Improved Gradient-Based Neural Architecture Search

Zhexuan HUANG
Supervised by: Bruno CONCHE

July 10, 2019

Overview

Introduction

Motivation of Neural Architecture Search (NAS)

- Designing neural network architectures is hard
- A lot of intuition and possibilities to design them
- Can we learn good architectures automatically?

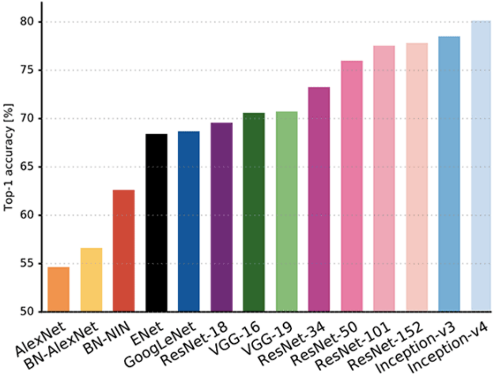
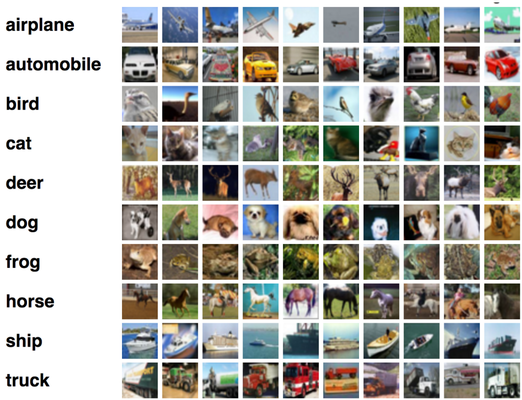


Image classification over CIFAR10

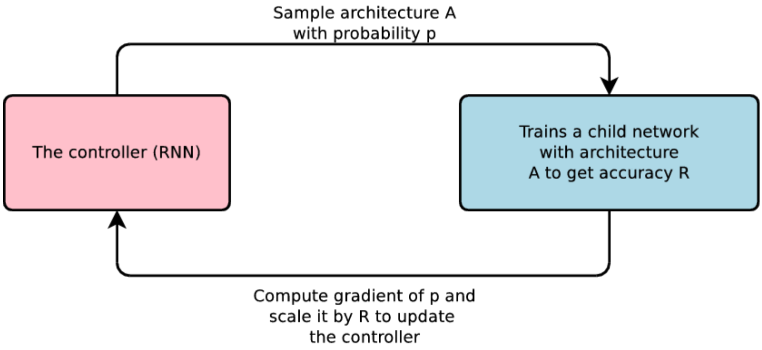
- CIFAR-10 dataset consists of 60,000 32x32 colour images in 10 classes
- To have a better comparaison since many NAS researches use CIFAR10 as dataset



State-of-the-art Neural Architecture Search (NAS) strategies

Original Neural Architecture Search (NAS)

- Architecture Search algorithm proposed by Google in 2016 (Zoph and Le, 2016, Neural Architecture Search with Reinforcement Learning)
- Use un controller(RNN) to generate features of Neural Networks
- Train each NN to converge and return its accuracy to controller as rewards, update controller with reinforce policy



Result of NAS

Author's implementation

Used 1800 GPU days to search a model with 2.65% error rates over CIFAR10

Analysis

- Reinforcement Learning(RL) based algorithm
- In each controller loop, need to train a child network to converge to get accuracy R
- Computational expensive and time consuming

List of different state-of-the-art NAS strategies

Reinforcement Learning based

- NAS(Zoph and Le, 2016), 1800 GPU days
- ENAS(Pham et al., 2018), 0.5 GPU days

Evolution based

- AmoebaNet(Real et al., 2019), 3150 GPU days
- Hierarchical Evolution(H. Liu, Simonyan, Vinyals, et al., 2017), 300 GPU days

Bayesian optimization

- PNAS(C. Liu et al., 2018), 225 GPU days

Gradient based

- DARTS(H. Liu, Simonyan, and Yang, 2019), 1 GPU days
- SNAS(Xie et al., 2019), 1.5 GPU days

Search space of Gradient based NAS

- Search space: set of all candidate architectures
- The same search space as in NAS, ENAS, DARTS and SNAS
- Search for computation cells, which is represented by a directed acyclic graph(DAG)
- Each edge (i, j) in DAG is associated with some operation $\tilde{O}_{i,j}$. The possible choice for $\tilde{O}_{i,j}$ is given in priority (e.g. Conv 3×3, Maxpool 3×3, None, Identity, etc).

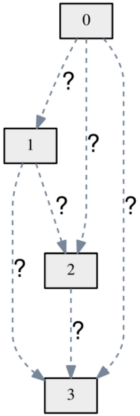


Figure: DAG, image retrieved from (H. Liu, Simonyan, and Yang, 2019)

Search space of Gradient based NAS

- Two kinds of computation cells: normal cell and reduction cell
- Whole architecture is obtained by stacking two kinds of cell together

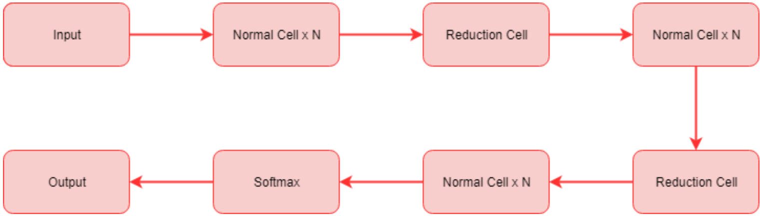
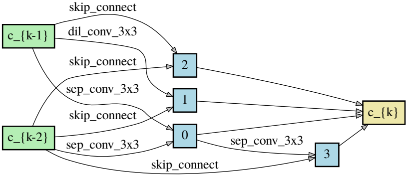
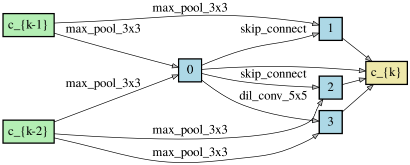


Figure: A conceptual of global structure.

Examples of cells



(a) Normal cell found by SNAS



(b) Reduction cell found by SNAS

Figure: Normal cell and reduction cell (child graph) found by SNAS on CIFAR-10 (Xie et al., 2019) (a) Normal cell. (b) Reduction cell.

Reformulate of problem with stochastic modeling

- As defined in SNAS, each $\tilde{O}_{i,j}$ is probabilistic:

$$\tilde{O}_{i,j} = \begin{cases} O_{i,j}^1 & \textbf{with probability } p_{i,j}^1 = \frac{\exp(\alpha_{i,j}^1)}{\sum_{k=1}^A \exp(\alpha_{i,j}^k)} \\ \dots & \\ O_{i,j}^A & \textbf{with probability } p_{i,j}^A = \frac{\exp(\alpha_{i,j}^A)}{\sum_{k=1}^A \exp(\alpha_{i,j}^k)}, \end{cases}$$

where $\alpha = (\alpha_{i,j}^k)_{i<j,k=1,\dots,A}$ are architecture parameters.

- Equivalent to $\tilde{O}_{i,j} = Z_{i,j} O_{i,j}$ where

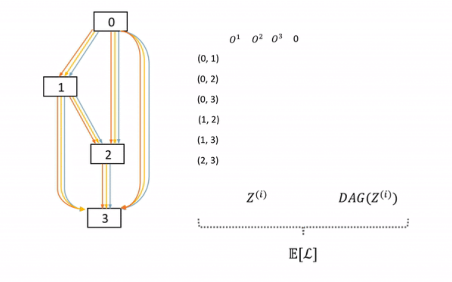
$$Z_{i,j} = \begin{cases} [1, 0, \dots, 0] & \textbf{with probability } p_{i,j}^1 = \frac{\exp(\alpha_{i,j}^1)}{\sum_{k=1}^A \exp(\alpha_{i,j}^k)} \\ \dots & \\ [0, 0, \dots, 1] & \textbf{with probability } p_{i,j}^A = \frac{\exp(\alpha_{i,j}^A)}{\sum_{k=1}^A \exp(\alpha_{i,j}^k)}, \end{cases}$$

Objective

- Two kinds of parameters: architecture parameters α and operation parameters θ
- Objective is to minimize the expected loss:

$$\min_{\alpha,\theta} \mathbb{E}_{Z\sim p_\alpha(Z)}[L_\theta(Z)], \tag{1}$$

  which can be estimated by Monte-Carlo sampling.
- A conceptual of calculation loss retrieved from Xie et al., 2019



Gradient estimators: Overview

- The objective $\min_{\alpha,\theta} \mathbb{E}_{Z\sim p_\alpha(Z)}[L_\theta(Z)]$ can be optimized with gradient-descent algorithm
- Impossible to calculate the exact gradients
- $\nabla_\theta \mathbb{E}_{Z\sim p_\alpha(Z)}[L_\theta(Z)] = \mathbb{E}_{Z\sim p_\alpha(Z)}[\nabla_\theta L_\theta(Z)]$ can be estimated by Monte-Carlo sampling
- Different techniques to estimate $\nabla_\alpha \mathbb{E}_{Z\sim p_\alpha(Z)}[L_\theta(Z)]$

Gradient estimators: Score Function (SF) Estimators

Also known as REINFORCE(J.Williams, 1992), based on identity:

$$\nabla_\alpha \mathbb{E}_{Z \sim p_\alpha(Z)}[L_\theta(Z)] = \mathbb{E}_{Z \sim p_\alpha(Z)}[L_\theta(Z) \nabla_\alpha \log p_\alpha(Z)]. \tag{2}$$

Analysis

- Unbiased estimator
- Depends only on the final result of $L_\theta(Z)$
- Not require to calculate the back-propagation
- Extremly high variance

Gradient estimators: Reparameterization Trick

If Z can be rewrite as $Z = g(\alpha, \epsilon)$ where $\epsilon \sim p_\epsilon$, then

$$\nabla_\alpha \mathbb{E}_{Z \sim p_\alpha(Z)}[L_\theta(Z)] = \mathbb{E}_{\epsilon \sim p_\epsilon}[L'_\theta(g(\alpha, \epsilon))\nabla_\alpha g(\alpha, \epsilon)]. \tag{3}$$

Example

If $X \sim \mathcal{N}(\mu, \sigma^2)$ then $\nabla_\sigma \mathbb{E}[f(X)] = \mathbb{E}[f'(\mu + \sigma N)N]$ for $N \sim \mathcal{N}(0, 1)$.

Analysis

- Unbiased, low variance, better than SF
- Only applicable for random variable which is reparameterizable
- Need to calculate the back-propagation

Gradient estimators: SNAS and Gumbel-Softmax

Main difficulty to use reparameterization trick

Z in our case is discrete random variable and is not reparameterizable.

Solution proposed in SNAS

- Relax Z to $Z \longrightarrow \tilde{Z}$ by Gumbel-Softmax, where

$$\tilde{Z}_{i,j}^{k} = g_{i,j}(\alpha, U) = \frac{\exp((\alpha_{i,j}^{k} - \log(-\log(U_{i,j}^{k})))/\lambda)}{\sum_{l=1}^{A} \exp((\alpha_{i,j}^{l} - \log(-\log(U_{i,j}^{l})))/\lambda)}, \tag{4}$$

  $U = \{U_{i,j}^{k}\}_{i,j,k}$ are some independent uniform random variables, λ is the temperature of the Gumbel softmax, which is annealed to zero in SNAS.

- Minimize the approximated loss function with reparameterization:

$$\min_{\alpha,\theta} \mathbb{E}_{\tilde{Z}\sim\tilde{p}_{\alpha}(\tilde{Z})}[L_{\theta}(\tilde{Z})] = \min_{\alpha,\theta} \mathbb{E}_{U}[L_{\theta}(g(\alpha, U))], \tag{5}$$

Gradient estimators: Intuition of Gumbel-Softmax

Proposition 1

$\mathbb{P}(\tilde{Z}_{i,j}^k > \tilde{Z}_{i,j}^l \textit{ for } l \neq k) = \frac{\exp(\alpha_{i,j}^k)}{\sum_{l=1}^{A} \exp(\alpha_{i,j}^l)}$.

Thus $Z_{i,j}^k$ can be obtained by taking arg max operation over $\tilde{Z}_{i,j}^k$, i.e.

$$Z_{i,j}^k = \begin{cases} 1 & \textbf{if } k = \arg\max_{l=1,\dots,A}\{\tilde{Z}_{i,j}^l\} \\ 0 & \textbf{otherwise} \end{cases} \tag{6}$$

$$\tilde{Z}_{i,j} = [0.1, 0.3, 0.1, 0.4, 0.05, 0.05] \longrightarrow Z_{i,j} = [0, 0, 0, 1, 0, 0] \tag{7}$$

Proposition 2

$\tilde{Z}_{i,j}^k$ converge to $Z_{i,j}^k$ in distribution when $\lambda \to 0$, i.e.

$\mathbb{P}(\lim_{\lambda \to 0} \tilde{Z}_{i,j}^k = 1) = \frac{\exp(\alpha_{i,j}^k)}{\sum_{l=1}^{A} \exp(\alpha_{i,j}^l)}$.

Gradient estimators: Intuition of Gumbel-Softmax

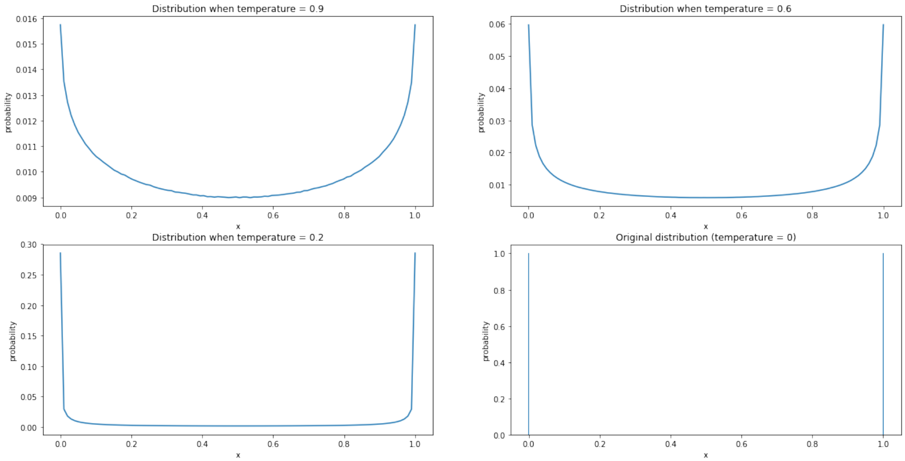


Figure: A visualization of distributions of Gumbel Softmax with different temperatures. Temperature $\lambda = 0$ correspond to original discrete distribution without relaxation. $\tilde{Z}^k_{i,j}$ becomes sharper and converge to discrete distribution as $\lambda \to 0$.

Gradient estimators: SNAS and Gumbel-Softmax

Analysis

- The gradient of the new objective $\mathbb{E}_U[L_\theta(g(\alpha, U))]$ can be estimated by calculating back-propagation with automatic differentiation libraries
- Biased estimator due to changes of objective
- Low variance thanks to reparameterization

NAS with unbiased and low variance gradient estimators

Gradient estimators: SF estimators with Control Variates

Control Variates

In SF estimator, the gradient is estimated using

$$\nabla_\alpha \mathbb{E}_{Z\sim p_\alpha(Z)}[L_\theta(Z)] = \mathbb{E}_{Z\sim p_\alpha(Z)}[L_\theta(Z)\nabla_\alpha \log p_\alpha(Z)]. \tag{8}$$

Alternatively, the gradient can also be estimated by

$$\nabla_\alpha \mathbb{E}_{Z\sim p_\alpha(Z)}[L_\theta(Z)] = \mathbb{E}_{Z\sim p_\alpha(Z)}[(L_\theta(Z) - c(Z))\nabla_\alpha \log p_\alpha(Z)] + \mathbb{E}_{Z\sim p_\alpha(Z)}[c(Z)\nabla_\alpha \log p_\alpha(Z)], \tag{9}$$

for some function $c(Z)$.

Analysis

- Lower variance than ordinary SF estimator if $c(Z)$ is positively correlated with $L_\theta(Z)\nabla_\alpha \log p_\alpha(Z)$.
- Require knoledge of $\mathbb{E}_{Z\sim p_\alpha(Z)}[c(Z)\nabla_\alpha \log p_\alpha(Z)]$

Gradient estimators: SF estimators with Control Variates

Constant Control Variates

If we take $c(Z) = c$ which is constant, then the estimation can be written as

$$\nabla_\alpha \mathbb{E}_{Z \sim p_\alpha(Z)}[L_\theta(Z)] = \mathbb{E}_{Z \sim p_\alpha(Z)}[(L_\theta(Z) - c)\nabla_\alpha \log p_\alpha(Z)]. \tag{10}$$

A common used technique is to take c as the moving average of $L_\theta(Z)$ in each iteration.

We have implemented SF estimator with constant control variates and call it SF in our later experiment.

Analysis

- Significantly reduce the variance in practice
- Little additional computational cost compare to ordinary version of SF estimator (additional computation is to calculate the moving average)
- Unbiased estimator

Gradient estimators: RELAX estimators

(Grathwohl et al., 2018) proposed an estimator which they call RELAX, to combine the SF estimator, reparameterization trick and control variates.

RELAX estimators

RELAX estimators are based on

$$\nabla_\alpha \mathbb{E}_{Z\sim p_\alpha(Z)}[L_\theta(Z)] = \mathbb{E}_{U,\tilde{Z}_{cond}}[(L_\theta(Z) - c_\phi(\tilde{Z}_{cond}))\nabla_\alpha \log p_\alpha(Z)] + \mathbb{E}_{U,\tilde{Z}_{cond}}[\nabla_\alpha(c_\phi(\tilde{Z}) - c_\phi(\tilde{Z}_{cond}))] \tag{11}$$

where $\tilde{Z}$ is Gumbel-Softmax variable, $\tilde{Z}_{cond} \sim \tilde{Z}|Z$ and c_ϕ is a neural network which is trained in each iteration to minimize the variance of the estimator.

Gradient estimators: Finite-Difference estimators (RAM)

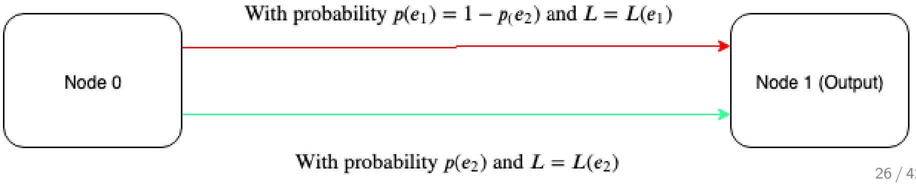
Estimator proposed by (Tokui and sato, 2016), which they call RAM (Reparameterization and Marginalization) estimator

A simple example

A single binary stochastic variable (i.e. a DAG with only two nodes: input node and output node, with two candidate operations: $\{O^1, O^2\}$). We denote $e_1 = [1, 0]$ and $e_2 = [0, 1]$ the ith entry vector. Then we can calculate the expected loss function as:

$$\mathbb{E}_{Z \sim p_\alpha(Z)}[L_\theta(Z)] = p(e_2)L_\theta(e_2) + (1 - p(e_2))L_\theta(e_1), \tag{12}$$

where $p(e_1) = \mathbb{P}(Z = e_1)$ and $p(e_2) = \mathbb{P}(Z = e_2)$.



Gradient estimators: Finite-Difference estimators (RAM)

Its gradient w.r.t. α can be calculated:

$$\nabla_\alpha \mathbb{E}_{Z\sim p_\alpha(Z)}[L_\theta(Z)] = \nabla_\alpha p(e_2)(L_\theta(e_2) - L_\theta(e_1)). \tag{13}$$

RAM for multi-node and multi-candidate operation

$$\nabla_\alpha \mathbb{E}_{Z\sim p_\alpha(Z)}[L_\theta(Z)] = \sum_{(i,j)\in E} \sum_{Z_{\backslash(i,j)}} p_{\backslash(i,j)}(Z_{\backslash(i,j)}) \times$$
$$\sum_{e_a,e_b} (\nabla_\alpha \alpha^a_{i,j}) p_{i,j}(e_a) p_{i,j}(e_b) [L_\theta(Z_{i,j}=e_a, Z_{\backslash(i,j)}) - L_\theta(Z_{i,j}=e_b, Z_{\backslash(i,j)})], \tag{14}$$

where $p_{i,j}$ is the marginal probability of $Z_{i,j}$, $Z_{\backslash(i,j)}$ represent Z without $Z_{i,j}$ and $p_{\backslash(i,j)}$ is the marginal probability of $Z_{\backslash(i,j)}$.

Gradient estimators: Finite-Difference estimators (RAM)

$$\sum_{(i,j)\in E} \sum_{Z_{\backslash(i,j)}} p_{\backslash(i,j)}(Z_{\backslash(i,j)}) \times$$
$$\sum_{e_a,e_b} (\nabla_\alpha \alpha^a_{i,j}) p_{i,j}(e_a) p_{i,j}(e_b)[L_\theta(Z_{i,j} = e_a, Z_{\backslash(i,j)}) - L_\theta(Z_{i,j} = e_b, Z_{\backslash(i,j)})]. \tag{15}$$

Sampling strategy

For each edge $(i,j) \in E$,

- sample $Z_{\backslash(i,j)}$
- calculate

$$\sum_{e_a,e_b} (\nabla_\alpha l^a_{i,j}) p_{i,j}(e_a) p_{i,j}(e_b)[L_\theta(Z_{i,j} = e_a, Z_{\backslash(i,j)}) - L_\theta(Z_{i,j} = e_b, Z_{\backslash(i,j)})] \tag{16}$$

for each e_a, e_b from candidate operation

Gradient estimators: Finite-Difference estimators (RAM)

Analysis

- Unbiased estimator
- Very low variance due to $L_\theta(Z_{i,j} = e_a, Z_{\setminus(i,j)}) - L_\theta(Z_{i,j} = e_b, Z_{\setminus(i,j)})$ are evaluated at the same $Z_{\setminus(i,j)}$
- Theoretically proved to be better (lower variance) than Score Function estimator with constant control variates (Tokui and sato, 2016)
- More computations compare to SF and RELAX

Gradient estimators conclusion

We have proposed 3 different estimators:

- Score Function estimators with constant control variates (SF)
- RELAX estimators (RELAX)
- RAM estimators (RAM)

Experiments

Experiments setting

- For a better comparison, use exactly the same setting as the other research papers: NAS, ENAS, DARTS, SNAS, etc.
- Search for two kinds of cells: Normal cell and Reduction cell
- Implement three kinds of strategies: SF with constant control variates, RELAX and RAM

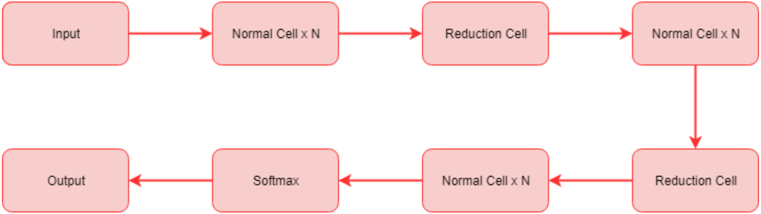
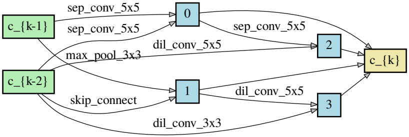
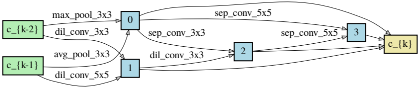


Figure: A conceptual of global structure. where reduction cells are located in 1/3 and 2/3 of the total depth of the neural network.

Results



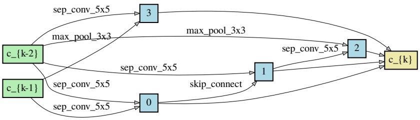
(a) Normal cell found by SF



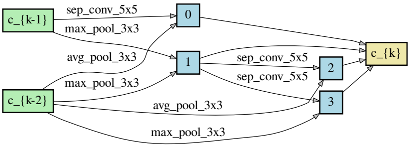
(b) Reduction cell found by SF

Figure: Normal cell and reduction cell (child graph) found by SF on CIFAR-10. (a) Normal cell. (b) Reduction cell.

Results



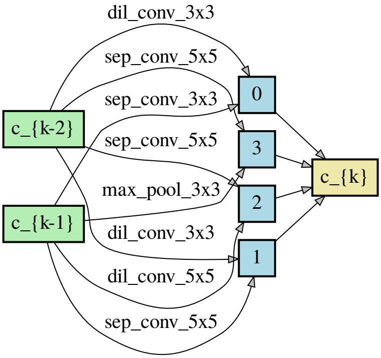
(a) Normal cell found by RELAX after restrain



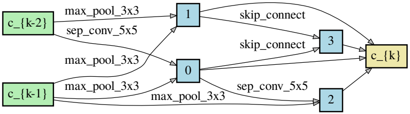
(b) Reduction cell found by RELAX after restrain

Figure: Normal cell and reduction cell (child graph) found by RELAX on CIFAR-10. (a) Normal cell. (b) Reduction cell.

Results



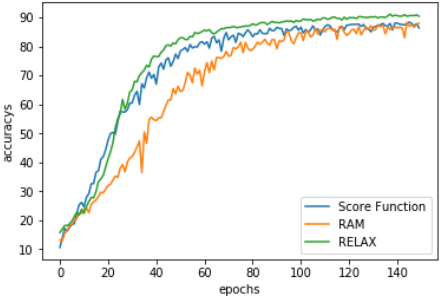
(a) Normal cell found by RAM



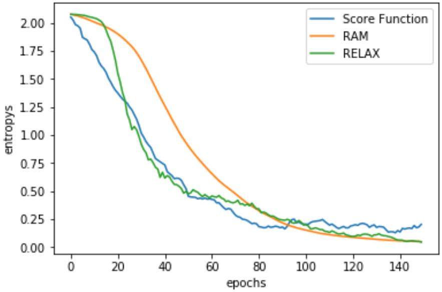
(b) Reduction cell found by RAM

Figure: Normal cell and reduction cell (child graph) found by RAM on CIFAR-10. (a) Normal cell. (b) Reduction cell.

Results



(a) Validation accuracy during architecture search with RAM, RELAX and Score Function estimator.



(b) Entropy of architecture distribution (i.e. $Z \sim p_\alpha(Z)$) during architecture search with RAM, RELAX and Score Function estimator.

Figure: Validation accuracy and entropy of architecture distribution during architecture search

Comparaison with other methods

Architecture	Test Error(%)	Params(M)	Search Cost (GPU days)	Search Method
NASNet-A[Zoph and Le, 2016]	2.65	3.3	1800	RL
AmoebaNet-A[Real et al., 2019]	3.34	3.2	3150	evolution
AmoebaNet-B[Real et al., 2019]	2.55	2.8	3150	evolution
Hierarchical Evo[Liu et al., 2017]	3.75	15.7	300	evolution
PNAS[Liu et al., 2018]	3.41	3.2	225	SMOB
ENAS[Pham et al., 2018]	2.89	4.6	0.5	RL
DARTS[Liu et al., 2019]	2.76	3.3	1	gradient-based
SNAS[Xie et al., 2019]	2.85	2.8	1.5	gradient-based
RAM(ours)	2.62	3.6	1.25	gradient-based
SF(ours)	2.64	3.4	0.4	gradient-based
RELAX(ours)	2.70	3.6	0.6	gradient-based

Figure: Classification errors of different estimators with other state-of-the-art image classifiers on CIFAR-10. All of our experiments are done using a single V100 GPU.

Conclusion

Conclusion

- Develop different gradient-based NAS strategies by introducing unbiased and low variance gradient estimators
- Low computational costs like other gradient-based NAS (around 1 GPU days)
- Results outperform other framework on the same search space

References I

Grathwohl, Will et al. (2018). "Backpropagation through the Void: Optimizing control variates for black-box gradient estimation". In: *arXiv:1711.00123v3 [cs.LG]*.

J.Williams, Ronald (1992). "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Reinforcement Learning, pp. 5-32*.

Liu, Chenxi et al. (2018). "Progressive Neural Architecture Search". In: *arXiv:1712.00559v3 [cs.CV]*.

Liu, Hanxiao, Karen Simonyan, Oriol Vinyals, et al. (2017). "Hierarchical Representations for Efficient Architecture Search". In: *arXiv:1711.00436v2 [cs.LG]*.

Liu, Hanxiao, Karen Simonyan, and Yiming Yang (2019). "DARTS: Differentiable Architecture Search". In: *arXiv:1806.09055v2 [cs.LG]*.

Pham, Hieu et al. (2018). "Efficient Neural Architecture Search via Parameter Sharing". In: *arXiv:1802.03268v2 [cs.LG]*.

References II

Real, Esteban et al. (2019). "Regularized Evolution for Image Classifier Architecture Search". In: *arXiv:1802.01548v7 [cs.NE]*.

Tokui, Seiya and Issei sato (2016). "Categorical Reparameterization with Gumbel-Softmax". In: *arXiv:1611.01239v1 [stat.ML]*.

Xie, Sirui et al. (2019). "SNAS: Stochastic Neural Architecture Search". In: *arXiv:1812.09926v2 [cs.LG]*.

Zoph, Barret and Quoc V. Le (2016). "Neural Architecture Search with Reinforcement Learning". In: *arXiv:1611.01578v2 [cs.LG]*.

Thanks!