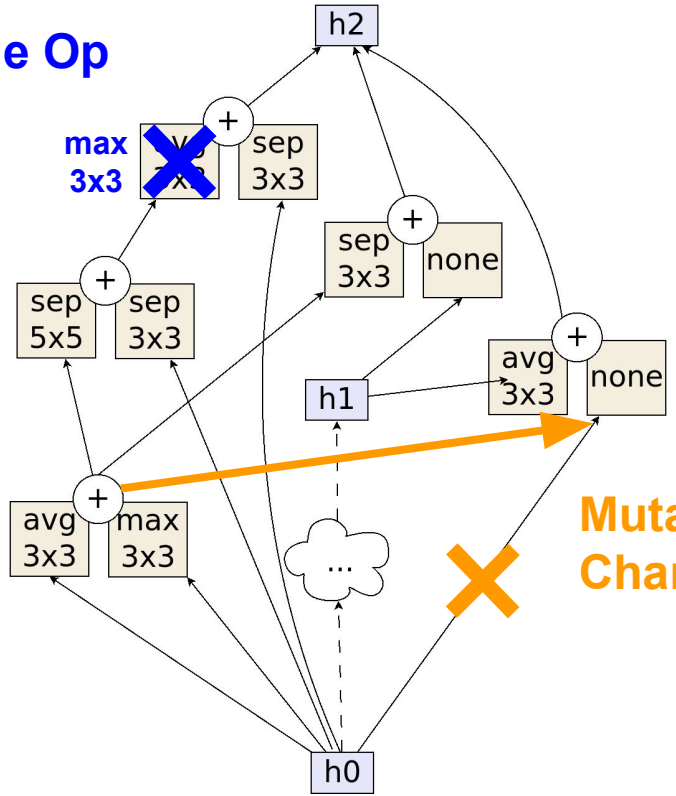
Regularized Evolution for Image Classifier Architecture Search
Real, Aggarwal, Huang, and Le (ICML 2018)

Evolutionary Method

- Initialize with random cells



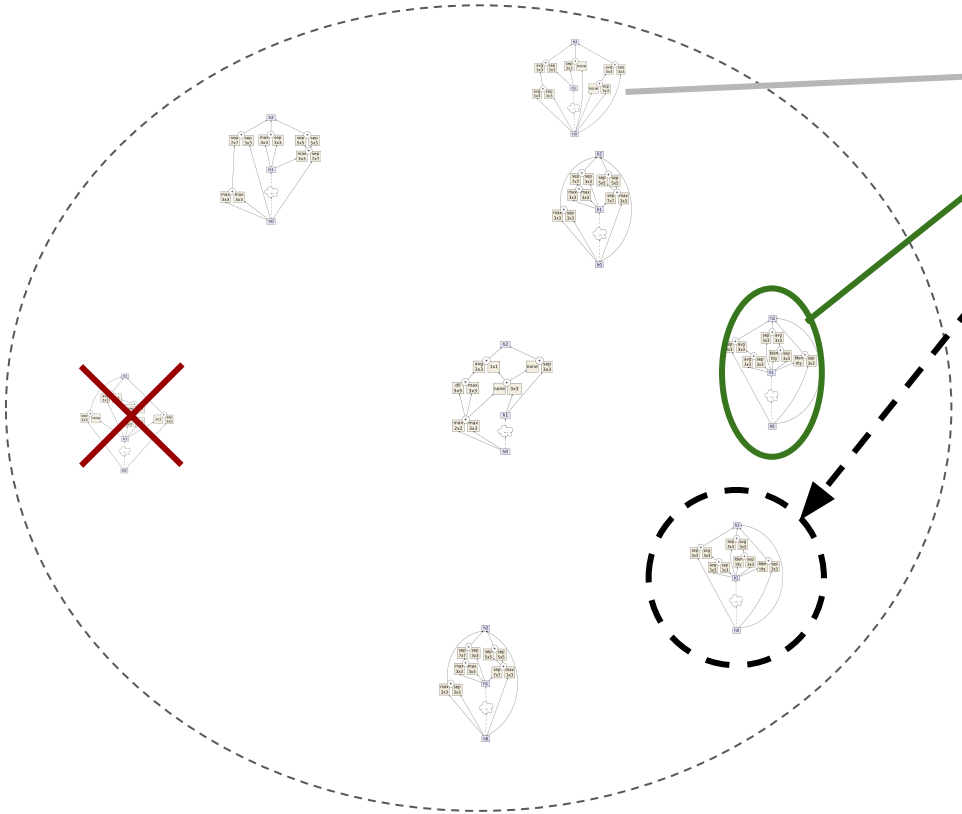
Mutation 1: Change Op



Mutation 2:
Change hidden state

Evolutionary Method

- Initialize with random cells



Worker

- Pick 2 at random
- Select best as parent

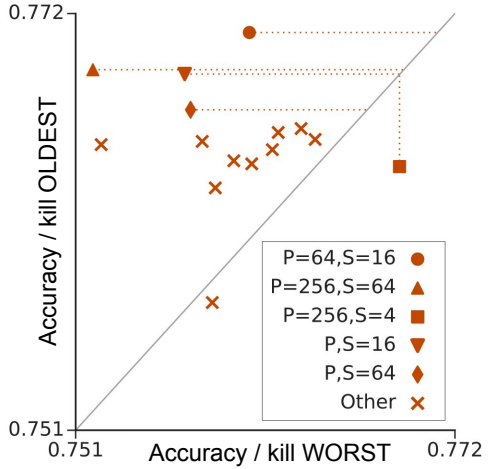
- Copy-mutate parent
- Train, evaluate child

Goldberg and Deb 1991.
A comparative analysis [...]

- **Kill oldest**

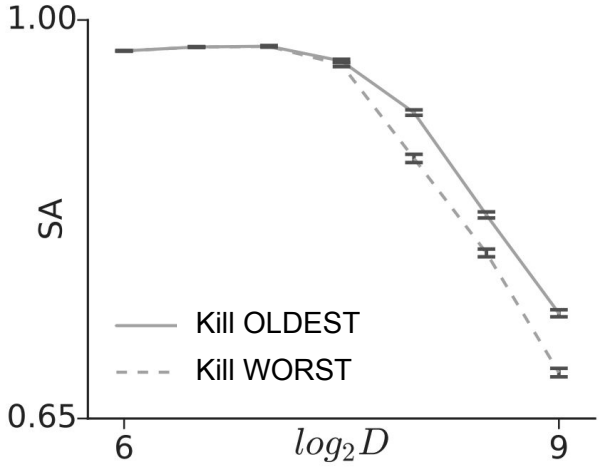
Regularized Evolution for Image Classifier Architecture Search
Real, Aggarwal, Huang, and Le (ICML 2018)

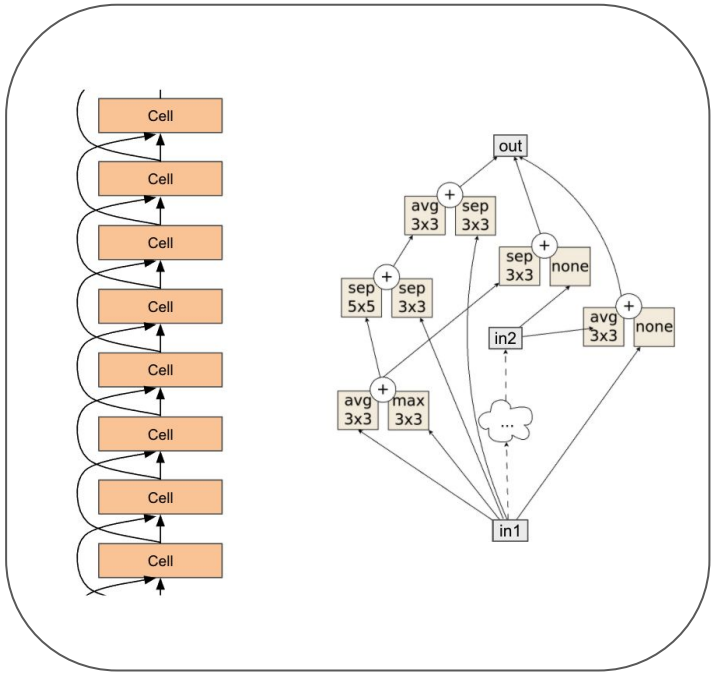
Effect of death



Why?

Hypothesis: death forces reevaluation.
⇒ Death should help when the evaluation is noisy.





"NASNet Search Space"

Can we avoid designing a complex search space?

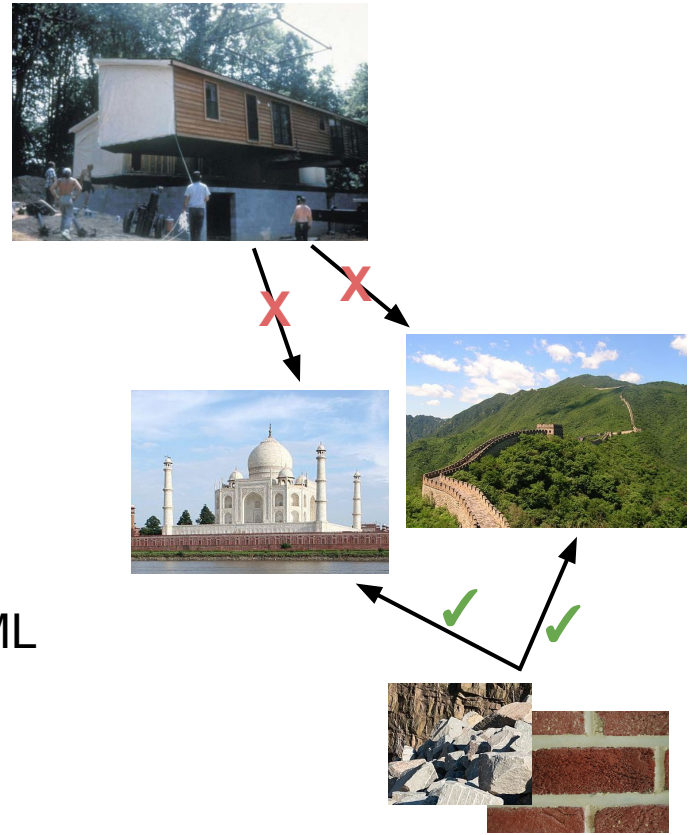
AutoML-Zero: Evolving ML Algorithms from Scratch

Esteban Real* Chen Liang*, David So and Quoc V. Le (*equal)

ICML 2020

What is AutoML-Zero?

- Discover everything: arch + alg
- Use only basic math ops
  No gradients! No backprop!
- Have few restrictions on form
- Search for whole algorithms



Why AutoML-Zero?

- AML→AML-Zero ~ Shallow→Deep ML
- Reduce bias by reducing design

Related Work

Prog. Synth

E.g. Early work

- Koza 1992, Schmidhuber 1987.

Task: string manipulation

- Polozov 2015. FlashMeta
- Parisotto 2016. Neuro-symbolic
- Devlin 2017. RobustFill

Task: QA on structured data

- Neelakantan 2015. Inducing...
- Liang 2016. Neural Symbolic...
- Liang 2018. Memory Augmented...

Task: sorting/counting/...

- Schmidhuber 2004. Optimal Ordered...
- Graves 2014. Neural Turing Machines
- Valkov 2018. HOUDINI

For us, task: ML

AutoML

Grow a neural net

- Miller 1989
- Stanley 2002. NEAT

Search for the cell architecture

- Zoph 2017. NASNet
- Real 2018. AmoebaNet
- Tan 2019. MNASNet

Search for the activation function

- Ramachandran 2017. Swish

Search for the hyperparameters

- Snoek 2012. Bayesian
- Jaderberg 2017. PBT
- Li 2018. Hyperband

Search for the data augmentation

- Cubuk 2019. AutoAugment
- Park 2019. SpecAugment
- Cubuk 2019. RandAugment

Search for the ...

- Fill the blank

Search for the optimizer

- Andrychowicz 2016
- Metz 2019
- Ravi 2017

- Bengio 1994
- Bello 2017

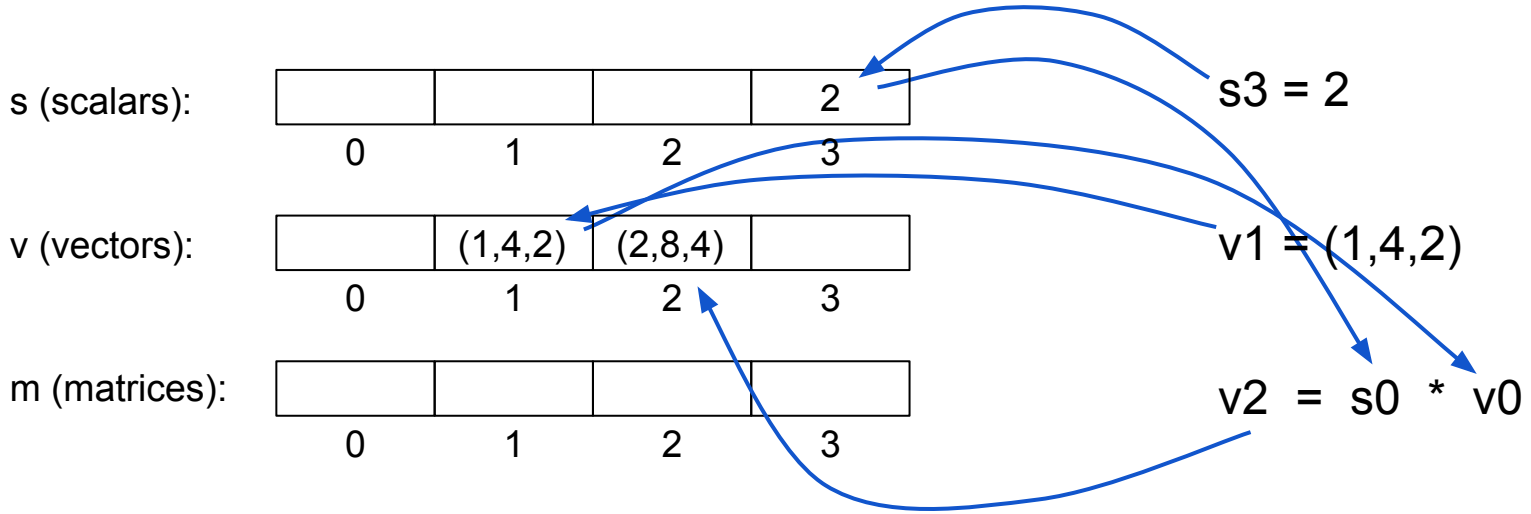
We want to search for all…
at the same time…
starting from scratch...

Search Space of Very Simple Functions

Memory

- Floats only.
- Address spaces: scalar, vector, matrix.

Example of a function:



- We search over *algorithms*.
- 1 algorithm has 3 *functions*: Setup(), Predict(), and Learn().

To evaluate an algorithm:

1. Execute **Setup()**
2. Train. For each example:
   a. Write features to v0
   b. Execute **Predict()**
   c. Write label to s0
   d. Execute **Learn()**
3. Validate. For each unseen example:
   a. Write features v0
   b. Execute **Predict()**
   c. Read prediction from s1
4. Compute loss (i.e. fitness) averaging over *many* tasks

Algorithm Example (linear regression)

```
def Setup():
  s2 = 0.01

def Predict():
  s1 = dot(v1, v0)

def Learn():
  s3 = s0 - s1
  s3 = s2 * s3
  v2 = s3 * v0
  v1 = v1 + v2
```

Search Space Difficulty

```
def Setup():
  s2 = 0.01
def Predict(v0):
  s1 = np.dot(v1, v0)
def Learn(v0, s0):
  s3 = s0 - s1
  s3 = s2 * s3
  v2 = s3 * v0
  v1 = v2 + v2
```

$1 \text{ in } 10^7$

```
def Setup():
  s2 = 0.01
def Predict(v0):
  s1 = np.dot(v1, v0)
  s5 = s1 + s4
def Learn(v0, s0):
  s3 = s0 - s1
  s3 = s2 * s3
  s4 = s4 + s3
  v2 = s3 * v0
  v1 = v2 + v2
```

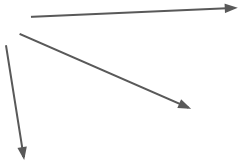
$1 \text{ in } 10^{12}$

```
# sX/vX/mX = scalar/vector/matrix at address X.
# The C_ are tunable hyperparameters (eg C1).
# "gaussian" produces Gaussian IID random numbers.
def Setup():
  m1 = gaussian(C1, C2) # Init. 1st layer weights
  s3 = C3 # Set learning rate
  v4 = gaussian(C5, C6) # Init. 2nd layer weights
def Predict(v0):  # v0=features
  v6 = np.dot(m1, v0) # Apply 1st layer weights
  v7 = np.maximum(0, v6) # Apply ReLU
  s1 = np.dot(v7, v4) # Compute prediction
  s1 = normalize(s1)
def Learn(v0, s0):  # v0=features, s0=label
  v3 = np.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 = np.outer(v3, v0) # 1st layer weight delta
  m1 = m1 + m0 # Update 1st layer weights
  v4 = v2 + v4 # Update 2nd layer weights
```

Evolved 2-layer Neural Network

Reduce bias even more...

- Variable program length
- 4x necessary addresses
- Many basic math ops



Pre-calculus
SCALAR_EXP_OP
SCALAR_LOG_OP

Vector arithmetic
VECTOR_SUM_OP
VECTOR_DIFF_OP
VECTOR_PRODUCT_OP
VECTOR_DIVISION_OP
VECTOR_MIN_OP
VECTOR_MAX_OP
VECTOR_ABS_OP
VECTOR_HEAVYSIDE_OP
VECTOR_RELU_OP
VECTOR_CONST_ASSIGN_OP

Trigonometry-related
SCALAR_SIN_OP
SCALAR_COS_OP
SCALAR_TAN_OP
SCALAR_ARCSIN_OP
SCALAR_ARCCOS_OP
SCALAR_ARCTAN_OP

Scalar arithmetic
SCALAR_SUM_OP
SCALAR_DIFF_OP
SCALAR_PRODUCT_OP
SCALAR_DIVISION_OP
SCALAR_MIN_OP
SCALAR_MAX_OP
SCALAR_ABS_OP
SCALAR_HEAVYSIDE_OP
SCALAR_CONST_ASSIGN_OP

Matrix arithmetic
MATRIX_SUM_OP
MATRIX_DIFF_OP
MATRIX_PRODUCT_OP
MATRIX_DIVISION_OP
MATRIX_MIN_OP
MATRIX_MAX_OP
MATRIX_ABS_OP
MATRIX_HEAVYSIDE_OP
MATRIX_CONST_SET_OP

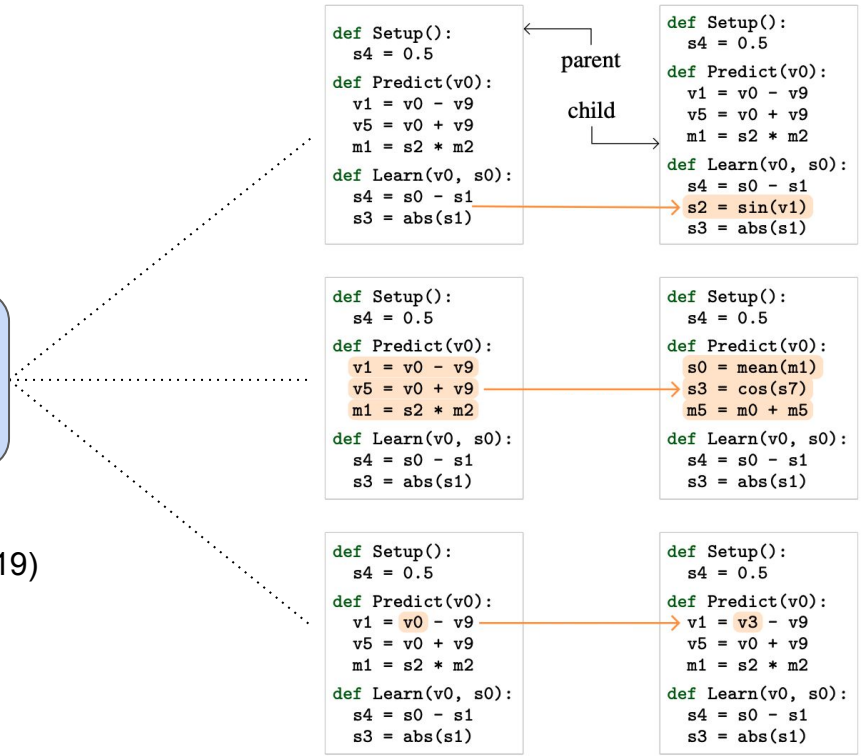
Linear algebra-related
SCALAR_VECTOR_PRODUCT_OP
VECTOR_INNER_PRODUCT_OP
VECTOR_OUTER_PRODUCT_OP
SCALAR_MATRIX_PRODUCT_OP
MATRIX_VECTOR_PRODUCT_OP
VECTOR_NORM_OP
MATRIX_NORM_OP
MATRIX_TRANSPOSE_OP
MATRIX_MATRIX_PRODUCT_OP

Probability-related
VECTOR_MEAN_OP
VECTOR_ST_DEV_OP
MATRIX_MEAN_OP
MATRIX_ST_DEV_OP
MATRIX_ROW_MEAN_OP
MATRIX_ROW_ST_DEV_OP
SCALAR_GAUSSIAN_ASSIGN_OP
VECTOR_GAUSSIAN_ASSIGN_OP
MATRIX_GAUSSIAN_ASSIGN_OP
SCALAR_UNIFORM_ASSIGN_OP
VECTOR_UNIFORM_ASSIGN_OP
MATRIX_UNIFORM_ASSIGN_OP

Method: Simple Evolutionary Search

Evolutionary method can take 3 actions

"Regularized Evolution" (Real 2019)
~Holland 1975, Goldberg 1991



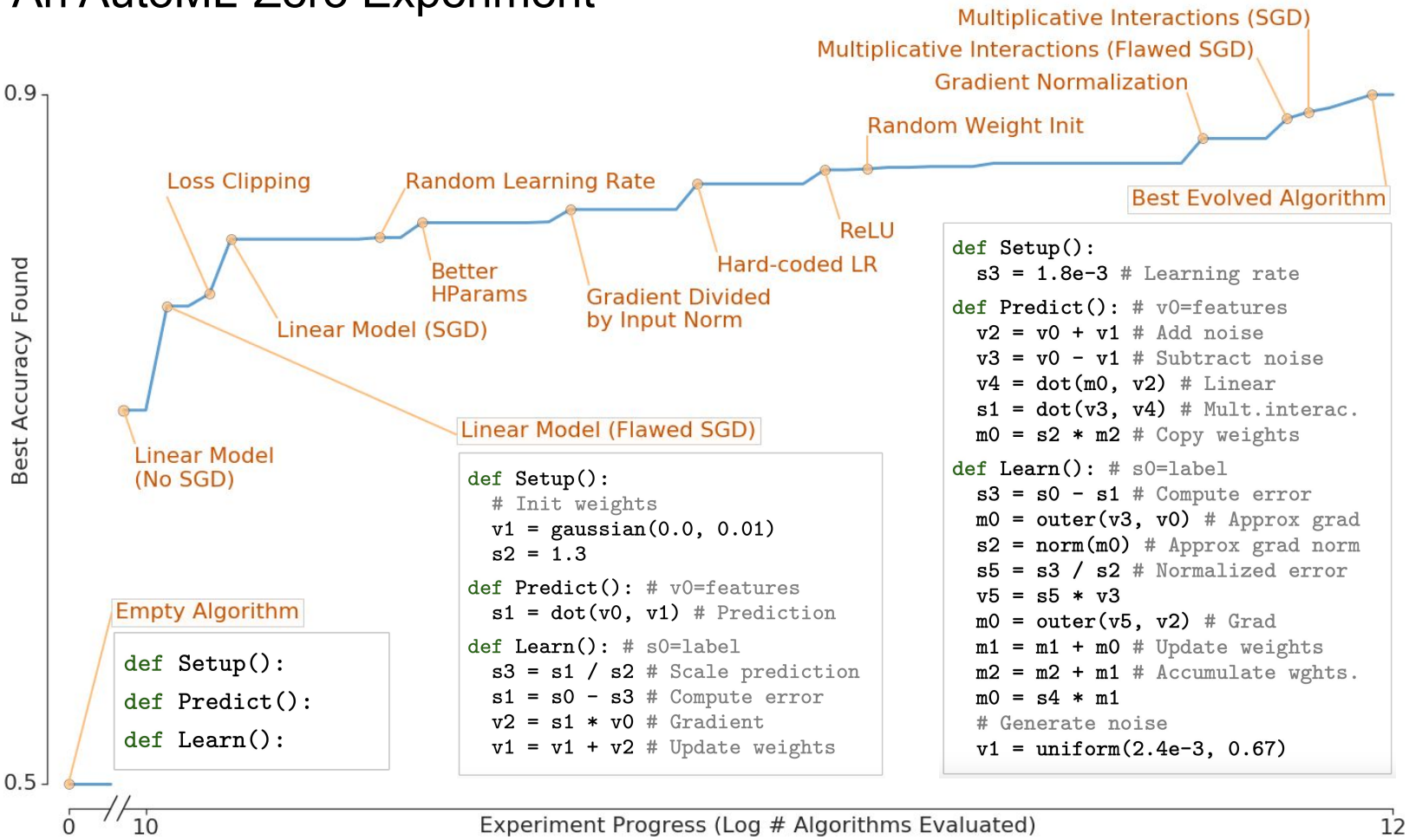
Action 1:
insert (or remove)
a random instruction

Action 2:
randomize many
instructions

Action 3:
Randomize address
or parameter

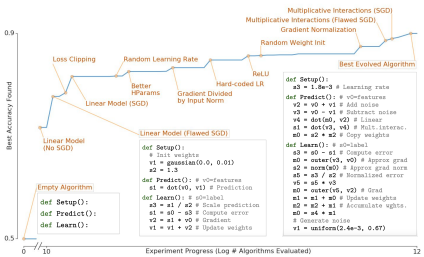
+ Optimizations ⇒ 10k algorithms / second / CPU core

An AutoML-Zero Experiment



The evolved algorithm generalizes.

Evolve algorithm on projected binary CIFAR-10



Test algorithm on unseen data

	Full-size Binary CIFAR-10	Binary ImageNet	Binary SVHN	Binary Fashion MNIST
Linear	77.65%	76.44%	59.58%	97.90%
Simple NN	82.22%	78.44%	86.14%	98.21%
Evolved	**84.06%**	**80.78%**	**88.12%**	**98.60%**

Outperforms hand-designs of similar complexity (even after thoroughly tuning the hand-designs)

What does the evolved algorithm do?

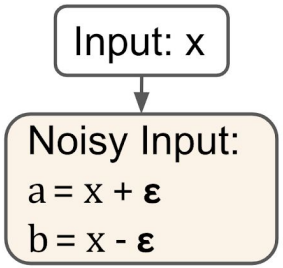
```
def Setup():
  s3 = 1.8e-3 # Learning rate
def Predict(): # v0=features
  v2 = v0 + v1 # Add noise
  v3 = v0 - v1 # Subtract noise
  v4 = dot(m0, v2) # Linear
  s1 = dot(v3, v4) # Mult.interac.
  m0 = s2 * m2 # Copy weights
def Learn(): # s0=label
  s3 = s0 - s1 # Compute error
  m0 = outer(v3, v0) # Approx grad
  s2 = norm(m0) # Approx grad norm
  s5 = s3 / s2 # Normalized error
  v5 = s5 * v3
  m0 = outer(v5, v2) # Grad
  m1 = m1 + m0 # Update weights
  m2 = m2 + m1 # Accumulate wghts.
  m0 = s4 * m1
  # Generate noise
  v1 = uniform(2.4e-3, 0.67)
```

Input: x

Label: y*

It adds noise to the input...

```
def Setup():
  s3 = 1.8e-3 # Learning rate
def Predict(): # v0=features
  v2 = v0 + v1 # Add noise
  v3 = v0 - v1 # Subtract noise
  v4 = dot(m0, v2) # Linear
  s1 = dot(v3, v4) # Mult.interac.
  m0 = s2 * m2 # Copy weights
def Learn(): # s0=label
  s3 = s0 - s1 # Compute error
  m0 = outer(v3, v0) # Approx grad
  s2 = norm(m0) # Approx grad norm
  s5 = s3 / s2 # Normalized error
  v5 = s5 * v3
  m0 = outer(v5, v2) # Grad
  m1 = m1 + m0 # Update weights
  m2 = m2 + m1 # Accumulate wghts.
  m0 = s4 * m1
  # Generate noise
  v1 = uniform(2.4e-3, 0.67)
```

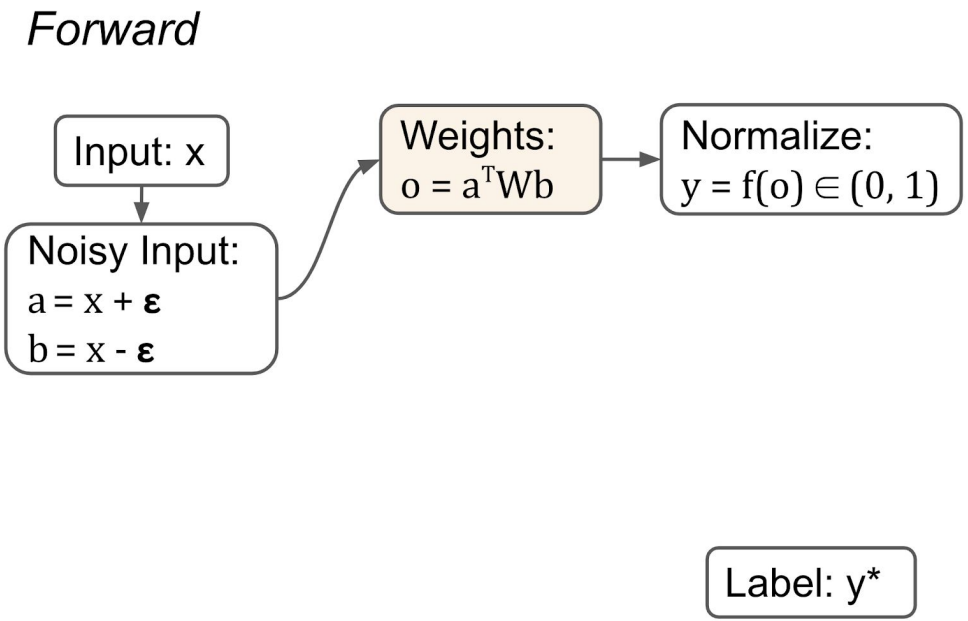


Label: y*

~ noted in Goodfellow 2016

It applies a bilinear transform...

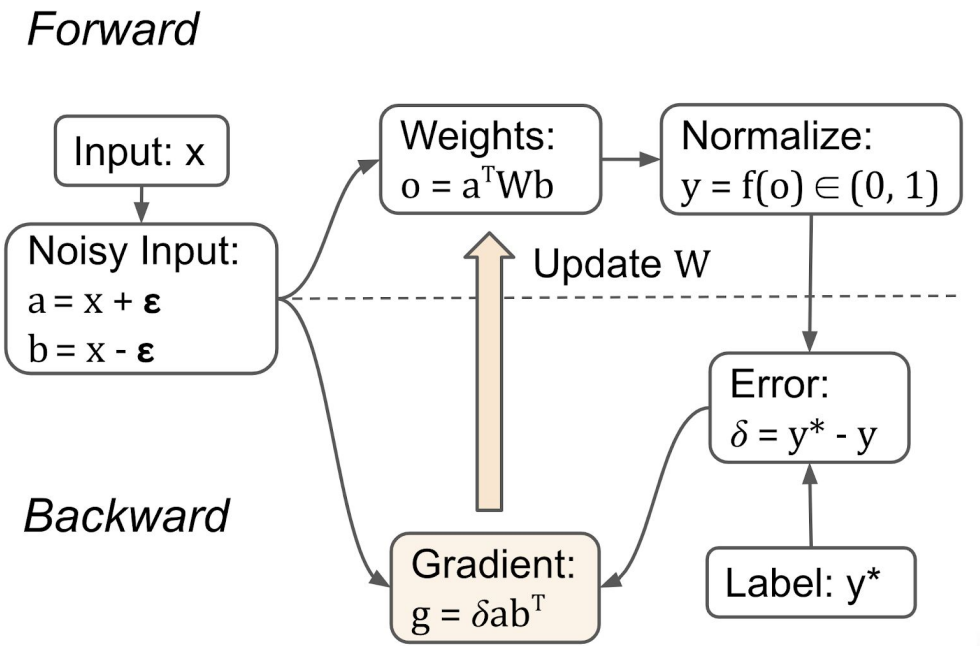
```
def Setup():
  s3 = 1.8e-3 # Learning rate
def Predict(): # v0=features
  v2 = v0 + v1 # Add noise
  v3 = v0 - v1 # Subtract noise
  v4 = dot(m0, v2) # Linear
  s1 = dot(v3, v4) # Mult.interac.
  m0 = s2 * m2 # Copy weights
def Learn(): # s0=label
  s3 = s0 - s1 # Compute error
  m0 = outer(v3, v0) # Approx grad
  s2 = norm(m0) # Approx grad norm
  s5 = s3 / s2 # Normalized error
  v5 = s5 * v3
  m0 = outer(v5, v2) # Grad
  m1 = m1 + m0 # Update weights
  m2 = m2 + m1 # Accumulate wghts.
  m0 = s4 * m1
  # Generate noise
  v1 = uniform(2.4e-3, 0.67)
```



~ noted in Jayakumar 2020

It computes the error gradient...

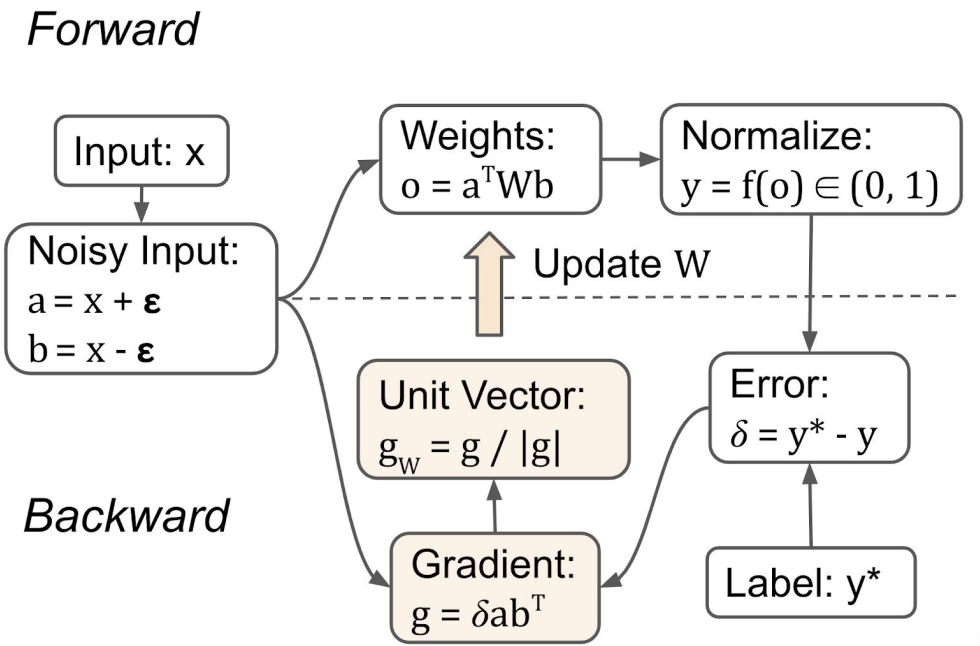
```
def Setup():
  s3 = 1.8e-3 # Learning rate
def Predict(): # v0=features
  v2 = v0 + v1 # Add noise
  v3 = v0 - v1 # Subtract noise
  v4 = dot(m0, v2) # Linear
  s1 = dot(v3, v4) # Mult.interac.
  m0 = s2 * m2 # Copy weights
def Learn(): # s0=label
  s3 = s0 - s1 # Compute error
  m0 = outer(v3, v0) # Approx grad
  s2 = norm(m0) # Approx grad norm
  s5 = s3 / s2 # Normalized error
  v5 = s5 * v3
  m0 = outer(v5, v2) # Grad
  m1 = m1 + m0 # Update weights
  m2 = m2 + m1 # Accumulate wghts.
  m0 = s4 * m1
  # Generate noise
  v1 = uniform(2.4e-3, 0.67)
```



~ standard SGD

It normalizes the gradient...

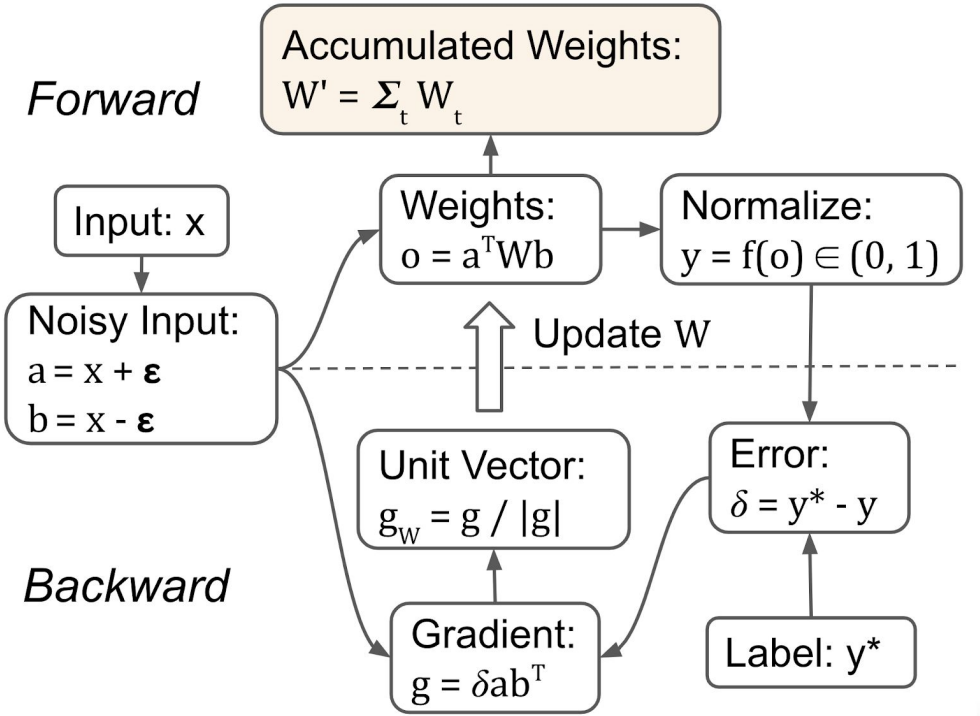
```
def Setup():
  s3 = 1.8e-3 # Learning rate
def Predict(): # v0=features
  v2 = v0 + v1 # Add noise
  v3 = v0 - v1 # Subtract noise
  v4 = dot(m0, v2) # Linear
  s1 = dot(v3, v4) # Mult.interac.
  m0 = s2 * m2 # Copy weights
def Learn(): # s0=label
  s3 = s0 - s1 # Compute error
  m0 = outer(v3, v0) # Approx grad
  s2 = norm(m0) # Approx grad norm
  s5 = s3 / s2 # Normalized error
  v5 = s5 * v3
  m0 = outer(v5, v2) # Grad
  m1 = m1 + m0 # Update weights
  m2 = m2 + m1 # Accumulate wghts.
  m0 = s4 * m1
  # Generate noise
  v1 = uniform(2.4e-3, 0.67)
```



~ Hazan 2015, Levy 2016

It implements a weight-averaging equivalent...

```
def Setup():
  s3 = 1.8e-3 # Learning rate
def Predict(): # v0=features
  v2 = v0 + v1 # Add noise
  v3 = v0 - v1 # Subtract noise
  v4 = dot(m0, v2) # Linear
  s1 = dot(v3, v4) # Mult.interac.
  m0 = s2 * m2 # Copy weights
def Learn(): # s0=label
  s3 = s0 - s1 # Compute error
  m0 = outer(v3, v0) # Approx grad
  s2 = norm(m0) # Approx grad norm
  s5 = s3 / s2 # Normalized error
  v5 = s5 * v3
  m0 = outer(v5, v2) # Grad
  m1 = m1 + m0 # Update weights
  m2 = m2 + m1 # Accumulate wghts.
  m0 = s4 * m1
  # Generate noise
  v1 = uniform(2.4e-3, 0.67)
```



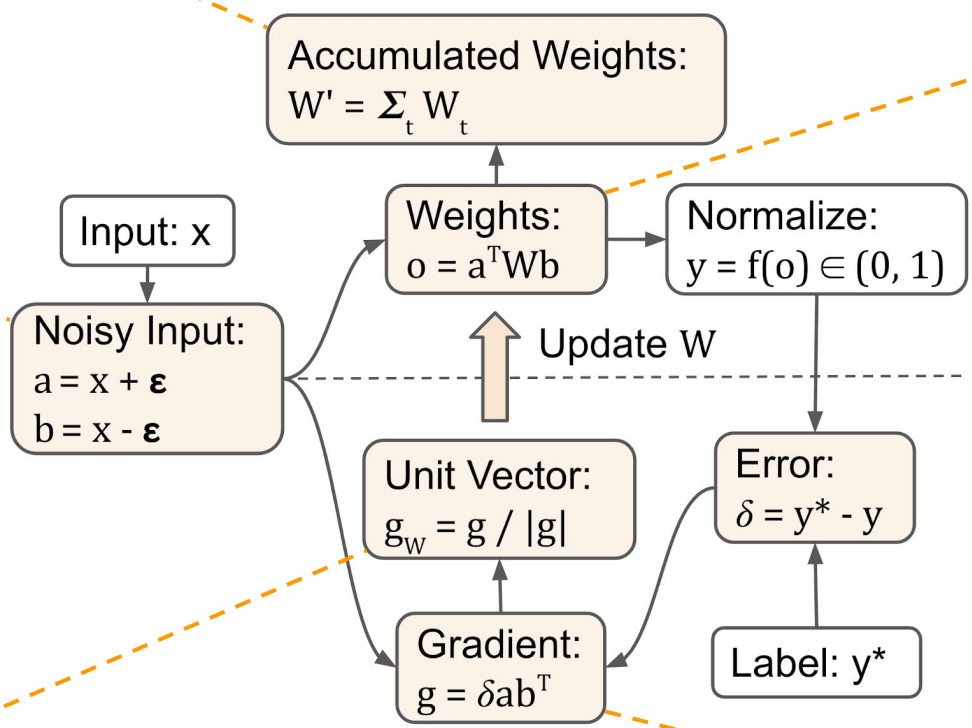
~ Polyak 1992, Collins 2002

What does the evolved algorithm do?

~ Polyak 1992, Collins 2002

~ noted in Jayakumar 2020

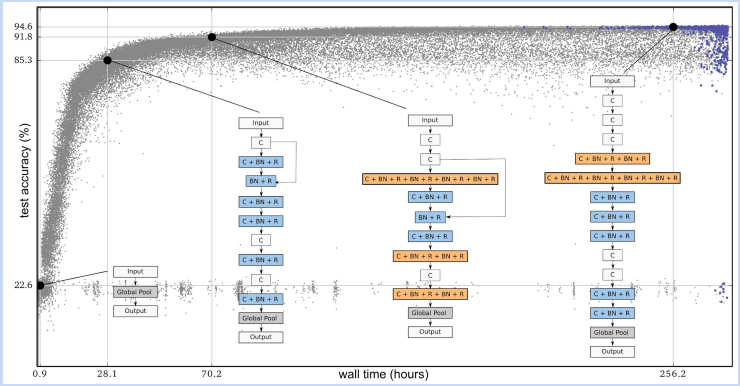
~ noted in Goodfellow 2016



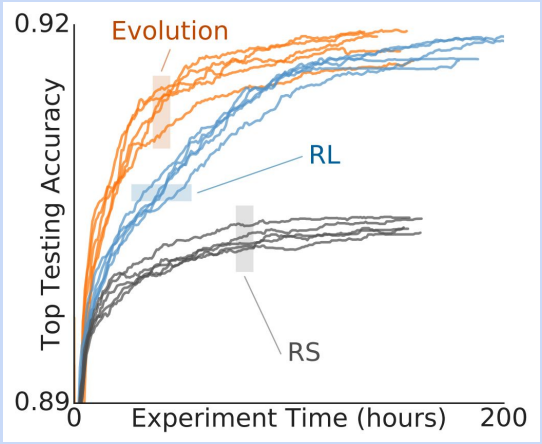
Next?

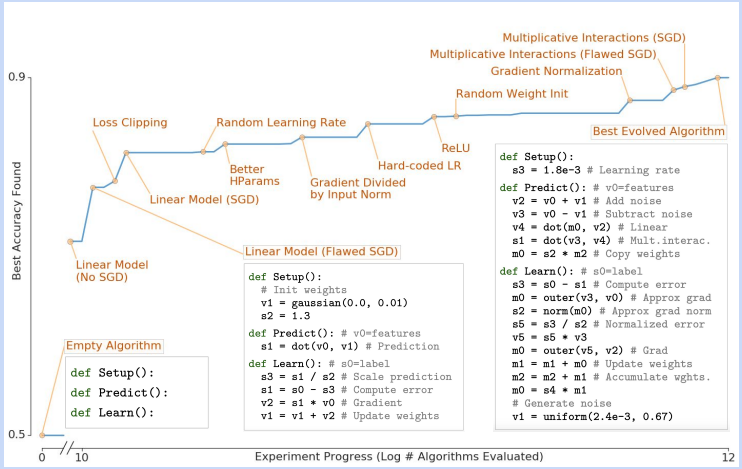
~ Hazan 2015, Levy 2016

~ standard SGD


94.6
91.8
85.3
22.6
test accuracy (%)
0.9
28.1
70.2
256.2
wall time (hours)

Thanks!


0.92
0.89
Top Testing Accuracy
Evolution
RL
RS
0
Experiment Time (hours)
200


Best Accuracy Found
0.9
0.5
Multiplicative Interactions (SGD)
Multiplicative Interactions (Flawed SGD)
Gradient Normalization
Random Weight Init
Loss Clipping
Random Learning Rate
Best Evolved Algorithm
ReLU
Better HParams
Hard-coded LR
Gradient Divided by Input Norm
Linear Model (SGD)
Linear Model (No SGD)
Linear Model (Flawed SGD)
Empty Algorithm
def Setup():
def Predict():
def Learn():
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
def Setup():
s3 = 1.8e-3 # Learning rate
def Predict(): # v0=features
v2 = v0 + v1 # Add noise
v3 = v0 - v1 # Subtract noise
v4 = dot(m0, v2) # Linear
s1 = dot(v3, v4) # Mult.interac.
m0 = s2 * m2 # Copy weights
def Learn(): # s0=label
s3 = s0 - s1 # Compute error
m0 = outer(v3, v0) # Approx grad
s2 = norm(m0) # Approx grad norm
s5 = s3 / s2 # Normalized error
v5 = s5 * v3
m0 = outer(v5, v2) # Grad
m1 = m1 + m0 # Update weights
m2 = m2 + m1 # Accumulate wghts.
m0 = s4 * m1
# Generate noise
v1 = uniform(2.4e-3, 0.67)
0
10
Experiment Progress (Log # Algorithms Evaluated)
12