Math Preliminaries & MLE

Deep Learning KU (DAT.C302UF)

Simon Hitzginger

Oct 22, 2025

Institute of Machine Learning and Neural Computing
Graz University of Technology, Austria

Goals of this Course

- Strengthen knowledge about mathematical foundations of deep learning
- Become familiar with `PyTorch`
- Acquire a solid understanding of modern deep learning models
  - Architectures, Training Paradigms, Model Selection, Evaluation
- Being able to apply deep learning to practical problems

Math Preliminaries

Mathematics for Deep Learning

Linear Algebra

- **Extremely important** from a computational perspective
- We'll use `PyTorch` to write "Linear Algebra Programs"
  - Core object: **Tensors** (same in Tensorflow and JAX)

Mathematics for Deep Learning

Linear Algebra

- **Extremely important** from a computational perspective
- We'll use `PyTorch` to write "Linear Algebra Programs"
  - Core object: **Tensors** (same in Tensorflow and JAX)

Calculus

- Minimizing continuous, differentiable **loss functions**
- Will often be **high-dimensional** $\rightarrow$ Multivariate Calculus (Matrix Calculus)

Mathematics for Deep Learning

Linear Algebra

- **Extremely important** from a computational perspective
- We'll use `PyTorch` to write "Linear Algebra Programs"
  - Core object: **Tensors** (same in Tensorflow and JAX)

Calculus

- Minimizing continuous, differentiable **loss functions**
- Will often be **high-dimensional** $\rightarrow$ Multivariate Calculus (Matrix Calculus)

Probability Theory

- "**Optimal Reasoning under Uncertainty**"
  - Noisy data, uncertainty in model parameters etc.
- Generative Modeling, Statistical Learning Theory, ...

Linear Algebra

Dot Product

- Given vectors

$$\mathbf{x} = (x_1, x_2, \ldots, x_N)^T \in \mathbb{R}^N, \qquad \mathbf{y} = (y_1, y_2, \ldots, y_N)^T \in \mathbb{R}^N$$

- We define the **dot product** as

$$\mathbf{x}^T\mathbf{y} = \mathbf{y}^T\mathbf{x} = \sum_{i=1}^{N} x_i y_i$$

- PyTorch: `x.T @ y`

Norms

Important Norms

For $\mathbf{x} = (x_1, \ldots, x_N)^T \in \mathbb{R}^N$, we define

- The ℓ_2 norm
  $\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^N x_i^2} = \sqrt{\mathbf{x}^T\mathbf{x}}$.
  - Thus, $\|\mathbf{x}\|_2^2 = \sum_{i=1}^N x_i^2 = \mathbf{x}^T\mathbf{x}$
- The ℓ_1 norm $\|\mathbf{x}\|_1 = \sum_{i=1}^N |x_i|$

Norms

Important Norms

For $\mathbf{x} = (x_1, \ldots, x_N)^T \in \mathbb{R}^N$, we define

- The ℓ_2 norm
  $\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^N x_i^2} = \sqrt{\mathbf{x}^T\mathbf{x}}$.
  - Thus, $\|\mathbf{x}\|_2^2 = \sum_{i=1}^N x_i^2 = \mathbf{x}^T\mathbf{x}$
- The ℓ_1 norm $\|\mathbf{x}\|_1 = \sum_{i=1}^N |x_i|$

PyTorch

- `torch.linalg.norm(x, ord=p)`
- $p = 1$ or $p = 2$ in this case

Norms

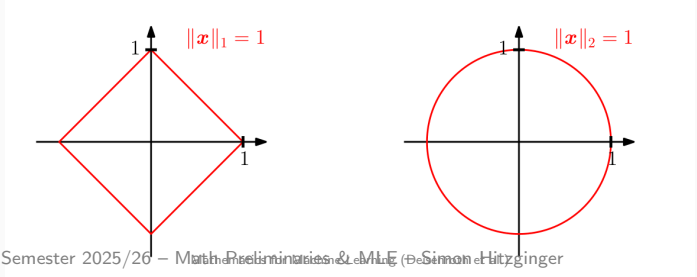
Important Norms

For $\mathbf{x} = (x_1, \ldots, x_N)^T \in \mathbb{R}^N$, we define

- The ℓ_2 norm
  $\|\mathbf{x}\|_2 = \sqrt{\sum_{i=1}^N x_i^2} = \sqrt{\mathbf{x}^T\mathbf{x}}$.
  - Thus, $\|\mathbf{x}\|_2^2 = \sum_{i=1}^N x_i^2 = \mathbf{x}^T\mathbf{x}$
- The ℓ_1 norm $\|\mathbf{x}\|_1 = \sum_{i=1}^N |x_i|$

PyTorch

- `torch.linalg.norm(x, ord=p)`
- $p = 1$ or $p = 2$ in this case



Matrix-vector product

- Matrix $A \in \mathbb{R}^{M \times N}$
  - Can be seen as **collection** of N column vectors $\in \mathbb{R}^M$

Matrix-vector product

$$\mathbf{y} = A\mathbf{x}, \qquad \mathbf{x} \in \mathbb{R}^N, \mathbf{y} \in \mathbb{R}^M$$

- PyTorch: `y = A @ x`
- A is a **linear map** that transforms $\mathbf{x} \in \mathbb{R}^N$ into $\mathbf{y} \in \mathbb{R}^M$

Differential Calculus

Derivative of Univariate Function

- Let $f : \mathbb{R} \to \mathbb{R}$ be a **univariate function**
- If it exists, **the derivative of** f w.r.t. x is defined as

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

Derivative of Univariate Function

- Let $f : \mathbb{R} \to \mathbb{R}$ be a **univariate function**
- If it exists, **the derivative of** f w.r.t. x is defined as

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

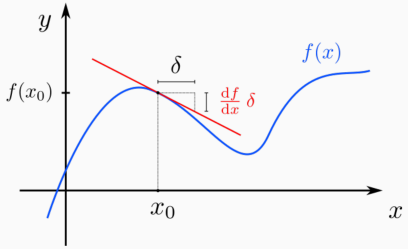
Intuition

- I know $f(x)$ for some x
- I **slightly change the input** (from x to e.g. $x + 0.00001$)
- How will the **output** $f(x + 0.00001)$ change (in proportion to the change in input)?
- i.e., **how sensitive** is f to changes to its inputs (at particular points)?

Best Linear Approximation

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

- **"Instantaneous" rate of change**
- $f'(x)$ is the **slope of the tangent line** going through x
- This is the best **local linear approximation**



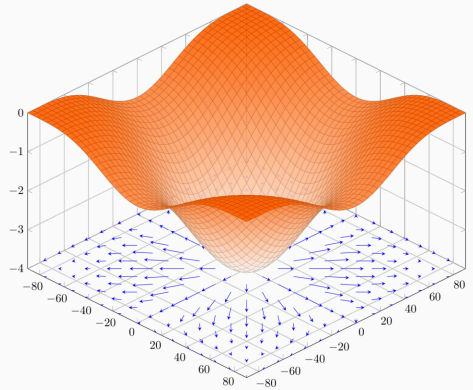
Gradients

- Let $f(x_1, x_2, \ldots, x_D)$ be a **multi-variate scalar-valued** function ($f : \mathbb{R}^D \to \mathbb{R}$)

- The **gradient** of f w.r.t. $\mathbf{x} = (x_1, \ldots, x_D)^T$ at some point $\mathbf{x}$ is defined as

$$\nabla f(\mathbf{x}) = \begin{pmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} \\ \frac{\partial f(\mathbf{x})}{\partial x_2} \\ \vdots \\ \frac{\partial f(\mathbf{x})}{\partial x_D} \end{pmatrix}$$

- $\nabla f(\mathbf{x})$ **points in the direction of steepest ascent**



https://commons.wikimedia.org/wiki/File:3d-gradient-cos.svg

Jacobian

- Let $f(\mathbf{x})$ be a **vector-valued** function ($f : \mathbb{R}^D \to \mathbb{R}^M$), defined as

$$f(\mathbf{x}) = (f_1(\mathbf{x}),\ f_2(\mathbf{x}),\ \ldots,\ f_M(\mathbf{x}))^T$$

  where $f_i(\mathbf{x}) : \mathbb{R}^D \to \mathbb{R}$.

- The **Jacobian matrix** of f at some point $\mathbf{x}$ is

$$J_f(\mathbf{x}) = \begin{pmatrix} \nabla f_1(\mathbf{x})^T \\ \nabla f_2(\mathbf{x})^T \\ \vdots \\ \nabla f_M(\mathbf{x})^T \end{pmatrix} = \begin{pmatrix} \frac{\partial f_1(\mathbf{x})}{\partial x_1} & \frac{\partial f_1(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial f_1(\mathbf{x})}{\partial x_D} \\ \frac{\partial f_2(\mathbf{x})}{\partial x_1} & \frac{\partial f_2(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial f_2(\mathbf{x})}{\partial x_D} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial f_M(\mathbf{x})}{\partial x_1} & \frac{\partial f_M(\mathbf{x})}{\partial x_2} & \cdots & \frac{\partial f_M(\mathbf{x})}{\partial x_D} \end{pmatrix} \in \mathbb{R}^{M \times D}$$

Chain Rule

Chain Rule in 1D

If $f : \mathbb{R} \to \mathbb{R}$ and $g : \mathbb{R} \to \mathbb{R}$, then

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

Chain Rule in arbitrary dimensions

If $g : \mathbb{R}^D \to \mathbb{R}^E$ and $f : \mathbb{R}^E \to \mathbb{R}^F$, then

$$J_{f \circ g}(\mathbf{x}) = J_f(g(\mathbf{x}))\ J_g(\mathbf{x})$$

Chain Rule

Chain Rule in 1D

If $f : \mathbb{R} \to \mathbb{R}$ and $g : \mathbb{R} \to \mathbb{R}$, then

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

Chain Rule in arbitrary dimensions

If $g : \mathbb{R}^D \to \mathbb{R}^E$ and $f : \mathbb{R}^E \to \mathbb{R}^F$, then

$$J_{f \circ g}(\mathbf{x}) = J_f(g(\mathbf{x}))\ J_g(\mathbf{x})$$

Important Special Case (Vector-Jacobian Product)

If $g : \mathbb{R}^D \to \mathbb{R}^E$ and $f : \mathbb{R}^E \to \mathbb{R}$, then $\nabla_{\mathbf{x}}\{f(g(\mathbf{x}))\}^T = \nabla f(g(\mathbf{x}))^T\ J_g(\mathbf{x})$

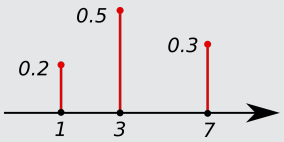
Jacobians in Practice

- Jacobians are **the most general form of** "derivatives"
  - Gradients are (transposed) Jacobians
  - Scalar derivatives are 1×1 Jacobians
- Training deep neural nets needs **gradients of loss function** w.r.t. model parameters
- **Backpropagation = efficiently applying the chain rule**
- **Autodiff (AD)**: PyTorch keeps track of Jacobians in the background
  - Systematic computation of the chain rule, directly on the numerical values
  - Maintains computational graph of function
  - Computes **vector-jacobian products** (reverse mode AD)
  - More on this later in the semester

Probability Theory

Discrete Random Variables

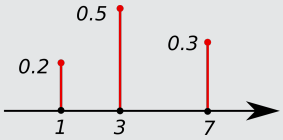
- X is a **discrete random variable** if the set of values $\mathcal{X}$ it can assume is **countable**
  - e.g., $\mathcal{X} = \{1, 3, 7\}$ or $\mathcal{X} = \mathbb{N}$
- **Probability Mass Function**: $p(x) = \mathbb{P}(X = x)$
  - $p : \mathcal{X} \to [0, 1]$
  - Probability of X assuming $x \in \mathcal{X}$



https://commons.wikimedia.org/wiki/File:Discrete_probability_distrib.svg

Discrete Random Variables

- X is a **discrete random variable** if the set of values $\mathcal{X}$ it can assume is **countable**
  - e.g., $\mathcal{X} = \{1, 3, 7\}$ or $\mathcal{X} = \mathbb{N}$
- **Probability Mass Function**: $p(x) = \mathbb{P}(X = x)$
  - $p : \mathcal{X} \to [0, 1]$
  - Probability of X assuming $x \in \mathcal{X}$

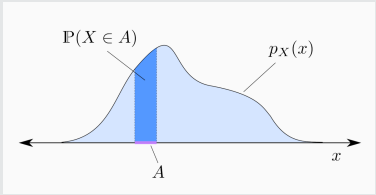


https://commons.wikimedia.org/wiki/File:Discrete_probability_distrib.svg

Continuous Random Variables

- Assume $\mathcal{X} = \mathbb{R}$
- **Probability Density Function** $p : \mathcal{X} \to [0, \infty)$ such that

$$\mathbb{P}(X \in A) = \int_A p(x)dx$$



Marginal Distributions

- Let $p(x, y)$ be a **joint distribution** (**mass function**)
  - Assigns probability mass to all possible configurations $\in \mathcal{X} \times \mathcal{Y}$
- Then, we can obtain the **marginal distribution** as follows:

$$p(x) = \sum_{y \in \mathcal{Y}} p(x, y)$$

- If $p(x, y)$ is a joint **density**, we compute

$$p(x) = \int_{\mathcal{Y}} p(x, y)\, dy$$

Conditional Distributions, Bayes' Law

- Let $p(x, y)$ be a mass function or density
- The **conditionals** $p(x \mid y)$ and $p(y \mid x)$ are given by

$$p(x \mid y) = \frac{p(x, y)}{p(y)}, \quad p(y \mid x) = \frac{p(x, y)}{p(x)}$$

- Which also shows how we can **factorize the joint**:

$$p(x \mid y)p(y) = p(x, y), \quad p(y \mid x)p(x) = p(x, y)$$

- The **relationship between** $p(y \mid x)$ and $p(x \mid y)$ is given by **Bayes' Law**

$$p(y \mid x) = \frac{p(x \mid y)p(y)}{p(x)}$$

Expectations

- Let $p(x)$ be a **probability mass function**, $x \in \mathcal{X}$:

$$\mathbb{E}_{x \sim p(x)}\left[f(x) \right] = \sum_{x \in \mathcal{X}} p(x) f(x)$$

- Let $p(x)$ be a **probability density function**, $x \in \mathcal{X} = \mathbb{R}$:

$$\mathbb{E}_{x \sim p(x)}\left[f(x) \right] = \int_{-\infty}^{\infty} p(x) f(x)\, dx$$

Monte Carlo Simulation

$$\mathbb{E}_{x \sim p(x)}\left[f(x) \right] \approx \frac{1}{N} \sum_{i=1}^{N} f(x_i)$$

where $x_1, \ldots, x_N \overset{\text{i.i.d.}}{\sim} p(x)$.

Expectations

Notation

$$\begin{aligned}\mathbb{E}_{x\sim p(x)}[f(x)] &= \int_{-\infty}^{\infty} p(x)\, f(x)\, dx \\ &= \mathbb{E}_{p(x)}[f(x)] \\ &= \mathbb{E}[f(x)]\end{aligned}$$

Example: For a joint distribution $p(x, y)$, the extended notation clarifies which variable we integrate over:

$$\mathbb{E}_{(\mathbf{x},y)\sim p^*(\mathbf{x},y)}\left[\mathcal{L}(y, f(\mathbf{x}))\right] = \mathbb{E}_{\mathbf{x}\sim p^*(\mathbf{x})}\left[\mathbb{E}_{y\sim p^*(y\,|\,\mathbf{x})}\left[\mathcal{L}(y, f(\mathbf{x}))\right]\right]$$

Expectations

Properties of Expectations

- **Linearity**:

$$\mathbb{E}_{p(x,y)}[aX + bY] = a\mathbb{E}_{p(x)}[X] + b\mathbb{E}_{p(y)}[Y]$$

with $a, b \in \mathbb{R}$.

- **Expectations of constants**:

$$\mathbb{E}[v] = v$$

with $v \in \mathbb{R}$.

Maximum Likelihood Estimation

Unsupervised Learning

- Given a **dataset** $= \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\}$
  - Measurement data $\mathbf{x}^{(i)} \in \mathbb{R}^D$
  - Images $\mathbf{x}^{(i)} \in \mathbb{R}^{H \times W \times C}$
  - Text, molecules, proteins ...
- We assume the **data is drawn i.i.d. from some data generating distribution** $p^*(\mathbf{x})$

Unsupervised Learning

- Given a **dataset** $= \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\}$
  - Measurement data $\mathbf{x}^{(i)} \in \mathbb{R}^D$
  - Images $\mathbf{x}^{(i)} \in \mathbb{R}^{H \times W \times C}$
  - Text, molecules, proteins ...
- We assume the **data is drawn i.i.d. from some data generating distribution** $p^*(\mathbf{x})$

Goal (informal)

Learn model $p_{\boldsymbol{\theta}}(\mathbf{x})$ with parameters $\boldsymbol{\theta}$ such that

$$p_{\boldsymbol{\theta}} \approx p^*$$

Maximum Likelihood

Idea

Find model p_θ such that the **observed data** is as **likely as possible** (under p_θ).

Maximum Likelihood

Idea

Find model p_θ such that the **observed data** is as **likely as possible** (under p_θ).

- The **Maximum Likelihood Estimate** (MLE) is thus given by

$$\theta_{\text{MLE}} = \operatorname*{argmax}_\theta p_\theta(\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)})$$

- Since $\mathbf{x}^{(i)} \overset{\text{i.i.d.}}{\sim} p^*(\mathbf{x})$, we have

$$p_\theta(\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)}) = \prod_{i=1}^{N} p_\theta(\mathbf{x}^{(i)})$$

- Hence,

$$\theta_{\text{MLE}} = \operatorname*{argmax}_\theta \prod_{i=1}^{N} p_\theta(\mathbf{x}^{(i)})$$

Maximum Likelihood (cont.)

- Since $\log(\cdot)$ is a monotonically increasing function:

$$\underset{\boldsymbol{\theta}}{\operatorname{argmax}} \prod_{i=1}^{N} p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \ \log \left(\prod_{i=1}^{N} p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) \right) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_{i=1}^{N} \log \left(p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) \right)$$

- Equivalently, we can minimize the *negative log-likelihood* (NLL):

$$\underset{\boldsymbol{\theta}}{\operatorname{argmax}} \sum_{i=1}^{N} \log \left(p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) \right) = \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \sum_{i=1}^{N} - \log \left(p_{\boldsymbol{\theta}}(\mathbf{x}^{(i)}) \right)$$

Supervised Learning

- In most scenarios, our goal is to learn additional information associated with the data, such as the objects present in an image.
- Consider $= \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(N)}, y^{(N)})\}$, with $\mathbf{x}^{(i)} \in \mathbb{R}^D, y^{(i)} \in \mathcal{C}$
  - e.g. binary classification, $\mathcal{C} = \{0, 1\}$
- **Assumption:** $(\mathbf{x}^{(i)}, y^{(i)}) \overset{\text{i.i.d.}}{\sim} p^*(\mathbf{x}, y)$
- **Discriminative classifier** tries to learn $p_{\boldsymbol{\theta}}(y \mid \mathbf{x})$
  - i.e., a distribution **for each x**

Goal (informal)

Learn model $p_{\boldsymbol{\theta}}(y \mid \mathbf{x})$ with parameters $\boldsymbol{\theta}$ such that

$$p_{\boldsymbol{\theta}}(y \mid \mathbf{x}) \approx p^*(y \mid \mathbf{x})$$

Maximum Likelihood – Supervised Learning

- **Conditional Maximum Likelihood**:

$$\theta_{\text{MLE}} = \underset{\theta}{\text{argmax}} \; p_\theta(y^{(1)}, \ldots, y^{(N)} \mid \mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)})$$

- We do not care about modeling $p^*(\mathbf{x})$

Conditional Likelihood

Notation

$$p_{\text{model}}(y \mid x; \theta) = p_\theta(y \mid x)$$

Both notations describe the **conditional likelihood** of the target y given input x under model parameters θ, informally described as:

$$p_\theta(y \mid x) = \text{“probability of } y \text{ given } x\text{, under parameters } \theta\text{”}.$$

Maximum A Posteriori (MAP) Estimation

Motivation

What if we already have **some prior knowledge** about what the parameters θ should look like?

Idea

We treat θ as a **random variable** and apply Bayes' rule:

$$p(\theta \mid \mathbf{X}) = \frac{p(\mathbf{X} \mid \theta)\, p(\theta)}{p(\mathbf{X})}.$$

The prior $p(\theta)$ encodes our beliefs about likely parameter values, while the likelihood $p(\mathbf{X} \mid \theta)$ comes from the data.

Maximum A Posteriori (MAP) Estimation

Since $p(\mathbf{X})$ does not depend on the parameters θ, we can disregard it when maximizing the posterior.

We therefore choose the parameter value that maximizes the posterior:

$$\theta_{\text{MAP}} = \arg\max_{\theta} \; p_{\text{model}}(\theta \mid \mathbf{X}) = \arg\max_{\theta} \; p(\mathbf{X} \mid \theta)\, p(\theta).$$

Decision Theory

Decision Theory

- Supervised learning: $= \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(N)}, y^{(N)})\}$, with $\mathbf{x}^{(i)} \in \mathbb{R}^D, y^{(i)} \in \mathcal{C}$
- Assume we have the **true posterior** $p^*(y \mid \mathbf{x})$
- We have perfect knowledge about the **uncertainty in the system**
- What if we **have to make a hard decision** about the value of y

Decision Theory

- Supervised learning: $= \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(N)}, y^{(N)})\}$, with $\mathbf{x}^{(i)} \in \mathbb{R}^D, y^{(i)} \in \mathcal{C}$
- Assume we have the **true posterior** $p^*(y \mid \mathbf{x})$
- We have perfect knowledge about the **uncertainty in the system**
- What if we **have to make a hard decision** about the value of y

Unfair Coin Toss Example

- Let $\mathcal{C} = \{0, 1\}$ (heads and tails)
- For a fixed $\mathbf{x}$, assume $p^*(y = 0 \mid \mathbf{x}) = 0.7$, and $p^*(y = 1 \mid \mathbf{x}) = 0.3$

If you want to be **correct as often as possible**,
what would be your prediction $\hat{y}$ before flipping the coin?

- You would **always predict** $\hat{y} = 0$, since $p^*(y = 0 \mid \mathbf{x}) > p^*(y = 1 \mid \mathbf{x})$

Decision Theory (cont.)

- What if we **alter the game**:
- When you predict correctly, you lose nothing
- When you predict $\hat{y} = 0$, but the coin lands on $y = 1$, you lose 100\$
- When you predict $\hat{y} = 1$, but the coin lands on $y = 0$, you lose 1\$

Is it still a good idea to always predict $\hat{y} = 0$?

Decision Theory (cont.)

- What if we **alter the game**:
- When you predict correctly, you lose nothing
- When you predict $\hat{y} = 0$, but the coin lands on $y = 1$, you lose 100$
- When you predict $\hat{y} = 1$, but the coin lands on $y = 0$, you lose 1$

Is it still a good idea to always predict $\hat{y} = 0$?

- **No, surely not**
- We don't care about being right **often**, we care about **losing as little as possible**
  - One type of error is much more painful (false negative)

Decision Theory – Objective

- We want to find a prediction that minimizes the **expected loss**

$$\underset{\hat{y} \in \{0,1\}}{\operatorname{argmin}} \; \mathbb{E}_{y \sim p^*(y \mid \mathbf{x})} \left[\mathcal{L}(y, \hat{y}) \right]$$

  with $\mathcal{L}(0,0) = \mathcal{L}(1,1) = 0$, $\mathcal{L}(0,1) = 1$, and $\mathcal{L}(1,0) = 100$.

- With $\hat{y} = 0$ we have

$$\mathbb{E}_{y \sim p^*(y \mid \mathbf{x})} \left[\mathcal{L}(y, \hat{y}) \right] = 0.7 \cdot 0\$ + 0.3 \cdot 100\$ = 30\$$$

- And with $\hat{y} = 1$ we have

$$\mathbb{E}_{y \sim p^*(y \mid \mathbf{x})} \left[\mathcal{L}(y, \hat{y}) \right] = 0.7 \cdot 1\$ + 0.3 \cdot 0\$ = 0.7\$$$

- We should predict $\hat{y} = 1$ in this game

Decision Functions

- We just focused on **one particular** conditional (**fixed x**)
- To globally act optimally, we can **act optimally for each x**
- For a different **x**, **the posterior** $p(y \mid \mathbf{x})$ **changes**
  - i.e., the coin changes
- We want a **decision function** $f : \mathbb{R}^D \to \{0, 1\}$ that minimizes

$$\mathbb{E}_{y \sim p^*(y \mid \mathbf{x})} \left[\mathcal{L}(y, f(\mathbf{x})) \right]$$

  for each x

- This f will then also minimize

$$\mathbb{E}_{(\mathbf{x},y) \sim p^*(\mathbf{x},y)} \left[\mathcal{L}(y, f(\mathbf{x})) \right] = \mathbb{E}_{\mathbf{x} \sim p^*(\mathbf{x})} \left[\mathbb{E}_{y \sim p^*(y \mid \mathbf{x})} \left[\mathcal{L}(y, f(\mathbf{x})) \right] \right]$$