Universiteit Leiden

Master Computer Science

Weight Agnostic Neural Networks on Atari Games

Name:	Antonio Mone
Student ID:	s3113159
Date:	19/07/2023
Specialisation:	Artificial Intelligence
1st supervisor:	Bram M. Renting
2nd supervisor:	Jan N. van Rijn
3rd supervisor:	Thomas Bäck

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS) Leiden University Niels Bohrweg 1 2333 CA Leiden The Netherlands

Abstract

Most current state-of-the-art solutions for reinforcement learning tasks, such as the Atari benchmark, are gradient-based approaches. These approaches can be costly to train computationally and financially. Furthermore, they can suffer from vanishing and exploding gradients, sensibility to the initialization of the weights, and their performances often depend on the architecture chosen by the researchers.

This work intends to find a competitive, less costly option instead of gradient-based algorithms for complex reinforcement learning tasks such as the Atari benchmark. It does so by extending the experimentation of the Weight Agnostic Neural Networks (WANNs), as conceptualized by David Ha and Adam Gaier in 2019. Their work proposes a new method that modifies the NEAT algorithm and does not rely on weight training during the neural network architecture evolutionary search. They aim to encode the correct behavior for a task already in the network architecture.

This work aims to determine up to which level of performance and with which limitations we can consider the Weight Agnostic Neural Networks as a competitive alternative to SGD-based algorithms for more complex environments such as the Atari benchmark suite, on both the RAM and the image representation.

Furthermore, we compare the WANNs against gradient-based baselines such as Deep Q-Network (DQN), Advantage Actor-Critic (A2C), and Proximal Policy Optimization (PPO). We tested the WANNs and the gradient-based baselines on five Atari 2600 games from the Arcade Learning Environment. Moreover, we performed a hyperparameter importance analysis over the WANNs hyperparameters. This analysis aims to understand first which hyperparameters most influence the score over a single game and second which hyperparameters are more influential over the set of considered games.

The hyperparameter importance analysis showed that the probability of mutating an activation function is the most important hyperparameter for the evolution of the WANNs. The results of the experiments show that without the finetuning, we cannot consider the WANNs a competitive alternative to gradient-based methods over such complex tasks. The finetuned WANNs with the RAM representation over the game Battle Zone reached performances comparable to DQN and PPO, outperforming A2C. We also discuss the limitations of the WANNs and highlight possible further developments.

Contents

1 Introduction

Most current state-of-the-art solutions for reinforcement learning tasks, such as the Atari benchmark, are gradient-based approaches [1, 2, 3]. While showing state-of-the-art performances, these approaches also have some drawbacks, such as high costs, vanishing and exploding gradients, sensibility to the initialization of the weights, and high dependency on the architecture chosen by the researchers.
This work intends to find a competitive, less costly option instead of gradient-based algorithms for complex reinforcement learning tasks such as the Atari benchmark. It does so by extending the experimentation of the Weight Agnostic Neural Networks (WANNs), as conceptualized by Gaier and Ha [4].
The inspiration behind the Weight Agnostic Neural Networks [4] sees its roots directly in biology, in the innate capacities of precocial species. The term precocial refers to those species in which the newly born already possess particular skills or abilities. The main idea is that the brain's structure at birth already encodes these skills. From here, Gaier and Ha [4] aimed to search for neural network architectures able to perform specific tasks even with random weights, avoiding weight training. The search for these architectures is guided by different mutations. The mutation probabilities are defined by specific hyperparameters whose values can be finetuned.
Their first step was to give the weights a less important role in the process compared to their crucial role in traditional approaches. Instead of having random weights, each connection in the network has the same weight, and they test each architecture with a variety of weight values.
Gaier and Ha [4] tested this method on reinforcement learning control tasks such as CartPoleSwingUp, BipedalWalker, and CarRacing. Control tasks such as CartPoleSwingUp and BipedalWalker have an observation space described by a state vector, while the observation space of CarRacing is described by an image. This leaves the following questions: how do the WANNs scale on more challenging tasks? Which hyperparameters influence their performances the most? Given the difference in observation spaces between the experimented tasks, how do the WANNs perform with different observation space representations?
Since the WANNs do not have the drawbacks mentioned above, and given the results reported by Gaier and Ha [4] on the tasks they experimented on, we decided to explore the potential of the WANNs on more complex benchmarks such as the Atari benchmark. The Atari benchmark also provides both a state vector representation of the observation space, the status of the RAM representation, and an image representation.
The aim of this work can be captured in the following research question:

Can Weight Agnostic Neural Networks be considered a competitive alternative to gradient-based algorithms on Atari environments?

To answer our main research question, we have devised the following four subquestions:

- **Q1:** Which WANNs hyperparameters are more influential for each game and across games?
- **Q2:** How do the WANNs perform on the RAM representation compared to the gradient-based approaches?

- **Q3:** How do the WANNs perform on the image representation compared to the gradient-based approaches?

- **Q4:** How much does the finetuning of WANNs impact the performances?

Instead of comparing the WANNs with the current state of the art (such as MuZero [1], Agent57 [2], and GDI-H3 [3]), given the differences in compute expenses and complexities, they are compared with the gradient-based agents Deep Q-Networks (DQN), Advantage Actor-Critic (A2C), Proximal Policy Approximation (PPO), trained using the stable baselines library [5]. Even if outdated, these agents still represent a reliable baseline against which to evaluate the performances of a new approach such as the one explored in this work.
We carry out this analysis using the Atari-5 benchmark [6], consisting of Battle Zone, Double Dunk, Name This Game, Phoenix and Q*bert.
These environments provide an image observation space and a RAM observation space. Both were used to carry out the experiments. In order to use images as input, Gaier and Ha [4] used a pre-trained Variational Autoencoder (VAE). The use of a pre-trained variational autoencoder means that we cannot exactly call the full system "weight agnostic" since it relies on an element trained with gradients. In order to explore the potentiality of the WANNs independently from gradient-based elements, we also experiment with the RAM representation, and we experiment with the gradient-based agents using this representation as well.
A hyperparameter importance analysis is carried out to investigate which hyperparameters have a more significant impact on performance during evolution over each game and which hyperparameters are more important across different games. Out of the explored configurations, the ones yielding better predicted performances are used to carry out the complete evolution process over each game.
Furthermore, the finetuning of the evolved neural networks is performed using the Covariance Matrix Adaptation Evolution Strategy (CMA-ES), a gradient-free stochastic optimization evolutionary algorithm.
We present the results achieved by the WANNs in comparison with the gradient-based agents before and after finetuning. The results before the finetuning show that the gradient-based approaches outperformed the WANNs in all the games using both the RAM and the image representation. The results after the finetuning process show interesting results over the RAM representation, with the RAM finetuned WANN performing comparably to DQN and PPO over the game Battle Zone, outperforming A2C. Over the game Phoenix, the RAM finetuned WANN outperformed DQN and A2C, performing slightly worse than PPO.
The main contributions of this work can be summed up as follows:

- A comprehensive evaluation was carried out, which covered 5 Atari games and compared the Weight Agnostic Neural Networks (WANNs), firstly evolved with shared weights and successively finetuned, using image representation and RAM representation. The WANNs were compared against gradient-based methods.

- Insights into the hyperparameter configurations that lead the evolutionary search of the WANNs through a hyperparameter importance study. This presents an analysis of how the tuning of each hyperparameter affects the predicted performance of the WANNs over each game. It also shows how the hyperparameter responsible for the probability of mutating the activation function is the most important across games.

This thesis is structured as follows. In Section 2, we will mention the Related Works. In Section 3, we will provide information on topics on which this work is based, such as reinforcement learning, evolutionary algorithms, and CMA-ES, used as a gradient-free method to finetune the weights of the evolved WANNs. In Section 4, we will present the methods used throughout this work, more specifically, the WANNs, the proposed VAE, DQN, A2C, and PPO. In Section 5, we will present the setup used to carry out the experiments, including the used benchmark, the hyperparameter importance analysis, the hyperparameters used for the WANNs, and the hyperparameters and architectures used for the gradient-based agents. In Section 6, we discuss the achieved results and highlight the potential and limitations of our work. In Section 7, we conclude our work outlining possible directions for further developments.

2 Related Work

As mentioned by the authors of the original WANNs paper Gaier and Ha [4], their work sees connections and differences not only with various topics in artificial intelligence and deep learning but also in other fields. Considering that our work is carried out as an expansion on top of their work, we see some of the same connections.

Neural Architecture Search This field traces its origins at the beginning of the 1990s with the first works on evolutionary computing as search algorithms for possible topologies for neural networks [7, 8, 9, 10, 11]. The goal of the techniques of this field is to search for topologies that, after training the weights, can perform better than human-designed topologies. Therefore, even if the network architecture gains slightly more importance, the weights are still the key and main actor in providing the solutions in encoding the knowledge [4].
The WANNs have a similar but different goal. They search for neural networks that encode the solution of a task directly in the architecture of the network, even without weight training. They minimize the importance of the weights in the process. This fundamental difference also highlights the difference in computational cost since the weights of each candidate architecture need to be trained in the classic neural architecture search techniques. This costly operation is instead avoided in the WANNs.

Variance Networks The WANN approach also connects with the work of Neklyudov et al. [12] since both approaches use a uniform distribution with zero mean to sample the weights from, and both approaches evaluate the performances on network ensembles. While the WANN uses a fixed uniform distribution [4], the uniform distribution of the variance layer concept is parametrized by its variance. Their work is also connected to **Bayesian Neural Networks** (BNNs) [13]. In BNNs, the weight parameters are not treated as fixed values but are considered random variables sampled from a distribution. Taking different values at each iteration, they add uncertainty to the model. The distribution's parameters can be learned from the data, but the number of parameters in the distribution is often larger than the number of weights in the network.

Network Pruning The process of Network Pruning was introduced by LeCun et al. [14] and researched in the following years, leading to its application also to CNNs [15]. Starting from a fully trained neural network architecture, it produces smaller networks by removing the connections with small weights, creating sparse networks that supposedly still perform as well

as the starting, trained one. While this can be seen as a top-down perspective, the WANN approaches the concept in the other direction, in a bottom-up fashion, starting from minimal networks and adding connections and, therefore, complexity. While the pruning of a network requires an already trained network that performs supposedly well on the task, the WANN evolutionary approach does not have these requirements.

Algorithmic Information Theory Following up on the concept of aiming for simpler neural networks, the WANNs also see connections with the field of Algorithmic Information Theory (AIT). Specifically, it connects with the Kolmogorov complexity of a computable object [16], the shortest length of a computer program capable of producing that object as an output, and the concept of Minimum Description Length (MDL) [17].
Throughout the years, much work has been done towards the simplification of neural network architectures, with ideas related to MDL, such as soft-weight sharing and the introduction of noise in the weights in order to reduce the amount of information [18, 19].
Instead of working on a predefined network architecture of which we want to understand the necessary (and minimal) information capacity to represent the weights, we focus on finding the minimal architecture required to encode solutions to a variety of tasks.
Instead of having random weights, the authors of the original WANNs paper Gaier and Ha [4] imposed for each connection in the network the same weight, and they tested each architecture with a variety of weight values.
A single shared weight value is preferred to a sampled set of random weights because, given the growth of the network generation after generation and a subsequential increase in dimensionality, the sampling of the weight space becomes unfeasible due to the curse of dimensionality.

Neuroscience The original WANNs paper [4] links itself to the concept of connectomes presented by Seung [20]. Connectomes are graphs representing the mapping of all the neural connections in a brain. Brains like the human one are incredibly complex and dense, therefore challenging to map. In contrast, more superficial structures like the ones created in this process can be analyzed to understand how the network structure learns, representing the acquired knowledge through connections.
In this work, connectomes are used to analyze the networks generated by the evolution process for each game, both with the RAM and the image representation.

Hyperparameter Importance Analysis Hutter et al. [21] proposed an algorithm that could compute the marginal of random forest predictions and successively use these predictions within a functional ANOVA framework to quantify the importance of hyperparameters and of the interactions between hyperparameters.
Based on this work, van Rijn and Hutter [22] explored the possibility of defining, given an algorithm, which are its most important hyperparameters not only on a specific dataset but across datasets. Similarly, Sharma et al. [23] worked on the definition of the most important hyperparameters across datasets for residual neural network architectures.
In the scope of this work, the importance of the WANNs hyperparameters guiding the exact evolutionary behavior of the networks is analyzed. This analysis aims to understand which hyperparameters have a more significant impact on the WANN performance over the evolution of the network for each game and which hyperparameters are more important across different games.

Atari Benchmark Since the release of the Arcade Learning Environment by Bellemare et al. [24], which includes a suite of 57 games originally developed to be played on the Atari 2600 video console, this framework has been considered a standard well-suited benchmark to analyze the performances of reinforcement learning agents on a wide set of different tasks, testing the general capacity of an agent, such that a significant number of state-of-the-art advancements in the field have been tested against this framework [1, 2, 25, 26, 27].
Bellemare et al. [24] affirm that the Arcade Learning Environment framework represents a good benchmark because, in an ideal situation, "the algorithm should be compared across domains that are (i) varied enough to claim generality, (ii) each interesting enough to be representative of settings that might be faced in practice, and (iii) each created by an independent party to be free of experimenter's bias." [24].
As observed by Obando-Ceron and Castro [28], applying traditional methods to the entire suite of 57 games is computationally expensive. As noticed by Aitchison et al. [6], such costs restrict the possibility of generating results on the whole benchmark to a handful of research groups, highlighting how out of 17 algorithms listed on the paperswithcode website[1] on the full Atari-57 benchmark, 16 of them are from research groups within Google or Deepmind.
Intending to address the issue of the high costs in generating results over the complete Atari-57 benchmark, Aitchison et al. [6] proposed a new methodology for selecting a subset of games within a certain benchmark suite, intending to keep this new subset small but still representative of the original full suite.
While different works throughout the years did select subsets of games (e.g., the Deep Q-learning Network (DQN) paper by Mnih et al. [25] focuses on a subset of 7 games), the authors suggest specific subsets of the Atari-57 benchmark, respectively called Atari-5, Atari-3, and Atari-1, basing the selection of games on their ability to predict median score estimates. In their work, firstly, they formalize the summary score of a subset of a Reinforcement Learning benchmark such that "it minimizes the number of environments, and is informative of the algorithm's performances" [6]. Subsequently, Aitchison et al. [6] point out how for the Arcade Learning Environment, a widely used summary score is the median score, and they aim to predict it as a target score. Taking only small subsets of environments, they consider the prediction of this target score as a viable methodology to understand which specific subset best predicts the target score through linear regression.
Given a set of algorithms, together with the individual performances on the environments and the aggregate score of each of the algorithms over the environments, taking a subset of environments together with the corresponding aggregated performances per algorithm, the goal is to find a mapping that best predicts the aggregated score from the subset of aggregated performances [6].
Over the Atari-57 benchmark, the possible scores within the games can differ by different orders of magnitude (e.g., the game Double Dunk has a score bounded in [-24,24], while the game Phoenix has no such bound having the current best score on paperswithcode of 959580 points, achieved by GDI-H3 [3]). Aitchison et al. [6] log normalized the scores to apply this methodology to the Atari-57 benchmark.
Aitchison et al. [6] created 57 single-game linear regression models, using the score of a single game to predict the median score over the whole Atari-57 benchmark. They used the score obtained in a specific game as the independent variable and the median score over all 57 games as the dependent variable. The games were then ranked based on how well the model could predict the median score.

[1] `www.paperswithcode.com`

Subsequently, Aitchison et al. [6] applied this procedure to all the five-games subsets, defining the best one as Atari-5, which includes Battle Zone, Double Dunk, Name This Game, Phoenix, and Q*bert [6]. Then from Atari-5, they evaluated all the triple-games subsets within Atari-5 to determine Atari-3 as the best subset of three games: Battle Zone, Name This Game, and Phoenix. Finally, they defined Atari-1 as the best subset containing only one game from the Atari-3 subset, identifying Name This Game as the best single-game subset within Atari-3. The results obtained by Aitchison et al. [6] point out how some games, such as Pong, Surround, and Tennis, carry almost no predictive abilities. Interestingly, the single game that better predicts the median score is not the game of the Atari-1 benchmark, Name This Game, but Zaxxon.

3 Background

3.1 Reinforcement Learning

The standard reinforcement learning setting is usually defined as an agent $\mathcal{A}$ interacting with an environment $\mathcal{E}$ for several discrete timesteps. At each timestep t, $\mathcal{A}$ receives the state s_t and selects an action to perform at that timestep a_t from the set of possible actions. The agents select the action based on the policy π, which represents a mapping from states to actions [27]. Interacting with the environment, the agent will receive back the next state of the environment s_{t+1} and a reward at that timestep r_t. Continuing throughout this loop, the agent will reach a terminal state and the game will be over, starting the process again for a new episode. The goal of the agents is to maximize the cumulative reward from timestep t until the end of the episode, $R_t = \sum_{k=0}^{\infty} \gamma^k r_k$, where each k represents the steps performed starting from t, and γ represents a discount factor for future and immediate rewards, aiming at maximizing the reward at each state [27]. We formalize the timesteps from t to infinity. In practice, the sum in the formula is not for infinite timesteps but for a specific finite horizon of timesteps, that can depend from the setup. The action-value function $Q^\pi(s, a)$ is responsible for estimating the expected return of performing a given action a in a given state s following a given policy π. Considering policy gradient algorithms such as A2C and PPO, the function can be represented as [27]

$$Q^\pi(s, a) = \mathbb{E}[R_t | s_t = s, a] \tag{1}$$

The value function $V^\pi(s)$ instead is responsible for estimating how good it is for the agent to be in a specific state in terms of future rewards. In the scope of policy gradient algorithms, it can be described as [27]

$$V^\pi(s) = \mathbb{E}[R_t | s_t = s] \tag{2}$$

Based on the action-value function, we can define the optimal action-value function as $Q^*(s, a)$, representing the maximum expected return attainable by following a particular policy π after observing a sequence of states s and performing an action a,

$$Q^*(s, a) = \max_\pi \mathbb{E}[R_t | s_t = s, a_t = a, \pi] \tag{3}$$

Reinforcement Learning algorithms can be categorized based on different perspectives:

- If the algorithm is value-based or policy-based:
  - Value-based methods aim at estimating the value function and then derive the policy indirectly from the value function.

  - Policy-based methods aim at explicitly learning the policy function $(\pi : s \rightarrow a)$ directly.

- If the algorithm is model-based or model-free:
  - Model-based methods aim at learning a model of the environment [29] and the dynamics of the environment to predict the environment's responses.
  - Model-free methods interact directly with the environments.

- If the algorithm is on-policy or off-policy:
  - In on-policy methods, the agent interacts with the environment following the same policy that is updated while experiencing the environment.
  - In off-policy methods, the agent interacts with the environment using a policy and uses the experience gained by this policy to update and optimize a different "target" policy.

In this work, we will see a different set of agents as elements of comparison: an off-policy, value-based, and model-free method such as DQN [25] in Section 4.2; an on-policy, policy-based, model-free such as PPO[26] in Section 4.4 and an on-policy, model-free one such as A2C[27] in Section 4.3, which combines value-based and policy-based components with its Actor-Critic approach.

3.2 Evolutionary Algorithms

Drawing inspiration from the concept of evolution, Evolutionary Algorithms are a class of population-based optimization algorithms that, through the application of biology-inspired genetic operators (such as mutation or recombination), iteratively optimize a population of candidate solutions.
Seeing its roots already in the 1960s and 1970s [30, 31], with the development of ideas such as evolution strategies and evolutionary programming, this class of algorithms did not take off until the 1980s [32], with publications such as Goldberg's doctoral thesis on gas pipeline optimization and later on with its book [33]. In the 1990s, much work has been carried out focusing on the differences between genetic algorithms and evolutionary strategies, extended research on selection mechanisms, self-adaptation, and surveys have been published [34, 35, 36, 37]. Books such as the one by Bäck [38] were published. In this book, the author overviews the main approaches considered in evolutionary algorithms, such as genetic algorithms, evolution strategies, and evolutionary programming.
An outline of the main elements and procedure of a general evolutionary algorithm is given as follows: The algorithm starts by initializing the population of individuals, where each individual represents a potential solution for the problem considered, often encoded as binary strings, integers, combinations of them, or real-valued vectors. Considering this algorithm as a search process, each individual can be seen as a search point in the solution space [39].
Each individual is then evaluated based on an objective function that will estimate a fitness value [39], determining the quality of that individual at solving the considered task.
Based on the fitness function and the chosen selection mechanism, each individual can have a higher or lower chance of being selected for the continuation of the process. There are several

selection mechanisms, and the one used by Gaier and Ha [4] is tournament selection, explained more in detail in Section 4.1.
The successive step, maintaining the inspiration from the biological process, is reproduction. The selected individuals are now part of a reproduction process as parents, and a new generation of individuals is created by applying previously mentioned operators such as recombination and mutation.
In the recombination operator, the two selected parents exchange genetic information. The mutation operator is supposed to apply minor modifications to one individual's genome [39]. While genetic algorithms emphasize the recombination operator as the leading actor in the search [39], evolution strategies instead employ normally distributed mutations to modify real-valued vectors. Finally, evolutionary programming accentuates mutation, not applying recombination of individuals [39].
This loop is repeated for several generations until the predefined budget runs out or when a satisfactory solution is found. In doing so, this algorithm explores the search space and improves the quality of the solutions.

3.3 Covariance Matrix Adaptation Evolution Strategy

Firstly introduced by Hansen and Ostermeier [40] and recently reintroduced by Hansen [41], the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) is a derivative-free stochastic optimization evolutionary algorithm. As mentioned in the previous section, in an evolutionary algorithm, the mutation operator is supposed to apply minor modifications to one individual's genome [39], which is the representation of a neural network in our work.
CMA-ES is based on the covariance matrix, a square matrix that describes the dependencies between pairs of variables in the distribution of a random vector.
Given an initial solution and an initial standard deviation, the CMA-ES process starts by defining an initial distribution from which individuals are sampled. It does so by initializing a covariance matrix, a mean vector, the step-size, and the evolution paths for the covariance matrix. The evolution paths can be seen as a historical record of the search directions, containing information about the correlation between consecutive steps. After the evaluation of the fitness of these individuals, the best ones are selected. These best solutions are then recombined and, through a step-size control based on the evolution path of the step-size (which defines the magnitude of the update), used to adapt the covariance matrix. In doing so iteratively, the algorithm explores promising areas of the search space.

4 Methods

In this section, we describe the methods used throughout this work. The section of interest in this work is Section 4.1, where we describe the WANNs. In Sections 4.2,4.3, and 4.4, we describe the gradient-based methods we want to compare the WANNs against.

4.1 Weight Agnostic Neural Networks

Introduced by Gaier and Ha [4], the Weight Agnostic Neural Networks framework is an evolutionary way to search for neural network architectures which encode solutions directly in the architecture of the network itself rather than in the weights.

The evolutionary search process for these architectures takes inspiration from the NeuroEvolution of Augmenting Topologies (NEAT) strategy proposed by Stanley and Miikkulainen [42]. While NEAT optimizes at the same time both the architecture and the weights, the WANNs avoid the expensive weight optimization step. Instead, each network is evaluated with a variety of shared-weight values. They enable weight-sharing on all the weights in the network, reducing the number of weights to sample to just one[4]. This allows for a more efficient sampling of weight values since it is reduced to the sampling of a single value. In doing so, they highlight the role of the architecture, reducing the importance of the weights.
Gaier and Ha [4] decided to enable weight-sharing instead of sampling random weight values because, given the growth of the network generation after generation and a subsequential increase in dimensionality, the sampling of the weight space becomes unfeasible due to the curse of dimensionality.
The performance of each architecture, and therefore of each individual, is defined as the cumulative reward of that individual with each available shared-weight value, averaged over the number of values tested.
The process, also shown in the Figure 1 from the original WANNs paper [4],is carried out as follows:

1. **Initialization** A first population of individuals is generated, where each individual is a minimal neural network.

2. **Evaluation** Each individual is tested over at least one rollout for each one of the possible shared weight values.

3. **Rank** The individuals are ranked based on the scores and complexity of the networks.

4. **Vary** Through tournament selection, the best ones are picked, and randomly mutating them, a new population is created.

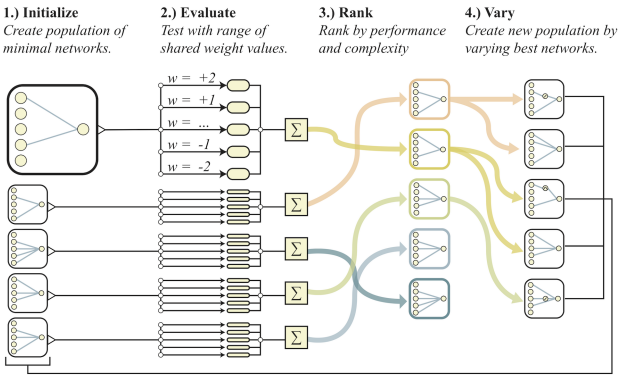


Figure 1: Graphic overview of the WANNs main evolutionary search loop [4]

After the last step, the algorithm starts back from the evaluation step. The minimal networks generated at the beginning of the loop consist of the input and output layers, and only a fraction of the possible connections between the two layers are activated.
The selection method used in this work is the tournament selection, given its efficiency and its possibility to adjust the selection pressure as needed, as observed by Miller and Goldberg [43]. The primary functioning of the tournament selection involves hosting a series of "tournaments"

between randomly picked individuals. The selected individuals are compared based on their ranking, and the ones with the highest ranking advance in the process. Since Gaier and Ha [4] opted for a deterministic version of the algorithm, the individual with the highest fitness wins the tournaments and is selected.
The main elements responsible for the modifications of the network topology search are three possible random mutations, which are also operators inspired by NEAT [42]. By splitting an existing connection, a new node is added with a random activation function assigned. Furthermore, a new connection can be established between two nodes that were not previously connected. Finally, it is possible to randomly reassign the activation function of one of the hidden nodes to one of the available ones.
As mentioned in Section 2, one of the aims is not only to have networks capable of encoding a solution of a task in the architecture but also that this architecture is as simple as possible. Aiming at having simpler networks, they applied the connection cost methodology from Clune et al. [44], ensuring that the less complex one will be chosen when comparing two networks with similar performances.
Each individual is evaluated over three elements: the mean reward of all the shared-weight values, the complexity of the architecture identified as the number of connections, and the highest peak performance reached within the evaluations of the various shared-weight values. Approaching finding the best individuals as a multi-objective optimization problem, they sorted the individuals on a Pareto dominance basis [45]. In doing so, the authors aim to pick a more complex network only if it shows a noticeable improvement in performance.
Since such an increase in performance might need a higher degree of complexity, they relax this constraint by ranking based on mean and peak performances 20% of the time and ranking following the criteria of mean performance and number of connections 80% of the time.
Given the introduction of the weight-sharing mechanism, we report the three metrics also reported by Gaier and Ha [4] in their code at the end of each generation:

- **Elite Fit (EF)** For each generation, the elite fit reports the average cumulative reward of the best-performing individual of that generation over the six shared-weight values.

- **Best Fit (BF)** The best fit reports the highest average cumulative reward achieved overall so far. It will be the same as the elite fit after the first generation. When a newer individual has a higher average cumulative reward, this network is evaluated in what can be considered an evaluation generation that is not part of the evolutionary process. If the average cumulative reward over this generation is higher than the previous best fit, the best fit will be updated to the newer value.

- **Peak Fit (PF)** The peak fit reports the single highest reward reached by the current best fit. This value can sometimes decrease. For instance, the current best fit's rewards are $[200, 200, 200, 200, 200, 400]$, the average reward will be 233.33, and the peak fit will be 400. In the next generation, the elite fit's rewards are $[300, 300, 300, 300, 300, 300]$, with an average reward of 300 and a peak fit of 300. This elite fit will be the new best fit because its cumulative reward is higher, and the peak fit will be the new best fit's peak fit, even if it is lower than the previous one.

4.1.1 WANN Hyperparameters

The WANNs project includes a series of hyperparameters with a list and description available in the original WANNs paper's repository[4].[2]
Some of these hyperparameters can depend on the hardware (i.e., the $popSize$ hyperparameter, the number of individuals in the population, which seems reasonable to set at the same number of CPU cores available to parallelize the process).
The hyperparameters responsible for guiding the architecture's evolution are the probabilities related to the possible mutation operators since the likelihood of a specific mutation will lead the search in different directions. Together with the probabilities of adding a node ($prob\_addNode$), adding a connection ($prob\_addConn$), and changing the activation function ($prob\_mutAct$); also the probabilities of enabling a disabled connection ($prob\_enable$) and of enabling each initial connection ($prob\_initEnable$) influence the direction of the search since they define the starting point of the search. Other hyperparameters, such as $select\_eliteRatio$, are in charge of the greediness of the algorithm, managing the number of individuals in the population that will pass on to the next generation without mutations.
In the original repository, JSON files are included, containing both the default configurations for the process and the modifications for each of the environments tested initially (CartPoleSwingUp, BipedalWakler-v2, CarRacing-v0). Each of these files refers to the specific task defined in the configuration file that will instantiate the correct environment with the newly updated hyperparameter configuration.

4.1.2 Variational Autoencoder

In the original WANN paper [4], the authors used a pre-trained Variational Autoencoder (VAE) to encode the images into a vector to use as an input for the WANNs. We trained a VAE specifically for each game.
The gradient-based approaches apply specific preprocessing to the screen, scaling the RGB 210x160 pixels to grayscale 84x84 images. The same preprocessing steps have been applied to the VAE at first. The architecture of the encoder of the first VAE was inspired by the structure of the CNN used by Mnih et al. [25].
We propose another VAE structure, described more in-depth in Appendix B. The reconstructions obtained with both architectures are shown in Appendix B.
The encoder of the proposed VAE takes as input the 160x160 RGB images instead of the 84x84 grayscale ones. The Gym environment's RAM representation is a vector of shape (128,). The dimension of the latent vector of the VAE used as input for the WANN has the same shape. The decoder takes as input a vector of shape (128,) and reconstructs the image.
The reconstructed images for the game Q*bert using this architecture are shown in Appendix B. While the convolutional layers of gradient-based approaches can learn to visualize the correct details while playing and exploring more levels, the VAE was trained on a dataset of images created using a random agent. This represents a potential limitation for the reconstruction quality of the image-based WANNs, since the reconstruction quality will gradually degrade because the VAE might struggle to encode the images when the individuals reach unseen levels, deviating from the initial dataset.
In a game like Q*Bert, a random agent can easily activate the cubes of the higher "floors" since the game starts with the character on the top cube. Behaving randomly, the agent does not

[2] `https://github.com/google/brain-tokyo-workshop/blob/master/WANNRelease/prettyNeatWann/p/hypkey.txt`

reach the lower cubes very often. This leads to the gradual degradation of the reconstruction quality previously mentioned. The cubes of the lower "floors", being reached less often, are reconstructed less accurately.

4.2 Deep Q-Network

Firstly introduced by Mnih et al. [25], and based on the Q-Learning algorithm introduced by Watkins and Dayan [46], the Deep Q-Network is a model-free off-policy algorithm initially created aims at connecting a reinforcement learning algorithm to a deep neural network efficiently operating on RGB training data. In a Temporal-Difference (TD) Learning method such as DQN [47], the optimal-value function is defined as

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_{t+1} + \gamma \text{max}_a Q(s_{t+1}, a) - Q(s_t, a_t)][47] \tag{4}$$

In algorithms such as DQN, the action-value function can be described using an approximator function, defined as $Q(s, a; \theta)$, to estimate the action-value function $Q(s, a; \theta) \approx Q^*(s, a)$. In this case, the approximator function is non-linear and is represented by a neural network, the Q-Network, with parameters θ. The process of optimization and training of the Q-Network minimizes a series of loss functions that changes at every ith iteration, as shown in

$$\mathcal{L}_i(\theta_i) = \mathbb{E}(r + \gamma \text{max}_{a'} Q(s', a'; \theta_{i-1}) - Q(s, a; \theta_i)^2)[27] \tag{5}$$

This process is called one-step Q-Learning because the action value $Q(s, a)$ is updated based on the one-step return given by $r + \gamma \max_{a'} Q(s', a'; \theta_{i-1})$, where s' represents the next state encountered by the agent. The authors also utilize the experience replay technique proposed by Lin [48] where the experience of the agent at each timestep, defined as $e_t = (s_t, a_t, r_t, s_{t+1})$, is stored into a dataset and pooled into a replay buffer. From this buffer, experiences are drawn, and Q-Learning updates are applied. After these updates, the agent will decide which action to perform, basing the decision on a ϵ-greedy policy [25]. The agent has to learn off-policy since the learning is based on experience replay and generated by parameters that differ from the current ones. Furthermore, since it does not try to model the environment, it is a model-free algorithm.

4.3 Advantage Actor-Critic

The Advantage Actor-Critic approach proposed by Mnih et al. [27] instead of the experience replay executes multiple agents in parallel, using multiple environment instances. The main idea behind this approach is to propose an alternative to the experience replay buffer, which requires more memory and computation per real interaction [27]. The agent will be experiencing a wide range of different states at every timestep. This different paradigm for deep reinforcement learning allows the robust application, using deep neural networks, of a broader range of on-policy algorithms [27]. Differently from the method explained in the previous section, the Advantage Actor-Critic algorithm is model-free policy-based, and it tries to directly parametrize the policy $\pi(a|s; \theta)$, and the parameters θ are updated through gradient ascent on the estimated return $\mathbb{E}[R_t]$. It is based on the REINFORCE family of algorithm [49], which performs updates of the policy parameters θ in the direction of

$$\nabla_\theta \log \pi(a_t|s_t; \theta) R_t \tag{6}$$

which is the unbiased estimate of $\nabla_\theta E[R_t]$. Williams [49] points out that this estimate shows a high variance. A way to reduce this variance is to subtract a learned function with the state as input $b_t(s_t)$, called baseline, from the return R_t [49]. The resulting gradient is then

$$\nabla_\theta \log \pi(a_t|s_t;\theta)(R_t - b_t(s_t))[27] \tag{7}$$

One of the baselines often used to scale the variance of the policy gradient is an estimate of the value function, $V^\pi(s_t) \approx b_t(s_t)$. This estimate allows us to define the concept of advantage of an action a_t in a state s_t as

$$A(s_t, a_t) = Q(a_t, s_t) - V(s_t) \tag{8}$$

considering the relation $R_t - b_t(s_t)$ as such since $Q^\pi(a_t, s_t)$ estimates R_t and $V^\pi(s_t)$ estimates b_t [27]. In doing so, emerges what can be seen as an actor-critic dynamic between the policy π (the actor) and the baseline b_t (the critic) [47].
In the scope of this work, instead of the asynchronous version of the actor-critic algorithm (A3C) proposed by Mnih et al. [27], we use its synchronous version, A2C. A2C waits for each actor to perform the number of predetermined t_{max} steps before the update, calculating the average of the reward reached by the actors until t_{max}.

4.4 Proximal Policy Optimization

Belonging to the family of policy gradient methods together with A2C, the Proximal Policy Optimization (PPO) algorithm introduced by Schulman et al. [26] makes use of a "surrogate" objective function, optimized while alternately sampling data from the environment. Based on the gradient shown in Equation 7 and the concept of advantage mentioned in the previous section and in Equation 8, a gradient estimator $\hat{g}$ can have the form

$$\hat{g} = \hat{\mathbb{E}}_t[\nabla_\theta \log \pi_\theta(a_t|s_t) A(s_t, a_t)] \tag{9}$$

The aim of Schulman et al. [26] was to propose an improvement over their previous work on Trust Region Policy Optimization (TRPO)[50]. In the TRPO paper, Schulman et al. [50] aim to optimize the policy through a constrained optimization problem, where the "surrogate" objective function is optimized but with a constraint on the size of the policy update in order to have a more stable optimization of the policy. They aim to maximize

$$\hat{\mathbb{E}}_t[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}\hat{A}] \tag{10}$$

where θ_{old} represents the parameters of the policy before the update, constrained by

$$\hat{\mathbb{E}}_t[\mathrm{KL}[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)]] \leq \delta \tag{11}$$

as shown by [50, 26]. Defining

$$\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)} \tag{12}$$

as the probability ratio $r_t(\theta)$, such that $r_t(\theta_{old} = 1)$. TRPO's maximization objective is

$$L(\theta) = \hat{\mathbb{E}}_t[r_t(\theta)\hat{A}_t][50] \tag{13}$$

while the new objective proposed by PPO is

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)][26] \tag{14}$$

where ϵ is a hyperparameter. While the first term in the min function is the TRPO's maximization objective, the second term clips the probability ratio modifying the surrogate objective. Considering the minimum, the final objective is a lower bound on the unclipped objective [26]. As seen for A2C, also PPO can be parallelized over a number of actors. At each iteration, N actors collect a predefined amount of T timesteps of data. Afterwards, a surrogate loss is computed over NT timesteps of data and optimized with minibatch stochastic gradient descent for a number of epochs.

5 Experimental Setup

While the WANNs were originally tested on relatively simple tasks, we evaluated them on more complex tasks, such as games from the Atari benchmark. In the tasks evaluated by Gaier and Ha [4] (CartPoleSwingUp, BipedalWalker, and CarRacing), the information in the observation space is well-defined, such as the cart position and velocity for CartPole.
CartPoleSwingUp and BipedalWalker have low-dimensional observation spaces in the form of a state vector of four dimensions for CartPoleSwingUp, and a state vector of 24 dimensions for BipedalWalker. The CarRacing environment observation space consists instead of a 96x96x3 pixel image. The authors used a pre-trained Variational Autoencoder (VAE) to interpret the pixels. In the Atari benchmark, the observation space of the environment can be described as a 210x160x3 image (it may vary, some environments have 250x160x3) or as a (128,) state vector representing the status of the RAM.
The RAM state vector represents the state of the game world at any time, such as the score, and the position of game objects, not including the screen memory. Each game has its own memory layout, each state vector is therefore game-specific.
Even with the RAM representation, the observation space and the action space of the Atari benchmark are significantly bigger and more complex. Furthermore, the Atari benchmark contains games with complex dynamics and intricate gameplays, where diverse strategies can be applied to succeed, with reward systems that require long-term planning. Gaier and Ha [4] also included the possibility of finetuning the evolved networks. Of the different algorithms they provide, we carried out the finetuning process with the gradient-free CMA-ES (mentioned in Section 3.3) as a step successive to the evolution of the network.

All the experiments have been carried out on the computing cluster GRACE of the ADA research group. This cluster can be used by the members of the group and consists of 34 nodes, of which 24 are CPU nodes (with 2 CPUs each), and 8 of them are GPU nodes (with 2 or 4 GPUs each depending on the node). Each CPU is an Intel(R) Xeon(R) CPU E5-2683 v4 @ 2.10GHz, while each GPU is an NVIDIA Corporation GP102 GeForce GTX 1080 Ti. The gradient-based methods with image representation have been run on GPU nodes, while the rest of the experiments have been carried out on CPU nodes.
Taking into account the availability of the shared cluster and considering the work on the Atari-57 benchmark carried out by Aitchison et al. [6], we decided to use the Atari-5 benchmark in this work. This benchmark includes the games Battle Zone, Double Dunk, Name This Game, Phoenix and Q*Bert.

5.1 Hyperparameter Importance Analysis

As mentioned in Section 4.1.1, Gaier and Ha [4] in their repository[3] provided defaults and specific modifications for each of the tasks explored by the authors.
Taking inspiration from the hyperparameter importance across datasets analysis proposed by van Rijn and Hutter [22], we carry out a hyperparameter importance analysis applying the functional analysis of variance (fANOVA) framework [51]. To do so, we used the fANOVA package introduced by Hutter et al. [21].
The analysis we carry out in this work aims to assess the relative importance of each hyperparameter in relation to the variation observed in the performance for each game in the benchmark and how the hyperparameters rank by importance among the games.
The fANOVA package introduced by Hutter et al. [21] can measure across the considered configuration space how much each hyperparameter explains the variance of the performance, which is predicted with an Empirical Performance Model (EPM) based on a random forest. This yields how much every single hyperparameter or combination of hyperparameters explains the variance of the performance [21, 22].
This framework depends on the concept of marginal of a hyperparameter, how a given value for a hyperparameter performs as part of a configuration, taking the average over all possible values of the remaining hyperparameters [23].
The default hyperparameters of the WANN process are described in Table 4 in Appendix A. Focusing on the hyperparameters responsible for leading the architecture search, as previously mentioned, we define a configuration space [52] as shown in Table 1:

Hyperparameter	**Values**	**Description**
prob_initEnable	[0.0,0.55]	chance to enable each initial connection
prob_mutAct	[0.0,0.55]	chance to change node activation function
prob_addNode	[0.0,0.55]	chance to add node
prob_addConn	[0.0,0.55]	chance to add connections
prob_enable	[0.0,0.55]	chance to enable disabled connection

Table 1: Hyperparameters responsible for leading the architecture search and the ranges from which the test configurations have been sampled for each of the hyperparameters. Each hyperparameter is sampled from a uniform distribution.

We sample 30 configurations from this configuration space and test each configuration on each game. Each hyperparameter is sampled from a uniform distribution. Due to time constraints, we were not able to sample more configurations and therefore test more configurations. The WANNs have been tested with these 30 configurations on every game for 10% of the available budget. Therefore, since the total budget amounts to 10 million steps, each configuration has been tested for 1 million steps. In doing so, we can explore how the WANNs perform with each configuration performs on the allocated budget, and how the prediction of the performances can vary for each game given the different values of each hyperparameter.
While the hyperparameters included in the configuration space were allowed to vary within the ranges shown in Table 1, other hyperparameters were fixed to guarantee the correct functioning of the process.
In Section 6, we will analyze the marginals of the hyperparameters in the configuration space, namely the probability of adding a new connection, the probability of adding a new node, the

[3] `https://github.com/google/brain-tokyo-workshop/blob/master/WANNRelease/`

probability of enabling a disabled connection, the probability of enabling each initial connection, and the probability of mutating activation function.

5.2 Weight Agnostic Neural Networks

We select the configurations that performed better during the runs over 1M timesteps for each game to carry out the full experiments on the WANNs. We use these configurations to perform complete evolutionary processes over each game with each representation. We set a budget of 10 million timesteps, the same amount of timesteps performed in the experiments of the gradient-based approaches [25, 26, 53].

We used timesteps as a budget to compare the performance of the gradient-based approaches and the WANNs. In spite of the differences in the operations these approaches perform, such as gradient updates and mutations, we can still compare them based on the timesteps each approach executes. Using timesteps as a budget also gives us an idea of how much each approach needs to interact with the environment to learn.

Since the gradient-based approaches apply some preprocessing to the Atari environments, the same preprocessing has been applied to the WANNs. In Section 4.1.2, a more in-depth explanation is provided of how images are treated in the image version of the experiments. Even if it is technically possible, it is not recommended to take images as input for the WANNs directly. The network would create one input neuron for each pixel, creating massive networks from the beginning and making the evolution process require a high amount of memory.

Given the available hardware with 32 cores, the population size of these experiments has been set to 32, which we also believe it to be a reasonable setting while keeping our limited computation budget in mind. A higher number of elements in the population could have provided a higher degree of diversity, potentially leading to a higher degree of exploration and enlarging the portion of solution space explored. However, given the predefined budget, this would lead to a lower number of generations.

We test every individual with six possible shared-weight values $[-2, -1, -0.5, 0.5, 1, 2]$ since the authors realized that, likely due to saturation of the activation functions, the networks with a shared-weight value greater than 2 performed similarly, while the values included in the set show the highest variance in performances of the networks. The individual will play the game and achieve a reward with each of these shared-weight values. The performance of the individual is then defined as the cumulative reward achieved with each shared-weight value, averaged over the number of values.

For instance, if an individual plays with the shared-weight values $[-2, -1, -0.5, 0.5, 1, 2]$ and achieves the rewards $[600, 100, 300, 400, 200, 500]$, its average cumulative reward will be $(600+100+300+400+200+500)/6 = 350$. At the end of each generation, we report the three metrics also reported by Gaier and Ha [4] in their code: the elite fit, the best fit, and the peak fit, previously mentioned in Section 4.1. In the WANNs graphs, together with the elite fit, also its smoothed function will be shown to facilitate the visualization.

These three metrics will lead the evolutionary process. As previously mentioned, we will also train and test three gradient-based methods (DQN, A2C, PPO) on the RAM and image representation of each game.

5.3 Gradient-based Agents

For the experiments involving the gradient-based agents, the hyperparameter configurations listed in their respective papers [25, 27, 26, 53] were followed. In the original papers, A2C and PPO used the CNN architecture presented by Mnih et al. [25].
This architecture consists of a first convolutional hidden layer with 16 filters of size 8 with stride 4, a second convolutional hidden layer with 32 filters of size 4 with stride 2, a third fully-connected hidden layer consisting of 256 rectifier linear units, and a fully-connected output layer with a single output for each valid action.
For DQN, we used the architecture proposed by Mnih et al. [53] since their original paper contains a wider overview of results for the DQN agent.
This architecture consists of three convolutional hidden layers and one fully-connected hidden layer, where the first one convolves 32 filters of size 8 and stride 4, the second one convolves 64 filters of size 4 and stride 2, the third one convolves 64 filters of size 3 and stride 1 and the fully-connected hidden layer has 512 output units, followed by the fully-connected output layer with one output unit for each valid action.
For the experiments regarding the RAM representation, only the fully-connected layers of the previous architectures have been used.
Hence, for the DQN agent with RAM representation, we have one hidden layer with 512 rectifier linear units, followed by a fully-connected output layer with one output unit for each valid action. For the PPO and A2C agents, there is one fully-connected hidden layer with 256 rectifier linear units, followed by the fully-connected output layer with one output unit for each valid action.
For DQN, the network architecture is the same between the Q-Network and the target Q-Network, while for PPO and A2C, both using the ActorCriticPolicy from stable-baselines3 [5], the network architecture is the same for the policy-net and the value-net. All the gradient-based approaches have been tested using the implementation provided by Raffin et al. [5].

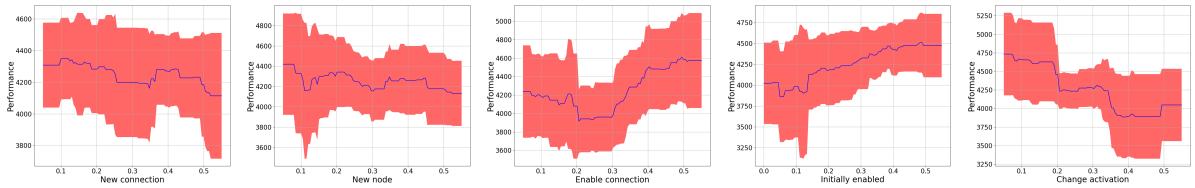
6 Results

In this section, we discuss the results of our experiments. First, we will analyze the results of the analysis carried out to identify the most important hyperparameters. With this analysis, we aim to answer the research subquestion **Q1**. Secondly, we present the results of the WANNs and the gradient-based agents over the Atari-5 benchmark. We first present the results achieved with the RAM representation in Section 6.2, then we present the results achieved with the image representation in Section 6.3. For both the RAM and the image representation, the plots containing the training curves of each agent for each game will be shown in their specific sections. For each representation, we first analyze the results achieved by the WANNs at the end of the evolution process, together with the results of the gradient-based agents. Analyzing the results of these experiments, we aim to answer the research subquestions **Q2** and **Q3**.
Afterwards, we provide the results of the finetuned WANNs compared with the gradient-based agents with each representation. Observing the results of the finetuned WANNs, we aim to answer the research subquestion **Q4**. In Sections 6.2 and 6.3, we provide tables summarizing the comparison for each representation between the evolved WANNs, the finetuned WANNs, and the gradient-based approaches.

The code to reproduce the experiments is available on GitHub.[4]

6.1 Hyperparameter Importance Analysis

Using the functional ANOVA package introduced by Hutter et al. [21], we can generate and visualize the predicted performance marginals of every hyperparameter for every game. On the x-axis, we will show the range of values the hyperparameter can take within the configuration space. On the y-axis, we plot the predicted marginal performance (representation of the reward of that configuration over that game).
In Figure 2, we show the marginal performance predictions of each hyperparameter in the configuration space for Battle Zone. In Figure 3, we show the marginal performance predictions of each hyperparameter over Double Dunk, which shows an unexpected trend due to a specific behavior assumed by the WANNs. The figures of the games Phoenix, Name This Game, and Q*Bert (Figures 20,19,21) can be found in Appendix A.2.

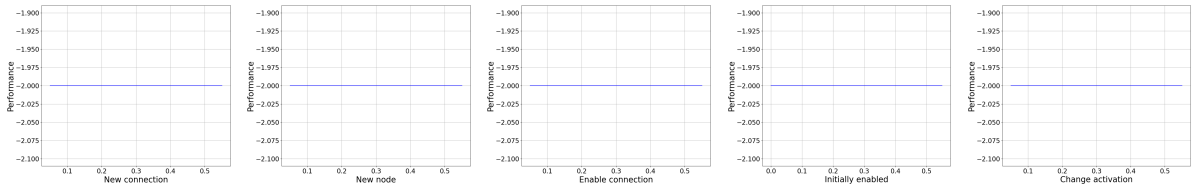


(a) (b) (c) (d) (e)

Figure 2: Marginal performance prediction for the hyperparameters of the configuration space for the Battle Zone game. The red area represents the standard deviation, while the blue line represents the marginal.

These plots are helpful in understanding trends in the interaction between the hyperparameter values and the performances [23]. For the probability of adding a new node, plotted in Figure 2a, we notice that a higher probability of adding a new connection leads to slightly worse predicted performances. The same trend arises in the plot of the probability of mutating an activation function, plotted in Figure 2e, which predicts worse performances when the hyperparameter value is higher. We notice a different trend in the plot of the probability of enabling each initial connection, showing higher predicted performances with higher probabilities. The plots of the marginal predictions of the hyperparameters over the game Double Dunk, shown in Figure 3, show an unexpected trend which will be explained more in detail in Section 6.2.
Given the differences in the trends across the different games, it is interesting to analyze the variance contribution of each hyperparameter across the benchmark, which we show in Figure 4.
We notice that the probability of mutating is generally more influential than the probability of enabling a connection. The three leading mutation operators are indeed the three most influential hyperparameters. The probability of enabling a connection has two outliers, and the distribution is positively skewed since the median is closer to the bottom of the interquartile range. The probability of enabling each initial connection has an average higher than the median, with a fourth quartile wider than the first quartile. The probability of adding a new node has a relatively homogeneous variance contribution since the whiskers have almost the same length. The probability of adding a new connection shows the presence of an outlier.

[4] `https://github.com/moneantonio/wannthesis`



(a) (b) (c) (d) (e)

Figure 3: Marginal Performance Prediction for the hyperparameters of the configuration space for the Double Dunk game. The blue line represents the marginal, and since the WANN over the Double Dunk game exhibited an unexpected behavior explained in Section 6.2, there is no standard deviation.

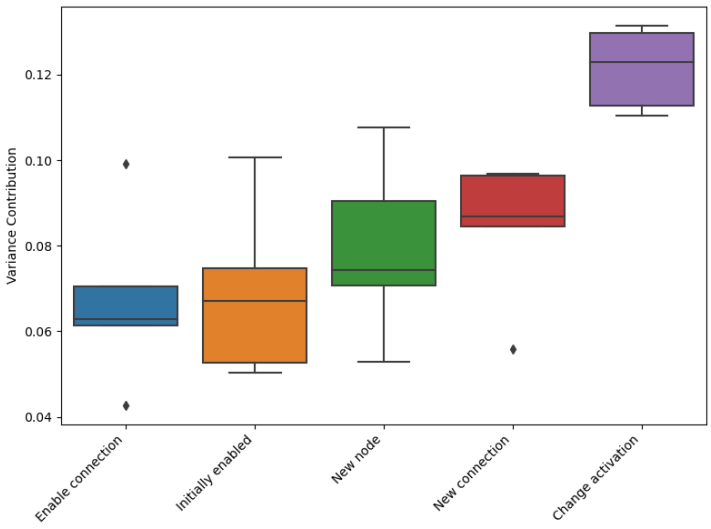


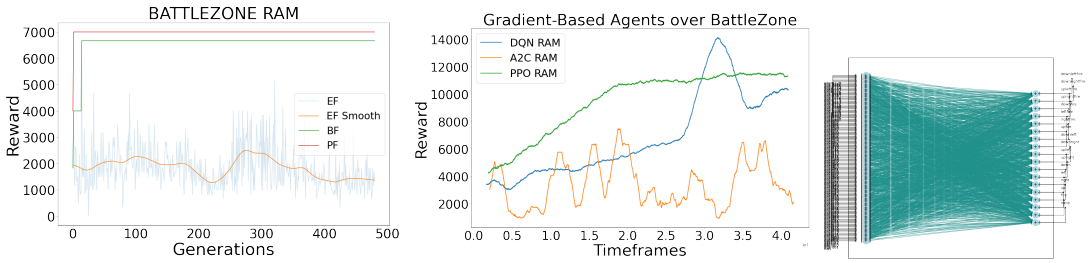
Figure 4: Boxplot of the variance contribution per hyperparameter across games

The probability of changing the activation function shows a higher variance contribution. It also shows whiskers of the same length and a median almost centered in the box, meaning that the distribution is similar to a normal distribution.
In the default configuration provided by Gaier and Ha [4], reported in Table 4 shown in Appendix A, the authors give a slightly higher probability to the creation of a new node (0.25) rather than to the establishment of a new connection (0.20). In contrast, they give the mutation of the activation function the highest probability (0.50). The probabilities given to each possible mutation will bias the search towards different kinds of modifications in the network architectures. Our analysis highlights the mutation of the activation function as the most influential hyperparameter, answering the first research subquestion **Q1**. Following the mutation of the activation function, we find the establishment of a new connection and the addition of a new node as the next most important hyperparameters.

6.2 RAM Representation

The monitor implemented by Raffin et al. [5] was used to visualize the training process of the gradient-based approaches. The preprocessing for these agents includes frame-skipping every 4 frames. As a result, the agent's training is plotted over 40 million frames, equivalent to 10

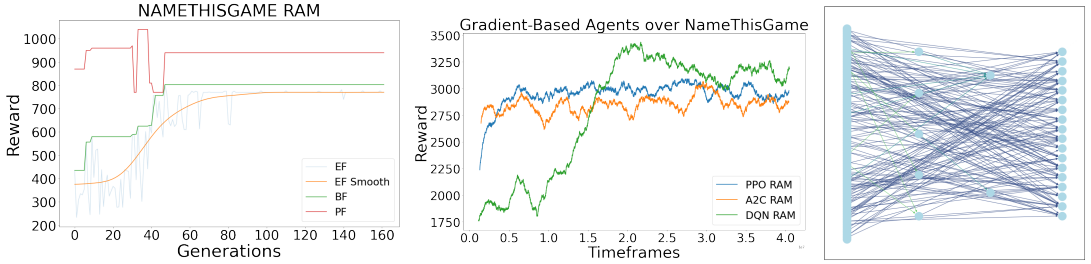
million timesteps with frame-skipping of 4 frames. The evolutions of the WANNs, on the other hand, are plotted over the generations they were able to complete within the allocated budget. We now look at the comparison between the evolutionary process and the gradient-based agents' training using the RAM representation.



(a) (b) (c)

Figure 5: In Figure 5a, the evolution process of the WANN over the Battle Zone game with the RAM representation is shown, reporting the three values previously mentioned: Elite Fit, Best Fit, and Peak Fit. In Figure 8b, the training process of the three gradient-based agents is shown over a span of 40M timeframes (which are roughly 10M timesteps with a frameskip value of 4). The connectome representing the network generated during the evolution is shown in Figure 5c

We first observe the evolution of the Battle Zone game shown in Figure 5a. We notice how the highest average score is achieved almost immediately, blocking the evolution in a local optimum. The rest of the generations were not able to improve from that, as shown in the elite fit smoothed curve, which ends with roughly the same score as it started, if not slightly worse. The connectome shown in Figure 5c shows that no new nodes were added to the network, even if the evolution process over Battle Zone was carried on for almost 500 generations.

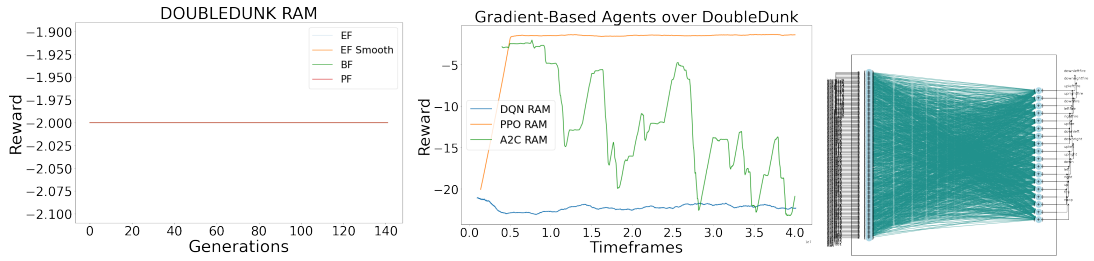


(a) (b) (c)

Figure 6: In Figure 6a, the evolution process of the WANN over the "Name This Game" game with the RAM representation is shown, reporting elite fit, best fit, and peak fit. In Figure 6b, the training process of the three gradient-based agents is shown over a span of 40M timeframes (which are roughly 10M timesteps with a frameskip value of 4). The connectome representing the network generated during the evolution is shown in Figure 6c.

We now observe the connectome and the evolution plot of the game Name This Game (Figures 6c,6a). A sparse but slightly more complex network has evolved after 160 generations, with only two minimal hidden layers, with 5 and 2 nodes each. We notice how the elite fit rewards have improved more steadily over the generations. At the same time, the peak fit is initially

unstable but stabilized afterward throughout the evolution, while the elites and the bests improve. This behavior suggests that while one of the shared-weight values performed well from the beginning, individuals learned to perform better with the other shared-weight values throughout the generations. In doing so, the individual improved the behavior with the other shared-weight values and got closer to the value that first led to the high score, converging to a local optimum after roughly 90 generations.
It is helpful to observe the plotted results and focus on the performances of the gradient-based approaches.
Using the best fit as a frame of reference for the WANNs, it would seem that the WANN performed better than the A2C RAM agent over Battle Zone, which shows an irregular curve with difficulties in learning from that specific representation. However, since the best fit represents the highest average reward reached over the entire evolutionary process, it does not represent a reliable and constant performance basis. Instead, it seems reasonable to use the elite fit as a frame of reference for the WANNs, since it represents the highest average reward reached at every single generation compared to the gradient-based agents. Considering this, we see how the gradient-based methods outperform the WANNs in most games, not considering the particular case of Double Dunk. Over Phoenix, the DQN RAM agent performed as poorly as the WANNs, and the same happened for the A2C RAM agent over Q*bert, where both approaches got stuck in similar behaviors.
A particular case is the Double Dunk game because, looking at the simple performance value at the end of the training, one would think that the WANN performed surprisingly well. However, some interesting behavior arises after observing the plots shown in Figure 7, the rendering of the agent playing and looking at the fitness values of the first generation of individuals.



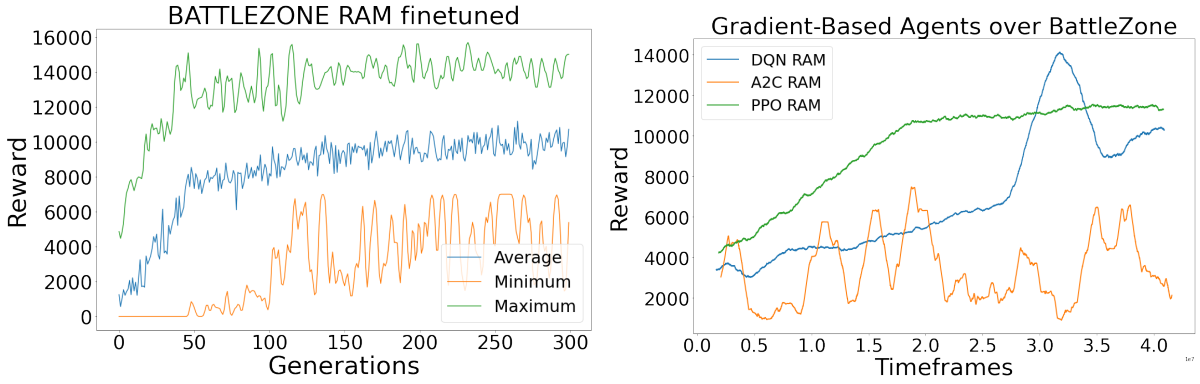
(a) (b) (c)

Figure 7: In Figure 7a, the evolution process of the WANN over the Double Dunk game with the RAM representation is shown, reporting elite fit, best fit, and peak fit. In Figure 7b, the training process of the three gradient-based agents is shown over a span of 40M timeframes. The connectome representing the network generated during the evolution is shown in Figure 7c.

According to the game description provided on the gym website, "at the beginning of each possession, the agent selects between a set of different plays and then executes them to either score or prevent the rivals from scoring. The game lasts a set amount of time or until one of the teams reaches a certain score" [54].[5]
In the first generation, a small number of individuals with some of the shared-weight values reached poor results as expected, reaching scores around the -18 and -20 marks. However, most individuals performed a fixed -2 for every shared-weight value. After observing the rendering

[5] https://www.gymlibrary.dev/environments/atari/double_dunk/

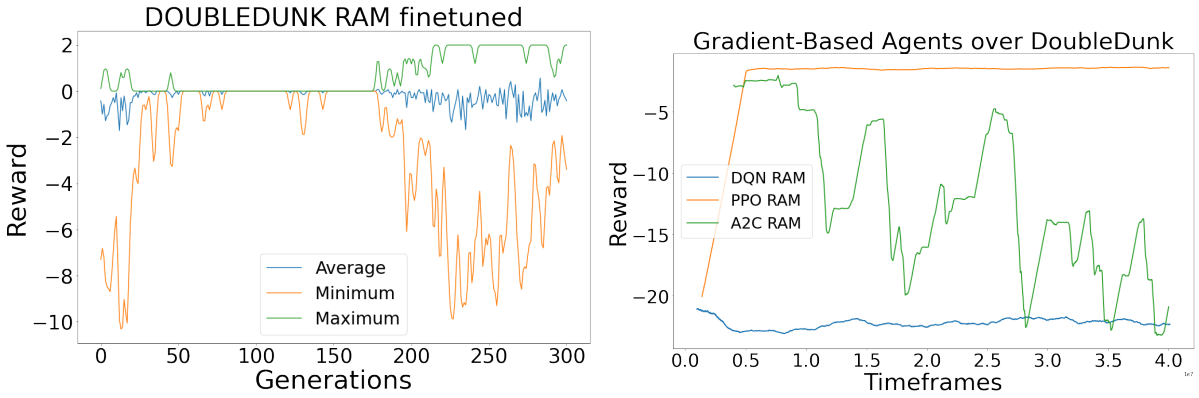
of the actual agent playing, it is fascinating to notice a specific behavior arose after the first possession in which the opponent scores. The agent can now choose between the offense and defense sets of action.
However, it alternates without picking one until the end of the available timesteps. Since at the end of each generation, the "best performing" ones are chosen accordingly to their performance value, this behavior is carried out throughout the whole evolutionary process. The evolution process over Double Dunk only evolved for roughly 140 generations (Figure 7a), while its network did not show any sign of evolution, as seen in Figure 7c.
All the plots depicting the evolution processes with the RAM representation before the application of the finetuning process and the plots describing the training of the gradient-based approaches with that representation over each game are reported in Appendix A.3 together with the connectomes of the networks evolved during the evolutionary process (Figure 27). We present the evaluation performances using the RAM representation of the evolved WANNs against the gradient-based approaches and the finetuned WANNs in Section 6.2.
As mentioned in the original paper [4], the project includes the possibility of finetuning the networks generated by the evolutionary process. We do this using the CMA-ES presented in Section 3.3. In the code provided by Gaier and Ha [4], the number of generations needed to have performances close to the best performances for the environments at hand was set at 150. After this number of generations, Gaier and Ha [4] affirm that one could simply kill the process since a termination criteria was not defined. In this work, the finetuning process of the RAM representation networks lasted 300 generations (which lasted approximately 10 minutes for each network). For time reasons, the finetuning process of the image representation networks lasted 150 generations (which lasted approximately 8 hours for each network).



(a) (b)

Figure 8: In Figure 8a, the finetuning process of the WANN over the Battle Zone game with the RAM representation is shown, reporting the minimum reward reached, the maximum reward reached, and the average reward reached with CMA-ES. In Figure 8b, the training process of the three gradient-based agents is shown over a span of 40M timeframes (which are roughly 10M timesteps with a frameskip value of 4).

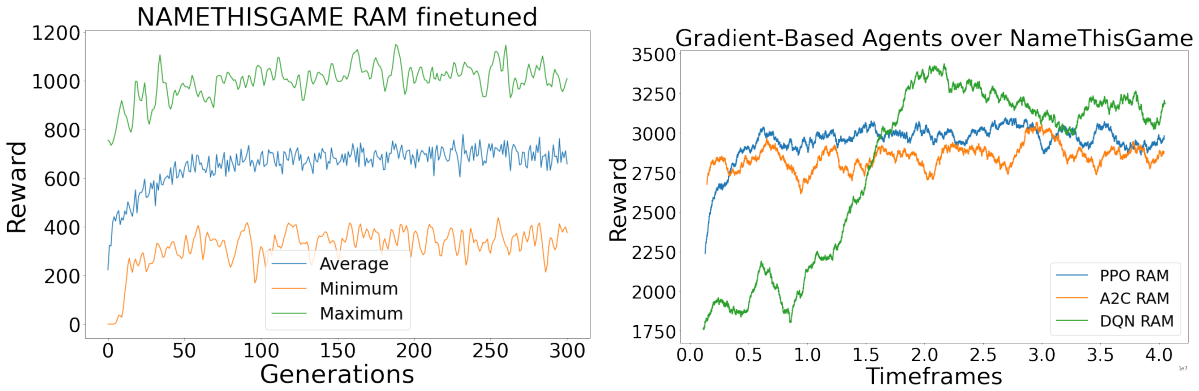
Observing the Battle Zone plot, as shown in Figure 8a, it is impressive to notice how the average reward of that network, previously averaging approximately 2000 points, at the end of the process reaches an average reward of approximately 11000 points, achieving performances comparable with the gradient-based approaches on the same game with the RAM representation.



(a) (b)

Figure 9: In Figure 9a, the finetuning process of the WANN over the Double Dunk game with the RAM representation is shown, reporting the minimum reward reached, the maximum reward reached, and the average reward reached with CMA-ES. In Figure 9b, the training process of the three gradient-based agents is shown over a span of 40M timeframes (which are roughly 10M timesteps with a frameskip value of 4).

Observing the Double Dunk plot, shown in Figure 9a, which showed a wrong behavior before finetuning as shown in Figure 7a, after finetuning, the reward was stuck in a local optimum until the first half of the allocated generations. Afterward, after roughly 180 generations, it was able to stray away from the local optimum, reaching lower minimums and slightly higher maximums, suggesting that it started exploring different behaviors.



(a) (b)

Figure 10: In Figure 10a, the finetuning process of the WANN over the "Name This Game" game with the RAM representation is shown, reporting the minimum reward reached, the maximum reward reached, and the average reward reached. In Figure 10b, the training process of the three gradient-based agents is shown over a span of 40M timeframes (which are roughly 10M timesteps with a frameskip value of 4).

The plot of Name This Game is shown in Figure 10. All three curves show a more pronounced improvement in the first 50 generations, while the average reward curve had a slight improvement until the 80th generation. All three curves then converged and did not get closer to the score reached by the gradient-based approaches shown in Figure 10b.
All the plots of the finetuning process with RAM representation, paired with the plots of the gradient-based approaches on the same games with that representation, are also shown in

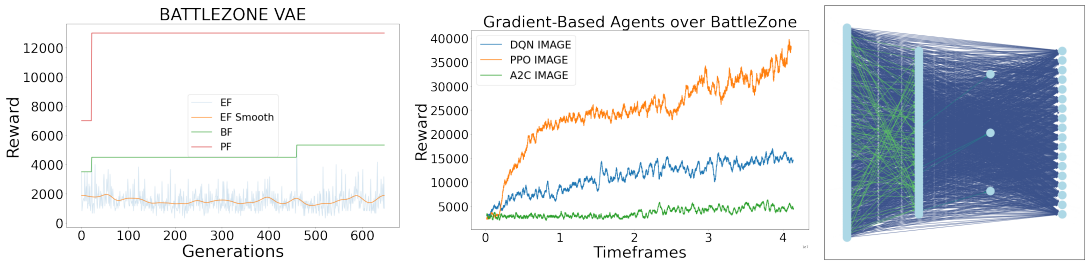
Appendix A.5.1.
In Table 2 we present the comparison of the rewards achieved by the evolved WANNs ("Evolved"), the finetuned WANNs ("Finetuned"), and the gradient-based agents (DQN, A2C, PPO) using the RAM representation:

Game	**Evolved**	**Finetuned**	**DQN**	**A2C**	**PPO**
Battle Zone	416.66	11200.00	11384.00	3019.00	12030.00
Double Dunk	-2^*	0.80^*	-21.64	-15.50	-1.31
Name This Game	591.00	778.00	3257.70	2801.78	2954.26
Phoenix	163.50	713.05	378.80	519.45	908.74
Q*bert	110.83	577.50	3598.47	239.52	1207.35

Table 2: Table reporting the reward reached by the WANNs before finetuning (Evolved), the WANNs after finetuning (Finetuned), and the gradient-based approaches on each game with the RAM representation averaged over 10 evaluation runs. For the evolved WANNs, the presented scores are the average over the 6 different shared-weight values. This means that each individual plays 10 times with each shared-weight value for a total of 60 runs.

Looking at the comparison of the reward achieved by the evolutionary process before finetuning and the gradient-based agents over 10 evaluation runs, we notice how the gradient-based approaches perform better in almost every game, answering the research subquestion **Q2**. Considering instead the performances reached by the WANNs after finetuning, we notice the impact of the finetuning in comparison with the scores of the evolved WANNs, where the finetuned WANNs showed an increase in performance on every game, answering the research subquestion **Q4** over the RAM representation. We notice the most significant increase over the Battle Zone game, where the finetuned WANN performs comparably with DQN and PPO, outperforming A2C. Over Phoenix, the finetuned WANN outperforms DQN and A2C. Over Q*bert, the finetuned WANN outperforms A2C but is still strongly outperformed by DQN and PPO.

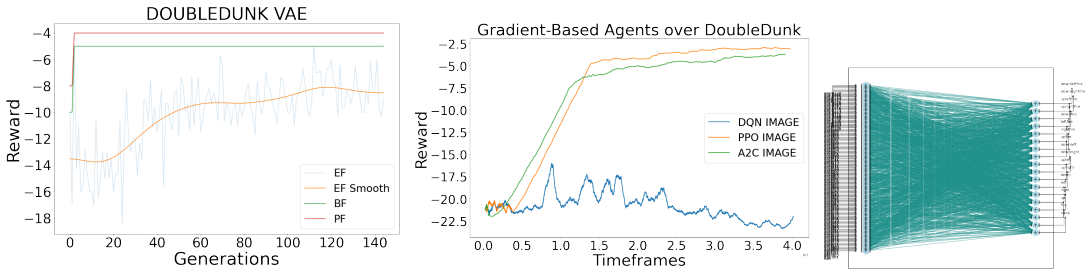
6.3 Image Representation



(a) (b) (c)

Figure 11: In Figure 11a, the evolution process of the WANN over the Battle Zone game with the image representation is shown, reporting elite fit, best fit, and peak fit. In Figure 11b, the training process of the three gradient-based agents is shown over a span of 40M timeframes (which are roughly 10M timesteps with a frameskip value of 4). The connectome representing the network generated during the evolution is shown in Figure 11c.

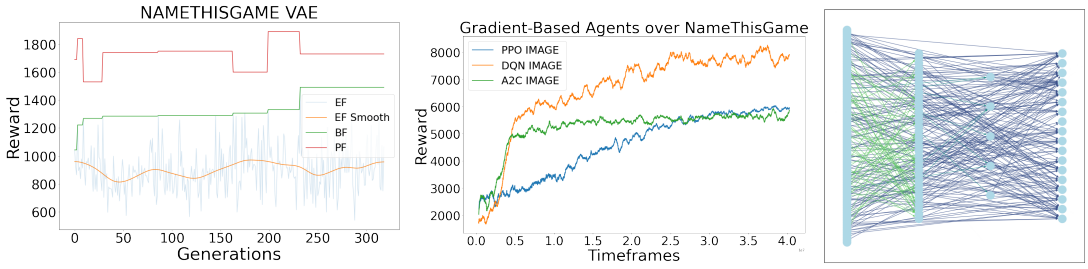
The evolution of the WANN, the training of the gradient-based approaches, and the connectome of the generated network are shown in Figure 11.
It is noticeable how the image representation, even if it shows a lower best fit, has a sensibly higher peak fit. The peak fit was achieved almost immediately, as it happened during the evolution using the RAM representation.
Furthermore, considering that the results of the gradient-based approaches align with the previous publications, with the PPO image agent performing even better, it is intriguing to point out that A2C yields only a slight improvement over the best fit of the WANN.
While the connectome of the RAM representation over Battle Zone showed no added nodes or layers, the connectome created by the process using the image representation (shown in Figure 11c) showed a first sign of evolution. It has a first sparse layer with 80 nodes and a second layer with only three nodes.



(a) (b) (c)

Figure 12: In Figure 12a, the evolution process of the WANN over the Double Dunk game with the image representation is shown, reporting elite fit, best fit, and peak fit. In Figure 12b, the training process of the three gradient-based agents is shown over a span of 40M timeframes (which are roughly 10M timesteps with a frameskip value of 4). The connectome representing the network generated during the evolution is shown in Figure 12c.

The comparison between the RAM and image representations over Double Dunk (presented in Figure 12) shows some intriguing results.
Unlike the RAM representation, the image version, probably aided by the spatial information provided by the VAE encoding and missing in the RAM representation, actually had performance improvements, starting from bad scores, as expected.
Observing the game's rendering, we can see how the agent plays and does not get stuck in the "waiting for the timesteps to end" behavior previously mentioned.
However, it is worth noting that at the end of the process, it is impossible to view how the agent acted when it yielded good rewards during the evolutionary process. It could be possible that while some of the weight-shared values led the agent actually to play the game, other ones still pushed for that kind of behavior, leading to what can be mistakenly interpreted as averaged good results, like the peak fit almost immediately reached.
The Double Dunk connectomes are instead identical, suggesting that the high average scores reached going further through the evolutionary process of the image version might also be influenced, at least partly, by a group of individuals pushing for the "wait for the timesteps to end" behavior.

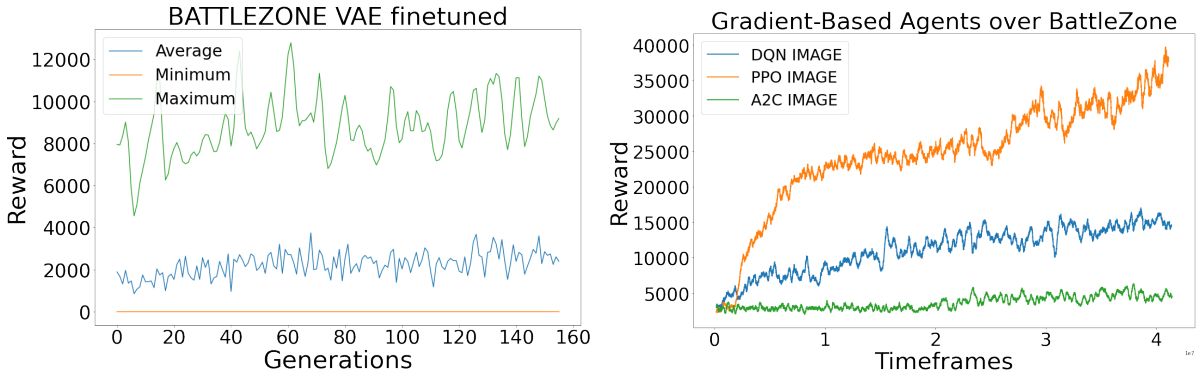


(a) (b) (c)

Figure 13: In Figure 13a, the evolution process of the WANN over the "Name This Game" game with the image representation is shown, reporting elite fit, best fit, and peak fit. In Figure 13b, the training process of the three gradient-based agents is shown over a span of 40M timeframes (which are roughly 10M timesteps with a frameskip value of 4). The connectome representing the network generated during the evolution is shown in Figure 13c.

The difference between the Name This Game plots (shown in Figure 13), instead, show a way higher peak fit in the image representation version, around 1750 points, while the ram version had a peak fit of approximately 950. While the RAM version got stuck at a local optimum pretty fast, the image version shows an unstable curve of the elite fit. We believe the exploration process was still active and did not get stuck in a local optimum as it happened for the RAM version.
Furthermore, the best fit value reached by the image version is even higher than the peak reached by the RAM version, showing the importance of spatial information in such a game.
The network generated by the image representation of Name This Game (shown in Figure 13c) has the same number of layers, 2. Still, it has a higher number of nodes, which might be the reason for the increased performance of the image version compared to the RAM version.
All the plots describing the evolution processes with the image representation before the application of the finetuning process, and the plots describing the training of the gradient-based approaches with that representation over each game are reported in the Appendix A.4 together with the connectomes of the networks evolved during the evolutionary process (Figure

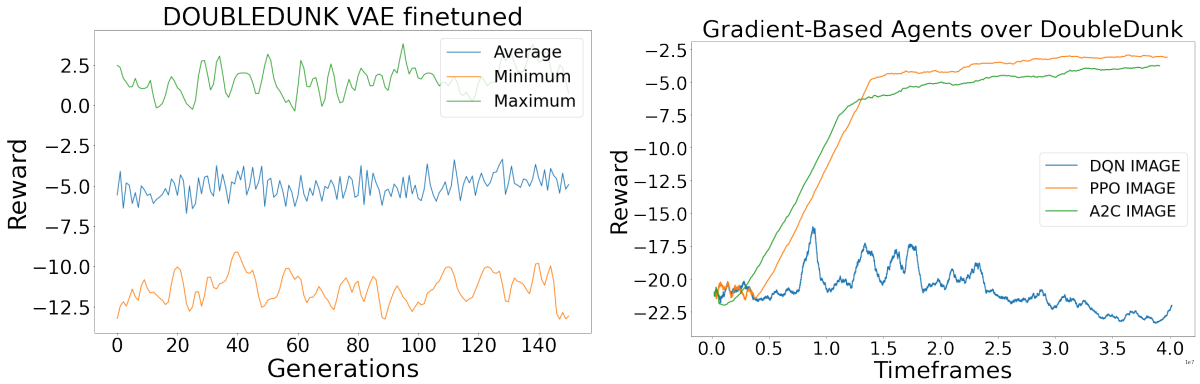
33). We present the evaluation performances using the image representation of the evolved WANNs against the gradient-based approaches and the finetuned WANNs in Section 6.3.
We now observe the results of the finetuning process over the networks generated while playing the games using the image representation.



(a) (b)

Figure 14: In Figure 14a, the finetuning process of the WANN over the Battle Zone game with the image representation is shown, reporting the minimum reward reached, the maximum reward reached, and the average reward reached. In Figure 14b, the training process of the three gradient-based agents is shown over a span of 40M timeframes (which are roughly 10M timesteps with a frameskip value of 4).

While the finetuning process yielded very positive rewards over the game Battle Zone with the RAM representation, it did not show the same improvement over the image representation, as shown in Figure 14.
After the application of the finetuning process, the maximum and average rewards did not change over the generations, confirming the gradient-based approaches as the best performers.

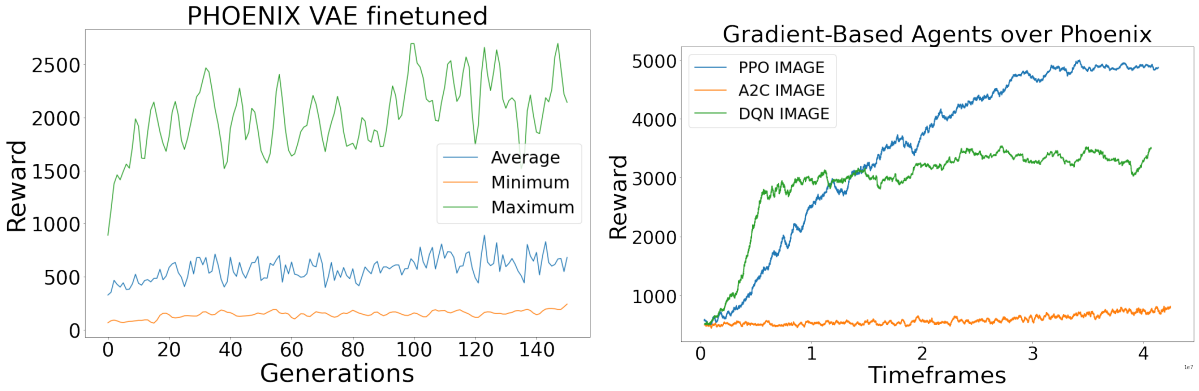


(a) (b)

Figure 15: In Figure 15a, the finetuning process of the WANN over the Double Dunk game with the image representation is shown, reporting the minimum reward reached, the maximum reward reached, and the average reward reached. In Figure 15b, the training process of the three gradient-based agents is shown over a span of 40M timeframes (which are roughly 10M timesteps with a frameskip value of 4).

Before the finetuning process, Double Dunk with the image representation yielded theoretically

good results, as shown in Figure 12. Still, given the behavior shown by the RAM representation, it is not possible to confirm the fact that the high scores of the image representation were also yielded by that same behavior that influenced the average score.
During the finetuning process, shown in Figure 15, there was no noticeable improvement, with irregular curves around the same mean from the beginning to the end of the process.



(a) (b)

Figure 16: In Figure 16a, the finetuning process of the WANN over the game Phoenix with the image representation is shown, reporting the minimum reward reached, the maximum reward reached, and the average reward reached. In Figure 16b, the training process of the three gradient-based agents is shown over a span of 40M timeframes (which are roughly 10M timesteps with a frameskip value of 4).

Over the game Phoenix, on the other hand, which showed an average of approximately 100 points during the evolution using the image representation, the finetuning process yielded some improvements, as shown in Figure 16a.
Already from the first generation, it showed an average reward of 350 points, even higher than the best fit reached during the evolution, reaching an average of 580 points by the end of the process.
The peak performance reached during the evolution has been surpassed, reaching an average of 2000 points. While PPO and DQN performed quite well in this game, A2C showed similar performances.
In Table 3 we present the comparison of the rewards achieved by the evolved WANNs ("Evolved"), the finetuned WANNs ("Finetuned"), and the gradient-based agents (DQN, A2C, PPO) using the image representation:

Game	Evolved	Finetuned	DQN	A2C	PPO
Battle Zone	683.33	3400.00	14438.00	4459.00	38093.00
Double Dunk	-14.83^*	-1.80^*	-22.07	-3.75	-3.07
Name This Game	569.44	624.00	7885.19	5828.33	5924.07
Phoenix	96.67	795.00	3480.64	853.15	4973.16
Q*bert	366.66	380.00	10354.40	7666.02	14111.07

Table 3: Table reporting the reward reached by the WANNs before finetuning (Evolved), the WANNs after finetuning (Finetuned), and the gradient-based approaches on each game with the image representation averaged over 10 evaluation runs. For the evolved WANNs, the presented scores are the average over the 6 different shared-weight values. This means that each individual plays 10 times with each shared-weight value for a total of 60 runs.

Looking at the comparison of the reward achieved by the evolutionary process before finetuning and the gradient-based agents over 10 evaluation runs, we can see that the gradient-based approaches outperformed the WANNs before and after the finetuning. Over the Battle Zone game, the WANN resulting after the finetuning gets closer to the score reached by A2C but is still outperformed by all the gradient-based approaches. Over Phoenix, the resulting WANN gets closer to the score achieved by A2C but is still outperformed by all the other gradient-based agents.
We noticed some interesting improvements with the finetuning of the evolved WANNs using the RAM representation, presented in Section 6.2. Over the image representation, we do not see a comparable enhancement in performance, answering the research subquestion **Q4** over the image representation.
Observing the results achieved with the image representation, it seems clear how the WANNs are not yet ready to be compared with the gradient-based approaches over a complex task such as the Atari benchmark, answering the research subquestion **Q3**.
Observing the results achieved with the RAM representation we see how the finetuning performed with CMA-ES had a significant impact on the evolved WANNs when using that representation, such that over the Battle Zone and Phoenix games the finetuned WANNs performed comparably with DQN and PPO, outperforming A2C.
The difference in impact of the finetuning process over the RAM and image representation can be related to the performance bottleneck of the VAE. The reconstruction quality of the VAE degrades gradually with the advancement in the levels of the games, making the encoding of the images less accurate. The status of the RAM instead, even if it lacks the visual information provided by the screen images, does not get degrade while advancing in the games.
All the plots of the finetuning process with image representation, paired with the plots of the gradient-based approaches on the same games with that representation, are shown in Appendix A.5.2.

7 Conclusions & Future Works

In this work, we conducted experiments on the Atari benchmark, which was not explored in the Weight Agnostic Neural Networks original paper. We compared the performance of the WANNs with three gradient-based agents, DQN, PPO, and A2C, over the RAM and the image representations of the observation space. We also conducted a hyperparameter importance

analysis to identify the most influential hyperparameters across games and the values of these hyperparameters that yield higher predicted performances for each game.
Our aim was to determine up to which level of performance and with which limitations we can consider the Weight Agnostic Neural Networks as a competitive alternative to SGD-based algorithms for more complex environments such as the Atari benchmark.
The hyperparameter importance analysis showed that the probability of mutating an activation function is the most important one for the evolution of the WANNs, followed by the probability of adding a new connection and the probability of adding a new node. Further developments in this area include an analysis with a higher number of configurations sampled from the configuration space and a higher number of hyperparameters included in the configuration space. Before finetuning, the gradient-based agents outperformed the evolved networks both with the RAM and the image representation. The RAM version of the WANNs exhibited an unexpected strategy over the Double Dunk game, where the agent would wait for the available timesteps to finish instead of playing the game.
The performances of the WANNs are poor compared to the gradient-based techniques over the image representation, indicating that they are not currently a competitive alternative to gradient-based approaches on the Atari benchmark. One of the main limitations of the WANNs is the potential bottleneck in performance caused by using a VAE due to how the training dataset was generated. Further developments regarding the VAE could involve the use of different agents to create the training datasets or exploring alternative ways to treat the image as input.
The finetuned WANN with the RAM representation over the game Battle Zone scored an average of 11200.00 points, outperforming A2C (3019.00 points) and performing comparably with DQN (11384.00 points) and PPO (12030.00 points), encouraging further future research. Applying the finetuning process to the networks generated using the image representation did not enhance the performances significantly. Further works include the application of the WANNs to different tasks that do not require the interpretation of the image input, such as the MuJoCo physics engine.

References

[1] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. P. Lillicrap, and D. Silver, "Mastering atari, go, chess and shogi by planning with a learned model," *Nat.*, vol. 588, no. 7839, pp. 604–609, 2020. [Online]. Available: https://doi.org/10.1038/s41586-020-03051-4

[2] A. P. Badia, B. Piot, S. Kapturowski, P. Sprechmann, A. Vitvitskyi, Z. D. Guo, and C. Blundell, "Agent57: Outperforming the atari human benchmark," in *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, ser. Proceedings of Machine Learning Research, vol. 119. PMLR, 2020, pp. 507–517. [Online]. Available: http://proceedings.mlr.press/v119/badia20a.html

[3] J. Fan and C. Xiao, "Generalized data distribution iteration," in *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA*, ser. Proceedings of Machine Learning Research, K. Chaudhuri, S. Jegelka, L. Song, C. Szepesvári, G. Niu, and S. Sabato, Eds., vol. 162. PMLR, 2022, pp. 6103–6184. [Online]. Available: https://proceedings.mlr.press/v162/fan22c.html

[4] A. Gaier and D. Ha, "Weight agnostic neural networks," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, December 8-14, 2019, Vancouver, BC, Canada*, H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, Eds., 2019, pp. 5365–5379. [Online]. Available: https://proceedings.neurips.cc/paper/2019/hash/e98741479a7b998f88b8f8c9f0b6b6f1-Abstract.html

[5] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *J. Mach. Learn. Res.*, vol. 22, pp. 268:1–268:8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html

[6] M. Aitchison, P. Sweetser, and M. Hutter, "Atari-5: Distilling the arcade learning environment down to five games," *CoRR*, vol. abs/2210.02019, 2022. [Online]. Available: https://doi.org/10.48550/arXiv.2210.02019

[7] S. A. Harp, T. Samad, and A. Guha, "Designing application-specific neural networks using the genetic algorithm," in *Advances in Neural Information Processing Systems 2, [NIPS Conference, Denver, Colorado, USA, November 27-30, 1989]*, D. S. Touretzky, Ed. Morgan Kaufmann, 1989, pp. 447–454. [Online]. Available: http://papers.nips.cc/paper/263-designing-application-specific-neural-networks-using-the-genetic-algorithm

[8] D. Dasgupta and D. R. McGregor, "Designing application-specific neural networks using the structured genetic algorithm," in *[Proceedings] COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks*. IEEE, 1992, pp. 87–96.

[9] V. Maniezzo, "Genetic evolution of the topology and weight distribution of neural networks," *IEEE Trans. Neural Networks*, vol. 5, no. 1, pp. 39–53, 1994. [Online]. Available: https://doi.org/10.1109/72.265959

[10] Y. Liu and X. Yao, "Towards designing neural network ensembles by evolution," in *Parallel Problem Solving from Nature - PPSN V, 5th International Conference, Amsterdam, The Netherlands, September 27-30, 1998, Proceedings*, ser. Lecture Notes in Computer Science, A. E. Eiben, T. Bäck, M. Schoenauer, and H. Schwefel, Eds., vol. 1498. Springer, 1998, pp. 623–632. [Online]. Available: https://doi.org/10.1007/BFb0056904

[11] S. Mizuta, T. Sato, D. M. Lao, M. Ikeda, and T. Shimizu, "Structure design of neural networks using genetic algorithms," *Complex Syst.*, vol. 13, no. 2, 2001. [Online]. Available: http://www.complex-systems.com/abstracts/v13_i02_a04.html

[12] K. Neklyudov, D. Molchanov, A. Ashukha, and D. P. Vetrov, "Variance networks: When expectation does not meet your expectations," in *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. [Online]. Available: https://openreview.net/forum?id=B1GAUs0cKQ

[13] V. Mullachery, A. Khera, and A. Husain, "Bayesian neural networks," *CoRR*, vol. abs/1801.07710, 2018. [Online]. Available: http://arxiv.org/abs/1801.07710

[14] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems 2, [NIPS Conference, Denver, Colorado, USA, November 27-30, 1989]*, D. S. Touretzky, Ed. Morgan Kaufmann, 1989, pp. 598–605. [Online]. Available: http://papers.nips.cc/paper/250-optimal-brain-damage

[15] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz, "Pruning convolutional neural networks for resource efficient inference," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. [Online]. Available: https://openreview.net/forum?id=SJGCiw5gl

[16] A. N. Kolmogorov, "Three approaches to the quantitative definition of information," *Problems of information transmission*, vol. 1, no. 1, pp. 1–7, 1965.

[17] J. Rissanen, "Modeling by shortest data description," *Autom.*, vol. 14, no. 5, pp. 465–471, 1978. [Online]. Available: https://doi.org/10.1016/0005-1098(78)90005-5

[18] S. J. Nowlan and G. E. Hinton, "Simplifying neural networks by soft weight-sharing," *Neural Comput.*, vol. 4, no. 4, pp. 473–493, 1992. [Online]. Available: https://doi.org/10.1162/neco.1992.4.4.473

[19] G. E. Hinton and D. van Camp, "Keeping the neural networks simple by minimizing the description length of the weights," in *Proceedings of the Sixth Annual ACM Conference on Computational Learning Theory, COLT 1993, Santa Cruz, CA, USA, July 26-28, 1993*, L. Pitt, Ed. ACM, 1993, pp. 5–13. [Online]. Available: https://doi.org/10.1145/168304.168306

[20] S. Seung, *Connectome: How the brain's wiring makes us who we are*. HMH, 2012.

[21] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "An efficient approach for assessing hyperparameter importance," in *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, ser. JMLR Workshop and Conference Proceedings, vol. 32. JMLR.org, 2014, pp. 754–762. [Online]. Available: http://proceedings.mlr.press/v32/hutter14.html

[22] J. N. van Rijn and F. Hutter, "Hyperparameter importance across datasets," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, KDD 2018, London, UK, August 19-23, 2018*, Y. Guo and F. Farooq, Eds. ACM, 2018, pp. 2367–2376. [Online]. Available: https://doi.org/10.1145/3219819.3220058

[23] A. Sharma, J. N. van Rijn, F. Hutter, and A. Müller, "Hyperparameter importance for image classification by residual neural networks," in *Discovery Science - 22nd International Conference, DS 2019, Split, Croatia, October 28-30, 2019, Proceedings*, ser. Lecture Notes in Computer Science, P. K. Novak, T. Smuc, and S. Dzeroski, Eds., vol. 11828. Springer, 2019, pp. 112–126. [Online]. Available: https://doi.org/10.1007/978-3-030-33778-0_10

[24] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *J. Artif. Intell. Res.*, vol. 47, pp. 253–279, 2013. [Online]. Available: https://doi.org/10.1613/jair.3912

[25] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013. [Online]. Available: http://arxiv.org/abs/1312.5602

[26] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[27] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19-24, 2016*, ser. JMLR Workshop and Conference Proceedings, M. Balcan and K. Q. Weinberger, Eds., vol. 48. JMLR.org, 2016, pp. 1928–1937. [Online]. Available: http://proceedings.mlr.press/v48/mniha16.html

[28] J. S. Obando-Ceron and P. S. Castro, "Revisiting rainbow: Promoting more insightful and inclusive deep reinforcement learning research," in *Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, ser. Proceedings of Machine Learning Research, M. Meila and T. Zhang, Eds., vol. 139. PMLR, 2021, pp. 1373–1383. [Online]. Available: http://proceedings.mlr.press/v139/ceron21a.html

[29] S. Ray and P. Tadepalli, "Model-based reinforcement learning," in *Encyclopedia of Machine Learning*, C. Sammut and G. I. Webb, Eds. Springer, 2010, pp. 690–693. [Online]. Available: https://doi.org/10.1007/978-0-387-30164-8_556

[30] I. Rechenberg, "Evolutionsstrategien," in *Simulationsmethoden in der Medizin und Biologie*, B. Schneider and U. Ranft, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1978, pp. 83–114.

[31] H. Jh, "Adaptation in natural and artificial systems," *Ann Arbor*, 1975.

[32] C. R. Reeves, "Genetic algorithms," in *Encyclopedia of Database Systems*, L. Liu and M. T. Özsu, Eds. Springer US, 2009, pp. 1224–1227. [Online]. Available: https://doi.org/10.1007/978-0-387-39940-9_562

[33] D. E. Goldberg, *Genetic Algorithms in Search Optimization and Machine Learning*. Addison-Wesley, 1989.

[34] F. Hoffmeister and T. Bäck, "Genetic algorithms and evolution strategies - similarities and differences," in *Parallel Problem Solving from Nature, 1st Workshop, PPSN I, Dortmund, Germany, October 1-3, 1990, Proceedings*, ser. Lecture Notes in Computer Science, H. Schwefel and R. Männer, Eds., vol. 496. Springer, 1990, pp. 455–469. [Online]. Available: https://doi.org/10.1007/BFb0029787

[35] T. Bäck and F. Hoffmeister, "Extended selection mechanisms in genetic algorithms," in *Proceedings of the 4th International Conference on Genetic Algorithms, San Diego, CA, USA, July 1991*, R. K. Belew and L. B. Booker, Eds. Morgan Kaufmann, 1991, pp. 92–99.

[36] T. Bäck, "Self-adaptation in genetic algorithms," 10 1994.

[37] T. Bäck, F. Hoffmeister, and H. Schwefel, "A survey of evolution strategies," in *Proceedings of the 4th International Conference on Genetic Algorithms, San Diego, CA, USA, July 1991*, R. K. Belew and L. B. Booker, Eds. Morgan Kaufmann, 1991, pp. 2–9.

[38] T. Bäck, *Evolutionary algorithms in theory and practice - evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press, 1996.

[39] T. Bäck, D. B. Fogel, and Z. Michalewicz, *Evolutionary computation 1: Basic algorithms and operators*. CRC press, 2018. [Online]. Available: https://books.google.nl/books?hl=it&lr=&id=4HMYCq9US78C&oi=fnd&pg=PA59&dq=+thomas+back+david+fogel&ots=AF9H47SUs2&sig=DPM0I_aSAuVCX_h5Gkr3CJf7F-k&redir_esc=y#v=onepage&q=thomas%20back%20david%20fogel&f=false

[40] N. Hansen and A. Ostermeier, "Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation," in *Proceedings of 1996 IEEE International Conference on Evolutionary Computation, Nayoya University, Japan, May 20-22, 1996*, T. Fukuda and T. Furuhashi, Eds. IEEE, 1996, pp. 312–317. [Online]. Available: https://doi.org/10.1109/ICEC.1996.542381

[41] N. Hansen, "The CMA evolution strategy: A tutorial," *CoRR*, vol. abs/1604.00772, 2016. [Online]. Available: http://arxiv.org/abs/1604.00772

[42] K. O. Stanley and R. Miikkulainen, "Evolving neural network through augmenting topologies," *Evol. Comput.*, vol. 10, no. 2, pp. 99–127, 2002. [Online]. Available: https://doi.org/10.1162/106365602320169811

[43] B. L. Miller and D. E. Goldberg, "Genetic algorithms, tournament selection, and the effects of noise," *Complex Syst.*, vol. 9, no. 3, 1995. [Online]. Available: http://www.complex-systems.com/abstracts/v09_i03_a02.html

[44] J. Clune, J. Mouret, and H. Lipson, "The evolutionary origins of modularity," *CoRR*, vol. abs/1207.2743, 2012. [Online]. Available: http://arxiv.org/abs/1207.2743

[45] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, 2002. [Online]. Available: https://doi.org/10.1109/4235.996017

[46] C. J. C. H. Watkins and P. Dayan, "Technical note q-learning," *Mach. Learn.*, vol. 8, pp. 279–292, 1992. [Online]. Available: https://doi.org/10.1007/BF00992698

[47] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. The MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/the-book-2nd.html

[48] L.-J. Lin, *Reinforcement learning for robots using neural networks*. Carnegie Mellon University, 1992.

[49] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," *Mach. Learn.*, vol. 8, pp. 229–256, 1992. [Online]. Available: https://doi.org/10.1007/BF00992696

[50] J. Schulman, S. Levine, P. Abbeel, M. I. Jordan, and P. Moritz, "Trust region policy optimization," in *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, ser. JMLR Workshop and Conference Proceedings, F. R. Bach and D. M. Blei, Eds., vol. 37. JMLR.org, 2015, pp. 1889–1897. [Online]. Available: http://proceedings.mlr.press/v37/schulman15.html

[51] I. Sobol, "Global sensitivity indices for nonlinear mathematical models and their monte carlo estimates," *Mathematics and Computers in Simulation*, vol. 55, no. 1, pp. 271–280, 2001, the Second IMACS Seminar on Monte Carlo Methods. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0378475400002706

[52] M. Lindauer, K. Eggensperger, M. Feurer, A. Biedenkapp, J. Marben, P. Müller, and F. Hutter, "BOAH: A tool suite for multi-fidelity bayesian optimization & analysis of hyperparameters," *CoRR*, vol. abs/1908.06756, 2019. [Online]. Available: http://arxiv.org/abs/1908.06756

[53] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. A. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nat.*, vol. 518, no. 7540, pp. 529–533, 2015. [Online]. Available: https://doi.org/10.1038/nature14236

[54] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *CoRR*, vol. abs/1606.01540, 2016. [Online]. Available: http://arxiv.org/abs/1606.01540

[55] P. Gavrikov, "visualkeras," https://github.com/paulgavrikov/visualkeras, 2020.

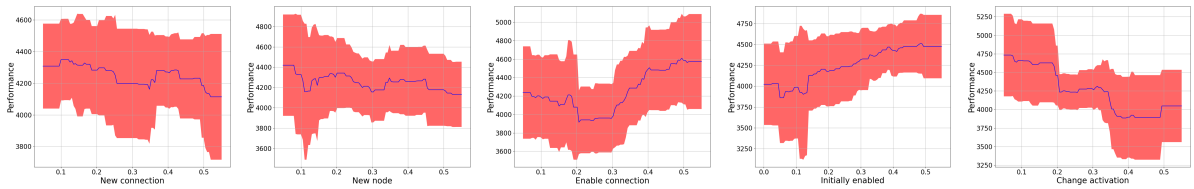
A Detailed Results

A.1 Default Hyperparameters

Hyperparameter	Default Value	Description
task	swingup	name of task
alg_wDist	standard	either 6 chosen values or linspace of alg_nVals
alg_nVals	6	number of weights to test when evaluating individual
alg_nReps	4	number of repetitions when evaluating individuals
alg_probMoo	0.80	chance of applying second objective when using MOO
maxGen	1024	number of generations to run algorithm
popSize	128	number of individuals in population
prob_crossover	0.0	chance of crossover
prob_mutAct	0.50	chance to change node activation function
prob_addNode	0.25	chance to add node
prob_addConn	0.20	chance to add connections
prob_enable	0.05	chance to enable disabled connection
prob_initEnable	0.5	chance to enable each initial connection
select_cullRatio	0.2	% of individuals to remove before selection
select_eliteRatio	0.2	% of individuals to pass on to next generation unchanged
select_tournSize	8	number of competitors in each tournament
save_mod	8	generations between saving results to disk
bestReps	20	number of times to test new 'best' solutions to confirm

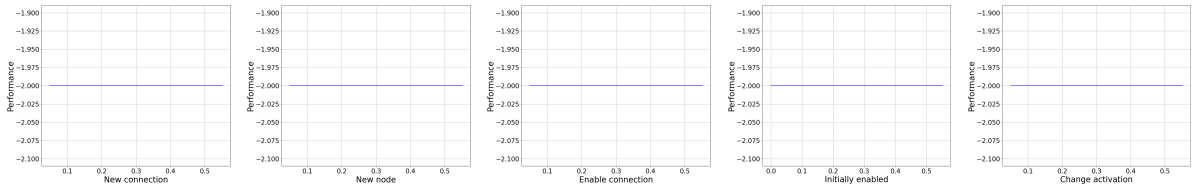
Table 4: Default hyperparameter values provided by the author of the original WANN paper [4], mentioned in Section 5.1

A.2 Marginal Performance Predictions



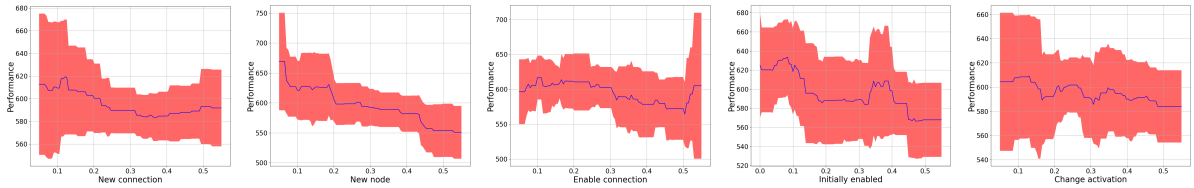
(a) (b) (c) (d) (e)

Figure 17: Marginal Performance Prediction for the hyperparameters of the configuration space for the Battle Zone game, mentioned in Section 6.1. The red area represents the standard deviation, while the blue line represents the marginal.



(a) (b) (c) (d) (e)

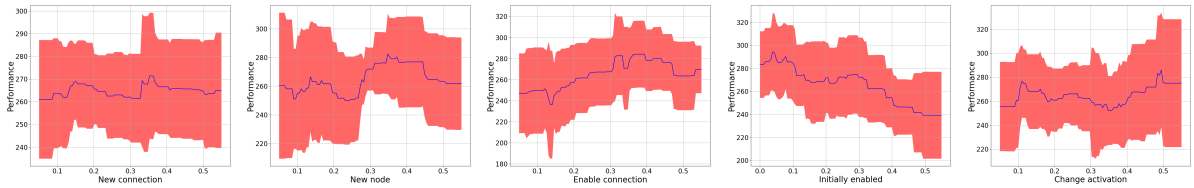
Figure 18: Marginal Performance Prediction for the hyperparameters of the configuration space for the Double Dunk game, mentioned in Section 6.1. The red area represents the standard deviation, while the blue line represents the marginal.



(a) (b) (c) (d) (e)

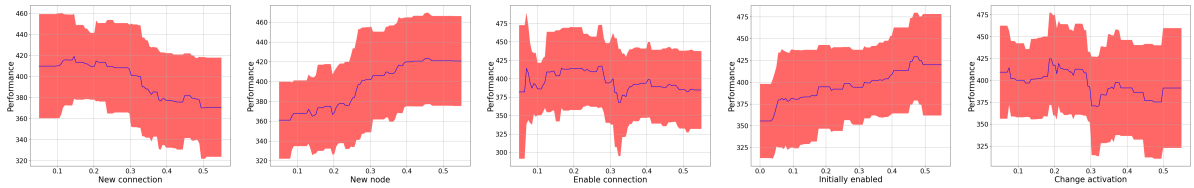
Figure 19: Marginal Performance Prediction for the hyperparameters of the configuration space for the Name This Game game, mentioned in Section 6.1. The red area represents the standard deviation, while the blue line represents the marginal.

.



(a) (b) (c) (d) (e)

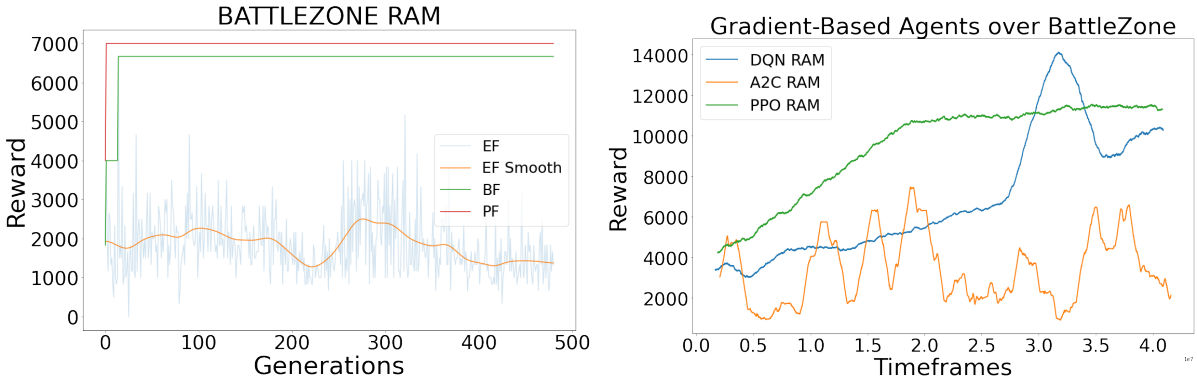
Figure 20: Marginal Performance Prediction for the hyperparameters of the configuration space for the Phoenix game, mentioned in Section 6.1. The red area represents the standard deviation, while the blue line represents the marginal.



(a) (b) (c) (d) (e)

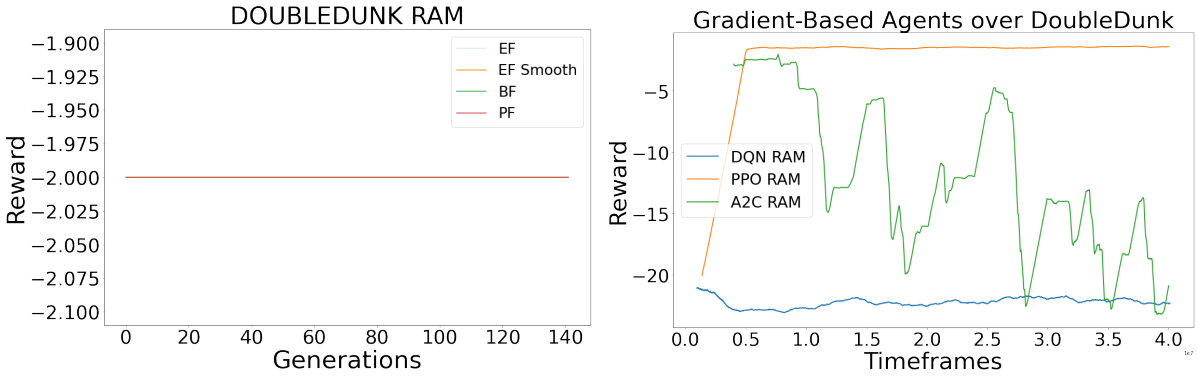
Figure 21: Marginal Performance Prediction for the hyperparameters of the configuration space for the Q*bert game, mentioned in Section 6.1. The red area represents the standard deviation, while the blue line represents the marginal.

A.3 RAM Representation



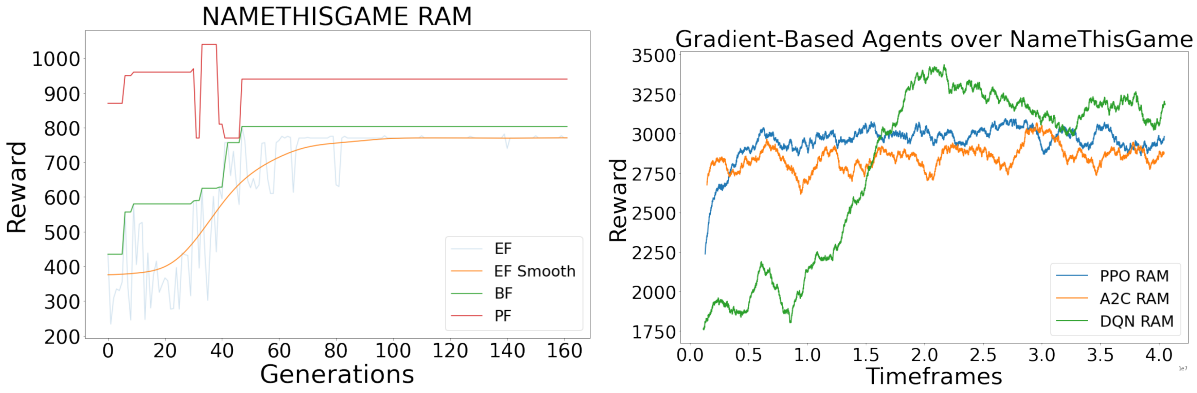
(a) (b)

Figure 22: In Figure 22a, the evolution process of the WANN over the Battle Zone game with the RAM representation is shown, reporting elite fit, best fit, and peak fit. In Figure 22b, the training process of the three gradient-based agents is shown over a span of 40M timeframes (which are roughly 10M timesteps with a frameskip value of 4).



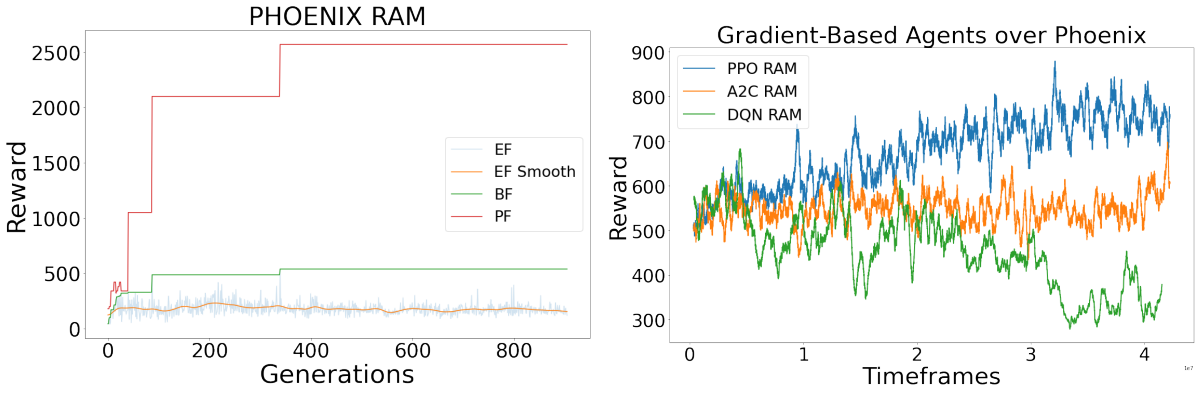
(a) (b)

Figure 23: In Figure 23a, the evolution process of the WANN over the Double Dunk game with the RAM representation is shown, reporting elite fit, best fit, and peak fit. In Figure 23b, the training process of the three gradient-based agents is shown over a span of 40M timeframes (which are roughly 10M timesteps with a frameskip value of 4).



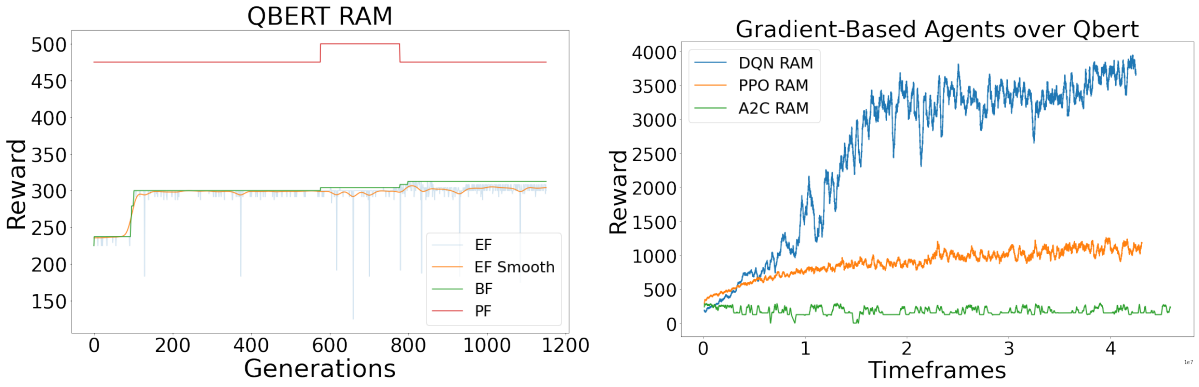
(a) (b)

Figure 24: In Figure 24a, the evolution process of the WANN over the "Name This Game" game with the RAM representation is shown, reporting elite fit, best fit, and peak fit. In Figure 24b, the training process of the three gradient-based agents is shown over a span of 40M timeframes (which are roughly 10M timesteps with a frameskip value of 4).



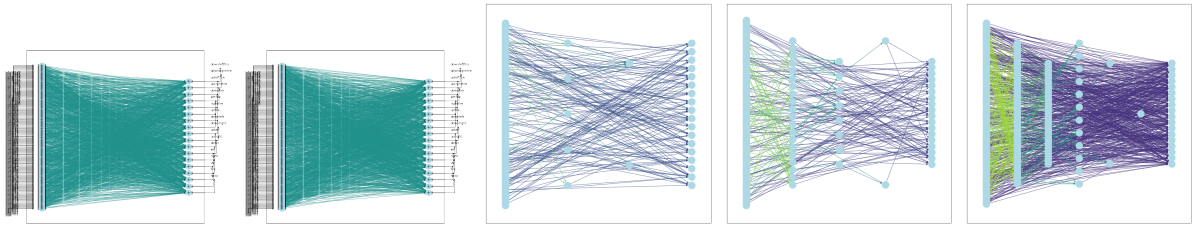
(a) (b)

Figure 25: In Figure 25a, the evolution process of the WANN over the game Phoenix with the RAM representation is shown, reporting elite fit, best fit, and peak fit. In Figure 25b, the training process of the three gradient-based agents is shown over a span of 40M timeframes (which are roughly 10M timesteps with a frameskip value of 4).



(a) (b)

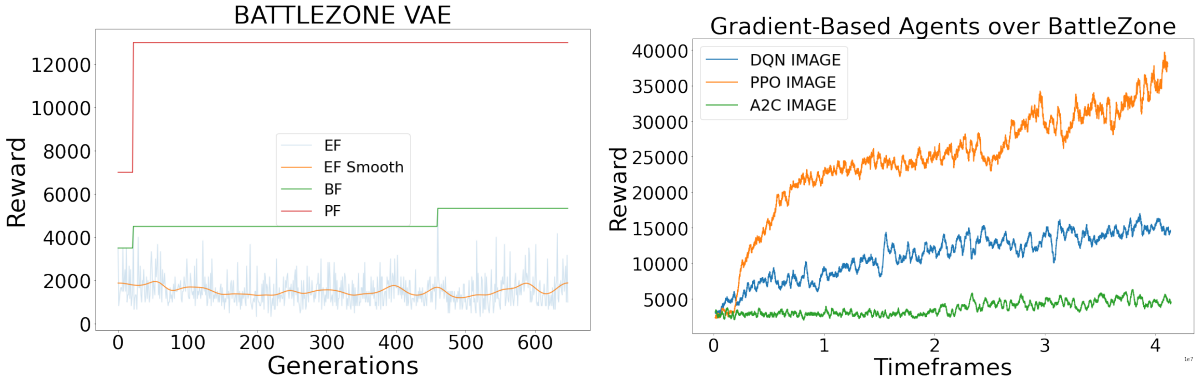
Figure 26: In Figure 26a, the evolution process of the WANN over the game Q*bert with the RAM representation is shown, reporting elite fit, best fit, and peak fit. In Figure 26b, the training process of the three gradient-based agents is shown over a span of 40M timeframes (which are roughly 10M timesteps with a frameskip value of 4).



(a) Battle Zone (b) Double Dunk (c) NTG (d) Phoenix (e) Q*bert

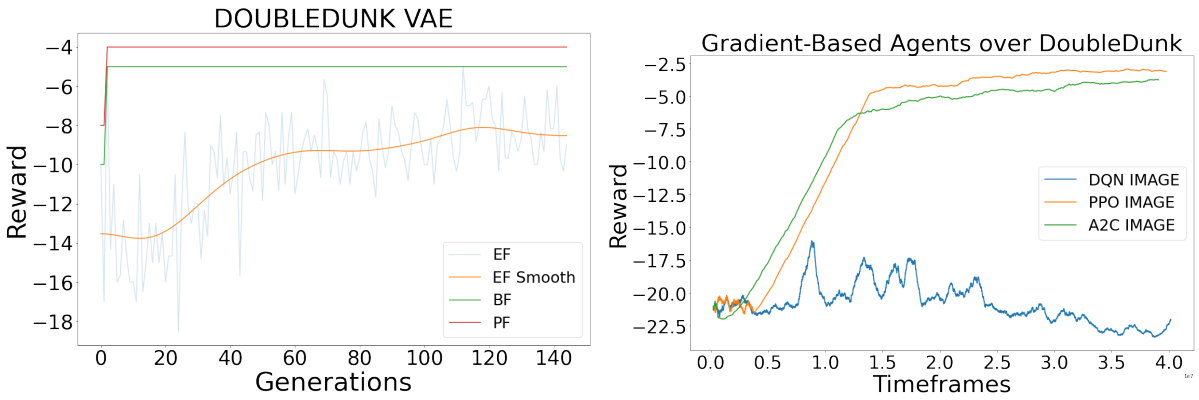
Figure 27: Connectomes of the networks generated during the evolutionary process over each game with the RAM representation.

A.4 Image Representation



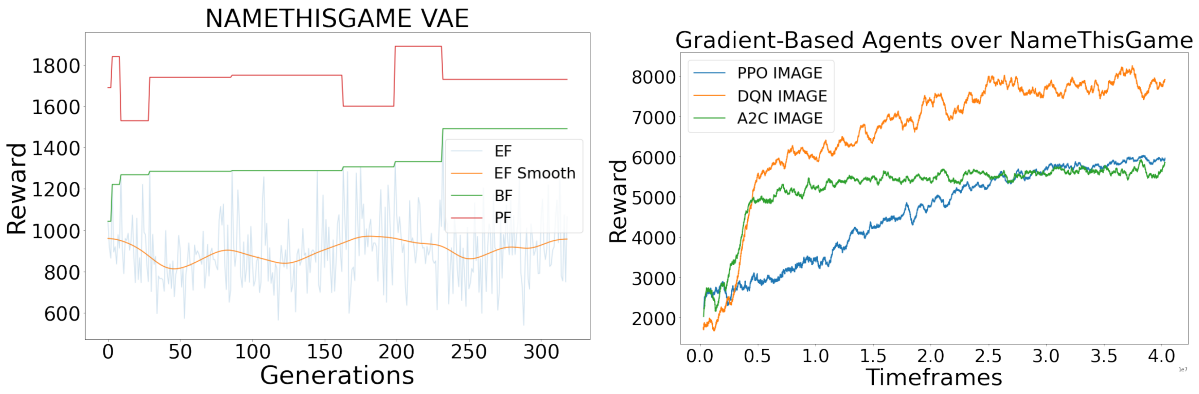
(a) (b)

Figure 28: In Figure 28a, the evolution process of the WANN over the Battle Zone game with the image representation is shown, reporting elite fit, best fit, and peak fit. In Figure 28b, the training process of the three gradient-based agents is shown over a span of 40M timeframes (which are roughly 10M timesteps with a frameskip value of 4).



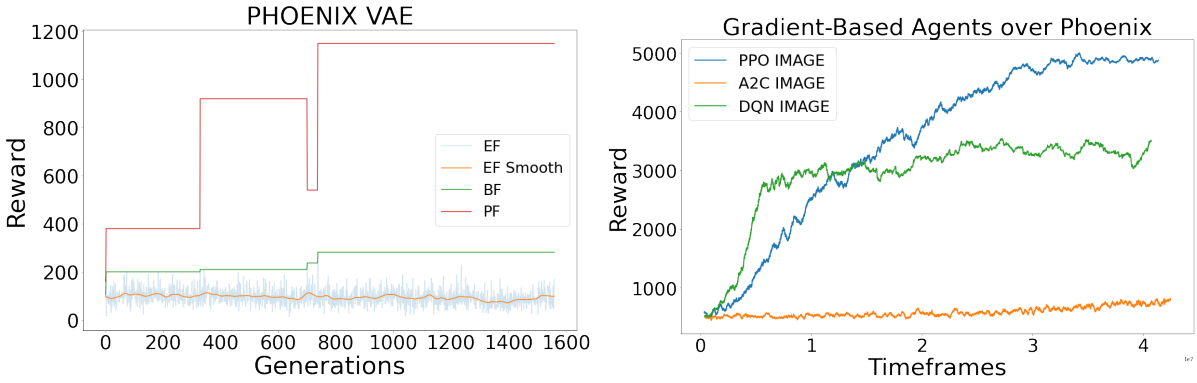
(a) (b)

Figure 29: In Figure 29a, the evolution process of the WANN over the Double Dunk game with the image representation is shown, reporting elite fit, best fit, and peak fit. In Figure 29b, the training process of the three gradient-based agents is shown over a span of 40M timeframes (which are roughly 10M timesteps with a frameskip value of 4).



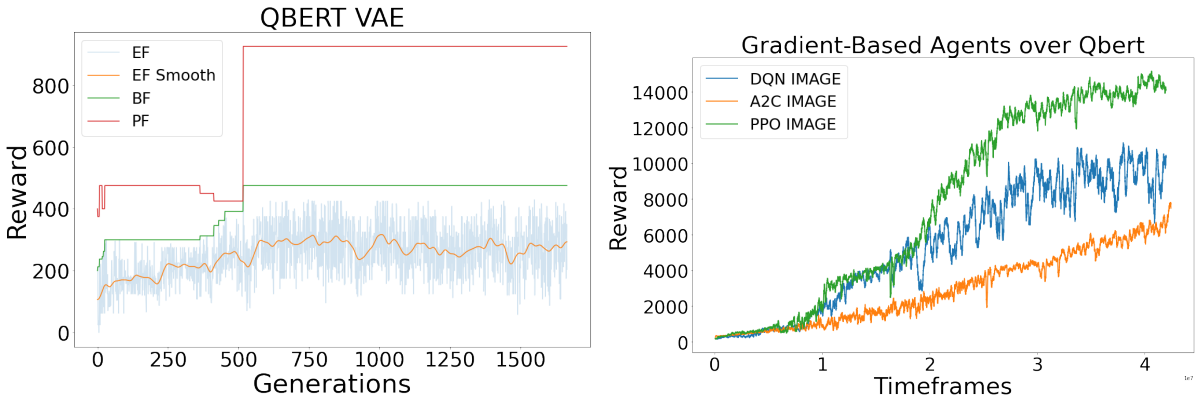
(a) (b)

Figure 30: In Figure 30a, the evolution process of the WANN over the "Name This Game" game with the image representation is shown, reporting elite fit, best fit, and peak fit. In Figure 30b, the training process of the three gradient-based agents is shown over a span of 40M timeframes (which are roughly 10M timesteps with a frameskip value of 4).



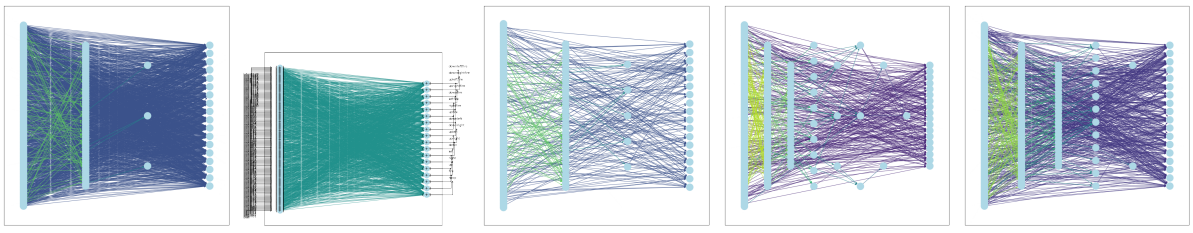
(a) (b)

Figure 31: In Figure 31a, the evolution process of the WANN over the Phoenix game with the image representation is shown, reporting elite fit, best fit, and peak fit. In Figure 31b, the training process of the three gradient-based agents is shown over a span of 40M timeframes (which are roughly 10M timesteps with a frameskip value of 4).



(a) (b)

Figure 32: In Figure 32a, the evolution process of the WANN over the Q*bert game with the image representation is shown, reporting elite fit, best fit, and peak fit. In Figure 32b, the training process of the three gradient-based agents is shown over a span of 40M timeframes (which are roughly 10M timesteps with a frameskip value of 4).

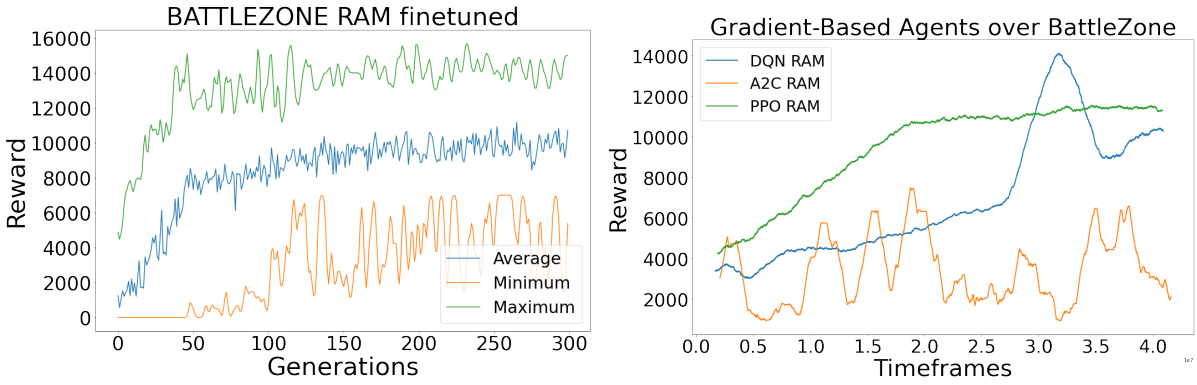


(a) Battle Zone (b) Double Dunk (c) NTG (d) Phoenix (e) Q*bert

Figure 33: Connectomes of the networks generated during the evolutionary process over each game with the Image representation.

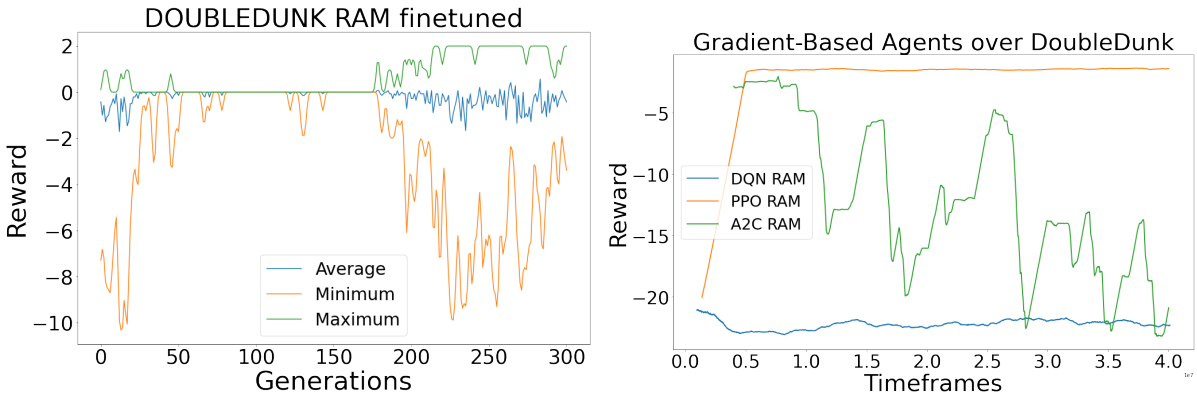
A.5 Finetuning

A.5.1 RAM Representation



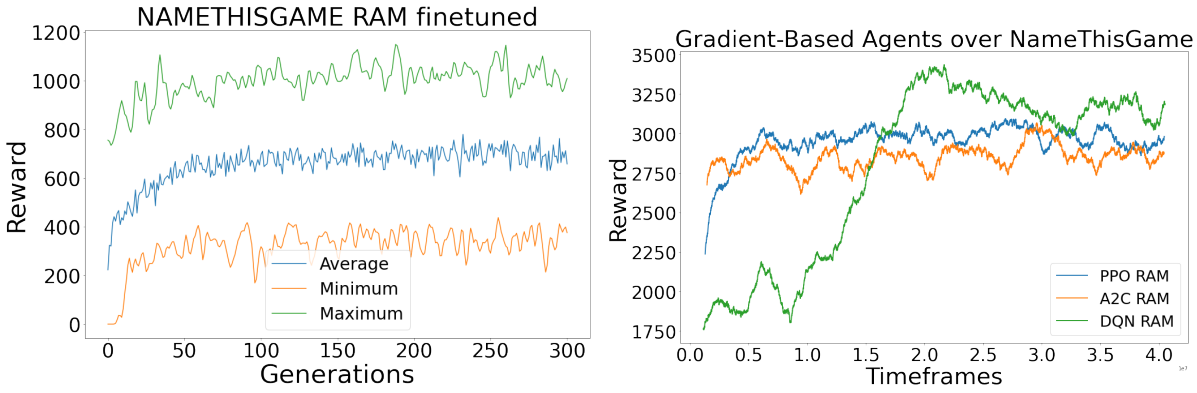
(a) (b)

Figure 34: In Figure 34a, the finetuning process of the WANN over the Battle Zone game with the RAM representation is shown, reporting the minimum reward reached, the maximum reward reached, and the average reward reached. In Figure 34b, the training process of the three gradient-based agents is shown over a span of 40M timeframes (which are roughly 10M timesteps with a frameskip value of 4).



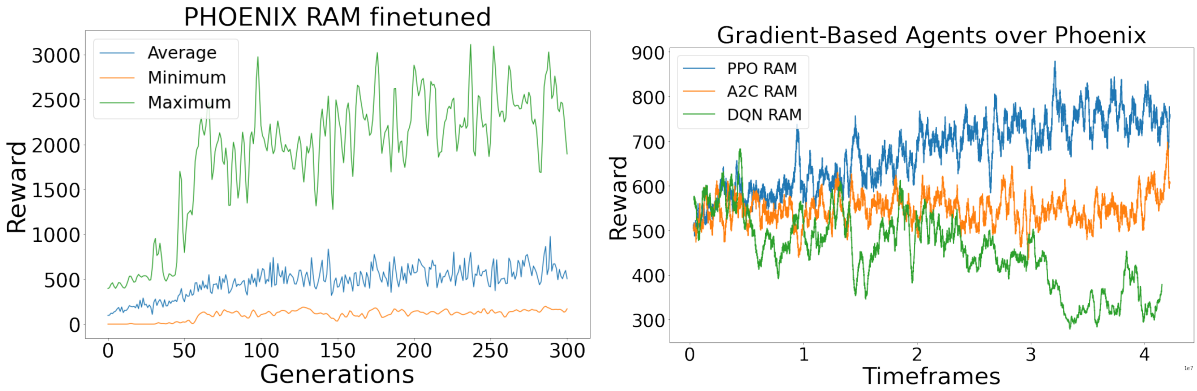
(a) (b)

Figure 35: In Figure 35a, the finetuning process of the WANN over the Double Dunk game with the RAM representation is shown, reporting the minimum reward reached, the maximum reward reached, and the average reward reached. In Figure 35b, the training process of the three gradient-based agents is shown over a span of 40M timeframes (which are roughly 10M timesteps with a frameskip value of 4).



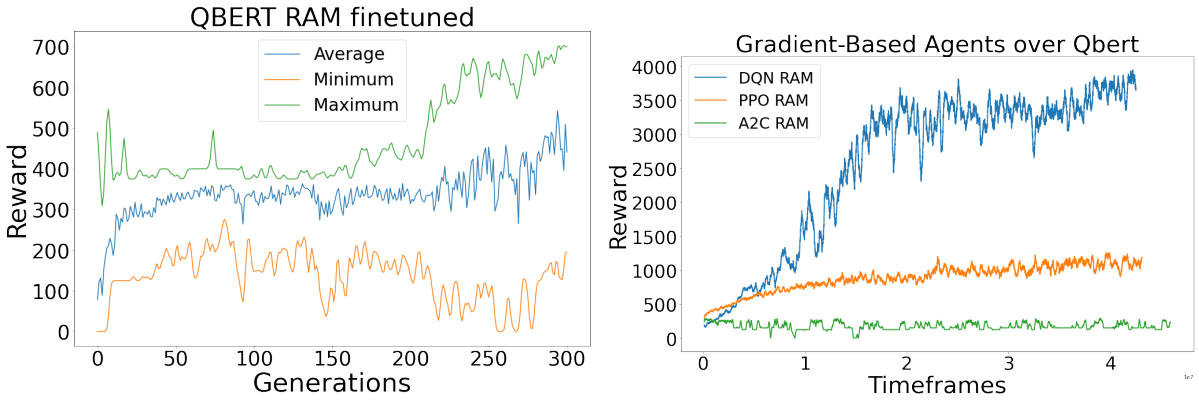
(a) (b)

Figure 36: In Figure 36a, the finetuning process of the WANN over the "Name This Game" game with the RAM representation is shown, reporting the minimum reward reached, the maximum reward reached, and the average reward reached. In Figure 36b, the training process of the three gradient-based agents is shown over a span of 40M timeframes (which are roughly 10M timesteps with a frameskip value of 4).



(a) (b)

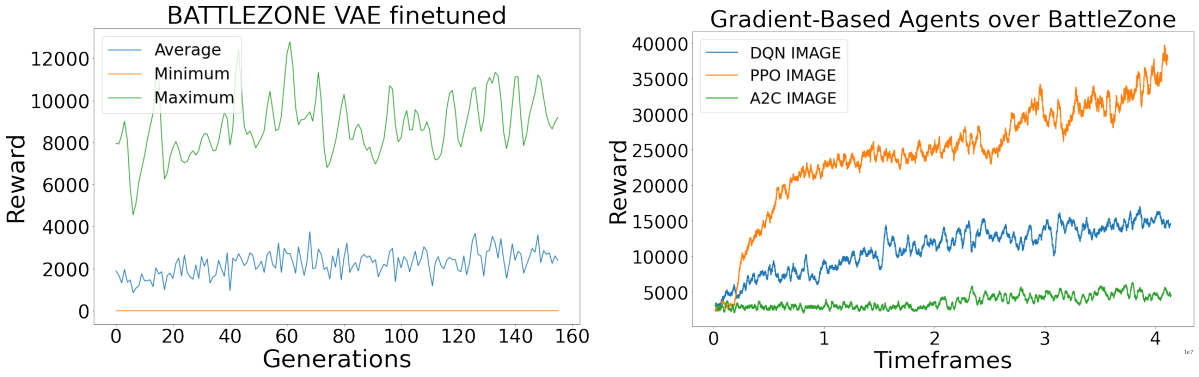
Figure 37: In Figure 37a, the finetuning process of the WANN over the game Phoenix with the RAM representation is shown, reporting the minimum reward reached, the maximum reward reached, and the average reward reached. In Figure 37b, the training process of the three gradient-based agents is shown over a span of 40M timeframes (which are roughly 10M timesteps with a frameskip value of 4).



(a) (b)

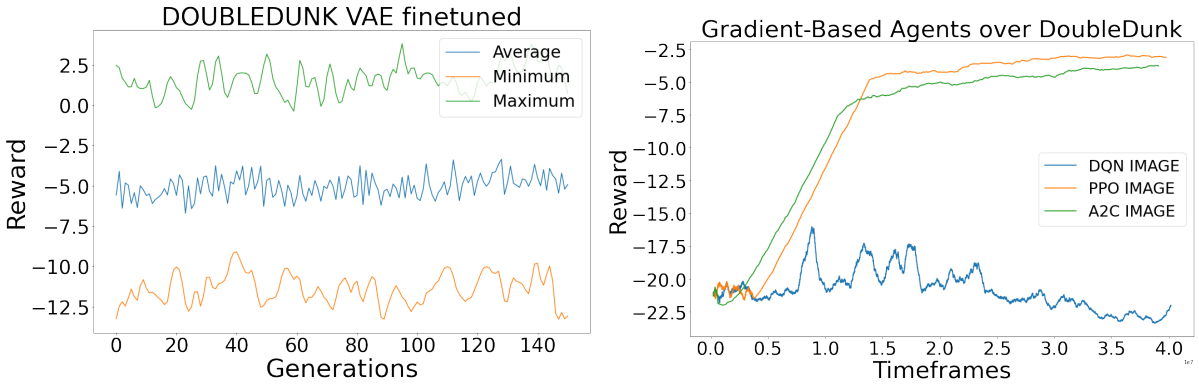
Figure 38: In Figure 38a, the finetuning process of the WANN over the game Q*bert with the RAM representation is shown, reporting the minimum reward reached, the maximum reward reached, and the average reward reached. In Figure 38b, the training process of the three gradient-based agents is shown over a span of 40M timeframes (which are roughly 10M timesteps with a frameskip value of 4).

A.5.2 Image Representation



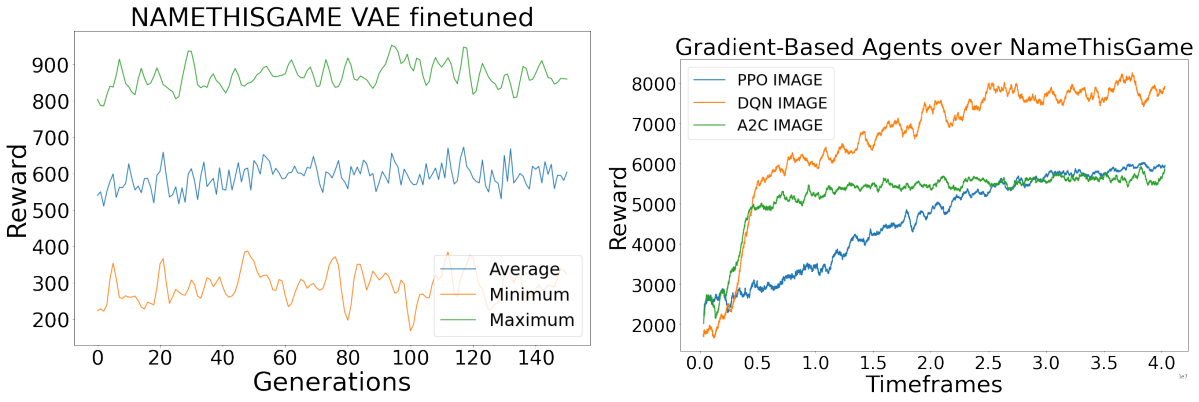
(a) (b)

Figure 39: In Figure 39a, the finetuning process of the WANN over the Battle Zone game with the image representation is shown, reporting the minimum reward reached, the maximum reward reached, and the average reward reached. In Figure 39b, the training process of the three gradient-based agents is shown over a span of 40M timeframes (which are roughly 10M timesteps with a frameskip value of 4).



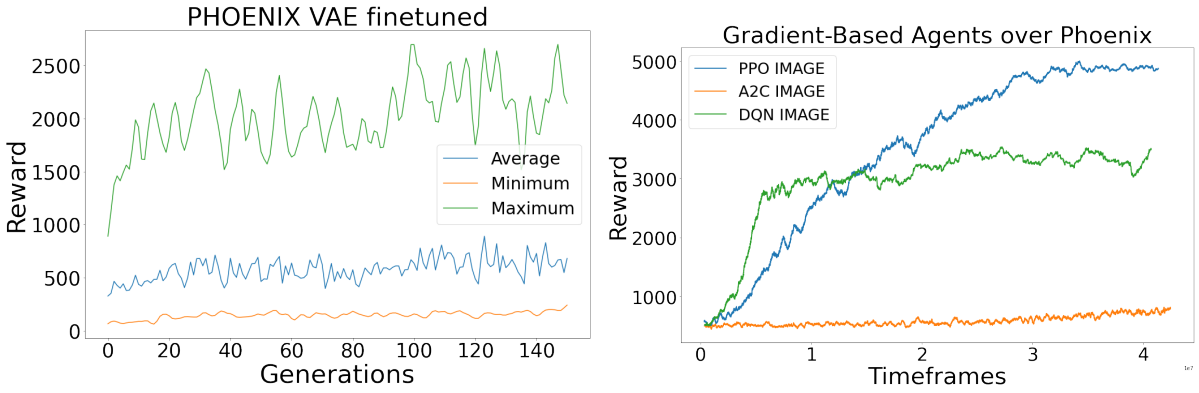
(a) (b)

Figure 40: In Figure 40a, the finetuning process of the WANN over the Double Dunk game with the image representation is shown, reporting the minimum reward reached, the maximum reward reached, and the average reward reached. In Figure 40b, the training process of the three gradient-based agents is shown over a span of 40M timeframes (which are roughly 10M timesteps with a frameskip value of 4).



(a) (b)

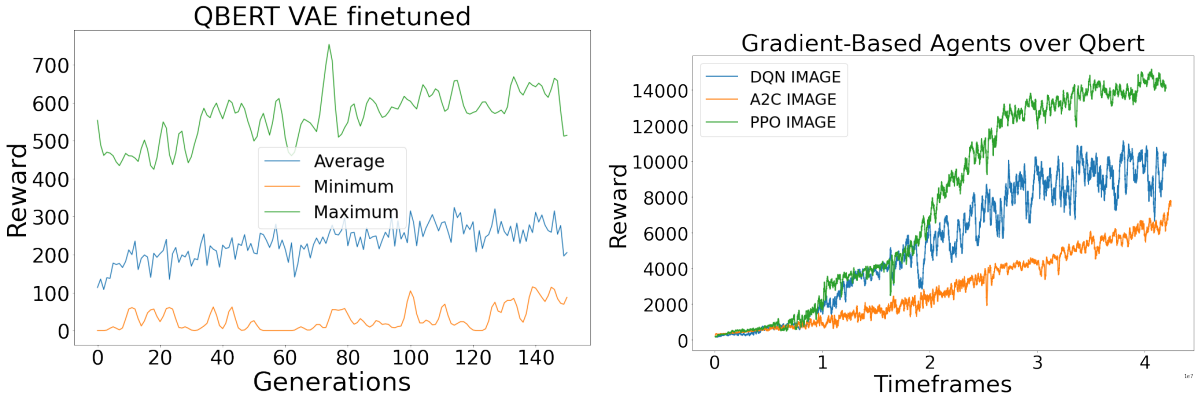
Figure 41: In Figure 41a, the finetuning process of the WANN over the "Name This Game" game with the image representation is shown, reporting the minimum reward reached, the maximum reward reached, and the average reward reached. In Figure 41b, the training process of the three gradient-based agents is shown over a span of 40M timeframes (which are roughly 10M timesteps with a frameskip value of 4).



(a) (b)

Figure 42: In Figure 42a, the finetuning process of the WANN over the game Phoenix with the image representation is shown, reporting the minimum reward reached, the maximum reward reached, and the average reward reached. In Figure 42b, the training process of the three gradient-based agents is shown over a span of 40M timeframes (which are roughly 10M timesteps with a frameskip value of 4).

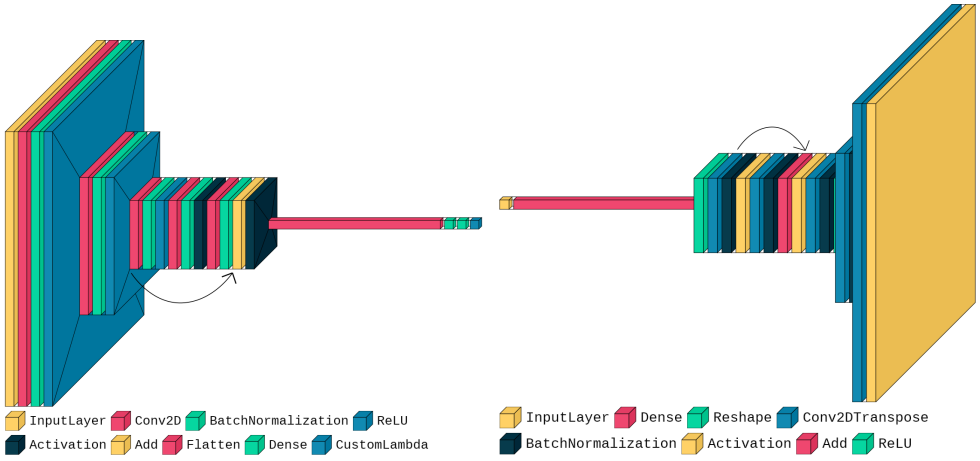
1



(a) (b)

Figure 43: In Figure 43a, the finetuning process of the WANN over the game Q*bert with the image representation is shown, reporting the minimum reward reached, the maximum reward reached, and the average reward reached. In Figure 43b, the training process of the three gradient-based agents is shown over a span of 40M timeframes (which are roughly 10M timesteps with a frameskip value of 4).

B Details on Variational Autoencoder



(a) Encoder (b) Decoder

Figure 44: VAE Architecture. Figures generated using Visualkeras [55].

The architecture of the encoder is shown in Figure 44a and summarized as follows:

- First convolutional hidden layer, with 32 filters of size 1 and stride 1
- Second convolutional hidden layer, with 32 filters of size 4 and stride 2
- Third convolutional hidden layer, with 64 filters of size 4 and stride 2
- Skip connection followed by 2 convolutional layers, each with 64 filters of size 3 and stride 1
- Add Layer and Flatten
- Two dense layers generate the distribution's mean and log variance, and a custom Lambda Layer combines the two

The architecture of the decoder is shown in Figure 44b and summarized as follows:

- A first Dense layer followed by a Reshape maps the (128,) input into a 4D tensor.
- Skip connection, followed by two transposed convolutional hidden layers, each with 64 filters of size 3 and stride 1
- Add layer
- Transposed convolutional hidden layer, with 64 filters of size 1 and stride 1
- Transposed convolutional hidden layer, with 64 filters of size 3 and stride 2
- Transposed convolutional hidden layer, with 3 filters of size 3 and stride 2

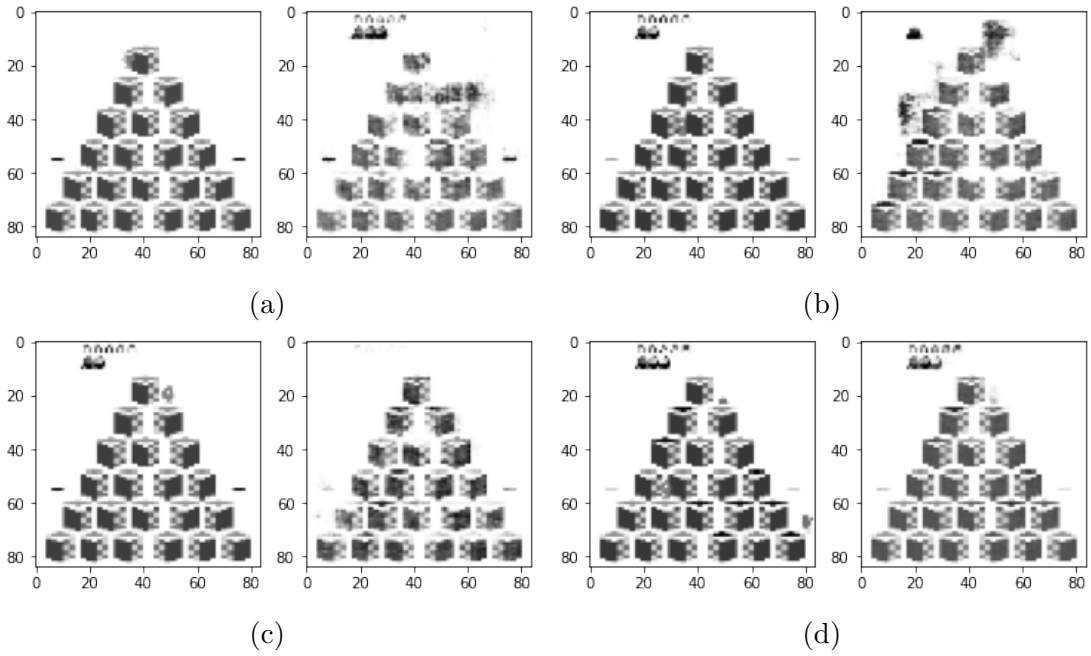


Figure 45: On the left of each of the presented Figures 45a,45b,45c,45d, the 84x84 resized grayscale screens are shown. On the right, the reconstructions of the 84x84 grayscale images from the game Q*bert, using decoder architecture inspired by the CNN architecture of the original DQN paper [25].

In Figures 45a,45b,45c,45d, we show the comparison between the original images (on the left) and the images reconstructed using the previously mentioned VAE (on the right). We notice how the reconstructions are often not correct or lack important details. The reconstruction in Figure 45d looks more accurate compared to the other ones, but it lacks some important details that have a key role in games such as Q*bert. The main character, present in the original frame in the lower right angle of the frame, is missing, and some of the cubes of the bottom "floor" have not been activated.
Using the setup presented in Section 4.1.2, the reconstructed images for the game Q*bert have been generated accurately, as shown in Figure 46.

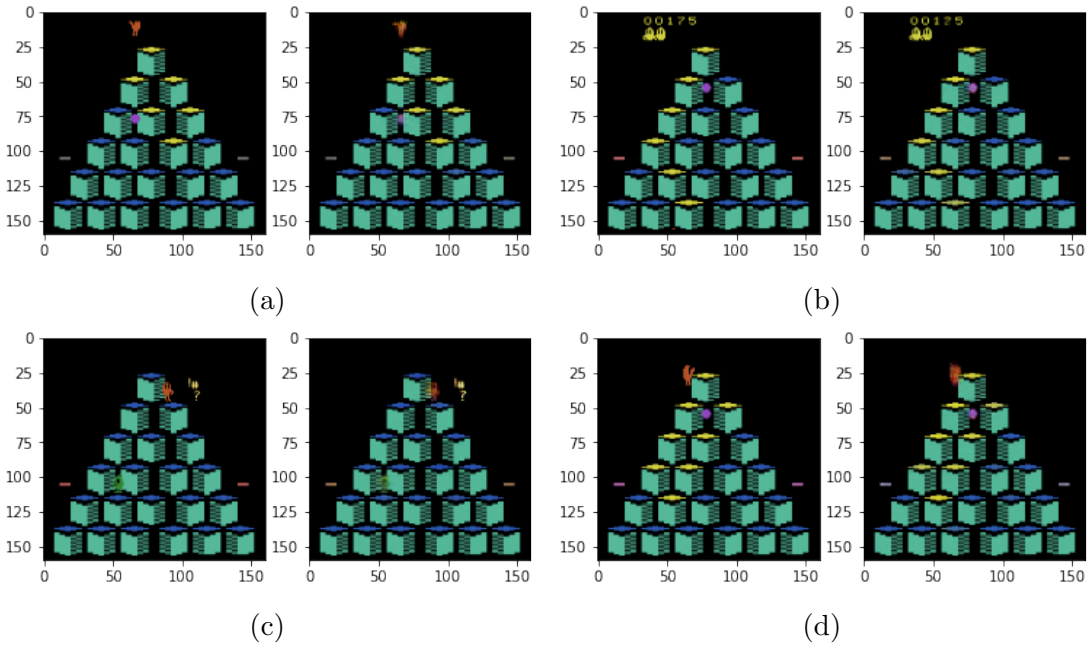


Figure 46: On the left of each of the presented Figures 46a46b,46c,46d, 160x160 resized RGB screens are shown. On the right, the reconstructions of 160x160 resized RGB images from the game Q*bert, using the proposed VAE.

As shown in Figure 45d, while the cubes of the higher "floors" are correctly activated, going to the lower "floors" the reconstruction of the activation gets less accurate. This issue is less pronounced in the RGB version, as we can see in Figure 46b. The cube activated on the lowest "floor" is reconstructed but with a slightly dark color.

Game	Reconstruction Loss	KL Loss	Total Loss
Battle Zone	10088.12	15.18	10103.30
Double Dunk	2528.98	9.87	2538.85
Name This Game	11466.80	14.50	11481.30
Phoenix	2181.17	10.51	2191.68
Q*bert	4385.25	17.77	4403.02

Table 5: Table reporting the reconstruction loss, the KL loss, and the total loss of the VAE during training for each of the games. The scale of the reconstruction and total losses can vary throughout the games.

Encoder Summary

```
    Model: "encoder"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
==================================================================================================
 encoder_input (InputLayer)     [(None, 160, 160, 3  0           []
                                )]

 encoder_conv_0 (Conv2D)        (None, 160, 160, 32  128         ['encoder_input[0][0]']
                                )
```

```
batch_normalization_18 (BatchN  (None, 160, 160, 32  128        ['encoder_conv_0[0][0]']
ormalization)                   )

re_lu_10 (ReLU)                 (None, 160, 160, 32  0          ['batch_normalization_18[0][0]']
                                )

encoder_conv_1 (Conv2D)         (None, 80, 80, 32)   16416      ['re_lu_10[0][0]']

batch_normalization_19 (BatchN  (None, 80, 80, 32)  128         ['encoder_conv_1[0][0]']
ormalization)

re_lu_11 (ReLU)                 (None, 80, 80, 32)   0          ['batch_normalization_19[0][0]']

encoder_conv_2 (Conv2D)         (None, 40, 40, 64)   32832      ['re_lu_11[0][0]']

batch_normalization_20 (BatchN  (None, 40, 40, 64)  256         ['encoder_conv_2[0][0]']
ormalization)

re_lu_12 (ReLU)                 (None, 40, 40, 64)   0          ['batch_normalization_20[0][0]']

conv2d_4 (Conv2D)               (None, 40, 40, 64)   36928      ['re_lu_12[0][0]']

batch_normalization_21 (BatchN  (None, 40, 40, 64)  256         ['conv2d_4[0][0]']
ormalization)

activation_8 (Activation)       (None, 40, 40, 64)   0          ['batch_normalization_21[0][0]']

conv2d_5 (Conv2D)               (None, 40, 40, 64)   36928      ['activation_8[0][0]']

batch_normalization_22 (BatchN  (None, 40, 40, 64)  256         ['conv2d_5[0][0]']
ormalization)

add_4 (Add)                     (None, 40, 40, 64)   0          ['batch_normalization_22[0][0]',
                                                                 're_lu_12[0][0]']

activation_9 (Activation)       (None, 40, 40, 64)   0          ['add_4[0][0]']

flatten_2 (Flatten)             (None, 102400)       0          ['activation_9[0][0]']

mu (Dense)                      (None, 128)          13107328   ['flatten_2[0][0]']

log_var (Dense)                 (None, 128)          13107328   ['flatten_2[0][0]']

custom_lambda_2 (CustomLambda)  (None, 128)          0          ['mu[0][0]',
                                                                 'log_var[0][0]']

==================================================================================================
Total params: 26,338,912
Trainable params: 26,338,400
Non-trainable params: 512
__________________________________________________________________________________________________
```

Decoder Summary

```
    Model: "decoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
```

```
==================================================================================================
 decoder_input (InputLayer)     [(None, 128)]        0           []

 dense_2 (Dense)                (None, 102400)       13209600    ['decoder_input[0][0]']

 reshape_2 (Reshape)            (None, 40, 40, 64)   0           ['dense_2[0][0]']

 conv2d_transpose_4 (Conv2DTran  (None, 40, 40, 64)  36928       ['reshape_2[0][0]']
 spose)

 batch_normalization_23 (BatchN  (None, 40, 40, 64)  256         ['conv2d_transpose_4[0][0]']
 ormalization)

 activation_10 (Activation)     (None, 40, 40, 64)   0           ['batch_normalization_23[0][0]']

 conv2d_transpose_5 (Conv2DTran  (None, 40, 40, 64)  36928       ['activation_10[0][0]']
 spose)

 batch_normalization_24 (BatchN  (None, 40, 40, 64)  256         ['conv2d_transpose_5[0][0]']
 ormalization)

 add_5 (Add)                    (None, 40, 40, 64)   0           ['batch_normalization_24[0][0]',
                                                                  'reshape_2[0][0]']

 activation_11 (Activation)     (None, 40, 40, 64)   0           ['add_5[0][0]']

 decoder_conv_t_0 (Conv2DTransp  (None, 40, 40, 64)  4160        ['activation_11[0][0]']
 ose)

 batch_normalization_25 (BatchN  (None, 40, 40, 64)  256         ['decoder_conv_t_0[0][0]']
 ormalization)

 re_lu_13 (ReLU)                (None, 40, 40, 64)   0           ['batch_normalization_25[0][0]']

 decoder_conv_t_1 (Conv2DTransp  (None, 80, 80, 64)  36928       ['re_lu_13[0][0]']
 ose)

 batch_normalization_26 (BatchN  (None, 80, 80, 64)  256         ['decoder_conv_t_1[0][0]']
 ormalization)

 re_lu_14 (ReLU)                (None, 80, 80, 64)   0           ['batch_normalization_26[0][0]']

 decoder_conv_t_2 (Conv2DTransp  (None, 160, 160, 3)  1731       ['re_lu_14[0][0]']
 ose)

 dc_reco (Activation)           (None, 160, 160, 3)  0           ['decoder_conv_t_2[0][0]']

==================================================================================================
Total params: 13,327,299
Trainable params: 13,326,787
Non-trainable params: 512
--------------------------------------------------------------------------------------------------
```