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

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

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

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

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

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

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

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

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

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

$\mathrm{Var}[\mathbf{A}\mathbf{X}]=\mathbf{A}\,\mathrm{Var}[\mathbf{X}]\,\mathbf{A}^\top \qquad \mathrm{Var}[aX+b]=a^2\,\mathrm{Var}[X]$

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

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

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

T. The moment generating function (MGF) $\psi_X(t)=\mathbb{E}[e^{tX}]$ characterizes the distr. of a rv

$$Be(p) \quad pe^t+(1-p)$$
$$\mathcal{N}(\mu,\sigma) \quad \exp\left(\mu t+\frac{1}{2}\sigma^2t^2\right)$$
$$Bin(n,p) \quad (pe^t+(1-p))^n$$
$$Gam(\alpha,\beta) \quad \left(\frac{1}{a-\beta t}^{\alpha}\right)$$

for $t<1/\beta$

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

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 $Y=\sum_{i=1}^n a_iX_i$ is $M_Y(t)=\prod_{i=1}^n M_{X_i}(a_it)$.

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

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

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

T. $P\left(\left[\begin{smallmatrix}\mathbf{a}_1\\ \mathbf{a}_2\end{smallmatrix}\right]\right)=\mathcal{N}\left(\left[\begin{smallmatrix}\mathbf{a}_1\\ \mathbf{a}_2\end{smallmatrix}\right]\,\middle|\,\left[\begin{smallmatrix}\mathbf{u}_1\\ \mathbf{u}_2\end{smallmatrix}\right],\begin{bmatrix}\boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12}\\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22}\end{bmatrix}\right)$

$\mathbf{a}_1,\mathbf{u}_1\in\mathbb{R}^e$, $\boldsymbol{\Sigma}_{11}\in\mathbb{R}^{e\times e}$ p.s.d. $\boldsymbol{\Sigma}_{12}\in\mathbb{R}^{e\times f}$ p.s.d.
$\mathbf{a}_2,\mathbf{u}_2\in\mathbb{R}^f$, $\boldsymbol{\Sigma}_{22}\in\mathbb{R}^{f\times f}$ p.s.d. $\boldsymbol{\Sigma}_{21}\in\mathbb{R}^{f\times e}$ p.s.d.

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

T. (Chebyshev) Let X be a rv with $\mathbb{E}[X]=\mu$ and variance $\mathrm{Var}[X]=\sigma^2<\infty$. 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)

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

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

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

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 $\forall\mathbf{x},\mathbf{x}'\in\Sigma,\ \forall\lambda\in[0,1]\colon\quad \lambda\mathbf{x}+(1-\lambda)\mathbf{x}'\in S$.

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

D. (Convex Function) A function $f\colon S\to\mathbb{R}$ 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** ($O(c^k)$ 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:

- **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)

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

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

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

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

D. (L-Lipschitz Continous Function) Given two *metric spaces* (X,d_X) and (Y,d_Y), a function $f\colon X\to Y$ is called *Lipschitz continuous*, if there exists a real constant $L\in\mathbb{R}_0^+$ (*Lipschitz constant*), such that

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

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

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

D. (PL Condition) A differentiable function $f(x)$ with global minimum f^* satisfies the μ-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:

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

3 Linear Algebra

Kernels are positive semi-definite matrices.

D. (Positive Semi-Definite Matrix) A symmetric matrix $\mathbf{A}\in\mathbb{R}^{n\times n}$ is PSD if for all non-zero vectors $\mathbf{x}\in\mathbb{R}^n$: $\mathbf{x}^T\mathbf{A}\mathbf{x}\geq 0$ **Properties**:

- All eigenvalues $\lambda_i\geq 0$.
- The trace $\mathrm{Tr}(A)\geq 0$ and determinant $\det(A)\geq 0$.
- Cholesky Decomposition exists: $A=LL^T$.

T. (Sylvester Criterion) A $d\times d$ matrix is positive semi-definite if and only if all the upper left $k\times k$ for $k=1,\dots,d$ have a positive determinant.

Negative definite: $det<0$ for all odd-sized minors, and $det>0$ for all even-sized minors

Otherwise: indefinite.

D. (Trace) of $\mathbf{A}\in\mathbb{R}^{n\times n}$ is $\mathsf{Tr}(\mathbf{A})=\sum_{i=1}^n a_{ii}$.

Properties:

- $\mathrm{Tr}(A)=\sum_i\lambda_i$ (sum of eigenvalues).
- Cyclic property: $\mathrm{Tr}(ABC)=\mathrm{Tr}(BCA)=\mathrm{Tr}(CAB)$.
- Linear: $\mathrm{Tr}(A+B)=\mathrm{Tr}(A)+\mathrm{Tr}(B)$ and $\mathrm{Tr}(cA)=c\mathrm{Tr}(A)$.
- $\mathrm{Tr}(A)=\mathrm{Tr}(A^T)$.

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

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

Properties:

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

4 Derivatives

4.1 Numerator and Denominator Convention

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

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

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

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

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

4.2 Scalar-by-Vector

Denominator Convention

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

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

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

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

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

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

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

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

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

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

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

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

4.3 Vector-by-Vector

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

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

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

$\frac{\partial}{\partial\mathbf{x}}[\mathbf{x}\odot\mathbf{a}]=\mathrm{diag}(\mathbf{a})$

$\frac{\partial}{\partial\mathbf{x}}[\mathbf{a}]=\mathbf{0}$

$\frac{\partial}{\partial\mathbf{x}}[\mathbf{x}]=\mathbf{I}$

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

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

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

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

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

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

4.4 Scalar-by-Matrix

$\frac{\partial}{\partial\mathbf{X}}[\mathbf{a}^\top\mathbf{X}\mathbf{b}]=\mathbf{a}\mathbf{b}^\top$

$\frac{\partial}{\partial\mathbf{X}}[\mathbf{a}^\top\mathbf{X}^\top\mathbf{b}]=\mathbf{b}\mathbf{a}^\top$

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

$\frac{\partial}{\partial\mathbf{X}}[\mathbf{a}^\top\mathbf{X}^\top\mathbf{X}\mathbf{b}]=\mathbf{X}(\mathbf{a}\mathbf{b}^\top+\mathbf{b}\mathbf{a}^\top)$

$\frac{\partial}{\partial\mathbf{X}}[\mathsf{Tr}(\mathbf{X})]=\mathbf{I}$

$\frac{\partial}{\partial\mathbf{X}}[\mathsf{Tr}(\mathbf{A}\mathbf{X}\mathbf{B})]=\mathbf{A}^\top\mathbf{B}^\top$

$\frac{\partial}{\partial\mathbf{X}}[\mathsf{Tr}(\mathbf{A}\mathbf{X}^\top\mathbf{B})]=\mathbf{B}\mathbf{A}$

4.5 Vector-by-Matrix (Generalized Gradient)

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

5 Information Theory

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

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

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

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

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

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

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

D. (Kullback-Leibler Divergence)

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

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

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

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

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

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

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

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

D. (Jensen-Shannon Divergence)

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

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

C. The JSD is symmetric!

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

6 Activation Functions

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

D. (Tanh)

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

D. (ReLU)

$$\mathrm{ReLU}(x)=\max(x,0)\qquad\mathrm{ReLU}'(x)=\begin{cases}1 & x\geq 0\\ 0 & x\leq 0\end{cases}$$

D. (Sigmoid/Logistic)

$$\sigma(\mathbf{x})_i=\frac{1}{1+e^{-x_i}}\in(0,1)\qquad\frac{\partial\sigma(\mathbf{x})}{\partial\mathbf{x}}=\sigma(\mathbf{x})\odot(1-\sigma(\mathbf{x}))$$

$$\sigma^{-1}(y)=\ln\left(\frac{y}{1-y}\right)\qquad\nabla_{\mathbf{x}}\sigma(\mathbf{x})=\mathbf{J}_\sigma(\mathbf{x})=\mathrm{diag}(\sigma(\mathbf{x})\odot(1-\sigma(\mathbf{x})))$$

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

Connection between Sigmoid and Tanh (Equal Representation Strength)

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

D. (Softmax) $\mathrm{softmax}(\mathbf{x})_i=f(x)_i$

$$f(x)_i=\frac{e^{x_i}}{\sum_{j=1}^K e^{x_j}}\qquad\frac{\partial f(x)_i}{\partial x_j}=\begin{cases}-f(x)_if(x)_j & i\neq j\\ f(x)_i-f(x)_if(x)_j & i=j\end{cases}$$

$$\nabla_xf(x)=\mathbf{J}_f(x)=\mathrm{diag}(f(x))-f(x)f(x)^T$$

7 Connectionism

McCulloch & Pitts (1943): MP-Neuron. Abstract model of neurons as linear threshold units. Inputs $x\in\{0,1\}^n$, synapses $\sigma\in\{-1,1\}^n$. $f(x)=\mathbb{I}(\sum\sigma_ix_i\geq\theta)$. No learning (fixed weights).

Perceptron (Rosenblatt 1958): Pattern recognition. $f(\mathbf{x})=\mathrm{sign}(\mathbf{w}^\top\mathbf{x}+b)$. Update rule (if misclassified): $\mathbf{w}_{new}\leftarrow\mathbf{w}_{old}+y_i\mathbf{x}_i$, $b_{new}\leftarrow b_{old}+y_i$. Cannot learn the XOR function.

T. (Novikoff) Guaranteed convergence if data is linearly separable.

T. (Cover) Capacity of perceptron in $\mathbb{R}^n$ is $2n$ random patterns. Max dichotomies of S (s points in gen. pos.) by linear classifier: $C(s+1,n)=2\sum_{i=0}^n\binom{s}{i}$.

Assymptotic Shattering:

$$\frac{C(s,n)}{2^s}=\begin{cases}1 & \text{if } s\leq n\\ 1-\mathcal{O}(e^{-n}) & \text{if } n<s<2n\\ \frac{1}{2} & \text{if } s=2n\\ \mathcal{O}(e^{-n}) & \text{otherwise}\end{cases}$$

Hopfield Networks: Associative Memory. Hebbian Learning: "Neurons that fire together, wire together". $w_{ij}\propto\sum x_ix_j$. Energy Min.: $E(\mathbf{x})=-\frac{1}{2}\mathbf{x}^\top\mathbf{W}\mathbf{x}$.

PDP (Rumelhart et al. 1986): Parallel Distributed Processing. Introduction of Backpropagation (Generalized Delta Rule). Differentiable activations allow δ propagation to hidden layers.

8 Regression

D. (Linear Regression): $f(x)=w^Tx+b$

$$\ell(w)=\frac{1}{2n}\sum_{i=1}^n(y_i-f(x_i))^2$$

Analytically solvable: $w^*=(X^TX)^{-1}X^Ty$

D. (Ridge Regression): $f(x)=w^Tx+b$ with ℓ_2 regularization

$$\ell(w)=\frac{1}{2n}\sum_{i=1}^n(y_i-f(x_i))^2+\lambda\|w\|^2$$

Analytically solvable: $w^*=(X^TX+\lambda I)^{-1}X^Ty$

D. (Lasso Regression): $f(x)=w^Tx+b$ with ℓ_1 regularization

$$\ell(w)=\frac{1}{2n}\sum_{i=1}^n(y_i-f(x_i))^2+\lambda\|w\|_1$$

D. (Logistic Regression): $f(x)=\sigma(w^Tx+b)$

It minimizes the cross-entropy loss (negative log-likelihood), which acts as a surrogate loss for the 0/1 classification error.

$$\ell(w)=-\sum_{i=1}^n(y_i\log(\sigma(w^Tx_i))+(1-y_i)\log(1-\sigma(w^Tx_i)))$$

9 Approximation Theory

9.1 Compositions of Maps

D. (Linear Function) A function $f\colon\mathbb{R}^n\to\mathbb{R}^m$ is a linear function if the following properties hold

- $\forall\mathbf{x},\mathbf{x}'\in\mathbb{R}^n\colon f(\mathbf{x}+\mathbf{x}')=f(\mathbf{x})+f(\mathbf{x}')$
- $\forall\mathbf{x}\in\mathbb{R}\,\forall\alpha\in\mathbb{R}\colon f(\alpha\mathbf{x})=\alpha f(\mathbf{x})$

T. $f\colon\mathbb{R}^n\to\mathbb{R}$ is linear $\iff f(\mathbf{x})=\mathbf{w}^\top\mathbf{x}$ for some $\mathbf{w}\in\mathbb{R}^n$

D. (Hyperplane)

$H:=\{\mathbf{x}\mid\langle\mathbf{w},\mathbf{x}-\mathbf{p}\rangle=0\}=\{\mathbf{x}\mid\langle\mathbf{w},\mathbf{x}\rangle=b\}$

where $b=\langle\mathbf{w},\mathbf{p}\rangle$. $\mathbf{w}$ =normal vector, $\mathbf{p}$ points onto a point on the plane.

9.2 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:

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

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. $\mathrm{span}(\{\phi(ax+b):a,b\in\mathbb{R}\})$ 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. (Shekhtman (ReLU Basis)) Any continuous piecewise linear function $g(x)$ on $[0,1]$ (a polygonal line) with breakpoints (knots) $t_1<\cdots<t_k$ can be represented exactly as a weighted sum of ReLU units:

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

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:

$$||f-f_n||_{L_2}^2\leq\frac{(2C_f)^2}{n}$$

Com. The approximation error decays at a rate of $O(1/\sqrt{n})$, which is independent of the input dimension d. This suggests neural networks can overcome the "Curse of Dimensionality" for specific classes of functions.

T. (Zaslavsky's Theorem) The number of distinct regions $R(m,d)$ created by an arrangement of m hyperplanes in a d-dimensional space is bounded by:

$$R(m,d)\leq\sum_{i=0}^d\binom{m}{i}$$

Com. In the context of ReLU networks, this bounds the number of linear regions a single layer with m neurons can create. For large m, this approximates to $O(m^d)$, showing polynomial growth with width (contrast with Montufar's exponential growth with depth).

9.2.1 Maxout Networks

In 2013 k-Hinges were re-discovered under the name of *Maxout* by Goodfellow et al.

D. (Maxout) is just the max non-linearity applied to k groups of linear functions. So the input $[1:d]$ (of the previous layer) is *partitioned* into k sets $A_1,\dots,A_k$, and then we define the activations $G_j(\mathbf{x})$ for $j\in\{1,\dots,k\}$ as

$G_j(\mathbf{x})=\max_{i\in A_j}\{\mathbf{w}_i^\top\mathbf{x}+b_i\}\qquad(i\in\{1,\dots,d\})$

Com. So, here we apply the nonlinearity in $\mathbb{R}^d$ (among some set members A_j) instead applying the nonlinearity in $\mathbb{R}$ (as with ridge functions).

T. (Goodfellow, 2013) Maxout networks with two maxout units that are applied to $2m$ linear functions are universal function approximators.

10 Feedforward Networks

10.1

D. (Feedforward Network) is a set of computational units arranged in a DAG (layer-wise processing)

$F=F^L\circ\cdots\circ F^1$

where each layer $l\in\{1,\dots,L\}$ is a composition of the following functions

$F^l\colon\mathbb{R}^{m_{l-1}}\to\mathbb{R}^{m_l}\qquad F^l=\sigma^l\circ\overline{F}^l$

where $\overline{F}^l\colon\mathbb{R}^{m_{l-1}}\to\mathbb{R}^{m_l}$ is the linear function in layer l

$\overline{F}^l(\mathbf{h}^{l-1})=\mathbf{W}^l\mathbf{h}^{l-1}+\mathbf{b},\quad\mathbf{W}^l\in\mathbb{R}^{m_l\times m_{l-1}},\quad\mathbf{b}\in\mathbb{R}^{m_l}$

and $\sigma^l\colon\mathbb{R}^{m_l}\to\mathbb{R}^{m_l}$ element-wise non-linearity at layer l.

Note that $\mathbf{h}^0:=\mathbf{x}$.

T. Feedforward Networks are invariant under permutations of units: The units within a hidden layer are interchangeable (along with their corresponding weights).

T. The units of feedforward networks are invariant along directions orthogonal to the weight vector $F[\mathbf{w},\mathbf{b}](\mathbf{x})=F[\mathbf{w},\mathbf{b}](\mathbf{x}+\delta\mathbf{x}):\forall\delta\mathbf{x}\perp\mathbf{w}$

T. A layer of width m can be embedded into a layer of width $m+1$ by adding "barren" (unused) units or by cloning existing units.

10.2 Empirical & Population Risk

In learning, we are interested in minimizing the population risk (test risk). We use the training risk (empirical risk) as an estimate of the population risk.

D. (Expected (training) Risk of a Function F)

$\ell[\theta](S)=\frac{1}{N}\sum_{i=1}^N\ell[\theta](x_i,y_i)$

D. (Population Risk/Test Risk)

$\mathcal{R}(\ell[\theta];p)=\mathbb{E}_{(\mathbf{x},y)\sim p}\ell(\mathbf{x},y)$

10.2.1 Linear Networks

Linear networks are composed of linear maps (with activation function $\phi=\mathrm{id}$). However, they do not gain representational power from depth because affine maps are closed under composition. They are primarily used for dimensionality reduction (contraction), c.f. Autoencoders.

10.3 Autoencoders

Goal: *Compress* the data into m-dim. ($m\leq d$) representation.

D. (Autoencoder) tries to learn the identity function.

$$\mathcal{R}(\theta)=\frac{1}{2n}\sum_{i=1}^n\|\mathbf{x}-F_\theta(\mathbf{x})\|_2^2=\mathbb{E}_{\mathbf{x}\sim p_{\mathrm{emp}}}[\ell(\mathbf{x},(H\circ G)(\mathbf{x}))]$$

Typically, a linear autoencoder can be broken into two parts G and H such that $F=H\circ G\approx\mathbf{x}\mapsto\mathbf{x}$, where $G\in\mathbb{R}^{m\times n}$ is the encoder and $H\in\mathbb{R}^{n\times m}$ is the decoder. The representation of G and H is not unique and by the Eckhart-Young theorem we can find the optimal representation (under the squared reconstruction error) by performing the SVD of the data matrix $\mathbf{X}$ (PCA).

$$\mathbf{X}=\begin{bmatrix}|&|&&|\\ \mathbf{x}_1&\mathbf{x}_2&\cdots&\mathbf{x}_k\\ |&|&&|\end{bmatrix}$$

is of the following form:

$$\mathbf{X}=\underset{n\times n}{\mathbf{U}}\underbrace{\mathrm{diag}^\dagger(\sigma_1,\dots,\sigma_{\min(n,k)})}_{=:\boldsymbol{\Sigma}\in\mathbb{R}^{n\times k}}\underset{k\times k}{\mathbf{V}^\top}.$$

And the matrices $\mathbf{U}$ and $\mathbf{V}$ are orthogonal; we have an orthogonal basis.

T. (Non-uniqueness) For any invertible matrix $\mathbf{A}$ we have

$$\underbrace{(\mathbf{U}_m\mathbf{A}^{-1})}_{\tilde{\mathbf{D}}}\underbrace{(\mathbf{A}\mathbf{U}_m^\top)}_{\tilde{\mathbf{C}}}=\mathbf{U}_m\mathbf{U}_m^\top$$

T. (Eckhart-Young) For $m\leq\min(n,k)$ and the objective

$$\underset{\hat{\mathbf{X}}\colon\ \mathrm{rank}(\hat{\mathbf{X}})=m}{\arg\min}\left\|\mathbf{X}-\hat{\mathbf{X}}\right\|_F^2=\mathbf{U}_m\mathrm{diag}(\sigma_1,\dots,\sigma_m)\mathbf{V}_m^\top$$

where the subscript m refers to the matrices of the SVD pruned to m columns.

C. This means that a linear auto-encoder with m hidden units cannot improve the SVD since $\mathrm{rank}(\mathbf{CD})\leq m$. However, the auto-encoder can achieve the result of the SVD.

10.4 Residual Networks

D. Residual Networks learn an incremental improvement: $F(x)=x+[\phi(Wx+b)]$. They utilize "skip connections to propagate inputs forward, allowing to mitigate the vanishing/exploding gradient problem. It allows us to increase depth even more. (Important paper: ResNet 2015). The skip connection can also be transformed with a matrix $\mathbf{A}$ to allow for more flexibility $F(x)=\mathbf{A}x+[\phi(Wx+b)]$.

10.4.1 Regularization Approaches

D. Weight Decay (L_2 Regularization): Adds a penalty to the loss function proportional to the square of the weights' magnitude to constrain complexity.

$$\ell(\theta)=\ell_{emp}(\theta)+\frac{\lambda}{2}\|\theta\|^2$$

D. Data Augmentation: Artificially expands the training set by creating modified versions of existing data (e.g., flipping, rotating) to teach invariance.

D. Noise Robustness: Injects random noise into inputs, weights, or gradients to make the model insensitive to small perturbations.

D. Semi-supervised Learning: Uses a small set of labeled data alongside unlabeled data. Unlabeled data helps learn the distribution $P(x)$ to regularize $P(y|x)$.

D. Dropout: Prevents co-adaptation by randomly deactivating neurons.

- **Training**: Each neuron is kept active with probability p (and dropped with $1-p$). This is mathematically applied using a mask vector $\mathbf{m}$:

$$y=\mathbf{m}\odot h,\quad\text{where }\mathbf{m}\sim\mathrm{Bernoulli}(p)$$

- **Inference (Weight Scaling)**: During testing, all neurons are active. To match the expected output magnitude from training (since $E[y_{train}]=p\cdot h$), we must scale the weights down:

$$W_{test}=p\cdot W_{train}$$

D. Early Stopping: Monitors validation error and stops training when performance degrades, effectively selecting parameters before overfitting occurs. Rigorous analysis shows that early stopping approximates a form of L_2 regularization.

D. Multi-task Learning: Trains on multiple tasks simultaneously using shared representations, introducing inductive bias that aids generalization.

D. Parameter Sharing: Constraints the model by forcing sets of parameters to be identical (e.g., filter weights in CNNs).

D. Ensembles: Combines predictions of multiple independent models to average out individual errors.

$$y_{final}=\frac{1}{k}\sum_{i=1}^kM_i(x)$$

D. Batch Normalization (BN) A technique to stabilize training by normalizing the inputs to a layer for each mini-batch. It addresses *Internal Covariate Shift* (the changing distribution of layer inputs during training).

1. **Normalization**: Calculate batch mean μ_B and variance σ_B^2 to normalize input x:

$$\hat{x}=\frac{x-\mu_B}{\sqrt{\sigma_B^2+\epsilon}}$$

2. **Scale and Shift**: The network learns parameters γ and β to restore representation power (allows the network to undo the normalization if needed):

$$y=\gamma\hat{x}+\beta$$

Benefits: Allows higher learning rates, reduces sensitivity to initialization, and acts as a weak regularizer.

10.4.2 Initialization Techniques

Proper initialization is crucial to prevent vanishing or exploding gradients at the start of training.

D. (LeCun Initialization)

- **Goal**: Keep the variance of the input and output unchanged for efficient backprop in linear networks.
- **Use Case**: SELU activation, or standard Sigmoid (older standard).
- **Variance**: $\mathrm{Var}(W)=\frac{1}{n_{in}}$

D. (Glorot Initialization (Xavier))

- **Goal**: Maintains the variance of activations during forward pass and gradients during backward pass. Assumes linear or Tanh/Sigmoid activations.
- **Use Case**: Tanh, Sigmoid, Softmax.
- **Variance**: $\mathrm{Var}(W)=\frac{2}{n_{in}+n_{out}}$

D. (He Initialization (Kaiming))

- **Goal**: accounts for the fact that ReLU zeroes out half of the inputs (reducing variance by half). It doubles the variance compared to LeCun to compensate.
- **Use Case**: ReLU, Leaky ReLU.
- **Variance**: $\mathrm{Var}(W)=\frac{2}{n_{in}}$

D. (Orthogonal Initialization)

- **Goal**: Initialize weight matrices as orthogonal matrices ($W^TW=I$). This ensures that the matrix eigenvalues have absolute value 1.
- **Use Case**: Recurrent Neural Networks (RNNs) and very deep networks. It prevents gradients from vanishing or exploding even after many matrix multiplications (time steps).

11 Backpropagation and Optimization

11.1 Automatic Differentiation

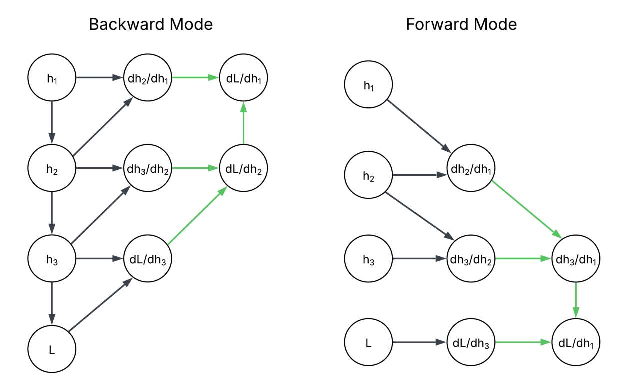
In neural networks we can efficiently compute the gradient of the loss w.r.t. the parameters $\nabla_\theta\mathcal{R}$ by exploiting the computational structure of the network.

Forward Mode AD (Forward Propagation of Tangents) Computes the derivative simultaneously with the function evaluation.

- **Mechanism**: Propagates perturbations $\dot{x}$ forward through the graph. For every node v_i, it computes the value v_i and the derivative $\dot{v}_i=\frac{\partial v_i}{\partial x}$.
- **DAG Flow**: Derivatives flow in the *same direction* as data (Input → Output). No need to store the full graph in memory.
- **Efficiency**: Efficient for functions $f:\mathbb{R}^n\to\mathbb{R}^m$ where $n\ll m$ (few inputs, many outputs).
- **Cost**: requires n passes to compute gradients for n parameters. In Deep Learning (millions of parameters), this is computationally prohibitive.
- $O(n)$ passes

Reverse Mode AD (Backpropagation) Computes derivatives after a full function evaluation by traversing the graph in reverse.

- **Mechanism**: First runs a *Forward Pass* to compute and store intermediate values (Primals). Then runs a *Backward Pass* to propagate adjoints $\bar{v}_i=\frac{\partial L}{\partial v_i}$.
- **DAG Flow**: Derivatives flow in the *opposite direction* (Output → Input). Requires storing the DAG and intermediate activations (memory intensive).
- **Efficiency**: Efficient for functions $f:\mathbb{R}^n\to\mathbb{R}^m$ where $n\gg m$ (many inputs, few outputs).
- **Relevance**: This is the standard in Deep Learning because we map millions of parameters (n) to a single scalar loss ($m=1$). It computes all gradients in a single backward pass.
- $O(m)$ passes



11.2 Jacobian Matrix

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

$$\mathbf{J}_F:=\begin{pmatrix}\text{---}(\nabla F_1)^\top\text{---}\\ \text{---}(\nabla F_2)^\top\text{---}\\ \vdots\\ \text{---}(\nabla F_m)^\top\text{---}\end{pmatrix}=\begin{pmatrix}\frac{\partial F_1}{\partial x_1}&\frac{\partial F_1}{\partial x_2}&\cdots&\frac{\partial F_1}{\partial x_n}\\ \frac{\partial F_2}{\partial x_1}&\frac{\partial F_2}{\partial x_2}&\cdots&\frac{\partial F_2}{\partial x_n}\\ \vdots&\vdots&\ddots&\vdots\\ \frac{\partial F_m}{\partial x_1}&\frac{\partial F_m}{\partial x_2}&\cdots&\frac{\partial F_m}{\partial x_n}\end{pmatrix}\in$$

T. (Jacobi Matrix Chain Rule)

$G:\mathbb{R}^n\to\mathbb{R}^q\quad H:\mathbb{R}^q\to\mathbb{R}^m$

$F:=H\circ G$

Chain-rule for a single component-function $F_i:\mathbb{R}^n\to\mathbb{R}$ of F:

$$\left.\frac{\partial F_i}{\partial x_j}\right|_{\mathbf{x}=\mathbf{x}_0}=\left.\frac{\partial(H\circ G)_i}{\partial x_j}\right|_{\mathbf{x}=\mathbf{x}_0}=\sum_{k=1}^q\left.\frac{\partial H_i}{\partial z_k}\right|_{\mathbf{z}=G(\mathbf{x}_0)}\left.\frac{\partial G_k}{\partial x_j}\right|_{\mathbf{x}=\mathbf{x}_0}$$

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. Altough, 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 (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.**Enough 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 (NAG)) A "look-ahead" version of GD. It computes the gradient at the *approximate future position* of the parameters rather than the current position.

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

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.

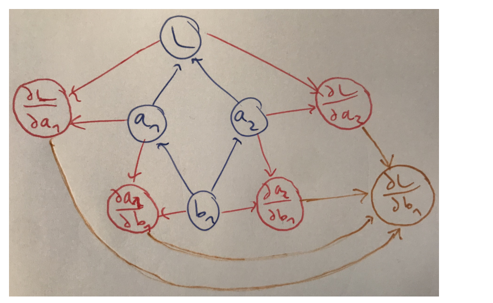
$$E[g^2]_t = \beta E[g^2]_{t-1} + (1-\beta)(\nabla J(\theta_t))^2$$
$$\theta_{t+1} = \theta_t - \frac{\alpha}{\sqrt{E[g^2]_t + \epsilon}}\nabla J(\theta_t)$$

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.

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

11.4 — Backpropagation Graphs

The approach a backpropagation graph for a loss L from a FF network graph is as follows:
1. (RED) Starting at the loss L backward: for each node n and for each of its outputs o which has a path to the loss, add the node $\frac{\partial o}{\partial n}$ (at the height of node n). Connect the output o and that node n to that newly created node.
2. (BROWN) Starting at the loss backward: for each node n create a node $\frac{\partial L}{\partial n}$ (if it doesn't exist already) and connect each previously created partial node ($\frac{\partial o}{\partial n}$) to it, and the previously crated (in this step) $\frac{\partial L}{\partial o}$'s too.



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 $T: \mathcal F \to \mathcal F'$.

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 $f,h: \mathbb R\to\mathbb R$, 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 = signal, and h = fast decaying kernel function.

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

T. (Convolutions are commutative and associative)

T. (Convolutions 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 $f,h: \mathbb Z\to\mathbb R$, 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 $f,h: \mathbb R^d\to\mathbb R$ we have

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

D. (Discrete Cross-Correlation) Let $f,h: \mathbb Z\to\mathbb R$, then

$$(h\star f)[u] := \sum_{t=-\infty}^\infty h[t]f[u+t] = \sum_{t=-\infty}^\infty h[-t]f[u-t]$$
$$= (\bar h * f)[u] = (f*\bar h)[u] \qquad\text{where } \bar h(t) = h(-t).$$

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 ofsetting it by one more very time (gives a band matrix (special case of Toeplitz matrix)). Then the convolution is just a matrix-vector product.

12.4 — Border Handling

There are different options to do this
- **D. (Padding of p)** Means we extend the image (or each dimension) by p on both sides (so $+2p$) and just fill in a constant there (e.g., zero).
- **D. (Same Padding)** Padding with zeros = *same padding* ("same" constant, i.e., 0, and we'll get a tensor of the "same" dimensions)
- **D. (Valid Padding)** Only retain values from windows that are fully-contained within the support of the signal f (see 2D example below) = *valid padding*

12.5 — Backpropagation for Convolutions

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

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

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

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

Due to *weight-sharing*, the kernel weight 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

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

$$\begin{aligned}\frac{\partial\mathcal R}{\partial w_i} &= \sum_k\frac{\partial\mathcal R}{\partial y_k^{(l)}}\frac{\partial y_k^{(l)}}{\partial w_i} = \sum_k\frac{\partial\mathcal R}{\partial y_k^{(l)}}\frac{\partial}{\partial w_i}\left[\mathbf y^{(l)} * \mathbf w\right]_k\\ &= \sum_k\frac{\partial\mathcal R}{\partial y_k^{(l)}}\frac{\partial}{\partial w_i}\left[\sum_{o=-p}^{p} y_{k-o}^{(l-1)}w_o\right] = \sum_k\frac{\partial\mathcal R}{\partial y_k^{(l)}}y_{k-i}^{(l-1)}\\ &= \sum_k\frac{\partial\mathcal R}{\partial y_k^{(l)}}y_{-(k-i)}^{(l-1)} = \sum_k\frac{\partial\mathcal R}{\partial y_k^{(l)}}\mathrm{rot180}(y^{(l-1)})_{k-i}\\ &= \left(\frac{\partial\mathcal R}{\partial\mathbf y^{(l)}} * \mathrm{rot180}(y^{(l-1)})\right)_i\end{aligned}$$

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

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

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

D. (Rotation180) $\forall i$: $\mathrm{rot180}(\mathbf x)_i = \mathbf x_{(-i)}$.

12.6 — Pooling

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

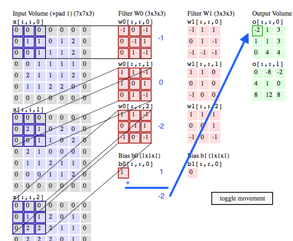
D. (Max-Pooling)
- 1D: $x_i^{\max} = \max\{x_{i+k} \mid 0 \le k < r\}$
- 2D: $x_{ij}^{\max} = \max\{x_{i+k,j+l} \mid 0\le k,l<r\}$

12.7 — Sub-Sampling (aka "Strides")

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

12.8 — Channels

Ex. Here we have
- an input signal that is 2D with 3 channels (7x7x3) (image x 3 channels)
- and we want to learn two filters $W0$ and $W1$, which each process the 3 channels, and sum the results of the convolutions across each channel leading to a tensor of size 3x3x2 (convolution result x num convolutions)



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

12.9 — CNNs in Computer Vision

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

12.9.1 — Classic CNN Architectures

D. (LeNet-5 (1998)) The pioneering CNN for handwritten digit recognition (MNIST).
- **Structure**: 2 Convolutional layers (with Average Pooling) followed by 3 Fully Connected layers.
- **Key Features**: Introduced the concepts of local receptive fields, shared weights, and spatial subsampling. Used Sigmoid/Tanh activations (pre-ReLU).

D. (AlexNet (2012)) The breakthrough model that popularized Deep Learning on ImageNet.
- **Structure**: Deeper than LeNet (5 Conv layers, 3 FC layers). Used large filters initially (11 × 11).
- **Innovations**: First large-scale use of **ReLU** (to solve vanishing gradients), **Dropout** (for regularization), and **Data Augmentation**. Trained on GPUs.

D. (VGG Network (2014)) Focused on the effect of network depth using a uniform architecture.
- **Philosophy**: Replace large filters (e.g., 5 × 5, 7 × 7) with stacks of small 3 × 3 **filters**.
- **Reasoning**: Two stacked 3 × 3 layers have the same receptive field as a 5 × 5 layer but with fewer parameters and more non-linearities (ReLU between layers).

D. (Inception Network (GoogLeNet, 2014)) Focused on computational efficiency and network "width".
- **Inception Module**: Instead of choosing a filter size, it performs 1 × 1, 3 × 3, and 5 × 5 convolutions (and pooling) *in parallel* and concatenates the outputs.
- **1 × 1 Convolutions**: Used as "Bottleneck layers" to reduce dimensionality (depth) before expensive operations, significantly reducing computational cost.

D. (U-Net (2015)) Designed for Biomedical Image **Segmentation** (pixel-wise classification).
- **Structure**: Symmetrical Encoder-Decoder architecture (U-shape).
- **Encoder**: Contracting path (Convs + Max Pooling) to capture context.
- **Decoder**: Expansive path (Up-Convs) to enable precise localization.
- **Skip Connections**: Concatenates high-resolution features from the encoder directly to the decoder to recover spatial details lost during downsampling.

12.9.2 — Convolutions in Sequences, NLP & Audio

D. (1D Convolutions (Temporal ConvNets)) Unlike 2D CNNs for images, sequence modeling uses 1D filters that slide over the time axis.
- **Input**: Tensor of shape (*Batch*, *Length*, *Channels*). In NLP, "Channels" are the dimensions of the Word Embeddings.
- **Function**: Captures local temporal dependencies (like n-grams in text) effectively.
- **Advantage**: Highly parallelizable (unlike RNNs which are sequential) and computationally efficient.

D. (Embeddings & NLP) CNNs are often applied on top of pre-trained word embeddings (e.g., Word2Vec, GloVe).
- A sentence is represented as a matrix (Length × Embedding Dim).
- Filters of different widths (e.g., covering 2, 3, or 4 words) act as feature detectors for phrases or specific linguistic patterns (e.g., "very good", "not bad") regardless of their position in the sentence.

D. (Dilated Convolutions) To handle long sequences without losing resolution (pooling), *dilation* introduces gaps between kernel elements.
- **Receptive Field**: Grows exponentially with the dilation factor d $(1,2,4,8\dots)$, allowing the network to capture long-range dependencies with few layers.

D. (WaveNet (2016)) A deep generative model for raw audio waveforms.
- **Causal Convolutions**: Strict ordering ensures the prediction at time t only depends on samples $x_{<t}$ (cannot see the future).
- **Dilated Causal Convolutions**: Stacks layers with increasing dilation factors. This allows the output neuron to have a receptive field of thousands of timesteps (milliseconds of audio) to generate realistic high-fidelity sound structure.
- **Skip Connections**: Uses residual and parameterized skip connections to speed up convergence and allow training of very deep networks.

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

Ex. input image $m\times n\times c$ (c = number of channels)
K convolution kernels: $p\times q$ (valid padding and stride 1)
output dimensions: $(m-p+1)\times(n-q+1)\times K$

#parameters CNN: $K(pqc+1)$

#parameters of fully-conn. NN with same number of 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 (RNNs)

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 $x_1,\dots,x_T$:

$$z_t = \phi(Uz_{t-1} + Vx_t)$$
$$\hat y_t = \psi(Wz_t)$$

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

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

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

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

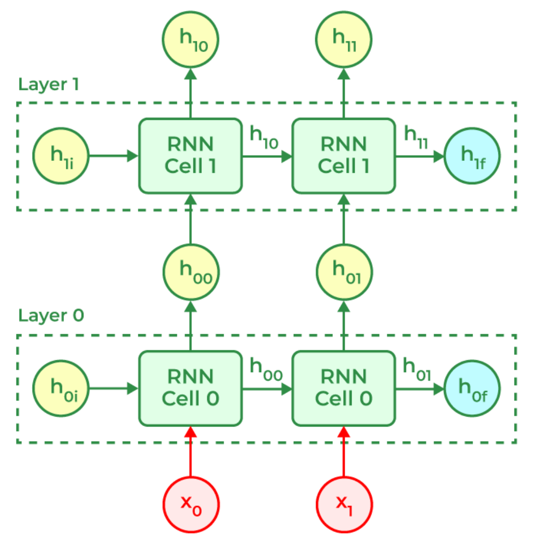
Gradient Problems: The gradient involves repeated multiplication of the recurrent weight matrix U (specifically its Jacobian).
- **Exploding Gradient**: If largest singular value $\sigma_{max}(U) > 1$.
- **Vanishing Gradient**: If $\sigma_{max}(U) < 1$. This makes learning long-term dependencies difficult.

13.1.1 — Structural Variants

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

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

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



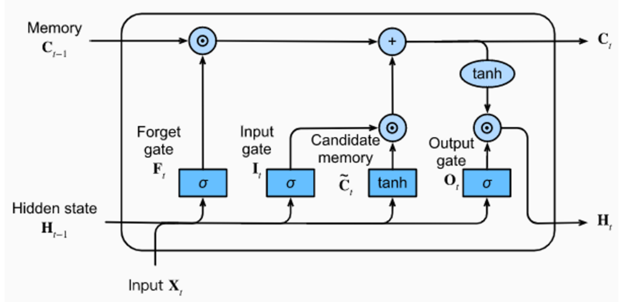
13.1.2 — Gated Memory

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

D. (Long Short-Term Memory (LSTM)): Maintains a separate cell state C_t controlled by three gates.

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

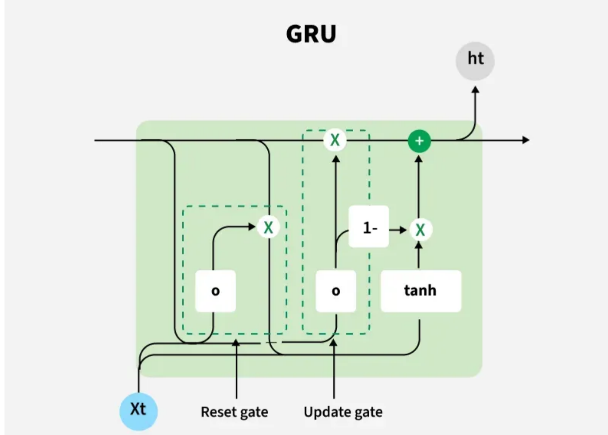
where $\tilde x^t = [x_t, z_{t-1}]$.



D. (Gated Recurrent Unit (GRU)): Simplifies LSTM by merging Cell/Hidden states and Forget/Input gates.

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

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



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 $\xi^t = P^{-1}z^t$:

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

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

13.1.4 — Sequence Learning

Generative Modeling: Decomposes joint probability of a sequence:

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

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

14 Optimization

14.1 — Objectives as Expectations

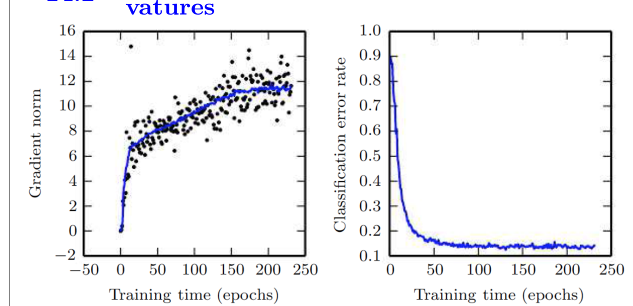
$$\nabla_\theta\mathcal R(D) = \mathbb E_{\mathcal S_N\sim p_D}[\nabla_\theta\mathcal R(\mathcal S_N)] = \mathbb E\left[\frac1N\sum_{i=1}^N\nabla_\theta\mathcal R(\theta;\{\mathbf x[i],y[i]\})\right]$$

T. We have the following chain of inclusions for functions over a *closed* and *bounded* (i.e., compact) subset of the real line.
Continously differentiable ⊆ Lipschitz continuous ⊆ (Uniformly) continous

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*.

14.2 — Optimization Challenges in NNs: Curvatures



14.2.1 — Convergence Rates

Under certain conditions SGD converges to the optimum:
- If we have a convex, or strongly convex objective,
- and if we have Lipschitz continous gradients,
- and a deaying learning rate, s.t.

$$\underbrace{\sum_{t=1}^\infty\eta_t = \infty}_{\text{we get far enough}},\qquad \underbrace{\sum_{t=1}^\infty\eta_t^2<\infty}_{\text{our steps get always smaller}}$$

typically $\eta_t = Ct^{-\alpha}$, $\frac12<\alpha<1$ (c.f. harmonic series.)
- or we use iterate (Polyak) averaging (once we start jumping around, we average the solutions over time).

Then, we can get the following convergence rates:
- strongly-convex case: can achieve a $\mathcal O(1/t)$ suboptimality rate (only polynomial convergence)
- non-strongly convex case: $\mathcal O(1/\sqrt t)$ suboptimality rate (even worse than polynomial convergence)

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 Geometric Deep Learning

15.1 — 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** π:
- **Cauchy Notation**: $\pi = \begin{pmatrix}1 & 2 & \cdots & M\\ \pi(1) & \pi(2) & \cdots & \pi(M)\end{pmatrix}$
- **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 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 a shared 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.

Input $\xrightarrow{\text{MLP}}$ Features $\xrightarrow{\text{Max Pool}}$ Global Feature $\xrightarrow{\text{MLP}}$ Output

15.2 — Graph Convolutional Networks (GCN)

Operates on Graph $G = (\mathcal V,\mathcal E)$ 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 $D = \mathrm{diag}(d_1,\dots,d_M)$ 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 $\tilde A = A + I$ and corresponding degree matrix $\tilde D_{mm} = \sum_n\tilde A_{nm} = d_m + 1$.

$$\bar A = \tilde D^{-\frac12}\tilde A\tilde D^{-\frac12} = \tilde D^{-\frac12}(A+I)\tilde D^{-\frac12}$$

D. Layer Update: Combines neighborhood aggregation ($\bar AX$) and feature transformation (W).

$$X' = \sigma(\bar AXW)$$

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.

15.3 — Spectral Graph Theory

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

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

$$L = D - A \quad\text{or Normalized: } \tilde L = I - D^{-\frac12}AD^{-\frac12}$$

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

$$x\star y = U((U^Tx)\odot(U^Ty))$$

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

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

15.4 — Attention GNNs (GAT)

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

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

16 Theory

16.1 — 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 $m\to\infty$ (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:

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

16.2 — Bayesian DNNs

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

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

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

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

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

This mimics a physical system with friction and thermal noise.

16.3 — Gaussian Processes (GPs)

D. Infinite Width Equivalence (Neal's Theorem): A single-hidden-layer neural network with infinite width ($m\to\infty$) 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}.

16.4 — Statistical Learning Theory

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

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

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).

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

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.

17 Chain-Rule and Jacobians for Tensors

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

D. (Tensor Multiplication)

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

$$\underset{r_1\times\cdots\times r_a\times s_1\times\cdots\times s_c}{\mathbf T} = \underset{r_1\times\cdots\times r_a\times s_1\times\cdots\times s_b}{\mathbf P}\times_{s_1,\dots,s_b}\underset{s_1\times\cdots\times s_b\times t_1\times\cdots}{\mathbf Q}$$

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 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:

$$T_{\underbrace{i_1,\dots,i_a}_{\text{"i"}},\underbrace{k_1,\dots,k_c}_{\text{"k"}}} := \sum_{\underbrace{j_1,\dots,j_b}_{\text{"j"}}}P_{\underbrace{i_1,\dots,i_a}_{\text{"i"}},\underbrace{j_1,\dots,j_b}_{\text{"j"}}}Q_{\underbrace{j_1,\dots,j_b}_{\text{"j"}},\underbrace{k_1,\dots,k_c}_{\text{"k"}}}$$

T. (Tensor Chain Rule)

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

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

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

18 Generative Models

18.1 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.

18.2 Normalizing Flows

Learns a bijective mapping $f : \mathcal{Z} \to \mathcal{X}$ 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 $x_{d+1:D}$:

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

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

18.3 Generative 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.

18.4 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:

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