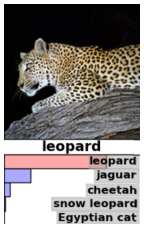
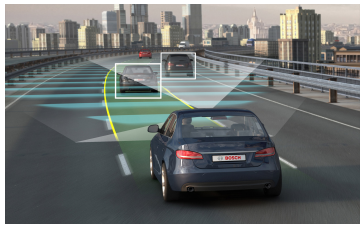
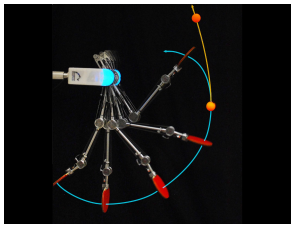
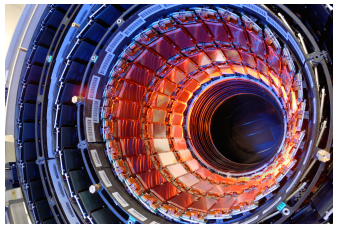
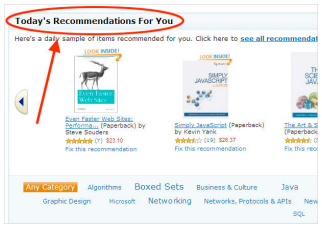
Mathematical Foundation in Deep Learning

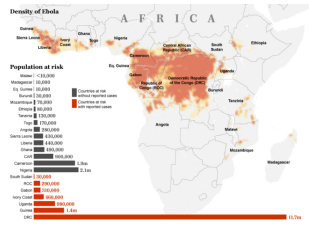
References:

- deeplearningbook.org
- Dive into Deep Learning, https://d2l.ai/index.html
- introtodeeplearning.com
- deeplearningindaba.com

Mathematical Concepts in Machine Learning





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

Outline

Key Concepts in Probability
- Probability density function
- Sum & Product Rules
- Bayes' Theorem
- Maximum Likelihood Estimation , Cross Entropy, KL Divergence

Differentiation
- Where it is used in ML?
- Types of differentiation
- Gradients in Neural Networks

Overview

Key Concepts in Probability

- Probability density function
- Sum & Product Rules
- Bayes' Theorem
- Maximum Likelihood Estimation , Cross Entropy, KL Divergence

Differentiation

- Where it is used in ML?
- Types of differentiation
- Gradients in Neural Networks

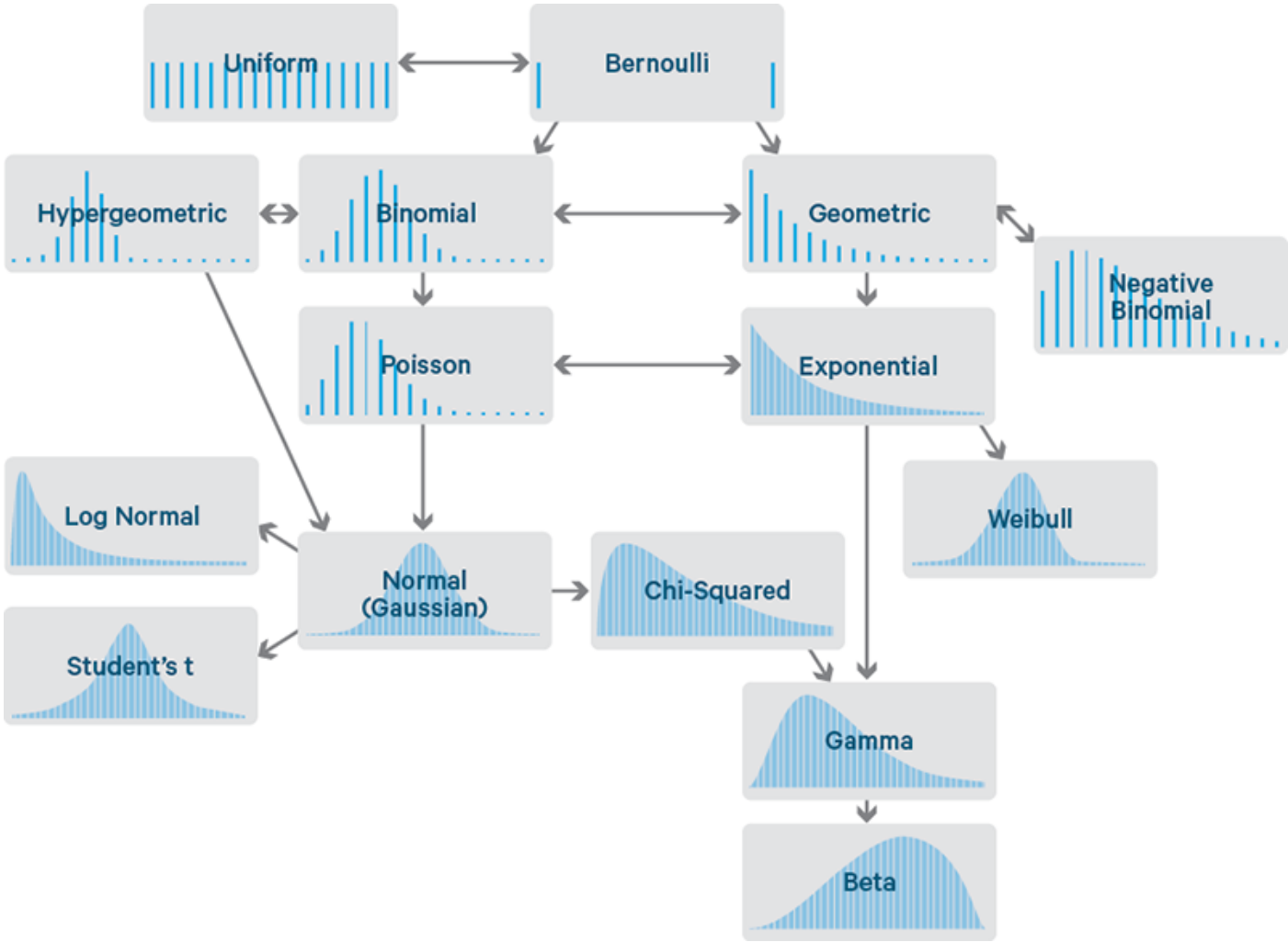
Probability density functions

A function $p : \mathbb{R}^D \to \mathbb{R}$ is called a probability density function if

1. Its integral exists,
2. $\forall \boldsymbol{x} \in \mathbb{R}^D : p(\boldsymbol{x}) \geqslant 0$ and
3. $\int_{\mathbb{R}^D} p(\boldsymbol{x}) d\boldsymbol{x} = 1$ *

* Here, $\boldsymbol{x} \in \mathbb{R}^D$ is a (continuous) random variable. For discrete random variables, the integral is replaced with a sum.

Probability density functions



\blue{http://www.math.wm.edu/~leemis/chart/UDR/UDR.html}

Expectation and Variance

- Expectation of random variable X

$$\text{Discrete: } E[X] = \sum_{x \in X} x \; p(x), \text{ Continuous: } E[X] = \int_X x \, p(x) dx$$

- Let g be any function of X,

Discrete: $E[g(X)] = \sum_{x \in X} g(x) \; p(x)$ Continuous: $E[g(X)] = \int_X g(x) \, p(x) dx$

- Variance:

$$Var[X] = E[(X - E[X])^2] = E[X^2] - (E[X])^2$$

- Covariance of variable X and Y:

$$Cov[X, Y] = E[(X - E[X])(Y - E[Y])]$$

- Some properties:

  - ✓ $E[c] = c$, where c is a constant
  - ✓ $E[cX] = cE(X)$
  - ✓ $E[X + Y] = E[X] + E[Y]$
  - ✓ $Var[c] = 0$
  - ✓ $Var[c + X] = Var[X]$
  - ✓ $Var[cX] = c^2 Var[X]$
  - ✓ $Var[X + Y] = Var[X] + Var[Y]$ **ONLY IF X and Y are independent!**
  - ✓ $Var[X + Y] = Var[X] + Var[Y] + \mathbf{2Cov[X, Y]}$ if X and Y are not independent!

Random vectors in high dimensions

- Random vector $X \in \mathbb{R}^n,\ X = \{X_1, X_2, \dots, X_n\}$
- Expectation vector: $\mu = E[X] = \{E[X_1], E[X_2], \dots, E[X_n]\} = \{\mu_1, \mu_2, \dots, \mu_n\}$
- Variance is replaced by the covariance matrix:
$$\Sigma = Cov[X] = E[(X - \mu)(X - \mu)^T] = E[XX^T] - \mu\mu^T$$
- Σ is $n \times n$, symmetric matrix, where
$$Cov[X]_{i,j} = E[(X_i - \mu_i)(X_j - \mu_j)] = E[X_i X_j] - \mu_i\, \mu_j$$
- For example: Bivariate Normal Distribution

$$\mathbf{x} = \begin{pmatrix} x \\ y \end{pmatrix} \qquad \boldsymbol{\mu} = \begin{pmatrix} \mu_X \\ \mu_Y \end{pmatrix} \qquad \boldsymbol{\Sigma} = \begin{pmatrix} \sigma_X^2 & \rho\sigma_X\sigma_Y \\ \rho\sigma_X\sigma_Y & \sigma_Y^2 \end{pmatrix}$$

Density $p(\mathbf{x}) = \frac{1}{2\pi}(\det \boldsymbol{\Sigma})^{-1/2} \exp\left[-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})\right]$

Sum & Product Rules

$$p(\boldsymbol{x}) = \int p(\boldsymbol{x}, \boldsymbol{y}) d\boldsymbol{y} \qquad \text{Sum rule/Marginalization property}$$

$$p(\boldsymbol{x}, \boldsymbol{y}) = p(\boldsymbol{y}|\boldsymbol{x}) p(\boldsymbol{x}) \qquad \text{Product rule}$$

- $p(\boldsymbol{x}, \boldsymbol{y})$ is the joint probability distribution of $\boldsymbol{x}$ and $\boldsymbol{y}$
- $p(\boldsymbol{x})$ and $p(\boldsymbol{y})$ are the marginal probability distributions of $\boldsymbol{x}$ and $\boldsymbol{y}$
- $p(\boldsymbol{y}|\boldsymbol{x})$ is the conditional probability distribution of $\boldsymbol{y}$ given $\boldsymbol{x}$

Sum & Product Rules

$$p(\boldsymbol{x}) = \int p(\boldsymbol{x}, \boldsymbol{y}) d\boldsymbol{y} \qquad \text{Sum rule/Marginalization property}$$

$$p(\boldsymbol{x}, \boldsymbol{y}) = p(\boldsymbol{y}|\boldsymbol{x}) p(\boldsymbol{x}) \qquad \text{Product rule}$$

- $p(\boldsymbol{x}, \boldsymbol{y})$ is the joint probability distribution of $\boldsymbol{x}$ and $\boldsymbol{y}$
- $p(\boldsymbol{x})$ and $p(\boldsymbol{y})$ are the marginal probability distributions of $\boldsymbol{x}$ and $\boldsymbol{y}$
- $p(\boldsymbol{y}|\boldsymbol{x})$ is the conditional probability distribution of $\boldsymbol{y}$ given $\boldsymbol{x}$

Sum & Product Rules

$$p(\boldsymbol{x}) = \int p(\boldsymbol{x}, \boldsymbol{y}) d\boldsymbol{y} \qquad \text{Sum rule/Marginalization property}$$

$$p(\boldsymbol{x}, \boldsymbol{y}) = p(\boldsymbol{y}|\boldsymbol{x}) p(\boldsymbol{x}) \qquad \text{Product rule}$$

- $p(\boldsymbol{x}, \boldsymbol{y})$ is the joint probability distribution of $\boldsymbol{x}$ and $\boldsymbol{y}$
- $p(\boldsymbol{x})$ and $p(\boldsymbol{y})$ are the marginal probability distributions of $\boldsymbol{x}$ and $\boldsymbol{y}$
- $p(\boldsymbol{y}|\boldsymbol{x})$ is the conditional probability distribution of $\boldsymbol{y}$ given $\boldsymbol{x}$

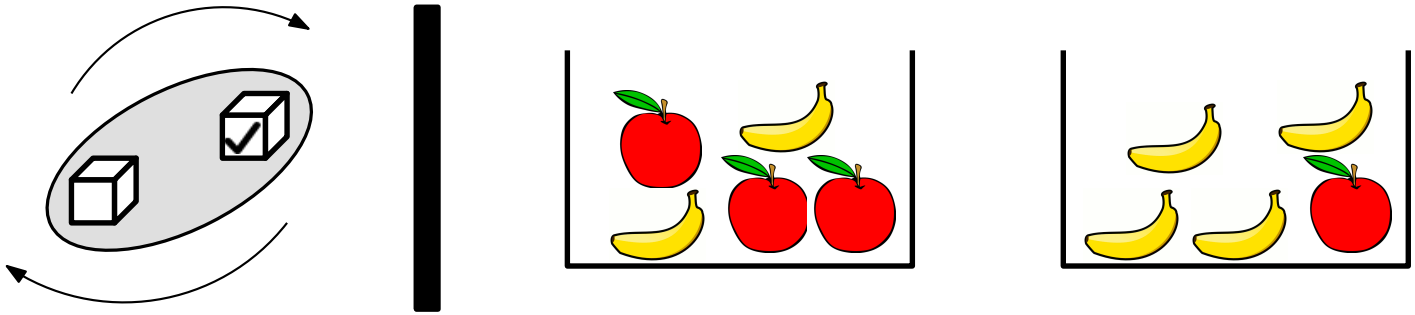
Sum & Product Rules

$$p(\boldsymbol{x}) = \int p(\boldsymbol{x}, \boldsymbol{y}) d\boldsymbol{y} \qquad \text{Sum rule/Marginalization property}$$

$$p(\boldsymbol{x}, \boldsymbol{y}) = p(\boldsymbol{y}|\boldsymbol{x}) p(\boldsymbol{x}) \qquad \text{Product rule}$$

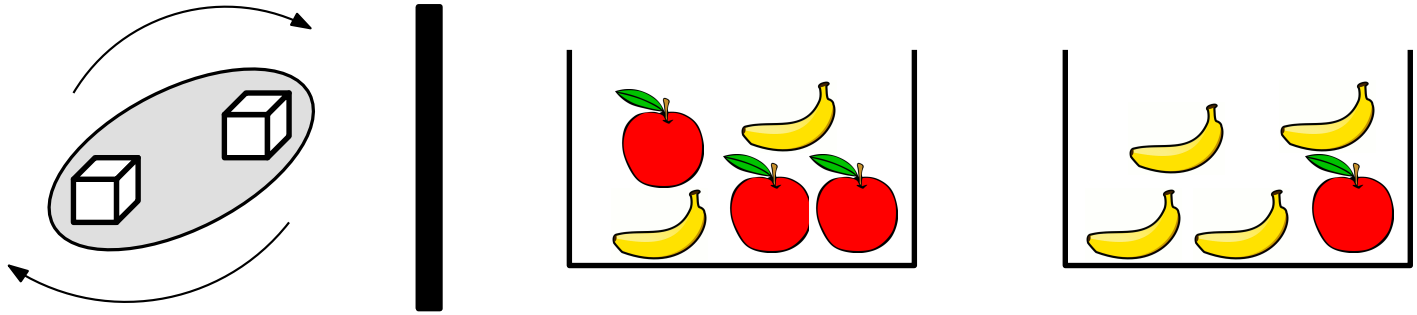
- $p(\boldsymbol{x}, \boldsymbol{y})$ is the joint probability distribution of $\boldsymbol{x}$ and $\boldsymbol{y}$
- $p(\boldsymbol{x})$ and $p(\boldsymbol{y})$ are the marginal probability distributions of $\boldsymbol{x}$ and $\boldsymbol{y}$
- $p(\boldsymbol{y}|\boldsymbol{x})$ is the conditional probability distribution of $\boldsymbol{y}$ given $\boldsymbol{x}$

Illustration: Bayesian Inference



Two cartons $\boldsymbol{x} = \{c_1, c_2\}$ with fruit in each $\boldsymbol{y} = \{\text{banana}, \text{apple}\}$
Say we pick a banana ($\boldsymbol{y} = \text{banana}$)
We want to estimate $p(\boldsymbol{x} = c_1 | \boldsymbol{y} = \text{banana})$

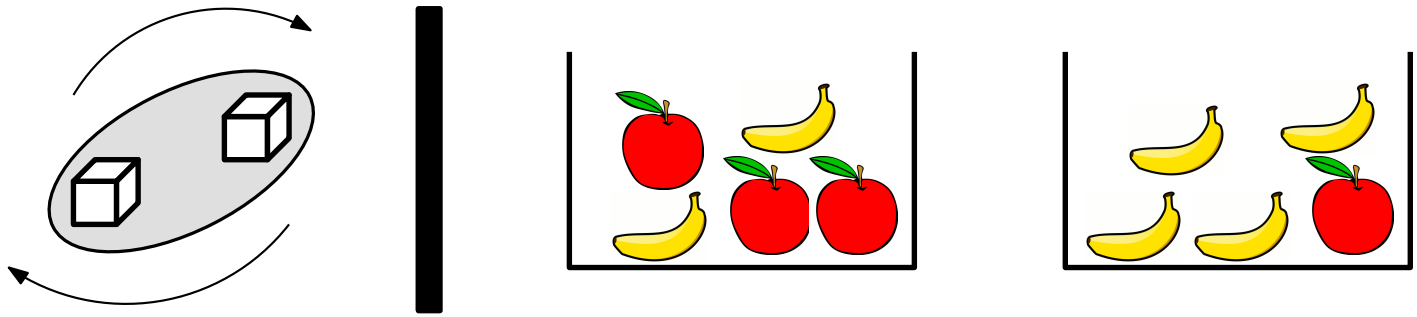
Illustration: Bayesian Inference



$$p(c_1) = 1/2 = p(c_2)$$

Prior

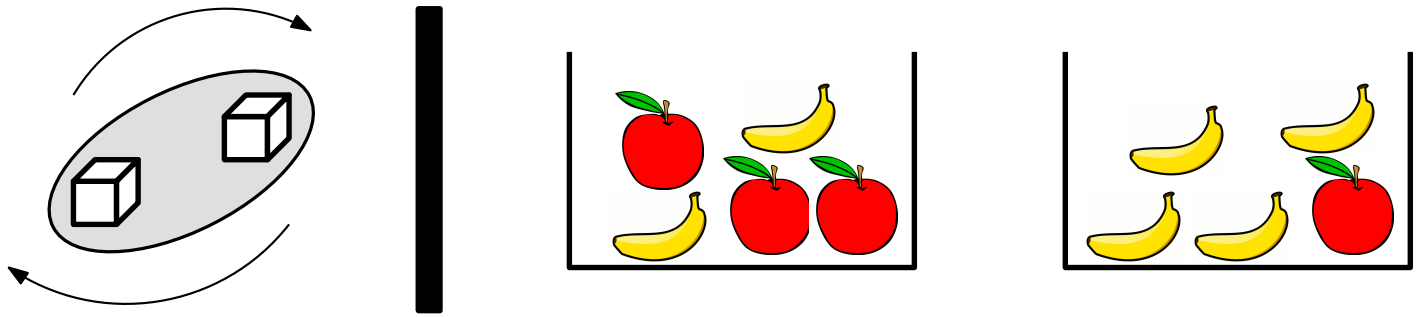
Illustration: Bayesian Inference



$p(c_1) = 1/2 = p(c_2)$ **Prior**

$p(\text{banana}|c_1) = 2/5\,, \quad p(\text{banana}|c_2) = 4/5$ **Likelihood**

Illustration: Bayesian Inference



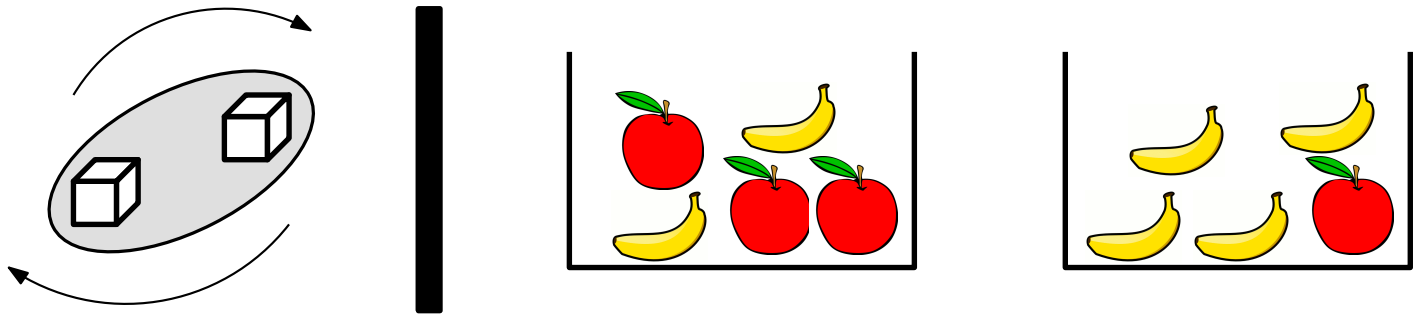
$p(c_1) = 1/2 = p(c_2)$ **Prior**

$p(\text{banana}|c_1) = 2/5\,, \quad p(\text{banana}|c_2) = 4/5$ **Likelihood**

$p(\text{banana}) = \sum_{\boldsymbol{x}} p(\boldsymbol{x}, \boldsymbol{y}) = \sum_{\boldsymbol{x}} p(\boldsymbol{y}|\boldsymbol{x})p(\boldsymbol{x}) =$ **Evidence**

$p(\text{banana}|c_1)p(c_1) + p(\text{banana}|c_2)p(c_2) = 6/10 = 3/5$

Illustration: Bayesian Inference



$p(c_1) = 1/2 = p(c_2)$ **Prior**

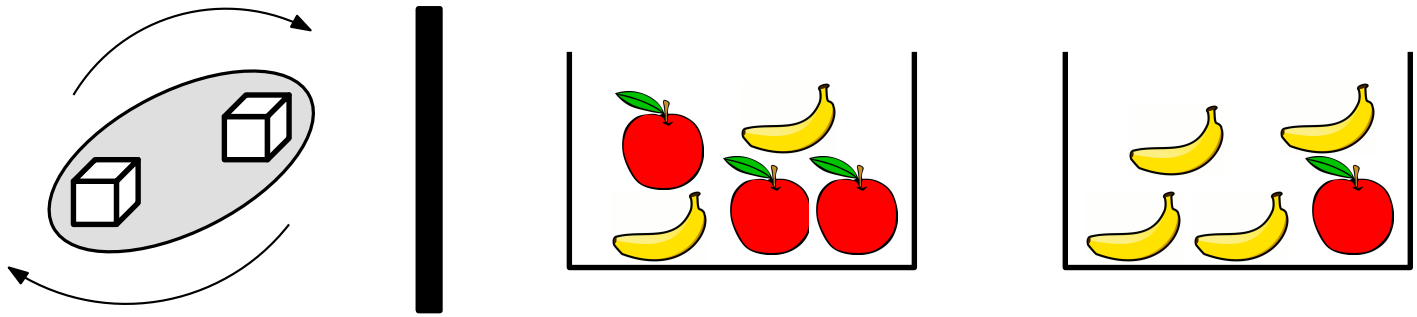
$p(\text{banana}|c_1) = 2/5\,, \quad p(\text{banana}|c_2) = 4/5$ **Likelihood**

$p(\text{banana}) = \sum_{\boldsymbol{x}} p(\boldsymbol{x}, \boldsymbol{y}) = \sum_{\boldsymbol{x}} p(\boldsymbol{y}|\boldsymbol{x})p(\boldsymbol{x}) =$ **Evidence**

$p(\text{banana}|c_1)p(c_1) + p(\text{banana}|c_2)p(c_2) = 6/10 = 3/5$

What is $p(\boldsymbol{x} = c_1|\boldsymbol{y} = \text{banana})$? **Posterior**

Illustration: Bayesian Inference



$p(c_1) = 1/2 = p(c_2)$ **Prior**

$p(\text{banana}|c_1) = 2/5\,, \quad p(\text{banana}|c_2) = 4/5$ **Likelihood**

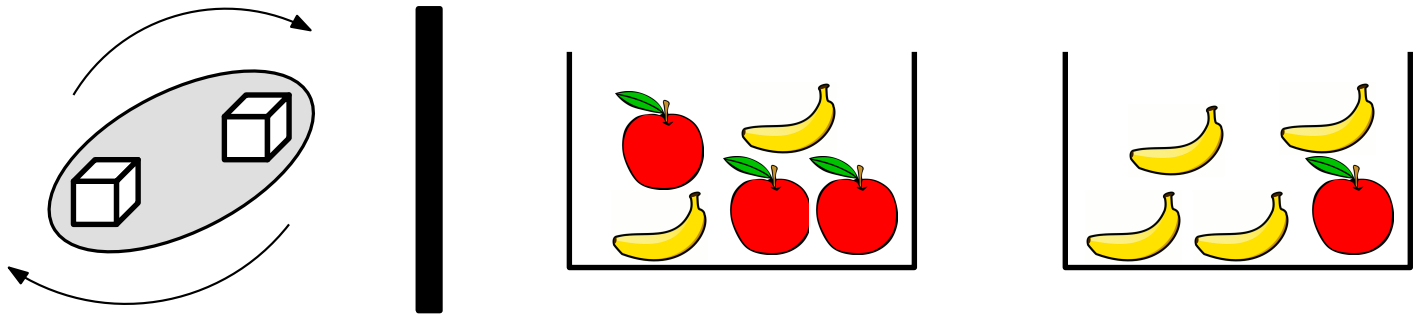
$p(\text{banana}) = \sum_{\boldsymbol{x}} p(\boldsymbol{x}, \boldsymbol{y}) = \sum_{\boldsymbol{x}} p(\boldsymbol{y}|\boldsymbol{x})p(\boldsymbol{x}) =$ **Evidence**

$p(\text{banana}|c_1)p(c_1) + p(\text{banana}|c_2)p(c_2) = 6/10 = 3/5$

What is $p(\boldsymbol{x} = c_1|\boldsymbol{y} = \text{banana})$? **Posterior**

Bayes' Theorem: $\text{Posterior} = \frac{\text{Likelihood} \times \text{Prior}}{\text{Evidence}}$

Illustration: Bayesian Inference



$p(c_1) = 1/2 = p(c_2)$ **Prior**

$p(\text{banana}|c_1) = 2/5\,, \quad p(\text{banana}|c_2) = 4/5$ **Likelihood**

$p(\text{banana}) = \sum_{\boldsymbol{x}} p(\boldsymbol{x}, \boldsymbol{y}) = \sum_{\boldsymbol{x}} p(\boldsymbol{y}|\boldsymbol{x})p(\boldsymbol{x}) =$ **Evidence**

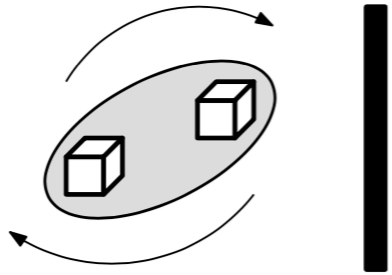
$p(\text{banana}|c_1)p(c_1) + p(\text{banana}|c_2)p(c_2) = 6/10 = 3/5$

What is $p(\boldsymbol{x} = c_1|\boldsymbol{y} = \text{banana})$? **Posterior**

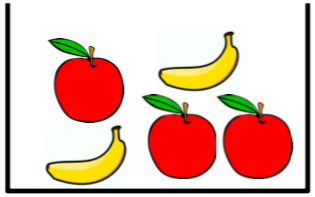
Bayes' Theorem: $\text{Posterior} = \frac{\text{Likelihood} \times \text{Prior}}{\text{Evidence}}$

$$p(\boldsymbol{x} = c_1|\boldsymbol{y} = \text{banana}) = \frac{p(\text{banana}|c_1)p(c_1)}{p(\text{banana})} = \frac{2/5 * 1/2}{3/5} = 1/3$$

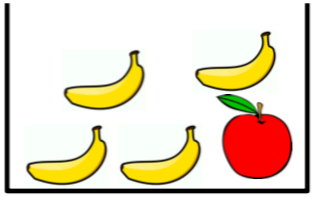
This example is analogous to sentiment detection



C: Positive or negative



d1: {good, fair, wonderful, …}



d2: {poor, bad, fair, …}

- For a document *d* and a class *c*

$$P(c \mid d) = \frac{P(d \mid c)P(c)}{P(d)}$$

Naïve Bayes Classifier (I)

$$c_{MAP} = \underset{c \in C}{\operatorname{argmax}}\, P(c \mid d)$$

MAP is "maximum a posteriori" = most likely class

$$= \underset{c \in C}{\operatorname{argmax}}\, \frac{P(d \mid c)P(c)}{P(d)}$$

Bayes Rule

$$= \underset{c \in C}{\operatorname{argmax}}\, P(d \mid c)P(c)$$

Dropping the denominator

Where $p(d|c) = p(w_1, w_2, \ldots, w_n|c)$

$= \prod_{i=1}^{n} p(w_i|c)$ (independence assumption)

Chain Rule (or Product Rule) of Conditional Probabilities

- Any joint probability distribution over many random variables may be decomposed into conditional distributions over only one variable:

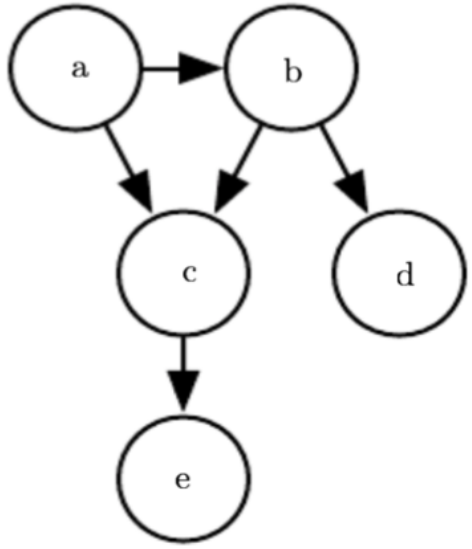
$$P(\mathrm{x}^{(1)}, \ldots, \mathrm{x}^{(n)}) = P(\mathrm{x}^{(1)}) \Pi_{i=2}^{n} P(\mathrm{x}^{(i)} \mid \mathrm{x}^{(1)}, \ldots, \mathrm{x}^{(i-1)})$$

$$\begin{aligned} P(\mathrm{a}, \mathrm{b}, \mathrm{c}) &= P(\mathrm{a} \mid \mathrm{b}, \mathrm{c}) P(\mathrm{b}, \mathrm{c}) \\ P(\mathrm{b}, \mathrm{c}) &= P(\mathrm{b} \mid \mathrm{c}) P(\mathrm{c}) \\ P(\mathrm{a}, \mathrm{b}, \mathrm{c}) &= P(\mathrm{a} \mid \mathrm{b}, \mathrm{c}) P(\mathrm{b} \mid \mathrm{c}) P(\mathrm{c}) \end{aligned}$$

- Sometimes, a variable may only depend on a few other variables, e.g. variable a only depends on b. Thus, $p(a|b,c) = p(a|b)$
- **Structured probabilistic model or graph model**: factorizations into conditional probability distributions

Structured Probabilistic Models

- Machine learning algorithms often involve probability distributions over a very large number of random variables, which may interact with each other.
- For example, for a document d, $p(d) = p(w_1, w_2, \ldots, w_n)$
- Structured probabilistic model or graphical model:
  - ✓ Node: random variable
  - ✓ Edge: conditional probability of target node given source node



$$p(\mathrm{a}, \mathrm{b}, \mathrm{c}, \mathrm{d}, \mathrm{e}) = p(\mathrm{a})p(\mathrm{b} \mid \mathrm{a})p(\mathrm{c} \mid \mathrm{a}, \mathrm{b})p(\mathrm{d} \mid \mathrm{b})p(\mathrm{e} \mid \mathrm{c})$$

Probability Chain Rule and Language Model

- Try the following tongue twister!

Peter Piper picked a peck of pickled pepper.
Where's the pickled pepper that Peter Piper picked?

Bigram model: $p(w_{n+1}|w_1, w_2, \ldots, w_n) = p(w_{n+1}|w_n)$

P("Where's the pickled pepper that Peter Piper picked") =?

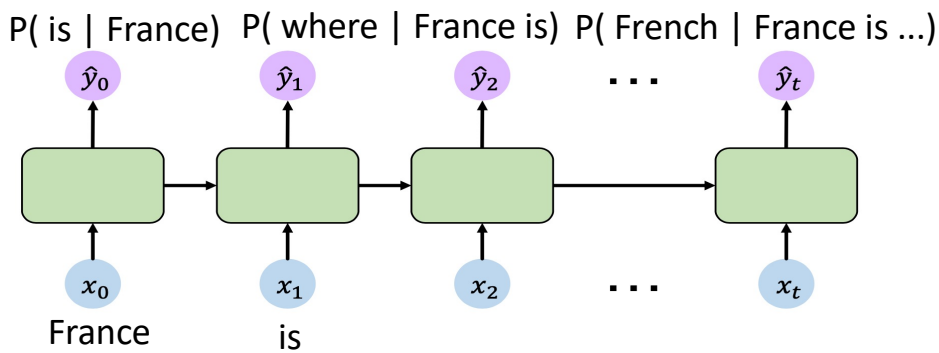
$$
\begin{aligned}
& \mathrm{P}\begin{pmatrix}\text{Peter Piper picked a peck of pickled pepper}\\ \text{that Peter Piper picked}\end{pmatrix} \\
= \;& \mathrm{P}(\text{Peter}|\text{<s>}) \times \mathrm{P}(\text{Piper}|\text{Peter}) \times \mathrm{P}(\text{picked}|\text{Piper}) \times \\
& \mathrm{P}(\text{a}|\text{picked}) \times \mathrm{P}(\text{peck}|\text{a}) \times \mathrm{P}(\text{of}|\text{peck}) \times \\
& \mathrm{P}(\text{pickled}|\text{of}) \times \mathrm{P}(\text{pepper}|\text{pickled}) \times \mathrm{P}(\text{that}|\text{pepper}) \times \\
& \mathrm{P}(\text{Peter}|\text{that}) \times \mathrm{P}(\text{Piper}|\text{Peter}) \times \mathrm{P}(\text{picked}|\text{Piper}) \times \\
& \mathrm{P}(\text{</s>}|\text{picked}) \\
= \;& .5 \times 1 \times 1 \times .5 \times 1 \times 1 \times 1 \times 1 \times .5 \times 1 \times 1 \times .5 \\
= \;& .0625
\end{aligned}
$$

$$\mathrm{P}(\text{Piper}|\text{Peter}) = \frac{|\text{Peter Piper}|}{|\text{Peter}|} = \frac{2}{2} = 1$$

Reference:
https://faculty.sbs.arizona.edu/hammond/archive/ling696f-sp03/snlp4.pdf

- Neural Network Language Model:

France is where I grew up, but I now live in Boston. I speak fluent ___.



Maximum Likelihood Estimation for supervised learning

- Assume supervised learning with dataset $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$
- A model with parameters θ
- $q_\theta(y_i|x_i)$: the predicted likelihood of y_i from the model for sample x_i.
- Find θ such that the product of the likelihood for all the samples in the dataset is maximized:

$$\underset{\theta}{\operatorname{argmax}} \prod_{i=1}^{n} q_\theta(y_i|x_i)$$

Equivalent to:

$$\underset{\theta}{\operatorname{argmax}} \sum_{i=1}^{n} \log q_\theta(y_i|x_i) \quad \text{or}$$

$$\underset{\theta}{\operatorname{argmin}} - \sum_{i=1}^{n} \log q_\theta(y_i|x_i)$$

Shannon Entropy

- Given a distribution p over a given variable X,

  Shannon Entropy: $\mathrm{H}(p) = E_p[-\log p]$

- If X is continuous: $\mathrm{H}(p) = -\int_X p(x)\log p(x)dx$
- If X is discrete: $\mathrm{H}(p) = -\sum_{x\in X} p(x)\log p(x)$

- Intuition:
  - ✓ $\mathrm{H}(p)$ quantifies the amount of uncertainty in p
  - ✓ It gives a lower bound on the number of bits needed on average to encode symbols drawn from p

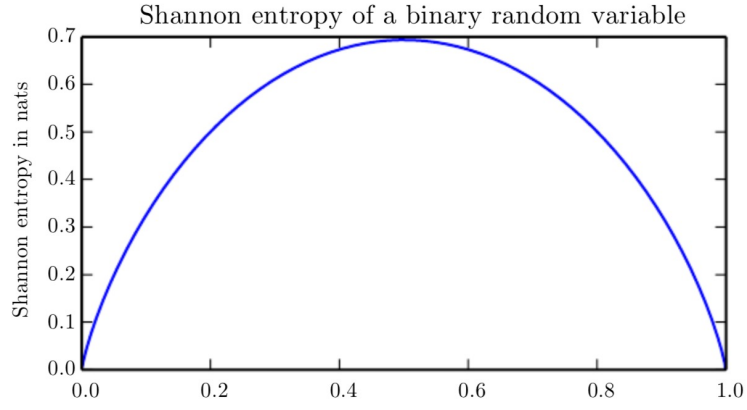


Figure 3.5: This plot shows how distributions that are closer to deterministic have low Shannon entropy while distributions that are close to uniform have high Shannon entropy. On the horizontal axis, we plot p, the probability of a binary random variable being equal to 1. The entropy is given by $(p-1)\log(1-p) - p\log p$. When p is near 0, the distribution is nearly deterministic, because the random variable is nearly always 0. When p is near 1, the distribution is nearly deterministic, because the random variable is nearly always 1. When $p = 0.5$, the entropy is maximal, because the distribution is uniform over the two outcomes.

Quick Quiz!

Cross Entropy

- Given two distributions p and q over a given variable X, cross entropy of q relative to p is:

$$\text{Cross Entropy } \mathrm{H}(p,q) = E_p[-\log q]$$

- If X is continuous:

$$\mathrm{H}(p,q) = -\int_X p(x)\log q(x)dx$$

- If X is discrete:

$$\mathrm{H}(p,q) = -\sum_{x\in X} p(x)\log q(x)$$

- $\mathrm{H}(p,q)$ measures how many bits needed on average to encode data if a coding scheme is optimized for an estimated probability distribution q rather than the true distribution p
- $\mathrm{H}(p,q)$ is used as a cost function. The more $p(x)$ approximates $q(x)$, the smaller the cost. Assuming x is binary, try the following:
  - If $p(x=1) = 0.99$ while $q(x=1) = 0.01, H(p,q) = ?$
  - If $p(x=1) = 0.99$ while $q(x=1) = 0.98, H(p,q) = ?$

Question: how to calculate cross entropy cost in single label or multi-label classification?

Cross Entropy Cost and Maximum Likelihood Estimation

- Again, supervised learning with dataset $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$
- $p(y|x_i)$: the distribution for the ground truth label (y) for feature x_i:

$$p(y|x_i) = \begin{cases} 1, & y = y_i \\ 0, & otherwise \end{cases}$$

- $q_\theta(y|x_i)$: the predicted likelihood of y from the model for sample x_i
- Cross entropy of $q_\theta(y|x_i)$ relative to $p(y|x_i)$:

$$H_i(p, q_\theta) = -\sum_{y \in Y} p(y|x_i) \log q_\theta(y|x_i) = -\log q_\theta(y_i|x_i)$$

- Total **cross entropy cost** (loss function) :

$$L = \sum_{i=1}^{n} H_i(p, q_\theta) = \sum_{i=1}^{n} -\log q_\theta(y_i|x_i)$$

- Thus the optimization goal is:

$$\underset{\theta}{\operatorname{argmin}} L = \sum_{i=1}^{n} -\log q_\theta(y_i|x_i)$$

- This is exactly **Maximum Likelihood Estimation**

Cross Entropy Cost and Maximum Likelihood Estimation

- Again, supervised learning with dataset $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$
- $p(y|x_i)$: the distribution for the ground truth label (y) for feature x_i:
  $p(y|x_i) = \begin{cases} 1, & y = y_i \\ 0, & otherwise \end{cases}$
- $q_\theta(y|x_i)$: the predicted likelihood of y from the model for sample x_i
- Cross entropy of $q_\theta(y|x_i)$ relative to $p(y|x_i)$:
  $$H_i(p, q_\theta) = -\sum_{y \in Y} p(y|x_i) \log q_\theta(y|x_i) = -\log q_\theta(y_i|x_i)$$
- Total **cross entropy cost** (loss function) :
  $$L = \sum_{i=1}^{n} H_i(p, q_\theta) = \sum_{i=1}^{n} -\log q_\theta(y_i|x_i)$$
- Thus the optimization goal is:
  $$\underset{\theta}{\operatorname{argmin}} L = \sum_{i=1}^{n} -\log q_\theta(y_i|x_i)$$
- This is exactly **Maximum Likelihood Estimation**

Kullback–Leibler (KL) Divergence

- Given two distributions p and q over a given variable X, KL divergence measures how different these two distributions are:

$$D_{KL}(p||q) = E_p[-\log\frac{q}{p}]$$

- If X is continuous: $D_{KL}(p||q) = -\int_X p(x)\log\frac{q(x)}{p(x)}\,dx$
- If X is discrete: $D_{KL}(p||q) = -\sum_{x\in X} p(x)\log\frac{q(x)}{p(x)}$
- $D_{KL}(p||q)$ properties:
  - Non-negative (let $f(x) = -log\frac{q(x)}{p(x)}$, show it by using Jensen inequality: $E(f(x)) \geq f(E(x))$ when $f(x)$ is convex)
  - $D_{KL}(p||q) = 0$ if and only if p and q are equal
  - Not symmetric, $D_{KL}(p||q) \neq D_{KL}(q||p)$ (not a true distance measure)
- It is also called relative entropy:

$$D_{KL}(p||q) = E_p[-\log\frac{q}{p}] = E_p[-\log q + \log p] = H(p,q) - H(p)$$

- Thus, $H(p,q) = D_{KL}(p||q) + H(p)$

Cross Entropy and KL Divergence

- Optimizing cross entropy :

$$\underset{\theta}{\operatorname{argmin}} L = \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{n} H_i(p, q_\theta)$$

$$= \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{n} [D_{KL,i}(p||q_\theta) + H_i(p)]$$

$$= \underset{\theta}{\operatorname{argmin}} \sum_{i=1}^{n} [D_{KL,i}(p||q_\theta)]$$

because $H_i(p)$ is independent of θ

- Thus, in classification problems, optimizing the sum of **cross entropy** over all the training samples is equivalent to
  - Optimizing the sum of **KL divergence** over all the training samples
  - **Maximum likelihood estimation**

Overview

Key Concepts in Probability

- Probability density function
- Sum & Product Rules
- Bayes' Theorem
- Maximum Likelihood Estimation , Cross Entropy, KL Divergence

Differentiation

- Linear algebra and notations
- Where it is used in ML?
- Types of differentiation
- Gradients in Neural Networks

Linear Algebra Notation

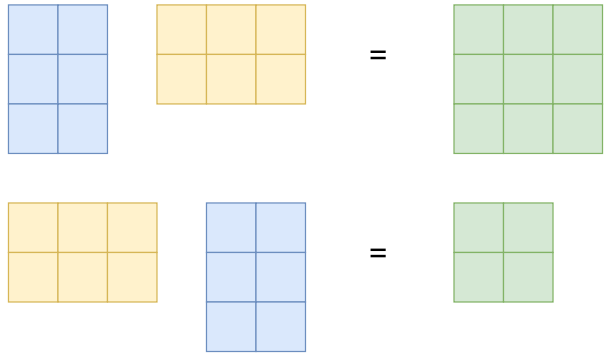
- **Scalar**: A single number
- **Vector**: An array of numbers, i.e. $\boldsymbol{x} \in \boldsymbol{R}^n$

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \text{ or } \boldsymbol{x} = [x_1, x_2, \ldots, x_n]^T$$

- **Matrix**: A 2-D array of numbers, e.g. $\boldsymbol{A} \in \boldsymbol{R}^{m \times n}$
  - Each element is identified by two indices, e.g. $A_{i,j}$
  - $A_{i,:}$ is the ith row and $A_{:,j}$ is the jth column
  - Transpose: $\boldsymbol{A}^T \in \boldsymbol{R}^{n \times m}$, $(\boldsymbol{A}^T)_{i,j} = \boldsymbol{A}_{j,i}$
  - A **vector** can be thought of as a **matrix with only one column**
  - The **transpose of a vector** is therefore a **matrix with only one row**
  - A scalar a can be thought of as a matrix with only one value. $a^T = a$.
- **Tensor**: An array of numbers with more than two axes is known as a tensor. The element of a tensor $\mathbf{A} \in \boldsymbol{R}^{m \times n \times p}$ at coordinates (i, j, k) is denoted as $\mathbf{A}_{i,j,k}$.

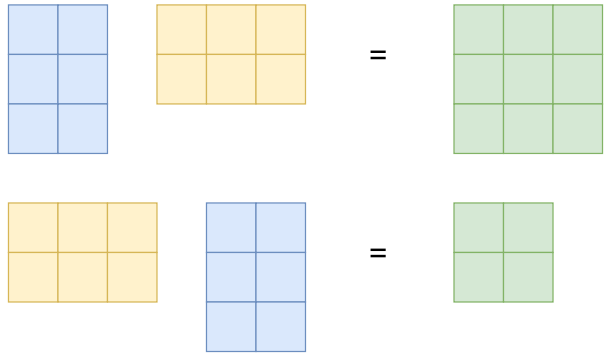
Background: Matrix Multiplication

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



Background: Matrix Multiplication

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

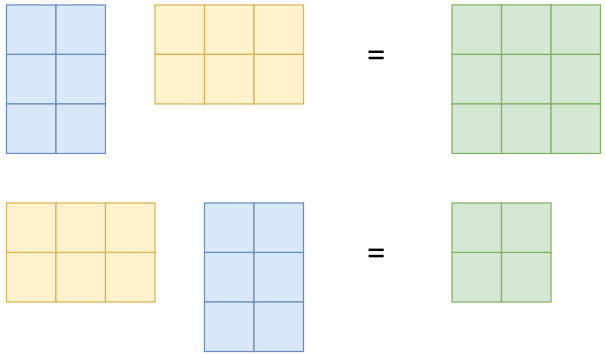


- When multiplying matrices, the "neighbouring" dimensions have to fit:

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

Background: Matrix Multiplication

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



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

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

	`import numpy as np`
$\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$	`y = A.dot(x)`
$y_i = \sum_j A_{ij} x_j$	`y = np.einsum('ij, j', A, x)`
$\boldsymbol{C} = \boldsymbol{A}\boldsymbol{B}$	`C = A.dot(B)`
$C_{ij} = \sum_k A_{ik} B_{kj}$	`C = np.einsum('ik, kj', A, B)`

Multiplying Matrices and Vectors

- **Matrix Product**:

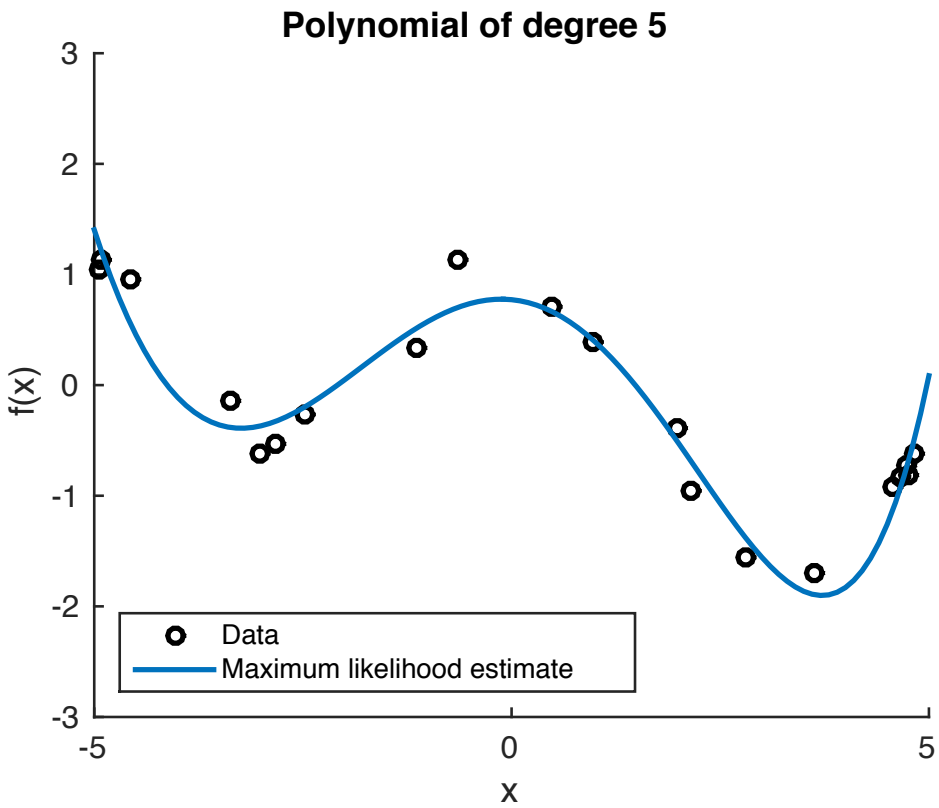
$$\boldsymbol{C} = \boldsymbol{AB}, \text{where } C_{i,j} = \sum_k A_{i,k} B_{k,j}$$

- **Element wise product** (Hadamard product):

$$\boldsymbol{C} = \boldsymbol{A} \odot \boldsymbol{B} \text{ , where } C_{i,j} = A_{i,j} B_{i,j}$$

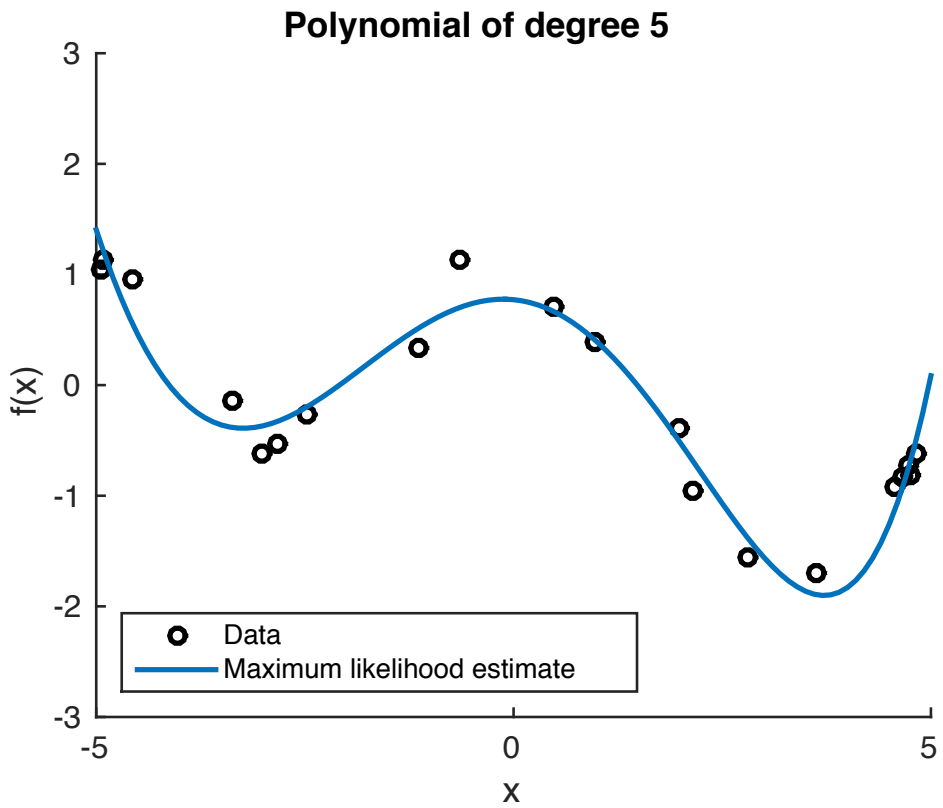
- **Dot product** between two vectors $\boldsymbol{x}$ and $\boldsymbol{y}$ ($\boldsymbol{x}, \boldsymbol{y} \in \boldsymbol{R}^n$) is $\boldsymbol{x}^{\mathrm{T}} \boldsymbol{y}$
  - What is the dimension of $\boldsymbol{x}^{\mathrm{T}} \boldsymbol{y}$?
  - We can think of the matrix product $\boldsymbol{C} = \boldsymbol{AB}$ as computing $C_{i,j}$ as the dot product between row i of $\boldsymbol{A}$ and column j of $\boldsymbol{B}$.
- Useful properties:
  - $\boldsymbol{AB} \neq \boldsymbol{BA}, \quad \boldsymbol{A}(\boldsymbol{BC}) = (\boldsymbol{AB})\boldsymbol{C}, \quad \boldsymbol{A}(\boldsymbol{B} + \boldsymbol{C}) = \boldsymbol{AB} + \boldsymbol{AC}$
  - $(\boldsymbol{AB})^{\boldsymbol{T}} = \boldsymbol{B}^{\boldsymbol{T}} \boldsymbol{A}^{\boldsymbol{T}}$
  - $\boldsymbol{x}^{\mathrm{T}} \boldsymbol{y} = \boldsymbol{y}^{\mathrm{T}} \boldsymbol{x}$ (how to demonstrate it?)

Scenario 1: Find optimal parameters to fit a 1D curve



- **Given**: N data pairs (x_i, y_i) (black dots) where x_i is input & y_i is output/target

Scenario 1: Find optimal parameters to fit a 1D curve



- **Given**: N data pairs (x_i, y_i) (black dots) where x_i is input & y_i is output/target
- **Goal**: Learn a model f (blue curve) that best fits the data

$$f(x_i) \approx y_i$$

Scenario 1: Find optimal parameters to fit a 1D curve

- Model f is D-degree polynomial parametrised by $\boldsymbol{\theta}$, or $f_{\boldsymbol{\theta}}$

$$f_{\boldsymbol{\theta}}(x) = \theta_0 x^0 + \theta_1 x^1 + \theta_2 x^2 + \cdots + \theta_{D-1} x^{D-1} + \theta_D x^D$$

Scenario 1: Find optimal parameters to fit a 1D curve

- Model f is D-degree polynomial parametrised by $\boldsymbol{\theta}$, or $f_{\boldsymbol{\theta}}$

$$f_{\boldsymbol{\theta}}(x) = \theta_0 x^0 + \theta_1 x^1 + \theta_2 x^2 + \cdots + \theta_{D-1} x^{D-1} + \theta_D x^D$$

$$= \begin{bmatrix} \theta_0 & \theta_1 & \theta_2 & \cdots & \theta_{D-1} & \theta_D \end{bmatrix} \begin{bmatrix} 1 \\ x \\ x^2 \\ \cdots \\ x^{D-1} \\ x^D \end{bmatrix}$$

Scenario 1: Find optimal parameters to fit a 1D curve

- Model f is D-degree polynomial parametrised by $\boldsymbol{\theta}$, or $f_{\boldsymbol{\theta}}$

$$f_{\boldsymbol{\theta}}(x) = \theta_0 x^0 + \theta_1 x^1 + \theta_2 x^2 + \cdots + \theta_{D-1} x^{D-1} + \theta_D x^D$$

$$= \begin{bmatrix} \theta_0 & \theta_1 & \theta_2 & \cdots & \theta_{D-1} & \theta_D \end{bmatrix} \begin{bmatrix} 1 \\ x \\ x^2 \\ \cdots \\ x^{D-1} \\ x^D \end{bmatrix}$$

$$= \boldsymbol{\theta}^\top \boldsymbol{x} \qquad \boldsymbol{x}, \boldsymbol{\theta} \in \mathbb{R}^{D+1}$$

Scenario 1: Find optimal parameters to fit a 1D curve

- Model f is D-degree polynomial parametrised by $\boldsymbol{\theta}$, or $f_{\boldsymbol{\theta}}$

$$f_{\boldsymbol{\theta}}(x) = \theta_0 x^0 + \theta_1 x^1 + \theta_2 x^2 + \cdots + \theta_{D-1} x^{D-1} + \theta_D x^D$$

$$= \begin{bmatrix} \theta_0 & \theta_1 & \theta_2 & \cdots & \theta_{D-1} & \theta_D \end{bmatrix} \begin{bmatrix} 1 \\ x \\ x^2 \\ \cdots \\ x^{D-1} \\ x^D \end{bmatrix}$$

$$= \boldsymbol{\theta}^\top \boldsymbol{x} \qquad \boldsymbol{x}, \boldsymbol{\theta} \in \mathbb{R}^{D+1}$$

- Training data: N pairs (x_i, y_i)
- Training the model means finding parameters $\boldsymbol{\theta}^*$, such that

$$f_{\boldsymbol{\theta}^*}(x_i) \approx y_i$$

Scenario 1: Find optimal parameters to fit a 1D curve

- Model f is D-dimensional polynomial parametrised by $\boldsymbol{\theta}$, or $f_{\boldsymbol{\theta}}$

$$f_{\boldsymbol{\theta}}(x) = \theta_0 x^0 + \theta_1 x^1 + \theta_2 x^2 + \cdots + \theta_{D-1} x^{D-1} + \theta_D x^D$$

$$= \begin{bmatrix} \theta_0 & \theta_1 & \theta_2 & \cdots & \theta_{D-1} & \theta_D \end{bmatrix} \begin{bmatrix} 1 \\ x \\ x^2 \\ \cdots \\ x^{D-1} \\ x^D \end{bmatrix}$$

$$= \boldsymbol{\theta}^\top \boldsymbol{x} \qquad \boldsymbol{x}, \boldsymbol{\theta} \in \mathbb{R}^{D+1}$$

- Define a loss function, $L_{\boldsymbol{\theta}}(x, y) = \sum_{i=1}^{N} (y_i - f_{\boldsymbol{\theta}}(x_i))^2$
- Minimisation of loss function is typically based on gradient descent

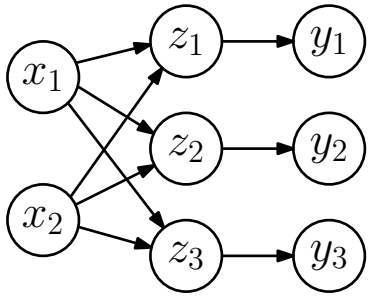
  ⏩ **Differentiation** required to compute gradients of $L_{\boldsymbol{\theta}}$ w.r.t $\boldsymbol{\theta}$

Scenario 2: Find optimal parameters of a Neural Network

$$\boldsymbol{x} \in \mathbb{R}^2, \boldsymbol{A} \in \mathbb{R}^{3 \times 2}, \boldsymbol{b} \in \mathbb{R}^3$$

$$\boldsymbol{z} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b} \in \mathbb{R}^3$$

$$\boldsymbol{y} = \sigma(\boldsymbol{z}) \in \mathbb{R}^3$$



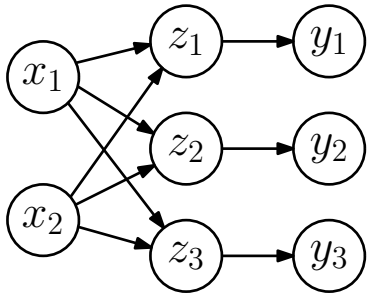
Scenario 2: Find optimal parameters of a Neural Network

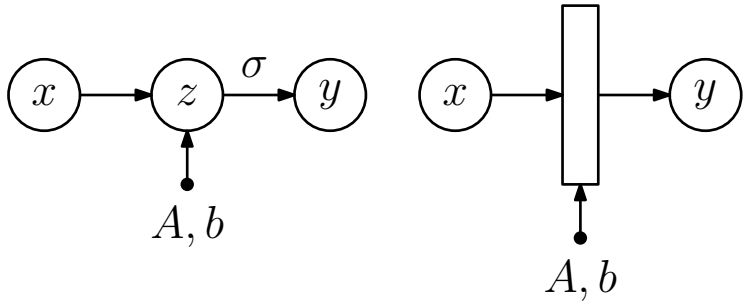
$$\boldsymbol{x} \in \mathbb{R}^2, \boldsymbol{A} \in \mathbb{R}^{3 \times 2}, \boldsymbol{b} \in \mathbb{R}^3$$

$$\boldsymbol{z} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b} \in \mathbb{R}^3$$

$$\boldsymbol{y} = \sigma(\boldsymbol{z}) \in \mathbb{R}^3$$

$$\boldsymbol{y} = f_{\boldsymbol{A},\boldsymbol{b}}(\boldsymbol{x})$$





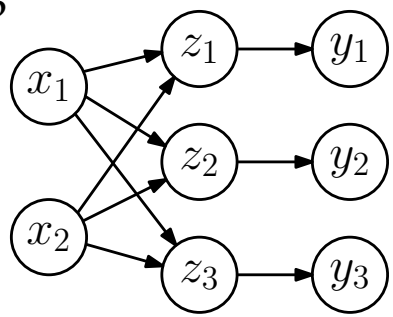
Scenario 2: Find optimal parameters of a Neural Network

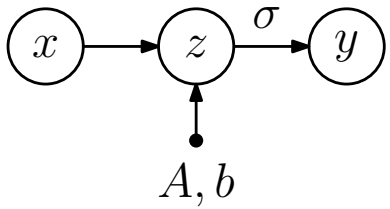
$$\boldsymbol{x} \in \mathbb{R}^2, \boldsymbol{A} \in \mathbb{R}^{3 \times 2}, \boldsymbol{b} \in \mathbb{R}^3$$

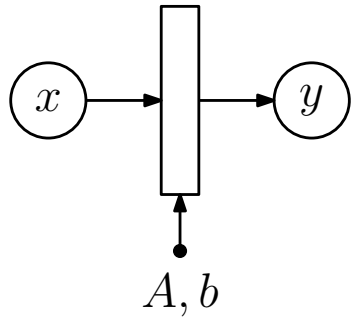
$$\boldsymbol{z} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b} \in \mathbb{R}^3$$

$$\boldsymbol{y} = \sigma(\boldsymbol{z}) \in \mathbb{R}^3$$

$$\boldsymbol{y} = f_{\boldsymbol{A},\boldsymbol{b}}(\boldsymbol{x})$$







- Training data: N pairs $(\boldsymbol{x}_i, \boldsymbol{y}_i)$
- Training the NN means finding parameters $\boldsymbol{A}^*, \boldsymbol{b}^*$, such that

$$f_{\boldsymbol{A}^*,\boldsymbol{b}^*}(\boldsymbol{x}_i) \approx \boldsymbol{y}_i$$

Similarly, parameters A, b are tuned to minimize the prediction error through gradient descent

Again, differentiation is used to compute gradients of the loss function with regard to matrix A and b

Rules

- Sum Rule

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

Rules

- Sum Rule

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

- Product Rule

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

Rules

- Sum Rule

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

- Product Rule

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

- Chain Rule

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

Rules

- Sum Rule

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

- Product Rule

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

- Chain Rule

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

- Quotient Rule

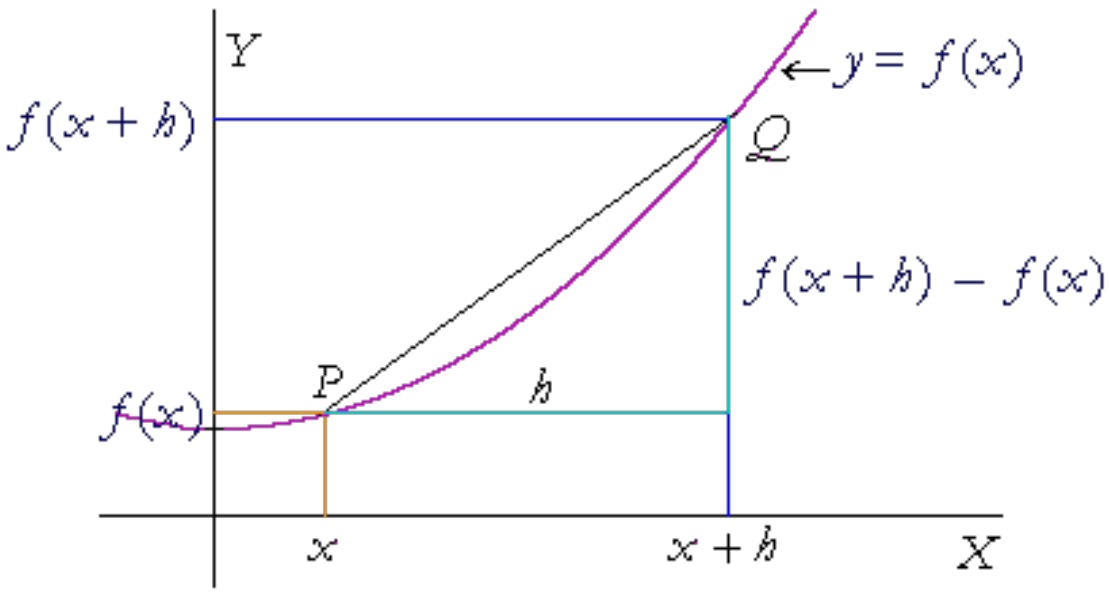
$$\left(\frac{f(x)}{g(x)}\right)' = \frac{f(x)'g(x)-f(x)g(x)'}{(g(x))^2} = \frac{\frac{df}{dx}g(x)-f(x)\frac{dg}{dx}}{(g(x))^2}$$

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

- Derivative defined as the limit of the difference quotient

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

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



Example: Scalar Chain Rule

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

Beginner

$$\begin{aligned} g(z) &= 6z + 3 \\ z &= f(x) = -2x + 5 \\ (g \circ f)'(x) &= \underbrace{(6)}_{dg/df}\underbrace{(-2)}_{df/dx} \\ &= -12 \end{aligned}$$

Advanced

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

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

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

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

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

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

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

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

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

- Jacobian vector (gradient) collects all partial derivatives:

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

Note: This is a row vector.

Example: Multivariate differentiation

Beginner

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

Advanced

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

Partial derivatives?

Work it out with your neighbours

Example: Multivariate differentiation

Beginner

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

Advanced

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

Partial derivatives

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

$$\frac{\partial f(x_1, x_2)}{\partial x_1} = 2(x_1 + 2x_2^3) \overbrace{(1)}^{\frac{d}{dx_1}(x_1+2x_2^3)}$$
$$\frac{\partial f(x_1, x_2)}{\partial x_2} = 2(x_1 + 2x_2^3) \underbrace{(6x_2^2)}_{\frac{d}{dx_2}(x_1+2x_2^3)}$$

Example: Multivariate Chain Rule

- Consider the function

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

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

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

Example: Multivariate Chain Rule

- Consider the function

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

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

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

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

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

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

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

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

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

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

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

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

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

- Jacobian matrix (collection of all partial derivatives)

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

Example: Vector field differentiation

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

$$\begin{bmatrix} y_1 \\ \vdots \\ y_M \end{bmatrix} = \begin{bmatrix} f_1(\boldsymbol{x}) \\ \vdots \\ f_M(\boldsymbol{x}) \end{bmatrix} = \begin{bmatrix} A_{11}x_1 + A_{12}x_2 + & \cdots & + A_{1(N-1)}x_{N-1} + A_{1N}x_N \\ \vdots & \vdots & \vdots \\ A_{M1}x_1 + A_{M2}x_2 + & \cdots & + A_{M(N-1)}x_{N-1} + A_{MN}x_N \end{bmatrix}$$

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

Example: Vector field differentiation

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

$$\begin{bmatrix} y_1 \\ \vdots \\ y_M \end{bmatrix} = \begin{bmatrix} f_1(\boldsymbol{x}) \\ \vdots \\ f_M(\boldsymbol{x}) \end{bmatrix} = \begin{bmatrix} A_{11}x_1 + A_{12}x_2 + & \cdots & + A_{1(N-1)}x_{N-1} + A_{1N}x_N \\ \vdots & \vdots & \vdots \\ A_{M1}x_1 + A_{M2}x_2 + & \cdots & + A_{M(N-1)}x_{N-1} + A_{MN}x_N \end{bmatrix}$$

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

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

What about df/dA? Dimension?

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

Derivatives with respect to Matrices

- Recall: function $f : \mathbb{R}^D \to \mathbb{R}^E$ has a gradient that is $E \times D$-matrix

  # target dimensions $\times$ # input dimensions

  with

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

Derivatives with respect to Matrices

- Recall: function $f : \mathbb{R}^D \to \mathbb{R}^E$ has a gradient that is $E \times D$-matrix

  $$\#\text{ target dimensions} \times \#\text{ input dimensions}$$

  with

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

- This generalises to when the inputs (D) or targets (E) are **matrices**

Derivatives with respect to Matrices

- Recall: function $f : \mathbb{R}^D \to \mathbb{R}^E$ has a gradient that is $E \times D$-matrix

  # target dimensions $\times$ # input dimensions

  with

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

- This generalises to when the inputs (D) or targets (E) are **matrices**
- Function $f : \mathbb{R}^{M \times N} \to \mathbb{R}^{P \times Q}$, has a gradient that is a $(P \times Q) \times (M \times N)$ object (tensor)

$$\frac{df}{dX} \in \mathbb{R}^{(P \times Q) \times (M \times N)}, \qquad df[p,q,m,n] = \frac{\partial f_{pq}}{\partial X_{mn}}$$

Example 1: Derivatives w.r.t Matrices

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

$$\begin{bmatrix} y_1 \\ \vdots \\ y_M \end{bmatrix} = \begin{bmatrix} f_1(\boldsymbol{x}) \\ \vdots \\ f_M(\boldsymbol{x}) \end{bmatrix} = \begin{bmatrix} A_{11}x_1 + A_{12}x_2 + & \cdots & + A_{1(N-1)}x_{N-1} + A_{1N}x_N \\ \vdots & \vdots & \vdots \\ A_{M1}x_1 + A_{M2}x_2 + & \cdots & + A_{M(N-1)}x_{N-1} + A_{MN}x_N \end{bmatrix}$$

$$\frac{d\boldsymbol{f}}{d\boldsymbol{A}} \in \mathbb{R}^{?}$$

Example 1: Derivatives w.r.t Matrices

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

$$\begin{bmatrix} y_1 \\ \vdots \\ y_M \end{bmatrix} = \begin{bmatrix} f_1(\boldsymbol{x}) \\ \vdots \\ f_M(\boldsymbol{x}) \end{bmatrix} = \begin{bmatrix} A_{11}x_1 + A_{12}x_2 + & \cdots & + A_{1(N-1)}x_{N-1} + A_{1N}x_N \\ \vdots & \vdots & \vdots \\ A_{M1}x_1 + A_{M2}x_2 + & \cdots & + A_{M(N-1)}x_{N-1} + A_{MN}x_N \end{bmatrix}$$

$$\frac{d\boldsymbol{f}}{d\boldsymbol{A}} \in \mathbb{R}^{\#\text{ target dim} \times \#\text{ input dim}} = M \times (M \times N)$$

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

Example 2: Derivatives w.r.t Matrices

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

$$\begin{bmatrix} y_1 \\ \vdots \\ y_i \\ \vdots \\ y_M \end{bmatrix} = \begin{bmatrix} f_1(\boldsymbol{x}) \\ \vdots \\ f_i(\boldsymbol{x}) \\ \vdots \\ f_M(\boldsymbol{x}) \end{bmatrix} = \begin{bmatrix} A_{11}x_1 + A_{12}x_2 + & \cdots & + A_{1(N-1)}x_{N-1} + A_{1N}x_N \\ \vdots & \vdots & \vdots \\ A_{i1}x_1 + A_{i2}x_2 & \cdots & + A_{i(N-1)}x_{N-1} + A_{iN}x_N \\ \vdots & \vdots & \vdots \\ A_{M1}x_1 + A_{M2}x_2 + & \cdots & + A_{M(N-1)}x_{N-1} + A_{MN}x_N \end{bmatrix}$$

$$\frac{\partial f_i}{\partial A_{iq}} = ? \qquad \frac{\partial f_i}{\partial A_{i,:}} = ? \qquad \frac{\partial f_i}{\partial A_{k\neq i,:}} = ? \qquad \frac{\partial f_i}{\partial \boldsymbol{A}} = ?$$

Example 2: Derivatives w.r.t Matrices

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

$$\begin{bmatrix} y_1 \\ \vdots \\ y_i \\ \vdots \\ y_M \end{bmatrix} = \begin{bmatrix} f_1(\boldsymbol{x}) \\ \vdots \\ f_i(\boldsymbol{x}) \\ \vdots \\ f_M(\boldsymbol{x}) \end{bmatrix} = \begin{bmatrix} A_{11}x_1 + A_{12}x_2 + & \cdots & + A_{1(N-1)}x_{N-1} + A_{1N}x_N \\ \vdots & \vdots & \vdots \\ A_{i1}x_1 + A_{i2}x_2 & \cdots & + A_{i(N-1)}x_{N-1} + A_{iN}x_N \\ \vdots & \vdots & \vdots \\ A_{M1}x_1 + A_{M2}x_2 + & \cdots & + A_{M(N-1)}x_{N-1} + A_{MN}x_N \end{bmatrix}$$

$$\frac{\partial f_i}{\partial A_{iq}} = \underbrace{x_q}_{\in \mathbb{R}} \qquad \frac{\partial f_i}{\partial A_{i,:}} = ? \qquad \frac{\partial f_i}{\partial A_{k \neq i,:}} = ? \qquad \frac{\partial f_i}{\partial \boldsymbol{A}} = ?$$

Example 2: Derivatives w.r.t Matrices

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

$$\begin{bmatrix} y_1 \\ \vdots \\ y_i \\ \vdots \\ y_M \end{bmatrix} = \begin{bmatrix} f_1(\boldsymbol{x}) \\ \vdots \\ f_i(\boldsymbol{x}) \\ \vdots \\ f_M(\boldsymbol{x}) \end{bmatrix} = \begin{bmatrix} A_{11}x_1 + A_{12}x_2 + & \cdots & + A_{1(N-1)}x_{N-1} + A_{1N}x_N \\ \vdots & \vdots & \vdots \\ A_{i1}x_1 + A_{i2}x_2 & \cdots & + A_{i(N-1)}x_{N-1} + A_{iN}x_N \\ \vdots & \vdots & \vdots \\ A_{M1}x_1 + A_{M2}x_2 + & \cdots & + A_{M(N-1)}x_{N-1} + A_{MN}x_N \end{bmatrix}$$

$$\frac{\partial f_i}{\partial A_{iq}} = \underbrace{x_q}_{\in \mathbb{R}} \qquad \frac{\partial f_i}{\partial A_{i,:}} = \underbrace{\boldsymbol{x}^\top}_{\in \mathbb{R}^{1 \times 1 \times N}} \qquad \frac{\partial f_i}{\partial A_{k \neq i,:}} = ? \qquad \frac{\partial f_i}{\partial \boldsymbol{A}} = ?$$

Example 2: Derivatives w.r.t Matrices

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

$$\begin{bmatrix} y_1 \\ \vdots \\ y_i \\ \vdots \\ y_M \end{bmatrix} = \begin{bmatrix} f_1(\boldsymbol{x}) \\ \vdots \\ f_i(\boldsymbol{x}) \\ \vdots \\ f_M(\boldsymbol{x}) \end{bmatrix} = \begin{bmatrix} A_{11}x_1 + A_{12}x_2 + & \cdots & +A_{1(N-1)}x_{N-1} + A_{1N}x_N \\ \vdots & \vdots & \vdots \\ A_{i1}x_1 + A_{i2}x_2 & \cdots & +A_{i(N-1)}x_{N-1} + A_{iN}x_N \\ \vdots & \vdots & \vdots \\ A_{M1}x_1 + A_{M2}x_2 + & \cdots & +A_{M(N-1)}x_{N-1} + A_{MN}x_N \end{bmatrix}$$

$$\frac{\partial f_i}{\partial A_{iq}} = \underbrace{x_q}_{\in \mathbb{R}} \qquad \frac{\partial f_i}{\partial A_{i,:}} = \underbrace{\boldsymbol{x}^\top}_{\in \mathbb{R}^{1\times 1\times N}} \qquad \frac{\partial f_i}{\partial A_{k\neq i,:}} = \underbrace{\boldsymbol{0}^\top}_{\in \mathbb{R}^{1\times 1\times N}} \qquad \frac{\partial f_i}{\partial \boldsymbol{A}} = ?$$

Example 2: Derivatives w.r.t Matrices

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

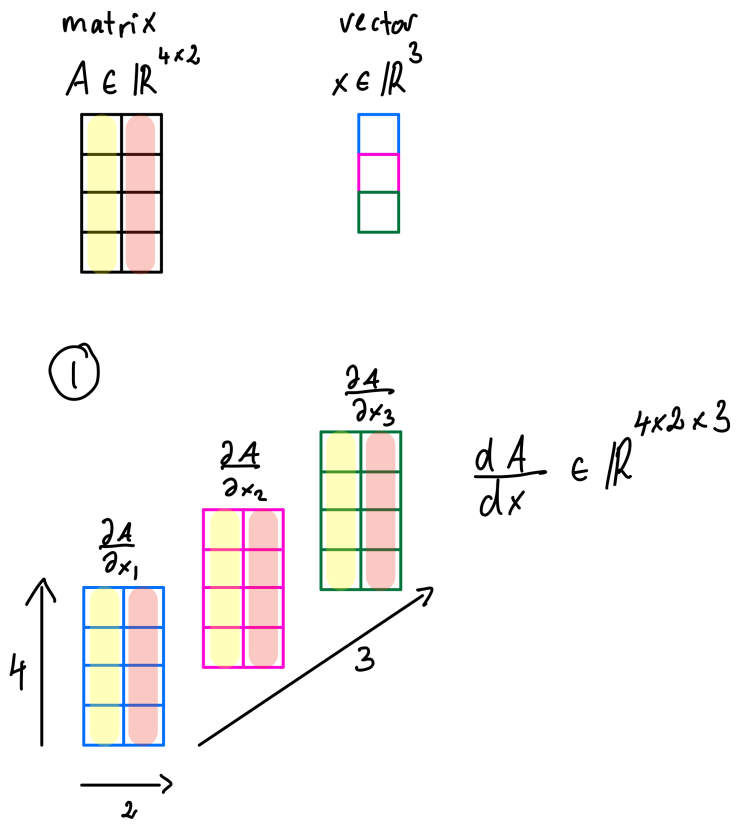
$$\begin{bmatrix} y_1 \\ \vdots \\ y_i \\ \vdots \\ y_M \end{bmatrix} = \begin{bmatrix} f_1(\boldsymbol{x}) \\ \vdots \\ f_i(\boldsymbol{x}) \\ \vdots \\ f_M(\boldsymbol{x}) \end{bmatrix} = \begin{bmatrix} A_{11}x_1 + A_{12}x_2 + & \cdots & + A_{1(N-1)}x_{N-1} + A_{1N}x_N \\ \vdots & \vdots & \vdots \\ A_{i1}x_1 + A_{i2}x_2 & \cdots & + A_{i(N-1)}x_{N-1} + A_{iN}x_N \\ \vdots & \vdots & \vdots \\ A_{M1}x_1 + A_{M2}x_2 + & \cdots & + A_{M(N-1)}x_{N-1} + A_{MN}x_N \end{bmatrix}$$

$$\frac{\partial f_i}{\partial A_{iq}} = \underbrace{x_q}_{\in \mathbb{R}} \qquad \frac{\partial f_i}{\partial A_{i,:}} = \underbrace{\boldsymbol{x}^\top}_{\in \mathbb{R}^{1 \times 1 \times N}} \qquad \frac{\partial f_i}{\partial A_{k \neq i,:}} = \underbrace{\mathbf{0}^\top}_{\in \mathbb{R}^{1 \times 1 \times N}}$$

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

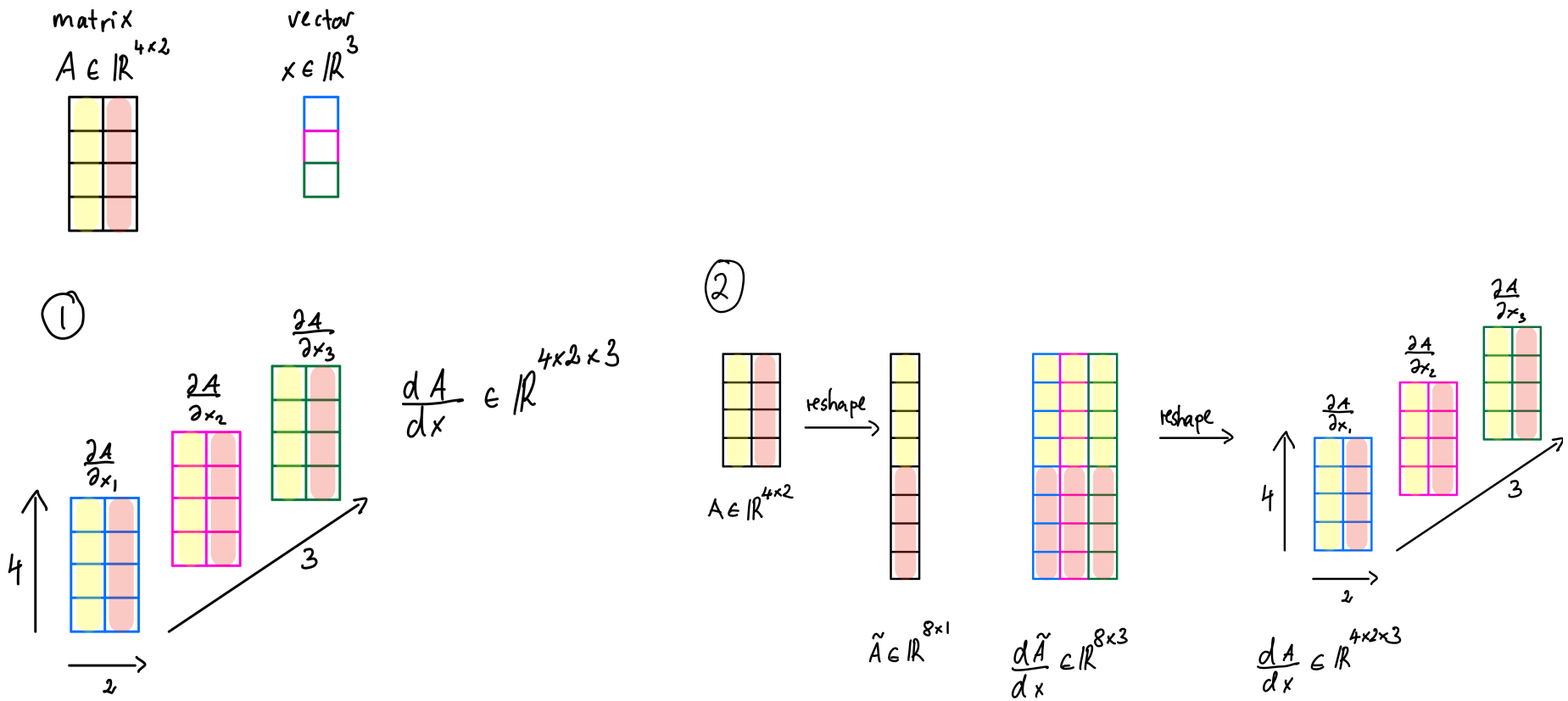
Example 3: Derivatives w.r.t Higher-Order Tensors

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

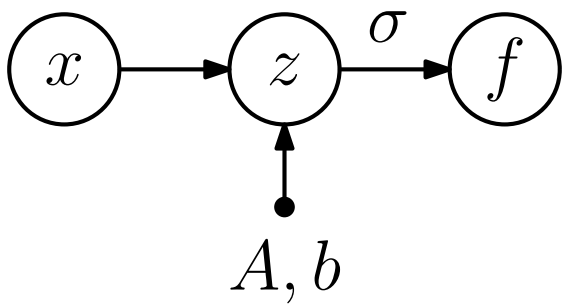


Example 3: Derivatives w.r.t Higher-Order Tensors

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



Gradients of a Single-Layer Neural Network



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

Gradients of a Single-Layer Neural Network

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

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{b}} =$$

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{A}} =$$

Gradients of a Single-Layer Neural Network

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

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

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{A}} =$$

Gradients of a Single-Layer Neural Network

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

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

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

Gradients of a Single-Layer Neural Network

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

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

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

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

Gradients of a Single-Layer Neural Network

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

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

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

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

Gradients of a Single-Layer Neural Network

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

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

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

$$\frac{\partial f}{\partial \boldsymbol{z}} = \underbrace{\operatorname{diag}(1 - \tanh^2(\boldsymbol{z}))}_{\in\mathbb{R}^{M\times M}} \quad \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}} = \underbrace{\boldsymbol{I}}_{\in\mathbb{R}^{M\times M}} \quad \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{A}} = \underbrace{\begin{bmatrix} \boldsymbol{x}^\top & \cdot & \boldsymbol{0}^\top & \cdot & \boldsymbol{0}^\top \\ \cdot & & \cdot & & \cdot \\ \boldsymbol{0}^\top & \cdot & \boldsymbol{x}^\top & \cdot & \boldsymbol{0}^\top \\ \cdot & & \cdot & & \cdot \\ \boldsymbol{0}^\top & \cdot & \boldsymbol{0}^\top & \cdot & \boldsymbol{x}^\top \end{bmatrix}}_{\in\mathbb{R}^{M\times(M\times N)}}$$

Putting Things Together

- Inputs $\boldsymbol{x} \in \mathbb{R}^N$

Putting Things Together

- Inputs $\boldsymbol{x} \in \mathbb{R}^N$
- Observed outputs $\boldsymbol{y} = f_{\boldsymbol{\theta}}(\boldsymbol{z}) + \boldsymbol{\epsilon} \in \mathbb{R}^M, \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$

Putting Things Together

- Inputs $\boldsymbol{x} \in \mathbb{R}^N$
- Observed outputs $\boldsymbol{y} = f_{\boldsymbol{\theta}}(\boldsymbol{z}) + \boldsymbol{\epsilon} \in \mathbb{R}^M, \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$
- Train single-layer neural network with

$$f_{\boldsymbol{\theta}}(\boldsymbol{z}) = \tanh(\boldsymbol{z}) \in \mathbb{R}^M, \quad \boldsymbol{z} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b} \in \mathbb{R}^M, \quad \boldsymbol{\theta} = \{\boldsymbol{A}, \boldsymbol{b}\}$$

Putting Things Together

- Inputs $\boldsymbol{x} \in \mathbb{R}^N$
- Observed outputs $\boldsymbol{y} = f_{\boldsymbol{\theta}}(\boldsymbol{z}) + \boldsymbol{\epsilon} \in \mathbb{R}^M, \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$
- Train single-layer neural network with

$$f_{\boldsymbol{\theta}}(\boldsymbol{z}) = \tanh(\boldsymbol{z}) \in \mathbb{R}^M, \quad \boldsymbol{z} = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b} \in \mathbb{R}^M, \quad \boldsymbol{\theta} = \{\boldsymbol{A}, \boldsymbol{b}\}$$

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

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

  is minimised

Putting Things Together

Partial derivatives:

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

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

$$\frac{\partial L}{\partial \boldsymbol{e}} = \underbrace{\boldsymbol{e}^\top}_{\in \mathbb{R}^{1\times M}} \qquad \frac{\partial \boldsymbol{e}}{\partial \boldsymbol{f}} = \underbrace{-\boldsymbol{I}}_{\in \mathbb{R}^{M\times M}} \qquad \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}} = \underbrace{\operatorname{diag}(1-\tanh^2(\boldsymbol{z}))}_{\in \mathbb{R}^{M\times M}}$$

$$\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{A}} = \underbrace{\begin{bmatrix} \boldsymbol{x}^\top & \cdot & \boldsymbol{0}^\top & \cdot & \boldsymbol{0}^\top \\ \cdot & & \cdot & & \cdot \\ \boldsymbol{0}^\top & \cdot & \boldsymbol{x}^\top & \cdot & \boldsymbol{0}^\top \\ \cdot & & \cdot & & \cdot \\ \boldsymbol{0}^\top & \cdot & \boldsymbol{0}^\top & \cdot & \boldsymbol{x}^\top \end{bmatrix}}_{\in \mathbb{R}^{M\times(M\times N)}} \qquad \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}} = \underbrace{\boldsymbol{I}}_{\in \mathbb{R}^{M\times M}}$$

Gradient descent for neural network optimisation

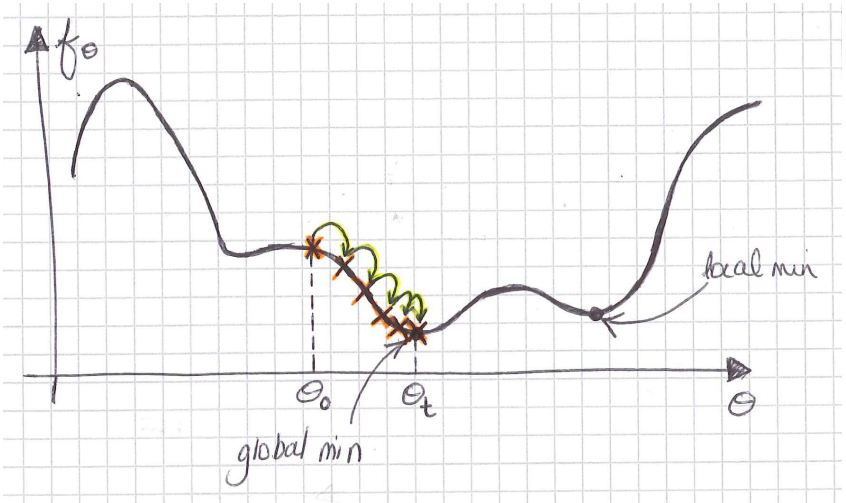
Goal: Find the local optimum of loss $L(\boldsymbol{\theta}^*) \in \mathbb{R}$ where $\boldsymbol{\theta}^* = \{\boldsymbol{A}^*, \boldsymbol{b}^*\}$

Gradient descent algorithm:

1. Initialise neural network parameters, $\boldsymbol{\theta}_0 = \{\boldsymbol{A}_0, \boldsymbol{b}_0\}$
2. Compute gradient of L w.r.t. parameters, $\frac{\partial L}{\partial \boldsymbol{\theta}_0}$
3. Update initial guess $\boldsymbol{\theta}_0$ using chain rule:

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i + \Delta\boldsymbol{\theta}_i = \boldsymbol{\theta}_i - \gamma(\frac{\partial L}{\partial \boldsymbol{\theta}_i}(\boldsymbol{\theta}_i))^\top$$

- Note: parameter update is the direction of steepest **de**scent



Gradient descent for neural network optimisation

Goal: Find the local optimum of loss $L(\boldsymbol{\theta}^*) \in \mathbb{R}$ where $\boldsymbol{\theta}^* = \{\boldsymbol{A}^*, \boldsymbol{b}^*\}$

Gradient descent algorithm:

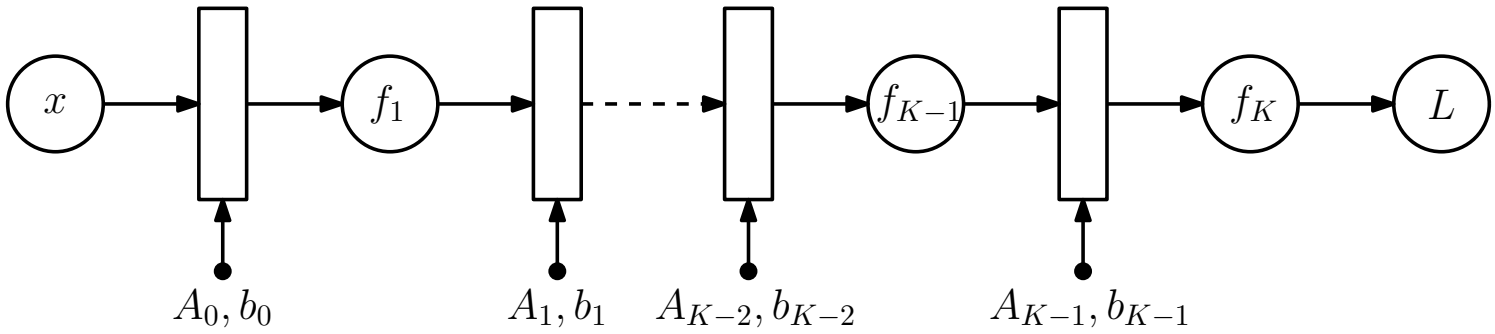
1. Initialise neural network parameters, $\boldsymbol{\theta}_0 = \{\boldsymbol{A}_0, \boldsymbol{b}_0\}$
2. Compute gradient of L w.r.t. parameters, $\frac{\partial L}{\partial \boldsymbol{\theta}_0}$
3. Update initial guess $\boldsymbol{\theta}_0$ using chain rule:

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i + \Delta\boldsymbol{\theta}_i = \boldsymbol{\theta}_i - \gamma(\frac{\partial L}{\partial \boldsymbol{\theta}_i}(\boldsymbol{\theta}_i))^\top$$

- Note: parameter update is the direction of steepest **de**scent

For suitable step-size γ, the sequence $L(\boldsymbol{\theta}_0) \geqslant L(\boldsymbol{\theta}_1) \geqslant \cdots$ converges to a local minimum $L(\boldsymbol{\theta}^*)$.

Gradients of a Multi-Layer Neural Network

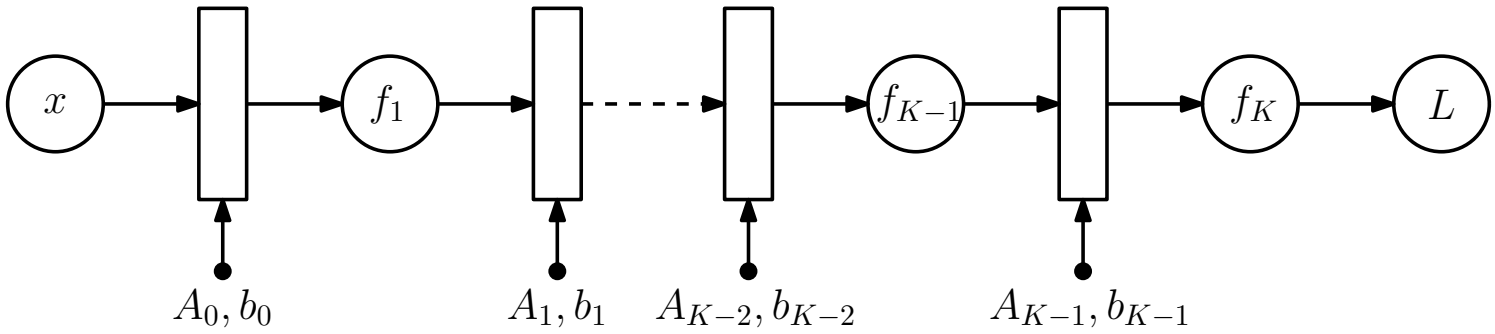


- Inputs $\boldsymbol{x}$, observed outputs $\boldsymbol{y}$
- Train K-layer neural network with

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

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

Gradients of a Multi-Layer Neural Network



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

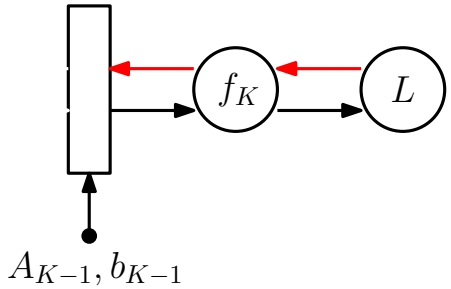
$$\boldsymbol{f}_0 = \boldsymbol{x}$$
$$\boldsymbol{f}_i = \sigma_i(\boldsymbol{A}_{i-1}\boldsymbol{f}_{i-1} + \boldsymbol{b}_{i-1}), \quad i = 1, \ldots, K$$

- Find $\boldsymbol{A}_k^*, \boldsymbol{b}_k^*$ for $k = 0, \ldots, K-1$, such that the squared loss

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

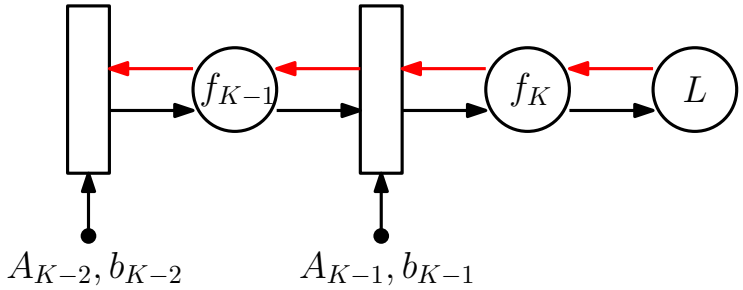
is minimised, where $\boldsymbol{\theta} = \{\boldsymbol{A}_k, \boldsymbol{b}_k\}, \quad k = 0, \ldots, K-1$

Gradients of a Multi-Layer Neural Network



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

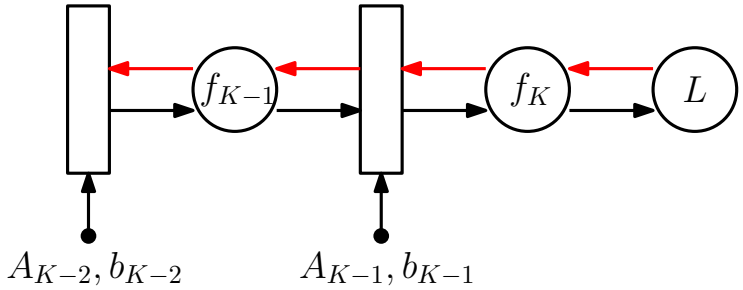
Gradients of a Multi-Layer Neural Network



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

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

Gradients of a Multi-Layer Neural Network

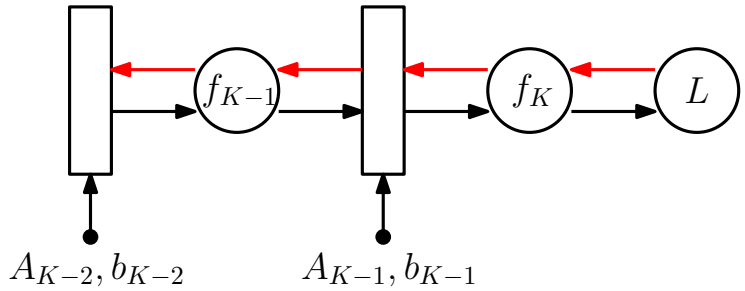


$$\frac{\partial L}{\partial \boldsymbol{\theta}_{K-1}} = \frac{\partial L}{\partial \boldsymbol{f}_K} \frac{\partial \boldsymbol{f}_K}{\partial \boldsymbol{\theta}_{K-1}}$$

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

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

Gradients of a Multi-Layer Neural Network



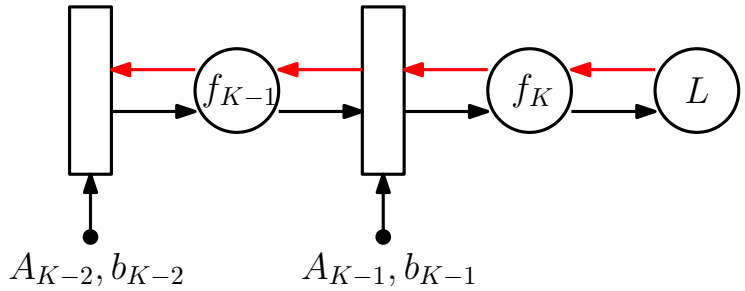
$$\frac{\partial L}{\partial \boldsymbol{\theta}_{K-1}} = \frac{\partial L}{\partial \boldsymbol{f}_K} \frac{\partial \boldsymbol{f}_K}{\partial \boldsymbol{\theta}_{K-1}}$$

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

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

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

Gradients of a Multi-Layer Neural Network



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

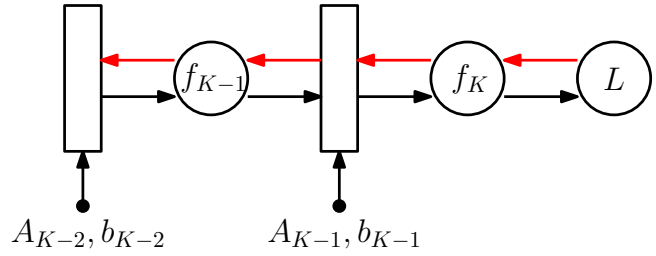
$$\frac{\partial L}{\partial \boldsymbol{\theta}_{K-2}} = \frac{\partial L}{\partial \boldsymbol{f}_K} \boxed{\frac{\partial \boldsymbol{f}_K}{\partial \boldsymbol{f}_{K-1}} \frac{\partial \boldsymbol{f}_{K-1}}{\partial \boldsymbol{\theta}_{K-2}}}$$

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

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

- Intermediate derivatives are stored during the forward pass

Gradients of a Multi-Layer Neural Network



$\boldsymbol{\delta_K}$

$$\frac{\partial L}{\partial \boldsymbol{\theta}_{K-1}} = \frac{\partial L}{\partial \boldsymbol{f}_K}\frac{\partial \boldsymbol{f}_K}{\partial \boldsymbol{\theta}_{K-1}} \Longrightarrow \delta_K = \frac{\partial L}{\partial f_K}$$

$$\frac{\partial L}{\partial \boldsymbol{\theta}_{K-2}} = \frac{\partial L}{\partial \boldsymbol{f}_K}\frac{\partial \boldsymbol{f}_K}{\partial \boldsymbol{f}_{K-1}}\frac{\partial \boldsymbol{f}_{K-1}}{\partial \boldsymbol{\theta}_{K-2}} \Longrightarrow \delta_{K-1} = \delta_K \frac{\partial f_K}{\partial f_{K-1}}$$

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

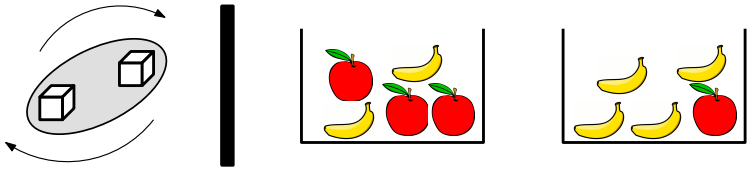
$$\frac{\partial L}{\partial \boldsymbol{\theta}_i} = \frac{\partial L}{\partial \boldsymbol{f}_K}\frac{\partial \boldsymbol{f}_K}{\partial \boldsymbol{f}_{K-1}}\cdots\frac{\partial \boldsymbol{f}_{i+2}}{\partial \boldsymbol{f}_{i+1}}\frac{\partial \boldsymbol{f}_{i+1}}{\partial \boldsymbol{\theta}_i} \Longrightarrow \delta_{i+1} = \delta_{i+2}\frac{\partial f_{i+2}}{\partial f_{i+1}}$$

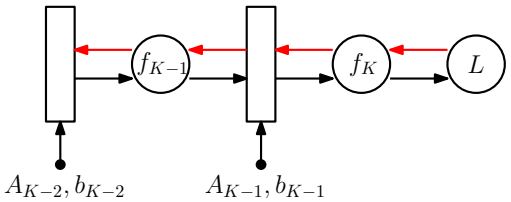
Backpropagation

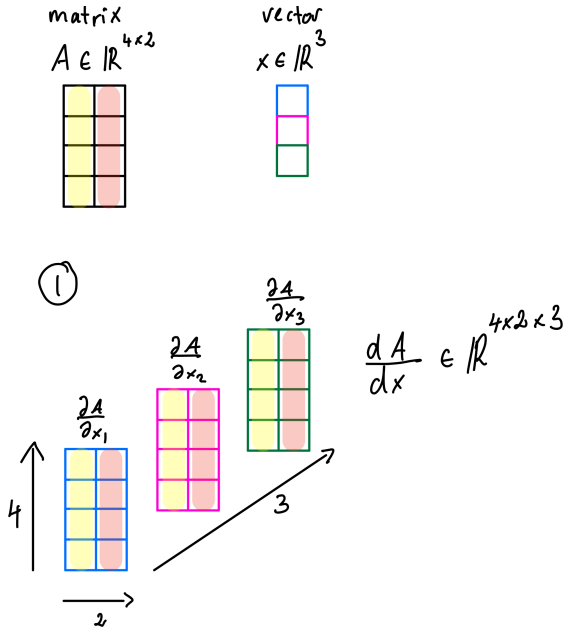
Backpropagate δ from K to $K-1, K-2, \ldots, i, \ldots$

How to calculate gradients:
1. Given x and θ, first do a forward pass to calculate L and f at each layer
2. Then follow these equations to get partial derivatives of each θ

Summary







- Key concepts in probability - PDFs, Sum & Product Rules, Bayes
- **Differentiation** for optimising parameters of ML models
  - Vector calculus
  - Chain rule
  - Single- & Multi-layer NNs
  - Maximum Likelihood Estimation Equivalence

Useful Resources

- Indaba 2017 slides/videos
- Textbooks:
  - Mathematics for ML - Marc Deisenroth, Aldo Faisal, Cheng Soon Ong
  - Deep Learning - Yoshua Bengio, Ian Goodfellow, Aaron Courville
  - Matrix Cookbook - Kaare Petersen, Michael Pedersen
- Online courses:
  - fast.ai Deep Learning by Jeremy Howard
  - deeplearning.ai/coursera Deep Learning Specialisation by Andrew Ng
  - Udacity Machine Learning Nanodegree Programs
  - Coursera Machine Learning by Stanford University
- Blogs, podcasts:
  - Reddit ML, KDnuggets, TWiML&AI, Talking Machines

More textbooks, online courses, papers, datasets and other resources at `github.com/ChristosChristofidis/awesome-deep-learning`

Chain Rule

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

Example: Combined Chain Rule

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

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

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

Example: Combined Chain Rule

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

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

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

- What are the dimensions of $df/d\boldsymbol{x}$ and $d\boldsymbol{x}/dt$?

Example: Combined Chain Rule

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

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

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

- What are the dimensions of $df/d\boldsymbol{x}$ and $d\boldsymbol{x}/dt$?

  1×2 and 2×1

- Compute the gradient df/dt using the chain rule:

  Work it out with your neighbours

Example: Combined Chain Rule

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

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

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

- What are the dimensions of $df/d\boldsymbol{x}$ and $d\boldsymbol{x}/dt$?

1×2 and 2×1

- Compute the gradient df/dt using the chain rule:

$$\frac{df}{dt} = \frac{df}{d\boldsymbol{x}}\frac{d\boldsymbol{x}}{dt} = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} \end{bmatrix} \begin{bmatrix} \frac{\partial x_1}{\partial t} \\ \frac{\partial x_2}{\partial t} \end{bmatrix} = \begin{bmatrix} 2\sin t & 2 \end{bmatrix} \begin{bmatrix} \cos t \\ -\sin t \end{bmatrix}$$

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

```
import numpy as np
def NN(x, A, b):
    M, N = A.shape
    z = A.dot(x) + b
    f = np.tanh(z)
    # partial derivatives
    dfdz = 1-f**2
    dzdx = A
    dzdb = np.eye(M)
    dzdA = np.zeros((M, M, N))
    for i in range(M):
        dzdA[i,i,:]  = x.T
    # gradients
    dfdx = np.einsum('il, lj', dfdz, dzdx)
    dfdb = np.einsum('il, lj', dfdz, dzdb)
    dfdA = np.einsum('il, ljk', dfdz, dzdA)
    return f, dfdA, dfdb, dfdx
```

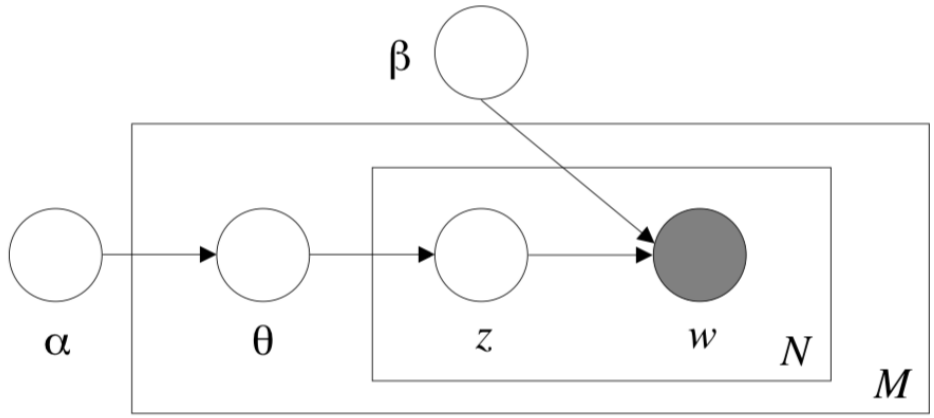
$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}[i,j] = \sum_{l=1}^{M} \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}[i,l] \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{x}}[l,j]$$

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

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

Structured Probabilistic Model of LDA

- Remember this?



Graphical model representation of LDA. The boxes are "plates" representing replicates. The outer plate represents documents, while the inner plate represents the repeated choice of topics and words within a document.

$$p(\theta, \mathbf{z}, \mathbf{w} \mid \alpha, \beta) = p(\theta \mid \alpha) \prod_{n=1}^{N} p(z_n \mid \theta) p(w_n \mid z_n, \beta)$$

LDA assumes the following generative process for each document $\mathbf{w}$ in a corpus D:

1. Choose $N \sim \text{Poisson}(\xi)$.
2. Choose $\theta \sim \text{Dir}(\alpha)$.
3. For each of the N words w_n:
   (a) Choose a topic $z_n \sim \text{Multinomial}(\theta)$.
   (b) Choose a word w_n from $p(w_n \mid z_n, \beta)$, a multinomial probability conditioned on the topic z_n.

Note: β is the word probabilities per topic, a $k \times V$ matrix, $\beta_{ij} = p(w_j = 1 \mid z_i = 1)$. β is a parameter to be estimated.