

[KICSV Special AI Lecture]
Mathematics for AI - Theory into Practice

Sunghee Yun

Co-Founder & CTO @ Erudio Bio, Inc. / Advisor @ CryptoLab, Inc.
Adjunct Professor @ Sogang Univ / Advisory Professor @ DGIST

About Speaker

- *Co-Founder & CTO @ Erudio Bio, San Jose & Novato, CA, USA*
- *Advisor & Evangelist @ CryptoLab, Inc., San Jose, CA, USA*
- Chief Business Development Officer @ WeStory.ai, Cupertino, CA, USA
- Advisory Professor, Electrical Engineering and Computer Science @ DGIST, Korea
- Adjunct Professor, Electronic Engineering Department @ Sogang University, Korea
- Global Advisory Board Member @ Innovative Future Brain-Inspired Intelligence System Semiconductor of Sogang University, Korea
- *KFAS-Salzburg Global Leadership Initiative Fellow @ Salzburg Global Seminar*, Salzburg, Austria
- Technology Consultant @ Gerson Lehrman Group (GLG), NY, USA
- *Co-Founder & CTO / Head of Global R&D & Chief Applied Scientist / Senior Fellow @ Gauss Labs, Inc., Palo Alto, CA, USA* 2020 ~ 2023

- Senior Applied Scientist @ Amazon.com, Inc., Vancouver, BC, Canada ~ 2020
- Principal Engineer @ Software R&D Center, DS Division, Samsung, Korea ~ 2017
- Principal Engineer @ Strategic Marketing & Sales Team, Samsung, Korea ~ 2016
- Principal Engineer @ DT Team, DRAM Development Lab, Samsung, Korea ~ 2015
- Senior Engineer @ CAE Team, Samsung, Korea ~ 2012
- PhD - Electrical Engineering @ Stanford University, CA, USA ~ 2004
- Development Engineer @ Vyan, Santa Clara, CA, USA ~ 2001
- MS - Electrical Engineering @ Stanford University, CA, USA ~ 1999
- BS - Electrical & Computer Engineering @ Seoul National University 1994 ~ 1998

Highlight of Career Journey

- BS in EE @ SNU, MS & PhD in EE @ Stanford University
 - *Convex Optimization - Theory, Algorithms & Software*
 - advised by *Prof. Stephen P. Boyd*
- Principal Engineer @ Samsung Semiconductor, Inc.
 - AI & Convex Optimization
 - collaboration with *DRAM/NAND Design/Manufacturing/Test Teams*
- Senior Applied Scientist @ Amazon.com, Inc.
 - e-Commerce AIs - anomaly detection, deep RL, and recommender system
 - Bezos's project - drove *\$200M* in additional sales via Amazon Mobile Shopping App
- *Co-Founder & CTO / Global R&D Head & Chief Applied Scientist @ Gauss Labs, Inc.*
- *Co-Founder & CTO* - AI Technology & Business Development @ Erudio Bio, Inc.

Today

- Machine Learning Prerequisites - 5
 - linear algebra basics, calculus basics, statistics basics
 - discrete random variables, continuous random variables
- Machine Learning Basics - 52
 - optimal estimator, bias & variance, MLE, MAP, Bayesian inference
 - supervised learning, unsupervised learning, reinforcement learning
 - ML, DL, CNN, RNN
 - DNN training using SGD with backpropagation
- Learning AI - 93
 - tips, some books, online courses, Andrew Ng!
- Appendices - 98
 - LLM
 - Generative AI (genAI) - 128
- Selected references - 146
- References - 148

ML Prerequisites

Linear Algebra Basics

Scalars, vectors, and matrices

- real number $a \in \mathbf{R}$, called *scalar*
- (ordered) collection of real numbers $(a_1, \dots, a_n) \in \mathbf{R}^n$, called *vector*

$$\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} \in \mathbf{R}^n \quad \text{- column vector}$$

$$\begin{bmatrix} a_1 & a_2 & \cdots & a_n \end{bmatrix} \in \mathbf{R}^{1 \times n} \quad \text{- row vector}$$

- (ordered) collection of 2-dimensional array, called *matrix*

$$\begin{bmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,n} \\ A_{2,1} & A_{2,2} & \cdots & A_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m,1} & A_{m,2} & \cdots & A_{m,n} \end{bmatrix} \in \mathbf{R}^{m \times n}$$

Transposes

- transpose of row vector is column vector & vice versa

$$\begin{bmatrix} a_1 & a_2 & \cdots & a_n \end{bmatrix}^T = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix} \quad \& \quad \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}^T = \begin{bmatrix} a_1 & a_2 & \cdots & a_n \end{bmatrix}$$

- transpose of m -by- n matrix is n -by- m matrix

$$\begin{bmatrix} A_{1,1} & A_{1,2} & \cdots & A_{1,n} \\ A_{2,1} & A_{2,2} & \cdots & A_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{m,1} & A_{m,2} & \cdots & A_{m,n} \end{bmatrix}^T = \begin{bmatrix} A_{1,1} & A_{2,1} & \cdots & A_{m,1} \\ A_{1,2} & A_{2,2} & \cdots & A_{m,2} \\ \vdots & \vdots & \ddots & \vdots \\ A_{1,n} & A_{2,n} & \cdots & A_{m,n} \end{bmatrix} \in \mathbb{R}^{n \times m}$$

Matrix-vector multiplication

- for matrix $A \in \mathbf{R}^{m \times n}$ & vector $b \in \mathbf{R}^n$
 - matrix-vector multiplication Ab defined by

$$Ab = \begin{bmatrix} A_{1,1}b_1 + A_{1,2}b_2 + \cdots + A_{1,n}b_n \\ A_{2,1}b_1 + A_{2,2}b_2 + \cdots + A_{2,n}b_n \\ \vdots \\ A_{m,1}b_1 + A_{m,2}b_2 + \cdots + A_{m,n}b_n \end{bmatrix} \in \mathbf{R}^m$$

in other words

$$(Ab)_i = \sum_{j=1}^n A_{i,j}b_j \quad \text{for } 1 \leq i \leq m$$

- resulting quantity is vector of length m
- number of columns of A *must* equal to length of b

Matrix-matrix multiplication

- for matrices $A \in \mathbf{R}^{m \times n}$ & $B \in \mathbf{R}^{n \times p}$

- matrix-matrix multiplication $AB \in \mathbf{R}^{m \times p}$ defined by

$$(AB)_{i,j} = \sum_{k=1}^n A_{i,k}B_{k,j} \quad \text{for } 1 \leq i \leq m$$

- resulting quantity is m -by- p matrix
 - *order matters* and number of columns of A *must* equal to number of rows of B
- note matrix-vector multiplication is *special case* of matrix-matrix multiplication

Calculus Basics

Functions

- $f : X \rightarrow Y$
 - $X = \text{dom } f$ - domain of f
 - Y - codomain of f
 - $\mathcal{R}(f) = \{f(x) \in Y | x \in X\}$ - range of f

Differentiation & derivatives

- for real-valued function $f : \mathbf{R} \rightarrow \mathbf{R}$

- derivative of f at $x \in \mathbf{R}$

$$f'(x) = \frac{d}{dx} f(x) = \lim_{h \rightarrow 0} \frac{f(x + h) - f(x)}{h} \in \mathbf{R}$$

- derivative exists *if and only if* limit exists
 - second derivative of f at $x \in \mathbf{R}$

$$f''(x) = \frac{d^2}{dx^2} f(x) = \lim_{h \rightarrow 0} \frac{f'(x + h) - f'(x)}{h} \in \mathbf{R}$$

- second derivative exists *if and only if* limit exists

Multivariate functions

- $f : \mathbf{R}^n \rightarrow \mathbf{R}$ - real-valued multivariate function

$$f(x) = f\left(\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}\right) = f(x_1, x_2, \dots, x_n) \in \mathbf{R}$$

- examples
 - $f : \mathbf{R}^3 \rightarrow \mathbf{R}$ - linear function

$$f(x) = x_1 + 3x_2 + 2x_3$$

- $f : \mathbf{R}^3 \rightarrow \mathbf{R}$ - convex quadratic function

$$f(x) = x_1^2 + x_1x_2 + 3x_2^2 + 5x_3^2$$

Multivariate vector functions

- $f : \mathbf{R}^n \rightarrow \mathbf{R}^m$ - real-valued multivariate vector function

$$f(x) = \begin{bmatrix} f_1(x) \\ f_2(x) \\ \vdots \\ f_m(x) \end{bmatrix} \in \mathbf{R}^m$$

where $f_j : \mathbf{R}^n \rightarrow \mathbf{R}$ for $1 \leq j \leq m$

- examples
 - $f : \mathbf{R}^3 \rightarrow \mathbf{R}^2$ - linear function

$$f(x) = \begin{bmatrix} x_1 + 3x_2 + 2x_3 \\ -3x_2 + x_3 \end{bmatrix} \in \mathbf{R}^2$$

- $f : \mathbf{R}^3 \rightarrow \mathbf{R}^3$ - componentwise function

$$f(x) = [\exp(x_1) \quad \exp(x_2) \quad \exp(x_3)]^T \in \mathbf{R}^3$$

Partial derivative & gradient

for $f : \mathbf{R}^n \rightarrow \mathbf{R}$

- i th partial derivative

$$\frac{\partial}{\partial x_i} f(x) = \frac{f(x + he_i) - f(x)}{h} = \frac{f(\dots, x_{i-1}, x_i + h, x_{i+1}, \dots) - f(x)}{h}$$

where $e_i \in \mathbf{R}^n$ is i th unit vector

- gradient is vector of partial derivatives

$$\nabla f(x) = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \\ \frac{\partial f(x)}{\partial x_2} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{bmatrix} \in \mathbf{R}^n$$

- we have

$$(\nabla f(x))_i = \frac{\partial}{\partial x_i} f(x) = e_i^T \nabla f(x) \in \mathbf{R}$$

Jacobian

for $f : \mathbf{R}^n \rightarrow \mathbf{R}^m$

- Jacobian matrix

$$Df(x) = \begin{bmatrix} \frac{\partial f_1(x)}{\partial x_1} & \frac{\partial f_1(x)}{\partial x_2} & \dots & \frac{\partial f_1(x)}{\partial x_n} \\ \frac{\partial f_2(x)}{\partial x_1} & \frac{\partial f_2(x)}{\partial x_2} & \dots & \frac{\partial f_2(x)}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m(x)}{\partial x_1} & \frac{\partial f_m(x)}{\partial x_2} & \dots & \frac{\partial f_m(x)}{\partial x_n} \end{bmatrix} \in \mathbf{R}^{m \times n}$$

- equivalently

$$Df(x) = \begin{bmatrix} \nabla f_1(x)^T \\ \nabla f_2(x)^T \\ \vdots \\ \nabla f_m(x)^T \end{bmatrix} \in \mathbf{R}^{m \times n}$$

Chain rule

- for $f : \mathbf{R} \rightarrow \mathbf{R}^m$, $g : \mathbf{R}^m \rightarrow \mathbf{R}$ & $h = g \circ f$, i.e., $h(x) = g(f_1(x), \dots, f_m(x))$, derivative of h at $x \in \mathbf{R}$

$$h'(x) = \sum_{j=1}^m \frac{\partial}{\partial y_j} g(f(x)) f'_j(x) = \sum_{j=1}^m \nabla g(f(x))_j f'_j(x) \in \mathbf{R}$$

- for $f : \mathbf{R}^n \rightarrow \mathbf{R}^m$, $g : \mathbf{R}^m \rightarrow \mathbf{R}^p$ & $h = g \circ f$, Jacobian of h at $x \in \mathbf{R}^n$

$$Dh(x) = Dg(f(x)) Df(x) \in \mathbf{R}^{p \times n}$$

- note $Dg(f(x)) \in \mathbf{R}^{p \times m}$ & $Df(x) \in \mathbf{R}^{m \times n}$
- first is *special case* of second

Statistics Basics

Random experiments & probability law

- *random experiment*
 - outcome varies in unpredictable fashion (even) when experiment is being repeated under same conditions
 - specified by stating experimental procedure and set of one or more measurements or observations
- probability law
 - rule assigning probabilities to events of experiment that belong to event class \mathcal{F}

$$p : \mathcal{F} \rightarrow \mathbf{R}_+$$

- properties (or axioms)
 - for event $A \in \mathcal{F}$, $p(A)$ called *probability* of A
 - for event $A, B \in \mathcal{F}$ with $A \cap B = \emptyset$

$$p(A \cup B) = p(A) + p(B)$$

Conditional probability

- probability of event A given that event B has occurred, called *conditional probability*, denoted by

$$p(A|B)$$

- formula

$$p(A|B) = \frac{p(A \cap B)}{p(B)}$$

- thus

$$p(A \cap B) = p(A|B)p(B) = p(B|A)p(A)$$

- Bayes' theorem

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)}$$

Independence

- for events A & B , when knowledge of occurrence of B does not alter probability of A
 - A said to be *independent* of B
- following statements are equivalent
 - A is independent of B
 - B is independent of A
 - $p(A|B) = p(A)$
 - $p(B|A) = p(B)$
 - $p(A \cap B) = p(A)p(B)$

Random variables

- *discrete* random variable X assumes values from countable set $\{x_1, x_2, \dots\}$
- *continuous* random variable X assumes values from \mathbf{R}
- random *vector* X assumes values from \mathbf{R}^n

PMF, PDF & CDF

- *probability mass function (PMF)* of discrete X

$$p_X(x) = p(X = x)$$

- *probability density function (PDF)* of continuous X

$$\int_a^b p_X(x) = p(a \leq X \leq b)$$

- *cumulative distribution function (CDF)* of (any) X

$$F_X(x) = p(X \leq x)$$

- for discrete X - $F_X(x) = \sum_{x' \leq x} p_X(x')$
- for continuous X - $F_X(x) = \int_{-\infty}^{\bar{x}} p_X(x') dx'$

Expected value, variance & covariance matrix

- expected value

- for discrete X

$$\mathbf{E} X = \sum_x x p_X(x)$$

- for continuous X

$$\mathbf{E} X = \int_{-\infty}^{\infty} x p_X(x) dx$$

- variance for scalar $X \in \mathbf{R}$

$$\mathbf{Var}(X) = \mathbf{E}(X - \mathbf{E} X)^2 = \mathbf{E} X^2 - (\mathbf{E} X)^2$$

- covariance matrix for vector $X \in \mathbf{R}^n$

$$\mathbf{Var}(X) = \mathbf{E}(X - \mathbf{E} X)(X - \mathbf{E} X)^T = \mathbf{E} XX^T - (\mathbf{E} X)(\mathbf{E} X)^T$$

Joint PMF, PDF & CDF

- *joint PMF* of discrete X & Y

$$p_{X,Y}(x, y) = p(X = x, Y = y)$$

- *join PDF* of continuous X & Y

$$\int_c^d \int_a^b p_{X,Y}(x, y) dx dy = p(a \leq X \leq b \ \& \ c \leq Y \leq d)$$

- *joint CDF* of X & Y

$$F_{X,Y}(x, y) = p(X \leq x \ \& \ Y \leq y)$$

Conditional expectation

for two random variables X & Y

- expected value of Y conditioned on X

$$\mathbf{E}(Y|X = x) = \int y p(y|x) dy$$

where

$$p(y|x) = \frac{p_{X,Y}(x,y)}{p_X(x)}$$

- note

$$\mathbf{E}_{X,Y} f(X, Y) = \mathbf{E}_X \mathbf{E}_Y (f(X, Y)|X)$$

because

$$\int \int f(x, y) p(x, y) dx dy = \int \left(\int f(x, y) p(y|x) dy \right) p(x) dx$$

Discrete Random Variables

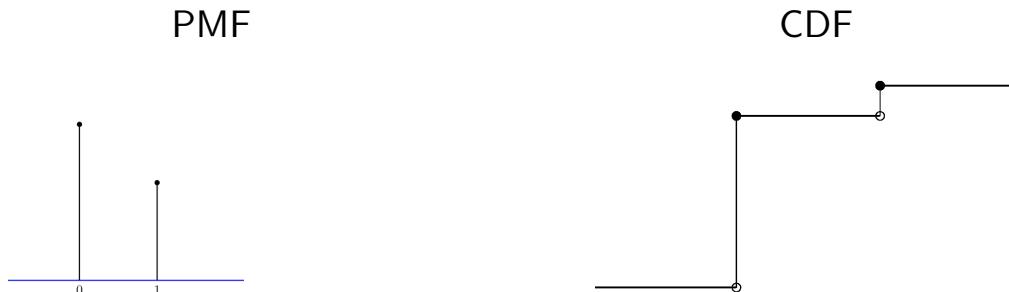
Bernoulli distribution

- model single binary trial with probability p of success (and, hence $(1 - p)$ of failure)
- PMF, mean, variance

$$p(k) = p^k(1-p)^{1-k} = \begin{cases} 1-p & \text{if } k=0 \\ p & \text{if } k=1 \end{cases}$$

$$\mathbf{E}(X) = p \quad \mathbf{Var}(X) = p(1-p)$$

- ML applications - (foundation for)
 - logistic regression, binary classification, modeling click-through rates, A/B testing outcomes



Binomial distribution

- model number of successes in n independent Bernoulli trials with probability p
- PMF, mean, variance

$$p(k) = \binom{n}{k} p^k (n-p)^{1-k} \quad \text{for } 1 \leq k \leq n$$

$$\mathbf{E}(X) = np \quad \mathbf{Var}(X) = np(1-p)$$

- ML applications
 - modeling conversion rates, quality control testing, ensemble voting methods, batch processing success rates



Multinomial distribution

- generalizes binomial distribution to multiple categories with probabilities p_1, \dots, p_k
- PMF, mean, variance

$$p(k) = \frac{n!}{x_1! \cdots x_k!} p_1^{x_1} \cdots p_k^{x_k}$$

$$\mathbf{E}(X_i) = np_i \quad \mathbf{Var}(X_i) = np_i(1 - p_i) \quad \mathbf{Cov}(X_i, X_j) = -np_ip_j$$

- ML applications
 - multi-class classification, topic modeling, document classification, market basket analysis
- widely used in *Bayesian inference* with Dirichlet priors



Geometric distribution

- model number of trials needed to achieve first success in independent Bernoulli trials
- PMF, mean, variance

$$p(k) = p(1 - p)^{k-1} \quad \mathbf{E}(X) = 1/p \quad \mathbf{Var}(X) = (1 - p)/p^2$$

- ML applications
 - modeling time-to-conversion, failure analysis, reinforcement learning episode lengths, web crawling stopping conditions
- memoryless property $p(X > m + n | X > m) = p(X > n)$

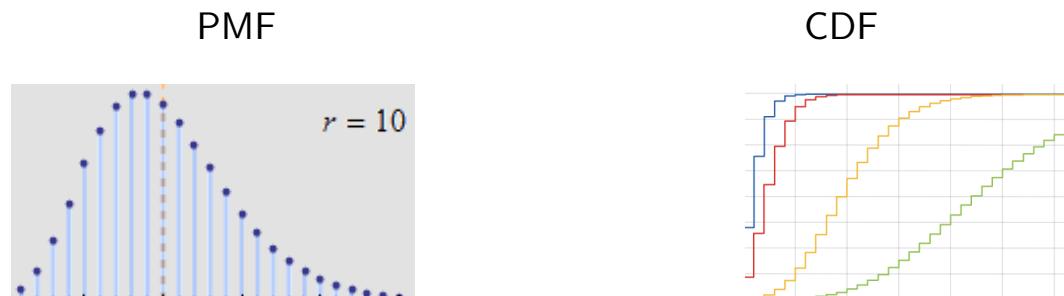


Negative binomial distribution

- model number of trials needed to achieve r successes in independent Bernoulli trials
- PMF, mean, variance

$$p(k) = \binom{k-1}{r-1} p^r (1-p)^{k-r} \quad \mathbf{E}(X) = r/p \quad \mathbf{Var}(X) = r(1-p)/p^2$$

- ML applications
 - modeling overdispersed count data, customer acquisition costs, reliability engineering, text analysis for word frequencies
- *often used when Poisson assumptions are violated due to overdispersion*



Poisson distribution

- model number of events occurring in fixed interval of time or space
- PMF, mean, variance ($\lambda > 0$)

$$p(k) = e^{-\lambda} \lambda^k / k! \quad \mathbf{E}(X) = \lambda \quad \mathbf{Var}(X) = \lambda$$

- ML applications
 - modeling web traffic, system failures
 - word counts (in NLP), user interactions (in recommendation systems)
- approximates binomial when n is large & p is small with $= np$
- sum of independent Poisson variables is Poisson



Hypergeometric distribution

- model number of successes in n draws without replacement from finite population of size N containing K successes
- PMF, mean, variance ($N, K \in \mathbf{N}$ with $N > K$)

$$p(k) = \frac{\binom{K}{k} \binom{N-K}{n-k}}{\binom{N}{n}} \quad \mathbf{E}(X) = \frac{nK}{N} \quad \mathbf{Var}(X) = \frac{nK}{N} \cdot \frac{N-K}{N} \cdot \frac{N-n}{N-1}$$

- ML applications
 - sampling without replacement, quality control testing
 - feature selection validation, A/B testing with finite populations



Continuous Random Variables

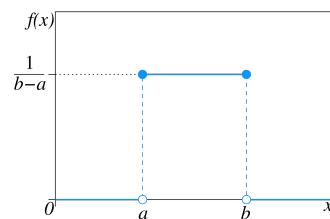
Uniform distribution

- model equally likely outcomes over continuous interval $[a, b]$ representing complete uncertainty within bounded range
- PDF, mean, variance ($a, b \in \mathbf{R}$ with $b > a$)

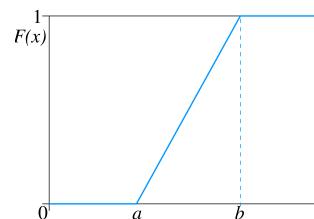
$$p(x) = 1/(b - a)I_{[a,b]}(x) \quad \mathbf{E}(X) = (a + b)/2 \quad \mathbf{Var}(X) = (b - a)^2/12$$

- ML applications
 - Monte Carlo sampling, generating baseline distributions for hypothesis testing
- maximum entropy distribution for bounded continuous support
- foundation for pseudo-random number generation and inverse transform sampling

PDF



CDF



Gaussian distribution

- most important continuous distribution
- model symmetric bell-shaped data arising from many natural processes
- PDF, mean, variance ($\mu \in \mathbf{R}$, $\sigma > 0$)

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/2\sigma^2} \quad \mathbf{E}(X) = \mu \quad \mathbf{Var}(X) = \sigma^2$$

- ML applications
 - linear regression error terms, NN weight initialization, PCA, noise modeling
- invariant under linear transformations, maximum entropy for given mean and variance

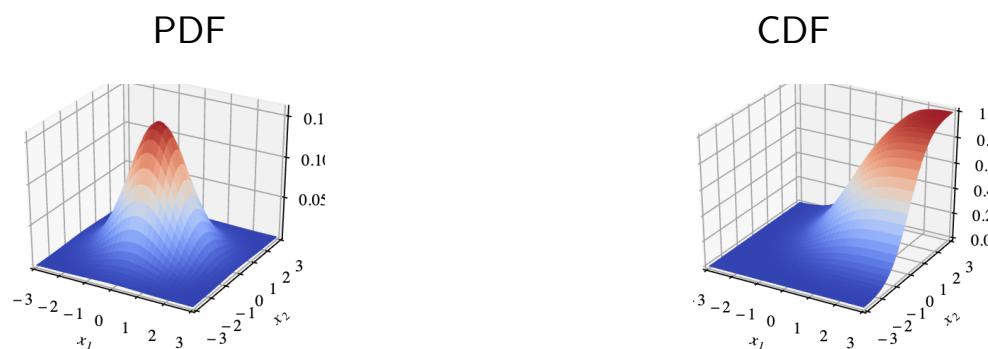


Multivariate Gaussian distribution

- generalize scalar Gaussian to random vector
- PDF, mean, variance ($\mu \in \mathbf{R}^n$, $\Sigma \in \mathbf{S}_{++}^n$)

$$p(x) = \frac{1}{(2\pi)^{n/2} \det(\Sigma)^{1/2}} e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1} (x-\mu)} \quad \mathbf{E}(X) = \mu \quad \mathbf{Cov}(X) = \Sigma$$

- ML applications
 - Gaussian mixture, PCA, Kalman filtering, Gaussian processes, latent variable models
- maximum likelihood estimation having closed-form solution, foundation for many Bayesian models



Exponential distribution

- model time between events in Poisson process, representing memoryless waiting times or lifetimes
- PDF, mean, variance ($\lambda > 0$)

$$p(x) = \lambda e^{-\lambda x} I_{[0,\infty)}(x) \quad \mathbf{E}(X) = 1/\lambda \quad \mathbf{Var}(X) = 1/\lambda^2$$

- ML applications
 - system failure times, web session durations, survival analysis
- memoryless property $p(X > s+t | X > s) = p(X > t)$ - only continuous distribution with this property, minimum of exponentials is exponential



Gamma distribution

- model positive continuous values - generalizing exponential distribution to allow for more flexible shapes, *e.g.*, for waiting times for multiple events
- PDF, mean, variance ($\alpha, \beta > 0$)

$$p(x) = \frac{\beta^\alpha}{\Gamma(\alpha)} x^{\alpha-1} e^{-\beta x} I_{[0,\infty)}(x) \quad \mathbf{E}(X) = \alpha/\beta \quad \mathbf{Var}(X) = \alpha/\beta^2$$

- ML applications
 - survival analysis, queuing theory
- exponential is special case when $\alpha = 1$, sum of independent exponentials is gamma, conjugate prior for Poisson and exponential distributions



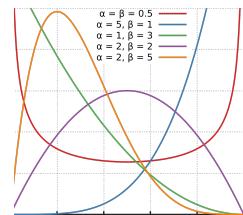
Beta distribution

- model probabilities and proportions, defined on $[0, 1]$ with flexible shapes from uniform to highly skewed
- PDF, mean, variance ($\alpha, \beta > 0$)

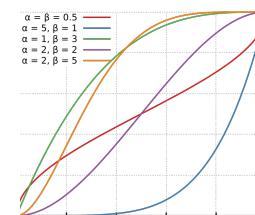
$$p(x) = \frac{\Gamma(\alpha, \beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1} (1-x)^{\beta-1} \quad \mathbf{E}(X) = \frac{\alpha}{\alpha + \beta} \quad \mathbf{Var}(X) = \frac{\alpha\beta}{(\alpha + \beta)^2(\alpha + \beta + 1)}$$

- ML applications
 - modeling success rates, A/B testing, probability calibration
- uniform is special case when $\alpha = \beta = 1$, conjugate prior for Bernoulli & binomial related to Dirichlet distribution

PDF



CDF



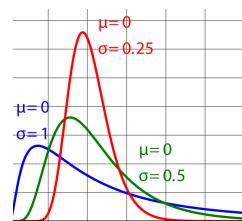
Log-normal distribution

- model positive values where logarithm follows normal distribution, representing multiplicative processes and heavy-tailed phenomena
- PDF, mean, variance ($\mu \in \mathbf{R}$, $\sigma > 0$)

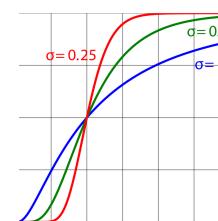
$$p(x) = e^{-(\log x - \mu)^2/2\sigma^2} / x\sigma\sqrt{2\pi} \quad \mathbf{E}(X) = e^{\mu + \sigma^2/2} \quad \mathbf{Var}(X) = (e^{\sigma^2} - 1)e^{2\mu + \sigma^2}$$

- ML applications
 - modeling income distributions, stock prices, file sizes, network traffic, biological measurements, computational complexity
- heavy right tail, multiplicative central limit theorem

PDF



CDF



Chi-square distribution

- model sum of squares of independent standard normal random variables, fundamental in statistical testing and confidence intervals
- PDF, mean, variance ($\nu \in \mathbf{N}$ - degree of freedom)

$$p(x) = \frac{1}{2^{\nu/2}\Gamma(\nu/2)}x^{\nu/2-1}e^{-x/2}I_{[0,\infty)}(x) \quad \mathbf{E}(X) = \nu \quad \mathbf{Var}(X) = 2\nu$$

- ML applications
 - goodness-of-fit testing, feature selection, confidence intervals for variance, regularization in NN
- special case of gamma distribution, sum of independent chi-squares is chi-square



Student's t -distribution

- model sum of squares of independent standard normal random variables, fundamental in statistical testing and confidence intervals
- PDF, mean, variance ($\nu > 0$ degrees of freedom - almost always positive integer)

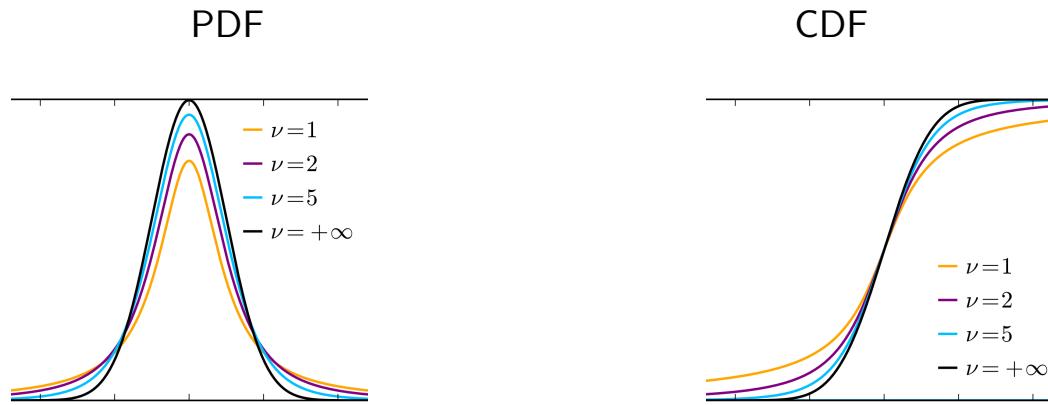
$$p(x) = \frac{\Gamma((\nu + 1)/2)}{\sqrt{\pi\nu}\Gamma(\nu/2)}(1 + x^2/\nu)^{-(\nu+1)/2}$$

$$\mathbf{E}(X) = \begin{cases} 0 & \text{if } \nu > 1 \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$\mathbf{Var}(X) = \begin{cases} \nu/(\nu - 2) & \text{if } \nu > 2 \\ \infty & \text{if } 1 < \nu \leq 2 \\ \text{undefined} & \text{otherwise} \end{cases}$$

- ML applications
 - Bayesian inference, robust regression, confidence intervals with small samples, uncertainty quantification in DL

- heavier tails than normal, approaches standard normal as ν approaches ∞ , symmetric around zero, undefined moments for small ν



Weibull distribution

- model survival times & failure rates with flexible hazard functions, generalizing exponential distribution for reliability analysis
- PDF, mean, variance ($\lambda, k > 0$)

$$p(x) = (k/\lambda)(x/\lambda)^{k-1} e^{-(x/\lambda)^k} I_{[0,\infty)}(x) \quad \mathbf{E}(X) = \lambda\Gamma(1 + 1/k)$$

- ML applications
 - survival analysis, reliability engineering, wind speed modeling, NN activation functions, extreme value theory
- flexible hazard function, minimum of Weibull variables is Weibull



Cauchy distribution

- model heavy-tailed symmetric data with undefined mean and variance, arising in physics and robust statistics
- PDF, mean, variance ($x_0 \in \mathbf{R}$, $\gamma > 0$)

$$p(x) = \frac{1}{\pi\gamma(1 + ((x - x_0)/\gamma)^2)} \quad \mathbf{E}(X) = \text{undefined} \quad \mathbf{Var}(X) = \text{undefined}$$

- ML applications
 - robust statistics, modeling outliers, Bayesian inference with heavy-tailed priors, physics simulations, anomaly detection
- no defined moments, stable distribution, ratio of two independent normals is Cauchy



Laplace distribution

- model symmetric data with heavier tails than normal, representing difference between two independent exponential variables
- PDF, mean, variance ($\mu \in \mathbf{R}$, $b > 0$)

$$p(x) = \frac{1}{2b} \exp(-|x - \mu|/b) \quad \mathbf{E}(X) = \mu \quad \mathbf{Var}(X) = 2b^2$$

- ML applications
 - lasso, robust regression, sparse coding, image processing, privacy-preserving ML
- maximum entropy for given mean absolute deviation, related to L1 penalty, robust to outliers (fundamentally more than normal distribution)



Pareto distribution

- model heavy-tailed phenomena following power-law distributions, representing “80-20 rule” and scale-free networks
- PDF, mean, variance ($x_m, \alpha > 0$)

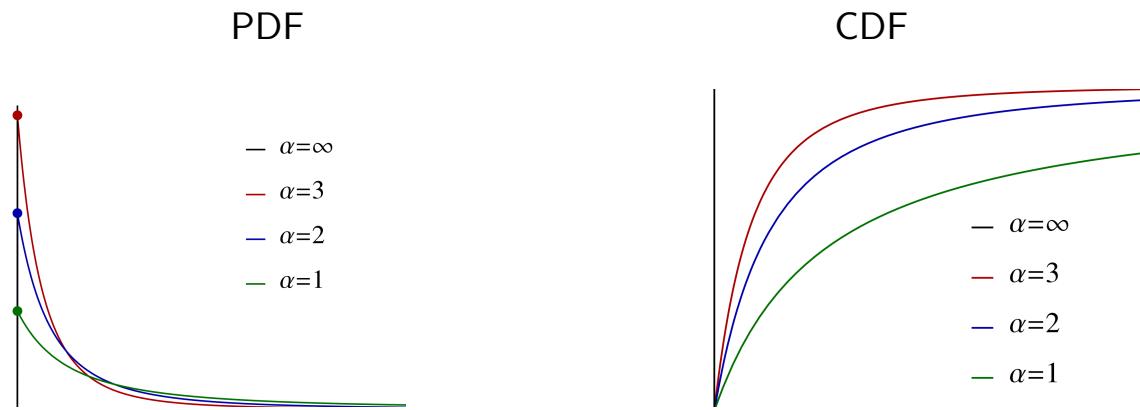
$$p(x) = \alpha x_m^\alpha / x^{\alpha+1}$$

$$\mathbf{E}(X) = \begin{cases} \infty & \text{if } \alpha \leq 1 \\ \alpha x_m / (\alpha - 1) & \text{if } \alpha > 1 \end{cases}$$

$$\mathbf{Var}(X) = \begin{cases} \infty & \text{if } \alpha \leq 2 \\ \alpha x_m^2 / (\alpha - 1)^2 (\alpha - 2) & \text{if } \alpha > 2 \end{cases}$$

- ML applications
 - model wealth distributions, network degree distributions, web page rankings, file sizes, NLP

- heavy right tail, scale-free property, finite moments only for sufficiently large α , basis for power-law distributions



ML Basics

Estimation, Regression, and Inference

The optimal estimator

- estimation problem
 - for two random variables $X \in \mathbf{R}^n$ & $Y \in \mathbf{R}^m$
 - design *estimator or predictor* $g : \mathbf{R}^n \rightarrow \mathbf{R}^m$ to make $g(X)$ *as close as possible* to Y
- when *closeness* measured by mean-square-error (MSE), *the optimal solution* exists

$$g^*(x) = \mathbf{E}(Y|X = x)$$

Proof of optimality

$$\begin{aligned}
 \mathbf{E}_{X,Y}((g(X) - g^*(X))^T(g^*(X) - Y)) &= \mathbf{E}_X \mathbf{E}_Y((g(X) - g^*(X))^T(g^*(X) - Y)|X) \\
 &= \mathbf{E}_X((g(X) - g^*(X))^T \mathbf{E}_Y(g^*(X) - Y)|X) \\
 &= 0
 \end{aligned}$$

hence

$$\begin{aligned}
 \mathbf{E} \|g(X) - Y\|_2^2 &= \mathbf{E} \|g(X) - g^*(X) + g^*(X) - Y\|_2^2 \\
 &= \mathbf{E} \|g(X) - g^*(X)\|_2^2 + \mathbf{E} \|g^*(X) - Y\|_2^2 + 2 \mathbf{E}(g(X) - g^*(X))^T(g^*(X) - Y) \\
 &= \mathbf{E} \|g(X) - g^*(X)\|_2^2 + \mathbf{E} \|g^*(X) - Y\|_2^2 \\
 &\geq \mathbf{E} \|g^*(X) - Y\|_2^2
 \end{aligned}$$

Regression

- in most cases, *not* possible to obtain g^* (unless, e.g., full knowledge of joint PDF)
- regression problem
 - given data set $D = \{(x_1, y_1), \dots, (x_N, y_N)\} \subset \mathbf{R}^n \times \mathbf{R}^m$
 - find $g : \mathbf{R}^n \rightarrow \mathbf{R}^m$ to make $g(X)$ *as close as possible* to Y
- given certain regression method, regressor depends on dataset D

$$g(\cdot; D)$$

Bias & variance

assuming \mathcal{D} is random variable for dataset D

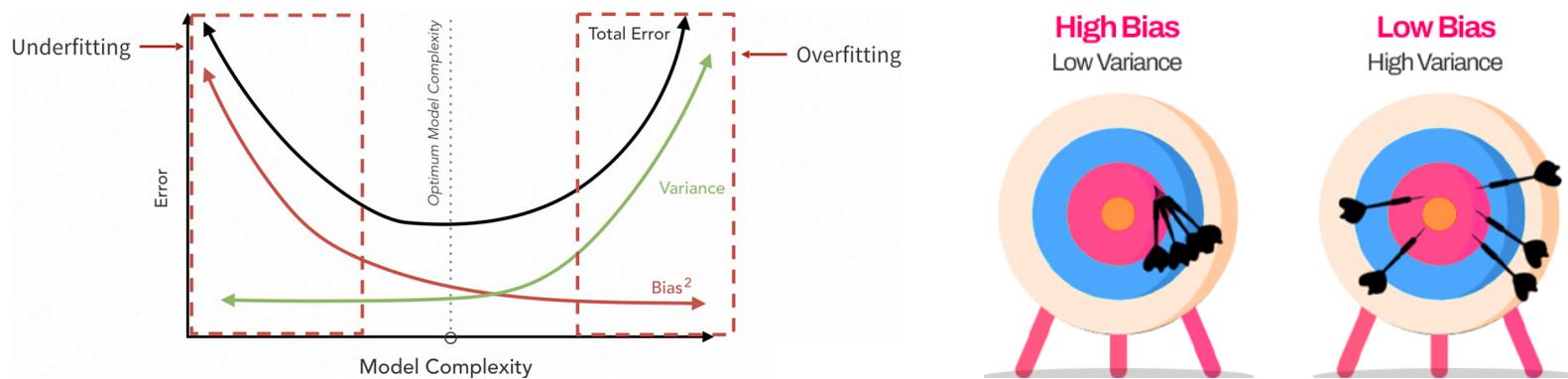
- estimation MSE is

$$\begin{aligned}
 & \mathbf{E}_{X,Y,\mathcal{D}} \|g(X; \mathcal{D}) - Y\|_2^2 \\
 &= \underbrace{\mathbf{E}_{X,\mathcal{D}} \|g(X; \mathcal{D}) - \mathbf{E}_{\mathcal{D}} g(X; \mathcal{D})\|_2^2}_{\text{variance}} + \underbrace{\mathbf{E}_X \|\mathbf{E}_{\mathcal{D}} g(X; \mathcal{D}) - g^*(X)\|_2^2}_{\text{bias}} + \underbrace{\mathbf{E}_{X,Y} \|g^*(X) - Y\|_2^2}_{\text{noise}} \\
 &= \underbrace{\mathbf{E}_{X,\mathcal{D}} \|g(X; \mathcal{D}) - \mathbf{E}_{\mathcal{D}} g(X; \mathcal{D})\|_2^2}_{\text{variance}} + \underbrace{\mathbf{E}_{X,Y} \|\mathbf{E}_{\mathcal{D}} g(X; \mathcal{D}) - Y\|_2^2}_{\text{bias + noise}}
 \end{aligned}$$

- bias & variance
 - *bias* measures how good model is in average
 - *variance* measures how much model varies depending on dataset it is trained on
- *noise* cannot be reduced even with the optimal predictor

Model choice & hyperparameter optimization

- want to choose model or modeling method to make both bias & variance low
 - (too) complex models have low bias, but high variance
 - (too) simple models have low variance, but high bias
- usually solved by *hyperparameter optimization*
 - sometimes called *hyperparameter tuning*



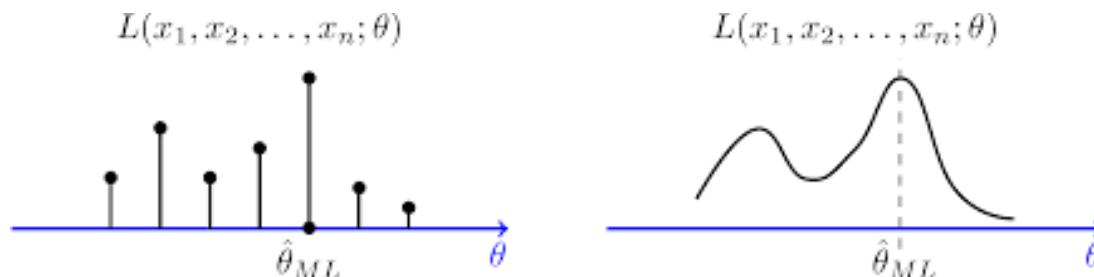
MLE

- maximum likelihood estimation (MLE)
 - assume parameterized distribution of $X \in \mathbb{R}^n$ by $\theta \in \Theta$ - $p(x; \theta)$
 - find θ maximizing *likelihood function*

$$p(x_1, \dots, x_N; \theta) = \prod_{i=1}^N p(x_i; \theta)$$

- MLE solution

$$\hat{\theta}_{\text{MLE}} = \underset{\theta \in \Theta}{\operatorname{argmax}} \prod_{i=1}^N p(x_i; \theta)$$



MAP estimation

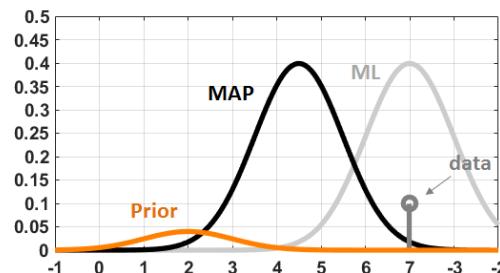
- maximum a posteriori (MAP) estimation
 - assume *prior knowledge* of θ - $p(\theta)$
 - assume parameterized distribution of $X \in \mathbb{R}^n$ by θ - $p(x|\theta)$
 - find θ maximizing *posteriori probability*

$$p(\theta|x_1, \dots, x_N)$$

– Bayes' theorem implies $p(\theta|x_1, \dots, x_N) \propto p(\theta) \prod_{i=1}^N p(x_i|\theta)$

- MAP solution

$$\hat{\theta}_{\text{MAP}} = \underset{\theta \in \Theta}{\operatorname{argmax}} p(\theta) \prod_{i=1}^N p(x_i|\theta)$$



Bayesian inference

- both MLE & MAP estimation are *point estimations*
- Bayesian inference
 - updates *prior distribution* by replacing it with posterior distribution
- conjugate prior
 - if prior can be further parameterized by hyperparameter α and posterior is in same probability distribution family, both prior and posterior called *conjugate distributions*, prior called *conjugate prior*

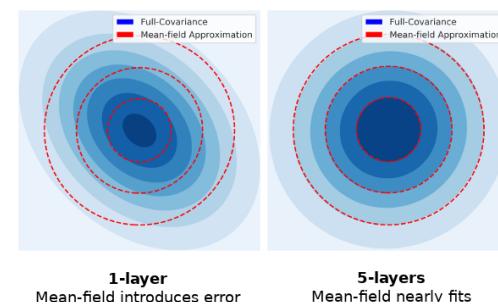
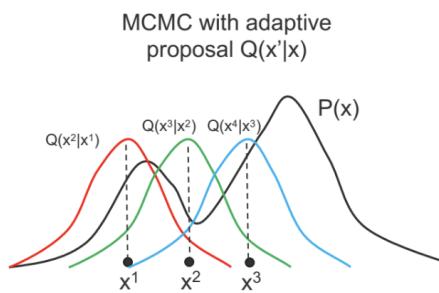
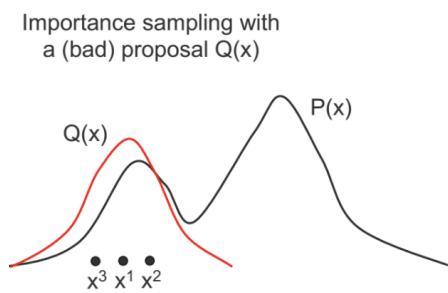
$$p(\theta; \alpha)$$

- in this case, can update hyperparameter α , i.e., find α^+ such that

$$p(\theta; \alpha^+) = p(\theta | x_1, \dots, x_N; \alpha) = \frac{p(\theta; \alpha) \prod_{i=1}^N p(x_i | \theta; \alpha)}{p(x_1, \dots, x_N; \alpha)}$$

Bayesian algorithms & methods

- exact inference methods
 - conjugate priors - *e.g.*, Beta-Binomial, Normal-Normal, *etc.*
- Markov Chain Monte Carlo (MCMC)
 - Metropolis-Hastings algorithm, Gibbs sampling, Hamiltonian Monte Carlo (HMC)
- variational inference (VI)
 - mean field variational Bayes - assuming parameter independence for tractability
 - structured variational inference - maintaining dependencies & inference tractability
 - variational autoencoder (VAE) - NN-based VI for complex distributions

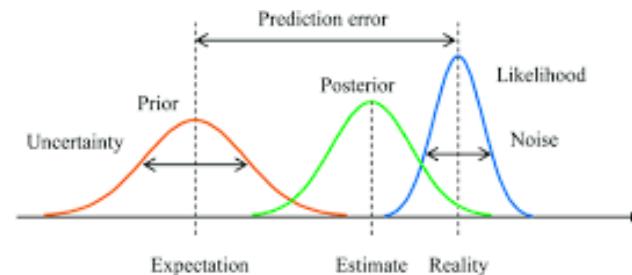


Pros & cons of Bayesian Inference

- pros
 - principled uncertainty quantification - providing complete probability distributions
 - incorporates prior knowledge - allowing to formally include domain expertise, *etc.*
 - coherent framework - providing mathematically consistent approach
 - natural sequential learning - easily handles streaming data or online learning scenarios
 - interpretable results - outputs directly interpretable as probabilities
- cons
 - computational complexity - often requiring sophisticated sampling methods
 - prior sensitivity, scalability issues, implementation difficulty, slower inference, model selection challenges

$$p(\theta | \text{data}) = \frac{p(\text{data} | \theta) \cdot p(\theta)}{p(\text{data})}$$

↑ Posterior ↓ Likelihood ↓ Prior
 ↑ Normalization

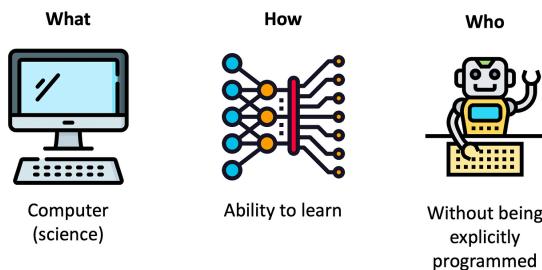


Machine Learning

Machine Learning

- ML

- subfield of computer science that
“gives computers the ability to learn without being explicitly programmed.”
- Arthur Samuel (1959)
- *not* magic, still less intelligent than humans for many cases
- *numerically minimizes* certain (mathematical) loss function to (indirectly) solve *some statistically meaningful* problems



Machine learning is the subfield of computer science that gives “computers the ability to learn without being explicitly programmed.”



Arthur Samuel

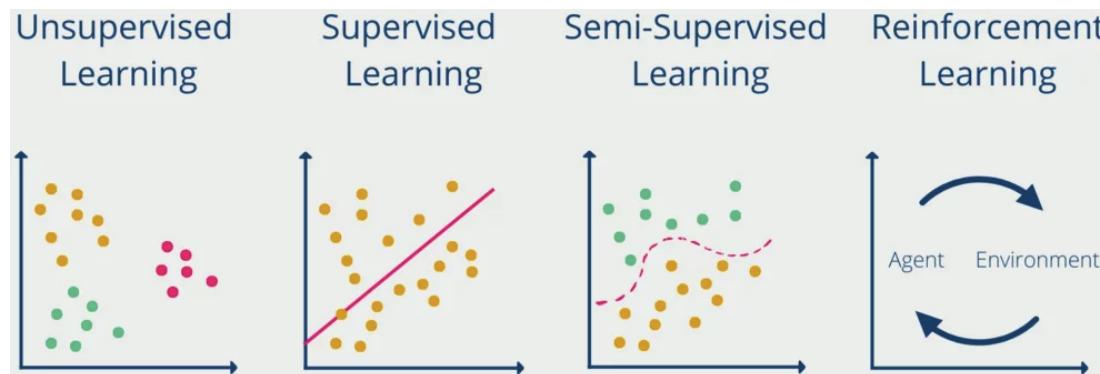
Two famous quotes

- Albert Einstein

The grand aim of all science is to cover the greatest number of empirical facts by logical deduction from the smallest possible number of hypotheses or axioms.

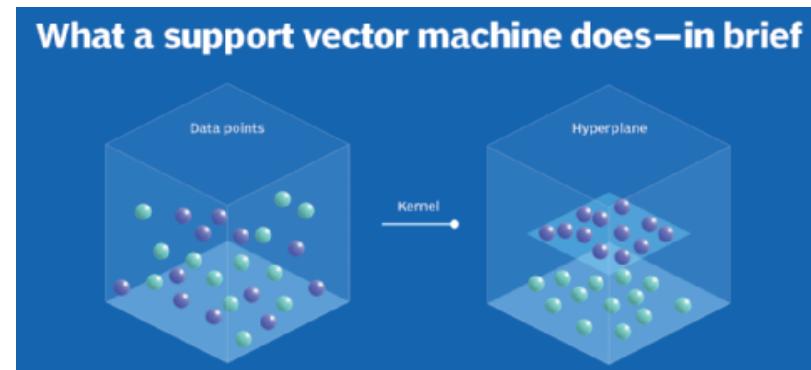
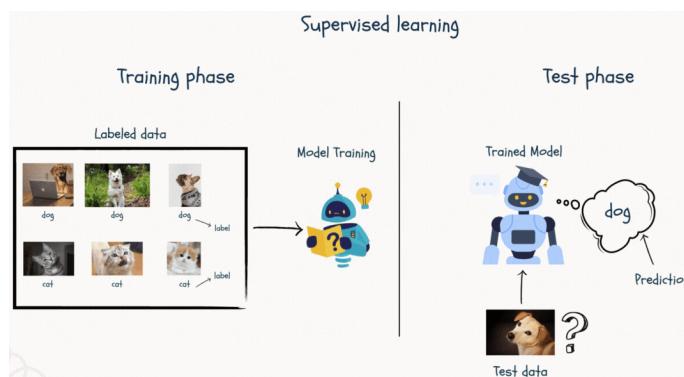
- Alfred North Whitehead

Civilization advances by extending the number of important operations which we can perform without thinking about them. - Operations of thought are like cavalry charges in a battle – they are strictly limited in number, they require fresh horses, and must only be made at decisive moments.



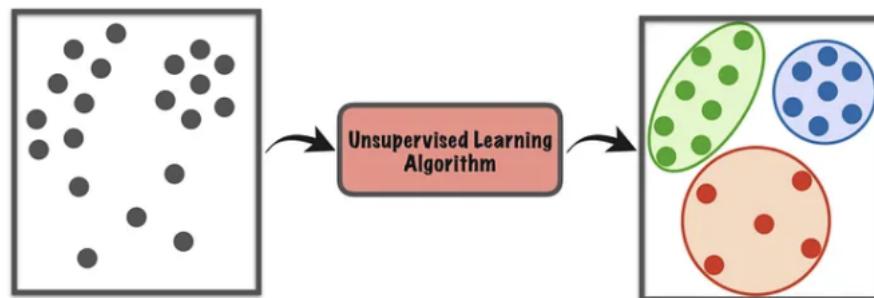
Supervised learning

- most basic and widely used type of ML
- model is trained on dataset where correct output or “label” is provided for each input
- use cases
 - image classification, object detection, semantic segmentation
 - natural language processing (NLP) - text classification, sentiment analysis
 - predictive modeling, medical diagnosis
- algorithms
 - linear regression, logistic regression, decision trees, random forest
 - support vector machine (SVM), k -nearest neighbors (kNN)



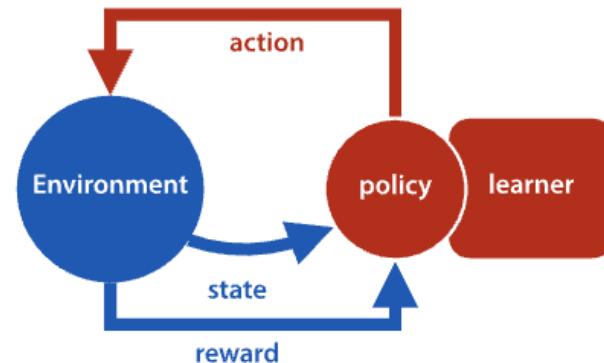
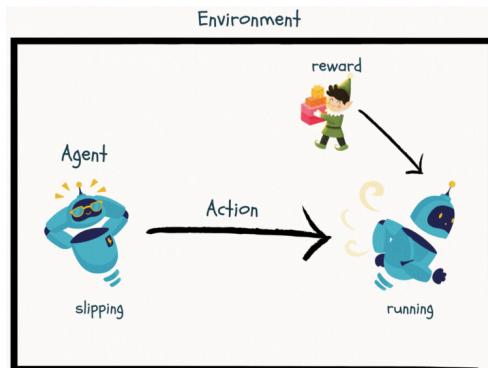
Unsupervised learning

- model is given dataset without any labels or output
- model finds patterns & structure within data on its own
- use cases
 - clustering, dimensionality reduction
 - anomaly detection, generative models
- algorithms
 - k-means clustering, hierarchical clustering, principal component analysis (PCA)
 - t-distributed stochastic neighbor embedding (t-SNE)



Reinforcement learning

- (quite different from supervised & unsupervised learnings)
- model learns from consequences of its actions
 - model receives feedback on its performance; feedback called *reward*
 - uses that information to adjust its actions and improve its performance over time
- use cases
 - robotics, game playing, autonomous vehicles, industrial control
 - healthcare, finance
- algorithms
 - Q-learning, SARSA, DQN, A3C, policy gradient



ML Formulations

Statistical problem formulation

- assume data set $X_m = \{x^{(1)}, \dots, x^{(m)}\}$
 - drawn independently from (true, but unknown) data generating distribution $p_{\text{data}}(x)$
- maximum likelihood estimation (MLE) is to solve

$$\text{maximize } p_{\text{model}}(X; \theta) = \prod_{i=1}^m p_{\text{model}}(x^{(i)}; \theta)$$

where optimization variable is θ

- find *most plausible or likely model* that fits data
- equivalent (but more numerically tractable) formulation

$$\text{maximize } \log p_{\text{model}}(X; \theta) = \sum_{i=1}^m \log p_{\text{model}}(x^{(i)}; \theta)$$

MLE & KL divergence

- in information theory, Kullback-Leibler (KL) divergence defines distance between two probability distributions p & q

$$D_{\text{KL}}(p\|q) = \mathbf{E}_{X \sim p} \log p(X)/q(X) = \int_{x \in \Omega} p(x) \log \frac{p(x)}{q(x)} dx$$

- KL divergence between data distribution p_{data} & model distribution p_{model} can be approximated by Monte Carlo method as

$$D_{\text{KL}}(p_{\text{data}}\|p_{\text{model}}(\theta)) \simeq \frac{1}{m} \sum_{i=1}^m (\log p_{\text{data}}(x^{(i)}) - \log p_{\text{model}}(x^{(i)}; \theta))$$

where $x^{(i)}$ are drawn (of course) according to p_{data}

- hence *minimizing KL divergence is equivalent to solving MLE problem!*

Equivalence of MLE to MSE

- assume model is Gaussian, *i.e.*, $y \sim \mathcal{N}(g_\theta(x), \Sigma)$ ($g_\theta(x) \in \mathbf{R}^p$, $\Sigma \in \mathbf{S}_{++}^p$)

$$p(y|x; \theta) = \frac{1}{\sqrt{2\pi}^p |\Sigma|^{1/2}} \exp \left(-\frac{1}{2} (y - g_\theta(x))^T \Sigma^{-1} (y - g_\theta(x)) \right)$$

- assuming that $\Sigma = \alpha I_p$, log-likelihood becomes

$$\begin{aligned} \sum_{i=1}^m \log p(x^{(i)}, y^{(i)}; \theta) &= \sum_{i=1}^m \log p(y^{(i)}|x^{(i)}; \theta) p(x^{(i)}) \\ &= - \sum_{i=1}^m \|y^{(i)} - g_\theta(x^{(i)})\|_2^2 / 2\alpha - \frac{pm}{2} \log(2\pi\alpha) + \sum_{i=1}^m \log p(x^{(i)}) \end{aligned}$$

- hence *minimizing mean-square-error (MSE) is equivalent to solving MLE problem!*

Numerical optimization problem formulation

- (true) problem to solve

$$\text{minimize } \mathbf{E} l(g_\theta(X), Y)$$

- *impossible* to solve

- basic formulation - surrogate problem to solve

$$\text{minimize } f(\theta) = \frac{1}{m} \sum_{i=1}^m l(g_\theta(x^{(i)}), y^{(i)})$$

- formulation with regularization

$$\text{minimize } f(\theta) = \frac{1}{m} \sum_{i=1}^m l(g_\theta(x^{(i)}), y^{(i)}) + \gamma r(\theta)$$

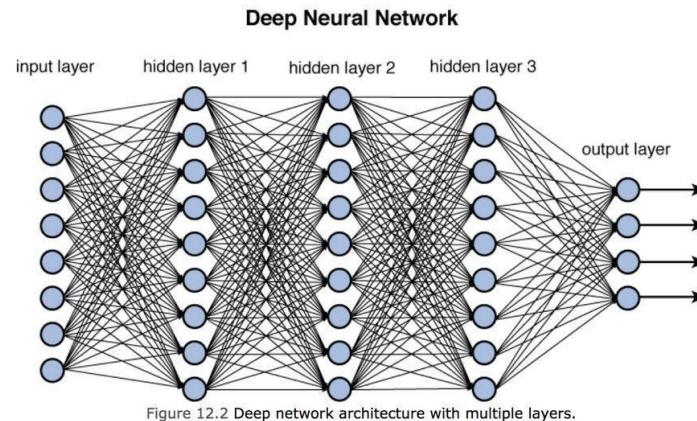
- stochastic gradient descent (SGD)

$$\theta^{(k+1)} = \theta^{(k)} - \alpha_k \nabla f(\theta^{(k)})$$

Deep Learning

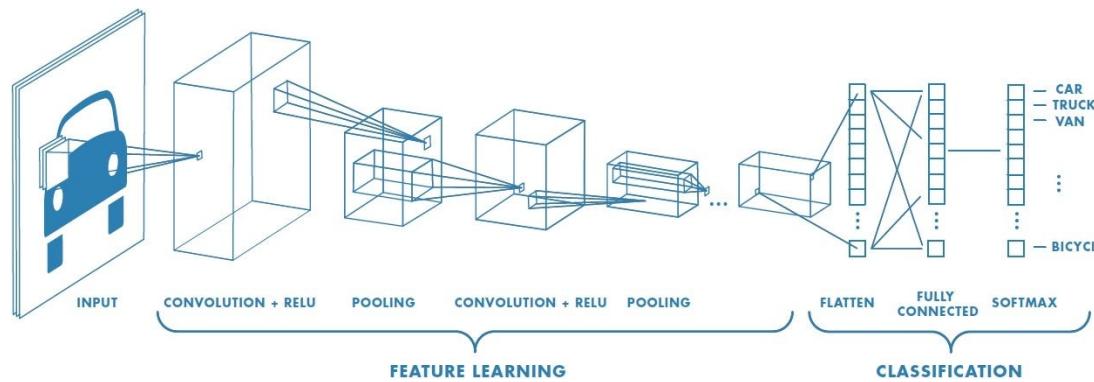
Deep learning (DL)

- machine learning using artificial neural networks with multiple layers for
 - automatically learning hierarchical representations of data
- key components
 - deep neural networks, hidden layers, backpropagation, activation functions
 - hierarchical feature learning, representation learning, end-to-end learning
- key breakthroughs enabling DL
 - massively available data, GPU computing, algorithmic advances



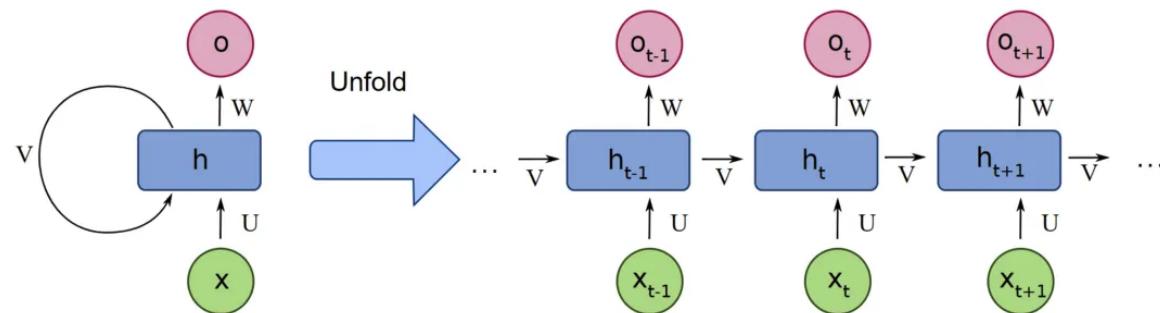
Convolutional neural network (CNN)

- specialized DL learning architecture designed for
 - processing grid-like data such as images
 - where spatial relationships between pixels matter
- key components
 - convolutional layers, pooling layers, activation functions, fully connected layers
- how it works
 - feature extraction, translation invariance, parameter sharing
- why it excels
 - local connectivity, hierarchical learning



Recurrent neural network (RNN)

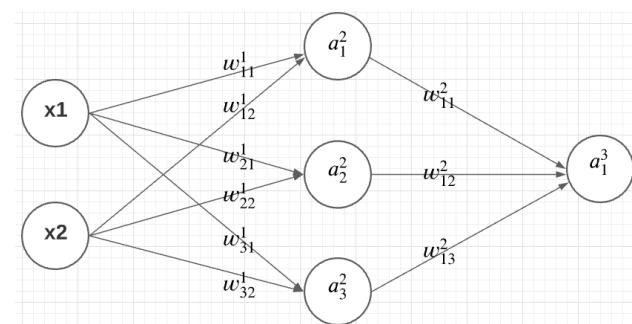
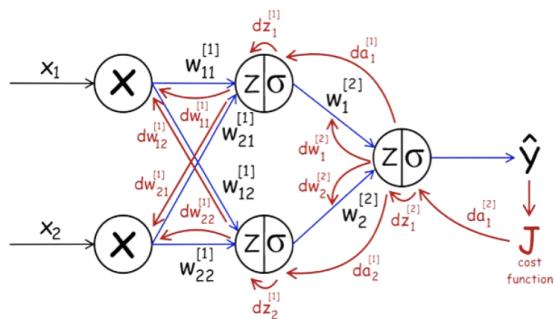
- neural network designed for
 - processing sequential data by maintaining memory of previous inputs
- key components
 - hidden states, recurrent connections, input/output layers, weight sharing
- how it works
 - sequential processing, memory mechanism, temporal dependencies
- why it excels
 - variable length input, context awareness, flexible architecture
- variants - long short-term memory (LSTM), gated recurrent unit (GRU)



Training DNN using SGD

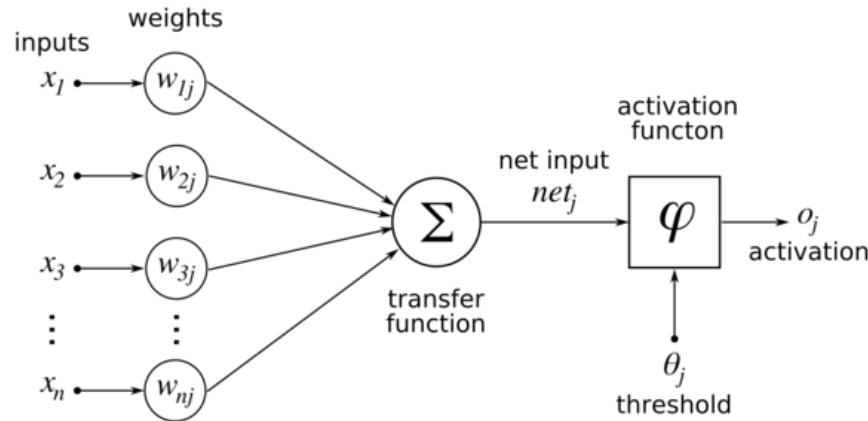
Notations

- p / q - dimension of input / output spaces
- $l : \mathbf{R}^q \times \mathbf{R}^q \rightarrow \mathbf{R}_+$ - loss function
- d - depth of neural network
- n_i ($1 \leq i \leq d$) - number of perceptrons in i th layer
- $z^{[i]} \in \mathbf{R}^{n_i}$ - input to i th layer
- $o^{[i]} \in \mathbf{R}^{n_i}$ - output of i th layer
- $W^{[i]} \in \mathbf{R}^{n_i \times n_{i-1}}$ - weights of connections between $(i-1)$ th and i th layer
- $w^{[i]} \in \mathbf{R}^{n_i \times n_{i-1}}$ - bias weights of i th layer
- $\phi^{[i]} : \mathbf{R}^{n_i} \rightarrow \mathbf{R}^{n_i}$ - activation functions of i th layer

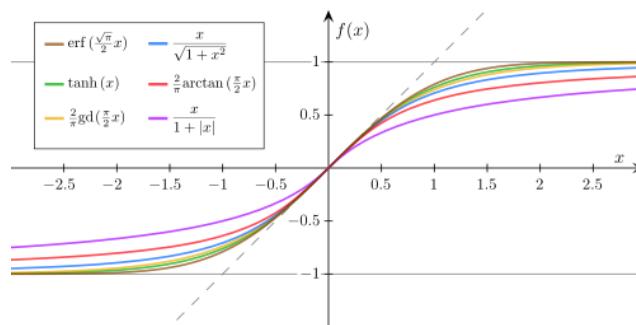


Basic unit & activation function

- basic unit



- activation function



Neural net equations

- modeling function for the (deep) neural network $g_\theta : \mathbf{R}^p \rightarrow \mathbf{R}^q$

$$g_\theta = \phi_\theta^{[d]} \circ \psi_\theta^{[d]} \circ \cdots \circ \phi_\theta^{[1]} \circ \psi_\theta^{[1]}$$

or equivalently

$$g_\theta(x) = \phi_\theta^{[d]}(\psi_\theta^{[d]}(\cdots(\phi_\theta^{[1]}(\psi_\theta^{[1]}(x)))))$$

- for i th layer
 - output via (componentwise) activation function

$$o^{[i]} = \phi^{[i]}(z^{[i]}) \Leftrightarrow o_j^{[i]} = \phi_j^{[i]}(z_j^{[i]}) \quad (1 \leq j \leq n_i)$$

- input via affine transformation $\psi^{[i]} : \mathbf{R}^{n_{i-1}} \rightarrow \mathbf{R}^{n_i}$

$$z^{[i]} = \psi^{[i]}(o^{[i-1]}) = W^{[i]}o^{[i-1]} + w^{[i]}$$

Stochastic gradient descent

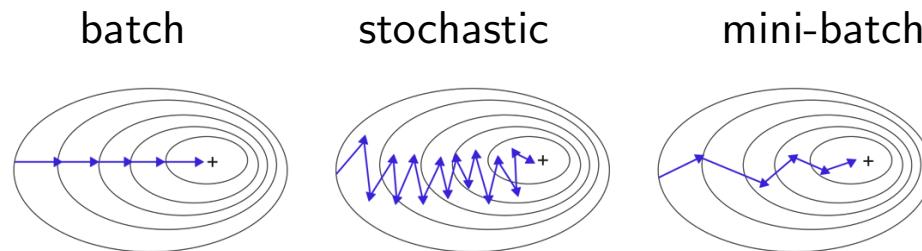
- ML training tries to minimize some loss function - $f(\theta)$ depends on (not only θ , but also) batch of data $(x^{(1)}, y^{(1)}), \dots, (x^{(m)}, y^{(m)})$

$$\text{minimize } f(\theta)$$

- while exist hundreds of optimization methods solving this problem
 - the only method used widely* is stochastic *gradient descent!*
- (stochastic) gradient descent

$$f(\theta^{k+1}) = f(\theta^k) - \alpha^k \nabla f(\theta^k)$$

- backpropagation* is used to evaluate this (stochastic) *gradient* using *chain rule*



Chain rule

- suppose
 - two functions $f : \mathbf{R}^n \rightarrow \mathbf{R}^m$ & $g : \mathbf{R}^m \rightarrow \mathbf{R}$
 - Jacobian of f - $Df : \mathbf{R}^n \rightarrow \mathbf{R}^{m \times n}$
 - gradient of g - $\nabla g : \mathbf{R}^m \rightarrow \mathbf{R}^m$
- gradient of composite function $h = g \circ f$

$$\nabla h(\theta) = Df(\theta)^T \nabla g(f(\theta)) \in \mathbf{R}^n \quad (\text{using matrix-vector multiplication})$$

in other words

$$\frac{\partial}{\partial \theta_i} h(\theta) = \sum_{j=1}^m \frac{\partial}{\partial \theta_i} f_j(\theta) \nabla_j g(f(\theta)) \quad (\text{scalar version})$$

Loss function & its gradient

- assume cost function of deep neural network is

$$f(\theta) = \frac{1}{m} \sum_{k=1}^m l(g_\theta(x^{(k)}), y^{(k)}) = \frac{1}{m} \sum_{k=1}^m f_k(\theta)$$

where

$$f_k(\theta) = l(g_\theta(x^{(k)}), y^{(k)})$$

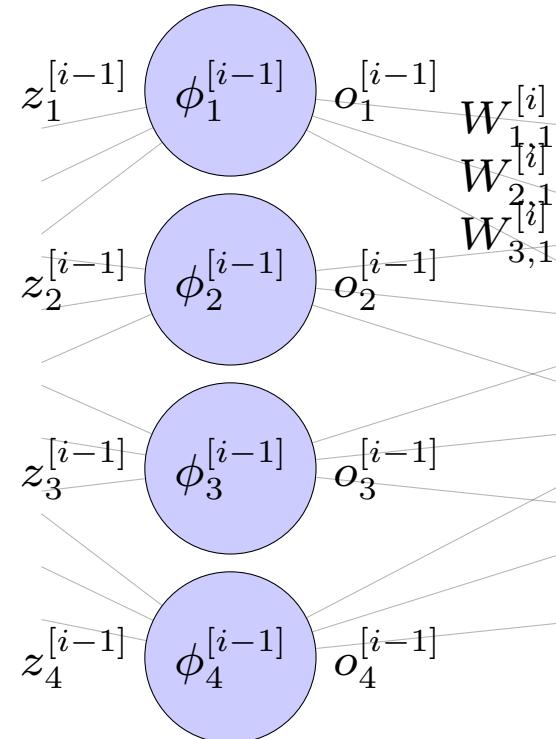
- gradient is

$$m \nabla_\theta f(\theta) = \sum_{k=1}^m \nabla_\theta l(g_\theta(x^{(k)}), y^{(k)}) = \sum_{k=1}^m \nabla_\theta f_k(\theta)$$

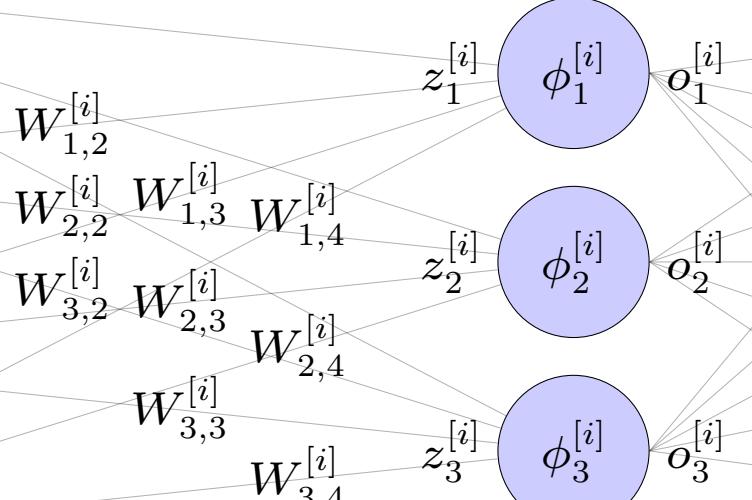
- i.e., evaluate gradient $\nabla_\theta f_k(\theta)$ for each data point $(x^{(k)}, y^{(k)})$

Hidden layers

(i - 1)th hidden layer



ith hidden layer



Backpropagation formula using chain rule

- for each data $(x^{(k)}, y^{(k)})$
 - via activation function

$$\frac{\partial}{\partial z_j^{[i]}} f_k(\theta) = \frac{\partial}{\partial o_j^{[i]}} f_k(\theta) \phi_j^{[i]'}(o_j^{[i]}) \quad \text{for } 1 \leq j \leq n_i \quad (1)$$

where $\phi_j^{[i]'}(o_j^{[i]})$ is derivative of activation function $\phi_j^{[i]}$ evaluated at $o_j^{[i]}$

- via affine transformation

$$\frac{\partial}{\partial W_{j,l}^{[i]}} f_k(\theta) = o_l^{[i-1]} \frac{\partial}{\partial z_j^{[i]}} f_k(\theta) \quad \text{for } 1 \leq j \leq n_i \text{ & } 1 \leq l \leq n_{i-1} \quad (2)$$

$$\frac{\partial}{\partial w_j^{[i]}} f_k(\theta) = \frac{\partial}{\partial z_j^{[i]}} f_k(\theta) \quad \text{for } 1 \leq j \leq n_i \quad (3)$$

$$\frac{\partial}{\partial o_l^{[i-1]}} f_k(\theta) = \sum_{j=1}^{n_i} W_{j,l}^{[i]} \frac{\partial}{\partial z_j^{[i]}} f_k(\theta) \quad \text{for } 1 \leq l \leq n_{i-1} \quad (4)$$

Backpropagation formula using matrix-vector multiplication

- for each data $(x^{(k)}, y^{(k)})$

- via activation function

$$\nabla_{z^{[i]}} f_k(\theta) = D\phi^{[i]} \nabla_{o^{[i]}} f_k(\theta) \quad (5)$$

where $D\phi^{[i]} = \text{diag}(\phi_1^{[i]'}(o_1^{[i]}), \dots, \phi_{n_i}^{[i]'}(o_{n_i}^{[i]}))$ is Jacobian of $\phi^{[i]}$ evaluated at $o^{[i]}$

- via affine transformation

$$\nabla_{W^{[i]}} f_k(\theta) = \nabla_{z^{[i]}} f_k(\theta) o^{[i-1]T} \in \mathbf{R}^{n_i \times n_{i-1}} \quad (6)$$

$$\nabla_{w^{[i]}} f_k(\theta) = \nabla_{z^{[i]}} f_k(\theta) \in \mathbf{R}^{n_i} \quad (7)$$

$$\nabla_{o^{[i-1]}} f_k(\theta) = W^{[i]T} \nabla_{z^{[i]}} f_k(\theta) \in \mathbf{R}^{n_{i-1}} \quad (8)$$

Backpropagation formula using Python numpy package

- for each data $(x^{(k)}, y^{(k)})$
 - via activation function

$$\text{grad_z} = \text{phi_dir} * \text{grad_o} \quad (9)$$

- where grad_z , phi_dir , grad_o are 1d numpy.ndarray of size n_i
- via affine transformation

$$\text{grad_W} = \text{numpy.dot}(\text{grad_z}, \text{val_o.T}) \quad (10)$$

$$\text{grad_w} = \text{grad_z.copy()} \quad (11)$$

$$\text{grad_o_prev} = \text{numpy.dot}(\text{grad_z}, \text{W}) \quad (12)$$

where val_o , grad_w are 1d numpy.ndarray of size n_i , grad_o_prev is 1d numpy.ndarray of size n_{i-1} , grad_W is 2d numpy.ndarray of shape (n_i, n_{i-1})

Gradient evaluation using backpropagation

- forward propagation - evaluate for each $(x^{(k)}, y^{(k)})$

$$g_{\theta}(x^{(k)}) = \phi_{\theta}^{[d]}(\psi_{\theta}^{[d]}(\cdots(\phi_{\theta}^{[1]}(\psi_{\theta}^{[1]}(x^{(k)})))))$$

- *backpropagation - evaluate partial derivatives backward*

- evaluate gradient with respect to output of output layer $o^{[d]} = g_{\theta}(x^{(k)})$

$$\nabla_{o^{[d]}} f_k(\theta) = \nabla_{y_1} l(g_{\theta}(x^{(k)}), y^{(k)})$$

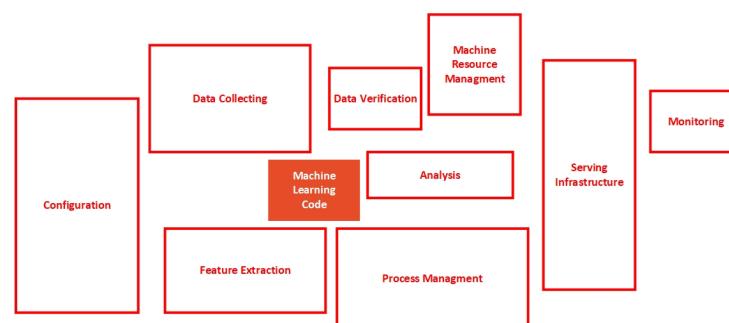
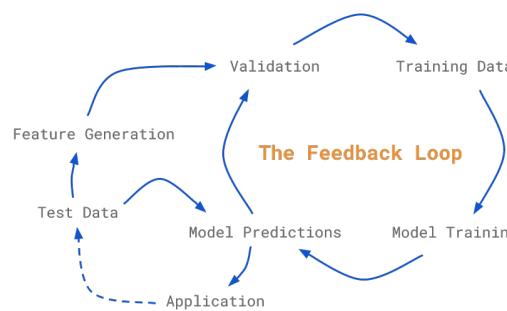
- evaluate gradient with respect to input from that with respect to output using (1), or equivalently, using (5) *i.e.*, evaluate $\nabla_{z^{[i]}} f_k(\theta)$ from $\nabla_{o^{[i]}} f_k(\theta)$
- evaluate gradient with respect to weights, bias, and intput of previous layer using (3), (4), & (2) or equivalently, using (7), (8), & (6) *i.e.*, evaluate $\nabla_{W^{[i]}} f_k(\theta)$, $\nabla_{w^{[i]}} f_k(\theta)$ & $\nabla_{o^{[i-1]}} f_k(\theta)$ from $\nabla_{z^{[i]}} f_k(\theta)$
- repeat back to input layer to evaluate all

$$\nabla_{W^{[1]}} f_k(\theta), \nabla_{w^{[1]}} f_k(\theta), \dots, \nabla_{W^{[d]}} f_k(\theta), \nabla_{w^{[d]}} f_k(\theta)$$

ML in Action

ML in practice

- define business problem - business objective, success metrics, establish baselines (early)
- data collection - data cleaning, validation & exploratory data analysis (EDA)
- feature engineering - based on domain expertise
- train/validation/test split - stratified sampling, chronological splits for time-series
- model selection or/and hyperparameter optimization
- monitoring, retraining & notification
- start simple, iterative fast (fail fast!), validate business impact - *e.g.*, A/B test



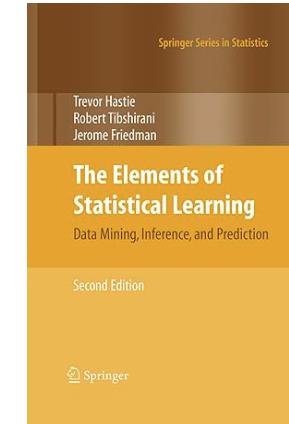
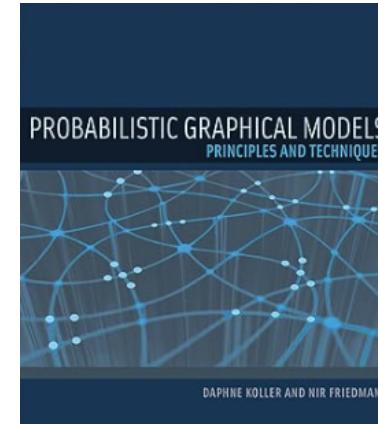
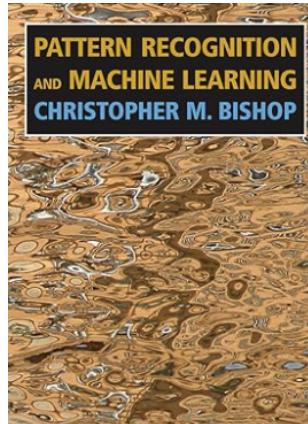
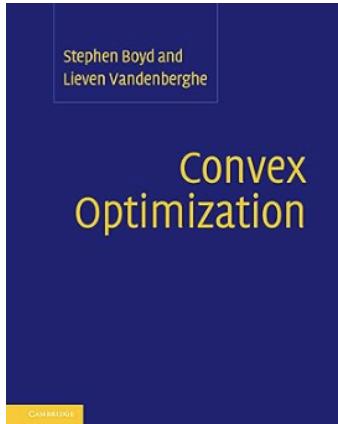
Learning AI

Best ways to learn AI & ML

- first, learn basics - college classes, online courses, (easy) books
 - no need to understand every mathematical details, but should know rough ideas!
- *hands-on is MUST!*
 - learn and practice coding - Python is MUST; do not do (only) R
 - learn git - know how to develop efficiently, plus import others' work
- I think *online courses are blessing to mankind!*
 - *can't* say “you can't do it because I don't have access to good resource or you don't go to good schools” because . . . they are available!
 - getting (expensive) certificates is good idea because . . . otherwise you wouldn't complete it! :) - and can post it on your LinkedIn!
- would be best if your task at work is related to ML
 - however, even if that's not the case or can't be the case, can always do your own personal projects – or contribute to public projects (on github)!

Books

- The Elements of Statistical Learning - Hastie, Tibshirani & Friedman [[HTF01](#)]
- Pattern Recognition and Machine Learning - Christopher M. Bishop [[Bis06](#)]
- Deep Learning - Ian Goodfellow, Yoshua Bengio & Aaron Courville [[GBC16](#)]
- Reinforcement Learning: An Introduction - Richard S. Sutton & Andrew G. Barto [[SB18](#)]
- Machine Learning: A Probabilistic Perspective - Kevin P. Murphy [[Mur12](#)]
- Probabilistic Graphical Models - Daphne Koller & Nir Friedman [[KF09](#)]
- Convex Optimization - Stephen Boyd & Lieven Vandenberghe [[BV04](#)]



Andrew Ng!

- Andrew Ng
 - (co-)founder of “Deep Learning.AI” and “Coursera”, prominent figure in ML & AI
 - his courses highly regarded because well-structured and provide insights
- [latest Andrew Ng courses](#)
 - AI Agents in LangGraph
 - AI Agentic Design Patterns with AutoGen
 - Introduction to On-device AI
 - Multi AI Agent Systems with Crew AI
 - Building Multimodal Search and RAG - contrastive learning, multimodality to RAG
 - Building Agentic RAG with LlamaIndex
 - Quantisation In Depth
 - In Prompt Engineering for Vision Models
 - Getting Started with Mistral - open-source models (Mistral 7B, Mixtral 8x7B)
 - Preprocessing Unstructured Data for LLM

Appendices

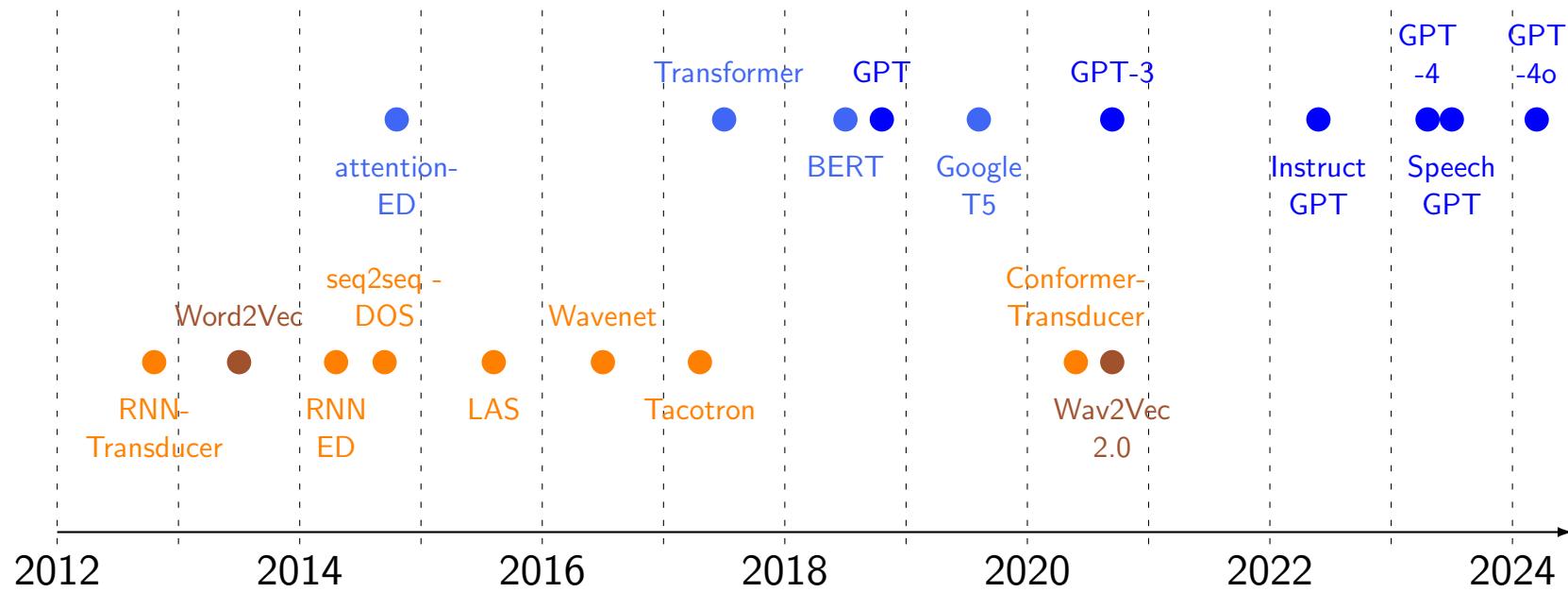
LLM

Language Models

History of language models

- bag of words - first introduced – 1954
- word embedding – 1980
- RNN based models - conceptualized by David Rumelhart – 1986
- LSTM (based on RNN) – 1997
- 380M-sized seq2seq model using LSTMs proposed – 2014
- 130M-sized seq2seq model using gated recurrent units (GRUs) – 2014
- Transformer - Attention is All You Need - A. Vaswani et al. @ Google – 2017
 - 100M-sized encoder-decoder multi-head attention model for machine translation
 - non-recurrent architecture, handle arbitrarily long dependencies
 - parallelizable, *simple* (linear-mapping-based) attention model

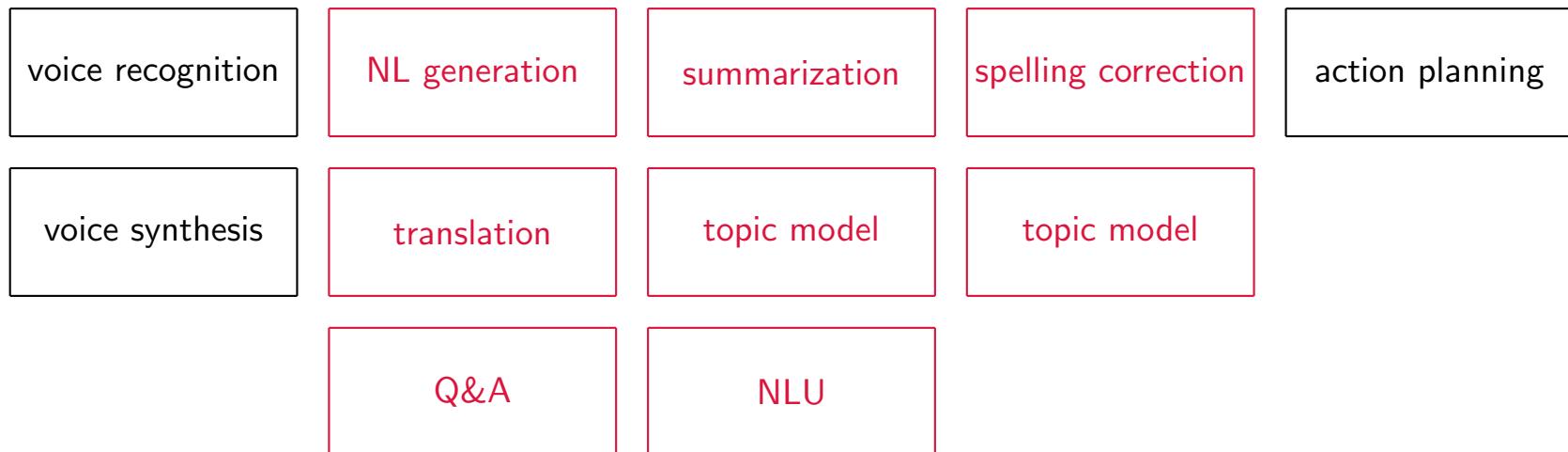
Recent advances in speech & language processing



- LAS: listen, attend, and spell, ED: encoder-decoder, DOS: decoder-only structure

Types of language models

- many of language models have **common requirements** - language representation learning
- can be learned via pre-training *high performing model* and fine-tuning/transfer learning/domain adaptation
- this *high performing model* learning essential language representation *is* (language) foundation model
 - actually, same for other types of learning, e.g., CV



NLP Market

NLP market size

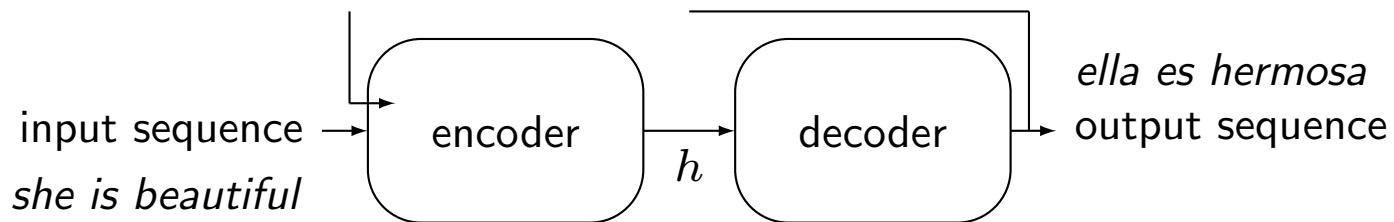
- global NLP market size estimated at USD 16.08B in 2022, is expected to hit USD 413.11B by 2032 - *CAGR of 38.4%*
- in 2022
 - north america NLP market size valued at USD 8.2B
 - high tech and telecom segment accounted revenue share of over 23.1%
 - healthcare segment held a 10% market share
 - (by component) solution segment hit 76% revenue share
 - (deployment mode) on-premise segment generated 56% revenue share
 - (organizational size) large-scale segment contributed highest market share
- source - [Precedence Research](#)



Sequence-to-Sequence Models

Sequence-to-sequence (seq2seq) model

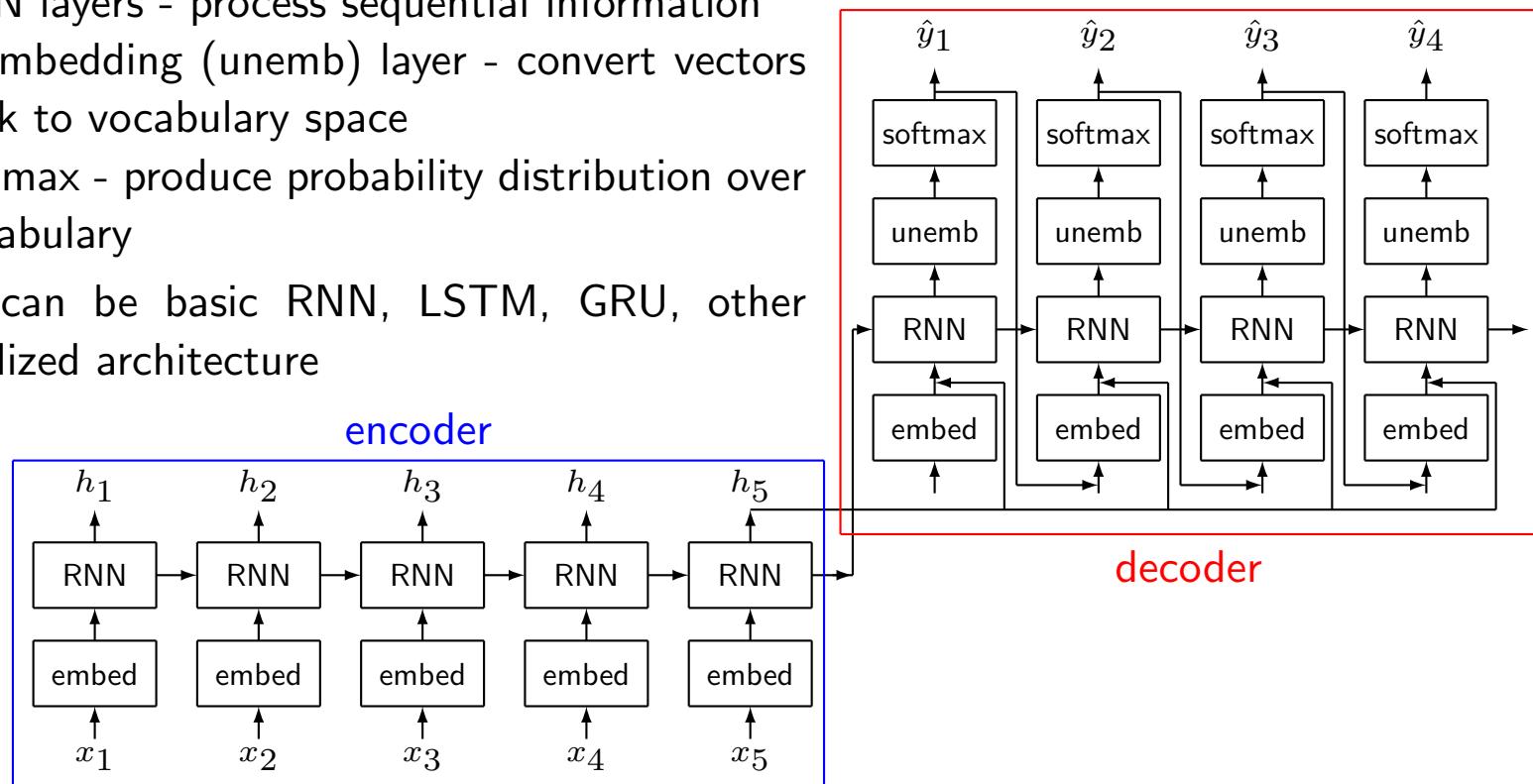
- seq2seq - take sequences as inputs and spit out sequences
- encoder-decoder architecture



- encoder & decoder can be RNN-type models
- $h \in \mathbf{R}^n$ - hidden state - *fixed length* vector
- (try to) condense and store information of input sequence (losslessly) in (fixed-length) hidden states
 - finite hidden state - not flexible enough, *i.e.*, cannot handle arbitrarily large information
 - memory loss for long sequences
 - LSTM was promising fix, but with (inevitable) limits

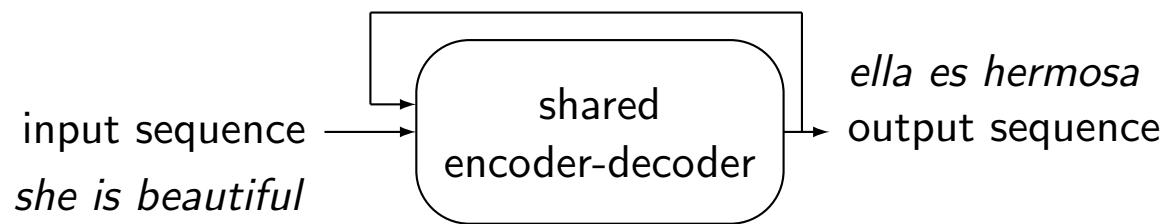
RNN-type encoder-decoder architecture

- components
 - embedding layer - convert input tokens to vector representations
 - RNN layers - process sequential information
 - unembedding (unemb) layer - convert vectors back to vocabulary space
 - softmax - produce probability distribution over vocabulary
- RNN can be basic RNN, LSTM, GRU, other specialized architecture



Shared encoder-decoder model

- single neural network structure can handle both encoding & decoding tasks
 - efficient architecture reducing model complexity
 - allow for better parameter sharing across tasks
- widely used in modern LLMs to process & generate text sequences
 - applications - machine translation, text summarization, question answering
- advantages
 - efficient use of parameters, versatile for multiple NLP tasks



Large Language Models

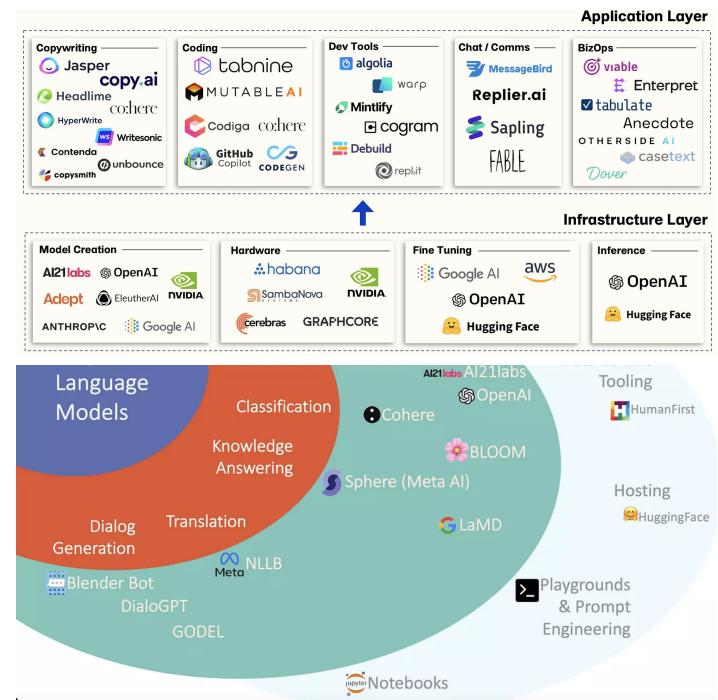
LLM

- LLM
 - type of AI aimed for NLP trained on massive corpus of texts & programming code
 - allow learn statistical relationships between words & phrases, *i.e.*, conditional probabilities
 - *amazing performance shocked everyone - unreasonable effectiveness of data (Halevy et al., 2009)*
- applications
 - conversational AI agent / virtual assistant
 - machine translation / text summarization / content creation / sentiment analysis / question answering
 - code generation
 - market research / legal service / insurance policy / triage hiring candidates
 - + virtually infinite # of applications



LLMs

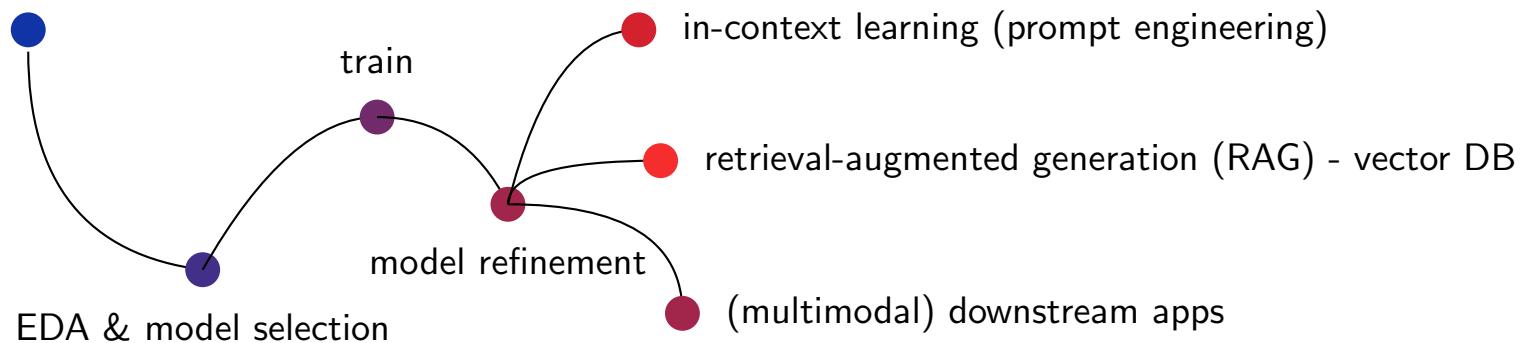
- Foundation Models
 - GPT-x/Chat-GPT - OpenAI, Llama-x - Meta, PaLM-x (Bard) - Google
- # parameters
 - generative pre-trained transformer (GPT) - GPT-1: 117M, GPT-2: 1.5B, GPT-3: 175B, GPT-4: 100T, GPT-4o: 200B
 - large language model Meta AI (Llama) - Llama1: 65B, Llama2: 70B, Llama3: 70B
 - scaling language modeling with pathways (PaLM) - 540B
- burns lots of cash on GPUs!
- applicable to many NLP & genAI applications



LLM building blocks

- data - trained on massive datasets of text & code
 - quality & size critical on performance
- architecture - GPT/Llama/Mistral
 - can make huge difference
- training - self-supervised/supervised learning
- inference - generates outputs
 - in-context learning, prompt engineering

goal and scope of LLM project



Transformer

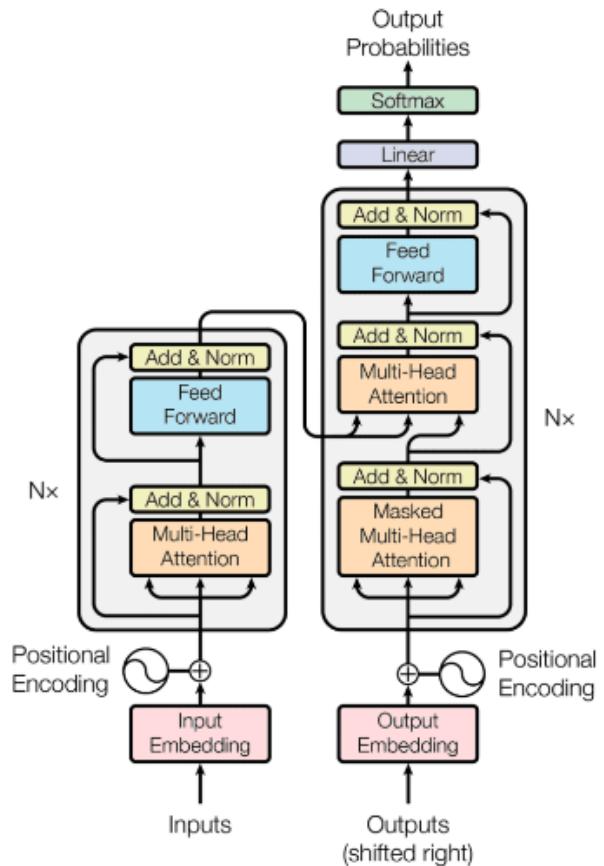
LLM architectural secret (or known) sauce

Transformer - simple parallelizable attention mechanism

A. Vaswani, et al. Attention is All You Need, 2017

Transformer architecture

- encoding-decoding architecture
 - input embedding space → multi-head & mult-layer representation space → output embedding space
- additive positional encoding - information regarding order of words @ input embedding
- multi-layer and multi-head attention followed by addition / normalization & feed forward (FF) layers
- *(relatively simple) attentions*
 - single-head (scaled dot-product) / multi-head attention
 - self attention / encoder-decoder attention
 - masked attention
- benefits
 - *evaluate dependencies between arbitrarily distant words*
 - has recurrent nature w/o recurrent architecture → parallelizable → fast w/ additional cost in computation



Single-head scaled dot-product attention

- values/keys/queries denote value/key/query *vectors*, d_k & d_v are lengths of keys/queries & vectors
- we use *standard* notions for matrices and vectors - not transposed version that (almost) all ML scientists (wrongly) use
- output: weighted-average of values where weights are attentions among tokens
- assume n queries and m key-value pairs

$$Q \in \mathbf{R}^{d_k \times n}, K \in \mathbf{R}^{d_k \times m}, V \in \mathbf{R}^{d_v \times m}$$

- attention! outputs n values (since we have n queries)

$$\text{Attention}(Q, K, V) = V \text{softmax} \left(K^T Q / \sqrt{d_k} \right) \in \mathbf{R}^{d_v \times n}$$

- *much simpler attention mechanism than previous work*
 - attention weights were output of complicated non-linear NN

Single-head - close look at equations

- focus on i th query, $q_i \in \mathbf{R}^{d_k}$, $Q = [\quad - \quad q_i \quad - \quad] \in \mathbf{R}^{d_k \times n}$
- assume m keys and m values, $k_1, \dots, k_m \in \mathbf{R}^{d_k}$ & $v_1, \dots, v_m \in \mathbf{R}^{d_v}$

$$K = [\ k_1 \ \ \cdots \ \ k_m \] \in \mathbf{R}^{d_k \times m}, V = [\ v_1 \ \ \cdots \ \ v_m \] \in \mathbf{R}^{d_v \times m}$$

- then

$$K^T Q / \sqrt{d_k} = \left[\begin{array}{ccc} & & \vdots \\ - & k_j^T q_i / \sqrt{d_k} & - \\ & & \vdots \end{array} \right]$$

e.g., dependency between i th output token and j th input token is

$$a_{ij} = \exp \left(k_j^T q_i / \sqrt{d_k} \right) / \sum_{j=1}^m \exp \left(k_j^T q_i / \sqrt{d_k} \right)$$

- value obtained by i th query, q_i in $\text{Attention}(Q, K, V)$

$$a_{i,1}v_1 + \cdots + a_{i,m}v_m$$

Multi-head attention

- evaluate h single-head attentions (in parallel)
- d_e : dimension for embeddings
- embeddings

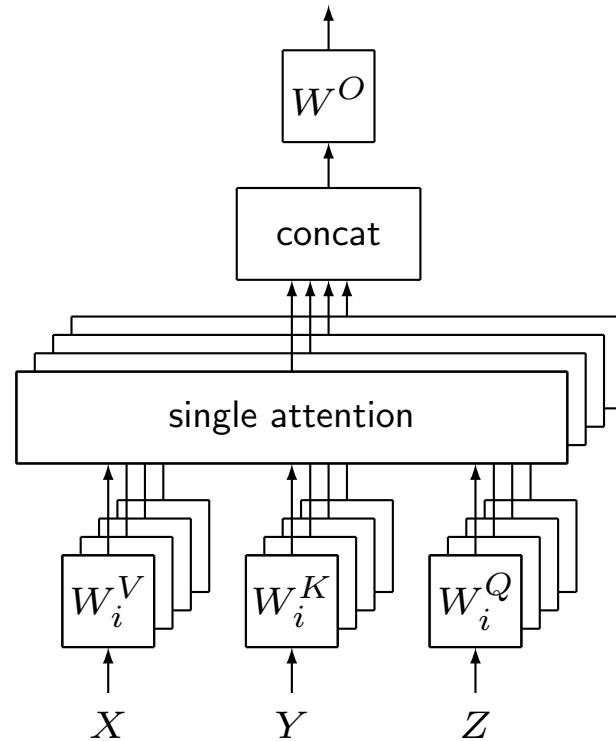
$$X \in \mathbb{R}^{d_e \times m}, Y \in \mathbb{R}^{d_e \times m}, Z \in \mathbb{R}^{d_e \times n}$$

e.g., n : input sequence length & m : output sequence length in machine translation

- h key/query/value weight matrices: $W_i^K, W_i^Q \in \mathbb{R}^{d_k \times d_e}$, $W_i^V \in \mathbb{R}^{d_v \times d_e}$ ($i = 1, \dots, h$)
- linear output layers: $W^O \in \mathbb{R}^{d_e \times hdv}$
- *multi-head attention!*

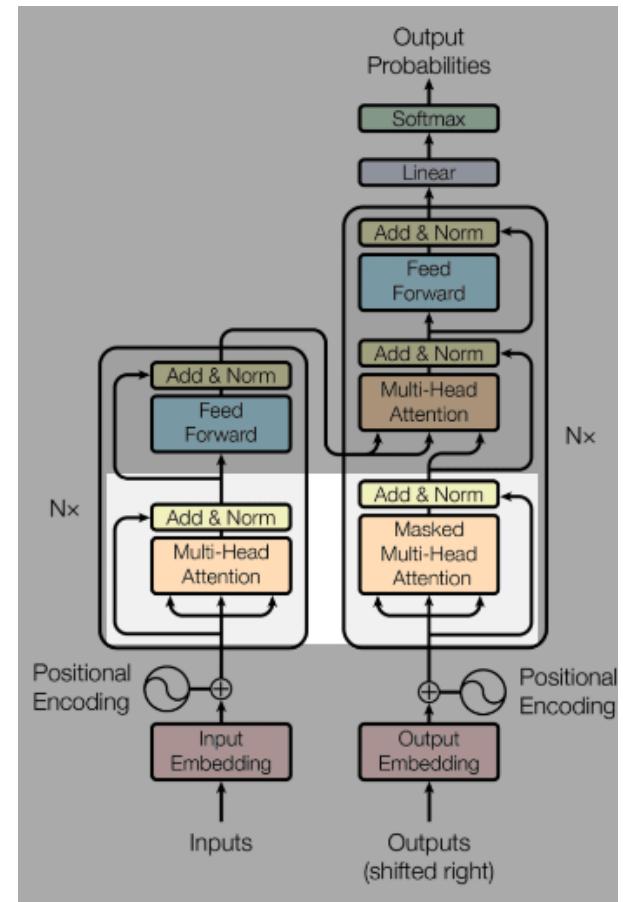
$$W^O \begin{bmatrix} A_1 \\ \vdots \\ A_h \end{bmatrix} \in \mathbb{R}^{d_e \times n},$$

$$A_i = \text{Attention}(W_i^Q Z, W_i^K Y, W_i^V X) \in \mathbb{R}^{d_v \times n}$$



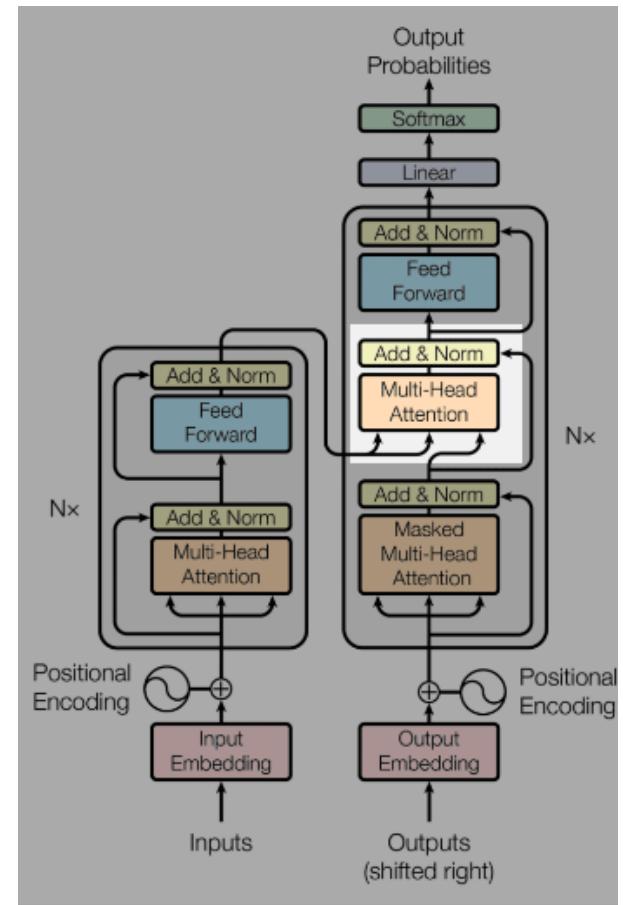
Self attention

- $m = n$
- encoder
 - keys & values & queries (K, V, Q) come from same place (from previous layer)
 - every token attends to every other token in input sequence
- decoder
 - keys & values & queries (K, V, Q) come from same place (from previous layer)
 - every token attends to other tokens up to that position
 - prevent leftward information flow to right to preserve causality
 - assign $-\infty$ for illegal connections in softmax (masking)



Encoder-decoder attention

- m : length of input sequence
- n : length of output sequence
- n queries (Q) come from previous decoder layer
- m keys / m values (K, V) come from output of encoder
- every token in output sequence attends to every token in input sequence

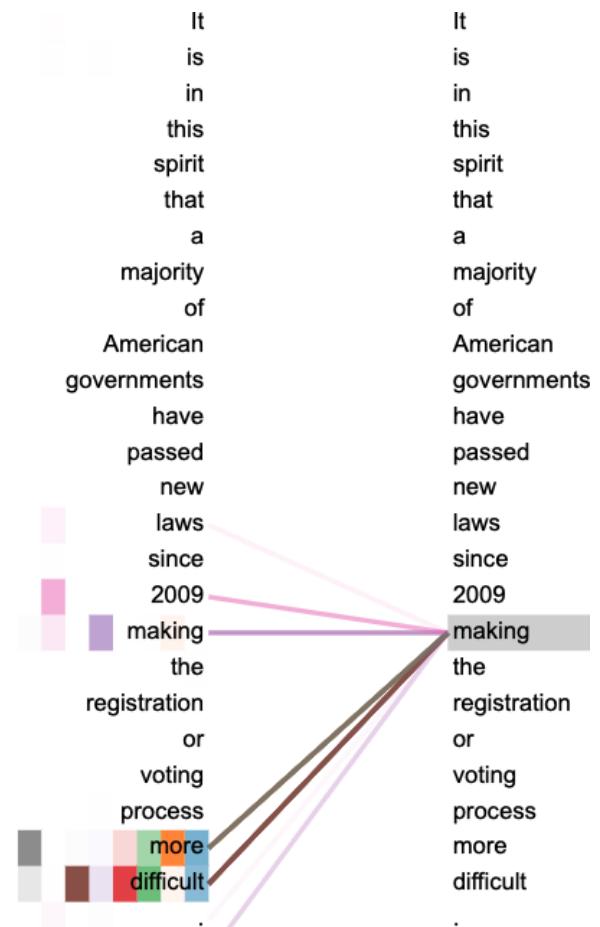


Visualization of self attentions

example sentence

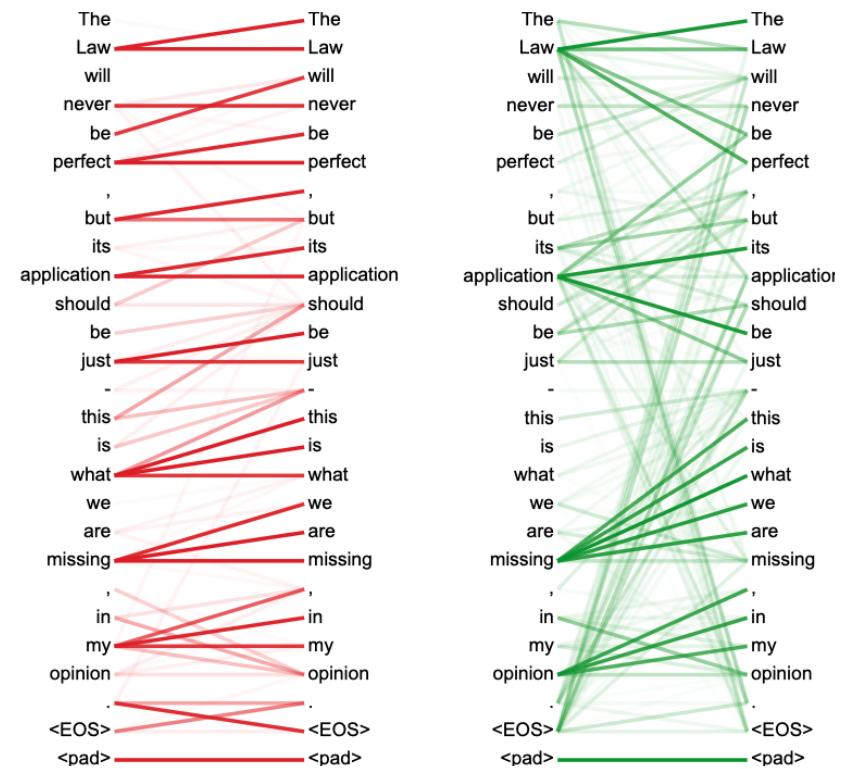
"It is in this spirit that a majority of American governments have passed new laws since 2009 making the registration or voting process more difficult."

- self attention of encoder (of a layer)
 - right figure
 - show dependencies between "making" and other words
 - different columns of colors represent different heads
 - "making" has strong dependency to "2009", "more", and "difficult"

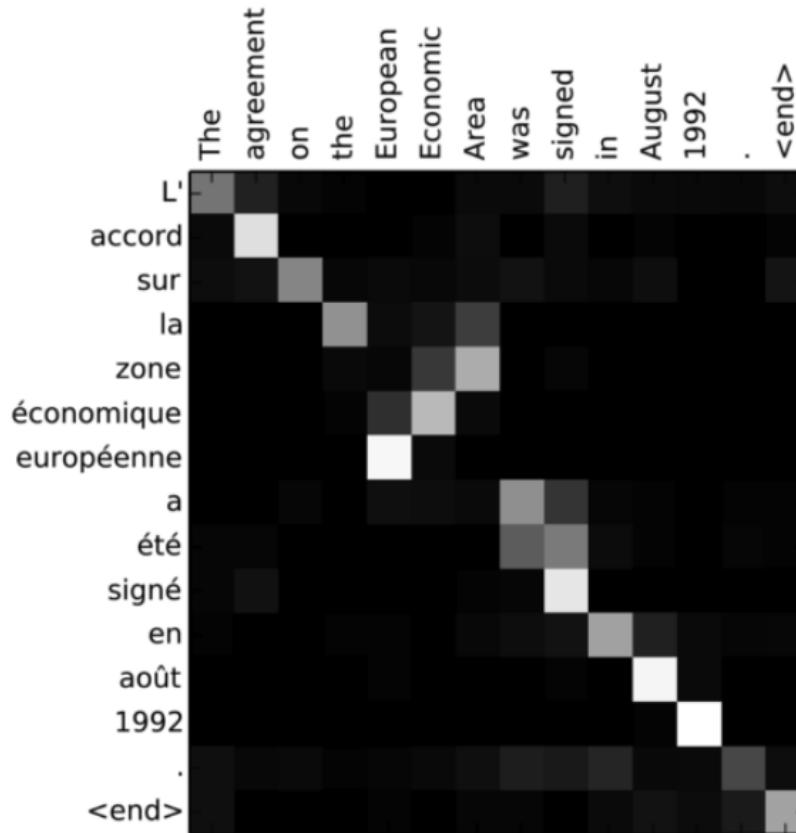


Visualization of multi-head self attentions

- self attentions of encoder for two heads (of a layer)
 - different heads represent different structures
→ advantages of multiple heads
 - multiple heads work together to collectively yield good results
 - dependencies *not* have absolute meanings (like embeddings in collaborative filtering)
 - randomness in resulting dependencies exists due to stochastic nature of ML training



Visualization of encoder-decoder attentions



- machine translation: English → French
 - input sentence: “The agreement on the European Economic Area was signed in August 1992.”
 - output sentence: “L’ accord sur la zone économique européenne a été signé en août 1992.”
- encoder-decoder attention reveals relevance between
 - European ↔ européenne
 - Economic ↔ européenne
 - Area ↔ zone

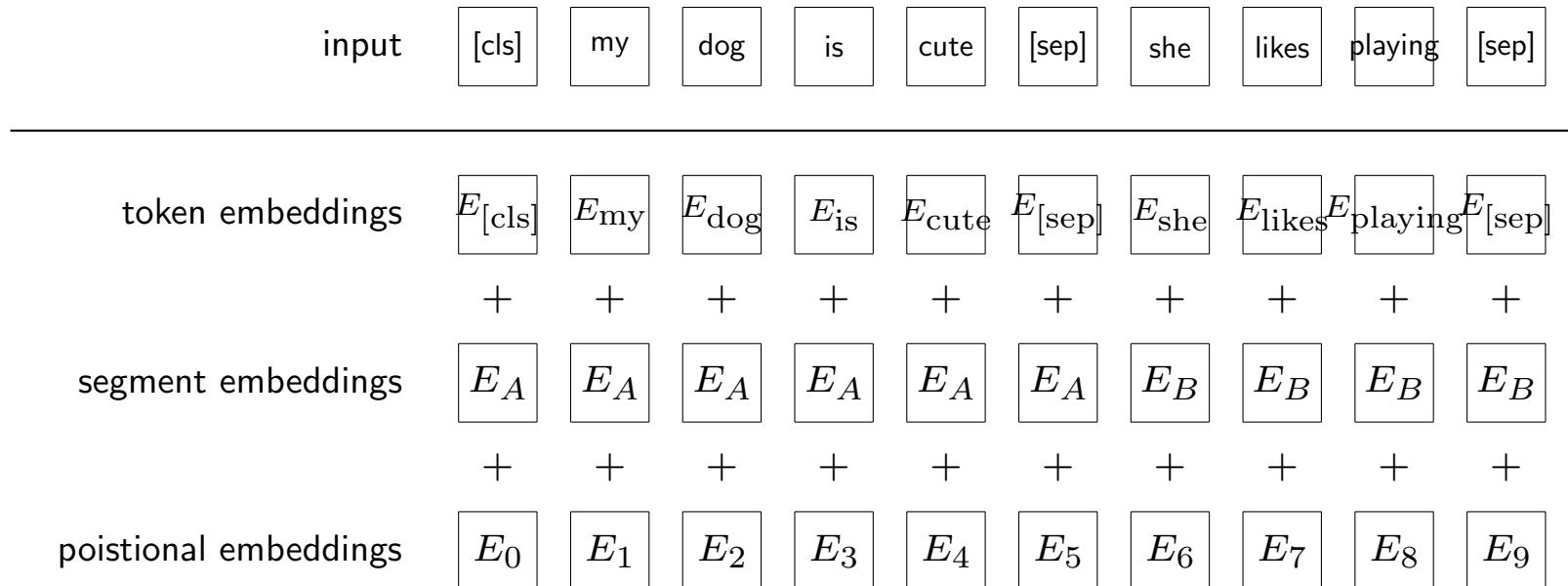
Model complexity

- computational complexity
 - n : sequence length, d : embedding dimension
 - complexity per layer - self-attention: $\mathcal{O}(n^2d)$, recurrent: $\mathcal{O}(1)$
 - sequential operations - self-attention: $\mathcal{O}(1)$, recurrent: $\mathcal{O}(n)$
 - maximum path length - self-attention: $\mathcal{O}(1)$, recurrent: $\mathcal{O}(n)$
- *massive parallel processing, long context windows*
 - makes NVidia more competitive, hence profitable!
 - makes SK Hynix prevail HBM market!

Variants of Transformer

Bidirectional encoder representations from transformers (BERT)

- Bidirectional Encoder Representations from Transformers [DCLT19]
- pre-train deep bidirectional representations from unlabeled text
- fine-tunable for multiple purposes



Challenges in LLMs

- *hallucination - can give entirely plausible outcome that is false*
- data poison attack
- unethical or illegal content generation
- huge resource necessary for both training & inference
- model size - need compact models
- outdated knowledge - can be couple of years old
- lack of reproducibility
- *biases - more on this later . . .*

do not, though, focus on downsides but on *infinite possibilities!*

- it evolves like internet / mobile / electricity
- only “tip of the iceberg” found & released

genAI

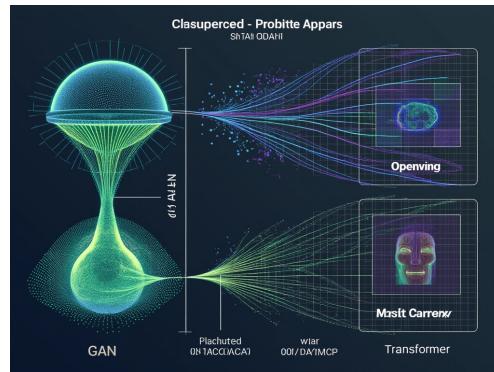
Definition of genAI

Generative AI

- genAI refers to systems capable of producing new (& original) contents based on patterns learned from training data (representation learning)
 - as opposed to discriminative models for, *e.g.*, classification, prediction & regression
 - here content can be text, images, audio, video, *etc.* - what about smell & taste?
- genAI model examples
 - generative adversarial networks (GANs), variational autoencoders (VAEs), diffusion models, Transformers



by Midjourney



by Grok 2 mini



by Generative AI Lab

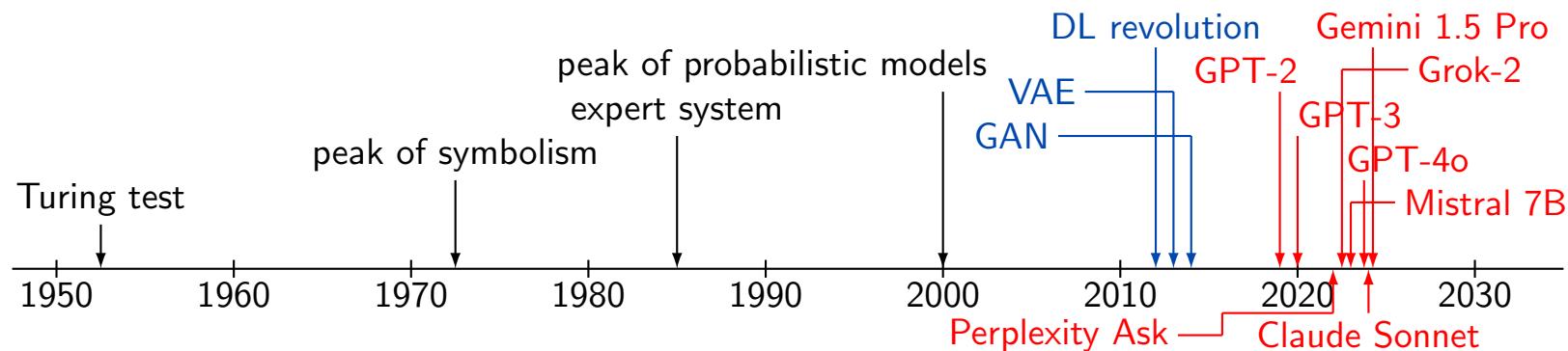
Examples of genAI in action

- text generation
 - Claude, ChatGPT, Mistral, Perplexity, Gemini, Grok
 - conversational agent writing articles, code & even poetry
- image generation
 - DALL-E - creates images based on textual descriptions
 - Stable Diffusion - uses diffusion process to generate high-quality images from text prompts (by denoising random noise)
 - MidJourney - art and visual designs generated through deep learning
- music generation
 - Amper Music - generates unique music compositions
- code generation
 - GitHub Copilot - generates code snippets based on natural language prompts

History of genAI

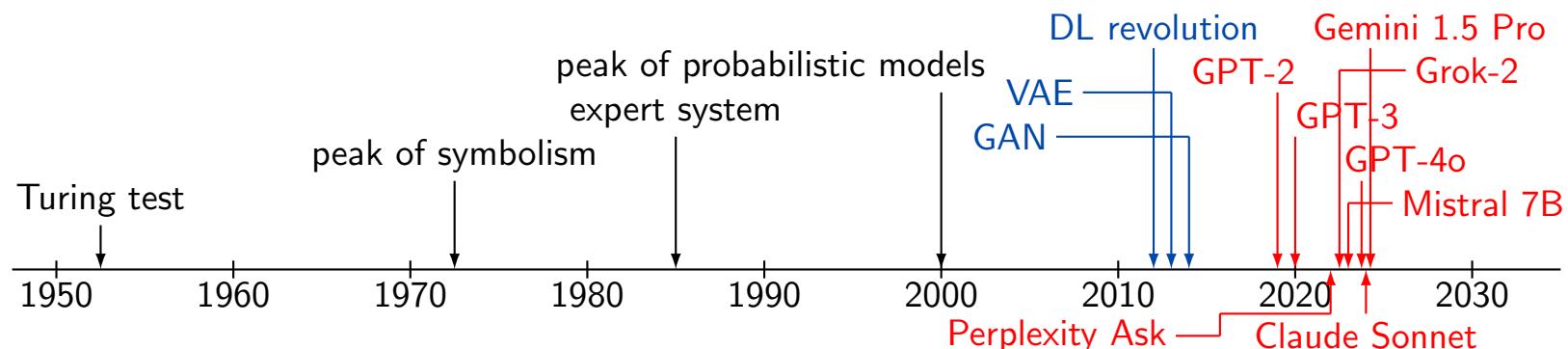
Birth of AI - early foundations & precursor technologies

- 1950s ~ 1970s
 - Alan Turing - concept of “*thinking machine*” & *Turing test* to evaluate machine intelligence (1950s)
 - *symbolists* (as opposed to connectionists) - early AI focused on symbolic reasoning, logic & problem-solving - Dartmouth Conference in 1956 by *John McCarthy, Marvin Minsky, Allen Newell & Herbert A. Simon*
 - precursor technologies - genetic algorithms (GAs), Markov chains & *hidden Markov models (HMMs)* - laying foundation for generative processes (1970s ~)



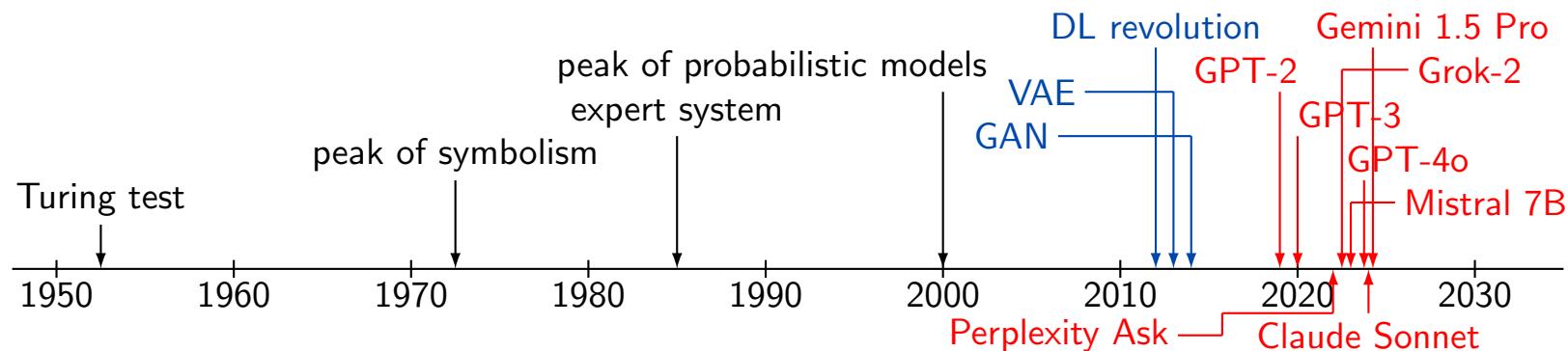
Rule-based systems & probabilistic models

- 1980s ~ early 2000s
 - *expert systems* (1980s) - AI systems designed to mimic human decision-making in specific domains
 - development of neural networks (NN) w/ backpropagation *training multi-layered networks* - setting stage for way more complex generative models
 - *probabilistic models* (including network models, *i.e.*, Bayesian networks) & Markov models - laying groundwork for data generation & pattern prediction



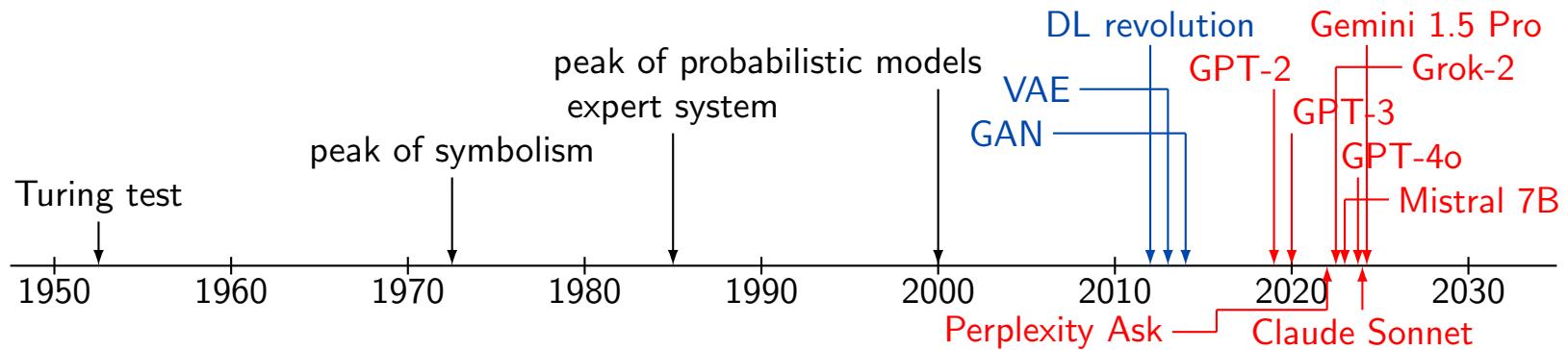
Rise of deep learning & generative models

- 2010s - breakthrough in genAI
 - *deep learning (DL) revolution* - advances in GPU computing and data availability led to the rapid development of deep neural networks.
 - *variational autoencoder (VAE)* (2013) - by Kingma and Welling - learns mappings between input and latent spaces
 - *generative adversarial network (GAN)* (2014) - by Ian Goodfellow - game-changer in generative modeling where two NNs compete each other to create realistic data
 - widely used in image generation & creative tasks



Transformer models & multimodal AI

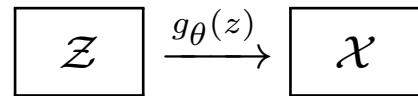
- late 2010s ~ Present
 - Transformer architecture (2017) - by Vaswani et al.
 - *revolutionized NLP*, e.g., LLM & various genAI models
 - GPT series - generative pre-trained transformer
 - GPT-2 (2019) - generating human-like texts - *marking leap in language models*
 - GPT-3 (2020) - 175B params - set *new standards for LLM*
 - multimodal systems - DALL-E & CLIP (2021) - *linking text and visual data*
 - emergence of diffusion models (2020s) - new approach for generating high-quality images - progressively “denoising” random noise (DALL-E 2 & Stable Diffusion)



Mathy Views on genAI

genAI models

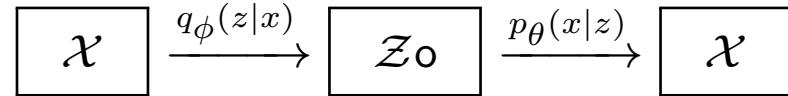
- definition of generative model



- *generate samples in original space, \mathcal{X} , from samples in latent space, \mathcal{Z}*
- g_θ is parameterized model *e.g.*, CNN / RNN / Transformer / diffuction-based model
- training
 - finding θ that minimizes/maximizes some (statistical) loss/merit function so that $\{g_\theta(z)\}_{z \in \mathcal{Z}}$ generates plausible point in \mathcal{X}
- inference
 - random samples z to generated target samples $x = g_\theta(z)$
 - *e.g.*, image, text, voice, music, video

VAE - early genAI model

- variational auto-encoder (VAE) [KW19]



- log-likelihood & ELBO - for any $q_\phi(z|x)$

$$\begin{aligned}
 \log p_\theta(x) &= \mathbf{E}_{z \sim q_\phi(z|x)} \log p_\theta(x) = \mathbf{E}_{z \sim q_\phi(z|x)} \log \frac{p_\theta(x, z)}{q_\phi(z|x)} \cdot \frac{q_\phi(z|x)}{p_\theta(z|x)} \\
 &= \mathcal{L}(\theta, \phi; x) + D_{KL}(q_\phi(z|x) \| p_\theta(z|x)) \geq \mathcal{L}(\theta, \phi; x)
 \end{aligned}$$

- (indirectly) maximize likelihood by maximizing evidence lower bound (ELBO)

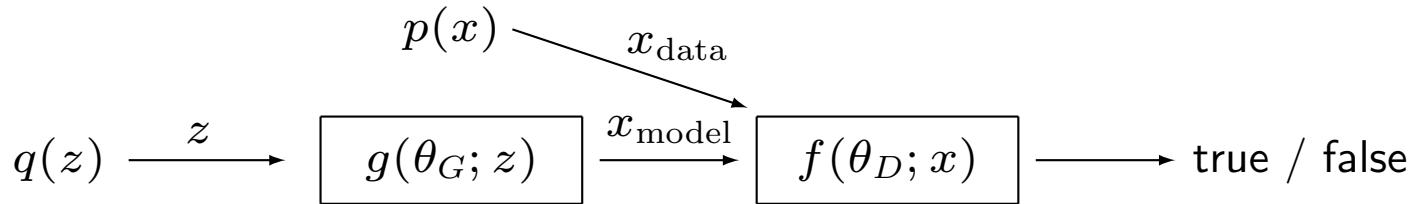
$$\mathcal{L}(\theta, \phi; x) = \mathbf{E}_{z \sim q_\phi(z|x)} \log \frac{p_\theta(x, z)}{q_\phi(z|x)}$$

- generative model

$$p_\theta(x|z)$$

GAN - early genAI model

- generative adversarial networks (GAN) [GPAM⁺14]



- value function

$$V(\theta_D, \theta_G) = \mathbf{E}_{x \sim p(x)} \log f(\theta_D; x)) + \mathbf{E}_{z \sim q(z)} \log(1 - f(\theta_D; g(\theta_G; z)))$$

- modeling via playing min-max game

$$\min_{\theta_G} \max_{\theta_D} V(\theta_D, \theta_G)$$

- generative model

$$g(\theta_G; z)$$

- variants: conditional / cycle / style / Wasserstein GAN

genAI - LLM

- *maximize conditional probability*

$$\underset{\theta}{\text{maximize}} \ d(p_{\theta}(x_t|x_{t-1}, x_{t-2}, \dots), p_{\text{data}}(x_t|x_{t-1}, x_{t-2}, \dots))$$

where $d(\cdot, \cdot)$ distance measure between probability distributions

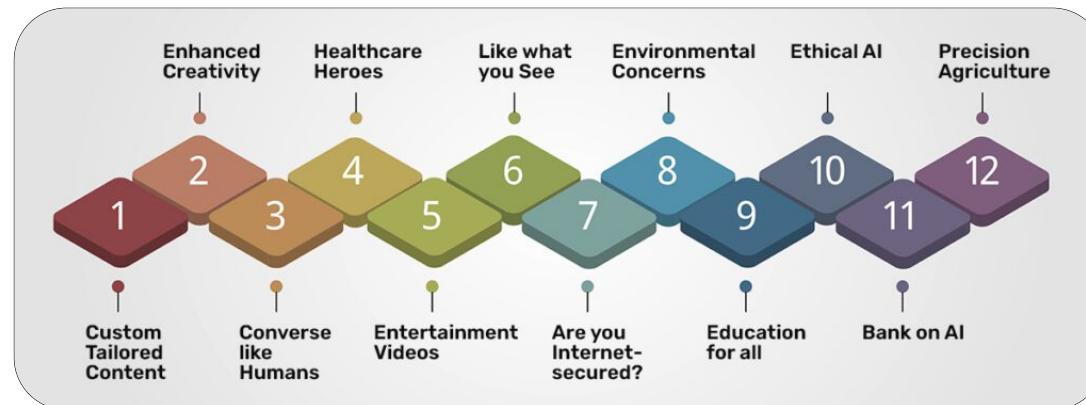
- previous sequence: x_{t-1}, x_{t-2}, \dots
- next token: x_t
- p_{θ} represented by (extremely) complicated model
 - e.g., containing multi-head & multi-layer Transformer architecture inside
- model parameters, e.g., for Llama2

$$\theta \in \mathbf{R}^{70,000,000,000}$$

Current Trend & Future Perspectives

Current trend of genAI

- rapid advancement in language models & multimodal AI capabilities
- rise of AI-assisted creativity & productivity tools
- growing adoption across industries
 - creative industries - design, entertainment, marketing, software development
 - life sciences - healthcare, medical, biotech
- infrastructure & accessibility, *e.g.*, Hugging Face democratizes AI development
- integration with cloud platforms & enterprise-level tools
- increased focus on AI ethics & responsible development



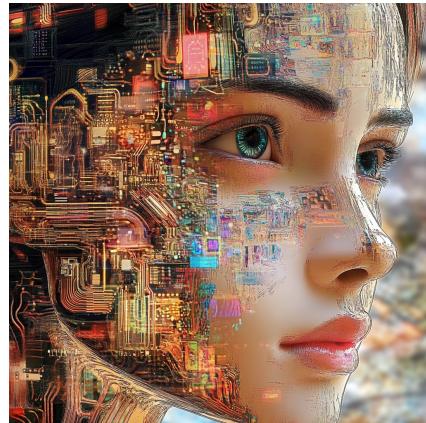
Industry & business impacts

- how genAI is transforming industries
 - creative industries - content creation - advertising, gaming, film
 - life science - enhance research, drug discovery & personalized treatments
 - finance - automating document generation, risk modeling & fraud detection
 - manufacturing & Design - rapid prototyping, 3D modeling & optimization
 - business operations - automate routine tasks to boost productivity



Future perspectives of genAI

- hyper-personalization - highly personalized content for individual users - music, products & services
- AI ethics & governance - concerns over deepfakes, misinformation & bias
- interdisciplinary synergies - integration with other fields such as quantum computing, neuroscience & robotics
- human-AI collaboration - augment human creativity rather than replace it
- energy efficiency - have to figure out how to dramatically reduce power consumption



Selected References & Sources

Selected references & sources

- Robert H. Kane “Quest for Meaning: Values, Ethics, and the Modern Experience” 2013
- Michael J. Sandel “Justice: What’s the Right Thing to Do?” 2009
- Daniel Kahneman “Thinking, Fast and Slow” 2011
- Yuval Noah Harari “Sapiens: A Brief History of Humankind” 2014
- M. Shanahan “Talking About Large Language Models” 2022
- A.Y. Halevy, P. Norvig, and F. Pereira “Unreasonable Effectiveness of Data” 2009
- A. Vaswani, et al. “Attention is all you need” @ NeurIPS 2017
- S. Yin, et. al. “A Survey on Multimodal LLMs” 2023
- Chris Miller “Chip War: The Fight for the World’s Most Critical Technology” 2022
- CEOs, CTOs, CFOs, COOs, CMOs & CCOs @ startup companies in Silicon Valley
- VCs on Sand Hill Road - Palo Alto, Menlo Park, Woodside in California, USA

References

References

- [Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.
- [BV04] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, New York, NY, USA, 2004.
- [DCLT19] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2019.
- [GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. The MIT Press, 2016.
- [GPAM⁺14] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

- [HTF01] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer, 2001.
- [KF09] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009.
- [KW19] Diederik P. Kingma and Max Welling. An introduction to variational autoencoders. *Foundations and Trends in Machine Learning*, 12(4):307–392, 2019.
- [LG94] Alberto Leon-Garcia. *Probability and Random Processes for Electrical Engineering*. Addison-Wesley, 2nd edition, 1994.
- [Mur12] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.
- [SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Bradford Books, 2nd edition, 2018.
- [VSP⁺17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you

need. In *Proceedings of 31st Conference on Neural Information Processing Systems (NIPS)*, 2017.

Thank You