

Sum Rule $P(X = x_i) = \sum_{j=1}^n p(X = x_i, Y = y_j)$			
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	$P(Y X) = \frac{P(X Y)P(Y)}{P(X)} = \frac{P(X Y_i)P(Y_i)}{\sum_{i=1}^k P(X Y_i)P(Y_i)}$		
Cond.	Ind.	$X \perp Y Z \implies P(X, Y Z) = P(X Z)P(Y Z)$	
$\mathbb{E}[X] = \int_{\mathcal{X}} t \cdot f_X(t) dt =: \mu_X$			
$\text{Cov}(X, Y) = \mathbb{E}_{x,y} [(X - \mathbb{E}_x[X])(Y - \mathbb{E}_y[Y])]$			
$\text{Cov}(X) := \text{Cov}(X, X) = \text{Var}[X]$			
$X, Y$ independent $\implies \text{Cov}(X, Y) = 0$			
$\mathbf{XX}^T \geq 0$ (symmetric positive semidefinite)			
$\text{Var}[X] = \mathbb{E}[X^2] - \mathbb{E}[X]^2$			
$\text{Var}[\mathbf{AX}] = \mathbf{A} \text{Var}[\mathbf{X}] \mathbf{A}^T \quad \text{Var}[aX + b] = a^2 \text{Var}[X]$			

$\text{Var}\left[\sum_{i=1}^n a_i X_i\right] = \sum_{i=1}^n a_i^2 \text{Var}[X_i] + 2 \sum_{i < j} a_i a_j \text{Cov}(X_i, X_j)$
$\frac{\partial}{\partial t} P(X \leq t) = \frac{\partial}{\partial t} F_X(t) = f_X(t)$ (derivative of c.d.f. is p.d.f)
$f_X(z) = \frac{1}{\sigma} f_Y(\frac{z}{\sigma})$
<b>T. (Moment Generating Function)</b> The moment generating function (MGF) $\psi_X(t) = \mathbb{E}[e^{t^T X}]$ characterizes the distribution of a random variable $X$ . $Be(p) \quad pe^t + (1 - p)$
$\mathcal{N}(\mu, \sigma) \quad \exp\left(\mu t + \frac{1}{2} \sigma^2 t^2\right)$
$Bin(n, p) \quad (pe^t + (1 - p))^n$
$Gam(\alpha, \beta) \quad \left(\frac{1}{a} - \beta t\right)^{-\alpha}$

$\frac{\partial}{\partial t} P(X \leq t) = \frac{\partial}{\partial t} F_X(t) = f_X(t)$ (derivative of c.d.f. is p.d.f)
$f_X(z) = \frac{1}{\sigma} f_Y(\frac{z}{\sigma})$

<b>T. (Moment Generating Function)</b> The moment generating function (MGF) $\psi_X(t) = \mathbb{E}[e^{t^T X}]$ characterizes the distribution of a random variable $X$ . $Be(p) \quad pe^t + (1 - p)$
$\mathcal{N}(\mu, \sigma) \quad \exp\left(\mu t + \frac{1}{2} \sigma^2 t^2\right)$
$Bin(n, p) \quad (pe^t + (1 - p))^n$
$Gam(\alpha, \beta) \quad \left(\frac{1}{a} - \beta t\right)^{-\alpha}$
for $t < 1/\beta$
$Pois(\lambda) \quad e^{\lambda} e^{-(1-\lambda)}$

**T.** If  $X_1, \dots, X_n$  are indep. rvs with MGFs  $M_{X_i}(t) = \mathbb{E}[e^{t^T X_i}]$ , 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  $Y$ :  $f_Z(z) = \int_{\mathbb{R}} f_X(t) f_Y(z - t) dt = \int_{\mathbb{R}} f_X(z - t) f_Y(t) dt$

<b>D. (Normal Distribution)</b> $\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{(2\pi)^d \det(\boldsymbol{\Sigma})}} \exp\left(-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x} - \boldsymbol{\mu})\right)$ $\hat{\boldsymbol{\mu}} = \frac{1}{n} \sum_{i=1}^n \mathbf{x}_i \quad \hat{\boldsymbol{\Sigma}} = \frac{1}{n} \sum_{i=1}^n (\mathbf{x} - \hat{\boldsymbol{\mu}})(\mathbf{x} - \hat{\boldsymbol{\mu}})^T$ <b>T.</b> $P\left(\begin{bmatrix} \hat{\mathbf{a}}_1 \\ \hat{\mathbf{a}}_2 \end{bmatrix}\right) = \mathcal{N}\left(\begin{bmatrix} \hat{\mathbf{a}}_1 \\ \hat{\mathbf{a}}_2 \end{bmatrix} \middle  \begin{bmatrix} \mathbf{U}_1 \\ \mathbf{U}_2 \end{bmatrix}, \begin{bmatrix} \boldsymbol{\Sigma}_{21}^{11} & \boldsymbol{\Sigma}_{21}^{12} \\ \boldsymbol{\Sigma}_{21}^{21} & \boldsymbol{\Sigma}_{22}^{22} \end{bmatrix}\right)$ $\mathbf{a}_1, \mathbf{u}_1 \in \mathbb{R}^c, \quad \boldsymbol{\Sigma}_{11} \in \mathbb{R}^{c \times c} \text{ p.s.d.}$ $\boldsymbol{\Sigma}_{12} \in \mathbb{R}^{c \times f} \text{ p.s.d.}$ $\mathbf{a}_2, \mathbf{u}_2 \in \mathbb{R}^f, \quad \boldsymbol{\Sigma}_{22} \in \mathbb{R}^{f \times f} \text{ p.s.d.}$ $\boldsymbol{\Sigma}_{21} \in \mathbb{R}^{f \times c} \text{ p.s.d.}$
--

$P(\mathbf{a}_2   \mathbf{a}_1 = \mathbf{z}) = \mathcal{N}\left(\mathbf{a}_2 \mid \mathbf{u}_2 + \boldsymbol{\Sigma}_{21} \boldsymbol{\Sigma}_{11}^{-1}(\mathbf{z} - \mathbf{u}_1), \boldsymbol{\Sigma}_{22} - \boldsymbol{\Sigma}_{21} \boldsymbol{\Sigma}_{11}^{-1} \boldsymbol{\Sigma}_{12}\right)$
--

**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
<b>Log-Trick (Identity):</b> $\nabla_{\theta} [p_{\theta}(\mathbf{x})] = p_{\theta}(\mathbf{x}) \nabla_{\theta} [\log(p_{\theta}(\mathbf{x}))]$
<b>T. (Cauchy-Schwarz)</b> $\forall \mathbf{u}, \mathbf{v} \in V: \langle \mathbf{u}, \mathbf{v} \rangle \leq  \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: $\mathbb{E}[XY]^2 \leq \mathbb{E}[X^2] \mathbb{E}[Y^2]$ .
<b>D. (Convex Set)</b> A set $S \subseteq \mathbb{R}^d$ is called <i>convex</i> if $\forall \mathbf{x}, \mathbf{x}' \in \Sigma, \forall \lambda \in [0, 1]: \quad \lambda \mathbf{x} + (1 - \lambda) \mathbf{x}' \in S$ .

**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  $\mu$ -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** ( $O(e^k)$  for some  $c < 1$ ).

- Condition Number:** The convergence speed depends on the condition number  $\kappa = L/\mu$ . If  $\kappa$  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 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.

**Com.** Momentum and Adaptive Learning Rate methods (like Adam) are specifically designed to mitigate the issues caused by high condition numbers.

<b>T. (Taylor-Lagrange Formula)</b> $f(x) = \sum_{k=0}^n \frac{f^{(k)}(x_0)}{k!} (x - x_0)^k + \int_{x_0}^x \frac{f^{(n+1)}(x-t)}{n!} dt$
<b>T. (Jensen)</b> $f$ convex/ <b>concave</b> , $\forall i: \lambda_i \geq 0, \sum_{i=1}^n \lambda_i = 1$ $f(\sum_{i=1}^n \lambda_i \mathbf{x}_i) \leq / \geq \sum_{i=1}^n \lambda_i f(\mathbf{x}_i)$ Special case: $f(\mathbb{E}[X]) \leq \mathbb{E}[f(X)]$ .
<b>D. (L-Lipschitz Continous Function)</b> Given two <i>metric spaces</i> $(X, d_X)$ and $(Y, d_Y)$ , a function $f: X \rightarrow Y$ is called <i>Lipschitz continuous</i> , if there exists a real constant $L \in \mathbb{R}_0^+$ ( <i>Lipschitz constant</i> ), such that $\forall \mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^n: \quad \ f(\mathbf{x}_1) - f(\mathbf{x}_2)\  \leq L \cdot \ \mathbf{x}_1 - \mathbf{x}_2\ .$

**Com.** If the objective function is  $L$ -smooth a step size of  $\eta = 1/L$  guarantees convergence.

**D. (Lagrangian Formulation)** of arg max $_{x,y} f(x, y)$  s.t.  $g(x, y) = c$ :  $\mathcal{L}(x, y, \gamma) = f(x, y) - \gamma(g(x, y) - c)$

**D. (PL Condition)** A differentiable function  $f(x)$  with global minimum  $f^*$  satisfies the  $\mu$ -Polyak-Łojasiewicz (PL) condition if there exists a constant  $\mu > 0$  such that for all  $x$ :

$$\frac{1}{2} \|\nabla f(x)\|^2 \geq \mu(f(x) - f^*)$$

**Significance:**

- Gradient Dominance:** It implies that the gradient magnitude dominates the suboptimality. If the gradient is small, the function value must be close to the optimal  $f^*$ .
- Convergence without Convexity:** The PL condition is weaker than strong convexity (it does not require convexity at all). However, it is sufficient to guarantee a **linear convergence rate** for Gradient Descent.
- In Deep Learning:** Over-parameterized neural networks often satisfy the PL condition in the neighborhood of a minimum, explaining fast convergence despite non-convexity.

**Convergence Rate:** Gradient Descent with step size  $\alpha = 1/L$  (where  $L$  is the Lipschitz constant) converges as:

$$f(x_k) - f^* \leq \left(1 - \frac{\mu}{L}\right)^k (f(x_0) - f^*)$$

3 Linear Algebra
Kernels are positive semi-definite matrices.
<b>D. (Positive Semi-Definite Matrix)</b> A symmetric matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is PSD if for all non-zero vectors $\mathbf{x} \in \mathbb{R}^n$ : $\mathbf{x}^T \mathbf{A} \mathbf{x} \geq 0$ . <b>Properties:</b> <ul style="list-style-type: none"><li>All eigenvalues <math>\lambda_i \geq 0</math>.</li> <li>The trace <math>\text{Tr}(A) \geq 0</math> and determinant <math>\det(A) \geq 0</math>.</li> <li>Cholesky Decomposition exists: <math>A = LL^T</math>.</li></ul>
<b>T. (Sylvester Criterion)</b> A $d \times d$ matrix is positive semi-definite if and only if all the upper left $k \times k$ for $k = 1, \dots, d$ have a positive determinant. Negative definite: $\det < 0$ for all odd-sized minors, and $\det > 0$ for all even-sized minors Otherwise: indefinite.
<b>D. (Trace)</b> of $\mathbf{A} \in \mathbb{R}^{n \times n}$ is $\text{Tr}(\mathbf{A}) = \sum_{i=1}^n a_{ii}$ . <b>Properties:</b> <ul style="list-style-type: none"><li><math>\text{Tr}(A) = \sum_i \lambda_i</math> (sum of eigenvalues).</li> <li>Cyclic property: <math>\text{Tr}(ABC) = \text{Tr}(BCA) = \text{Tr}(CAB)</math>.</li> <li>Linear: <math>\text{Tr}(A + B) = \text{Tr}(A) + \text{Tr}(B)</math> and <math>\text{Tr}(cA) = c \text{Tr}(A)</math>.</li> <li><math>\text{Tr}(A) = \text{Tr}(A^T)</math>.</li></ul>

**D. (Frobenius Norm ( $\|\cdot\|_F$ ))** The square root of the sum of the absolute squares of its elements.

$$\|A\|_F = \sqrt{\sum_{i,j} |A_{ij}|^2}$$

**Properties:**

- Relation to Trace:  $\|A\|_F = \sqrt{\text{Tr}(A^T A)}$ .
- Invariant under orthogonal rotations:  $\|QA\|_F = \|A\|_F$  for orthogonal  $Q$ .
- Relation to Singular Values:  $\|A\|_F = \sqrt{\sum_i \sigma_i^2}$ .

4 Derivatives
<b>- 4.1 — Numerator and Denominator Convention</b>

**Jacobian Layout Convention** **Com.** We use the *numerator-layout*  
For a vector-valued function  $f: \mathbb{R}^n \rightarrow \mathbb{R}^m$  we define

$$\left(\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right)_{ij} := \frac{\partial f_i}{\partial x_j}, \quad \frac{\partial \mathbf{f}}{\partial \mathbf{x}} \in \mathbb{R}^{n \times m}.$$

Hence, gradients of scalar-valued functions are row vectors, and the chain rule takes the form

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

**Remark.** Sometimes the *denominator-layout* is used, where the Jacobian is defined as  $(\mathbf{J}_f)_{ij} = \partial f_i / \partial x_i \in \mathbb{R}^{n \times m}$ . The two conventions are related by transposition.

<b>- 4.2 — Scalar-by-Vector</b>
<b>Denominator Convention</b> $\frac{\partial}{\partial \mathbf{x}} [u(\mathbf{x})v(\mathbf{x})] = u(\mathbf{x}) \frac{\partial v(\mathbf{x})}{\partial \mathbf{x}} + v(\mathbf{x}) \frac{\partial u(\mathbf{x})}{\partial \mathbf{x}}$ $\frac{\partial}{\partial \mathbf{x}} [u(v(\mathbf{x}))] = \frac{\partial u(v)}{\partial v} \frac{\partial v(\mathbf{x})}{\partial \mathbf{x}}$ $\frac{\partial}{\partial \mathbf{x}} [\mathbf{f}(\mathbf{x})^T \mathbf{g}(\mathbf{x})] = \frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}}^T \mathbf{g}(\mathbf{x}) + \frac{\partial \mathbf{g}(\mathbf{x})}{\partial \mathbf{x}} \mathbf{f}(\mathbf{x}) = \mathbf{J}_f \mathbf{g}(\mathbf{x}) + \mathbf{J}_g \mathbf{f}(\mathbf{x})$ $\frac{\partial}{\partial \mathbf{x}} [\mathbf{f}(\mathbf{x})^T \mathbf{A} \mathbf{g}(\mathbf{x})] = \frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}}^T \mathbf{A} \mathbf{g}(\mathbf{x}) + \frac{\partial \mathbf{g}(\mathbf{x})}{\partial \mathbf{x}} \mathbf{A}^T \mathbf{f}(\mathbf{x})$ $\frac{\partial}{\partial \mathbf{x}} [\mathbf{a}^T \mathbf{x}] = \frac{\partial}{\partial \mathbf{x}} [\mathbf{x}^T \mathbf{a}] = \mathbf{a}$ $\frac{\partial}{\partial \mathbf{x}} [\mathbf{x}^T \mathbf{x}] = 2\mathbf{x}$ $\frac{\partial}{\partial \mathbf{x}} [\mathbf{b}^T \mathbf{A} \mathbf{x}] = \mathbf{A}^T \mathbf{b}$ $\frac{\partial}{\partial \mathbf{x}} [\mathbf{x}^T \mathbf{A} \mathbf{x}] = (\mathbf{A} + \mathbf{A}^T) \mathbf{x}$

$$\frac{\partial}{\partial \mathbf{x}} [(\mathbf{A} \mathbf{x} + \mathbf{b})^T \mathbf{C} (\mathbf{D} \mathbf{x} + \mathbf{e})] = \mathbf{D}^T \mathbf{C}^T (\mathbf{A} \mathbf{x} + \mathbf{b}) + \mathbf{A}^T \mathbf{C} (\mathbf{D} \mathbf{x} + \mathbf{e})$$

$$\frac{\partial}{\partial \mathbf{x}} [\|\mathbf{f}(\mathbf{x})\|_2^2] = \frac{\partial}{\partial \mathbf{x}} [\mathbf{f}(\mathbf{x})^T \mathbf{f}(\mathbf{x})] = 2 \frac{\partial}{\partial \mathbf{x}} [\mathbf{f}(\mathbf{x})] \mathbf{f}(\mathbf{x}) = 2 \mathbf{J}_f \mathbf{f}(\mathbf{x})$$

<b>- 4.3 — Vector-by-Vector</b>
<b>A, C, D, a, b, e</b> not a function of $\mathbf{x}$ , $\mathbf{f} = \mathbf{f}(\mathbf{x})$ , $\mathbf{g} = \mathbf{g}(\mathbf{x})$ , $\mathbf{h} = \mathbf{h}(\mathbf{x})$ , $u = u(x)$ , $v = v(x)$
$\frac{\partial}{\partial \mathbf{x}} [u(\mathbf{x})\mathbf{f}(\mathbf{x})] = u(\mathbf{x}) \frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}} + \mathbf{f}(\mathbf{x}) \frac{\partial u(\mathbf{x})}{\partial \mathbf{x}}$ $\frac{\partial}{\partial \mathbf{x}} [\mathbf{x} \odot \mathbf{a}] = \text{diag}(\mathbf{a})$ $\frac{\partial}{\partial \mathbf{x}} [\mathbf{a}] = \mathbf{0}$ $\frac{\partial}{\partial \mathbf{x}} [\mathbf{x}] = \mathbf{I}$ $\frac{\partial}{\partial \mathbf{x}} [\mathbf{A} \mathbf{x}] = \mathbf{A}$ $\frac{\partial}{\partial \mathbf{x}} [\mathbf{x}^T \mathbf{A}] = \mathbf{A}^T$ $\frac{\partial}{\partial \mathbf{x}} [\mathbf{a} \mathbf{f}(\mathbf{x})] = a \frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}} = a \mathbf{J}_f$ $\frac{\partial}{\partial \mathbf{x}} [\mathbf{A} \mathbf{f}(\mathbf{x})] = \mathbf{A} \frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}}$ $\frac{\partial}{\partial \mathbf{x}} [\mathbf{f}(\mathbf{g}(\mathbf{x}))] = \frac{\partial \mathbf{f}}{\partial \mathbf{g}} \frac{\partial \mathbf{g}}{\partial \mathbf{h}} \frac{\partial \mathbf{h}}{\partial \mathbf{x}}$ $\frac{\partial}{\partial \mathbf{x}} [\mathbf{J}_f(\mathbf{g}) \mathbf{J}_g(\mathbf{x})]$

<b>- 4.4 — Scalar-by-Matrix</b>
$\frac{\partial}{\partial \mathbf{x}} [\mathbf{a}^T \mathbf{X} \mathbf{b}] = \mathbf{a}^T$ $\frac{\partial}{\partial \mathbf{x}} [\mathbf{a}^T \mathbf{X}^T \mathbf{b}] = \mathbf{a}^T$ $\frac{\partial}{\partial \mathbf{x}} [\mathbf{a}^T \mathbf{X}^T \mathbf{X} \mathbf{b}] = \mathbf{X}(\mathbf{a} \mathbf{b}^T + \mathbf{b} \mathbf{a}^T)$ $\frac{\partial}{\partial \mathbf{x}} [\text{Tr}(\mathbf{X})] = \mathbf{I}$ $\frac{\partial}{\partial \mathbf{x}} [\text{Tr}(\mathbf{A} \mathbf{X} \mathbf{B})] = \mathbf{A}^T \mathbf{B}^T$ $\frac{\partial}{\partial \mathbf{x}} [\text{Tr}(\mathbf{A} \mathbf{X}^T \mathbf{B})] = \mathbf{B} \mathbf{A}$

- 4.5 — Vector-by-Matrix (Generalized Gradient)
<b>5 Information Theory</b>

**D. (Entropy)** Let  $\mathbf{X}$  be a random variable distributed according to  $p(\mathbf{X})$ . Then the entropy of  $\mathbf{X}$   $H(\mathbf{X}) = \mathbb{E}[-\log(P(\mathbf{X}))] = -\sum_{\mathbf{x} \in \mathcal{X}} p(\mathbf{x}) \log(p(\mathbf{x})) \geq 0$ . It describes the expected information content  $I(\mathbf{X})$  of  $\mathbf{X}$ .

**D. (Cross-Entropy)** For distributions  $p$  and  $q$  over a given set is  $H(p, q) = -\sum_{\mathbf{x} \in \mathcal{X}} p(\mathbf{x}) \log(q(\mathbf{x})) = \mathbb{E}_{\mathbf{x} \sim p}[-\log(q(\mathbf{x}))] \geq 0$ .

$$H(X; p, q) = H(X) + KL(p, q) \geq 0, \quad \text{where } H \text{ uses } p.$$

**Com.** The minimizer of the cross-entropy is  $q := p$ , due to the second formulation.

**Com.** Usually,  $q$  is the approximation of the unknown  $p$ .

**D. (Kullback-Leibler Divergence)**

For probability distributions  $p$  and  $q$  defined on the same probability space, the KL-divergence between  $p$  and  $q$  is defined as

$$KL(p, q) = - \sum_{\mathbf{x} \in \mathcal{X}} p(\mathbf{x}) \log \left( \frac{q(\mathbf{x})}{p(\mathbf{x})} \right) = \sum_{\mathbf{x} \in \mathcal{X}} p(\mathbf{x}) \log \left( \frac{p(\mathbf{x})}{q(\mathbf{x})} \right) \geq 0.$$

$$KL(p, q) = -\mathbb{E}_{\mathbf{x} \sim p} \left[ \log \left( \frac{q(\mathbf{x})}{p(\mathbf{x})} \right) \right] = \mathbb{E}_{\mathbf{x} \sim p} \left[ \log \left( \frac{p(\mathbf{x})}{q(\mathbf{x})} \right) \right] \geq 0.$$

$$KL(X; p, q) = H(p, q) - H(X), \quad \text{where } H \text{ uses } p.$$

The KL-divergence is defined only if  $\forall \mathbf{x}: q(\mathbf{x}) = 0 \implies p(\mathbf{x}) = 0$  (absolute continuity). Whenever  $p(\mathbf{x})$  is zero the contribution of the corresponding term is interpreted as zero because  $\lim_{x \rightarrow 0+} x \log(x) = 0$ .

In ML it is a measure of the amount of information lost, when  $q$  (model) is used to approximate  $p$  (true).

**Com.**  $KL(p, y) = 0 \iff p \equiv q$ .

**Com.** Note that the KL-divergence is not symmetric!

**D. (Jensen-Shannon Divergence)**

$$JSD(P, Q) = \frac{1}{2} KL(P, M) + \frac{1}{2} KL(Q, M) \in [0, \log(n)]$$

$$M = \frac{1}{2} (P + Q)$$

**C** The JSD is symmetric!

**Com.** The JSD is a symmetrized and smoothed version of the KL-divergence.

6 Activation Functions
Activation functions are non-linear functions that are applied element-wise to the output of a linear function.

**D. (Tanh)**

$$\tanh(\mathbf{x}) = \frac{e^{\mathbf{x}} - e^{-\mathbf{x}}}{e^{\mathbf{x}} + e^{-\mathbf{x}}} \quad \tanh'(\mathbf{x}) = 1 - \tanh^2(\mathbf{x})$$

<b>D. (ReLU)</b>
$\text{ReLU}(x) = \max(x, 0) \quad \text{ReLU}'(x) = \begin{cases} 1 & x \geq 0 \\ 0 & x \leq 0 \end{cases}$

<b>D. (Sigmoid/Logistic)</b>
$\sigma(\mathbf{x})_i = \frac{1}{1 + e^{-x_i}} \in (0, 1) \quad \frac{\partial \sigma(\mathbf{x})}{\partial \mathbf{x}} = \sigma(\mathbf{x}) \odot (1 - \sigma(\mathbf{x}))$

$$\sigma^{-1}(y) = \ln \left( \frac{y}{1 - y} \right)$$

$$\nabla_{\mathbf{x}} \sigma(\mathbf{x}) = \mathbf{J}_{\sigma}(\mathbf{x}) = \text{diag}(\sigma(\mathbf{x}) \odot (1 - \sigma(\mathbf{x})))$$

$$\sigma'(x) = \frac{1}{4} \tanh'(\frac{1}{2}x) = \frac{1}{4} (1 - \tanh^2(\frac{1}{2}x))$$

$$\sigma(-x) = 1 - \sigma(x)$$

**Connection between Sigmoid and Tanh** (Equal Representation Strength)

$$\sigma(x) = \frac{1}{2} \tanh \left( \frac{1}{2} x \right) + \frac{1}{2} \iff \tanh(x) = 2\sigma(2x) - 1$$



**11.3 — Gradient Descent** — **D. (Gradient Descent (GD))** Iteratively moves parameters  $\theta$  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  $\theta_T$ , 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  $\alpha_t$  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  $\alpha_t = \frac{1}{\sqrt{t}}$  satisfies both, whereas  $\alpha_t = \frac{1}{\sqrt[3]{t}}$ . 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).

$$\theta^{t+1} = \theta^t - \eta \nabla J(\theta^t) + \beta(\theta^t - \theta^{t-1})$$

where  $\beta \in [0, 1)$  is the momentum term (friction).

**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. (RMSProp)** 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  $\alpha, \beta$ ,  $T(\alpha f + \beta g) = \alpha(Tf) + \beta(Tg)$ .

**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

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

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

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

and its inverse as

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

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

$$F(f * h) = Ff \cdot Fh$$

and then transformed back using the inverse Fourier transform.

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

— **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

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

**D. (Discrete Cross-Correlation)**

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

$$\begin{aligned} (h * f)[u] &:= \\ \sum_{t=-\infty}^{\infty} h[t] f[u + t] &= \sum_{t=-\infty}^{\infty} h[-t] f[u - t] \\ (\bar{h} * f)[u] &= (f * \bar{h})[u] \quad \text{where } \bar{h}(t) = h(-t). \end{aligned}$$

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 (so  $+2p$ ) 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  $\mathcal{I}_l^i$  of  $x_i^l$ )**

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

**Com.** Hence, the receptive field of a node  $x_i^l$  are just nodes the 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  $\forall j \neq i: \mathcal{I}_l^i: \frac{\partial x_i^l}{\partial x_j^{l-1}} = 0$ ,

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

$$\frac{\partial \mathcal{R}}{\partial h_{ij}^l} = \sum_{i=1}^{m_l} \frac{\partial \mathcal{R}}{\partial x_i^l} \frac{\partial x_i^l}{\partial h_{ij}^l}$$

**Backpropagations of Convolutions as Convolutions**

**y<sup>(l)</sup>** output of  $l$ -th layer **y<sup>(l−1)</sup>** output of  $(l - 1)$ -th layer **w** input to  $l$ -th layer **w** convolution filter  $\frac{\partial \mathcal{R}}{\partial \mathbf{y}^{(l)}}$  known **y<sup>(l+1)</sup> = y<sup>(l)</sup> \* w**

$$\frac{\partial \mathcal{R}}{\partial w_i} = \sum_k \frac{\partial \mathcal{R}}{\partial y_k^{(l)}} \frac{\partial y_k^{(l)}}{\partial w_i} = \sum_k \frac{\partial \mathcal{R}}{\partial y_k^{(l)}} \frac{\partial}{\partial w_i} \left[ \mathbf{y}^{(l)} * \mathbf{w} \right]_k$$

$$\begin{aligned} &= \sum_k \frac{\partial \mathcal{R}}{\partial y_k^{(l)}} \frac{\partial}{\partial w_i} \left[ \sum_{o=p}^p y_{k-o}^{(l-1)} w_o \right] = \sum_k \frac{\partial \mathcal{R}}{\partial y_k^{(l)}} y_{k-i}^{(l-1)} \\ &= \sum_k \frac{\partial \mathcal{R}}{\partial y_k^{(l)}} y_{-(k-i)}^{(l-1)} = \sum_k \frac{\partial \mathcal{R}}{\partial y_k^{(l)}} \text{rot}180(y^{(l-1)})_{k-i} \\ &= \left( \frac{\partial \mathcal{R}}{\partial \mathbf{y}^{(l)}} * \text{rot}180(y^{(l-1)}) \right)_i \end{aligned}$$

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

Note that we just used generalized indices  $i, k, o$  which may be multi-dimensional.

This example omits activation functions and biases, but that could be easily included with the chain-rule.

**D. (Rotation180)**  $\forall i: \text{rot}180(\mathbf{x})_i = \mathbf{x}_{(-i)}$ .

— **12.6 — Pooling** —

There are min, max, avg, and softmax pooling. Max pooling is the most frequently used one.

**D. (Max-Pooling)**

- 1D:  $x_{ij}^{\text{max}} = \max \{x_{i+k} \mid 0 \leq k < r\}$
- 2D:  $x_{ij}^{\text{max}} = \max \{x_{i+k, j+l} \mid 0 \leq k, l < r\}$

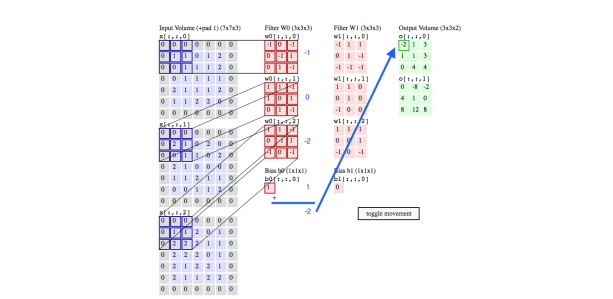
— **12.7 — Sub-Sampling (aka "Strides")** —

Often, it is desirable to reduce the size of the feature maps. This can be achieved by skipping some of the input values in the convolution. The stride is the number of steps the kernel takes in each direction.

— **12.8 — Channels** —

**Ex.** Here we have

- an input signal that is 2D with 3 channels (7x7x3) (image x channels)
- and we want to learn two filters  $W_0$  and  $W_1$ , which each process the 3 channels, and sum the results of the convolutions across each channel leading to a tensor of size 3x3x2 (convolution result x num convolutions)



Usually we convolve over all of the channels together, such that each convolution has the information of all channels at its disposition and the order of the channels hence doesn't matter.

— **12.9 — CNNs in Computer Vision** —

So the typical use of convolution that we have in vision is: a sequence of convolutions

- that *reduce* the spatial dimensions (sub-sampling)
- that *increase* the number of channels

The deeper we go in the network, we transform the spatial information into a semantic representation. Usually, most of the parameters lie in the fully connected layers

— **12.9.1 — Classic CNN Architectures** —

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

- Structure:** 2 Convolutional layers (with Average Pooling) followed by 3 Fully Connected layers.
- Key Features:** Introduced the concepts of local receptive fields, shared weights, and spatial subsampling. Used Sigmoid/Tanh activations (pre-ReLU).

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

- Philosophy:** Replace large filters (e.g., 5 × 5, 7 × 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. (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 × 1, 3 × 3, and 5 × 5 convolutions (and pooling) in *parallel* and concatenates the outputs.
- 1 × 1 **Convolutions:** Used as "Bottleneck layers" to reduce dimensionality (depth) before expensive operations, significantly reducing computational cost.

**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.

— **12.9.2 — Convolutions in Sequences, NLP & Audio** —

**D. (1D Convolutions)** Unlike 2D CNNs for images, sequence modeling (Temporal ConvNets) uses 1D filters that slide over the time axis.

• **Input:** Tensor of shape *(Batch, Length, Channels)*. In NLP, "Channels" are the dimensions of the Word Embeddings.

• **Function:** Captures local temporal dependencies (like  $n$ -grams in text) effectively.

• **Advantage:** Highly parallelizable (unlike RNNs which are sequential) and computationally efficient.

**D. (Embeddings & NLP)** CNNs are often applied on top of pre-trained word embeddings (e.g., Word2Vec, GloVe).

- A sentence is represented as a matrix (Length × Embedding Dim).

• Filters of different widths (e.g., covering 2, 3, or 4 words) act as feature detectors for phrases or specific linguistic patterns (e.g., "very good", "not bad") regardless of their position in the sentence.

**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...), 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).
- Dilated Causal Convolutions:** Stacks layers with increasing dilation factors. This allows the output neuron to have a receptive field of thousands of timesteps (milliseconds of audio) to generate realistic high-fidelity sound structure.
- Skip Connections:** Uses residual and parameterized skip connections to speed up convergence and allow training of very deep networks.

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

**Ex:** input image  $m \times n \times c$  ( $c$  = number of 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 out-puts as CNN:

$mnc((m - p + 1)(n - q + 1) + 1)K$

#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.

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

$$\begin{aligned} z_t &= \phi(Wz_{t-1} + Vx_t) \\ \hat{y}_t &= \psi(Wz_t) \end{aligned}$$

where  $U, V, W$  are shared weight matrices and  $\phi, \psi$  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:

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

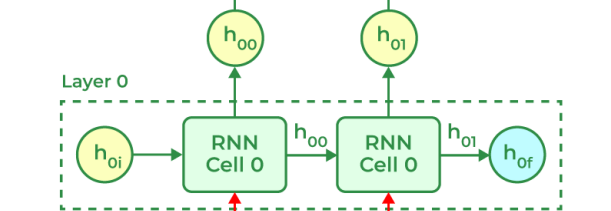
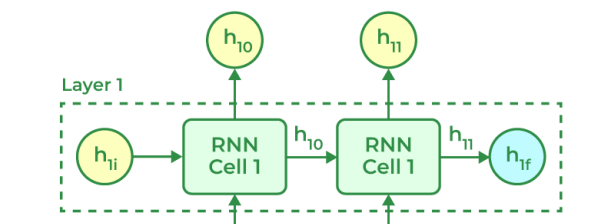
**Gradient Problems:** The gradient involves repeated multiplication of the recurrent weight matrix  $U$  (specifically its Jacobian).

- Exploding Gradient:** If largest singular value  $\sigma_{\max}(U) > 1$ .
- Vanishing Gradient:** If  $\sigma_{\max}(U) < 1$ . This makes learning long-term dependencies difficult.

— **13.1.1 — Structural Variants** —

**D. (Bidirectional RNNs):** Process sequence in both directions to capture past and future context.

$$\hat{y}_t = \psi(Wz_t^{\rightarrow} + \tilde{W}z_t^{\leftarrow})$$



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

— **13.1.2 — Gated Memory** —

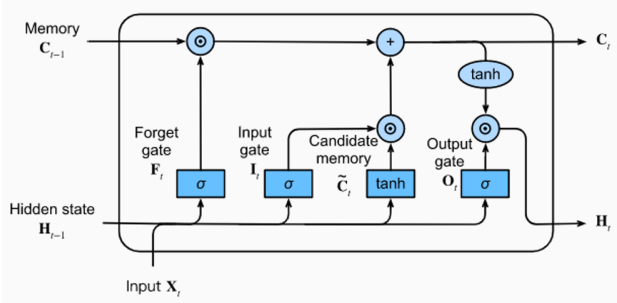
Gates use multiplicative interactions (sigmoid  $\sigma$ ) to control information flow, stabilizing gradients and allowing long-term memory.

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

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

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

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



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

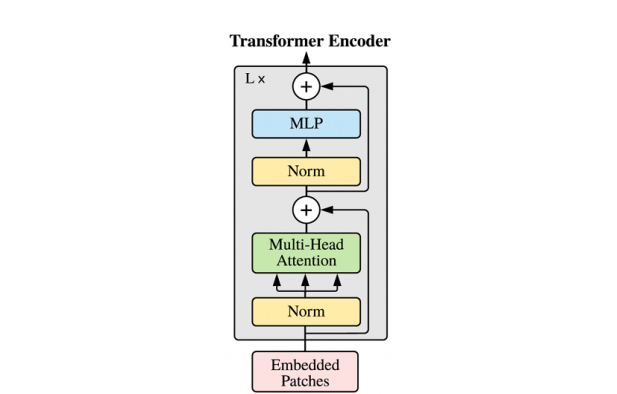
$$z^t = (1 - \Gamma_u) \odot z^{t-1} + \Gamma_u \odot \tilde{z}^t$$



- 15.3 — Vision Transformers**
- use  $16 \times 16$  non-overlapping pixel-patches as tokens
- flatten patches p and linearly project to embedding space

$$\mathbf{p}^t \in \mathbb{R}^{p \times p \times q} \longmapsto x^t \equiv V \text{ vec}(\mathbf{p}^t) \in \mathbb{R}^n, V \in \mathbb{R}^{n \times (qp^2)}$$

- pre-processing ignores the 2D structure of images (unproblematic for large datasets)
- no built-in translation equivariance like in CNNs, but lower inductive bias & little spatial awareness



## 16 Geometric Deep Learning

### – 16.1 — Sets & Point Clouds (Deep Sets) —

Standard NNs assume fixed input order. Sets require **Permutation Invariance**.

**D.** Permutations: Represented by a \*\*Permutation Matrix\*\*  $P$  (one 1 per row/col) or \*\*Cauchy Two-Line Notation\*\*  $\pi$ :

- Cauchy Notation**:  $\pi = \left( \begin{smallmatrix} \pi(1) & \pi(2) & \cdots & \pi(M) \end{smallmatrix} \right)$
- Matrix Properties**:  $P^{-1} = P^T$  and  $PP^T = I$ .
- Row Permutation**:  $PX$  permutes rows (samples).
- Column Permutation**:  $XP^T$  permutes columns (features).

**D.** Invariance vs. Equivariance: Let  $\pi$  be a permutation of indices  $\{1, \dots, M\}$ .

- Invariant**: Output remains unchanged (e.g., classification).

$$f(x_1, \dots, x_M) = f(x_{\pi(1)}, \dots, x_{\pi(M)})$$

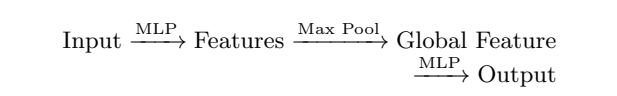
- Equivariant**: Output permutes exactly as input does (e.g., segmentation).

$$f(PX) = Pf(X) \quad (\text{where } P \text{ is a permutation matrix})$$

**D.** Deep Sets Theorem: Any invariant function  $f$  can be decomposed into an element-wise encoder  $\phi$  and an invariant aggregator  $\rho$  (e.g., sum, max).

$$f(X) = \rho \left( \sum_{m=1}^M \phi(x_m) \right)$$

**D.** PointNet: Architecture for 3D point clouds. Uses a **T-Net** to predict affine transformations (canonicalization) for rotation invariance.



### – 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).

**D.** Coupling Matrix ( $\tilde{A}$ ): Standard symmetric normalized formulation. We define the self-loop adjacency  $\tilde{A} = A + I$  and corresponding degree matrix  $\tilde{D}_{mm} = \sum_n \tilde{A}_{nm} = d_m + 1$ .

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

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

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

**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  $A$ .

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

**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  $\alpha_{ij}$  instead of static adjacency.

**D.** Attention Mechanism:

- Score**:  $e_{ij} = \text{LeakyReLU}(\mathbf{a}^T [Wx_i || Wx_j])$
- Normalize**:  $\alpha_{ij} = \text{softmax}_j(e_{ij}) = \frac{\exp(e_{ij})}{\sum_{k \in \mathcal{N}_i} \exp(e_{ik})}$
- 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  $\theta_0$  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  $\beta$  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  $\xi$  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  $k_{\infty}$  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  $k_{\infty}$ . The evolution of outputs follows:

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

### – 17.2 — Bayesian DNNs —

**D.** Bayesian Paradigm: Instead of finding a point estimate  $\theta^*$ , 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  $\mu(x) = 0$  and a specific kernel  $k(x, x')$ .

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

$$K_1(x, x') = E[\phi(f_{1-1}(x))\phi(f_{1-1}(x'))]$$

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

### – 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)  $\mathbf{T}$**   $\in \mathbb{R}^{d_1 \times d_2 \times \cdots \times d_k}$

**D. (Tensor Multiplication)**

$$\mathbf{T} \begin{smallmatrix} \in \mathbb{R}^{(a+c)} \\ \\ \end{smallmatrix} = \mathbf{P} \begin{smallmatrix} \in \mathbb{R}^{(a+b)} \\ \\ \end{smallmatrix} \times_b \mathbf{Q} \begin{smallmatrix} \in \mathbb{R}^{(b+c)} \\ \\ \end{smallmatrix}$$

$$\mathbf{P} \begin{smallmatrix} \times_{r_1} \times \cdots \times_{r_a} \times_{r_a} \times_{s_1} \times \cdots \times_{s_b} \times_{s_1} \cdots \times_{s_b} \times_{s_1} \times \cdots \times_{s_b} \times_{t_1} \times \cdots \times_{t_c} \end{smallmatrix} = \mathbf{T} \begin{smallmatrix} \times_{r_1} \times \cdots \times_{r_a} \times_{s_1} \times \cdots \times_{s_b} \times_{s_1} \cdots \times_{s_b} \times_{t_1} \times \cdots \times_{t_c} \end{smallmatrix}$$

Note that this is just the sum of the multiplications of two numbers which are in corresponding locations in **P** and **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:

$$T_{\underset{r_1}{i_1}, \dots, \underset{r_a}{i_a}, \underset{s_1}{j_1}, \dots, \underset{s_b}{j_b}, \underset{t_1}{k_1}, \dots, \underset{t_c}{k_c}} := \sum_{\underset{r_1}{j_1}, \dots, \underset{r_b}{j_b}} P_{\underset{r_1}{i_1}, \dots, \underset{r_a}{i_a}, \underset{s_1}{j_1}, \dots, \underset{s_b}{j_b}} Q_{\underset{r_1}{j_1}, \dots, \underset{r_b}{j_b}, \underset{s_1}{k_1}, \dots, \underset{t_c}{k_c}}$$

**T. (Tensor Chain Rule)**

$$y(W): \mathbb{R}^{d_1 \times d_2} \rightarrow \mathbb{R}^{d_3 \times d_4}, \; L(y): \mathbb{R}^{d_3 \times d_4} \rightarrow \mathbb{R}$$

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial W} \times_{d_3, d_4} \frac{\partial y}{\partial W}$$

then we have:  $T_{i,j,k,l} = \frac{\partial y_{j,l}}{\partial W_{k,l}}$

## 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):

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

- Encoder** ( $q_{\phi}$ ): Maps input  $x$  to latent parameters  $\mu, \Sigma$ .
- Decoder** ( $p_{\theta}$ ): 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  $\phi$  and fixed noise  $\epsilon$ .

### – 19.2 — Factor Analysis —

defines a proper probabilistic model of the data ( $m \ll n$ ):

- choose a probability density function  $p_z$  over the latents
- define a conditional probability density function  $p_{X|Z}$  over observables
- integrate out the latent variables

$$p_X(x) = \int p_Z(z) p_{X|Z}(x | z) dz$$

- use the Gaussian prior density  $z \sim \mathcal{N}(0, I), \quad z \in \mathbb{R}^m$
- add linear observation model for  $x \in \mathbb{R}^n$

$$x = \mu + Wz + \eta, \quad \eta \sim \mathcal{N}(0, \Sigma), \quad \Sigma = \text{diag}(\sigma_1^2, \dots, \sigma_n^2)$$

$$x \in \mathbb{R}^n, \quad z \in \mathbb{R}^m, \quad W \in \mathbb{R}^{n \times m}$$

- observational noise  $\eta$  is independent of the latents  $z$
- The induced density is itself normal

$$x \sim \mathcal{N}(\mu, WW^T + \Sigma), \quad \mu = \frac{1}{s} \sum_{i=1}^s x_i$$

- factors are only identifiable up to orthogonal transformations (rotations, reflections, or permutations) in  $\mathbb{R}^m$ , because for any orthogonal matrix  $Q \in \mathbb{R}^{m \times m}$  holds:

$$(WQ)(WQ)^T = WQQ^TW^T = WW^T \quad \text{if } QQ^T = I$$

$$\mu_{z|x} = W^T(WW^T + \Sigma)^{-1}(x - \mu)$$

$$\Sigma_{z|x} = I - W^T(WW^T + \Sigma)^{-1}W.$$

- For vanishing isotropic noise (probabilistic PCA) ( $\Sigma = \sigma^2 I, \sigma^2 \rightarrow 0$ ), the pseudo-inverse of  $W$  converges (if  $W$  has orthogonal colums  $W^+ = W^T$ )

$$W^T(WW^T + \sigma^2 I)^{-1} \longrightarrow W^+ \in \mathbb{R}^{m \times n}$$

- Given W, it is thus easy to calc. the posterior over  $z$ .

$$\mu_{z|x} \longrightarrow W^+(x - \mu), \quad \Sigma_{z|x} \longrightarrow 0$$

- MLE with a sample set  $S: \mu, W \leftarrow \log p(\mu, W)(S)$
- The optimality condition of the  $i$ -th column of  $W$  is

$$w_i = \rho_i \, u_i, \quad \rho_i = \max\{0, \, \sqrt{\lambda_i - \sigma^2}\}.$$

where  $u_i$  is the  $i$ -th principal eigenvector of the sample covariance matrix and  $\lambda_i$  the corresponding eigenvalue.

- For  $\sigma^2 \rightarrow 0$ , we recover PCA

### – 19.3 — 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:

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

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  $J = \prod \exp(s(x_{1:d}))$ .

### – 19.4 — Adversarial Networks (GANs) —

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

**D. (Minimax Objective)**:

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

**Optimality**:

- Optimal Discriminator**: For a fixed  $G$ ,  $D^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_g(\pi)}$ .
- Global Minimum**: Achieved when  $p_g = p_{data}$ . 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  $D$ .

### – 19.5 — Denoising Diffusion Models (DDPM) —

Learns to reverse a gradual noising process.

**D. (Forward Process (Fixed))**: Markov chain adding Gaussian noise according to schedule  $\beta_t$ :

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

**Closed form** sampling at step  $t$  (using  $\alpha_t = 1 - \beta_t$  and  $\bar{\alpha}_t = \prod \alpha_i$ ):

$$x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon, \quad \epsilon \sim \mathcal{N}(0, I)$$

**D. (Reverse Process (Learned))**: Approximated by a neural network with parameters  $\theta$ :

$$p_{\theta}(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_{\theta}(x_t, t), \Sigma_{\theta}(x_t, t))$$

**D. (Simplified Objective)**: Instead of predicting the image mean  $\mu$ , we predict the noise  $\epsilon$  added at step  $t$ :

$$L_{\text{simple}} = \mathbb{E}_{t, x_0, \epsilon} [\|\epsilon - \epsilon_{\theta}(\sqrt{\bar{\alpha}_t}x_0 + \sqrt{1 - \bar{\alpha}_t}\epsilon, t)\|^2]$$

### – 19.6 — Autoregressive Models —

**T. (AR for Density Estimation)**

Any joint density can be factorized as a product of conditionals

$$p(\mathbf{x}) = \prod_{i=1}^d p(x_i | \mathbf{x}_{<i})$$

Traditional linear AR models in statistics use:

$$p(x_i | \mathbf{x}_{<i}) = \mathcal{N}\left(x_i; \sum_{k=1}^K \theta_k x_{i-k}, \sigma^2\right)$$

To obtain richer models, we can model each conditional with a DNN Properties:

- exact likelihood evaluation (fully tractable)
- sampling is sequential over dimensions (can be slow)
- AR structure  $\rightarrow$  triangular Jacobian in flows

VAEs & GANs are complicated to learn / not always successful. AR models are simple (look at chainrule  $p(x_1, \dots, x_m) = \prod_{i=1}^m p(x_i | x_{1:i-1})$ )

## 20 Ethics

### – 20.1 — Robustness —

**D. (Adversarial examples)** (Classification Perspective) Input  $x$ , label  $y$ , budget  $\epsilon$ , norm  $\|\cdot\|_p$  (usually  $p \in \{2, \infty\}$ ) for each attack type:

- Untargeted**:  $\|\delta\|_p \leq \epsilon$  and  $\hat{y}(x + \delta) \neq y$
- Targeted**:  $\|\delta\|_p \leq \epsilon$  and  $\hat{y}(x + \delta) = t, t \neq y$
- Loss-based**:  $\max_{\|\delta\|_p \leq \epsilon} \mathcal{L}(f(x + \delta), y)$

**Binary Classification**:  $f(x) = w^T x + b$ , adversarial perturbation pushes  $x$  across decision boundary if  $y(w^T(x + \delta) + b) \leq 0$ .

**D. (Norms)**

- $\|x\|_p = (\sum_{i=1}^d |x_i|^$