1 Linear Algebra

Vector Properties

Linear independence — Linear combination $Au = u_1 a_1 + ... + u_n a_n = \sum_{i=1}^{n} u_i a_i$ only has unique solutions for u (*unique representation theorem*), if $Au = 0$ then $u = 0$, and A is full rank

Unit vector — $u = \frac{\tilde{u}}{\|\tilde{u}\|}$, therefore $\|u\|^2 = 1$

Inner product —
$u \cdot v = u^\top v = \sum_{i=1}^{n} u_i v_i = cos(\varphi)\ \|u\|\ \|v\|$
resp.
$\langle u, v \rangle_W = u^\top W v = \sum_i u_i v_i w_i$ where W is a diagonal matrix with $w_i > 0$ — Properties:

- $u \cdot v = v \cdot u$
- $(u+v) \cdot w = u \cdot w + v \cdot w$
- $(\alpha u) \cdot v = \alpha (u \cdot v)$
- Positive definite: $u \cdot u \geq 0$
- $u \cdot u = 0 \Leftrightarrow u = 0_v$

Norm — $\|u\| = \sqrt{u \cdot u}$
resp.
$\|u\|_W^2 = u^\top W u = \sum_i u_i^2 w_i$ where W is a diagonal matrix with $w_i > 0$ — Properties:

- $\|u+v\| = \|u\| + \|v\|$
- $\|\alpha u\| = |\alpha| \|u\|$

Distance and angle between two vectors —

- Distance: $d = \|u - v\| = \sqrt{(u_1 - v_1)^2 + ... + (u_n - v_n)^2}$
- Angle: $cos(\varphi) = \frac{u \cdot v}{\|u\|\ \|v\|}$

Cauchy Schwarz inequality — $\|u \cdot v\| \leq \|u\|\ \|v\|$ with equality iff $\varphi = 0$ i.e. $u = \alpha v$ or if u or $v = 0_v$
Proof:

- First direction of proof: If $u = \alpha v$ or u or $v = 0_v$, we can show that the equality holds
- Second direction of proof: If $u \neq \alpha v$ or u and $v \neq 0_v$, we can show that the inequality cannot hold:
  - u can be decomposed into $u_v + u_{v^\perp}$
  - Then, we have $\|u \cdot v\| = \|(u_v + u_{v^\perp}) \cdot v\| = \|u_v\| \cdot \|v\|$
  - Based on Pythagorean theorem, we know that $\|u\|^2 > \|u_v\|^2$
  - Then, we have $\|u \cdot v\| = \|u_v\| \cdot \|v\| < \|u\| \cdot \|v\|$

Triangle inequality — $\|u+v\| \leq \|u\| + \|v\|$ with equality iff $\varphi = 0$ i.e. $u = \alpha v$ or if u or $v = 0_v$

Other inequalities —

- $\|n^k\| \leq \|n\|^k$
- $|\sum_i n_i| \leq \sum_i |n_i|$

Orthogonal vectors — Properties:

- $u \cdot v = 0$
- $\|u+v\|^2 = \|u\|^2 + \|v\|^2$
- *Pythagorean theorem*: $\|u - v\|^2 = \|u\|^2 + \|v\|^2$
- Non-zero pairwise orthogonal vectors u_n and u_m are linearly independent
  Proof:
  - Let $\sum_n \alpha_n u_n = 0_v$
  - Then, $= 0_v \cdot u_m = (\sum_n \alpha_n u_n) \cdot u_m = \sum_n \alpha_n (u_n) \cdot u_m) = \alpha_m \|u_m\|^2$
  - Then, $\alpha_m = 0$ for all m, meaning that all u_m are linearly independent

Orthonormal vectors — Vectors are orthonormal iff $\|u\| = \|v\| = 1$ and $u \cdot v = 0$

Projection — Projection of $v \in V$ onto $s \in S$ given by: $v_S = \frac{v \cdot s}{\|s\|^2} s = (v \cdot s) s$ if s is a unit vector

Vector Spaces

Vector space V — Properties:

- Additive closure: If $u, v \in V$ then $u + v \in V$
- Scalar closure: If $u \in V$ then $\alpha u \in V$
- $\exists 0_v$ such that $u + 0_v = u$
- $\exists -u$ such that $u + -u = 0_v$
- $u + v = v + u$
- $(u+v) + w = u + (v+w)$
- $\alpha(\beta u) = (\alpha\beta) u$
- $\alpha(u+v) = \alpha u + \alpha v$
- $u(\alpha + \beta) = \alpha u + \beta u$

Subspace S — Properties: S is a subspace of V iff:

- $0_v \in S$
- Additive closure
- Scalar closure

Proof:

- If S is a subspace of V subspace properties immediately follow
- If subspace properties are satisfied for S, S must be a subspace of V because operations are inherited (for addition, multiplication) resp. can be derived from subspace properties (for $0_V, -v$)

Affine subspace L — TBA

Invariant subspace H — H is an invariant subspace of S spanned by S if $Sh \in H$ for all $h \in H$ — Properties:

- S has an eigenvector in H
- If S is symmetric, $H^\perp$ is also an invariant subspace of S

Orthogonal complement $S^\perp$ — Subspace, composed of set of vectors that are orthogonal to S — Properties:

- The intersection of S and $S^\perp$ is $\{0_v\}$
- $\dim(S) + \dim(S^\perp) = \dim(V)$

Span — Span of $\{s_i\}_{i=1}^n$ is the set of all vectors that can be expressed as a linear combination of $\{s_i\}_{i=1}^n$. $\sum_{i=1}^n u_i s_i$
Span of matrix A is the span of its column vectors. $Au = u_1 a_1 + ... + u_n a_n = \sum_{i=1}^n u_i a_i$
A span is a subspace, since for a linear combination, we can derive additive closure and scalar closure.

(Orthonormal) basis — Unique set of all (orthonormal) vectors that are linearly independent and span the whole of a subspace.

- *Orthonormal representation theorem*: Any vector $x \in S$ can be expressed in terms of orthonormal basis: $x = \sum_i (x \cdot s_i) s_i$
- *Parseval's theorem*: Extension of orthonormal representation theorem: $x \cdot y = \sum_i (x \cdot s_i)(y \cdot s_i)$
- *Gram Schmidt orthonormalization*: Procedure to generate orthonormal basis $\{s_i\}_{i=1}^n$ from linearly independent vectors $\{x^{(i)}\}_{i=1}^n$:
  - $\tilde{s_1} = x_1$
  - $\tilde{s_k} = x_k - \sum_{i=1}^{k-1} (x_k \cdot s_1) s_1$ for $k > 1$
  - $s_i = \frac{\tilde{s_i}}{\|\tilde{s_i}\|}$

Dimension d — Number of vectors in basis of S. Each vector has d elements.

Orthogonal vectors in spaces —

- Let S be spanned by orthonormal $s_1, s_2, ...$ and $v \in V$
- *Orthogonal decomposition theorem*: $v = v_S + v_{S^\perp}$ where $v \in V$, $v_S \in S$ and $v_{S^\perp} \in S^\perp$
- *Orthogonality principle*
  - v_S is the projection of $v \in V$ to S iff $(v - v_S) \cdot s_i = 0$
  - This can be rewritten to linear equation system $v \cdot s_i = v_S \cdot s_i = \sum_k \alpha_k (s_k \cdot s_i)$ since $v_S = \sum_k \alpha_k s_k$
- $v_{S^\perp} = v - v_S = v - \sum_k (v \cdot s_k) s_k$
- *Approximation in a subspace theorem*:
  - Unique best representation of v in S is given by projection of v to S: $\|v - s'\| \geq \|v - v_S\|$ for some arbitrary $s' \in S$
  - Any subset U of S is closest to v iff it is closest to v_S
    Proof:
    - $\text{argmin}_u \|v - u\| = \text{argmin}_u \|v - u\|^2 = \text{argmin}_u \|v_S + v_{S^\perp} - u\|^2 = \text{argmin}_u \|v_S - u\|^2 + \|v_{S^\perp}\|^2$ given Pythagorean theorem $= \text{argmin}_u \|v_S - u\|^2$

Linear Equations

Let $Xb = y$ where $X \in \mathbb{R}^{n \times m}$, $b \in \mathbb{R}^m$, $y \in \mathbb{R}^n$ and b is unknown

- Number of distinct equations = Number of linearly independent rows in $[X|b]$ = $\text{rank}([X|b]) \leq \min(n, m+1)$
- Number of LHS solutions should = Number of RHS solutions = $\text{rank}(X) \leq \min(n, m)$

Solutions:

- If $\text{rank}(X) < \text{rank}([X|b])$, system is inconsistent (no solution)
- If $\text{rank}(X) = \text{rank}([X|b]) < m$, system is singular (infinitely many solutions) and underdetermined because we have fewer distinct equations than unknowns
- If $\text{rank}(X) = \text{rank}([X|b]) = m = n$, system is non-singular (exactly one solution) and exactly determined
- If $\text{rank}(X) = \text{rank}([X|b]) = m < n$, system is non-singular and overdetermined

General Matrix Properties

Matrices —

- $A \in \mathbb{R}^{n \times m}$ with elements A_{ij}, rows $i = 1, ..., n$, columns $j = 1, ..., m$
- Transpose $A^\top$
- Identity matrix I with 1 on diagonal, 0 elsewhere
- Scalar matrix K with k on diagonal, 0 elsewhere

Operations —

- Element-wise addition: Returns matrix of same size
- Element-wise scalar multiplication: Returns matrix of same size
- Matrix multiplication:
  - $A^{n \times p} B^{p \times m} = C^{n \times m}$
    - $r_v \times c_v = s$
    - $c_v \times r_v = M$
    - $M \times c_v = c_v$
    - $r_v \times M = r_v$
    - $M \times M = M$
  - Element in C is sum-product of row in A and column in B: $C_{ij} = A^{(i)} \cdot B^{(j)}$
  - Column vector in C is a linear combination of the columns in A: $C^{(j)} = AB^{(j)} = \sum_p A^{(j=p)} b_p^{(j)}$
  - Row vector in C is a linear combination of the rows in B: $C^{(i)} = A^{(i)} B = \sum_p a_p^{(i)} B^{(i=p)}$
  - $C = A[B^{(j=1)}|...|B^{(j=m)}]$
  - $C = [A^{(i=1)}|...|A^{(i=n)}]^\top B = [A^{(i=1)} B|...|A^{(i=n)} B]^\top$

Impliations —

- $Ae_k = A^{(j=k)}$ and $e_k^\top A = A^{(i=k)}$ where e_k = 1 on k^{th} element and 0 everywhere else
- Matrix form:
  - In following $^{(j)}$ refers to column vector and $^{(i)}$ to row vector, however written as column vector
  - $u \cdot v = u^\top v = \sum_i u_i v_i = c$
  - $uv^\top = C$ with $u_i v_j = C_{ij}$
  - $Au = \sum_{j=i} A^{(j)} u_i = c$ with $A^{(i)} \cdot u = A^{(i)\top} u = c_i$
  - $u^\top A = \sum_{j=i} A^{(i)\top} u_j = c^\top$ with $u \cdot A^{(j)} = u^\top A^{(j)} = c_j$
  - $AB = \sum_{j=i} A^{(j)} B^{(i)\top} = C$ with $A^{(i)} \cdot B^{(j)} = A^{(i)\top} B^{(j)} = C_{ij}$
- Moving between instance-level → data-level:
  - $x^{(i)} y = a \rightarrow X^\top y = a$ where X consists of rows $x^{(i)}$
  - $x^{(i)} x^{(i)\top} = A \rightarrow X^\top X = A$ where X consists of rows $x^{(i)}$
  - $x^{(i)} \cdot \beta = y_i \rightarrow X\beta = y$ where X consists of rows $x^{(i)}$

Properties —

- $(A+B)^\top = A^\top + B^\top$
- $(\alpha A)^\top = \alpha A^\top$
- $(AB)^\top = B^\top A^\top$
- $(A+B)+C = A+(B+C)$
- $A+B = B+A$
- $\alpha(A+B) = \alpha A + \alpha B$
- $(\alpha+\beta)A = \alpha A + \beta A$
- $(\alpha\beta)A = \alpha(\beta A)$
- $(A+B)x = Ax + Bx = Cx$
- $(AB)x = A(Bx) = Cx$
- $A = 0.5(A + A^\top) + 0.5(A - A^\top) = B + C$ where B is symmetric, but not C
- $A = AI = IA$
- $Ak = AK = KA$
- $\text{rank}(AB) = \min(\text{rank}(A), \text{rank}(B))$
- $A^\top A$ satisfies:
  - Symmetric
  - Psd
  - Has rank m iff it is pd
  - Invertible iff it has rank m and it is pd
  - $\text{rank}(A^\top A) = \text{rank}(A) = \text{rank}(A^\top)$
  - $\text{rank}(A^\top A) = \text{rank}([A^\top A | A^\top x])$

Matrix terminology —

- Kernel $\text{null}(X)$ contains set of vectors b such that linear map $Xb = 0$
- Nullity = $\dim(\text{null}(X))$
- Image $\text{range}(X)$ contains set of vectors b generated by linear map Xb resp. is space spanned by columns of (X)
- Row space is space spanned by rows of (X)
- Column rank = $\dim(\text{colspace}(X))$ = number of linearly independent columns, row rank = $\dim(\text{rowspace}(X))$ = number of linearly independent rows
- Rank = column rank = row rank = $\dim(\text{range}(X)) = \dim(\text{range}(X^\top)) \leq min(n, m)$
- *Rank nullity theorem*: $\text{Rank}(X) + \text{nullity}(X) = m$

Matrices as linear maps — X maps b from $\mathbb{R}^m$ to $\mathbb{R}^n$: $Xb = y$ with $X \in \mathbb{R}^{n \times m}$, $b \in \mathbb{R}^m$, $y \in \mathbb{R}^n$

- Injective: $Xb = y$ has at most one solution, happens iff columns of X are linearly independent ($\text{rank}(X) = m \leq n$)
- Surjective: $Xb = y$ has at least one solution, happens iff rows of X are linearly independent ($\text{rank}(X) = n \leq m$)
- Bijective: Mapping is both injective and surjective, i.e. $m = n$

Projection matrices —
Generally:

- Projection matrix satisfies $P = P^2$
- Proof:
  - Let S be spanned by $\{y_i\}_{i=1}^n$, which are column vectors of the matrix $A \in \mathbb{R}^{m \times n}$
  - Then, Ac are linear combinations of $\{y_i\}_{i=1}^n$
  - A vector $(x - Ac)$ is orthogonal to the columnspace of A, if: $\text{columnspace}(A) \cdot (x - Ac) = A^\top (x - Ac) = A^\top x - A^\top Ac = 0$
  - Then, $c = (A^\top A)^{-1} A^\top x$
  - With this definition of c, we have $A(A^\top A)^{-1} A^\top$ as the projection matrix P
  - $P^2 = (A(A^\top A)^{-1} A^\top)(A(A^\top A)^{-1} A^\top) = A(A^\top A)^{-1} A^\top = P$

Via orthonormal basis: Let S be spanned by orthonormal $\{b_i\}_{i=1}^n$, which are column vectors of the matrix $B \in \mathbb{R}^{m \times n}$

- Projection of x onto S is given by: $u = \sum_i (x \cdot b_i) b_i = \sum_i b_i b_i^\top x = BB^\top x = Cx$
- Projection of x onto $S^\perp$ is given by: $x - u = Ix - Cx$

Via SVD: Let S be spanned by $\{y_i\}_{i=1}^n$, which are column vectors of the matrix $A \in \mathbb{R}^{m \times n}$

- Projection of x onto S is given by: $s = AA^\# x$ since $AA^\#$ is a projection matrix due to $AA^\# = (AA^\#)^2$
- $s = U_+ U_+^\top x$ where U is obtained from SVD of A
- $\sum_{l=1}^m |u_k \cdot y_l| = \sigma_k^2$
  Proof: $\sum_{l=1}^m |u_k \cdot y_l| = \|u_k^\top A\| = u_k^\top AA^\top u_k = u_k^\top USV^\top VS^\top U^\top u_k = e_k^\top SS^\top e_k = \sigma_k^2$

Via basis: Let S be spanned by $\{b_i\}_{i=1}^n$, which are column vectors of the matrix $B \in \mathbb{R}^{m \times n}$ TBA

Square Matrix Properties

Square matrix terminology —

- Diagonal matrix:
  - Def: Has $\{d_i\}_{i=1}^n$ on diagonal and 0 everywhere else
  - For diagonal matrices: $DD^\top = D^\top D$
- Inverse matrix:
  - Def: $A^{-1} A = I$
  - Is unique
  - For diagonal matrices: A^{-1} can be calculated by inverting all diagonal elements
- Symmetric (Hermitian) matrix:
  - $A^\top = A$
  - Properties:

  - $(x+A^{-1}b)^\top A(x+A^{-1}b)-b^\top A^{-1}b = x^\top Ax+2x^\top b$
  - If A and B are symmetric, $A+B$ is also symmetric
- Orthogonal (unitary) matrix:
  - Def: $A^\top = A^{-1}$
  - $AA^\top = A^\top A = I$
  - Rows and columns are orthonormal
  - $\|Ax\| = \|x\|$
  - $(Ax)\cdot(Ay) = x\cdot y$
- Involution matrix: $A^{-1} = A$
- Determinant:
  - Function which maps A to a scalar
  - Properties:
    - $\det(I) = 1$
    - $\det(AB) = \det(A)\det(B)$
    - $\det(A^\top) = \det(A)$
    - $\det(A^{-1}) = (\det(A))^{-1}$
    - $\det(\alpha A) = \alpha^2\det(A)$
    - Determinant of diagonal matrix is product of diagonal elements

Invertible matrix theorem — Following statements are equivalent for square matrix $A\in\mathbb{R}^{n\times n}$:
- A is invertible
- Only solution to $Ax=0$ is $x=0_v$
  Proof:
  - $A^{-1}Ax=0\Rightarrow Ix=0\Rightarrow x=0_v$
- A is non-singular
- Columns (and rows) of A are linearly independent
- $\text{rank}(A)=n$
- $\det(A)=0$

Inversely, if A is not invertible, the columns and rows are not linearly independent, etc.

Matrix inversion lemma —
- Let $B\in\mathbb{R}^{n\times n}$, $D\in\mathbb{R}^{m\times m}$, $C\in\mathbb{R}^{n\times m}$. Then, $A=B^{-1}+CD^{-1}C^\top$ is invertible: $A^{-1}=B-BC(D+C^\top BC)^{-1}C^\top B$
- Let $v\in\mathbb{R}^n$. Then, $(\alpha I+vv^\top)^{-1}v=(\alpha+\|v\|^2)^{-1}v = v^\top(\alpha I+vv^\top)^{-1}=v^\top(\alpha+\|v\|^2)^{-1}$

Quadratic form — Quadratic form of square matrix M: $x^\top Mx$. Can be expressed as quadratic form of a symmetric matrix A: $x^\top Ax$ where $A=0.5\times(M+M^\top)+0.5\times(M-M^\top)$.

Eigenvectors and eigenvalues —
- q is an eigenvector of A associated with an eigenvalue λ if it remains on the same line after transformation by a linear map: $Aq=\lambda q$
- Let $A\in\mathbb{R}^{n\times n}$. A can have between $1-n$ eigenvalues, each with multiple eigenvectors. Eigenvectors for distinct eigenvalues are linearly independent
- *Spectral radius*: $\rho(A)$ is the largest eigenvalue of A
- If there exists a non-trivial solution for q, $(A-\lambda I)$ is not invertible and characteristic polynomial $\det(A-\lambda I)=0$
- *Eigendecomposition resp. diagonalization*: $A=Q\Lambda Q^{-1}$ where Q is a matrix with the eigenvectors as columns and Λ is a diagonal matrix with the eigenvalues on the diagonal
- $\det(A)=\det(Q\Lambda Q^{-1})=\prod_{i=1}^n\lambda_i$
- *Symmetric eigendecomposition resp. unitary diagonalization*: For symmetric A: $A=Q\Lambda Q^\top$ where Q is an orthogonal matrix with the eigenvectors as columns and Λ is a diagonal matrix with the eigenvalues on the diagonal
- *Spectral theorem*: Square matrix A is symmetrically diagonizable, iff $AA^\top=A^\top A$
- *Spectral theorem for symmetric matrices*: Every symmetric matrix A is symmetrically diagonizable (due to Spectral theorem) and all its eigenvalues are real

Positive definite (pd) and positive semi-definite matrices (psd) —
- $A>0$ iff $x^\top Ax>0$
- $A\geq 0$ iff $x^\top Ax\geq 0$

Properties:
- If A is p(s)d, αA is also p(s)d
- If A and B are p(s)d, $A+B$ is also p(s)d
- If $\det(A)=\prod_{i=1}^n\lambda_i>(\geq)\,0$ resp. $\{\lambda_i\}_{i=1}^n>(\geq)\,0$ for pd (psd)

Pd properties:
- I is pd
- If A is pd, A^{-1} is pd
- *Cholesky decomposition*: If A is pd, $A=BB^\top$
- If A and B are pd, $(AB)^{-1}=B^{-1}A^{-1}$

Psd properties:
- If A is psd, $BAB^\top$ is psd

Singular Value Decomposition (SVD)

SVD —
- For $A\in\mathbb{R}^{n\times m}$, orthogonal rotation matrix $U\in\mathbb{R}^{n\times n}$, diagonal scaling and projection matrix $S\in\mathbb{R}^{n\times m}$, and orthogonal rotation matrix $V\in\mathbb{R}^{m\times m}$: $A=USV^\top$
- For symmetric $A\in\mathbb{R}^{n\times n}$: $A=USU^\top$
- In S:
  - Diagonal elements $\sigma_1,\ldots$ are the *singular values* of A
  - If $\sigma_1\geq\sigma_2\ldots\geq 0$, S is unique
  - *Spectral norm* = $\sigma_{max}=\|A\|$
  - Largest singular value σ_{max} is always greater than largest eigenvalue $\rho(A)$
  - *Condition number* = $\sigma_{max}/\sigma_{min}$
  - For square A: Iff $\sigma_1,\sigma_2,\ldots>0$, A is invertible
- SVD is closely related to spectral theorem:
  - According to spectral theorem, every matrix A is symmetrically diagonizable (i.e. $A=Q\Lambda Q^\top$), iff $AA^\top=A^\top A$
  - If we apply SVD to $AA^\top$ resp. $A^\top A$:
    - $AA^\top=USV^\top VS^\top U^\top=U(SS^\top)U^\top$ since V is orthogonal and $V^\top V=I$
    - $A^\top A=VS^\top U^\top USV^\top=V(S^\top S)V^\top$ since U is orthogonal and $U^\top U=I$
  - $SS^\top$ and $S^\top S$ are diagonal matrices with elements $\sigma_1^2,\sigma_2^2,\ldots$
  - Given symmetric diagonalization for any matrix, we see that
    - S contains square root of eigenvalues of $AA^\top$ resp. $A^\top A$
    - U contains eigenvectors of $AA^\top$ as columns resp. V contains eigenvectors of $A^\top A$ as columns
  - According to spectral theorem, symmetric matrix A is symmetrically diagonizable (i.e. $A=Q\Lambda Q^\top$)
  - If we apply SVD to symmetric matrix A, we see that
    - S contains absolute value of eigenvalues of A
    - U contains eigenvectors of A as columns

Pseudo Inverse —
- Pseudo Inverse satisfies certain conditions that make it behave like an inverse for matrices that might not be invertible in the usual sense
- $A^\#=VS^\#U^\top$ where $S^\#$ is obtained from S by transposing it and taking the inverse of non-zero diagonal elements
- $A^\#$ is unique
- If $rank(A)$ = number of rows in A then:
  - $AA^\#=I$
  - $A^\#=A^\top(AA^\top)^{-1}$
  - Pseudo inverse provides mininum norm solution, when system $y=Ax$ is underdetermined: $x=A^\top(AA^\top)^{-1}y$
- If $rank(A)$ = number of columns in A then:
  - $A^\#A=I$
  - $A^\#=(A^\top A)^{-1}A^\top$
  - Pseudo inverse provides least squares solution, when system $y=Ax$ is overdetermined: $x=(A^\top A)^{-1}A^\top y$

Properties —
- $AA^\#A=A$
- $A^\#AA^\#=A^\#$
- $(A^\top)^\#=(A^\#)^\top$
- $(AA^\top)^\#=(A^\#)^\top A^\#$
- $A^\#x=0\Leftrightarrow x^\top A=0\Leftrightarrow A^\top x=0$
- Properties can be proven by replacing A by its SVD and $A^\#$ by its definition
- Column space of $A^\#$ equals column space of $A^\top$
- Property can be proven by replacing A and $A^\#$ by their SVD

Hilbert Space S

Equivalence modulo norm zero:
- Challenge: In some cases, $v\cdot v=\Leftrightarrow v=0_v$ does not hold
- Issue is resolved by defining equivalence classes of vectors: $[v]=\{v'\in V:\|v-v'\|=0\}$
- Implications: Modified meaning of equality:
  - For functions: $f=g$ means $\int_T|f(t)-g(t)|^2dt=0$
  - For random variables: $X=Y$ means $\mathbb{E}[|X-Y|^2]=0$

Existence (convergence) of the inner product:
- Challenge: In some cases, inner product does not exist for all $v,w\in V$
- Issue is resolved by restricting attention to subsets of V where the norm is finite: $V_{fn}=\{v\in|\|v\|<\infty\}$

Hilbert spaces:
- Vector space with an inner product that satisfies the additional condition of *completeness*:
  - Every Cauchy sequence in V converges to an element in V resp.
  - Limit vectors, that Cauchy sequences tend towards, are also elements of V
  - Cauchy sequence: Sequence of points that get closer and closer
- If we make modifications for above challenges (equivalence modulo norm zero, existence of inner product), vector spaces can be transformed to Hilbert spaces

A vector space V with an inner product is a Hilbert space if every Cauchy sequence in V converges to an element in V. The following spaces, with appropriate modifications, are Hilbert spaces:
- $\ell^2(\mathbb{Z})$: square-summable discrete-time signals.
- $L^2(\mathbb{R})$: square-integrable functions on $\mathbb{R}$.
- $L^2(T)$: square-integrable functions on an interval T.
- Random variables with finite second moments.

The spaces mentioned above, with modifications for norm equivalence and finiteness of the inner product, are Hilbert spaces.

Semiring

Semiring —
- A 5-tuple $\mathcal{S}=(\mathcal{A},\oplus,\otimes,\underline{0},\underline{1})$ with the following properties:
  - $(\mathcal{A},\oplus,\underline{0})$ is a commutative monoid
  - $(\mathcal{A},\otimes,\underline{1})$ is a monoid
  - $\otimes$ distributes over $\oplus$: $(a\oplus b)\otimes c=(a\otimes c)\oplus(b\otimes c)$ $c\otimes(a\oplus b)=(c\otimes a)\oplus(c\otimes b)$
  - $\underline{0}$ is an annihilator for $\otimes$: $a\otimes\underline{0}=\underline{0},\quad \underline{0}\otimes a=\underline{0}$
- A *commutative semiring*: semiring where $\otimes$ is commutative: $a\otimes b=b\otimes a$
- An *idempotent semiring*: semiring where $\oplus$ is idempotent: $a\oplus a=a$
- A *closed semiring*: semiring augmented with additional unary operation: *Kleene star* $*$ (or *asteration*):
  - $x^*=\bigoplus_{n=0}^\infty x^{\otimes n}$
  - Allows computation of infinite sums
  - Kleene star must obey: $x^*=\underline{1}\oplus x\otimes x^*$ $x^*=\underline{1}\oplus x^*\otimes x$
  - For example, the geometric series is a Kleene star:
    - $x^*=\sum_{n=0}^\infty x^n=\frac{1}{1-x}$ for $x\in(0,1)$
      Proof:
      $x^*=\frac{1}{1-x}=1+\frac{1}{1-x}-1=1+\frac{1-1+x}{1-x}=1+\frac{x}{1-x}=1+x\frac{1}{1-x}=1+xx^*$
  - We can approximate Kleene star: $\sum_{k=0}^K M^k\approx M^*$ as $K\to\infty$ if $\rho(M)<1$ resp. $\sigma_{\max}(M)=\|M\|^2<1$
  - Truncation error of this approximation: $\|M^*-\sum_{k=0}^K M^k\|\leq\frac{\sigma_{\max}(M)^{K+1}}{1-\sigma_{\max}(M)}$
    Proof:
    - $M^*-\sum_{n=0}^K M^n=\sum_{n=k+1}^\infty M^n=M^{k+1}\sum_{n=k+1}^\infty M^{n-(k+1)}=M^{k+1}\sum_{m=0}^\infty M^m=M^{k+1}M^*$
    - Then: $\|M^*-\sum_{k=0}^K M^k\|=\|M^{k+1}M^*\|$
    - Using the Cauchy-Schwarz inequality: $\|M^{k+1}M^*\|<\|M^{k+1}\|\,\|M^*\|$
    - For $\|M^{k+1}\|$: $\|M^{k+1}\|\leq\|M\|^{k+1}=\sigma_{\max}(M)^{k+1}$
    - For $\|M^*\|$: $\sum_{n=0}^\infty\|M^n\|\leq\sum_{n=0}^\infty\|M\|^n=\sum_{n=0}^\infty\sigma_{\max}(M)^n=\frac{1}{1-\sigma_{\max}(M)}$ where the second-to-last term is a geometric series
    - Then: $\|M^*-\sum_{n=0}^K M^n\|\leq\frac{\sigma_{\max}(M)^{K+1}}{1-\sigma_{\max}(M)}$
  - Good approximation, especially if $\sigma_{\max}\ll 1$, since then the error becomes very small
- A *0-closed semiring*:
  - $\underline{1}\oplus a=\underline{1}$
  - Examples: tropical and arctic semiring
  - Properties:
    - $x^*=\bigoplus_{n=0}^{N-1}x^{\otimes n}$ since cycles in a path of length $\geq N$ do not contribute $=\bigoplus_{n=0}^{N-1}M^n=(I+M)^{N-1}$ if x is a matrix $M=\bigoplus_{n=0}^{N-1}\bigotimes_{k=0}^{\lfloor\log_2 n\rfloor}M^{a_k2^k}$ if we use binary decomposition on matrix M
    - Idempotent

Kinds of semirings:
- Boolean semiring $(\{0,1\},\vee,\wedge,0,1)$
- Real semiring $(\mathbb{R},+,\times,0,1)$
- Probability semiring $([0,1],+,\times,0,1)$
- Tropical semiring $(\mathbb{R}\cup\{\infty\},\min,+,\infty,0)$
- Arctic semiring resp. max-plus semiring $(\mathbb{R}\cup\{-\infty\},\max,+,-\infty,0)$
- Log semiring: $(\mathbb{R}\cup\{-\infty\},\oplus_{\log},+,-\infty,0)$ where $a\oplus_{\log}b=\log(e^a+e^b)$
- Language semiring: $(2^{\Sigma^*},\cup,\circ,\{\},\{\epsilon\})$ where 2^{Σ^*} is the set of all possible languages and $A\circ B=\{a\circ b\mid a\in A,b\in B\}$ is the concatenation

Monoid —
- Consists of a set $\mathcal{A}$, an operation $*$, and an identity element e, such that:
  - Associativity: $(a*b)*c=a*(b*c)$
  - Identity: $a*e=e*a=a$
- A *commutative monoid*: additionally commutative: $a*b=b*a$

2 Calculus

Derivatives

Rules —
- Sum rule: $\frac{\partial f+g}{\partial x}=\frac{\partial f}{\partial x}+\frac{\partial g}{\partial x}$
- Product rule: $\frac{\partial f\times g}{\partial x}=f\times\frac{\partial g}{\partial x}+g\times\frac{\partial f}{\partial x}$
- Chain rule: $\frac{\partial f(g)}{\partial x}=\frac{\partial f}{\partial g}\times\frac{g}{\partial x}$

Common derivatives —
- $\frac{\partial x^n}{\partial x}=nx^{n-1}$
- $\frac{\partial log(x)}{\partial x}=\frac{1}{x}$
- $\frac{\partial e^{kx}}{\partial x}=k\times e^{kx}$
- $\frac{\partial\sqrt{x}}{\partial x}=\frac{1}{2\sqrt{x}}$

Partial and directional derivative —
- For a function that depends on n variables $\{x_i\}_{i=2}^n$, partial derivative is slope of tangent line along direction of one specific variable x_i
- Directional derivative is slope of tangent line along direction of selected unit vector u

Gradient —

- Given scalar-valued function $f:\mathbb{R}^n \to \mathbb{R}$, returns vector containing first-order partial derivatives:
  $\nabla_x f : [\frac{\partial f}{\partial x_1} ... \frac{\partial f}{\partial x_n}]^\top$
- Gradient points in direction of greatest upward slope of f
- Magnitude of gradient equals rate of change when moving into direction of greatest upward slope

Hessian —
- Given scalar-valued function $f:\mathbb{R}^n \to \mathbb{R}$, returns matrix containing second-order partial derivatives:
  $$\mathcal{H} = \nabla_x^2 f : \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & ... & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ ... & ... & ... \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & ... & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix}$$
- $\mathcal{H}$ is symmetric

Jacobian —
- Given vector-valued function $f:\mathbb{R}^n \to \mathbb{R}^m$ with $f = [f_1(x),...,f_m(x)]^\top$, returns matrix containing first-order partial derivatives:
  $$\nabla_x f : \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & ... & \frac{\partial f_1}{\partial x_n} \\ ... & ... & ... \\ \frac{\partial f_m}{\partial x_1} & ... & \frac{\partial f_m}{\partial x_n} \end{bmatrix}$$

Scalar-by-matrix derivative — TBA

Total derivative — TBA

Matrix calculus rules —
- $\frac{\partial a^\top x}{\partial x} = a$
- $\frac{\partial x^\top A x}{\partial x} = (A + A^\top)x$
- $\frac{\partial a^\top A b}{\partial A} = ab^\top$
- For symmetric A:
  $\frac{\partial x^\top A x}{\partial x} = 2Ax$
- For square A:
  - $\frac{\partial a^\top A^{-1} b}{\partial A} = -(A^\top)^{-1} a c^\top (A^\top)^{-1}$
  - $\frac{\partial log(|A|)}{\partial A} = (A^\top)^{-1}$

Extrema

Conditions for local minima and maxima —
- Point is a stationary point, i.e. first-order derivative = 0
- If Hessian is pd, it's a local minimum, if Hessian is nd, it's a local maximum, if Hessian is indefinite, it's a saddle point
- Local minima and maxima are the unique global minima and maxima in strictly convex functions resp. one of possibly infinitely many global minima and maxima in convex functions

Convexity —
- For a convex function:
  - $f(\lambda x + (1-\lambda)y \leq \lambda f(x) + (1-\lambda) f(y)$ with $\lambda \in [0,1]$
  - Hessian of stationary point(s) is psd
  - Global minimum exists, but may not be unique
- For a strictly convex function:
  - $f(\lambda x + (1-\lambda)y < \lambda f(x) + (1-\lambda) f(y)$ with $\lambda \in [0,1]$
  - Hessian of stationary point is pd
  - Unique global minimum exists
- Sum of convex functions $f_2(x) + f_1(x)$ is also convex, sum of convex and strictly convex function is strictly convex
- Chain of convex functions $f_2(f_1(x))$, where outer function $f_2(x)$ is increasing, is also convex
- Scalar multiple of convex function $\lambda f(x)$, where $\lambda \geq 0$, is also convex
- Any norm is convex

Nature of optimum — What does Hessian and function look like?
- If Hessian is pd and loss function is strictly convex, stationary point is a global minimum, and there is a unique solution
- If Hessian is psd and loss function is convex, stationary point is a global minimum, and there may be a geometrically unique or infinitely many solutions
- If Hessian is p(s)d but loss function is not convex, stationary point may be a local minimum and there may be a geometrically unique or infinitely many solutions

Optimization approach — Is function differentiable, continuous, and are relevant terms invertible?
- If yes, analytically solvable
- If no, numerically solvable (e.g. via gradient descent)

Constrained optimization —
- Lagrangian function: $\mathcal{L}(x,\lambda) = f(x) + \lambda g(x)$, where $g(x)$ is an $(m-1)$ dimensional constraint surface and λ is the Lagrange multiplier
- $\nabla_x \mathcal{L} = \nabla_x f(x) + \lambda \nabla_x g(x)$
- $\nabla_\lambda \mathcal{L} = g(x)$
- Solution is feasible if it fulfills constraints and optimal, if no other feasible solution produces a lower error
- Minimizing over Lagrangian $\mathcal{L}(x,\lambda) = f(x) + \lambda g(x)$ corresponds to minimizing log-loss resp. negative likelihood:
  - $\hat{x} = argmax_{x p(D|x) \rho(x)}$
  - $= argmin_{x(-logp(D|x)+k(x))}$
  - where $k(x) = -log\rho(x)$

  $\mathcal{L}(x,\lambda) = f(x) + \lambda g(x)$

For equality constraints: Minimize $f(x)$ subject to $g(x) = 0$
- Gradient of $f(x)$ must be orthogonal to constraint surface, otherwise (if it points into any direction along the constraint surface) $f(x)$ could still decrease for movements along the constraint surface
- On the constraint surface, $g(x)$ is a constant, so moving along any direction on the constraint surface has a directional derivative of 0. Since the gradient of $g(x)$ points into the direction of steepest ascent, it must be orthogonal to the constraint surface, otherwise (if it points into any direction along the constraint surface) $g(x)$ would not be constant on the constraint surface
- Then, gradients are parallel at optimum: $\nabla_x f(x^*) = \lambda \times \nabla_x g(x^*)$
- To find x^* and λ^*:
  - $\nabla_x L = 0$, expresses parallelity condition at minimum x^*
  - $\nabla_\lambda L = 0$, expresses constraint
  - This is an unconstrained optimization problem
- Optimum x^* and λ^* represents a saddle point of $\mathcal{L}$

For inequality constraints: Minimize $f(x)$ subject to $g(x) \leq 0$
- If x^* lies in $g(x) < 0$, constraint is inactive
- Otherwise, if x^* lies in $g(x) = 0$, constraint is active:
  - Gradient of $f(x)$ must point towards $g(x) < 0$ region, otherwise (if it would point away from $g(x) < 0$ region) the optimum would lie in this region
  - Then, gradients are anti-parallel at optimum: $\nabla_x f(x^*) = -\lambda \times \nabla_x g(x^*)$
- To find x^* and λ^*:
  - $\nabla_x \mathcal{L} = 0$ subject to *Karush Kuhn Tucker conditions*:
    - $g(x) \leq 0$
    - $\lambda \geq 0$
    - *Complementary slackness condition*: $\lambda g(x) = 0$, with $\lambda = 0, g(x) < 0$ for inactive constraints and $\lambda > 0, g(x) = 0$ for active constraints
  - $\nabla_\lambda \mathcal{L} = 0$ given complementary slackness condition
  - This is not an unconstrained optimization problem, but can be solved via duality
- Optimum x^* and λ^* represents a saddle point of $\mathcal{L}$

For multiple constraints: Minimize $f(x)$ subject to m inequality constraints $\{g^{(i)}(x) \leq 0\}_{i=1}^m$ and p equality constraints $\{h^{(j)}(x) = 0\}_{j=1}^p$
- Then, Lagrangian is given by: $\mathcal{L}(x,\lambda,\mu) = f(x) + \sum_{i=1}^m \mu^{(i)} g^{(i)}(x) + \sum_{j=1}^p \lambda^{(j)} h^{(j)}(x)$
- Then, general solution x^*, λ^*, μ^* is given by: $\nabla_x \mathcal{L} = 0$ subject to:
  - $\{g^{(i)}(x) \leq 0\}_{i=1}^m$ and $\{h^{(j)}(x) = 0\}_{j=1}^p$
  - $\{\mu^{(i)} \geq 0\}_{i=1}^m$
  - $\{\mu^{(i)} g^{(i)}(x) = 0\}_{i=1}^m$

Primal problem:
- $\min_x[\max_{\lambda,\mu} \mathcal{L}]$
- $\max_{\lambda,\mu} \mathcal{L} = f(x) + \max_{\lambda,\mu}[\sum_{i=1}^m \mu^{(i)} g^{(i)}(x) + \sum_{j=1}^p \lambda^{(j)} h^{(j)}(x)]$
- Second term gives rise to barrier function:
  - $= 0$ subject to constraints being met, given complementary slackness condition for inequality constraints and $h^{(j)}(x) = 0$ for equality constraints, which implies that dual problem becomes $\min_x(f(x))$
  - $= \infty$ otherwise, which implies that primal problem cannot be solved

Solving inequality constraints via duality – *weak duality*:
- Weak duality always holds and gives a lower bound of minimum of primal problem
- Given minimax theorem, $\min_x[\max_{\lambda,\mu} \mathcal{L}] = f(x)$ (provided barrier function) $\geq \max_{\lambda,\mu}[\min_x \mathcal{L}]$
- $\min_x \mathcal{L}$ is an unconstrained optimization problem
- $\max_{\lambda,\mu}[\min_x \mathcal{L}]$ is a concave maximization problem

Solving inequality constraints via duality – *strong duality*:
- Strong duality holds under certain conditions, for example *Slater's condition* if there exists a solution that strictly fulfills all inequality constraints $\{g^{(i)}(x) < 0\}_{i=1}^m$
- Then, $\min_x[\max_{\lambda,\mu} \mathcal{L}] = \max_{\lambda,\mu}[\min_x \mathcal{L}]$
- $\min_x \mathcal{L}$ can be solved for general solution x^* in terms of λ, μ
- Plug x^* back into $\mathcal{L}$ and maximize to find solutions λ^*, μ^*
- Specify x^* based on λ^*, μ^*

Integrals

Indefinite integral —
- $F(x) = \int f(x) dx$
- $F'(x) = f(x)$

Definite integral —
- $F(b) - F(a) = \int_a^b f(x) dx$
- $F'(x) = f(x)$

Common integrals —
- $f(x) = x^n \to F(x) = \frac{x^{n+1}}{n+1}$ for $n \neq 1$
- $f(x) = \frac{1}{x} \to F(x) = log(x)$
- $f(x) = e^x \to F(x) = e^x$

Other

Common exp and log rules —
- $a^m \cdot a^n = a^{m+n}$
- $\frac{a^m}{a^n} = a^{m-n}$
- $(ab)^n = a^n b^n$
- $\left(\frac{a}{b}\right)^n = \frac{a^n}{b^n}$
- $a^{-n} = \frac{1}{a^n}$
- $a^0 = 1$
- $a^1 = a$
- $\log(xy) = \log x + \log y$
- $\log\left(\frac{x}{y}\right) = \log x - \log y$
- $\log(x^n) = n \log x$
- $\log 1 = 0$
- $\log(x<1) < 0$
- $\log(x>1) > 0$
- $e^{log(x)} = log(e^x) = x$

Geometric series —
- Finite: $S_n = \sum_{i=1}^n a_i r^{i-1} = a_1(\frac{1-r^n}{1-r})$
- Infinite: $S = \sum_{i=0}^\infty a_i r^i = \frac{a_1}{1-r}$ for $r < 1$

3 Probability and Statistics

Terminology

Kolmogorov axioms — Probability space defined by:
- Sample space: All possible outcomes $\Omega = \{\omega_1,...,\omega_n\}$
- Event space: All possible results, where an event is a subset of the sample space
- Probability measure: Function that assigns a probability to an event

Axioms:
- Event space must be a *sigma algebra*:
  - If A is in sample space with $P = a$, its complement is also in sample space with $P = 1 - a$
  - If $A_1,...A_n$ are in sample space with $P = a_1,...,a_n$, their union is also in sample space with $P = a_1 + ... + a_n$
- Probability measure must satisfy:
  - $0 \leq \mathbb{P}(A) \leq 1$
  - $\mathbb{P}(\Omega) = 1$
  - If $A_1, A_2, ...$ are in sample space and do not intersect, then $\mathbb{P}(A_1 \cup A_2 \cup ...) = \int_{n=1}^\infty \mathbb{P}(A_n)$

Further properties:
- All sets than can be formed from left and right inclusive interval $[0,a]$ are events. On that basis:
  - $(a,1] = [0,a]^c \in$ event space, with $P = 1 - a$
  - $(a,b] = ([0,a] \cup (b,1])^c = ([0,a] \cup [0,b]^c)^c \in$ event space, with $P = 1 - (a + 1 - b) = 1 - a - 1 + b = b - a$
  - $\{0\} \in$ event space, with $P = 0$
  - $\{a\} \in$ event space, with $P = 0$
  - $[a,b] = \{a\} \cup (a,b] = \{a\} \cup ([0,a] \cup [0,b]^c)^c \in$ event space, with $P = 0 + b - a$
  - $[a,b) = [a,b] \backslash \{b\} = ([0,a] \cup [0,b]^c)^c \backslash \{b\} \in$ event space, with $P = b - a - 0$

Variables —
- Random variable:
  - Discrete random variable: Characterized by pmf
  - Continuous random variable: Characterized by pdf
- Independent random variables:
  - $\mathbb{P}(A|B) = \mathbb{P}(A)$ and $\mathbb{P}(B|A) = \mathbb{P}(B)$
  - From this follows that $f_{A|B}(a|b) = f_A(a)$ and $\mathbb{E}(A|B) = \mathbb{E}(A)$
  - $\mathbb{P}(A \cap B) = \mathbb{P}(A)\mathbb{P}(B)$
  - From this it follows that $f_{A,B}(a,b) = f_A(a) f_B(b)$ and $\mathbb{E}(A,B) = \mathbb{E}(A)\mathbb{E}(B)$
  - Correlation is 0
  - Functions of independent random variables are also independent
- Conditionally independent random variables: Two random variables $\mathcal{X}$ and $\mathcal{Y}$ are conditionally independent, if there is a confounder $\mathcal{L}$ that causally affects both variables, but if we control for this confounder, the variables are not causally connected
- I.I.D. random variables: Independent and from identical distribution

Events —
- Complement: $\mathbb{P}(A^C) = 1 - \mathbb{P}(A)$ and $\mathbb{P}(A \cup A^C) = \mathbb{P}(A)\mathbb{P}(A^C)$
- Disjoint / mutually exclusive vs. joint / mutually inclusive
- Subset $A \subset B$ with $\mathbb{P}(A) < \mathbb{P}(B)$

Probabilities —
- Marginal probability $\mathbb{P}(A)$: Probability for single variable: $p(\mathcal{X}) = \sum_{\mathcal{Y}} p(x,y)$ resp. $f(\mathcal{X}) = \int_{\mathcal{Y}} f(x,y) dy$
- Joint probability $\mathbb{P}(A \cap B)$: Probability for combination of variables, given by all possible combinations resp. convolution of their pdfs
- Conditional probability $\mathbb{P}(A|B) = \frac{\mathbb{P}(A \cap B)}{\mathbb{P}(B)}$: Probability for variable, given other variable: $p(\mathcal{X}|\mathcal{Y}) = \frac{p(x,y)}{\sum_{\mathcal{X}} p(x,y)}$ resp. $f(\mathcal{X}|\mathcal{Y}) = \frac{f(x,y)}{\int_{\mathcal{X}} f(x,y) dy}$
  - $\mathbb{P}(A|B) = 1 - \mathbb{P}(A^C|B)$
  - $\mathbb{P}(A_1|B) + \mathbb{P}(A_2|B) + ... = 1$

- If conditioning on subset S:

$$p(x|S) = \begin{cases} p(x)/p(x \in S) & x \in S \\ 0 & x \notin S \end{cases}$$

- Bayesian terminology:
  - Prior $\mathbb{P}(\text{parameter})$
  - Posterior $\mathbb{P}(\text{parameter}|\text{data})$
  - Likelihood $\mathbb{P}(\text{data}|\text{parameter})$
  - Evidence $\mathbb{P}(\text{data})$
- *Bayes theorem*: Posterior $\mathbb{P}(A|B) = \frac{\text{Likelihood } \mathbb{P}(B|A) \times \text{Prior } \mathbb{P}(A)}{\text{Evidence } \mathbb{P}(B)}$
- Attention! In $p(\cdot|\theta)$ the $|$ can either refer to parametrizing on θ (parameter is part of the distribution's form but isn't observed or fixed) or conditioning on θ (parameter takes a observed and fixed value, and we evaluate the distribution on this condition)

Measures

n^{th} moment — $\mathbb{E}(\mathcal{X}^n) = \int_{-\infty}^{\infty} x^n \times f(x)dx$

Expected value — $\mathbb{E}(\mathcal{X}) = \sum_{\mathcal{X}} x \times p(x)$ resp. $\mathbb{E}(\mathcal{X}) = \int_{-\infty}^{\infty} x \times f(x)dx$ with pmf resp. pdf —
Properties:
- $\mathbb{E}(a) = a$
- $\mathbb{E}(a\mathcal{X} + \beta) = a\mathbb{E}(\mathcal{X}) + \beta$
- $\mathbb{E}(a\mathcal{X} + \beta\mathcal{Y}) = a\mathbb{E}(\mathcal{X}) + \beta\mathbb{E}(\mathcal{Y})$
- For orthogonal variables:
  - $\mathbb{E}(\mathcal{X}\mathcal{Y}) = 0$
  - $\mathbb{E}((\mathcal{X} + \mathcal{Y})^2) = \mathbb{E}(\mathcal{X}^2) + \mathbb{E}(\mathcal{Y}^2)$
- For independent variables: $\mathbb{E}(\mathcal{X}\mathcal{Y}) = \mathbb{E}(\mathcal{X})\mathbb{E}(\mathcal{Y})$
- For vector $\boldsymbol{y} = \boldsymbol{Ax}$: $\mathbb{E}_{\boldsymbol{y}}(\boldsymbol{Ax}) = \boldsymbol{A}\mathbb{E}_X(\boldsymbol{x})$
- For conditional probability (A is an event, X is a random variable):
  - $\mathbb{E}[P(X|A)] = \sum_x P(X = x) \times P(X = x|A)$ resp. $\int_{-\infty}^{\infty} f_X(x) \times f_{X|A}(x)dx$
  - $\mathbb{E}[P(A|X)] = P(A) = \sum_x P(X = x) \times P(A|X = x)$ resp. $\int_{-\infty}^{\infty} f_X(x) \times P(A|X = x)dx$
- For conditional events resp. variables (A is an event, X is a random variable):
  - $\mathbb{E}(X|A) = \sum_x x \times P(X = x|A)$ resp. $\int_{-\infty}^{\infty} x \times f_{X|A}(x)dx$
  - $\mathbb{E}(A|X) = P(A|X)$
  - $\mathbb{E}[\mathbb{E}(X|A)] = \mathbb{E}(X)$
    Proof:
    * $\mathbb{E}[\mathbb{E}(X|A)] = \int_{-\infty}^{\infty} f_A(a)\mathbb{E}(X|A)da = \int_{-\infty}^{\infty} f_A(a)\int_{-\infty}^{\infty} x f_{X|A}(x|a)dxda = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} x f_{X,A}(x,a)dxda = \int_{-\infty}^{\infty} x\int_{-\infty}^{\infty} f_{X,A}(x,a)dadx = \int_{-\infty}^{\infty} x f_X(x)dx = \mathbb{E}(X)$
- *Cauchy Schwarz inequality*: $\mathbb{E}(\mathcal{X},\mathcal{Y})^2 \leq \mathbb{E}(\mathcal{X}^2)\mathbb{E}(\mathcal{Y}^2)$

Median — Real number M defined by $P(X < M) = P(X > M)$

Standard deviation — $\sqrt{\mathbb{V}(\mathcal{X})}$

Covariance —
- Univariate variance of a random variable: $\mathbb{V}(\mathcal{X}) = \mathbb{E}((\mathcal{X} - \mathbb{E}(\mathcal{X}))^2) = \mathbb{E}(\mathcal{X}^2) - \mathbb{E}(\mathcal{X})^2$ where $\mathbb{E}(\mathcal{X}^2)$ is the unnormalized correlation resp. inner product
- Univariate covariance of two random variables: $\mathbb{C}\text{ov}(\mathcal{X},\mathcal{Y}) = \mathbb{E}((\mathcal{X} - \mathbb{E}(\mathcal{X}))(\mathcal{Y} - \mathbb{E}(\mathcal{Y}))) = \mathbb{E}(\mathcal{X}\mathcal{Y}) - \mu_{\mathcal{X}}\mu_{\mathcal{Y}}$ where $\mathbb{E}(\mathcal{X}\mathcal{Y})$ is the unnormalized correlation resp. inner product
- Proof (schematically for variance): $\mathbb{V}(\mathcal{X}) = \mathbb{E}((\mathcal{X} - \mathbb{E}(\mathcal{X}))^2) = \mathbb{E}[\mathcal{X}^2 - \mathcal{X}\mathbb{E}(\mathcal{X}) - \mathcal{X}\mathbb{E}(\mathcal{X}) + \mathbb{E}(\mathcal{X})^2] = \mathbb{E}[\mathcal{X}^2] - \mathbb{E}[\mathcal{X}]\mathbb{E}(\mathcal{X}) - \mathbb{E}[\mathcal{X}]\mathbb{E}(\mathcal{X}) + \mathbb{E}(\mathcal{X})^2 = \mathbb{E}(\mathcal{X}^2) - \mathbb{E}(\mathcal{X})^2$ where $\mathbb{E}(\mathcal{X}^2)$
- Multivariate covariance matrix of a vector:
  - $\Sigma = \mathbb{C}\text{ov}(\boldsymbol{\mathcal{X}}) = \mathbb{E}((\boldsymbol{\mathcal{X}} - \mathbb{E}(\boldsymbol{\mathcal{X}}))(\boldsymbol{\mathcal{X}} - \mathbb{E}(\boldsymbol{\mathcal{X}}))^\intercal) = \mathbb{E}(\boldsymbol{\mathcal{X}}\boldsymbol{\mathcal{X}}^\intercal) - \mathbb{E}(\boldsymbol{\mathcal{X}})\mathbb{E}(\boldsymbol{\mathcal{X}})^\intercal = \begin{bmatrix} \mathbb{V}\text{ar}(\mathcal{X}_1) & \dots & \mathbb{C}\text{ov}(\mathcal{X}_1,\mathcal{X}_m) \\ \dots & \dots & \dots \\ \mathbb{C}\text{ov}(\mathcal{X}_m,\mathcal{X}_1) & \dots & \mathbb{V}\text{ar}(\mathcal{X}_m) \end{bmatrix}$ where $\boldsymbol{R} = \mathbb{E}(\boldsymbol{\mathcal{X}}\boldsymbol{\mathcal{X}}^\intercal)$ is the unnormalized correlation matrix
  - Σ and $\boldsymbol{R}$ are symmetric and psd
  - $\Sigma = \boldsymbol{R} - \boldsymbol{\mu}_X\boldsymbol{\mu}_X^\intercal$

Properties - variance:
- $\mathbb{V}(a) = 0$
- $\mathbb{V}(a\mathcal{X} + \beta) = a^2\mathbb{V}(\mathcal{X})$
- $\mathbb{V}(\mathcal{X} + \mathcal{Y}) = \mathbb{V}(\mathcal{X}) + 2\mathbb{C}\text{ov}(\mathcal{X},\mathcal{Y}) + \mathbb{V}(\mathcal{Y})$
- For uncorrelated (and independent) variables: $\mathbb{V}(\mathcal{X} + \mathcal{Y}) = \mathbb{V}(\mathcal{X}) + \mathbb{V}(\mathcal{Y})$
- For independent variables: $\mathbb{V}(\mathcal{X}\mathcal{Y}) = \mathbb{E}((\mathcal{X}\mathcal{Y})^2)\mathbb{E}(\mathcal{X}\mathcal{Y})^2$
- For vector $\boldsymbol{y} = \boldsymbol{Ax}$: $\mathbb{V}_{\boldsymbol{y}} = \boldsymbol{A}\mathbb{V}_X\boldsymbol{A}^\intercal$
- For zero-mean variable: $\mathbb{V}(\mathcal{X}) = \mathbb{E}(\mathcal{X}^2) - \mathbb{E}(\mathcal{X})^2 = \mathbb{E}(\mathcal{X}^2)$ since $\mathbb{E}(\mathcal{X}) = 0$

Properties - covariance:
- $\mathbb{C}\text{ov}(\mathcal{X},\mathcal{X}) = \mathbb{V}(\mathcal{X})$
- $\mathbb{C}\text{ov}((a\mathcal{X} + \beta\mathcal{Y}),\mathcal{Z}) = a\mathbb{C}\text{ov}(\mathcal{X},\mathcal{Z}) + \beta\mathbb{C}\text{ov}(\mathcal{Y},\mathcal{Z})$
- If covariance of two random variables is 0, they are uncorrelated, but not necessarily independent. Then, $\mathbb{E}(\mathcal{X}\mathcal{Y}) = \mathbb{E}(\mathcal{X})\mathbb{E}(\mathcal{Y})$
- If covariance and unnormalized correlation of two random variables is 0, they are orthogonal, but not necessarily independent. Then, $\mathbb{E}(\mathcal{X}\mathcal{Y}) = 0$
- For vector $\boldsymbol{y} = \boldsymbol{Ax}$:
  - $\Sigma_{\boldsymbol{y}} = \boldsymbol{A}\Sigma_X\boldsymbol{A}^\intercal$
  - $\boldsymbol{R}_{\boldsymbol{y}} = \boldsymbol{A}\boldsymbol{R}_X\boldsymbol{A}^\intercal$
- For zero-mean variables: $\mathbb{C}\text{ov}(\mathcal{X},\mathcal{Y}) = \mathbb{E}(\mathcal{X}\mathcal{Y}) - \mu_{\mathcal{X}}\mu_{\mathcal{Y}} = \mathbb{E}(\mathcal{X},\mathcal{Y})$ since $\mu_{\mathcal{X}} = \mu_{\mathcal{Y}} = 0$

Cauchy Schwarz inequality:
- $\mathbb{C}\text{ov}(\mathcal{X},\mathcal{Y})^2 \leq \mathbb{V}(\mathcal{X})\mathbb{V}(\mathcal{Y})$
- $\mathbb{E}(\mathcal{X}\mathcal{Y})^2 \leq \mathbb{E}(\mathcal{X}^2)\mathbb{V}(\mathcal{Y}^2)$

Correlation — Normalized covariance
- Univariate correlation of a random variable: $\mathbb{C}\text{or}(\mathcal{X},\mathcal{Y}) = \frac{\mathbb{C}\text{ov}(\mathcal{X},\mathcal{Y})}{\sqrt{\mathbb{V}(\mathcal{X})}\sqrt{\mathbb{V}(\mathcal{Y})}}$
- Multivariate correlation matrix of a vector:
  - $\boldsymbol{P} = \mathbb{C}\text{or}(\boldsymbol{\mathcal{X}}) = \begin{bmatrix} 1 & \dots & \mathbb{C}\text{or}(\mathcal{X}_1,\mathcal{X}_m) \\ \dots & \dots & \dots \\ \mathbb{C}\text{or}(\mathcal{X}_m,\mathcal{X}_1) & \dots & 1 \end{bmatrix}$
  - $\boldsymbol{P}$ is symmetric and psd
  - Correlation is bounded between 0 and 1, given Cauchy Schwarz Inequality
  - If correlation of two random variables is 0, they are not necessarily independent

Skewness — 3^{rd} moment TBA

Kurtosis — 4^{th} moment TBA

Probability Distributions

PMF, CDF, PDF —
- Cumulative density function $F(r)$ (CDF): $F(r) = p(x \leq r)$
- Probability mass function $p(x)$ (PMF) for discrete random variables: $p(x)$
- Probability density function (PDF) $f(x)$ for continuous random variables: $\int_{-\infty}^{r} f(x)dx = p(x \leq r) = \int_{-\infty}^{\infty}\int_{-\infty}^{r} f(x,y)dxdy = p(x \leq r) = F(r)$
- Properties of CDF and PDF:
  - Derivative of CDF returns PDF, integral of PDF returns CDF
  - Monotonically non-decreasing: If $s < r, F(s) < F(r)$
  - $lim_{r\to-\infty} F(r) = 0$
  - $lim_{r\to\infty} F(r) = 1$
  - Right-continuous: $lim_{s\to-r^+} F(s) = F(r)$
  - $lim_{s\to-r^-} F(s) = F(x < r) = F(s) - F(x = r)$
  - $\int_a^b f(x)dx = F(b) - F(a) = p(a < x \leq b)$
  - $\int_{-\infty}^{\infty} f(x)dx = 1$

Normal distribution — $\mathcal{X} \sim \mathcal{N}(\mu,\sigma^2)$

For univariate, PDF: $\frac{1}{\sqrt{2\pi\sigma}} exp(\frac{-(x-\mu)^2}{2\sigma^2})$

For multivariate, PDF: $\frac{1}{2\pi\sigma^{n/2}} \frac{1}{|\Sigma|^{1/2}} exp(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^\intercal\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu}))$ where the term in the exponent is a quadratic form

Convolution: $\int \mathcal{N}(a;Bc,D) \times \mathcal{N}(c;e,F)dc = \int \mathcal{N}(a;Be,D + BFB^\intercal$

Standard normal distribution — Normal distribution, standardized via z-score $z = \frac{x-\mu}{\sigma}$, which results in $\mu = 0$ and $\sigma = 1$

Bernoulli distribution — trial with success (probability p) or failure (probability $1-p$)
- $\mathcal{X} \sim \text{Bernoulli}(p)$
- Mean: $\mathbb{E}(x) = p$
- PDF: $p(x)p^x(1-p)^x$
- Variance: $\mathbb{V}(x) = p(1-p)$

Binomial distribution — n independent Bernoulli trials with k successes
- $\mathcal{X} \sim \text{Bin}(n,p)$
- Mean: $\mathbb{E}(x) = np$
- PDF: $\binom{n}{x}p^k(1-p)^{n-k}$
- Variance: $\mathbb{V}(x) = np(1-p)$

Poisson distribution —
- $\mathcal{X} \sim \text{Pois}(\lambda)$
- Mean: $\mathbb{E}(x) = \lambda$
- PDF: $e^{-\lambda}\frac{\lambda^x}{x!}$
- Variance: $\mathbb{V}(x) = \lambda$

Beta distribution —
- X takes values $\in [0,1]$
- Represents the probability of a Bernoulli process after observing $\alpha - 1$ successes and $\beta - 1$ failures
- $\mathcal{X} \sim \text{Beta}(\alpha,\beta)$ where $\alpha,\beta > 0$
- PDF: $\frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)+\Gamma(\beta)} x^{\alpha-1}(1-x)^{\beta-1}$ where $\Gamma(\alpha) = \int_0^\infty u^{\alpha-1}e^{-u}du$
- Mean: $\mathbb{E}(x) = \frac{\alpha}{\alpha+\beta}$
- Variance: $\mathbb{V}(x) = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$

Dirichlet distribution —
- X takes values $\in [0,1]$
- Multivariate extension of Beta distribution
- $Dir(\boldsymbol{x}|\boldsymbol{\alpha}) = \frac{1}{B(\boldsymbol{\alpha})}\prod_{k=1}^n u_k^{\alpha_k-1}$, where $B(\alpha)$ is the multivariate generalization of the Beta function: $B(\alpha) = \frac{\prod_{k=1}^n \Gamma(\alpha_k)}{\Gamma(\sum_{k=1}^n \alpha_k)}$

Uniform distribution —
- Assume x is uniformly distributed between $[a,b]$
- PDF: $f(x) = \begin{cases} \frac{1}{b-a} & a \leq x \leq b \\ 0 & \text{otherwise} \end{cases}$
- CDF: $F(x) = \begin{cases} 0 & x < a \\ \frac{x-a}{b-a} & a \leq x \leq b \\ 1 & x > b \end{cases}$

Exponential families — Family of probability distributions (incl. Gaussian, Poisson, Bernoulli, Categorical, Gamma, Beta) of the form: $p(x \mid \theta) = \frac{1}{Z(\theta)}\boldsymbol{h}(x)\exp(\theta \cdot \boldsymbol{\Phi}(x))$ where
- $Z(\theta)$ is a *partition function* which acts as a normalization constant to $[0,1]$
- $h(x)$ determines the *support* of the function
- θ are the *parameters*
- $\boldsymbol{\Phi}(x)$ are the *sufficient statistics*

Properties:
- Finite sufficient statistics
- *Conjugate priors*:
  - Prior distribution p for a likelihood distribution q such that the posterior distribution induced by p and q is of the same form as p
  - This helps to conduct Bayesian machine learning: Posterior parameters can be updated using prior parameters and do not have to be re-derived
  - Example: Bernoulli distribution with Beta prior:
    * Assume prior $P(\theta) \sim \text{Beta}(\alpha,\beta)$
    * Likelihood $P(X \mid \theta) = \theta^k(1-\theta)^{n-k}$, where n = trials and k = successes
    * Posterior $P(\theta \mid X) \propto P(X \mid \theta)P(\theta) \propto \theta^k(1-\theta)^{n-k}\theta^{\alpha-1}(1-\theta)^{\beta-1} \propto \theta^{k+\alpha-1}(1-\theta)^{n-k+\beta-1}$
    * Due to conjugate prior, posterior $\sim \text{Beta}(\alpha',\beta')$, with $\alpha' = k + \alpha$ and $\beta' = n - k + \beta$
- Corresponds to maximum entropy distributions, if sufficient statistics match training data (see below)

Derivation from maximum entropy:
- Maximum entropy principle: When estimating a probability distribution over some constraints (e.g. known E), we should choose the distribution that has the highest entropy
- Entropy for discrete x: $H(x) = -\sum_{x\in\mathcal{X}} p(x)\log p(x)$
- Assume we know $\mathbb{E}(\boldsymbol{\Phi}(x)) = F$
- Using the maximum entropy principle, we want to find $J(p) = \sum_{x\in\mathcal{X}} p(x)\log p(x)$ subject to
  - Non-negativity constraint: $p(x) \geq 0 \quad \forall x$
  - Sum-to-one constraint: $\sum_{x\in\mathcal{X}} p(x) = 1$
  - User-defined constraint: $\mathbb{E}(\boldsymbol{\Phi}(x)) = \sum_{x\in\mathcal{X}} p(x)\boldsymbol{\Phi}(x) = F$
- This is a constrained optimization problem with the objective function: $J(p,\lambda) = -\sum_{x\in\mathcal{X}} p(x)\log p(x) - \lambda_0(1 - \sum_{x\in\mathcal{X}} p(x)) + \sum_k \lambda_k(F_k - \sum_{x\in\mathcal{X}} p(x)\Phi_k(x))$
- Then, the gradient is given by: $\frac{\partial J(p,\lambda)}{\partial p(x)} = -1 - \log(p(x)) - \lambda_0 - \sum_k \lambda_k\Phi_k(x) = 0$
- $\Rightarrow \log(p(x)) = -1 - \lambda_0 - \sum_k \lambda_k\Phi_k(x) = \log(\frac{1}{Z}\exp(-\sum_k \lambda_k\Phi_k(x)))$ where $Z = \exp(1 + \lambda_0)$
- $\Rightarrow p(x) = \frac{1}{Z}\exp(-\sum_k \lambda_k\Phi_k(x))$
- $\Rightarrow \sum_{x\in\mathcal{X}} p(x) = \sum_{x\in\mathcal{X}} \frac{1}{Z}\exp(-\sum_k \lambda_k\Phi_k(x)) = 1$ by sum-to-one constraint
- $\Rightarrow Z = \sum_{x\in\mathcal{X}} \exp(-\sum_k \lambda_k\Phi_k(x))$
- If we plug this Z back into $p(x)$, we get: $p(x) = \frac{\exp(-\sum_k \lambda_k\Phi_k(x))}{\sum_{x\in\mathcal{X}} \exp(-\sum_k \lambda_k\Phi_k(x))} = \frac{1}{\sum_{x\in\mathcal{X}} \exp(-\boldsymbol{\lambda}^\intercal\boldsymbol{\Phi}(x))}\exp(-\boldsymbol{\lambda}^\intercal\boldsymbol{\Phi}(x))$
- Then, we see that this is in the exponential family

Proof that Gaussian is an exponential distribution:
- Gaussian PDF: $f(x \mid \mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$
- $= \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2-2\mu x+\mu^2}{2\sigma^2}}$
- $= \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}+\frac{2\mu x}{2\sigma^2}-\frac{\mu^2}{2\sigma^2}}$
- $= \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}} e^{\frac{\mu x}{\sigma^2}} e^{-\frac{\mu^2}{2\sigma^2}}$
- $= \frac{e^{-\frac{\mu^2}{2\sigma^2}}}{\sqrt{2\pi\sigma^2}} \times 1 \times e^{\frac{\mu x}{\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$
- $= \frac{e^{-\frac{\mu^2}{2\sigma^2}}}{\sqrt{2\pi\sigma^2}} \times 1 \times e^{\frac{\mu}{\sigma^2}x - \frac{1}{2\sigma^2}x^2}$
- $= \frac{e^{-\frac{\mu^2}{2\sigma^2}}}{\sqrt{2\pi\sigma^2}} \times 1 \times e^{[\frac{\mu}{\sigma^2},\frac{-1}{2\sigma^2}]\cdot[x,x^2]}$
- Thus, parameters of exponential family given by:
  - $\frac{1}{Z(\theta)} = \frac{e^{-\frac{\mu^2}{2\sigma^2}}}{\sqrt{2\pi\sigma^2}}$
    $\Rightarrow Z(\theta) = \sqrt{2\pi\sigma^2} \times \frac{1}{e^{-\frac{\mu^2}{2\sigma^2}}} = \sqrt{2\pi\sigma^2}e^{\frac{\mu^2}{2\sigma^2}}$
  - $\boldsymbol{h}(x) = 1$
  - $\boldsymbol{\theta} = \begin{bmatrix} \mu/\sigma^2 \\ -1/2\sigma^2 \end{bmatrix}$
  - $\boldsymbol{\Phi}(x) = \begin{bmatrix} x \\ x^2 \end{bmatrix}$

Proof that Poisson is an exponential distribution:
- Poisson PDF: $f(x \mid \lambda) = \frac{\lambda^x e^{-\lambda}}{x!}$
- $= \frac{1}{x!}e^{-\lambda}\lambda^x$
- $= \frac{1}{e^\lambda}\frac{1}{x!}e^{log(\lambda^x)}$
- Thus, parameters of exponential family given by:
  - $\frac{1}{Z(\theta)} = \frac{1}{e^\lambda} \Rightarrow Z(\boldsymbol{\theta}) = e^\lambda$
  - $\boldsymbol{h}(x) = \frac{1}{x!}$
  - $\boldsymbol{\theta} = [log(\lambda)]$

- $\Phi(x) = [x]$

Proof that Beta is an exponential distribution:

- Beta PDF: $f(x \mid \alpha, \beta) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha,\beta)}$
- $= \frac{1}{B(\alpha,\beta)} \times 1 \times e^{(\alpha-1)\ln(x)+(\beta-1)\ln(1-x)}$
- $= \frac{1}{B(\alpha,\beta)} \times 1 \times e^{[(\alpha-1),(\beta-1)]\cdot[\ln(x),\ln(1-x)]}$
- Thus, parameters of exponential family given by:
  - $\frac{1}{Z(\alpha,\beta)} = \frac{1}{B(\alpha,\beta)} \Rightarrow Z(\alpha,\beta) = B(\alpha,\beta)$ $\Rightarrow Z(\theta) = B(\theta_1+1, \theta_2+1)$
  - $h(x) = 1$
  - $\theta = \begin{bmatrix}(\alpha-1)\\(\beta-1)\end{bmatrix}$
  - $\Phi(x) = \begin{bmatrix}\ln(x)\\\ln(1-x)\end{bmatrix}$

Proof that Chi Square is an exponential distribution:

- Chi Square PDF: $f(x \mid k) = \frac{x^{k/2-1}e^{-x/2}}{2^{k/2}\Gamma(k/2)}$
- $= \frac{1}{2^{k/2}\Gamma(k/2)} e^{-x/2} e^{(k/2-1)\ln(x)}$
- Thus, parameters of exponential family given by:
  - $\frac{1}{Z(k)} = \frac{1}{2^{k/2}\Gamma(k/2)} \Rightarrow Z(k) = 2^{k/2}\Gamma(k/2)$ $\Rightarrow Z(\theta) = 2^{\theta+1}\Gamma(\theta+1)$
  - $h(x) = e^{-x/2}$
  - $\theta = [k/2-1]$
  - $\Phi(x) = [\ln(x)]$

Proof that Binomial is an exponential distribution:

- Binomial PDF: $f(x \mid n, \pi) = \binom{n}{x}\pi^x(1-\pi)^{n-x}$
- $= \binom{n}{x} e^{x\ln(\pi)} e^{(n-x)\ln(1-\pi)}$
- $= \binom{n}{x} e^{x\ln(\pi)-x\ln(1-\pi)+n\ln(1-\pi)}$
- $= \binom{n}{x} e^{x\ln(\frac{\pi}{1-\pi})} e^{n\ln(1-\pi)}$
- $= e^{n\ln(1-\pi)} \binom{n}{x} e^{\ln(\frac{\pi}{1-\pi})x}$
- Thus, parameters of exponential family given by:
  - $\frac{1}{Z(\pi)} = e^{n\ln(1-\pi)} \Rightarrow Z(\pi) = \frac{1}{e^{n\ln(1-\pi)}}$ $\Rightarrow Z(\theta) = e^{n\ln(1+e^{\theta})}$
  - $h(x) = \binom{n}{x}$
  - $\theta = \left[\ln(\frac{\pi}{1-\pi})\right]$
  - $\Phi(x) = [x]$

Other Concepts

Laws of large numbers — Sample mean of iid variables converges to population mean as $n \to \infty$

Jensen's inequality — Relates expected value of a convex function of a random variable to the convex function of the expected value of that random variable $\mathbb{E}(f(\mathcal{X})) \geq f(\mathbb{E}(\mathcal{X}))$

Markov's inequality — $p(x \geq t) \leq \frac{\mathbb{E}(x)}{t}$

Interesting only for $t \geq \mathbb{E}(x)$ because $p(x \geq t)$ must then be less than or equal to 1

Generalizations:

- $p(|x| \geq t) \leq \frac{\mathbb{E}(g(|x|))}{g(t)}$
- $p(|x| \geq t) \leq \frac{\mathbb{E}(|x|^n)}{t^n}$

Chebychev's inequality — $p(|x-\mu_x| \geq \alpha|\sigma_x|) \leq \frac{1}{\alpha^2}$

Interesting only for $\alpha > 1$

Implications:

- For n variables: $p(|S_n - \mu_x| \geq \epsilon) \leq \frac{\sigma_x^2}{n\epsilon^2}$ where S_n is the sample mean

Sufficient statistics —

- $Z = g(Y)$ is a sufficient statistic for estimating X if X can be estimated as well from Z as from Y, i.e. condensing Y to Z does not entail any loss of information about X
- Conditioned on Z, Y is independent of X: $p(Y|Z,X) = p(Y|Z)$
- For sufficient statistics, the MLE of X from Y is the same as the MLE of X from Z: $argmag_x p(Y|X)\rho(y) = argmag_x p(Z|X)\rho(y)$
- $p(X|Z) = p(X|Y)$

Hypothesis Testing

Taxonomy —

- Parametric tests: Test-statistic follows a specific distribution, whose parameters are a function of the sample data
- Non-parametric tests: Test-statistic follows a specific distribution, whose parameters are not a function of the sample data, or we do not many any assumptions on which distribution the test statistic follows

Terminology —

- Hypothesis:
  - H_0: Accepted null hypothesis, e.g. $p = p_0$, $p_1 - p_2 = p_{0,1} - p_{0,2} = 0$
  - H_A: Alternative hypothesis, e.g. $p \neq p_0$, $p_1 - p_2 \neq p_{0,1} - p_{0,2} \neq 0$
- Errors:
  - True positive: Chose H_0, and H_0 obtains
  - False negative, type I error: Chose H_A, but H_0 obtains
  - True negative: Chose H_A, and H_A obtains
  - False positive, type II error: Chose H_0, but H_A obtains
- Significance level α:
  - $\alpha \geq p(\text{type I error})$ with equality for continuous variables
  - If α is small, the probability that we are erroneously rejecting H_0 is very small
  - Set by us, typically at 5%
- Critical value z:
  - For two-sided: $z_{\alpha/2}$, $z_{1-\alpha/2}$
  - For one-sided upper tail: $z_{1-\alpha}$
  - For one-sided lower tail: z_α
  - Associated z-score with α
- P-value p:
  - For two-sided: $p = P(|z| \geq z_n)$
  - For one-sided upper tail: $p = P(z \geq z_n)$
  - For one-sided lower tail: $p = P(z \leq z_n)$
  - Probability, given H_0 that we observe a value as or more extreme as the observed value z_n
  - Smallest significance level resp. largest confidence level, at which we can reject H_0 given the sample observed
  - If p-value is less than significance level resp. if observed value is more extreme than critical value, reject H_0, because the probability that we are erroneously doing so is very small
- Confidence level: $1-\alpha$, probability, given H_0, that we retain H_0
- Beta: $\beta = p(\text{type II error})$
- Power: $1-\beta$, probability, given H_A, that we reject H_0
- Test types:
  - Two-sided: $H_0 : p = p_0, H_A : p \neq p_0$
  - One-sided upper tail: $H_0 : p \leq p_0, H_A : p > p_0$
  - One-sided lower tail: $H_0 : p \geq p_0, H_A : p < p_0$

Corrections for multiple test — Bonferroni correction: New significance level set to $\alpha* = \alpha/K$ TBA

Sample size calculation — TBA

Neyman Pearson test —

- Maximizes power while controlling type I errors
- Sets α such that $\alpha \geq p(\text{type I error})$
- Then minimizes $p(\text{type II error})$
- This is achieved by a likelihood-ratio test with threshold θ, such that α equals or is as close as possible to $p(\text{type I error})$:
  - If $\Lambda(x) = \frac{p(x|p_0)}{p(x|p_A)} > \theta$, we reject H_0
  - Then, we have $P(\Lambda(x) > \theta|H_0) = P(\frac{p(x|p_0)}{p(x|p_A)} > \theta|H_0) = P(\text{type I error}) = \alpha$
  - The smaller α, the larger θ

Bayesian Hypothesis Testing —

- If $\Lambda(x) = \frac{p(x|p_0)}{p(x|p_A)} > \theta$, we reject H_0
- $\theta = \frac{k(p_A,p_0)P(p_0)}{k(p_0,p_A)P(p_A)}$
- In this case, θ subsumes both the prior $p(x)$ and the costs $k(\hat{x},x)$

Parametric hypothesis tests (Z,T,F) — 5x2 cross-validation paired t-test:

- Compares the performance of two classifiers
- Split data in 2 folds, train each classifier on one fold and test on the other, then swap the folds and repeat
- Repeat this process for 5 rounds
- For both of the 2 folds, record the performance difference between the two classifiers: $d_i = p_{1,i} - p_{2,i}$
- The mean performance for each of the $1,...,5$ rounds is $\overline{d} = \frac{d_1+d_2}{2}$, the variance is $s^2 = \frac{(d_1-\overline{d})^2+(d_2-\overline{d})^2}{2}$
- Test statistic: $t = -\frac{\sum_{i=1}^5 \overline{d}_i}{\sqrt{\sum_{i=1}^5 s_i^2}}$
- Test statistic is compared to the critical value from the t-distribution with 5 degrees of freedom
- If p-value $< \alpha$, reject H_0

TBA

Non-parametric hypothesis tests —

McNemar's test:

- Compares the disagreement probability of two classifiers
- In 2×2 contingency table (correct vs. incorrect predictions for each classifier) focuses on the off-diagonal elements b and c because these represent the cases where classifiers differ in outcomes, with one being correct, and the other incorrect
- $H_0 : p_b = p_c$ $H_A : p_b \neq p_c$
- Test statistic: $\mathcal{X}^2 = \frac{(b-c)^2}{b+c}$
- Test statistic is compared to the critical value from the chi-square distribution table with 1 degree of freedom
- If p-value $< \alpha$, reject H_0

Permutation test:

- Tests whether a classifier's performance is significantly better than random chance
- H_0 : Performance is random, there is no true relationship between the features and the response H_A : Performance is not random
- Train classifier and record performance, then shuffle response labels while keeping the features unchanged (simulate H_0), re-train classifier and record performance, repeat
- Calculate the p-value, i.e. proportion of shuffled performance values that are greater than or equal to the initially observed performance
- If p-value $< \alpha$, reject H_0

TBA

Confidence intervals — TBA

4 Information Theory

Description

Entropy —

- $H(x) = -\sum_x p(x)log(p(x)) = -\sum_{x,y} p(x,y)log(p(x))$ resp. $H(x) = -\int p(x)log(p(x))dx$
- Measure of randomness in a variable resp. quantifies uncertainty of a distribution

Properties:

- $H(x) \geq 0$
- $H(x)$ is maximized, when x is a uniform random variable
- For independent variables: $H(x,y) = H(x) + H(y)$

Conditional entropy —

- $H(x|y) = -\sum_{x,y} p(y)p(x|y)log(p(x|y)) = -\sum_{x,y} p(x,y)log(\frac{p(x,y)}{p(y)})$
- Measure of how much information of x is revealed by y

Properties:

- $0 \leq H(x|y) \leq H(x)$ with equality if when x is independent with y resp. if y completely determines x

Mutual information —

- $I(x;y) = H(x) - H(x|y) = -\sum_{x,y} p(x,y)log(\frac{p(x)p(y)}{p(x,y)})$
- Measure of how much information of x is left after y is revealed

Properties:

- $0 \leq I(x;y) \leq H(x)$ with equality if y completely determines x resp. if x is independent with y

KL divergence —

- $KL(p;q) = \sum_x p(x)log(\frac{p(x)}{q(x)})$
- Measures the extra information or inefficiency when approximating a true distribution over x, p, with a predicted one, q

Properties:

- $KL(p;q) \geq 0$

Cross entropy —

- $CE(p|q) = KL(p;q) + H(p) = -\sum_x p(x)log(q(x))$
- Measures the total uncertainty when using the predicted distribution q to represent the true distribution p, combining both the model's error and the intrinsic uncertainty of the true distribution

Properties:

- $KL(p;q) \geq 0$

5 ML Set-Up

Formalization

Data —

- Features $\mathcal{X} \in \mathbb{R}^m$
- Response $\mathcal{Y}$
- Training data $\mathcal{D}$ vs. test data

Representation —

- Model resp. function f:
  - Models relationship between features and response based on noisy training data
  - $f_\theta : \mathbb{R}^m \to \mathcal{Y}$
  - θ are parameters characterizing f which are optimized during training
  - f is constrained by hyperparameters which are tuned during validation
- Model resp. function class $\mathcal{F}$: Determines model resp. function structure

Loss resp. objective function —

- $\mathcal{L}(\theta) = \frac{1}{n}\sum_{i=1}^n \ell(y_i, \hat{y}_i)$
- Distinguishes good from bad model resp. function
- Training vs. test error: Test error empirically estimates the true error

Optimization —

- During training, algorithm selects function with parameters θ^* that yields optimal results, based on loss resp. objective function
- Search for optimal parameters is governed by hyperparameters
- Models can be selected for:
  - Prediction performance: Model with best performance based on loss resp. objective function
  - Inference: Model which best explains the underlying process generating the data

Evaluation metric —

- Can coincide with loss resp. objective function, but not always the case

Steps

TBA

6 ML Paradigms

Paradigms I

Frequentism

Description —

- Parametric approach
- θ as fixed, unknown quantity, X as random, and known quantity
- Makes point estimate
- Focuses on maximizing likelihood $p(X|\theta)$ to infer posterior $p(\theta|X)$

- Only requires differentiation methods
- High variance, but low bias

MLE estimator

- Maximizes log-likelihood: $\hat{\theta} = argmax_\theta(L) = argmax_\theta(\prod_{i=1}^n p(\boldsymbol{x_i}|\theta)) = argmax_\theta(\sum_{i=1}^n log(p(\boldsymbol{x_i}|\theta)))$
- *Score*:
  - The score is the derivative of the log-likelihood:
    $\Lambda = \frac{\partial}{\partial\theta} log(p(\boldsymbol{x}|\theta)) = \frac{\frac{\partial}{\partial\theta} p(\boldsymbol{x}|\theta)}{p(\boldsymbol{x}|\theta)}$
  - The expected score is given by:
    $\mathbb{E}(\Lambda) = \int p(\boldsymbol{x}|\theta) \frac{\frac{\partial}{\partial\theta} p(\boldsymbol{x}|\theta)}{p(\boldsymbol{x}|\theta)} dx =$
    $\frac{\partial}{\partial\theta} \int p(\boldsymbol{x}|\theta) dx = \frac{\partial}{\partial\theta} \times 1 = 0$
- Advantages
  - Consistent: $\hat{\theta} \to \theta$ as $n \to \infty$
  - Asymptotically normal: $\frac{1}{\sqrt{n}}(\hat{\theta} - \theta)$ coverges to $\mathcal{N}(0, \boldsymbol{J}^{-1}(\theta)\boldsymbol{I}(\theta)\boldsymbol{J}^{-1}(\theta))$ where $\boldsymbol{J} = -\mathbb{E}[\frac{\partial^2 log(p(\boldsymbol{x}|\theta))}{\partial\theta\partial\theta^\top}]$ and where $\boldsymbol{I}$ is the Fisher information
  - Asymptotically efficient: $\hat{\theta}$ minimizes $\mathbb{E}[(\hat{\theta} - \theta)^2]$ as $n \to \infty$
    - Not necessarily the best estimator, especially for small samples in a multivariate context
    - (cf. Rao-Cramer bound)
  - Equivariant: If $\hat{\theta}$ is MLE of θ, then $g(\hat{\theta})$ is MLE of $g(\theta)$
- Proofs of advantages
  - Asymptotically normal:
    - We start with the score and set it to 0 for optimization with regard to θ:
      $\Lambda = \frac{\partial}{\partial\theta} log(p(\boldsymbol{x}|\theta)) = 0$
    - With a Taylor expansion, we can show that $(\hat{\theta} - \theta)\sqrt{n} =$
      $\frac{1}{\sqrt{n}}\Lambda[-\frac{1}{n}\frac{\partial^2}{\partial\theta\partial\theta^\top}\sum_{i=1}^n log(p(x_i|\theta))]^{-1}$
      where Λ is the score
    - We set
      $\boldsymbol{J} = [-\frac{1}{n}\frac{\partial^2}{\partial\theta\partial\theta^\top}\sum_{i=1}^n log(p(x_i|\theta))]$
    - $\frac{1}{\sqrt{n}}\Lambda$ is a random vector with covariance matrix $\boldsymbol{I}$ and converges to the normal distribution $\sim \mathcal{N}(0, \boldsymbol{I})$
    - Then,
      $(\hat{\theta} - \theta)\sqrt{n} = \boldsymbol{J}^{-1}\frac{1}{\sqrt{n}}\Lambda \sim \boldsymbol{J}^{-1}\mathcal{N}(0, \boldsymbol{I})$
    - $\mathbb{V}(\boldsymbol{J}^{-1}\frac{1}{\sqrt{n}}\Lambda) = \mathbb{E}[\boldsymbol{J}^{-1}\boldsymbol{I}\boldsymbol{J}^{-1}]$
    - This equality is given because $\mathbb{V}(x) = \mathbb{E}[x - \mathbb{E}(x)] = \mathbb{E}[x]$ if $\mathbb{E}(x) = 0$, which is the case here, given that the expected score is 0
    - So we have shown that
      $\hat{\theta} - \theta)\sqrt{n} = \boldsymbol{J}^{-1}\frac{1}{\sqrt{n}}\Lambda \sim \mathcal{N}(0, \boldsymbol{J}^{-1}\boldsymbol{I}\boldsymbol{J}^{-1})$
  - Equivariant:
    - Let $t = g(\theta)$ and $h = g^{-1}$
    - Then, $\theta = h(t) = h(g(\theta))$
    - For all t we have: $L(t) =$
      $\prod_i p(\boldsymbol{x^{(i)}}|h(t)) = p(\boldsymbol{x^{(i)}}|\theta) = L(\theta)$
    - Hence, for all t we can say:
      $L(t) = L(\theta)$ and $L(\hat{t}) = L(\hat{\theta})$

Probably Approximately Correct (PAC) estimator

- Generates probabilistic bounds for parameter θ that is approximately known with a high probability:
  - Probability of being correct: $1 - \delta$
  - Degree of approximation: ϵ

Setting:

- *Hypothesis class* $\mathcal{H}$: Set of functions that can be expressed by the algorithm
- *Concept class* $\mathcal{C}$: Set of possible target functions that represent true mappings from input to output
- *Specific concept* c:
  - True function c that maps inputs to outputs
  - Estimated function $\hat{c}$
- *Learning algorithm* $\mathcal{A}$:
  - Receives samples $\mathcal{Z} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ as inputs
  - $c(x_i) = y_i$ for all i
  - $\mathcal{A}$ outputs $\hat{c} \in \mathcal{H}$

PAC learning model:

- $\mathcal{A}$ can learn c from $\mathcal{C}$ if, given a sufficiently large sample, it outputs $\hat{c}$ that generalizes well with high probability resp.
  if given $\mathcal{Z}$ of size $n > \text{poly}\left(1/\epsilon, 1/\delta, \text{size}(c)\right)$, it outputs $\hat{c}$ such that: $P(\mathcal{R}(\hat{c}) \leq \epsilon) \geq 1 - \delta$ where:
  - ϵ: Error tolerance (how much $\hat{c}$ deviates from c), is between 0 and 0.5
  - δ: Confidence (how likely $\hat{c}$ generalizes well), is between 0 and 0.5
  - $\text{size}(c)$: Complexity of the concept
- A concept class $\mathcal{C}$ is *PAC-learnable* from a hypothesis class $\mathcal{H}$ if there is an algorithm $\mathcal{A}$ that can learn any concept in $\mathcal{C}$
- If algorithm $\mathcal{A}$ runs in time polynomial in $1/\epsilon$ and $1/\delta$, then $\mathcal{C}$ is *efficiently PAC-learnable*
- *Strong PAC learning*: Demand arbitrarily small error ϵ with high probability $1 - \delta$
- *Weak PAC learning*: Demand that risk is bounded for large (not trivial) error ϵ, used frequently in ensemble learning

Scenario 1: If $\mathcal{C}$ is finite and $\mathcal{H} = \mathcal{C}$:
Deterministic scenario:

- In this case, algorithm $\mathcal{A}$ returns a consistent hypothesis $\hat{c}$, i.e. $\mathcal{R}(\hat{c}) = 0$
- We aim to bound the risk of $\hat{c}$ as follows: $P(\mathcal{R}(\hat{c}) > \epsilon) \leq \delta$
- According to *Hoeffding's inequality* for a single hypothesis:
  $P(|\mathcal{R}(c) - \mathcal{R}(\hat{c})| > \epsilon) = P(\mathcal{R}(c) > \epsilon) \leq \exp(-n\epsilon)$
- To account for all hypotheses in $\mathcal{C}$, we apply a *union bound*:
  $P(\exists c \in \mathcal{C} : \mathcal{R}(c) > \epsilon) \leq |\mathcal{C}| \times \exp(-n\epsilon)$
- We then get: $|\mathcal{C}| \times \exp(-n\epsilon) \leq \delta$
- From this, we can derive:
  $\log(|\mathcal{C}|) - n\epsilon \leq \log(\delta)$
  $\log(|\mathcal{C}|) - \log(\delta) \leq n\epsilon$
  $\frac{1}{\epsilon}\left[\log(|\mathcal{C}|) + \log\left(\frac{1}{\delta}\right)\right] \leq n$
- Then, we have derived a minimum required sample size to bound the risk of $\hat{c}$ as follows: $P(\mathcal{R}(\hat{c}) > \epsilon) \leq \delta$

Stochastic scenario:

- If the instance's true label is not deterministic, but modeled by a distribution $\mathcal{D}$
- *Bayes optimal classifier:*
  - Classifier that achieves minimum possible error
  - Outputs labels based on true probabilities of labels given input features: $\hat{y} = argmax_y p(y \mid \boldsymbol{x})$
  - If the Bayes optimal classifier is not in $\mathcal{C}$, there will always be a gap between the error of the best hypothesis $\hat{c}$ and Bayes optimal error
  - We then aim to bound the risk of $\hat{c}$ as follows: $P\left(\mathcal{R}(\hat{c}) - \inf_{c \in \mathcal{C}} \mathcal{R}(c) \leq \epsilon\right) \geq 1 - \delta$

Scenario 2: If $\mathcal{C}$ is finite and $\mathcal{H} \neq \mathcal{C}$:

- According to *Hoeffding's inequality* for a single hypothesis:
  $P\left(\mathcal{R}(\hat{c}^*) - \inf_{c \in \mathcal{C}} \mathcal{R}(c) > \epsilon\right) \leq 2\exp(-n\epsilon^2)$
- To account for all hypotheses in $\mathcal{C}$, we apply a *union bound*: $P\left(\exists c \in \mathcal{C} :\right.$
  $\left.\mathcal{R}(\hat{c}^*) - \inf_{c \in \mathcal{C}} \mathcal{R}(c) > \epsilon\right) \leq 2|\mathcal{C}|\exp(-n\epsilon^2)$

Scenario 3: If $\mathcal{C}$ has finite VC-dimension, but is infinite:

- The *VC-dimension* of a concept class $\mathcal{C}$ is a measure of its complexity: VC-dimension V_c is the size of the largest set A that can be shattered by $\mathcal{C}$
  - A set of instances A is *shattered* by the concept class $\mathcal{C}$ if, for every subset $S \subseteq A$, there is a concept $c_S \in \mathcal{C}$ such that $S = c_S \cap A$
  - This means the concept class can "realizeör "perfectly labelëvery possible labeling of A
  - E.g. for 2^n points, there are 2^n possible ways to assign binary labels to the points. If $\mathcal{C}$ can express all 2^n labelings, it shatters A
- If $V_c > 2$: $P\left(\mathcal{R}(\hat{c}_n^*) - \inf_{c \in \mathcal{C}} \mathcal{R}(c) > \epsilon\right) \leq$
  $g n^{V_c} \exp\left(-\frac{n\epsilon^2}{32}\right)$
- If $\mathcal{C}$ has a finite VC-dimension:
  $P\left(\mathcal{R}(\hat{c}_n^*) - \inf_{c \in \mathcal{C}} \mathcal{R}(c) > \epsilon\right) \leq$
  $g n^{V_c} \exp\left(-\frac{n\epsilon^2}{32}\right) \to 0$ as $n \to \infty$

Bayesianism

Description —

- Parametric approach
- θ as random, unknown quantity, X as random, and known quantity
- Makes estimate in form of distribution
- Leverages prior and likelihood to infer posterior: $p(\theta|\boldsymbol{X}, \boldsymbol{y}) = \frac{p(\theta)p(\boldsymbol{y}|\boldsymbol{X}, \theta)}{p(\boldsymbol{y}|\boldsymbol{X})} =$
  $\frac{p(\theta)p(\boldsymbol{y}|\boldsymbol{X}, \theta)}{\int p(\theta)p(\boldsymbol{y}|\boldsymbol{X}, \theta)d\theta} \propto p(\theta)p(\boldsymbol{y}|\boldsymbol{X}, \theta) = p(\theta, \boldsymbol{y}|\boldsymbol{X})$
- Focuses on minimizing cost function
  $\mathbb{E}[k(\theta', \Theta)|\boldsymbol{X}, \boldsymbol{y}] = \int_\theta p(\theta|\boldsymbol{X}, \boldsymbol{y}) \times k(\theta', \theta)d\theta \propto$
  $\int_\theta p(\theta, \boldsymbol{y}|\boldsymbol{X}) \times k(\theta', \theta)d\theta$ resp.
  $\sum p(\theta|\boldsymbol{X}, \boldsymbol{y}) \times k(\theta', \theta)$
- Requires integration methods for normalizing constant in denominator, which can be intractable, in which case MAP estimator can provide an alternative
- Low variance, but high bias

Mean estimator

- Minimizes mean squared error as cost function $k(\hat{\theta}, \theta) = |\hat{\theta} - \theta|^2$
- The resulting estimate is the mean of the posterior: $\hat{\theta} = \mathbb{E}[\theta|\boldsymbol{X}, \boldsymbol{y}]$
- Returns estimate that reflects central tendency and overall uncertainty

MAP estimator

- Generally, maximizes posterior: $\hat{\theta} = argmax_\theta(p(\theta|\boldsymbol{X}))$ resp. $p(\theta|\boldsymbol{X})p(\boldsymbol{X})$
- In discrete cases, MAP minimizes following cost function:
  $$k(\hat{\theta}, \theta) = \begin{cases} 1 & \hat{\theta} \neq \theta \\ 0 & \hat{\theta} = \theta \end{cases}$$
- Returns single point estimate

Statistical Learning

Description —

- We want to minimize expected risk $\mathcal{R}(f) = \mathbb{E}_{X,Y}[1f(X) \neq Y]$, but this is difficult because
  - We don't have access to the joint distribution of X, Y
  - We cannot find f, without any assumptions on its structure
  - It's unclear how to minimize the expected value
- Therefore, we make following choices:
  - We collect sample Z
  - We restrict space of possible choices of f to a set $\mathcal{H}$
  - We use a loss function to approximate the expected value
- With these choices, we approximate the expected risk via the empirical risk
  $\hat{\mathcal{R}}(f) = \hat{L}(Z, f) = \frac{1}{n}\sum_i L(y_i, f(x_i))$

Paradigms II

Probabilistic

Probabilistic inference by learning about posterior *Generative* —

- Learn about posterior indirectly via prior and likelihood
- Requires to learn more parameters vs. discriminative approach
- E.g. N-gram models, Markov random fields, RNNs

Discriminative —

- Learn about posterior directly
- Requires to learn fewer parameters vs. generative approach
- E.g. conditional random fields, RNNs

Distribution-free

Learned —

- E.g. perceptron, SVM

Manually crafted —

- E.g. conetxt free grammars, linear indexed grammars

7 Model Taxonomy

Taxonomy

Supervised vs. unsupervised — TBA

Reinforcement learning — TBA

Transfer Learning

Transfer learning — TBA

Active Learning

Active learning —

- Assume:
  - Domain space $\mathcal{X}$
  - Sample space $S \subseteq \mathcal{X}$
  - Labeled data $D_{n-1}(x_i, y_i)_{i<n}$
  - Target space $\mathcal{A} \subseteq \mathcal{X}$
  - We estimate $y_x = f_x + \epsilon_x$
- We aim to find the next x_n that gives us the most information about f in $\mathcal{A}$
- Information gain can be quantified as maximizing the conditional mutual information between y_x and f:
  $IG[f_x; y_x|D_{n-1}] =$
  $H(y_x|D_{n-1}) - H(y_x|f_x, D_{n-1})$ where $H(y_x|D_{n-1})$ is the uncertainty about y_x before labeling x_n and $H(y_x|f_x, D_{n-1})$ is the uncertainty about y_x after labeling x_n. We want to minimize the latter, i.e. we want to maximize the delta between the former and the latter
- We pick $x_n = argmax_{x \in S} IG[f_x; y_x|D_{n-1}]$
- To find a closed-form solution, we assume that f is a Gaussian process with a known mean and kernel function:
  - $f \sim \mathcal{GP}(\mu, k)$
  - $\boldsymbol{f} = (f_{x_1}, f_{x_2}, \ldots) \sim \mathcal{N}(\boldsymbol{\mu}, \Sigma)$ where elements in mean vector are $\mu_i = \mu(x_i)$ and elements in covariance matrix are $\Sigma_{ij} = k(x_i, x_j)$
- Under this assumption, we can show that
  $IG[f_x; y_x|D_{n-1}] = \frac{1}{2}log(\frac{\mathbb{V}(y_x|D_{n-1})}{\mathbb{V}(y_x|f_x, D_{n-1})})$
  Proof:
  - Gaussian entropy of $a|B$ given by:
    $H(a|B) = -\int p(a|B)log(p(a|B))da =$
    $-\mathbb{E}[log\mathcal{N}(\mu, \sigma)] =$
    $-\mathbb{E}[log((2\pi\sigma^2)^{-1/2}exp(-\frac{(x-\mu)^2}{2\sigma^2}))] =$
    $\frac{1}{2}log(2\pi\sigma^2) + \frac{1}{2\sigma^2}\mathbb{E}[(x-\mu)^2] =$
    $\frac{1}{2}log(2\pi\sigma^2) + \frac{1}{2} \times 1 =$
    $\frac{1}{2}log(2\pi\sigma^2) + \frac{1}{2}log(e) = \frac{1}{2}log(2\pi e\sigma^2)$
  - Plugging this in, we get:
    $H(y_x|D_{n-1}) - H(y_x|f_x, D_{n-1}) =$
    $\frac{1}{2}log(2\pi e\mathbb{V}(y_x|D_{n-1})) -$
    $\frac{1}{2}log(2\pi e\mathbb{V}(y_x|f_x, D_{n-1})) =$
    $\frac{1}{2}log(\frac{\mathbb{V}(y_x|D_{n-1})}{\mathbb{V}(y_x|f_x, D_{n-1})})$

Safe Bayesian learning —

- Bayesian approach to active learning
- Assume:
  - We have stochastic process f^*
  - We can iteratively choose points $x_1, \ldots, x_{n-1} \in \mathcal{X}$ and observe $y_i = f^*(x_1), \ldots, y_{n-1} = f^*(x_{n-1})$
  - Points should lie in safe area S^* which is the set of $x \in \mathcal{X}$ such that another stochastic process $g^*(x) \geq 0$
  - For chosen points, we can also observe $z_i = g^*(x_1), \ldots, z_{n-1} = g^*(x_{n-1})$ which are measurements of confidence, indicating high confidence when above 0
- We aim to find estimates of sample space S and target space $\mathcal{A}$
- To do so, we fit a Gaussian process on

observed $\{(x_i, y_i)\}_{i<n}$ and $\{(x_i, z_i)\}_{i<n}$. Gaussian process over f and g induces two bounds respectively, which provide the 95% confidence interval of $\mathbb{E}[f(x)]$ resp. $\mathbb{E}[g(x)]$:

- Upper bound function $u_n^f(x)$ resp. $u_n^g(x)$
- Lower bound function $l_n^f(x)$ resp. $l_n^g(x)$

- Gaussian process over g allows to derive pessimistic and optimistic estimate of safe area:
  - Pessimistic: $S_n = \{x : l_n^g(x) \geq 0\}$
  - Optimistic: $\hat{S}_n = \{x : u_n^g(x) \geq 0\}$
- We then gather estimates, where upper bound of f lies above baseline set by maximum value of lower bound of f:
  $\mathcal{A}_n = \{x \in \hat{S}_n : u_n^f(x) \geq max_{x' \in S_n} l_n^f(x')\}$
- We can then perform active learning with sample space $S = S_n$ and target space $\mathcal{A} = \mathcal{A}_n$

Batch active learning —

- Variant of active learning
- Assume:
  - Domain space $\mathcal{X}$ and distribution P over $\mathcal{X}$
  - Oracle to unknown function $f : \mathcal{X} \to \mathcal{Y}$
  - Population set $\mathcal{X} = \{x_1, ..., x_m\} \subseteq \mathcal{X}$
  - Budget $b \leq m$
- We aim to find next batch of data points $L \subseteq \mathcal{X}$ subject to $|L| = b$ that gives us the most information
- Suppose we know $Z = \{(x, f(x)) : x \in L\}$
- 1-nearest-neighbor classifier $\hat{f}$ is fitted to Z
- Let $B_\delta(x) = \{x' \in \mathcal{X} : \|x - x'\| \leq \delta\}$ be the set of sufficiently close points to x
  - We consider $B_\delta(x)$ pure if f yields same results for all of $B_\delta(x)$
- Impurity of δ is given by $\hat{\pi}(\delta) = P(\{x \in \mathcal{X} : B_\delta(x) \text{ is not pure}\})$
- Let $C(L, S) = \bigcup_{x \in L} B_\delta(x)$ be the union of all sets B
  - $C = C_r \cup C_w = \{x \in C : \hat{f}(x) = f(x)\} \cup \{x \in C : \hat{f}(x) \neq f(x)\}$
- We have $C_w \subseteq \{x \in \mathcal{X} : B_\delta(x) \text{ is not pure}\}$. Then, $P(C_w) \leq \hat{\pi}(\delta)$
- $\{x : \hat{f}(x) \neq f(x)\} \subseteq C_w \cup C_r^C \subseteq \{x \in \mathcal{X} : B_\delta(x) \text{ is not pure}\} \cup C^C$
- Then, we have $\mathcal{R}(\hat{f}) = P(\hat{f}(x) \neq f(x) \leq \hat{\pi}(\delta) + 1 - P(C)$
- We need to choose L and δ such that $\mathcal{R}(\hat{f})$ is minimized
- We approach this by minimizing the upper bound, by picking δ and choosing C that maximizes $P(C)$:
  $argmax_{L \subseteq \mathcal{X}, |L| = b} P(\bigcup_{x \in L} B_\delta(x))$
- Two challenges:
  1. We don't know the distribution
  2. Problem is NP-hard
- We address 1) by using the empirical distribution induced by X. Then, we have:
  $argmax_{L \subseteq \mathcal{X}, |L| = b} \frac{1}{|X|} |\{x' : \|x' - x\| \leq \delta, \text{ for some } x \in L\}|$
- We address 2) with greedy algorithm:
  - Input: $x \subseteq \mathcal{X}, b \in \mathbb{N}$
  - Output: $L \subseteq X$ of size b
  1. $G = (x, E)$ where $E = \{(x, x') : \|x - x'\| \leq \delta\}$
  2. $L = \emptyset$
  3. For $i = 1, ..., b$:
     (a) $\hat{x} \leftarrow argmax_{x \in \mathcal{X}} |\{x' : (x, x') \in E, x \in \mathcal{X}\}|$
     (b) $L \leftarrow L \cup \hat{x}$
     (c) $E \leftarrow E - (\{\hat{x}\} \times (B_\delta(\hat{x}) \cap x))$
  4. Return L

Ensemble Methods

Motivation —

- Let $\hat{f}_1(x), ..., \hat{f}_B(x)$ be estimators
- When averaging estimators...
  - Average remains unbiased, if all estimators are unbiased
    Proof:
    Bias $= \mathbb{E}[\hat{f}(x)] - \mathbb{E}[y|x] = \frac{1}{B}\sum_{i=1}^B \mathbb{E}[\hat{f}_i(x)] - \mathbb{E}[y|x] = \frac{1}{B}\sum_{i=1}^B \text{bias}(\hat{f}_i(x))$
  - Variance is reduced by a factor of $\frac{1}{B}$, if the estimators have similar variance and no covariance
    Proof:
    - Variance $= \mathbb{E}[\hat{f}(x) - \mathbb{E}[\hat{f}(x)]]^2 = \mathbb{E}[\frac{1}{B}\sum_{i=1}^B \hat{f}_i(x) - \frac{1}{B}\sum_{i=1}^B \mathbb{E}[\hat{f}_i(x)]]^2 = \mathbb{E}[\frac{1}{B}\sum_{i=1}^B (\hat{f}_i(x) - \mathbb{E}[\hat{f}_i(x)])]^2 = \frac{1}{B^2}\sum_{i=1}^B \mathbb{V}[\hat{f}_i(x)] + \frac{1}{B^2}\sum\sum_{i \neq j=1}^B \text{Cov}[\hat{f}_i(x), \hat{f}_j(x)]$
    - Assuming variance are similar ($\approx \sigma^2$) and covariances are small (≈ 0), we get: Variance $= \frac{1}{B^2}\sum_{i=1}^B \sigma^2 = \frac{1}{B^2} B\sigma^2 = \frac{\sigma^2}{B}$

Requirements — Diversity of estimators, to reduce covariance. Achieved by:

- Different subsets of data for each estimator, e.g. via bootstrapping
- Different features for each estimators
- Decorrelating estimators during training

Variants —

- Regression: Average output of all estimators: $\hat{r}_B(x) = \frac{1}{B}\sum_{b=1}^B r_b(x))$
- Classification: Majority or weighted voting: $\hat{c}_B(x) = sgn(\sum_{b=1}^B \alpha_b c_b(x))$ with majority voting if $\alpha = 1$

Bootstrap Aggregating (Bagging) —

- Algorithm
  1. For $b = 1, ..., B$:
     (a) Hold out $\frac{1}{3}$ of sample
     (b) Construct $Z_b^* = b^{th}$ bootstrap sample from remaining $\frac{2}{3}$ of sample
     (c) Construct estimator f_b based on Z_b^*
  2. Return $\hat{f}_B(x) =$ weighted average of $f_1(x), ..., f_B(x)$
  3. Calculate out-of-bag error on held out sample

If desired: Estimators can be constructed in multiple function classes $f', f'', ...$ and set of estimators in the function class, which generates the lowest empirical error, is returned

Random Forest —

- Algorithm:
  1. Generate multiple training sets via bootstrapping
  2. Construct multiple decision trees based on the generated training sets, where each tree selects a random set of features at each split via bootstrapping
     - Classification: Choose $m = \sqrt{k}$ predictors at each split
     - Regression: Choose $m = \frac{k}{3}$ predictors at each split
  3. Generate prediction by averaging or voting on trees
  4. Calculate out-of-bag error
- 2 sources of randomness:
  - Each tree trained on bootstrap sample of instances
  - At each split, bootstrap sample of features is considered

AdaBoost — Algorithm:

1. Each instance weight is initially $w_i = \frac{1}{n}$
2. Train first classifier $\hat{c}^{(1)}$ generating output $\hat{y}^{(1)}$
3. Calculate weighted error rate of j^{th} classifier: $r^{(j)} = \frac{\sum_{i=1}^n w_i \mathbb{I}_{\{\hat{y}_i^{(j)} \neq y_i\}}}{\sum_{i=1}^n w_i}$
4. Calculate classifier weight, which is higher, if the classifier has a lower error rate: $\alpha^{(j)} = \eta log(\frac{1 - r^{(j)}}{r^{(j)}})$
5. Update instance weights: For $i = 1, ..., n$:
$$w_i = \begin{cases} w_i & \hat{y}_i^{(j)} = y_i \\ w_i e^{\alpha^{(j)}} & \hat{y}_i^{(j)} \neq y_i \end{cases}$$
6. Normalize instance weights: $w_i = \frac{w_i}{\sum_{i=1}^n w_i}$
7. Continue by training next classifier
8. ...
9. Generate prediction:
   $\hat{y}(x) = argmax_k \sum_{j=1}^B \alpha_j \mathbb{I}_{\{\hat{c}^{(j)}(x) = k\}}$

AdaBoost is an additive logistic model that minimizes the exponential loss function:

- Exponential loss function: $\mathbb{E}[e^{-yF(x)}] = p(y = 1|x)e^{-F(x)} + p(y = -1|x)e^{F(x)}$
- Minimizer of exponential loss function is the log-odds: $\mathbb{E}[e^{-yF(x)}]$ is minimized at $\frac{\partial \mathbb{E}[e^{-yF(x)}]}{\partial F(x)} = \frac{1}{2} log(\frac{p(y=1|x)}{p(y=-1|x)}) = F(x) = 0$
- Then, $p(y = 1|x) = \frac{e^{F(x)}}{e^{-F(x)} + e^{F(x)}}$
- Log-odds minimizer corresponds to the AdaBoost minimizer, since AdaBoost models the final output as a sum of classifiers:
  $log(\frac{p(y=1|x)}{p(y=-1|x)}) = \sum_{j=1}^B \alpha_j \hat{c}^{(j)}(x) = F(x)$

AdaBoost as gradient descent:

- Discrete AdaBoost model can be viewed as performing gradient descent on an additive logistic model
- Cost function: Minimize $J(F) = \mathbb{E}[e^{-yF(x)}]$ where $F(x)$ is the combined classifier, consisting of multiple weak classifiers
- Suppose we consider an improved function $F'(x)$: $F'(x) = F(x) + \alpha c(x)$ where $c(x)$ is a new weak classifier
- To approximate $J(F')$ we can perform a Taylor expansion around 0:
  $J(F') = \mathbb{E}[e^{-y(F(x) + \alpha c(x))}] = \mathbb{E}[e^{-yF(x)}(1 - y\alpha c(x) + \frac{1}{2}\alpha^2)] + O(\alpha^3)$
- To minimize $J(F')$ wrt $c(x)$:
  - We define weighted expectation:
    - Weights: $w(x, y) = e^{-yF(x)}$
    - Weighted expectation of $g(x, y)$: $\mathbb{E}[g(x, y)|x] = \frac{\mathbb{E}[w(x,y)g(x,y)|x]}{\mathbb{E}[w(x,y)|x]}$
  - We aim to choose $c(x)$ that maximizes weighted expectation of $g(x, y) = yc(x)$ since this is equivalent to minimizing exponential loss
  - In $O(\alpha^2)$ this yields: $c(x) =$
    - $argmin_c = \mathbb{E}[1 - y\alpha c(x) + \frac{1}{2}\alpha^2|x]$
    - $argmax_c = \mathbb{E}[yc(x)|x]$
    - $= 1$ if $\mathbb{E}[y|x] = 1 \times p(y = 1|x) + (-1) \times p(y = -1|x) > 0$
    - $= -1$ otherwise
  - We can approximate the exponential via the quadratic loss. Then, we have:
    $\frac{1}{2}\mathbb{E}[(y - c(x))^2] - 1 = \mathbb{E}[y^2 - 2yc(x) + c(x)^2]\frac{1}{2} - 1 = \mathbb{E}[1 - 2yc(x) + 1]\frac{1}{2} - 1 = -\mathbb{E}[yc(x)]$
  - Then, we have:
    - $argmin_c = -\mathbb{E}[yc(x)|x]$
    - $argmax_c = \mathbb{E}[yc(x)|x]$
  - Minimizing the quadratic approximation leads to a Newton-like step for choosing $c(x)$, making it a weighted least squares choice of $c(x)$
- To minimize $J(F')$ wrt α:
  - $\alpha^* = argmin_\alpha \mathbb{E}[e^{-y\alpha c(x)}] = e^\alpha \mathbb{E}[\mathbb{I}_{\{y \neq c(x)\}}] + e^{-\alpha}\mathbb{E}[\mathbb{I}_{\{y = c(x)\}}] = \frac{1}{2} log(\frac{1 - err}{err})$ where $err = \mathbb{E}[\mathbb{I}_{\{y \neq c(x)\}}]$
- Combination yields AdaBoost update:
  - $F'(x) \leftarrow F(x) + \alpha^* c(x) = F(x) + \frac{1}{2} log(\frac{1-err}{err})$
  - $w(x, y) \leftarrow w(x, y) \times e^{-\alpha^* yc(x)} = w(x, y) \times e^{\alpha^*(2\mathbb{I}_{\{y \neq c(x)\}} - 1)} = w(x, y) \times e^{log(\frac{1-err}{err})\mathbb{I}_{\{y \neq c(x)\}}} \times e^{-\alpha^*}$ where $e^{-\alpha^*}$ is a constant

8 Feature Engineering

TBA

9 Model Optimization

Gradient Descent

Numeric optimization procedure

Gradient descent —

- Uses entire training set to evaluate whether new parameter is more optimal than previous one
- Slow and less likely to escape local minima due to randomness, but accurate
- Algorithm:
  1. Set $\eta > 0$
  2. Randomly initialize $\beta_{(t=0)}$
  3. $\beta_{(t+1)} \leftarrow \beta_{(t)} - \eta \nabla_\beta LO|_{\beta = \beta_{(t)}}$
  4. $t \leftarrow t + 1$
  5. Repeat 3 and 4 until $\nabla_\beta LO = 0$

Stochastic gradient descent —

- Uses only one training sample or mini-batch to evaluate whether new parameter is more optimal than previous one
- Fast and more likely to escape local minima due to randomness, but represents an approximation
- Algorithm:
  1. Set $\eta > 0$
  2. Randomly initialize $\beta_{(t=0)}$
  3. Shuffle training data and initialize $i \leftarrow 1$
  4. $\beta_{(t+1)} \leftarrow \beta_{(t)} - \eta \nabla_\beta LO$ for observation i $|_{\beta = \beta_{(t)}}$
  5. $t \leftarrow t + 1$
  6. $i \leftarrow i + 1$
  7. Repeat 4 to 6 until $i = n + 1$
  8. Repeat 2 to 6 until $\nabla_\beta LO = 0$
- Justification for SGD is given by *Robbins-Monro algorithm* which iteratively find the root (or zero) of an unknown function when only noisy observations of the function are available:
  - Algorithm:
    1. Choose learning rates $\eta_1, \eta_2, ...$, typically decreasing over time
    2. Randomly initialize $\beta_{(t=0)}$
    3. $\beta_{(t+1)} \leftarrow \beta_{(t)} - \eta \nabla_\beta LO|_{\beta = \beta_{(t)}}$ where LO is noisy
  - For convergence:
    - $\sum_{t=1}^\infty \eta_t = \infty$ to ensure sufficient exploration
    - $\sum_{t=1}^\infty \eta_t^2 \leq \infty$ to avoid overly large updates
    - Then, $lim_{t \to \infty} p(|y_t - y| > \epsilon) = 0$ for any $\epsilon > 0$

Hyperparameters —

- Learning rate η: Determines step size, if too small algorithm is slow to converge, if too large algorithm may diverge
- Batch size b: Number of samples from training set used to evaluate optimality of β at each step
- Epoch: Number of times model works through entire training set. Every epoch, β is updated n/b times

Modifications —

- Data should be standardized resp. scaled, otherwise the gradient of the largest predictor dominates the gradient of the loss function, leading to uneven updating of β and slow convergence
- A momentum term can be added to the updating function to ensure smooth updating of β: $\beta_{(t+1)} \leftarrow \beta_{(t)} - \eta \nabla_\beta LO|_{\beta = \beta_{(t)}} + \alpha(\beta_{(t)} - \beta_{(t)-1})$
- For stochastic gradient descent, a smoothing step can be added because stochastic gradient descent hovers around desired solution: $\hat{\beta}_{(t+1)} \leftarrow \frac{1}{L+1}\sum_{j=t-L}^t \beta_{(t)}$

10 Model Tuning

Hyperparameter tuning approaches

Nested cross-validation:

- Outer loop (model evaluation): Partition data $\mathcal{Z}$ into K equally sized disjoint subsets, of which 1 fold is testing set and $K-1$ folds are training set
- Inner loop (hyperparemeter tuning): Split outer training set into M equally sized disjoint subsets, of which 1 fold is validation set and $M-1$ folds are training set
- For each outer fold, first conduct inner loop
- In each inner loop, determine best hyperparameters by averaging performance for each hyperparameter set across all M folds
- Then train outer fold with best hyperparameters
- Compute final score by averaging performance on outer folds

TBA

11 Model Evaluation

Estimator Evaluation Criteria

Crtiteria —

- Consistency: $\hat{\theta} \to \theta$ as $n \to \infty$
- Bias: $\mathbb{E}(\hat{\theta}) - \theta$
  - Unbiased: $\mathbb{E}(\hat{\theta}) = \theta$
  - Asymptotically unbiased: $\mathbb{E}[(\hat{\theta}-\theta)^2] = 0$ as $n \to \infty$
  - Asymptotically efficient: $\mathbb{E}[(\hat{\theta}-\theta)^2] = I$ as $n \to \infty$ where I is Fisher information

Fisher information —

- $\boldsymbol{I} = \mathbb{E}[(\Lambda)^2] = \mathbb{E}[(\frac{\partial}{\partial\theta} log(p(\boldsymbol{x}|\theta)))^2] = \mathbb{V}(\frac{\partial log(p(\boldsymbol{x}|\theta))}{\partial\theta})$ where Λ is the score
- Equality is given because $\mathbb{V}(x) = \mathbb{E}[(x-\mathbb{E}(x))^2] = \mathbb{E}[x^2]$ if $\mathbb{E}(x) = 0$, which is the case here, given that the expected score is 0

Rao-Cramer bound —

- Shows that there does not exist an asymptotically unbiased parameter estimator
- For each unbiased estimator, $\mathbb{E}[(\hat{\theta}-\theta)^2] \geq \frac{1}{I}$ where I is the Fisher information
- For estimators in general, $\frac{(\frac{\partial}{\partial\theta}\text{bias}+1)^2}{I} + \text{bias}^2 \leq \mathbb{E}[(\hat{\theta}-\theta)^2]$, so there is a trade-off if the bias derivative is negative and the squared bias is positive, whereby a biased estimator may produce better results than an unbiased estimator

Proof:

- Given Cauchy Schwarz inequality, we can say: $\mathbb{E}[(\Lambda - \mathbb{E}((\Lambda))(\hat{\theta} - \mathbb{E}(\hat{\theta}))]^2 \leq \mathbb{E}[(\Lambda - \mathbb{E}((\Lambda))^2]\mathbb{E}[(\hat{\theta} - \mathbb{E}(\hat{\theta}))^2]$ where Λ is the score
- We know that $\mathbb{E}(\Lambda) = 0$
- Let's look at the LHS of the equation:
  - Since $\mathbb{E}(\Lambda) = 0$, we can simplify to $\mathbb{E}[\Lambda(\hat{\theta} - \mathbb{E}(\hat{\theta}))] = \mathbb{E}[\Lambda\hat{\theta}] - \mathbb{E}[\Lambda]\mathbb{E}[\hat{\theta}] = \mathbb{E}[\Lambda\hat{\theta}] - 0$
  - This can be developed to: $\mathbb{E}[\Lambda\hat{\theta}] = \int p(\boldsymbol{x}|\theta)\frac{\frac{\partial}{\partial\theta}p(\boldsymbol{x}|\theta)}{p(\boldsymbol{x}|\theta)}\hat{\theta}dx = \frac{\partial}{\partial\theta}(\int p(\boldsymbol{x}|\theta)\hat{\theta}dx - \theta) + 1$ where the last part $(-\theta)+1)$ can be added, because $\frac{\partial}{\partial\theta} - \theta = -1$ and we compensate this with $+1$
  - This is equal to the derivative of the bias + 1: $\frac{\partial}{\partial\theta}(\int p(\boldsymbol{x}|\theta)\hat{\theta}dx - \theta) + 1 = \frac{\partial}{\partial\theta}(\mathbb{E}[\hat{\theta}] - \theta) + 1 = \frac{\partial}{\partial\theta}\text{bias} + 1$
- Let's look at the RHS of the equation: Since $\mathbb{E}(\Lambda) = 0$, first term is $\mathbb{E}(\Lambda^2) = \boldsymbol{I}$
- Then, we have: $(\frac{\partial}{\partial\theta}\text{bias}+1)^2 \leq \boldsymbol{I} \times \mathbb{E}[(\hat{\theta} - \mathbb{E}(\hat{\theta}))^2] = \boldsymbol{I} \times \mathbb{E}[(\hat{\theta} - \theta - \mathbb{E}(\hat{\theta}) + \theta)^2] = ... = \boldsymbol{I} \times \mathbb{E}[(\hat{\theta} - \theta)^2] - \text{bias}^2$
- Then, we have $\frac{(\frac{\partial}{\partial\theta}\text{bias}+1)^2}{\boldsymbol{I}} + \text{bias}^2 \leq \mathbb{E}[(\hat{\theta} - \theta)^2]$

Bias Variance Tradeoff

- Mean squared error $\mathbb{E}[(\hat{f}(\boldsymbol{X}) - \boldsymbol{y})^2]$ can be decomposed into: $(\mathbb{E}[\hat{f}(\boldsymbol{X})] - f(\boldsymbol{X}))^2 + \mathbb{V}(\hat{f}(\boldsymbol{X})) + \mathbb{E}[\epsilon^2] = \text{bias}^2 + \text{variance} + \text{irreducible error}$

  Proof:
  - $\boldsymbol{y} = f(\boldsymbol{X}) + \epsilon$
  - $\mathbb{E}[(\hat{f}(\boldsymbol{X}) - \boldsymbol{y})^2] = \mathbb{E}[(\hat{f}(\boldsymbol{X}) - \mathbb{E}[\hat{f}(\boldsymbol{X})] + \mathbb{E}[\hat{f}(\boldsymbol{X})] - f(\boldsymbol{X}) + \epsilon)^2] = \mathbb{E}[(\hat{f}(\boldsymbol{X}) - \mathbb{E}[\hat{f}(\boldsymbol{X})])^2] + \mathbb{E}[(\mathbb{E}[\hat{f}(\boldsymbol{X})] - f(\boldsymbol{X}))^2] + \mathbb{E}[\epsilon^2] - 2\mathbb{E}[(\hat{f}(\boldsymbol{X}) - \mathbb{E}[\hat{f}(\boldsymbol{X})])(\mathbb{E}[\hat{f}(\boldsymbol{X})] - f(\boldsymbol{X}) + \epsilon)]$
  - Fourth term equals 0:
    - $2\mathbb{E}[(\hat{f}(\boldsymbol{X}) - \mathbb{E}[\hat{f}(\boldsymbol{X})])(\mathbb{E}[\hat{f}(\boldsymbol{X})] - f(\boldsymbol{X}) + \epsilon)] = 2(\mathbb{E}[\hat{f}(\boldsymbol{X})] - f(\boldsymbol{X}) + \epsilon)\mathbb{E}[(\hat{f}(\boldsymbol{X}) - \mathbb{E}[\hat{f}(\boldsymbol{X})])]$ because $(\mathbb{E}[\hat{f}(\boldsymbol{X})] - f(\boldsymbol{X}) + \epsilon)$ is deterministic
    - In last equation, second term equals 0, so whole equation is 0
  - Then, we are left with: $\text{variance} + \text{bias}^2 + \text{irreducible error}$
- Bias: Error generated by the fact that we approximate a complex relationship via a simpler model (small function class) with a certain presupposed parametric form
- Variance: Error generated by the fact that we estimate the model parameters with a noisy training sample (small sample), rather than the population
- Irreducible error: Error generated by measurement error and the fact that we estimate y as a function of X, when it is a function of many other factors
- Bias variance tradeoff: Bias and variance cannot be reduced simultaneously
  - High variance associated with overfitting: Model corresponds too closely to particular training set resp. performs poorly on unseen data, but well on training set
  - High bias associated with underfitting: Model fails to capture underlying relationships resp. performs poorly on both training set and unseen data

Approximating Generalisation Loss via Empirical Loss

Via resampling methods —

Cross-validation:

- Partition data $\mathcal{Z}$ into K equally sized disjoint subsets: $\mathcal{Z} = \mathcal{Z}_1 \cup \mathcal{Z}_2 \cup ... \cup \mathcal{Z}_K$
- Produce estimator $\hat{f}^{-v}$ from $\mathcal{Z} \backslash \mathcal{Z}_v$ for $v \leq K$
- Empirical loss given by: $\hat{\mathcal{R}}^{cv} = \frac{1}{n}\sum_{i \leq n} LO(y_i - \hat{f}^{-k(i)}(x_i))$ where $k(i)$ maps i to partition $\mathcal{Z}_{k(i)}$ where (x_i, y_i) belongs

Bootstrapping:

- Draw B samples with replacement of size n from data $\mathcal{Z}$: $\mathcal{Z}^{*b}$
- Compute estimate $S(\mathcal{Z}^{*b})$ for each bootstrap sample
- For each estimate $S(\mathcal{Z}^{*b})$, we can give a mean and variance:
  - $\overline{S} = \frac{1}{B}\sum_b S(\mathcal{Z}^{*b})$
  - $\sigma^2(S) = \frac{1}{B-1}\sum_b (S(\mathcal{Z}^{*b}) - \overline{S})^2$
- Empirical loss given by: $\hat{\mathcal{R}}(\mathcal{A}) = \frac{1}{B}\sum_{b=1}^{B}\frac{1}{n}\sum_{i=1}^{n} LO(y_i - \hat{f}^{*b}(x_i))$
- Out-of-bag loss given by: $\hat{\mathcal{R}}^{bs} = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{|C^{-i}|}\sum_{b \in C^{-i}} LO(y_i - \hat{f}^{*b}(x_i))$ where C^{-i} contains all bootstrap indices b so that $\mathcal{Z}^{*b}$ does not contain (x_i, y_i)
- Empirical loss of bootstrap uses training data to estimate $\hat{\mathcal{R}}$, i.e. it is generally too optimistic. We can correct this by combining the empirical and out-of-bag loss:
  - Probability that (x_i, y_i) is not in sample $\mathcal{Z}^{*b}$ of size n is given by $(1 - \frac{1}{n})^n = \frac{1}{e}$ as $n \to \infty \approx \frac{1}{3}$
  - Probability that (x_i, y_i) is in sample $\mathcal{Z}^{*b}$ of size n is given by $1 - \frac{1}{e}$ as $n \to \infty \approx \frac{2}{3}$
  - We then define: $\hat{\mathcal{R}}^{(0.632)} = 0.368\hat{\mathcal{R}}(\mathcal{A}) + 0.632\hat{\mathcal{R}}^{bs}$

Common Loss Measures

Confusion matrix —

- Precision = Correctly predicted positive / predicted positive
- Recall = Correctly predicted positive / positive in reality
- Accuracy = $(TP + TN)/N$
- F1 score = Harmonic mean between precision and recall = $(2PR)/(P + R)$

Curve scores —

- ROC AUC curve:
  - Plots the TPR (recall, y) against the False Positive Rate FPR (x) at various threshold levels
  - A perfect model has an AUC of 1, a random model has an AUC of 0.5
  - Useful when we care about the trade-off between sensitivity and specificity
- Precision recall curve:
  - Plots the precision (y) against the recall (x) at various threshold levels
  - A perfect model has precision and recall of 1, the curve for a random model is a horizontal line at the baseline precision, reflecting the proportion of positive samples in the dataset
  - Useful when we care about the trade-off between sensitivity and specificity

Structured prediction and NLP —

- Some outcomes are more similar than others, e.g. if a translation is almost right vs. a translation is completely wrong
- NLP evaluation metrics:
  - Intrinsic evaluation:
    - Held-out log likelihood $l(w)$
    - Perplexity: $perplexity(w) = 2^{-\frac{l(w)}{M}}$
    - Lower perplexity = higher likelihood
    - Cosine similarity of embeddings for words, which are humanly considered similar
    - Word analogy testing (e.g. king vs. queen, man vs. woman)
  - Extrinsic evaluation on downstream task

TBA

12 Estimating Common Distributions

Gaussian

Frequentism (MLE) —

- Likelihood (excl. constants): $L = (\frac{1}{\sigma})^n \prod_{i=1}^{n} exp(-\frac{1}{2\sigma^2}(\boldsymbol{x}^{(i)} - \boldsymbol{\mu})^2)$
- Log-likelihood: $LL = -nlog(\sigma) - \sum_{i=1}^{n}(\frac{1}{2\sigma^2}(\boldsymbol{x}^{(i)} - \boldsymbol{\mu})^2)$
- μ_{MLE} is sample mean: $\frac{1}{n}\sum_{i=1}^{n}\boldsymbol{x}^{(i)}$:
  - Derivative of log-likelihood wrt μ: $\nabla_{\mu}LL = \nabla_{\mu}(-\sum_{i=1}^{n}(\frac{\boldsymbol{x}^{(i)^2} - 2\boldsymbol{x}^{(i)}\mu + \mu^2}{2\sigma^2})) = \nabla_{\mu}(-\sum_{i=1}^{n}(-\frac{\boldsymbol{x}^{(i)}\mu}{\sigma^2} + \frac{\mu^2}{2\sigma^2})) = -\sum_{i=1}^{n}(-\frac{\boldsymbol{x}^{(i)}}{\sigma^2} + \frac{2\mu}{2\sigma^2}) = \sum_{i=1}^{n}(\frac{\boldsymbol{x}^{(i)} - \mu}{\sigma^2}) = \sum_{i=1}^{n}\boldsymbol{x}^{(i)} - n\mu = 0$
- ${\sigma^2}_{MLE}$ is sample variance: $\frac{1}{n}\sum_{i=1}^{n}(\boldsymbol{x}^{(i)} - \boldsymbol{\mu})^2$:
  - Derivative of log-likelihood wrt σ: $\nabla_{\sigma}LL = -n\nabla_{\sigma}log(\sigma) - \nabla_{\sigma}(\sum_{i=1}^{n}(\frac{(\boldsymbol{x}^{(i)} - \mu)^2}{2\sigma^2})) = \frac{-n}{\sigma} - \nabla_{\sigma}(\sum_{i=1}^{n}\frac{1}{2}\sigma^{-2}(\boldsymbol{x}^{(i)} - \mu)^2) = \frac{-n}{\sigma} - (\sum_{i=1}^{n} -1\sigma^{-3}(\boldsymbol{x}^{(i)} - \mu)^2) = -n + \sum_{i=1}^{n}(\frac{(\boldsymbol{x}^{(i)} - \mu)^2}{\sigma^2}) = 0$
  - ${\sigma^2}_{MLE}$ is problematic when $n \leq m$, since covariance matrix Σ is then only psd (not pd) and in the diagonalization $\Sigma = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{\intercal}$ at least one diagonal element of Λ is zero. Solution: Regularization: ${\sigma^2}_{Adj} = {\sigma^2}_{MLE} + \frac{\rho}{n}$

Bayesianism —

- Assume Σ is known and $\mu \sim \mathcal{N}(\mu_0, \Sigma_0)$ is the outcome of a random variable
- $p(\mu|\boldsymbol{X}, \mu_0, \Sigma_0) \propto p(\boldsymbol{X}|\mu, \Sigma)p(\mu|\mu_0, \Sigma_0)$
- $p(\boldsymbol{X}|\mu, \Sigma) = \frac{1}{2\pi^{mn/2}}\frac{1}{|\Sigma|^{n/2}}exp(\frac{1}{2}\sum_{i=1}^{n}(\boldsymbol{x}^{(i)} - \mu)^{\intercal}\Sigma^{-1}(\boldsymbol{x}^{(i)} - \mu))$
- $p(\mu|\mu_0, \Sigma_0) = \frac{1}{2\pi^{m/2}}\frac{1}{|\Sigma_0|^{n/2}}exp(\frac{1}{2}\sum_{i=1}^{n}(\mu - \mu_0)^{\intercal}\Sigma_0^{-1}(\mu - \mu_0))$
- $p(\mu|\boldsymbol{X}, \mu_0, \Sigma_0) \propto exp(-\frac{1}{2}(\mu^{\intercal}\Sigma_0^{-1}\mu + n\mu^{\intercal}\Sigma^{-1}\mu - 2\mu_0{}^{\intercal}\Sigma_0^{-1}\mu - 2n\overline{\boldsymbol{x}}^{\intercal}\Sigma^{-1}\mu))$ after combining exponents of the prior and likelihood, expanding, absorbing terms unrelated to μ into a constant, and replacing $\sum_{i=1}^{n}\boldsymbol{x}^{(i)\intercal}$ by $n\overline{\boldsymbol{x}}^{\intercal}$
- We now apply a symmetric matrix property $\boldsymbol{x}^{\intercal}\boldsymbol{A}\boldsymbol{x} + 2\boldsymbol{x}^{\intercal}\boldsymbol{b} = (\boldsymbol{x} + \boldsymbol{A}^{-1}\boldsymbol{b})^{\intercal}\boldsymbol{A}(\boldsymbol{x} + \boldsymbol{A}^{-1}\boldsymbol{b}) - \boldsymbol{b}^{\intercal}\boldsymbol{A}^{-1}\boldsymbol{b}$, with $\boldsymbol{\mu} = \boldsymbol{x}$, $-(\Sigma_0^{-1} + n\Sigma^{-1})^{-1} = \boldsymbol{A}^{-1}$ and $(\Sigma^{-1}n\overline{\boldsymbol{x}} + \Sigma_0^{-1}\mu_0) = \boldsymbol{b}$
- Through this, we get $p(\mu|\boldsymbol{X}, \mu_0, \Sigma_0) \propto \exp(\frac{1}{2}(\mu(\Sigma_0^{-1} + n\Sigma^{-1})^{-1}(\Sigma^{-1}n\overline{\boldsymbol{x}} + \Sigma_0^{-1}\mu_0))^{\intercal}(\Sigma_0^{-1} + n\Sigma^{-1})(\mu - (\Sigma_0^{-1} + n\Sigma^{-1})^{-1}(\Sigma^{-1}n\overline{\boldsymbol{x}} + \Sigma_0^{-1}\mu_0))) = exp(\frac{1}{2}(\mu - \mu_n)^{\intercal}\Sigma_n^{-1}(\mu - \mu_n))$
- Thus, $p(\mu|\boldsymbol{X}, \mu_0, \Sigma_0) \sim \mathcal{N}(\mu_n, \Sigma_n)$ with
  - $\mu_n = (\Sigma_0^{-1} + n\Sigma^{-1})^{-1}(\Sigma^{-1}n\overline{\boldsymbol{x}} + \Sigma_0^{-1}\mu_0) =$ (if Σ equals 1) $\frac{n\overline{\boldsymbol{x}}\Sigma_0 + \mu_0}{n\Sigma_0 + 1}$
  - $\Sigma_n = (\Sigma_0^{-1} + n\Sigma^{-1})^{-1} =$ (if Σ equals 1) $\frac{\Sigma_0}{n\Sigma_0 + 1}$
- For Bayesian parameter μ_n:
  - μ_n is a compromise between MLE and prior, approximating prior for small n and MLE for large n
  - If prior variance is small (i.e. if we are certain of our prior), prior mean weighs more strongly
- For Bayesian parameter Σ_n:
  - Σ_n approximates prior for small n and MLE for large n
  - If prior variance is small (i.e. if we are certain of our prior), posterior variance is also small

Bayesianism: Absolute Error —

- Conditional median of y given $X = x$ is the Bayesian estimation of y from $X = x$, when we take the absolute error $|\hat{y} - y|$ as the cost function: $\mathbb{E}[|\hat{y} - y||X = x]$

13 Linear Regression

Description

Task — Regression

Description —

- Supervised
- Parametric

Formulation

- $y^{(i)} = \beta \cdot \boldsymbol{x}^{(i)}$ resp. $\boldsymbol{y} = \boldsymbol{X}\beta$ where $\boldsymbol{X}$ contains n rows, each of which represents an instance, and m columns, each of which represents a feature
- $\hat{\boldsymbol{y}} = \boldsymbol{X}\beta$ is a projection of $\boldsymbol{y}$ to the columnspace of $\boldsymbol{X}$
- β lies in the rowspace of $\boldsymbol{X}$ resp. columnspace of $\boldsymbol{X}^{\intercal}$

Optimization

Parameters — Find parameters β

Objective function — Ordinary least squares estimator (OLSE):

- Minimize mean squared error: $LO = \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - \beta \cdot \boldsymbol{x}^{(i)})^2$ resp. $LO = (\boldsymbol{y} - \boldsymbol{X}\beta)^{\intercal}(\boldsymbol{y} - \boldsymbol{X}\beta)$

Orthogonality principle:
- Yields same result as OLSE
- $\hat{y} = X\beta$ is a projection of y to the columnspace of X
- Then, by the orthogonality principle, $X \cdot (\hat{y} - y) = X \cdot (X\beta - y) = 0$
- $\Rightarrow \beta = (X^\top X)^{-1} X^\top y$
- Alternatively, β lies in the columnspace of $X^\top$
- Then, we can express β as $X^\top [\alpha_1, ..., \alpha_n]^\top$
- This yields an equation system $y = XX^\top [\alpha_1, ..., \alpha_n]^\top$ which can be solved for α_i
- On that basis, β can be calculated

PCA:
- Yields same result as OLSE
- Instances $y^{(i)}, x^{(i)} = \xi^{(i)}$ can be projected onto hyperplane given by $X\beta$
- Projections are given by $\hat{\xi}^{(i)}$
- Residuals are given by $e^{(i)} = \xi^{(i)} - \hat{\xi}^{(i)}$
- Since $e^{(i)}$ is orthogonal to $\hat{\xi}^{(i)}$, we can write using Pythagorean theorem: $\|e^{(i)}\|^2 = \|\xi^{(i)}\|^2 - \|\hat{\xi}^{(i)}\|^2$
- This is a PCA via SVD problem

MLE:
- Yields same result as OLSE

Gradient descent:
- Minimum-norm solution
- Yields same result as OLSE

Pseudo Inverse:
- Yields same result as OLSE
- Minimum-norm solution
- β minimizes MSE if $\hat{y} = X\beta$ is a projection of y to the columnspace of X
- Given matrix projection via SVD, $XX^{\#}y$ is that projection
- $\Rightarrow \beta = X^{\#}y = (X^\top X)^{-1} X^\top y$

Optimization —
- $\nabla_\beta LO = \frac{1}{2}\nabla_\beta((y - \beta \cdot x)^2 = (y - \beta \cdot x)x = 0$ resp. $\nabla_\beta LO = \frac{1}{2}\nabla_\beta((y - X\beta)^\top (y - X\beta)) = \frac{1}{2}\nabla_\beta(\beta^\top X^\top X\beta - 2y^\top X\beta) = X^\top X\beta - X^\top y = X^\top(X\beta - y) = 0$
- $\Rightarrow \beta = (X^\top X)^{-1} X^\top y$

Hypothesis Testing of Found Parameters —
- Let $y|X \sim \mathcal{N}(y, \sigma^2 I) = \mathcal{N}(X\beta, \sigma^2 I)$
- Let $\hat{\beta} = (X^\top X)^{-1} X^\top y = X^+ y$ be the OLSE where X^+ is a scalar
- Then, $\hat{\beta} \sim \mathcal{N}(X^+ X\beta, X^{+\top} \sigma^2 X^+) = \mathcal{N}(\beta, (X^\top X)^{-1} \sigma^2)$
  Proof:
  - $\mathcal{N}(X^+ X\beta, X^{+\top} \sigma^2 X^+) = \mathcal{N}(I\beta, \sigma^2 X^+ X^{+\top})$ since X^+ is a scalar
  - Further, we have $\mathcal{N}(I\beta, \sigma^2 X^+ ((X^\top X)^{-1} X^\top)^\top) = \mathcal{N}(\beta, \sigma^2 X^+ X (X^\top X)^{-1\top}) = \mathcal{N}(\beta, \sigma^2 (X^\top X)^{-1})$ since $(X^\top X)$ is symmetric
- We can estimate σ^2 unbiasedly as: $\hat{\sigma}^2 = \frac{1}{n-m} \sum_{i \le n} (X\hat{\beta} - y)^2$
- Then, confidence interval for $\hat{\beta}_j$ given by: $\hat{\beta}_j \pm z_{\alpha/2} \hat{se}(\hat{\beta}_j)$ where
  - $z_{\alpha/2} = \Phi^{-1}(\alpha/2)$ is Gaussian CDF
  - $\hat{se}(\hat{\beta}_j)$ is the j^{th} diagonal element of the covariance matrix $\sigma^2 (X^\top X)^{-1}$
- We can perform a hypothesis test on $\hat{\beta}$ with the *Wald test*:
  - $H_0 : \beta = \beta_0$ (typically 0)
    $H_1 : \beta \neq \beta_0$
  - Wald statistic: $W = \frac{\hat{\beta} - \beta_0}{\hat{se}}$
  - If p-value associated with W is smaller than α resp. if $|W|$ is greater than or equal to the critical value $z_{\alpha/2}$, we reject H_0

Evaluation —
- OLSE is unbiased if noise ϵ has zero mean:
  - Given $y = X\beta + \epsilon$, we can substitute $\hat{\beta} = (X^\top X)^{-1} X^\top (X\beta + \epsilon) = \beta + (X^\top X)^{-1} X^\top \epsilon$
  - Taking the expected value on both sides, we have: $\mathbb{E}(\hat{\beta}) = \beta + (X^\top X)^{-1} X^\top \mathbb{E}(\epsilon)$
  - Then, $\mathbb{E}(\hat{\beta}) = \beta$ if the noise has zero mean
- *Gauss Markov theorem*: OLSE is best (lowest variance, lowest MSE) unbiased estimator, if assumptions (X is full rank and there is no multicollinearity, heteroskedasticity, and exogeneity) are met
  Proof:
  - Let $A^\top y = (X^\top X)^{-1} X^\top y$ be the OLSE
  - Let $C^\top y$ be another unbiased estimator
  - $\mathbb{V}(A^\top y) = A^\top \mathbb{V}(y) A$ since A is constant
  - We can further develop to: $A^\top \sigma^2 I_m A = \sigma^2 A^\top A$ since variance is given by error term
  - Similarly, $\mathbb{V}(C^\top y) = \sigma^2 C^\top C$
  - For the OLSE, we can plug in $(X^\top X)^{-1} X^\top$ for A which yields: $\mathbb{V}(A^\top y) = \sigma^2 (X^\top X)^{-1} X^\top X (X^\top X)^{-1} = \sigma^2 (X^\top X)^{-1}$
  - Then, we have shown that $\mathbb{V}(A^\top y) \le \mathbb{V}(C^\top y)$
- Nonetheless, there may be biased estimators that generate a lower variance and MSE

Characteristics —
- Convex with psd Hessian
- Has global minimum
- Has unique or infinitely many solutions
- Can be solved analytically, if $(X^\top X)$ is invertible
- If it has infinitely many solutions, the preferred solution is the *minimum-norm solution*, which minimizes $\|\beta\|$

14 Linear Minimum Mean Squared Error Estimation (LMMSE)

Description
- Minimizes mean squared error of two random variables, leveraging information about their mean and covariance
- Linear regression with large samples approximates LMMSE

Formulation
- y is observed
- x is a row vector and quantity of interest
- We estimate x as $\hat{x} = h^\top Y = \sum_i h_i y_i$ where X contains m rows, each of which represents the n-sized vector for a random variable
- This can be considered as a projection of x to the rowspace of Y

Optimization
Parameters — Find parameters h
Objective function —
- Minimize expected squared error: $LO = \mathbb{E}[|\hat{x} - x|]$

Optimization —
- By the orthogonality principle, $\mathbb{E}[(\hat{x} - x) \cdot y_i] = \mathbb{E}[(\sum_{l=1}^n h_l y_l - x) \cdot y_i] = 0$ for $i = 1, ..., n$
- Then, $\sum_{l=1}^n \mathbb{E}[y_l \cdot y_i] h_l = \mathbb{E}[x \cdot y_i]$ for $i = 1, ..., n$ which in matrix notation corresponds to $\begin{bmatrix} \mathbb{E}[y_1 \cdot y_1] & ... & \mathbb{E}[y_1 \cdot y_n] \\ ... & ... & ... \\ \mathbb{E}[y_n \cdot y_1] & ... & \mathbb{E}[y_n \cdot y_n] \end{bmatrix} \cdot \begin{bmatrix} h_1 \\ ... \\ h_n \end{bmatrix} = \begin{bmatrix} \mathbb{E}[x \cdot y_1] \\ ... \\ \mathbb{E}[x \cdot y_n] \end{bmatrix}$ resp. concisely $h^\top \mathbb{E}[YY^\top] = \mathbb{E}[xY^\top]$

Interpretation
TBA

Assumptions
TBA

15 Bayesian Linear Regression

Description
Task — Regression
Description —
- Supervised
- Parametric

Formulation
- $y^{(i)} = \beta \cdot x^{(i)}$ resp. $y = X\beta$
- $\beta \sim \mathcal{N}(0, T^2 I_m)$
- $p(\beta) \propto -\frac{1}{2T^2} \beta^\top \beta$

Optimization
Parameters — Find distribution of parameters β
Optimization —
- Likelihood:
  - Conditional on β, $y \propto \mathcal{N}(X\beta, \sigma^2 I_m)$
  - $p(y|X, \beta) = \frac{1}{(2\pi\sigma^2)^{n/2}} exp(-\frac{1}{2\sigma^2}(y - X\beta)^\top (y - X\beta))$
- Posterior: $p(\beta|X, y) \propto p(X, \beta) \times p(\beta) \propto exp(-\frac{1}{2\sigma^2}(y - X\beta)^\top (y - X\beta)) \times exp(-\frac{1}{2T^2}\beta^\top \beta) = exp(-\frac{1}{2}(\frac{1}{\sigma^2} y^\top y - 2\beta^\top X^\top y + \beta^\top X^\top X\beta) + \frac{1}{2T^2}\beta^\top \beta) \propto exp(-\frac{1}{2}(\beta^\top (\frac{1}{\sigma^2} X^\top X + \frac{1}{2T^2} I_m)\beta - \frac{2}{\sigma^2}\beta^\top X^\top y)$
- We now apply a symmetric matrix property $x^\top Ax + 2x^\top b = (x + A^{-1}b)^\top A(x + A^{-1}b) - b^\top A^{-1} b$, with $\beta = x$, $(\frac{1}{\sigma^2} X^\top X + \frac{1}{2T^2} I_m) = A$ and $(\frac{1}{\sigma^2} X^\top y) = b$
- Through this, we get $p(\beta|X, y) \propto \exp(\frac{1}{2}(\beta + (\frac{1}{\sigma^2} X^\top X + \frac{1}{T^2} I_m)^{-1} (\frac{1}{\sigma^2} X^\top y))^\top (\frac{1}{\sigma^2} X^\top X + \frac{1}{T^2} I_m)(\beta + (\frac{1}{\sigma^2} X^\top X + \frac{1}{T^2} I_m)^{-1} (\frac{1}{\sigma^2} X^\top y)))$
- Thus, $p(\beta|X, y) \sim \mathcal{N}(\mu, \Sigma)$ with
  - $\mu = \Sigma \times \frac{1}{\sigma^2} X^\top y$
  - $\Sigma = (\frac{1}{\sigma^2} X^\top X + \frac{1}{T^2} I_m)^{-1}$
- Posterior mean corresponds to parameter β found by ridge regression, if $\lambda = \frac{\sigma^2}{T^2}$
- If we set an infinitely broad prior T^2 then the Bayesian estimate converges to the MLE estimate – if we have n = 0 training instances, the Bayesian estimate reverts to the prior

Characteristics —
- Convex with psd Hessian
- Has global minimum
- Can be solved analytically

16 Ridge (ℓ_2) Regression

Description
Task — Regression
Description —
- Supervised
- Parametric

Formulation
- $y^{(i)} = \beta \cdot x^{(i)}$ resp. $y = X\beta$

Optimization
Parameters — Find parameters β subject to $\|\beta\|^2 \le t$ resp. $\|\beta\|^2 - t \le 0$
Objective function —
- Minimize mean squared error subject to constraint
- Lagrangian formulation: $LO = \frac{1}{n}\sum_{i=1}^n (y^{(i)} - \beta \cdot x^{(i)})^2 + \lambda(\|\beta\|^2 - t)$ resp. $LO = (y - X\beta)^\top (y - X\beta) + \lambda(\|\beta\|^2 - t)$
- Still a OLSE problem, since we can rewrite the objective to minimize $(X\beta - y)$ as the objective to minimize $\|(X'\beta - y')\|^2$ with $X' = \begin{bmatrix} X \\ \lambda I \end{bmatrix}$ and $y' = \begin{bmatrix} y \\ 0 \end{bmatrix}$
- Corresponds to MAP estimation, when X is modeled as a vector of independent zero-mean Gaussian random variables

Optimization —
- $\nabla_\beta LO = 0$
- $\Rightarrow \beta = (X^\top X + \lambda I)^{-1} X^\top y$

Effect —
- Shrinks certain elements of β to near 0
  Proof:
  - Gradient at optimality given by $\frac{\partial (y - X\beta)^\top (y - X\beta)}{\partial \beta} + 2\lambda\beta = 0$
  - Then, $\beta^* = -\frac{1}{2\lambda} \frac{\partial (y - X\beta)^\top (y - X\beta)}{\partial \beta}$
  - This means that each parameter is shrunk by a factor determined by size of λ - the larger λ, the more the parameters are shrunk
  - Larger parameters experience a larger shrinkage

Characteristics —
- Strictly with pd Hessian, since Lagrangian term is strictly convex and the sum of a strictly convex function with a convex function is strictly convex
- Has global minimum
- Has unique solution, as $(X^\top X + \lambda I)$ has linearly independent columns
- Can be solved analytically, as $(X^\top X + \lambda I)$ is always invertible

17 Lasso (ℓ_1) Regression

Description
Task — Regression
Description —
- Supervised
- Parametric

Formulation
- $y^{(i)} = \beta \cdot x^{(i)}$ resp. $y = X\beta$

Optimization
Parameters — Find parameters β subject to $|\beta| \le t$ resp. $|\beta| - t \le 0$
Objective function —
- Minimize mean squared error subject to constraint
- Lagrangian formulation: $LO = \frac{1}{n}\sum_{i=1}^n (y^{(i)} - \beta \cdot x^{(i)})^2 + \lambda(|\beta| - t)$ resp. $LO = (y - X\beta)^\top (y - X\beta) + \lambda(|\beta| - t)$
- Corresponds to MAP estimation, when X is modeled as a vector of independent zero-mean Laplacian random variables

Effect —
- Shrinks certain elements of β to 0
  Proof:
  - Gradient at optimality given by $\frac{\partial (y - X\beta)^\top (y - X\beta)}{\partial \beta} + \frac{\partial \lambda|\beta|}{\partial \beta} = 0$
  - $\frac{\partial \lambda|\beta|}{\partial \beta}$ non-differentiable because there is a sharp edge at $\beta = 0$, but we can work with subgradients for $\beta \neq 0$:
    $$\frac{\partial}{\partial \beta}|\beta| = sgn(\beta) = \begin{cases} -1 & \beta < 0 \\ 0 & \beta = 0 \\ 1 & \beta > 0 \end{cases}$$
  - If we have $-\lambda < \frac{\partial (y - X\beta)^\top (y - X\beta)}{\partial \beta} < \lambda$ the optimum is given by $\beta = 0$
  - This means that some parameters are set to 0 - the larger λ, the more parameters are set to 0
  - Small parameter values (i.e. unimportant features) are more likely to be set to 0
  - For parameters that are not set to 0, LASSO regression has a similar effect as ridge regression and shrinks these parameters towards 0

Characteristics —
- Convex, but not strictly convex
- Has global minimum
- Has unique or infinitely many solutions
- Cannot be solved analytically, since $|\beta|$ is not differentiable at $\beta_i = 0$

18 Polynomial Regression

Description
Task — Regression
Description —
- Supervised
- Parametric

Formulation
- $y^{(i)} = \beta \cdot \phi(x^{(i)})$ resp. $y = \Phi\beta$ where Φ is the transformed design matrix with rows $\phi(x^{(i)})^\top$

Optimization

Parameters — Find parameters β

Objective function —

- Ordinary least squares estimator
- Minimize mean squared error

Optimization —

- $\nabla_\beta LO = 0$
- $\Rightarrow \beta = (\Phi^\top \Phi)^{-1} \Phi^\top y$

Characteristics —

- Convex with psd Hessian
- Has global minimum
- Has unique or infinitely many solutions
- Can be solved analytically, if $(\Phi^\top \Phi)$ is invertible

19 Kernel Methods

Background on Kernel Methods

Description —

- Mechanism for tractably resp. implicitly mapping data into higher-dimensional feature space so that linear models can be used in this feature space
- To do so, we can employ the *kernel trick* and the *representer theorem*
- The requirements are that the kernel function fulfills *Mercer's theorem*, i.e. the kernel is a Mercer kernel

Kernel trick —

- Allows to operate in higher-dimensional feature space, without explicitly calculating this transformation, but instead implicitly computing the inner product in this feature space via a kernel function
- Given two inputs $x^{(i)}, x^{(j)}$ and a feature map $\varphi : \mathbb{R}^m \to \mathbb{R}^k$ we can define an inner product on $\mathbb{R}^k$ via the kernel function: $k(x^{(i)}, x^{(j)}) = \varphi(x^{(i)}) \cdot \varphi(x^{(j)})$
- If a prediction function is described solely in terms of inner products in the input space, it can be lifted into the feature space by replacing the inner product with the kernel function
- Kernel trick requires that span of training instances $span(\varphi(x^{(1)}), ..., \varphi(x^{(N)})) = \mathbb{R}^k$ and, thus, that $N \geq k$.
  Proof:

$$dim(span(...)) = \begin{cases} N & \text{if } N < k \\ k & \text{if } N \geq k \end{cases}$$

- Kernel trick cannot be used in conjunction with feature selection resp. sparsity inducing regularize (e.g. ℓ_1), as this does not satisfy the representer theorem

Representer theorem —

- Allows to avoid directly seeking the k parameters, but only the n parameters that characterize α
- Allows to avoid calculating $\varphi(z)$ when evaluating novel instance, but only sum over weighted set of n kernel function outputs

Mercer's theorem —

- Kernel function is psd and symmetric iff $k(x^{(i)}, x^{(j)}) = \varphi(x^{(i)}) \cdot \varphi(x^{(j)})$
  - Psd: $x^\top K x \geq 0$ where K is the kernel matrix
  - Symmetric: $k(x^{(i)}, x^{(j)}) = k(x^{(j)}, x^{(i)})$
- Kernel that satisfies Mercer's theorem is a Mercer kernel, i.e. we can prove a kernel is a Mercer kernel either if it is psd and symmetric or by finding a feature map such that the kernel function corresponds to an inner product

Formulation

- Feature map $\varphi : \mathbb{R}^m \to \mathbb{R}^k$
- Linear prediction function: $\beta \cdot \varphi(x^{(i)})$
- Regularized loss function: $LO = \sum_{i=1}^n LO(y^{(i)}, \beta \cdot \varphi(x^{(i)}) + \Omega(\beta))$
- Iff $\Omega(\beta))$ is a non-decreasing function, then the parameters β that minimize the loss function can be rewritten as: $\beta = \sum_{i=1}^n \alpha^{(i)} \varphi(x^{(i)})$
- Outcome of novel instance can be predicted as: $\beta \cdot \varphi(z) = \sum_{i=1}^n \alpha^{(i)} \varphi(x^{(i)}) \cdot \varphi(z) = \sum_{i=1}^n \alpha^{(i)} k(x^{(i)}, z)$
- Act of prediction becomes act of measuring similarity to training instances in feature map space

Kernel Types

Polynomial kernel —

- $\varphi(x) = [x^\alpha]_{\alpha \in \mathbb{N}^m}$ where $\alpha = (\alpha_1, ..., \alpha_m)$ is the multi-index representing the power and $x^\alpha = x_1^{\alpha_1} \times ... \times x_m^{\alpha_m}$ is the mononomial term corresponding to the multi-index α
- E.g. if degree = 2, then $k(x^{(i)}, x^{(j)}) = 1 + 2x_1^{(i)} x_1^{(j)} + 2x_2^{(i)} x_2^{(j)} + (x_1^{(i)} x_1^{(j)})^2 + (x_2^{(i)} x_2^{(j)})^2 + 2x_1^{(i)} x_1^{(j)} x_2^{(i)} x_2^{(j)}$
- Inner product diverges to infinity
- To address this, we often use RBF kernel instead

RBF kernel —

- Gives access to infinite feature space
- $\varphi(x) = exp(-\frac{1}{2}\|x\|^2)[\frac{x^\alpha}{\sqrt{\alpha!}}]_{\alpha \in \mathbb{N}^m}$
- $k(x^{(i)}, x^{(j)}) = \sigma^2 exp(-\frac{\|x^{(i)} - x^{(j)}\|^2}{2l^2})$
  Proof:
  - $exp(-\frac{1}{2}\|x^{(i)}\|^2) exp(-\frac{1}{2}\|x^{(j)}\|^2) \sum_\alpha [\frac{x^{(i)^\alpha} x^{(j)^\alpha}}{\alpha!}]$
  - Given multinomial series expansion, $\sum_\alpha [\frac{x^{(i)^\alpha} x^{(j)^\alpha}}{\alpha!}] = exp(x^{(i)\top} x^{(j)})$
  - $exp(-\frac{1}{2}\|x^{(i)}\|^2 - \frac{1}{2}\|x^{(j)}\|^2 + x^{(i)\top} x^{(j)}) = exp(-\frac{\|x^{(i)} - x^{(j)}\|^2}{2\sigma^2})$
- Length scale parameter l controls how quickly the similarity decays with distance: If l is large, points with high distance still have high covariance
- Variance parameter σ controls the vertical scale of the function
- RBF kernel is stationary, meaning that only the relative distance between two points determines the value output by the kernel function
- Challenge: Cannot ignore irrelevant dimensions (whereas e.g. a neural network can do this by setting the associated weights to 0)

Kernel compositions —

- New valid kernels can be composed via:
  - Addition: $k_1 + k_2$
  - Multiplication: $k_1 \times k_2$
  - Scaling: $c \times k_1$ for $c > 0$
  - Composition: $f(k_1)$ where f is a polynomial with positive coefficients or the exponential function
- Valid kernels:
  - $k'(x_1, x_2) = ck(x_1, x_2)$, since $\varphi'(x) = \sqrt{c}\varphi(x)$
  - $k'(x_1, x_2) = f(x_1) k(x_1, x_2) f(x_2)$, since $\varphi'(x) = f(x)\varphi(x)$
  - $k'(x_1, x_2) = k_1(x_1, x_2) + k_2(x_1, x_2)$, since the requirements for a valid kernel are that its psd and symmetric, which is retained when two psd and symmetric matrices are added
  - $k'(x_1, x_2) = k_1(x_1, x_2) k_2(x_1, x_2)$, since new kernel is given by the i^{th} feature value under feature map φ_1 multiplied by the j^{th} feature value under feature map φ_2
  - $k'(x_1, x_2) = exp(k(x_1, x_2))$, since we can apply Taylor series expansion $\sum_{n=1}^r \frac{k(x_1, x_2)^r}{r!} = exp(k(x_1, x_2)) = k'(x_1, x_2)$ as $r \to \infty$ and we know that exponentiation, addition, and scaling produces valid new kernels from above

20 Polynomial Kernel Regression

Description

Task — Regression

Description —

- Supervised
- Parametric

Formulation

- $y = \beta \cdot \varphi(x^{(i)})$

Optimization

Parameters — Find parameters β

Objective function —

- Ordinary least squares estimator (OLSE)
- Minimize mean squared error: $LO = \frac{1}{n} \sum_{i=1}^n (y^{(i)} - \beta \cdot \varphi(x^{(i)}))^2$

Optimization —

- Primal solution:
  - Parameters can be estimated as: $\beta = (\Phi^\top \Phi)^{-1} \Phi^\top y$
  - Prediction for novel instance: $\beta \cdot \varphi(z) = (\Phi^\top \Phi)^{-1} \Phi^\top y \cdot \varphi(z) = y^\top \Phi (\Phi^\top \Phi)^{-1} \varphi(z)$
- Let us define $K = \Phi\Phi^\top$ as the kernel matrix of the training data with $K_{ij} = \varphi(x^{(i)}) \cdot \varphi(x^{(j)})$
- Dual solution α if we have no regularization, i.e. $\lambda = 0$:
  - Parameters can be estimated as: $\beta = \Phi^\top K^{-1} y$
    Proof:
    - $(\Phi^\top \Phi + \lambda I)\beta = \Phi^\top y$
    - $\Rightarrow \Phi^\top \Phi \beta + \lambda I \beta = \Phi^\top y$
    - $\Rightarrow I\beta = \Phi^\top \lambda^{-1} (y - \Phi\beta)$
    - Since we know from the representer theorem that $\beta = \Phi^\top \alpha$, we can say: $\alpha = \lambda^{-1}(y - \Phi\beta)$
    - We can further develop this to: $\lambda\alpha = (y - \Phi\beta)$
    - Replacing β by $\Phi^\top \alpha$ yields: $\lambda\alpha = (y - \Phi\Phi^\top \alpha)$
    - $\Rightarrow \alpha = (\Phi\Phi^\top + \lambda I)^{-1} y = K^{-1} y$
    - With this, we can calculate the parameters: $\beta = \Phi^\top \alpha = \Phi^\top (\Phi\Phi^\top + \lambda I)^{-1} y = \Phi^\top K^{-1} y$
  - Prediction for novel instance: $\beta \cdot \varphi(z) = y^\top (\Phi\Phi^\top)^{-1} \Phi\varphi(z) = y^\top (\Phi\Phi^\top)^{-1} k$ where $k = \Phi\varphi(z) = [k(x^{(1)}, z), ..., k(x^{(n)}, z)]^\top = [\varphi(x^{(1)}) \cdot \varphi(z), ..., \varphi(x^{(n)}) \cdot \varphi(z)]^\top$ is a kernel vector, consisting of kernel values between training instances and new instance

Algorithm — Training:

1. Compute kernel matrix given RBF kernel
   Time complexity: $\mathcal{O}(n^2 \times m)$ for n^2 kernel matrix values and m number of features in each instance vector
2. Perform training by solving $\alpha = K^{-1} y$ for α
   Time complexity: $\mathcal{O}(n^3)$
3. Store α
   Space complexity: $\mathcal{O}(n^2)$

Prediction:

1. Compute kernel vector
   Time complexity: $\mathcal{O}(n \times m \times d)$ for d new instances, given n instances in training data and m features in each instance vector
2. Store k
   Space complexity: $\mathcal{O}(n \times d)$ for d new instances, given n as length of kernel vector
3. Predict response using stored kernel vector
   Time complexity: $\mathcal{O}(n \times d)$ for d new instances, given n as length of α

Value:

- Primal solution training is of time complexity $\mathcal{O}(k^3)$ and prediction is of time complexity $\mathcal{O}(k)$
- Dual solution speeds this up as seen above in the algorithm

Characteristics —

- Convex with psd Hessian
- Has global minimum
- Has unique or infinitely many solutions
- Can be solved analytically

21 Gaussian Processes

Description

Task — Models a distribution over functions

Description —

- Supervised
- Non-parametric

Formulation

- $y^{(i)} = \beta \cdot x^{(i)} + \epsilon$ resp. $y = X\beta + \epsilon$
- $\beta \sim \mathcal{N}(0, \Lambda^{-1})$
- $\epsilon \sim \mathcal{N}(0, \sigma I_m)$

Optimization

Optimization —

- If we compute the moment of the Gaussian:
  - $\mathbb{E}[y] = X^\top \mathbb{E}(\beta) = X^\top 0 = 0$
  - $Cov(y) = \mathbb{E}[(X^\top \beta + \epsilon)(X^\top \beta + \epsilon)^\top] = X\mathbb{E}(\beta\beta^\top) = X^\top + X\mathbb{E}(\beta)\mathbb{E}(\epsilon^\top) + \mathbb{E}(\epsilon)\mathbb{E}(\beta^\top)X^\top + \mathbb{E}(\epsilon\epsilon^\top) = 0$ where
    - $\mathbb{V}(\beta) = \mathbb{E}(\beta\beta^\top)$ and $\mathbb{V}(\epsilon) = \mathbb{E}(\epsilon\epsilon^\top)$ because $\mathbb{V}(x) = \mathbb{E}[(x - \mathbb{E}(x))^2] = \mathbb{E}[x^2]$ if $\mathbb{E}(x) = 0$, which is the case here due to the defined distributions
    - $\mathbb{E}(\epsilon) = 0$
  - Plugging in the variance for β and ϵ, we have $Cov(y) = X\Lambda^{-1}X^\top + \sigma^2 I_m$
  - This can be written as a Kernel matrix K:
    $$\begin{bmatrix} K_{1,1} + \sigma^2 & ... & ... & K_{1,n} \\ ... & K_{2,2} + \sigma^2 & ... & ... \\ ... & ... & ... & ... \\ K_{n,1} & ... & ... & K_{n,n} + \sigma^2 \end{bmatrix}$$
    with $K_{ij} = x^{(i)\top} \Lambda^{-1} x^{(j)}$
  - In this kernel matrix, the kernel function can take any shape
- On this basis, Gaussian process is defined as collection of random variables such that every finite subset of variables is jointly Gaussian: $f \sim \mathcal{GP}(\mu, K)$
- A new instance follows the distribution $p(y_{n+1}) = \mathcal{N}(k^\top C_n^{-1} y, c - k^\top C_n^{-1} k)$ where
  - $k = k(x^{(1)}, x^{(n+1)}), ..., k(x^{(n)}, x^{(n+1)})]^\top = [\varphi(x^{(1)}) \cdot \varphi(x^{(n+1)}), ..., \varphi(x^{(n)}) \cdot \varphi(x^{(n+1)})]^\top$ is the kernel vector
  - $C_n = k(x^{(i)}, x^{(j)}) + \sigma^2 I_m$
  - $c = k(x^{(n+1)}, x^{(n+1)}) + \sigma^2 I_m$

  Proof:
  - We derive the joint distribution $p(\begin{bmatrix} y \\ y_{n+1} \end{bmatrix}) \sim \mathcal{N}[0, \begin{bmatrix} C_n & k \\ k^\top & c \end{bmatrix}]$
  - To obtain a closed-form solution for this, we can make use of the following theorem:
    - Given a joint Gaussian distribution: $\begin{bmatrix} a_1 \\ a_2 \end{bmatrix}) \sim \mathcal{N}[\begin{bmatrix} u_1 \\ u_2 \end{bmatrix}, \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}]$
    - The conditional Gaussian distribution is given by: $p(a_2 | a_1 = z) = \mathcal{N}(u_2 + \Sigma_{21}\Sigma_{11}^{-1}(z - u_1), \Sigma_{22} - \Sigma_{21}\Sigma_{11}^{-1}\Sigma_{12})$
  - Then, we get $p(y_{n+1}) = \mathcal{N}(k^\top C_n^{-1} y, c - k^\top C_n^{-1} k)$
- A new instance vector follows the distribution $p(y_*) = \mathcal{N}(K_{*n} K_{nn}^{-1} y_n, K_{**} - K_{*n} K_{nn}^{-1} K_{n*})$ where
  - Subscript $*$ indicates new instances, subscript n indicates old instances
  - $K$is the kernel matrix, e.g. K_{*n} has new instances on rows and old instances on columns

Algorithm —

1. Compute kernel matrix based on observed data
2. Compute kernel vector based on observed data and new instance
3. Calculate mean and variance of predicted distribution
4. Return predicted distribution

22 Kernel SVM

Description

Task — Classification

Description —

- Supervised
- Parametric

Optimization

Optimization —

- Cost function $argmin_\beta \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^n \xi^{(i)}$ resp. dual Lagrangian objective function $argmax_\alpha \sum_{i=1}^n \alpha^{(i)} - \frac{1}{2}\sum_{i=1}^n\sum_{j=1}^n \alpha^{(i)}\alpha^{(j)}y^{(i)}y^{(j)}\varphi(\boldsymbol{x}^{(i)})\cdot\varphi(\boldsymbol{x}^{(j)})$ of soft-margin SVM is eligible for kernel trick
- Thus, $\beta^* = \sum_{i=1}^n \tilde{\alpha}^{(i)}\varphi(\boldsymbol{x}^{(i)})$ where $\tilde{\alpha}$ refers to Kernel theorem
- Given general Lagrangian solution, $\beta^* = \sum_{i=1}^n \alpha^{(i)}y^{(i)}\varphi(\boldsymbol{x}^{(i)})$ where α refers to Lagrange multiplier, we know that: $\tilde{\alpha}^{(i)} = \alpha^{(i)}y^{(i)}$
- Outcome of novel instance can be predicted as: $\beta^*\cdot\varphi(\boldsymbol{z}) = \sum_{i=1}^n \alpha^{(i)}y^{(i)}\varphi(\boldsymbol{x}^{(i)})\cdot\varphi(\boldsymbol{z}) = \sum_{i=1}^n \alpha^{(i)}y^{(i)}k(\boldsymbol{x}^{(i)},\boldsymbol{z})$

23 Other Regressions (DiD, FE, IV, SC)

TBA

24 Log Linear Models

Formulation

- Family of probability distributions defined as: $p(y \mid \boldsymbol{x},\boldsymbol{\theta}) = \frac{1}{Z(\boldsymbol{\theta})}\exp(\boldsymbol{\theta}\cdot f(\boldsymbol{x},y))$ where:
  - Z is a *partition function*, which acts as a normalization constant to $[0,1]$ and is given by $Z(\boldsymbol{\theta}) = \sum_y' \exp(\boldsymbol{\theta}\cdot f(\boldsymbol{x},y'))$
  - $\boldsymbol{\theta}\cdot f(\boldsymbol{x},y)$ is a freely chosen *linear scoring function*, that says how good x and y are together, and is given by: $\boldsymbol{\theta}\cdot f(\boldsymbol{x},y) = \sum_{i=1}^m \theta_i f_i(\boldsymbol{x},y)$
  - $\boldsymbol{\theta}$ is a set of parameters
  - $f(\boldsymbol{x},y)$ is a feature function
- If we take the log of $p(y \mid \boldsymbol{x},\boldsymbol{\theta})$, we get a linear function: $\log(p(y \mid \boldsymbol{x},\boldsymbol{\theta})) = \boldsymbol{\theta}\cdot f(\boldsymbol{x},y) + \text{constant}$
- $f(\boldsymbol{x},y)$ can be freely chosen

Optimization

Parameters — Find parameters $\boldsymbol{\theta}$

Objective function —

- $\boldsymbol{\theta} = argmax_{\boldsymbol{\theta}}$ log loss
- Log loss $= -\sum_{i=1}^n \log\left(p(y^{(i)} \mid \boldsymbol{x}^{(i)},\boldsymbol{\theta})\right)$
- $= -\sum_{i=1}^n \log\left(\frac{\exp(\boldsymbol{\theta}\cdot f(\boldsymbol{x}^{(i)},y^{(i)}))}{\sum_{y'}\exp(\boldsymbol{\theta}\cdot f(\boldsymbol{x}^{(i)},y'))}\right)$
- $= -\sum_{i=1}^n \boldsymbol{\theta}\cdot f(\boldsymbol{x}^{(i)},y^{(i)}) - \log\left(\sum_{y'}\exp(\boldsymbol{\theta}\cdot f(\boldsymbol{x}^{(i)},y'))\right)$

Optimization — First-order derivative:

- Gradient of log loss given by: $\frac{\partial}{\partial\boldsymbol{\theta}}\text{log loss} = -\sum_{i=1}^n \frac{\partial}{\partial\boldsymbol{\theta}}\left(\boldsymbol{\theta}\cdot f(\boldsymbol{x}^{(i)},y^{(i)}) - \log(\sum_{y'}\exp(\boldsymbol{\theta}\cdot f(\boldsymbol{x}^{(i)},y')))\right)$
- $= -\sum_{i=1}^n \left(f(\boldsymbol{x}^{(i)},y^{(i)}) - \frac{\sum_{y'}\exp(\boldsymbol{\theta}\cdot f(\boldsymbol{x}^{(i)},y'))\times f(\boldsymbol{x}^{(i)},y')}{\sum_{y'}\exp(\boldsymbol{\theta}\cdot f(\boldsymbol{x}^{(i)},y'))}\right)$ since: $\frac{\partial}{\partial x}\log e^{ax} = \frac{1}{e^{ax}}\times\frac{\partial e^{ax}}{\partial x} = \frac{1}{e^{ax}}\times e^{ax}\times\frac{\partial ax}{\partial x} = \frac{1}{e^{ax}}\times e^{ax}\times\alpha$
- $= -\sum_{i=1}^n \left(f(\boldsymbol{x}^{(i)},y^{(i)}) - \sum_{y'} p(y' \mid \boldsymbol{x}^{(i)},\boldsymbol{\theta})\times f(\boldsymbol{x}^{(i)},y')\right)$
- $= -\sum_{i=1}^n f(\boldsymbol{x}^{(i)},y^{(i)}) + \sum_{i=1}^n\sum_{y'} p(y' \mid \boldsymbol{x}^{(i)},\boldsymbol{\theta})\times f(\boldsymbol{x}^{(i)},y')$
- $= -\sum_{i=1}^n f(\boldsymbol{x}^{(i)},y^{(i)}) + \sum_{i=1}^n \mathbb{E}(f(\boldsymbol{x}^{(i)},y'))$
- If we set gradient to 0, we have expectation matching:
  - $\sum_{i=1}^n f(\boldsymbol{x}^{(i)},y^{(i)}) = \sum_{i=1}^n \mathbb{E}(f(\boldsymbol{x}^{(i)},y'))$
  - Optimum is where observed features = expected features

Second-order derivative:

- Hessian of log loss given by: $\frac{\partial^2\,\text{log loss}}{\partial\boldsymbol{\theta}\partial\boldsymbol{\theta}^\intercal} = \frac{\partial}{\partial\boldsymbol{\theta}^\intercal}$ gradient of log loss
- $= \frac{\partial}{\partial\boldsymbol{\theta}^\intercal}\left(-\sum_{i=1}^n\left(f(\boldsymbol{x}^{(i)},y^{(i)}) - \frac{\sum_{y'}\exp(\boldsymbol{\theta}\cdot f(\boldsymbol{x}^{(i)},y'))\times f(\boldsymbol{x}^{(i)},y')}{\sum_{y'}\exp(\boldsymbol{\theta}\cdot f(\boldsymbol{x}^{(i)},y'))}\right)\right)$ where
  - $f(x) = h(x)\times k(x)$
  - $h(x) = \exp(\boldsymbol{\theta}\cdot f(\boldsymbol{x}^{(i)},y'))$
  - $k(x) = \frac{1}{\sum_{y'}\exp(\boldsymbol{\theta}\cdot f(\boldsymbol{x}^{(i)},y'))}$
  - $g(x) = f(\boldsymbol{x}^{(i)},y')$
- $= 0 + \sum_{i=1}^n\sum_{y'}(h'(x)k(x) + k'(x)h(x))g(x) + g'(x)h(x)k(x)$ where
  - $h'(x) = \exp(\boldsymbol{\theta}\cdot f(\boldsymbol{x}^{(i)},y'))\times f(\boldsymbol{x}^{(i)},y')$
  - $k'(x) = -\frac{1}{(\sum_{y'}\exp(\boldsymbol{\theta}\cdot f(\boldsymbol{x}^{(i)},y')))^2}\times\sum_{y'}\exp(\boldsymbol{\theta}\cdot f(\boldsymbol{x}^{(i)},y'))\times f(\boldsymbol{x}^{(i)},y')$ since: $\frac{\partial}{\partial x}\frac{1}{e^{ax}} = \frac{\partial}{\partial x}(e^{ax})^{-1} = -\frac{1}{(e^{ax})^2}\times\frac{\partial e^{ax}}{\partial x} = -\frac{1}{(e^{ax})^2}\times e^{ax}\times\frac{\partial ax}{\partial x} = -\frac{1}{(e^{ax})^2}\times e^{ax}\times\alpha$
  - $g'(x) = 0$
- $= \sum_{i=1}^n\sum_{y'}\exp(\boldsymbol{\theta}\cdot f(\boldsymbol{x}^{(i)},y'))\times f(\boldsymbol{x}^{(i)},y')\times\frac{1}{\sum_{y'}\exp(\boldsymbol{\theta}\cdot f(\boldsymbol{x}^{(i)},y'))}\times f(\boldsymbol{x}^{(i)},y') - \frac{1}{(\sum_{y'}\exp(\boldsymbol{\theta}\cdot f(\boldsymbol{x}^{(i)},y')))^2}\times\sum_{y'}\exp(\boldsymbol{\theta}\cdot f(\boldsymbol{x}^{(i)},y'))\times f(\boldsymbol{x}^{(i)},y')\times\exp(\boldsymbol{\theta}\cdot f(\boldsymbol{x}^{(i)},y'))\times f(\boldsymbol{x}^{(i)},y') + 0$
- $= \sum_{i=1}^n\sum_{y'} p(y' \mid \boldsymbol{x}^{(i)},\boldsymbol{\theta})f(\boldsymbol{x}^{(i)},y')f(\boldsymbol{x}^{(i)},y')^\intercal - p(y' \mid \boldsymbol{x}^{(i)},\boldsymbol{\theta})f(\boldsymbol{x}^{(i)},y')\sum_{y'} p(y' \mid \boldsymbol{x}^{(i)},\boldsymbol{\theta})f(\boldsymbol{x}^{(i)},y')^\intercal$
- $= \sum_{i=1}^n \mathbb{E}[f(\boldsymbol{x}^{(i)},y')f(\boldsymbol{x}^{(i)},y')^\intercal] - \sum_{i=1}^n\left((\sum_{y'} p(y' \mid \boldsymbol{x}^{(i)},\boldsymbol{\theta})f(\boldsymbol{x}^{(i)},y'))(\sum_{y'} p(y' \mid \boldsymbol{x}^{(i)},\boldsymbol{\theta})f(\boldsymbol{x}^{(i)},y')^\intercal\right)$
- $= \sum_{i=1}^n \mathbb{E}[f(\boldsymbol{x}^{(i)},y')f(\boldsymbol{x}^{(i)},y')^\intercal] - \sum_{i=1}^n\left(\mathbb{E}[f(\boldsymbol{x}^{(i)},y')]\mathbb{E}[f(\boldsymbol{x}^{(i)},y')]\right)$
- $= \sum_{i=1}^n Cov(f(\boldsymbol{x}^{(i)},y'))$
- We can set a different prior by introducing an offset:
  - $p_k = \frac{e^{z_k+t_k}}{\sum_{\ell=1}^k e^{z_\ell+t_\ell}}$ where $t_k = \ln(\frac{q(x_k)}{p(x_k)})$
  - Then, the prior on x ($p(x)$) is replaced by $q(x)$

25 Logistic Regression

Description

Task — Binary classification

Description —

- Supervised
- Parametric

Formulation

- Probability of each class is estimated via sigmoid function: $\sigma(z) = \frac{1}{1+e^{-z}} = \frac{e^z}{1+e^z}$
- $P(y=1 \mid \boldsymbol{x}) = \frac{1}{1+e^{-\boldsymbol{\beta}\cdot\boldsymbol{x}}} = \frac{e^{\boldsymbol{\beta}\cdot\boldsymbol{x}}}{1+e^{\boldsymbol{\beta}\cdot\boldsymbol{x}}}$
- $P(y=0 \mid \boldsymbol{x}) = \frac{1}{1+e^{\boldsymbol{\beta}\cdot\boldsymbol{x}}} = \frac{e^{-\boldsymbol{\beta}\cdot\boldsymbol{x}}}{1+e^{-\boldsymbol{\beta}\cdot\boldsymbol{x}}}$
- Odds: $\frac{P(y=1 \mid \boldsymbol{x})}{P(y=0 \mid \boldsymbol{x})} = e^{\boldsymbol{\beta}\cdot\boldsymbol{x}}$
- Log-odds: $ln(\frac{P(y=1 \mid \boldsymbol{x})}{P(y=0 \mid \boldsymbol{x})}) = \boldsymbol{\beta}\cdot\boldsymbol{x} = ln(\frac{P(y=1)}{P(y=0)}) + ln(\frac{P(\boldsymbol{x} \mid y=1)}{P(\boldsymbol{x} \mid y=0)})$
- Geometrically, $z = \boldsymbol{\beta}\cdot\boldsymbol{x}$ defines a linear separating hyperplane:
  - When $z > 0$ resp. log-odds > 0, then odds > 1 resp. $P(y=1 \mid \boldsymbol{x}) > P(y=0 \mid \boldsymbol{x})$, then predict $y = 1$
  - Otherwise predict $y = 0$

Optimization

Parameters — Find parameters $\boldsymbol{\beta}$

Objective function —

- Likelihood: $L(\boldsymbol{\beta}) = \prod_{i=1}^n P(y^{(i)} \mid \boldsymbol{x}^{(i)};\boldsymbol{\beta}) = \prod_{i=1}^n \sigma(z^{(i)})^{y^{(i)}}(1-\sigma(z^{(i)}))^{1-y^{(i)}}$
- Maximize log-likelihood: $\log L(\boldsymbol{\beta}) = \sum_{i=1}^n\left[y^{(i)}\log\sigma(z^{(i)}) + (1-y^{(i)})\log(1-\sigma(z^{(i)}))\right] = \sum_{i=1}^n\left[y^{(i)}\log\frac{1}{1+e^{-z^{(i)}}} + (1-y^{(i)})\log\frac{e^{-z^{(i)}}}{1+e^{-z^{(i)}}}\right] = \sum_{i=1}^n\left[y^{(i)}z^{(i)} - \log(1+e^{z^{(i)}})\right]$
- Minimize log-loss: $-\log L(\boldsymbol{\beta})$

Optimization —

- $\frac{\partial -\log L(\boldsymbol{\beta})}{\partial\boldsymbol{\beta}} = -\sum_{i=1}^n \frac{\partial}{\partial\boldsymbol{\beta}}[y^{(i)}\log\sigma(z^{(i)}) + (1-y^{(i)})\log(1-\sigma(z^{(i)}))] = \sum_{i=1}^n[\sigma(z^{(i)}) - y^{(i)}]\boldsymbol{x}^{(i)}$

Proof:

- Derivative of the sigmoid: $\frac{\partial\sigma(z^{(i)})}{\partial z^{(i)}} = \sigma(z^{(i)})(1-\sigma(z^{(i)}))$
- Derivative of the first term: $\frac{\partial}{\partial\boldsymbol{\beta}}\left[y^{(i)}\log\sigma(z^{(i)})\right] = y^{(i)}\frac{1}{\sigma(z^{(i)})}\frac{\partial\sigma(z^{(i)})}{\partial z^{(i)}}\frac{\partial z^{(i)}}{\partial\boldsymbol{\beta}} = y^{(i)}\frac{1}{\sigma(z^{(i)})}\sigma(z^{(i)})(1-\sigma(z^{(i)}))\boldsymbol{x}^{(i)} = y^{(i)}(1-\sigma(z^{(i)}))\boldsymbol{x}^{(i)}$
- Derivative of the second term: $\frac{\partial}{\partial\boldsymbol{\beta}}\left[(1-y^{(i)})\log(1-\sigma(z^{(i)}))\right] = (1-y^{(i)})\frac{1}{1-\sigma(z^{(i)})}(-1)\frac{\partial\sigma(z^{(i)})}{\partial z^{(i)}}\frac{\partial z^{(i)}}{\partial\boldsymbol{\beta}} = (1-y^{(i)})\frac{1}{1-\sigma(z^{(i)})}(-1)\sigma(z^{(i)})(1-\sigma(z^{(i)}))\boldsymbol{x}^{(i)} = -(1-y^{(i)})\sigma(z^{(i)})\boldsymbol{x}^{(i)} = y^{(i)}\sigma(z^{(i)})\boldsymbol{x}^{(i)} - \sigma(z^{(i)})\boldsymbol{x}^{(i)}$

- If we set gradient to 0, we have expectation matching: $\sum_{i=1}^n \sigma(z^{(i)})\boldsymbol{x}^{(i)} = \sum_{i=1}^n y^{(i)}\boldsymbol{x}^{(i)}$ resp. optimum is where expected feature counts, weighted by predicted probability = observed feature counts, weighted by true labels

Characteristics —

- Convex (since sum of convex functions in log loss is convex) with psd Hessian
- Has global minimum
- Has unique or infinitely many solutions
- Can be solved numerically

Proof that log loss is convex:

- Sum of convex functions is convex
- Thus, we need to prove convexity of $\ln(1+e^{\boldsymbol{\beta}\cdot\boldsymbol{x}^{(i)}})$ and $-y^{(i)}\boldsymbol{\beta}\cdot\boldsymbol{x}^{(i)}$
- For second term:
  - $\mathcal{H}(\boldsymbol{\beta}) = \nabla^2_{\boldsymbol{\beta}}\left(-y^{(i)}\boldsymbol{\beta}\cdot\boldsymbol{x}^{(i)}\right) = 0$
  - $\mathcal{H} \geqslant 0$
- For first term:
  - $\mathbb{H}(\boldsymbol{\beta}) = \nabla^2_{\boldsymbol{\beta}}\ln(1+e^{\boldsymbol{\beta}\cdot\boldsymbol{x}^{(i)}})$
  - $= v(x)\times u'(x) - v'(x)\times u(x) = \frac{1}{1+e^{\boldsymbol{\beta}\cdot\boldsymbol{x}}}\times e^{\boldsymbol{\beta}\cdot\boldsymbol{x}}\times\boldsymbol{x}\boldsymbol{x}^\intercal - \frac{e^{\boldsymbol{\beta}\cdot\boldsymbol{x}}\times\boldsymbol{x}}{(1+e^{\boldsymbol{\beta}\cdot\boldsymbol{x}})^2}\times\boldsymbol{x}^\intercal\times e^{\boldsymbol{\beta}\cdot\boldsymbol{x}}$
  - $= \frac{e^{\boldsymbol{\beta}\cdot\boldsymbol{x}}\boldsymbol{x}\boldsymbol{x}^\intercal(1+e^{\boldsymbol{\beta}\cdot\boldsymbol{x}}) - e^{\boldsymbol{\beta}\cdot\boldsymbol{x}}\boldsymbol{x}\boldsymbol{x}^\intercal e^{\boldsymbol{\beta}\cdot\boldsymbol{x}}}{(1+e^{\boldsymbol{\beta}\cdot\boldsymbol{x}})^2}$
  - $= \frac{e^{\boldsymbol{\beta}\cdot\boldsymbol{x}}\boldsymbol{x}\boldsymbol{x}^T}{(1+e^{\boldsymbol{\beta}\cdot\boldsymbol{x}})^2}$
  - $\mathcal{H} \geqslant 0$

    Proof: $\boldsymbol{a}^T\mathcal{H}\boldsymbol{a} = \frac{e^{\boldsymbol{\beta}\cdot\boldsymbol{x}}}{(1+e^{\boldsymbol{\beta}\cdot\boldsymbol{x}})^2}\boldsymbol{a}^T\boldsymbol{x}\boldsymbol{x}^T\boldsymbol{a} = \frac{e^{\boldsymbol{\beta}\cdot\boldsymbol{x}}}{(1+e^{\boldsymbol{\beta}\cdot\boldsymbol{x}})^2}\|\boldsymbol{a}^T\boldsymbol{x}\|^2 \geq 0$

Regularization —

- Perfectly separable data requires regularization
- Let weights for each class k be scaled by c as $c\tilde{\boldsymbol{\beta}}_k$
- Gradient of log-loss with respect to c is always negative, causing gradient descent to grow c without bound

  Proof:
  - Log loss $= \sum_{i=1}^n \ln\left(1+e^{c\tilde{\boldsymbol{\beta}}\cdot\boldsymbol{x}^{(i)}}\right) - y^{(i)}c\tilde{\boldsymbol{\beta}}\cdot\boldsymbol{x}^{(i)}$
  - ∇_c log loss $= \sum_{i=1}^n \frac{1}{1+e^{c\tilde{\boldsymbol{\beta}}\cdot\boldsymbol{x}^{(i)}}}\times e^{c\tilde{\boldsymbol{\beta}}\cdot\boldsymbol{x}^{(i)}}\times\tilde{\boldsymbol{\beta}}\cdot\boldsymbol{x}^{(i)} - y^{(i)}\tilde{\boldsymbol{\beta}}\cdot\boldsymbol{x}^{(i)}$
  - $= \sum_{i=1}^n \tilde{\boldsymbol{\beta}}\cdot\boldsymbol{x}^{(i)}\left(\frac{e^{c\tilde{\boldsymbol{\beta}}\cdot\boldsymbol{x}^{(i)}}}{1+e^{c\tilde{\boldsymbol{\beta}}\cdot\boldsymbol{x}^{(i)}}} - y^{(i)}\right)$
  - Given perfect separation:
    - If $y^{(i)} = 1$, $\tilde{\boldsymbol{\beta}}\cdot\boldsymbol{x}^{(i)} > 0$, $\frac{e^{c\tilde{\boldsymbol{\beta}}\cdot\boldsymbol{x}^{(i)}}}{1+e^{c\tilde{\boldsymbol{\beta}}\cdot\boldsymbol{x}^{(i)}}} - y^{(i)} < 0$
    - If $y^{(i)} = 0$, $\tilde{\boldsymbol{\beta}}\cdot\boldsymbol{x}^{(i)} < 0$, $\frac{e^{c\tilde{\boldsymbol{\beta}}\cdot\boldsymbol{x}^{(i)}}}{1+e^{c\tilde{\boldsymbol{\beta}}\cdot\boldsymbol{x}^{(i)}}} - y^{(i)} > 0$
  - Thus, for all i, ∇_c log loss < 0
  - Thus gradient descent will cause c to grow without bound

26 Multinomial Logistic Regression

Description

Task — Multiclass classification

Description —

- Supervised
- Parametric

Formulation

- Probability of each class is estimated via the softmax function (generalizes the sigmoid function to multiple classes): $P(y=k \mid \boldsymbol{x}) = \frac{e^{f_i(x)/T}}{\sum_{j=1}^k e^{f_j(x)/T}} = \frac{e^{\boldsymbol{\beta}_k\cdot\boldsymbol{x}/T}}{\sum_{j=1}^k e^{\boldsymbol{\beta}_j\cdot\boldsymbol{x}/T}}$ where T is the temperature and allows to smoothly interpolate between the differentiable softmax ($T=1$) and the non-differentiable argmax ($T=0$)
- The predicted class is the one with the highest probability: $\hat{y} = \arg\max_k P(y=k \mid \boldsymbol{x})$
- Geometrically, the softmax defines $k-1$ linear separating hyperplanes

Proof of softmax function:

- Take class k as reference class
- We start with log-odds:
  - $\log\left(\frac{P_y(y=i \mid \boldsymbol{x})}{P_y(y=k \mid \boldsymbol{x})}\right) = f_i(\boldsymbol{x}) - f_k(\boldsymbol{x}) = (\boldsymbol{\beta}_i - \boldsymbol{\beta}_k)\cdot\boldsymbol{x} + (\beta_{i0} - \beta_{K0})$
  - $\frac{P_y(y=i \mid \boldsymbol{x})}{P_y(y=k \mid \boldsymbol{x})} = \exp(f_i(\boldsymbol{x}) - f_k(\boldsymbol{x})) = \exp((\boldsymbol{\beta}_i - \boldsymbol{\beta}_k)\cdot\boldsymbol{x} + (\beta_{i0} - \beta_{K0}))$
  - $\sum_{i=1}^{k-1}(\frac{P_y(y=i \mid \boldsymbol{x})}{P_y(y=k \mid \boldsymbol{x})}) = \frac{1-P_y(y=k \mid \boldsymbol{x})}{P_y(y=k \mid \boldsymbol{x})} = \sum_{i=1}^{k-1}\exp(f_i(\boldsymbol{x}) - f_k(\boldsymbol{x}))$
- We can re-form last equation to posterior: $P_y(y=k \mid \boldsymbol{x}) = \frac{1}{1+\sum_{i=1}^{k-1}\exp(f_i(\boldsymbol{x}) - f_k(\boldsymbol{x}))}$
- We can re-form secondlast equation to posterior:
  - $P_y(y=i \mid \boldsymbol{x}) = 1 - \frac{1}{1+\sum_{i=1}^{k-1}\exp(f_i(\boldsymbol{x}) - f_k(\boldsymbol{x}))}$
  - $= \frac{\exp(f_i(\boldsymbol{x}) - f_k(\boldsymbol{x}))}{1+\sum_{i=1}^{k-1}\exp(f_i(\boldsymbol{x}) - f_k(\boldsymbol{x}))}$
  - $= \frac{\frac{\exp(f_i(\boldsymbol{x}))}{\exp(f_k(\boldsymbol{x}))}}{1+\sum_{i=1}^{k-1}\frac{\exp(f_i(\boldsymbol{x}))}{\exp(f_k(\boldsymbol{x}))}}$
  - $= \frac{\frac{\exp(f_i(\boldsymbol{x}))}{\exp(f_k(\boldsymbol{x}))}}{\sum_{i=1}^{k-1}\frac{\exp(f_k(\boldsymbol{x}))+\exp(f_i(\boldsymbol{x}))}{\exp(f_k(\boldsymbol{x}))}}$
  - $= \frac{\exp(f_i(\boldsymbol{x}))\times\exp(f_k(\boldsymbol{x}))}{\exp(f_k(\boldsymbol{x}))\times\sum_{i=1}^{k-1}(\exp(f_k(\boldsymbol{x}))+\exp(f_i(\boldsymbol{x})))}$
  - $= \frac{\exp(f_i(\boldsymbol{x}))}{\sum_{j=1}^k\exp(f_j(\boldsymbol{x}))}$

Proof that softmax $\to$ argmax if $\tau \to 0$:

- Assume $\boldsymbol{x} = [x_1, x_2]^\intercal$
- $\lim_{\tau\to0} p(\boldsymbol{x}) = \lim_{\tau\to0}\frac{e^{x_1/\tau}}{e^{x_1/\tau}+e^{x_2/\tau}}$
- $= \lim_{\tau\to0}\frac{e^{x_1/\tau}e^{-x_1/\tau}}{(e^{x_1/\tau}+e^{x_2/\tau})e^{-x_1/\tau}}$
- $= \lim_{\tau\to0}\frac{1}{1+e^{(x_2-x_1)/\tau}}$
- $\lim_{\tau\to0} e^{(x_2-x_1)/\tau} = \begin{cases}0 & \text{if } x_1 > x_2\\ 1 & \text{if } x_1 = x_2\\ \infty & \text{otherwise}\end{cases}$
- Plugging back into $\lim_{\tau\to0} p(\boldsymbol{x}) = \frac{1}{1+e^{(x_2-x_1)/\tau}}$: $p(\boldsymbol{x}) = \begin{cases}[1,0]^\intercal & \text{if } x_1 > x_2\\ [0.5,0.5]^\intercal & \text{if } x_1 = x_2\\ [0,1]^\intercal & \text{otherwise}\end{cases}$

Optimization

Parameters — Find parameters $\boldsymbol{\beta}_1,\ldots,\boldsymbol{\beta}_k$

Objective function —

- Likelihood: $L(\boldsymbol{\beta}) = \prod_{i=1}^n P(y^{(i)} \mid \boldsymbol{x}^{(i)};\boldsymbol{\beta}) =$

$\prod_{i=1}^{n}\prod_{\ell=1}^{k}(\frac{e^{\beta_\ell\cdot x^{(i)}}}{\sum_{j=1}^{k}e^{\beta_j\cdot x^{(i)}}})^{\delta\{y^{(i)}=\ell\}}$

- Maximize log-likelihood: $\log L(\beta)=\sum_{i=1}^{n}\sum_{\ell=1}^{k}\delta\{y^{(i)}=\ell\}[\beta_\ell\cdot x^{(i)}-\log(\sum_{j=1}^{k}e^{\beta_j\cdot x^{(i)}})]$
- Minimize log-loss: $-\log L(\beta)$

Optimization —

- $\frac{\partial-\log L(\beta)}{\partial\beta_k}=-\sum_{i=1}^{n}\sum_{\ell=1}^{k}\frac{\partial}{\partial\beta_k}[\delta\{y^{(i)}=\ell\}[\beta_\ell\cdot x^{(i)}-\log(\sum_{j=1}^{k}e^{\beta_j\cdot x^{(i)}})]]=-\sum_{i=1}^{n}\delta\{y^{(i)}=k\}x^{(i)}-P(y=k|x^{(i)})x^{(i)}$

  Proof:
  - Derivative of the first term: $\frac{\partial}{\partial\beta_k}(\sum_{\ell=1}^{k}\delta\{y^{(i)}=\ell\}\beta_\ell\cdot x^{(i)})=\delta\{y^{(i)}=k\}x^{(i)}$
  - Derivative of the second term: $\frac{\partial}{\partial\beta_k}(-\log(\sum_{j=1}^{k}e^{\beta_j\cdot x^{(i)}}))=-\frac{1}{\sum_{j=1}^{k}e^{\beta_j\cdot x^{(i)}}}\frac{\partial}{\partial\beta_k}(\sum_{j=1}^{k}e^{\beta_j\cdot x^{(i)}})=-\frac{e^{\beta_k\cdot x^{(i)}}}{\sum_{j=1}^{k}e^{\beta_j\cdot x^{(i)}}}x^{(i)}=-P(y=k|x^{(i)})x^{(i)}$
  - For reference: Softmax derivative if $\ell=k$: $\frac{\partial P(y=\ell|x)}{\partial\beta_k}$:
    - Using the quotient rule: $\frac{\partial P(y=\ell|x)}{\partial\beta_\ell}=\frac{\frac{\partial}{\partial\beta_\ell}e^{\beta_\ell\cdot x}(\sum_{j=1}^{k}e^{\beta_j\cdot x})-e^{\beta_\ell\cdot x}\frac{\partial}{\partial\beta_\ell}(\sum_{j=1}^{k}e^{\beta_j\cdot x})}{(\sum_{j=1}^{k}e^{\beta_j\cdot x})^2}$
    - $\frac{\partial}{\partial\beta_\ell}e^{\beta_\ell\cdot x}=\frac{\partial}{\partial\beta_\ell}(\sum_{j=1}^{k}e^{\beta_j\cdot x})=e^{\beta_\ell\cdot x}x$
    - After plugging this in, we get: $\frac{\partial P_\ell}{\partial\beta_\ell}=\frac{e^{\beta_\ell\cdot x}(\sum_{j=1}^{k}e^{\beta_j\cdot x})-e^{\beta_\ell\cdot x}e^{\beta_\ell\cdot x}x}{(\sum_{j=1}^{k}e^{\beta_j\cdot x})^2}=P(y=\ell|x)(1-P(y=\ell|x))x$
  - For reference: Softmax derivative if $\ell\neq k$: $\frac{\partial P(y=\ell|x)}{\partial\beta_k}$:
    - Using the quotient rule: $\frac{\partial P(y=\ell|x)}{\partial\beta_k}=\frac{\frac{\partial}{\partial\beta_k}e^{\beta_\ell\cdot x}(\sum_{j=1}^{k}e^{\beta_j\cdot x})-e^{\beta_\ell\cdot x}\frac{\partial}{\partial\beta_k}(\sum_{j=1}^{k}e^{\beta_j\cdot x})}{(\sum_{j=1}^{k}e^{\beta_j\cdot x})^2}$
    - First term vanishes, since it does not depend on β_k
    - $\frac{\partial}{\partial\beta_k}(\sum_{j=1}^{k}e^{\beta_j\cdot x})=e^{\beta_k\cdot x}x$
    - After plugging this in, we get: $\frac{\partial P_\ell}{\partial\beta_k}=\frac{-e^{\beta_\ell\cdot x}e^{\beta_k\cdot x}x}{(\sum_{j=1}^{k}e^{\beta_j\cdot x})^2}=-P(y=\ell|x)P(y=k|x)x$
- If we set gradient to 0, we have expectation matching:
  - $\sum_{i=1}^{n}\delta\{y^{(i)}=k\}x^{(i)}=\sum_{i=1}^{n}P(y=k|x^{(i)})x^{(i)}$
  - $x^{(l)}=\mathbb{E}_{j\sim\text{softmax}}[x^{(l)}]$
  - Optimum is where observed features = expected features

Characteristics —

- Convex (sum of convex functions in log-loss is convex) with psd Hessian
- Has global minimum
- Has unique or infinitely many solutions (depending on feature linear independence)
- Can be solved numerically

27 SVM Classifier

General

Description

Task — Classification

Description —

- Supervised
- Parametric

Hard-Margin SVM Classifier

Formulation

- Assume $y\in\{-1,1\}$
- Discriminant:
  - $f=sgn(\beta\cdot x+b)$
  - If sign is positive, f outputs 1, else -1
  - Separating hyperplane given by $z=\beta\cdot x+b=0$
  - Is a linear discriminant
  - $z\perp\beta$
- For some point $\tilde{x}$ closest to the origin:
  - The perpendicular distance to the origin is given by: $\beta\cdot\tilde{x}+b=0$ since $\tilde{x}$ lies on the separating hyperplane
  - Then, $\|\beta\|\,\|\tilde{x}\|cos(\varphi)+b=\|\beta\|\,\|\tilde{x}\|(-1)+b=0$ because $\varphi=180$ degrees
  - Then, $\|\tilde{x}\|=\frac{b}{\|\beta\|}$
- For some point $x^{(i)}$ above z:
  - Projection of instance onto direction of β: $x^{(i)'}=\frac{x^{(i)}\cdot\beta}{\|\beta\|^2}\beta$
  - Distance of projection to the origin is given by $\|x^{(i)'}\|=cos(\varphi^{(i)})\|x^{(i)}\|=\frac{cos(\varphi^{(i)})\|x^{(i)}\|\,\|\beta\|}{\|\beta\|}=\frac{\beta\cdot x^{(i)}}{\|\beta\|}$
  - Margin $\gamma^{(i)}$ of instance given by: $\gamma^{(i)}=\|x^{(i)'}\|+\|\tilde{x}\|=\frac{\beta\cdot x^{(i)}+b}{\|\beta\|}$
- For some point $x^{(i)}$ below z:
  - Margin $\gamma^{(i)}$ of instance given by: $\gamma^{(i)}=-\frac{\beta\cdot x^{(i)}+b}{\|\beta\|}$
- For well-classified points, $\gamma^{(i)}>0$, for mis-classified points, $\gamma^{(i)}<0$
- Given that $y\in\{-1,1\}$ and thus $y^{(i)}(\beta\cdot x+b)>0$: $\gamma^{(i)}=\frac{y^{(i)}(\beta\cdot x^{(i)}+b)}{\|\beta\|}$
- Margin of system defined by smallest margin for instance: $\gamma=min_i\gamma^{(i)}=\frac{1}{\|\beta\|}min_i y^{(i)}(\beta\cdot x^{(i)}+b)$
- Margin is invariate to scaling of β and b
- Thus, we can write:
  - $min_i\gamma^{(i)}=min_i y^{(i)}(\beta\cdot x^{(i)}+b)=1$
  - Then, $y^{(i)}(\beta\cdot x^{(i)}+b)\geq 1$ (resp. $\geq m$ without scaling)
  - Moreover, since margin for system is defined by smallest margin for instance, $\gamma=\frac{1}{\|\beta\|}$

Optimization

Parameters — Find parameters β and b

Objective function —

- Objective function: $\gamma=\frac{1}{\|\beta\|}$ subject to $y^{(i)}(\beta\cdot x^{(i)}+b)\geq 1$
- Equivalent cost function: $\gamma=\frac{1}{2}\|\beta\|^2$ subject to $1-y^{(i)}(\beta\cdot x^{(i)}+b)\leq 0$
- Cost function in Lagrangian formulation: $\mathcal{L}=\frac{1}{2}\|\beta\|^2+\sum_{i=1}^{n}\alpha^{(i)}(1-y^{(i)}(\beta\cdot x^{(i)}+b))$
- By Slater's condition (linear separability), strong duality holds
- Objective function in dual Lagrangian: $\mathcal{D}=\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha^{(i)}\alpha^{(j)}y^{(i)}y^{(j)}x^{(i)}\cdot x^{(j)}+\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha^{(i)}-\alpha^{(i)}\alpha^{(j)}y^{(i)}y^{(j)}x^{(i)}\cdot x^{(j)}=\sum_{i=1}^{n}\alpha^{(i)}-\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha^{(i)}\alpha^{(j)}y^{(i)}y^{(j)}x^{(i)}\cdot x^{(j)}$

Optimization —

- General solution:
  - $\nabla_\beta\mathcal{L}=\beta-\sum_{i=1}^{n}\alpha^{(i)}y^{(i)}x^{(i)}=0$
  - $\Rightarrow\beta^*=\sum_{i=1}^{n}\alpha^{(i)}y^{(i)}$
  - $\nabla_b\mathcal{L}=-\sum_{i=1}^{n}\alpha^{(i)}y^{(i)}=0$
  - Subject to:
    - $1-y^{(i)}(\beta\cdot x^{(i)}+b)\leq 0$
    - $\alpha^{(i)}\geq 0$
    - $\alpha^{(i)}(1-y^{(i)}(\beta\cdot x^{(i)}+b))=0$
- Dual optimization: Maximize α subject to
  - $\alpha^{(i)}\geq 0$
  - $\sum_{i=1}^{n}\alpha^{(i)}y^{(i)}=0$ due to $\nabla_b\mathcal{L}$
- Note that only support vectors ($\alpha^{(i)}>0$, sit on the hyperplane $1=y^{(i)}(\beta\cdot x^{(i)}+b)=1$) matter in establishing β^*:
  - Based on complementary slackness condition $\alpha^{(i)}(1-y^{(i)}(\beta\cdot x^{(i)}+b))=0$: We either have
    - $\alpha^{(i)}=0$ and $(1-y^{(i)}(\beta\cdot x^{(i)}+b))>0$ resp. $y^{(i)}(\beta\cdot x^{(i)}+b)>1$ or
    - $\alpha^{(i)}>0$ and $(1-y^{(i)}(\beta\cdot x^{(i)}+b))=0$ resp. $y^{(i)}(\beta\cdot x^{(i)}+b)=1$

Characteristics —

- Strictly convex with psd Hessian
- Has global minimum
- Has unique solution

Soft-Margin SVM Classifier

Optimization

Objective function —

- Cost function: Hinge loss: $max(0,1-\gamma^{(i)})$
- Mis-classified instances incur a loss
- Well-classified instances incur a loss, if their margin $\gamma^{(i)}<1$
- Always is equal to or dominates the plain misclassification error
- To translate hinge loss into inequality constraint, we introduce slack variables $\xi^{(i)}$:
  - $y^{(i)}(\beta\cdot x^{(i)}+b)\geq 1-\xi^{(i)}$
  - By setting $\xi=0$, we get pulled down towards hinge loss
  - For
    - Well-classified points outside of margin $\xi^{(i)}<0$
    - Well-classified points within of margin $0<\xi^{(i)}<1$
    - Points on decision boundary $\xi^{(i)}=1$
    - Mis-classified points $\xi^{(i)}>1$
- We can then write cost function as slack variables penalized by ℓ_1 norm: $\frac{1}{2}\|\beta\|^2+C\sum_{i=1}^{n}\xi^{(i)}$ where
  - $\|\beta\|^2$ maximizes margin and $C\sum_{i=1}^{n}\xi^{(i)}$ minimizes hinge loss
  - C is a hyperparameter that determines how tolerant we are of margin errors: If C is large, we are less tolerant, margin will decrease, and the soft-margin will become a hard-margin.

  subject to:
  - $1-y^{(i)}(\beta\cdot x^{(i)}+b)\leq\xi^{(i)}$ resp. $\xi^{(i)}+y^{(i)}(\beta\cdot x^{(i)}+b)-1\geq 0$
  - $\xi^{(i)}\geq 0$
- Cost function in Lagrangian formulation: $\mathcal{L}=\frac{1}{2}\|\beta\|^2-\sum_{i=1}^{n}\alpha^{(i)}(y^{(i)}(\beta\cdot x^{(i)}+b)-1+\xi^{(i)})-\sum_{i=1}^{n}\zeta^{(i)}\xi^{(i)}+C\sum_{i=1}^{n}\xi^{(i)}$
- By Slater's condition (linear separability), strong duality holds
- Objective function in dual Lagrangian: $\mathcal{D}=\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha^{(i)}\alpha^{(j)}y^{(i)}y^{(j)}x^{(i)}\cdot x^{(j)}+\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha^{(i)}-\alpha^{(i)}\alpha^{(j)}y^{(i)}y^{(j)}x^{(i)}\cdot x^{(j)}=\sum_{i=1}^{n}\alpha^{(i)}-\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha^{(i)}\alpha^{(j)}y^{(i)}y^{(j)}x^{(i)}\cdot x^{(j)}$

Optimization —

- General solution:
  - $\nabla_\beta\mathcal{L}=\beta-\sum_{i=1}^{n}\alpha^{(i)}y^{(i)}x^{(i)}=0$
  - $\Rightarrow\beta^*=\sum_{i=1}^{n}\alpha^{(i)}y^{(i)}x^{(i)}$
  - $\nabla_b\mathcal{L}=-\sum_{i=1}^{n}\alpha^{(i)}y^{(i)}=0$
  - $\nabla_{\xi^{(i)}}\mathcal{L}=C-\alpha^{(i)}-\zeta^{(i)}=0$
  - Subject to:
    - $-\xi^{(i)}-y^{(i)}(\beta\cdot x^{(i)}+b)+1\leq 0$
    - $\xi^{(i)}\leq 0$
    - $\alpha^{(i)},\zeta^{(i)}\geq 0$
    - $\alpha^{(i)}(-\xi^{(i)}-y^{(i)}(\beta\cdot x^{(i)}+b)+1)=0$
    - $\zeta^{(i)}(-\xi^{(i)})=0$
- Dual optimization: Maximize α subject to
  - $\alpha^{(i)},\zeta^{(i)}\geq 0$
  - $\sum_{i=1}^{n}\alpha^{(i)}y^{(i)}=0$ due to $\nabla_b\mathcal{L}$
  - $0\leq\alpha^{(i)}\leq C$ due to $\nabla_{\xi^{(i)}}\mathcal{L}$
- Note that only support vectors ($\alpha^{(i)}>0$, sit in or on the hyperplane $1\geq y^{(i)}(\beta\cdot x^{(i)}+b)=1$) matter in establishing β^*:
  - Based on complementary slackness condition $\alpha^{(i)}(1-y^{(i)}(\beta\cdot x^{(i)}+b))=0$: We either have
    - $\alpha^{(i)}=0$ and $(1-y^{(i)}(\beta\cdot x^{(i)}+b))>0$ resp. $y^{(i)}(\beta\cdot x^{(i)}+b)>1$ or
    - $\alpha^{(i)}>0$ and $(1-y^{(i)}(\beta\cdot x^{(i)}+b))=0$ resp. $y^{(i)}(\beta\cdot x^{(i)}+b)=1$
  - Similarly, we either have
    - $\zeta^{(i)}=0$ and $-\xi^{(i)}<0$ resp. $\xi^{(i)}>0$ or
    - $\zeta^{(i)}>0$ and $-\xi^{(i)}=0$ resp. $\xi^{(i)}=0$
  - Then, each instance lies in one of three areas:
    - Beyond γ:
      - $\alpha^{(i)}=0$
      - $C=\zeta^{(i)}$ due to $\nabla_{\xi^{(i)}}\mathcal{L}$
      - $\xi^{(i)}=0$
      - $1<y^{(i)}(\beta\cdot x^{(i)}+b)$
    - On γ:
      - $\alpha^{(i)},\zeta^{(i)}>0$
      - $0<\alpha^{(i)}<C$ due to $\nabla_{\xi^{(i)}}\mathcal{L}$
      - $\xi^{(i)}=0$
      - $1=y^{(i)}(\beta\cdot x^{(i)}+b)$
    - Within γ:
      - $\alpha^{(i)}>0$
      - $\zeta^{(i)}=0$
      - $\alpha^{(i)}=C$ due to $\nabla_{\xi^{(i)}}\mathcal{L}$
      - $\xi^{(i)}>0$
      - $1>y^{(i)}(\beta\cdot x^{(i)}+b)$

Characteristics —

- Strictly convex with psd Hessian
- Has global minimum
- Has unique solution

28 Decision Tree for Regression

Description

Task — Regression

Description —

- Supervised
- Non-parametric

Formulation

- Predictor space is split along $J-1$ internal nodes
- Thus, we get J distinct regions resp. terminal nodes $R_1,...,R_J$

Algorithm

- Splitting approach: Recursive binary splitting (CART):
  - Top-down: Successively splits the predictor space
  - Greedy: At each step, the best split is made
  - Steps:
    1. Select predictor x_j and threshold s such that splitting the predictor space into the regions $\{x_j\leq s\}$ and $\{x_j>s\}$ leads to the smallest MSE
    2. Repeat the process for one of the previously identified regions
    3. The process ends when the maximum depth or minimum number of observations is reached
- Prediction approach:
  - For each observation falling into R_j, the prediction is the mean or median of the response values for the training observations in R_j, denoted as $\hat{y}_{R_j}$
  - At each leaf node, 2 numbers are

stored: The sum of outputs and the number of samples

Optimization

Parameters — Find regions $R_1,...,R_J$

Objective function —

- Minimize $\sum_{j=1}^{J}\sum_{i\in R_j}(y_i-\hat{y}R_j)^2$

29 Decision Tree for Classification

Description

Task — Classification

Description —

- Supervised
- Non-parametric

Formulation

- Predictor space is split along $J-1$ internal nodes
- Thus, we get J distinct regions resp. terminal nodes $R_1,...,R_J$

Algorithm

- Splitting approach: Recursive binary splitting (CART):
  - Top-down: Successively splits the predictor space
  - Greedy: At each step, the best split is made
  - Steps:
    1. Select predictor x_j and threshold s such that splitting the predictor space into the regions $\{x_j \le s\}$ and $\{x_j > s\}$ leads to the smallest Gini-index or cross-entropy
    2. Repeat the process for one of the previously identified regions
    3. The process ends when the maximum depth or minimum number of observations is reached
- Prediction approach:
  - For each observation falling into R_j, the prediction is the most common response class for the training observations in R_j, denoted as $\hat{y}_{R_j}$
  - At each leaf node, k numbers are stored: Number of samples in each of the k classes

Optimization

Parameters — Find regions $R_1,...,R_J$

Objective function —

- Gini-index:
  - $G_j=\sum_{k=1}^{K}\hat{p}_{jk}(1-\hat{p}_{jk})$ where $\hat{p}_{jk}$ is the proportion of training observations in R_j that are from class k
  - G_j takes a small value when nodes are pure, i.e., $\hat{p}_{jk}$ is 0 or 1
- Cross-entropy:
  - $D_j=-\sum_{k=1}^{K}\hat{p}_{jk}\log(\hat{p}_{jk})$ where $\hat{p}_{jk}$ is the proportion of training observations in R_j
  - Alternatively $J(\boldsymbol{B})=-\frac{1}{h}\sum_{i=1}^{h}\sum_{k=1}^{K}y_i^{(k)}\log(\hat{p}_i^{(k)})$ where $y_i^{(k)}$ indicates whether the i^{th} training observation belongs to class k
  - D_j (or $J(\boldsymbol{B})$) takes a small value when nodes are pure, i.e., $\hat{p}_{jk}$ (or $\hat{p}_i^{(k)}$ is 0 or 1
- Information gain: Reduction in cross-entropy

30 K-Means Clustering

Description

Task — Clustering

Description —

- Unsupervised
- Non-parametric

Formulation

- We specify that we want k clusters in total
- Clusters $j=1,...,k$ defined by centroid $\boldsymbol{\mu}^{[j]}\in\mathbb{R}^m$
- Instances $i=1,...,n$ given by $\boldsymbol{x}^{(i)}$
- Each instance i has k indicator variables, which describe whether instance i is assigned to cluster j, given by $\{p^{i[j]}\}_{j=1}^{k}\in[0,1]$

Optimization

Parameters — Find centroids $\mu^{[j]}$, i.e. find cluster assignments (instance always assigned to closest cluster)

Objective function —

- Minimize distance between each instance and the centroid of its closest cluster: $j^*=argmin_l\|\boldsymbol{x}^{(i)}-\boldsymbol{\mu}^{[l]}\|^2$ with

$$p^{i[j]}=\begin{cases}1 & \text{if } j=j^*\\ 0 & \text{otherwise}\end{cases}$$

- Distortion function given by: $\Theta=\sum_{i=1}^{n}\sum_{j=1}^{k}p^{i[j]}\|\boldsymbol{x}^{(i)}-\boldsymbol{\mu}^{[j]}\|^2$

Optimization — Lloyd's algorithm:

1. Randomly initialize each $\boldsymbol{\mu}^{[j]}$
2. E-Step:
   - Re-assign instances while keeping all centroids fixed, i.e. minimize Θ with respect to $p^{i[j]}$
3. M-Step:
   - Re-compute centroids while keeping all instance assignments fixed, i.e. minimize Θ with respect to $\boldsymbol{\mu}^{[j]}$
   - $\nabla_{\boldsymbol{\mu}^{[j]}}\Theta=2\sum_{i=1}^{n}p^{i[j]}(\boldsymbol{\mu}^{[j]}-\boldsymbol{x}^{(i)})=0$
   - Then, $\boldsymbol{\mu}^{[j]}=\frac{\sum_{i=1}^{n}p^{i[j]}\boldsymbol{x}^{(i)}}{\sum_{i=1}^{n}p^{i[j]}}$
   - Θ is strictly convex, thus, we find a global minimum and a unique solution here
4. Repeat E- and M-Step until convergence

Proof of convergence:

- $\Theta_t=\sum_{i=1}^{n}\sum_{j=1}^{k}p_t^{i[j]}\|\boldsymbol{x}^{(i)}-\boldsymbol{\mu}_t^{[j]}\|^2$
- In M-Step: $\Theta_t\le\sum_{i=1}^{n}\sum_{j=1}^{k}p_t^{i[j]}\|\boldsymbol{x}^{(i)}-\boldsymbol{\mu}_{t-1}^{[j]}\|^2$
- In E-Step: $\sum_{i=1}^{n}\sum_{j=1}^{k}p_t^{i[j]}\|\boldsymbol{x}^{(i)}-\boldsymbol{\mu}_{t-1}^{[j]}\|^2\le\sum_{i=1}^{n}\sum_{j=1}^{k}p_{t-1}^{i[j]}\|\boldsymbol{x}^{(i)}-\boldsymbol{\mu}_{t-1}^{[j]}\|^2=\Theta_{t-1}$
- Thus, $\Theta_t\le\Theta_{t-1}$

Characteristics —

- Challenges:
  - Has no way representing size of shape of a cluster, all that matters is distance
  - Hard cluster assignment, rather than soft cluster assignment, which implies that all instances in a cluster have an ëqual vote"within that cluster and no votes in any other cluster with regards to the assignment of a new point
- Not convex
- Has local minimum
- Has unique or infinitely many solutions
- Can be solved numerically

31 Non-Parametric Bayesian Clustering

Motivation

- In GMM, $\pi^{[i]}\sim\text{Dir}(\alpha_1,...,\alpha_k)$
- Dir is finite, i.e., all k clusters will be realized with probability 1
- Challenge: selecting k in advance
- Potential solution: Select $k\to\infty$, then only when $n\to\infty$ all clusters will be realized with probability 1
- Better solution: select $k=\infty$, this leads to non-parametric Bayesian models
- Question: How do we get infinite probabilities that sum to 1? We need:
  1. A suitable distribution
  2. A way to sample from it

Dirichlet process (DP)

- Distribution over probability distributions of a space Θ
- $\text{DP}(\alpha,H)$ where α is the concentration parameter and H is the base measure on Θ
- Sample $G\sim\text{DP}(\alpha,H)$ is a probability distribution

Stick Breaking Process

- Observation: Sampling $\pi^{[i]}\sim\text{Dir}(\alpha_1,...,\alpha_k)$ is equivalent to sampling: $\pi^{[1]}\sim\text{Beta}(\alpha_1,...,\alpha_k)$ and $(\pi^{[2]},...,\pi^{[k]})\sim\text{Dir}(\alpha_2,...,\alpha_k)$
- For finite length k, we have:
  - $\pi^{[1]}=\beta_1$ with $\beta_1\sim\text{Beta}(\alpha_1,...,\alpha_k)$
  - $\pi^{[2]}=(1-\beta_1)\beta_2$ with $\beta_2\sim\text{Beta}(\alpha_2,...,\alpha_k)$
  - $\pi^{[3]}=(1-\beta_1)(1-\beta_2)\beta_3$ with $\beta_3\sim\text{Beta}(\alpha_3,...,\alpha_k)$
- If we fix α, we have a revised form:
  - $\pi^{[1]}=\beta_1$ with $\beta_1\sim\text{Beta}(\alpha)$
  - $\pi^{[2]}=(1-\beta_1)\beta_2$ with $\beta_2\sim\text{Beta}(\alpha)$
  - $\pi^{[k]}=(1-\sum_{j=1}^{k-1}\pi^{[j]})\beta_k$ with $\beta_k\sim\text{Beta}(\alpha)$
- This is the GEM distribution: $\pi\sim\text{GEM}(\alpha)$ with $\pi=\{\pi^{[k]}\}_{k=1}^{\infty}$
- Connection to DP:
  - If $\pi\sim\text{GEM}(\alpha)$ and $\theta_k\sim H$, then $G(\theta)\sim\sum_{k=1}^{\infty}\pi^{[k]}\delta_{\theta_k}(\theta)$ is a sample from $\text{DP}(\alpha,H)$ and is a distribution over Θ
  - If we repeatedly sample $\theta^{[1]},\theta^{[2]},...$ from G, we have $\theta^{[i]}=\theta_{k_i}$ where value sometimes has not been observed before ($k_i\ne k_j$ for all $j<i$), sometimes value has been observed before ($k_i=k_j$ for some $j<i$)
  - $\theta^{[i]},\theta^{[j]}$ with $k_i=k_j$ belong to the same cluster

Chinese restaurant process (CRP)

- Customers are observations $\theta^{[i]}$
- Tables are clusters θ_k

Connection to clustering:

- When a new customer arrives, he either:
  - Joins an existing table with probability proportional to the number of customers already sitting there
  - Starts a new table with probability proportional to α
- Let $\mathcal{P}$ be the current seating arrangement
- The probability that customer $n+1$ joins table T is: $P(\text{customer } n+1 \text{ joins table } T\,|\,\mathcal{P})=\begin{cases}\frac{|T|}{\alpha+n} & \text{if table } T \text{ is in } \mathcal{P},\\ \frac{\alpha}{\alpha+n} & \text{otherwise}\end{cases}$
- The joint probability for the seating arrangement $\mathcal{P}$ is: $P(\mathcal{P})=P(\text{customer 1 joins table T})\times P(\text{customer 2, ...})=\frac{\alpha^{|\mathcal{P}|}}{\alpha^{(n)}}\prod_{T\in\mathcal{P}}(|T|-1)!$
- The expected number of tables is: $\mathbb{E}(|\mathcal{P}|)=\mathcal{O}(\alpha\log(N))$

Connection to DP:

- Note: CRP is order and labeling independent
- A sequence x is exchangeable if random vectors $(x_1,x_2,...)$ and $(x_{\tau(1)},x_{\tau(2)},...)$ have the same distribution, where $\tau(\cdot)$ is a random permutation
- According to *De Finetti's Theorem*: For an exchangeable sequence: $P(x_1,x_2,...)=\int\prod_i P(x_i|G)dP(G)$ where $G\sim\text{DP}(\alpha,H)$
- We can apply this theorem to CRP

CRP for continuous distributions:

- For continuous distributions, the probability of drawing an x_k that matches exactly one of the previous samples drawn is 0
- Thus: $p(x_k)=\frac{\alpha}{\alpha+k-1}$ and $\sum_{k=1}^{n}p(x_k)=\sum_{k=1}^{n}\frac{\alpha}{\alpha+k-1}=S(n)$
- We can approximate the sum $S(n)$ with the integral $I(n)$: $I(n)=\int_1^{n+1}\frac{\alpha}{\alpha+x-1}dx$
- Integral is bounded above by the sum: $I(n)\le S(n)$
- Integral is bounded below by: $I(n)\ge\sum_{k=2}^{n+1}\frac{\alpha}{\alpha+k-1}$ since the sum grows smaller as k grows larger. Therefore: $I(n)\ge S(n)-\frac{\alpha}{\alpha}+\frac{\alpha}{\alpha+n}=S'(n)$
- So we have: $S(n)-1+\frac{\alpha}{\alpha+n}\le I(n)\le S(n)$ which we can manipulate to: $I(n)+1-\frac{\alpha}{\alpha+n}\ge S(n)\ge I(n)$
- Computing the integral $I(n)$: $I(n)=\int_1^{n+1}\frac{\alpha}{\alpha+x-1}dx=F(n+1)-F(1)=\alpha(\ln(\alpha+n)-\ln(\alpha))$
- As $n\to\infty$, $I(n)\to\alpha(\ln(n))$

Thus, as $n\to\infty$, $S(n)\to\alpha(\ln(n))$

DP mixture model

- Let:
  - $\Theta=\mathbb{R}$, space of parameters that defines family of probability distributions
  - $H=\mathcal{N}(\mu_0,\sigma_0)$, base measure on Θ, representing prior belief over parameters
- On that basis, we define the DP mixture model:
  - Probabilities of clusters: $\pi=(\pi^{[1]},...)\sim\text{GEM}(\alpha)$
  - Cluster centers: $\mu_k\sim\mathcal{N}(\mu_0,\sigma_0)$
  - Assignments of datapoints to clusters: $z^{(i)}\sim\text{Categorical}(\pi)$
  - Coordinates of datapoints: $x^{(i)}\sim\mathcal{N}(\mu^{[z^{(i)}]},\sigma^{[z^{(i)}]})$
- Fitting the model:
  - Prior: Probability of cluster assignment, based on cluster size
  - Posterior: Probability of datapoint, given cluster center
  - Gibbs sampling:
    - Idea: Sample each variable in turn, conditioned on values of all other variables in the distribution
    - $P(z^{(i)}=k\,|\,\boldsymbol{z}^{(-i)},\boldsymbol{x},\alpha,\boldsymbol{\mu})\propto P(z^{(i)}=k\,|\,\boldsymbol{z}^{(-i)})P(\boldsymbol{x}\,|\,z^{(i)}=k,\boldsymbol{z}^{(-i)})$ due to Baye's rule
    - $\propto P(z^{(i)}=k\,|\,\boldsymbol{z}^{(-i)})P(x^{(i)}\,|\,z^{(i)}=k,\boldsymbol{z}^{(-i)},\boldsymbol{x}^{(-i)})P(\boldsymbol{x}^{(-i)}\,|\,z^{(i)}=k,\boldsymbol{z}^{(-i)})$ due to product rule
    - $\propto P(z^{(i)}=k\,|\,\boldsymbol{z}^{(-i)})P(x^{(i)}\,|\,z^{(i)}=k,\boldsymbol{z}^{(-i)},\boldsymbol{x}^{(-i)})$ because in last term $\boldsymbol{x}^{(-i)}\perp z^{(i)}\,|\,\boldsymbol{z}^{(-i)}$ by d-separation, so this term is constant with respect to k
    - We then have: $P(z^{(i)}=k\,|\,\boldsymbol{z}^{(-i)},\alpha)P(x^{(i)}\,|\,z^{(i)}=k,\boldsymbol{z}^{(-i)},\boldsymbol{x}^{(-i)},\boldsymbol{\mu})=$ prior $\times$ posterior, with interchangeable $z^{(1)},...,z^{(k)}$
    - Prior: Comes from CRP, given by: $P(z^{(i)})=k\,|\,\boldsymbol{z}^{(-i)},\alpha)=\begin{cases}\frac{N_{k,-i}}{\alpha+N-1} & \text{if } k \text{ is an existing cluster}\\ \frac{\alpha}{\alpha+N-1} & \text{otherwise}\end{cases}$
    - Posterior: We only have to consider points in x that are assigned to cluster k, given by: $P(z^{(i)}\,|\,z^{(i)}=k,\boldsymbol{z}^{(-i)},\boldsymbol{x}^{(-i)},\boldsymbol{\mu})=\begin{cases}P(x^{(i)}\,|\,\boldsymbol{x}^{(-i,k)},\boldsymbol{\mu})=\frac{P(x^{(i)},\boldsymbol{x}^{(-i,k)}|\boldsymbol{\mu})}{P(\boldsymbol{x}^{(-i,k)}|\boldsymbol{\mu})} & \text{for}\\ P(x^{(-i)}\,|\,\boldsymbol{\mu}) & \text{oth}\end{cases}$
    - Thus, Gibbs sampler: $P(z^{(i)}=k\,|\,\boldsymbol{z}^{(-i)},\boldsymbol{x},\alpha,\boldsymbol{\mu})=\begin{cases}\frac{N_{k,-i}}{\alpha+N-1}\times P(x^{(i)}\,|\,\boldsymbol{x}^{(-i,k)},\boldsymbol{\mu}) & \text{for existin}\\ \frac{\alpha}{\alpha+N-1}\times P(x^{(-i)}\,|\,\boldsymbol{\mu}) & \text{otherwise}\end{cases}$

Latent dirichlet allocation (LDA)

- Extension of DP Mixture Model where each $x^{(i)}$ can be assigned multiple clusters (multivariate), but we don't know in advance how many (non-parametric, as in DP Mixture Model)
- Distribution of topics in document d: $\theta_d\sim\text{Dir}(\alpha)$
- What topic word w belongs to in document d: $z^{(d,w)}\sim\text{Categorical}(\theta_d)$
- Distribution of words in topic k: $\varphi_k\sim\text{Dir}(\beta)$

- What is word w in document d: $w^{(d,w)} \sim \text{Categorical}(\varphi_{z^{(d,w)}})$
- α controls prior weights of topics in documents
- β controls prior weights of words in topics

32 Principal Component Analysis (PCA)

Maximize Variance Approach

Description

Task — Dimensionality reduction via projection, create uncorrelated features

Description —
- Unsupervised
- Non-parametric

Overview — Identifies lower-dimensional subspace and projects data onto it such that the maximum amount of variance in the data is preserved. In lower-dimensional subspace:
- Axes are called *principal components*, where the first principal component is the axis accounting for the largest variance
- Each axis is given by an eigenvector with *loadings*, indicating how much each variable in the original data contributes to this eigenvector
- Variance captured along each axis is given by the corresponding eigenvalue

Formulation

- Project data $\{x^{(i)}\}_{i=1}^n \in \mathbb{R}^m$ onto space $\mathbb{R}^d$ spanned by orthonormal basis $\{u^{[j]}\}_{j=1}^d \in \mathbb{R}^m$ where $d << m$
- Each instance $x^{(i)}$ is projected onto each basis vectors $u^{[j]} \cdot x^{(i)}$
- Each basis vector $u^{[j]}$ contains m loadings $[u_i^{[j]}, ..., u_m^{[j]}]$, whose value indicates how important each feature m is for the j^{th} principal component
- Mean of projected data for a given basis vector: $u^{[j]} \cdot \overline{X} = u^{[j]} \cdot \frac{1}{n} \sum_{i=1}^n x^{(i)}$
- Variance of projected data for a given basis vector: $\frac{1}{n} \sum_{i=1}^n (u^{[j]} \cdot x^{(i)} - u^{[j]} \cdot \overline{X})^2 = u^{[j]\top} S u^{[j]}$ where S is the covariance matrix $S = \frac{1}{n} \sum_{i=1}^n (x^{(i)} - \overline{X})(x^{(i)} - \overline{X})^\top = \frac{1}{n} X^\top X$

Optimization

Parameters — Find $\{u^{[j]}\}_{j=1}^d$

Objective function —
- Maximize variance $\sum_{j=1}^d u^{[j]\top} S u^{[j]}$ subject to orthonormal $\{u^{[j]}\}_{j=1}^d$
- Gives rise to Lagrangian formulation
- Lagrangian formulation for $u^{[1]}$ capturing the most variance: $\mathcal{L} = u^{[1]\top} S u^{[1]} - \lambda^{[1]}(u^{[1]} \cdot u^{[1]} - 1)$ where $\lambda^{[1]}$ captures the orthonormality constraint that $u^{[1]} \cdot u^{[1]} = 1$
- Lagrangian formulation for $u^{[2]}$ capturing the secondmost variance: $\mathcal{L} = u^{[2]\top} S u^{[2]} - \lambda^{[2]}(u^{[2]} \cdot u^{[2]} - 1) - \lambda^{[1][2]}(u^{[1]} \cdot u^{[2]} - 0)$ where $\lambda^{[1][2]}$ captures the orthogonality constraint that $u^{[1]} \cdot u^{[2]} = 0$

Optimization — For $u^{[1]}$:
- $\nabla_{u^{[1]}} \mathcal{L} = 2Su^{[1]} - 2\lambda^{[1]} u^{[1]} = 0$
- $\Rightarrow Su^{[1]} = \lambda^{[1]} u^{[1]}$
- This is the eigenvector/eigenvalue equation, so $u^{[1]}$ is the eigenvector of S and $\lambda^{[1]}$ is the associated eigenvalue
- We see that the variance of the projected data is equal to $\lambda^{[1]}$: $u^{[1]\top} S u^{[1]} = u^{[1]\top} \lambda^{[1]} u^{[1]} = \lambda^{[1]} u^{[1]\top} u^{[1]} = \lambda^{[1]} \times 1$

For $u^{[2]}$:
- $\nabla_{u^{[2]}} \mathcal{L} = 2Su^{[2]} - 2\lambda^{[2]} u^{[2]} - \lambda^{[1][2]} u^{[1]} = 0$
- $\Rightarrow Su^{[2]} = \lambda^{[2]} u^{[2]}$
  Proof:
  - Multiplying with $u^{[1]\top}$: $2u^{[1]\top} Su^{[2]} - 2\lambda^{[2]} u^{[1]\top} u^{[2]} - \lambda^{[1][2]} u^{[1]\top} u^{[1]} = 0$
  - $= 2u^{[1]\top} Su^{[2]} - 0 - \lambda^{[1][2]} \times 1 = 0$ because of orthogonality resp. orthonormality
  - $= 2u^{[2]\top} Su^{[1]} - \lambda^{[1][2]} = 0$ because the variance is a scalar and can be transposed and because the covariance matrix is symmetric
  - $= 2u^{[2]\top} \lambda^{[1]} u^{[1]} - \lambda^{[1][2]} = 0$ after plugging in the first found basis vector
  - $= 2\lambda^{[1]} \times 0 - \lambda^{[1][2]} = 0$
  - $= \lambda^{[1][2]} = 0$

... continue as for previous vector

In the end, we have a total projected variance of $\sum_{j=1}^d \lambda^{[j]}$

Characteristics —
- Convex
- Has global minimum
- Has unique or infinitely many solutions
- Can be solved analytically

SVD Approach

Formulation

- Project data $\{x^{(i)}\}_{i=1}^n \in \mathbb{R}^m$ onto space $\mathbb{R}^d$ spanned by orthonormal basis $\{u^{[j]}\}_{j=1}^d \in \mathbb{R}^m$ where $d << m$
- $\tilde{x}^{(i)} = \sum_{j=1}^d \alpha_{ij} u^{[j]} + \sum_{j=d+1}^m \gamma_j u^{[j]}$ where α_{ij} is particular to the instance, γ_j is the same for all instances and maps up to the subspace

Optimization

Objective function —
- Reconstruction error:
  - $J = \frac{1}{n} \sum_{i=1}^n \|x^{(i)} - \tilde{x}^{(i)}\|^2 = \frac{1}{n} \sum_{i=1}^n \|x^{(i)}\|^2 - \|BB^T x^{(i)}\|^2 = \frac{1}{n} \sum_{i=1}^n \|x^{(i)}\|^2 - \|B^T x^{(i)}\|^2$ where the last step follows since B is orthogonal
  - $x^{(i)}$ is a column of $X^\top$
  - If $A = X^\top$ and SVD of A is $USV^\top$, then B is given by $U^{(j \le d)}$, i.e. the first d columns of U
  - Then, reconstruction error is given by: $J = \frac{1}{n} \sum_{i=1}^n \|x^{(i)}\|^2 - \|B^T x^{(i)}\|^2 = \frac{1}{d} \sum_{i=1}^d \sigma_d^2$

33 Gaussian Mixture Models (GMM)

Description

Task — Clustering

Description —
- Unsupervised
- Non-parametric

Formulation

- Instances $\{x^{(i)}\}_{i=1}^n$
- Each instance has a latent cluster assignment given by: $z^{(i)} \in \{1, ..., k\}$
- Probability that cluster assigned to instance i is cluster j is given by: $\pi^{[j]} = p(z^{(i)} = j)$
- $z^{(i)} \sim \text{Categorical}(\pi^{[1]}, ..., \pi^{[k]})$
- $\pi^{[j]} \sim \text{Dir}(\alpha_1, ..., \alpha_k)$
- Contingent on cluster assignment, each instance is the outcome of a random variable associated with a given cluster: $x^{(i)}|z^{(i)} = j \sim \mathcal{N}(x^{(i)}|\mu^{[j]}, \Sigma^{[j]})$ where $\mu^{[j]}$ is the mean and $\Sigma^{[j]}$ is the covariance associated with cluster j
- Then, marginal distribution of each instance is given by: $p(x^{(i)}) = \sum_{j=1}^k p(x^{(i)}|z^{(i)}) p(z^{(i)}) = \sum_{j=1}^k \mathcal{N}(x^{(i)}|\mu^{[j]}, \Sigma^{[j]}) \times \pi^{[j]}$
- This is the GMM, characterized by parameters $\theta = \{\mu^{[j]}, \Sigma^{[j]}, \pi^{[j]}\}_{j=1}^k$

Optimization

Parameters — Find parameters $\theta = \{\mu^{[j]}, \Sigma^{[j]}, \pi^{[j]}\}_{j=1}^k$

Objective function —
- Maximize likelihood $L = \sum_{i=1}^n \log p(x^{(i)}) = \sum_{i=1}^n \log(\sum_{j=1}^k p(x^{(i)}|z^{(i)}) p(z^{(i)})) = \sum_{i=1}^n \log(\sum_{j=1}^k \mathcal{N}(x^{(i)}|\mu^{[j]}, \Sigma^{[j]}) \times \pi^{[j]})$ subject to $\sum_{j=1}^k \pi^{[j]} = 1$ and $\Sigma^{[j]} > 0$
- Challenge: This is a constrained, not concave, not analytically solvable optimization problem, due to the sum within the log
- Solution: Expectation maximization algorithm, which we now motivate
- Let's temporarily assume we know which cluster each instance is associated with, i.e. we know $z^{(i)}$
- We can rewrite likelihood from $L = \sum_{i=1}^n \log p_\theta(x^{(i)})$ to $L = \sum_{i=1}^n \log \sum_{z^{(i)}} p_\theta(x^{(i)}, z^{(i)})$
- For each instance, let $q_{z^{(i)}}$ be some probability distribution over $z^{(i)}$
- Then, we can further expand likelihood: $L = \sum_{i=1}^n \log \sum_{z^{(i)}} q(z^{(i)}) \frac{p_\theta(x^{(i)}, z^{(i)})}{q(z^{(i)})}$
- According to Jensen's inequality: $L = \sum_{i=1}^n \log \sum_{z^{(i)}} q(z^{(i)}) \frac{p_\theta(x^{(i)}, z^{(i)})}{q(z^{(i)})} \ge \sum_{i=1}^n \sum_{z^{(i)}} q(z^{(i)}) \log \frac{p_\theta(x^{(i)}, z^{(i)})}{q(z^{(i)})}$
- We have thus derived a lower bound for L
- We can tighten this bound and achieve equality by selecting $q_{z^{(i)}}$ accordingly: $q_{z^{(i)}} =$

  Proof 1:
  - $\sum_{i=1}^n \sum_{z^{(i)}} q(z^{(i)}) \log \frac{p_\theta(x^{(i)}, z^{(i)})}{q(z^{(i)})} = \sum_{i=1}^n \sum_{z^{(i)}} p_\theta(z^{(i)}|x^{(i)}) \log \frac{p_\theta(x^{(i)}, z^{(i)})}{p_\theta(z^{(i)}|x^{(i)})} = \sum_{i=1}^n \sum_{z^{(i)}} p_\theta(z^{(i)}|x^{(i)}) \log \frac{p_\theta(x^{(i)}) p_\theta(z^{(i)}|x^{(i)})}{p_\theta(z^{(i)}|x^{(i)})} = \sum_{i=1}^n \sum_{z^{(i)}} p_\theta(z^{(i)}|x^{(i)}) \log p_\theta(x^{(i)}) = \sum_{i=1}^n \times 1 \times \log p_\theta(x^{(i)}) = L$

  Proof 2:
  - $L = \mathbb{E}_q[log(p_\theta(x^{(i)}))] = \mathbb{E}_q[log(\frac{p_\theta(x^{(i)}, z^{(i)})}{p_\theta(z^{(i)}|x^{(i)})} \frac{q(z^{(i)})}{q(z^{(i)})})] = \mathbb{E}_q[log(\frac{p_\theta(x^{(i)}, z^{(i)})}{q(z^{(i)})})] + \mathbb{E}_q[log(\frac{q(z^{(i)})}{p_\theta(z^{(i)}|x^{(i)})})] = M + E$
- E corresponds to the KL divergence between $q(z^{(i)})$ and $p(z^{(i)}|x^{(i)})$
- $L \ge M \Leftrightarrow E \ge 0$, which we can show to be the case:
  - $E = \mathbb{E}_q[log(\frac{q(z^{(i)})}{p_\theta(z^{(i)}|x^{(i)})})] = \mathbb{E}_q[-log(\frac{p_\theta(z^{(i)}|x^{(i)})}{q(z^{(i)})})]$
  - According to Jensen's inequality: $E \ge -log(\mathbb{E}_q[\frac{p_\theta(z^{(i)}|x^{(i)})}{q(z^{(i)})}]) = -log(\sum_{i=1}^k q(z^{(i)}) \frac{p_\theta(z^{(i)}|x^{(i)})}{q(z^{(i)})}) = -log(\sum_{i=1}^k p_\theta(z^{(i)}|x^{(i)})) = -log(1) = 0$
- $L = M \Leftrightarrow E = 0$, i.e. when $q(z^{(i)}) = p_\theta(z^{(i)}|x^{(i)})$
- We can further specify $q_{z^{(i)}} = p_\theta(z^{(i)}|x^{(i)})$ to: $q_{z^{(i)}} = p_\theta(z^{(i)}|x^{(i)}) = \frac{p_\theta(x^{(i)}|z^{(i)}) p(z^{(i)})}{\sum_{j=1}^k p_\theta(x^{(i)}|z^{(i)}) p(z^{(i)})} = \frac{\mathcal{N}(x^{(i)}|\mu^{[j]}, \Sigma^{[j]}) \times \pi^{[j]}}{\sum_{j=1}^k \mathcal{N}(x^{(i)}|\mu^{[j]}, \Sigma^{[j]}) \times \pi^{[j]}} = \gamma^{i[j]}$

Optimization — Expectation maximization algorithm

1. Randomly initialize $\theta^{(t)} = \{\mu^{[j](t)}, \Sigma^{[j](t)}, \pi^{[j](t)}\}_{j=1}^k$
2. *E-step*: Minimize E, by computing $q(z^{(i)})$ given $x^{(i)}$ and $\theta^{(t)}$
3. *M-step*: Maximize M, by updating $\theta^{(t)}$ based on MLE for Gaussians, while keeping $q(z^{(i)})$ fixed:
   - $\mu^{[j](t+1)} = \frac{\sum_{i=1}^n q(z^{(i)}) x^{(i)}}{\sum_{i=1}^n q(z^{(i)})}$
   - $\Sigma^{[j](t+1)} = \frac{\sum_{i=1}^n q(z^{(i)})(x^{(i)} - \mu^{[j](t+1)})(x^{(i)} - \mu^{[j](t+1)})^\top}{\sum_{i=1}^n q(z^{(i)})}$
   - $\pi^{[j](t+1)} = \frac{1}{n} \sum_{i=1}^n q(z^{(i)})$
4. Repeat 2 and 3 until convergence

Further proofs — Recall: Likelihood is: $L = \sum_{i=1}^n \log\left(\sum_{j=1}^k \pi^{[j]} \mathcal{N}\left(x^{(i)}; \mu^{[j]}, \Sigma^{[j]}\right)\right)$

Proof of optimal mean in M-step:
- $\nabla_{\mu^{[j]}} L = \sum_{i=1}^n \frac{\delta L}{\delta \mathcal{N}(...)} \times \frac{\delta \mathcal{N}(...)}{\delta \mu^{[j]}} = \sum_{i=1}^n \frac{\pi^{[j]}}{\sum_{j=1}^k \pi^{[j]} \mathcal{N}\left(x^{(i)}; \mu^{[j]}, \Sigma^{[j]}\right)} \times \frac{\delta \mathcal{N}(...)}{\delta \mu^{[j]}} = \sum_{i=1}^n \frac{\pi^{[j]} \mathcal{N}\left(x^{(i)}; \mu^{[j]}, \Sigma^{[j]}\right)}{\sum_{j=1}^k \pi^{[j]} \mathcal{N}\left(x^{(i)}; \mu^{[j]}, \Sigma^{[j]}\right)} \times \frac{\delta \log \mathcal{N}\left(x^{(i)}; \mu^{[j]}, \Sigma^{[j]}\right)}{\delta \mu^{[j]}}$

  because $\frac{\delta \log \mathcal{N}(...)}{\delta \mu^{[j]}} = \frac{1}{\mathcal{N}(...)} \times \frac{\delta \mathcal{N}(...)}{\delta \mu^{[j]}}$

  $\Rightarrow \frac{\delta \mathcal{N}(...)}{\delta \mu^{[j]}} = \mathcal{N}(...) \times \frac{\delta \log \mathcal{N}(...)}{\delta \mu^{[j]}}$

  $= \sum_{i=1}^n \gamma^{i[j]} \times \frac{\delta \log\left(\text{constant} \times \frac{1}{|\Sigma|^{1/2}} \times \exp\left(-\frac{1}{2}(x^{(i)} - \mu^{[j]})^\top \Sigma^{-1}(x^{(i)} - \right.\right.}{\delta \mu^{[j]}}$

  $= \sum_{i=1}^n \gamma^{i[j]} \times \frac{\delta}{\delta \mu^{[j]}} \left(\log(1) - \frac{1}{2}\log(|\Sigma|) - \frac{1}{2}(x^{(i)} - \mu^{[j]})^\top \Sigma^{-1}(x^{(i)} - \right.$

  $= \sum_{i=1}^n \gamma^{i[j]} \times \left(-\frac{1}{2} \times \frac{\delta}{\delta \mu^{[j]}} \left[\log(|\Sigma|) + (x^{(i)} - \mu^{[j]})^\top \Sigma^{-1}(x^{(i)} - \mu\right.\right.$

  $= \sum_{i=1}^n \gamma^{i[j]} \times \left(-\frac{1}{2} \times 2\Sigma^{-1}(x^{(i)} - \mu^{[j]}) \times -1\right)$

  due to matrix calculus

  $= \sum_{i=1}^n \gamma^{i[j]} \Sigma^{-1}(x^{(i)} - \mu^{[j]}) = 0$
- $= \sum_{i=1}^n \gamma^{i[j]} \Sigma^{-1} x^{(i)} = \sum_{i=1}^n \gamma^{i[j]} \Sigma^{-1} \mu^{[j]}$
- $\mu^{[j]*} = \frac{\sum_{i=1}^n \gamma^{i[j]} x^{(i)}}{\sum_{i=1}^n \gamma^{i[j]}}$

Proof of optimal variance in M-step:
- $\nabla_{\Sigma^{[j]}} L = ...\text{as above} = \sum_{i=1}^n \gamma^{i[j]} \times \left(-\frac{1}{2} \times \frac{\delta}{\delta \Sigma^{[j]}} \left[\log(|\Sigma|) + (x^{(i)} - \mu^{[j]})^\top \Sigma^{-1}(x^{(i)} - \right.\right.$

  $= \sum_{i=1}^n \gamma^{i[j]} \times \left(-\frac{1}{2} \times (\Sigma^{-1} - \Sigma^{-1}(x^{(i)} - \mu^{[j]})(x^{(i)} - \mu^{[j]})^\top \Sigma^{-1}\right.$

  0 due to matrix calculus
- $\sum_{i=1}^n \gamma^{i[j]} \times \frac{1}{2} \times \Sigma^{-1} = \sum_{i=1}^n \gamma^{i[j]} \times \frac{1}{2} \times \Sigma^{-2}(x^{(i)} - \mu^{[j]})(x^{(i)} - \mu^{[j]})^\top$

  $\Rightarrow \sum_{i=1}^n \gamma^{i[j]} \Sigma = \sum_{i=1}^n \gamma^{i[j]} (x^{(i)} - \mu^{[j]})(x^{(i)} - \mu^{[j]})^\top$
- $\Sigma^{[j]*} = \frac{\sum_{i=1}^n \gamma^{i[j]} (x^{(i)} - \mu^{[j]})(x^{(i)} - \mu^{[j]})^\top}{\sum_{i=1}^n \gamma^{i[j]}}$

Proof of optimal π in M-step:
- Lagrangian optimization: $\arg\min_{\pi^{[j]}} L$ subject to $\sum_{j=1}^k \pi^{[j]} = 1$, i.e. $\sum_{j=1}^k \pi^{[j]} - 1 = 0$
- Lagrangian: $\mathcal{L}(\pi^{[j]}) = \sum_{i=1}^n \log\left(\sum_{j=1}^k \pi^{[j]} \mathcal{N}\left(x^{(i)}; \mu^{[j]}, \Sigma^{[j]}\right)\right) + \lambda\left(\sum_{j=1}^k \pi^{[j]} - 1\right)$
- Derivative: $\nabla_{\pi^{[j]}} \mathcal{L}(\pi^{[j]}) =$

$\sum_{i=1}^n \frac{\sum_{j=1}^k \mathcal{N}(\boldsymbol{x}^{(i)};\boldsymbol{\mu}^{[j]},\Sigma^{[j]})}{\sum_{j=1}^k \pi^{[j]}\mathcal{N}(\boldsymbol{x}^{(i)};\boldsymbol{\mu}^{[j]},\Sigma^{[j]})} + \lambda$

$= \sum_{i=1}^n \sum_{j=1}^k \frac{\gamma^{i[j]}}{\pi^{[j]}} + \lambda = 0$

- $\sum_{i=1}^n \sum_{j=1}^k \gamma^{i[j]} = -\lambda \sum_{j=1}^k \pi^{[j]}$
- Solving for λ:
  $\frac{\sum_{i=1}^n \sum_{j=1}^k \pi^{[j]}\mathcal{N}(\boldsymbol{x}^{(i)};\boldsymbol{\mu}^{[j]},\Sigma^{[j]})}{\sum_{i=1}^n \sum_{j=1}^k \pi^{[j]}\mathcal{N}(\boldsymbol{x}^{(i)};\boldsymbol{\mu}^{[j]},\Sigma^{[j]})} = -\lambda \times 1$ due to constraint $n = -\lambda$
- Solving for $\pi^{[j]}$: $\pi^{[j]} = \frac{\sum_{i=1}^n \gamma^{i[j]}}{-\lambda}$
  Plugging in λ: $\pi^{[j]*} = \frac{1}{n}\sum_{i=1}^n \gamma^{i[j]}$

Proof that M-step objective is concave:

- We aim top optimize $\log(p(\boldsymbol{x^{(i)}}, z^{(i)})) = \log(p(\boldsymbol{x^{(i)}}|z^{(i)})p(z^{(i)})) = \log(\mathcal{N}(\boldsymbol{x^{(i)}}|\boldsymbol{\mu^{[j]}}, \Sigma^{[j]})) + \log(\pi^{[j]})$
- This is a sum of concave functions

Proof that EM-algorithm converges:

- According to Jensen's inequality: $L_t \geq \sum_{i=1}^n \sum_{z^{(i)}} q_t(z^{(i)}) \log \frac{p_{\theta_t}(\boldsymbol{x^{(i)}}, z^{(i)})}{q_t(z^{(i)})}$
- In previous M-step, L was maximized by setting θ_t: $\theta_t = argmax_\theta (q_t(z^{(i)}) \log \frac{p_\theta(\boldsymbol{x^{(i)}}, z^{(i)})}{q_t(z^{(i)})})$
- Thus, $\sum_{i=1}^n \sum_{z^{(i)}} q_t(z^{(i)}) \log \frac{p_{\theta_t}(\boldsymbol{x^{(i)}}, z^{(i)})}{q_t(z^{(i)})} \geq \sum_{i=1}^n \sum_{z^{(i)}} q_t(z^{(i)}) \log \frac{p_{\theta_{t-1}}(\boldsymbol{x^{(i)}}, z^{(i)})}{q_t(z^{(i)})}$
- RHS of this equation is output of previous E-step, where q_t was set such that: $\sum_{i=1}^n \sum_{z^{(i)}} q_t(z^{(i)}) \log \frac{p_{\theta_{t-1}}(\boldsymbol{x^{(i)}}, z^{(i)})}{q_t(z^{(i)})} = L_{t-1}$
- Then $L_t \geq L_{t-1}$

Characteristics —

- Not convex
- May converge to local minimum
- Not analytically solvable
- Always converges, since $L \geq M$ and $M^{(t+1)} \geq M^{(t)}$ due to maximizing over M at each step

34 Neural Networks

Formulation

Formulation —

- Model architecture:
  - Input features (X) > weights (B) > weighted sums (S) > activation functions (φ) > hidden states (H) > weights > ... > hidden states > activation function > outputs
  - Can be seen as a composition of:
    - Encoders that construct disentangled and robust latent representation from input, which maximizes mutual information between representation and input, as according to infomax principle
    - Linear estimators
- Neuron (j) in layer $[k]$ given training instance $x^{(i)[0]}$ resp. instance from previous layer $\boldsymbol{h}^{(i)[k-1]}$ given by: $\boldsymbol{h}^{(j)[1]} = \varphi(\boldsymbol{x}^{(i)[0]} \cdot \boldsymbol{\beta}^{(j)[1]})$ resp. $\boldsymbol{h}^{(j)[k]} = \varphi(\boldsymbol{h}^{(i)[k-1]} \cdot \boldsymbol{\beta}^{(j)[k]})$
- Outputs for neurons $1,...,j$ in fixed layer (notation for layer omitted below) given by: $H = \varphi(\boldsymbol{XB}) = \varphi(S)$ where
  - $X \in \mathbb{R}^{n \times m+1}$ (incl. bias term)
  - $B \in \mathbb{R}^{m+1 \times j}$ with a weight vector for each neuron in each column (incl. bias term)
  - $S \in \mathbb{R}^{n \times j}$ with the weighted sum (prior to activation) for instance i in neuron j is on the i^{th} row and j^{th} column
  - The activation function differs by neuron

Activation functions —

- Introduce non-linearities
- Can differ by neuron
- Sigmoid:
  - $[0,1]$
  - $\varphi(z) = \sigma(z) = \frac{1}{1+e^{-z}} = \frac{e^z}{e^z+1}$
  - $\varphi'(z) = \frac{e^{-z}}{(1+e^{-z})^2}$
- Hyperbolic Tangent:
  - $[-1,1]$
  - $\varphi(z) = tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} = \frac{1-e^{-2z}}{1+e^{-2z}}$
  - $\varphi'(z) = 1 - tanh(z)^2$
- ReLu:
  - $\varphi(z) = max(0, z)$
  - $\varphi'(z) = 1$ if $z > 0$; 0 otherwise

Optimization

Parameters — Find parameters $\theta = B$

Objective function —

- Minimize standard objectives, e.g. MSE

Optimization —

- Perform *forward pass* with randomly initialized parameters, to calculate loss
- Perform *backpropagation*, to calculate gradient:
  - $\frac{\partial L}{\partial \theta} = [\frac{\partial L}{\partial \boldsymbol{B}^{[0]}}, ..., \frac{\partial L}{\partial \boldsymbol{B}^{[output]}}]$
  - $\frac{\partial L}{\partial \boldsymbol{B}^{[k]}} = \frac{\partial L}{\partial \boldsymbol{H}^{[l]}} \frac{\partial \boldsymbol{H}^{[l]}}{\partial \boldsymbol{B}^{[k]}} = C$
    - When $l > k+1$, i.e. when going several layers back: $\frac{\partial L}{\partial \boldsymbol{B}^{[k]}} = \frac{\partial L}{\partial \boldsymbol{H}^{[l]}} \frac{\partial \boldsymbol{H}^{[l]}}{\partial \boldsymbol{S}^{[l-1]}} \frac{\partial \boldsymbol{S}^{[l-1]}}{\partial \boldsymbol{H}^{[l-1]}} \frac{\partial \boldsymbol{H}^{[l-1]}}{\partial \boldsymbol{B}^{[k]}}$
    - When $l = k+1$, i.e. when going one layer back: $\frac{\partial L}{\partial \boldsymbol{B}^{[k]}} = \frac{\partial L}{\partial \boldsymbol{H}^{[k+1]}} \frac{\partial \boldsymbol{H}^{[k+1]}}{\partial \boldsymbol{S}^{[k]}} \frac{\partial \boldsymbol{S}^{[k]}}{\partial \boldsymbol{B}^{[k]}}$
- Perform gradient descent to find best weights

Backpropagation — TBA

Challenges — TBA

35 xxx

Big Picture

- In ANNs, we learn both f and θ, meaning that the objective function is not convex
- ANN objective: $argmin_\theta L(\theta) = argmin_\theta \sum_{(\boldsymbol{x},y)\in D} \ell(f(\boldsymbol{x};\theta), y)$
- Objective is optimized via gradient descent
- Gradient descent update rule: $\theta_{t+1} \leftarrow \theta_t - \eta \nabla_\theta L(\theta)|_{\theta=\theta_t}$
- Backpropagation uses the chain rule in combination with dynamic programming techniques to compute the gradient $\nabla_\theta L(\theta)$

Point of Departure

Composite function — Ordered series of equations (*primitives*), where each equation is a function only of the preceding equations

For example: $f(x,z) = x^2 + 3z$

$a = x^2, \quad b = 3z, \quad c = a + b, \quad f(x,z) = c$

Computation graph $\mathcal{G}$ — Graphical representation of a composite function

- Directed acyclic hypergraph (E, V) where V is the set of vertices (*nodes*) representing variables and E is the set of edges representing functions
  - Directed: One direction
  - Acyclic: No cycles
  - An edge can connect any number of vertices, not just two
- Edge from $v' \to v$: labeled with a differentiable function f_v
- Vertex v with outgoing edges: argument to f_v
- Vertex v with incoming edges: result of f_v
- If we have n input nodes and $|E| = M$ edges, we have $|V| = M + n$ total nodes
- Edge from $v' \to v$: labeled with a differentiable function f_v.
- Vertex v with outgoing edges: argument to f_v.
- Vertex v with incoming edges: result of f_v.

Bauer's Formula — Extension of partial derivative

- For two nodes i and j in $\mathcal{G}$, let the Bauer path $P(i,j)$ define the set of all directed paths starting at j and ending at i
- Let z_i and z_j represent these nodes as variables
- The partial derivative of z_i with respect to z_j is: $\frac{\partial z_i}{\partial z_j} = \sum_{p\in P(j,i)} \prod_{(k,l)\in p} \frac{\partial z_l}{\partial z_k}$ where we sum over all paths and take the product over all nodes on a given path

Proof:

- Base case: $i = j$, i.e., $z_i = z_j$: $\frac{\partial z_i}{\partial z_j} = 1$
- Inductive hypothesis: Bauer's formula holds for all $j \leq i$
- Inductive step:
  - Let $m < j$ (i.e., z_m comes before z_j in topological order, and $j \in \text{out}(m)$)
  - $\frac{\partial z_i}{\partial z_m} = \sum_{j\in\text{out}(m)} \frac{\partial z_i}{\partial z_j} \frac{\partial z_j}{\partial z_m}$ by the chain rule
  - $\frac{\partial z_i}{\partial z_m} = \sum_{j\in\text{out}(m)} (\sum_{p\in P(j,i)} \prod_{(k,l)\in p} \frac{\partial z_l}{\partial z_k}) \frac{\partial z_j}{\partial z_m}$ due to inductive hypothesis
  - $\frac{\partial z_i}{\partial z_m} = \sum_{j\in\text{out}(m)} (\sum_{p\in P(j,i)} \frac{\partial z_j}{\partial z_m} \prod_{(k,l)\in p} \frac{\partial z_l}{\partial z_k})$ due to distributivity
  - $\frac{\partial z_i}{\partial z_m} = \sum_{p\in P(m,i)} \prod_{(k,l)\in p} \frac{\partial z_l}{\partial z_k}$ by concatenating paths

- Challenge of naively calculating this: With $\sum_{p\in P(j,i)}$, we are summing over an exponential number of paths, leading to a runtime complexity of $O(|P(j,i)|) = O(2^{|E|})$

Forward Propagation

Initialize values of input nodes, and on that basis, calculate values of all descendant nodes

Algorithm $(f, \boldsymbol{x})$:

1. Initialize node values: $z_i = \begin{cases} x_i & \text{if } i \leq n \quad \text{(n input nodes initialized to value)} \\ 0 & \text{if } i > n \quad \text{(non-input nodes initialized to 0)} \end{cases}$
   $i = n+1, \ldots, M$ non-input nodes: $z_i = g_i(\langle z_{\text{pa}(i)} \rangle)$ where g_i denotes the primitive at edge i and $\langle z_{\text{pa}(i)} \rangle$ denotes the ordered set of parent nodes of z_i
2. Return $\{z_1, z_2, \ldots, z_M\}$

Properties:

- Runtime complexity: $O(|E|)$, i.e., linear in the number of edges
- Space complexity: $O(|V|)$, i.e., linear in the number of vertices

Backpropagation

After forward propagation, compute the gradient of f with respect to input nodes

Algorithm:

1. Perform forward propagation: $\boldsymbol{z} \leftarrow$ forward propagate$(f, \boldsymbol{x})$
2. Initialize: $\frac{\partial f}{\partial z_i} = \begin{cases} 1 & \text{if } i = M \\ & \text{(initialize gradient of } f \text{ with respect to output node as 1, since } f(\boldsymbol{x}) = z_M) \\ 0 & \text{otherwise} \end{cases}$
3. For $i = M-1, \ldots, 1$: $\frac{\partial f}{\partial z_i} = \sum_{j:i\in\text{Pa}(i)} \frac{\partial f}{\partial z_j} \frac{\partial z_j}{\partial z_i} = \sum_{j:i\in\text{Pa}(i)} \frac{\partial f}{\partial z_j} \frac{g_j(\langle z_{\text{pa}(j)} \rangle)}{\partial z_i}$
4. Return $\left[\frac{\partial f}{\partial z_1}, \frac{\partial f}{\partial z_2}, \ldots, \frac{\partial f}{\partial z_M}\right]$

Properties:

- This is a dynamic program
- Presents improvement over naive Bauer's formula: partial derivatives that appear on multiple paths are memoized
- Runtime complexity: $O(|E|)$, i.e., the same as forward propagation (which computes f)
- Space complexity: $O(|V|)$, i.e., the same as forward propagation (which computes f)
- *Cheap gradient principle*: Calculating the gradient has the same complexity as evaluating the function

Extension of backpropagation to k^th order derivatives:

- Backpropagation on a graph with M edges, for inputs $\boldsymbol{x} = (x_1, \ldots, x_n)^\top$, for the k^{th} order derivative has runtime $\mathcal{O}(Mn^{k-1})$

Proof:

- For second-order derivative: $\nabla_2 f(\boldsymbol{x})$(i.e. Hessian) $= \begin{bmatrix} \nabla(\boldsymbol{e}_1^\top \nabla f(\boldsymbol{x}) \text{(i.e. Jabobian)}) \\ \ldots \\ \nabla(\boldsymbol{e}_n^\top \nabla f(\boldsymbol{x})) \end{bmatrix}$ because $\boldsymbol{e}_k^\top \boldsymbol{A}$ returns k^{th} row of $\boldsymbol{A}$
- For third-order derivative, similar principle
- We first differentiate in M edges (∇_1), which means complexity is $1 \times M$
- We then differentiate by n variables in M edges (∇_2), which means complexity is $n \times M$
- We then differentiate these n variables by another n variables in M edges (∇_3), which means complexity is $n^2 \times M$
- For ...

Requirements for backpropagation:

- Weights need to be initialized to different values: If they are initialized to the same constant, each neuron produces the same output (since the constant weight is applied to the input), and then during backpropagation, all neurons receive the same gradient updates
  - Weights initialized to 0:
    - $\boldsymbol{h} = \varphi(\boldsymbol{xB}^{[1]}) = 0$ where $\boldsymbol{h}$ corresponds to z, φ is the primitive, and x corresponds to the input nodes
    - $\boldsymbol{y} = \boldsymbol{hB}^{[2]} = 0$ where y corresponds to z'
    - $\frac{\partial L}{\partial \boldsymbol{B}^{[2]}} = \frac{\partial L}{\partial \boldsymbol{y}} \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{B}^{[2]}} = \frac{\partial L}{\partial \boldsymbol{y}} \boldsymbol{h} = 0$, i.e. $\boldsymbol{B}^{[2]}$ is not updated
    - $\frac{\partial L}{\partial \boldsymbol{B}^{[1]}} = \frac{\partial L}{\partial \boldsymbol{y}} \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{h}} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{B}^{[1]}} = \frac{\partial L}{\partial \boldsymbol{y}} \boldsymbol{B}^{[2]} \frac{\partial \boldsymbol{h}}{\partial \boldsymbol{B}^{[1]}} = 0$, i.e. $\boldsymbol{B}^{[1]}$ is not updated
    - Thus, weights will always remain 0 and network will not learn
  - Weights initialized to same constant:
    - $\boldsymbol{B}^{[1]} = \boldsymbol{B}^{[2]}$
    - $\boldsymbol{h}_1 = \boldsymbol{h}_2$
    - $\frac{\partial L}{\partial \boldsymbol{B}^{[1]}} = \frac{\partial L}{\partial \boldsymbol{B}^{[2]}}$
    - Thus, weights will always receive same updates, will remain equal, and network will not learn
- At least one activation must be non-linear so that there is a non-zero gradient

Example: INSERT IMAGE 1

36 Bayesian Neural Networks

Setting

- In Bayesian setting, normalization constant is computationally intractable

Formulation

Since original setting is computationally intractable, we can turn to *variational inference*:

- Variational inference approximates true posterior $p(w|D)$ by simpler, parametrized distribution $q(w|\theta)$
- We assume $q(w|\theta) \sim \mathcal{N}(\mu, \sigma^2 I)$ with $\theta = (\mu, \sigma)$

Optimization

Parameters — Find parameters θ

Objective function —

- Minimize KL divergence: $\theta^* = argmin_\theta KL[q(w|\theta) \| p(w|D)] =$

$argmin_\theta \mathbb{E}_{w\sim q} log(q(w|\theta)) - \mathbb{E}_{w\sim q} log(p(\boldsymbol{D}|w)) - \mathbb{E}_{w\sim q} log(p(w))$

Proof:

- $argmin_\theta KL[q(w|\theta)|p(w|\boldsymbol{D})] =$
  $argmin_\theta \mathbb{E}_{w\sim q}[log(\frac{q(w|\theta)}{p(w|\boldsymbol{D})})] =$
  $argmin_\theta \mathbb{E}_{w\sim q}[log(q(w|\theta))] - \mathbb{E}_{w\sim q}[log(p(w|\boldsymbol{D}))] =$
  $argmin_\theta \mathbb{E}_{w\sim q}[log(q(w|\theta))] - \mathbb{E}_{w\sim q}[\frac{p(\boldsymbol{D}|w)\times p(w)}{p(\boldsymbol{D})}] =$
  $argmin_\theta \mathbb{E}_{w\sim q} log(q(w|\theta)) - \mathbb{E}_{w\sim q} log(p(\boldsymbol{D}|w)) - \mathbb{E}_{w\sim q} log(p(w)) + \mathbb{E}_{w\sim q} log(p(\boldsymbol{D})) =$
  $argmin_\theta \mathbb{E}_{w\sim q} log(q(w|\theta)) - \mathbb{E}_{w\sim q} log(p(\boldsymbol{D}|w)) - \mathbb{E}_{w\sim q} log(p(w)) +$ const.

Optimization —

- To calculate gradient, we can leverage the *reparametrization trick*
- $\frac{\partial}{\partial\theta}\mathbb{E}_{w\sim q}[log(q(w|\theta)) - log(p(\boldsymbol{D}|w)) - log(p(w))] = \frac{\partial}{\partial\theta}\mathbb{E}_{w\sim q}[F(w,\theta)]$ can be reparametrized to:
  - $\frac{\partial}{\partial\mu}\mathbb{E}_{\epsilon\sim\mathcal{N}(0,I)}[\frac{\partial}{\partial w}F(w,\theta) + \frac{\partial}{\partial\mu}F(w,\theta)]$
  - $\frac{\partial}{\partial\sigma}\mathbb{E}_{\epsilon\sim\mathcal{N}(0,I)}[\epsilon^\top \frac{\partial}{\partial w}F(w,\theta) + \frac{\partial}{\partial\sigma}F(w,\theta)]$
- To optimize this, we can use gradient descent with the following algorithm:
  1. Initialize μ and σ
  2. For $t = 1, 2, ...$
     (a) Sample $\epsilon \sim \mathcal{N}(0, \boldsymbol{I})$
     (b) Compute $F(w,\theta)$
     (c) $\mu_{t+1} \leftarrow \mu_t - \eta_t[\frac{\partial}{\partial w}F(w,\theta) + \frac{\partial}{\partial\mu}F(w,\theta)]|_{\mu=\mu_t}$
     (d) $\sigma_{t+1} \leftarrow \sigma_t - \eta_t[\epsilon^\top \frac{\partial}{\partial w}F(w,\theta) + \frac{\partial}{\partial\sigma}F(w,\theta)]|_{\sigma=\sigma_t}$

37 Convolutional Neural Networks (CNNs)

Formulation

Formulation —

- Model architecture:
  - Input: Composed of channels (e.g. R with 3 channels)
  - Convolutional layer: Composed of feature maps
  - Channel: Sublayer in input and output, composed of pixels
  - Feature map: Sublayer in convolutional layer, composed of neurons, each neuron is generated by applying filter to all receptive fields across all sublayers in lower layer, weights and biases shared across all neurons in feature map
  - Receptive field: Group of neurons in lower layer, that single neuron in higher layer is connected to
  - Filter resp. convolutional kernel: Weights applied to all receptive fields across all sublayers in lower layer
  - Zero padding: Padding applied to retain same dimensions in each layer
  - Stride: By how many neurons receptive field shifts, if stride > 1, spatial dimensions in subsequent layer decrease (convolution), if stride < 1, spatial dimensions increase (deconvolution)
- Output of neuron in layer n: $z_{i,j,k} = b_k + \sum_{f_n}\sum_{f_w}\sum_{f'_n} x_{i',j',k'} \cdot w_{u,v,k',k}$, i.e. sum of element-wise matrix product over all receptive fields and all feature maps, where
  - $z_{i,j,k}$ is the output of neuron in row i and column j on feature map k in layer n
  - f_n and f_w are dimensions of the receptive field in layer $n-1$
  - f'_n is the number of feature maps in layer $n-1$
  - $x_{i',j',k'}$ is the output of neuron in row i' and column j' on feature map k' in layer $n-1$
  - $w_{u,v,k',k}$ is the connection weight between any neuron on feature map k in layer n and its input at u, v on feature map k'
- Output of neurons in layer n: $z_k = b_k + \sum_{f_n}\sum_{f_w}\sum_{f'_n} W_{k',k}X_{k'}$

Optimization

Parameters — Find parameters $\theta = W$

Objective function —

- Minimize standard objectives, e.g. MSE

Optimization —

- Perform *forward pass* with randomly initialized parameters, to calculate loss
- Perform *backpropagation*, to calculate gradient
- Perform gradient descent to find best weights

38 Recurrent Neural Networks (RNN)

Description

Description —

- Can deal with sequential data and the persistence of information over time
- Can be seq-to-seq, seq-to-vec, or vec-to-seq
- Can be unidirectional or bidirectional
- Challenge: Cannot preserve long-term dependencies well. Solution: LSTM

Formulation

Formulation —

- Model architecture:
  - Input layer
  - Hidden layer resp. memory cell: Cell state h_t, cell output y_t
  - Output layer
- Output of neuron in layer n: $z_{i,j,k} = b_k + \sum_{f_n}\sum_{f_w}\sum_{f'_n} x_{i',j',k'} \cdot w_{u,v,k',k}$, i.e. sum of element-wise matrix product over all receptive fields and all feature maps, where
  - $z_{i,j,k}$ is the output of neuron in row i and column j on feature map k in layer n
  - f_n and f_w are dimensions of the receptive field in layer $n-1$
  - f'_n is the number of feature maps in layer $n-1$
  - $x_{i',j',k'}$ is the output of neuron in row i' and column j' on feature map k' in layer $n-1$
  - $w_{u,v,k',k}$ is the connection weight between any neuron on feature map k in layer n and its input at u, v on feature map k'
- Output of neurons in layer n: $z_k = b_k + \sum_{f_n}\sum_{f_w}\sum_{f'_n} W_{k',k}X_{k'}$

Optimization

Parameters — Find parameters $\theta = W_x, W_y, \boldsymbol{b}, \boldsymbol{X}_t$, which are shared across time steps

Objective function —

- Maximize log likelihood

Optimization —

- Perform *forward pass* with randomly initialized parameters, to calculate loss
- Perform *backpropagation through time*, to calculate gradient:
  - $\boldsymbol{y} = \varphi(\boldsymbol{X}_h \boldsymbol{W}_h)$
  - $\nabla_{\boldsymbol{W}_h} L \propto \sum_{k=1}^{t}(\prod_{i=k+1}^{t}\frac{\partial h_i}{\partial h_{i-1}})\frac{\partial h_k}{\partial \boldsymbol{W}_k}$
  - $\frac{\partial h_{i+k}}{\partial h_i} = \prod_{j=0}^{k-1}\frac{\partial h_{i+k-j}}{\partial h_{i+k-j-1}}$
- Perform gradient descent to find best weights

39 Long-Short-Term Memory (LSTM)

Description

Description —

- Can deal with sequential data and the persistence of information over time
- Can be seq-to-seq, seq-to-vec, or vec-to-seq
- Can be unidirectional or bidirectional

Formulation

Formulation —

- Model architecture:
  - Input layer
  - Hidden layer resp. memory cell:
    - Short-term state h_t, long-term state c_t
    - Cell output y_t
  - Output layer
- Forget gate:
  - Sigmoid layer
  - Serves to decide which information to keep from previous cell state, 1 : retain, 0: forget
  - $f_t = \sigma(w_f \cdot [h_{t-1}, x_t] + b_f$ where $w_f = \begin{bmatrix} w_{xf} & = \text{connection weight for } x_t \\ w_{hf} & = \text{connection weight for } h_{t-1} \end{bmatrix}$
- Input gate:
  - Two stages:
    - Sigmoid layer, determines if values are updated, 1 : update values, 0: do not update values
    - Tanh layer, creates vector of new candidate values that could be added to the cell state
  - Serves to decide which information will be stored in the cell state
  - $i_t = \sigma(w_i \cdot [h_{t-1}, x_t] + b_i$ where $w_i = \begin{bmatrix} w_{xi} & = \text{connection weight for } x_t \\ w_{hi} & = \text{connection weight for } h_{t-1} \end{bmatrix}$
  - $\tilde{c}_t = tanh(w_{\tilde{c}} \cdot [h_{t-1}, x_t] + b_c$ where $w_{\tilde{c}} = \begin{bmatrix} w_{x\tilde{c}} & = \text{connection weight for } x_t \\ w_{h\tilde{c}} & = \text{connection weight for } h_{t-1} \end{bmatrix}$
- Cell state:
  - Two stages:
    - Calculate what is left from previous cell state after forget care
    - Calculate what we need to add to cell state after input gate
  - Serves to update old cell state into new cell state
  - $c_t = f_t \otimes c_{t-1} + i_t \otimes \tilde{c}_t$ where $\otimes$ is element-wise multiplication
- Output gate:
  - Three stages:
    - Sigmoid layer, determines which part of cell state to output
    - Tanh layer, activates cell state values
    - Multiply activated cell state and output gate values to get h_t
  - Serves to output h_t for next time step, which is a filtered version of cell state c_t
  - $o_t = \sigma(w_o \cdot [h_{t-1}, x_t] + b_o$ where $w_o = \begin{bmatrix} w_{xo} & = \text{connection weight for } x_t \\ w_{ho} & = \text{connection weight for } h_{t-1} \end{bmatrix}$
  - $h_t = o_t \otimes tanh(c_t)$

Optimization

Parameters — Find parameters θ, which are shared across time steps

Objective function —

- Maximize log likelihood

Optimization —

- Perform *forward pass* with randomly initialized parameters, to calculate loss
- Perform *backpropagation through time*, to calculate gradient
- Perform gradient descent to find best weights

40 Attention

Description

- Attention helps specify which inputs we need to pay attention to when producing a given output
- Can be used:
  - As cross-attention: Between encoder and decoder
  - As self-attention: Within a single hidden layer resp. within the encoder or decoder
- Usually applied after embedding and before applying an activation function

Naive method

- Let E be the embedding matrix $\mathbb{R}^{s\times d}$ where s = number of tokens, d = embedding dimensionality
- Steps:
  - Re-weight embeddings: $\tilde{E} = Ew$
  - Compute similarity matrix: $S = \sigma(\tilde{E}\tilde{E}^\top)$ where σ is softmax, normalizing the rows of S to sum to 1, $\tilde{E}\tilde{E}^\top$ is similarity between two tokens in $\tilde{E}$
  - Compute attention-weighted embedding matrix: $A = S\tilde{E}$
- In this context, E and w are trainable parameters
- Challenge: Similarity matrix is symmetric, given that dot product $\tilde{E}\tilde{E}^\top$ is symmetric, i.e. attention paid by token i to token j (S_{ij}) is same as attention paid by token j to token i (S_{ji})

Adjusted method

- Steps:
  - Generate three sets of re-weighted embeddings:
    - $Q = EW^q$ resp. $q_i = e_i W^q$
      - E $(n \times h)$
      - W_q $(h \times d_k)$
      - Q $(m \times d_k)$
      - q_i is row vector $(1 \times d_k)$
      - e_i is row vector $(1 \times h)$
    - $K = EW^k$ resp. $k_i = e_i W^k$
      - E $(n \times h)$
      - W_k $(h \times d_k)$
      - K $(n \times d_k)$
      - k_i is row vector $(1 \times d_k)$
      - e_i is row vector $(1 \times h)$
    - $V = EW^v$ resp. $v_i = e_i W^v$ where
      - E $(n \times h)$
      - W_v $(h \times d_v)$
      - V $(n \times d_v)$
      - v_i is row vector $(1 \times d_v)$
      - e_i is row vector $(1 \times h)$
    - (Note: if Q, K, V contain multiple heads, they are expanded in this step)
  - Compute similarity matrix: $A = \sigma(\frac{QK^\top}{\sqrt{d_k}})$ in $(m \times n)$ resp. $\alpha_{ti} = \frac{exp(q_t \cdot k_i)}{\sum_{i'} exp(q_t \cdot k_{i'})}$
  - Compute attention-weighted embedding matrix: $Z = AV$ in $(m \times d_v)$ resp. $z_t = \sum_i \alpha_{ti} v_i$
  - (Note: if A contains multiple heads, they are flattened in this step by multiplying with d_v)
- In cross-attention:
  - Attention for decoder-encoder alignment
  - Q is generated with decoder input during traning
  - V, K are generated with encoder outputs during training
- In self-attention:
  - Attention for encoder resp. decoder inputs
  - Q, V, K are generated with input during training
  - In masked self-attention:
    - We first calculate $P = QK^\top$ where masked elements (e.g. states with time $\geq m$ in decoder) are set to $-\infty$
    - $S = \sigma(\frac{P}{\sqrt{d_k}})$
- In multi-head attention:
  - Creates multiple sets of Q, K, V and calculates attention correspondingly
  - Concatenates generated matrices Z

41 Encoder Decoder RNNs

Description

Task — Generate embeddings, perform sequence-to-sequence tasks

Formulation

Formulation —

- Model architecture:

- Inputs fed into encoder in reverse order
- Encoder:
  - Sequence-to-vector
  - Hidden states $h_n^{l(e)} = f(W_l^{(e)} h_{n-1}^{l(e)} + W_{l-1}^{(e)} h_n^{l-1(e)})$ where f is activation function, l is layer, n is time step in input sequence
- Outputs from encoder to decoder are weighted by attention weights:
  - Context vector $z_m = \sum_i \alpha_{mi} h_i^{(e)}$ where
    - $h_i^{(e)}$ is the encoder hidden state (= V)
    - α_{mi} is attention weight at decoder time step m for encoder hidden state $h_i^{(e)}$
    - $\alpha_{mi} = \text{softmax}(h_{m-1}^{(d)} \cdot h_i^{(e)})$ where $h_{m-1}^{(d)}$ is previous decoder hidden state (= Q) and $h_i^{(e)}$ are the final encoder hidden states at each time step (= K)
- Alongside context vectors, target sequence inputs are fed into decoder with one time step lag during training
- Decoder:
  - Vector-to-sequence
  - Hidden states $h_m^{(d)} = f(W_2^{(d)} h_{m-1}^{(d)} + W_1^{(d)} w'_{m-1})$ where f is activation function, m is time step in target sequence, w'_{m-1} is word at index $m-1$ in target sequence (assuming one layer), $h_0^{(d)} = h_N^{(e)}$, i.e. last encoder output is first decoder input
- Challenge: Sequential, cannot be parallelized. Solution: Transformers

Variants

Variants —
- ELMO: Bi-directional LSTM

42 Encoder Decoder Transformers

Description

Task — Generate embeddings, perform sequence-to-sequence tasks

Formulation

Formulation —
- Model architecture:
  - Inputs fed into encoder in reverse order
  - Encoder:
    - Sequence-to-vector
    - Takes in input sequence token embeddings (semantic vector) and positional embeddings (sinusoidal pointer vector for word position, given that model is not sequential) and adds them
    - Multi-head self-attention, applied to all tokens jointly, where Q, K, V are tokens in input sequence
    - Addition and normalization: Skip connections (from token + positional embeddings) added back and normalized
    - Feed-forward network, parallel for each token
    - Addition and normalization: Skip connections (from first addition and normalization) added back and normalized
    - Generates hidden states $h_n^{(e)} = f(W_1^{(e)} h_{n-1}^{(e)} + W_2 w_n)$ where f is activation function, w_n is token at index n in input sequence
  - Outputs from encoder to decoder are weighted by attention weights:
    - Context vector $z_m = \sum_{n=1}^{N} \alpha_{m,n} h_n^{(e)}$ where
      - $h_n^{(e)}$ is the encoder hidden state (= V)
      - $\alpha_{m,n}$ is attention weight at decoder time step m for encoder hidden state $h_n^{(e)}$
      - $\alpha_{m,n} = \phi(h_{m-1}^{(d)}, [h_1^{(e)}, ..., h_N^{(e)}]$ where $h_{m-1}^{(d)}$ is previous decoder hidden state (= Q) and $[h_1^{(e)}), ..., h_N^{(e)})]$ are the final encoder hidden states at each time step (= K)
  - Alongside context vectors, target sequence inputs are fed into decoder with one time step lag during training
  - Decoder:
    - Vector-to-sequence
    - Takes in target sequence token embeddings (semantic vector) and positional embeddings (sinusoidal pointer vector for word position, given that model is not sequential) and adds them
    - Masked multi-head self-attention, applied to all tokens jointly, where Q, K, V are tokens in target sequence
    - Addition and normalization: Skip connections (from token + positional embeddings) added back and normalized
    - Cross-attention to incorporate encoder outputs
    - Addition and normalization: Skip connections (from first addition and normalization) added back and normalized
    - Feed-forward network, parallel for each token
    - Addition and normalization: Skip connections (from second addition and normalization) added back and normalized
    - Hidden states $h_m^{(d)} = f(W_3^{(d)} h_{m-1}^{(d)} + W_2^{(d)} w'_{m-1} + W_1^{(d)} z_m)$ where f is activation function, m is time step in target sequence, w'_{m-1} is token at index $m-1$ in target sequence, $h_0^{(d)} = h_N^{(e)}$, i.e. last encoder output is first decoder input
  - Linear layer applied to $h_M^{(d)}$
  - Softmax layer applied to select token with highest probability: Since it is intractable to search for best sequence overall (explore), we turn to deterministic or stochastic variants (exploit):
    - Greedy decoding: Select highest-probability token at each step
    - Beam search: Keep n-highest-probability tokens in memory (beam size) and return k-most-likely sequences (top beams)
    - Nucleus sampling: Sample tokens from items that cover $p\%$ of PMF
- Advantage:
  - Relies on attention to obtain a fixed-size representation of a sequence (in contrast to RNN)
  - Allows to learn longer-range dependencies than RNNs
  - Allows for parallelization, whereas RNNs must run sequentially

Variants

Variants —
- GPT: Uni-directional, decoder-only transformer, predicts next word in sequence
- BERT: Bi-directional, encoder-only transformer, predicts masked word from context

43 Variational Autoencoders (VAE)

Description

Task — Autoencoders as representation learning (learn to copy inputs to outputs)

Description — Variational autoencoders as probabilistic autoencoders

Formulation

Formulation —
- Model architecture:
  - Encoder / recognition network: Converts input to latent representation
  - Decoder / generative network: Converts latent representation to output
- Assumes data x is generated by latent variable h: $p(x,h) = p_\theta(x|h) \times p_{\theta'}(h)$
- Prior $p_{\theta'}(h) \sim \mathcal{N}(0, \sigma^2 I)$
- Encoder produces a probability distribution $q_\Phi(h|x)$ which approximates the true posterior $p_{\theta,\theta'}(h|x)$ since the true posterior is intractable:
  - Produces $\mu_{h|x}$ and $\Sigma_{h|x}$ according to which h is approximately distributed in latent space
- Decoder reconstructs input $p_\theta(x|h)$ by sampling from latent space $h \sim q_\Phi(h|x)$:
  - Produces $\mu_{x|h}$ and $\Sigma_{x|h}$ according to which $\hat{x}$ is approximately distributed in output space

Optimization

Parameters — Find parameters θ' (prior), θ (likelihood), Φ (approximate posterior)

Objective function —
- Maximize log likelihood $\sum_{i=1}^{n} \log p_{\theta,\theta'}(x^{(i)})$
- Challenge: This does not involve the encoder
- Solution:

$$argmax_{\theta,\theta',\Phi} \sum_{i=1}^{n} \mathbb{E}_{q(h|x)} \log[\frac{p_{\theta,\theta'}(x^{(i)},h)}{p_{\theta,\theta'}(h|x^{(i)})} \times \frac{q_\Phi(h|x^{(i)})}{q_\Phi(h|x^{(i)})}] = \sum_{i=1}^{n} \mathbb{E} \log[\frac{p_{\theta,\theta'}(x^{(i)},h)}{q_\Phi(h|x^{(i)})}] + \mathbb{E} \log[\frac{q_\Phi(h|x^{(i)})}{p_{\theta,\theta'}(h|x^{(i)})}] = \sum_{i=1}^{n} \text{ELBO}_{\theta,\theta',\Phi}(x^{(i)}) + \text{KL divergence}(q_\Phi(\cdot|x^{(i)}) | p_{\theta,\theta'}(\cdot|x^{(i)}))$$

- ELBO provides lower bound of log likelihood: $\log p_{\theta,\theta'}(x^{(i)}) \geq$

$$\text{ELBO}_{\theta,\theta',\Phi}(x^{(i)}) = \mathbb{E} \log[\frac{p_{\theta,\theta'}(x^{(i)},h)}{q_\Phi(h|x^{(i)})}] = \mathbb{E} \log[\frac{p_{\theta'}(h) \times p_\theta(x^{(i)}|h)}{q_\Phi(h|x^{(i)})}] = \mathbb{E} \log[p_\theta(x^{(i)}|h)] + \mathbb{E} \log[\frac{p_{\theta'}(h)}{q_\Phi(h|x^{(i)})}] =$$

mutual information resp. cross-entropy from decoder − KL divergence $(q_\Phi(\cdot|x^{(i)}) | p_{\theta'}(\cdot))$ from encoder

- Mutual information enforces infomax principle of ANNs
- KL divergence acts as regularization

44 Generative Adversarial Nets (GAN)

Description

Task — Generate new samples

Formulation

Formulation —
- Model architecture:
  - Generator: Encoder decoder network, aims to produce samples that pass as real in the discriminator network
  - Discriminator: Fake detection network, aims to tell fake from real samples

Optimization

Steps —
- Encoder decoder network is trained in autoencoder mode to reproduce its input
- Decoder is fed with Gaussian noise input vectors $z^{(i)}$ and produces corresponding samples $\tilde{y}^{(i)}$
- Fake detection network produces $\hat{\xi} = 0$ for real samples $y^{(i)}$ and $\hat{\xi} = 1$ for $\tilde{y}^{(i)}$

Objective function —
- Encoder is trained to produce $\hat{z}^{(i)} = z^{(i)}$
- Decoder is trained to produce $\hat{\xi} = 0$
- Fake detection network is trained to correctly produce ξ
- Trained in interleaving manner

45 Diffusion Models

Description

Task — Generate new samples

Formulation

Formulation —
- Model architecture:
  - VAE for compression: Compress original data into latent space
  - Forward diffusion: Gradually add Gaussian noise to latent representation
  - Reverse diffusion: Gradually remove noise from latent representation, potentially guided by condition (e.g. text input), until latent representation is restored
  - Decoder: Decodes latent representation back to original data
- Generally implemented as a U-Net

Components

Forward diffusion —
- Real data: $x_0 \sim p(x)$
- Gaussian noise: $x_T \sim \mathcal{N}(0, \sigma^2 I)$
- $x_0 \to x_1 \to ... \to x_T$
- Stochastic differential equation: $dx = f(x,t)dt + g(t)dW$
  - x : Variable being diffused
  - $f(x,t)$: Deterministic *drift term*: Represents the mean rate of change of x at time t
  - $g(t)$: *Diffusion coefficient*: Function that scales the magnitude of the noise
  - dW : Random *Wiener process resp. Brownian motion*: Represents random fluctuations
  - Function shows how x changes over time due to deterministic and random influences
- Two transitions:
  - $q(x_t|x_{t-1}) \sim \mathcal{N}(x_t | \sqrt{1-\beta} x_{t-1}, \beta_t I)$
    - β_t : Added noise
  - $q(x_t|x_0) \sim \mathcal{N}(x_t | \sqrt{\overline{\alpha}_t} x_0, (1-\overline{\alpha}_t) I)$
    - $\alpha_t = 1 - \beta_t$: Retained signal
    - $\overline{\alpha}_t = \prod_{s \leq t} \alpha_t$: Cumulative retained signal until t
    - $1 - \overline{\alpha}_t$: Cumulative added noise until t

Reverse diffusion —
- $x_T \to x_{T-1} \to ... \to x_0$
- Stochastic differential equation: $dx = [f(x,t) - g(t)^2 \nabla_x \log p(x,t)]dt + g(t)dW$
  - x : Variable being diffused
  - $f(x,t)$: Deterministic *drift term*: Represents the mean rate of change of x at time t
  - $\nabla_x \log p(x,t)$: Score function: Captures gradient of log probability of system state at time t. Guides the system to reverse the noise by pulling the state x to higher-probability regions.
  - $g(t)^2$: Function that scales the impact of the score function
  - $g(t)$: Function that scales the magnitude of the noise
  - dW : Random *Wiener process resp. Brownian motion*: Represents random fluctuations
  - Function shows how x changes over time due to deterministic and random influences
- To approximate the score function resp. predict the noise, we train a model $f_\theta(x,t) \approx \nabla_x \log p(x,t)$:
  - $f_\theta(x,t) : \mathcal{X} \times [0,1] \to \mathcal{X}$, where $t \in [0,1]$ and $x \sim p(x)$
  - Proof that $f_\theta(x,t)$ approximates the score function and predicts the noise:
    - Gradient of log probability of Gaussian is: $\frac{-(x_t - \mu)}{\sigma^2}$
    - Therefore and based on $q(x_t|x_0)$ from forward diffusion, score is given by:

$\nabla_x \log p(x,t) = \frac{-(x_t - \sqrt{\bar{\alpha}_t} x_0)}{(1-\bar{\alpha}_t)}$
  - x_0 is unknown and must be estimated as $\hat{x}_0$
  - Then, we have $\nabla_x \log p(x,t) = \frac{-(x_t - \sqrt{\bar{\alpha}_t} \hat{x}_0)}{(1-\bar{\alpha}_t)}$, where $f_\theta(x,t) = (x_t - \sqrt{\bar{\alpha}_t}\hat{x}_0)$, which is the predicted noise
  - From this we see, that $f_\theta(x,t)$ is equal to $\nabla_x \log p(x,t)$ up to a scaling factor $-\frac{1}{(1-\bar{\alpha}_t)}$
- $f_\theta(x,t)$ is trained to minimize the difference between the added and predicted noise: $min_\theta \mathbb{E}_{x,\epsilon,t} \|\epsilon - f_\theta(x + \sigma_t \epsilon, t)\|^2$
- Reconstruction formula: $x_{t-1} = \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t,t)) + \sigma_t z$
- $\frac{1}{\sqrt{\alpha_t}}$: Scaling factor
- $\frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t,t)$: Predicted noise
- In formula, predicted noise is subtracted from x_t, after which a small amount of random noise $\sigma_t z$ is re-introduced to maintain stochasticity

Compression —
- Motivation: Images often have high dimensionality, which can be easily compressed, since there are many redundant dimensions that do not contain unique information
- VAE first encodes x into latent representation z and then decodes z into original representation $\hat{x}$

Conditional denoising —
- Extension of diffusion model to include additional condition y, e.g. a text prompt, and guiding the denoising process such that the generated sample $\hat{x}$ aligns with y
- VAE first encodes x into latent representation z and then decodes z into original representation $\hat{x}$
- To approximate the score function resp. predict the noise, we train a model $f_\theta(x,y,t) \approx \nabla_x \log p(x,t)$:
  - $f_\theta(x,y,t) : \mathcal{X} \times \mathcal{Y} \times [0,1] \to \mathcal{X}$, where $t \in [0,1]$ and $x \sim p(x)$
  - $f_\theta(x,y,t)$ is trained to minimize the difference between the added and predicted noise: $min_\theta \mathbb{E}_{x,y,\epsilon,t} \|\epsilon - f_\theta(x + \sigma_t \epsilon, y, t)\|^2$
- Condition y is injected via:
  - Concatenating y with latent feature maps
  - Cross-attention:
    - Dynamically weighs importance of different input features by paying attention to specific regions of the feature map based on guidance
    - Queries from latent feature map z
    - Keys and values from condition y
    - Dimensionality analysis:
      - Latent feature map z: $B \times C \times h \times w$
      - Text embedding y: $B \times M \times D_E$
      - W_q: $F \times C \times D_K \times f_h \times f_w$
      - Q: $B \times F \times h \times w \times D_K$
      - W_k: $F \times D_E \times D_K$
      - K: $B \times F \times M \times D_K$
      - W_v: $F \times D_E \times D_V$
      - V: $B \times F \times M \times D_V$
      - P and S: $B \times F \times h \times w \times M$
      - SV: $B \times F \times h \times w \times D_V$
    - When y is text embedding, but z is image embedding, we need *CLIP resp. contrastive learning* to match text and image:
      - Let true caption embeddings be $[t_1, ..., t_N]$ and corresponding image embeddings be $[i_1, ..., i_N]$
      - Matrix representation given by $\begin{bmatrix} t_1 \cdot i_1 & ... & t_N \cdot i_1 \\ ... & ... & ... \\ t_1 \cdot i_N & ... & t_N \cdot i_N \end{bmatrix}$
      - Aim is to maximize similarity between image and its true caption on diagonal, but minimize other pairs
  - Spatial transformer: Dynamically transforms spatial properties of an input (e.g. scale, rotation, location) to focus on most relevant parts

U-Net —
- Architecture:
  - CNN (encoder): Downsamples image to create compressed, low-dimensional representation, uses pooling and ReLu
  - Inverted CNN (decoder): Upsamples compressed, low-dimensional representation to generate image, uses upsampling and softmax
- Key operations:
  - Skip connections going directly from encoder to decoder
  - Convolution operation: Apply spatial filters to extract spatial features (e.g. edges, textures, patterns)
  - Down- and upsampling: Reduce spatial dimensions of input (e.g. via max-pooling, strided convolutions) resp. reconstruct spatial resolution (e.g. via transposed convolution, interpolation), low-level details (e.g. edges) captured at shallow layers, high-level patterns (e.g. objects) captured at deeper layers

46 Natural Language Processing (NLP) Basics

Formulation

Training data —
- $y = X\beta$
- For example: $\begin{bmatrix} \text{score for doc 1} \\ ... \\ \text{score for doc n} \end{bmatrix} = \begin{bmatrix} \text{feature 1 for doc 1} ... \text{feature m for doc 1} \\ ... \\ \text{feature 1 for doc n} ... \text{feature m for doc n} \end{bmatrix} \times \begin{bmatrix} \text{coefficient for feature 1} \\ ... \\ \text{coefficient for feature m} \end{bmatrix}$
- Where attribute j for doc i = $\boldsymbol{w}^{(i)} \cdot \boldsymbol{a}^{(i)} = [\text{weight of word 1 in doc i} ... \text{weight of word n in doc i}] \times \begin{bmatrix} \text{attribute j of vocab word 1} \\ ... \\ \text{attribute j of vocab word n} \end{bmatrix}$

Terminology —
- Corpus
- Document:Contained in corpus, constitutes one of n instances for model
- Tokens: Document split into preprocessed words, which are the tokens
- Vocabulary: Contains unique tokens in corpus
- Token-level features:
  - One-hot encoding: Vector of length of vocabulary, with 1 at index of token and 0 everywhere else
  - Embeddings: Measure semantic and syntactic word similarities in higher dimensional space, e.g. Word2Vec, GloVE
- Document-level features: Generated by pooling token-level features
- Pooling methods:
  - Sum pooling: Sum up token-level features $e(\boldsymbol{d}) = \sum_{t \in \boldsymbol{d}} e(\boldsymbol{t})$
  - Mean pooling: Average token-level features $e(\boldsymbol{d}) = \frac{1}{|\boldsymbol{d}|} \sum_{t \in \boldsymbol{d}} e(\boldsymbol{t})$ with token weights:
    - One-hot encoding: e.g., $[0,1,0]$
    - Bag-of-words: e.g., $[1,2,0]$
    - TF-IDF: Bag-of-words counts given by: $\frac{\text{vocab word frequency in document}}{\text{vocab word frequency in corpus}}$
  - Max pooling: Maximum of token-level features $e(\boldsymbol{d}) = \max_{t \in \boldsymbol{d}} e(\boldsymbol{t})$

47 Natural Language Processing (NLP) Basics

Description

Task —
- Predict context word given center word
- Generate word embeddings

Formulation

- Let us consider vocabulary $\mathcal{V}$, for which we wish to create embeddings $e(w)$ by considering the T preceding and following words of a center word
- Start by processing each document into a set $\mathcal{D}$ of $\mathcal{O}(T \cdot |\mathcal{V}|)$ pairs of center words w and context words w': $\{(w_i, w_t)\}$ where w_i is center word, w_t is context word, $c \in \{-T, -T+1, ..., -1, 1, ..., T-1, T\}$

Optimization

Parameters — Find parameters xxx

Objective function —
- Given by bi-linear softmax function
- Bilinear model is linear, if other variables are held constant, in this case e_{wrd} or e_{ctx}
- Likelihood: $\prod_{(w_i,w_t) \in \mathcal{D}} p(w_t \mid w_i)$ where $p(w_t \mid w_i) = \frac{\exp(e_{\text{wrd}}(w_i) \cdot e_{\text{ctx}}(w_t))}{Z(w_i)}$ where
  - $Z(w_i) = \sum_{w' \in \mathcal{V}} \exp(e_{\text{wrd}}(w_i) \cdot e_{\text{ctx}}(w'))$
  - $e_{\text{wrd}}(w_i) \cdot e_{\text{ctx}}(w_t) = (\boldsymbol{w} \cdot \boldsymbol{e}_{oh}(w_i)) \cdot (\boldsymbol{w} \cdot \boldsymbol{e}_{oh}(w_t))$ where $\boldsymbol{e}_{oh}(w)$ is the one hot encoding
- Log-likelihood: $\sum_{(w_i,w_t) \in \mathcal{D}} \log(p(w_t \mid w_i)) = e_{\text{wrd}}(w_i) \cdot e_{\text{ctx}}(w_t) - \log(Z(w_i)) =$
- Challenge: $Z(w_i)$ can take a long time to compute for large $|\mathcal{V}|$, since there are $2|\mathcal{V}|$ parameters
- Solution: Use *negative sampling*:
  - For each pair (w_i, w_t), we randomly sample with replacement a set $\mathcal{C}^-$ from $\mathcal{V}$
  - We compute sigmoid, instead of softmax
  - Then we have: $\sum_{(w_i,w_t,\mathcal{C}^-)} (\log(p(w_t \mid w_i)) + \sum_{w^- \in \mathcal{C}^-} \log(1 - p(w^- \mid w_i)))$
  - $= \sum_{(w_i,w_t,\mathcal{C}^-)} \frac{1}{1+\exp(-e_{\text{wrd}}(w_i) \cdot e_{\text{ctx}}(w_t))} + \sum_{w^- \in \mathcal{C}^-} \log(1 - p(w^- \mid w_i))$
  - Here:
    - $\log(p(w_t \mid w_i))$ is computed with sigmoid, instead of softmax
    - $\sum_{w^- \in \mathcal{C}^-} \log(1 - p(w^- \mid w_i))$ replaces $-\log(Z(w_i))$

Optimization —
- $argmax_{w,w'}$ log-likelihood
- Gradient descent
- We usually use center word embeddings w and throw away context word embeddings w'
- Evaluation via:
  - Cosine similarity between words
  - Word analogies (e.g., man → woman has same distance and direction as king → queen)

48 Language Models

Background

- Alphabet Σ: Finite set of symbols; in NLP: vocabulary of tokens
- String y over an alphabet: Finite sequence of alphabet symbols
- Set of all possible strings (incl. empty string ε) given by Kleene closure, i.e. the set of all possible strings (incl. the empty string ε) that can be formed by concatenating 0 or more elements from Σ
- BOS: Beginning of sequence token
- EOS: End of sequence token

Description

Task —
- *Structured prediction task*: Predict over exponentially (or even infinitely) large set of classes
- Assign a probability to y, i.e., fit a probability distribution over all Σ^*

Formulation

Globally normalized language model:
- Compute the probability of sentence y and normalize over all $\boldsymbol{y} \in \Sigma^*$: $p(\boldsymbol{y}) = \frac{\prod_{t=1}^{|\boldsymbol{y}|} \theta_{y \leq t}}{Z}$ where $\theta_{y \leq t}$ is the weight for transition from $y_{t-1} \to y_t$, conditioned on having taken the path $y_1, ..., y_{t-1}$ and Z is the normalization constant given by $Z = \sum_{\boldsymbol{y}' \in \Sigma^*} \prod_{t=1}^{|\boldsymbol{y}'|} \theta_{y' \leq t}$
- Challenge: Since Σ^* is infinite, Z is infinite
- Solution: Locally normalized language model

Locally normalized language model:
- Probability of a word y_t in a sentence $\boldsymbol{y}$ is conditioned only on the preceding context
- Compute the probability of sentence $\boldsymbol{y}$ and normalize over all possible words in the vocabulary: $p(\boldsymbol{y}) = p(y_1 \mid \text{BOS}) \times p(y_2 \mid \text{BOS } y_1) \times ... \times p(y_N \mid \boldsymbol{y}_{<N}) \times p(\text{EOS} \mid \boldsymbol{y})$
- Can be visualized as a prefix tree:
  - INSERT IMAGE 1

Tight models:
- If the language model is locally normalized and sums to 1, the language model is *tight*
- To enforce tightness: $p(\text{EOS} \mid \cdots) > \delta > 0$
  - Assume we want the probability sentences of less than length n to be $(1-\delta)$
  - Then, the probability of a sentence of length n is δ
  - The probability over all sentences is then given by: $\sum_{n=1}^{\infty} (1-\delta)^{n-1} \delta = \frac{\delta}{1-(1-\delta)} = 1$ because it forms a geometric series

49 Part of Speech (POS) Tagging with Conditional Random Field (CRF)

Description

Task —
- Tag parts of speech, e.g., adverb, verb, noun, etc.
- *Structured prediction task:* Predict over exponentially (or even infinitely) large set of classes

Formulation

- Input: Sequence of words w
- Output: Sequence of tags $t \in T$
- Can be considered a shortest or highest-scoring path search problem on a *trellis*:
  - Directed Acyclic Graph (DAG), where nodes are divided into vertical slices and each slice is associated with a timestamp
  - Each node corresponds to a distinct state
  - Each arrow represents a transition to a new state in the next slice

Point of departure —
- Log-linear model: $p(t \mid w) = \frac{\exp(\text{score}(t,w))}{\sum_{t'} \exp(\text{score}(t',w))}$
- Challenge: Naively computing the normalizer $\sum_{t'} \exp(\text{score}(t',w))$ would take exponential time $\mathcal{O}(|T|^N)$, where $N = |w|$ is the length of the sentence
- Solution: Consider a scoring function that is additively decomposable over tag bigrams: $\text{score}(t,w) = \sum_{n=1}^{N} \text{score}(\langle t_{n-1}, t_n \rangle, w)$ where $t_0 = \text{BOS}$ tag and $t_N = \text{EOS}$ tag
- Then, we have: $p(t \mid w) = \frac{\exp\left(\sum_{n=1}^{N} \text{score}(\langle t_{n-1}, t_n \rangle, w)\right)}{\sum_{t'} \exp\left(\sum_{n=1}^{N} \text{score}(\langle t'_{n-1}, t'_n \rangle, w)\right)}$
- If we take the denominator:
  - $\sum_{t_1:t_N} \exp\left(\sum_{n=1}^{N} \cdots\right) = \sum_{t_1:t_N} \prod_{n=1}^{N} \exp(\dots) = \sum_{t_1:t_{N-1}} \sum_{t_N} \prod_{n=1}^{N} \exp(\dots)$ since last tag does not depend on predecessors
  - $= \sum_{t_1:t_{N-1}} \prod_{n=1}^{N-1} \exp(\dots) \sum_{t_N} \prod_{n=N}^{N} \exp(\text{sco}$ based on distributivity of $\otimes$ over $\oplus$
  - Probabilities of all children nodes, given their parent node, sum to 1
  - All nodes have EOS as a child to ensure that each of the (possibly infinitely many) paths has a finite length

- $= \sum_{t_1:t_{N-1}} \prod_{n=1}^{N-1} \exp(\dots) \sum_{t_N} \exp(\text{score}(\langle t_{N-1}, t_N \rangle, w))$
- $= \sum_{t_1} [\exp(\text{score}(\langle t_0, t_1 \rangle, w)) \times \sum_{t_2} [\exp(\dots) \times \dots \times \sum_{t_N} \exp(\text{score}(\langle t_{N-1}, t_N \rangle, w))]]$

- We've moved from an exponential number of terms to a linear number of terms
- We go from having a score for each path in the trellis to a score for each edge

Backward algorithm —

- Algorithm for computing the denominator
  1. For t_{N-1}: $\beta(w, t_{N-1}, N-1) \leftarrow \exp(\text{score}(\langle t_{N-1}, \text{EOS} \rangle, w))$
     - Handles scores of edges incoming to EOS
     - β is memoized
  2. For $n \in N-2, \dots, 1$: For $t_n \in T$: $\beta(w, t_n, n) \leftarrow \sum_{t_{n+1}} \exp(\text{score}(\langle t_n, t_{n+1} \rangle, w)) \times \beta(w, t_{n+1}, n+1)$
     - Handles scores over normal edges
     - β is memoized
  3. For t_1: $\beta(w, t_0, 0) \leftarrow \beta(w, t_0, 0) + \exp(\text{score}(\langle t_0, t_1 \rangle, w)) \times \beta(w, t_1, 1)$
     - Handles scores of edges outgoing from BOS
     - β is returned
- $\beta(w, t_n, n)$ are *backward variables* that contain the sum of the scores of all paths starting at EOS and ending at tag t_n in position n.
- Therefore: $\text{denominator} = \beta(w, t_0, 0) = \beta(w, \text{BOS}, 0)$
- Complexity:
  - Time complexity: $\mathcal{O}(N|T|^2)$, given that we compute $|T| \cdot N$ backward variables and for each of them, compute the sum over $|T|$ successor tags
  - Space complexity: $\mathcal{O}(N|T|)$, since we have to keep $|T| \cdot N$ backward variables in memory
- Alternatively, the same can be achieved with a *forward algorithm*, starting from BOS t_0 and going forward towards EOS t_N

Viterbi algorithm —

- Motivation:
  - Let us now assume we are not interested in the exact probability $p(t \mid w)$, but only in which sequence of tags t has the highest probability
  - Then we can ignore the denominator and have: $p(t \mid w) \propto \exp(\text{score}(t, w))$
  - We need to solve: $t^* = \text{argmax}_t \exp(\text{score}(t, w)) = \text{argmax}_t \prod_{n=1}^{N} \exp(\text{score}(\langle t_{n-1}, t_n \rangle, w))$
- Algorithm for computing the score of the best tag sequence t^* and recovering the sequence itself
  1. For t_{N-1}: $v(w, t_{N-1}, N-1) \leftarrow \exp(\text{score}(\langle t_{N-1}, \text{EOS} \rangle, w))$
  2. For $n \in N-2, \dots, 1$: For $t_n \in T$:
     - $v(w, t_n, n) \leftarrow \max_{t_{n+1} \in T} [\exp(\text{score}(\langle t_n, t_{n+1} \rangle, w)) \times v(w, t_{n+1}, n+1)]$ for each tag t_n, stores the score of the next best tag t_{n+1}
     - $b(t_n, n) \leftarrow \text{argmax}_{t_{n+1} \in T} [\exp(\text{score}(\langle t_n, t_{n+1} \rangle, w)) \times v(w, t_{n+1}, n+1)]$ for each tag t_n, stores the tag t_{n+1} that gave the best score
  3. For t_1:
     - $v(w, t_0, 0) \leftarrow \max_{t_1} [v(w, t_0, 0), \exp(\text{score}(\langle t_0, t_1 \rangle, w)) \times v(w, t_1, 1)]$
     - $b(t_0, 0) \leftarrow \text{argmax}_{t_1} [v(w, t_0, 0), \exp(\text{score}(\langle t_0, t_1 \rangle, w)) \times v(w, t_1, 1)]$
  4. For $n = 1, \dots, N$: Recover the best sequence using backpointers, by always plugging in last t: $t_n \leftarrow b(t_{n-1}, n-1)$
  5. Return $t_{1:N}$ and $v(w, t_0, 0)$
- $b(t_n, n)$ are *backpointers* that point to the t_{n+1} tag in t^*
- $v(w, t_n, n)$ are *Viterbi variables* that contain the score of t^* starting at EOS and ending with tag t_n in position n
- Complexity:
  - For scoring function over tag bigrams: $\mathcal{O}(N|T|^2)$
  - For scoring function over tag trigrams: $\mathcal{O}(N|T|^3)$
  - For scoring function over k-order dependencies: $\mathcal{O}(N|T|^{k+1})$
- Alternatively, the same can be achieved with a *forward Viterbi algorithm*, starting from BOS t_0 and going forward towards EOS t_N
- Then the backpointers point to the t_{n-1} tag in t^*

Dijkstra's algorithm —

- Alternative to the Viterbi algorithm
  1. Initialize:
     - Popped $\leftarrow \{\}$.
     - Queue $\leftarrow$ PriorityQueue() (returns best scores first)
     - $\gamma \leftarrow -\infty$
  2. Push $\langle \langle 0, \text{BOS} \rangle, 0 \rangle$ to queue
  3. While $|\text{queue}| > 0$:
     (a) Pop $\langle \langle n, t \rangle, \text{score} \rangle$ from queue, add $\langle \langle n, t \rangle, \text{score} \rangle$ to popped, $\gamma[n, t] \leftarrow \text{score}$
     (b) If $n = |w|$ or stop early: Return score
     (c) If $n < |w|$: For $t' \in T$: If $\langle n+1, t' \rangle$ is not in popped: Push: $\langle \langle n+1, t' \rangle, \text{score}(t, t', w) + \gamma[n, t] \rangle$ to queue
  4. Return γ
- Complexity:
  - For scoring function over tag bigrams: $\mathcal{O}(N|T|^2 \log(N|T|))$, where $\log(N|T|)$ stems from push and update operations on the queue
  - Dijkstra is generally slower than Viterbi, unless we leverage early stopping conditions

Generalized algorithm —

- A more general formulation of backward, Viterbi, and Dijkstra algorithms using semirings
- Backward algorithm: Uses inside semiring: $\langle \mathbb{R}_0^+ \cup \{+\infty\}, +, \times, 0, 1 \rangle$
- Viterbi algorithm:
  - Uses Viterbi semiring: $\langle [0, 1], \max, \times, 0, 1 \rangle$
  - Challenge: Multiplying probabilities for long sequences may cause numbers to go to 0, leading to numerical underflow
  - Solution: Revert to log probabilities with the semiring: $\langle \mathbb{R} \cup \{-\infty\}, \max, +, -\infty, 0 \rangle$.
- Dijkstra's algorithm: Uses arctic semiring: $\langle \mathbb{R} \cup \{-\infty\}, \max, +, -\infty, 0 \rangle$
- In addition to probabilities (as we calculate in the inside/Viterbi/log/arctic semirings), we can compute entropy using the expectation semiring:
  - Expectation semiring given by $\langle \mathbb{R} \times \mathbb{R}, \oplus, \otimes, \underline{0}, \underline{1} \rangle$ where:
    - $\langle x, y \rangle \oplus \langle x', y' \rangle = \langle x + x', y + y' \rangle$
    - $\langle x, y \rangle \otimes \langle x', y' \rangle = \langle x \cdot x', x \cdot y' + x' \cdot y \rangle$
    - $\underline{0} = \langle 0, 0 \rangle$
    - $\underline{1} = \langle 1, 0 \rangle$
  - Instead of $\omega = \exp(\text{score}(\langle t_{n-1}, t_n \rangle, w))$, we do: $\omega = \langle w, -\omega \log(w) \rangle$ as the weight for each arc in the CRF
  - Running the backward algorithm with these weights and in the expectation semiring, we can additionally compute unnormalized entropy: $H_u = -\sum_{t \in T^N} \exp(\text{score}(t, w)) \cdot \text{score}(t, w)$
  - We can then compute the normalized entropy as: $H = Z^{-1} H_u + \log(Z)$

Scoring functions —

- So far, we only imposed the requirement that the scoring function is additively decomposable, but we have not specified it further
- *Hidden Markov Model*: $\text{score} = \text{transition}(t_{n-1}, t_n) + \text{emission}(t_n, w_n) =$ transition probability (tag-tag pairs) + emission probability (word-tag pairs, excl. BOS and EOS)
- A more complex example of scoring functions is a neural network with inputs $\langle t_{n-1}, t_n \rangle$ and w
- E.g. for the sentence 'The girl drinks':
  - $\text{score}(w = \text{The girl drinks}, t = \langle D, N, V \rangle) = \sum_{n=1}^{n+1} \text{score}(\langle t_{n-1}, t_n \rangle, w)$
  - $= \text{score}(\langle \text{BOS}, D \rangle, w) + \text{score}(\langle D, N \rangle, w) + \text{score}(\langle N, V \rangle, w) + \text{score}(\langle V, \text{EOS} \rangle, w)$
  - Combination of emission and transition features: $(w_1 = \text{The}, y_1 = D) + (y_1 = D, y_0 = \text{BOS}) + (w_2 = \text{girl}, y_2 = N) + (y_2 = N, y_1 = D) + (w_3 = \text{drinks}, y_3 = V) + (y_3 = V, y_2 = N) + (y_4 = \text{EOS}, y_3 = V)$

Optimization

Objective function —

- Assume we have a dataset of K iid data points $(w^{(i)}, t^{(i)})$
- Log Likelihood: $\sum_{k=1}^{K} \log(p(t^{(k)} \mid w^{(k)})) = \sum_{k=1}^{K} \log\left(\frac{\exp(\text{score}(t^{(k)}, w^{(k)}))}{\sum_{t'} \exp(\text{score}(t', w^{(k)}))}\right) = \sum_{k=1}^{K} \left[\text{score}(t^{(k)}, w^{(k)}) - \log \sum_{t'} \exp(\text{score}(t', w^{(k)}))\right]$
- Risk of overflow since sum makes things big and log makes things small
- If we want to use a temperature parameter T, we have: $\sum_{k=1}^{K} \log(p(t^{(k)} \mid w^{(k)})) = \sum_{k=1}^{K} \log\left(\frac{\exp(\text{score}(t^{(k)}, w^{(k)}))/T}{\sum_{t'} \exp(\text{score}(t', w^{(k)}))/T}\right) = \sum_{k=1}^{K} \left[\text{score}(t^{(k)}, w^{(k)}) - T \log \sum_{t'} \exp\left(\frac{\text{score}(t', w^{(k)})}{T}\right)\right]$
- As $T \to 0$: $\sum_{k=1}^{K} \left[\text{score}(t^{(k)}, w^{(k)}) - \max_{t'} \text{score}(t', w^{(k)})\right]$

Optimization —

- Gradient can be computed using backpropagation linear in N, given the Viterbi algorithm for forward propagation linear in N
- Can be optimized using subgradient descent, since max is not differentiable everywhere
- Because the linear objective is convex, subgradient descent will converge

50 Syntax | Syntactic Parsing

Description

Task — Assign syntactic structure, i.e., parse tree, to a sentence by breaking it down into a hierarchy of constituents that is grammatically correct (similar to dependency parsing)

Terminology —

- *Constituent*: Sequence of words that function as a coherent unit resp. nodes in tree
  - *Terminals*: Words
  - *Non-terminals*: Abstractions over words, e.g., noun phrases
- *Grammar*: Set of rules (productions), according to which sentences (strings) can be formed from a vocabulary
  - *Context-free grammar (CFG)*: Grammar where rules are applied regardless of context
- *Parse tree*:
  - Represents both syntactic structure of a string and its derivation under grammar
  - Can be considered a bag of production rules and a multiset (i.e., set with repeats)

Context-Free Parsing

Context-free grammar $(\mathcal{N}, \mathcal{S}, \Sigma, \mathcal{R})$ —

- Finite set of non-terminal symbols $\mathcal{N} = \{N_1, N_2, \dots\}$
- Distinguished start non-terminal $\mathcal{S}$
- Alphabet of terminal symbols $\Sigma = \{a_1, a_2, \dots\}$
- Set of production rules of the form $N \to \alpha$, where N is non-terminal and $\alpha \in (\mathcal{N} \cup \Sigma)^*$, i.e. α is either terminal or non-terminal
- Examples:
  - Language $\{a^i b^j c^k \mid i, j, k \geq 0, i + j = k\}$ is generated by: $\mathcal{S} \to a\mathcal{S}c \mid W \mid \varepsilon$ $W \to bW \mid \varepsilon$
  - Language $\{a^i b^j c^k \mid i, j, k \geq 0, i + k = j\}$ is generated by: $\mathcal{S} \to KW$ $K \to aKb \mid \varepsilon$ $W \to bWc \mid \varepsilon$
  - Language $\{w \in \{a, b\}^* \mid w \text{ contains min. } 3a\}$ is generated by: $\mathcal{S} \to WaWaWaW$ $W \to aW \mid ab \mid \varepsilon$

Pumping lemma —

- In any CFG, long enough strings have some kind of repeating structure
- Then, there exists a k such that a string s with $|s| > k$ can be written as $s = uxyzv$, where:
  - u, y, v are "fixed"parts, whereas x and z are "pumpable"parts
  - $ux^n yz^n v$ is also in the CFG
  - x and z are not empty
  - $|xyz| < k$

  Proof:
  - Consider a parse tree with height $|\mathcal{V}| + 1 = N$. This tree uses all symbols in $|\mathcal{V}|$. Thus, any tree with height $> N$ must include some repetition of symbols
  - Let $k >$ length of any string yielded by a tree with max height N
  - If $|s| \geq k$, then s must be yielded by a tree with height $> N$, meaning s must include repetition of at least one symbol A
  - From this, properties follow:
    - If T is the entire parse tree, T_A is the parse tree with root upper A, and T_{AA} is the parse tree with root lower A
    - T_A has height $\leq N$, i.e., $|xyz| < K$
    - If x and z are empty (i.e., we replace T_A with T_{AA}), we have a smaller tree that yields s, which is a contradiction
    - We can extend the proof to any n
- Pumping lemma shows that languages that require strict equality between counts of 3+ symbols cannot be generated by a CFG

INSERT IMAGE 1

Span —

- Contiguous segment of a sentence from word w_i to w_{j-1}: $[i, j]$
- Span is admissible if it is possible to construct a parse tree, which covers exactly that span and is a valid constituent according to the grammar, i.e., there exists a non-terminal X such that $X \to w[i : j]$
- For string of length M, we have: $[M, M+1)$ for span size $= 1$ $[1, M+1)$ for span size $= M$ More generally: $[M - \text{span size} + 1, M + 1)$

Chomsky Normal Form (CNF) —

- CFG, where production rules follow specific structure:
  - $N_1 \to N_2 N_3$ are non-terminal productions
  - $N_- \to a_-$ are terminal productions
- Prohibits cyclic rules

Probabilistic CFGs — Motivation:

- Often, multiple syntactic structures are grammatically admissible, i.e., the string

is ambiguous
- This is the case when a non-terminal can be produced by two rules

Probabilistic CFGs $(\mathcal{N}, \mathcal{S}, \Sigma, \mathcal{R}, \mathcal{P})$:
- Additionally define a set of probabilities $\mathcal{P}$ for each production rule
- Probabilities are locally normalized over each transition: $\sum_k p(N \to \alpha_k) = 1$ where $N \to \alpha_1, \ldots, N \to \alpha_k$ are expansions of node N
- For example: $\text{NP} \to \text{VP}(p = 0.6) \mid \text{Adj}(p = 0.4)$ with total $p = 1$
- Then, the probability of a parse tree is the multiplication of the probabilities of the rules used to create the tree: $p(t) = \prod_{r \in t} p(r) = p(S \to S_1 S_2)^{M-1} \times p(S \to X)^M \times p(X \to Y)^M$ where:
  - $p(S \to S_1 S_2)^{M-1}$: repeats the rule $S \to SS$ $M-1$ times to obtain M starting nodes
  - t: tree
  - r: rule
  - S: start non-terminal
  - X: non-terminal
  - Y: terminal
  - M: sentence length

Weighted CFGs —
- A more general formulation of PCFGs
- Probabilities are globally normalized over all possible parse trees
- Probability of a parse tree: $p(t) = \frac{1}{Z} \prod_{r \in t} \exp(\text{score}(r))$ where: $Z = \sum_{t' \in \mathcal{T}} \prod_{r \in t'} \exp(\text{score}(r))$
- Challenge: The denominator $Z(s)$ is infinitely large, potentially even larger than Σ^* for ambiguous strings
- Initial solution: Probability of a parse tree, conditioned on string s: $p(t \mid s) = \frac{1}{Z(s)} \prod_{r \in t} \exp(\text{score}(r))$ where: $Z = \sum_{t' \in T(s)} \prod_{r \in t'} \exp(\text{score}(r))$ where $T(s)$ is set of trees that yield s
- Challenge: The denominator $Z(s)$ is still potentially infinitely large due to cyclic rules
- Final solution: Revert to CNF. Then, $|T(s)|$ for a string of length M is the number of rooted binary trees, which is the Catalan number C_{M-1}
- With CNF, probability of a parse tree: $p(t \mid s) = \frac{1}{Z(s)} \prod_{N_i \to N_j N_k, \in t} \exp(\text{score}(N_i \to N_j N_k)) \times \prod_{N_l \to a, \in t} \exp(\text{score}(N_l \to a))$ where
  - $Z(s) = \sum_{t' \in T(s)} \prod_{N_i \to N_j N_k, \in t'} \exp(\text{score}(N_i \to N_j N_k)) \times \prod_{N_l \to a, \in t'} \exp(\text{score}(N_l \to a))$
  - Non-terminal productions: $\prod_{N_i \to N_j N_k, \in t} \exp(\text{score}(N_i \to N_j N_k))$
  - Terminal productions: $\prod_{N_l \to a, \in t} \exp(\text{score}(N_l \to a))$
- Can be trained via MLE with backpropagation and gradient descent

Cocke-Kasami-Younger (CKY) Algorithm —
INSERT IMAGE 2
- Tasks:
  - Solve the *recognition problem*: Determine if a given string is admissible by the grammar
  - Compute the normalizing constant Z
  - Find one or k best syntactic parses
- Requires grammar in CNF
- Algorithm to compute normalizing constant Z:
  - Uses real semiring with $+, \times, 0, 1$
  - Alternatively, uses log-sum-exp semiring with $\log_+, +, -\infty, 0$ where $\log_+(x, y) = \log(e^x + e^y)$,

1. $N \leftarrow |s|$ chart $\leftarrow 0$
2. For $n = 1, \ldots, N$:
   For $X \to S_n$ (handles single word tokens) where $X \equiv N_l$, $S_n \equiv a$ in weighted CGF:
   (a) If real semiring: $\text{Chart}[n, n+1, X]$ += $\exp(\text{score}(X \to S_n))$ (weight assigned to the given rule)
   (b) If log-sum-exp semiring: $\text{Chart}[n, n+1, X] = \log_+(\text{Chart}[n, n+1, X])$
3. For span $= 2, \ldots, N$:
   For $i = 1, \ldots, N - \text{span} + 1$ (i: beginning of span):
   $k \leftarrow i + \text{span} - 1$ (k: end of span)
   For $j = i+1, \ldots, k-1$ (j: breaking point of span):
   For $X \to YZ$ where $X \equiv N_i$, $Y \equiv N_j$, $Z \equiv N_k$ in weighted CGF:
   (a) If real semiring: $\text{Chart}[i, k, X]$ += $\exp(\text{score}(X \to YZ)) \times \text{Chart}[i, j, Y] \times \text{Chart}[j, k, Z]$ where
   - Score is weight assigned to the given rule
   - Chart entries contains normalization factors for subtrees at i, j resp. j, k

   (b) If log-sum-exp semiring: $\text{Chart}[i, k, X] = \log_+(\text{Chart}[i, k, X])$
4. Return $\text{Chart}[1, N+1, \mathcal{S}]$, which contains the normalization constant

- Complexity:
  - Runtime complexity: $\mathcal{O}(N^3 |\mathcal{R}|)$
  - Space complexity: $\mathcal{O}(N^2 |\mathcal{R}|)$
  - If, for each span length, there is only one admissible span and we know i (beginning) and j (breaking point), we can omit looping over i and j and set: $i = N - \text{span size} + 1$, $j = i + 1$
  - Then, runtime complexity: $\mathcal{O}(N |\mathcal{R}|)$
- Algorithm to find best parse: Run above algorithm with $\max, +$ semiring
- Algorithm to find whether string is admissible by grammar: Run above algorithm with Boolean semiring

51 Syntax | Dependency Parsing

Description

Task — Assign syntactic structure, i.e., parse tree, to a sentence by breaking it down into a hierarchy of dependencies (similar to syntactic parsing)

Formulation

- Done via *spanning tree*:
  - Connects N nodes via $N-1$ edges
  - Contains no cycles
- For dependency parsing, we also require:
  - The tree has exactly one root node, with only one outgoing edge
  - All non-root nodes have exactly one incoming edge
  - Edges are directed
  - Edges are labeled
- We can turn syntactic parsing into dependency parsing based on rules
- E.g. INSERT IMAGE 1
- Types of dependency trees:
  - *Projective trees*:
    - No crossing arcs
    - Closely related to constituents (syntactic parsing)
    - Can be treated with algorithms similar to CKY
    - Note: In syntactic parsing, dependency relationships are always nested within constituents, meaning that syntactic parsing trees are always projective
    - Not in focus in the following
  - *Non-projective trees*:
    - Crossing arcs
    - In focus in the following

Probability of spanning tree —
- $p(t \mid w) = \frac{1}{Z} \exp(\text{score}(t, w))$ where $Z = \sum_{t'} \exp(\text{score}(t', w))$
- Challenge: Naively, if $|w| = N$:
  - It takes $\mathcal{O}(N^N)$ time to compute Z, if we don't impose a spanning tree structure and every node can depend on any other node, including itself
  - There are $\mathcal{O}(N^{N-2})$ spanning trees we can form from N nodes, if we impose a spanning tree structure constraint
  - There are $\mathcal{O}((N-1)^{N-2})$ spanning trees we can form from N nodes, if we impose the requirement that there is exactly one root node
- Solution:
  - *Edge factored assumption*: $p(t \mid w) = \frac{1}{Z} \prod_{(i \to j) \in t} \exp(\text{score}(i, j, w)) \exp(\text{score}(r, w))$ where $(i \to j)$ is an edge in the tree, r is the root, and $Z = \sum_{t'} \prod_{(i \to j) \in t'} \exp(\text{score}(i, j, w)) \exp(\text{score}(r, w))$
  - *Kirchhoff's Matrix-Tree Theorem*: Method for counting the number of undirected spanning trees in $\mathcal{O}(N^3)$ time
  - *Tutte's Matrix-Tree Theorem*: Generalizes Kirchhoff's Matrix-Tree Theorem to directed spanning trees:
    - Take adjacency matrix and root vector:
      - *Adjacency matrix*: $A_{ij} = \exp(\text{score}(i, j, w))$
      - *Root vector*: $\rho_j = \exp(\text{score}(j, w))$
    - Construct *Laplacian matrix*: Not accounting for constraint that there is only one root node: $L_{ij} = \begin{cases} A_{ij} & \text{if } i \neq j \\ \rho_j + \sum_{k \neq i} A_{kj} & \text{otherwise} \end{cases}$
      Accounting for constraint that there is only one root node: $L_{ij} = \begin{cases} \rho_j & \text{if } i = 1 \\ \sum_{k=1, k \neq i} A_{kj} & \text{if } i = j \\ -A_{ij} & \text{otherwise} \end{cases}$
    - $|L| = Z$
  - Then, we can compute Z under the edge factored assumption in $\mathcal{O}(N^3)$
  - We optimize log-likelihood to derive parameters: $\sum_{i=1}^{l} \log p(t^{(i)} \mid w^{(i)}) = \sum_{i=1}^{l} \text{score}(t^{(i)}, w^{(i)}) - \log Z(w^{(i)})$
    Gradient of log-likelihood can be computed via backpropagation in $\mathcal{O}(N^3)$
  -

Construct and decode tree — Construct tree:
- $N = |w|$ nodes plus 1 extra root node
- $(N+1)^{(N+1)}$ edges
- Each edge with a label and weight

Decode tree:
- Task of finding the maximum-weight directed spanning tree
- Challenge: To perform decoding, greedy *Kruskal's algorithm* does not work, since this selects the maximum-weight arc and then builds from there, which may be suboptimal in the directed case
- Solution: *Chu-Liu/Edmonds algorithm*

Chu-Liu/Edmonds algorithm: INSERT IMAGES
- Without root constraint: Can be run in $\mathcal{O}(N^2)$ time
  1. Greedy algorithm selects best incoming edge for each node → this can cause cycles
  2. Contract cycles
     - Edges fully in the cycle are dead
     - Edges fully outside the cycle are external
     - Edges exiting the cycle are exits
     - Edges entering the cycle are enters
  3. Break cycle: For each edge, break cycle by removing edges that are also incoming at the node where the enter edge is
  4. Reweight: Add weights of remaining edges to enter edge
  5. Now, greedy algorithm selects tree without cycles
  6. Then, we can re-expand: Pick edge of highest-scoring enter edge in contracted form, pick edges of remaining dead edges for that enter edge, which were used to re-weight
- With root constraint:
  - Naively: Run above algorithm N times, fixing each edge as the only one emanating from the root, adds factor N to runtime
  - Solution: Adjusted algorithm
  1. Run steps $1-4$ as above
  2. If there are multiple edges emanating from the root: For each root edge, calculate cost of deleting this edge:
     Cost = Weight of root edge - weight of next-best incoming edge to target node
  3. Remove edge with lowest cost
  4. Re-run greedy algorithm in contracted form
  5. If this leads to a cycle: Contract, then re-expand
     Otherwise: Re-expand

52 Semantics | Semantic Parsing

Description

Task — Assign semantic structure, i.e. logical form, to a sentence

Description —
- Principle of compositionality:
  - The meaning of a complex expression is a function of the meanings of that expression's constituent parts
  - → We can syntactically parse a sentence
  - → We can then construct the semantic representation bottom-up
- Applying the principle of compositionality:
  - Syntactic rule, e.g. S → NP VP
  - Induces semantic rule, e.g. s.sem → VP.sem(NP.sem): to get the semantic representation of sentence s, we apply the semantic representation of the verb phrase (a function) to the semantic representation of the noun phrase

Lambda Calculus

Basics —
- Basic terms: Variables $x, y, z, \ldots$
  - Free variables: Do not occur in the scope of any abstraction that holds its name
  - Otherwise, bound variables: Bound by the abstraction with the smallest scope
  - E.g. $((\lambda x.[\lambda y.[(x((\lambda x.[x]x)y))]]\lambda x.[x])z)$ where
    - z is unbound
    - Other variables are bound by same-colored abstractions
    - Scopes of abstractions are indicated by square brackets

  New terms can be constructed using 2 recursive rules:
  - 1) Abstraction:
    - If M is a term, N is a term, and x is a variable:
      - λx is an abstraction
      - $\lambda x.M$ is a function that takes x as input and produces M as output by replacing every free occurrence of x in M with whatever the function is applied to (e.g. N)
      - $\lambda x.MN$ is a function applied to N
      - We denote the output of $\lambda x.MN$ as $M[x := N]$
      - In the output of $\lambda x.MN$, only $M[x := N]$ remains, and $\lambda x.$ and N disappear
    - Scope of abstraction: The scope of abstraction λx in expression $\lambda x.M$ is M
  - 2) Application: If M and N are terms, (MN) is a term
- Enriched lambda calculus offers additional components:

- Logical constants: Represent objects (e.g. Boston) and relationships (e.g. likes)
- Variables:
  - Undetermined logical constants
  - Objects are represented in lowercase as $x, y, z, \ldots$ and are input to $\lambda x.f(x)$)
  - Relations are represented in uppercase as $P, Q, R, \ldots$ and are input to $\lambda P.P(\ldots)$
- Literals: Formed by applying relations to objects (e.g. likes(Alex, y), $P(x, y)$)
- Every relation symbol has an *arity* that determines the number of objects it relates
- Logical connectives and quantifiers: $\exists, \forall, \neg, \wedge, \vee$

Example — INSERT IMAGE 1

Alpha conversion —

- Process of renaming a variable in a lambda term
- We can rename a variable in an abstraction together with all its occurrences in the scope of the abstraction, provided it remains bound to the same abstraction after renaming
- If the variable remains bound to the same abstraction, and no new variables become bound to the abstraction, the renaming is valid
- E.g.
  $\lambda x.\lambda y.(x((\lambda x.xx)y)) \to \lambda z.\lambda y.(z((\lambda x.xz)y))$ is valid
  $\lambda x.(xy) \to \lambda y.(yy)$ is not valid, since y was free before and is now bound

Beta reduction —

- Process of applying one lambda term to another: $(\lambda x.M)N$
- We can apply one lambda term to another if the free variables in N remain free in $M[x := N]$
- If this is not the case, first apply alpha-conversion to M
- Termination issue: Repeatedly applying beta reductions may not terminate
- E.g. $\lambda y.(z((\lambda x.xz)y)) \to \lambda y.(z(zy))$
  $(\lambda x.\lambda y.(x((\lambda x.xx)y))z) \to \lambda y.(z((\lambda x.xz)y))$
  where
  - *blue* corresponds to M
  - *orange* corresponds to N
  - *red* corresponds to $M[x := N]$
  - ***bold*** corresponds to free variables in M

Equivalence — Two lambda terms are equivalent if they can be obtained from each other via a series of α- and β-conversions

Combinatory Logic resp. SK-Calculus

- Alternative to lambda calculus
- Does not use abstractions
- Basic terms:
  - Variables: $x, y, z, \ldots$
  - Primitive functions resp. combinators: $I, J, K, \ldots$
    - I combinator: Identity function: $Ix = x$
    - K combinator: Constant function that always outputs x: $Kxy = ((Kx)y) = x$
    - S combinator: $Sxyz = (xz(yz)) = ((xz)(yz)$
- Terms are recursively constructed via application: If M and N are terms, (MN) is a term
- Additional combinators:
  - B combinator resp. composition combinator: Composition function: $Bxyz = (x(yz))$
  - C combinator: $Cxyz = ((xz)y)$
  - T combinator resp. type-raising combinator: $Txy = yx$
- Parentheses are left-associative, e.g. $(Kxyz) = (((Kx)y)z)$
- Reduction:
  - SKK and I are extensionally equivalent: $S(KK)x = SKKx = Kx(Kx) = x$
  - Therefore, the SK basis is complete, and we don't need I operator

Transforming Lambda Calculus into SK-Calculus

Apply recursively defined transformation T:

1. $T(x) = x$ for every variable x
2. $T[(E_1 E_2)] = (T[E_1]T[E_2])$ for all lambda terms E_1 and E_2
3. $T[\lambda x.E] = (KT[E])$ for every lambda term where x is bound
4. $T[\lambda x.x] = (SKK) = I$
5. $T[\lambda x.\lambda y.E] = T[\lambda x.T[\lambda y.E]]$ for every lambda term E where x is free
6. $T[\lambda x.(E_1 E_2)] = (ST[\lambda x.E_1]T[\lambda x.E_2])$ for all lambda terms E_1, E_2 where x is free in at least one of the two terms

Linear Indexed Grammar resp. Combinatory Categorial Grammars (CCG)

Features —

- Lexicalized grammar: Every word is associated with a syntactic category (including information about syntax and semantics) that defines how it combines with other words
- Compositionality: Semantic representation of a sentence is built in tandem with its syntactic derivation
- Flexible word order: Supports languages with complex or free word order (e.g. cross-serial dependencies, long-distance dependencies, coordination)
- The latter two are advantages over context-free grammars

Linear Indexed Grammar $\langle \mathcal{N}, S, \mathcal{I}, \Sigma, \mathcal{R} \rangle$ —

- $\mathcal{N}$: Set of non-terminal symbols $N_1, N_2, N_3, \ldots$
- S: Start non-terminal
- $\mathcal{I}$: Finite set of indices $f, g, h, \ldots$
- Σ: Alphabet of terminal symbols $a_1, a_2, a_3, \ldots$
- $\mathcal{R}$: Set of production rules of the following forms:
  - $N[\sigma] \to \alpha M[\sigma]\beta$
  - $N[\sigma] \to \alpha M[f\sigma]\beta$
  - $N[f\sigma] \to \alpha M[\sigma]\beta$
  - Where
    - N and M are non-terminals
    - α and β are strings of terminals and non-terminals
    - σ is the stack passed to exactly one non-terminal on RHS

Combinatory Categorial Grammar $\langle \mathcal{V}_T, \mathcal{V}_N, S, f, \mathcal{R} \rangle$ —

- $\mathcal{V}_T$: Set of terminals
- $\mathcal{V}_N$: Atomic categories: Set of non-terminals
- S: Start non-terminal $\in \mathcal{V}_N$
- $f : \mathcal{V}_T \to \mathcal{C}(\mathcal{V}_N)$: Lexicon: Function that maps elements of $\mathcal{V}_T \cup \{\varepsilon\}$ to finite subsets of $\mathcal{C}(\mathcal{V}_N)$, where $\mathcal{C}(\mathcal{V}_N)$ is the set of categories
- $\mathcal{R}$: Set of production rules, e.g. application

Composed of 2 parts:

- Lexicon: Associates words with categories, which encode the structure
  → Difference to context-free grammar, which encodes the structure in the rules
  - Atomic categories:
    - Category of complete constituents
    - E.g. Harry := ***NP***
  - Complex categories:
    - Built recursively from atomic categories via operators
    - Category of incomplete constituents
    - Function that specifies the type of result and type and direction of its arguments
    - E.g. walks := (**S\NP**).
  - Set of categories $\mathcal{C}(\mathcal{V}_N)$ is the smallest set such that:
    - If $C \in \mathcal{V}_N$, then $C \in \mathcal{C}(\mathcal{V}_N)$
    - If $C_1, C_2 \in \mathcal{V}_N$, then $(C_1/C_2) \in \mathcal{C}(\mathcal{V}_N)$ and $(C_1 \backslash C_2) \in \mathcal{C}(\mathcal{V}_N)$
  - Every category can be written in the form $X = A \mid_m X_m \cdots \mid_1 X_1, \quad m \geq 0$ where
    - A: Atomic category = target of X
    - $X_1, \ldots, X_m$: Arbitrary categories = arguments of X
    - $\mid$ = Backward or forward slash
    - m = Arity of X
- Rules:
  - Specify how categories can be combined into other categories
  - Pattern: $X/Y \quad Y \to X$ where X/Y is function, Y is argument, and X is result
  - Types of rules:
    - Function application:
      - Forward application: $(>)$: If a category expects an argument to the right of (X/Y) and argument Y is available, they combine to form X: $X/Y \quad Y \to X$
      - Backward application: $(<)$: If a category expects an argument to the left of $(X\backslash Y)$ and argument Y is available, they combine to form X: $Y \quad X\backslash Y \to X$
      - AB grammar resp. basic categorical grammar resp. pure categorical grammar:
      - CCGs that only have application rules
      - Have power of CFG:
        $A \to a$ in CCG $\equiv A \to a$ in CFG
        $A \to (A/C)C$ in CCG $\equiv A \to BC$ in CFG
    - Function composition
      - Two functions combine to form a single function: $f, g \to f \circ g$
      - Forward composition: $((B_>)$: $(X/Y) \quad (Y/Z) \to (X/Z)$
      - Backward composition: $((B_<)$: $(Y\backslash Z) \quad (X\backslash Y) \to (X\backslash Z)$
    - Higher-order rules:
      - Forward: $(>^n)$ $X/Y \; Y \mid_n Y_n \ldots \mid_1 Y_1 \to X \mid_n Y_n \ldots \mid_1 Y_1$
      - Backward: $(<^n)$ $Y \mid_n Y_n \ldots \mid_1 Y_1 \; X\backslash Y \to X \mid_n Y_n \ldots \mid_1 Y_1$
      - n is the *degree* of the rule
      - If $n = 0$, corresponds to application rule
    - Type raising:
      - Turn an atomic category into a complex category so it can participate in higher-order functions
      - Forward Type Raising: $(T_>)$ $X \to T/(T\backslash X)$
      - Backward Type Raising: $(T_<)$ $X \to T\backslash(T/X)$
      - E.g.:
      - Intransitive S\NP (e.g. walked)
      - Transitive (S\NP)/NP (e.g. respected)
      - Ditransitive ((S\NP)/NP)/NP (e.g. gave)
  - Operators:
    - Backward slash: $X\backslash Y$: "Give me a Y to my left, and I return an X", means that function looks for argument to its left
    - Forward slash: X/Y: "Give me a Y to my right, and I return an X", means that function looks for argument to its right
    - Operators need to be read from outside in, e.g. (S\NP)/NP:
      - Needs NP to its right
      - Then needs NP to its left
      - Then produces sentence S
  - Rule instance is obtained by substituting $X, Y, Z, \ldots$ by concrete categories, e.g. S, NP, VP, etc.
  - CCGs have finite rules (2^n forward, 2^n backward rules) and, thus, of non-terminals but infinitely many rule instances

CCG parsing:

- Deductive process
- General rule of inference: $\frac{A_1 \quad \ldots \quad A_k}{B}$ where B is a consequence of $A_1, \ldots, A_k$
- Derivation trees:
  - Consist of unary and binary branches:
    - Unary branches: Lexicon entries
    - Binary branches: Composition rules:
      - Can have primary and secondary input:
      - Primary input: Function of rule
      - Secondary input: Argument of rule
      - E.g. $X/Y \; Y\beta \to X\beta$ where X/Y is primary input and $Y\beta$ is secondary input
    - Structure: INSERT IMAGE 2

CKY-style parsing algorithm:

- Let w be the sentence to be parsed, w_i be a token in the sentence, and $w[i, j]$ be the substring from $w_{i+1} \ldots w_j$, and $w_{ii} = \varepsilon$
- Aim: Construct item $[S, 0, n]$
- Derivation tree leading to this outcome has internal nodes $[X, i, j]$
- Axioms: $[x, i, i+1]$ where $w_{i+1} = x$ is a lexicon entry
- Inference rules:
  - Forward:
    $\frac{[X/Y, i, j] \quad [Y\beta, j, k]}{[X\beta, i, k]} \quad X/Y \quad Y\beta \to X\beta$
  - Backward:
    $\frac{[Y\beta, i, j] \quad [X\backslash Y, j, k]}{[X\beta, i, k]} \quad Y\beta \quad X\backslash Y \to X\beta$
- Challenge: With the composition rule, we can grow the arity of the primary input categories and get exponentially many primary input categories
- Solution: Restrict the arity of the categories

Polynomial time algorithm:

- Arity of categories is bounded by grammar constant C_g: $[X, i, j]$ where $\text{ar}(X) \leq C_g$
- Challenge: Categories with arity $> C_g$ can no longer be derived
- Solution:
  - Introduce new rules that decompose longer derivations into smaller pieces
  - Derivation context:
    - None of the combinatory rules used in c touches X, i.e. c may pop and push the argument Y, but never touches X
    - For this reason, c is a derivation context and can be used for any category X across multiple derivations
    - INSERT IMAGE 3
  - New items:
    - $[\mid Y, \beta, i, i', j, j']$ where $\text{ar}(Y\beta) \leq C_g$
    - For any category X, if we can build a derivation tree t' with yield $w[i', j']$ and type $X \mid Y$, then we can build t with yield $w[i, j]$ and type $X\beta$
    - Derivation context C can be combined with any t' with item of form $[X \mid Y, i', j']$
    - c can have any internal structure, as long as X is untouched
    - t' can have any internal structure, only positions i', j' and argument $/Y$ matter for combination with c
  - New inference rules:
    - Context items are derived when the composition of two categories would result in a category with arity $> C_g$:
      $\frac{[X/Y, i, j] \quad [Y\beta, j, k]}{[\mid Y, \beta, i, i, j, k}$ where
      - $X \mid Y \quad Y\beta \to X\beta$
      - $\text{ar}(X\beta) > C_g$
    - For denominator of above equation:
      - When the arity of the context is small enough, it can be recombined with the original derivation: $\frac{[X \mid Y, i', j'] \quad [\mid Y, \beta, i, i', j', j]}{[X\beta, i, j]}$ where $\text{ar}(X\beta) \leq C_g$
      - Contexts can be extended similar to derivation trees:
        $\frac{[\mid Y, \beta/Z, i, i', j', j] \quad [Z_\gamma, j, k,]}{[\mid Y, \beta_\gamma, i, i', j', k]}$ where β_γ

refers to β without $/Z$ (can be ε) and where
  - $X/Z \quad Z_\gamma \to X_\gamma$
  - $\text{ar}(Y\beta_\gamma) \le C_g$

  resp. $\frac{[Z_\gamma,i,j] \quad [[Y,\beta\backslash Z,j,j',k',k]}{[[Y,\beta_\gamma,i,j',k',k]}$ where β_γ refers to β without $/Z$ (can be ε) and where
  - $Z_\gamma \quad X\backslash Z \to X_\gamma$
  - $\text{ar}(Y\beta_\gamma) \le C_g$
  - For denominator of above equations:
  - If the context extended in this way has arity $> C_g$, a new context is derived from the previous context: $\frac{[[Y,\beta/Z,i,i',j',j] \quad [Z_\gamma,j,k]}{[\backslash Z,\gamma,i,i,j,k]}$
  - When the arity of the extended context is small enough, it can be recombined with the original context: $\frac{[[_1Y,\beta|_2Z,i'',i',j',j'']\quad [[_2Z,\gamma,i,i'',j'',j]}{[[_1Y,\beta,i,i',j',j]}$
- Grammar constant C_g is at least as large as maximum arity of category in axioms $= \ell$ (e.g. if "likesïs in axioms, $C_g \ge 2$)
- Grammar constant C_g is at least as large as maximal Y (determined by largest arity a in lexicon) and β (determined by the maximum degree n of composition rules), since we need to restrict the size of $Y\beta$
- Together: $C_g \ge \max\{\ell, a+n\}$

CCG combine syntactic and semantic information when paired with lambda calculus:
- E.g.
  Mary := NP : MARY(x)
  likes := (S\NP)/NP : $\lambda x.\ \lambda y.$ likes(x,y)
  natural language := syntax : semantics
- If we parse a sentence syntactically, we can derive semantics bottom-up in the same order
- hIINSERT IMAGE 4

53 Weighted Finite State Automata (WFSA)

Finite State Automata (FSA)

Task — Determine whether a string is an element of a given language

Formalization —
- $\mathcal{A}$ is a 5-tuple $(\Sigma, Q, I, F, \delta)$
  - Σ is an alphabet
  - Q is a finite set of states
  - $I \subseteq Q$ is the set of initial states
  - $F \subseteq Q$ is the set of final or accepting states
  - $\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$ resp. $q \xrightarrow{a \in \Sigma \cup \{\varepsilon\}} q'$ is a finite multiset of transitions from one state in Q to another state in Q via a symbol in Σ
- Can be represented as a directed, possibly cyclical graph: INSERT IMAGE 1
- Sequentially reads individual symbols of an input string s and transitions from state q to state q' upon reading a symbol a iff $(q,a,q') \in \delta$
- If the automaton, after reading the last symbol of s, ends up in a state $q_f \in F$, the automaton accepts the string

Weighted Finite State Automata (WFSA)

- Generalization of FSA, where transitions are weighted with a semiring
- Unweighted FSA corresponds to WFSA weighted with Boolean semiring
- Formalization: $\mathcal{A}$ is a 7-tuple $(\Sigma, Q, I, F, \delta, \lambda, \rho)$ over a semiring $\mathcal{W} = (\mathbb{K}, \oplus, \otimes, 0, 1)$.
  - Σ is an alphabet
  - Q is a finite set of states
  - $I = \{q \in Q \mid \lambda(q) \ne 0\} \subseteq Q$ is the set of initial states
  - $F = \{q \in Q \mid \rho(q) \ne 0\} \subseteq Q$ is the set of final or accepting states
  - $\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times \mathbb{K} \times Q$ is a finite multiset of transitions from one state in Q to another state in Q via a symbol in Σ
  - $\lambda : Q \to \mathbb{K}$ is an initial weighting function over Q
  - $\rho : Q \to \mathbb{K}$ is a final weighting function over Q
- Can be represented as a directed, possibly cyclical graph: INSERT IMAGE 2

(W)FSA Terminology

- Paths:
  - A path π is an element of δ^* with consecutive transitions: $q_1 \xrightarrow{a_1} q_2 \xrightarrow{a_2} q_3 \cdots q_{N-1} \xrightarrow{a_N} q_N$
  - $\text{p}(\pi) = q_1$ is the beginning state of the path
  - $\text{q}(\pi) = q_N$ is the ending state of the path
  - Length of path $|\pi|$ is the number of transitions
  - Yield of path $s(\pi)$ is the concatenation of symbols on the path
  - Path sets: Denoted by capital Π
    - $\Pi(\mathcal{A})$: Set of all paths in the automaton
    - $\Pi(s, \mathcal{A})$: Set of all paths with yield s
    - $\Pi(q, \mathcal{A})$: Set of all paths starting at q
    - $\Pi(q, q', \mathcal{A})$: Set of all paths from q to q'
    - $\Pi(q, s, q', \mathcal{A})$: Set of all paths from q to q' with yield s
    - $\Pi(\mathcal{Q}, \mathcal{A}) = \bigcup_{q \in Q} \Pi(q)$
    - $\Pi(\mathcal{Q}, \mathcal{Q}', \mathcal{A}) = \bigcup_{q \in Q, q' \in Q'} \Pi(q, q')$
    - $\Pi(\mathcal{Q}, s, \mathcal{Q}'\mathcal{A}) = \bigcup_{q \in Q, q' \in Q'} \Pi(q, s, q')$
- Cycles:
  - A path is a cycle if starting and finishing states are the same
  - A path contains a cycle if the same state appears multiple times
- Transitions:
  - Outgoing arcs from q: $E\mathcal{A}(q) = \{a, b, \dots \mid (q, a, b, \dots) \in \delta\}$
  - Incoming arcs to q: $E^{-1}\mathcal{A}(q) = \{c, d, \dots \mid (\dots, c, d, q) \in \delta\}$
- States:
  - State q is accessible iff $q \in I$ or there exists a path whose weight is not 0 from I to q
  - State q is accessible iff $q \in F$ or there exists a path whose weight is not 0 from q to F
- (W)FSA is unambiguous iff for every string s there is at most one accepting path

(W)FSA Applications

N-gram models, CRFs, HMMs can be represented with (W)FSAs

N-gram models —
- Let $\boldsymbol{y} = (y_1, \dots, y_L)$ be a string with $y_1 = \text{BOS}$, $y_L = \text{EOS}$
- We want to model $p(\boldsymbol{y}) = \prod_{n=1}^{L} p(y_n \mid y_{n-N+1}, \dots, y_{n-1})$
- If we represent this with WFSA:
  - States Q represent all possible sequences of words of length N, i.e., all possible N-grams
  - There are $m + |V|^{n-1} + 1$ states for an N-gram:
    - 1 end state
    - $|V|^{n-1}$ intermediate states
    - $m = \begin{cases} |V|^{n-2} & \text{for N-grams with } n \ge 2 \\ 0 & \text{for unigrams} \end{cases}$
  - Transitions between states represent transitions from one N-gram to another
  - Only transitions between N-grams which can follow each other are allowed, e.g. $y_{-N}\ y_{-N+1} \cdots y_{-1}$ and $y_{-N+1}\ y_{-N+2} \cdots y_0$ receive weights > 0, with the weight representing the probability of observing the new word y_0 given the starting N-gram
- Formalization:
  - $\Sigma = \mathbb{Y} \cup \{\langle\text{BOS}\rangle, \langle\text{EOS}\rangle\}$
  - $Q = \bigcup_{n=0}^{N} \{\langle\text{BOS}\rangle\}^{N-n} \times \mathbb{Y}^{n-1} \times (\mathbb{Y} \cup \{\langle\text{EOS}\rangle\})$
  - $I = \{\langle\text{BOS}, \dots, \text{BOS}\rangle\}$ (N times)
  - $F = Q$
  - $\delta = \Big\{(y_{-N} \cdots y_{-1}, y_0, p(y_0 \mid y_{-N+1}, \dots, y_{-1}), y_{-N+1} \cdots y_0) \text{ for } (y_{-N+1} \cdots y_{-1}) \in \bigcup_{n=0}^{N-1} \{\langle\text{BOS}\rangle\}^{N-1-n} \times \mathbb{Y}^n\Big\}$
    $\delta(q_{ij}, w, 1, q_{ki}) = \begin{cases} \log(p(w_m = k \mid w_{m-1}) = i, \dots, w_{m-n+1} = j)) & \text{if } w \text{ is } \\ -\infty & \text{otherwise} \end{cases}$
  - $\lambda = \langle\text{BOS}\rangle \dots \langle\text{BOS}\rangle \to 1$ (N times)
    $\lambda(q_{ij}) = \log(p(w_m = l \mid w_{m-1} = \text{BOS}, \dots, w_{m-n+1} = \text{BOS}))$
  - $\rho = y_{n-N+1} \cdots y_1\ \langle\text{EOS}\rangle \to 1$
    $\rho(q_{ij}) = \log(p(w_m = \text{EOS} \mid w_{m-n} = i, \dots, w_{m-n+1} = j))$

CRFs —
- Let $|\boldsymbol{x}| = L$ and $|\boldsymbol{y}| = M$
  $\text{score}(\boldsymbol{y}, \boldsymbol{x}) = \boldsymbol{w} \cdot \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y})$
- We assume f decomposes additively: $f(\boldsymbol{x}, \boldsymbol{y}) = \sum_{n=2}^{L} f(y_n, y_{n-1}, \boldsymbol{x}, n)$
- We want to model: $p(\boldsymbol{y} \mid \boldsymbol{x}) = \frac{\exp(\text{score}(\boldsymbol{y}, \boldsymbol{x}))}{Z(\boldsymbol{x})} = \frac{\prod_{n=2}^{L} \exp(\boldsymbol{w} \cdot f(y_n, y_{n-1}, \boldsymbol{x}, n))}{Z(\boldsymbol{x})}$
  where $Z(\boldsymbol{x}) = \sum_{\boldsymbol{y}'} \exp(\text{score}(\boldsymbol{y}', \boldsymbol{x})) = \sum_{y'} \prod_{n=2}^{L} \exp(\boldsymbol{w} \cdot f(y'_n, y'_{n-1}, \boldsymbol{x}, n))$
- $Z(\boldsymbol{x})$ can be computed via the pathsum algorithm, as a generalization of the backward algorithm
- If we represent this with WFSA:
  - States Q represent all hidden states
  - Transitions between states represent CRF scores assigned to hidden state y' after hidden state y, given input sequence $\boldsymbol{x}$
- Formalization:
  - $\Sigma = \{\varepsilon\}$
  - $Q = \mathbb{Y}^M$
  - $I = Q$
  - $F = Q$
  - $\delta = \{(y_n, \varepsilon, \exp(\boldsymbol{w} \cdot f(y_n, y_m, \boldsymbol{x})), y_m)\}$
  - $\lambda = y_n \to \exp(\boldsymbol{w} \cdot f(y_n, \langle\text{BOS}\rangle, \boldsymbol{x}))$
  - $\rho = y_n \to \exp(\boldsymbol{w} \cdot f(\langle\text{EOS}\rangle, y_n, \boldsymbol{x}))$

HMMs — TBA

Weighted Finite State Transducers (WFST)

Task — Transforms input string in a given language to output string in a given language

Formalization —
- $\mathcal{T}$ is an 8-tuple $(\Sigma, \Omega, Q, I, F, \delta, \lambda, \rho)$ over a semiring $\mathcal{W} = (\mathbb{K}, \oplus, \otimes, 0, 1)$
  - Σ is the input alphabet, containing letters or symbols
  - Ω is the output alphabet, containing letters or symbols
  - Q is a finite set of states
  - $I = \{q \in Q \mid \lambda(q) \ne 0\} \subseteq Q$ is the set of initial states
  - $F = \{q \in Q \mid \rho(q) \ne 0\} \subseteq Q$ is the set of final or accepting states
  - $\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times (\Omega \cup \{\varepsilon\}) \times \mathbb{K} \times Q$ is a finite multiset of transitions from one state in Q to another state in Q via a symbol in Σ and in Ω
  - $\lambda : Q \to \mathbb{K}$ is an initial weighting function over Q
  - $\rho : Q \to \mathbb{K}$ is a final weighting function over Q
- Furthermore:
  - s: String over alphabet
  - ε: Empty string
  - $\Sigma^0 = \{\varepsilon\}$
  - $\Sigma^* = \{\varepsilon, \dots\}$ is the set of all possible strings (incl. the empty string ε) that can be formed by concatenation of 0 or more elements from Σ
  - L: String language over an alphabet, subset of Σ^*, containing words
  - x: Input string
  - y: Transliteration of input string
- Can be represented as a directed, possibly cyclical graph INSERT IMAGE 3

Compositions

- Operation to combine two or more WFSTs
- $\mathcal{T}_1 \circ \mathcal{T}_2$ is the composition of $\mathcal{T}_1 = (\Sigma, \Omega, Q_1, I_1, F_1, \delta_1, \lambda_1, \rho_1)$ and $\mathcal{T}_2 = (\Omega, \Gamma, Q_2, I_2, F_2, \delta_2, \lambda_2, \rho_2)$ and is given by: $\mathcal{T} = (\Sigma, \Gamma, Q, I, F, \delta, \lambda, \rho)$ such that: $\mathcal{T}(x, y) = \bigoplus_{z \in \Omega^*} \mathcal{T}_1(x, z) \otimes \mathcal{T}_2(z, y)$
- Naive algorithm to compute composition:
  1. $\mathcal{T} \leftarrow (\Sigma, \Omega, Q, I, F, \delta, \lambda, \rho)$ to create a new WFST
  2. For $q_1, q_2 \in Q_1 \times Q_2$:
     For $(q_1 \xrightarrow{a:b/w_1} q'_1), (q_2 \xrightarrow{c:d/w_2} q'_2) \in E_{\mathcal{T}_1}(q_1) \times E_{\mathcal{T}_2}(q_2)$:
     If $b = c$
     - Add states (q_1, q_2), (q'_1, q'_2) to $\mathcal{T}$ if they have not yet been added
     - Add arc $(q_1, q_2) \xrightarrow{a:d/w_1 \otimes w_2} (q'_1, q'_2)$ to $\mathcal{T}$
  3. For $(q_1, q_2) \in Q$:
     - $\lambda_{\mathcal{T}} = \lambda_1(q_1) \otimes \lambda_2(q_2)$
     - $\rho_{\mathcal{T}} = \rho_1(q_1) \otimes \rho_2(q_2)$
  4. Return $\mathcal{T}$
- Challenge: Runs through many useless states, has time complexity $\mathcal{O}(|Q_1||Q_2|)$
- Solution: Accessible algorithm
- Accessible algorithm:
  - Intuition: Construct all possible pairs of initial states and then expand outwards (depth-first search), adding accessible states
  - Attention: Produced states may still be non-co-accessible
  1. stack $\leftarrow [(q_1, q_2) \mid q_1 \in I_1, q_2 \in I_2]$ (ordered list, allows duplicates)
     visited $\leftarrow \{(q_1, q_2) \mid q_1 \in I_1, q_2 \in I_2\}$ (unordered set, does not allow duplicates)
  2. $\mathcal{T} \leftarrow (\Sigma, \Omega, Q, I, F, \delta, \lambda, \rho)$ to create a new WFST
  3. While |stack| > 0:
     - $q_1, q_2 \leftarrow$ stack.pop()
     - For: $(q_1 \xrightarrow{a:b/w_1} q'_1), \quad (q_2 \xrightarrow{c:d/w_2} q'_2) \in E_{\mathcal{T}_1}(q_1) \times E_{\mathcal{T}_2}(q_2)$:
       If $b = c$:
       · Add states (q_1, q_2), (q'_1, q'_2) to $\mathcal{T}$ if they have not yet been added
       · Add arc $(q_1, q_2) \xrightarrow{a:d/w_1 \otimes w_2} (q'_1, q'_2)$ to $\mathcal{T}$
       · If (q'_1, q'_2) not in visited:
       · Push (q'_1, q'_2) to stack and visited
  4. For $(q_1, q_2) \in Q$:
     - $\lambda_{\mathcal{T}} = \lambda_1(q_1) \otimes \lambda_2(q_2)$
     - $\rho_{\mathcal{T}} = \rho_1(q_1) \otimes \rho_2(q_2)$
- E.g. INSERT IMAGE 4

Transliteration

Basics —
- Mapping strings in one character set to strings in another character set E.g. Muse $\to \mu o \upsilon \sigma \alpha$
- Approach:
  1. Design WFST $\mathcal{T}$ that maps string over Σ to string over Ω: $\mathcal{T}$ acts as a rule set
  2. Encode source sequence $x \in \Sigma^*$ as WFST $\mathcal{T}_x$: Prepare x for alignment
  3. Compute normalizer Z as pathsum of composition $\mathcal{T}_x \circ \mathcal{T}_y$: Compute denominator of $p(\boldsymbol{y} \mid \boldsymbol{x})$
  4. Encode target sequence $y \in \Omega^*$ as WFST $\mathcal{T}_y$: Prepare y for alignment
  5. Sum over all possible alignments using pathsum of composition $\mathcal{T}_x \circ \mathcal{T} \circ \mathcal{T}_y$: Compute numerator of $p(\boldsymbol{y} \mid \boldsymbol{x})$
  6. Compute highest scoring path of composition $\mathcal{T}_x \circ \mathcal{T}$ using e.g. Dijkstra or Floyd-Warshall: Identify most probable transliteration
- How do we design $\mathcal{T}$? Path π from $\mathcal{T}$'s initial states to a final state represents an

alignment of x to y
- How do we use training data?
  - Dataset contains pairs (x, y), where sometimes there is an alignment between x and y
  - We treat the alignment as a latent variable
  - If we represent x as transducer $\mathcal{T}_x$, the composition $\mathcal{T}_x \circ \mathcal{T}$ yields a new transducer whose paths correspond to the paths in $\mathcal{T}$ that have input x: Each path in $\mathcal{T}_x \circ \mathcal{T}$ represents one alignment of x to some string in Ω^*
  - If we represent y as transducer $\mathcal{T}_y$, the composition $\mathcal{T}_x \circ \mathcal{T} \circ \mathcal{T}_y$ yields a new transducer whose paths correspond to the paths in $\mathcal{T}$ that align x to y
  - The probability $p(y \mid x)$ is the total probability of all paths that align x to y
- How do we train? Maximize log-likelihood
  $\sum_{x,y \in \mathcal{D}} \log p(y \mid x) =$ pathsum of $\mathcal{T}_x \circ \mathcal{T} \circ \mathcal{T}_y$, divided by pathsum of $\mathcal{T}_x \circ \mathcal{T}$
- How do we perform inference?
  - Given x
  - Construct $\mathcal{T}_x \circ \mathcal{T}$, defining a probability distribution over all aligned y
  - If we condition on x in WFST $\mathcal{T}_x \circ \mathcal{T}$, this WFST becomes a WFSA
  - Goal is then to find the highest scoring path in this graph
- Why can't we use a CRF?
  - In CRFs, we can only align sequences of the same length, e.g., $|x| = |y|$
  - In WFSTs, we can align sequences of different lengths

Pathsum —
- Inner path weight $w_|(\pi)$ of a path $\pi = q_0 \xrightarrow{a_1/w_1} q_1 \cdots q_{N-1} \xrightarrow{a_N/w_N} q_N$: $w_|(\pi) = \bigotimes_{n=1}^{N} w_n$
- Path weight $w(\pi)$ of a path: $w(\pi) = \lambda(\mathrm{p}(\pi)) \otimes w_| \otimes \rho(\mathrm{q}(\pi))$ where $\mathrm{p}(\pi)$ is the beginning state and $\mathrm{q}(\pi)$ is the ending state
- A path is accepting or successful iff $w(\pi) \neq 0$
- String sum of a string $s \in \Sigma^*$ under $\mathcal{A}$: $A(s) = \bigoplus_{\pi \in \Pi(I,s,F)} w(\pi)$
- Path sum of $\mathcal{A}$: $Z(\mathcal{A}) = \bigoplus_{\pi \in \Pi(\mathcal{A})} w(\pi)$
- Path sum between two states q_n, q_m: $Z(q_n, q_m) = \bigoplus_{\pi \in \Pi(q_n, q_m)} w(\pi)$
- Path sum is equal to the shortest path weight, which can be seen, e.g., with the tropical semiring where $\otimes = +$ and $\oplus = \min$
- In a $\overline{0}$-closed semiring, the shortest path weight depends only on paths of length $N-1$, since paths of length $\geq N$ contain cycles, but cycles do not improve the path sum in $\overline{0}$-closed semirings, since $\overline{1} + a = \overline{1}$
- Challenge: There may be an infinite number of accepting paths (guaranteed if WFSA is cyclical)
- Nonetheless, we can compute the pathsum
- The pathsum problem subsumes several other problems:
  - N-gram normalization
  - CRF normalization
  - Computing marginals in HMMs
- Pathsum in acyclic WFSA:
  - In acyclic WFSA, the number of accepting paths is finite
  - To compute pathsum $Z(\mathcal{A})$, we need a topological sort of the nodes
  - Backward algorithm:
    - Computes $Z(\mathcal{A})$
      1. For $q \in$ Rev-Top$(\mathcal{A})$: (sorts nodes topologically)
         $\beta(q) = \rho(q) \oplus \bigoplus_{q \xrightarrow{a/w} q'} w \otimes \beta(q')$
         i.e. proceeds in reverse topological order, $\oplus$-summing all weights of all paths coming from node q and ending in final state q_F.
      2. Return: $\bigoplus_{q \in I} \lambda(q_|) \otimes \beta(q_|)$
    - Runs in $\mathcal{O}(|\delta|)$ time and $\mathcal{O}(|Q|)$ space
    - Could alternatively also be computed as forward algorithm. Then we would:
      - Visit incoming edges (instead of outgoing)
      - $\oplus$-add the initial weights (instead of final weights) in step (1)
      - Post-$\otimes$-multiply the final weights (instead of initial weights) in step (2)
- Pathsum in cyclical WFSA:
  - In cyclical WFSA, the number of accepting paths is infinite
  - To compute pathsum $Z(\mathcal{A})$, we need a closed semiring
  - Lehmann algorithm:
    - Computes $Z(\mathcal{A})$ by computing Kleene closure of matrixes over arbitraty closed semirings
    - We must first represent WFSA as a matrix:
      - Define $|\Sigma|$ *adjacency matrices*, one for each transition symbol: $\boldsymbol{W}(a)$ is the adjacency matrix of transitions over a
      - INSERT IMAGE 5
      - Since all paths are summed, we can collapse all transitions from q_n to q_m into a single transition without a label, whose weight is the $\oplus$-sum of all original transitions: $\boldsymbol{W}(\mathcal{A}) = \bigoplus_{a \in \Sigma \cup \{\varepsilon\}} \boldsymbol{W}(a)$
    - Broad steps:
      1. Lehmann algorithm computes the $\oplus$-sum over the path between two nodes q_i, q_k with the matrix $\boldsymbol{W}$: $\boldsymbol{R}_{ik} = \bigoplus_{\pi \in \Pi(q_i, q_k)} w_|(\pi)$
         - Cyclical terms in calculation of $\boldsymbol{R}_{ik}$ are denoted with $*$
         - $\boldsymbol{R}_{ik}$ has $|Q|^2$ entries
      2. From $\boldsymbol{R}$, we can compute $Z(\mathcal{A})$: $Z(\mathcal{A}) = \bigoplus_{i,k=1}^{|Q|} \lambda(q_i) \otimes \boldsymbol{R}_{ik} \otimes \rho(q_k)$
    - E.g. INSERT IMAGE 6
    - Intuition:
      - Partition path π between q_i and q_k into subsets of paths: $\boldsymbol{M}^{\leq j}(q_i, q_k)$ are paths that do not cross nodes with indices $> j$
        $\boldsymbol{R}_{ik}^{\leq j}$ is the pathsum for these paths
      - Any path either crosses j or not
      - For paths that do not cross j:
      - $\pi \in \Pi^{\leq j-1}(q_i, q_k)$
      - In that case, the pathsum is: $\boldsymbol{R}_{ik}^{\leq j-1}$
      - For paths that do cross j:
      - π can be decomposed into cycle that starts and ends in j, part before cycle, and part after cycle: $\pi = \pi_{ij} \pi_{jj}^* \pi_{jk}$
      - In that case, we can also decompose the pathsum into partial pathsums: $\boldsymbol{R}_{ik}^{\leq j} = \boldsymbol{R}_{ij}^{\leq j-1} \otimes \boldsymbol{R}_{jj}^{\leq j-1 *} \otimes \boldsymbol{R}_{jk}^{\leq j-1}$
      - This defines a natural procedure to build up the pathsum from the partial pathsums
      - All that's left is the starting condition: $R^{\leq 0} = \boldsymbol{W}$ since $\boldsymbol{W}$ captures paths not passing through intermediary nodes
    - Algorithm:
      1. $\boldsymbol{R}^{(0)} \leftarrow \boldsymbol{W}$
      2. For $j \leftarrow 1$ up to $|Q|$:
         For $i \leftarrow 1$ up to $|Q|$:
         For $k \leftarrow 1$ up to $|Q|$: $R_{ik}^{(j)} \leftarrow R_{ik}^{(j-1)} \oplus \left(R_{ij}^{(j-1)} \otimes R_{jj}^{(j-1)*} \otimes R_{jk}^{(j-1)}\right)$
      3. Return: $I \bigoplus R^{(|Q|)}$
    - E.g. INSERT IMAGE 7
    - Runs in $\mathcal{O}(|Q|^3)$ time
    - Lehmann algorithm encompasses Floyd-Warshall algorithm and Gauss-Jordan algorithm

---

Transliteration — Pathsum in x-conditioned WFSA:
- Let (x, y) be an unaligned training pair
- $\mathcal{A}_x$ is the x-conditioned WFSA:
  - λ is a vector of start weights for starting states: $\lambda_n = \lambda(q_n)$
  - ρ is a vector of end weights for ending states: $\rho_m = \rho(q_m)$
  - Set of transition matrices $\boldsymbol{W}^{(\omega)}$ for any $\omega \in \Omega \cup \{\varepsilon\}$: $\boldsymbol{W}_{nm}^{(\omega)}$ is the weight for $q_n \xrightarrow{\omega} q_m$
  - $\boldsymbol{W}^{(\omega)}$ can be summarized into a single matrix: $\boldsymbol{W} = \sum_{\omega \in \Omega \cup \{\varepsilon\}} \boldsymbol{W}^{(\omega)}$
- For instance: Likelihood: $p(y \mid x) = \frac{\exp(\mathrm{score}(x,y))}{Z(x)}$
- $\mathrm{score}(x, y) = \log \sum_{\pi \in \Pi(y)} w(\pi)$
- Normalizing constant is the pathsum: $Z(x) = \sum_{y' \in \Omega^*} \exp(\mathrm{score}(x, y'))$
  $= \sum_{y' \in \Omega^*} \exp\left(\log \sum_{\pi \in \Pi(y')} w(\pi)\right)$
  $= \sum_{y' \in \Sigma^*} \sum_{\pi \in \Pi(y')} \lambda(\mathrm{p}(\pi)) \otimes w_|(\pi) \otimes \rho(\mathrm{q}(\pi))$
  $= \sum_{y' \in \Sigma^*} \sum_{\pi \in \Pi(y')} \lambda(\mathrm{p}(\pi)) \otimes \prod_{n=1}^{|\pi|} w_n(\pi) \otimes \rho(\mathrm{q}(\pi))$ which is the pathsum
  $= \sum_{q_n, q_m \in Q} \left(\sum_{\pi \in \Pi(q_n, q_m)} w(\pi)\right)$
  $= \sum_{q_n, q_m \in Q} \sum_{\pi \in \Pi(q_n, q_m)} \lambda(\mathrm{p}(\pi)) \otimes w_|(\pi) \otimes \rho(\mathrm{q}(\pi))$
  $= \sum_{q_n, q_m \in Q} \sum_{\pi \in \Pi(q_n, q_m)} \lambda(q_n) \otimes w_|(\pi) \otimes \rho(q_m)$
  $= \sum_{q_n, q_m \in Q} \lambda(q_n) \otimes \left(\sum_{\pi \in \Pi(q_n, q_m)} w_|(\pi)\right) \otimes \rho(q_m)$
  $= \sum_{q_n, q_m \in Q} \lambda(q_n) \otimes (\boldsymbol{W}^*)_{nm} \otimes \rho(q_m)$
  $= \boldsymbol{\lambda}^\top \boldsymbol{W}^* \boldsymbol{\rho}$ where $\boldsymbol{W}$ can be computed via Lehmann
- For instance: Log-likelihood: $\log p(y \mid x)$
- For a full training dataset: Log-likelihood: $\sum_{n=1}^{N} \log p(y^{(n)} \mid x^{(n)}) = \sum_{n=1}^{N} \left[\mathrm{score}(y^{(n)}, x^{(n)}) - \log Z(x^{(n)})\right]$
- This can be optimized with backpropagation (in $\mathcal{O}(|Q|^3)$) and gradient descent
- To train transliteration WFST, we use the log semiring
- To estimate the probability of the most likely transliteration y for a given input x, we use the arctic semiring

54 Other

Causal Models

Causal scenarios —
- Causal scenario without selection bias: $\mathcal{X}$ affects $\mathcal{Y}$ and there is no selection bias
  - Some features $\mathcal{X}_{\perp Y}$ do not causally affect $\mathcal{Y}$, but are affected by $\mathcal{W}$
  - Some features $\mathcal{X}_{\perp W}$ causally affect $\mathcal{Y}$, but are not affected by $\mathcal{W}$
  - Some features $\mathcal{X}_{W \& Y}$ causally affect $\mathcal{Y}$ and are affected by $\mathcal{W}$ as well as $\mathcal{X}_{\perp Y}$ and $\mathcal{X}_{\perp W}$
- Anti causal scenario: We assume $\mathcal{Y}$ affects $\mathcal{X}$, rather than the other way around
- Causal scenario with selection bias: $\mathcal{X}$ affects $\mathcal{Y}$ and there is a selection bias

---

Counterfactual invariance —
- Counterfactual invariance: Results of estimator remain consistent across different counterfactual scenarios, i.e. if $\mathcal{Y}$ is affected by $\mathcal{X}$, and $\mathcal{X}$ is affected by $\mathcal{W}$, but $\mathcal{W}$ does not affect $\mathcal{Y}$, our estimator should be invariant to states of $\mathcal{W}$, i.e. $f(\mathcal{X}(\mathcal{W}_1)) = f(\mathcal{X}(\mathcal{W}_2))$
- For counterfactual invariance, the following must hold:
  - Causal scenario without selection bias: $f(\mathcal{X}) \perp \mathcal{W}$, i.e. estimate f only depends on $\mathcal{X}_{\perp W}$
  - Anti causal scenario: $(f(\mathcal{X}) \perp \mathcal{W}) | \mathcal{Y}$, i.e. estimate f only depends on $\mathcal{X}_{\perp W}$, provided $\mathcal{Y}$ is known
  - Causal scenario with selection bias: $(f(\mathcal{X}) \perp \mathcal{W}) | \mathcal{Y}$ as long as $\mathcal{X}_{\perp Y}$ and $\mathcal{X}_{W \& Y}$ do not influence $\mathcal{Y}$ whatsoever, i.e. $(\mathcal{Y} \perp \mathcal{X}) | \mathcal{X}_{\perp W}, \mathcal{W}$
- For causal scenario without selection bias we need to show: $\mathcal{X}_{\perp W} \perp \mathcal{W}$
- For anti causal scenario we need to show: $(\mathcal{X}_{\perp W} \perp \mathcal{W}) | \mathcal{Y}$
- This can be shown via *d-separation*

D separation —
- Undirected path of n nodes is d-separated, if it contains 3 nodes following any of the following forms and if this form is blocked:
  - Chain structure: $X \to Z \to Y$ or $Y \to Z \to X$ – is blocked, if we condition on Z, i.e. Z is known
  - Fork structure: $X \leftarrow Z \to Y$ – is blocked, if we condition on Z, i.e. Z is known
  - Collider structure: $X \to Z \leftarrow Y$ – is blocked, if we don't condition on Z or any of its descendants
- Random variables X and Y are conditionally independent if each path between them is d-separated
  $\to$ as soon as we have one blocked triple on path, entire path is blocked
  $\to$ as soon as one path is active, we cannot guarantee conditional independence
- For causal scenario without selection bias we can show $\mathcal{X}_{\perp W} \perp \mathcal{W}$ since all paths are blocked
- For anti causal scenario we can show $(\mathcal{X}_{\perp W} \perp \mathcal{W}) | \mathcal{Y}$ since all paths are blocked, conditioned on $\mathcal{Y}$, i.e. if $\mathcal{Y}$ is observed

---

Proofs

Proofs —
- To prove $p \to q$:
  - Prove $p \to \neg q$ is impossible
  - Prove $\neg q \to \neg p$
- To prove $p \leftrightarrow q$: Prove $p \to q$ and $q \to p$
- To prove statement by induction:
  - Prove base case for $n = 0$ or $n = 1$
  - Assume inductive hypothesis: Assume statement holds for $n = k$
  - Prove inductive step: Prove statement holds for $n = k + 1$