Technische Universität München
Lehrstuhl VI: Echtzeitsysteme und Robotik

Studies in Continuous Black-box Optimization

Tom Schaul

Vollständiger Abdruck der von der Fakultät für Informatik der Technischen Universität München zur Erlangung des akademischen Grades eines

Doktors der Naturwissenschaften (Dr. rer. nat.)

genehmigten Dissertation.

Die Dissertation wurde am 06.06.2011 bei der Technischen Universität München eingereicht und durch die Fakultät für Informatik am 28.06.2011 angenommen.

Abstract

Optimization is the research field that studies that studies the design of algorithms for finding the best solutions to problems we humans throw at them. While the whole domain is of important practical utility, the present thesis will focus on the subfield of *continuous black-box optimization*, presenting a collection of novel, state-of-the-art algorithms for solving problems in that class.

First, we introduce a general-purpose algorithm called *Natural Evolution Strategies* (NES). In contrast to typical evolutionary algorithms which search in the vicinity of the fittest individuals in a population, evolution strategies aim at repeating the type of *mutations* that led to those individuals. We can characterize those mutations by a *search distribution.* The key idea of NES is to ascend the gradient on the parameters of that distribution towards higher expected fitness. We show how plain gradient ascent is destined to fail, and provide a viable alternative that instead descends along the *natural gradient* to adapt the search distribution, which appropriately normalizes the update step with respect to its uncertainty. Being derived from first principles, the NES approach can be extended to all types of search distributions that allow a parametric form, not just the classical multivariate Gaussian one. We derive a number of NES variants for different distributions, and show how they are useful on different problem classes. In addition, we rein in the computational cost, avoiding costly matrix inversions through an incremental change of coordinates. Two additional, novel techniques, *importance mixing* and *adaptation sampling*, allow us to automatically tune the learning rate and batch size to the problem, and thereby further reduce the average number of required fitness evaluations. A third technique, *restart strategies*, provides the algorithm with additional robustness in the presence of multiple local optima, or noise.

Second, we introduce a new approach to *costly* black-box optimization, when fitness evaluations are very expensive. Here, we model the fitness function using state-of-the-art Gaussian process regression, and use the principle of *artificial curiosity* to direct exploration towards the most informative next evaluation candidate. Both the expected fitness improvement and the expected information gain can be derived explicitly from the Gaussian process model, and our method constructs a front of Pareto-optimal points according to these two criteria. This makes the exploration-exploitation trade-off explicit, and permits maximally informed candidate selection.

We empirically validate our algorithms on a broad set of benchmarks. We also include a study of how continuous black-box optimization can solve challenging neuro-evolution problems, where multi-dimensional recurrent neural networks are trained to play the *game of Go*. At the same time, we establish to what degree those network architectures can transfer good playing behavior from small to large board sizes.

In summary, this dissertation presents a collection of novel algorithms, for the general problem of continuous black-box optimization as well as a number of special cases, each validated empirically.

Dedicated to

Joé Schaul

In memory of

Adina Mosincat

Table of Contents

Abstract . . . . **iii**

Table of Contents . . . . **vii**

List of Tables . . . . **xi**

List of Figures . . . . **xiii**

List of Algorithms . . . . **xv**

Preface . . . . **xvii**

0.1 Thesis Outline xvii
0.2 Related Publications xviii
0.3 Notation xxi

Acknowledgments . . . . **xxiii**

1 Introduction: Continuous Black-box Optimization **1**

1.1 Problem Definition 1
1.1.1 Continuous Optimization: Real-valued Solution Space . . 2
1.1.2 Black-box Optimization: Unknown Function 2
1.1.3 Local Optimization: Find a Nearby Optimum 3
1.1.4 Noisy Optimization: Functions Corrupted by Noise 3
1.1.5 Multi-objective Optimization 4
1.2 Evaluating Optimization Algorithms 5
1.3 State-of-the-Art Approaches 6
1.3.1 Evolutionary Methods 6
1.3.2 Response Surface Methods 7
1.4 Impact and Applications 8
1.5 Related Problem Domains 8
1.6 Open Questions 9

2 Natural Evolution Strategies **11**

2.1 The NES Family 11
2.1.1 Chapter Outline 12
2.2 Search Gradients 13

2.2.1 Search Gradients for Gaussian Distributions 14
2.2.2 Limitations of Plain Search Gradients 15
2.3 Using the Natural Gradient 16
2.4 Performance and Robustness Techniques 19
2.4.1 Fitness Shaping 20
2.4.2 Fitness Baselines 20
2.4.3 Importance Mixing 22
2.4.4 Adaptation Sampling 24
2.4.5 Restart Strategies 25
2.5 Techniques for Multinormal Distributions 28
2.5.1 Using Exponential Parameterization 28
2.5.2 Using Natural Coordinates 29
2.5.3 Orthogonal Decomposition of Multinormal Parameter Space 31
2.5.4 Connection to CMA-ES 32
2.5.5 Elitism 35
Use for Multi-objective NES 37
2.6 Beyond Multinormal Distributions 38
2.6.1 Separable NES 38
2.6.2 Rotationally-symmetric Distributions 40
Sampling from Radial Distributions 41
Computing the Fisher Information Matrix 41
2.6.3 Heavy-tailed NES 42
Groups of Invariances 43
Cauchy Distribution 44
2.7 Experiments 45
2.7.1 Experimental Setup and Hyperparameters 46
2.7.2 Black-box Optimization Benchmarks 47
2.7.3 Separable NES 48
Separable and Non-separable Benchmarks 48
Neuro-evolution 53
Lennard-Jones Potentials 54
2.7.4 Heavy Tails and Global Optimization 56
2.7.5 Results Summary 57
2.8 Discussion 58

3 Curiosity-driven Optimization 61
3.1 Artificial Curiosity 61
3.1.1 Background 62
3.1.2 Artificial Curiosity as a Guide for Optimization 62
3.1.3 Formal Framework 63
3.2 Exploration/Exploitation Trade-off. 63
3.3 Curiosity-driven Optimization (General Form) 65
3.4 Models of Expected Fitness and Information Gain 65

3.4.1 A Good Model Choice: Gaussian Processes 66
3.4.2 Derivation of Gaussian Process Information Gain 66
3.5 Curiosity-driven Optimization with Gaussian Processes 67
3.6 Minimal Asymptotic Requirements 68
3.6.1 Reaching Optima at Arbitrary Distance 68
3.6.2 Locating Optima with Arbitrary Precision 69
3.6.3 Guaranteed to Find Global Optimum 69
3.7 Proof-of-concept 70
3.8 Discussion 71

4 A Study in Scalability 73
4.1 Scalability in Problem Size 73
4.2 Example Domain: the Game of Go 74
4.2.1 Rules of the Game 74
4.2.2 Challenges of Go 74
4.2.3 Simpler Variants of Go 75
4.2.4 Computer Opponents 76
4.3 Scalable Neural Architectures 76
4.3.1 Multi-dimensional RNN 77
4.3.2 Multi-dimensional LSTM 78
4.3.3 A Custom Architecture for Go 78
4.4 Experiments 79
4.4.1 Experimental Setup 79
4.4.2 Random-weight Networks 80
4.4.3 Optimized Networks 82
4.4.4 Coevolved Networks 84
4.5 Discussion 90

5 Conclusions 91
5.1 Perspectives 92
5.2 Closing Words 93

A Appendix: Implementations in Python 95

B Appendix: Weighted Mann-Whitney Test 97

Bibliography 99

Index 109

List of Tables

1 Notation and symbols used. xxi

2.1 Default parameters 46
2.2 Summary of enhancing techniques 59

4.1 Fitness correlations across board sizes (random networks) 80
4.2 Fitness correlations across board sizes (optimized networks) . . . 84
4.3 Dominance numbers found during coevolution 87

A.1 Implementations in PyBrain 96

List of Figures

1.1 Illustration of local and global optima 4

2.1 Lack of scale invariance 17
2.2 Illustration of plain vs. natural gradient 18
2.3 Illustration of importance mixing 23
2.4 Illustration of adaptation sampling 26
2.5 Illustration of restart strategies 27
2.6 Illustration of exponential map 29
2.7 Computation time per function evaluation 47
2.8 Unimodal benchmarks 49
2.9 Multi-modal benchmarks 50
2.10 Noisy benchmarks 51
2.11 Noisy benchmarks (continued) 52
2.12 SNES vs. xNES 53
2.13 Non-Markovian double-pole balancing 54
2.14 Lennard-Jones clusters 55
2.15 Lennard-Jones benchmark results 55
2.16 Double-funnel benchmark 57
2.17 Deceptive global optima 58

3.1 Escaping linear regions 69
3.2 Precisely locating optima 70
3.3 Global optimization on the Branin function 72

4.1 MDRNN structure diagram 77
4.2 Random-weight networks across different tasks 81
4.3 Random-weight networks across different architectures 82
4.4 Learning curves for different games (naive opponent) 83
4.5 Learning curves for training against different opponent strengths 85
4.6 Illustrations of fitness scalability on Atari-Go 86
4.7 Typical coevolutionary run 88
4.8 CIAO plot: champions of both populations 89
4.9 Illustrative games of coevolution champions 90

List of Algorithms

1 Canonical Search Gradient algorithm 14
2 Search Gradient algorithm: Multinormal distribution 15
3 Canonical Natural Evolution Strategies (NES) 19
4 Natural Evolution Strategies with baselines 22
5 Exponential Natural Evolution Strategies (xNES) 33
6 (1+1)-NES: radial Gaussian distribution 37
7 (1+1)-xNES: multi-variate Gaussian distribution 37
8 Separable NES (SNES) 40
9 (1+1)-NES with multi-variate Cauchy distribution 45

10 Curiosity-driven Optimization 65
11 Curiosity-driven Optimization with Gaussian processes (CO-GP) . 68

Preface

Research does not always proceed in a straight line, and my case is no exception. Driven by curiosity and with an enthusiasm triggered by different colleagues, I ended up working on a whole range of research topics. The result is a thesis which covers only a selection of this work, and is not completely homogeneous: Although all pertaining to continuous black-box optimization, the three core chapters are relatively self-contained, with only minimal overlap.

0.1 Thesis Outline

Chapter 1 introduces the field of continuous black-box optimization, illustrates its broad usefulness in a whole range of application domains and describes the major state-of-the-art approaches. It proceeds to carve out the open questions that in turn motivate the remainder of the dissertation.

Chapter 2 is the most substantial chapter (outlined in detail in section 2.1.1). It presents a novel family of black-box optimization algorithms, Natural Evolution Strategies (NES), that are very flexible, and derived from first principles. We present a number of concrete instantiations, each of these algorithms being appropriate for a different class of optimization problems. The chapter also contains a collection of techniques to reduce computational and sample complexity, as well as increase robustness.

Chapter 3 presents the principle of artificial curiosity and investigates how it can be employed to guide exploration, especially in the context of black-box optimization problems with high cost per function evaluation. The resulting algorithm relies on Gaussian process regression to determine the most informative points to explore.

Chapter 4 looks at an application of black-box optimization to the challenging problem of learning to play the game of Go, using a specialized recurrent neural networks architecture, which turns out to be highly scalable.

Chapter 5 concludes the dissertation, revisiting the open questions from section 1.6, and laying out a number of possible extensions and ideas for future work.

0.2 Related Publications

Most parts of this dissertation have already appeared as peer-reviewed articles. This section details for each chapter the publications it is based upon, including partially reused text, derivations, results and figures. As should be clear from the many different co-authors, the published work was not done solely by me. However, my contributions to each article were substantial, and in this dissertation, I empathized and expanded upon those elements of which I was the main contributor. Also, I conducted and analyzed all of the experiments presented here, and implemented the corresponding algorithms and benchmarks.

0.2.1 Incorporated in Chapter 2

The original work on Natural Evolution Strategies was done jointly with Daan Wierstra in 2007, and we then developed the consecutive improvements and extensions with significant help of two more collaborators, Sun Yi and Tobias Glasmachers.

- D. Wierstra, T. Schaul, J. Peters, and J. Schmidhuber. Natural Evolution Strategies. In *IEEE Congress on Evolutionary Computation (CEC)*, 2008a
- Y. Sun, D. Wierstra, T. Schaul, and J. Schmidhuber. Stochastic Search using the Natural Gradient. In *International Conference on Machine Learning (ICML)*, number 1, 2009b
- Y. Sun, D. Wierstra, T. Schaul, and J. Schmidhuber. Efficient Natural Evolution Strategies. In *Genetic and Evolutionary Computation Conference (GECCO)*, 2009a
  Best Paper Award
- T. Schaul and J. Schmidhuber. Towards Practical Universal Search. In *Conference on Artificial General Intelligence (AGI)*, Lugano, 2010b
- T. Rückstieß, F. Sehnke, T. Schaul, D. Wierstra, Y. Sun, and J. Schmidhuber. Exploring Parameter Space in Reinforcement Learning. *Paladyn Journal of Behvioral Robotics*, 1(1):14–24, 2010
- T. Glasmachers, T. Schaul, and J. Schmidhuber. A Natural Evolution Strategy for Multi-Objective Optimization. In *Parallel Problem Solving from Nature (PPSN)*, 2010a
- T. Glasmachers, T. Schaul, Y. Sun, D. Wierstra, and J. Schmidhuber. Exponential Natural Evolution Strategies. In *Genetic and Evolutionary Computation Conference (GECCO)*, Portland, OR, 2010b
  Nominated for Best Paper Award
- T. Schaul, T. Glasmachers, and J. Schmidhuber. High Dimensions and Heavy Tails for Natural Evolution Strategies. In *Genetic and Evolutionary Computation Conference (GECCO)*, Dublin, Ireland, 2011a

0.2.2 Incorporated in Chapter 3

The chapter on curiosity-driven optimization is based mainly on one paper, but the motivation is partly shared by other curiosity-related work.

- T. Schaul, Y. Sun, D. Wierstra, F. Gomez, and J. Schmidhuber. Curiosity-Driven Optimization. In *IEEE Congress on Evolutionary Computation (CEC)*, 2011c
- T. Schaul, L. Pape, T. Glasmachers, V. Graziano, and J. Schmidhuber. Coherence Progress: A Measure of Interestingness Based on Fixed Compressors. In *Fourth Conference on Artificial General Intelligence (AGI)*, 2011b

0.2.3 Incorporated in Chapter 4

The chapter on optimizing scalable neural networks is built upon the following two conference papers.

- T. Schaul and J. Schmidhuber. A Scalable Neural Network Architecture for Board Games. In *IEEE Symposium on Computational Intelligence in Games (CIG)*, 2008
- T. Schaul and J. Schmidhuber. Scalable Neural Networks for Board Games. In *International Conference on Artificial Neural Networks (ICANN)*, 2009

0.2.4 Incorporated in Appendix A

The code for all the benchmarks and algorithms of this thesis is part of the PyBrain open source library.

- T. Schaul, J. Bayer, D. Wierstra, Y. Sun, M. Felder, F. Sehnke, T. Rückstieß, and J. Schmidhuber. PyBrain. *Journal of Machine Learning Research*, 11:743–746, 2010

0.2.5 Other Publications

The following papers were published during the same time period, but are only tangentially related to the dissertation.

- J. Togelius, T. Schaul, J. Schmidhuber, and F. Gomez. Countering Poisonous Inputs with Memetic Neuroevolution. In *Parallel Problem Solving from Nature (PPSN)*. Springer-Verlag, 2008
- D. Wierstra, T. Schaul, J. Peters, and J. Schmidhuber. Fitness Expectation Maximization. In *Parallel Problem Solving from Nature (PPSN)*. Springer-Verlag, 2008b

- D. Wierstra, T. Schaul, J. Peters, and J. Schmidhuber. Episodic Reinforcement Learning by Logistic Reward-Weighted Regression. In *International Conference on Artificial Neural Networks (ICANN)*, 2008c
- J. Togelius, T. Schaul, D. Wierstra, C. Igel, F. Gomez, and J. Schmidhuber. Ontogenetic and Phylogenetic Reinforcement Learning. (3):30–33, 2009
- M. Grüttner, F. Sehnke, T. Schaul, and J. Schmidhuber. Multi-Dimensional Deep Memory Atari-Go Players for Parameter Exploring Policy Gradients. In *International Conference on Artificial Neural Networks (ICANN)*, 2010
- Y. Sun, T. Glasmachers, T. Schaul, and J. Schmidhuber. Frontier Search. In *Conference on Artificial General Intelligence (AGI)*, Lugano, 2010
  Kurzweil Prize for Best AGI Paper
- T. Schaul and J. Schmidhuber. Metalearning. *Scholarpedia*, 5(6):4650, 2010a
- A. A. de Moura Meneses, C. J. G. Pinheiro, P. Rancoita, T. Schaul, L. M. Gambardella, R. Schirru, R. C. Barroso, and L. F. de Oliveira. Assessment of neural networks training strategies for histomorphometric analysis of synchrotron radiation medical images. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 2010
- V. Graziano, T. Glasmachers, T. Schaul, L. Pape, G. Cuccu, J. Leitner, and J. Schmidhuber. Artificial Curiosity for Autonomous Space Exploration. *Acta Futura (in press)*, 2011
- M. B. Ring and T. Schaul. Q-error as a Selection Mechanism in Modular Reinforcement-Learning Systems. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2011

0.3 Notation

Table 1 lists a mixture of the notation used throughout the dissertation, as well as and commonly used symbols.

Table 1: Notation and symbols used.

Symbol	Description
$\mathbb{N}, \mathbb{Z}, \mathbb{R}$	set of natural, integer, real numbers
$\mathcal{S}$	search space
d	search space dimension
$f(\cdot)$	objective/fitness function
$\mathcal{O}(\cdot)$	order of (computational) complexity
$\delta(\cdot, \cdot)$	Kronecker delta function
$\Gamma(\cdot)$	Gamma function
$\mathbf{x}$	vector (bold)
$\\|\mathbf{x}\\|$	Euclidean norm of $\mathbf{x}$
$\mathbf{M}$	matrix (bold, capitalized)
$\mathbf{M}^\top$	transpose of $\mathbf{M}$
$\mathrm{tr}(\mathbf{M})$	trace of matrix $\mathbf{M}$
$\mathbb{I}$	identity matrix
$\mathbb{E}[\cdot]$	expectation
$\pi(\cdot)$	probability density
$\sim \mathcal{N}(\cdot, \cdot)$	sampling from Gaussian distribution
$\Phi(\cdot)$	cumulative Gaussian distribution function
$\phi(\cdot)$	Gaussian density function
$\mathbb{D}(\cdot\\|\cdot)$	Kullback-Leibler divergence
$\mathbf{F}$	Fisher information matrix
$H(\cdot), H(\cdot\vert\cdot)$	entropy, conditional entropy
$I(X;Y)$	mutual information
$\psi(\cdot\vert\cdot), \Psi(\cdot\vert\cdot)$	information gain, expected information gain
$\mathcal{G}$	Gaussian process
$k(\cdot, \cdot)$	kernel function
θ	search distribution parameters
J	objective (expected fitness)
$\nabla_\theta J$	plain gradient
$\widetilde{\nabla}_\theta J$	natural gradient
$\boldsymbol{\mu}$	mean vector
σ^2	variance
$\boldsymbol{\Sigma}$	covariance matrix
$\mathbf{z}$	sample point
λ	population size
η	learning rate

Acknowledgments

Many people have a share in this thesis coming about. First of all, I would like to thank my supervisor Jürgen Schmidhuber for his support, advice, and for leaving me the freedom to explore my topics of interest.

Most of my research has been collaborative, and this thesis would not be half of what it is without the insights, contributions and advice of these colleagues that have become friends: Daan Wierstra, Sun Yi, Tobias Glasmachers, Julian Togelius and Faustino Gomez.

I am grateful to all the people at IDSIA for making my stay truly enjoyable, on both a professional and a personal level; and in particular to Cinzia Daldini whose magic powers resolved all administrative issues. For stimulating discussions, memorable experiences and just generally the good times, I'd like to thank Alex, Anderson, Andreas, Bas, Cassio, Cyrus, Dan, Daniil, Dennis, Engin, Florian, Fred, Gianni, Giovanni, Giuse, Hung, Jan, Jan, Jörg, Jonathan, Juxi, Kail, Leo, Linus, Mark, Matt, Matteo, Niels, Nikos, Ola, Paola, Santi, Simon, Shane, Somayeh, Ueli, Varun, Vincent and all the others that transformed IDSIA into a great place to work, and Lugano into a great place to live.

I would like to thank the rest of the PyBrain team, namely Thomas Rückstieß, Frank Sehnke and especially Justin Bayer for speeding up the PyBrain neural network implementations to such a level that the extensive experiments became feasible; and also Mandy Grüttner for implementing the framework to interact with the heuristic Atari-Go player.

I thank the many anonymous reviewers that have invested their time to provide feedback on the papers that became part of the dissertation: Especially the more critical comments helped improve the quality a lot. A special thanks also to Jan, Claudia and Joé who patiently proofread the dissertation and helped iron out the kinks.

I am also grateful to the Swiss National Science Foundation that provided the principal funding for this research (grant number 200020-122124/1).

I am deeply indebted and thankful to my parents, for an upbringing and education infused with happiness, curiosity and freedom, and for supporting and encouraging me throughout all those years abroad.

Above all, I want to thank Claudia, for her energy, her patience and her love.

Chapter 1

Introduction: Continuous Black-box Optimization

If you refuse to accept anything but the best
you will very often get it.

W. Somerset Maugham

As the foundation for the entire thesis, this chapter formally introduces optimization, and presents a brief taxonomy of its variants, zooming in on continuous black-box optimization in particular. We discuss evaluation methods, and review some of the most pertinent state-of-the-art approaches. By showing the breadth of applicability, this chapter also provides context and motivation for the algorithms presented in the following chapters. Finally, we look at some of the unanswered questions of the field.

1.1 Problem Definition

The aim of optimization is to find the best solution to some type of problem, for example, finding the combination of brewing temperature and pressure that leads to the best coffee taste. Formally, function optimization is defined as finding the *best solution* $x_\star$ in a *search space* $\mathcal{S}$ (or solution space), which satisfies:

$$x_\star = \arg\max_{x \in \mathcal{S}} f(x),$$

where $f : \mathcal{S} \mapsto \mathbb{R}$ is called the *objective function* (also known as *fitness* function, see also Figure 1.1). In the coffee example, the search points x would be combinations of temperatures and pressures and f would be a subjective rating given by the taster. Together, the objective function f and the search space $\mathcal{S}$ define an *optimization problem.* An equivalent formulation describes the minimization case, where that $x_\star$ is sought that minimizes f, in which case the objective function is often called *cost function* (or energy potential).

The solution $x_\star$ does not need to be unique in $\mathcal{S}$, in which case it is sufficient to find one of them. If elements of $\mathcal{S}$ are composed of multiple variables we call those *decision variables*.

The ensuing subsections will specify a number of useful distinctions and special cases.

1.1.1 Continuous Optimization: Real-valued Solution Space

Continuous optimization is the class of optimization problems where $\mathcal{S}$ is a subset of $\mathbb{R}^d$ (thus d is the *dimension* of the solution space). In addition, $\mathcal{S}$ is generally assumed to be *connected* (i.e., cannot be partitioned into two open subsets). Note that the term 'continuous' refers to the continuous decision variables (that is, the components of x), and does not imply that f is a continuous function.

Working in a continuous solution space has both advantages and disadvantages: The fundamental problem is that the space is infinite, and therefore finding $x_\star$ in finite time can only be guaranteed in special cases. On the other hand, many realistic objective functions are relatively smooth, which permits a whole plethora of approaches that take advantage of this (e.g., gradient ascent methods), which cannot be applied on discrete solution spaces, so in practice, continuous problems are often solved faster than discrete ones (which are usually NP-hard).

The subfield complementary to continuous optimization is discrete optimization, and includes, among others, integer programming. The same dichotomy is sometimes expressed as *numerical optimization* (continuous) versus *combinatorial optimization* (discrete). Hybrid cases exist as well, where part of the solution x is continuous and part is discrete, e.g., mixed integer programming problems.

A *constrained* continuous search space is a subset of $\mathbb{R}^d$ bounded by a collection of inequalities (generally linear ones), which define the set of *feasible* solutions. Constraints add additional difficulty to optimization problems, in particular when the optimum lies on the search space boundary (i.e., is not an interior point). In this case, optimization methods require an initial feasible solution x_0 from which to start the optimization. Entire research fields are dealing with the issues specific to constrained optimization (most notably, convex optimization), but approaches designed for unconstrained optimization can be applied too: for example, the objective function can be modified to include a *penalty term* that keeps the solutions found within the feasible region, after which continuous optimization techniques can be employed to solve the constrained problem.

1.1.2 Black-box Optimization: Unknown Function

While optimization was originally developed for objective functions that were known, but where the optimum could not be calculated analytically, some of the algorithms apply equally well to the case where the function is unknown, which spawned the subfield of *black-box optimization* (alluding to the fact that the objective function is like a black box, we know nothing about its internals, can only observe what goes in, and what comes out). The field is also known as direct search, or derivative-free optimization, because one of the key unavailable pieces of information is the derivative of the function, which many other optimization methods rely on; it is closely related to the field of *metaheuristics*. Also, most algorithms for black-box optimization fall into the category of *randomized search methods*, which introduce randomness in the search process.

It is worth noting that while the methods of black-box optimization were developed for problems where the function is unknown, they are sometimes applied successfully when the analytical form *is* available. Prominent examples are non-differentiable functions, deceptive functions (where gradient information might lead the search astray) and functions with uninformative gradients (e.g., piece-wise constant).

1.1.3 Local Optimization: Find a Nearby Optimum

Many continuous optimization problems are *multi-modal*, that is, there exist multiple solutions x_+ that are locally optimal:

$$f(x_+) = \max_{||x-x_+||<\epsilon} f(x)$$

for some small radius ϵ (see also Figure 1.1). In general, only one local optimum is also the global optimum. It is challenging to determine whether the best local optimum found so far is $x_\star$, but for some purposes a local optimum is a sufficient result. We thus distinguish *global optimization* methods, which aim at finding the true optimum $x_\star$, and *local optimization* methods which contend themselves with a (reasonably good) local optimum. A local optimization algorithm with a randomized component can be restarted multiple times, in order to increase the probability of hitting the global optimum, or just reaching a better local optimum.

1.1.4 Noisy Optimization: Functions Corrupted by Noise

In many real-world problems, it is unrealistic to assume that we can measure the objective function f precisely, as it is easily corrupted by measurement or process noise, and multiple measurements of $f(x)$ do not give the same result (for a given x). Noisy functions can be decomposed into a deterministic component f_{det} and a stochastic function g_{sto}:

$$f(x) \sim g_{sto}(x, f_{det}(x))$$

Some black-box optimization methods are ill-equipped to deal with noise

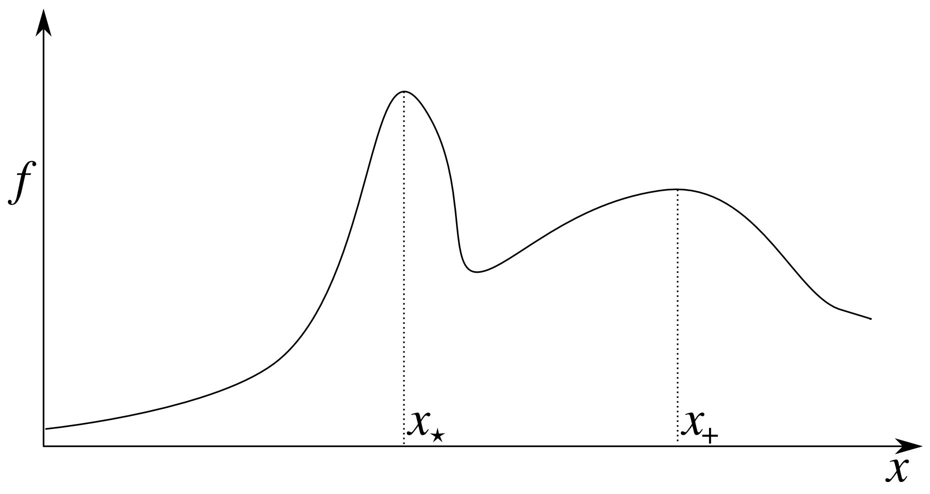


Figure 1.1: Illustration of a fitness function $f(x)$ (in this case, continuous and one-dimensional $\mathcal{S} = \mathbb{R}$) with the global optimum at $x_\star$ and a local optimum at x_+.

(see Auger et al., 2010), which underlines the need for robust approaches.

1.1.5 Multi-objective Optimization

One of the recent extensions of the optimization framework has been from single-objective to *multi-objective* optimization (MOO, also known as multi-criteria optimization, Deb et al., 2002), where the aim is to maximize multiple objective functions $f_1, \ldots, f_m$ of the same solution space, simultaneously. The solution to a multi-objective optimization problem is no longer a single point $x_\star$, but a (potentially infinite) set $X_\star \in \mathcal{S}$, known as the *Pareto optimum* (or Pareto-front), which includes all elements of $\mathcal{S}$ that are *non-dominated*, that is

$$\forall x \in X_\star, \nexists y \in \mathcal{S}, s.t. \begin{cases} f_i(y) > f_i(x) & \\ f_j(y) \geq f_j(x) & \forall j \neq i \end{cases},$$

In other words, if x is Pareto-optimal, then no other point y can be better or equal on all objectives and strictly better on at least one objective.

Many problems can very naturally be viewed as having multiple objectives (e.g., minimizing fuel consumption while simultaneously maximizing speed). While traditionally those objectives have been traded off into a single objective to be optimized, a number of arguments can be put forward in favor of handling the multiple objectives explicitly. For one, in many practical applications it is more advantageous to find and then choose from the Pareto-front, rather than deciding upon a trade-off a priori and then maximizing it. Furthermore, in multi-objective evolutionary methods the diversity of encountered solutions is larger than for single-objective optimization with fixed trade-offs, which in turn can improve over the single-objective optimization performance at its own game, as it may allow the search to circumnavigate local optima (Knowles et al., 2001).

For the remainder of this dissertation, we will no longer concern ourselves with discrete or constrained solution spaces, nor with known objective functions, and thus take the term "optimization" to refer to the continuous, unconstrained black-box optimization, if not specified otherwise. Most of the problems under consideration are noise-free, but some have their objective functions corrupted by noise (section 2.7.2). Also, we will mostly consider single objectives, but in a couple of instances extend our work to multi-objective optimization (e.g., sections 16 and 3.2).

1.2 Evaluating Optimization Algorithms

To evaluate and compare different optimization algorithms on the same problem, we have basically three criteria that the algorithm should minimize:

1. The expected (or worst-case) *number of function evaluations* n until the optimum is found (and directly related, the probability of finding the optimum within a specified number of evaluations),

2. the expected (or worst-case) total *computation time* until the optimum is found,

3. the expected (or worst-case) *cumulative regret* after evaluating candidate solutions x_1 to x_n, which is

$$R_n = \sum_{i=1}^{n} f(x_\star) - f(x_i)$$

   in other words, the difference between the sum of fitnesses (or costs) during the search, and the best achievable sum if the optimum had been known in advance.

Note that the desired optimum in all those cases does not need to be the true optimum, but could be a local optimum x_+, or an ϵ-approximation $f(\hat{x_\star}) \geq f(x_\star) - \epsilon$.

For some applications, some of these measures will be more useful than others. In the present work, we will mostly focus on the first measure, because it is the simplest one, and independent of implementation and hardware questions (unlike the second). Also, in contrast to using expected regret, counting function evaluations puts the emphasis on the final outcome of the optimization.

There is a class of problems that deserves special attention because it explicitly demands the minimization of the number of function evaluations, where each evaluation is very costly: we speak of *costly optimization* (or expensive optimization), which covers many real-world optimization problems (see Section 1.4 for examples). For this class, even a small reduction in the required number of evaluations justifies a significant investment of computational resources.

Analyzing the *computational complexity* of a single iteration of an algorithm can be another fruitful measure, especially when the algorithm under consideration is applied to high-dimensional search spaces. The measure of computation time (2) contains this information implicitly.

Convergence analysis allows analyzing and comparing algorithms on a theoretical level. Assume that

$$\lim_{t\to\infty} \frac{|f(x_{t+1}) - f(x_\star)|}{|f(x_t) - f(x_\star)|^p} = c$$

holds for some constants $0 \leq c < 1$ and $p \geq 1$, where x_t are the candidate solutions evaluated by the algorithm in iteration t. If $p > 1$, then we have *superlinear* convergence, and call p the *order of convergence*. If $c = 1$ and the algorithm converges (i.e. $\lim_{t\to\infty} |f(x_t) - f(x_\star)| = 0$), the convergence is *sublinear*. Otherwise ($0 < c < 1$ and $p = 1$), we speak of *linear* convergence, and c is called the *rate of convergence*.

Clearly, evaluating and comparing algorithms on a single problem is not sufficient to determine their quality, as much of their benefit lies in their performance generalizing to large classes of problems. One of the goals of research in optimization is, arguably, to provide practitioners with reliable, powerful and general-purpose algorithms. This is why we test our algorithms on a whole battery of benchmark functions, taken from different problem classes.

1.3 State-of-the-Art Approaches

Here, we will briefly review the spectrum of methods that have been applied to continuous, unconstrained black-box optimization (some of which are more generally applicable). Attempting to be exhaustive would be very difficult given the breadth of the field[1]. Instead we will try to point out the most representative algorithms, going into the most depth for those approaches that are currently producing state-of-the-art results, and that are most closely related to our own.

A first class of methods was inspired by classic optimization methods, including simplex methods such as Nelder-Mead (Nelder and Mead, 1965), as well as members of the quasi-Newton family of algorithms (Fletcher, 1987). Simulated annealing (Kirkpatrick et al., 1983), a popular method introduced in 1983, was inspired by thermodynamics, and is in fact an adaptation of the Metropolis-Hastings algorithm (Hastings, 1970; Metropolis et al., 1953).

1.3.1 Evolutionary Methods

Most prominent among the more heuristic methods are those inspired by *evolution*, developed from the early 1950s on, including the broad class of genetic

[1]Also, it turns out that some of the approaches in metaheuristics, derived from colorful analogies, can be reduced entirely to much earlier work; for a discussion of the striking case of 'Harmony Search', a veiled derivative of evolution strategies, see Weyland, 2010.

algorithms (GA; Goldberg, 1989; Holland, 1992). Mimicking natural evolution (Darwin, 1859), these approaches maintain a 'population' of 'individuals' (search points) that are evaluated in batch. In each 'generation' the best individuals (those with the highest fitness f, the 'survivors') are *selected* to 'procreate' and produce offspring (i.e., the population in the next generation), while the others are discarded. Procreation can introduce small-scale changes to the the individuals' *genotype* x ('mutations'), or perform large-scale recombinations of individuals ('cross-over'). Among the most successful GA for continuous optimization are differential evolution (Storn and Price, 1997) where the mutations are based upon the difference vectors between the members of the population, and the related particle swarm optimization (Kennedy and Eberhart, 2001).

Estimation of distribution algorithms (EDA; Larrañaga, 2002) on the other hand rely on modeling the population of (fittest) search points by a distribution, from which new individuals are then drawn; in contrast to GA, using only the estimated distribution and not the old population. Among its representatives are Estimation of Multivariate Normal Algorithm (EMNA) and the closely related cross-entropy method (CEM; Rubinstein and Kroese, 2004), where the search distribution is a multivariate Gaussian, as well as Fitness Expectation-Maximization (FEM; Wierstra et al., 2008b) where the distribution is updated using an expectation-maximization approach.

Evolution strategies (ES), introduced by Ingo Rechenberg and Hans-Paul Schwefel in the 1960s and 1970s (Rechenberg and Eigen, 1973; Schwefel, 1977), were designed to cope with high-dimensional continuous-valued domains and have remained an active field of research for more than four decades (Beyer and Schwefel, 2002). They are distinct from GA in that mutations modify each of the continuous decision variables simultaneously, but only very slightly; this process, after several generations, was shown to lead to reasonable to excellent results for many difficult continuous optimization problems. The algorithm framework has been developed extensively over the years to include self-adaptation of the search parameters, and the representation of correlated mutations by the use of a full covariance matrix. This allowed the framework to capture interrelated dependencies by exploiting the covariances while 'mutating' individuals for the next generation. The culminating algorithm, covariance matrix adaptation evolution strategies (*CMA-ES*; Hansen and Ostermeier, 2001, see also section 2.5.4 for more detail), has proven successful in numerous studies (e.g., Friedrichs and Igel, 2005; Muller et al., 2002; Shepherd et al., 2006).

1.3.2 Response Surface Methods

The standard tool for global optimization are *response surface methods* (RSM; Box and Wilson, 1951; Moore and Schneider, 1996). They store all available evaluations (some possibly given in advance) and use them to model the cost function, which useful for dimensionality reduction, visualization, assessing uncertainty, and ultimately determining good points to explore (Booker et al., 1998; Jones et al., 1998). A multitude of regression techniques have been used for modeling the response surface, from the original polynomials (Box and Wilson, 1951) to more recent Gaussian processes (Jones, 2001; Rasmussen and Williams, 2006). In addition, a statistical model of the cost function allows expert knowledge to be incorporated in the form of a Bayesian prior.

1.4 Impact and Applications

Many real world optimization problems that are too difficult or complex to model directly have been solved in using black-box optimization. Here, we give a selection of work in different fields that illustrate the very broad applicability of the framework.

In health sciences, optimization techniques were employed for matching CT-scans to ultra-sound images (Winter et al., 2005), and for forensic identification (Ibáñez et al., 2009). In chemistry, black-box optimization were used for chromatography (Jebalia et al., 2007) and finding stable crystalline structures (Wales and Doye, 1998). In urban development, black-box methods have helped optimize resource flows (Kämpf and Robinson, 2009), groundwater quality (Bayer et al., 2007), power distribution (Miguez et al., 1998; Wang et al., 2007) and radio network design (Mendes et al., 2006). Furthermore, optimization techniques can aid in determining appropriate features, for example for speaker identification (Charbuillet et al., 2009). In space research, they have been utilized, among others, for evolving orbit transfer maneuvers (Minisci and Avanzini, 2008) and docking approaches (Leitner et al., 2010). They also have a long tradition in device design (e.g. Klockgether and Schwefel, 1970) or aeronautics (Booker et al., 1998; Hasenjäger et al., 2005), as well as control (Hansen et al., 2009).

Besides the broad range of direct applications, black-box optimization is also a common component of other machine learning techniques. It is used to train neural networks ('neuro-evolution', see e.g. Gomez et al., 2008, or sections 2.7 and 4.4), to optimize kernel parameters (model selection, see Friedrichs and Igel, 2005, or section 7) or as a component of multi-objective optimization algorithms (Igel et al., 2007, or section 16).

1.5 Related Problem Domains

Black-box optimization is closely related to a number of other domains, so that techniques and enhancements developed for optimization can often be transfered or adapted to one of them, or vice-versa.

Reinforcement Learning. The goal of (RL; Sutton and Barto, 1998) is to optimize the behavior of an agent that interacts with an environment without being told the correct behavior (as in supervised learning); instead, the agent is only informed about how well it did, in terms of a scalar value called *reward*. There is a double link between RL and optimization. On one hand, we may consider optimization to be a simple sub-problem of RL, with only a single environment state and a single time-step per episode, where the fitness corresponds directly to the reward (i.e., a bandit problem). On the other hand, more interestingly, the return of a whole RL *episode* can be interpreted as a single fitness evaluation, where the search space $\mathcal{S}$ is that of policy parameters. In this case, parameter-based exploration in RL is equivalent to black-box optimization. Moreover, when the exploration in parameter space is normally distributed, a direct link between RL and evolution strategies can be established.

Active Learning. In active learning, the aim is to actively query the label of an unlabeled data point, in such a way that the classifier or predictor improves maximally. This procedure is closely related to expensive global optimization, in that each search point to be evaluated is chosen with care (Krause and Guestrin, 2007). The goal is different, however: active learning is concerned with obtaining an accurate model of the data, while modeling is secondary in optimization, and only useful inasmuch as it facilitates locating optima more efficiently. Therefore, active learning cannot be used naively for optimization.

Markov chain Monte Carlo. Sampling algorithms, in particular the widely used class of Markov chain Monte Carlo methods (Andrieu et al., 2003), are related to optimization in the sense that seeking out high-probability areas (modes) in a multi-dimensional space is very similar to finding high fitness values in optimization; and in fact differential evolution techniques have already been applied to sampling (Braak, 2006). The crucial difference however is that in optimization, the goal is not to produce collection of samples that correspond to the distribution, but only to find the best search point.

1.6 Open Questions

Approaches to continuous black-box optimization are far from unified. While evolution strategies prove to be among the most effective framework, their ad hoc procedures remain heuristic in nature. Thoroughly analyzing the actual dynamics of the procedure turns out to be difficult, the considerable efforts of various researchers notwithstanding (Auger, 2005; Beyer, 2001; Jebalia et al., 2010). In other words, ES (including CMA-ES), while powerful, still lack a clear derivation from first principles. However, for a black-box optimization to become a true end-user product, it should be reliable, understood, flexible and efficient. In chapter 2, therefore, we will concern ourselves with the question:

Can we design a family of algorithms that are derived from first principles, which reach state-of-the-art performance, while at the same

> *time being flexible enough to permit variants for a whole range of problem classes?*

For domains where data remains expensive and scarce, it is promising to harness the ever-cheaper computational resources for analyzing which search point to evaluate next. One principle to guide this analysis, which has not yet been investigated in this context is *artificial curiosity* (Schmidhuber, 1991, 2007). Thus, in chapter 3, we will ask:

> *Can artificial curiosity be an effective guiding principle for determining the most informative search points in costly optimization?*

Chapter 2

Natural Evolution Strategies

You can't help someone get up a hill
without getting closer to the top yourself.

H. Norman Schwarzkopf

Natural Evolution Strategies (NES) are a family of algorithms that constitutes a more principled approach to black-box optimization than established evolutionary algorithms. NES maintains a parameterized distribution on the set of solution candidates, and the natural gradient is used to update the distribution's parameters in the direction of higher expected fitness. We introduce a collection of techniques that address issues of convergence, robustness, sample complexity, computational complexity and sensitivity to hyperparameters. This chapter explores a number of implementations of the NES family, ranging from general-purpose multi-variate normal distributions to heavy-tailed and separable distributions tailored towards global optimization and search in high-dimensional spaces, respectively. Our results show best published performance on various standard benchmarks, as well as competitive performance on others.

2.1 The NES Family

Natural Evolution Strategies (NES) are well-grounded continuous black-box optimization algorithms, which instead of maintaining a population of search points, iteratively update a *search distribution*. Like CMA-ES, they can be cast into the framework of evolution strategies.

The general procedure is as follows: the parameterized search distribution is used to produce a batch of search points, and the fitness function is evaluated at each such point. The distribution's parameters (which include strategy parameters) allow the algorithm to adaptively capture the (local) structure of the fitness function. For example, in the case of a Gaussian distribution, this comprises the mean and the covariance matrix. From the samples, NES estimates a search gradient on the parameters towards higher expected fitness. NES then performs a gradient ascent step along the *natural gradient*, a second-order method which, unlike the plain gradient, renormalizes the update w.r.t. uncertainty. This step is crucial, since it prevents oscillations, premature convergence, and undesired effects stemming from a given parameterization. The entire process reiterates until a stopping criterion is met.

All members of the 'NES family' operate based on the same principles. They differ in the type of distribution and the gradient approximation method used. Different search spaces require different search distributions; for example, in low dimensionality it can be highly beneficial to model the full covariance matrix. In high dimensions, on the other hand, a more scalable alternative is to limit the covariance to the diagonal only. In addition, highly multi-modal search spaces may benefit from more heavy-tailed distributions (such as Cauchy, as opposed to the Gaussian). A last distinction arises between distributions where we can analytically compute the natural gradient, and more general distributions where we need to estimate it from samples.

2.1.1 Chapter Outline

As this chapter is the longest one of the dissertation, we briefly outline its structure. Section 2.2 presents the general idea of search gradients, outlining how to perform stochastic search using parameterized distributions while doing gradient ascent towards higher expected fitness. The limitations of the plain gradient are exposed in section 2.2.2, and subsequently addressed by the introduction of the natural gradient (section 2.3), resulting in the canonical NES algorithm.

Section 2.4 then regroups a collection of techniques that enhance NES's performance and robustness. This includes fitness shaping (designed to render the algorithm invariant w.r.t. order-preserving fitness transformations, section 2.4.1), importance mixing (designed to recycle samples so as to reduce the number of required fitness evaluations, section 2.4.3), adaptation sampling, which is a novel technique for adjusting learning rates online (section 2.4.4), and finally restart strategies, designed to improve success rates on multi-modal problems (section 2.4.5).

In section 2.5, we look in more depth at multivariate Gaussian search distributions, constituting the most common case. We show how to constrain the covariances to positive-definite matrices using the exponential map (section 2.5.1), and how to use a change of coordinates to reduce the computational complexity from $\mathcal{O}(d^6)$ to $\mathcal{O}(d^3)$, resulting in the xNES algorithm (section 2.5.2). We also investigate the connection to CMA-ES in depth (section 2.5.4) and introduce a hill-climber variant (section 2.5.5).

Next, in section 2.6, we develop the breadth of the framework, motivating its usefulness and deriving a number of NES variants with different search distributions. First, we show how a restriction to diagonal parameterization permits the approach to scale to very high dimensions due to its linear complexity (section 2.6.1). Second, we provide a novel formulation of NES for the whole class of multi-variate versions of distributions with rotation symmetries (section 2.6.2), including heavy-tailed distributions with infinite variance, such as the Cauchy distribution (section 2.6.3).

The ensuing experimental investigations show the competitiveness of the approach on a broad range of benchmarks (section 2.7). The chapter ends with a discussion on the effectiveness of the different techniques and types of distributions (section 2.8).

2.2 Search Gradients

The core idea of Natural Evolution Strategies is to use a *search gradient* to update the parameters of the search distribution. We define the search gradient as the sampled gradient of expected fitness. The search distribution can be taken to be a multinormal distribution, but could in principle be any distribution for which we can find derivatives of its log-density w.r.t. its parameters. For example, useful distributions include Gaussian mixture models and the Cauchy distribution with its heavy tail.

If we use θ to denote the parameters of distribution $\pi(\mathbf{z}\,|\,\theta)$ and $f(x)$ to denote the fitness function for samples $\mathbf{z}$, we can write the expected fitness under the search distribution as

$$J(\theta) = \mathbb{E}_\theta[f(\mathbf{z})] = \int f(\mathbf{z})\ \pi(\mathbf{z}\,|\,\theta)\ d\mathbf{z}. \tag{2.1}$$

and rewrite the derivatives as:

$$\begin{aligned}
\nabla_\theta J(\theta) =& \nabla_\theta \int f(\mathbf{z})\ \pi(\mathbf{z}\,|\,\theta)\ d\mathbf{z}\\
=& \int f(\mathbf{z})\ \nabla_\theta \pi(\mathbf{z}\,|\,\theta)\ d\mathbf{z}\\
=& \int f(\mathbf{z})\ \nabla_\theta \pi(\mathbf{z}\,|\,\theta)\ \frac{\pi(\mathbf{z}\,|\,\theta)}{\pi(\mathbf{z}\,|\,\theta)}\ d\mathbf{z}\\
=& \int \Big[f(\mathbf{z})\ \nabla_\theta \log \pi(\mathbf{z}\,|\,\theta)\Big]\ \pi(\mathbf{z}\,|\,\theta)\ d\mathbf{z}\\
=& \mathbb{E}_\theta\left[f(\mathbf{z})\ \nabla_\theta \log \pi(\mathbf{z}\,|\,\theta)\right].
\end{aligned}$$

From this form we obtain the Monte Carlo estimate of the search gradient from samples $\mathbf{z}_1 \ldots \mathbf{z}_\lambda$ as

$$\nabla_\theta J(\theta) \approx \frac{1}{\lambda} \sum_{k=1}^{\lambda} f(\mathbf{z}_k)\ \nabla_\theta \log \pi(\mathbf{z}_k\,|\,\theta), \tag{2.2}$$

where λ is the population size. This gradient on expected fitness provides a search direction in the space of search distributions. A straightforward gradient ascent scheme can thus iteratively update the search distribution

$$\theta \leftarrow \theta + \eta \nabla_\theta J(\theta),$$

where η is a learning rate parameter. Algorithm 1 provides the pseudocode for this very general approach to black-box optimization by using a search gradient on search distributions.

Algorithm 1: Canonical Search Gradient algorithm

```
  input: $f$, $\theta_{init}$
1 repeat
2     for $k = 1 \ldots \lambda$ do
3         draw sample $\mathbf{z}_k \sim \pi(\cdot|\theta)$
4         evaluate the fitness $f(\mathbf{z}_k)$
5         calculate log-derivatives $\nabla_\theta \log \pi(\mathbf{z}_k|\theta)$
6     end
7     $\nabla_\theta J \leftarrow \frac{1}{\lambda} \sum_{k=1}^{\lambda} \nabla_\theta \log \pi(\mathbf{z}_k|\theta) \cdot f(\mathbf{z}_k)$
8     $\theta \leftarrow \theta + \eta \cdot \nabla_\theta J$
9 until stopping criterion is met
```

Utilizing the search gradient in this framework is similar to evolution strategies in that it iteratively generates the fitnesses of batches of vector-valued samples – the ES's so-called candidate solutions. It is different however, in that it represents this 'population' as a parameterized distribution, and in the fact that it uses a search gradient to update the parameters of this distribution, which is computed using the fitnesses.

2.2.1 Search Gradients for Gaussian Distributions

In the case of the 'default' d-dimensional multi-variate normal distribution, we collect the parameters of the Gaussian, the mean $\boldsymbol{\mu} \in \mathbb{R}^d$ (candidate solution center) and the covariance matrix $\mathbf{\Sigma} \in \mathbb{R}^{d \times d}$ (mutation matrix), in a single concatenated vector $\theta = \langle \boldsymbol{\mu}, \mathbf{\Sigma} \rangle$. However, to sample efficiently from this distribution we need a square root of the covariance matrix (a matrix $\mathbf{A} \in \mathbb{R}^{d \times d}$ fulfilling $\mathbf{A}^\top \mathbf{A} = \mathbf{\Sigma}$). Then $\mathbf{z} = \boldsymbol{\mu} + \mathbf{A}^\top \mathbf{s}$ transforms a standard normal vector $\mathbf{s} \sim \mathcal{N}(0, \mathbb{I})$ into a sample $\mathbf{z} \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{\Sigma})$. Here, $\mathbb{I} = \mathrm{diag}(1, \ldots, 1) \in \mathbb{R}^{d \times d}$ denotes the identity matrix. Let

$$\begin{aligned} \pi(\mathbf{z} \,|\, \theta) &= \frac{1}{(\sqrt{2\pi})^d \det(\mathbf{A})} \cdot \exp\left(-\frac{1}{2} \left\| \mathbf{A}^{-1} \cdot (\mathbf{z} - \boldsymbol{\mu}) \right\|^2 \right) \\ &= \frac{1}{\sqrt{(2\pi)^d \det(\mathbf{\Sigma})}} \cdot \exp\left(-\frac{1}{2} (\mathbf{z} - \boldsymbol{\mu})^\top \mathbf{\Sigma}^{-1} (\mathbf{z} - \boldsymbol{\mu}) \right) \end{aligned}$$

denote the density of the multinormal search distribution $\mathcal{N}(\boldsymbol{\mu}, \mathbf{\Sigma})$.

In order to calculate the derivatives of the log-likelihood with respect to individual elements of θ for this multinormal distribution, first note that

$$\log \pi(\mathbf{z}|\theta) = -\frac{d}{2}\log(2\pi) - \frac{1}{2}\log\det\mathbf{\Sigma} - \frac{1}{2}(\mathbf{z}-\boldsymbol{\mu})^\top \mathbf{\Sigma}^{-1}(\mathbf{z}-\boldsymbol{\mu}).$$

We will need its derivatives, that is, $\nabla_{\boldsymbol{\mu}} \log \pi(\mathbf{z}|\theta)$ and $\nabla_{\mathbf{\Sigma}} \log \pi(\mathbf{z}|\theta)$. The first is trivially

$$\nabla_{\boldsymbol{\mu}} \log \pi(\mathbf{z}|\theta) = \mathbf{\Sigma}^{-1}(\mathbf{z}-\boldsymbol{\mu}), \tag{2.3}$$

while the latter is

$$\nabla_{\mathbf{\Sigma}} \log \pi(\mathbf{z}|\theta) = \frac{1}{2}\mathbf{\Sigma}^{-1}(\mathbf{z}-\boldsymbol{\mu})(\mathbf{z}-\boldsymbol{\mu})^\top \mathbf{\Sigma}^{-1} - \frac{1}{2}\mathbf{\Sigma}^{-1}. \tag{2.4}$$

In order to preserve symmetry, to ensure non-negative variances and to keep the mutation matrix $\mathbf{\Sigma}$ positive definite, $\mathbf{\Sigma}$ needs to be constrained. One way to accomplish that is by representing $\mathbf{\Sigma}$ as a product $\mathbf{\Sigma} = \mathbf{A}^\top\mathbf{A}$ (for a more sophisticated solution to this issue, see section 2.5.1). Instead of using the log-derivatives on $\nabla_{\mathbf{\Sigma}} \log \pi(\mathbf{z})$ directly, we then compute the derivatives with respect to $\mathbf{A}$ as

$$\nabla_{\mathbf{A}} \log \pi(\mathbf{z}) = \mathbf{A}\left[\nabla_{\mathbf{\Sigma}} \log \pi(\mathbf{z}) + \nabla_{\mathbf{\Sigma}} \log \pi(\mathbf{z})^\top\right].$$

Using these derivatives to calculate $\nabla_\theta J$, we can then update parameters $\theta = \langle\boldsymbol{\mu}, \mathbf{\Sigma} = \mathbf{A}^\top\mathbf{A}\rangle$ as $\theta \leftarrow \theta + \eta\nabla_\theta J$ using learning rate η. This produces a new center $\boldsymbol{\mu}$ for the search distribution, and simultaneously self-adapts its associated covariance matrix $\mathbf{\Sigma}$. To summarize, we provide the pseudocode for following the search gradient in the case of a multinormal search distribution in Algorithm 2.

Algorithm 2: Search Gradient algorithm: Multinormal distribution

```
   input: $f$, $\boldsymbol{\mu}_{init}$,$\mathbf{\Sigma}_{init}$
1  repeat
2      for $k = 1 \ldots \lambda$ do
3          draw sample $\mathbf{z}_k \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{\Sigma})$
4          evaluate the fitness $f(\mathbf{z}_k)$
           calculate log-derivatives:
5            $\nabla_{\boldsymbol{\mu}} \log \pi(\mathbf{z}_k|\theta) = \mathbf{\Sigma}^{-1}(\mathbf{z}_k - \boldsymbol{\mu})$
             $\nabla_{\mathbf{\Sigma}} \log \pi(\mathbf{z}_k|\theta) = -\frac{1}{2}\mathbf{\Sigma}^{-1} + \frac{1}{2}\mathbf{\Sigma}^{-1}(\mathbf{z}_k - \boldsymbol{\mu})(\mathbf{z}_k - \boldsymbol{\mu})^\top \mathbf{\Sigma}^{-1}$
6      end
7      $\nabla_{\boldsymbol{\mu}} J \leftarrow \frac{1}{\lambda}\sum_{k=1}^{\lambda} \nabla_{\boldsymbol{\mu}} \log \pi(\mathbf{z}_k|\theta) \cdot f(\mathbf{z}_k)$
8      $\nabla_{\mathbf{\Sigma}} J \leftarrow \frac{1}{\lambda}\sum_{k=1}^{\lambda} \nabla_{\mathbf{\Sigma}} \log \pi(\mathbf{z}_k|\theta) \cdot f(\mathbf{z}_k)$
9      $\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} + \eta \cdot \nabla_{\boldsymbol{\mu}} J$
10     $\mathbf{\Sigma} \leftarrow \mathbf{\Sigma} + \eta \cdot \nabla_{\mathbf{\Sigma}} J$
11 until stopping criterion is met
```

2.2.2 Limitations of Plain Search Gradients

As the attentive reader will have realized, there exists at least one major issue with applying the search gradient as-is in practice: It is impossible to *precisely locate* a (quadratic) optimum, even in the one-dimensional case. Let $d = 1$, $\theta = \langle \mu, \sigma \rangle$, and samples $z \sim \mathcal{N}(\mu, \sigma^2)$. Equations (2.3) and (2.4), the gradients on μ and σ, become

$$\begin{aligned} \nabla_\mu J &= \frac{z-\mu}{\sigma^2} \\ \nabla_\sigma J &= \frac{(z-\mu)^2 - \sigma^2}{\sigma^3} \end{aligned}$$

and the updates, assuming simple hill-climbing (i.e. a population size $\lambda = 1$) read:

$$\begin{aligned} \mu &\leftarrow \mu + \eta \frac{z-\mu}{\sigma^2} \\ \sigma &\leftarrow \sigma + \eta \frac{(z-\mu)^2 - \sigma^2}{\sigma^3}. \end{aligned}$$

For any objective function f that requires locating an (approximately) quadratic optimum with some degree of precision (e.g. $f(\mathbf{z}) = \mathbf{z}^2$), σ must decrease, which in turn increases the variance of the updates, as $\Delta\mu \propto \frac{1}{\sigma}$ and $\Delta\sigma \propto \frac{1}{\sigma}$ for a typical sample z. In fact, the updates become increasingly unstable, the smaller σ becomes, an effect which a reduced learning rate or an increased population size can only delay but not avoid. Figure 2.1 illustrates this effect. Conversely, whenever $\sigma \gg 1$ is large, the magnitude of a typical update is severely reduced.

Clearly, this update is not at all *scale-invariant*: Starting with $\sigma \gg 1$ makes all updates minuscule, whereas starting with $\sigma \ll 1$ makes the first update huge and therefore unstable.

We conjecture that this limitation constitutes one of the main reasons why search gradients have not been developed before: in isolation, the plain search gradient's performance is both unstable and unsatisfying, and it is only the application of natural gradients (introduced in section 2.3) which tackles these issues and renders search gradients into a viable optimization method.

2.3 Using the Natural Gradient

Instead of using the plain stochastic gradient for updates, NES follows the *natural gradient*. The natural gradient was first introduced by Amari in 1998, and has been shown to possess numerous advantages over the plain gradient (Amari, 1998; Amari and Douglas, 1998). In terms of mitigating the slow convergence of plain gradient ascent in optimization landscapes with ridges and plateaus, natural gradients are a more principled (and hyper-parameter-free) approach than, for example, the commonly used momentum heuristic.

The plain gradient ∇J simply follows the steepest ascent in the space of the actual parameters θ of the distribution. This means that for a given small step-

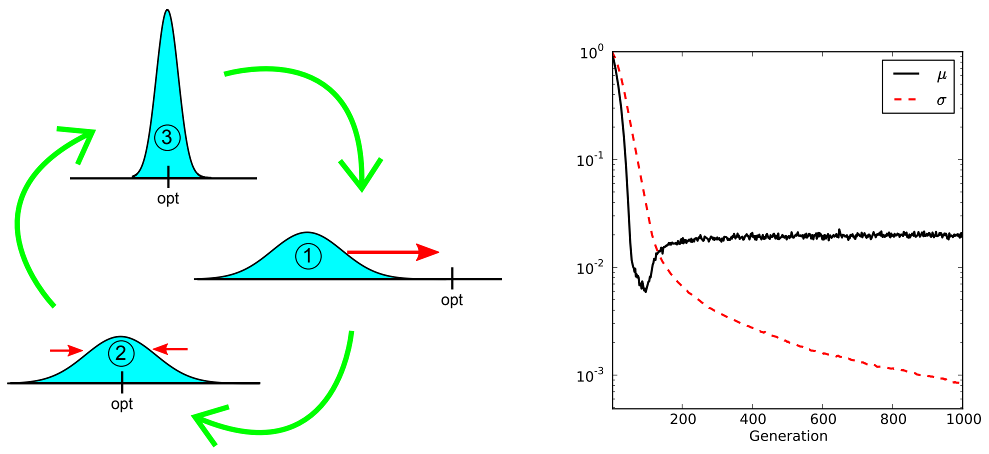


Figure 2.1: **Left:** Schematic illustration of how the search distribution adapts in the one-dimensional case: from (1) to (2), μ is adjusted to make the distribution cover the optimum. From (2) to (3), σ is reduced to allow for a precise localization of the optimum. The step from (3) to (1) then is the problematic case, where a small σ induces a largely overshooting update, making the search start over again. **Right:** Progression of μ (black) and σ (red, dashed) when following the search gradient towards minimizing $f(\mathbf{z}) = \mathbf{z}^2$, executing Algorithm 2. Plotted are median values over 1000 runs, with a small learning rate $\eta = 0.01$ and $\lambda = 10$, both of which mitigate the instability somewhat, but still show the failure to precisely locate the optimum (for which both μ and σ need to approach 0).

size ε, following it will yield a new distribution with parameters chosen from the hypersphere of radius ϵ and center θ that maximizes J. In other words, the Euclidean distance in parameter space between subsequent distributions is fixed. Clearly, this makes the update dependent on the particular parameterization of the distribution, therefore a change of parameterization leads to different gradients and different updates. See also Figure 2.2 for an illustration of how this effectively renormalizes updates w.r.t. uncertainty.

The key idea of the natural gradient is to remove this dependence on the parameterization by relying on a more 'natural' measure of distance $\mathbb{D}(\theta' \| \theta)$ between probability distributions $\pi(\mathbf{z}|\theta)$ and $\pi(\mathbf{z}|\theta')$. One such natural distance measure between two probability distributions is the Kullback-Leibler divergence (Kullback and Leibler, 1951). The natural gradient can then be formalized as the solution to the constrained optimization problem

$$\begin{aligned} &\max_{\delta\theta} J(\theta + \delta\theta) \approx J(\theta) + \delta\theta^\top \nabla_\theta J, \\ &s.t.\ \mathbb{D}(\theta + \delta\theta \| \theta) = \varepsilon, \end{aligned}$$

where $J(\theta)$ is the expected fitness of equation (2.1), and ε is a small increment size. Now, we have for $\delta\theta \to 0$,

$$\mathbb{D}(\theta + \delta\theta \| \theta) = \frac{1}{2}\delta\theta^\top \mathbf{F}(\theta)\,\delta\theta,$$

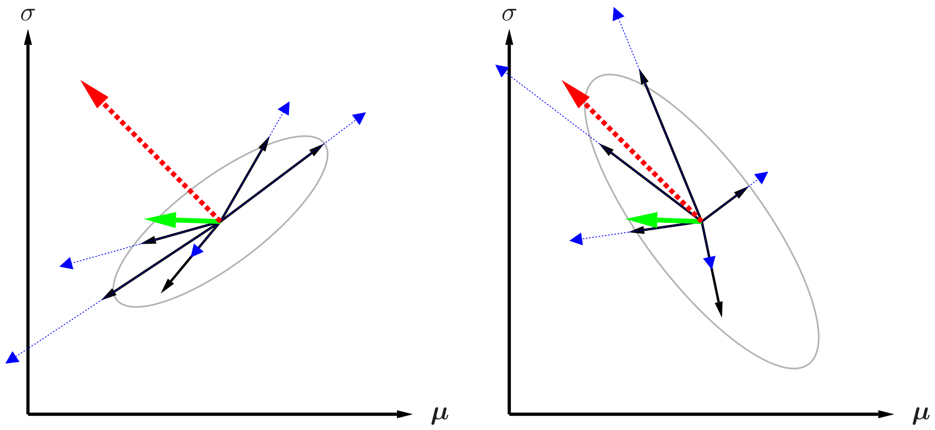


Figure 2.2: Illustration of plain versus natural gradient in parameter space. Consider two parameters, e.g. $\theta = (\mu, \sigma)$, of the search distribution. In the plot on the left, the solid (black) arrows indicate the gradient samples $\nabla_\theta \log \pi(\mathbf{z} \mid \theta)$, while the dotted (blue) arrows correspond to $f(\mathbf{z}) \cdot \nabla_\theta \log \pi(\mathbf{z} \mid \theta)$, that is, the same gradient estimates, but scaled with fitness. Combining these, the bold (green) arrow indicates the (sampled) fitness gradient $\nabla_\theta J$, while the bold dashed (red) arrow indicates the corresponding natural gradient $\tilde{\nabla}_\theta J$.
Being random variables with expectation zero, the distribution of the black arrows is governed by their covariance, indicated by the gray ellipse. Notice that this covariance is a quantity in *parameter space* (where the θ reside), which is not to be confused with the covariance of the distribution in the *search space* (where the samples $\mathbf{z}$ reside). In contrast, solid (black) arrows on the right represent $\tilde{\nabla}_\theta \log \pi(\mathbf{z} \mid \theta)$, and dotted (blue) arrows indicate the *natural* gradient samples $f(\mathbf{z}) \cdot \tilde{\nabla}_\theta \log \pi(\mathbf{z} \mid \theta)$, resulting in the natural gradient (dashed red).
The covariance of the solid arrows on the right hand side turns out to be the inverse of the covariance of the solid arrows on the left. This has the effect that when computing the natural gradient, directions with high variance (uncertainty) are penalized and thus shrunken, while components with low variance (high certainty) are boosted, since these components of the gradient samples deserve more trust. This makes the (dashed red) natural gradient a much more trustworthy update direction than the (green) plain gradient.

where

$$\begin{aligned}\mathbf{F} &= \int \pi\left(\mathbf{z}|\theta\right) \nabla_\theta \log \pi\left(\mathbf{z}|\theta\right) \nabla_\theta \log \pi\left(\mathbf{z}|\theta\right)^\top d\mathbf{z}, \\ &= \mathbb{E}\left[\nabla_\theta \log \pi\left(\mathbf{z}|\theta\right) \nabla_\theta \log \pi\left(\mathbf{z}|\theta\right)^\top\right]\end{aligned}$$

is the *Fisher information matrix* of the given parametric family of search distributions. The solution to this can be found using a Lagrangian multiplier (Peters, 2007), yielding the necessary condition

$$\mathbf{F}\delta\theta = \beta\nabla_\theta J, \tag{2.5}$$

for some constant $\beta > 0$. The direction of the natural gradient $\widetilde{\nabla}_\theta J$ is given by $\delta\theta$ thus defined. If $\mathbf{F}$ is invertible[1], the natural gradient amounts to

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

The Fisher matrix can be estimated from samples, reusing the log-derivatives $\nabla_\theta \log \pi(\mathbf{z}|\theta)$ that we already computed for the gradient $\nabla_\theta J$. Then, updating the parameters following the natural gradient instead of the steepest gradient leads us to the general formulation of NES, as shown in Algorithm 3.

Algorithm 3: Canonical Natural Evolution Strategies (NES)

```
   input: $f$, $\theta_{init}$
1  repeat
2      for $k = 1 \ldots \lambda$ do
3          draw sample $\mathbf{z}_k \sim \pi(\cdot|\theta)$
4          evaluate the fitness $f(\mathbf{z}_k)$
5          calculate log-derivatives $\nabla_\theta \log \pi(\mathbf{z}_k|\theta)$
6      end
7      $\nabla_\theta J \leftarrow \frac{1}{\lambda}\sum_{k=1}^{\lambda} \nabla_\theta \log \pi(\mathbf{z}_k|\theta) \cdot f(\mathbf{z}_k)$
8      $\mathbf{F} \leftarrow \frac{1}{\lambda}\sum_{k=1}^{\lambda} \nabla_\theta \log \pi\left(\mathbf{z}_k|\theta\right) \nabla_\theta \log \pi\left(\mathbf{z}_k|\theta\right)^\top$
9      $\theta \leftarrow \theta + \eta \cdot \mathbf{F}^{-1}\nabla_\theta J$
10 until stopping criterion is met
```

2.4 Performance and Robustness Techniques

In the following we will present and introduce crucial techniques to improves NES's performance and robustness. Fitness shaping is designed to make the algorithm invariant w.r.t. arbitrary yet order-preserving fitness transformations

[1]Care has to be taken because the Fisher matrix estimate may not be (numerically) invertible even if the exact Fisher matrix is.

(section 2.4.1). Importance mixing is designed to recycle samples so as to reduce the number of required fitness evaluations, and is subsequently presented in section 2.4.3. Adaptation sampling, a novel technique for adjusting learning rates online, is introduced in section 2.4.4, and finally restart strategies, designed to improve success rates on multi-modal problems, is presented in section 2.4.5.

2.4.1 Fitness Shaping

NES utilizes rank-based fitness shaping in order to render the algorithm *invariant* under monotonically increasing (i.e., rank preserving) transformations of the fitness function. For this purpose, the fitness of the population is transformed into a set of utility values $u_1 \geq \cdots \geq u_\lambda$. Let $\mathbf{z}_i$ denote the i^{th} best individual (the i^{th} individual in the population, sorted by fitness, such that $\mathbf{z}_1$ is the best and $\mathbf{z}_\lambda$ the worst individual). Replacing fitness with utility, the gradient estimate of equation (2.2) becomes, with slight abuse of notation,

$$\nabla_\theta J(\theta) = \sum_{k=1}^{\lambda} u_k \, \nabla_\theta \log \pi(\mathbf{z}_k \,|\, \theta). \tag{2.6}$$

To avoid entangling the utility-weightings with the learning rate, we require that $\sum_{k=1}^{\lambda} |u_k| = 1$.

The choice of utility function can in fact be seen as a free parameter of the algorithm. Throughout this chapter we will use the following

$$u_k = \frac{\max\left(0, \log(\frac{\lambda}{2}+1) - \log(i)\right)}{\sum_{j=1}^{\lambda} \max\left(0, \log(\frac{\lambda}{2}+1) - \log(j)\right)} - \frac{1}{\lambda},$$

which is directly related to the one employed by CMA-ES (Hansen and Ostermeier, 2001), for ease of comparison. In our experience, however, this choice has not been crucial to performance, as long as it is monotonous and based on ranks instead of raw fitness (e.g., a function which simply increases linearly with rank).

In addition to robustness, these utility values provide us with an elegant formalism to describe the (1+1) hill-climber version of the algorithm within the same framework, by using different utility values, depending on success (see section 2.5.5 later in this chapter).

2.4.2 Fitness Baselines

Estimating the gradient ∇J (and the Fisher matrix) from samples, we are concerned with keeping the variance of the estimate $\mathrm{Var}\left[\nabla J\right]$ low, so as to reduce the number of samples required to correctly estimate it. As variance grows quadratically with the average magnitude of the fitnesses, it can be significantly reduced if we introduce a *fitness baseline* b, which does not bias the estimator, as:

$$\begin{aligned}\nabla J &= \mathbb{E}\left[\nabla \log \pi\left(\mathbf{z}\right) f\left(\mathbf{z}\right)\right] \\ &= \mathbb{E}\left[\nabla \log \pi\left(\mathbf{z}\right)\left[f\left(\mathbf{z}\right) - b\right]\right].\end{aligned}$$

We now have have a free fitness baseline parameter b which can be used to reduce the estimation variance, and equation (2.5) can be rewritten as:

$$\mathbb{E}\left[\nabla \log \pi\left(\mathbf{z}\right) \nabla \log \pi\left(\mathbf{z}\right)^\top\right] \delta\theta = \mathbb{E}\left[\nabla \log \pi\left(\mathbf{z}\right) f\left(\mathbf{z}\right)\right] - \mathbb{E}\left[\nabla \log \pi\left(\mathbf{z}\right) b\right]. \quad (2.7)$$

We obtain b by realizing the lower bound

$$\begin{aligned}\mathrm{Var}\left[\nabla \log \pi\left(\mathbf{z}\right)\left[f\left(\mathbf{z}\right) - b\right]\right] &= \bar{f}^2\, \mathrm{Var}\left[\nabla \log \pi\left(\mathbf{z}\right) \frac{\left(f\left(\mathbf{z}\right) - \bar{f}\right)}{\bar{f}}\right] \\ &\quad + \mathrm{Var}\left[\nabla \log \pi\left(\mathbf{z}\right)\left[\bar{f} - b\right]\right] \\ &\geq \bar{f}^2 \mathbb{E}\left[\nabla \log \pi\left(\mathbf{z}\right) \nabla \log \pi\left(\mathbf{z}\right)^\top\right] \\ &\quad + \mathrm{Var}\left[\nabla \log \pi\left(\mathbf{z}\right)\left[\bar{f} - b\right]\right],\end{aligned}$$

where $\bar{f} = \mathbb{E}\left[f\left(\mathbf{z}\right)\right]$. Thus, we have a minimum at $b = \bar{f}$, which does not yet take into account any interplay between $\delta\theta$ and b, however. The next step, then, is to use $\mathbb{E}\left[\nabla \log \pi\left(\mathbf{z}\right)\right] = 0$ to obtain the equation

$$\begin{aligned}0 + b &= \mathbb{E}\left[f\left(\mathbf{z}\right)\right], \\ \mathbb{E}\left[\nabla \log \pi\left(\mathbf{z}\right)\right]^\top \delta\theta + b &= \mathbb{E}\left[f\left(\mathbf{z}\right)\right],\end{aligned}$$

which, together with equation (2.7), gives the system of equations

$$\begin{array}{rlrl}\mathbb{E}\left[\nabla \log \pi\left(\mathbf{z}\right) \nabla \log \pi\left(\mathbf{z}\right)^\top\right]\delta\theta & +\mathbb{E}\left[\nabla \log \pi\left(\mathbf{z}\right)\right] & b = & \mathbb{E}\left[\nabla \log \pi\left(\mathbf{z}\right) f\left(\mathbf{z}\right)\right] \\ \mathbb{E}\left[\nabla \log \pi\left(\mathbf{z}\right)\right]^\top\delta\theta & + & b = & \mathbb{E}\left[f\left(\mathbf{z}\right)\right],\end{array}$$

which can be solved straightforwardly as a linear regression problem using the pseudoinverse. When replacing the $\mathbb{E}\left[\cdot\right]$ by sample averages, we then obtain the general natural gradient estimator

$$\delta\theta = (\mathbf{\Phi}^\top\mathbf{\Phi})^{-\mathbf{1}}\mathbf{\Phi}^\top\mathbf{r}$$

where

$$\begin{aligned}\mathbf{\Phi} &= \begin{bmatrix} \nabla_\theta \log \pi(\mathbf{z}_1) & 1 \\ & \vdots \\ \nabla_\theta \log \pi(\mathbf{z}_\lambda) & 1 \end{bmatrix} \\ \mathbf{r} &= \left[f(\mathbf{z}_1), \ldots, f(\mathbf{z}_\lambda)\right]^\top.\end{aligned}$$

This solution forms the basis of the distribution-agnostic pseudocode of Algorithm 4.

```
Algorithm 4: Natural Evolution Strategies with baselines
    input: $f$, $\theta_{init}$
1   repeat
2       for $k = 1 \ldots \lambda$ do
3           draw sample $\mathbf{z}_k \sim \pi(\cdot|\theta)$
4           evaluate the fitness $f(\mathbf{z}_k)$
5           calculate log-derivatives $\nabla_\theta \log \pi(\mathbf{z}_k|\theta)$
6       end
7       $\mathbf{\Phi} = \begin{bmatrix} \nabla_\theta \log \pi(\mathbf{z}_1) & 1 \\ \vdots & \vdots \\ \nabla_\theta \log \pi(\mathbf{z}_\lambda) & 1 \end{bmatrix}$
8       $\mathbf{r} = [f(\mathbf{z}_1), \ldots, f(\mathbf{z}_\lambda)]^\top$
9       $\delta\theta = (\mathbf{\Phi}^\top \mathbf{\Phi})^{-1} \mathbf{\Phi}^\top \mathbf{r}$
10      $\theta \leftarrow \theta + \eta \cdot \delta\theta$
11  until stopping criterion is met
```

2.4.3 Importance Mixing

In each batch, we evaluate λ new samples generated from search distribution $\pi(\mathbf{z}|\theta)$. However, since small updates ensure that the KL divergence between consecutive search distributions is generally small, most new samples will fall in the high density area of the previous search distribution $\pi(\mathbf{z}|\theta')$. This leads to redundant fitness evaluations in that same area. We improve the efficiency with a new procedure called *importance mixing*, which aims at *reusing* fitness evaluations from the previous batch, while ensuring the updated batch conforms to the new search distribution.

Importance mixing works in two steps: In the first step, rejection sampling is performed on the previous batch, such that sample $\mathbf{z}$ is accepted with probability

$$\min\left\{1, (1-\alpha)\frac{\pi(\mathbf{z}|\theta)}{\pi(\mathbf{z}|\theta')}\right\}. \tag{2.8}$$

Here $\alpha \in [0,1]$ is an algorithm hyperparameter called the *minimal refresh rate*. Let λ_a be the number of samples accepted in the first step. In the second step, reverse rejection sampling is performed as follows: Generate samples from $\pi(\mathbf{z}|\theta)$ and accept $\mathbf{z}$ with probability

$$\max\left\{\alpha, 1 - \frac{\pi(\mathbf{z}|\theta')}{\pi(\mathbf{z}|\theta)}\right\} \tag{2.9}$$

until $\lambda - \lambda_a$ new samples are accepted. The λ_a samples from the old batch and $\lambda - \lambda_a$ newly accepted samples together constitute the new batch. Figure 2.3 illustrates the procedure. Note that only the fitnesses of the newly accepted

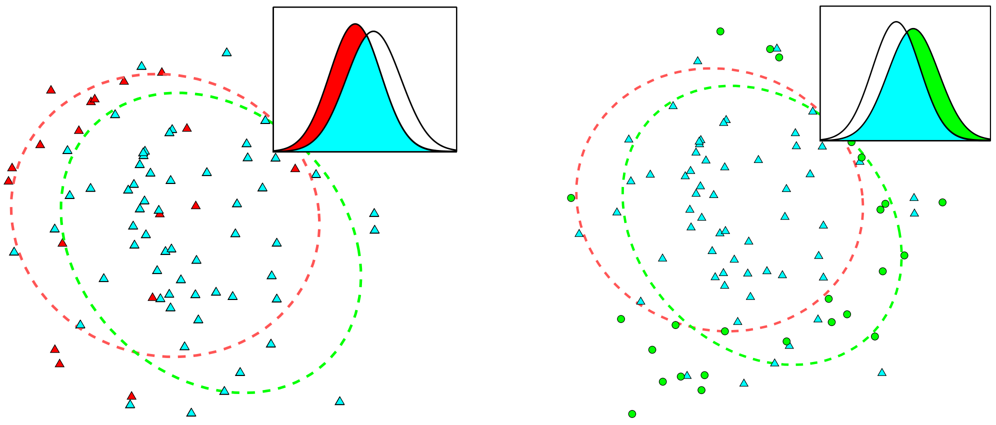


Figure 2.3: Illustration of importance mixing. **Left:** In the first step, old samples are eliminated (red triangles) according to (2.8), and the remaining samples (blue triangles) are reused. **Right:** In the second step, new samples (green circles) are generated according to (2.9).

samples need to be evaluated. The advantage of using importance mixing is twofold: On the one hand, we reduce the number of fitness evaluations required in each batch, on the other hand, if we fix the number of newly evaluated fitnesses, then many more fitness evaluations can potentially be used to yield more reliable and accurate gradients.

The minimal refresh rate parameter α has two uses. First, it avoids too low an acceptance probability at the second step when $\pi(\mathbf{z}|\theta')/\pi(\mathbf{z}|\theta) \simeq 1$. And second, it permits specifying a lower bound on the expected proportion of newly evaluated samples $\rho = \mathbb{E}\left[\frac{\lambda-\lambda_a}{\lambda}\right]$, namely $\rho \geq \alpha$, with the equality holding if and only if $\theta = \theta'$. In particular, if $\alpha = 1$, all samples from the previous batch will be discarded, and if $\alpha = 0$, ρ depends only on the distance between $\pi(\mathbf{z}|\theta)$ and $\pi(\mathbf{z}|\theta')$. Normally we set α to be a small positive number, e.g., in this chapter we use $\alpha = 0.1$ throughout.

It can be proven that the updated batch conforms to the search distribution $\pi(\mathbf{z}|\theta)$. In the region where

$$(1-\alpha)\frac{\pi(\mathbf{z}|\theta)}{\pi(\mathbf{z}|\theta')} \leq 1,$$

the probability that a sample from previous batches appears in the new batch is

$$\pi(\mathbf{z}|\theta') \cdot (1-\alpha)\frac{\pi(\mathbf{z}|\theta)}{\pi(\mathbf{z}|\theta')} = (1-\alpha)\,\pi(\mathbf{z}|\theta).$$

The probability that a sample generated from the second step entering the batch is $\alpha\pi(\mathbf{z}|\theta)$, since

$$\max\left\{\alpha, 1-\frac{\pi(\mathbf{z}|\theta')}{\pi(\mathbf{z}|\theta)}\right\} = \alpha.$$

So the probability of a sample entering the batch is just $p(\mathbf{z}|\theta)$ in that region.

The same result holds also for the region where

$$(1-\alpha)\frac{\pi(\mathbf{z}|\theta)}{\pi(\mathbf{z}|\theta')} > 1.$$

When using importance mixing in the context of NES, this reduces the sensitivity to the hyperparameters, particularly the population size λ, as importance mixing implicitly adapts it to the situation by reusing some or many previously evaluated sample points.

2.4.4 Adaptation Sampling

To reduce the burden on determining appropriate hyper-parameters such as the learning rate, we develop a new self-adaptation or meta-learning technique (Schaul and Schmidhuber, 2010a), called *adaptation sampling* that can automatically adapt the settings in a principled and economical way.

We model this situation as follows: Let π_θ be a distribution with hyper-parameter θ and $\psi(\mathbf{z})$ a quality measure for each sample $\mathbf{z} \sim \pi_\theta$. Our goal is to adapt θ such as to maximize the quality ψ. A straightforward method to achieve this, henceforth dubbed *adaptation sampling*, is to evaluate the quality of the samples $\mathbf{z}'$ drawn from $\pi_{\theta'}$, where $\theta' \neq \theta$ is a slight variation of θ, and then perform hill-climbing: Continue with the new θ' if the quality of its samples is significantly better (according, e.g., to a Mann-Whitney U-test), and revert to θ otherwise. Note that this proceeding is similar to the NES algorithm itself, but applied at a meta-level to algorithm parameters instead of the search distribution. The goal of this adaptation is to maximize the *pace* of progress over time, which is slightly different from maximizing the fitness function itself.

Virtual adaptation sampling is a lightweight alternative to adaptation sampling that is particularly useful whenever evaluating ψ is expensive :

- do importance sampling on the existing samples $\mathbf{z}_i$, according to $\pi_{\theta'}$:

  $$w'_i = \frac{\pi(\mathbf{z}|\theta')}{\pi(\mathbf{z}|\theta)}$$

  (this is always well-defined, because $\mathbf{z} \sim \pi_\theta \Rightarrow \pi(\mathbf{z}|\theta) > 0$).

- compare $\{\psi(\mathbf{z}_i)\}$ with weights $\{w_i = 1, \forall i\}$ and $\{\psi' = \psi(\mathbf{z}_i), \forall i\}$ with weights $\{w'_i\}$, using a weighted version of the Mann-Whitney test, as introduced in Appendix A.

Beyond determining whether θ or θ' is better, choosing a non-trivial confidence level ρ allows us to avoid parameter drift, as θ is only updated if the improvement is significant enough.

There is one caveat, however: the rate of parameter change needs to be adjusted such that the two resulting distributions are not too similar (otherwise the difference won't be statistically significant), but also not too different, (otherwise the weights w' will be too small and again the test will be inconclusive).

If, however, we explicitly desire large adaptation steps on θ, we have the possibility of interpolating between adaptation sampling and virtual adaptation sampling by drawing a few new samples from the distribution $\pi_{\theta'}$ (each assigned weight 1), where it is overlapping least with π_θ. The best way of achieving this is importance mixing, as introduced in Section 2.4.3, uses jointly with the reweighted existing samples.

For NES algorithms, the most important parameter to be adapted by adaptation sampling is the learning rate η, starting with a conservative guess. This is because half-way into the search, after a local attractor has been singled out, it may well pay off to increase the learning rate in order to more quickly converge to it.

In order to produce variations η' which can be judged using the above-mentioned U-test, we propose a procedure similar in spirit to Rprop-updates (Igel and Husken, 2003; Riedmiller and Braun, 1993), where the learning rates are either increased or decreased by a multiplicative constant whenever there is evidence that such a change will lead to better samples.

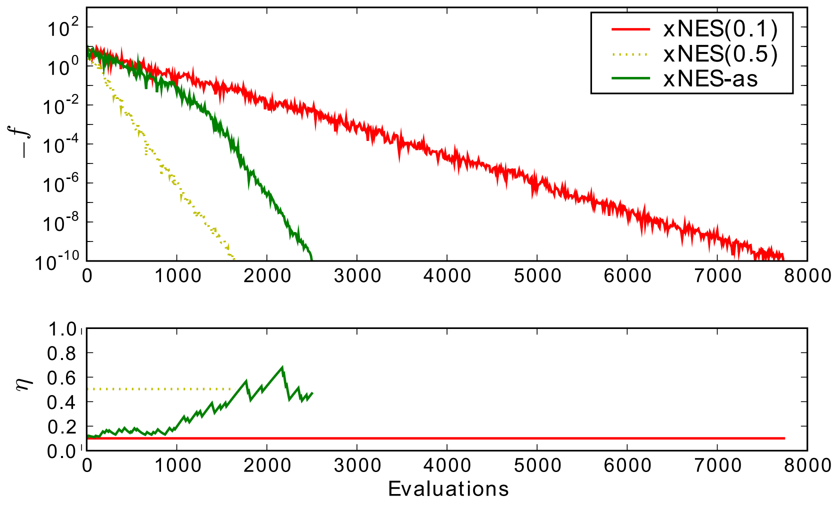
More concretely, when using adaptation sampling for NES we test for an improvement with the hypothetical distribution θ' generated with $\eta' = 1.5\eta$. Each time the statistical test is successful with a confidence of at least $\rho = \frac{1}{2} - \frac{1}{3(d+1)}$ (this value was determined empirically) we increase the learning rate by a factor of $c^+ = 1.1$, up to at most $\eta = 1$. Otherwise we bring it closer to its initial value: $\eta \leftarrow 0.9\eta + 0.1\eta_{init}$.

Figure 2.4 illustrates the effect of the virtual adaptation sampling strategy on two different 10-dimensional unimodal benchmark functions, the Sphere function f_1 and the Rosenbrock function f_8 (see section 2.7.2 for details). We find that, indeed, adaptation sampling boosts the learning rates to the appropriate high values when quick progress can be made (in the presence of an approximately quadratic optimum), but keeps them at carefully low values otherwise.

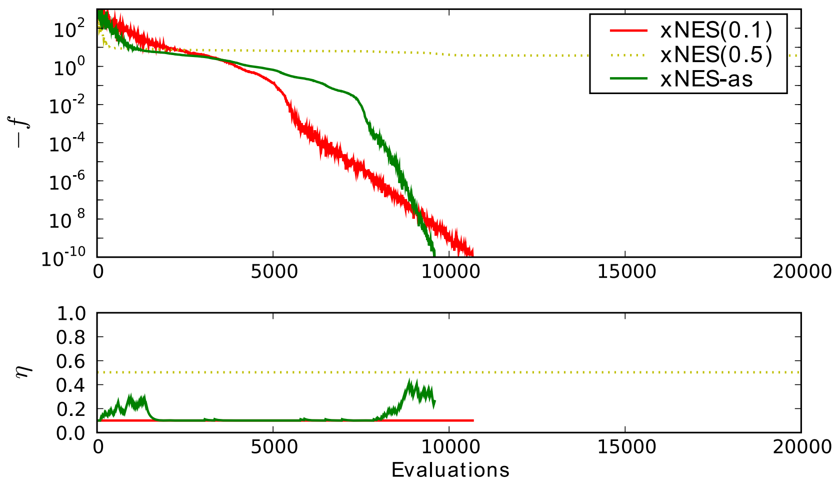
2.4.5 Restart Strategies

A simple but widespread method for mitigating the risk of finding only local optima in a strongly multi-modal scenario is to *restart* the optimization algorithm a number of times with different initializations, or just with a different random seed. This is even more useful in the presence of parallel processing resources, in which case multiple runs are executed simultaneously.

In practice, where the parallel capacity is limited, we need to decide when to stop or suspend an ongoing optimization run and start or resume another one. In this section we provide one such restart strategy that takes those decisions. Inspired by recent work on practical universal search (Schaul and Schmidhuber, 2010b), this results in an effective use of resources independently of the problem.



(a) 10-dimensional sphere (easy)



(b) 10-dimensional Rosenbrock (hard)

Figure 2.4: **Illustration of the effect of adaptation sampling.** We show the increase in fitness during a NES run (above) and the corresponding learning rates (below) on two setups: 10-dimensional sphere function (a), and 10-dimensional Rosenbrock function (b). Plotted are three variants of xNES (algorithm 5): fixed default learning rate of $\eta = 0.1$ (dashed, red) fixed large learning rate of $\eta = 0.5$ (dotted, yellow), and an adaptive learning rate starting at $\eta = 0.1$ (green). (a) We see that for the (simple) Sphere function, it is advantageous to use a large learning rate, and adaptation sampling automatically finds that one. However, using the overly greedy updates of a large learning rate fails on harder problems (b). Here adaptation sampling really shines: it boosts the learning rate in the initial phase (entering the Rosenbrock valley), then quickly reduces it while the search needs to carefully navigate the bottom of the valley, and boosts it again at the end when it has located the optimum and merely needs to zoom in precisely.

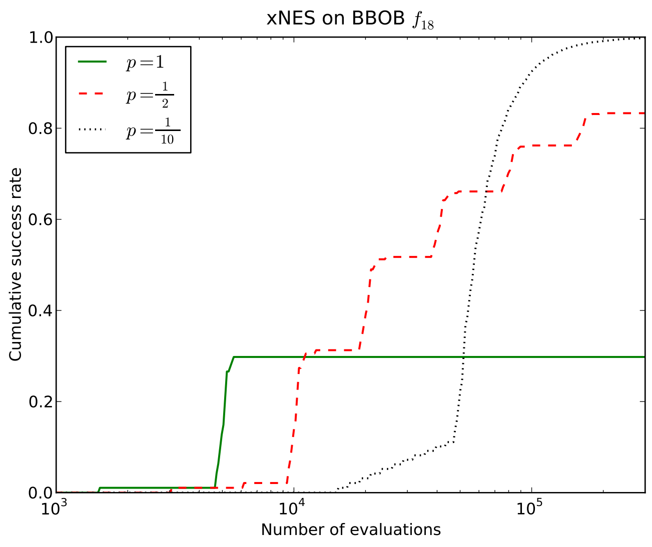


Figure 2.5: Illustrating the effect of different restart strategies. Plotted is the cumulative empirical success probability, as a function of the total number of evaluations used. Using no restarts, corresponding to $p = 1$ (green) is initially faster but unreliable, whereas $p = \frac{1}{10}$ (dotted black) reliably finds the optimum within 300000 evaluations, but at the cost of slowing down success for all runs by a factor 10. In-between these extremes, $p = \frac{1}{2}$ (broken line, red) trades off slowdown and reliability.

The strategy consists in reserving a fixed fraction p of the total time for the first run, and then subdividing the remaining time $1 - p$ in the same way, recursively (i.e. $p(1-p)^{i-1}$ for the i^{th} run). The recursive decomposition of the time budget stops when the assigned time-slice becomes smaller than the overhead of swapping out different runs. In this way, the number of runs with different seeds remains finite, but increases over time, as needed. Figure 2.5 illustrates the effect of the restart strategy, for different values of p, on the example of a multi-modal benchmark function f_{18} (see section 2.7.2 for details), where most runs get caught in local optima. Whenever used in the rest of the chapter, the fraction is $p = \frac{1}{5}$.

Let $s(t)$ be the success probability of the underlying search algorithm at time t. Here, time is measured by the number of generations or fitness evaluations. Accordingly, let $S_p(t)$ be the boosted success probability of the restart scheme with parameter p. Approximately, that is, assuming continuous time, the probabilities are connected by the formula

$$S_p(t) = 1 - \prod_{i=1}^{\infty} \left[1 - s\big(p(1-p)^{i-1}t\big)\right].$$

Two qualitative properties of the restart strategy can be deduced from this formula, even if in discrete time the sum is finite ($i \leq \log_2(t)$), and the times $p(1-p)^{i-1}t$ need to be rounded:

- If there exists t_0 such that $s(t_0) > 0$ then $\lim_{t\to\infty} S_p(t) = 1$ for all $p \in (0, 1)$.

- For sufficiently small t, we have $S_p(t) \leq s_p(t)$.

This captures the expected effect that the restart strategy results in an initial slowdown, but eventually solves the problem reliably.

2.5 Techniques for Multinormal Distributions

In this section we will describe two crucial techniques to enhance performance of the NES algorithm as applied to multinormal distributions. First, the method of exponential parameterization is introduced, guaranteeing that the covariance matrix stays positive-definite. Second, a novel method for changing the coordinate system into a "natural" one is laid out, making the algorithm computationally efficient.

2.5.1 Using Exponential Parameterization

Gradient steps on the covariance matrix $\mathbf{\Sigma}$ result in a number of technical problems. When updating $\mathbf{\Sigma}$ directly with the gradient step $\nabla_{\mathbf{\Sigma}} J$, we have to ensure that $\mathbf{\Sigma} + \eta \nabla_{\mathbf{\Sigma}} J$ remains a valid, positive definite covariance matrix. This is not guaranteed *a priori*, because the (natural) gradient $\nabla_{\mathbf{\Sigma}} J$ may be any symmetric matrix. If we instead update a factor $\mathbf{A}$ of $\mathbf{\Sigma}$, it is at least ensured that $\mathbf{A}^\top \mathbf{A}$ is symmetric and positive semi-definite. But when shrinking an eigenvalue of $\mathbf{A}$ it may happen that the gradient step swaps the sign of the eigenvalue, resulting in undesired oscillations.

An elegant way to fix these problems is to represent the covariance matrix using the *exponential map* for symmetric matrices (see e.g., Glasmachers and Igel, 2005 for a related approach). Let

$$\mathcal{S}_d := \left\{ \mathbf{M} \in \mathbb{R}^{d \times d} \,\middle|\, \mathbf{M}^\top = \mathbf{M} \right\}$$

and

$$\mathcal{P}_d := \left\{ \mathbf{M} \in \mathcal{S}_d \,\middle|\, \mathbf{v}^\top \mathbf{M} \mathbf{v} > 0 \text{ for all } \mathbf{v} \in \mathbb{R}^d \setminus \{0\} \right\}$$

denote the vector space of symmetric and the (cone) manifold of symmetric positive definite matrices, respectively. Then the exponential map

$$\exp : \mathcal{S}_d \to \mathcal{P}_d \,, \qquad \mathbf{M} \mapsto \sum_{n=0}^{\infty} \frac{\mathbf{M}^n}{n!} \tag{2.10}$$

is a diffeomorphism: The map is bijective, and both exp as well as its inverse map $\log : \mathcal{P}_d \to \mathcal{S}_d$ are smooth. The mapping can be computed in cubic time, for example by decomposing the matrix $\mathbf{M} = \mathbf{U}\mathbf{D}\mathbf{U}^\top$ into orthogonal $\mathbf{U}$ (eigen-vectors) and diagonal $\mathbf{D}$ (eigen-values), taking the exponential of $\mathbf{D}$ (which

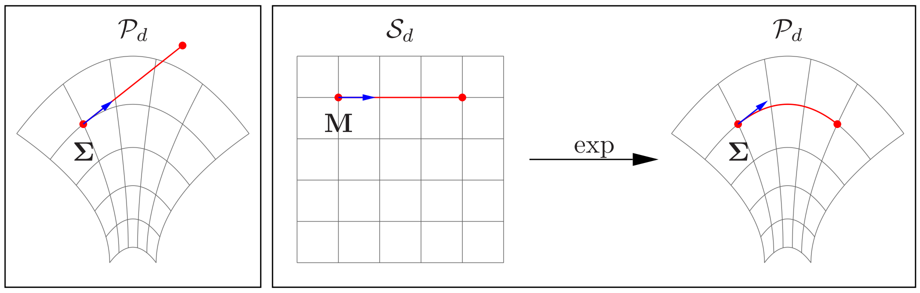


Figure 2.6: **Left:** updating a covariance matrix $\boldsymbol{\Sigma}$ directly can end up outside the manifold of symmetric positive-definite matrices $\mathcal{P}_d$. **Right:** first performing the update on $\mathbf{M} = \frac{1}{2}\log(\boldsymbol{\Sigma})$ in $\mathcal{S}_d$ and then mapping back the result into the original space $\mathcal{P}_d$ using the exponential map is both *safe* (guaranteed to stay in the manifold) and *straight* (the update follows a geodesic).

amounts to taking the element-wise exponentials of the diagonal entries), and composing everything back[2] as $\exp(\mathbf{M}) = \mathbf{U}\exp(\mathbf{D})\mathbf{U}^\top$.

Thus, we can represent the covariance matrix $\boldsymbol{\Sigma} \in \mathcal{P}_d$ as $\exp(\mathbf{M})$ with $\mathbf{M} \in \mathcal{S}_d$. The resulting gradient update for $\mathbf{M}$ has two important properties: First, because $\mathcal{S}_d$ is a *vector space*, any update automatically corresponds to a valid covariance matrix.[3] Second, the update of $\mathbf{M}$ makes the gradient invariant w.r.t. linear transformations of the search space $\mathbb{R}^d$. This follows from an information geometric perspective, viewing $\mathcal{P}_d$ as the Riemannian parameter manifold equipped with the Fisher information metric. The property is a direct consequence of the Cartan-Hadamard theorem (Cartan, 1928). See also Figure 2.6 for an illustration.

However, the exponential parameterization considerably complicates the computation of the Fisher information matrix $\mathbf{F}$, which now involves partial derivatives of the matrix exponential (2.10). This can be done in cubic time per partial derivative according to Najfeld and Havel (1994), resulting in an unacceptable complexity of $\mathcal{O}(d^7)$ for the computation of the Fisher matrix.

2.5.2 Using Natural Coordinates

In this section we describe a technique that allows us to avoid the computation of the Fisher information matrix altogether, for some specific but common classes of distributions. The idea is to iteratively change the coordinate system in such a way that it becomes possible to follow the natural gradient without any costly inverses of the Fisher matrix (actually, without even constructing it explicitly). We introduce the technique for the simplest case of multinormal search distributions, and in section 2.6.2, we generalize it to the whole class of distributions that they are applicable to (namely, rotationally-symmetric distributions).

[2]The same computation works for the logarithm, and thus also for powers $\mathcal{P}_d \to \mathcal{P}_d$, $\mathbf{M} \mapsto \mathbf{M}^c = \exp(c \cdot \log(\mathbf{M}))$ for all $c \in \mathbb{R}$, for example for the (unique) square root $(c = 1/2)$.
[3]The tangent bundle $T\mathcal{P}_d$ of the manifold $\mathcal{P}_d$ is isomorphic to $\mathcal{P}_d \times \mathcal{S}_d$ and globally trivial. Thus, arbitrarily large gradient steps are meaningful in this representation.

Instead of using the 'global' coordinates $\mathbf{\Sigma} = \exp(\mathbf{M})$ for the covariance matrix, we linearly transform the coordinate system in each iteration to a coordinate system in which the current search distribution is the standard normal distribution with zero mean and unit covariance. Let the current search distribution be given by $(\boldsymbol{\mu}, \mathbf{A}) \in \mathbb{R}^d \times \mathcal{P}_d$ with $\mathbf{A}^\top \mathbf{A} = \mathbf{\Sigma}$. We use the tangent space $T_{(\boldsymbol{\mu},\mathbf{A})}(\mathbb{R}^d \times \mathcal{P}_d)$ of the parameter manifold $\mathbb{R}^d \times \mathcal{P}_d$, which is isomorphic to the vector space $\mathbb{R}^d \times \mathcal{S}_d$, to represent the updated search distribution as

$$(\boldsymbol{\delta},\, \mathbf{M}) \mapsto (\boldsymbol{\mu}_{\text{new}},\, \mathbf{A}_{\text{new}}) = \left(\boldsymbol{\mu} + \mathbf{A}^\top \boldsymbol{\delta},\, \mathbf{A} \exp\left(\frac{1}{2}\mathbf{M}\right)\right). \tag{2.11}$$

This coordinate system is *natural* in the sense that the Fisher matrix w.r.t. an orthonormal basis of $(\boldsymbol{\delta}, \mathbf{M})$ is the identity matrix. The current search distribution $\mathcal{N}(\boldsymbol{\mu}, \mathbf{A}^\top \mathbf{A})$ is encoded as

$$\pi(\mathbf{z}|\boldsymbol{\delta}, \mathbf{M}) = \mathcal{N}\left(\boldsymbol{\mu} + \mathbf{A}^\top \boldsymbol{\delta},\ \mathbf{A}^\top \exp(\mathbf{M})\mathbf{A}\right),$$

where at each step we change the coordinates such that $(\boldsymbol{\delta}, \mathbf{M}) = (0, 0)$. In this case, it is guaranteed that for the variables $(\boldsymbol{\delta}, \mathbf{M})$ the plain gradient and the natural gradient coincide ($\mathbf{F} = \mathbb{I}$). Consequently the computation of the natural gradient costs $\mathcal{O}(d^3)$ operations.

In the new coordinate system we produce standard normal samples $\mathbf{s} \sim \mathcal{N}(0, \mathbb{I})$ which are then mapped back into the original coordinates $\mathbf{z} = \boldsymbol{\mu} + \mathbf{A}^\top \cdot \mathbf{s}$. The log-density becomes

$$\begin{aligned}\log \pi(\mathbf{z} \,|\, \boldsymbol{\delta}, \mathbf{M}) = &- \frac{d}{2}\log(2\pi) - \log\big(\det(\mathbf{A})\big) - \frac{1}{2}\operatorname{tr}(\mathbf{M}) \\ &- \frac{1}{2}\left\| \exp\left(-\frac{1}{2}\mathbf{M}\right) \mathbf{A}^{-\top} \cdot (\mathbf{z} - (\boldsymbol{\mu} + \mathbf{A}^\top \boldsymbol{\delta}))\right\|^2,\end{aligned}$$

and thus the log-derivatives (at $\boldsymbol{\delta} = 0$, $\mathbf{M} = 0$) take the following, surprisingly simple forms:

$$\begin{aligned}\nabla_{\boldsymbol{\delta}}\big|_{\boldsymbol{\delta}=0} \log \pi(\mathbf{z} \,|\, \mathbf{M} = 0, \boldsymbol{\delta}) &= \nabla_{\boldsymbol{\delta}}\big|_{\boldsymbol{\delta}=0} \left[-\frac{1}{2}\left\|\mathbf{A}^{-\top} \cdot (\mathbf{z} - (\boldsymbol{\mu} + \mathbf{A}^\top \boldsymbol{\delta}))\right\|^2\right] \\ &= -\frac{1}{2}\left[-2 \cdot \mathbf{A}^{-\top}\mathbf{A}^\top \cdot \mathbf{A}^{-\top} \cdot (\mathbf{z} - (\boldsymbol{\mu} + \mathbf{A}^\top \boldsymbol{\delta}))\right]\Big|_{\boldsymbol{\delta}=0} \\ &= \mathbf{A}^{-\top}(\mathbf{z} - \boldsymbol{\mu}) \\ &= \mathbf{s} \end{aligned} \tag{2.12}$$

$$
\begin{aligned}
\nabla_{\mathbf{M}}\big|_{\mathbf{M}=0} \log \pi(\mathbf{z} \,|\, \boldsymbol{\delta}=0,\mathbf{M}) &= -\frac{1}{2}\nabla_{\mathbf{M}}\big|_{\mathbf{M}=0}\left[\mathrm{tr}(\mathbf{M}) + \left\| \exp\left(-\frac{1}{2}\mathbf{M}\right)\mathbf{A}^{-\top}(\mathbf{z}-\boldsymbol{\mu})\right\|^2\right] \\
&= -\frac{1}{2}\Big[\mathbb{I} + 2\cdot\left(-\frac{1}{2}\right)\cdot[\mathbf{A}^{-\top}(\mathbf{z}-\boldsymbol{\mu})] \\
&\qquad \cdot \exp\left(-\frac{1}{2}\mathbf{M}\right)\cdot[\mathbf{A}^{-\top}(\mathbf{z}-\boldsymbol{\mu})]^{\top}\Big]\bigg|_{\mathbf{M}=0} \\
&= -\frac{1}{2}\left[\mathbb{I} - [\mathbf{A}^{-\top}(\mathbf{z}-\boldsymbol{\mu})]\cdot\mathbb{I}\cdot[\mathbf{A}^{-\top}(\mathbf{z}-\boldsymbol{\mu})]^{\top}\right] \\
&= \frac{1}{2}(\mathbf{s}\mathbf{s}^{\top} - \mathbb{I}) \qquad (2.13)
\end{aligned}
$$

These results give us the updates in the natural coordinate system

$$\nabla_{\boldsymbol{\delta}} J = \sum_{k=1}^{\lambda} f(\mathbf{z}_k)\cdot\mathbf{s}_k \qquad (2.14)$$

$$\nabla_{\mathbf{M}} J = \sum_{k=1}^{\lambda} f(\mathbf{z}_k)\cdot(\mathbf{s}_k\mathbf{s}_k^{\top} - \mathbb{I}) \qquad (2.15)$$

which are then mapped back onto $(\boldsymbol{\mu}, \mathbf{A})$ using equation (2.11):

$$
\begin{aligned}
\boldsymbol{\mu}_{\text{new}} \leftarrow \boldsymbol{\mu} + \mathbf{A}^{\top}\boldsymbol{\delta} &= \boldsymbol{\mu} + \eta\mathbf{A}^{\top}\nabla_{\boldsymbol{\delta}} J \\
&= \boldsymbol{\mu} + \eta\mathbf{A}^{\top}\left(\frac{1}{\lambda}\sum_{k=1}^{\lambda}\nabla_{\boldsymbol{\delta}}\log\pi(\mathbf{z}_k|\theta)\cdot f(\mathbf{z}_k)\right) \\
&= \boldsymbol{\mu} + \frac{\eta}{\lambda}\mathbf{A}^{\top}\left(\sum_{k=1}^{\lambda} f(\mathbf{z}_k)\cdot\mathbf{s}_k\right) \\
\mathbf{A}_{\text{new}} \leftarrow \mathbf{A}\cdot\exp\left(\frac{1}{2}\mathbf{M}\right) &= \mathbf{A}\cdot\exp\left(\eta\frac{1}{2}\nabla_{\mathbf{M}} J\right) \\
&= \mathbf{A}\cdot\exp\left(\frac{\eta}{2}\cdot\frac{1}{\lambda}\sum_{k=1}^{\lambda}\nabla_{\mathbf{M}}\log\pi(\mathbf{z}_k|\theta)\cdot f(\mathbf{z}_k)\right) \\
&= \mathbf{A}\cdot\exp\left(\frac{\eta}{4\lambda}\sum_{k=1}^{\lambda} f(\mathbf{z}_k)\cdot(\mathbf{s}_k\mathbf{s}_k^{\top} - \mathbb{I})\right)
\end{aligned}
$$

2.5.3 Orthogonal Decomposition of Multinormal Parameter Space

We decompose the parameter vector space $(\boldsymbol{\delta},\mathbf{M}) \in \mathbb{R}^d \times \mathcal{S}_d$ into the product

$$\mathbb{R}^d \times \mathcal{S}_d = \underbrace{\mathbb{R}^d}_{(\boldsymbol{\delta})} \times \underbrace{\mathcal{S}_d^{\parallel}}_{(\sigma)} \times \underbrace{\mathcal{S}_d^{\perp}}_{(\mathbf{B})} \;,$$

of orthogonal subspaces. The one-dimensional space $\mathcal{S}_d^{\parallel} = \{\lambda\cdot\mathbb{I} \,|\, \lambda \in \mathbb{R}\}$ is spanned by the identity matrix $\mathbb{I}$, and $\mathcal{S}_d^{\perp} = \{\mathbf{M} \in \mathcal{S}_d \,|\, \mathrm{tr}(\mathbf{M}) = 0\}$ denotes its orthogonal complement in $\mathcal{S}_d$. The different components have roles with clear interpretations: The ($\boldsymbol{\delta}$)-component $\nabla_{\boldsymbol{\delta}} J$ describes the update of the center of the search distribution, the (σ)-component with value $\nabla_\sigma J \cdot \mathbb{I}$ for $\nabla_\sigma J = \operatorname{tr}(\nabla_{\mathbf{M}} J)/d$ has the role of a step size update, which becomes clear from the identity $\det(\exp(\mathbf{M})) = \exp(\operatorname{tr}(\mathbf{M}))$, and the ($\mathbf{B}$)-component $\nabla_{\mathbf{B}} J = \nabla_{\mathbf{M}} J - \nabla_\sigma J$ describes the update of the transformation matrix, normalized to unit determinant, which can thus be attributed to the shape of the search distribution. This decomposition is canonical in being the finest decomposition such that updates of its components result in of the search algorithm under linear transformations of the search space.

On these subspaces we introduce independent learning rates $\eta_{\boldsymbol{\delta}}$, η_σ, and $\eta_{\mathbf{B}}$, respectively. For simplicity we also split the transformation matrix $\mathbf{A} = \sigma \cdot \mathbf{B}$ into the step size $\sigma \in \mathbb{R}^+$ and the normalized transformation matrix $\mathbf{B}$ with $\det(\mathbf{B}) = 1$. Then the resulting update is

$$\boldsymbol{\mu}_{\text{new}} = \boldsymbol{\mu} + \eta_{\boldsymbol{\delta}} \cdot \nabla_{\boldsymbol{\delta}} J = \boldsymbol{\mu} + \eta_{\boldsymbol{\delta}} \cdot \sum_{k=1}^{\lambda} f(\mathbf{z}_k) \cdot \mathbf{s}_k \tag{2.16}$$

$$\sigma_{\text{new}} = \sigma \cdot \exp\left(\frac{\eta_\sigma}{2} \cdot \nabla_\sigma\right) = \sigma \cdot \exp\left(\frac{\eta_\sigma}{2} \cdot \frac{\operatorname{tr}(\nabla_{\mathbf{M}} J)}{d}\right) \tag{2.17}$$

$$\mathbf{B}_{\text{new}} = \mathbf{B} \cdot \exp\left(\frac{\eta_{\mathbf{B}}}{2} \cdot \nabla_{\mathbf{B}} J\right) = \mathbf{B} \cdot \exp\left(\frac{\eta_{\mathbf{B}}}{2} \cdot \left(\nabla_{\mathbf{M}} J - \frac{\operatorname{tr}(\nabla_{\mathbf{M}} J)}{d} \cdot \mathbb{I}\right)\right), \tag{2.18}$$

with $\nabla_{\mathbf{M}} J$ from equation (2.15). In case of $\eta_\sigma = \eta_{\mathbf{B}}$, in this case referred to as $\eta_{\mathbf{A}}$, the updates (2.17) and (2.18) simplify to

$$\begin{aligned}\mathbf{A}_{\text{new}} &= \mathbf{A} \cdot \exp\left(\frac{\eta_{\mathbf{A}}}{2} \cdot \nabla_{\mathbf{M}}\right) \qquad (2.19)\\ &= \mathbf{A} \cdot \exp\left(\frac{\eta_{\mathbf{A}}}{2} \cdot \sum_{k=1}^{\lambda} f(\mathbf{z}_k) \cdot (\mathbf{s}_k \mathbf{s}_k^\top - \mathbb{I})\right).\end{aligned}$$

The resulting algorithm is called *exponential* NES (xNES), and shown in Algorithm 5. We also give the pseudocode for its hill-climber variant (see also section 2.5.5).

Updating the search distribution in the natural coordinate system is an alternative to the exponential parameterization (section 2.5.1) for making the algorithm invariant under linear transformations of the search space, which is then achieved in a direct and constructive way.

2.5.4 Connection to CMA-ES

It has been noticed independently by Glasmachers et al. (2010a) and Akimoto et al. (2010) that the natural gradient updates of xNES and the strategy updates of the CMA-ES algorithm (Hansen and Ostermeier, 2001) are closely connected. However, since xNES does not feature evolution paths, this connection

Algorithm 5: Exponential Natural Evolution Strategies (xNES), for multinormal distributions

```
   input: $f$, $\boldsymbol{\mu}_{init}$, $\boldsymbol{\Sigma}_{init} = \mathbf{A}^\top \mathbf{A}$

1  initialize  $\sigma \leftarrow \sqrt[d]{|\det(\mathbf{A})|}$
               $\mathbf{B} \leftarrow \mathbf{A}/\sigma$
2  repeat
3      for $k = 1 \ldots \lambda$ do
4          draw sample $\mathbf{s}_k \sim \mathcal{N}(0, \mathbb{I})$
5          $\mathbf{z}_k \leftarrow \boldsymbol{\mu} + \sigma \mathbf{B}^\top \mathbf{s}_k$
6          evaluate the fitness $f(\mathbf{z}_k)$
7      end
8      sort $\{(\mathbf{s}_k, \mathbf{z}_k)\}$ with respect to $f(\mathbf{z}_k)$ and compute utilities $u_k$

9      compute gradients
         $\nabla_{\boldsymbol{\delta}} J \leftarrow \sum_{k=1}^{\lambda} u_k \cdot \mathbf{s}_k$        $\nabla_{\mathbf{M}} J \leftarrow \sum_{k=1}^{\lambda} u_k \cdot (\mathbf{s}_k \mathbf{s}_k^\top - \mathbb{I})$
         $\nabla_{\sigma} J \leftarrow \mathrm{tr}(\nabla_{\mathbf{M}} J)/d$        $\nabla_{\mathbf{B}} J \leftarrow \nabla_{\mathbf{M}} J - \nabla_{\sigma} J \cdot \mathbb{I}$

                             $\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} + \eta_{\boldsymbol{\delta}} \cdot \sigma \mathbf{B} \cdot \nabla_{\boldsymbol{\delta}} J$
10     update parameters     $\sigma \leftarrow \sigma \cdot \exp(\eta_\sigma/2 \cdot \nabla_\sigma J)$
                             $\mathbf{B} \leftarrow \mathbf{B} \cdot \exp(\eta_{\mathbf{B}}/2 \cdot \nabla_{\mathbf{B}} J)$
11 until stopping criterion is met
```

is restricted to the so-called rank-μ-update (in the terminology of this study, rank-λ-update) of CMA-ES.

First of all observe that xNES and CMA-ES share the same invariance properties. But more interestingly, although derived from different heuristics, their updates turn out to be nearly equivalent. A closer investigation of this equivalence promises synergies and new perspectives on the working principles of both algorithms. In particular, this insight shows that CMA-ES can be explained as a natural gradient algorithm, which may allow for a more thorough analysis of its updates, and xNES can profit from CMA-ES's mature settings of algorithms parameters, such as search space dimension-dependent population sizes, learning rates and utility values.

Both xNES and CMA-ES parameterize the search distribution with three functionally different parameters for mean, scale, and shape of the distribution. xNES uses the parameters $\boldsymbol{\mu}$, σ, and $\mathbf{B}$, while the covariance matrix is represented as $\sigma^2 \cdot \mathbf{C}$ in CMA-ES, where $\mathbf{C}$ can by any positive definite symmetric matrix. Thus, the representation of the scale of the search distribution is shared among σ and $\mathbf{C}$ in CMA-ES, and the role of the additional parameter σ is to allow for an adaptation of the step size on a faster time scale than the full covariance update. In contrast, the NES updates of scale and shape parameters σ and $\mathbf{B}$ are properly decoupled.

Let us start with the updates of the center parameter $\boldsymbol{\mu}$. The update (2.16) is very similar to the update of the center of the search distribution in CMA-ES, see (Hansen and Ostermeier, 2001). The utility function exactly takes the role of the weights in CMA-ES, which assumes a fixed learning rate of one. For the covariance matrix, the situation is more complicated. From equation (2.19) we deduce the update rule

$$\begin{aligned}\mathbf{\Sigma}_{\text{new}} =& (\mathbf{A}_{\text{new}})^\top \cdot \mathbf{A}_{\text{new}} \\ =& \mathbf{A}^\top \cdot \exp\left(\eta_{\mathbf{\Sigma}} \cdot \sum_{k=1}^{\lambda} u_k \left(\mathbf{s}_k \mathbf{s}_k^\top - \mathbb{I}\right)\right) \cdot \mathbf{A}\end{aligned}$$

for the covariance matrix, with learning rate $\eta_{\mathbf{\Sigma}} = \eta_{\mathbf{A}}$. This term is closely connected to the exponential parameterization of the natural coordinates in xNES, while CMA-ES is formulated in global linear coordinates. The connection of these updates can be shown either by applying the xNES update directly to the natural coordinates without the exponential parameterization, or by approximating the exponential map by its first order Taylor expansion. Akimoto et al. (2010) established the same connection directly in coordinates based on the Cholesky decomposition of $\mathbf{\Sigma}$, see (Sun et al., 2009a,b). The arguably simplest derivation of the equivalence relies on the invariance of the natural gradient under coordinate transformations, which allows us to perform the computation, without loss of generality, in natural coordinates. We use the first order Taylor approximation of exp to obtain

$$\exp\left(\eta_{\mathbf{\Sigma}} \cdot \sum_{k=1}^{\lambda} u_k \left(\mathbf{s}_k \mathbf{s}_k^\top - \mathbb{I}\right)\right) \approx \mathbb{I} + \eta_{\mathbf{\Sigma}} \cdot \sum_{k=1}^{\lambda} u_k \left(\mathbf{s}_k \mathbf{s}_k^\top - \mathbb{I}\right) \quad ,$$

so the first order approximate update yields

$$\begin{aligned}\mathbf{\Sigma}'_{new} =& \mathbf{A}^\top \cdot \left(\mathbb{I} + \eta_{\mathbf{\Sigma}} \cdot \sum_{k=1}^{\lambda} u_k \left(\mathbf{s}_k \mathbf{s}_k^\top - \mathbb{I}\right)\right) \cdot \mathbf{A} \\ =& (1 - U \cdot \eta_{\mathbf{\Sigma}}) \cdot \mathbf{A}^\top \mathbf{A} + \eta_{\mathbf{\Sigma}} \cdot \sum_{k=1}^{\lambda} u_k \left(\mathbf{A}^\top \mathbf{s}_k\right) \left(\mathbf{A}^\top \mathbf{s}_k\right)^\top \\ =& (1 - U \cdot \eta_{\mathbf{\Sigma}}) \cdot \mathbf{\Sigma} + \eta_{\mathbf{\Sigma}} \cdot \sum_{k=1}^{\lambda} u_k \left(\mathbf{z}_k - \boldsymbol{\mu}\right) \left(\mathbf{z}_k - \boldsymbol{\mu}\right)^\top\end{aligned}$$

with $U = \sum_{k=1}^{\lambda} u_k$, from which the connection to the CMA-ES rank-μ-update is obvious (see Hansen and Ostermeier, 2001). Finally, the updates of the global step size parameter σ turn out to be identical in xNES and CMA-ES without evolution paths.

Having highlighted the similarities, let us have a closer look at the differences between xNES and CMA-ES, which are mostly two aspects. CMA-ES uses the well-established technique of evolution paths to smoothen out random effects over multiple generations. This technique is particularly valuable when working with minimal population sizes, which is the default for both algorithms. Thus, evolution paths are expected to improve stability; further interpretations have been provided by Hansen and Ostermeier (2001). However, the presence of evolution paths has the conceptual disadvantage that the state of the CMA-ES algorithms is not completely described by its search distribution. The other difference between xNES and CMA-ES is the exponential parameterization of the updates in xNES, which results in a multiplicative update equation for the covariance matrix, in contrast to the additive update of CMA-ES. We argue that just like for the global step size σ, the multiplicative update of the covariance matrix is natural.

A valuable perspective offered by the natural gradient updates in xNES is the derivation of the updates of the center $\boldsymbol{\mu}$, the step size σ, and the normalized transformation matrix $\mathbf{B}$, all from the *same* principle of natural gradient ascent. In contrast, the updates applied in CMA-ES result from different heuristics for each parameter. Hence, it is even more surprising that the two algorithms are closely connected. This connection provides a post-hoc justification of the various heuristics employed by CMA-ES, and it highlights the consistency of the intuition behind these heuristics.

2.5.5 Elitism

The principle of the NES algorithm is to follow the natural gradient of expected fitness. This requires sampling the fitness gradient. Naturally, this amounts to what, within the realm of evolution strategies, is generally referred to as "comma-selection", that is, updating the search distribution based solely on the current batch of "offspring" samples, disregarding older samples such as the "parent" population. This seems to exclude approaches that retain some of the best samples, like elitism, hill-climbing, or even steady-state selection (Goldberg, 1989). In this section we show that elitism (and thus a wide variety of selection schemes) is indeed compatible with NES. We exemplify this technique by deriving a NES algorithm with (1+1) selection, i.e., a hill-climber.

It is impossible to estimate any information about the fitness gradient from a single sample, since at least two samples are required to estimate even a finite difference. The (1+1) selection scheme indicates that this dilemma can be resolved by considering two distinguished samples, namely the elitist or parent $\mathbf{z}^{\text{parent}} = \boldsymbol{\mu}$, and the offspring $\mathbf{z}$. Considering these two samples in the update is in principle sufficient for estimating the fitness gradient w.r.t. the parameters θ.

Care needs to be taken for setting the algorithm parameters, such as learning rates and utility values. The extremely small population size of one indicates that learning rates should generally be small in order to ensure stability of the algorithm. Another guideline is the well known observation (Rechenberg and Eigen, 1973) that a step size resulting in a success rate of roughly $1/5$ maximizes progress. This indicates that a self-adaptation strategy should increase the learning rate in case of too many successes, and decrease it when observing too few successes.

Let us consider the basic case of *radial* Gaussian search distributions

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

with parameters $\boldsymbol{\mu} \in \mathbb{R}^d$ and $\sigma > 0$. We encode these parameters as $\theta = (\boldsymbol{\mu}, \ell)$ with $\ell = \log(\sigma)$. Let $\mathbf{s} \sim \mathcal{N}(0, 1)$ be a standard normally distributed vector, then we obtain the offspring as $\mathbf{z} = \boldsymbol{\mu} + \sigma \cdot \mathbf{s} \sim \mathcal{N}(\boldsymbol{\mu}, \sigma^2)$, and the natural gradient components are

$$\begin{aligned}
\tilde{\nabla}_{\boldsymbol{\mu}} J &= u_1^{(\boldsymbol{\mu})} \cdot \mathbf{0} + u_2^{(\boldsymbol{\mu})} \cdot \sigma \cdot \mathbf{s} \\
\tilde{\nabla}_{\ell} J &= u_1^{(\ell)} \cdot (-1) + u_2^{(\ell)} \cdot (\|\mathbf{s}\|^2 - 1).
\end{aligned}$$

The corresponding strategy parameter updates read

$$\begin{aligned}
\boldsymbol{\mu} &\leftarrow \boldsymbol{\mu} + \eta_{\boldsymbol{\mu}} \cdot \left[u_1^{(\boldsymbol{\mu})} \cdot \mathbf{0} + u_2^{(\boldsymbol{\mu})} \cdot \sigma \cdot \mathbf{s}\right] \\
\sigma &\leftarrow \sigma \cdot \exp\left(\eta_\ell \cdot \left[u_1^{(\ell)} \cdot (-1) + u_2^{(\ell)} \cdot (\|\mathbf{s}\|^2 - 1)\right]\right).
\end{aligned}$$

The indices 1 and 2 of the utility values refer to the ‘samples’ $\boldsymbol{\mu}$ and $\mathbf{z}$, namely parent and offspring. Note that these samples correspond to the vectors $\mathbf{0}$ and $\mathbf{s}$ in the above update equations (see also section 2.5.2). The superscripts of the utility values indicate the different parameters. Now elitist selection and the 1/5 rule dictate the settings of these utility values as follows:

- The elitist rule requires that the mean remains unchanged in case of no success ($u_1^{(\boldsymbol{\mu})} = 1$ and $u_2^{(\boldsymbol{\mu})} = 0$), and that the new sample replaces the mean in case of success ($u_1^{(\boldsymbol{\mu})} = 0$ and $u_2^{(\boldsymbol{\mu})} = 1$, with a learning rate of $\eta_{\boldsymbol{\mu}} = 1$).
- Setting the utilities for ℓ to $u_1^{(\ell)} = 1$ and $u_2^{(\ell)} = 0$ in case of no success, effectively reduces the learning rate. Setting $u_1^{(\ell)} = -5$ and $u_2^{(\ell)} = 0$ in case of success has the opposite effect and roughly implements the 1/5-rule. The self-adaptation process can be stabilized with a small learning rate η_ℓ.

Note that we change the utility values based on success or failure of the offspring. This seems natural, since the utility of information encoded in the sample $\mathbf{z}$ depends on its success. Highlighting elitism in the selection, we call these utility values success-based. This is similar but not equivalent to rank-based utilities for the joint population $\{\boldsymbol{\mu}, \mathbf{z}\}$.

The NES hill-climber for radial Gaussian search distributions is illustrated in algorithm 6. This formulation offers a more standard perspective on the hill-climber by using explicit case distinctions for success and failure of the offspring instead of success-based utilities. The same procedure can be generalized to more flexible search distributions. A conservative strategy is to update further shape-related parameters only in case of success, which can be expressed by means of success-based utility values in the very same way. The corresponding algorithm for multi-variate Gaussian search distributions is Algorithm 7 (in section 2.5.2) and for multi-variate Cauchy it is Algorithm 9 (in section 2.6.3).

Algorithm 6: (1+1)-NES: radial Gaussian distribution

```
   input: $f$, $\boldsymbol{\mu}_{init}$, $\sigma_{init}$
1  $f_{\text{best}} \leftarrow f(\boldsymbol{\mu}_{init})$
2  repeat
3      draw sample $\mathbf{s} \sim \mathcal{N}(0,1)$
4      create offspring $\mathbf{z} \leftarrow \boldsymbol{\mu} + \sigma \cdot \mathbf{s}$
5      evaluate the fitness $f(\mathbf{z})$
6      if $f(\mathbf{z}) > f_{best}$ then
7          update mean $\boldsymbol{\mu} \leftarrow \mathbf{z}$
8          $\sigma \leftarrow \sigma \cdot \exp(5\eta_\sigma)$
9          $f_{\text{best}} \leftarrow f(\mathbf{z})$
10     else
11         $\sigma \leftarrow \sigma \cdot \exp(-\eta_\sigma)$
12     end
13 until stopping criterion is met;
```

Algorithm 7: (1+1)-xNES: multi-variate Gaussian distribution

```
   input: $f$, $\boldsymbol{\mu}_{init}$, $\boldsymbol{\Sigma}_{init} = \mathbf{A}^\top \mathbf{A}$
1  $f_{max} \leftarrow -\infty$
2  repeat
3      draw sample $\mathbf{s} \sim \mathcal{N}(0, \mathbb{I})$
4      evaluate the fitness $f(\mathbf{z} = \mathbf{A}^\top \mathbf{s} + \boldsymbol{\mu})$
5      calculate log-derivatives $\nabla_{\mathbf{M}} \log \pi\left(\mathbf{z}|\theta\right) = \frac{1}{2}\left(\mathbf{s}\mathbf{s}^\top - \mathbb{I}\right)$
6      $(\mathbf{z}_1, \mathbf{z}_2) \leftarrow (\boldsymbol{\mu}, \mathbf{z})$
7      if $f_{max} < f(\mathbf{z})$ then
8          $f_{max} \leftarrow f(\mathbf{z})$
9          $\mathbf{u} \leftarrow (-4, 1)$
10         update mean $\boldsymbol{\mu} \leftarrow \mathbf{z}$
11     else
12         $\mathbf{u} \leftarrow (\frac{4}{5}, 0)$
13     end
14     $\nabla_{\mathbf{M}} J \leftarrow \frac{1}{2}\sum_{k=1}^{2} \nabla_{\mathbf{M}} \log \pi\left(\mathbf{z}_k|\theta\right) \cdot u_k = -\frac{u_1}{2}\mathbb{I} + \frac{u_2}{4}\left(\mathbf{s}\mathbf{s}^\top - \mathbb{I}\right)$
15     $\mathbf{A} \leftarrow \mathbf{A} \cdot \exp\left(\frac{1}{2}\eta \nabla_{\mathbf{M}} J\right)$
16 until stopping criterion is met
```

Use for Multi-objective NES

Using the hill-climber version as an ingredient, we can follow the successful scheme developed in (Igel et al., 2007) to design *multi-objective* NES algorithm (MO-NES). It maintains a population of (1+1)-NES hill-climbers, with the goal of maximally approximating the Pareto-front. Each generation, the $k \in \mathbb{N}$ hill-climbers generate one offspring each. As in multi-objective optimization there is no unique notion of success due to multiple contradicting objectives, the selection scheme of the individual hill-climbers becomes meaningless. Instead, we adopt the indicator-based selection scheme used in (Igel et al., 2007) which consists of two stages. Parents and offspring are merged into a single population and ranked according to (1) the dominance relation, and (2) an indicator that permits aggregating the relative value of each individual within its front into a single number (in contrast to the m-dimensional fitness vector), for example the hypervolume contribution (Zitzler and Thiele, 1998). Selection then amounts to keeping the best k out of $2k$ individuals according to this two-tiered ranking. The resulting MO-NES algorithm is constructed by replacing the (1+1)-CMA-ES module from MO-CMA-ES (Igel et al., 2007) with a NES hill-climber, usually (1+1)-xNES (see Glasmachers et al., 2010a for details).

2.6 Beyond Multinormal Distributions

In the previous section we have seen how natural gradient ascent is applied to multi-variate normal distributions, which arguably constitute the most important class of search distributions in modern evolution strategies. In this section we expand the NES framework in breadth, motivating the usefulness and deriving a number of NES variants with different search distributions.

2.6.1 Separable NES

Adapting the full covariance matrix of the search distribution can be disadvantageous, particularly in high-dimensional search spaces, for two reasons.

For many problems it can be safely assumed that the computational costs are governed by the number of fitness evaluations. This is particularly true if such evaluations rely on expensive simulations. However, for applications where fitness evaluations scale gracefully with the search space dimension, the $\mathcal{O}(d^3)$ xNES update (due to the matrix exponential) can dominate the computation[4]. One such application is the evolutionary training of recurrent neural networks (i.e., neuro-evolution), where the number of weights d in the network can grow quadratically with the number of neurons k, resulting in a complexity of $\mathcal{O}(k^6)$ for a single NES update.

A second reason not to adapt the full covariance matrix in high dimensional search spaces is sample efficiency. The covariance matrix has $d(d+1)/2 \in \mathcal{O}(d^2)$ degrees of freedom, which can be huge in large dimensions. Obtaining a stable estimate of this matrix based on samples may thus require many (costly) fitness evaluations, in turn requiring very small learning rates. As a result, the algorithm may simply not have enough time to adapt its search distribution to the problem with a given budget of fitness evaluations. In this case, it may be

[4] Even if we sacrifice the exponential parameterization (which leads to other problems, see section 2.5.1) to perform more efficient updates, a cost of $\mathcal{O}(d^2)$ is unavoidable, already for sampling offspring.

advantageous to restrict the class of search distributions in order to adapt at all, even if this results in a less steep learning curve in the (then practically irrelevant) limit of infinitely many fitness evaluations.

The only two distinguished parameter subsets of a multi-variate distribution that do not impose the choice of a particular coordinate system onto our search space are the 'size' σ of the distribution, corresponding to the $(2d)$-th root of the determinant of the covariance matrix, and its orthogonal complement, the covariance matrix normalized to constant determinant $\mathbf{B}$ (see section 2.5.3). The first of these candidates results in a standard evolution strategy without covariance adaptation at all, which may indeed be a viable option in some applications, but is often too inflexible. The set of normalized covariance matrices, on the other hand, is not interesting because it is clear that the size of the distribution needs to be adjusted in order to ensure convergence to an optimum.

Thus, it has been proposed to give up some invariance properties of the search algorithm, and to adapt the class of search distribution with diagonal covariance matrices *in some predetermined coordinate system* (Ros and Hansen, 2008). Such a choice is justified in many applications where a certain degree of independence can be assumed among the fitness function parameters. It has even been shown by Ros and Hansen (2008) that this approach can work surprisingly well even for highly non-separable fitness functions.

Restricting the class of search distributions to Gaussians with diagonal covariance matrix corresponds to restricting a general class of multi-variate search distributions to separable distributions

$$p(\mathbf{z} \,|\, \theta) = \prod_{i=1}^{d} \tilde{p}(\mathbf{z}_i \,|\, \theta_i) \quad ,$$

where $\tilde{p}$ is a family of densities on the reals, and $\theta = (\theta_1, \ldots, \theta_d)$ collects the parameters of all of these distributions. In most cases these parameters amount to $\theta_i = (\boldsymbol{\mu}_i, \boldsymbol{\sigma}_i)$, where $\boldsymbol{\mu}_i \in \mathbb{R}$ is a position and $\boldsymbol{\sigma}_i \in \mathbb{R}^+$ is a scale parameter (i.e., mean and standard deviation, if they exist), such that $\mathbf{z}_i = \boldsymbol{\mu}_i + \boldsymbol{\sigma}_i \cdot \mathbf{s}_i \sim \tilde{p}(\cdot \,|\, \boldsymbol{\mu}_i, \boldsymbol{\sigma}_i)$ for $\mathbf{s}_i \sim \tilde{p}(\cdot \,|\, 0, 1)$.

Obviously, this allows us to sample new offspring in $\mathcal{O}(d)$ time (per sample). Since the adaptation of each component's parameters is independent, the strategy update step also takes only $\mathcal{O}(d)$ time.

Thus, the sacrifice of invariance, amounting to the selection of a distinguished coordinate system, allows for a linear time algorithm (per individual), which still maintains a reasonable amount of flexibility in the search distribution, and allows for a considerably faster adaptation of its parameters. The resulting NES variant, called *separable* NES (SNES), is illustrated in algorithm 8 for Gaussian search distributions. Note that each of the steps requires only $\mathcal{O}(d)$ operations. Later in this section we extend this algorithm to other search distributions, without affecting its computational complexity.

Algorithm 8: Separable NES (SNES)

```
   input: $f$, $\boldsymbol{\mu}_{init}$, $\boldsymbol{\sigma}_{init}$
1  repeat
2      for $k = 1 \ldots \lambda$ do
3          draw sample $\mathbf{s}_k \sim \mathcal{N}(0, \mathbb{I})$
4          $\mathbf{z}_k \leftarrow \boldsymbol{\mu} + \boldsymbol{\sigma}\mathbf{s}_k$
5          evaluate the fitness $f(\mathbf{z}_k)$
6      end
7      sort $\{(\mathbf{s}_k, \mathbf{z}_k)\}$ with respect to $f(\mathbf{z}_k)$ and compute utilities $u_k$
8      compute gradients  $\nabla_{\boldsymbol{\mu}} J \leftarrow \sum_{k=1}^{\lambda} u_k \cdot \mathbf{s}_k$
                          $\nabla_{\boldsymbol{\sigma}} J \leftarrow \sum_{k=1}^{\lambda} u_k \cdot (\mathbf{s}_k^2 - 1)$
9      update parameters  $\boldsymbol{\mu} \leftarrow \boldsymbol{\mu} + \eta_{\boldsymbol{\mu}} \cdot \boldsymbol{\sigma} \cdot \nabla_{\boldsymbol{\mu}} J$
                          $\boldsymbol{\sigma} \leftarrow \boldsymbol{\sigma} \cdot \exp(\eta_{\boldsymbol{\sigma}}/2 \cdot \nabla_{\boldsymbol{\sigma}} J)$
10 until stopping criterion is met;
```

2.6.2 Rotationally-symmetric Distributions

The class of *radial* or rotationally-symmetric search distributions are distributions with the property $p(x) = p(Ux)$, for all $x \in \mathbb{R}^d$ and all orthogonal matrices $U \in \mathbb{R}^{d \times d}$. Let $Q_{\boldsymbol{\tau}}(\mathbf{z})$ be a family of densities of a rotationally symmetric probability distributions in $\mathbb{R}^d$ with parameter $\boldsymbol{\tau}$. Thus, we can write $Q_{\boldsymbol{\tau}}(\mathbf{z}) = q_{\boldsymbol{\tau}}(r^2)$ with $r^2 = \|\mathbf{z}\|^2$ for some family of functions $q_{\boldsymbol{\tau}} : \mathbb{R}^{\geq 0} \to \mathbb{R}^{\geq 0}$. In the following we consider classes of search distributions with densities

$$\pi\big(\mathbf{z} \,\big|\, \boldsymbol{\mu}, \mathbf{A}, \boldsymbol{\tau}\big) = \frac{1}{\det(\mathbf{A})} \cdot q_{\boldsymbol{\tau}}\big(\| \left(\mathbf{A}^{-1}\right)^{\top} (\mathbf{z} - \boldsymbol{\mu})\|^2\big)$$

with additional transformation parameters $\boldsymbol{\mu} \in \mathbb{R}^d$ and invertible $\mathbf{A} \in \mathbb{R}^{d \times d}$. The function $q_{\boldsymbol{\tau}}$ is the accordingly transformed density of the variable $\mathbf{s} = \left(\mathbf{A}^{-1}\right)^{\top} (\mathbf{z} - \boldsymbol{\mu})$. This setting is rather general. It covers many important families of distributions and their multi-variate forms, such as multi-variate Gaussians. In addition, parameters of the radial distribution, most prominently its tail (controlling whether large mutations are common or rare) can be controlled with the parameter $\boldsymbol{\tau}$.

We apply the procedure presented in section 2.5.2 to this more general case. In local exponential coordinates

$$(\boldsymbol{\delta}, \mathbf{M}) \mapsto (\boldsymbol{\mu}_{\text{new}}, \mathbf{A}_{\text{new}}) = \left(\boldsymbol{\mu} + \mathbf{A}^{\top}\boldsymbol{\delta}, \mathbf{A} \exp\left(\frac{1}{2}\mathbf{M}\right)\right)$$

we obtain the three components of log-derivatives

$$\nabla_{\boldsymbol{\delta}, \mathbf{M}, \boldsymbol{\tau}}\big|_{\boldsymbol{\delta}=0, \mathbf{M}=0} \log \pi\left(\mathbf{z} \,|\, \boldsymbol{\mu}, \mathbf{A}, \boldsymbol{\tau}, \boldsymbol{\delta}, \mathbf{M}\right) = \left(g_{\boldsymbol{\delta}}, g_{\mathbf{M}}, g_{\boldsymbol{\tau}}\right),$$

$$\begin{aligned} g_{\boldsymbol{\delta}} &= -2 \cdot \frac{q'_{\boldsymbol{\tau}}(\|\mathbf{s}\|^2)}{q_{\boldsymbol{\tau}}(\|\mathbf{s}\|^2)} \cdot \mathbf{s} \\ g_{\mathbf{M}} &= -\frac{1}{2}\mathbb{I} - \frac{q'_{\boldsymbol{\tau}}(\|\mathbf{s}\|^2)}{q_{\boldsymbol{\tau}}(\|\mathbf{s}\|^2)} \cdot \mathbf{s}\mathbf{s}^\top \\ g_{\boldsymbol{\tau}} &= \frac{1}{q_{\boldsymbol{\tau}}(\|\mathbf{s}\|^2)} \cdot \nabla_{\boldsymbol{\tau}}\, q_{\boldsymbol{\tau}}(\|\mathbf{s}\|^2) \end{aligned}$$

where $q'_{\boldsymbol{\tau}} = \frac{\partial}{\partial(r^2)} q_{\boldsymbol{\tau}}$ denotes the derivative of $q_{\boldsymbol{\tau}}$ with respect to r^2, and $\nabla_{\boldsymbol{\tau}}\, q_{\boldsymbol{\tau}}$ denotes the gradient w.r.t. $\boldsymbol{\tau}$.

In the special case of Gaussian search distributions, q does not depend on a parameter $\boldsymbol{\tau}$ and we have

$$\begin{aligned} q(r^2) &= \frac{1}{(2\pi)^{d/2}} \cdot \exp\left(-\frac{1}{2} r^2\right) \\ q'(r^2) &= -\frac{1}{2} \cdot \frac{1}{(2\pi)^{d/2}} \cdot \exp\left(-\frac{1}{2} r^2\right) = -\frac{1}{2} \cdot q(r^2) \ , \end{aligned}$$

resulting in $g_{\boldsymbol{\delta}} = \mathbf{s}$ and $g_{\mathbf{M}} = \frac{1}{2}(\mathbf{s}\mathbf{s}^\top - \mathbb{I})$, recovering equations (2.12) and (2.13).

Sampling from Radial Distributions

In order to use this class of distributions for search we need to be able to draw samples from it. The central idea is to first draw a sample $\mathbf{s}$ from the 'standard' density $\pi(\mathbf{s} \,|\, \boldsymbol{\mu} = 0, \mathbf{A} = \mathbb{I}, \boldsymbol{\tau})$, which is then transformed into the sample $\mathbf{z} = \mathbf{A}^\top \mathbf{s} + \boldsymbol{\mu}$, corresponding to the density $\pi(\mathbf{z} \,|\, \mathbf{A}, \boldsymbol{\mu}, \boldsymbol{\tau})$. In general, sampling $\mathbf{s}$ can be decomposed into sampling the (squared) radius component $r^2 = \|\mathbf{z}\|^2$ and a unit vector $\mathbf{v} \in \mathbb{R}^d$, $\|\mathbf{v}\| = 1$. The squared radius has the density

$$\tilde{q}_{\boldsymbol{\tau}}(r^2) = \int_{\|\mathbf{z}\|^2 = r^2} Q_{\boldsymbol{\tau}}(\mathbf{z})\, d\mathbf{z} = \frac{2\pi^{d/2}}{\Gamma(d/2)} \cdot (r^2)^{(d-1)/2} \cdot q_{\boldsymbol{\tau}}(r^2) \ ,$$

where $\Gamma(\cdot)$ denotes the gamma function. In the following we assume that we have an efficient method of drawing samples from this one-dimensional density. Besides the radius we draw a unit vector $\mathbf{v} \in \mathbb{R}^d$ uniformly at random, for example by normalizing a standard normally distributed vector. Then $\mathbf{s} = r \cdot \mathbf{v}$ is effectively sampled from $\pi(\mathbf{s} \,|\, \boldsymbol{\mu} = 0, \mathbf{A} = \mathbb{I}, \boldsymbol{\tau})$, and the composition $\mathbf{z} = r\mathbf{A}^\top \mathbf{v} + \boldsymbol{\mu}$ follows the density $\pi(\mathbf{z} \,|\, \boldsymbol{\mu}, \mathbf{A}, \boldsymbol{\tau})$. In many special cases, however, there are more efficient ways of sampling $\mathbf{s} = r \cdot \mathbf{v}$ directly.

Computing the Fisher Information Matrix

In section 2.5.2 the natural gradient coincides with the plain gradient, because coordinate system is constructed in such a way that the Fisher information matrix for the parameters $(\boldsymbol{\delta}, \mathbf{M})$ is the identity. However, this is in general not possible in the presence of parameters $\boldsymbol{\tau}$, typically controlling the radial shape, and in particular the tail, of the search distribution.

Consider the case $\boldsymbol{\tau} \in \mathbb{R}^{d'}$. The dimensions of $\boldsymbol{\delta}$ and $\mathbf{M}$ are d and $d(d+1)/2$, respectively, making for a total number of $m = d(d+3)/2+d'$ parameters. Thus, the Fisher information matrix is an $(m \times m)$ matrix of the form

$$\mathbf{F} = \begin{pmatrix} \mathbb{I} & v \\ v^\top & c \end{pmatrix}$$

with

$$v = \frac{\partial^2 \log \pi(\mathbf{z})}{\partial(\boldsymbol{\delta}, \mathbf{M}) \partial \boldsymbol{\tau}} \in \mathbb{R}^{(m-d') \times d'}, \quad c = \frac{\partial^2 \log \pi(\mathbf{z})}{\partial \boldsymbol{\tau}^2} \in \mathbb{R}^{d' \times d'}.$$

Using the Woodbury identity, we compute the inverse of the Fisher matrix as

$$\mathbf{F}^{-1} = \begin{pmatrix} \mathbb{I} & v \\ v^\top & c \end{pmatrix}^{-1} = \begin{pmatrix} \mathbb{I} + Hvv^\top & -Hv \\ -Hv^\top & H \end{pmatrix}$$

with $H = (c - v^\top v)^{-1}$, and exploiting $H^\top = H$. The natural gradient becomes

$$\mathbf{F}^{-1} \cdot g = \begin{pmatrix} (g_{\boldsymbol{\delta}}, g_{\mathbf{M}}) - Hv(v^\top (g_{\boldsymbol{\delta}}, g_{\mathbf{M}}) - g_{\boldsymbol{\tau}}) \\ H(v^\top (g_{\boldsymbol{\delta}}, g_{\mathbf{M}}) - g_{\boldsymbol{\tau}}) \end{pmatrix} ,$$

which can be computed efficiently in only $\mathcal{O}(d'^3 + md')$ operations. Assuming that d' does not grow with d, this complexity corresponds to $\mathcal{O}(d^2)$ operations in terms of the search space dimension, compared to $\mathcal{O}(d^6)$ operations required for a naive inversion of the full Fisher matrix. In other words, the benefits of the natural coordinate system carry over, even if we no longer have $\mathbf{F} = \mathbb{I}$.

2.6.3 Heavy-tailed NES

Natural gradient ascent and plain gradient ascent in natural coordinates provide two equivalent views on the working principle of NES. In this section we introduce yet another interpretation, with the goal of extending NES to heavy-tailed distributions, in particular distributions with infinite variance, like the Cauchy distribution. The problem posed by these distributions within the NES framework is that they do not induce a Riemannian structure on the parameter space of the distribution via their Fisher information, which renders the information geometric interpretation of natural coordinates and natural gradient ascent invalid.

Many important types of search distributions have strong *invariance* properties, e.g., multi-variate and heavy-tailed distributions. In this still very general case, the NES principle can be derived solely based on invariance properties, without ever referring to information geometric concepts.

The direction of the gradient $\nabla_\theta J(\theta)$ depends on the inner product $\langle \cdot, \cdot \rangle$ in use, corresponding to the choice of a coordinate system or an (orthonormal) set of basis vectors. Thus, expressing a gradient ascent algorithm in arbitrary coordinates results in (to some extent) arbitrary and often sub-optimal updates. NES resolves this dilemma by relying on the natural gradient, which corresponds to the distinguished coordinate system (of the tangent space of the family of search distributions) corresponding to the Fisher information metric.

The natural coordinates of a multi-variate Gaussian search distribution turn out to be those local coordinates w.r.t. which the current search distribution has zero mean and unit covariance. This coincides with the coordinate system in which the invariance properties of multi-variate Gaussians are most apparent. This connection turns out to be quite general. In the following we exploit this property systematically and apply it to distributions with infinite (undefined) Fisher information.

Groups of Invariances

The invariances of a search distribution can be expressed by a group $\mathcal{G}$ of (affine) linear transformations. Typically, $\mathcal{G}$ is a sub-group of the group of orthogonal transformations (i.e., rotations) w.r.t. a local coordinate system. For the above example of a rotationally-symmetric density $Q : \mathbb{R}^d \to \mathbb{R}_0^+$ (e.g., a Gaussian), the densities

$$\pi(\mathbf{z} \,|\, \boldsymbol{\mu}, \mathbf{A}) = \frac{1}{\det(\mathbf{A})} \cdot Q\left(\mathbf{A}^{-1}(\mathbf{z} - \boldsymbol{\mu})\right)$$

with $\boldsymbol{\mu} \in \mathbb{R}^d$ and $\mathbf{A} \in \mathbb{R}^{d\times d}$, $\det(\mathbf{A}) \neq 0$ form the corresponding multi-variate distribution. Let $\mathcal{G}_{(\boldsymbol{\mu},\mathbf{A})}$ be the group of invariances of $\pi(\mathbf{z} \,|\, \boldsymbol{\mu}, \mathbf{A})$, that is, $\mathcal{G}_{(\boldsymbol{\mu},\mathbf{A})} = \left\{ g \,\middle|\, \pi(g(\mathbf{z}) \,|\, \boldsymbol{\mu}, \mathbf{A}) = \pi(\mathbf{z} \,|\, \boldsymbol{\mu}, \mathbf{A}) \,\forall \mathbf{z} \in \mathbb{R}^d \right\}$. We have $\mathcal{G}_{(0,\mathbb{I})} = \mathbb{O}_{\langle\cdot,\cdot\rangle}(\mathbb{R}^d) = \left\{ g \,\middle|\, \langle g(\mathbf{z}), g(\mathbf{z}')\rangle = \langle \mathbf{z}, \mathbf{z}'\rangle \,\forall \mathbf{z}, \mathbf{z}' \in \mathbb{R}^d \right\}$, where the right hand side is the group of orthogonal transformations w.r.t. an inner product, defined as the (affine) linear transformations that leave the inner product (and thus the properties induced by the orthonormal coordinate system) invariant. Here the inner product is the one w.r.t. which the density Q is rotation invariant. For general $(\boldsymbol{\mu}, \mathbf{A})$ we have $\mathcal{G}_{(\boldsymbol{\mu},\mathbf{A})} = h \circ \mathcal{G}_{(0,\mathbb{I})} \circ h^{-1}$, where $h(\mathbf{z}) = \mathbf{A}\mathbf{z} + \boldsymbol{\mu}$ is the affine linear transformation corresponding to the current search distribution. In general, the group of invariances is only a subgroup of an orthogonal group, e.g., for a separable distribution Q, $\mathcal{G}$ is the finite group generated by coordinate permutations and axis flips.

We argue that it is most natural to rely on a gradient or coordinate system which is *compatible* with the invariance properties of the search distribution in use. In other words, we should ensure the compatibility condition

$$\mathcal{G}_{(\boldsymbol{\mu},\mathbf{A})} \subset \mathbb{O}_{\langle\cdot,\cdot\rangle}(\mathbb{R}^d)$$

for the inner product $\langle\cdot,\cdot\rangle$ with respect to which we compute the gradient ∇J. This condition has a straight-forward connection to the natural coordinate system introduced in section 2.5.2: It is fulfilled by performing all updates in local coordinates, in which the current search distribution is expressed by the density $\pi(\cdot \,|\, 0, \mathbb{I}) = Q(\cdot)$. In these coordinates, the distribution is already rotationally symmetric by construction (or similar for separable distributions), where the rotational symmetry is defined in terms of the 'standard' inner product of the local coordinates. Local coordinates save us from the cumbersome explicit construction of an inner product that is left invariant by the group $\mathcal{G}_{(\boldsymbol{\mu},\mathbf{A})}$.

Note, however, that $Q(\mathbf{z})$ and $Q(\sigma \cdot \mathbf{z})$ have the same invariance properties. Thus, the invariance properties make only the gradient components $\nabla_{\boldsymbol{\mu}} J$ and $\nabla_{\mathbf{B}} J$ unique, but not the scale component $\nabla_\sigma J$. Luckily this does not affect the (1+1) hill-climber variant of NES, which relies on a success-based step size adaptation rule (see section 2.5.5). Also note that this derivation of the NES updates works only for families of search distributions with strong invariance properties, while natural gradient ascent extends to much more general distributions, such as mixtures of Gaussians.

Cauchy Distribution

Given these results, NES, formulated in local coordinates, can be used with heavy-tailed search distributions without modification. This applies in particular to the (1+1) hill-climber, which is the most attractive choice for heavy-tailed search distributions, because when the search distribution converges to a local optimum and a better optimum is located by a mutation, then averaging this step over the offspring population will usually result in a sub-optimal step that stays within the same basin of attraction. In contrast, a hill-climber can jump straight into the better basin of attraction, and can thus make better use the specific advantages of heavy-tailed search distributions.

Of course, the computation of the plain gradient changes depending on the distribution in use. Once this gradient is computed in the local coordinate system respecting the invariances of the current search distribution, it can be used for updating the search parameters $\boldsymbol{\mu}$ and $\mathbf{B}$ without further corrections like multiplying with the (in general undefined) inverse Fisher matrix. For the multi-variate Cauchy distribution we have

$$q(\mathbf{s}) = \frac{\Gamma((d+1)/2)}{\pi^{(d+1)/2}} \cdot (\|\mathbf{s}\|^2 + 1)^{-(d+1)/2} \quad ,$$

which results in the gradient components

$$\begin{aligned} \nabla_{\boldsymbol{\delta}} J =& \frac{d+1}{\|\mathbf{s}\|^2 + 1} \cdot \mathbf{s} \\ \nabla_{\mathbf{M}} J =& \frac{d+1}{2 \cdot (\|\mathbf{s}\|^2 + 1)} \cdot \mathbf{s}\mathbf{s}^\top - \frac{1}{2} \cdot \mathbb{I} \quad . \end{aligned}$$

The full NES hill-climber with multi-variate Cauchy mutations is provided in algorithm 9.

Algorithm 9: (1+1)-NES with multi-variate Cauchy distribution

```
   input: $f$, $\boldsymbol{\mu}_{init}$, $\boldsymbol{\Sigma}_{init} = \mathbf{A}^\top\mathbf{A}$
1  $f_{\text{best}} \leftarrow -\infty$
2  repeat
                                $\mathbf{s} \sim \mathcal{N}(0, \mathbb{I})$
3      draw sample              $r \sim \pi_{Cauchy}(0, 1)$
                                $\mathbf{z} \leftarrow r\mathbf{A}^\top\mathbf{s} + \boldsymbol{\mu}$
4      evaluate the fitness $f(\mathbf{z})$
5      $(\mathbf{s}_1, \mathbf{s}_2) \leftarrow (0, \mathbf{s})$
6      $(\mathbf{z}_1, \mathbf{z}_2) \leftarrow (\boldsymbol{\mu}, \mathbf{z})$
7      if $f(\mathbf{z}) > f_{best}$ then
8          update mean $\boldsymbol{\mu} \leftarrow \mathbf{z}$
9          $f_{\text{best}} \leftarrow f(\mathbf{z})$
10         $\mathbf{u} \leftarrow (-4, 1)$
11     else
12         $\mathbf{u} \leftarrow (\frac{4}{5}, 0)$
13     end
14     calculate log-derivatives $\nabla_{\mathbf{M}} \log \pi(\mathbf{z}_k|\theta) \leftarrow \frac{1}{2}\left(\frac{d+1}{r^2+1}\mathbf{s}_k\mathbf{s}_k^\top - \mathbb{I}\right)$
15     $\nabla_{\mathbf{M}} J \leftarrow \frac{1}{2}\sum_{k=1}^{2} \nabla_{\mathbf{M}} \log \pi(\mathbf{z}_k|\theta) \cdot u_k$
16     $\mathbf{A} \leftarrow \mathbf{A} \cdot \exp\left(\frac{1}{2}\eta_{\mathbf{A}} \nabla_{\mathbf{M}} J\right)$
17 until stopping criterion is met;
```

2.7 Experiments

In this section, we empirically validate the new algorithms, with the goal of answering the following questions:

- How do NES algorithms perform compared to state-of-the-art evolution strategies?
- Can we identify specific strengths and limitations of the different variants, such as SNES (designed for separable problems) and Cauchy-NES (with heavy-tailed distribution)?
- Going beyond standardized benchmarks, should natural evolution strategies be applied to real-world problems?

We conduct a broad series of experiments on standard benchmarks, as well as more specific experiments testing special capabilities. In total, six different algorithm variants are tested and their behaviors compared qualitatively as well as quantitatively, w.r.t. different modalities.

We start by detailing and justifying the choices of hyperparameters, then we proceed to evaluate the performance of a number of different variants of NES (with and without the importance mixing and adaptation sampling techniques) on a broad collection of benchmarks. We further conduct experiments using the separable variant on high-dimensional problems, and address the question of global optimization and to what degree a heavy-tailed search distribution (namely multivariate Cauchy) can alleviate the problem of getting stuck in local optima.

2.7.1 Experimental Setup and Hyperparameters

Across all NES variants, we distinguish three hyperparameters: the population size λ, the learning rates η and the utility function u (because we always use fitness shaping, see section 2.4.1). In particular, for the multivariate Gaussian case (xNES) we have the three learning rates $\eta_{\boldsymbol{\mu}}$, η_σ, and $\eta_{\mathbf{B}}$.

It is highly desirable to have good default settings that scale with the problem dimension and lead to robust performance on a broad class of benchmark functions. Table 2.1 provides such default values as functions of the problem dimension d for xNES. We borrowed several of the settings from CMA-ES (Hansen and Ostermeier, 2001), which seems natural due to the apparent similarity discussed in section 2.5.4. Both the population size λ and the learning rate $\eta_{\boldsymbol{\mu}}$ are the same as for CMA-ES, even if this learning rate never explicitly appears in CMA-ES. For the utility function we copied the weighting scheme of CMA-ES, but we shifted the values such that they sum to zero, which is the simplest form of implementing a fitness baseline; Jastrebski and Arnold (2006) proposed a similar approach for CMA-ES. The remaining parameters were determined via an empirical investigation, aiming for robust performance. In addition, in the separable case (SNES) the number of parameters in the covariance matrix is reduced from $d(d+1)/2 \in \mathcal{O}(d^2)$ to $d \in \mathcal{O}(d)$, which allows us to increase the learning rate $\eta_{\boldsymbol{\sigma}}$ by a factor of $d/3 \in \mathcal{O}(d)$, a choice which has proven robust in practice (Ros and Hansen, 2008).

Table 2.1: Default parameter values for xNES and SNES (including the utility function) as a function of problem dimension d.

parameter	default value
λ	$4 + \lfloor 3\log(d) \rfloor$
$\eta_{\boldsymbol{\mu}}$	1
$\eta_\sigma = \eta_{\mathbf{B}}$	$\dfrac{3(3+\log(d))}{5d\sqrt{d}}$
$\eta_{\boldsymbol{\sigma}}$	$\dfrac{3+\log(d)}{5\sqrt{d}}$
u_k	$\dfrac{\max\left(0, \log(\frac{\lambda}{2}+1) - \log(i)\right)}{\sum_{j=1}^{\lambda} \max\left(0, \log(\frac{\lambda}{2}+1) - \log(j)\right)} - \dfrac{1}{\lambda}$

The six algorithm variants that we will be evaluating below are xNES (algorithm 5), its hill-climber variant (1+1)-xNES (see algorithm 7), “xNES-im-as”, which is xNES using both importance mixing (section 2.4.3) and adaptation sampling (section 2.4.4), the separable SNES (as in algorithm 8), its own hill-climber variant (1+1)-SNES (pseudocode not shown), and finally the heavy-

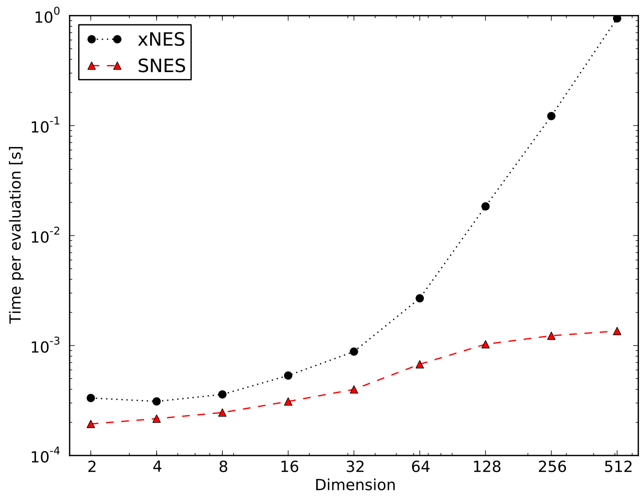


Figure 2.7: Computation time per function evaluation, for two representative algorithms, on problem dimensions ranging from 2 to 512. SNES scales linearly, whereas the cost grows cubically for xNES.

tailed variant (1+1)-NES-Cauchy (as in algorithm 9).

Figure 2.7 shows a comparison plot for the runtime of the two groups of algorithms, xNES with cubic runtime and the linear-cost SNES[5]. Note that the computation time is given *per function evaluation*. A Python implementation of all these is available within the open-source machine learning library PyBrain (see appendix A).

2.7.2 Black-box Optimization Benchmarks

For a practitioner it is important to understand how NES algorithms compare to other methods on a wide range black-box optimization scenarios. Thus, we evaluate our algorithm on all the benchmark functions of the 'Black-Box Optimization Benchmarking' collection (BBOB) from the 2010 GECCO Workshop for Real-Parameter Optimization. The collection consists of 24 noise-free functions (12 unimodal, 12 multi-modal; Hansen and Finck, 2010a) and 30 noisy functions (Hansen and Finck, 2010b). In order to make our results fully comparable, we also use the identical setup (Hansen and Auger, 2010), which transforms the pure benchmark functions to make the parameters non-separable (for some) and avoid trivial optima at the origin. To facilitate the comparison for the reader without overcrowding the plots, we provide the GECCO 2010 results of (1,4)-CMA-ES alongside our own (chosen as the most comparable representative among the 13 CMA-ES-based submissions to the workshop).

On all of these benchmarks, we compare xNES (as described in Algorithm 5) and xNES-im-as, that is, the same algorithm but augmented with both impor-

[5]The reference machine has an Intel Core i7 processor with 1.6GHz and 4GB of RAM.

tance mixing and adaptation sampling. On the multi-modal, as well as the noisy benchmarks, we also use restart strategies (see section 2.4.5).

For the unimodal benchmarks (see Figure 2.8), we plot how the performance of xNES scales with problem dimension (between 2 and 40). Shown are the median number of evaluations required to reach a fitness of -10^{-7}. We find that xNES is on par with CMA-ES for most benchmarks, but generally more robust (e.g., on the StepEllipsoid benchmark). Employing importance mixing and adaptation sampling further increases performance (most significantly on the simple benchmarks like the Sphere function), but at the cost of robustness.

For the multi-modal benchmarks (see Figure 2.9), as well as the noisy benchmarks (see Figures 2.10 and 2.11), where not all runs succeed, we instead show the cumulative success rates (again, for reaching a fitness of -10^{-7}), for problem dimension $d = 5$. We find that xNES together with our restart strategies lead to very good performance, clearly outperforming CMA-ES on almost all multi-modal functions, and many noisy functions. For xNES-im-as, the results correspond to the expected trade-off that these techniques improve performance, with a substantial boost on the noise-free multi-modal functions, but reduce robustness, as is evident from the results on many of the harder noisy benchmarks (only attenuated in part by the restart strategy).

2.7.3 Separable NES

The SNES algorithm is expected to perform at least as well as xNES on separable problems, while it should show considerably worse performance in the presence of highly dependent variables. We first present a number of experiments on standard benchmarks with the aim of understanding its behavior and in particular its limitations in more detail. Furthermore, the algorithm is specifically designed to scale gracefully to high-dimensional search problems. We thus apply SNES to two tasks with a scalable and considerably high search space dimension, namely neuro-evolutionary controller design, and the problem of finding low energy states in Lennard-Jones potentials.

Separable and Non-separable Benchmarks

First, we evaluate SNES on a subset of the unimodal benchmark problems from the BBOB framework (Hansen and Auger, 2010; Hansen and Finck, 2010a). These benchmarks test the capability of SNES to descend quickly into local optima, a key property of most evolution strategies. The results in Figure 2.12 show how SNES dominates when the function is separable (f_1 through f_6), and converges much slower than xNES in non-separable benchmarks, as expected. In particular, on the rotated ellipsoid function f_{10}, which is designed to make separable methods fail, SNES requires 4 orders of magnitude more evaluations. In dimensions $d > 2$ it fails completely because the resolution of double precision numbers is insufficient for this task.

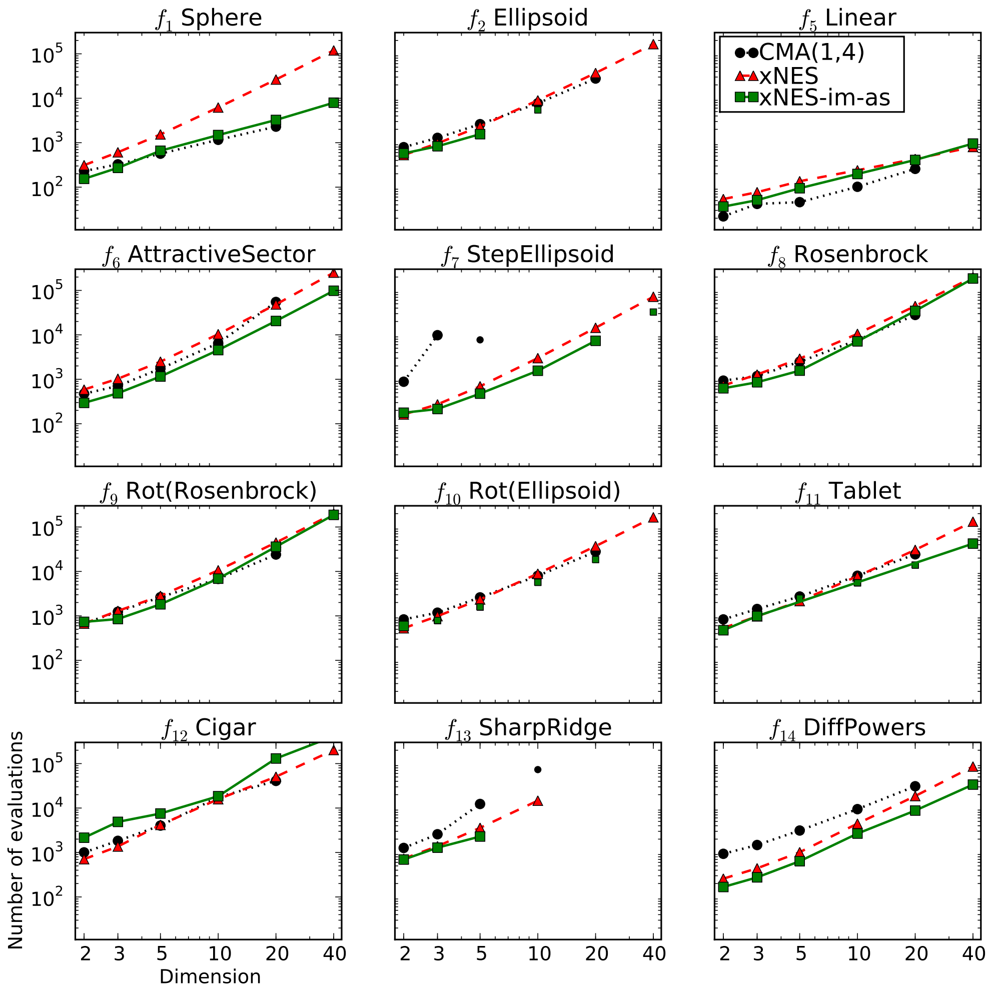


Figure 2.8: **Unimodal benchmarks.** Log-log plot of the median number of fitness evaluations (over 100 trials) required to reach the target fitness value of -10^{-7} for the 12 unimodal benchmark functions on dimensions 2 to 40. Disconnected (smaller) plot markers denote cases where the corresponding algorithm converged prematurely in at least 50% of the runs (cases for which 90% or more prematurely converged are not shown at all). No data was available for (1,4)-CMA-ES on dimension 40. Note that xNES (red triangles) consistently solves all benchmarks, with a scaling factor that is almost the same over all functions. Also, xNES appears to be more stable, especially on the SharpRidge function. When employing importance mixing and adaptation sampling (xNES-im-as, green squares), performance increases, most substantially on the Sphere function, while robustness decreases.

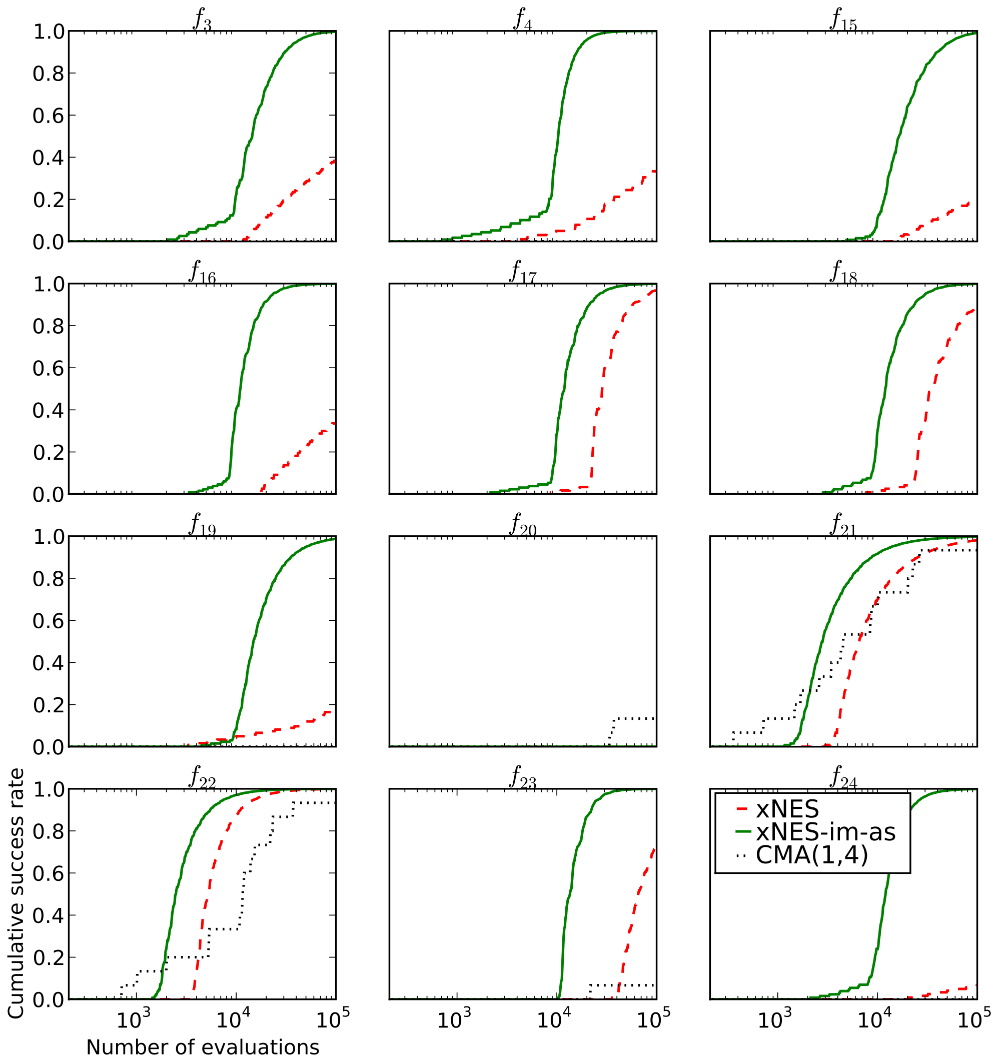


Figure 2.9: Multi-modal benchmarks, comparing the performance of xNES with restart strategies (dashed red) and (1,4)-CMA-ES (dotted black). Shown is the empirical cumulative success rate (over 100 runs, 15 for CMA-ES). Note that xNES clearly outperforms CMA-ES on all functions except f_{21} and f_{22} (both of which are simply combinations of 101 and 21 Gaussian peaks, respectively, without any global structure), where the results are very similar. When additionally employing importance mixing and adaptation sampling (xNES-im-as, solid green), performance improves even further.

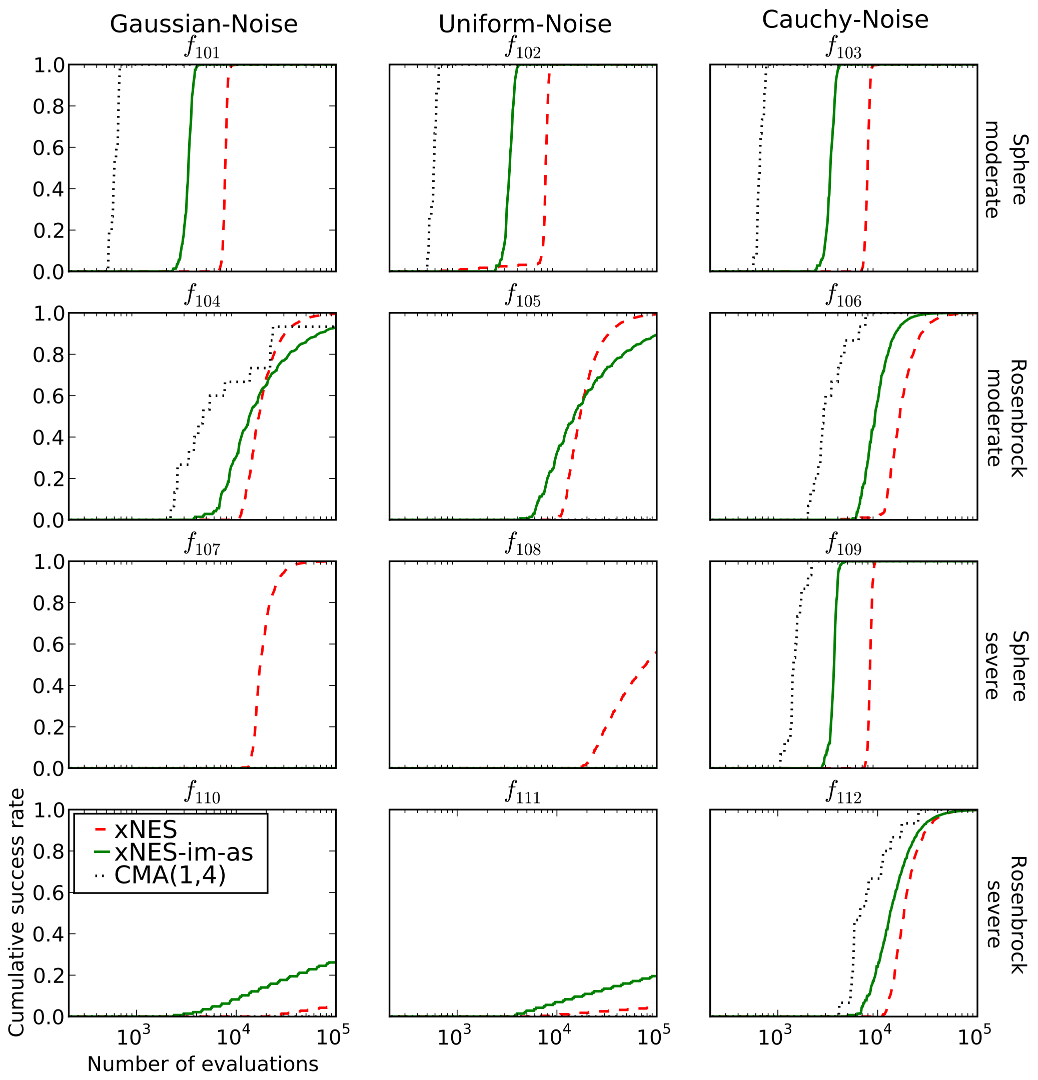


Figure 2.10: Noisy benchmarks, comparing the performance of xNES with restart strategies (dashed red), and (1,4)-CMA-ES (dotted black). Shown is the empirical cumulative success rate (over 100 runs, 15 for CMA-ES). The benchmarks are grouped by type of noise (vertical) and underlying function (horizontal). Generally speaking, Cauchy-noise is the least harmful, as the rare outliers do not affect the ranks all that much. We find that xNES outperforms CMA-ES on the majority of the functions. Additionally employing importance mixing and adaptation sampling (xNES-im-as, solid green), however, improves performance only on some of the functions. Continued in Figure 2.11.

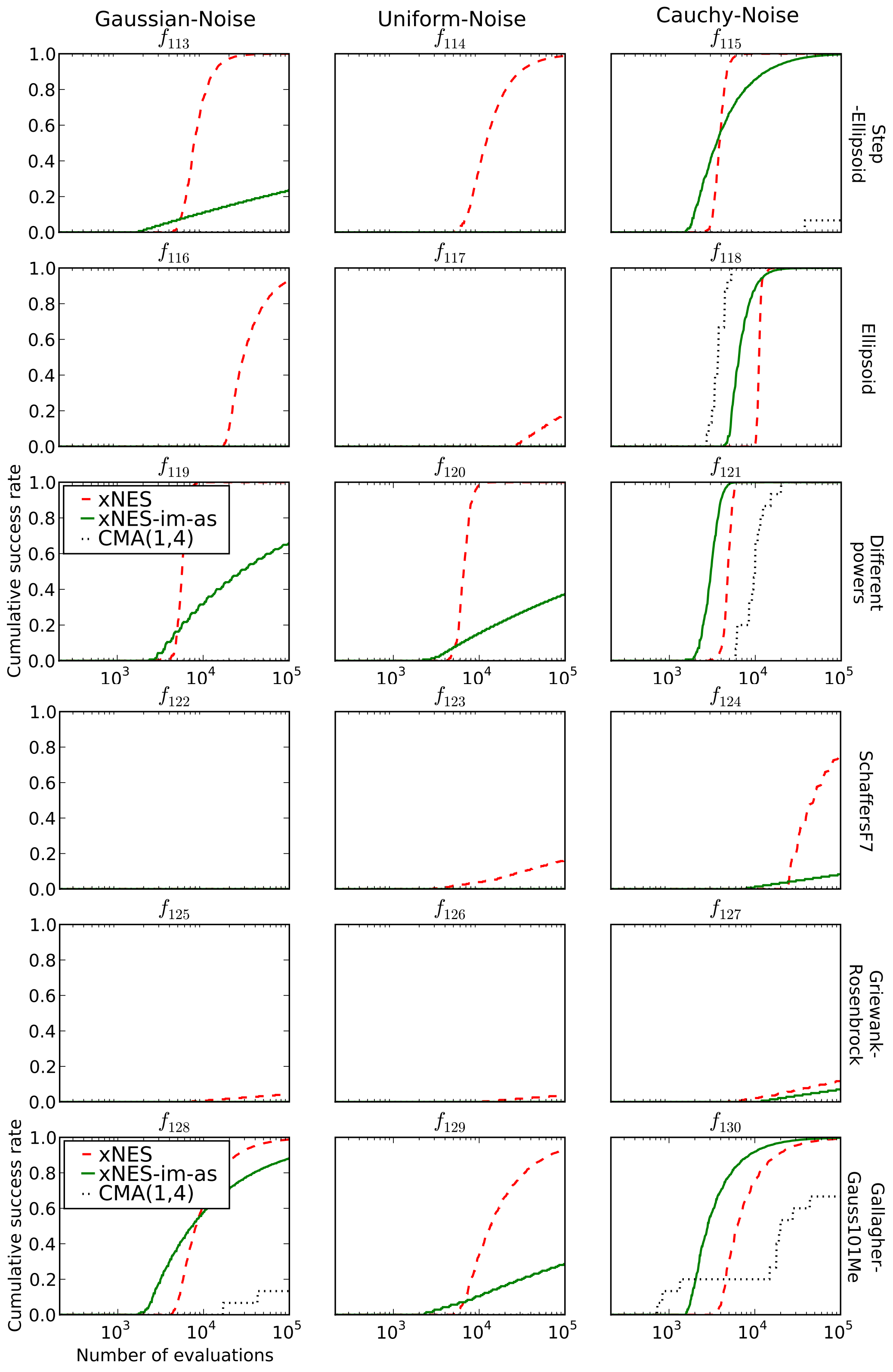


Figure 2.11: Noisy benchmarks (continued from Figure 2.10).

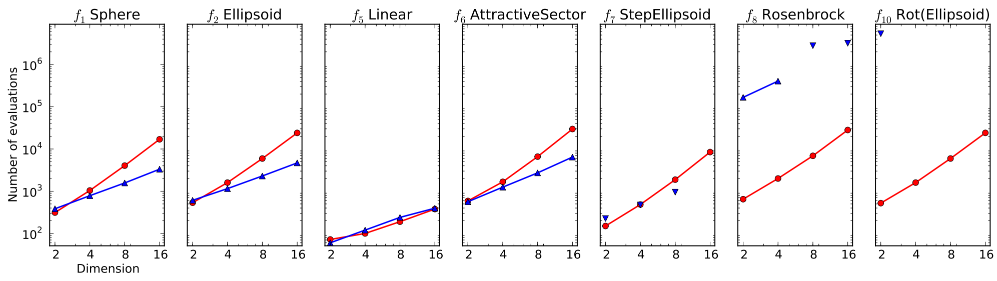


Figure 2.12: Comparison of the performance of xNES (red circles) and SNES (blue triangles) on a subset of the unimodal BBOB benchmark functions. The log-log plots show the median number of evaluations required to reach the target fitness 10^{-7}, for problem dimensions ranging from 2 to 16 (over 20 runs). The first 4 benchmark functions are separable, the other three are not. The inverted triangles indicate cases where SNES converged to the optimum in less than 90% of the runs.

Neuro-evolution

In the second experiment, we show how SNES is well-suited for neuro-evolution problems because they tend to be high-dimensional, multi-modal, but with highly redundant global optima (there is not a unique set of weights that defines the optimal behavior). In particular, we run it on Non-Markovian double-pole balancing, a task which involves balancing two differently sized poles hinged on a cart that moves on a finite track. The single control consists of the force F applied to the cart, and observations include the cart's position and the poles' angles, but no velocity information, which makes this task partially observable. It provides a perfect testbed for algorithms focusing on learning fine control with memory in continuous state and action spaces (Wieland, 1991). The controller is represented by a simple recurrent neural network, with three inputs, (position x and the two poles' angles β_1 and β_2), and a variable number n of tanh units in the output layer, which are fully connected (recurrently), resulting in a total of $n(n+3)$ weights to be optimized. The activation of the first of these recurrent neurons directly determines the force to be applied. We use the implementation found in PyBrain (see appendix A).

An evaluation is considered a success if the poles do not fall over for $100,000$ time-steps. We experimented with recurrent layers of sizes $n = 1$ to $n = 32$ (corresponding to between 4 and 1120 weights). It turns out that a single recurrent neuron is sufficient to solve the task (Figure 2.13, left). In fact, both the xNES and SNES results are state-of-the-art, outperforming the previously best algorithm (CoSyNE; Gomez et al., 2008, with a median of 410 evaluations) by a factor two.

In practical scenarios however, we cannot know the best network size a priori, and thus the prudent choice consists in overestimating the required size. An algorithm that graciously scales with problem dimension is therefore highly desirable, and we find (Figure 2.13, right) that SNES is exhibiting precisely that behavior. The fact that SNES outperforms xNES with increasing dimension,

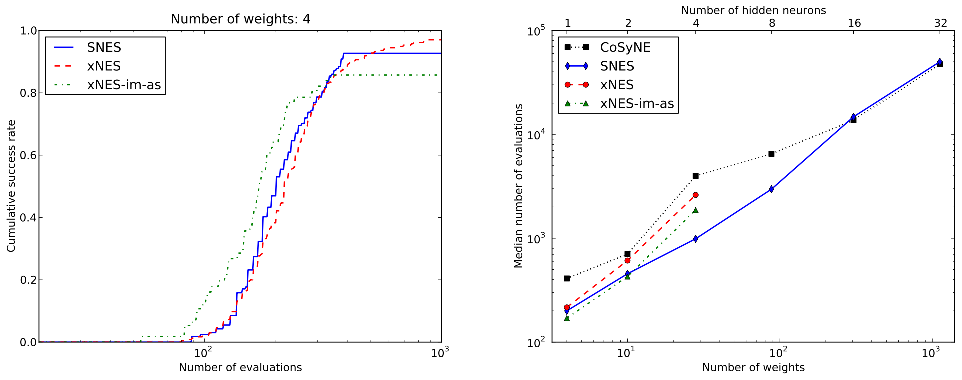


Figure 2.13: **Left:** Plotted are the cumulative success rates on the non-Markovian double-pole balancing task after a certain number of evaluations, empirically determined over 100 runs for each algorithm, using a single tanh-unit ($n = 1$) (i.e., optimizing 4 weights). We find that all three algorithm variants give state-of-the-art results, with a slightly faster but less robust performance for xNES with importance mixing and adaptation sampling. **Right:** Median number of evaluations required to solve the same task, but with increasing number of neurons (and corresponding number of weights). We limited the runtime to one hour per run, which explains why no results are available for xNES on higher dimensions (cubic time complexity). The fact that SNES quickly outperforms xNES, also in number of function evaluations, indicates that the benchmark is (sufficiently close to) separable, and it is unnecessary to use the full covariance matrix. For reference we also plot the corresponding results of the previously best performing algorithm CoSyNE (Gomez et al., 2008).

also in number of function evaluations, indicates that the benchmark is separable, and it is unnecessary to use the full covariance matrix. We conjecture that this a property shared with the majority of neuro-evolution problems that have enough weights to exhibit redundant global optima (some of which can be found without considering all parameter covariances).

Lennard-Jones Potentials

In our third benchmark, we show the performance of SNES on the widely studied problem of minimizing the Lennard-Jones atom cluster potentials, which is known for being extremely multi-modal (Wales and Doye, 1998). For that reason we employ the separable hill-climber variant (1+1)-SNES. The objective consists in finding that configuration of N atoms which minimizes the potential energy function

$$E_{LJ} \propto \sum_{i,j \leq N} \left[\left(\frac{1}{r_{ij}} \right)^{12} - \left(\frac{1}{r_{ij}} \right)^{6} \right] ,$$

where r_{ij} is the distance between atoms i and j (see also Figure 2.14 for an illustration). For the setup here, we initialized $\boldsymbol{\mu}$ near 0 and the step-sizes at $\boldsymbol{\sigma}_i = 0.01$ to avoid jumping into a local optimum in the fist generation. The results are plotted in Figure 2.15, showing how SNES scales convincingly to hundreds of parameters (each run up to $500d$ function evaluations).

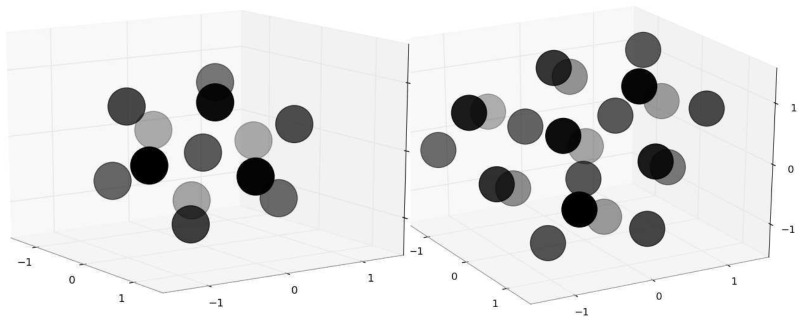


Figure 2.14: Illustration of the best configuration found for 13 atoms (symmetric, left), and 22 atoms (asymmetric, right).

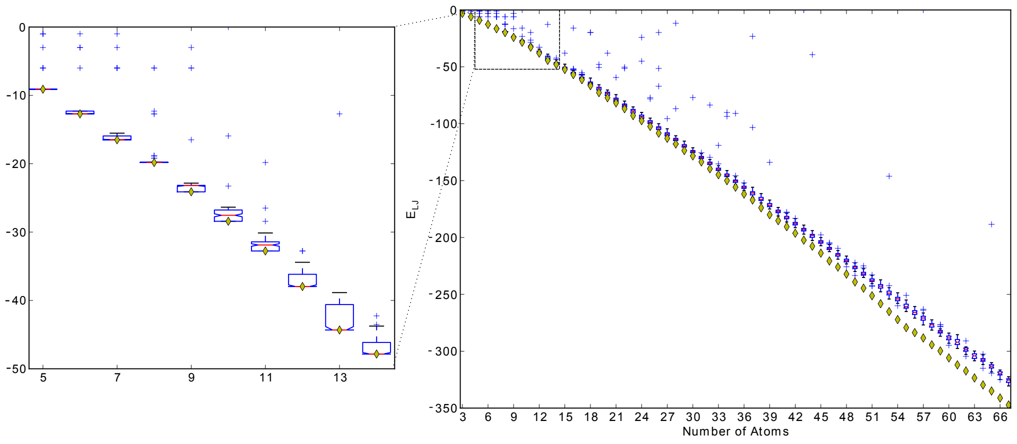


Figure 2.15: Performance of (1+1)-SNES on the Lennard-Jones benchmark for atom clusters ranging from 3 to 67 atoms (corresponding to problem dimensions d of 9 to 201). The yellow diamonds indicate the best known configurations (taken from (Wales and Doye, 1998)), and the box-plots show upper and lower quartile performance (the red line being the median) of SNES, over 100 runs. The inset is a zoom on the behavior in small dimensions, where SNES succeeds in locating the true optimum in a large fraction of the runs.

2.7.4 Heavy Tails and Global Optimization

Can NES algorithms with heavy-tailed distributions enhance the capability of escaping local optima? Our tests with the extremely heavy-tailed Cauchy distribution investigate the handling of multi-modality.

The first benchmark function is

$$f_{2Rosen}(\mathbf{z}) = \min\left\{ f_8(-\mathbf{z}-10), 5 + f_8\left(\frac{\mathbf{z}-10}{4}\right)\right\} \quad ,$$

where f_8 is the well-known Rosenbrock function (Hansen and Finck, 2010a), and the transformation $(\mathbf{z}-10)/4$ is component-wise. Our variant has a deceptive *double-funnel* structure, with a large valley containing a local optimum and a smaller but deeper valley containing the global optimum. The global structure will tend to guide the search towards the local optimum (see also Figure 2.16, left, for an illustration). For this experiment, the search distribution is initialized at mid-distance between the two optima, and the initial step-size σ is varied. Figure 2.16(right) shows the proportion of runs that converge to the global optimum, instead of the (easier to locate) local one, comparing for a multivariate Cauchy and Gaussian (1+1)-NES.

The second experiment uses the following 'random-basin' benchmark function:

$$\begin{aligned} f_{rb}(\mathbf{z}) = 1 - &\frac{9}{10} r\left(\left\lfloor\frac{\mathbf{z}_1}{10}\right\rfloor, \ldots, \left\lfloor\frac{\mathbf{z}_d}{10}\right\rfloor\right) \\ &- \frac{1}{10} r(\lfloor\mathbf{z}_1\rfloor, \ldots, \lfloor\mathbf{z}_d\rfloor) \cdot \prod_{i=1}^{d} \sin^2(\pi\mathbf{z}_i)^{\frac{1}{20d}} \end{aligned}$$

to investigate the degree to which a heavy-tailed distribution can be useful when the objective function is highly multi-modal, but there is no global structure to exploit. Here $r : \mathbb{Z}^d \to [0,1]$ is a pseudo-random number generator, which approximates an i.i.d. uniformly random distribution for each tuple of integers, while still being deterministic, i.e., each tuple evaluates to the same value each time. In practice, we implement it as a Mersenne twister (Matsumoto and Nishimura, 1998), seeded with the hash-value of the integers. Further, to avoid axis-alignment, we rotate the function by multiplying with an orthonormal random $d \times d$ matrix.

One interesting property of this function is that each unit-sized hypercube is an "attractor" of a local optimum. Thus, while sampling points from one hypercube, an ES will contract its search distribution, making it harder to escape from that local optimum. Furthermore, the values of the local optima are uniformly distributed in $[0,1]$, and do not provide a systematic global trend (in contrast to the Rastrigin function). If the optimization results in a value of, say, 0.11, then we know that only 11% of the local optima are better than this.

Figure 2.17 shows the results: not surprisingly, employing the Cauchy distribution for search permits longer jumps, and thus enables the algorithm to find

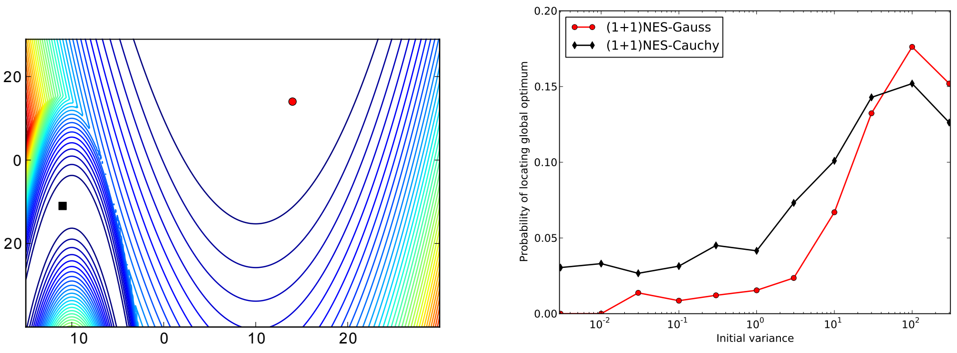


Figure 2.16: **Left:** Contour plot of the 2-dimensional Double-Rosenbrock function f_{2Rosen}, illustrating its deceptive double-funnel structure. The global structure leads the search to the local optimum ((14,14), red circle), whereas the true optimum ((-11,-11), black square) is located in the smaller valley. **Right:** Empirical success probabilities (of locating the global optimum), evaluated over 200 runs, of 1000 function evaluations each, on the same benchmark, while varying the size of the initial search distribution. The results clearly show the robustness of using a heavy-tailed distribution.

better local optima on average. The Cauchy version outperforms the Gaussian version by a factor of two to three, depending on the problem dimension. Note that the improvement (for both distributions) is due the number of neighbor-cubes increasing exponentially with dimension, thus increasing the chance that a relatively small jump will reach a better local optimum. At the same time, the adaptation of the step size is slowed down by the dimension-dependency of the learning rate, which leaves the algorithm more time to explore before it eventually converges into one of the local optima.

2.7.5 Results Summary

Our results have a number of implications. First of all, the results on the BBOB benchmarks show that NES algorithms are competitive with the state-of-the-art across a wide variety of black-box optimization problems.

Beyond this very general statement, we have demonstrated advantages and limitations of specific variants, and as such established the generality and flexibility of the NES framework. Experiments with heavy-tailed and separable distributions demonstrate the viability of the approach on high-dimensional and complex, deceptively multi-modal domains. We obtained best reported results on the difficult task of training a neural controller for double pole-balancing. This test, together with good results on the Lennard-Jones problem, show the feasibility of the algorithm for real-world search and optimization problems.

The multi-start strategy, although simple and non-adaptive in spirit, brings considerable improvements on many difficult multi-modal benchmarks. Its technique of interleaving multiple runs has advantages over truly sequential restart strategies, waiting for convergence before restarting.

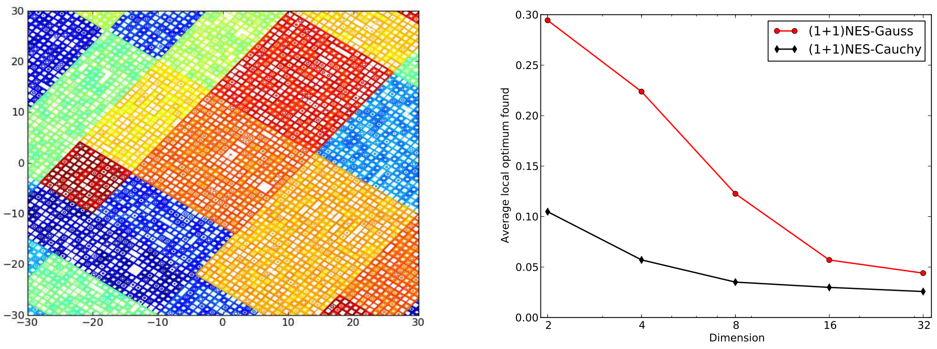


Figure 2.17: **Left:** Contour plot of (one instantiation of) the deceptive global optimization benchmark function f_{rb}, in two dimensions. It is constructed to contain local optima in unit-cube-sized spaces, whose best value is uniformly random. In addition, it is superimposed on 10^d-sized regional plateaus, also with uniformly random value. **Right:** Value of the local optimum discovered on f_{rb} (averaged over 250 runs, each with a budget of $100d$ function evaluations) as a function of problem dimension. Since the locally optimal values are uniformly distributed in $[0, 1]$, the results can equivalently be interpreted as the top percentile in which the found local optimum is located. E.g., on the 4-dimensional benchmark, NES with the Cauchy distribution tends to find one of the 6% best local optima, whereas employing the Gaussian distribution only leads to one of the best 22%.

In summary, our results demonstrate that it is indeed possible for an algorithm with a clean derivation from first principles to achieve state-of-the-art to best performance in the heuristics-dominated field of black-box optimization.

2.8 Discussion

Table 2.2 summarizes the various techniques we introduced. The plain search gradient suffers from premature convergence and lack of scale invariance (see section 2.2.2). Therefore, we use the natural gradient instead, which turns NES into a viable optimization method. To improve performance and robustness, we introduced several novel techniques. Fitness shaping makes the NES algorithm invariant to order-preserving transformations of the fitness function, thus increasing robustness. Importance mixing reduces the number of necessary samples to estimate the search gradient, while adaptation sampling adjusts NES's learning rates online. Empirically we found that using both importance mixing and adaptation sampling yields highly performant results on the standard benchmarks. In addition, restart strategies substantially improve success probabilities on multi-modal functions and noisy benchmarks, clearly outperforming alternative approaches. Lastly, the exponential parameterization is crucial for maintaining positive-definite covariance matrices, and the use of the natural coordinate system guarantees computational feasibility.

NES applies to general parameterizable distributions. In this chapter, we have experimentally investigated three variants, adjusted to the particular prop-

Table 2.2: Summary of enhancing techniques. Note that the last two techniques are applicable only to radial distributions.

Technique	Issue addressed	Section
Natural gradient	Scale-invariance, many more	2.3
Fitness shaping	Robustness	2.4.1
Fitness baseline	Robustness	2.4.2
Importance mixing	Performance, parameter sensitivity	2.4.3
Adaptation sampling	Performance, parameter sensitivity	2.4.4
Restart strategies	Reduced sensitivity to local optima	2.4.5
Exponential parameterization	Covariance constraints	2.5.1
Natural coordinate system	Efficiency	2.5.2

erties of different problem classes. We demonstrated the power of the xNES variant using a full multinormal distribution, which is invariant under arbitrary translations and rotations, on the canonical suite of standard benchmarks. Additionally, we showed that the restriction of the covariance matrix to a diagonal parameterization (SNES) allows for scaling to very high dimensions, both on the difficult non-Markovian double-pole balancing task and the Lennard-Jones cluster potential optimization problem. Furthermore, we demonstrated that using heavy-tailed distributions (Cauchy) instead of Gaussian distributions yields substantial benefits in global optimization scenarios with multiple or deceptive local optima.

Unlike many black-box optimization algorithms, NES boasts a clean derivation from first principles. The relationship of NES to methods from other fields, notably evolution strategies (Hansen and Ostermeier, 2001) and policy gradients (Bagnell and Schneider, 2003; Kakade, 2002; Peters and Schaal, 2008), should be evident to readers familiar with both of these domains, as it marries the concept of fitness-based black-box optimization from evolutionary methods with the concept of Monte Carlo-based gradient estimation from the policy gradient framework.

Chapter 3

Curiosity-driven Optimization

To achieve great things, two things are needed:
a plan, and not quite enough time.

Leonard Bernstein

The principle of *artificial curiosity* directs exploration towards the most informative or most *interesting* data. In this chapter, we show how it can be used to focus on the most pertinent search points in costly black-box optimization. We present a novel response surface method, which employs a memory-based model to estimate the interestingness of each candidate point using Gaussian process regression. For each candidate point this model estimates expected fitness improvement, and yields a closed-form expression of expected information gain. The algorithm continually pushes the boundary of a *Pareto-front* of candidates not dominated by any other known point according to both an information and a fitness criterion. This makes the exploration-exploitation trade-off explicit, and permits maximally informed search point selection. We illustrate the robustness of our approach in a number of experimental scenarios.

3.1 Artificial Curiosity

The ability to focus on novel, yet learnable patterns in observations is an essential aspect of intelligence that has led mankind to explore its surroundings, all the way to our current understanding of the universe. When designing artificial agents, we have exactly this vision in mind. However, if an artificial agent is to exhibit some level of intelligence, or at least the ability to learn and adapt quickly in its environment, then it is essential to guide this agent to experience such patterns, a drive known as *artificial curiosity* (Schmidhuber, 1991, 2006, 2009). However, this approach requires a principled way to judge and rank data, in order to drive itself towards observations exhibiting novel, yet learnable patterns. This property is compactly captured by the subjective notion of *interestingness*. Artificial learning agents are dependent on the interestingness of their observations. A number of formalizations of interestingness exist, although some of these have shortcomings. Our aim here is to find a formal measure of interestingness that can be used to guide exploration in the less general case of black-box optimization.

3.1.1 Background

Curiosity is the drive to actively explore the interesting regions in search space that most improve the model's predictions or explanations of what is going on in the world. Originally introduced for reinforcement learning (Schmidhuber, 1991; Storck et al., 1995), the curiosity framework has been used for active learning (Deisenroth et al., 2009; Pfingsten, 2006), to explain certain patterns of human visual attention better than previous approaches (Itti and Baldi, 2006), and to explain concepts such as beauty, attention and creativity (Schmidhuber, 2007, 2009).

The interestingness of a new observation is the difference between the performance of an adaptive model on the observation history before and after including the new point. The goal of the active data point selection mechanism is to maximize expected cumulative future interestingness. Various proposed distance measures include:

- the difference in data compressibility before and after letting the learning algorithm take the new data into account (Schmidhuber, 2007, 2009),
- the difference in mean squared prediction error on the observation history before and after re-training with the new point (Schmidhuber, 1991, 2006),
- the Kullback-Leibler (KL) divergence between belief distributions before and after the new observation (Storck et al., 1995).

Note that interestingness is observer-dependent and dynamic: a point that was interesting early on can become boring over time.

3.1.2 Artificial Curiosity as a Guide for Optimization

Motivated by *costly optimization* (as described in section 1.2), we here aim to define an appropriate variant of artificial curiosity that can guide the exploratory drive of an optimization algorithm to pick the most interesting next search point, an thus minimize the number of evaluations required.

The optimization framework is a restricted case of the RL scenario for which artificial curiosity has been put forth, in that fitness evaluations are atemporal (the order in which a set of points is evaluated does not affect their fitness). This allows for a simplified measure of interestingness that only captures the instantaneous informativeness of a search point.

Further, to permit the incorporation of a Bayesian prior, we will focus on probabilistic models and use a particular variant of the KL-based approach

(Storck et al., 1995) to maximize *information gain* (Chaloner and Verdinelli, 1995; Cohn, 1994; Fedorov, 1972; Hwang et al., 1991; MacKay, 1992; Plutowski et al., 1994). The KL-divergence or relative entropy between prior and posterior (before and after seeing the new point) is invariant under any transformation of the parameter space.

3.1.3 Formal Framework

Formally, let Y_{env} be the environment of interest, and y_{pre} be our current knowledge[1]. The information gain (interestingness) $\psi(y|y_{pre})$ brought about by the observation y is defined as

$$\begin{aligned}\psi(y|y_{pre}) &= \mathbb{D}\left(\pi\left(Y_{env}|y_{pre};y\right)\|\pi\left(Y_{env}|y_{pre}\right)\right)\\ &= \int \pi\left(y_{env}|y_{pre};y\right)\log\frac{\pi\left(y_{env}|y_{pre};y\right)}{\pi\left(y_{env}|y_{pre}\right)}dy_{env},\end{aligned}$$

where $\pi(\cdot|\cdot)$ denotes a conditional probability and $\mathbb{D}(\cdot\|\cdot)$ denotes the KL-divergence. For a set of observations y_{pre}, it is also useful to define the leave-one-out (LOO) information gain for each observation y_o w.r.t. the remaining $y_{pre\backslash o}$ as

$$\psi_{LOO}(y_o) = \psi\left(y_o|y_{pre\backslash o}\right).$$

The information gain $\psi(y|y_{pre})$ is defined *a posteriori*, meaning that it is only defined after we see the value y. However, in most cases, we want to assess the information gain of an observation *a priori*, i.e., before seeing the value. This leads to the *expected information gain* of random variable Y, defined by

$$\begin{aligned}\Psi(Y|y_{pre}) &= \mathbb{E}\left[\psi(Y|y_{pre})\right]\\ &= \int\pi(y|y_{pre})\int\pi\left(y_{env}|y_{pre};y\right)\log\frac{\pi\left(y_{env}|y_{pre};y\right)}{\pi\left(y_{env}|y_{pre}\right)}dy_{env}dy\\ &= I\left(Y;Y_{env}|y_{pre}\right),\end{aligned}$$

which turns out to be the conditional mutual information between the observation and the environment.

3.2 Exploration/Exploitation Trade-off.

In this section, we propose a way of handling the fundamental exploration-exploitation trade-off. To make informed data selection decisions, they are postponed until the entire Pareto-optimal front – with respect to both an exploration and an exploitation objective – is known.

Exploration Objective: Information Gain. The previous section introduced maximal expected information gain as a possible objective for exploration,

[1]We use upper case letters for random variables, and the corresponding lower case letters for specific configurations.

as an instantiation of the curiosity principle. It determines the most informative points in terms of model improvement. But, as it does not promise maximal fitness improvement, we need a second objective.

Exploitation Objective: Expected Fitness Improvement The straightforward choice of objective for exploitation would be to maximize the expected fitness. However, in optimization, there is an asymmetry in utility: solutions that are better than the best currently found f_{max} largely outweigh those that are almost as good. Thus exploitation really aims at maximizing the *expected improvement* in fitness with respect to f_{max}. It can be shown (Jones, 2001) that the expected improvement takes the following form:

$$\Delta(x) = \sigma(Y|y_o)\left(s\Phi(s) + \phi(s)\right),$$

where

$$s = \frac{f_{max} - \mathbb{E}[Y|y_o]}{\sigma(Y|y_o)},$$

while $\Phi(\cdot)$ and $\phi(\cdot)$ are the cumulative distribution and density functions of Gaussian distributions, respectively.

Combining the Objectives. Optimizing conflicting objectives necessarily involves some form of trade-off, which is typically handled by using a weighted sum of both objectives, where the weights are set manually, or tuned to the problem. Combining two objectives of different scale into a single utility measure is common practice (Chaloner and Verdinelli, 1995), but problematic (Zhang et al., 2003), as one of the objectives can completely dominate in some regions of the fitness landscape, while being dominated in others.

Therefore we propose turning the problem around and only deciding on the trade-off after *first* having evaluated both objectives for a large set of candidate points. This means finding the Pareto-front of candidates that are non-dominated w.r.t. expected improvement and expected information gain, which can be performed by any multi-objective optimization method (see section 1.1.5), for example the Non-dominated Sorting Genetic Algorithm version II (NSGA-II; Deb et al., 2002) which is used in the experiments in section 3.7.

All non-dominated candidates are considered "good" solutions, and therefore each should be assigned a non-zero probability of being chosen. Ideally, this probability should favor candidates that stand out on the Pareto-front, in terms of combining both objectives, but it should also be insensitive to quirks of the algorithm that builds the front (i.e. varying candidate densities), and to any smooth order-preserving transformation of the fitness function. In addition, it can allow us to shift the focus from one objective to the other, e.g. exploitation becoming more important over time.

In the absence of an optimal way of handling this decision, we opt for an unbiased solution, which consists of choosing the next point uniformly at random from the Pareto-front.

3.3 Curiosity-driven Optimization (General Form)

Algorithm 10 combines the components described in the previous section into a general framework for curiosity-driven optimization. In each iteration it first fits a probabilistic model M_f to all known points X, and then uses M_f to determine the LOO-information gain at each point. This interestingness function is then approximated using a second model, M_ψ. The Pareto-front of the candidate points is then computed using a multi-objective optimization algorithm, each model providing an estimate for one of the two objectives. Finally, a new point x^* is chosen, as described in section 3.2.

Algorithm 10: Curiosity-driven Optimization

```
   input: fitness function $f$, models $M_f$ and $M_\psi$, initial points $X$
1  repeat
2      Fit $M_f$ to $X$
3      for $s$ in $X$ do
4          $\psi_{LOO}(s) = \mathbb{D}\left(M_f(X) \| M_f(X_{-s})\right)$
5      end
6      Fit $M_\psi$ to $\psi_{LOO}$
7      Find a set $C$ of non-dominated candidate points
8              maximizing information gain (estimated by $M_\psi$) and
9              maximizing fitness (estimated by $M_f$)
10     Choose $x^*$ from $C$
11     $X \leftarrow X \cup \{(x^*, f(x^*))\}$
12 until stopping criterion is met
```

3.4 Models of Expected Fitness and Information Gain

The class of probabilistic models used for M_f and M_ψ should be general and flexible enough to fit multi-modal and highly non-linear fitness functions. Ideally, for every unknown point, such a model should be able to (efficiently) predict the expected value, the expected uncertainty associated with that prediction, and provide an analytical expression for computing information gain.

One option would be to use a mixture of Gaussians on the joint parameter-fitness function space (as in Cohn et al., 1995). However, this approach has the drawback of being sensitive to the number of Gaussians used, as well as giving poor interpolation in regions with few sampled points. Neural networks are another viable option, but coming with a propensity of overfitting the scarce data.

3.4.1 A Good Model Choice: Gaussian Processes

In this section we present an implementation of curiosity-driven optimization which satisfies all the above criteria by using *Gaussian processes* to model the fitness function.

Gaussian processes (GP, Rasmussen and Williams, 2006) can be seen as a probability distribution over functions, as evaluated on an arbitrary but finite number of points. Given a number of observations, a Gaussian process associates a Gaussian probability distribution to the function value for each point in the input space. Gaussian processes are capable of modeling highly complex fitness landscapes through the use of appropriate covariance (kernel) functions, and are commonly used for regression and function modeling (Rasmussen and Williams, 2006).

Formally, we consider the Gaussian process with zero mean and the kernel function

$$k\left(x, x'\right) + \sigma_n^2 \delta\left(x, x'\right),$$

where $\delta\left(\cdot,\cdot\right)$ is the Kronecker delta function. Thus, for any values y, y' at x, x', $\mathbb{E}\left[yy'\right] = k\left(x, x'\right) + \sigma_n^2$. We make the assumption that the function k is smooth and local in the sense that $k\left(x, x'\right) \to 0$ when $|x - x'|$ goes to infinity.

3.4.2 Derivation of Gaussian Process Information Gain

The concept of information gain can easily be mapped onto Gaussian processes, but previous work has failed to provide a closed form expression for efficiently computing it for each candidate point (Krause and Guestrin, 2007). Let us consider a collection of fixed reference points x_r, and view their value Y_r as the environment of interest. Our prior knowledge y_{pre} consists of all previously evaluated points x_o with value y_o. The expected information gain of the value Y at point x is thus defined by

$$\Psi_r\left(x|y_o\right) = I\left(Y_r; Y|y_o\right) = H\left(Y|y_o\right) - H\left(Y|Y_r, y_o\right),$$

where $H(\cdot|\cdot)$ is the conditional entropy. A major disadvantage of this definition is that the expected information gain depends on the reference points x_r. However, we may consider the situation where the number of reference points goes to infinity. By definition, $\pi\left(Y|Y_r, y_o\right)$ is a Gaussian distribution, and

$$H\left(Y|Y_r, y_o\right) = \frac{1}{2}\log 2\pi e\sigma^2\left(Y|Y_r, y_o\right).$$

Here $\sigma^2\left(Y|Y_r, y_o\right)$ is the predictive variance at x given by

$$\begin{aligned}\sigma^2\left(Y|Y_r, y_o\right) &= \sigma_n^2 + k\left(x, x\right) - k\left(x, x_{ro}\right)\\ &\qquad \left(k\left(x_{ro}, x_{ro}\right) + \sigma_n^2\mathbb{I}\right)^{-1} k\left(x_{ro}, x\right)\\ &= \sigma_n^2 + \sigma_e^2.\end{aligned}$$

with $x_{ro} = [x_r, x_o]$. We take advantage of the fact that in GP, the predictive variance depends only on the location of observations. In particular, σ_e^2 is the variance of the predicted mean $\bar{y} = \mathbb{E}[Y]$, and

$$0 \leq \sigma_e^2 = \sigma^2(\bar{y}|Y_r, y_o) \leq \sigma^2(\bar{y}|Y_r) = \sigma_s^2,$$

because conditioning always reduces variance for Gaussian distributions. According to (Rasmussen and Williams, 2006), σ_s^2 converges to 0 when the number of reference points x_r around x goes to infinity. This indicates that σ_e^2 converges to 0 uniformly w.r.t. y_o, thus $\sigma^2(Y|Y_r, y_o)$ converges to σ_n^2 uniformly w.r.t. y_o.

When the number of observation points is sufficiently large around any given point x, we have

$$\begin{aligned}\Psi_r(x|y_o) &= H(Y|y_o) - H(Y|Y_r, y_o)\\ &\to \frac{1}{2}\log 2\pi e\sigma^2(Y|y_o) - \frac{1}{2}\log 2\pi e\sigma_n^2\\ &= \frac{1}{2}\log \sigma^2(Y|y_o) - \frac{1}{2}\log \sigma_n^2.\end{aligned}$$

The limit no longer depends on the reference points, thus it can be used as an 'objective' measure for the expected information gain at point x:

$$\Psi(x|y_o) = \frac{1}{2}\log \sigma^2(Y|y_o) - \frac{1}{2}\log \sigma_n^2.$$

The second term is constant, therefore there is a direct connection between the expected information gain and the predictive variance given the observation, which can be computed efficiently. Note that Seo et al. (2000) found the predictive variance to be a useful criterion for exploration, without realizing that it is equivalent to information gain.

3.5 Curiosity-driven Optimization with Gaussian Processes

Choosing a Gaussian process to model the fitness function significantly simplifies the general algorithm introduced in Section 3.3. First, it allows us to compute the expected information gain Ψ instead of the less robust LOO-information gain. Second, the model M_ψ is no longer necessary, as Ψ can be computed for unknown points as well. The resulting algorithm (CO-GP) is shown in Algorithm 11. The remainder of this section discusses some practical considerations.

Computational Complexity. The computational complexity of each iteration of CO-GP is dominated by one matrix inversion of $\mathcal{O}(n^3)$, where n is the total number of evaluations. Building the Pareto-front consumes most of the computation time early on, but scales with $\mathcal{O}(n^2)$. The computational complexity of Gaussian processes can be reduced e.g. by implementing them online and

Algorithm 11: Curiosity-driven Optimization with Gaussian processes (CO-GP)

```
input: fitness function $f$, kernel function, initial points $X$
1 repeat
2     Fit a Gaussian process $\mathcal{G}$ to $X$
3     Find a set $C$ of non-dominated candidate points $x$ maximizing $\Psi_{\mathcal{G}}(x)$
      and $\Delta_{\mathcal{G}}(x)$
4     Choose $x^*$ from $C$
5     $X \leftarrow X \cup \{(x^*, f(x^*))\}$
6     Optionally optimize the kernel hyperparameters w.r.t. the marginal
      likelihood
7 until stopping criterion is met
```

using a reduced base vector set, containing only the most informative points (Csató and Opper, 2002). We have not implemented these yet, as computation time was not a major concern in our experiments.

Choosing Kernel Parameters: Model Selection. Gaussian process regression only gives reasonable results when the kernel hyperparameters are set properly. Depending on how much computation time is available, the hyperparameters could be optimized periodically with respect to the marginal likelihood. We use xNES (see chapter 2) for this purpose. Potentially, we could also employ diagnostic methods (Jones et al., 1998) to determine whether the model is appropriate.

Informed Multi-objective Search. At each iteration, an inner multi-objective optimization algorithm is used (in our case, NSGA-II, Deb et al., 2002). We can make use of our available information to make this step more efficient. For example, we initialize the search with the Pareto-front found in the previous iteration. Furthermore, as we want the search to roughly cover the range of known points, we adjust the scale (for step-sizes) accordingly.

3.6 Minimal Asymptotic Requirements

To demonstrate the practical viability of CO-GP, we first investigate how it handles a number of common but problematic scenarios and then illustrate it on a standard benchmark function. Following Rasmussen and Williams (2006), all experiments use the common Gaussian kernel (also known as radial basis function) with noise, which is a very robust choice in practice.

3.6.1 Reaching Optima at Arbitrary Distance

Many fitness function landscapes contain large linear regions. Specifically, if the scale of the region covered by the initial points is too small, almost any landscape will appear linear. An ideal algorithm should be able to exploit the linearity

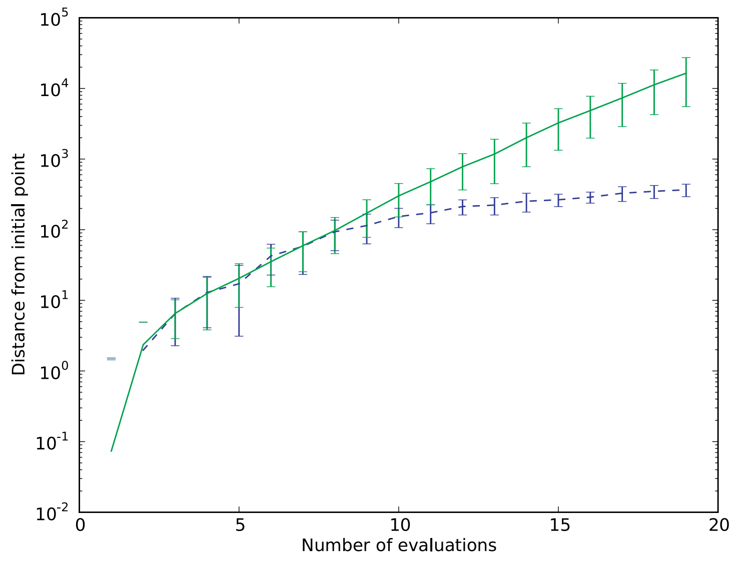


Figure 3.1: **Escaping linear regions.** The plot shows the performance of CO-GP on a linear surface (f_5, see section 2.7.2), in terms of the distance from the best point found so far to the initial point (averaged over 20 independent runs). The green (solid) and blue (dashed) curves correspond to CO-GP with hyperparameter adaptation enabled and disabled, respectively. We observe that the distance from the initial point (and thus the increase in fitness) grows exponentially if the hyperparameters are adapted, but only linearly otherwise.

of such regions. In particular, it is highly desirable to have the searched region grow exponentially with the number of points. Note that many well-known algorithms, such as estimation of distribution algorithms (see section 1.3), do not have this property, and instead rely either on correct initialization or heuristics (Bosman et al., 2008). In contrast, CO-GP does have this property, as our results on the linear function show (see Figure 3.1).

3.6.2 Locating Optima with Arbitrary Precision

While designed primarily for multi-modal fitness landscapes, we investigated how our approach handles simple fitness landscapes with a single optimum. The success criterion for this case is to have the distance to the optimum decrease exponentially with the number of points. While we cannot prove that this is the case in general, Figure 3.2 shows that it holds for the multi-dimensional sphere function. This indicates that CO-GP can locate optima up to a high precision, at least whenever, locally, the fitness function is approximately quadratic.

3.6.3 Guaranteed to Find Global Optimum

Every global optimization algorithm should provide a guarantee that in the limit its chosen points will cover the search space densely, which is the only way to

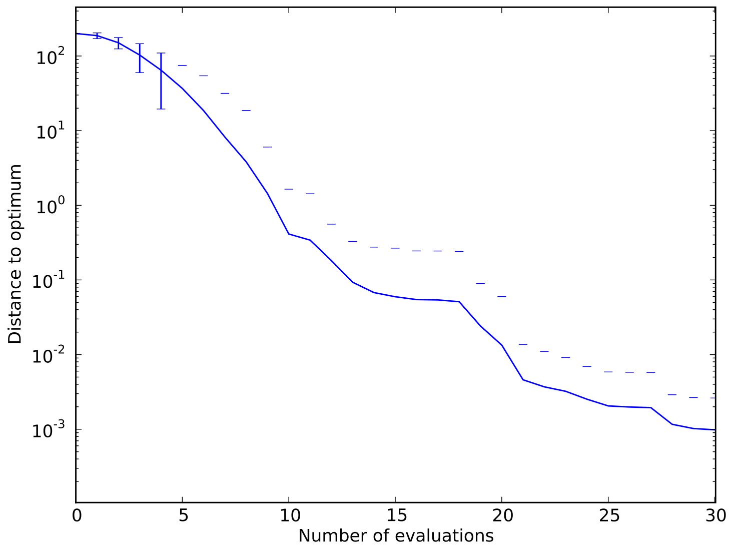


Figure 3.2: **Precisely locating optima.** The plot shows the performance of CO-GP on a unimodal, quadratic surface (f_1, see section 2.7.2), in terms of the distance from the optimum to the best point found so far (averaged over 20 independent runs). This distance decreases exponentially with the number of points.

ensure that it will eventually find the global optimum. Optimization based on expected improvement has been shown to have this property (Locatelli, 1997). It turns out that if we remove the information gain objective from CO-GP, the algorithms are equivalent. Therefore, as one extreme of the Pareto-front will always correspond to the point maximizing expected improvement exclusively, and that point has a non-zero probability of being chosen, CO-GP inherits the property that it always finds the global optimum in the limit.

3.7 Proof-of-concept

The Branin function (Jones, 2001; Jones et al., 1998) is a commonly used benchmark for global optimization of the form:

$$f_{Branin}(x_1, x_2) = a(x_2 - bx_1^2 + cx_1 - d)^2 + e(1 - f)\cos(x_1) + e,$$

where the standard parameter settings are $a = 1$, $b = \frac{5}{4\pi^2}$, $c = \frac{5}{\pi}$, $d = 6$, $e = 10$, $f = \frac{1}{8\pi}$. The function has three global optima, at $(-\pi, 12.275)$, $(\pi, 2.275)$ and $(9.42478, 2.475)$, with value $f_{Branin}(x_\star) = 0.397887$, a bounded domain and a non-trivial structure. Figure 3.3 illustrates the behavior of CO-GP on the Branin function over the course of a single run, starting with four points on the boundary corners. The Gaussian process model produces a good fit of the true function after about 30 iterations. Locating one optimum (up to a precision of 0.1) requires only 28 ± 8 evaluations, locating all three requires 119 ± 31. The qualitative behavior of the algorithm is very intuitive, placing part of its search points spaced broadly within the domain, while the other part forms clusters of points ever closer around the optima. Although this experiment is intended as a proof of concept, not an empirical comparison to other global optimization algorithms, the quantitative results indicate that CO-GP is on par with the best results reported in the literature (Jones, 2001).

3.8 Discussion

The results in sections 3.6 and 3.7 demonstrate that curiosity-driven optimization properly handles a number of typical optimization challenges. Although based on the general and theoretically powerful principle of artificial curiosity, our current implementation exhibits certain weaknesses.

In particular, despite the derived closed-form expressions for the objectives, the method's computational cost is still high, limiting the application domain to (relatively) costly optimization problems. On the other hand, many real-world problems (see section 1.4) are precisely of this type. Nevertheless, in future research we would like to extend the method such that it is possible to specify the amount of acceptable computational effort for the selection of each data point, and have the algorithm make the best choice under this constraint.

Another drawback of our approach is that it is greedier than the original curiosity framework: it does not necessarily maximize cumulative future expected information gain, but greedily selects the next data point that is expected to be the immediately most interesting. More sophisticated reinforcement learning algorithms will be necessary to maximize the expected *sum* of future intrinsic rewards (each reward being proportional to the information gain of the corresponding observed data).

The results bode well for applying the general template (Algorithm 10) to related domains such as constrained or discrete optimization, or even mixed-integer programming. One application domain where we expect curiosity-based methods to be counter-productive, however, are problems that require minimizing cumulative evaluated regret, because those tend to require risk-averse exploration (which in practice generally translates to careful local search): here curiosity may kill the cat.

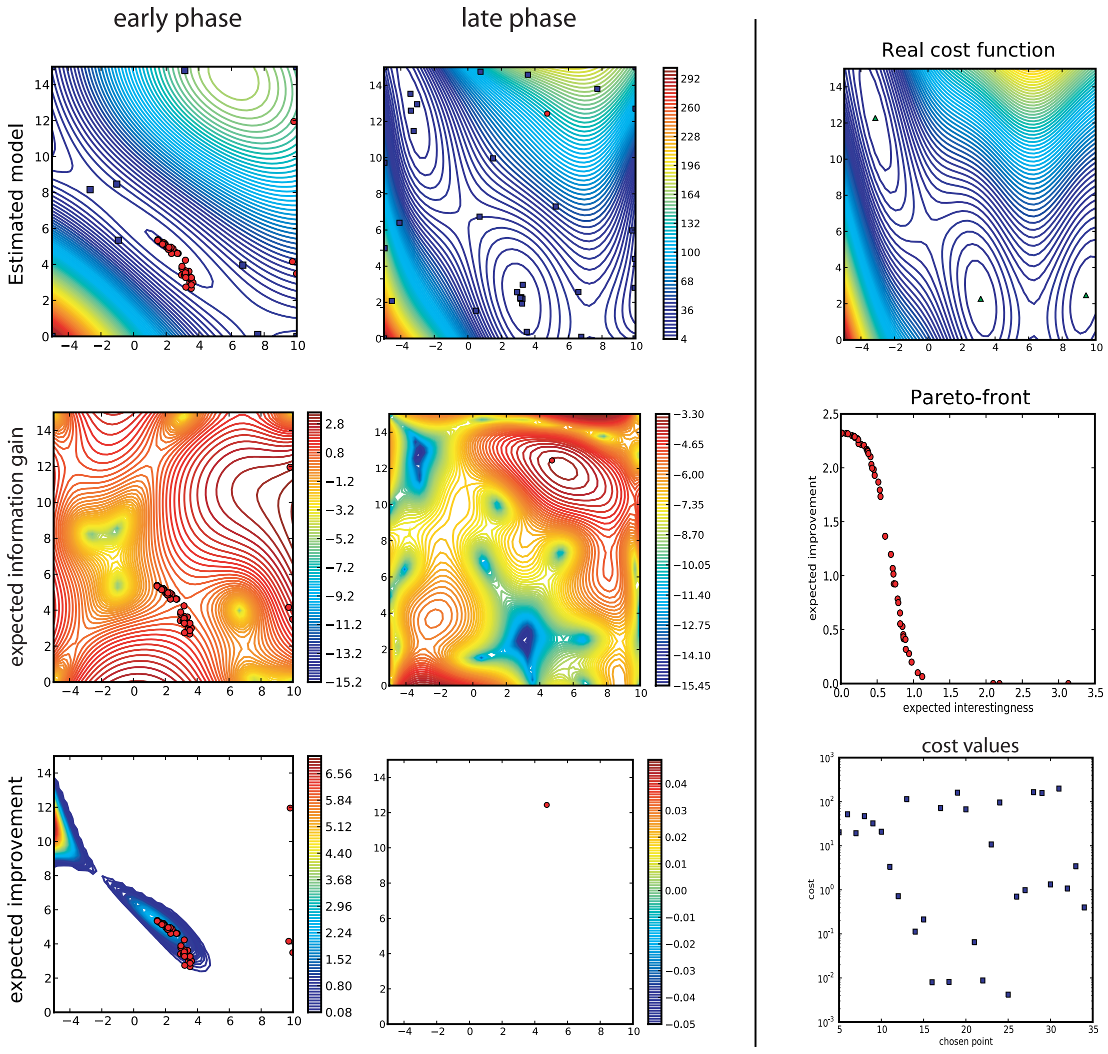


Figure 3.3: **Optimization on the Branin function.** The plot in the top right corner shows a contour plot of the Branin function with the three global optima marked with triangles. The left column shows the estimated fitness function model (top), and the two competing objectives, expected information gain (middle) and expected improvement (bottom), in an early phase after 10 points (blue squares) have been observed. The red circles are the points on the Pareto-front being considered for the next choice. The middle column shows the same information after 30 iterations. Note that in this later stage the model is very close to the true function (top). The plot in the middle of the right column shows the shape of the Pareto-front corresponding to the situation in the left column (10 points), and the plot on the bottom right shows the values of the fitness function at all the chosen points (the initial 4 corner points are not shown). In the early phase, the Pareto-front contains a continuum of points in the center that trade off improvement and information gain, plus a few isolated points with high information gain, but very low expected improvement. After 30 iterations, two of the global optima have been located precisely. The expected improvement is zero everywhere, so the Pareto-front is collapsed to the single point with highest information gain. CO-GP now performs a purely exploratory step, and will continue to do so until it leads to non-zero expected improvement (e.g. around the third optimum). On average, CO-GP requires about 120 points to locate all three optima with high accuracy.

Chapter 4

A Study in Scalability

You've got to think about big things
while you're doing small things,
so that all the small things
go in the right direction.

Alvin Toffler

LEARNING to solve small instances of a problem should help in solving large instances. This type of scalability is the subject of this chapter, where we show how black-box optimization can be employed to optimize the weights of a novel neural network architecture, which in turn results in a transfer of performance on flexible-size board games. The networks developed (multi-dimensional recurrent ones) exhibit a high degree of scalability, which allows them to be trained from scratch up to the level of human beginners, without using domain knowledge.

4.1 Scalability in Problem Size

In a wide range of domains it is possible to learn from a simple version of a problem and then use this knowledge on a larger one. This particular form of incremental learning is commonly employed by humans, and for machine learning it is especially useful when training on the large version is much more expensive.

Board games are a particularly suitable domain for investigating this form of *scalability*, because for many of them either the board size can be varied, or the rules can be trivially adjusted to make it variable. In addition, despite being described by a small set of formal rules, they often involve highly complex strategies. One of the most interesting board games is the ancient game of *Go*, among other reasons, because computer programs are still weaker than human players. Its extremely large search space defies traditional search-based methods. Human experts rely heavily on patterns, and thus it is not surprising that a substantial amount of research effort has been devoted to applying neural networks – which are good at pattern recognition – to Go (Richards et al., 1997; Wu and Baldi, 2007). Most of these methods do not scale well w.r.t. board size, i.e. networks trained successfully on small boards (where training is efficient) do not play well when the board is enlarged (Runarsson and Lucas, 2005; Stanley and Miikkulainen, 2004b).

We propose a scalable approach based on Multi-dimensional Recurrent Neural Networks (MDRNNs; Graves, 2008; Graves et al., 2007) which enhances the ability of that architecture to capture long-distance dependencies. We conduct experiments on three different Go-inspired games, which is possible without modifying our network architecture as it is free of domain knowledge. We train it against opponents of varying difficulty and measure how the playing performance scales to larger board sizes. Furthermore, we put our architecture into context by comparing it to a number of competing ones.

4.2 Example Domain: the Game of Go

Games are a particularly interesting domain for studies of machine learning techniques. They can usually be described by a small set of formal rules and clear success criteria, and yet they often involve highly complex strategies. Board games generally exhibit these features to a high degree, and so it is not surprising that the field of machine learning has devoted major efforts to their study, with the result that in almost all popular board games, most notably chess, computer programs can beat all human players. Probably the most interesting exception is the ancient game of *Go*, which can be solved for small boards (van der Werf et al., 2003) but is very challenging for larger ones (Richards et al., 1997; Runarsson and Lucas, 2005).

4.2.1 Rules of the Game

The rules of Go are simple (Iwamoto, 1972), but the strategies emerging from them are highly complex. Players alternately place stones onto any of the unoccupied intersections of the board (minimum size of 5x5, to 19x19 on a regular board), with the goal of conquering maximal territory. A player can capture a single stone or a connected group of his opponent's stones by completely surrounding them with his own stones. A move is not legal if it leads to a previously seen board position (to avoid cycling). The game is over when both players pass.

4.2.2 Challenges of Go

Go has a very high branching factor because at any moment there are about as many legal moves as there are free positions on the board (on average 200). This makes using traditional search-based methods prohibitively expensive.

Go also exhibits a number of interesting symmetries. Apart from the four straightforward axes of symmetry, it also has an approximate translational invariance, which is stronger, the further from the border a pattern is situated.

Among the practical difficulties with conducting experiments on Go are the need to distinguish dead groups from alive and seki ones (Iwamoto, 1972), keep track of the history of all board configurations, and the difficulty of handling pass moves. Many other machine learning approaches to Go simplify the rules to prevent some or all of those problems.

4.2.3 Simpler Variants of Go

A number of variations of Go, of varying degrees of similarity, are played on the same board and share the symmetry properties of Go. The most common ones are Irensei, Renju, Tanbo, Connect, Atari-Go, Go-Moku and Pente. We conduct experiments on the latter three.

Atari-Go, also known as Ponnuki-Go or 'Capture Game', is a simplified version of Go that is widely used for teaching the game of Go to new players because it introduces many of the key concepts without overwhelming the novice player with the full complexity of Go (Konidaris et al., 2002). The rules are the same as for Go, except that passing is not allowed, and the first player to capture a predetermined number (here: one) of their opponent's stones wins. Compared to Go, this variant makes playing independent of history and it simplifies determining the winner, which in turn allows for easier and faster automated playing. It retains the concept of territory: as in the end no player may pass, each one has to fill their own territory and therefore the player with most territory wins. Other strategies of Go, such as building eyes, or recognizing life-and-death situations still exist in Atari-Go, but are less important. See also Figure 4.9 for an illustration.

Go-Moku is also known as 'Five-in-a-row'. Players alternate putting stones onto any of the intersections on the board. The first player to have five connected stones in a row, column or diagonal, wins. While trying to block off the opponent's lines, a player tries to keep their own lines unblocked, and potentially do multiple attacks at the same time, not all of which can be countered. These strategies, again, are heavily pattern-based (Freisleben and Luttermann, 1996).

Pente has similar rules to Go-Moku, except that it is now possible to capture stones, in pairs, by putting stones at both ends of a pair of the opponent. The game is won by the first player who either has five connected stones, or has captured five pairs.

4.2.4 Computer Opponents

For our experiments, we have the following predefined opponents associated with each game variant:

1. a *random* player, which randomly chooses any of the legal moves,
2. a *naive* player, which does a one-ply-deep search. If possible, it always picks a move that makes it win the game immediately, and never picks a move that would make it lose the game immediately. In all other cases (the large majority), it randomly picks a legal move,
3. a publicly available *heuristic* player (only for Atari-Go), based on a set of hand-coded heuristic tactics (exploited greedily, see Gherman, 2000; Grüttner, 2008). Its difficulty can be adjusted by imposing that a proportion ϵ of its moves are chosen randomly. According to Gherman (2000), the level of play, with $\epsilon = 0$, is 'challenging for beginners'.

4.3 Scalable Neural Architectures

We consider a neural network architecture to be scalable if it is not tied to a fixed input dimension. This section provides an overview of such architectures that have been proposed for solving board games, it then describes the multi-dimensional extensions of recurrent neural networks (RNNs) in general, and finally gives the details of the specific instantiation we propose.

One approach to designing scalable network architectures is to scan across the board, processing the inputs from a limited *receptive field*, independently of their positions. The outputs of that stage are then fed into another architecture that combines them (e.g. Silver et al., 2007). An extension of this idea are the *convolutional networks* (Lecun and Bengio, 1995), which repeat this step on multiple levels, thereby capturing higher-level features instead of just local patterns. These architectures introduce a trade-off: a small receptive field limits the kind of patterns that can be recognized, whereas a large one makes learning very difficult (because of the exploding number of parameters).

'Roving-eye'-based architectures (Stanley and Miikkulainen, 2004b) contain one component with a fixed receptive field that can be aimed at any part of the board. This is then combined with an active component that decides where to rove over the board, and when to choose an output action.

Other architectures have been proposed (Freisleben and Luttermann, 1996; Schraudolph et al., 1994) which make use of weight-sharing to capture domain-specific symmetries, but these are limited to a particular game, and also restricted w.r.t. what kind of strategies they can learn. Simultaneous Recurrent Networks (Pang and Werbos, 1996) are structured like cellular automata. They successfully incorporate the whole context and make use of symmetries, but are not very efficient.

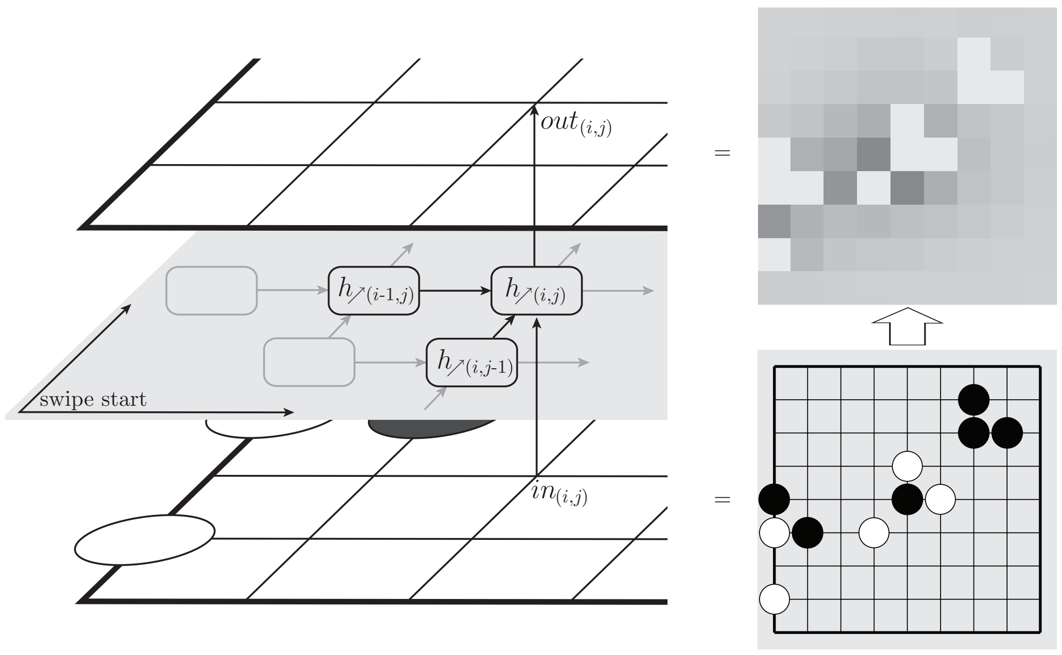


Figure 4.1: **MDRNN for Go.** The structure diagram on the left shows the connections of a hidden layer in one of the swiping directions: it receives its two earlier activations, as well as the local board input. The network as a whole takes the game board as input (bottom right) and outputs move preferences (top right). The darker the square, the higher the preference for the corresponding move (illegal ones are ignored).

For the related but different problem of scaling the problem *resolution*, a number of approaches for generative encodings of neural networks have been found to be successful (e.g. Compositional Pattern Producing Networks, Gauci and Stanley, 2007).

4.3.1 Multi-dimensional RNN

Multi-dimensional Recurrent Neural Networks (MDRNNs; Graves, 2008; Graves et al., 2007), are an extension of bi-directional RNN proposed by Schuster and Paliwal (1997), and a special case of the DAG-RNNs proposed by Baldi and Pollastri (2003). Their unbounded receptive fields (explained below) make them scalable by design. Successful applications include vision (Graves et al., 2007), handwriting recognition (Graves and Schmidhuber, 2008), and supervised learning of expert Go moves (Wu and Baldi, 2007).

Unlike standard recurrent neural networks which are only effective for handling sequences with a single (time-)dimension, MDRNNs are are applicable to multi-dimensional sequences (Graves, 2008). In the case of Go, the single time dimension is replaced by the two space dimensions of the game board.

Consider a hidden layer $h_{\nearrow}$ that *swipes* diagonally over the board from bottom-left to top-right. At each board position (i, j) its activation $h_{\nearrow(i,j)}$ is a function of the current input $in_{i,j}$ and its own earlier activations $h_{\nearrow(i-1,j)}$ and

$h_{\nearrow(i,j-1)}$:

$$h_{\nearrow(i,j)} = g_h \left(\mathbf{w}_{in} \cdot in_{i,j} + \mathbf{w}_h \cdot h_{\nearrow(i-1,j)} + \mathbf{w}_h \cdot h_{\nearrow(i,j-1)} \right)$$

where $\mathbf{w}_{\_\_}$ are connection weights and g_h is typically the sigmoid function. On the boundaries we use a fixed default *bias* value $h_{\nearrow(0,i)} = h_{\nearrow(i,0)} = \mathbf{w}_b$, for $0 < i \leq n$. See also Figure 4.1 for an illustration.

Because of the recurrent connections, the layer has indirect access to board information from the whole rectangle between $(0, 0)$ and (i, j). In order to have access to the whole board, we use four such swiping layers, one for each of the diagonal swiping directions in $D = \{\searrow, \nearrow, \swarrow, \nwarrow\}$. The output layer then, for every position, combines the outputs of these four layers into a single value $out_{i,j}$ (which is indirectly derived from the information of the entire input). More formally:

$$out_{i,j} = g_{out} \left(\sum_{\diamond \in D} \mathbf{w}_{out} \cdot h_{\diamond(i,j)} \right)$$

where the function g_{out} is typically the sigmoid function.

4.3.2 Multi-dimensional LSTM

Typically a swiping layer is composed of k sigmoidal neurons (e.g. $g_h = \tanh$). Although in theory such an MDRNN can learn to make use of the whole board context, it is very difficult to achieve in practice, because the information is propagated recurrently through non-linear units and thus tends to decay quickly with distance (Hochreiter and Schmidhuber, 1997). One solution to this problem is to use *Long short-term memory* cells (LSTM), which are based on protecting the recurrent state with *gating* sub-units (Hochreiter and Schmidhuber, 1997). As in (Graves, 2008; Graves et al., 2007), we therefore also use networks with swiping layer composed of k LSTM cells, and call them *MDLSTM*.

4.3.3 A Custom Architecture for Go

We instantiate MDRNNs such that they are appropriately generating a playing policy, given a symmetric game board. At each position, the network takes two *inputs* which indicate the presence of a stone at this position. The first one is 1 if a stone of the network's own color is present and 0 otherwise, the second input encodes the presence of an opponent's stone in the same way. A black/white symmetric encoding, as used in other approaches (e.g. Schraudolph et al., 1994) is not applicable here, because the output is not symmetrical: the best move for both players might be the same.

The *output* value at each position expresses the network's preference for playing there (see also Figure 4.1). Assuming that a deterministic playing behavior is desired, moves are selected greedily, randomly breaking ties. This is the case in our experiments because the opponents act stochastically. In practice, we ignore the network's preferences for illegal moves. For stochastic play one can interpret the preferences probabilistically, e.g. by drawing a position from the corresponding Gibbs distribution.

MDRNNs are invariant w.r.t. stationary shifts of the input. In order to also enforce rotation and reflection symmetry, we use the same connection weights for all swiping directions and the same $\mathbf{w}_b$ on all boundaries.

In our implementation we unfold the MDRNN along both spacial dimensions, leading to a large but simple feed-forward network. On a normal desktop computer (Intel Xeon 2.8 GHz), a network needs about 3ms to choose a move on a 7x7 board, and 25ms on a 19x19 board. The total number of weights is $4k + k^2$ for sigmoidal swiping layers (MDRNN) and $16k + 5k^2$ for LSTM layers (MDLSTM). The neural network, as well as the game benchmark code is available as part of the PyBrain library (see also Appendix A).

4.4 Experiments

As training networks is expensive, we start by empirically investigating the performance of the different networks (and their parameters) with random weights. Moreover, we determine under what circumstances the performance scales to larger boards. We then train the networks on small boards and analyze whether their performance improvement is transferred to larger boards. Finally, as training against a fixed opponent biases the direction of evolution towards beating the fixed opponent, we also perform experiments with coevolution.

4.4.1 Experimental Setup

As *fitness* we use the average outcome of 100 games against a fixed opponent, counting a win as 1, a draw as 0 and a loss as -1. Each player plays 50 times as black and 50 times as white. A positive fitness therefore implies that the agent is better on average than the opponent.

In addition to MDRNNs with sigmoidal neurons or LSTM cells (as described in section 4.3.3), we use – as a performance baseline – standard multi-layer perceptrons (MLP), containing a single fully connected hidden layer of size k, with tanh units. We compare the performance of our architecture to simple convolutional networks (CONV), with one layer of k feature maps (of identical receptive field size ρ by ρ), no subsampling, and a sigmoid output layer that combines the features. Here, the input board is padded with additional positions around the borders. They have $k(\rho^2 + 1) + 1$ parameters.

One the one hand, we analyze the performance of networks produced by the simplest possible algorithm, namely random weight guessing (drawing from $\mathcal{N}(0, 1)$). On the other hand we train the networks using CMA-ES (Hansen and Ostermeier, 2001) to optimize all the weights[1].

[1]We could obviously have used xNES instead, and obtained very similar results (see section 2.7), but at the time the research in this chapter was conducted, xNES did not yet

Our two quantitative measures of scalability are: *a*) the linear correlation (Pearson coefficient) between the fitness on different board sizes *b*) the proportion p of networks for which the fitness is higher on the larger board than on the smaller one.

4.4.2 Random-weight Networks

In a first set of experiments, we determine the bias of different architectures. We measure the performance of different kinds of networks with randomly guessed weights, on different games and against various opponents. Figure 4.2(a) shows the percentiles of fitness of random MDRNNs, giving an indication of the difficulty of the different opponents on each game. Training is easier if initial weights with reasonably good fitness (> 0) can be found relatively easily. This is indeed the case for the naive and the random opponent but not for the heuristic one. For MLPs, however, reasonable initial weights are very rare even for the naive player, as shown in Figure 4.2(b).

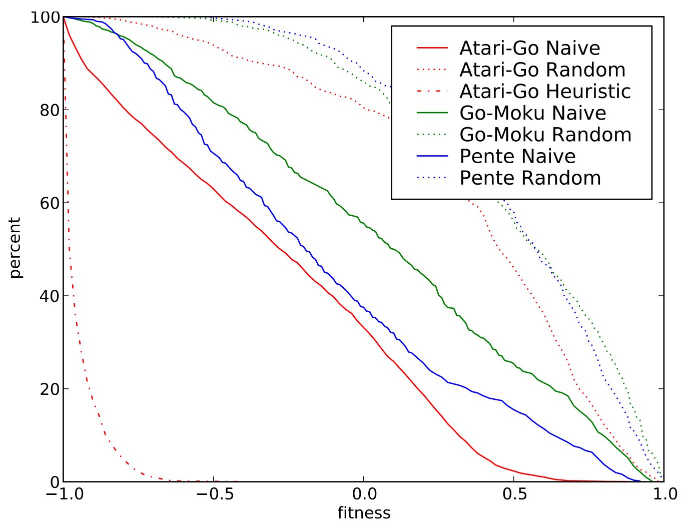
In Figure 4.3, we choose one scenario (Atari-Go, naive opponent) to explicitly show the differences between network architectures, comparing MDRNNs, MDLSTMs (for varying k), CONVs (for varying ρ) and MLPs. Those results, despite corresponding to random-weight networks, already indicate that small values of k are appropriate for MDRNNs (we will fix $k = 3$ hereafter), and do not bode well for MLPs.

Table 4.1: Correlations between the fitnesses of *random* MDLSTMs on different board sizes (based on 100 networks per scenario, evaluated against the naive opponent). They are high in all cases except for Go-Moku between 5x5 and larger boards, which is due to the fact that it is disproportionately easier for a game to end in a draw on a 5x5 board.

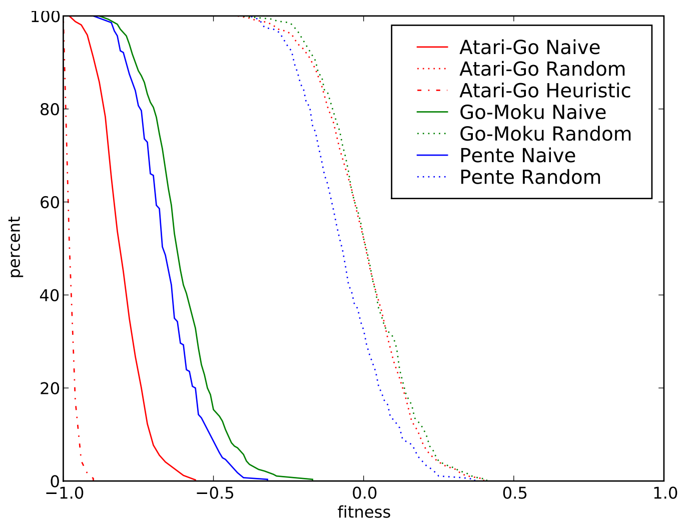
Sizes	Atari-Go	Go-Moku	Pente
5x5 vs. 7x7	0.86	0.20	0.47
5x5 vs. 9x9	0.72	0.09	0.31
5x5 vs. 11x11	0.67	0.37	0.49
5x5 vs. 15x15	0.40	0.29	0.52
5x5 vs. 19x19	0.37	0.38	0.46
7x7 vs. 9x9	0.88	0.83	0.83
7x7 vs. 11x11	0.82	0.85	0.87
7x7 vs. 15x15	0.60	0.81	0.67
7x7 vs. 19x19	0.62	0.59	0.64
9x9 vs. 11x11	0.92	0.92	0.90
9x9 vs. 15x15	0.75	0.94	0.72
9x9 vs. 19x19	0.71	0.76	0.64

We determine the scalability of random networks by evaluating the fitness on multiple board sizes and then computing their correlation (see Table 4.1).

exist.



(a) MDRNNs on different tasks.



(b) MLPs on different tasks.

Figure 4.2: Fitness of random networks evaluated on 7x7 (400 networks per scenario). The percentiles show what proportion of random networks reach at least a given fitness. For example, at least 25% or random MDRNNs win at least 75% of Go-Moku games against the naive opponent, i.e. have a fitness > 0.5.

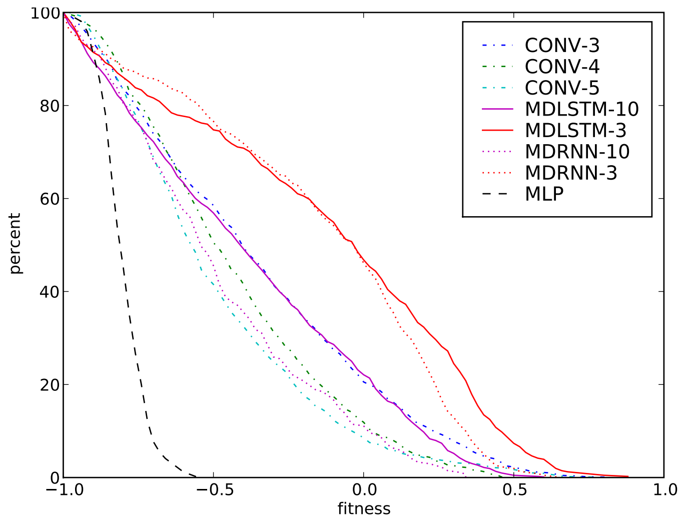


Figure 4.3: Fitness of different network architectures (CONVs of varying ρ, MDRNNs and MDLSTMs of varying k, and MLPs), when evaluated with random weights on 7x7 Atari-Go, against the naive opponent (400 networks per scenario). The percentiles show what proportion of random networks reach at least a given fitness. MDRNNs and MDLSTMs, each with $k = 3$, seem to be the most promising architectures, with over 40% of randomly initialized networks already playing equally good as the naive player.

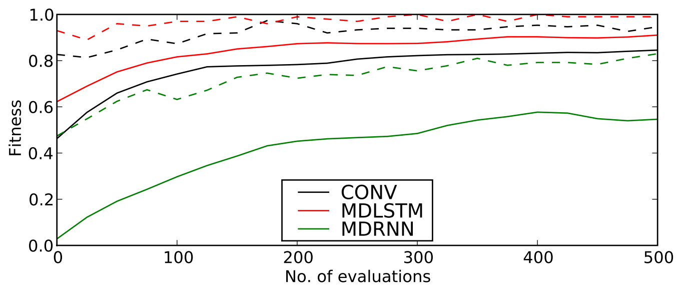
As the linear correlation by itself can be a misleading measure, we provide a visual intuition about the high correlation in Figure 4.6(a). The results indicate that one can train networks on boards as small as 7x7 and use them to play with similar performance on 19x19, for all three games.

4.4.3 Optimized Networks

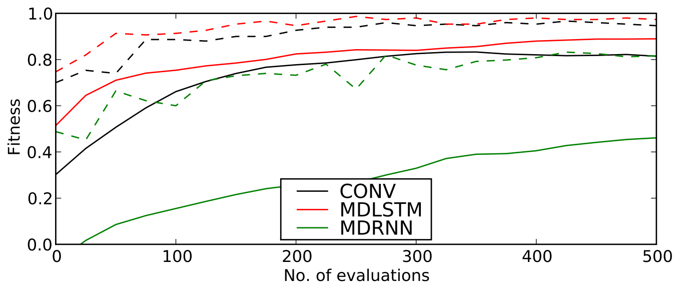
In the second set of experiments, we train different networks against the naive player on 7x7, using CMA-ES to optimize the weights. Figures 4.4(a), 4.4(b) and 4.4(c) show the learning curves for Go-Muko, Pente and Atari-Go, respectively. MLPs are in this comparison as a baseline, but clearly fail to learn how to play. The other architectures learn to beat the naive opponent, with MDLSTMs outperforming the others. Convolutional networks are learning slightly slower, but still faster than MDRNNs.

Learning to play against the significantly stronger heuristic opponent is a bigger challenge. Figures 4.5(a) and 4.5(b) show the learning curves against the heuristic player, with settings $\epsilon = 0.2$ and $\epsilon = 0.05$ respectively (averaged over 5 runs). Here, MDLSTMs clearly outperform convolutional networks, as well as MDRNNs, for which only the best results are shown (produced with $\rho = 5$)[2]. We suspect that this is due to the limited receptive field: at a certain level of play it becomes unavoidable to use non-local information. MDLSTMs can

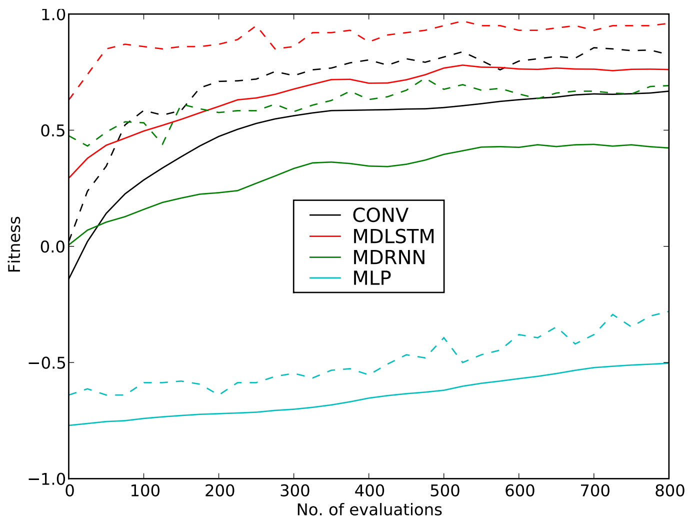
[2]Increasing ρ further vastly increases the number of parameters to be optimized, which slows down search



(a) Go-Moku



(b) Pente



(c) Atari-Go.

Figure 4.4: Learning curves for different network architectures when training against the *naive* opponent on the three different games (board size 7x7, averaged over 10 independent runs). The solid line corresponds to the average fitness per generation, the broken one corresponds to the best fitness per generation. For Atari-Go, the game is hardest to learn (c). For all three games the MDLSTMs achieve the best performance, closely followed by CONVs. Optimizing MLPs is much harder (only shown for Atari-Go).

learn how much context is necessary, and automatically increase their effective receptive field during optimization.

The scalability results for optimized networks are summarized in Table 4.2. Generally, there is a low correlation for the convolutional networks, but a relatively high one for MDLSTMs. Figure 4.6(b) illustrates this difference for networks trained against the naive opponent in Atari-Go. Note the large number of convolutional networks on the bottom right, for which good performance on 7x7 corresponds to poor performance on 11x11.

Table 4.2: Scalability of networks trained on 7x7 against the naive opponent (based on 100 networks per scenario). The correlations are higher for MDLSTMs than for convolutional networks. Also, note the really high proportion p of MDLSTMs that are stronger on 19x19 than on 7x7, for all games.

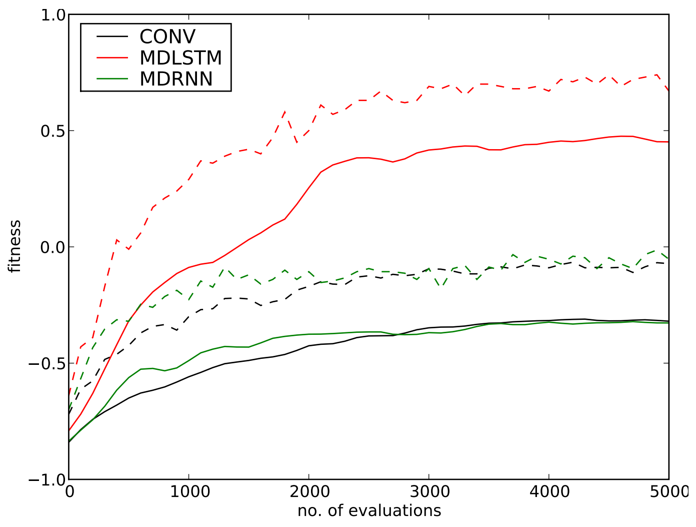
Game	Sizes	CONV		MDLSTM	
		Correlation	p	Correlation	p
Atari-Go	7x7 vs. 9x9	0.13	20%	0.38	48%
Atari-Go	7x7 vs. 11x11	0.17	18%	0.27	45%
Atari-Go	7x7 vs. 19x19	0.17	21%	0.38	76%
Go-Moku	7x7 vs. 9x9	0.06	42%	0.47	67%
Go-Moku	7x7 vs. 11x11	0.15	61%	0.38	87%
Go-Moku	7x7 vs. 19x19	0.04	68%	0.66	84%
Pente	7x7 vs. 9x9	0.05	45%	0.08	61%
Pente	7x7 vs. 11x11	0.24	58%	0.39	79%
Pente	7x7 vs. 19x19	0.23	58%	-0.05	95%

Comparing the correlations and the proportions p to the values for random MDLSTMs, we find the correlations to be lower but p to be higher (especially when scaling to 19x19). This means that the fitness on a small board is not perfectly predictive of the fitness on the large board, *but* it is almost always higher on the large board. Of particular interest is the scalability of the networks trained against the strongest opponent – as Figure 4.6(c) illustrates (for 7x7 versus 15x15), MDLSTMs achieve a high correlation (0.64), unlike convolutional networks (0.08).

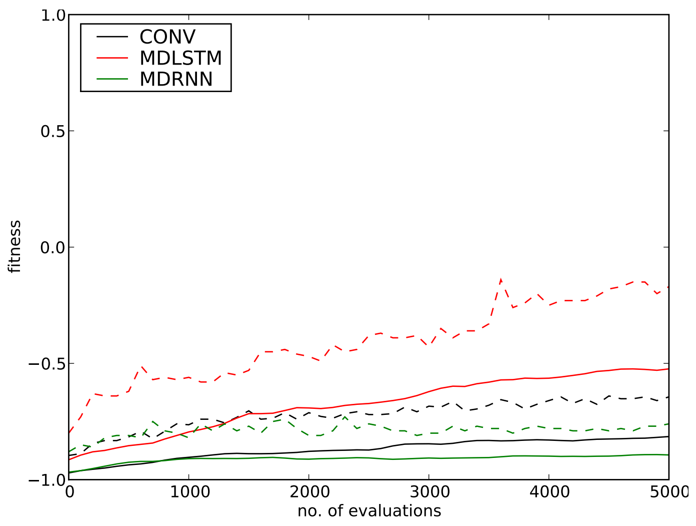
4.4.4 Coevolved Networks

Training against a fixed opponent both biases the direction of evolution and limits performance to that of the opponent, so we decided to also perform experiments with *coevolution*, which overcomes that limitation.

We use population-based competitive coevolution, based on the *host-parasite* paradigm (as described in Lubberts and Miikkulainen, 2001). In order to preserve diversity, we use the following standard enhancements (from Rosin and Belew, 1995): shared fitness, shared sampling and hall of fame. We use a population size of two times 15, with an elitist selection of $\frac{1}{3}$ based on shared fitness. At every generation, every host is evaluated against 15 opponents, 5 of them

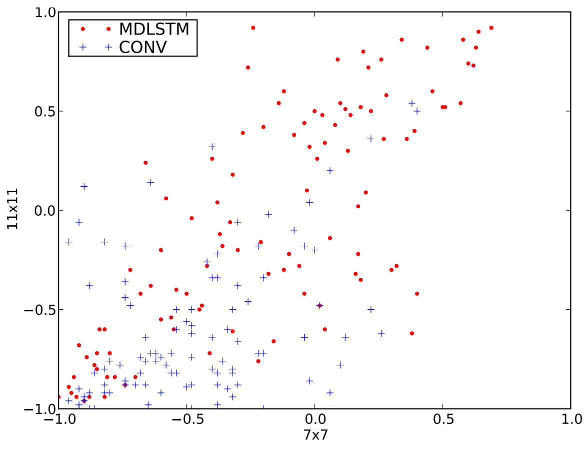


(a) Heuristic opponent with $\epsilon = 0.2$.

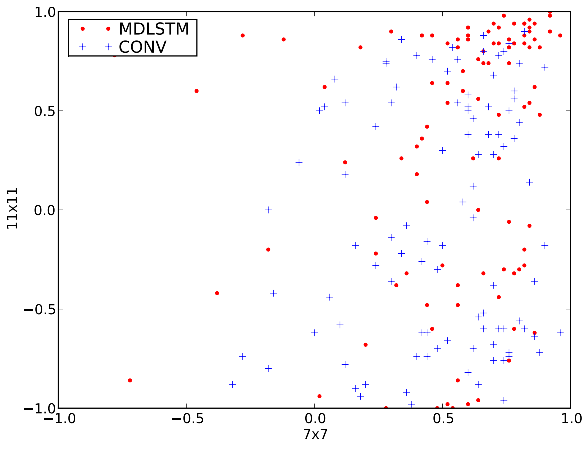


(b) Heuristic opponent with $\epsilon = 0.05$.

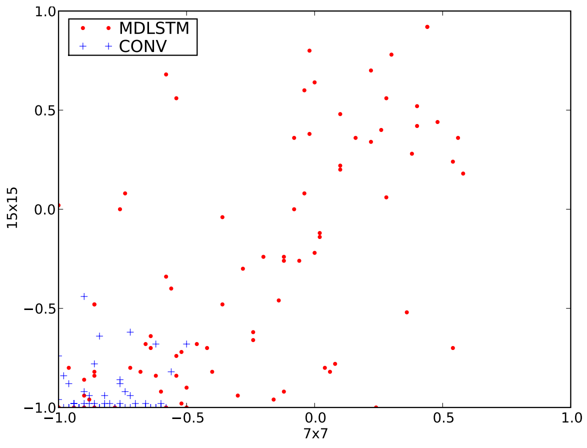
Figure 4.5: Learning curves for training against *heuristic* opponents of different strength, on 7x7 Atari-Go (averaged over 10 independent runs). The solid line corresponds to the average fitness per generation, the broken one corresponds to the best fitness per generation. MDLSTMs clearly outperform the other two architectures.



(a) Random networks, naive opponent.



(b) Optimized networks, naive opponent.



(c) Optimized networks, heuristic opponent with $\epsilon = 0.2$.

Figure 4.6: Illustrations of fitness scalability on Atari-Go. Points correspond to MDLSTMs, crosses to convolutional networks. (a) Performance is correlated for random networks, and often higher on 11x11 than 7x7, at least for MDLSTMs. (b) In contrast to the large number of CONVs on the bottom right, for which good performance on 7x7 corresponds to poor performance on 11x11, performance is correlated for MDLSTMs. (c) Among networks trained against the heuristic player on 7x7, only MDLSTMs achieve good results, and those results scale well to 15x15.

parasites (according to shared sampling), and 10 players out of the hall of fame (which contains the player with the best relative fitness of each generation, i.e., the generation champions). Both populations exchange roles at each generation. We use the implementation found in PyBrain (see appendix A).

Coevolution implies that there is no external fitness measure (as in section 4.4.3), only a relative fitness. We compute the relative fitness of two players as the average score over 100 games, with alternating starting players. In order to make it more informative than pure win/lose, we compute the score for a single game as follows:

$$\text{score} = \begin{cases} 1 - \beta \frac{M - M_{min}}{M_{max} - M_{min}} & \text{if game won} \\ 0 & \text{if draw} \\ -1 + \beta \frac{M - M_{min}}{M_{max} - M_{min}} & \text{if game lost} \end{cases},$$

with $\beta = 0.2$ being the game-length trade-off, M the number of moves done before the game is over, M_{min} the length of the shortest game and M_{max} the length of the longest game possible. Apart from smoothing the fitness landscape, this choice favors networks that either exploit the opponents mistakes quickly, or avoid making mistakes themselves.

Among the possible ways for tracking progress in coevolutionary runs – in the absence of an absolute fitness – are dominance tournaments (Stanley and Miikkulainen, 2004a). In a dominance tournament, a list of dominant individuals is maintained. The first dominant individuals is the first generation champion, and for every other generation champion, we add it to the list if it can beat all previous dominant individuals. The length of that list then gives us the *dominance number*, and it is reasonable to consider that the higher that number is, the more progress has been made. Table 4.3 shows average dominance numbers for a number of different training scenarios.

Table 4.3: Dominance numbers for two of the games, and two different coevolutionary setups (400 generations, averaged over 5 runs). Elitism helps, and there is more progress for Atari-Go.

Game	Parameters	Dominance number
Atari-Go	Non-elitist	14.8 ± 3
Atari-Go	Elitist	27.6 ± 14
Go-Moku	Non-elitist	11.0 ± 5
Go-Moku	Elitist	15.6 ± 11

Figure 4.7 shows a typical coevolution run. The performance plotted is the one measured against the naive player (inaccessible during coevolution). It is interesting to note that this performance is not strictly increasing, and that even the champions of the dominance tournament do not always correspond to high absolute performance. Nonetheless, the high dominance numbers show progress, which indicates that evolution based on relative fitness is producing

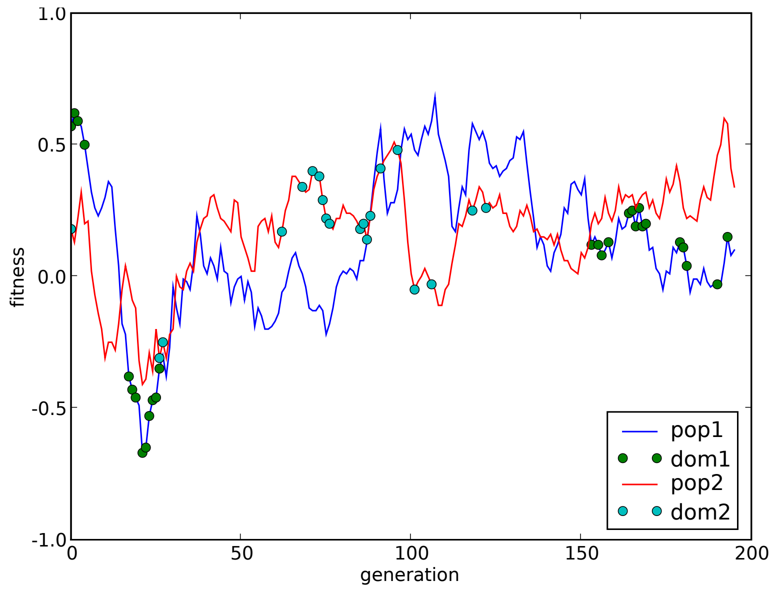


Figure 4.7: **A typical coevolutionary run.** Plotted is the absolute fitness of the generation champions of both populations against the naive player. The circles mark the dominant individuals (i.e. that beat all previous dominant ones).

very different kinds of strategies than directly optimizing absolute fitness (as in section 4.4.3).

To visualize the relative progress of the two populations during a coevolutionary run, we use a CIAO plot (Cliff and Miller, 1995), which show the performance of all generation champions of one population (horizontal axis) playing against all of those of the other (vertical axis). If coevolution is successful, we expect later generations to be better against earlier ones, thus to see brighter areas in the lower left and upper right. Figure 4.8 shows a typical CIAO plot, which exhibits this property to a small degree. However, we can make two other, unpredicted, observations:

- the score values themselves are more extreme (i.e. the games are shorter) with players of earlier generations and more balanced in later generations (more grayish colors). This means that coevolution tends to lead to more careful players, which lose late if they do, but at the cost of not being able to win quickly against each other.

- carefully looking at individual lines in the plot, we find that often, if one player that is winning against one group of opponents and another player is losing to that group, then there is another group of opponents where those relations are exactly opposite. This seems to indicate a kind of rock-paper-scissors situation, which could be the reason why the absolute level of play does not progress as much as expected during coevolution.

Even with few weights, the MDRNN is capable of a wide *variety* of behaviors. When inspecting the generation champions of a coevolutionary run, we find a

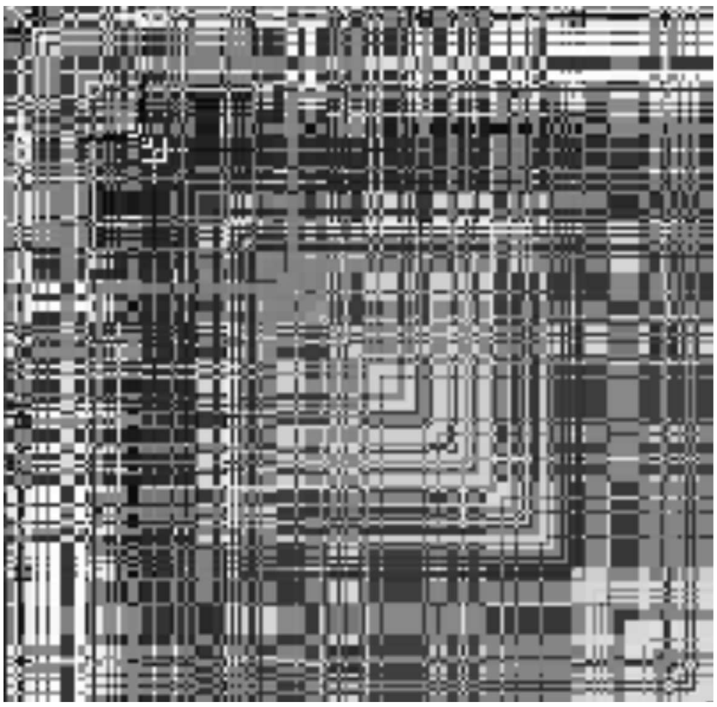


Figure 4.8: CIAO plot of the champions of both populations (one on the vertical axis, one on the horizontal one) during a typical run (the same run than Figure 4.7). Bright points correspond to a high score, dark ones to a low one. In successful coevolution, later generations (right/down) beat earlier ones (left/up), which is visible in the brighter areas in the lower left and upper right of the plot.

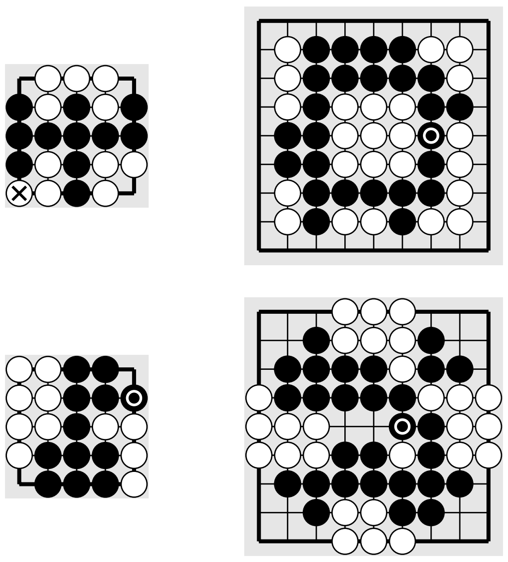


Figure 4.9: **Illustrative games of some generation champions.** The last move is marked by a circle for winning and cross for losing. Above: both players avoid traps and corners in 5x5, which when scaled to the 9x9 board becomes border-avoidance. Below, black is the same player than above, but white is a different champion, leading to very different games.

large diversity of those behaviors. Figure 4.9 shows a few illustrative games, with the same players (generation champions of a typical coevolutionary run) on board sizes 5x5 and 9x9. Naturally the behavioral patterns look different on a small board and on a large one, but they share common features.

4.5 Discussion

MDRNNs are scalable because they are approximately translation-invariant, in addition to capturing the board symmetries. They handle the same situation on a bigger board (with empty positions around it) in the exact same way as on a smaller board. This allows them to use a comparatively low number of weights (MDRNNs and MDLSTMs with $k = 3$ have 21 and 93 weights respectively, independently of board size), thereby reducing the dimensionality of the search space and making optimization efficient. Incorporating LSTM cells in the swiping layer further enables the architecture to better handle long-distance context, which is necessary for learning complex strategies, as our experiments with the heuristic opponent show. The results show that MDLSTMs transfer the strategies learned on small boards to large ones, leading to a level of play on 15x15 that is on par with human beginners. Finally, due to the generality of our approach and the similarity of our three games to Go, we expect our results to carry over the game of Go itself.

Chapter 5

Conclusions

Epigraphs are pretentious,
especially self-referential ones.

Anonymous

In conclusion, we addressed the two questions that we set out to answer in section 1.6:

> *Can we design a family of algorithms that are derived from first principles, which reach state-of-the-art performance, while at the same time being flexible enough to permit variants for a whole range of problem classes?*

In chapter 2, we introduced Natural Evolution Strategies, a novel family of algorithms that constitutes a principled approach to black-box optimization. Maintaining a parameterized distribution on the set of solution candidates, the natural gradient is used to update the distribution's parameters in the direction of higher expected fitness. A collection of techniques have been introduced that addresses issues of convergence, robustness, computational complexity, sensitivity to hyperparameters and algorithm speed. We investigated a number of instantiations of the NES family ranging from general-purpose multi-variate normal distributions to heavy-tailed and separable distributions specialized in global optimization and high dimensionality, respectively. The results show best published performance on various standard benchmarks, as well as competitive performance on others.

> *Can artificial curiosity be an effective guiding principle for determining the most informative search points in costly optimization?*

In chapter 3 we demonstrated how the principle of artificial curiosity can guide exploration in the context of costly optimization. We introduced a response surface method, which estimates the interestingness of each candidate point using Gaussian process regression. In addition, simultaneously optimizing the objective of expected information gain and the objective of expected improvement makes the exploration-exploitation trade-off explicit, and permits maximally informed search point selection.

As an additional study, in chapter 4 we applied black-box optimization to the challenging problem of leaning to play the game of Go. We trained multi-dimensional recurrent neural networks to play on small boards, and showed that the learned skills translate to good quality of play on large boards as well.

5.1 Perspectives

Naturally, for every answered question, many new ones come to light; so here we list a few thoughts for extensions and future work.

Linkage learning. One NES variant could operate with *sparse* covariance matrices, where the sparse entries are learned. This would involve two theoretical developments. First, we could define a greedy procedure over randomly sampled parameter-pairs to determine which ones to add to the matrix. The most efficient procedure will involve collecting correlation statistics from a pool of candidate pairs before they are added, because clearly, in high dimensions we can not investigate all possible pairs. Also, we would periodically revisit the existing sparse entries and eliminate the least relevant ones. Second, we could analytically derive the exact Fisher information matrix for sparse matrices, which is needed to compute the natural gradient that NES follows. This approach would allow us to effectively exploit structure in black-box problems without relying on any domain knowledge. For example, on neuro-evolution problems, we would expect to approximately recover the neuron-centered block-structure (Gomez et al., 2008).

Low-rank approximations. A different NES extension with the same goal of computational efficiency would be to use a *low-rank* approximation of the covariance matrix, that is, with a small number of predominant search axes; the rest of the dimensions are searched using mutations from a diagonal matrix. Such a parametrization would allow the algorithm to follow low-dimensional 'valleys' embedded in high-dimensional space, enabling optimization on highly non-separable problems, with a computational complexity that remains linear in the number of dimensions.

Search distributions on discrete domains. It is important to realize that the NES framework is not limited to continuous domains. In fact, NES can be designed to directly operate on discrete search spaces, for example trees, graphs or grammars. Program space constitutes a discrete search space of particular interest, and it seems promising to extend the NES approach to *program search* (genetic programming, Koza, 1992), building upon probabilistic representations of program trees (e.g., Bosman and Jong, 2004; Salustowicz and Schmidhuber, 1997) for which the natural gradient can be estimated.

Stronger Go players. We can identify a number of possible avenues for attaining stronger results on training neural networks to play Go. Directly training against a stronger heuristic opponent (e.g. $\epsilon = 0.0$, see chapter 4) is like finding a needle in a haystack, as most initial networks will lose all games. So, future work could address this issue by training against incrementally harder opponents (Gomez and Miikkulainen, 1997). Also, we expect to eventually reach a limit on the complexity of strategies that MDLSTMs can represent. For this case we propose increasing their expressive power by stacking two or more MDLSTMs on top of each other, the lower one producing a map of high-level features that the top one scans over. Clearly, another viable extension is to incorporate domain knowledge, e.g. by feeding the network a number of domain-specific features (Wu and Baldi, 2007) instead of the raw board.

5.2 Closing Words

Now, at the end of this dissertation, I would like to take the opportunity to thank *you*, the reader: The thought that someone will read until the end helped me persevere in writing it. I hope you found something of value among these thoughts, in the best case possibly even inspiring a new idea of your own!

Appendix A

Appendix: Implementations in Python

Virtually all the code implemented in the process of this work is available within the open-source machine learning library *PyBrain* (Schaul et al., 2010). In order to make our research as reproducible as possible, we wanted to make the code available to the community, and we found the most effective way for this to be the integration into an open-source library (that I am a core developer of), along with documentation and code examples.

PyBrain is a machine learning library written in Python, designed to facilitate both the application of and research on premier learning algorithms. PyBrain is implemented in Python, with the scientific library SciPy being its only strict dependency. As is typical for programming in Python/SciPy, development time is greatly reduced as compared to languages such as Java/C++, at the cost of lower speed. PyBrain embodies a compositional setup, which means that it is designed to be able to connect various types of architectures and algorithms.

PyBrain goes beyond existing Python libraries in breadth, in that it provides a toolbox for supervised, unsupervised and reinforcement learning as well as black-box and multi-objective optimization. In addition to standard algorithms (some of which, to the best of our knowledge, are not available as Python implementations elsewhere) for application-oriented users, it contains reference implementations of a number of algorithms at the bleeding edge of research. Furthermore, it sets itself apart by its flexibility for composing custom neural networks architectures, ranging from (multi-dimensional) recurrent networks to restricted Boltzmann machines or convolutional networks.

Table A.1 gives pointers to the implementations corresponding to the work presented in the previous chapters. The documentation, tutorials and more information can be found on `www.pybrain.org`. The code itself is available through github at `github.com/pybrain/pybrain`. Finally, the PyBrain community communicates mainly through the mailing list, where many common questions have been answered already (`groups.google.com/group/pybrain`).

Table A.1: PyBrain implementations of algorithms, neural networks, benchmarks and tools used in this dissertation, as well as a number other black-box optimization algorithms.

Name	Code file(s) in `pybrain.*`	Mentioned in chapter(s)
GA	`optimization.populationbased.ga`	1
ES	`optimization.populationbased.es`	2
PSO	`optimization.populationbased.pso`	1
FEM	`optimization.distributionbased.fem`	1
CMA-ES	`optimization.distributionbased.cmaes`	1, 2, 4
xNES	`optimization.distributionbased.xnes`	2, 3, 4
Coevolution	`optimization.populationbased.coevolution.*`	4
NSGA-II	`optimization.populationbased.multiobjective.nsga2`	3
RNN	`structure.networks.recurrent`	2, 4
convolutional networks	`structure.networks.convolutional`	4
LSTM	`structure.modules.lstm`	4
MDRNN	`structure.networks.multidimensional`	4
MDLSTM	`structure.networks.custom.*`	4
BBOB benchmarks	`rl.environments.functions.bbob2010`	2
pole balancing	`rl.environments.cartpole.*`	2
Lennard-Jones	`rl.environments.functions.lennardjones`	2
Branin benchmark	`rl.environments.functions.multimodal`	3
Atari-Go	`rl.environments.twoplayergames.capturegame`	4
Go-Muko	`rl.environments.twoplayergames.gomuko`	4
Pente	`rl.environments.twoplayergames.pente`	4
importance mixing	`auxiliary.importancemixing`	2
Gaussian processes	`auxiliary.gaussprocess`	3
plotting tools	`tools.plotting.*`	

Appendix B

Appendix: Weighted Mann-Whitney Test

This appendix defines a *weighted Mann-Whitney* test, as used in section 2.4.4 for virtual adaptation sampling. Although the derivations are trivial, we are not aware of any previous attempt to create a test with the same purpose.

Background. The classical Mann-Whitney test determines (with confidence ρ) whether two sets of samples $S = \{s_i\}$ and $S' = \{s'_i\}$ are likely to come from the same distribution. For that, the so-called U-statistic is computed:

$$U = \sum_{s_i > s'_j} 1 + \sum_{s_i = s'_j} \frac{1}{2}$$

Let $\mu = \frac{nn'}{2}$ and $\sigma = \sqrt{\frac{nn'(n+n'+1)}{12}}$, where n and n' are the number of samples in S and S', respectively. We can then determine the significance of the difference between S and S'. They are different with confidence ρ if:

- $\Phi(\frac{U-\mu}{\sigma}) > 1 - \rho$ (if S has larger values), or
- $\Phi(\frac{U-\mu}{\sigma}) < \rho$ (if S' has larger values).

Introducing Weights. Now, assume that every sample in S and S' has a (positive) weight (w_i or w'_i) associated to it. We can generalize the Mann-Whitney test by interpreting the weights as fractional number of occurrences in the sets:

$$U = \sum_{s_i > s'_j} w_i w'_j + \sum_{s_i = s'_j} \frac{1}{2} w_i w'_j$$

Accordingly, we also need to adjust the number of samples: $m = \sum_{i=1}^{n} w_i$ and $m' = \sum_{i=1}^{n'} w'_i$, and thus $\mu = \frac{mm'}{2}$ and $\sigma = \sqrt{\frac{mm'(m+m'+1)}{12}}$.

We can see this weighted U-statistic as an interpolation between the cases covered by the classical one. In fact, if the weights are integers, a sample s with weight w can be replaced equivalently by w occurrences of the same sample s (each with weight 1).

Bibliography

Y. Akimoto, Y. Nagata, I. Ono, and S. Kobayashi. Bidirectional Relation between CMA Evolution Strategies and Natural Evolution Strategies. In *Parallel Problem Solving from Nature (PPSN)*, 2010. (Cited on pages 32 and 34.)

S. Amari. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, 1998. (Cited on page 16.)

S. Amari and S. C. Douglas. Why natural gradient? In *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '98)*, volume 2, pages 1213–1216, 1998. (Cited on page 16.)

C. Andrieu, N. D. Freitas, A. Doucet, and M. I. Jordan. An Introduction to MCMC for Machine Learning. *Machine Learning*, 50(1):5–43, 2003. (Cited on page 9.)

A. Auger. Convergence results for the (1,λ)-SA-ES using the theory of ϕ-irreducible Markov chains. *Theoretical Computer Science*, 334(1-3):35 – 69, 2005. (Cited on page 9.)

A. Auger, S. Finck, N. Hansen, and R. Ros. BBOB 2010: Comparison Tables of All Algorithms on All Noisy Functions. Technical Report RT-389, INRIA, 09 2010. (Cited on page 4.)

J. A. Bagnell and J. Schneider. Covariant policy search. In *Proceedings of the 18th international joint conference on Artificial intelligence*, pages 1019–1024, San Francisco, CA, USA, 2003. Morgan Kaufmann Publishers Inc. (Cited on page 59.)

P. Baldi and G. Pollastri. The principled design of large-scale recursive neural network architectures DAG-RNNs and the protein structure prediction problem. *Journal of Machine Learning Research*, 4:575–602, 2003. (Cited on page 77.)

P. Bayer, C. M. Bürger, and M. Finkel. Solving computationally-demanding reliability-based design problems in hydrogeology. In *Proceedings 6th Int. Conf. on Calibration and Reliability in Groundwater Modeling - Credibility in Modeling*, pages 22–26. International Association of Hydrological Sciences, 2007. (Cited on page 8.)

H.-G. Beyer. *The theory of evolution strategies*. Springer-Verlag New York, Inc., New York, NY, USA, 2001. ISBN 3-540-67297-4. (Cited on page 9.)

H.-G. Beyer and H.-P. Schwefel. Evolution strategies: A comprehensive introduction. *Natural Computing*, 1:3–52, 2002. (Cited on page 7.)

A. Booker, J. J. Dennis, P. Frank, and D. Serafini. Optimization using surrogate objectives on a helicopter test example. *Computational Methods in Optimal Design and Control*, 1998. (Cited on page 8.)

P. Bosman, J. Grahl, and D. Thierens. Enhancing the Performance of Maximum-Likelihood Gaussian EDAs Using Anticipated Mean Shift. In *Parallel Problem Solving from Nature – PPSN X*, pages 133 – 143, 2008. (Cited on page 69.)

P. A. Bosman and E. D. D. Jong. Learning Probabilistic Tree Grammars for Genetic Programming. In *Parallel Problem Solving from Nature - PPSN VIII*, volume 3242 of *Lecture Notes in Computer Science*, pages 192–201, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. (Cited on page 93.)

G. E. P. Box and K. B. Wilson. On the Experimental Attainment of Optimum Conditions. *Journal of the Royal Statistical Society*, 13(1):1–45, 1951. (Cited on pages 7 and 8.)

C. J. F. T. Braak. A Markov Chain Monte Carlo version of the genetic algorithm Differential Evolution: easy Bayesian computing for real parameter spaces. *Statistics and Computing*, 16(3):239–249, 2006. (Cited on page 9.)

É. Cartan. Sur la représentation géométrique des systèmes matérieles non holonomes. In *Proc Int Congr Math, Bologna*, volume 4, pages 253–261, 1928. (Cited on page 29.)

K. Chaloner and I. Verdinelli. Bayesian experimental design: A review. *Statistical Science*, 10:273–304, 1995. (Cited on pages 63 and 64.)

C. Charbuillet, B. Gas, M. Chetouani, and J. Zarader. Optimizing feature complementarity by evolution strategy: Application to automatic speaker verification. *Speech Communication*, 51(9):724 – 731, 2009. Special issue on non-linear and conventional speech processing - NOLISP 2007. (Cited on page 8.)

D. Cliff and G. F. Miller. Tracking the red queen: Measurements of adaptive progress in co-evolutionary simulations. In *Advances In Artificial Life*, pages 200–218. Springer Verlag, 1995. (Cited on page 88.)

D. A. Cohn. Neural network exploration using optimal experiment design. In *Advances in Neural Information Processing Systems*, pages 679–686, 1994. (Cited on page 63.)

D. A. Cohn, Z. Ghahramani, and M. I. Jordan. Active learning with statistical models. *Journal of Artificial Intelligence Research*, 4:129–145, 1995. (Cited on page 65.)

L. Csató and M. Opper. Sparse on-line gaussian processes. *Neural Computation*, 14, 2002. (Cited on page 68.)

C. Darwin. *On the origin of species by means of natural selection*, volume 146. John Murray, 1859. (Cited on page 7.)

A. A. de Moura Meneses, C. J. G. Pinheiro, P. Rancoita, T. Schaul, L. M. Gambardella, R. Schirru, R. C. Barroso, and L. F. de Oliveira. Assessment of neural networks training strategies for histomorphometric analysis of synchrotron radiation medical images. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 2010. (Cited on page xx.)

K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multi-objective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6:182 – 197, 2002. (Cited on pages 4, 64, and 68.)

M. P. Deisenroth, C. E. Rasmussen, and J. Peters. Gaussian process dynamic programming. *Neurocomputing*, pages 1508–1524, 2009. (Cited on page 62.)

V. V. Fedorov. *Theory of optimal experiments*. Academic press, New York, 1972. (Cited on page 63.)

R. Fletcher. *Practical Methods of Optimisation, (2nd Ed.)*. John Wiley, 1987. (Cited on page 6.)

B. Freisleben and H. Luttermann. Learning to Play the Game of Go-Moku: A Neural Network Approach. *Australian Journal of Intelligent Information Processing Systems, Vol. 3, No. 2*, pages 52 – 60, 1996. (Cited on pages 75 and 76.)

F. Friedrichs and C. Igel. Evolutionary tuning of multiple SVM parameters. *Neurocomputing*, 64:107–117, 2005. (Cited on pages 7 and 8.)

J. Gauci and K. Stanley. Generating large-scale neural networks through discovering geometric regularities. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 997–1004, 2007. (Cited on page 77.)

S. Gherman. Atari-Go Applet in Java, 2000. URL http://www.361points.com/capturego/. (Cited on page 76.)

T. Glasmachers and C. Igel. Gradient-based Adaptation of General Gaussian Kernels. *Neural Computation*, 17(10):2099–2105, 2005. (Cited on page 28.)

T. Glasmachers, T. Schaul, and J. Schmidhuber. A Natural Evolution Strategy for Multi-Objective Optimization. In *Parallel Problem Solving from Nature (PPSN)*, 2010a. (Cited on pages xviii, 32, and 38.)

T. Glasmachers, T. Schaul, Y. Sun, D. Wierstra, and J. Schmidhuber. Exponential Natural Evolution Strategies. In *Genetic and Evolutionary Computation Conference (GECCO)*, Portland, OR, 2010b. (Cited on page xviii.)

D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989. ISBN 0201157675. (Cited on pages 7 and 35.)

F. Gomez and R. Miikkulainen. Incremental Evolution of Complex General Behavior. *Adaptive Behavior*, 5:317–342, 1997. (Cited on page 93.)

F. Gomez, J. Schmidhuber, and R. Miikkulainen. Accelerated Neural Evolution through Cooperatively Coevolved Synapses. *Journal of Machine Learning Research*, 2008. (Cited on pages 8, 53, 54, and 92.)

A. Graves. *Supervised Sequence Labelling with Recurrent Neural Networks*. Ph.D. in Informatics, Fakultat für Informatik – Technische Universität München, 2008. (Cited on pages 74, 77, and 78.)

A. Graves and J. Schmidhuber. Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks. In *NIPS*, 2008. (Cited on page 77.)

A. Graves, S. Fernández, and J. Schmidhuber. Multidimensional Recurrent Neural Networks. In *Proceedings of the 2007 International Conference on Artificial Neural Networks*, September 2007. (Cited on pages 74, 77, and 78.)

V. Graziano, T. Glasmachers, T. Schaul, L. Pape, G. Cuccu, J. Leitner, and J. Schmidhuber. Artificial Curiosity for Autonomous Space Exploration. *Acta Futura (in press)*, 2011. (Cited on page xx.)

M. Grüttner. Evolving Multidimensional Recurrent Neural Networks for the Capture Game in Go. Bachelor thesis, Techniche Universität München, 2008. (Cited on page 76.)

M. Grüttner, F. Sehnke, T. Schaul, and J. Schmidhuber. Multi-Dimensional Deep Memory Atari-Go Players for Parameter Exploring Policy Gradients. In *International Conference on Artificial Neural Networks (ICANN)*, 2010. (Cited on page xx.)

N. Hansen and A. Auger. Real-parameter black-box optimization benchmarking 2010: Experimental setup, 2010. (Cited on pages 47 and 48.)

N. Hansen and S. Finck. Real-parameter black-box optimization benchmarking 2010: Noiseless functions definitions, 2010a. (Cited on pages 47, 48, and 56.)

N. Hansen and S. Finck. Real-Parameter Black-Box Optimization Benchmarking 2010: Noisy Functions Definitions, 2010b. (Cited on page 47.)

N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001. (Cited on pages 7, 20, 32, 33, 34, 35, 46, 59, and 79.)

N. Hansen, A. S. P. Niederberger, L. Guzzella, and P. Koumoutsakos. A method for handling uncertainty in evolutionary optimization with an application to feedback control of combustion. *IEEE Transactions on Evolutionary Computation*, 13:180–197, 2009. (Cited on page 8.)

M. Hasenjäger, B. Sendhoff, T. Sonoda, and T. Arima. Three dimensional evolutionary aerodynamic design optimization with CMA-ES. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, GECCO '05, pages 2173–2180, New York, NY, USA, 2005. ACM. (Cited on page 8.)

W. K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57(1):97–109, 1970. (Cited on page 6.)

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(9):1735–1780, 1997. (Cited on page 78.)

J. H. Holland. *Adaptation in natural and artificial systems.* MIT Press, Cambridge, MA, USA, 1992. ISBN 0-262-58111-6. (Cited on page 7.)

J.-N. Hwang, J. J. Choi, S. Oh, and R. J. I. Marks. Query-based learning applied to partially trained multilayer perceptrons. *IEEE Transactions on Neural Networks*, 2:131–136, 1991. (Cited on page 63.)

O. Ibáñez, L. Ballerini, O. Cordón, S. Damas, and J. Santamar'ia. An Experimental Study on the Applicability of Evolutionary Algorithms to Craniofacial Superimposition in Forensic Identification. *Information Sciences*, 179(23): 3998–4028, 2009. (Cited on page 8.)

C. Igel and M. Husken. Empirical evaluation of the improved Rprop learning algorithm. *Neurocomputing*, 50:2003, 2003. (Cited on page 25.)

C. Igel, N. Hansen, and S. Roth. Covariance Matrix Adaptation for Multi-objective Optimization. *Evolutionary Computation*, 15(1):1–28, 2007. (Cited on pages 8, 37, and 38.)

L. Itti and P. Baldi. Bayesian surprise attracts human attention. In Y. W. Platt, B. Schölkopf, and J., editors, *Advances in Neural Information Processing Systems*, pages 547—-554. MIT Press, 2006. (Cited on page 62.)

K. Iwamoto. *Go for beginners*. Ishi Press, 1972. (Cited on pages 74 and 75.)

G. A. Jastrebski and D. V. Arnold. Improving Evolution Strategies through Active Covariance Matrix Adaptation. In *IEEE Congress on Evolutionary Computation*, 2006. (Cited on page 46.)

M. Jebalia, A. Auger, M. Schoenauer, F. James, and M. Postel. Identification of the isotherm function in chromatography using CMA-ES. In *IEEE Congress on Evolutionary Computation*, pages 4289–4296, 2007. (Cited on page 8.)

M. Jebalia, A. Auger, and N. Hansen. Log-linear convergence and divergence of the scale-invariant (1+1)-ES in noisy environments. *Algorithmica*, pages 1–36, 2010. online first. (Cited on page 9.)

D. R. Jones. A taxonomy of global optimization methods based on response surfaces. *Journal of Global Optimization*, 21:345–383, 2001. (Cited on pages 8, 64, 70, and 71.)

D. R. Jones, M. Schonlau, and W. J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13:455–492, 1998. (Cited on pages 8, 68, and 70.)

S. Kakade. A natural policy gradient. *Advances in Neural Information Processing Systems 14*, 2:1531–1538, 2002. (Cited on page 59.)

J. H. Kämpf and D. Robinson. A hybrid CMA-ES and HDE optimisation algorithm with application to solar energy potential. *Applied Soft Computing*, 9(2):738–745, 2009. (Cited on page 8.)

J. Kennedy and R. Eberhart. *Swarm Intelligence*. Morgan Kaufmann, San Francisco, CA, 2001. (Cited on page 7.)

S. Kirkpatrick, C. D. Gelatt, Jr, and M. P. Vecchi. Optimization by Simulated Annealing. *Science*, 220:671–680, 1983. (Cited on page 6.)

J. Klockgether and H. P. Schwefel. Two-phase nozzle and hollow core jet experiments. In *Proc. 11th Symp. Engineering Aspects of Magnetohydrodynamics*, pages 141–148, 1970. (Cited on page 8.)

J. Knowles, R. Watson, and D. Corne. *Evolutionary Multi-Criterion Optimization*, volume 1993 of *Lecture Notes in Computer Science*. Springer, 2001. (Cited on page 4.)

G. Konidaris, D. Shell, and N. Oren. Evolving Neural Networks for the Capture Game. In *Proceedings of the SAICSIT Postgraduate Symposium*, 2002. (Cited on page 75.)

J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, 1992. (Cited on page 93.)

A. Krause and C. Guestrin. Nonmyopic active learning of gaussian processes: An exploration-exploitation approach. In *Proceedings of the International Conference on Machine Learning*, 2007. (Cited on pages 9 and 66.)

S. Kullback and R. A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951. (Cited on page 17.)

P. Larrañaga. *Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation*, chapter An introduction to probabilistic graphical models, pages 25–54. Kluwer Academic Publishers, 2002. (Cited on page 7.)

Y. Lecun and Y. Bengio. *Convolutional Networks for Images, Speech and Time Series*, pages 255–258. The MIT Press, 1995. (Cited on page 76.)

J. Leitner, C. Ampatzis, and D. Izzo. Evolving anns for spacecraft rendezvous and docking. In *10th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS)*, August 2010. (Cited on page 8.)

M. Locatelli. Bayesian algorithms for one-dimensional global optimization. *Journal of Global Optimization*, pages 57–76, 1997. (Cited on page 70.)

A. Lubberts and R. Miikkulainen. Co-Evolving a Go-Playing Neural Network. In *Genetic and Evolutionary Computation Conference Workshop Program*, pages 14–19. Morgan Kaufmann, 2001. (Cited on page 84.)

D. J. C. MacKay. Information-based objective functions for active data selection. *Neural Computation*, 4:550–604, 1992. (Cited on page 63.)

M. Matsumoto and T. Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions On Modeling And Computer Simulation*, 8(1):3–30, 1998. (Cited on page 56.)

S. Mendes, J. Gomez Pulido, M. Vega Rodriguez, M. Jaraiz Simon, and J. Sanchez Perez. A Differential Evolution Based Algorithm to Optimize the Radio Network Design Problem. *2006 Second IEEE International Conference on eScience and Grid Computing eScience06*, 2(1):4–9, 2006. (Cited on page 8.)

N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of State Calculations by Fast Computing Machines. *The Journal of Chemical Physics*, 21(6):1087–1092, 1953. (Cited on page 6.)

E. Miguez, E. Diaz-Dorado, and J. Cidras. An application of an evolution strategy in power distribution system planning. In *IEEE International Conference on Evolutionary Computation Proceedings*, pages 241 –246, may 1998. (Cited on page 8.)

E. Minisci and G. Avanzini. Comparative study on the application of evolutionary optimization techniques to orbit transfer maneuvers. In *IAC 2008*, September 2008. (Cited on page 8.)

A. W. Moore and J. Schneider. Memory-based stochastic optimization. In *Advances in Neural Information Processing Systems*, 1996. (Cited on page 7.)

S. D. Muller, J. Marchetto, S. Airaghi, and P. Koumoutsakos. Optimization based on bacterial chemotaxis. *IEEE Transactions on Evolutionary Computation*, 6:6–16, 2002. (Cited on page 7.)

I. Najfeld and T. F. Havel. Derivaties of the Matrix Exponential and Their Computation. *Adv. Appl. Math*, 16:321–375, 1994. (Cited on page 29.)

J. A. Nelder and R. Mead. A Simplex Method for Function Minimization. *The Computer Journal*, 7(4):308–313, 1965. (Cited on page 6.)

X. Pang and P. J. Werbos. Neural Network Design for J Function Approximation. In *in Dynamic Programming, Math. Modelling and Scientific Computing (a Principia Scientia journal*, 1996. (Cited on page 76.)

J. Peters. *Machine Learning of Motor Skills for Robotics*. PhD thesis, epartment of Computer Science, University of Southern California, 2007. (Cited on page 19.)

J. Peters and S. Schaal. Natural actor-critic. *Neurocomputing*, 71(7-9):1180–1190, 2008. (Cited on page 59.)

T. Pfingsten. Bayesian active learning for sensitivity analysis. In *Machine Learning: ECML 2006*, pages 353 – 364, 2006. (Cited on page 62.)

M. Plutowski, G. Cottrell, and H. White. Learning Mackey-Glass from 25 Examples, Plus or Minus 2. In *Advances in Neural Information Processing Systems*, pages 1135–1142, 1994. (Cited on page 63.)

C. E. Rasmussen and C. K. I. Williams. *Gaussian processes for machine learning*. MIT Press, 2006. (Cited on pages 8, 66, 67, and 68.)

I. Rechenberg and M. Eigen. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog Stuttgart, 1973. (Cited on pages 7 and 35.)

N. Richards, D. E. Moriarty, and R. Miikkulainen. Evolving neural networks to play Go. *Applied Intelligence*, 8:85–96, 1997. (Cited on page 74.)

M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *IEEE International Conference on Neural Networks*, pages 586–591. IEEE Press, 1993. (Cited on page 25.)

M. B. Ring and T. Schaul. Q-error as a Selection Mechanism in Modular Reinforcement-Learning Systems. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 2011. (Cited on page xx.)

R. Ros and N. Hansen. A Simple Modification in CMA-ES Achieving Linear Time and Space Complexity. In R. et al., editor, *Parallel Problem Solving from Nature, PPSN X*, pages 296–305. Springer, 2008. (Cited on pages 39 and 46.)

C. D. Rosin and R. K. Belew. Methods for Competitive Co-Evolution: Finding Opponents Worth Beating. In L. J. Eshelman, editor, *Proceedings of the Sixth International Conference on Genetic Algorithms*. San Francisco, CA: Morgan Kaufmann, 1995. (Cited on page 84.)

R. Y. Rubinstein and D. P. Kroese. *The Cross-Entropy Method: A Unified Approach to Combinatorial Optimization, Monte-Carlo Simulation and Machine Learning (Information Science and Statistics)*. Springer, 2004. (Cited on page 7.)

T. Rückstieß, F. Sehnke, T. Schaul, D. Wierstra, Y. Sun, and J. Schmidhuber. Exploring Parameter Space in Reinforcement Learning. *Paladyn Journal of Behvioral Robotics*, 1(1):14–24, 2010. (Cited on page xviii.)

T. P. Runarsson and S. M. Lucas. Co-evolution versus Self-play Temporal Difference Learning for Acquiring Position Evaluation in Small-board Go. *IEEE Transactions on Evolutionary Computation*, pages 628–640, 2005. (Cited on page 74.)

R. P. Salustowicz and J. Schmidhuber. Probabilistic Incremental Program Evolution. *Evolutionary Computation*, 5:123–141, 1997. (Cited on page 93.)

T. Schaul and J. Schmidhuber. A Scalable Neural Network Architecture for Board Games. In *IEEE Symposium on Computational Intelligence in Games (CIG)*, 2008. (Cited on page xix.)

T. Schaul and J. Schmidhuber. Scalable Neural Networks for Board Games. In *International Conference on Artificial Neural Networks (ICANN)*, 2009. (Cited on page xix.)

T. Schaul and J. Schmidhuber. Metalearning. *Scholarpedia*, 5(6):4650, 2010a. (Cited on pages xx and 24.)

T. Schaul and J. Schmidhuber. Towards Practical Universal Search. In *Conference on Artificial General Intelligence (AGI)*, Lugano, 2010b. (Cited on pages xviii and 25.)

T. Schaul, J. Bayer, D. Wierstra, Y. Sun, M. Felder, F. Sehnke, T. Rückstieß, and J. Schmidhuber. PyBrain. *Journal of Machine Learning Research*, 11: 743–746, 2010. (Cited on pages xix and 95.)

T. Schaul, T. Glasmachers, and J. Schmidhuber. High Dimensions and Heavy Tails for Natural Evolution Strategies. In *Genetic and Evolutionary Computation Conference (GECCO)*, Dublin, Ireland, 2011a. (Cited on page xviii.)

T. Schaul, L. Pape, T. Glasmachers, V. Graziano, and J. Schmidhuber. Coherence Progress: A Measure of Interestingness Based on Fixed Compressors. In *Fourth Conference on Artificial General Intelligence (AGI)*, 2011b. (Cited on page xix.)

T. Schaul, Y. Sun, D. Wierstra, F. Gomez, and J. Schmidhuber. Curiosity-Driven Optimization. In *IEEE Congress on Evolutionary Computation (CEC)*, 2011c. (Cited on page xix.)

J. Schmidhuber. Curious model-building control systems. *IEEE International Joint Conference on Neural Networks*, pages 1458 – 1463, 1991. (Cited on pages 10, 61, and 62.)

J. Schmidhuber. Developmental robotics, optimal artificial curiosity, creativity, music, and the fine arts. *Connection Science*, 18:173–187, 2006. (Cited on pages 61 and 62.)

J. Schmidhuber. Simple algorithmic principles of discovery, subjective beauty, selective attention, curiosity and creativity. *Lecture Notes In Artificial Intelligence*, 4754, 2007. (Cited on pages 10 and 62.)

J. Schmidhuber. Driven by compression progress: A simple principle explains essential aspects of subjective beauty, novelty, surprise, interestingness, attention, curiosity, creativity, art, science, music, jokes. *Anticipatory Behavior in Adaptive Learning Systems, from Sensorimotor to Higher-level Cognitive Capabilities*, 2009. (Cited on pages 61 and 62.)

N. N. Schraudolph, P. Dayan, and T. J. Sejnowski. Temporal Difference Learning of Position Evaluation in the Game of Go. In J. D. Cowan, G. Tesauro, and J. Alspector, editors, *Advances in Neural Information Processing Systems*, volume 6, pages 817–824. Morgan Kaufmann, San Francisco, 1994. (Cited on pages 76 and 78.)

M. Schuster and K. K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45:2673–2681, November 1997. (Cited on page 77.)

H.-P. Schwefel. *Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie*, volume 26. Birkhaeuser, Basel/Stuttgart, 1977. (Cited on page 7.)

S. Seo, M. Wallat, T. Graepel, and K. Obermayer. Gaussian process regression: Active data selection and test point rejection. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN)*, pages 241–246. IEEE, 2000. (Cited on page 67.)

J. Shepherd, D. McDowell, and K. Jacob. Modeling morphology evolution and mechanical behavior during thermo-mechanical processing of semi-crystalline polymers. *Journal of the Mechanics and Physics of Solids*, 54(3):467 – 489, 2006. (Cited on page 7.)

D. Silver, R. S. Sutton, and M. M. 0003. Reinforcement Learning of Local Shape in the Game of Go. In *IJCAI*, pages 1053–1058, 2007. (Cited on page 76.)

K. O. Stanley and R. Miikkulainen. Competitive coevolution through evolutionary complexification. *Journal of Artificial Intelligence Research*, 21:63–100, 2004a. (Cited on page 87.)

K. O. Stanley and R. Miikkulainen. Evolving a Roving Eye for Go. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 2004b. (Cited on pages 74 and 76.)

J. Storck, J. Hochreiter, and J. Schmidhuber. Reinforcement-driven information acquisition in non-deterministic environments. In *International Conference on Artificial Neural Networks (ICANN)*, pages 159–164, Paris, 1995. (Cited on pages 62 and 63.)

R. Storn and K. Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11:341–359, December 1997. (Cited on page 7.)

Y. Sun, D. Wierstra, T. Schaul, and J. Schmidhuber. Efficient Natural Evolution Strategies. In *Genetic and Evolutionary Computation Conference (GECCO)*, 2009a. (Cited on pages xviii and 34.)

Y. Sun, D. Wierstra, T. Schaul, and J. Schmidhuber. Stochastic Search using the Natural Gradient. In *International Conference on Machine Learning (ICML)*, number 1, 2009b. (Cited on pages xviii and 34.)

Y. Sun, T. Glasmachers, T. Schaul, and J. Schmidhuber. Frontier Search. In *Conference on Artificial General Intelligence (AGI)*, Lugano, 2010. (Cited on page xx.)

R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998. (Cited on page 8.)

J. Togelius, T. Schaul, J. Schmidhuber, and F. Gomez. Countering Poisonous Inputs with Memetic Neuroevolution. In *Parallel Problem Solving from Nature (PPSN)*. Springer-Verlag, 2008. (Cited on page xix.)

J. Togelius, T. Schaul, D. Wierstra, C. Igel, F. Gomez, and J. Schmidhuber. Ontogenetic and Phylogenetic Reinforcement Learning. (3):30–33, 2009. (Cited on page xx.)

E. van der Werf, H. J. van den Herik, and J. Uiterwijk. Solving Go on small boards. *International Computer Games Association Journal*, 26, 2003. (Cited on page 74.)

D. Wales and J. Doye. Global Optimization by Basin-Hopping and the Lowest Energy Structures of Lennard-Jones Clusters Containing up to 110 Atoms. *The Journal of Physical Chemistry A*, 101(28):8, 1998. (Cited on pages 8, 54, and 55.)

S.-K. Wang, J.-P. Chiou, and C.-W. Liu. Non-smooth/non-convex economic dispatch by a novel hybrid differential evolution algorithm. *Generation, Transmission Distribution, IET*, 1(5):793 –803, september 2007. (Cited on page 8.)

D. Weyland. A Rigorous Analysis of the Harmony Search Algorithm - How the Research Community can be misled by a "novel" Methodology. *International Journal of Applied Metaheuristic Computing*, 1(2):50–60, 2010. (Cited on page 6.)

A. Wieland. Evolving Neural Network Controllers for Unstable Systems. In *Proceedings of the International Joint Conference on Neural Networks (Seattle, WA)*, pages 667–673, 1991. (Cited on page 53.)

D. Wierstra, T. Schaul, J. Peters, and J. Schmidhuber. Natural Evolution Strategies. In *IEEE Congress on Evolutionary Computation (CEC)*, 2008a. (Cited on page xviii.)

D. Wierstra, T. Schaul, J. Peters, and J. Schmidhuber. Fitness Expectation Maximization. In *Parallel Problem Solving from Nature (PPSN)*. Springer-Verlag, 2008b. (Cited on pages xix and 7.)

D. Wierstra, T. Schaul, J. Peters, and J. Schmidhuber. Episodic Reinforcement Learning by Logistic Reward-Weighted Regression. In *International Conference on Artificial Neural Networks (ICANN)*, 2008c. (Cited on page xx.)

S. Winter, B. Brendel, and C. Igel. Registration of bone structures in 3D ultrasound and CT data: Comparison of different optimization strategies. *International Congress Series*, 1281:242 – 247, 2005. (Cited on page 8.)

L. Wu and P. Baldi. A Scalable Machine Learning Approach to Go. In B. Schölkopf, J. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 1521–1528. MIT Press, Cambridge, MA, 2007. (Cited on pages 74, 77, and 93.)

Y. Zhang, W. Xu, and J. Callan. Exploration and exploitation in adaptive filtering based on bayesian active learning. In *International Conference on Machine Learning (ICML)*, pages 896–903, 2003. (Cited on page 64.)

E. Zitzler and L. Thiele. Multiobjective Optimization Using Evolutionary Algorithms – A Comparative Case Study. In *Parallel Problem Solving from Nature (PPSN V)*, pages 292–301. Springer, 1998. (Cited on page 38.)

Index

active learning, 9
adaptation sampling, 12, 24, 46, 97
artificial curiosity, 10, 61
Atari-Go, 75, 82, 96

BBOB, 47, 53, 57, 96
black-box optimization, 3, 92

Cauchy, 13, 37, 44, 47, 56, 59
CMA-ES, 7, 11, 20, 32, 38, 46, 47, 79, 82, 96
coevolution, 84, 96
combinatorial optimization, 2
computation time, 5
computational complexity, 6, 12, 38, 67, 92
constrained optimization, 2
continuous optimization, 2
convergence analysis, 6
convolutional networks, 76, 82, 96
cost function, 1
costly optimization, 5, 62, 71, 73
covariance matrix, 12, 29
cumulative regret, 5
curiosity-driven optimization, 65, 66, 71

decision variables, 1
differential evolution, 7, 9
discrete optimization, 2
dominance number, 87
double-funnel, 56

elitism, 35
estimation of distribution algorithms, 7, 69, 96
evaluation, 5
evolution strategies, 7, 9, 14, 96
expected improvement, 64
expected information gain, 63, 71
exponential map, 12, 28, 38

Fisher information matrix, 19, 29, 92
fitness baseline, 20
fitness shaping, 12, 20, 46

Gaussian processes, 8, 66, 68, 96
genetic algorithms, 7, 96
genetic programming, 93
global optimization, 3, 59, 69, 70
Go, 73, 74, 93
Go-Moku, 75
Go-Muko, 96
gradient ascent, 12

heavy-tailed distribution, 12, 13, 42, 45, 47, 56, 59
hill-climber, 20, 36, 46, 54
host-parasite, 84

importance mixing, 12, 22, 46, 96
interestingness, 61
invariance, 20, 29, 32, 34, 42, 43, 75

kernel, 66, 68

leave-one-out, 63
Lennard-Jones, 54, 96
linear correlation, 80, 84

local optimization, 3, 69
low-rank, 92
LSTM, 78, 79, 96

Markov chain Monte Carlo, 9
MDLSTM, 78, 82, 90, 93, 96
MDRNN, 77–79, 90, 96
metaheuristics, 3
mixture of Gaussians, 65
multi-modal, 3, 12, 25, 47, 48, 50, 56, 58, 65, 69
multi-objective optimization, 4, 37, 64, 68, 96

natural coordinates, 29
Natural Evolution Strategies, 11
natural gradient, 12, 16, 93
noisy optimization, 3, 48
non-dominated, 4, 38, 64
numerical optimization, 2

objective function, 1
order of convergence, 6

parameter space, 18
Pareto-front, 4, 37, 61, 64, 67, 68
particle swarm optimization, 7, 96
penalty term, 2
Pente, 75, 82, 96
pole balancing, 54, 96
program search, 92
PyBrain, xix, 47, 53, 79, 87, 95, 96

quasi-Newton, 6

radial distribution, 36, 40, 41, 59
randomized search methods, 3
rate of convergence, 6
receptive field, 76
recurrent neural networks, 53, 76, 96
reinforcement learning, 8, 62, 71
response surface methods, 7, 61
restart strategies, 3, 12, 25, 48

scalability, 73, 76, 84
scale-invariant, 16
search distribution, 11
search gradient, 12–14
search space, 1, 18
separable distribution, 13, 38, 40, 48
simplex methods, 6
simulated annealing, 6
SNES, 46, 47
sparse, 92
stochastic search, 12
sublinear, 6
superlinear, 6

unconstrained optimization, 2
utility, 20, 33, 35, 46

weighted Mann-Whitney, 97

xNES, 26, 32, 38, 46, 47, 68, 79, 96