Cultural Algorithm Guided Policy Gradient with Parameter Exploration

Mark Nuppnau[1], **Khalid Kattan**[2], **R.G. Reynolds**[1]

[1]Dept. of Computer Science Wayne State University
[2]Dept. of Computer Science University of Michigan - Dearborn

Abstract

This study explores the integration of cultural algorithms (CA) with the Policy Gradients with Parameter-Based Exploration (PGPE) algorithm for the task of MNIST handwritten digit classification within the EvoJAX framework. The PGPE algorithm is enhanced by incorporating a belief space, consisting on Domain, Situational, and History knowledge sources (KS), to guide the search process and improve convergence speed. The PGPE algorithm, implemented within the EvoJAX framework, can efficiently find an optimal parameter-space policy for the MNIST task. However, increasing the complexity of the task and policy space, such as the CheXpert dataset and DenseNet, requires a more sophisticated approach to efficiently navigate the search space. We introduce CA-PGPE, a novel approach that integrates CA with PGPE to guide the search process and improve convergence speed. Future work will focus on incorporating exploratory knowledge sources and evaluate the enhanced CA-PGPE algorithm on more complex datasets and model architectures, such as CIFAR-10 and CheXpert with DenseNet.

Methodology

Data Preparation and Model Architecture

The CA-PGPE algorithm is evaluated on the MNIST dataset (LeCun and Cortes 2010) and compared to the original PGPE algorithm (Sehnke et al. 2008). The MNIST dataset is a widely-used benchmark for hand-written digit classification, consisting of 60,000 training images and 10,000 testing images. Each image is a 28x28 gray-scale representation of a hand-written digit from 0 to 9. The dataset is preprocessed and normalized following standard practices.

The EvoJAX framework (Tang, Tian, and Ha 2022) provides an implementation of the PGPE algorithm for MNIST classification. The EvoJAX implementation for MNIST classification utilizes a convolutional neural network (CNN) as the policy network. The CNN architecture consists of two convolutional layers, each followed by a ReLU activation function and max pooling. The first convolutional layer has 8 filters with a 5x5 kernel size and *same* padding, while the second convolutional layer has 16 filters with the same kernel size and padding. After the convolutional layers, the feature maps are flattened and passed through a dense layer with 10 units, corresponding to the 10 digit classes. Finally, a log-softmax activation function is applied to obtain the output probabilities.

It is important to note that the MNIST dataset and the EvoJAX implementation were already available and well-established prior to this research. The focus of this work is on integrating CA (Reynolds 2021) with the existing PGPE algorithm within the EvoJAX framework to enhance the optimization process and improve classification performance.

PGPE Algorithm

Overview The PGPE algorithm is a gradient-based optimization technique used primarily as a reinforcement learning technique for robotic control type problems which typically involve controlling the movements and actions of a robot in a continuous space to perform tasks such as manipulation or navigation. However, it can be repurposed for problems such as MNIST classification. This is accomplished by setting the MNIST dataset as the task state that provides the input images to the policy network, a CNN as the policy state that represents the learnable parameters that define the mapping from input images to output predictions, and the predictions as the set of actions taken by the policy.

Unlike traditional policy gradient methods that typically operate by exploring the action space, PGPE explores the parameter space of a policy network. Traditional approaches, like REINFORCE, operate by using the parameters θ to determine a probabilistic policy $\pi_\theta(a_t|s_t) = p(a_t|s_t, \theta)$, which often results in high variance in the gradient estimates due to the probabilistic nature of action selection at every step. This conventional approach approximates the gradient of the objective function as:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta p(a_t^i|s_t^i, \theta) r(h_i) \qquad (1)$$

where $\nabla_\theta J(\theta)$ is the gradient of the expected reward with respect to the parameters, and $r(h_i)$ denotes the reward obtained from history h_i (Sehnke et al. 2008).

Copyright © 2024, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

To address the high variance issue inherent in traditional methods, PGPE adopts a different approach by replacing the probabilistic policy with a probability distribution over the parameters themselves. Instead of sampling actions from a policy parameterized by θ, PGPE samples θ from a distribution controlled by hyperparameters ρ, fundamentally altering the approach to policy exploration, represented as:

$$p(a_t|s_t, \rho) = \int_{\Theta} p(\theta|\rho)\delta_{F_\theta(s_t),a_t} d\theta \tag{2}$$

where ρ are the hyperparameters that define the distribution over the parameters θ, $F_\theta(s_t)$ is the deterministic action chosen by the model with parameters θ in state s_t, and δ represents the Dirac delta function, indicating that the action a_t is precisely the one chosen by $F_\theta(s_t)$ (Sehnke et al. 2008).

By employing this parameter space exploration strategy, PGPE effectively reduces the variance of the gradient estimate by detaching the stochasticity from the action selection process and embedding it into the parameter distribution. This allows the parameters of the CNN used for tasks like MNIST classification to be optimized more efficiently towards better performance, as the exploration now occurs in a continuous parameter space rather than the discrete space of possible actions.

Thus, through direct adjustment of the parameters based on sampled distributions and observing the resulting performance, PGPE can steer the learning process toward an optimal parameter space, potentially leading to faster learning and enhanced policy performance in comparison to traditional policy gradient methods.

Hyperparameters In the EvoJAX implementation of the PGPE algorithm, the navigation through the parameter space is controlled by three hyperparameters; center-lr, std-lr, and init-lr. For the MNIST classification task, the default values for these hyperparameters are 0.006, 0.089, and 0.039, respectively. These hyperparameters play crucial roles in controlling the exploration and learning processes.

The *center-lr* hyperparameter is the learning rate for updating the center of the Gaussian distribution, and is fundamental in dictating how quickly the algorithm adapts its policy parameters. The *std-lr* hyperparameter controls the learning rate for updating the standard deviation of the Gaussian distribution. This aspect of PGPE is critical for managing the exploration strategy of the algorithm. The *init-std* hyperparameter sets the initial standard deviation of the Gaussian distribution. This value determines the initial breadth of exploration.

By fine-tuning these hyperparameters, the exploration-exploitation trade-off can be balanced, the learning rate can be adapted to the specifics of the environment, and ultimately the algorithm can be guided toward more efficient and effective policy optimization.

Ask/Tell Mechanism The ask-tell mechanism, within the EvoJAX implementation of PGPE, is foundational to parameter space exploration. In this context, PGPE functions as the *solver*, a term used throughout the iterations of the algorithm.

During each iteration, the *ask* functionality provides a set of parameters. These inform actions (predictions), after which rewards are calculated. Subsequently, the *tell* functionality uses the outcomes to guide the algorithm towards optimal distribution areas, leveraging the best-performing individuals.

Parameter sets generated by the *ask* are sampled from a Gaussian distribution centered around the current policy parameters, promoting variation and exploration. This exploration is governed by the parameters' mean (μ) and standard deviation (σ), adapted based on performance feedback:

$$\Delta\mu_i = \alpha(r - b)(\theta_i - \mu_i) \tag{3}$$

$$\Delta\sigma_i = \alpha(r - b)\frac{(\theta_i - \mu_i)^2 - \sigma_i^2}{\sigma_i} \tag{4}$$

where α is a constant step size, r is the reward, b is a baseline subtracted from the reward to reduce variance, θ_i represents the parameters of the individual, μ_i is the mean, and σ_i is the standard deviation of the Gaussian distribution from which parameters are sampled. The updates aim to refine exploration by adjusting the distribution's center and scale according to the performance (reward) feedback.

Symmetric perturbations ensure balanced exploration. Each parameter set is paired with a counterpart, mirroring parameter alterations in the opposite direction, aiding in a more accurate gradient estimation of the performance landscape.

Post-environment interaction, the *tell* function evaluates individual fitness based on environmental rewards. These assessments inform updates to μ and σ, steering the algorithm towards more effective parameter configurations. The learning process, fueled by iterative adjustments via the ask and tell functionalities, navigates the parameter space towards optimal policy parameters, enhancing agent performance in complex environments.

Cultural Algorithms

Overview Cultural algorithms are an extension of evolutionary algorithms, specifically genetic algorithms, to better model the process of cultural evolution. They were inspired by the idea that cultural knowledge and values are transmitted and evolve in a manner similar to biological evolution. The purpose of cultural algorithms is to solve complex optimization problems that are difficult to solve using traditional methods by providing a belief space, which represents the cultural knowledge and values that are relevant to the problem at hand, to help guide through a search space.

Integration With the mean, also referred to as center, (μ) and standard deviation (σ) of the distribution playing such an important role in the navigation through the parameter space, our research focuses on integrating CA to help guide the iterative updates to center and standard deviation values based on Domain, Situational, and History knowledge sources. We adjust equations 3 and 4 as follows:

$$\begin{aligned}\Delta\mu_{i,\text{cultural}} = 0.45 \times \alpha(r-b)(\theta_i - \mu_{\text{KS}}) \\ + 0.55 \times \alpha(r-b)(\theta_i - \mu_i)\end{aligned} \tag{5}$$

$$\begin{aligned}\Delta\sigma_{i,\text{cultural}} = 0.45 \times \alpha(r-b)((\theta_i - \mu_{\text{KS}})^2 - \sigma_{\text{KS}}^2) \\ + 0.55 \times \alpha(r-b)((\theta_i - \mu_i)^2 - \sigma_i^2)\end{aligned} \tag{6}$$

where α represents the learning rate, r is the reward, b is the baseline, θ_i denotes individual parameters, and $\mu_{\text{KS}}, \sigma_{\text{KS}}$ are the knowledge source-informed center and standard deviation, respectively. The weighting factors (0.45 for cultural knowledge and 0.55 for individual or current parameters) balance between learned cultural knowledge and individual exploration.

We also measure the magnitude of the perturbations to the center when parameters are sampled from a Gaussian distribution. This magnitude is used to guide the perturbations of a small subset of individuals each iteration.

In each iteration, center and standard deviation values are updated, guiding the parameter distribution for the population. To maintain exploration symmetry and enhance informed decision-making, *scaled_noises* are used to generate parameter sets for individuals. These noises are then culturally adjusted:

For each individual, a noise magnitude is calculated based on the norm of their associated noise vector:

```
for i, noise in enumerate
  (self._scaled_noises):
       magnitude = jnp.linalg.norm(noise)
       self.population_space.individuals[i]
       .noise_magnitude = magnitude
```

Using predefined indexes for each Knowledge Source (KS), the *scaled_noises* at these indexes are updated based on the corresponding individuals' noise magnitudes within the KS. For instance, if the Situational KS is assigned indexes $[5, 10, 20, 22]$, then for each index, the scaled noise is modified by the respective individual's noise magnitude in this KS.

Subsequently, solutions are computed as follows, integrating cultural influences into the generation of scaled noises:

```
next_key, key = random.split(self._key)
scaled_noises = random.normal(key,
    [self._num_directions,
    self._solution_size])
    * stdev
self._scaled_noises =
    self.belief_space.influence(
    scaled_noises)
self._solutions = jnp.hstack(
    [center + self._scaled_noises,
    center - self._scaled_noises]
    ).reshape(-1, self._solution_size)
```

The number of indexes assigned to each Situational and History KS for this study is one for every twelve individuals in a population. The Domain KS has a single index assigned to it because its population size is always one. The effect of this on a population of 64 is the influencing of 11 individuals. Increasing the number of indexes assigned to more than one for every eight individuals in a population hinders the initial exploration and the algorithm has difficulty converging to an optimal parameter space.

Experiments and Results

Hardware Specifications

The experiments were conducted on a personal computer running Arch Linux operating system. The system is equipped with the following hardware components:

- Processor (CPU): AMD Ryzen 5800X, an 8-core/16-thread CPU with a base clock speed of 3.8 GHz and a boost clock speed of 4.7 GHz.
- Graphics Processing Units (GPUs): Two NVIDIA RTX-3090 GPUs, each with 24 GB of GDDR6X memory. The RTX-3090 is a high-performance graphics card designed for deep learning and computationally intensive tasks. The presence of two GPUs allows for massive parallelization, which is a key feature of the JAX library used in EvoJAX. The PGPE algorithm can leverage the parallel processing capabilities of the GPUs to significantly accelerate the training and evaluation of the CNN model.
- Random Access Memory (RAM): 128 GB of DDR4 RAM running at 3600 MHz.
- Storage: A 4 TB Western Digital Black NVMe solid-state drive (SSD) is used as the primary storage device. The NVMe SSD provides fast read and write speeds, enabling quick access to data and reducing I/O bottlenecks.

The combination of the powerful AMD Ryzen 5800X CPU, two high-end NVIDIA RTX-3090 GPUs, ample high-speed RAM, and fast NVMe SSD storage creates an ideal computing environment for leveraging the capabilities of the EvoJAX framework and the PGPE algorithm. JAX, the library on which EvoJAX is built, is designed for high-performance numerical computing and supports just-in-time (JIT) compilation, automatic differentiation, and parallelization. The PGPE algorithm implemented in EvoJAX takes full advantage of these features, allowing for efficient and scalable training of the CNN model on the MNIST dataset.

Results

To summarize the algorithms parameters mentioned throughout this paper for this experiment:

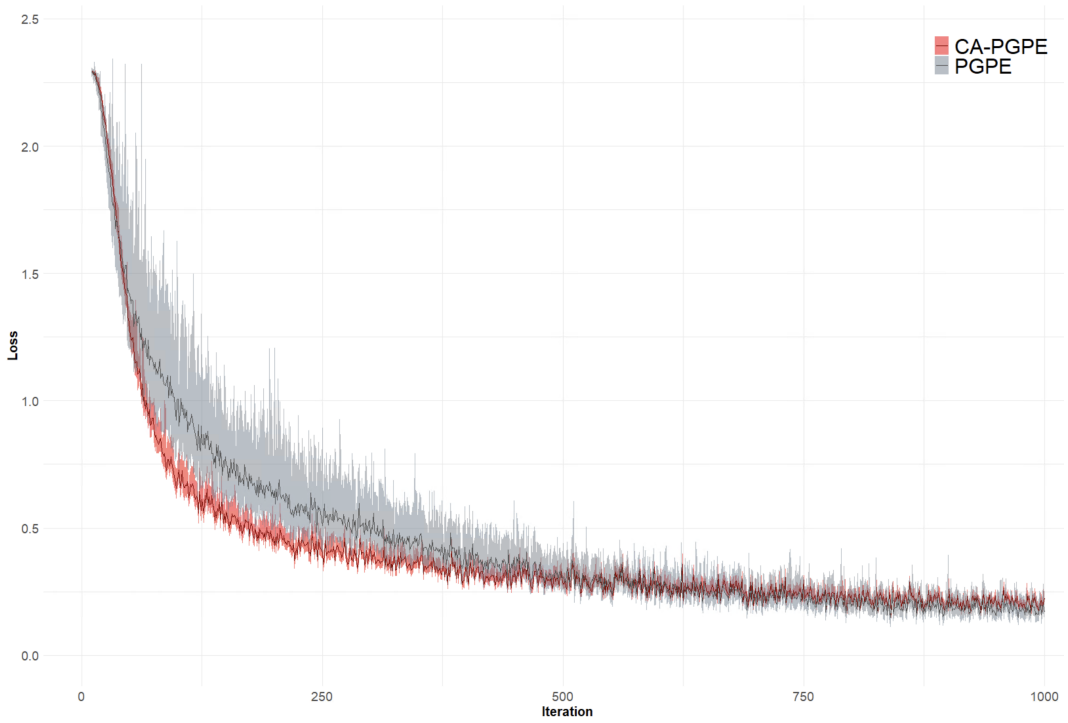

Figure 1: Training Loss: PGPE and CA-PGPE

- The base population consists of 64 individuals
- The center-lr is set to 0.006, std-lr to 0.089, and init-std to 0.039
- The algorithm runs for 1,000 iterations
- For each iteration that performs worse than the previous, the knowledge sources influence the center and standard deviation values with a weight of 0.45
- Each iteration, the knowledge sources influence the scaled noises of 11 individuals
- The History KS uses a decay factor of 0.8
- The Situational KS keeps information from the top 100 individuals

As shown in 1, early iterations (0-300) demonstrate quicker convergence to better-performing parameter spaces for CA-PGPE compared to the original PGPE, attributed to the integration of exploitative KS. However, as iterations approach 1,000, performance diverges, likely due to the absence of exploratory KS such as Topographic and Normative, highlighting the balance between exploration and exploitation. The lack of exploration can also be observed by how CA-PGPE maintains a much more narrow range of loss values across the population for each iteration.

At 1,000 iterations, evaluation is done on the validation set. Due to CA-PGPE being currently equipped with only explotative KS, more optimal parameter spaces are missed after 500 iterations. With PGPE maintaining a wider range of loss values amongst its individuals for each iteration, its able to find the more optimal parameters spaces over 500 iterations. Thus, at 1,000 iterations, CA-PGPE loss values are still within range of the PGPE loss values, but are at the higher end of that range. The PGPE algorithms achieves .9636 test accuracy after 1,000 iterations while CA-PGPE achieves .9539.

Ablation Study

With multiple adjustments made from the original PGPE algorithm to CA-PGPE, it's important to understand each adjustment. An ablation study was conducted to improve this understanding. We look at the overall performance without each KS over 1,000 iterations. We also look at the performance without adjustments to center and standard deviation values and only scaled noises, and vice versa.

Performance without Each KS The overall performance after 1,000 iterations with the removal of any of the three KS is worse than with all three, but there are subtle differences between the performances without each of the KS, as we can observe in 2.

The worst performance overall comes from removing the Situational KS. After 1,000 iterations, CA-PGPE achieves 0.9414 test accuracy without the SItuational KS. This is similar to CA-PGPE without Domain KS which achieves 0.9417 test accuracy.

However, CA-PGPE without Domain KS remains more exploratory as seen in the wider range of loss values up to 250 iterations. It's most noticeable around iteration 70, where CA-PGPE without Domain KS loss values are still within range of CA-PGPE and CA-PGPE without Situa-

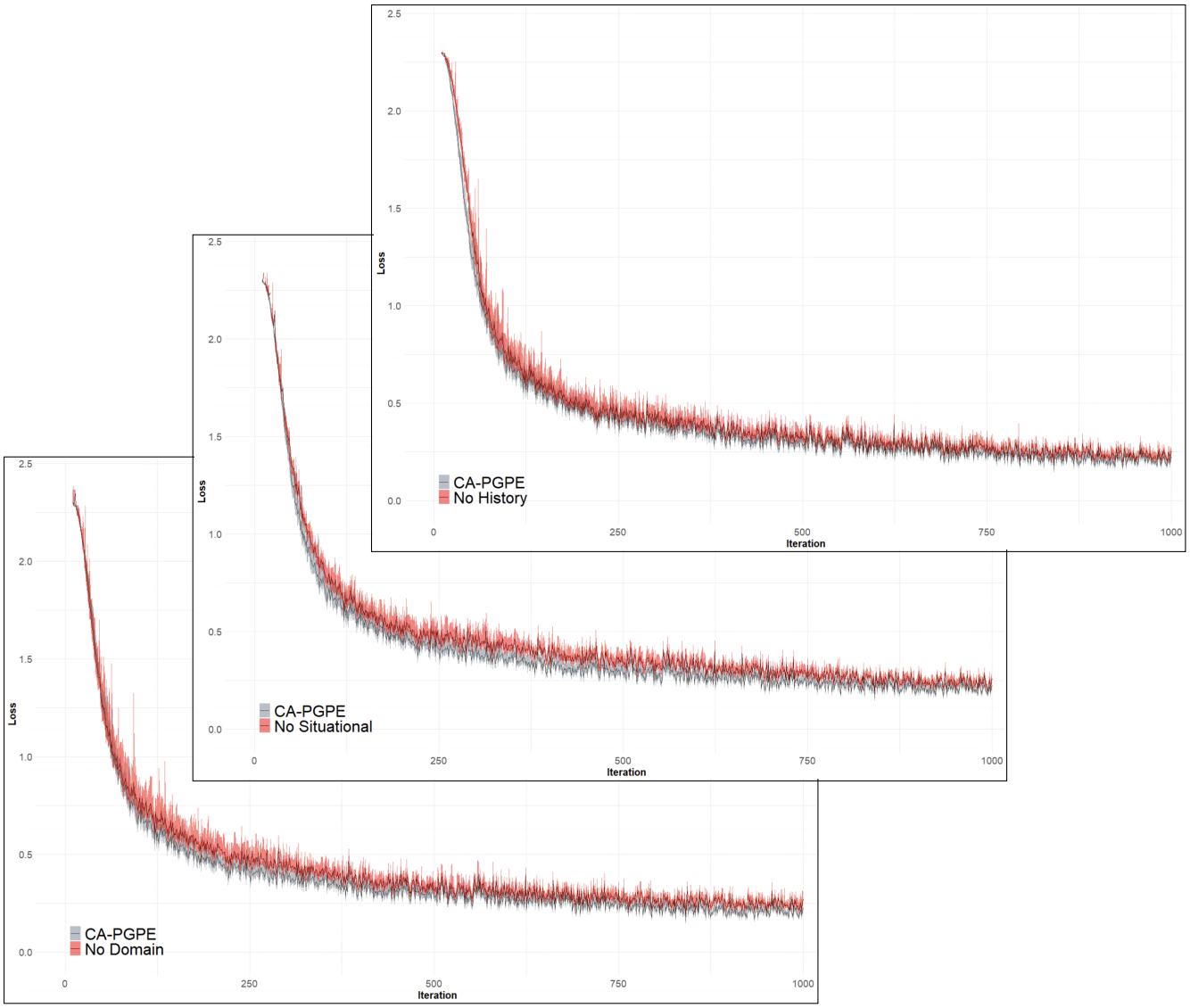

Figure 2: Training Loss: CA-PGPE and CA-PGPE without each KS

tional KS are not. This confirms that the Domain KS restricts exploration by focusing on a single individual from the most recent iteration.

The best performance overall is CA-PGPE without History KS which achieves 0.9468 test accuracy after 1,000 iterations. Despite the overall better performance, the removal of the History KS shows worse performance in early iterations.

This provides a couple insights into the HIstory KS. In early iterations, 1 to 50, individuals from a handful of recent iterations (not just the most recent iteration as given by Domain KS and not all previous iterations as given by Situational KS in early iterations) helps find an optimal parameter space.

Also, the decay factor of 0.8 used to weight the averaging of center values may need to be adjusted to lower the weights applied to individuals from the earliest iterations.

Performance without Center and Standard Deviation Adjustments Next, we test the overall performance of the algorithm while keeping all knowledge sources but removing the influence of center and standard deviation values, or noise scaling values. While the results in 3 seem self-explanatory, there are some key insights collected from these experiments.

Starting with the top-right plot in 3, we are comparing the PGPE algorithm performance against the CA-PGPE algorithm without adjusting center and standard deviation values via KS. The only influencing done by the set of KS is on

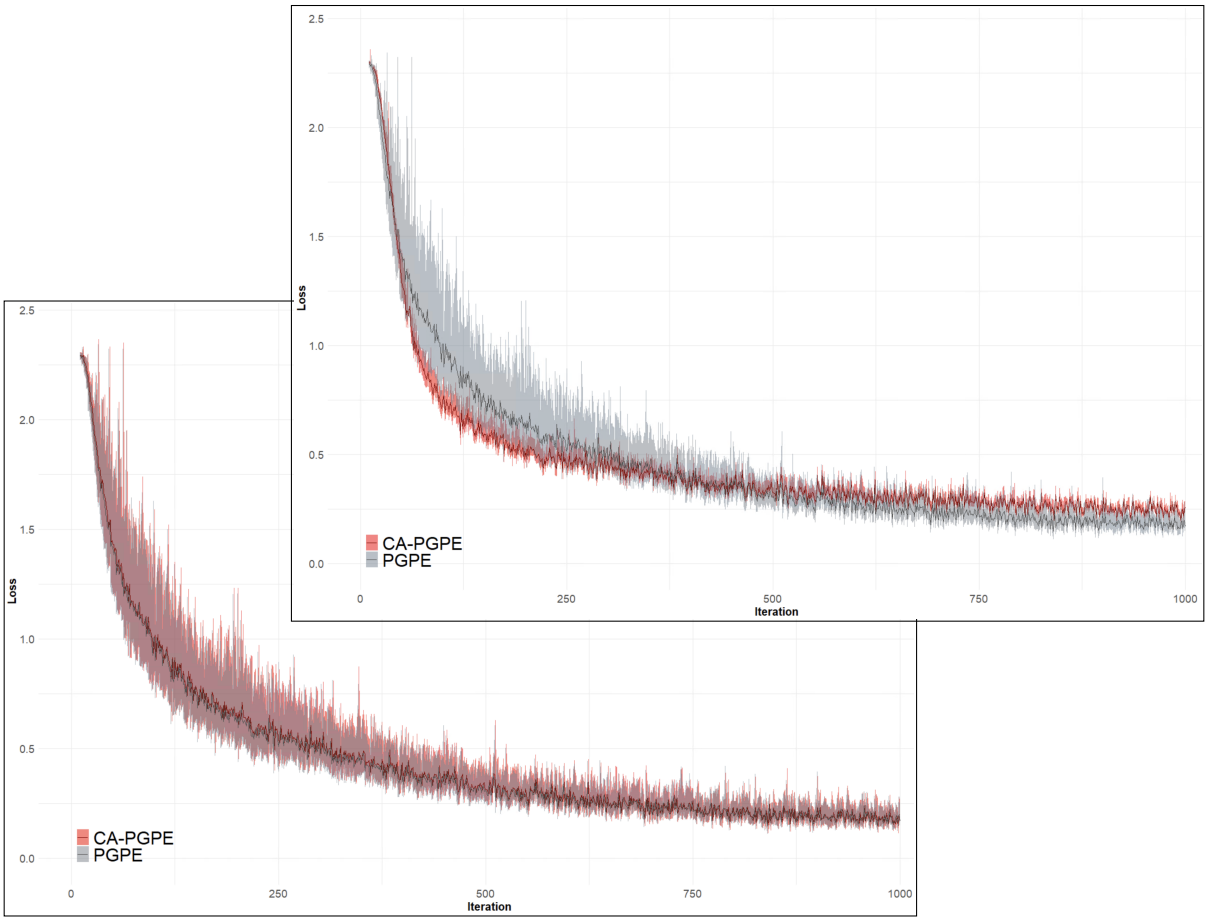

Figure 3: Training Loss: PGPE and CA-PGPE without scaled noises adjustments (bottom-left) and CA-PGPE without center and standard deviation adjustments (top-right)

scaled noise values.

This plot looks similar to 1, but the overall performance is worse which is visible at two main points. After 100 iterations, there is no separation from the PGPE algorithm. At iteration 500, the CA-PGPE algorithm loss values are near the top of the PGPE loss value range, and continues to diverge from there. This CA-PGPE algorithm achieves 0.9389 test accuracy, worse than removing any of the individual KS.

In the bottom-left plot in 3, we are comparing the PGPE algorithm performance against the CA-PGPE algorithm without adjusting scaled noise values via KS. The only influencing done by the set of KS is on center and standard deviation values. The overall performance is very similar to that of the original PGPE algorithm. This suggests the impact of center and standard deviation adjustments may be overshadowed by the inherent exploration and learning capabilities of the PGPE algorithm itself.

Limitations and Future Work

As mentioned in the abstract and as observed in the results, the more exploratory Topographic and Normative KS are not implemented in this study. Furthermore, the weighting of the implemented KS is static throughout algorithm. The next step in our research includes both the addition of the Topographic and Normative KS, and a cyclic weighting system (i.e., sine function) to dynamically weight each KS over the course of the iterations.

Our expectations are that these updates will provide more exploration in general to prevent the algorithm from plateauing early on by alternating between exploitation and exploration phases during the search process. The algorithm will start by increasing weights assigned to each exploitative KS while reaching peak amplitude around 150 iterations and then beginning to assign more weights to each exploratory KS, with the period of the function being roughly 600 iterations. This will allow the algorithm to strike a balance between leveraging the accumulated knowledge and exploring new regions of the search space.

With an optimal implementation of CA-PGPE for MNIST hand-written digit classification, our vision for near-term research involves performing these same experiments using higher-dimensional data and parameter spaces. This in-

cludes applying CA-PGPE to the CIFAR-10 and CheXpert (Irvin et al. 2019) task states, and DenseNet policy state. We've conducted some preliminary work on these task and policy states with the original PGPE algorithm.

The complexity of the search space of CheXpert and DenseNet seems too difficult of a task for the original PGPE algorithm. The CIFAR-10 dataset is higher-dimensional data space then MNIST but lower-dimensional data space than CheXpert which makes it a good candidate for testing an algorithms ability to navigate through a search space. The original PGPE algorithm takes over 50,000 iterations to achieve a test accuracy over 0.70 for the CIFAR-10 task and DenseNet policy. We hypothesize that CA-PGPE improves convergence in more complex search spaces.

With evidence of CA improving navigation through high-dimensional search spaces, our longer-term vision is to implement the CoDeepNEAT algorithm (Miikkulainen et al. 2017) in the EvoJAX framework and integrate CA into CoDeepNEAT. In some preliminary work, we have shown CoDeepNEAT to evolve DenseNet-like model architectures over four generations with improved performance over each generation for the task of pathology detection using the CheXpert dataset. We hypothesize that CA can improve the convergence toward an optimal model architecture that is highly-efficient and performant.

References

Irvin, J.; Rajpurkar, P.; Ko, M.; Yu, Y.; Ciurea-Ilcus, S.; Chute, C.; Marklund, H.; Haghgoo, B.; Ball, R.; Shpanskaya, K.; Seekins, J.; Mong, D. A.; Halabi, S. S.; Sandberg, J. K.; Jones, R.; Larson, D. B.; Langlotz, C. P.; Patel, B. N.; Lungren, M. P.; and Ng, A. Y. 2019. CheXpert: A Large Chest Radiograph Dataset with Uncertainty Labels and Expert Comparison. arXiv:1901.07031.

LeCun, Y.; and Cortes, C. 2010. MNIST handwritten digit database.

Miikkulainen, R.; Liang, J. Z.; Meyerson, E.; Rawal, A.; Fink, D.; Francon, O.; Raju, B.; Shahrzad, H.; Navruzyan, A.; Duffy, N.; and Hodjat, B. 2017. Evolving Deep Neural Networks. *CoRR*, abs/1703.00548.

Reynolds, R. G. 2021. *Cultural Algorithms: Tools for the Engineering of Social Intelligence into Complex Cultural Systems,*.

Sehnke, F.; Osendorfer, C.; Rückstieß, T.; Graves, A.; Peters, J.; and Schmidhuber, J. 2008. Policy Gradients with Parameter-Based Exploration for Control. In Kůrková, V.; Neruda, R.; and Koutník, J., eds., *Artificial Neural Networks - ICANN 2008*, 387–396. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-540-87536-9.

Tang, Y.; Tian, Y.; and Ha, D. 2022. EvoJAX: hardware-accelerated neuroevolution. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '22. ACM.