Deep Learning Optimization - Neural Architecture Search 2

April 19, 2023

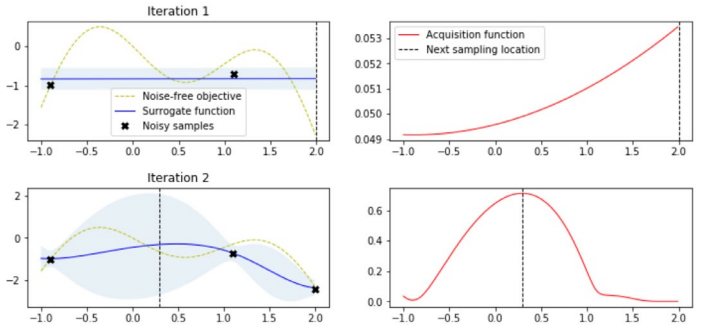
Eunhyeok Park

Black Box Optimization

- Try to optimization hyper-parameter/configuration of target function when only $f(x)$ is available
  - $\nabla f(x)\ or\ \nabla^2 f(x)$ are limited or not available

- Mostly based on heuristics, but some with theoretical support
  - REINFORCEMENT learning
    - Q-learning, SARSA, policy optimization, …
  - Evolutionary algorithms
    - Genetic algorithm, CMA-es,…
  - Simulated annealing
  - Bayesian optimization
  - …

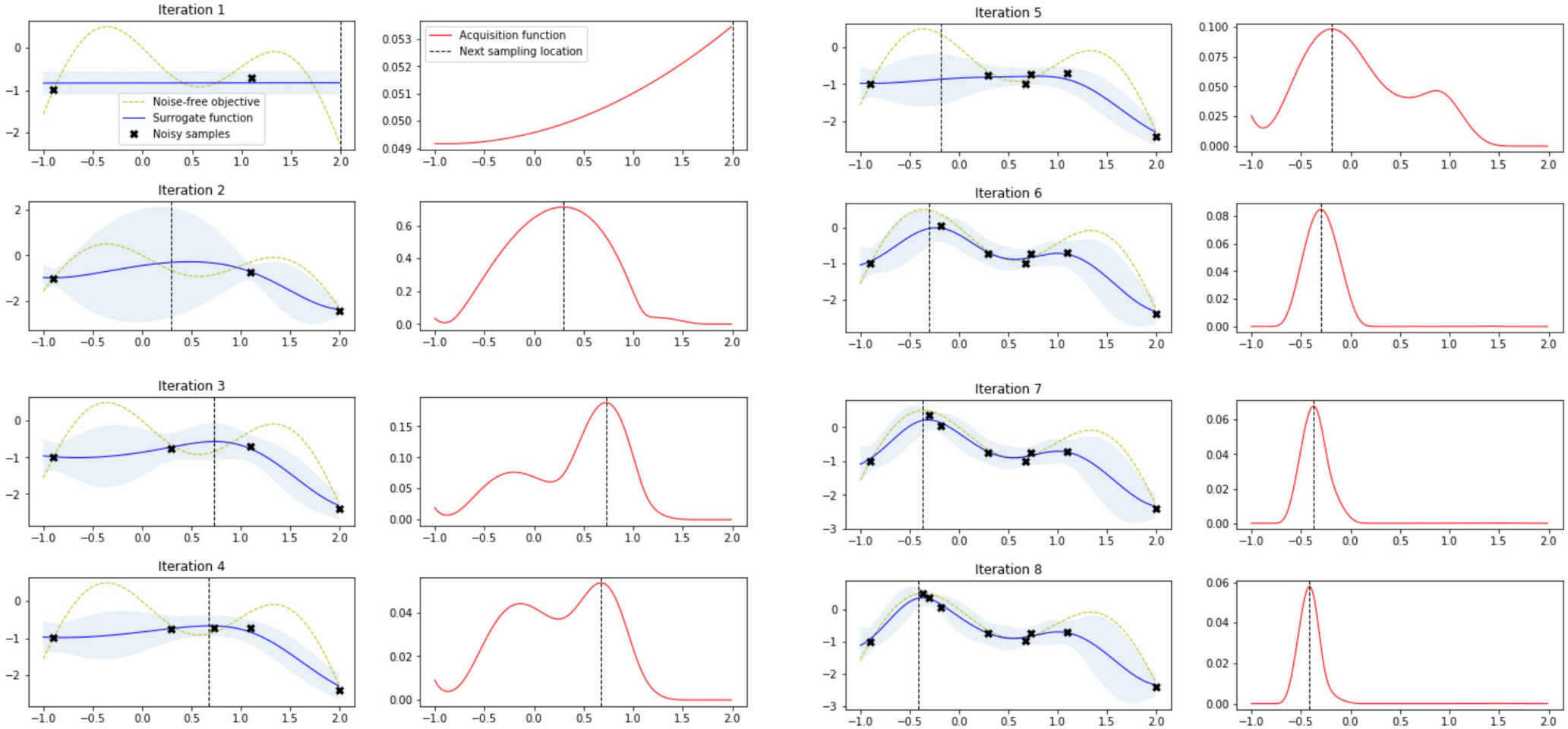
Brief Intro – Bayesian Optimization



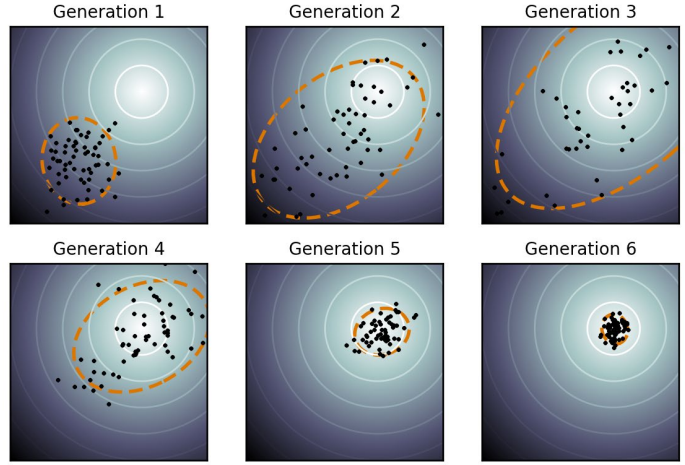
- Address the problem of optimizing a real-value function $f(x)$ over some bounded domain $\mathcal{X}$
- Surrogate model is the model used for approximating the objective function
  - Often use Gaussian processes that define a prior over functions
  - The sampled data update the surrogate model, and the posterior is used to propose points in search space that are expected to have the largest improvement

- Proposing sampling points in the search space is done by acquisition functions.
  - Trade off exploitation and exploration
  - Sampling where the surrogate model predicts a high objective
  - Sampling at locations where the prediction uncertainty is high

Brief Intro – Bayesian Optimization -2

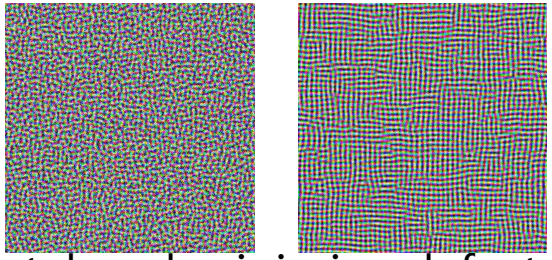


Brief Intro – CMA-ES

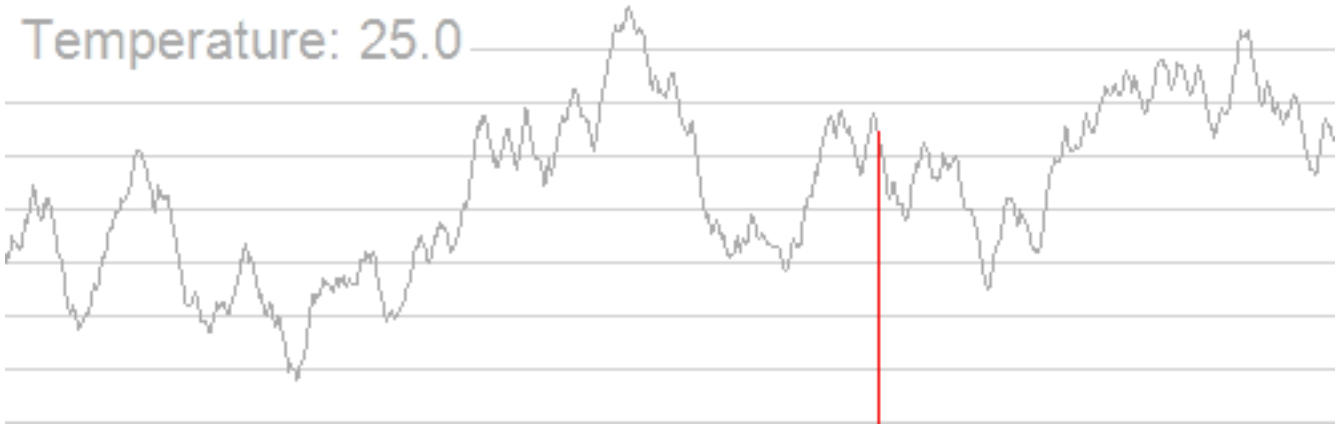


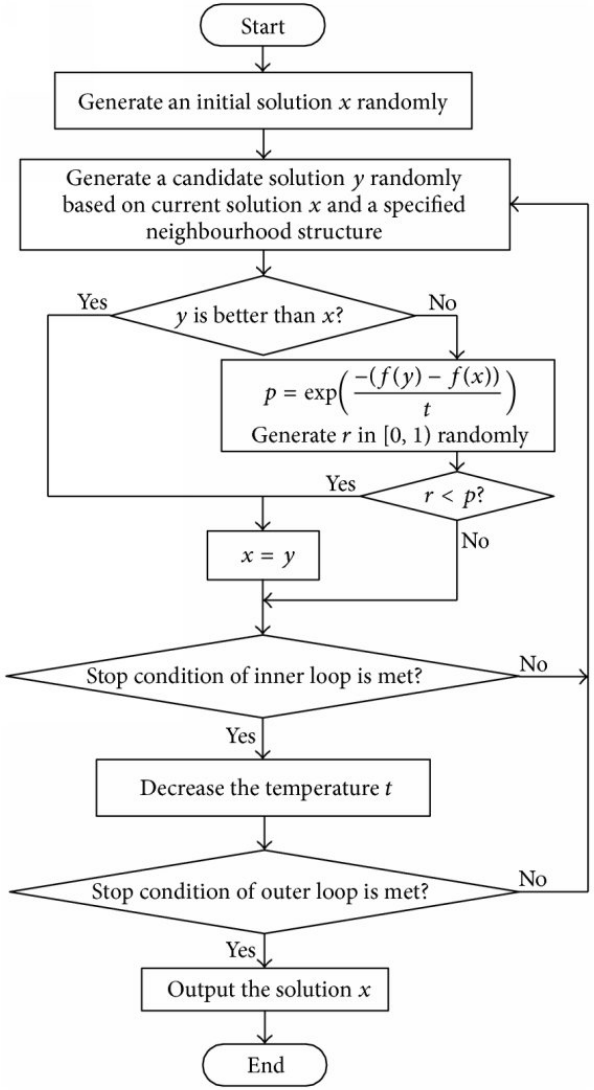
- Mixture of covariance matrix adaptation and evolution strategy
  - In each generation,
    - New individuals are generated in a stochastic way
      - Their distribution is modeled by a multivariate normal distribution
      - Covariance matrix of the distribution is adjusted to maximize fitness gain
    - Some individuals are selected to become the parents in the next generation
    - Repeat variation and selection until converges
  - Mainly used for optimizing continuous space
    - Works well when the loss surface is smooth and has a consistent trend

Brief Intro – Simulated Annealing



- Probabilistic heuristic approach
  - Inspired by metallurgy annealing
    - Control cooling speed to raise crystal and minimize defects
  - Try to finding the global optimum in a stochastic manner
    - High temperature: a bad point can be selected
    - Low temperature: extensive exploration near the local minimum





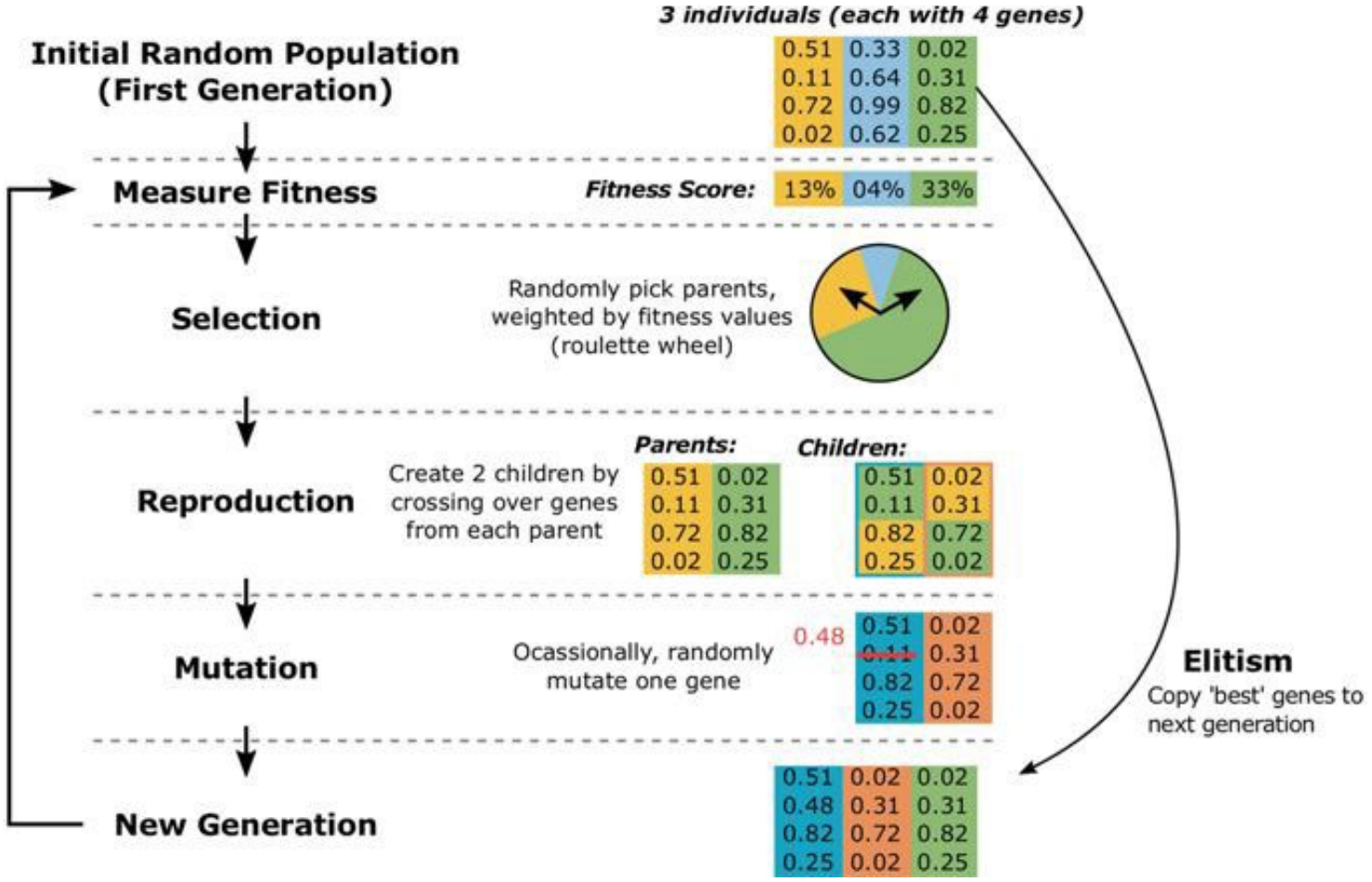
Brief Intro – Genetic Algorithm

- Inspired by Charles Darwin's natural selection
  - In real life, living things thrive through reproduction
    - Good traits are inherited through generations
      - An object suitable for survival is more likely to adapt to the environment
      - The superior object has more breeding opportunities
    - Inherit traits from two parents
      - Increase diversity through the intersecting genes of two parents
    - The mutation explores new possibilities
    - An object suitable for survival continues reproduction

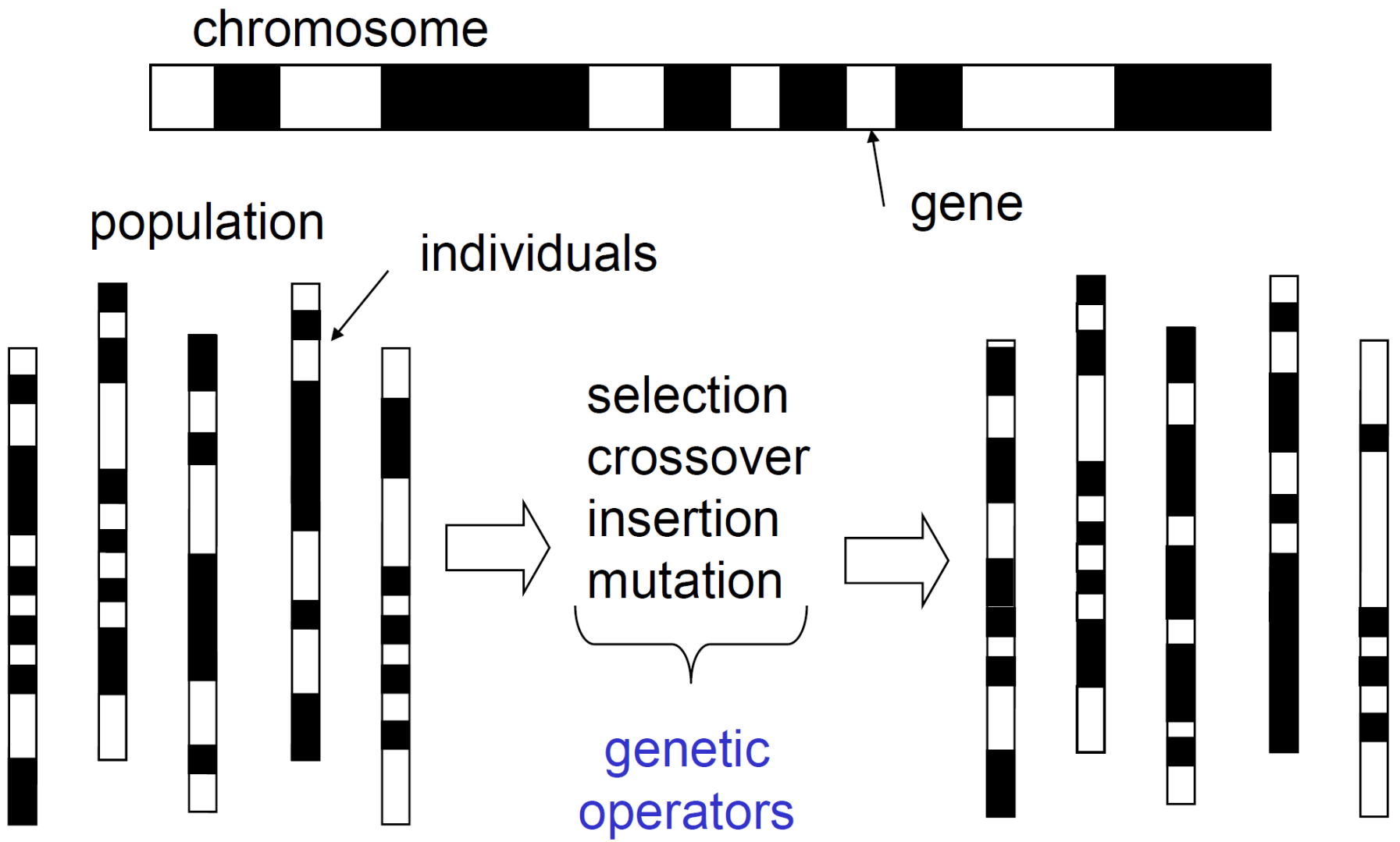
- Genetic algorithms are hard to see as specific algorithms
  - This is more like a methodology to solve various problems

Overview - GA

- [ESD.77 Lecture 11, Genetic algorithms I - MIT OpenCourseWare]



GA Terminology



Implementing GA on Computer

1. Define the representation (encoding-decoding)
2. Define “fitness” function F
   - Incorporate feasibility (constraints) and objectives
3. Define the genetic operators
   - Initialization, selection, crossover, mutation, insertion
4. Execute initial algorithm run-monitor average population fitness
   - Identify best individual
5. Tune algorithm
   - Adjust selection, insertion strategy, mutation rate

1. Define the representation

genotype
coded domain

phenotype
decision domain

Biology

UGCAACCGU
(“DNA” blocks)

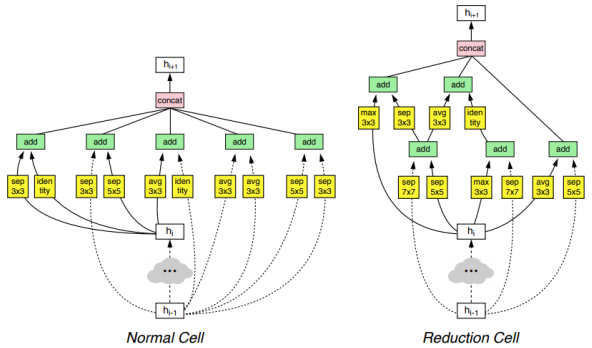
expression →
← *sequencing*

“blue eye”

Design

10010011110
(chromosome)

decoding →
← *encoding*



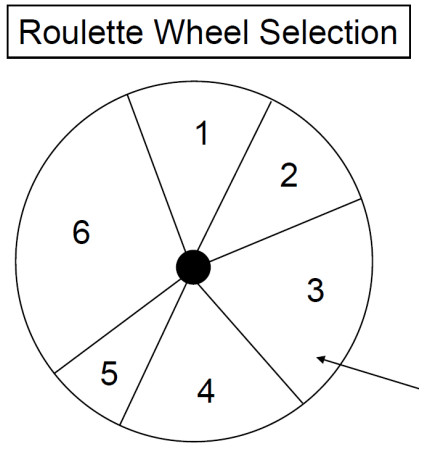
2. Define “fitness” Function F

- Choosing the right fitness function is very important, but also quite difficult
- GAs do not have explicit “constraints”
- Constraints can be handled in different ways:
  - via the fitness function
    - Penalty for violation
  - via the selection operator (“reject constraint violators”)
  - Implicitly via representation/coding
    - e.g., only allow representations of the TSP that correspond to a valid tour
  - Implement a repair capability for infeasible individuals

3. Define the Genetic Operators (Selection)

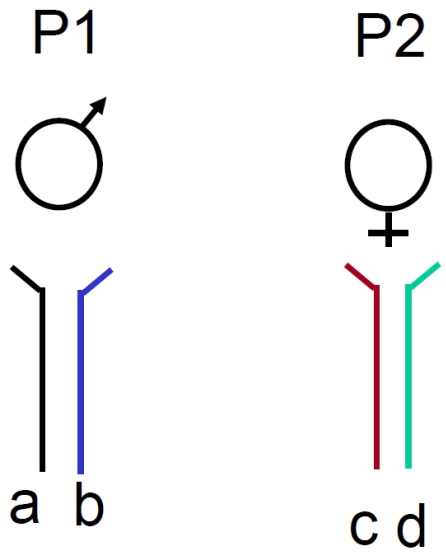
- Goal is to select parents for crossover
  - Should create a bias towards more fitness
  - Must preserve diversity in the population

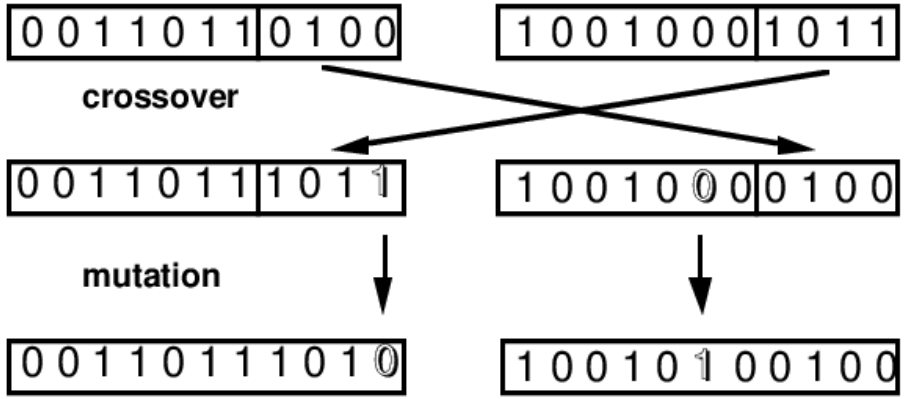
  - Ranking, roulette wheel selection, tournament selection, etc…



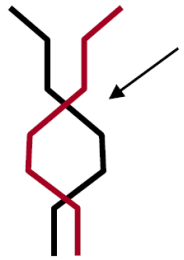
Probabilistically select individuals based on some measure of their performance.

3. Define the Genetic Operators (Crossover / Mutation)





Child



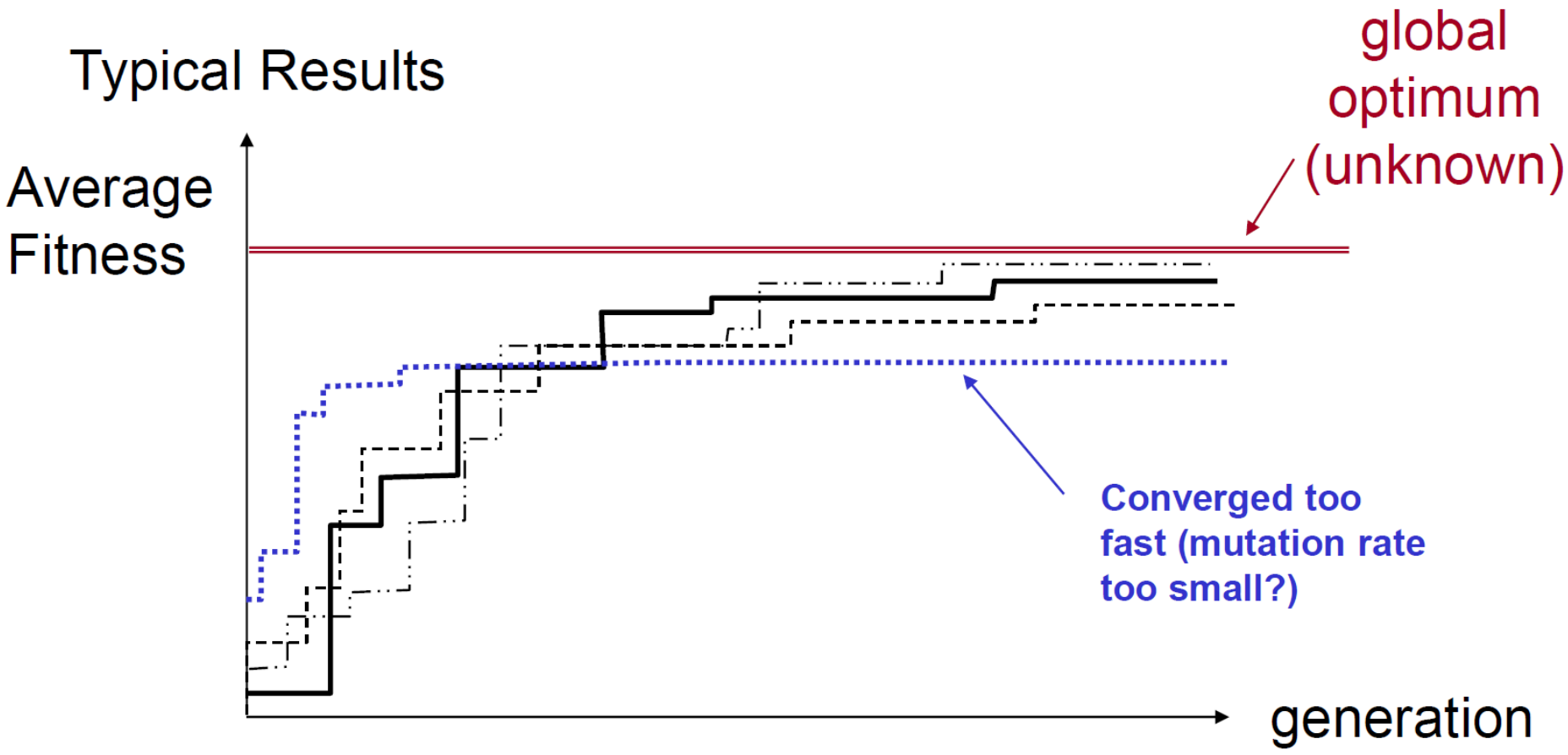
This is where the word crossover comes from

ac

⇨ Crossover produces either of these results for each chromosome

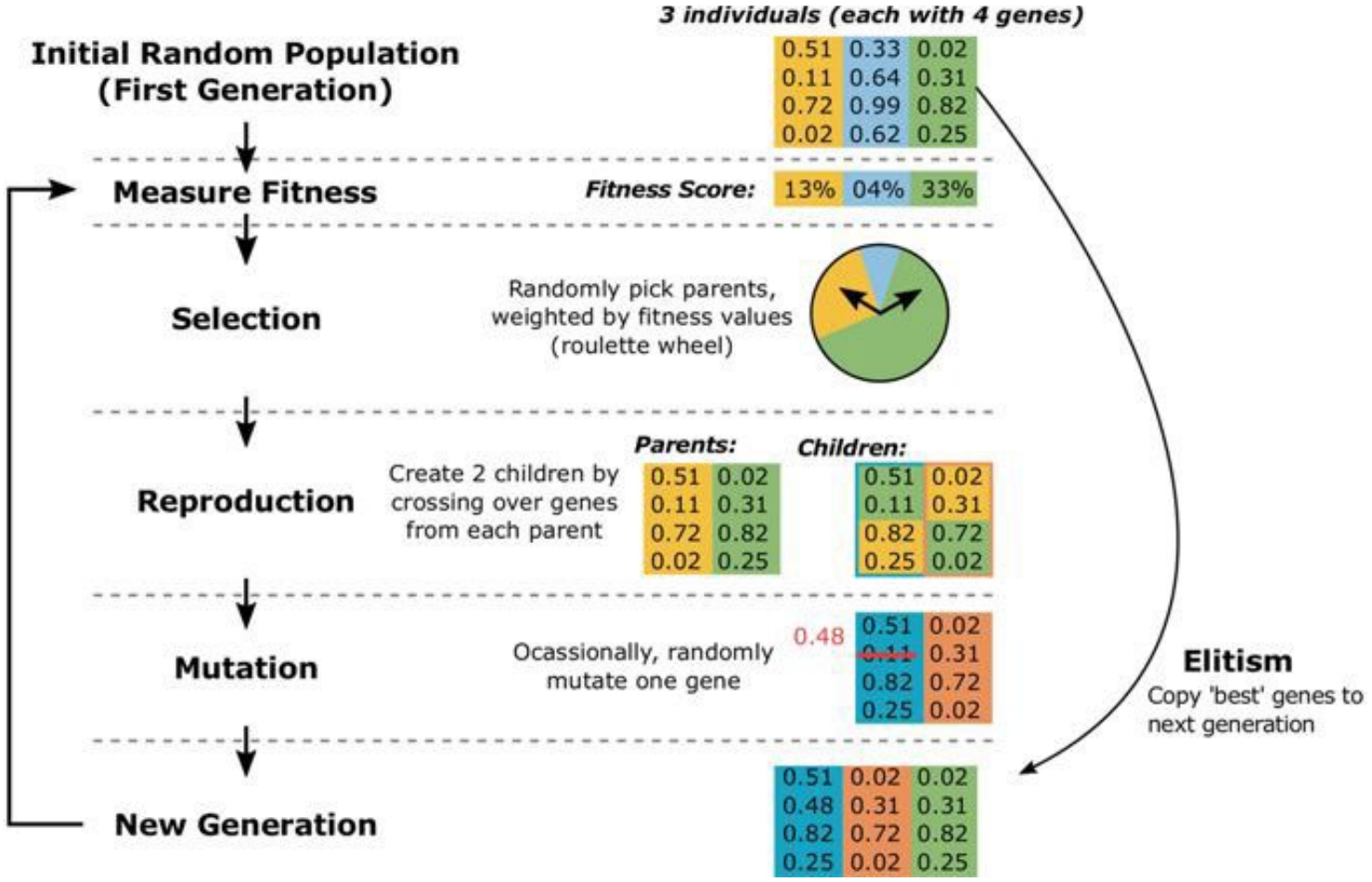
ac OR ad OR bc OR bd

4. Execute & Monitor



Overview - GA

- [ESD.77 Lecture 11, Genetic algorithms I - MIT OpenCourseWare]



AmoebaNet

Implementation

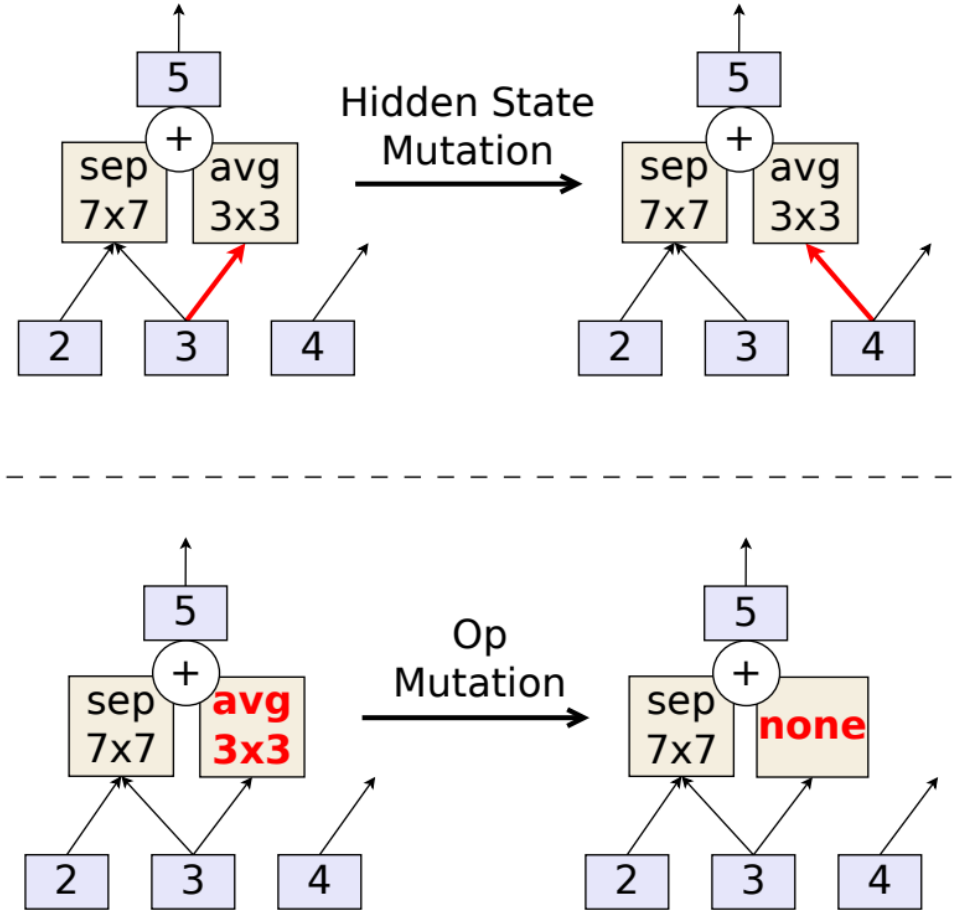


Figure 2: Illustration of the two mutation types.

Algorithm 1 Aging Evolution

$population \leftarrow$ empty queue ▷ The population.
$history \leftarrow \varnothing$ ▷ Will contain all models.
while $|population| < P$ **do** ▷ Initialize population.
  $model.arch \leftarrow$ RANDOMARCHITECTURE()
  $model.accuracy \leftarrow$ TRAINANDEVAL($model.arch$)
  add $model$ to right of $population$
  add $model$ to history
end while
while $|history| < C$ **do** ▷ Evolve for C cycles.
  $sample \leftarrow \varnothing$ ▷ Parent candidates.
  while $|sample| < S$ **do**
    $candidate \leftarrow$ random element from $population$
      ▷ The element stays in the $population$.
    add $candidate$ to $sample$
  end while
  $parent \leftarrow$ highest-accuracy model in $sample$
  $child.arch \leftarrow$ MUTATE($parent.arch$)
  $child.accuracy \leftarrow$ TRAINANDEVAL($child.arch$)
  add $child$ to right of $population$
  add $child$ to $history$
  remove $dead$ from left of $population$ ▷ Oldest.
  discard $dead$
end while
return highest-accuracy model in $history$

Results

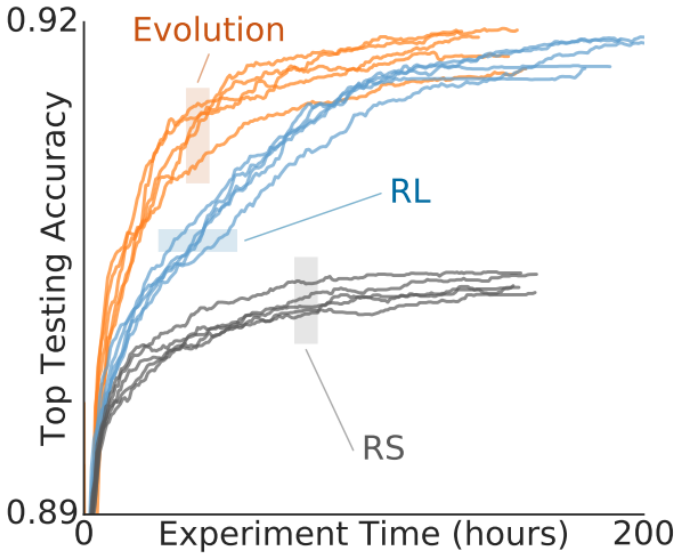
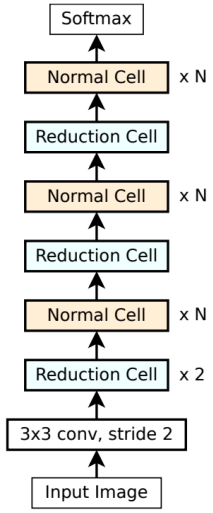
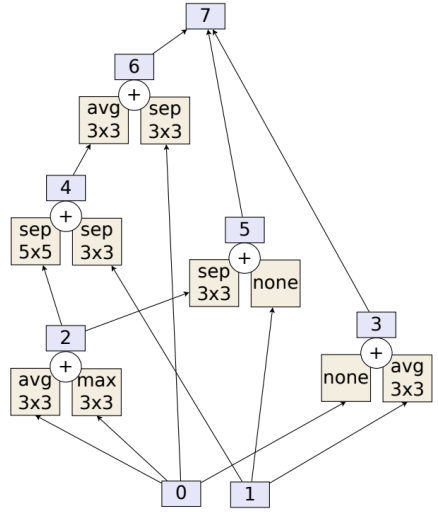
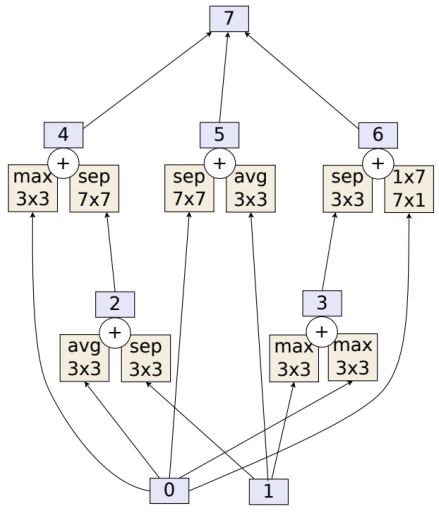


Figure 3: Time-course of 5 identical large-scale experiments for each algorithm (evolution, RL, and RS), showing accuracy before augmentation on CIFAR-10. All experiments were stopped when 20k models were evaluated, as done in the baseline study. Note this plot does not show the compute cost of models, which was higher for the RL ones.







Model	# Parameters	# Multiply-Adds	Top-1 / Top-5 Accuracy (%)
Incep-ResNet V2 [44]	55.8M	13.2B	80.4 / 95.3
ResNeXt-101 [48]	83.6M	31.5B	80.9 / 95.6
PolyNet [51]	92.0M	34.7B	81.3 / 95.8
Dual-Path-Net-131 [7]	79.5M	32.0B	81.5 / 95.8
GeNet-2 [47]*	156M	–	72.1 / 90.4
Block-QNN-B [52]*	–	–	75.7 / 92.6
Hierarchical [30]*	64M	–	79.7 / 94.8
NASNet-A [54]	88.9M	23.8B	82.7 / 96.2
PNASNet-5 [29]	86.1M	25.0B	82.9 / 96.2
AmoebaNet-A (N=6, F=190)*	86.7M	23.1B	82.8 / 96.1
AmoebaNet-A (N=6, F=448)*	469M	104B	83.9 / 96.6

DARTS:
Differentiable Architecture Search

Basic Idea

- Differentiable approach for the black box optimization
  - Adopt attention (weighted sum) idea to the architecture search
  - Instead of searching a discrete set of candidates, the architecture can be optimized through gradient descent

Neural Attention

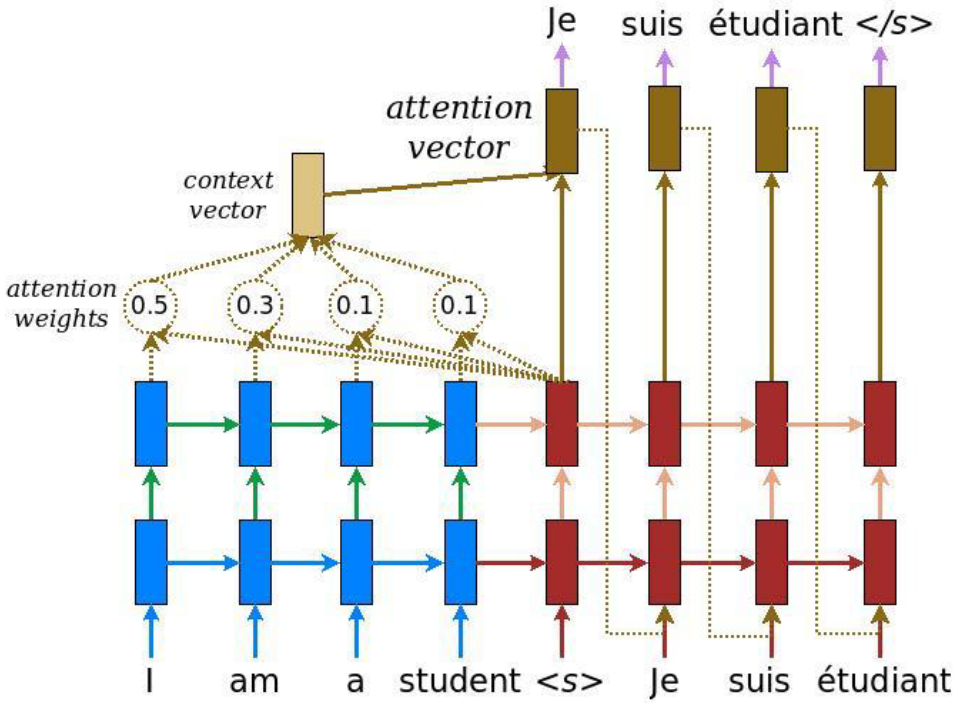
- Proposed at the previous neural translation study
  - Calculate where to focus, or the interesting point
  - Measure similarity between embeddings and transferred the score to the attention weight
    - Final output is calculated by the weighted sum

$$\alpha_{ts} = \frac{\exp\left(\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s)\right)}{\sum_{s'=1}^{S} \exp\left(\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_{s'})\right)} \quad \text{[Attention weights]} \quad (1)$$

$$\boldsymbol{c}_t = \sum_s \alpha_{ts} \bar{\boldsymbol{h}}_s \quad \text{[Context vector]} \quad (2)$$

$$\boldsymbol{a}_t = f(\boldsymbol{c}_t, \boldsymbol{h}_t) = \tanh(\boldsymbol{W_c}[\boldsymbol{c}_t; \boldsymbol{h}_t]) \quad \text{[Attention vector]} \quad (3)$$

$$\text{score}(\boldsymbol{h}_t, \bar{\boldsymbol{h}}_s) = \begin{cases} \boldsymbol{h}_t^\top \boldsymbol{W} \bar{\boldsymbol{h}}_s & \text{[Luong's multiplicative style]} \\ \boldsymbol{v}_a^\top \tanh\left(\boldsymbol{W_1} \boldsymbol{h}_t + \boldsymbol{W_2} \bar{\boldsymbol{h}}_s\right) & \text{[Bahdanau's additive style]} \end{cases} \quad (4)$$

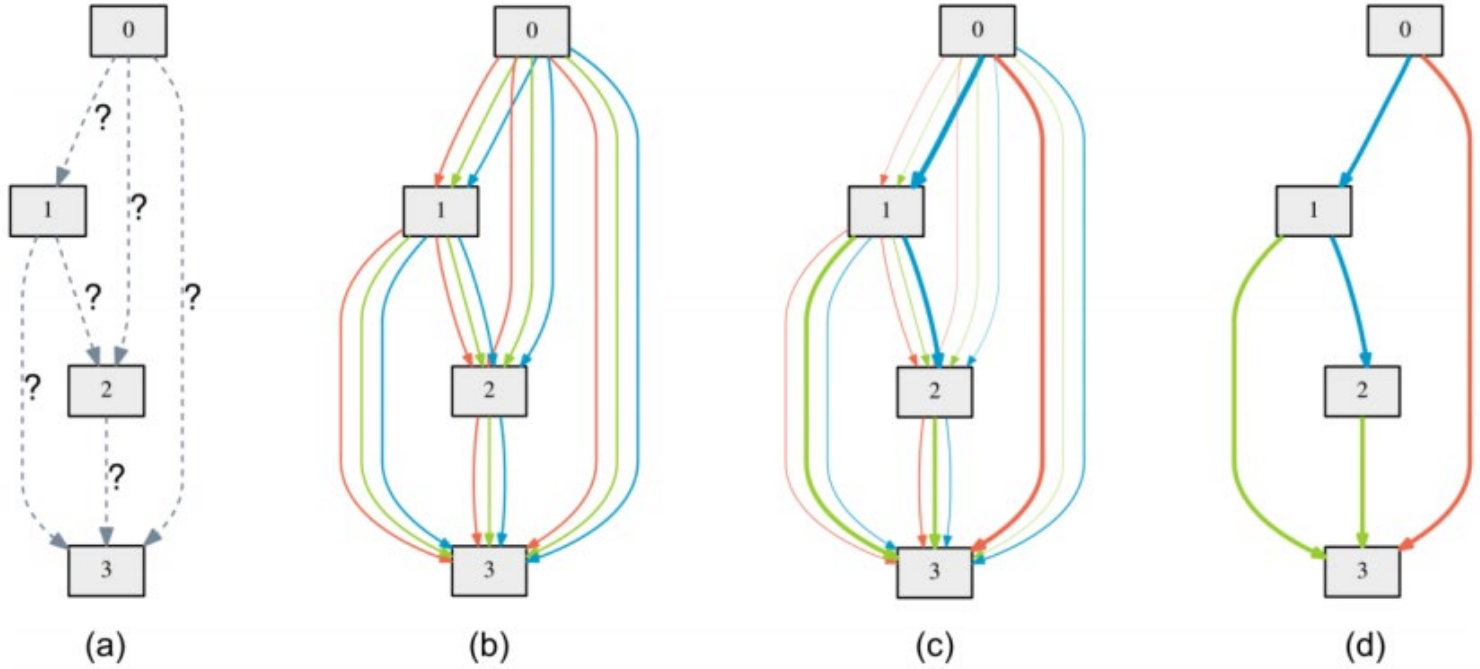


Structured Attention of DARTS

- Each intermediate node is computed based on all of its predecessors

$$x^{(j)} = \sum_{i<j} o^{(i,j)}(x^{(i)})$$

  - Progressively cut the edges having the lowest weight



Structured Attention of DARTS

- Each intermediate node is computed based on all of its predecessors

$$x^{(j)} = \sum_{i<j} o^{(i,j)}(x^{(i)})$$

- Let O be a set of candidate operations, the catorical choise of a particular operation is relaxed by a softmax over all possible operations whose importance is parameterized by $\alpha = \{\alpha^{(i,j)}\}$

$$\bar{o}^{(i,j)}(x) = \sum_{o \in \mathcal{O}} \frac{\exp(\alpha_o^{(i,j)})}{\sum_{o' \in \mathcal{O}} \exp(\alpha_{o'}^{(i,j)})} o(x)$$

- The problem is formulated by a bilevel optimization problem

$$\begin{aligned} \min_{\alpha} \quad & \mathcal{L}_{val}(w^*(\alpha), \alpha) \\ \text{s.t.} \quad & w^*(\alpha) = \text{argmin}_w \ \mathcal{L}_{train}(w, \alpha) \end{aligned}$$

Training Pipeline

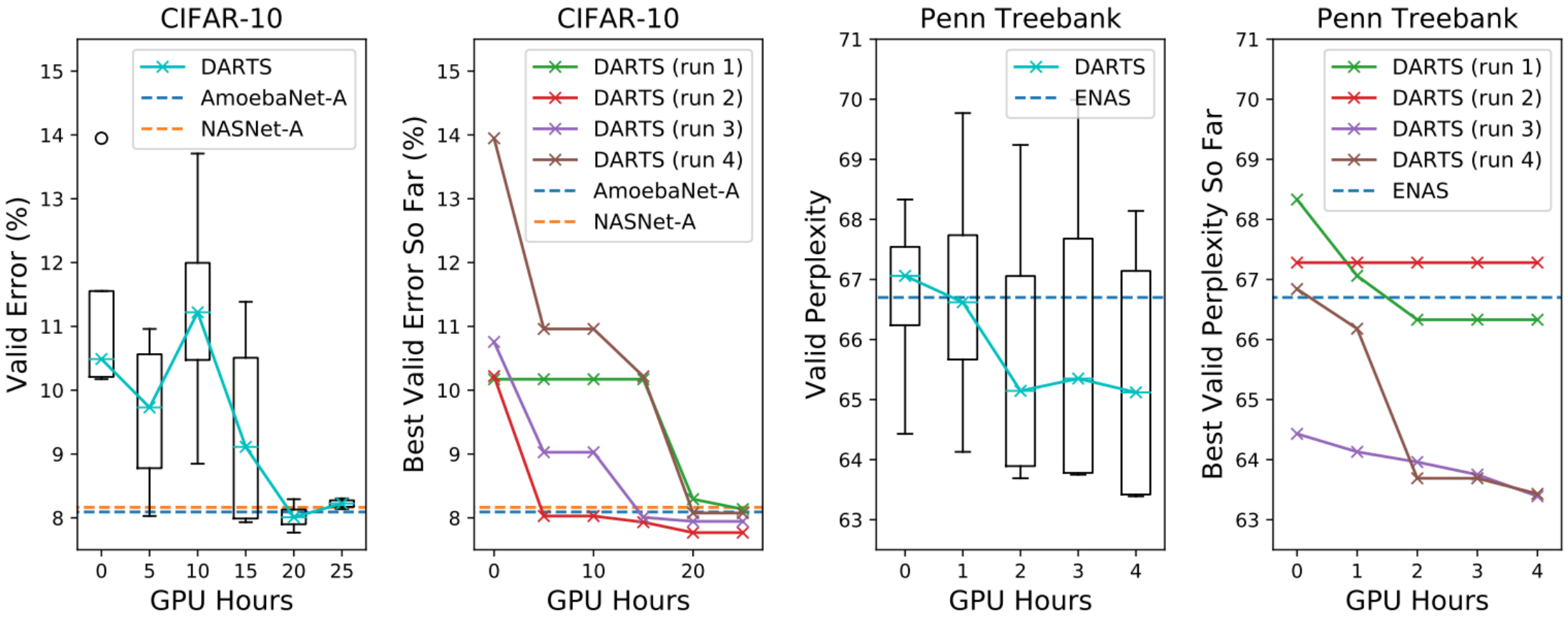
Algorithm 1: DARTS – Differentiable Architecture Search

Create a mixed operation $\bar{o}^{(i,j)}$ parametrized by $\alpha^{(i,j)}$ for each edge (i, j)
while *not converged* **do**
  1. Update architecture α by descending $\nabla_\alpha \mathcal{L}_{val}(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha)$
     ($\xi = 0$ if using first-order approximation)
  2. Update weights w by descending $\nabla_w \mathcal{L}_{train}(w, \alpha)$

Derive the final architecture based on the learned α.

1. Prepare a base model having all weights for every operations
2. Update architecture α using a validation set
   - Please note that this validation set is not the same as the validation used to report the accuracy
3. Update weights of the network using a training set
4. Cut the unimportant connections (low α) and drive the final architecture

Results


CIFAR-10
Valid Error (%)
GPU Hours
DARTS
AmoebaNet-A
NASNet-A
CIFAR-10
Best Valid Error So Far (%)
GPU Hours
DARTS (run 1)
DARTS (run 2)
DARTS (run 3)
DARTS (run 4)
AmoebaNet-A
NASNet-A
Penn Treebank
Valid Perplexity
GPU Hours
DARTS
ENAS
Penn Treebank
Best Valid Perplexity So Far
GPU Hours
DARTS (run 1)
DARTS (run 2)
DARTS (run 3)
DARTS (run 4)
ENAS

Results

Table 1: Comparison with state-of-the-art image classifiers on CIFAR-10 (lower error rate is better). Note the search cost for DARTS does not include the selection cost (1 GPU day) or the final evaluation cost by training the selected architecture from scratch (1.5 GPU days).

Architecture	Test Error (%)	Params (M)	Search Cost (GPU days)	#ops	Search Method
DenseNet-BC (Huang et al., 2017)	3.46	25.6	–	–	manual
NASNet-A + cutout (Zoph et al., 2018)	2.65	3.3	2000	13	RL
NASNet-A + cutout (Zoph et al., 2018)†	2.83	3.1	2000	13	RL
BlockQNN (Zhong et al., 2018)	3.54	39.8	96	8	RL
AmoebaNet-A (Real et al., 2018)	3.34 ± 0.06	3.2	3150	19	evolution
AmoebaNet-A + cutout (Real et al., 2018)†	3.12	3.1	3150	19	evolution
AmoebaNet-B + cutout (Real et al., 2018)	2.55 ± 0.05	2.8	3150	19	evolution
Hierarchical evolution (Liu et al., 2018b)	3.75 ± 0.12	15.7	300	6	evolution
PNAS (Liu et al., 2018a)	3.41 ± 0.09	3.2	225	8	SMBO
ENAS + cutout (Pham et al., 2018b)	2.89	4.6	0.5	6	RL
ENAS + cutout (Pham et al., 2018b)*	2.91	4.2	4	6	RL
Random search baseline‡ + cutout	3.29 ± 0.15	3.2	4	7	random
DARTS (first order) + cutout	3.00 ± 0.14	3.3	1.5	7	gradient-based
DARTS (second order) + cutout	2.76 ± 0.09	3.3	4	7	gradient-based

* Obtained by repeating ENAS for 8 times using the code publicly released by the authors. The cell for final evaluation is chosen according to the same selection protocol as for DARTS.

† Obtained by training the corresponding architectures using our setup.

‡ Best architecture among 24 samples according to the validation error after 100 training epochs.

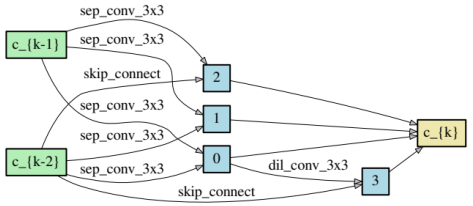


Figure 4: Normal cell learned on CIFAR-10.

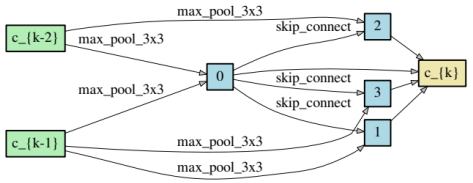


Figure 5: Reduction cell learned on CIFAR-10.

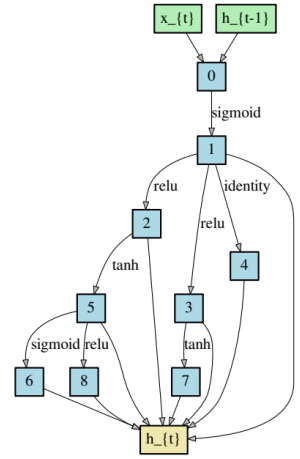


Figure 6: Recurrent cell learned on PTB.

Discussion

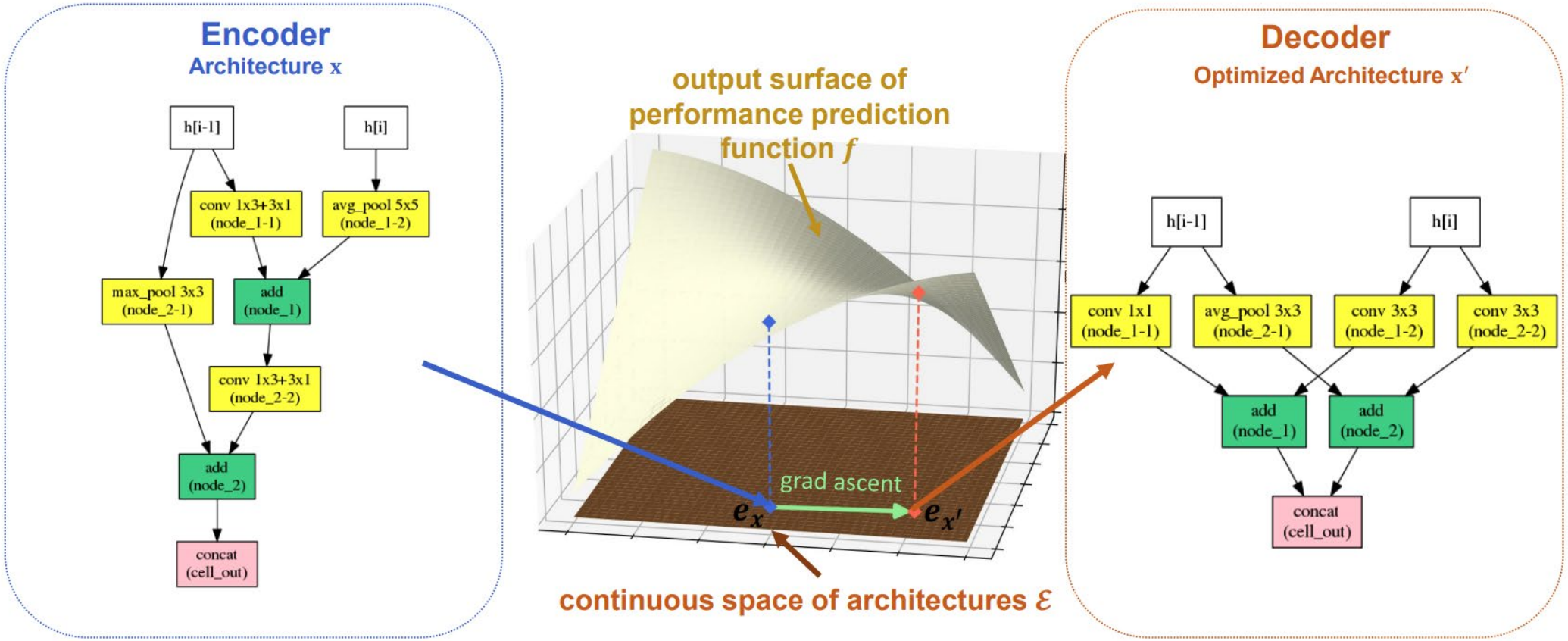
- Pros
  - Intuitive method
  - Simple implementation
  - Searched architecture shows good result

- Cons
  - Dependent on the initial condition
  - All operators are needed to be evaluated
    - Large memory footprint
    - Expensive computation overhead

Neural Architecture Optimization

Key Idea

- Training encoder/decoder that transfer a neural network from/to embedding space.
- Gradient ascent in continuous embedding space

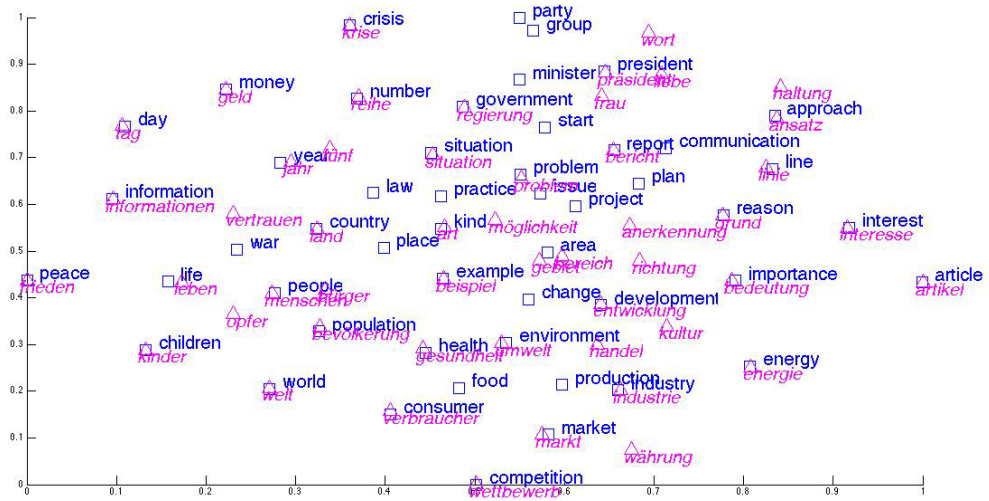


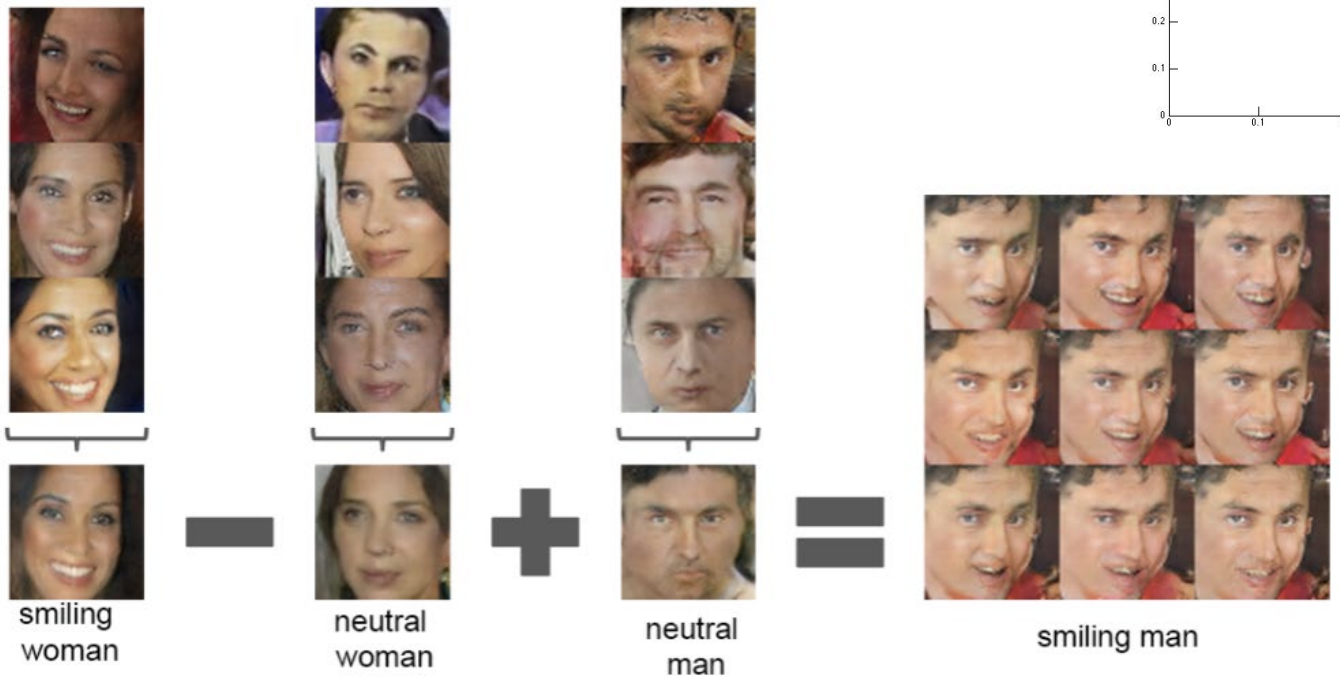
Components of NAO

- Encoder
  - Takes the sequence of architecture configuration and map it into a continuous space $\mathcal{E}$ $E: \mathcal{X} \rightarrow \mathcal{E}$
  - A single layer LSTM as the basic model of encoder and the hidden states of the LSTM are used as the continuous representation of x. $e_x = \{h_1, h_2, \ldots, h_T\} \in R^{T \times d}$
- Performance Predictor
  - It maps the embedding e_x of architecture x into its performance s_x
  - $f(\bar{e}_x) = f(E[h_t]) = s_x, \;\; L_{pred} = \left(s_x - f(\bar{e}_x)\right)$
- Decoder
  - Decode out the string token in x from e_x
  - LSTM with attention
  - Goal is to recover the original architecture configuration

$$P_D(x|e_x) = \Pi_r P_D(x_r|e_x, x_{<r})$$

Example – Embedding Space





Training & Search

- Jointly train the encoder, performance predictor and decoder with a combination loss
  - $L = \lambda L_{pp} + (1-\lambda)L_{rec} = \lambda \Sigma_x \left(s_x - f\left(E(x)\right)\right)^2 - (1-\lambda)\Sigma_x \log P_D(x|E(x))$
  - Performance predictor can be act as a regularizer for the encoder unit
- Progressively searching the best architecture by the gradient induced by f
  - $h_t' = h_t + \eta \frac{\partial f}{\partial h_t}$

Algorithm 1 Neural Architecture Optimization

Input: Initial candidate architectures set X to train NAO model. Initial architectures set to be evaluated denoted as $X_{eval} = X$. Performances of architectures $S = \emptyset$. Number of seed architectures K. Step size η. Number of optimization iterations L.
for $l = 1, \cdots, L$ **do**
  Train each architecture $x \in X_{eval}$ and evaluate it to obtain the dev set performances $S_{eval} = \{s_x\}, \forall x \in X_{eval}$. Enlarge S: $S = S \bigcup S_{eval}$.
  Train encoder E, performance predictor f and decoder D by minimizing Eqn.(1), using X and S.
  Pick K architectures with top K performances among X, forming the set of seed architectures X_{seed}.
  For $x \in X_{seed}$, obtain a better representation $e_{x'}$ from $e_{x'}$ using Eqn. (2), based on encoder E and performance predictor f. Denote the set of enhanced representations as $E' = \{e_{x'}\}$.
  Decode each x' from $e_{x'}$ using decoder, set X_{eval} as the set of new architectures decoded out: $X_{eval} = \{D(e_{x'}), \forall e_{x'} \in E'\}$. Enlarge X as $X = X \bigcup X_{eval}$.
end for
Output: The architecture within X with the best performance

Results

Model	B	N	F	#op	Error (%)	#params
DenseNet-BC [20]	/	100	40	/	17.18	25.6M
Shake-shake [15]	/	/	/	/	15.85	34.4M
Shake-shake + Cutout [11]	/	/	/	/	15.20	34.4M
NASNet-A [51]	5	6	32	13	19.70	3.3M
NASNet-A [51] + Cutout	5	6	32	13	16.58	3.3M
NASNet-A [51] + Cutout	5	6	128	13	16.03	50.9M
PNAS [27]	5	3	48	8	19.53	3.2M
PNAS [27] + Cutout	5	3	48	8	17.63	3.2M
PNAS [27] + Cutout	5	6	128	8	16.70	53.0M
ENAS [37]	5	5	36	5	19.43	4.6M
ENAS [37] + Cutout	5	5	36	5	17.27	4.6M
ENAS [37] + Cutout	5	5	36	5	16.44	52.7M
AmoebaNet-B [39]	5	6	128	19	17.66	34.9M
AmoebaNet-B [39] + Cutout	5	6	128	19	15.80	34.9M
NAONet + Cutout	5	6	36	11	15.67	10.8M
NAONet + Cutout	5	6	128	11	14.75	128M

Table 2: Performances of different CNN models on CIFAR-100 dataset. 'NAONet' represents the best architecture discovered by NAO on CIFAR-10.

Results

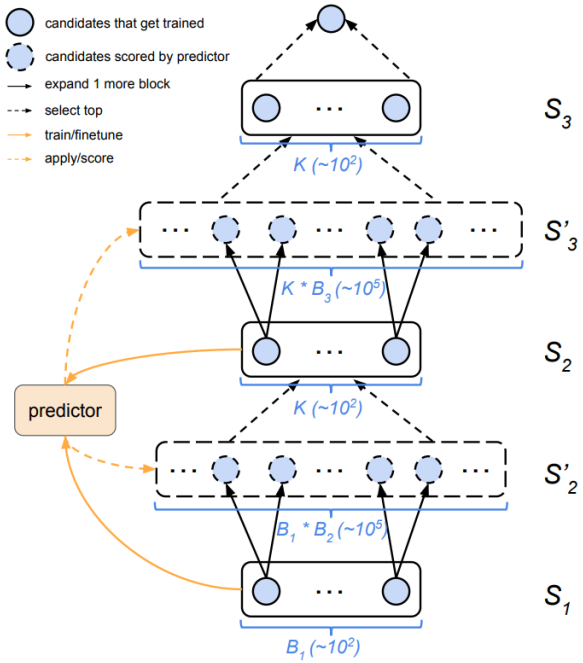
Model	Top-1(%)	Top-5(%)	Params(M)	Mult-Adds (M)
Inception-v1 [43]	30.2	10.1	6.6	1448
MobileNet [18]	29.4	10.5	4.2	569
ShuffleNet 2× (v1) [49]	29.1	10.2	~ 5	524
ShuffleNet 2× (v2) [49]	26.3	-	~ 5	524
NASNet-A [51]	26.0	8.4	5.3	564
NASNet-B [51]	27.2	8.7	5.3	488
NASNet-C [51]	27.5	9.0	4.9	558
AmoebaNet-A [39]	25.5	8.0	5.1	555
AmoebaNet-B [39]	26.0	8.5	5.3	555
AmoebaNet-C [39]	24.3	7.6	6.4	570
PNAS [27]	25.8	8.1	5.1	588
DARTS [29]	26.9	9.0	4.9	595
NAONet	25.7	8.2	11.35	584

Table 3: Performances of different CNN models on ImageNet dataset. 'NAONet' represents the best architecture discovered by NAO on CIFAR-10.

Neural Predictor for Neural Architecture Search

Performance Prediction?

- In Progressive NAS, performance predictor is used to predict the accuracy of the network with a progressive update
  - What if we depend solely on the performance predictor solely for the architecture search?



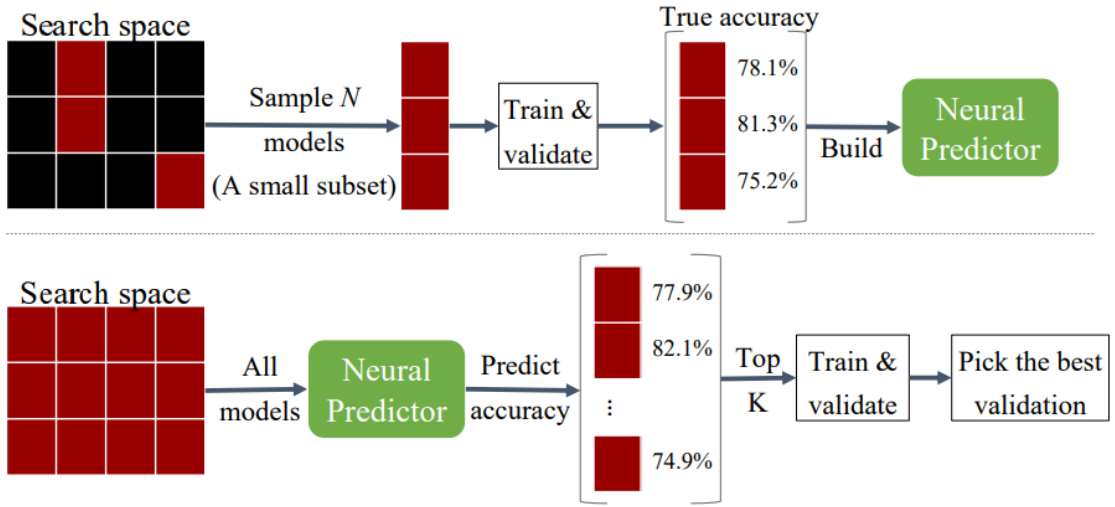
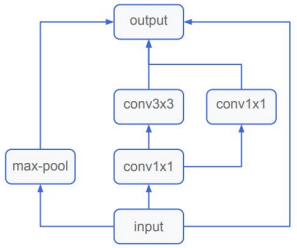
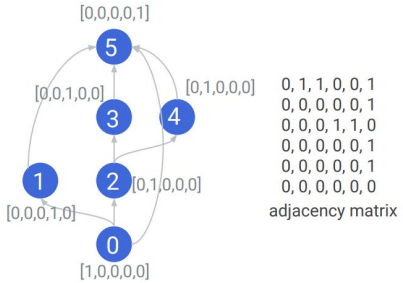


Figure 1: Training and applying the Neural Predictor.

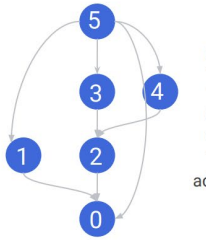
Neural Predictor Design



A cell or a network
Operations: [input, conv1x1, conv3x3, max-pool, output]



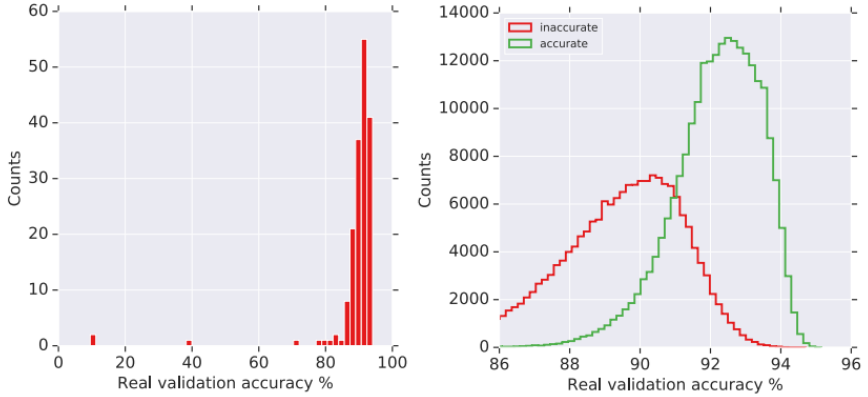
0, 1, 1, 0, 0, 1
0, 0, 0, 0, 0, 1
0, 0, 0, 1, 1, 0
0, 0, 0, 0, 0, 1
0, 0, 0, 0, 0, 1
0, 0, 0, 0, 0, 0
adjacency matrix



0, 0, 0, 0, 0, 0
1, 0, 0, 0, 0, 0
1, 0, 0, 0, 0, 0
0, 0, 1, 0, 0, 0
0, 0, 1, 0, 0, 0
1, 1, 0, 1, 1, 0
adjacency matrix transpose

- Graph Convolution Networks based Modeling
  - Starting from an initial feature vector $V_0 \in R^{I \times D_0}$
  - With a trainable weight matrix $W_l \in R^{D_l \times D_{l+1}}$ and adjacency matrix $A \in R^{I \times I}$, iteratively update the node representation
    - $V_{l+1} = ReLU(AV_lW_l)$
    - $V_{l+1} = \frac{1}{2}ReLU(AV_lW_l^+) + \frac{1}{2}ReLU(AV_lW_l^-)$ for bidirectional information flow
  - Stack these networks several layers and use the average of the final representation

Training & Search Pipeline



- Training neural predictor with 172 models
  - The sampled network contains many bad and unstable networks
  - Use binary GCN classifier to filter out the bad networks
    - Networks less than 91 % of accuracy are ignored
  - GCN regressor is used to predict the validation accuracy

- Greedy search and validation
  - About 800 samples having the highest predicted accuracy are evaluated
  - Different view: rough estimation / precise evaluation

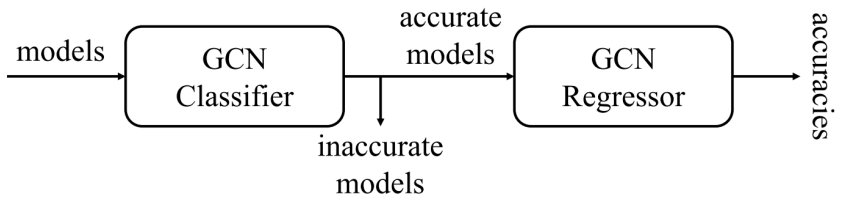
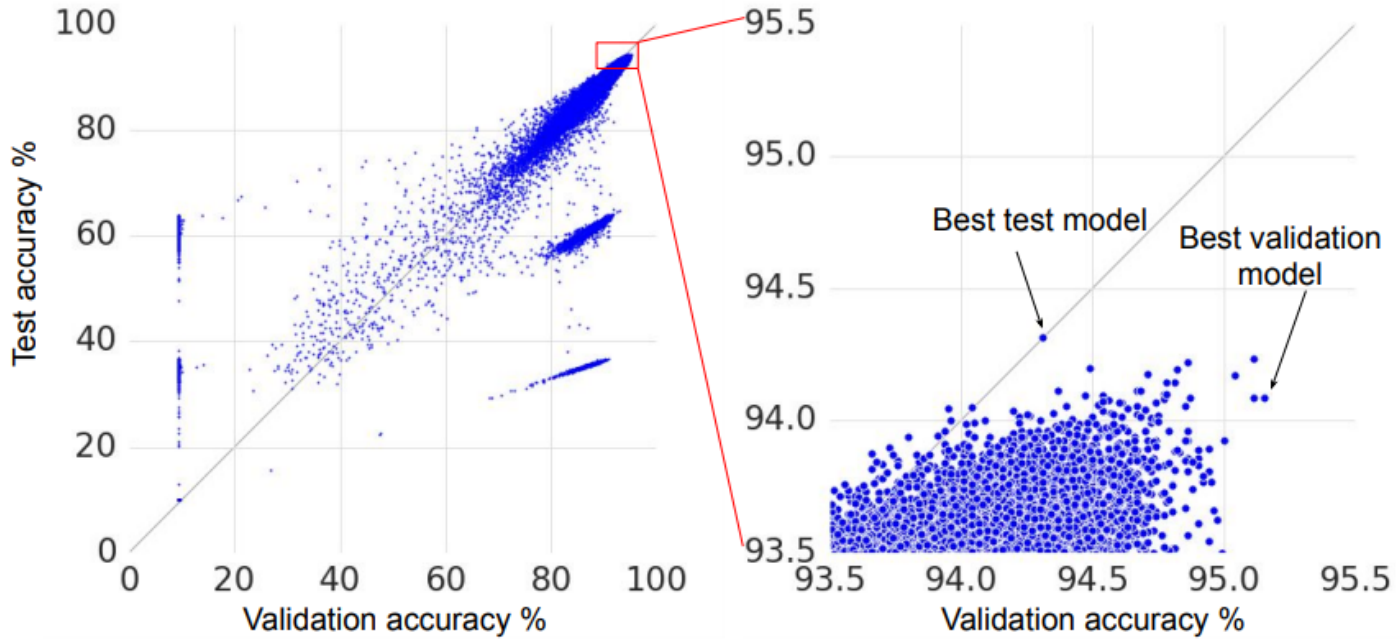


Figure 4: Neural Predictor on NASBench-101. It is a cascade of a classifier and a regressor. The classifier filters out inaccurate models and the regressor predicts accuracies of accurate models.

Results



- It is not possible to predict which model will perform best on the test set based on the validation accuracy

Results

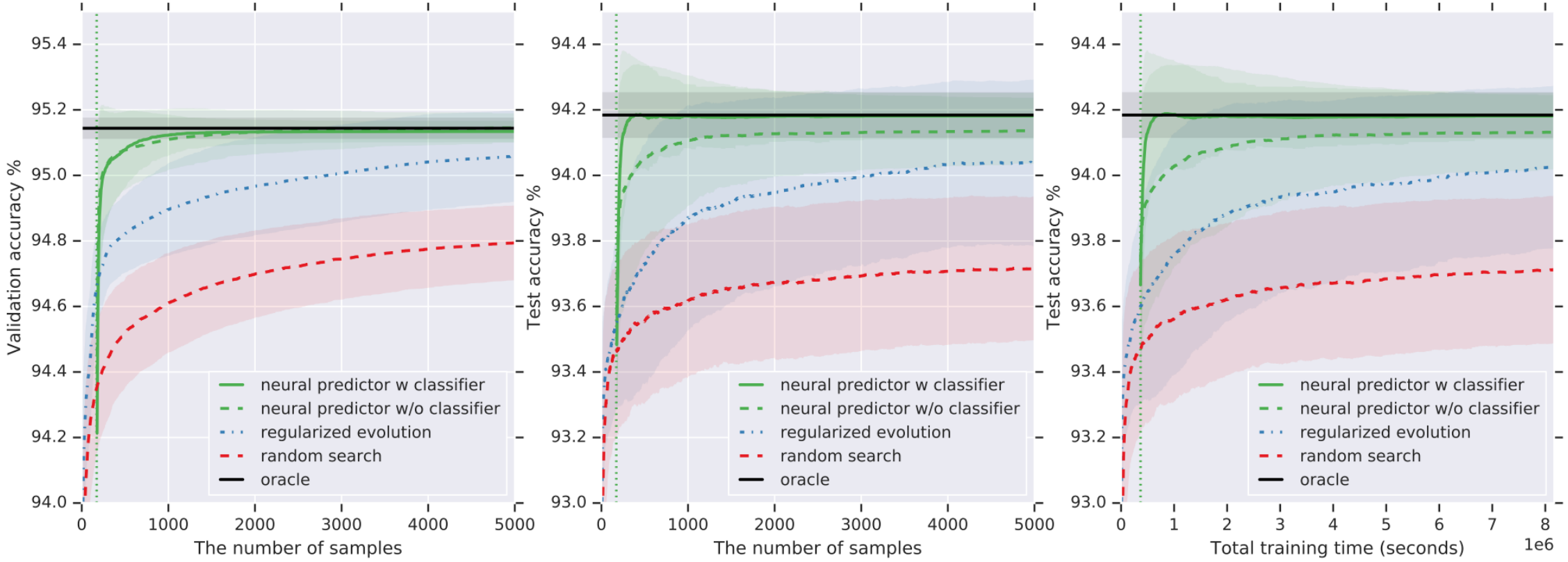


Figure 6: The comparison of search efficiency among oracle, random search, Regularized Evolution and our Neural Predictor (with and without 2 stage regressor). All experiments are averaged over 600 runs. The x-axis represents the total compute budget $N + K$. The vertical dotted line is at $N = 172$ and represents the number of samples (or total training time) used to build our Neural Predictor. From this line on we start from $K = 1$ and increase it as we use more architectures for final validation. The shaded region indicates standard deviation of each search method.

Results

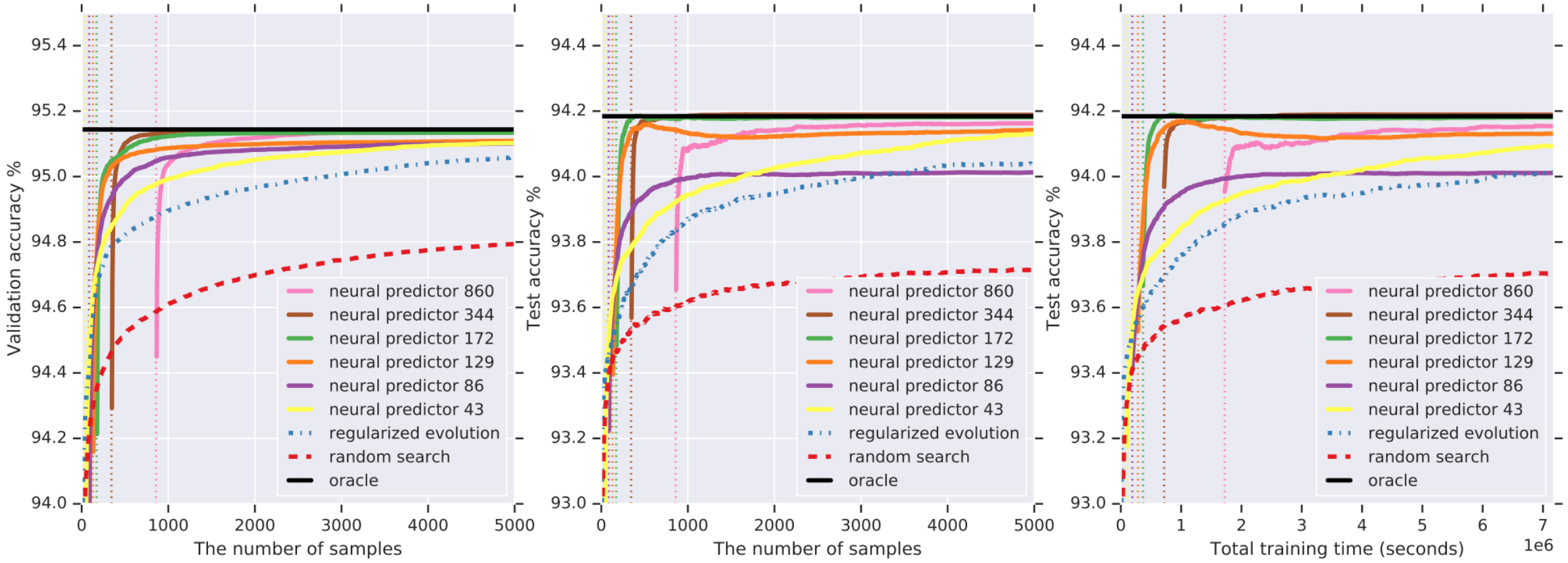


Figure 7: Analysis of the trade-off between N training samples vs K final validation samples in the neural predictor. The x-axis is the total compute budget $N + K$. The vertical lines indicate different choices for N, the number of training samples and the point where we start validating K models. All experiments are averaged over 600 runs.