CS231N Project Proposal
Deep Neuroevolution with Shared Parameters: Ensembling a Better Solution

Michael Painter
Stanford University
mp703@stanford.edu

1. Problem Introduction

Although a lot of work has gone into designing neural network architectures, however, they are often still difficult to design, and often many architectures are tried and tested to select the best one. Automating this process leads to the notion of a neural architecture search [9, 21, 20]. However, naively implemented, this involves training hundreds of models from scratch, and typically uses hundreds of GPUs. This has been improved upon by using transfer learning (via function preserving network transforms, defined by Cai et al. [2]) and parameter sharing to reduce repeated work in training multiple models [1, 8].

However, many of these architecture searches tend to lead to state of the art performance in the tasks that they are implemented for. To understand why, we consider that rather optimizing a function $J : \Theta \to \mathbb{R}$ over parameters θ, we can instead optimize a function $J : \mathcal{H} \times \Theta \to \mathbb{R}$, and a naive architecture search is similar to a random search in this space.

2. Methods And Algorithms

As an objective function J cannot be differentiable with respect to the architecture space $\mathcal{H}$, we can instead attempt to use neuroevolution to optimize J with respect to $\mathcal{H}$. We will still use gradient based methods to optimize with respect to Θ.

Our main work will involve extending the work of Cai et al. [2], to define and implement function preserving transformations, that is, given $h_1, h_2 \in \mathcal{H}$, and parameters θ_1 find parameters θ_2 such that $h_1(\cdot;\theta_1) = h_2(\cdot;\theta_2)$. Specifically, we will use Inception networks [14, 15, 13] for our architecture space $\mathcal{H}$, and we will define *zero initializations* that allow us to initialize Inception modules, which when added into a (trained) network, don't change the output, given any input, shown in figure 1. We will then follow the work of Cai et al. [2], to check that our network transformations are valid, and can sufficiently learn.

Finally, we will implement our neuroevolution *meta-algorithm*, which will take the following form:

1. Initialize and train a network with small capacity, for N_1 steps. (The current population is of size 1).
2. While the population size is less than M_1, randomly pick a member of the population, and 'mutate' it, by performing a network widen or deepen operation on it.
3. Train each of the M networks, for N_2 steps.
4. Reduce the population to the $M_2 < M_1$ best performing networks.
5. Perform steps 2 to 4 for another $K - 1$ iterations, so that K iterations are performed in total.

As no existing weights are altered, we can use parameter sharing between the different networks of the population. This should lead to significant computational improvements (allowing a single GPU to be used), without hindering performance too much, as explained by Pham et al [8].

At the end, of training, we will incorporate multiple of the learned networks into an ensemble model.

3. Data

We will be use Imagenet [3] to evaluate the performance of the algorithms and architectures, which is openly available.

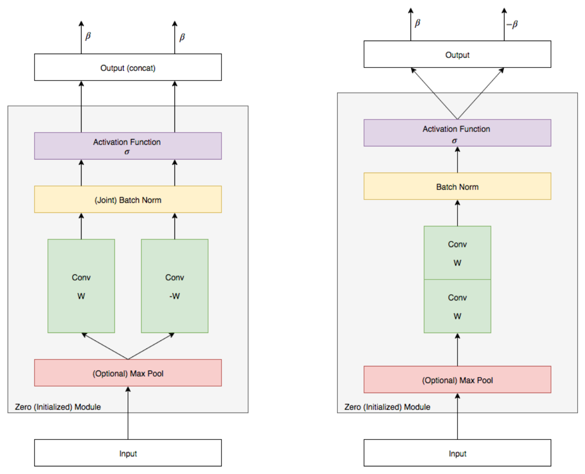

Figure 1. Left: Our initial concept for a zero initialized module. Adding two convolutional filters, one initialized with weights W and the other with weights $-W$. The split arrows represent duplication here. Right: Altered zero initialized module, so that it can be represented as a single filter, and also allows for non-symmetry in the activation function, weights β and $-\beta$ must be used on the output, to provide the zero output. The split arrows represent splitting into the two sets of filters here.

4. Evaluation

Our evaluation metric will be accuracy on the test set of Imagenet [3]. As we are looking to perform an architecture search efficiently, without requiring hundreds of GPUs, we also will analyze the computational efficiency. Specifically, we should compare the number of floating point operation (FLOPs) (rather than training time or epochs) with respect to training and test accuracies between models. In particular, for this comparison, we would look to compare training a single network, an ensemble of networks and our method.

5. Reading

We have already undertaken a reasonable amount of reading for this project, however, in total, we have identified the following topics as important to survey prior to implementation. Firstly, we draw a lot of material from architecture search [9, 21, 20], especially "efficient" architecture searches [1, 8], as well as the methods they are using to be efficient, such as transfer learning [2, 5]. As we will use (architectural) neuroevolution in our methods, it is important to survey older and more resent work in this area [4, 9, 12, 11, 16]. Finally, as we will evaluate using Image Classification, it is important to be up to date on architectures that achieve a good performance in this task [7, 13, 17], as well as the current state of the arts [6, 10, 18, 17, 19, 21].

References

[1] H. Cai, T. Chen, W. Zhang, Y. Yu, and J. Wang. Reinforcement learning for architecture search by network transformation. *arXiv preprint arXiv:1707.04873*, 2017.

[2] T. Chen, I. Goodfellow, and J. Shlens. Net2net: Accelerating learning via knowledge transfer. *arXiv preprint arXiv:1511.05641*, 2015.

[3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[4] S. E. Fahlman and C. Lebiere. The cascade-correlation learning architecture. In *Advances in neural information processing systems*, pages 524–532, 1990.

[5] S. Gutstein, O. Fuentes, and E. Freudenthal. Knowledge transfer in deep convolutional neural nets. *International Journal on Artificial Intelligence Tools*, 17(03):555–567, 2008.

[6] J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. *arXiv preprint arXiv:1709.01507*, 2017.

[7] G. Huang, Z. Liu, K. Q. Weinberger, and L. van der Maaten. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, volume 1, page 3, 2017.

[8] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*, 2018.

[9] E. Real, S. Moore, A. Selle, S. Saxena, Y. L. Suematsu, Q. Le, and A. Kurakin. Large-scale evolution of image classifiers. *arXiv preprint arXiv:1703.01041*, 2017.

[10] S. Sabour, N. Frosst, and G. E. Hinton. Dynamic routing between capsules. In *Advances in Neural Information Processing Systems*, pages 3859–3869, 2017.

[11] K. O. Stanley, D. B. D'Ambrosio, and J. Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15(2):185–212, 2009.

[12] K. O. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[13] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, volume 4, page 12, 2017.

[14] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, et al. Going deeper with convolutions. Cvpr, 2015.

[15] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.

[16] S. Whiteson. Reinforcement learning state of the art. *Adaptation, learning, and optimization*, 12:325–355, 2012.

[17] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 5987–5995. IEEE, 2017.

[18] Y. Yamada, M. Iwamura, and K. Kise. Shakedrop regularization. *arXiv preprint arXiv:1802.02375*, 2018.

[19] Z. Zhong, L. Zheng, G. Kang, S. Li, and Y. Yang. Random erasing data augmentation. *arXiv preprint arXiv:1708.04896*, 2017.

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

[21] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le. Learning transferable architectures for scalable image recognition. *arXiv preprint arXiv:1707.07012*, 2017.