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

Introduction

Differentiation

Integration
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 \( A, b \)
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 \( A, b \)
- 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., $AB \neq BA$
Matrix multiplication is not commutative, i.e., $AB \neq BA$

When multiplying matrices, the “neighboring” dimensions have to fit:

\[
A^{n \times k} B^{k \times m} = C^{n \times m}
\]
Matrix multiplication is not commutative, i.e., $AB \neq BA$

When multiplying matrices, the “neighboring” dimensions have to fit:

\[
\begin{align*}
A_{n \times k} & \quad B_{k \times m} & \quad C_{n \times m} \\
\end{align*}
\]

\[
\begin{align*}
y & = Ax \\
y_i & = \sum_j A_{ij} x_j \\
C & = AB \\
C_{ij} & = \sum_k A_{ik} B_{kj}
\end{align*}
\]

\[
\begin{align*}
y & = \text{A.dot}(x) \\
y & = \text{np.einsum}(\text{'}ij, j\text{'}, \text{A}, x) \\
C & = \text{A.dot}(B) \\
C & = \text{np.einsum}(\text{'}ik, kj\text{'}, \text{A}, \text{B})
\end{align*}
\]
Curve Fitting (Regression) in Machine Learning (1)

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

Introduction

Differentiation

Integration
Differentiation: Outline

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

- 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

\[ f(x) = x^n \quad f'(x) = nx^{n-1} \]
\[ f(x) = \sin(x) \quad f'(x) = \cos(x) \]
\[ f(x) = \tanh(x) \quad f'(x) = 1 - \tanh^2(x) \]
\[ f(x) = \exp(x) \quad f'(x) = \exp(x) \]
\[ f(x) = \log(x) \quad f'(x) = \frac{1}{x} \]
Rules

- Sum Rule

\[(f(x) + g(x))' = f'(x) + g'(x) = \frac{df}{dx} + \frac{dg}{dx}\]
Rules

- Sum Rule

\[ (f(x) + g(x))' = f'(x) + g'(x) = \frac{df}{dx} + \frac{dg}{dx} \]

- Product Rule

\[ (f(x)g(x))' = f'(x)g(x) + f(x)g'(x) = \frac{df}{dx}g(x) + f(x)\frac{dg}{dx} \]
Rules

- **Sum Rule**

\[
(f(x) + g(x))' = f'(x) + g'(x) = \frac{df}{dx} + \frac{dg}{dx}
\]

- **Product Rule**

\[
(f(x)g(x))' = f'(x)g(x) + f(x)g'(x) = \frac{df}{dx}g(x) + f(x)\frac{dg}{dx}
\]

- **Chain Rule**

\[
(g \circ f)'(x) = (g(f(x)))' = g'(f(x))f'(x) = \frac{dg}{df}\frac{df}{dx}
\]
Example: Chain Rule

\[(g \circ f)'(x) = (g(f(x)))' = g'(f(x))f'(x) = \frac{dg}{df} \frac{df}{dx}\]

\[g(z) = \tanh(z)\]
\[z = f(x) = x^n\]

\[(g \circ f)'(x) = \]
Example: Chain Rule

\[(g \circ f)'(x) = (g(f(x)))' = g'(f(x))f'(x) = \frac{dg}{df} \frac{df}{dx}\]

\[g(z) = \tanh(z)\]
\[z = f(x) = x^n\]

\[(g \circ f)'(x) = \left(1 - \tanh^2(x^n)\right) nx^{n-1}\]
\[
\underbrace{\frac{dg}{df}}_{\text{dg/df}} \underbrace{\frac{df}{dx}}_{\text{df/dx}}
\]
\[ f : \mathbb{R}^N \rightarrow \mathbb{R} \]

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

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

\[
\frac{\partial f}{\partial x_i} = \lim_{h \to 0} \frac{f(x_1, \ldots, x_{i-1}, x_i + h, x_{i+1}, \ldots, x_N) - f(x)}{h}
\]
\[ f : \mathbb{R}^N \to \mathbb{R} \]

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

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

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

- **Jacobian vector** (gradient) collects all partial derivatives:

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

Note: This is a row vector.
Example

\[ f : \mathbb{R}^2 \rightarrow \mathbb{R} \]
\[ f(x_1, x_2) = x_1^2x_2 + x_1x_2^3 \in \mathbb{R} \]
Example

\[ f : \mathbb{R}^2 \rightarrow \mathbb{R} \]
\[ f(x_1, x_2) = x_1^2 x_2 + x_1 x_2^3 \in \mathbb{R} \]

- 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

\[ f : \mathbb{R}^2 \rightarrow \mathbb{R} \]
\[ f(x_1, x_2) = x_1^2 x_2 + x_1 x_2^3 \in \mathbb{R} \]

- 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:
  \[
  \frac{df}{dx} = \left[ \frac{\partial f(x_1, x_2)}{\partial x_1}, \frac{\partial f(x_1, x_2)}{\partial x_2} \right] = \left[ 2x_1 x_2 + x_2^3, x_1^2 + 3x_1 x_2^2 \right] \in \mathbb{R}^{1 \times 2}.
  \]
Rules

- **Sum Rule**
  
  \[
  \frac{\partial}{\partial x} (f(x) + g(x)) = \frac{\partial f}{\partial x} + \frac{\partial g}{\partial x}
  \]

- **Product Rule**
  
  \[
  \frac{\partial}{\partial x} (f(x)g(x)) = \frac{\partial f}{\partial x}g(x) + f(x)\frac{\partial g}{\partial x}
  \]

- **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 \( dL/dx \)?
  - \( dL/dx \in \mathbb{R}^{1 \times N} \)

\[
\begin{align*}
\frac{dL}{dx} &= \frac{dL}{de} \frac{de}{dx} \\
\frac{dL}{de} &= e^\top \in \mathbb{R}^{1 \times M} \\
\frac{de}{dx} &= -A \in \mathbb{R}^{M \times N}
\end{align*}
\]

\[ \Rightarrow \frac{dL}{dx} = e^\top (-A) = -(y - Ax)^\top A \in \mathbb{R}^{1 \times N} \]
\[ f : \mathbb{R}^N \rightarrow \mathbb{R}^M \]

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

\[
\begin{bmatrix}
y_1 \\
\vdots \\
y_M
\end{bmatrix}
= \begin{bmatrix}
f_1(x) \\
\vdots \\
f_M(x)
\end{bmatrix}
= \begin{bmatrix}
f_1(x_1, \ldots, x_N) \\
\vdots \\
f_M(x_1, \ldots, x_N)
\end{bmatrix}
\]
\[ f : \mathbb{R}^N \rightarrow \mathbb{R}^M \]

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

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

- **Jacobian matrix** (collection of all partial derivatives)

\[
\begin{bmatrix}
\frac{dy_1}{dx} \\
\vdots \\
\frac{dy_M}{dx}
\end{bmatrix}
= 
\begin{bmatrix}
\frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_N} \\
\vdots & \ddots & \vdots \\
\frac{\partial f_M}{\partial x_1} & \cdots & \frac{\partial f_M}{\partial x_N}
\end{bmatrix}
\in \mathbb{R}^{M \times N}
\]
Example

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

- Compute the gradient \( df/dx \)
  - Dimension of \( df/dx \):
Example

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

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

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

- 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:

\[
fi = \sum_{j=1}^{N} A_{ij}x_j \quad \Rightarrow \quad \frac{\partial f_i}{\partial x_j} = A_{ij}
\]

(3)
Example

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

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

\[
\begin{align*}
  f_i &= \sum_{j=1}^{N} A_{ij} x_j \quad \Rightarrow \quad \frac{\partial f_i}{\partial x_j} = A_{ij} \\
  \Rightarrow \quad \frac{df}{dx} &= \begin{bmatrix}
  \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_N} \\
  \vdots & \ddots & \vdots \\
  \frac{\partial f_M}{\partial x_1} & \cdots & \frac{\partial f_M}{\partial x_N}
  \end{bmatrix} = \begin{bmatrix}
  A_{11} & \cdots & A_{1N} \\
  \vdots & \ddots & \vdots \\
  A_{M1} & \cdots & A_{MN}
  \end{bmatrix} = A \quad (3)
\end{align*}
\]
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

- Consider \( f : \mathbb{R}^2 \to \mathbb{R}, \quad x : \mathbb{R} \to \mathbb{R}^2 \)

\[
f(x) = f(x_1, x_2) = x_1^2 + 2x_2, \\
x(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} = \begin{bmatrix} \sin(t) \\ \cos(t) \end{bmatrix}
\]
Example

- Consider \( f : \mathbb{R}^2 \to \mathbb{R}, \quad x : \mathbb{R} \to \mathbb{R}^2 \)

\[
f(x) = f(x_1, x_2) = x_1^2 + 2x_2, \]

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

- The dimensions \( df/dx \) and \( dx/dt \) are
Example

- Consider \( f : \mathbb{R}^2 \to \mathbb{R} \), \( x : \mathbb{R} \to \mathbb{R}^2 \)

\[
f(x) = f(x_1, x_2) = x_1^2 + 2x_2,
\]

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

- The dimensions \( df/dx \) and \( dx/dt \) are \( 1 \times 2 \) and \( 2 \times 1 \), respectively
- Compute the gradient \( df/dt \) using the chain rule.
Example

- Consider \( f : \mathbb{R}^2 \to \mathbb{R}, \quad x : \mathbb{R} \to \mathbb{R}^2 \)

\[
f(x) = f(x_1, x_2) = x_1^2 + 2x_2,
\]

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

- The dimensions \( df/dx \) and \( dx/dt \) are \( 1 \times 2 \) and \( 2 \times 1 \), respectively.

- Compute the gradient \( df/dt \) using the chain rule.

\[
\begin{aligned}
\frac{df}{dt} &= \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} \end{bmatrix} \begin{bmatrix} \frac{\partial x_1}{\partial t} \\ \frac{\partial x_2}{\partial t} \end{bmatrix} \\
&= \begin{bmatrix} 2 \sin t & 2 \end{bmatrix} \begin{bmatrix} \cos t \\ -\sin t \end{bmatrix} \\
&= 2 \sin t \cos t - 2 \sin t = 2 \sin t (\cos t - 1)
\end{aligned}
\]
BREAK
Derivatives with Matrices

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

  \[
  \text{# target dimensions} \times \text{# parameters}
  \]

  with

  \[
  \frac{df}{dx} \in \mathbb{R}^{E \times D}, \quad df[e,d] = \frac{\partial f_e}{\partial x_d}
  \]
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 × # parameters

with

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

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

- Re-cap: Gradient of a function $f : \mathbb{R}^D \rightarrow \mathbb{R}^E$ is an $E \times D$-matrix:
  
  \[
  \text{# target dimensions} \times \text{# parameters}
  \]

  with
  
  \[
  \frac{df}{dx} \in \mathbb{R}^{E \times D}, \quad df[e, d] = \frac{\partial f_e}{\partial x_d}
  \]

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

- Assume $f : \mathbb{R}^{M \times N} \rightarrow \mathbb{R}^{P \times Q}$, 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)

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

\[ f_i = \sum_{j=1}^{N} A_{ij} x_j , \quad i = 1, \ldots, M \]
Derivatives with Matrices: Example (2)

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

\[ \frac{\partial f_i}{\partial A_{iq}} = x_q \]
Derivatives with Matrices: Example (2)

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

\[ \frac{\partial f_i}{\partial A_{iq}} = x_q \Rightarrow \frac{\partial f_i}{\partial A_{ij}} = x^\top \in \mathbb{R}^{1 \times 1 \times N} \]
Derivatives with Matrices: Example (2)

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

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

\[ \frac{\partial f_i}{\partial A_{k \neq i:}} = 0^\top \in \mathbb{R}^{1 \times 1 \times N} \]
Derivatives with Matrices: Example (2)

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

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

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

\[ \frac{\partial f_i}{\partial A} = \begin{bmatrix} 0^\top \\ \vdots \\ x^\top \\ 0^\top \\ \vdots \\ 0^\top \end{bmatrix} \in \mathbb{R}^{1 \times (M \times N)} \] (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 $dA(x)/dx \in \mathbb{R}^{4 \times 2 \times 3}$ in two equivalent ways:
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 $dA(x)/dx \in \mathbb{R}^{4 \times 2 \times 3}$ in two equivalent ways:
Gradients of a Single-Layer Neural Network (1)

\[ f = \tanh(AX + b) \in \mathbb{R}^M, \quad x \in \mathbb{R}^N, A \in \mathbb{R}^{M \times N}, b \in \mathbb{R}^M \]

\[ f = \tanh(AX + b) \in \mathbb{R}^M, \quad x \in \mathbb{R}^N, A \in \mathbb{R}^{M \times N}, b \in \mathbb{R}^M \]
Gradients of a Single-Layer Neural Network (1)

\[ f = \tanh(Ax + b) \in \mathbb{R}^M, \quad x \in \mathbb{R}^N, A \in \mathbb{R}^{M \times N}, b \in \mathbb{R}^M \]

\[ \frac{\partial f}{\partial b} = \frac{\partial f}{\partial z} \frac{\partial z}{\partial b} \in \mathbb{R}^{M \times M} \]
Gradients of a Single-Layer Neural Network (1)

\[ f = \tanh(Ax + b) \in \mathbb{R}^M, \quad x \in \mathbb{R}^N, A \in \mathbb{R}^{M \times N}, b \in \mathbb{R}^M \]

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

\[ \frac{\partial f}{\partial A} = \underbrace{\frac{\partial f}{\partial z}}_{M \times M} \underbrace{\frac{\partial z}{\partial A}}_{M \times (M \times N)} \in \mathbb{R}^{M \times (M \times N)} \]
Gradients of a Single-Layer Neural Network (2)

\[
\begin{align*}
  f &= \tanh(Ax + b) \in \mathbb{R}^M, \quad x \in \mathbb{R}^N, A \in \mathbb{R}^{M \times N}, b \in \mathbb{R}^M \\
  &=: z \in \mathbb{R}^M \\
  \frac{\partial f}{\partial b} &= \frac{\partial f}{\partial z} \frac{\partial z}{\partial b} \in \mathbb{R}^{M \times M}
\end{align*}
\]
Gradients of a Single-Layer Neural Network (2)

\[ f = \tanh(Ax + b) \in \mathbb{R}^M, \quad x \in \mathbb{R}^N, A \in \mathbb{R}^{M \times N}, b \in \mathbb{R}^M \]

\[ \frac{\partial f}{\partial b} = \begin{bmatrix} \frac{\partial f}{\partial z} & \frac{\partial z}{\partial b} \end{bmatrix} \in \mathbb{R}^{M \times M} \]

\[ \frac{\partial f}{\partial z} = \text{diag}(1 - \tanh^2(z)) \in \mathbb{R}^{M \times M} \]
Gradients of a Single-Layer Neural Network (2)

\[
f = \tanh(Ax + b) \in \mathbb{R}^M, \quad x \in \mathbb{R}^N, A \in \mathbb{R}^{M \times N}, b \in \mathbb{R}^M
\]

\[
\frac{\partial f}{\partial b} = \begin{bmatrix} \frac{\partial f}{\partial z} & \frac{\partial z}{\partial b} \end{bmatrix} \in \mathbb{R}^{M \times M}
\]

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

\[
\frac{\partial z}{\partial b} = I \in \mathbb{R}^{M \times M}
\]

\[
\frac{\partial f}{\partial b}[i, j] = \sum_{l=1}^{M} \frac{\partial f}{\partial z}[i, l] \frac{\partial z}{\partial b}[l, j]
\]
Gradients of a Single-Layer Neural Network (2)

\[ f = \tanh(Ax + b) \in \mathbb{R}^M, \quad x \in \mathbb{R}^N, A \in \mathbb{R}^{M \times N}, b \in \mathbb{R}^M \]

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

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

\[ \frac{\partial z}{\partial b} = I \in \mathbb{R}^{M \times M} \quad (5) \]

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

\[ \text{dfdb} = \text{np.einsum('il, lj', dfdz, dzdb)} \]
Gradients of a Single-Layer Neural Network (3)

\[ f = \tanh(Ax + b) \in \mathbb{R}^M, \quad x \in \mathbb{R}^N, A \in \mathbb{R}^{M \times N}, b \in \mathbb{R}^M \]

\[
\frac{\partial f}{\partial A} = \begin{bmatrix} \frac{\partial f}{\partial z} \\ \frac{\partial z}{\partial A} \end{bmatrix} \in \mathbb{R}^{M \times (M \times N)}
\]
Gradients of a Single-Layer Neural Network (3)

\[ f = \tanh(Ax + b) \in \mathbb{R}^M, \quad x \in \mathbb{R}^N, A \in \mathbb{R}^{M \times N}, b \in \mathbb{R}^M \]

\[ \frac{\partial f}{\partial A} = \begin{pmatrix} \frac{\partial f}{\partial z} \\ \frac{\partial z}{\partial A} \end{pmatrix} \in \mathbb{R}^{M \times (M \times N)} \]

\[ \frac{\partial f}{\partial z} = \text{diag}(1 - \tanh^2(z)) \in \mathbb{R}^{M \times M} \] (6)
Gradients of a Single-Layer Neural Network (3)

\[ f = \tanh(Ax + b) \in \mathbb{R}^M, \quad x \in \mathbb{R}^N, A \in \mathbb{R}^{M \times N}, b \in \mathbb{R}^M \]

\[
\frac{\partial f}{\partial A} = \underbrace{\frac{\partial f}{\partial z}}_{M \times M} \underbrace{\frac{\partial z}{\partial A}}_{M \times (M \times N)} 
\]

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

\[
\frac{\partial z}{\partial A} \quad \text{See (4)}
\]

\[
\frac{\partial f}{\partial A}[i, j, k] = \sum_{l=1}^{M} \frac{\partial f}{\partial z}[i, l] \frac{\partial z}{\partial A}[l, j, k]
\]
Gradients of a Single-Layer Neural Network (3)

\[ f = \tanh(Ax + b) \in \mathbb{R}^M, \quad x \in \mathbb{R}^N, A \in \mathbb{R}^{M \times N}, b \in \mathbb{R}^M \]

\[ \frac{\partial f}{\partial A} = \begin{pmatrix} \frac{\partial f}{\partial z} \\ \frac{\partial z}{\partial A} \end{pmatrix} \in \mathbb{R}^{M \times (M \times N)} \]

\[ \frac{\partial f}{\partial z} = \text{diag}(1 - \tanh^2(z)) \in \mathbb{R}^{M \times M} \quad \text{(6)} \]

\[ \frac{\partial z}{\partial A} \quad \gg \text{See (4)} \]

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

\[ \text{dfdA} = \text{np.einsum}('il, ljk', \text{dfdz}, \text{dzdA}) \]
Putting Things Together

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

- Inputs $x$, observed outputs $y = f(z, \theta) + \epsilon$, $\epsilon \sim \mathcal{N}(0, \Sigma)$
- Train single-layer neural network with
  
  $$f(z, \theta) = \tanh(z), \quad z = Ax + b, \quad \theta = \{A, b\}$$
Putting Things Together

- Inputs $x$, observed outputs $y = f(z, \theta) + \epsilon, \epsilon \sim \mathcal{N}(0, \Sigma)$
- Train single-layer neural network with
  \[ f(z, \theta) = \tanh(z), \quad z = Ax + b, \quad \theta = \{A, b\} \]
- Find $A, b$, such that the squared loss
  \[ L(\theta) = \frac{1}{2} \|e\|^2, \quad e = y - f(z, \theta) \]
  is minimized
Putting Things Together

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

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

- Find $A, b$, such that the squared loss

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

is minimized

- Partial derivatives:

$$\begin{align*}
\frac{\partial L}{\partial A} &= \frac{\partial L}{\partial e} \frac{\partial e}{\partial f} \frac{\partial f}{\partial z} \frac{\partial z}{\partial A} \\
\frac{\partial L}{\partial b} &= \frac{\partial L}{\partial e} \frac{\partial e}{\partial f} \frac{\partial f}{\partial z} \frac{\partial z}{\partial b}
\end{align*}$$

$$\begin{align*}
\frac{\partial L}{\partial e} &\quad \quad (1) \\
\frac{\partial e}{\partial f} &\quad \quad (2), (3) \\
\frac{\partial f}{\partial z} &\quad \quad (6)
\end{align*}$$

$$\begin{align*}
\frac{\partial e}{\partial z} &\quad \quad (4) \\
\frac{\partial f}{\partial z} &\quad \quad (5)
\end{align*}$$
Inputs $x$, observed outputs $y$ 

Train multi-layer neural network with

$$
\begin{align*}
  f_0 &= x \\
  f_i &= \sigma_i(A_{i-1} f_{i-1} + b_{i-1}), \quad i = 1, \ldots, L
\end{align*}
$$
Gradients of a Multi-Layer Neural Network

- Inputs $x$, observed outputs $y$
- Train multi-layer neural network with

\[ f_0 = x \]
\[ f_i = \sigma_i(A_{i-1} f_{i-1} + b_{i-1}), \quad i = 1, \ldots, L \]

- Find $A_j, b_j$ for $j = 0, \ldots, L - 1$, such that the squared loss

\[ L(\theta) = \|y - f_L(\theta, x)\|^2 \]

is minimized, where $\theta = \{A_j, b_j\}, \quad j = 0, \ldots, L - 1$
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

\[
\frac{\partial L}{\partial \theta_{L-1}} = \frac{\partial L}{\partial f_L} \frac{\partial f_L}{\partial \theta_{L-1}}
\]

\[
\frac{\partial L}{\partial \theta_{L-2}} = \frac{\partial L}{\partial f_L} \frac{\partial f_L}{\partial f_{L-1}} \frac{\partial f_{L-1}}{\partial \theta_{L-2}}
\]
Gradients of a Multi-Layer Neural Network

\[
\frac{\partial L}{\partial \theta_{L-1}} = \frac{\partial L}{\partial f_L} \frac{\partial f_L}{\partial \theta_{L-1}}
\]

\[
\frac{\partial L}{\partial \theta_{L-2}} = \frac{\partial L}{\partial f_L} \frac{\partial f_L}{\partial f_{L-1}} \frac{\partial f_{L-1}}{\partial \theta_{L-2}}
\]

\[
\frac{\partial L}{\partial \theta_{L-3}} = \frac{\partial L}{\partial f_L} \frac{\partial f_L}{\partial f_{L-1}} \frac{\partial f_{L-1}}{\partial f_{L-2}} \frac{\partial f_{L-2}}{\partial \theta_{L-3}}
\]
Gradients of a Multi-Layer Neural Network

\[
\frac{\partial L}{\partial \theta_{L-1}} = \frac{\partial L}{\partial f_L} \frac{\partial f_L}{\partial \theta_{L-1}}
\]

\[
\frac{\partial L}{\partial \theta_{L-2}} = \frac{\partial L}{\partial f_L} \frac{\partial f_L}{\partial f_{L-1}} \frac{\partial f_{L-1}}{\partial \theta_{L-2}}
\]

\[
\frac{\partial L}{\partial \theta_{L-3}} = \frac{\partial L}{\partial f_L} \frac{\partial f_L}{\partial f_{L-1}} \frac{\partial f_{L-1}}{\partial f_{L-2}} \frac{\partial f_{L-2}}{\partial \theta_{L-3}}
\]

\[
\frac{\partial L}{\partial \theta_i} = \frac{\partial L}{\partial f_L} \frac{\partial f_L}{\partial f_{L-1}} \ldots \frac{\partial f_{i+2}}{\partial f_{i+1}} \frac{\partial f_{i+1}}{\partial \theta_i}
\]
Gradients of a Multi-Layer Neural Network

\[
\frac{\partial L}{\partial \theta_{L-1}} = \frac{\partial L}{\partial f_L} \frac{\partial f_L}{\partial \theta_{L-1}}
\]

\[
\frac{\partial L}{\partial \theta_{L-2}} = \frac{\partial L}{\partial f_L} \frac{\partial f_L}{\partial f_{L-1}} \frac{\partial f_{L-1}}{\partial \theta_{L-2}}
\]

\[
\frac{\partial L}{\partial \theta_{L-3}} = \frac{\partial L}{\partial f_L} \frac{\partial f_L}{\partial f_{L-1}} \frac{\partial f_{L-1}}{\partial f_{L-2}} \frac{\partial f_{L-2}}{\partial \theta_{L-3}}
\]

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

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 $y = NN(x, \theta) + \epsilon$, $\epsilon \sim \mathcal{N}(0, I)$ then the log-likelihood is
    $$\log p(y|X, \theta) = -\frac{1}{2} \|y - NN(x, \theta)\|^2$$
Training Neural Networks as Maximum Likelihood Estimation

- Training a neural network in the above way corresponds to maximum likelihood estimation:
  - If $y = NN(x, \theta) + \epsilon$, $\epsilon \sim \mathcal{N}(0, I)$ then the log-likelihood is
    \[
    \log p(y|X, \theta) = -\frac{1}{2}||y - NN(x, \theta)||^2
    \]
  - Find $\theta^*$ 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 $y = NN(x, \theta) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, I)$ then the log-likelihood is
    $$\log p(y|X, \theta) = -\frac{1}{2} \|y - NN(x, \theta)\|^2$$

- Find $\theta^*$ 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 $M$:

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

$$f(x, \theta) = \theta_0 + \theta_1 x + \theta_2 x^2 + \cdots + \theta_M x^M = \sum_{i=0}^{M} \theta_i x^i$$
Example: Linear Regression (1)

- Linear regression with a polynomial of order $M$:

$$y = f(x, \theta) + \epsilon, \quad \epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$$
$$f(x, \theta) = \theta_0 + \theta_1 x + \theta_2 x^2 + \cdots + \theta_M x^M = \sum_{i=0}^{M} \theta_i x^i$$

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

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

Polynomial of degree 16

Data

Maximum likelihood estimate

Regularization, model selection etc. can address overfitting

Tutorials later this week

Mathematics for Machine Learning
Marc Deisenroth
@Deep Learning Indaba, September 10, 2017
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 $\theta^*$, we can average over all plausible parameters
  - Bayesian integration:
  $$p(y|x) = \int p(y|x, \theta) p(\theta) d\theta$$
Bayesian Integration to Avoid Overfitting

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

  ▶ Bayesian integration:

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

- More details on what $p(\theta)$ is ▶ Tutorials later this week
Bayesian Integration to Avoid Overfitting

- Instead of fitting a single set of parameters \( \theta^* \), we can average over all plausible parameters
  - Bayesian integration:
    \[
p(y|x) = \int p(y|x, \theta) p(\theta) d\theta
    \]

- More details on what \( p(\theta) \) 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:

\[
\mathbb{E}_x[x] = \int x p(x) dx =: \mu_x \\
\mathbb{V}_x[x] = \int (x - \mu_x)(x - \mu_x)^T dx \\
\text{Cov}[x, y] = \iint (x - \mu_x)(y - \mu_y)^T dx dy
\]
Computing Statistics of Random Variables

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

\[
\mathbb{E}_x[x] = \int xp(x)\,dx =: \mu_x
\]

\[
\nabla_x[x] = \int (x - \mu_x)(x - \mu_x)^\top \,dx
\]

\[
\text{Cov}[x, y] = \iint (x - \mu_x)(y - \mu_y)^\top \,dxdy
\]

- These integrals can often not be computed in closed form

\[\Box\quad \text{Approximations}\]
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:

\[
E[f(x)] = \int f(x) p(x) dx
\]

- Example: Means/variances of distributions, predictions
- **Complication:** Integral cannot be evaluated analytically
Problem 2: Monte Carlo Estimation

- **Computing expectations** via statistical sampling:

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

\[
p(y|x) = \int p(y|\theta, x) \underbrace{p(\theta)}_{\text{Parameter distribution}} d\theta
\]

\[
\approx \frac{1}{S} \sum_{s=1}^{S} p(y|\theta^{(s)}, x), \quad \theta^{(s)} \sim p(\theta)
\]
Problem 2: Monte Carlo Estimation

- **Computing expectations** via statistical sampling:

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

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

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

- **Key problem:** Generating samples from \(p(x)\) or \(p(\theta)\)

 ▷ Need to solve **Problem 1**
Sampling Discrete Values

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

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 $z_0$

Adapted from PRML (Bishop, 2006)
Properties

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

Adapted from PRML (Bishop, 2006)
Properties

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

Adapted from PRML (Bishop, 2006)
Properties

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

Adapted from PRML (Bishop, 2006)
Shortcomings

- Finding the upper bound $k$ is tricky
Shortcomings

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

Adapted from PRML (Bishop, 2006)
Shortcomings

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

Adapted from PRML (Bishop, 2006)
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*):

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

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

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

\[
E_p[f(x)] = \int f(x)p(x)dx
\]

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

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

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):

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

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

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


References II


