Neural Network Autoencoders for Compressed Neuroevolution

Anonymous Author(s)
Affiliation
Address
email

1 Motivation

Neuroevolution is a gradient-free training method for deep reinforcement learning (RL) agents based on the principles of natural selection. Such et al. [2017] showed that RL agents trained via neuroevolution can avoid common shortcomings of gradient-based learning algorithms, such as falling into local traps preventing agents from finding better solutions. Subsequently, Evolutionary Reinforcement Learning (ERL) by Khadka and Tumer [2018] showed that neuroevolution methods can compliment gradient-based training methods to achieve even better performance. This idea was then expanded by Khadka et al. [2019] in CERL where a portfolio of agents drawing on gradient-free and gradient-based training methods achieve state-of-the-art performance on various RL benchmarks. The integration of gradient-based and gradient-free methods in frameworks such as CERL, however, still has not fully mitigated the primary weakness of neuroevolution techniques: low sample efficiency. Given that weakness, we propose a novel method to help mitigate the low sample efficiency of neuroevolution as a training method of RL agents, both in isolation as well as in a collaborative framework such as CERL.

2 Neural Network Autoencoders

A major reason why neuroevolution is very sample inefficient for training RL policy nets is due to the fact that the neural networks have a large number of weights and parameters. Moreover, the nature of the weight distribution is often sparse and contains many superfluous weights that overparametrize the problem, leading to extraneous computation during training. Hence, we aim to find a lower dimensional embedding of neural network weights by using autoencoders. As shown in Figure 1, the goal is to encode the network weights and thereafter perform the evolutionary operations in the latent subspace derived by the encoder. Since the latent subspace has fewer parameters, and would ideally be less sparse, there should be less wasted computation during neuroevolution. Autoencoders have been used in other areas of deep learning, particularly in computer vision, to generate latent codes of a variety of objects without making many assumptions about the nature the underlying distribution [Goodfellow et al., 2016].

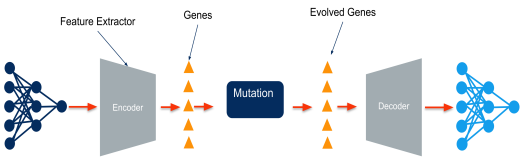


Figure 1: Mutation Performed on Latent Code

3 Current Work

The first challenge we face in order to test our hypothesis is to train an autoencoder and determine whether we can find a compressed representation for neural network weights. The training is

Submitted to 33rd Conference on Neural Information Processing Systems (NeurIPS 2019). Do not distribute.

based on the following loss function: $\mathcal{L} = \sqrt{\sum_{j=0}^{N}(y_D - y_I)^2}$, which is the L2 norm of the difference of the output distributions of the decoded network y_D and the input network y_I. This is a general purpose loss function that can be used to train neural network autoencoders regardless of the underlying task. Figure 2 shows a schematic of the actual training procedure for neural network autoencoders for deep RL policy networks. In order to make the problem more tractable, the training task was performed in a layer-by-layer fashion, which is preferred due to memory concerns of passing the entire collection of neural network parameters through an autoencoder structure. Furthermore, this process also makes the training process more modular, allowing one to find influential layers, as shown in the preliminary results for RL architectures in Figure 3. Figure 3 shows the results for training autoencoders on a distribution of networks that solved the Mujoco [Todorov et al., 2012] based Hopper-v2 environment. The input data for the autoencoders consisted of a collection of state-action pairs obtained by deployment previously trained Hopper networks in the environment, combined with a given percentage of random state-action pairs, which provides a more decorrelated set of inputs the autoencoder framework. The set of trained networks was split into training, validation, and test sets. In the testing phase, the networks produced by the decoder were deployed on the actual task and compared to the performance of the test set input networks on the same task. As seen in Figure 3, the training process allows the autoencoders to

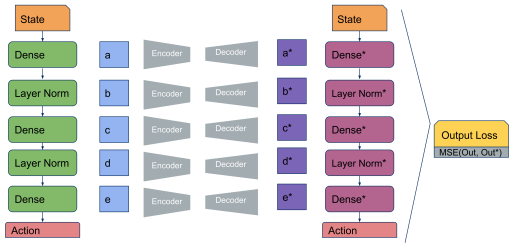
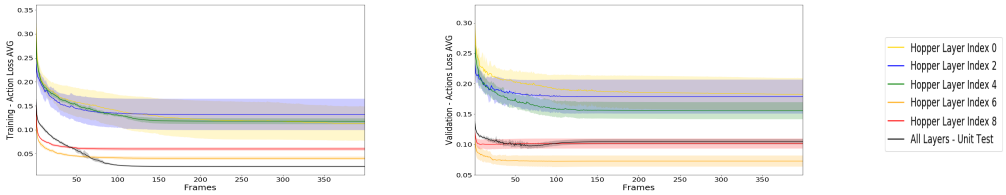


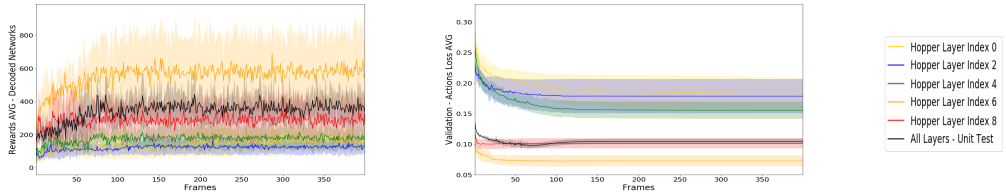
Figure 2: Layer-by-Layer Autoencoders for RL Policy Networks



(a) Training of Autoencoders on Performance Loss

(b) Validation of Autoencoders on Performance Loss

(c) Legend



(d) Testing - Decoded Networks Score in Hopper-v2 Environment

(e) Testing - Input Networks Score in Hopper-v2 Environment

(f) Legend

Figure 3: Training Neural Network Autoencoders on the Hopper-v2 Domain isolating different Layer Autoencoders - Layer Indexing starting from the layer closest to the input.

learn the parameter distribution of the training networks pretty well, while the validation and testing phase show a significantly higher variability in the results. The results in Figure 3 also show the different effects of autoencoding various layers in isolation on the overall performance. This suggests that rather than compressing all layers of a neural network, it is possible to find individual layers which provide better performance when compressed in isolation. Future work for this project will focus on finding ways to improve the testing performance of the autoencoders, and subsequently apply compressed evolution to train RL policy nets, in isolation as well as CERL-like collaborative frameworks, on the Mujoco benchmarks.

References

I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

S. Khadka and K. Tumer. Evolution-guided policy gradient in reinforcement learning. In *Advances in Neural Information Processing Systems*, pages 1196–1208, 2018.

S. Khadka, S. Majumdar, T. Nassar, Z. Dwiel, E. Tumer, S. Miret, Y. Liu, and K. Tumer. Collaborative evolutionary reinforcement learning. *arXiv preprint arXiv:1905.00976v2*, 2019.

F. P. Such, V. Madhavan, E. Conti, J. Lehman, K. O. Stanley, and J. Clune. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. 2017.

E. Todorov, T. Erez, and Y. Tassa. Mujoco: A physics engine for model-based control. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 5026–5033. IEEE, 2012.