

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

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

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

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

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

$\mathbf{XX}^T \geq 0$ (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.)

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

T. (Moment Generating Function)

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

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

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

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

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

D. (Normal Distribution)

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

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

T.

$P(\lfloor \mathbf{a}_2 \rfloor) = \mathcal{N}(\lfloor \mathbf{a}_2 \rfloor \mid \lfloor \mathbf{u}_2 \rfloor, [\Sigma_{21}^{11}, \Sigma_{22}^{12}, \Sigma_{21}^{22}])$

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

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

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

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

$P(\mathbf{a}_2 | \mathbf{a}_1 = \mathbf{z}) =$

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

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

2 Analysis

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

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

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

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

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

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

T. (Properties of Convex Functions)

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

$\cdot f'(x) \geq 0$

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

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

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

\cdot 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:

\cdot Guarantee: Ensures a unique global minimum exists.

\cdot Convergence: Gradient Descent on strongly convex (and Lipschitz smooth) functions guarantees a linear convergence rate ($O(c^k)$ 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:

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

\cdot 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.

4.2 Scalar-by-Vector

Denominator Convention

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

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

$$\frac{\partial}{\partial \mathbf{x}} [\mathbf{f}(\mathbf{x})^T \mathbf{g}(\mathbf{x})] = \frac{\partial \mathbf{f}(\mathbf{x})}{\partial \mathbf{x}}^T \mathbf{g}(\mathbf{x}) + \frac{\partial \mathbf{g}(\mathbf{x})}{\partial \mathbf{x}}^T \mathbf{f}(\mathbf{x}) = \mathbf{J}_f \mathbf{g}(\mathbf{x}) + \mathbf{J}_g \mathbf{f}(\mathbf{x})$$

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

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

$$\frac{\partial}{\partial \mathbf{x}} [\mathbf{x}^T \mathbf{A}] = 2\mathbf{x}$$

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

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

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

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

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

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

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

4.2 Universal Approximation

T. (Universal Approximation Theorem)

MLPs with one hidden layer and a non-polynomial activation function can approximate any continuous function on a compact subset of \mathbb{R}^d to arbitrary accuracy $\epsilon > 0$. Let φ be a non-constant, bounded, and continuous activation function (e.g., Sigmoid, ReLU). There exist weights v, w, b and an integer N (number of neurons) such that the network output $F(x)$ satisfies the condition:

$$|f(x) - F(x)| < \epsilon, \quad \forall x \in K$$

D. (Uniform Approximation)

This specific type of convergence guarantees that the maximum error across the entire domain K is bounded by ϵ . It is stricter than pointwise approximation.

T. (Weierstrass Approximation)

Let f be a continuous real-valued function defined on a closed interval $[a, b]$. For every $\epsilon > 0$, there exists a polynomial $P(x)$ such that for all $x \in [a, b]$:

$$|f(x) - P(x)| < \epsilon$$

Implication: Polynomials are dense in the space of continuous functions $C[a, b]$. This is the foundation for Universal Approximation, as neural networks can approximate polynomials.

T. (Dimension Lifting (Leshno's Theorem))

By the universal approximation theorem, we know that we can approximate $C(\mathbb{R})$ with a single hidden layer of ReLUs. The lifting theorem then allows us to lift the dimension of the function space to a higher dimension, i.e. $\text{span}(\{\phi(ax + b) : a, b \in \mathbb{R}\})$ universally approximates $C(\mathbb{R}^n)$.

T. (Montufar (Linear Regions))

The number of linear regions carved out by a Deep ReLU Network grows exponentially with depth L , but only polynomially with width n . For a network with L layers and width n (where $n \geq d$):

$$\#\text{Regions} = O\left(\left(\frac{n}{d}\right)^{(L-1)d} n^d\right)$$

Com. Deep networks are exponentially more expressive (in terms of complex decision boundaries) than shallow networks of the same parameter count.

T. (Shekhman (ReLU Basis))

Any continuous piecewise linear function $g(x)$ on $[0, 1]$ (a polygonal line) with breakpoints (knots) $t_1 < \dots < t_k$ can be represented exactly as a weighted sum of ReLU units:

$$g(x) = a + bx + \sum_{i=1}^k c_i \text{ReLU}(x - t_i)$$

Com. A single hidden layer of ReLUs acts as a universal basis for 1D splines. This establishes the theoretical link between splines and ReLU networks.

T. (Barron's Theorem)

For a function f with a finite Fourier moment C_f (a measure of smoothness), there exists a neural network f_n with one hidden layer of n sigmoidal units such that the integrated squared error is bounded by:

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).

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

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.

$$E[g^2]_t = \beta E[g^2]_{t-1} + (1 - \beta)(\nabla J(\theta_t))^2$$

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

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

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

12 Convolutional Neural Networks

12.1 Convolutional Layers

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

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

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

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

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

T. Any integral transform is a linear transform.

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

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

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

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

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

T. (Convs are commutative and associative)

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

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

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

12.2.1 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).

12.2.2 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

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

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

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

12.3 Convolution via Matrices

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

12.4 Border Handling

There are different options to do this

- **D. (Padding of p)** Means we extend the image (or each dimension) by p on both sides (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 I_i^l of x_i^l)

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

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

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

D. (Recursive Receptive Field Formula)

The receptive field size r_l of layer l is calculated as:

$$r_l = r_{l-1} + (k_l - 1) \times j_{l-1}$$

Where:

- r_l : Receptive field size of the current layer.
- r_{l-1} : Receptive field size of the previous layer.
- k_l : Kernel size of the current layer.
- j_{l-1} : Cumulative stride product of all previous layers ($j_{l-1} = \prod_{i=1}^{l-1} s_i$).

We have $\forall j \neq I_i^l: \frac{\partial x_i^l}{\partial x_j^{l-1}} = 0$,

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

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

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 :

$$z_t = \phi(U z_{t-1} + V x_t)$$

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.

13.2 CNNs in Computer Vision

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

1. that *reduce* the spatial dimensions (sub-sampling)
2. 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

13.1.1 Classic CNN Architectures

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

Structure: 2 Convolutional layers (with Average Pooling) followed by 5 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 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.

13.1.2 Structural Variants

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

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

13.1.3 Linear Recurrent Units (LRU)

Motivation: Bridges the gap between RNNs (inference efficiency) and Transformers (training parallelizability).

Dynamics: Uses a linear recurrence relation (diagonalizable):

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

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

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

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

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

13.1.4 Sequence Learning

Generative Modeling: Decomposes joint probability of a sequence:

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

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

Seq2Seq (Encoder-Decoder):

- **Encoder:** Compresses input sequence into a fixed context vector z_T .
- **Exploding Gradient:** If largest singular value $\sigma_{max}(U) > 1$.
- **Vanishing Gradient:** If $\sigma_{max}(U) < 1$. This makes learning long-term dependencies difficult.

13.1.5 Structural Variants

D. (Deep RNNs): Stacking multiple RNN layers to increase representational power. The output of layer l becomes the input to layer $l+1$.

14 Optimization

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

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

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

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

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

14.1 Optimization Challenges in NNs: Curvatures

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.

14.1.1 Convergence Rates

Under certain conditions SGD converges to the optimum:

- If we have a convex, or strongly convex objective,
- and if we have Lipschitz continuous gradients,
- and a decaying learning rate, s.t.

$$\sum_{t=1}^{\infty} \eta_t = \infty, \quad \sum_{t=1}^{\infty} \eta_t^2 < \infty$$

we get far enough our steps get always smaller

14.1.2 Gated Memory

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

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

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

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

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

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

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.

14.1.4 Attention

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

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

So, even if the convergence rates are not super nice, thanks to the cheap gradient computation (only one example at the time), we may even converge faster than computing the gradient on the full dataset everytime.

15 Attention & Transformers

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 *Permuation Matrix* P (one per row/column) or *Cauchy Two-Line Notation* π :

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

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

16.2 Neural Tangent Kernel (NTK)

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:

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

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 .

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

16.4 Attention GNNs (GAT)

Learn dynamic edge weights α_{ij} instead of static adjacency.

D. Attention Mechanism:

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

17 Theory

17.1 Neural Tangent Kernel (NTK)

D. Linearized DNNs: 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:

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

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.

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

This mimics a physical system with friction and thermal noise.

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.

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

D. Deep Sets Theorem: Any invariant function f can be decomposed into an element-wise encoder $\phi</$

18 Chain-Rule and Jacobians for Tensors

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

D. (Tensor Multiplication)

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

$$r_1 \times \dots \times r_a \times_{s_1 \times \dots \times s_b} s_1 \times \dots \times s_c =$$

$$\mathbf{P}_{r_1 \times \dots \times r_a \times s_1 \times \dots \times s_b} \times_{s_1, \dots, s_b} \mathbf{Q}_{s_1 \times \dots \times s_b \times t_1 \times \dots \times t_c}$$

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

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

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

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

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

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

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):

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

Encoder (q_ϕ): Maps input x to latent parameters μ, Σ .

Decoder (p_θ): Reconstructs x from sampled z .

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

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

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

- 19.2 - Normalizing Flows

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

D. Change of Variables:

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

Or in log-domain (maximizing likelihood):

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

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

$$y_{1:d} = x_{1:d}$$

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

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

- 19.3 - Gen. Adversarial Networks (GANs)

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

D. (Minimax Objective):

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

Optimality:

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

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 .

- 19.4 - Denoising Diffusion Models (DDPM)

Learns to reverse a gradual noising process.

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

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

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 θ :

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

D. (Simplified Objective): Instead of predicting the image mean μ , we predict the noise ϵ added at step t :

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

20 Ethics

- 20.1 - Robustness

D. (Adversarial examples) (Classification Perspective) Input x , label y , budget ϵ , 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 $\hat{y}(x + \delta) = t \neq y$
- Loss-based: $\max_{\|\delta\|_p \leq \epsilon} \mathcal{L}(f(x + \delta), y)$

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

D. (Norms)

- $\|x\|_p = (\sum_{i=1}^d |x_i|^p)^{1/p}$
- $\|x\|_\infty = \max_{i=1, \dots, d} |x_i|$

T. (Min L₂ adversarial perturbation): Robustness increases with margin $|w^\top x + b|$ and decreases with $\|w\|_2$

$$\delta^* = -\frac{w^\top x + b}{\|w\|_2^2} w, \quad \|\delta^*\|_2 = \frac{|w^\top x + b|}{\|w\|_2}$$

T. (L_{infinity} threat model): If $\|w\|_2 \leq \epsilon$, then $w^\top \delta$ is minimized by choosing $\delta = -\text{sign}(yw)$.

Multiclass: $f_k(x) = w_k^\top x + b_k$, A Perturbation δ is (untargeted) adversarial if it violates at least one inequality:

$$\exists j \neq y: \quad (w_y - w_j)^\top (x + \delta) + (b_y - b_j) \leq 0$$

D. (Margin to class $j \neq y$) $m_j(x) := (w_y - w_j)^\top x + (b_y - b_j)$

Nearest Competing Class: $j^*(x) := \operatorname{argmin}_{j \neq y} \frac{m_j(x)}{\|w_y - w_j\|_2}$

Neural Networks: Local Linearization

D. (1st Order Approx.): $f(x + \delta) \approx f(x) + J(x)\delta$

$J(X) \in \mathbb{R}^{K \times d}$ is the Jacobian with rows $\nabla_x f_k(x)^\top$.

Fast-grad.-sign-method (FGSM) Attack: max. Loss.

$\delta = \epsilon \operatorname{sign}(\nabla_x \mathcal{L}(f(x), y)), x^{\text{adv}} = x + \delta_{\text{FGSM}}$

Projected Grad. Descent (PGD) Attack: iterative refinement of FGSM with projection back onto threat set.

$$\delta_{t+1} = \operatorname{Proj}_{\|\delta\|_p \leq \epsilon}(\delta_t + \alpha g_t), \quad g_t \in \partial_\delta \mathcal{L}(f(x + \delta_t), y).$$

For $p = \infty$, a common choice is $g_t = \operatorname{sign}(\nabla_\delta \mathcal{L}(f(x + \delta_t), y))$.

Adversarially robust training: Instead of X we evaluate at worst-case loss within neighborhood.

$$\min_f \mathbb{E} \left[\max_{\delta \in \mathcal{S}} \ell(Y, f(X + \delta)) \right], \quad \mathcal{S} = \{\delta: \|\delta\|_p \leq \epsilon\}$$

T. (Distribution (P, Q) Shift): Data P → Deployment Q.

Seek $\sup_{Q \in \mathcal{U}(P)} \mathbb{E}_Q [\ell(f(Z))]$

Robust statistics studies the stability of statistical procedures under small deviations from an assumed model.

D. (Huber's contamination model): for arbitrary contaminating dist. Q .

$$P_\epsilon = (1 - \epsilon)P + \epsilon Q$$

T. (Distributionally Robust Optim.) also known as (DRO) seeks to find a model f that is robust to small deviations from an assumed model P .

It is also known as (SAGE), $\mathcal{Q}(S)$ is a performance-level quantity, attributes risk reduction:

$$Q_{\text{SAGE}}(S) = \mathbb{E}[f(X) | X_S = x_S]$$

D. (Shapley Additive Global importance)

It is also known as (SAGE), $\mathcal{Q}(S)$ is a performance-level quantity, attributes risk reduction:

$$Q_{\text{SAGE}}(S) = -\mathcal{R}(f_S), \quad \mathcal{R}(f_S) = \mathbb{E}[\ell(Y, f_S(X_S))]$$

SHAP and SAGE provide additive explanations (each variable is assigned a contribution, and the sum of these contributions recovers the total effect), because Shapley Values $\{\phi_j\}_{j=1}^p$ satisfy the

T. (Efficiency (Additivity) Property):

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

Causal effect: what is the causal effect of x_j on y ?

D. (Causal ordering) induces a factorization of the joint distribution with $\text{Pa}(X_j)$ denoting the parents(direct causes) of X_j :

$$P(X_1, \dots, X_p) = \prod_{j=1}^p P(X_j | \text{Pa}(X_j))$$

The joint distribution corresponds to a sequential sampling procedure. This ordering reflects causal, not merely statistical, dependencies.

D. (Structural equation models (SEM)) U_j is an exogenous noise variable:

$$X_j = f_j(\text{Pa}(X_j), U_j)$$

Intervention in the generative view replaces the sampling step $X_j \leftarrow x_j$.

Intervention in the SEM view replaces the structural equation $X_j = f_j(\text{Pa}(X_j), U_j)$ by $X_j = x_j$.

Counterfactuals correspond to comparing $Y(X)$ and $Y(\text{do}(X_j = x'_j))$ within the same underlying causal model.

- 20.2 - Interpretability

Interpretability aims to understand the behaviour of a fixed function f or how the learned function f_S depends on a subset of variables $S \subseteq \{1, \dots, p\}$

Examples of Local questions:

ceteris paribus: How does the prediction $f(x)$ change when varying a feature x_j while keeping all other features fixed?

$$x'_j \mapsto f(x'_j, x_{-j}), \quad x = (x_j, x_{-j})$$

Missing Information: How does the prediction $f(x)$ change when a feature x_j is not observed? Use the marginalization theorem.

Intervention: How would the target value change if one could intervene and change the value of a feature x_j ? Use the notion of sensitivity.

T. (Marginalization) when the variable x_j is unobserved, replace the prediction $f(X)$ with

$$\mathbb{E}[f(X) | X_{-j} = x_{-j}]$$

The contribution of x_j can be assessed via: $f(X) - \mathbb{E}[f(X) | X_{-j}]</math$