Ineffectiveness of Gradient-based Neural Architecture Search

Garrett Bingham
Yale University
New Haven, CT
garrett.bingham@yale.edu

Abstract

Neural architecture search aims to automatically discover an effective neural network architecture for a given task. While most approaches to neural architecture search utilize reinforcement learning or neuro-evolutionary methods, DARTS (Liu et al., 2018) uses a continuous relaxation of the architecture representation to search for architectures using gradient descent. By training language models on Penn treebank, we demonstrate that gradient descent explores the search space ineffectively, and find that randomly initialized architectures are often able to outperform those discovered after extensive searching. We argue that gradient descent simply serves as a proxy for arbitrarily modifying the architecture, and show that gradient descent does not discover more capable architectures with each iteration of architecture search. These findings underscore the need for future neural architecture search approaches to be evaluated against more concrete baselines.

1 Introduction

The success of a neural network frequently depends on its architecture. Machine learning experts will often undergo significant trial and error before finally designing an effective neural network. This has inspired recent interest in developing techniques to automatically design effective neural network architectures for a given task. For a recent survey of neural architecture search, see (Elsken et al., 2018b).

Reinforcement learning approaches treat the neural architecture as the agent's action, and an estimate of the architecture's future performance as the agent's reward (Baker et al., 2016; Zoph and Le, 2016; Zoph et al., 2017; Zhong et al., 2018).

Neuro-evolutionary methods, on the other hand, use evolutionary algorithms to optimize the neural architecture, and utilize gradient-based methods to optimize an architecture's weights (Liu et al., 2017; Suganuma et al., 2017; Xie and Yuille, 2017; Real et al., 2017, 2018; Elsken et al., 2018a; Miikkulainen et al., 2019). Evolutionary algorithms evaluate a population of neural network architectures. At each evolution step, architectures are sampled to serve as parents. Mutations are applied to the parent architectures to obtain offspring which are trained and evaluated before being added to the population.

DARTS differs from the majority of neural architecture search approaches by using a continuous relaxation of the architecture representation to search for architectures using gradient descent. Reinforcement learning and neuro-evolutionary approaches are computationally expensive, and gradient descent offers a cheaper alternative. However, we show that gradient descent explores the search space ineffectively, and find that randomly initialized architectures are often able to outperform those discovered after extensive searching.

2 Overview of DARTS

A DARTS recurrent cell is a directed acyclic graph of N ordered nodes, where each node $x^{(i)}$ is a latent representation and each directed edge (i, j) corresponds to an operation $o^{(i,j)}$ that transforms $x^{(i)}$. A cell has two inputs: the input at the current step and the hidden state from the previous step. The output of the cell is obtained by concatenating all intermediate nodes, where each intermediate node is computed based on the previous nodes.

$$x^{(i)} = \sum_{j<i} o^{(i,j)}(x^{(j)}) \tag{1}$$

Let $\mathcal{O}$ be the set of all operations. For DARTS recurrent cells, $\mathcal{O} = \{\text{zero, tanh, relu, sigmoid, identity}\}$, where the *zero* operation indicates the

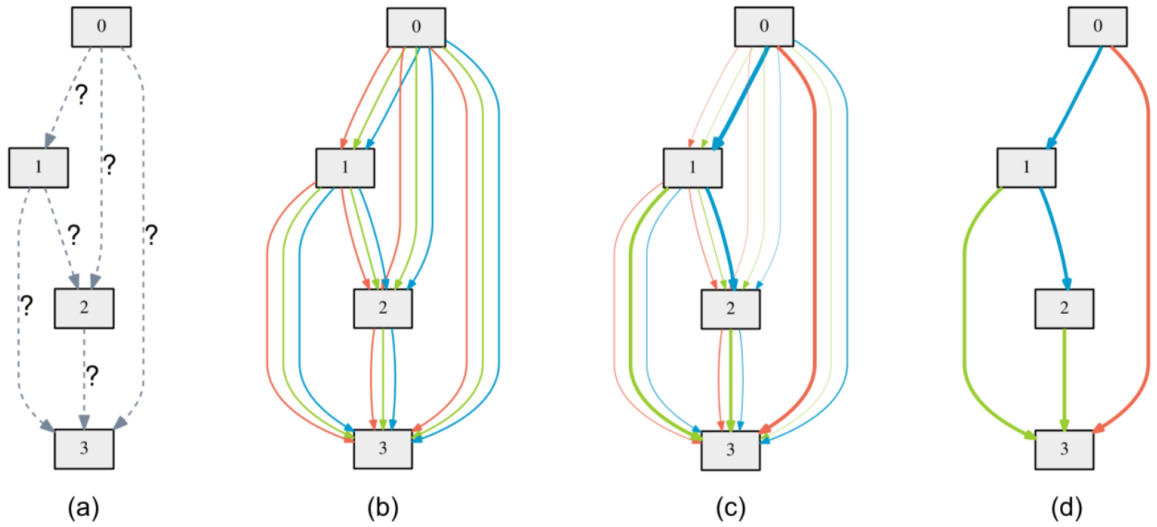

Figure 1: An overview of DARTS: (a) Operations on the edges are initially unknown. (b) Continuous relaxation of the search space by placing a mixture of candidate operations on each edge. (c) Joint optimization of the mixing probabilities and the network weights by solving a bilevel optimization problem. (d) Inducing the final architecture from the learned mixing probabilities. (Liu et al., 2018)

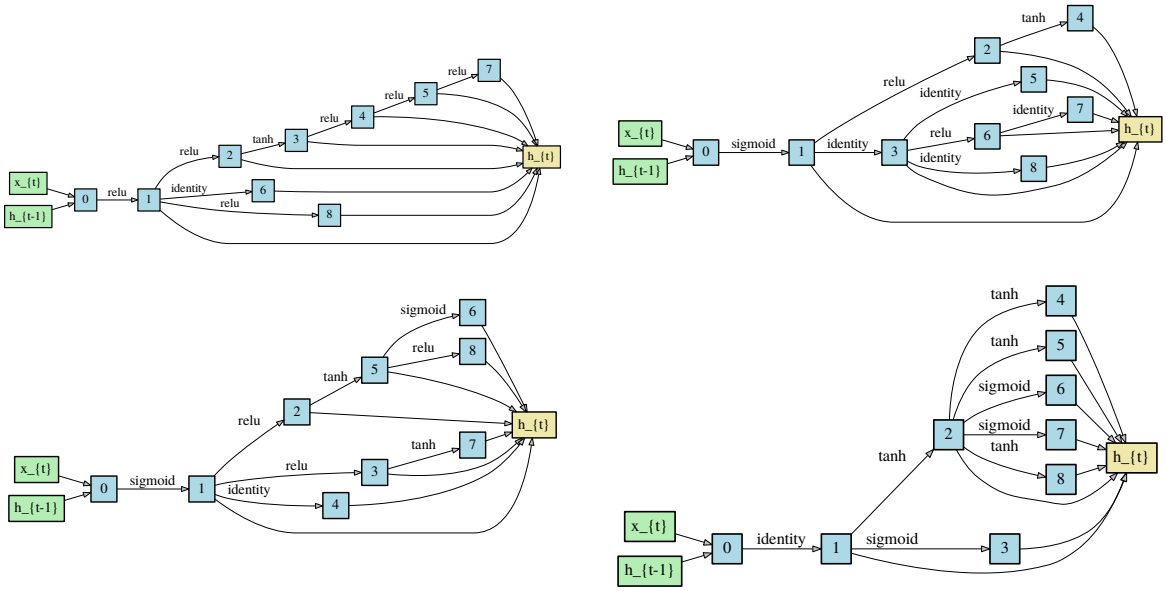

Figure 2: Example DARTS recurrent cells. The left two cells were included in the publicly available DARTS code, while the right two were randomly initialized.

absence of an edge between two nodes. To make the search space continuous, the choice of a particular operation is represented as a softmax over all possible operations.

$$\bar{o}^{(i,j)}(x) = \sum_{o \in \mathcal{O}} \frac{\exp\left(\alpha_o^{(i,j)}\right)}{\sum_{o' \in \mathcal{O}} \exp\left(\alpha_{o'}^{(i,j)}\right)} o(x) \quad (2)$$

The mixture of operations for an edge (i, j) is parameterized by the vector $\alpha^{(i,j)}$ of dimension $|\mathcal{O}|$. The task of architecture search becomes a task in learning the set of continuous variables $\alpha = \{\alpha^{(i,j)}\}$. Discrete architectures are obtained by replacing each mixed operation $\bar{o}^{(i,j)}$ with the most likely operation $o^{(i,j)} = \text{argmax}_{o \in \mathcal{O}}\, \alpha_o^{(i,j)}$.

Let $\mathcal{L}_{train}$ be the training loss and $\mathcal{L}_{val}$ be the validation loss. DARTS attempts to find the architecture α^* that minimizes the validation loss $\mathcal{L}_{val}(w^*, \alpha^*)$, where the weights w^* associated with the architecture minimize the training loss $w^* = \text{argmin}_w\, \mathcal{L}_{train}(w, \alpha^*)$. This corresponds to the bilevel optimization problem:

$$\min_{\alpha} \quad \mathcal{L}_{val}(w^*(\alpha), \alpha) \quad (3)$$

$$\text{s.t.} \quad w^*(\alpha) = \text{argmin}_w\, \mathcal{L}_{train}(w, \alpha) \quad (4)$$

It is expensive to solve the bilevel optimization problem exactly, so DARTS compromises by alternating gradient steps in the weights w and in the architecture representation α. The weights are optimized by descending in the direction $\nabla_w \mathcal{L}_{train}(w, \alpha)$, while the architecture is optimized by descending in the direction $\nabla_\alpha \mathcal{L}_{val}(w - \xi \nabla_w \mathcal{L}_{train}(w, \alpha), \alpha)$, where ξ is set equal to the learning rate for the weights optimizer. This algorithm does not necessarily converge, and as a result DARTS is sensitive to the initial random seed.

The process of searching for architectures is illustrated in Figure 1. After the architecture search phase is complete, the best performing architecture is retrained from scratch. While DARTS is capable of creating both convolutional and recurrent cells, we focus on recurrent cells in this paper. Example DARTS recurrent cells are shown in Figure 2. For more details, see the DARTS paper (Liu et al., 2018).

3 Experiments

3.1 Original DARTS Results

DARTS operates in two phases: the architecture search phase and the training phase. During the architecture search phase, DARTS alternates between taking gradient steps in the architecture representation α and gradient steps in the weights w of the architecture itself. After the architecture search phase is complete, the resulting architecture is trained completely from scratch in the training phase.

Liu et al. (2018) attempt to show that DARTS discovers increasingly capable architectures. They do so by fully training the architectures discovered after 0, 1, 2, 3, and 4 GPU hours of architecture search, and showing that the perplexity for language modeling on Penn treebank decreases over time (lower perplexity is better). Five data points, however, is unconvincing, and for some initial random seeds we found perplexity to *increase* with time. A more thorough experiment would demonstrate improvement by fully training the current architecture at every epoch of architecture search.

The DARTS authors also claim that architecture search by gradient descent outperforms a randomly initialized architecture. However, we argue that pitting the several dozen architectures considered during architecture search against a single randomly initialized architecture is not a fair comparison. A more appropriate experiment would consider one random architecture for each epoch of architecture search.

3.2 Our Experiment

To investigate whether gradient descent is capable of discovering increasingly better architectures, we run DARTS architecture search for 50 epochs. At each epoch of architecture search, we have a different architecture that DARTS predicts to be the most capable architecture discovered so far. This architecture represents the architecture we would use if the search process were stopped at that time. We train all 50 architectures from scratch for 100 epochs, and report test set perplexity for language modeling on Penn treebank. We also randomly initialize 50 architectures using random seeds 0-49. Each architecture is similarly trained for 100 epochs and compared against those discovered through architecture search. The results are summarized in Figure 3.

3.3 Discussion

We can draw a number of conclusions from the charts in Figure 3. First, it is evident that gradient descent is ineffective at discovering better DARTS architectures. The test perplexity of the

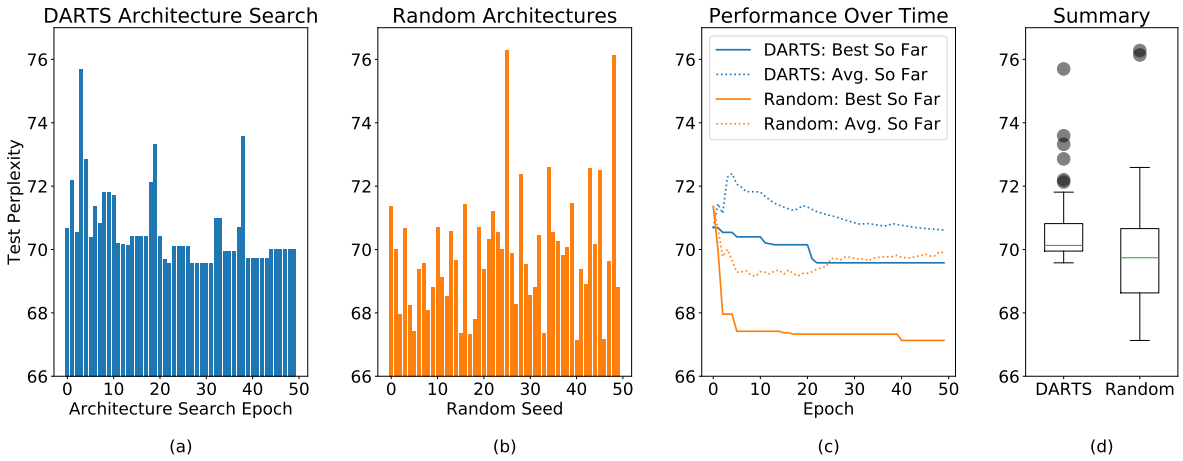

Figure 3: DARTS architecture search by gradient descent is compared with randomly initialized architectures. **(a)** DARTS architecture search is run for 50 epochs. At each epoch, the current architecture is trained from scratch on Penn treebank and test perplexity is reported (lower is better). **(b)** Fifty architectures are randomly initialized using random seeds 0-49. Each architecture is similarly trained on Penn treebank. **(c)** The performance of gradient-based architecture search against random architectures is compared. "Best So Far" represents the best performing architecture discovered up to that time, while "Avg. So Far" reports the average test perplexity among all architectures up to that time step. **(d)** The performances of all architectures discovered through architecture search and by random initialization are summarized.

architectures does not decrease over time. Even after 50 architecture search epochs (roughly 5.5 GPU hours), the resulting architecture is not significantly better than the initial architecture before searching began.

Second, we see many places in chart (a) of Figure 3 where the test perplexity does not change across several epochs. This is because gradient descent is stuck at a single architecture. This leads us to conclude that gradient descent, as currently implemented in DARTS, does not explore enough of the architecture space.

Third, random architectures are surprisingly effective. Of the 50 random architectures, 22 of them are better than the best architecture across all 50 epochs of gradient-based architecture search.

5.5 GPU hours of architecture search results in an architecture that performs reasonably well. On the other hand, architectures can be randomly initialized in seconds or less, and as we show here, chances are one of them will outperform the best architecture discovered via searching. We do not necessarily advocate for using random architectures in practice. Instead, we consider random architectures a reasonable baseline that any neural architecture search algorithm ought to outperform. Unfortunately, DARTS with gradient descent does not meet this criteria.

The benefit of gradient descent over reinforcement learning or neuro-evolution is that it is considerably less expensive. To improve DARTS, one might consider running multiple searches in parallel in order to explore more of the architecture space. Another strategy could involve triggering random restarts any time the search becomes stuck at a single architecture for multiple epochs.

4 Conclusion

Surprisingly, we find that gradient descent is easily outperformed by architecture search through random initialization. Our results imply that gradient descent does not explore enough of the architecture search space and often becomes stuck at a single architecture for many iterations. Furthermore, we do not find convincing evidence that gradient descent discovers more capable architectures with each iteration of architecture search. Rather, it appears that gradient descent simply serves as a proxy for arbitrarily modifying the architecture.

Our findings lead us to argue for stronger verification of future neural architecture search techniques. Any neural architecture search approach should be able to demonstrate that it progressively discovers more capable architectures throughout the architecture search process and that the architecture search strategy outperforms some meaningful baseline (such as randomly initialized architectures).

References

Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. 2016. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2018a. Efficient multi-objective neural architecture search via lamarckian evolution. *arXiv preprint arXiv:1804.09081*.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2018b. Neural architecture search: A survey. *arXiv preprint arXiv:1808.05377*.

Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. 2017. Hierarchical representations for efficient architecture search. *arXiv preprint arXiv:1711.00436*.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*.

Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, et al. 2019. Evolving deep neural networks. In *Artificial Intelligence in the Age of Neural Networks and Brain Computing*, pages 293–312. Elsevier.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. 2018. Regularized evolution for image classifier architecture search. *arXiv preprint arXiv:1802.01548*.

Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. 2017. Large-scale evolution of image classifiers. In *International Conference on Machine Learning*, pages 2902–2911.

Masanori Suganuma, Shinichi Shirakawa, and Tomoharu Nagao. 2017. A genetic programming approach to designing convolutional neural network architectures. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 497–504. ACM.

Lingxi Xie and Alan Yuille. 2017. Genetic cnn. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 1388–1397. IEEE.

Zhao Zhong, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu. 2018. Practical block-wise neural network architecture generation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2423–2432.

Barret Zoph and Quoc V Le. 2016. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. 2017. Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012*, 2(6).