

1 Probability

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

T. (Moment Generating Function)

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

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

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

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

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

for $t < 1/\beta$

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

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

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

D. (Normal Distribution)

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

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

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

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

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

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

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

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

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

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: $(\sum x_i y_i)^2 \leq (\sum x_i^2)(\sum y_i^2)$

Special case: $E[XY]^2 \leq E[X^2]E[Y^2]$

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

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

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

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

T. (Properties of Convex Functions)

- $f(y) \geq f(x) + \nabla f(x)^T (y - x)$
- $f'(x) \geq 0$
- Local minima are global minima, strictly convex functions have a unique global minimum

- If f, g are convex then $\alpha f + \beta g$ is convex for $\alpha, \beta \geq 0$
- If f, g are convex then $\max(f, g)$ is convex
- If f is convex and g is convex and non-decreasing then $g \circ f$ is convex

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

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

Relation to Optimization:

- Guarantee: Ensures a unique global minimum exists.

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

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

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

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

Implications for Optimization:

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

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

4.3 Vector-by-Vector

A, C, D, a, b, e not a function of \mathbf{x} .

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

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

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

$$\frac{\partial}{\partial \mathbf{x}} [\mathbf{a}^T \mathbf{A} \mathbf{g}(\mathbf{x})] = \frac{\partial \mathbf{a}^T}{\partial \mathbf{x}} \mathbf{A} \mathbf{g}(\mathbf{x}) + \mathbf{a}^T \frac{\partial \mathbf{A}}{\partial \mathbf{x}} \mathbf{g}(\mathbf{x})$$

$$\frac{\partial}{\partial \mathbf{x}} [\mathbf{b}^T \mathbf{A} \mathbf{x}] = \mathbf{A}^T \mathbf{b}$$

$$\frac{\partial}{\partial \mathbf{x}} [\mathbf{a}^T \mathbf{A} \mathbf{x}] = (\mathbf{A} + \mathbf{A}^T) \mathbf{x}$$

4.4 Scalar-by-Vector

Denominator Convention

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

11.3 Gradient Descent

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

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

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

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

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

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

Benefits:

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

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

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

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

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

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

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

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

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

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

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

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

D. (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 α, β , $T(\alpha f + \beta g) = \alpha(Tf) + \beta(Tg)$.

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

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

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

T. Any integral transform is a linear transform.

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

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

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

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

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

T. (Comms are commutative and associative)

T. (Comms 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

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

and its inverse as

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

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

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

and then transformed back using the inverse Fourier transform.

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

12.2 Discrete Time Convolutions

D. (Discrete Convolution)

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

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

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

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

D. (Multidimensional Discrete Convolution)

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

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

D. (Discrete Cross-Correlation)

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

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

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

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

1. **reduce** the spatial dimensions (sub-sampling)
2. **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., $5 \times 5, 7 \times 7$) with stacks of small 3×3 filters.

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

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

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

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

D. (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* \times Embedding Dim).

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

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

Backpropagations of Convolutions as Convolutions

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

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

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.

12.7 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 ($7 \times 7 \times 3$) (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)

D. (Max-Pooling)

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

12.9 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 outputs 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.

Dynamics: Uses a linear recurrence relation (diagonalizable):

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

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

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

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

Diagonalization: If $A = PAP^{-1}$, we can operate in the diagonal basis $\xi^t = P^{-1}z^t$:

$$\xi^{t+1} = \Lambda\xi^t + \tilde{B}x^t$$

Stability: Eigenvalues of A (diagonal elements of Λ) must be initialized inside the complex unit circle ($|\lambda| \leq 1$) to prevent explosion.

13.1.4 Sequence Learning

Generative Modeling: Decomposes joint probability of a sequence:

$$P(x_1, \dots, x_T) = \prod_{t=1}^T P(x_t | x_1, \dots, x_{t-1})$$

D. (Multi-Headed Attention)

- runs multiple attention mechanisms in parallel (*heads*)
- Replicate matrices U_k, U_Q, W, R , r times, perform the attention-based propagation $X \mapsto \Xi_i$ ($1 \leq i \leq r$) and then concatenate the matrices Ξ_i along the feature dimension
- each head itself is a key-value attention

$$\begin{aligned} F_1(\xi, (x^t, z^t)) \\ G(\xi, (x^t, z^t)_{i=1}^r) = W \\ F_r(\xi, (x^t, z^t)) \\ F_j(\xi, (x^t, z^t)) = F(W_j^T \xi, (W^x_j x^t, W^z_j z^t)) \end{aligned}$$

T. (Feature Transformation)

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

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

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

D. (Cross-Attention) to the representations produced by the encoder. Often decoder and encoder are of the same depth and cross-attention is implemented across layers of the same depth. Note that in cross-attention the queries emerge from the decoder, whereas keys and values are derived from the encoder states.

15 Attention & Transformers

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

$$x_1, \dots, x_T, \quad X = [x_1, \dots, x_T] \in \mathbb{R}^{n \times T}$$

The fundamental idea is to map the non-contextualized (or less-contextualized) embeddings to contextualized (more-contextualized) representations.

$$\xi_1, \dots, \xi_T, \quad \Xi = [\xi_1, \dots, \xi_T] \in \mathbb{R}^{m \times T}$$

T. (Masked Attention, Decoder)

When transformers operate in coupled encoder-decoder pairs the decoder is operated in an autoregressive fashion, where the next output for the t -th token depends on the encoder, but also the produced token representations for $s < t$. This can be accomplished in a transformer by simply restricting attention to the past (no masks are worn).

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

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

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

GRU

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.

D. (Attention weights) α_{st} cleverly indexes the "memory" of the model and retrieves the important bits. Note that attention weights have a **source** (where attention emerges, index s) and a **target** (where attention extracts information, index t), with Value matrix W .

$$\xi_s \equiv \sum_t \alpha_{st} W x_t, \quad \alpha_{st} \geq 0, \quad \sum_t \alpha_{st} = 1$$

D. (Attention Matrix) summarizes the Attention weights

$$A = (a_{st}) \in \mathbb{R}^{T \times T}, \quad \text{s.t. } \Xi = WA^\top$$

Bi-Linear Query-Key Matching Function Query matrix U_Q , key value U_K project X to a query and key matrix $Q = U_Q X, \quad K = U_K X, \quad U_Q, U_K \in \mathbb{R}^{q \times n}$

Produce **D. (Matching Matrix)** via inner products of queries and keys

$$Q^\top K = X^\top U_Q^\top U_K X \in \mathbb{R}^{T \times T}, \quad \text{rank}(Q^\top K) \leq q$$

The matching matrix already matches the dimensions of the attention matrix. But is commonly passed through a softmax (to normalize & guarantee non-negativity).

D. (Transformer)

is an architecture that takes the idea of attention to the extreme. They are computationally efficient in some aspects

E.g. in machine translation it is beneficial to train a model monolingually (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, $p_\theta(x_i | x_{<i})$ 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.

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)
- Stacked 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

D. (Optimization)

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

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

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

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

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

14 Optimization Challenges in NNs: Curvatures

14.1 Optimization Challenges in NNs: Curvatures

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

$$X \mapsto \Xi \mapsto F_\theta(\Xi), \quad F_\theta$$

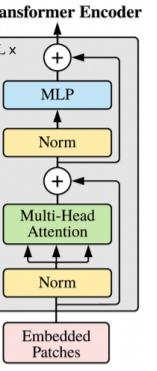
15.3 — Vision Transformers

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

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

• 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 per row/col) or **Cauchy Two-Line Notation** π :

• **Cauchy Notation:** $\pi = (\pi(1) \ \pi(2) \ \dots \ \pi(M))$

• **Matrix Properties:** $P^{-1} = P^T$ and $PP^T = I$.

• **Row Permutation:** PX permutes rows (samples).

• **Column Permutation:** X^TP permutes columns (features).

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

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

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

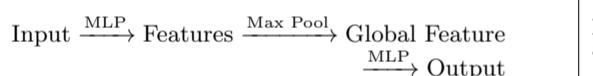
• **Equivariant:** Output permutes exactly as input does (e.g., segmentation).

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

D. Deep Sets Theorem: Any invariant function f can be decomposed into an element-wise encoder ϕ and an invariant aggregator ρ (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 $G = (V, E)$ 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 $\{n, m\} \in E$, 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 (\bar{A}): Standard symmetric normalized formulation. We define the self-loop adjacency $\bar{A} = A + I$ and corresponding degree matrix $\bar{D}_{mm} = \sum_n \bar{A}_{nm} = d_m + 1$.

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

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

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

Com. Limitations:

• **Oversmoothing:** In deep GCNs, repeated mixing causes all node embeddings to converge to the same value.

• **Oversquashing:** Exponentially growing information from distant nodes fails to fit into fixed-size vectors.

— 16.3 — Spectral Graph Theory

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

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

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

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

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

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 α_{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 θ_0 using a first-order Taylor expansion:

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

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

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

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

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

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

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

Linear Dynamics: The training dynamics become identical to Kernel Ridge Regression with kernel k_∞ . The evolution of outputs follows:

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

— 17.2 — Bayesian DNNs

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

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

Predictions are made by marginalizing over the posterior: $p(y|x) = \int p(y|x, \theta)p(\theta|S)d\theta$.

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

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

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:

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

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. 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 .

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

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

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

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

— 18. Chain-Rule and Jacobians for Tensors

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

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

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

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

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

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

Com. Limitations:

• **Oversmoothing:** In deep GCNs, repeated mixing causes all node embeddings to converge to the same value.

• **Oversquashing:** Exponentially growing information from distant nodes fails to fit into fixed-size vectors.

— 16.3 — Spectral Graph Theory

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

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

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

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

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

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.

— 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 .

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

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

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

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.

— 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 .

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

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