Cooperative Coevolutionary Spatial Topologies for Autoencoder Training

Cooperative Coevolutionary Spatial Topologies for Autoencoder Training

Erik Hemberg[**]
hembergerik@csail.mit.edu
University of Malaga, Spain

Jamal Toutouh
jamal@uma.es
University of Malaga, Spain

Una-May O'Reilly
unamay@csail.mit.edu
MIT, USA

ABSTRACT

Training autoencoders is non-trivial. Convergence to the identity function or overfitting are common pitfalls. Population based algorithms like coevolutionary algorithms can provide diversity. To more robustly train autoencoders, we introduce a novel cooperative coevolutionary algorithm that exploits a spatial topology. We investigate the impact of algorithm parameters and design choices on the performance. On a simple tunable benchmark problem we observe that the performance can be improved over that of an conventionally trained autoencoder. However, the training convergence can be slow, despite the final model performance being competitive with a conventional autoencoder.

CCS CONCEPTS

• **Theory of computation → Evolutionary algorithms**.

KEYWORDS

cooperation, autoencoder, evolutionary algorithms

ACM Reference Format:
Erik Hemberg, Jamal Toutouh, and Una-May O'Reilly. 2024. Cooperative Coevolutionary Spatial Topologies for Autoencoder Training. In *Genetic and Evolutionary Computation Conference (GECCO '24), July 14–18, 2024, Melbourne, VIC, Australia.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3638529.3654127

1 INTRODUCTION

An autoencoder is a two-part artificial neural network (ANN) that learns an efficient lower dimensional representation (embedding) of data [11]. It consists of: an encoder that transforms the (higher dimensional) input data to a (lower dimensional) latent representation, and a decoder that translates the latent representation back to the input data representation. Auto-encoding requires coordination between the encoder and decoder in terms of representation dimensionality reduction and reconstruction.

Training autoencoders is non-trivial. Training can result in collapse to the identity function or in overfitting [11]. To improve training robustness, variants that integrate regularization have been introduced [3]. In this contribution we introduce and evaluate an alternate approach, that of cooperative coevolution. Like prior work that *competitively* coevolves high performing Generative Adversarial Networks (GANS) [16], this approach also potentially offers training robustness [20]. Working with two distinct populations - encoders and decoders, the encoders and decoders can be paired with one another in multiple combinations. Some of the resulting autoencoders may perform better than others, implying that some encoders may, after gradient-based training, on average (or best case) perform better across the decoders they are paired with. The same applies to decoders. When the coevolutionary algorithm selects within each population for superior performance, it is evolving toward encoders and decoders that work well with others. The evolutionary selection pressure and reproduction guides population exploitation, while variation offers exploration. Additionally, population diversity can overcome overfitting and sensitivity to initialization that may lead to collapse to the identity function. This approach can be seen as *cooperative* due to the mutualism between an encoder and decoder which are trained as one with gradient-based parameter optimization and the autoencoder's loss function. There is no prior work on cooperative coevolutionary algorithm training separate populations of encoders and decoders.

Exploiting and extending the value of a coevolutionary algorithm, we introduce a novel cooperative coevolutionary algorithm (CooCA), named Lipi-AE, that uses a spatial topology to train autoencoders, see Figure 1. The spatial topology partitions the populations into smaller, overlapping sub-populations. This reduces the quadratic complexity to linear. It also potentially leverages the implicit communication between overlapping neighborhoods and contributes additional diversity, key to robustness. Prior work has shown the value of spatially distributing coevolutionary algorithms, e.g. Lipizzaner [16]. Lipizzaner is similar to CooCA; the competitive formulation of GAN training in Lipizzaner is replaced with the cooperative formulation of AE training in Lipi-AE.

Given a general goal of learning efficient embeddings of unlabeled data by using autoencoders, our research questions are:

RQ-1 What is the performance impact of replacing the competitive formulation of GAN training with the cooperative formulation of AE training in a spatial coevolutionary ANN training framework?

RQ-2 How sensitive is CooCA to population size, problem difficulty, the solution concept used, and topology?

RQ-3 Diversity implies a more robust likelihood of one combination being high performing. Are the evolved populations of encoders or decoders diverse?

Our contributions are:

• Lipi-AE A novel cooperative coevolutionary method for training of AEs using a spatial topology. This contrasts with `Lipizzaner` because it integrates a cooperative loss-function for the two modules of the ANN, instead of the competitive (minimax) for the discriminator and generator of a GAN.

[*]Also with MIT, USA.

GECCO '24, July 14–18, 2024, Melbourne, VIC, Australia

ACM ISBN 979-8-4007-0494-9/24/07.
https://doi.org/10.1145/3638529.3654127

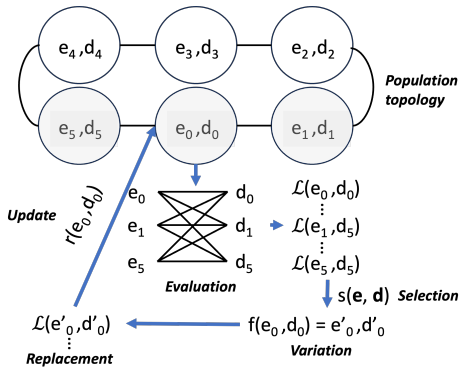


Figure 1: Overview Lipi-AE. A spatial ring topology of size 6 with radius one is shown. The neighbors are copied to form a subpopulation. The Euclidean product of AEs are evaluated. For each batch high quality encoders and decoders are selected. Then encoders and decoders are varied through training and replaced. Finally, the Euclidean product of AEs are evaluated and the encoder and decoder at the node are updated by the new best encoder and decoder.

- A comparative analysis of Lipi-AE versus canonical AE performance. On simple tunable benchmark problems we observe that the reconstruction error can be improved. However, the convergence rate can be slow, even though the final model reconstruction error is competitive.
- An analysis of evolved solution diversity shows that Lipi-AE does not converge to the identity. Diversity is influenced by population size, topology radius, and what solution concept is used.
- We find that the neighborhood size, i.e. topology, and solution concept of Lipi-AE are two components key to its performance. Neighborhood size impacts signal propagation, and a radius of 3 is ideal for the binary clustering problem we study. We performed a parametric sweep of population size and problem difficulty to compare Lipi-AE performance. We also compared performance to a Variational AE, see Figure 3.

The paper proceeds as follows. In Section 2 we present background. In Section 3 we present related work. In Section 4 we present our methodology. In Section 5 we present experiments. In Section 6 we present conclusions and future work.

2 BACKGROUND

We use the following notation. An ANN is a parameterized function $f_\theta : \mathcal{X} \rightarrow \mathcal{Y}, y = f_\theta(x)$. The ANN parameters θ are update iteratively $\theta_t = \gamma \mathcal{L}(y, f_\theta(x)) + \theta_{t-1}, \gamma \in [0, 1], \theta \in \mathbb{R}^n$, and $\theta_0 \sim \mathcal{U}$ is uniform random. The loss function $\mathcal{L} : \mathcal{Y} \times \mathcal{X} \rightarrow \mathbb{R}, \mathcal{L}(y, f_\theta(x))$ determines the magnitude of the parameter update. The update interval is determined by the cardinality of the data $|x| = |X|/c, 0 \leq c \leq |X|$, c is the batch size. The objective of ANN training is to minimize some loss, $\arg\min_{\theta \in \Theta} \mathcal{L}(y, f_\theta(x|D), D = [(x_0, y_0), \dots]$

In this section we present background. In Section 2.1 we present autoencoders. In Section 2.2 we present coevolutionary algorithms.

2.1 Autoencoders

An autoencoder has the input domain $\mathbf{x} \in \mathcal{X}$ and the embedding $\mathbf{z} \in \mathbb{R}^n$[3]. The parameterized encoder function $e_\theta : \mathcal{X} \rightarrow \mathbb{R}^n, z = f_\theta(x)$ transforms (encodes) the input domain to a latent space. The parameterized decoder function $d_\theta : \mathbb{R}^n \rightarrow \mathcal{X}, x' = d_\theta(z)$. We can write $x' = g_\theta(f_\theta(x))$ as the autoencoder. The basic AE reconstructive loss is $\mathcal{L}_{ae} = |x - x'|$. The objective of AE training is to minimize the reconstructive loss, $\arg\min_{\theta \in \Theta} \mathcal{L}_{ae}(x, d_\theta(e_\theta(x|D))$. The autoencoder loss can be decomposed to $\mathcal{L}_{ae}(x, d_\theta(e_\theta(x|D)) = \alpha \mathcal{L}_d(x, d_\theta(e_\theta(x|D)) + \beta \mathcal{L}_e((e_\theta(x|D), \beta = 1 - \alpha, \alpha \in [0, 1]$.

The parameterization can have separate parameters for the decoder ϕ. We can write $x' = g_\phi(f_\theta(x))$ as the autoencoder. The objective of AE training is to minimize the reconstructive loss, $\arg\min_{\theta \in \Theta, \phi \in \Phi} \mathcal{L}(x, d_\phi(e_\theta(x|D))$. With the updates

$$\theta_t = \gamma \mathcal{L}_{ae}(x, d_\phi(f_\theta(x|D))) + \theta_{t-1}$$

and

$$\phi_t = \gamma \mathcal{L}_{ae}(x, d_\phi(f_\theta(x|D)) + \phi_{t-1}$$

.

2.2 Coevolutionary Algorithms

Biological coevolution refers to the influences two or more species exert on each other's evolution [26]. A seminal paper on reciprocal relationships between insects and plants coined "coevolution" [8]. Coevolution can be cooperative, i.e., mutual benefit, or competitive, i.e., interactions arising from contested resources. An EA typically evolves individual solutions, e.g., fixed length bit strings as in Genetic Algorithms [10] using an *a-priori* defined fitness function to evaluate an individual's quality. In contrast, coevolutionary algorithms (CA) calculate an individual's fitness based on its interactions with other individuals or a dynamic environment allowing them to mimic coupled natural species-to-species interactions. This contribution extends the prior work studying coevolutionary algorithms [1, 15, 18, 21–23, 26, 29].

The dynamics of CAs are difficult to analyze because their fitness evaluation is derived *interactively*, [23]. Effectively an individual's fitness is a sample-based estimate of their performance where the samples are drawn from the other population which itself is evolving. A more formal approach, using the *solution* and *test* perspective, describes fitness assignments as *solution concepts* [23]. Solution concepts include: best worst case, and maximization of expected utility.

Spatial Topologies for Coevolutionary Algorithms. `Lipizzaner` is used for GAN training and promotes diversity with a population, topology, operators, data and objectives[16, 31]. The key methods in `Lipizzaner` are: (1) a topology with overlapping cells between neighborhoods, with sub-populations of generators and discriminators in the neighborhoods; (2) gradient-based learning to update the ANN parameters of the generator, discriminator; (3) Gaussian-based mutations to update the learning rate and weight ensemble parameters; and (4) evolutionary selection and replacement to maintain convergence.

Figure 1 shows an example of a 1D ring. In the ring topology, the cells are distributed in a one-dimensional grid of size $1 \times Z$ and neighbors are sideways, e.g. left and/or right, i.e. an index position or more away. The best ANN pair (*center*) after an evolutionary epoch at a cell is updated. Neighborhood cells retrieve this updated ANN pair when they refresh their sub-population at the end of their training epochs, carrying signals in two directions around the ring. Figure 1 shows populations of six individuals (N=6) organized in a ring topology with neighborhood radius one (r=1). The shaded

areas illustrate the overlapping neighborhoods of the cells (0) and (4), with dotted and solid outlines, respectively. The updates in the *center* of (0) will be propagated to the (5) and (1). Note that for a 1D ring topology, where $r > 0$ the neighborhoods size is

$$s = 2r + 1 \tag{1}$$

3 RELATED WORK

There are a number of surveys on evolutionary deep learning, e.g. [20], in its taxonomy this work falls in the model generation category. We review efforts related to evolutionary autoencoders.Table 1 introduces work related to the evolutionary computation method, autoencoder setup, training topology.

Table 1: Overview of work at intersection of Evolutionary Computation and Autoencoders. EC is the type of evolutionary computation. AE is the autoencoder setup, sh. means E and D share weights. Topo. is if there is a training topology.

Ath.	Title	EC	AE	Topo.
Evolutionary Computation Representation				
[16]	Spatial coevolution for generative adversarial network training	CCA	No	Yes
[25]	Evolving autoencoding structures through genetic programming	GP	GP as E, D	No
[28]	A Genetic Programming Encoder for Increasing Autoencoder Interpretability	GP	GP as E	No
[27]	Image feature learning with a genetic programming autoencoder	GP	GP as E,D	No
[13]	An Evolutionary Approach to Variational Autoencoders	ES	E, D sh.	No
[24]	Autoencoding with a classifier system	LCS	ANN as LCS	No
Neural Architecture Search				
[5]	EvoAAA: An evolutionary methodology for automated neural autoencoder architecture search	EA	E, D sh.	No
[19]	EvoAE–a new evolutionary method for training autoencoders for deep learning networks	EA	E, D sh.	No
[14]	Evolving deep autoencoders	EA	E, D sh.	Yes
[2]	Automatic evolution of autoencoders for compressed representations	EA	E, D sh.	No
[6]	Evolving deep convolutional variational autoencoders for image classification	EA	E, D sh.	No
[12]	Evolving multi-view autoencoders for text classification	EA	E, D Sh.	No
[30]	Exploiting the potential of standard convolutional autoencoders for image restoration by evolutionary search	EA	E, D Sh.	No
[9]	Evolutionary multi-objective design of autoencoders for compact representation of histopathology whole slide images	MOO	E, D Sh.	No

Evolutionary Computation Autoencoder Representation. Some work use EC to represent encoders and/or decoders. E.g. to evolve autoencoding structures through Genetic Programming (GP) [25]. The learning of image features was also explored with a GP autoencoder [27]. In addition, a GP encoder was investigated in order to increase the autoencoder interpretability [28]. Learning classifier systems used ANNs a components for autoencoding [24]. Finally, evolutionary strategies has been used to train variational autoencoders, in this work the common architecture of an encoder and decoder with shared weights was used [13].

Neural Architecture Search (NAS). There have been a number of studies regarding using evolutionary computation for NAS, i.e. finding ANN architectures. When NAS has been used for AEs it has mainly searched for Autoencoder architectures where the Encoder and Decoder share weights. For example, evolutionary computation methods have been used for training autoencoders [2, 5, 14, 19].

The application domains have been image classification [6] and image restoration by evolutionary search [30]. In addition, multi-objective design of histopathology images have been explored [9]. Furthermore, there have been studies on evolving autoencoders for text classification [12]. Finally, the speed of the evolutionary training of AEs is improved by using a distributed environment [14].

Arguably a subset of NAS is to use EC only during the ANN training. One example is the spatial coevolution for generative adversarial network weights in the `Lipizzaner` framework which can improve both speed and performance of GAN training [16]. Our work is closest to `Lipizzaner`, with the major difference is in the use of AE (minimization) instead of GAN (minimax). The gap that Lipi-AE fills is that it uses a novel cooperative coevolution to train autoencoders with a spatially distributed topology.

4 METHOD

We present a cooperative coevolutionary framework for training autoencoders, Lipi-AE. At the highest level Lipi-AE searches for and returns an autoencoder $(\mathbf{e}, \mathbf{d})$ encoder-decoder from the best node, see Figure 1. Lipi-AE starts with Alg. 1 which runs in parallel on each cell n. Each cell neighborhood $\mathbf{n}$ gets a dataset Θ_D and initializes a single decoder d and encoder e ANNs (we refer to these as models when decoders and encoders are interchangeable). Thus there are two global populations of the same size as the topology N, $[e_1, \ldots, e_N]$ a population of encoders and $[d_1, \ldots, d_N]$ a population of decoders. These populations are a source of diversity in Lipi-AE, occurring in the representation space, genome space. Each cell n also initializes its hyper-parameter – learning rate δ.

Algorithm 1: `Lipi-AE`: In parallel, for each cell, initialize settings then iterate over each generation t. In each generation t, collect neighbors for the encoder $\mathbf{e}_k$ and decoder $\mathbf{d}_k$ sub-populations, evolve encoders $\mathbf{e}_k$ and decoders $\mathbf{d}_k$, and replace self $n_{k,1}$ with best.

Input : T : Total generations, $\mathbf{N}$: Population on topology cells, s : Neighborhood size, θ_{COEV} : Parameters for `CoevolveAndTrain`, θ_D : Training dataset
Return: $(\mathbf{e}, \mathbf{d})$: Autoencoder, an encoder-decoder pair
`//Asynchronous parallel execution of all cells in topology`
for $k \in [1, \ldots, |\mathbf{N}|]$ **do**
  $\mathbf{n}_k, w, f_w \leftarrow$ initializeNeighborhood(θ_D, $\mathbf{s}$) `//Uniformly random initialization`
  `//Iterate over generations`
  for $t \in [0, \ldots, T]$ **do**
    $\mathbf{n}'_k \leftarrow$ copyNeighbors($k, \theta_D, \mathbf{N}$) `//collect neighbor cells individuals for the sub-populations`
    $\mathbf{n}'_k \leftarrow$ CoevolveAndTrainModels($\mathbf{n}'_k, \theta_{COEV}$) `//Coevolve ANNs using Alg. 2`
    $\mathbf{n}_k \leftarrow$ updateNeighborhood($\mathbf{n}'_k, k, \mathbf{N}$)`//Update neighborhood (population), i.e.` Y
return $(\mathbf{e}^*, \mathbf{d}^*)$ `//Best autoencoder`

Alg. 1 directs each cell to asynchronously and in parallel to iterate over generations (epochs). In each generation t, a cell, n_k, starts by copying the latest versions of its neighbors $\mathbf{n}_k$ to set up its sub-populations $\mathbf{e}_k$ and $\mathbf{d}_k$. Next, each cell independently executes Alg. 2. This algorithm terminates with the cell neighborhood updating its own models, i.e. $\mathbf{e}_{k,1}$, $\mathbf{d}_{k,1}$. The two key steps for information propagation are: the update at the start of a generation t when a cell n forms a sub-population $\mathbf{n}$ by copying models from its neighbors. The replacement of the best models in the cell after application of coevolution within the sub-population of the cell.

After all generations or a computational deadline is reached Alg. 1 selects the best performing autoencoder $(\mathbf{e}^*, \mathbf{d}^*)$ across the entire topology and returns them as Lipi-AE's solution.

Algorithm 2: `CoevolveAndTrain`: Select a new sub-population from the current. Each mini-batch trains decoders $\mathbf{d}$ with a randomly drawn encoder and encoders $\mathbf{e}$ with a randomly drawn decoder, using SGD. Evaluate all against each other, using the loss value to choose how to replace center $n_{k,1}$. Return this new cell neighborhood $\mathbf{n}$.

Input : τ : Tournament size, B : Input training dataset batches, β : Mutation probability, $\mathbf{n}$: Cell neighborhood sub-population
Return : $\mathbf{n}$: Cell neighborhood sub-population

1 $\mathbf{n}_\delta \leftarrow$ mutateLearningRate($\mathbf{n}_\delta, \beta$) `//Update neighborhood learning rate with with Gaussian mutation`
2 $\phi \leftarrow$ EvaluateAEs($B, \mathbf{n}$) `//Evaluate all updated ANN pairs, Alg. 3`
3 $\mathbf{e}, \mathbf{d} \leftarrow$ tournamentSelect($\mathbf{n}, \phi, \tau$) `//Select using loss(`$\mathcal{L}$`) as fitness`
4 **for** $B \in \mathbf{B}$ **do**
  `//Loop over batches`
5 $d \leftarrow$ getRandomOpponent($\mathbf{d}$) `//Get uniform random decoder`
6 **for** $e \in \mathbf{e}$ **do**
    `//Evaluate encoders and train with SGD`
7 $\nabla_e \leftarrow$ computeGradient(e, d, B) `//Compute gradient`
8 $e \leftarrow$ updateNN(e, ∇_e, B) `//Update with gradient`
9 $e \leftarrow$ getRandomOpponent($\mathbf{e}$) `//Get uniform random encoder`
10 **for** $d \in \mathbf{d}$ **do**
    `//Evaluate decoder and train with SGD`
11 $\nabla_d \leftarrow$ computeGradient($d, e,$) `//Compute gradient`
12 $d \leftarrow$ updateNN(d, ∇_d, B) `//Update with gradient`
13 $\phi \leftarrow$ EvaluateAEs($\mathbf{B}, \mathbf{n}$) `//Evaluate all updated ANNs, Alg. 3`
14 $\mathbf{n} \leftarrow$ replaceCenterIndividuals($\mathbf{n}, \phi$) `//Best encoder and decoder are placed in the center based on loss`
15 **return** $\mathbf{n}$

Selection promotes fitter models over less fit ones when updating a sub-population $\mathbf{n}$. Lipi-AE uses tournament selection, Alg. 2 line 3. First, the latest neighbors' models are copied to form the sub-populations and then selection is applied. After all ANN training is completed and all models are reevaluated, the encoder and decoder with the best training loss $\mathcal{L}$ replace the worst in the sub-populations, Alg. 2 line 14.

The models' performance $\mathcal{L}$ in sub-populations depends on its collaborator, see Alg. 3. The fitness ϕ of a model ($e_i \in \mathbf{e}$ or $d_j \in \mathbf{d}$) is based on the best according to a solution concept, $f_s \in \{f_s^{max}, f_s^{min}, f_s^{mean}\}$, i.e., f_s^{max} is "best case", f_s^{min} is "worst case", and f_s^{mean} is "mean case" $\mathcal{L}$ with all its collaborators.

In Lipi-AE, ANN training is a black box component which applies stochastic gradient descent (SGD) to update the parameters of the models, Alg. 2 lines 7, 8, 11, and 12. The main differences between Lipi-AE and `Lipizzaner` [16] are: (1) The populations

Algorithm 3: `EvaluateAEs`: Evaluate all models with each other using the solution concept f_s based on the loss $\mathcal{L}$.

Input : $\mathbf{B}$: Minibatches, $\mathbf{n}$: Cell neighborhood sub-population, f_s : Solution concept
Return : ϕ : Encoder and decoder fitnesses, based on training loss $\mathcal{L}$

1 $B_r \leftarrow$ getRandomMiniBatch($\mathbf{B}$) `//Get random minibatch`
2 **for** $e, d \in \mathbf{e} \times \mathbf{d}$ **do**
  `//Evaluate all encoders and decoders`
3 $\mathcal{L}_{e,d} \leftarrow$ evaluate(e, d, B_r) `//Evaluate ANNs`
4 $\phi_\mathbf{e} \leftarrow f_s(\mathcal{L}_{\cdot,d})$ `//Fitness for encoder is from solution concept and loss value (`$\mathcal{L}$`)`
5 $\phi_\mathbf{d} \leftarrow f_s(\mathcal{L}_{e,\cdot})$ `//Fitness for decoder is from solution concept and loss value (`$\mathcal{L}$`)`
6 **return** ϕ

are encoders and decoders instead of generator and discriminators. (2) The Loss function is cooperative (minimization) instead of competitive (minimax). (3) A single AE instead of a mixture of generators is returned.

4.0.1 Complexity Analysis. We approximate the computational complexity of Lipi-AE and compare it with canonical AE training. First, autoencoders have an encoder and decoder with parameters θ and ϕ, this gives a size of $S = |\theta| + |\phi|$. Often the architecture of the encoder and decoder are symmetric, and the weights are shared $\theta = \phi$ giving $S = |\theta|$. Lipi-AE does not do weight sharing, so there are more weights to optimize.

Next we comparatively analyze of the different complexity aspects from training the AE. We denote $d = |X|$ as the size of the data set, t as the number of iterations (generations/epochs), N as the size of the population, s as the size of the neighborhood, and $b \leq d$ as the number of batches used for evaluation. The complexities considered are the following. For *fitness Evaluations* the different methods have: AE : $O(td)$ Lipi-AE : $O(tb2N^2s)$. We observe that the relationship between d and $b2N^2s$ is what differs between AE and Lipi-AE. For *Updates (Variation)* we have: AE : $O(td)$ Lipi-AE : $O(td2sN)$. We observe that the relationship between d and $b2Ns$ is what differs between AE and Lipi-AE. Finally, *Size (Storage)* has: AE : $O(S)$ Lipi-AE : $O(2NsS)$. We observe that $2Ns$ is what differs between AE and Lipi-AE. Note that Lipi-AE provides multiple solutions.

5 EXPERIMENTS

In Section 5.1 we describe the tunable binary clustering problem we use for the empirical investigation. In Section 5.2 we describe our experimental setup. In Section 5.3 we present the experimental results and analysis. In Section 5.4 we discuss the experiments.

5.1 Binary Clustering Problem

We use a simple tunable binary clustering problem to explore the reconstruction ability of Lipi-AE , see Figure 2. The input is m binary vectors $X = \{x_1, \ldots, x_m\}, |x_i| = n, x_{ij} \in \{0, 1\}$. The output is k binary vectors (centroids) $C = \{c_1, \ldots, c_m\}, |c_i| = n, c_{ij} \in \{0, 1\}$ a function $f : \{0, 1\}^{n \times m} \rightarrow: \{0, 1\}^{n \times m}, C = f(X)$, $\min \sum_{t=1}^{m} \Delta(x_t, f(x_t))$ where Δ is the hamming distance. The binary clustering problem is NP-complete [3].

[4] uses $n = 100, m = 100, k = 10$. The centroid has n-bits. Each bit in the centroid is drawn independently from a binomial distribution with $p = 0.5$, $c \sim \mathcal{B}$. An example is generated by starting

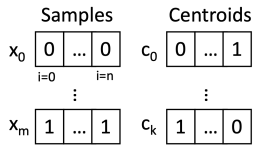


Figure 2: Illustration of the Binary Clustering Problem. Sample vectors and centroids are shown.

from the centroid and introducing noise, $q = 0.05$ probability of a bit flip. Training and test data both have nm samples.

5.2 Setup

The settings are described in more detail in Appendix A and in the repository[1]. We run 30 independent trials for all the experiment variations. We use reconstruction error in the form of L_1 distance to compare results. We use a canonical training of AEs as a baseline.

5.2.1 Experimental Design. We study the following properties that have been shown to be important for spatial coevolutionary algorithms [16, 23, 31]: *What is the impact of multiple models?* We use population size $N \in \{5, 10, 15, 20, 25, 30\}$. *What is the impact of the problem difficulty?* We use binary clustering (retrieval) and alter number of clusters $k \in \{10, 20, 40\}$ and dimension of cluster $n \in \{1000, 2000, 4000\}$. *What is the impact of coevolutionary fitness method?* We use solution concept f_s: best f_s^{min} "worst case", f_s^{max} "best case", and f_s^{mean} "mean expected utility". *What is the impact of the spatial topology and how signal propagates?* We use radius $r \in \{1, 2, 3\}$ of the neighborhood in the ring topology.

Preliminary Studies. We perform preliminary studies to determine ANN architecture and hyper-parameters. We test three variations of Autoencoder architecture: a standard autoencoder (SAE), a denoising autoencoder (DAE), and a variational autoencoder (VAE) [7]. The VAE has the best performance, so we use this architecture for all our experiments. We also sweep the important hyper-parameters of learning rate and batch size. We use learning rate 0.00001 and batch size 2 and train for $t = 400$ epochs (generations). Note that due to space constraints we do not show our preliminary studies.

5.3 Results & Analysis

We show the final performance in the form of reconstruction error (L_1) in Section 5.3.1 and the evolution of the reconstruction error. We show the effect for signal propagation in Section 5.3.2. We show the effect of solution concepts in Section 5.3.2. In Section 5.3.3 we analyze the diversity of Lipi-AE encoders and decoders. Note, for visual clarity we sometimes only plot the median value of the runs. We provide tables with values in Appendix B.

5.3.1 Lipi-AE performance – RQ-1. The answer to RQ-1 is that Lipi-AE can improve AE performance. The details for the final reconstruction error and evolutionary trajectory are presented here.

Final reconstruction error, problem complexity and population size. We first study the impact of the population size when using Lipi-AE on problems of varying difficulty. Figure 3 shows boxplots of the final reconstruction loss (L_1) for the best encoder-decoder pair on hold-out data. We observe in Figure 3a that Lipi-AE has lower L_1 than VAE for $(n = 500, k = 10, 20)$. When the difficulty increases in the form of more clusters the VAE performs as well or better than Lipi-AE . We observe that increasing n and k both reduces the AE performance. In addition, we observe that increasing the population size mostly improves the performance and reduces the variance. Note, in Appendix B.1 we show all the values and significant p-values based on Wilcoxon Ranksum test with Bonferroni correction. The difference between Lipi-AE and VAE is significant for $(n = 500, k = 10, 20)$. For all instances of $k = 50$ and $n = 4000$ there is no statistically significant difference. This implies that for the current settings and architecture of Lipi-AE these problem variants are too complex.

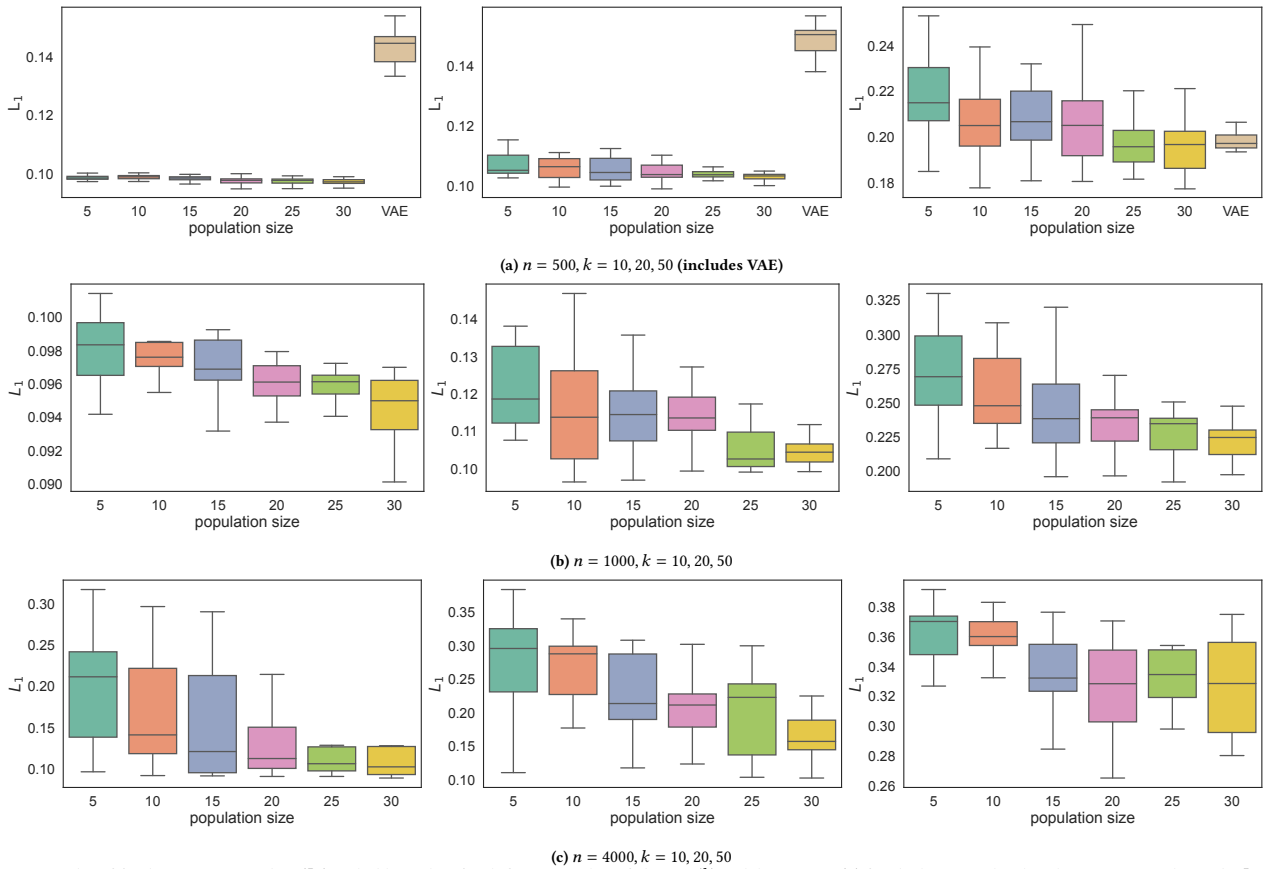
Evolution of reconstruction loss. We now study the impact of the population size when using Lipi-AE on problems of varying difficulty. Figure 4 shows a lineplot of the reconstruction loss (L_1) during training for different number of clusters (k) and dimensions (n) for the best encoder-decoder pair at each epoch. We observe in Figure 4a that the L_1 loss for the canonical VAE converges after around 100 epochs. The Lipi-AE variants take some more epochs to reach this value, but they converge at lower values. In addition for $(n = 500, k = 50)$ it looks like the Lipi-AE variants have not converged. Note, this result was implied in above when we investigated the final values. In Figure 4b and Figure 4c we observe what looks like different "phases" in the training, as distinguished by the slope of the L_1 loss. Changing the problem difficulty impacts these phases.

5.3.2 Lipi-AE sensitivity analysis – RQ-2. We identify the neighborhood size (the radius value) and the solution concept as two key components as an answer to RQ-2.

Signal propagation. We now study the impact of the neighborhood size s, by changing the radius r (Eq. 1), when using Lipi-AE . Note, due to space limitations we only show one problem variation $(n = 1000, k = 20)$ and population size $(N = 20)$. In Figure 5 we show the final reconstruction error (L_1) on hold-out data for different radius values for the best encoder-decoder pair. We observe that $r = 3$ has the lowest L_1 value. Furthermore, in Figure 6 we show the reconstruction error (L_1) for different radius values for the best encoder-decoder pair at each epoch. Here we observe that $r = 3$ has the lowest L_1 value during training. This implies that Lipi-AE can perform better with more rapid signal propagation, through a larger neighborhood. We hypothesize that this can be due to the risk of divergence for the AE training from low signal propagation.

Solution concept. We now study the impact of the solution concept f_s^{max} is "best case", f_s^{min} is "worst case", and f_s^{mean} is "mean case". Note, due to space limitations we only show one problem variation $(n = 1000, k = 20)$ and population size $(N = 20)$. In Figure 7 we show the final reconstruction error (L_1) on hold-out data for different solution concepts for the best encoder-decoder pair. We observe that the f_s^{max} solution concept has the best L_1 value. In Figure 8 we show the reconstruction loss (L_1) for different solution concepts for the best encoder-decoder pair at each epoch. Here we also observe that the f_s^{max} solution concept has the lowest L_1 loss during training. We hypothesize that the solution concept impacts

[1] https://github.com/ALFA-group/lipizzaner-ae



(a) $n = 500, k = 10, 20, 50$ **(includes VAE)**

(b) $n = 1000, k = 10, 20, 50$

(c) $n = 4000, k = 10, 20, 50$

Figure 3: Boxplot of final reconstruction loss (L_1) on hold-out data for different number of clusters (k) and dimensions (n) for the best encoder-decoder pair. Y-axis shows the L_1 loss on hold-out data. X-axis shows different population sizes for Lipi-AE (VAE is the canonical AE baseline)

the convergence rate. The "best case" might perform better since it could reduce the divergence.

5.3.3 Lipi-AE Diversity – RQ-3. The answer to RQ-3 is that Lipi-AE can increase the encoder and decoder diversity and does not converge to identity. Properties that affect the diversity are the population size (N), radius (r) and solution concept (f_s). We investigate this by measuring the L_2 distance between the encoders/decoders in the population. Note, we show the L_2 values in Appendix B.

In Figure 9 we show the diversity for encoders at $t = 400$ for $(n = 1000, k = 20)$ for the median run. The increased brightness for the cells in the top row (Figures 9a, 9b, 9c) indicates that a larger population size has more diversity. Figure 9d indicates that an increase in radius from $r = 1$ to $r = 3$ can reduce the diversity, as expected by the increased in signal propagation. For Figure 9f on the bottom row we see how the solution concept "best case" increase the diversity slightly and Figure 9e "mean expected utility" decreases it slightly. Moreover, the results show similar trends for decoders in Figure 10.

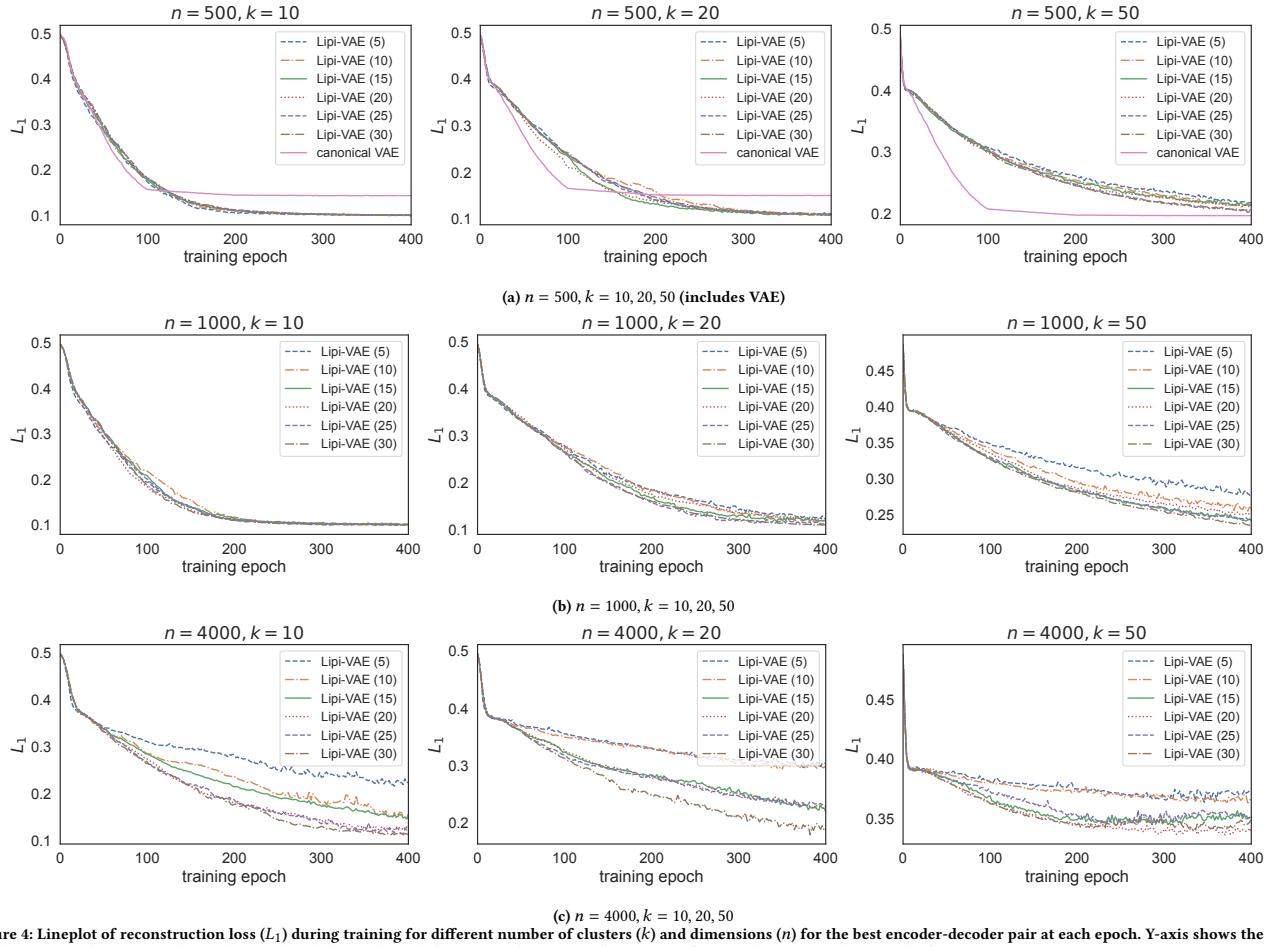
5.4 Discussion

Autoencoders can overfit due to poor initialization or collapse to the identity function. Lipi-AE introduces diversity in the encoder and decoder training. However, diversity can also lead to slower convergence. We observe that Lipi-AE can take longer to reach certain L_1 loss values. We postulate that this can be due to too much divergence in the AE training, e.g. the encoders and decoders change too often. In addition, we did not observe that everything converged to the same embedding. On the contrary there might have been too much divergence.

In our subsequent analysis of Lipi-AE we identified that the signal propagation in the form neighborhood size impacts the performance. In addition, the solution concept, how the performance for an individual is calculated, when choosing the best performing pairs ("best case") had the best performance. This can imply that the diversity can be reduced.

The computational cost of Lipi-AE is higher than canonical AE. However, for some settings canonical AE training performance had stagnated and was not able to reach the Lipi-AE performance, regardless of number of training epochs. We also observed significant differences in performance for different population sizes (which implies number of fitness evaluations) for Lipi-AE . Another consideration for Lipi-AE is the size of the search space. The search space when separating the encoder and decoder weights is larger than when they are shared. This can also impact the evolutionary search performance and training time.

Finally, a natural question is how Lipi-AE relates to the seminal paper about improving neural networks by preventing co-adaptation of feature detectors which introduced dropout to reduce



(a) $n = 500, k = 10, 20, 50$ **(includes VAE)**

(b) $n = 1000, k = 10, 20, 50$

(c) $n = 4000, k = 10, 20, 50$

Figure 4: Lineplot of reconstruction loss (L_1) during training for different number of clusters (k) and dimensions (n) for the best encoder-decoder pair at each epoch. Y-axis shows the L_1 loss on the training data. X-axis shows the epoch. The lines show different population sizes for Lipi-AE (VAE is the canonical AE baseline)

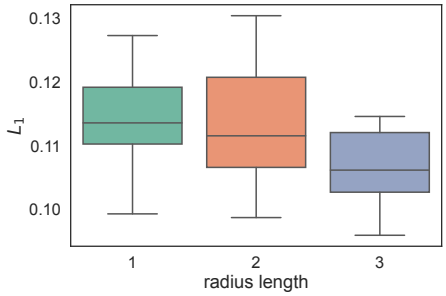


Figure 5: Boxplot of final reconstruction loss (L_1) on hold-out data for different radius values for the best encoder-decoder pair. Y-axis shows the L_1 loss on hold-out data. X-axis shows different radius values for Lipi-AE . Population size $N = 20$ and $(n = 1000, k = 20)$

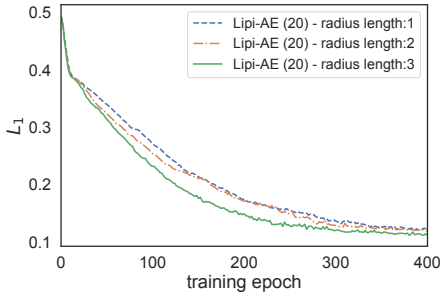


Figure 6: Lineplot of reconstruction loss (L_1) for different radius values for the best encoder-decoder pair at each epoch. Y-axis shows the L_1 loss on the training data. X-axis shows the epoch. The lines show different radius values for Lipi-AE . Population size $N = 20$ and $(n = 1000, k = 20)$

overspecializing an ANN to the training data [17]. Arguably, Lipi-AE has an approach to avoid co-adaptation in a dropout like manner. Dropout provides an average of the weight $w = w_0/N$. Lipi-AE provides $w = 1/n \sum_{t=1} \delta(w_t, w_{t-1})$. Instead of randomly dropping out nodes Lipi-AE co-adapts encoders and decoders. Thus, dropout is complementary to Lipi-AE , i.e. an ANN with dropout can be used with Lipi-AE .

Limitations. The results for Lipi-AE demonstrate that spatial cooperative coevolution can improve ANN training however we only investigated a few problem variations in a single domain. In

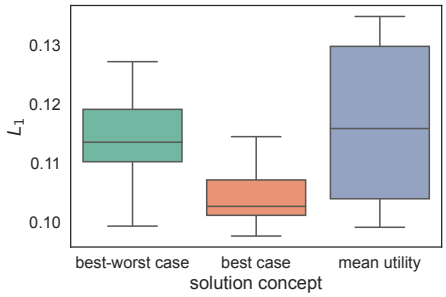


Figure 7: Boxplot of final reconstruction loss (L_1) on hold-out data for different solution concepts for the best encoder-decoder pair. Y-axis shows the L_1 loss on hold-out data. X-axis shows different solution concepts for Lipi-AE . Population size $N = 20$ and $(n = 1000, k = 20)$

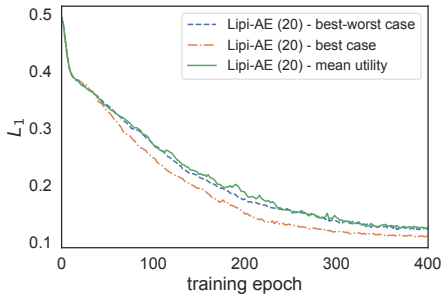
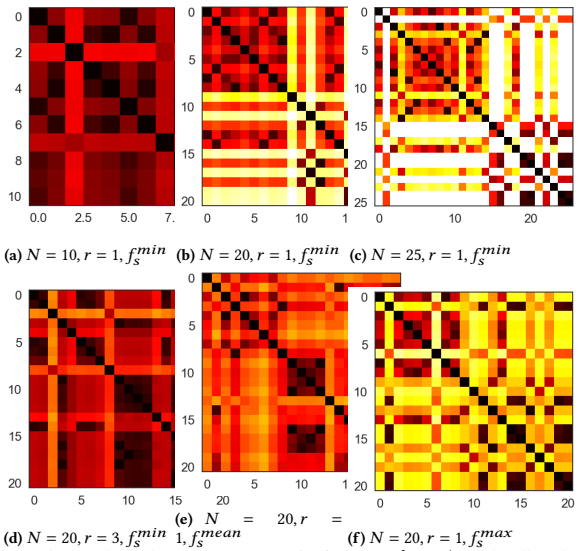


Figure 8: Lineplot of reconstruction loss (L_1) for different concepts for the best encoder-decoder pair at each epoch. Y-axis shows the L_1 loss on the training data. X-axis shows the epoch. The lines show different solution concepts for Lipi-AE . Population size $N = 20$ and $(n = 1000, k = 20)$

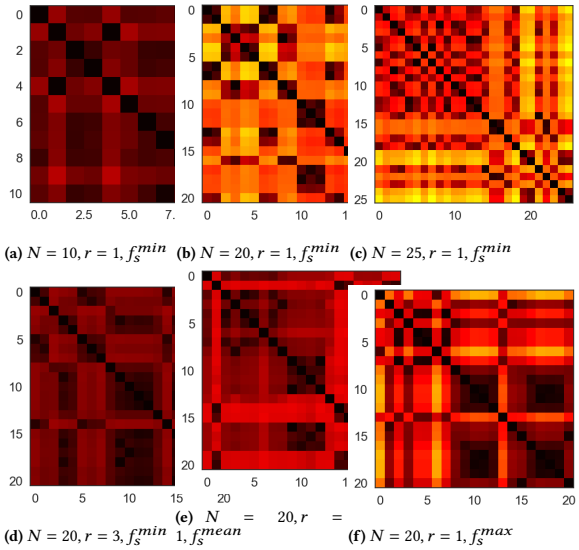


(a) $N = 10, r = 1, f_s^{min}$ (b) $N = 20, r = 1, f_s^{min}$ (c) $N = 25, r = 1, f_s^{min}$

(d) $N = 20, r = 3, f_s^{min}$ (e) $N = 20, r = 1, f_s^{mean}$ (f) $N = 20, r = 1, f_s^{max}$

Figure 9: Encoder similarity heatmap at $t = 400$ for $(n = 1000, k = 20)$. Each cell in the heatmap shows L_2 distance between Encoders in the population. Black indicates $L_2 = 0$ and brightness indicates larger L_2 value.

addition, we examined only a limited number of AE architectures and hyper-parameters.



(a) $N = 10, r = 1, f_s^{min}$ (b) $N = 20, r = 1, f_s^{min}$ (c) $N = 25, r = 1, f_s^{min}$

(d) $N = 20, r = 3, f_s^{min}$ (e) $N = 20, r = 1, f_s^{mean}$ (f) $N = 20, r = 1, f_s^{max}$

Figure 10: Decoder similarity heatmap at $t = 400$ for $(n = 1000, k = 20)$. Each cell in the heatmap shows L_2 distance between decoders in the population. Black indicates $L_2 = 0$ and brightness indicates larger L_2 value.

6 CONCLUSIONS & FUTURE WORK

Overfitting is a challenge for autoencoders. This can happen due to e.g. the initialization or a collapse to the identity function. We tried to mitigate this through the use of coevolutionary algorithms. We used cooperative coevolution with a spatial topology to train autoencoders with a system we call Lipi-AE . It has a population of encoders and a population of decoders spatially distributed in a ring topology. This creates overlapping neighborhoods of subpopulations. The best pair of encoder and decoder from the subpopulation is updated. In our experiments on simple tunable benchmark problems we observe that Lipi-AE can perform better than canonical AE training. However, Lipi-AE could take longer to reach certain loss values. We postulate that this could be due to excess divergence in the AE training. Furthermore, in the subsequent analysis of Lipi-AE we identified that the signal propagation in the form neighborhood size impacts the performance. Moreover, the solution concept, how the performance for an individual is calculated, also impacts the performance. Finally, the diversity of the encoders and decoders was maintained and a collapse to the identity function was prevented.

Future work will investigate more problems in different problem domains, e.g. images and text. We will also investigate the dynamics of the encoder-decoder pairs similarity and convergence to improve the understanding of AE convergence during training. Furthermore, we will investigate the impact of the AE architecture. Finally, we will expand the theoretical analysis.

ACKNOWLEDGMENTS

This research was partially funded by the Universidad de Málaga, Spain; TAILOR ICT-48 Network (No. 952215) funded by EU Horizon 2020 research and innovation programme; and by Universidad de Málaga, Spain (grant B1-2022_18). The authors thank the Supercomputing and Bioinformatics center at the UMA for their computer resources and assistance.

REFERENCES

[1] L. M. Antonio and C. A. C. Coello. 2018. Coevolutionary Multi-objective Evolutionary Algorithms: A Survey of the State-of-the-Art. *IEEE Transactions on Evolutionary Computation* (2018), 1–16. https://doi.org/10.1109/TEVC.2017.2767023

[2] Filipe Assuncao, David Sereno, Nuno Lourenco, Penousal Machado, and Bernardete Ribeiro. 2018. Automatic evolution of autoencoders for compressed representations. In *2018 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 1–8.

[3] Pierre Baldi. 2012. Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of ICML workshop on unsupervised and transfer learning*. JMLR Workshop and Conference Proceedings, 37–49.

[4] Pierre Baldi and Peter Sadowski. 2016. A theory of local learning, the learning channel, and the optimality of backpropagation. *Neural Networks* 83 (2016), 51–74.

[5] Francisco Charte, Antonio J Rivera, Francisco Martínez, and María J del Jesus. 2020. EvoAAA: An evolutionary methodology for automated neural autoencoder architecture search. *Integrated Computer-Aided Engineering* 27, 3 (2020), 211–231.

[6] Xiangru Chen, Yanan Sun, Mengjie Zhang, and Dezhong Peng. 2020. Evolving deep convolutional variational autoencoders for image classification. *IEEE Transactions on Evolutionary Computation* 25, 5 (2020), 815–829.

[7] Carl Doersch. 2016. Tutorial on variational autoencoders. *arXiv preprint arXiv:1606.05908* (2016).

[8] Paul R Ehrlich and Peter H Raven. 1964. Butterflies and plants: a study in coevolution. *Evolution* 18, 4 (1964), 586–608.

[9] Davood Zaman Farsa, Shahryar Rahnamayan, Azam Asilian Bidgoli, and HR Tizhoosh. 2024. Evolutionary multi-objective design of autoencoders for compact representation of histopathology whole slide images. *Computers & Operations Research* 162 (2024), 106483.

[10] David E. Goldberg. 1989. *Genetic Algorithms in Search, Optimization and Machine Learning* (1st ed.). Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[11] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[12] Tuan Ha and Xiaoying Gao. 2021. Evolving multi-view autoencoders for text classification. In *IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology*. 270–276.

[13] Jeff Hajewski and Suely Oliveira. 2020. An Evolutionary Approach to Variational Autoencoders. In *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*. 0071–0077. https://doi.org/10.1109/CCWC47524.2020.9031239

[14] Jeff Hajewski, Suely Oliveira, and Xiaoyu Xing. 2020. Evolving deep autoencoders. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*. 123–124.

[15] Erik Hemberg, Jacob Rosen, Geoff Warner, Sanith Wijesinghe, and Una-May O'Reilly. 2016. Detecting tax evasion: a co-evolutionary approach. *Artificial Intelligence and Law* 24 (2016), 149–182.

[16] Erik Hemberg, Jamal Toutouh, Abdullah Al-Dujaili, Tom Schmiedlechner, and Una-May O'Reilly. 2021. Spatial coevolution for generative adversarial network training. *ACM Transactions on Evolutionary Learning and Optimization* 1, 2 (2021), 1–28.

[17] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580* (2012).

[18] Krzysztof Krawiec and Malcolm Heywood. 2016. Solving Complex Problems with Coevolutionary Algorithms. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*. ACM, 687–713.

[19] Sean Lander and Yi Shang. 2015. EvoAE–a new evolutionary method for training autoencoders for deep learning networks. In *2015 IEEE 39th annual computer software and applications conference*, Vol. 2. IEEE, 790–795.

[20] Nan Li, Lianbo Ma, Guo Yu, Bing Xue, Mengjie Zhang, and Yaochu Jin. 2023. Survey on Evolutionary Deep Learning: Principles, Algorithms, Applications, and Open Issues. *ACM Comput. Surv.* 56, 2, Article 41 (sep 2023), 34 pages. https://doi.org/10.1145/3603704

[21] Melanie Mitchell. 2006. Coevolutionary learning with spatially distributed populations. *Computational intelligence: principles and practice* 400 (2006).

[22] Una-May O'Reilly, Jamal Toutouh, Marcos Pertierra, Daniel Prado Sanchez, Dennis Garcia, Anthony Erb Luogo, Jonathan Kelly, and Erik Hemberg. 2020. Adversarial genetic programming for cyber security: A rising application domain where GP matters. *Genetic Programming and Evolvable Machines* 21 (2020), 219–250.

[23] Elena Popovici, Anthony Bucci, R. Paul Wiegand, and Edwin D. De Jong. 2012. *Coevolutionary Principles*. Springer Berlin Heidelberg, Berlin, Heidelberg, 987–1033.

[24] Richard J Preen, Stewart W Wilson, and Larry Bull. 2021. Autoencoding with a classifier system. *IEEE Transactions on Evolutionary Computation* 25, 6 (2021), 1079–1090.

[25] Lino Rodriguez-Coayahuitl, Alicia Morales-Reyes, and Hugo Jair Escalante. 2019. Evolving autoencoding structures through genetic programming. *Genetic Programming and Evolvable Machines* 20 (2019), 413–440.

[26] Christopher D Rosin and Richard K Belew. 1997. New methods for competitive coevolution. *Evolutionary Computation* 5, 1 (1997), 1–29.

[27] Stefano Ruberto, Valerio Terragni, and Jason H Moore. 2020. Image feature learning with a genetic programming autoencoder. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference Companion*. 245–246.

[28] Finn Schofield, Luis Slyfield, and Andrew Lensen. 2023. A Genetic Programming Encoder for Increasing Autoencoder Interpretability. In *European Conference on Genetic Programming (Part of EvoStar)*. Springer, 19–35.

[29] Karl Sims. 1994. Evolving 3D morphology and behavior by competition. *Artificial life* 1, 4 (1994), 353–372.

[30] Masanori Suganuma, Mete Ozay, and Takayuki Okatani. 2018. Exploiting the potential of standard convolutional autoencoders for image restoration by evolutionary search. In *International Conference on Machine Learning*. PMLR, 4771–4780.

[31] Jamal Toutouh and Una-May O'Reilly. 2021. Signal propagation in a gradient-based and evolutionary learning system. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 377–385.