Imperial College London

Mathematics for Machine Learning

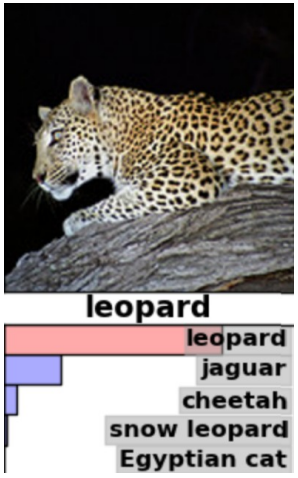
Marc Deisenroth

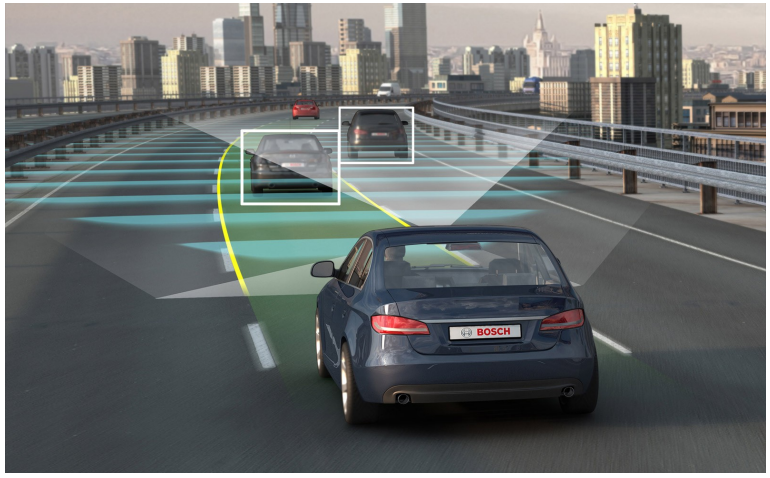
Statistical Machine Learning Group
Department of Computing
Imperial College London

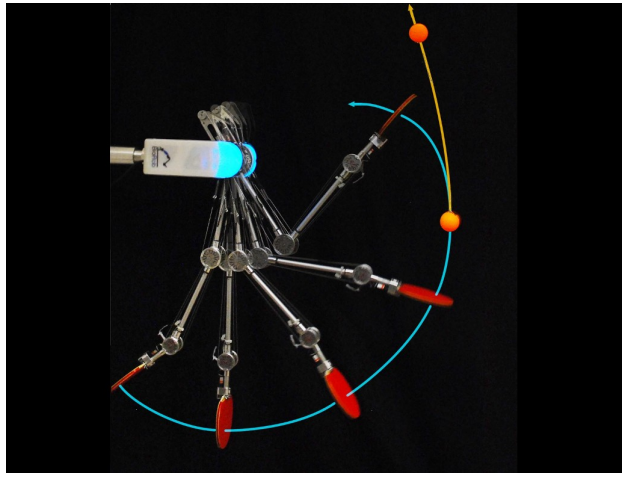
@mpd37
m.deisenroth@imperial.ac.uk
marc@prowler.io

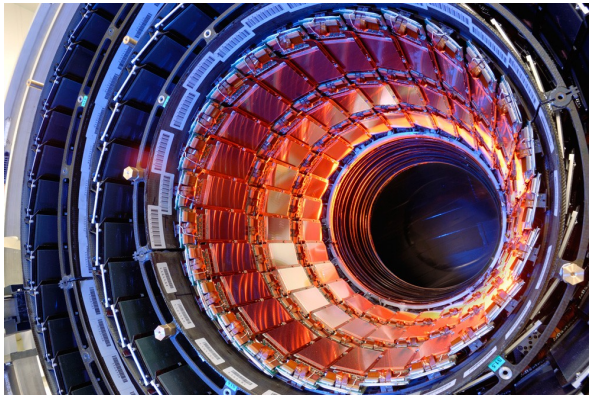
Deep Learning 2017

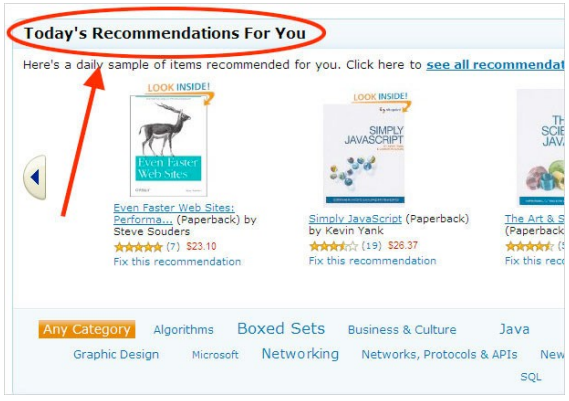
Applications of Machine Learning

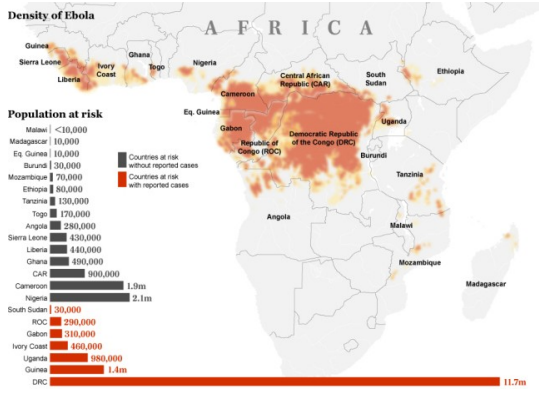




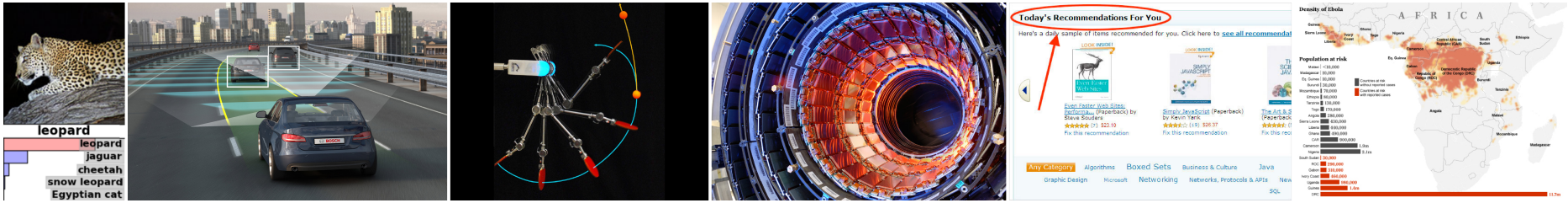








Mathematical Concepts in Machine Learning



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

Outline

Overview

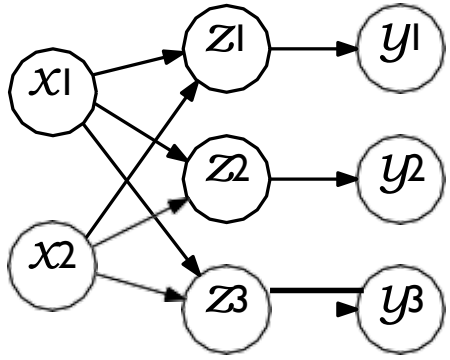
Introduction

Differentiation

Integration

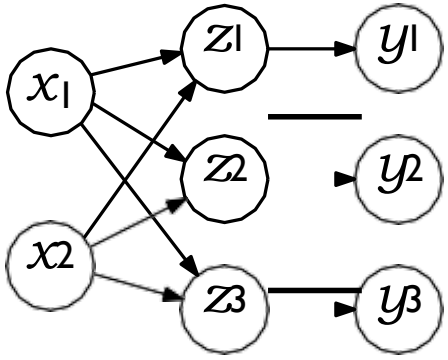
Feedforward Neural Network

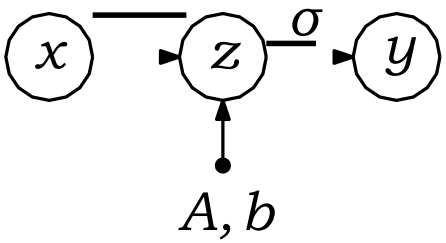
$$y = \sigma(z)$$
$$z = Ax + b$$



Feedforward Neural Network

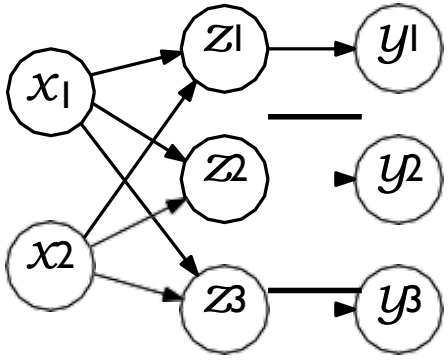
$$y = \sigma(z)$$
$$z = Ax + b$$

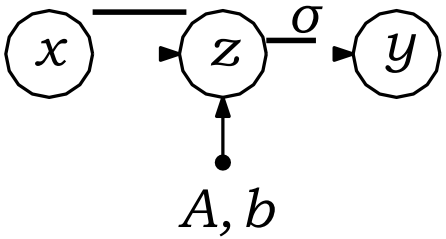


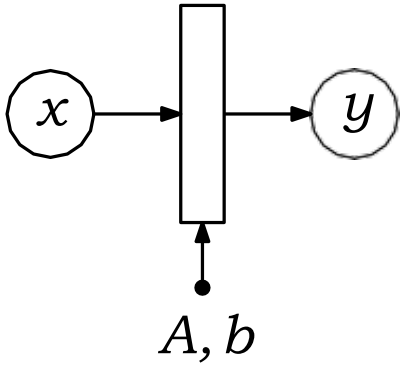


Feedforward Neural Network

$$y = \sigma(z)$$
$$z = Ax + b$$

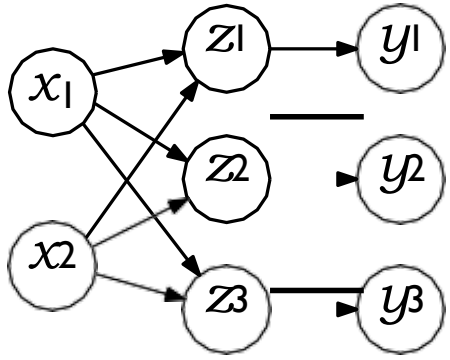


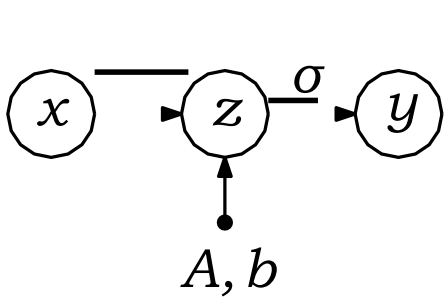


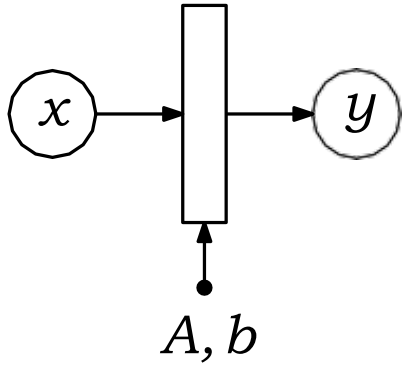


Feedforward Neural Network

$$y = \sigma(z)$$
$$z = Ax + b$$



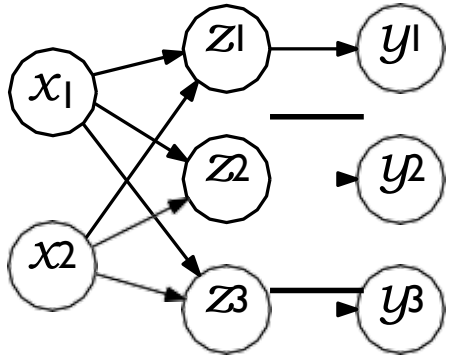


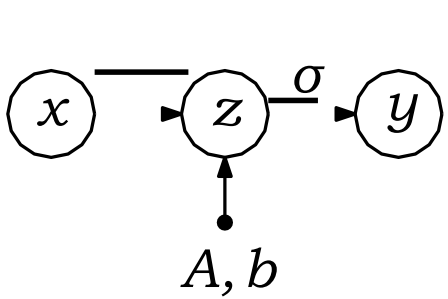


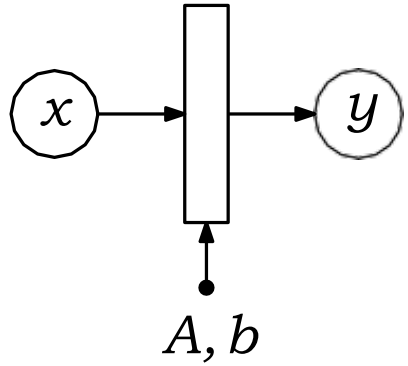
- Training a neural network means parameter optimization:
  Typically via some form of gradient descent
  - **Challenge 1: Differentiation.** Compute gradients of a loss function with respect to neural network parameters $\boldsymbol{A}, \boldsymbol{b}$

Feedforward Neural Network

$$y = \sigma(z)$$
$$z = Ax + b$$



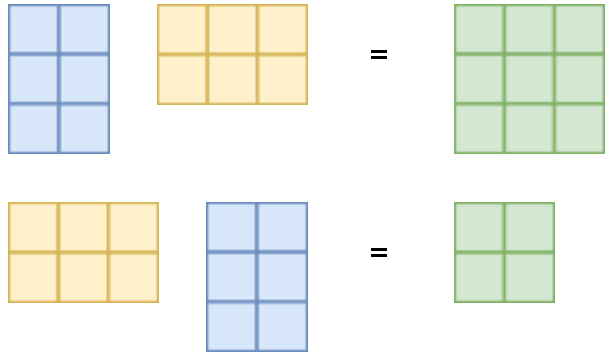




- Training a neural network means parameter optimization:
  Typically via some form of gradient descent
  - **Challenge 1: Differentiation.** Compute gradients of a loss function with respect to neural network parameters $\boldsymbol{A}, \boldsymbol{b}$
- Computing statistics (e.g., means, variances)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

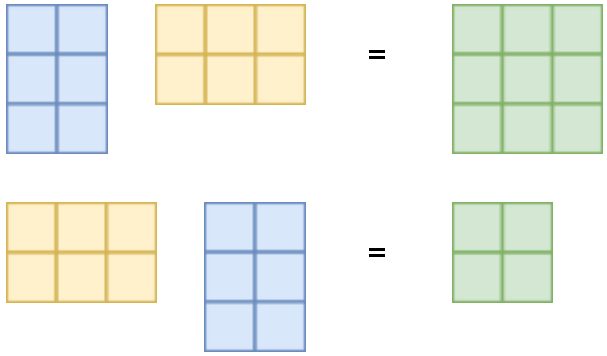
Background: Matrix Multiplication

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



Background: Matrix Multiplication

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

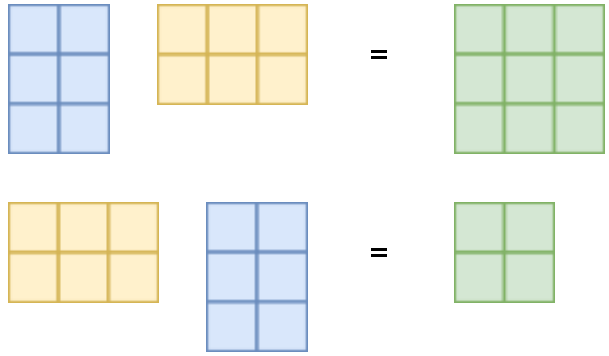


- When multiplying matrices, the "neighboring" dimensions have to fit:

$$\underbrace{[\,\cdot\,]_{n\times k}}_{n\times k}\;\underbrace{[\,\cdot\,]_{k\times m}}_{k\times m} = \underbrace{[\,\cdot\,]_{n\times m}}_{n\times m}$$

Background: Matrix Multiplication

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



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

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

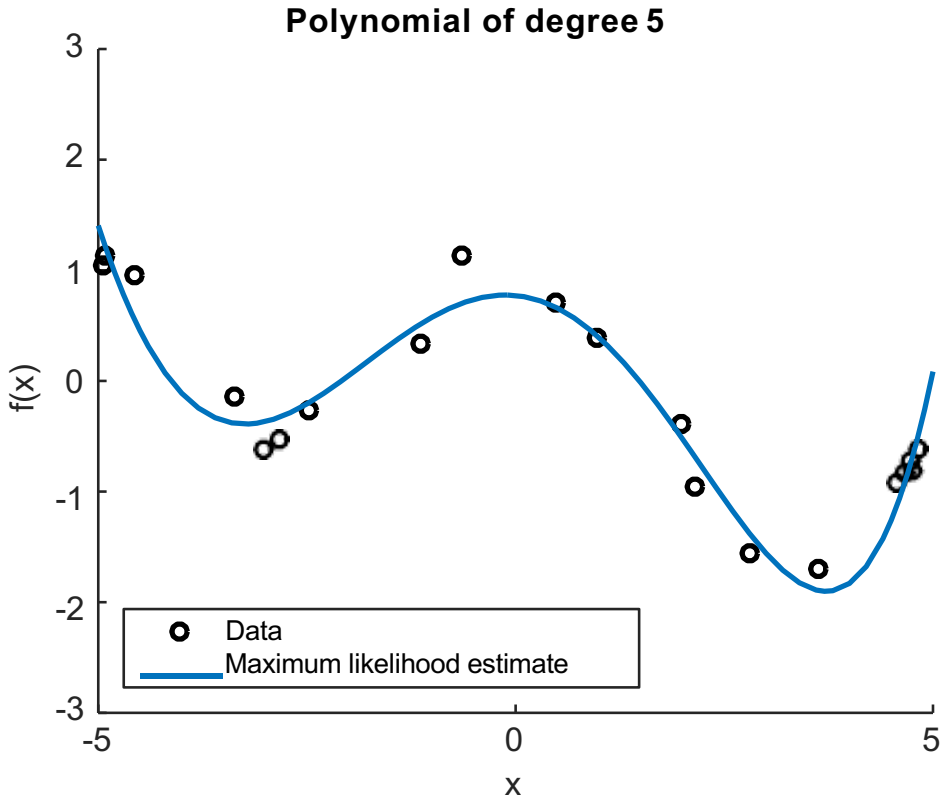
$$y = Ax \qquad \texttt{y = A.dot(x)}$$

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

$$C = AB \qquad \texttt{C= A.dot(B)}$$

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

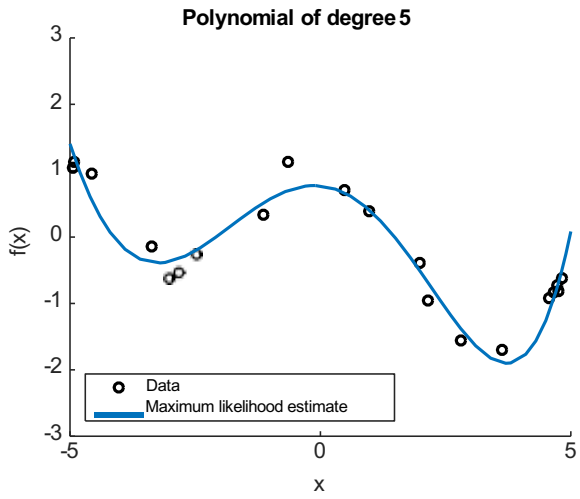
Curve Fitting (Regression) in Machine Learning (1)



- Setting: Given inputs x, predict outputs/targets y
- Model f that depends on parameters θ. Examples:
  - Linear model: $f(x, \theta) = \theta^\top x, \quad x, \theta \in \mathbb{R}^D$
  - Neural network: $f(x, \theta) = NN(x, \theta)$
- Training data, e.g., N pairs (x, y) of inputs x and observations y

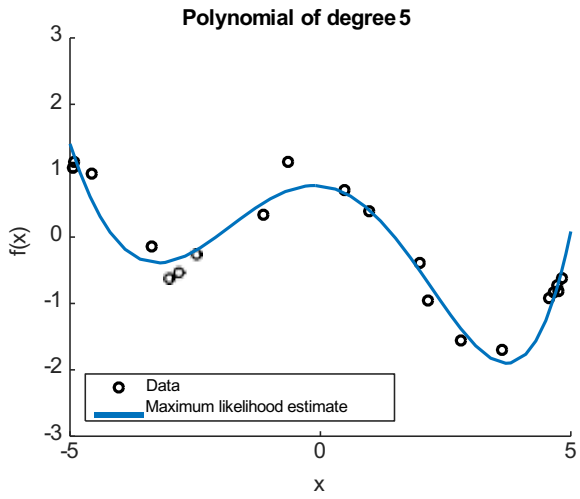
Curve Fitting (Regression) in Machine Learning (2)

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



Curve Fitting (Regression) in Machine Learning (2)

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



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

Overview

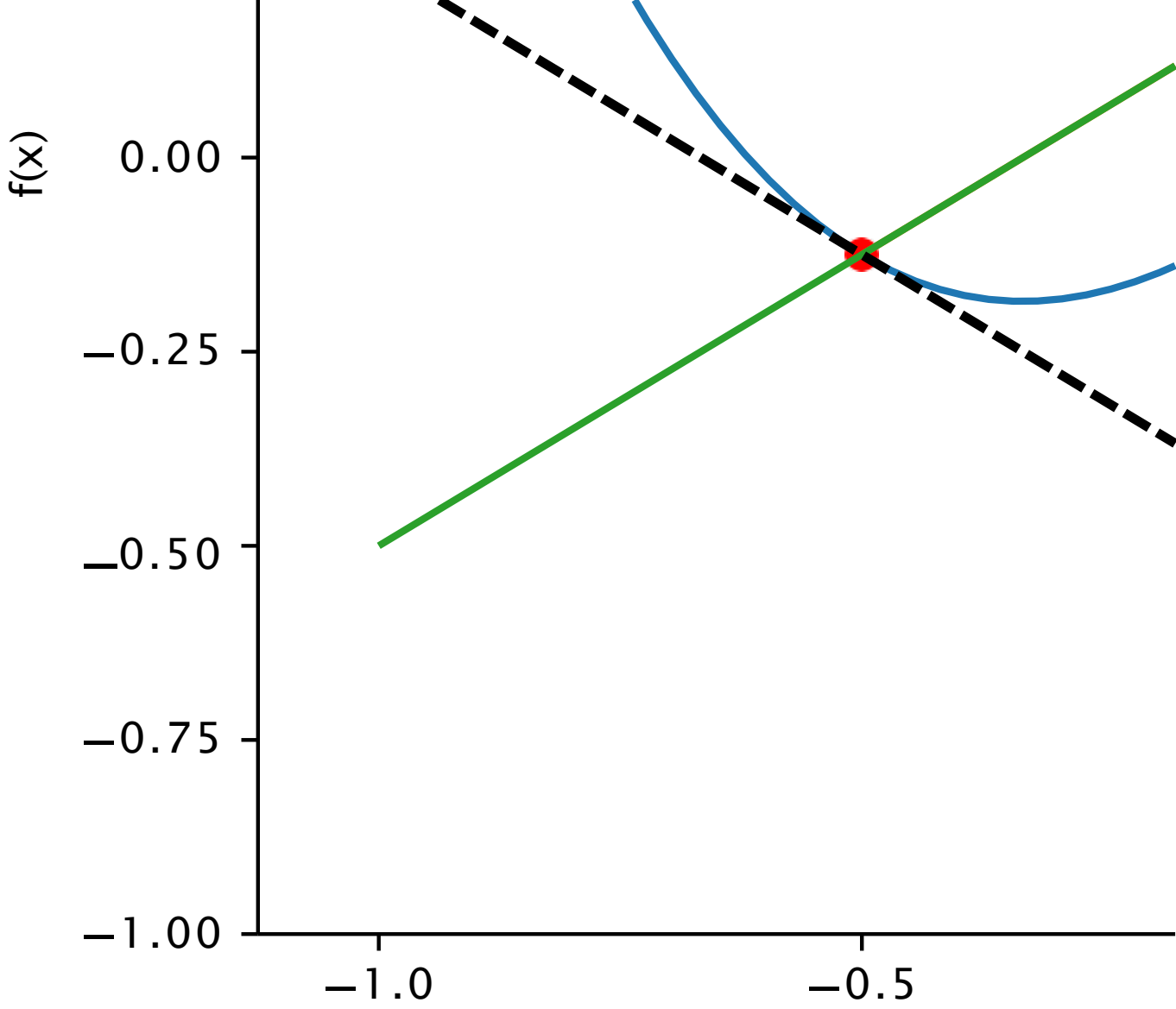
Differentiation: Outline

1. Scalar differentiation: $f : \mathbf{R} \tilde{\rightarrow} \mathbf{R}$
2. Multivariate case: $f : \mathbf{R}^N \tilde{\rightarrow} \mathbf{R}$
3. Vector fields: $f : \mathbf{R}^N \tilde{\rightarrow} \mathbf{R}^M$
4. General derivatives: $f : \mathbf{R}^{M \hat{} N} \tilde{\rightarrow} \mathbf{R}^{P \hat{} Q}$

Scalar Diff

- Derivati v

- Slope



Examples

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

Rules

- Sum Rule

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

Rules

- Sum Rule

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

- Product Rule

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

Rules

§ Sum Rule

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

§ Product Rule

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

§ Chain Rule

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

Example: Chain Rule

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

$$g(z) = \tanh(z)$$

$$z = f(x) = x^n$$

$$(g \circ f)'(x)$$

Example: Chain Rule

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

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

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

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

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

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

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

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

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

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

- Jacobian vector (gradient) collects all partial derivatives:

$$\frac{df}{dx} = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \cdots & \frac{\partial f}{\partial x_N} \end{bmatrix} \in \mathbf{R}^{1 \times N}$$

Note: This is a row vector.

Example

$$f : \mathbf{R}^2 \to \mathbf{R}$$

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

Example

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

- 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

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

- 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:

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

Rules

- Sum Rule

$$\frac{\partial}{\partial x}\left(f(x)+g(x)\right)=\frac{\partial f}{\partial x}+\frac{\partial g}{\partial x}$$

- Product Rule

$$\frac{\partial}{\partial x}\left(f(x)g(x)\right)=\frac{\partial f}{\partial x}g(x)+f(x)\frac{\partial g}{\partial x}$$

- Chain Rule

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

Example: Chain Rule

- Consider the function

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

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

- Compute dL/dx. What is the dimension/size of dL/dx?

Example: Chain Rule

§ Consider the function

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

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

§ Compute dL/dx. What is the dimension/size of dL/dx?

§ $dL/dx \in \mathbb{R}^{1 \times N}$

$$\frac{dL}{dx} = \frac{dL}{de}\frac{de}{dx}$$

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

$$\frac{de}{dx} = -A \in \mathbb{R}^{M \times N} \tag{2}$$

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

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

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

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

$f : \mathbf{R}^N \tilde{N} \mathbf{R}^M$

$$y \text{“} f(x) \in \mathbf{R}^M, \quad x \in \mathbf{R}^N$$

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

§ Jacobian matrix (collection of all partial derivatives)

$$\begin{bmatrix} \frac{dy_1}{dx} \\ \vdots \\ \frac{dy_M}{dx} \end{bmatrix} \text{“} \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_N} \\ \vdots & & \vdots \\ \frac{\partial f_M}{\partial x_1} & \cdots & \frac{\partial f_M}{\partial x_N} \end{bmatrix} \in \mathbf{R}^{M \hat{} N}$$

Example

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

- Compute the gradient df/dx
  - Dimension of df/dx:

Example

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

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

Example

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

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

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

Example

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

- Compute the gradient $\frac{df}{dx}$
  - Dimension of $\frac{df}{dx}$:
    Since $f : \mathbb{R}^N \to \mathbb{R}^M$, it follows that $\frac{df}{dx} \in \mathbb{R}^{M \times N}$
  - Gradient:

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

$$\frac{df}{dx} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_N} \\ \vdots & & \vdots \\ \frac{\partial f_M}{\partial x_1} & \cdots & \frac{\partial f_M}{\partial x_N} \end{bmatrix} = \begin{bmatrix} A_{11} & \cdots & A_{1N} \\ \vdots & & \vdots \\ A_{M1} & \cdots & A_{MN} \end{bmatrix} = A \tag{3}$$

Chain Rule

$$\frac{\mathrm{B}}{\mathrm{B}x}\mathrm{p}g^{\prime\prime}\ f\mathrm{q}\mathrm{p}x\mathrm{q}^{\text{“}}\ \ \text{-}\frac{\mathrm{B}}{\mathrm{B}x}\ g\mathrm{p}f\ \mathrm{p}x\mathrm{q}\mathrm{q}^{\text{“}}\ \ \ \frac{\mathrm{B}g}{\mathrm{B}f}\frac{\mathrm{B}f}{\mathrm{B}x}$$

Example

§ Consider $f : \mathbf{R}^2 \to \mathbf{R}$, $x : \mathbf{R} \to \mathbf{R}^2$

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

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

Example

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

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

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

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

Example

§ Consider $f : \mathbf{R}^2 \to \mathbf{R}$, $x : \mathbf{R} \to \mathbf{R}^2$

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

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

§ 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 $f : \mathbf{R}^2 \to \mathbf{R}$, $x : \mathbf{R} \to \mathbf{R}^2$

$$f(x) = f(x_1, x_2) = x_1^2 + 2x_2$$

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

§ The dimensions df/dx and dx/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 $f: \mathbf{R}^D$ Ñ $\mathbf{R}^E$ is an E ˆ D-matrix: #

target dimensions ˆ #parameters

with

$$\frac{df}{dx} \mathrm{P} \mathbf{R}^{E\hat{}\ D}, \qquad d\,f\,\mathrm{r}e, d\mathrm{s}\text{“} \frac{\mathrm{B}f_e}{\mathrm{B}x_d}$$

Derivatives with Matrices

§ Re-cap: Gradient of a function $f: \mathbf{R}^D \tilde{N} \mathbf{R}^E$ is an $E \hat{} D$-matrix: #

target dimensions $\hat{}$ #parameters

with

$$\frac{df}{dx} P \mathbf{R}^{E \hat{} D}, \qquad df re, ds" \frac{Bf_e}{Bx_d}$$

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

Derivatives with Matrices

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

target dimensions × #parameters

with

$$\frac{df}{dx} \in \mathbf{R}^{E \times D}, \qquad df[e, d] = \frac{\partial f_e}{\partial x_d}$$

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

§ Assume $f: \mathbf{R}^{M \times N} \to \mathbf{R}^{P \times Q}$, 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)

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

Derivatives with Matrices: Example (1)

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

$$\frac{df}{dA} \in \mathbb{R}^{M \times (M \times N)}$$

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

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)

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

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

Derivatives with Matrices: Example (2)

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

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

(4)

Derivatives with Matrices: Example (2)

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

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

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

Derivatives with Matrices: Example (2)

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

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

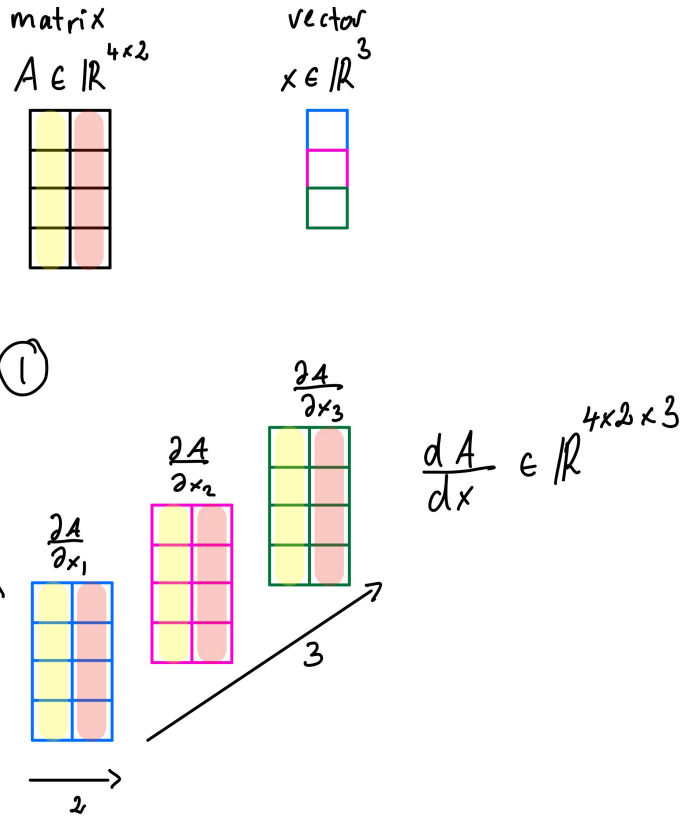
$$\frac{\partial f_i}{\partial A_{k \neq i,:}} = \mathbf{0}^\top \in \mathbb{R}^{1 \times 1 \times N}$$

$$\rightsquigarrow \mathbf{0}^\top$$

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

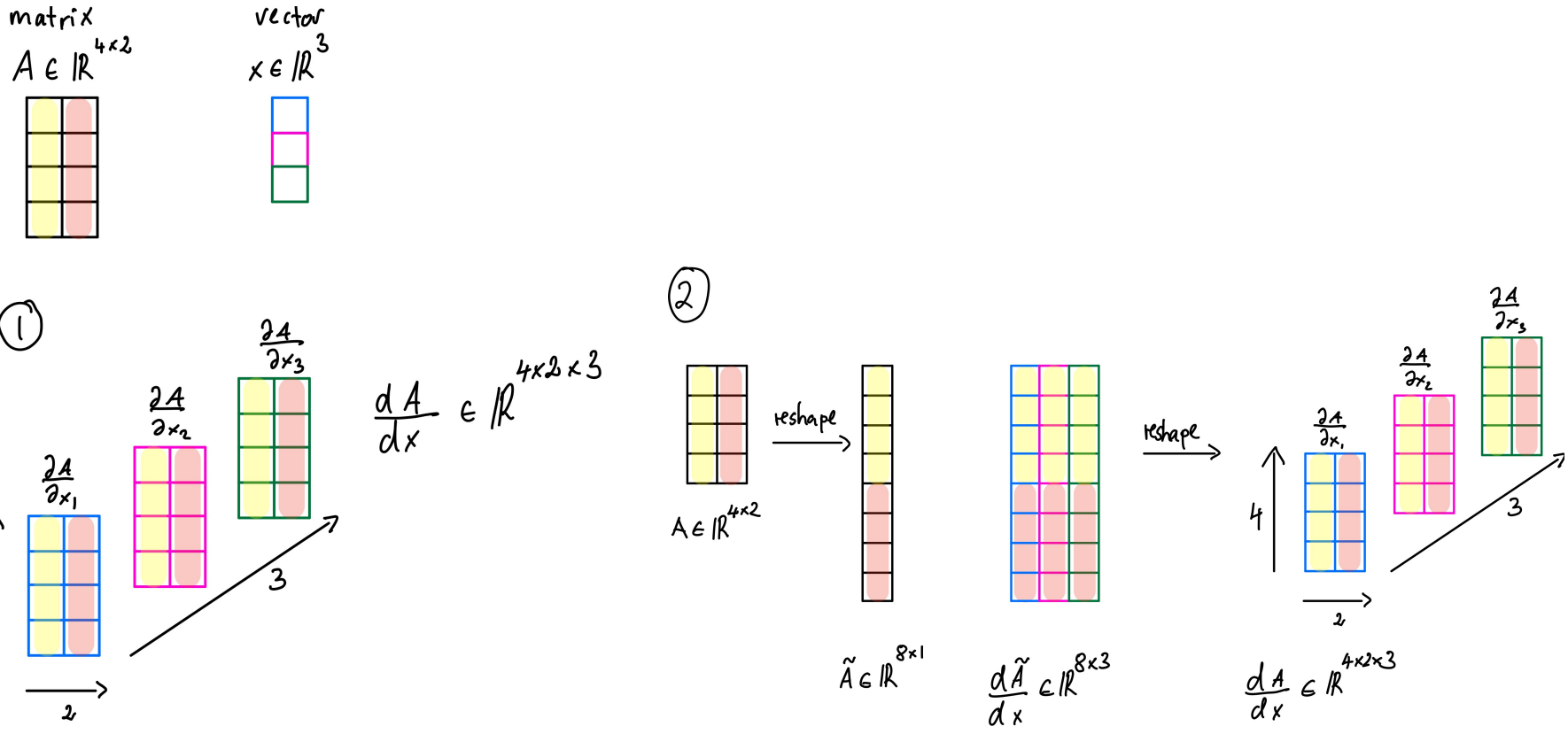
Example: Higher-Order Tensors

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

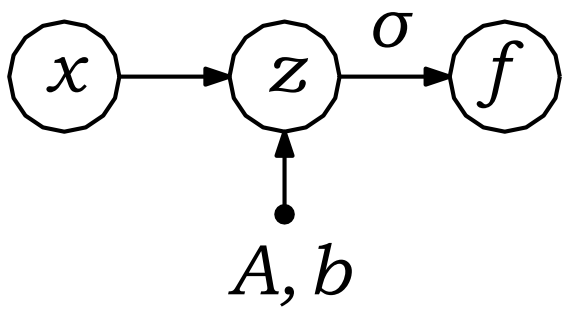


Example: Higher-Order Tensors

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

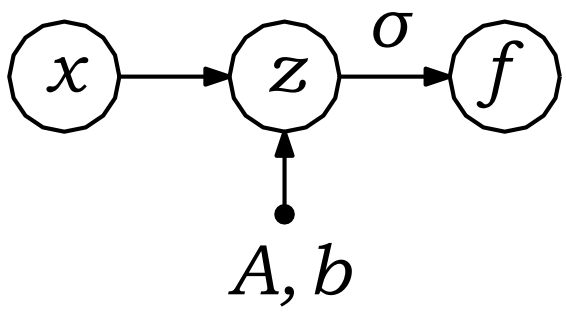


Gradients of a Single-Layer Neural Network (1)



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

Gradients of a Single-Layer Neural Network (1)

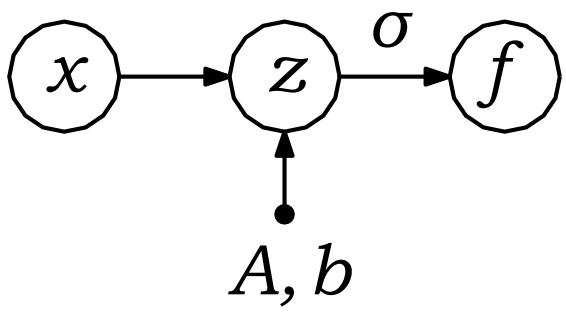


$$f = \tanh(\underbrace{Ax + b}_{=:z \in \mathbb{R}^M}) \in \mathbb{R}^M, \quad x \in \mathbb{R}^N, A \in \mathbb{R}^{M \times N}, b \in \mathbb{R}^M$$

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

$M \times M$ $M \times M$

Gradients of a Single-Layer Neural Network (1)



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

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

$$\frac{\partial f}{\partial z} \in \mathbb{R}^{M \times M} \quad \frac{\partial z}{\partial b} \in \mathbb{R}^{M \times M}$$

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

$$\frac{\partial f}{\partial z} \in \mathbb{R}^{M \times M} \quad \frac{\partial z}{\partial A} \in \mathbb{R}^{M \times (M \times N)}$$

Gradients of a Single-Layer Neural Network (2)

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

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

$M \times M$ $M \times M$

Gradients of a Single-Layer Neural Network (2)

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

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

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

Gradients of a Single-Layer Neural Network (2)

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

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

$$\frac{\partial f}{\partial z} \in \mathbb{R}^{M \times M}, \quad \frac{\partial z}{\partial b} = I \in \mathbb{R}^{M \times M}$$

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

$$\frac{\partial z}{\partial b} = I \in \mathbb{R}^{M \times M} \tag{5}$$

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

Gradients of a Single-Layer Neural Network (2)

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

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

$$\frac{\partial f}{\partial z} \in \mathbb{R}^{M \times M} \quad \frac{\partial z}{\partial b} \in \mathbb{R}^{M \times M}$$

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

$$\frac{\partial z}{\partial b} = I \in \mathbb{R}^{M \times M} \tag{5}$$

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

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

Gradients of a Single-Layer Neural Network (3)

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

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

Gradients of a Single-Layer Neural Network (3)

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

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

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

Gradients of a Single-Layer Neural Network (3)

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

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

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

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

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

Gradients of a Single-Layer Neural Network (3)

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

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

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

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

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

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

Putting Things Together

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

Putting Things Together

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

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

Putting Things Together

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

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

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

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

is minimized

Putting Things Together

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

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

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

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

is minimized

- Partial derivatives:

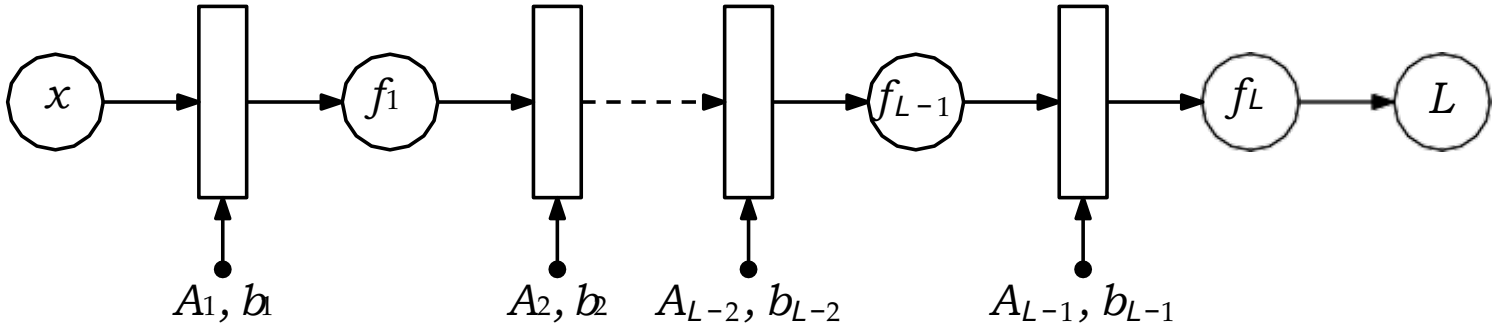
$$\frac{\partial L}{\partial \boldsymbol{A}} = \frac{\partial L}{\partial \boldsymbol{e}} \frac{\partial \boldsymbol{e}}{\partial f} \frac{\partial f}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{A}}$$

$$\frac{\partial L}{\partial \boldsymbol{b}} = \frac{\partial L}{\partial \boldsymbol{e}} \frac{\partial \boldsymbol{e}}{\partial f} \frac{\partial f}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}$$

$\frac{\partial L}{\partial \boldsymbol{e}}$ ⏩ (1) $\quad \frac{\partial \boldsymbol{e}}{\partial f}$ ⏩ (2), (3) $\quad \frac{\partial f}{\partial \boldsymbol{z}}$ ⏩ (6)

$\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{A}}$ ⏩ (4) $\quad \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}$ ⏩ (5)

Gradients of a Multi-Layer Neural Network

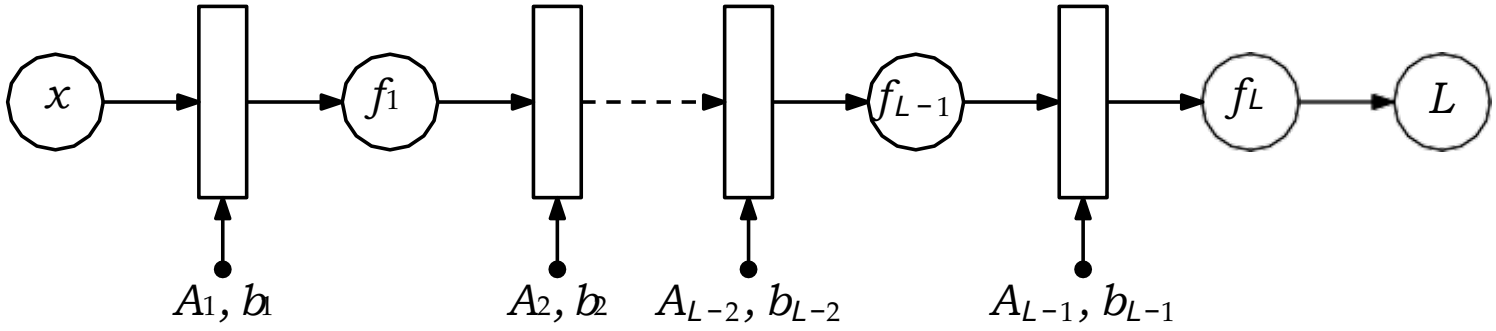


- Inputs $\boldsymbol{x}$, observed outputs $\boldsymbol{y}$
- Train multi-layer neural network with

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

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

Gradients of a Multi-Layer Neural Network



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

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

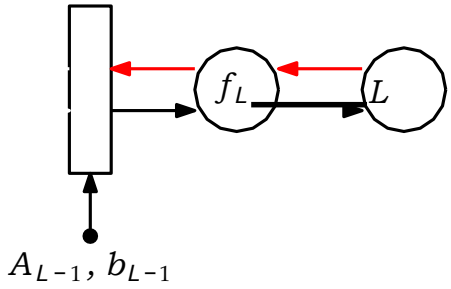
$$\boldsymbol{f}_i := \sigma_i(\boldsymbol{A}_{i-1}\boldsymbol{f}_{i-1} + \boldsymbol{b}_{i-1}), \quad i = 1, \ldots, L$$

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

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

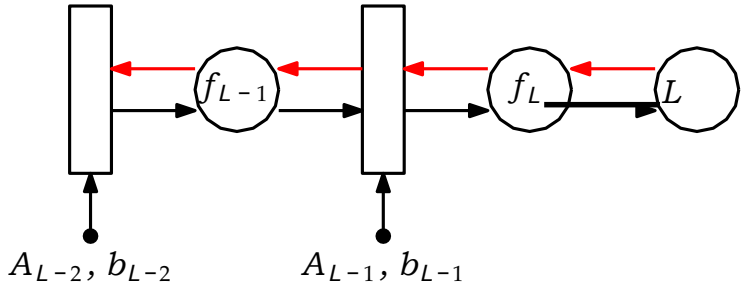
is minimized, where $\boldsymbol{\theta} := (\boldsymbol{A}_j, \boldsymbol{b}_j), \quad j = 0, \ldots, L-1$

Gradients of a Multi-Layer Neural Network



$$\frac{\partial L}{\partial \theta_{L-1}} = \frac{\partial L}{\partial f_L} \frac{\partial f_L}{\partial \theta_{L-1}}$$

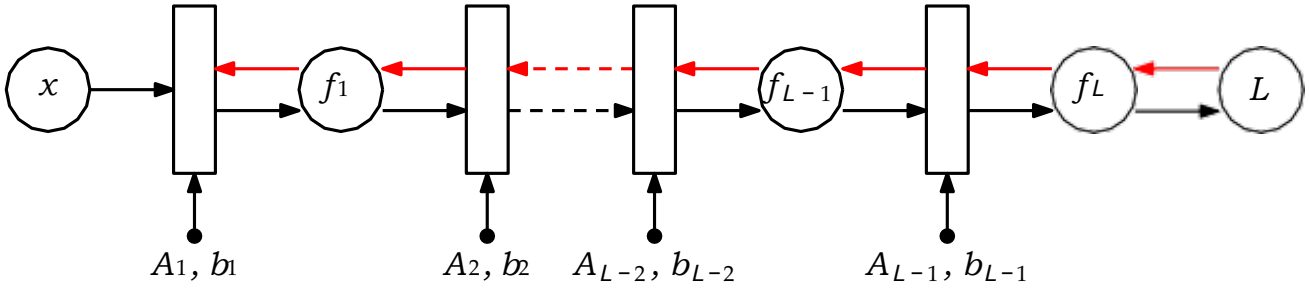
Gradients of a Multi-Layer Neural Network



$$\frac{\partial L}{\partial \theta_{L-1}} = \frac{\partial L}{\partial f_L} \frac{\partial f_L}{\partial \theta_{L-1}}$$

$$\frac{\partial L}{\partial \theta_{L-2}} = \frac{\partial L}{\partial f_L} \boxed{\frac{\partial f_L}{\partial f_{L-1}} \frac{\partial f_{L-1}}{\partial \theta_{L-2}}}$$

Gradients of a Multi-Layer Neural Network

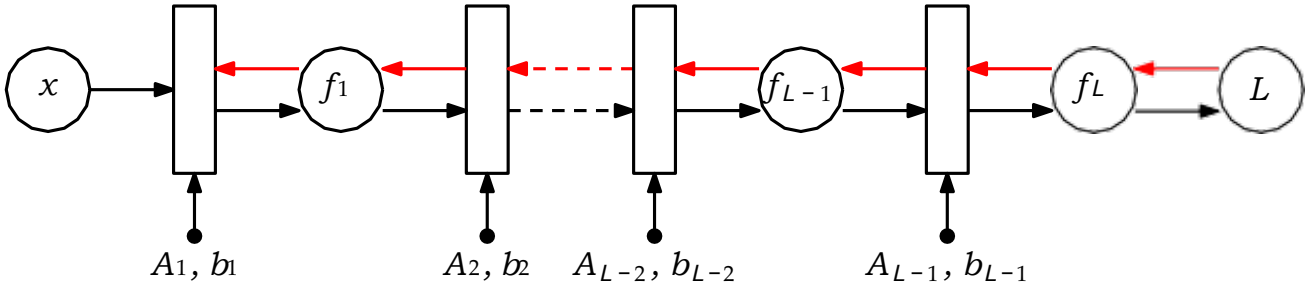


$$\frac{\partial L}{\partial \theta_{L-1}} = \frac{\partial L}{\partial f_L} \frac{\partial f_L}{\partial \theta_{L-1}}$$

$$\frac{\partial L}{\partial \theta_{L-2}} = \frac{\partial L}{\partial f_L} \boxed{\frac{\partial f_L}{\partial f_{L-1}} \frac{\partial f_{L-1}}{\partial \theta_{L-2}}}$$

$$\frac{\partial L}{\partial \theta_{L-3}} = \frac{\partial L}{\partial f_L} \frac{\partial f_L}{\partial f_{L-1}} \boxed{\frac{\partial f_{L-1}}{\partial f_{L-2}} \frac{\partial f_{L-2}}{\partial \theta_{L-3}}}$$

Gradients of a Multi-Layer Neural Network



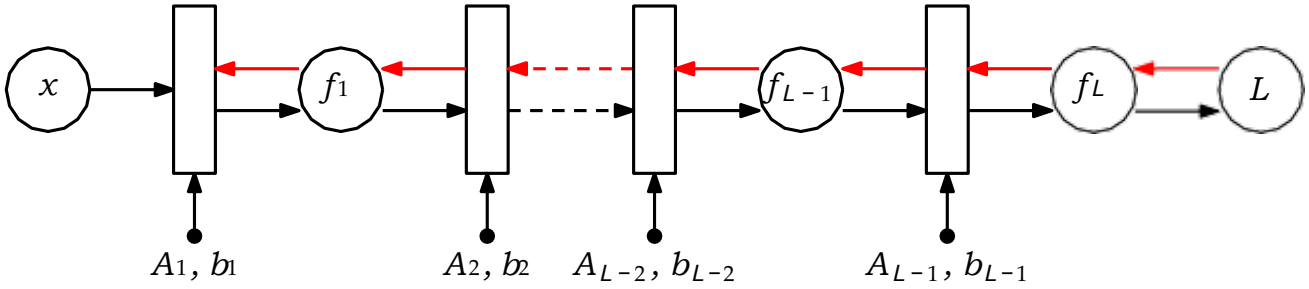
$$\frac{\partial L}{\partial \theta_{L-1}} = \frac{\partial L}{\partial f_L} \frac{\partial f_L}{\partial \theta_{L-1}}$$

$$\frac{\partial L}{\partial \theta_{L-2}} = \frac{\partial L}{\partial f_L} \boxed{\frac{\partial f_L}{\partial f_{L-1}} \frac{\partial f_{L-1}}{\partial \theta_{L-2}}}$$

$$\frac{\partial L}{\partial \theta_{L-3}} = \frac{\partial L}{\partial f_L} \frac{\partial f_L}{\partial f_{L-1}} \boxed{\frac{\partial f_{L-1}}{\partial f_{L-2}} \frac{\partial f_{L-2}}{\partial \theta_{L-3}}}$$

$$\frac{\partial L}{\partial \theta_i} = \frac{\partial L}{\partial f_L} \frac{\partial f_L}{\partial f_{L-1}} \cdots \boxed{\frac{\partial f_{i+2}}{\partial f_{i+1}} \frac{\partial f_{i+1}}{\partial \theta_i}}$$

Gradients of a Multi-Layer Neural Network



$$\frac{\partial L}{\partial \theta_{L-1}} = \frac{\partial L}{\partial f_L}\frac{\partial f_L}{\partial \theta_{L-1}}$$

$$\frac{\partial L}{\partial \theta_{L-2}} = \frac{\partial L}{\partial f_L}\boxed{\frac{\partial f_L}{\partial f_{L-1}}\frac{\partial f_{L-1}}{\partial \theta_{L-2}}}$$

$$\frac{\partial L}{\partial \theta_{L-3}} = \frac{\partial L}{\partial f_L}\frac{\partial f_L}{\partial f_{L-1}}\boxed{\frac{\partial f_{L-1}}{\partial f_{L-2}}\frac{\partial f_{L-2}}{\partial \theta_{L-3}}}$$

$$\frac{\partial L}{\partial \theta_i} = \frac{\partial L}{\partial f_L}\frac{\partial f_L}{\partial f_{L-1}}\cdots\boxed{\frac{\partial f_{i+2}}{\partial f_{i+1}}\frac{\partial f_{i+1}}{\partial \theta_i}}$$

- 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 $\boldsymbol{y} = NN(\boldsymbol{x}, \boldsymbol{\theta}) + \boldsymbol{\varepsilon}, \quad \boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ then the log-likelihood is

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

Training Neural Networks as Maximum Likelihood Estimation

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

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

  - Find θ^* by minimizing the negative log-likelihood:

$$\theta^* = \arg\min_\theta -\log p(y|x, \theta)$$
$$= \arg\min_\theta \frac{1}{2}\|y - NN(x, \theta)\|^2$$
$$= \arg\min_\theta L(\theta)$$

Training Neural Networks as Maximum Likelihood Estimation

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

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

  - Find θ^* by minimizing the negative log-likelihood:

$$\theta^* = \arg\min_\theta -\log p(y|x, \theta)$$
$$= \arg\min_\theta \frac{1}{2}\|y - NN(x, \theta)\|^2$$
$$= \arg\min_\theta L(\theta)$$

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

Example: Linear Regression (1)

- Linear regression with a polynomial of order M:

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

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

Example: Linear Regression (1)

- Linear regression with a polynomial of order M:

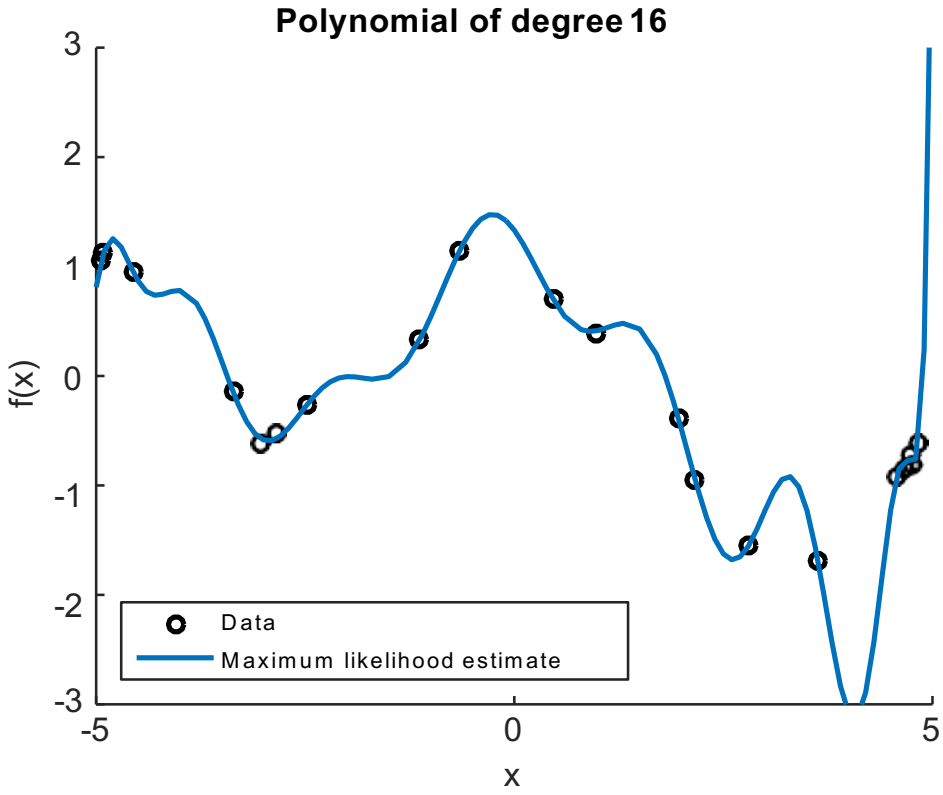
$$y = f(x, \boldsymbol{\theta}) + \varepsilon, \quad \varepsilon \sim \mathcal{N}(0, \sigma_\varepsilon^2)$$

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

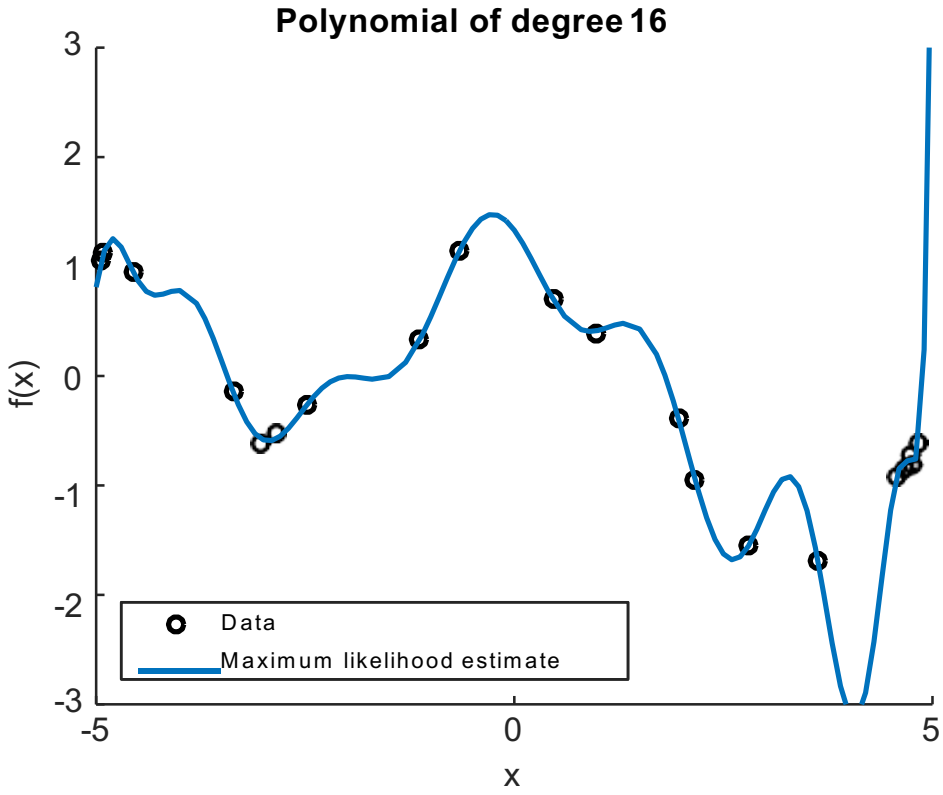
- Given inputs x_i and corresponding (noisy) observations y_i, $i = 1, \ldots, N$, find parameters $\boldsymbol{\theta} = [\theta_0, \ldots, \theta_M]^\top$, that minimize the squared loss (equivalently: maximize the likelihood)

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

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

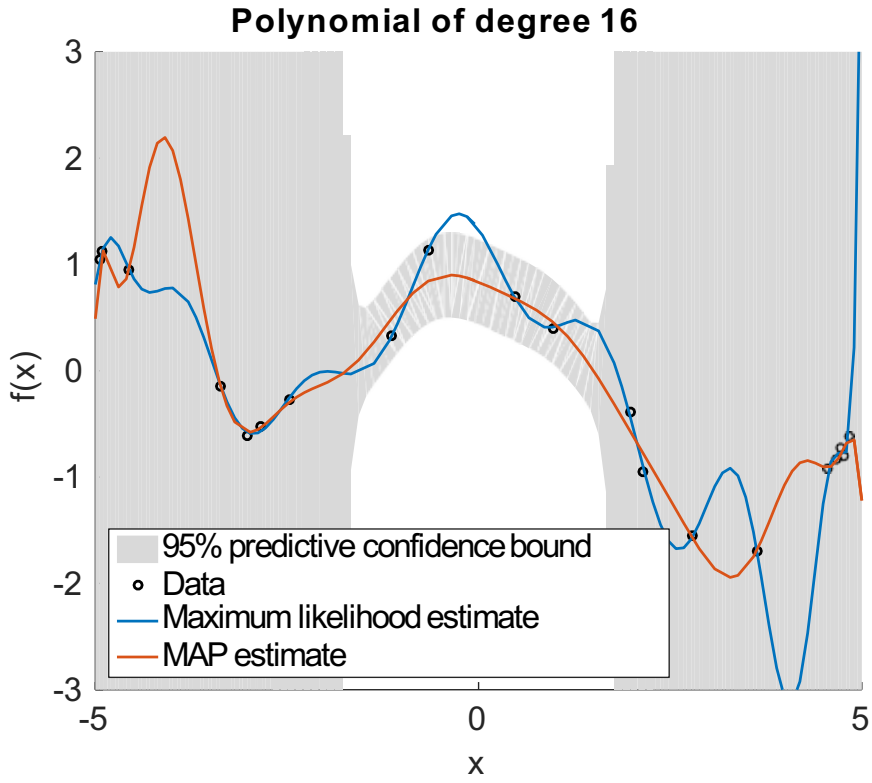
Integration: Outline

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:

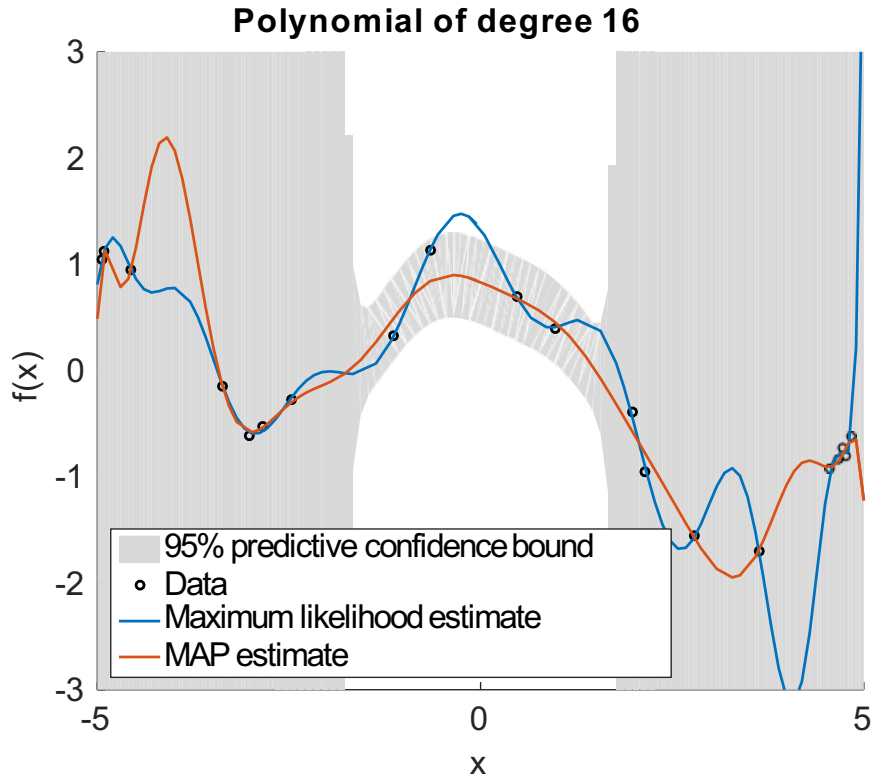
$$p(y|x) = \int p(y|x, \boldsymbol{\theta}) p(\boldsymbol{\theta}) d\boldsymbol{\theta}$$



Bayesian Integration to Avoid Overfitting

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

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

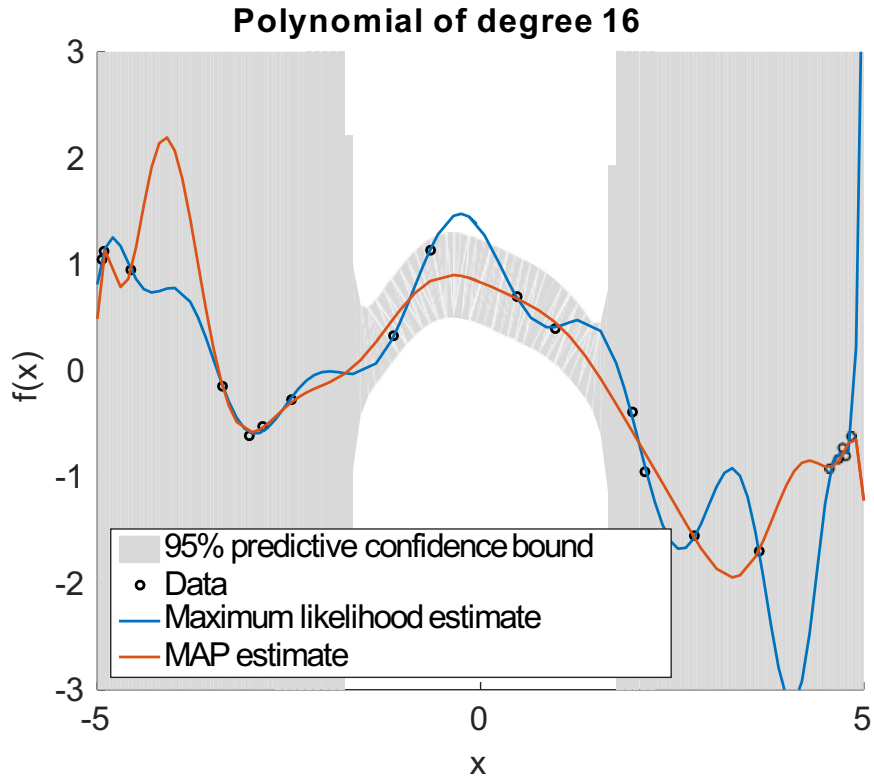


- More details on what $p(\boldsymbol{\theta})$ is ⏩ Tutorials later this week

Bayesian Integration to Avoid Overfitting

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

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



- 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:

$$\mathbf{E}_x[x] = \int_{\mathcal{X}} x\, p(x)\, dx = \boldsymbol{\mu}_x$$

$$\mathbf{V}_x[x] = \int_{\mathcal{X}} (x - \boldsymbol{\mu}_x)(x - \boldsymbol{\mu}_x)^\top dx$$

$$\mathrm{Cov}[x, y] = \iint_{\mathcal{X},\mathcal{Y}} (x - \boldsymbol{\mu}_x)(y - \boldsymbol{\mu}_y)^\top dx\, dy$$

Computing Statistics of Random Variables

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

$$\mathbf{E}_x[x] = \int x p(x)\,dx = \mu_x$$

$$\mathbf{V}_x[x] = \int (x - \mu_x)(x - \mu_x)^\top p(x)\,dx$$

$$\mathrm{Cov}[x, y] = \iint (x - \mu_x)(y - \mu_y)^\top p(x, y)\,dx\,dy$$

- 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 $\{x^{(s)}\}$ from a given probability distribution $p(x)$, 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 $\{x^{(s)}\}$ from a given probability distribution $p(x)$, e.g., for simulation or representations of data distributions
  2. **Problem 2:** Compute expectations of functions under that distribution:

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

Monte Carlo Methods — Motivation

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

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

  - Example: Means/variances of distributions, predictions

Complication: Integral cannot be evaluated analytically

Problem 2: Monte Carlo Estimation

§ Computing expectations via statistical sampling:

$$\mathbb{E}_x[f(x)] = \int f(x)p(x)dx$$

$$\approx \frac{1}{S}\sum_{s=1}^{S} f(x^{(s)}), \quad x^{(s)} \sim p(x)$$

Problem 2: Monte Carlo Estimation

- Computing expectations via statistical sampling:

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

$$\approx \frac{1}{S}\sum_{s=1}^{S} f(x^{(s)}), \quad x^{(s)} \sim p(x)$$

- Making predictions (e.g., Bayesian regression with inputs x and targets y)

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

Posterior

Parameter distribution

$$\approx \frac{1}{S}\sum_{s=1}^{S} p(y|\theta^{(s)}, x), \quad \theta^{(s)} \sim p(\theta)$$

Problem 2: Monte Carlo Estimation

- Computing expectations via statistical sampling:

$$\mathbb{E}_x[f(x)] = \int f(x)p(x)dx$$

$$\approx \frac{1}{S}\sum_{s=1}^{S} f(x^{(s)}), \quad x^{(s)} \sim p(x)$$

- Making predictions (e.g., Bayesian regression with inputs x and targets y)

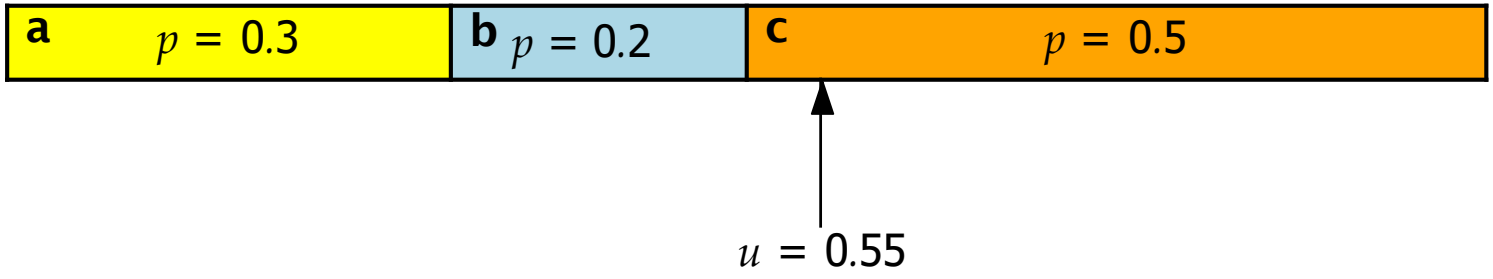
$$p(y|x) = \int p(y|\theta, x)p(\theta)d\theta$$

Parameter distribution

$$\approx \frac{1}{S}\sum_{s=1}^{S} p(y|\theta^{(s)}, x), \quad \theta^{(s)} \sim p(\theta)$$

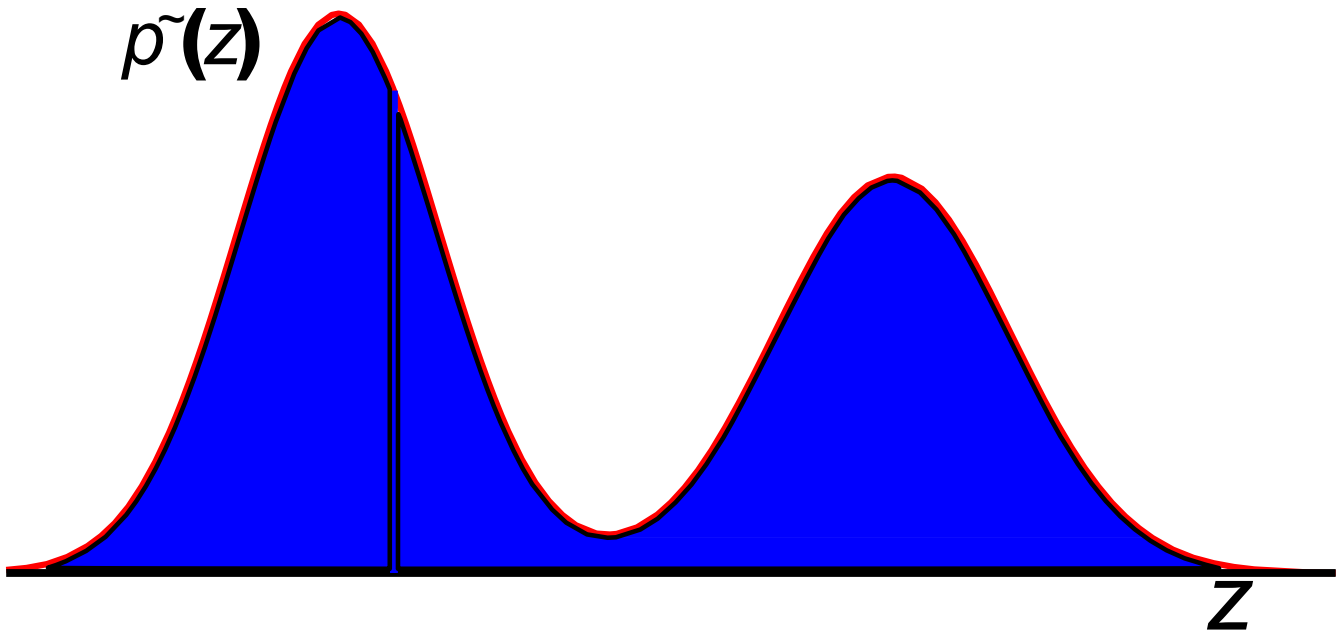
- **Key problem:** Generating samples from $p(x)$ or $p(\theta)$
  - Need to solve **Problem 1**

Sampling Discrete Values



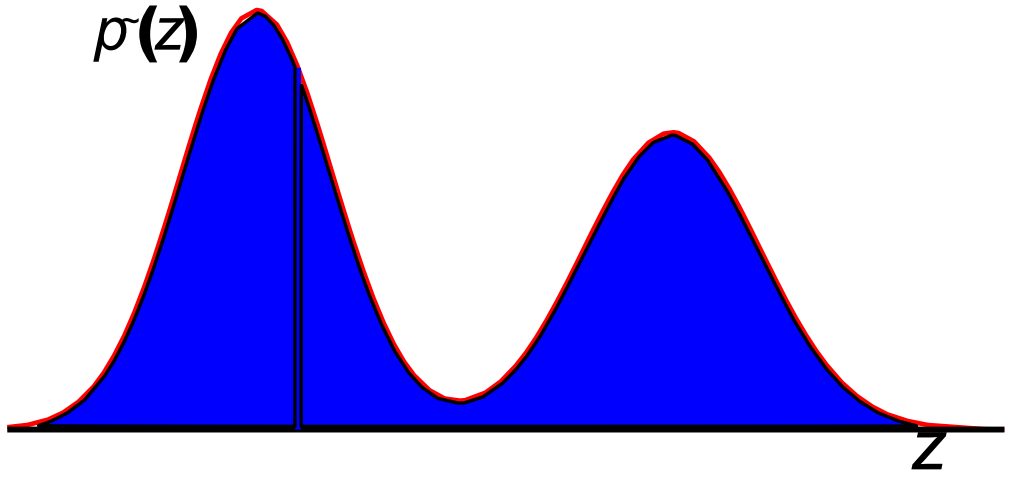
- $u \sim U[0, 1]$, where U is the uniform distribution
- $u = 0.55 \Rightarrow x = c$

Continuous Variables



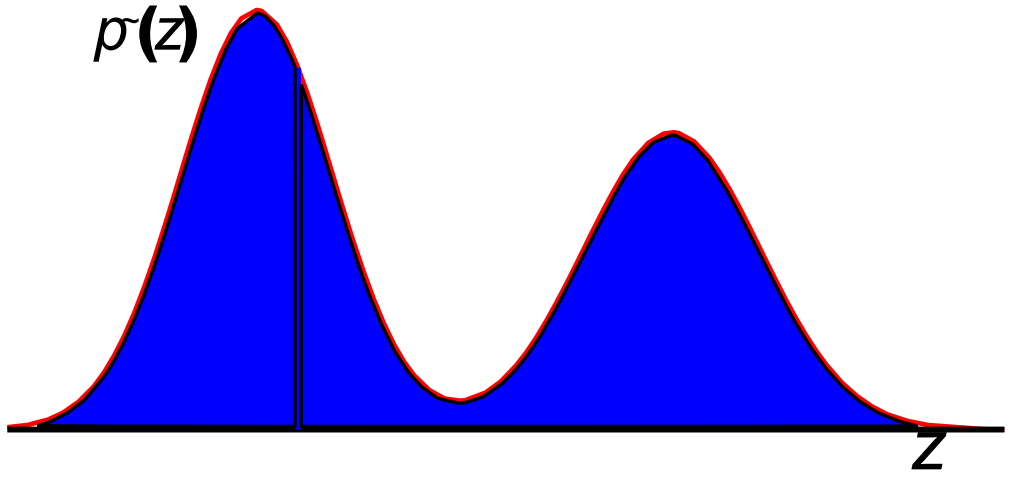
- More complicated
- Geometrically, we wish to sample uniformly from the area under the curve
  - Two algorithms here:
    - Rejection sampling
    - Importance sampling

Rejection Sampling: Setting



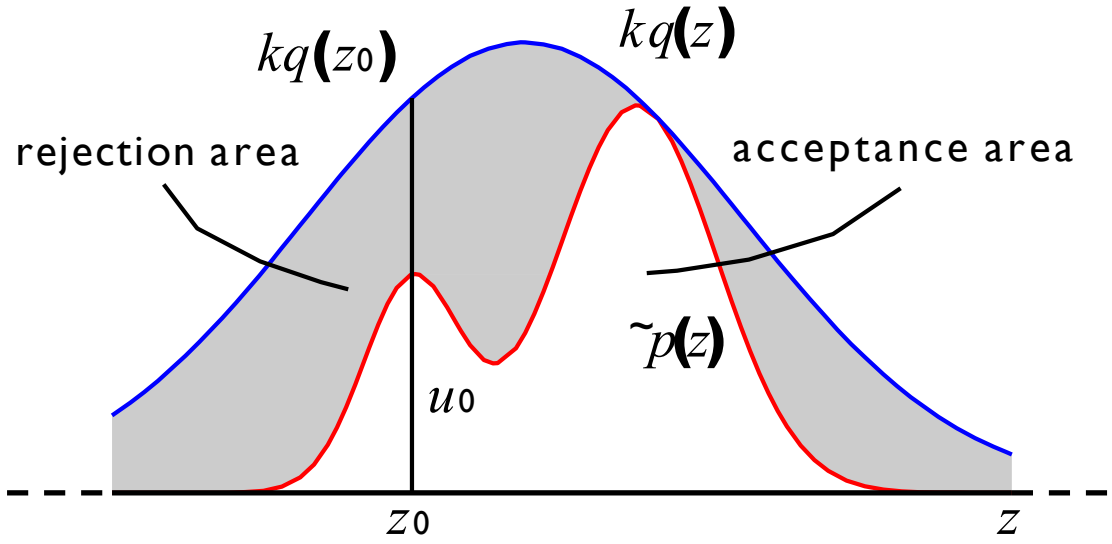
- Assume:
  - Sampling from $p(z)$ is difficult
  - Evaluating $\tilde{p}(z)$" $Zp(z)$ is easy (and Z may be unknown)

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 $kq(z) \geq \tilde{p}(z)$

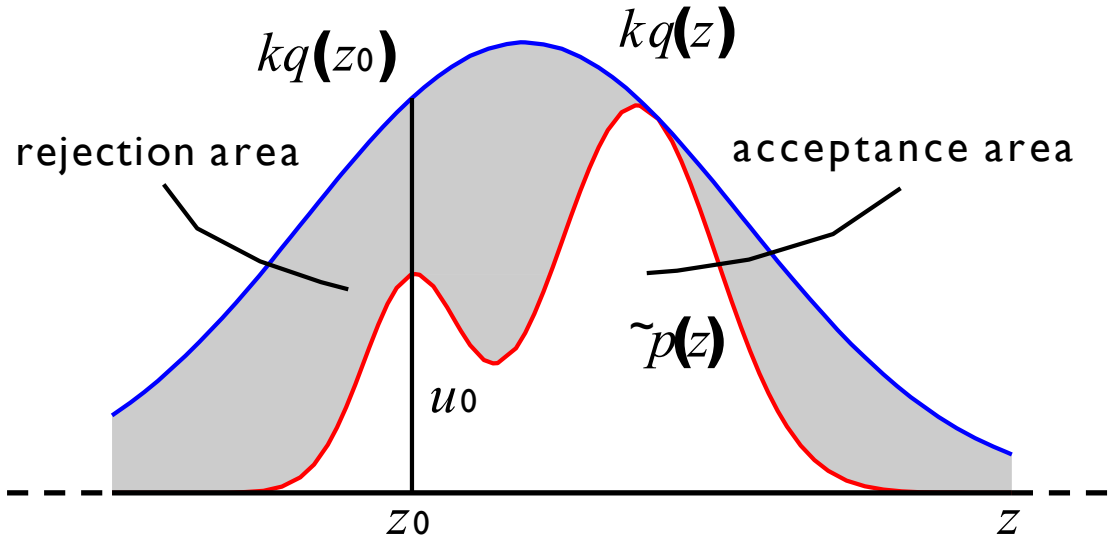
Rejection Sampling: Algorithm



Adapted from PRML (Bishop, 2006)

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

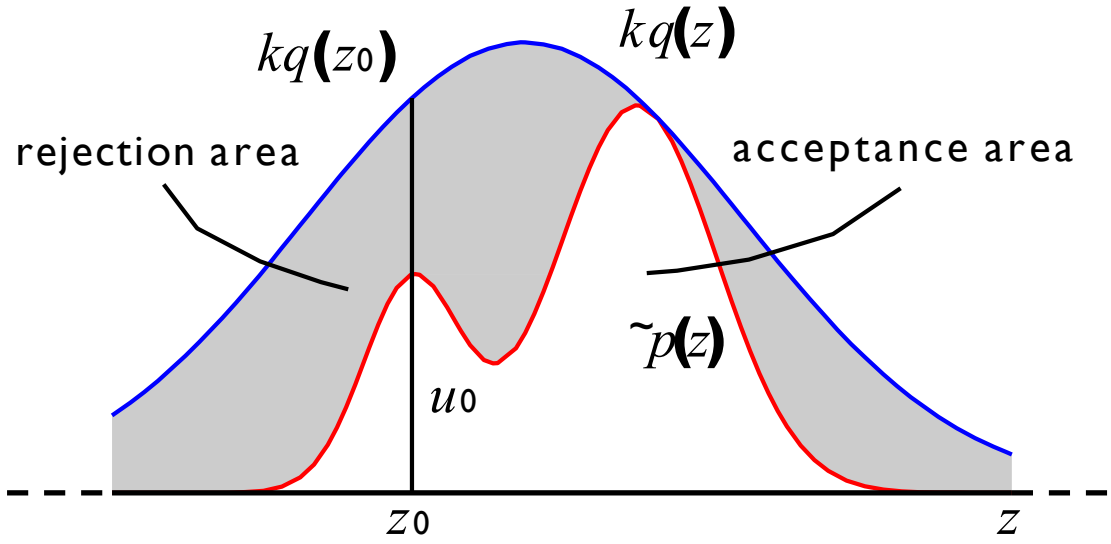
Properties



Adapted from PRML (Bishop, 2006)

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

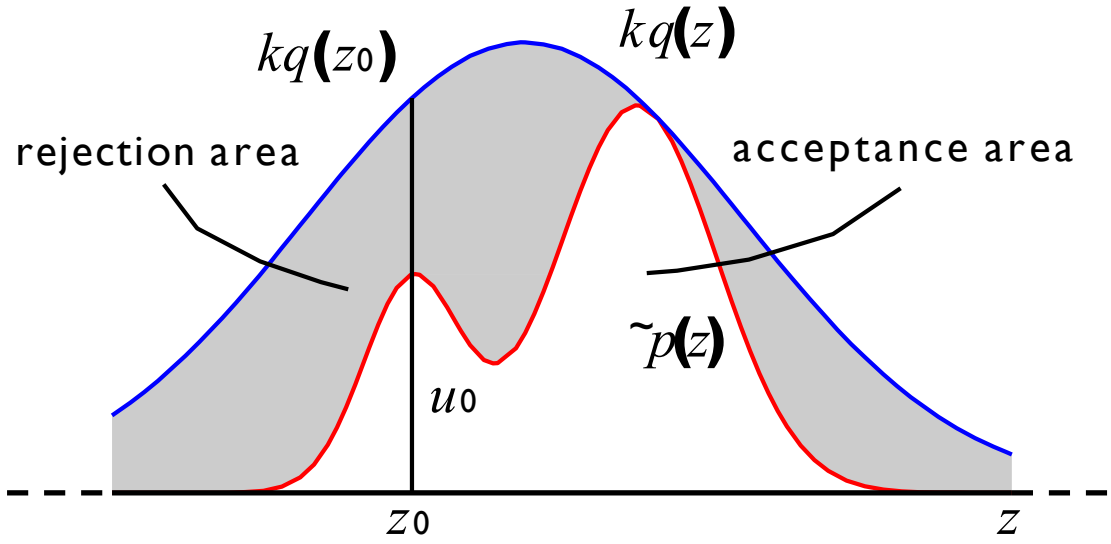
Properties



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 $p(z)$

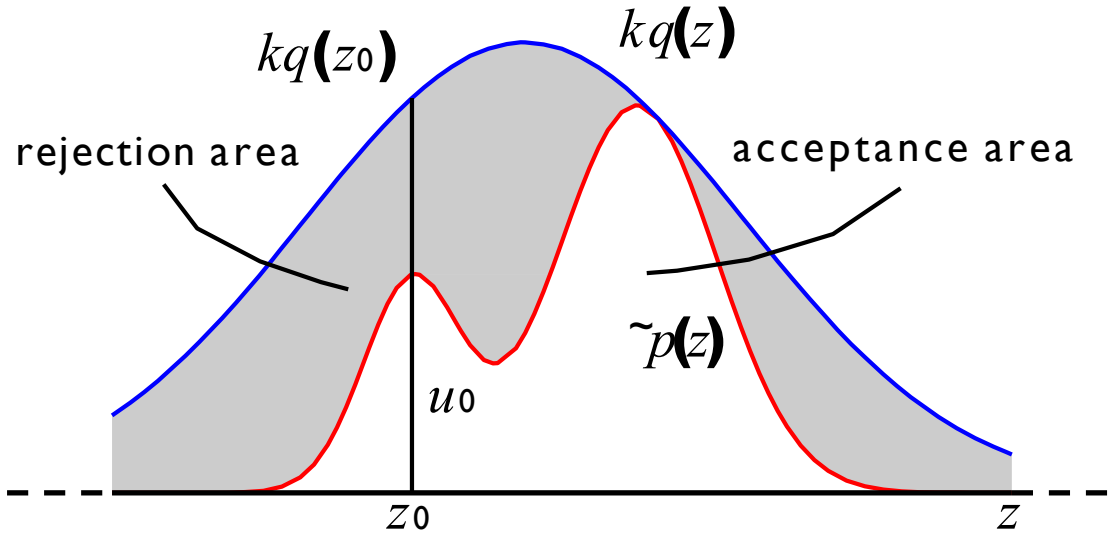
Properties



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 $p(z)$

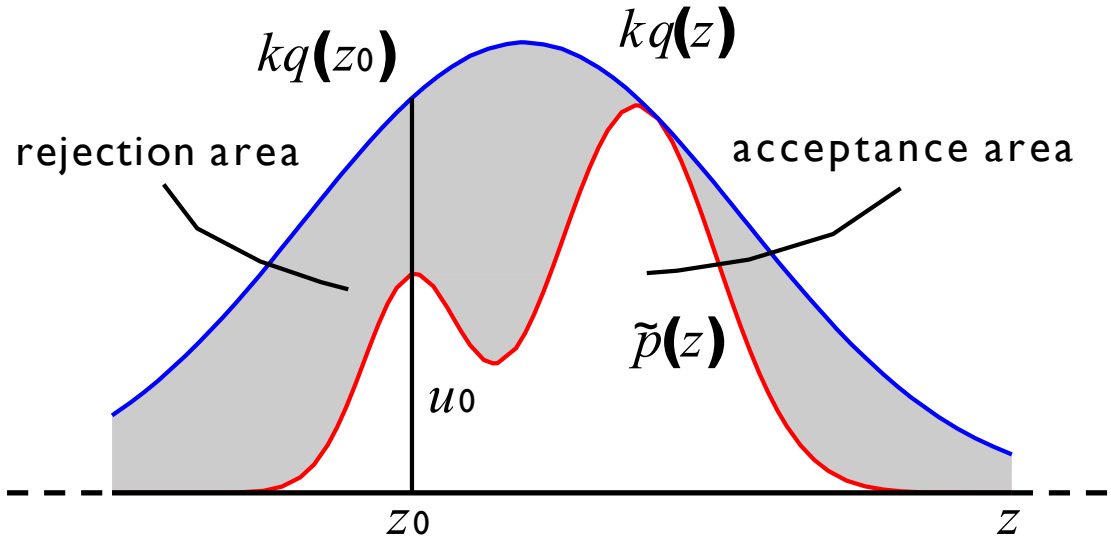
Shortcomings



Adapted from PRML (Bishop, 2006)

§ Finding the upper bound k is tricky

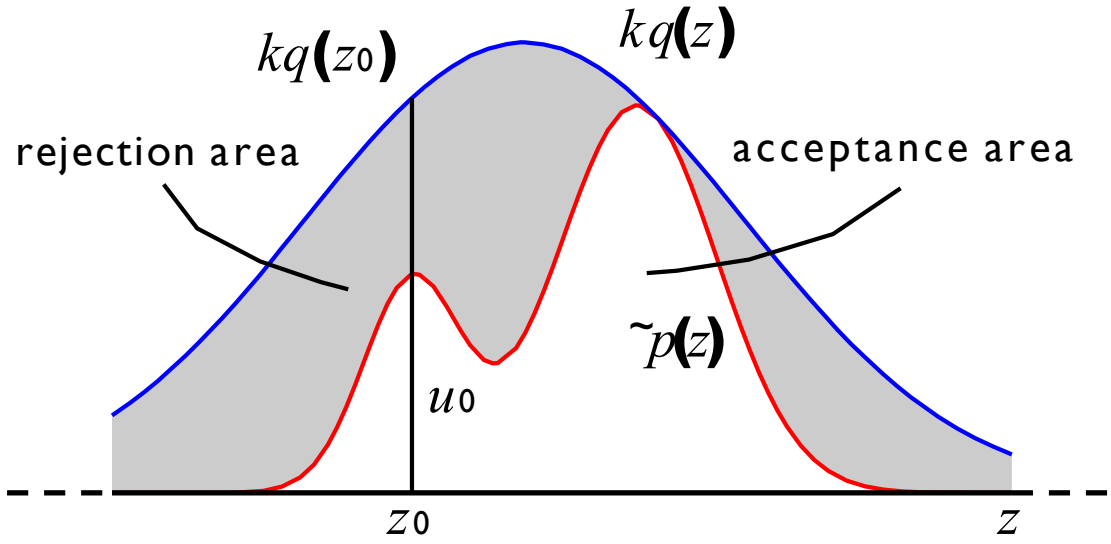
Shortcomings



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):

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

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):

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

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):

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

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):

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

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

$$= \mathbb{E}_q[f(x)\frac{p(x)}{q(x)}]$$

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):

$$\mathbb{E}_p[f(x)] = \int f(x)p(x)dx$$
$$= \int f(x)p(x)\frac{q(x)}{q(x)}dx = \int f(x)\frac{p(x)}{q(x)}q(x)dx$$
$$= \mathbb{E}_q\left[f(x)\frac{p(x)}{q(x)}\right]$$

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

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

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):

$$\mathbb{E}_p[f(x)] = \int f(x)p(x)\,dx = \int f(x)\frac{p(x)}{q(x)}q(x)\,dx = \mathbb{E}_q\left[f(x)\frac{p(x)}{q(x)}\right]$$

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

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

Properties

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

Properties

- Unbiased if $q(z) > 0$ where $p(z) > 0$ 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 $q(z) > 0$ where $p(z) > 0$ 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 $q(z) > 0$ where $p(z) > 0$ 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

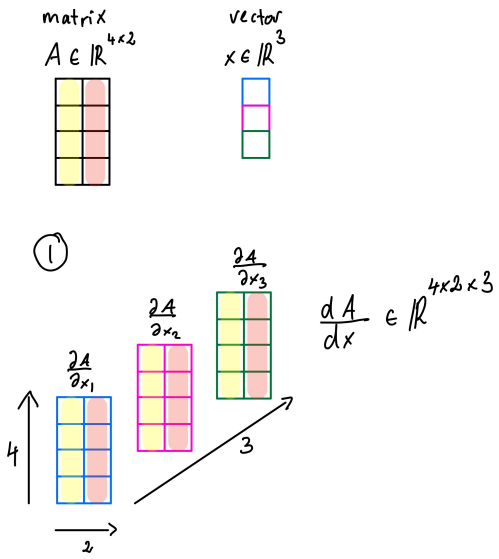
- Unbiased if $q(\theta) > 0$ where $p(\theta) > 0$ and if we can evaluate p
- Breaks down if we do not have enough samples (puts nearly all weight on a single sample)
  - Degeneracy, see also Particle Filtering and SMC (Thrun et al., 2005; Doucet et al., 2000)
- Many draws from proposal density q required, especially in high dimensions
- Requires to be able to evaluate true p. Generalization exists for $p(\theta) = \frac{1}{Z}\tilde{p}(\theta)$. This generalization is biased (but consistent).
- Does not scale to interesting (high-dimensional) problems

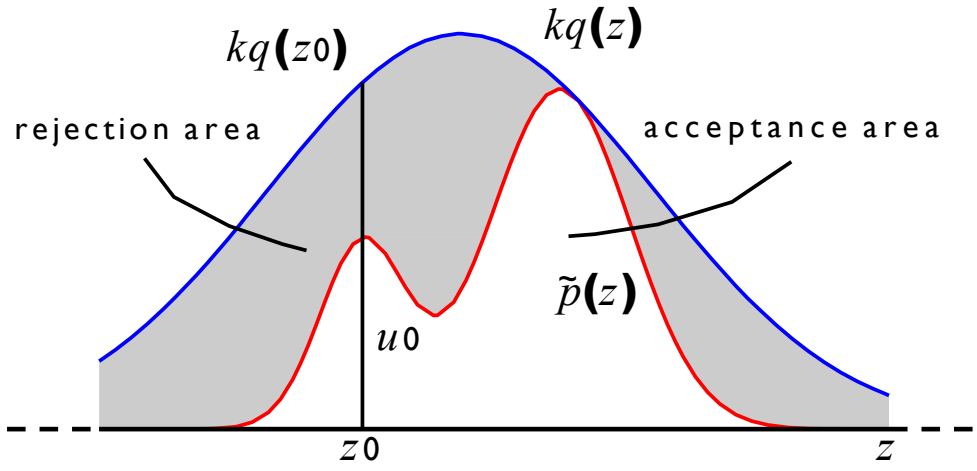
Properties

- Unbiased if $q(z) > 0$ where $p(z) > 0$ 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)

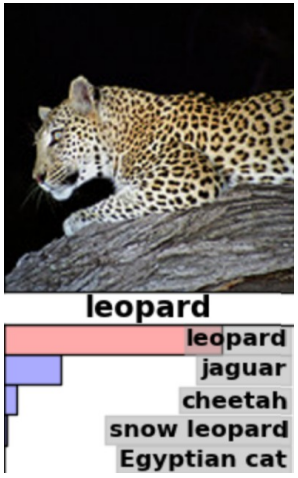
Summary

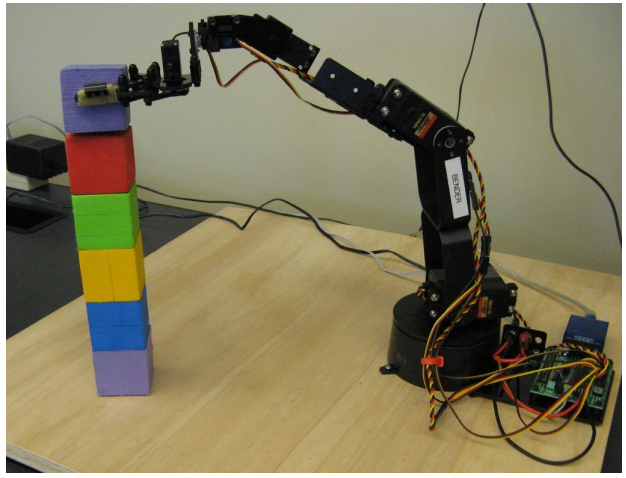


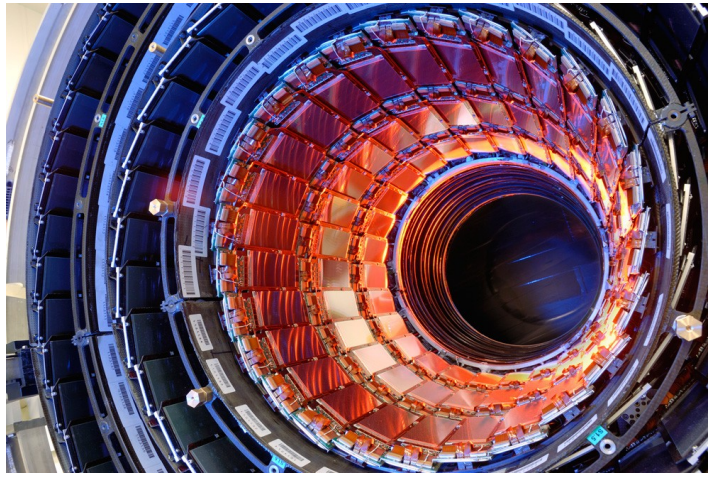


- Two mathematical challenges in machine learning
  - **Differentiation** for optimizing parameters of machine learning models
    - ⏩ Vector calculus and chain rule
  - **Integration** for computing statistics (e.g., means, variances) and as a principled way to address overfitting issue
    - ⏩ Monte-Carlo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