

1 Probability

Sum Rule $P(X = x_i) = \sum_{j=1}^J p(X = x_i, Y = y_i)$

Product rule $P(X, Y) = P(Y|X)P(X)$

Independence $P(X, Y) = P(X)P(Y)$

Conditional Independence $P(X, Y|Z) = P(X|Z)P(Y|Z)$

Bayes' Rule $P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)} = \frac{\prod_{i=1}^k P(X_i|Y_i)P(Y)}{\prod_{i=1}^k P(X_i)}$

Cond. Ind. $X \perp Y|Z \implies P(X, Y|Z) = P(X|Z)P(Y|Z)$

Cond. Ind. $X \perp Y|Z \implies P(X|Y, Z) = P(X|Z)$

$\mathbb{E}[X] = \int_X t \cdot f_X(t) dt =: \mu_X$

$\text{Cov}(X, Y) = \mathbb{E}_{x,y}[(X - \mathbb{E}_x[X])(Y - \mathbb{E}_y[Y])]$

$\text{Cov}(X) := \text{Cov}(X, X) = \text{Var}[X]$

X, Y independent $\implies \text{Cov}(X, Y) = 0$

$\mathbf{XX}^T \geq 0$ (symmetric positive semidefinite)

$\text{Var}[X] = \mathbb{E}[X^2] - \mathbb{E}[X]^2$

$\text{Var}[\mathbf{AX}] = \mathbf{A} \text{Var}[X] \mathbf{A}^T \quad \text{Var}[aX + b] = a^2 \text{Var}[X]$

$\text{Var} \left[\sum_{i=1}^n a_i X_i \right] = \sum_{i=1}^n a_i^2 \text{Var}[X_i] + 2 \sum_{i < j} a_i a_j \text{Cov}(X_i, X_j)$

$\frac{\partial}{\partial t} P(X \leq t) = \frac{\partial}{\partial t} F_X(t) = f_X(t)$ (derivative of c.d.f. is p.d.f.)

$f_{aY}(z) = \frac{1}{\alpha} f_Y(\frac{z}{\alpha})$

T. (Moment Generating Function)

The moment generating function (MGF) $\psi_X(t) = \mathbb{E}[e^{tX}]$ characterizes the distribution of a random variable X .
 $B(e^t) = pe^t + (1-p)$

$\mathcal{N}(\mu, \sigma^2) \exp\left(\mu t + \frac{1}{2}\sigma^2 t^2\right)$

$\text{Bin}(n, p) \quad (pe^t + (1-p))^n$

$\text{Gam}(\alpha, \beta) \quad \left(\frac{1}{a - \beta t}\right)$

for $t < 1/\beta$

$\text{Pois}(\lambda) \quad e^{\lambda(e^t - 1)}$

T. If X_1, \dots, X_n are indep. rvs with MGFs $M_{X_i}(t) = \mathbb{E}[e^{tX_i}]$, then the MGF of $Y = \sum_{i=1}^n a_i X_i$ is $M_Y(t) = \prod_{i=1}^n M_{X_i}(a_i t)$.

T. Let X, Y be indep., then the p.d.f. of $Z = X + Y$ is the conv. of the p.d.f. of X and Y : $f_Z(z) = \int_{\mathbb{R}} f_X(x) f_Y(z-x) dt = \int_{\mathbb{R}} f_X(x) f_Y(z-x) dt$

D. (Normal Distribution)

$\mathcal{N}(\mathbf{x}; \mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^d \det(\Sigma)}} \exp(-\frac{1}{2}(\mathbf{x} - \mu)^T \Sigma^{-1} (\mathbf{x} - \mu))$

$\hat{\mu} = \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i \quad \hat{\Sigma} = \frac{1}{n} \sum_{i=1}^n (\mathbf{x}_i - \hat{\mu})(\mathbf{x}_i - \hat{\mu})^T$

T. $P(\begin{bmatrix} \mathbf{a}_1 \\ \vdots \\ \mathbf{a}_k \end{bmatrix}) = \mathcal{N}(\begin{bmatrix} \mathbf{a}_1 \\ \vdots \\ \mathbf{a}_k \end{bmatrix} \mid \begin{bmatrix} \mathbf{U}_1 \\ \vdots \\ \mathbf{U}_k \end{bmatrix}, \begin{bmatrix} \Sigma_{11} & \cdots & \Sigma_{1k} \\ \vdots & \ddots & \vdots \\ \Sigma_{k1} & \cdots & \Sigma_{kk} \end{bmatrix})$

$\mathbf{a}_1, \mathbf{u}_1 \in \mathbb{R}^e, \Sigma_{11} \in \mathbb{R}^{e \times e}$ p.s.d.

$\Sigma_{12} \in \mathbb{R}^{e \times f}$ p.s.d.

$\mathbf{a}_2, \mathbf{u}_2 \in \mathbb{R}^f, \Sigma_{22} \in \mathbb{R}^{f \times f}$ p.s.d.

$\Sigma_{21} \in \mathbb{R}^{f \times e}$ p.s.d.

$P(\mathbf{a}_2 | \mathbf{a}_1 = \mathbf{z}) = \mathcal{N}(\mathbf{a}_2 | \mathbf{u}_2 + \Sigma_{21} \Sigma_{11}^{-1} (\mathbf{z} - \mathbf{u}_1), \Sigma_{22} - \Sigma_{21} \Sigma_{11}^{-1} \Sigma_{12})$

T. (Chebyshev) Let X be a rv with $\mathbb{E}[X] = \mu$ and variance $\text{Var}[X] = \sigma^2 < \infty$. Then for any $\epsilon > 0$, we have $P(|X - \mu| \geq \epsilon) \leq \frac{\sigma^2}{\epsilon^2}$.

2 Analysis

Log-Trick (Identity): $\nabla_{\theta} [\log(p_{\theta}(\mathbf{x}))] = \frac{\partial}{\partial \mathbf{x}} \log(p_{\theta}(\mathbf{x})) [\log(p_{\theta}(\mathbf{x}))]$

T. (Cauchy-Schwarz)

$\forall \mathbf{u}, \mathbf{v} \in V: \langle \mathbf{u}, \mathbf{v} \rangle \leq \|\mathbf{u}\| \|\mathbf{v}\|$.

$\forall \mathbf{u}, \mathbf{v} \in V: 0 \leq \langle \mathbf{u}, \mathbf{v} \rangle \leq \|\mathbf{u}\| \|\mathbf{v}\|$.

Special case: $(\sum x_i y_i)^2 \leq (\sum x_i^2)(\sum y_i^2)$.

Special case: $\mathbb{E}[XY]^2 \leq \mathbb{E}[X^2]\mathbb{E}[Y^2]$.

D. (Convex Set) A set $S \subseteq \mathbb{R}^d$ is called *convex* if $\forall \mathbf{x}, \mathbf{x}' \in S, \forall \lambda \in [0, 1]: \lambda \mathbf{x} + (1 - \lambda)\mathbf{x}' \in S$.

Com. Any point on the line between two points is within the set. \mathbb{R}^d is convex.

D. (Convex Function) A function $f: S \rightarrow \mathbb{R}$ defined on a convex set $S \subseteq \mathbb{R}^d$ is called *convex* if $\forall \mathbf{x}, \mathbf{x}' \in S, \lambda \in [0, 1]: f(\lambda \mathbf{x} + (1 - \lambda)\mathbf{x}') \leq \lambda f(\mathbf{x}) + (1 - \lambda)f(\mathbf{x}')$

Com. A function is strictly convex if the line segment between any two points on the graph of the function lies strictly above the graph. This guarantees that there is a unique global minimum.

T. (Properties of Convex Functions)

• $f(y) \geq f(x) + \nabla f(x)^T (y - x)$

• $f'(x) \geq 0$

• Local minima are global minima, strictly convex functions have a unique global minimum

• If f, g are convex then $\alpha f + \beta g$ is convex for $\alpha, \beta \geq 0$

• If f, g are convex then $\max(f, g)$ is convex

• If f is convex and g is convex and non-decreasing then $g \circ f$ is convex

D. (Strongly Convex Function) A function f is μ -strongly convex if it curves up at least as much as a quadratic function with curvature $\mu > 0$. For all x, y :

$f(y) \geq f(x) + \nabla f(x)^T (y - x) + \frac{\mu}{2} \|y - x\|^2$

Relation to Optimization:

• Guarantee: Ensures a unique global minimum exists.

• Convergence: Gradient Descent on strongly convex (and Lipschitz smooth) functions guarantees a linear convergence rate ($O(\epsilon^k)$ for some $c < 1$).

Condition Number: The convergence speed depends on the condition number $\kappa = L/\mu$. If κ is large (poor conditioning), convergence slows down.

D. (Condition Number) The condition number $\kappa(A)$ measures the sensitivity of a function's output to small perturbations in the input. For a symmetric positive semi-definite matrix (like the Hessian H of a loss function), it is the ratio of the largest to the smallest eigenvalue:

$$\kappa(H) = \frac{|\lambda_{\max}|}{|\lambda_{\min}|} \geq 1$$

Implications for Optimization:

• **Well-conditioned** ($\kappa \approx 1$): The contours of the loss function are nearly spherical. Gradient Descent converges quickly and directly toward the minimum.

• **Ill-conditioned** ($\kappa \gg 1$): The contours form narrow, elongated ellipses (steep valleys). Gradient Descent tends to oscillate ("zigzag") across the narrow valley rather than moving down the slope, leading to very slow convergence.

A. C, D, a, b, e not a function of x.

$\mathbf{f} = \mathbf{f}(\mathbf{x}), \mathbf{g} = \mathbf{g}(\mathbf{x}), \mathbf{h} = \mathbf{h}(\mathbf{x}), u = u(\mathbf{x}), v = v(\mathbf{x})$

$\frac{\partial}{\partial \mathbf{x}} [\mathbf{u}(\mathbf{x}) \mathbf{f}(\mathbf{x})] = u(\mathbf{x}) \frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}} + \mathbf{f}(\mathbf{x}) \frac{\partial u(\mathbf{x})}{\partial \mathbf{x}}$

$\frac{\partial}{\partial \mathbf{x}} [\mathbf{v}(\mathbf{x}) \mathbf{f}(\mathbf{x})] = \frac{\partial \mathbf{v}(\mathbf{x})}{\partial \mathbf{x}} \mathbf{f}(\mathbf{x}) + \mathbf{f}(\mathbf{x}) \frac{\partial \mathbf{v}(\mathbf{x})}{\partial \mathbf{x}}$

$\frac{\partial}{\partial \mathbf{x}} [\mathbf{a}(\mathbf{x}) \mathbf{f}(\mathbf{x})] = \mathbf{a}(\mathbf{x}) \frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}} + \mathbf{f}(\mathbf{x}) \frac{\partial \mathbf{a}(\mathbf{x})}{\partial \mathbf{x}}$

$\frac{\partial}{\partial \mathbf{x}} [\mathbf{b}(\mathbf{x}) \mathbf{f}(\mathbf{x})] = \mathbf{b}(\mathbf{x}) \frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}} + \mathbf{f}(\mathbf{x}) \frac{\partial \mathbf{b}(\mathbf{x})}{\partial \mathbf{x}}$

$\frac{\partial}{\partial \mathbf{x}} [\mathbf{c}(\mathbf{x}) \mathbf{f}(\mathbf{x})] = \mathbf{c}(\mathbf{x}) \frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}} + \mathbf{f}(\mathbf{x}) \frac{\partial \mathbf{c}(\mathbf{x})}{\partial \mathbf{x}}$

$\frac{\partial}{\partial \mathbf{x}} [\mathbf{d}(\mathbf{x}) \mathbf{f}(\mathbf{x})] = \mathbf{d}(\mathbf{x}) \frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}} + \mathbf{f}(\mathbf{x}) \frac{\partial \mathbf{d}(\mathbf{x})}{\partial \mathbf{x}}$

$\frac{\partial}{\partial \mathbf{x}} [\mathbf{e}(\mathbf{x}) \mathbf{f}(\mathbf{x})] = \mathbf{e}(\mathbf{x}) \frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}} + \mathbf{f}(\mathbf{x}) \frac{\partial \mathbf{e}(\mathbf{x})}{\partial \mathbf{x}}$

$\frac{\partial}{\partial \mathbf{x}} [\mathbf{f}(\mathbf{x}) \mathbf{g}(\mathbf{x})] = \mathbf{f}(\mathbf{x}) \frac{\partial \mathbf{g}(\mathbf{x})}{\partial \mathbf{x}} + \mathbf{g}(\mathbf{x}) \frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}}$

$\frac{\partial}{\partial \mathbf{x}} [\mathbf{g}(\mathbf{x}) \mathbf{h}(\mathbf{x})] = \mathbf{g}(\mathbf{x}) \frac{\partial \mathbf{h}(\mathbf{x})}{\partial \mathbf{x}} + \mathbf{h}(\mathbf{x}) \frac{\partial \mathbf{g}(\mathbf{x})}{\partial \mathbf{x}}$

$\frac{\partial}{\partial \mathbf{x}} [\mathbf{h}(\mathbf{x}) \mathbf{u}(\mathbf{x})] = \mathbf{h}(\mathbf{x}) \frac{\partial \mathbf{u}(\mathbf{x})}{\partial \mathbf{x}} + \mathbf{u}(\mathbf{x}) \frac{\partial \mathbf{h}(\mathbf{x})}{\partial \mathbf{x}}$

$\frac{\partial}{\partial \mathbf{x}} [\mathbf{u}(\mathbf{x}) \mathbf{v}(\mathbf{x})] = \mathbf{u}(\mathbf{x}) \frac{\partial \mathbf{v}(\mathbf{x})}{\partial \mathbf{x}} + \mathbf{v}(\mathbf{x}) \frac{\partial \mathbf{u}(\mathbf{x})}{\partial \mathbf{x}}$

$\frac{\partial}{\partial \mathbf{x}} [\mathbf{u}(\mathbf{x}) \mathbf{w}(\mathbf{x})] = \mathbf{u}(\mathbf{x}) \frac{\partial \mathbf{w}(\mathbf{x})}{\partial \mathbf{x}} + \mathbf{w}(\mathbf{x}) \frac{\partial \mathbf{u}(\mathbf{x})}{\partial \mathbf{x}}$

$\frac{\partial}{\partial \mathbf{x}} [\mathbf{u}(\mathbf{x}) \mathbf{v}(\mathbf{x}) \mathbf{w}(\mathbf{x})] = \mathbf{u}(\mathbf{x}) \frac{\partial \mathbf{v}(\mathbf{x}) \mathbf{w}(\mathbf{x})}{\partial \mathbf{x}} + \mathbf{v}(\mathbf{x}) \frac{\partial \mathbf{u}(\mathbf{x}) \mathbf{w}(\mathbf{x})}{\partial \mathbf{x}} + \mathbf{w}(\mathbf{x}) \frac{\partial \mathbf{u}(\mathbf{x}) \mathbf{v}(\mathbf{x})}{\partial \mathbf{x}}$

$\frac{\partial}{\partial \mathbf{x}} [\mathbf{u}(\mathbf{x}) \mathbf{v}(\mathbf{x}) \mathbf{w}(\mathbf{x}) \mathbf{d}(\mathbf{x})] = \mathbf{u}(\mathbf{x}) \frac{\partial \mathbf{v}(\mathbf{x}) \mathbf{w}(\mathbf{x}) \mathbf{d}(\mathbf{x})}{\partial \mathbf{x}} + \mathbf{v}(\mathbf{x}) \frac{\partial \mathbf{u}(\mathbf{x}) \mathbf{w}(\mathbf{x}) \mathbf{d}(\mathbf{x})}{\partial \mathbf{x}} + \mathbf{w}(\mathbf{x}) \frac{\partial \mathbf{u}(\mathbf{x}) \mathbf{v}(\mathbf{x}) \mathbf{d}(\mathbf{x})}{\partial \mathbf{x}} + \mathbf{d}(\mathbf{x}) \frac{\partial \mathbf{u}(\mathbf{x}) \mathbf{v}(\mathbf{x}) \mathbf{w}(\mathbf{x})}{\partial \mathbf{x}}$

$\frac{\partial}{\partial \mathbf{x}} [\mathbf{u}(\mathbf{x}) \mathbf{v}(\mathbf{x}) \mathbf{w}(\mathbf{x}) \mathbf{d}(\mathbf{x}) \mathbf{e}(\mathbf{x})] = \mathbf{u}(\mathbf{x}) \frac{\partial \mathbf{v}(\mathbf{x}) \mathbf{w}(\mathbf{x}) \mathbf{d}(\mathbf{x}) \mathbf{e}(\mathbf{x})}{\partial \mathbf{x}} + \mathbf{v}(\mathbf{x}) \frac{\partial \mathbf{u}(\mathbf{x}) \mathbf{w}(\mathbf{x}) \mathbf{d}(\mathbf{x}) \mathbf{e}(\mathbf{x})}{\partial \mathbf{x}} + \mathbf{w}(\mathbf{x}) \frac{\partial \mathbf{u}(\mathbf{x}) \mathbf{v}(\mathbf{x}) \mathbf{d}(\mathbf{x}) \mathbf{e}(\mathbf{x})}{\partial \mathbf{x}} + \mathbf{d}(\mathbf{x}) \frac{\partial \mathbf{u}(\mathbf{x}) \mathbf{v}(\mathbf{x}) \mathbf{w}(\mathbf{x}) \mathbf{e}(\$

11.3 Gradient Descent

D. (Gradient Descent (GD)) Iteratively moves parameters θ in the direction of the negative gradient of the loss function $J(\theta)$.

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta J(\theta_t)$$

Com. In Stochastic GD (SGD), the gradient is approximated using a single sample (or mini-batch) to introduce noise and escape local minima. Although, the gradient is unbiased it adds variance, this can help to escape local minima and saddle points.

Com. Gradient Flow can be seen as the numerical integration of the continuous-time ordinary differential equation (ODE) $\dot{x} = -\nabla f(x)$.

D. (Polyak Averaging (Averaged SGD)) Instead of using the final parameter vector θ_T , this method uses the arithmetic mean of the parameters traversed during training.

$$\bar{\theta}_t = \frac{1}{t} \sum_{i=1}^t \theta_i \quad \text{or} \quad \bar{\theta}_t = (1-\beta)\bar{\theta}_{t-1} + \beta\theta_t$$

Benefits:

- It effectively increases the effective batch size and reduces the variance of the estimate.
- Allows the use of larger learning rates (longer steps) while still converging to the optimal solution asymptotically.
- Often achieves the optimal convergence rate of $O(1/t)$ for convex problems.

D. (Learning Rate Condition) (Robbins-Monro Conditions), for Stochastic Gradient Descent (SGD) to guarantee convergence to a local minimum (in non-convex cases) or global minimum (in convex cases), the step size schedule α_t must satisfy two conditions:

1. **Explore Forever:** The steps must sum to infinity to ensure the algorithm can reach the optimum from any starting point, no matter how far.

$$\sum_{t=1}^{\infty} \alpha_t = \infty$$

2. **Decay Fast Enough:** The squared steps must sum to a finite value to ensure the variance (noise) of the updates tends to zero, preventing the parameters from oscillating forever around the minimum.

$$\sum_{t=1}^{\infty} \alpha_t^2 < \infty$$

Example: A schedule of $\alpha_t = \frac{1}{t}$ satisfies both, whereas $\alpha_t = \frac{1}{\sqrt{t}}$ satisfies the first but not the second.

D. (Momentum) Accelerates SGD by navigating along the relevant direction and softening oscillations in irrelevant directions. It maintains a velocity vector v (exponential moving average of past gradients).

$$\begin{aligned} \theta^{t+1} &= \theta^t - \eta \nabla J(\theta^t) + \beta(\theta^t - \theta^{t-1}) \\ \text{where } \beta \in [0, 1] \text{ is the momentum term (friction).} \end{aligned}$$

D. (Nesterov Accelerated Gradient) A "look ahead" version of GD, also called NAG. It computes the gradient at the *approximate future position* of the parameters rather than the current position.

$$\begin{aligned} \theta^{t+1} &= \theta^t + \beta(\theta^t - \theta^{t-1}) \\ \theta^{t+1} &= \theta^{t+1} - \eta \nabla f(\theta^{t+1}) \end{aligned}$$

D. (Adaptive Learning Rate Methods) These methods adjust the learning rate for each parameter individually, scaling them based on the history of gradient magnitudes.

D. (RMSPProp) Designed to resolve the diminishing learning rates of Adagrad. It uses a decaying average of squared gradients.

$$\begin{aligned} E[g^2]_t &= \beta E[g^2]_{t-1} + (1-\beta)(\nabla J(\theta_t))^2 \\ \theta_{t+1} &= \theta_t - \frac{\alpha}{\sqrt{E[g^2]_t + \epsilon}} \nabla J(\theta_t) \end{aligned}$$

D. (Adam (Adaptive Moment Estimation)) Combines Momentum (first moment m_t) and RMSProp (second moment v_t). It also includes bias correction terms \hat{m}_t, \hat{v}_t to account for initialization at zero.

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) \nabla J(\theta_t) \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) (\nabla J(\theta_t))^2 \\ \theta_{t+1} &= \theta_t - \frac{\alpha}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t \end{aligned}$$

12 Convolutional Neural Networks

- 12.1 Convolutional Layers

D. (Transform) A transform T is a mapping from one function space \mathcal{F} to another function space \mathcal{F}' . So $T: \mathcal{F} \rightarrow \mathcal{F}'$.

D. (Linear Transform) A transform T is linear, if for all functions f, g and scalars α, β , $T(\alpha f + \beta g) = \alpha(Tf) + \beta(Tg)$.

D. (Integral Transform) An *integral transform* is any transform T of the following form

$$(Tf)(u) = \int_{t_1}^{t_2} K(t, u) f(t) dt.$$

Com. The fourier transform is an example of an integral transform.

T. Any integral transform is a linear transform.

D. (Convolution) Given two functions $f, h: \mathbb{R} \rightarrow \mathbb{R}$, their convolution is defined as

$$(f * h)(u) := \int_{-\infty}^{\infty} h(t) f(u-t) dt = \int_{-\infty}^{\infty} h(u-t) f(t) dt$$

Com. Whether the convolution exists depends on the properties of f and h (the integral might diverge).

However, a typical use is $f = \text{signal}$, and $h = \text{fast decaying kernel function}$.

T. (Convolution Theorem) Any linear, translation-invariant transformation T can be written as a *convolution* with a suitable h .

T. (Convs are commutative and associative)

T. (Convs are shift-invariant), we define $f_\Delta(t) := f(t + \Delta)$. Then

$$(f_\Delta * h)(u) = (f * h)_\Delta(u)$$

The derivative $\frac{\partial \mathcal{R}}{\partial y^{(l)}_k}$ is analogous.

D. (Fourier Transform) The fourier transform of a function f is defined as

$$(\mathcal{F}f)(u) := \int_{-\infty}^{\infty} f(t) e^{-2\pi i u t} dt$$

and its inverse as

$$(\mathcal{F}^{-1}f)(u) := \int_{-\infty}^{\infty} f(t) e^{2\pi i u t} dt$$

Com. Convolutional operators can be efficiently computed with point wise multiplication using the Fourier transform.

$$\mathcal{F}(f * h) = \mathcal{F}f \cdot \mathcal{F}h$$

and then transformed back using the inverse Fourier transform.

$$\mathcal{F}^{-1}(\mathcal{F}(f * h)) = \mathcal{F}^{-1}(\mathcal{F}f \cdot \mathcal{F}h) = f * h$$

- 12.2 Discrete Time Convolutions

D. (Discrete Convolution)

For $f, h: \mathbb{Z} \rightarrow \mathbb{R}$, we can define the discrete convolution via

$$(f * h)[u] := \sum_{t=-\infty}^{\infty} f[t] h[u-t] = \sum_{t=-\infty}^{\infty} f[u-t] h[t]$$

Com. Note that the use of rectangular brackets suggests that we're using "arrays" (discrete-time samples).

Com. Typically we use a h with finite support (window size).

D. (Multidimensional Discrete Convolution)

For $f, h: \mathbb{R}^d \rightarrow \mathbb{R}$ we have

$$(f * h)[u_1, \dots, u_d] = \sum_{t_1=-\infty}^{\infty} \dots \sum_{t_d=-\infty}^{\infty} f(t_1, \dots, t_d) h(u_1 - t_1, \dots, u_d - t_d) = \sum_{t_1=-\infty}^{\infty} \dots \sum_{t_d=-\infty}^{\infty} f(u_1 - t_1, \dots, u_d - t_d) h(t_1, \dots, t_d)$$

D. (Discrete Cross-Correlation)

Let $f, h: \mathbb{Z} \rightarrow \mathbb{R}$, then

$$(h * f)[u] := \sum_{t=-\infty}^{\infty} h[t] f[u+t] = \sum_{t=-\infty}^{\infty} h[-t] f[u-t]$$

$$(h * f)[u] = (f * \bar{h})[u] \quad \text{where } \bar{h}(t) = h(-t).$$

aka "sliding inner product", non-commutative, kernel "flipped over" ($u+t$ instead of $u-t$). If kernel symmetric: cross-correlation = convolution.

- 12.3 Convolution via Matrices

Represent the input signal, the kernel and the output as *vectors*. Copy the kernel as columns into the matrix offsetting it by one more very time (gives a band matrix (special case of Toeplitz matrix)). Then the convolution is just a matrix-vector product.

- 12.4 Border Handling

There are different options to do this

D. (Padding of p) Means we extend the image (or each dimension) by p on both sides ($sos + 2p$) and just fill in a constant there (e.g., zero).

D. (Same Padding) Padding with zeros = *same padding* ("same" constant, i.e., 0, and we'll get a tensor of the "same" dimensions)

D. (Valid Padding) Only retain values from windows that are fully-contained within the support of the signal f (see 2D example below) = *valid padding*

- 12.5 Backpropagation for Convolutions

D. (Receptive Field I_i^l of x_i^l)

The receptive field I_i^l of node x_i^l is defined as $I_i^l := \{j \mid W_{ij}^l \neq 0\}$ where W^l is the Toeplitz matrix of the convolution at layer l .

Com. Hence, the receptive field of a node x_i^l are just nodes which are connected to it and have a non-zero weight.

Com. One may extend the definition of the receptive field over several layers. The further we go back in layer, the bigger the receptive field becomes due to the nested convolutions. The receptive field may be even the entire image after a few layers. Hence, the convolutions have to be small.

We have $\forall j \neq I_i^l: \frac{\partial x_i^l}{\partial x_j^l} = 0$,

Due to *weight-sharing*, the kernel weight h_i^l is re-used for every unit in the target layer at layer l , so when computing the derivative $\frac{\partial \mathcal{R}}{\partial h_i^l}$ we just build an additive combination of all the derivatives (note that some of them might be zero).

D. (U-Net (2015)) Designed for Biomedical Image Segmentation (pixel-wise classification).

Structure: Symmetrical Encoder-Decoder architecture (U-shape).

Encoder: Contracting path (Convs + Max Pooling) to capture context.

Decoder: Expansive path (Up-Convs) to enable precise localization.

Skip Connections: Concatenates high-resolution features from the encoder directly to the decoder to recover spatial details lost during downsampling.

- 12.6 Gated Memory

Gates use multiplicative interactions (sigmoid σ) to control information flow, stabilizing gradients and allowing long-term memory.

D. (LSTMs): Long Short Term Memory models maintain a separate cell state C_t controlled by three gates.

$$C^t = \underbrace{\sigma(F\tilde{x}^t) \odot C^{t-1}}_{\text{Forget}} + \underbrace{\sigma(I\tilde{x}^t) \odot \tanh(\tilde{C}\tilde{x}^t)}_{\text{Input/Update}}$$

$$z^t = \sigma(O\tilde{x}^t) \odot \tanh(C^t)$$

where $\tilde{x}^t = [x_t, z_{t-1}]$.

15 Attention & Transformers

We generally work under the *token paradigm* (sequence of tokens in embedding-form).

Memory

D. (Dilated Convolutions) To handle long sequences without losing resolution (pooling), *dilation* introduces gaps between kernel elements.

Receptive Field: Grows exponentially with the dilation factor d ($1, 2, 4, 8, \dots$), allowing the network to capture long-range dependencies with few layers.

D. (WaveNet (2016)) A deep generative model for raw audio waveforms.

Causal Convolutions: Strict ordering ensures the prediction at time t only depends on samples $x_{<t}$ (cannot see the future).

D. (GRU): Gated Recurrent Unit models simplify LSTMs by merging Cell/Hidden states and Forget/Input gates.

$$z^t = (1 - \Gamma_u) \odot z^{t-1} + \Gamma_u \odot \tilde{z}^t$$

where $\Gamma_u = \sigma(G[x_t, z_{t-1}])$ is the update gate.

GRU

12.10 Comparison of #Parameters (CNNs, FC, LC)

Ex. Here we have

- an input signal that is 2D with 3 channels ($7 \times 7 \times 3$) (image x channels)
- K convolution kernels: $p \times q$ (valid padding and stride 1) output dimensions: $(m-p+1) \times (n-q+1) \times K$
- #parameters CNN: $K(pqc + 1)$
- #parameters of fully-conn. NN with same number of outputs as CNN: $nmc((m-p+1)(n-q+1) + 1)K$
- #parameters of locally-conn. NN with same connections as CNN: $pqc((m-p+1)(n-q+1) + 1)K$

13 Recurrent Neural Networks

- 13.1 Simple Recurrent Networks

D. (Concept): Unlike CNNs (fixed filter widths), RNNs model temporal/sequence data of variable length. They maintain a hidden state z_t acting as a "memory" of the history.

Dynamics: Uses a linear recurrence relation (diagonalizable):

$$z^{t+1} = Az^t + Bx^t$$

Formulation: Given input sequence x_1, \dots, x_T :

$$z_t = \phi(Uz_{t-1} + Vx_t)$$

$$\hat{y}_t = \psi(Wz_t)$$

where U, V, W are shared weight matrices and ϕ, ψ are non-linearities.

Unrolling: An RNN is equivalent to a deep feedforward network with T layers and *shared weights*.

Backpropagation Through Time (BPTT): Gradients are propagated backward through the unrolled graph. Since weights are shared, the total gradient is the sum of gradients at each time step:

$$\frac{\partial L}{\partial w} = \sum_{t=1}^T \frac{\partial L}{\partial \hat{y}_t} \frac{\partial \hat{y}_t}{\partial z_t} \frac{\partial z_t}{\partial w}$$

Teacher Forcing: During training, feed the *ground truth* y_{t-1}^* as input to step t rather than the model's own prediction \hat{y}_{t-1} . This speeds up convergence but causes "explosion bias" (train-test discrepancy).

Seq2Seq (Encoder-Decoder):

- **Encoder:** Compresses input sequence into a fixed context vector z_T .
- **Decoder:** Generates output sequence from z_T .
- **Attention:** Allows Decoder to "look back" at specific Encoder states z_T via learned weights, solving the bottleneck of a fixed-size context vector.

14 Optimization

Optimization is the process of finding the best solution to a problem.

T. We have the following chain of inclusions for functions over a *closed* and *bounded* (i.e., compact) subset of the real line.

Continuously differentiable \subseteq Lipschitz continuous \subseteq (Uniformly) continuous

T. If we use Nesterov acceleration (in the general case), then we get a polynomial convergence rate of $\mathcal{O}(t^{-2})$.

Com. The trick used in the Nesterov approach is *momentum*.

D. (Inception Network) Focused on computational efficiency and network "width". It was developed by Google in 2014.

Inception Module: Instead of choosing a filter size, it performs $1 \times 1, 3 \times 3$, and 5×5 convolutions (and pooling) in *parallel* and concatenates the outputs.

1x1 Convolutions: Used as "Bottleneck layers" to reduce dimensionality (depth) before expensive operations, significantly reducing computational cost.

D. (VGG Network (2014)) Focused on the effect of network depth using a uniform architecture.

Philosophy: Replace large filters (e.g., $5 \times 5, 7 \times 7$) with stacks of small 3×3 filters.

Reasoning: Two stacked 3×3 layers have the same receptive field as a 5×5 layer but with fewer parameters and more non-linearities (ReLU between layers).

D. (AlexNet (2012)) The breakthrough model that popularized Deep Learning on ImageNet.

Structure: Deeper than LeNet (5 Conv layers, 3 FC layers). Used large filters initially (11×11).

Innovations: First large-scale use of ReLU (to solve vanishing gradients), Dropout (for regularization), and Data Augmentation. Trained on GPUs.

D. (LeNet-5 (1998)) The pioneering CNN for handwritten digit recognition (MNIST).

12.9.1 Classic CNN Architectures

D. (Multi-Headed Attention)

runs multiple attention mechanisms in parallel (r **heads**). Replicates matrices U_k, U_Q, W_r , r times, perform the attention-based propagation $X \mapsto \Xi_i$ ($1 \leq i \leq r$) and then concatenate the matrices Ξ_i along the feature dimension.

T. (Feature Transformation)

In order to gain more representational power, one also needs to allow for a learnable transformation of features. Transformers do this following the network-within-network design pattern by applying a per-token transformation with a feed-forward network (MLP). F_θ is an MLP applied columnwise to Ξ :

$$X \mapsto \Xi \mapsto F_\theta(\Xi), \quad F_\theta(\Xi) = (F_\theta(\xi_1), \dots, F_\theta(\xi_T)).$$

When we add residual connections and normalization layers, we arrive at a typical encoder block of a Transformer.

16.2 Graph Convolutional Networks (GCN)

Operates on Graph $G = (V, \mathcal{E})$ with feature matrix $X \in \mathbb{R}^{M \times F}$.

D. Graph Definitions:

- **Adjacency Matrix (A):** Symmetric $M \times M$ matrix. $A_{nm} = 1$ if $\{n, m\} \in \mathcal{E}$, else 0. Zeros on diagonal.
- **Degree Matrix (D):** Diagonal matrix $D = \text{diag}(d_1, \dots, d_M)$ where $d_m = \sum_n A_{nm}$ (number of neighbors).
- **Coupling Matrix (\bar{A}):** Standard symmetric normalized formulation. We define the self-loop adjacency $\bar{A} = A + I$ and corresponding degree matrix $\bar{D}_{mm} = \sum_n \bar{A}_{nm} = d_m + 1$.

$$\bar{A} = \bar{D}^{-\frac{1}{2}} \bar{A} \bar{D}^{-\frac{1}{2}} = \bar{D}^{-\frac{1}{2}} (A + I) \bar{D}^{-\frac{1}{2}}$$

D. Layer Update: Combines neighborhood aggregation ($\bar{A}X$) and feature transformation (W).

$$X' = \sigma(\bar{A}XW)$$

Com. Limitations:

- **Oversmoothing:** In deep GCNs, repeated mixing causes all node embeddings to converge to the same value.
- **Oversquashing:** Exponentially growing information from distant nodes fails to fit into fixed-size vectors.

- 16.3 Spectral Graph Theory

Generalizes convolutions using the Graph Laplacian (L), discrete curvature).

D. Laplacian: Defined using the Degree matrix D and Adjacency A .

$$L = D - A \quad \text{or Normalized: } \tilde{L} = I - D^{-\frac{1}{2}} AD^{-\frac{1}{2}}$$

D. Spectral Convolution: Uses the Convolution Theorem via the Graph Fourier Transform (Eigenvectors U of L).

$$x * y = U((U^T x) \odot (U^T y))$$

D. ChebNet: Approximates spectral filters using Chebyshev polynomials T_k to avoid expensive Eigendecomposition ($O(N^3)$). It is strictly K -localized.

$$g_\theta(L) \approx \sum_{k=0}^K \theta_k T_k(\tilde{L})$$

16.4 — Attention GNNs (GAT)

Learns dynamic edge weights e_{ij} instead of static adjacency.

D. Attention Mechanism:

- Score: $e_{ij} = \text{LeakyReLU}(\mathbf{a}^T [Wx_i || Wx_j])$
- Normalize: $\alpha_{ij} = \text{softmax}(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})}$
- Aggregate: $x'_i = \sigma \left(\sum_{j \in \mathcal{N}_i} \alpha_{ij} Wx_j \right)$

17 Theory

— 17.1 — Neural Tangent Kernel (NTK)

D. Linearized DNN: To analyze non-linear DNNs, we can linearize them around initialization θ_0 using a first-order Taylor expansion:

$$h(\beta)(x) = f(x; \theta_0) + \beta \cdot \nabla_\theta f(x; \theta_0)$$

Here, $\nabla_\theta f(x; \theta_0)$ acts as a fixed, random feature map determined by initialization. The optimization of β becomes a convex problem.

D. NTK Definition: The kernel corresponding to these gradient feature maps is the Neural Tangent Kernel:

$$k(x, \xi) = \langle \nabla_\theta f(x), \nabla_\theta f(\xi) \rangle$$

It encodes the similarity between samples x and ξ based on how much their predictions change when parameters are updated.

D. Infinite Width Limit (NTK Regime): As network width $m \rightarrow \infty$ (under specific "NTK parameter scaling"):

- Deterministic Limit: The initial NTK converges to a deterministic kernel k_∞ that depends only on the initialization law, not the specific random weights.
- NTK Constancy: The kernel remains constant during training ($\frac{d}{dt} k = 0$). This means the feature map does not evolve ("Lazy Training").
- Linear Dynamics: The training dynamics become identical to Kernel Ridge Regression with kernel k_∞ . The evolution of outputs follows:

$$\dot{f} = K(\theta)(y - f)$$

— 17.2 — Bayesian DNNs

D. Bayesian Paradigm: Instead of finding a point estimate θ^* , we compute a posterior distribution $p(\theta|S)$ to capture uncertainty.

$$p(\theta|S) \propto p(S|\theta)p(\theta)$$

Predictions are made by marginalizing over the posterior:

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

D. Langevin Dynamics: Since exact inference is intractable, we use sampling. Langevin dynamics injects noise into Gradient Descent to explore the posterior distribution (sampling from $p(\theta|S)$) rather than collapsing to a minimum.

$$v_{t+1} = (1 - \eta\gamma)v_t - \eta\nabla\tilde{E}(\theta) + 2\gamma\eta\epsilon, \epsilon \sim \mathcal{N}(0, I)$$

This mimics a physical system with friction and thermal noise.

— 17.3 — Gaussian Processes (GPs)

D. Infinite Width Equivalence (Neal's Theorem): A single-hidden-layer neural network with infinite width ($m \rightarrow \infty$) and i.i.d. priors on weights converges to a Gaussian Process (GP).

Com. Mechanism: By the Central Limit Theorem, the pre-activations (sums of many independent random variables) become Gaussian.

Com. Result: The network output $f(x)$ is a draw from a GP with mean $\mu(x)$ and a specific kernel $k(x, x')$.

D. Deep GPs: This equivalence extends to deep networks. The kernel is defined recursively:

$$K_t(x, x') = E[\phi(f_{t-1}(x))\phi(f_{t-1}(x'))]$$

where the expectation is taken over the GP of the previous layer f_{t-1} .

— 17.4 — Statistical Learning Theory

D. Generalization Error: The gap between performance on training data (empirical risk) and unseen data (expected risk).

$$\text{Gen}(f) = R[f] - R_{\text{emp}}[f]$$

Classical theory (VC-dimension) predicts overfitting for huge models, but DNNs exhibit Double Descent: test error decreases, rises (at interpolation threshold), and then decreases again as width grows.

D. PAC-Bayesian Bounds: Provides generalization bounds for stochastic classifiers (posterior Q) based on their distance from a prior P (KL-divergence).

$$EQ[R(f)] \leq R_{\text{emp}}(Q) + 2sKL(Q||P) + \ln(2s/\delta)$$

Com. This links generalization to the "flatness" of minima: flat minima allow for a posterior Q with high variance (large entropy) that still fits the data, minimizing the KL term.

18 Chain-Rule and Jacobians for Tensors

D. (*k*-Dimensional Tensor) $\mathbf{T} \in \mathbb{R}^{d_1 \times d_2 \times \dots \times d_k}$

D. (Tensor Multiplication)

$$\mathbf{T}_{\mathbb{R}^{(a+c)}} = \mathbf{P}_{\mathbb{R}^{(a+b)}} \times_b \mathbf{Q}_{\mathbb{R}^{(b+c)}} = \mathbf{P}_{r_1 \times \dots \times r_a \times s_1 \times \dots \times s_b} \times_{s_1, \dots, s_b} \mathbf{Q}_{s_1 \times \dots \times s_b \times t_1 \times \dots \times t_c}$$

where each entry of \mathbf{T} is computed as follows:

$$T_{i_1, \dots, i_a, k_1, \dots, k_c} := \sum_{j_1, \dots, j_b} P_{i_1, \dots, i_a, j_1, \dots, j_b} Q_{j_1, \dots, j_b, k_1, \dots, k_c}$$

Note that this is just the sum of the multiplications of two numbers which are in corresponding locations in \mathbf{P} and \mathbf{Q} . Essentially, it's the dot product across the dimensions s_1, \dots, s_b .

Note how this tensor-tensor-multiplication is isomorphic to some matrix-matrix product:

$$T_{i_1, \dots, i_a, k_1, \dots, k_c} := \sum_{j_1, \dots, j_b} P_{i_1, \dots, i_a, j_1, \dots, j_b} Q_{j_1, \dots, j_b, k_1, \dots, k_c}$$

— 19 Generative Models

— 19.1 — Variational Autoencoders (VAEs)

Latent variable models $p(x, z) = p(x|z)p(z)$ where the posterior $p(z|x)$ is intractable. VAEs approximate it using a parametric encoder $q_\phi(z|x)$.

D. (Evidence Lower Bound (ELBO)): Since $\ln p(x)$ is intractable, we maximize a lower bound (Jensen's Inequality):

$$\ln p_\theta(x) \geq \mathcal{L}(\theta, \phi; x) = \mathbb{E}_{q_\phi(z|x)} [\ln p_\theta(x|z)] - D_{KL}(q_\phi(z|x) || p(z))$$

- Encoder (q_ϕ): Maps input x to latent parameters μ, Σ .
- Decoder (p_θ): Reconstructs x from sampled z .

D. (Reparameterization Trick): To backpropagate through the stochastic node $z \sim \mathcal{N}(\mu, \Sigma)$, we move the noise outside:

$$z = \mu + \Sigma^{1/2} \odot \epsilon, \quad \text{where } \epsilon \sim \mathcal{N}(0, I)$$

This makes z a deterministic, differentiable function of ϕ and fixed noise ϵ .

— 19.2 — Normalizing Flows

Learns a bijective mapping $f: \mathcal{Z} \rightarrow \mathcal{X}$ from a simple distribution p_z (e.g., Gaussian) to the complex data distribution p_x . Allows **exact likelihood** computation.

D. Change of Variables:

$$p_x(x) = p_z(z) \left| \det \frac{\partial f^{-1}(x)}{\partial x} \right| = p_z(f^{-1}(x)) |\det J_{f^{-1}}(x)|$$

Or in log-domain (maximizing likelihood):

$$\ln p_x(x) = \ln p_z(z) - \ln \left| \det \frac{\partial f(z)}{\partial z} \right|$$

D. Coupling Layers (RealNVP): To ensure the Jacobian determinant is computationally cheap, we split variables $x_{1:D}$ and $x_{d+1:D}$:

$$y_{1:d} = x_{1:d} \\ y_{d+1:D} = x_{d+1:D} \odot \exp(s(x_{1:d})) + t(x_{1:d})$$

The Jacobian is triangular, so $\det J = \prod \exp(s(x_{1:d}))$.

— 19.3 — Gen. Adversarial Networks (GANs)

A minimax game between a Generator G (creates fakes) and Discriminator D (classifies real vs. fake).

D. (Minimax Objective):

$$\min_G \max_D V(D, G) = \mathbb{E}_{x \sim p_{\text{data}}} [\log D(x)] + \mathbb{E}_{z \sim p_z} [\log(1 - D(G(z)))]$$

Optimality:

Optimal Discriminator: For a fixed G , $D^*(x) = \frac{p_{\text{data}}(x)}{p_{\text{data}}(x) + p_g(x)}$.

Global Minimum: Achieved when $p_g = p_{\text{data}}$. The value is $-\log 4$ (related to Jensen-Shannon Divergence).

Com. Training Issues:

Vanishing Gradients: If D is perfect, $\log(1 - D(G(z)))$ saturates. Fix: Train G to maximize $\log D(G(z))$ (Non-Saturating Loss).

Mode Collapse: G maps all z to a single plausible x to cheat D .

— 19.4 — Denoising Diffusion Models (DDPM)

Learns to reverse a gradual noising process.

D. (Forward Process (Fixed)): Markov chain adding Gaussian noise according to schedule β_t :

$$q(x_t | x_{t-1}) = \mathcal{N}(x_t; \sqrt{1 - \beta_t}x_{t-1}, \beta_t I)$$

where the expectation is taken over the GP of the previous layer f_{t-1} .

— 17.4 — Statistical Learning Theory

D. Generalization Error: The gap between performance on training data (empirical risk) and unseen data (expected risk).

$$\text{Gen}(f) = R[f] - R_{\text{emp}}[f]$$

Classical theory (VC-dimension) predicts overfitting for huge models, but DNNs exhibit Double Descent: test error decreases, rises (at interpolation threshold), and then decreases again as width grows.

D. PAC-Bayesian Bounds: Provides generalization bounds for stochastic classifiers (posterior Q) based on their distance from a prior P (KL-divergence).

$$EQ[R(f)] \leq R_{\text{emp}}(Q) + 2sKL(Q||P) + \ln(2s/\delta)$$

Com. This links generalization to the "flatness" of minima: flat minima allow for a posterior Q with high variance (large entropy) that still fits the data, minimizing the KL term.

18 Chain-Rule and Jacobians for Tensors

D. (*k*-Dimensional Tensor) $\mathbf{T} \in \mathbb{R}^{d_1 \times d_2 \times \dots \times d_k}$

D. (Tensor Multiplication)

$$\mathbf{T}_{\mathbb{R}^{(a+c)}} = \mathbf{P}_{\mathbb{R}^{(a+b)}} \times_b \mathbf{Q}_{\mathbb{R}^{(b+c)}} = \mathbf{P}_{r_1 \times \dots \times r_a \times s_1 \times \dots \times s_b} \times_{s_1, \dots, s_b} \mathbf{Q}_{s_1 \times \dots \times s_b \times t_1 \times \dots \times t_c}$$

where each entry of \mathbf{T} is computed as follows:

$$T_{i_1, \dots, i_a, k_1, \dots, k_c} := \sum_{j_1, \dots, j_b} P_{i_1, \dots, i_a, j_1, \dots, j_b} Q_{j_1, \dots, j_b, k_1, \dots, k_c}$$

Note that this is just the sum of the multiplications of two numbers which are in corresponding locations in \mathbf{P} and \mathbf{Q} . Essentially, it's the dot product across the dimensions s_1, \dots, s_b .

Note how this tensor-tensor-multiplication is isomorphic to some matrix-matrix product:

— 19 Generative Models

— 19.1 — Variational Autoencoders (VAEs)

Latent variable models $p(x, z) = p(x|z)p(z)$ where the posterior $p(z|x)$ is intractable. VAEs approximate it using a parametric encoder $q_\phi(z|x)$.

D. (Evidence Lower Bound (ELBO)): Since $\ln p(x)$ is intractable, we maximize a lower bound (Jensen's Inequality):

$$\ln p_\theta(x) \geq \mathcal{L}(\theta, \phi; x) = \mathbb{E}_{q_\phi(z|x)} [\ln p_\theta(x|z)] - D_{KL}(q_\phi(z|x) || p(z))$$

where we have: $T_{i_1, \dots, i_a, k_1, \dots, k_c} = \sum_{j_1, \dots, j_b} P_{i_1, \dots, i_a, j_1, \dots, j_b} Q_{j_1, \dots, j_b, k_1, \dots, k_c}$

D. (Tensor Chain Rule)

D. (Tensor Chain Rule)

D. (Min L2 adversarial perturbation)

D. (LOFO via marginalization)

D. (LOFO via Permutation importance)

D. (LOFO via Marginal Contribution of X_j in context of S)

D. (SHAP (SHapley Additive exPlanations))

D. (Shapley Additive Global importance)

D. (Structural equation models (SEM))

D. (Causal ordering)

D. (Causal ordering)

D. (Counterfactuals)

D. (Efficiency (Additivity) Property)

$$\sum_{j=1}^p \phi_j = \mathcal{Q}(\{1, \dots, p\}) - \mathcal{Q}(\emptyset)$$

D. (Causal effect): what is the causal effect of x_j on y ?