AA222 Final Project Report: Application and Study of Evolution Strategies

Yutai Zhou
MIT Lincoln Laboratory
yutai@stanford.edu

Chuanqi Chen
KLA Corp.
cchuanqi@stanford

Abstract

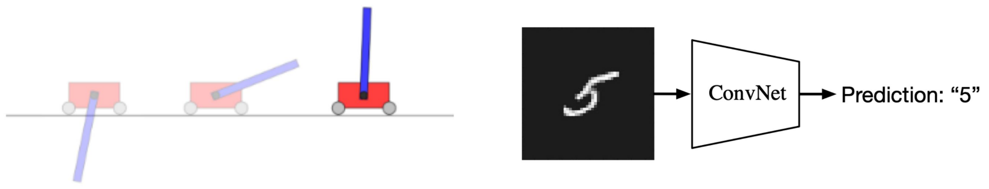
Inspired by evolutionary algorithms (EA) and gradient-free optimization of neural networks, we implemented two neuroevolution algorithms to solve tasks from computer vision and reinforcement learning: Cross Entropy Method (CEM) and Genetic Algorithm (GA). We experimented with multiple variants of both methods to solve the MNIST image recognition task and the Cartpole balancing task. The experiment configurations, chosen hyperparameters, and results, are all automatically organized and saved via wandb and Hydra, which dramatically improved our experiment iteration time. We find that although CEM somewhat outperforms GA on the Cartpole balancing task and was able to find a near-optimal behavior while GA could not, CEM's performance on the MNIST image recognition task lags far behind GA's performance. All experiments were done on a single NVIDIA GeForce RTX 2080 Ti GPU, using Jax's interface for easy hardware accelerator usage and EvoJax's interface for defining neuroeveolution algorithms.

1 Introduction

1.1 Motivation

From domains as diverse as computer vision, natural language processing, and reinforcement learning, gradient-based training of deep neural networks via back-propagation (BP) and stochastic gradient descent (SGD) has been the commonality in their massive successes stories. The seemingly magical combination of BP and SGD is not without flaws, however. Xie et al. (2020) theoretically and empirically revealed that SGD favors flat minima exponentially more than sharp minima. Beyond the engineering difficulty associated with large-scale gradient-based training, (e.g. gradient syncing across nodes and GPUs, gradient accumulation to overcome memory constraints), the computation of the gradient itself can be expensive. Despite advances in the software ecosystem of automatic differentiation, computing the gradients of deep neural networks with many parameters still remain a bottleneck in gradient-based training methods.

Figure 1: Cart-Pole Swing Up and MNIST Classification



Preprint. Under review.

On the other hand, despite the popularity of gradient-based methods, gradient-free optimization, particularly nature-inspired methods such as evolutionary algorithms (EA, Sloss and Gustafson (2019)), has remained an attractive option due to its lack of requirement for model differentiability as well as its simpler implementation challenges. Typically, gradient-free optimizers can be horizontally scaled almost linearly with the number of cores available. Morse and Stanley (2016) showed over several benchmarks that for neural networks with over 1,000 weights, a simple evolutionary algorithm rivals the performance of the then-state-of-the-art SGD variant RMSProp. Cui et al. (2018) proposed Evolutionary Stochastic Gradient Descent for Optimization of Deep Neural Networks.

1.2 Problem Definition

Although model differentiability is not a hard requirement for EA, we opt to use neural network models anyway. We apply and compare multiple neuroevolution algorithms on a task from the domain of control, particularly, in evolving policies that can reliably solve the task of Cartpole Swing Up (Manrique et al. (2020)) (refer to Figure 1) There is an unstable pendulum bolted to a cart, with its center of gravity away from pivot point. The pendulum can be controlled by horizontally moving the cart, causing the the pendulum to swing upwards. The goal is to keep the pole upright for as long as possible by controlling the cart's movement to swing the pole. The state space is a vector containing the position of cart, velocity of cart, angle of pole and rotation rate of pole; the action space is to move the cart in either horizontal directions. +1 reward is given for every time step taken, and an episode ends when the pole falls or when the cart veers off the screen.

We also applied EA to the domain of computer vision (e.g. MNIST classification, refer to Figure 1). The MNIST database of handwritten digits contains 60,000 training images and 10,000 testing images of handwritten digits from 0 to 9. In this task we train and evaluate these models' performance in terms of prediction accuracy of recognizing digits correctly. The state space is the 28x28 pixels available to represent an image. The action space is to classify an image as a digit in the range of 0-9. During training, the reward is the cross-entropy loss between the ground truth labels and the prediction; during testing, the reward is the accuracy of the trained classifier.

2 Related work

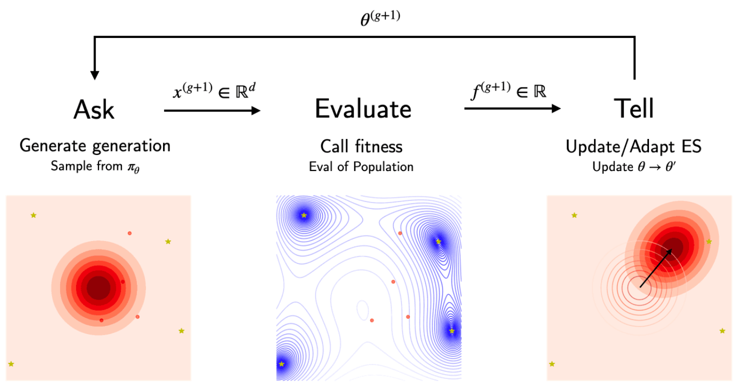
After some literature search (Sloss and Gustafson (2019), etc.), we have identified two primary classes of algorithms: those that use some parameterized proposal distribution, and those that use some non-parametric representation, i.e. population. The former class includes cross entropy method, natural evolution strategies, and CMA-ES; the second class includes genetic algorithms, cuckoo search, particle swarm optimization. While we focus on optimizing model weights with a fixed network topology, prior works have also focused on evolving network topology. (Gaier and Ha (2019), Stanley and Miikkulainen (2002)).

Jax (Bradbury et al. (2018) and EvoJAX(Tang et al. (2022)) enables neurorevolution algorithm to work with neural networks running in parallel cross hardware accelerators, cutting down run-time from many hour or days using CPUs down to within within an hour or even minutes using GPUs.

3 Methodologies and Algorithms

Evolutionary algorithms (refer to Figure 2) draw inspiration from biological evolution. They use the information about the performance of the previous generation's individuals to inform the construction of the next generation's individuals. The exact details of how the information is utilized and how the construction is done vary between algorithms, but generally, the probability of fitter individuals spawning in the next generation is increased over time.

Figure 2: Evolution Algorithm (Lange (2022))



In particular, we used variations of Cross Entropy Method and Genetic Algorithm to evolve our neural networks. CEM uses explicit proposal distribution whereas GA does not. Both are gradient-free algorithms adapted for neuroevolution. The neural network architecture for MNIST image recognition task is a convolutional neural network (CNN), whereas for Cartpole, a Permutatation-Invaraiant Multilayer Perceptron (PIMLP) is used(Tang and Ha (2021)). CNN has parameter count of 11,274 whereas PIMLP (Tang and Ha (2021)) has parameter count of 993; this difference in network size resulted in a much bigger population size for the Cartpole tasks as the GPU is able to hold more models.

3.1 Cross Entropy Method

We use multivariate normal distribution for the proposal distribution. For simplicity and efficiency, we used diagonal covariance matrix. At each generation, the mean and variance of every parameter is calculated over the top performing subset of the population; the updated parameters are then used to sample the individuals of the next generation. The parameter to be used for testing is the mean of the proposal distribution. To mitigate early convergence towards a local optima, we add time-varying Gaussian noise to the diagonals of the covariance matrix (Szita and Lörincz (2006)). Our initial proposal distributions are designed to have zero-mean and large variance to get a good coverage over the parameter space.

3.2 Genetic Algorithm

We use a simple version of GA that produces the next generation using only selection and mutation, with no crossovers; while linear interpolation is a common crossover technique for real-valued genetic representations, its applicability for neural network weights is dubious at best. Particularly, we use Gaussian mutation to perturb the parameters, scaled by some scalar hyperparameter that controls the magnitude of the perturbation. We experiment with truncation, tournament and roulette wheel selection methods to choose the parents of the next generation. We also use the elitism technique, where the best performing individual of a generation is copied into the next generation without mutation. The initial individual is just the zero vector, so this simple GA method just amounts to application of repeated Gaussian noise to a zero vector.

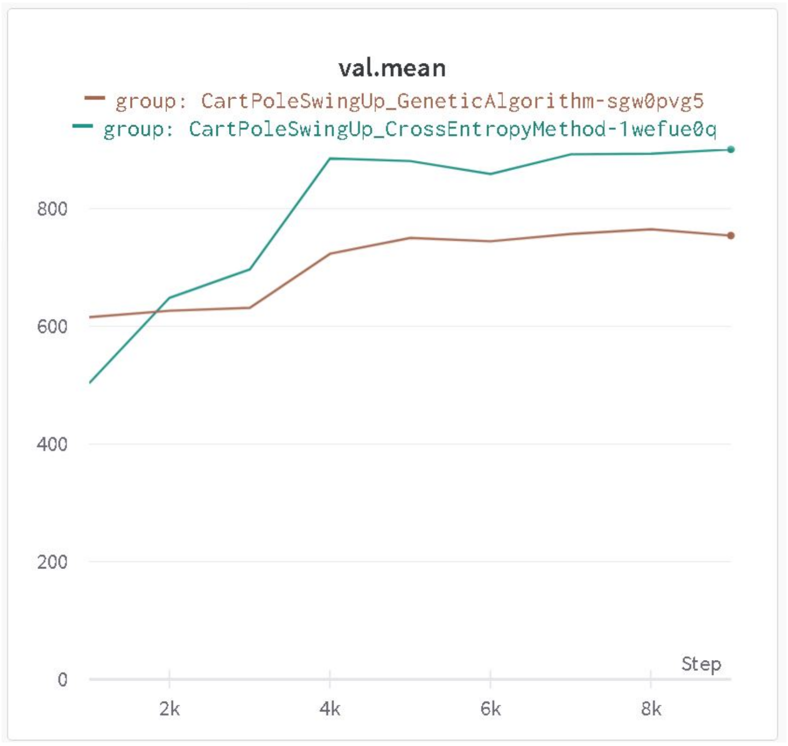
4 Experiments

Evaluation is based on the model performance in their respective tasks: total undiscounted reward of the policy in the Cartpole Swing Up (Manrique et al. (2020)) task, and classifier accuracy in MNIST classification task.

4.1 Reinforcement Learning Task With Cartpole

CEM outperforms GA. (900 vs. 754 total return on balancing task, refer to Figure 3), and actually learns near optimal policy (refer to CartPole_Swing_Up_CEM_Video), GA does not. Both algorithms

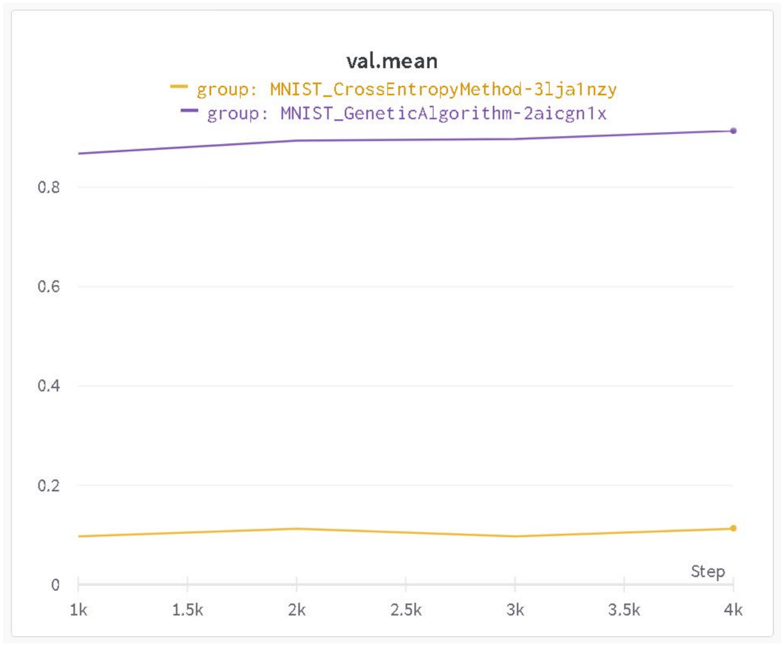
Figure 3: Cart-pole task GA vs. CEM



used population size of 128 and elite size of 32, with max iteration of 10,000. CEM took 54 minutes to train while GA took 51 minutes to train.

4.2 Computer Vision Task With MNIST

Figure 4: MNIST task GA vs. CEM



GA far outperforms CEM: 92% vs. 11% classification accuracy (refer to Figure 4). Both algorithms used population size of 64 and elite size of 16, with max iteration of 5,000. CEM took 11 minutes to train while GA took 18 minutes to train.

4.3 Selection Method Comparison for Genetic Algorithm

Multiple Genetic Algorithm selection methods are explored for both tasks: truncation, tournament and roulette wheel (aka fitness proportionate). Tournament selection method performed the best on MNIST image classification task (refer to Figure 5); truncation selection method performed the best on cartpole control task (refer to Figure 6). Roulette wheel selection method consistently underperformed across tasks.

Figure 5: GA Selection Methods Comparison on MNIST task

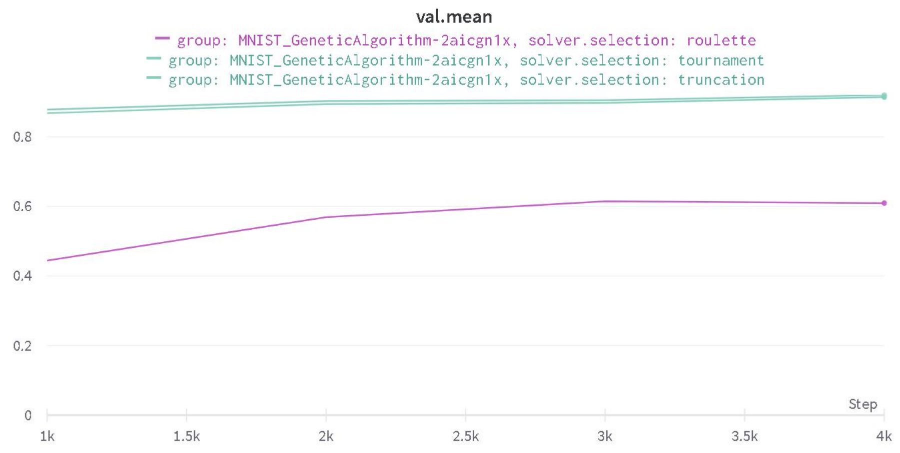
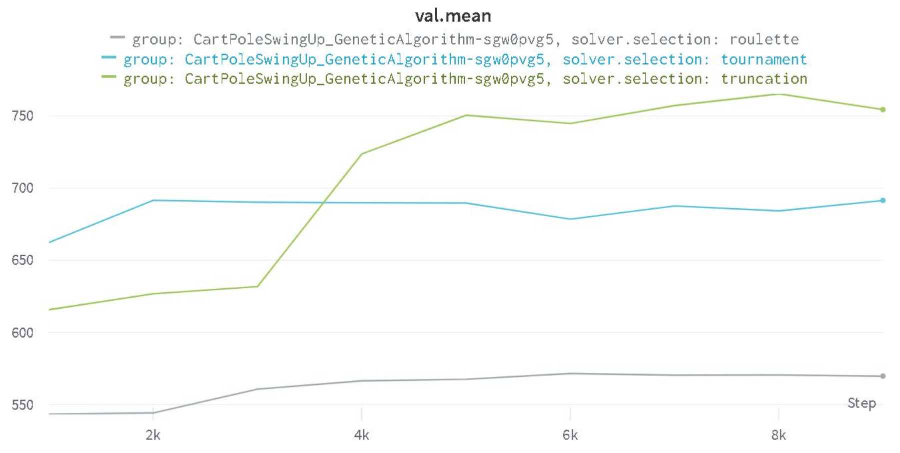


Figure 6: GA Selection Methods Comparison on MNIST Task



5 Software, Hardware, Implementation and Code Repository

Due to its simple interface for bulk-array programming and leveraging accelerators, We used Jax (Bradbury et al. (2018)) as our numerical computation backend. EvoJax (Tang et al. (2022)) is a Jax-based neuroevolution library that is optimized to efficiently run neuroevolution algorithms on hardware accelerators. It also has an intuitive interface for creating new algorithms and distributing tasks across many cores. We use Weights and Biases (Biewald (2020)) for experiment tracking and Hydra (Biewald (2020)) for configuration file management. We used one machine with 4

NVIDIA GeForce RTX 2080 Ti GPU, though each task used only a single GPU. Code respository: `https://github.com/chuanqichen/AA222_Final_Project`

6 Conclusion and future work

We chose to implement neuroevolution algorithms with evojax due to our shared interests in gradient-free optimization of neural networks as well as Jax. We successfully implemented two common zeroth-order optimization algorithms that can evolve both CNN and MLP networks to solve tasks from both computer vision as well as reinforcement learning domain. The usage of wandb and hydra allowed for much faster experiment iteration time, and Jax's design allowed us to effortlessly leverage our available computing resources.

For future work, we would like to experiment on the task of design and control co-optimization of soft robotics using EvoGym (Lange (2022).) We would also like to explore evolution of neural network topology, with less emphasis or even complete disregard for parameters. (Gaier and Ha (2019), Stanley and Miikkulainen (2002)) For visualization, we would like to explore VINE, a tool designed specifically for visualization of neuroevolution algorithms. (Wang (2018)).

References

Xie, Z.; Sato, I.; Sugiyama, M. *CoRR* **2020**, *abs/2002.03495*.

Sloss, A. N.; Gustafson, S. 2019 Evolutionary Algorithms Review. 2019; `https://arxiv.org/abs/1906.08870`.

Morse, G.; Stanley, K. O. *Proceedings of the Genetic and Evolutionary Computation Conference 2016* **2016**,

Cui, X.; Zhang, W.; Tüske, Z.; Picheny, M. *CoRR* **2018**, *abs/1810.06773*.

Manrique, C.; Pappalardo, C.; Guida, D. *Applied Sciences* **2020**, *10*, 9013.

Gaier, A.; Ha, D. Weight Agnostic Neural Networks. 2019; `https://arxiv.org/abs/1906.04358`.

Stanley, K. O.; Miikkulainen, R. *Evolutionary Computation* **2002**, *10*, 99–127.

Bradbury, J.; Frostig, R.; Hawkins, P.; Johnson, M. J.; Leary, C.; Maclaurin, D.; Necula, G.; Paszke, A.; VanderPlas, J.; Wanderman-Milne, S.; Zhang, Q. JAX: composable transformations of Python+NumPy programs. 2018; `http://github.com/google/jax`.

Tang, Y.; Tian, Y.; Ha, D. *arXiv preprint arXiv:2202.05008* **2022**,

Lange, R. T. evosax: JAX-based Evolution Strategies. 2022; `http://github.com/RobertTLange/evosax`.

Tang, Y.; Ha, D. The Sensory Neuron as a Transformer: Permutation-Invariant Neural Networks for Reinforcement Learning. 2021; `https://arxiv.org/abs/2109.02869`.

Szita, I.; Lörincz, A. *Neural Computation* **2006**, *18*, 2936–2941.

CartPole_Swing_Up_CEM_Video, CartPole Swing Up CEM Near Optimal Policy. `https://github.com/chuanqichen/AA222_Final_Project/blob/main/CartPoleSwringUp_CEM_NearOptimalPolicy.gif`.

Biewald, L. Experiment Tracking with Weights and Biases. 2020; `https://www.wandb.com/`, Software available from wandb.com.

Wang, R. VINE: An Open Source Interactive Data Visualization Tool for Neuroevolution. 2018.