arXiv:2403.14606v1 [cs.LG] 21 Mar 2024

The Elements of Differentiable Programming

Mathieu Blondel
Google DeepMind
mblondel@google.com

Vincent Roulet
Google DeepMind
vroulet@google.com

Draft (last update: March 21, 2024)

Contents

1 Introduction **4**
1.1 What is differentiable programming? 4
1.2 Book goals and scope 6
1.3 Intended audience 7
1.4 How to read this book? 7
1.5 Related work 7

I Fundamentals **9**

2 Differentiation **10**
2.1 Univariate functions 10
  2.1.1 Derivatives 10
  2.1.2 Calculus rules 13
  2.1.3 Leibniz's notation 15
2.2 Multivariate functions 16
  2.2.1 Directional derivatives 16
  2.2.2 Gradients 17
  2.2.3 Jacobians 20
2.3 Linear differentiation maps 26
  2.3.1 The need for linear maps 26
  2.3.2 Euclidean spaces 27

2.3.3 Linear maps and their adjoints . . . 28
2.3.4 Jacobian-vector products . . . 29
2.3.5 Vector-Jacobian products . . . 30
2.3.6 Chain rule . . . 31
2.3.7 Functions of multiple inputs (fan-in) . . . 32
2.3.8 Functions of multiple outputs (fan-out) . . . 34
2.3.9 Extensions to non-Euclidean linear spaces . . . 34
2.4 Second-order differentiation . . . 36
2.4.1 Second derivatives . . . 36
2.4.2 Second directional derivatives . . . 36
2.4.3 Hessians . . . 37
2.4.4 Hessian-vector products . . . 39
2.4.5 Second-order Jacobians . . . 39
2.5 Higher-order differentiation . . . 40
2.5.1 Higher-order derivatives . . . 40
2.5.2 Higher-order directional derivatives . . . 41
2.5.3 Higher-order Jacobians . . . 41
2.5.4 Taylor expansions . . . 42
2.6 Differential geometry . . . 43
2.6.1 Differentiability on manifolds . . . 43
2.6.2 Tangent spaces and pushforward operators . . . 44
2.6.3 Cotangent spaces and pullback operators . . . 45
2.7 Generalized derivatives . . . 48
2.7.1 Rademacher's theorem . . . 49
2.7.2 Clarke derivatives . . . 49
2.8 Summary . . . 52

3 Probabilistic learning **54**
3.1 Probability distributions . . . 54
3.1.1 Discrete probability distributions . . . 54
3.1.2 Continuous probability distributions . . . 55
3.2 Maximum likelihood estimation . . . 56
3.2.1 Negative log-likelihood . . . 56
3.2.2 Consistency w.r.t. the Kullback-Leibler divergence . 56
3.3 Probabilistic supervised learning . . . 57
3.3.1 Conditional probability distributions . . . 57

3.3.2 Inference 57
3.3.3 Binary classification 58
3.3.4 Multiclass classification 60
3.3.5 Regression 61
3.3.6 Multivariate regression 62
3.3.7 Integer regression 63
3.3.8 Loss functions 63
3.4 Exponential family distributions 65
3.4.1 Definition 65
3.4.2 The log-partition function 67
3.4.3 Maximum entropy principle 68
3.4.4 Maximum likelihood estimation 69
3.4.5 Probabilistic learning with exponential families . . . 69
3.5 Summary 71

II Differentiable programs 72

4 Parameterized programs 73
4.1 Representing computer programs 73
4.1.1 Computation chains 73
4.1.2 Directed acylic graphs 74
4.1.3 Computer programs as DAGs 76
4.1.4 Arithmetic circuits 78
4.2 Feedforward networks 79
4.3 Multilayer perceptrons 79
4.3.1 Combining affine layers and activations 79
4.3.2 Link with generalized linear models 80
4.4 Activation functions 81
4.4.1 Scalar-to-scalar nonlinearities 81
4.4.2 Vector-to-scalar nonlinearities 81
4.4.3 Scalar-to-scalar probability mappings 82
4.4.4 Vector-to-vector probability mappings 83
4.5 Residual neural networks 85
4.6 Recurrent neural networks 86
4.6.1 Vector to sequence 86

4.6.2 Sequence to vector 88
4.6.3 Sequence to sequence (aligned) 88
4.6.4 Sequence to sequence (unaligned) 88
4.7 Summary 89

5 Control flows **90**
5.1 Comparison operators 90
5.2 Soft inequality operators 92
5.2.1 Heuristic definition 92
5.2.2 Stochastic process perspective 92
5.3 Soft equality operators 93
5.3.1 Heuristic definition 93
5.3.2 Gaussian process perspective 94
5.4 Logical operators 95
5.5 Continuous extensions of logical operators 96
5.5.1 Probabilistic continuous extension 96
5.5.2 Triangular norms and co-norms 98
5.6 If-else statements 98
5.6.1 Differentiating through branch variables 99
5.6.2 Differentiating through predicate variables 100
5.6.3 Continuous relaxations 101
5.7 Else-if statements 102
5.7.1 Encoding K branches 103
5.7.2 Conditionals 104
5.7.3 Differentiating through branch variables 105
5.7.4 Differentiating through predicate variables 106
5.7.5 Continuous relaxations 106
5.8 For loops 108
5.9 Scan functions 109
5.10 While loops 110
5.10.1 While loops as cyclic graphs 110
5.10.2 Unrolled while loops 111
5.10.3 Markov chain perspective 113
5.11 Summary 116

III Differentiating through programs 117

6 Finite differences 118
6.1 Forward differences 118
6.2 Backward differences 119
6.3 Central differences 120
6.4 Higher-accuracy finite differences 121
6.5 Higher-order finite differences 122
6.6 Complex-step derivatives 123
6.7 Complexity 124
6.8 Summary 124

7 Automatic differentiation 126
7.1 Computation chains 126
7.1.1 Forward-mode 127
7.1.2 Reverse-mode 129
7.1.3 Complexity of entire Jacobians 134
7.2 Feedforward networks 136
7.2.1 Computing the adjoint 136
7.2.2 Computing the gradient 137
7.3 Computation graphs 139
7.3.1 Forward-mode 139
7.3.2 Reverse-mode 140
7.3.3 Complexity, the Baur-Strassen theorem 140
7.4 Implementation 141
7.4.1 Primitive functions 141
7.4.2 Closure under function composition 142
7.4.3 Examples of JVPs and VJPs 143
7.4.4 Automatic linear transposition 144
7.5 Checkpointing 145
7.5.1 Recursive halving 146
7.5.2 Dynamic programming 148
7.5.3 Online checkpointing 150
7.6 Reversible layers 151
7.6.1 General case 151
7.6.2 Case of orthonormal JVPs 151

7.7 Randomized forward-mode estimator . . . 152
7.8 Summary . . . 152

8 Second-order automatic differentiation 154
8.1 Hessian-vector products . . . 154
  8.1.1 Four possible methods . . . 154
  8.1.2 Complexity . . . 155
8.2 Gauss-Newton matrix . . . 159
  8.2.1 An approximation of the Hessian . . . 159
  8.2.2 Gauss-Newton chain rule . . . 160
  8.2.3 Gauss-Newton vector product . . . 160
  8.2.4 Gauss-Newton matrix factorization . . . 161
  8.2.5 Stochastic setting . . . 162
8.3 Fisher information matrix . . . 162
  8.3.1 Definition using the score function . . . 162
  8.3.2 Link with the Hessian . . . 163
  8.3.3 Equivalence with the Gauss-Newton matrix . . . 163
8.4 Inverse-Hessian vector product . . . 165
  8.4.1 Definition as a linear map . . . 165
  8.4.2 Implementation with matrix-free linear solvers . . . 165
  8.4.3 Complexity . . . 166
8.5 Second-order backpropagation . . . 167
  8.5.1 Second-order Jacobian chain rule . . . 167
  8.5.2 Computation chains . . . 169
  8.5.3 Fan-in and fan-out . . . 170
8.6 Block diagonal approximations . . . 171
  8.6.1 Feedforward networks . . . 171
  8.6.2 Computation graphs . . . 173
8.7 Diagonal approximations . . . 173
  8.7.1 Computation chains . . . 174
  8.7.2 Computation graphs . . . 175
8.8 Randomized estimators . . . 176
  8.8.1 Girard-Hutchinson estimator . . . 176
  8.8.2 Bartlett estimator for the factorization . . . 177
  8.8.3 Bartlett estimator for the diagonal . . . 178
8.9 Summary . . . 179

9 Inference in graphical models as differentiation **180**
9.1 Chain rule of probability 180
9.2 Conditional independence 181
9.3 Inference problems 182
9.3.1 Joint probability distributions 182
9.3.2 Likelihood 182
9.3.3 Maximum a-posteriori inference 182
9.3.4 Marginal inference 183
9.3.5 Expectation, convex hull, marginal polytope 183
9.3.6 Complexity of brute force 185
9.4 Markov chains 185
9.4.1 The Markov property 186
9.4.2 Time-homogeneous Markov chains 188
9.4.3 Higher-order Markov chains 189
9.5 Bayesian networks 189
9.5.1 Expressing variable dependencies using DAGs . . . 189
9.5.2 Parameterizing Bayesian networks 190
9.5.3 Ancestral sampling 191
9.6 Markov random fields 191
9.6.1 Expressing factors using undirected graphs 191
9.6.2 MRFs as exponential family distributions 192
9.6.3 Conditional random fields 194
9.6.4 Sampling 194
9.7 Inference on chains 194
9.7.1 The forward-backward algorithm 195
9.7.2 The Viterbi algorithm 196
9.8 Inference on trees 198
9.9 Inference as differentiation 199
9.9.1 Inference as gradient of the log-partition 199
9.9.2 Semirings and softmax operators 200
9.9.3 Inference as backpropagation 202
9.10 Summary 204

10 Differentiating through optimization **205**
10.1 Implicit functions 205
10.1.1 Optimization problems 206

10.1.2 Nonlinear equations 206
10.1.3 Application to bilevel optimization 206
10.2 Envelope theorems 207
10.2.1 Danskin's theorem 208
10.2.2 Rockafellar's theorem 209
10.3 Implicit function theorem 210
10.3.1 Univariate functions 210
10.3.2 Multivariate functions 211
10.3.3 JVP and VJP of implicit functions 213
10.3.4 Proof of the implicit function theorem 214
10.4 Adjoint state method 214
10.4.1 Differentiating nonlinear equations 214
10.4.2 Relation with envelope theorems 216
10.4.3 Proof using the method of Lagrange multipliers . . 216
10.4.4 Proof using the implicit function theorem 217
10.4.5 Reverse mode as adjoint method with backsubstitution 217
10.5 Inverse function theorem 220
10.5.1 Differentiating inverse functions 220
10.5.2 Link with the implicit function theorem 220
10.5.3 Proof of inverse function theorem 221
10.6 Summary 222

11 Differentiating through integration **224**
11.1 Differentiation under the integral sign 224
11.2 Differentiating through expectations 225
11.2.1 The easy case 226
11.2.2 Exact gradients 226
11.2.3 Application to expected loss functions 227
11.2.4 Application to experimental design 228
11.3 Score function estimators, REINFORCE 229
11.3.1 Scalar-valued functions 229
11.3.2 Variance reduction 231
11.3.3 Vector-valued functions 233
11.3.4 Second derivatives 234
11.4 Path gradient estimators, reparametrization trick 235
11.4.1 Location-scale transforms 235

11.4.2 Inverse transforms . . . 236
11.4.3 Pushforward operators . . . 238
11.4.4 Change-of-variables theorem . . . 240
11.5 Stochastic programs . . . 240
11.5.1 Stochastic computation graphs . . . 241
11.5.2 Examples . . . 243
11.5.3 Unbiased gradient estimators . . . 245
11.5.4 Local vs. global expectations . . . 247
11.6 Differential equations . . . 248
11.6.1 Parameterized differential equations . . . 248
11.6.2 Continuous adjoint method . . . 251
11.6.3 Gradients via the continuous adjoint method . . . 252
11.6.4 Gradients via reverse-mode on discretization . . . 254
11.6.5 Reversible discretization schemes . . . 255
11.6.6 Proof of the continuous adjoint method . . . 257
11.7 Summary . . . 259

IV Smoothing programs **261**

12 Smoothing by optimization **262**
12.1 Primal approach . . . 262
12.1.1 Infimal convolution . . . 262
12.1.2 Moreau envelope . . . 263
12.2 Legendre–Fenchel transforms, convex conjugates . . . 265
12.2.1 Definition . . . 265
12.2.2 Closed-form examples . . . 266
12.2.3 Properties . . . 267
12.2.4 Conjugate calculus . . . 269
12.2.5 Fast Legendre transform . . . 270
12.3 Dual approach . . . 270
12.3.1 Duality between strong convexity and smoothness . 270
12.3.2 Smoothing by dual regularization . . . 271
12.3.3 Equivalence between primal and dual regularizations 273
12.4 Examples . . . 273
12.4.1 Smoothed ReLU functions . . . 273

12.4.2 Smoothed max operators . . . 274
12.4.3 Relaxed step functions (sigmoids) . . . 276
12.4.4 Relaxed argmax operators . . . 277
12.5 Summary . . . 278

13 Smoothing by integration **279**
13.1 Convolution . . . 279
13.1.1 Convolution operators . . . 279
13.1.2 Convolution with a kernel . . . 280
13.1.3 Discrete convolution . . . 281
13.1.4 Differentiation . . . 283
13.1.5 Multidimensional convolution . . . 283
13.1.6 Link between convolution and infimal convolution . 283
13.2 Fourier and Laplace transforms . . . 284
13.2.1 Convolution theorem . . . 284
13.2.2 Link between Fourier and Legendre transforms . . . 285
13.2.3 The soft Legendre-Fenchel transform . . . 285
13.3 Examples . . . 289
13.3.1 Smoothed step function . . . 289
13.3.2 Smoothed ReLU function . . . 290
13.4 Perturbation of blackbox functions . . . 291
13.4.1 Expectation in a location-scale family . . . 291
13.4.2 Gradient estimation by reparametrization . . . 292
13.4.3 Gradient estimation by SFE, Stein's lemma . . . 293
13.4.4 Link between reparametrization and SFE . . . 294
13.4.5 Variance reduction and evolution strategies . . . 295
13.4.6 Zero-temperature limit . . . 296
13.5 Gumbel tricks . . . 296
13.5.1 The Gumbel distribution . . . 296
13.5.2 Perturbed comparison . . . 297
13.5.3 Perturbed argmax . . . 298
13.5.4 Perturbed max . . . 300
13.5.5 Gumbel trick for sampling . . . 301
13.5.6 Perturb-and-MAP . . . 301
13.5.7 Gumbel-softmax . . . 303
13.6 Summary . . . 304

V Optimizing differentiable programs **306**

14 Optimization basics **307**
- 14.1 Objective functions . . . 307
- 14.2 Oracles . . . 308
- 14.3 Variational perspective of optimization algorithms . . . 309
- 14.4 Classes of functions . . . 309
  - 14.4.1 Lipschitz functions . . . 309
  - 14.4.2 Smooth functions . . . 310
  - 14.4.3 Convex functions . . . 312
  - 14.4.4 Strongly-convex functions . . . 314
  - 14.4.5 Nonconvex functions . . . 315
- 14.5 Performance guarantees . . . 316
- 14.6 Summary . . . 319

15 First-order optimization **320**
- 15.1 Gradient descent . . . 320
  - 15.1.1 Variational perspective . . . 320
  - 15.1.2 Convergence for smooth functions . . . 321
  - 15.1.3 Momentum and accelerated variants . . . 323
- 15.2 Stochastic gradient descent . . . 323
  - 15.2.1 Stochastic gradients . . . 324
  - 15.2.2 Vanilla SGD . . . 325
  - 15.2.3 Momentum variants . . . 326
  - 15.2.4 Adaptive variants . . . 327
- 15.3 Projected gradient descent . . . 328
  - 15.3.1 Variational perspective . . . 328
  - 15.3.2 Optimality conditions . . . 329
  - 15.3.3 Commonly-used projections . . . 329
- 15.4 Proximal gradient method . . . 330
  - 15.4.1 Variational perspective . . . 331
  - 15.4.2 Optimality conditions . . . 331
  - 15.4.3 Commonly-used proximal operators . . . 332
- 15.5 Summary . . . 333

16 Second-order optimization **334**
16.1 Newton's method 334
16.1.1 Variational perspective 334
16.1.2 Regularized Newton method 335
16.1.3 Approximate direction 336
16.1.4 Convergence guarantees 336
16.1.5 Linesearch 336
16.1.6 Geometric interpretation 337
16.1.7 Stochastic Newton's method 338
16.2 Gauss-Newton method 339
16.2.1 With exact outer function 340
16.2.2 With approximate outer function 341
16.2.3 Linesearch 342
16.2.4 Stochastic Gauss-Newton 342
16.3 Natural gradient descent 343
16.3.1 Variational perspective 343
16.3.2 Stochastic natural gradient descent 344
16.4 Quasi-Newton methods 345
16.4.1 BFGS 345
16.4.2 Limited-memory BFGS 346
16.5 Approximate Hessian diagonal inverse preconditionners . . 346
16.6 Summary 346

17 Duality **348**
17.1 Dual norms 348
17.2 Fenchel duality 349
17.3 Bregman divergences 352
17.4 Fenchel-Young loss functions 355
17.5 Summary 356

References **357**

The Elements of Differentiable Programming

Mathieu Blondel[1] and Vincent Roulet[1]

[1]*Google DeepMind*

ABSTRACT

Artificial intelligence has recently experienced remarkable advances, fueled by large models, vast datasets, accelerated hardware, and, last but not least, the transformative power of differentiable programming. This new programming paradigm enables end-to-end differentiation of complex computer programs (including those with control flows and data structures), making gradient-based optimization of program parameters possible.

As an emerging paradigm, differentiable programming builds upon several areas of computer science and applied mathematics, including automatic differentiation, graphical models, optimization and statistics. This book presents a comprehensive review of the fundamental concepts useful for differentiable programming. We adopt two main perspectives, that of optimization and that of probability, with clear analogies between the two.

Differentiable programming is not merely the differentiation of programs, but also the thoughtful design of programs intended for differentiation. By making programs differentiable, we inherently introduce probability distributions over their execution, providing a means to quantify the uncertainty associated with program outputs.

Notation

Table 1: Naming conventions

Notation	Description
$\mathcal{X} \subseteq \mathbb{R}^D$	Input space (e.g., features)
$\mathcal{Y} \subseteq \mathbb{R}^M$	Output space (e.g., classes)
$\mathcal{S}_k \subseteq \mathbb{R}^{D_k}$	Output space on layer or state k
$\mathcal{W} \subseteq \mathbb{R}^P$	Weight space
$\Lambda \subseteq \mathbb{R}^Q$	Hyperparameter space
$\Theta \subseteq \mathbb{R}^R$	Distribution parameter space, logit space
N	Number of training samples
T	Number of optimization iterations
$\boldsymbol{x} \in \mathcal{X}$	Input vector
$\boldsymbol{y} \in \mathcal{Y}$	Target vector
$\boldsymbol{s}_k \in \mathcal{S}_k$	State vector k
$\boldsymbol{w} \in \mathcal{W}$	Network (model) weights
$\boldsymbol{\lambda} \in \Lambda$	Hyperparameters
$\boldsymbol{\theta} \in \Theta$	Distribution parameters, logits
$\pi \in [0, 1]$	Probability value
$\boldsymbol{\pi} \in \triangle^M$	Probability vector

Table 2: Naming conventions (continued)

Notation	Description
f	Network function
$f(\cdot;\boldsymbol{x})$	Network function with $\boldsymbol{x}$ fixed
L	Objective function
ℓ	Loss function
κ	Kernel function
ϕ	Output embedding, sufficient statistic
step	Heaviside step function
logistic_σ	Logistic function with temperature σ
logistic	Shorthand for logistic_1
$p_{\boldsymbol{\theta}}$	Model distribution with parameters $\boldsymbol{\theta}$
ρ	Data distribution over $\mathcal{X} \times \mathcal{Y}$
$\rho_{\mathcal{X}}$	Data distribution over $\mathcal{X}$
$\boldsymbol{\mu}, \sigma^2$	Mean and variance
Z	Random noise variable

1

Introduction

1.1 What is differentiable programming?

A computer program is a sequence of elementary instructions for performing a task. In traditional programming, the program is typically hand-written by a programmer. However, for certain tasks, such as image recognition or text generation, hand-writing a program to perform such tasks is nearly impossible.

This has motivated the need for statistical approaches based on machine learning. With differentiable programming, while the overall structure of the program is typically designed by a human, parameters of the program (such as weights in a neural network) can be automatically adjusted to achieve a task or optimize a criterion. This paradigm has also been referred to as “software 2.0”. We give an informal definition.

> **Definition 1.1** (Differentiable programming)**.** Differentiable programming is a programming paradigm in which complex computer programs (including those with control flows and data structures) can be differentiated end-to-end automatically, enabling gradient-based optimization of parameters in the program.

In differentiable programming, a program is also defined as the composition of elementary operations, forming a **computation graph**. The key difference with classical computer programming is that the program can be differentiated end-to-end, using **automatic differentiation** (autodiff). Typically, it is assumed that the program defines a **mathematically valid function** (a.k.a. pure function): the function should return identical values for identical arguments and should not have any side effects. Moreover, the function should have **well-defined derivatives**, ensuring that it can be used in a gradient-based optimization algorithm. Therefore, differentiable programming is not only the art of differentiating through programs but also of **designing** meaningful differentiable programs.

Why are derivatives important?

Machine learning typically boils down to optimizing a certain objective function, which is the composition of a loss function and a model (network) function. Derivative-free optimization is called **zero-order optimization**. It only assumes that we can evaluate the objective function that we wish to optimize. Unfortunately, it is known to suffer from the **curse of dimensionality**, i.e., it only scales to small dimensional problems, such as less than 10 dimensions. Derivative-based optimization, on the other hand, is much more efficient and can scale to millions or billions of parameters. Algorithms that use first and second derivatives are known as **first-order** and **second-order** algorithms, respectively.

Why is autodiff important?

Before the autodiff revolution, researchers and practitioners needed to manually implement the gradient of the functions they wished to optimize. Manually deriving gradients can become very tedious for complicated functions. Moreover, every time the function is changed (for example, for trying out a new idea), the gradient needs to be re-derived. Autodiff is a game changer because it allows users to focus on quickly and creatively experimenting with functions for their tasks.

Differentiable programming is not just deep learning

While there is clearly overlap between deep learning and differentiable programming, their focus is different. Deep learning studies artificial neural networks composed of multiple layers, able to learn **intermediate representations** of the data. Neural network architectures have been proposed with various **inductive biases**. For example, convolutional neural networks are designed for images and transformers are designed for sequences. On the other hand, differentiable programming studies the techniques to differentiate through complex programs. It is useful beyond deep learning: for instance in reinforcement learning, probabilistic programming and scientific computing in general.

Differentiable programming is not just autodiff

While autodiff is a key ingredient of differentiable programming, this is not the only one. Differentiable programming is also concerned with the design of principled differentiable operations. In fact, much research on differentiable programming has been devoted to make classifical computer programming operations compatible with autodiff. As we shall see, many differentiable relaxations can be interpreted in a probabilistic framework. A core theme of this book is the interplay between optimization, probability and differentiation. Differentiation is useful for optimization and conversely, optimization can be used to design differentiable operators.

1.2 Book goals and scope

The present book aims to provide an comprehensive introduction to differentiable programming with an emphasis on **core mathematical tools**.

- In Part I, we review **fundamentals**: differentiation and probabilistic learning.

- In Part II, we review **differentiable programs**. This includes neural networks, sequence networks and control flows.

- In Part III, we review how to **differentiate through programs**. This includes automatic differentiation, but also differentiating through optimization and integration (in particular, expectations).
- In Part IV, we review **smoothing programs**. We focus on two main techniques: infimal convolution, which comes from the world of optimization and convolution, which comes from the world of integration. We also strive to spell out the connections between them.
- In Part V, we review **optimizing programs**: basic optimization concepts, first-order algorithms, second-order-algorithms and duality.

Our goal is to present the fundamental techniques useful for differentiable programming, **not** to survey how these techniques have been used in various applications.

1.3 Intended audience

This book is intended to be a graduate-level introduction to differentiable programming. Our pedagogical choices are made with the machine learning community in mind. Some familiarity with calculus, linear algebra, probability theory and machine learning is beneficial.

1.4 How to read this book?

This book does not need to be read linearly chapter by chapter. When needed, we indicate at the beginning of a chapter what chapters are recommended to be read as a prerequisite.

1.5 Related work

Differentiable programming builds upon a variety of connected topics. We review in this section relevant textbooks, tutorials and software.

Standard textbooks on backpropagation and automatic differentiation are that of Werbos (1994) and Griewank and Walther (2008).

A tutorial with a focus on machine learning is provided by Baydin *et al.* (2018). Automatic differentiation is also reviewed as part of more general textbooks, such as those of Deisenroth *et al.* (2020), Murphy (2022) (from a linear algebra perspective) and Murphy (2023) (from a functional perspective; autodiff section authored by Roy Frostig). The present book was also influenced by Peyré (2020)'s textbook on data science. The history of reverse-mode autodiff is reviewed by Griewank (2012).

A tutorial on different perspectives of backpropagation is "There and Back Again: A Tale of Slopes and Expectations" (link), by Deisenroth and Ong. A tutorial on implicit differentiation is "Deep Implicit Layers - Neural ODEs, Deep Equilibirum Models, and Beyond" (link), by Kolter, Duvenaud, and Johnson.

The standard reference on inference in graphical models and its connection with exponential families is that of Wainwright and Jordan (2008). Differential programming is also related to probabilistic programming; see, e.g., Meent *et al.* (2018).

A review of smoothing from the infimal convolution perspective is provided by Beck and Teboulle (2012). A standard textbook on convex optimization is that of Nesterov (2018). A textbook on first-order optimization methods is that of Beck (2017).

Autodiff implementations that accelerated the autodiff revolution in machine learning are Theano (Bergstra *et al.*, 2010) and Autograd (Maclaurin *et al.*, 2015). Major modern implementations of autodiff include Tensorflow (Abadi *et al.*, 2016), JAX (Bradbury *et al.*, 2018), and PyTorch (Paszke *et al.*, 2019). We in particular acknowledge the JAX team for influencing our view of autodiff.

Part I

Fundamentals

2

Differentiation

In this chapter, we review key differentiation concepts. In particular, we emphasize on the fundamental role played by linear maps.

2.1 Univariate functions

2.1.1 Derivatives

Before studying derivatives, we briefly recall the definition of function **continuity**.

> **Definition 2.1** (Continuous function)**.** A function $f : \mathbb{R} \to \mathbb{R}$ is continuous at a point $w \in \mathbb{R}$ if
>
> $$\lim_{v \to w} f(v) = f(w).$$
>
> A function f is said to be continuous if it is continuous at all points in its domain.

In the following, we use Landau's little o notation. We write

$$g(v) = o(f(v)) \text{ as } v \to w$$

if

$$\lim_{v \to w} \frac{|g(v)|}{|f(v)|} = 0.$$

That is, the function f dominates g in the limit $v \to w$. For example, f is continuous at w if and only if

$$f(w + \delta) = f(w) + o(1) \text{ as } \delta \to 0.$$

We now explain derivatives. Consider a function $f : \mathbb{R} \to \mathbb{R}$. As illustrated in Fig. 2.1, its value on an interval $[w_0, w_0 + \delta]$ can be approximated by the secant between its values $f(w_0)$ and $f(w_0 + \delta)$, a linear function with slope $(f(w_0 + \delta) - f(w_0))/\delta$. In the limit of an infinitesimal variation δ around w_0, the secant converges to the **tangent** of f at w_0 and the resulting slope defines the derivative of f at w_0. The definition below formalizes this intuition.

Definition 2.2 (Derivative)**.** The **derivative** of $f : \mathbb{R} \to \mathbb{R}$ at $w \in \mathbb{R}$ is defined as

$$f'(w) := \lim_{\delta \to 0} \frac{f(w + \delta) - f(w)}{\delta}, \tag{2.1}$$

provided that the limit exists. If $f'(w)$ is well-defined at a particular w, we say that the function f is **differentiable** at w.

Here, and in the following definitions, if f is differentiable at any $w \in \mathbb{R}$, we say that it is **differentiable everywhere** or differentiable for short. If f is differentiable at a given w, then it is necessarily **continuous** at w.

Proposition 2.1 (Differentiability implies continuity)**.** If $f : \mathbb{R} \to \mathbb{R}$ is differentiable at $w \in \mathbb{R}$, then it is continuous at $w \in \mathbb{R}$.

Proof. In little o notation, f is differentiable at w if there exists $f'(w) \in \mathbb{R}$, such that

$$f(w + \delta) = f(w) + f'(w)\delta + o(\delta) \text{ as } \delta \to 0.$$

Since $f'(w)\delta + o(\delta) = o(1)$ as $\delta \to 0$, f is continuous at w. ☐

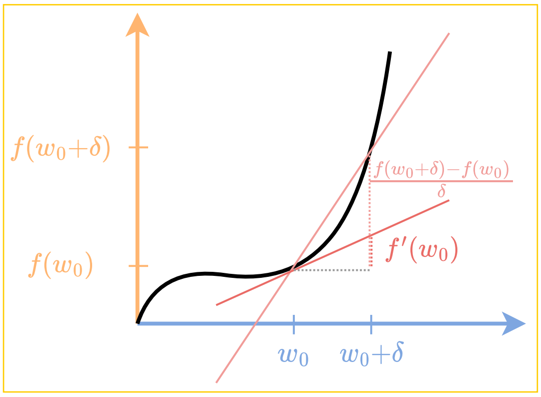

Figure 2.1: A function f can be locally approximated around a point w_0 by a secant, a linear function $w \mapsto aw+b$ with slope a and intercept b, crossing f at w_0 with value $y_0 = f(w_0)$ and crossing at $w_0 + \delta$ with value $u_\delta = f(w_0 + \delta)$. Using $u_0 = aw_0 + b$ and $u_\delta = a(w_0 + \delta) + b$, we find that its slope is $a = (f(w_0 + \delta) - f(w_0))/\delta$ and the intercept is $b = f(w_0) - aw_0$. The derivative $f'(w)$ of a function f at a point w_0 is then defined as the limit of the slope a when $\delta \to 0$. It is the slope of the tangent of f at w_0. The value $f(w)$ of the function at w can then be locally approximated around w_0 by $w \mapsto f'(w_0)w + f(w_0) - f'(w_0)w_0 = f(w_0) + f'(w_0)(w - w_0)$.

In addition to enabling the construction of a linear approximation of f in a neighborhood of w, since it is the slope of the tangent of f at w, the derivative f' informs us about the **monotonicity** of f around w. If $f'(w)$ is positive, the function is increasing around w. Conversely, if $f'(w)$ is negative, the function is decreasing. Such information can be used to develop iterative algorithms seeking to minimize f by computing iterates of the form $w_{t+1} = w_t - \gamma f'(w_t)$ for $\gamma > 0$, which move along descent directions of f around w_t.

For several elementary functions such as w^n, e^w, $\ln w$, $\cos w$ or $\sin w$, their derivatives can be obtained directly by applying the definition of the derivative in Eq. (2.1) as illustrated in Example 2.1.

Example 2.1 (Derivative of power function). Consider $f(w) = w^n$

for $w \in \mathbb{R}$, $n \in \mathbb{N} \setminus \{0\}$. For any $\delta \in \mathbb{R}$, we have

$$\begin{aligned}\frac{f(w+\delta)-f(w)}{\delta} &= \frac{(w+\delta)^n - w^n}{\delta}\\ &= \frac{\sum_{k=0}^{n}\binom{n}{k}\delta^k w^{n-k} - w^n}{\delta}\\ &= \sum_{k=1}^{n}\binom{n}{k}\delta^{k-1}w^{n-k}\\ &= \binom{n}{1}w^{n-1} + \sum_{k=2}^{n}\binom{n}{k}\delta^{k-1}w^{n-k},\end{aligned}$$

where, in the second line, we used the binomial theorem. Since $\binom{n}{1} = n$ and $\lim_{\delta\to 0}\sum_{k=2}^{n}\binom{n}{k}\delta^{k-1}w^{n-k} = 0$, we get $f'(w) = nw^{n-1}$.

Remark 2.1 (Functions on a subset $\mathcal{U}$ of $\mathbb{R}$)**.** For simplicity, we presented the definition of the derivative for a function defined on the whole set of real numbers $\mathbb{R}$. If a function $f : \mathcal{U} \to \mathbb{R}$ is defined on a subset $\mathcal{U} \subseteq \mathbb{R}$ of the real numbers, as it is the case for $f(w) = \sqrt{w}$ defined on $\mathcal{U} = \mathbb{R}_+$, the derivative of f at $w \in \mathcal{U}$ is defined by the limit in (2.1) provided that the function f is well defined on a neighborhood of w, that is, there exists $r > 0$ such that $w + \delta \in \mathcal{U}$ for any $|\delta| \leq r$. The function f is then said **differentiable everywhere** or differentiable for short if it is differentiable at any point w in the **interior** of $\mathcal{U}$, the set of points $w \in \mathcal{U}$ such that $\{w+\delta : |\delta| \leq r\} \subseteq \mathcal{U}$ for r sufficiently small. For points lying at the boundary of $\mathcal{U}$ (such as a and b if $\mathcal{U} = [a, b]$), one may define the right and left derivatives of f at a and b, meaning that the limit is taken by approaching a from the right or b from the left.

2.1.2 Calculus rules

For a given $w \in \mathbb{R}$ and two functions $f : \mathbb{R} \to \mathbb{R}$ and $g : \mathbb{R} \to \mathbb{R}$, the derivative of elementary operations on f and g such as their sums, products or compositions can easily be derived from the definition of the derivative, under appropriate conditions on the differentiability properties of f and g at w. For example, if the derivative of f and g

exist at w then the derivative of their weighted sum or the derivatives or their products exist and lead to the following rules

$$\forall a, b \in \mathbb{R},\ (af + bg)'(w) = af'(w) + bg'(w) \qquad \text{(Linearity)}$$
$$(fg)'(w) = f'(w)g(w) + f(w)g'(w), \qquad \text{(Product rule)}$$

where $(fg)(w) = f(w)g(w)$. The linearity can be verified directly from the linearity of the limits. For the product rule, in little o notation, we have, as $\delta \to 0$,

$$\begin{aligned}(fg)(w+\delta) &= (f(w) + f'(w)\delta + o(\delta))(g(w) + g'(w)\delta + o(\delta)) \\ &= f(w)g(w) + f'(w)g(w)\delta + f(w)g'(w)\delta + o(\delta),\end{aligned}$$

hence the result.

If the derivatives of g at w and of f at $g(w)$ exist, then the derivative of the composition $(f \circ g)(w) := f(g(w))$ at w exist and is given by

$$(f \circ g)'(w) = f'(g(w))g'(w). \qquad \text{(Chain rule)}$$

We prove this result more generally in Proposition 2.2. As seen in the sequel, the linearity and the product rule can be seen as byproducts of the chain rule, making the chain rule the cornerstone of differentiation.

For now, consider a function that can be expressed as sums, products or compositions of elementary functions such as $f(w) = e^w \ln w + \cos w^2$. Its derivative can be computed by applying the aforementioned rules on the decomposition of f into elementary operations and functions, as illustrated in Example 2.2.

> **Example 2.2** (Applying rules of differentiation). Consider $f(w) = e^w \ln w + \cos w^2$. The derivative of f on $w > 0$ can be computed

step by step as follows, denoting $\mathrm{sq}(w) := w^2$,

$$\begin{aligned}
f'(w) &= (\exp \cdot \ln)'(w) + (\cos \circ \mathrm{sq})'(w) && \text{(Linearity)} \\
(\exp \cdot \ln)'(w) &= \exp'(w) \cdot \ln(w) + \exp(w) \cdot \ln'(w) && \text{(Product rule)} \\
(\cos \circ \mathrm{sq})'(w) &= \cos'(\mathrm{sq}(w))\, \mathrm{sq}'(w) && \text{(Chain rule)} \\
\exp'(w) &= \exp(w), \qquad \ln'(w) = 1/w, && \text{(Elem. func.)} \\
\mathrm{sq}'(w) &= 2w, \qquad \cos'(w) = -\sin(w). && \text{(Elem. func.)}
\end{aligned}$$

We obtain then that $f'(w) = e^w \ln w + e^w/w - 2w \sin w^2$.

Such a process is purely mechanical and lends itself to an automated procedure, which is the main idea of automatic differentiation presented in Chapter 7.

2.1.3 Leibniz's notation

The notion of derivative was first introduced independently by Newton and Leibniz in the 18^{th} century (Ball, 1960). The latter considered derivatives as the quotient of infinitesimal variations. Namely, denoting $u = f(w)$ a variable depending on w through f, Leibniz considered the derivative of f as the quotient

$$f' = \frac{du}{dw} \quad \text{with} \quad f'(w) = \left.\frac{du}{dw}\right|_w$$

where du and dw denote infinitesimal variations of u and w respectively and the symbol $|_w$ denotes the evaluation of the derivative at a given point w. This notation simplifies the statement of the chain rule first discovered by Leibniz (Rodriguez and Lopez Fernandez, 2010) as we have for $v = g(w)$ and $u = f(v)$

$$\frac{du}{dw} = \frac{du}{dv} \cdot \frac{dv}{dw}.$$

This hints that derivatives are multiplied when considering compositions. At evaluation, the chain rule in Leibniz notation recovers the formula presented above as

$$\left.\frac{du}{dw}\right|_w = \left.\frac{du}{dv}\right|_{g(w)} \left.\frac{dv}{dw}\right|_w = f'(g(w))g'(w) = (f \circ g)'(w).$$

The ability of Leibniz's notation to capture the chain rule as a mere product of quotients made it popular throughout the centuries, especially in mechanics (Ball, 1960). The rationale behind Leibniz's notation, that is, the concept of "infinitesimal variations" was questioned by later mathematicians for its potential logical issues (Ball, 1960). The notation $f'(w)$ first introduced by Euler and further popularized by Lagrange (Cajori, 1993) has then taken over in numerous mathematical textbooks. The concept of infinitesimal variations has been rigorously defined by considering the set of hyperreal numbers. They extend the set of real numbers by considering each number as a sum of a non-infinitesimal part and an infinitesimal part (Hewitt, 1948). The formalism of infinitesimal variations further underlies the development of automatic differentiation algorithms through the concept of dual numbers.

2.2 Multivariate functions

2.2.1 Directional derivatives

Let us now consider a function $f : \mathbb{R}^P \to \mathbb{R}$ of multiple inputs $\boldsymbol{w} = (w_1, \ldots, w_P) \in \mathbb{R}^P$. The most important example in machine learning is a function which, to the parameters $\boldsymbol{w} \in \mathbb{R}^P$ of a neural network, associates a loss value in $\mathbb{R}$. Variations of f need to be defined along specific directions, such as the variation $f(\boldsymbol{w} + \delta \boldsymbol{v}) - f(\boldsymbol{w})$ of f around $\boldsymbol{w} \in \mathbb{R}^P$ in the direction $\boldsymbol{v} \in \mathbb{R}^P$ by an amount $\delta > 0$. This consideration naturally leads to the definition of the directional derivative.

Definition 2.3 (Directional derivative)**.** The **directional derivative** of f at $\boldsymbol{w}$ in the **direction** $\boldsymbol{v}$ is given by

$$\partial f(\boldsymbol{w})[\boldsymbol{v}] := \lim_{\delta \to 0} \frac{f(\boldsymbol{w} + \delta \boldsymbol{v}) - f(\boldsymbol{w})}{\delta},$$

provided that the limit exists.

One example of directional derivative consists in computing the derivative of a function f at $\boldsymbol{w}$ in any of the canonical directions

$$\boldsymbol{e}_i := (0, \ldots, 0, \underbrace{1}_{i}, 0, \ldots, 0).$$

This allows us to define the notion of **partial derivatives**, denoted for $i \in [P]$

$$\partial_i f(\boldsymbol{w}) := \partial f(\boldsymbol{w})[\boldsymbol{e}_i] = \lim_{\delta \to 0} \frac{f(\boldsymbol{w} + \delta \boldsymbol{e}_i) - f(\boldsymbol{w})}{\delta}.$$

This is also denoted in Leibniz's notation as $\partial_i f(\boldsymbol{w}) = \frac{\partial f(\boldsymbol{w})}{\partial w_i}$ or $\partial_i f(\boldsymbol{w}) = \partial_{w_i} f(\boldsymbol{w})$. By moving along only the i^{th} coordinate of the function, the partial derivative is akin to using the function $\phi(\omega_i) = f(w_1, \ldots, \omega_i, \ldots, w_P)$ around ω_i, letting all other coordinates fixed at their values $\boldsymbol{w}_i$.

2.2.2 Gradients

We now introduce the gradient vector, which gathers the partial derivatives. We first recall the definitions of linear map and linear form.

Definition 2.4 (Linear map, linear form). A function $l : \mathbb{R}^P \to \mathbb{R}^M$ is a **linear map** if for any $a_1, a_2 \in \mathbb{R}$, $\boldsymbol{v}_1, \boldsymbol{v}_2 \in \mathbb{R}^D$,

$$l[a_1 \boldsymbol{v}_1 + a_2 \boldsymbol{v}_2] = a_1 l(\boldsymbol{v}_1) + a_2 l[\boldsymbol{v}_2].$$

A linear map with values in $\mathbb{R}$, $l : \mathbb{R}^P \to \mathbb{R}$, is called a **linear form**.

Linearity plays a crucial role in the differentiability of a function.

Definition 2.5 (Differentiability, single-output case). A function $f : \mathbb{R}^P \to \mathbb{R}$ is **differentiable** at $\boldsymbol{w} \in \mathbb{R}^P$ if its directional derivative is defined along any direction, linear in any direction, and if

$$\lim_{\|\boldsymbol{v}\|_2 \to 0} \frac{|f(\boldsymbol{w} + \boldsymbol{v}) - f(\boldsymbol{w}) - \partial f(\boldsymbol{w})[\boldsymbol{v}]|}{\|\boldsymbol{v}\|_2} = 0.$$

We can now introduce the gradient.

Definition 2.6 (Gradient). The **gradient** of a differentiable function $f : \mathbb{R}^P \to \mathbb{R}$ at a point $\boldsymbol{w} \in \mathbb{R}^P$ is defined as the vector of partial derivatives

$$\nabla f(\boldsymbol{w}) := \begin{pmatrix} \partial_1 f(\boldsymbol{w}) \\ \vdots \\ \partial_P f(\boldsymbol{w}) \end{pmatrix} = \begin{pmatrix} \partial f(\boldsymbol{w})[\boldsymbol{e}_1] \\ \vdots \\ \partial f(\boldsymbol{w})[\boldsymbol{e}_P] \end{pmatrix}.$$

By linearity, the directional derivative of f at $\boldsymbol{w}$ in the direction $\boldsymbol{v} = \sum_{i=1}^P v_i \boldsymbol{e}_i$ is then given by

$$\partial f(\boldsymbol{w})[\boldsymbol{v}] = \sum_{i=1}^P v_i \partial f(\boldsymbol{w})[\boldsymbol{e}_i] = \langle \boldsymbol{v}, \nabla f(\boldsymbol{w})\rangle.$$

In the definition above, the fact that the gradient can be used to compute the directional derivative is a mere consequence of the linearity. However, in more abstract cases presented in later sections, the gradient is defined through this property.

As a simple example, any linear function of the form $f(\boldsymbol{w}) = \boldsymbol{a}^\top \boldsymbol{w} = \sum_{i=1}^P a_i w_i$ is differentiable as we have $(\boldsymbol{a}^\top(\boldsymbol{w}+\boldsymbol{v}) - \boldsymbol{a}^\top \boldsymbol{w} - \boldsymbol{a}^\top \boldsymbol{v})/\|\boldsymbol{v}\|_2 = 0$ for any $\boldsymbol{v}$ and in particular for $\|\boldsymbol{v}\| \to 0$. Moreover, its gradient is naturally given by $\nabla f(\boldsymbol{w}) = \boldsymbol{a}$.

Generally, to show that a function is differentiable and find its gradient, one approach is to approximate $f(\boldsymbol{w}+\boldsymbol{v})$ around $\boldsymbol{v} = 0$. If we can find a vector $\boldsymbol{g}$ such that

$$f(\boldsymbol{w}+\boldsymbol{v}) = f(\boldsymbol{w}) + \langle \boldsymbol{g}, \boldsymbol{v}\rangle + o(\|\boldsymbol{v}\|_2),$$

then f is differentiable at $\boldsymbol{w}$ since $\langle \boldsymbol{g}, \cdot\rangle$ is linear. Moreover, $\boldsymbol{g}$ is then the gradient of f at $\boldsymbol{w}$.

Remark 2.2 (Gateaux and Fréchet differentiability). Multiple definitions of differentiability exist. The one presented in Definition 2.5 is about **Fréchet differentiable** functions. Alternatively, if $f : \mathbb{R}^P \to \mathbb{R}$ has well-defined directional derivatives along any directions then the function is **Gateaux differentiable**. Note that the existence of directional derivatives in any directions is not a sufficient condition for the function to be differentiable. In other words, any Fréchet differentiable function is Gateaux differentiable, but the converse is not true. As a counter-example, one can verify that the function $f(x_1, x_2) = x_1^3/(x_1^2 + x_2^2)$ is Gateaux differentiable at 0 but not (Fréchet) differentiable at 0 (because the directional derivative at 0 is not linear).

Some authors also require Gateaux differentiable functions to have linear directional derivatives along any direction. These are still not Fréchet differentiable functions. Indeed, the limit in Definition 2.5 is over any vectors tending to 0 (potentially in a pathological way), while directional derivatives look at such limits uniquely in terms of a single direction.

In the remainder of this chapter, all definitions of differentiability are in terms of Fréchet differentiability.

Example 2.3 illustrates how to compute the gradient of the logistic loss and validate its differentiability.

Example 2.3 (Gradient of logistic loss). Consider the logistic loss $\ell(\boldsymbol{\theta}, \boldsymbol{y}) := -\boldsymbol{y}^\top \boldsymbol{\theta} + \log \sum_{i=1}^M e^{\theta_i}$, that measures the prediction error of the logits $\boldsymbol{\theta} \in \mathbb{R}^M$ w.r.t. the correct label $\boldsymbol{y} \in \{\boldsymbol{e}_1, \ldots, \boldsymbol{e}_M\}$. Let us compute the gradient of this loss w.r.t. $\boldsymbol{\theta}$ for fixed $\boldsymbol{y}$, i.e., we want to compute the gradient of $f(\boldsymbol{\theta}) := \ell(\boldsymbol{\theta}, \boldsymbol{y})$. Let us decompose f as $f = l + \mathrm{logsumexp}$ with $l(\boldsymbol{\theta}) := \langle -\boldsymbol{y}, \boldsymbol{\theta} \rangle$ and

$$\mathrm{logsumexp}(\boldsymbol{\theta}) := \log \sum_{i=1}^M e^{\theta_i},$$

the log-sum-exp function. The function l is linear so differentiable with gradient $\nabla l(\theta) = -\boldsymbol{y}$. We therefore focus on logsumexp. Denoting $\exp(\boldsymbol{\theta}) = (\exp(\theta_1), \ldots, \exp(\theta_M))$, and using that $\exp(1 + x) = 1 + x + o(x)$, and $\log(1 + x) = x + o(x)$, we get

$$\begin{aligned}
\mathrm{logsumexp}(\boldsymbol{\theta} + \boldsymbol{v}) &= \log(\langle \exp(\boldsymbol{\theta} + \boldsymbol{v}), \mathbf{1} \rangle) \\
&= \log(\langle \exp(\boldsymbol{\theta}), \mathbf{1} \rangle + \langle \exp(\boldsymbol{\theta}), \boldsymbol{v} \rangle + o(\|\boldsymbol{v}\|_2)) \\
&= \log(\langle \exp(\boldsymbol{\theta}), \mathbf{1} \rangle) + \left\langle \frac{\exp(\boldsymbol{\theta})}{\langle \exp(\boldsymbol{\theta}), \mathbf{1} \rangle}, \boldsymbol{v} \right\rangle + o(\|\boldsymbol{v}\|_2),
\end{aligned}$$

The above decomposition of $\mathrm{logsumexp}(\boldsymbol{\theta} + \boldsymbol{v})$ shows that it is differentiable, and that $\nabla \mathrm{logsumexp}(\boldsymbol{\theta}) = \mathrm{softargmax}(\boldsymbol{\theta})$, where

$$\mathrm{softargmax}(\boldsymbol{\theta}) := \left(e^{\theta_1} \Big/ \left(\sum_{j=1}^M e^{\theta_j} \right), \ldots, e^{\theta_M} \Big/ \left(\sum_{j=1}^M e^{\theta_j} \right) \right).$$

In total, we then get that $\nabla f(\boldsymbol{\theta}) = -\boldsymbol{y} + \mathrm{softargmax}(\boldsymbol{\theta})$.

Linearity of gradients

The notion of differentiability for multi-inputs functions naturally inherits from the linearity of derivatives for single-input functions. For any $u_1, \ldots, u_M \in \mathbb{R}$ and any multi-inputs functions $f_1, \ldots, f_M$ differentiable at $\boldsymbol{w}$, the function $u_1 f_1 + \ldots + u_M f_M$ is differentiable at $\boldsymbol{w}$ and its gradient is

$$\nabla(u_1 f_1 + \ldots + u_M f_M)(\boldsymbol{w}) = u_1 \nabla f_1(\boldsymbol{w}) + \ldots + u_M \nabla f_M(\boldsymbol{w}).$$

Why is the gradient useful?

The gradient defines the steepest ascent direction of f from $\boldsymbol{w}$. To see why, notice that

$$\operatorname*{arg\,max}_{\boldsymbol{v} \in \mathbb{R}^P, \|\boldsymbol{v}\|_2 \leq 1} \partial f(\boldsymbol{w})[\boldsymbol{v}] = \operatorname*{arg\,max}_{\boldsymbol{v} \in \mathbb{R}^P, \|\boldsymbol{v}\|_2 \leq 1} \langle \boldsymbol{v}, \nabla f(\boldsymbol{w}) \rangle = \nabla f(\boldsymbol{w}) / \|\nabla f(\boldsymbol{w})\|_2,$$

where we assumed $\nabla f(\boldsymbol{w}) \neq \mathbf{0}$. The gradient $\nabla f(\boldsymbol{w})$ is orthogonal to the level set of the function (the set of points $\boldsymbol{w}$ sharing the same value $f(\boldsymbol{w})$) and points towards higher values of f as illustrated in Fig. 2.2. Conversely, the negated gradient $-\nabla f(\boldsymbol{w})$ points towards lower values of f. This observation motivates the development of optimization algorithms such as gradient descent. It is based on iteratively performing the update $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \gamma \nabla f(\boldsymbol{w}_t)$, for $\gamma > 0$. It therefore seeks for a minimizer of f by moving along the direction of steepest descent around $\boldsymbol{w}_t$ given, up to a multiplicative factor, by $-\nabla f(\boldsymbol{w}_t)$.

2.2.3 Jacobians

Let us now consider a multi-output function $f : \mathbb{R}^P \to \mathbb{R}^M$ defined by $f(\boldsymbol{w}) := (f_1(\boldsymbol{w}), \ldots, f_M(\boldsymbol{w}))$, where $f_j : \mathbb{R}^P \to \mathbb{R}$. A typical example in machine learning is a neural network. The notion of directional derivative can be extended to such function by defining it as the vector composed of the coordinate-wise directional derivatives:

$$\partial f(\boldsymbol{w})[\boldsymbol{v}] := \lim_{\delta \to 0} \frac{f(\boldsymbol{w} + \delta \boldsymbol{v}) - f(\boldsymbol{w})}{\delta} = \lim_{\delta \to 0} \begin{pmatrix} \frac{f_1(\boldsymbol{w}+\delta\boldsymbol{v}) - f_1(\boldsymbol{w})}{\delta} \\ \vdots \\ \frac{f_M(\boldsymbol{w}+\delta\boldsymbol{v}) - f_M(\boldsymbol{w})}{\delta} \end{pmatrix} \in \mathbb{R}^M,$$

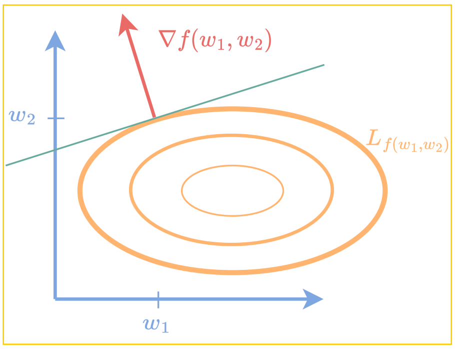

Figure 2.2: The gradient of a function $f : \mathbb{R}^2 \to \mathbb{R}$ at (w_1, w_2) is the normal vector to the tangent space of the level set $L_{f(w_1,w_2)} = \{(w_1', w_2') : f(w_1', w_2') = f(w_1, w_2)\}$ and points towards points with higher function values.

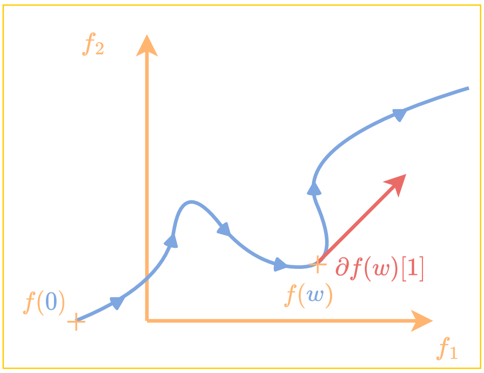

Figure 2.3: The directional derivative of a parameterized curve $f : \mathbb{R} \to \mathbb{R}^2$ at t is the tangent to the curve at the point $f(w) \in \mathbb{R}^2$.

where the limits (provided that they exist) are applied coordinate-wise. The directional derivative of f in the direction $\boldsymbol{v} \in \mathbb{R}^P$ is therefore the vector that gathers the directional derivative of each f_j, i.e., $\partial f(\boldsymbol{w})[\boldsymbol{w}] = (\partial f_j(\boldsymbol{v})[\boldsymbol{v}])_{j=1}^M$. In particular, we can define the **partial derivatives** of f at $\boldsymbol{w}$ as the vectors

$$\partial_i f(\boldsymbol{w}) := \partial f(\boldsymbol{w})[\boldsymbol{e}_i] = \begin{pmatrix} \partial_i f_1(\boldsymbol{w}) \\ \vdots \\ \partial_i f_M(\boldsymbol{w}) \end{pmatrix} \in \mathbb{R}^M.$$

As for the usual definition of the derivative, the directional derivative can provide a linear approximation of a function around a current input as illustrated in Fig. 2.3 for a parameterized curve $f : \mathbb{R} \to \mathbb{R}^2$.

Just as in the single-output case, differentiability is defined not only as the existence of directional derivatives in any direction but also by the linearity in the chosen direction.

> **Definition 2.7** (Differentiability, multi-output case). A function $f : \mathbb{R}^P \to \mathbb{R}^M$ is (Fréchet) **differentiable** at a point $\boldsymbol{w} \in \mathbb{R}^P$ if its directional derivative is defined along any directions, linear along

any directions, and,

$$\lim_{\|\boldsymbol{v}\|_2 \to 0} \frac{\|f(\boldsymbol{w}+\boldsymbol{v}) - f(\boldsymbol{w}) - \partial f(\boldsymbol{w})[\boldsymbol{v}]\|_2}{\|\boldsymbol{v}\|_2} = 0.$$

The partial derivatives of all function coordinates are gathered in the **Jacobian matrix**.

Definition 2.8 (Jacobian). The **Jacobian** of a differentiable function $f : \mathbb{R}^P \to \mathbb{R}^M$ at $\boldsymbol{w}$ is defined as the matrix of all partial derivatives of all coordinate functions provided they exist,

$$\boldsymbol{\partial} f(\boldsymbol{w}) := \begin{pmatrix} \partial_1 f_1(\boldsymbol{w}) & \dots & \partial_P f_1(\boldsymbol{w}) \\ \vdots & \ddots & \vdots \\ \partial_1 f_M(\boldsymbol{w}) & \dots & \partial_P f_M(\boldsymbol{w}) \end{pmatrix} \in \mathbb{R}^{M \times P}.$$

The Jacobian can be represented by stacking columns of partial derivatives or rows of gradients,

$$\boldsymbol{\partial} f(\boldsymbol{w}) = \Big(\partial_1 f(\boldsymbol{w}), \dots, \partial_P f(\boldsymbol{w})\Big) = \begin{pmatrix} \nabla f_1(\boldsymbol{w})^\top \\ \vdots \\ \nabla f_M(\boldsymbol{w})^\top \end{pmatrix}.$$

By linearity, the directional derivative of f at $\boldsymbol{w}$ along any input direction $\boldsymbol{v} = \sum_{i=1}^P v_i \boldsymbol{e}_i \in \mathbb{R}^P$ is then given by

$$\partial f(\boldsymbol{w})[\boldsymbol{v}] = \sum_{i=1}^P v_i \partial_i f(\boldsymbol{w}) = \boldsymbol{\partial} f(\boldsymbol{w}) \boldsymbol{v} \in \mathbb{R}^M.$$

Notice that we use bold $\boldsymbol{\partial}$ to indicate the Jacobian matrix. The Jacobian matrix naturally generalizes the concepts of derivatives and gradients presented earlier. As for the single input case, to show that a function is differentiable, one approach is to approximate $f(\boldsymbol{w}+\boldsymbol{v})$ around $\boldsymbol{v} = \boldsymbol{0}$. If we find a linear map l such that

$$f(\boldsymbol{w}+\boldsymbol{v}) = f(\boldsymbol{w}) + l[\boldsymbol{v}] + o(\|\boldsymbol{v}\|_2),$$

then f is differentiable at $\boldsymbol{w}$. Moreover, if l is represented by matrix $\boldsymbol{J}$ such that $l[\boldsymbol{v}] = \boldsymbol{J}\boldsymbol{v}$ then $\boldsymbol{J} = \boldsymbol{\partial} f(\boldsymbol{w})$.

As a simple example, any linear function $f(\boldsymbol{w}) = \boldsymbol{A}\boldsymbol{w}$ for $\boldsymbol{A} \in \mathbb{R}^{M\times P}$ is differentiable, since all its coordinate-wise components are single-output linear functions, and the Jacobian of f at any $\boldsymbol{w}$ is given by $\boldsymbol{\partial} f(\boldsymbol{w}) = \boldsymbol{A}$.

Remark 2.3 (Special cases of the Jacobian). For single-output functions $f : \mathbb{R}^P \to \mathbb{R}$, i.e., $M = 1$, the Jacobian matrix reduces to a row vector identified as the transpose of the gradient, i.e.,

$$\boldsymbol{\partial} f(\boldsymbol{w}) = \nabla f(\boldsymbol{w})^\top \in \mathbb{R}^{1\times P}.$$

For a single-input function $f : \mathbb{R} \to \mathbb{R}^M$, the Jacobian reduces to a single column vector of directional derivatives, denoted

$$\boldsymbol{\partial} f(w) = f'(w) := \begin{pmatrix} f_1'(w) \\ \vdots \\ f_M'(w) \end{pmatrix} \in \mathbb{R}^{M\times 1}.$$

For a single-input single-output function $f : \mathbb{R} \to \mathbb{R}$, the Jacobian reduces to the derivative of f, i.e.,

$$\boldsymbol{\partial} f(w) = f'(w) \in \mathbb{R}.$$

Example 2.4 illustrates the form of the Jacobian matrix for the element-wise application of a differentiable function such as the softplus activation. This example already shows that the Jacobian takes a simple diagonal matrix form. As a consequence, the directional derivative associated with this function is simply given by an element-wise product rather than a full matrix-vector product as suggested in Definition 2.8. We will revisit this point in Section 2.3.

Example 2.4 (Jacobian matrix of the softplus activation). Consider the element-wise application of the softplus defined for $\boldsymbol{w} \in \mathbb{R}^P$ by

$$f(\boldsymbol{w}) := \begin{pmatrix} \sigma(w_1) \\ \vdots \\ \sigma(w_P) \end{pmatrix} \in \mathbb{R}^P \quad \text{where} \quad \sigma(w) := \log(1 + e^w).$$

Since σ is differentiable, each coordinate of this function is differentiable and the overall function is differentiable. The j^{th} coordinate of f is independent of the i^{th} coordinate of $\boldsymbol{w}$ for $i \neq j$, so $\partial_i f_j(\boldsymbol{w}) = 0$ for $i \neq j$. For $i = j$, the result boils down to the derivative of σ at w_j. That is, $\partial_j f_j(\boldsymbol{w}) = \sigma'(w_j)$, where $\sigma'(w) = e^w/(1+e^w)$. The Jacobian of f is therefore a diagonal matrix

$$\boldsymbol{\partial} f(\boldsymbol{w}) = \mathbf{diag}(\sigma'(w_1), \dots, \sigma'(w_P)) := \begin{pmatrix} \sigma'(w_1) & 0 & \dots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \sigma'(w_P) \end{pmatrix}.$$

Variations along outputs

Rather than considering variations of f along an **input** direction $\boldsymbol{v} \in \mathbb{R}^P$, we may also consider the variations of f along an **output** direction $\boldsymbol{u} \in \mathbb{R}^M$, namely, computing the gradients $\nabla(\boldsymbol{u}^\top f)(\boldsymbol{w})$ of single-output functions, where we defined

$$(\boldsymbol{u}^\top f)(\boldsymbol{w}) := \boldsymbol{u}^\top f(\boldsymbol{w}) \in \mathbb{R}.$$

In particular, we may consider computing the gradients $\nabla f_j(\boldsymbol{w})$ of each function coordinate $f_j = \boldsymbol{e}_j^\top f$ at $\boldsymbol{w}$, where $\boldsymbol{e}_j$ is the j^{th} canonical vector in $\mathbb{R}^M$. The infinitesimal variations of f at $\boldsymbol{w}$ along any output direction $\boldsymbol{u} = \sum_{j=1}^M u_j \boldsymbol{e}_j \in \mathbb{R}^M$ are given by

$$\nabla(\boldsymbol{u}^\top f)(\boldsymbol{w}) = \sum_{j=1}^M u_j \nabla f_j(\boldsymbol{w}) = \boldsymbol{\partial} f(\boldsymbol{w})^\top \boldsymbol{u} \in \mathbb{R}^P,$$

where $\boldsymbol{\partial} f(\boldsymbol{w})^\top$ is the Jacobian's transpose. Using the definition of derivative as a limit, we obtain for $i \in [P]$

$$\nabla_i(\boldsymbol{u}^\top f)(\boldsymbol{w}) = [\boldsymbol{\partial} f(\boldsymbol{w})^\top \boldsymbol{u}]_i = \lim_{\delta \to 0} \frac{\boldsymbol{u}^\top (f(\boldsymbol{w} + \delta \boldsymbol{e}_i) - f(\boldsymbol{w}))}{\delta}.$$

Chain rule

Equipped with a generic definition of differentiability and the associated objects, gradients and Jacobians, we can now generalize the chain rule, previously introduced for single-input single-output functions.

Proposition 2.2 (Chain rule). Consider $f : \mathbb{R}^P \to \mathbb{R}^M$ and $g : \mathbb{R}^M \to \mathbb{R}^R$. If f is differentiable at $\boldsymbol{w} \in \mathbb{R}^P$ and g is differentiable at $f(\boldsymbol{w}) \in \mathbb{R}^M$, then the composition $g \circ f$ is differentiable at $\boldsymbol{w} \in \mathbb{R}^P$ and its Jacobian is given by

$$\boldsymbol{\partial}(g \circ f)(\boldsymbol{w}) = \boldsymbol{\partial} g(f(\boldsymbol{w}))\boldsymbol{\partial} f(\boldsymbol{w}).$$

Proof. We progressively approximate $g \circ f(\boldsymbol{w} + \boldsymbol{v})$ using the differentiability of f at $\boldsymbol{w}$ and g at $f(\boldsymbol{w})$,

$$\begin{aligned} g(f(\boldsymbol{w} + \boldsymbol{v})) &= g(f(\boldsymbol{w}) + \boldsymbol{\partial} f(\boldsymbol{w})\boldsymbol{v} + o(\|\boldsymbol{v}\|)) \\ &= g(f(\boldsymbol{w})) + \boldsymbol{\partial} g(f(\boldsymbol{w}))\boldsymbol{\partial} f(\boldsymbol{w})\boldsymbol{v} + o(\|\boldsymbol{v}\|). \end{aligned}$$

Hence, $g \circ f$ is differentiable at $\boldsymbol{w}$ with Jacobian $\boldsymbol{\partial} g(f(\boldsymbol{w}))\boldsymbol{\partial} f(\boldsymbol{w})$. ☐

Proposition 2.2 can be seen as the cornerstone of any derivative computations. For example, it can be used to rederive the linearity or the product rule associated to the derivatives of single-input single-outptut functions.

When g is scalar-valued, combined with Remark 2.3, we obtain a simple expression for $\nabla(g \circ f)$.

Proposition 2.3 (Chain rule, scalar-valued case). Consider $f : \mathbb{R}^P \to \mathbb{R}^M$ and $g : \mathbb{R}^M \to \mathbb{R}$. The gradient of the composition is given by

$$\nabla(g \circ f)(\boldsymbol{w}) = \boldsymbol{\partial} f(\boldsymbol{w})^\top \nabla g(f(\boldsymbol{w})).$$

This is illustrated with linear regression in Example 2.5.

Example 2.5 (Linear regression). Consider the squared residuals of a linear regression of N inputs $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \in \mathbb{R}^D$ onto N targets $y_1, \ldots, y_N \in \mathbb{R}$ with a vector $\boldsymbol{w} \in \mathbb{R}^D$, that is, $f(\boldsymbol{w}) = \|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}\|_2^2 = \sum_{i=1}^N (\boldsymbol{x}_i^\top \boldsymbol{w} - y_i)^2$ for $\boldsymbol{X} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N)^\top \in \mathbb{R}^{N \times D}$ and $\boldsymbol{y} = (y_1, \ldots, y_N)^\top \in \mathbb{R}^N$.

The function f can be decomposed into a linear mapping $f_1(\boldsymbol{w}) = \boldsymbol{X}\boldsymbol{w}$ and a squared error $f_2(\boldsymbol{p}) = \|\boldsymbol{p} - \boldsymbol{y}\|_2^2$, so that $f = f_2 \circ f_1$. We can then apply the chain rule in Proposition 2.3 to get

$$\nabla f(\boldsymbol{w}) = \boldsymbol{\partial} f_1(\boldsymbol{w})^\top \nabla f_2(f_1(\boldsymbol{w}))$$

provided that f_1, f_2 are differentiable at $\boldsymbol{w}$ and $f_1(\boldsymbol{w})$, respectively.

The function f_1 is linear so differentiable with Jacobian $\boldsymbol{\partial} f_1(\boldsymbol{w}) = \boldsymbol{X}$. On the other hand the partial derivatives of f_2 are given by $\partial_j f_2(\boldsymbol{p}) = 2(p_j - y_j)$ for $j \in \{1, \ldots, N\}$. Therefore, f_2 is differentiable at any $\boldsymbol{p}$ and its gradient is $\nabla f_2(\boldsymbol{p}) = 2(\boldsymbol{p} - \boldsymbol{y})$. By combining the computations of the Jacobian of f_1 and the gradient of f_2, we then get the gradient of f as

$$\nabla f(\boldsymbol{w}) = 2\boldsymbol{X}^\top (f_1(\boldsymbol{w}) - \boldsymbol{y}) = 2\boldsymbol{X}^\top (\boldsymbol{X}\boldsymbol{w} - \boldsymbol{y}).$$

2.3 Linear differentiation maps

The Jacobian matrix is useful as a representation of the partial derivatives. However, the core idea underlying the definition of differentiable functions, as well as their implementation in an autodiff framework, lies in the access to two key **linear maps**. These two maps encode infinitesimal variations along **input** or **output** directions and are referred to, respectively, as **Jacobian-vector product** (JVP) and **Vector-jacobian product** (VJP). This section formalizes these notions, in the context of Euclidean spaces.

2.3.1 The need for linear maps

So far, we have focused on functions $f \colon \mathbb{R}^P \to \mathbb{R}^M$, that take a vector as input and produce a vector as output. However, functions that use matrix or even tensor inputs/outputs are common place in neural networks. For example, consider the function of matrices of the form $f(\boldsymbol{W}) = \boldsymbol{W}\boldsymbol{x}$, where $\boldsymbol{x} \in \mathbb{R}^D$ and $\boldsymbol{W} \in \mathbb{R}^{M \times D}$. This function takes a matrix as input, not a vector. Of course, a matrix $\boldsymbol{W} \in \mathbb{R}^{M \times D}$ can always be “flattened” into a vector $\boldsymbol{w} \in \mathbb{R}^{MD}$, by stacking the columns of $\boldsymbol{W}$. We denote this operation by $\boldsymbol{w} = \mathrm{vec}(\boldsymbol{W})$ and its inverse by $\boldsymbol{W} = \mathrm{vec}^{-1}(\boldsymbol{w})$. We can then equivalently write $f(\boldsymbol{W})$ as $\tilde{f}(\boldsymbol{w}) = f(\mathrm{vec}^{-1}(\boldsymbol{w})) = \mathrm{vec}^{-1}(\boldsymbol{w})\boldsymbol{x}$, so that the previous framework applies. However, we will now see that this would be inefficient.

Indeed, the resulting Jacobian of $\tilde{f}$ at any $\boldsymbol{w}$ consists in a matrix of size $\mathbb{R}^{M\times MD}$, which, after some computations, can be observed to be mostly filled with zeros. Getting the directional derivative of f at $\boldsymbol{W} \in \mathbb{R}^{M\times D}$ in a direction $\boldsymbol{V} \in \mathbb{R}^{M\times D}$ would consist in (i) vectorizing $\boldsymbol{V}$ into $\boldsymbol{v} = \mathrm{vec}(\boldsymbol{V})$, (ii) computing the matrix-vector product $\boldsymbol{\partial}\tilde{f}(\boldsymbol{w})\boldsymbol{v}$ at a cost of M^3D^2 computations (ignoring the fact that the Jacobian has zero entries), (iii) re-shaping the result into a matrix.

On the other hand, since f is linear in its matrix input, we can infer that the directional derivative of f at any $\boldsymbol{W} \in \mathbb{R}^{M\times D}$ in any direction $\boldsymbol{V} \in \mathbb{R}^{M\times D}$ is simply given by the function itself applied on $\boldsymbol{V}$. Namely, we have $\partial f(\boldsymbol{W})[\boldsymbol{V}] = f(\boldsymbol{V}) = \boldsymbol{V}\boldsymbol{x}$, which is simple to implement and clearly only requires MD^2 operations. Note that the cost would have been the same, had we ignored the non-zero entries of $\boldsymbol{\partial}\tilde{f}(\boldsymbol{w})$. The point here is that by considering the operations associated to the differentiation of a function as linear maps rather than using the associated representation as a Jacobian matrix, we can streamline the associated implementations and exploit the structures of the underlying input or output space. To that end, we now recall the main abstractions necessary to extend the previous definitions in the context of Euclidean spaces.

2.3.2 Euclidean spaces

Linear spaces, a.k.a. **vector spaces**, are spaces equipped (and closed under) an addition rule compatible with multiplication by a scalar (we limit ourselves to the field of reals). Namely, in a vector space $\mathcal{E}$, there exists the operations $+$ and $\cdot$, such that for any $\boldsymbol{u}$, $\boldsymbol{v} \in \mathcal{E}$, and $a \in \mathbb{R}$, we have $\boldsymbol{u} + \boldsymbol{v} \in \mathcal{E}$ and $a \cdot \boldsymbol{u} \in \mathcal{E}$. **Euclidean spaces** are linear spaces equipped with a basis $\boldsymbol{e}_1, \ldots, \boldsymbol{e}_P \in \mathcal{E}$. Any element $\boldsymbol{v} \in \mathcal{E}$ can be decomposed as $\boldsymbol{v} = \sum_{i=1}^{P} v_i \boldsymbol{e}_i$ for some unique scalars $v_1, \ldots, v_P \in \mathbb{R}$. A canonical example of Euclidean space is the set $\mathbb{R}^P$ of all vectors of size P that we already covered. The set of matrices $\mathbb{R}^{P_1\times P_2}$ of size $P_1 \times P_2$ is also naturally a Euclidean space generated by the set of canonical matrices $\boldsymbol{E}_{ij} \in \{0,1\}^{P_1\times P_2}$ for $i \in [P_1], j \in [P_2]$ filled with zero except at the $(i,j)^{\text{th}}$ entry filled with one. For example, $\boldsymbol{W} \in \mathbb{R}^{P_1\times P_2}$ can be written $\boldsymbol{W} = \sum_{i,j=1}^{P_1,P_2} W_{ij}\boldsymbol{E}_{ij}$.

Euclidean spaces are naturally equipped with a notion of inner product defined below.

Definition 2.9 (Inner product). An **inner product** on a vector space $\mathcal{E}$ is a function $\langle \cdot, \cdot \rangle : \mathcal{E} \times \mathcal{E} \to \mathbb{R}$ that is

- bilinear, that is, $\boldsymbol{x} \mapsto \langle \boldsymbol{x}, \boldsymbol{w} \rangle$ and $\boldsymbol{y} \mapsto \langle \boldsymbol{v}, \boldsymbol{y} \rangle$ are linear for any $\boldsymbol{w}, \boldsymbol{v} \in \mathcal{E}$
- symmetric, that is, $\langle \boldsymbol{w}, \boldsymbol{v} \rangle = \langle \boldsymbol{w}, \boldsymbol{v} \rangle$ for any $\boldsymbol{w}, \boldsymbol{v} \in \mathcal{E}$
- positive definite, that is, $\langle \boldsymbol{w}, \boldsymbol{w} \rangle \geq 0$ for any $\boldsymbol{w} \in \mathcal{E}$, and $\langle \boldsymbol{w}, \boldsymbol{w} \rangle = 0$ if and only if $\boldsymbol{w} = 0$.

An inner product defines a norm $\|\boldsymbol{w}\| := \sqrt{\langle \boldsymbol{w}, \boldsymbol{w} \rangle}$.

The norm induced by an inner product defines a distance $\|\boldsymbol{w} - \boldsymbol{v}\|$ between $\boldsymbol{w}, \boldsymbol{v} \in \mathcal{E}$, and therefore a notion of convergence.

For vectors, where $\mathcal{E} = \mathbb{R}^P$, the inner product is the usual one $\langle \boldsymbol{w}, \boldsymbol{v} \rangle = \sum_{i=1}^P w_i v_i$. For matrices, where $\mathcal{E} = \mathbb{R}^{P_1 \times P_2}$, the inner product is the so-called Frobenius inner product. It is defined for any $\boldsymbol{W}, \boldsymbol{V} \in \mathbb{R}^{P_1 \times P_2}$ by

$$\langle \boldsymbol{W}, \boldsymbol{V} \rangle := \langle \mathrm{vec}(\boldsymbol{W}), \mathrm{vec}(\boldsymbol{V}) \rangle = \sum_{i,j=1}^{P_1, P_2} W_{ij} V_{ij} = \mathrm{tr}(\boldsymbol{W}^\top \boldsymbol{V}),$$

where $\mathrm{tr}(\boldsymbol{Z}) := \sum_{i=1}^P Z_{ii}$ is the trace operator defined for square matrices $\boldsymbol{Z} \in \mathbb{R}^{P \times P}$. For tensors of order R, which generalize matrices to $\mathcal{E} = \mathbb{R}^{P_1 \times \ldots \times P_R}$, the inner product is defined similarly for $\mathbf{W}, \mathbf{V} \in \mathbb{R}^{P_1 \times \ldots \times P_R}$ by

$$\langle \mathbf{W}, \mathbf{V} \rangle := \langle \mathrm{vec}(\mathbf{W}), \mathrm{vec}(\mathbf{V}) \rangle = \sum_{i_1, \ldots, i_R = 1}^{P_1, \ldots, P_R} \mathbf{W}_{i_1 \ldots i_R} \mathbf{V}_{i_1 \ldots i_R},$$

where $\mathbf{W}_{i_1 \ldots i_R}$ is the $(i_1, \ldots, i_R)^{\text{th}}$ entry of $\mathbf{W}$.

2.3.3 Linear maps and their adjoints

The notion of linear map defined in Definition 2.4 naturally extends to Euclidean spaces. Namely, a function $l : \mathcal{E} \to \mathcal{F}$ from a Euclidean space $\mathcal{E}$ onto a Euclidean space $\mathcal{F}$ is a **linear map** if for any $\boldsymbol{w}, \boldsymbol{v} \in \mathcal{E}$ and $a, b \in \mathbb{R}$, we have $l[a\boldsymbol{w} + b\boldsymbol{v}] = a \cdot l[\boldsymbol{w}] + b \cdot l[\boldsymbol{v}]$. When $\mathcal{E} = \mathbb{R}^P$ and $\mathcal{F} = \mathbb{R}^M$, there always exists a matrix $\boldsymbol{A} \in \mathbb{R}^{M \times P}$ such that $l[\boldsymbol{x}] = \boldsymbol{A}\boldsymbol{x}$. Therefore, we can think of $\boldsymbol{A}$ as the "materialization" of l.

We can define the adjoint operator of a linear map.

Definition 2.10 (Adjoint operator)**.** Given two Euclidean spaces $\mathcal{E}$ and $\mathcal{F}$ equipped with inner products $\langle \cdot, \cdot \rangle_{\mathcal{E}}$ and $\langle \cdot, \cdot \rangle_{\mathcal{F}}$, the **adjoint** of a linear map $l : \mathcal{E} \to \mathcal{F}$ is the unique linear map $l^* : \mathcal{F} \to \mathcal{E}$ such that for any $\boldsymbol{v} \in \mathcal{E}$ and $\boldsymbol{u} \in \mathcal{F}$,

$$\langle l[\boldsymbol{v}], \boldsymbol{u} \rangle_{\mathcal{F}} = \langle \boldsymbol{v}, l^*[\boldsymbol{u}] \rangle_{\mathcal{E}}.$$

The adjoint can be thought as the counterpart of the matrix transpose for linear maps. When $l[\boldsymbol{v}] = \boldsymbol{A}\boldsymbol{v}$, we have $l^*[\boldsymbol{u}] = \boldsymbol{A}^\top \boldsymbol{u}$ since

$$\langle l[\boldsymbol{v}], \boldsymbol{u} \rangle_{\mathcal{F}} = \langle \boldsymbol{A}\boldsymbol{v}, \boldsymbol{u} \rangle_{\mathcal{F}} = \langle \boldsymbol{v}, \boldsymbol{A}^\top \boldsymbol{u} \rangle_{\mathcal{E}} = \langle \boldsymbol{v}, l^*[\boldsymbol{u}] \rangle_{\mathcal{E}}.$$

2.3.4 Jacobian-vector products

We now define the directional derivative using linear maps, leading to the notion of Jacobian-vector product (JVP). This can be used to facilitate the treatment of functions on matrices or be used for further extensions to infinite-dimensional spaces. In the following, $\mathcal{E}$ and $\mathcal{F}$ denote two Euclidean spaces equipped with norms $\langle \cdot, \cdot \rangle_{\mathcal{E}}$ and $\langle \cdot, \cdot \rangle_{\mathcal{F}}$. We start by defining differentiability in general Euclidean spaces.

Definition 2.11 (Differentiability in Euclidean spaces)**.** A function $f : \mathcal{E} \to \mathcal{F}$ is **differentiable** at a point $\boldsymbol{w} \in \mathcal{E}$ if the **directional derivative** along $\boldsymbol{v} \in \mathcal{E}$

$$\partial f(\boldsymbol{w})[\boldsymbol{v}] := \lim_{\delta \to 0} \frac{f(\boldsymbol{w} + \delta \boldsymbol{v}) - f(\boldsymbol{w})}{\delta} = l[\boldsymbol{v}],$$

is well-defined for any $\boldsymbol{v} \in \mathcal{E}$, linear in $\boldsymbol{v}$ and if

$$\lim_{\|\boldsymbol{v}\|_2 \to 0} \frac{\|f(\boldsymbol{w} + \boldsymbol{v}) - f(\boldsymbol{w}) - l[\boldsymbol{v}]\|_2}{\|\boldsymbol{v}\|_2} = 0.$$

We can now formally define the Jacobian-vector product.

Definition 2.12 (Jacobian-vector product). For a differentiable function $f : \mathcal{E} \to \mathcal{F}$, the **linear map** $\partial f(\boldsymbol{w}) : \mathcal{E} \to \mathcal{F}$, mapping $\boldsymbol{v}$ to $\partial f(\boldsymbol{w})[\boldsymbol{v}]$ is called the **Jacobian-vector product** (JVP) by analogy with Definition 2.8. Note that $\partial f : \mathcal{E} \to (\mathcal{E} \to \mathcal{F})$.

Strictly speaking, $\boldsymbol{v}$ belongs to $\mathcal{E}$. Therefore it may not be a vector, if for instance $\mathcal{E}$ is the set of real matrices. We adopt the name JVP, as it is standard in the literature.

Recovering the gradient

Previously, we saw that for differentiable functions with vector input and scalar output, the directional derivative is equal to the inner product between the direction and the gradient. The same applies when considering differentiable functions from a Euclidean space with single outputs, except that the gradient is now an element of the input space and the inner product is the one associated with the input space.

Proposition 2.4 (Gradient). If a function $f : \mathcal{E} \to \mathbb{R}$ is differentiable at $\boldsymbol{w} \in \mathcal{E}$, then there exists $\nabla f(\boldsymbol{w}) \in \mathcal{E}$, called the **gradient** of f at $\boldsymbol{w}$ such that the directional derivative of f at $\boldsymbol{w}$ along any input direction $\boldsymbol{v} \in \mathcal{E}$ is given by

$$\partial f(\boldsymbol{w})[\boldsymbol{v}] = \langle \nabla f(\boldsymbol{w}), \boldsymbol{v} \rangle_{\mathcal{E}}.$$

In Euclidean spaces, the existence of the gradient can simply be shown by decomposing the partial derivative along a basis of $\mathcal{E}$. Such a definition generalizes to infinite-dimensional (e.g., Hilbert spaces) spaces as seen in Section 2.3.9.

2.3.5 Vector-Jacobian products

For functions with vector input and vector outputs, we already discussed infinitesimal variations along output directions. The same approach applies for Euclidean spaces and is tied to the adjoint of the JVP as detailed in Proposition 2.5.

Proposition 2.5 (Vector-Jacobian product)**.** If a function $f : \mathcal{E} \to \mathcal{F}$ is differentiable at $\boldsymbol{w} \in \mathcal{E}$, then its infinitesimal variation along an output direction $\boldsymbol{u} \in \mathcal{F}$ is given by the **adjoint map** $\partial f(\boldsymbol{w})^* : \mathcal{F} \to \mathcal{E}$ of the JVP, called the **vector-Jacobian product** (VJP). It satisfies

$$\nabla \langle \boldsymbol{u}, f \rangle_{\mathcal{F}}(\boldsymbol{w}) = \partial f(\boldsymbol{w})^*[\boldsymbol{u}],$$

where we denoted $\langle \boldsymbol{u}, f \rangle_{\mathcal{F}}(\boldsymbol{w}) \coloneqq \langle \boldsymbol{u}, f(\boldsymbol{w}) \rangle_{\mathcal{F}}$. Note that $\partial f(\cdot)^* : \mathcal{E} \to (\mathcal{F} \to \mathcal{E})$.

Proof. The chain rule presented in Proposition 2.2 naturally generalizes to Euclidean spaces (see Proposition 2.6). Since $\langle \boldsymbol{u}, \cdot \rangle$ is linear, its directional derivative is itself. Therefore, the directional derivative of $\langle \boldsymbol{u}, f \rangle_{\mathcal{F}}$ is

$$\begin{aligned} \partial(\langle \boldsymbol{u}, f \rangle_{\mathcal{F}})(\boldsymbol{w})[\boldsymbol{v}] &= \langle \boldsymbol{u}, \partial f(\boldsymbol{w})[\boldsymbol{v}] \rangle_{\mathcal{F}} \\ &= \langle \partial f(\boldsymbol{w})^*[\boldsymbol{u}], \boldsymbol{v} \rangle_{\mathcal{F}}. \end{aligned}$$

As this is true for any $\boldsymbol{v} \in \mathcal{E}$, $\partial f(\boldsymbol{w})^*[\boldsymbol{u}]$ is the gradient of $\langle \boldsymbol{u}, f \rangle_{\mathcal{F}}$ per Proposition 2.4. □

2.3.6 Chain rule

The chain rule presented before in terms of Jacobian matrices can readily be formulated in terms of linear maps to take advantage of the implementations of the JVP and VJP as linear maps.

Proposition 2.6 (Chain rule, general case)**.** Consider $f : \mathcal{E} \to \mathcal{F}$ and $g : \mathcal{F} \to \mathcal{G}$ for $\mathcal{E}, \mathcal{F}, \mathcal{G}$ some Euclidean spaces. If f is differentiable at $\boldsymbol{w} \in \mathcal{E}$ and g is differentiable at $f(\boldsymbol{w}) \in \mathcal{F}$, then the composition $g \circ f$ is differentiable at $\boldsymbol{w} \in \mathcal{E}$. Its JVP is given by

$$\partial(g \circ f)(\boldsymbol{w})[\boldsymbol{v}] = \partial g(f(\boldsymbol{w}))[\partial f(\boldsymbol{w})[\boldsymbol{v}]]$$

and its VJP is given by

$$\partial(g \circ f)(\boldsymbol{w})^*[\boldsymbol{u}] = \partial f(\boldsymbol{w})^*[\partial g(f(\boldsymbol{w}))^*[\boldsymbol{u}]].$$

The proof follows the one of Proposition 2.2. When the last function is scalar-valued, which is often the case in machine learning, we obtain the following simplified result.

Proposition 2.7 (Chain rule, scalar case)**.** Consider $f: \mathcal{E} \to \mathcal{F}$ and $g: \mathcal{F} \to \mathbb{R}$, the gradient of the composition is given by

$$\nabla(g \circ f)(\boldsymbol{w}) = \partial f(\boldsymbol{w})^*[\nabla g(f(\boldsymbol{w}))].$$

2.3.7 Functions of multiple inputs (fan-in)

Oftentimes, the inputs of a function do not belong to only one Euclidean space but to a product of them. An example is $f(\boldsymbol{x}, \boldsymbol{W}) = \boldsymbol{W}\boldsymbol{x}$, which is defined on $\mathcal{E} = \mathbb{R}^D \times \mathbb{R}^{M \times D}$. In such a case, it is convenient to generalize the notion of partial derivatives to handle blocks of inputs.

Consider a function $f(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_S)$ defined on $\mathcal{E} = \mathcal{E}_1 \times \ldots \times \mathcal{E}_S$, where $\boldsymbol{w}_i \in \mathcal{E}_i$. We denote the partial derivative with respect to the i^{th} input $\boldsymbol{w}_i$ along $\boldsymbol{v}_i \in \mathcal{E}_i$ as $\partial_i f(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_S)[\boldsymbol{v}_i]$. Equipped with this notation, we can analyze how JVPs or VJPs are decomposed along several inputs.

Proposition 2.8 (Multiple inputs)**.** Consider a differentiable function of the form $f(\boldsymbol{w}) = f(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_S)$ with signature $f: \mathcal{E} \to \mathcal{F}$, where $\boldsymbol{w} := (\boldsymbol{w}_1, \ldots, \boldsymbol{w}_S) \in \mathcal{E}$ and $\mathcal{E} := \mathcal{E}_1 \times \cdots \times \mathcal{E}_S$. Then the JVP with the input direction $\boldsymbol{v} = (\boldsymbol{v}_1, \ldots, \boldsymbol{v}_S) \in \mathcal{E}$ is given by

$$\begin{aligned}\partial f(\boldsymbol{w})[\boldsymbol{v}] &= \partial f(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_S)[\boldsymbol{v}_1, \ldots, \boldsymbol{v}_S] \in \mathcal{F} \\ &= \sum_{i=1}^{S} \partial_i f(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_S)[\boldsymbol{v}_i].\end{aligned}$$

The VJP with the output direction $\boldsymbol{u} \in \mathcal{F}$ is given by

$$\begin{aligned}\partial f(\boldsymbol{w})^*[\boldsymbol{u}] &= \partial f(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_S)^*[\boldsymbol{u}] \in \mathcal{E} \\ &= (\partial_1 f(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_S)^*[\boldsymbol{u}], \ldots, \partial_S f(\boldsymbol{w}_1, \ldots, \boldsymbol{w}_S)^*[\boldsymbol{u}]).\end{aligned}$$

Example 2.6 (Matrix-vector product). Consider $f(\boldsymbol{x}, \boldsymbol{W}) = \boldsymbol{W}\boldsymbol{x}$, where $\boldsymbol{W} \in \mathbb{R}^{M \times D}$ and $\boldsymbol{x} \in \mathbb{R}^D$. This corresponds to setting $\mathcal{E} = \mathcal{E}_1 \times \mathcal{E}_2 = \mathbb{R}^D \times \mathbb{R}^{M \times D}$ and $\mathcal{F} = \mathbb{R}^M$. For the JVP, letting $\boldsymbol{v} \in \mathbb{R}^D$ and $\boldsymbol{V} \in \mathbb{R}^{M \times D}$, we obtain

$$\partial f(\boldsymbol{x}, \boldsymbol{W})[\boldsymbol{v}, \boldsymbol{V}] = \boldsymbol{W}\boldsymbol{v} + \boldsymbol{V}\boldsymbol{x} \in \mathcal{F}.$$

We can also access the individual JVPs as

$$\begin{aligned}\partial_1 f(\boldsymbol{x}, \boldsymbol{W})[\boldsymbol{v}] &= \boldsymbol{W}\boldsymbol{v} \in \mathcal{F}, \\ \partial_2 f(\boldsymbol{x}, \boldsymbol{W})[\boldsymbol{V}] &= \boldsymbol{V}\boldsymbol{x} \in \mathcal{F}.\end{aligned}$$

For the VJP, letting $\boldsymbol{u} \in \mathbb{R}^M$, we obtain

$$\partial f(\boldsymbol{x}, \boldsymbol{W})^*[\boldsymbol{u}] = (\boldsymbol{W}^\top \boldsymbol{u}, \boldsymbol{u}\boldsymbol{x}^\top) \in \mathcal{E}.$$

We can access the individual VJPs by

$$\begin{aligned}\partial_1 f(\boldsymbol{x}, \boldsymbol{W})^*[\boldsymbol{u}] &= \boldsymbol{W}^\top \boldsymbol{u} \in \mathcal{E}_1, \\ \partial_2 f(\boldsymbol{x}, \boldsymbol{W})^*[\boldsymbol{u}] &= \boldsymbol{u}\boldsymbol{x}^\top \in \mathcal{E}_2.\end{aligned}$$

Remark 2.4 (Nested inputs). It is sometimes convenient to group inputs in meaningful parts. For instance, if the input is naturally broken down into two parts $\boldsymbol{x} = (\boldsymbol{x}_1, \boldsymbol{x}_2)$, where $\boldsymbol{x}_1$ is a text part and $\boldsymbol{x}_2$ is an image part, and the network parameters are naturally grouped into three layers $\boldsymbol{w} = (\boldsymbol{w}_1, \boldsymbol{w}_2, \boldsymbol{w}_3)$, we can write $f(\boldsymbol{x}, \boldsymbol{w}) = f((\boldsymbol{x}_1, \boldsymbol{x}_2), (\boldsymbol{w}_1, \boldsymbol{w}_2, \boldsymbol{w}_3))$. This is mostly a convenience and we can again reduce it to a function of a single input, thanks to the linear map perspective in Euclidean spaces.

Remark 2.5 (Hiding away inputs). It will often be convenient to ignore inputs when differentiating. We use the semicolon for this purpose. For instance, a function of the form $L(\boldsymbol{w}; \boldsymbol{x}, \boldsymbol{y})$ (notice the semicolon) has signature $L \colon \mathcal{W} \to \mathbb{R}$ because we treat $\boldsymbol{x}$ and $\boldsymbol{y}$ as constants. Therefore, the gradient is $\nabla L(\boldsymbol{w}; \boldsymbol{x}, \boldsymbol{y}) \in \mathcal{W}$. On the other hand, the function $L(\boldsymbol{w}, \boldsymbol{x}, \boldsymbol{y})$ (notice the comma) has signature $L \colon \mathcal{W} \times \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ so its gradient is $\nabla L(\boldsymbol{w}, \boldsymbol{x}, \boldsymbol{y}) \in \mathcal{W} \times \mathcal{X} \times \mathcal{Y}$. If we need to access partial gradients, we use indexing, e.g., $\nabla_1 L(\boldsymbol{w}, \boldsymbol{x}, \boldsymbol{y}) \in \mathcal{W}$ or $\nabla_{\boldsymbol{w}} L(\boldsymbol{w}, \boldsymbol{x}, \boldsymbol{y}) \in \mathcal{W}$ when there is no ambiguity.

2.3.8 Functions of multiple outputs (fan-out)

Similarly, it is often convenient to deal with functions of multiple outputs.

Proposition 2.9 (Multiple outputs)**.** Consider a differentiable function of the form $f(\boldsymbol{w}) = (f_1(\boldsymbol{w}), \ldots, f_T(\boldsymbol{w}))$, with signatures $f \colon \mathcal{E} \to \mathcal{F}$ and $f_i \colon \mathcal{E} \to \mathcal{F}_i$, where $\mathcal{F} := \mathcal{F}_1 \times \cdots \times \mathcal{F}_T$. Then the JVP with the input direction $\boldsymbol{v} \in \mathcal{E}$ is given by

$$\partial f(\boldsymbol{w})[\boldsymbol{v}] = (\partial f_1(\boldsymbol{w})[\boldsymbol{v}], \ldots, \partial f_T(\boldsymbol{w})[\boldsymbol{v}]) \in \mathcal{F}.$$

The VJP with the output direction $\boldsymbol{u} = (\boldsymbol{u}_1, \ldots, \boldsymbol{u}_T) \in \mathcal{F}$ is

$$\begin{aligned}\partial f(\boldsymbol{w})^*[\boldsymbol{u}] &= \partial f(\boldsymbol{w})^*[\boldsymbol{u}_1, \ldots, \boldsymbol{u}_T] \in \mathcal{E} \\ &= \sum_{i=1}^{T} \partial f_i(\boldsymbol{w})^*[\boldsymbol{u}_i].\end{aligned}$$

Combined with the chain rule, we obtain that the Jacobian of

$$\boldsymbol{h}(\boldsymbol{w}) := g(f(\boldsymbol{w})) = g(f_1(\boldsymbol{w}), \ldots, f_T(\boldsymbol{w}))$$

is $\partial \boldsymbol{h}(\boldsymbol{w}) = \sum_{i=1}^{T} \partial_i f(g(\boldsymbol{w})) \circ \partial g_i(\boldsymbol{w})$ and therefore the JVP is

$$\partial \boldsymbol{h}(\boldsymbol{w})[\boldsymbol{v}] = \sum_{i=1}^{T} \partial_i f(g(\boldsymbol{w}))[\partial g_i(\boldsymbol{w})[\boldsymbol{v}]].$$

2.3.9 Extensions to non-Euclidean linear spaces

We focused on Euclidean spaces, i.e., linear spaces with a finite basis. However, the notions introduced earlier can be defined in more generic spaces.

For example, **directional derivatives** (see Definition 2.11) can be defined in any linear space equipped with a norm and complete with respect to this norm. Such spaces are called **Banach spaces**. Completeness is a technical assumption that requires that any Cauchy sequence converges (a Cauchy sequence is a sequence whose elements become arbitrarily close to each other as the sequence progresses). A function $f : \mathcal{E} \to \mathcal{F}$ defined from a Banach space $\mathcal{E}$ onto a Banach space $\mathcal{F}$ is then called **Gateaux differentiable** if its directional derivative is defined along any direction (where limits are defined w.r.t. the norm in $\mathcal{F}$). Some authors also require the directional derivative to be linear to define a Gateaux differentiable function.

Fréchet differentiability can also naturally be generalized to Banach spaces. The only difference is that, in generic Banach spaces, the linear map l satisfying Definition 2.11 must be continuous, i.e., there must exist $C > 0$, such that $l[\boldsymbol{v}] \leq C\|\boldsymbol{v}\|$, where $\|\cdot\|$ is the norm in the Banach space $\mathcal{E}$.

The definitions of gradient and VJPs require in addition a notion of inner product. They can be defined in **Hilbert spaces**, that is, linear spaces equipped with an inner product and complete with respect to the norm induced by the inner product (they could also be defined in a Banach space by considering operations in the dual space, see, e.g. (Clarke *et al.*, 2008)). The existence of the gradient is ensured by **Riesz's representation theorem** which states that any continuous linear form in a Hilbert space can be represented by the inner product with a vector. Since for a differentiable function $f : \mathcal{E} \to \mathbb{R}$, the JVP $\partial f(\boldsymbol{w}) : \mathcal{E} \to \mathbb{R}$ is a linear form, Riesz's representation theorem ensures the existence of the gradient as the element $g \in \mathcal{E}$ such that $\partial f(\boldsymbol{w})\boldsymbol{v} = \langle g, \boldsymbol{v}\rangle$ for any $\boldsymbol{v} \in \mathcal{E}$. The VJP is also well-defined as the adjoint of the JVP w.r.t. the inner product of the Hilbert space.

As an example, the space of squared integrable functions on $\mathbb{R}$ is a Hilbert space equipped with the inner product $\langle a, b\rangle := \int a(x)b(x)dx$. Here, we cannot find a finite number of functions that can express all possible functions on $\mathbb{R}$. Therefore, this space is not a mere Euclidean space. Nevertheless, we can consider functions on this Hilbert space (called **functionals**) to distinguish them from the elements of the space).

The associated directional derivatives and gradients, can be defined and are called respectively, **functional derivative** and **functional gradient**, see, e.g., Frigyik *et al.* (2008) and references therein.

2.4 Second-order differentiation

2.4.1 Second derivatives

For a single-input, single-output differentiable function $f : \mathbb{R} \to \mathbb{R}$, its derivative at any point is itself a function $f' : \mathbb{R} \to \mathbb{R}$. We may then consider the derivative of the derivative at any point: the **second derivative**.

Definition 2.13 (Second derivative)**.** The **second derivative** $f^{(2)}(w)$ of a differentiable function $f : \mathbb{R} \to \mathbb{R}$ at $w \in \mathbb{R}$ is defined as the derivative of f' at w, that is,

$$f^{(2)}(w) := \lim_{\delta \to 0} \frac{f'(w + \delta) - f'(w)}{\delta},$$

provided that the limit is well-defined. If the second derivative of a function f is well-defined at w, the function is said **twice differentiable** at w.

If f has a small second derivative at a given w, the derivative around w is almost constant. That is, the function behaves like a line around w, as illustrated in Fig. 2.4. Hence, the second derivative is usually interpreted as the **curvature** of the function at a given point.

2.4.2 Second directional derivatives

For a multi-input function $f : \mathbb{R}^P \to \mathbb{R}$, we saw that the directional derivative encodes infinitesimal variations of f along a given direction. To analyze the second derivative, the curvature of the function at a given point $\boldsymbol{w}$, we can consider the variations along a pair of directions, as defined below.

Definition 2.14 (Second directional derivative)**.** The **second directional derivative** of $f : \mathbb{R}^P \to \mathbb{R}$ at $\boldsymbol{w} \in \mathbb{R}^P$ along $\boldsymbol{v}, \boldsymbol{v}' \in \mathbb{R}^P$

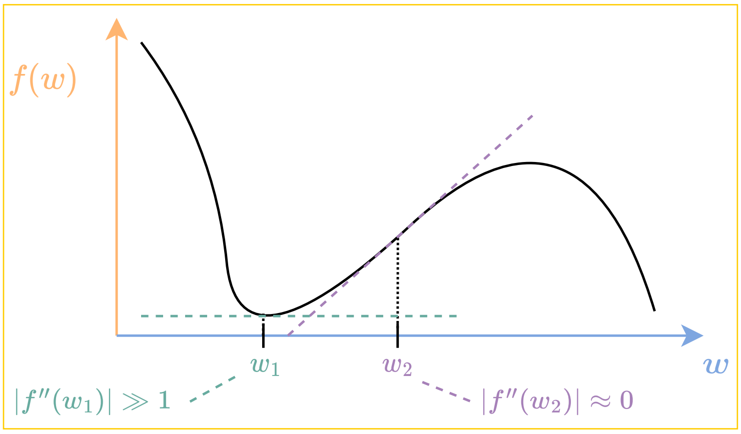

Figure 2.4: Points at which the second derivative is small are points along which the function is well approximated by its tangent line. On the other hand, point with large second derivative tend to be badly approximated by the tangent line.

> is defined as the directional derivative of $\boldsymbol{w} \mapsto \partial f(\boldsymbol{w})[\boldsymbol{v}]$ along $\boldsymbol{v}'$, that is,
>
> $$\partial^2 f(\boldsymbol{w})[\boldsymbol{v}, \boldsymbol{v}'] := \lim_{\delta \to 0} \frac{\partial f(\boldsymbol{w} + \delta \boldsymbol{v}')[\boldsymbol{v}] - \partial f(\boldsymbol{w})[\boldsymbol{v}]}{\delta},$$
>
> provided that $\partial f(\boldsymbol{w})[\boldsymbol{v}]$ is well-defined around $\boldsymbol{w}$ and that the limit exists.

Of particular interest are the variations of a function around the canonical directions: the **second partial derivatives**, defined as

$$\partial^2_{ij} f(\boldsymbol{w}) := \partial^2 f(\boldsymbol{w})[\boldsymbol{e}_i, \boldsymbol{e}_j]$$

for $\boldsymbol{e}_i$, $\boldsymbol{e}_j$ the i^{th} and j^{th} canonical directions in $\mathbb{R}^P$, respectively. In Leibniz notation, the second partial derivatives are denoted

$$\partial^2_{ij} f(\boldsymbol{w}) = \frac{\partial^2 f(\boldsymbol{w})}{\partial w_i \partial w_j}.$$

2.4.3 Hessians

For a multi-input function, twice differentiability is simply defined as the differentiability of any directional derivative $\partial f(\boldsymbol{w})[\boldsymbol{v}]$ w.r.t. $\boldsymbol{w}$.