Mathematics for Machine Learning

Marc Deisenroth

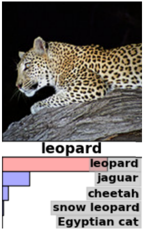
Statistical Machine Learning Group
Department of Computing
Imperial College London

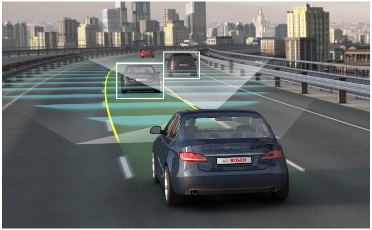
@mpd37
m.deisenroth@imperial.ac.uk
marc@prowler.io

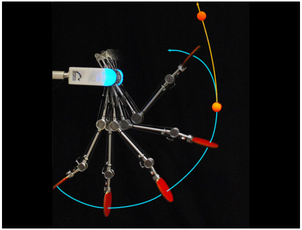
Deep Learning Indaba
University of the Witwatersrand
Johannesburg, South Africa

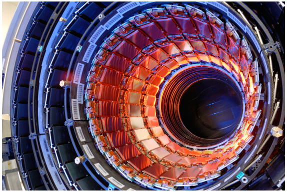
September 10, 2017

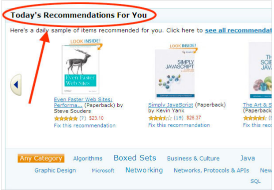
Applications of Machine Learning

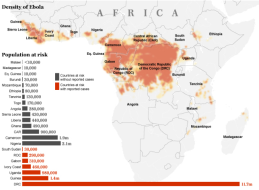




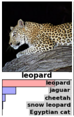


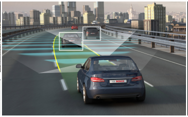
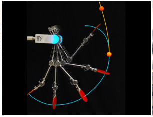
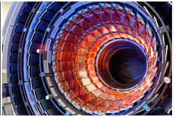
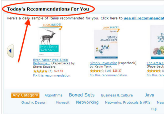


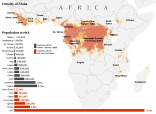




Mathematical Concepts in Machine Learning





- Linear algebra and matrix decomposition
- **Differentiation**
- Optimization
- **Integration**
- Probability theory and Bayesian inference
- Functional analysis

Outline

Overview

Introduction

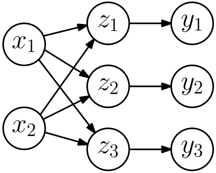
Differentiation

Integration

Feedforward Neural Network

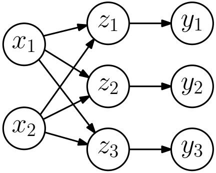
$$\boldsymbol{y} = \sigma(\boldsymbol{z})$$

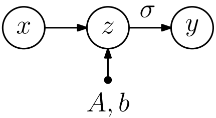
$$\boldsymbol{z} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}$$



Feedforward Neural Network

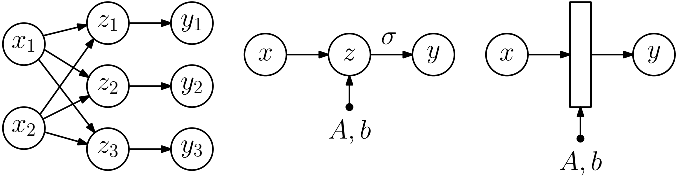
$$\boldsymbol{y} = \sigma(\boldsymbol{z})$$
$$\boldsymbol{z} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}$$





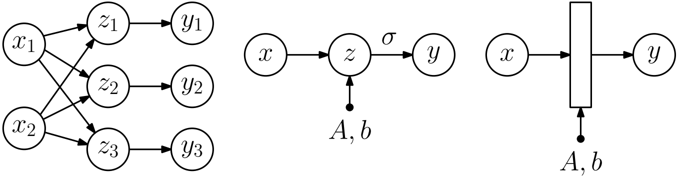
Feedforward Neural Network

$$\boldsymbol{y} = \sigma(\boldsymbol{z})$$
$$\boldsymbol{z} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}$$



Feedforward Neural Network

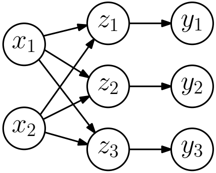
$$\boldsymbol{y} = \sigma(\boldsymbol{z})$$
$$\boldsymbol{z} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}$$

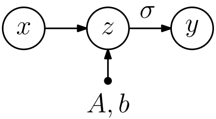


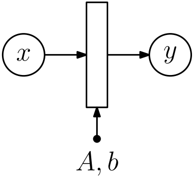
- Training a neural network means parameter optimization: Typically via some form of gradient descent
  ▶▶ **Challenge 1: Differentiation.** Compute gradients of a loss function with respect to neural network parameters $\boldsymbol{A}, \boldsymbol{b}$

Feedforward Neural Network

$$\boldsymbol{y} = \sigma(\boldsymbol{z})$$
$$\boldsymbol{z} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}$$



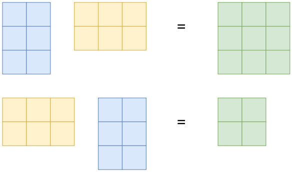




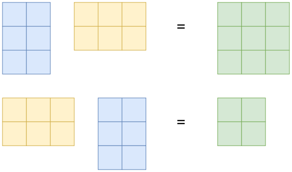
- Training a neural network means parameter optimization: Typically via some form of gradient descent
  ▶▶ **Challenge 1: Differentiation.** Compute gradients of a loss function with respect to neural network parameters $\boldsymbol{A}, \boldsymbol{b}$
- Computing statistics (e.g., means, variances) of predictions
  ▶▶ **Challenge 2: Integration.** Propagate uncertainty through a neural network

Background: Matrix Multiplication

- Matrix multiplication is not commutative, i.e., $\boldsymbol{AB} \neq \boldsymbol{BA}$



Background: Matrix Multiplication

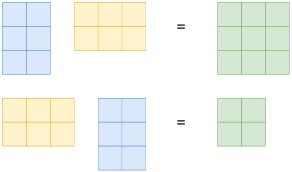


- Matrix multiplication is not commutative, i.e., $\boldsymbol{A}\boldsymbol{B} \neq \boldsymbol{B}\boldsymbol{A}$

- When multiplying matrices, the "neighboring" dimensions have to fit:

$$\underbrace{\boldsymbol{A}}_{n \times k} \underbrace{\boldsymbol{B}}_{k \times m} = \underbrace{\boldsymbol{C}}_{n \times m}$$

Background: Matrix Multiplication



- Matrix multiplication is not commutative, i.e., $\boldsymbol{A}\boldsymbol{B} \neq \boldsymbol{B}\boldsymbol{A}$

- When multiplying matrices, the "neighboring" dimensions have to fit:

$$\underbrace{\boldsymbol{A}}_{n\times k}\;\underbrace{\boldsymbol{B}}_{k\times m} = \underbrace{\boldsymbol{C}}_{n\times m}$$

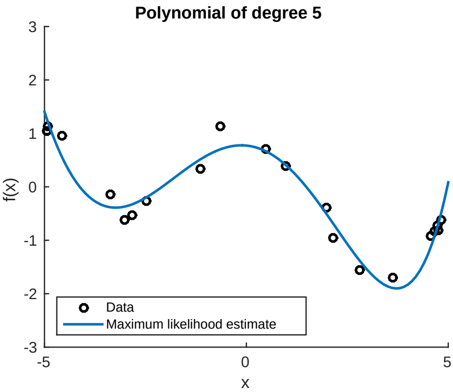
$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$ `y = A.dot(x)`

$y_i = \sum_j A_{ij}x_j$ `y = np.einsum('ij, j', A, x)`

$\boldsymbol{C} = \boldsymbol{A}\boldsymbol{B}$ `C = A.dot(B)`

$C_{ij} = \sum_k A_{ik}B_{kj}$ `C = np.einsum('ik, kj', A, B)`

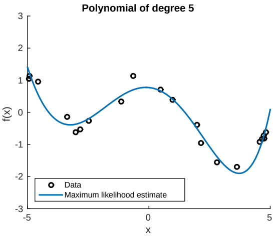
Curve Fitting (Regression) in Machine Learning (1)



- Setting: Given inputs $\boldsymbol{x}$, predict outputs/targets $\boldsymbol{y}$
- Model f that depends on parameters $\boldsymbol{\theta}$. Examples:
    - Linear model: $f(\boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{\theta}^\top \boldsymbol{x}, \quad \boldsymbol{x}, \boldsymbol{\theta} \in \mathbb{R}^D$
    - Neural network: $f(\boldsymbol{x}, \boldsymbol{\theta}) = NN(\boldsymbol{x}, \boldsymbol{\theta})$

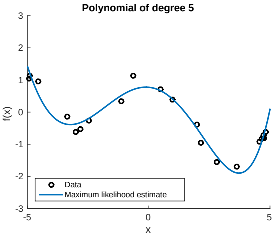
Curve Fitting (Regression) in Machine Learning (2)

- Training data, e.g., N pairs $(\boldsymbol{x}_i, y_i)$ of inputs $\boldsymbol{x}_i$ and observations y_i
- Training the model means finding parameters $\boldsymbol{\theta}^*$, such that $f(\boldsymbol{x}_i, \boldsymbol{\theta}^*) \approx y_i$



Curve Fitting (Regression) in Machine Learning (2)

- Training data, e.g., N pairs $(\boldsymbol{x}_i, y_i)$ of inputs $\boldsymbol{x}_i$ and observations y_i
- Training the model means finding parameters $\boldsymbol{\theta}^*$, such that $f(\boldsymbol{x}_i, \boldsymbol{\theta}^*) \approx y_i$



- Define a loss function, e.g., $\sum_{i=1}^{N}(y_i - f(\boldsymbol{x}_i, \boldsymbol{\theta}))^2$, which we want to optimize
- Typically: Optimization based on some form of gradient descent
  ▸▸ Differentiation required

Overview

Introduction

Differentiation

Integration

Differentiation: Outline

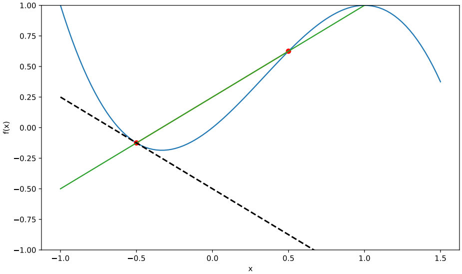
1. Scalar differentiation: $f : \mathbb{R} \to \mathbb{R}$
2. Multivariate case: $f : \mathbb{R}^N \to \mathbb{R}$
3. Vector fields: $f : \mathbb{R}^N \to \mathbb{R}^M$
4. General derivatives: $f : \mathbb{R}^{M \times N} \to \mathbb{R}^{P \times Q}$

Scalar Differentiation $f : \mathbb{R} \to \mathbb{R}$

- Derivative defined as the limit of the difference quotient

$$f'(x) = \frac{df}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

▶▶ Slope of the secant line through $f(x)$ and $f(x+h)$



Examples

$$
\begin{aligned}
f(x) &= x^n & f'(x) &= nx^{n-1} \\
f(x) &= \sin(x) & f'(x) &= \cos(x) \\
f(x) &= \tanh(x) & f'(x) &= 1 - \tanh^2(x) \\
f(x) &= \exp(x) & f'(x) &= \exp(x) \\
f(x) &= \log(x) & f'(x) &= \frac{1}{x}
\end{aligned}
$$

Rules

- Sum Rule

$$\big(f(x) + g(x)\big)' = f'(x) + g'(x) = \frac{df}{dx} + \frac{dg}{dx}$$

Rules

- Sum Rule

$$\big(f(x)+g(x)\big)' = f'(x) + g'(x) = \frac{df}{dx} + \frac{dg}{dx}$$

- Product Rule

$$\big(f(x)g(x)\big)' = f'(x)g(x) + f(x)g'(x) = \frac{df}{dx}g(x) + f(x)\frac{dg}{dx}$$

Rules

- Sum Rule

$$\big(f(x)+g(x)\big)' = f'(x) + g'(x) = \frac{df}{dx} + \frac{dg}{dx}$$

- Product Rule

$$\big(f(x)g(x)\big)' = f'(x)g(x) + f(x)g'(x) = \frac{df}{dx}g(x) + f(x)\frac{dg}{dx}$$

- Chain Rule

$$(g \circ f)'(x) = \big(g(f(x))\big)' = g'(f(x))f'(x) = \frac{dg}{df}\frac{df}{dx}$$

Example: Chain Rule

$$(g \circ f)'(x) = \big(g(f(x))\big)' = g'(f(x))f'(x) = \frac{dg}{df}\frac{df}{dx}$$

$$\begin{aligned} g(z) &= \tanh(z) \\ z &= f(x) = x^n \\ (g \circ f)'(x) &= \end{aligned}$$

Example: Chain Rule

$$(g \circ f)'(x) = \big(g(f(x))\big)' = g'(f(x))f'(x) = \frac{dg}{df}\frac{df}{dx}$$

$$\begin{aligned} g(z) &= \tanh(z) \\ z &= f(x) = x^n \\ (g \circ f)'(x) &= \underbrace{\left(1 - \tanh^2(x^n)\right)}_{dg/df} \underbrace{nx^{n-1}}_{df/dx} \end{aligned}$$

$f : \mathbb{R}^N \to \mathbb{R}$

$$y = f(\boldsymbol{x}) \,, \quad \boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix} \in \mathbb{R}^N$$

- Partial derivative (change one coordinate at a time):

$$\frac{\partial f}{\partial x_i} = \lim_{h \to 0} \frac{f(x_1, \dots, x_{i-1}, x_i + h, x_{i+1}, \dots, x_N) - f(\boldsymbol{x})}{h}$$

$f : \mathbb{R}^N \to \mathbb{R}$

$$y = f(\boldsymbol{x}), \quad \boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix} \in \mathbb{R}^N$$

- Partial derivative (change one coordinate at a time):

$$\frac{\partial f}{\partial x_i} = \lim_{h \to 0} \frac{f(x_1, \ldots, x_{i-1}, x_i + h, x_{i+1}, \ldots, x_N) - f(\boldsymbol{x})}{h}$$

- Jacobian vector (gradient) collects all partial derivatives:

$$\frac{df}{d\boldsymbol{x}} = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \cdots & \frac{\partial f}{x_N} \end{bmatrix} \in \mathbb{R}^{1 \times N}$$

Note: This is a row vector.

Example

$$f : \mathbb{R}^2 \to \mathbb{R}$$
$$f(x_1, x_2) = x_1^2 x_2 + x_1 x_2^3 \in \mathbb{R}$$

Example

$$f : \mathbb{R}^2 \to \mathbb{R}$$
$$f(x_1, x_2) = x_1^2 x_2 + x_1 x_2^3 \in \mathbb{R}$$

- Partial derivatives:

$$\frac{\partial f(x_1, x_2)}{\partial x_1} = 2x_1 x_2 + x_2^3$$
$$\frac{\partial f(x_1, x_2)}{\partial x_2} = x_1^2 + 3x_1 x_2^2$$

Example

$$f : \mathbb{R}^2 \to \mathbb{R}$$
$$f(x_1, x_2) = x_1^2 x_2 + x_1 x_2^3 \in \mathbb{R}$$

- Partial derivatives:

$$\frac{\partial f(x_1, x_2)}{\partial x_1} = 2x_1 x_2 + x_2^3$$
$$\frac{\partial f(x_1, x_2)}{\partial x_2} = x_1^2 + 3x_1 x_2^2$$

- Gradient:

$$\frac{df}{d\boldsymbol{x}} = \begin{bmatrix} \frac{\partial f(x_1,x_2)}{\partial x_1} & \frac{\partial f(x_1,x_2)}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 2x_1 x_2 + x_2^3 & x_1^2 + 3x_1 x_2^2 \end{bmatrix} \in \mathbb{R}^{1\times 2}.$$

Rules

- Sum Rule

$$\frac{\partial}{\partial \boldsymbol{x}}\big(f(\boldsymbol{x}) + g(\boldsymbol{x})\big) = \frac{\partial f}{\partial \boldsymbol{x}} + \frac{\partial g}{\partial \boldsymbol{x}}$$

- Product Rule

$$\frac{\partial}{\partial \boldsymbol{x}}\big(f(\boldsymbol{x})g(\boldsymbol{x})\big) = \frac{\partial f}{\partial \boldsymbol{x}}g(\boldsymbol{x}) + f(\boldsymbol{x})\frac{\partial g}{\partial \boldsymbol{x}}$$

- Chain Rule

$$\frac{\partial}{\partial \boldsymbol{x}}(g \circ f)(\boldsymbol{x}) = \frac{\partial}{\partial \boldsymbol{x}}\big(g(f(\boldsymbol{x}))\big) = \frac{\partial g}{\partial f}\frac{\partial f}{\partial \boldsymbol{x}}$$

Example: Chain Rule

- Consider the function

$$L(\boldsymbol{e}) = \tfrac{1}{2}\|\boldsymbol{e}\|^2 = \tfrac{1}{2}\boldsymbol{e}^\top \boldsymbol{e}$$

$$\boldsymbol{e} = \boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\,, \quad \boldsymbol{x} \in \mathbb{R}^N\,, \boldsymbol{A} \in \mathbb{R}^{M\times N}\,, \boldsymbol{e}, \boldsymbol{y} \in \mathbb{R}^M$$

- Compute $dL/d\boldsymbol{x}$. What is the dimension/size of $dL/d\boldsymbol{x}$?

Example: Chain Rule

- Consider the function

$$L(\boldsymbol{e}) = \tfrac{1}{2}\|\boldsymbol{e}\|^2 = \tfrac{1}{2}\boldsymbol{e}^\top \boldsymbol{e}$$

$$\boldsymbol{e} = \boldsymbol{y} - \boldsymbol{A}\boldsymbol{x}\,, \quad \boldsymbol{x} \in \mathbb{R}^N\,, \boldsymbol{A} \in \mathbb{R}^{M\times N}\,, \boldsymbol{e}, \boldsymbol{y} \in \mathbb{R}^M$$

- Compute $dL/d\boldsymbol{x}$. What is the dimension/size of $dL/d\boldsymbol{x}$?
- $dL/d\boldsymbol{x} \in \mathbb{R}^{1\times N}$

$$\frac{dL}{d\boldsymbol{x}} = \frac{dL}{d\boldsymbol{e}}\frac{d\boldsymbol{e}}{d\boldsymbol{x}}$$

$$\frac{dL}{d\boldsymbol{e}} = \boldsymbol{e}^\top \in \mathbb{R}^{1\times M} \tag{1}$$

$$\frac{d\boldsymbol{e}}{d\boldsymbol{x}} = -\boldsymbol{A} \in \mathbb{R}^{M\times N} \tag{2}$$

$$\Rightarrow \frac{dL}{d\boldsymbol{x}} = \boldsymbol{e}^\top(-\boldsymbol{A}) = -(\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x})^\top \boldsymbol{A} \in \mathbb{R}^{1\times N}$$

$f : \mathbb{R}^N \to \mathbb{R}^M$

$$\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}) \in \mathbb{R}^M, \quad \boldsymbol{x} \in \mathbb{R}^N$$

$$\begin{bmatrix} y_1 \\ \vdots \\ y_M \end{bmatrix} = \begin{bmatrix} f_1(\boldsymbol{x}) \\ \vdots \\ f_M(\boldsymbol{x}) \end{bmatrix} = \begin{bmatrix} f_1(x_1, \ldots, x_N) \\ \vdots \\ f_M(x_1, \ldots, x_N) \end{bmatrix}$$

$f : \mathbb{R}^N \to \mathbb{R}^M$

$$\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}) \in \mathbb{R}^M\,, \quad \boldsymbol{x} \in \mathbb{R}^N$$

$$\begin{bmatrix} y_1 \\ \vdots \\ y_M \end{bmatrix} = \begin{bmatrix} f_1(\boldsymbol{x}) \\ \vdots \\ f_M(\boldsymbol{x}) \end{bmatrix} = \begin{bmatrix} f_1(x_1, \ldots, x_N) \\ \vdots \\ f_M(x_1, \ldots, x_N) \end{bmatrix}$$

- Jacobian matrix (collection of all partial derivatives)

$$\begin{bmatrix} \frac{dy_1}{d\boldsymbol{x}} \\ \vdots \\ \frac{dy_M}{d\boldsymbol{x}} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_N} \\ \vdots & & \vdots \\ \frac{\partial f_M}{\partial x_1} & \cdots & \frac{\partial f_M}{\partial x_N} \end{bmatrix} \in \mathbb{R}^{M \times N}$$

Example

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x}\,, \qquad \boldsymbol{f}(\boldsymbol{x}) \in \mathbb{R}^M, \quad \boldsymbol{A} \in \mathbb{R}^{M\times N}, \quad \boldsymbol{x} \in \mathbb{R}^N$$

- Compute the gradient $d\boldsymbol{f}/d\boldsymbol{x}$
  - Dimension of $d\boldsymbol{f}/d\boldsymbol{x}$:

Example

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x}\,, \qquad \boldsymbol{f}(\boldsymbol{x}) \in \mathbb{R}^M, \quad \boldsymbol{A} \in \mathbb{R}^{M\times N}, \quad \boldsymbol{x} \in \mathbb{R}^N$$

- Compute the gradient $d\boldsymbol{f}/d\boldsymbol{x}$
  - Dimension of $d\boldsymbol{f}/d\boldsymbol{x}$:
    Since $\boldsymbol{f} : \mathbb{R}^N \to \mathbb{R}^M$, it follows that $d\boldsymbol{f}/d\boldsymbol{x} \in \mathbb{R}^{M\times N}$

Example

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x}\,, \qquad \boldsymbol{f}(\boldsymbol{x}) \in \mathbb{R}^M, \quad \boldsymbol{A} \in \mathbb{R}^{M\times N}, \quad \boldsymbol{x} \in \mathbb{R}^N$$

- Compute the gradient $d\boldsymbol{f}/d\boldsymbol{x}$
  - Dimension of $d\boldsymbol{f}/d\boldsymbol{x}$:
    Since $\boldsymbol{f} : \mathbb{R}^N \to \mathbb{R}^M$, it follows that $d\boldsymbol{f}/d\boldsymbol{x} \in \mathbb{R}^{M\times N}$
  - Gradient:

$$f_i = \sum_{j=1}^{N} A_{ij}x_j \qquad \Rightarrow \frac{\partial f_i}{\partial x_j} = A_{ij} \tag{3}$$

Example

$$\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x}\,, \qquad \boldsymbol{f}(\boldsymbol{x}) \in \mathbb{R}^M, \quad \boldsymbol{A} \in \mathbb{R}^{M\times N}, \quad \boldsymbol{x} \in \mathbb{R}^N$$

- Compute the gradient $d\boldsymbol{f}/d\boldsymbol{x}$
  - Dimension of $d\boldsymbol{f}/d\boldsymbol{x}$:
    Since $\boldsymbol{f} : \mathbb{R}^N \to \mathbb{R}^M$, it follows that $d\boldsymbol{f}/d\boldsymbol{x} \in \mathbb{R}^{M\times N}$
  - Gradient:

$$f_i = \sum_{j=1}^{N} A_{ij}x_j \qquad \Rightarrow \frac{\partial f_i}{\partial x_j} = A_{ij}$$

$$\Rightarrow \frac{d\boldsymbol{f}}{d\boldsymbol{x}} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_N} \\ \vdots & & \vdots \\ \frac{\partial f_M}{\partial x_1} & \cdots & \frac{\partial f_M}{\partial x_N} \end{bmatrix} = \begin{bmatrix} A_{11} & \cdots & A_{1N} \\ \vdots & & \vdots \\ A_{M1} & \cdots & A_{MN} \end{bmatrix} = \boldsymbol{A} \tag{3}$$

Chain Rule

$$\frac{\partial}{\partial \boldsymbol{x}}(g \circ f)(\boldsymbol{x}) = \frac{\partial}{\partial \boldsymbol{x}}\big(g(f(\boldsymbol{x}))\big) = \frac{\partial g}{\partial f}\frac{\partial f}{\partial \boldsymbol{x}}$$

Example

- Consider $f : \mathbb{R}^2 \to \mathbb{R}, \quad \boldsymbol{x} : \mathbb{R} \to \mathbb{R}^2$

$$f(\boldsymbol{x}) = f(x_1, x_2) = x_1^2 + 2x_2\,,$$

$$\boldsymbol{x}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} = \begin{bmatrix} \sin(t) \\ \cos(t) \end{bmatrix}$$

Example

- Consider $f : \mathbb{R}^2 \to \mathbb{R}, \quad \boldsymbol{x} : \mathbb{R} \to \mathbb{R}^2$

$$f(\boldsymbol{x}) = f(x_1, x_2) = x_1^2 + 2x_2\,,$$

$$\boldsymbol{x}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} = \begin{bmatrix} \sin(t) \\ \cos(t) \end{bmatrix}$$

- The dimensions $df/d\boldsymbol{x}$ and $d\boldsymbol{x}/dt$ are

Example

- Consider $f : \mathbb{R}^2 \to \mathbb{R}, \quad \boldsymbol{x} : \mathbb{R} \to \mathbb{R}^2$

$$f(\boldsymbol{x}) = f(x_1, x_2) = x_1^2 + 2x_2\,,$$

$$\boldsymbol{x}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} = \begin{bmatrix} \sin(t) \\ \cos(t) \end{bmatrix}$$

- The dimensions $df/d\boldsymbol{x}$ and $d\boldsymbol{x}/dt$ are 1×2 and 2×1, respectively
- Compute the gradient df/dt using the chain rule.

Example

- Consider $f : \mathbb{R}^2 \to \mathbb{R}, \quad \boldsymbol{x} : \mathbb{R} \to \mathbb{R}^2$

$$f(\boldsymbol{x}) = f(x_1, x_2) = x_1^2 + 2x_2\,,$$

$$\boldsymbol{x}(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} = \begin{bmatrix} \sin(t) \\ \cos(t) \end{bmatrix}$$

- The dimensions $df/d\boldsymbol{x}$ and $d\boldsymbol{x}/dt$ are 1×2 and 2×1, respectively
- Compute the gradient df/dt using the chain rule.

$$\frac{df}{dt} = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} \end{bmatrix} \begin{bmatrix} \frac{\partial x_1}{\partial t} \\ \frac{\partial x_2}{\partial t} \end{bmatrix}$$

$$= \begin{bmatrix} 2 \sin t & 2 \end{bmatrix} \begin{bmatrix} \cos t \\ -\sin t \end{bmatrix}$$

$$= 2 \sin t \cos t - 2 \sin t = 2 \sin t (\cos t - 1)$$

BREAK

Derivatives with Matrices

- Re-cap: Gradient of a function $f : \mathbb{R}^D \to \mathbb{R}^E$ is an $E \times D$-matrix:

  # target dimensions × # parameters

  with

$$\frac{d\boldsymbol{f}}{d\boldsymbol{x}} \in \mathbb{R}^{E \times D}, \qquad df[e,d] = \frac{\partial f_e}{\partial x_d}$$

Derivatives with Matrices

- Re-cap: Gradient of a function $f : \mathbb{R}^D \rightarrow \mathbb{R}^E$ is an $E \times D$-matrix:

  # target dimensions $\times$ # parameters

  with

$$\frac{d\boldsymbol{f}}{d\boldsymbol{x}} \in \mathbb{R}^{E \times D}, \qquad df[e,d] = \frac{\partial f_e}{\partial x_d}$$

- Generalization to cases, where the parameters (D) or targets (E) are matrices, apply immediately

Derivatives with Matrices

- Re-cap: Gradient of a function $f : \mathbb{R}^D \to \mathbb{R}^E$ is an $E \times D$-matrix:

  # target dimensions × # parameters

  with

$$\frac{d\boldsymbol{f}}{d\boldsymbol{x}} \in \mathbb{R}^{E\times D}, \qquad df[e,d] = \frac{\partial f_e}{\partial x_d}$$

- Generalization to cases, where the parameters (D) or targets (E) are matrices, apply immediately
- Assume $f : \mathbb{R}^{M\times N} \to \mathbb{R}^{P\times Q}$, then the gradient is a $(P \times Q) \times (M \times N)$ object (tensor) where

$$df[p,q,m,n] = \frac{\partial f_{pq}}{\partial X_{mn}}$$

Derivatives with Matrices: Example (1)

$$\boldsymbol{f} = \boldsymbol{A}\boldsymbol{x}\,, \quad \boldsymbol{f} \in \mathbb{R}^M, \boldsymbol{A} \in \mathbb{R}^{M\times N}, \boldsymbol{x} \in \mathbb{R}^N$$

Derivatives with Matrices: Example (1)

$$\boldsymbol{f} = \boldsymbol{A}\boldsymbol{x}\,, \quad \boldsymbol{f} \in \mathbb{R}^M, \boldsymbol{A} \in \mathbb{R}^{M\times N}, \boldsymbol{x} \in \mathbb{R}^N$$

$$\frac{d\boldsymbol{f}}{d\boldsymbol{A}} \in \mathbb{R}^{M\times(M\times N)}$$

$$\frac{d\boldsymbol{f}}{d\boldsymbol{A}} = \begin{bmatrix} \frac{\partial f_1}{\partial \boldsymbol{A}} \\ \vdots \\ \frac{\partial f_M}{\partial \boldsymbol{A}} \end{bmatrix}, \quad \frac{\partial f_i}{\partial \boldsymbol{A}} \in \mathbb{R}^{1\times(M\times N)}$$

Derivatives with Matrices: Example (2)

$$f_i = \sum_{j=1}^{N} A_{ij} x_j, \quad i = 1, \ldots, M \tag{4}$$

Derivatives with Matrices: Example (2)

$$f_i = \sum_{j=1}^{N} A_{ij} x_j, \quad i = 1, \ldots, M$$

$$\frac{\partial f_i}{\partial A_{iq}} = x_q \tag{4}$$

Derivatives with Matrices: Example (2)

$$f_i = \sum_{j=1}^{N} A_{ij} x_j, \quad i = 1, \dots, M$$

$$\frac{\partial f_i}{\partial A_{iq}} = x_q \Rightarrow \frac{\partial f_i}{\partial A_{i,:}} = \boldsymbol{x}^\top \in \mathbb{R}^{1\times 1\times N} \tag{4}$$

Derivatives with Matrices: Example (2)

$$
\begin{aligned}
f_i &= \sum_{j=1}^{N} A_{ij} x_j, \quad i = 1, \dots, M \\
\frac{\partial f_i}{\partial A_{iq}} &= x_q \Rightarrow \frac{\partial f_i}{\partial A_{i,:}} = \boldsymbol{x}^\top \in \mathbb{R}^{1\times 1\times N} \\
\frac{\partial f_i}{\partial A_{k\neq i,:}} &= \mathbf{0}^\top \in \mathbb{R}^{1\times 1\times N}
\end{aligned} \tag{4}
$$

Derivatives with Matrices: Example (2)

$$f_i = \sum_{j=1}^{N} A_{ij} x_j, \quad i = 1, \ldots, M$$

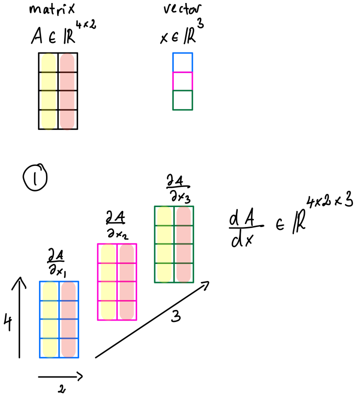
$$\frac{\partial f_i}{\partial A_{iq}} = x_q \Rightarrow \frac{\partial f_i}{\partial A_{i,:}} = \boldsymbol{x}^\top \in \mathbb{R}^{1\times 1\times N}$$

$$\frac{\partial f_i}{\partial A_{k\neq i,:}} = \mathbf{0}^\top \in \mathbb{R}^{1\times 1\times N}$$

$$\frac{\partial f_i}{\partial \boldsymbol{A}} = \begin{bmatrix} \mathbf{0}^\top \\ \vdots \\ \boldsymbol{x}^\top \\ \mathbf{0}^\top \\ \vdots \\ \mathbf{0}^\top \end{bmatrix} \in \mathbb{R}^{1\times(M\times N)} \tag{4}$$

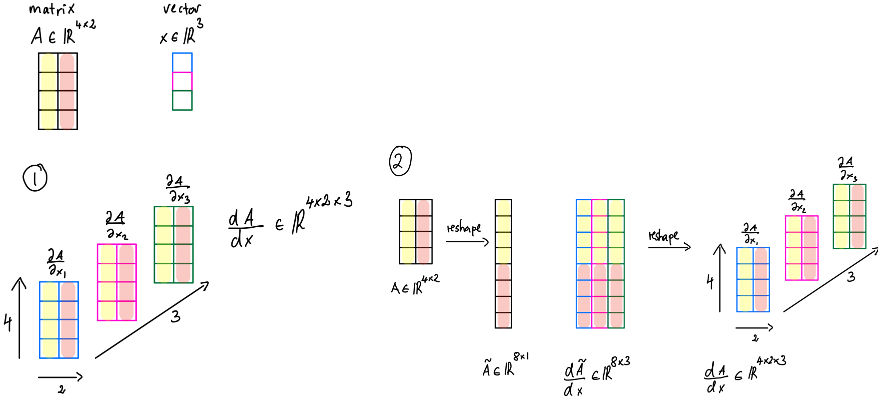
Example: Higher-Order Tensors

- Consider a matrix $\boldsymbol{A} \in \mathbb{R}^{4\times 2}$ whose entries depend on a vector $\boldsymbol{x} \in \mathbb{R}^3$
- We can compute $d\boldsymbol{A}(\boldsymbol{x})/d\boldsymbol{x} \in \mathbb{R}^{4\times 2\times 3}$ in two equivalent ways:

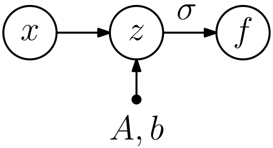


Example: Higher-Order Tensors

- Consider a matrix $\boldsymbol{A} \in \mathbb{R}^{4\times 2}$ whose entries depend on a vector $\boldsymbol{x} \in \mathbb{R}^3$
- We can compute $d\boldsymbol{A}(\boldsymbol{x})/d\boldsymbol{x} \in \mathbb{R}^{4\times 2\times 3}$ in two equivalent ways:

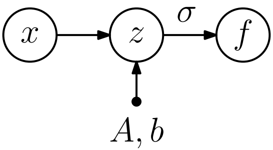


Gradients of a Single-Layer Neural Network (1)



$$\boldsymbol{f} = \tanh(\underbrace{\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}}_{=:\boldsymbol{z}\in\mathbb{R}^M}) \in \mathbb{R}^M, \quad \boldsymbol{x} \in \mathbb{R}^N, \boldsymbol{A} \in \mathbb{R}^{M\times N}, \boldsymbol{b} \in \mathbb{R}^M$$

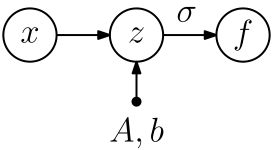
Gradients of a Single-Layer Neural Network (1)



$$\boldsymbol{f} = \tanh(\underbrace{\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}}_{=:\boldsymbol{z}\in\mathbb{R}^M}) \in \mathbb{R}^M, \quad \boldsymbol{x} \in \mathbb{R}^N, \boldsymbol{A} \in \mathbb{R}^{M\times N}, \boldsymbol{b} \in \mathbb{R}^M$$

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{b}} = \underbrace{\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}}_{M\times M} \underbrace{\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}}_{M\times M} \in \mathbb{R}^{M\times M}$$

Gradients of a Single-Layer Neural Network (1)



$$\boldsymbol{f} = \tanh(\underbrace{\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}}_{=:\boldsymbol{z} \in \mathbb{R}^M}) \in \mathbb{R}^M, \quad \boldsymbol{x} \in \mathbb{R}^N, \boldsymbol{A} \in \mathbb{R}^{M \times N}, \boldsymbol{b} \in \mathbb{R}^M$$

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{b}} = \underbrace{\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}}_{M \times M} \underbrace{\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}}_{M \times M} \in \mathbb{R}^{M \times M}$$

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{A}} = \underbrace{\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}}_{M \times M} \underbrace{\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{A}}}_{M \times (M \times N)} \in \mathbb{R}^{M \times (M \times N)}$$

Gradients of a Single-Layer Neural Network (2)

$$\boldsymbol{f} = \tanh(\underbrace{\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}}_{=:\boldsymbol{z} \in \mathbb{R}^M}) \in \mathbb{R}^M, \quad \boldsymbol{x} \in \mathbb{R}^N, \boldsymbol{A} \in \mathbb{R}^{M \times N}, \boldsymbol{b} \in \mathbb{R}^M$$

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{b}} = \underbrace{\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}}_{M \times M} \underbrace{\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}}_{M \times M} \in \mathbb{R}^{M \times M}$$

Gradients of a Single-Layer Neural Network (2)

$$\boldsymbol{f} = \tanh(\underbrace{\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}}_{=:\boldsymbol{z} \in \mathbb{R}^M}) \in \mathbb{R}^M, \quad \boldsymbol{x} \in \mathbb{R}^N, \boldsymbol{A} \in \mathbb{R}^{M \times N}, \boldsymbol{b} \in \mathbb{R}^M$$

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{b}} = \underbrace{\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}}_{M \times M} \underbrace{\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}}_{M \times M} \in \mathbb{R}^{M \times M}$$

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}} = \operatorname{diag}(1 - \tanh^2(\boldsymbol{z})) \in \mathbb{R}^{M \times M}$$

Gradients of a Single-Layer Neural Network (2)

$$\boldsymbol{f} = \tanh(\underbrace{\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}}_{=:\boldsymbol{z} \in \mathbb{R}^M}) \in \mathbb{R}^M, \quad \boldsymbol{x} \in \mathbb{R}^N, \boldsymbol{A} \in \mathbb{R}^{M \times N}, \boldsymbol{b} \in \mathbb{R}^M$$

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{b}} = \underbrace{\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}}_{M \times M} \underbrace{\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}}_{M \times M} \in \mathbb{R}^{M \times M}$$

$$\begin{aligned} \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}} &= \operatorname{diag}(1 - \tanh^2(\boldsymbol{z})) \in \mathbb{R}^{M \times M} \\ \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}} &= \boldsymbol{I} \in \mathbb{R}^{M \times M} \\ \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{b}}[i,j] &= \sum_{l=1}^{M} \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}[i,l] \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}[l,j] \end{aligned} \tag{5}$$

Gradients of a Single-Layer Neural Network (2)

$$\boldsymbol{f} = \tanh(\underbrace{\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}}_{=:\boldsymbol{z}\in\mathbb{R}^M}) \in \mathbb{R}^M, \quad \boldsymbol{x} \in \mathbb{R}^N, \boldsymbol{A} \in \mathbb{R}^{M\times N}, \boldsymbol{b} \in \mathbb{R}^M$$

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{b}} = \underbrace{\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}}_{M\times M} \underbrace{\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}}_{M\times M} \in \mathbb{R}^{M\times M}$$

$$\begin{aligned} \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}} &= \operatorname{diag}(1 - \tanh^2(\boldsymbol{z})) \in \mathbb{R}^{M\times M} \\ \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}} &= \boldsymbol{I} \in \mathbb{R}^{M\times M} \\ \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{b}}[i,j] &= \sum_{l=1}^{M} \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}[i,l] \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}[l,j] \end{aligned} \tag{5}$$

```
dfdb = np.einsum('il, lj', dfdz, dzdb)
```

Gradients of a Single-Layer Neural Network (3)

$$\boldsymbol{f} = \tanh(\underbrace{\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}}_{=:\boldsymbol{z} \in \mathbb{R}^M}) \in \mathbb{R}^M, \quad \boldsymbol{x} \in \mathbb{R}^N, \boldsymbol{A} \in \mathbb{R}^{M \times N}, \boldsymbol{b} \in \mathbb{R}^M$$

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{A}} = \underbrace{\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}}_{M \times M} \underbrace{\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{A}}}_{M \times (M \times N)} \in \mathbb{R}^{M \times (M \times N)}$$

Gradients of a Single-Layer Neural Network (3)

$$\boldsymbol{f} = \tanh(\underbrace{\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}}_{=:\boldsymbol{z}\in\mathbb{R}^M}) \in \mathbb{R}^M, \quad \boldsymbol{x} \in \mathbb{R}^N, \boldsymbol{A} \in \mathbb{R}^{M\times N}, \boldsymbol{b} \in \mathbb{R}^M$$

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{A}} = \underbrace{\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}}_{M\times M} \underbrace{\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{A}}}_{M\times(M\times N)} \in \mathbb{R}^{M\times(M\times N)}$$

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}} = \mathrm{diag}(1 - \tanh^2(\boldsymbol{z})) \in \mathbb{R}^{M\times M} \tag{6}$$

Gradients of a Single-Layer Neural Network (3)

$$\boldsymbol{f} = \tanh(\underbrace{\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}}_{=:\boldsymbol{z}\in\mathbb{R}^M}) \in \mathbb{R}^M, \quad \boldsymbol{x} \in \mathbb{R}^N, \boldsymbol{A} \in \mathbb{R}^{M\times N}, \boldsymbol{b} \in \mathbb{R}^M$$

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{A}} = \underbrace{\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}}_{M\times M} \underbrace{\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{A}}}_{M\times(M\times N)} \in \mathbb{R}^{M\times(M\times N)}$$

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}} = \mathrm{diag}(1 - \tanh^2(\boldsymbol{z})) \in \mathbb{R}^{M\times M} \tag{6}$$

$$\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{A}}$$ ▶▶ See (4)

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{A}}[i,j,k] = \sum_{l=1}^{M} \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}[i,l] \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{A}}[l,j,k]$$

Gradients of a Single-Layer Neural Network (3)

$$\boldsymbol{f} = \tanh(\underbrace{\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}}_{=:\boldsymbol{z}\in\mathbb{R}^M}) \in \mathbb{R}^M, \quad \boldsymbol{x} \in \mathbb{R}^N, \boldsymbol{A} \in \mathbb{R}^{M\times N}, \boldsymbol{b} \in \mathbb{R}^M$$

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{A}} = \underbrace{\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}}_{M\times M} \underbrace{\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{A}}}_{M\times(M\times N)} \in \mathbb{R}^{M\times(M\times N)}$$

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}} = \operatorname{diag}(1 - \tanh^2(\boldsymbol{z})) \in \mathbb{R}^{M\times M} \tag{6}$$

$$\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{A}}$$

▶▶ See (4)

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{A}}[i,j,k] = \sum_{l=1}^{M} \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}[i,l] \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{A}}[l,j,k]$$

```
dfdA = np.einsum('il, ljk', dfdz, dzdA)
```

Putting Things Together

- Inputs $\boldsymbol{x}$, observed outputs $\boldsymbol{y} = f(\boldsymbol{z}, \boldsymbol{\theta}) + \boldsymbol{\epsilon}$, $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma})$

Putting Things Together

- Inputs $\boldsymbol{x}$, observed outputs $\boldsymbol{y} = f(\boldsymbol{z}, \boldsymbol{\theta}) + \boldsymbol{\epsilon}\,, \boldsymbol{\epsilon} \sim \mathcal{N}\big(\mathbf{0}, \boldsymbol{\Sigma}\big)$
- Train single-layer neural network with

$$\boldsymbol{f}(\boldsymbol{z}, \boldsymbol{\theta}) = \tanh(\boldsymbol{z})\,, \quad \boldsymbol{z} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}\,, \quad \boldsymbol{\theta} = \{\boldsymbol{A}, \boldsymbol{b}\}$$

Putting Things Together

- Inputs $\boldsymbol{x}$, observed outputs $\boldsymbol{y} = f(\boldsymbol{z}, \boldsymbol{\theta}) + \boldsymbol{\epsilon}\,, \boldsymbol{\epsilon} \sim \mathcal{N}\big(\boldsymbol{0}, \boldsymbol{\Sigma}\big)$
- Train single-layer neural network with

$$\boldsymbol{f}(\boldsymbol{z}, \boldsymbol{\theta}) = \tanh(\boldsymbol{z})\,, \quad \boldsymbol{z} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}\,, \quad \boldsymbol{\theta} = \{\boldsymbol{A}, \boldsymbol{b}\}$$

- Find $\boldsymbol{A}, \boldsymbol{b}$, such that the squared loss

$$L(\boldsymbol{\theta}) = \tfrac{1}{2}\|\boldsymbol{e}\|^2\,, \quad \boldsymbol{e} = \boldsymbol{y} - \boldsymbol{f}(\boldsymbol{z}, \boldsymbol{\theta})$$

  is minimized

Putting Things Together

- Inputs $\boldsymbol{x}$, observed outputs $\boldsymbol{y} = f(\boldsymbol{z}, \boldsymbol{\theta}) + \boldsymbol{\epsilon}\,, \boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{\Sigma})$
- Train single-layer neural network with

$$\boldsymbol{f}(\boldsymbol{z}, \boldsymbol{\theta}) = \tanh(\boldsymbol{z})\,, \quad \boldsymbol{z} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}\,, \quad \boldsymbol{\theta} = \{\boldsymbol{A}, \boldsymbol{b}\}$$

- Find $\boldsymbol{A}, \boldsymbol{b}$, such that the squared loss

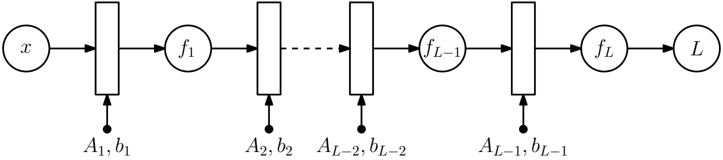
$$L(\boldsymbol{\theta}) = \tfrac{1}{2}\|\boldsymbol{e}\|^2\,, \quad \boldsymbol{e} = \boldsymbol{y} - \boldsymbol{f}(\boldsymbol{z}, \boldsymbol{\theta})$$

  is minimized

- Partial derivatives:

$$\left.\begin{aligned} \frac{\partial L}{\partial \boldsymbol{A}} &= \frac{\partial L}{\partial \boldsymbol{e}} \frac{\partial \boldsymbol{e}}{\partial \boldsymbol{f}} \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{A}} \\ \frac{\partial L}{\partial \boldsymbol{b}} &= \frac{\partial L}{\partial \boldsymbol{e}} \frac{\partial \boldsymbol{e}}{\partial \boldsymbol{f}} \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}} \end{aligned}\right\} \quad \frac{\partial L}{\partial \boldsymbol{e}} \blacktriangleright\blacktriangleright (1) \quad \frac{\partial \boldsymbol{e}}{\partial \boldsymbol{f}} \blacktriangleright\blacktriangleright (2), (3) \quad \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}} \blacktriangleright\blacktriangleright (6) \quad \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{A}} \blacktriangleright\blacktriangleright (4) \quad \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}} \blacktriangleright\blacktriangleright (5)$$

Gradients of a Multi-Layer Neural Network

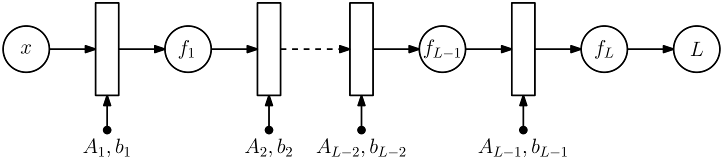


- Inputs $\boldsymbol{x}$, observed outputs $\boldsymbol{y}$
- Train multi-layer neural network with

$$\boldsymbol{f}_0 = \boldsymbol{x}$$

$$\boldsymbol{f}_i = \sigma_i(\boldsymbol{A}_{i-1}\boldsymbol{f}_{i-1} + \boldsymbol{b}_{i-1}), \quad i = 1, \dots, L$$

Gradients of a Multi-Layer Neural Network



- Inputs $\boldsymbol{x}$, observed outputs $\boldsymbol{y}$
- Train multi-layer neural network with

$$\boldsymbol{f}_0 = \boldsymbol{x}$$

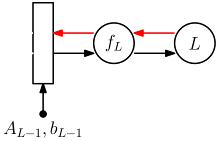
$$\boldsymbol{f}_i = \sigma_i(\boldsymbol{A}_{i-1}\boldsymbol{f}_{i-1} + \boldsymbol{b}_{i-1}), \quad i = 1, \ldots, L$$

- Find $\boldsymbol{A}_j, \boldsymbol{b}_j$ for $j = 0, \ldots, L-1$, such that the squared loss

$$L(\boldsymbol{\theta}) = \|\boldsymbol{y} - \boldsymbol{f}_L(\boldsymbol{\theta}, \boldsymbol{x})\|^2$$

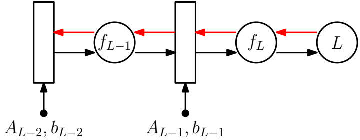
is minimized, where $\boldsymbol{\theta} = \{\boldsymbol{A}_j, \boldsymbol{b}_j\}, \quad j = 0, \ldots, L-1$

Gradients of a Multi-Layer Neural Network



$$\frac{\partial L}{\partial \boldsymbol{\theta}_{L-1}} = \frac{\partial L}{\partial \boldsymbol{f}_L} \frac{\partial \boldsymbol{f}_L}{\partial \boldsymbol{\theta}_{L-1}}$$

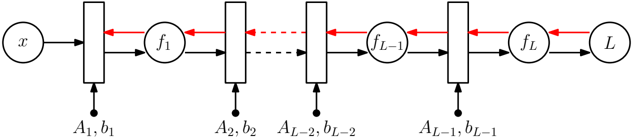
Gradients of a Multi-Layer Neural Network



$$\frac{\partial L}{\partial \boldsymbol{\theta}_{L-1}} = \frac{\partial L}{\partial \boldsymbol{f}_L} \frac{\partial \boldsymbol{f}_L}{\partial \boldsymbol{\theta}_{L-1}}$$

$$\frac{\partial L}{\partial \boldsymbol{\theta}_{L-2}} = \frac{\partial L}{\partial \boldsymbol{f}_L} \boxed{\frac{\partial \boldsymbol{f}_L}{\partial \boldsymbol{f}_{L-1}} \frac{\partial \boldsymbol{f}_{L-1}}{\partial \boldsymbol{\theta}_{L-2}}}$$

Gradients of a Multi-Layer Neural Network

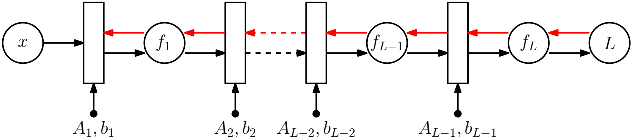


$$\frac{\partial L}{\partial \boldsymbol{\theta}_{L-1}} = \frac{\partial L}{\partial \boldsymbol{f}_L} \frac{\partial \boldsymbol{f}_L}{\partial \boldsymbol{\theta}_{L-1}}$$

$$\frac{\partial L}{\partial \boldsymbol{\theta}_{L-2}} = \frac{\partial L}{\partial \boldsymbol{f}_L} \boxed{\frac{\partial \boldsymbol{f}_L}{\partial \boldsymbol{f}_{L-1}} \frac{\partial \boldsymbol{f}_{L-1}}{\partial \boldsymbol{\theta}_{L-2}}}$$

$$\frac{\partial L}{\partial \boldsymbol{\theta}_{L-3}} = \frac{\partial L}{\partial \boldsymbol{f}_L} \frac{\partial \boldsymbol{f}_L}{\partial \boldsymbol{f}_{L-1}} \boxed{\frac{\partial \boldsymbol{f}_{L-1}}{\partial \boldsymbol{f}_{L-2}} \frac{\partial \boldsymbol{f}_{L-2}}{\partial \boldsymbol{\theta}_{L-3}}}$$

Gradients of a Multi-Layer Neural Network



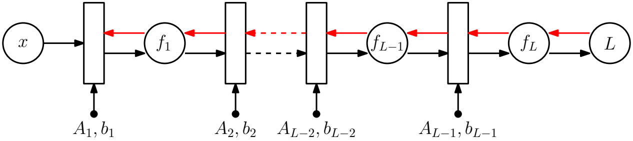
$$\frac{\partial L}{\partial \boldsymbol{\theta}_{L-1}} = \frac{\partial L}{\partial \boldsymbol{f}_L} \frac{\partial \boldsymbol{f}_L}{\partial \boldsymbol{\theta}_{L-1}}$$

$$\frac{\partial L}{\partial \boldsymbol{\theta}_{L-2}} = \frac{\partial L}{\partial \boldsymbol{f}_L} \boxed{\frac{\partial \boldsymbol{f}_L}{\partial \boldsymbol{f}_{L-1}} \frac{\partial \boldsymbol{f}_{L-1}}{\partial \boldsymbol{\theta}_{L-2}}}$$

$$\frac{\partial L}{\partial \boldsymbol{\theta}_{L-3}} = \frac{\partial L}{\partial \boldsymbol{f}_L} \frac{\partial \boldsymbol{f}_L}{\partial \boldsymbol{f}_{L-1}} \boxed{\frac{\partial \boldsymbol{f}_{L-1}}{\partial \boldsymbol{f}_{L-2}} \frac{\partial \boldsymbol{f}_{L-2}}{\partial \boldsymbol{\theta}_{L-3}}}$$

$$\frac{\partial L}{\partial \boldsymbol{\theta}_i} = \frac{\partial L}{\partial \boldsymbol{f}_L} \frac{\partial \boldsymbol{f}_L}{\partial \boldsymbol{f}_{L-1}} \cdots \boxed{\frac{\partial \boldsymbol{f}_{i+2}}{\partial \boldsymbol{f}_{i+1}} \frac{\partial \boldsymbol{f}_{i+1}}{\partial \boldsymbol{\theta}_i}}$$

Gradients of a Multi-Layer Neural Network



$$\frac{\partial L}{\partial \boldsymbol{\theta}_{L-1}} = \frac{\partial L}{\partial \boldsymbol{f}_L} \frac{\partial \boldsymbol{f}_L}{\partial \boldsymbol{\theta}_{L-1}}$$

$$\frac{\partial L}{\partial \boldsymbol{\theta}_{L-2}} = \frac{\partial L}{\partial \boldsymbol{f}_L} \boxed{\frac{\partial \boldsymbol{f}_L}{\partial \boldsymbol{f}_{L-1}} \frac{\partial \boldsymbol{f}_{L-1}}{\partial \boldsymbol{\theta}_{L-2}}}$$

$$\frac{\partial L}{\partial \boldsymbol{\theta}_{L-3}} = \frac{\partial L}{\partial \boldsymbol{f}_L} \frac{\partial \boldsymbol{f}_L}{\partial \boldsymbol{f}_{L-1}} \boxed{\frac{\partial \boldsymbol{f}_{L-1}}{\partial \boldsymbol{f}_{L-2}} \frac{\partial \boldsymbol{f}_{L-2}}{\partial \boldsymbol{\theta}_{L-3}}}$$

$$\frac{\partial L}{\partial \boldsymbol{\theta}_{i}} = \frac{\partial L}{\partial \boldsymbol{f}_L} \frac{\partial \boldsymbol{f}_L}{\partial \boldsymbol{f}_{L-1}} \cdots \boxed{\frac{\partial \boldsymbol{f}_{i+2}}{\partial \boldsymbol{f}_{i+1}} \frac{\partial \boldsymbol{f}_{i+1}}{\partial \boldsymbol{\theta}_{i}}}$$

▸▸ More details (including efficient implementation) later this week

Training Neural Networks as Maximum Likelihood Estimation

- Training a neural network in the above way corresponds to maximum likelihood estimation:
  - If $\boldsymbol{y} = NN(\boldsymbol{x}, \boldsymbol{\theta}) + \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$ then the log-likelihood is

$$\log p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta}) = -\tfrac{1}{2}\|\boldsymbol{y} - NN(\boldsymbol{x}, \boldsymbol{\theta})\|^2$$

Training Neural Networks as Maximum Likelihood Estimation

- Training a neural network in the above way corresponds to maximum likelihood estimation:
  - If $\boldsymbol{y} = NN(\boldsymbol{x}, \boldsymbol{\theta}) + \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}\big(\boldsymbol{0}, \boldsymbol{I}\big)$ then the log-likelihood is

$$\log p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta}) = -\tfrac{1}{2}\|\boldsymbol{y} - NN(\boldsymbol{x}, \boldsymbol{\theta})\|^2$$

  - Find $\boldsymbol{\theta}^*$ by minimizing the negative log-likelihood:

$$\begin{aligned}\boldsymbol{\theta}^* &= \arg\min_{\boldsymbol{\theta}} -\log p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta}) \\ &= \arg\min_{\boldsymbol{\theta}} \tfrac{1}{2}\|\boldsymbol{y} - NN(\boldsymbol{x}, \boldsymbol{\theta})\|^2 \\ &= \arg\min_{\boldsymbol{\theta}} L(\boldsymbol{\theta})\end{aligned}$$

Training Neural Networks as Maximum Likelihood Estimation

- Training a neural network in the above way corresponds to maximum likelihood estimation:
  - If $\boldsymbol{y} = NN(\boldsymbol{x}, \boldsymbol{\theta}) + \boldsymbol{\epsilon}, \quad \boldsymbol{\epsilon} \sim \mathcal{N}\big(\boldsymbol{0}, \boldsymbol{I}\big)$ then the log-likelihood is

$$\log p(\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\theta}) = -\tfrac{1}{2}\|\boldsymbol{y} - NN(\boldsymbol{x}, \boldsymbol{\theta})\|^2$$

  - Find $\boldsymbol{\theta}^*$ by minimizing the negative log-likelihood:

$$\begin{aligned}\boldsymbol{\theta}^* &= \arg\min_{\boldsymbol{\theta}} -\log p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta}) \\ &= \arg\min_{\boldsymbol{\theta}} \tfrac{1}{2}\|\boldsymbol{y} - NN(\boldsymbol{x}, \boldsymbol{\theta})\|^2 \\ &= \arg\min_{\boldsymbol{\theta}} L(\boldsymbol{\theta})\end{aligned}$$

- Maximum likelihood estimation can lead to overfitting (interpret noise as signal)

Example: Linear Regression (1)

- Linear regression with a polynomial of order M:

$$y = f(x, \boldsymbol{\theta}) + \epsilon\,, \quad \epsilon \sim \mathcal{N}\big(0,\, \sigma_\epsilon^2\big)$$

$$f(x, \boldsymbol{\theta}) = \theta_0 + \theta_1 x + \theta_2 x^2 + \cdots + \theta_M x^M = \sum_{i=0}^{M} \theta_i x^i$$

Example: Linear Regression (1)

- Linear regression with a polynomial of order M:

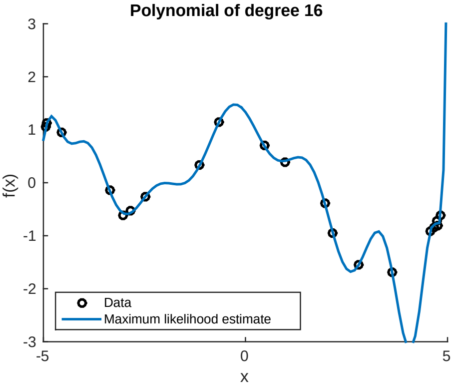
$$y = f(x, \boldsymbol{\theta}) + \epsilon\,, \quad \epsilon \sim \mathcal{N}\big(0,\, \sigma_\epsilon^2\big)$$

$$f(x, \boldsymbol{\theta}) = \theta_0 + \theta_1 x + \theta_2 x^2 + \cdots + \theta_M x^M = \sum_{i=0}^{M} \theta_i x^i$$

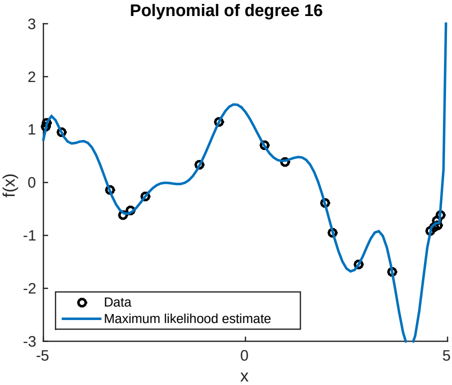
- Given inputs x_i and corresponding (noisy) observations y_i, $i = 1, \ldots, N$, find parameters $\boldsymbol{\theta} = [\theta_0, \ldots, \theta_M]^\top$, that minimize the squared loss (equivalently: maximize the likelihood)

$$L(\boldsymbol{\theta}) = \sum_{i=1}^{N} (y_i - f(x_i, \boldsymbol{\theta}))^2$$

Example: Linear Regression (2)



Example: Linear Regression (2)



- Regularization, model selection etc. can address overfitting
  ▶▶ Tutorials later this week
- Alternative approach based on integration

Overview

Introduction

Differentiation

Integration

Integration: Outline

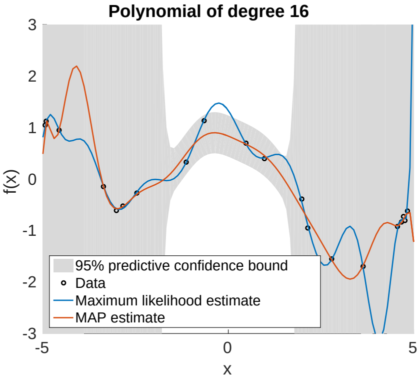
1. Motivation
2. Monte-Carlo estimation
3. Basic sampling algorithms

Bayesian Integration to Avoid Overfitting

- Instead of fitting a single set of parameters $\boldsymbol{\theta}^*$, we can average over all plausible parameters

  ▶▶ Bayesian integration:

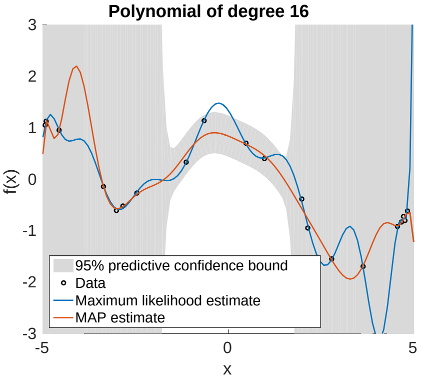
$$p(\boldsymbol{y}|\boldsymbol{x}) = \int p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta}$$



Bayesian Integration to Avoid Overfitting

- Instead of fitting a single set of parameters $\boldsymbol{\theta}^*$, we can average over all plausible parameters
  ⏩ Bayesian integration:

$$p(\boldsymbol{y}|\boldsymbol{x}) = \int p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta})p(\boldsymbol{\theta})d\boldsymbol{\theta}$$

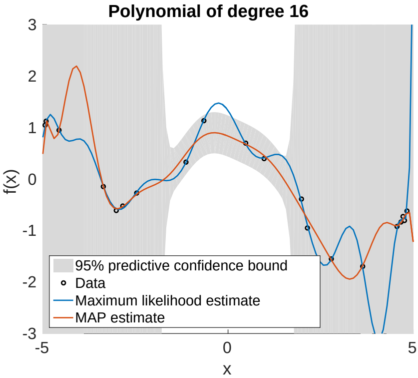


- More details on what $p(\boldsymbol{\theta})$ is ⏩ Tutorials later this week

Bayesian Integration to Avoid Overfitting

- Instead of fitting a single set of parameters $\boldsymbol{\theta}^*$, we can average over all plausible parameters
  ⏩ Bayesian integration:

$$p(\boldsymbol{y}|\boldsymbol{x}) = \int p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{\theta}) p(\boldsymbol{\theta}) d\boldsymbol{\theta}$$



- More details on what $p(\boldsymbol{\theta})$ is ⏩ Tutorials later this week
- For neural networks this integration is intractable
  ⏩ Approximations

Computing Statistics of Random Variables

- Computing means/(co)variances also requires solving integrals:

$$\mathbb{E}_x[\boldsymbol{x}] = \int \boldsymbol{x} p(\boldsymbol{x}) d\boldsymbol{x} =: \boldsymbol{\mu}_x$$

$$\mathbb{V}_x[\boldsymbol{x}] = \int (\boldsymbol{x} - \boldsymbol{\mu}_x)(\boldsymbol{x} - \boldsymbol{\mu}_x)^\top d\boldsymbol{x}$$

$$\text{Cov}[\boldsymbol{x}, \boldsymbol{y}] = \iint (\boldsymbol{x} - \boldsymbol{\mu}_x)(\boldsymbol{y} - \boldsymbol{\mu}_y)^\top d\boldsymbol{x} d\boldsymbol{y}$$

Computing Statistics of Random Variables

- Computing means/(co)variances also requires solving integrals:

$$\mathbb{E}_x[\boldsymbol{x}] = \int \boldsymbol{x} p(\boldsymbol{x}) d\boldsymbol{x} =: \boldsymbol{\mu}_x$$

$$\mathbb{V}_x[\boldsymbol{x}] = \int (\boldsymbol{x} - \boldsymbol{\mu}_x)(\boldsymbol{x} - \boldsymbol{\mu}_x)^\top d\boldsymbol{x}$$

$$\mathrm{Cov}[\boldsymbol{x}, \boldsymbol{y}] = \iint (\boldsymbol{x} - \boldsymbol{\mu}_x)(\boldsymbol{y} - \boldsymbol{\mu}_y)^\top d\boldsymbol{x} d\boldsymbol{y}$$

- These integrals can often not be computed in closed form

▸▸ Approximations

Approximate Integration

- Numerical integration (low-dimensional problems)
- Bayesian quadrature, e.g., O'Hagan (1987, 1991); Rasmussen & Ghahramani (2003)
- Variational Bayes, e.g., Jordan et al. (1999)
- Expectation Propagation, Opper & Winther (2001); Minka (2001)
- **Monte-Carlo Methods**, e.g., Gilks et al. (1996), Robert & Casella (2004), Bishop (2006)

Monte Carlo Methods—Motivation

- Monte Carlo methods are computational techniques that make use of random numbers
- Two typical problems:
  1. **Problem 1:** Generate samples $\{\boldsymbol{x}^{(s)}\}$ from a given probability distribution $p(\boldsymbol{x})$, e.g., for simulation or representations of data distributions

Monte Carlo Methods—Motivation

- Monte Carlo methods are computational techniques that make use of random numbers
- Two typical problems:
  1. **Problem 1:** Generate samples $\{\boldsymbol{x}^{(s)}\}$ from a given probability distribution $p(\boldsymbol{x})$, e.g., for simulation or representations of data distributions
  2. **Problem 2:** Compute expectations of functions under that distribution:

$$\mathbb{E}[f(\boldsymbol{x})] = \int f(\boldsymbol{x})p(\boldsymbol{x})d\boldsymbol{x}$$

Monte Carlo Methods—Motivation

- Monte Carlo methods are computational techniques that make use of random numbers
- Two typical problems:
  1. **Problem 1:** Generate samples $\{\boldsymbol{x}^{(s)}\}$ from a given probability distribution $p(\boldsymbol{x})$, e.g., for simulation or representations of data distributions
  2. **Problem 2:** Compute expectations of functions under that distribution:

$$\mathbb{E}[f(\boldsymbol{x})] = \int f(\boldsymbol{x})p(\boldsymbol{x})d\boldsymbol{x}$$

▶▶ Example: Means/variances of distributions, predictions

Complication: Integral cannot be evaluated analytically

Problem 2: Monte Carlo Estimation

- Computing expectations via statistical sampling:

$$\mathbb{E}[f(\boldsymbol{x})] = \int f(\boldsymbol{x})p(\boldsymbol{x})d\boldsymbol{x}$$
$$\approx \frac{1}{S}\sum_{s=1}^{S} f(\boldsymbol{x}^{(s)}), \quad \boldsymbol{x}^{(s)} \sim p(\boldsymbol{x})$$

Problem 2: Monte Carlo Estimation

- Computing expectations via statistical sampling:

$$\mathbb{E}[f(\boldsymbol{x})] = \int f(\boldsymbol{x})p(\boldsymbol{x})d\boldsymbol{x}$$
$$\approx \frac{1}{S}\sum\nolimits_{s=1}^{S} f(\boldsymbol{x}^{(s)}), \quad \boldsymbol{x}^{(s)} \sim p(\boldsymbol{x})$$

- Making predictions (e.g., Bayesian regression with inputs $\boldsymbol{x}$ and targets $\boldsymbol{y}$)

$$p(\boldsymbol{y}|\boldsymbol{x}) = \int p(\boldsymbol{y}|\boldsymbol{\theta}, \boldsymbol{x}) \underbrace{p(\boldsymbol{\theta})}_{\text{Parameter distribution}} d\boldsymbol{\theta}$$
$$\approx \frac{1}{S}\sum\nolimits_{s=1}^{S} p(\boldsymbol{y}|\boldsymbol{\theta}^{(s)}, \boldsymbol{x}), \quad \boldsymbol{\theta}^{(s)} \sim p(\boldsymbol{\theta})$$

Problem 2: Monte Carlo Estimation

- Computing expectations via statistical sampling:

$$\mathbb{E}[f(\boldsymbol{x})] = \int f(\boldsymbol{x})p(\boldsymbol{x})d\boldsymbol{x}$$
$$\approx \frac{1}{S}\sum_{s=1}^{S} f(\boldsymbol{x}^{(s)}), \quad \boldsymbol{x}^{(s)} \sim p(\boldsymbol{x})$$

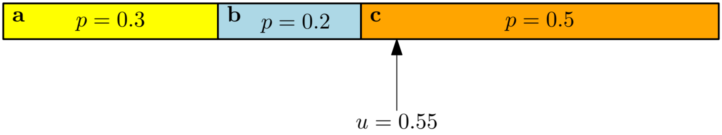
- Making predictions (e.g., Bayesian regression with inputs $\boldsymbol{x}$ and targets $\boldsymbol{y}$)

$$p(\boldsymbol{y}|\boldsymbol{x}) = \int p(\boldsymbol{y}|\boldsymbol{\theta}, \boldsymbol{x}) \underbrace{p(\boldsymbol{\theta})}_{\text{Parameter distribution}} d\boldsymbol{\theta}$$
$$\approx \frac{1}{S}\sum_{s=1}^{S} p(\boldsymbol{y}|\boldsymbol{\theta}^{(s)}, \boldsymbol{x}), \quad \boldsymbol{\theta}^{(s)} \sim p(\boldsymbol{\theta})$$

- **Key problem:** Generating samples from $p(\boldsymbol{x})$ or $p(\boldsymbol{\theta})$

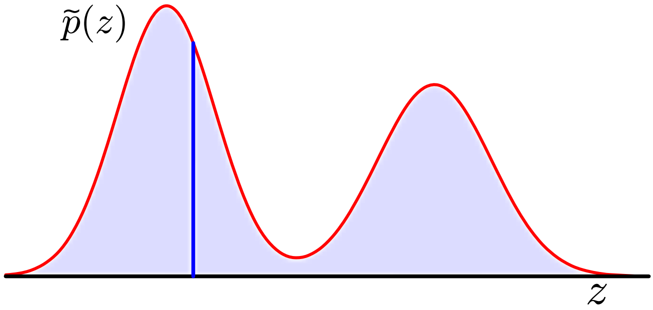
  ▶▶ Need to solve **Problem 1**

Sampling Discrete Values



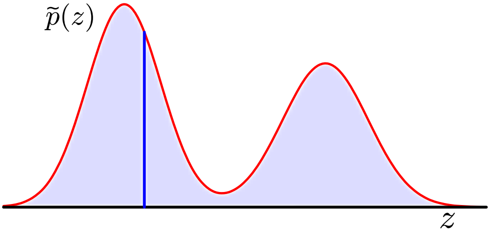
- $u \sim \mathcal{U}[0, 1]$, where $\mathcal{U}$ is the uniform distribution
- $u = 0.55 \Rightarrow x = c$

Continuous Variables



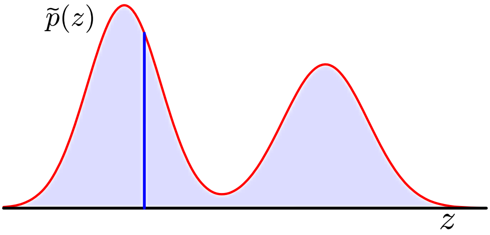
- More complicated
- Geometrically, we wish to sample uniformly from the area under the curve
- Two algorithms here:
  - Rejection sampling
  - Importance sampling

Rejection Sampling: Setting



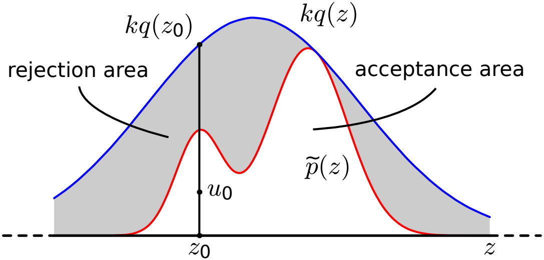
- Assume:
  - Sampling from $p(z)$ is difficult
  - Evaluating $\tilde{p}(z) = Zp(z)$ is easy (and Z may be unknown)

Rejection Sampling: Setting



- Assume:
  - Sampling from $p(z)$ is difficult
  - Evaluating $\tilde{p}(z) = Zp(z)$ is easy (and Z may be unknown)
- Find a simpler distribution (proposal distribution) $q(z)$ from which we can easily draw samples (e.g., Gaussian, Laplace)
- Find an upper bound $kq(z) \geqslant \tilde{p}(z)$

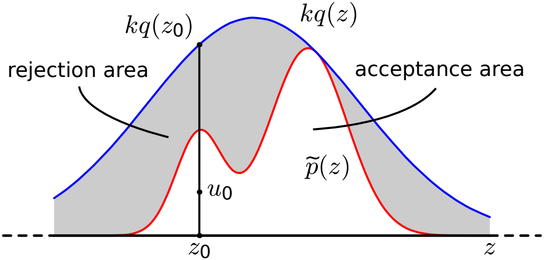
Rejection Sampling: Algorithm



Adapted from PRML (Bishop, 2006)

1. Generate $z_0 \sim q(z)$
2. Generate $u_0 \sim \mathcal{U}[0, kq(z_0)]$
3. If $u_0 > \tilde{p}(z_0)$, reject the sample. Otherwise, retain z_0

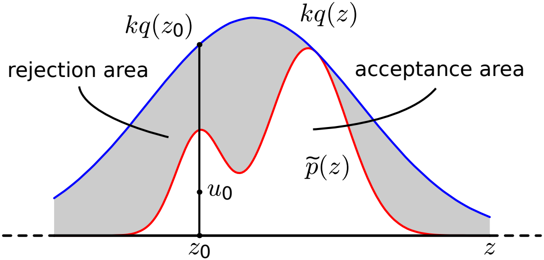
Properties



Adapted from PRML (Bishop, 2006)

- Accepted pairs (z, u) are uniformly distributed under $\tilde{p}(z)$

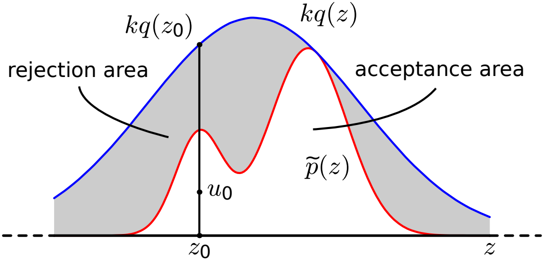
Properties



Adapted from PRML (Bishop, 2006)

- Accepted pairs (z, u) are uniformly distributed under $\tilde{p}(z)$
- Probability density of the z-coordiantes of accepted points must be proportional to $p(z)$

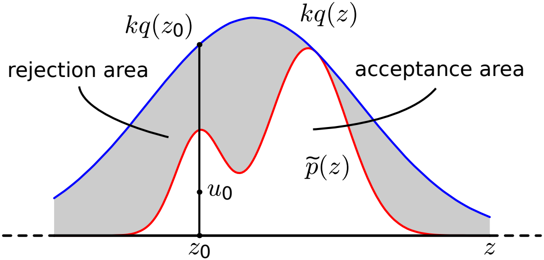
Properties



Adapted from PRML (Bishop, 2006)

- Accepted pairs (z, u) are uniformly distributed under $\tilde{p}(z)$
- Probability density of the z-coordiantes of accepted points must be proportional to $p(z)$
- Samples are independent samples from $p(z)$

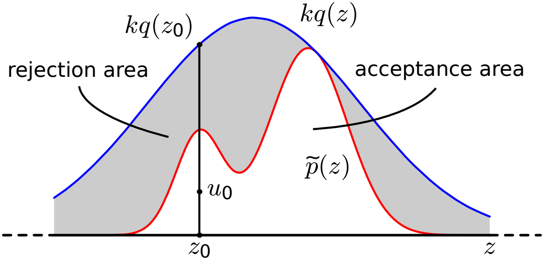
Shortcomings



Adapted from PRML (Bishop, 2006)

- Finding the upper bound k is tricky

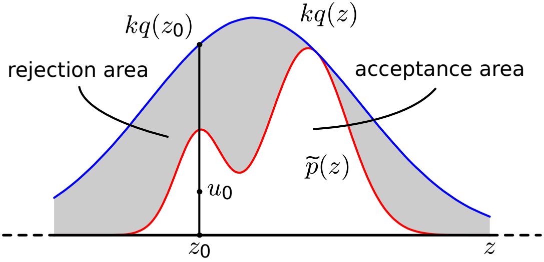
Shortcomings



Adapted from PRML (Bishop, 2006)

- Finding the upper bound k is tricky
- In high dimensions the factor k is probably huge

Shortcomings



Adapted from PRML (Bishop, 2006)

- Finding the upper bound k is tricky
- In high dimensions the factor k is probably huge
- **Low acceptance rate/high rejection rate** of samples

Importance Sampling

Key idea: Do not throw away all rejected samples, but give them lower weight by rewriting the integral as an expectation under a simpler distribution q (proposal distribution):

$$\mathbb{E}_p[f(\boldsymbol{x})] = \int f(\boldsymbol{x})p(\boldsymbol{x})d\boldsymbol{x}$$

Importance Sampling

Key idea: Do not throw away all rejected samples, but give them lower weight by rewriting the integral as an expectation under a simpler distribution q (proposal distribution):

$$\mathbb{E}_p[f(\boldsymbol{x})] = \int f(\boldsymbol{x})p(\boldsymbol{x})d\boldsymbol{x}$$
$$= \int f(\boldsymbol{x})p(\boldsymbol{x})\frac{q(\boldsymbol{x})}{q(\boldsymbol{x})}d\boldsymbol{x}$$

Importance Sampling

Key idea: Do not throw away all rejected samples, but give them lower weight by rewriting the integral as an expectation under a simpler distribution q (proposal distribution):

$$\mathbb{E}_p[f(\boldsymbol{x})] = \int f(\boldsymbol{x})p(\boldsymbol{x})d\boldsymbol{x}$$

$$= \int f(\boldsymbol{x})p(\boldsymbol{x})\frac{q(\boldsymbol{x})}{q(\boldsymbol{x})}d\boldsymbol{x} = \int f(x)\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}q(\boldsymbol{x})d\boldsymbol{x}$$

Importance Sampling

Key idea: Do not throw away all rejected samples, but give them lower weight by rewriting the integral as an expectation under a simpler distribution q (proposal distribution):

$$\begin{aligned}\mathbb{E}_p[f(\boldsymbol{x})] &= \int f(\boldsymbol{x})p(\boldsymbol{x})d\boldsymbol{x} \\ &= \int f(\boldsymbol{x})p(\boldsymbol{x})\frac{q(\boldsymbol{x})}{q(\boldsymbol{x})}d\boldsymbol{x} = \int f(x)\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}q(\boldsymbol{x})d\boldsymbol{x} \\ &= \mathbb{E}_q\left[f(\boldsymbol{x})\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\right]\end{aligned}$$

Importance Sampling

Key idea: Do not throw away all rejected samples, but give them lower weight by rewriting the integral as an expectation under a simpler distribution q (proposal distribution):

$$\begin{aligned}
\mathbb{E}_p[f(\boldsymbol{x})] &= \int f(\boldsymbol{x})p(\boldsymbol{x})d\boldsymbol{x} \\
&= \int f(\boldsymbol{x})p(\boldsymbol{x})\frac{q(\boldsymbol{x})}{q(\boldsymbol{x})}d\boldsymbol{x} = \int f(x)\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}q(\boldsymbol{x})d\boldsymbol{x} \\
&= \mathbb{E}_q\left[f(\boldsymbol{x})\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\right]
\end{aligned}$$

If we choose q in a way that we can easily sample from it, we can approximate this last expectation by Monte Carlo:

$$E_q\left[f(\boldsymbol{x})\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\right] \approx \frac{1}{S}\sum_{s=1}^{S} f(\boldsymbol{x}^{(s)})\frac{p(\boldsymbol{x}^{(s)})}{q(\boldsymbol{x}^{(s)})}\qquad, \quad \boldsymbol{x}^{(s)} \sim q(\boldsymbol{x})$$

Importance Sampling

Key idea: Do not throw away all rejected samples, but give them lower weight by rewriting the integral as an expectation under a simpler distribution q (proposal distribution):

$$\begin{aligned}\mathbb{E}_p[f(\boldsymbol{x})] &= \int f(\boldsymbol{x})p(\boldsymbol{x})d\boldsymbol{x} \\ &= \int f(\boldsymbol{x})p(\boldsymbol{x})\frac{q(\boldsymbol{x})}{q(\boldsymbol{x})}d\boldsymbol{x} = \int f(x)\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}q(\boldsymbol{x})d\boldsymbol{x} \\ &= \mathbb{E}_q\left[f(\boldsymbol{x})\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\right]\end{aligned}$$

If we choose q in a way that we can easily sample from it, we can approximate this last expectation by Monte Carlo:

$$E_q\left[f(\boldsymbol{x})\frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\right] \approx \frac{1}{S}\sum_{s=1}^{S} f(\boldsymbol{x}^{(s)})\frac{p(\boldsymbol{x}^{(s)})}{q(\boldsymbol{x}^{(s)})} = \frac{1}{S}\sum_{s=1}^{S} w_s f(\boldsymbol{x}^{(s)}), \quad \boldsymbol{x}^{(s)} \sim q(\boldsymbol{x})$$

Properties

- Unbiased if $q > 0$ where $p > 0$ and if we can evaluate p

Properties

- Unbiased if $q > 0$ where $p > 0$ and if we can evaluate p
- Breaks down if we do not have enough samples (puts nearly all weight on a single sample)

  ▶▶ Degeneracy, see also Particle Filtering and SMC (Thrun et al., 2005; Doucet et al., 2000)

Properties

- Unbiased if $q > 0$ where $p > 0$ and if we can evaluate p
- Breaks down if we do not have enough samples (puts nearly all weight on a single sample)

  ▸▸ Degeneracy, see also Particle Filtering and SMC (Thrun et al., 2005; Doucet et al., 2000)
- Many draws from proposal density q required, especially in high dimensions

Properties

- Unbiased if $q > 0$ where $p > 0$ and if we can evaluate p
- Breaks down if we do not have enough samples (puts nearly all weight on a single sample)
  ▸▸ Degeneracy, see also Particle Filtering and SMC (Thrun et al., 2005; Doucet et al., 2000)
- Many draws from proposal density q required, especially in high dimensions
- Requires to be able to evaluate true p. Generalization exists for $\tilde{p}$. This generalization is biased (but consistent).

Properties

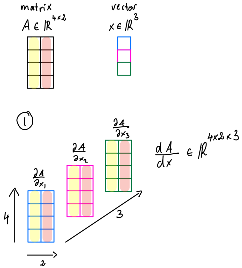
- Unbiased if $q > 0$ where $p > 0$ and if we can evaluate p
- Breaks down if we do not have enough samples (puts nearly all weight on a single sample)
  ⏩ Degeneracy, see also Particle Filtering and SMC (Thrun et al., 2005; Doucet et al., 2000)
- Many draws from proposal density q required, especially in high dimensions
- Requires to be able to evaluate true p. Generalization exists for $\tilde{p}$. This generalization is biased (but consistent).
- Does not scale to interesting (high-dimensional) problems

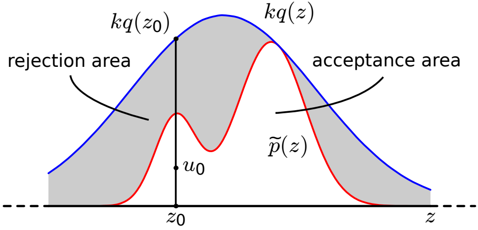
Properties

- Unbiased if $q > 0$ where $p > 0$ and if we can evaluate p
- Breaks down if we do not have enough samples (puts nearly all weight on a single sample)
  ▸▸ Degeneracy, see also Particle Filtering and SMC (Thrun et al., 2005; Doucet et al., 2000)
- Many draws from proposal density q required, especially in high dimensions
- Requires to be able to evaluate true p. Generalization exists for $\tilde{p}$. This generalization is biased (but consistent).
- Does not scale to interesting (high-dimensional) problems

▸▸ Different approach to sample from complicated (high-dimensional) distributions: Markov Chain Monte Carlo (e.g., Gilks et al., 1996)

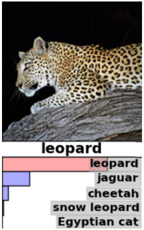
Summary

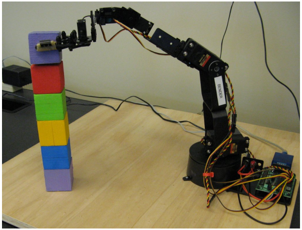


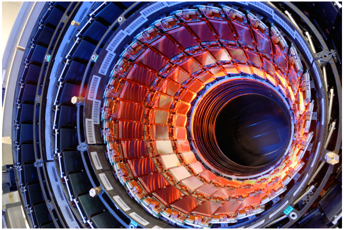


- Two mathematical challenges in machine learning
  - **Differentiation** for optimizing parameters of machine learning models
    ⏩ Vector calculus and chain rule
  - **Integration** for computing statistics (e.g., means, variances) and as a principled way to address overfitting issue
    ⏩ Monte-Carlo integration to solve intractable integrals

Some Application Areas







- Image/speech/text/language processing using deep neural networks (e.g., Krizhevsky et al., 2012 or overview in Goodfellow et al., 2016)
- Data-efficient reinforcement learning and robot learning using Gaussian processes (e.g., Deisenroth & Rasmussen, 2011)
- High-energy physics using deep neural networks or Gaussian processes (e.g., Sadowski et al. 2014; Bertone et al., 2016)

References I

[1] G. Bertone, M. P. Deisenroth, J. S. Kim, S. Liem, R. R. de Austri, and M. Welling. Accelerating the BSM Interpretation of LHC Data with Machine Learning. arXiv preprint arXiv:1611.02704, 2016.

[2] C. M. Bishop. *Pattern Recognition and Machine Learning*. Information Science and Statistics. Springer-Verlag, 2006.

[3] M. P. Deisenroth, D. Fox, and C. E. Rasmussen. Gaussian Processes for Data-Efficient Learning in Robotics and Control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(2):408–423, Feb. 2015.

[4] M. P. Deisenroth and C. E. Rasmussen. PILCO: A Model-Based and Data-Efficient Approach to Policy Search. In *Proceedings of the International Conference on Machine Learning*, pages 465–472. ACM, June 2011.

[5] A. Doucet, S. J. Godsill, and C. Andrieu. On Sequential Monte Carlo Sampling Methods for Bayesian Filtering. *Statistics and Computing*, 10:197–208, 2000.

[6] W. R. Gilks, S. Richardson, and D. J. Spiegelhalter, editors. *Markov Chain Monte Carlo in Practice: Interdisciplinary Statistics*. Chapman & Hall, 1996.

[7] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT press, 2016.

[8] M. I. Jordan, Z. Ghahramani, T. S. Jaakkola, and L. K. Saul. An Introduction to Variational Methods for Graphical Models. *Machine Learning*, 37:183–233, 1999.

[9] S. Kamthe and M. P. Deisenroth. Data-Efficient Reinforcement Learning with Probabilistic Model Predictive Control. *arXiv:1706.06491*, abs/1706.06491, 2017.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet Classification with Deep Convolutional Neural Networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[11] T. P. Minka. *A Family of Algorithms for Approximate Bayesian Inference*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, Jan. 2001.

[12] R. M. Neal. *Bayesian Learning for Neural Networks*. PhD thesis, Department of Computer Science, University of Toronto, 1996.

[13] A. O'Hagan. Monte Carlo is Fundamentally Unsound. *The Statistician*, 36(2/3):247–249, 1987.

References II

[14] A. O'Hagan. Bayes-Hermite Quadrature. *Journal of Statistical Planning and Inference*, 29:245–260, 1991.

[15] M. Opper and O. Winther. Adaptive and Self-averaging Thouless-Anderson-Palmer Mean-field Theory for Probabilistic Modeling. *Physical Review E: Statistical, Nonlinear, and Soft Matter Physics*, 64:056131, 2001.

[16] C. E. Rasmussen and Z. Ghahramani. Bayesian Monte Carlo. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 489–496. The MIT Press, Cambridge, MA, USA, 2003.

[17] C. P. Robert and G. Casella. *Monte Carlo Methods*. Wiley Online Library, 2004.

[18] P. Sadowski, J. Collado, D. Whiteson, and P. Baldi. Deep Learning, Dark Knowledge, and Dark Matter. In *NIPS Workshop on High-energy Physics and Machine Learning*, pages 81–87, 2014.

[19] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics*. The MIT Press, Cambridge, MA, USA, 2005.