DEPARTMENT OF INFORMATICS
UNIVERSITY OF FRIBOURG (SWITZERLAND)

Extending the Applicability of Neuroevolution

THESIS

Presented to the Faculty of Sciences and Medicine of the University of Fribourg (Switzerland)
in consideration for the award of the academic grade of
Doctor of Philosophy in Computer Science

by

GIUSEPPE CUCCU

from

ITALY

Thesis No: 2101
UniPrint
2018

Accepted by the Faculty of Sciences and Medicine of the University of Fribourg (Switzerland) upon the recommendation of:

Prof. Dr. Ulrich Ultes-Nietsche, University of Fribourg (Switzerland), president of the jury.
Prof. Dr. Philippe Cudré-Mauroux, University of Fribourg (Switzerland), thesis supervisor.
Prof. Dr. Denis Lalanne, University of Fribourg (Switzerland), internal examiner.
Prof. Dr. Julian Togelius, New York University (NY, United States), external examiner.

Fribourg, September 19, 2018

Thesis supervisor | *Dean*

Prof. Dr. Philippe Cudré-Mauroux | Prof. Dr. Christian Bochet

Sa brucca lompi a funti fincas' a si sfundari.
Sa coa est sa prus' mala a scorgiari.

Chi è accanto a te con te c'è,
chi non è accanto a te non c'è.

To Philippe and Somayeh

Acknowledgements

The jug fetches water until it breaks.
The tail is the hardest to skin.

— Dicius (Sardinian proverbs)

Those who are next to you are with you,
those who are not near are not.

— Tormento, 1998

I suppose I should learn Lisp, but it seems so foreign.

— Paul Graham, 1983

Family first, always. My parents and brothers are my roots and pride. Yet here I would focus on acknowledging those who made this work possible.

To the superlative Machine Learning experts next to whom I had the privilege of growing over the years. In random order: Cassio, Kail, Sun Yi, Leonardo, Matt, Tom, Tobias, Daan, Klaus, Gianni, Julian, Somayeh, Juxi. You showed me the researcher I wanted to be. Even to Jan and everyone else around during those troubled times, regardless: credit is due, and I am thankful.

To Julian: such a big heart, your frame looks tight in comparison. To Tobias: I still find myself quoting your advice to younger students. To Cinzia: generations of researchers owe to your care – we remember.

To the *eXascale Infolab* as an institution, to each of its members and alumni, and to the Department of Informatics at the University of Fribourg as a whole: thanks for nurturing me back into a researcher. Special mention (chronologically) to William (pre-XI), Michael, Mourad, Alberto T., Julien and Alberto L.: you were there in the time of need, I am honored by your friendship.

To my supervisor, Prof. Dr. Philippe Cudré-Mauroux. Words fail here. Thanks for realizing a viable academic environment, and for teaching us by example the poise comported with being a gentleman.

To Somayeh, my *home*: closure at last! Onwards!

Fribourg, September 28, 2018

Abstract

In recent years, machine learning has achieved results which were considered unthinkable just a decade ago. From object recognition in images to beating humans at *Go*, new results are introduced at a steady rate. Most come from a single, recent subfield of machine learning, namely deep learning, which trains deep (i.e. many-layer) neural networks using variations of the backpropagation algorithm. Backpropagation itself comes with a set of strict limitations and requirements however, which are arguably not warranted in a majority of applications. Alternative methods exist that sidestep such requirements entirely. One of these, considered the state-of-the-art training technique until the advent of deep learning, is *neuroevolution*. Neuroevolution trains neural networks using black-box optimization algorithms known as evolutionary algorithms, able to directly *search* in the space of neural networks. Extending the applicability of neuroevolution would enable taking on classes of problems for which deep learning is not suited. This thesis studies and addresses some of the major issues currently limiting the applicability of neuroevolution, such as improving results on smaller networks, mitigating fitness stagnation, and scaling to large networks (i.e. higher dimensions). After an introduction to the key ideas and fundamental concepts in Chapters 1 and 2, Chapter 3 makes the case for smaller, shallow networks, which can achieve high performance in complex tasks if the training data is carefully pre-processed. This has application in a wide variety of real-world problems which produce too little data and/or of too low quality to meet the requirements of deep learning methods. Chapter 4 considers the problem of fitness stagnation through a novel restart strategy. Rather than terminating the run upon reaching convergence, the search is restarted on the most promising area of the space as derived from the history of the search so far. Chapters 5 and 6 take on increasingly complex tasks with high dimensional observations. The feature extraction is separated from the decision making: with the former delegated to an external component, smaller networks are devoted entirely to decision making, highlighting their performance. Finally Chapter 7 presents an evolutionary algorithm specifically designed for neuroevolution offering state-of-the-art performance. Its highly parallelizable implementation offers constant scaling over the size of the network, solely limited by the availability of parallel computational resources.

Keywords: Neuroevolution, Neural Networks, Evolutionary Algorithms, Continuous Control, Reinforcement Learning, Data-Driven Processes.

Résumé

Au cours des dernières années, l'apprentissage automatique a donné des résultats qui étaient considérés comme impensables il y a dix ans. De la reconnaissance d'objets en images jusqu'à battre les humains au jeu de *Go*, de nouveaux résultats sont introduits à un rythme soutenu. La plupart proviennent d'un seul sous-domaine récent de l'apprentissage automatique, à savoir le deep learning, qui forme des réseaux neuronaux profonds en utilisant des variations de l'algorithme de rétropropagration. La rétropagation elle-même est cependant accompagnée d'un ensemble de limitations et de stricts pré-requis, qui ne sont pas garantis dans la majorité des applications. Une méthode alternative, considérée comme l'état de l'art jusqu'à l'avènement de l'apprentissage profond, est la *neuroévolution*. La neuroévolution entraine des réseaux de neurones à l'aide d'algorithmes d'optimisation de boîtes noires connus sous le nom d'algorithmes évolutifs qui explorent l'espace des réseaux de neurones. L'extension de l'applicabilité de la neuroévolution permettrait de l'appliquer sur des classes de problèmes pour lesquels l'apprentissage profond n'est pas adapté. Cette thèse étudie et propose des solutions à certaines des principales questions qui limitent actuellement l'applicabilité de la neuroévolution. Après une introduction aux idées clés et aux concepts fondamentaux dans le Chapitres 1 et 2, le Chapitre 3 étudie les avantages de réseaux plus petits et peu profonds, qui peuvent s'appliquer à une grande variété de problèmes du monde réel qui produisent trop peu de données et/ou de qualité trop faible pour satisfaire les exigences des méthodes du deep learning. Le Chapitre 4 aborde le problème de la stagnation du fitness avec une nouvelle stratégie de redémarrage. Plutôt que terminer l'exécution après avoir atteint la convergence, la recherche est redémarrée dans la région la plus prometteuse de l'espace, comme il ressort de l'historique de la recherche jusqu'ici. Les Chapitres 5 et 6 proposent des solutions à des tâches de plus en plus complexes basées sur des données de grande dimension. L'extraction des descripteurs est déléguée à une composante externe, tandis qu'un plus petit réseau est entièrement dévolu à la prise de décision, qui met en évidence sa performance. Enfin, le Chapitre 7 présente un algorithme évolutif spécifiquement conçu pour la neuroévolution, offrant des performances de pointe. Son implémentation hautement parallélisable offre une mise à l'échelle constante de la taille du réseau, uniquement limitée par la disponibilité des ressources de calcul.

Mots clefs : Neuroévolution, réseaux neuronaux, algorithmes évolutifs, contrôle continu, apprentissage par renforcement, processus pilotés par les données.

Contents

1 Introduction

Machine Learning is the art of adapting generic solvers to address specific problems. In recent years neural networks [Rosenblatt, 1958] have taken the spotlight as flexible, broadly applicable solvers, typically trained by using variants of the *backpropagation* algorithm [Werbos, 1982, LeCun, 1985], as e.g. in deep learning [LeCun et al., 2015]. Backpropagation works by tuning the network weights for a fixed structure: given a target output, the weights are iteratively updated as to minimize the error between this and the network's actual outputs, refining the network from a random initialization to addressing a precise task.

The applicability of backpropagation-based methods however is limited by strict requirements. For example, an *error* needs to be available and differentiable on a per-input basis, as the algorithm is designed to learn over a large collection of *labeled* data (supervised learning).

An alternative training method can be found in *neuroevolution* [Yao, 1993, Floreano et al., 2008], which was considered the state of the art for many applications right until the advent of deep learning. Neuroevolution uses evolutionary algorithms [Fogel et al., 1966, Fraser et al., 1970, Fogel, 1998, Rechenberg, 1973] to *search* in the space of neural networks, a much more generic approach than backpropagation, unrestricted by the latter's requirements. Direction for improvement is provided by a *fitness function*, defined over networks rather than single data points, thus directly applicable in the absence of labeled data (reinforcement learning). Moreover the evolutionary algorithms' internal processes are inherently exploratory, making neuroevolution results resilient to variations of initialization conditions and hyperparameters.

So why was backpropagation selected for training networks in deep learning rather than neuroevolution? Two major reasons are: (i) the performance of state-of-the-art evolutionary algorithms scales *superlinearly* in the number of parameters (i.e. weights), with deep networks often having millions; and (ii) when successful, backpropagation typically performs better.

Many problems do indeed satisfy the requirements for backpropagation to be applicable. The first results in deep learning for example focused on object detection and image classification [Krizhevsky et al., 2012], where large collections of labeled images provide an ideal

training environment for backpropagation. When all conditions are satisfied, backpropagation is a simple and fast algorithm with great optimization performance.

A large majority of problems though does not accommodate such requirements. For example *continuous control* problems require an agent to dynamically interact with an environment by answering to stimuli (i.e. observations) with appropriate actions. This is usually done under a *reinforcement learning* paradigm, where information about correct actions for each sensor reading is typically unavailable, replaced by a *reward* signal often available only at the end of the run. Moreover, planning ahead usually requires *memory* components in the network, which can break the differentiability of the error.

This is typically addressed in deep learning by introducing network-based variations of *classical reinforcement learning* methods [Sutton and Barto, 1998]: though computationally expensive, these methods can achieve impressive performance on hard problems, provided their own requirements are satisfied (e.g. typically discrete observation space and action space) [Mnih et al., 2015, Silver et al., 2016]. Recent work though begins to question how much of that effort is actually a band-aid to the inherent limitations of the methods of choice, and how much is actually addressing task complexity instead [Cuccu et al., 2018].

The main value of neuroevolution is arguably not in trying to replace backpropagation and deep learning, but to provide an option to address tasks beyond its application. The overarching goal of my research is thereby defined as to extend the applicability of neuroevolution to a broader array of applications, by addressing the main limitations of neuroevolution itself as described in the following core research questions.

1.1 Research Questions

My research addresses the following questions:

1. What types of application are *beyond the applicability of deep learning* and what are competitive alternatives?

2. How can evolutionary algorithms overcome the problem of *fitness stagnation*, where all individuals become indistinguishable based on performance alone, which deprives the search for a direction of improvement?

3. How can neuroevolution tackle problems that require parsing of *high-dimensional data*, such as visual data?

4. Can neuroevolution scale to *high-complexity problems* using smaller, shallow networks, such as providing sophisticated control based on high-dimensional observations?

5. The complexity of the function approximated by the neural network is ultimately still bound by its size, which means that eventually large networks will still be the only option

left. Can neuroevolution scale to evolve networks of *arbitrary size*?

These questions will be addressed in the following chapters.

1.2 Finding Answers

The next chapters will follow my work over some of the main issues currently restricting the adoption of neuroevolution:

- *Chapter 2* proposes an eagle's eye perspective on learning paradigms, neural networks, backpropagation, neuroevolution, and a few techniques recurring in the following chapters.
- *Chapter 3* begins with a practical application where the data is of insufficient quality and quantity for deep learning; a careful pre-processing step however enables shallow networks to address the problem with state-of-the-art results (Question 1).
- *Chapter 4* tackles one of the most troublesome issues in evolutionary algorithms, namely fitness stagnation (Question 2). It opens the discussion towards explicit exploration techniques and intrinsic motivation.
- *Chapter 5* addresses the problem of extracting meaningful information from high-dimensional data (Question 3), via a two-step process that separates feature extraction from the (smaller) decision making network.
- *Chapter 6* takes the work in Chapter 5 one step further, by addressing a common deep learning testbed with a new observation encoder plus a very tiny network, two orders of magnitudes smaller than commonly found in the deep learning literature (Question 4).
- *Chapter 7* finally addresses the scalability issue of evolutionary algorithms with a new algorithm with linear performance scaling over networks size (Question 5). Together with its high parallelizability, it unlocks the potential to evolve networks of unprecedented size.
- *Chapter 8* concludes with a summary of the work, and provides an overlook on its future implications.

1.3 Published Papers

The work presented in this thesis has been published in the following papers:

- Chapter 3: *A Data-Driven Approach to Predict NOx-Emissions of Gas Turbines* [Cuccu et al., 2017]
- Chapter 4: *Novelty-Based Restarts for Evolution Strategies* [Cuccu et al., 2011a]

- Chapter 5: *Intrinsically Motivated Neuroevolution for Vision-Based Reinforcement Learning* [Cuccu et al., 2011b]
- Chapter 6: *Playing Atari with Six Neurons* [Cuccu et al., 2018]
- Chapter 7: *Block Diagonal Natural Evolution Strategies* [Cuccu and Gomez, 2012]

1.4 What this Thesis is Not About

All the work presented in this thesis uses smaller, shallow networks; the lack of results using *deep networks*, considered almost a given nowadays, makes for a notable absence. On one side, much of the reason can be reduced to neuroevolution being particularly apt at training smaller networks to the best of their abilities (see Chapter 2.4). On the other hand, the innovations presented in this thesis greatly boost both training and solution performance, to a point where shallow networks produce surprisingly good results. Chapter 6 for example presents competitive results on an application typically addressed with networks of hundreds of neurons, by using only six.

In order to comprehend the scope of the results proposed in the following chapters though, the first step is to lay a solid, clear foundations of what machine learning and deep learning can and cannot do.

2 Fundamentals and state of the art

Over the course of just a few years, Machine Learning (ML) has become ubiquitous. Public media hypes over new findings, the largest companies have dedicated teams, and even smaller businesses are starting to adopt increasingly advanced techniques. As always however such buildup should be taken with a grain of salt: the applicability of most ML techniques is restricted to precisely distinct sets of problems, and scaling to generic behavior or *intelligence* is a goal still firmly set in the future. Extending the applicability of ML techniques to new classes of problems is a step in this direction, but it first requires a firm understanding of how ML algorithms work and their restriction.

This chapter offers a high-level overview of the current state of (a small part of) machine learning, building the foundations to understand what works and why, and what challenges still lie ahead. Sections 2.5 and 2.6 will then delve deeper in the technical details of Natural Evolution Strategies and Novelty Search, two recurrent topics in the chapters to follow.

2.1 Machine Learning as Model Adaptation

Modern ML routinely addresses problems of high complexity, considered unthinkable just a few years past. Often this is treated as an upper bound, as if any task perceived as comparable or lower difficulty should now be considered as solved. Problem complexity however is highly variegated and difficult to grasp, where applications perceived as simple can be deceptively hard for a given ML technique. Consider for example how deep learning leverages methods which are fundamentally over 30 years old, but could not express their full potential until recent innovations in big data (e.g. large labeled datasets) and hardware performance (e.g. graphics processing units).

Machine learning works by adapting generic solvers to specific problems. This is true throughout the spectrum, from humble linear regression to the latest deep learning results [Friedman et al., 2001, Pearl, 2014, Ho, 1995, LeCun et al., 2015]. The common requirement is the availability of an *objective function* of sort, i.e. a mathematical insight pointing towards the behavior

of an ideal solution. Depending on the available objective function, three main learning paradigms are derived.

- *Supervised learning:* the objective is implicitly specified by a set of *labeled* examples of correct interactions (pairs of input plus expected output).
- *Unsupervised learning:* a similarity measure partitions the data into areas, which are associated to different interpretations (i.e. clustering).
- *Reinforcement learning:* candidate solutions are tested, and ranked based on their performance. This imposes a gradient of improving performance over the solutions' space.

A *solution* in this context is a specific and plausible way to address a problem, and can refer either to an ideal behavior, or an approximation of it from an instance of a solver. A *solver* is a generic, adaptable mathematical form which can represent (based on its parameterization) any number of solutions. Machine learning incrementally tailors generic solvers toward approximating the ideal solution, by following the objective function.

To understand what a mathematical representation of a solver/solution might be, consider the problem of correctly distinguishing whether or not a cat is present in a picture; or controlling a robot in a maze, alternating reading from range-finder sensors and sending voltage controls to its wheel engines (i.e. *continuous control*). Mathematical functions can be used to describe either interaction:

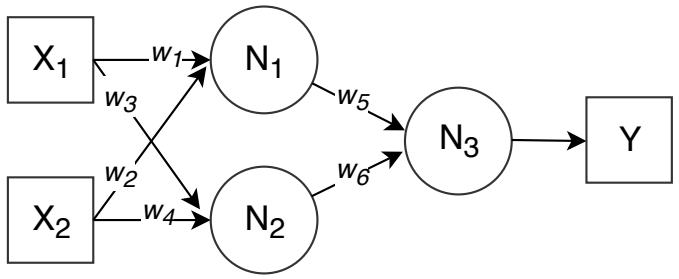
$$
\begin{aligned}
&\mathrm{sol}_{\mathrm{cat}} : \texttt{Images} \mapsto \{\textit{true}, \textit{false}\}, \texttt{Images} \in [0,255]^{\mathrm{size}} \subset \mathbb{N}^{\mathrm{size}} \\
&\mathrm{sol}_{\mathrm{bot}} : \texttt{Sensors} \mapsto \texttt{EnginesVolts}, \\
&\qquad \texttt{Sensors} \in \mathbb{R}^{\#\ \mathrm{sensors}}, \\
&\qquad \texttt{EnginesVolts} \in [\mathrm{minvolt}, \mathrm{maxvolt}]^{\#\ \mathrm{motors}} \subset \mathbb{R}^{\#\ \mathrm{motors}} \quad .
\end{aligned}
$$

Such equations describe the form of plausible solutions. The corresponding objective functions would then lead towards the ideal solutions, based on the problem and learning paradigm:

$$
\begin{aligned}
&\mathrm{obj}_{\mathrm{cat}} : \operatorname*{argmin}_{\theta} \left[\text{error between } f_\theta(\texttt{Image}) \text{ and correct classification}\right] \ , \\
&\mathrm{obj}_{\mathrm{bot}} : \operatorname*{argmin}_{\theta} \left[\text{time taken by a robot controlled by } f_\theta(\texttt{Sensors}) \text{ to escape the maze}\right] \ ,
\end{aligned}
$$

where f_θ is a *candidate solution*, defined by solver f and parameters θ.

The next step is to choose a parametrized solver, which can be adapted or *trained* following



(a) Graph

$$Y = Y_{N_3} = \sigma[w_5 Y_{N_1} + w_6 Y_{N_2}]$$
$$Y_{N_1} = \sigma(w_1 X_1 + w_2 X_2)$$
$$Y_{N_2} = \sigma(w_3 X_1 + w_4 X_2)$$
$$Y = \sigma[w_5 \sigma(w_1 X_1 + w_2 X_2) + w_6 \sigma(w_3 X_1 + w_4 X_2)]$$

(b) Equations

Figure 2.1 – **Neural network representations.** *(a)* The graph on the left depicts a three-neurons feed-forward neural network with two inputs on the inputs layer, two neurons in a single hidden layer, and one neuron in the output layer. The output of the network corresponds to the output of the latter. Neurons are drawn round, while squares represent numerical variables. *(b)* The underlying equations derived from the graph. This form exposes its nature of *parametrized function approximator*, reminiscent of Taylor series and Fourier transforms. Each neuron takes inputs based on its connections, scaling them based on the connection's weight. These are aggregated with a sum, then passed to an activation function $\sigma(\cdot)$. The neural network is a function uniquely identified by its parameters (here structure, activations and weights $\boldsymbol{w} = \{w_1, w_2, \ldots, w_6\}$). The output of the network is uniquely defined by the equation(s) of the neuron(s) in the output layer, computed on the network's inputs.

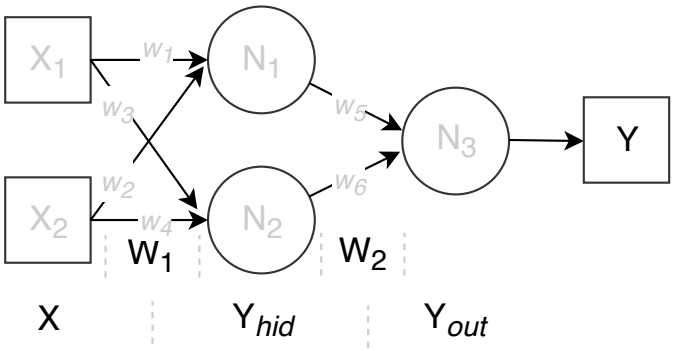
the objective function to approximate the desired solution. A common choice nowadays are *neural networks*, which are easy to reason about and scale in complexity.

2.2 Neural Networks

Neural Networks (NN; Rosenblatt [1958]) are generic function approximators [Hornik et al., 1989]. This means that their mathematical formulation is capable in principle to approximate any function to arbitrary precision. Particularly, the function complexity and/or precision of a network is only bound by its parameters (mostly structure and activation).

A neural network is typically depicted as a directed weighted graph, where the nodes have computational capability, while the links describe the direction of the computation (inputs and outputs). The nodes are most commonly *perceptrons* [Rosenblatt, 1958], also called *neurons* because originally inspired by the biological cells that carry electric signals in the animal nervous system.

From a functional perspective, neurons have a set of input connections and a single output. The output is computed by weighing the inputs based on the correspondent connection (link), summing up the results, then passing the total to an *activation function*. The latter could be a simple identity function (i.e. return the sum of weighted inputs itself [Mikolov et al., 2013]), a complex nonlinear equation [LeCun et al., 1998], or a modern version with optimized performance [Nair and Hinton, 2010]. A detailed description of the transition from graph to



(a) Graph

$$Y = Y_{\text{out}} = \text{map}[\sigma, W_2 Y_{\text{hid}}]$$
$$Y_{\text{hid}} = \text{map}[\sigma, W_1 X]$$
$$Y = \text{map}[\sigma, W_2\ \text{map}[\sigma, W_1 X]]$$

(b) Equations

Figure 2.2 – **Neural network and linear algebra.** *(a)* The same network seen in Figure 2.1(a), this time highlighting the layers and weight matrices. Notice how the input layer X has no computational capacity, and the Y is just the output the output layer. *(b)* Linear algebra form of the neural network. This form simplifies the implementation, particularly with the goal of leveraging acceleration from specialized hardware (e.g. *GPUs*). The *map* operator highlights how the activation function σ needs to be called independently on each sum of weighted inputs. Layers have common inputs and present a common output: the output of the hidden layer can be seen as a system of two equations in X. The output layer could in principle have arbitrary size, representing a system of equations in Y_{hid}. By leveraging *function composition*, the output equation Y raises in complexity quickly with the number of layers.

mathematical representation can be found in Figure 2.1.

Neurons are typically arranged in *layers*, i.e. groups of neurons that share common inputs and present their outputs together (similarly to a system of equations in the same variables). The output of each neuron in a layer can be computed independently (and possibly in parallel) from the others. Neurons on the next layer however will depend on the availability of *all* the outputs of the previous layer, making the network activation inherently sequential. This simple architecture is called *feed-forward*. Figure 2.2 extends the correspondence between graph and equations with the introduction of a linear algebra notation, which leverages this layered construction.

Larger networks can in principle approximate more complex functions, as more terms are added to the corresponding equation. In practice though this only raises the complexity upper bound, as a large network can still have minimal complexity: setting all weights to zeros for example will always yield a constant function. Training large networks corresponds to updating more parameters, and hence requires proportionally larger amounts of data. This can sometimes limit the applicability of large networks to task generating a sufficiently large dataset. Data quantity requirements can be mitigated by raising the data quality, e.g. by reducing artifacts such as noise, missing values and redundancy. Processing techniques such as cleaning, feature extraction and compression often enable addressing complex problems through simpler networks, as will be further discussed in the next chapters.

Distributing the neurons over multiple layers typically increases the network's functional complexity faster than expanding a flatter structure, by means of leveraging *function composition* [LeCun et al., 2015]. More complex networks can approximate more complex functions, hence the common choice of *deep networks* (i.e. with many layers) to address sophisticated applications. Other ways to raise complexity include the use of different input patterns (i.e. *convolutional* networks [Hubel and Wiesel, 1968, LeCun et al., 2015]), more complex connective patterns (e.g. *recurrent* networks [Hopfield, 1982]), or nodes with advanced computational capability (e.g. LSTM [Hochreiter, 1991]). Out of these, recurrent neural networks will see the most application in the next chapters, as they are particularly apt in tasks requiring simple memory capabilities, such as stream processing and continuous control. The reason for their name is that its neurons (or more typically, whole layers) have feedback loop connections: upon subsequent activations, the previous output of the network is fed back into the neuron as an input to decide the next activation.

2.3 Backpropagation and Deep Learning

Training a neural network is the process of iteratively adapting the network's parameters to improve its approximation of an ideal solution, through optimization of the objective function. For neural networks, this role traditionally belongs to the *backpropagation* algorithm [Werbos, 1982, LeCun, 1985, Parker, 1985], a single-agent gradient descent technique which works based on *error assignment*. This means that only a single network instance (i.e. parametrization) is considered and updated at any point in time.

The network is activated on an input, producing an output (*forward pass*); this is compared with the corresponding expected target, their difference constituting an error which needs to be minimized. This is done by computing the derivative of the error over the network's parameters (i.e. weights) to produce a gradient. The weights of the last layer of the network are then updated based on how much they contributed to such an error (i.e. based on their current value). The algorithm proceeds then *backwards* through the network's layers (hence the name of the algorithm), distributing the error proportionally through each layer and updating each parameter in turn. This process is repeated (possibly multiple times) for each training point available, until a termination criterion is met (e.g. arbitrary precision). A full derivation is beyond the scope of this thesis but can be found in LeCun et al. [2015].

The backpropagation algorithm thus poses a tight set of requirements for its applicability:

- Availability of per-input expected targets. The algorithm needs to compute an error on a single activation: this requires the availability of examples of correct outputs (*labeled data*), restricting its application to supervised learning.
- The error function should be differentiable over the network's parameters. Single-agent approaches take an irreversible step in the dark with each update: if they were to cross over e.g. a discontinuity in the error function, they could be unable to get back on track

or otherwise recover.

- Large quantities of data. As an error is propagated through a network with many layers, the contribution of each parameter is increasingly smaller, which brings to correspondingly fading updates (*vanishing gradient*). This needs to be balanced by increasing the number of training iterations, which in turn requires additional data.

- Precise initialization and setup. Single-agent algorithms are limited in their exploratory capabilities: the quality of the final solution depends significantly on network initialization and hyperparameters.

- Sequential processing power. The last years have seen the flattening of Moore's law, with increasing hardware performance being linked to parallel processing rather than ramping speeds. Since each layer cannot be updated until the error has been propagated to it through all the precedent layers, backpropagation is inherently sequential, and thus less suited to gain advantage from increases in parallel capabilities.

Its success in deep learning is of course well deserved: on problems meeting these conditions, backpropagation delivers complex, highly refined solutions. The previous cat image classifier example can be addressed with a large feed-forward convolutional network, trained using backpropagation with supervised learning, thanks to the modern availability of large collections of (labeled) cat images. The quality of results in computer vision and object recognition is but increasing year over year [Krizhevsky et al., 2012]. Other applications however may not be so accommodating. In the example of the maze navigator, building a dataset of sensor readings with corresponding correct motor voltages is less than obvious. And there is no direct correspondence between a sensory observation and a correct motor-voltage action: planning and *memory* play a fundamental role, possibly breaking the differentiability of the error signal. As a result, the standard approach is to apply variants of classical reinforcement learning algorithms (learning the *value function*) rather than directly learning neural network agents (*policies)*.

Researchers have since explored alternative training options than backpropagation. In particular *neuroevolution* has been considered the state of the art in many applications in the years directly prior to the advent of deep learning, as it is capable of direct search in the space of neural network controllers.

2.4 Neuroevolution

Neuroevolution (NE; Floreano et al. [2008], Yao [1999], Igel [2003], Risi and Togelius [2017]) trains neural networks by *searching* in the space of network parameters using Evolutionary Algorithms (EAs; Fogel et al. [1966], Fraser et al. [1970], Fogel [1998], Rechenberg [1973]). EAs are multi-agent algorithms: rather than updating a single solution, they maintain a *population* of candidate solutions over the space of network parameters (*individuals*). This is traditionally

fixed in size, though work has been done towards varying the number of individuals throughout the run [Vanneschi and Cuccu, 2009a,b, 2011]. Each individual corresponds to a *genotype*, a vector of parameters which in turn uniquely represents a *phenotype* such as e.g. a neural network.

The *genotype to phenotype* (G2P) function builds the phenotype from the genotype's parameters. In *direct encoding* neuroevolution, each parameter is used as a network's weight. Alternatively, in *indirect encoding* the genotype values are interpreted using G2P functions of arbitrary complexity. For example [Koutník et al., 2013a,b] evolves genotypes which are interpreted as coefficients of a compressed representation, which is in turn de-compressed into a much larger phenotype network. Notably [Stanley et al., 2009] uses the genotype to build a (smaller) intermediate network, which in turn is used to generate the parameters for the larger phenotype network.

The phenotype thus represents a plausible solution, which is scored using the *fitness* function. This maps networks to scores based on performance on the task, such as e.g. the proportion of correctly classified cat pictures in a dataset. At the same time, the fitness can also be derived from the time it takes the maze navigator to reach the exit, making neuroevolution suited to different learning paradigms. Scoring the population builds a *Monte Carlo estimate* [Eckhardt, 1987] of the fitness function over the (network) parameters space, providing the search with a direction (gradient) for improvement.

The population is then iteratively improved by replacing low-performance individuals with new ones (*offspring*) which are created from high-performance individuals (*parents*). *Genetic Algorithms* (GA; Fraser et al. [1970], Koza [1992]) for example partition the genotypes of two parents at a random position, then swap halves between the two (*crossover*). Values in the genotype are also stochastically altered based on a minor chance (*mutation*). A large number of variations for either operator are available in the literature. *Evolution Strategies* (ES; Rechenberg [1973]) on the other hand constructs the offspring based only an adaptive mutation on a selected parent. This can be interpreted as generating a statistical sample on the random variable represented by the parent, with the variance determined by the mutation parameter. Some state-of-the-art ES expand on such an idea by keep the population *implicit* and maintain instead a *probability distribution* over the parameters space.

2.5 Natural Evolution Strategies

Natural Evolution Strategies (NES; Wierstra et al. [2014a, 2008], Yi et al. [2009], Sun et al. [2009], Glasmachers et al. [2010], Schaul et al. [2011]) is a state-of-the-art family of ES with fast adaptation, invariant to local properties of the search space. Their applicability as randomized black-box continuous algorithms enable them to target unconstrained continuous problems, making them well suited for the optimization of the weights of a neural network. Algorithms in the NES family maintain a probability distribution over the space of parameters, which is adapted following the *natural gradient* as obtained by rescaling the vanilla gradient by the

Fischer Information Matrix [Amari and Douglas, 1998].

The next sections will derive the update functions for the distribution parameters, both in their generic forms and with regard to algorithms based on Gaussian distributions.

2.5.1 NES update functions

Algorithms of the Natural Evolution Strategies family maintain a distribution $\mathscr{D}$ over the search space parameterized by θ, and iteratively update it to maximize the expected fitness $f : \mathbb{R}^d \rightarrow \mathbb{R}$ of its samples. The fitness gradient is approximated through Monte Carlo estimation by sampling $\lambda \in \mathbb{N}$ *individuals* $\mathbf{z}_k \sim \mathscr{D}$, $k \in \{1, \ldots, \lambda\}$ from the distribution.

Given the distribution density $p(\mathbf{z}|\theta)$, the expected fitness under the search distribution is:

$$J(\theta) \;=\; \mathbb{E}_\theta[f(\mathbf{z})] \;=\; \int f(\mathbf{z})\; p(\mathbf{z}|\theta)\; d\mathbf{z} \;\; .$$

The fitness gradient over θ thus becomes:

$$\nabla_\theta J(\theta) \;=\; \nabla_\theta \int f(\mathbf{z})\; p(\mathbf{z}|\theta)\; d\mathbf{z} \;=\; \mathbb{E}_\theta \left[f(\mathbf{z})\; \nabla_\theta \log\big(p(\mathbf{z}|\theta)\big) \right]$$

(see [Wierstra et al., 2008] for the full derivation), which is approximated through Monte Carlo estimate as:

$$\nabla_\theta J(\theta) \;\approx\; \frac{1}{\lambda} \sum_{k=1}^{\lambda} f(\mathbf{z}_k)\; \nabla_\theta \log\big(p(\mathbf{z}_k|\theta)\big) \;\; . \tag{2.1}$$

The distinctive trait of NES is to rescale this *vanilla* gradient by the Fischer Information Matrix $\mathscr{F} = \mathbb{E}\left[\nabla_\theta \log\big(p\,(\mathbf{z}|\theta)\big)\, \nabla_\theta \log\big(p\,(\mathbf{z}|\theta)\big)^\top \right]$, a measure of the search confidence, to obtain the *natural gradient* [Amari and Douglas, 1998]:

$$\widetilde{\nabla}_\theta J \;=\; \mathscr{F}^{-1} \nabla_\theta J(\theta) \;\; ,$$

leading to the following NES update equation (with η learning rate):

$$\theta \leftarrow \theta - \eta \widetilde{\nabla}_\theta J = \theta - \eta \mathscr{F}^{-1} \nabla_\theta J(\theta) \ .$$

A generation of NES is thus constituted of:

1. Sampling a population of individuals from the search distribution
2. Evaluate them all based on the fitness function
3. Compute the approximated fitness gradient through Monte Carlo estimation
4. Rescale this vanilla gradient into the natural gradient
5. Update the search distributions parameters in the direction of expected fitness improvement

For the iterative loop see Algorithm 1.

Algorithm 1 Basic NES loop

Initialize:
  $\theta \leftarrow (\boldsymbol{\mu}, \Sigma)$
while not solved **do**
  for $i \leftarrow 1 \dots \lambda$ **do**
    $\mathbf{z}^i \leftarrow \{\mathbf{z}^i \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)\}$ ▷ Individual from distribution samples
    $\text{fit}^i \leftarrow f(\mathbf{z}^i)$ ▷ Compute the fitness of the individuals
  $\theta \leftarrow \text{NESUPDATE}(\theta, \text{fit}, \mathbf{z})$

In addition *fitness shaping* [Wierstra et al., 2008] makes the algorithm invariant under monotonic (i.e. *rank-preserving*) transformations, including scaling. This enables the algorithm to automatically adapt to the scale of the problem, albeit unknown. The individuals are sorted by fitness, and their contribution to the update is scaled based on their rank using coefficients called *utilities* $u_k \in \mathbb{R}$, $k \in \{1, \dots, \lambda\}$, typically:

$$\nabla_\theta J = \sum_{k=1}^{\lambda} u_k \cdot \nabla_{(\theta)} \log\big(p(\mathbf{z}_{k:\lambda} \,|\, \theta)\big) \ . \tag{2.2}$$

Utilities for a given population size most commonly sum to one and have zero mean, thus providing a normalized step independent from the magnitude of the actual elements of the individuals or their fitness. Typically the few best performing individuals are individually used as *attractors*, with utilities higher in value and with positive sign, with the least performing ones generating a generic *repulsor*, with many negative utilities with small absolute value.

2.5.2 Gaussian distributions

The main variants of NES use multivariate Gaussian search distributions $\mathscr{D} = \mathscr{N}(\mathbf{z}|\theta)$, parametrized by $\theta = \langle \boldsymbol{\mu}, \Sigma \rangle$, with the method adapting the full covariance matrix (XNES; Glasmachers et al. [2010]) behaving somewhat similarly to the widely adopted Covariance Matrix Adaptation Evolution Strategy (CMA-ES; Hansen and Ostermeier [2001]). Other versions trade convergence speed and computational requirements by restricting the form of the covariance matrix.

Separable NES (SNES; Schaul et al. [2011]) expects independent parameters and thus restricts the covariance to a diagonal matrix. This offers a trade-off between convergence speed and execution speed: the computational cost per-generation is $\mathscr{O}(k^3)$ for XNES in the number of parameters k – but only $\mathscr{O}(k)$ for SNES. A rule of thumb is to apply XNES if at all possible, while higher-dimensional problems beyond the applicability of XNES can still be tackled by SNES. Chapter 7 will present Block Diagonal NES [Cuccu and Gomez, 2012], which generalizes both XNES and SNES allowing for a finer control over this trade-off.

At each generation, the current distribution is sampled to produce a population of $\lambda \in \mathbb{N}$ individuals

$$\mathbf{z}_i \sim \pi(\mathbf{z}|\theta),\ i \in \{1,\dots,\lambda\} \ .$$

To derive the update equations for $\boldsymbol{\mu}$ and Σ [Wierstra et al., 2008], the latter needs first to be decomposed in two factors $\Sigma = \mathbf{A}^\top \mathbf{A}$ to leverage local exponential coordinates [Glasmachers et al., 2010]. This is done through Cholesky decomposition in XNES, while SNES simplifies this greatly with Σ being a diagonal matrix.

The gradient components based on standard normal samples $\mathbf{s}_i \sim \mathscr{N}(0,\mathbb{I})$ moved to the distribution as $\mathbf{z}_i = \boldsymbol{\mu} + \Sigma\mathbf{A}^\top \mathbf{s}_k$ become then:

$$\begin{aligned}
\tilde{\nabla}_{\boldsymbol{\mu}} J &= \sum_{i=1}^{\lambda} f(\mathbf{z}_i) \cdot \mathbf{s}_i \\
\tilde{\nabla}_{\mathbf{M}} J &= \sum_{i=1}^{\lambda} f(\mathbf{z}_i) \cdot (\mathbf{s}_i \mathbf{s}_i^\top - \mathbb{I}) \\
\tilde{\nabla}_{\Sigma} J &= \mathrm{tr}(\tilde{\nabla}_{\mathbf{M}} J)/k \\
\tilde{\nabla}_{\mathbf{A}} J &= \tilde{\nabla}_{\mathbf{M}} J - \tilde{\nabla}_{\Sigma} J \cdot \mathbb{I} \ .
\end{aligned}$$

The final update equations for the distribution's parameters (corresponding to NESUPDATE in Algorithm 1) are:

$$
\begin{aligned}
\boldsymbol{\mu} &\leftarrow \boldsymbol{\mu} + \eta_{\boldsymbol{\mu}} \cdot \Sigma\mathbf{A} \cdot \tilde{\nabla}_{\boldsymbol{\mu}} J \\
\Sigma &\leftarrow \Sigma \cdot \exp(\eta_{\Sigma}/2 \cdot \tilde{\nabla}_{\Sigma} J) \\
\mathbf{A} &\leftarrow \mathbf{A} \cdot \exp(\eta_{\mathbf{A}}/2 \cdot \tilde{\nabla}_{\mathbf{A}} J)
\end{aligned} \quad ,
$$

with $\eta_{\boldsymbol{\mu}}$, η_{Σ} and $\eta_{\mathbf{A}}$ learning rates (further details in [Glasmachers et al., 2010]).

A generation of SNES thus consists of building a population by sampling the current distribution, approximate the fitness gradient through Monte Carlo estimation, convert it into the natural gradient, and consequently update the distribution parameters. An open-source implementation can be found at my GitHub repository[1].

2.6 Novelty Search

Novelty Search (NS; Lehman and Stanley [2008, 2010]) proposes an alternative take on EAs and particularly on NE: to discard the classic extrinsic fitness in favor of an intrinsic *novelty signal*. Individuals are hence solely scored based on how "novel" they are with respect to individuals generated during the search so far, regardless of their performance on the task.

Two components are key to this method: (i) a *novelty distance*, which computes an arbitrary similarity between two individuals; and (ii) a *novelty archive*, a set of individuals which have been found to be sufficiently novel (w.r.t a *novelty threshold*) in the search so far. Adding an individual to the archive though immediately reduces the novelty of all similar ones: novelty is thus a *dynamic signal*, which drives the search towards areas of the search space which are known but least explored at any given moment.

The novelty of an individual (*novelty score*) is typically calculated as the average novelty distance over the k nearest neighbors, as taken from both the archive and the current population.

$$
\text{nov}(x_i) = \frac{1}{k} \sum_{j=1}^{k} \text{dist}(x_i, x_{i:j}) \quad , \tag{2.3}
$$

where $x_{i:j}$ is the j-th nearest neighbor of x_i w.r.t. the novelty distance $\text{dist}(\cdot,\cdot)$.

In order to achieve a high novelty score an individual needs to be relatively unique with respect to the features considered by the novelty distance. These could be as simple as its genotype values, while applications such as *continuous control* obtaining improved performance by using the individual's *behavior*, defined as an history of their interaction with the task's environment [Lehman and Stanley, 2010]. While previous work in EAs mostly focuses on *genotypical*

[1] `https://github.com/giuse/machine_learning_workbench/`

diversity [Glover and Laguna, 1998, Mahfoud, 1995], with its focus on behavioral diversity Novelty Search does not distinguish between different networks with similar performance, aiming at *novel behaviors* to emerge instead. To build on the previous example of the robotic maze navigator, a behavior could be defined as an individual's entire observation-action sequence in a run, with the novelty distance being the *edit distance* [Navarro, 2001] between two such sequences between two sequence. Another example could be represented by the coordinates of the final resting position in the maze at the end of the run, with novelty distance being derived from the *Euclidean distance* from a reference such as the start or goal.

The choice of a novelty distance greatly influences the outcome of the search. Let us expand on the example using end-of-run positions in the maze exploration with Euclidean novelty: during early generations, controllers exhibit simpler behaviors, and can all be expected to terminate not far from their starting position. An individual capable of terminating a bit further away will be considered novel: as such it will produce offspring, which could in principle be expected of reaching roughly the same area. These solutions will be added to the archive, saturation that area over time (dependent on the novelty threshold). At this point individuals able to reach that same area will not be novel anymore: the search will be directed towards exploring new directions.

This choice of behavior and novelty distance thus drives the search towards generating controllers exploring the maze at increasing resolution, with the dynamic nature of the novelty signal providing *scale invariance*. Initially the archive tends to saturate the space at a constant resolution (depending on the novelty threshold). As the coverage becomes uniform though, areas of the space in between archived individuals will become increasingly viable as relatively novel, naturally increasing the resolution of the search over time. Eventually the exit of the maze is going to become reachable by the increasingly specialized controllers, leading to solving the task without ever consulting the fitness function.

This makes Novelty Search a strong method for *intrinsic motivation*, i.e. where the algorithm poses an objective itself rather then relying on an extrinsic one being provided from the user. Further applications will be explored in Chapters 4 and 5. But first, the next chapter arguments whether methods other than deep learning should or should not be still considered viable at all.

3 Addressing Complex Problems with Shallow Networks

As introduced in Chapter 1, deep learning can train neural networks to address highly complex problems, as long as the training data satisfies its requirements. Particularly in regard to data availability, these algorithms require relatively large data sets to correctly train the network. Many real-world applications though do not offer the opportunity to collection sufficient data, and the data collected is often riddled with noise, missing values and redundancy. An extreme example can be found in gas turbine engines for large-scale power generation. (i) Operating under limit conditions of temperature, pressure and vibration generates high quantity of sensor noise. (ii) Being kept at optimal (constant) regimes for days at a time introduces high redundancy in the data. (iii) While reading the sensors every 5 minutes is sufficient for practical applications, this corresponds to generating a mere 100′000 data points per year, insufficient for most deep learning applications.

This chapter addresses the task of predicting the state of modern heavy-duty gas turbines for large-scale power generation, with the goal of enabling informed decisions on their operation and maintenance. The emissions behavior however is coupled to a multitude of operating parameters and to the state and aging of the engine, making the underlying mechanisms very complex to model through physical, first-order approaches. Machine learning methods requiring high quantity of data are equally inapplicable, as the data available is insufficient – by several orders of magnitude. An alternative is presented in a custom data-cleaning pipeline, which considerably reduces defects in the data, in turn enabling simpler machine learning techniques to accurately model the engine's emissions using the little data available.

3.1 Introduction

Many modern thermal power plants use gas turbines to generate electricity (Figure 3.1; Güthe et al. [2008]). These (i) compress ambient air to elevated pressures, (ii) add the (gaseous) fuel inside a combustion chamber, (iii) release the chemical energy of the mixture through combustion, and (iv) expand the hot gas through a multi-stage turbine. Such a process converts the heat of the combustion into mechanical energy, which is further converted into

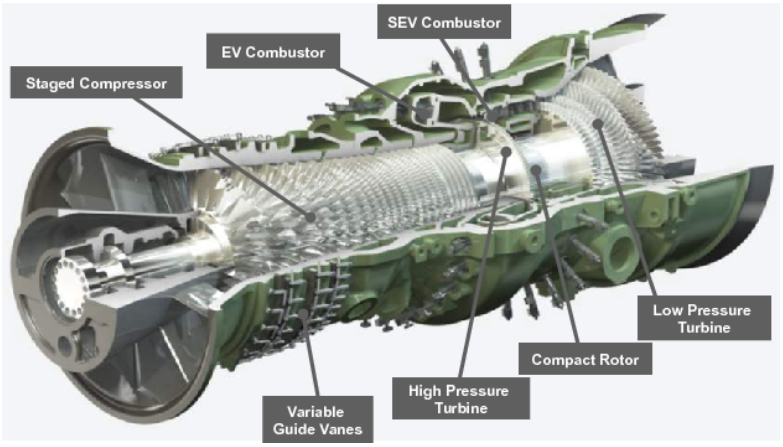


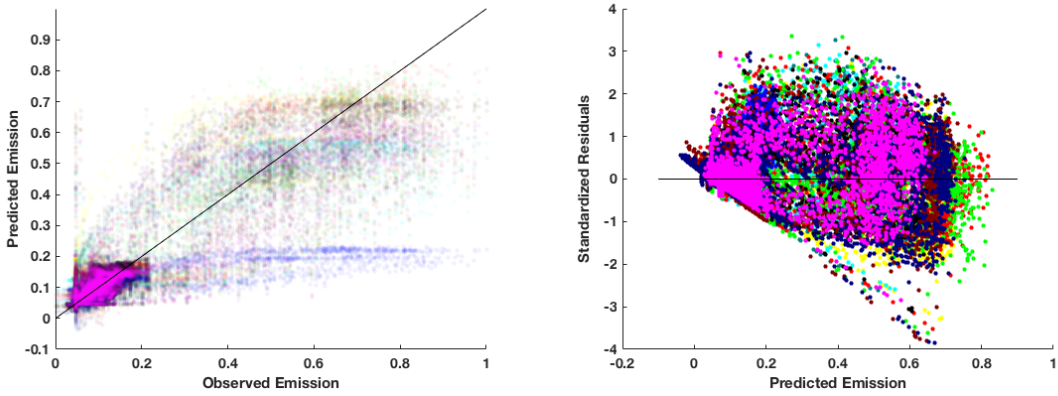
Figure 3.1 – **Architecture and components** of a *GT24/GT26* gas turbine. The staged compressor efficiently compresses inlet air into high pressure. The first combustor (EV) injects fuel and ignites the mix for the first burn. A second combustor (SEV) injects more fuel and ignites a second burn, which improves efficiency and lowers exhaust gas emissions such as NOx. Finally, the exhaust pushes the turbine into rotation, producing power through an external generator connected via a shaft.

electricity by the generator. In a *combined-cycle* arrangement, the residual heat of the exhaust gas is also recovered by a steam cycle for further power generation.

Heavy-duty gas turbines are capable of running reliably for extended periods of time requiring little to no maintenance. Over time however some engine parts may degrade due to the exposure to thermo-mechanical stress, thus negatively affect the emission level. This enforces maintenance if the emissions surpass the legal permits. Such maintenance shutdowns are costly, both due to the operative effort and because of production losses during the intervention, and as such are kept to a minimum. Advanced planning and effective decision support may thereby help reducing the associated costs.

During maintenance, the engine operation set-points are adjusted to a configuration featuring lower emissions without negatively affecting output and efficiency that respect the operative envelope of the gas turbine. A model able to accurately predict engine emissions would enable both to optimally plan maintenance timing, and to prescribe an appropriate adjustment of operation set-points to minimize outage duration and frequency.

Compared to the traditional approaches discussed in Section 3.1.2, this chapter explores the feasibility of accurately modeling gas turbine NOx emissions using machine learning, based on historical data collected from long-term engine operation. Sections 3.1.3 and 3.1.4 highlights the challenges encountered and scientific contributions. Section 3.2 introduces the proposed system, with Section 3.3 detailing the most successful configurations. Section 3.3.2 presents the best results, achieving a precision on par with sensor-level data. Finally, Section 3.4 offers a final discussion.



(a) Predicted vs. observed values

(b) Prediction residuals vs. predicted values

Figure 3.2 – **Performance on raw (unprepared) data.** Results from one of the best-performing algorithms (ν-SVR) on unprepared (i.e. original, unfiltered) data. The training finds no gradient for improvement: the resulting models have no success on emissions prediction. The predicted vs. observed graph (a) was expected to show a dense, tight diagonal, as each prediction corresponds to its observation. The scattered cloud of points corresponds to the algorithm being incapable of narrowing down the correspondence. The residuals vs. predicted graph (b) further highlights this by showing a broad scatter with distinct vertical patterns on its right side.

3.1.1 Engine Description

Heavy-duty gas turbines are high-performance engines at the core of many modern combined-cycle power plants. The gas turbine topping cycle consists in a thermodynamic Brayton cycle where the working fluid (air) is compressed, heated by the combustion of a (usually gaseous) fuel, and expanded in a turbine to convert the generated heat into mechanical power. The high-temperature exhaust heat is further converted in a bottoming steam Rankine cycle to provide additional power from the residual heat. A generator set transforms the mechanical power at the turbine shaft into electricity, which is connected to the electricity grid.

The engines considered in this work belong to the General Electrics *GT24* / *GT26* family and provide a rated output of above 400 MW in combined-cycle operation. Displayed in Figure 3.1, their particularity consists in a sequential combustion architecture, where two subsequent combustion chambers are separated by a high pressure turbine stage. The combustion chambers are of annular design, and the *lean premix technology* ensures that high firing temperatures (leading to high cycle efficiency) can be reached at very low nitrogen oxide (NOx) emission levels. A more detailed description of this engine type can be found in [Güthe et al., 2008]. The work presented in this chapter specifically refers to NOx prediction.

3.1.2 Related Work

Continuous Monitoring and Diagnostics (M&D) of engine performance and health is nowadays standard practice to ensure proper and reliable operation of power plants. Most traditional M&D relies on deterministic systems based on physical rules [Therkorn, 2005]. This comprises both understanding long-term performance trends and detecting anomalies in order to prevent component failure and avoid costly downtime and repair.

Physical approaches are well suited for modeling degradation and failure mechanisms that are known and can be identified through a limited number of measurements. However, they are more difficult to apply to complex degradation modes and evolving engine characteristics where no pattern is easily discernible at first glance.

The application of performance and emission degradation has been addressed in a series of works based on physical modeling approaches. [Rudolf et al., 2015] have applied a thermodynamic model of the engine for reconciliation of the commercial operation data to obtain physically consistent information for a precise assessment of degradation. A semi-empirical NOx emission model has then been identified from engine commissioning data and applied to identify and track degradation at base load [Rudolf et al., 2016]. In later work, Lipperheide et al. [2017] has consolidated the emission models and extended them to the entire commercial operation range, whereas Weidner et al. [2017] proposes a similar semi-empirical approach to model thermo-acoustic combustion dynamics. These models provide a detailed understanding of the underlying mechanisms that drive the long-term evolution of combustion behavior. However, they are relatively cumbersome to set up and maintain. If a detailed physical diagnosis of the behavior is not a primary objective, modern data-based analytics thus bear promise as an alternative, efficient way to enhance combustion monitoring with predictive and prescriptive information.

Such methods have been studied for long in the scientific community, but are still less commonly adopted in industrial practice. Vanderhaegen et al. [2010] contributes a Predictive Emission Monitoring System (PEMS) based on a neural network, which shall serve as a back-up to cover periods when the measurement-based Continuous Emission Monitoring System (CEMS) is unavailable. Palmé et al. [2013] proposes kernel regression-based monitoring of temperature distributions measured at the turbine exhaust, which has shown to be informative on the health status and allows for an early detection of component failure. The data-driven approach proposed in this work however is less common in predictive emission monitoring applications, but superior to previous work as capable of reaching the precision of the CEMS in accuracy.

3.1.3 Working with Historical Data

Working with heavy-duty gas turbines historical data poses a set of unique challenges:

- Working with real historical data is challenging relating to data quality: *sensor noise, sensor biasing, partial observability, observation aliasing, missing values.*
- Heavy-duty gas turbines are sophisticated machines working at extreme physical ranges: *high sensitivity to minimal changes, noise amplification, unfeasible (imputed) sensors.*
  - *High sensitivity*: heavy, thermally loaded parts on a fast-rotating machine are sensitive to minimal changes, exacerbating the consequences of aging and degradation;
  - *High noise*: the severe working conditions (temperature, pressure, rotational speed) tend to amplify noise;
  - *Imputed readings*: some parts of the engine are subject to such extreme physical conditions that designing sufficiently resilient sensors becomes impractical or too costly: values are instead imputed from surrounding readings, which accumulates errors;
- Local environment, working conditions and actual usage history make each engine unique: *uniqueness of build, uniqueness of state, local regulations.*
  - *Unique build and state*: minor tolerances in part manufacturing, engine assembly and adjustment may sensibly influence combustor flow distribution and performance;
  - *Unique activity*: engine operation depends on the local electrical production and usage, with starts and shutdowns resulting in greater engine wear and tear than continuous operation in a steady-state;
  - *Unique regulations*: local laws define pollution limits, making engines installed in different countries run at distinctive regimes;
  - *Data separation*: each engine is strictly defined by its own distinctive history, making quantitative model transfer between engines often inapplicable.

These challenges are addressed as follows:

- **High noise, bias, imputed readings**: filter outliers first, and use algorithms robust to noise in the modeling;
- **Observation aliasing, missing data**: chose models with high generalization capability, and efficient training methods requiring less training data;
- **Data separation**: select training algorithms with high performance and low training time, making it feasible to train a different, independent model for each engine.

3.1.4 Contributions

This chapter introduces the following contributions:

- *Data preparation* (Section 3.2.1): a comprehensive data preparation process enables the application of a broad range of ML algorithms; selecting only data describing normal

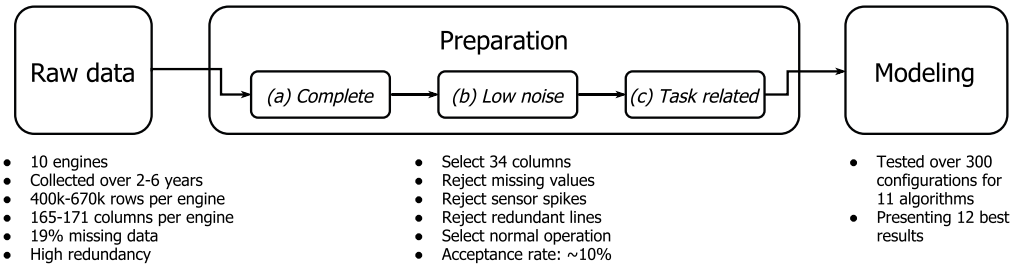


Figure 3.3 – **System pipeline.** The raw data collected from the engine's sensors over the years undergoes a tight filtering process: (a) the data is made complete by dropping lines with missing values, (b) noise is lowered by dropping lines with sensor readings outside physical plausibility, and (c) only data pertinent to the task of choice (normal operation) is selected. Only about 10% of the original data is accepted by the filter and finally used for modeling. The 34 columns mentioned include 33 inputs for the model and the target exhaust gas.

operation provides an additional performance boost. Counter-intuitively, better results are achieved through *smaller* training sets, of carefully selected data.

- *Comparative study* (Section 3.2.2): a broad selection of machine learning algorithms is tested. Comparison details the 12 best performing techniques tested.
- *High-precision modeling* (Section 3.3.2): The best results achieve a precision comparable to the accuracy of the Continuous Emission Measurement System (CEMS) sensors used in industrial long-term operation.

The next section describes the approach leading to the proposed solution.

3.2 System Overview

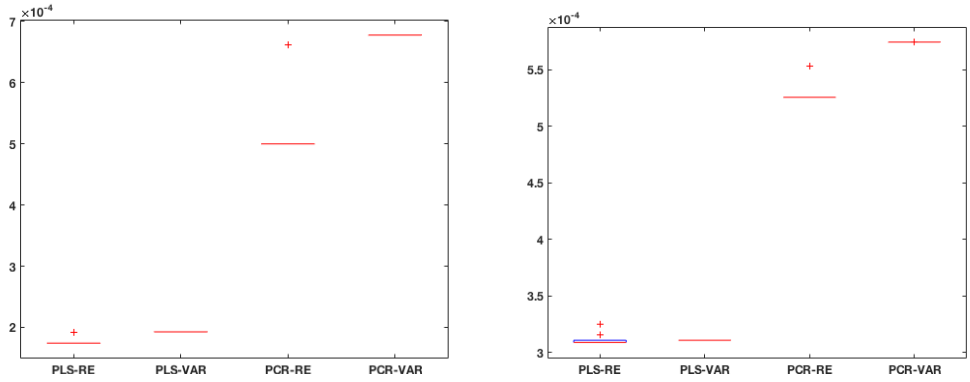
The framework proposed features a machine learning model predicting engine emissions based on the engine configuration and current state (as from sensor readings).

The scope and approach is further detailed in the following sections; an overview is presented in Figure 3.3.

3.2.1 Data Pre-Processing

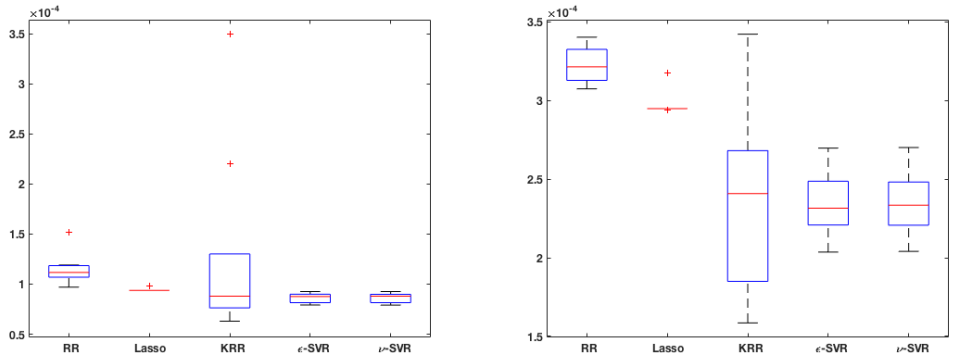
The initially available data amounts to roughly 500 000 rows per engine. Each row holds readings from 180 sensors of various kinds scattered around the engine, such as air pressure after compressor, fuel flow and temperature and engine vibrations, just to name a few.

Direct application of machine learning techniques to the raw data has proven unsuccessful (Figure 3.2), highlighting the need for thorough data preparation. The following describes the processing pipeline put forward in this chapter:



(a) Engine A (b) Engine B

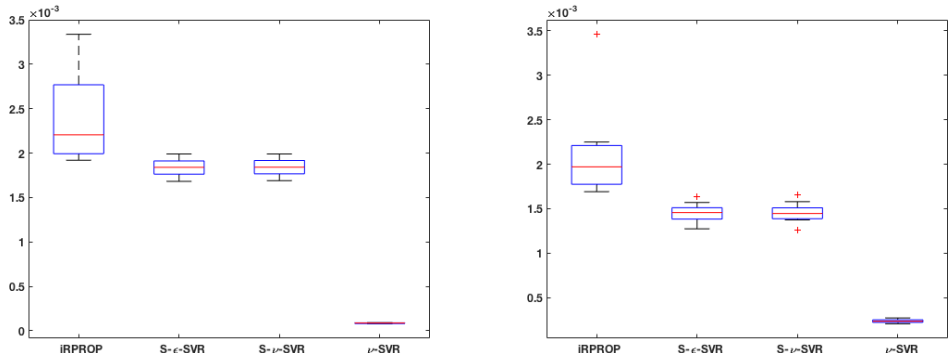
Figure 3.4 – **PLS and PCR.** Boxplots of Mean Squared Errors of methods based on Partial Least Squares and Principal Component Regression. The two graphs (a) and (b) correspond to results on two different engines for generalization. The next figures will present further results on the same two engines. Notice the error bars are compacted into single lines. The difference in reconstruction error is minimal as the scale of the graph is 10^{-4}, but the methods' performance remains consistent across engines.



(a) Engine A (b) Engine B

Figure 3.5 – **RR, Lasso and SVR on *scored* data.** Boxplots of Mean Squared Errors of methods based on data normalized with z-score, namely Ridge Regression (and its Kernel variant), Lasso and Support Vector Regression (in two variants). The two graphs (a) and (b) correspond to results on two different engines for generalization. The difference in reconstruction error is again minimal, but some methods perform consistently better than others across engines although by a varying factor. For example, Lasso has the smallest variation in between runs (tighter error bars) while KRR has the highest. The two SVR methods perform comparably and consistently better than all others.

- Based on statistical analysis, a subset of the columns is selected to provide predictors with the least redundancy. The dependency of the targets to predictors, and independence of the predictors from each other, are assessed by means of HSIC [Gretton et al., 2005], which takes all possible linear and nonlinear correlation into account.



(a) Engine A (b) Engine B

Figure 3.6 – **iRPROP, S-ϵ-SVR and S-ν-SVR on *scaled* data.** Boxplots of Mean Squared Errors of methods based on data normalized with feature scaling, namely Neural Networks (trained with Improved Resilient Backpropagation, iRPROP) and Support Vector Regression (in two variants). The fourth (last) error-bar corresponds to ν-SVR on scored data from the previous plot 3.5b for comparison. Please also note that the scale in this graph is one order of magnitude larger. The average sensor noise is orders of magnitude larger than the reconstruction error: higher precision would correspond to overfitting, and is thus important to refer to the *prediction vs. observation* plots in the next figures.

- Values corresponding to implausible sensor readings (i.e. negative quantities, temperatures below or above real engine ranges, etc.) are discarded as if missing.
- Rows with missing values are simply removed. A viable alternative could have been *data imputation*, but the remaining, complete data proved sufficient to fully train the models.
- Redundant lines (i.e. where all values exactly duplicate another row) are removed. This reduces the training bias towards engine states which are constant over time.
- Since normal engine operation alone is of interest for this study, data recording other engine processes (start up, shutdown, etc.) is carefully selected, and discarded as misleading.

The above mentioned process rejects roughly 90% of the original data, culling a mere ~50 000 lines for the modeling phase.

Considering an error margin of 5%, with 95% confidence level and 50% standard deviation of responses, samples of size at least 382 rows are needed for a statistically significant representation of the population (whole data) [1]. Own empirical experimentation has confirmed a sample size of 1 000 as ideal for all methods to achieve their best performance, robust to 10% variations.

The data is hence partitioned (no intersection) into 10 training sets of size 1 000, with 1 (common) test set comprising of all the remaining data (~40 000 lines). Since the aim is to build a static model (i.e. independent from time), the data constituting each training set is

[1] *Sample size calculator* by Raosoft, inc. Available online at: `http://www.raosoft.com/samplesize.html`

selected randomly (with uniform distribution) across the whole dataset, and then shuffled.

The training sets and the test set are used across all configurations and algorithms without any variation in size, data and order.

3.2.2 Algorithm Selection

Addressing the limitations found in physically rigorous models (Section 3.1.2), the proposed approach is based on machine learning. In particular, regression analysis addresses predicting a real-valued response $Y = f(X)$ for new values of predictors $X = (X_1, X_2, .., X_p)$ [Bishop, 2006, Friedman et al., 2001].

In the context of this project, three main families of algorithms have been considered:

- **Linear regression methods.** Striving for simplicity, and with the goal of establishing a baseline, the following linear regression (with respect to both input and regression coefficients) methods were applied first:
  - *Linear regression* [Friedman et al., 2001], the simplest regression method, provides hindsight on the linearity (or lack thereof) and complexity of the task.
  - *Ridge Regression* (RR; Bell [1978]), introduces a regularization term to the linear regression function, lessening overfitting to predictors.
  - *Lasso* [Tibshirani, 1996], improves RR normalization by enforcing selection of sparser predictors.
  - *Principal Component Regression* (PCR; Hotelling [1957], Jeffers [1967]), uses Principal Component Analysis (PCA; Person [1901]) to derive new predictors which are linearly uncorrelated. Improves the performance of basic regression for cases where the original predictors are linearly correlated.
  - *Partial Least Square Regression* (PLS; Wold [1966], Abdi [2010]). improves on the results of PCR by deriving principle components which maximize the covariance between regression predictors and output.
- **Kernel-based approaches.** Kernelized regression methods are characterized by the usage of kernels. The nonlinear relationship between predictors and outputs is captured in the kernel definition. The following methods were considered:
  - *Kernel Ridge Regression* (KRR; Shawe-Taylor and Cristianini [2004]), which applies Ridge Regression on data embedded in a Reproducing Kernel Hilbert Space (RKHS; Shawe-Taylor and Cristianini [2004]),
  - *Support Vector Regression* (SVR), which constructs using a sparse set of of predictors (*support vectors*) in RKHS. Parameter ϵ in ϵ-SVR [Shawe-Taylor and Cristianini, 2004, Smola and Schölkopf, 2004] and ν in ν-SVR [Schölkopf et al., 2000] control the smoothness of the regression function and the number of support vectors respectively.
- **Feed-forward Artificial Neural Networks** [Bottou and Gallinari, 1990]. A parametrized

generic function approximator is fitted to predict the current emissions based on the current engine state. The complexity of the approximated function depends mainly on the network structure, meaning that a smaller network will provide a simpler model, less prone to overfitting. Generalization capabilities are then leveraged to predict the emissions relative to states previously unseen. The use of non-linear activation functions and multilayer structure allow for approximating complex nonlinear functions. The following choice of backpropagation algorithms has been tested:

- *Incremental Backpropagation* [Rosenblatt, 1961], The original backpropagation algorithm, where the weights of the network are updated for each point in the training set based on its reconstruction error.
- *Batch Backpropagation* [Rumelhart et al., 1986], A standard backpropagation algorithm, where the weights of the network are updated only once per epoch, based on mean square error of all reconstructions. Slower than incremental backpropagation in execution, but faster in convergence.
- *Improved Resilient Backpropagation* (iRPROP; Igel and Hüsken [2000]), An improved, faster variation of the RPROP algorithm [Riedmiller and Braun, 1993], itself an adaptive backpropagation algorithm. The learning step size is adapted to fit both large magnitudes and high precision.
- *QuickProp* [Fahlman, 1988] Advanced batch backpropagation algorithm, optimized for speed in execution as well as in convergence. Particularly effective when scaling to large networks.

Results shown in 3.3.2 come from iRPROP, which in this study consistently achieved best performance. The implementation was based on Fast Artificial Neural Networks (FANN; Nissen [2003]). Training converged after 20 000 iterations in average.

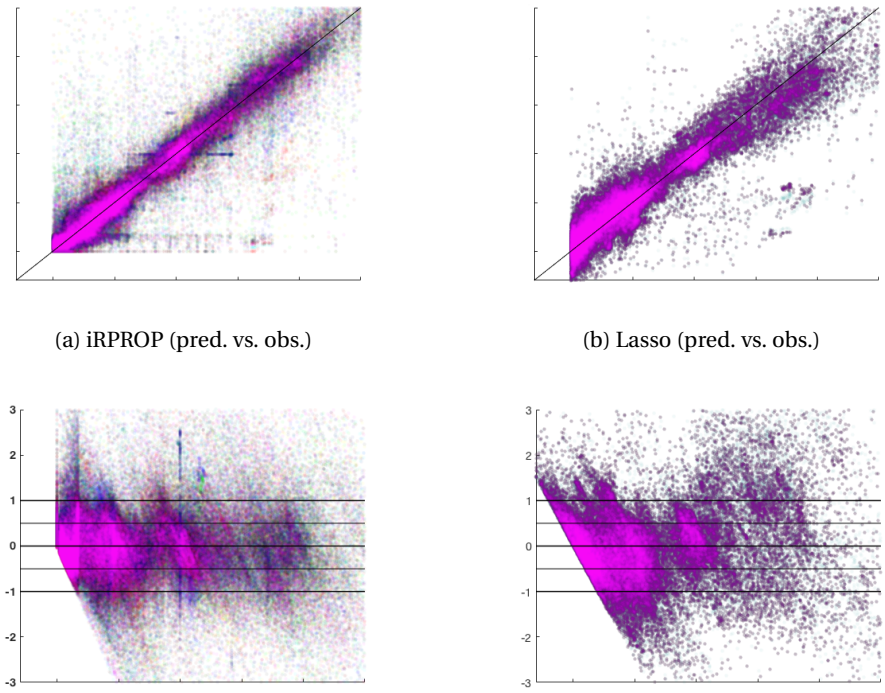
3.2.3 Parameter Cross-Validation

Method-specific parameters were selected based on 10-fold cross-validation. The following are included for reproducibility:

- 'λ' (regularization parameter) in Lasso (10^{-4}), RR (10^{-1}), and KRR (10^{-4});
- 'σ' (Gaussian kernel standard deviation) in KRR (10), ϵ-SVR (1) and ν-SVR (1);
- 'ϵ' (margin) in ϵ-SVR (10^{-2}); 'ν' (number of support vectors) in ν-SVR (0.5);
- NN: λ (learning rate) = 0.7, η_- (decrease factor) = 0.5, η_+ (increase factor) = 1.2, $\Delta_0 = 0.1$, $\Delta_{min} = 0.0$, $\Delta_{max} = 50.0$, and μ (momentum) = 0, structure $\{in = 33+b,\ hid = [10],\ out = 1\}$ (feed-forward, fully-connected), and activation function $\sigma(x) = \frac{1}{1+e^{-x}}$.

The number of principal components in PCR-RE was selected by reconstruction error minimization, and in PLS-VAR by variance maximization.

The next section details the proposed system pipeline.



(a) iRPROP (pred. vs. obs.)

(b) Lasso (pred. vs. obs.)

(c) iRPROP (res. vs. pred.)

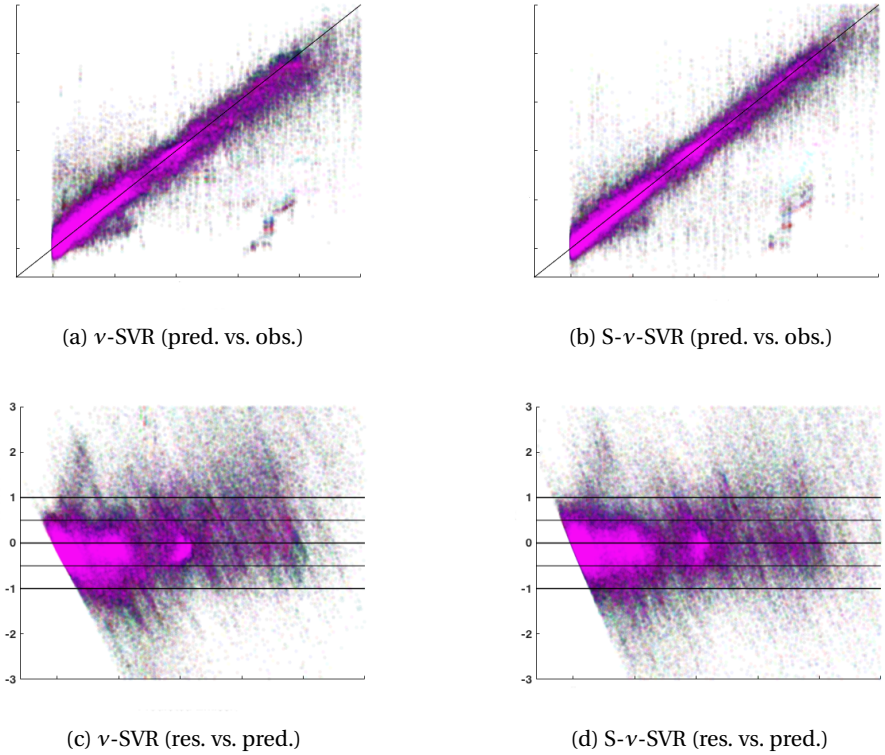
(d) Lasso (res. vs. pred.)

Figure 3.7 – **Results for Lasso and iRPROP.** Graphs (a) and (b) are scatterplots of predicted values (vertical axis) versus observed emissions (horizontal axis) respectively for neural networks trained with iRPROP and for Lasso. The ideal result here is for all points to be on the diagonal, meaning each observation corresponds to a correctly predicted value. Neural networks tend to ignore outliers, producing instead a tighter band on the diagonal. The plots in (c) and (d) show scatterplots of standardized residuals vs. predicted emissions, with bands corresponding to the magnitude of sensor noise.

3.3 Experimental Setup

To predict the engine exhaust based on the engine state, each configuration of algorithm plus parameter set is trained on each of the 10 training sets as described in section 3.2.1. The resulting 10 models per configuration are always tested on the common test set, generating 40 000 predictions. Aggregated configuration results correspond to averages over the ten runs.

All regression methods in this study are applied to standardized data (z-score), except iRPROP which works with scaled data (feature scaling). To see the effect of data normalization, this work investigates the result of SVR methods with both scaled and standardized data.



(a) ν-SVR (pred. vs. obs.)

(b) S-ν-SVR (pred. vs. obs.)

(c) ν-SVR (res. vs. pred.)

(d) S-ν-SVR (res. vs. pred.)

Figure 3.8 – **Results for ν-SVR and S-ν-SVR.** Scatterplots of predicted vs. observed values (a) and (b), and of standardized residuals vs. predicted emissions (c) and (d) for ν Support Vector Regression on z-scored data (ν-SVR) and on feature scaled data (S-ν-SVR). The methods perform comparably to the neural networks trained with iRPROP presented in Figure 3.7a and 3.7c. A detailed discussion and interpretation can be found in the caption of Figure 3.7.

3.3.1 Performance Measures

Standard Mean Squared Error (MSE) of the residuals can become uninformative when the test targets feature high noise. Overfitting the highest-magnitude outliers quickly improves the score, endangering the model's generalization capability. Section 3.1.3 discusses why noise of high magnitude is to be expected in this application.

As a complementary measure of performance, scatterplots of *predicted versus observed emissions* are proposed. This allows to intuitively expect clustering along a diagonal line (i.e. $y = x$), with cluster thickness proportional to model precision (plus sensor noise). Partial transparency of the dots proportionally reduces the visual impact of outliers. Please note the different plotting scales in Figure 3.6b: results for ν-SVR from plot 3.5b are added for comparison.

Scatterplots of *standardized residuals versus predicted emissions* are also presented: a thick line around $y = 0$ corresponds to a uniform distribution of the predictions with normally dis-

tributed residuals, as expected of unbiased noise. Any other clear pattern would be indicative of yet-unlearned relations in the data.

For simplicity, results on only one of the available engines are shown unless clearly stated. Performance across engines for the proposed methods remains comparable, as shown in Figure 3.6.

3.3.2 Results

Among linear regression methods, Lasso and RR obtained better results than PCR and PLS (respectively), implying that regularizing the regression function based on 1-norm ℓ_1 or 2-norm ℓ_2 is better suited to the data rather than regularization using PCA. PLS has smaller MSE compared to PCR because it uses the set of principle components that are best suited to predict target variance.

Nonlinear regression methods obtained the best results, accentuating the importance of nonlinear relationships between output and predictors. Methods that enforce sparsity worked better than non-sparse methods: for instance, SVR is better than KRR, and Lasso is better than RR. This also highlights the importance of variable selection for this task.

Figure 3.4 presents a boxplot comparing the MSE of different regression methods over 10 experiments. The MSE is consistently low across methods, in the order of 10^{-4} of scored emissions. Figure 3.6 shows how comparable results are achieved on other engines.

Figure 3.7 presents predictions versus observations plots for ν-SVR on both scored (3.8a, "ν-SVR") and scaled data (3.8b, "S-ν-SVR"), and Lasso (3.7b, as the best performing linear method) and neural networks (trained with iRPROP, 3.7a) on scaled data. Even though the MSE of ν-SVR is best, the predictions better align with the observations in iRPROP and S-ν-SVR.

The scatterplot of standardized residuals versus predicted emissions in Figure 3.8 shows the residuals having a narrow range of $[-1,1]$ in iRPROP and S-ν-SVR, while ν-SVR residuals mostly fall in the $[-1.5,0.5]$ range. Comparing to ν-SVR, S-ν-SVR and iRPROP, Lasso shows both worse range $[-2,1.5]$ and worse MSE (Figure 3.5a).

The algorithms best performing are feed-forward neural networks trained with iRPROP on scaled data (FFNN+iRPROP), and ν support vector regression on scored data (S-ν-SVR). Their MSEs are both one order of magnitude larger than other methods (still minor, at 10^{-3}), but boasting a much tighter clustering around $y = x$ on the predictions vs. observations plot (fig. 3.7a and 3.8a). Moreover, the plot of residuals vs. predictions shows normally distributed sensor noise, uniformly spread outliers, and no discernible unlearned patterns (fig. 3.7c and 3.8c), implying that the pattern generating the data has been fully learned.

Prediction precision is on the same scale as the expected sensor noise, implying that no better reconstruction can be made without overfitting. Such precision is also deemed sufficient for a

live deployment of the application. Both methods train a new model in less than 10 minutes for a single run, easily allowing for daily builds as new data becomes available.

3.4 Discussion

Heavy-duty gas turbine engines are common, efficient and reliable means to produce electric power. Over the years however their performance gradually degrades, eventually requiring maintenance. Stopping and servicing the engine is costly, compelling to minimize at the same time both the number of scheduled interventions and the risk of unscheduled stops. Accurately predicting the engine's emissions (particularly NOx) can provide information to support decisions over engine adjustment and maintenance scheduling.

Physically rigorous modeling approaches have found limited application because of the complexity of the involved phenomena and the required effort for model set-up and maintenance in an industrially productive environment. At the same time most advanced machine learning methods require data in quantities which are not available in this specific application. This chapter addressed the problem by raising the quality of the data at the cost of further reducing its size, producing an ideal environment for a selection of simpler machine learning algorithms. The custom-filtered dataset is complete, has low noise, and is limited to task-related data. Importantly, the results have been proven impossible to reproduce without such thorough data preparation.

The collection of machine learning algorithms proposed ranges from linear regression to neural networks, allowing key insight as to the level of sophistication necessary for this application. Each method is run on an spectrum of different parameter configurations through cross-validation. Two methods in particular (FFNN+iRPROP, and S-ν-SVR) are shown to be capable of reliably training models with the highest feasible precision (i.e. sensor-level), without overfitting sensor noise. Both methods are comparable in performance and offer short wall-clock running time. Further analysis shows that no unlearned pattern is left in the data at the end of the training.

Modeling NOx emissions to sensor-level precision enables informed scheduling of engine maintenance, minimizing the risk of unplanned shutdowns caused by surpassing the legal limits of NOx production. The data cleaning pipeline proposed can be applied in principle to other datasets featuring the same set of problems such as noise, incompleteness, and redundancy. A broad range of techniques for both denoising and missing values reconstruction is available in the literature, which could lead to the availability of larger amounts of training data. Data generated by reconstructing and cleaning algorithms though can arguably be considered less indicative than the smaller subset of complete, clean data already included in the dataset. The advantages are also limited, as even recovering the full data available would result in orders of magnitude less data than required by the most advanced machine learning techniques, rendering the extra work inconsequential. Selecting an algorithm that is capable of achieving high precision already on a tiny, high quality subset of the original data

has proven highly successful in this application.

The results obtained with neural networks are of particular significance for this thesis. Using an external component to prepare the data for neural network consumption greatly boosts the performance of smaller, shallow networks, to the point of reaching the highest possible performance. This concept will be revisited and further expanded in Chapters 5 and 6. Boosting the effectiveness and efficiency of smaller networks greatly reduces the burden on the training algorithm, enabling the application in the presence of fewer data or heavier training algorithms, such as with evolutionary algorithms (neuroevolution).

To further boost the performance of evolutionary algorithms, the next chapter will focus on addressing *fitness plateaus*, where the evolutionary algorithm's population is comprised of individuals which are indistinguishable based on performance alone.

4 Overcoming Stagnation with Novelty

As seen in Chapters 1 and 2, reinforcement learning neuroevolution provides a way to learn complex interactions based on nothing but a *score* of the performance of candidate solutions. A common issue though is the loss of direction for improvement due to candidate solutions becoming indistinguishable based on performance.

This commonly happens in two cases: fitness plateaus and premature convergence. (i) In the first case, the fitness function presents large plateaus over the space of individuals, i.e. even different individuals obtain the exact same score. This is often the case with problems that accumulate reward based on discrete events: often a large spectrum of individuals will obtain the same score, and the search is stuck until some mutation triggers the next scoring interaction. In the second case (ii) the population maintained by the evolutionary algorithm becomes more and more homogeneous (individuals become indistinguishable by genotype) as a result of the search increasing the resolution and minimizing variance. Unless the convergence basin leads to the global optimum (or a an otherwise sufficient result), the convergence is deemed premature, becoming stuck for all practical purposes. Proper resuming of the search requires a major parametric overhaul.

A common approach in these cases is to restart the search from a new location; selecting a proper location though is itself nontrivial [Fukunaga, 1998]. This chapter proposes to keep track of the exploration during the standard evolution process, and then select as restart location the area in parameter space which has been least explored, leveraging the concept of *novelty search*, as described in Chapter 2.6. Results on two multi-modal problems suggest that this method strikes a balance between completely random exploration and standard exploitation mechanisms.

4.1 Introduction

Evolution Strategies (ES) are a modern class of evolutionary algorithms with advanced features such as scale invariance (invariant to the scale of parameters) and fitness shaping (invariant

to the scale of fitness), making them ideal for black-box optimization applications. While extremely apt at converging in a given attraction basin though, *escaping* such basin to search for better optima requires external mechanisms such as restart strategies [Beasley et al., 1993, Fukunaga, 1998, Auger and Hansen, 2005].

Over the past decade Novelty Search (NS; Lehman and Stanley [2008, 2010]) has received attention due to the starkly opposite approach: disregarding fitness altogether, individuals are ranked by their *novelty*, i.e. by their distance to an archive of reference individuals marking previously explored volumes of the search space. This work was originally aimed at continuous evolution, proposing that (given enough time) a search uniquely based on novelty can be led by the environment's limitation towards the goal even with no knowledge whatsoever of the latter. Further work leverages the method for explicit exploration [Graziano et al., 2011], integrating it with the standard fitness signal to achieve superior results in standard evolutionary algorithms [Cuccu and Gomez, 2011].

This chapter takes yet another direction, by leveraging novel individuals to identify volumes of the search space which are most promising for restarting the search, i.e. where individuals were generated while following the fitness gradient, but where the search has not yet reach the same level of granularity as in other parts of the space. Two functions are used to test this method's performance: a custom function which provide an easy parametrization of plateau and local optima, to push the method to its limits; and the standard Rastrigin function, demonstrating the scaling to higher dimensions.

4.1.1 Radial NES

Natural Evolution Strategies is a family of black-box optimization algorithms presented in Chapter 2.5. Algorithms of this family have characteristics and properties which are considered state-of-the-art such as fast adaptation and invariance to linear transformations. Choosing an algorithm from the NES family for our benchmarks allows to verify that the proposed restart strategy does not break such properties, and is thus compatible with state-of-the-art implementations.

On the other hand though, the algorithms available in the literature (such as XNES and SNES: see Chapter 2.5.2) include advanced mechanics which make them resilient to local optima and plateaus. Since the proposed restart strategy aims at overcoming these problems altogether and independently, the selection of a simpler algorithm, less capable of inherently mitigating these problems, arguably provides a more objective perspective on the restart strategy itself.

To such end, this section introduces *Radial NES* (RNES): a novel, baseline algorithm of the NES family, based on a *radial* Gaussian distribution. This means it maintains a single variance over all dimensions, reducing the distribution parameters to $\theta = (\boldsymbol{\mu}, \sigma)$. The density of the distribution is thereby

$$p(\mathbf{z}|\theta) = \frac{1}{(\sqrt{2\pi}\sigma)^d} \cdot \exp\left(-\frac{\|\mathbf{z}-\boldsymbol{\mu}\|^2}{2\sigma^2}\right) ,$$

which together with the exponential coordinates discussed in Chapter 2.5.2 yields the update equations

$$\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} + \eta_\mu \cdot \sigma \cdot \sum_{i=1}^{\lambda} u_k \cdot \mathbf{s}_k \tag{4.1}$$

$$\sigma \leftarrow \sigma \cdot \exp\left(\frac{\eta_\sigma}{2} \cdot \sum_{i=1}^{\lambda} u_k \cdot \left(\|\mathbf{s}_k\|^2 - d\right)\right) , \tag{4.2}$$

with standard normal samples $\mathbf{s}_k \sim \mathcal{N}(0, \mathbb{I})$ generating the corresponding individuals as $\mathbf{z}_k = \boldsymbol{\mu} + \sigma \cdot \mathbf{s}_k$.

One generation of RNES consists of (i) generating a population of individuals by sampling the search distribution, (ii) sort them based on their fitness, (iii) update the distribution parameters. The execution thus follows Algorithm 1, as seen in Chapter 2.5.1, where Equations 4.1 and 4.2 implement NesUpdate.

4.2 Leveraging the Novelty Archive

As seen in Chapter 2.6, the novelty archive is key to computing the non-stationary novelty signal and adapt to increasing resolution. This characteristic is fundamental to maintain the invariant properties of the evolutionary search. In the proposed setup the novelty score is defined directly in parameter space for RNES, but the method can in principle scale back to higher complexity applications such as neuroevolution and behavioral novelty.

The thought process leading eventually to the restart strategy stemmed from two possible ways to combine novelty and fitness. The next two subsections provide a rationale as to how this could be achieved, the corresponding limitation, and the reasons why a restart strategy is more likely to succeed in this context.

4.2.1 Linear Combination

Combining fitness and novelty has been explored in [Cuccu and Gomez, 2011] using the equation

$$\rho \cdot \text{nov}(\mathbf{z}_k) + (1-\rho) \cdot f(\mathbf{z}_k) \ ,$$

with nov(·) denoting novelty[1] and $fit(\cdot)$ fitness. In RNES this would replace $fit(\mathbf{z}_k)$ in Equation (2.1), providing a smooth gradient even in cases of fitness plateaus.

Such an application expects fitness and novelty to be normalized in the same range in order to meaningfully blend together. In general such normalization could be non-straightforward, as it requires knowing the absolute minimum and maximum values for both fitness and novelty. In NES though this score is solely utilized to sort the individuals in a fitness-shaping context. This enables computing the scores (position in sorted order) individually for fitness and novelty, and compute the blending equation on the corresponding utilities instead, i.e. by replacing u_k in equation (2.2):

$$\rho \cdot u_k^n + (1-\rho) \cdot u_k^f \ ,$$

where u_k^n is the utility of individual $\mathbf{z}_k$ with respect to novelty (equation 2.3), u_k^f its utility with respect to fitness, and $\rho \in [0,1]$ is the same blending parameter.

While such a simple integration mechanism is intriguing, it turns out not to work as intended, because of the interaction between scale-invariance and non-stationary novelty. Based on the chosen value of ρ and point of initialization, the search will initiate a trend towards convergence or divergence, which is then constant on all scales. This implies a sudden transition on the value of ρ, which hinders the search by promoting at once both convergence to local optima and explosive divergence based on a highly unstable switching point.

4.2.2 Hard Switch

A natural follow-up would be to consider ρ restricted to $\{0,1\}$, embracing the hard switch between fitness and novelty. Rendering the two phases independent would allow following both greedy exploitation and explicit exploration in alternating phases. Once convergence is achieved on a local minimum, enabling novelty search would ensure exploration out of that basin towards less explored areas of the parameter space. Finding a new basin of attraction and switching to exploitation once again hopefully allows to converge on a better optimum.

This requires the design of a switching point between the two modes. The fitness-based exploitation can for example go on until a convergence criterion is met. If more evaluations are still alloted to the algorithm, rather than terminating (returning the best individual en-

[1] In this work, novelty is computed against both the archive and the current population.

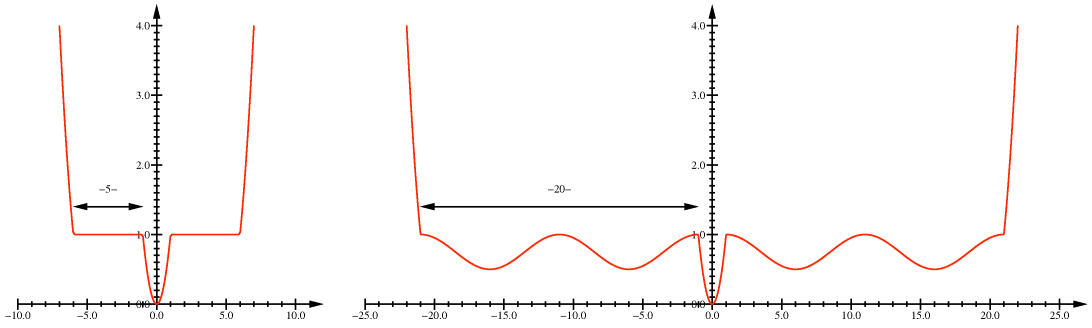


Figure 4.1 – **The function** $f_{\ell,w}$, for configurations ($\ell = 5$, $w = 0$) and ($\ell = 20$, $w = 2$). The first configuration tests the performance of the algorithm in presence of fitness plateaus, while the second and much harder also adds deceptive local minima to the function.

countered so far) the algorithm would then switch to novelty search. In RNES such a criterion is straightforward, as σ is an explicit step size: once this falls below an arbitrary threshold (say 10^{-5}, or a numerical accuracy limit), the algorithm can be considered to have achieved convergence, thus switching to the novelty phase.

4.2.3 Evolutionary Restarts

While intuitively sound, upon further examination the hard switch described above shows some important limitations. When $\rho = 0$ computing the novelty of the individuals could be in principle skipped, although in practice it is still necessary in order to update the novelty archive. At the same time, when $\rho = 1$ the potentially expensive fitness evaluation can be skipped entirely, unless its byproduct are used in computing the novelty (e.g. behavioral novelty). In our case so far, novelty is based on genotypic similarity, meaning it uniquely depends on the parameters set.

With no feedback from the application, how much novelty is enough to decide to switch back to fitness-based exploitation? Examining the distribution variance σ provides no further clue, since while its lower bound is 0, its upper value is unbound, and novelty will naturally bring the search to diverge. Working in isolation presents further challenge regarding updating the novelty archive: adding an individual actively represents having tested and verified that part of the space, but unless the fitness function is actually run, potentially best-performing individuals will not be considered as solutions.

Moreover, where should the search restart from? Switching the search to novelty-based exploration while maintaining the same $\boldsymbol{\mu}$ and σ of the end of exploitation makes it difficult to leave the current attraction basin, by explicitly starting from its center ($\boldsymbol{\mu}$) and with a potentially very small update step size (σ). Once the option of a search restart based on novelty comes under consideration, the issues discussed so far become irrelevant.

	$\ell = 5$	$\ell = 15$	$\ell = 30$
$w = 0$	97	87	85
$w = 2$	19	3	0
$w = 10$	60	13	2

Table 4.1 – **Performance without novelty restarts.** Percentage of trials (out of 100 runs) in which RNES *without novelty restarts* finds the global optimum of the custom function $f_{\ell,w}$, with ℓ length of the plateaus and w number of local optima ("waves") on each plateau.

This chapter proposes to restart the search on the the individual with highest novelty so far: high novelty implies an area of the space which has been reached by the evolution in the past, but has been explored the least. To restart the search, the new $\boldsymbol{\mu}$ is set to the novel individual, while σ is calculated as the average distance between $\boldsymbol{\mu}$ and a set of k nearest neighbors, with k fixed at initialization. This allows to immediately resume fitness-based exploitation, aiming to converge to a potentially new and hopefully better optimum. And even if not, searching around $\boldsymbol{\mu}$ will automatically lower its novelty, ensuring that further restarts will not consider the same area again.

The new starting parameters for the search distribution respect the invariants of the underlying ES: $\boldsymbol{\mu}$ had been previously generated by the search and is thus a plausible value, while σ is derived from previous exploration around the same volume and is thus also consistent. Using novelty uniquely for its novelty archive rather than for an intrinsic score for the individual relates this approach to other previous archive-based evolutionary methods as common in multi-objective optimization [Zhang and Sanderson, 2007, Knowles and Corne, 1999], except for the use of novelty as its underlying criterion.

4.3 Results

Novelty-based restarts are tested in the following sections to verify its contribution in presence of fitness plateaus and multimodal fitnesses. One important consideration is how the first run up until the restart is identical between an algorithm enhanced by novelty restarts and a standard ES, thus the addition of the restart strategy does not alter minimal performance. This particularly implies that applying novelty restarts to a task with a unimodal fitness landscapes, while providing no performance improvements, will produce no downside, making the decision to adopt this method much more straightforward.

As in [Schaul et al., 2011], the default values for population size and learning rate come from optimized heuristics in CMA-ES [Hansen and Ostermeier, 2001]: for all experiments the population size was set to $\lambda = 4 + \lfloor 3\log(d) \rfloor$, and the learning rates to $\eta_\mu = 1$, and $\eta_\sigma = (3 + \log(d))/(5\sqrt{d})$ respectively, with d number of parameters (dimensions).

The function $f_{\ell,w}$ utilized in the following experiments is designed as follows:

$$f_{\ell,w}(x) = \begin{cases} x^2 & \text{for } |x| \leq 1 \\ 1 - \frac{1}{2}\sin^2\left(\frac{\pi w(|x|-1)}{\ell}\right) & \text{for } 1 \leq |x| \leq \ell + 1 \\ (|x| - \ell)^2 & \text{for } |x| \geq \ell + 1 \end{cases},$$

with parameters $\ell \geq 0$ and $w \in \mathbb{N}$. This function is based on a standard parabola and thus symmetric around its global optimum at 0. The parabola is though chopped open at $x = +1$ and $x = -1$, and the branches diverging from the central basin are shifted outwards by plateaus of length ℓ. Additionally, w waves can be added to these extensions, corresponding to an equal number of local optima (Figure 4.1). The distribution is initialized at every trial with parameters $\boldsymbol{\mu} = \ell + 1$ and $\sigma = 1$.

4.3.1 Overcoming Multimodal Plateaus

The first set of experiments studies the performance of the algorithm on both fitness plateaus and simple multimodal landscapes. Fitness plateaus render individuals indistinguishable by awarding exactly the same fitness to all, thus providing no gradient nor direction for improvement. Adding shallow local optima with attraction basins of arbitrary size further increases the difficulty, as a search reaching the very edge between the global optimum attraction basin and the last (shallow) local optimum may still be attracted back away from the global optimum.

RNES was run with and without novelty restarts on the f function, with all combinations of parameters $\ell \in \{5, 15, 30\}$ and $w \in \{0, 2, 10\}$. Flat fitness plateaus correspond to $w = 0$. Each run was alloted 10 000 generations, corresponding to 40 000 fitness evaluations. The stopping criterion, also used to trigger restarts, was $\sigma < 10^{-10}$.

Table 4.1 shows the percentage of runs in which RNES *without* restarts was able to converge to the global optimum. RNES copes relatively well with plateaus of limited length (proportional to its starting σ), but the addition of multiple local optima (*waves*) makes the task much harder. Particularly, the task is hardest for only two local optima (per side), because for any given ℓ the size of each attractor basins is inversely proportional to the number of basins present. A large ℓ and low w correspond to (fewer but) larger attraction basins, which are harder to escape.

Table 4.2 reports instead the median number of generations it took RNES *with* novelty restarts to identify the optimum with an accuracy of 10^{-5}. The algorithm was successful in all runs for most configurations. The restart strategy helps to identify the optimum reliably, and scales gracefully with plateau length and number of local optima. Figure 4.2 depicts a typical novelty archive at the end of a run. The number of attraction basins visited is a testament to the ability of the restarts to overcome local convergence and stagnation. At the same time, the fitness-based search makes sure that the interesting regions around the local optima are sampled at a

	$\ell = 5$	$\ell = 15$	$\ell = 30$
$w = 0$	79	298	319
$w = 2$	267	668	(677)*
$w = 10$	101	792	903

Table 4.2 – **Performance with novelty restarts.** Median number of generations to find the global optimum, out of 5 runs. RNES + novelty restarts consistently succeeds in all runs but for one configuration. The target function is the same $f_{\ell,w}$ as Table 4.1, with ℓ length of plateaus and w number of local optima ("waves") on each plateau. The hardest configuration is $\langle \ell = 30, w = 2 \rangle$ which corresponds to the longest plateau and smallest (nonzero) number of local optima, the latter consequently have the largest attraction basins. (*) This configuration was solved only in one out of 5 runs, in 677 generations.

much higher resolution than the rest of the space.

4.3.2 High Dimensions and the Rastrigin Function

The function $f_{\ell,w}$ represents by design a worst-case scenario for ES. The lack of gradient in the flat version $w = 0$ induces such divergence in the search that the algorithm struggles to move $\boldsymbol{\mu}$ and gradually reduce σ even if individuals are luckily sampled from the global optimum basin, rendering recovery unlikely in the finite number of generations alloted. The introduction of deception with $w > 0$ only exacerbates the condition, by making the algorithm converge on suboptimal basins, in turn diminishing its exploratory capability.

The restart strategy here not only drastically improves the performance of the search, but actually enables as simple an ES as RNES to be applied to complex fitness landscapes. Novel elements are treated as milestones, from which the search can resume whenever meaningful local improvements are no longer detected.

However this could in principle be imputed to an archive-filling behavior, where all local optima are in turn added an archive practically used as an exclusion list. Such a case would hinder the ability of the algorithm to scale to higher dimensions, as the number of local optima scales exponentially, beyond the ability of any explicit archive to keep track of them.

To test whether or not this is the case, the next experiments are based on the Rastrigin function, a common benchmark for search performance on high dimensional spaces. It presents a d-dimensional quadratic global trend, overlayed with a grid of local optima:

$$f_{\text{Rastrigin}} : \mathbb{R}^d \to \mathbb{R}, \quad x \mapsto 10d + \sum_{i=1}^{d} \left[x_i^2 - 10 \cdot \cos(2\pi x_i) \right] \ .$$

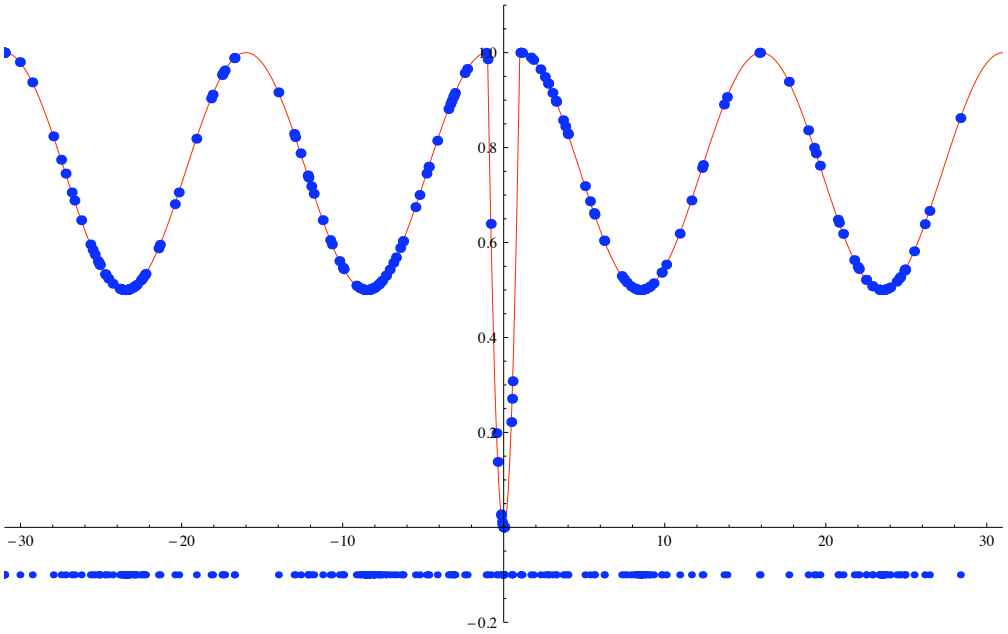


Figure 4.2 – **Novelty archive** at the end of a typical run of RNES with novelty restarts for our hardest configuration of $\ell = 30$ and $w = 2$. Exploring the local optima at increasing resolution essentially marks them as "explored", prompting the next restart to look elsewhere. These results by themselves could be in principle imputed to an archive-filling behavior: further experiments in Section 4.3.2 however address this concern by proving it unnecessary.

$d = 2$	$d = 5$	$d = 10$	$d = 20$
24.6	25.6	18.9	16.2

Table 4.3 – **Percentage of trials** in which RNES *without novelty restarts* finds the global optimum of the Rastrigin function, out of 1 000 runs.

RNES with novelty restarts was tested on this setup initializing $\boldsymbol{\mu}_0$ randomly from a uniform distribution in $[-5,5]^d$, with $\sigma = 1$. Similarly to the previous setup, Table 4.3 reports the percentage of successful runs for RNES *without* novelty restarts, while Table 4.4 highlights the number of generations needed for RNES *with* novelty restarts to succeed. Experiments were conducted for $d \in \{2,5,10,20\}$, with 10 000 generations per run.

The results show that novelty restarts are capable of scaling to relatively high dimensions. For perspective, consider that even in the hypercube $[-1.5,1.5]^d$ the Rastrigin function has 3^d *local optima*. Sensibly representing them all in its explicit archive would require a scaling of at least $\Omega(3^d)$, which would not be feasible. Our results show instead a much more graceful scaling, thanks to the Rastrigin problem providing a global trend that favors exploration, making it (from our algorithm's standpoint) much less deceptive than the purposely-built $f_{\ell,w}$ function. This is arguably a much more common condition in most problems and real-world applications, which emphasizes the actual difficulty of our custom engineered, worst-case

$d = 2$	$d = 5$	$d = 10$	$d = 20$
352	234	64.3	$(1374)^*$

Table 4.4 – **Median number of generations** (over 25 runs) it takes RNES *with* novelty restarts to identify the global optimum of a d-dimensional Rastrigin function. With $d = 20$ the algorithm succeeded in 23 out of 25 runs in the alloted budget of 10 000 generations.

$f_{\ell,w}$.

4.4 Discussion

Fitness stagnation is a common problem for ES when the fitness landscape includes plateaus (as common in reinforcement learning) and multimodality (which leads to premature convergence). Novelty search sidesteps these problems by uniquely focus on individual novelty rather than fitness. Integrating a novelty-based component into fitness-based ES would lead to improved performance.

A direct integration of fitness and novelty in the objective score however leads to renouncing scale invariance, which limits its application to extremely simplistic ES. A hard switch between fitness- and novelty-based objectives is also proved unsuccessful, causing the algorithm to diverge. In order to carry novelty-based improvements to state-of-the-art ES, this paper presents novelty-based restarts. This is a simple yet powerful restart strategy which stores information about the search so far in an explicit archive, leveraging it then to propose promising configurations for resuming an already-converged run. Notably this setting respects the black-box invariants which make modern ES so flexible, since all information for the restart is extracted from the run itself, without introducing extra hyperparameters.

In order to clearly highlight the advantage of such an approach, this work presents a worst-case-scenario fitness function to study how the algorithm performs in presence of fitness plateaus and multimodal problems. A new algorithm of the NES family then provides an ES as simple as possible while retaining state-of-the-art characteristics. This allows verifying that the restart strategy respects the requirements of (and thus can be ported to) modern ESs.

A second benchmark is based instead on the Rastrigin function, arguably less deceptive but more realistic, and capable of scaling to arbitrarily higher dimensions. The results highlight the ability of the algorithm to perform in high dimensions without relying on an archive-filling behavior, which would require a potentially exponential quantity of elements as the dimensionality grows.

The next chapter continues on this direction by further investigating the applications of Novelty Search to sustain diversity in the EA's population.

5 Feature Extraction and Decision Making

Novelty Search (Chapter 2.6) can provide an EA with *intrinsic motivation* to further the search even when individuals are indistinguishable based on fitness alone (i.e. *stagnation*). In the previous Chapter (4) this was implemented through a novel restart strategy: the EA is allowed to completely converge before being redirected to a new interesting area of the search space. Information about individual novelty though could support the evolution itself even *before* convergence, if the setting allows so; this happens to be the case with neuroevolution.

Further work in this direction is motivated by the need to address reinforcement learning problems sporting complex, high dimensional observations such as raw images. Recent results on evolving deep neural network controllers end-to-end in this context has recently proven the viability of such an approach [Salimans et al., 2017, Chrabaszcz et al., 2018, Such et al., 2017, Risi and Togelius, 2017]. This method though requires high computational resources, while blending (and thus hiding) the distinction between feature extraction and decision making.

This chapter explores the application of unsupervised learning as a *compressor* (i.e. pre-processor, feature extractor) that takes the observation and returns a compact code to the network decision maker. Such a compressor is trained online along with the network evolution, on images obtained by the network controllers as they interact with the environment. This incidentally makes the compressor aware of the history of observations seen so far, as yet-unseen images will be harder to compress. Since obtaining novel observations requires the network to interact with the environment in novel ways, the network's score is boosted proportionally to the novelty of the images obtained by its interaction with the environment.

Experimental results are based on a new vision-based version of the classic *mountain car* benchmark [Sutton, 1996]: the network's only input is a third-person representation of the simulation rather than the standard vectors of position and velocity. This greatly raises the task complexity, with the controller now deciding the next action based on a higher-dimensional and less informative input.

5.1 Introduction

Until the advent of deep learning, training neural networks using evolutionary computation (*neuroevolution* (NE; Yao [1993], Floreano et al. [2008]) used to have clear advantages over single-agent gradient-descent methods for reinforcement learning (RL) tasks [Runarsson and Lucas, 2005, Lucas and Runarsson, 2006, Lucas and Togelius, 2007], especially in application requiring *memory* [Gomez, 2003, Stanley, 2004]. Applications to high-dimensional inputs such as images were limited due to computational resources limitations, which promoted work using *indirect encodings* to generate larger networks from a smaller set of parameters [Gruau, 1994]. While in direct encoding the parameters identically correspond to network weights (as described in Chapter 2.4), indirect encoding uses complex genotype to phenotype functions. For example intermediate networks [Gauci and Stanley, 2007] or (de)compression algorithms [Koutník et al., 2010, 2013b], defined over the EA's parameters, are used to ultimately generate the network weights.

This chapter proposes an alternative approach: instead of exposing the network directly to the high-dimensional input, an unsupervised learning (UL) *compressor* produces a smaller encoding from the observation. This is used in turn by the network as input, greatly reducing the size of its input layer, and negating the need to dedicate additional layers to feature extraction. This work also trains the compressor *online*, on the very observations obtained by the individuals of a population at each generation, in turn removing dependencies on prior knowledge and bootstrapping.

The work described in this chapter, which was previously presented as a paper [Cuccu et al., 2011b], was the first instance of UL used as pre-processing to boost neuroevolution. Previous work on single-agent RL methods [Lange and Riedmiller, 2010, Legenstein et al., 2010, Fernández and Borrajo, 2008, Gisslén et al., 2011, Pierce and Kuipers, 1997, Jodogne and Piater, 2007] showed limits in the quality of the produced features: a single agent interacting deterministically with the environment will hardly obtain novel observations until a late stage, and allowing high training flexibility in late stages typically causes forgetfulness of early-learned features. This is tackled in this chapter through neuroevolution by training a single, common compressor on observations obtained by all the individuals in the population. This guarantees a larger pool of behaviors at each generation, which generates a broader sample of observations.

Another key improvement is the introduction of intrinsic motivation [Schmidhuber, 1991, 2006, 2010] to boost the score of some individuals. Since the compressor is trained online on the very observations obtained by the individuals during evolution, it will naturally be biased towards such observations, which make it less apt at compressing previously-unseen images. At the same time, while obtaining a previously-seen observation may or may not correspond to a previously-seen state of the environment, a previously-unseen observation correspond necessarily to a novel environment state (under the assumption of an deterministic environment).

The ability of an individual to produce novel observations is thus directly linked to its ability to bring the environment to novel states, which implies a behavior capable of novel (previously unseen) interaction. Rewarding such ability steers the search towards producing offspring that further explores such interactions, offering direction for improvement even in presence of fitness plateaus or multimodal landscapes, similarly to what previously discussed in Chapter 4.

This work presents a novel compressor based on Vector Quantization (VQ; Gray [1984]), and uses Separable Natural Evolution Strategies (SNES; Schaul et al. [2011]) as the evolutionary algorithm. The genotypes are interpreted into single-layer fully-connected recurrent neural networks (RNN). The new visual mountain car environment has been introduced by this work to provide a complex, high-dimensional problem to assess the performance of this system. The following section will delve deeper into sensory compression (Section 5.2), framework architecture (Section 5.3) and current implementation (Section 5.4), before presenting experimental results (Section 5.5.3) and discussion (Section 5.6).

5.2 Compressing Observations

This section provides a formal definition of the process described in the work.

At each time step t, an observation $\mathbf{o}_t \in \mathbb{R}^{dim}$ is obtained from the environment based on its current state. This is encoded by compressor C into a code vector $\mathbf{c}_t \in \mathbb{R}^n$ where $n \ll$ dim as follows:

$$\mathbf{c}_t = C(\mathbf{o}_t) \ .$$

The action $\mathbf{a}_t$ is decided by the controller, a neural network which approximates policy π, by using the current code as input. The use of recurrent connections in the networks makes them potentially aware of the entire history of code inputs and actions ($\{\mathbf{c}_t, \mathbf{a}_t, \ldots, \mathbf{c}_0, \mathbf{a}_0\}$) through its internal representation $\mathbf{m_t}$:

$$\begin{aligned} \mathbf{a}_t &= \pi(\mathbf{c}_t, \mathbf{m}_t) \\ \mathbf{m}_{t+1} &= G(\mathbf{m}_t, \mathbf{c}_t, \mathbf{a}_t) \ . \end{aligned}$$

The neural network thus takes input $\mathbf{c}_t$ and computes the internal activation as $G(\cdot)$, yielding the following equation:

$$\mathbf{a}_t = \pi(\mathbf{c}_t, G(\mathbf{m}_{t-1}, \mathbf{c}_{t-1}, \mathbf{a}_{t-1})) \ .$$

Inverting the compression operation uses operator $R(\cdot)$ to produce a reconstruction $\hat{\mathbf{o}}$ as follows:

$$\hat{\mathbf{o}}_t = R(C(\mathbf{o}_t)) \ .$$

This enables measuring the quality of the reconstruction in terms of distortion (i.e. *reconstruction error*) $d(\mathbf{o}_t, \hat{\mathbf{o}}_t)$, using a measure such as the squared error:

$$d(\mathbf{o}_t, \hat{\mathbf{o}}_t) = \frac{1}{2} \|\mathbf{o}_t - \hat{\mathbf{o}}_t\|^2 \ . \tag{5.1}$$

The overall quality of the compressor can then be evaluated based on its expected distortion:

$$D(C,R) = \sum_{\mathbf{o}} p(\mathbf{o}) \ d(\mathbf{o}, R(C(\mathbf{o}))) \ , \tag{5.2}$$

where $p(\mathbf{o})$ indicates the probability for observation $\mathbf{o}$ to be returned by the environment. Let the compressor be parameterized by $\mathbf{w}$; if the gradient $\nabla_{\mathbf{w}} D(C,R)$ can be computed or approximated, the compressor can be improved through unsupervised learning.

While the code generated this way is guaranteed to be lower dimensional than the original observation, no guarantees are given in terms of *observation aliasing*, where different observations can in principle generate the same code. This adds up to the intrinsic observation aliasing of the environment, where different internal states can in principle lead to identical observations. The problem is mitigated through the use of RNNs, which leverage their internal memory to distinguish states with identical codes, based on the sequence of observations and actions which led to them.

5.3 Proposed System

The proposed architecture augments the standard neuroevolution setup with a compressor which pre-processes the observation into a lower-dimensional code, as seen in Figure 5.1

Standard neuroevolution maintains a population of individual genotypes (parameter sets), initialized at random, each corresponding to a neural network phenotype. Evaluating a genotype (individual) corresponds to generating the corresponding phenotype (the neural network), and interactively running it against an environment (e.g. in a RL task), which obtains a score used as the individual fitness. A new population of offspring can then be produced based on the most fit chromosomes, corresponding to an update of the distribution

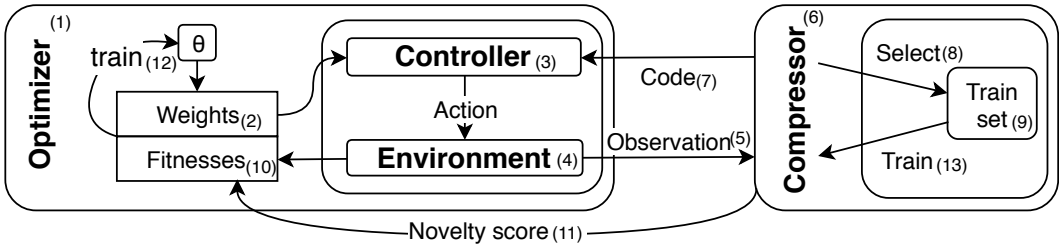


Figure 5.1 – **System architecture.** Following classic neuroevolution, an evolutionary algorithm (1) generates a population of individuals (2), each of which is interpreted into a neural network (3) and evaluated in a task against an environment (4). The difference though here is that each observation (5) is sent to an external compressor (6) rather than the network directly. The compressor produces a compact code (7) that the network uses as input. Each observation is independently considered by the compressor (8) to further its training (9), which is done in-between generations. At the end of each individual run, the fitness of the individual is computed by a composition of the environment's cumulative reward (10) and the compressor's estimate of the individual's novelty (11). The training update of both the evolutionary algorithm (12) and the compressor (13) can take place independently once all individuals have undergone the evaluation process.

parameters in SNES (Section 2.5).

During an individual's evaluation, observations are *interpreted* into network inputs. The network is then activated, and its outputs are again interpreted into a selection of the next action. The observation interpretation however does not usually go beyond preparing the input vector for network consumption, e.g. by normalizing the observation's values in the range required by the activation function of choice. The work here presented proposes the inclusion of a *compressor* producing a lower-dimensional code for the network, which can be interpreted as an advanced observation preparation (Algorithm 2). A smaller input size implies a reduced input layer for the network's, the latter often accounting for the majority of the network's overall weights (and thus complexity).

The same compressor is shared between all individuals in a population, ensuring fair individual comparison. The training is delayed to the end of each generation. The training set is composed by a list of observations selected by the compressor itself out of all observation it receives for compression. These observations are generated by the environment as a consequence of its interaction with (one of) the population's individuals. Interleaving the *online training* with the evolution process thus makes the compressor aware of the *novelty* of each observation, simply computed through reconstruction error of the compressed image.

During the first generation, individuals will generate most likely relatively simpler controllers, in terms of their ability to interact with the environment towards addressing the task's goal. These individuals will thereby have limited capabilities of exploring the range of possible interactions with the environments, in turn obtaining only a subset of the possible observations from the environment. During earlier generations, the compressor will use these for training,

specializing on such subset.

Later generations will see the emergence of individuals capable of interacting with the environment in ever novel ways, leading in turn to new observations. The compressor is by definition not trained to efficiently compress these observations, which in turn corresponds to higher reconstruction error and thus higher novelty (as seen in Chapter 2.6). In turn this makes them prime candidates to be selected for compressor training, meaning the next generation of individual will be able to rely on a compressor ready to encode the previously novel observations.

At the same time, while low-novelty observations can come in principle from either old or new ways of interacting with the environment, high novelty observations can only be generated through novel ways of interacting with the environment. This means that an individual's *behavioral novelty* can be approximated by the novelty of the observations generated during its interaction with the environment, e.g. as the highest novelty obtained by an observation during its run.

At the end of an individual's evaluation, the fitness score typically returned to the evolutionary algorithm is based on the total accumulated reward obtained by the individual throughout the run. This work augments this score with a measure of the novelty of the individual, as deduced from the novelty of the images it obtained from the environment, following [Cuccu and Gomez, 2011]. The complete loop thus becomes (details in Algorithm 2): (i) the environment produces an observation for the agent current run; (ii) the compressor produces a short code for the agent's neural network, which will use it to select an action leading to the next observation; (iii) internally, the compressor produces a reconstruction based on the code, used to compute the observation's novelty; (iv) this novelty is used both to decide whether to include this image in the training set, and to compute the individual's novelty (max over all observations).

The networks' evolution and the compressor's training thus work independently but in synergy. At the beginning, networks and compressor are both initialized at random: the compressor produces a code of little value, but the initial networks would anyway not profit by high-quality codes. Encoded observations nonetheless enable the networks to produce actions, which lead to more observations from the environment, which are used by the compressor for training, increasing the quality of its encodings over time. At the same time the evolutionary algorithm picks up a fitness gradient to improve the neural network population. As the control sophistication rises in later generations, new environment states are reached, leading to ever novel images, supporting a continuous improving of the compressor.

5.4 Experimental Setup

The evolutionary algorithm selected for the proposed setup is Separable NES (SNES; Schaul et al. [2011]). This algorithm and its Natural Evolution Strategies family are discussed in Chapter 2.5. SNES provides the fastest performance of the family, making it applicable to

evolve larger networks: this enables running the same experiments on larger networks using raw pixel as inputs, providing a better performance assessment.

The specialized, online version of Vector Quantization (VQ; Gray [1984]) used as the compressor is instead a novel method using per-centroid decaying learning rates to enable late learning while limiting early-features forgetfulness, introduced in the next sections.

Algorithm 2 UL+NE

```
Initialize( best_FIT, best_IND)
o_t ← Initialize(ENV)
while not solved(Env) do
    for IND ∈ population do
        Initialize(NOV_IND)
        for step = 1... maxsteps do
            [error, code] ← COMPRESS(o_t, 𝒟)
            if error > NOV_IND then
                NOV_IND ← error
                o_max ← o_t
            a_t ← SelectAction(IND, code)
            o_t ← ENVUPDATE(a_t)
        fit← f_task ⊕ NOV_IND
        if fit > best_FIT then                    ▷ Best performing individual so far
            best_FIT ← fit
            best_IND ← IND
        𝒳 ≪ o_max                                 ▷ ≪ Append to compressor training set
    UPDATEEA()
    𝒟 ← TRAINCOMPRESSOR(𝒟, 𝒳)
    𝒳 ← ∅
return best_IND
```

5.4.1 Vector Quantization and Online Learning

Vector Quantization (VQ; Gray [1984]) is an algorithm for lossy compression and dimensionality reduction. Widely regarded for its simplicity, it originated in the context of signal processing. It works by maintaining a dictionary $\mathscr{D}$ of centroids $d_0 \dots d_{|\mathscr{D}|}$, with $|\mathscr{D}|$ being the dictionary size. An input vector $\mathbf{o}$ is compared with each centroid in the dictionary, producing a vector of $|\mathscr{D}|$ values collectively called a *code* $\mathbf{c}$. In its original implementation, the code is a vector of zeros with a single 1 in correspondence of the centroid deemed most similar to the input vector (lowest reconstruction error).

$$C(\mathbf{o}_t) = \mathbf{c} = [c_0 \dots c_{|\mathscr{D}|}] \ , \ c_i = \begin{cases} 1 & \text{argmin}_i \left[d\big(\mathbf{o}_t, R(i)\big) \right] \\ 0 & \text{otherwise} \ . \end{cases} \tag{5.3}$$

Reconstructing an input vector $\mathbf{o}$ based on its code $\mathbf{c}$ corresponds to returning this most similar centroid, which can be obtained with a product of the code and the dictionary $\hat{\mathbf{o}}_t = \mathbf{c}\mathscr{D}$. This basically partitions the input space in $|\mathscr{D}|$ regions based on the centroids and the chosen distortion / error measure.

Minimizing the expected squared-error distortion in batch for all possible observation yields the K-means algorithm. Porting the algorithm to online learning, where the batch has unbound size and is not available beforehand, requires instead to follows the gradient of decreasing distortion $\nabla_{\mathscr{D}} D(C,R)$ (after Equation 5.2). The update rule for each centroid hence becomes:

$$\begin{aligned} \mathscr{d}_i &\leftarrow \mathscr{d}_i + \alpha\, c_i\, (\mathbf{o}_t - \mathscr{d}_i) \\ &= (1 - \alpha\, c_i)\, \mathscr{d}_i + \alpha\, c_i\, \mathbf{o}_t \ , \end{aligned}$$

where $c_i = 0$ for all elements but where $C(\mathbf{o}_t) = i$, for which it is 1. Intuitively, this moves the trained centroid towards the training element $\mathbf{o}_t$ with a step size defined by the learning rate α, decreasing the overall distortion, in a fashion similar to a "mixture of experts" system [Dietterich, 2000].

As discussed earlier though, some of the observations are seen by the compressor only during later stages of training. The compressor needs to maintain a consistent encoding for observations seen earlier, while still being flexible enough to learn new features in the future. Particularly in the case of VQ, this means that some centroids need to specialize to earlier observations, while others need to retain the adaptability to ensure later training.

To obtain this, each centroid is augmented with a different learning rate depending on the number of times it has been selected for training. The training keeps selecting the same centroid for training images which are most similar to the centroid itself, which specializes but lowers its learning rate the further it is trained. Novel observations will instead be matched to untrained centroids, which will retain a higher learning rate and will thus quickly adapt to their new specialization. In this work, the centroid-specific learning rate α_{h_i} is set to $1/h_i$, with h_i being the number of times centroid i has been trained (its *history*). A constant ω acts as a lower bound to prevent convergence to zero, which would render the centroid unable of further adaptation. The learning rate thus *decays* by $1/h$ until reaching ω, from which point on it remains constant:

$$\alpha_{h_i} = \begin{cases} 1/h_i & \text{if } 1/h_i > \omega \\ \omega & \text{otherwise} \ . \end{cases} \tag{5.4}$$

5.5 Experimental Setup

All experiments use fully-connected single-layer recurrent neural networks with a total of three neurons, one corresponding to each action, directly connected to all inputs. Action selection is performed stochastically using a softmax function on the output layer:

$$p(a_i) = \frac{\exp(a_i/\tau)}{\sum_{j=1}^{3} \exp(a_j/\tau)} \quad , \tag{5.5}$$

with temperature $\tau = 0.0001$.

The compressor's dictionary size is fixed at $|\mathscr{D}| = 8$ centroids. Each centroid is an array of 450 elements, matching the observations from the environment, initialized at random. The code is thus *56 times smaller* than the raw observation, and the network's input connections consist of $8 \times 3 = 24$ weights instead of $450 \times 3 = 1350$ weights – plus 3 recurrent connections and 3 biases in both cases. The training set for the compressor is built by selecting a single image from each individual at each generation. Under the reasoning that a useful image for training is arguably the one that the compressor struggled the most to reconstruct [Schmidhuber, 1991], the image selected for training is the one with highest novelty, i.e. the highest reconstruction error. Notably the images are set aside during the individuals' runs, as the novelty of each observation is already calculated to estimate the individual's novelty for fitness scoring, nicely integrating the setup. The compressor is then trained at the end of each generation, which resets the training set to empty.

The next sections presents the new visual mountain car task, this work's setup, and the corresponding results.

5.5.1 Visual Mountain Car

Our configuration is tested on a new, visual-based version of the *mountain car* benchmark [Moore, 1991, Singh and Sutton, 1996], a standard benchmark for the ability of an algorithm to perform in presence of large fitness plateaus (Figure 5.3). The car, controlled by the agent, can only accelerate forward or backwards (plus a `NOOP` option). At full acceleration from a standstill on the bottom of the valley, the car does not have sufficient torque to climb the steep slope and reach the goal. A successful agent thereby needs to learn a *swinging* behavior, where a stage of forward thrust and one of backward thrust are alternated at the stalling points. This allows accumulating the necessary momentum for the car to overcome the steepness and reach the rightmost peak, thus succeeding in the task.

The environment state is determined by the car's position and velocity. The results presented are governed by the following update rules:

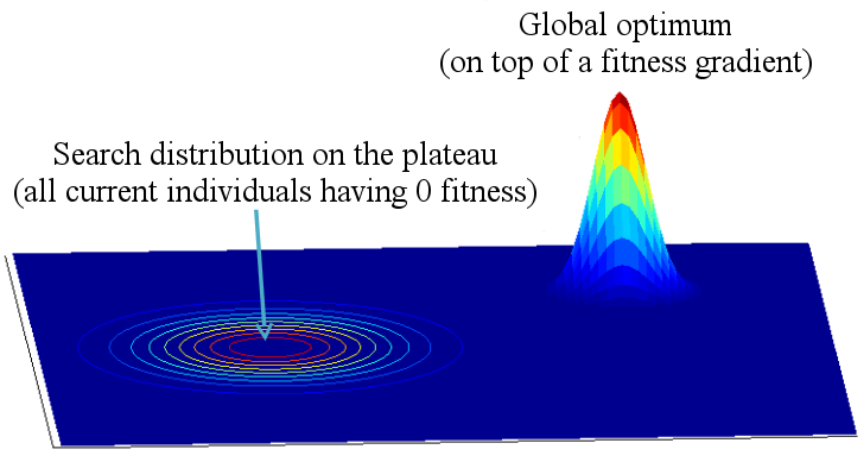


Figure 5.2 – **Fitness plateau**, conceptualized. The fitness of the controllers depends on time to completion, so all individuals unable to complete the task (as common in initial populations) equally receive zero fitness. Such a simple scoring mechanism, common to many applications (especially in reinforcement learning), has the drawback of making the controllers indistinguishable based on performance. The fitness slope leading to the optimum can only be reached through exploration.

$$v_{t+1} = \left[v_t + 0.001\ a_t + g\cos(3p_t)\right] \tag{5.6}$$

$$p_{t+1} = \left[p_t + v_{t+1}\right]\ , \tag{5.7}$$

with time step t, position $p_t \in [-1.2, 0.5]$, velocity $v_t \in [-0.07, 0.07]$, action $a_t \in \{-1, 0, 1\}$, and gravity g. The altitude of the car is thus calculated as $\cos(3p)$. The trial ends with success at time t if the position $p = 0.5$ is reached, with reward $= 1\,000 - t$), or at time 1 000 with failure and reward $= 0$.

Rather than providing direct access to the environment's state as in the classic version, the visual mountain car produces a high-dimensional observation by returning a $[15 \times 30] = 450$ pixels image to the controller. The car is drawn as a color block against white background (Figure 5.4), with coordinates corresponding to a discretized (thus imprecise) approximation of $\langle p, \cos(3p)\rangle$ normalized over the image coordinates.

The original mountain car benchmark was designed to highlight the sensitivity of classical reinforcement learning methods to fitness plateaus. Neuroevolution performs better in this context (to the extent of making the classical mountain car somehow trivial), but has limited applicability to higher-dimensional problems. The visual mountain car instead stresses both points at once, by providing a higher-dimensional problem with a wide initial fitness plateau, with sufficient complexity to highlight the synergy between the online-trained compressor and the evolutionary algorithm's fitness.

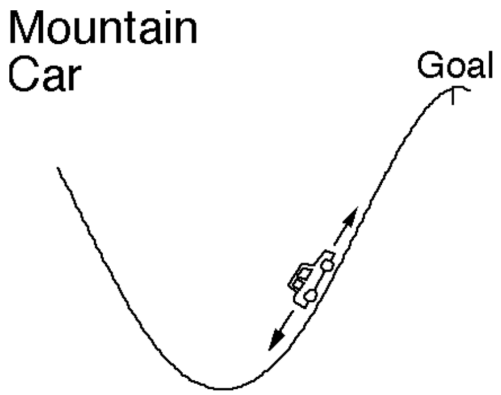


Figure 5.3 – **Original mountain car task.** In the original version, the controller receives both the car's position and velocity as input. Figure courtesy of [Sutton, 1996].

The task is further complexified by the lack of velocity information, both as in speed (the vector's magnitude) and heading (vector sign). The recurrent network needs to compute and maintain internally such crucial information in order to develop the swinging behavior. To reduce the impact of luck on initialization, the car is initially positioned at the bottom of the valley, i.e. the environment is initialized with $p_0 \in [-0.7, -0.3]$, and negligible velocity $v_0 \in [-0.07/4, 0.07/4]$. Finally, to require deeper, multiple swings, the gravity was doubled from $g = -0.0025$ to $g = -0.005$. The low gravity was meant for (and required by) classical RL methods to ensure the task could be solved by chance in a reasonable number of trials, thus booting the learning process. Raising it greatly increases the task's difficulty, as the fitness plateau is much larger (Figure 5.2).

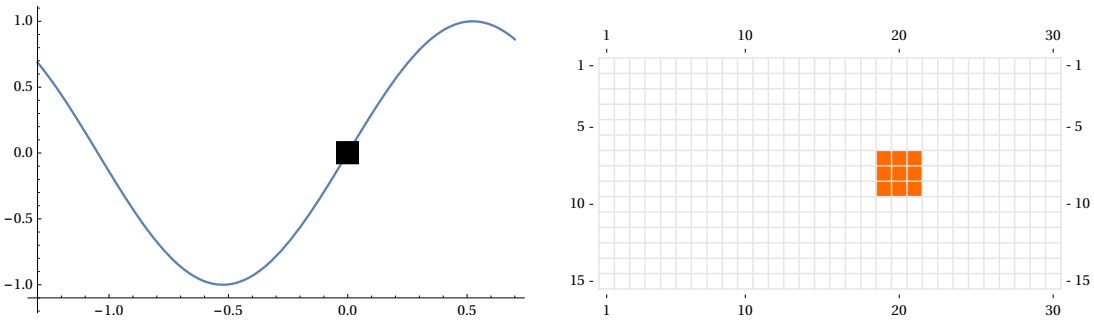
5.5.2 Configurations

In order to clearly distinguish and assess the advantage in boosting the neuroevolution score with intrinsic motivation, four different experiment configurations are evaluated:

1. Random Weight Guessing (RWG; Schmidhuber et al. [2001]): the compressor encoding and training remains as discussed, but the networks are generated with randomly selected (i.i.d.) weights rather than being trained with SNES. The best network found is ultimately returned. RWG tends to perform surprisingly well on low-difficulty tasks, providing a reliable baseline and a measure of the task complexity.

2. Raw images: end-to-end evolution of neural networks using raw pixel input. The compressor is therefore excluded from this setup. The network weight counts is proportionally larger, but no extra layers are added. This provides an insight on the contribution of the compressor encoding over raw data.

3. Base system: the system presented so far, with the inclusion of the compressor, but



(a) (b)

Figure 5.4 – **Visual mountain car task.** (a) Third person perspective of the car (black square) on the slope (cosine function). (b) Observation returned from the environment as a $[15 \times 30]$ matrix, where the background is represented as 0s (white) and the car's approximate location is represented as 1s (color). Both images refer to the car positioned at $p = 0$ on the standard slope $\cos(3p)$, $p \in [-1.2, 0.5]$. No velocity information is available to the controller. The goal is to reach the top of the rightmost peak, at $p = 0.5$.

limited to evaluate individuals based on fitness alone, without the addition of the compressor's novelty signal.

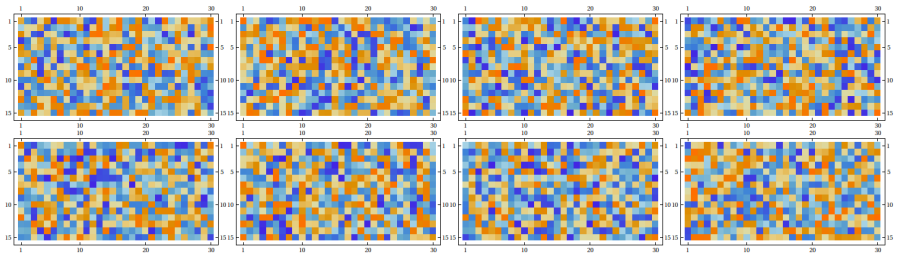
4. Proposed system: the full contribution presented in this work. The compressor encodes the observations, the network decides actions based on the code, the compressor is trained online on a subset of the observations, and the individual score is a combination of the task's cumulative reward and the novelty computed by the compressor.

The evolutionary algorithm was initialized with $\boldsymbol{\mu}$ randomly sampled i.i.d. from $[-0.5, 0.5]$, and variance set to 1 for all parameters. The population size was fixed to 20 samples / individuals, with learning rates $\alpha_{\boldsymbol{\mu}} = \alpha_{\sigma} = 0.35$. Each setup was alloted 35 generations. All coefficients were tested to be robust to small variations. The fitness for experiments including novelty signal was a simple sum of the reward from the environment and the distortion calculated on the most novel image. As the novelty magnitude is smaller than the environment's reward, further normalization was not needed: fitness plateaus assign the same fitness to all individuals, making the novelty signal the unique differentiation anyway. SNES utilizes fitness shaping as seen in Section 2.5, thus ensuring the difference in scale is not penalized in the evolution.

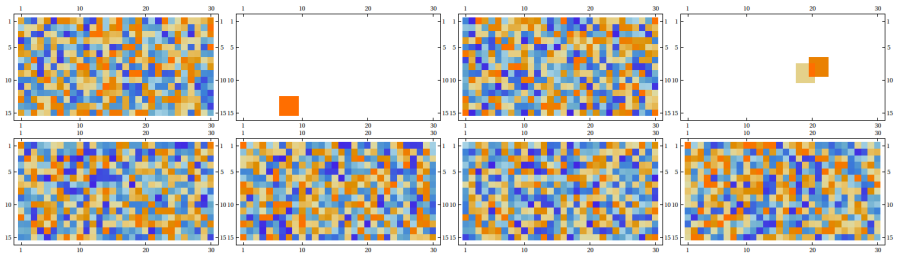
5.5.3 Results

Training the compressor incrementally specializes the centroids, as highlighted in Figure 5.5. Results on the four setups described above (Section 5.5.2) are described in Figure 5.6.

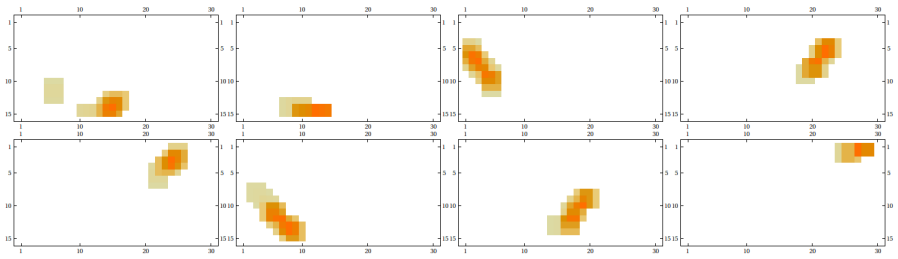
The low performance of Random Weight Guessing (RWG) suggests that the task is of non-trivial complexity. Moreover, Figure 5.6 only reports results of RWG networks trained to use codes from the UL compressor as inputs, because training networks on raw pixel inputs using



(a)



(b)



(c)

Figure 5.5 – **Online VQ centroids.** (a) Random initialization. (b) After training on observations from the first generation. (c) At task end. The latter is capable of recognizing all possible environment states and distinguish them in 8 classes, where similar positions tend to receive the same representation. Action selection still requires memory to distinguish between the forward and backward swinging motions, switching behavior on stalling.

RWG was never successful on the task. Interestingly, the 8-dimensional compressed code is still four times larger than the state vector of the original task (constituted of position and velocity), while at the same time much less informative (at best, the encoding can capture an approximated position).

In order to test the limits of the contribution of the novelty signal to the overall evolution, the gravity was raised from $g = -0.005$ to $g = -0.00545$, while the maximum number of steps was *halved* from 1000 to 500. This makes the task barely solvable at all, lowering the probability of finding a solution by mere chance. Results are presented on the right-hand side in Figure 5.6. Not only the performance of the RWG setup is reduced, but the performance of the one based on SNES is drastically hindered. The setup based on raw pixel images fed to a SNES-trained network was never successful, and has as such been removed from the plot.

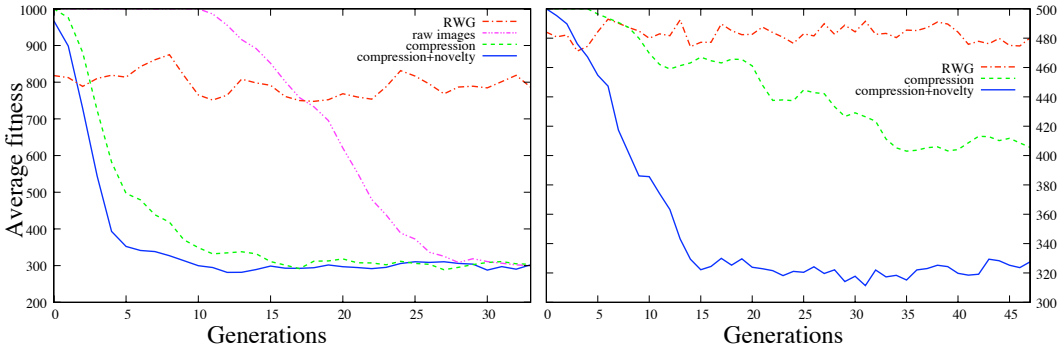


Figure 5.6 – **Experimental performance.** Results are presented as the average number of steps required to solve the visual mountain car task at each generation (smoothed with moving average, window size 3). The plot on the left-hand side shows performance based on the experiment setup as described in Section 5.5.2. Compressing the observations significantly reduces the training time. Intrinsic motivation has a distinctive but minor positive impact on performance. The right-hand side plot corresponds to a harder task with increased gravity and reduced alloted time, as described in Section 5.5.3. This greatly broadens the fitness plateau constituted by individuals unable to solve the task, thus receiving zero reward. The addition of behavioral novelty from the compressor as intrinsic motivation provides a search direction even inside the plateau, greatly boosting the algorithm's performance and reliability.

While the setup of SNES plus compressor – but without novelty – still shows an improving trend, its convergence speed is notably reduced. The setup including the novelty signal instead shows a learning trend comparable to the much simpler settings on the left-hand side. There is in principle no guarantee that exploration alone will overcome an arbitrarily large fitness plateau in limited time; the novelty signal however introduces *a direction of principled exploration* which points the search towards improvement even in presence of fitness plateau. This acts as a sort of "recovery system" for the search, leading away from known sub-optimal volumes (since already explored without the search succeeding) even when a direction towards higher fitness is not available, which increases overall performance. These results are comparable with the work discussed in Chapter 4, where the search was restarted entirely in a new area of the search which was promising from a novelty perspective.

5.6 Discussion

This chapter proposes the use of a compressor to produce a low-dimensional encoding from high-dimensional observations. This is used as input by neural network controllers in a reinforcement learning neuroevolution task. The compressor is trained online on the very images produced by the individuals during their runs, thereby accumulating a history of the interactions with the environment across the evolutionary process so far. This enables it to evaluate a new observation for its novelty, which in turn is tied to the ability of the individual which obtained it to interact with the environment in a novel way, i.e. *behavioral novelty*. Constructing the fitness of the individual from a combination of the environment's reward

and the compressor's novelty score helps overcoming fitness plateaus in the search, improving overall performance.

The method is tested on a new, harder, visual version of the mountain car task set to return high-dimensional *images* from a third-person perspective rather than the usual highly informative and low-dimensional vectors of position and velocity. Different setups were considered, including combinations with random weight guessing and raw pixel input (i.e. skipping the compressor) to highlight the contribution of each part. The final setup includes a very small (3 neurons) single-layer fully-connected recurrent neural network for the controller, Separable Natural Evolution Strategies for the evolutionary algorithm (these two in a classic *neuroevolution* setup), and a novel compressor based on Vector Quantization suited for online learning.

A first set of results shows how the compressor greatly simplifies the task, enabling RWG to reach the goal at all and SNES to be consistently successful, with the novelty signal from the compressor only marginally improving the results. An additional set of experiments, where the environment parameters are set for a harder task pushing the boundaries of feasibility, highlights the true value of the inclusion of the novelty signal. This implementation alone remains resilient to the increase in task complexity, easily surpassing the performance of all other methods.

The next chapter will continue along this line by addressing a problem orders of magnitudes more complex.

6 Complex Domains, Compressors and Shallow Networks

The previous chapter highlights the advantages of observation pre-processing in neuroevolution applications. The results presented refer to a single baseline though, leaving open the question of whether the system could scale to larger, more complex applications, and whether it could maintain the generalization necessary to tackle different tasks with minimal adaptation.

Deep reinforcement learning on Atari games for example already provides results by means of mapping pixel directly to actions (see Chapter 6.2.1); internally, the deep neural network bears the responsibility of both extracting useful information and making decisions based on it. By separating the image processing from decision-making, one could better understand the complexity of each task, as well as potentially find smaller policy representations that are easier for humans to understand and may generalize better.

To this end, this chapter proposes a new method for learning policies and compact state representations separately but simultaneously for policy approximation in reinforcement learning. State representations are generated by an encoder based on two novel algorithms: Increasing Dictionary Vector Quantization makes the encoder capable of growing its dictionary size over time, to address new observations as they appear in an open-end online-learning context; Direct Residuals Sparse Coding encodes observations by disregarding reconstruction error minimization, and aiming instead for highest information inclusion. The encoder selects autonomously (online) observations to train on, in order to maximize code sparsity.

As the dictionary size increases, the encoder produces increasingly larger inputs for the neural network: this is addressed by a variation of the Exponential Natural Evolution Strategies algorithm which adapts its probability distribution dimensionality along the run. The system is tested on a selection of Atari games using tiny neural networks of only 6 to 18 neurons (depending on the game's controls). These are still capable of achieving results comparable—and occasionally superior—to state-of-the-art techniques which use two orders of magnitude more neurons.

6.1 Introduction

In deep reinforcement learning, a large network learns to map complex, high dimensional input (often visual) to actions, in a direct policy approximation. When a giant network with hundreds of thousands of parameters learns a relatively simple task (such as playing Qbert) it stands to reason that only a small part of what is learned is the actual policy. A common understanding is that the network internally learns to extract useful information (features) from the image observation with the first layers by mapping pixels to intermediate representations, allowing the last (few) layer(s) to map these representations to actions. The policy is thus learned at the same time as the intermediate representations, making it almost impossible to study the policy in isolation.

Separating the representation learning from the policy learning allows in principle for higher component specialization, enabling smaller networks dedicated to policy learning to address problems typically tackled by much larger networks. This size difference represents a net performance gain, as larger networks can be devoted to addressing problems of higher complexity. For example, current results on Atari games are achieved using networks of hundreds of neurons; making the same game playable (with comparable performance) by a network k times smaller paves the road to training larger networks on k independent games, using currently available methods and resources.

Separating the policy network from the image parsing also allows to better understand how network complexity contributes to accurately representing the policy. While vision-based tasks are often addressed with very large networks, the learned policies by themselves would in principle not require such high-capacity models. Yet another reason to investigate how to learn smaller policy networks by addressing the image processing with a separate component is that smaller networks may offer better generalization. This phenomenon is well-known from supervised learning, where smaller-capacity models tend to overfit less, but has not been explored much in reinforcement learning.

The key contribution of this chapter is a new method for learning policy and features *simultaneously* but *separately* in a complex reinforcement learning setting. This is achieved through two novel algorithms: Increasing Dictionary Vector Quantization (IDVQ) and Direct Residuals Sparse Coding (DRSC).

IDVQ maintains a dictionary of centroids in the observation space, which can then be used for encoding. The two main differences with standard VQ are that the centroid (i) are trained online by (ii) disregarding reconstruction error. Online training is achieved with the algorithm autonomously selecting images for its training from among the observations it receives to be encoded, obtained by the policies as they interact with the environment. The disregard for reconstruction error comes instead from shifting the focus of the algorithm to the arguably more crucial criterion (from the perspective of the application at hand) of ensuring that all of the information present in the observation is represented in the centroids. This is done by means of constructing new centroids as a residual image from the encoding while ignoring

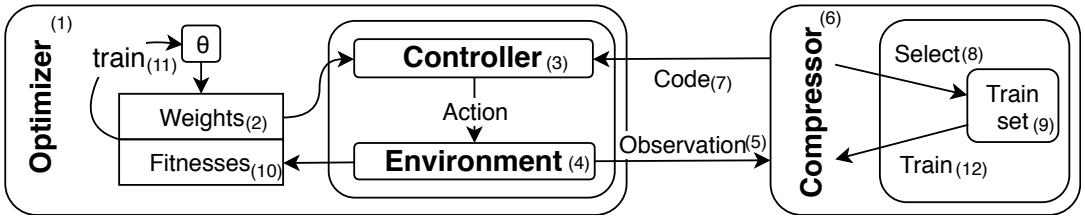


Figure 6.1 – **System diagram.** At each generation the optimizer (1) generates sets of weights (individuals) (2) for the neural network controller (3). Each network is evaluated episodically against the environment (4). At each step the environment sends an observation (5) to an external compressor (6), which produces a compact encoding (7). The network uses that encoding as input. Independently, the compressor selects observations (8) for its training set (9). At the end of the episode, the environment returns the fitness (cumulative reward; 10) to the optimizer for training (neuroevolution; 11). Compressor training (12) takes place in between generations.

reconstruction artifacts. See Section 6.3.2 for further discussion.

The dictionary trained by IDVQ is then used by DRSC to produce a compact code for each observation. This code will be used in turn by the neural network (policy) as input to select the next action. The code is a binary string: a value of '1' indicates that the corresponding centroid contains information also present in the image, and a limited number of centroids are used to represent the totality of the information.

As the training progresses and more sophisticated policies are learned, complex interactions with the environment result in increasingly novel observations; the dictionary reflects this by *growing in size*, including centroids that account for newly discovered features. A larger dictionary corresponds to a larger code, forcing the neural network to grow in input size. This is handled using a specialized version of Exponential Natural Evolution Strategy which adapts the dimensionality of the underlying multivariate Gaussian.

With the goal of minimizing the network size while maintaining comparable scores, experimental results show that this approach can effectively learn both components simultaneously, achieving state-of-the-art performance on several ALE games while using a neural network of *only 6 to 18 neurons*, i.e. *two orders of magnitude smaller* than any known previous implementation. This research paves the road for training deep networks entirely dedicated to policy approximation, aiming at problems of unprecedented complexity.

6.2 Related work

6.2.1 Video games as AI benchmarks

Games are useful as AI benchmarks as they are often designed to challenge human cognitive capacities. Board games such as Chess and Go have been used as AI benchmarks since the

inception of artificial intelligence research, and have been increasingly used for testing and developing AI methods [Yannakakis and Togelius, 2018]. Though various video game-based AI competitions and frameworks exist, the introduction of the Arcade Learning Environment (ALE) did much to catalyze the use of arcade games as AI benchmarks [Bellemare et al., 2013].

ALE is based on an emulation of the Atari 2600, the first widely available video game console with exchangeable games, released in 1977. This was a very limited piece of hardware: 128 bytes of RAM, up to 4 kilobytes of ROM per games, no video memory, and an 8-bit processor operating at less than 2 MHz. The limitations of the original game console mean that the games are visually and thematically simple. Most ALE games feature two-dimensional movement and rules mostly triggered by sprite intersection. In the most common setup, the raw pixel output of the ALE framework is used as inputs to a neural network, and the outputs are interpreted as commands for playing the game. No forward model is available, so planning algorithms cannot be used. Using this setup, Mnih et al. reached above human level results on a majority of 57 Atari games that come with the standard distribution of ALE [Mnih et al., 2015]. Since then, a number of improvements have been suggested that have improved game-playing strength on most of these games [Hessel et al., 2017].

6.2.2 Neuroevolution

Neuroevolution refers to the use of evolutionary algorithms to train neural networks [Floreano et al., 2008, Yao, 1999, Igel, 2003, Risi and Togelius, 2017]. Typically, this means training the connection weights of a fixed-topology neural network, though some algorithms are also capable of evolving the topology at the same time as the weights [Stanley and Miikkulainen, 2002].

When using neuroevolution for reinforcement learning, a key difference is that the network is only trained in between episodes, rather than at every frame or time step. In other words, learning happens between episodes rather than during episodes; this has been called *phylogenetic* rather than *ontogenetic* reinforcement learning [Togelius et al., 2009]. While it could be argued that evolutionary reinforcement learning should learn more slowly than ontogenetic approaches such as Q-learning, as the network is updated more rarely and based on more aggregated information, the direct policy search performed by evolutionary algorithms allows in principle for a freer movement in policy space. Empirically, neuroevolution has been found to reach state-of-the-art performance on reinforcement learning problems which can be solved with small neural networks [Gomez et al., 2008] and to reach close to state-of-the-art performance on games in the ALE benchmark played with visual input [Salimans et al., 2017, Chrabaszcz et al., 2018]. In general, neuroevolution performs worse in high-dimensional search spaces such as induced by deep neural networks, but there have also been recent results where genetic algorithms have been shown to be competitive with gradient descent for training deep networks for reinforcement learning [Such et al., 2017]. Neuroevolution has also been found to learn high-performing strategies for a number of other more modern

games including racing games and first-person shooters, though using human-constructed features [Risi and Togelius, 2017].

For training the weights of a neural network only, modern variants of evolution strategies can be used. The Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [Hansen and Ostermeier, 2001] represents the population implicitly as a distribution of possible search points; it is very effective at training small-size networks in reinforcement learning settings [Igel, 2003]. Another high-performing development of evolution strategies, introduced in Chapter 2.5, is the Natural Evolution Strategies (NES) family of algorithms [Wierstra et al., 2014b]. While both CMA and NES suffer from having a number of parameters required for evolution growing superlinearly with the size of the neural network, there are versions that overcome this problem, as discussed in Chapter 7.

6.2.3 Compressed representation in reinforcement learning

The high dimensionality of visual input is a problem not only for evolutionary methods, but generally for learning technique. The origin of the success of deep learning can be traced to how deep convolutional networks handle large dimensional inputs; up until a few years ago, reinforcement learning generally relied on low-dimensional features, either by using intrinsically low-dimensional sensors (such as infrared or laser range-finders) or by using hard-coded computer vision techniques to extract low-dimensional state representations from image data. Such hard mappings however do not lend themselves to generalization; in order to create a more general reinforcement learning method, the mapping must be automatically constructed or learned.

Several approaches have been proposed in that sense in reinforcement learning. Some of them rely on neural networks, in particular on various forms of autoencoders [Alvernaz and Togelius, 2017, Ha and Schmidhuber, 2018]. An alternative is to use compressors as seen in Chapter 5, where a number of prototype vectors are found and each vector is used as a feature detector–the value of that feature being the similarity between the actual high-dimensional input and the vector, similar to a radial basis function network.

6.3 Method

The system proposed is divided into four main components: (i) the Environment is an Atari game, taking actions and providing observations; (ii) the Compressor extracts a low-dimensional code from the observation, while being trained online with the rest of the system; (iii) the Controller is the policy approximizer, i.e. the neural network; finally (iv) the Optimizer is the learning algorithm, improving the performance of the network over time, in this case an Evolution Strategy. Each component is described in more detail below.

6.3.1 Environment

The testing environment of choice is the Arcade Learning Environment (ALE), interfaced through the OpenAI Gym framework [Brockman et al., 2016]. As discussed above, ALE is built on top of an emulator of the Atari 2600, with all the limitations of that console. In keeping with ALE conventions, the observation consists of a $[210 \times 180 \times 3]$ tensor, representing the RGB pixels of the screen input. The output of the network is interpreted (using one-hot encoding) as one of 18 discrete actions, representing the potential inputs from the joystick. The frame-skipping is fixed at 5 by following each action with 4 `NOOP` commands.

6.3.2 Compressor

The role of the compressor is to provide a compact representation for each observation coming from the environment, enabling the neural network to entirely focus on decision making. This is done through unsupervised learning on the very same observations that are obtained by the network interacting with the environment, in an *online learning* fashion.

This work address such limitations through a new algorithm based on Vector Quantization (VQ), named Increasing Dictionary VQ, coupled with a new Sparse Coding (SC) method named Direct Residuals SC. Together they aim at supporting the study of the spaces of observations and features, while offering top performance for online learning. The only prior work using unsupervised learning as a pre-processor for neuroevolution is Cuccu et al. [2011b], Alvernaz and Togelius [2017]. The following sections will derive IDVQ+DRSC starting from the vanilla VQ, explaining the design choices which led to these algorithms

Vanilla vector quantization

The standard VQ algorithm [Gray, 1984] maintains a fixed-size set of *centroids* (called a *dictionary*) in the space of observations. A *target vector* is approximated by a linear combination of the dictionary and a set of coefficients (the *code*), corresponding to a *similarity* between the target vector and each centroid. The dictionary is adapted by minimizing reconstruction error during training.

Applications to *online* reinforcement learning however present a few limitations. Additional training data is not only unavailable until late stages, but is also only accessible if obtained by individuals through interaction with the environment. Take for example an Atari game with different enemies in each level: observing a second-level enemy depends on the ability to solve the first level of the game, requiring in turn the compressor to recognize the first-level enemies. A successful run should thereby alternate improving the dictionary with improving the candidate solutions: at any stage, the dictionary should provide an encoding supporting the development of sophisticated behavior.

In online learning though, two opposite needs are in play: on one hand, the centroids need

Algorithm 3 IDVQ

Inputs:
$\mathscr{X}$: training set, $X \in \mathscr{X}$
$\mathscr{D}$: current dictionary
δ: minimal aggregated residual for inclusion
Initialize:
$\mathscr{D} \leftarrow \varnothing$ ▷ dictionary initialized empty
for X in $\mathscr{X}$ **do**
$\mathscr{P} \leftarrow X$ ▷ residual information to encode
$\mathbf{c} \leftarrow DRSC(X, \mathscr{D}, \epsilon, \Omega)$ ▷ ϵ and Ω given
$\hat{\mathscr{P}} \leftarrow \mathbf{c}\mathscr{D}$
$\mathscr{R} \leftarrow \mathscr{P} - \hat{\mathscr{P}}$
$r_i \leftarrow \max(0, r_i),\ \forall r_i \in \mathscr{R}$ ▷ remove artifacts
if $\Sigma|\mathscr{R}| > \delta$ **then**
$\mathscr{D} \ll \mathscr{R}$ ▷ append $\mathscr{R}$ to $\mathscr{D}$
return $\mathscr{D}$

to be trained in order to provide a useful and consistent code; on the other hand, late stage training on novel observations requires at least some centroids to be preserved untrained. Comparing to vanilla VQ, random centroids cannot be used for the code. The work in Chapter 5.4.1 is only able to do so because the observations are already very simplistic.

Uniformly drawing from the space of *all possible images* generates in principle an enormously sparse spread w.r.t. the small sub-volume of an Atari game's image. The similarity of a random centroid to any such image will be about the same: using random centroids as the dictionary consequently produces an *almost constant* code for any image from a same game[1]. Image differentiation is relegated to the least significant digits, making it suboptimal as a neural network input. Directly addressing this problem naturally calls for starting with a smaller dictionary size, and increasing it at later stages as new observations call for it.

Increasing Dictionary VQ

Increasing Dictionary VQ (IDVQ, Algorithm 3) is a new compressor based on VQ which automatically increases the size of its dictionary over successive training iterations, specifically tailored for online learning. Rather than having a fixed-size dictionary, IDVQ starts with an *empty* dictionary, thus requiring *no initialization*, and adds new centroids as the learning progresses.

This is done by building new centroids from the positive part of the reconstruction error, which corresponds to the information from the original image (rescaled between 0 and 1) which is not reconstructed by the current encoding (see Algorithm 3). Growth in dictionary size is regulated by a threshold δ, indicating the minimal aggregated residual considered to

[1] This has also been empirically verified in earlier iterations of this work

Algorithm 4 DRSC

Inputs:
X: vector to encode (observation)
$\mathscr{D}$: dictionary trained with IDVQ
ϵ: minimal aggregated residual loss
Ω: maximum nonzero elements in the code

Initialize:
$\mathscr{P} \leftarrow X$ ▷ residual information to encode
$\mathbf{c} \leftarrow \vec{0}$ ▷ output code
$\omega \leftarrow 0$ ▷ non-zero elements in the code

while $\Sigma|\mathscr{P}| > \epsilon$ and $\omega < \Omega$ **do**
$\mathscr{S} \leftarrow \text{sim}(\mathscr{P}, \mathscr{d}_i), \forall \mathscr{d}_i \in \mathscr{D}$
msc $\leftarrow$ index of max $(\mathscr{S})$
$\mathbf{c}_{\text{msc}} \leftarrow 1$ ▷ $\mathbf{c} = [\mathbf{c}_1 \dots \mathbf{c}_n]$
$\omega \leftarrow \omega + 1$
$\mathscr{P} \leftarrow \mathscr{P} - \mathscr{d}_{\text{msc}}$ ▷ $\mathscr{D} = [\mathscr{d}_1 \dots \mathscr{d}_n]$
$\rho_i \leftarrow \max(0, \rho_i),\ \forall \rho_i \in \mathscr{P}$

return c

be a meaningful addition. The training set is built by uniformly sampling the observations obtained by all individuals in a generation.

Centroids added to the dictionary are not further refined. This is in line with the goal of image differentiation rather than minimizing reconstruction error: each centroid is purposely constructed to represents one particular feature, which was found in an actual observation and was not available in the dictionary before.

Growing the dictionary size however alters the code size, and thus the neural network input size. This requires careful updates in both the controller and the optimizer, as addressed in Sections 6.3.3 and 6.3.4 respectively.

Direct Residuals Sparse Coding

The performance of dictionary-based algorithms depends more on the encoding than on dictionary training – to the point where the best performing algorithms have but a marginal improvement in performance when using sophisticatedly trained centroids versus random training samples [Coates and Ng, 2011]. Sparse coding algorithms have been shown in recent years to consistently surpass the performance of other encoding algorithms in compression and reconstruction tasks [Mairal et al., 2014, Zhang et al., 2015].

The reconstruction is commonly a linear combination of the dictionary using the code as coefficients. Most implementations utilize a dot product for this task; while conceptually elegant, an avoidable performance overhead comes from multiple (most) products having null coefficients. On top of that, state-of-the-art encoding algorithms often take an iterative

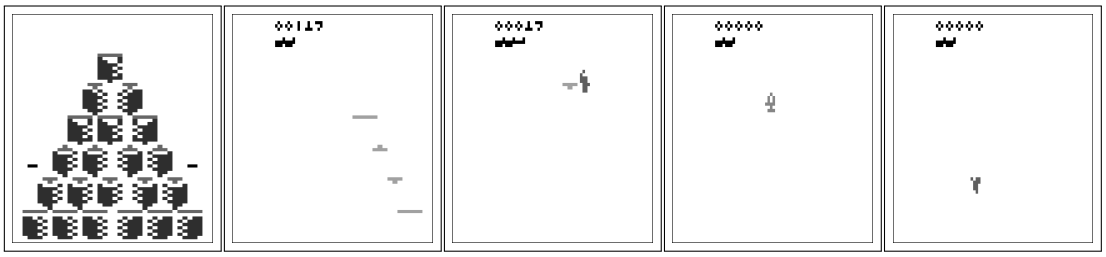


Figure 6.2 – **Trained centroids.** A few centroids trained with IDVQ during a run of the game Qbert. Notice how the first captures the initial state of the game (backgroud), while the others build features as subsequent residuals: lit cubes, avatar and enemy. Colors are inverted for printing purposes.

approach [Mallat and Zhang, 1993, Pati et al., 1993] where (i) few centroids are selected, (ii) a corresponding code is built, (iii) the reconstruction error is computed, and (iv) the algorithm loops back to (i) to try alternative combinations of centroids. Sparse coding algorithms usually use an over-complete dictionary and enforce sparsity using the ℓ_0 or ℓ_1 of the code as a regularization term, further increasing the toll on performance. In the proposed context though, the time taken by the encoding proportionally reduces the time alloted for decision making. Prioritizing *distinction over precision* requires an overhaul of the objective function from the ground up, as the code will be used to recognize a state and issuing a consequent action rather than reconstructing the original input.

Direct Residuals Sparse Coding (DRSC, Algorithm 4) addresses these issues: a novel sparse coding algorithm specifically tailored to produce highly differentiating encoding in a minimum amount of time. Its key characteristics are: (i) it uses compact dictionaries rather than comprehensive ones, based on atomic centroids constructed as *residual images* from the reconstruction of training vectors; (ii) it produces binary encodings, reducing the reconstruction process to an unweighted sum over the centroids selected by the code's nonzero coefficients; (iii) it produces the code in a single pass based on the reconstruction residuals, and terminates early after a small number of centroids are selected. Leveraging IDVQ, where the dictionary is designed to represent most of the observation's content through consecutive residuals, DRSC iteratively selects the centroid that most closely resembles the remaining information in the encoded observation.

Step-by-step breakdown

Increasing Dictionary VQ is used to train a dictionary, used by Direct Residuals SC to encode (compress, extract features from) an observation (image). To understand how these algorithms work together, consider a working starting dictionary and see how DRSC produces an encoding.

The initialization includes two steps: the code, as an arrays of zeros with the same size as the dictionary, and the *residual information* still needing encoding, initially the whole original

image. The algorithm then loops to select centroids to add to the encoding, based on how much of the residual information can they encode. To select the most similar centroid, the algorithm computes the differences between the residual information and each centroid in the dictionary, aggregating each of these differences by summing all values. The centroid with the smallest aggregated difference is thereby the most similar to the residual information, and is chosen to be included in the encoding. The corresponding bit in the binary code is then flipped to '1', and the residual information is updated by subtracting the new centroid.

The sign of the values in the updated residual information (old residual minus new centroid, the order matters) are now significant: (i) values equal to zero mean a perfect correspondence between the pixel information in the old residual and the corresponding value in the new centroid; (ii) positive values correspond to information that was present in the old residual but not covered by the new centroid; (iii) negative values correspond to information present in the new centroid, but absent (or of smaller magnitude) in the old residual. This is crucial as the goal of the algorithm is to fully represent the totality of the original information, and from this end is free to disregard *reconstruction artifacts* as found in (iii).

Encoding algorithms typically do not distinguish the sign of these values: aiming at minimizing reconstruction error, they focus on the error's magnitude, not on its origin. DRSC is instead focused solely on representing all the information initially present in the image. The artifacts in the negative values are disregarded by setting them to zeros. The result is a residual image of information present in the original image but not yet captured by the reconstruction.

The algorithm then keeps looping and adding centroids until the (aggregated) residual information is lower than a threshold, corresponding to an arbitrary precision in capturing the information in the original image. To enforce sparsity, a secondary stopping criterion for the encoding loop is when too many centroids are added to the code, based on another threshold. Images with high residual information after encoding are prime candidates for compressor training.

The dictionary is trained with IDVQ by adding new centroids to minimize leftover residual information in the encoding. The training begins by selecting an image from the training set and encoding it with DRSC, producing the binary code as described above. A dot product between the code and the dictionary (corresponding to summing the selected centroids since the code is binary) produces a reconstruction of the original image, similarly to other VQ-based algorithms.

The difference between the training image and the reconstruction then produces a reconstruction error (image), where the sign of the values once more corresponds to their origin: positive values are leftover information from the image which is not encoded in the reconstruction, while negative values are reconstruction artifacts with no relation to the original image. This reconstruction error image is then aggregated (with a sum) to estimate the quantity of information missed by the encoding. If it is above a given threshold, a new centroid should be added to the dictionary to enable DRSC to make a more precise reconstruction. The new

centroid is the reconstruction error itself, as it will by definition enable a perfect capture of the leftover information of the current encoding.

6.3.3 Controller

The controller for all experiments is a single-layer fully-connected recurrent neural network (RNN). Each neuron receives the following inputs through weighted connections: the inputs to the network, the output of all neurons from the previous activation (initially zeros), and a constant bias (always set to 1). The number of inputs is equal at any given point in time to the size of the code coming from the *compressor*. As the compressor's dictionary grows in size, so does the network's input. In order to ensure continuity in training (i.e. the change needs to be transparent to the training algorithm), it is necessary to define an invariance across this change, where the network with expanded weights is equivalent to the previous one. This is done by setting the weights of all new connections to zero, making the new network mathematically equivalent to the previous one, as any input on the new connections cancels out. The same principle can be ported to any neural network application.

The number of neurons in the output layer is kept equal to the dimensionality of the action space for each game, as defined by the ALE simulator. This is as low as 6 in some games, and 18 at most. Actions are selected deterministically in correspondence to the maximum activation. No hidden layer nor extra neurons were used in any of the presented results. The increase in dimensionality in the input connections' weights corresponds to a growth in the parameter vector of the optimizer, as described below in Section 6.3.4.

6.3.4 Optimizer

The optimizer used in the experiments is a variation of Exponential Natural Evolution Strategy(XNES, introduced in Chapter 2.5.2. This section derives a version tailored for evolving networks with dynamic varying size.

Since the parameters are interpreted as network weights in *direct encoding* neuroevolution, changes in the network structure need to be reflected by the optimizer in order for future samples to include the new weights. Particularly, the multivariate Gaussian acquires new dimensions: θ should be updated keeping into account the order in which the coefficients of the distribution samples are inserted in the network topology.

Section 6.3.3 explains how the network update is carried through by initializing the new weights to zeros. In order to respect the network's invariance, the expected value of the distribution (μ) for the new dimension should be zero. As for Σ, the values for the new rows and columns need to be inserted in correspondence to the new dimensions, knowing that (i) the new weights did not vary so far in relation to the others (as they were equivalent to being fixed to zero until now), and that (ii) everything learned by the algorithm until now was based on the samples having always zeros in these positions. So Σ must have for all new dimensions

Table 6.1 – **Game scores.** Scores on a sample of Atari games (sorted alphabetically), compared to results from HyperNeat [Hausknecht et al., 2014] and OpenAI ES [Salimans et al., 2017]. Results from GA (1B) [Such et al., 2017] and NSRA-ES [Conti et al., 2017] are also provided (though the intersection between games sets is minimal) to include work aimed at expanding the network size, rather than shrinking it. All methods were trained from scratch on raw pixel input but for NSRA-ES, which uses a compact state representation read from the simulated Atari RAM. Column '*#n*' indicates how many neurons were used in this work in a single layer (output) for each game. The number of neurons corresponds to the number of available actions in each game, i.e. no neurons are added for performance purpose.

Game	HyperNeat	OpenAI ES	GA (1B)	NSRA-ES	**IDVQ+DRSC+XNES**	#n
DemonAttack	3590	1166.5	-	-	325	6
FishingDerby	-49	-49	-	-	-10	18
Frostbite	2260	370	4536	3785	300	18
Kangaroo	800	11200	3790		1200	18
NameThisGame	6742	4503	-		920	6
Phoenix	1762	4041	-		4600	8
Qbert	695	147.5	-	1350	1250	6
Seaquest	716	1390	798	960	320	18
SpaceInvaders	1251	678.5	-	-	830	6
TimePilot	7340	4970	-	-	4600	10

Table 6.2 – **Results.** The presented method achieves comparable scores (sometimes better) using up to *two orders of magnitude* less neurons, and no hidden layers. The proposed feature extraction algorithm IDVQ+DRSC is simple enough (using basic, linear operations) to be arguably unable to contribute to the decision making process in a sensible manner (see Section 6.3.2). This implies that the tiny network trained on decision making alone is of sufficient complexity to learn a successful policy, potentially prompting for reconsidering the actual complexity of this standard benchmark. The following numbers refer to networks for games with the largest action set (18). See Table6.1 for the actual number of neurons used in the output layer for each game.

	HyperNeat	OpenAI ES	GA (1B)	NSRA-ES	**IDVQ+DRSC+XNES**
# neurons	~3034	~650	~650	~650	**~18**
# hidden layers	2	3	3	3	**0**
# connections	~906k	~436k	~436k	~436k	**~3k**

(i) zeros covariance and (ii) arbitrarily small variance (diagonal), only in order to bootstrap the search along these new dimensions.

Take for example a one-neuron feed-forward network with 2 inputs plus bias, totaling 3 weights. A function mapping the optimizer's parameters to the weights in the network structure (i.e. the *genotype to phenotype* function) will first fill the values of all input connections, then all bias connections. Extending the input size to 4 requires the optimizer to consider two more

weights before filling in the bias:

$$\mu = \begin{bmatrix} \mu_1 & \mu_2 & \mu_b \end{bmatrix} \rightarrow \begin{bmatrix} \mu_1 & \mu_2 & 0 & 0 & \mu_b \end{bmatrix}$$

$$\Sigma = \begin{bmatrix} \sigma_1^2 & c_{12} & c_{1b} \\ c_{21} & \sigma_2^2 & c_{2b} \\ c_{b1} & c_{b2} & \sigma_b^2 \end{bmatrix} \rightarrow \begin{bmatrix} \sigma_1^2 & c_{12} & 0 & 0 & c_{1b} \\ c_{21} & \sigma_2^2 & 0 & 0 & c_{2b} \\ 0 & 0 & \epsilon & 0 & 0 \\ 0 & 0 & 0 & \epsilon & 0 \\ c_{b1} & c_{b2} & 0 & 0 & \sigma_b^2 \end{bmatrix}$$

with c_{ij} being the covariance between parameters i and j, σ_k^2 the variance on parameter k, and ϵ being arbitrarily small (0.0001 here). The complexity of this step of course increases considerably with more sophisticated mappings, for example when accounting for recurrent connections and multiple neurons, but the basic idea stays the same. The evolution can pick up from this point on as if simply resuming, and learn how the new parameters influence the fitness.

6.4 Experimental setup

The experimental setup further highlights the performance gain achieved, and is thus crucial to properly understand the results presented in the next section:

- All experiments were run on a single machine, using a 32-core Intel(R) Xeon(R) E5-2620 at 2.10GHz, with only 3GB of ram per core (including the Atari simulator and Python wrapper).
- The maximum run length on all games is capped to 200 interactions, meaning the agents are alloted a mere $1'000$ frames, given a constant frameskip of 5. This was done to limit the run time, but in most games longer runs correspond to higher scores.
- Population size and learning rates are dynamically adjusted based on the number of parameters, based on the XNES minimal population size and default learning rate [Glasmachers et al., 2010]. The population size is rescaled by 1.5 and the learning rate by 0.5. In all runs on all games, the population size is between 18 and 42, again very limited in order to optimize run time on the available hardware.
- The dictionary growth is roughly controlled by δ (see Algorithm 3), but depends on the graphics of each game. The average dictionary size by the end of the run is around 30-50 centroids, but games with many small moving parts tend to grow over 100. In such games there seems to be direct correlation between higher dictionary size and performance, but the reference machine performed poorly over 150 centroids. Numbers close to $\delta = 0.005$ are robust in the setup across all games.
- Graphics resolution is reduced from $[210 \times 180 \times 3]$ to $[70 \times 80]$, averaging the color channels to obtain a grayscale image. This also contributes to lower run times.

- Every individual is evaluated 5 times to reduce fitness variance.
- Experiments are allotted a mere 100 generations, which averages to 2 to 3 hours of run time on the reference machine.

These computational restrictions are *extremely tight* compared to what is typically used in studies utilizing the ALE framework. Limited experimentation indicates that relaxing any of them, i.e. by accessing the kind of hardware usually dedicated to modern deep learning, consistently improves the results on the presented games. The full implementation is available open-source on my GitHub[2].

6.5 Results

The goal of this work is not to propose a new generic feature extractor for Atari games, nor a novel approach to beat the best scores from the literature. The declared goal is to show that dividing feature extraction from decision making enables tackling hard problems with minimal resources and simplistic methods, and that the deep networks typically dedicated to this task can be substituted for simple encoders and tiny networks while maintaining comparable performance. Table 6.2 emphasizes such findings in this regard.

Under these assumptions, Table 6.1 presents comparative results over a set of 10 Atari games from the hundreds available on the ALE simulator. This selection is the result of the following filtering steps: (i) games available through the OpenAI Gym; (ii) games with the same observation resolution of $[210, 160]$ (simply for implementation purposes); (iii) games not involving 3D perspective (to simplify the feature extractor). The resulting list was further narrowed down due to hardware and runtime limitations. A broader selection of games would support a broader applicability of the presented specialized setup; this work on the other hand aims at highlighting that a simple setup is indeed able to play Atari games with competitive results.

Results on each game differ depending on the hyperparameter setup. To offer a more direct comparison, the same settings as described above are used for all games, rather than specializing the parameters for each game. Some games performed well with these parameters (e.g. Phoenix); others feature many small moving parts in the observations, which would require a larger number of centroids for a proper encoding (e.g. Name This Game, Kangaroo); still others have complex dynamics, difficult to learn with such tiny networks (e.g. Demon Attack, Seaquest).

The resulting scores are mainly compared with two recent publications that offer a broad set of results across Atari games on comparable settings, namely Salimans et al. [2017] and Hausknecht et al. [2014]. The list of games and correspondent results are available in Table 6.1. Notably, the setup achieves high scores on *Qbert*, arguably one of the harder games for its requirement of strategic planning.

[2] https://github.com/giuse/DNE

The resulting scores are compared with recent papers that offer a broad set of results across Atari games on comparable settings, namely Salimans et al. [2017], Such et al. [2017], Conti et al. [2017], Hausknecht et al. [2014]. The list of selected games and correspondent results are available in Table 6.1. Notably, the proposed setup achieves high scores on *Qbert*, arguably one of the harder games for its requirement of strategic planning.

The real results of the paper however are highlighted in Table 6.2, which compares the number of neurons, hidden layers and total connections utilized by each approach. The proposed setup uses up to two order of magnitude less neurons, two orders of magnitude less connections, and is the only one using only one layer (no hidden). connections.

6.6 Discussion

This chapter presented a method to address complex learning tasks such as learning to play Atari games by decoupling policy learning from feature construction, learning them independently but simultaneously to further specializes each role. Features are extracted from raw pixel observations coming from the game using a novel and efficient sparse coding algorithm named Direct Residual Sparse Coding. The resulting compact code is based on a dictionary trained online with yet another new algorithm called Increasing Dictionary Vector Quantization, which uses the observations obtained by the networks' interactions with the environment as the policy search progresses. Finally, tiny neural networks are evolved to decide actions based on the encoded observations, to achieving results comparable with the deep neural networks typically used for these problems while being *two orders of magnitude* smaller.

This work shows how a relatively simple and efficient feature extraction method, which counter-intuitively does not use reconstruction error for training, can effectively extract meaningful features from a range of different games. The implication is that feature extraction on some Atari games is not as complex as often considered. On top of that, the neural network trained for policy approximation is also very small in size, showing that the decision making itself can be done by relatively simple functions.

The method is empirically evaluated on a set of well-known Atari games using the ALE benchmark. Tight performance restrictions are posed on these evaluations, which can run on common personal computing hardware as opposed to the large server farms often used for deep reinforcement learning research. The source code is open sourced for further reproducibility.

The game scores are in line with the state of the art in neuroevolution, while using but a minimal fraction of the computational resources usually devoted to this task. One goal of this work is to clear the way for new approaches to learning, and to call into question a certain orthodoxy in deep reinforcement learning, namely that image processing and policy should be learned together. Future work will address training deep networks entirely dedicated to

policy learning, capable in principle of scaling to problems of unprecedented complexity. To this end it is crucial to design EAs capable of graciously scaling their performance with the size of the network, as discussed in the next chapter.

7 Scaling Towards Large Networks

As seen in Chapter 2.5, Natural Evolution Strategies is a family of evolutionary algorithms which maintain a probability distribution over the search space instead of the classic explicit population. They have high performance in convergence speed, but their applicability is limited by their heavy computational requirements.

Trade-offs have been investigated, such as XNES vs. SNES (see Chapter 2.5: the former maintains full covariance information (fast convergence, slow execution), the latter with diagonal covariance (slow convergence, fast execution). Depending on the task at hand however, a finer control over this trade-off could in principle improve performance.

This chapter proposes *Block Diagonal NES* (BD-NES), a novel algorithm of the NES family which enables users to customize the trade-off depending on the application. Grouping together parameters with expected high variance, while ignoring inter-group covariance, gives a *block-diagonal* shape to the distribution's covariance matrix Σ, hence the name. Moreover, BD-NES generalizes both XNES and SNES (see Chapter 2.5.2), as full covariance information corresponds to having 1 block of size n, while a diagonal covariance matrix corresponds to n blocks of size 1.

This algorithm is tested on a novel benchmark crossing the Octopus Arm and the Acrobot, two widely common benchmarks; integrating both greatly increases the task complexity. Neural network controllers trained with BD-NES achieve higher performance than using SNES, while being too large to train using XNES.

7.1 Introduction

Natural Evolution Strategies, discussed in Chapter 2.5, is a family of evolutionary algorithms where the population is maintained implicitly, through a probability distribution over the parameters' space. The distribution's parameters are updated based on the *natural* gradient, providing state-of-the-art performance. Algorithms of the family have been successfully applied to evolving neural networks (*neuroevolution*) for policy approximation in a variety of

reinforcement learning problems [Schaul et al., 2011, Yi et al., 2009, Cuccu et al., 2011b].

Two algorithms in particular, both based on Gaussian distributions, have seen wider adoption (see Chapter 2.5.2). *XNES* [Glasmachers et al., 2010] maintains a standard multivariate Gaussian as denoted by its mean vector and covariance matrix (similarly to CMA-ES [Hansen and Ostermeier, 2001]); *SNES* [Schaul et al., 2011] on the other hand constraints its search to distributions which can be described using a *diagonal* covariance matrix, i.e. where all search parameters are considered *separable* (i.e. independent). These offer a trade-off between what are basically two different kinds of speed:

- XNES has high *convergence speed*, meaning it produces higher quality solutions in fewer iterations. This comes at a cost of each iteration taking a longer time to execute.
- SNES instead has high *execution speed*, implying that each execution is done in a shorter wall-clock time. The trade-off here is the requirement of far more iterations than XNES to produce solutions of comparable quality, as the convergence is slow.

XNES thus constitutes the better choice in all cases where the user can afford the wall-clock time for its execution. Unfortunately its computational performance is cubic on the number of parameters $\mathcal{O}(p^3)$, which limits its applicability to problems with few thousands parameters – and this, on modern hardware. SNES instead, while slower on per-iteration convergence, offers *linear* performance $\mathcal{O}(p)$, thus scaling to problems with hundreds of thousands of parameters.

Most problems however include at least some degree of inter-parameter correlation. For neuroevolution applications in particular, weights of connections entering a same neuron contribute together to the neuron's activation: updates should be proportional across all weights, rather than independent. While not precluding *a priori* the possibility to converge to the same solutions as XNES, missing those relationships means that SNES will typically require orders-of-magnitude more generations to reach comparable results. The ability to fine-tune the balance between execution speed and convergence speed based on the problem specifications could be expected in principle to improve overall performance.

This chapter leverages this intuition by introducing a novel algorithm of the NES family, which generalizes both XNES and SNES, by maintaining a *block diagonal* covariance matrix, namely Block Diagonal NES (BD-NES). The user selects which subset of the parameters are expected to have higher correlation, thus injecting human domain knowledge in an otherwise black-box setting.

The rest of the chapter will cover the derivation for BD-NES (Section 7.2), comparative results against SNES on problems beyond the applicability of XNES (Section 7.3), and a discussion of these results and implications (Section 7.4).

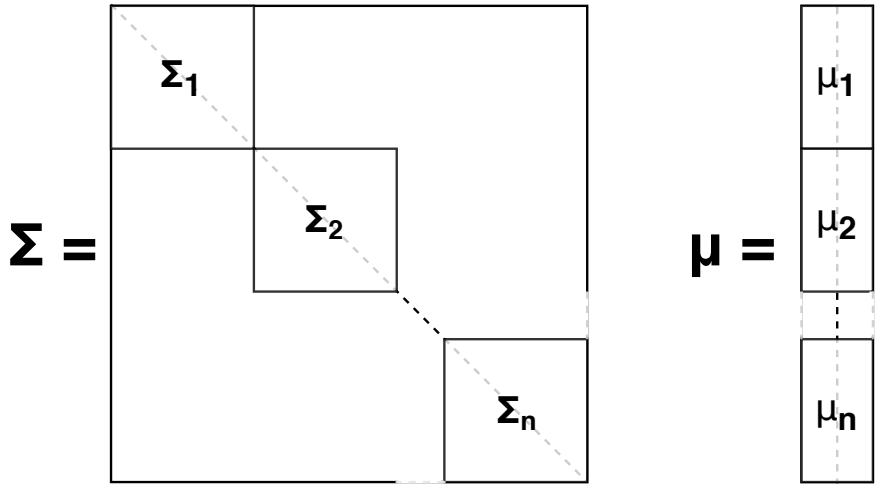


Figure 7.1 – **Block Diagonal parameterization.** Both covariance Σ and mean $\boldsymbol{\mu}$ are partitioned into blocks. In particular Σ is a block-diagonal matrix, hence it is populated with zeros outside the blocks. Each pair $\theta_i = \langle \Sigma_i, \boldsymbol{\mu}_i \rangle$ defines a sub-distribution which internally holds full covariance information while being independent from other blocks. By changing the number and size of the blocks, users can select a trade-off between convergence-speed and execution-speed. Particularly notable are the boundary settings: 1 block of p parameters (fastest convergence, slowest execution) corresponds to *XNES*, while p blocks of size 1 (slowest convergence, fastest execution) corresponds to *SNES*.

7.2 Algorithm Design

BD-NES is characterized by maintaining a Gaussian distribution with a block-diagonal covariance matrix over the space of parameters. Blocks in the covariance matrix corresponds to groups of parameters between which full covariance information is maintained. Blocks are however independent from each other, so parameters in different blocks are considered independent for sampling purposes. Size can vary between blocks.

Sampling the main distribution intuitively corresponds to sampling a set of sub-distributions (possibly in parallel) and joining the resulting samples into a full individual. This limits the intuition of BD-NES as simply a set of XNES algorithms running in parallel, as the fitness function is defined over full individuals and not block-localized subsamples. After the individual is constructed and evaluated, the individual score is used for the constituting subsamples in each underlying XNES.

In neuroevolution in particular, parameter correlation can be partially evinced from the network structure independently of the actual application or task. Network weights can be grouped at either neuron or layer level, as discussed in Section 7.1. Figure 7.1 exemplifies a layout where weights entering a same neuron are grouped together.

Algorithm 5 showcases the final algorithm applied to neuroevolution. The parameters correspond to the network weights (in *direct encoding*), grouping together those entering a same

neuron. As the groups can be described by maintaining independent distributions, each group is assigned a Gaussian with a mean $\boldsymbol{\mu}_i$ and a covariance Σ_i. The overall BD-NES distribution is parametrized by a mean built by concatenating the means corresponding to all parameters groups $\boldsymbol{\mu} = \langle \boldsymbol{\mu}_0 \dots \boldsymbol{\mu}_n \rangle$, and covariance built by placing the parameters groups covariances as the block of a block-diagonal matrix $\Sigma = \texttt{Diag}(\Sigma_0, \dots, \Sigma_n)$.

A generation of BD-NES is thus constituted of (i) sample λ times (with λ population size) each sub-distribution parametrized by $\boldsymbol{\mu}_i$, Σ_i; (ii) produce λ individuals by joining the sub-samples from each of the distributions; (iii) evaluate each individual on the task (in neuroevolution: generate the corresponding network phenotype, run it on the task, return the accumulated reward); (iv) assign the individual score as the fitness for each of the sub-samples that constitute it; (v) run the XNES update on each sub-distribution using the thus scored sub-samples.

Algorithm 5 provides the algorithm pseudocode, while an open-source implementation can be found at my GitHub repository[1].

Algorithm 5 BD-NES

Inputs:
n: number of blocks
λ: population size
Initialize:
$\theta_b \leftarrow (\boldsymbol{\mu}_b, \Sigma_b), \ \ b \in 1 \dots n$
while not solved **do**
for $i \leftarrow 1 \dots \lambda$ **do**
$\mathbf{z}^i \leftarrow \{\mathbf{z}^i_b \sim \mathcal{N}(\boldsymbol{\mu}_b, \Sigma_b)\}, \ \ \forall b \in 1 \dots n$ ▷ individual from concatenating sub-samples
$\text{fit}^i_b \leftarrow f(\mathbf{z}^i), \ \ \forall b \in 1 \dots n$ ▷ assign the individual's fitness to each sub-sample
$\theta_b \leftarrow \text{XNES\_UPDATE}(\theta_b, \ \text{fit}_b, \ \mathbf{z}_b), \ \ \forall b \in 1 \dots n$ ▷ update sub-distributions

Performance Analysis

As mentioned in Section 7.1, BD-NES generalizes the two main NES algorithms (SNES and XNES; see Chapter 2.5). This is because both full matrices and diagonal matrices are special cases of block-diagonal matrices: the first with one block of size p, the second with p blocks of size one. From a performance perspective thus BD-NES complexity is bound by the two, i.e. having a lower bound of $\mathcal{O}(p)$ (SNES) and upper bound of $\mathcal{O}(p^3)$ (XNES).

More precisely, Section 7.2 and Algorithm 5 highlight how each block borrows the update equation from XNES. The performance of a particular instantiation of BD-NES is thus $\mathcal{O}(B \times b^3)$ with B number of blocks (known constant) and b size of the largest block, which is $\mathcal{O}(b^3)$. This implies that any further subdivision of the parameters set into blocks reduces the cubic term while increasing a constant, enabling for large performance gains from just few blocks subdivision.

[1] `https://github.com/giuse/machine_learning_workbench/`

Since the cubic term in XNES comes from the covariance matrix updates, inspecting said matrix provides a direct intuition of the performance savings. Subdividing the parameters set into two groups immediately corresponds to a covariance matrix made of four quadrants, two of which (top-right and bottom-left, corresponding to half of the values of the matrix) are set to zeros and do not need maintaining.

Parallelization

BD-NES is an algorithm particularly suited for a parallel implementation. Since all the sub-distributions are independent, they can all be sampled and then updated independently in parallel. Moreover, individual scoring is also independent across individuals, meaning that all evaluations can also be run in parallel. The algorithm only requires synchronization in two points: (i) to build full individuals by concatenating sub-samples from all sub-distributions, and (ii) to return the scores of all individuals so that each sub-distribution can execute the XNES update.

Since both sampling and update are executed independently on all blocks, (each represented by an independent XNES instance), a parallel implementation has *constant scaling* on the number of groups, and thus *on the number of neurons or layers*, which removes the known hidden constant B from $\mathcal{O}(B \times b^3)$. While this in principle does not change the big-O performance of $\mathcal{O}(b^3)$, in practice this means that as long as parallel resources are available there is *no limit* on the number of parameters and thus network size. This partially decouples the algorithm performance from the performance of currently available hardware, a feat usually achieved only through the use of the most simplistic evolution strategies [Salimans et al., 2017].

A further improvement in performance can be obtained by spreading the covariance blocks over machines augmented with processing units optimized for linear algebra such as Graphics Processing Units (GPUs) and Tensor Processing Units (TPUs). If a $[b \times b]$ matrix can be fit in the processing unit's dedicated memory, there is no need to further move it, as only distribution samples are to be extracted and distributed through the network, enabling a potentially considerable speedup of the $\mathcal{O}(b^3)$ update step. The availability of such hardware has greatly increased in the past years thanks to deep learning applications, making it a prime time for parallel evolutionary algorithms with state-of-the-art convergence such as BD-NES.

7.3 Experimental Setup

To test the performance of BD-NES on a complex neuroevolution task, this work presents a new reinforcement learning control benchmark coming as a crossover between the (already quite complex) Octopus Arm and Acrobot. Such an environment requires large network controllers, with a number of parameters (weights) beyond the applicability of full-covariance methods such as XNES and CMA-ES [Hansen and Ostermeier, 2001]. Such problems have been addressed by SNES in past work, but at the cost of convergence speed and often quality

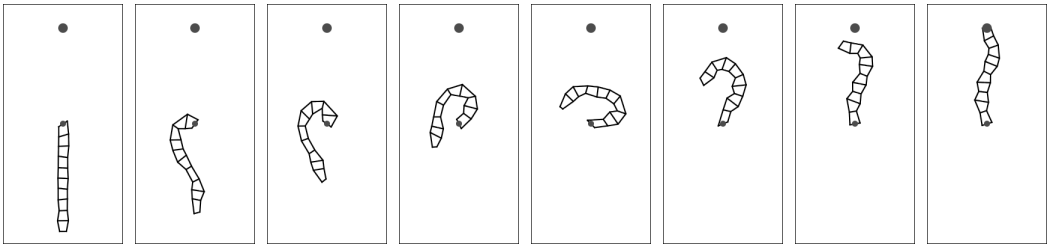


Figure 7.2 – **The Octopus Arm Acrobot task.** An arm composed of flexible segments is controlled by a neural network. Initially hanging down, goal of the task is to raise it against gravity and stretch it to touch the point directly above its fulcrum. Limited arm strength imposes the need to learn a wind-and-unwind behavior through direct muscle control. The frames shown correspond to the behavior of a neural network evolved with BD-NES.

of result. Ideally BD-NES should provide a better trade-off by considering intra-neuron correlation while ignoring any inter-neuron.

7.3.1 The Octopus-Arm Acrobot Task

The proposed Octopus Arm Acrobot task blends two classic RL benchmarks for increased complexity:

- *From the Octopus Arm* [Yekutieli et al., 2005, Woolley and Stanley, 2010]: the neural network controls a segmented non-rigid arm by directly controlling the "muscles" that compose it. Each segment has constant volume (non-compressible) and includes three muscles which can be independently contracted.

- *From the Acrobot* [Spong, 1994]: the arm is suspended from a swiveling pivot, pointing downwards following gravity. The pivot provides two more torque (rotational) controls. The goal of the task is to raise the arm against gravity and touch a point stretching directly above the pivot, against gravity. The pivot does not have sufficient torque to raise the arm directly, thus requiring to learn a wind-and-unwind behavior (Figure 7.2)

For each segment, the input to the network is constructed as two pairs of $\langle x, y \rangle$ coordinates corresponding to the positions of the two extending corners of the segment, plus the corresponding velocities $\langle \mathbf{x}, \mathbf{y} \rangle$. This totals to 8 inputs per segment; to this is added the pivot's rotational torque and velocity, for a total of $8s+2$ inputs with s number of segments in the arm. The output of the network is interpreted as whether to contract any of the 3 muscles plus a torque and direction control for the pivot, for a total of $3s+2$ control variables. The result proposed are based on a 10-segments arm, for a total of 82 inputs and 32 controls.

The pivot is under-actuated, i.e. the available torque is not sufficient to simply swing up the outstretched arm. Also, rather than using the simpler, high level actions which were defined in the classic version of the octopus arm (allowing to simultaneously contract groups of muscles),

this work opted for the much harder direct and independent muscle control.

7.3.2 Parameterization

Results shown utilize a fully-connected single-layer recurrent neural network of 32 neurons. Each neuron outputs corresponds to one of the 32 control variables of the task. The recurrent connections provide the controller with limited *memory* capabilities, enabling it to internally compute and consider momentum in the swing. For simplicity, no hidden layer have been added to the architecture, as the network's complexity is shown sufficient to learn a prompt and smooth trajectory.

The network accounts for 32×82 input connections, plus 32×32 recurrent connections and 32 bias connections, totaling 3 680 weights. Searching in 3 680 dimensions with XNES was unfeasible at the time of this work. The full covariance matrix has $3\,680 \times 3\,680 = 13\,542\,400$ entries. At the same time though, each of the 32 neurons accounts for only 1/32 of the incoming connections, which is $82 + 32 + 1 = 115$. Grouping the corresponding parameters for BD-NES yields 32 115×115 blocks, each thus made of 13 225 values, *two orders of magnitude smaller* than the full covariance tackled by XNES.

7.3.3 Complete Setup

This experiment fixes the population size to $\lambda = 50$. The learning rate for the mean was set using the NES defaults (as seen in Chapter 2.5) of $\eta_{\boldsymbol{\mu}} = \left[\log(p) + 3\right] / \left[5\sqrt{p}\right]$ and $\eta_{\Sigma} = \eta_{\boldsymbol{\mu}}/2$. Each run was alloted 200 generations, corresponding to a total of 10 000 individual evaluations.

The fitness function considers both the distance between the arm tip and the goal (to avoid the fitness plateau of all unsuccessful individuals) and the time taken to solve the task (to develop an optimized behaviors with a direct, fluid and short movement). Let t the time taken to touch the goal (i.e. number of steps until success), T the maximum alloted number of steps before defaulting to failure (100 in this experiment), and d and D the Euclidean distances between the tip of the arm at its final position and the target vs. its initial position respectively. The fitness of an individual $\mathbf{z}$ is then computed as:

$$f(\mathbf{z}) = 1 - \frac{t}{T}\frac{d}{D} \quad . \tag{7.1}$$

This differs from the original fitness proposed in [Woolley and Stanley, 2010] because the latter only focuses on the distance between the arm tip and the goal, thus promoting locally greedy behaviors that minimize the distance to the goal without including any swinging motion. The integration with the acrobot task, together with the decision for direct muscle control (vs. the original high-level actions) makes this choice less ideal. The fitness function proposed in this

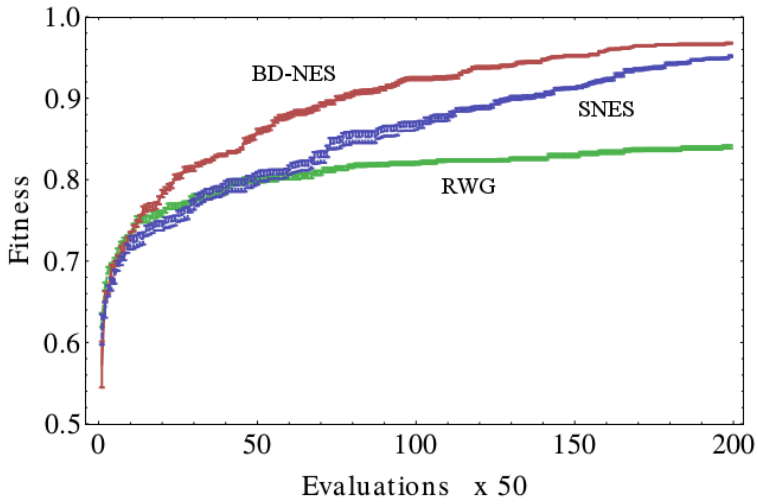


Figure 7.3 – **Performance comparison.** The performance of the BD-NES algorithm (top) is compared against SNES (middle) and RWG (bottom) on the octopus arm acrobot task. What look like curves are actually scatterplots, with (tight) error bars over 20 runs. BD-NES displays the more consistent behavior, achieving higher fitness with fewer iterations (faster convergence), and learns solutions of distinctively higher quality.

work instead includes a time-to-solve factor which rewards execution speed, leading to much more fluid and purposeful movements.

To establish the complexity of the octopus arm acrobot task, a baseline was established through Random Weight Guessing (RWG; Schmidhuber et al. [2001]): comparably to SNES and BD-NES, 10 000 networks are generated, in this case by selecting the network weights i.i.d. in a given range. These are divided into "generations" of λ individuals to provide direct comparison with the two ES. At each generation the networks are evaluated, and the best individual found is returned at the end of the task.

7.3.4 Results

Experimental results are depicted in Figure 7.3, as performance of the best individual found up to each generation. While looking like a line plot at first glance, the graph is a dense scatterplot with very tight error bars, corresponding to 20 runs for statistical significance. BD-NES converges notably faster than SNES, reaching a performance equivalent to the latter with roughly 30% less evaluations (7 000 at generation 140 vs. 10 000 at generation 200). Arguably even more importantly, it does this in shorter wall-clock time: while the main advantage of SNES is speed, BD-NES reaches comparable results in 15% less CPU time (55 minutes for BD-NES vs. 65 minutes for SNES). For fair comparison, BD-NES was implemented in a standard sequential fashion over a single CPU, thus not leveraging parallel resources nor GPU/TPU computation, both of which would further and considerably shorten the total run time.

All experiments were run on a machine with Intel i7 640M processor running at 3.33GHz, with 4GB of DDR3 RAM at 1066MHz. All code was implemented in Wolfram Mathematica but for linking the Java implementation of the Octopus Arm, which is available online at `http://www.cs.mcgill.ca/~idprecup/workshops/ICML06/octopus.html`.

7.4 Discussion

Natural Evolution Strategies is a family of black box optimization algorithms with state of the art performance and a history of applications to neuroevolution. The most prominent algorithms of the family (XNES and SNES; see Chapter 2.5.2) maintain a Gaussian distribution over the space of network weights, as described in Chapter 2.4. The two algorithms differ on whether covariance between parameters is maintained or ignored, trading off convergence speed and execution speed. While faster convergence typically yields higher solution quality for a given number of iterations, choosing an algorithm with high execution speed enables applying the algorithm to problem with much higher dimensionality.

Block Diagonal NES (BD-NES) generalizes both XNES and SNES, thus sidestepping the choice, while enabling the user to select a more fine-grained trade-off. The parameters set is partitioned into groups of (expectedly) high-correlated parameters. Full covariance information is then maintained inside each group, while covariance between groups is instead ignored, leading to the construction of a *block diagonal* covariance matrix.

When used to train neural networks (neuroevolution), parameter correlation can be partially reduced to the (known) network structure, making parameters grouping straightforward for weights entering a same neuron or layer. This information is always available independently from the actual task of application.

Algorithm performance is tested on a new, high-complexity reinforcement learning control task called the *octopus arm acrobot*, built by combining the classical octopus arm and acrobot tasks. Results are compared with SNES for algorithm performance, since the size of the network and hence the number of parameters is too high for XNES to be applicable. Results from random weight guessing also provide a baseline for the task's complexity. BD-NES performance is shown to surpass SNES both in higher convergence speed (i.e. less evaluations) and shorter wall-clock time for comparable results. Moreover, BD-NES is ultimately capable of producing solutions of consistently higher quality in the total time alloted.

A sequential implementation of BD-NES scales *linearly* with the number of blocks, which in neuroevolution corresponds to neurons and/or layers. A parallel implementation has *constant scaling* over the number of blocks, since the updates for each block are run independently, and performance is only bound by the size of the largest block. This makes BD-NES a prime candidate for *deep neuroevolution*, i.e. searching for deep networks with potentially millions of weights. A further boost in performance can be achieved by running the updates independently on multiple machines (as many as there are blocks) augmented with GPUs or TPUs.

Each single covariance block update is independent from the others, the block size can be tailored to fit in the device's dedicated memory, and no further moving of the matrix is necessary, the only communication between blocks being restricted to individual sub-samples and fitness scores.

This chapter concludes the overview over the research questions stipulated in Chapter 1.1. It is now time to tally the results so far and discuss whether these have been addressed to satisfaction.

8 Conclusions

The overarching goal of the work presented in this thesis is to extend the applicability of neuroevolution to a broader range of applications, somehow complementary to deep learning.

Neural networks have caught the public spotlight over the past years thanks to their superior performance and adaptability to a large set of applications. Deep learning however trains the networks based on the *backpropagation* algorithm, a single-agent gradient descent technique with a strict set of requirements limiting its applicability. An alternative can be found in *neuroevolution*, which searches in the space of neural networks using evolutionary algorithms, sidestepping these limitations entirely. While this broadens the scope of problems that can be addressed by neural networks, the performance of traditional neuroevolution methods is not yet on par with deep learning results, requiring further research.

This thesis began (Question 1) by examining whether there is any reason to consider alternatives to deep learning at all (Chapter 3). Given the extreme complexity attainable by deep neural networks, it could be argued that selecting a sufficiently large network should be sufficient in principle to address problems of arbitrary complexity. One of the limitations of deep learning though is in the amount of data consequently required to train deep networks: in many applications simply not enough data is available. The chapter highlights such limitations in the scope of a highly complex industrial application. The proposed approach is instead based on extracting small quantities of high-quality data during pre-processing, constructing an ideal environment for the training of simpler models such as *shallow* networks, with results reaching the highest precision attainable (i.e. sensor-level predictions). Deep learning is an extremely powerful tool, but more suited for problems of scale. Depending on the application, it is crucial to maintain a wide perspective over machine learning as a whole, to be able to select an appropriate tool for each application.

Next, Question 2 addressed *fitness stagnation*, which is a major limitation to the otherwise robust applicability of neuroevolution. Evolutionary algorithms performance deteriorates in the presence of deceptive fitness landscapes such as plateaus and local attraction basins. In *reinforcement learning* for example, applications with sporadic reward returns generate

ample fitness plateaus, where individuals are simply *indistinguishable* based on performance alone. Chapter 4 tackles this issue by means of a novel *restart strategy*, inspired by *novelty search*. Upon detecting convergence (the algorithm has stopped improving), the search is restarted in an area of the space which is considered promising from a novelty perspective, i.e. which has been visited before but not thoroughly explored, and where individuals with uncommon behaviors are found. This makes the algorithm extremely resilient to stagnation, while boosting its scaling invariance: even after full convergence, the search has a chance to keep using the start anew with a larger scope over the next most promising area. The method can be applied to any evolutionary algorithm with no disadvantages, and is likely to improve performance in most applications.

The thesis then moved on (Question 3) to address the scalability of neuroevolution towards problems typically requiring larger sized neural networks. The question here points to how can neuroevolution scale to problems with high dimensionality. Particularly, as networks need to deal with high-dimensional inputs, a large amount of their weights (i.e. their computational power) is devoted to *feature extraction*, i.e. extracting sensible information from the data. The actual decision making can be seen as an additional process on top of it. Chapter 5 delegated feature extraction to a simple yet specialized *unsupervised learning* method. With minimal performance overhead, this produces a compact *code* on top of which a relatively small network can achieve high performance. The feature extractor (or *encoder*) is trained online along the normal neuroevolution iterations, based on data obtained by (and for) the networks generated through the evolution. This not only does eliminate the need for bootstrapping or prior knowledge, but makes the encoder aware of the interactions between individuals and environment throughout the whole search so far, across generations. Such awareness enables the encoder to produce a *novelty score* for each individual, based on its ability to interact with the environment in novel ways, which in turn further boosts the performance of the evolutionary search. Results show how such a system, decoupling feature extraction from decision making, is indeed advantageous and requires further investigation.

The work of Chapter 5 extends Chapter 6 in order to verify (Question 4) whether such techniques are sufficient to address problems commonly considered "complex" for modern machine learning. A new system of encoder plus network was tested on a modern deep learning benchmark based on a simulator of the Atari gaming console. This time the encoder was designed anew specifically for pre-processing observations in the context of *continuous control*, emphasizing its ability to *differentiate* observations as an approximation of state differentiation. A tiny decision-making networks on top is then proven capable of playing Atari games with *as little as six neurons*, whereas the literature usually calls for hundreds. Rather than making a point on smaller networks though, the main implication is that large networks (as currently trained to address Atari games) could in principle be better applied to problems of much higher complexity, by dedicating their full computational power to decision making alone.

Last, Question 5 could be considered the "elephant in the room" over the previous few chap-

ters: can evolutionary algorithms scale to deep networks? The methods presented in the chapters so far enable addressing higher complexity problems with smaller networks, but eventually more complex tasks still require proportionally larger networks. Chapter 7 answers by proposing a new evolution strategy capable of state-of-the-art results, flexible performance and high parallelization. Particularly the algorithm allows to adapt the size of the blocks to the performance available on single machines, then scales *linearly* with the size of the network, or *constantly* over multiple machines if available, unlocking the potential in principle to evolve neural networks of unprecedented size.

8.1 Limitations and Perspectives

To understand the applicability and correspondingly the limitations of Neuroevolution it is important to understand just how general are the techniques upon which it is founded: neural networks and evolutionary algorithms.

Neural networks are universal approximators [Hornik et al., 1989]. A network with one hidden layer and nonlinear activation can in principle approximate any function to arbitrary precision. The catch is of course in the cost: using a single layer requires a finite but otherwise undisclosed number of neurons, often impractical. And whether such a parametrization is easy (if possible at all) to learn is beyond the scope of the theorem.

Networks with neurons on multiple layers tend to be more efficient from this perspective: as function composition raises the functional complexity more quickly, less neurons are required to approximate a function of given complexity. Increasing the number of layers thus improves the effectiveness of training methods by reducing the number of neurons (and thus weights, parameters) to learn. At the same time however it also makes the process harder for algorithms which propagate the error back through the network, because of the vanishing gradient problem: it is hard to impute correctly the contribution of a first-layer weight to the construction of a given error in a many-layered network, as accountability decays with each layer. Neuroevolution on the other hand does not compute gradients over the weights, and thus does not suffer from negative effects with network depth.

Still there are other downsides to the adoption of neural networks which are not as easily sidestepped. Most importantly, in many modern applications there is a need for understandability and maintainability. While neural networks have been crucial in constructing impressive results in several applications from autonomous driving to medical diagnosis, their deployment has been rather limited because the process through which the answers are generated is not humanly fathomable. Being unable to understand *why* and *how* a certain answer is generated drastically reduces the confidence in using such answers in critical domains and applications. The equations implicit in deep networks are too large for human reading, let alone understanding. The behavior of a network can be known in any given case, but this does not imply any expectation or deductibility on the expected behavior in an unseen context. This is a limit unfortunately orthogonal to neuroevolution, inherent in neural networks:

their mathematical parametrization offers unlimited adaptability, at the cost of being beyond human comprehension.

In order to obtain results which are understandable by humans, it is necessary to consider alternative function approximators, built from the ground up for human interpretability. Incidentally, while gradient descent methods such as backpropagation would require a mathematical overhaul to be applicable to new models, black box optimization techniques such as evolutionary algorithms by definition are not concerned with the nature of the application, and will thereby remain relevant and viable for training the newly designed models. The only requirement for the application of evolutionary algorithms is for a fitness function to score individuals (i.e. parameter sets) based on their performance on the task. Some work gets away with setting aside even this requirement, showing that evolutionary algorithms are capable of performing based uniquely on intrinsic criteria [Lehman and Stanley, 2008, 2010]. In principle there is no application (that can be described by a set of parameters) for which evolutionary algorithms are not applicable. In practice though, as shown throughout this thesis, performance limitations still apply, and need to be further addressed in future work.

8.2 Future Work

The first and natural step concerns the implementation of an encompassing system that integrates all the work presented so far: input pre-processing, novelty-based exploration, and scaling to deep networks. Each technique described in these chapters, while been developed in isolation, has been designed with the goal of integrating with each other in mind. Implementing such a system still remains a challenge though, mainly because of the number of components and their complexity.

A first design was described in Chapter 5 as having three components: a model, an optimizer (to train the model) and a pre-processor – plus naturally an environment providing the task. Chapter 5 proposes a system that addresses a continuous control task using fully-connected recurrent neural networks as the model, Separable NES as the optimizer, and a custom VQ with per-centroid decaying learning rates for the pre-processor. Chapter 6 offers a different instance, with tiny single-layer recurrent neural networks as the model, eXponential NES as the optimizer, and Increasing Dictionary VQ plus Direct Residuals SC addressing pre-processing. A natural next step would be to introduce Block Diagonal NES as the optimizer, in order to scale the model to deep recurrent networks. Still each block can be improved independently, then be fit back in the architecture in a plug-in fashion.

New and more sophisticated feature extractors should be created to improve on the specialized techniques presented in Chapters 5 and 6. The pre-processor in Chapter 6 in particular has been designed with simplicity in mind, to highlight how all decision making had to reside in the network component. Addressing environments with higher complexity however will require methods capable of extracting generic features with no prior knowledge of the environment. Neural network-based techniques such as convolutional networks and autoencoders are

themselves prime candidates, with BD-NES enabling the alternative option of training them with neuroevolution. The ability to reconstruct the original observation from the compressed code, set aside in Chapter 6 in favor of bare simplicity, should also be taken into account in further design. This would enable to reintroduce the ability to generate a novelty signal to improve the scoring, as presented in Chapter 5.

Finally, modern evolution strategies are yet to reach their pinnacle of performance and optimization. BD-NES, presented in Chapter 7, enables scaling the network size based on the available computational resources. This is achieved by grouping together parameters with high covariance, while discarding altogether the covariance between parameters of different groups. The same concept could in principle be applied to other evolutionary algorithms, paving the way for an entire class of new parallel implementations.

Bibliography

Hervé Abdi. Partial least squares regression and projection on latent structure regression (PLS Regression). *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(1):97–106, 2010. ISSN 1939-0068. doi: 10.1002/wics.51.

Samuel Alvernaz and Julian Togelius. Autoencoder-augmented neuroevolution for visual doom playing. In *Computational Intelligence and Games (CIG), 2017 IEEE Conference on*, pages 1–8. IEEE, 2017.

Shunichi Amari and Scott C. Douglas. Why natural gradient? In *ICASSP*, volume 98, pages 1213–1216. Citeseer, 1998.

Anne Auger and Nikolaus Hansen. A restart CMA evolution strategy with increasing population size. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 2, pages 1769–1776. IEEE, 2005.

David Beasley, David R. Bull, and Ralph R. Martin. A sequential niche technique for multimodal function optimization. *Evolutionary Computation*, 1(2):101–125, 1993.

John B. Bell. Solutions of ill-posed problems. 1978.

Marc G. Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 47:253–279, 2013.

Christopher M. Bishop. *Pattern recognition and machine learning*, volume 1. Springer, 2006.

Léon Bottou and Patrick Gallinari. A framework for the cooperation of learning algorithms. In *Proceedings of the 3rd International Conference on Neural Information Processing Systems*, pages 781–788. Morgan Kaufmann Publishers Inc., 1990.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. OpenAI Gym, 2016.

Patryk Chrabaszcz, Ilya Loshchilov, and Frank Hutter. Back to basics: Benchmarking canonical evolution strategies for playing atari. *arXiv preprint arXiv:1802.08842*, 2018.

Adam Coates and Andrew Y. Ng. The importance of encoding versus training with sparse coding and vector quantization. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 921–928, 2011.

Edoardo Conti, Vashisht Madhavan, Felipe Petroski Such, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Improving exploration in evolution strategies for deep reinforcement learning via a population of novelty-seeking agents. *arXiv preprint arXiv:1712.06560*, 2017.

Giuseppe Cuccu and Faustino Gomez. When novelty is not enough. In *Proceedings of evo* 2011 (Torino, Italy)*, 2011.

Giuseppe Cuccu and Faustino Gomez. Block diagonal natural evolution strategies. In *International Conference on Parallel Problem Solving from Nature*, pages 488–497. Springer, 2012.

Giuseppe Cuccu, Faustino Gomez, and Tobias Glasmachers. Novelty-based restarts for evolution strategies. In *Evolutionary Computation (CEC), 2011 IEEE Congress on*, pages 158–163. IEEE, 2011a.

Giuseppe Cuccu, Matthew Luciw, Jürgen Schmidhuber, and Faustino Gomez. Intrinsically motivated neuroevolution for vision-based reinforcement learning. In *Development and Learning (ICDL), 2011 IEEE International Conference on*, volume 2, pages 1–7. IEEE, 2011b.

Giuseppe Cuccu, Somayeh Danafar, Philippe Cudré-Mauroux, Martin Gassner, Stefano Bernero, and Krzysztof Kryszczuk. A data-driven approach to predict nox-emissions of gas turbines. In *Big Data (Big Data), 2017 IEEE International Conference on*, pages 1283–1288. IEEE, 2017.

Giuseppe Cuccu, Julian Togelius, and Philippe Cudre-Mauroux. Playing Atari with six neurons. *ArXiv e-prints*, 2018.

Thomas G Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer, 2000.

Roger Eckhardt. Stan Ulam, John von Neumann, and the Monte Carlo method. *Los Alamos Science*, 15(131-136):30, 1987.

Scott E. Fahlman. Faster learning variations on back-propagation: An empirical study. In *Proceedings of the Connectionist Models Summer School*, pages 38–51. Morgan Kaufmann, 1988.

Fernando Fernández and Daniel Borrajo. Two steps reinforcement learning. *International Journal of Intelligent Systems*, 23(2):213–245, 2008.

Dario Floreano, Peter Dürr, and Claudio Mattiussi. Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1(1):47–62, 2008.

David B Fogel. *Evolutionary computation: the fossil record*. Wiley-IEEE Press, 1998.

L.J. Fogel, A.J. Owens, and M.J. Walsh. *Artificial Intelligence through Simulated Evolution*. Willey, New York, 1966.

Alex Fraser, Donald Burnell, et al. *Computer models in genetics*. Mcgraw-Hill Book Co., New York., 1970.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

Alex S. Fukunaga. Restart scheduling for genetic algorithms. In *International Conference on Parallel Problem Solving from Nature*, pages 357–366. Springer, 1998.

Jason Gauci and Kenneth O. Stanley. Generating large-scale neural networks through discovering geometric regularities. In *Proceedings of the Conference on Genetic and Evolutionary Computation*, pages 997–1004, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-697-4. doi: http://doi.acm.org/10.1145/1276958.1277158.

Linus Gisslén, Matt Luciw, Vincent Graziano, and Jürgen Schmidhuber. Sequential Constant Size Compressors and Reinforcement Learning. In *Proceedings of the Fourth Conference on Artificial General Intelligence*, 2011.

Tobias Glasmachers, Tom Schaul, Sun Yi, Daan Wierstra, and Jürgen Schmidhuber. Exponential natural evolution strategies. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 393–400. ACM, 2010.

Fred Glover and Manuel Laguna. Tabu search. In *Handbook of combinatorial optimization*, pages 2093–2229. Springer, 1998.

Faustino Gomez, Jürgen Schmidhuber, and Risto Miikkulainen. Accelerated neural evolution through cooperatively coevolved synapses. *Journal of Machine Learning Research*, 9(May): 937–965, 2008.

Faustino J. Gomez. *Robust Nonlinear Control through Neuroevolution*. PhD thesis, Department of Computer Sciences, University of Texas at Austin, 2003.

Robert Gray. Vector quantization. *IEEE Assp Magazine*, 1(2):4–29, 1984.

Vincent Graziano, Tobias Glasmachers, Tom Schaul, Leo Pape, Giuseppe Cuccu, Jürgen Leitner, and Jürgen Schmidhuber. Artificial curiosity for autonomous space exploration. 4:41–51, 2011.

Arthur Gretton, Olivier Bousquet, Alexander Smola, and Bernhard Schölkopf. Measuring statistical dependence with Hilbert-Schmidt norms. In *Algorithmic Learning Theory*, pages 63–77. Springer, 2005. doi: 10.1007/11564089_7.

Frederic Gruau. *Neural Network Synthesis using Cellular Encoding and the Genetic Algorithm*. PhD thesis, l'Universite Claude Bernard-Lyon 1, France, 1994.

Felix Güthe, Jaan Hellat, and Peter Flohr. The reheat concept: The proven pathway to ultralow emissions and high efficiency and flexibility. *Journal of Engineering for Gas Turbines and Power*, 131(2), 2008. ISSN 0742-4795. doi: 10.1115/1.2836613.

David Ha and Jürgen Schmidhuber. World models. *arXiv preprint arXiv:1803.10122*, 2018.

Nikolaus Hansen and Andreas Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary computation*, 9(2):159–195, 2001.

Matthew Hausknecht, Joel Lehman, Risto Miikkulainen, and Peter Stone. A neuroevolution approach to general Atari game playing. *IEEE Transactions on Computational Intelligence and AI in Games*, 6(4):355–366, 2014.

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. *arXiv preprint arXiv:1710.02298*, 2017.

Tin Kam Ho. Random decision forests. In *Document analysis and recognition, 1995., proceedings of the third international conference on*, volume 1, pages 278–282. IEEE, 1995.

Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91(1), 1991.

John J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

Harold Hotelling. The relations of the newer multivariate statistical methods to factor analysis. *British Journal of Statistical Psychology*, 10(2):69–79, 1957. ISSN 2044-8317. doi: 10.1111/j.2044-8317.1957.tb00179.x.

David H. Hubel and Torsten N. Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, 195(1):215–243, 1968.

Christian Igel. Neuroevolution for reinforcement learning using evolution strategies. In *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on*, volume 4, pages 2588–2595. IEEE, 2003.

Christian Igel and Michael Hüsken. Improving the RPROP learning algorithm. In *Proceedings of the Second International ICSC Symposium on Neural Computation (NC 2000)*, volume 2000, pages 115–121, 2000.

JNR Jeffers. Two case studies in the application of principal component analysis. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 16(3):225–236, 1967. ISSN 0035-9254. doi: 10.2307/2985919.

Sébastien Jodogne and Justus H. Piater. Closed-loop learning of visual control policies. *Journal of Artificial Intelligence Research*, 28(1):349–391, 2007.

Joshua Knowles and David Corne. The pareto archived evolution strategy: A new baseline algorithm for pareto multiobjective optimisation. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 1, pages 98–105. IEEE, 1999.

Jan Koutník, Faustino Gomez, and Jürgen Schmidhuber. Evolving neural networks in compressed weight space. In *Proceedings of the Conference on Genetic and Evolutionary Computation (GECCO-10)*, 2010.

Jan Koutník, Giuseppe Cuccu, Jürgen Schmidhuber, and Faustino Gomez. Evolving large-scale neural networks for vision-based reinforcement learning. In *Proceedings of the 15th annual conference on Genetic and evolutionary computation*, pages 1061–1068. ACM, 2013a.

Jan Koutník, Giuseppe Cuccu, Jürgen Schmidhuber, and Faustino Gomez. Evolving large-scale neural networks for vision-based TORCS. In *Foundations of Digital Games (FDG)*, pages 206–212, 2013b. ISBN ISBN 978-0-9913982-0-1.

John R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1097–1105, 2012.

Sascha Lange and Martin Riedmiller. Deep auto-encoder neural networks in reinforcement learning. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2010.

Yann LeCun. Une procedure d'apprentissage pour reseau a seuil asymetrique [a learning procedure for asymmetric threshold network]. *Proceedings of Cognitiva*, 85:599–604, 1985.

Yann LeCun, Léon Bottou, Genevieve B. Orr, and Klaus-Robert Müller. Efficient BackProp. In G. Orr and Muller K., editors, *Neural Networks: Tricks of the trade*. Springer, 1998.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436, 2015.

Robert Legenstein, Niko Wilbert, and Laurenz Wiskott. Reinforcement learning on slow features of high-dimensional input streams. *PLoS Computational Biology*, 6(8):e1000894, 2010.

Joel Lehman and Kenneth O. Stanley. Exploiting open-endedness to solve problems through the search for novelty. In *Proceedings of the Eleventh International Conference on Artificial Life (ALIFE XI)*. MIT, 2008.

Joel Lehman and Kenneth O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation Journal*, 2010.

Moritz Lipperheide, Frank Weidner, Manfred Wirsum, Martin Gassner, and Stefano Bernero. Long-term NOx emission behavior of heavy duty gas turbines: An approach for model-based monitoring and diagnostics. In *ASME Turbo Expo 2017: Turbomachinery Technical Conference and Exposition*, pages V006T05A001–V006T05A001. American Society of Mechanical Engineers, 2017.

Simon M. Lucas and Thomas P. Runarsson. Temporal difference learning versus co-evolution for acquiring othello position evaluation. In *Computational Intelligence and Games, 2006 IEEE Symposium on*, pages 52–59. IEEE, 2006.

Simon M. Lucas and Julian Togelius. Point-to-point car racing: an initial study of evolution versus temporal difference learning. In *Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on*, pages 260–267. IEEE, 2007.

Samir W. Mahfoud. Niching methods for genetic algorithms. *Urbana*, 51(95001):62–94, 1995.

Julien Mairal, Francis Bach, Jean Ponce, et al. Sparse modeling for image and vision processing. *Foundations and Trends® in Computer Graphics and Vision*, 8(2-3):85–283, 2014.

Stéphane G Mallat and Zhifeng Zhang. Matching pursuits with time-frequency dictionaries. *IEEE Transactions on signal processing*, 41(12):3397–3415, 1993.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

Andrew W. Moore. Knowledge of knowledge and intelligent experimentation for learning control. In *International Joint Conference on Neural Networks*, volume 2, pages 683–688. IEEE, 1991.

Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted Boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

Gonzalo Navarro. A guided tour to approximate string matching. *ACM computing surveys (CSUR)*, 33(1):31–88, 2001.

Steffen Nissen. Implementation of a fast artificial neural network library (FANN). *Department of Computer Science University of Copenhagen (DIKU)*, 2003.

Thomas Palmé, Francois Liard, and Dirk Therkorn. Similarity based modeling for turbine exit temperature spread monitoring on gas turbines. In *ASME Turbo Expo*, page V004T06A020, 2013.

David B Parker. Learning-logic. Technical Report TR-47, 1985.

Yagyensh Chandra Pati, Ramin Rezaiifar, and Perinkulam Sambamurthy Krishnaprasad. Orthogonal matching pursuit: Recursive function approximation with applications to wavelet decomposition. In *Signals, Systems and Computers, 1993. 1993 Conference Record of The Twenty-Seventh Asilomar Conference on*, pages 40–44. IEEE, 1993.

Judea Pearl. *Probabilistic reasoning in intelligent systems: Networks of plausible inference*. Elsevier, 2014.

K. Person. On Lines and Planes of Closest Fit to System of Points in Space. *Philiosophical Magazine*, 2:559–572, 1901.

David Pierce and Benjamin Kuipers. Map learning with uninterpreted sensors and effectors. *Artificial Intelligence*, 92:169–229, 1997.

Ingo Rechenberg. Evolutionsstrategie: Optimierung technischer systeme nach prinzipien der biologischen evolution. *Frommann-Holzboog Verlag, Stuttgart*, 1973.

Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *Neural Networks, 1993., IEEE International Conference on*, pages 586–591. IEEE, 1993.

Sebastian Risi and Julian Togelius. Neuroevolution in games: State of the art and open challenges. *IEEE Transactions on Computational Intelligence and AI in Games*, 9(1):25–41, 2017.

Frank Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

Frank Rosenblatt. Principles of neurodynamics. Perceptrons and the theory of brain mechanisms. Technical report, 1961.

Christian Rudolf, Manfred Wirsum, Martin Gassner, and Stefano Bernero. Analysis of long-term gas turbine operation with a model-based data reconciliation technique. In *ASME Turbo Expo*, page V006T05A008, 2015.

Christian Rudolf, Manfred Wirsum, Martin Gassner, Benjamin Timo Zoller, and Stefano Bernero. Modelling of gas turbine NOx emissions based on long-term operation data. In *ASME Turbo Expo*, page V04BT04A006, 2016.

David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

Thomas P. Runarsson and Simon M. Lucas. Coevolution versus self-play temporal difference learning for acquiring position evaluation in small-board go. *IEEE Transactions on Evolutionary Computation*, 9(6):628–640, 2005.

Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning. *arXiv preprint arXiv:1703.03864*, 2017.

Tom Schaul, Tobias Glasmachers, and Jürgen Schmidhuber. High dimensions and heavy tails for natural evolution strategies. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 845–852. ACM, 2011.

Jüergen Schmidhuber, Sepp Hochreiter, and Yoshua Bengio. Evaluating benchmark problems by random guessing. In S. C. Kremer and J. F. Kolen, editors, *A Field Guide to Dynamical Recurrent Neural Networks*. IEEE press, 2001.

Jürgen Schmidhuber. A possibility for implementing curiosity and boredom in model-building neural controllers. In J. A. Meyer and S. W. Wilson, editors, *Proc. of the International Conference on Simulation of Adaptive Behavior: From Animals to Animats*, pages 222–227. MIT Press/Bradford Books, 1991.

Jürgen Schmidhuber. Developmental robotics, optimal artificial curiosity, creativity, music, and the fine arts. *Connection Science*, 18(2):173–187, 2006.

Jürgen Schmidhuber. Formal Theory of Creativity, Fun, and Intrinsic Motivation (1990–2010). *Autonomous Mental Development, IEEE Transactions on*, 2(3):230–247, 2010. ISSN 1943-0604.

Bernhard Schölkopf, Alex J. Smola, Robert C. Williamson, and Peter L. Bartlett. New Support Vector Algorithms. *Neural Computation*, 12(5):1207–1245, 2000. ISSN 0899-7667. doi: 10.1162/089976600300015565.

John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, Cambridge, 2004. ISBN 978-0-511-80968-2.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587): 484, 2016.

Satinder P. Singh and Richard S. Sutton. Reinforcement learning with replacing eligibility traces. *Machine learning*, 22(1-3):123–158, 1996.

Alex J. Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and Computing*, 14(3):199–222, 2004. ISSN 0960-3174, 1573-1375. doi: 10.1023/B:STCO.0000035301.49549.88.

Mark W. Spong. Swing up control of the acrobot. In *Proceedings of the 1994 IEEE Conference on Robotics and Automation*, volume 46, pages 2356–2361, San Diego, CA, 1994.

Kenneth O. Stanley. *Efficient Evolution of Neural Networks Through Complexification*. PhD thesis, Department of Computer Sciences, University of Texas at Austin, August 2004. Technical Report AI-TR-04-314.

Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

Kenneth O. Stanley, David B. D'Ambrosio, and Jason Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15(2):185–212, 2009.

Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning. *arXiv preprint arXiv:1712.06567*, 2017.

Yi Sun, Daan Wierstra, Tom Schaul, and Jürgen Schmidhuber. Efficient natural evolution strategies. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 539–546. ACM, 2009.

Richard S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in neural information processing systems*, pages 1038–1044, 1996.

Richard S. Sutton and Andrew G. Barto. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

Dirk Therkorn. Remote monitoring and diagnostic for combined-cycle power plants. In *ASME Turbo Expo*, pages 697–703, 2005. doi: 10.1115/GT2005-68710.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

Julian Togelius, Tom Schaul, Daan Wierstra, Christian Igel, Faustino Gomez, and Jürgen Schmidhuber. Ontogenetic and phylogenetic reinforcement learning. *Künstliche Intelligenz*, 23(3):30–33, 2009.

Evert Vanderhaegen, Michaël Deneve, Hannes Laget, Nathalie Faniel, and Jan Mertens. Predictive Emissions Monitoring Using a Continuously Updating Neural Network. In *ASME Turbo Expo*, pages 769–775, 2010.

Leonardo Vanneschi and Giuseppe Cuccu. A study of genetic programming variable population size for dynamic optimization problems. In *International Conference on Evolutionary Computation (ICEC)*, pages 119–126, 2009a.

Leonardo Vanneschi and Giuseppe Cuccu. Variable size population for dynamic optimization with genetic programming. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, GECCO '09, pages 1895–1896. ACM, 2009b. ISBN 978-1-60558-325-9. doi: 10.1145/1569901.1570222.

Leonardo Vanneschi and Giuseppe Cuccu. Reconstructing dynamic target functions by means of genetic programming using variable population size. In *Computational Intelligence*, volume 343, pages 121–134. Springer, Berlin, Heidelberg, 2011. doi: 10.1007/978-3-642-20206-3_8.

Frank Weidner, Moritz Lipperheide, Manfred C. Wirsum, Stefano Bernero, and Martin Gassner. Pulsations in gas turbine operation: Identification and modeling with the purpose of online engine monitoring and optimization. In *ASME Turbo Expo 2017: Turbomachinery Technical Conference and Exposition*, pages V04BT04A011–V04BT04A011. American Society of Mechanical Engineers, 2017.

Paul J Werbos. Applications of advances in nonlinear sensitivity analysis. In *System modeling and optimization*, pages 762–770. Springer, 1982.

Daan Wierstra, Tom Schaul, Jan Peters, and Jürgen Schmidhuber. Natural evolution strategies. In *Evolutionary Computation, 2008. CEC 2008.(IEEE World Congress on Computational Intelligence). IEEE Congress on*, pages 3381–3387. IEEE, 2008.

Daan Wierstra, Tom Schaul, Tobias Glasmachers, Yi Sun, Jan Peters, and Jürgen Schmidhuber. Natural evolution strategies. *Journal of Machine Learning Research*, 15:949–980, 2014a.

Daan Wierstra, Tom Schaul, Tobias Glasmachers, Yi Sun, Jan Peters, and Jürgen Schmidhuber. Natural evolution strategies. *Journal of Machine Learning Research*, 15(1):949–980, 2014b.

Herman Wold. *Estimation of principal components and related models by iterative least squares*. Academic Press, 1966.

Brian G. Woolley and Kenneth O. Stanley. Evolving a single scalable controller for an octopus arm with a variable number of segments. In *PPSN (2)*, pages 270–279, 2010.

Georgios N. Yannakakis and Julian Togelius. *Artificial Intelligence and Games*. Springer, 2018. http://gameaibook.org.

Xin Yao. A review of evolutionary artificial neural networks. *International Journal of Intelligent Systems*, 4:203–222, 1993. URL http://citeseer.nj.nec.com/yao93review.html.

Xin Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.

Yoram Yekutieli, Roni Sagiv-Zohar, Ranit Aharonov, Yaakov Engel, Binyamin Hochner, and Tamar Flash. Dynamic model of the octopus arm. I. Biomechanics of the octopus reaching movement. *Journal of neurophysiology*, 94(2):1443–1458, 2005.

Sun Yi, Daan Wierstra, Tom Schaul, and Jürgen Schmidhuber. Stochastic search using the natural gradient. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1161–1168. ACM, 2009.

Jingqiao Zhang and Arthur C. Sanderson. Self-adaptive differential evolution with fast and reliable convergence performance. In *Proceedings of the IEEE Congress on Evolutionary Computation (CEC'07)*, pages 2251–2258, 2007.

Zheng Zhang, Yong Xu, Jian Yang, Xuelong Li, and David Zhang. A survey of sparse representation: algorithms and applications. *IEEE access*, 3:490–530, 2015.

Giuseppe Cuccu giuse@exascale.info
Applied Machine Learning Researcher exascale.info/giuse

Objectives

Looking to apply and advance the state of the art of Machine Learning in applications with tight constraints and limited resources. Searching for challenging problems with real-world impact. Self reliant and independent but very sociable, favor teamwork and collaborations.

Education

2018: **Ph.D. in Machine Learning** (Faculty of Computer Science), University of Fribourg (CH).
Thesis: Extending the Applicability of Neuroevolution. *Expected graduation: September 2018*
Advisor: Prof. Dr. Philippe Cudré-Mauroux

2009: M.Sc. in Computer Science (major: Machine Learning), University of Milan-Bicocca (IT).
Thesis: Variable Size Populations in Genetic Programming. *Summa cum Laude*

Professional Experience

2017-current: *Research Assistant*, University of Fribourg (CH), eXascale Infolab
Evolving Neural Networks for Reinforcement Learning Control Problems *(w/ New York University).*

2016-2017: *Research Assistant*, Institute of Applied Simulations, ZHAW, Zurich (CH),
Predictive and Prescriptive Analytics for Gas Turbine Engines *(w/ General Electrics).*

2015-2016: *Software Engineer*, Technology Astronauts, Bern (CH)
Member of a small team doing advanced software development for medium-size companies.

2013-2015: *Founder and CEO*, Partytech, Lugano (CH)
Machine learning-augmented social events.

2009-2013: *Research Assistant*, Dalle Molle Institute for Artificial Intelligence (IDSIA), Lugano (CH)
Evolving Neural Networks for Reinforcement Learning Control Problems.

Selected Publications

Playing Atari with Six Neurons. **G. Cuccu**, J. Togelius, P. Cudré-Mauroux. ArXiv e-Prints, 2018. *Submitted.*

A Data-Driven Approach to Predict NOx-Emissions of Gas Turbines. **G. Cuccu**, S. Danafar, P. Cudré-Mauroux, M. Gassner, S. Bernero, K. Kryszczuk. IEEE Big Data, 2017. Boston, MA, US.

Intrinsically Motivated Neuroevolution for Vision-Based Reinforcement Learning. **G. Cuccu**, M. Luciw, J. Schmidhuber, and F. Gomez. IEEE International Conference On Development and Learning (ICDL), 2011. Taormina, IT.

Reconstructing Dynamic Target Functions by Means of Genetic Programming Using Variable Population Size. L. Vanneschi and **G. Cuccu**. *Book chapter:* Studies in Computational Intelligence, 2011, Volume 343/2011, 121-134.

Awards

FDG 2013 Runner-up Best Paper Award
ICEC 2009 Best Paper Award
M.Sc. Fellowship 2007-2009
B.Sc. Fellowship 2003-2006

Further Information

Languages: Proficient in English and Italian, with technical writing experience.
Intermediate French, basic German and Spanish.

Hobbies: Avid reader, all-year motorbike rider, several sports depending on season.

References: Available upon request.

August 2018