Energy-Aware Neural Architecture Optimization With Splitting Steepest Descent

Dilin Wang*[1], Lemeng Wu*[1], Meng Li[2], Vikas Chandra[2], Qiang Liu [1]
{dilin, lmwu, lqiang}@cs.utexas.edu, {meng.li, vchandra}@fb.com
[1] UT Austin
[2] Facebook Inc.

Abstract

Designing energy-efficient networks is of critical importance for improving the applicability of deep learning to mobile and edge settings. In this work, we introduce a *splitting steepest descent* algorithm for fast energy-aware neural architecture optimization. Our main idea is to frame the architecture optimization into a continuous splitting process, which gradually grows the network by splitting existing neurons into multiple off-springs. Empirical results show that our method can train highly accurate and energy-efficient networks on challenging benchmarks such as ImageNet.

1 Introduction

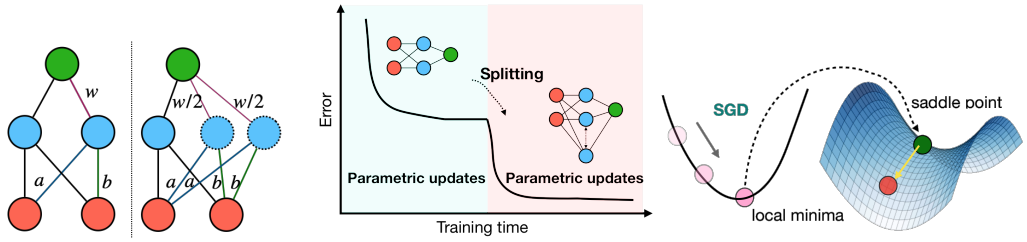
Deep neural networks (DNNs) has demonstrated remarkable performance in solving various challenge problems such as image classification, object detection and language understanding. Although large deep networks have good empirical and theoretical properties, they yield large and redundant networks that are slow and costly in the inference phase. This imposes a great challenge for the broader applicability of deep networks due to inference inefficiency, especially for mobile and edge settings where the computation and energy budgets are highly limited.

Neural architecture search (NAS) has been found a powerful tool for automating energy-efficient architecture design. Most existing NAS methods are based on black-box optimization techniques, including reinforcement learning, evolutionary algorithms. However, these methods are often extremely time consuming due to the enormous search space of possible architectures and the high cost for evaluating the performance of each candidate network. More recent approaches have made the search more efficient by using weight-sharing (e.g. Pham et al., 2018), which, however, suffers from the so-called multi-model forgetting problem (Benyahia et al., 2019) that causes training instability and performance degradation during search. Overall, designing the best architectures using NAS still requires a lot of expert knowledge and trial-and-errors.

In contrast, pruning-based methods construct smaller networks from a pretrained over-parameterized neural network by gradually removing the least important neurons. Various pruning strategies have been developed based on different heuristics (e.g., Li et al., 2016; Liu et al., 2017), including energy-aware pruning methods that use energy consumption related metrics to guide the pruning process (e.g., Gordon et al., 2018; He et al., 2018). However, a common issue of these methods is to alter the standard training objective with sparsity-induced regularization which necessities sensitive hyper-parameters tuning. Furthermore, the final performance is largely limited by the initial hand-crafted network, which may not be optimal in the first place.

In this work, we introduce a novel splitting steepest descent algorithm for neural architecture optimization. Instead of treating architecture search as a discrete optimization problem, our approach is to frame the architecture search problem into a novel continuous splitting process, which alternates

EMC^2: 5th Edition Co-located with NeurIPS'19



(a) (b) (c)

Figure 1: (a) Splitting one neuron into two off-springs. (b) Steepest descent on the overall architecture space consists of both standard gradient descent on the parameters (with fixed network structures), and updates of the network structures (via splitting). (c) The local optima in the low dimensional space is turned into a saddle point in a higher dimensional of the augmented networks, and hence can be escaped by our splitting strategy, yielding monotonic decrease of the loss.

between a standard parametric gradient descent phase to reach a parametric local minima, and a splitting phase that grows the network structure by splitting a subset of existing neurons into several off-springs in an optimal way. Our preliminary results show encouraging results on various image classification benchmarks. We refer the reader to Liu et al. (2019) for more mathematical treatments.

2 Splitting Steepest Descent

Our key idea is to view neural architecture search as a continuous optimization in the configuration space of the neuron parameters, on which steepest descent can be introduced for efficient optimization. To fix the idea, assume the neural network structure is specified by a set of size parameters $\boldsymbol{m} = \{m_1, \ldots, m_K\}$, where each m_k could denote the number of neurons of in the k-th layer, or of the kth type. Denote by $\Theta_{\boldsymbol{m}}$ the parameter space of networks of size $\boldsymbol{m}$, then $\Theta_\infty = \cup_{\boldsymbol{m}\in\mathbb{N}^K} \Theta_{\boldsymbol{m}}$, called the configuration space, is the space of all possible neural architectures and parameters.

In this view, learning parameters for a fixed network is minimizing the loss inside an individual sub-region $\Theta_{\boldsymbol{m}}$, while optimizing in the overall configuration space Θ_∞ admits the co-optimization of both the architecture and parameters. The key observation is that by considering optimization on Θ_∞ (instead, e.g., on the discrete size parameters $\boldsymbol{m}$), we obtain an (infinite dimensional) continuous optimization on Θ_∞. With a proper definition of distance on Θ_∞, we can then derive generalized steepest descent on Θ_∞ that leverages valuable gradient information to optimize the architectures more efficiently.

Intuitively, the steepest descent on Θ_∞ should consist of two phases: 1) the steepest descent inside each $\Theta_{\boldsymbol{m}}$ with a fixed $\boldsymbol{m}$, which reduces to the standard gradient descent on parameters, and 2) the descent across the boundary of different sub-regions $\Theta_{\boldsymbol{m}}$, which corresponds to optimal updates on the network structures. The structural descent across boundaries of $\Theta_{\boldsymbol{m}}$ can be viewed as escaping saddle points in the configuration space. As shown in Figure 1(b), when the parametric training inside a fixed $\Theta_{\boldsymbol{m}}$ gets saturated, structural descent allows us to escape local optima by jumping into a higher dimensional sub-region. The idea is that the local optima inside $\Theta_{\boldsymbol{m}}$ can be turned into a saddle point in the higher dimensional configuration space $\Theta_{\boldsymbol{m}}$ (Figure 1(c)), which can be escaped easily using stochastic gradient descent on the higher dimensional space.

Escaping local minima via splitting Our view motivates us to derive a steepest descent algorithm on Θ_∞, with a proper notion of distances. In Liu et al. (2019), steepest descent with ∞-Wasserstein distance was considered, which is shown to naturally correspond to practical procedures of splitting neurons into multiple off-springs (Figure 1(a)) to enlarge the capacity of the neural network. This yields a new style of highly efficient "growing-based" approaches with superb performance on challenging tasks in energy efficient NAS.

Specifically, consider a network with one neuron θ, assume the loss has a form,

$$\mathcal{L}(\theta) := \mathbb{E}_{x\sim\mathcal{D}}[\Phi(\sigma(\theta, x))],$$

where $\mathcal{D}$ is a collection of training data, $\sigma(\theta, x)$ represents the activation function and Φ defines the training loss given input x. We split θ into m off-springs $\{\theta_i\}_{i=1}^m$, and replace the neuron $\sigma(\theta, x)$

Algorithm 1 Splitting Steepest Descent for Neural Architecture Optimization

Starting from a small base network (viewed as the "seed" or template), we gradually grow the neural network by alternating between the following two phases until convergence:

1. Parametric Updates: Optimize neuron weights using standard methods (e.g., SGD) until no further improvement can be made by only updating parameters.

2. Splitting Neurons: Sort the neurons by their splitting indexes (see Equation equation 1), and split the top ranked neurons with negative splitting indexes into two off-springs along their splitting gradients.

with a weighted sum of the off-spring neurons $\sum_{i=1}^{m} w_i \sigma(\theta_i, x)$, where $\{w_i\}_{i=1}^m$ is a set of positive weights assigned on the off-springs, and satisfies $\sum_{i=1}^{m} w_i = 1$, $w_i > 0$. See Figure 1 (a) for an illustration. This yields an augmented loss function

$$\mathcal{L}(\{\theta_i, w_i\}) = \mathbb{E}_{x\sim\mathcal{D}}\left[\Phi\left(\sum_{i=1}^{m} w_i \sigma(\theta_i, x)\right)\right].$$

It is easy to see that if we set $\theta_i = \theta$ for all the off-springs, the network remains unchanged. Therefore, as we change θ_i in a small neighborhood of θ, it introduces a smooth change on the loss function without changing the functionality of the network significantly. This allows us to achieve monotone improvements of the loss through architecture updates as shown in Figure 1(c).

Deriving optimal splitting solutions It remains to derive the optimal splitting strategy, including deciding the number of off-springs m, the values of the weights $\{w_i\}$ and the parameters for the off-springs $\{\theta_i\}$. This is formulated into the following optimization problem:

$$\min_{m,\{\theta_i,w_i\}_{i=1}^m} \left\{\mathcal{L}(\{\theta_i, w_i\}) - \mathcal{L}(\theta) \quad \text{s.t.} \;\; ||\theta_i - \theta|| \le \epsilon, \; \sum_{i=1}^{m} w_i = 1, \;\; w_i > 0, \quad \forall\, i\right\}.$$

where we restrict the change of parameters within an infinitesimal ϵ-ball of the parameter of the original neuron, that is, $||\theta_i - \theta|| \le \epsilon$, with ϵ a step size parameter. When ϵ is very small, the optimum of this problem is determined (asymptotically) by a *splitting matrix* $S(\theta)$ defined as follows,

$$\min\left\{\mathcal{L}(\{\theta_i, w_i\}) - \mathcal{L}(\theta)\right\} \approx \frac{\epsilon^2}{2}\lambda_{\min}(S(\theta)), \quad \text{with} \;\; S(\theta) = \mathbb{E}_{x\sim\mathcal{D}}[\Phi'(\sigma(\theta, x))\nabla^2_{\theta\theta}\sigma(\theta, x)],$$

where the splitting matrix $S(\theta)$ is a $d \times d$ matrix (d is the dimension of θ); its eigenvalue $\lambda_{\min}(S(\theta))$ (denoted as *splitting index*) determines the optimal gain that can be achieved by splitting. In addition, this optimum can be achieved by a simple strategy of splitting the neuron into two copies with equal weights, whose parameters are updated along the minimum eigen-vectors $v_{\min}(S(\theta))$ of $S(\theta)$, that is,

$$m = 2, \qquad \theta_1 = \theta + \epsilon v_{\min}(S(\theta)), \qquad \theta_2 = \theta - \epsilon v_{\min}(S(\theta)), \qquad w_1 = w_2 = 1/2.$$

Since the decrease of the loss is proportional to $\lambda_{\min}(S(\theta))$, the splitting can achieve improvement only when $\lambda_{\min}(S(\theta)) < 0$, in which case the neuron is called "splittable". This suggests that we should apply the splitting strategy above only on splittable neurons.

Splitting deep neural networks The result above can be naturally extended to more general cases when we need to split multiple neurons in deep neural networks. Consider a neural network with n neurons $\theta^{[1:n]} = \{\theta^1, \cdots, \theta^n\}$. Assume we split a subset A of neurons with the optimal strategies above following their own splitting matrices, the improvement of the overall loss equals the sum of individual gains:

$$G(A) = \sum_{\ell\in A} \lambda_{\min}(\ell),$$

where $\lambda_{\min}(\ell)$ denotes the minimum eigenvalue of the splitting matrix $S_\ell(\theta^\ell)$ associated with neuron ℓ. Therefore, given a budget of splitting at most a given number of neurons, the optimal subset of neurons to split are the top ranked neurons with the most negative eigenvalues.

3 Neural architecture optimization via splitting

The method above allows us to select the best subset of neurons to split to yield the steepest descent on the loss function. In practice, however, splitting different neurons incurs a different amount of increase on the model size, computational cost, and physical energy consumption. For example, splitting a neuron connecting to a large number of inputs and outputs increases the size and computational cost of the network much more significantly than splitting the neurons with less inputs and outputs. Intuitively, one may want to split fewer neurons in layers close to inputs, which typically have larger input resolutions and a high energy cost, while split more neurons in deep layers which are responsible for the final classification and have lower computational cost. A better splitting strategy should consider taking the different cost of splitting different neurons into account.

To address this problem, we propose to explicitly incorporate the splitting cost different neurons to better guide the splitting process. Specifically, we propose to decide the optimal splitting set by solving the following constrained optimization:

$$\min_{\boldsymbol{\beta}} \sum_{\ell=1}^{n} \beta_\ell \lambda_{\min}(\ell), \quad \sum_{\ell=1}^{n} e_\ell \beta_\ell \leq e_{budget}, \quad \beta_\ell \in \{0,1\}, \ \ \forall \ell, \tag{1}$$

where β_ℓ denotes if enuron i should be split, and e_ℓ represents the cost of splitting neuron i at the current iteration of splitting steepest descent. In this work, we take e_ℓ to be energy cost, measured by flops. Note that the cost of splitting the same neuron changes across iterations as the network model size changes. Therefore, we re-evaluate the cost of every neuron at each network growing stage, based on the topology of the current network. See Algorithm 1 for an overview of our method.

4 Experiments

Results on CIFAR100 We first test on the CIFAR100 dataset using MobileNetV1 as backbone. We keep the network topology of our architectures as the same as the original MobileNetV1 while searching the best number of channels for each layer via splitting.

We compare with the popular *width multiplier* baseline (Howard et al., 2017) and several strong pruning methods: *Pruning (Bn)* (Liu et al., 2017), *Pruning (L1)* (Li et al., 2016) and Morphnet (Gordon et al., 2018). Figure 2 (a) shows the results on the CIFAR100 dataset, in which our method performs the best when targeting similar flops.

To further demonstrate the efficacy of our method in discovering better energy-efficient neural networks, we prune the final network learned by our splitting algorithm to attain a sequence of smaller models while maintaining similar flops as our splitting checkpoints using *Pruning (Bn)* (Liu et al., 2017). As we can see from Figure 2 (b), it is clear that our splitting checkpoints form a better flops-accuracy trade-off curve than pruned models.

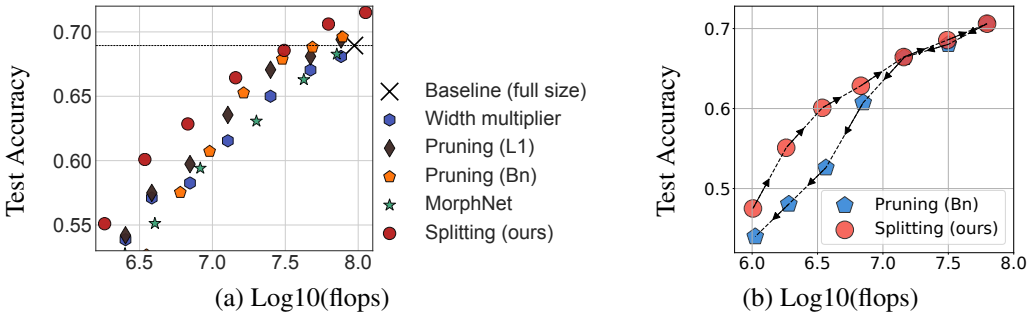


Figure 2: (a) Results on CIFAR100 using MobileNet(Howard et al., 2017) as backbone; (b) our preliminary results show that growing networks using our splitting steepest descent can learn more accurate and energy-efficient (with small flops) than pruning methods.

Results on ImageNet We also conduct experiments on large-scale ImageNet dataset. We choose both MobilenetV1 (Howard et al., 2017) and MobileNetV2 (Sandler et al., 2018) as our backbone

architectures, which were strong baselines and specifically hand-designed and heavily tuned to optimize accuracy under a flops-constrain on the ImageNet dataset. For our method, we start with relative small models (denoted by *Splitting-0 (seed)*) by shrinking the network uniformly with width-multiplier 0.3, and gradually grow the network via splitting. As we see from table 1 and 2, our splitting models yield better performance compared with prior art expert-designed architectures and AMC (He et al., 2018) on all flops-regimes.

Model	MACs (G)	Top-1 Accuracy	Top-5 Accuracy
MobileNet (1.0x) (He et al., 2019)	0.569	72.93	91.14
Splitting-4	0.561	**73.98**	**91.51**
MobileNet (0.75x)	0.325	70.25	89.49
AMC (He et al., 2018)	0.301	70.50	89.30
Splitting-3	**0.292**	**71.46**	**89.68**
MobileNet (0.5x)	0.150	65.20	86.34
Splitting-2	**0.140**	**68.25**	**87.94**
Splitting-1	0.082	64.07	85.30
Splitting-0 (seed)	0.059	59.20	81.82

Table 1: Results on ImageNet using MobileNetV1.

Model	MACs (G)	Top-1 Accuracy	Top-5 Accuracy
MobileNetV2 (1.0x) (He et al., 2019)	0.300	72.04	90.57
Splitting-3	**0.298**	**72.84**	**90.83**
MobileNetV2 (0.75x)	0.209	69.80	89.60
AMC (He et al., 2018)	0.210	70.85	89.91
Splitting-2	**0.208**	**71.76**	**90.07**
MobileNetV2 (0.5x)	0.097	65.40	86.40
Splitting-1	**0.095**	**66.60**	**87.06**
Splitting-0 (seed)	0.039	55.61	79.55

Table 2: Results on ImageNet using MobileNetV2.

5 Conclusions

In this paper, we introduce a splitting stepest descent idea for neural architecture optimization. Results on large-scale ImageNet benchmark using MobileNetV1 and MoibileNetV2 demonstrate the effectiveness of our method.

References

Yassine Benyahia, Kaicheng Yu, Kamil Bennani-Smires, Martin Jaggi, Anthony Davison, Mathieu Salzmann, and Claudiu Musat. Overcoming multi-model forgetting. *ICML*, 2019.

Ariel Gordon, Elad Eban, Ofir Nachum, Bo Chen, Hao Wu, Tien-Ju Yang, and Edward Choi. Morphnet: Fast & simple resource-constrained structure learning of deep networks. In *CVPR*, 2018.

Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of tricks for image classification with convolutional neural networks. In *CVPR*, 2019.

Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *ECCV*, 2018.

Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

Qiang Liu, Lemeng Wu, and Dilin Wang. Splitting steepest descent for growing neural architectures. *NeurIPS*, 2019.

Zhuang Liu, Jianguo Li, Zhiqiang Shen, Gao Huang, Shoumeng Yan, and Changshui Zhang. Learning efficient convolutional networks through network slimming. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2736–2744, 2017.

Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018.