University of Massachusetts Dartmouth

Department of Computer and Information Science

A TUTORIAL ON SUPERVISED LEARNING FROM THE PERSPECTIVE OF MATHEMATICAL OPTIMIZATION

A Thesis in

Computer Science

by

Justin Lovinger

Copyright 2018 by Justin Lovinger

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

May 2018

We approve the thesis of Justin Lovinger

Date of Signature

Iren Valova
Professor, Department of Computer and Information Science
Thesis Advisor

Xiaoqin Zhang
Associate Professor and Chairperson
Department of Computer and Information Science
Thesis Committee

Ming Shao
Assistant Professor, Department of Computer and Information Science
Thesis Committee

Haiping Xu
Graduate Program Director
Department of Computer and Information Science

Ramprasad Balasubramanian
Dean, College of Engineering

Tesfay Meressi
Associate Provost for Graduate Studies

Abstract

A tutorial on supervised learning from the perspective of mathematical optimization

by Justin Lovinger

Popular methods in supervised learning, from regression and neural networks to support vector machines, are commonly presented from the perspective of statistics or biology. Instead, we present common techniques in supervised learning as applications of mathematical optimization and examine the practical benefits this perspective brings.

Under the optimization perspective, linear regression is understood as the function $f(\boldsymbol{X}) = \boldsymbol{X}\boldsymbol{W} + \vec{b}$ that is trained by solving $\arg\min_{\boldsymbol{W},\vec{b}} \xi(f(\boldsymbol{X}), \boldsymbol{Y})$, where $\boldsymbol{W}$ is a weight matrix, $\boldsymbol{X}$ is a matrix of training arguments, $\boldsymbol{Y}$ is a matrix of training targets, and ξ is an error function such as mean squared error.

Similarly, a multilayer perceptron neural network is understood as a function of the form $f(\boldsymbol{X}) = t_{n-1}(\cdots(t_2(t_1(\boldsymbol{X}\boldsymbol{W}_1 + \vec{b})\boldsymbol{W}_2)\cdots)\boldsymbol{W}_{n-1})$, where t_i is the i^{th} transfer function, $\boldsymbol{W}_i$ is the i^{th} weight matrix, and n is the number of layers. Under the optimization perspective, training a multilayer perceptron is the same as training a regression model: solve $\arg\min_{\boldsymbol{W}_1,\boldsymbol{W}_2,\ldots,\boldsymbol{W}_{n-1},\vec{b}} \xi(f(\boldsymbol{X}), \boldsymbol{Y})$.

Mathematical optimization serves as the workhorse for training by solving the arg min problem. Powerful optimization methods such as Broyden-Fletcher-Goldfarb-Shanno (BFGS) and its limited-memory L-BFGS variant can efficiently solve this

problem. The inclusion of line search or trust region techniques removes the need for hand tuned learning rates and drastically improve performance and consistency. Through the rapid training enabled by efficient optimization, more complex models can be applied and larger datasets learned. Bigger data, faster real time learning, and more effective image recognition are possible.

From the optimization perspective, explanation of models is simplified and implementation is naturally modular and flexible. Optimization techniques are easily reused between models. The development of a new generalization improving error function is easily propagated to existing and future models. Datasets are better learned and accuracy improved by easily applying, developing, and testing a multitude of models. When supervised learning is performed from the optimization perspective, an equation with adjustable parameters or an error function is all that is necessary to implement a new model and better solve problems in data science and machine learning.

Acknowledgments

I would like to thank my advisor Dr. Iren Valova for her support and aid in my academic career and for allowing me to pursue my research with freedom and vigor. I could not develop the concepts in this thesis without the freedom to discover important and non-traditional ideas in machine learning.

I would like to thank my friends and family for their support of my academic pursuits and career. I would like to express gratitude to my rabbit Professor BunBun, who allows me to calmly solve problems of even the greatest difficulty.

Finally, I would like to thank the general community of academics whose books, papers, and materials provide me with the understanding and inspiration necessary to thrive.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Supervised learning algorithms are frequently inspired by statistics, probability, or biology. From naive Bayes to neural networks. However, when translated into mathematics and computer code, original inspiration often proves cumbersome and unnecessary. The multilayer perceptron underwent a significant revival once the biological inspiration of thresholds were replaced with the mathematical convenience of differentiable transfer functions [1]. Once framed under the umbrella of mathematics and computer science, supervised learning falls under the purview of mathematical optimization, where the objective is to minimize the error of a model or otherwise provide an optimal output.

When framed as applications of mathematical optimization, several facets of supervised learning [2, 3, 4, 5, 6, 7] are simplified and models unified. Regression [8, 9, 10, 11, 12, 13] and multilayer perceptrons [1, 14, 15, 16, 17, 18] are shown to be nearly equivalent. Decision trees [19, 20, 21, 22, 23, 24, 25, 26] are portrayed as a specialized optimizer. Under this unification, the development of algorithms is improved as techniques and code can be reused. An optimizer used for regression is easily applied to a multilayer perceptron or radial basis function network [27, 28, 29, 30, 31, 32].

Mathematical optimization [33, 34, 35, 36, 37, 38, 39] is the process of finding parameters that maximize or minimize a particular quantity. Formally,

$$\arg\min_{\vec{x}} f(\vec{x}) \tag{1.1}$$

finds a parameter vector $\vec{x}$ that minimizes the quantity $f(\cdot)$. In machine learning, optimization can minimize the error of a supervised learning model, or maximize the utility or reward of a reinforcement learning model. Unsupervised learning models are frequently developed as closed form approximations of optimization models.

When the equation to optimize is differentiable, as many machine learning models are, powerful gradient optimization [33, 34, 35, 40, 41, 42] methods become available. Newton [33, 43, 44, 45, 46], quasi-newton [33, 47, 48, 49, 50, 51, 52, 53], conjugant gradient [33, 40, 54, 55, 56, 57, 58, 59] and other optimizers can solve complex optimization problems with alarming alacrity. It is the power of differentiable optimization, more than any other factor, that has enabled the modern explosion of supervised learning, neural networks, and their application to big data.

When derivatives are unavailable, derivative-free methods, such as Nelder-Mead downhill simplex [60, 33, 61, 62, 63, 64, 65] or genetic algorithms [61, 65, 66, 67, 68, 69, 70], are still capable of general optimization. However, in the absence of heuristic information provided by derivatives, performance suffers drastically. To compensate, specialized algorithms are often developed and applied to solve particular non-differentiable problems. Examples include the ID3 [23] algorithm for training the inequality based decision tree model [19, 20, 21].

The following chapters will examine several optimization algorithms, supervised learning models, and their interaction and performance. Gradient descent and line search are discussed and details provided. Multiple derivative-free optimizers are also included. Implementation of a number of state-of-the-art supervised learning models are shown from the optimization perspective, including derivatives for gradient optimization. With these details, the development of a fully featured supervised learning library with many models and optimizers is possible. To further aid optimizer and model selection for practical supervised learning, comprehensive optimizer and model comparisons are provided and discussed. Notes on mathematical notation used throughout this report are given in Appendix A.

Chapter 2

Gradient Optimization

Utilizing the slope information of differentiable functions, we can effectively slide down the landscape of an objective function f, one step at a time, settling in basins of low error. The simplest application of gradient descent moves parameters $\vec{x}$ directly down the direction of the objective derivative f', a.k.a. gradient: $-f'(\vec{x})$. While this first derivative information provides significant optimization improvements over derivative-free optimization, even greater performance can be achieved with second, or higher, derivatives. These Newton methods can converge to optimal or near-optimal parameters several times faster than first derivative, or steepest descent, methods. However, second derivatives are often slow or infeasible to calculate. An effective compromise is quasi-Newton methods, which approximate second-derivatives without direct calculation. Quasi-Newton optimization algorithms are highly refined and capable of rapid convergence with few iterations and fast calculation. Low memory variants allow for effective second derivative Hessian approximation on problems with thousands of parameters, where the n^2 elements Hessian is normally infeasible for high n parameters.

Finding a step direction $\vec{p}$ is only the first step of a gradient descent iteration. Next, we must know how far to move parameters in the step direction. We must find the step size α. A naive implementation of gradient descent simply provides a static α, that is the same every iteration. However, this provides poor performance. On some iterations, high α can massively overshoot, passing over a basin of attraction and ending up in a poor region of the objective landscape. On other iterations, low α may make little difference. A slightly better, but still naive, implementation starts with high α and gradually reduces α every iteration. This implementation relies on the assumption that earlier iterations require high α and later iterations benefit from low α. However, this assumption is frequently false, especially with Newton and quasi-Newton methods, which have an optimal $\alpha = 1$ near an optima.

An effective solution is line-search [33, 71, 72, 73, 74, 75, 76], where several step sizes are examined through a organized algorithm that quickly converges to an effective α. Although this requires multiple objective function evaluations, the benefit of finding an effective α frequently outweighs the additional cost.

Note that most line search methods require or benefit from an initial step size α_0. Consequently, several methods exist for determining an initial step size, frequently using information from previous optimization iterations. A typical line search optimization iteration involves:

1. Find a step direction.
2. Find an initial step size, typically using previous optimization iterations and the current step direction.
3. Find a step size, typically using an initial step size and the current step direction.

The trust region [33, 77, 78, 79, 80, 81, 82] alternative to line search postulates that the gradient at a point $\vec{x}$ is only useful in a small region around $\vec{x}$. Beyond this trust region, the gradient cannot be trusted. Therefore, the length of an optimization step $\alpha\vec{p}$ should not exceed the trust region. Formally, $\|\alpha\vec{p}\| \leq \Delta$, where Δ is trust region. With this requirement, we can reformulate the optimization problem (1.1) as

$$\begin{aligned} &\arg\min_{\vec{x}} f(\vec{x}) \\ &\text{s.t. } \|\alpha_k \vec{p}_k\| \leq \Delta_k \forall k \end{aligned} \tag{2.1}$$

where subscript k denotes an optimization iteration. In other words, trust region optimization minimizes an objective function f and includes a constraint that the length of each step never exceeds its trust region. Note that presentations of trust region methods typically eschew the α variable, implicitly presenting $\vec{p}$ as $\alpha\vec{p}$ because α and $\vec{p}$ are commonly discovered in tandem in a trust region method.

2.1 Step Direction

Obtaining a direction $\vec{p}$ to move a given parameter vector $\vec{x}$, based on first or second derivative information, is the core of gradient based optimization. With an effective $\vec{p}$ we can guarantee eventual or immediate improvement in objective value. This guarantee differentiates gradient optimization from derivative-free optimization and allows rapid convergence.

2.1.1 Steepest Descent

Steepest descent [33, 83, 84, 85, 86, 87, 88] is the purest expression of gradient descent. In steepest descent, step direction $\vec{p}$ is simply the gradient of the objective function f: $\vec{p} = -f'(\vec{x})$ given parameters $\vec{x}$. As a result, the optimization routine for steepest descent is likewise simple, as shown in Algorithm 1. In lieu of further sophistication, steepest descent moves the parameter vector $\vec{x}$ down the gradient of the objective function until it settles in a basin of attraction, as defined by a small gradient. Most of the implementation details of steepest descent lie in determining an effective step size.

Algorithm 1 Steepest Descent

Given objective function f
$\vec{x} \leftarrow$ a randomly generated initial parameter vector
while $\|f'(\vec{x})\| >$ a low value **do** ▷ Test for convergence
  $\vec{p} \leftarrow -f'(\vec{x})$
  $\alpha \leftarrow$ an effective step size given f, $\vec{x}$, and $\vec{p}$ ▷ See Section 2.2 for details
  $\vec{x} \leftarrow \vec{x} + \alpha\vec{p}$
end while
return $\vec{x}$

However, the algorithms for finding step size are general to all gradient optimizers, reusable between them, and are discussed in Section 2.2.

2.1.2 BFGS

The Broyden-Fletcher-Goldfarb-Shanno (BFGS) [33, 76, 89, 90, 91, 92, 93] quasi-Newton method improves upon steepest descent by utilizing second derivative information of an objective function f. However, unlike pure Newton methods, quasi-Newton methods can quickly approximate the second derivative f'' using first derivative f' information from multiple optimization iterations. Iteration after iteration, a quasi-Newton method improves its second derivative approximation $\hat{f}''$ until $\hat{f}'' \approx f''$. With $\hat{f}''$, quasi-Newton methods avoid the need to directly calculate f'', avoiding potentially expensive calculations and easing implementation requirements. With f'' or $\hat{f}''$, we can obtain an improved step direction $-f'(\vec{x})(f''(\vec{x})^{-1})^T$ at point $\vec{x}$. To avoid an expensive matrix inverse operation, most quasi-Newton methods directly approximate the inverse $\hat{f}''(\cdot)^{-1}$.

Although other quasi-Newton methods exist, BFGS is a very popular and well refined variant. The core of BFGS is its inverse Hessian approximation equation

$$\hat{f}''(\vec{x}_k)^{-1} = (I - \rho_k \vec{s}_k^T \vec{y}_k)\hat{f}''(\vec{x}_{k-1})^{-1}(I - \rho_k \vec{y}_k^T \vec{s}_k) + (\rho_k \vec{s}_k^T \vec{s}_k) \tag{2.2}$$

where I is an identity matrix with the same dimensions as the Hessian of f, $\vec{x}$ is a parameter vector, subscript k denotes a variable from the k^{th} optimization iteration, $\vec{s}_k = \vec{x}_k - \vec{x}_{k-1}$, $\vec{y}_k = f'(\vec{x}_k) - f'(\vec{x}_{k-1})$, and $\rho_k = 1/(\vec{s}_k \cdot \vec{y}_k)$. Note that $\hat{f}''(\vec{x}_{k-1})^{-1}$ can be cached for performance and $\hat{f}''(\vec{x}_0)^{-1} = I$ for the first optimization iteration. With an approximate inverse Hessian $\hat{f}''(\vec{x}_k)^{-1}$, we can supplement the gradient f' to obtain an improved step direction $\vec{p}_k = -f'(\vec{x}_k)(\hat{f}''(\vec{x}_k)^{-1})^T$. A full optimization procedure with BFGS is given in Algorithm 2. Note that periodically resetting the

Algorithm 2 BFGS

```
Given objective function f
x⃗_0 ← a randomly generated initial parameter vector
k ← 0
while ‖f′(x⃗_k)‖ > a low value do                      ▷ Test for convergence
    if k = 0 then
        H_k ← I                                        ▷ f̂″(x⃗_0) = I
    else
        H_k ← f̂″(x⃗_k) (2.2)
    end if
    p⃗_k ← −f′(x⃗_k) H_k^T
    α_k ← an effective step size given f, x⃗_k, and p⃗_k   ▷ See Section 2.2 for details
    x⃗_{k+1} ← x⃗_k + α_k p⃗_k
    k ← k + 1
end while
return x⃗_k
```

approximate inverse hessian H_k to identity I is important for non-convex problems where the approximation can lose accuracy over time.

2.1.3 L-BFGS

Although the Hessian information can drastically improve convergence time, $f''(\vec{x})$ requires the storage and computation n^2 elements, where n is the number of elements in a parameter vector $\vec{x}$. When n is large, as is true for complex supervised learning models and even simple models on big data problems, calculating $f''(\vec{x})$ is infeasible. However, when calculating a Newton or quasi-Newton step direction for $\vec{x}$, the Hessian $f''(\vec{x})$ is not intrinsically essential. We can sidestep the problem of calculating $f''(\vec{x})$ by algorithmically approximating the product $f'(\vec{x})(f''(\vec{x})^{-1})^T$ from previous optimization iterations.

When the $f'(\vec{x})(f''(\vec{x})^{-1})^T$ product is approximated from the BFGS rule, the resulting algorithm is designated: limited-memory BFGS (L-BFGS) [33, 94, 95, 96, 97, 98, 99]. The L-BFGS approximated step direction procedure is given in Algorithm 3. Using Algorithm 3, a full L-BFGS optimizer is derived and presented in Algorithm 4.

2.2 Line Search

An optimal step size $\alpha > 0$ minimizes the quantity $f(\vec{x} + \alpha\vec{p})$ and is formalized as

$$\arg\min_{\alpha} f(\vec{x} + \alpha\vec{p}) \tag{2.3}$$

where f is an objective function, $\vec{x}$ is a parameter vector for f, and $\vec{p}$ is a step direction. However, solving this minimization problem every optimization iteration is unnecessarily slow. Instead, line search uses fast heuristic based methods to find an effective, if not optimal, α.

The sufficient decrease, or Armijo, condition [33, 100, 101, 102, 103] is a simple to calculate heuristic stating that decrease in objective value should be justified by the magnitude of a step. Formally, the sufficient decrease condition is given by

$$f(\vec{x} + \alpha\vec{p}) \leq f(\vec{x}) + c_1\alpha(\vec{p} \cdot f'(\vec{x})) \tag{2.4}$$

where $c_1 \in (0, 1)$ is a sufficient decrease strictness parameter. This condition is satisfied when the decrease in objective value, given by $f(\vec{x} + \alpha\vec{p}) - f(\vec{x})$, exceeds a combination of step size and the directional objective function derivative $\vec{p} \cdot f'(\vec{x})$. Note

Algorithm 3 L-BFGS Hessian Gradient Product

$k \leftarrow$ current optimization iteration
Given objective function f,
memory limit m,
a sequence of stored parameter differences $\vec{s}_{k-m+1} \ldots \vec{s}_k$,
and a sequence of stored gradient differences $\vec{y}_{k-m+1} \ldots \vec{y}_k$

$\vec{r} \leftarrow f'(\vec{x}_k)$ ▷ Running result for $f'(\vec{x})(f''(\vec{x})^{-1})^T$
$i \leftarrow k$
while $i > k - m$ **do** ▷ Backwards pass from newest iteration to oldest stored
    $\rho_i \leftarrow 1/(\vec{s}_i \cdot \vec{y}_i)$
    $\alpha_i \leftarrow \rho_i \vec{s}_i \cdot \vec{r}$
    $\vec{r} \leftarrow \vec{r} - \alpha_i \vec{s}_i$
    $i \leftarrow i - 1$
end while

if $k > 0$ **then** ▷ Not first optimization iteration
    $\vec{r} \leftarrow \frac{\vec{s}_k \cdot \vec{y}_k}{\vec{y}_k \cdot \vec{y}_k} \vec{r}$ ▷ Analogue for initial Hessian in BFGS
end if

$i \leftarrow k - m + 1$
while $i \leq k$ **do** ▷ Forwards pass from oldest stored iteration to newest
    $\vec{r} \leftarrow \vec{r} + \vec{s}_i(\alpha_i - \rho_i(\vec{y}_i \cdot \vec{r}))$
    $i \leftarrow i + 1$
end while

return $\vec{r}$

that this presentation of the sufficient decrease condition assumes a minimization, not maximization, problem. Effectively, this condition states that, the greater the step size, the more the objective value should decrease.

A second condition, intended to prevent unreasonably small α, compliments the sufficient decrease condition. The curvature condition [33, 104, 105], given a strictness parameter $c_2 \in (c_1, 1)$,

$$f'(\vec{x} + \alpha\vec{p}) \cdot \vec{p} \geq c_2 f'(\vec{x}) \cdot \vec{p} \tag{2.5}$$

examines the slope of f and dictates that, if the slope is steeply negative at a given $\vec{x}$, α can safely increase. This implicitly provides a lower bound on α that complements the implicit upper bound provided by the sufficient decrease condition. Collectively, the sufficient decrease and curvature conditions are known as the Wolfe conditions. Consistently selecting α that satisfies (2.4) and (2.5) is an essential component of numerous numerical optimization convergence proofs. In other words, the Wolfe conditions lead to fast and consistent optimization when paired with compatible optimizers.

2.2.1 Backtracking Line Search

A simple but effective method for obtaining step size α is backtracking line search [33, 100, 106, 107, 108, 109, 110, 111, 112], based on the sufficient decrease condition (2.4). This technique begins with an initial, typically high, α and gradually decreases it until the sufficient decrease condition is met. This condition provides an effective and easy to calculate heuristic to quickly find a satisfactory α, and because it is easier to satisfy as α decreases, the backtracking method will eventually satisfy it.

Backtracking line search is detailed in Algorithm 5. Although this algorithm requires a number of hyperparameters, these hyperparameters are flexible and easy to assign in practice. The sufficient decrease strictness parameter c_1 is typically given

Algorithm 4 L-BFGS

Given objective function f
$\vec{x}_0 \leftarrow$ a randomly generated initial parameter vector
$k \leftarrow 0$
while $\|f'(\vec{x}_k)\| >$ a low value **do** ▷ Test for convergence
  if $k > m$ **then** ▷ Memory limit reached, discard excess
    Remove $\vec{s}_{k-m}$ from memory
    Remove $\vec{y}_{k-m}$ from memory
  end if
  if $k > 0$ **then** ▷ Not first optimization iteration
    ▷ Note that only $\vec{x}_k$, $\vec{x}_{k-1}$, $f'(\vec{x}_k)$, and $f'(\vec{x}_{k-1})$ must be stored in memory
    $\vec{s}_k \leftarrow \vec{x}_k - \vec{x}_{k-1}$
    $\vec{y}_k \leftarrow f'(\vec{x}_k) - f'(\vec{x}_{k-1})$
  end if
  $\vec{p}_k \leftarrow -\vec{r}$, where $\vec{r}$ is calculated using Algorithm 3
  $\alpha_k \leftarrow$ an effective step size given f, $\vec{x}_k$, and $\vec{p}_k$ ▷ See Section 2.2 for details
  $\vec{x}_{k+1} \leftarrow \vec{x}_k + \alpha_k \vec{p}_k$
  $k \leftarrow k + 1$
end while
return $\vec{x}_k$

Algorithm 5 Backtracking Line Search

Given sufficient decrease strictness parameter c_1,
and backtracking rate $\rho \in (0, 1)$

Given objective function f,
parameter vector $\vec{x}$,
and step direction $\vec{p}$

$\alpha \leftarrow$ initial step size α_0 ▷ See Section 2.2.3 for details
while Sufficient decrease condition (2.4) is **false** given α, f, $\vec{p}$, and c_1 **do**
  $\alpha \leftarrow \rho\alpha$
end while
return α

a high value $c_1 \approx 0.5$. The backtracking rate ρ is typically given a value $\rho \in [0.1, 0.8]$ such as $\rho = 0.5$. Decreasing ρ leads to faster convergence but potentially worse step size due to overshooting. Additionally, this line search requires a number of arguments from the current optimization iteration: f, $\vec{x}$, and $\vec{p}$. Initial step size α_0 can be determined with a method detailed in Section 2.2.3 or elsewhere. The increment previous step size method pairs especially well with backtracking line search. With Newton and quasi-Newton methods, a static $\alpha_0 = 1$ can be effective because these methods have optimal $\alpha = 1$ near an optima.

The simplicity of backtracking line search makes it a popular line search method. It is particularly effective with well scaled optimizers such as Newton optimizers, where α varies little between optimization iterations. With steepest descent and conjugate gradient optimizers, backtracking line search is ineffective because these methods may require $\alpha > \alpha_0$, making choice of α_0 problematic when paired with the varying required α of these poorly scaled optimizers. Backtracking line search is more effective with quasi-Newton than steepest descent methods, but less effective than with Newton methods. Although quasi-Newton optimizers are eventually well scaled, once the Hessian is well approximated and behavior resembles Newton optimizers, they closer resemble steepest descent during earlier iterations.

2.2.2 Wolfe Line Search

Wolfe line search [33, 113, 114] is an alternative to the backtracking method with greater flexibility and frequently better performance. Based on the Wolfe conditions (2.4) and (2.5), this approach can quickly generate any $\alpha \in (0, \alpha_{\max})$, for a given max step size $\alpha_{\max}$, regardless of initial step size. The Wolfe line search consists of two phases. The first phase increases α while examining f with regard to α, until bounds on α are discovered that satisfy the sufficient decrease condition. The second phase then refines α within its given bounds, returning α that satisfies both the sufficient decrease and curvature conditions.

Wolfe line search is detailed in Algorithm 6. Note that an upper limit on iterations for Wolfe line search and the zoom procedure may be necessary given numerical precision errors of floating point computer operations. As with backtracking line search, Wolfe line search relies on the sufficient decrease condition and has a corresponding strictness hyperparameter c_1. Wolfe line search can manage a significantly stricter $c_1 \approx 1\text{e}{-4} = 0.0001$. Unlike backtracking line search, Wolfe line search utilizes the curvature condition and has a corresponding strictness hyperparameter c_2. A relatively high $c_2 \approx 0.9$ is effective in practice. Wolfe line search also requires a number of arguments from the current optimization iteration: f, $\vec{x}$, and $\vec{p}$. Initial step size α_0 can be determined with a method detailed in Section 2.2.3 or elsewhere. Wolfe line search is a robust method, able to quickly determine an effective α regardless of α_0. This makes it a popular method for poorly scaled optimizers such as steepest descent and conjugate gradient. When paired with such optimizers, an interpolating quadratic or first-order change initial step is effective to further overcome the varying required α of these optimizers, and unlike backtracking line search, Wolfe line search can return $\alpha > \alpha_0$, mitigating any problematically small α_0 returned by interpolating quadratic or first-order change.

2.2.3 Initial Step Size

Most line search methods operate by incrementally adjusting a step size until their criteria is met. Consequently, a starting point is required. The following are techniques for quickly approximating an effective step size. These techniques are not intended to immediately generate effective step sizes but rather return educated guesses with

Algorithm 6 Wolfe Line Search

Given sufficient decrease strictness parameter c_1,
curvature strictness parameter c_2,
and Wolfe increment rate $r > 1$

Given objective function f,
parameter vector $\vec{x}$,
and step direction $\vec{p}$

$i \leftarrow 0$
$\alpha_{-1} \leftarrow 0$
$\alpha_0 \leftarrow$ initial step size ▷ See Section 2.2.3 for details
loop
  if ($i > 0$ and $f(\vec{x} + \alpha_i \vec{p}) \geq f(\vec{x} + \alpha_{i-1}\vec{p})$)
      or sufficient decrease condition (2.4) is **false** given α_i, f, $\vec{p}$, and c_1 **then**
    return WOLFE_ZOOM(α_{i-1}, α_i) ▷ See Algorithm 7
  end if
  if $|f'(\vec{x} + \alpha_i \vec{p}) \cdot \vec{p}| \leq -c_2 f'(\vec{x}) \cdot \vec{p}$ **then**
    return α_i
  end if
  if $f'(\vec{x} + \alpha_i \vec{p}) \cdot \vec{p} \geq 0$ **then**
    return WOLFE_ZOOM(α_i, α_{i-1}) ▷ See Algorithm 7
  end if
  $\alpha_{i+1} \leftarrow r\alpha_i$ ▷ Or increase α_i by interpolating between α_i and a given max α.
  $i \leftarrow i + 1$
end loop

Algorithm 7 Wolfe Line Search Procedures

function WOLFE_ZOOM(α_{lo}, α_{hi}) ▷ such that $f(\vec{x} + \alpha_{lo}\vec{p}) < f(\vec{x} + \alpha_{hi}\vec{p})$
    loop
            ▷ Get α between α_{lo} and α_{hi}. Common techniques are bisect, quadratic interpolation, and cubic interpolation.
        $\alpha \leftarrow$ BISECT$(\min(\alpha_{lo}, \alpha_{hi}), \max(\alpha_{lo}, \alpha_{hi}))$
        if $f(\vec{x} + \alpha\vec{p}) \geq f(\vec{x} + \alpha_{lo}\vec{p})$ or (2.4) is **false** given α, f, $\vec{p}$, and c_1 **then**
            $\alpha_{hi} \leftarrow \alpha$
        else ▷ α is an improvement
            if $|f'(\vec{x} + \alpha\vec{p}) \cdot \vec{p}| \leq -c_2 f'(\vec{x}) \cdot \vec{p}$ **then**
                return α
            end if
            if $(\alpha_{hi} - \alpha_{lo})(f'(\vec{x} + \alpha\vec{p}) \cdot \vec{p}) \geq 0$ **then**
                $\alpha_{hi} \leftarrow \alpha_{lo}$
            end if
            $\alpha_{lo} \leftarrow \alpha$
        end if
    end loop
end function

function BISECT(Lower value a, higher value $b \geq a$)
    return $a + 0.5(b - a)$
end function

minimal computation. These initial step sizes can then be refined with an appropriate line search.

An analogue for backtracking line search, that pairs perfectly with it, is the increment previous step size method [33]. While backtracking decreases step size until the sufficient decrease condition is met, the increment method for initial step size increases the step size used in the previous optimization iteration by a fixed factor $r > 1$, and uses it as the initial step size α_0 for the current iteration k.

$$\alpha_{0k} = r\alpha_{k-1} \tag{2.6}$$

where subscript k denotes a value for the k^{th} optimization iteration. A lower bound l and upper bound u on α_0 can optionally constrain $l \leq \alpha_0 \leq u$:

$$\alpha_{0k} = \max(l, \min(u, r\alpha_{k-1})) \tag{2.7}$$

Effective r varies greatly with corresponding line search method. When paired with backtracking line search, incrementing slightly more than one backtracking step with no upper bound is effective in practice: $r = \frac{2}{\rho} - 1$, where ρ is backtracking rate. When paired with Wolfe line search, small $r = 1.05$ and upper bound $u = 1$ is appropriate, because Wolfe line search can return $\alpha > \alpha_0$. Upper bound $u = 1$ is important for Newton and quasi-Newton methods, where $\alpha = 1$ is eventually always effective, but is less useful for other optimizers. If paired with backtracking line search and a Newton or quasi-Newton optimizer, a lower bound $l = 1$ is important because low α may not satisfy the curvature condition. However, note that no lower bound guarantees the curvature condition is satisfied. For the first iteration, when previous step size is undefined, α_0 can arbitrarily equal 1.

Another, slightly more sophisticated, method for approximating α_0 is to assume that the first-order change is the same as the previous optimization iteration [33, 115]:

$$\alpha_0 = \alpha_{k-1} \frac{f'(\vec{x}_{k-1}) \cdot \vec{p}_{k-1}}{f'(\vec{x}_k) \cdot \vec{p}_k} \tag{2.8}$$

which is derived from $\alpha_0 f'(\vec{x}_k) \cdot \vec{p}_k = \alpha_{k-1} f'(\vec{x}_{k-1}) \cdot \vec{p}_{k-1}$. An advantage of the first-order change strategy is that is requires no hyperparameters. This method is effective for optimizers that produce step directions with varying magnitudes. Unlike the increment previous step method, the first-order change equation can produce significantly different α_0 every iteration, which is important when each optimization iteration requires vastly different α.

Another method, that has similar use to the first-order change method, is the interpolating quadratic [33, 115]. As the name implies, this strategy interpolates a quadratic through $f(\vec{x}_{k-1})$, $f(\vec{x}_k)$, and $f'(\vec{x}_{k-1}) \cdot \vec{p}_{k-1}$, giving

$$\alpha_0 = \frac{2(f(\vec{x}_k) - f(\vec{x}_{k-1}))}{f'(\vec{x}_k) \cdot \vec{p}_k} \tag{2.9}$$

Similar to the first-order change method, interpolating quadratic has no hyperparameters and is effective on optimizers with poorly scaled $\vec{p}$, such as steepest descent and conjugate gradient.

Chapter 3

Derivative-Free and Non-Smooth Optimization

When an objective function is differentiable, gradient optimization can quickly and effectively find optima. However, several real and important problems are represented by non-differentiable equations or simulations. Under these circumstances, alternatives to direct gradient optimization must be explored

An approach that still utilizes the gradient descent techniques in Chapter 2 involves approximating derivatives with finite differences. Given parameter vector $\vec{x}$ and objective function f, we can approximate the i^{th} partial derivative of $\vec{x}$ as

$$\frac{df}{d\vec{x}_i} = \frac{f(\vec{x} + \epsilon\vec{o}_i) - f(\vec{x} - \epsilon\vec{o}_i)}{2\epsilon} \tag{3.1}$$

where ϵ is a small scalar and $\vec{o}_i$ is a vector with all 0 elements except for a single 1 for the i^{th} element. In other words, the partial derivative of an element in $\vec{x}$ is the difference obtained by a small perturbation in that element. Although this technique is feasible for $\vec{x}$ with a small number of elements, the computation required to approximate a single Jacobian $f'(\vec{x})$ scales linearly with the length of $\vec{x}$. This quickly becomes infeasible on problems with hundreds or thousands of parameters.

Alternatively, gradient descent is discarded, and alternative techniques that eschew derivatives entirely are utilized. Arguably the most popular derivative-free optimizer is the genetic algorithm (GA) [61, 65, 66, 67, 68, 69, 70]. GA, inspired by Darwinian evolution, combines and mutates parameter vectors selected by fitness (objective function value), thereby stochastically increasing fitness over many iterations.

Population-based incremental learning (PBIL) [116, 117, 118, 119, 120, 121] maintains a probability distribution for each element of a parameter vector, generates new parameters by sampling from this set of probabilities, and adjusts the probability distributions based on the objective value of each sample.

Gravitational search algorithm (GSA) [122, 123, 124, 125, 126, 127] maintains a population of parameter vectors and moves these vectors through objective value space by approximating the laws of gravitational attraction. Each parameter vector has mass proportional to its objective function value, thereby drawing low value vectors towards higher value vectors. Semi-stochastic movement allows greater exploration of the problem space.

Without derivative information, gradient norm cannot be a stopping criteria. The simplest stopping criteria commonly used in derivative-free optimization is a maximum number of iterations. Optimization continues for i_{max} iterations, where i_{max} is a user defined maximum number of iterations. However, this can result in poor performance if i_{max} is too small or unnecessary computation if an optima is found in an iteration significantly less than i_{max}. Alternatively, optimization can stop after a number of iterations i_{imprv} without an improvement in objective value. With i_{imprv}, no more than i_{imprv} iterations occur after an optima is found, and optimization can continue as long as improvements are made. Finally, if bounds on objective value are known a priori, optimization can stop when a near global optima $\vec{x}$ is found, as defined by $f(\vec{x}) \approx f(\vec{x}^*)$, where $f(\vec{x}^*)$ is the objective value of a global optima.

3.1 Genetic Algorithm

A genetic algorithm (GA) is made up of three core components: selection, crossover, and mutation. Every iteration, parameter vectors with good objective value are *selected* for *crossover* and resulting vectors may *mutate*. However, before this process can begin, GA requires a population of multiple parameter vectors P. An initial GA population often consists of random vectors. Note that GA literature refers to these parameter vectors as chromosomes and objective value as fitness.

Parameter vectors for GA are frequently binary, as defined by containing only 0 and 1 elements. However, just as computer circuits represent complex data, base 10 digits, and real numbers with binary, binary GA vectors can represent a variety of data, including real number vectors. We do not differentiate between binary and real GA vectors unless otherwise specified, with the understanding that binary vectors can represent real vectors.

The high level GA optimization loop is given in Algorithm 8. GA continues this process, combining and mutating vectors, to improve the objective value of vectors in each population. After many iterations, the best discovered vector is returned. Note that the best vector is not always in the last population. Tracking the best vector over all iterations is necessary to return the best discovered.

A high population size $n \geq 20$ is recommended because most GA diversity comes from initial population. Larger parameter vectors require larger populations for effective optimization. A high crossover chance $p_c \approx 0.7$ is recommended to further exploration and exploitation between iterations. Choice of mutation chance p_m depends largely on choice of mutation function f_m. If f_m incorporates its own probability, we can set $p_m = 1$ to instead rely on the mutation probability in f_m.

Algorithm 8 Genetic Algorithm Optimization Loop

```
Given population size n,
crossover chance p_c,
mutation chance p_m,
selection function f_s,
crossover function f_c,
and mutation function f_m

                  ▷ Initial population can contain binary vectors or real valued vectors
P ← an initial population of n random vectors
while a stopping criteria is not met do                        ▷ Main optimization loop
    P_s ← n vectors in P selected with f_s                                  ▷ Selection
    P ← an empty list                                                 ▷ Begin crossover
    for each pair of vectors x⃗_1 and x⃗_2 in P_s do
        if A random roll exceeds p_c then                ▷ Crossover with probability p_c
            Add the results of f_c(x⃗_1, x⃗_2) to P         ▷ Add two crossed vectors to P
        else
            Add x⃗_1 and x⃗_2 to P
        end if
    end for
    for vector x⃗ ∈ P do                                                ▷ Begin mutation
        if A random roll exceeds p_m then                   ▷ Mutate with probability p_m
            f_m(x⃗)                                                          ▷ Mutate x⃗
        end if
    end for
end while

return Vector x⃗ with best f(x⃗) among all populations
```

3.1.1 GA Selection

Selection is the primary GA mechanism that drives overall fitness improvement between iterations by selecting vectors with high objective value. By comparison, crossover and mutation do not traditionally depend on objective value. The exact mechanism by which vectors of good value are selected differentiates selection functions.

A straightforward and popular mechanism for selecting vectors is roulette selection [128, 129, 130]. As the name implies, a virtual roulette wheel is spun to select a vector for the next population. This process is repeated n times to generate a population of n vectors. By scaling the size of virtual roulette slices by vector objective value, such that vectors with better value have larger slices, average objective value of the selected population stochastically increases. The roulette selection procedure is given in Algorithm 9.

Another selection function, popular for its simplicity and efficiency, is tournament selection [131, 129, 130]. In tournament selection, t random parameter vectors in a population are selected and the vector with the best objective value is added to the next population. This process is repeated n times to generate a population of n vectors. The tournament selection procedure is given in Algorithm 10. Note that each participant objective value $f(\vec{p})$ can be cached to avoid repeat evaluations. A simple tournament size $t = 2$ is popular in practice.

3.1.2 GA Crossover

Once vectors are selected, we can further explore the problem space by combining these vectors. Crossover takes elements from two vectors to return new vectors with potentially better objective value. Crossover forms one half of the GA improvement

Algorithm 9 Roulette Selection

Given a set of parameter vectors $P = \vec{x}_1, \dots \vec{x}_n$,
and objective function f

for $i = 1, \dots, n$ **do**
  $f_i \leftarrow -f(\vec{x}_i)$ ▷ Negative for minimization problem
end for
$f_M \leftarrow \min(f_1, \dots, f_n)$
for $i = 1, \dots, n$ **do**
  $f_i \leftarrow f_i - f_M$ ▷ Scale each objective value to a positive number
end for
$f_S \leftarrow \sum_{i=1}^{n} f_i$
for $i = 1, \dots, n$ **do**
  $p_i \leftarrow {}^{f_i}/_{f_S}$ ▷ Convert objective values to probabilities, by scaling to a sum of 1
end for
$r_0 \leftarrow 0$
for $i = 1, \dots, n$ **do**
  $r_i \leftarrow r_{i-1} + p_i$ ▷ Prepare probabilities for roulette
end for

function ROULETTE
  $\zeta \leftarrow$ a random number in range $[0, 1]$
  for $i = 1, \dots, n$ **do**
    if $r_i \geq \zeta$ **then**
      return $\vec{x}_i$
    end if
  end for
end function
for $i = 1, \dots, n$ **do**
  $\vec{s}_i \leftarrow$ ROULETTE
end for
return $\vec{s}_1, \dots, \vec{s}_n$

Algorithm 10 Tournament Selection

Given tournament size t,
a set of parameter vectors $P = \vec{x}_1, \dots \vec{x}_n$,
and objective function f
for $i = 1, \dots, n$ **do**
  $\vec{p}_1, \dots, \vec{p}_t \leftarrow t$ randomly selected vectors in P ▷ Get random participants
  $\vec{s}_i \leftarrow \arg\min_{\vec{p} \in \vec{p}_1, \dots, \vec{p}_t} f(\vec{p})$ ▷ Select participant with best objective value
end for
return $\vec{s}_1, \dots, \vec{s}_n$

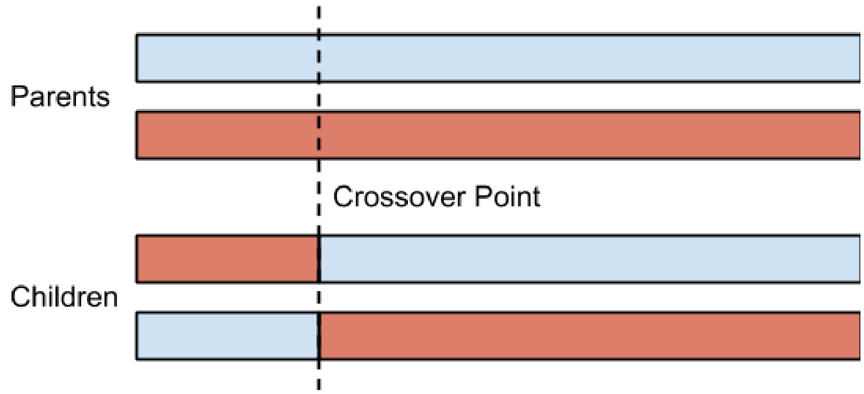

Figure 3.1: One-point crossover generating two child vectors from two parent vectors [135]

mechanism, the other half being mutation. While mutation provides random exploration of a problem space, crossover allows exploitation of existing high value vectors.

The traditional crossover mechanism that closely mirrors biological reproduction is one-point crossover [132, 133, 134]. In one-point crossover, a random position between two adjacent elements is selected. Elements from the first vector to the left of this position is combined with element from the second vector to the right of this position. Another vector can be formed from the inverse. One-point crossover is depicted in Figure 3.1 and described in Algorithm 11. Note that subscript sequence

Algorithm 11 One-Point Crossover

Given parent vectors $\vec{a}$ and $\vec{b}$ ▷ where $|\vec{a}| \leq |\vec{b}|$ and $|\cdot|$ is the length of a vector
$c_p \leftarrow$ a random integer in range $[1, |\vec{a}| - 1]$
return $\vec{a}_{1,\ldots,c_p}$ concatenated with $\vec{b}_{c_p+1,\ldots,|\vec{b}|}$ and $\vec{b}_{1,\ldots,c_p}$ concatenated with $\vec{a}_{c_p+1,\ldots,|\vec{a}|}$

$i, \ldots, j$ is a slice of the i^{th} element to the j^{th} element in a vector.

When adjacent elements of parameter vectors have little in common, one-point crossover can perform poorly. An alternative that does not rely on adjacent similarity in parameter vectors, is uniform crossover [136, 137, 138, 132]. In uniform crossover, a virtual coin is flipped for each element. On heads, the element from the first parent vector is added to the first child vector, and the corresponding element from the second parent vector is added to the second child vector. On tails, the element from the first parent vector is added to the second child vector, and the corresponding element from the second parent vector is added to the first child vector. As such, elements between the two parent vectors are randomly mixed into two new vectors. The uniform crossover procedure is given in Algorithm 12.

Algorithm 12 Uniform Crossover

Given parent vectors $\vec{a}$ and $\vec{b}$ ▷ where $|\vec{a}| \leq |\vec{b}|$ and $|\cdot|$ is the length of a vector
for $i = 1, \ldots, |\vec{a}|$ **do**
  if a random number in range $[0, 1] \leq 0.5$ **then**
    $\vec{c}_i = \vec{a}_i$
    $\vec{d}_i = \vec{b}_i$
  else
    $\vec{c}_i = \vec{b}_i$
    $\vec{d}_i = \vec{a}_i$
  end if
end for
  ▷ If $|\vec{b}| > |\vec{a}|$, add remaining elements to $\vec{d}$. Note that typically, $|\vec{b}| = |\vec{a}|$.
return $\vec{c}$ and $\vec{d}$ concatenated with $\vec{b}_{|\vec{a}|+1,\ldots,|\vec{b}|}$

3.1.3 GA Mutation

While crossover can generate previously unexplored vectors, it cannot generate new elements for a vector. For example, given a population of 3 binary vectors $\langle 0, 1, 1\rangle$, $\langle 0, 1, 0\rangle$, and $\langle 0, 0, 1\rangle$, crossover will never generate a vector with 1 as the first element, such as $\langle 1, 1, 1\rangle$. Mutation fills this gap by randomly modifying elements of a vector.

Bit flip mutation [139, 140] examines every element of a binary vector and with probability p_b, flips the bit from 0 to 1 or 1 to 0. Conversely, with probability $1-p_b$ the element remains unchanged. The bit flip mutation procedure is given in Algorithm 13. Bit flip mutation is an effective and simple mechanism for mutating binary vectors.

Algorithm 13 Bit Flip Mutation

Given bit flip chance p_b
and vector $\vec{x}$
for $i = 1, \ldots, |\vec{x}|$ **do** ▷ where $|\cdot|$ is the length of a vector
  if a random number in range $[0, 1] \leq p_b$ **then**
    $\vec{x}_i = 1 - \vec{x}_i$
  end if
end for

A relatively low mutation chance $p_b \approx 0.05$ is recommended to avoid negating the objective value improving effect of selection.

For real valued vectors, an alternative to bit flip mutation is necessary. Gaussian mutation [141] examines every element of a real valued vector and with probability p_g, adds a randomly generated real number drawn from a Gaussian distribution with mean 0 and variance v, to the element. The Gaussian mutation procedure is given in Algorithm 14. As with bit flip mutation, a relatively low mutation chance $p_g \approx 0.05$ is

Algorithm 14 Gaussian Mutation

Given mutation chance p_g,
Gaussian variance v,
and vector $\vec{x}$
for $i = 1, \ldots, |\vec{x}|$ **do** ▷ where $|\cdot|$ is the length of a vector
  if a random number in range $[0, 1] \leq p_g$ **then**
    $\vec{x}_i = \vec{x}_i +$ a Gauss random number with mean 0 and variance v
  end if
end for

recommended. Variance v depends largely on problem. If small changes in parameter values leads to large changes in objective value, v should be small. Note that higher p_g can be offset with lower v.

3.2 Population-Based Incremental Learning

Population-based incremental learning (PBIL) repeatedly samples random binary vectors from a probability distribution $\vec{p}$, where each element of $\vec{p}$ is the probability that the corresponding element of a sampled binary vector is 1. Each iteration, a number of parameter vector samples n_s are generated and $\vec{p}$ is adjusted closer to the best parameter vector among samples at rate α, increasing the probability of generating similar vectors in subsequent iterations. To discourage premature convergence, elements of $\vec{p}$ are randomly mutated with probability p_m. When an element of $\vec{p}$ mutates, it is adjusted towards a random number in range $[0, 1]$ at rate α_m.

The PBIL procedure is given in Algorithm 15. As with GA, larger parameter vectors require more samples per iteration because of the larger problem space. Unlike GA, PBIL does not rely on an initial population for diversity and is effective with smaller numbers of samples. A relatively low adjustment rate $\alpha \approx 0.1$ is recommended to avoid premature convergence, allowing greater exploration of the problem space before honing in on local optima. A mutation chance $p_m = {}^1/_{n_p}$, where n_p is the number of parameters, is recommended for an average of 1 probability element adjusted per iteration. A low mutation rate $\alpha_m \approx {}^{\alpha}/_2$ is recommended to avoid negating the objective value improving effect of adjusting probabilities towards high value parameter vectors.

PBIL provides a simple and effective alternative to derivative-free optimizers like GA. Despite the ease of implementing PBIL, it proves an efficient optimizer in practice, frequently outperforming more complicated optimizers like GA. PBIL does not rely on a population of parameter vectors like GA or adjustments to an individual vector like gradient descent, thereby allowing PBIL to more easily avoid local optima and the low diversity traps that GA is susceptible to. PBIL is a good first choice for derivative-free optimization because of its ease of implementation, simple hyperparameters, and robust effectiveness.

Algorithm 15 Population-Based Incremental Learning

```
Given number of parameters n_p,
Given number of samples per iteration n_s,
probability adjustment rate α,
mutation chance p_m,
mutation probability adjustment rate α_m,
and objective function f

for i = 1, ..., n_p do                                ▷ Initialize p⃗ with uniform probability
    p⃗_i = 0.5
end for
while a stopping criteria is not met do               ▷ Main optimization loop
    for i = 1, ..., n_s do                            ▷ Generate random samples
        for j = 1, ..., n_p do                        ▷ Generate a random binary vector
            if a random number in range [0, 1] ≤ p⃗_j then
                S_ij = 1
            else
                S_ij = 0
            end if
        end for
    end for
                                                      ▷ Adjust p⃗ towards best parameter vector
    p⃗ ← (1 − α)p⃗ + α arg min_{x⃗=S_1,...,S_{n_s}} f(x⃗)    ▷ arg min for minimization problem

    for i = 1, ..., n_p do                            ▷ Mutate p⃗
        if a random number in range [0, 1] ≤ p_m then
            p⃗_i = (1 − α_m)p⃗_i + α_m × a random number in range [0, 1]
        end if
    end for
end while

return Vector x⃗ with best f(x⃗) among all samples in all iterations
```

3.3 Gravitational Search Algorithm

Gravitational search algorithm (GSA) is inspired by the gravitational motion of bodies in free space, as dictated by Newton's laws. Each iteration, small adjustments are made to an initially random population of vectors, as opposed to generating completely new solutions from iteration to iteration as GA and PBIL does. GSA draws low value vectors towards high value vectors as a form of exploitation, as depicted in Figure 3.2.

In GSA, every parameter vector has a mass proportional to its objective value. As in natural physics, bodies attract one another, higher mass bodies attract more and are less affected by the attraction of other bodies. This mechanism alone can quickly cause all bodies to converge towards the best value bodies. Unless global optima lies along the path of one of these bodies, this results in premature convergence to a local optima. To encourage exploration of the problem space, the summation of force on a body and the velocity update for a body is randomized. Specifically, the total force on a body in one dimension is given as

$$\boldsymbol{F}_{ij}(t) = \sum_{b=1}^{n_b} \zeta \boldsymbol{F}_{bj}(i,t) \text{ where } b \neq i \tag{3.2}$$

where $\boldsymbol{F}_{ij}(t)$ is the total force on body i in dimension j at time step t, ζ is a random number in range $[0,1]$, $\boldsymbol{F}_{bj}(i,t)$ is the force of body b on body i in dimension j at time step t, and n_b is the number of bodies. As such, the force that each body applies to every other body in each dimension is randomized, as depicted in Figure 3.3.

The velocity update is likewise randomized for each body by taking a random fraction of its velocity in each dimension and adding its acceleration in that dimension

$$V_{ij}(t+1) = \zeta V_{ij}(t) + A_{ij}(t) \tag{3.3}$$

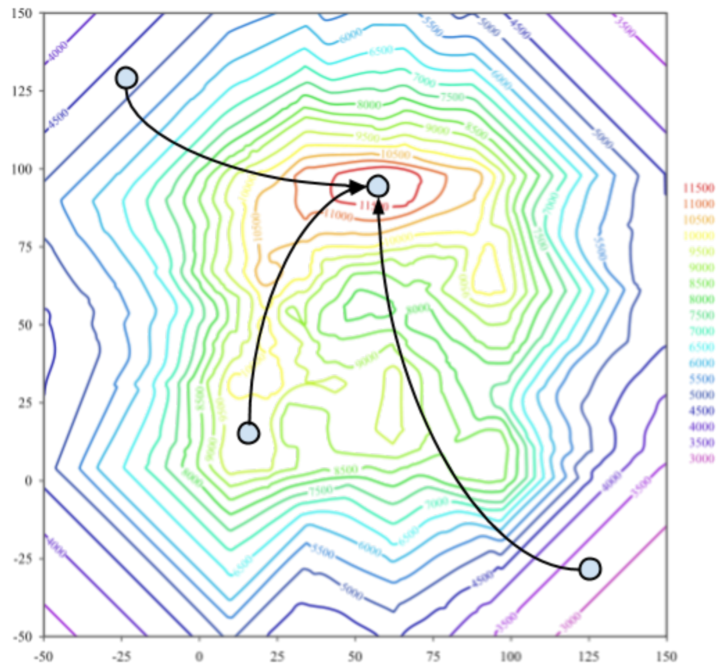

Figure 3.2: Vectors drawn towards high objective value in GSA [142]

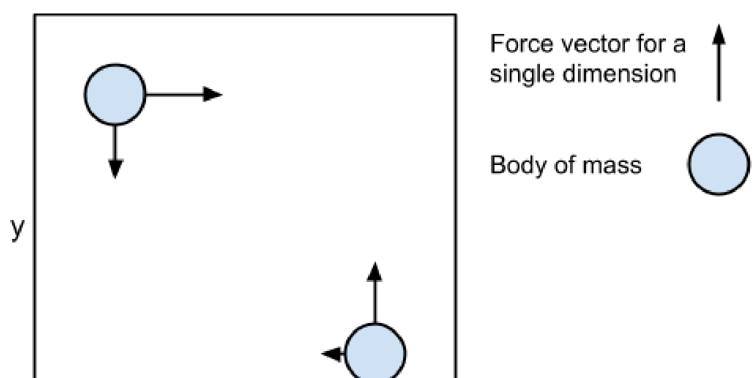

Figure 3.3: Movement of bodies in GSA — Two bodies of equal mass, equidistant in the x and y dimension, are depicted. The force for each dimension is equal according to Newton's laws, unlike GSA [135].

where $V_{ij}(t)$ is the velocity vector of body i in dimension j at time step t, ζ is a random number in range $[0, 1]$, and $A_{ij}(t)$ is the acceleration vector of body i in dimension j at time step t. As such, bodies may randomly stop or slow down in any direction between time steps.

The GSA optimization procedure, containing further details, is given in Algorithm 16. A high initial gravity $g_0 \approx 100$ is recommended to offset low initial velocity. As bodies gain momentum, gravity can rapidly decrease at a gravity reduction rate $\alpha \approx 20$. Lower and upper bounds are problem specific. The narrowest bounds that are likely to contain an optimal solution should be chosen. As with optimizers like GA and PBIL, larger parameter vectors require more bodies to explore the larger problem space.

GSA is designed to excel in real value parameter spaces. In non-smooth and binary problem spaces, GSA will suffer. Note that binary variants of GSA exist, but are not explored in this work [123].

Algorithm 16 Gravitational Search Algorithm

Given number of parameters n_p,
number of bodies n_b,
a set of lower bounds $l_1, \ldots, l_{n_p}$
a set of upper bounds $u_1, \ldots, u_{n_p}$
initial gravity g_0
gravity reduction rate α
max time T ▷ Also known as max iterations
and objective function f

for $i = 1, \ldots, n_b$ **do** ▷ Initialize bodies $\boldsymbol{B}$, a.k.a. parameter vectors
    for $j = 1, \ldots, n_p$ **do** ▷ Initialize body $\boldsymbol{B}_i$
        $\boldsymbol{B}_{ij} \leftarrow$ a random number in range $[l_j, u_j]$
    end for
end for
for $i = 1, \ldots, n_b$ **do** ▷ Initialize body velocities V
    $\boldsymbol{V}_i \leftarrow \vec{0}$ ▷ Initialize veclocity to vector containing n_p 0 elements
end for
$t = 1$ ▷ Also known as iteration
while a stopping criteria is not met **do** ▷ Main optimization loop
    $g \leftarrow g_0 e^{-\alpha \frac{t}{T}}$ ▷ Update gravity g, where e is Euler's number
    $\vec{m} \leftarrow$ SCALED_MASSES($\boldsymbol{B}$, f) ▷ See Algorithm 17
    $\boldsymbol{F} \leftarrow$ FORCE_VECTORS(g, $\boldsymbol{B}$, $\vec{m}$, $\lfloor n_b - (n_b - 1)\frac{t}{T} \rfloor$) ▷ See Algorithm 17
    for $i = 1, \ldots, n_b$ **do** ▷ Get acceleration vector applied to each body
        $\boldsymbol{A}_i \leftarrow \frac{\boldsymbol{F}_i}{\vec{m}_i}$
    end for
    for $i = 1, \ldots, n_b$ **do** ▷ Update velocity of each body
        for $j = 1, \ldots, n_p$ **do**
            $\zeta \leftarrow$ a random number in range $[0, 1]$
            $\boldsymbol{V}_{ij} \leftarrow \zeta \boldsymbol{V}_{ij} + \boldsymbol{A}_{ij}$
        end for
    end for
    for $i = 1, \ldots, n_b$ **do** ▷ Update position of each body
        $\boldsymbol{B}_i \leftarrow \boldsymbol{B}_i + \boldsymbol{V}_i$
        for $j = 1, \ldots, n_p$ **do** ▷ Constrain body to lower and upper bounds
            $\boldsymbol{B}_{ij} \leftarrow \max(l_j, \min(u_j, \boldsymbol{B}_{ij}))$
        end for
    end for
    $t = t + 1$ ▷ Increment time step
end while

return $\arg\min_{\boldsymbol{B}_i \in \boldsymbol{B}_1, \ldots, \boldsymbol{B}_{n_b}} f(\boldsymbol{B}_i)$ ▷ Return body with best objective value

Algorithm 17 Gravitational Search Algorithm Procedures

```
function SCALED_MASSES(bodies B, objective function f)
    n_b ← number of bodies and rows in B
    for i = 1, ..., n_b do
        f_i ← −f(B_i)                                  ▷ Negative for minimization problem
    end for
    f_min ← min(f_1, ..., f_{n_b})
    f_max ← max(f_1, ..., f_{n_b})
    for i = 1, ..., n_b do                             ▷ Get masses from objective values
        m⃗_i ← (f_i − f_min) / (f_max − f_min)
    end for
    m⃗_sum ← Σ_{i=1}^{n_b} m⃗_i
    for i = 1, ..., n_b do                             ▷ Scale masses to a sum of 1
        m⃗_i ← m⃗_i / m⃗_sum
    end for
    return m⃗
end function

function FORCE_VECTORS(gravity g, bodies B, masses m⃗, number of bodies to
apply force k)
    n_b ← number of bodies and rows in B
    n_p ← number of parameters in each body
    for i = 1, ..., n_b do
        F_i ← 0⃗                         ▷ Initialize force vector on each body to 0 vector
    end for
    for i = 1, ..., n_b do                         ▷ Get force vector applied to each body
        for b ∈ indices of k bodies with greatest mass do
            if b ≠ i then
                for j = 1, ..., n_p do                 ▷ Apply force from k best bodies
                    ζ ← a random number in range [0, 1]
                    F_ij ← F_ij + ζ g (m⃗_b m⃗_i)/(‖B_b − B_i‖_2 + ϵ) (B_bj − B_ij)   ▷ where ϵ is a small scalar
                end for
            end if
        end for
    end for
    return F
end function
```

Chapter 4

Supervised Learning

Supervised learning methods can be decomposed into a model mapping arguments to predictions and a training method for adjusting the parameters of this model to maximize accuracy or minimize error. The model component is easily represented as a function f. When predictions $\hat{\boldsymbol{Y}}$ are required, $f(\boldsymbol{X}) = \hat{\boldsymbol{Y}}$, where $\boldsymbol{X}$ is a matrix of model arguments. Once a model is represented as a function, training is a simple application of mathematical optimization: $\arg\min_P \xi(f(\boldsymbol{X}), \boldsymbol{Y})$, where P is a set of adjustable model parameters, $\boldsymbol{X}$ is a matrix of training arguments, $\boldsymbol{Y}$ is a matrix of training targets, and ξ is an error function such as mean squared error. Note that P may need linearization into a vector for application with most optimizers, such as gradient optimizers. Parameters for our examined models are presented in their natural form but may need linearization when implemented.

Any set of parameters can be linearized without affecting performance because, with rare exception, an optimizer does not depend on the order or configuration of elements in its parameter vector. Even when an optimizer is affected by the configuration of its parameters, such as a genetic algorithm, parameters can be arranged to compliment the optimizer. To demonstrate linearization, given weight matrix $\boldsymbol{W}$ and bias vector $\vec{b}$, these parameters can be linearized as

$$\left\langle \boldsymbol{W}_{11}, \boldsymbol{W}_{12}, \ldots, \boldsymbol{W}_{1n}, \boldsymbol{W}_{21}, \boldsymbol{W}_{22}, \ldots, \boldsymbol{W}_{2n}, \ldots, \boldsymbol{W}_{mn}, \vec{b}_1, \vec{b}_2, \ldots, \vec{b}_m \right\rangle$$

where matrix subscript ij denote the element in the i^{th} row and j^{th} column, and vector subscript i denote the i^{th} element.

4.1 Regression

Regression in its pure simplicity is an excellent example of supervised learning from the perspective of mathematical optimization. Each regression variant is a model described by a function mapping an argument matrix $\boldsymbol{X}$ to an output matrix $\hat{\boldsymbol{Y}}$. The regression model contains a number of parameters that can be adjusted to improve the models fit on a given dataset. These parameters are typically portrayed as a weight matrix and bias vector. Some presentations of regression take an argument vector and return a single scalar, which requires a weight vector and bias scalar. Note that most regression models can replace bias parameters with extra weight parameters by including an extra element with a constant value of 1 in the argument vector of each pattern, a.k.a. row of $\boldsymbol{X}$.

4.1.1 Linear Regression

A linear regression model [8, 9, 10] is a function

$$f(\boldsymbol{X}) = \boldsymbol{X}\boldsymbol{W} + \vec{b} \tag{4.1}$$

where $\boldsymbol{W}$ is a $n_x \times n_y$ weight matrix and $\vec{b}$ is a bias vector with n_y elements, where n_x is the number of columns in $\boldsymbol{X}$ and attributes in the dataset and n_y is the number of columns in $\boldsymbol{Y}$. This model is trained by optimizing the training objective

$$f_t(\boldsymbol{X}, \boldsymbol{Y}) = \xi(f(\boldsymbol{X}), \boldsymbol{Y}) \tag{4.2}$$

with regard to model parameters $\boldsymbol{W}$ and $\vec{b}$:

$$\underset{\boldsymbol{W},\vec{b}}{\arg\min}\, f_t(\boldsymbol{X}, \boldsymbol{Y}) \tag{4.3}$$

where $\boldsymbol{X}$ is a matrix of training arguments, $\boldsymbol{Y}$ is a matrix of training targets, and ξ is an error function such as mean squared error. That is to say, this linear regression model is trained by adjusting its parameters with the goal of minimizing error between the result of f on a matrix of training arguments $\boldsymbol{X}$ and corresponding training targets $\boldsymbol{Y}$. When the model has low error, we say it is well trained an has converged on its given dataset.

While many optimizers function effectively without derivative information, providing the derivative of the optimization problem (4.2) allows for impressively effective optimization through gradient based optimizers such as BFGS. The derivatives of the linear regression training objective (4.2) with regard to model parameters are:

$$\frac{df_t}{d\boldsymbol{W}} = \boldsymbol{X}^T \xi'(f(\boldsymbol{X}), \boldsymbol{Y}) = \boldsymbol{X}^T \xi'(\boldsymbol{X}\boldsymbol{W} + \vec{b}, \boldsymbol{Y}) \tag{4.4}$$

and

$$\frac{df_t}{d\vec{b}} = \sum_{i=1}^{m} \xi'(f(\boldsymbol{X}), \boldsymbol{Y})_i = \sum_{i=1}^{m} \xi'(\boldsymbol{X}\boldsymbol{W} + \vec{b}, \boldsymbol{Y})_i \tag{4.5}$$

where ξ' is the derivative of the models error function and matrix subscript i selects the i^{th} row of the matrix. Note that ξ' is a Jacobian and is further explored in Section 4.4. Furthermore, because $\xi'(X)$ is a scalar function, we ignore the first dimension of its Jacobian, and consider the Jacobian an $m_y \times n_y$ matrix, where m_y is the number of rows in $\boldsymbol{Y}$ and n_y is the number of columns in $\boldsymbol{Y}$.

4.1.2 Logistic Regression

Logistic regression is identical to linear regression with the exception of the function is represents:

$$f(\boldsymbol{X}) = \frac{1}{1 + e^{-(\boldsymbol{XW}+\vec{b})}} \tag{4.6}$$

where e is Euler's number, and the matrix exponential $e^{\boldsymbol{M}}$ is defined as an element-wise exponential on a matrix $\boldsymbol{M}$. Effectively, logistic regression is a linear regression model $\boldsymbol{XW} + \vec{b}$ passed to a logit function $\text{logit}(x) = \frac{1}{1+e^{-x}}$, resulting in $\text{logit}(\boldsymbol{XW} + \vec{b}) = \frac{1}{1+e^{-(\boldsymbol{XW}+\vec{b})}}$. Training of a logistic regression model is the same as linear regression, using equations (4.2) and (4.3), differing only by function f. The derivatives of the logistic regression training objective with regard to model parameters are:

$$\frac{df_t}{d\boldsymbol{W}} = \boldsymbol{X}^T f'(\boldsymbol{X}) \odot \xi'(f(\boldsymbol{X}), \boldsymbol{Y}) = \boldsymbol{X}^T \frac{e^{\boldsymbol{XW}+\vec{b}}}{\left(1 + e^{\boldsymbol{XW}+\vec{b}}\right)^2} \odot \xi' \left(\frac{1}{1 + e^{-(\boldsymbol{XW}+\vec{b})}}, \boldsymbol{Y} \right) \tag{4.7}$$

and

$$\frac{df_t}{d\vec{b}} = \sum_{i=1}^{m} \left(f'(\boldsymbol{X}) \odot \xi'(f(\boldsymbol{X}), \boldsymbol{Y})\right)_i = \sum_{i=1}^{m} \left(\frac{e^{\boldsymbol{XW}+\vec{b}}}{\left(1 + e^{\boldsymbol{XW}+\vec{b}}\right)^2} \odot \xi' \left(\frac{1}{1 + e^{-(\boldsymbol{XW}+\vec{b})}}, \boldsymbol{Y} \right) \right)_i \tag{4.8}$$

where $\odot$ is a Hadamard, a.k.a., element-wise, product. Note that both derivatives have $\frac{e^{\boldsymbol{XW}+\vec{b}}}{\left(1+e^{\boldsymbol{XW}+\vec{b}}\right)^2} \odot \xi' \left(\frac{1}{1+e^{-(\boldsymbol{XW}+\vec{b})}}, \boldsymbol{Y} \right)$, the majority of each equation, in common, allowing for improved performance when calculated together.

While linear regression is effective for regression problems, where training targets can be a wide range of continuous values, logistic regression is effective for classification problems where targets can be represented as one-hot vectors such as $\langle 1, 0, 0 \rangle$. The logit function constrains outputs to the range $[0, 1]$, allowing logistic regression to easily match one-hot vectors.

4.2 Multilayer Perceptron

The popular multilayer perceptron (MLP) [1, 14, 15, 16, 17, 18] neural network is a staple of supervised learning. An MLP is a function of the form

$$f(\boldsymbol{X}) = t_{n-1}(\cdots(t_2(t_1(\boldsymbol{X}\boldsymbol{W}_1 + \vec{b})\boldsymbol{W}_2)\cdots)\boldsymbol{W}_{n-1}) \tag{4.9}$$

where t_i is the i^{th} transfer function and $\boldsymbol{W}_i$ is the i^{th} weight matrix. MLP is typically described as having layers of neurons, which can be presented with a shape hyperparameter $(s_0, s_1, \ldots, s_{n-1})$, where n is the number of layers. The shape hyperparameter directly corresponds to the shape of each layers output: s_i corresponds to the number of columns in the matrix outputted by the i^{th} layer; s_0 is the number of columns in $\boldsymbol{X}$ and the number of attributes accepted by the model; and s_n is the number of columns in $\boldsymbol{Y}$ and the number of targets values outputted by the model. To support these layer outputs, the number of rows in $\boldsymbol{W}_i$ is s_{i-1} and the number of columns in $\boldsymbol{W}_i$ is s_i. Additionally, $\vec{b}$ has s_1 elements.

As with logistic regression, training an MLP model is the same as linear regression, using equation (4.2) and solving

$$\begin{aligned} \underset{\boldsymbol{W}_1,\boldsymbol{W}_2,\ldots,\boldsymbol{W}_{n-1},\vec{b}}{\arg\min}\ & f_t(\boldsymbol{X}, \boldsymbol{Y}) = \\ & \underset{\boldsymbol{W}_1,\boldsymbol{W}_2,\ldots,\boldsymbol{W}_{n-1},\vec{b}}{\arg\min}\ \xi(t_{n-1}(\cdots(t_2(t_1(\boldsymbol{X}\boldsymbol{W}_1 + \vec{b})\boldsymbol{W}_2)\cdots)\boldsymbol{W}_{n-1}), \boldsymbol{Y}) \end{aligned} \tag{4.10}$$

Note that (4.10) differs only from the linear regression optimization problem (4.3) by the parameters they optimize. The derivatives of the MLP training objective with regard to model parameters are:

$$
\begin{aligned}
\frac{df_t}{d\boldsymbol{W}_{n-1}} &= t_{n-2}(\cdots(t_1(\boldsymbol{X}\boldsymbol{W}_1+\vec{b})\cdots)\boldsymbol{W}_{n-2})\xi'(t_{n-1}(\cdots(t_1(\boldsymbol{X}\boldsymbol{W}_1+\vec{b})\cdots)\boldsymbol{W}_{n-1}),Y) \\
&\qquad t'_{n-1}(\cdots(t_1(\boldsymbol{X}\boldsymbol{W}_1+\vec{b})\cdots)\boldsymbol{W}_{n-1}) \\
\frac{df_t}{d\boldsymbol{W}_{n-2}} &= t_{n-3}(\cdots(t_1(\boldsymbol{X}\boldsymbol{W}_1+\vec{b})\cdots)\boldsymbol{W}_{n-3})\xi'(t_{n-1}(\cdots(t_1(\boldsymbol{X}\boldsymbol{W}_1+\vec{b})\cdots)\boldsymbol{W}_{n-1}),Y) \\
&\qquad t'_{n-1}(\cdots(t_1(\boldsymbol{X}\boldsymbol{W}_1+\vec{b})\cdots)\boldsymbol{W}_{n-1})\boldsymbol{W}_{n-1}^T t'_{n-2}(\cdots(t_1(\boldsymbol{X}\boldsymbol{W}_1+\vec{b})\cdots)\boldsymbol{W}_{n-2}) \\
&\cdots \\
\frac{df_t}{d\boldsymbol{W}_1} &= \boldsymbol{X} \\
&\quad \xi'(t_{n-1}(\cdots(t_1(\boldsymbol{X}\boldsymbol{W}_1+\vec{b})\cdots)\boldsymbol{W}_{n-1}),Y)t'_{n-1}(\cdots(t_1(\boldsymbol{X}\boldsymbol{W}_1+\vec{b})\cdots)\boldsymbol{W}_{n-1}) \\
&\quad \boldsymbol{W}_{n-1}^T t'_{n-2}(\cdots(t_1(\boldsymbol{X}\boldsymbol{W}_1+\vec{b})\cdots)\boldsymbol{W}_{n-2}) \\
&\quad \cdots \\
&\quad \boldsymbol{W}_2^T t'_1(\boldsymbol{X}\boldsymbol{W}_1+\vec{b}) \\
\frac{df_t}{d\vec{b}} &= \sum_{i=1}^{m}\Bigg(\xi'(t_{n-1}(\cdots(t_1(\boldsymbol{X}\boldsymbol{W}_1+\vec{b})\cdots)\boldsymbol{W}_{n-1}),Y) \\
&\qquad t'_{n-1}(\cdots(t_1(\boldsymbol{X}\boldsymbol{W}_1+\vec{b})\cdots)\boldsymbol{W}_{n-1}) \\
&\qquad \boldsymbol{W}_{n-1}^T t'_{n-2}(\cdots(t_1(\boldsymbol{X}\boldsymbol{W}_1+\vec{b})\cdots)\boldsymbol{W}_{n-2}) \\
&\qquad \cdots \\
&\qquad \boldsymbol{W}_2^T t'_1(\boldsymbol{X}\boldsymbol{W}_1+\vec{b})\Bigg)_i
\end{aligned}
\tag{4.11}
$$

However, naive calculation of these derivatives requires a significant amount of computation. Thankfully, all of these derivatives share many common components that lead to the development of the backpropagation algorithm [143, 144], given in Algorithm 18.

Algorithm 18 MLP Backpropagation

Given argument matrix $\boldsymbol{X}$,
target matrix $\boldsymbol{Y}$,
error function ξ,
a set of $n-1$ weight matrices $\boldsymbol{W}_1, \ldots, \boldsymbol{W}_{n-1}$
and bias vector $\vec{b}$

$\boldsymbol{N}_0 \leftarrow \boldsymbol{X}$
$\boldsymbol{T}_1 \leftarrow \boldsymbol{X}\boldsymbol{W}_1 + \vec{b}$
$\boldsymbol{N}_1 \leftarrow t_1(\boldsymbol{T}_1)$
$i \leftarrow 2$
▷ Calculate output of MLP $f(\boldsymbol{X})$ while saving intermediary operations
while $i < n$ **do**
  $\boldsymbol{T}_i \leftarrow \boldsymbol{N}_i\boldsymbol{W}_i$
  $\boldsymbol{N}_{i+1} \leftarrow t_i(\boldsymbol{T}_i)$
  $i \leftarrow i + 1$
end while

$\boldsymbol{J}_{n-1} \leftarrow \xi(\boldsymbol{N}_n, \boldsymbol{Y})t'_{n-1}(\boldsymbol{T}_{n-1})$ ▷ $\boldsymbol{N}_n = f(\boldsymbol{X})$
$i \leftarrow n - 2$
while $i > 0$ **do** ▷ Calculate common components of each weight matrix derivative
  $\boldsymbol{J}_i \leftarrow \boldsymbol{J}_{i+1}\boldsymbol{W}_{i+1}^T t'_i(\boldsymbol{T}_i)$
  $i \leftarrow i - 1$
end while

$i \leftarrow 0$
while $i < n$ **do** ▷ Finalize derivatives with non-common components
  $\frac{d}{d\boldsymbol{W}_i} \leftarrow \boldsymbol{N}_{i-1}^T\boldsymbol{J}_i$
  $i \leftarrow i + 1$
end while
$\frac{d}{d\vec{b}} \leftarrow \sum_{i=1}^{m} \boldsymbol{J}_{1i}$ ▷ Sum rows of $\boldsymbol{J}_1$

Backpropagation efficiently calculates all MLP derivatives (4.11) through a three pass strategy. The first pass calculates the MLP output $f(\boldsymbol{X})$ while caching the argument and output of each transfer function for use in the second and third passes. The second pass calculates the common components of each derivative:

$$\begin{aligned}
\boldsymbol{J}_{n-1} &= \xi'(t_{n-1}(\cdots(t_1(\boldsymbol{X}\boldsymbol{W}_1+\vec{b})\cdots)\boldsymbol{W}_{n-1}),Y)t'_{n-1}(\cdots(t_1(\boldsymbol{X}\boldsymbol{W}_1+\vec{b})\cdots)\boldsymbol{W}_{n-1}) \\
\boldsymbol{J}_{n-2} &= \xi'(t_{n-1}(\cdots(t_1(\boldsymbol{X}\boldsymbol{W}_1+\vec{b})\cdots)\boldsymbol{W}_{n-1}),Y)t'_{n-1}(\cdots(t_1(\boldsymbol{X}\boldsymbol{W}_1+\vec{b})\cdots)\boldsymbol{W}_{n-1}) \\
&\qquad \boldsymbol{W}^T_{n-1}t'_{n-2}(\cdots(t_1(\boldsymbol{X}\boldsymbol{W}_1+\vec{b})\cdots)\boldsymbol{W}_{n-2}) \\
&\cdots \\
\boldsymbol{J}_{1} &= \xi'(t_{n-1}(\cdots(t_1(\boldsymbol{X}\boldsymbol{W}_1+\vec{b})\cdots)\boldsymbol{W}_{n-1}),Y)t'_{n-1}(\cdots(t_1(\boldsymbol{X}\boldsymbol{W}_1+\vec{b})\cdots)\boldsymbol{W}_{n-1}) \\
&\qquad \boldsymbol{W}^T_{n-1}t'_{n-2}(\cdots(t_1(\boldsymbol{X}\boldsymbol{W}_1+\vec{b})\cdots)\boldsymbol{W}_{n-2})\cdots\boldsymbol{W}^T_2 t'_1(\boldsymbol{X}\boldsymbol{W}_1+\vec{b})
\end{aligned}$$

Note that only the series of layer arguments

$$\boldsymbol{X}, t_1(\boldsymbol{X}\boldsymbol{W}_1+\vec{b}), t_2(t_1(\boldsymbol{X}\boldsymbol{W}_1+\vec{b})\boldsymbol{W}_2), \ldots, t_{n-2}(\cdots(t_1(\boldsymbol{X}\boldsymbol{W}_1+\vec{b})\cdots)\boldsymbol{W}_{n-2})$$

are missing from these incomplete derivatives. The third and final pass multiplies these missing components with corresponding incomplete derivatives. The bias vector $\vec{b}$ instead multiplies a vector of ones $\vec{1}$ with $\boldsymbol{J}_1$, which is equivalent to a summation of rows $\sum_{i=1}^{m}\boldsymbol{J}_{1i}$.

Without transfer functions, MLP would reduce to a linear regression model. The only requirement for a transfer function is non-linearity. However, useful transfer functions are also continuous and differentiable. Furthermore, choice of transfer functions can greatly change the prior or a MLP model and thereby affect accuracy and training time. The state-of-the-art in MLP transfer functions use softplus [145, 146, 147] for transfer layers other than the last, a linear or lack of transfer function for the last layer

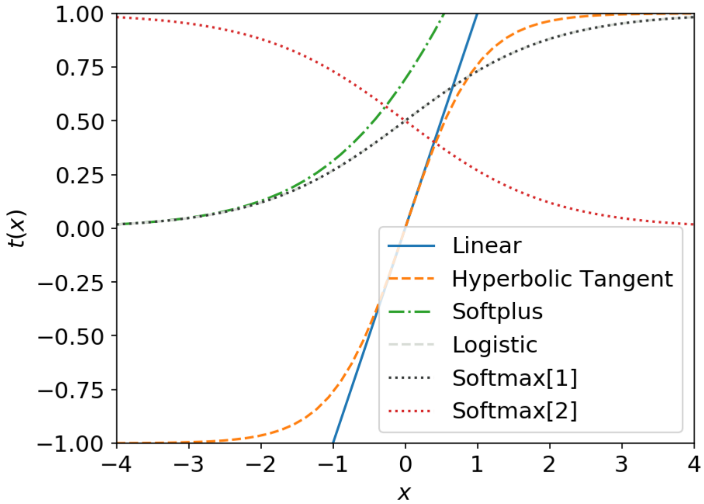

Figure 4.1: Transfer functions — Output of linear, logistic, hyperbolic tangent, softplus, and softmax transfer functions. Softmax is calculated on a $[x, 0]$ vector, and each element of softmax(x) is depicted on the y axis as dictated by [coordinate]. Note that logistic and the first element of softmax overlap for the given softmax vector.

on regression problems, and softmax transfer [148] for the last layer on classification problems. Logistic and hyperbolic tangent transfer functions have been historically popular but have recently lost popularity in favor of softplus. These transfer functions are depicted in Figure 4.1. Note that, when applied to a matrix, softmax is applied to each row of the matrix and other transfer function are applied to each element of the matrix.

4.3 Radial Basis Function Network

A radial basis function network (RBF) [27, 28, 29, 30, 31, 32] differentiates itself from models like linear regression and MLP through the addition of clustering.

During training the model domain is clustered using $\boldsymbol{X}$, providing a matrix of cluster centers $\boldsymbol{C}$. When an RBF network is activated, the similarity between each center $\vec{c} \in \boldsymbol{C}$ and each argument vector $\vec{x} \in \boldsymbol{X}$ is calculated by a similarity function ϕ, giving a similarity matrix $\boldsymbol{S} = \phi(\boldsymbol{X}, \boldsymbol{C})$, where each row $\boldsymbol{S}_i$ in $\boldsymbol{S}$ is the similarity between each $\vec{c} \in \boldsymbol{C}$ and $\boldsymbol{X}_i$. The RBF output is given by

$$f(\boldsymbol{X}) = \phi(\boldsymbol{X}, \boldsymbol{C})\boldsymbol{W} + \vec{b} \tag{4.12}$$

Note the resemblance to linear regression. An RBF model is effectively a combination of clustering and linear regression. More precisely, the output component of RBF is a linear regression model using outputs of the clustering component as arguments.

A similarity function is a radial basis function $\phi(\cdot, \cdot)$ that satisfies the condition

$$\phi(\vec{a}_1, \vec{b}) \geq \phi(\vec{a}_2, \vec{b}) \text{ for all } \vec{a}_1, \vec{a}_2, \vec{b} \in \mathbb{R}^n \text{ given } \|\vec{a}_1 - \vec{b}\| < \|\vec{a}_2 - \vec{b}\| \tag{4.13}$$

A similarity function does not decrease in value when distance between its arguments $\vec{a}$ and $\vec{b}$ decreases. Useful, but not necessary, conditions for a similarity function are: it is continuous; it monotonically increases in value as distance decreases. A similarity function is maximized when $\|\vec{a} - \vec{b}\| = 0$ and minimized when $\|\vec{a} - \vec{b}\| = \infty$. Note that, as a radial basis function, a similarity function also has the condition that its value depends only on the distance between its arguments.

A popular similarity function is the Gaussian

$$\phi(\vec{a}, \vec{b}) = e^{-(\frac{\|\vec{a}-\vec{b}\|^2}{v})} \tag{4.14}$$

where v is a variance parameter. With this Gaussian similarity function, $\phi(\vec{a}, \vec{b})$ approaches 1 as the distance between $\vec{a}$ and $\vec{b}$ approaches 0. Conversely, $\phi(\vec{a}, \vec{b})$ rapidly approaches 0 as distance approaches ∞.

Normalizing similarity between an $\vec{x} \in \boldsymbol{X}$ and each $\vec{c} \in \boldsymbol{C}$ to a sum of 1 improves the consistency of a similarity function. A Normalized similarity function, called on

$\vec{x}$ and $\vec{c}$

$$\phi_n(\vec{x}, \vec{c}) = \frac{\phi(\vec{x}, \vec{c})}{\sum_{\vec{c} \in \boldsymbol{C}} \phi(\vec{x}, \vec{c})} \tag{4.15}$$

divides each similarity by the sum of all relevant similarities. Normalizing similarity can improve generalization performance by providing a smooth transition between cluster centers.

Training RBF requires two separate phases and optimizers. The first phase performs clustering on $\boldsymbol{X}$ to obtain $\boldsymbol{C}$. The number of rows in $\boldsymbol{C}$ is an RBF hyperparameter that determines the number of columns in $\boldsymbol{S} = \phi(\boldsymbol{X}, \boldsymbol{C})$ and effectively reduces dimensionality for the subsequent regression operation.

The second phase is a linear regression model that replaces $\boldsymbol{X}$ with $\phi(\boldsymbol{X}, \boldsymbol{C})$. As with linear regression, the second phase of RBF is trained using equations (4.2) and (4.3), differing only by function f. The derivatives of the RBF training objective with regard to $\boldsymbol{W}$ and $\vec{b}$ are:

$$\frac{df_t}{d\boldsymbol{W}} = \boldsymbol{X}^T \xi'(f(\boldsymbol{X}), \boldsymbol{Y}) = \boldsymbol{X}^T \xi'(\phi(\boldsymbol{X}, \boldsymbol{C})\boldsymbol{W} + \vec{b}, \boldsymbol{Y}) \tag{4.16}$$

and

$$\frac{df_t}{d\vec{b}} = \sum_{i=1}^{m} \xi'(f(\boldsymbol{X}), \boldsymbol{Y})_i = \sum_{i=1}^{m} \xi'(\phi(\boldsymbol{X}, \boldsymbol{C})\boldsymbol{W} + \vec{b}, \boldsymbol{Y})_i \tag{4.17}$$

which are the linear regression derivatives (4.4) and (4.5) with $\boldsymbol{X}$ replaced by $\phi(\boldsymbol{X}, \boldsymbol{C})$.

4.4 Error Functions

Under the mathematical optimization perspective, an error function ξ is an independent and reusable component of the supervised learning training process. An error function used for the optimization of one model is easily applied to another, without modification. This has the substantial benefits of generating reusable code and allowing extension of existing models by devising new error functions, as is seen in Section 4.5.

Note that classification problems are often formulated with discrete labels for targets. When applying numerical methods, like optimization, it is often useful or necessary to convert these discrete labels into one-hot vectors. A one-hot vector has length n where n is the number of classes. A one-hot vector has exactly one non-zero element. The non-zero element is 1. The coordinate of the non-zero element corresponds to a unique label. For example, given a classification problem with two labels FOO and BAR, one-hot vectors are generated as

$$\text{one-hot}(\text{label}) = \begin{cases} \langle 1, 0 \rangle & \text{if label} = \text{FOO} \\ \langle 0, 1 \rangle & \text{if label} = \text{BAR} \end{cases} \tag{4.18}$$

The common mean squared error (MSE) [149, 150, 151, 152, 153] function is simply given as

$$\text{mse}(\hat{\boldsymbol{Y}}, \boldsymbol{Y}) = \text{mean}((\hat{\boldsymbol{Y}} - \boldsymbol{Y})^2) \tag{4.19}$$

where $\hat{\boldsymbol{Y}}$ is a matrix of predictions provided by the activation of a model, $\boldsymbol{Y}$ is a matrix of ground truth targets, matrix square $\boldsymbol{M}^2$ squares each element of $\boldsymbol{M}$, and $\text{mean}(\boldsymbol{M}) = \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} \boldsymbol{M}_{ij}}{mn}$ for a matrix $\boldsymbol{M}$ with m rows and n columns. Note that matrix subscript $\boldsymbol{M}_{ij}$ denotes the element of $\boldsymbol{M}$ in row i and column j. With gradient based optimizers, the derivative of mse is essential for high performance:

$$\text{mse}'(\hat{\boldsymbol{Y}}, \boldsymbol{Y}) = \frac{d\text{mse}}{d\hat{\boldsymbol{Y}}} = \frac{2}{mn}(\hat{\boldsymbol{Y}} - \boldsymbol{Y}) \tag{4.20}$$

MSE is arguably the most common error function in supervised learning. It provides an intuitive and general definition of error, defining error as the squared difference between actual and expected values. Additionally, the derivative of MSE is simple and smooth.

An alternative to MSE with similar use cases is mean absolute error (MAE) [154, 155, 152, 156, 157]

$$\mathrm{mae}(\hat{\boldsymbol{Y}}, \boldsymbol{Y}) = \mathrm{mean}(\mathrm{abs}(\hat{\boldsymbol{Y}} - \boldsymbol{Y})) \tag{4.21}$$

where abs is element-wise absolute value. MAE has the derivative

$$\mathrm{mae}'(\hat{\boldsymbol{Y}}, \boldsymbol{Y}) = \frac{d\mathrm{mae}}{d\hat{\boldsymbol{Y}}} = \frac{\mathrm{sign}(\hat{\boldsymbol{Y}} - \boldsymbol{Y})}{mn} \tag{4.22}$$

where

$$\mathrm{sign}(x) = \begin{cases} -1 & \text{if } x < 0 \\ 0 & \text{if } x = 0 \\ 1 & \text{if } x > 0 \end{cases} \tag{4.23}$$

for scalar argument x and is applied to each element of a given matrix. MAE provides a straightforward and simple definition of error, defining error as the difference between actual and expected values. However, the derivative of MAE is non-smooth, leading to jagged problem spaces that are difficult to optimize. This is a significant detriment for gradient based optimizers, but is less important for non-smooth derivative free optimizers.

4.5 Support Vector Machine

The support vector machine (SVM) [158, 159, 160, 161, 162] method is a model originally designed for classification. SVM is designed under the theory that maximizing the margin between separated samples improves generalization accuracy in classification problems, as depicted in Figure 4.2.

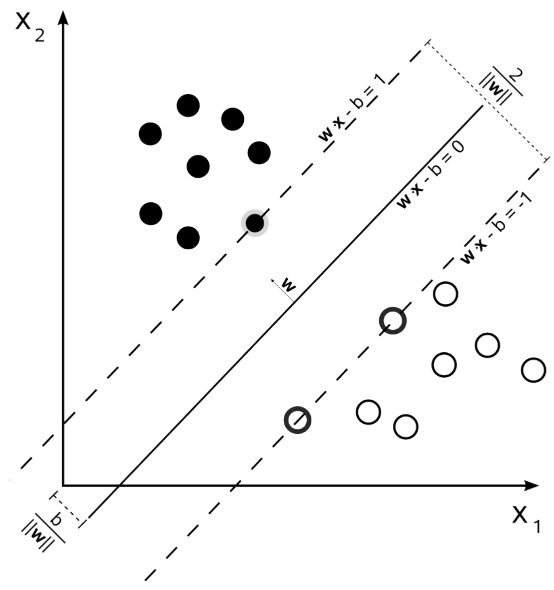

Figure 4.2: SVM maximizing the margin between its decision boundary and samples used as support vectors [163].

Using a regression model, such as those seen in Section 4.1, SVM is trained to minimize an error that is robust to outliers and creates an effective decision boundary. This use of a unique error function, such as epsilon insensitive loss [164, 165] for regression problems or hinge loss [166, 167, 168] for classification, differentiates SVM from regression models. Note that, while SVM traditionally uses a regression model, any model compatible with error functions can be an SVM model.

One effective error function for SVM is hinge loss (HL)

$$\mathrm{hl}(\hat{y}, y) = \max(0, 1 - \hat{y}y) \tag{4.24}$$

where the target value y is expected to be one of -1 or 1, and the predicted value $\hat{y}$ should likewise scale to the range $[-1, 1]$. The hinge loss derivative is

$$\text{hl}'(\hat{y}, y) = \frac{d\text{hl}}{\hat{y}} = \begin{cases} -y & \text{if } \hat{y}y < 1 \\ 0 & otherwise \end{cases} \tag{4.25}$$

Note that unlike our presentation of MSE and MAE, hinge loss is defined for scalar arguments. Traditional hinge loss is only defined for binary classifiers. However, we can extend it to multi-class problems and multiple samples by taking the hinge loss of corresponding elements in $\hat{\boldsymbol{Y}}$ and $\boldsymbol{Y}$, returning the mean

$$\text{HL}(\hat{\boldsymbol{Y}}, \boldsymbol{Y}) = \text{mean}(\max(0, 1 - \hat{\boldsymbol{Y}} \odot \boldsymbol{Y})) \tag{4.26}$$

where $\odot$ is the Hadamard, a.k.a., element-wise, product, and max is applied to each element of a matrix. The derivative of this multi-class HL is

$$\frac{d\text{HL}}{\hat{\boldsymbol{Y}}_{ij}} = \begin{cases} \frac{-\boldsymbol{Y}_{ij}}{mn} & \text{if } \hat{\boldsymbol{Y}}_{ij}\boldsymbol{Y}_{ij} < 1 \\ 0 & otherwise \end{cases} \tag{4.27}$$

where subscript ij is the element in the i^{th} row and j^{th} column, m is the number of rows in $\hat{\boldsymbol{Y}}$, and n is the number of columns in $\hat{\boldsymbol{Y}}$. Note the piecewise derivative where the partial derivative of each element depends only on that element and its corresponding element in $\boldsymbol{Y}$.

4.6 Non-Traditional Optimization and Decision Trees

Models such as regression and multilayer perceptrons effectively fit into the role of traditional mathematical optimization because their models are differentiable, allowing fast parameter training with gradient optimizers. However, some supervised learning models are non-differentiable and are not traditionally trained with gradient descent based methods. The function for a decision tree [19, 20, 21] model consists of if-else statements and inequalities, making differentiation infeasible. While derivative-free optimizers are capable of optimizing the parameters of such a model, they are inefficient without specialized information. To circumvent this issue, specialized algorithms, such as ID3 [22, 23], are developed to train decision trees, and similar algorithms exist for other non-differentiable models. Although these specialized algorithms are not commonly referred to as derivative-free optimizers, they fit the definition, and under the umbrella of supervised learning from the perspective of mathematical optimization, they are treated as such.

Chapter 5

Supervised Learning Comparison and Results

Through empirical tests we examine the performance of various optimizers, models, and error functions. Robust tests with many datasets and models reveal the typical performance of different optimizers and aid in optimizer selection when solving supervised learning problems. A comparison of models with tuned hyperparameters on a variety of datasets show their strengths and weaknesses, aiding in model selection. Models, error functions, and gradient optimizers are provided by the Learning python library [169]. Derivative-free optimizers are provided by the Optimal python library [170].

5.1 Datasets

The AND dataset is a simple boolean dataset consisting of all length two combinations of $\langle 0, 1\rangle$ labeled by $\vec{x}_1$ AND $\vec{x}_2$, where $\vec{x}$ is the attribute vector of a sample. The AND dataset is displayed in Figure 5.1.

The XOR dataset is a simple boolean dataset consisting of all length two combinations of $\langle 0, 1\rangle$ labeled by $\vec{x}_1$ XOR $\vec{x}_2$, where $\vec{x}$ is the attribute vector of a sample. The XOR dataset is displayed in Figure 5.1.

The popular iris dataset [171] contains real value attributes describing various characteristics of a flower: sepal width, septal length, petal width, and petal length. The goal is to classify a flower into three types of iris. The iris dataset provides a light challenge with low dimensionality and low noise.

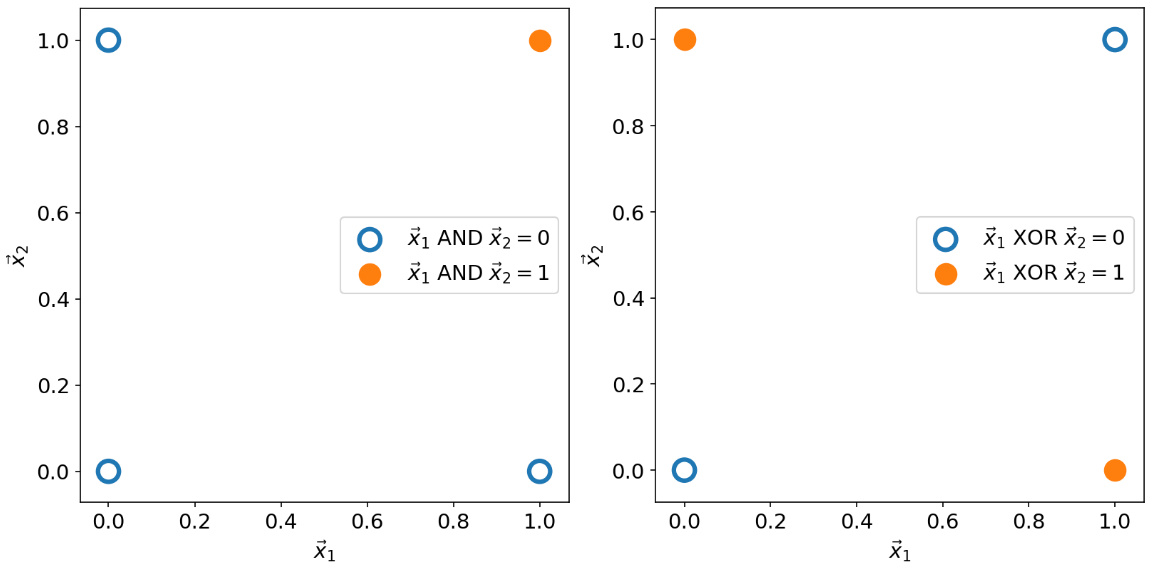

Figure 5.1: Each sample $\vec{x}$ in AND dataset (left) and XOR dataset (right)

The Wisconsin breast cancer diagnostic dataset (Cancer) [171] contains real valued attributes of cell nuclei measured from images of fine needle aspirate from breast mass. The goal is to classify each mass as malignant or benign. The cancer dataset provides a moderate classification challenge with medium dimensionality.

The Haberman's survival dataset (HM) [171] is a medical dataset predicting survival of patients after breast cancer surgery. It contains integer attributes for patient age, year of operation, and detected positive axillary nodes. The goal is to classify whether the patient dies within 5 years of surgery or survives.

The yeast dataset [171] contains real valued attributes describing a yeast cell. The goal is to classify the localization site of protein for each cell, as given by a discrete set of sites.

California housing (CH) [172, 173, 174, 175] is a regression dataset predicting house value based on neighborhood and house statistics. Attributes are obtained using all block groups in California from the 1990 census.

The US postal service hand-written digit dataset (USPS) [176] contains 16×16 grey-scale images of hand written digits, gathered at the Center of Excellence in Document Analysis and Recognition (CEDAR) at SUNY Buffalo, as part of a project sponsored by the US postal service. Images are scanned from post office mail and contain a multitude of writers and styles. USPS is a medium dimensional image datasets with a low number of classes.

The CMU Pose, Illumination, and Expression dataset (PIE) [177] is an image classification dataset containing facial images of 68 people in 13 different poses, 43 different illumination conditions, and with 4 different expressions. The PIE dataset benchmarked here is a subset of the original containing 67 people in the frontal view pose, under 21 illumination conditions with background light off, and a neutral expression. Images are made grey scale and scaled to 30×30 pixels. This dataset aims to predict the person. PIE is our most challenging benchmark dataset, given its high dimensional problem space and many classes.

Table 5.1 presents the problem type of each dataset, number of samples in the dataset, number of samples in the training set, attributes in each sample, and number of classes or regression values. For classification datasets, the training set is given an even distribution of classes.

5.2 Optimizer Comparison

Improving supervised learning performance allows for larger models and the solution of larger and more complex problems. Details of a model can affect training time in addition to generalization accuracy. However, there is often a tradeoff between model complexity and accuracy. On a complex problem, a large model may be necessary. Reducing the complexity of a model to improve training time is not ideal. Thankfully,

Table 5.1: Benchmark Datasets

Dataset	Type	Samples	Training Samples	Attributes	Classes / Outputs
AND	Boolean Classification	4	4	2	2
XOR	Boolean Classification	4	4	2	2
Iris	Classification	150	90	4	3
Cancer	Classification	569	280	30	2
HM	Classification	306	100	3	2
Yeast	Classification	1484	50	8	10
CH	Regression	20640	500	8	1
USPS	Image Classification	11000	500	256	10
PIE	Image Classification	1407	670	900	67

choice of optimizer and error function can drastically affect the convergence time of a supervised learning model.

Tables 5.2 and B.1 compare various optimizers detailed in Chapters 2 and 3 and error functions detailed in Sections 4.4 and 4.5. Mean squared error (MSE), mean absolute error (MAE), and hinge loss (HL) are examined. Each optimizer and error function (ξ) is used to train a number of models detailed in Chapter 4 on a number of datasets given in Section 5.1. Image classification datasets are excluded because optimizing these large datasets with an inefficient optimizer is extremely computationally expensive. Note that HL is not tested with regression datasets.

Models are trained until convergence, as defined by $\|\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\| < 4\text{e}{-4}$, where f is a function giving model predictions, $\boldsymbol{X}$ is an argument matrix, and $\boldsymbol{Y}$ is a target matrix. If training exceeds 10,000 iterations, or 200 iterations without improvement, training is stopped pre-maturely.

Full results are given in Table B.1. Objective function evaluations ξ_c plus the evaluations of its derivative ξ'_c, error $\xi(f(\boldsymbol{X}), \boldsymbol{Y})$ after training on the training set, and gradient norm ($\|\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\|$) after training on the training set, to indicate degree of convergence, are given.

Linear regression (LinReg), logistic regression (LogReg), multilayer perceptron (MLP), and radial basis function network (RBF) models are trained. Note that LogReg is not evaluated on regression datasets. The number of MLP hidden neurons are given in parenthesis and SM is given in parenthesis if softmax is used as the output layer. Softplus is used for hidden layers. Note that MLP with SM is not evaluated on regression datasets. The number of RBF clusters is given in parenthesis. Cluster centers are obtained with k-means clustering [178, 179, 180, 181, 182, 183, 184, 185] and normalized (4.15) Gaussian similarity (4.14) is used for similarity.

Steepest descent (SD), Broyden-Fletcher-Goldfarb-Shanno (BFGS), and limited-memory BFGS (L-BFGS) are utilized. Line search method, such as backtracking line search (B) and Wolfe line search (W), are given in parenthesis. Initial step size method, such as increment previous step (I), first-order change (F), and interpolating quadratic (Q), is given in parenthesis.

Derivative-free genetic algorithm (GA) with a population size of 100 binary vectors, tournament selection, one-point crossover, and bit flip mutation; population-based incremental learning (PBIL) with 100 samples per iteration; and gravitational search algorithm (GSA) with 100 bodies, lower bound of -50 and upper bound of 50 for each parameter are benchmarked. GA and PBIL decode every 24 sequential bits into a real number in the range $[-50, 50]$.

Optimizers use hyperparameters recommended in their respective sections unless otherwise stated. In addition to these state-of-the-art optimizers, a random optimizer that generates 100 random vectors, with each element in the range $[-50, 50]$, is included to provide baseline performance for the comparison.

For easier interpretation, Table 5.2 provides results aggregated over all models and datasets. Every error function and optimizer in Table B.1 is presented and mean iterations, MSE, and gradient norm are presented.

Table 5.2: Aggregate Optimizer Comparison

ξ	Optimizer	Mean $\xi_c + \xi_c'$	Mean $\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	Mean $\|\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\|$
MSE	SD(B(I))	13184.18	0.079	0.001
	SD(W(I))	15080.58	0.079	0.001
	SD(W(F))	11593.07	0.079	0.001
	SD(W(Q))	12537.01	0.080	0.001
	BFGS(B(I))	1240.29	0.082	0.001
	BFGS(W(I))	1289.84	0.078	0.000
	BFGS(W(F))	1495.53	0.078	0.000
	BFGS(W(Q))	994.30	0.078	0.000
	L-BFGS(B(I))	3839.21	0.096	0.000
	L-BFGS(W(I))	3007.10	0.076	0.000
	L-BFGS(W(F))	2900.93	0.078	0.000
	L-BFGS(W(Q))	3365.53	0.076	0.001
	GA	36363.71	473.935	143.961
	PBIL	77810.03	2.404	0.157
	GSA	68744.47	0.147	0.100
	Random	31292.41	8401.204	980.512
MAE	SD(B(I))	11500.63	0.209	0.053
	SD(W(I))	31492.08	0.201	0.050
	SD(W(F))	30335.89	0.196	0.041
	SD(W(Q))	33298.52	0.197	0.049
	BFGS(B(I))	34791.75	0.203	0.227
	BFGS(W(I))	129917.21	0.178	0.147
	BFGS(W(F))	112808.33	0.196	0.137
	BFGS(W(Q))	126330.66	0.189	0.137
	L-BFGS(B(I))	31530.27	0.256	0.131
	L-BFGS(W(I))	28770.03	0.190	0.064
	L-BFGS(W(F))	28912.38	0.182	0.083
	L-BFGS(W(Q))	28436.79	0.193	0.059
	GA	34002.07	2.270	0.338
	PBIL	58888.26	0.579	0.119
	GSA	78823.92	0.214	0.159
	Random	33659.53	15.343	1.756

(cont. on next page)

Table 5.2 (cont.)

ξ	Optimizer	Mean $\xi_c + \xi_c'$	Mean $\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	Mean $\|\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\|$
HL	SD(B(I))	12927.47	0.315	0.046
	SD(W(I))	29677.61	0.305	0.067
	SD(W(F))	27722.52	0.294	0.059
	SD(W(Q))	28955.70	0.296	0.063
	BFGS(B(I))	29820.09	0.328	0.049
	BFGS(W(I))	137597.61	0.272	0.044
	BFGS(W(F))	116342.39	0.294	0.044
	BFGS(W(Q))	141906.70	0.297	0.042
	L-BFGS(B(I))	15098.36	0.356	0.023
	L-BFGS(W(I))	20324.36	0.290	0.031
	L-BFGS(W(F))	19590.58	0.289	0.056
	L-BFGS(W(Q))	19947.45	0.287	0.069
	GA	28847.23	0.794	0.105
	PBIL	58708.85	0.328	0.032
	GSA	95517.15	0.239	0.105
	Random	31880.79	3.666	0.377

We expect BFGS and its limited-memory variant, with their additional second derivative information, to outperform steepest descent. This assumption holds true with the smooth and continuous ξ = MSE. However, when paired with the non-smooth MAE, the second derivative information of BFGS and L-BFGS prove ineffective, allowing steepest descent to outperform both in function calls and gradient norm convergence. A similar phenomena is observed with HL due to its piecewise derivative, but the effect is less pronounced and L-BFGS provides more consistent convergence with only marginally more function calls.

With MSE, Wolfe line search provides marginally faster convergence, especially when paired with quasi-Newton optimizers. This is not surprising because these quasi-Newton optimizers are only proved to converge when its step size satisfies the Wolfe conditions [33]. Satisfying the Wolfe conditions is less important for steepest descent, but the more flexible Wolfe line search still outperforms backtracking line search with

MSE. With MAE and HL, Wolfe line search struggles to find effective step sizes for steepest descent, allowing backtracking line search to significantly outperform it.

No one initial step size method is optimal on all combinations. First-order change (F) provides the fastest or near fastest convergence for Wolfe-line search, with the exception of BFGS paired with MSE. On this outlier, quadratic interpolation (Q) significantly outperforms F. The increment previous step (I) method proves less effective with Wolfe line search.

Without derivative information, the derivative-free GA, PBIL, and GSA optimizers are unable to consistently converge to a local optima, as evidenced by high gradient norm. With MSE, all gradient optimizers consistently converge and achieve relatively low error while derivative-free optimizers struggle to achieve even moderate error. However, with the non-smooth MAE and HL, the more effective PBIL and GSA derivative-free optimizers can achieve comparable error. GA consistency performs several times worse than either PBIL or GSA and has especially high error with MSE. The inability of PBIL and GSA to converge is offset by their ability to more thoroughly explore the problem space. Regardless, derivative-free optimizers require considerably more iterations to achieve comparable or worse error than gradient optimizers. When objective function derivatives are available, gradient based optimizers are preferred.

Overall, the smooth MSE allows for significantly faster and more consistent convergence than the non-smooth MAE and piece-wise HL. MAE and MSE have similar use cases, which leads to a lack of popularity for MAE due to it's significantly slower and more difficult convergence. HL however can improve generalization performance with its maximized margin. Whether or not to use HL over MSE depends on how much it improves accuracy, if at all, and whether that increased accuracy is worth slower convergence.

5.3 Model Comparison

While optimizers are the key to effectively training a model, they have little effect on ultimate accuracy beyond whether or not the model is fully trained. Choice of model, with its associated prior, and error function to an arguably lesser extent, ultimately determines accuracy and degree of success in a supervised learning problem. If a model and the equation it represents closely matches the underlying mechanics, equation, or simulation that generated a dataset, the model will perform well at learning and generalizing the dataset.

5.3.1 Selecting Hyperparameters

Hyperparameters for all models are discovered with derivative-free optimization that aims to maximize accuracy (or minimize error on regression problems) on a validation set, eliminating researcher bias and providing fair and robust hyperparameter selection. The validation set is a subset of the training set with $^2/_3$ samples for training the validation model and $^1/_3$ for validation.

The objective function f for classification problems is $f(m, D) = \text{acc}(m_{tr}, D_{te}) + 0.05 \times \text{acc}(m_{tr}, D_{tr})$, where m_{tr} is model m after training on a training set, acc is the accuracy of a model on a dataset, D_{te} is the testing set of dataset D, D_{tr} is the training set of dataset D, and the optimizer maximizes this quantity. A small amount of value is added for performance on the training set to discourage over-fitting on the validation set. The objective function for regression problems is $f(m, D) = \text{mse}(m_{tr}, D_{te}) + 0.05 \times \text{mse}(m_{tr}, D_{tr})$, where mse is mean squared error and the optimizer minimizes this quantity.

Our derivative-free optimizer for hyperparameter optimization is population-based incremental learning (PBIL), as we see its effectiveness in Section 5.2. This PBIL runs for 10 iterations with 10 samples every iteration. A low number of iterations is necessary because the objective function requires fully training and evaluating a model, which is a very computationally expensive operation. The recommended adjustment rate $\alpha \approx 0.1$ is increased to $\alpha = 0.2$ to allow rapid exploitation and convergence with few iterations. PBIL mutation is disabled because 10 iterations is too brief to adequately explore the problem space with mutation.

Linear (LinReg) and logistic (LogReg) regression models can select mean squared error (MSE) or hinge loss (HL) error function ξ, allowing automatic selection of regression or support vector machine (SVM). Multilayer perceptron (MLP) models can select a number of hidden neurons n_h in the range $[1, 128]$ and MSE or HL error function ξ. Radial basis function network (RBF) models can select a number of cluster centers n in the range $[1, 128]$, Gaussian variance v in the range $[4\sqrt{n_a} \times 0.01, 4\sqrt{n_a} \times 4]$ where n_a is the number of attributes in the dataset, and MSE or HL error function ξ. Gaussian variance is scaled by number of attributes to account for increased maximum euclidean distance in higher dimensional spaces. The quantity $4\sqrt{n_a}$ is the maximum distance between vectors $\langle -2, \ldots, -2 \rangle$ and $\langle 2, \ldots, 2 \rangle$ in n_a dimensional space. Cluster centers are determined with k-means clustering.

5.3.2 Model Comparison Results

Table 5.3 compares the generalization ability of several models detailed in Chapter 4. Each model is tested on the datasets presented in Section 5.1, except for the boolean toy datasets. Models with a number of parameters $p > 500$ are trained using L-BFGS with memory limit $m = 5$, Wolfe line search with $c_1 = 1\text{e}{-4}$ and $c_2 = 0.9$, and increment previous initial step with increment rate $r = 1.05$ and upper bound $u = 1$. Models with $p \leq 500$ are trained using BFGS reset every 100 iterations and the same Wolfe line search as L-BFGS. Accuracy on training and testing sets is presented for classification datasets, and mean squared error (MSE) is presented for regression datasets. Hyperparameters selected for each model, via the derivative-free search detailed in Section 5.3.1, are also included.

We see that, with a few exceptions, all models have comparable training and testing accuracy or error on all datasets. This is not surprising, given that all benchmarked models are state-of-the-art, commonly utilized in practice, and effectively trained with efficient optimizers. Although MLP is heralded for its complex function that allows approximation of a wide variety of equations, this complexity also causes overfitting that can decrease testing performance and makes optimization difficult, which allows the simpler regression models to outperform MLP on several datasets. We also see that both MSE and HL prove optimal with different datasets and models. HL appears particularly useful with linear regression. Note that the RBF model includes a linear regression function.

Ultimately, no one model is optimal on all problems. Each benchmarked model achieves or ties for the greatest testing performance on at least one dataset. This is an instance of the no-free-lunch theorem, which, in brief, states that no one algorithm is optimal on all problems. When solving a supervised learning problem, several models should be benchmarked without researcher bias. Empirical evidence and testing accuracy should guide the researcher to select a model. Researchers should not feel compelled to focus on a particular favorite model. Just as the underlying models that generate data are incredibly varied, the models we test and utilize must be equally varied. The optimization perspective allows easy experimentation and implementation of models. When optimization is handled by an existing optimizer, an equation with adjustable parameters or an error function is all that is necessary to implement a new model.

Table 5.3: Model Comparison

Dataset	Model	Training	Testing	Hyperparameters
Iris	LinReg	96.67%	96.67%	ξ: HL
	LogReg	98.89%	96.67%	ξ: MSE
	MLP	98.89%	98.33%	ξ: MSE, n_h: 32
	RBF	98.89%	96.67%	ξ: HL, n: 127, v: 0.08
Cancer	LinReg	100.00%	95.16%	ξ: HL
	LogReg	99.64%	96.89%	ξ: MSE
	MLP	99.64%	96.89%	ξ: MSE, n_h: 24
	RBF	98.57%	95.85%	ξ: MSE, n: 119, v: 0.643
HM	LinReg	69.00%	69.42%	ξ: HL
	LogReg	70.00%	61.65%	ξ: MSE
	MLP	73.00%	58.74%	ξ: HL, n_h: 3
	RBF	71.00%	67.48%	ξ: HL, n: 108, v: 12.552
Yeast	LinReg	76.00%	47.77%	ξ: MSE
	LogReg	84.00%	44.07%	ξ: MSE
	MLP	86.00%	41.49%	ξ: MSE, n_h: 79
	RBF	74.00%	46.03%	ξ: HL, n: 18, v: 18.106
CH	LinReg	0.071	0.085	ξ: MSE
	MLP	0.055	0.074	ξ: MSE, n_h: 2
	RBF	0.061	0.085	ξ: MSE, n: 80, v: 3.367
USPS	LinReg	100.00%	78.05%	ξ: HL
	LogReg	98.80%	75.47%	ξ: MSE
	MLP	100.00%	88.02%	ξ: MSE, n_h: 84
	RBF	96.60%	88.39%	ξ: HL, n: 120, v: 157.635
PIE	LinReg	100.00%	94.44%	ξ: HL
	LogReg	100.00%	98.64%	ξ: MSE
	MLP	100.00%	98.91%	ξ: MSE, n_h: 88
	RBF	99.85%	91.04%	ξ: HL, n: 117, v: 451.924

Chapter 6

Conclusion

The field of mathematical optimization is both extensive and under-represented in the machine learning community. Efficient gradient optimization methods like steepest descent and Broyden-Fletcher-Goldfarb-Shanno (BFGS) make big data problems with complex non-linear models feasible. However, despite these contributions, supervised learning models such as neural networks or support vector machines are frequently discussed from a biological or statistical perspective without mention of the importance of the optimization methods that make efficient training possible.

The importance of effective optimization cannot be understated. Our exploration of state-of-the-art optimizers in Chapters 2 and 3 provides an examination of this rapidly evolving field with implementation details for several important methods. Chapter 4 provides important details to implement a number of popular state-of-the-art supervised learning models with powerful and efficient optimizers. From the optimization perspective, several otherwise obscured similarities are revealed between these models, allowing for deeper understanding and modular implementation with many reusable components. The comparative analysis in Chapter 5 takes advantage of the modularity of the optimization perspective to easily generate an extensive and comprehensive analysis with many datasets, error functions, models, and optimizers.

Our comparison of models in Section 5.3 shows the importance of testing many models when attempting to maximize performance on a given dataset. When the function and parameters of a model are decoupled from training procedure, as the optimization perspective provides, development and implementation of a new model is trivial. One can simply adjust the function of a model. For example, if a linear regression model performs well on a given dataset, but we suspect that the first and second attribute are related with regard to the first class, we can create a new model

$$f(\boldsymbol{X}) = w(\boldsymbol{X}_1^T \odot \boldsymbol{X}_2^T) +_{i1} (\boldsymbol{X}\boldsymbol{W} + \vec{b}) \tag{6.1}$$

to multiply the first attribute with the second attribute of each pattern, multiply the result by a weight parameter w, and add the resulting vector to the first column of a linear regression model $\boldsymbol{X}\boldsymbol{W} + \vec{b}$, where $\boldsymbol{X}$ is a matrix of pattern attributes (a.k.a. model arguments), $\boldsymbol{W}$ is a weight matrix, and $\vec{b}$ is a bias vector. Note that $\odot$ is the Hadamard, a.k.a., element-wise, product and $+_{i1}$ denotes addition of a vector to the first column of a matrix. In this brief paragraph, we developed a new and specialized model to potentially improve performance on a hypothetical problem. Under the optimization perspective, training this model only requires applying an optimizer to minimize error while adjusting weight and bias parameters.

Likewise, a new error function can improve accuracy of current and future models. The hinge loss function described in Section 4.5 is one such example. However, under traditional supervised learning, hinge loss is part of a specific model known as a support vector machine (SVM). Under the optimization perspective, we combine hinge loss with regression, multilayer perceptron, radial basis function network (RBF), and more. In Section 5.3 we even see that hinge loss improves the performance of RBF. This revelation is intuitive when supervised learning is viewed as an application of mathematical optimization.

Appendix A

Notation

- Vectors are presented with an arrow above, such as $\vec{x}$.
- All vectors are row vectors unless otherwise specified.
- Specific vectors are given in arrow brackets, such as $\langle 0, 1, 2 \rangle$.
- Matrices are presented as bold uppercase letters, such as $\boldsymbol{X}$.
- Norms, such as $\|\vec{x}\|$, are euclidean norms, $\|\vec{x}\|_2$, unless otherwise specified.
- Function derivatives are presented with prime notation, such as f' for first derivative, a.k.a. Jacobian, and f'' for second derivative, a.k.a. Hessian.
- A scalar added to a vector or matrix, such as $a + \vec{x}$ or $a + \boldsymbol{X}$, adds the scalar to every element of the vector or matrix. A vector added to a matrix, such as $\vec{x} + \boldsymbol{X}$, adds the vector to every row of the matrix.

Appendix B

Raw Results

Table B.1: Optimzier Comparison

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\lVert\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\rVert$
AND	MSE	LinReg	SD(B(I))	30	0.063	0.000
			SD(W(I))	22	0.063	0.000
			SD(W(F))	10	0.062	0.000
			SD(W(Q))	10	0.062	0.000
			BFGS(B(I))	30	0.063	0.000
			BFGS(W(I))	10	0.062	0.000
			BFGS(W(F))	10	0.062	0.000
			BFGS(W(Q))	10	0.062	0.000
			L-BFGS(B(I))	30	0.063	0.000
			L-BFGS(W(I))	10	0.062	0.000
			L-BFGS(W(F))	10	0.062	0.000
			L-BFGS(W(Q))	10	0.062	0.000
			GA	69841	15.030	5.471
			PBIL	17140	0.375	0.791
			GSA	52502	0.074	0.150
			Random	24202	86.077	13.116
		LogReg	SD(B(I))	41	0.000	0.000
			SD(W(I))	212	0.000	0.000
			SD(W(F))	94	0.000	0.000
			SD(W(Q))	42	0.000	0.000
			BFGS(B(I))	37	0.000	0.000
			BFGS(W(I))	48	0.000	0.000
			BFGS(W(F))	48	0.000	0.000
			BFGS(W(Q))	28	0.000	0.000
			L-BFGS(B(I))	38	0.000	0.000
			L-BFGS(W(I))	44	0.000	0.000
			L-BFGS(W(F))	48	0.000	0.000
			L-BFGS(W(Q))	28	0.000	0.000
			GA	26729	0.000	0.000
			PBIL	16105	0.000	0.000
			GSA	149602	0.000	0.000
			Random	39602	0.000	0.000
		MLP(2)	SD(B(I))	9502	0.000	0.000
			SD(W(I))	12366	0.000	0.001
			SD(W(F))	8840	0.000	0.000

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\Vert\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\Vert$
			SD(W(Q))	12716	0.000	0.001
			BFGS(B(I))	343	0.000	0.001
			BFGS(W(I))	360	0.000	0.001
			BFGS(W(F))	440	0.000	0.002
			BFGS(W(Q))	364	0.000	0.000
			L-BFGS(B(I))	756	0.000	0.000
			L-BFGS(W(I))	424	0.000	0.000
			L-BFGS(W(F))	516	0.000	0.000
			L-BFGS(W(Q))	444	0.000	0.001
			GA	65703	0.286	0.131
			PBIL	26260	0.125	0.002
			GSA	51002	0.160	0.062
			Random	23702	0.383	0.114
		MLP(2,SM)	SD(B(I))	64	0.063	0.000
			SD(W(I))	222	0.000	0.001
			SD(W(F))	72	0.063	0.000
			SD(W(Q))	52	0.063	0.000
			BFGS(B(I))	38	0.063	0.000
			BFGS(W(I))	46	0.063	0.000
			BFGS(W(F))	118	0.063	0.000
			BFGS(W(Q))	58	0.063	0.000
			L-BFGS(B(I))	37	0.063	0.000
			L-BFGS(W(I))	46	0.063	0.000
			L-BFGS(W(F))	60	0.063	0.000
			L-BFGS(W(Q))	44	0.063	0.000
			GA	19310	0.000	0.000
			PBIL	20155	0.000	0.000
			GSA	20202	0.000	0.000
			Random	20302	0.000	0.000
		MLP(4)	SD(B(I))	2364	0.000	0.000
			SD(W(I))	3190	0.000	0.000
			SD(W(F))	2602	0.000	0.000
			SD(W(Q))	2822	0.000	0.001
			BFGS(B(I))	150	0.000	0.000
			BFGS(W(I))	166	0.000	0.000
			BFGS(W(F))	214	0.000	0.000
			BFGS(W(Q))	212	0.000	0.000
			L-BFGS(B(I))	216	0.000	0.000
			L-BFGS(W(I))	314	0.000	0.000
			L-BFGS(W(F))	388	0.000	0.000
			L-BFGS(W(Q))	290	0.000	0.000
			GA	21011	0.382	0.077
			PBIL	71031	0.125	0.001
			GSA	70602	0.045	0.138
			Random	26802	743.376	323.548
		MLP(4,SM)	SD(B(I))	75	0.000	0.000
			SD(W(I))	176	0.000	0.000
			SD(W(F))	92	0.000	0.000

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\|\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\|$
			SD(W(Q))	82	0.000	0.000
			BFGS(B(I))	60	0.063	0.000
			BFGS(W(I))	244	0.000	0.000
			BFGS(W(F))	74	0.000	0.000
			BFGS(W(Q))	102	0.000	0.000
			L-BFGS(B(I))	67	0.000	0.000
			L-BFGS(W(I))	58	0.000	0.000
			L-BFGS(W(F))	48	0.000	0.000
			L-BFGS(W(Q))	44	0.000	0.000
			GA	19077	0.000	0.000
			PBIL	20202	0.000	0.000
			GSA	20202	0.000	0.000
			Random	20202	0.000	0.000
		MLP(8)	SD(B(I))	1351	0.000	0.000
			SD(W(I))	1758	0.000	0.000
			SD(W(F))	1168	0.000	0.001
			SD(W(Q))	1406	0.000	0.000
			BFGS(B(I))	113	0.000	0.000
			BFGS(W(I))	106	0.000	0.000
			BFGS(W(F))	148	0.000	0.000
			BFGS(W(Q))	142	0.000	0.000
			L-BFGS(B(I))	137	0.000	0.000
			L-BFGS(W(I))	258	0.000	0.000
			L-BFGS(W(F))	168	0.000	0.001
			L-BFGS(W(Q))	172	0.000	0.001
			GA	22150	4195.584	3377.078
			PBIL	98089	0.000	0.000
			GSA	96302	0.016	0.075
			Random	29802	109195.377	17636.439
		MLP(8,SM)	SD(B(I))	89	0.000	0.000
			SD(W(I))	218	0.000	0.000
			SD(W(F))	118	0.000	0.000
			SD(W(Q))	70	0.000	0.000
			BFGS(B(I))	61	0.000	0.000
			BFGS(W(I))	84	0.000	0.000
			BFGS(W(F))	60	0.000	0.000
			BFGS(W(Q))	68	0.000	0.000
			L-BFGS(B(I))	47	0.000	0.000
			L-BFGS(W(I))	50	0.000	0.000
			L-BFGS(W(F))	52	0.000	0.000
			L-BFGS(W(Q))	68	0.000	0.000
			GA	19170	0.000	0.000
			PBIL	20202	0.000	0.000
			GSA	20402	0.000	0.000
			Random	20202	0.000	0.000
		RBF(2)	SD(B(I))	102	0.184	0.000
			SD(W(I))	104	0.184	0.000
			SD(W(F))	134	0.184	0.000

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\lVert\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\rVert$
			SD(W(Q))	100	0.184	0.000
			BFGS(B(I))	35	0.184	0.000
			BFGS(W(I))	26	0.184	0.000
			BFGS(W(F))	74	0.184	0.000
			BFGS(W(Q))	84	0.184	0.000
			L-BFGS(B(I))	53	0.184	0.000
			L-BFGS(W(I))	26	0.184	0.000
			L-BFGS(W(F))	100	0.184	0.000
			L-BFGS(W(Q))	78	0.184	0.000
			GA	27622	0.691	0.972
			PBIL	18776	0.184	0.005
			GSA	30302	0.187	0.068
			Random	34202	4.376	2.077
		RBF(4)	SD(B(I))	79	0.000	0.000
			SD(W(I))	88	0.000	0.000
			SD(W(F))	102	0.000	0.000
			SD(W(Q))	102	0.000	0.000
			BFGS(B(I))	48	0.000	0.000
			BFGS(W(I))	38	0.000	0.000
			BFGS(W(F))	86	0.000	0.000
			BFGS(W(Q))	90	0.000	0.000
			L-BFGS(B(I))	44	0.000	0.000
			L-BFGS(W(I))	42	0.000	0.000
			L-BFGS(W(F))	82	0.000	0.000
			L-BFGS(W(Q))	86	0.000	0.000
			GA	25019	16.734	5.478
			PBIL	30591	0.206	0.369
			GSA	30502	0.009	0.112
			Random	23302	82.191	10.455
		RBF(8)	SD(B(I))	111	0.000	0.000
			SD(W(I))	122	0.000	0.000
			SD(W(F))	120	0.000	0.000
			SD(W(Q))	122	0.000	0.000
			BFGS(B(I))	44	0.000	0.000
			BFGS(W(I))	26	0.000	0.000
			BFGS(W(F))	82	0.000	0.000
			BFGS(W(Q))	68	0.000	0.000
			L-BFGS(B(I))	44	0.000	0.000
			L-BFGS(W(I))	26	0.000	0.000
			L-BFGS(W(F))	68	0.000	0.000
			L-BFGS(W(Q))	62	0.000	0.000
			GA	51692	17.069	3.916
			PBIL	43821	0.000	0.007
			GSA	26102	0.005	0.051
			Random	21502	19.981	3.446
	MAE	LinReg	SD(B(I))	25	0.250	0.000
			SD(W(I))	14	0.250	0.000
			SD(W(F))	14	0.250	0.000

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi_c'$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\lVert \xi'(f(\boldsymbol{X}), \boldsymbol{Y}) \rVert$
			SD(W(Q))	22	0.250	0.000
			BFGS(B(I))	49	0.250	0.000
			BFGS(W(I))	28	0.250	0.000
			BFGS(W(F))	38	0.250	0.000
			BFGS(W(Q))	36	0.250	0.000
			L-BFGS(B(I))	49	0.250	0.000
			L-BFGS(W(I))	22	0.250	0.000
			L-BFGS(W(F))	26	0.250	0.000
			L-BFGS(W(Q))	24	0.250	0.000
			GA	34717	3.959	0.707
			PBIL	18741	0.500	0.661
			GSA	28802	0.250	0.000
			Random	24202	7.224	0.707
		LogReg	SD(B(I))	32	0.000	0.000
			SD(W(I))	204	0.001	0.000
			SD(W(F))	52	0.000	0.000
			SD(W(Q))	40	0.000	0.000
			BFGS(B(I))	37	0.000	0.000
			BFGS(W(I))	46	0.000	0.000
			BFGS(W(F))	38	0.000	0.000
			BFGS(W(Q))	30	0.000	0.000
			L-BFGS(B(I))	42	0.000	0.000
			L-BFGS(W(I))	50	0.000	0.000
			L-BFGS(W(F))	42	0.000	0.000
			L-BFGS(W(Q))	42	0.000	0.000
			GA	22502	0.000	0.000
			PBIL	16359	0.000	0.000
			GSA	100102	0.000	0.000
			Random	39602	0.000	0.000
		MLP(2)	SD(B(I))	9532	0.450	0.252
			SD(W(I))	41294	0.250	0.022
			SD(W(F))	41264	0.250	0.019
			SD(W(Q))	41404	0.250	0.023
			BFGS(B(I))	9913	0.250	0.006
			BFGS(W(I))	452312	0.125	0.944
			BFGS(W(F))	41092	0.250	0.010
			BFGS(W(Q))	485166	0.125	1.380
			L-BFGS(B(I))	9241	0.250	0.280
			L-BFGS(W(I))	41376	0.250	0.273
			L-BFGS(W(F))	41462	0.250	0.268
			L-BFGS(W(Q))	41280	0.250	0.020
			GA	28288	0.379	0.217
			PBIL	21589	0.253	0.299
			GSA	56002	0.261	0.235
			Random	23702	0.417	0.282
		MLP(2,SM)	SD(B(I))	55	0.125	0.000
			SD(W(I))	174	0.125	0.000
			SD(W(F))	66	0.125	0.001

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi_c'$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\|\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\|$
			SD(W(Q))	60	0.125	0.000
			BFGS(B(I))	37	0.125	0.000
			BFGS(W(I))	138	0.125	0.000
			BFGS(W(F))	36	0.125	0.000
			BFGS(W(Q))	34	0.125	0.000
			L-BFGS(B(I))	30	0.625	0.000
			L-BFGS(W(I))	78	0.125	0.000
			L-BFGS(W(F))	36	0.125	0.000
			L-BFGS(W(Q))	32	0.125	0.000
			GA	20530	0.000	0.000
			PBIL	20087	0.000	0.000
			GSA	20202	0.000	0.000
			Random	21502	0.000	0.000
		MLP(4)	SD(B(I))	2175	0.251	0.092
			SD(W(I))	41542	0.251	0.013
			SD(W(F))	41276	0.250	0.011
			SD(W(Q))	41390	0.250	0.010
			BFGS(B(I))	28869	0.000	1.373
			BFGS(W(I))	59080	0.250	0.010
			BFGS(W(F))	112908	0.000	0.741
			BFGS(W(Q))	41250	0.250	0.010
			L-BFGS(B(I))	12820	0.250	0.008
			L-BFGS(W(I))	41298	0.250	0.010
			L-BFGS(W(F))	41314	0.251	0.371
			L-BFGS(W(Q))	41270	0.250	0.012
			GA	31007	0.408	0.155
			PBIL	34421	0.125	0.368
			GSA	27202	0.162	0.394
			Random	26802	17.041	4.391
		MLP(4,SM)	SD(B(I))	60	0.000	0.000
			SD(W(I))	104	0.000	0.000
			SD(W(F))	90	0.000	0.000
			SD(W(Q))	46	0.125	0.000
			BFGS(B(I))	59	0.125	0.000
			BFGS(W(I))	46	0.250	0.000
			BFGS(W(F))	114	0.248	0.010
			BFGS(W(Q))	84	0.250	0.003
			L-BFGS(B(I))	47	0.125	0.000
			L-BFGS(W(I))	160	0.125	0.000
			L-BFGS(W(F))	30	0.125	0.000
			L-BFGS(W(Q))	30	0.125	0.000
			GA	19113	0.000	0.000
			PBIL	20202	0.000	0.000
			GSA	20202	0.000	0.000
			Random	20202	0.000	0.000
		MLP(8)	SD(B(I))	1416	0.283	0.303
			SD(W(I))	42318	0.261	0.174
			SD(W(F))	41434	0.251	0.281

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\Vert\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\Vert$
			SD(W(Q))	41346	0.248	0.016
			BFGS(B(I))	35370	0.000	0.964
			BFGS(W(I))	197686	0.000	0.363
			BFGS(W(F))	176256	0.000	0.481
			BFGS(W(Q))	59346	0.246	0.021
			L-BFGS(B(I))	12381	0.011	1.874
			L-BFGS(W(I))	41324	0.248	0.016
			L-BFGS(W(F))	41488	0.248	0.016
			L-BFGS(W(Q))	41292	0.248	0.015
			GA	40711	0.821	0.956
			PBIL	169862	0.125	0.336
			GSA	39902	0.129	1.317
			Random	47202	234.711	36.937
		MLP(8,SM)	SD(B(I))	74	0.000	0.000
			SD(W(I))	326	0.000	0.000
			SD(W(F))	96	0.000	0.000
			SD(W(Q))	44	0.136	0.007
			BFGS(B(I))	51	0.250	0.000
			BFGS(W(I))	298	0.125	0.000
			BFGS(W(F))	22	0.125	0.000
			BFGS(W(Q))	66	0.250	0.001
			L-BFGS(B(I))	37	0.125	0.000
			L-BFGS(W(I))	124	0.000	0.000
			L-BFGS(W(F))	116	0.000	0.000
			L-BFGS(W(Q))	54	0.243	0.018
			GA	19147	0.000	0.000
			PBIL	20202	0.000	0.000
			GSA	20702	0.000	0.000
			Random	20202	0.000	0.000
		RBF(2)	SD(B(I))	1654	0.446	0.434
			SD(W(I))	42038	0.255	0.404
			SD(W(F))	40138	0.256	0.207
			SD(W(Q))	41878	0.252	0.340
			BFGS(B(I))	7155	0.250	0.215
			BFGS(W(I))	252686	0.250	0.345
			BFGS(W(F))	192600	0.250	0.639
			BFGS(W(Q))	60016	0.250	0.510
			L-BFGS(B(I))	5253	0.250	0.323
			L-BFGS(W(I))	41824	0.250	0.207
			L-BFGS(W(F))	41502	0.250	0.340
			L-BFGS(W(Q))	41382	0.250	0.434
			GA	35866	0.724	0.340
			PBIL	16549	0.263	0.207
			GSA	24802	0.265	0.533
			Random	22202	1.714	0.486
		RBF(4)	SD(B(I))	1868	0.121	0.423
			SD(W(I))	42018	0.022	0.338
			SD(W(F))	46528	0.038	0.241

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\\|\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\\|$
			SD(W(Q))	43100	0.103	0.386
			BFGS(B(I))	8498	0.000	0.424
			BFGS(W(I))	219812	0.000	0.350
			BFGS(W(F))	924	0.000	0.000
			BFGS(W(Q))	42352	0.000	0.350
			L-BFGS(B(I))	19883	0.000	0.243
			L-BFGS(W(I))	41998	0.000	0.554
			L-BFGS(W(F))	41718	0.000	0.243
			L-BFGS(W(Q))	41668	0.000	0.424
			GA	54274	2.804	0.605
			PBIL	26001	0.040	0.491
			GSA	47502	0.076	0.329
			Random	58702	5.782	0.609
		RBF(8)	SD(B(I))	26459	0.187	0.353
			SD(W(I))	42726	0.161	0.342
			SD(W(F))	41680	0.165	0.342
			SD(W(Q))	42092	0.174	0.331
			BFGS(B(I))	1443	0.000	0.000
			BFGS(W(I))	199944	0.000	0.342
			BFGS(W(F))	159320	0.000	0.395
			BFGS(W(Q))	81342	0.000	0.216
			L-BFGS(B(I))	20388	0.000	0.216
			L-BFGS(W(I))	42458	0.000	0.442
			L-BFGS(W(F))	42826	0.000	0.451
			L-BFGS(W(Q))	42986	0.000	0.364
			GA	21474	3.563	0.433
			PBIL	34988	0.004	0.433
			GSA	57902	0.043	0.353
			Random	21502	4.122	0.353
	HL	LinReg	SD(B(I))	32	0.000	0.000
			SD(W(I))	22	0.000	0.000
			SD(W(F))	26	0.000	0.000
			SD(W(Q))	20	0.000	0.000
			BFGS(B(I))	38	0.000	0.000
			BFGS(W(I))	18	0.000	0.000
			BFGS(W(F))	26	0.000	0.000
			BFGS(W(Q))	22	0.000	0.000
			L-BFGS(B(I))	20	0.000	0.000
			L-BFGS(W(I))	18	0.000	0.000
			L-BFGS(W(F))	26	0.000	0.000
			L-BFGS(W(Q))	22	0.000	0.000
			GA	19201	0.000	0.000
			PBIL	16943	0.000	0.000
			GSA	20602	0.000	0.000
			Random	20302	0.000	0.000
		LogReg	SD(B(I))	44	0.000	0.000
			SD(W(I))	220	0.001	0.000
			SD(W(F))	52	0.000	0.000

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\|\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\|$
			SD(W(Q))	60	0.000	0.000
			BFGS(B(I))	38	0.000	0.000
			BFGS(W(I))	52	0.000	0.000
			BFGS(W(F))	32	0.000	0.000
			BFGS(W(Q))	28	0.000	0.000
			L-BFGS(B(I))	42	0.000	0.000
			L-BFGS(W(I))	52	0.000	0.000
			L-BFGS(W(F))	32	0.000	0.000
			L-BFGS(W(Q))	42	0.000	0.000
			GA	21098	0.000	0.000
			PBIL	15864	0.000	0.000
			GSA	59402	0.000	0.000
			Random	39602	0.000	0.000
		MLP(2)	SD(B(I))	2428	0.474	0.344
			SD(W(I))	41770	0.415	0.255
			SD(W(F))	42682	0.476	0.319
			SD(W(Q))	42454	0.479	0.364
			BFGS(B(I))	78	0.250	0.000
			BFGS(W(I))	296	0.250	0.000
			BFGS(W(F))	322	0.250	0.000
			BFGS(W(Q))	98	0.250	0.000
			L-BFGS(B(I))	503	0.250	0.000
			L-BFGS(W(I))	120	0.250	0.000
			L-BFGS(W(F))	78	0.250	0.000
			L-BFGS(W(Q))	84	0.250	0.000
			GA	19700	0.000	0.000
			PBIL	20796	0.000	0.000
			GSA	20302	0.000	0.000
			Random	20502	0.000	0.000
		MLP(2,SM)	SD(B(I))	51	0.250	0.000
			SD(W(I))	190	0.250	0.000
			SD(W(F))	74	0.250	0.000
			SD(W(Q))	50	0.250	0.000
			BFGS(B(I))	57	1.000	0.000
			BFGS(W(I))	140	0.250	0.000
			BFGS(W(F))	78	0.250	0.000
			BFGS(W(Q))	26	0.250	0.000
			L-BFGS(B(I))	52	0.250	0.000
			L-BFGS(W(I))	104	0.250	0.000
			L-BFGS(W(F))	94	0.250	0.000
			L-BFGS(W(Q))	46	0.250	0.000
			GA	19287	0.000	0.000
			PBIL	20190	0.000	0.000
			GSA	20202	0.000	0.000
			Random	20202	0.000	0.000
		MLP(4)	SD(B(I))	1977	0.489	0.417
			SD(W(I))	41488	0.435	0.321
			SD(W(F))	41440	0.457	0.219

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\lVert\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\rVert$
			SD(W(Q))	41750	0.435	0.329
			BFGS(B(I))	62	0.000	0.000
			BFGS(W(I))	78	0.000	0.000
			BFGS(W(F))	76	0.000	0.000
			BFGS(W(Q))	58	0.000	0.000
			L-BFGS(B(I))	1035	0.000	0.000
			L-BFGS(W(I))	86	0.000	0.000
			L-BFGS(W(F))	318	0.000	0.000
			L-BFGS(W(Q))	84	0.000	0.000
			GA	19374	0.000	0.000
			PBIL	20202	0.000	0.000
			GSA	20902	0.000	0.000
			Random	20302	0.000	0.000
		MLP(4,SM)	SD(B(I))	60	0.000	0.000
			SD(W(I))	106	0.000	0.000
			SD(W(F))	126	0.000	0.000
			SD(W(Q))	92	0.000	0.000
			BFGS(B(I))	316	0.250	0.000
			BFGS(W(I))	222	0.250	0.000
			BFGS(W(F))	142	0.250	0.000
			BFGS(W(Q))	54	0.486	0.050
			L-BFGS(B(I))	37	0.250	0.000
			L-BFGS(W(I))	74	0.409	0.070
			L-BFGS(W(F))	30	0.250	0.000
			L-BFGS(W(Q))	142	0.250	0.000
			GA	19056	0.000	0.000
			PBIL	20202	0.000	0.000
			GSA	20202	0.000	0.000
			Random	20202	0.000	0.000
		MLP(8)	SD(B(I))	30037	0.518	0.271
			SD(W(I))	54604	0.500	0.539
			SD(W(F))	42544	0.505	0.533
			SD(W(Q))	42094	0.398	0.271
			BFGS(B(I))	40	0.000	0.000
			BFGS(W(I))	92	0.000	0.000
			BFGS(W(F))	102	0.000	0.000
			BFGS(W(Q))	74	0.000	0.000
			L-BFGS(B(I))	463	0.000	0.000
			L-BFGS(W(I))	168	0.000	0.000
			L-BFGS(W(F))	112	0.000	0.000
			L-BFGS(W(Q))	86	0.000	0.000
			GA	19487	0.000	0.000
			PBIL	20502	0.000	0.000
			GSA	21402	0.000	0.000
			Random	20302	0.000	0.000
		MLP(8,SM)	SD(B(I))	78	0.000	0.000
			SD(W(I))	120	0.000	0.000
			SD(W(F))	108	0.000	0.000

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi_c'$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\|\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\|$
			SD(W(Q))	80	0.000	0.000
			BFGS(B(I))	50	0.250	0.000
			BFGS(W(I))	104	0.250	0.000
			BFGS(W(F))	154	0.000	0.000
			BFGS(W(Q))	126	0.245	0.013
			L-BFGS(B(I))	37	0.250	0.000
			L-BFGS(W(I))	106	0.250	0.000
			L-BFGS(W(F))	170	0.000	0.000
			L-BFGS(W(Q))	136	0.000	0.000
			GA	19147	0.000	0.000
			PBIL	20202	0.000	0.000
			GSA	20202	0.000	0.000
			Random	20202	0.000	0.000
		RBF(2)	SD(B(I))	45902	0.511	0.220
			SD(W(I))	42224	0.510	0.132
			SD(W(F))	42502	0.508	0.328
			SD(W(Q))	41638	0.503	0.328
			BFGS(B(I))	5018	0.500	0.402
			BFGS(W(I))	276992	0.500	0.361
			BFGS(W(F))	59256	0.500	0.344
			BFGS(W(Q))	334132	0.500	0.361
			L-BFGS(B(I))	5714	0.500	0.697
			L-BFGS(W(I))	41280	0.503	0.328
			L-BFGS(W(F))	41292	0.501	0.481
			L-BFGS(W(Q))	41542	0.500	0.621
			GA	27525	1.014	0.373
			PBIL	21304	0.514	0.132
			GSA	45302	0.552	0.493
			Random	28702	1.612	0.445
		RBF(4)	SD(B(I))	31	0.000	0.000
			SD(W(I))	30	0.000	0.000
			SD(W(F))	30	0.000	0.000
			SD(W(Q))	18	0.000	0.000
			BFGS(B(I))	31	0.000	0.000
			BFGS(W(I))	24	0.000	0.000
			BFGS(W(F))	18	0.000	0.000
			BFGS(W(Q))	30	0.000	0.000
			L-BFGS(B(I))	37	0.000	0.000
			L-BFGS(W(I))	24	0.000	0.000
			L-BFGS(W(F))	18	0.000	0.000
			L-BFGS(W(Q))	30	0.000	0.000
			GA	19145	0.000	0.000
			PBIL	20600	0.000	0.000
			GSA	20602	0.000	0.000
			Random	20802	0.000	0.000
		RBF(8)	SD(B(I))	45901	0.315	0.395
			SD(W(I))	42162	0.034	0.306
			SD(W(F))	42270	0.061	0.306

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\Vert\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\Vert$
			SD(W(Q))	42524	0.080	0.306
			BFGS(B(I))	40	0.000	0.000
			BFGS(W(I))	48	0.000	0.000
			BFGS(W(F))	26	0.000	0.000
			BFGS(W(Q))	46	0.000	0.000
			L-BFGS(B(I))	45	0.000	0.000
			L-BFGS(W(I))	56	0.000	0.000
			L-BFGS(W(F))	36	0.000	0.000
			L-BFGS(W(Q))	58	0.000	0.000
			GA	19327	0.000	0.000
			PBIL	20202	0.000	0.000
			GSA	21502	0.000	0.000
			Random	21102	0.000	0.000
XOR	MSE	LinReg	SD(B(I))	30	0.250	0.000
			SD(W(I))	22	0.250	0.000
			SD(W(F))	10	0.250	0.000
			SD(W(Q))	10	0.250	0.000
			BFGS(B(I))	30	0.250	0.000
			BFGS(W(I))	10	0.250	0.000
			BFGS(W(F))	10	0.250	0.000
			BFGS(W(Q))	10	0.250	0.000
			L-BFGS(B(I))	30	0.250	0.000
			L-BFGS(W(I))	10	0.250	0.000
			L-BFGS(W(F))	10	0.250	0.000
			L-BFGS(W(Q))	10	0.250	0.000
			GA	42792	20.257	6.326
			PBIL	18979	0.500	0.707
			GSA	24602	0.260	0.138
			Random	24202	89.740	13.378
		LogReg	SD(B(I))	21	0.250	0.000
			SD(W(I))	84	0.250	0.000
			SD(W(F))	44	0.250	0.000
			SD(W(Q))	46	0.250	0.000
			BFGS(B(I))	28	0.250	0.000
			BFGS(W(I))	22	0.250	0.000
			BFGS(W(F))	50	0.250	0.000
			BFGS(W(Q))	44	0.250	0.000
			L-BFGS(B(I))	25	0.250	0.000
			L-BFGS(W(I))	18	0.250	0.000
			L-BFGS(W(F))	42	0.250	0.000
			L-BFGS(W(Q))	48	0.250	0.000
			GA	19020	0.250	0.000
			PBIL	18320	0.250	0.000
			GSA	20302	0.250	0.000
			Random	20202	0.250	0.000
		MLP(2)	SD(B(I))	79	0.250	0.000
			SD(W(I))	88	0.250	0.000
			SD(W(F))	112	0.250	0.001

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\|\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\|$
			SD(W(Q))	86	0.250	0.001
			BFGS(B(I))	43	0.250	0.000
			BFGS(W(I))	34	0.250	0.000
			BFGS(W(F))	88	0.250	0.000
			BFGS(W(Q))	72	0.250	0.000
			L-BFGS(B(I))	47	0.250	0.000
			L-BFGS(W(I))	34	0.250	0.000
			L-BFGS(W(F))	74	0.250	0.000
			L-BFGS(W(Q))	66	0.250	0.000
			GA	28004	0.290	0.127
			PBIL	40347	0.252	0.035
			GSA	60902	0.198	0.081
			Random	51002	0.392	0.085
		MLP(2,SM)	SD(B(I))	457	0.063	0.000
			SD(W(I))	564	0.063	0.000
			SD(W(F))	460	0.063	0.001
			SD(W(Q))	442	0.063	0.000
			BFGS(B(I))	117	0.125	0.000
			BFGS(W(I))	180	0.125	0.000
			BFGS(W(F))	338	0.063	0.000
			BFGS(W(Q))	144	0.125	0.000
			L-BFGS(B(I))	375	0.562	0.000
			L-BFGS(W(I))	178	0.063	0.000
			L-BFGS(W(F))	170	0.062	0.000
			L-BFGS(W(Q))	228	0.062	0.000
			GA	21533	0.062	0.000
			PBIL	20679	0.062	0.000
			GSA	20702	0.063	0.000
			Random	25902	0.062	0.000
		MLP(4)	SD(B(I))	501	0.000	0.001
			SD(W(I))	696	0.000	0.000
			SD(W(F))	750	0.000	0.001
			SD(W(Q))	704	0.000	0.000
			BFGS(B(I))	154	0.000	0.000
			BFGS(W(I))	170	0.000	0.000
			BFGS(W(F))	286	0.000	0.000
			BFGS(W(Q))	232	0.000	0.000
			L-BFGS(B(I))	224	0.000	0.000
			L-BFGS(W(I))	302	0.000	0.001
			L-BFGS(W(F))	332	0.000	0.000
			L-BFGS(W(Q))	268	0.000	0.000
			GA	37853	0.329	0.120
			PBIL	58625	0.125	0.009
			GSA	42102	0.136	0.079
			Random	26802	739.854	323.015
		MLP(4,SM)	SD(B(I))	277	0.000	0.000
			SD(W(I))	420	0.000	0.000
			SD(W(F))	388	0.000	0.000

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi_c'$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\lVert\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\rVert$
			SD(W(Q))	96	0.000	0.000
			BFGS(B(I))	66	0.000	0.000
			BFGS(W(I))	144	0.000	0.000
			BFGS(W(F))	90	0.000	0.000
			BFGS(W(Q))	152	0.000	0.000
			L-BFGS(B(I))	79	0.000	0.000
			L-BFGS(W(I))	272	0.000	0.000
			L-BFGS(W(F))	146	0.000	0.000
			L-BFGS(W(Q))	162	0.000	0.000
			GA	19451	0.000	0.000
			PBIL	21102	0.000	0.000
			GSA	21102	0.000	0.000
			Random	20202	0.000	0.000
		MLP(8)	SD(B(I))	851	0.000	0.001
			SD(W(I))	1092	0.000	0.000
			SD(W(F))	844	0.000	0.001
			SD(W(Q))	908	0.000	0.001
			BFGS(B(I))	148	0.000	0.000
			BFGS(W(I))	202	0.000	0.000
			BFGS(W(F))	230	0.000	0.000
			BFGS(W(Q))	210	0.000	0.000
			L-BFGS(B(I))	159	0.000	0.000
			L-BFGS(W(I))	260	0.000	0.000
			L-BFGS(W(F))	264	0.000	0.000
			L-BFGS(W(Q))	256	0.000	0.000
			GA	22163	4197.382	3383.403
			PBIL	158421	0.102	0.038
			GSA	57602	0.187	0.136
			Random	29802	109460.142	17647.991
		MLP(8,SM)	SD(B(I))	327	0.000	0.000
			SD(W(I))	402	0.000	0.000
			SD(W(F))	368	0.000	0.000
			SD(W(Q))	406	0.000	0.000
			BFGS(B(I))	154	0.000	0.000
			BFGS(W(I))	138	0.000	0.000
			BFGS(W(F))	210	0.000	0.000
			BFGS(W(Q))	142	0.000	0.000
			L-BFGS(B(I))	232	0.750	0.000
			L-BFGS(W(I))	112	0.000	0.000
			L-BFGS(W(F))	168	0.000	0.000
			L-BFGS(W(Q))	102	0.000	0.000
			GA	19122	0.000	0.000
			PBIL	20202	0.000	0.000
			GSA	21202	0.000	0.000
			Random	22202	0.000	0.000
		RBF(2)	SD(B(I))	88	0.235	0.000
			SD(W(I))	108	0.235	0.000
			SD(W(F))	140	0.235	0.000

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi_c'$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\lVert\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\rVert$
			SD(W(Q))	120	0.235	0.000
			BFGS(B(I))	53	0.235	0.000
			BFGS(W(I))	30	0.235	0.000
			BFGS(W(F))	96	0.235	0.000
			BFGS(W(Q))	94	0.235	0.000
			L-BFGS(B(I))	53	0.235	0.000
			L-BFGS(W(I))	30	0.235	0.000
			L-BFGS(W(F))	86	0.235	0.000
			L-BFGS(W(Q))	88	0.235	0.000
			GA	24390	0.971	1.248
			PBIL	21330	0.235	0.001
			GSA	52502	0.235	0.030
			Random	34202	4.735	1.817
		RBF(4)	SD(B(I))	79	0.000	0.000
			SD(W(I))	88	0.000	0.000
			SD(W(F))	90	0.000	0.000
			SD(W(Q))	94	0.000	0.000
			BFGS(B(I))	48	0.000	0.000
			BFGS(W(I))	26	0.000	0.000
			BFGS(W(F))	76	0.000	0.000
			BFGS(W(Q))	74	0.000	0.000
			L-BFGS(B(I))	44	0.000	0.000
			L-BFGS(W(I))	26	0.000	0.000
			L-BFGS(W(F))	80	0.000	0.000
			L-BFGS(W(Q))	66	0.000	0.000
			GA	63031	21.452	3.424
			PBIL	26725	0.080	0.352
			GSA	41202	0.012	0.113
			Random	23302	70.139	9.944
		RBF(8)	SD(B(I))	111	0.000	0.000
			SD(W(I))	122	0.000	0.000
			SD(W(F))	110	0.000	0.000
			SD(W(Q))	146	0.000	0.000
			BFGS(B(I))	44	0.000	0.000
			BFGS(W(I))	26	0.000	0.000
			BFGS(W(F))	100	0.000	0.000
			BFGS(W(Q))	78	0.000	0.000
			L-BFGS(B(I))	44	0.000	0.000
			L-BFGS(W(I))	30	0.000	0.000
			L-BFGS(W(F))	84	0.000	0.000
			L-BFGS(W(Q))	66	0.000	0.000
			GA	30525	15.958	4.580
			PBIL	41615	0.005	0.049
			GSA	39702	0.004	0.087
			Random	21502	21.599	3.420
	MAE	LinReg	SD(B(I))	5	0.500	0.000
			SD(W(I))	6	0.500	0.000
			SD(W(F))	6	0.500	0.000

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\\|\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\\|$
			SD(W(Q))	6	0.500	0.000
			BFGS(B(I))	5	0.500	0.000
			BFGS(W(I))	6	0.500	0.000
			BFGS(W(F))	6	0.500	0.000
			BFGS(W(Q))	6	0.500	0.000
			L-BFGS(B(I))	5	0.500	0.000
			L-BFGS(W(I))	6	0.500	0.000
			L-BFGS(W(F))	6	0.500	0.000
			L-BFGS(W(Q))	6	0.500	0.000
			GA	36628	3.412	0.661
			PBIL	17938	0.500	0.612
			GSA	24002	0.500	0.000
			Random	22202	7.257	0.500
		LogReg	SD(B(I))	58	0.250	0.000
			SD(W(I))	166	0.376	0.000
			SD(W(F))	128	0.375	0.000
			SD(W(Q))	98	0.250	0.000
			BFGS(B(I))	11719	0.496	0.217
			BFGS(W(I))	172	0.375	0.000
			BFGS(W(F))	430	0.625	0.331
			BFGS(W(Q))	218	0.374	0.005
			L-BFGS(B(I))	28	0.625	0.000
			L-BFGS(W(I))	462	0.375	0.000
			L-BFGS(W(F))	41292	0.375	0.217
			L-BFGS(W(Q))	270	0.375	0.217
			GA	22426	0.250	0.000
			PBIL	17447	0.250	0.000
			GSA	60102	0.250	0.000
			Random	56102	0.250	0.000
		MLP(2)	SD(B(I))	1077	0.500	0.003
			SD(W(I))	41252	0.462	0.112
			SD(W(F))	41340	0.488	0.047
			SD(W(Q))	41342	0.492	0.047
			BFGS(B(I))	11634	0.432	0.178
			BFGS(W(I))	60042	0.484	0.094
			BFGS(W(F))	41268	0.470	0.283
			BFGS(W(Q))	60438	0.485	0.092
			L-BFGS(B(I))	5663	0.500	0.000
			L-BFGS(W(I))	41244	0.497	0.044
			L-BFGS(W(F))	41262	0.470	0.212
			L-BFGS(W(Q))	41260	0.495	0.065
			GA	34824	0.376	0.216
			PBIL	22721	0.375	0.227
			GSA	48202	0.376	0.208
			Random	51002	0.436	0.111
		MLP(2,SM)	SD(B(I))	273	0.125	0.001
			SD(W(I))	264	0.375	0.000
			SD(W(F))	350	0.125	0.000

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi_c'$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\lVert\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\rVert$
			SD(W(Q))	298	0.125	0.000
			BFGS(B(I))	362	0.375	0.000
			BFGS(W(I))	216	0.125	0.000
			BFGS(W(F))	108	0.375	0.000
			BFGS(W(Q))	180	0.125	0.000
			L-BFGS(B(I))	14	0.375	0.000
			L-BFGS(W(I))	180	0.125	0.000
			L-BFGS(W(F))	170	0.125	0.000
			L-BFGS(W(Q))	164	0.125	0.000
			GA	19662	0.125	0.000
			PBIL	35753	0.071	0.081
			GSA	20702	0.125	0.000
			Random	25902	0.125	0.000
		MLP(4)	SD(B(I))	1083	0.498	0.014
			SD(W(I))	41228	0.498	0.016
			SD(W(F))	41412	0.498	0.016
			SD(W(Q))	41210	0.498	0.016
			BFGS(B(I))	50204	0.000	1.608
			BFGS(W(I))	41032	0.498	0.016
			BFGS(W(F))	41206	0.498	0.429
			BFGS(W(Q))	41014	0.498	0.016
			L-BFGS(B(I))	53882	0.252	0.279
			L-BFGS(W(I))	41220	0.498	0.016
			L-BFGS(W(F))	41200	0.498	0.016
			L-BFGS(W(Q))	41202	0.498	0.016
			GA	32447	0.400	0.172
			PBIL	46082	0.130	0.366
			GSA	42802	0.175	0.367
			Random	26802	16.791	4.391
		MLP(4,SM)	SD(B(I))	152	0.125	0.000
			SD(W(I))	62	0.000	0.000
			SD(W(F))	86	0.000	0.000
			SD(W(Q))	80	0.000	0.000
			BFGS(B(I))	222	0.500	0.000
			BFGS(W(I))	128	0.250	0.000
			BFGS(W(F))	126	0.000	0.000
			BFGS(W(Q))	98	0.250	0.000
			L-BFGS(B(I))	20	0.250	0.000
			L-BFGS(W(I))	48	0.250	0.000
			L-BFGS(W(F))	210	0.000	0.000
			L-BFGS(W(Q))	130	0.000	0.000
			GA	19454	0.000	0.000
			PBIL	22402	0.000	0.000
			GSA	21102	0.000	0.001
			Random	25302	0.000	0.000
		MLP(8)	SD(B(I))	1301	0.494	0.031
			SD(W(I))	41164	0.473	0.305
			SD(W(F))	41512	0.469	0.086

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi_c'$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\|\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\|$
			SD(W(Q))	41212	0.469	0.086
			BFGS(B(I))	28908	0.000	1.397
			BFGS(W(I))	280624	0.000	1.008
			BFGS(W(F))	41016	0.469	0.086
			BFGS(W(Q))	41410	0.469	0.086
			L-BFGS(B(I))	49579	0.000	0.553
			L-BFGS(W(I))	41142	0.469	0.320
			L-BFGS(W(F))	42424	0.469	0.658
			L-BFGS(W(Q))	41204	0.469	0.086
			GA	28257	0.593	0.169
			PBIL	101713	0.125	0.372
			GSA	42902	0.159	0.763
			Random	47202	234.724	36.938
		MLP(8,SM)	SD(B(I))	34	0.250	0.000
			SD(W(I))	242	0.250	0.000
			SD(W(F))	144	0.250	0.000
			SD(W(Q))	104	0.250	0.000
			BFGS(B(I))	115	0.500	0.000
			BFGS(W(I))	56	0.250	0.000
			BFGS(W(F))	110	0.250	0.000
			BFGS(W(Q))	62	0.250	0.000
			L-BFGS(B(I))	37	0.250	0.000
			L-BFGS(W(I))	110	0.250	0.000
			L-BFGS(W(F))	126	0.000	0.000
			L-BFGS(W(Q))	132	0.250	0.000
			GA	19848	0.000	0.000
			PBIL	20902	0.000	0.000
			GSA	21202	0.000	0.000
			Random	23702	0.000	0.000
		RBF(2)	SD(B(I))	1172	0.495	0.075
			SD(W(I))	41684	0.371	0.319
			SD(W(F))	41638	0.438	0.075
			SD(W(Q))	42146	0.395	0.319
			BFGS(B(I))	13334	0.367	0.217
			BFGS(W(I))	79882	0.367	0.162
			BFGS(W(F))	97938	0.367	0.311
			BFGS(W(Q))	120432	0.367	0.381
			L-BFGS(B(I))	13346	0.367	0.232
			L-BFGS(W(I))	41234	0.393	0.075
			L-BFGS(W(F))	41238	0.385	0.075
			L-BFGS(W(Q))	41236	0.385	0.075
			GA	24779	0.839	0.381
			PBIL	18038	0.383	0.075
			GSA	25502	0.396	0.075
			Random	22202	1.603	0.381
		RBF(4)	SD(B(I))	1855	0.026	0.383
			SD(W(I))	42278	0.023	0.387
			SD(W(F))	41822	0.006	0.496

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\|\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\|$
			SD(W(Q))	41826	0.009	0.491
			BFGS(B(I))	18662	0.000	0.299
			BFGS(W(I))	100882	0.000	0.246
			BFGS(W(F))	20540	0.000	0.000
			BFGS(W(Q))	119556	0.000	0.174
			L-BFGS(B(I))	19048	0.000	0.304
			L-BFGS(W(I))	42000	0.000	0.339
			L-BFGS(W(F))	41852	0.000	0.465
			L-BFGS(W(Q))	42856	0.000	0.341
			GA	69292	2.789	0.609
			PBIL	28547	0.040	0.335
			GSA	33002	0.084	0.491
			Random	58702	5.897	0.491
		RBF(8)	SD(B(I))	8515	0.179	0.433
			SD(W(I))	44756	0.009	0.198
			SD(W(F))	41682	0.008	0.198
			SD(W(Q))	42422	0.027	0.353
			BFGS(B(I))	23968	0.000	0.153
			BFGS(W(I))	141310	0.000	0.198
			BFGS(W(F))	888	0.000	0.000
			BFGS(W(Q))	42200	0.000	0.153
			L-BFGS(B(I))	20701	0.000	0.250
			L-BFGS(W(I))	42486	0.000	0.667
			L-BFGS(W(F))	41268	0.000	0.364
			L-BFGS(W(Q))	42336	0.000	0.364
			GA	19957	3.679	0.612
			PBIL	40005	0.004	0.353
			GSA	33102	0.060	0.433
			Random	21502	4.122	0.353
	HL	LinReg	SD(B(I))	5	1.000	0.000
			SD(W(I))	6	1.000	0.000
			SD(W(F))	6	1.000	0.000
			SD(W(Q))	6	1.000	0.000
			BFGS(B(I))	5	1.000	0.000
			BFGS(W(I))	6	1.000	0.000
			BFGS(W(F))	6	1.000	0.000
			BFGS(W(Q))	6	1.000	0.000
			L-BFGS(B(I))	5	1.000	0.000
			L-BFGS(W(I))	6	1.000	0.000
			L-BFGS(W(F))	6	1.000	0.000
			L-BFGS(W(Q))	6	1.000	0.000
			GA	36628	3.912	0.661
			PBIL	17938	1.000	0.612
			GSA	22902	1.000	0.000
			Random	22202	7.757	0.500
		LogReg	SD(B(I))	49	0.500	0.000
			SD(W(I))	192	0.751	0.000
			SD(W(F))	124	0.750	0.000

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\|\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\|$
			SD(W(Q))	102	0.750	0.000
			BFGS(B(I))	133	0.750	0.000
			BFGS(W(I))	114	0.750	0.000
			BFGS(W(F))	314	0.750	0.433
			BFGS(W(Q))	41514	0.500	0.433
			L-BFGS(B(I))	28	1.250	0.000
			L-BFGS(W(I))	460	0.750	0.000
			L-BFGS(W(F))	288	0.750	0.433
			L-BFGS(W(Q))	41272	0.750	0.433
			GA	22426	0.500	0.000
			PBIL	18186	0.500	0.000
			GSA	58902	0.500	0.000
			Random	56102	0.500	0.000
		MLP(2)	SD(B(I))	1067	1.000	0.006
			SD(W(I))	41252	0.936	0.433
			SD(W(F))	41244	0.975	0.097
			SD(W(Q))	41976	0.903	0.254
			BFGS(B(I))	3889	1.000	0.000
			BFGS(W(I))	60774	0.500	0.001
			BFGS(W(F))	60384	0.502	0.015
			BFGS(W(Q))	60550	0.501	0.003
			L-BFGS(B(I))	7842	0.500	0.000
			L-BFGS(W(I))	41542	0.989	0.114
			L-BFGS(W(F))	44776	0.536	1.374
			L-BFGS(W(Q))	41916	0.800	1.287
			GA	30200	0.251	0.001
			PBIL	20497	0.500	0.000
			GSA	92402	0.250	0.000
			Random	37002	0.250	0.000
		MLP(2,SM)	SD(B(I))	112	0.250	0.000
			SD(W(I))	106	0.750	0.000
			SD(W(F))	364	0.250	0.001
			SD(W(Q))	346	0.250	0.001
			BFGS(B(I))	18	0.750	0.000
			BFGS(W(I))	246	0.250	0.000
			BFGS(W(F))	210	0.250	0.000
			BFGS(W(Q))	122	0.750	0.000
			L-BFGS(B(I))	14	0.750	0.000
			L-BFGS(W(I))	130	0.750	0.000
			L-BFGS(W(F))	114	0.750	0.000
			L-BFGS(W(Q))	254	0.250	0.001
			GA	19662	0.250	0.000
			PBIL	35639	0.143	0.161
			GSA	20702	0.250	0.000
			Random	25902	0.250	0.000
		MLP(4)	SD(B(I))	1630	0.996	0.027
			SD(W(I))	41224	0.995	0.033
			SD(W(F))	41210	0.995	0.031

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\|\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\|$
			SD(W(Q))	41412	0.995	0.031
			BFGS(B(I))	1475	0.000	0.000
			BFGS(W(I))	41222	0.995	0.424
			BFGS(W(F))	41206	0.995	0.424
			BFGS(W(Q))	41206	0.995	0.424
			L-BFGS(B(I))	2532	0.250	0.000
			L-BFGS(W(I))	41422	0.995	0.032
			L-BFGS(W(F))	42412	0.995	0.032
			L-BFGS(W(Q))	41202	0.995	0.032
			GA	31468	0.000	0.000
			PBIL	20602	0.000	0.000
			GSA	21102	0.410	4.492
			Random	24702	0.000	0.000
		MLP(4,SM)	SD(B(I))	241	0.000	0.000
			SD(W(I))	58	0.000	0.000
			SD(W(F))	78	0.000	0.000
			SD(W(Q))	88	0.000	0.000
			BFGS(B(I))	164	0.750	0.000
			BFGS(W(I))	100	0.500	0.000
			BFGS(W(F))	120	0.000	0.000
			BFGS(W(Q))	72	0.500	0.000
			L-BFGS(B(I))	19	0.500	0.000
			L-BFGS(W(I))	244	0.000	0.000
			L-BFGS(W(F))	82	0.000	0.000
			L-BFGS(W(Q))	56	0.000	0.000
			GA	19365	0.000	0.000
			PBIL	20602	0.000	0.000
			GSA	21102	0.000	0.002
			Random	20202	0.000	0.000
		MLP(8)	SD(B(I))	1081	0.987	0.063
			SD(W(I))	41640	0.787	0.704
			SD(W(F))	41584	0.845	0.730
			SD(W(Q))	41880	0.717	0.803
			BFGS(B(I))	2805	0.000	0.000
			BFGS(W(I))	310	0.000	0.000
			BFGS(W(F))	41434	0.904	0.244
			BFGS(W(Q))	19704	0.000	0.000
			L-BFGS(B(I))	1499	0.000	0.000
			L-BFGS(W(I))	128	0.000	0.000
			L-BFGS(W(F))	1118	0.000	0.000
			L-BFGS(W(Q))	41436	0.879	0.788
			GA	20842	0.000	0.000
			PBIL	21602	0.000	0.000
			GSA	24102	0.000	0.000
			Random	23702	0.000	0.000
		MLP(8,SM)	SD(B(I))	70	0.500	0.000
			SD(W(I))	262	0.000	0.000
			SD(W(F))	212	0.000	0.000

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\|\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\|$
			SD(W(Q))	154	0.000	0.000
			BFGS(B(I))	65	0.500	0.000
			BFGS(W(I))	86	0.500	0.000
			BFGS(W(F))	290	0.500	0.000
			BFGS(W(Q))	278	1.500	0.000
			L-BFGS(B(I))	53	0.500	0.000
			L-BFGS(W(I))	118	0.500	0.000
			L-BFGS(W(F))	64	0.500	0.000
			L-BFGS(W(Q))	66	0.500	0.000
			GA	19092	0.000	0.000
			PBIL	20202	0.000	0.000
			GSA	21102	0.000	0.000
			Random	20302	0.000	0.000
		RBF(2)	SD(B(I))	1173	0.987	0.150
			SD(W(I))	41108	0.754	0.365
			SD(W(F))	41480	0.798	0.464
			SD(W(Q))	41364	0.744	0.325
			BFGS(B(I))	13166	0.734	0.623
			BFGS(W(I))	336686	0.734	0.493
			BFGS(W(F))	135638	0.734	0.434
			BFGS(W(Q))	136456	0.734	0.325
			L-BFGS(B(I))	12761	0.734	0.365
			L-BFGS(W(I))	41258	0.749	0.325
			L-BFGS(W(F))	41472	0.741	0.365
			L-BFGS(W(Q))	41654	0.747	0.325
			GA	26823	1.368	0.686
			PBIL	16487	0.738	0.365
			GSA	50802	0.753	0.150
			Random	22202	1.627	0.365
		RBF(4)	SD(B(I))	33	0.000	0.000
			SD(W(I))	28	0.000	0.000
			SD(W(F))	38	0.000	0.000
			SD(W(Q))	24	0.000	0.000
			BFGS(B(I))	24	0.000	0.000
			BFGS(W(I))	24	0.000	0.000
			BFGS(W(F))	20	0.000	0.000
			BFGS(W(Q))	24	0.000	0.000
			L-BFGS(B(I))	26	0.000	0.000
			L-BFGS(W(I))	26	0.000	0.000
			L-BFGS(W(F))	28	0.000	0.000
			L-BFGS(W(Q))	24	0.000	0.000
			GA	19164	0.000	0.000
			PBIL	20745	0.000	0.000
			GSA	22102	0.000	0.000
			Random	21002	0.000	0.000
		RBF(8)	SD(B(I))	45	0.000	0.000
			SD(W(I))	68	0.000	0.000
			SD(W(F))	86	0.000	0.000

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\lVert\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\rVert$
			SD(W(Q))	38	0.000	0.000
			BFGS(B(I))	30	0.000	0.000
			BFGS(W(I))	30	0.000	0.000
			BFGS(W(F))	22	0.000	0.000
			BFGS(W(Q))	32	0.000	0.000
			L-BFGS(B(I))	31	0.000	0.000
			L-BFGS(W(I))	40	0.000	0.000
			L-BFGS(W(F))	30	0.000	0.000
			L-BFGS(W(Q))	22	0.000	0.000
			GA	19177	0.000	0.000
			PBIL	21202	0.000	0.000
			GSA	22302	0.000	0.000
			Random	22802	0.000	0.000
Iris	MSE	LinReg	SD(B(I))	321	0.093	0.000
			SD(W(I))	308	0.093	0.000
			SD(W(F))	304	0.093	0.000
			SD(W(Q))	370	0.093	0.001
			BFGS(B(I))	76	0.093	0.000
			BFGS(W(I))	90	0.093	0.000
			BFGS(W(F))	126	0.093	0.000
			BFGS(W(Q))	108	0.093	0.000
			L-BFGS(B(I))	99	0.093	0.000
			L-BFGS(W(I))	70	0.093	0.000
			L-BFGS(W(F))	116	0.093	0.000
			L-BFGS(W(Q))	104	0.093	0.000
			GA	29841	70.541	6.955
			PBIL	42079	3.032	0.766
			GSA	32702	0.172	0.177
			Random	21802	139.813	6.863
		LogReg	SD(B(I))	518	0.063	0.000
			SD(W(I))	666	0.063	0.000
			SD(W(F))	468	0.063	0.001
			SD(W(Q))	598	0.063	0.001
			BFGS(B(I))	195	0.062	0.000
			BFGS(W(I))	268	0.062	0.000
			BFGS(W(F))	440	0.062	0.000
			BFGS(W(Q))	350	0.062	0.000
			L-BFGS(B(I))	178	0.060	0.000
			L-BFGS(W(I))	98	0.062	0.000
			L-BFGS(W(F))	144	0.062	0.001
			L-BFGS(W(Q))	144	0.062	0.001
			GA	31752	0.090	0.004
			PBIL	60164	0.067	0.000
			GSA	47402	0.063	0.003
			Random	25102	0.131	0.004
		MLP(2)	SD(B(I))	3238	0.113	0.000
			SD(W(I))	3894	0.113	0.001
			SD(W(F))	2588	0.113	0.001

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\lVert\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\rVert$
			SD(W(Q))	4740	0.113	0.001
			BFGS(B(I))	223	0.113	0.000
			BFGS(W(I))	320	0.113	0.000
			BFGS(W(F))	1068	0.112	0.000
			BFGS(W(Q))	274	0.113	0.000
			L-BFGS(B(I))	1018	0.112	0.001
			L-BFGS(W(I))	664	0.113	0.000
			L-BFGS(W(F))	1324	0.112	0.000
			L-BFGS(W(Q))	1586	0.112	0.000
			GA	40140	0.315	0.018
			PBIL	48922	0.318	0.001
			GSA	40202	0.224	0.033
			Random	37902	0.328	0.005
		MLP(2,SM)	SD(B(I))	4102	0.008	0.000
			SD(W(I))	4938	0.008	0.001
			SD(W(F))	3968	0.008	0.001
			SD(W(Q))	5500	0.008	0.001
			BFGS(B(I))	14	0.222	0.000
			BFGS(W(I))	392	0.007	0.000
			BFGS(W(F))	344	0.007	0.000
			BFGS(W(Q))	298	0.007	0.000
			L-BFGS(B(I))	17	0.230	0.000
			L-BFGS(W(I))	266	0.007	0.000
			L-BFGS(W(F))	404	0.007	0.001
			L-BFGS(W(Q))	342	0.007	0.000
			GA	45089	0.082	0.003
			PBIL	56003	0.077	0.000
			GSA	42102	0.069	0.006
			Random	21902	0.041	0.163
		MLP(4)	SD(B(I))	45852	0.018	0.005
			SD(W(I))	51424	0.020	0.005
			SD(W(F))	40164	0.019	0.002
			SD(W(Q))	40030	0.021	0.002
			BFGS(B(I))	1791	0.011	0.001
			BFGS(W(I))	2350	0.012	0.001
			BFGS(W(F))	3652	0.012	0.000
			BFGS(W(Q))	1476	0.013	0.001
			L-BFGS(B(I))	8617	0.011	0.001
			L-BFGS(W(I))	7402	0.012	0.000
			L-BFGS(W(F))	5668	0.013	0.000
			L-BFGS(W(Q))	8186	0.011	0.000
			GA	52656	0.328	0.011
			PBIL	89282	0.294	0.002
			GSA	39802	0.140	0.066
			Random	32102	0.337	0.013
		MLP(4,SM)	SD(B(I))	4478	0.008	0.000
			SD(W(I))	5426	0.008	0.000
			SD(W(F))	3552	0.008	0.001

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi_c'$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\lVert\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\rVert$
			SD(W(Q))	5676	0.008	0.000
			BFGS(B(I))	244	0.007	0.000
			BFGS(W(I))	432	0.007	0.000
			BFGS(W(F))	386	0.007	0.000
			BFGS(W(Q))	340	0.007	0.000
			L-BFGS(B(I))	352	0.007	0.000
			L-BFGS(W(I))	310	0.007	0.000
			L-BFGS(W(F))	306	0.007	0.000
			L-BFGS(W(Q))	254	0.007	0.000
			GA	20630	0.049	0.002
			PBIL	76464	0.009	0.000
			GSA	58102	0.008	0.001
			Random	55102	0.040	0.002
		MLP(8)	SD(B(I))	45851	0.022	0.009
			SD(W(I))	51752	0.023	0.004
			SD(W(F))	40172	0.023	0.003
			SD(W(Q))	40086	0.024	0.003
			BFGS(B(I))	2480	0.010	0.001
			BFGS(W(I))	4092	0.010	0.001
			BFGS(W(F))	5310	0.011	0.001
			BFGS(W(Q))	3304	0.010	0.001
			L-BFGS(B(I))	23580	0.007	0.000
			L-BFGS(W(I))	7854	0.011	0.001
			L-BFGS(W(F))	7104	0.011	0.001
			L-BFGS(W(Q))	10854	0.010	0.002
			GA	28242	1466.307	445.059
			PBIL	196877	0.286	0.004
			GSA	41402	0.113	0.017
			Random	40002	102018.850	9840.083
		MLP(8,SM)	SD(B(I))	4267	0.008	0.000
			SD(W(I))	5478	0.008	0.000
			SD(W(F))	4246	0.008	0.001
			SD(W(Q))	5346	0.008	0.001
			BFGS(B(I))	271	0.007	0.000
			BFGS(W(I))	398	0.007	0.000
			BFGS(W(F))	420	0.007	0.000
			BFGS(W(Q))	346	0.007	0.000
			L-BFGS(B(I))	163	0.007	0.000
			L-BFGS(W(I))	200	0.007	0.000
			L-BFGS(W(F))	238	0.008	0.001
			L-BFGS(W(Q))	170	0.007	0.000
			GA	37730	0.022	0.000
			PBIL	29402	0.007	0.000
			GSA	105902	0.004	0.011
			Random	35002	0.022	0.000
		RBF(2)	SD(B(I))	93	0.114	0.000
			SD(W(I))	90	0.114	0.000
			SD(W(F))	108	0.114	0.000

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\lVert\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\rVert$
			SD(W(Q))	98	0.114	0.000
			BFGS(B(I))	39	0.114	0.000
			BFGS(W(I))	30	0.114	0.000
			BFGS(W(F))	66	0.114	0.000
			BFGS(W(Q))	68	0.114	0.000
			L-BFGS(B(I))	35	0.114	0.000
			L-BFGS(W(I))	30	0.114	0.000
			L-BFGS(W(F))	88	0.114	0.000
			L-BFGS(W(Q))	84	0.114	0.000
			GA	33385	10.802	4.595
			PBIL	30202	0.134	0.123
			GSA	26602	0.121	0.065
			Random	35602	9.263	2.806
		RBF(4)	SD(B(I))	940	0.054	0.000
			SD(W(I))	1080	0.054	0.001
			SD(W(F))	866	0.054	0.001
			SD(W(Q))	1134	0.054	0.001
			BFGS(B(I))	85	0.054	0.000
			BFGS(W(I))	90	0.054	0.000
			BFGS(W(F))	138	0.054	0.000
			BFGS(W(Q))	120	0.054	0.000
			L-BFGS(B(I))	91	0.054	0.000
			L-BFGS(W(I))	78	0.054	0.000
			L-BFGS(W(F))	132	0.054	0.000
			L-BFGS(W(Q))	144	0.054	0.000
			GA	44583	9.642	2.346
			PBIL	59603	0.757	0.161
			GSA	62302	0.083	0.076
			Random	20202	46.678	4.049
		RBF(8)	SD(B(I))	2146	0.032	0.000
			SD(W(I))	2678	0.032	0.000
			SD(W(F))	1918	0.032	0.000
			SD(W(Q))	2550	0.031	0.001
			BFGS(B(I))	158	0.032	0.000
			BFGS(W(I))	182	0.032	0.000
			BFGS(W(F))	256	0.032	0.000
			BFGS(W(Q))	232	0.032	0.000
			L-BFGS(B(I))	262	0.031	0.001
			L-BFGS(W(I))	270	0.031	0.000
			L-BFGS(W(F))	212	0.031	0.000
			L-BFGS(W(Q))	246	0.031	0.000
			GA	49269	35.191	4.424
			PBIL	89265	7.001	0.564
			GSA	54702	0.061	0.045
			Random	48002	59.368	5.134
	MAE	LinReg	SD(B(I))	9580	0.231	0.021
			SD(W(I))	55142	0.231	0.017
			SD(W(F))	60412	0.231	0.016

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi_c'$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\|\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\|$
			SD(W(Q))	47900	0.231	0.017
			BFGS(B(I))	14671	0.230	0.115
			BFGS(W(I))	374694	0.230	0.011
			BFGS(W(F))	464668	0.230	0.022
			BFGS(W(Q))	384840	0.230	0.025
			L-BFGS(B(I))	56715	0.230	0.113
			L-BFGS(W(I))	43380	0.230	0.108
			L-BFGS(W(F))	41546	0.230	0.014
			L-BFGS(W(Q))	41930	0.230	0.029
			GA	48962	5.540	0.404
			PBIL	52922	1.578	0.082
			GSA	78802	0.261	0.254
			Random	21802	9.000	0.227
		LogReg	SD(B(I))	368	0.099	0.001
			SD(W(I))	474	0.099	0.000
			SD(W(F))	212	0.123	0.001
			SD(W(Q))	462	0.097	0.001
			BFGS(B(I))	58	0.133	0.000
			BFGS(W(I))	86	0.133	0.000
			BFGS(W(F))	240	0.133	0.009
			BFGS(W(Q))	236	0.133	0.009
			L-BFGS(B(I))	54	0.123	0.000
			L-BFGS(W(I))	96	0.120	0.001
			L-BFGS(W(F))	72	0.124	0.001
			L-BFGS(W(Q))	56	0.122	0.000
			GA	31045	0.094	0.003
			PBIL	25020	0.101	0.002
			GSA	87902	0.089	0.000
			Random	41302	0.142	0.001
		MLP(2)	SD(B(I))	16269	0.327	0.097
			SD(W(I))	43308	0.318	0.106
			SD(W(F))	46288	0.322	0.082
			SD(W(Q))	49038	0.324	0.088
			BFGS(B(I))	80219	0.257	2.207
			BFGS(W(I))	568378	0.154	2.521
			BFGS(W(F))	379068	0.155	2.089
			BFGS(W(Q))	224436	0.157	1.376
			L-BFGS(B(I))	15387	0.242	0.009
			L-BFGS(W(I))	41850	0.228	0.142
			L-BFGS(W(F))	43222	0.179	0.926
			L-BFGS(W(Q))	41966	0.190	0.518
			GA	55042	0.329	0.007
			PBIL	41646	0.313	0.007
			GSA	23502	0.265	0.063
			Random	37902	0.332	0.010
		MLP(2,SM)	SD(B(I))	3634	0.008	0.000
			SD(W(I))	4104	0.008	0.001
			SD(W(F))	2744	0.008	0.001

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\|\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\|$
			SD(W(Q))	4070	0.008	0.001
			BFGS(B(I))	211	0.220	0.000
			BFGS(W(I))	294	0.022	0.000
			BFGS(W(F))	406	0.022	0.000
			BFGS(W(Q))	216	0.022	0.000
			L-BFGS(B(I))	14	0.444	0.000
			L-BFGS(W(I))	236	0.022	0.000
			L-BFGS(W(F))	180	0.022	0.000
			L-BFGS(W(Q))	268	0.141	0.003
			GA	38649	0.015	0.000
			PBIL	57825	0.146	0.001
			GSA	110802	0.007	0.000
			Random	21902	0.042	0.116
		MLP(4)	SD(B(I))	7449	0.232	0.021
			SD(W(I))	75130	0.115	0.024
			SD(W(F))	40170	0.167	0.051
			SD(W(Q))	42446	0.144	0.035
			BFGS(B(I))	244817	0.022	1.626
			BFGS(W(I))	428324	0.030	0.172
			BFGS(W(F))	236302	0.034	0.123
			BFGS(W(Q))	517028	0.052	0.357
			L-BFGS(B(I))	139003	0.259	3.541
			L-BFGS(W(I))	57702	0.100	0.044
			L-BFGS(W(F))	45190	0.092	0.072
			L-BFGS(W(Q))	44440	0.096	0.044
			GA	20256	0.332	0.003
			PBIL	70995	0.309	0.007
			GSA	28902	0.214	0.061
			Random	32102	0.339	0.011
		MLP(4,SM)	SD(B(I))	3675	0.008	0.000
			SD(W(I))	96	0.183	0.018
			SD(W(F))	3222	0.008	0.001
			SD(W(Q))	4678	0.008	0.001
			BFGS(B(I))	196	0.030	0.000
			BFGS(W(I))	328	0.104	0.000
			BFGS(W(F))	244	0.119	0.000
			BFGS(W(Q))	514	0.232	0.000
			L-BFGS(B(I))	22	0.652	0.000
			L-BFGS(W(I))	330	0.015	0.000
			L-BFGS(W(F))	204	0.015	0.000
			L-BFGS(W(Q))	210	0.133	0.000
			GA	25359	0.030	0.000
			PBIL	28002	0.007	0.000
			GSA	20202	0.148	0.000
			Random	38602	0.045	0.033
		MLP(8)	SD(B(I))	19965	0.233	0.032
			SD(W(I))	42550	0.232	0.035
			SD(W(F))	40116	0.233	0.021

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\lVert\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\rVert$
			SD(W(Q))	62644	0.233	0.029
			BFGS(B(I))	88939	0.009	1.318
			BFGS(W(I))	232264	0.010	0.644
			BFGS(W(F))	98782	0.015	0.544
			BFGS(W(Q))	216770	0.012	0.939
			L-BFGS(B(I))	271493	0.065	0.029
			L-BFGS(W(I))	43190	0.109	0.034
			L-BFGS(W(F))	53406	0.086	0.028
			L-BFGS(W(Q))	42662	0.112	0.037
			GA	62045	9.332	2.404
			PBIL	203506	0.306	0.007
			GSA	97702	0.208	0.043
			Random	27102	152.290	9.210
		MLP(8,SM)	SD(B(I))	3180	0.008	0.000
			SD(W(I))	3960	0.008	0.000
			SD(W(F))	2588	0.008	0.001
			SD(W(Q))	3516	0.008	0.001
			BFGS(B(I))	154	0.022	0.000
			BFGS(W(I))	196	0.030	0.000
			BFGS(W(F))	172	0.022	0.000
			BFGS(W(Q))	206	0.030	0.000
			L-BFGS(B(I))	11	0.667	0.000
			L-BFGS(W(I))	214	0.015	0.000
			L-BFGS(W(F))	150	0.015	0.000
			L-BFGS(W(Q))	178	0.015	0.000
			GA	22729	0.022	0.000
			PBIL	32502	0.007	0.000
			GSA	299802	0.007	0.000
			Random	35002	0.022	0.000
		RBF(2)	SD(B(I))	10097	0.288	0.111
			SD(W(I))	47360	0.283	0.172
			SD(W(F))	45974	0.273	0.109
			SD(W(Q))	45490	0.280	0.180
			BFGS(B(I))	11724	0.231	0.411
			BFGS(W(I))	210130	0.231	0.055
			BFGS(W(F))	251310	0.231	0.160
			BFGS(W(Q))	273392	0.231	0.140
			L-BFGS(B(I))	3215	0.231	0.139
			L-BFGS(W(I))	41790	0.231	0.412
			L-BFGS(W(F))	42508	0.231	0.140
			L-BFGS(W(Q))	41790	0.231	0.211
			GA	35757	1.678	0.618
			PBIL	24818	0.246	0.095
			GSA	49902	0.248	0.270
			Random	35602	2.561	0.479
		RBF(4)	SD(B(I))	14104	0.137	0.014
			SD(W(I))	41880	0.135	0.010
			SD(W(F))	40536	0.136	0.015

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\Vert\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\Vert$
			SD(W(Q))	84400	0.137	0.017
			BFGS(B(I))	19869	0.133	0.010
			BFGS(W(I))	41670	0.133	0.002
			BFGS(W(F))	341410	0.133	0.009
			BFGS(W(Q))	63402	0.133	0.002
			L-BFGS(B(I))	24098	0.133	0.004
			L-BFGS(W(I))	41916	0.135	0.010
			L-BFGS(W(F))	41630	0.135	0.027
			L-BFGS(W(Q))	43304	0.133	0.015
			GA	42671	3.090	0.313
			PBIL	37491	1.131	0.137
			GSA	62102	0.165	0.219
			Random	71802	4.404	0.271
		RBF(8)	SD(B(I))	45864	0.121	0.027
			SD(W(I))	42888	0.117	0.018
			SD(W(F))	40252	0.119	0.029
			SD(W(Q))	44382	0.119	0.030
			BFGS(B(I))	24960	0.095	0.012
			BFGS(W(I))	211676	0.095	0.012
			BFGS(W(F))	422582	0.095	0.020
			BFGS(W(Q))	519926	0.095	0.013
			L-BFGS(B(I))	38239	0.095	0.010
			L-BFGS(W(I))	43194	0.096	0.018
			L-BFGS(W(F))	43308	0.095	0.010
			L-BFGS(W(Q))	43824	0.095	0.014
			GA	63173	4.375	0.395
			PBIL	84305	1.718	0.061
			GSA	78002	0.156	0.052
			Random	26302	7.029	0.217
	HL	LinReg	SD(B(I))	6415	0.227	0.024
			SD(W(I))	44278	0.219	0.014
			SD(W(F))	50602	0.219	0.014
			SD(W(Q))	43056	0.219	0.013
			BFGS(B(I))	11152	0.205	0.010
			BFGS(W(I))	426142	0.205	0.012
			BFGS(W(F))	41704	0.206	0.010
			BFGS(W(Q))	81018	0.206	0.004
			L-BFGS(B(I))	32230	0.208	0.013
			L-BFGS(W(I))	41918	0.207	0.010
			L-BFGS(W(F))	42266	0.208	0.009
			L-BFGS(W(Q))	41844	0.205	0.012
			GA	40260	1.288	0.201
			PBIL	51681	0.345	0.056
			GSA	56002	0.223	0.076
			Random	25102	2.720	0.128
		LogReg	SD(B(I))	887	0.185	0.000
			SD(W(I))	1152	0.185	0.000
			SD(W(F))	954	0.182	0.001

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\|\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\|$
			SD(W(Q))	970	0.183	0.001
			BFGS(B(I))	77	0.259	0.000
			BFGS(W(I))	144	0.259	0.000
			BFGS(W(F))	41030	0.267	0.009
			BFGS(W(Q))	312	0.259	0.017
			L-BFGS(B(I))	66	0.245	0.000
			L-BFGS(W(I))	94	0.238	0.000
			L-BFGS(W(F))	80	0.247	0.000
			L-BFGS(W(Q))	56	0.244	0.000
			GA	31045	0.188	0.007
			PBIL	25020	0.201	0.003
			GSA	110802	0.178	0.000
			Random	41302	0.283	0.002
		MLP(2)	SD(B(I))	1483	0.643	0.093
			SD(W(I))	42218	0.362	0.349
			SD(W(F))	41728	0.343	0.158
			SD(W(Q))	41878	0.635	0.208
			BFGS(B(I))	2394	0.259	0.000
			BFGS(W(I))	900796	0.230	0.289
			BFGS(W(F))	398654	0.236	0.223
			BFGS(W(Q))	688398	0.233	0.583
			L-BFGS(B(I))	4130	0.445	0.000
			L-BFGS(W(I))	41510	0.255	0.197
			L-BFGS(W(F))	42026	0.244	0.202
			L-BFGS(W(Q))	41548	0.271	0.230
			GA	25522	0.279	0.016
			PBIL	53254	0.246	0.001
			GSA	237002	0.238	0.002
			Random	81602	0.305	0.007
		MLP(2,SM)	SD(B(I))	4395	0.015	0.000
			SD(W(I))	5398	0.015	0.000
			SD(W(F))	3530	0.015	0.001
			SD(W(Q))	5186	0.015	0.000
			BFGS(B(I))	44	0.444	0.000
			BFGS(W(I))	336	0.059	0.000
			BFGS(W(F))	310	0.044	0.000
			BFGS(W(Q))	216	0.044	0.000
			L-BFGS(B(I))	14	0.889	0.000
			L-BFGS(W(I))	154	0.059	0.000
			L-BFGS(W(F))	214	0.030	0.000
			L-BFGS(W(Q))	184	0.059	0.000
			GA	38649	0.030	0.000
			PBIL	57825	0.292	0.001
			GSA	149202	0.015	0.000
			Random	21902	0.085	0.231
		MLP(4)	SD(B(I))	5198	0.063	0.033
			SD(W(I))	45098	0.056	0.036
			SD(W(F))	46610	0.056	0.033

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi_c'$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\lVert\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\rVert$
			SD(W(Q))	44814	0.057	0.036
			BFGS(B(I))	242118	0.016	0.010
			BFGS(W(I))	117534	0.029	0.012
			BFGS(W(F))	46316	0.030	0.009
			BFGS(W(Q))	175422	0.028	0.006
			L-BFGS(B(I))	25336	0.030	0.006
			L-BFGS(W(I))	41586	0.040	0.005
			L-BFGS(W(F))	41540	0.037	0.006
			L-BFGS(W(Q))	41668	0.046	0.014
			GA	54828	0.257	0.011
			PBIL	176872	0.122	0.308
			GSA	51902	0.033	0.030
			Random	32102	0.668	0.003
		MLP(4,SM)	SD(B(I))	4849	0.015	0.000
			SD(W(I))	4672	0.015	0.000
			SD(W(F))	4180	0.015	0.001
			SD(W(Q))	5974	0.015	0.001
			BFGS(B(I))	291	0.044	0.000
			BFGS(W(I))	252	0.356	0.000
			BFGS(W(F))	162	0.044	0.000
			BFGS(W(Q))	124	0.356	0.000
			L-BFGS(B(I))	31	1.319	0.000
			L-BFGS(W(I))	202	0.356	0.000
			L-BFGS(W(F))	192	0.030	0.000
			L-BFGS(W(Q))	112	0.296	0.000
			GA	25359	0.059	0.000
			PBIL	28002	0.015	0.000
			GSA	20202	0.296	0.000
			Random	38602	0.090	0.066
		MLP(8)	SD(B(I))	8012	0.066	0.041
			SD(W(I))	49232	0.052	0.025
			SD(W(F))	50496	0.057	0.025
			SD(W(Q))	46326	0.058	0.040
			BFGS(B(I))	12433	0.000	0.000
			BFGS(W(I))	561908	0.027	0.024
			BFGS(W(F))	65752	0.029	0.006
			BFGS(W(Q))	107160	0.025	0.008
			L-BFGS(B(I))	37561	0.029	0.033
			L-BFGS(W(I))	42908	0.039	0.020
			L-BFGS(W(F))	44860	0.032	0.012
			L-BFGS(W(Q))	41604	0.045	0.039
			GA	23128	1.645	0.784
			PBIL	164611	0.229	0.009
			GSA	117902	0.015	0.008
			Random	35502	28.939	4.125
		MLP(8,SM)	SD(B(I))	4372	0.015	0.000
			SD(W(I))	108	0.279	0.086
			SD(W(F))	3604	0.015	0.001

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\|\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\|$
			SD(W(Q))	4738	0.015	0.001
			BFGS(B(I))	14	0.444	0.000
			BFGS(W(I))	318	0.030	0.000
			BFGS(W(F))	184	0.059	0.000
			BFGS(W(Q))	276	0.059	0.000
			L-BFGS(B(I))	11	1.333	0.000
			L-BFGS(W(I))	228	0.030	0.000
			L-BFGS(W(F))	142	0.030	0.000
			L-BFGS(W(Q))	240	0.044	0.000
			GA	22729	0.044	0.000
			PBIL	32502	0.015	0.000
			GSA	139502	0.003	0.016
			Random	35002	0.044	0.000
		RBF(2)	SD(B(I))	1851	0.456	0.310
			SD(W(I))	41912	0.416	0.266
			SD(W(F))	41616	0.406	0.095
			SD(W(Q))	41480	0.420	0.210
			BFGS(B(I))	7503	0.262	0.189
			BFGS(W(I))	435170	0.262	0.283
			BFGS(W(F))	207080	0.262	0.092
			BFGS(W(Q))	191120	0.262	0.092
			L-BFGS(B(I))	12328	0.262	0.092
			L-BFGS(W(I))	41588	0.263	0.279
			L-BFGS(W(F))	41474	0.262	0.054
			L-BFGS(W(Q))	41328	0.263	0.065
			GA	31633	0.524	0.286
			PBIL	24755	0.737	0.054
			GSA	73202	0.266	0.283
			Random	33502	0.885	0.190
		RBF(4)	SD(B(I))	2547	0.176	0.013
			SD(W(I))	42006	0.173	0.009
			SD(W(F))	42328	0.174	0.010
			SD(W(Q))	41964	0.174	0.010
			BFGS(B(I))	41247	0.121	0.010
			BFGS(W(I))	86634	0.156	0.009
			BFGS(W(F))	86284	0.154	0.002
			BFGS(W(Q))	616468	0.122	0.002
			L-BFGS(B(I))	20650	0.140	0.009
			L-BFGS(W(I))	41372	0.170	0.009
			L-BFGS(W(F))	41634	0.169	0.002
			L-BFGS(W(Q))	41778	0.151	0.010
			GA	20864	0.343	0.021
			PBIL	42507	0.233	0.014
			GSA	37102	0.179	0.019
			Random	40802	0.843	0.220
		RBF(8)	SD(B(I))	5765	0.084	0.013
			SD(W(I))	47410	0.069	0.013
			SD(W(F))	40368	0.067	0.009

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi_c'$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\lVert\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\rVert$
			SD(W(Q))	46114	0.069	0.009
			BFGS(B(I))	42323	0.035	0.002
			BFGS(W(I))	490796	0.037	0.001
			BFGS(W(F))	1116188	0.035	0.008
			BFGS(W(Q))	1142390	0.035	0.008
			L-BFGS(B(I))	19857	0.041	0.001
			L-BFGS(W(I))	65118	0.045	0.001
			L-BFGS(W(F))	41454	0.051	0.003
			L-BFGS(W(Q))	42116	0.044	0.008
			GA	30976	0.302	0.075
			PBIL	82888	0.065	0.002
			GSA	138702	0.047	0.012
			Random	52302	1.588	0.244
Cancer	MSE	LinReg	SD(B(I))	45482	0.052	0.000
			SD(W(I))	52322	0.052	0.001
			SD(W(F))	40292	0.052	0.001
			SD(W(Q))	40028	0.052	0.001
			BFGS(B(I))	453	0.052	0.000
			BFGS(W(I))	968	0.052	0.000
			BFGS(W(F))	1100	0.052	0.000
			BFGS(W(Q))	504	0.052	0.001
			L-BFGS(B(I))	1920	0.052	0.001
			L-BFGS(W(I))	2886	0.052	0.000
			L-BFGS(W(F))	2006	0.052	0.000
			L-BFGS(W(Q))	1882	0.052	0.002
			GA	37081	456.203	14.028
			PBIL	216327	44.557	1.474
			GSA	152502	0.332	0.607
			Random	37902	402.979	14.187
		LogReg	SD(B(I))	2220	0.015	0.000
			SD(W(I))	3046	0.014	0.000
			SD(W(F))	1992	0.014	0.001
			SD(W(Q))	2728	0.014	0.001
			BFGS(B(I))	10605	0.009	0.014
			BFGS(W(I))	402	0.013	0.000
			BFGS(W(F))	780	0.012	0.000
			BFGS(W(Q))	322	0.014	0.000
			L-BFGS(B(I))	248	0.004	0.000
			L-BFGS(W(I))	470	0.005	0.000
			L-BFGS(W(F))	134	0.019	0.000
			L-BFGS(W(Q))	154	0.012	0.001
			GA	28588	0.061	0.003
			PBIL	188124	0.014	0.000
			GSA	127602	0.013	0.002
			Random	32402	0.087	0.003
		MLP(2)	SD(B(I))	45854	0.040	0.003
			SD(W(I))	51526	0.040	0.003
			SD(W(F))	40188	0.040	0.003

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\Vert\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\Vert$
			SD(W(Q))	40022	0.041	0.002
			BFGS(B(I))	2279	0.032	0.002
			BFGS(W(I))	2466	0.032	0.000
			BFGS(W(F))	3564	0.033	0.001
			BFGS(W(Q))	2960	0.032	0.001
			L-BFGS(B(I))	8801	0.032	0.001
			L-BFGS(W(I))	7634	0.033	0.001
			L-BFGS(W(F))	6944	0.033	0.000
			L-BFGS(W(Q))	7976	0.032	0.000
			GA	33085	0.498	0.003
			PBIL	97707	0.493	0.000
			GSA	29802	0.281	0.508
			Random	30302	0.500	0.001
		MLP(2,SM)	SD(B(I))	25535	0.004	0.000
			SD(W(I))	33216	0.004	0.001
			SD(W(F))	25042	0.004	0.000
			SD(W(Q))	33460	0.004	0.001
			BFGS(B(I))	146	0.011	0.000
			BFGS(W(I))	174	0.011	0.000
			BFGS(W(F))	230	0.011	0.000
			BFGS(W(Q))	168	0.011	0.000
			L-BFGS(B(I))	287	0.011	0.001
			L-BFGS(W(I))	538	0.013	0.013
			L-BFGS(W(F))	454	0.004	0.000
			L-BFGS(W(Q))	256	0.011	0.001
			GA	51544	0.059	0.000
			PBIL	36702	0.011	0.000
			GSA	153402	0.011	0.001
			Random	46502	0.046	0.010
		MLP(4)	SD(B(I))	45857	0.028	0.004
			SD(W(I))	51352	0.030	0.009
			SD(W(F))	40190	0.030	0.006
			SD(W(Q))	40012	0.032	0.007
			BFGS(B(I))	7458	0.000	0.003
			BFGS(W(I))	9306	0.006	0.000
			BFGS(W(F))	12142	0.006	0.000
			BFGS(W(Q))	7442	0.005	0.005
			L-BFGS(B(I))	32410	0.003	0.000
			L-BFGS(W(I))	42010	0.008	0.000
			L-BFGS(W(F))	43206	0.004	0.004
			L-BFGS(W(Q))	44494	0.002	0.007
			GA	52942	0.499	0.012
			PBIL	171919	0.486	0.001
			GSA	79202	0.281	0.304
			Random	21502	11037.637	3201.286
		MLP(4,SM)	SD(B(I))	25989	0.004	0.000
			SD(W(I))	33902	0.004	0.000
			SD(W(F))	24776	0.004	0.001

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\|\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\|$
			SD(W(Q))	32946	0.004	0.001
			BFGS(B(I))	227	0.007	0.000
			BFGS(W(I))	338	0.007	0.000
			BFGS(W(F))	186	0.007	0.000
			BFGS(W(Q))	180	0.007	0.000
			L-BFGS(B(I))	175	0.011	0.000
			L-BFGS(W(I))	440	0.004	0.001
			L-BFGS(W(F))	322	0.004	0.000
			L-BFGS(W(Q))	232	0.007	0.000
			GA	20206	0.039	0.004
			PBIL	29602	0.014	0.000
			GSA	177902	0.011	0.000
			Random	26002	0.057	0.000
		MLP(8)	SD(B(I))	45856	0.028	0.005
			SD(W(I))	51584	0.029	0.004
			SD(W(F))	40166	0.030	0.006
			SD(W(Q))	40022	0.032	0.007
			BFGS(B(I))	4521	0.000	0.002
			BFGS(W(I))	7332	0.001	0.000
			BFGS(W(F))	10036	0.001	0.000
			BFGS(W(Q))	5784	0.000	0.001
			L-BFGS(B(I))	49203	0.000	0.001
			L-BFGS(W(I))	43792	0.002	0.002
			L-BFGS(W(F))	43224	0.001	0.001
			L-BFGS(W(Q))	44506	0.001	0.018
			GA	28972	21557.391	2676.510
			PBIL	276627	0.488	0.000
			GSA	231402	1.788	2.040
			Random	35602	127779.608	7451.997
		MLP(8,SM)	SD(B(I))	26750	0.004	0.000
			SD(W(I))	34762	0.004	0.001
			SD(W(F))	24552	0.004	0.001
			SD(W(Q))	32462	0.004	0.001
			BFGS(B(I))	145	0.011	0.000
			BFGS(W(I))	172	0.011	0.000
			BFGS(W(F))	192	0.011	0.000
			BFGS(W(Q))	176	0.011	0.000
			L-BFGS(B(I))	131	0.011	0.000
			L-BFGS(W(I))	316	0.011	0.000
			L-BFGS(W(F))	202	0.011	0.000
			L-BFGS(W(Q))	162	0.011	0.000
			GA	50768	0.036	0.000
			PBIL	30002	0.014	0.000
			GSA	424302	0.017	0.036
			Random	31302	0.043	0.000
		RBF(2)	SD(B(I))	75	0.052	0.000
			SD(W(I))	92	0.052	0.000
			SD(W(F))	128	0.052	0.000

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\|\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\|$
			SD(W(Q))	104	0.052	0.000
			BFGS(B(I))	30	0.052	0.000
			BFGS(W(I))	30	0.052	0.000
			BFGS(W(F))	96	0.052	0.000
			BFGS(W(Q))	76	0.052	0.000
			L-BFGS(B(I))	35	0.052	0.000
			L-BFGS(W(I))	34	0.052	0.000
			L-BFGS(W(F))	110	0.052	0.000
			L-BFGS(W(Q))	66	0.052	0.000
			GA	39846	0.718	1.317
			PBIL	17544	0.065	0.082
			GSA	85302	0.053	0.022
			Random	34202	3.691	1.575
		RBF(4)	SD(B(I))	134	0.048	0.000
			SD(W(I))	172	0.048	0.000
			SD(W(F))	156	0.048	0.000
			SD(W(Q))	148	0.048	0.000
			BFGS(B(I))	67	0.048	0.000
			BFGS(W(I))	50	0.048	0.000
			BFGS(W(F))	84	0.048	0.000
			BFGS(W(Q))	82	0.048	0.000
			L-BFGS(B(I))	62	0.048	0.000
			L-BFGS(W(I))	50	0.048	0.000
			L-BFGS(W(F))	106	0.048	0.000
			L-BFGS(W(Q))	74	0.048	0.000
			GA	36250	18.971	5.600
			PBIL	27670	0.097	0.120
			GSA	26802	0.053	0.089
			Random	37402	47.345	8.688
		RBF(8)	SD(B(I))	340	0.035	0.000
			SD(W(I))	442	0.035	0.000
			SD(W(F))	326	0.035	0.001
			SD(W(Q))	176	0.035	0.001
			BFGS(B(I))	94	0.035	0.000
			BFGS(W(I))	110	0.035	0.000
			BFGS(W(F))	128	0.035	0.000
			BFGS(W(Q))	116	0.035	0.000
			L-BFGS(B(I))	86	0.035	0.000
			L-BFGS(W(I))	86	0.035	0.000
			L-BFGS(W(F))	108	0.035	0.000
			L-BFGS(W(Q))	108	0.035	0.000
			GA	28034	68.189	4.986
			PBIL	41139	3.138	0.940
			GSA	65602	0.066	0.122
			Random	41802	81.127	5.003
	MAE	LinReg	SD(B(I))	7464	0.205	0.031
			SD(W(I))	42226	0.197	0.039
			SD(W(F))	56554	0.198	0.025

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\lVert\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\rVert$
			SD(W(Q))	71576	0.199	0.029
			BFGS(B(I))	44585	0.173	0.021
			BFGS(W(I))	385198	0.173	0.018
			BFGS(W(F))	310760	0.173	0.017
			BFGS(W(Q))	791402	0.173	0.021
			L-BFGS(B(I))	60489	0.177	0.021
			L-BFGS(W(I))	44222	0.177	0.014
			L-BFGS(W(F))	43868	0.177	0.020
			L-BFGS(W(Q))	47186	0.177	0.018
			GA	33775	15.187	0.498
			PBIL	220899	6.640	0.071
			GSA	133202	0.459	0.285
			Random	37902	15.240	0.267
		LogReg	SD(B(I))	2008	0.022	0.000
			SD(W(I))	2704	0.022	0.000
			SD(W(F))	1818	0.021	0.001
			SD(W(Q))	2446	0.021	0.001
			BFGS(B(I))	103	0.057	0.000
			BFGS(W(I))	41322	0.038	0.007
			BFGS(W(F))	172	0.038	0.000
			BFGS(W(Q))	55104	0.033	0.007
			L-BFGS(B(I))	79	0.029	0.000
			L-BFGS(W(I))	156	0.022	0.000
			L-BFGS(W(F))	120	0.027	0.000
			L-BFGS(W(Q))	100	0.028	0.001
			GA	31354	0.054	0.003
			PBIL	241761	0.013	0.000
			GSA	605102	0.012	0.000
			Random	32402	0.092	0.003
		MLP(2)	SD(B(I))	16227	0.166	0.025
			SD(W(I))	75118	0.162	0.022
			SD(W(F))	40314	0.163	0.028
			SD(W(Q))	48608	0.163	0.030
			BFGS(B(I))	134606	0.091	0.387
			BFGS(W(I))	129236	0.093	0.039
			BFGS(W(F))	46018	0.093	0.045
			BFGS(W(Q))	74958	0.092	0.056
			L-BFGS(B(I))	111693	0.150	0.049
			L-BFGS(W(I))	50976	0.129	0.053
			L-BFGS(W(F))	44680	0.134	0.053
			L-BFGS(W(Q))	43614	0.135	0.043
			GA	44885	0.497	0.008
			PBIL	40946	0.497	0.007
			GSA	25502	0.448	0.283
			Random	30302	0.500	0.000
		MLP(2,SM)	SD(B(I))	10096	0.011	0.000
			SD(W(I))	10706	0.011	0.000
			SD(W(F))	8892	0.011	0.001

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\|\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\|$
			SD(W(Q))	10934	0.011	0.001
			BFGS(B(I))	20	0.488	0.000
			BFGS(W(I))	188	0.270	0.000
			BFGS(W(F))	180	0.270	0.000
			BFGS(W(Q))	174	0.270	0.000
			L-BFGS(B(I))	88	0.514	0.000
			L-BFGS(W(I))	150	0.261	0.000
			L-BFGS(W(F))	138	0.263	0.001
			L-BFGS(W(Q))	124	0.268	0.000
			GA	38878	0.029	0.082
			PBIL	27302	0.014	0.000
			GSA	136702	0.014	0.000
			Random	46502	0.048	0.005
		MLP(4)	SD(B(I))	28884	0.166	0.039
			SD(W(I))	43520	0.155	0.026
			SD(W(F))	40346	0.155	0.032
			SD(W(Q))	69144	0.155	0.029
			BFGS(B(I))	91648	0.012	0.704
			BFGS(W(I))	50134	0.013	1.086
			BFGS(W(F))	46326	0.018	0.127
			BFGS(W(Q))	46428	0.012	0.858
			L-BFGS(B(I))	121312	0.056	0.060
			L-BFGS(W(I))	59440	0.059	0.027
			L-BFGS(W(F))	69544	0.051	0.064
			L-BFGS(W(Q))	65678	0.047	0.072
			GA	54157	0.500	0.000
			PBIL	165944	0.495	0.008
			GSA	110602	0.244	1.085
			Random	73202	5.640	0.712
		MLP(4,SM)	SD(B(I))	10481	0.011	0.000
			SD(W(I))	13154	0.011	0.000
			SD(W(F))	9338	0.011	0.001
			SD(W(Q))	12362	0.011	0.001
			BFGS(B(I))	155	0.032	0.000
			BFGS(W(I))	198	0.021	0.166
			BFGS(W(F))	154	0.029	0.000
			BFGS(W(Q))	132	0.025	0.000
			L-BFGS(B(I))	73	0.500	0.000
			L-BFGS(W(I))	326	0.014	0.000
			L-BFGS(W(F))	178	0.021	0.000
			L-BFGS(W(Q))	158	0.021	0.000
			GA	25276	0.029	0.000
			PBIL	31202	0.011	0.000
			GSA	188002	0.011	0.000
			Random	26002	0.057	0.000
		MLP(8)	SD(B(I))	17034	0.179	0.034
			SD(W(I))	43706	0.162	0.028
			SD(W(F))	65272	0.167	0.030

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi_c'$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\|\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\|$
			SD(W(Q))	72546	0.163	0.040
			BFGS(B(I))	112061	0.012	0.422
			BFGS(W(I))	48218	0.014	0.262
			BFGS(W(F))	45990	0.018	0.501
			BFGS(W(Q))	46414	0.013	0.943
			L-BFGS(B(I))	112691	0.048	0.043
			L-BFGS(W(I))	66184	0.048	0.039
			L-BFGS(W(F))	58600	0.061	0.081
			L-BFGS(W(Q))	62090	0.050	0.025
			GA	21327	17.764	1.313
			PBIL	223295	0.493	0.011
			GSA	211902	1.330	0.943
			Random	35602	73.474	3.689
		MLP(8,SM)	SD(B(I))	10683	0.011	0.000
			SD(W(I))	13582	0.011	0.001
			SD(W(F))	9856	0.011	0.001
			SD(W(Q))	12914	0.011	0.001
			BFGS(B(I))	565	0.021	0.000
			BFGS(W(I))	196	0.021	0.000
			BFGS(W(F))	148	0.014	0.000
			BFGS(W(Q))	164	0.018	0.000
			L-BFGS(B(I))	287	0.018	0.000
			L-BFGS(W(I))	128	0.014	0.000
			L-BFGS(W(F))	206	0.018	0.000
			L-BFGS(W(Q))	196	0.014	0.000
			GA	22610	0.046	0.000
			PBIL	32202	0.014	0.000
			GSA	227902	0.017	0.030
			Random	31302	0.043	0.000
		RBF(2)	SD(B(I))	1705	0.146	0.005
			SD(W(I))	41536	0.146	0.005
			SD(W(F))	41966	0.146	0.003
			SD(W(Q))	41608	0.146	0.005
			BFGS(B(I))	10957	0.146	0.004
			BFGS(W(I))	196642	0.146	0.003
			BFGS(W(F))	196428	0.146	0.006
			BFGS(W(Q))	152638	0.146	0.005
			L-BFGS(B(I))	2933	0.146	0.005
			L-BFGS(W(I))	41858	0.146	0.003
			L-BFGS(W(F))	42032	0.146	0.005
			L-BFGS(W(Q))	41404	0.146	0.005
			GA	19786	0.978	0.655
			PBIL	22835	0.241	0.298
			GSA	43602	0.152	0.177
			Random	34202	1.656	0.383
		RBF(4)	SD(B(I))	1780	0.112	0.010
			SD(W(I))	42202	0.112	0.004
			SD(W(F))	42042	0.112	0.006

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\|\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\|$
			SD(W(Q))	42538	0.112	0.010
			BFGS(B(I))	8393	0.112	0.009
			BFGS(W(I))	124364	0.112	0.006
			BFGS(W(F))	240636	0.112	0.007
			BFGS(W(Q))	262180	0.112	0.009
			L-BFGS(B(I))	14238	0.112	0.006
			L-BFGS(W(I))	43092	0.112	0.004
			L-BFGS(W(F))	41520	0.112	0.010
			L-BFGS(W(Q))	42410	0.112	0.006
			GA	33428	2.815	0.380
			PBIL	28225	0.916	0.183
			GSA	44602	0.145	0.436
			Random	46902	5.195	0.433
		RBF(8)	SD(B(I))	45888	0.099	0.014
			SD(W(I))	51590	0.099	0.007
			SD(W(F))	54964	0.099	0.007
			SD(W(Q))	57782	0.099	0.017
			BFGS(B(I))	13333	0.097	0.006
			BFGS(W(I))	362012	0.097	0.005
			BFGS(W(F))	252906	0.097	0.005
			BFGS(W(Q))	226222	0.097	0.005
			L-BFGS(B(I))	44763	0.097	0.008
			L-BFGS(W(I))	43626	0.097	0.011
			L-BFGS(W(F))	44324	0.097	0.013
			L-BFGS(W(Q))	44580	0.097	0.009
			GA	61824	4.845	0.242
			PBIL	58379	0.331	0.172
			GSA	35902	0.147	0.158
			Random	44002	6.429	0.236
	HL	LinReg	SD(B(I))	5601	0.094	0.024
			SD(W(I))	56846	0.081	0.017
			SD(W(F))	40118	0.093	0.035
			SD(W(Q))	62762	0.094	0.035
			BFGS(B(I))	2654	0.000	0.000
			BFGS(W(I))	24680	0.000	0.000
			BFGS(W(F))	36638	0.000	0.000
			BFGS(W(Q))	63792	0.000	0.000
			L-BFGS(B(I))	26507	0.001	0.005
			L-BFGS(W(I))	44642	0.006	0.019
			L-BFGS(W(F))	43144	0.016	0.005
			L-BFGS(W(Q))	43664	0.024	0.014
			GA	38037	1.839	0.220
			PBIL	186606	0.468	0.009
			GSA	112402	0.095	0.024
			Random	49002	3.069	0.254
		LogReg	SD(B(I))	4489	0.032	0.000
			SD(W(I))	5798	0.032	0.000
			SD(W(F))	3674	0.032	0.000

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\Vert\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\Vert$
			SD(W(Q))	6098	0.030	0.001
			BFGS(B(I))	10813	0.062	0.012
			BFGS(W(I))	118	0.072	0.000
			BFGS(W(F))	41362	0.071	0.014
			BFGS(W(Q))	130	0.064	0.002
			L-BFGS(B(I))	76	0.057	0.000
			L-BFGS(W(I))	90	0.054	0.000
			L-BFGS(W(F))	308	0.054	0.027
			L-BFGS(W(Q))	140	0.036	0.000
			GA	31354	0.108	0.005
			PBIL	241761	0.026	0.000
			GSA	932502	0.022	0.000
			Random	32402	0.184	0.006
		MLP(2)	SD(B(I))	16759	0.080	0.034
			SD(W(I))	49980	0.076	0.026
			SD(W(F))	52194	0.078	0.040
			SD(W(Q))	54100	0.076	0.039
			BFGS(B(I))	69	0.971	0.000
			BFGS(W(I))	446	0.000	0.000
			BFGS(W(F))	384	0.000	0.000
			BFGS(W(Q))	398	0.000	0.000
			L-BFGS(B(I))	22793	0.033	0.031
			L-BFGS(W(I))	43018	0.054	0.061
			L-BFGS(W(F))	44538	0.017	0.062
			L-BFGS(W(Q))	42838	0.014	0.187
			GA	40125	0.543	0.010
			PBIL	46622	0.511	0.000
			GSA	78202	0.074	0.028
			Random	48902	0.603	0.011
		MLP(2,SM)	SD(B(I))	12329	0.022	0.000
			SD(W(I))	15282	0.022	0.000
			SD(W(F))	11280	0.022	0.001
			SD(W(Q))	14674	0.022	0.001
			BFGS(B(I))	46	1.000	0.000
			BFGS(W(I))	132	0.539	0.000
			BFGS(W(F))	130	0.537	0.001
			BFGS(W(Q))	180	0.536	0.000
			L-BFGS(B(I))	17	1.057	0.000
			L-BFGS(W(I))	150	0.514	0.000
			L-BFGS(W(F))	138	0.529	0.000
			L-BFGS(W(Q))	122	0.521	0.000
			GA	38878	0.058	0.163
			PBIL	27302	0.029	0.000
			GSA	136202	0.029	0.000
			Random	46502	0.097	0.010
		MLP(4)	SD(B(I))	5359	0.081	0.043
			SD(W(I))	46900	0.077	0.047
			SD(W(F))	58252	0.070	0.041

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\Vert\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\Vert$
			SD(W(Q))	47966	0.077	0.048
			BFGS(B(I))	3340	0.000	0.000
			BFGS(W(I))	352	0.000	0.000
			BFGS(W(F))	304	0.000	0.000
			BFGS(W(Q))	386	0.000	0.000
			L-BFGS(B(I))	9050	0.000	0.000
			L-BFGS(W(I))	42192	0.001	0.066
			L-BFGS(W(F))	826	0.000	0.000
			L-BFGS(W(Q))	42992	0.000	0.055
			GA	27383	0.600	0.200
			PBIL	33902	0.514	0.001
			GSA	174102	0.041	0.025
			Random	56702	1.207	0.744
		MLP(4,SM)	SD(B(I))	12494	0.022	0.000
			SD(W(I))	16104	0.022	0.001
			SD(W(F))	11838	0.022	0.001
			SD(W(Q))	15772	0.022	0.001
			BFGS(B(I))	150	0.036	0.000
			BFGS(W(I))	168	0.036	0.000
			BFGS(W(F))	324	1.071	0.000
			BFGS(W(Q))	118	0.036	0.000
			L-BFGS(B(I))	108	0.064	0.000
			L-BFGS(W(I))	196	0.036	0.000
			L-BFGS(W(F))	338	0.971	0.000
			L-BFGS(W(Q))	146	0.036	0.000
			GA	25276	0.057	0.000
			PBIL	31202	0.021	0.000
			GSA	239002	0.021	0.000
			Random	26002	0.114	0.000
		MLP(8)	SD(B(I))	7079	0.086	0.028
			SD(W(I))	78814	0.068	0.035
			SD(W(F))	48464	0.066	0.035
			SD(W(Q))	86138	0.067	0.053
			BFGS(B(I))	319	0.000	0.000
			BFGS(W(I))	42864	0.006	0.010
			BFGS(W(F))	424	0.000	0.000
			BFGS(W(Q))	502	0.000	0.000
			L-BFGS(B(I))	14976	0.000	0.000
			L-BFGS(W(I))	75550	0.000	0.019
			L-BFGS(W(F))	42020	0.009	0.023
			L-BFGS(W(Q))	2414	0.000	0.000
			GA	26236	1.116	0.519
			PBIL	269643	0.117	0.005
			GSA	159702	0.210	0.354
			Random	41302	14.531	3.177
		MLP(8,SM)	SD(B(I))	13026	0.022	0.000
			SD(W(I))	16818	0.022	0.001
			SD(W(F))	11940	0.022	0.001

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\lVert\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\rVert$
			SD(W(Q))	16444	0.022	0.001
			BFGS(B(I))	111	0.029	0.000
			BFGS(W(I))	128	0.036	0.000
			BFGS(W(F))	116	0.036	0.000
			BFGS(W(Q))	254	0.021	0.000
			L-BFGS(B(I))	113	0.036	0.000
			L-BFGS(W(I))	162	0.029	0.000
			L-BFGS(W(F))	160	0.029	0.000
			L-BFGS(W(Q))	202	0.029	0.000
			GA	22610	0.093	0.000
			PBIL	32202	0.029	0.000
			GSA	168002	0.043	0.002
			Random	31302	0.086	0.000
		RBF(2)	SD(B(I))	18051	0.127	0.002
			SD(W(I))	782	0.127	0.005
			SD(W(F))	402	0.127	0.005
			SD(W(Q))	398	0.127	0.005
			BFGS(B(I))	110	0.127	0.001
			BFGS(W(I))	124	0.127	0.006
			BFGS(W(F))	146	0.127	0.004
			BFGS(W(Q))	16870	0.127	0.005
			L-BFGS(B(I))	114	0.127	0.000
			L-BFGS(W(I))	100	0.127	0.004
			L-BFGS(W(F))	308	0.127	0.000
			L-BFGS(W(Q))	340	0.127	0.000
			GA	22706	0.265	0.025
			PBIL	19275	0.134	0.007
			GSA	21702	0.145	0.008
			Random	23802	0.201	0.074
		RBF(4)	SD(B(I))	45879	0.132	0.006
			SD(W(I))	42814	0.130	0.005
			SD(W(F))	45500	0.130	0.005
			SD(W(Q))	45120	0.130	0.006
			BFGS(B(I))	18092	0.118	0.002
			BFGS(W(I))	164754	0.127	0.001
			BFGS(W(F))	126784	0.122	0.001
			BFGS(W(Q))	206488	0.124	0.001
			L-BFGS(B(I))	13370	0.129	0.005
			L-BFGS(W(I))	41348	0.129	0.002
			L-BFGS(W(F))	41580	0.129	0.002
			L-BFGS(W(Q))	41624	0.129	0.006
			GA	27622	0.377	0.113
			PBIL	26788	0.239	0.016
			GSA	22702	0.149	0.037
			Random	39202	0.689	0.203
		RBF(8)	SD(B(I))	45862	0.105	0.010
			SD(W(I))	44082	0.096	0.005
			SD(W(F))	46792	0.096	0.005

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\lVert\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\rVert$
			SD(W(Q))	44162	0.096	0.005
			BFGS(B(I))	71824	0.072	0.005
			BFGS(W(I))	524364	0.074	0.006
			BFGS(W(F))	390266	0.074	0.005
			BFGS(W(Q))	911194	0.074	0.006
			L-BFGS(B(I))	14097	0.076	0.004
			L-BFGS(W(I))	42150	0.074	0.008
			L-BFGS(W(F))	41414	0.091	0.006
			L-BFGS(W(Q))	43712	0.074	0.005
			GA	38481	0.330	0.055
			PBIL	64667	0.146	0.015
			GSA	31402	0.116	0.021
			Random	23002	2.000	0.120
HM	MSE	LinReg	SD(B(I))	198	0.210	0.001
			SD(W(I))	264	0.210	0.000
			SD(W(F))	190	0.210	0.001
			SD(W(Q))	224	0.210	0.000
			BFGS(B(I))	71	0.210	0.000
			BFGS(W(I))	54	0.210	0.000
			BFGS(W(F))	98	0.210	0.000
			BFGS(W(Q))	86	0.210	0.000
			L-BFGS(B(I))	67	0.210	0.000
			L-BFGS(W(I))	66	0.210	0.000
			L-BFGS(W(F))	126	0.210	0.000
			L-BFGS(W(Q))	76	0.210	0.000
			GA	20434	19.189	2.384
			PBIL	29283	0.337	0.200
			GSA	32502	0.220	0.136
			Random	21502	66.399	9.788
		LogReg	SD(B(I))	593	0.199	0.000
			SD(W(I))	778	0.199	0.000
			SD(W(F))	488	0.199	0.002
			SD(W(Q))	690	0.199	0.001
			BFGS(B(I))	108	0.199	0.000
			BFGS(W(I))	134	0.199	0.000
			BFGS(W(F))	200	0.199	0.000
			BFGS(W(Q))	156	0.199	0.000
			L-BFGS(B(I))	68	0.199	0.000
			L-BFGS(W(I))	100	0.199	0.000
			L-BFGS(W(F))	94	0.199	0.001
			L-BFGS(W(Q))	98	0.199	0.000
			GA	22877	0.251	0.007
			PBIL	21400	0.235	0.006
			GSA	26302	0.201	0.007
			Random	21202	0.283	0.016
		MLP(2)	SD(B(I))	45852	0.199	0.001
			SD(W(I))	51244	0.199	0.001
			SD(W(F))	40284	0.199	0.001

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\lVert\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\rVert$
			SD(W(Q))	40054	0.199	0.001
			BFGS(B(I))	3255	0.193	0.002
			BFGS(W(I))	2690	0.194	0.001
			BFGS(W(F))	2628	0.194	0.006
			BFGS(W(Q))	2430	0.194	0.008
			L-BFGS(B(I))	6670	0.195	0.000
			L-BFGS(W(I))	7502	0.195	0.000
			L-BFGS(W(F))	5806	0.196	0.000
			L-BFGS(W(Q))	10030	0.194	0.001
			GA	44882	0.464	0.009
			PBIL	37386	0.480	0.000
			GSA	32502	0.252	0.019
			Random	21302	0.495	0.003
		MLP(2,SM)	SD(B(I))	45852	0.173	0.001
			SD(W(I))	50494	0.173	0.001
			SD(W(F))	40176	0.173	0.000
			SD(W(Q))	40036	0.173	0.001
			BFGS(B(I))	994	0.174	0.000
			BFGS(W(I))	780	0.173	0.000
			BFGS(W(F))	822	0.172	0.000
			BFGS(W(Q))	536	0.174	0.000
			L-BFGS(B(I))	510	0.173	0.000
			L-BFGS(W(I))	1150	0.169	0.000
			L-BFGS(W(F))	602	0.169	0.001
			L-BFGS(W(Q))	794	0.169	0.001
			GA	37436	0.217	0.003
			PBIL	39834	0.183	0.001
			GSA	20702	0.228	0.008
			Random	55902	0.220	0.109
		MLP(4)	SD(B(I))	45856	0.189	0.002
			SD(W(I))	51972	0.189	0.005
			SD(W(F))	40190	0.189	0.002
			SD(W(Q))	40046	0.190	0.003
			BFGS(B(I))	10875	0.159	0.007
			BFGS(W(I))	11500	0.157	0.001
			BFGS(W(F))	8328	0.160	0.003
			BFGS(W(Q))	3820	0.160	0.004
			L-BFGS(B(I))	14653	0.155	0.001
			L-BFGS(W(I))	9386	0.165	0.000
			L-BFGS(W(F))	8888	0.169	0.000
			L-BFGS(W(Q))	24758	0.161	0.001
			GA	42566	0.492	0.002
			PBIL	70048	0.468	0.001
			GSA	54402	0.239	0.022
			Random	25002	0.758	0.797
		MLP(4,SM)	SD(B(I))	45853	0.132	0.001
			SD(W(I))	51384	0.134	0.001
			SD(W(F))	40136	0.134	0.002

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\Vert\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\Vert$
			SD(W(Q))	40046	0.136	0.002
			BFGS(B(I))	517	0.167	0.000
			BFGS(W(I))	770	0.176	0.000
			BFGS(W(F))	1856	0.157	0.001
			BFGS(W(Q))	638	0.172	0.002
			L-BFGS(B(I))	688	0.158	0.000
			L-BFGS(W(I))	1544	0.121	0.000
			L-BFGS(W(F))	796	0.158	0.000
			L-BFGS(W(Q))	1050	0.145	0.000
			GA	38516	0.221	0.001
			PBIL	29623	0.190	0.001
			GSA	33602	0.197	0.017
			Random	32202	0.229	0.093
		MLP(8)	SD(B(I))	45855	0.187	0.004
			SD(W(I))	51852	0.187	0.006
			SD(W(F))	40114	0.187	0.003
			SD(W(Q))	40024	0.188	0.003
			BFGS(B(I))	20661	0.090	0.004
			BFGS(W(I))	19636	0.113	0.002
			BFGS(W(F))	8440	0.121	0.002
			BFGS(W(Q))	14116	0.112	0.003
			L-BFGS(B(I))	48030	0.112	0.003
			L-BFGS(W(I))	20510	0.124	0.001
			L-BFGS(W(F))	24170	0.119	0.000
			L-BFGS(W(Q))	33958	0.118	0.001
			GA	41198	0.622	0.288
			PBIL	176011	0.465	0.001
			GSA	63902	0.238	0.025
			Random	22002	13783.255	2139.225
		MLP(8,SM)	SD(B(I))	45851	0.129	0.001
			SD(W(I))	51788	0.131	0.001
			SD(W(F))	40124	0.132	0.002
			SD(W(Q))	40060	0.134	0.002
			BFGS(B(I))	434	0.150	0.000
			BFGS(W(I))	920	0.148	0.001
			BFGS(W(F))	1614	0.147	0.000
			BFGS(W(Q))	844	0.140	0.001
			L-BFGS(B(I))	16850	0.117	0.001
			L-BFGS(W(I))	7558	0.137	0.000
			L-BFGS(W(F))	5696	0.137	0.000
			L-BFGS(W(Q))	824	0.130	0.000
			GA	31971	0.220	0.000
			PBIL	33302	0.160	0.000
			GSA	96602	0.214	0.043
			Random	21202	0.260	0.000
		RBF(2)	SD(B(I))	239	0.249	0.000
			SD(W(I))	296	0.249	0.000
			SD(W(F))	258	0.249	0.001

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi_c'$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\lVert\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\rVert$
			SD(W(Q))	226	0.249	0.001
			BFGS(B(I))	53	0.249	0.000
			BFGS(W(I))	28	0.249	0.000
			BFGS(W(F))	106	0.249	0.000
			BFGS(W(Q))	96	0.249	0.000
			L-BFGS(B(I))	53	0.249	0.000
			L-BFGS(W(I))	30	0.249	0.000
			L-BFGS(W(F))	92	0.249	0.000
			L-BFGS(W(Q))	110	0.249	0.000
			GA	69110	0.353	0.432
			PBIL	18736	0.256	0.025
			GSA	36102	0.251	0.072
			Random	34202	1.246	0.924
		RBF(4)	SD(B(I))	496	0.215	0.000
			SD(W(I))	674	0.215	0.000
			SD(W(F))	496	0.215	0.001
			SD(W(Q))	546	0.215	0.001
			BFGS(B(I))	76	0.215	0.000
			BFGS(W(I))	86	0.215	0.000
			BFGS(W(F))	96	0.215	0.000
			BFGS(W(Q))	116	0.215	0.000
			L-BFGS(B(I))	106	0.215	0.000
			L-BFGS(W(I))	88	0.215	0.000
			L-BFGS(W(F))	110	0.215	0.000
			L-BFGS(W(Q))	96	0.215	0.002
			GA	38605	5.091	1.396
			PBIL	27237	1.406	0.260
			GSA	32302	0.226	0.083
			Random	23602	19.223	4.232
		RBF(8)	SD(B(I))	564	0.213	0.001
			SD(W(I))	728	0.213	0.000
			SD(W(F))	458	0.213	0.001
			SD(W(Q))	672	0.213	0.001
			BFGS(B(I))	90	0.213	0.000
			BFGS(W(I))	106	0.213	0.000
			BFGS(W(F))	132	0.213	0.000
			BFGS(W(Q))	132	0.213	0.000
			L-BFGS(B(I))	111	0.213	0.000
			L-BFGS(W(I))	100	0.213	0.000
			L-BFGS(W(F))	138	0.213	0.000
			L-BFGS(W(Q))	122	0.213	0.000
			GA	25073	29.368	4.665
			PBIL	62656	0.713	0.094
			GSA	35202	0.234	0.054
			Random	41802	40.402	2.044
	MAE	LinReg	SD(B(I))	1530	0.390	0.014
			SD(W(I))	41854	0.390	0.014
			SD(W(F))	41988	0.390	0.010

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\lVert\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\rVert$
			SD(W(Q))	41502	0.389	0.014
			BFGS(B(I))	12887	0.389	0.018
			BFGS(W(I))	213018	0.389	0.015
			BFGS(W(F))	448984	0.389	0.016
			BFGS(W(Q))	329734	0.389	0.012
			L-BFGS(B(I))	2291	0.389	0.018
			L-BFGS(W(I))	41308	0.389	0.018
			L-BFGS(W(F))	41346	0.389	0.025
			L-BFGS(W(Q))	41406	0.389	0.016
			GA	38012	3.867	0.714
			PBIL	29839	1.678	0.125
			GSA	58902	0.396	0.050
			Random	21502	6.244	0.710
		LogReg	SD(B(I))	1617	0.261	0.000
			SD(W(I))	2092	0.261	0.001
			SD(W(F))	1298	0.260	0.001
			SD(W(Q))	2284	0.258	0.001
			BFGS(B(I))	10161	0.371	0.009
			BFGS(W(I))	55858	0.280	0.019
			BFGS(W(F))	316	0.280	0.133
			BFGS(W(Q))	324	0.280	0.024
			L-BFGS(B(I))	62	0.230	0.000
			L-BFGS(W(I))	86	0.231	0.000
			L-BFGS(W(F))	122	0.232	0.000
			L-BFGS(W(Q))	104	0.230	0.000
			GA	25762	0.273	0.001
			PBIL	34785	0.252	0.000
			GSA	225502	0.250	0.001
			Random	28702	0.317	0.009
		MLP(2)	SD(B(I))	2096	0.390	0.019
			SD(W(I))	42580	0.388	0.025
			SD(W(F))	42710	0.390	0.018
			SD(W(Q))	42976	0.390	0.024
			BFGS(B(I))	136219	0.329	0.367
			BFGS(W(I))	104384	0.334	0.051
			BFGS(W(F))	41930	0.353	0.013
			BFGS(W(Q))	127958	0.332	0.059
			L-BFGS(B(I))	59683	0.500	0.000
			L-BFGS(W(I))	42058	0.359	0.050
			L-BFGS(W(F))	44330	0.358	0.067
			L-BFGS(W(Q))	42192	0.363	0.043
			GA	32049	0.493	0.005
			PBIL	30785	0.432	0.093
			GSA	37102	0.463	0.019
			Random	41002	0.497	0.012
		MLP(2,SM)	SD(B(I))	37373	0.231	0.000
			SD(W(I))	106	0.280	0.000
			SD(W(F))	33352	0.231	0.001

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\lVert\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\rVert$
			SD(W(Q))	40064	0.232	0.001
			BFGS(B(I))	227	0.490	0.000
			BFGS(W(I))	202	0.230	0.000
			BFGS(W(F))	254	0.270	0.000
			BFGS(W(Q))	206	0.230	0.000
			L-BFGS(B(I))	16801	0.346	0.209
			L-BFGS(W(I))	92	0.270	0.000
			L-BFGS(W(F))	206	0.270	0.000
			L-BFGS(W(Q))	174	0.270	0.000
			GA	32456	0.238	0.006
			PBIL	25388	0.230	0.000
			GSA	73502	0.226	0.017
			Random	45802	0.240	0.001
		MLP(4)	SD(B(I))	2407	0.392	0.013
			SD(W(I))	41790	0.387	0.014
			SD(W(F))	42292	0.389	0.016
			SD(W(Q))	41904	0.389	0.028
			BFGS(B(I))	70865	0.258	0.140
			BFGS(W(I))	511974	0.272	0.085
			BFGS(W(F))	216292	0.257	0.161
			BFGS(W(Q))	73062	0.282	0.060
			L-BFGS(B(I))	38262	0.355	0.055
			L-BFGS(W(I))	42890	0.357	0.028
			L-BFGS(W(F))	42282	0.357	0.033
			L-BFGS(W(Q))	42144	0.361	0.037
			GA	42577	0.496	0.007
			PBIL	39794	0.483	0.011
			GSA	60502	0.421	0.033
			Random	25002	0.564	0.104
		MLP(4,SM)	SD(B(I))	37836	0.231	0.000
			SD(W(I))	47856	0.231	0.000
			SD(W(F))	38418	0.231	0.001
			SD(W(Q))	40064	0.232	0.001
			BFGS(B(I))	51	0.500	0.000
			BFGS(W(I))	202	0.385	0.000
			BFGS(W(F))	206	0.385	0.000
			BFGS(W(Q))	166	0.270	0.000
			L-BFGS(B(I))	40	0.500	0.000
			L-BFGS(W(I))	176	0.270	0.001
			L-BFGS(W(F))	180	0.260	0.000
			L-BFGS(W(Q))	142	0.264	0.055
			GA	29936	0.230	0.000
			PBIL	27198	0.230	0.000
			GSA	79002	0.225	0.033
			Random	28402	0.260	0.000
		MLP(8)	SD(B(I))	8057	0.391	0.025
			SD(W(I))	43652	0.389	0.027
			SD(W(F))	53626	0.391	0.023

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi_c'$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\|\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\|$
			SD(W(Q))	42590	0.390	0.015
			BFGS(B(I))	83899	0.201	0.504
			BFGS(W(I))	170760	0.244	0.355
			BFGS(W(F))	333784	0.169	0.855
			BFGS(W(Q))	301838	0.217	0.787
			L-BFGS(B(I))	240272	0.342	0.034
			L-BFGS(W(I))	43324	0.348	0.043
			L-BFGS(W(F))	42526	0.348	0.015
			L-BFGS(W(Q))	42316	0.348	0.066
			GA	37840	0.526	0.040
			PBIL	91348	0.476	0.015
			GSA	52102	0.406	0.030
			Random	22002	29.629	3.460
		MLP(8,SM)	SD(B(I))	37969	0.231	0.000
			SD(W(I))	47842	0.231	0.001
			SD(W(F))	35768	0.231	0.001
			SD(W(Q))	40060	0.232	0.001
			BFGS(B(I))	109	0.280	0.000
			BFGS(W(I))	206	0.280	0.000
			BFGS(W(F))	156	0.280	0.000
			BFGS(W(Q))	160	0.280	0.000
			L-BFGS(B(I))	28	0.480	0.000
			L-BFGS(W(I))	232	0.230	0.000
			L-BFGS(W(F))	166	0.230	0.000
			L-BFGS(W(Q))	148	0.230	0.000
			GA	40650	0.230	0.008
			PBIL	27502	0.220	0.016
			GSA	79402	0.217	0.013
			Random	21202	0.260	0.000
		RBF(2)	SD(B(I))	3203	0.494	0.018
			SD(W(I))	538	0.480	0.000
			SD(W(F))	568	0.480	0.000
			SD(W(Q))	354	0.480	0.000
			BFGS(B(I))	298	0.480	0.000
			BFGS(W(I))	366	0.480	0.000
			BFGS(W(F))	332	0.480	0.000
			BFGS(W(Q))	304	0.480	0.000
			L-BFGS(B(I))	150	0.480	0.000
			L-BFGS(W(I))	164	0.480	0.013
			L-BFGS(W(F))	302	0.480	0.000
			L-BFGS(W(Q))	294	0.480	0.000
			GA	31857	0.772	0.616
			PBIL	24516	0.532	0.102
			GSA	49502	0.484	0.063
			Random	28002	0.884	0.177
		RBF(4)	SD(B(I))	8064	0.401	0.011
			SD(W(I))	43332	0.397	0.002
			SD(W(F))	44416	0.401	0.014

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\Vert\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\Vert$
			SD(W(Q))	43642	0.401	0.015
			BFGS(B(I))	13861	0.397	0.012
			BFGS(W(I))	290742	0.397	0.012
			BFGS(W(F))	371404	0.397	0.009
			BFGS(W(Q))	446184	0.397	0.014
			L-BFGS(B(I))	13419	0.397	0.012
			L-BFGS(W(I))	41640	0.397	0.002
			L-BFGS(W(F))	41612	0.397	0.012
			L-BFGS(W(Q))	41370	0.397	0.002
			GA	33898	1.934	0.335
			PBIL	32722	0.778	0.057
			GSA	51802	0.411	0.077
			Random	31102	3.213	0.237
		RBF(8)	SD(B(I))	27060	0.396	0.019
			SD(W(I))	43658	0.388	0.021
			SD(W(F))	48988	0.389	0.020
			SD(W(Q))	75402	0.390	0.018
			BFGS(B(I))	29374	0.380	0.013
			BFGS(W(I))	289832	0.380	0.010
			BFGS(W(F))	274294	0.380	0.020
			BFGS(W(Q))	403288	0.380	0.017
			L-BFGS(B(I))	8875	0.381	0.031
			L-BFGS(W(I))	41702	0.381	0.015
			L-BFGS(W(F))	41774	0.381	0.025
			L-BFGS(W(Q))	41792	0.381	0.026
			GA	31624	3.964	0.173
			PBIL	52166	1.032	0.061
			GSA	44802	0.396	0.120
			Random	31402	4.891	0.362
	HL	LinReg	SD(B(I))	1875	0.710	0.017
			SD(W(I))	46982	0.707	0.008
			SD(W(F))	44128	0.707	0.012
			SD(W(Q))	42006	0.707	0.016
			BFGS(B(I))	8038	0.707	0.030
			BFGS(W(I))	148994	0.707	0.004
			BFGS(W(F))	76060	0.707	0.006
			BFGS(W(Q))	409794	0.707	0.006
			L-BFGS(B(I))	16693	0.707	0.014
			L-BFGS(W(I))	41540	0.707	0.004
			L-BFGS(W(F))	41312	0.707	0.013
			L-BFGS(W(Q))	41398	0.707	0.013
			GA	42964	3.742	0.501
			PBIL	29061	1.977	0.110
			GSA	27202	0.733	0.167
			Random	31402	5.550	0.278
		LogReg	SD(B(I))	3791	0.501	0.000
			SD(W(I))	4766	0.501	0.000
			SD(W(F))	2628	0.501	0.001

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi_c'$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\|\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\|$
			SD(W(Q))	5330	0.496	0.001
			BFGS(B(I))	10206	0.741	0.020
			BFGS(W(I))	166	0.560	0.000
			BFGS(W(F))	122	0.560	0.000
			BFGS(W(Q))	536	0.560	0.048
			L-BFGS(B(I))	65	0.460	0.000
			L-BFGS(W(I))	102	0.460	0.000
			L-BFGS(W(F))	41288	0.460	0.140
			L-BFGS(W(Q))	98	0.460	0.000
			GA	25762	0.546	0.003
			PBIL	34785	0.504	0.001
			GSA	239402	0.501	0.002
			Random	28702	0.634	0.017
		MLP(2)	SD(B(I))	2172	0.711	0.046
			SD(W(I))	43836	0.708	0.025
			SD(W(F))	47478	0.706	0.022
			SD(W(Q))	46286	0.710	0.025
			BFGS(B(I))	293179	0.637	0.070
			BFGS(W(I))	163346	0.628	0.077
			BFGS(W(F))	98946	0.648	0.015
			BFGS(W(Q))	99638	0.645	0.044
			L-BFGS(B(I))	2122	1.000	0.000
			L-BFGS(W(I))	42014	0.668	0.059
			L-BFGS(W(F))	44902	0.666	0.047
			L-BFGS(W(Q))	41712	0.669	0.063
			GA	33292	0.938	0.011
			PBIL	26213	0.903	0.003
			GSA	40102	0.884	0.013
			Random	30302	0.946	0.010
		MLP(2,SM)	SD(B(I))	45853	0.462	0.001
			SD(W(I))	51714	0.524	0.002
			SD(W(F))	40246	0.462	0.001
			SD(W(Q))	40080	0.463	0.002
			BFGS(B(I))	433	0.500	0.000
			BFGS(W(I))	196	0.460	0.000
			BFGS(W(F))	426	0.680	0.000
			BFGS(W(Q))	178	0.540	0.000
			L-BFGS(B(I))	93	0.540	0.000
			L-BFGS(W(I))	310	0.540	0.000
			L-BFGS(W(F))	392	0.710	0.000
			L-BFGS(W(Q))	150	0.540	0.000
			GA	32456	0.477	0.013
			PBIL	25388	0.460	0.000
			GSA	96202	0.452	0.011
			Random	45802	0.480	0.001
		MLP(4)	SD(B(I))	3642	0.713	0.041
			SD(W(I))	50756	0.704	0.035
			SD(W(F))	43856	0.706	0.044

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi_c'$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\|\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\|$
			SD(W(Q))	44654	0.707	0.038
			BFGS(B(I))	176986	0.480	0.221
			BFGS(W(I))	79806	0.484	0.231
			BFGS(W(F))	179152	0.502	0.186
			BFGS(W(Q))	197010	0.496	0.092
			L-BFGS(B(I))	45963	0.577	0.028
			L-BFGS(W(I))	43760	0.582	0.069
			L-BFGS(W(F))	42682	0.566	0.065
			L-BFGS(W(Q))	43650	0.573	0.090
			GA	58324	0.939	0.002
			PBIL	71214	0.838	0.009
			GSA	56602	0.789	0.029
			Random	31602	0.980	0.000
		MLP(4,SM)	SD(B(I))	45852	0.462	0.001
			SD(W(I))	51956	0.462	0.001
			SD(W(F))	40212	0.462	0.002
			SD(W(Q))	40038	0.463	0.002
			BFGS(B(I))	373	0.980	0.000
			BFGS(W(I))	246	0.560	0.000
			BFGS(W(F))	158	0.770	0.000
			BFGS(W(Q))	160	0.540	0.000
			L-BFGS(B(I))	40	1.000	0.000
			L-BFGS(W(I))	370	0.520	0.000
			L-BFGS(W(F))	198	0.520	0.001
			L-BFGS(W(Q))	146	0.500	0.000
			GA	29936	0.460	0.000
			PBIL	27198	0.460	0.000
			GSA	195402	0.420	0.002
			Random	28402	0.520	0.000
		MLP(8)	SD(B(I))	19048	0.706	0.031
			SD(W(I))	45792	0.703	0.012
			SD(W(F))	47290	0.707	0.045
			SD(W(Q))	43338	0.706	0.027
			BFGS(B(I))	181344	0.310	0.318
			BFGS(W(I))	80998	0.371	0.264
			BFGS(W(F))	709216	0.360	0.197
			BFGS(W(Q))	386400	0.375	0.043
			L-BFGS(B(I))	115594	0.485	0.136
			L-BFGS(W(I))	43770	0.584	0.055
			L-BFGS(W(F))	47248	0.608	0.157
			L-BFGS(W(Q))	41842	0.653	0.049
			GA	20980	0.990	0.000
			PBIL	123392	0.815	0.014
			GSA	41902	0.713	0.106
			Random	22002	32.680	4.360
		MLP(8,SM)	SD(B(I))	45852	0.462	0.001
			SD(W(I))	52096	0.462	0.001
			SD(W(F))	40208	0.462	0.001

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\|\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\|$
			SD(W(Q))	40074	0.463	0.001
			BFGS(B(I))	103	0.540	0.000
			BFGS(W(I))	312	0.460	0.000
			BFGS(W(F))	204	0.460	0.000
			BFGS(W(Q))	212	0.540	0.000
			L-BFGS(B(I))	295	0.460	0.000
			L-BFGS(W(I))	186	0.460	0.000
			L-BFGS(W(F))	242	0.460	0.000
			L-BFGS(W(Q))	156	0.460	0.000
			GA	40650	0.460	0.015
			PBIL	27502	0.440	0.032
			GSA	56302	0.467	0.065
			Random	21202	0.520	0.000
		RBF(2)	SD(B(I))	1428	0.961	0.000
			SD(W(I))	892	0.961	0.000
			SD(W(F))	826	0.961	0.000
			SD(W(Q))	968	0.961	0.013
			BFGS(B(I))	247	0.961	0.000
			BFGS(W(I))	132	0.961	0.013
			BFGS(W(F))	266	0.961	0.000
			BFGS(W(Q))	288	0.961	0.013
			L-BFGS(B(I))	115	0.961	0.000
			L-BFGS(W(I))	308	0.961	0.000
			L-BFGS(W(F))	268	0.961	0.013
			L-BFGS(W(Q))	290	0.961	0.000
			GA	35464	1.296	0.579
			PBIL	18033	0.967	0.014
			GSA	58002	0.962	0.038
			Random	28002	1.333	0.173
		RBF(4)	SD(B(I))	27162	0.737	0.018
			SD(W(I))	44622	0.730	0.013
			SD(W(F))	44306	0.730	0.013
			SD(W(Q))	47582	0.733	0.019
			BFGS(B(I))	9648	0.729	0.016
			BFGS(W(I))	343972	0.729	0.012
			BFGS(W(F))	440138	0.729	0.016
			BFGS(W(Q))	306772	0.729	0.012
			L-BFGS(B(I))	16158	0.729	0.012
			L-BFGS(W(I))	41426	0.729	0.012
			L-BFGS(W(F))	41418	0.729	0.013
			L-BFGS(W(Q))	41418	0.729	0.012
			GA	20756	2.284	0.416
			PBIL	38122	1.518	0.041
			GSA	34502	0.737	0.073
			Random	48702	2.708	0.211
		RBF(8)	SD(B(I))	3362	0.735	0.020
			SD(W(I))	42770	0.727	0.015
			SD(W(F))	77238	0.727	0.018

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\lVert\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\rVert$
			SD(W(Q))	44014	0.731	0.028
			BFGS(B(I))	15503	0.721	0.007
			BFGS(W(I))	313004	0.721	0.012
			BFGS(W(F))	421000	0.721	0.013
			BFGS(W(Q))	403368	0.721	0.016
			L-BFGS(B(I))	15198	0.721	0.005
			L-BFGS(W(I))	41678	0.722	0.017
			L-BFGS(W(F))	41736	0.721	0.017
			L-BFGS(W(Q))	41582	0.721	0.013
			GA	23808	3.403	0.581
			PBIL	50301	1.403	0.047
			GSA	68902	0.708	0.038
			Random	37402	4.463	0.140
Yeast	MSE	LinReg	SD(B(I))	1141	0.053	0.001
			SD(W(I))	1518	0.053	0.000
			SD(W(F))	946	0.053	0.001
			SD(W(Q))	1412	0.053	0.001
			BFGS(B(I))	163	0.053	0.000
			BFGS(W(I))	202	0.053	0.000
			BFGS(W(F))	252	0.053	0.000
			BFGS(W(Q))	240	0.053	0.000
			L-BFGS(B(I))	157	0.053	0.000
			L-BFGS(W(I))	182	0.053	0.000
			L-BFGS(W(F))	186	0.052	0.001
			L-BFGS(W(Q))	172	0.052	0.000
			GA	21315	834.938	25.645
			PBIL	256736	76.966	1.951
			GSA	104202	0.240	0.120
			Random	27402	728.984	15.112
		LogReg	SD(B(I))	873	0.040	0.000
			SD(W(I))	1248	0.040	0.001
			SD(W(F))	916	0.040	0.001
			SD(W(Q))	1428	0.038	0.001
			BFGS(B(I))	608	0.040	0.000
			BFGS(W(I))	1606	0.030	0.000
			BFGS(W(F))	2146	0.035	0.000
			BFGS(W(Q))	1648	0.032	0.000
			L-BFGS(B(I))	342	0.025	0.000
			L-BFGS(W(I))	216	0.032	0.001
			L-BFGS(W(F))	196	0.031	0.000
			L-BFGS(W(Q))	272	0.030	0.001
			GA	36039	0.090	0.002
			PBIL	203254	0.073	0.000
			GSA	347402	0.061	0.002
			Random	22102	0.122	0.002
		MLP(2)	SD(B(I))	421	0.080	0.000
			SD(W(I))	618	0.080	0.000
			SD(W(F))	376	0.080	0.001

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\|\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\|$
			SD(W(Q))	534	0.080	0.000
			BFGS(B(I))	223	0.080	0.000
			BFGS(W(I))	238	0.080	0.000
			BFGS(W(F))	332	0.080	0.000
			BFGS(W(Q))	268	0.080	0.000
			L-BFGS(B(I))	113	0.080	0.000
			L-BFGS(W(I))	124	0.080	0.000
			L-BFGS(W(F))	158	0.080	0.000
			L-BFGS(W(Q))	154	0.080	0.000
			GA	46450	0.099	0.001
			PBIL	118838	0.094	0.000
			GSA	91502	0.090	0.001
			Random	50502	0.100	0.001
		MLP(2,SM)	SD(B(I))	45	0.081	0.005
			SD(W(I))	72	0.081	0.000
			SD(W(F))	76	0.081	0.001
			SD(W(Q))	58	0.081	0.001
			BFGS(B(I))	56	0.081	0.000
			BFGS(W(I))	110	0.081	0.000
			BFGS(W(F))	174	0.081	0.001
			BFGS(W(Q))	140	0.081	0.000
			L-BFGS(B(I))	39	0.081	0.000
			L-BFGS(W(I))	70	0.081	0.000
			L-BFGS(W(F))	70	0.081	0.000
			L-BFGS(W(Q))	66	0.081	0.000
			GA	47829	0.074	0.000
			PBIL	130179	0.072	0.000
			GSA	33002	0.082	0.001
			Random	49902	0.076	0.041
		MLP(4)	SD(B(I))	1972	0.065	0.000
			SD(W(I))	2782	0.065	0.000
			SD(W(F))	1666	0.065	0.000
			SD(W(Q))	2710	0.065	0.001
			BFGS(B(I))	442	0.065	0.000
			BFGS(W(I))	694	0.065	0.000
			BFGS(W(F))	482	0.065	0.000
			BFGS(W(Q))	486	0.065	0.000
			L-BFGS(B(I))	654	0.065	0.000
			L-BFGS(W(I))	510	0.065	0.001
			L-BFGS(W(F))	662	0.065	0.000
			L-BFGS(W(Q))	664	0.065	0.001
			GA	22289	0.099	0.001
			PBIL	195407	0.090	0.001
			GSA	49702	0.090	0.001
			Random	51902	0.100	0.000
		MLP(4,SM)	SD(B(I))	44498	0.014	0.000
			SD(W(I))	20744	0.021	0.001
			SD(W(F))	28952	0.019	0.001

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\lVert\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\rVert$
			SD(W(Q))	40088	0.016	0.000
			BFGS(B(I))	108	0.074	0.000
			BFGS(W(I))	466	0.067	0.000
			BFGS(W(F))	260	0.074	0.000
			BFGS(W(Q))	194	0.074	0.000
			L-BFGS(B(I))	1278	0.015	0.000
			L-BFGS(W(I))	1874	0.025	0.000
			L-BFGS(W(F))	164	0.066	0.000
			L-BFGS(W(Q))	520	0.048	0.001
			GA	36341	0.073	0.002
			PBIL	26202	0.070	0.000
			GSA	99702	0.085	0.012
			Random	27102	0.079	0.003
		MLP(8)	SD(B(I))	45853	0.045	0.001
			SD(W(I))	51576	0.045	0.001
			SD(W(F))	40246	0.046	0.001
			SD(W(Q))	40028	0.047	0.001
			BFGS(B(I))	7690	0.036	0.001
			BFGS(W(I))	7002	0.036	0.001
			BFGS(W(F))	14460	0.037	0.000
			BFGS(W(Q))	8348	0.036	0.000
			L-BFGS(B(I))	11724	0.035	0.000
			L-BFGS(W(I))	14004	0.036	0.001
			L-BFGS(W(F))	10146	0.038	0.001
			L-BFGS(W(Q))	12696	0.037	0.001
			GA	30159	923.649	476.767
			PBIL	370868	0.089	0.002
			GSA	114702	0.082	0.003
			Random	23202	134795.577	12433.600
		MLP(8,SM)	SD(B(I))	19326	0.019	0.000
			SD(W(I))	40568	0.026	0.001
			SD(W(F))	16914	0.019	0.001
			SD(W(Q))	32894	0.018	0.001
			BFGS(B(I))	1437	0.025	0.000
			BFGS(W(I))	1612	0.022	0.000
			BFGS(W(F))	3206	0.027	0.000
			BFGS(W(Q))	1924	0.018	0.000
			L-BFGS(B(I))	1341	0.028	0.000
			L-BFGS(W(I))	2684	0.027	0.000
			L-BFGS(W(F))	2058	0.044	0.000
			L-BFGS(W(Q))	1094	0.019	0.000
			GA	48043	0.078	0.000
			PBIL	360421	0.058	0.003
			GSA	53202	0.107	0.033
			Random	32102	0.104	0.020
		RBF(2)	SD(B(I))	132	0.080	0.000
			SD(W(I))	168	0.080	0.000
			SD(W(F))	146	0.080	0.000

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi_c'$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\lVert\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\rVert$
			SD(W(Q))	138	0.080	0.000
			BFGS(B(I))	53	0.080	0.000
			BFGS(W(I))	38	0.080	0.000
			BFGS(W(F))	84	0.080	0.000
			BFGS(W(Q))	72	0.080	0.000
			L-BFGS(B(I))	53	0.080	0.000
			L-BFGS(W(I))	32	0.080	0.000
			L-BFGS(W(F))	70	0.080	0.000
			L-BFGS(W(Q))	88	0.080	0.000
			GA	20234	159.068	8.863
			PBIL	89979	0.297	0.109
			GSA	70802	0.090	0.055
			Random	28302	187.595	8.775
		RBF(4)	SD(B(I))	130	0.065	0.001
			SD(W(I))	204	0.065	0.000
			SD(W(F))	170	0.065	0.000
			SD(W(Q))	132	0.065	0.000
			BFGS(B(I))	85	0.065	0.000
			BFGS(W(I))	86	0.065	0.000
			BFGS(W(F))	128	0.065	0.000
			BFGS(W(Q))	100	0.065	0.000
			L-BFGS(B(I))	53	0.065	0.000
			L-BFGS(W(I))	58	0.065	0.000
			L-BFGS(W(F))	82	0.065	0.000
			L-BFGS(W(Q))	84	0.065	0.000
			GA	51100	205.560	7.571
			PBIL	133224	10.652	0.633
			GSA	49202	0.094	0.078
			Random	31902	248.449	8.451
		RBF(8)	SD(B(I))	1186	0.052	0.000
			SD(W(I))	1600	0.052	0.000
			SD(W(F))	984	0.052	0.001
			SD(W(Q))	1460	0.051	0.001
			BFGS(B(I))	163	0.051	0.000
			BFGS(W(I))	200	0.051	0.000
			BFGS(W(F))	338	0.051	0.000
			BFGS(W(Q))	250	0.051	0.000
			L-BFGS(B(I))	139	0.051	0.000
			L-BFGS(W(I))	202	0.051	0.000
			L-BFGS(W(F))	162	0.051	0.000
			L-BFGS(W(Q))	174	0.051	0.000
			GA	23491	200.741	5.814
			PBIL	217390	13.010	0.690
			GSA	134002	0.132	0.048
			Random	50102	239.630	6.221
	MAE	LinReg	SD(B(I))	45895	0.098	0.041
			SD(W(I))	41498	0.093	0.049
			SD(W(F))	40064	0.093	0.047

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi_c'$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\|\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\|$
			SD(W(Q))	40006	0.094	0.045
			BFGS(B(I))	60614	0.082	0.216
			BFGS(W(I))	133368	0.082	0.207
			BFGS(W(F))	190646	0.082	0.127
			BFGS(W(Q))	183008	0.082	0.139
			L-BFGS(B(I))	49409	0.083	0.233
			L-BFGS(W(I))	42512	0.083	0.176
			L-BFGS(W(F))	44910	0.083	0.071
			L-BFGS(W(Q))	56614	0.083	0.070
			GA	19772	20.230	0.364
			PBIL	251181	7.708	0.081
			GSA	131702	0.274	0.127
			Random	27402	20.764	0.318
		LogReg	SD(B(I))	35	0.082	0.000
			SD(W(I))	170	0.083	0.000
			SD(W(F))	50	0.082	0.000
			SD(W(Q))	46	0.082	0.000
			BFGS(B(I))	11	0.900	0.000
			BFGS(W(I))	66	0.100	0.000
			BFGS(W(F))	234	0.900	0.503
			BFGS(W(Q))	56	0.100	0.001
			L-BFGS(B(I))	11	0.900	0.000
			L-BFGS(W(I))	74	0.082	0.000
			L-BFGS(W(F))	36	0.082	0.000
			L-BFGS(W(Q))	48	0.082	0.000
			GA	30345	0.094	0.002
			PBIL	83666	0.077	0.000
			GSA	230802	0.075	0.000
			Random	22102	0.125	0.003
		MLP(2)	SD(B(I))	18961	0.104	0.063
			SD(W(I))	56374	0.102	0.018
			SD(W(F))	57652	0.102	0.018
			SD(W(Q))	59768	0.102	0.039
			BFGS(B(I))	17894	0.100	0.000
			BFGS(W(I))	188230	0.100	0.004
			BFGS(W(F))	61044	0.100	0.101
			BFGS(W(Q))	85392	0.100	0.029
			L-BFGS(B(I))	80840	0.100	0.000
			L-BFGS(W(I))	46336	0.100	0.015
			L-BFGS(W(F))	43566	0.100	0.015
			L-BFGS(W(Q))	42824	0.100	0.019
			GA	19121	0.100	0.000
			PBIL	20802	0.100	0.000
			GSA	32502	0.100	0.000
			Random	23202	0.100	0.000
		MLP(2,SM)	SD(B(I))	42	0.162	0.000
			SD(W(I))	58	0.162	0.000
			SD(W(F))	442	0.146	0.000

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi_c'$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\|\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\|$
			SD(W(Q))	22422	0.095	0.001
			BFGS(B(I))	21	0.173	0.000
			BFGS(W(I))	110	0.162	0.000
			BFGS(W(F))	134	0.162	0.000
			BFGS(W(Q))	124	0.162	0.000
			L-BFGS(B(I))	122	0.198	0.000
			L-BFGS(W(I))	302	0.145	0.000
			L-BFGS(W(F))	272	0.147	0.000
			L-BFGS(W(Q))	72	0.160	0.006
			GA	22693	0.138	0.000
			PBIL	162312	0.088	0.000
			GSA	87702	0.130	0.012
			Random	54202	0.140	0.000
		MLP(4)	SD(B(I))	45869	0.103	0.015
			SD(W(I))	41440	0.101	0.014
			SD(W(F))	40042	0.101	0.011
			SD(W(Q))	40138	0.102	0.013
			BFGS(B(I))	178757	0.082	0.128
			BFGS(W(I))	170890	0.082	0.186
			BFGS(W(F))	356786	0.092	0.002
			BFGS(W(Q))	414254	0.092	0.001
			L-BFGS(B(I))	71834	0.092	0.013
			L-BFGS(W(I))	49334	0.092	0.029
			L-BFGS(W(F))	56242	0.092	0.007
			L-BFGS(W(Q))	57830	0.092	0.014
			GA	21482	0.100	0.000
			PBIL	22102	0.100	0.000
			GSA	89502	0.094	0.004
			Random	38902	0.100	0.000
		MLP(4,SM)	SD(B(I))	1390	0.108	0.000
			SD(W(I))	62	0.164	0.000
			SD(W(F))	412	0.124	0.000
			SD(W(Q))	7548	0.069	0.001
			BFGS(B(I))	246	0.175	0.000
			BFGS(W(I))	230	0.155	0.000
			BFGS(W(F))	550	0.130	0.000
			BFGS(W(Q))	186	0.156	0.000
			L-BFGS(B(I))	73	0.184	0.000
			L-BFGS(W(I))	482	0.112	0.000
			L-BFGS(W(F))	220	0.118	0.000
			L-BFGS(W(Q))	294	0.110	0.000
			GA	33274	0.124	0.000
			PBIL	37202	0.092	0.000
			GSA	22302	0.141	0.011
			Random	20302	0.128	0.000
		MLP(8)	SD(B(I))	38334	0.104	0.019
			SD(W(I))	66778	0.102	0.021
			SD(W(F))	40052	0.103	0.021

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\lVert\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\rVert$
			SD(W(Q))	40266	0.103	0.022
			BFGS(B(I))	180618	0.082	0.064
			BFGS(W(I))	477654	0.082	0.022
			BFGS(W(F))	234390	0.082	0.031
			BFGS(W(Q))	342028	0.082	0.016
			L-BFGS(B(I))	58022	0.082	0.017
			L-BFGS(W(I))	51276	0.082	0.017
			L-BFGS(W(F))	48798	0.082	0.025
			L-BFGS(W(Q))	48910	0.082	0.024
			GA	50575	0.728	0.719
			PBIL	24602	0.100	0.000
			GSA	227702	0.092	0.008
			Random	23202	178.271	15.947
		MLP(8,SM)	SD(B(I))	2827	0.105	0.000
			SD(W(I))	1324	0.108	0.000
			SD(W(F))	856	0.108	0.000
			SD(W(Q))	1190	0.108	0.001
			BFGS(B(I))	29	0.160	0.000
			BFGS(W(I))	58	0.160	0.000
			BFGS(W(F))	70	0.160	0.000
			BFGS(W(Q))	60	0.160	0.000
			L-BFGS(B(I))	24	0.160	0.000
			L-BFGS(W(I))	164	0.140	0.000
			L-BFGS(W(F))	128	0.148	0.000
			L-BFGS(W(Q))	258	0.158	0.017
			GA	22949	0.128	0.000
			PBIL	83240	0.092	0.000
			GSA	35202	0.132	0.010
			Random	22402	0.128	0.000
		RBF(2)	SD(B(I))	13968	0.106	0.110
			SD(W(I))	46698	0.106	0.042
			SD(W(F))	45466	0.106	0.075
			SD(W(Q))	46266	0.106	0.081
			BFGS(B(I))	11345	0.093	0.277
			BFGS(W(I))	169596	0.093	0.179
			BFGS(W(F))	234494	0.093	0.286
			BFGS(W(Q))	126476	0.093	0.231
			L-BFGS(B(I))	22346	0.098	0.118
			L-BFGS(W(I))	43752	0.098	0.134
			L-BFGS(W(F))	46314	0.098	0.157
			L-BFGS(W(Q))	46118	0.098	0.137
			GA	21953	8.903	0.374
			PBIL	68193	0.317	0.219
			GSA	49002	0.174	0.320
			Random	30902	9.217	0.367
		RBF(4)	SD(B(I))	45870	0.115	0.066
			SD(W(I))	47384	0.109	0.102
			SD(W(F))	43428	0.112	0.085

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\Vert\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\Vert$
			SD(W(Q))	53184	0.111	0.129
			BFGS(B(I))	28022	0.082	0.210
			BFGS(W(I))	229762	0.082	0.231
			BFGS(W(F))	211760	0.082	0.140
			BFGS(W(Q))	352110	0.082	0.137
			L-BFGS(B(I))	45047	0.083	0.162
			L-BFGS(W(I))	60494	0.096	0.125
			L-BFGS(W(F))	43410	0.095	0.155
			L-BFGS(W(Q))	49068	0.093	0.180
			GA	39019	11.734	0.295
			PBIL	127816	1.486	0.078
			GSA	74902	0.179	0.193
			Random	31902	10.993	0.252
		RBF(8)	SD(B(I))	45863	0.105	0.040
			SD(W(I))	41432	0.101	0.051
			SD(W(F))	40036	0.103	0.041
			SD(W(Q))	40074	0.104	0.033
			BFGS(B(I))	86918	0.080	0.062
			BFGS(W(I))	149346	0.080	0.074
			BFGS(W(F))	200204	0.080	0.083
			BFGS(W(Q))	237738	0.080	0.057
			L-BFGS(B(I))	48513	0.082	0.052
			L-BFGS(W(I))	88974	0.082	0.030
			L-BFGS(W(F))	76608	0.082	0.032
			L-BFGS(W(Q))	77432	0.082	0.032
			GA	39615	11.202	0.166
			PBIL	192807	1.917	0.046
			GSA	128402	0.204	0.088
			Random	25002	12.936	0.220
	HL	LinReg	SD(B(I))	5907	0.164	0.022
			SD(W(I))	50332	0.161	0.016
			SD(W(F))	40072	0.161	0.020
			SD(W(Q))	67738	0.164	0.026
			BFGS(B(I))	135375	0.089	0.045
			BFGS(W(I))	517310	0.089	0.014
			BFGS(W(F))	340588	0.089	0.013
			BFGS(W(Q))	628930	0.089	0.004
			L-BFGS(B(I))	63322	0.097	0.015
			L-BFGS(W(I))	46584	0.106	0.015
			L-BFGS(W(F))	42760	0.103	0.013
			L-BFGS(W(Q))	43616	0.105	0.017
			GA	31102	7.832	0.103
			PBIL	226037	0.955	0.018
			GSA	158702	0.159	0.037
			Random	21302	10.147	0.209
		LogReg	SD(B(I))	39	0.164	0.000
			SD(W(I))	198	0.165	0.000
			SD(W(F))	52	0.164	0.000

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\|\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\|$
			SD(W(Q))	46	0.164	0.000
			BFGS(B(I))	87	0.200	0.000
			BFGS(W(I))	94	0.180	0.000
			BFGS(W(F))	168	0.164	0.000
			BFGS(W(Q))	104	0.164	0.000
			L-BFGS(B(I))	65	0.180	0.000
			L-BFGS(W(I))	80	0.164	0.000
			L-BFGS(W(F))	32	0.164	0.000
			L-BFGS(W(Q))	52	0.164	0.000
			GA	30345	0.189	0.003
			PBIL	84727	0.154	0.000
			GSA	387902	0.150	0.000
			Random	22102	0.250	0.007
		MLP(2)	SD(B(I))	13905	0.206	0.031
			SD(W(I))	41486	0.201	0.010
			SD(W(F))	57294	0.202	0.024
			SD(W(Q))	55314	0.203	0.024
			BFGS(B(I))	7297	0.200	0.000
			BFGS(W(I))	519598	0.172	0.127
			BFGS(W(F))	204684	0.200	0.000
			BFGS(W(Q))	310578	0.164	0.013
			L-BFGS(B(I))	20040	0.200	0.000
			L-BFGS(W(I))	43752	0.200	0.032
			L-BFGS(W(F))	42596	0.200	0.013
			L-BFGS(W(Q))	43512	0.200	0.008
			GA	37253	0.180	0.000
			PBIL	36100	0.164	0.000
			GSA	33202	0.200	0.000
			Random	22502	0.200	0.000
		MLP(2,SM)	SD(B(I))	39857	0.202	0.000
			SD(W(I))	56	0.324	0.000
			SD(W(F))	40342	0.183	0.001
			SD(W(Q))	96	0.320	0.016
			BFGS(B(I))	21	0.346	0.000
			BFGS(W(I))	126	0.324	0.000
			BFGS(W(F))	128	0.324	0.000
			BFGS(W(Q))	128	0.324	0.000
			L-BFGS(B(I))	118	0.396	0.000
			L-BFGS(W(I))	224	0.295	0.000
			L-BFGS(W(F))	372	0.292	0.005
			L-BFGS(W(Q))	68	0.324	0.001
			GA	22693	0.276	0.000
			PBIL	140078	0.162	0.009
			GSA	209402	0.225	0.040
			Random	54202	0.280	0.000
		MLP(4)	SD(B(I))	45889	0.201	0.011
			SD(W(I))	78894	0.164	0.006
			SD(W(F))	40052	0.200	0.012

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\Vert\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\Vert$
			SD(W(Q))	67066	0.201	0.016
			BFGS(B(I))	210501	0.116	0.379
			BFGS(W(I))	868906	0.097	0.048
			BFGS(W(F))	393332	0.164	0.003
			BFGS(W(Q))	226618	0.144	0.005
			L-BFGS(B(I))	80111	0.200	0.000
			L-BFGS(W(I))	44870	0.164	0.004
			L-BFGS(W(F))	46506	0.164	0.008
			L-BFGS(W(Q))	42964	0.164	0.007
			GA	28489	0.184	0.000
			PBIL	196162	0.164	0.000
			GSA	92402	0.164	0.000
			Random	38902	0.200	0.000
		MLP(4,SM)	SD(B(I))	1982	0.224	0.000
			SD(W(I))	58	0.328	0.001
			SD(W(F))	522	0.248	0.000
			SD(W(Q))	488	0.248	0.000
			BFGS(B(I))	224	0.350	0.000
			BFGS(W(I))	466	0.260	0.000
			BFGS(W(F))	380	0.268	0.000
			BFGS(W(Q))	316	0.260	0.000
			L-BFGS(B(I))	205	0.367	0.000
			L-BFGS(W(I))	344	0.244	0.000
			L-BFGS(W(F))	174	0.276	0.001
			L-BFGS(W(Q))	484	0.224	0.000
			GA	33274	0.247	0.001
			PBIL	37202	0.184	0.000
			GSA	22302	0.281	0.023
			Random	20302	0.256	0.000
		MLP(8)	SD(B(I))	38543	0.200	0.020
			SD(W(I))	43846	0.164	0.011
			SD(W(F))	40038	0.174	0.044
			SD(W(Q))	40168	0.196	0.029
			BFGS(B(I))	132667	0.026	0.265
			BFGS(W(I))	611558	0.014	0.034
			BFGS(W(F))	806810	0.034	0.082
			BFGS(W(Q))	676922	0.021	0.051
			L-BFGS(B(I))	202841	0.102	0.030
			L-BFGS(W(I))	42160	0.164	0.013
			L-BFGS(W(F))	47502	0.164	0.022
			L-BFGS(W(Q))	42494	0.164	0.028
			GA	47140	0.201	0.003
			PBIL	30202	0.172	0.000
			GSA	194402	0.153	0.001
			Random	23202	93.660	7.913
		MLP(8,SM)	SD(B(I))	4669	0.177	0.001
			SD(W(I))	1582	0.216	0.000
			SD(W(F))	1062	0.216	0.000

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\lVert\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\rVert$
			SD(W(Q))	1524	0.216	0.001
			BFGS(B(I))	33	0.320	0.000
			BFGS(W(I))	56	0.320	0.000
			BFGS(W(F))	66	0.320	0.000
			BFGS(W(Q))	56	0.320	0.000
			L-BFGS(B(I))	24	0.320	0.000
			L-BFGS(W(I))	164	0.288	0.000
			L-BFGS(W(F))	160	0.288	0.000
			L-BFGS(W(Q))	254	0.292	0.102
			GA	22949	0.256	0.000
			PBIL	86554	0.176	0.000
			GSA	146702	0.245	0.030
			Random	22402	0.256	0.000
		RBF(2)	SD(B(I))	45873	0.217	0.131
			SD(W(I))	106832	0.192	0.105
			SD(W(F))	44992	0.191	0.041
			SD(W(Q))	65488	0.220	0.063
			BFGS(B(I))	11857	0.180	0.378
			BFGS(W(I))	143552	0.180	0.094
			BFGS(W(F))	216648	0.180	0.058
			BFGS(W(Q))	144146	0.180	0.043
			L-BFGS(B(I))	43594	0.180	0.027
			L-BFGS(W(I))	43370	0.180	0.190
			L-BFGS(W(F))	41834	0.181	0.028
			L-BFGS(W(Q))	42156	0.181	0.032
			GA	38375	2.510	0.070
			PBIL	67878	0.251	0.044
			GSA	34502	0.277	0.120
			Random	55802	3.964	0.241
		RBF(4)	SD(B(I))	45875	0.183	0.034
			SD(W(I))	55866	0.180	0.085
			SD(W(F))	40136	0.182	0.037
			SD(W(Q))	49784	0.182	0.039
			BFGS(B(I))	210696	0.164	0.179
			BFGS(W(I))	367036	0.164	0.006
			BFGS(W(F))	358964	0.164	0.007
			BFGS(W(Q))	330204	0.164	0.009
			L-BFGS(B(I))	22494	0.164	0.006
			L-BFGS(W(I))	46444	0.164	0.006
			L-BFGS(W(F))	42734	0.164	0.016
			L-BFGS(W(Q))	44886	0.164	0.007
			GA	22144	4.003	0.119
			PBIL	129269	0.546	0.022
			GSA	46502	0.221	0.044
			Random	32802	5.361	0.077
		RBF(8)	SD(B(I))	45888	0.166	0.013
			SD(W(I))	56806	0.165	0.014
			SD(W(F))	61958	0.165	0.014

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\|\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\|$
			SD(W(Q))	74690	0.166	0.013
			BFGS(B(I))	68642	0.098	0.007
			BFGS(W(I))	425744	0.106	0.011
			BFGS(W(F))	490724	0.110	0.015
			BFGS(W(Q))	405610	0.107	0.013
			L-BFGS(B(I))	65112	0.131	0.005
			L-BFGS(W(I))	44214	0.139	0.006
			L-BFGS(W(F))	43370	0.134	0.012
			L-BFGS(W(Q))	42752	0.141	0.010
			GA	54866	3.325	0.074
			PBIL	198770	0.318	0.009
			GSA	188002	0.165	0.019
			Random	32902	5.325	0.093
CH	MSE	LinReg	SD(B(I))	37371	0.071	0.000
			SD(W(I))	47908	0.071	0.001
			SD(W(F))	36616	0.071	0.001
			SD(W(Q))	40024	0.071	0.001
			BFGS(B(I))	141	0.071	0.000
			BFGS(W(I))	192	0.071	0.000
			BFGS(W(F))	292	0.071	0.000
			BFGS(W(Q))	228	0.071	0.000
			L-BFGS(B(I))	399	0.071	0.000
			L-BFGS(W(I))	570	0.071	0.000
			L-BFGS(W(F))	540	0.071	0.002
			L-BFGS(W(Q))	460	0.071	0.001
			GA	48466	10.298	7.505
			PBIL	30953	3.638	0.633
			GSA	47702	0.121	0.458
			Random	32102	23.347	6.973
		MLP(2)	SD(B(I))	45856	0.068	0.002
			SD(W(I))	51710	0.069	0.002
			SD(W(F))	40220	0.069	0.002
			SD(W(Q))	40026	0.069	0.002
			BFGS(B(I))	1540	0.058	0.001
			BFGS(W(I))	946	0.065	0.001
			BFGS(W(F))	1656	0.065	0.000
			BFGS(W(Q))	1008	0.065	0.000
			L-BFGS(B(I))	1911	0.063	0.001
			L-BFGS(W(I))	4838	0.062	0.000
			L-BFGS(W(F))	3228	0.063	0.002
			L-BFGS(W(Q))	2084	0.059	0.000
			GA	38025	0.226	5.723
			PBIL	39168	0.268	0.000
			GSA	66602	0.154	0.018
			Random	23002	0.271	0.017
		MLP(4)	SD(B(I))	45856	0.068	0.002
			SD(W(I))	51272	0.068	0.002
			SD(W(F))	40140	0.068	0.004

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi_c'$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\Vert\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\Vert$
			SD(W(Q))	40058	0.069	0.003
			BFGS(B(I))	3012	0.048	0.001
			BFGS(W(I))	4024	0.049	0.001
			BFGS(W(F))	2866	0.051	0.000
			BFGS(W(Q))	2154	0.051	0.001
			L-BFGS(B(I))	9762	0.049	0.000
			L-BFGS(W(I))	7046	0.053	0.000
			L-BFGS(W(F))	8722	0.050	0.000
			L-BFGS(W(Q))	7972	0.049	0.001
			GA	36494	0.260	0.220
			PBIL	33701	0.227	0.062
			GSA	31202	0.130	0.062
			Random	20602	0.273	0.000
		MLP(8)	SD(B(I))	45856	0.067	0.003
			SD(W(I))	52022	0.068	0.003
			SD(W(F))	40168	0.067	0.005
			SD(W(Q))	40030	0.068	0.004
			BFGS(B(I))	4022	0.039	0.002
			BFGS(W(I))	7602	0.042	0.000
			BFGS(W(F))	14122	0.038	0.001
			BFGS(W(Q))	5008	0.040	0.000
			L-BFGS(B(I))	33682	0.039	0.001
			L-BFGS(W(I))	20584	0.044	0.000
			L-BFGS(W(F))	22898	0.039	0.002
			L-BFGS(W(Q))	22162	0.039	0.001
			GA	49575	0.273	0.000
			PBIL	135615	0.266	0.000
			GSA	54802	0.210	0.304
			Random	33602	994.489	407.509
		RBF(2)	SD(B(I))	250	0.231	0.000
			SD(W(I))	268	0.231	0.000
			SD(W(F))	162	0.231	0.001
			SD(W(Q))	214	0.231	0.001
			BFGS(B(I))	40	0.231	0.000
			BFGS(W(I))	56	0.231	0.000
			BFGS(W(F))	108	0.231	0.000
			BFGS(W(Q))	88	0.231	0.000
			L-BFGS(B(I))	40	0.231	0.000
			L-BFGS(W(I))	64	0.231	0.000
			L-BFGS(W(F))	96	0.231	0.000
			L-BFGS(W(Q))	102	0.231	0.000
			GA	66478	0.232	0.046
			PBIL	11862	0.231	0.000
			GSA	28202	0.231	0.011
			Random	32802	0.250	0.202
		RBF(4)	SD(B(I))	199	0.231	0.000
			SD(W(I))	252	0.231	0.000
			SD(W(F))	200	0.231	0.001

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\lVert\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\rVert$
			SD(W(Q))	252	0.231	0.001
			BFGS(B(I))	67	0.231	0.000
			BFGS(W(I))	92	0.231	0.000
			BFGS(W(F))	96	0.231	0.000
			BFGS(W(Q))	88	0.231	0.000
			L-BFGS(B(I))	80	0.231	0.000
			L-BFGS(W(I))	120	0.231	0.000
			L-BFGS(W(F))	104	0.231	0.000
			L-BFGS(W(Q))	116	0.231	0.000
			GA	39655	0.419	0.533
			PBIL	18116	0.254	0.051
			GSA	39802	0.232	0.015
			Random	58402	1.781	1.534
		RBF(8)	SD(B(I))	1744	0.156	0.000
			SD(W(I))	2564	0.156	0.000
			SD(W(F))	1652	0.156	0.001
			SD(W(Q))	2274	0.156	0.001
			BFGS(B(I))	112	0.156	0.000
			BFGS(W(I))	160	0.156	0.000
			BFGS(W(F))	180	0.156	0.000
			BFGS(W(Q))	166	0.156	0.000
			L-BFGS(B(I))	138	0.156	0.000
			L-BFGS(W(I))	238	0.156	0.000
			L-BFGS(W(F))	160	0.156	0.000
			L-BFGS(W(Q))	178	0.156	0.000
			GA	64059	5.826	3.021
			PBIL	22360	0.267	0.103
			GSA	63802	0.162	0.029
			Random	38602	7.243	0.968
	MAE	LinReg	SD(B(I))	6278	0.208	0.017
			SD(W(I))	53664	0.207	0.012
			SD(W(F))	56116	0.207	0.009
			SD(W(Q))	46990	0.207	0.014
			BFGS(B(I))	11755	0.197	0.013
			BFGS(W(I))	41440	0.197	0.002
			BFGS(W(F))	69102	0.197	0.001
			BFGS(W(Q))	112524	0.197	0.001
			L-BFGS(B(I))	17093	0.197	0.002
			L-BFGS(W(I))	42302	0.197	0.009
			L-BFGS(W(F))	41882	0.197	0.015
			L-BFGS(W(Q))	41834	0.197	0.010
			GA	31311	2.519	1.126
			PBIL	37550	2.897	0.117
			GSA	52402	0.223	0.133
			Random	38902	3.530	0.463
		MLP(2)	SD(B(I))	4516	0.213	0.029
			SD(W(I))	50346	0.213	0.022
			SD(W(F))	53664	0.212	0.020

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\lVert\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\rVert$
			SD(W(Q))	57436	0.211	0.020
			BFGS(B(I))	246804	0.174	0.009
			BFGS(W(I))	42992	0.179	0.009
			BFGS(W(F))	81766	0.184	0.006
			BFGS(W(Q))	76448	0.172	0.023
			L-BFGS(B(I))	27970	0.451	0.000
			L-BFGS(W(I))	42402	0.197	0.034
			L-BFGS(W(F))	43296	0.197	0.162
			L-BFGS(W(Q))	41686	0.200	0.026
			GA	62899	0.403	3.550
			PBIL	24093	0.444	0.625
			GSA	34202	0.275	0.096
			Random	54402	0.446	0.014
		MLP(4)	SD(B(I))	14771	0.213	0.020
			SD(W(I))	69910	0.211	0.020
			SD(W(F))	40410	0.211	0.020
			SD(W(Q))	56352	0.216	0.023
			BFGS(B(I))	97223	0.153	0.125
			BFGS(W(I))	62312	0.167	0.028
			BFGS(W(F))	62408	0.166	0.012
			BFGS(W(Q))	97942	0.157	0.028
			L-BFGS(B(I))	73389	0.172	0.011
			L-BFGS(W(I))	43446	0.190	0.025
			L-BFGS(W(F))	42142	0.198	0.007
			L-BFGS(W(Q))	43978	0.186	0.011
			GA	21246	0.406	1.641
			PBIL	40475	0.443	0.006
			GSA	32702	0.274	0.090
			Random	20602	0.451	0.000
		MLP(8)	SD(B(I))	9149	0.209	0.031
			SD(W(I))	53942	0.208	0.021
			SD(W(F))	47974	0.208	0.022
			SD(W(Q))	47290	0.208	0.022
			BFGS(B(I))	91993	0.130	0.123
			BFGS(W(I))	262444	0.138	0.117
			BFGS(W(F))	116222	0.136	0.074
			BFGS(W(Q))	124840	0.136	0.160
			L-BFGS(B(I))	142755	0.404	0.009
			L-BFGS(W(I))	45336	0.173	0.014
			L-BFGS(W(F))	45256	0.169	0.026
			L-BFGS(W(Q))	43416	0.177	0.015
			GA	47463	0.450	0.001
			PBIL	124186	0.255	0.603
			GSA	62802	0.232	0.158
			Random	33602	5.194	1.480
		RBF(2)	SD(B(I))	1288	0.371	0.001
			SD(W(I))	41558	0.371	0.002
			SD(W(F))	41514	0.371	0.001

(cont. on next page)

Table B.1 (cont.)

Dataset	ξ	Model	Optimizer	$\xi_c + \xi'_c$	$\xi(f(\boldsymbol{X}), \boldsymbol{Y})$	$\|\xi'(f(\boldsymbol{X}), \boldsymbol{Y})\|$
			SD(W(Q))	41532	0.371	0.001
			BFGS(B(I))	1316	0.371	0.002
			BFGS(W(I))	80120	0.371	0.001
			BFGS(W(F))	99134	0.371	0.001
			BFGS(W(Q))	138260	0.371	0.005
			L-BFGS(B(I))	1455	0.371	0.005
			L-BFGS(W(I))	41454	0.371	0.001
			L-BFGS(W(F))	41288	0.371	0.005
			L-BFGS(W(Q))	41270	0.371	0.002
			GA	23248	0.373	0.093
			PBIL	10685	0.371	0.001
			GSA	25202	0.371	0.005
			Random	34802	0.389	0.100
		RBF(4)	SD(B(I))	28360	0.371	0.006
			SD(W(I))	43804	0.371	0.005
			SD(W(F))	41924	0.371	0.005
			SD(W(Q))	41834	0.371	0.006
			BFGS(B(I))	4165	0.371	0.005
			BFGS(W(I))	97138	0.371	0.004
			BFGS(W(F))	41348	0.371	0.004
			BFGS(W(Q))	57364	0.371	0.001
			L-BFGS(B(I))	11422	0.371	0.001
			L-BFGS(W(I))	41298	0.371	0.005
			L-BFGS(W(F))	41580	0.371	0.004
			L-BFGS(W(Q))	41414	0.371	0.001
			GA	82613	0.600	0.573
			PBIL	22974	0.371	0.005
			GSA	44102	0.371	0.046
			Random	58402	0.955	0.537
		RBF(8)	SD(B(I))	3608	0.295	0.006
			SD(W(I))	44384	0.295	0.006
			SD(W(F))	44360	0.295	0.006
			SD(W(Q))	43958	0.295	0.005
			BFGS(B(I))	11258	0.294	0.003
			BFGS(W(I))	248482	0.294	0.002
			BFGS(W(F))	324878	0.294	0.002
			BFGS(W(Q))	137986	0.294	0.005
			L-BFGS(B(I))	12111	0.294	0.004
			L-BFGS(W(I))	41702	0.294	0.005
			L-BFGS(W(F))	41576	0.295	0.002
			L-BFGS(W(Q))	41714	0.294	0.009
			GA	27721	1.723	0.307
			PBIL	26343	0.341	0.028
			GSA	64302	0.303	0.024
			Random	38602	2.340	0.203

Bibliography

[1] Martin T Hagan, Howard B Demuth, Mark H Beale, et al. *Neural network design*. Pws Pub. Boston, 1996.

[2] Christopher Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer-Verlag New York, 1 edition, 2006.

[3] Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd international conference on Machine learning*, pages 161–168. ACM, 2006.

[4] Xiaoshuang Shi, Yujiu Yang, Zhenhua Guo, and Zhihui Lai. Face recognition by sparse discriminant analysis via joint l 2, 1-norm minimization. *Pattern Recognition*, 47(7):2447–2453, 2014.

[5] Chen Chi Hau. *Handbook of pattern recognition and computer vision*. World Scientific, 2015.

[6] Justin Lovinger and Iren Valova. Neural field: Supervised apportioned incremental learning (sail). In *2016 International Joint Conference on Neural Networks (IJCNN)*, pages 2500–2506, July 2016.

[7] Leo Breiman. *Classification and regression trees*. Routledge, 2017.

[8] John Neter, Michael H Kutner, Christopher J Nachtsheim, and William Wasserman. *Applied linear statistical models*, volume 4. Irwin Chicago, 1996.

[9] George AF Seber and Alan J Lee. *Linear regression analysis*, volume 936. John Wiley & Sons, 2012.

[10] Douglas C Montgomery, Elizabeth A Peck, and G Geoffrey Vining. *Introduction to linear regression analysis*. John Wiley & Sons, 2015.

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

[12] Robert Tibshirani. Regression shrinkage and selection via the lasso: a retrospective. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73(3):273–282, 2011.

[13] John Wright, Allen Y Yang, Arvind Ganesh, S Shankar Sastry, and Yi Ma. Robust face recognition via sparse representation. *IEEE transactions on pattern analysis and machine intelligence*, 31(2):210–227, 2009.

[14] Roberto Tagliaferri, Giuseppe Longo, Leopoldo Milano, Fausto Acernese, Fabrizio Barone, Angelo Ciaramella, Rosario De Rosa, Ciro Donalek, Antonio Eleuteri, Giancarlo Raiconi, et al. Neural networks in astronomy. *Neural Networks*, 16(3):297–319, 2003.

[15] Mohammad Zare, Hamid Reza Pourghasemi, Mahdi Vafakhah, and Biswajeet Pradhan. Landslide susceptibility mapping at vaz watershed (iran) using an artificial neural network model: a comparison between multilayer perceptron (mlp) and radial basic function (rbf) algorithms. *Arabian Journal of Geosciences*, 6(8):2873–2888, 2013.

[16] Ali Bou Nassif, Danny Ho, and Luiz Fernando Capretz. Towards an early software estimation using log-linear regression and a multilayer perceptron model. *Journal of Systems and Software*, 86(1):144–160, 2013.

[17] Mohammad Hemmat Esfe, Masoud Afrand, Somchai Wongwises, Ali Naderi, Amin Asadi, Sara Rostami, and Mohammad Akbari. Applications of feedforward multilayer perceptron artificial neural networks and empirical correlation for prediction of thermal conductivity of mg (oh) 2–eg using experimental data. *International Communications in Heat and Mass Transfer*, 67:46–50, 2015.

[18] Leon A Gatys, Alexander S Ecker, and Matthias Bethge. A neural algorithm of artistic style. *arXiv preprint arXiv:1508.06576*, 2015.

[19] Paul E Utgoff. Incremental induction of decision trees. *Machine learning*, 4(2):161–186, 1989.

[20] Helmut Schmid. Probabilistic part-ofispeech tagging using decision trees. In *New methods in language processing*, page 154. Routledge, 2013.

[21] Biswajeet Pradhan. A comparative study on the predictive ability of the decision tree, support vector machine and neuro-fuzzy models in landslide susceptibility mapping using gis. *Computers & Geosciences*, 51:350–365, 2013.

[22] Claude Elwood Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001.

[23] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[24] Tin Kam Ho. Random decision forests. In *Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on*, volume 1, pages 278–282. IEEE, 1995.

[25] Ramón Díaz-Uriarte and Sara Alvarez De Andres. Gene selection and classification of microarray data using random forest. *BMC bioinformatics*, 7(1):1, 2006.

[26] Victor Francisco Rodriguez-Galiano, Bardan Ghimire, John Rogan, Mario Chica-Olmo, and Juan Pedro Rigol-Sanchez. An assessment of the effectiveness of a random forest classifier for land-cover classification. *ISPRS Journal of Photogrammetry and Remote Sensing*, 67:93–104, 2012.

[27] David Lowe and D Broomhead. Multivariable functional interpolation and adaptive networks. *Complex syst*, 2:321–355, 1988.

[28] David S Broomhead and David Lowe. Radial basis functions, multi-variable functional interpolation and adaptive networks. Technical report, DTIC Document, 1988.

[29] Ying Tan, Jun Wang, and Jacek M Zurada. Nonlinear blind source separation using a radial basis function network. *Neural Networks, IEEE Transactions on*, 12(1):124–134, 2001.

[30] Ke-Lin Du and MNS Swamy. Radial basis function networks. In *Neural Networks and Statistical Learning*, pages 299–335. Springer, 2014.

[31] Ruey-Jing Lian. Adaptive self-organizing fuzzy sliding-mode radial basis-function neural-network controller for robotic systems. *IEEE Transactions on Industrial Electronics*, 61(3):1493–1503, 2014.

[32] Qichao Que and Mikhail Belkin. Back to the future: Radial basis function networks revisited. In *AISTATS*, pages 1375–1383, 2016.

[33] S. Wright Jorge Nocedal. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer-Verlag New York, 2 edition, 2006.

[34] Jan Snyman. *Practical mathematical optimization: an introduction to basic optimization theory and classical and new gradient-based algorithms*, volume 97. Springer Science & Business Media, 2005.

[35] Michael D Intriligator. *Mathematical optimization and economic theory*, volume 39. Siam, 1971.

[36] Philip E Gill, Walter Murray, and Margaret H Wright. *Practical optimization*. Academic Press, 1981.

[37] Dervis Karaboga. An idea based on honey bee swarm for numerical optimization. Technical report, Technical report-tr06, Erciyes University, Engineering Faculty, Computer Engineering Department, 2005.

[38] Gai-Ge Wang, Amir H Gandomi, Xiangjun Zhao, and Hai Cheng Eric Chu. Hybridizing harmony search algorithm with cuckoo search for global numerical optimization. *Soft Computing*, 20(1):273–285, 2016.

[39] Gai-Ge Wang, Amir H Gandomi, Amir H Alavi, and Guo-Sheng Hao. Hybrid krill herd algorithm with differential evolution for global numerical optimization. *Neural Computing and Applications*, 25(2):297–308, 2014.

[40] Behrooz Keshtegar and Mahmoud Miri. Introducing conjugate gradient optimization for modified hl-rf method. *Engineering Computations*, 31(4):775–790, 2014.

[41] Youhei Akimoto, Anne Auger, and Nikolaus Hansen. Comparison-based natural gradient optimization in high dimension. In *Proceedings of the 2014 Annual*

Conference on Genetic and Evolutionary Computation, pages 373–380. ACM, 2014.

[42] Georgios Tzimiropoulos. Project-out cascaded regression with an application to face alignment. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3659–3667, 2015.

[43] Sen Zhang, Lihua Xie, and Martin David Adams. Feature extraction for outdoor mobile robot navigation based on a modified gauss–newton optimization approach. *Robotics and Autonomous Systems*, 54(4):277–287, 2006.

[44] Andreas Fischer. A special newton-type optimization method. *Optimization*, 24(3-4):269–284, 1992.

[45] Michel Sarkis, Klaus Diepold, and Knut Huper. A fast and robust solution to the five-pint relative pose problem using gauss-newton optimization on a manifold. In *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*, volume 1, pages I–681. IEEE, 2007.

[46] Davide A Cucci and Matteo Matteucci. Position tracking and sensors self-calibration in autonomous mobile robots by gauss-newton optimization. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 1269–1275. IEEE, 2014.

[47] Rafal Zdunek and Andrzej Cichocki. Non-negative matrix factorization with quasi-newton optimization. In *International conference on artificial intelligence and soft computing*, pages 870–879. Springer, 2006.

[48] Dmitry Avdeev and Anna Avdeeva. 3d magnetotelluric inversion using a limited-memory quasi-newton optimization. *Geophysics*, 74(3):F45–F57, 2009.

[49] Oleg Trott and Arthur J Olson. Autodock vina: improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading. *Journal of computational chemistry*, 31(2):455–461, 2010.

[50] Umut Simsekli, Roland Badeau, Taylan Cemgil, and Gaël Richard. Stochastic quasi-newton langevin monte carlo. In *International Conference on Machine Learning*, pages 642–651, 2016.

[51] Richard H Byrd, Samantha L Hansen, Jorge Nocedal, and Yoram Singer. A stochastic quasi-newton method for large-scale optimization. *SIAM Journal on Optimization*, 26(2):1008–1031, 2016.

[52] Jascha Sohl-Dickstein, Ben Poole, and Surya Ganguli. Fast large-scale optimization by unifying stochastic gradient and quasi-newton methods. In *International Conference on Machine Learning*, pages 604–612, 2014.

[53] Wen Huang, Kyle A Gallivan, and P-A Absil. A broyden class of quasi-newton methods for riemannian optimization. *SIAM Journal on Optimization*, 25(3):1660–1685, 2015.

[54] Martin Hanke. *Conjugate gradient type methods for ill-posed problems*. Routledge, 2017.

[55] Martin Fodslette Møller. A scaled conjugate gradient algorithm for fast supervised learning. *Neural networks*, 6(4):525–533, 1993.

[56] Michael James David Powell. Restart procedures for the conjugate gradient method. *Mathematical programming*, 12(1):241–254, 1977.

[57] Jonathan Richard Shewchuk et al. An introduction to the conjugate gradient method without the agonizing pain. Technical report, Carnegie-Mellon University. Department of Computer Science, 1994.

[58] Gonglin Yuan and Maojun Zhang. A three-terms polak–ribière–polyak conjugate gradient algorithm for large-scale nonlinear equations. *Journal of Computational and Applied Mathematics*, 286:186–195, 2015.

[59] Pieter Ghysels and Wim Vanroose. Hiding global synchronization latency in the preconditioned conjugate gradient algorithm. *Parallel Computing*, 40(7):224–238, 2014.

[60] John A Nelder and Robert Mead. The downhill simplex algorithm. *Computer Journal*, 7(S 308), 1965.

[61] Noriyasu Maehara and Yoshiyuki Shimoda. Application of the genetic algorithm and downhill simplex methods (nelder–mead methods) in the search for the optimum chiller configuration. *Applied Thermal Engineering*, 61(2):433–442, 2013.

[62] Nikolaus Hansen. Benchmarking the nelder-mead downhill simplex algorithm with many local restarts. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, pages 2403–2408. ACM, 2009.

[63] R John Koshel. Enhancement of the downhill simplex method of optimization. In *International Optical Design Conference*, page ITuC2. Optical Society of America, 2002.

[64] M Huang, Cheryl J Aine, Selma Supek, Elaine Best, Douglas Ranken, and ER Flynn. Multi-start downhill simplex method for spatio-temporal source

localization in magnetoencephalography. *Electroencephalography and Clinical Neurophysiology/Evoked Potentials Section*, 108(1):32–44, 1998.

[65] M Nawaz, A Zeeshan, R Ellahi, S Abbasbandy, and Saman Rashidi. Joules and newtonian heating effects on stagnation point flow over a stretching surface by means of genetic algorithm and nelder-mead method. *International Journal of Numerical Methods for Heat & Fluid Flow*, 25(3):665–684, 2015.

[66] Darrell Whitley. A genetic algorithm tutorial. *Statistics and computing*, 4(2):65–85, 1994.

[67] Jihoon Yang and Vasant Honavar. Feature subset selection using a genetic algorithm. *IEEE Intelligent Systems and their Applications*, 13(2):44–49, 1998.

[68] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE transactions on evolutionary computation*, 6(2):182–197, 2002.

[69] Harun Uğuz. A two-stage feature selection method for text categorization by using information gain, principal component analysis and genetic algorithm. *Knowledge-Based Systems*, 24(7):1024–1032, 2011.

[70] Justin Lovinger, Iren Valova, MacKenzie Rogers, Ryan Nadeau, and Natacha Gueorguieva. Harnessing mother nature: Optimizing genetic algorithms for adaptive systems. *Procedia Computer Science*, 36:523–528, 2014.

[71] Andreas Wächter and Lorenz T Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106(1):25–57, 2006.

[72] Luigi Grippo, Francesco Lampariello, and Stephano Lucidi. A nonmonotone line search technique for newtonâĂŹs method. *SIAM Journal on Numerical Analysis*, 23(4):707–716, 1986.

[73] William W Hager and Hongchao Zhang. A new conjugate gradient method with guaranteed descent and an efficient line search. *SIAM Journal on optimization*, 16(1):170–192, 2005.

[74] Jorge J Moré and David J Thuente. Line search algorithms with guaranteed sufficient decrease. *ACM Transactions on Mathematical Software (TOMS)*, 20(3):286–307, 1994.

[75] T Gerasimov and L De Lorenzis. A line search assisted monolithic approach for phase-field computing of brittle fracture. *Computer Methods in Applied Mechanics and Engineering*, 312:276–303, 2016.

[76] Gonglin Yuan, Zengxin Wei, and Xiwen Lu. Global convergence of bfgs and prp methods under a modified weak wolfe–powell line search. *Applied Mathematical Modelling*, 47:811–825, 2017.

[77] Danny C Sorensen. NewtonâĂŹs method with a model trust region modification. *SIAM Journal on Numerical Analysis*, 19(2):409–426, 1982.

[78] Jorge J Moré and Danny C Sorensen. Computing a trust region step. *SIAM Journal on Scientific and Statistical Computing*, 4(3):553–572, 1983.

[79] Thomas F Coleman and Yuying Li. An interior trust region approach for nonlinear minimization subject to bounds. *SIAM Journal on optimization*, 6(2):418–445, 1996.

[80] Andrew R Conn, Nicholas IM Gould, and Ph L Toint. *Trust region methods*, volume 1. Siam, 2000.

[81] Ya-xiang Yuan. Recent advances in trust region algorithms. *Mathematical Programming*, 151(1):249–281, 2015.

[82] Frank E Curtis, Daniel P Robinson, and Mohammadreza Samadi. A trust region algorithm with a worst-case iteration complexity of \mathcal {O}(\epsilonˆ{-3/2}) o (Ïţ-3/2) for nonconvex optimization. *Mathematical Programming*, 162(1-2):1–32, 2017.

[83] Allen A Goldstein. On steepest descent. *Journal of the Society for Industrial and Applied Mathematics, Series A: Control*, 3(1):147–151, 1965.

[84] Roberto Battiti. First-and second-order methods for learning: between steepest descent and newton's method. *Neural computation*, 4(2):141–166, 1992.

[85] Isao Yamada. The hybrid steepest descent method for the variational inequality problem over the intersection of fixed point sets of nonexpansive mappings. *Inherently parallel algorithms in feasibility and optimization and their applications*, 8:473–504, 2001.

[86] Rouhollah Tavakoli. Multimaterial topology optimization by volume constrained allen–cahn system and regularized projected steepest descent method. *Computer Methods in Applied Mechanics and Engineering*, 276:534–565, 2014.

[87] Yi-Fei Pu, Ji-Liu Zhou, Yi Zhang, Ni Zhang, Guo Huang, and Patrick Siarry. Fractional extreme value adaptive training method: fractional steepest descent approach. *IEEE transactions on neural networks and learning systems*, 26(4):653–662, 2015.

[88] Jared Tanner and Ke Wei. Low rank matrix completion by alternating steepest descent methods. *Applied and Computational Harmonic Analysis*, 40(2):417–429, 2016.

[89] Ya-xiang Yuan. A modified bfgs algorithm for unconstrained optimization. *IMA Journal of Numerical Analysis*, 11(3):325–332, 1991.

[90] Dong-Hui Li and Masao Fukushima. A modified bfgs method and its global convergence in nonconvex minimization. *Journal of Computational and Applied Mathematics*, 129(1-2):15–35, 2001.

[91] Yu-Hong Dai. Convergence properties of the bfgs algoritm. *SIAM Journal on Optimization*, 13(3):693–701, 2002.

[92] Aryan Mokhtari and Alejandro Ribeiro. Res: Regularized stochastic bfgs algorithm. *IEEE Transactions on Signal Processing*, 62(23):6089–6104, 2014.

[93] Frank E Curtis, Tim Mitchell, and Michael L Overton. A bfgs-sqp method for nonsmooth, nonconvex, constrained optimization and its evaluation using relative minimization profiles. *Optimization Methods and Software*, 32(1):148–181, 2017.

[94] Dong C Liu and Jorge Nocedal. On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45(1-3):503–528, 1989.

[95] Stephen G Nash and Jorge Nocedal. A numerical study of the limited memory bfgs method and the truncated-newton method for large scale optimization. *SIAM Journal on Optimization*, 1(3):358–372, 1991.

[96] Ciyou Zhu, Richard H Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software (TOMS)*, 23(4):550–560, 1997.

[97] Weizhu Chen, Zhenghao Wang, and Jingren Zhou. Large-scale l-bfgs using mapreduce. In *Advances in Neural Information Processing Systems*, pages 1332–1340, 2014.

[98] Philipp Moritz, Robert Nishihara, and Michael Jordan. A linearly-convergent stochastic l-bfgs algorithm. In *Artificial Intelligence and Statistics*, pages 249–258, 2016.

[99] Robert Gower, Donald Goldfarb, and Peter Richtárik. Stochastic block bfgs: Squeezing more curvature out of data. In *International Conference on Machine Learning*, pages 1869–1878, 2016.

[100] Larry Armijo. Minimization of functions having lipschitz continuous first partial derivatives. *Pacific Journal of mathematics*, 16(1):1–3, 1966.

[101] Pei-Ling Liu and Armen Der Kiureghian. Optimization algorithms for structural reliability. *Structural safety*, 9(3):161–177, 1991.

[102] Sigen Ye and Rick S Blum. Optimized signaling for mimo interference systems with feedback. *IEEE Transactions on Signal Processing*, 51(11):2839–2848, 2003.

[103] Juan Carlos De los Reyes. Numerical optimization methods. In *Numerical PDE-Constrained Optimization*, pages 43–68. Springer, 2015.

[104] Philip Wolfe. Convergence conditions for ascent methods. *SIAM review*, 11(2):226–235, 1969.

[105] Philip Wolfe. Convergence conditions for ascent methods. ii: Some corrections. *SIAM review*, 13(2):185–188, 1971.

[106] Jorge Nocedal and Ya-xiang Yuan. Combining trust region and line search techniques. In *Advances in nonlinear programming*, pages 153–175. Springer, 1998.

[107] Andreas Wächter and Lorenz T Biegler. Line search filter methods for nonlinear programming: Motivation and global convergence. *SIAM Journal on Optimization*, 16(1):1–31, 2005.

[108] Richard A Waltz, José Luis Morales, Jorge Nocedal, and Dominique Orban. An interior algorithm for nonlinear optimization that combines line search and trust region steps. *Mathematical programming*, 107(3):391–408, 2006.

[109] Yonggang Pei and Detong Zhu. An affine scaling interior trust-region algorithm combining backtracking line search with filter technique for nonlinear constrained optimization. *Numerical Functional Analysis and Optimization*, 36(8):1046–1066, 2015.

[110] Yuxin Chen and Emmanuel Candes. Solving random quadratic systems of equations is nearly as easy as solving linear systems. In *Advances in Neural Information Processing Systems*, pages 739–747, 2015.

[111] Clément W Royer and Stephen J Wright. Complexity analysis of second-order line-search algorithms for smooth nonconvex optimization. *arXiv preprint arXiv:1706.03131*, 2017.

[112] Xiangrong Li, Xiaoliang Wang, Zhou Sheng, and Xiabin Duan. A modified conjugate gradient algorithm with backtracking line search technique for large-

scale nonlinear equations. *International Journal of Computer Mathematics*, 95(2):382–395, 2018.

[113] Roger Fletcher. *Practical methods of optimization*. John Wiley & Sons, 2013.

[114] Yu-Hong Dai and Cai-Xia Kou. A nonlinear conjugate gradient algorithm with an optimal property and an improved wolfe line search. *SIAM Journal on Optimization*, 23(1):296–320, 2013.

[115] Robert B. Schnabel J. E. Dennis. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Classics in Applied Mathematics. Society for Industrial and Applied Mathematics, 1 edition, 1987.

[116] Shumeet Baluja. Population-based incremental learning. a method for integrating genetic search based function optimization and competitive learning. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Dept Of Computer Science, 1994.

[117] Michèle Sebag and Antoine Ducoulombier. Extending population-based incremental learning to continuous search spaces. In *International Conference on Parallel Problem Solving from Nature*, pages 418–427. Springer, 1998.

[118] Shengxiang Yang and Xin Yao. Experimental study on population-based incremental learning algorithms for dynamic optimization problems. *Soft Computing*, 9(11):815–834, 2005.

[119] Gai-Ge Wang, Suash Deb, Amir H Gandomi, and Amir H Alavi. A hybrid pbil-based krill herd algorithm. In *Computational and Business Intelligence (ISCBI), 2015 3rd International Symposium on*, pages 39–44. IEEE, 2015.

[120] Meng-Hui Chen, Pei-Chann Chang, and Jheng-Long Wu. A population-based incremental learning approach with artificial immune system for network intrusion detection. *Engineering Applications of Artificial Intelligence*, 51:171–181, 2016.

[121] Xianghu Meng, Jun Li, MengChu Zhou, Xianzhong Dai, and Jianping Dou. Population-based incremental learning algorithm for a serial colored traveling salesman problem. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 48(2):277–288, 2018.

[122] Esmat Rashedi, Hossein Nezamabadi-Pour, and Saeid Saryazdi. GSA: a gravitational search algorithm. *Information sciences*, 179(13):2232–2248, 2009.

[123] Esmat Rashedi, Hossein Nezamabadi-Pour, and Saeid Saryazdi. BGSA: binary gravitational search algorithm. *Natural Computing*, 9(3):727–745, 2010.

[124] Melvin Gauci, Tony J Dodd, and Roderich Groß. Why 'GSA: a gravitational search algorithm' is not genuinely based on the law of gravity. *Natural Computing*, 11(4):719–720, 2012.

[125] Binod Shaw, Vivekananda Mukherjee, and Saktiprasad Ghoshal. Solution of reactive power dispatch of power systems by an opposition-based gravitational search algorithm. *International Journal of Electrical Power & Energy Systems*, 55:29–40, 2014.

[126] Seyedali Mirjalili and Andrew Lewis. Adaptive gbest-guided gravitational search algorithm. *Neural Computing and Applications*, 25(7-8):1569–1584, 2014.

[127] Mohammad Rasoul Narimani, Ali Azizi Vahed, Rasoul Azizipanah-Abarghooee, and Mahshid Javidsharifi. Enhanced gravitational search algorithm for multi-

objective distribution feeder reconfiguration considering reliability, loss and operational cost. *IET Generation, Transmission & Distribution*, 8(1):55–69, 2014.

[128] Kenneth Alan De Jong. *An Analysis of the Behavior of a Class of Genetic Adaptive Systems*. PhD dissertation, University of Michigan, Ann Arbor, MI, USA, 1975.

[129] David E Goldberg and Kalyanmoy Deb. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of genetic algorithms*, volume 1, pages 69–93. Elsevier, 1991.

[130] Anupriya Shukla, Hari Mohan Pandey, and Deepti Mehrotra. Comparative review of selection techniques in genetic algorithm. In *Futuristic Trends on Computational Analysis and Knowledge Management (ABLAZE), 2015 International Conference on*, pages 515–519. IEEE, 2015.

[131] Anne Brindle. *Genetic algorithms for function optimization*. PhD dissertation, University of Alberta, Edmonton, Alberta, Canada, 1981.

[132] Tomasz Dominik Gwiazda. *Crossover for single-objective numerical optimization problems*, volume 1. Tomasz Gwiazda, 2006.

[133] Riccardo Poli and William B Langdon. Schema theory for genetic programming with one-point crossover and point mutation. *Evolutionary Computation*, 6(3):231–252, 1998.

[134] Lauro de Paula, Anderson S Soares, Telma Woerle de Lima, and Clarimar J Coelho. Feature selection using genetic algorithm: An analysis of the bias-property for one-point crossover. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*, pages 1461–1462. ACM, 2016.

[135] Justin Lovinger, Iren Valova, and Chad Clough. Gist: general integrated summarization of text and reviews. *Soft Computing*, pages 1–13, 2017.

[136] William M Spears and Kenneth A De Jong. An analysis of multi-point crossover. In *Foundations of genetic algorithms*, volume 1, pages 301–315. Elsevier, 1991.

[137] Kenneth A De Jong and William M Spears. A formal analysis of the role of multi-point crossover in genetic algorithms. *Annals of mathematics and Artificial intelligence*, 5(1):1–26, 1992.

[138] William M Spears and Kenneth D De Jong. On the virtues of parameterized uniform crossover. Technical report, Naval Research Lab Washington DC, 1995.

[139] Francisco Chicano and Enrique Alba. Exact computation of the expectation curves of the bit-flip mutation using landscapes theory. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 2027–2034. ACM, 2011.

[140] Francisco Chicano, Andrew M Sutton, L Darrell Whitley, and Enrique Alba. Fitness probability distribution of bit-flip mutation. *Evolutionary computation*, 23(2):217–248, 2015.

[141] Robert Hinterding. Gaussian mutation and self-adaption for numeric genetic algorithms. In *Evolutionary Computation, 1995., IEEE International Conference on*, volume 1, page 384. IEEE, 1995.

[142] MHz‘as. 2 dimensional contour graph. https://commons.wikimedia.org/wiki/File:Contour2D.svg, 2012. Accessed: 2016-08-21.

[143] Robert Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural networks for perception*, pages 65–93. Elsevier, 1992.

[144] Louis B Rall. Automatic differentiation: Techniques and applications. 1981.

[145] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.

[146] László Tóth. Phone recognition with deep sparse rectifier neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 6985–6989. IEEE, 2013.

[147] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. ICML*, volume 30, page 1, 2013.

[148] Nasser M Nasrabadi. Pattern recognition and machine learning. *Journal of electronic imaging*, 16(4):049901, 2007.

[149] Narasimha GN Prasad and Jon NK Rao. The estimation of the mean squared error of small-area estimators. *Journal of the American statistical association*, 85(409):163–171, 1990.

[150] Michael Tuchler, Andrew C Singer, and Ralf Koetter. Minimum mean squared error equalization using a priori information. *IEEE Transactions on Signal Processing*, 50(3):673–683, 2002.

[151] Zhou Wang and Alan C Bovik. Mean squared error: Love it or leave it? a new look at signal fidelity measures. *IEEE signal processing magazine*, 26(1):98–117, 2009.

[152] Tianfeng Chai and Roland R Draxler. Root mean square error (rmse) or mean absolute error (mae)?–arguments against avoiding rmse in the literature. *Geoscientific model development*, 7(3):1247–1250, 2014.

[153] Badong Chen, Lei Xing, Junli Liang, Nanning Zheng, and Jose C Principe. Steady-state mean-square error analysis for adaptive filtering under the maximum correntropy criterion. *IEEE signal processing letters*, 21(7):880–884, 2014.

[154] Edward J Coyle and J-H Lin. Stack filters and the mean absolute error criterion. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 36(8):1244–1254, 1988.

[155] Cort J Willmott and Kenji Matsuura. Advantages of the mean absolute error (mae) over the root mean square error (rmse) in assessing average model performance. *Climate research*, 30(1):79–82, 2005.

[156] Nicholas G Reich, Justin Lessler, Krzysztof Sakrejda, Stephen A Lauer, Sopon Iamsirithaworn, and Derek AT Cummings. Case study in evaluating time series prediction models using the relative mean absolute error. *The American Statistician*, 70(3):285–292, 2016.

[157] Arnaud De Myttenaere, Boris Golden, Bénédicte Le Grand, and Fabrice Rossi. Mean absolute percentage error for regression models. *Neurocomputing*, 192:38–48, 2016.

[158] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[159] Terrence S Furey, Nello Cristianini, Nigel Duffy, David W Bednarski, Michel Schummer, and David Haussler. Support vector machine classification and

validation of cancer tissue samples using microarray expression data. *Bioinformatics*, 16(10):906–914, 2000.

[160] Simon Tong and Daphne Koller. Support vector machine active learning with applications to text classification. *Journal of machine learning research*, 2(Nov):45–66, 2001.

[161] Patrick Rebentrost, Masoud Mohseni, and Seth Lloyd. Quantum support vector machine for big data classification. *Physical review letters*, 113(13):130503, 2014.

[162] Vladimir Vapnik and Rauf Izmailov. Knowledge transfer in svm and neural networks. *Annals of Mathematics and Artificial Intelligence*, 81(1-2):3–19, 2017.

[163] Cyc. Graphic showing the maximum separating hyperplane and the margin. https://commons.wikimedia.org/wiki/File:Svm_max_sep_hyperplane_with_margin.png, 2008. Accessed: 2017-07-28.

[164] Johan AK Suykens and Joos Vandewalle. Least squares support vector machine classifiers. *Neural processing letters*, 9(3):293–300, 1999.

[165] Guang-Bin Huang, Hongming Zhou, Xiaojian Ding, and Rui Zhang. Extreme learning machine for regression and multiclass classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 42(2):513–529, 2012.

[166] Claudio Gentile and Manfred K Warmuth. Linear hinge loss and average margin. In *Advances in Neural Information Processing Systems*, pages 225–231, 1999.

[167] Yichao Wu and Yufeng Liu. Robust truncated hinge loss support vector machines. *Journal of the American Statistical Association*, 102(479):974–983, 2007.

[168] Junqi Jin, Kun Fu, and Changshui Zhang. Traffic sign recognition with hinge loss trained convolutional neural networks. *IEEE Transactions on Intelligent Transportation Systems*, 15(5):1991–2000, 2014.

[169] Justin Lovinger. Learning. https://github.com/JustinLovinger/Learning, 2017. Accessed: 2018-04-20.

[170] Justin Lovinger. Optimal. https://github.com/JustinLovinger/Optimal, 2014. Accessed: 2018-05-01.

[171] M. Lichman. UCI machine learning repository. http://archive.ics.uci.edu/ml, 2013. Accessed: 2015-12-08.

[172] R. Kelley Pace. California housing. http://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html, 1997. Accessed: 2015-12-08.

[173] R Kelley Pace and Ronald Barry. Sparse spatial autoregressions. *Statistics & Probability Letters*, 33(3):291–297, 1997.

[174] Guang-Bin Huang, Paramasivan Saratchandran, and Narasimhan Sundararajan. A generalized growing and pruning rbf (ggap-rbf) neural network for function approximation. *IEEE Transactions on Neural Networks*, 16(1):57–67, 2005.

[175] Guang-Bin Huang, Lei Chen, Chee Kheong Siew, et al. Universal approximation using incremental constructive feedforward networks with random hidden nodes. *IEEE Trans. Neural Networks*, 17(4):879–892, 2006.

[176] Jonathan J. Hull. A database for handwritten text recognition research. *IEEE Transactions on pattern analysis and machine intelligence*, 16(5):550–554, 1994.

[177] Terence Sim, Simon Baker, and Maan Bsat. The cmu pose, illumination, and expression (pie) database. In *Automatic Face and Gesture Recognition, 2002. Proceedings. Fifth IEEE International Conference on*, pages 53–58. IEEE, 2002.

[178] Hugo Steinhaus. Sur la division des corp materiels en parties. *Bull. Acad. Polon. Sci*, 1(804):801, 1956.

[179] Edward W Forgy. Cluster analysis of multivariate data: efficiency versus interpretability of classifications. *Biometrics*, 21:768–769, 1965.

[180] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967.

[181] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[182] David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[183] Xiaoli Cui, Pingfei Zhu, Xin Yang, Keqiu Li, and Changqing Ji. Optimized big data k-means clustering using mapreduce. *The Journal of Supercomputing*, 70(3):1249–1259, 2014.

[184] Nameirakpam Dhanachandra, Khumanthem Manglem, and Yambem Jina Chanu. Image segmentation using k-means clustering algorithm and subtractive clustering algorithm. *Procedia Computer Science*, 54(2015):764–771, 2015.

[185] Michael B Cohen, Sam Elder, Cameron Musco, Christopher Musco, and Madalina Persu. Dimensionality reduction for k-means clustering and low rank

approximation. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*, pages 163–172. ACM, 2015.