

# 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  $P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)} = \frac{\sum_{i=1}^k P(X|Y_i)P(Y_i)}{\sum_{i=1}^k P(X|Y_i)}$   
 Cond. Ind.  $X \perp Y|Z \implies P(X, Y|Z) = P(X|Z)P(Y|Z) = P(X|Z)P(Y|Z)$   
 Cond. Ind.  $X \perp Y|Z \implies P(X|Y, Z) = P(X|Z)$

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

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

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

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

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

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

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

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

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

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

## T. (Moment Generating Function)

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

$$Be(p) = pe^t + (1-p)$$

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

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

$$\text{Gam}(\alpha, \beta) = \left(\frac{1}{\alpha} e^{-\beta}\right)^{\alpha}$$

for  $t < 1/\beta$

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

**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  $Y$ :  $f_Z(z) = \int_{\mathbb{R}} f_X(x) f_Y(z-x) dx$

## D. (Normal Distribution)

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

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

$$\mathcal{P}(\mathbf{a} | \mathbf{a}_2) = \mathcal{N}(\mathbf{a}_2 | [\mathbf{a}_2], [\Sigma_{21}, \Sigma_{22}, \Sigma_{12}])$$

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

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

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

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

**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):  $\nabla_\theta [p_\theta(\mathbf{x})] = p_\theta(\mathbf{x}) \nabla_\theta [\log(p_\theta(\mathbf{x}))]$

**T. (Cauchy-Schwarz)**  $\forall u, v \in V: \langle u, v \rangle \leq \|u\| \|v\|$ .

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

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

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  $\forall x, x' \in S, \forall \lambda \in [0, 1]: \lambda x + (1-\lambda)x' \in S$ .

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  $\forall x, x' \in S, \lambda \in [0, 1]: f(\lambda x + (1-\lambda)x') \leq \lambda f(x) + (1-\lambda)f(x')$

**D. (Convex Function)** A function  $f: S \subseteq \mathbb{R}^d$  is called *convex* if it is strictly 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** ( $O(c^\ell)$  for some  $c < 1$ ).

**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  $x, y$ :

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

**Jacobian Layout Convention** Com. We use the *numerator-layout*

For a vector-valued function  $f: \mathbb{R}^n \rightarrow \mathbb{R}^m$  we define

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

**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 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.

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

**T. (Taylor-Lagrange Formula)**

$$f(x) = \sum_{k=0}^n \frac{f^{(k)}(x_0)}{k!} (x - x_0)^k + \int_{x_0}^x f^{(n+1)}(t) \frac{(x-t)^n}{n!} dt$$

**T. (Jensen)**  $f$  convex/concave,  $\forall i: \lambda_i \geq 0, \sum_{i=1}^n \lambda_i = 1$

$$f\left(\sum_{i=1}^n \lambda_i \mathbf{x}_i\right) \leq \sum_{i=1}^n \lambda_i f(\mathbf{x}_i)$$

Special case:  $f(\mathbb{E}[X]) \leq \mathbb{E}[f(X)]$ .

**D. (L-Lipschitz Continuous Function)**

Given two metric spaces  $(X, d_X)$  and  $(Y, d_Y)$ , a function  $f: X \rightarrow Y$  is called *Lipschitz continuous*, if there exists a real constant  $L \in \mathbb{R}_+^+$  (*Lipschitz constant*), such that

$$\forall \mathbf{x}_1, \mathbf{x}_2 \in \mathbb{R}^n: \|f(\mathbf{x}_1) - f(\mathbf{x}_2)\| \leq L \cdot \|\mathbf{x}_1 - \mathbf{x}_2\|.$$

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-Lojasiewicz (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:

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

**4.5 Vector-by-Matrix (Generalized Gradient)**

$$\frac{\partial}{\partial \mathbf{x}} [\mathbf{X} \mathbf{a}] = \mathbf{X}^T \mathbf{a}$$

**4.6 Subdifferential**

$f: \mathbb{R}^n \rightarrow \mathbb{R} \cup \{+\infty\}$  be a convex function, and let  $x$  be a point in its domain. The **subdifferential** of  $f$  at  $x$ , denoted

$$\partial f(x)$$

is the set of all vectors  $g \in \mathbb{R}^n$

$$f(y) \geq f(x) + g^T(y - x) \quad \text{for all } y \in \mathbb{R}^n.$$

Any vector  $g \in \partial f(x)$  is called a **D. (subgradient)** of  $f$  at  $x$ .

**5 Information Theory**

**D. (Entropy)** Let  $X$  be a random variable distributed according to  $p(\mathbf{x})$ . Then the entropy of  $X$

$$H(X) = -\mathbb{E}[-\log(p(\mathbf{x}))] = -\sum_{\mathbf{x} \in \mathcal{X}} p(\mathbf{x}) \log(p(\mathbf{x})) \geq 0.$$

It describes the expected information content  $I(\mathbf{X})$  of  $X$ .

**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

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

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

KL( $X; p, q$ ) =  $H(p, q) - H(X)$ , where  $H$  uses  $p$ .

**11.2 - Jacobian Matrix**

**D. (Jacobi Matrix of a Map)** The Jacobian  $J_F$  of a map  $F: \mathbb{R}^n \rightarrow \mathbb{R}^m$  is defined as Numerator Layout

$$J_F := \begin{pmatrix} \frac{\partial F_1}{\partial x_1} & \frac{\partial F_1}{\partial x_2} & \dots & \frac{\partial F_1}{\partial x_n} \\ \frac{\partial F_2}{\partial x_1} & \frac{\partial F_2}{\partial x_2} & \dots & \frac{\partial F_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial F_m}{\partial x_1} & \frac{\partial F_m}{\partial x_2} & \dots & \frac{\partial F_m}{\partial x_n} \end{pmatrix} \in \mathbb{R}^{m \times n}$$

**- 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)  $\dot{x} = -\nabla f(x)$ .

**Com.** For an  $L$ -smooth function, GD with a fixed step size  $\eta \leq \frac{1}{L}$  is optimal.

**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  $\alpha_t = \frac{1}{t}$  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).

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

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.

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

**D. (Muon Optimizer)** is an optimizer that takes into account the matrix structure of model parameters. Specifically, while optimizing a loss function  $\mathcal{L}(W)$  that depends on a weight matrix  $W \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$ , at iteration  $t$  we follow an update:

$$\begin{aligned} M_t &= \mu M_{t-1} + \nabla \mathcal{L}(W_t) \\ P_t &= \text{orthogonalize}(M_t) \\ W_{t+1} &= W_t - \eta \sqrt{\frac{d_{\text{out}}}{d_{\text{in}}}} P_t. \end{aligned}$$

Muon is motivated by its ability to increase the scale of "rare directions" that are otherwise ignored during learning. Orthogonalization might require SVD ( $U\Sigma V^T$ , with unitary  $U \in \mathbb{R}^{m \times m}$ , rectangular diagonal matrix  $\Sigma \in \mathbb{R}^{m \times n}$ , and unitary  $V \in \mathbb{R}^{n \times n}$ ).

## 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  $\alpha, \beta$ ,  $T(\alpha f + \beta g) = \alpha T(f) + \beta T(g)$ .

**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

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

**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

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

The derivative  $\frac{\partial \mathcal{R}}{\partial y^{(l)}}$  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)**  $\forall i: \text{rot180}(x)_i = x_{(-i)}$ .

### - 12.6 - Pooling

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

### - 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

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

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

#### D. (Channels)

Ex. Here we have

• an input signal that is 2D with 3 channels ( $7 \times 3 \times 3$ ) (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  $3 \times 3 \times 2$  (convolution result x num convolutions)

#### - 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 ( $+2p$ ) 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  $I_i^l := \{j \mid W_{ij} \neq 0\}$  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 those which are connected to it and have a non-zero weight.

## 12.10 Convolutional Neural Networks

### - 12.10.1 - 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 ( $11 \times 11$ ).

• **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.,  $5 \times 5$ ,  $7 \times 7$ ) with stacks of small  $3 \times 3$  filters.

• **Reasoning:** Two stacked  $3 \times 3$  layers have the same receptive field as a  $5 \times 5$  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  $1 \times 1$ ,  $3 \times 3$ , and  $5 \times 5$  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. (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.

### - 12.10.2 - Convolutions in Sequences, NLP & Audio

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

$$C^t = \$$

## 15.2 — Large Language Models (LLMs)

LLMs make use of the Bayes' rule paradigm.

$$\Pr(\text{target} \mid \text{source}) \propto \Pr(\text{source} \mid \text{target}) \Pr(\text{target})$$

E.g., in machine translation it is beneficial to train a model mono-lingually (lots of training data). Thereby we make it easier for the conditional, bi-lingual model.

**D. (Perplexity)** is a common measure for LLM accuracy. Lower perplexity indicates better prediction.

$$PPL(X) = 2^{H(p,q)}$$

where  $H(p,q)$  is the **D. (Cross-Entropy)**,  $X = (x_0, x_1, \dots, x_t)$  is a tokenized sequence,  $p_\theta(x_i \mid x_{<i})$  is the model's predicted probability of token  $x_i$  given all previous tokens, and  $t$  is the sequence length.

$$H(p,q) = -\mathbb{E}_{x \sim p} [\log q(x)] = -\frac{1}{N} \sum_{i=1}^N \log q(x_i)$$

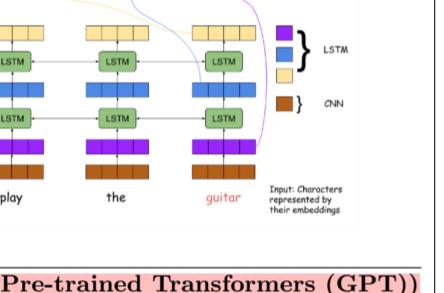
where  $p$  is the true data distribution,  $q$  is the model's predicted distribution, and  $N$  is the sequence length.

Bidirectional Encoder Representations from Transformers **T. (BERT)**

- is a transformer just bigger (24 blocks, 16 heads, 1024 latent space dim., 340M param.)
- task-independent basis ("Pre-Training") that enables lots of NLP tasks ("Fine-Tuning") → game-changer
- subword tokenization & special tokens (CLS, SEP)
- BERT is not autoregressive → mask 15% of the total words (80% mask, 10% random token, 10% unchanged)
- trained on two tasks: Self-supervised learning & Sentence pair classification

embeddings from language model **T. (ELMo)** is a word embedding method for representing a sequence of words as a corresponding sequence of vectors.

- Character-based model: morphology, OOV
- Left-to-right stacked LSTM (summarize left context of each word)
- Right-to-left stacked LSTM (summarize right context of each word)
- Shared LSTM layers, shared character embeddings and output embeddings
- collapse all layers  $2L + 1$  linearly (task-specific weights)



**T. (Generative Pre-trained Transformers (GPT))**

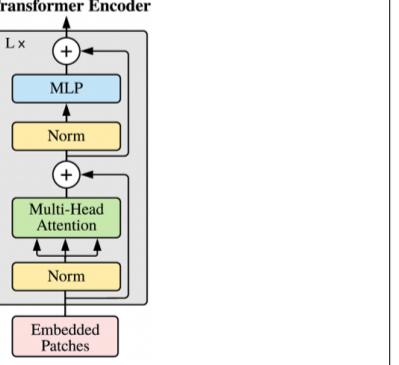
- relies on decoder part of transformer architecture
- trained with masked attention on next token prediction tasks
- prompts control generation. But at the heart is usually an alignment process, typically based on large amounts of human feedback.
- powerful zero-shot & few-shot learning capabilities

**15.3 — Vision Transformers**

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

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

- 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:  $\pi = \begin{pmatrix} 1 & & \\ \pi(1) & \pi(2) & \cdots & \pi(M) \end{pmatrix}$
- 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).

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

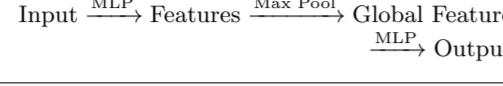
**Equivariant:** Output permutes exactly as input does (e.g., segmentation).

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

**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  $D = \operatorname{diag}(d_1, \dots, d_M)$  where  $d_m = \sum_n A_{nm}$  (number of neighbors).
- **Incidence Matrix (B):** Matrix of size  $|E| \times |V|$ . For an edge  $e = (i, j)$ , the entry  $b_{ij}$  is:

$$b_{ik} = \begin{cases} -1 & \text{if } k = i \text{ (source)} \\ +1 & \text{if } k = j \text{ (target)} \\ 0 & \text{otherwise} \end{cases}$$

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

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

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

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

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

**D. Attention Mechanism:**

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

## 16.5 — Weisfeiler-Lehman (WL) Test

Iterative graph isomorphism test based on color refinement. It can distinguish non-isomorphic graphs but not all of them.

**D. Connection to GNNs:**

- **1-WL Limitation:** Standard MP-GNNs (Message Passing GNNs) are at most as powerful as the 1-WL test in distinguishing non-isomorphic graphs.
- **GIN (Graph Isomorphism Network):** A GNN architecture designed to be maximally expressive, reaching the discriminative power of the 1-WL test.

## 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:

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

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  $k_*$  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").

**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  $\epsilon$ .

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

## 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.

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

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:

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

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).

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

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).

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

**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 Transformers

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

### D. (Tensor Multiplication)

$$\mathbf{T} = \underset{\in \mathbb{R}^{(a+b)}}{\mathbf{P}} \times_{\in \mathbb{R}^{(b+c)}} \mathbf{Q}$$

$$\mathbf{T} = \underset{\in \mathbb{R}^{r_1 \times \dots \times r_a \times s_1 \times \dots \times s_b}}{\mathbf{P}} \times_{\in \mathbb{R}^{s_1 \times \dots \times s_b \times t_1 \times \dots \times t_c}} \underset{\in \mathbb{R}^{s_1 \times \dots \times s_b \times t_1 \times \dots \times t_c}}{\mathbf{Q}}$$

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