Exploring NEAT: Convergence, Performance Analysis and Input Reduction with Echo State Networks

Valentina Villa
Deep Learning for games and simulations course
email: vavi@itu.dk

Abstract

This study explores Neuro-Evolution of Augmenting Topologies (NEAT) in tackling OpenAI's Cart Pole and Bipedal Walker challenges. Focused on convergence speed with varying number of inputs, experiments extend from Cart-Pole's 4 inputs to 50, 100, and 500, using random additions. To enhance NEAT's performance in larger input spaces, secondary experiment combines environment and Echo State Network outputs as dual inputs to NEAT. The study assesses whether preprocessing inputs through an ESN improves convergence and overall performance. Echo State Networks are introduced for dimensionality reduction Despite rigorous experimentation, results indicate no significant performance improvement. These findings offer insights into NEAT's effectiveness and the potential benefits of ESNs. While not yielding improved performance, the study contributes to optimizing reinforcement learning, providing nuanced insights into the interplay between input dimensionality, convergence speed, and algorithmic performance in OpenAI games.

I Introduction

A primary goal of artificial intelligence is to create agents capable of doing difficult tasks in complicated, uncertain contexts. The most prevalent paradigm for evaluating such problems has been the use of a class of reinforcement learning (RL) algorithms based on the Markov Decision Process (MDP) formalism and the value function idea. This method has resulted in many successes like computers playing expert-level Go [13].
Black-box optimization is an alternative method for tackling RL problems. If applied to neural networks this strategy can take the form of direct policy search [11] or neuro-evolution [1]. These optimization methods have several benefits: a lack of concern about reward distribution (sparse or dense), the absence of the necessity for back-propagating gradients, and the ability to tolerate potentially arbitrarily long time horizons [2]. It was shown that evolution strategies can compete with RL algorithms on some of the most difficult environments studied by the deep RL community today, and that this approach can also be scaled to more parallel workers [5].
Optimizing the efficiency and robustness of neural network architectures is one of the goals of artificial intelligence. One evolutionary algorithm that has garnered attention for its promising attributes is the Neuro-Evolution of Augmenting Topologies (NEAT) algorithm [6]. NEAT, known for its adaptability in evolving neural network structures, has demonstrated many remarkable successes in various applications. In real-world scenarios, practical problems often involve high-dimensional input spaces where not all inputs are directly relevant to solve the given problem. To simulate this reality, random inputs were introduced to the original set of inputs; this addition aims to emulate the presence of extraneous or non-contributing variables, mirroring the challenges of navigating complex and diverse input spaces commonly encountered in real-world applications. The introduction of random input data into the initial set poses an intriguing question: *How does the performance of NEAT evolve with the inclusion of random inputs, and does this evolution exhibit a consistent trend? Will the solutions take into account the newly introduced inputs?*
The author’s hypothesis is that an increase in the number of inputs may lead to a degradation in algorithmic performance. This conjecture is grounded in the belief that, as the complexity of input space expands, the algorithm might face challenges in adapting and evolving neural network structures effectively, especially because NEAT starts from a random pool of neural networks. Having more inputs means that each neural network has more input nodes. Accordingly, the number of possible neural networks grows exponentially as the number of inputs, making the solution space that NEAT algorithm must explore more extensive. Understanding the behavior of NEAT in the presence of random inputs can be important for enhancing the algorithm's applicability and comprehending its limitations. By exploring this aspect, the goal of this paper is to bring valuable insights to the broader field of neuroevolutionary algorithms.

The second question this paper aims to address is “*If the inputs we add are not fully random, how does NEAT perform and behave?”*
To do so, the proposal is to integrate inputs preprocessed through an Echo State Network (ESN) into the initial set; ESNs are a particular type of recurrent neural network, whose output is related to the given input.
Since such preprocessing could provide a more structured and informative input space compared to complete randomness, the expectation is that in spite of a potential performance degradation as the solutions space expands, the outcome may be better when compared to first scenario.

The expectation is to get comparable results to NEAT with the original input set.

The last question this paper aims to address is *"What are the implications on the results when NEAT is exclusively fed with data processed by the ESN? Can the ESN effectively reduce the dimensionality of the input space?"*
This third question arises from the interest in the exclusive utilization of ESN-processed data. It is posited that this approach may provide unique insights into the algorithm's behavior when confronted with a more refined and dimensionally reduced input space. This question also addresses the potential of ESN to act as a tool for input space dimensionality reduction, thereby contributing to a deeper understanding of ESN's role in enhancing NEAT's performance.
The idea to use ESNs as dimensionality reduction tools arises from the fact that they are recurrent neural networks (RNN) known for their ability to process sequential data. The expectation is that when applied to dimensionality reduction, ESNs can capture the essential patterns in high-dimensional data while reducing its overall dimensionality.

II Background

ANN [14]
Artificial Neural Networks (ANN) are computational processing structures similar in structure and function to biological nervous systems; they are very powerful tools for control and prediction able to approximate every continuous function. They are often called Neural Networks (NN).
Neurons are called nodes, and they are the fundamental processing unit. To form a full network many neurons are connected through weighted connections. Any network can be seen in terms of layers:

1. An input layer that receives inputs from the external world, these neurons pass their activation forward over weighted connections to other neurons.
2. The output layer is the last layer in a neural network, its output is used by the system containing the neural network.
3. In between the input and output layers are hidden neurons, which are in "hidden layers." They do not have to be in strict layers.

Each neuron j, given an input vector x has output:

$$y_j = \sigma\left(\sum w_{ij} x_i\right)$$

Where σ is the activation function, whose aim is to squash the value into a range between 0 and 1. If the activation starts at the inputs and flows to the outputs it is called feed-forward network, otherwise if the networks have feedback connections that go backward, they are called recurrent. This second type of networks is useful for learning temporal dependencies. Recurrent networks can form loops and cycles that give a "memory" to the network: information is propagated not only forward but also backward, and this can create and maintain an internal state that allows it to exhibit dynamic temporal properties and keep a memory of past events.
Feed-forward networks instead "live in the present": they do not recall any past information. An example of this type of network is the perceptron, an ANN with exclusively direct feed-forward connections between the inputs and the outputs and where the activation function is a simple function like the identity or the step one []; the simplicity of this network has however a drawback: it can only solve linearly separable problems. More complex multilayer architectures with hidden nodes can learn to solve more complex problems, and in fact approximate any function to a given accuracy given a large enough number of hidden neurons.
Neural networks are often trained using gradient descent methods like backpropagation [2], the drawback of such methods is that they can get stacked in a local minimum and they do not work well in sparse reinforcement domains, and they require output target. A valid option to this, as it will be explained later, is neuroevolution. Usually, training recurrent networks takes longer and is less reliable than training feedforward networks.

ESN [8,9]
Echo state networks are a type of fast and efficient recurrent neural network. A typical ESN consists of an input layer, a recurrent layer (called reservoir, with a large number of sparsely connected neurons), and an output layer. The connection weights of the input layer and the reservoir layer are fixed after initialization. The extreme training efficiency of the ESN approach derives from the fact that only the weights from the reservoir to the output layer are trained, while the weights of input and reservoir are initialized under certain conditions [8] and then are left untrained.
Often used as an alternative to gradient descent training based RNNs, Echo State Network (ESN) [6] is one of the key Reservoir computing (RC).
ESNs employ the multiple high-dimensional projection in the large number of states of the reservoir with strong nonlinear mapping capabilities, to capture the dynamics of the input. [6] An important property for ESN performance is the Echo State Property (ESP), which ensures that the impact of initial conditions diminishes over time. This property states that the states of reservoir should asymptotically depend only on the driving input signal, which means the state is an echo of the input, meanwhile, and the influence of initial conditions should progressively vanish with time. The inputs with more similar short-term history will evoke closer echo states, which ensure the dynamical stability of the reservoir. ESNs are suitable for process-chaotic time series.
In an ESN, u(t) ∈ R^{Dx1} is the input value at time t, x(t) ∈ R^{Nx1} is the state of the reservoir at time t and y(t) ∈ R^{Mx1} is the output value. W_{in}∈ R^{NxD} is the matrix representing connection weights between the input layer and the hidden layer, W_{res}∈ R^{NxN} is the matrix of the connection weights in hidden layer. Given f a nonlinear function, the state transition equation is:

$$x(t) = f\left(W_{in} * u(t) + W_{res} * x(t-1)\right)$$
$$y(t) = W_{out} * x(t)$$

Wres weights change accordingly to the following equation:

W'res = Wres + WinWout

Only parameters Wout are subject to training, they can be trained using a closed-form solution by extremely fast algorithms such as ridge regression

The Spectral Radius, ρ(A), is a mathematical measure of a square matrix A's largest absolute eigenvalue. In the context of ESNs, the spectral radius of the reservoir's weight matrix plays a pivotal role. A spectral radius less than 1 typically leads to stable dynamics, preventing unbounded growth of activations. Conversely, a spectral radius exceeding 1 may result in potentially unstable dynamics. Proper adjustment of the spectral radius is crucial for optimizing ESN behavior, influencing stability and the manifestation of the Echo State Property. All the experiments part of this study will be conducted using a spectral Radius below one.

The apparent simplicity of ESNs can be deceptive. Successfully applying ESNs needs some experience and understanding the correct parameters to use can result complex.

Neuroevolution [1,2,3,4,5]

Neuroevolution is a subfield of artificial intelligence (AI) and machine learning that involves evolving artificial neural networks (ANNs) using evolutionary algorithms. The primary goal is to optimize the structure and weights of neural networks to perform specific tasks. This approach draws inspiration from natural evolution, where genetic algorithms drive the adaptation and selection of individuals in a population over successive generations. It uses evolutionary principles such as mutation, crossover, and selection to improve neural network performances. Instead of relying on handcrafted architectures or learning through backpropagation, neuroevolution explores the evolutionary search space to discover effective neural network structures and parameter configurations; it can be used for supervised, unsupervised and reinforcement learning tasks. Neuroevolution holds significance in its capacity to discover novel neural network architectures and configurations, potentially outperforming handcrafted designs. It is particularly valuable in scenarios where the optimal network structure is not known a priori, and the search space is big and complex. Any neuroevolution algorithm has this structure: a population of genotypes that encode ANNs is evolved to find a network (weight and/or topology) that can solve a computational problem on which each network is tested. The performance or fitness obtained in this way is recorded and once the fitness values for all the networks in the current population are determined, a new population is generated starting from the best individuals of the previous population by mutating them, or by combining them (crossover). The new networks obtained in this way replace genotypes with lower fitness values, thereby forming a new population. This loop is typically repeated hundreds or thousands of times, to find better performing networks.

However, it is important to highlight that neuroevolution is a black box method with the following drawback: human cannot easily work out what the evolved network do by looking at them. This can potentially be a problem of quality assurance, as it becomes very hard to understand and debug learned behavior.

NEAT [6,7]

In this project, neuroevolutionary algorithm NEAT has been used; it is designed for evolving artificial neural networks (ANNs) with both structural and weight modifications. NEAT was introduced to address and solve the challenge of evolving neural network topologies effectively. The algorithm dynamically adapts the structure of neural networks over successive generations, allowing for the exploration of a diverse set of architectures. It is based on:

- *Historical Markings*: they serve to identify genes with the same origin. NEAT maintains a historical record of structural innovations, including the addition and removal of nodes and connections in the evolving neural networks. Each structural change is assigned a historical marker, enabling the algorithm to track the lineage of innovations.
- *Speciation*: a mechanism that groups individuals into species based on the similarity of neural network topologies. This promotes diversity within the population safeguarding newly introduced structural innovations by providing them with the necessary time to undergo optimization before competing with other niches in the population and prevents premature convergence, allowing different neural network structures to coexist and evolve independently. It uses a compatibility distance metric to quantify the similarity between two neural networks, it is used to classify the networks into species. This metric considers both structural and parametric differences, facilitating the comparison of networks with varying architectures and weights.

NEAT starts with a population of simple neural networks with some specified characteristics and evolve and augment the networks over generations. This process allows the algorithm to discover novel network topologies that effectively capture the intricate relationships between the observation of the environment and the action to take to get a higher fitness.

NEAT defines neural network through two concepts:

- The *genotype*, which is the genetic encoding of the neural network. It includes information about nodes and connections, and each connection is assigned a unique ID. Nodes are classified as either input, output, or hidden.
- The *phenotype*, which represents the actual neural network structure and functionality. It is derived from the genotype through a process of decoding, where the encoded genes determine the existence of nodes and connections in the actual neural network.

An initial population of potential solutions (neural networks) is created randomly, starting from a given configuration. Each solution represents a set of parameters or characteristics that define a potential solution to the problem. The number of inputs corresponds to the information the neural network receives about the environment. The number of outputs is fixed and produced by the neural network; those dictate the actions the agent can take. Each individual in the population is evaluated using a fitness function, which quantifies how well the solution solves the given problem. The fitness function guides the algorithm by assigning a numerical score to each solution based on its performance. Individuals with the best fitness are selected from the population to serve as parents for the next generation. Pairs of selected individuals undergo crossover, where their genetic material (parameters or characteristics) is exchanged to create new offspring (mimicking the process of genetic recombination in biological reproduction). Random changes are introduced to some individuals' genetic material. During these operations more nodes and connections can be added, adapting in this way the complexity of the network; this allows to introduce diversity in the population and explore new areas of the solution space. The new offspring, with a predefined number of individuals from the previous generation, form the next generation. The least fit individuals may be replaced by the new offspring, ensuring a constant population size. The algorithm iterates through the steps of selection, crossover, mutation, and replacement for multiple generations, or until a termination criterion is met (for example a termination criterion can be reaching a specified number of generations or achieving a satisfactory solution).

NEAT offers the possibility to evolve both feed forward and recurrent neural networks.

Related works

In this project we will try to improve the performances of NEAT using Echo State Networks. Despite the scarcity of contributions in the literature adopting the same approach, many studies on NEAT ability to solve complex tasks, like in [12, 3] and its efficiency compared to other neuroevolutionary algorithms [7]. Often the results obtained by NEAT are compared to the ones achievable with Reinforcement Learning techniques. Moreover, no literature on the use of ESNs as input reduction tool was found as the literature around ESNs focuses on using them as a tool for predictions [8,9].

Therefore, the efficiency of NEAT compared to other reinforcement learning algorithms has been assessed, but never combined with ESN.

III Benchmark mechanics

The Cart Pole environment represents a classical problem in reinforcement learning, where an agent is tasked with balancing a pole affixed to a movable cart. The environment encompasses continuous state spaces, including 4 parameters: the cart's position, velocity, pole angle, and pole angular velocity. Rewards are contingent upon the agent's capacity to sustain the pole in an upright position. Episodes conclude either upon the pole exceeding a predetermined angle or the cart moving beyond a specified range or when the reward gets to 500. The application of AI, specifically reinforcement learning, is needed for addressing the Cart Pole challenge. The nature of the problem necessitates the agent to acquire a policy to maintain the pole equilibrium. Through iterative exploration of various actions in diverse states, the AI agent discerns optimal actions that get maximal cumulative rewards, thereby mastering an effective strategy for pole balance.

The Bipedal Walker environment is characterized by a two-legged robot controlled by the agent, analogous to the Cart Pole scenario. The goal is to proficiently navigate the environment starting at the left end of the terrain with the hull horizontal and both legs in the same position with a slight knee angle; the state space encompasses information about the robot's position, velocity, joint angles, and angular velocities (the observation of the environment is made of 24 signals). The agent's actions involve controlling the robot's joints to facilitate movement (you have to pick 4 actions, one per joint). Rewards are contingent upon the robot's ability to advance while maintaining a stable posture, totaling 300+ points up to the far end and losing 100 points if the robot falls. Applying motor torque costs a small number of points. The episode will terminate if the hull gets in contact with the ground or if the walker exceeds the right end of the terrain length so if it gets 300 points. The Bipedal Walker challenge necessitates the application of AI, specifically reinforcement learning, due to the intricate control and decision-making requirements within continuous state and action spaces. Traditional methodologies may encounter difficulties given the high-dimensional and dynamic nature of the task. AI algorithms, leveraging reinforcement learning principles, excel in learning optimal policies for the robot to navigate the environment adeptly, showcasing adaptability and robust learning capabilities.

This environment is significantly more complex than Cart Pole, due to the increased degree of freedom and coordination needed: Cart pole system has two major degrees of freedom – horizontal movement of the cart and angular movement of the pole; on the other hand, a bipedal walker locomotion involves coordinating multiple joints and limbs to maintain balance and move forward. This requires the control of various degrees of freedom, including hip, knee, and ankle joints, each contributing to the overall stability and dynamic movement of the walker.

IV Methods

To evolve neural networks for solving some environment from the OpenAI Gym using NEAT the following methodology has been proposed:

1. Define the evaluation function to assess the fitness of each genome (neural network) in the population. This function will be called during each generation to determine how well each neural network performs in the given environment. In this function for each genome in the population, a neural network (net) is created using the NEAT function *neat.nn.FeedForwardNetwork.create*. This function takes the genome and the NEAT configuration as inputs and produces a feedforward neural network. Each neural network is evaluated in the environment for a specified number of runs.

   A new environment is created, the observation is reset, and the fitness is set to zero.

   Afterwards, the following loop starts:

- The neural network is activated based on the current observation of the environment and eventual additional inputs (or using directly the environment processed information) to determine the action.

- The action is applied to the environment, and the reward is obtained and summed to the fitness of the episode.
- The loop continues until the episode is finished. The total reward obtained is recorded in the fitness value that reflects how well the neural network performed in the task.

2. The second step is the configuration setup. Through a configuration file the hyperparameters of NEAT algorithm are set up.
3. The population is initialized, and the reporters are set up to be able to see the statistics.
4. The last step is the Evolutionary Run: the run command executes the NEAT algorithm for a specified number n of generations. The winning genome (the one with the highest fitness) is stored in the winner variable.

V Results

The initial phase of this study focuses on assessing the convergence speed of the NEAT algorithm when additional random inputs are introduced. Three distinct types of initial connections were examined: unconnected, where no connections are initially present; fully connected, involving connections between each input node with all hidden and output nodes, and between each hidden node with all output nodes; partially connected, where each connection has a 0.5 probability of being present.

In this analysis, certain hyperparameters have been held constant, including initializing the number of hidden nodes to zero due to the relatively straightforward nature of the CartPole environment. The population size has been fixed at 250 individuals, with a maximum stagnation threshold set to 20 generations (species exhibiting no improvement for more than 20 generations were deemed stagnant and removed). Preservation of the five most fit individuals in each species from one generation to the next has been established, along with a fraction of 0.2 representing the proportion of each species allowed to reproduce each generation. A maximum genomic distance of 3 has been set to determine whether two individuals belong to the same species. The activation function chosen has been "clamped," and a 0.2 probability for adding or deleting a node or connection has been set. Additionally, a 0.5 probability has been assigned for mutations to change the weight of a connection by adding a random value, as well as a 0.1 probability for mutations to replace the weight of a connection with a newly chosen random value, simulating the introduction of a new connection. Furthermore, it has been chosen to evolve feed forward neural network instead of recurrent ones since Cart Pole is a relatively easy game.

With those parameters set 100 runs for each scenario have been analyzed, and the results collected are shown in Figure1.

The y-axis represents the number of generations, revealing an exponential increase in the generations required to achieve a perfect solution (500 points) for the Cart-Pole task as the number of inputs expands. This conjecture is grounded in the idea that an escalating number of inputs results in an exponential growth in the potential neural network configurations. Consequently, to comprehensively

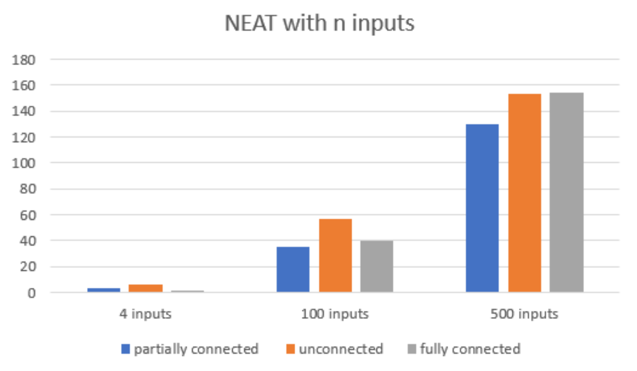

Figure 1, NEAT with the addition of random inputs

explore all feasible combinations necessary for attaining the desired outcome, a substantially greater number of generations is needed, especially because only a few inputs give useful information, and the algorithm has to figure out who they are. The average generation time is 1.0 second for the 4 inputs scenario, 1.2 seconds for the 100 inputs and 3.5 seconds for the 500.

In analyzing the results of the 500-input and 100-input problems, a distinct pattern emerges regarding the utilization of random inputs among the different network configurations. Notably, the fully connected solution consistently incorporates the highest average number of random inputs, followed by the partially connected configuration, while the unconnected setup employs the least number of random inputs. This intriguing observation prompts an exploration of the underlying factors influencing these trends. One plausible explanation lies in the adaptability and versatility of the neural networks evolved under each architecture. The fully connected network, starting with its extensive interconnections, exhibits a heightened capacity for accommodating a wide array of input scenarios, this is also probably due to the low probability of deleting a node set in the configuration file. This could be advantageous in scenarios where the complexity introduced by the increased number of inputs is relevant for the solution of the problem. On the other hand, the partially connected architecture strikes a balance, utilizing a moderate number of connections to achieve a nuanced responsiveness to random inputs. Conversely, the unconnected setup, characterized by minimal connections, probably also due to the low probability of adding connections, may face limitations in effectively integrating diverse inputs, potentially hindering its performance especially in cases where the all the inputs give useful information; however, in cases like this one where the majority of the inputs are irrelevant, this configuration finds the simplest network which includes the least amount of irrelevant inputs.

In this context where only a limited set of inputs holds significant information, an intriguing finding surfaces when we test the winner: the neural network derived from the unconnected setup exhibits, on average, superior performance compared to its counterparts in the fully connected and partially connected configurations. This phenomenon is hypothesized to be attributed to the number of irrelevant inputs considered by the network.

The second analysis delves into a scenario where additional inputs are not entirely random but instead stem from the processing of environmental inputs through an Echo State Network (ESN). In configuring the ESN, the spectral radius was set at 0.9, the connectivity at 0.1, and the reservoir size has been set to 10. This exploration aims to discern how incorporating inputs derived from an ESN, a more structured and contextually informed source, influences the performance dynamics of evolved neural networks. It is important to highlight that the ESN here has not been trained, since it was not supposed to be used to make predictions but only as a tool to create signals related to the environment output.

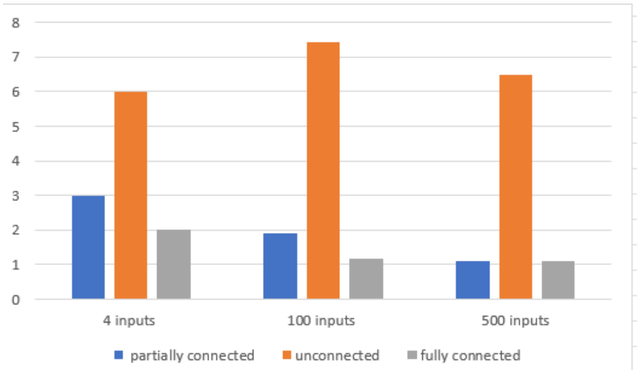

Figure 2: NEAT with combined inputs

As initially supposed, NEAT exhibits a notable performance improvement compared to its counterpart scenario involving random inputs. In this context, the convergence dynamics for both 100 and 500 inputs are strikingly comparable, in terms of speed, to the scenario with only 4 inputs sourced from the environment. The average generation time is 1 second for the 4 inputs, 1.2 for the 100 and 5.3 for 500. Examining the outcomes across various setups, a distinctive observation emerges: the winning neural network from the unconnected configuration is considerably simpler than its counterparts in the partially connected and fully connected setups. Specifically, in the 500 inputs scenario, the unconnected network averages only three connections, whereas the partially connected and fully connected counterparts comprise 250 and 500 connections, respectively. This highlights a tradeoff between the speed of convergence and the complexity, and so the readability, of the resultant neural network in a scenario where all inputs provide significant information, even if redundantly.

Let’s analyze now what happens when NEAT is fed only with inputs preprocessed through the ESN. The connectivity has been deliberately set at 0.1, the reservoir size at 10 and the spectral radius at 0.9.
Surprisingly, the results shown in Figure3 of the experiment reveal an unexpected trend: as the number of inputs increases, all three versions of NEAT require fewer generations to achieve the desired outcome. One plausible explanation for this phenomenon is that, in the scenario with only four inputs, the reservoir size is larger than the output size. This disparity potentially renders the output more challenging for NEAT to converge upon a solution. To validate this hypothesis, an additional experiment has been conducted with a smaller and smaller reservoir size down to 2. However, the outcomes turned out to be consistently

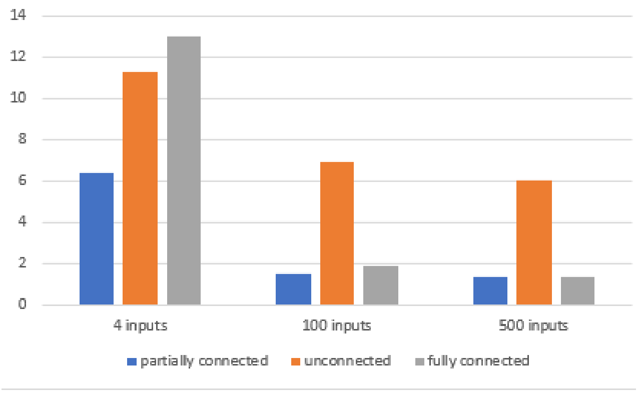

Figure 3: NEAT with ESN-preprocessed inputs

poorer across all configurations. Conversely, when the reservoir size has been increased to 20, the number of generations required for convergence notably decreased, resulting in an improved overall performance. This suggests that the relationship between reservoir size and output complexity plays a crucial role in the convergence speed of NEAT in scenarios where inputs are preprocessed through an ESN. The average generation time is 0.8 seconds for 4 inputs, 1.8 seconds for 100 inputs and 5.2 seconds for the 500.
Despite the intriguing trend of improved convergence with increasing inputs, it's noteworthy that the results are not favorable when compared to NEAT fed with the combination of the original input and the ESN output. This observation aligns with the pattern seen in many winning solutions, where direct input from CartPole was frequently utilized. Evidently, the synergy between the raw environmental input and the ESN-processed input proved to be more effective in fostering successful neural network architectures.

Similar experiments have been conducted on a second OpenAI game: Bipedal Walker. This environment is much more complex than the previous one, however the results NEAT gets are pretty good. The hyperparameters of the Cart Pole experiments have been maintained with a few exceptions: threshold is set to 300 (the maximum fitness achievable), and the probabilities of adding and deleting connections and nodes have been raised to 30%.
Afterwards, initial connection has been set as partial direct with probability 0.5. The results obtained are showed in Figure4 as an average over 10 runs with a population of 250 networks.

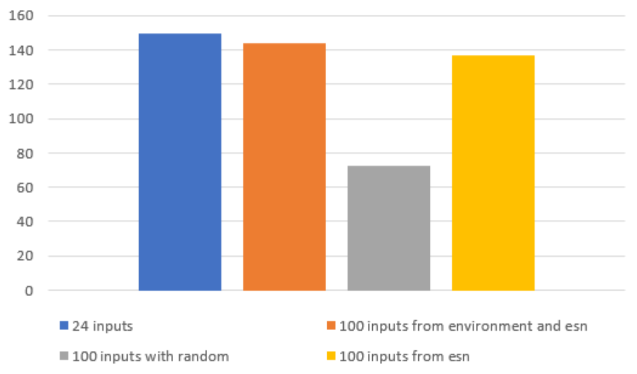

Figure 4: results of NEAT on Bipedal walker

What found is aligned with the results seen so far with Cart Pole: the worst results are the ones where random inputs have been added followed by the use of input coming only from ESN. The results of the other two scenarios are instead comparable, the only difference being the complexity of the network obtained: using only inputs coming directly from the environment originates a simpler and more readable network while when both inputs from the environment and the ESN are used the complexity of the network is much higher making it more difficult to read.

The last experiment conducted is an attempt to use ESN as an input reduction tool using Cart Pole. Starting from the previous scenario where random values were added to the observation coming from the environment, those inputs have been processed by an ESN with input size 100, reservoir size 70 and output size 20 and then used by NEAT to pick the action. Many attempts to change the ESN parameters have been made but without significant improvements. The speed convergence of NEAT is considerably worse, taking more than 100 generations to converge to a solution. Similar results have been obtained when trying the same experiment starting from a combination of inputs coming from the environment and the same inputs processed through a second ESN.

VI Discussion

The results of the first experiment highlight the importance of choosing the right configuration between fully connected, partially connected and unconnected in a setup where the relevance of the input variables to solve the problem is uncertain. In these situations, the results suggest the use of the unconnected setup, resulting in a simpler, more readable network that performs on average better than the other two setups. The explanation is in the number of irrelevant inputs considered by the network: the best performer is the one from the unconnected setup which is the simpler among the three networks and the one that considers the least number of random inputs, so it works as a feature selection mechanism.
On the other hand, in the opposite situation where there is a certain confidence on the input relevance, in this study analyzed in the second and third experiment, it has been shown that the other two setups are faster than the unconnected one. The fully connected one is slightly faster but giving on average more complex and less human readable networks.

The use of ESN to improve the performance of NEAT does not meet the expectations. Among all the input options tested the one with better performances is the use of only inputs coming directly from the environment, both for Cart Pole and for Bipedal Walker. From the theoretical point of view this can be explained with the fact that the additional inputs added do not give new information: they are a re-elaboration of inputs the NEAT algorithm already has from the environment. The initial expectation was that NEAT would have been able to recognize patterns in a faster way if more preprocessed inputs were given. Even if such were the case, there would still have been a trade-off between how quickly the algorithm converged and how readable the resulting network would have been: having more inputs signals means having more possible connections, and since those signals are preprocessed, it would not be clear what they represent, making the overall network less and less human readable. Even the use of only information coming from the environment preprocessed with ESN has showed poor performance results. For future works, it could be interesting to preprocess the information coming from the environment with different types of neural networks and analyze if the performances improve somehow.

The last relevant observation lies in the use of ESNs as a dimensionality reduction tool. They are a suitable reduction tool according to the literature; however, the experiment reveals an outcome below expectations. Nonetheless, it is important to note that the effectiveness of ESNs as a dimensionality reduction tool depends on the specific characteristics of the data and the task at hand. Overall this paper suggest that the use of ESNs as dimensionality reduction tool is not recommended.

References

[1] Sebastian Risi, Julian Togelius, “Neuroevolution in Games: State of the Art and Open Challenges.

[2] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, Ilya Sutskever, “Evolution Strategies as a Scalable Alternative to Reinforcement Learning”, OpenAI

[3] M. Hausknecht, J. Lehman, R. Miikkulainen, and P. Stone. “A neuroevolution approach to general Atari game playing”, IEEE Transactions on Computational Intelligence and AI in Games, 2013.

[4] K. O. Stanley, J. Clune, J.Lehman, R. Mikkulainen, “Designing neural networks through neuroevolution”

[5] D. Floreano, P. Durr, and C. Mattiussi. “Neuroevolution: from architectures to learning”.

[6] K. O. Stanley, R. Mikkulainen, “Evolving Neural Network through Augmenting topologies”

[7] K. O. Stanley, R. Mikkulainen, “Efficient Evolution of Neural Network Topologies”.

[8] Chenxi Sun, Moxian Song, Shenda Hong and Hongyan Li, “A review of design and applications of echo state networks”

[9] Herbert Jaeger, “Adaptive Nonlinear System Identification with Echo State Networks”

[10] H. Jaeger, I. B. Yildiz, S. J. Kiebel, “Re-visiting the echo state network property”

[11] J. Schmidhuber, J. Zhao, “direct policy search and uncertain policy evaluation”, 1998

[12] Mark Wittkamp, Luigi Barone and Philip Hingston, “Using NEAT for Continuous Adaptation and Teamwork Formation in Pacman”

[13] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, Demis Hassabis,

"Mastering the game of Go with deep neural networks and tree search"

[14] Kenneth Owen Stanley , "Efficient Evolution of Neural Networks through Complexification"