A geometric encoding for neural network evolution

Paul Templier, Emmanuel Rachelson, Dennis G Wilson

A Geometric Encoding for Neural Network Evolution

Paul Templier[1], **Emmanuel Rachelson**[1], **and Dennis G. Wilson**[1]

[1]ISAE-SUPAERO, Université de Toulouse, Toulouse, France

A major limitation to the optimization of artificial neural networks (ANN) with evolutionary methods lies in the high dimensionality of the search space, the number of weights growing quadratically with the size of the network. This leads to expensive training costs, especially in evolution strategies which rely on matrices whose sizes grow with the number of genes. We introduce a geometric encoding for neural network evolution (GENE) as a representation of ANN parameters in a smaller space that scales linearly with the number of neurons, allowing for efficient parameter search. Each neuron of the network is encoded as a point in a latent space and the weight of a connection between two neurons is computed as the distance between them. The coordinates of all neurons are then optimized with evolution strategies in a reduced search space while not limiting network fitness and possibly improving search.

Correspondence: ***paul.templier@isae-supaero.fr***

1: Introduction

Artificial neural networks are an attractive representation for function approximation due to their ability to represent a wide variety of functions when using appropriate parameters. The problem of finding the ANN parameters θ, to represent a given function $F(x)$, can be cast as an optimization task. In supervised learning, the function represented by the current network parameters $H_\theta(x)$ is compared to a desired function $F(x)$ and $L(F(x), H_\theta(x))$ is minimized. The neural network function can also be used to solve a sequential decision making task. In this case, the optimization goal is to reach a certain outcome, such as winning a game, where the function is used to decide the actions taken in the game. Formally, we can consider an environment as a Markov Decision Process (26) where taking an action a in a state s results in receiving a reward $r(s,a)$, and reaching state s' with probability $p(s'|s,a)$. We consider the neural network with parameters θ as a policy π_θ which determines the distribution $\pi_\theta(a|s)$ over the actions to take in state s. The goal is then to maximize the total reward over an episode in the environment:

$$R = \sum_{t=0}^{\infty} r(s_t, \pi_\theta(|s_t)). \quad \textbf{(1)}$$

Evolutionary algorithms have historically been used to search for ANN-based agents which maximize total reward in simulated environments. Recently, with the advent of deep learning and specifically deep reinforcement learning, renewed interest has been shown in applying evolutionary methods to more challenging benchmark tasks such as the Atari Learning Environment (1). Evolutionary strategies (27) in particular are well-suited for the continuous optimization of ANN parameters and CMA-ES has been previously applied to the Atari benchmark (17).

However, neural networks are often composed of thousands, if not millions, of parameters. The use of evolutionary strategies on each parameter of the network risks inefficiency in computing population-wide gradient estimates such as the covariance matrix of CMA-ES, and inability to scale to larger networks. Furthermore, ANN weights often include a level of redundancy which could be exploited to reduce the parameter search space.

In this work, we propose GENE, a novel encoding method intended for continuous optimization of ANN parameters through evolutionary strategies. In this encoding, neural network weights are represented as the distance between nodes whose position is optimized, reducing the dimensionality of the search problem to a linear factor of the number of neurons, not of the number of weights. We show that this compact parameter representation improves search and enables the application of evolutionary strategies with costly population estimates, notably XNES, to ANNs with high dimensionality.

The structure of this article is as follows. In section 2, we cover related research in the evolution of neural networks, both for parameter and architecture search, highlighting different parameter encoding methods. In section 3, we present the GENE method, including a hyperparameter study in different distance metrics for weight calculation. In section 4 we compare evolution of an ANN using a direct encoding and the GENE encoding on the Atari benchmark (1) with multiple different evolutionary strategies. Finally in section 5, we conclude that the GENE representation allows for more efficient neuroevolution and propose future directions for this research.

2: Related work

Neuroevolution (NE) focuses on training neural networks with evolutionary methods, seeing it as a black-box optimization problem. The use of evolutionary algorithms for optimizing ANNs has a long history (24) which has grown as ANNs and computing power have changed. Different approaches focus on evolving only the architecture of the network, as in the field of *Neural Architecture Search* (21), on optimizing the parameters (synaptic weights and neuron biases) of fixed architectures, or on optimizing both the architecture and parameters in a single evolution, as in NEAT (34). Our work belongs to the second group since it focuses on op-

timizing the parameters of networks with predefined architectures.

A fundamental question in neuroevolution is the representation of ANNs as a genome, both in optimizing architecture and parameters. Two types of approaches can be identified: direct encoding, which directly represents neural network parameters in a genome, and indirect encoding, which uses a transformation to interpret the ANN from a genome.

A. Direct encoding.

Direct encoding in neuroevolution consists in mapping the parameters of the network to be optimized (*e.g.* position of neurons, connectivity, weight, bias...) to a specific gene, each gene encoding only one parameter. Hence, mutating one gene only changes one part of the network and the number of genes to optimize grows with the size and complexity of the network.

For methods which optimize both the architecture and weights of a network, direct encoding methods use a mixed genome of binary or discrete variables, e.g. for encoding the presence of a synapse, and continuous values, e.g. for a synaptic weight. A well-known example is the NEAT algorithm (34) which encodes links as integers and weights and biases as floating point values. Similarly, in CGPANN, connections and activation functions are encoded by integers and weights by floating point values (18). For these approaches, the mix of discrete and continuous values in the same genome constrains the choice of evolutionary algorithm and general optimization methods (e.g., a genetic algorithm for NEAT and the $1+\lambda$ EA for CGPANN). The network parameter optimization therefore often uses uninformed mutation operators, drawing new parameters from a uniform distribution.

The optimization of neural weights and biases in a direct encoding can be considered a continuous optimization problem, akin to gradient descent (30) on weights in a fixed architecture network. Although genetic algorithms have been used for this purpose (24, 31, 35), evolutionary strategies (ES) such as Natural Evolution Strategies (NES) (38) and the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) (16) offer useful benefits. These methods may be better suited for the continuous optimization task of ANN parameter search as they use approximations of the gradient of the search space to inform population movement. ES has been succesfully applied to neuroevolution using direct encoding for control tasks (13) and in Atari game playing (4, 17).

While neuroevolution with direct encoding ensures that, with a long enough search, any optimal policy of the policy space can be reached, its main drawback lies in the high dimensionality of the search space. Indeed, the size of the genome is equal to the number of parameters, which grows quadratically with the size of the network, and some matrix-based optimization methods such as Exponential NES (XNES) (38) or CMA-ES may themselves scale quadratically on the number of parameters. While small networks are manageable, more complex architectures require larger computational resources and longer training time, due to expensive matrix operations. Although this can be reduced by the use of methods developed for high dimensions and hence linear in complexity with the number of parameters such as separable NES (SNES) (38), these methods are not suited for problems with highly dependent variables, which is the case for neural network parameters.

B. Indirect encoding.

The goal of indirect encoding is to reuse genome information multiple times to generate neural network parameters, reducing the number of genes needed to represent the neural network and producing a smaller search space, termed a *latent space*. A key characteristic of a mapping from the latent space to the parameter space is its ability to improve the reachability of optimal solutions by reducing the complexity of the search in the latent space.

Reusing genetic material multiple times to create the final networks, while compressing information, also brings regularities into the final phenotype, allowing the network to exploit regularities in the problem. Moreover, indirect encoding methods like generative encoding can reuse the same genome for multiagent control tasks, sharing traits without needing them to appear in each agent separately (8).

The encoding of a network parameters can be described as:

$$\theta = f_\phi(\sigma), \tag{2}$$

with θ the parameters of the final network (the phenotype), f_ϕ the mapping function defined by the parameters ϕ, and σ is the set of parameters on which f_ϕ is evaluated to generate the network's weights. Searching for the optimal θ value then comes down to optimizing f_ϕ and σ together. In direct encoding methods, f_ϕ is the identity function and σ the genome.

An approach to develop an indirect encoding method is to automatically search for the mapping f_ϕ, as in (19). For this, a smaller network, called a *generator network*, can be used to produce the weights of each connection in the final fixed-size network; search such as evolution can then be used to find the generator network. An example of this is HyperNEAT (33), where a Compositional Pattern-Producing Network (CPPN) (32) infers the weight of each connection between 2 neurons from their locations (layer, position in the layer) in the final network. HyperNEAT has been applied to many different problems and is still a state-of-the-art method for neuroevolution (10).

Inspired by HyperNEAT, other generator network types and training methods have been used. In HyperNEAT (33), the generator network (CPPN) is evolved through NEAT and is similar to the ANNs evolved by NEAT, but includes multiple possible activation functions. In (11), the Differentiable CPPN uses NEAT to evolve the generator network architecture but trains the generator weights using backpropagation. In (14), a fixed generator network is used and is trained with backpropagation, showing that f_ϕ can be optimized through other methods than evolution.

In all these cases, the generator is trained and then used to determine the weights of a fixed-size final network. The SMASH method (3) combines neural architecture search on

the final network with a generator network, generating the weights instead of training a model to evaluate each architecture.

These algorithms use the position of the neurons as a fixed σ, optimizing ϕ to generate the f_ϕ mapping function. Other developmental methods like grammatical evolution or cellular encoding (37) can be linked to this approach since they evolve rules f_ϕ that generate the final network from the initial minimal network σ by being applied recursively onto the developing network until a termination criteria is met.

In Evolvable Substrate HyperNEAT, (28), the node positions in the latent space, σ are discovered during search in addition to the CPPN, which represents f_ϕ. While this secondary search is expensive, iterative discovery of node positions during search demonstrated superior performance to HyperNEAT with fixed positions (29). It was also demonstrated in this work that the search for f_ϕ can inform the search for σ.

A second approach relies on a fixed predetermined mapping function f_ϕ and the optimization of σ in the latent space. Such methods include transforms such as Discrete Cosine Transform (DCT) (20), Discrete Fourrier Transform (DFT), or wavelet-based encoding (36), to ignore high-frequency coefficients in the gene space. A goal of these methods is to provide continuity in the genotype-phenotype mapping, such that small changes to a genotype do not cause large changes in its phenotype. By using a fixed mapping function f_ϕ, this problem can be avoided.

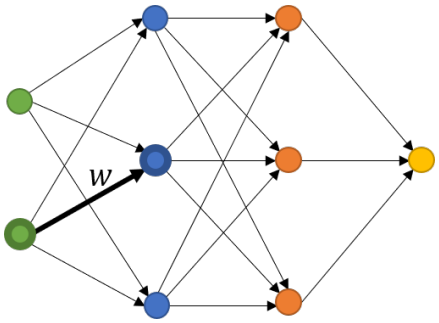
These methods have used the Cooperative Synapse NeuroEvolution algorithm (CoSyNE, (12)) for optimizing σ. While this algorithm does use a probabilistic mutation to transform search based on parameter importance, it does not perform gradient estimation as in ES. (36) studies the wavelet encoding using SNES (38), demonstrating the advantage of continuous optimization algorithms like ES for σ optimization.

In this work, we present the Geometric Encoding for NeuroEvolution (GENE) method where f_ϕ is fixed and the σ is projected into a latent space where neuron positions are optimized. As such, the size of σ grows linearly with the total number of neurons in the network. Approaches like HyperNEAT which optimize f_ϕ often use a latent space of variable size, expanding through the NEAT algorithm, but not directly related to network complexity. Methods which optimize σ often use latent spaces which correlate with the number of total parameters, but the compression level is a hyperparameter of the method. For example, in (20), $c \leq N$ DCT coefficients are used, where N is the total number of parameters and reducing c offers more compression but can negatively impact search. In GENE, the size of the latent space is directly related to the number of neurons in the defined architecture, n; it is $n(D+1)$ in this paper, where the coordinate dimension is D and biases are used.

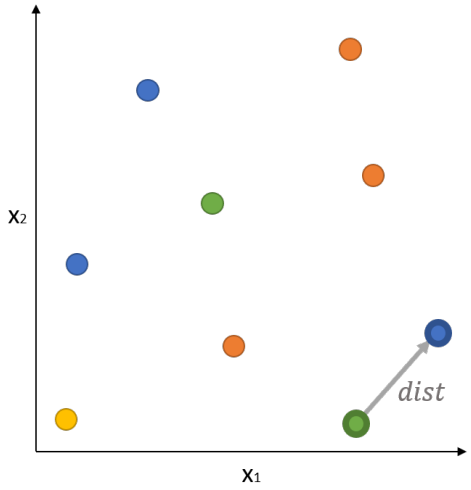
The advantages of GENE over other indirect encoding methods of σ are its simplicity and flexibility. In this work, we present GENE and study it as an encoding method when compared to direct encoding using the same evolutionary algorithms. Comparison with other indirect encoding methods which use different evolutionary algorithms, such as HyperNEAT, is left as future work.

3: Geometric Encoding for NeuroEvolution

GENE operates by projecting each neuron of a fixed-size, fully-connected network onto a latent Euclidean space of small dimension (typically 2 to 10). Distance functions between two neurons in the latent space are used to compute the weight of a connection between two neurons (Figure 1).



(a) Neural Network with 2 input neurons *(green)* , 2 hidden layers of 3 neurons *(blue and orange)* and 1 output neuron *(yellow)*.



(b) Neurons in a *coordinates* space with 2 dimensions (x_1 and x_2). The weight of the highlighted connection w in Figure 1a is determined by the distance between the 2 neurons in the *coordinates* system.

Fig. 1. Fully-connected feed-forward neural network (Figure 1a) and its representation in GENE (fig Figure 1b).

Using the coordinates of each neuron in the latent space multiple times to generate the network allows for a smaller genome, since its size grows linearly with the number of neurons, as opposed to quadratically. This is particularly important for networks of fully-connected layers of neurons, but could also be applied to recurrent networks or convolutional neural networks.

The coordinates of all neurons in the latent space are concatenated into a genome and optimized by evolution. Hyper-

parameters such as the network architecture, activation function, and weight distance function are configured and are not optimized. It is important to note that, although the term "coordinates" is used to describe the parameters of a neuron in the latent space to explain the introduction of a distance function, the positions of the neurons in the network and its architecture do not depend on the evolution: they are fixed at the beginning of evolution.

A. Distance functions.

In GENE, a configured distance function creates the mapping between the genotype and the phenotype of the individual. It determines the weight of a connection between 2 neurons from their *coordinates*:

$$w_{i,j} = dist(g_i, g_j) \tag{3}$$

with g_i and g_j the genomes of neuron i and j respectively, and $w_{i,j}$ the weight of the connection between them. Biases are optimized through direct encoding, since they are directly added to the genome next to the neuron *coordinates*.
In this work , we study three distance functions, but any mapping from a Euclidean space to a real number could in theory be used. We first offer a study on a simple distance function, the Euclidean distance, which we term L2-GENE.
L2-GENE computes the weight of a connection from neuron n_1 to neuron n_2 as the Euclidean distance between their coordinates in the latent space:

$$d_{L2}(n_1, n_2) = \sqrt{\sum_{j=1}^{D} \left(n_1^j - n_2^j\right)^2} \tag{4}$$

where $(n_1^j)_j$ (resp. $(n_2^j)_j$) are the coordinates of neuron n_1 (resp. n_2) in the latent space of D dimensions.
It can be noted that the phenotype space produced by L2-GENE only contains positive weight values, while direct encoding can explore networks with both positive and negative weights.Reducing the search space notably blocks the evolution from introducing inhibition behaviors between features, highlighting the need for a function with a distribution centered around zero.
In order to allow for negative distances, we introduce α in Equation 5 as the identity function bounded to -1 and 1.

$$\alpha : \begin{cases} \text{if } x \geq 1 : \alpha(x) = 1 \\ \text{if } x \leq -1 : \alpha(x) = -1 \\ \text{else: } \alpha(x) = x \end{cases} \tag{5}$$

This value is used in our first proposed distance function, pL2-GENE (for product-L2), which is the Euclidean distance multiplied by the bounded product of all components of the vector from a neuron to the other in the *coordinates* space, as defined in Equation 6.

$$d_{pL2}(n_1, n_2) = \alpha(\prod_{i=1}^{D} n_1^i - n_2^i) \sqrt{\sum_{j=1}^{D} \left(n_1^j - n_2^j\right)^2} \tag{6}$$

One possible drawback of the pL2-GENE distance function is that it enforces symmetry in changes to a neuron's input and output weights. To separate the weights of incoming connections, we introduce a second distance function where the first component of the *coordinates* is only used to compute the weights of incoming connections and the other components are only used to compute the weights of outgoing connections. Thus, changing a gene in an individual only impacts one layer. This distance function defined in Equation 7 is inspired by the protein tag distances used in Artificial Gene Regulatory Networks (9); we therefore refer to it as tag-GENE. This distance uses exponential decay of the difference between one value from a node and the other values from the other node, generalised to any number of dimensions with the α function defined earlier to determine signs in front of each exponential. Finally, as exponential decay yields small values, the distance is multiplied by a factor of 50 to create a weight range closer to that of other methods.

$$d_{tag}(n_1, n_2) = \sum_{j=2}^{D} \alpha(n_1^j - n_2^1) e^{-|n_1^j - n_2^1|} \tag{7}$$

While these two distance functions are studied extensively in this article, we tried other distance functions and believe that the co-evolution of distance function and latent space position is a strong future direction for GENE.

B. Hyperparameters.

This representation relies on hyperparameters for which values are not optimized in the algorithm.

Network characteristics. The neural networks considered for the control tasks in this representation are fully-connected networks with no recurrent nor convolutional layers. Since GENE does not encode the architecture but only the values of weights and biases, the number of layers and the number of neurons in each layer (ie the network architecture) must be determined before evolution.
The activation function not being encoded in the genome, it has to be determined beforehand too. The activation function is the same for all neurons across the network, and for all individuals. Among the considered functions were cosine, Rectified Linear Units (25), sigmoid, and hyperbolic tangent.

Coordinates dimensions. The *coordinates dimension* is the number of parameters used to describe each neuron, and used as input to the distance function to compute the weight of a connection. The number of parameters in the network being directly proportional to this value, small values (from 2 to 10) were considered.
The bias of each neuron can also be concatenated as an additional dimension in the genome, or ignored in which case the output of a neuron is the weighted sum of the inputs through the activation function.

Optimization. GENE is an encoding method which can be used with any continuous black-box optimization algorithm

for search in the latent space. In this work, we compare encoding methods for Natural Evolution Strategies, specifically Separable Natural Evolutionary Strategies (*SNES*) and Exponential Natural Evolutionary Strategies (*XNES*) (38). We also considered the Covariance Matrix Adaptation Evolution Strategy (*CMA-ES*) (16) using the pycma package (15). However, XNES consistently yielded similar or superior results across all games; we therefore only present the results from SNES and XNES.

As the CMA-ES and XNES algorithms rely on operations with matrices of size $n * n$ with n being the number of genes, the computing time and memory required for each generation can be a limiting factor with direct encoding. In this work, memory limitations during experiments upper bounded the size of the networks that we considered for direct encoding; specifically we did not thoroughly explore direct encoding using XNES. This demonstrates an advantage of GENE; the possibility of using such population metrics for the evolution of large neural networks.

For both SNES and XNES, default parameters were used (see (38)). The number of total genes depends on the encoding used, which then influences the population size. Specifically, for a problem of dimension n, the standard population size of $4 + 3\log(n)$ is used. For direct encoding, this corresponds to a population of 34 and for GENE with $D = 3$, a population of 25.

4: Experiments

In this work, we study the impact of indirect encoding through GENE on a set of policy search tasks from the Atari Learning Environment (ALE) (1). This environment provides many different tasks of varying complexity and has been widely used in the reinforcement learning literature. We compare GENE and direct encoding for the search of policy networks using SNES and XNES. Our aim is to study the impact of indirect encoding for relatively large networks to determine the impact of a compressed search space.

In the ALE, game state can be represented either as a screen of pixels or by the RAM (in-game memory). While many methods which use the ALE as a benchmark use the pixel representation of the game state, in this work we use the in-game memory. This is a string of 128 8-bit integers which we convert to a floating point vector by dividing by 255. The RAM as a state representation provides many benefits, including its computational efficiency as the screen capture of the ALE is much more expensive than recovering the RAM or taking actions. The RAM state also properly defines a Markov Decision Process, whereas 4 frames of pixels are usually combined when using pixel space to recover the MDP property. The compact input space of 128 floating point values also allows us to focus on fully connected ANN architectures in this first study of GENE, as opposed to convolutional networks.

As a result of using RAM space, the results in this paper are not directly comparable to methods on Atari which use pixel input as the state representation. However, we do provide in Table 1 baseline references from methods which use pixel inputs, namely Deep-Q Networks (DQN (23)) and Implicit Quantile Networks (IQN (7)).

The games used in this work are Ice Hockey, Space Invaders, Amidar, Asteroids, Krull, and Seaquest. These games were chosen to display a variety a search requirements, with some games requiring exploration of very different policies to achieve higher reward and others benefitting from local improvements to policies.

Hyperparameters such as network architecture and activation function dimensions were chosen after a performance comparison and using standard practices: the activation function is a Rectified Linear Unit (ReLU) for all neurons, and each network used on Atari RAM has 128 inputs, 2 layers of 64 neurons, and the outputs corresponding to the possible actions. For comparison with direct encoding, GENE uses a latent space with 3 dimensions. For a game like Krull with 18 possible actions, this represents a genome of 1096 genes with GENE while direct encoding requires 13714 genes, more than 12 times the size of the GENE genome.

The GENE algorithm and its direct encoding counterpart for comparison were implemented in Julia (2) and based on the Cambrian.jl library[1]. The SNES and XNES implementations are from the EvolutionaryStrategies.jl package[2]. The code for GENE and all experiments is available online[3].

A. Comparison with Direct Encoding.

The primary study of this work is a a comparison between GENE and direct encoding, using multiple different distance functions for GENE. The fitness values of best individuals are presented in Table 1, and graphs of the evolution are presented in Figure 2. Each configuration was run for 10 independent trials with different random seeds. The fitness of the best individual found was recorded during evolution and the final measurement presented in this work is the average fitness over all trials.

We use the Mann–Whitney–Wilcoxon test (22) to test the equality of means between each GENE variant and the results from direct encoding optimized with SNES. This test relies on the ranking of the final fitness values attained by each evolution, comparing each algorithm with the direct encoding method using SNES on the same problem. This allows for comparison with the critical value being 0.5: if the metric is far from 0.5, the distributions of the results from the GENE algorithm and direct encoding can be considered different, conversely, if it is close to 0.5, they are similar. This metric and the p-value are provided in Table 1. Results from common baselines are also provided, although they are not directly comparable as these operate in pixel space inputs while all GENE and direct encoding results use RAM inputs.

We observe that GENE can reach scores competitive with direct encoding: for all games there is at least one GENE variant with the same distribution as direct encoding. This demonstrates that the search space can be drastically reduced without losing performance for neuroevolution. Furthermore,

[1] https://github.com/d9w/Cambrian.jl
[2] https://github.com/d9w/EvolutionaryStrategies.jl
[3] https://github.com/TemplierPaul/GENE.jl

Algorithm	Encoding	Ice Hockey	Space Invaders	Amidar	Asteroids	Krull	Seaquest
Average fitness							
SNES	Direct	10 (7.74)	**1162 (199.53)**	**288 (49.21)**	**9273 (1189.51)**	**10099 (2461.00)**	826 (26.75)
	tag-GENE	13 (2.77)	1007 (138.03)	244 (43.33)	4747 (486.33)	8988 (1088.12)	772 (191.42)
	pL2-GENE	**14 (2.87)**	**1162 (164.81)**	256 (36.91)	8639 (804.84)	9718 (1665.35)	796 (64.50)
XNES	tag-GENE	12 (0.94)	1004 (58.54)	227 (23.74)	4509 (489.59)	9202 (1253.67)	**946 (399.89)**
	pL2-GENE	**14 (3.10)**	1053 (170.85)	285 (43.48)	7907 (876.46)	9304 (1039.85)	860 (213.33)
Mann-Whitney-Wilcoxon test							
SNES	tag-GENE	**0.44** (**0.34**)	0.24 (0.027)	0.24 (0.033)	0.0 (9.1e-05)	**0.42** (**0.3**)	0.17 (0.0073)
	pL2-GENE	**0.43** (**0.31**)	**0.5** (**0.48**)	0.32 (**0.096**)	0.34 (**0.12**)	**0.47** (**0.43**)	0.25 (0.03)
XNES	tag-GENE	**0.5** (**0.48**)	0.23 (0.021)	0.12 (0.0031)	0.0 (9e-05)	**0.41** (**0.26**)	0.35 (**0.13**)
	pL2-GENE	0.38 (**0.21**)	0.32 (**0.087**)	**0.49** (**0.5**)	0.14 (0.0041)	**0.45** (**0.37**)	0.33 (**0.1**)
Baselines							
DQN		-1.9	1692.3	978.0	1364.5	8422.3	5860.6
IQN		0.2	28888.0	2946.0	2898.0	10707.0	30140.0
Human Average		0.9	1668.7	1719.5	47388.7	2665.5	42054.7
Random Actions		-11.2	148.0	5.8	719.1	1598.0	68.4

Table 1. Comparison of GENE with 3 dimensions and direct encoding; average values are presented with standard deviation in parentheses and the best scores are highlighted. Results from the Mann–Whitney–Wilcoxon test are presented as the normalised MWW metric and p-value in parentheses; bold values indicate similarity to direct encoding. Common baselines of a random agent and pixel-based total reward values of DQN (23), IQN (7), and an average human player are also provided.

evolution with GENE has a smaller standard deviation than direct encoding in all games but Seaquest, showing more stability over different trials.

In most case, pL2-GENE reaches higher values than tag-GENE, showing the importance of the design of the distance function in the algorithm. This raises the question of which other functions could be used to improve performance further, which is a future direction for GENE.

A surprising result is that SNES is competitive with or outperforms XNES on many games. XNES, like CMA-ES, uses population-level statistics to inform movement in the search space; intuitively, this should yield better exploration. An advantage of the reduced search space of GENE is the ability to use methods like XNES or CMA-ES, but for the current task of policy search in Atari, it does not appear necessary for most games. Rather, methods like SNES which consider all genes as independent perform well.

To understand this phenomena, we also explored random search on a reduced set of games. We found that it did not perform as well as informed methods like SNES (820 on SpaceInvaders), but only by a small margin. This is consistent with other results on Atari (**?**), suggesting that neuroevolution for Atari games may not benefit greatly from informed search and that its fitness landscape may be deceptive.

B. Impact of number of dimensions.

In order to evaluate the impact of the number of dimensions of the Euclidean space used to encode all neurons, we compare evolution on Seaquest, Amidar, and Space Invaders with dimensions 3, 5, and 10, in Table 2. With $D = 10$, GENE genomes are still 4.5 times smaller than those required by direct encoding; the genome for this network in GENE would require a 50-dimensional latent space to reach the size of the genome in direct encoding.

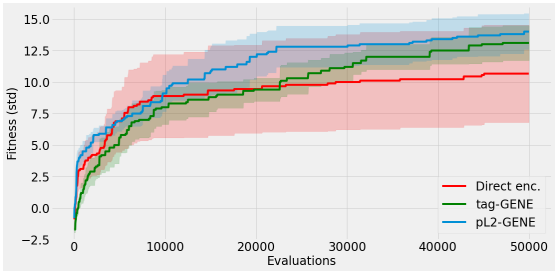
Using a Euclidean space with 5 dimension seems to help reach higher scores for Amidar and Seaquest thanks to a larger search space allowing for more potential solutions to be explored. However, increasing the number of dimensions to 10 yields lower scores, which could be explained by a search space becoming too large, hence needing more time to find high-performing individuals. The dimensionality of GENE therefore appears to be an important hyperparameter which allows balancing between a smaller search space at low dimensions and more complex networks with a larger number of parameters at higher dimensions.

D	Encoding	Space Invaders	Amidar	Seaquest
0	Direct	**1162 (199.53)**	288 (49.21)	826 (26.75)
3	tag-GENE	1007 (138.03)	244 (43.33)	772 (191.42)
	pL2-GENE	**1162 (164.81)**	256 (36.91)	796 (64.50)
5	tag-GENE	1082 (33.58)	254 (33.30)	828 (276.72)
	pL2-GENE	1023 (115.53)	**292 (27.70)**	**940 (379.05)**
10	tag-GENE	1037 (63.15)	232 (22.33)	711 (33.33)
	pL2-GENE	1095 (99.06)	221 (21.94)	748 (225.41)

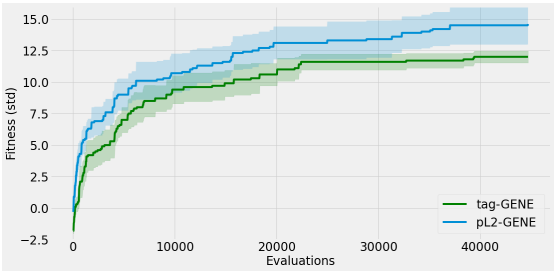
Table 2. Comparison of different number of dimensions D; mean final value and standard deviation are presented.

C. Computational cost.

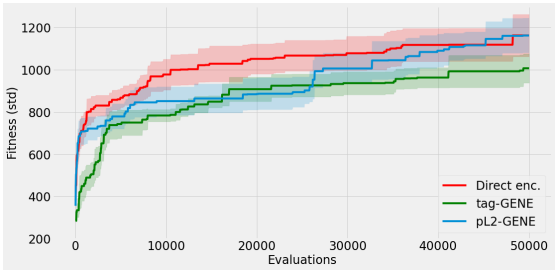
One goal of this indirect encoding is to reduce the computational cost of training neural networks with evolution strategies. Since the total evolution time depends heavily on variable evaluation time which is influenced by performance (fin-



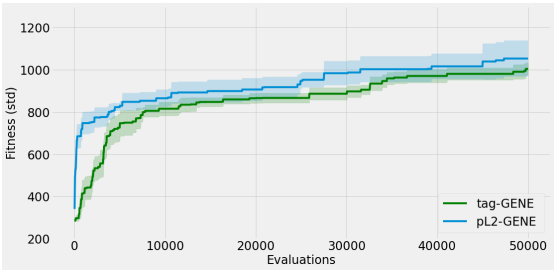
(a) SNES on IceHockey



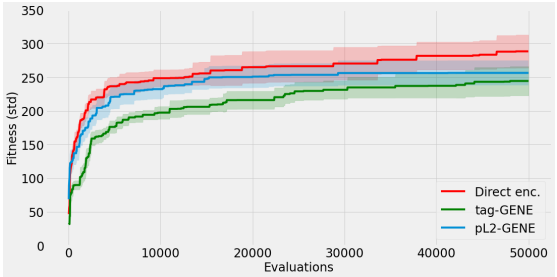
(b) XNES on IceHockey



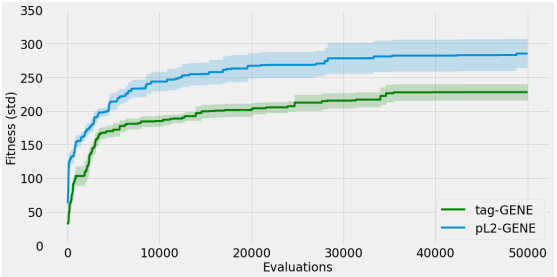
(c) SNES on Space Invaders



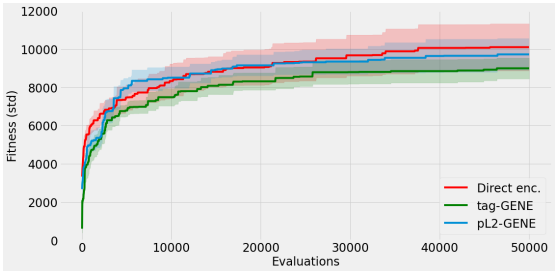
(d) XNES on Space Invaders



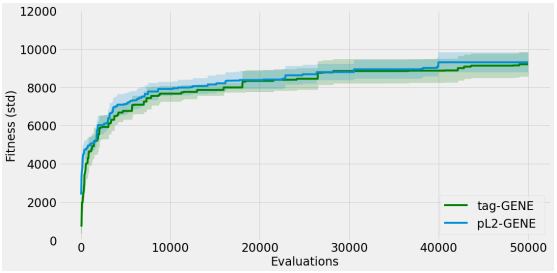
(e) SNES on Amidar



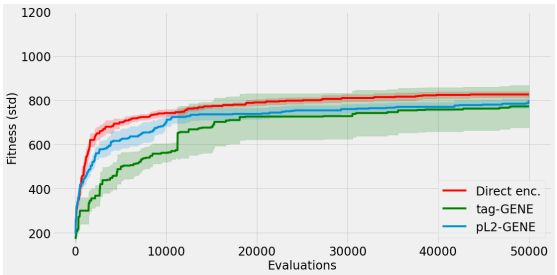
(f) XNES on Amidar



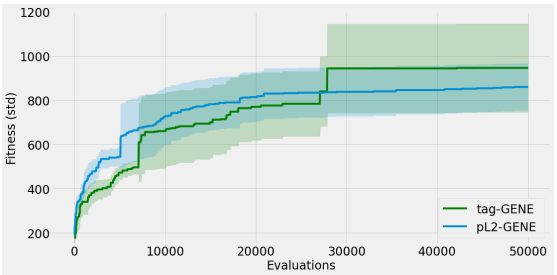
(g) SNES on Krull



(h) XNES on Krull



(i) SNES on Seaquest



(j) XNES on Seaquest

Fig. 2. Evolution on 5 of the tested Atari games using the 128-64-64-10 architecture and $D = 3$. Means over 10 trials are represented as lines and one standard deviation as ribbons.

ishing tasks faster or surviving longer bringing higher rewards depending on the game), we compare the two other steps in the evolution: *build*, the creation of individuals, and *update*, the population step. The comparison is done on smaller networks than used in the results to allow for the use of XNES with direct encoding, bigger networks being

intractable with XNES and direct encoding. The BenchmarkTools.jl package is used to measure time and memory usage, with default parameters. Each configuration is run multiple times, from 10000 times for fast steps to only 1 for extremely long ones such as XNES updates on direct encoding.

The *build* step takes a raw genome and creates the corresponding network, either by allocating weights to the right connection in direct encoding or by computing all weights from *coordinates* in GENE. Since computing distances takes longer, build time is naturally higher for GENE; however, it can be easily computed for all individuals in parallel as part of the evaluation.

The *update* step determines the parameters of the distribution which creates the next population from the population of evaluated individuals. It mostly depends on the optimization method and on the genome size since algorithms such as XNES use large matrices, and happens once each generation. For the *update* step, GENE is twice to four times as fast as direct encoding for SNES, which does not perform any population-level matrix calculation. Combined with the longer *build* time, though, the total computational cost difference between GENE and direct encoding is minor for SNES.

Encoding	D	Mean time (s)	Memory (KiB)
pL2-GENE	3	0.001380	2897.92
pL2-GENE	10	0.001692	3676.16
Direct	-	0.000022	46.53

Table 3. Average time and memory for the build step of 1 individual, for a 128-32-32-9 network. All GENE functions have similar build time, so only pL2-GENE is used.

Encoding	D		Mean time (s)	Memory (KiB)
pL2-GENE	3	SNES	0.000357	630
pL2-GENE	10	SNES	0.000678	1372
Direct	-	SNES	0.001350	3133
pL2-GENE	3	XNES	1.475	1352663
pL2-GENE	10	XNES	14.244	11806965
Direct	-	XNES	119.976	79765176

Table 4. Average time and memory for the update step for a 128-32-32-9 network

However, the update in XNES becomes extremely long and costly in memory when applied to the large genomes of direct encoding, and GENE provides a speedup of up to 80 times faster with a latent space of 3 dimensions. In this case, the additional cost of building the networks is insignificant compared to the cost of each evolution step. Similar behaviors was observed with CMA-ES, which made the use of XNES and CMA-ESprohibitively expensive in terms of computational time and memory for direct encoding evolution.

GENE reduces complexity in the *update* step, especially in methods that rely on large matrix operations that scale quadratically with the number of parameters to optimize. This property can be seen as a simple speedup in evolution time, but it also allows for using these more complex optimization methods on larger networks when they are intractable with direct encoding.

5: Discussion and Conclusions

This work presents a first study into a new neuroevolution encoding method. While we demonstrate some advantages of searching in a compressed representation through experiments on policy search, we believe that GENE could be used in other NE domains and algorithms. We identify future directions for this work and reflections on NE for policy search.

A. Policy search in a latent space.

Changes in weight space (or direct encoding space) cause changes in policy space. Although correlated, the amplitude of the latter is difficult to control with the former. For instance, many very different networks actually correspond to the same function (because of symmetries, output summations, etc.); looking for diversity in weight space does not necessarily provide diversity in policy space. Our rationale is that a smaller (indirect) encoding space facilitates the diversity of policies, rather than the diversity of weights. For environments that require vast exploration, GENE is more likely to generate a policy that, in turn, is likely to explore the relevant parts of the state-action space.

Quality diversity methods such as MAP-Elites search explicitly for diversity in policy space (6). One future direction for GENE is to study how policy diversity is changed by using an indirect encoding when compared with explicit search for diversity.

B. Future work.

To improve GENE, we aim to study a larger family of distance functions and also their optimization. For example, HyperNEAT evolves a distance function in the form of a CPPN (33). Co-evolution of the function and the GENE genome could be used and is the primary future direction for this work.

A second approach for developing GENE is to mix indirect and direct encoding in the same evolution, as in (5). This would add a switch mechanism during the evolution, starting with indirect encoding to boost exploration in a smaller search space, and switching to direct encoding to refine solutions found in the step. The weights provided by GENE could, for example, be used as the starting point for backpropagation-based fine-tuning of weights.

Some distance functions being differentiable, the use of backpropagation and gradient descent to optimize GENE genomes could be also studied. One possible problem is the α function in pL2-GENE and tag-GENE, along with symmetry issues in the latent space. This also potentially requires the determination of the inverse transform from a network to a GENE genome.

Progressively increasing the number of dimensions during the evolution would also be a way to gradually grow the search space when GENE stagnates, while still keeping

smaller genomes than direct encoding. The proposed distance functions are scalable to the addition of a dimension (if initialised with the new coordinate as 0), allowing for growth in network complexity over evolution.

Finally, the encoding of fully-connected layers in GENE raises the question of the encoding of other neural network types like convolutional and recurrent networks. We believe that with the proposal of GENE, we have demonstrated that a simple indirect encoding scheme of neural network weights can offer many possibilities and new directions for neuroevolution.

Acknowledgments

This work was performed using HPC resources from CALMIP (Grant P21001).

Bibliography

1. Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. 2013. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research* 47 (2013), 253–279.
2. Jeff Bezanson, Alan Edelman, Stefan Karpinski, and Viral B. Shah. 2017. Julia: A Fresh Approach to Numerical Computing. *SIAM Rev.* 59, 1 (2017), 65–98. https://doi.org/10.1137/141000671 _eprint: https://doi.org/10.1137/141000671.
3. Andrew Brock, Theodore Lim, J. M. Ritchie, and Nick Weston. 2017. SMASH: One-Shot Model Architecture Search through HyperNetworks. *arXiv:1708.05344 [cs]* (Aug. 2017). http://arxiv.org/abs/1708.05344 arXiv: 1708.05344.
4. Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. 2018. Back to basics: benchmarking canonical evolution strategies for playing Atari. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*. 1419–1426.
5. J. Clune, K. O. Stanley, R. T. Pennock, and C. Ofria. 2011. On the Performance of Indirect Encoding Across the Continuum of Regularity. *IEEE Transactions on Evolutionary Computation* 15, 3 (June 2011), 346–367. https://doi.org/10.1109/TEVC.2010.2104157 Conference Name: IEEE Transactions on Evolutionary Computation.
6. Antoine Cully, Jeff Clune, Danesh Tarapore, and Jean-Baptiste Mouret. 2015. Robots that can adapt like animals. *Nature* 521, 7553 (2015), 503–507.
7. Will Dabney, Georg Ostrovski, David Silver, and Rémi Munos. 2018. Implicit quantile networks for distributional reinforcement learning. In *International conference on machine learning*. PMLR, 1096–1105.
8. David B. D'Ambrosio and K. Stanley. 2008. Generative encoding for multiagent learning. In *GECCO '08*. https://doi.org/10.1145/1389095.1389256
9. Jean Disset, Dennis G Wilson, Sylvain Cussat-Blanc, Stéphane Sanchez, Hervé Luga, and Yves Duthen. 2017. A Comparison of Genetic Regulatory Network Dynamics and Encoding. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO '17)*. Association for Computing Machinery, New York, NY, USA, 91–98. https://doi.org/10.1145/3071178.3071322
10. David B D'Ambrosio, Jason Gauci, and Kenneth O Stanley. 2014. HyperNEAT: The first five years. *Growing adaptive machines* (2014), 159–185.
11. Chrisantha Fernando, Dylan Banarse, Malcolm Reynolds, Frederic Besse, David Pfau, Max Jaderberg, Marc Lanctot, and Daan Wierstra. 2016. Convolution by Evolution: Differentiable Pattern Producing Networks. *arXiv:1606.02580 [cs]* (June 2016). http://arxiv.org/abs/1606.02580 arXiv: 1606.02580.
12. Faustino Gomez, Jürgen Schmidhuber, Risto Miikkulainen, and Melanie Mitchell. 2008. Accelerated Neural Evolution through Cooperatively Coevolved Synapses. *Journal of Machine Learning Research* 9, 5 (2008).
13. David Ha. 2017. Evolving Stable Strategies. *blog.otoro.net* (2017). http://blog.otoro.net/2017/11/12/evolving-stable-strategies/
14. David Ha, Andrew M. Dai, and Quoc V. Le. 2016. HyperNetworks. (Oct. 2016). https://openreview.net/forum?id=rkpACe1lx
15. Nikolaus Hansen, Youhei Akimoto, and Petr Baudis. 2019. *CMA-ES/pycma on Github*. https://doi.org/10.5281/zenodo.2559634 Published: Zenodo, DOI:10.5281/zenodo.2559634.
16. Nikolaus Hansen and Andreas Ostermeier. 2001. Completely Derandomized Self-Adaptation in Evolution Strategies. *Evolutionary Computation* 9, 2 (June 2001), 159–195. https://doi.org/10.1162/106365601750190398
17. Matthew Hausknecht, Joel Lehman, Risto Miikkulainen, and Peter Stone. 2014. A neuroevolution approach to general atari game playing. *IEEE Transactions on Computational Intelligence and AI in Games* 6, 4 (2014), 355–366.
18. Maryam Mahsal Khan, Arbab Masood Ahmad, Gul Muhammad Khan, and Julian F Miller. 2013. Fast learning neural networks using cartesian genetic programming. *Neurocomputing* 121 (2013), 274–289.
19. Hiroaki Kitano. 1990. Designing neural networks using genetic algorithms with graph generation system. *Complex systems* 4 (1990), 461–476.
20. Jan Koutnik, Faustino Gomez, and Jürgen Schmidhuber. 2010. Evolving neural networks in compressed weight space. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*. 619–626.
21. Yuqiao Liu, Yanan Sun, Bing Xue, Mengjie Zhang, and Gary Yen. 2020. A Survey on Evolutionary Neural Architecture Search. *arXiv:2008.10937 [cs]* (Aug. 2020). http://arxiv.org/abs/2008.10937 arXiv: 2008.10937.
22. H. B. Mann and D. R. Whitney. 1947. On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. *The Annals of Mathematical Statistics* 18, 1 (1947), 50 – 60. https://doi.org/10.1214/aoms/1177730491
23. Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, and others. 2015. Human-level control through deep reinforcement learning. *nature* 518, 7540 (2015), 529–533. Publisher: Nature Publishing Group.
24. David J Montana, Lawrence Davis, and Mouiton St. 1989. Training Feedforward Neural Networks Using Genetic Algorithms. (Aug. 1989), 6.
25. Vinod Nair and Geoffrey E Hinton. [n.d.]. Rectified Linear Units Improve Restricted Boltzmann Machines. ([n. d.]), 8.
26. Martin L Puterman. 1994. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.
27. Ingo Rechenberg. 1989. Evolution Strategy: Nature's Way of Optimization. In *Optimization: Methods and Applications, Possibilities and Limitations*, H. W. Bergmann (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 106–126.
28. Sebastian Risi, Joel Lehman, and Kenneth O Stanley. 2010. Evolving the placement and density of neurons in the hyperneat substrate. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*. 563–570.
29. Sebastian Risi and Kenneth O Stanley. 2011. Enhancing es-hyperneat to evolve more complex regular neural networks. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*. 1539–1546.
30. Herbert Robbins and Sutton Monro. 1951. A Stochastic Approximation Method. *Annals of Mathematical Statistics* 22, 3 (1951), 400–407. https://doi.org/10.1214/aoms/1177729586 Publisher: The Institute of Mathematical Statistics.
31. Edmund Ronald and Marc Schoenauer. 1994. Genetic lander: An experiment in accurate neuro-genetic control. In *Parallel Problem Solving from Nature — PPSN III*. Vol. 866. Springer Berlin Heidelberg, Berlin, Heidelberg, 452–461. https://doi.org/10.1007/3-540-58484-6_288 Series Title: Lecture Notes in Computer Science.
32. Kenneth O. Stanley. 2007. Compositional pattern producing networks: A novel abstraction of development. (2007).
33. Kenneth O Stanley, David D'Ambrosio, and Jason Gauci. 2009. A Hypercube-Based Indirect Encoding for Evolving Large-Scale Neural Networks. (2009), 39.
34. Kenneth O Stanley and Risto Miikkulainen. 2002. Evolving neural networks through augmenting topologies. *Evolutionary computation* 10, 2 (2002), 99–127. Publisher: MIT Press.
35. Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. 2018. Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning. arXiv:cs.NE/1712.06567
36. Sjoerd van Steenkiste, Jan Koutník, Kurt Driessens, and Schmidhuber, Jurgen. 2016. A Wavelet-based Encoding for Neuroevolution. (2016), 8.
37. Darrell Whitley, Frédéric Gruau, and Larry Pyeatt. 1970. Cellular Encoding Applied to Neurocontrol. (Feb. 1970).
38. Daan Wierstra, Tom Schaul, Jan Peters, and Juergen Schmidhuber. 2008. Natural Evolution Strategies. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*. IEEE, Hong Kong, China, 3381–3387. https://doi.org/10.1109/CEC.2008.4631255