Understanding Neural Architecture Search Techniques

George Adam
Department of Computer Science
University of Toronto
alex.adam@mail.utoronto.ca

Jonathan Lorraine
Department of Computer Science
University of Toronto
lorraine@cs.toronto.edu

Abstract

Automatic methods for generating state-of-the-art neural network architectures without human experts have generated significant attention recently. This is because of the potential to remove human experts from the design loop which can reduce costs and decrease time to model deployment. Neural architecture search (NAS) techniques have improved significantly in their computational efficiency since the original NAS was proposed. This reduction in computation is enabled via weight sharing such as in Efficient Neural Architecture Search (ENAS) Pham et al. [2018]. However, recently a body of work confirms our discovery that ENAS does not do significantly better than random search with weight sharing, contradicting the initial claims of the authors.

We provide an explanation for this phenomenon by investigating the interpretability of the ENAS controller's hidden state. We are interested in seeing if the controller embeddings are predictive of any properties of the final architecture - for example, graph properties like the number of connections, or validation performance. We find models sampled from identical controller hidden states have no correlation in various graph similarity metrics. This failure mode implies the RNN controller does not condition on past architecture choices. Importantly, we may need to condition on past choices if certain connection patterns prevent vanishing or exploding gradients.

Lastly, we propose a solution to this failure mode by forcing the controller's hidden state to encode pasts decisions by training it with a memory buffer of previously sampled architectures. Doing this improves hidden state interpretability by increasing the correlation controller hidden states and graph similarity metrics.

Keywords: RL, RNN, architecture search, neural architecture search, NAS

1 Introduction

1.1 Contributions

1. We demonstrate that the probability of the most likely action at any given time step is higher for a trained controller than a random controller. This suggests that a trained controller becomes more biased in its architecture sampling.
2. We show a validation performance distribution comparison between a trained and random controller immediately after the search phase, without final tuning. Although the performance for a trained controller seems significantly better, there is an explanation which does not imply the controller is actually sampling better architectures. Rather, the performance difference is an artifact of the weight-sharing scheme used.
3. We demonstrate that the controller embeddings learned by the ENAS controller are not encoding past actions, which is further evidence the controller is not doing anything meaningful. To rectify this, we provide a regularization technique for enforcing conditioning on past actions in the controller's hidden state space.
4. Lastly, we show an improved correlation between intuitive notions of architecture similarity and the distance between controller hidden states using this regularization technique.

2 Background & Related Work

We focus on the Efficient Neural Architecture Search (ENAS) algorithm presented by Pham et al. [2018]. There are various other competing NAS algorithms using one-shot architecture search (OSAS) [Bender et al., 2018], like DARTS [Liu et al., 2018]. These models have been applied to generating convolutional and recurrent architectures. Here, we analyze recurrent architectures which are evaluated on language modeling with Penn Treebank (PTB) [Marcus et al., 1993]. When generating recurrent models for language modeling tasks, the ENAS controller samples DAGs by making the following choices for each node in the DAG: (1) An activation function to use (2) A prior node to connect as input to the current node. An RNN is used for the controller architecture since it allows for arbitrarily many sequential sampling choices.

The induced search space is highly constrained, and different architectures do not vary significantly in performance since they all have high enough capacity to fit the PTB data set. What matters is the inductive bias of the DAGs and how easy they are to train. Concurrent work from Sciuto et al. [2019], Li and Talwalkar [2019] have found that the architectures found by state of the art OSAS models have similar performance to random search. We build on their work by confirming the performance of random baselines. Additionally, we propose potential causes and solutions for the lack of performance.

3 Experiments

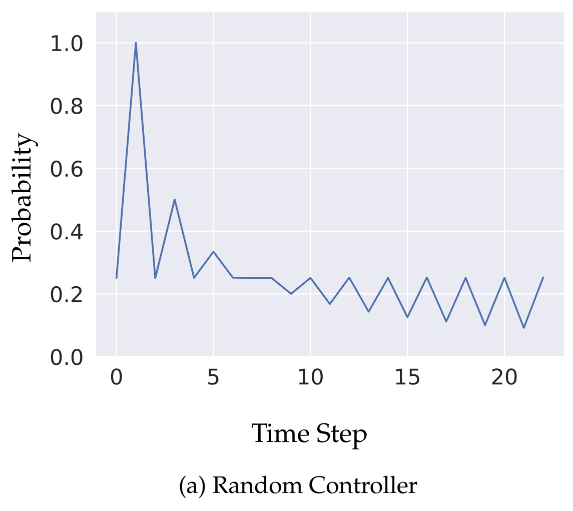
3.1 Biased Exploration

The most basic indication that the ENAS controller does something different and potentially better than random sampling is a significantly biased probability distribution at each time step. This indicates that it has decided to narrow the search space to a subset where certain connections/activations are more prominent. Such a phenomenon is seen in figure 1. We stress that this plot is not enough to conclude that the trained controller is sampling architectures that are in any way better than a random controller. The biased sampling could be due to a rich-get-richer phenomenon: because the ENAS training loop alternates between updating controller and shared parameters, if previously sampled architectures are sampled again, their probability of being sampled is increased simply because the shared parameters were improved.

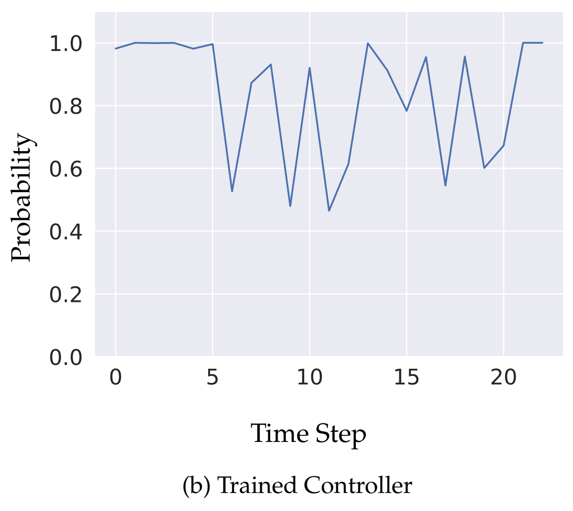
3.2 Quantifying Random Performance

Figure 2 suggests less variance in the performance of a trained vs. untrained ENAS controller at the end of the architecture search phase, without final tuning. The trained controller samples architectures whose performance is restricted to a narrower range that often performs better than the random controller. However, we can not conclude the trained controller sampled better architectures for the PTB language modeling task. A trained controller samples a less diverse architectures than a random controller. Thus, the shared parameters for a trained controller only have to be effective for a smaller set of architectures. Additionally, since it is unlikely that any of the architectures sampled during training for the random controller setting will be sampled at the end of training when the shared parameters have been tuned, it is not reasonable to expect those shared parameters to be effective for never before seen architectures.

This highlights a fundamental problem in OSAS NAS techniques. It is difficult to disentangle the factors behind validation set performance: architecture inductive bias and the OSAS shared weights for an architecture. Also, the entropy penalty in ENAS is critical in balancing exploration/exploitation. Increasing it results in more diverse architectures being



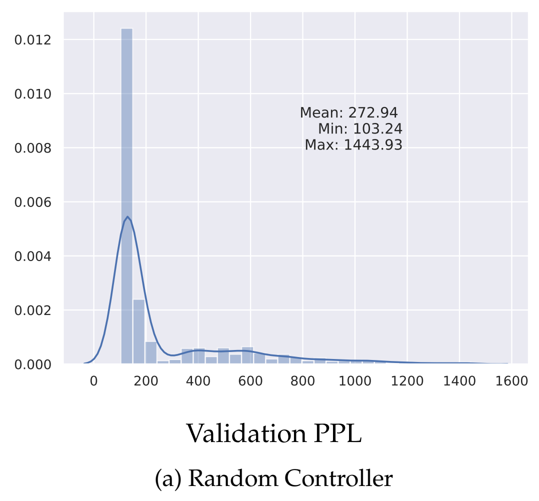
(a) Random Controller



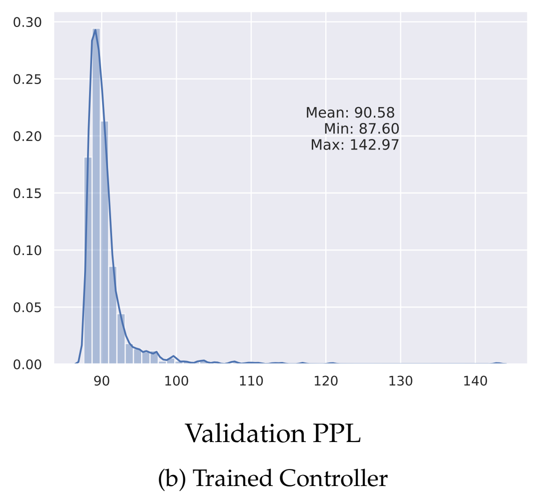
(b) Trained Controller

Figure 1: A trained controller is more certain about architecture choices than a random controller, indicating that architecture sampling occurs in a small subset of the entire architecture search space.

sampled during training, while weight sharing hinders the performance of any single model as the number of possible architectures increases. Thus, deciding which sampled architecture to use as the ”best” architecture depends on a highly sensitive entropy coefficient and random seed used.



(a) Random Controller



(b) Trained Controller

Figure 2: A trained controller seems to sample better architectures than a random controller. There are several explanations for this besides the inductive bias of the sampled architectures. For example, limitations of the OSAS weight sharing scheme.

3.3 Investigating ENAS Controller Embeddings

We inspect the hidden state of the ENAS controller for 100 different sampled architectures and find that it converges to the same vector, regardless of the choices that were made (figure 3a). This means the controller's hidden state cannot be used to differentiate between architectures, so it does not encode information regarding the structure of the DAG. The controller is unable to learn to effectively sample certain DAGs (ex. chains) without conditioning on past actions.

Specifically, the controller models the architecture choice c_t at step t as:

$$P(c_t|c_1, ..., c_{t-1}) \approx P(c_t) \tag{1}$$

We propose that this assumption is too simple and that the choices should be conditioned on the past:

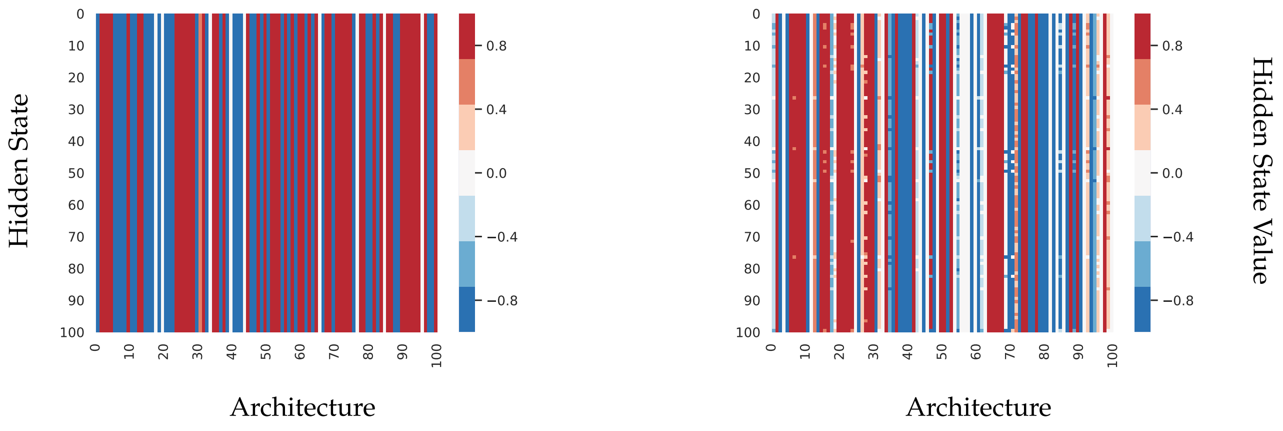
$$P(c_t|c_1, ..., c_{t-1}) = P(c_t|s_{t-1}) \tag{2}$$

where the state at a given time step s_t encodes (potentially) all choices made thus far. To encourage the encoding of past decisions, we applied the following regularization technique

- After 5 epochs of training, sample, and store 1000 architectures per epoch (up to limit of 10,000). Once the buffer is full, randomly replace 100 existing architectures every epoch
- At the 10^{th} epoch, add a supervised penalty for reconstructing a random sample of 100 architectures from the memory buffer of architectures. This loss is added to the policy gradient loss at each step of controller training: $\mathcal{L} = \mathcal{L}_{PG} + \mathcal{L}_{Sup}$

This regularization technique forces the controller embeddings to depend on previous choices. The way the memory buffer of architectures is constructed is important. If no buffer is used, and a new set of architectures is sampled at each step the controller can cheat by producing the same deterministic architecture, even when the entropy penalty is increased. Thus, there is a trade-off between constructing architectures that were sampled using old controller parameters, and architectures sampled using current controller parameters. This is similar to using experience replay when training GANs [Salimans et al., 2016].

Figure 3 shows the difference in the final hidden state for a supervised and an unsupervised controller. There is significant variability in the hidden state for the supervised controller - note that most columns are not solid lines, unlike the unsupervised controller. Furthermore, Figure 4 shows that the probability of the most likely action depends on previous decisions using our regularization, whereas it is independent of previous decisions without regularization.



(a) Unsupervised Controller (b) Supervised Controller

Figure 3: Final hidden state for 100 sampled architectures. Each row corresponds to the controller hidden state for a single architecture. Each column represents a single neuron in the controller's hidden state. Note how the supervised controller has significantly more variability per column, indicating it can distinguish between different architectures.

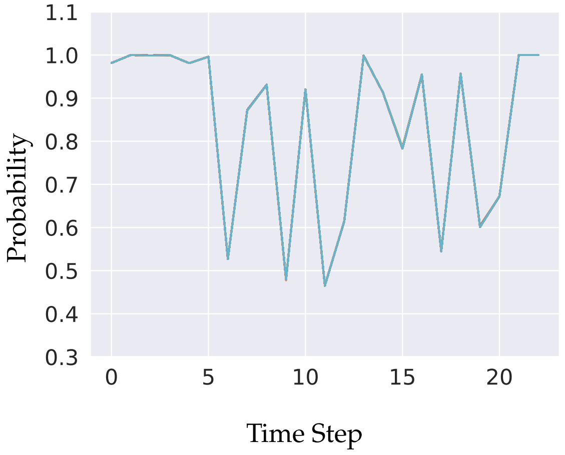
3.4 Architecture Similarities

Given that the regularized controller is able to discriminate between different architectures, we investigate if distances in this embedding space reflect architecture similarities. For two generated architectures $\mathcal{A}_1$ and $\mathcal{A}_2$ let $h(\mathcal{A}_1)$ and $h(\mathcal{A}_2)$ represent the hidden states of the controller at the final time-step of the architecture sampling process Does $||h(\mathcal{A}_1) - h(\mathcal{A}_2)||_2$ correlate with the number of common activation functions between the architectures, and the number of common connections between the architectures?

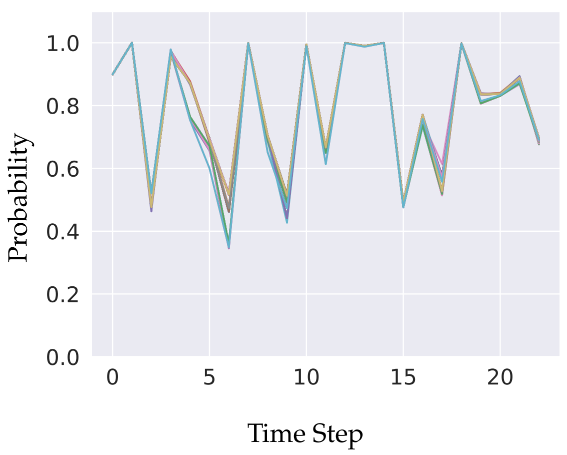
We compute the Spearman correlation between these measures of architecture similarity for a random, supervised, and an unsupervised controller in table 1. The regularization of the controller clearly helps improve the correlation between measured notions of similarity. Most impressively, there is now a significant correlation between controller embedding distance and absolute difference in model performance. This indicates that the controller's embedding space is arranged in a way that potentially captures validation set performance.

4 Conclusion

We showed that NAS algorithms have a common failure mode of not conditioning on past actions, which explains why they perform similarly to random search. Next, we demonstrated a way to regularize NAS to condition on past actions.



(a) Unsupervised Controller



(b) Supervised Controller

Figure 4: Argmax controller probability at each time-step for 100 different sampled architectures showing the advantage of our supervision method. The unsupervised controller collapses to a single line, indicating that probabilities do not change based on prior actions. The supervised controller has a visible separation between the lines, showing future actions are influenced by past actions.

Table 1: Correlation between L2 distance of architecture embeddings and four standard notions of similarity: number of common activation functions, number of common connections, graph edit distance, absolute validation set performance difference.

Train Type	# Common Activations	# Common Connections	GED	Abs. Performance Diff.
Random	-0.090	-0.130	-0.064	0.004
Supervised	**-0.404**	**-0.160**	**0.100**	**0.163**
Unsupervised	-0.315	-0.072	-0.028	-0.001

Finally, we provide experimental evidence that controllers under this regularization condition on past actions. We hope that this provides a step forward in allowing NAS to beat random baselines.

References

Gabriel Bender, Pieter-Jan Kindermans, Barret Zoph, Vijay Vasudevan, and Quoc Le. Understanding and simplifying one-shot architecture search. In *International Conference on Machine Learning*, pages 549–558, 2018.

Liam Li and Ameet Talwalkar. Random search and reproducibility for neural architecture search. *arXiv preprint arXiv:1902.07638*, 2019.

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

Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.

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.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in neural information processing systems*, pages 2234–2242, 2016.

Christian Sciuto, Kaicheng Yu, Martin Jaggi, Claudiu Musat, and Mathieu Salzmann. Evaluating the search phase of neural architecture search. *arXiv preprint arXiv:1902.08142*, 2019.