그룹단위 후보 연산 선별을 사용한 자동화된 최적 신경망 구조 탐색: 후보 연산의 gradient 를 기반으로

박성진, 송하윤
홍익대학교 컴퓨터공학과
demi-tasse@naver.com, hayoon@ hongik.ac.kr

DG-DARTS: Operation Dropping Grouped by Gradient Differentiable Neural Architecture Search

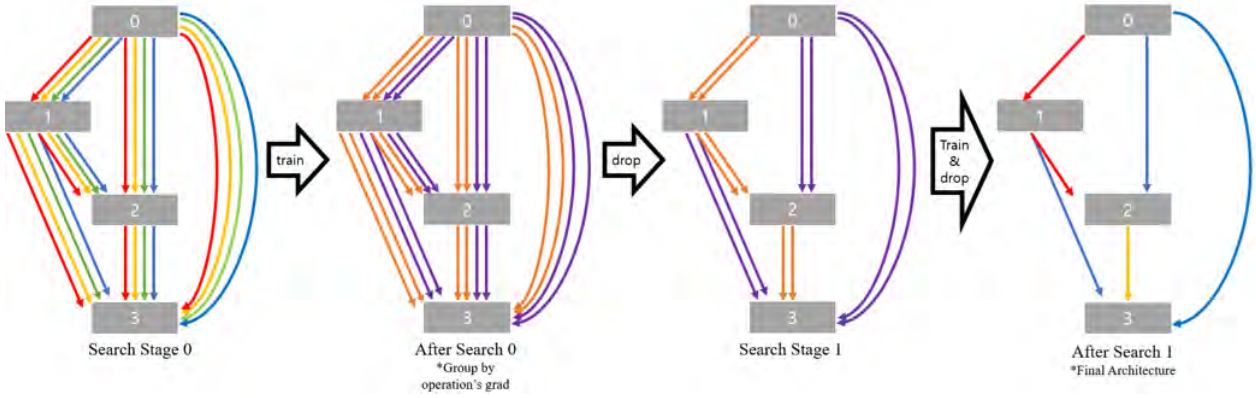
SeongJin Park, Ha Yoon Song
Department of Computer Engineering, Hongik University

요 약

gradient decent 를 기반으로 한 Differentiable architecture search(DARTS)는 한 번의 Architecture Search 로 모든 후보 연산 중 가장 가중치가 높은 연산 하나를 선택한다. 이 때 비슷한 종류의 연산이 가중치를 나누어 갖는 "표의 분산"이 나타나, 성능이 더 좋은 연산이 선택되지 못하는 상황이 발생한다. 본 연구에서는 이러한 상황을 막기위해 Architecture Parameter 가중치의 gradient 를 기반으로 연산들을 클러스터링 하여 그룹화 한다. 그 후 그룹별로 가중치를 합산하여 높은 가중치를 갖는 그룹만을 사용하여 한 번 더 Architecture Search 를 진행한다. 각각의 Architecture Search 는 DARTS 의 절반 epoch 만큼 이루어지며, 총 epoch 이 같으나 두번째의 Architecture Search 는 선별된 연산 그룹을 사용하므로 DARTS 에 비해 더 적은 Search Cost 가 요구된다. "표의 분산"문제를 해결하고, 2 번으로 나뉜 Architecture Search 에 따라 CIFAR 10 데이터 셋에 대해 2.46%의 에러와 0.16 GPU-days 의 탐색 시간을 얻을 수 있다.

1. 서론

Neural Architecture Search(NAS)는 Automatic Machine Learning(AutoML)의 한 분야로서 최근 주목을 받고 있는 분야이다. 다양한 후보 연산들을 조합하여 주어진 Task 를 해결하는 최적의 모델을 찾도록 하는 방법이 NAS 이다. NAS 분야의 기반을 잡은 NASnet[1]은 가능한 모델의 경우의 수가 약 10^{15} 개이다. 따라서 NAS 분야는 탐색 범위를 줄이고, 연산을 선택하는 기준을 바꾸는 등의 방법으로 Search Cost 를 줄이면서 성능을 상승시키는 쪽으로 발전되어 가고 있다. 초기 주류였던 강화학습[1], 유전알고리즘[2]을 사용한 NAS 알고리즘은 몇 천 몇 백 GPU-days 라는 막대한 Search Cost 가 요구되었으나 Gradient decent 를 기반으로 한 DARTS[3]는 불과 4 GPU-days 와 1 대의 GPU 만으로 기존 연구를 뛰어넘는 정확도를 보여주었다. DARTS 는 주어진 Task 를 해결하는 최적의 Network Architecture 를 찾기 위해 Cell 내부의 Node 를 잇는 연산을 후보 연산을 모두 사용한 Mixed Operation 으로 사용한다. 학습이 진행됨에 따라 연산들은 각각 Architecture Parameter 가중치가 변경되게 되며, 학습이 끝났을 때 Mixed Operation 에서 가장 큰 가중치를 갖는 후보 연산 하나만을 선택하여 최종 평가 Network 를 구성한다. DARTS[3]의 후보 연산들 중 비슷한 연산 결과를 유도하는 연산들(e.g., max pool, average pool)이 존재한다. 따라서 만약 특정 두 Node 를 잇는 연산이 pooling 연산이 되어야 가장 좋은 성능을 얻을 수 있다고 가정한다면, max pool 과 average pool 이 가중치를 나누어 갖기 때문에 제 3 의 연산(e.g., skip connection)이 선택되는 경우가 나타날 수 있다. 이러한 상황을 본 연구에서는 "표의 분산"문제라고 정의하며 이 문제를 해결하기 위해 Architecture Search 를 2 개의 Stage 로 나누어 진행한다. 첫 번째 Stage 의 학습 후 학습의 진행됨에 따른 가중치의 gradient 를 기준으로 연산들을 그룹 짓는다. 그 후 그룹별 가중치의 합을 계산하여 높은 가중치의 그룹을 사용하여 두 번째 Stage 를 진행한다. 이 과정을 통해 "표의 분산" 문제를 해결할 수 있으며, 두 번째 Stage 는 더 적은 후보 연산을 갖게 되므로 DARTS[3]와 동일 epoch 을 학습하더라도 더 짧은 시간(0.16 GPU-days)에 더 좋은 성능(2.46% test error, CIFAR10[4])를 얻을 수 있다.



(그림 3) 본 연구의 Architecture Search step 의 시각화.

2. 관련 연구

DARTS[3]의 단점을 해결하기 위한 다양한 논문들이 존재한다. P-DARTS[5]는 DARTS 가 8cell 의 proxy 모델을 통해 최종 Network 인 20cell 모델을 찾기 때문에 발생하는 Depth gap 문제를 해결하기 위해 3 회의 stage 를 거쳐 점진적으로 cell 의 개수를 늘려가며 Architecture Search 를 진행한다. PC-DARTS[6]는 DARTS 의 메모리 부담을 줄이기 위해 Network 의 Channel 을 부분적으로 연결하여 batch 크기를 4 배로 늘려 Search Cost 를 DARTS 의 1/4 로 줄일 수 있었다. STACNAS[7]는 DARTS 의 two-level optimization 로 인한 문제를 해결하기 위해 학습 전에 후보 연산의 Feature map 을 구하기 위한 모델을 만들고 이를 통해 상관 계수를 구해 연산을 그룹화 한다. 그 후 각 그룹의 대표 연산과 승리한 연산 그룹의 연산을 사용하여 stage0 과 stage1 을 진행한다.

본 연구는 "표의 분산"문제를 STACNAS 와 달리 학습 과정에서 해결함을 목표로 한다. 이를 통해 연산이 종류가 다른 경우도, 종류는 같으나 필터의 크기가 같은 경우에도 같은 그룹으로 분류될 수 있어 그룹의 기준 자유도가 높아지는 장점을 갖는다.

3. 이론

3.1 서론: DIFFERENTIABLE ARCHITECTURE SEARCH (DARTS)

본 연구는 DARTS[3]의 프레임워크를 기반으로 하고 있다. DARTS 의 목표는 network 를 구성하는 L 개의 cell 의 구조를 찾는 것이다. 각각의 cell 은 N 개의 node 를 가진 유향 비 순환 그래프(directed acyclic graph, 이하 DAG)로 표현되며 각각의 node 는 network layer 라고 정의된다. 사전에 정의된 연산의 집합(e.g., convolution, max pooling, *zero*)을 $\mathcal{O}$ 라 표기하며, operation 집합에서 선택된 후보군(e.g., convolution, *zero*) 함수는 $o(\cdot)$ 로 표기한다. DARTS 목표인 Cell 의 구조를 찾는 다는 의미는 각 node 의 쌍을 잇는 하나의 연산을 집합 $\mathcal{O}$에서 선택하는 것이다. 이 때의 node 쌍을 (i, j)라 표기하며 총 N 개의 노드가 존재하므로 i,j 의 범위는 $0 \le i < j \le N-1$이다.

$$f_{i,j}(x_i) = \sum_{o \in \mathcal{O}_{i,j}} \frac{\exp\left(\alpha_o^{(i,j)}\right)}{\sum_{o' \in \mathcal{O}} \exp\left(\alpha_{o'}^{(i,j)}\right)} \cdot o(x_i)$$

(수식 1) DARTS 의 main idea

x_i 는 i 번째 node 의 출력을, $\alpha^{i,j}$ 는 architecture parameter 의 연산 가중치를 의미한다. 각 node 의 출력은 모든 입력들의 수식 1 의 연산 후 합 ($x_j = \sum_{i<j} f_{i,j}(x_i)$) 이며, 또한 각 Cell 의 출력은 $concat(x_2, x_3, \cdots, x_{N-1})$로 구성된다.

3.2 변화량을 기준으로 한 연산 그룹 선별.

DARTS[3]는 8 개의 cell 로 구성된 proxy network 에서 도메인 데이터에 대한 최적의 cell 구조를 찾는다. 찾은 cell 에 대한 성능 평가는 이전단계에서 찾은 cell 을 20 개 적층 하여 network 를 구성하고 이를 통해 평가를 진행한다. Search step 에서 DART 는 proxy network 1 회 만들고, node(i,j)를 잇는 edge 는 집합 $\mathcal{O}$ 의 k 개의 원소 중 1 개의 원소를 선택한다. 이와 같은 방식은 비슷한 종류의 연산들을 search space 로 사용하는 경우 표의 분산 효과를 일으킬 가능성이 높다. 예를 들어, Node1 과 2 를 잇는 edge 가 convolution 연산이라면 loss function 이 최소가 되는 경우를 가정하자. 이 경우 학습이 진행됨에 따라 architecture parameter 는 convolution 연산의 가중치를 높이고, 나머지 연산의 가중치를 낮출 것이다. 만약 convolution 연산이 4 종류인 경우 convolution 연산이 가져야 하는 가중치를 4 개의 연산이 나누어 갖게 된다. 이를 쉽게 설명하기 위해 투표에 비유할 수 있다. 정당 A, B 의 후보들 중 1 명의 국회의원을 뽑는 상황을 가정한다.

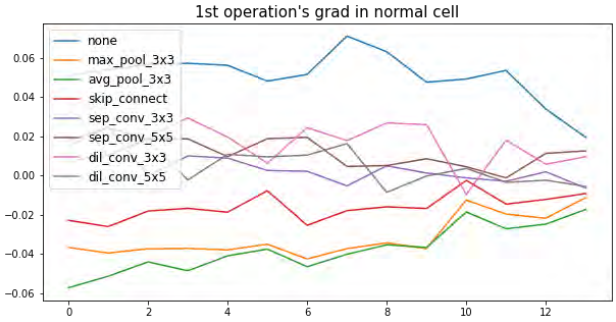
정당 A 를 지지하는 시민들은 정당 A 의 후보들 중 한 사람에게 투표를 하게 될 것이다. 후보 단일화가 진행된 B 정당을 지지하는 시민들의 경우 모두 한 사람에게 투표를 하게 된다. 따라서 정당 A 의 총 투표수가 정당 B 보다 높더라도, 최종적으로 표를 가장 많이 받은 정당 B 의 후보가 당선이 될 수 있다. 위와 같은 문제를 방지하기 위해 정책(gard)을 기준으로 후보들(operation)을 정당(clustered group)으로 분류하여 표(architecture parameter weights)를 취합하는 첫번째 투표(stage 0)을 진행하고, 당선된 정당($\mathcal{O}_{new}$)의 후보들 중 국회의원 (최종 operation)를 뽑는 최종 투표(stage 1)를 진행한다.

본 연구에서는 Search step 을 2 개의 stage 로 나누어 진행한다. Stage 0 에서는 DARTS 와 동일한 방법으로 학습을 진행한다. 학습이 끝난 후, architecture parameter 의 grad 를 기준으로 연산 'none'을 제외한 집합 $\mathcal{O}$ 의 연산들을 k-means 알고리즘을 사용하여 3 개의 클러스터로 분류한다. None 을 하나의 클러스터로 취급하여 총 4 개로 분류된 클러스터별로 architect parameter 의 weights 를 더하여 가장 높은 weights 를 갖는 클러스터의 연산들을 stage1 에서의 search space 인 집합 $\mathcal{O}_{new}$에 추가한다. ($\mathcal{O}_{new}$ 의 원소의 개수가 일정 값 미만이라면, 다음으로 weights 의 합이 높은 클러스터의 연산들 중 가중치가 큰 순서로 $\mathcal{O}_{new}$에 추가한다.) 이후 stage1 은 $\mathcal{O}_{new}$ 를 search space 로 하는 DARTS 와 동일하게 진행되며, stage0 보다 적은 수의 연산을 사용하므로 stage0 에 비해 더 빠르게 진행된다.

4. 실험 및 결과

4.1 데이터셋 및 실험 방법

Computer Vision 분야에서 가장 많이 사용되는 데이터 셋 중 하나인 CIFAR10[4]을 사용하여 실험이 진행되었다. CIFAR10 데이터셋은 10 개의 카테고리를 갖는 32 × 32 해상도의 이미지 6 만장으로 구성되어있다. 이 중 5 만장이 학습용, 1 만장이 test 용으로 분류 되어 있으며, 본 연구에서는 5 만장의 학습용 데이터를 두 그룹으로 나누어 network 의 학습을 위해 2 만 5 천장, architecture parameter 의 학습을 위해 2 만 5 천장을 사용하였다.



(그림 4) normal cell 의 1 번째 mixed operation 의 epoch(가로축)에 따른 gradient(세로축)의 변화량

Proxy network parameter 는 batch size 와 epoch, drop path prob 을 제외하고 모두 DARTS 와 동일한 값으로 진행되었으며 최초의 $\mathcal{O}$의 원소 역시 DARTS 와 동일한 8 개의 연산('none', 'max_pool_3x3', 'avg_pool_3x3', 'skip_connect', 'sep_conv_3x3', 'sep_conv_5x5', 'dil_conv_3x3', 'dil_conv_5x5')을 사용한다. 기존 DARTS 에서 1 회의 학습, 50epoch 으로 진행되었고, 본 연구는 총 2 회의 학습을 하게 되므로 기존의 절반인 25epoch 으로 한번의 학습을 진행한다. 이 때 warm start 를 위해 architecture parameter 를 고정하고 10epoch 을 학습 후 network, architecture parameter 를 둘 다 학습하는 15epoch 을 진행한다((10+15)×2 = 50). Batch size 는 96, drop path prob 은 0.3 설정하였다. 위의 설정으로 Neural Architecture Search 에 Tesla P100 을 사용하여 3.7 시간이 소요되었다.

Architecture	Test Err.(%)	Search Cost (GPU-days)	Search Method
DenseNet-BC [9]	3.46	-	수작업
NASNet-A [1]+ cutout	2.65	1800	RL
DARTS (1st order) [3] + cutout	3.00	1.5	Gradient-based
DARTS (2nd order) [3]+ cutout	2.76	4	Gradient-based
SNAS (moderate) [10]+ cutout	2.8	1.5	Gradient-based
P-DARTS [5]+ cutout	2.50	0.3	Gradient-based
PC-DARTS [6]+ cutout	2.57	0.1	Gradient-based
DG-DARTS(ours) + cutout + auto argument	2.46	0.16	Gradient-based

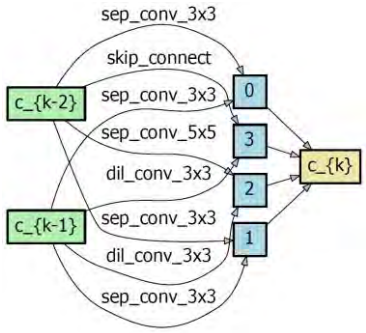
(표 1) CIFAR 10 데이터셋을 사용한 state-of art network architecture 들 과의 비교

연산의 그룹을 위하여 사용한 Elkan[8] k-means 알고리즘의 parameter 인 max iterator = 300, tolerance = 1e-4, number of cluster = 3 으로 진행되었다.

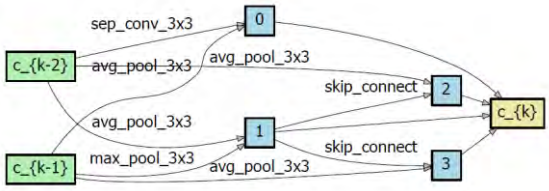
Operation 1 은 max_pool_3x3, avg_pool_3x3// skip_connect // sep_conv_3x3, sep_conv_5x5, dil_conv_3x3, dil_conv_5x5 으로 분류되었으며 이는 그림 4 를 통해 확인 할 수 있다. 이러한 클러스터링을 통해 단순 상위 n 개의 연산을 고를 때 다음 stage 로 가지 못한 연산이 다음 stage 로 가는 상황을 기대할 수 있게 된다.

4.2 Architecture 평가

20 개의 cell 과 36 개의 채널을 갖는 최종 평가 Network 는 Batch size 96 으로 총 600 epoch 을 학습한다. Cutout regularization[11] length = 16, auto argumentation[12]이 사용되었으며, Auxiliary towers[13] 의 가중치 = 0.4, drop-path-prob = 0.3 적용하였으며, SGD optimizer 의 wight decay = 0.003, momentum = 0.9, 초기 learning rate = 0.025 이며 cosine annealing 을 사용하였다. 평가의 결과는 표 1 에 요약되어 있다.



(그림 5. a) Normal Cell 의 구조



(그림 5. b) Reduce Cell 의 구조

(그림 5) DG-DARTS 를 통해 찾은 Cell 의 구조

5. 결론

본 연구에서는 DARTS[3]에서 생길 수 있는 "표의 분산"문제를 해결하기 위해 Architecture Search 를 2 개의 stage 로 나누어 진행하였다. Stage0 에서 Stage1 로 넘어갈 때, 후보 연산을 줄이기 위해서 그룹 단위로 연산을 선별하였다. 이 때 그룹의 기준을 epoch 에 따른 architecture parameter 의 gradient 로 하여 표를 나눠 갖는 연산의 가중치를 합산해주었다. 이러한 과정을 통해 기존의 DARTS 알고리즘에서 선택되지 못하고 버려졌던 연산을 사용할 수 있게 되었으며 이에 따라 Test Accuracy 가 기존 DARTS 의 97%에서 97.54%로 상승하였다. 또한 동일 epoch 를 학습하더라도 stage1 은 적은 수의 후보 연산을 사용하므로 DARTS 보다 적은 Search Cost 가 소비되며 이로 인해 기존 1.5GPU days 에서 0.16 gpu days 로 약 9 배의 Cost 감소를 얻을 수 있음을 본 연구에서 확인하였다.

사사

이 연구는 정부(교육과학기술부)의 재원으로 한국연구재단의 지원을 받아 수행됨(NRF-2019R1F1A1056123).

참고문헌

[1] Zoph, B., and Le, Q. V. Neural architecture search with reinforcement learning. arXiv preprint arXiv:1611.01578. 2016.

[2] Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. In AAAI, 2019.

[3] Liu, H. Simonyan, K. and Yang, Y. DARTS: Differentiable architecture search. In ICLR. 2019.

[4] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[5] Xin Chen, Lingxi Xie, JunWu, and Qi Tian. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In ICCV, 2019.

[6] Yuhui Xu et al. Pc-darts: Partial channel connections for memory-efficient architecture search, International Conference on Learning Representations, 2019.

[7] Li Guilin, Zhang Xing, Wang Zitong, Li Zhenguo, Zhang Tong, StacNAS: Towards Stable and Consistent Optimization for Differentiable Neural Architecture Search, In ICLR, 2020.

[8] Charles Elkan. Using the triangle inequality to accelerate k-means. In Tom Fawcett and Nina Mishra, editors, ICML, pp.147-153. AAAI Press, 2003.

[9] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In CVPR, 2017.

[10] Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. SNAS: Stochastic neural architecture search. In ICLR, 2019.

[11] Terrance DeVries and Graham W Taylor. Improved regularization of convolutional neural networks with cutout. arXiv preprint arXiv:1708.04552, 2017.

[12] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment:Learning augmentation policies from data. arXiv preprint arXiv:1805.09501, 2018.

[13] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In CVPR, 2015.