MSc Project Report

Title
An Evolutionary Strategy for the optimisation of Deep Convolutional Neural Networks in Real-time Complex Visual-only scenarios

Author
Carlos Fernandez Musoles

Programme
MSc Intelligent Systems

Year
2016

Section .....
of.....

FOT
FACULTY OF TECHNOLOGY

An Evolutionary Strategy for the optimisation of Deep Convolutional Neural Networks in Real-time Complex Visual-only scenarios

Dissertation for MSc Intelligent Systems

Author: Carlos Fernandez Musoles

DeMontfort University, 2016

Word count: 13791

Abstract

Since the early days of Computer Science games have been used by researchers as a test-bed for Artificial Intelligence. From the *rote learning* of Samuel's checkers to IBM's Watson *cognitive computing* winning a live Jeopardy TV show, machine learning continues to reach milestones previously considered unachievable.

In the recent past computer scientist and developers have started to leverage the power of the Graphics Processing Units (GPU) for non-graphics related tasks. Researchers have taken advantage of this new source of computational power allowing for more complex AI models. Deep Convolutional Neural Networks (CNN) are one such models that has open the doors to highly-complex visual data to be used in Machine Learning.

This work presents a successful Evolutionary Reinforcement Learning Strategy for the optimisation of Deep Neural Networks to develop a controller to solve complex simulated visual tasks. The approach demonstrates the use of Convolutional Neural Networks as **feature extractors** to reduce the complexity of dealing with raw-pixel data from RGB images. Moreover, the weights of the CNN are tuned using an unsupervised evolutionary process that focuses on diversity of output.

The features extracted by the CNN are used as input to a **secondary neural network** that is trained using Neuroevolutionary algorithm to determine which action the agent should take. The evolutionary algorithm performs a search on the **policy space** -i.e. mapping states to single actions- which yields better results than a search on the value function space -mapping a state with the values for all possible actions.

The strategy completes two challenging Doom scenarios and outperforms proposed deep reinforcement learning alternatives such as Deep-Q-Network.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank my family for their constant support, both financial and moral, specially to my wife, for patiently being there in the busy times, and my parents, for always showing their faith on me.

Thanks to all the lecturers I had during my time at DeMontfort University, in particular to David Elizondo for his encouragement, and Ben Passow for its guidance.

List of acronyms

Acronyms used throughout the document In alphabetical order:

- AI → Artificial Intelligence
- CNN → Convoluted Neural Network
- DQN → Deep Q-Learning Network
- EA → Evolutionary Algorithm
- FS-NEAT → Feature Selection NEAT
- GA → Genetic Algorithm
- GPGPU → General Purpose Computing on Graphics Processing Unit
- NE → Neuroevolution
- NEAT → Neuroevolution through Augmented Topologies
- RGB → Red-Green-Blue
- RL → Reinforcement Learning
- TD → Temporal Difference
- TWEANS → Topology and Weights Evolving Artificial Neural Networks

1. Introduction

1.1. Overview

Since the early days of Computer Science games have been used by researchers as a test-bed for Artificial Intelligence. From the *rote learning* of Samuel's checkers [1] to IBM's Deep Blue clever use of *searching algorithms* in order to beat the world's chess champion[1], and Watson *cognitive computing*[2] winning a live Jeopardy TV show, machine learning continues to reach milestones previously considered unachievable.

To aid the advancement of machine learning, researchers have traditionally made use of frameworks and fostered competitions around them. In the case of complex real-time games, perhaps two of the most prominent test-beds are TORCS[3] -a driving simulation game- and Starcraft[4] -a Real Time Strategy game. Diverse strategies capable of learning to play in those scenarios have been presented: TORCS drivers using neural networks [2][3], fuzzy logic [4] and genetic algorithms [5] to navigate around a race track; StarCraft [6] bots to handle resource management, attack and defend tactics and strategies. Although their success is undeniable, they tend to rely on domain specific information and internal preprocessed engine data -such as exact distances to beacons, position of opponents, etc.- forced by constraints on computational resources and the lack of algorithms to extract meaningful information from raw sensory data.

As humans, we interact with the world directly through our senses, mainly sight, and have to derive information from it by ourselves. However, processing real-time raw sensory data directly in machine learning algorithms brings difficulties such as high dimensionality and pattern identification -this is particularly complex in the case of images with different illumination, different colours, objects not in a specific position or orientation, etc.

In the recent past computer scientist and developers[5] have started to leverage the power of the Graphics Processing Units (GPU) for non-graphics related tasks -commonly known

1Information about the match: https://www.research.ibm.com/deepblue/, accessed on 01/09/2016

2http://www.ibm.com/watson/, accessed on 01/09/2016

3Official website: http://torcs.sourceforge.net/, accessed on 01/09/2016

4Several competitions held regularly: https://sites.google.com/site/starcraftaic/, https://webdocs.cs.ualberta.ca/~cdavid/starcraftaicomp/, both accessed on 01/09/2016

5The two main GPU manufacturers have established robust libraries for parallel computing: NVIDIA's CUDA http://www.nvidia.co.uk/object/cuda-parallel-computing-uk.html and AMD's GPUOpen http://gpuopen.com/

as General-Purpose computation on the GPU (GPGPU). GPUs accelerate computation in two ways: 1) it can work independently from the CPU on different tasks; and 2) due to its original purpose -do millions of vector maths calculations per frame- it is designed to be able to compute massively in parallel. AI researchers have taken advantage of this source of computational power in what has been known as Deep Learning, allowing for more complex models to be used in Machine Learning. Deep Convolutional Neural Networks (CNN) are one such models that has been used in complex image recognition tasks with very notable examples within the game sphere -AlphaGo [7] and Atari general players [8].

Aside from the nature of the input information, another aspect that is common to humans and other animals way of learning is that they tend to use reward / punishment feedback from their environment to improve. Researchers have used that intuition in a learning model called Reinforcement Learning (RL). The agent knows its current state and the set of actions available, and it receives a reward (positive or negative) in certain states. The challenge is to learn an optimal policy -sequence of actions- that would yield the maximum total reward.

1.2. Motivation

Traditional RL used Q-tables -tables that represent the reward value per state- which quickly becomes infeasible with very large state-action spaces such as the ones using visual input or continuous actions. Authors [7], [8] have introduced the use of Deep CNN as function approximations of Q-values with very promising results.

CNN have not been widely explored in complex continuous decision spaces (near-infinite state-actions). Furthermore, training and optimising CNN is complex and requires massive amounts of data, either collected by a trainer and labelled -supervised learning- or using RL -which generates its own labels from environment rewards. However, both methods to learning the weights of the CNN rely on a pre-designed network topology. Evolutionary Algorithms (EA) such as [9], [10] have offered a working alternative to network design by allowing the algorithm to discover the best topology as part of the learning process. The idea this work attempts to explore is whether evolutionary strategies could aid in the optimisation of CNN for complex real-time decision making scenarios?

1.3. Objectives

This work explores an EA approach to train and tune a CNN to be used in a complex real-time scenario (3D Doom) using visual-only input and rewards from the environment, and

compares it to both supervised CNN training and Deep-Q-Learning. The goals can be summarised as follows:

1. To *explore* the suitability of EAs for the optimisation of deep neural networks in a complex, real-time, large decision space scenario.
2. To use the findings to *implement* visual controllers to complete task-specific scenarios in Doom.

1.4. Scope

The focus of this work is *to explore the suitability of Evolutionary algorithms for the optimisation of convolutional networks* in a complex, real-time scenario based on visual input. Thus, the scope is limited to the design, implementation and evaluation of an evolutionary approach to CNN design and training. The ViZDoom [11] framework[6] is chosen, as it provides the challenge of a complex visual scenario (3D mazes with enemies, items, traps, etc.) as well as being specifically tuned to facilitate RL strategies -with in-built rewards system.

Due to the restrictions imposed by the framework, the agent only receives information from the world in two forms: 1) visual data from the rendered screen; and 2) state information such as ammo, weapon selected, health status, etc. The agent is required to perform actions such as navigating through a 3D maze, attack other agents, and pick up items to stay alive.

The use of the ViZDoom framework is limited to serve as a test tool to evaluate the EA strategy, i.e. as a means to an end and not the goal itself. Two challenging scenarios are designed -described in Section 3.

1.5. Document structure

The remainder of the document is organised as follows: Section 2 offers a critical overview of the literature concerning techniques and algorithms within the scope of this work; Section 3 defines the proposed evolutionary approach to optimise deep neural networks, as well as describe the frameworks and tools that shape the scope of the task; Section 4 sets the experimental design boundaries to evaluate the proposed solution; Section 5 includes the quantitative results from the experimentation; Section 6 critically analyses and discusses the data obtained through experimentation; and Section 7 lists the conclusions derived from this work as well as outlining potential areas for future research.

6ViZDoom is a reinforcement learning framework built on top of the Doom engine to facilitate AI research in complex visual-only scenarios. http://vizdoom.cs.put.edu.pl/, accessed on 01/09/2016

2. Literature review

2.1. Evolutionary algorithms

2.1.1. Principles, encoding and types

Evolutionary algorithms (EA) are a family of optimisation methods that use the core principles of natural evolution to train and tune models. Although there are many types of EA -see Table 1- all share the same fundamental logic depicted in Figure 1:

1. An *initial population* is formed from a randomly generated collection of potential solutions.
2. At each generation, the *fitness of each solution is evaluated* by allowing it to solve the problem task and measuring its performance. Candidates that solve the task better are given a higher fitness.
3. Based on the fitness they are *given a probability* to go to the next generation
4. At some point, *variation operators act on individuals* to produce random changes -which is what allows a population to improve.
5. The process is repeated until some condition is satisfied -maximum fitness score reached, set number of generations, etc.

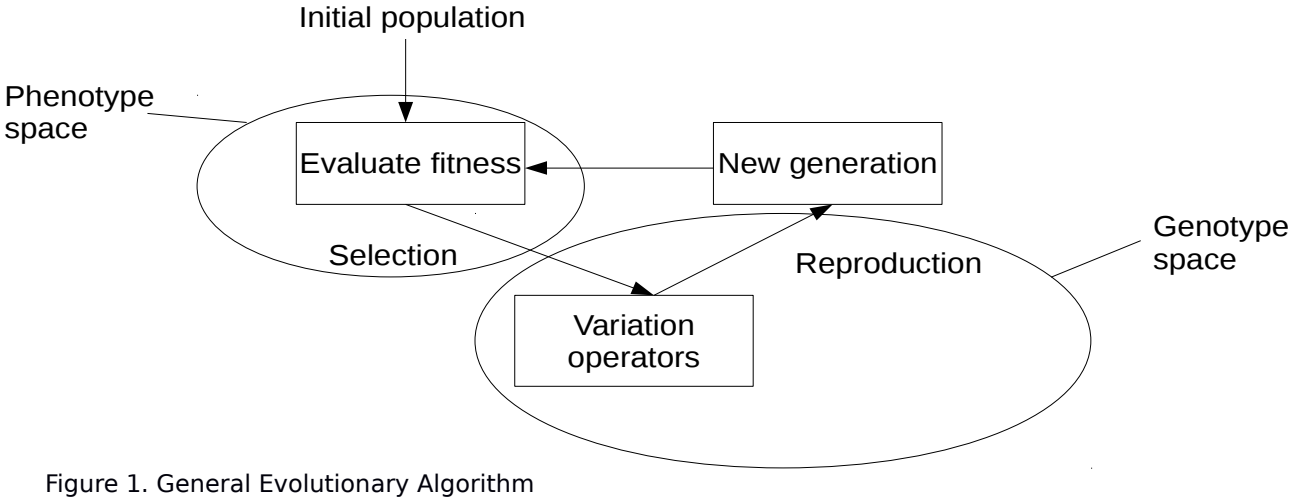

Figure 1. General Evolutionary Algorithm

EA are a class of unsupervised learning methods, as they do not require a training sample or expert knowledge to operate. They formally perform a systematic *search* on the entire solution space. EA can be used in any task whether a known solution is known or not, provided there is a known way of evaluating how well a potential candidate performs. Thus, EA are ideally placed for scenarios where there is little-to-none training

data available or to search for global optimal solutions where other suboptimal alternatives are known.

One of the key aspects of any EA is the dichotomy genotype-phenotype, or the difference between the candidate solution and its encoding. Like in natural evolution, the fitness of individuals is measured on *how it performs* in its environment. Evidently, the information on *how to recreate* that solution needs to be encoded somehow so it can be passed along to future generations. The candidate's behaviour or tangible features are called its phenotype, whereas the encoding is its genotype. As shown in Figure 1, although very closely related, different evolutionary mechanisms work over the phenotype space (evaluation of fitness) and the genotype space (variation operators).

There are two possible ways of encoding a candidate solution: direct and indirect encoding. In direct encoding, the genome contains a full description on the features of the individual on a 1 to 1 relation. On the other hand, indirect encoding contains only the rules on how to construct the features. In the example shown in Figure 2, the genome encodes the solution -in this case a number which in turn can represent a value on a grey scale spectrum- as a 7 bits binary number; genome '0000101' produces the phenotype 5, or light grey. The example of indirect encoding shown in Figure 2 is taken from the Hyper-NEAT algorithm [10], where the weights and connections of a neural network are encoded in a secondary, simpler network called a Compositional pattern-producing network (CPPN).

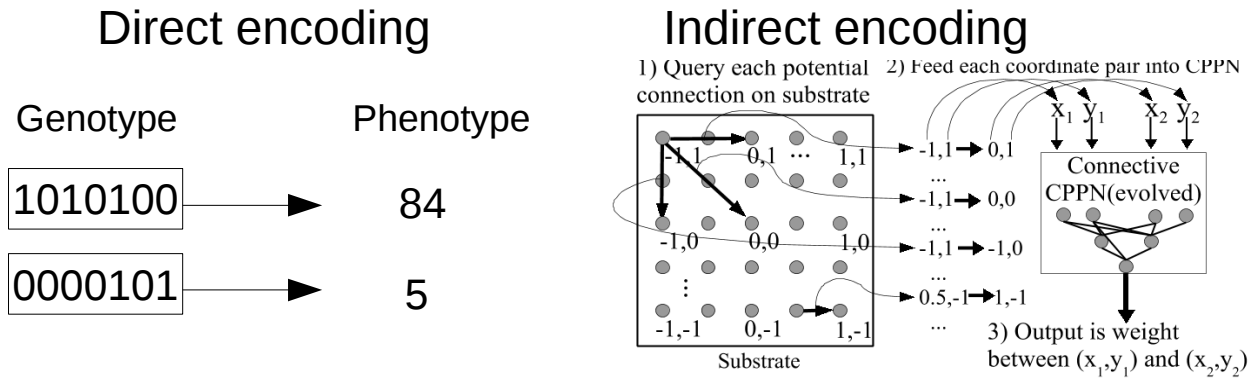

Figure 2. Direct (left) vs indirect (right) encoding of a genotype for EA. Source image [10].

Although direct encoding approaches are easier to design and work with, they tend to scale poorly: their complexity increases proportional to the size and complexity of the solution's parameters. Indirect encoding has no scaling issues but they are notoriously more difficult to implement. The choice of encoding is very much a problem-specific decision that is influenced by many constrains such as memory, computational power, time scale, etc.

Evolutionary algorithm	**Variability introduction**	**Parent selection and reproduction**
Evolutionary programming	Mutation only	Fitest candidates go to next generation; one parent
Genetic algorithm	Crossover and mutation	Two parents per child solution; parent selection based on fitness
Evolutionary strategy	Mutation and crossover; evolutionary parameters are evolved as well	Parents selected at random
Swarm intelligence	Mutation only	One parent, using moving operator
Differential evolution	Mutation prior to crossover	One-to-one spawning, survivor selection (keep if new solution is better)

Table 1. Classification of some types of EA depending on the way they introduce variability into the population and how they select and reproduce their individuals

2.1.2. Evolution and Neural Networks

Evolutionary algorithms have been used in conjunction with other machine learning methods. One popular combination is to use EA to train Artificial Neural Networks (ANN). Traditionally ANN are trained in a supervised manner from a data set of correctly classified examples through gradient descent -backpropagation algorithm [12]. However, this approach *assumes* the network topology is fixed prior to training, making the design of ANN a less an exact science and more a trial and error process, with the risk of finding local optima solutions. The use of EA to optimise ANN has been called Neuroevolution (NE), and it can automate the learning process in two ways: tuning the parameters of the ANN (weights and bias) and generating an appropriate topology (neurons and links).

An EA that evolves both weights and topology has been called TWEANN (Topology and Weight Evolving Artificial Neural Network) [9]. According to [13], a good TWEANN algorithm must deal with the following challenges:

1. Competing conventions: when recombining two parent solutions with different network architectures, the result is worse than any of the original solutions. This happens mostly when the behaviour is influenced by symmetry and other structural elements in the network, which are broken when combined.
2. Protect innovation for long enough to allow weights of a new structure to evolve and be optimised -not to discard a structural solution before it has been sufficiently explored, i.e. it has had a chance to tune its weights.
3. Evolve minimal solutions: starting with random network topologies may result in overly-complex structures that are not global optima solutions.

Two examples of TWEANN are the co-evolutionary approach CoSyNE [14] and the well-known NEAT algorithm [9] (NeuroEvolution of Augmented Topologies). NEAT uses direct graph encoding of neurons and links -see Figure 3- and mutation and crossover operators to evolve both weights and the topology of the network.

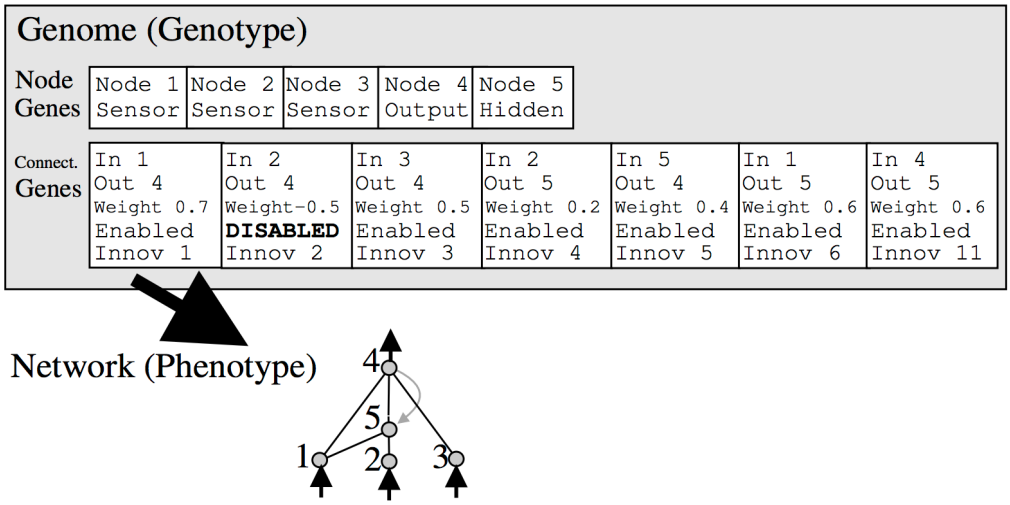

Figure 3. NEAT direct graph encoding of genotype, source [9]

The main contributions of NEAT is how it tackles the TWEANN challenges:

- An innovation number is added to identify genes -portions of the network. Crossover is allowed to happen at the gene level, avoiding the competing conventions issue.
- Innovation is protected at two levels: 1) Individuals similarity is measured taking into account number of neurons, links and the weights of the network; individuals close to each other are considered members of a species, and recombination is only allowed within that group -often interspecies recombination is permitted on very low probabilities. 2) novel solutions are given a fitness score boost.
- Topology candidates start very minimally, with no hidden units and only links between input and output.

NEAT was originally able to solve simple XOR task as well as a more complex pole balancing task [9] but has since been used in a variety of domains such as completing a level of Super Mario[7], a machine learning-based game NERO [15] and successfully playing a push-inertia car game [16].

After NEAT's initial publication, key improvements have been proposed. Feature Selection NEAT (FS-NEAT) [17] includes input selection as part of the evolutionary process by starting with very minimal networks that only include links from a subset of inputs to the outputs. By leaving inputs out and allowing the algorithm to add more as part of the

7MarI/O is a Lua implementation of NEAT on an emulated version of Super Mario https://www.youtube.com/watch?v=qv6UVOQ0F44, accessed on 01/09/2016

mutation operators, FS-NEAT has been shown to be able to reduce the amount of input features used with no impact on performance on scenarios where certain inputs are redundant.

Another proposed improvement increases the size of networks that can be produced. Instead of directly evolving the ANN, Hyper-NEAT [10] uses the NEAT algorithm to evolve a Compositional pattern-producing network (CPPN), which in turn determines the connectivity of the final network. Thus, Hyper-NEAT uses indirect encoding (the CPPN) to produce a more complex network (its phenotype). It uses a fixed substrate network -with input, hidden and output units predetermined- and to express the connections between units it employs the CPPN; the weight of the connection is the output of the CPPN given the substrate coordinates of both units. By using the CPPN as indirect encoding, Hyper-NEAT can produce networks of different resolutions -by changing the size of the fixed substrate.

Although Hyper-NEAT requires a fixed substrate to work, this limitation has been overcome in a further improvement, Enhance Substrate Hyper-NEAT [18]. ES-Hyper-NEAT does not presuppose the location or number of hidden units, making it easier to find global optima solutions. Both Hyper-NEAT and ES-Hyper-NEAT work best specifically when there are structural relationships in the input to be exploited such as symmetry, repetition, etc.

2.2. Simplification of highly-dimensional problems

Problems such as those using raw visual data face the *curse of dimensionality*[8], and many machine learning methods do not scale well to deal with large input spaces or highly-dimensional problems. The usual approach to tackle them has been to apply reduction techniques such as *dimension reduction* and *feature extraction or detection*. These methods attempt to simplify the input size and complexity by either detecting and removing redundancies or applying mathematical transformations to obtain a subset of features.

2.2.1. Dimension reduction

Dimension reduction methods attempt to reduce the number of variables or dimensions on a data set to make it easier -or even possible- to analyse and process. The assumption they make is that not all the variables are equally important, or that they are somewhat correlated. [19] distinguishes two categories of dimension reduction techniques, depending on how the original dimensions are combined: linear and non-linear. The most popular methods are: linear and non-linear principal component analysis which combines

8The expression “curse of dimensionality” was coined by Richard E. Bellman and refers to the problems that arise when processing data in high-dimensional spaces.

correlated variables; factor analysis where a subset of features is assumed to influence others to allow simplification; multidimensional scaling to reduce dimensions keeping the relative distance between observations; and even neural networks and genetic algorithms -though these fall into the feature extraction and detection techniques.

2.2.2. Feature extraction and feature detection

Although conceptually not different from dimension reduction techniques, feature extraction and detection methods are those that go beyond direct mathematical transformation of the data. Feature selection methods are those which perform a selection of the input set available; whereas feature extraction methods transform the available features into new ones.

Feature extraction and selection methods usually involve a machine learning algorithm to do the transformation -common choices are genetic algorithms and neural networks. [20] presents an alternative to linear transformation for feature extraction using genetic algorithms. A feature vector that transforms the original input set is evolved, its fitness calculated based on classification accuracy over a labelled train set -supervised process.

One example of unsupervised feature extractor for the visual domain is found in [21] where a deep neural network is evolved to detect a set of features based on a diversity index that forces the feature detector to output different features for different images. The algorithm will be fully discussed in Section 3.

2.3. Game AI approaches that use visual input

Strategies that use visual input have been applied in various scenarios: driving agents [21]; 3D agent control [22]; and even attempts to General Game Play [8], [23].

In the driving domain, one of the first implementations to use visual information as the main input for steering is ALVINN [24], a system capable of steering using a recurrent neural network that accepts a camera image and laser ranger data as input. In the TORCS environment, [21] showed how a deep neural network layout could be constructed using evolutionary algorithms to navigate –steer, accelerate and brake- a simple track using visual data, completely unsupervised.

NeuroQuake [22] is an agent policy for the Quake II game that demonstrates the use of a visual-only ANN to successfully control a player in a complex 3D environment. The key findings of NeuroQuake are: 1) when downscaling the input image, a human-like retina approach (with more resolution, i.e. pixel density, in the middle and less in the edges) yields better results than uniform distribution; 2) a secondary non-visual ANN can be used to train the visual-only ANN to avoid the need for an expert training.

More recently authors have started to take advantage of the extra computational power that comes with GPGPU and created more complex deep neural networks to attempt to solve increasingly difficult problems. Arguably one of the most challenging problems in game AI is the General Video-game Playing (GVGP) scenario where an agent must learn to play a *collection* of games. [23] compares the performance impact of different state representations on neuroevolutionary algorithms in Atari games. The representations include various approaches to interpret visual input -from raw pixel data to preset classes of objects- showing that visual control is possible, although directly use of pixel data significantly underperformed the alternatives. The authors speculate that this could be the case because Atari games can be better learnt with compact -i.e. preprocessed- information and the extra complexity of pixel information does not benefit the algorithm. However, the simplicity of the networks (feedforward ANN) could act as a bigger limitation. This is evident from the work of [8] and [25], who use a deep CNN trained with Deep Q-Learning (DQN), a modification of the Q-Learning algorithm, to successfully play a variety of Atari games, some of them significantly better than a human expert. The CNN uses visual-only information which is processed by a series of convolutional layers to assign values to the actions available to the agent. This approach will be discussed in more detail on Section 2.5.

2.4. Convolutional Neural Networks

Machine Learning has enjoyed a resurgence in the last few years in no small part due to the popularisation of Convolutional Neural Networks. Deep CNN training and computing has become practical thanks to the massively parallel computation brought by GPGPU. CNN are specially ideal for image processing and have shown incredible results in image classification [26][27], handwriting recognition [28], [29] and pattern recognition [30], [31]. Perhaps where CNN have attracted more recent attention has been in the field of game AI as function value approximators, achieving and surpassing human expert levels in Atari games [25] and beating the Go world's champion[9] with AlphaGo [7], a long sought after milestone in AI.

CNN are multi-layered neural networks inspired by the receptive fields of animal's visual cortex. In essence they are like regular ANN in that each layer transforms a volume of activations through a differentiable function. However, they assume the input is an image, normally 3 dimensional -width, height and depth, or number of channels. There are three common types of layers in a CNN: convolutional, pooling (up or down-sampling) and fully-connected (like regular networks, each unit has links to all units in the next layer). Figure 4 shows an example of CNN with two pairs of Convolution-Downsampling

9AlphaGo won 5-0 against the European Champions Fan Hui on October 2015, and 4-1 against the World Champion Lee Sedol on Match 2016 https://deepmind.com/alpha-go accessed on 01/09/2016

layers connected to two fully connected layers -popularised by the LeNet-5 handwriting recognition system [28].

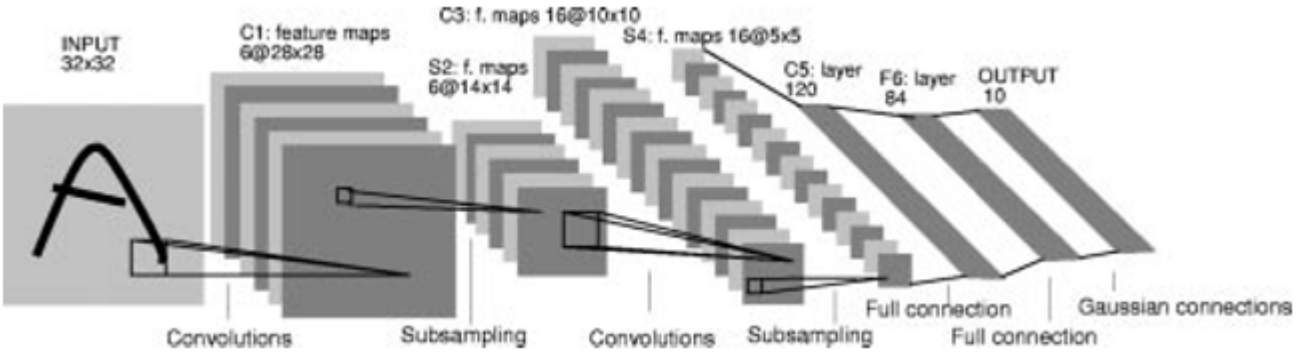

Figure 4. Example of a Convolutional Neural Network with several layers taken from LeNet-5 [28]

For the most part CNN do not operate on a fully-connected basis like normal ANN. Instead each convoluted layer contains a set of filters or *kernels*. During a forward pass, each kernel is convolved -or slided- along the activation volume to produce a 2-dimensional activation map, which is then fed to the next layer. Typically, each convoluted layer receives a multidimensional activation volume and outputs a collection of 2-dimensional maps.

There are two properties of CNN that make them ideal for image processing: *local connectivity* and *parameter sharing*. Local connectivity refers to restricting the signals that are received per neuron to a region of the input -determined by the size of the kernel or its *receptive field*. Each kernel acts as a feature detector over the input; since a convoluted layer contains multiple kernels and normally there are multiple layers, CNN result in massive image pre-processing filters. Parameter sharing refers to the sharing of weights and bias along the kernels of the same depth, significantly reducing the number of parameters to be learnt -also making CNN scalable when compared to ANN. This allows convolutional networks to detect features irrespective of their position or orientation in the image, as the same kernels are applied to the entire image.

2.5. Reinforcement learning

2.5.1. Concept and formulation

Reinforcement Learning (RL) is a scheme where an agent learns from the feedback it gets from its environment -positive, or reward; and negative, or punishment. Through trial and error, the agent identifies sequences of actions that lead to maximum reward. This is a natural way human and other animals learn in many circumstances. RL is the ideal machine learning method for unsupervised, sequential scenarios where the environment can provide information about the goodness of an action, given the state the agent is in.

Formally, at any time t the agent is said to reside in state s_t in which it performs action a_t. The environment assigns a numerical reward r_t and changes the state to s_{t+1} following the transition mapping $s_{t+1}=\delta(s_t,a_t)$. The agent continues to select actions until s_t is a terminal state -i.e. the task has been finished. In RL, the agent must learn a policy that maps states and actions $\pi: S \rightarrow A$; the optimal policy π^x is defined as the policy that gives the maximum reward to the agent $\pi^x = argmax\, V(s)$ where $V(s)$ is the cumulative reward associated with reaching state s. To calculate the reward of a current state-action pair, one calculates the sum of all rewards obtained to reach said state; usually a discount rate is used to reduce long distance rewards overtime:

$V^{\pi}(S_t)=\sum_{i=0}^{\infty} \gamma\, r_{t+i}$ where γ is the discount factor.

2.5.2. Policy vs value function search

There are two main approaches to learning the optimal policy: *policy-search* and *value function-search*. The former maintains an explicit representation of the policy which is modified through search operators; whereas the latter approximates $V(s_t)$ to determine the best action given state s_t. In other words, policy-search explores the policy space to find the mapping $\pi: S \rightarrow A$ that yields the highest final reward. Value function-search on the other hand aims to approximate the function $V^{\pi}(S_t)=\sum_{i=0}^{\infty} \gamma\, r_{t+i}$ so the agent can select the action yielding the highest immediate reward.

Whilst evolutionary algorithms are well suited for policy-space search [32], Temporal Difference (TD) is the common choice for value function-search. In TD, observed reward values from consecutive states are used to update the predictions made by the agent over said reward using the following update rule:

$V(s_t)=V(s_t)+\alpha(V(s_{t+1})-V(s_t)+r_t)$ where α is the learning rate

The agent selects its next action based on its current predictions, choosing always the action associated with the highest reward for the current state.

2.5.3. Value function-search: Deep-Q-Network

The work by [7], [8] demonstrates the potential of CNN in large, complex space-action spaces. Deep-Q-Network (DQN) uses a CNN to approximate the state-action reward values in visual-only domains. It proposes a variation of Q-Learning [33], which uses current reward predictions and actual feedback from the environment to generate training data, which is in turn used to alter the weights of the CNN with Stochastic

Gradient Descent [34] -a batched form of backpropagation. A common problem faced by RL agents is the issue of training from *correlated data*: as the agent progresses through the environment, often the data it generates to train its value function -the rewards- are somewhat correlated to one another; that is the case as the states the rewards are taken from are themselves correlated through the sequence of actions. The authors used *experience replay* to reduce this problem and to smooth the training distribution. A fixed-size memory of current state-reward-next state triplets is maintained by adding new elements through experience; each batch training is done with a random selection of the memory set.

In the Atari General Player, the 5 layers CNN (3 convoluted and 2 fully connected) accepts as input an 84x84 grey scale image and outputs the estimated values for each action in the current state (buttons to be pressed). Results show good performance on 49 games, often better than humans and consistently better than any tested alternative such as HyperNEAT and Sarsa.

2.5.4. Evolutionary algorithms in RL

Evolutionary algorithms have been proven to be an effective tool for RL tasks [13] [35]. They are suited particularly for policy-search, for the following reasons:

- Cope well with partial observability of the state.
- Can be easier to represent as only the best state-action is required, no need to evaluate all state-action pairs.
- Simple way to tackle large continuous action spaces

Often ANN are used either as value functions -predicting the reward values associated with each action available- or, more commonly. as direct policies -the current state decides what action to take. Hence Neuroevolutionary algorithms such as NEAT or Hyper-NEAT are suitable methods. Although individual evolution -one chromosome per solution- is more frequently used, distributed cooperative approaches have been proposed [14] where the decision policy is discretised into sub-goals, each of them represented by a chromosome.

Some reported limitations of EA in Reinforcement Learning include:

- online learning is difficult or impossible
- Rare states -infrequently visited- are often ignored, i.e. if something does not have sufficient impact (low evolutionary pressure) it will be removed or not included.

2.5.5. A hybrid approach: NEAT+Q

NEAT+Q [16] is an algorithm that combines Temporal Difference with Neuroevolutionary algorithms to design an agent capable of solving RL tasks. The approach uses ANN as a value function approximator that is trained using Q-Learning, but attempts to enhance TD by not requiring a fixed ANN topology to be decided beforehand. Instead, a parallel neuroevolutionary process is carried out using NEAT. The idea is to have NEAT drive the evolution of the network's topology; at each generation, individuals are assessed in their ability to perform on their environment *after* a TD session to optimise its weights. Naturally, topologies that are conducive of learning better through TD are scored high in the fitness evaluation, thus placing the evolutionary pressure on the topology.

Although NEAT+Q shows good results in relatively simple scenarios, the authors acknowledge the learning is orders of magnitude slower than alternatives, due to high sampling costs, as each NEAT solution needs to be TD trained. For that reason it cannot be practically used in a large continuous state-action space.

2.6. Evolutionary algorithms for deep neural networks

The standard neuroevolutionary algorithms reviewed so far -NEAT, Hyper-NEAT, Genetic programming, etc.- normally target small to medium sized topologies, in number of units or complexity of layers. Even though Hyper-NEAT has the potential to generate networks with millions of units, the substrate network is rarely made to evolve more than a few layers in depth.

[36] presents a neuroevolution algorithm that uses Hyper-NEAT in combination with traditional backpropagation to produce deep networks able to recognise handwritten digits in the MNIST dataset[10]. The deep network is divided into two parts, a CNN that acts as a feature extractor -whose goal is to transform the input- and a fully-connected small network that receives the feature vector and produces the final output -digit classification. Similarly to NEAT+Q, Hyper-NEAT drives the evolution of the topology of the feature extractor, where the fully-connected part is trained using traditional backpropagation with labelled data. Even though the results of the combined algorithm are positive, it suffers from the same issues as NEAT+Q, namely high sampling times. Moreover the algorithm requires labelled data to assess the fitness throughout generations.

Although evolution of CNN and deep networks remains an area of potential research, other authors have proposed attempts to use genetic algorithms in conjunction with CNN including the use of *autoencoders* and CNN as *feature extractors*.

10The MNIST dataset has become a de-facto standard for researchers interested in creating handwriting recognition algorithms

An autoencoder or autoassociator is a special type of ANN used to learn an efficient representation of some data. They can be thought of as a dimensionality reduction technique. Autoencoders ANN goal is to learn a set of weights that lead the network to output the same information as it receives. Formally, given a set of data $x_1, x_2, \ldots, x_n$ the ANN must output a set of n values $o_1, o_2, \ldots, o_n$ that satisfy $x_i = o_i \forall i \in [0, n]$. The key of autoencoder networks is to have one or more hidden layers with less number of units than the size of the input / output layers -a symmetric funnel-like shape-, so the network is forced to learn to compress features from which to reconstruct the original data.

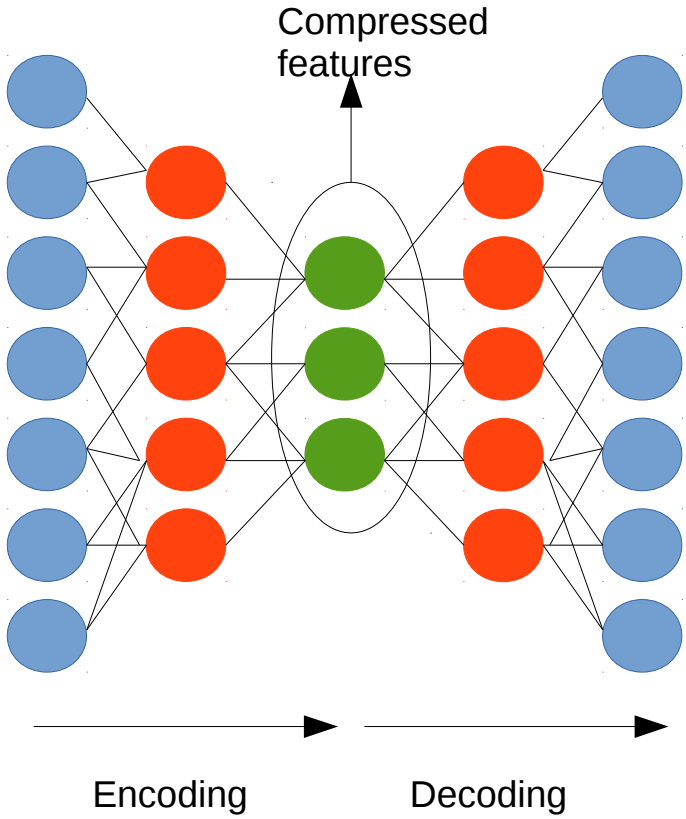

Figure 5. Example of an autoencoder network producing a smaller feature vector.

[37] shows how genetic algorithms can be used to train autoencoder-like networks in an attempt to reduce the complexity of an image. [38] presents a Genetic algorithm – backpropagation hybrid approach to train and tune autoencoders, where a chromosome represents a set of weights on the network; the autoencoders candidates that rank higher undergo a backpropagation training session. The classification error on a sample of MNIST images reached was slightly better than the SVM common alternative (1.44% vs 1.85%).

According to the authors [21] is one of the first uses of deep learning in the context of Reinforcement Learning, certainly the first using Evolutionary algorithms to to train Deep CNN in an unsupervised manner. Their system use the CNN as a feature compressor, reducing the input image to a feature vector which is then fed to a recurrent neural

network (RNN) to choose amongst the available actions -the agent operates on a driving environment within TORCS, so the actions are accelerate/brake and left/right. The intuition here is that the CNN is capable of synthesise a highly-dimensional image into a 3D vector, similarly to a Principal Component Analysis would. The CNN has a fixed topology with four Convolutional-Subsampling pairs and one fully-connected layer, leading to a 3 units output. The image -a driving seat view of the track- is pre-processed to a HSB -Hue, Saturation, Brightness- scheme and only the saturation plane is used.

The main contribution of their work is the unsupervised method used to train the CNN. A genetic algorithm is used where the evolutionary pressure is placed on the amount of diversity that a feature compressor outputs given a collection of images; individuals that are able to generate feature vectors -one per training image- more distant from each other would be ranked higher. The fitness function is:

$$f_{total} = min(D) + mean(D)$$

where D is a list of all Euclidean distances

$$d_{i,j} = |f_i - f_j|, \forall\, i > j$$

between k normalised feature vectors $\{f_1 \ldots f_k\}$. Each of the k vectors corresponds to the normalised output of the CNN for a single image. This approach will be detailed in Section 3.

The features are fed to an RNN that is trained using CoSyNE coevolution [14] to maximise the distance travelled on the track, maximum speed of the car, minimise damage taken. Although the overall performance -tested on a simple track and its mirror for generality- is below than that of hardcoded policies, the agent reaches acceptable levels of driving, completing both tracks without the use of internal engine data. The results suggest evolved CNN can act as feature compressors.

3. Design proposal

3.1. Overall architecture

The goal of this work is to explore and evaluate the use Evolutionary algorithms to aid the training and tuning of deep networks. The control of an agent in a high-dimensional complex visual environment -ViZDoom, a version of Doom- is used as a test task.

Other authors [23] have showed that direct neuroevolution of weights and topologies to fully evolve a visual controller network struggle to produce results. We are interested as well on an algorithm that finds a solution for the complex task in reasonable times in a completely unsupervised manner. Hence approaches such as NEAT+Q [16] -extremely long training times for simple tasks- and Hyper-NEAT feature extractor [36] -uses supervised methods to train and evaluate candidate solutions- are considered unaffordable.

Our approach involves reducing the high-dimensionality problem of dealing with visual input using feature extraction techniques so a neuroevolution algorithm can be applied on a reduced set of features to produce the controller. This work explores the application of EA to both processes: automatic *feature extraction* from raw image data; and the tuning and *training of a controller*.

On a practical level, the proposed algorithm -see diagram in Figure 6- attempts to evolve the weights and (partially) the structure of a CNN to control an agent on a complex 3D scenario based on visual input:

- The convoluted part of the network (*feature extraction*) has a fixed topology and the weights are evolved using a custom GA
- The last part of the network (*controller evolution*) -partially connected layers- is evolved using NEAT; both weights and topology are tuned.

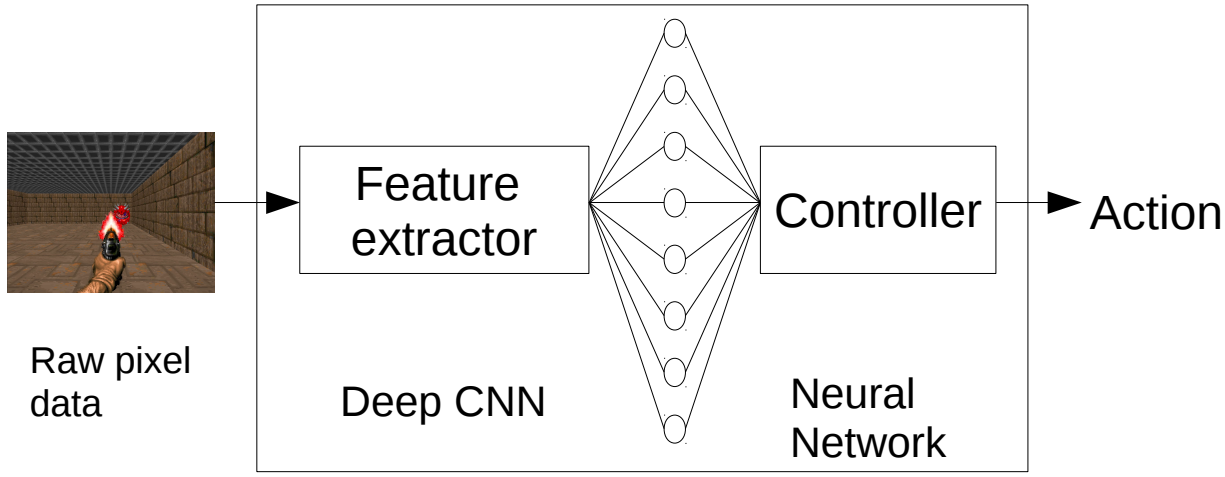

Figure 6. Overview of the two-steps Neural Network approach, with a Feature Extractor and a Controller for ViZDoom agents

3.2. Feature extractor

Inspired by the work of [21] the first part of our algorithm is a Deep CNN that is evolved using a custom genetic algorithm. Figure 7 shows the fixed architecture employed, with three pairs of Convolutional-Subsampling layers and a single fully-connected one. All layers using a ReLU activation function. The number of outputs, i.e. the size of the feature vector, as well as the range of values are the objects of testing -see Section 5.

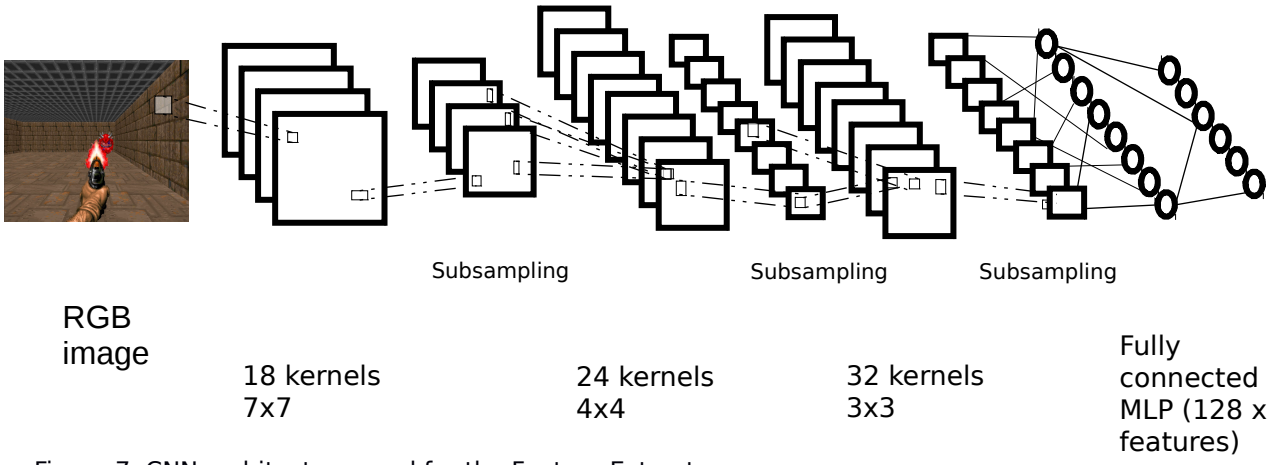

Figure 7. CNN architecture used for the Feature Extractor

A feature extractor approach differs from compression techniques, such as autoencoders, in that by reducing an image to a very small set of features (8, 16 or 32), the algorithm forces the CNN to learn to focus on important elements, losing less relevant information. Note that the goal is not to perform lossless compression, but *precisely to induce loss of unimportant features* so the controller can focus only on those during game play.

A custom GA is used to train the weights of the CNN with the following principles:

- Direct encoding of solutions. Each genome explicitly represents the weights of the network. Figure 8 depicts a partial encoding of a candidate CNN. The genotype-phenotype mapping is done to conserve similarity, as suggested for optimal EA [35].
- Variation operators: 1) mutation of individual weights and of entire layers; and 2) crossover at the layer level -offspring solutions generated by combining layers from the parents- to minimise the competing conventions problem. Figure 9 shows some variation operations and how they produce offspring solutions.

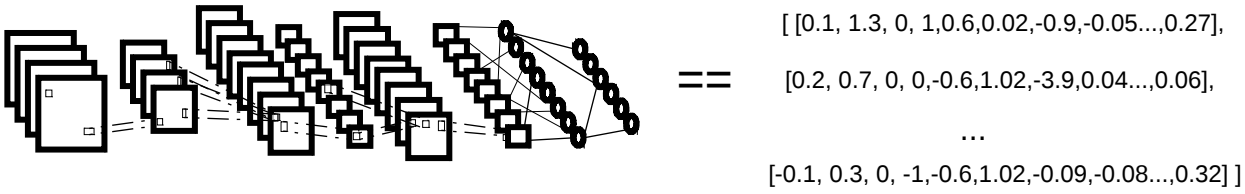

Figure 8. Direct encoding of a CNN for the custom genetic algorithm

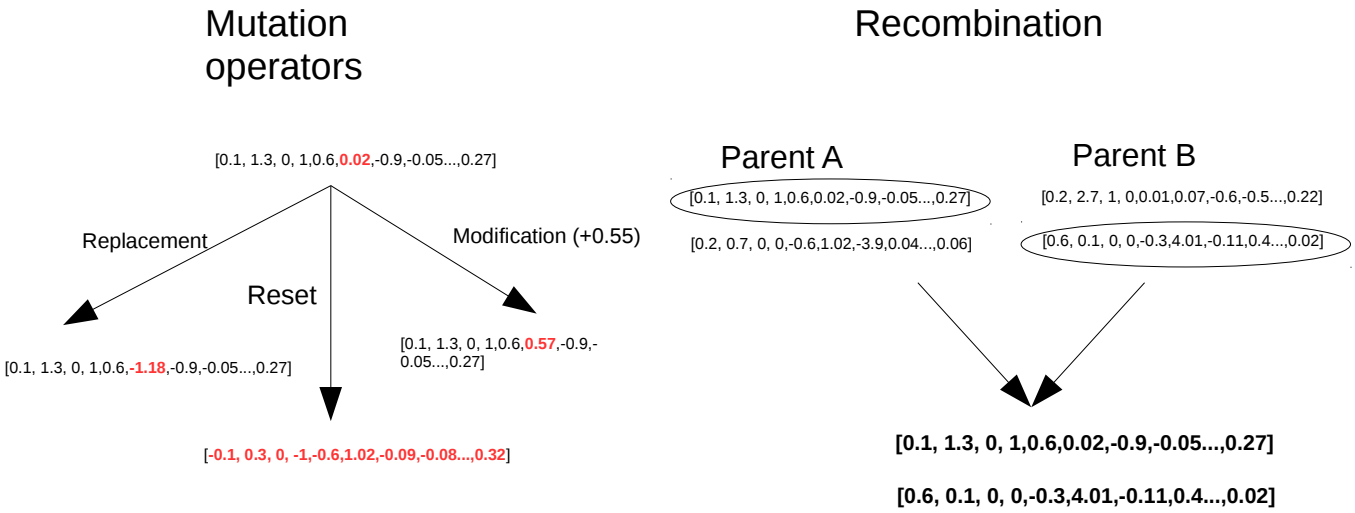

Figure 9. Variation operators for the custom GA (mutation and recombination)

The GA uses a training collection of images representing different elements that an agent may encounter in the scenario -enemies, pick-up items, health packs, etc.)- in various screen locations. A key aspect of a GA is what is used to evaluate candidate solutions, i.e. what is the fitness function and hence what are going to be the evolutionary pressures that drive the process. In [21] the authors use a *distance-based measurement* that encourages solutions to classify test images as points in an n-dimensional space (n = number of features) to be as disparate as possible. We evaluate this and compare to a different measurement based on an information theory metrics called Shannon diversity index[39], originally proposed to quantify the entropy in strings of text and commonly used in ecology as a way of quantifying how diverse and ecosystem is -i.e. more species

and an equal distribution in terms of number of individuals per species would lead to a high Shannon index.

A quality of a good feature extractor is one that produces a distinct set of values when certain elements appear in the image. If we make the features have values between 0 and 1 -using a sigmoid function-, we can then use the *weighted average* of those values to determine the final *species* that image belongs to (if there are 16 features, the image can belong to species 0 to 15); then use the Shannon index to quantify the diversity of the candidate solution over the training set, and measure its fitness based on that -solutions that spread the images into as many species as possible and with equal distribution will be considered better. An alternative to the weighted average is using *binary encoding* to determine the species the image belongs to. Forcing the feature values to be either 0 or 1, the feature vector is interpreted as a binary number, which is then read in decimal to determine the species. Eg.: 0001 would be species 1, whereas 1001 would be categorised as species 9. The collection of decimal values (species) is then used to calculate diversity of the population -i.e. image training set.

Figure 10 shows the overall process of calculating the fitness of a candidate CNN solution, where each image produces a feature vector and then depending on the fitness measurement a global fitness value is assigned.

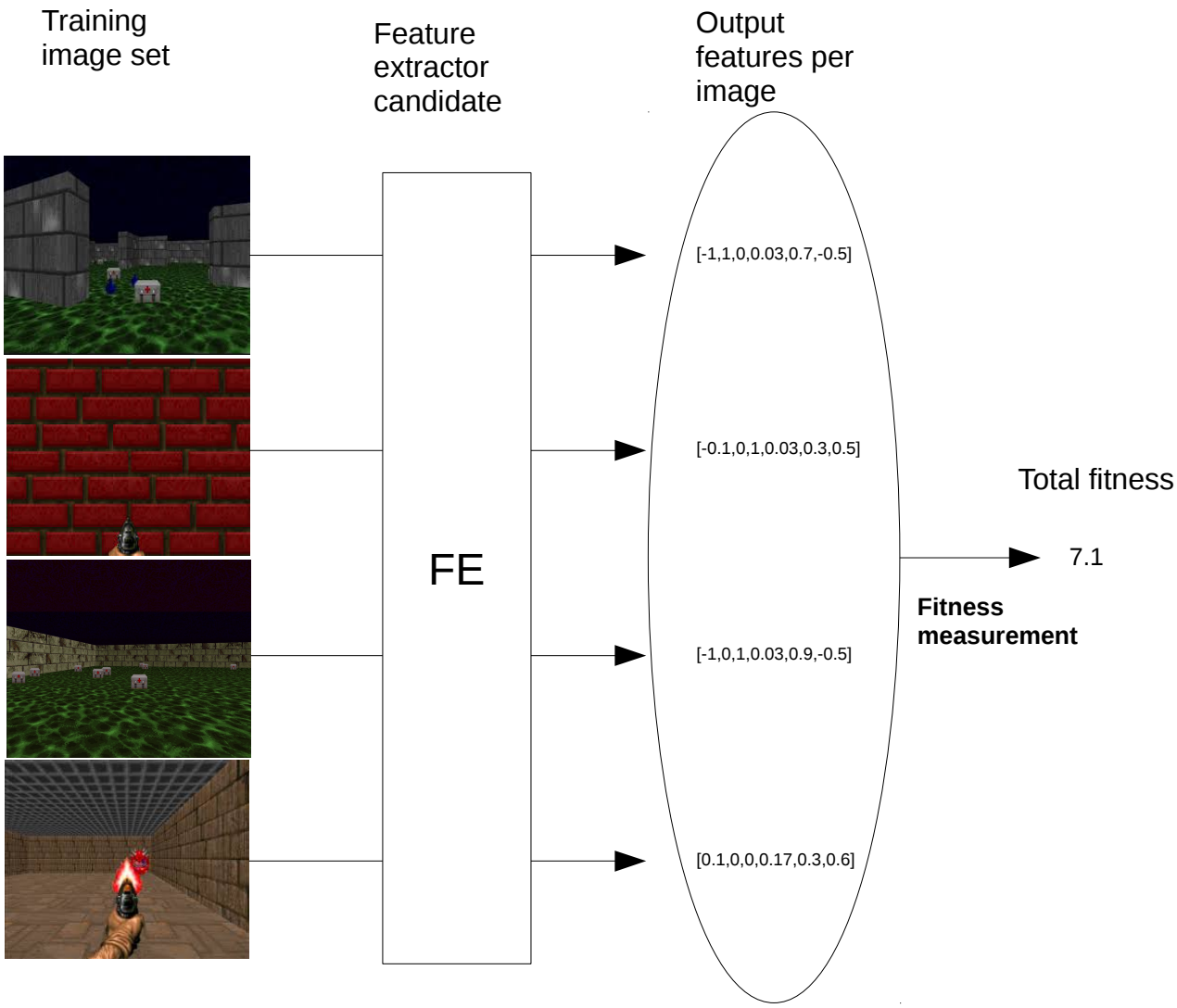

Figure 10. Evaluation of Feature Extractor candidates over a set of training images.

Figures 11, 12 and 13 shown how each of the fitness measurements evaluates the feature vectors to produce an overall fitness value per candidate feature extractor.

- Distance-base: calculates the euclidean distance of all normalised feature vectors to each other. The overall fitness is then $Fitness = min(D) + average(D)$
- Weighted average-Shannon: calculate the species each of the feature vectors belongs to by using a weighted average. Then the overall fitness is the Shannon index of the entire population.
- Binary encoding-Shannon: similar to the weighted average, but the species are determined by the decimal representation of the binary number that is each feature vector.

Image feature vectors

A = [-1,1,0]

B = [-1,0,1]

C = [-1,1,1]

D = [0,0,0]

Distances

	A	B	C	D
A	-	1.414	1	1.414
B	-	-	1	1.414
C	-	-	-	1.732
D	-	-	-	-

Average: 1.329
Min: 1
Fitness = min + average = 1.329 + 1 = **2.329**

Figure 11. Fitness measurement based on euclidean distance

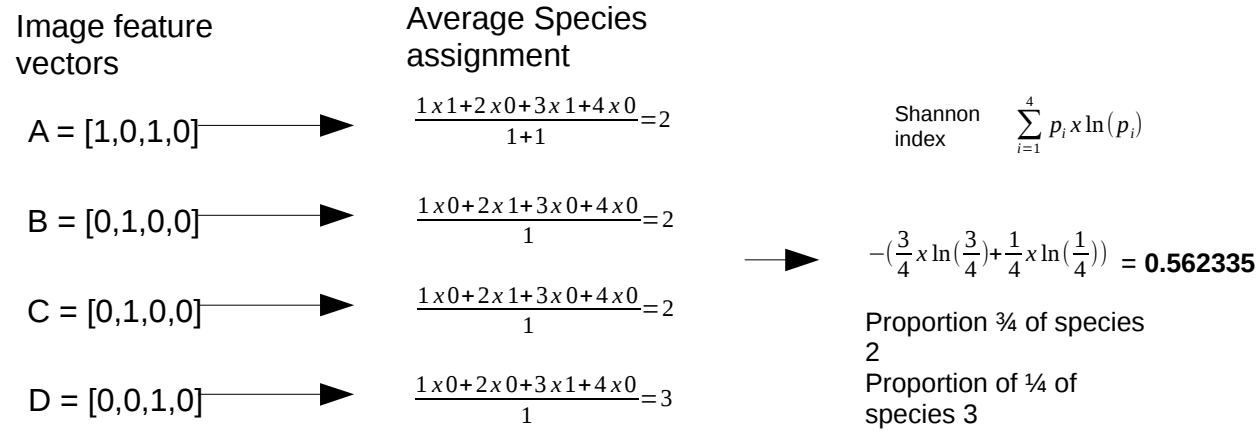

Figure 12. Fitness measurement based on Shannon index, species calculated as the average of feature vector

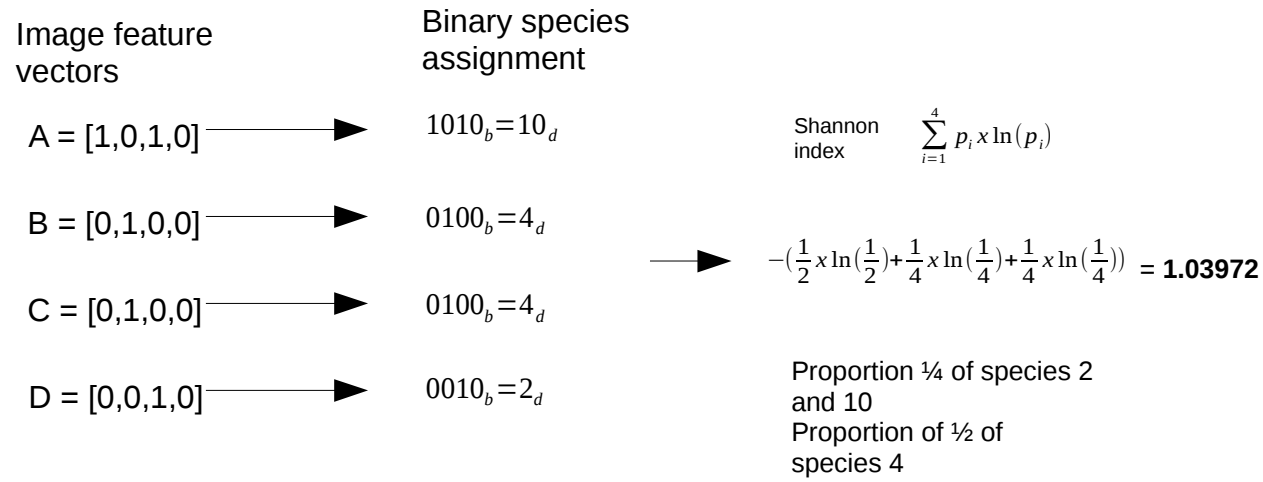

Figure 13. Fitness measurement based on Shannon index, species calculated as the binary encoding of the feature vector

3.3. Controller evolution

Once a simplified version of the input image is obtained, the problem is modelled as a Reinforcement Learning task where the goal is to find the best policy to map feature vectors and actions in the scenario. This work evaluates the use of NEAT and its variants to produce a (possibly recurrent) neural network that outputs the action to be taken at each decision point. Two variants are tested: basic NEAT and FS-NEAT (Feature selection NEAT), which starts with only some inputs connected to the network -selecting only the features that are most relevant.

Two alternative action selection procedures are tested -see Figure 14: 1) *action selection* or winner-takes-all forces the EA to search in the value function space by assigning a value to each possible action and then selecting the one with highest score; whereas 2) *axis based selection* makes the EA search in the policy space by explicitly mapping a state to a concrete action.

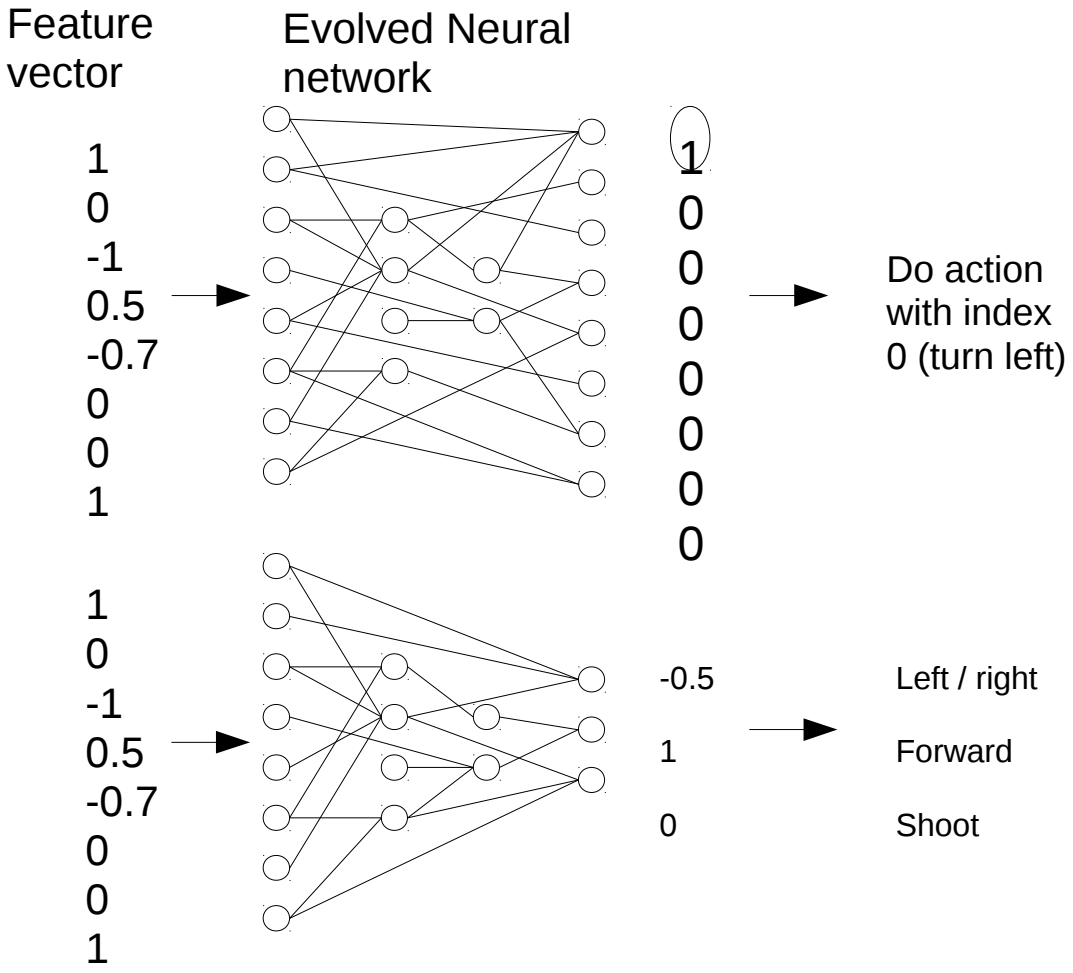

Figure 14. Example of a neural network topology evolved to control a ViZDoom agent. Top: a network that selects the action with highest value (value function space); bottom: a network that explicitly maps a feature vector with the action (policy space)

Each candidate solution is evaluated by letting it control an agent in a ViZDoom scenario. The total reward gathered is used as its fitness. Each scenario awards rewards differently. To help with the credit assignment problem (identify actions that led to high rewards or high punishments from a single final reward value), shaping rewards are introduced for performing actions that are deemed as positive -such as picking up health packs or ammo. A description of each scenario and its rewards is included in Section 4.

Recurrent links are allowed to arise, but only 1 level of recursion is permitted. since we only do the forward pass 4 times. This allows output to hidden recursion, or hidden to hidden. Up to 2 hidden layers are allowed. This limitations are enforced for performance reasons and it is assumed it does not affect the network capabilities -two hidden layers are enough for non-linear classification.

4. Experimental design

4.1. Frameworks

4.1.1. ViZDoom

ViZDoom [11] is a novel framework developed for the advancement of visual Reinforcement Learning techniques. It is based on an open version of the Doom game -ZDoom[11] tuned specifically to facilitate RL tasks and with direct access to the image buffer. The framework allows the use of various custom rewards to be associated with certain actions, as well as granting access to game-specific variables such as ammo, health, armour level, etc., in multiple extendible scenarios -with enemies, items, health packs, etc.

The authors of ViZDoom demonstrate that the framework can accommodate strategies as complex as Deep-Q-Networks. In their original paper they show DQN is capable of successfully learning simple find and shoot the enemy task and shows promise on a more complex health pack gathering scenario -both scenarios will be covered in detail in Section 4.

The framework has the potential to become a standard test-bed in the field of visual RL. To aid in that goal, the authors have organised a competition to call for solutions in a series of challenging scenarios[12].

4.1.2. Keras

This work makes use of Keras, a comprehensive API that offers simple and comprehensible access to many deep neural network functionality useful for our tasks, such as functional layers -fully connected, recurrent, convoluted, etc.- and learning algorithms -backpropagation, stochastic gradient descent.

The reasons for the choice of Keras as deep learning framework above others are:

- Fast: it can run on GPU as well as many CPU cores
- Solid standard: Based on Theano and TensorFlow.
- Easy to use directly in ViZDoom with python interface.

11 http://zdoom.org/News accessed on 01/09/2016

12 Visual Doom AI Competition is held as part of the CIG conference of 2016. http://vizdoom.cs.put.edu.pl/competition-cig-2016 accessed on 01/09/2016

Other options considered were:

- Theano and TensorFlow: in which keras is based but somewhat more difficult to use.
- Caffe: complex to setup and configure, hyperparameters cannot be easily configured programmatically.
- Lasagne: less comprehensive than Keras.
- Torch: does not have a python interface.

4.2. Tasks and scenarios

As the task is set as a Reinforcement Learning problem, it is important to describe how the scenarios are configured and what constitutes a reward. Careful consideration is taken when designing scenarios to avoid the evolutionary algorithm to take advantage of geometrical exploits (eg.: if health packs are placed always near walls, an undesirable exploit would be to find a wall and go along it, ignoring everything else); or regularity exploits (eg.: enemies placed in a regular interval from the player, for instances always spawning in front of the initial position; an undesirable behaviour of standing still and constantly shooting could arise). The two examples of exploits were found quickly when this considerations were not taken.

The two scenarios used in this work are *pursuit and gather* and *health gathering* -Figure 15 includes a floor plan and a screen-shot of each. Both introduce a constant poisoning to the player to discourage solutions from remaining standing still -hence solutions that explore and pick health packs are favoured. In the case of *pursuit and gather*, if the agent remains idle, the poisoning would kill it in 14 seconds. In *health and gathering* it would do the same in 8 seconds.

4.2.1. Pursuit and gather

This scenario contains one single irregularly shaped room where the player starts at the centre. Demons -melee attacking enemies- are spawn in various parts of the room. Health packs and ammo clips are scattered around. The player is allowed to move forward, turn left and right and shoot.

The player receives the following rewards -those that are shaping reward only are marked with an [S]:

- +100 for killing an enemy
- +1 per tick alive
- -35 per bullet fired [S]

- +75 per health pack picked [S]
- +50 per ammo picked [S]

4.2.2. Health gathering

The player is placed on a maze and it is allowed to move forward and backward and turn left and right. The goal is to survive for 1 minute walking in poisonous ground by regularly collecting the health packs that are scattered around and avoiding poison vials.

The player receives the following rewards:

- +100 per health pack picked [S]
- -100 per poison vial picked [S]
- +1 per tick alive

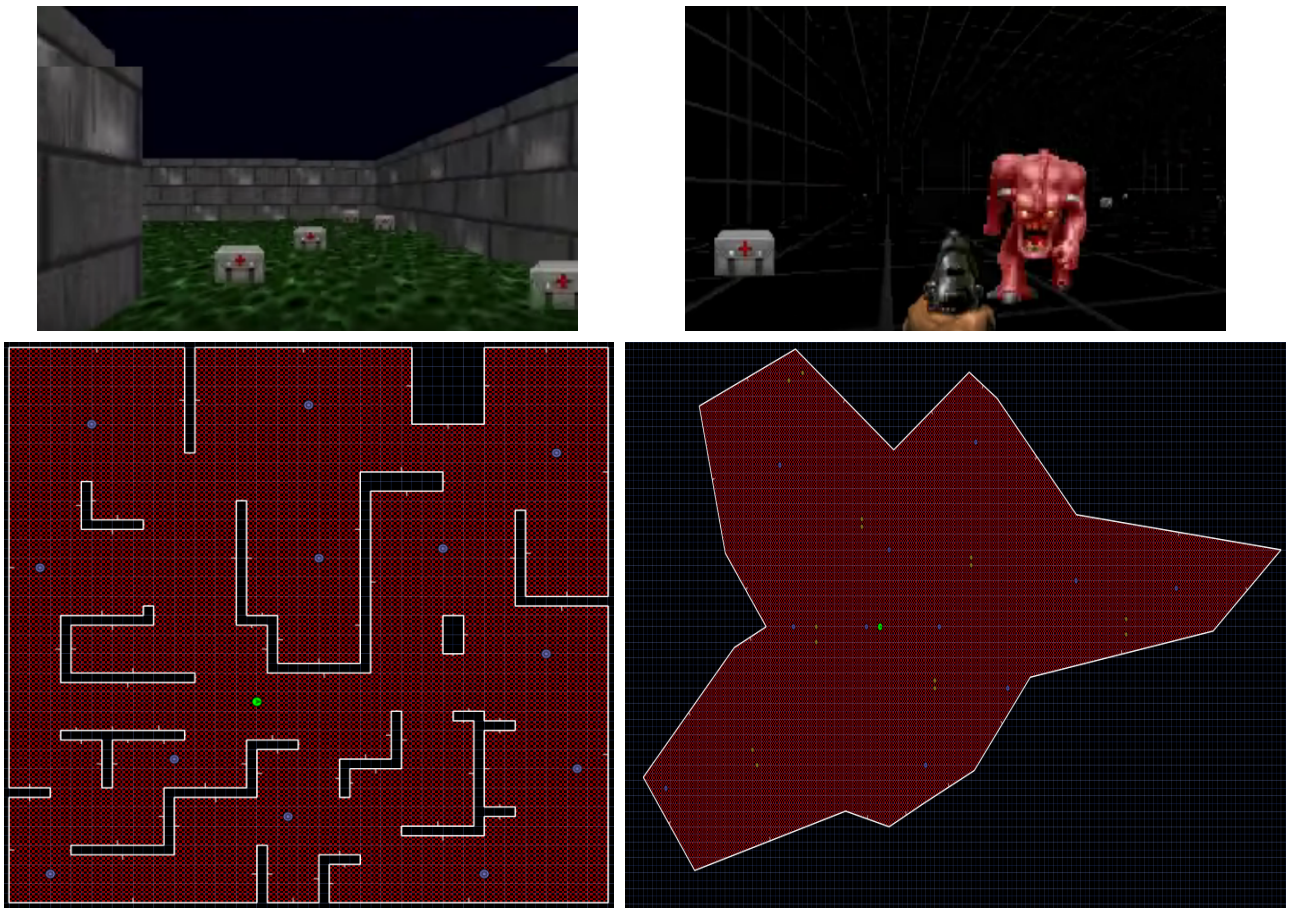

Figure 15. Floor plans and screenshots of the two ViZDoom scenarios employed. Left: health gathering scenario; right: pursuit and gather

4.3. Experiments

4.3.1. Assumptions and design decisions

For simplicity, only one weapon and one enemy type are considered:

- Pistol was chosen as it requires the agent to have perfect aim without having splash damage.
- Demon enemy was chosen as it is visually less complex (mostly pink, easier to identify) and only deals melee attacks.

The bulk of the testing is done on a **single scenario** -pursuit and gather- for the following reasons:

- Offers high complexity to solve it -requires killing enemies and collect health packs to survive, also avoid enemies and pick ammo.
- Simplicity → combinatorial explosion with testing all solutions in the 2 scenarios
- The other scenario is used to test final performance of the best candidate solution -to compare it to other algorithms (full RL and supervised CNN)

To ensure statistical significance throughout testing each trial is *repeated five times* and the results are averaged. When training and testing, the simulations are run with a **skiprate** of 3 (the agents only decide what action to take every 4^{th} frame, each decision made is continued throughout 3 frames) to accelerate the training process -shown to affect positively the process in [11]

4.3.2. Evaluation of the Feature extractor evolution

The Feature extractor (FE) candidates are evolved using the custom GA described in Section 3 with the following parameters: mutation rate 0.35, elite ratio 0.05, population size 100, epochs 400.

Three candidate solutions are tested, depending on the fitness measurement and hence the evolutionary pressure: euclidean distance, weighted average Shannon diversity and with binary encoding. All of them are compared to a randomly generated network -baseline.

To assess the importance of the size of the feature vector, three **different lengths for each candidate solution are tested**: 8, 16 and 32.

Since candidate solutions are trained in an unsupervised manner and using different fitness measurements, direct comparison is not possible. To compare them, each individual FE is used to train a Controller using standard settings:

- NEAT to drive the evolutionary process, using winner-takes-all action selection, controller output range [-1,1], standard NEAT parameters (population size 100, mutation rate 0.25, crossover rate 0.70, mutation operators add, remove, modify link, add, modify neuron unit, elite ratio 0.1, run for 230 generations). To measure the fitness of a solution, 2 full runs on the scenario are done.

To make the final comparison, two sets of data are compared:

- **Learning graph**: comparing the fitness of the best and average solution per generation; the focus is on improved fitness over generations.
- **Performance graph**: the best of each generation is evaluated 4 times in the scenario; focus is on improved performance over generation. Two types of graphs are provided: *reward only* (performance measuring rewards only and not shaping rewards) and *shooting reward only* (performance measuring shooting and killing rewards only).

4.3.3. Evaluation of the Controller evolution

The best candidate from the FE experiments described in Section 4.3.2 (size of feature vector, fitness measurement) is selected to carry out further experimentation on the Controller evolution. The goal is to quantify the effect of certain parameters on the evolution of the Controller in the final in the final performance of an agent.

The evolution is driven by the NEAT algorithm using the following parameters: population size 100, elite ratio 0.1, crossover rate 0.7, mutation rate 0.25, and innovation protection (young age 5, young boost 1.5). Hidden units have ReLU activation function. Total epochs 230.

To avoid combinatorial explosion of experiments, the tests are carried out sequentially, focusing on one element at a time. The parameters used are the standard ones mentioned in section 4.3.2 unless otherwise stated.

1. **Network output function** signed sigmoid (output range [-1,1] in a sigmoid curve) or linear (output range [-R,R] linear)
2. **NEAT vs FS-NEAT** to determine if allowing feature selection affects performance
3. **Action selection procedures**: Winner-takes-all vs axis-based action -this determines whether the neuroevolutionary serach is done in the policy space or in the value function space.

As with the FE evolution, learning and performance graphs are used to evaluate the results.

4.3.4. Benchmark comparison

Finally the best FE – Controller candidate is then compared with two benchmark solutions: supervised CNN and Deep-Q-Network. The comparison is done over two scenarios -pursuit and gather and health gathering- each run for 100 episodes and the results averaged. To add statistical significance, all tests are repeated five times.

The **supervised CNN** solution is trained directly from a collection of images from the scenario paired with the expected action to be taken. The training set is generated from human gameplay, and the CNN is trained using stochastic gradient descent with batch size 64 and for 1000 epochs.

The **DQN solution** uses replay memory size 10000, with 5000 training steps per epoch and run for 100 epochs, with e-greedy strategy with decaying epsilon to select the best action.

For the comparison to be fair, benchmark solutions use the same Deep CNN architecture as the one proposed in Section 3. Only the best candidate of each approach is compared against the others -for supervised CNN and DQN there is only one candidate (after all training), but for the EA strategy the highest scoring generation champion from the last 10 generations is chosen.

5. Results

5.1. Feature extractor evolution

To make the results more readable and comparable, the graphs for each feature vector length are joined together.

Figure 16 shows the learning graph for all fitness measurement types per feature vector length. Each value shows the best fitness on a training generation. With the exception of Shannon binary encoding with feature vector length 16 and 32, all experiments showed better measured fitness than their random baseline -blue line on the graphs.

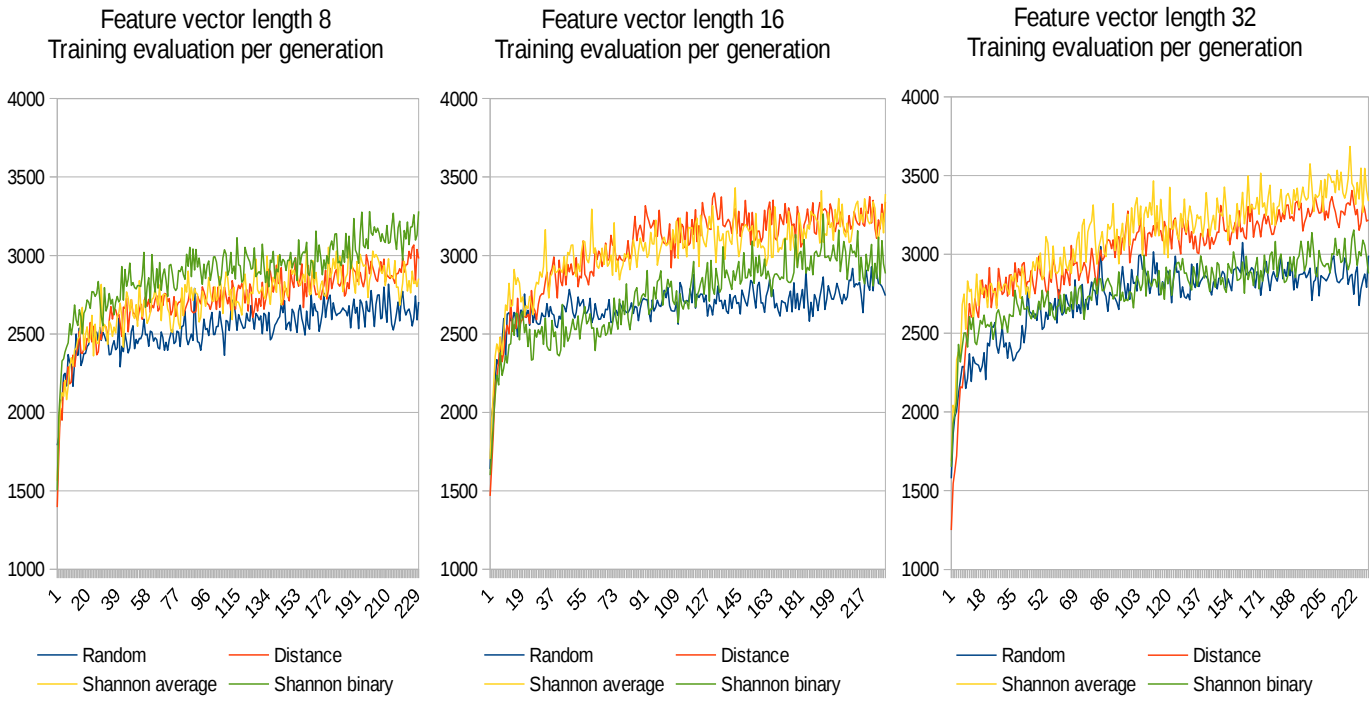

Figure 16. Learning graphs per fitness measurement and feature vector length

The performance graphs in Figure 17 (rewards only) and 18 (shooting rewards only) measure the actual average performance of each generational champion on 4 episodes. For simplicity, only the best across all vector lengths are shown (8–Shannon binary, 16–Shannon average and 16–distance). However, full results for each feature vector length are included in the Appendix.

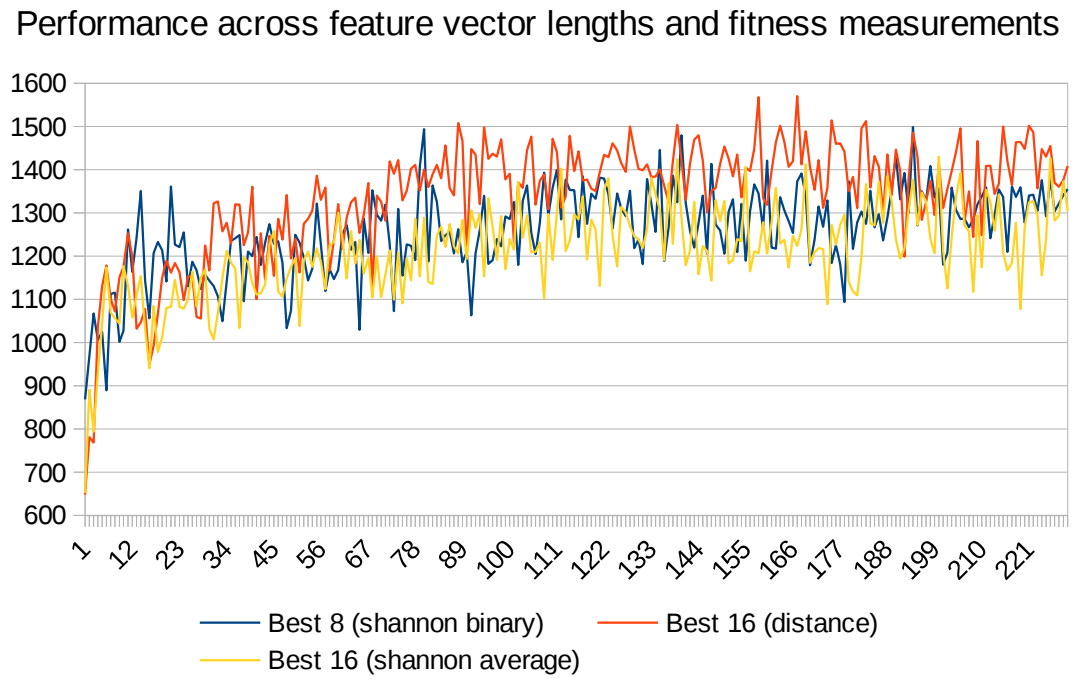

Figure 17. Total rewards per fitness measurement and feature vector lengths

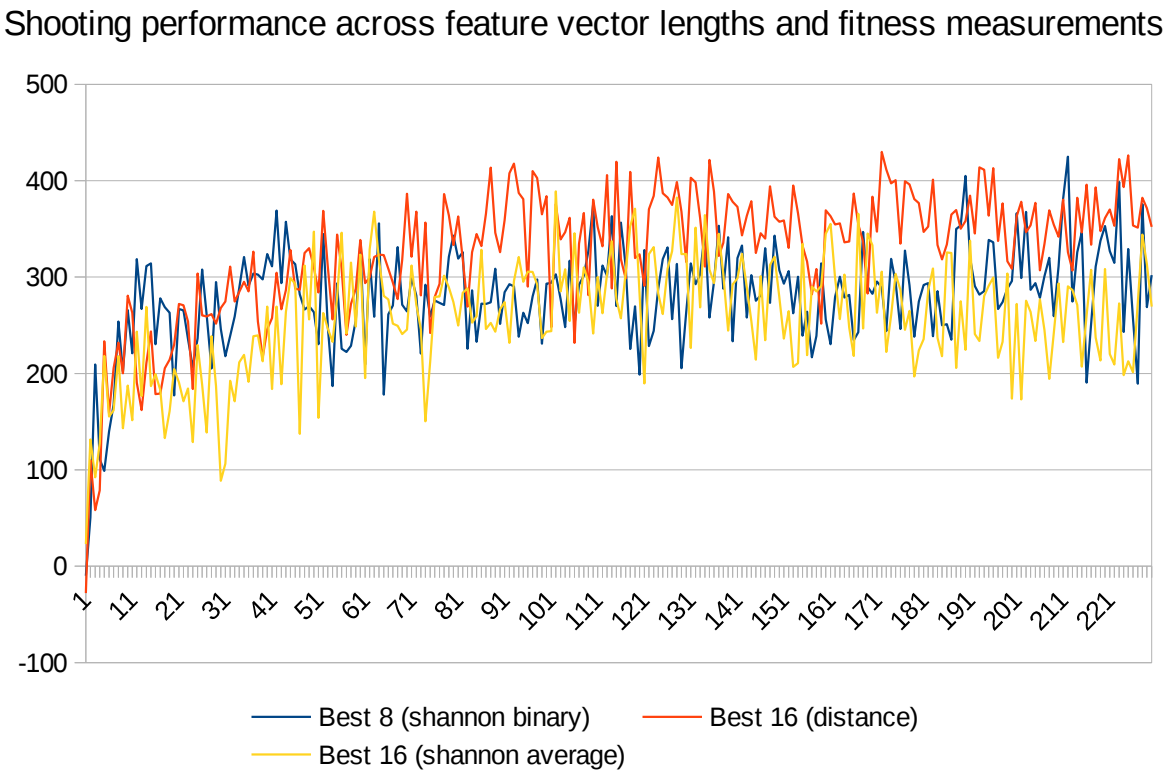

Figure 18. Shooting rewards across fitness measurements and feature vector lengths

Learning and performance graphs indicate that the best fitness measurement and feature vector length combination is 16-distance. For confirmation, Figure 19 and 20 show the performance graphs for all feature vector lengths (8, 16 and 32) using euclidean distance.

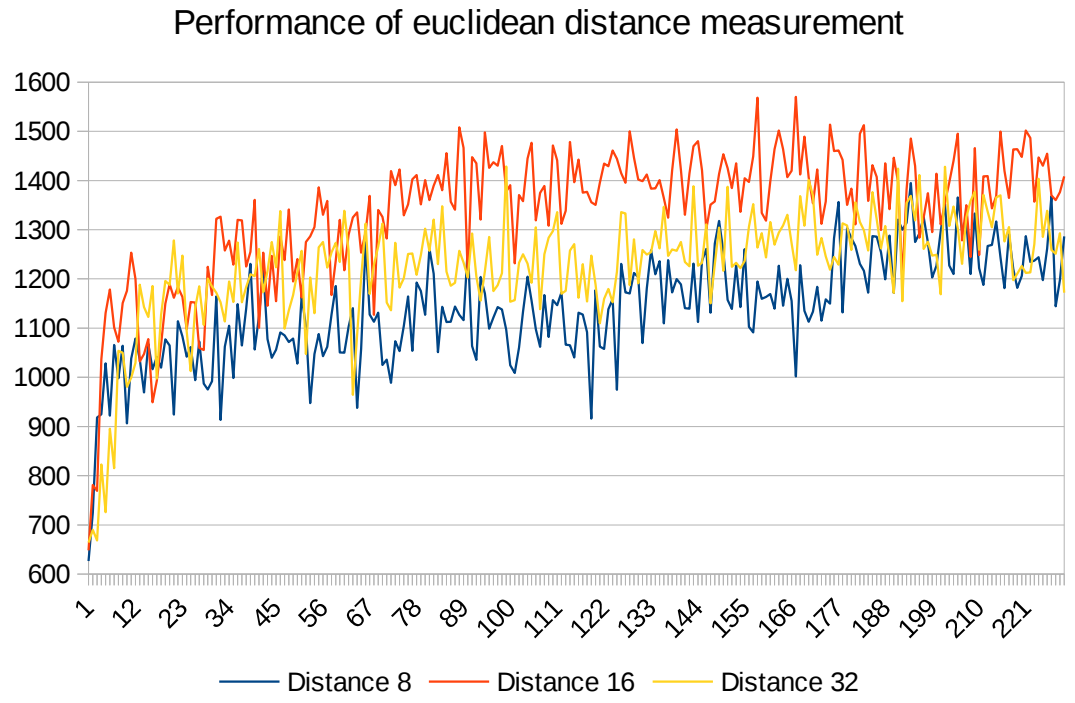

Figure 19. Total rewards for all feature lengths using euclidean distance

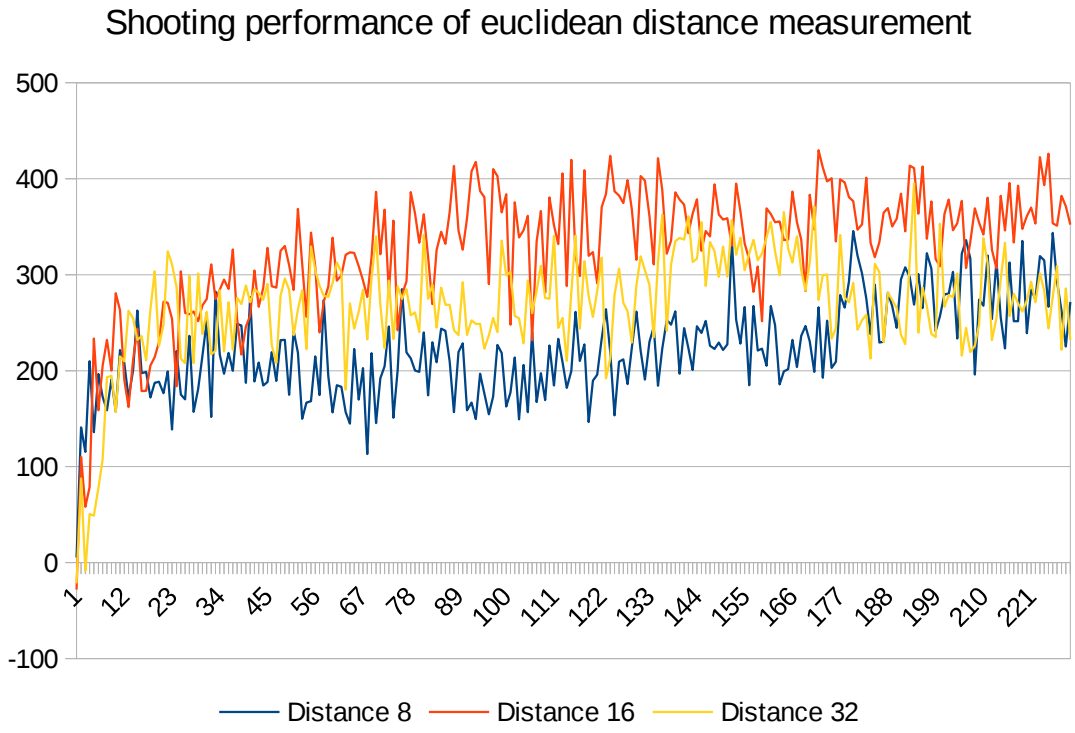

Figure 20. Shooting rewards for all feature lengths using euclidean distance

5.2. Controller evolution

Using 16-distance feature extractor, the following experiments determine the impact of certain design decisions on the neuroevolution of the controller. Three tests are run:

1. Nature of the output function (sigmoid vs linear)
2. Neuroevolutionary algorithm employed: FS-NEAT vs NEAT
3. Action selection method (winner-takes-all action selection vs axis-based action)

5.2.1. Network output function

In the winner-takes-all action selection strategy, the agent chooses the action which output contains the highest value. Thus, the type of activation function used in those units may affect the decision making ability of the overall system. Two types of activation functions are tested here: signed sigmoid (range [-1,1] and with a sigmoid shape) and linear (range [-R,R] with linear shape). Figure 21 shows the learning graph for both options, indicating the linear alternative seems to reach a higher fitness total at the end of the training.

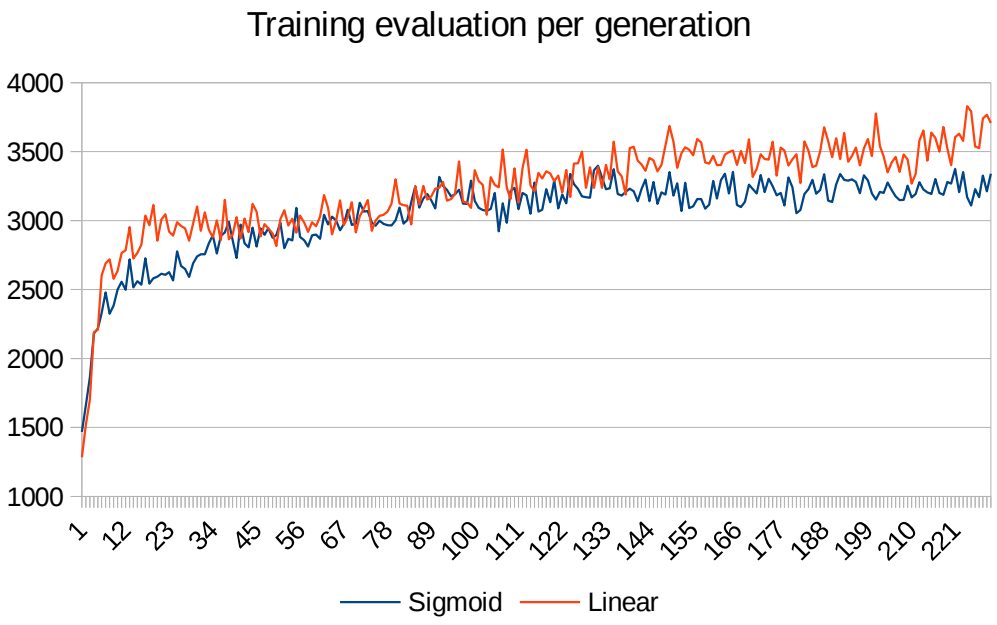

Figure 21. Learning graph for alternatives using Sigmoid vs Linear output activation functions

However, performance graphs in Figure 22 and 23 do not show the actual performance is significantly different across the training process. This indicates that in terms of shooting and main rewards (time alive and enemies killed) there seems to be no difference between the alternatives; but training fitness was higher for linear, suggesting it did better in shaping rewards such as picking ammo clips -which are only counted in training.

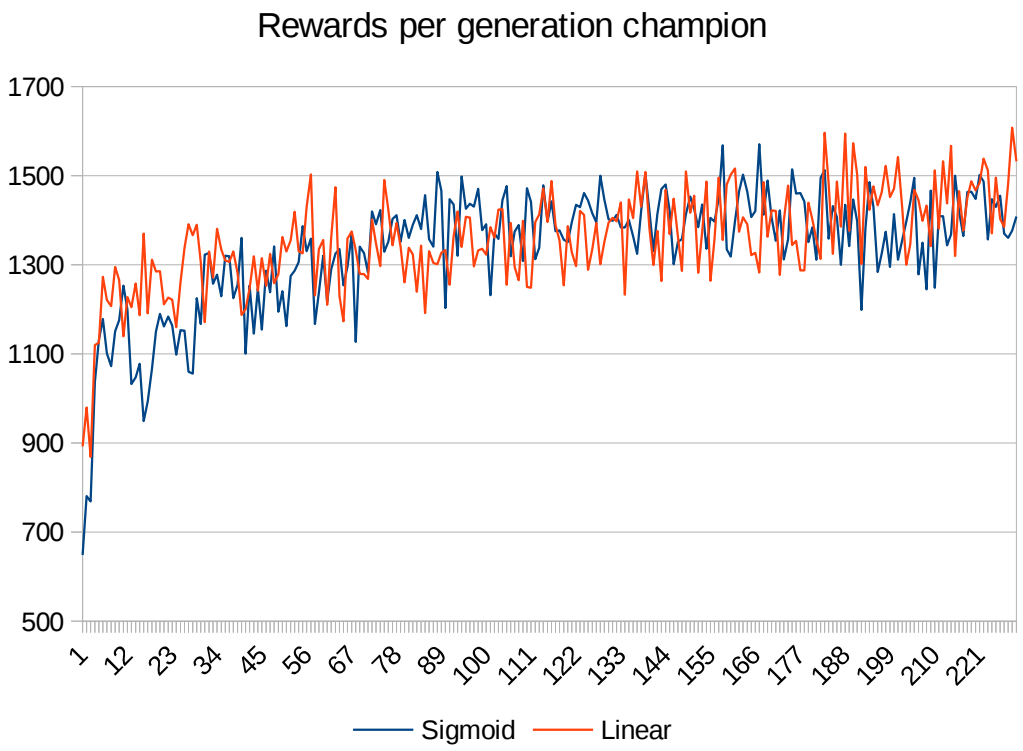

Figure 22. Total rewards for Sigmoid and Linear output activation function alternatives

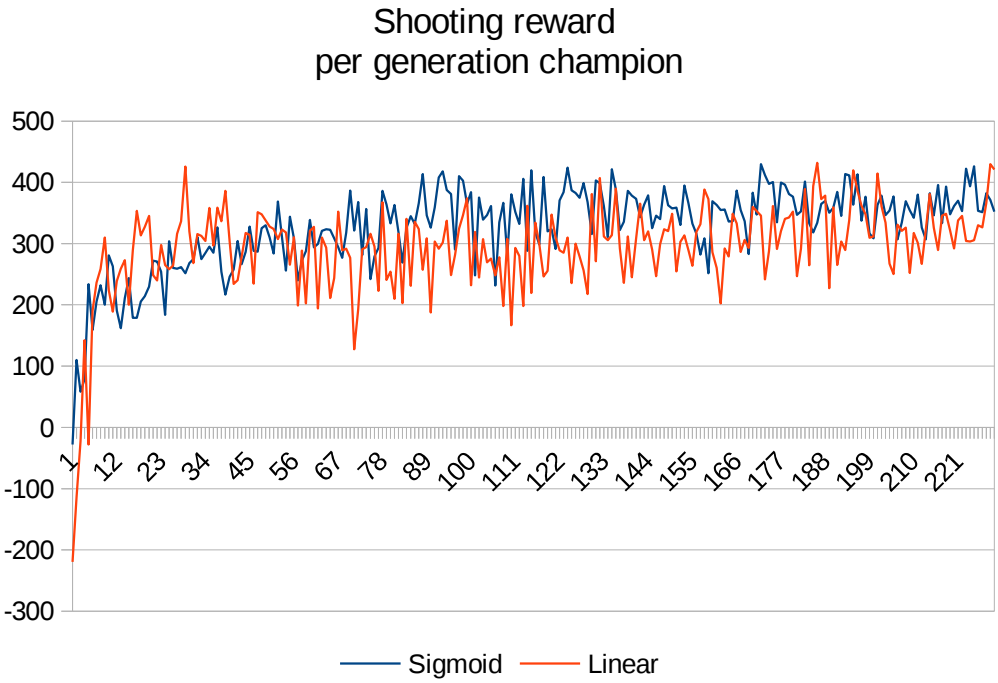

Figure 23. Shooting rewards for Sigmoid and Linear output activation function alternatives

As a final test to see if linear activation may be better than the signed sigmoid alternative, the best candidate solutions[13] for both alternatives is compared over 100 trial episodes, counting rewards only (not shaping rewards). Table 2 clearly shows that the

13Best candidate selected as the one with higher fitness amongst the last 30 generation champions

linear champions scored significantly higher than the sigmoid. The standard error is calculated as $St.error = \frac{\sigma}{\sqrt{n}}$ where σ is the standard deviation on the 100 trials and *n* is the size of the population (in this case, 100).

	Sigmoid	Linear
Average	1423.262	1592.74
St. error	13.646649435	16.93109106

Table 2. Performance comparison Sigmoid vs Linear output functions on 100 trial scenarios

Although the results are very similar, linear is considered better because:

1. Learning graph is ascending and higher, as opposed to the stabilised sigmoid, indicating it does better with shaping rewards.
2. Performance of the best individual is higher in linear candidate.

5.2.2. Neuroevolutionary algorithm and action selection

The last two tests, neuroevolutionary algorithm used and action selection method, are combined in order to understand if one NE does better with a particular action selection. Hence, the following combinations are tested:

1. NEAT with winner-takes-all action selection
2. NEAT with axis-based action
3. FS-NEAT with winner-takes-all action selection
4. FS-NEAT with axis-based action

All the alternatives use linear output activation function and the standard settings. However, the FS-NEAT options are allowed to run for 650 instead of 230 generations to allow the algorithm to select as many input features as it needs -it was observed that at around 650 generations, an FS-NEAT evolution process would have had enough time to incorporate all the inputs.

Figure 24 shows the learning graph for all 4 alternatives, with NEAT-axis action as the clear superior option. Looking at the FS-NEAT lines, the graph is what would be expected if all or most of the input features were relevant: fitness significantly below the NEAT alternatives until the algorithm has had a chance to incorporate more input features.

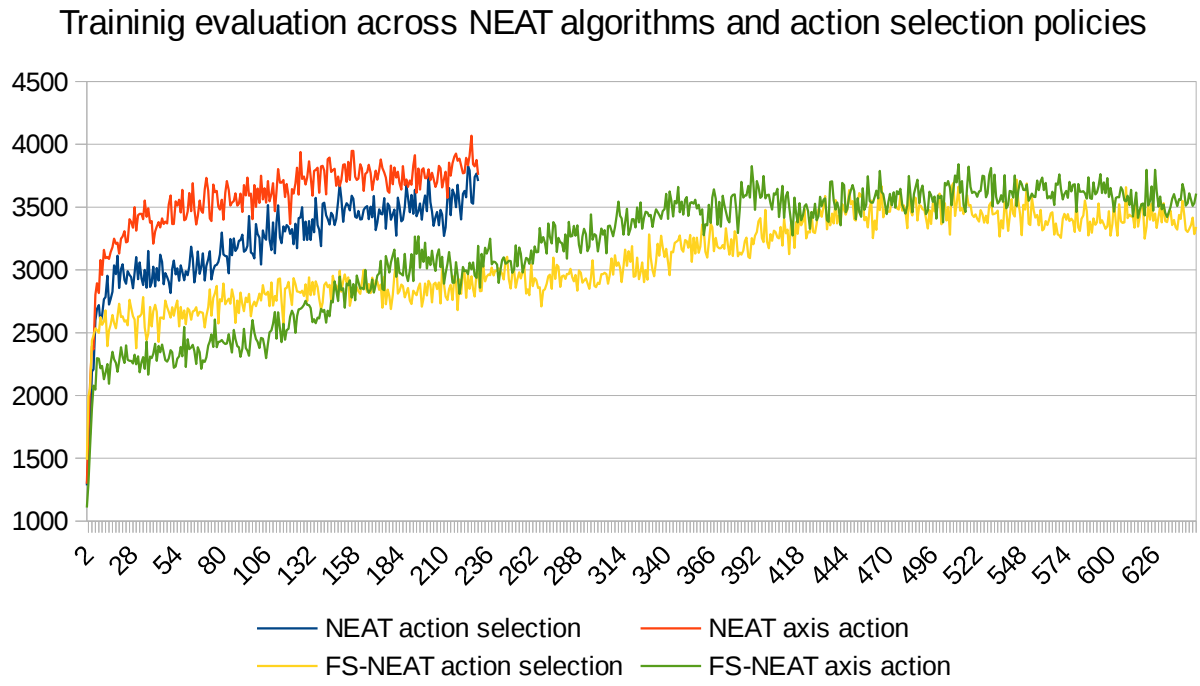

Figure 24. Learning graph comparing neuroevolutionary algorithms and action selection policies

The performance (reward only and shaping rewards) is plotted in Figures 25 and 26, confirming NEAT-axis action as the superior candidate.

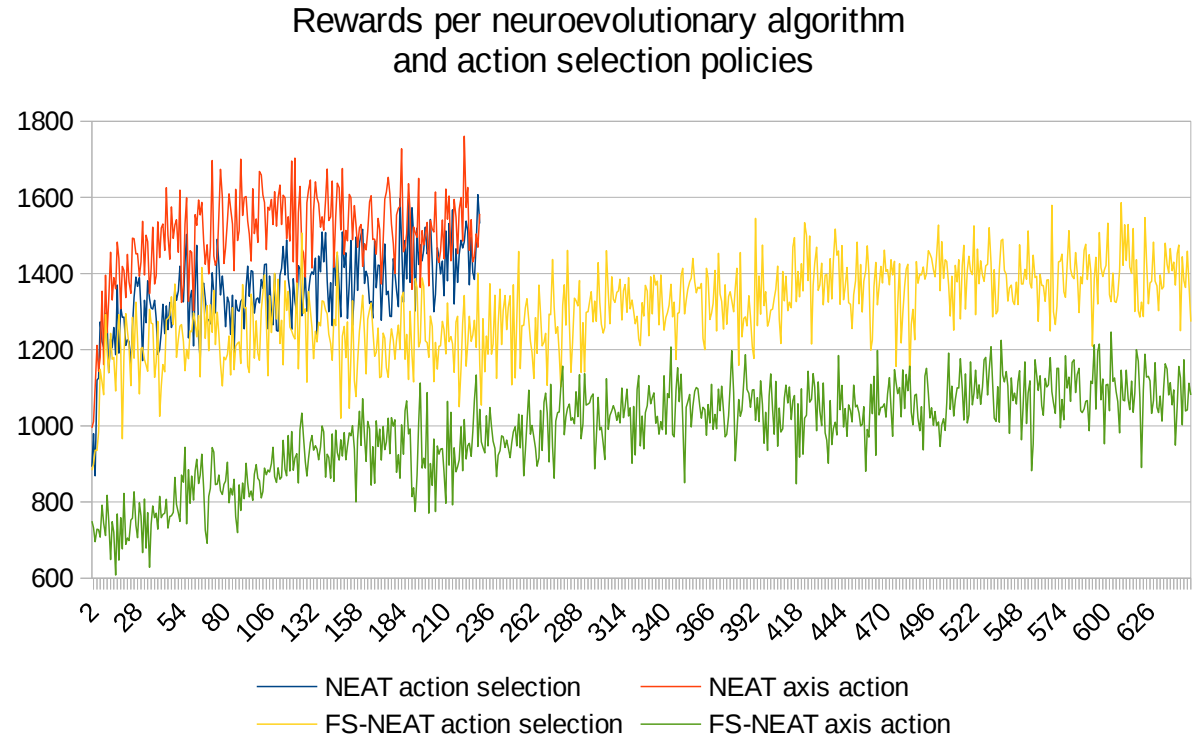

Figure 25. Total rewards for different Neuroevolutionary algorithms and action selection policies

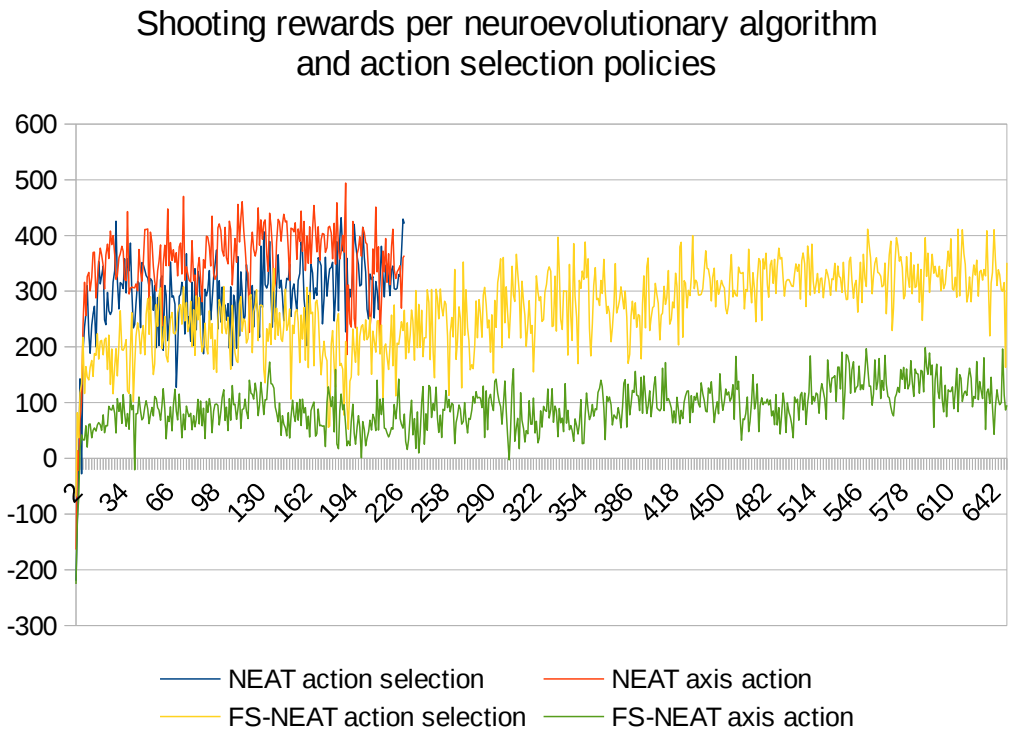

Figure 26. Shooting rewards for different Neuroevolutionary algorithms and action selection policies

For a final comparison, the performance of the best candidate solutions for each alternative is averaged over 100 trial episodes. Table 3 includes the results, with NEAT-axis action as the candidate with better scores; and FS-NEAT as consistently below the NEAT counterpart.

	NEAT winner-takes-all	**NEAT axis action**	**FS-NEAT winner-takes-all**	**FS-NEAT axis action**
Average	1592.74	1650.76	1424.752	1118.76
St. error	16.9310910602	15.5974139987	14.9136533117	14.7675356447

Table 3. Performance comparison of different neuroevolutionary algorithms and action selection policies on 100 trials

5.3. Comparison benchmark

The best EA candidate found after experimentation is:

- Feature extractor with 16 features, evolved using euclidean distance measurement
- Controller network output Linear activation function
- Controller evolution through NEAT

- Axis action selection

In the last test, the EA strategy is compared against two benchmark solutions: supervised CNN trained from human game play, and DQN reinforcement learning. As with the other tests, each alternative is trained five times and each time the average performance measured over 100 trial episodes; thus, the scores are a final average over 500 episodes. Figure 27 shows the results for both scenarios, with the EA strategy outperforming the other two approaches.

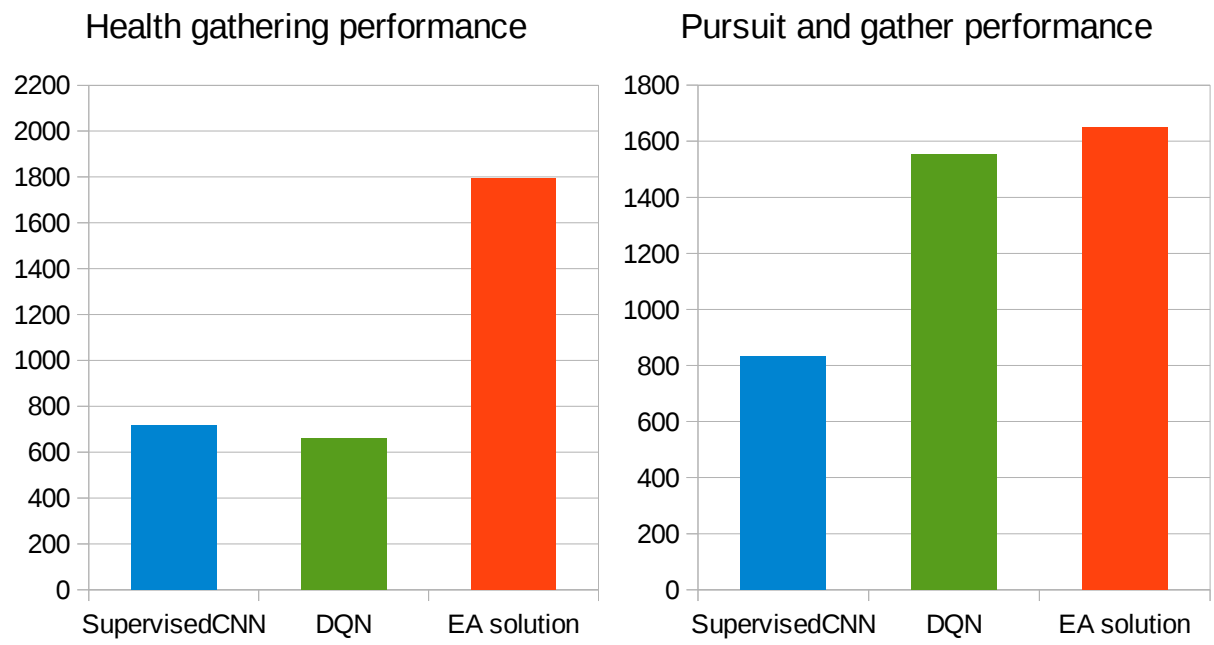

Figure 27. Evolutionary solution compared to benchmark alternatives

6. Research findings and analysis

6.1. Diversity measurements

The results in this work suggest that the idea in [21] can be extended to a more complex scenario to produce good results. A CNN is a valid tool to act as a feature extractor of RGB images in 3D maze-like environments in a rich decision space (move, shoot, pick up items, etc.). Section 5.1 provides evidence to accept diversity-based fitness as a good evolutionary pressure to select a feature vector from images. Shannon diversity with binary encoding and with weighted average achieve good performance levels, but using the euclidean distance to measure diversity gives the best performance overall.

One thing to note is the poor performance of Shannon binary encoding with feature vectors of length 16 and specially 32. They are not significantly better than the random baseline. This is somewhat expected given the way the Shannon binary encoding determines the species of each image. Recalling from Section 3, when using a binary encoding the species is determined by interpreting the feature vector -which can have features as 0 or 1- as a binary string. The goal evolutionary process is to capture as much diversity as possible, which means that if at any point a candidate solution is able to separate all the training images in different species, it should stop, as the maximum diversity is reached. Naturally, when the size of the string -the length of the feature vector- is short, it is harder for the CNN to reach this limit. With a length of 8 there are $2^8 = 256$ possible species; in a normal training set there are 90-100 (depending on the scenario) images, which means to assign one species to each image would require the CNN to be fine tuned. When the length of the feature vector is increased, the number of possible species increases exponentially -65,536 for 16 digits, 4,294,967,296 for 32. When the number of possible species is this high, the evolutionary process finds it easy to stop early; however, this results in networks that are almost as good as random, thus the poor performance displayed.

Despite being randomly generated, the Controllers evolved from random feature extractors resulted in suboptimal but acceptable behaviour; generally they exploit one behaviour (rotate one side, move forward) until an enemy is found, then shoot -with some accuracy. Items -health packs, ammo clips- were largely ignored.

All diversity approaches -Shannon indices and euclidean distance- performed significantly better than the random behaviour, displaying wall awareness (turning when facing a wall) and a tendency to seek items and enemies.

The fact that a randomly generated feature extractor can lead to partially solving a complex scenario such as *pursuit and gather* indicates the strength of the approach -reducing image complexity and applying a NE algorithm.

6.2. Learning and performance graphs

Throughout this work two types of graphs have been used to measure candidate solutions: learning and performance graphs. Though they fundamentally quantify the ability of an agent to solve a particular scenario, each has a particular focus.

The learning graph shows the process that drives the evolutionary progress. Each value corresponds to the fitness value of the generational best candidate and it includes all types of rewards -normal and shaping rewards. Thus, by looking at the graph it gives an indication of how well the solution is doing overall.

The performance graphs on the other hand focus exclusively on a type of reward. They too are built with the generational best candidate, but on the average score on four different episodes. Depending on the type of performance graph, a kind of reward is counted: for reward only graphs, shaping rewards are ignored; for shooting reward only, the score just takes into account killing and shooting count. Hence, performance graphs provide a closer look into the agent's ability to perform certain actions.

By looking at the three graphs one gets a good idea of how the agent is doing. For instance, in the case of deciding network output activation function, the performance graphs are very similar, but the learning graph indicates that *linear* alternative is ascending and superior -hence doing better in shaping rewards and potentially giving more room for improvement.

6.3. Network complexity

The experiments involving the Neuroevolutionary algorithm used to evolve the agent Controller suggest that FS-NEAT does not have a positive impact on performance. According to [17] FS-NEAT results in similar performance as NEAT with less features *only* when the input features are (partially) redundant. In the cases where features cannot be simplified, the expected results are similar to the ones we obtained in Section 5.2.2, that is only reaching equivalent performance when the evolutionary process has given it enough time to incorporate as many features as possible. Table 4 shows approximate complexity of Controller networks evolved with each of the NE and action selection alternatives.

	Start complexity (Neurons-Links)	**End complexity (Neurons-Links)**
NEAT action selection	28-171	45-220
NEAT axis action	22-57	35-110
FS-NEAT action selection	28-18	55-130
FS-NEAT axis action	22-6	50-150

Table 4. Relative complexity of the networks evolved with different NE algorithms and action selection methods

The lower performance seen by both FS-NEAT alternatives is largely due to the under-use of the input features and confirms that the feature vector evolved is compact and non-redundant.

6.4. Policy search vs value function search

This work evaluates controllers that search the policy space (axis action, which maps a state with an action) and the value function space (action selection, which maps a state with values for each of the possible actions). As Section 5.2.2 results indicate, a search on policy space seems to be superior to that of a value function space. The intuition is that the network does not need to learn the value of each action on every state, just the action to perform, which would result in faster training.

Moreover, mapping states and single actions facilitates certain features to have an impact directly on actions or sub actions, which is difficult to have when predicting values for multiple actions. For instance, if a feature gives information about whether or not there is an enemy on the screen, a single link to the shooting neuron can determine whether to shoot or not; whereas if actions are not atomised, more links are needed if there are multiple shooting actions (as is the case in ViZDoom, with *shoot and forward*, *shoot and turn left, shoot and turn right* and *shoot*)

6.5. Benchmark

The final comparison with benchmark alternatives shows the EA strategy is viable and at least as good as the DQN in *pursuit and gather*, but much better in *health gathering* scenario. Although these results serve as an indicator of the goodness of EA strategy, the performance of DQN and the supervised CNN may not be optimal due to the training conditions imposed. However our DQN *health gathering* results are consistent with those

of other authors [11], where the learner struggles to perform on average but occasionally solves the level.

Contrary to other visual neuroevolutionary strategies that attempt to evolve large networks such as NEAT+Q [16] and Hyper-NEAT [36], the EA strategy has very affordable training[14] and execution times, which makes it a good candidate for real-time control of visual-only environments.

6.5.1. EA strategy behaviour

The best EA strategy can be said to solve both scenarios -pursuit and gather and health gathering. In *pursuit and* gather, often the agent stays alive for the duration of the test episode (around 35 seconds) killing enemies along the way to avoid being damaged.

In *health gathering*, the agent manages to stay alive an average of 86% of the total episode length (around 52 seconds out of the 60 possible), frequently reaching the 100% mark.

These two results are evidence of the intelligent behaviour displayed by the agent, as the baseline behaviour (idle) for both scenarios would have resulted in 14 seconds on pursuit and gather and 8 on health gathering -due to the constant poisoning introduced on the levels.

6.6. Limitations

Even though the results so far are very promising and the two scenarios can be considered solved by the EA strategy, there is still room for improvement. More complex scenarios, such as those including a variety of enemies, or weapons that cause delayed and splash damage, such as the rocket launcher, still pose a challenge for the strategy and remain unsolved.

Note that the results in this work have been averaged. The actual performance varies over repetitions. Both how the feature extractor is evolved and how the controller is trained seem to influence and leads to variability in the results.

14Moving most of the execution to a single GPU GeForce GTX 980, end-to-end training takes less than 10 hours on either scenario; the evaluation of the network takes milliseconds.

7. Conclusions and further work

7.1. Conclusions

This work presents a successful Evolutionary Strategy for the optimisation of Deep Neural Networks in developing a controller to solve complex simulated visual-only tasks. The strategy is completely unsupervised and outperforms proposed deep reinforcement learning alternatives such as Deep-Q-Network when playing various scenarios in the ViZDoom environment.

The approach demonstrates the use of Convolutional Neural Networks as feature extractors to reduce the complexity of dealing with raw-pixel data from RGB images. Moreover, the weights of the CNN are tuned using an evolutionary process that focuses on diversity of output. Although a Shannon index-based diversity gives positive results, using euclidean distances to measure diversity produces the best outcome.

The features extracted by the CNN serve as the input for a secondary neural network that is trained using NEAT to determine what action the agent should take. A search on the policy space -i.e. mapping states to single actions- yields better results than a search on the value function space -mapping a state with the values for all possible actions.

7.2. Further work

Although the results shown in this work are promising, there are areas that can benefit from further research:

- Looking at co-evolution and distributed methods to evolve feature detectors and Controllers [14] [40]
- Online training [15] and the challenges of rarely visited states
- Limitations on generalisation: problems with complex scenarios -more advanced weapons -splash and delayed damage-, variety of enemies -textures and behaviours-, difficult scenarios -mirror textures, variable illumination, etc.
- Methods to evolve and optimise the topology of the CNN

8. References

[1] A. L. Samuel, “Some Studies in Machine Learning Using the Game of Checkers,” *IBM J. Res. Dev.*, vol. 3, no. 3, pp. 210–229, 1959.

[2] J. Munoz, G. Gutierrez, and A. Sanchis, “Controller for TORCS created by imitation,” *CIG2009 - 2009 IEEE Symp. Comput. Intell. Games*, pp. 271–278, 2009.

[3] C. Athanasiadis, D. Galanopoulos, and A. Tefas, “Progressive neural network training for the Open Racing Car Simulator,” *2012 IEEE Conf. Comput. Intell. Games, CIG 2012*, pp. 116–123, 2012.

[4] L. Cardamone, P. L. Lanzi, D. Loiacono, and E. Onieva, “Advanced overtaking behaviors for blocking opponents in racing games using a fuzzy architecture,” *Expert Syst. Appl.*, vol. 40, no. 16, pp. 6447–6458, 2013.

[5] L. Cardamone, D. Loiacono, and P. L. Lanzi, “Applying cooperative coevolution to compete in the 2009 TORCS Endurance World Championship,” *2010 IEEE World Congr. Comput. Intell. WCCI 2010 - 2010 IEEE Congr. Evol. Comput. CEC 2010*, 2010.

[6] S. Ontanon, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss, “A survey of real-time strategy game AI research and competition in starcraft,” *IEEE Trans. Comput. Intell. AI Games*, vol. 5, no. 4, pp. 293–311, 2013.

[7] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, “Mastering the game of Go with deep neural networks and tree search,” *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, “Playing Atari with Deep Reinforcement Learning,” *arXiv:1312.5602*, pp. 1–9, 2013.

[9] K. O. Stanley and R. Miikkulainen, “Evolving Neural Networks through Augmenting Topologies,” *Int. Secur.*, vol. 6, no. 2, pp. 209–217, 2003.

[10] K. O. Stanley, D. B. D’Ambrosio, and J. Gauci, “A hypercube-based encoding for evolving large-scale neural networks.,” *Artif. Life*, vol. 15, no. 2, pp. 185–212, 2009.

[11] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaśkowski, “ViZDoom: A Doom-based AI Research Platform for Visual Reinforcement Learning,” *arXiv:1605.02097v1 [cs.LG]*, 2016.

[12] D. Rumelhart, G. Hinton, and R. Williams, “Learning representations by back-propagating errors,” *Nature*, vol. 323, pp. 533–536, 1986.

[13] S. Whiteson, “Evolutionary Computation for Reinforcement Learning,” *Reinf. Learn. State Art*, vol. 12, no. c, pp. 325–355, 2012.

[14] F. Gomez, J. Schmidhuber, and R. Miikkulainen, “Accelerated Neural Evolution through Cooperatively Coevolved Synapses,” *J. Mach. Learn. Res.*, vol. 9, pp. 937–965, 2008.

[15] K. O. Stanley, B. D. Bryant, and R. Miikkulainen, “Real-time neuroevolution in the NERO video game,” *IEEE Trans. Evol. Comput.*, vol. 9, no. 6, pp. 653–668, 2005.

[16] S. Whiteson and P. Stone, *Evolutionary Function Approximation for Reinforcement Learning*, vol. 7, no. AI05-320. 2006.

[17] S. Whiteson, P. Stone, K. O. Stanley, R. Miikkulainen, and N. Kohl, “Automatic Feature Selection in Neuroevolution,” *Proc. 2005 Conf. Genet. Evol. Comput. - GECCO ’05*, no. June, p. 1225, 2005.

[18] S. Risi and K. O. Stanley, “An Enhanced Hypercube-Based Encoding for Evolving the Placement, Density, and Connectivity of Neurons,” *Artif. Life*, vol. 18, no. 4, pp. 331–363, 2012.

[19] I. K. Fodor, “A survey of dimension reduction techniques,” *Center for Applied Scientific Computing Lawrence Livermore National Laboratory*, no. 1. pp. 1–18, 2002.

[20] H. Zhang, P. Mazumder, L. Ding, and K. Yang, “Dimensionality Reduction Using Genetic Algorithms Michael,” *IEEE Trans. Evol. Comput.*, vol. 4, no. 2, pp. 472–480, 2000.

[21] J. Koutník, J. Schmidhuber, and F. Gomez, “Evolving deep unsupervised convolutional networks for vision-based reinforcement learning,” *Proc. 2014 Conf. Genet. Evol. Comput. - GECCO ’14*, pp. 541–548, 2014.

[22] M. Parker and B. D. Bryant, “Neurovisual control in the Quake II Environment,” *IEEE Trans. Comput. Intell. AI Games*, vol. 4, no. August, pp. 44–54, 2012.

[23] M. Hausknecht, J. Lehman, R. Miikkulainen, and P. Stone, “A Neuroevolution Approach to General Atari Game Playing,” vol. 6, no. 4, pp. 1–18, 2013.

[24] D. a Pomerleau, “Alvinn: An autonomous land vehicle in a neural network,” *Adv. Neural Inf. Process. Syst. 1*, pp. 305–313, 1989.

[25] V. Mnih, K. Kavukcuoglu, D. Silver, A. a Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. 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,” *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[26] Y. Wei, W. Xia, M. Lin, J. Huang, B. Ni, J. Dong, Y. Zhao, and S. Yan, “HCP: A Flexible CNN Framework for Multi-label Image Classification,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 8828, no. XX, pp. 1–1, 2015.

[27] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” *Arxiv.Org*, vol. 7, no. 3, pp. 171–180, 2015.

[28] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2323, 1998.

[29] C. Nebauer, “Evaluation of convolutional neural networks for visual recognition.,” *IEEE Trans. Neural Netw.*, vol. 9, no. 4, pp. 685–696, 1998.

[30] S. Zhu, Z. Shi, C. Sun, and S. Shen, “Deep neural network based image annotation,” *Pattern Recognit. Lett.*, vol. 65, pp. 103–108, 2015.

[31] K. Zhang, Q. Liu, Y. Wu, and M.-H. Yang, “Robust Visual Tracking via Convolutional Networks,” *arXiv*, vol. 25, no. 4, pp. 1–18, 2015.

[32] M. E. Taylor, S. Whiteson, and P. Stone, “Comparing Evolutionary and Temporal Difference Methods in a Reinforcement Learning Domain Matthew,” *Proc. Genet. Evol. Comput. Conf.*, pp. 1321–1328, 2006.

[33] C. J. C. H. Watkins and P. Dayan, “Technical Note: Q-Learning,” *Mach. Learn.*, vol. 8, no. 3, pp. 279–292, 1992.

[34] M. Hardt, B. Recht, and Y. Singer, “Train faster, generalize better: Stability of stochastic gradient descent,” *srXiv:1509.01240*, pp. 1–24, 2015.

[35] D. E. Moriarty, D. E. Moriarty, A. Schultz, A. Schultz, J. Grefenstette, and J. Grefenstette, “Evolutionary algorithms for reinforcement learning,” *J. Artif. Intell. Res.*, vol. 11, no. 3, pp. 241–276, 1999.

[36] P. Verbancsics and J. Harguess, “Generative NeuroEvolution for Deep Learning,” *arXiv1312.5355 [cs]*, pp. 1–9, 2013.

[37] J. D. Lamos-Sweeney, “Deep Learning Using Genetic Algorithms,” *Rochester Inst. Technol.*, 2012.

[38] O. E. David, T. Aviv, and I. Greental, “Genetic Algorithms for Evolving Deep Neural Networks,” *Proc. Companion Publ. 2014 Annu. Conf. Genet. Evol. Comput.*, pp. 1451–1452, 2014.

[39] C. E. Shannon, “A mathematical theory of communication,” *Bell Syst. Tech. J.*, vol. 27, no. July 1928, pp. 379–423, 1948.

[40] A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. De Maria, V. Panneershelvam, M. Suleyman, C. Beattie, S. Petersen, S. Legg, V. Mnih, K. Kavukcuoglu, and D. Silver, “Massively Parallel Methods for Deep Reinforcement Learning,” *arXiv:1507.04296*, p. 14, 2015.

9. Appendix

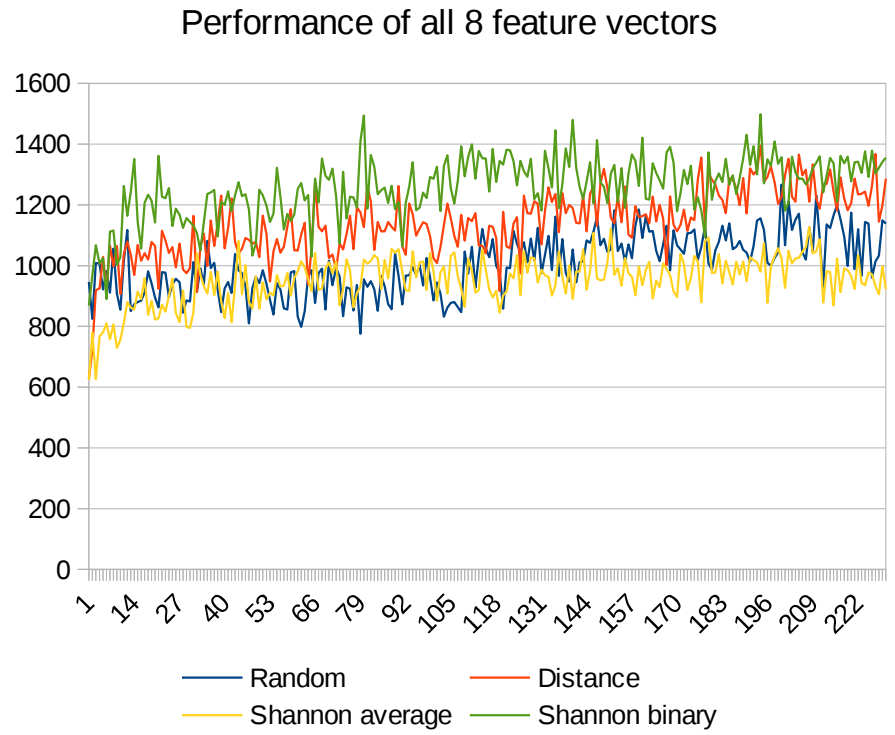

Figure 28. Total rewards for all fitness measurement alternatives with feature vector length 8

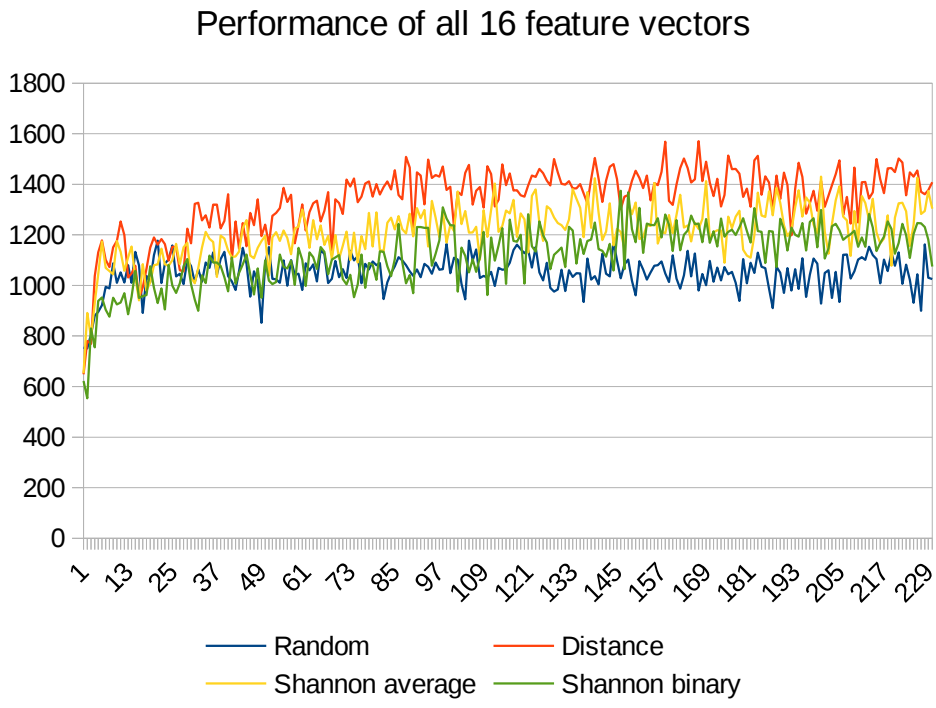

Figure 29. Total rewards for all fitness measurement alternatives with feature vector length 16

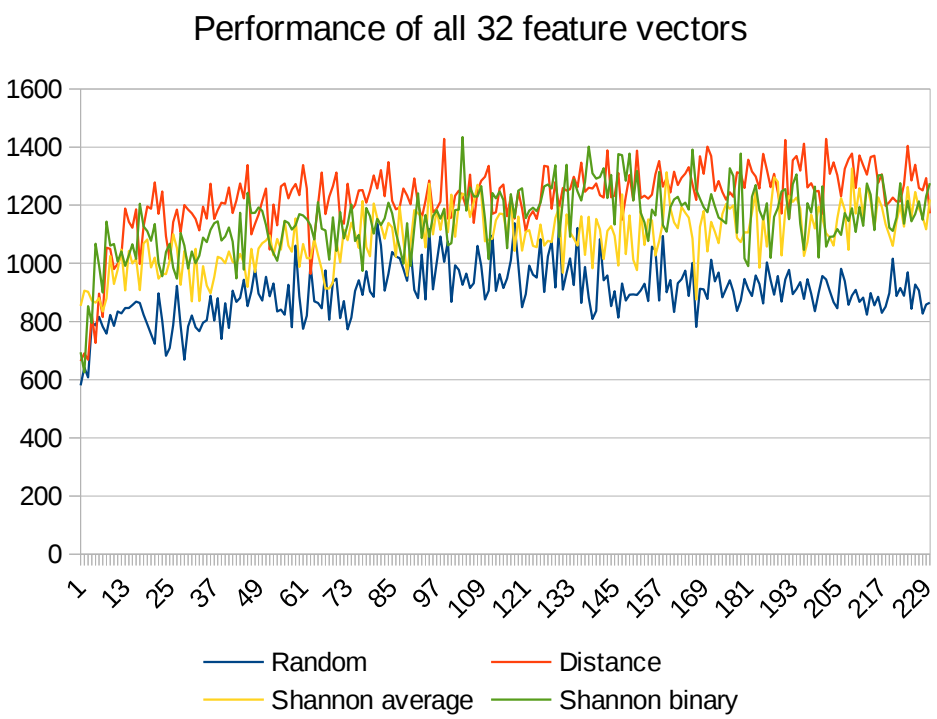

Figure 30. Total rewards for all fitness measurement alternatives with feature vector length 32

	SupervisedCNN	DQN	EA solution
Average	715.648	661.98	1793.674
St. error	15.6805991228	14.7623842207	23.7210227955

Table 5. Average total reward for Health and gather scenario on 100 trials

	SupervisedCNN	DQN	EA solution
Average	831.766	1554.438	1650.76
St. error	4.1623274909	11.486336088	13.1943128244

Table 6. Average total reward for Pursue and gather scenario on 100 trials