CS231N Project Proposal (long version)
Deep Neuroevolution with Shared Parameters: Ensembling a Better Solution

Michael Painter
Stanford University
mp703@stanford.edu

Abstract

The 200-400 words submitted as a project proposal: todo

1. Introduction

Recently, there is a number of works that investigate performing an architecture search, to autonomize neural network architecture design. Naively, this involves training a neural network architectures from scratch and comparing their performance on the test set [20, 9]. However, this tends to be computationally very expensive, and is paralleled by a random search over weights, when optimizing a neural network. Clearly this is extremely inefficient, and it can and has been improved upon [1, 8]. In this project, we have identified that a number of methods used to perform (efficient) architecture searches could be combined, so we will investigate the effect this has, similar to how Rainbow [**?**] combined a number of Deep Reinforcment Learning methods to increase performance.

Specifically, we will implement an architecture search by neuroevolution. We will use a constrained space of architectures, that can be formed by "extending each other" using function preserving transformations as in [1], using function preserving transformations similar to net2net [2]. Moreover, the transformations will be implemented in such a way that requires no existing parameters to be altered, and we can therefore share parameters across models, as shown effective by ___ et al [8], allowing models to quickly be trained an evaluated without enormous computational cost.

For the purpose of this project, define our domain as $\mathcal{X}$, and output domain $\mathcal{Y}$. We define a model architecture space $\mathcal{H}$, where for each model $h \in \mathcal{H}$, we let Θ_h to be the set of possible weights (i.e. if h have k parameters, then $\Theta_h = \mathbb{R}^k$). That is, we have $h : \mathcal{X} \times \Theta_h \to \mathcal{Y}$.

Often in machine learning, we consider the optimization of an objective function $J : \Theta \to \mathbb{R}$, where Θ is the set of possible weights, given some fixed model. We can extend this idea, to instead optimize over the function $J : \mathcal{H} \times \Theta \to \mathbb{R}$, and we've set $\Theta = \bigcup_h \Theta_h$. We observe that J is now not differentiable, as $\mathcal{H}$ isn't a continuous domain, and therefore appeal to neuroevolutionary updates to optimize with respect to $\mathcal{H}$, as neuroevolutionary strategies can be applied to non-differentiable domains, and will use classical gradient based methods to optimize each $J(h, \cdot)$ with respect to Θ.

Finally, our work will involve the creation of a 'meta-algorithm', which will perform the neuroevolutionary architecture search. The algorithm is 'meta' as it we can use different forms of updates with it, for example, we could use it to train networks for classification (minimizing a cross-entropy loss for classification), or we could alternatively train a policy network for Reinforcement Learning tasks (using proximal policy optimization updates [**?**]).

2. Reading

I have already read a fair number of papers for this project, but here we reference all of the background reading that will be useful.

Firstly, we as we will be implementing some form of architecture search, I will read about some architecture searches, which allowed for huge computational costs [20, 9], and a number of 'efficient' architecture searches, run on less than 5 GPUs [1, 8].

As our architecture search is based on neuroevolution and function preserving transforms, it will be useful to read about some classic (topological) neuroevolution [16, 12, 11, 4], as well as work on transfer learning and function preserving transforms (which is a specific method of transfer learning) [5, 2].

If time, we will look at combining a number of models into an ensemble. Specifically, we will use a heirarchical ensemble, as described in [**?**]. For this to be effective, it will require coevolutionary strategies to be used, as described by Whiteson et al here [16].

As we will be evaluating the algorithms and architectures on image classification, we also need to survey the current

state of the art in image classification [10, 18, 6, 19, 17, 20], and the current state of the art in CNN architectures. Specifically, we will look at inception networks [14, 15, 13], residual networks [**?**, **?**, 17] and densenets [7].

3. Data

We will be use Imagenet [3] to evaluate the performance of the algorithms and architectures, which is openly available. Given enough time, we intend to submit an entry to OpenAI's retro contest [**?**], where we would train a policy network using PPO updates [**?**]. PPO was chosen because OpenAI claims that it is a good trade off between performance and ease of implementation. For prototyping, we will use a smaller image classification dataset, such as Tiny Imagenet, or CIFAR-10. So algorithm and architecture desicions may be made based on results with these datasets.

4. Methods and Algorithms

- TODO: write out the algorithm using algorithmic (do it in a MODULAR way, using *evolutionaryStrategy*. Then we can define *naiveEvolution : evolutionaryStrategy* function, and also *coevolutionaryStrategy : evolutionaryStrategy* function.
- It would be:
- Initialize population
- For some number of loops:
- Extend population, using some *mutationStrategy*
- Train, for some number of steps (either cycle through networks, or choose randomly)
- Select best networks from population, using some *evolutionaryStrategy*

To begin with, we will look at implementing function preserving transforms of neural networks. Because of the modular form of Inception networks [14, 15, 13], we will use this as a basis for our architecture space.

Net2net [2] defines operations to deepen and widen an inception network, however, we wish to not alter any weights that already exist in the network, and will have to define a new network widen transformation. To do so, we will derive an initialization that allows us to add Inception modules into the network, without altering it's overall function that the network currently represents. This amounts to being able to initialize a convolutional module such that it always has a contribution of 'zero' to the overall output, for every possible. I have already derived and tested that this is possible. Conceptually, this is similar to being able to write, for any f, g, β, that $f(x) = f(x) + \beta g(x) - \beta g(x)$, where f would be the current network, and g represents any additional module added. I will leave full details of this until the progress report, .

Next, we will run tests, similar to those run in net2net, to confirm that training a small network and then transforming it into to a larger network and continuing training will still hit the same performance as if we had have trained the larger model from scratch. Moreover, we will compare the training time of this two stage training procedure to the randomly initialized training, to confirm that the transfer learning is beneficial and doesn't take longer to train.

Finally, we will implement our neuroevolution meta-algorithm, which will take the following form: This will involve, first, training a 'small' network, for say N_1 steps. We will then generate a population of size M from this initial network, by randomly adding inception modules for 'widen' or 'deepen' transformations. The rest of training will then proceed as follows:

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. (Or, for $M_1 N_2$ steps, randomly pick a network and perform an update for it).
4. Reduce the population to the $M_2 < M_1$ best performing networks. (Naively, pick the M_2 with best performance, but hopefully we will explore more interesting ways to select networks).
5. Perform steps 2 to 4 for another $K - 1$ iterations, so that K iterations are performed in total.

We note, that because we will be defining the transformations such that no existing weights are altered, we can use parameter sharing between the different networks of the population. This should lead to significant computational improvements, without hindering performance too much, as explained by Pham et al [8].

Finally, we will attempt to take the M_3 best performing networks at the end of the training, and we will use them in a hierarchical ensemble to see if we can boost the performance. We expect that this by itself will not boost performance much, but, if we have time, we will try more interesting evolutionary strategies that encourage a more diverse population of networks.

Further ideas to try, given enough time:

- Try changing what "the best X performing network" means. Specifically, we wish to explore coevolutionary strategies

- We want to enter OpenAI's retro contest [**?**], which aims to be a new benchmark for 'meta-learning', or transfer learning algorithms. If enough time, we will use our 'meta-algorithm' with PPO [**?**], as OpenAI claims that it has good performance, and is easy to implement. Our main motivation is that this would provide further evaluation on if our neuroevolution 'meta-algorithm' is useful.

5. 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 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.

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

A. Layerwise Dropout Experiments

- Diagram
- Table of accuracy vs perfomance. (Experiment with 5 random masks for the dropout, and then combine them into an ensemble)
- A graph of performance vs dropout prob, of a one layer network on mnist. Many curves for different numbers of filters.

B. Zero Initialized Modules

- Gritty details on implementing the shared parameters, likely using masks
- Diagram of the module