Learning to Walk - Reward relevance within an enhanced Neuroevolution approach

I. Colucci
NCLab, DIEM, University of Salerno, Salerno, ITALY
i.colucci2@studenti.unisa.it

G. Pellegrino
NCLab, DIEM, University of Salerno, Salerno, ITALY
g.pellegrino27@studenti.unisa.it

A. Della Cioppa*
NCLab, DIEM, University of Salerno, Salerno, ITALY
adellacioppa@unisa.it

A. Marcelli
NCLab, DIEM, University of Salerno, Salerno, ITALY
amarcelli@unisa.it

ABSTRACT

Recent advances in human motion sensing technologies and machine learning have enhanced the potential of AI to simulate artificial agents exhibiting human-like movements. Human movements are typically explored via experimental recordings with the aim of establishing relationships between neural and mechanical activities. A recent trend in AI research shows that AI algorithms work remarkably well when combined with sufficient computing resources and data. One common criticism is that all of these methods are gradient-based, which involves a gradient approximation, thus suffering of the well-known vanishing gradient problem.

In this paper, the goal is to build an ANN-based controller that enables an agent to walk in a human-like way. In particular, the proposed methodology is based on a new approach to Neuroevolution based on NeuroEvolution of Augmenting Topologies (NEAT). The original algorithm has been endowed with a different type of selection-reproduction mechanism and a modified management of the population, with the aim to improve the performance and to reduce the computational effort. Experiments have evidenced the effectiveness of the proposed approach and have highlighted the interdependence among three key aspects: the reward framework, the Evolutionary Algorithm chosen and the hyper-parameters' configuration. As a consequence, none of the above aspects can be ignored and their balancing is crucial for achieving suitable results and good performance.

ACM Reference Format:
I. Colucci, G. Pellegrino, A. Della Cioppa, and A. Marcelli. 2020. Learning to Walk - Reward relevance within an enhanced Neuroevolution approach. In *Genetic and Evolutionary Computation Conference Companion (GECCO '20 Companion), July 8–12, 2020, Cancún, Mexico.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3377929.3398126

*Also with Institute for High–Performance Computing and Networking, CNR, Naples, ITALY.

GECCO '20 Companion, July 8–12, 2020, Cancún, Mexico

ACM ISBN 978-1-4503-7127-8/20/07.
https://doi.org/10.1145/3377929.3398126

1 INTRODUCTION

Artificial Intelligence (AI), which includes the fields of Machine Learning, Natural Language Processing and Robotics, can be applied to almost all fields of healthcare [27], and its potential contributions to biomedical research, medical training and healthcare delivery seem limitless. With its solid ability to integrate and learn from large series of clinical data, AI can fill roles in diagnosis [1], clinical decision-making processes [4], and personalized medicine [8].

One of the most challenging application areas is the simulation of artificial agents exhibiting human-like movements [22]. Human movements are typically explored via experimental recordings with the aim of establishing relationships between neural and mechanical activities. Such knowledge is then used to built AI agents exhibiting similar movements. Most of the solutions proposed are based on Artificial Neural Networks (ANNs) trained by using gradient-based algorithms. Anyway, how the brain controls the voluntary movements involved in the walk into the space is still an open question [14].

The AI success in solving such task is mainly due to the use of ANNs, in general, and Deep Learning, in particular [2]. However, applying an ANN to a given problem depends on how the artificial neurons interact with each other and their functional interconnections are arranged. Moreover, a paramount factor is represented by the type of mechanism by means of which the ANN actually learns to solve a problem. Reinforcement Learning (RL) is a type of learning that recently has attracted an increasing interest [19, 23]. RL has its roots in behavioural psychology where a learner performs an action based on a series of rewards or punishments [21]. This learning process is based on the idea that good behavior is rewarded and, therefore, is encouraged to be repeated, while bad behavior is punished or penalized and must eventually stop [21]. As a consequence, maximizing the reward is a central theme in RL.

A recent trend in AI research, in general, and Machine Learning, in particular, is that AI algorithms work remarkably well when combined with sufficient computational resources and data. That has been the story for (1) back-propagation applied to deep neural networks in supervised learning tasks such as computer vision [16] and voice recognition [34], (2) back-propagation for deep neural networks combined with traditional RL algorithms, such as Q-learning [13] or Policy Gradient (PG) methods [33], and (3) Evolution Strategies (ES) applied to RL benchmarks [28]. One common criticism is that all of these methods are gradient-based, including ES, which involves a gradient approximation similar to finite differences [29]. In fact, it is well known that algorithms using

gradient approximations have significant limitations when dealing with multimodal and non-differentiable cost functions. Moreover, they are particularly inefficient and time-consuming when dealing with problems with a huge number of local optima and many plateaus on the error landscape [12]. These criticisms raise the question of whether a similar story will play out for gradient-free methods, such as population-based Evolutionary Algorithms (EAs).

From the analysis of the solutions proposed in literature and of the rationale behind them, it results that most researchers use either canonical approaches [15], such as RL algorithms like Deep Deterministic Policy Gradient (DDPG) [24] and Proximal Policy Optimization (PPO) [31], both gradient-based, or a specific neural circuitry that is able to control human locomotion behavior [37] or a mathematical model such as a truncated Fourier series [7].

A totally different approach, which could be a breakthrough in building research and does not make any use of any kind of predefined model, is presented here with the aim to evolve an ANN that will enable a model to walk as far as possible in a human-like way. In order to produce a human brain-like controller that could make an agent move, it is necessary to choose an environment to work with. The interface adopted for the environments is the one provided by Gym [25], realized by OpenAI. The chosen environment is *Walker2D* from MuJoCo [26]. The proposed methodology is based on a new approach to Neuroevolution: NeuroEvolution of Augmenting Topologies (NEAT), characterized by explicitly evolving the topology and the weights of ANNs. NEAT's original algorithm has been endowed with a different type of selection-reproduction mechanisms and a modified management of the population, with the aim to improve the performance and to reduce the computational effort. Moreover, different kind of rewards have been designed. The different reward frameworks tested highlight the importance of the reward for a RL algorithm and the key characteristics for the realization of a performance measure suitable for an EA. The results achieved on the methodological and configuration part show a higher efficiency with respect to the canonical version of NEAT. Such improvements relate to the selection-reproductive mechanism and population management over the generations. Moreover, the methodology presented has proven to be valid for the problem under consideration on different simulation environments, such as [18]. In addition, based on exploratory experiments on [17], we observed that it is applicable to different kind of problems.

The paper is organized as follows: Section 2 revisits related work; Section 3 presents the fitness function and the algorithm improvements; Section 4 sets out the results and, finally, in Section 6 some conclusions are drawn.

2 RELATED WORKS

In the following, we report and discuss the most relevant contributions to the state of art on how to walk on the Walker2D environment. It is based on the Gym interface provided by OpenAI, that is a toolkit for developing and comparing RL algorithms. Several algorithms have been used by researchers and some are reported by S. R. Lee in his website [35]. All the ones reported by Lee are enhanced versions of the same RL algorithms: DDPG [20, 36], PPO [31] and Trust Region Policy Optimization (TRPO) [32]. They are all gradient-based, which means that they cannot be easily parallelized, because there is just one agent that interacts with the environment, unless specific provision are implemented as shown in [15]. Moreover, they all start with a general architecture and then proceed with weight training and parameter tuning. However, the chosen architecture may be either too complex for the task (leading to a waste of computational resources) or too simple (failing to resolve the task).

In addition to canonical RL algorithms, there are two approaches worth to be mentioned: ES, as an alternative to gradient descend, and Imitation Learning (IL) algorithms. ES are an optimization technique that basically resembles simple hill-climbing in a high-dimensional space. With respect to canonical RL algorithms, they are simpler to implement, have fewer hyper-parameters to be set, don't need backpropagation and are highly parallelizable because weights are randomly mutated. In a comparison between ES and canonical RL algorithms [28], the OpenAI team concluded that they are a good solution to avoid gradient explosion in gradient-based algorithms, but still need a model that is difficult to be chosen for the given task and ,likely, it will be more complex than the one Neuroevolution is able to obtain.

IL is a form of Supervised Learning applied to task that are typically solved with RL approaches. In particular, two projects are remarkable: *Humanoid Imitation Learning from Diverse Sources* [10] and *Imitiation Learning and Mujoco* [6]. The former describes the implementation of a system that learns locomotion skills for a humanoid robot model using GAIL [11], that accepts as input three different models: a RL expert, motion capture data and real-life video. The latter, on the other hand, uses a simple training set for different MuJoCo RL expert models. From the projects presented it is clear that IL assumes that data on human walk are either available or there is the possibility to get them. Anyway, even if they are available, such approach can be seen as having a supervisor that teaches someone how to walk. When a baby learns how to move and finally walk, he does not have, in general, a teacher. In fact, although he learns by observing the movements of his parents, for example, he won't walk in the exact same way. Therefore, his parents will be around, helping him standing, but not all the time, so there will be moments in which he will try to walk alone [30]. Overall, as babies have to build the structure of brain's neural network from scratch, a Neuroevolution approach seems the most appropriate to simulate the learning process babies adopt.

Last but not least, the DeepMind group described how increasing the complexity of the environment promotes the learning of complex behaviors [9]. They simulated several models taken from MuJoCo on a diverse set of challenging terrains and obstacles, using a simple reward function based on forward progress. The discovery made by DeepMind is very interesting, although they had great resources available and they did not aim to enable models to walk in a human-like way. Instead, they just wanted models to overcome obstacles and difficulties they placed in the environment to uncover complex behaviors.

Overall, all the different techniques presented here, are either non compliant with human learning or lead to agents exhibiting an unnatural walk. To obtain a human-like walking, we adopt a reward function specifically designed to overcome these limitations, as it is explained in the next section.

3 LEARNING TO WALK

The canonical version of NEAT is able to evolve variable ANNs topologies by using two key mechanisms: i) an historical marking that allows to divide the population into species based on the topological similarity and to swap similar parts of two ANNs; ii) a population management mechanism that allows to start the evolution with a uniform population of simple ANNs with no hidden nodes, differing only in their initial random weights, and to gradually accumulate novel topologies over evolution thanks to speciation, truncation selection over all the population, elitism within species and elimination of stagnating species. In the following we will describe in detail how NEAT works, what are its main drawbacks and how we have enhanced some basic mechanisms.

3.1 The proposed method: enhanced NEAT

In order to design an enhanced version of the canonical Neuroevolution algorithm for evolving ANNs topologies, a python version of NEAT [38] has been considered, analysing its behavior and its performance in different environments. This version, that is the starting point of this research, will be detailed in the following. First, a population of new individuals (from now on individuals and genomes with reference to NEAT will be considered equivalent) is generated. Each individual is encoded using a genome, which encloses information about nodes and connections. In the evolutionary process after evaluating the individuals, they undergo to reproduction, thus creating the offspring population for the next generation. Reproduction takes place intra-species by means of the following mechanism: all the individuals are sorted according to their fitness and a certain percentage of them is selected for reproduction. The parents are typically chosen from the whole population in a completely random way. Subsequently, the new population is divided into species based on the genomic distance among the individuals according to both the criterion of the nearest neighbor and a threshold value to be exceeded for species change. By dividing into species, the topological diversity of the solutions is guaranteed. To ensure that each species contributes to the whole evolution, there is a stagnation system that keeps track of the species that are stagnating. As they reach a predefined stagnation threshold value, they are removed from the population. Removed individuals will be then replaced by offspring of a different species by using a distribution that is a function of the number of individuals in the species and the fitness of that species.

Selection. By default, NEAT uses a truncation selection mechanism. In truncation selection the candidate solutions are ranked by their fitness values and only the best $t\%$ individuals are selected as parents and reproduced $\frac{1}{t}$ times. Compared to the other selection methods of modeling natural selection, truncation selection is an artificial selection method and it is typically used when dealing with large populations. Selected parents produce offspring that depends by chance and by parents' fitness. The parameter for truncation selection is called *survival_threshold* and indicates the proportion of the population to be selected as parents. In general, it takes values ranging from 1% to 20%. Individuals below the truncation threshold do not produce offspring.

Another selection mechanism NEAT makes use of is the *Tour*-way tournament selection. According to such mechanism, *tour*-individuals are selected at random. Then, a tournament among them takes place and only the fittest candidate amongst them is chosen and is passed to the recombination and mutation mechanisms. Given that *Tour*-way tournament has proven to be more effective when dealing with small populations with respect to the truncation selection [3], we have decided to use the *Tour*-way tournament for the proposed method.

Population management. The population management of the canonical version of NEAT (referred to from now on as *canonical-NEAT*) consists in replacing the old population of individuals with the one obtained by the reproduction mechanism. This procedure can be indicated as of type (μ, λ)-NEAT, where μ indicates the starting individuals of the generic i-th generation and the ones to be selected for the next generation, while λ represents the number of offspring generated from μ individuals. So, if elitism is not present and the population size has to be preserved, μ and λ will coincide. In the event of elitism, the number of individuals to be kept in the population will be subtracted from λ. Consequently, in the next generation there are individuals of the old population, especially the best individuals of each species. Although there is elitism, the stagnation mechanism does not prevent the extinction of a species. Such event can take place also for the species containing the best individual, which means that the trend of the fitness of the best individual in the population, as the generations pass, is not a strictly ascending function. Hence, with this mechanism the average fitness of the population is a far cry from the fitness value of the best individual. In the same way, however, by regenerating new individuals from the previous ones and by replacing them directly in the next population the stagnation mechanism maintains a very low selective pressure. In order to improve the canonical-NEAT performance, we propose to manage the population differently. To this end, the (μ, λ) selection has been replaced with a modified $(\mu + \lambda)$ strategy [3].

The proposed approach is shown in the Figure 1, which reports the entire population generation process. The starting population (composed by μ individuals) is divided into species according to the topology. Intra-species reproduction is then carried out, generating a new offspring population, which are then evaluated. The starting population and the offspring population are merged according to a $(\mu + \lambda)$-strategy and the speciation procedure repeated. Each species is ordered according to the fitness and $T\%$ of the individuals of each species is selected by eliminating less suitable ones. So, a new population of μ individual is created, preserving diversity thanks to the splitting into species (different topologies are maintained).

Various versions of NEAT were tested before reaching this one. First, the idea was to compare the offspring population with the one made of parents and replace only in case of improved fitness. Second, after grouping offspring and parents into a single population, the best individuals from this set were chosen for the next generation. However, these solutions led to a loss of diversity of solutions too quickly and the selective pressure exerted is also high. In order to preserve topological changes and allow the weights to be updated for that topology, it has been introduced a mechanism that allows to maintain a buffer of individuals keeping track of the various

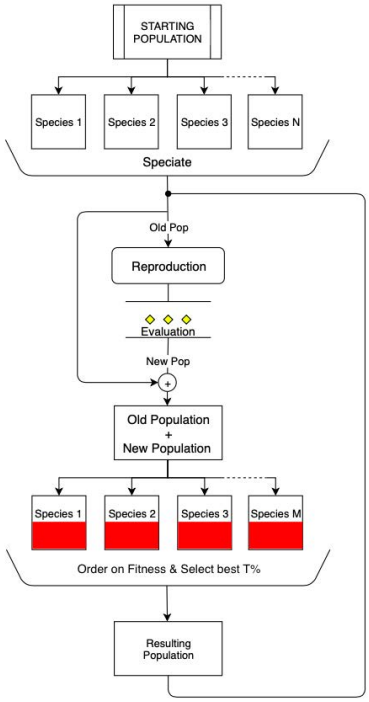

Figure 1: Proposed evolutionary scheme for NEAT: the diagram shows the basic algorithm components, including reproduction, $(\mu + \lambda)$-strategy and speciation.

offspring generated during generations. Hence, once a number b of generations is reached, the buffer is taken and speciation is performed. Finally, n_{ind} individuals are selected from the buffer just created, with the same criterion used in eNEAT (selecting $\frac{1}{b}$%).

The canonical-NEAT changes described above represent the enhanced version of NEAT, which from now on will be called *elite NEAT* (**eNEAT**) and it is listed in Algorithm 1.

3.2 Reward functions

Crafting reward functions for RL models is not easy for the same reason that crafting incentive plans for employees is not easy. When setting a reward function it is important to keep in mind that: *you get what you incentivize, not what you intend*. Typically, the fitness function for a RL problem is simply the sum of the rewards in the simulation time, but there are exceptions in which it can be represented by *a posteriori* processing the simulation. Reward is a feedback from the environment that incentives or disincentives a specific behavior. Hence, there are positive and negative rewards. When designing a reward, the way it is numerically structured can make a great difference. If it consists of only positive rewards, the system will try to accumulate as much as possible. This can lead to an interesting behavior: if, for instance, the simulation provides small rewards in sequence, then the agent will repeat that action forever, because its purpose is to get the highest possible reward. Generally, positive rewards promote the accumulation of them, thus generating the highest possible reward at the end of the task. Negative rewards are different. They encourage the agent

Algorithm 1: eNEAT Algorithm

```
generate an initial population of μ genotypes with no
  hidden layers;
evaluate their fitness;
apply a division into species;
repeat
    for each species in turn do
        compute the number of members of the species;
        repeat
            select parents using e% tournament selection;
            apply crossover;
            apply mutation to newly-created offspring;
        until required number of offspring are generated;
    evaluate their fitness;
    assign new individuals either to an existing species or
      to a new one;
    select individuals from each species;
    select only the best T% of the members of each species;
    select a random member of each species to represent
      that species;
    for each genotype in turn do
        compute the shared fitness of genotype;
    insert the current population into the next generation;
until terminating conditions are not met;
```

to complete the task as quickly as possible, to constantly avoid receiving penalties.

In literature, the most widely adopted reward for the walking task is the distance travelled by the agent, without any kind of penalty on the way the walking is performed. Starting from this consideration, we have designed several reward functions with the aim to promote a human-like walking. After a testing phase, the reward that has performed best on the walking task was the *Independent Leg* reward. It is essentially based on a distance measure, but it also takes into account some aspects that have proven to be relevant. In fact, for a proper walk it is necessary to have alternation, regularity and symmetry between the two legs. Obviously, taking into account such aspects is not an easy task. In particular, it results very difficult to introduce symmetry within the reward measure.

Algorithm 2 shows the pseudo-code of the reward function. The idea behind it lies in considering the two legs independent from each other. That is to say, the reward is computed according to the leg that dominates between the two (i.e., the forward one). If the constraints imposed in the single simulation step are respected, only the dominant leg will receive a reward. The reward will depend on the extra distance traveled (Δ), compared to the previous step, and the lope in the current step (d), as shown in the Figure 2.

The constraints to which the agent is subject are:

- to travel a positive distance from the previous step;
- to have at least one foot on the ground;
- to do not lift the legs above the torso.

Once the rewards are collected independently on the two legs, the fitness value associated with the individual will be computed as the sum of the two rewards on the legs minus the difference

Algorithm 2: Independent Leg Reward

$L_l = 0$, $L_r = 0$; $R = 0$;
repeat
  execute the current simulation step;
  compute the extra traveled distance Δ;
  compute the current lope d;
  if *distance covered is positive* **then**
    $\Delta = \min(\Delta, Th_r)$;
    if *legs are under the torso AND at least one foot is down* **then**
      modulate the lope with a Gaussian model based on the threshold lope value Th_l;
      $r = d * \Delta$;
      if *left leg is farther then right* **then**
        increment L_l with r;
      else
        increment L_r with r;
  $t = t + 1$;
until $t \leq t_{max}$;
return $R = L_l + L_r - |L_l - L_r|$

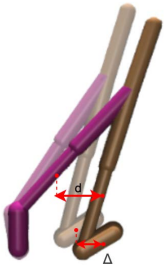

Figure 2: When all the constraints are satisfied, the independent leg reward equals to $d \cdot \Delta$.

between the two as follows:

$$R = L_l + L_r - |L_l - L_r| \quad (1)$$

where R is the total reward, L_l and L_r are the rewards gathered by left and right leg respectively during the simulation. Using R as fitness function leads evolution to optimize both the reward values since they identically affect the final fitness value, thus avoiding the agent to use one leg more than the other one to accomplish the task.

However, when considering only the reward independently on the legs, without placing constraints and using only the product between extra distance traveled and the lope, the agent is brought to run. The highlighted behavior is characterized by a posture very similar to a running agent, with high knees and a high flight time. This behavior is due to the fact that maximizing the distance between two successive footsteps is the same as increasing the speed. In addition, to receive more reward the agent tends to bring the knee forward, so as to increase the opening of the legs.

To avoid such running behaviour a saturation with a threshold value (Th_r) on the distance between two simulation steps is introduced to equally reward individuals with different speeds. Moreover, to avoid a lope too large, it has been weighed by an ideal value (Th_l) that considers its distance from a proper value

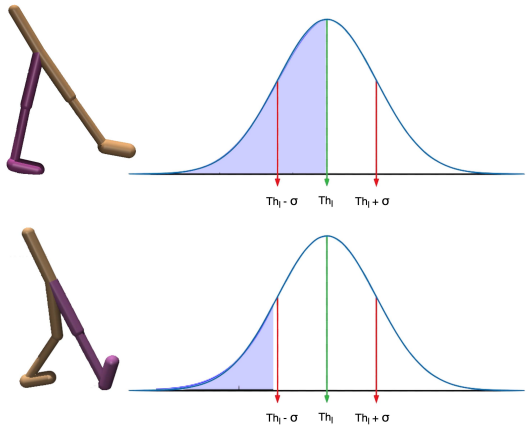

Figure 3: Threshold on the lope. Top image shows the ideal value for the lope, while bottom image shows a lower one.

which depends on the agent's height. It was chosen to modulate the lope by means of a Gaussian distribution (as shown in Figure 3), in such a way to have 1 at the desired value and gradually lower values away from it. The constraint on the positive distance avoids increasing the reward by standing indefinitely without walking, while the one on the feet punishes individuals with excessive flight time. Finally, to put a brake on unrealistic movements, a constraint was introduced on the height of the legs during the walk, so that they do not exceed the torso generating unnatural movements. In case at least one of the constraints is violated, the reward of the current simulation step will be lost.

4 RESULTS

4.1 Hyper-parameters' configuration

In order to choose proper values for the hyper-parameters the eNEAT algorithm depends to, we have effected a preliminary tuning phase. Since the number of hyper-parameters is large, we have divided such tuning phase in several steps.

NEAT algorithm parameters. The main hyper-parameters the canonical NEAT algorithm depends to are the size of the population, a stopping condition given by a threshold above or below which the solution produced by the evolution can be considered satisfactory. As regards the population size, from the simulations carried out, a number of individuals, in the population, less than 100 is inadequate, while values greater than 200 do not produce evolutionary advantages. So for the simulations, a population of 200 individuals was chosen. As regards the stopping condition, given that it is difficult to *a-priori* choose a specific value, it is set to a maximum number of evaluation equal to 20,000.

Stagnation and Reproduction parameters. The fundamental mechanism of NEAT is the division into species. Preliminary simulations showed that having the same species that drives evolution can lead to stagnation. As a consequence, to be in an evolutionary regimen, a minimum number of species needs to be maintained. The stagnation of a species can be defined either according to the best individual or to the average fitness of the species. In the first case, a species is stagnant if the best does not

increase its fitness value, while in the second case if the average fitness of the species does not improve. If a species is stagnant for more than a given number Th_s of generations, then it will be removed and their individuals will be distributed over other species. According to the eNEAT algorithm we can set the minimum number of species to keep in the population. In this case, even if the species is stagnant, it will not be eliminated. According to the results of the tuning phase we choose Th_s equal to 20 .

From a reproductive point of view, according to the tournament selection and the $(\mu + \lambda)$ population management method proposed by us, the only hyper-parameter to be set is the percentage value of individuals participating in the tournament. For it we have chosen $e\% = 10\%$. In order to bring the population back to the starting size, the $T\%$ parameter was chosen at 50%.

Genome parameters. Once network inputs and outputs are defined, the seeding parameters of the individuals are critical, such as the number of hidden neurons to be added and the starting network topology. In fact, given the inputs, hidden neurons and output neurons, these can be connected in several ways. A number of hidden neurons other than 0 can greatly influence the final result, depending on the initialization of individuals (n the worst case a lot of time is wasted removing neurons), while zero hidden neurons leaves to the algorithm the freedom to add them. Therefore, starting without connections is a good thing, but evolution takes a lot of time to add connections between input and output layers, similarly starting with a fully connected network implies the removal of some connections. The right compromise has been shown to be to start with half of all the possible active connections, randomly scattered for each individual. The remainder of the hyper-parameters regulate the nodes and the connections. With regards to the nodes, the key parameters are: the probabilities of adding-removing nodes and the set of activation functions to use. On the other hand, with regards to the connections: the probabilities of adding, removing and modifying the enabling of the connections. The odds of addition and removal, both of the nodes and of the connections, act as a learning rate for NEAT. A change in the structure of the network is unlikely to lead immediately to an improvement, because the value of the weights has to be adjusted. So after each structural change, a weight adjustment period is needed to decide whether this change is useful or not. This means that very high probability values lead to a worsening of evolution rather than improvements; i.e., the values used for the nodes are about 0.02, very different from the default values of 0.2 or 0.5. Similarly with regards to the activation functions, the adoption of the *clamped* function, within the set of functions, was a winning choice.

4.2 Simulation findings

In this section, the results of the simulations are presented. Both the two versions of NEAT use two hyper-parameters configurations: the one suggested in the canonical version, as in [38], and the new configuration obtained during the fine-tuning phase mentioned above. This approach leads to 3 different algorithms:

- **NC-NEAT**, the new tuned configuration with the canonical version of NEAT;
- **CC-eNEAT**, the NEAT canonical configuration with the elite version of NEAT;
- **NC-eNEAT**, the new tuned configuration with the elite version of NEAT.

In the following, **NC-NEAT** and **NC-eNEAT** are compared to highlight the best algorithm for the same type of hyper-parameters configuration. Subsequently, once eNEAT has been proven to lead to better performance, the comparison between the **CC-eNEAT** and **NC-eNEAT** is presented to determine which configuration leads to better performance. All the simulation effected are characterized by the following environment parameter settings:

- the maximum number of simulation-steps has been set to 5000;
- the frame-skip is set to 4, which means the controller chooses one action every 4 simulation-steps, so the maximum actual number of frames is 20000;

All the simulations will be analyzed in terms of the fitness trends of the average best individual and the diversity of the population. Diversity is computed using a k-Nearest Neighbors (KNN) algorithm, where k is equal to μ, in symbols:

$$S(x) = \frac{1}{n} \sum_{i=0}^{n} D(x, \mu_i) \tag{2}$$

where μ is the population size, μ_i the i-th member of the population and D is the Euclidean distance. Each individual has two phenotypic properties, which represent the coordinates of the individual in an euclidean space. The first one is the Independent Leg reward, explained in Section 3.2. The second one, instead, is the x position of the torso of the model at the end of the simulation, divided by 7, which is an acceptable distance to achieve in the environment. The division is needed in order to have the same order of magnitude for the coordinates.

Each simulation has been repeated 10 times, with different random initialization, which from now on will be called **trials**. The results in terms of average, standard deviation and best fitness over all the triads are reported in Table 1.

	Fitness			Diversity		
Algorithm	Mean	StdDev	Best	Mean	StdDev	Best
NC-eNEAT	1.352	0.617	2.577	0.163	0.114	0.382
CC-eNEAT	0.526	0.202	0.970	0.061	0.057	0.143
NC-NEAT	0.486	0.256	0.957	0.145	0.130	0.287

Table 1: The results obtained in terms of fitness and diversity over 10 trials.

4.3 Mujoco Walker2D Environment

As discussed above, all the simulations have been performed by using the Walker2D environment, based on the MuJoCo physics engine. It provides a two-dimensional bipedal model that consists of 17 input values and 6 output values. As a consequence, the ANNs evolved by eNEAT have an input layer with 17 neurons and an output layer with 6 neurons. The inputs encode the observation provided by the environment, i.e., the y position and the angle of torso, followed by the joint angles (8 values) and the x, y and angular velocity of torso, followed by joint angular velocities (9 values). The outputs encode the parameters of the two legs, i.e., thigh, leg and foot joints (6 values). Each joint can be activated in the range [-1,1], and each joint has a different sensitivity.

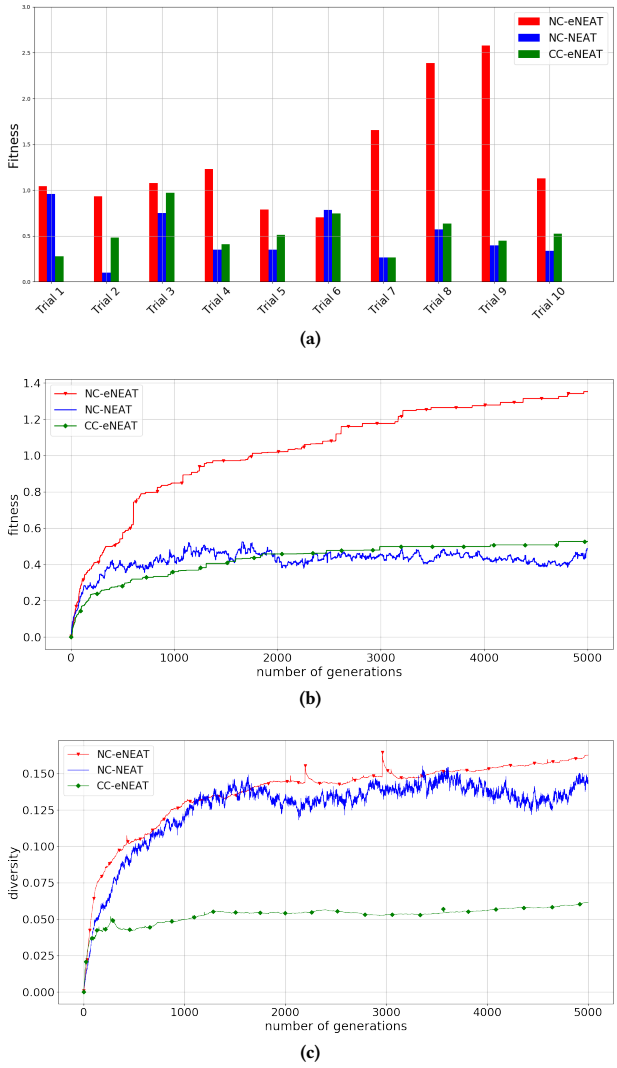

Figure 4: NC-eNEAT vs NC-NEAT vs CC-eNEAT: comparison of the best individuals obtained in terms of mean fitness over the trials (a), mean fitness trend (b) and mean diversity trend in the population (c).

4.4 Algorithm Comparison

Let us consider the comparison in terms of the fitness of the best individual between **NC-NEAT** and **NC-eNEAT** methods shown in Figure 4 for the 10 trials effected. It should be noted that **NC-eNEAT** obtains individuals with better fitness than **NC-NEAT**. Moreover, **NC-eNEAT** exhibits better performance on almost all trials, except trial 6 in which the performance are very similar. This "anomaly" has been also found in subsequent graphs, implying that the initial population in trial 6 may be particularly unlucky.

Figure 4 (b) shows the average of the fitness of the best individuals computed over the different trials. Analyzing the average on **NC-NEAT** it presents a swinging trend. In a single trial, this behavior is highlighted both on the best and especially

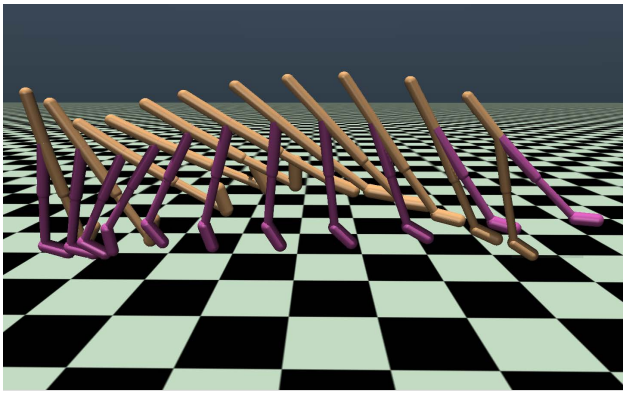

Figure 5: The best agent evolved by NC-eNEAT among all the trials [5].

on the average fitness. In fact, replacing all the population at each generation changes the performance a lot. **NC-eNEAT**, on the other hand, has an increasing trend for all generations.

Diversity is shown in Figure 4 (c), in terms of the average. The average diversity is very similar between the two algorithms, so **NC-eNEAT** is able to exhibit better performance and the same exploration capability of **NC-NEAT**.

The results show that **eNEAT** performs better on the task at hand. So, in order to find the best methodology, in the next section, the two configurations mentioned above are compared.

4.5 Configuration Comparison

Let us consider the comparison in terms of the fitness of the best individual between **CC-eNEAT** and **NC-eNEAT** shown in Figure 4 (a) for the 10 trials. It is evident that **NC-eNEAT** has better performance on almost all trials, except trial 6.

Figure 4 (b) shows the average of the fitness of the best individuals computed over all the trials. The trend of average fitness of the best individual in the population using the new-configuration always dominates the canonical-configuration. In addition, it should be highlighted that for canonical-configuration the evolution stagnates. In fact, the angular coefficient of the tangent to the curve described by the trend in blue in Figure 4 (b) is very low.

Diversity is represented in Figure 4 (c), which show the average of diversity in the population. It should be noted how the new-configuration improves the diversity in the population, which results in better exploration of the search space that, in addition to the already good exploitation of the solutions found, leads to better results. The peaks in both the graphs are a reflection of the extinction of a species from the population.

It should be noted that the new configuration obtains much better results. Hence, **NC-eNEAT** is the best method. The best agent evolved by **NC-eNEAT** is shown in Figure 5.

5 DISCUSSION

A few general considerations can be drawn from the process of finding the most suitable reward framework for the problem under exam. It is important to keep in mind that there are many ways to

achieve the desired goal. Traveling simply distance seems like a natural consequence of the task. But the ways in which distance can be covered are many and can differ from the natural human pace. A key aspect is the complexity of the solutions. The human brain is the result of million years of evolution, so the complexity of the neural connections is high. The ANN-based controller, on the other hand, builds on its own connections, so evolution will choose the fastest way, and therefore less complex, to achieve positive rewards. This is a problem found in all proposed reward frameworks.

Moreover, in the initial evolutionary phase the diversity of solutions in the population is strictly related to the direction of research. In fact, if different solutions have the same fitness value evolution can stagnate. The above considerations have suggested that learning sequence of tasks of increasing complexity could be the right solution. In this way, the fundamental features of human walking can be obtained, to incentivize them within a progressive reward system. Constraining the agent too much results in a slow evolution, so it is crucial to understand which are the most relevant features. From the experiments conducted, considering separate rewards on both legs implicitly encodes the alternation of these and the regularity of the pitch. This makes evident that evaluating an individual at the end of the simulation can be misleading. That is to say, if you want to imprint a behavior, this, if possible, must be considered from the beginning of the evolution.

As regards the hyper-parameters setting some consideration can be drawn. The NEAT configuration that describes the parameters for changing the topology and weights of the ANN influences significantly the correct evolution of the algorithm. Low probability of adding or removing neurons and connections makes the algorithm stable and allows to better explore the landscape. In fact, acting as a learning rate, too high probabilities bring the algorithm from an evolutionary regime to a chaotic one. The initialization chosen is extremely influential: keeping an ANN with no hidden nodes initially proved to be the best choice. Furthermore, not all connections are needed for the problem at hand. In the first generations, the algorithm tends to remove connections, so an initialization that randomly preserves only a certain percentage of connections can reduce the execution time. Finally, the *clamped* activation function has proven to get higher performance with respect to *sigmoid*, *tanh* and *relu*.

Finally, from the computational effort point of view it is possible to compare NEAT and eNEAT considering the minimum of the means of the best fitness value, i.e., 0.524. Such a value is reached by NC-NEAT at generation 1662 and by NC-eNEAT at generation 474. As a consequence, by using eNEAT leads to a reduction of the number of evaluations needed to reach such a value of 71.48%.

6 CONCLUSIONS

The aim of the paper is to propose an enhanced version of one of the latest Neuroevolution paradigms to solve the challenging problem of learning an ANN-based controller with a RL approach to control a walker model. Experiments have highlighted the interdependence among three key aspects: the reward framework, the EA chosen and the configuration parameters. The results show that these aspects are crucial in achieving good performance. Therefore, to obtain suitable results none of them can be ignored.

By simply encoding in the measure of performance a specific behavior that seems to be correct is not sufficient. Evolution can find individuals who can achieve good rewards exhibiting different behaviors than the desired one. This means that it is important to encourage correct behaviours and inhibit misbehaviours. It should be noted that for a Neuroevolution algorithm to have a continuous performance measure is a need. The final fitness function was tested and showed excellent results on this environment. What has struck most is that in a few generations an appreciable behavior of the agent emerges. For different reward frameworks, however, good results were achieved with a high number of simulations.

In the course of trials on the different techniques of Neuroevolution, the importance of topology for completing the task emerged. In fact, we might expect a more complex network to solve the problem more easily than a simple one, but this is not true.

From the comparison between the canonical version of NEAT and the one proposed here, it can be deduced that the canonical NEAT strategy is not elitist and it provides a good exploration, but a very bad exploitation. Instead, the proposed version of NEAT with a $(\mu + \lambda)$-strategy has a strong elitism strategy allowing to preserve the best individuals of each species. However, different network topologies are preserved. These two factors allow an excellent exploitation (thanks to elitism) and a good exploration (thanks to the preservation of diversity).

The comparison of the two algorithms shows that:

- in NEAT the average fitness of the bests among all the trials is much lower than the fitness of the best individual;
- in NEAT good solutions can be discarded prematurely due to stagnation;
- in eNEAT minimum performance are guaranteed also for different initialization of the algorithm;
- eNEAT shows proper evolutionary behavior, in which the average fitness increases over time;
- the best solutions at the end of all the eNEAT trials are much better than the canonical version of NEAT in majority of the cases analysed.

The inability to run simulations on GPU and the limited computational resources affected results and testing. Anyway, the results are remarkable especially considering that they have been obtained by exploiting limited computational resources and that the experiments terminated in evolutionary regime.

In the future, we aim to analyze more in detail how each of the proposed components, i.e., selection, reproduction, management of the population and reward function affect the performance, and eventually, to compare eNEAT with other improved versions of NEAT.

ACKNOWLEDGMENTS

The authors would like to thank dr. I. De Falco and dr. U. Scafuri (National Research Council of Italy) for the useful discussions on this research. This work was completed in part with resources provided by the University of Naples "Parthenope", Department of Science and Technologies, Research Computing Facilities (https://rcf.uniparthenope.it).

REFERENCES

[1] F. Amato, A. Lopez, M.E. Pena-Mendez, P. Vanhara, A. Hampl, and J. Havel. 2013. Artificial neural networks in medical diagnosis. *Journal of Applied Biomedicine* 11, 2 (2013), 47–58.

[2] L. Arnold, S. Rebecchi, S. Chevallier, and H. Paugam-Moisy. 2011. An Introduction to Deep Learning. *Proceedings of the European Symposium of Artificial Neural Network* Vol. 1, 477–488.

[3] T Blickle and L Thiele. 1996. A Comparison of Selection Schemes Used in Evolutionary Algorithms. *Evolutionary Computation* 4, 4 (1996), 361–394.

[4] C.B. Casey and Kris H. 2013. Artificial intelligence framework for simulating clinical decision-making: A Markov decision process approach. *Artificial Intelligence in Medicine* 57, 1 (2013), 9 – 19.

[5] I. Colucci, G. Pellegrino, A. Della Cioppa, and A. Marcelli. [n.d.]. *Independent Leg reward on MuJoCo's Walker2D environment.* https://www.youtube.com/watch?v=sFZeF2rfDd4

[6] S. Debnath, A. Devos, E. Heiden, R. Julian, and F. Khatana. [n.d.]. *Humanoid Imitation Learning from Diverse Sources.* https://www.endtoend.ai/envs/gym/mujoco/walker2d/

[7] N. Di Palo. [n.d.]. *Learning to walk with evolutionary algorithms applied to a bio-mechanical model.* https://towardsdatascience.com/learning-to-walk-with-evolutionary-algorithms-applied-to-a-bio-mechanical-model-1ccc094537ce

[8] S.E. Dilsizian and E.L. Siegel. 2013. Artificial Intelligence in Medicine and Cardiac Imaging: Harnessing Big Data and Advanced Computing to Provide Personalized Medical Diagnosis and Treatment. *Current Cardiology Reports* 16, 1 (13 Dec 2013), 441.

[9] Nicolas Heess et al. 2017. Emergence of Locomotion Behaviours in Rich Environments. *CoRR* (2017). arXiv:1707.02286

[10] Holly Grimm. 2019. *Imitiation Learning and Mujoco.* https://hollygrimm.com/rl_bc

[11] J. Ho and S. Ermon. 2016. Generative Adversarial Imitation Learning. arXiv:1606.03476

[12] Sepp Hochreiter. 1991. Untersuchungen zu dynamischen neuronalen Netzen. (1991).

[13] Watkins C. J. and Dayan P. 1992. Technical Note: Q-learning. *Machine Learning* 8 (1992), 279––292.

[14] A. Karniel. 2011. Open questions in computational motor control. *Journal of integrative neuroscience* 10, 3 (2011), 385–411.

[15] L. et al. Kidzinski. 2018. Learning to Run challenge solutions: Adapting reinforcement learning methods for neuromusculoskeletal environments. *CoRR* (2018). arXiv:1804.00361

[16] A. Krizhevsky, I. Sutskever, and G.E Hinton. 2017. ImageNet Classification with Deep Convolutional Neural Networks. *Commun. ACM* 6, 6 (2017), 84–90.

[17] Stanford Neuromuscular Biomechanics Laboratory. [n.d.]. . http://osim-rl.stanford.edu/docs/models/#arm2denv

[18] Stanford Neuromuscular Biomechanics Laboratory. [n.d.]. *NeurIPS 2019: Learn to Move - Walk Around.* https://github.com/stanfordnmbl/osim-rl

[19] Yuxi Li. 2017. Deep Reinforcement Learning: An Overview. arXiv:cs.LG/1701.07274

[20] T.P. Lillicrap, J.J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. 2015. Continuous control with deep reinforcement learning. arXiv:cs.LG/1509.02971

[21] S.A. McLeod. 2018. Skinner - Operant Conditioning. *Simply Psychology* (2018).

[22] L.C. Melo, M.R.O.A. Maximo, and A.M. Cunha. 2019. Learning Humanoid Robot Motions Through Deep Neural Networks. arXiv:1901.00270

[23] V. Mnih, K. Kavukcuoglu, D. Silver, Graves. A., I. Antonoglou, D. Wierstra, and M. Riedmiller. 2013. Playing Atari with Deep Reinforcement Learning. arXiv:1312.5602

[24] OpenAI. [n.d.]. *Deep Deterministic Policy Gradient.* https://spinningup.openai.com/en/latest/algorithms/ddpg.html

[25] OpenAI. [n.d.]. *A python library that is a collection of test problems.* https://gym.openai.com/envs/#classic_control

[26] OpenAI. [n.d.]. *A set of continuous control tasks, running in a fast physics simulator.* https://gym.openai.com/envs/Walker2d-v2/

[27] A. N. Ramesh, C. Kambhampati, J. R. T. Monson, and P. J. Drew. 2004. Artificial intelligence in medicine. *Annals of the Royal College of Surgeons of England* 86, 5 (Sep 2004), 334–338.

[28] T. Salimans, J. Ho, X. Chen, Sidor. S., and I. Sutskever. 2017. Evolution Strategies as a Scalable Alternative to Reinforcement Learning. arXiv:1703.03864

[29] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever. 2017. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864* (2017).

[30] M. Sanjeevi. [n.d.]. *Ch 12:Reinforcement learning Complete Guide towardsAGI.* https://medium.com/deep-math-machine-learning-ai/ch-12-reinforcement-learning-complete-guide-towardsagi-ceea325c5d53

[31] J. Schulman, Wolski. F., P. Dhariwal, A. Radford, and O. Klimov. 2017. Proximal Policy Optimization Algorithms. arXiv:1707.06347

[32] J. Schulman, S. Levine, P. Moritz, M.J. Jordan, and P. Abbeel. 2015. Trust Region Policy Optimization. arXiv:1502.05477

[33] F. Sehnke, C. Osendorfer, T. Rückstieß, A. Graves, Peters J., and Schmidhuber J. 2010. Parameter-exploring policy gradients. *Neural Networks* 23, 4 (2010), 551–559.

[34] F. Seide, G. Li, and D. Yu. 2011. Conversational Speech Transcription Using Context-Dependent Deep Neural Networks. In *Interspeech 2011* (interspeech 2011 ed.). International Speech Communication Association.

[35] R.L. Seungjae. [n.d.]. *MuJoCo Walker2D Environment.* https://www.endtoend.ai/envs/gym/mujoco/walker2d/

[36] D. Silver, G. Lever, N.M.O. Heess, T. Degris, Wierstra. D., and M.A. Riedmiller. 2014. Deterministic Policy Gradient Algorithms. In *ICML*.

[37] S. Song and H. Geyer. 2015. A neural circuitry that emphasizes spinal feedback generates diverse behaviours of human locomotion. *The Journal of physiology* 593, 16 (2015), 3493–3511.

[38] Kenneth O. Stanley. [n.d.]. *NEAT-Python.* https://neat-python.readthedocs.io/en/latest/