Statistical Machine Learning Summary

Machine Learning Concepts: instance, attribute, label, model, supervised learning, unsupervised learning, evaluation metric: accuracy, contingency table, precision-recall, ROC curve.

Probability Theory:

- Basics:

- A probability space:
  - Set Ω of possible outcomes
  - Set *F* of events (subsets of outcomes)
  - Probability measure P: *F* → **R**
- Example: a die roll
  - {1, 2, 3, 4, 5, 6}
  - { φ, {1}, …, {6}, {1,2}, …, {5,6}, …, {1,2,3,4,5,6} }
  - P(φ)=0, P({1})=1/6, P({1,2})=1/3, …

- Axioms of probability:

1. F contains all of: Ω; all complements $\Omega \backslash f, f \in F$; the union of any countable set of events in F.
2. $P(f) \geq 0$ for every event $f \in F$.
3. $P\left(\bigcup_f f\right) = \sum_f P(f)$ for all countable sets of pairwise disjoint events.
4. $P(\Omega) = 1$

- Random Variables:

- A random variable *X* is a numeric function of outcome $X(\omega) \in \boldsymbol{R}$
- $P(X \in A)$ denotes the probability of the outcome being such that *X* falls in the range *A*
- Example: *X* winnings on $5 bet on even die roll
  - *X* maps 1,3,5 to -5
    X maps 2,4,6 to 5
  - P(*X*=5) = P(*X*=-5) = ½

- Discrete Distribution: The distribution of discrete random variables, described by probability mass function (PMF) and cumulative mass function (CMF). E.g. Bernoulli, Binomial, Multinomial, Possion.
- Continuous Distribution: The distribution governed by continuous random variables, described by probability density function (PDF) and cumulative density function (CDF). E.g. Uniform, Normal, Laplace, Gamma, Beta, Dirichlet.
- Expectation: The weighted average of the distribution.
  Discrete: $\mathbb{E}[X] = \sum_x x\, P(X = x)$
  Continuous: $\mathbb{E}[X] = \int_x x\, P(X = x)\, dx$
  Properties: $\mathbb{E}[aX + b] = a\mathbb{E}[X] + b$
  $\mathbb{E}[X + Y] = \mathbb{E}[X] + \mathbb{E}[Y]$
- Variance: The measurement of random variables spreading out from the average value.
  $$Var = \mathbb{E}[(X - \mathbb{E}[X])^2] = \mathbb{E}[X^2] - \mathbb{E}[X]^2$$
- Independence:

- *X*, *Y* are independent if
  - $P(X \in A, Y \in B) = P(X \in A)P(Y \in B)$
  - Similarly for densities: $p_{X,Y}(x, y) = p_X(x)p_Y(y)$
  - **Intuitively**: knowing value of *Y* reveals nothing about *X*
  - **Algebraically**: the joint on *X*,*Y* factorises!

- Conditioning:
  - Conditional probability
    - $P(A|B) = \frac{P(A \cap B)}{P(B)}$
    - Similarly for densities $p(y|x) = \frac{p(x,y)}{p(x)}$
    - **Intuitively**: probability event *A* will occur given we know event *B* has occurred
    - X,Y independent equiv to $P(Y = y|X = x) = P(Y = y)$

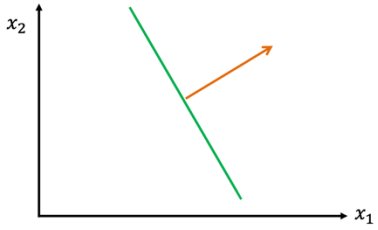
- Marginalization:
  - Marginals: probabilities of individual variables
  - Marginalisation: summing away all but r.v.'s of interest $P(A) = \sum_b P(A, B = b)$
- Bayes Theorem:
  - $P(A \cap B) = P(A|B)P(B) = P(B|A)P(A)$
  - $P(A|B) = \frac{P(B|A)\,P(A)}{P(B)}$

Linear Algebra:

- Dot Product:
- Given two m-dimensional vectors $\boldsymbol{u}$ and $\boldsymbol{v}$, their dot product is $\boldsymbol{u} \cdot \boldsymbol{v} \equiv \boldsymbol{u}'\boldsymbol{v} \equiv \sum_{i=1}^{m} u_i v_i$
  - E.g., weighted sum of terms is a dot product $\boldsymbol{x}'\boldsymbol{w}$
- If k is a scalar, $\boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}$ are vectors then
$$(k\boldsymbol{a})'\boldsymbol{b} = k(\boldsymbol{a}'\boldsymbol{b}) = \boldsymbol{a}'(k\boldsymbol{b})$$
$$\boldsymbol{a}'(\boldsymbol{b} + \boldsymbol{c}) = \boldsymbol{a}'\boldsymbol{b} + \boldsymbol{a}'\boldsymbol{c}$$
- Given two m-dimensional Euclidean vectors $\boldsymbol{u}$ and $\boldsymbol{v}$, their dot product is $\boldsymbol{u} \cdot \boldsymbol{v} \equiv \boldsymbol{u}'\boldsymbol{v} \equiv \|\boldsymbol{u}\|\|\boldsymbol{v}\| \cos\theta$

- Hyperplane and Normal Vector:

- A *hyperplane* defined by parameters $\boldsymbol{w}$ and b is a set of points $\boldsymbol{x}$ that satisfy $\boldsymbol{x}'\boldsymbol{w} + b = 0$
- In 2D, a hyperplane is a line: a line is a set of points that satisfy $w_1 x_1 + w_2 x_2 + b = 0$



A *normal vector* for a hyperplane is a vector perpendicular to that hyperplane

- Norms:

- Throughout the subject we will often encounter norms that are functions $\mathbb{R}^n \to \mathbb{R}$ of a particular form
  - Intuitively, norms measure lengths of vectors in some sense
  - Often component of objectives or stopping criteria in optimisation-for-ML
- More specifically, we will often use the L_2 norm (*aka* Euclidean distance)

$$\|\boldsymbol{a}\| = \|\boldsymbol{a}\|_2 \equiv \sqrt{a_1^2 + \cdots + a_n^2}$$

- And also the L_1 norm (*aka* absolute norm or Manhattan distance)

$$\|\boldsymbol{a}\|_1 \equiv |a_1| + \cdots + |a_n|$$

Linear Combination:

- A linear combination of vectors $v_1, \ldots, v_k \in V$ some vector space, is a new vector $\sum_{i=1}^{k} a_i v_i$ for some scalars $a_1, \ldots, a_k$
- A set of vectors $\{v_1, \ldots, v_k\} \subseteq V$ is called linearly dependent if one element v_j can be written as a linear combination of the other elements
- A set that isn't linearly dependent is linearly independent

- Span:

- The span of vectors $v_1, \ldots, v_k \in V$ is the set of all obtainable linear combinations (ranging over all scalar coefficients) of the vectors

- Basis:

- A set of vectors $\{v_1, \ldots, v_k\} \subseteq V$ is called a basis for a vector subspace $V' \subseteq V$ if
  1. The set is linearly independent; and
  2. Every v $\in V'$ is a linear combination of the set.
- An orthonormal basis is a basis in which each
  1. Pair of basis vectors are orthogonal (zero dot prod); and
  2. Basis vector has norm equal to 1.

- Matrix Operation:

- A rectangular array, often denoted by upper-case, with two indices first for row, second for column
- Square matrix has equal dimensions (numbers of rows and columns)
- Matrix transpose $\mathbf{A}'$ or $\mathbf{A}^\top$ of m by n matrix $\mathbf{A}$ is an n by m matrix with entries $A'_{ij}=A_{ji}$
- A square matrix $\mathbf{A}$ with $\mathbf{A}=\mathbf{A}'$ is called symmetric
- The (square) identity matrix $\mathbf{I}$ has 1 on the diagonal, 0 off-diagonal
- Matrix inverse $\mathbf{A}^{-1}$ of square matrix $\mathbf{A}$ (if it exists) satisfies $\mathbf{A}^{-1}\mathbf{A}=\mathbf{I}$

- Eigenvector and Eigenvalue

- Scalar, vector pair (λ, v) are called an eigenvalue-eigenvector pair of a square matrix $\mathbf{A}$ if $\mathbf{Av} = \lambda\mathbf{v}$
  - Intuition: matrix $\mathbf{A}$ doesn't rotate $\mathbf{v}$ it just stretches it
  - Intuition: the eigenvalue represents stretching factor

Properties: The eigenvalues of symmetric matrices are always real; A matrix with linear dependent columns have some zero eigenvalues.

- Positive (Semi)-definite Matrix:

- A symmetric square matrix $\mathbf{A}$ is called positive semidefinite if for all vectors $\mathbf{v}$ we have $\mathbf{v}'\mathbf{Av} \geq 0$.
  - Then $\mathbf{A}$ has non-negative eigenvalues
  - For example, any $\mathbf{A} = \mathbf{X}'\mathbf{X}$ since: $\mathbf{v}'\mathbf{X}'\mathbf{Xv} = \|\mathbf{Xv}\|^2 \geq 0$
- Further if $\mathbf{v}'\mathbf{Av} > 0$ holds as a strict inequality then $\mathbf{A}$ is called positive definite
  - Then $\mathbf{A}$ has (strictly) positive eigenvalues

Limits and Convergence:

- A sequence $\{x_i\}_{i\in\mathbb{N}}$ converges if its elements become and remain arbitrarily close to a fixed limit point L.
- Formally: $x_i \to L$ if, for all $\varepsilon > 0$, there exists $N \in \mathbb{N}$ such that for all $n \geq N$ we have $\|x_n - L\| < \varepsilon$

Supremum: The supremum is there exists an upper bound u, such that it is no bigger than any other upper bound set S, written as $\sup(S)$,

Infimum: The infimum is there exists an lower bound i, such that it is no bigger than any other upper bound set S, written as $\inf(S)$.

Frequentist Statistics: The idea is the unknown parameters are treated as having a fixed but unknown value. E.g Given n coin flips, determine the probability of landing heads.

Estimator Bias: $Bias_\theta(\hat{\theta}) = \mathbb{E}_\theta[\hat{\theta}(X_1, \dots, X_n)] - \theta$

Estimator Variance: $Var_\theta(\hat{\theta}) = \mathbb{E}_\theta\left[(\hat{\theta} - \mathbb{E}_\theta[\hat{\theta}])^2\right]$

Asymptotic Properties:

Asymptotic properties often hold! ☺

Bias closer and closer to zero

- Consistency: $\hat{\theta}(X_1, \dots, X_n) \to \theta$ in probability
- Asymptotic efficiency: $\mathrm{Var}_\theta\left(\hat{\theta}(X_1, \dots, X_n)\right)$ converges to the smallest possible variance of any estimator of θ

Variance closer & closer to optimal

Maximum Likelihood Estimation: Find a parameter θ that maximize data likelihood.

$$\hat{\theta}(x_1, x_2, \dots, x_n) = \underset{\theta \in \Theta}{\operatorname{arg}max} \prod_{i=1}^{n} p_\theta(x_i)$$

Statistical Decision Theory: Act to maximize utility by minimizing empirical risk $\hat{R}_\theta[\delta] = \frac{1}{n}\sum_{i=1}^{n} l(\delta(X_i), \theta)$ in order to approximate generalization error, $R_\theta = \mathbb{E}_{X \sim \theta}[l(\delta(X_i), \theta)]$.

Bayesian Statistics: The unknown model parameters are reflected by prior distribution and then updated by posterior distribution.

$$P(\theta|X) = \frac{P(X|\theta)P(\theta)}{P(X)} \propto P(X|\theta)P(\theta)$$

Maximum A Posteriori: To make point estimation of posterior distribution, i.e., maximize posterior distribution. $\underset{\theta}{\operatorname{arg}max}\, P(\theta|X) = \underset{\theta}{\operatorname{arg}max}\, P(X|\theta)P(\theta)$.

Parametric and Non-parametric Model:

Parametric	Non-Parametric
Determined by fixed, finite number of parameters	Number of parameters grows with data, potentially infinite
Limited flexibility	More flexible
Efficient statistically and computationally	Less efficient

Generative and Discriminative Models: Generative approach models full distribution $P(X, Y)$ while discriminative approach models conditional distribution $P(Y|X)$.

Linear Regression: Find a linear model to fit observed data X and target value Y.

- Loss Function: Mean squared error: $MSE = \frac{1}{n}\sum_{i=1}^{n}\left(\boldsymbol{w}^T x^{(i)} - y^{(i)}\right)^2$
- Analytic Solution: $\boldsymbol{w}^* = (X^T X)^{-1} X y$
- Probabilistic Model: $Y = \boldsymbol{w}^T X + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma^2)$. Therefore, the model can be:

  $$p_{\boldsymbol{w},\sigma^2}(y|\boldsymbol{x}) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y - \boldsymbol{w}^T\boldsymbol{x})^2}{2\sigma^2}\right)$$

  Thus, we can use maximum likelihood estimate to find the best parameter $\boldsymbol{w}$.
- Basis Expansion: Expand raw data dimension in order to fit non-linear model, i.e. $\boldsymbol{x} \to \varphi(\boldsymbol{x})$, where $\boldsymbol{x} \in \mathbb{R}^m$ and $\varphi(\boldsymbol{x}) \in \mathbb{R}^k$.
- Polynomial Function:
  $y = w_0 + w_1\varphi_1(x) + w_2\varphi_2(x) = w_0 + w_1 x + w_2 x^2$
- Radial Basis Function:
  - $\varphi(\boldsymbol{x}) = \|\boldsymbol{x} - \boldsymbol{z}\|$
  - $\varphi(\boldsymbol{x}) = \exp\left(-\frac{1}{\sigma}\|\boldsymbol{x} - \boldsymbol{z}\|^2\right)$

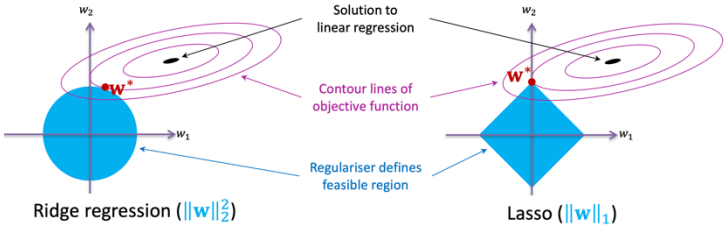
Regularization: Employ additional term to prevent overfitting problem.

- Irrelevant Features: Adding a Δ to one feature and subtracting another Δ will not affect predictions and errors, which means the features are co-linear, i.e., matrix $X^T X$ has no inverse.
- Ridge Regression:

The mean squared error is:

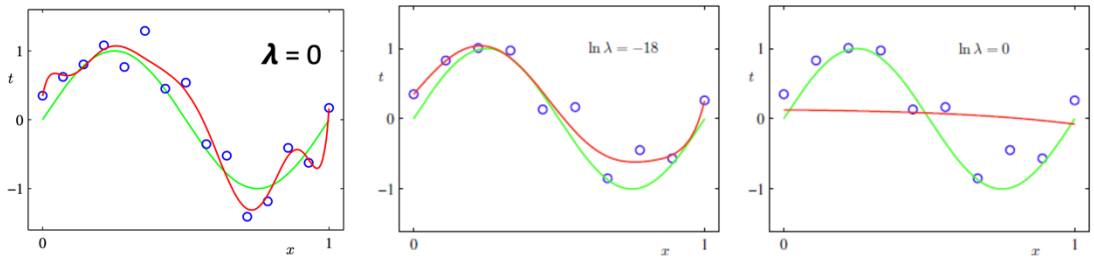
$$MSE = \frac{1}{n}\sum_{i=1}^{n}\left(\boldsymbol{w}^T x^{(i)} - y^{(i)}\right)^2 + \lambda\left|\left|\boldsymbol{w}\right|\right|^2$$

- Analytic Solution: $\boldsymbol{w}^* = (X^T X + \lambda I)^{-1} X y$
- Regulariser as Constraint:
  - For illustrative purposes, consider a *modified problem*:
    minimise $\|\mathbf{y} - \mathbf{X}\mathbf{w}\|_2^2$ *subject to* $\|\mathbf{w}\|_2^2 \leq \lambda$ for $\lambda > 0$



where L_1 has no analytic solution.

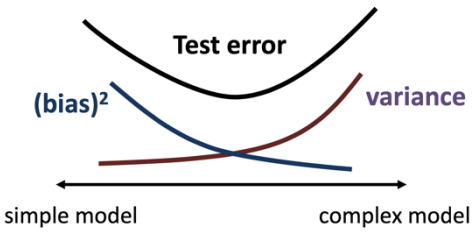
- Non-linear Model: Regularization term λ will affect the model performance. When λ is small, the model is prone to overfit, while too large λ will underfit the data.



- Bias-Variance Tradeoff

$$\mathbb{E}\left[l\left(Y, \hat{f}(\boldsymbol{X}_0)\right)\right] = \left(\mathbb{E}[Y] - \mathbb{E}[\hat{f}]\right)^2 + Var[\hat{f}] + Var[Y]$$

  - simple model ➔ high bias, low variance
  - complex model ➔ low bias, high variance



Logistic Regression: To solve linear classification problem, we use linear combination of the parameters and instance followed by sigmoid function $\sigma = \frac{1}{1+\exp(-x)}$. The decision boundary is typically when σ is larger than or smaller than 0.5.

- Probabilistic Model:
  - Logistic regression assumes a Bernoulli distribution with parameter given by logistic transform of linear model
    $p(y|\boldsymbol{x}) = Bernoulli(\text{logistic}(\boldsymbol{x}'\boldsymbol{w}))$

$$\log P(Y|X) = \log \prod_{i=1}^{n} p(y_i|x_i)$$

$$= \log \prod_{i=1}^{n} (\sigma(\boldsymbol{w^T x_i}))^{y_i} (\sigma(1 - \boldsymbol{w^T x_i}))^{1-y_i}$$

$$= \sum_{i=1}^{n} \log(\sigma(\boldsymbol{w^T x_i}))^{y_i} (\sigma(1 - \boldsymbol{w^T x_i}))^{1-y_i}$$

$$= \sum_{i=1}^{n} \log y_i (\sigma(\boldsymbol{w^T x_i})) + (1 - y_i) \log(\sigma(1 - \boldsymbol{w^T x_i}))$$

$$\nabla_{\boldsymbol{w}} \log P(Y|X) = \sum_{i=1}^{n} \big(y_i - \sigma(\boldsymbol{w^T x_i})\big) \boldsymbol{x_i}$$

- Cross Entropy:

$$CE = -y_i \log \hat{y}_i$$

Optimization:

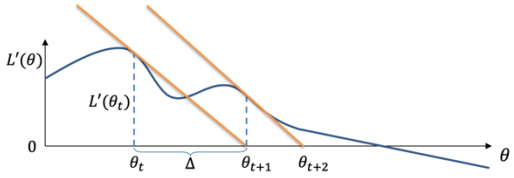
- Gradient Descent:

1. Choose $\boldsymbol{\theta}^{(0)}$ and some T
2. For i from 1 to T*
   1. $\boldsymbol{\theta}^{(i)} = \boldsymbol{\theta}^{(i-1)} - \eta \nabla L(\boldsymbol{\theta}^{(i-1)})$
3. Return $\widehat{\boldsymbol{\theta}} \approx \boldsymbol{\theta}^{(T)}$

- Note: η dynamically updated per step
- Variants: Momentum, AdaGrad, …
- Stochastic gradient descent: two loops
  - Outer for loop: each loop (called epoch) sweeps through all training data
  - Within each epoch, randomly shuffle training data; then for loop: do gradient steps only on batches of data. Batch size might be 1 or few

- Convex Objective Function:
  Formally* $f: D \to \mathbf{R}$ is convex
  if $\forall \boldsymbol{a}, \boldsymbol{b} \in D, t \in [0,1]$:
  $f(t\boldsymbol{a} + (1-t)\boldsymbol{b}) \leq tf(\boldsymbol{a}) + (1-t)f(\boldsymbol{b})$
  Strictly convex if inequality is strict (<)

- Newton-Raphson Method:



- Critical points of $L(\theta) =$ Zero-crossings of $L'(\theta)$
- Consider case of scalar θ. Starting at given/random θ_0, iteratively:
  1. Fit tangent line to $L'(\theta)$ at θ_t
  2. Need to find $\theta_{t+1} = \theta_t + \Delta$ using linear approximation's zero crossing
  3. Tangent line given by derivative: rise/run = $-L''(\theta_t) = L'(\theta_t)/\Delta$
  4. Therefore iterate is $\theta_{t+1} = \theta_t - L'(\theta_t)/L''(\theta_t)$

PAC Learning Theory:

- Startup: We consider the binary classification problem, i.e., $\mathcal{Y} \in \{-1, 1\}$; Data is independently identically distributed, i.e., $(x_i, y_i)_{i=1}^{m} \sim \mathcal{D}$; We consider a class of classifiers $\mathcal{F}$, which maps from

$\mathcal{X}$ into $\mathcal{Y}$.

- Risk:

$$\underbrace{R[f_m] - R^*}_{\text{Excess risk}} = \underbrace{(R[f_m] - R[f^*])}_{\text{Estimation error}} + \underbrace{(R[f^*] - R^*)}_{\text{Approximation error}}$$

- **Good**: what we'd aim for in function class, with infinite data
  - $R[f^*]$ true risk of best in class $f^* \in \operatorname{argmin}_{f\in\mathcal{F}} R[f]$
- **Bad**: we get what we get and don't get upset
  - $R[f_m]$ true risk of learned $f_m \in \arg\min_{f\in\mathcal{F}} \hat{R}[f] + C\|f\|^2$ (e.g.)
- **Ugly**: we usually cannot even hope for perfection!
  - $R^* \in \inf_f R[f]$ called the Bayes risk; noisy labels makes large
- Bayes risk $R^* \in \inf_f R[f]$
  - Best risk possible, ever; but can be large
  - Depends on distribution and loss function
- Bayes classifier achieves Bayes risk
  - $f_{Bayes}(x) = \operatorname{sgn} \mathbb{E}(Y|X = x)$

- The relation between true risk and empirical risk: The probability of true risk $R[f]$ within ϵ of empirical risk $\hat{R}[f]$ will be high.

$R[f] \le \hat{R}[f] + \sqrt{\frac{\log(1/\delta)}{2m}}$ with high probability (w.h.p.) $\ge 1 - \delta$

- Uniform Derivation Bound: The difference between true risk and empirical risk for any classifier f is smaller than or equal to the supremum of the difference for all classifiers:

$$R[f_m] - \hat{R}[f_m] \le \sup_{f\in\mathcal{F}} (R[f] - \hat{R}[f])$$

- Relation between best empirical risk and best true risk is:

$$\begin{aligned} R[f_m] &\le \left(\hat{R}[f^*] - \hat{R}[f_m]\right) + R[f_m] - R[f^*] + R[f^*] \\ &= \hat{R}[f^*] - R[f^*] + R[f_m] - \hat{R}[f_m] + R[f^*] \\ &\le \left|R[f^*] - \hat{R}[f^*]\right| + \left|R[f_m] - \hat{R}[f_m]\right| + R[f^*] \\ &\le 2\sup_{f\in\mathcal{F}}\left|R[f] - \hat{R}[f]\right| + R[f^*] \end{aligned}$$

- Union Bound: The probability of any one of events happens is at most the sum of probabilities of all events happen.

$$\Pr\left(\bigcup_i A_i\right) \le \sum_i \Pr(A_i)$$

- Error Bound for Finite Classes $\mathcal{F}$.

Theorem: Consider any $\delta > 0$ and finite class $\mathcal{F}$. Then w.h.p at least $1 - \delta$: For all $f \in \mathcal{F}$, $R[f] \le \hat{R}[f] + \sqrt{\frac{\log|\mathcal{F}| + \log(1/\delta)}{2m}}$

- Dichotomy:
- Definition: Given sample $x_1, \dots, x_m$ and family $\mathcal{F}$, a **dichotomy** is a $(f(x_1), \dots, f(x_m)) \in \{-1, +1\}^m$ for some $f \in \mathcal{F}$.

Properties:

Even when $\mathcal{F}$ infinite, $|\mathcal{F}(\mathbf{x})| \leq 2^m$ (why?)

And also (relevant for $\mathcal{F}$ finite, tiny), $|\mathcal{F}(\mathbf{x})| \leq |\mathcal{F}|$ (why?)

- Growth Function:
  - Definition: The **growth function** $S_{\mathcal{F}}(m) = \sup_{\mathbf{x}\in\mathcal{D}^m}|\mathcal{F}(\mathbf{x})|$ is the max number of label patterns achievable by $\mathcal{F}$ for any m sample.

e.g. $S_{\mathcal{F}}(3) = 2^3 = 8$ and $S_{\mathcal{F}}(4) = 14 = 2^4$.

- Error Bound with Growth Function:
  - Theorem: Consider any $\delta > 0$ and any class $\mathcal{F}$. Then w.h.p. at least $1-\delta$: For all $f \in \mathcal{F}$

$$R[f] \leq \hat{R}[f] + 2\sqrt{2\frac{\log S_{\mathcal{F}}(2m) + \log(4/\delta)}{m}}$$

  - A few negligible extra constants (the 2s, the 4)
  - $|\mathcal{F}|$ has become $S_{\mathcal{F}}(2m)$
  - $S_{\mathcal{F}}(m) \leq |\mathcal{F}|$, not "worse" than union bound for finite $\mathcal{F}$
  - $S_{\mathcal{F}}(m) \leq 2^m$, very bad for big family with exponential growth function gets $R[f] \leq \hat{R}[f] + Big\ Constant$. Even $R[f] \leq \hat{R}[f] + 1$ meaningless!!

- VC-dimension:
  - Definition: The VC dimension $\mathrm{VC}(\mathcal{F})$ of a family $\mathcal{F}$ is the largest m such that $S_{\mathcal{F}}(m) = 2^m$.
  - Points $\mathbf{x} = (x_1, \ldots, x_m)$ are shattered by $\mathcal{F}$ if $|\mathcal{F}(\mathbf{x})| = 2^m$
  - So $\mathrm{VC}(\mathcal{F})$ is the size of the largest set shattered by $\mathcal{F}$

- Sauer-Shelah Lemma:
  - Lemma (Sauer-Shelah): Consider any $\mathcal{F}$ with finite $\mathrm{VC}(\mathcal{F}) = k$, any sample size m. Then $S_{\mathcal{F}}(m) \leq \sum_{i=0}^{k}\binom{m}{i}$.
    - From basic facts of Binomial coefficients
      - Bound is $O(m^k)$: finite VC ⇒ eventually polynomial growth!
      - For $m \geq k$, it is bounded by $\left(\frac{em}{k}\right)^k$
  - Theorem (VC bound): Consider any $\delta > 0$ and any VC-k class $\mathcal{F}$. Then w.h.p. at least $1-\delta$: For all $f \in \mathcal{F}$

$$R[f] \leq \hat{R}[f] + 2\sqrt{2\frac{k\log\frac{2em}{k} + \log\frac{4}{\delta}}{m}}$$

- Error Bound for Infinite Classes $\mathcal{F}$.

Perceptron: A linear model with step activation function.

- Model Function:

$$\hat{y} = \begin{cases} 1 & \text{if } \sum_{i=0}^{m} w_i x_i \geq 0 \\ -1 & \text{if } \sum_{i=0}^{m} w_i x_i < 0 \end{cases}$$

i.e. $\hat{y} = sgn(\sum_{i=0}^{m} w_i x_i)$

where w_0 is the intercept term.

- Loss Function: if $\hat{y}$ and y have the same sign, the loss is evaluated to 0, while if they do not have the same sign, the loss is evaluated to $|\hat{y}|$, which can be rewritten as:

$$L(\hat{y}, y) = \max(0, y\hat{y})$$

- Gradient of Loss Function:

$$\nabla_w L(\hat{y}, y) = \begin{cases} 0 & \text{if } y\hat{y} > 0 \\ -x_i & \text{if } y = 1 \text{ and } \hat{y} < 0 \\ x_i & \text{if } y = -1 \text{ and } \hat{y} > 0 \end{cases}$$

- SGD:

- Randomly shuffle/split all training examples in B batches
- Choose initial $\boldsymbol{\theta}^{(1)}$
- For i from 1 to T

Iterations over the entire dataset are called *epochs*

- For j from 1 to B
- Do gradient descent update using data from batch j

When classified correctly, weights are unchanged

When misclassified: $\boldsymbol{w}^{(k+1)} = -\eta(\pm\boldsymbol{x})$
($\eta > 0$ is called *learning rate*)

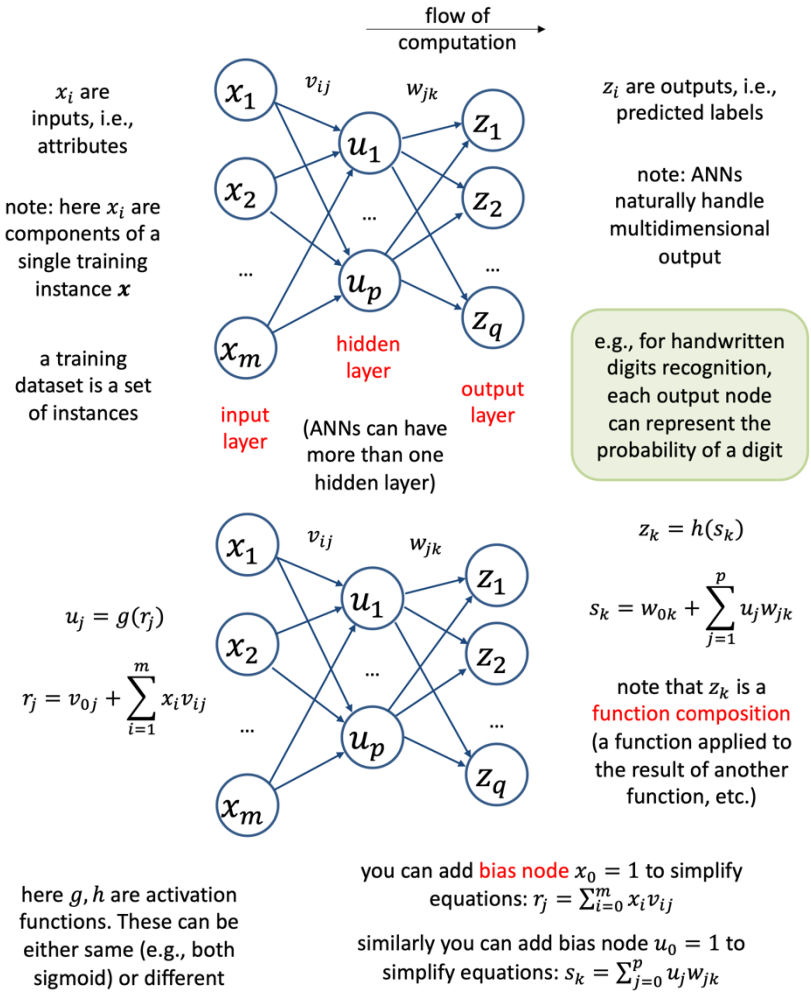
If $y = 1$, but $s < 0$	If $y = -1$, but $s \geq 0$
$w_i \leftarrow w_i + \eta x_i$	$w_i \leftarrow w_i - \eta x_i$
$w_0 \leftarrow w_0 + \eta$	$w_0 \leftarrow w_0 - \eta$

- Pros and Cons

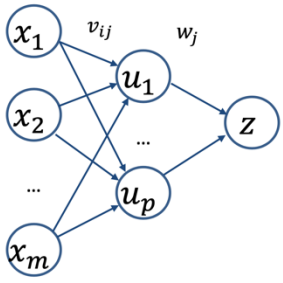
- If the data is linearly separable, the perceptron training algorithm will converge to a correct solution
  - There is a formal proof ← good!
  - It will converge to some solution (separating boundary), one of infinitely many possible ← bad!
- However, if the data is not linearly separable, the training will fail completely rather than give some approximate solution
  - Ugly ☹

Multilayer Perceptron:

- Feed-forward Neural Network



- Universal Approximator:
  - *Universal approximation theorem (Cybenko 1989)*: An ANN with a hidden layer with a finite number of units, and mild assumptions on the activation function, can approximate continuous functions on compact subsets of $\boldsymbol{R}^n$ arbitrarily well
- Depth and Width: Depth is the number of hidden layers, and width is the number of hidden units.
- Backpropagation: Use chain rule to update parameters for each neuron and for each hidden layer.
  - Backpropagation makes use of this fact by applying the chain rule for derivatives
  - $\frac{\partial L}{\partial w_j} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial s}\frac{\partial s}{\partial w_j}$
  - $\frac{\partial L}{\partial v_{ij}} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial s}\frac{\partial s}{\partial u_j}\frac{\partial u_j}{\partial r_j}\frac{\partial r_j}{\partial v_{ij}}$



Replace some of terms as special token, we have:

- Now define

$$\delta \equiv \frac{\partial L}{\partial s} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial s}$$

$$\varepsilon_j \equiv \frac{\partial L}{\partial r_j} = \frac{\partial L}{\partial z}\frac{\partial z}{\partial s}\frac{\partial s}{\partial u_j}\frac{\partial u_j}{\partial r_j}$$

- Here $L = 0.5(z-y)^2$ and $z = s$

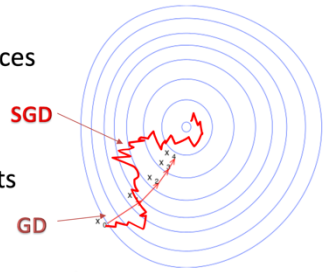
  Thus $\boxed{\delta = (z-y)}$

- Here $s = \sum_{j=0}^{p} u_j w_j$ and $u_j = g(r_j)$

  Thus $\boxed{\varepsilon_j = \delta w_j g'(r_j)}$

- Batch Gradient Descent, Stochastic Gradient Descent, and Mini-batch Gradient Descent:

- SGD works on single instances
  - high variance in gradients
  - many, quick, updates
- GD works on whole datasets
  - stable update, but slow
  - computationally expensive



- Compromise: mini-batch *(often just called "SGD")*
  - process batches of size $1 < b < N$, e.g., $b = 100$
  - balances computation and stability

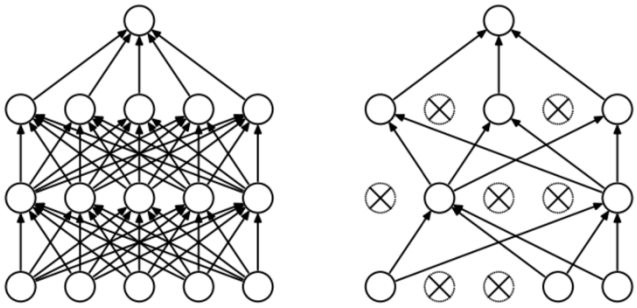
Gradient descent algorithm is guaranteed to find the global optima for convex function, but only local optima for non-convex function. Learning rate is crucial to the convergence of optimizer. Too large learning rate will fluctuate around the optima, even diverge. Too small learning rate will be more likely to be stuck into the local optima.

- SGD Momentum: Buffer the previous gradients as the velocity that are used for the next update.
  - $\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \boldsymbol{v}^{(t)}$
  - $\boldsymbol{v}^{(t)} = \alpha \boldsymbol{v}^{(t-1)} + \eta \Delta L(\boldsymbol{\theta}^{(t)})$
  - α decays the velocity, e.g., 0.9
- Adagrad: Provide bigger step for less frequent parameters and smaller step for more frequent parameters.
  - $g_i^{(t)} = g_i^{(t-1)} + \Delta L(\boldsymbol{\theta}^{(t)})_i^2$
  - $\theta_i^{(t+1)} = \theta_i^{(t)} - \frac{\eta}{\sqrt{g_i^{(t)} + \epsilon}} \Delta L(\boldsymbol{\theta}^{(t)})_i$

  Typically $\epsilon = 10^{-8}$, $\eta = 0.01$
- Adam: Combine momentum with adaptive learning rate.
  - $\boldsymbol{m}^{(t)} = \beta_1 \boldsymbol{m}^{(t-1)} + (1-\beta_1)\Delta L(\boldsymbol{\theta}^{(t)})$
  - $\boldsymbol{v}^{(t)} = \beta_2 \boldsymbol{v}^{(t-1)} + (1-\beta_2)\Delta L(\boldsymbol{\theta}^{(t)})^2$
  - $\theta^{(t+1)} = \boldsymbol{\theta}^{(t)} - \frac{\eta}{\sqrt{\boldsymbol{v}^{(t)}/_{1-\beta_2}} + \epsilon} \boldsymbol{m}^{(t)}/_{1-\beta_1}$
  - $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$

element-wise operations

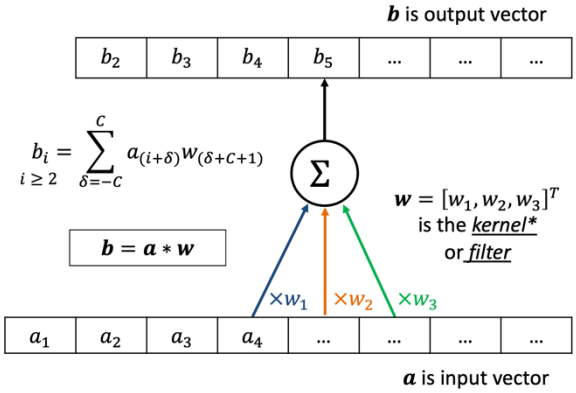
- Dropout: Mask some of neurons with probability of p during training to reduce the connectivity of the neural network, and no masking during evaluation.



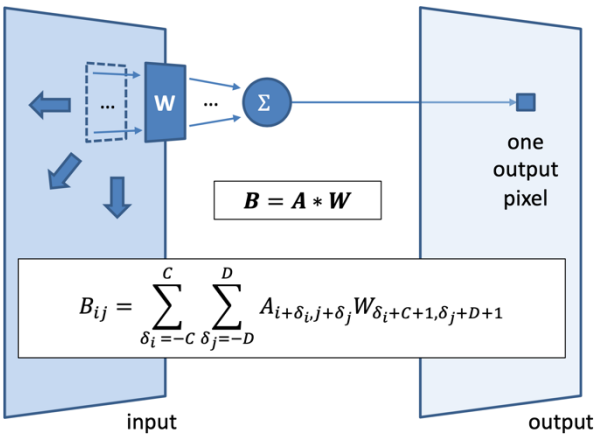
(a) Standard Neural Net (b) After applying dropout.

CNN

- Why Fully-connected network fails to image classification: Too large number of parameters, no spatial invariance, and no parameter sharing.
- Convolutional 1D:



- Convolutional 2D:



Relations between input feature maps and output feature maps: Suppose the input image is (H, W, C), kernel size is (h, w, C), padding is p, stride is s and the number of filters is f. The output feature map will be:

$$H' = \left\lfloor \frac{H - h + 2p}{s} \right\rfloor + 1$$

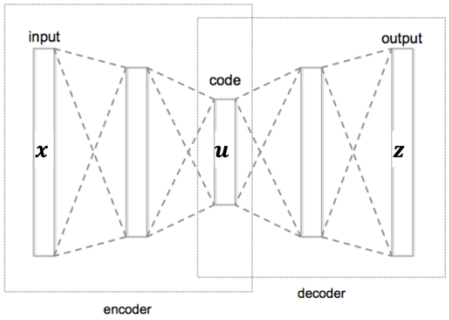
$$W' = \left\lfloor \frac{W - w + 2p}{s} \right\rfloor + 1$$

$$C' = f$$

- Pooling: Downsampling via Max-pooling or Average-pooling for rescaling to smaller resolution. The gradient of max-pooling is 1 for the maximum element in a patch and 0 for other elements.
- Residual Network: Use highway connection to copy input to output for better optimization (i.e. solving vanishing gradient problem).

Autoencoder: For unsupervised learning, initialization, and dimensionality reduction.

- Topology:



- Encoder-Decoder: Use encoder-decoder structure to output $D(E(\boldsymbol{x}))$ to approximate $\boldsymbol{x}$.
- Undercomplete and Overcomplete:

* undercomplete: model with thinner bottleneck than input forced to generalise
* overcomplete: wider bottleneck than input, can just "copy" input directly to output

- Methods for Learning Autoencoder

• Sparse autoencoders
  * includes regularisation over the code layer
• Denoising autoencoders
  * learn to recover true data given noise-corrupted input
• Contractive autoencoders
  * explicit penalty term to ensure robustness to local changes in input

* Use regularisation penalty in training objective, e.g.,

$$\lambda \sum_i |u_i|$$

* May need to use with over-complete hidden layer, and multi-layer encoder/decoder
* Sparsity can simplify data analysis
* Now modelling $\mathrm{p}(\boldsymbol{x}|\tilde{\boldsymbol{x}})$ where $\tilde{\boldsymbol{x}}$ is noised input
* The model must recover corrupted parts of the input, thus must learn underlying structure of the data

Include penalty term over derivatives wrt input

$$\lambda \sum_i ||\nabla_{\boldsymbol{x}} u_i||^2$$

* local changes to input $\boldsymbol{x}$ have limited effect on code $\boldsymbol{u}$
* tangent vectors to the manifold encode data variation

- Use of Autoencoder

- Data visualisation & clustering
  - Unsupervised first step towards understanding properties of the data
- As a feature representation
  - Allowing the use of off-the-shelf ML methods, applied to much smaller and informative representations of input
- Pre-training of deep models
  - Warm-starting training by initialising model weights with encoder parameters
  - In some fields like vision, mostly replaced with supervised pre-training on very large datasets

Variational Autoencoder: Model data with latent variables $\boldsymbol{z}$.

- $p(\boldsymbol{x}) = \sum_{\boldsymbol{z}} p(\boldsymbol{x}, \boldsymbol{z})$ (*sum denotes integral or summation*)
- latent variable to capture structure in data, such as clusters, style, etc
- framework includes several models, e.g., RBMs, DBMs etc

- Derivation:

- $\log p(\boldsymbol{x})$

$$= \log \sum_{\boldsymbol{z}} p(\boldsymbol{x}|\boldsymbol{z}) p(\boldsymbol{z})$$

assume model factorises

$$= \log \sum_{\boldsymbol{z}} p(\boldsymbol{x}|\boldsymbol{z}) p(\boldsymbol{z}) \frac{q(\boldsymbol{z}|\boldsymbol{x})}{q(\boldsymbol{z}|\boldsymbol{x})}$$

introduce q distribution (chosen carefully)

$$= \log E_{\boldsymbol{z}\sim q(\boldsymbol{z}|\boldsymbol{x})} \left[p(\boldsymbol{x}|\boldsymbol{z}) p(\boldsymbol{z}) \frac{1}{q(\boldsymbol{z}|\boldsymbol{x})} \right]$$

$$\geq E_{\boldsymbol{z}\sim q(\boldsymbol{z}|\boldsymbol{x})} \left[\log p(\boldsymbol{x}|\boldsymbol{z}) \frac{p(\boldsymbol{z})}{q(\boldsymbol{z}|\boldsymbol{x})} \right]$$

use Jensen's inequality to move log inside expectation

- prior $p(\boldsymbol{z})$ a unit Gaussian, $N(0,1)$
- $q(\boldsymbol{z}|\boldsymbol{x})$ also a Gaussian, $N(\mu(\boldsymbol{x}), \sigma^2(\boldsymbol{x}))$ as neural network with vector outputs, and parameters ϕ
- likelihood $p(\boldsymbol{x}|\boldsymbol{z})$ another neural network, parameters θ

- Objective Function:

$$\arg\max_{\theta,\phi} E_{\boldsymbol{z}\sim q(\boldsymbol{z}|\boldsymbol{x})} \left[\log p(\boldsymbol{x}|\boldsymbol{z}) + \log \frac{p(\boldsymbol{z})}{q(\boldsymbol{z}|\boldsymbol{x})} \right]$$

reconstruction term

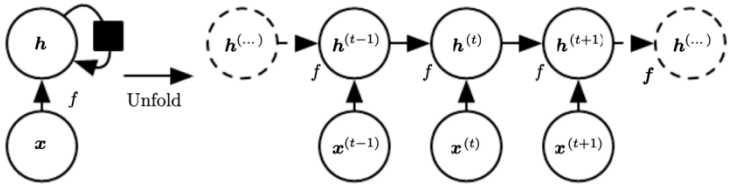
Kullbach-Leibler divergence $-D_{\mathrm{KL}}(q(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}))$

- Generation: First sample from prior distribution $p(\boldsymbol{z})$ and then sample from decoder, i.e. $\boldsymbol{x}\sim p(\boldsymbol{x}|\boldsymbol{z})$.

PCA: For dataset with m dimensions, we find the top k vectors, such that these vectors coordinate concentrate most of the data variance. PCA can be used for dimensionality reduction.

RNN: Recurrent neural network for sequence data, such as text, audio clip, stocks, etc.

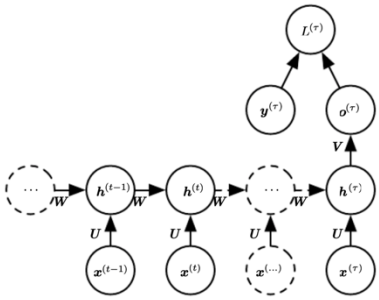
* given sequence of inputs $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(t)}$
* process each symbol from left to right, to form a sequence of hidden states $\boldsymbol{h}^{(t)}$
* each $\boldsymbol{h}^{(t)}$ encodes all inputs up to t



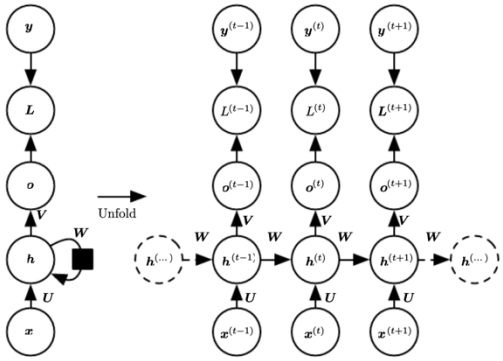
The shape of input matrix must be the same; parameters are shared across all sequences.

- RNN Tasks:

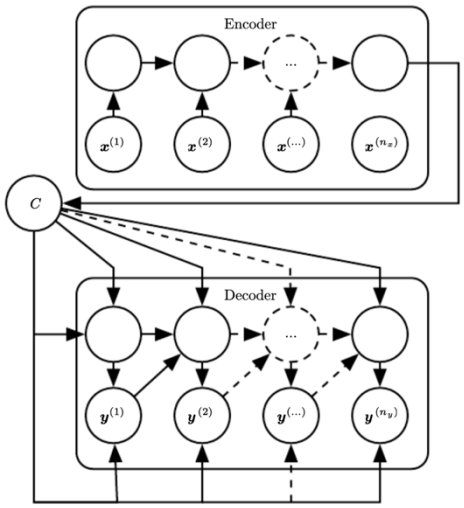
  (i) Sentiment Classification:



  (ii) POS Tagging:



  (iii) Machine Traslation (Seq-Seq Model):



- Parameterization:

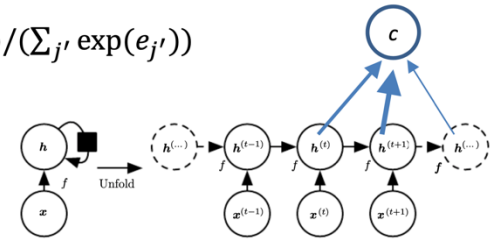
$$\begin{aligned} \boldsymbol{a}^{(t)} &= \boldsymbol{b} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{U}\boldsymbol{x}^{(t)}, \\ \boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{a}^{(t)}), \\ \boldsymbol{o}^{(t)} &= \boldsymbol{c} + \boldsymbol{V}\boldsymbol{h}^{(t)}, \\ \hat{\boldsymbol{y}}^{(t)} &= \operatorname{softmax}(\boldsymbol{o}^{(t)}), \end{aligned}$$

- Backpropagation Though Time: Gradients from the loss at every position must be propagated back to the very start of the network.
- Vanishing Gradient:

* Consider linear RNN, gradients of $\frac{\partial \boldsymbol{h}^{(T)}}{\partial \boldsymbol{h}^{(1)}} = \boldsymbol{W}^{T-1}$, thus can explode or vanish with large *T*, depending on largest eigenvalue of **W** (i.e., greater than cf less than one).
* Can't learn long distance phenomena (over 10+ steps)

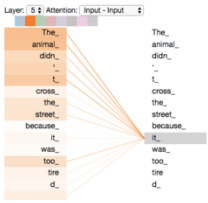
- GRU and LSTM: Use gate control to solve gradient vanishing problem.
- Attention Mechanism: The weighted average of all hidden states.

* $\boldsymbol{c} = \sum_j \alpha_j \boldsymbol{h}^{(j)}$
* $\alpha_j = \exp(e_j)/(\sum_{j'} \exp(e_{j'}))$
* $e_j = f(\boldsymbol{h}^{(j)})$



- Self-attention:

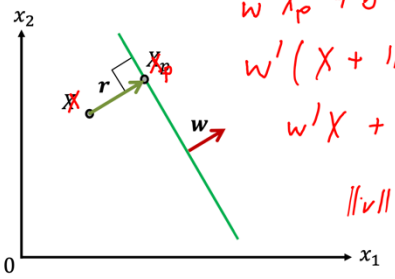
- Transformers use attention as means of representing sequences directly, instead of RNN
  * Representation of item *i* is based on attention to the rest of the sequence
  * Use item *i* as the query in attention against all items $j \neq i$
- Compared to RNNs
  * No explicit position information (add to each symbol position index)
  * Cheap: easily done in parallel



SVM: The objective is to find a hyperplane that maximizes the margins between two separated classes. The points on the margin is called support vectors. Margin is the distance between hyperplane and the support vector.

- Distance from Point to Hyperplane:

- Distance is $\|\boldsymbol{r}\| = -\frac{\boldsymbol{w}'\boldsymbol{x}+b}{\|\boldsymbol{w}\|}$, or more generally $\|\boldsymbol{r}\| = \pm\frac{\boldsymbol{w}'\boldsymbol{x}+b}{\|\boldsymbol{w}\|}$



$$x_p = x + \|r\| \frac{w}{\|w\|}$$

$$w'x_p + b = 0$$

$$w'\left(x + \|r\| \frac{w}{\|w\|}\right) + b = 0$$

$$w'x + \|r\| \frac{w'w}{\|w\|} + b = 0$$

$$\|r\| = -\frac{(w'x + b)}{\|w\|}$$

Since the training data is labeled as -1 and 1, we reformulate the distance as:

$$\|\boldsymbol{r}_i\| = \frac{y_i(\boldsymbol{w}'\boldsymbol{x}_i + b)}{\|\boldsymbol{w}\|}$$

- Hard-margin SVM: The objective function of SVM is to maximize $\underset{i=1,\dots,n}{min} \frac{y_i(\boldsymbol{w}^T x_i + b)}{\|\boldsymbol{w}\|}$. If we introduce an extra requirement $\frac{y_{i^*}(\boldsymbol{w}^T x_{i^*} + b)}{\|\boldsymbol{w}\|} = \frac{1}{\|\boldsymbol{w}\|}$. Thus, our objective function becomes:

$$\arg\min_{\boldsymbol{w}} \|\boldsymbol{w}\| \quad \text{s.t. } y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \geq 1 \text{ for } i = 1,2,\dots,n$$

- Hard-margin SVM Loss:

$$L_\infty = \begin{cases} 0 & \text{if } 1 - y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \leq 0 \\ \infty & \text{if } 1 - y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) > 0 \end{cases}$$

which means if the data point is $\frac{1}{\|\boldsymbol{w}\|}$ away from the hyperplane, the loss is 0. Otherwise, we penalize the instance with infinity loss.

- Soft-margin SVM: We allow points to be inside the margin or even on the wrong side of the margin. Thus, our loss function (hinge loss) reformulates as:

$$L_h = \begin{cases} 0 & \text{if } 1 - y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \leq 0 \\ 1 - y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) & \text{otherwise} \end{cases}$$

$$\Rightarrow \quad L_h = \max(0,\ 1 - y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b))$$

We can also define a slack variable $\xi_i \geq l_h$, such that:

$$\arg\min_{\boldsymbol{w},\xi} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n}\xi_i \quad \text{s.t. } \xi_i \geq 1 - y_i(\boldsymbol{w}^T\boldsymbol{x}_i + b) \text{ for } i = 1,2,\dots,n;\ \xi_i \geq 0 \text{ for } i = 1,2,\dots,n$$

- Duality: Instead of solving primal problem, solve dual problem that is the reformulation of the primal problem.
- Constrained Optimization:

Constrained optimisation: canonical form

$$\text{minimise } f(\boldsymbol{x})$$

$$\text{s.t. } g_i(\boldsymbol{x}) \leq 0,\ i = 1,\dots,n$$

$$h_j(\boldsymbol{x}) = 0,\ j = 1,\dots,m$$

- Lagrange Multipliers: Transform constrained optimization problem to unconstrained one, or primal

to dual.

Introduce auxiliary objective function via auxiliary variables

$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\nu}) = f(\boldsymbol{x}) + \sum_{i=1}^{n} \lambda_i g_i(\boldsymbol{x}) + \sum_{j=1}^{m} \nu_j h_j(\boldsymbol{x})$ ← Primal constraints became penalties

* Called the *Lagrangian* function
* New $\boldsymbol{\lambda}$ and $\boldsymbol{\nu}$ are called the *Lagrange multipliers* or *dual variables*

(Old) primal program: $\min_{\boldsymbol{x}} \max_{\boldsymbol{\lambda} \geq \mathbf{0}, \boldsymbol{\nu}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\nu})$

(New) dual program: $\max_{\boldsymbol{\lambda} \geq \mathbf{0}, \boldsymbol{\nu}} \min_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\nu})$ ← May be easier to solve, advantageous

- KKT Conditions:

Primal feasibility:
* $g_i(\boldsymbol{x}^*) \leq 0, i = 1, \dots, n$
* $h_j(\boldsymbol{x}^*) = 0, j = 1, \dots, m$

Souped-up version of necessary condition "derivative is zero" in **unconstrained** optimisation.

Dual feasibility: $\lambda_i^* \geq 0$ for $i = 1, \dots, n$

Complementary slackness: $\lambda_i^* g_i(\boldsymbol{x}^*) = 0, i = 1, \dots, n$ ← Don't penalise if constraint satisfied

Stationarity: $\nabla_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}^*, \boldsymbol{\lambda}^*, \boldsymbol{\nu}^*) = \mathbf{0}$

- KKT for Hard-margin:

The Lagrangian

$$\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\lambda}) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{i=1}^{n} \lambda_i (y_i(\boldsymbol{w}'\boldsymbol{x}_i + b) - 1)$$

KKT conditions:

* Feasibility: $y_i\left((\boldsymbol{w}^*)'\boldsymbol{x}_i + b^*\right) - 1 \geq 0$ for $i = 1, \dots, n$
* Feasibility: $\lambda_i^* \geq 0$ for $i = 1, \dots, n$
* Complementary slackness: $\lambda_i^*\left(y_i\left((\boldsymbol{w}^*)'\boldsymbol{x}_i + b^*\right) - 1\right) = 0$
* Stationarity: $\nabla_{\boldsymbol{w},b} \mathcal{L}(\boldsymbol{w}^*, b^*, \boldsymbol{\lambda}^*) = \mathbf{0}$

Stationarity conditions give us more information:

$\frac{\partial \mathcal{L}}{\partial b} = \sum_{i=1}^{n} \lambda_i y_i = 0$ ⟹ New constraint

$\frac{\partial \mathcal{L}}{\partial w_j} = {w_j}^* - \sum_{i=1}^{n} \lambda_i y_i (\boldsymbol{x}_i)_j = 0$ ⟹ Eliminates primal variables

The Lagrangian becomes (with additional constraint, above)

$$\mathcal{L}(\boldsymbol{\lambda}) = \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j \boldsymbol{x}_i' \boldsymbol{x}_j$$

Having minimised the Lagrangian with respect to primal variables, now maximising w.r.t dual variables yields the dual program

$$\underset{\boldsymbol{\lambda}}{\text{argmax}} \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j \boldsymbol{x}_i' \boldsymbol{x}_j$$

$$\text{s.t. } \lambda_i \geq 0 \text{ and } \sum_{i=1}^{n} \lambda_i y_i = 0$$

- Predict:

Testing: classify new instance $\boldsymbol{x}$ based on sign of

$$s = b^* + \sum_{i=1}^{n} \lambda_i^* y_i \boldsymbol{x}_i' \boldsymbol{x}$$

- Soft-margin SVM Dual:

- Training: find $\boldsymbol{\lambda}$ that solves

$$\underset{\boldsymbol{\lambda}}{\text{argmax}} \sum_{i=1}^{n} \lambda_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \lambda_i \lambda_j y_i y_j \boldsymbol{x}_i' \boldsymbol{x}_j$$

box constraints

$$\text{s.t. } C \geq \lambda_i \geq 0 \text{ and } \sum_{i=1}^{n} \lambda_i y_i = 0$$

- Making predictions: same pattern as in as in hard-margin case

Kernelization: Apply the original feature space to a higher dimensional feature space. Typically, we use $\varphi(x)$ to denote kernel function. Kernel function can be described as the dot product of feature space. $K(\boldsymbol{u}, \boldsymbol{v}) = \varphi(\boldsymbol{u})^T \varphi(\boldsymbol{v})$. However, transforming to new feature space and then computing dot product will be computationally inefficient. What we need to do is use kernel trick to compute from the original feature space. The kernel matrix is $K_{ij} = \varphi(\boldsymbol{x_i})^T \varphi(\boldsymbol{x_j})$, where K_{ij} is the i-th row and j-th column of the kernel matrix.

- Polynomial Kernel:

$$K(\boldsymbol{x_i}, \boldsymbol{x_j}) = (c + \boldsymbol{x_i}^T \boldsymbol{x_j})^d$$

- Gaussian Kernel:

$$K(\boldsymbol{x_i}, \boldsymbol{x_j}) = \exp(-\gamma ||\boldsymbol{x_i} - \boldsymbol{x_j}||)^2$$

where γ is the hyperparameter to adjust the smoothness of the decision boundary.

- Kernel Properties:
  - $K(\boldsymbol{u}, \boldsymbol{v}) = K_1(\boldsymbol{u}, \boldsymbol{v}) + K_2(\boldsymbol{u}, \boldsymbol{v})$
  - $K(\boldsymbol{u}, \boldsymbol{v}) = cK_1(\boldsymbol{u}, \boldsymbol{v})$
  - $K(\boldsymbol{u}, \boldsymbol{v}) = f(\boldsymbol{u}) K_1(\boldsymbol{u}, \boldsymbol{v}) f(\boldsymbol{v})$
- Mercer's Theorem: If the kernel matrix K is positive semi-definite that holds for all possible sequences $\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_n}$, we call K is a valid kernel.

Multi-armed Bandit:

- Settings:

- Possible actions $\{1, \ldots, k\}$ called "arms"
  - Arm i has distribution P_i on bounded rewards with mean μ_i
- In round $t = 1 \ldots T$
  - Play action $i_t \in \{1, \ldots, k\}$ *(possibly randomly)*
  - Receive reward $R_{i_t}(t) \sim P_{i_t}$
- Goal: minimise cumulative regret
  - $\mu^* T - \sum_{t=1}^{T} E[R_{i_t}(t)]$

Expected cumulative reward of bandit

Best expected cumulative reward with hindsight

where $\mu^* = \max_i \mu_i$

  - Intuition: Do as well as a rule that is simple but has knowledge of the future

- Greedy Approach:

* Estimate value of each arm *i* as average reward observed

$$Q_{t-1}(i) = \begin{cases} \frac{\sum_{s=1}^{t-1} R_i(s) 1[i_s = i]}{\sum_{s=1}^{t-1} 1[i_s = i]}, & \text{if } \sum_{s=1}^{t-1} 1[i_s = i] > 0 \\ Q_0, & \text{otherwise} \end{cases}$$

... some init constant $Q_0(i) = Q_0$ used until arm *i* has been pulled

* Exploit, baby, exploit!

$$i_t \in \arg\max_{1 \le i \le k} Q_{t-1}(i)$$

* Tie breaking randomly

- ε-greedy:

* Estimate value of each arm *i* as average reward observed

$$Q_{t-1}(i) = \begin{cases} \frac{\sum_{s=1}^{t-1} R_i(s) 1[i_s = i]}{\sum_{s=1}^{t-1} 1[i_s = i]}, & \text{if } \sum_{s=1}^{t-1} 1[i_s = i] > 0 \\ Q_0, & \text{otherwise} \end{cases}$$

... some init constant $Q_0(i) = Q_0$ used until arm *i* has been pulled

* Exploit, baby exploit... probably; or possibly explore

$$i_t \sim \begin{cases} \arg\max_{1 \le i \le k} Q_{t-1}(i) & w.p.\ 1-\varepsilon \\ \text{Unif}(\{1, \dots, k\}) & w.p.\ \varepsilon \end{cases}$$

* Tie breaking randomly

If we set Q_0 below the observable rewards, we only explore one arm and then never explore other arms.
If we set Q_0 above the observable rewards, we explore each arm at least once.
Limitation: Too deterministic, lowering ε as the number of pulling arms is better; Initialization trick is important.

- Upper Confidence Bound:

* Estimate value of each arm *i* as average reward observed

$$Q_{t-1}(i) = \begin{cases} \hat{\mu}_{t-1}(i) + \sqrt{\frac{2\log(t)}{N_{t-1}(i)}}, & \text{if } \sum_{s=1}^{t-1} 1[i_s = i] > 0 \\ Q_0, & \text{otherwise} \end{cases}$$

...some constant $Q_0(i) = Q_0$ used until arm *i* has been pulled; where:

$$N_{t-1}(i) = \sum_{s=1}^{t-1} 1[i_s = i] \qquad \hat{\mu}_{t-1}(i) = \frac{\sum_{s=1}^{t-1} R_i(s) 1[i_s = i]}{\sum_{s=1}^{t-1} 1[i_s = i]}$$

* "Optimism in the face of uncertainty"

$$i_t \sim \arg\max_{1 \le i \le k} Q_{t-1}(i)$$

...tie breaking randomly

We can introduce a tunable term ρ, which adjusts the scale of exploration.

$$Q_{t-1}(i) = \begin{cases} \hat{\mu}_{t-1}(i) + \sqrt{\frac{\rho\log(t)}{N_{t-1}(i)}}, & \text{if } \sum_{s=1}^{t-1} 1[i_s = i] > 0 \\ Q_0, & \text{otherwise} \end{cases}$$

* Captures different ε rates & bounded rewards outside [0,1]

- Contextual Bandits: For each round, we observe arbitrary context vector representing state $\boldsymbol{X_i}$, so reward estimation is $\mathbb{E}[R_i(t)|\boldsymbol{X_i}]$ instead of $\mathbb{E}[R_i(t)]$.

Bayesian Regression: We use posterior distribution to fit the parameters $\boldsymbol{w}$, instead of a point estimate. When making predictions, we consider all these weights, scaled by their probability, which is less sensitive to overfitting.

- Conjugate Prior: The product of likelihood and prior results in the same distribution as the prior.

	likelihood	conjugate prior
regression	Normal	Normal (for mean)
	Normal	Inverse Gamma (for variance) or Inverse Wishart (covariance)
classification	Binomial	Beta
	Multinomial	Dirichlet
counts	Poisson	Gamma

- Bayesian Linear Regression:

$$
\begin{aligned}
p(\mathbf{w}|\mathbf{X},\mathbf{y},\sigma^2) &\propto \text{Normal}(\mathbf{w}|\mathbf{0},\gamma^2\mathbf{I}_D)\text{Normal}(\mathbf{y}|\mathbf{X}\mathbf{w},\sigma^2\mathbf{I}_N) \\
&\propto \text{Normal}(\mathbf{w}|\mathbf{w}_N,\mathbf{V}_N)
\end{aligned}
$$

where $\mathbf{w}_N = \frac{1}{\sigma^2}\mathbf{V}_N\mathbf{X}'\mathbf{y}$

$$\mathbf{V}_N = \sigma^2(\mathbf{X}'\mathbf{X} + \frac{\sigma^2}{\gamma^2}\mathbf{I}_D)^{-1}$$

$$
\begin{aligned}
p(y_*|\mathbf{x}_*,\mathbf{X},\mathbf{y},\sigma^2) &= \int p(\mathbf{w}|\mathbf{X},\mathbf{y},\sigma^2)p(y_*|\mathbf{x}_*,\mathbf{w},\sigma^2)d\mathbf{w} \\
&= \int \text{Normal}(\mathbf{w}|\mathbf{w}_N,\mathbf{V}_N)\text{Normal}(y_*|\mathbf{x}_*'\mathbf{w},\sigma^2)d\mathbf{w} \\
&= \text{Normal}(y_*|\mathbf{x}_*'\mathbf{w}_N,\sigma_N^2(\mathbf{x}_*)) \\
\sigma_N^2(\mathbf{x}_*) &= \sigma^2 + \mathbf{x}_*'\mathbf{V}_N\mathbf{x}_*
\end{aligned}
$$

- Sequential Bayesian Updating:

1. Start from prior $p(\mathbf{w})$
2. See new labelled datapoint
3. Compute posterior $p(\mathbf{w}|\mathbf{X},\mathbf{y},\sigma^2)$
4. The ***posterior now takes role of prior***
   & repeat from step 2

Bayesian Classification:

Model is *discriminative*, with parameters defined using logistic sigmoid*

$$
\begin{aligned}
p(y|q,\mathbf{x}) &= q^y(1-q)^{1-y} \\
q &= \sigma(\mathbf{x}'\mathbf{w})
\end{aligned}
$$

* need prior over ***w***, not *q*
* no known conjugate prior (!), thus use a Gaussian prior

$$
\begin{aligned}
p(\mathbf{w}|\mathbf{X},\mathbf{y}) &\propto p(\mathbf{w})p(\mathbf{y}|\mathbf{X},\mathbf{w}) \\
&= \text{Normal}(\mathbf{0},\sigma^2\mathbf{I})\prod_{i=1}^{n}\sigma(\mathbf{x}_i'\mathbf{w})^{y_i}(1-\sigma(\mathbf{x}_i'\mathbf{w}))^{1-y_i}
\end{aligned}
$$

Probability Graphical Models:

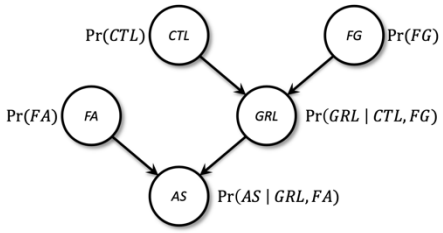
- Directed PGMs: To represent joint distribution in a more efficient way. Full joint distribution requires $2^M - 1$ free parameters, if M is the number of boolean variables, which means we need

to spend very long time marginalization and the parameters to fit are way more larger.

Lazy Lecturer Model	Model details	# params
Our model with S, T independence	$\Pr(S, T)$ factors to $\Pr(S)\Pr(T)$	2
	$\Pr(L \mid T, S)$ modelled in full	4
Assumption-free model	$\Pr(L, T, S)$ modelled in full	7

The conditional independence is like $P(child|parents)$, and the joint distribution is the product of all conditional distributions.

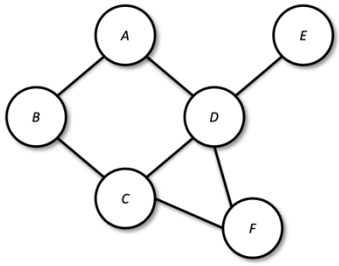
- Core temperature
  → Temperature Gauge
  → Alarm
- Model uncertainty in monitoring failure
  - GRL: gauge reads low
  - CTL: core temperature low
  - FG: faulty gauge
  - FA: faulty alarm
  - AS: alarm sounds



Joint $\Pr(CTL, FG, FA, GRL, AS)$ given by

- Undirected PGMs: Each conditional probability has a factor C for normalization.

- Based on notion of
  - *Clique: a set of fully connected nodes (e.g., A-D, C-D, C-D-F)*
  - *Maximal clique: largest cliques in graph (not C-D, due to C-D-F)*



- Joint probability defined as

$$P(a,b,c,d,e,f) = \frac{1}{Z}\psi_1(a,b)\psi_2(b,c)\psi_3(a,d)\psi_4(d,c,f)\psi_5(d,e)$$

  - where each ψ is a positive function and Z is the normalising 'partition' function

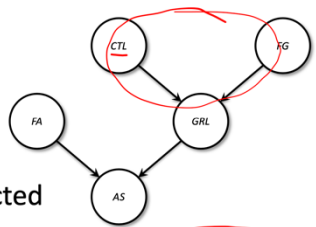
$$Z = \sum_{a,b,c,d,e,f} \psi_1(a,b)\psi_2(b,c)\psi_3(a,d)\psi_4(d,c,f)\psi_5(d,e)$$

22

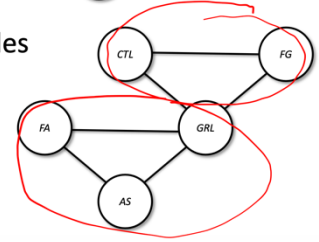
- Convert D-PGMs to U-PGMs: Copy nodes and and edges, and moralize all parent nodes.

$P(GRL \mid CTL, FG)$

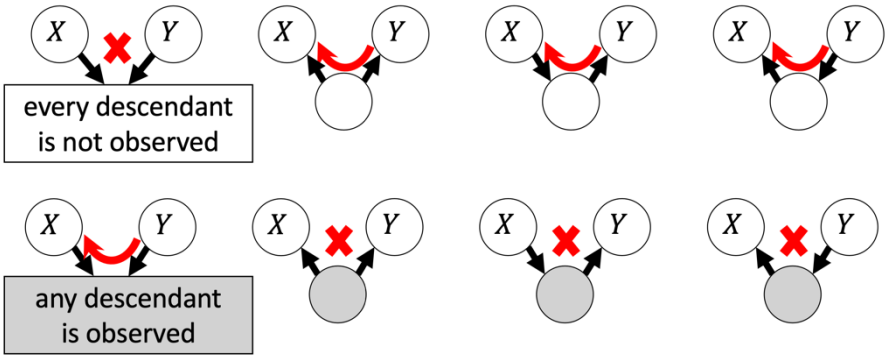
1. copy nodes
2. copy edges, undirected
3. 'moralise' parent nodes



$\psi(CTL, FG, GRL)$



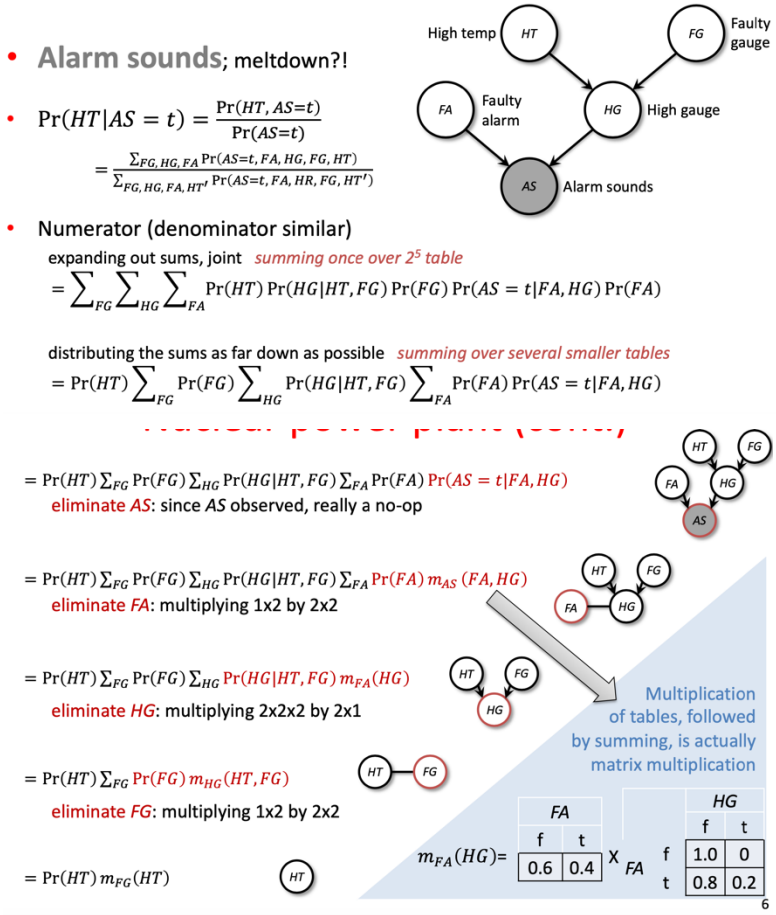
- Marginal Independence: $P(X,Y) = P(X)P(Y)$. The method we determine two nodes in a graph is marginally independent is write joint distribution and then factorize in order to eliminate other variables. The marginal independence is denoted to $X \perp Y$.
- Conditional Independence: $P(X,Y \mid Z) = P(X \mid Z)P(Y \mid Z)$. The method we determine two nodes in a graph is conditionally independent given a node can factorize the joint distribution to eliminate other variables. The conditional independence is denoted to $X \perp Y \mid Z$.
- Topology:



For undirected graph, if there is no path from one node to another, then the two nodes are independent.

- Probabilistic Inference: Compute marginal or conditional distributions from the joint of PGMs using Bayes rule or marginalization.

e.g.



Each time one node is eliminated, the graph will be reconstructed, i.e. moralization.

- Runtime of Elimination Algorithm:

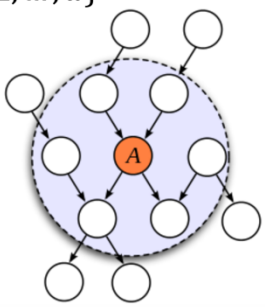
moralisation

- Each step of elimination
  - Removes a node
  - Connects node's remaining neighbours
    → forms a clique in the "reconstructed" graph
    (cliques are exactly r.v.'s involved in each sum)
- Time complexity exponential in largest clique
- Different elimination orderings produce different cliques
  - Treewidth: minimum over orderings of the largest clique
  - Best possible time complexity is exponential in the treewidth e.g. $O(2^{tw})$

- Approximate Inference: When probabilistic inference is expensive or the summation or integration has no analytical solution, we can approximate it by sampling. However, naïve sampling methods are inefficient in high dimensions.
- Gibbs Sampling:

1. Initialise with a starting $\mathbf{X}^{(0)} = \left(X_1^{(0)}, \dots, X_d^{(0)}\right)$ with $\mathbf{X}_E^{(0)} = \mathbf{x}_E$
2. Repeat many times
   a) Pick non-evidence node X_j uniformly at random
   b) Sample single node $X_j' \sim p\left(X_j | X_1^{(i-1)}, \dots, X_{j-1}^{(i-1)}, X_{j+1}^{(i-1)}, \dots, X_d^{(i-1)}\right)$
   c) Save entire joint sample $\mathbf{X}^{(i)} = \left(X_1^{(i-1)}, \dots, X_{j-1}^{(i-1)}, X_j', X_{j+1}^{(i-1)}, \dots, X_d^{(i-1)}\right)$

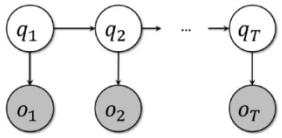
- Markov Blanket:
- Consider node X_i in D-PGM on nodes $N = \{1, \dots, d\}$
- Markov blanket $\mathrm{MB}(i)$ of X_i:
  - Nodes $B \subseteq N \backslash \{i\}$ such that…
  - X_i independent of $\mathbf{X}_{\bar{B} \backslash \{i\}}$ given $\mathbf{X}_B$
  - $p(X_i \mid X_1, \dots, X_{i-1}, X_{i+1}, \dots, X_d) = p(X_i \mid \mathrm{MB}(X_i))$
- In D-PGM Markov blanket is:
  - Parents of i, children of i, parents of children of i
  - $p(X_i \mid \mathrm{MB}(X_i)) \propto p(X_i | X_{\pi_i}) \prod_{k:i \in \pi_k} p(X_k | X_{\pi_k})$



- Statistical Inference: Based on observations, we estimate the marginal or conditional probability. If all nodes are fully observable, we use maximum likelihood estimate to inference. If there are any unobserved variables, we use other techniques, such as EM-algorithm.

Hidden Markov Model: Used for sequential data.

- Formulated as directed PGM
  - therefore joint expressed as



$$P(\mathbf{o}, \mathbf{q}) = P(q_1) P(o_1 | q_1) \prod_{i=2}^{T} P(q_i | q_{i-1}) P(o_i | q_i)$$

  - **bold** variables are shorthand for vector of *T* values
- Parameters (for *homogenous* HMM)

$A = \{a_{ij}\}$ transition probability matrix; $\forall i : \sum_j a_{ij} = 1$
$B = \{b_i(o_k)\}$ output probability matrix; $\forall i : \sum_k b_i(o_k) = 1$
$\Pi = \{\pi_i\}$ the initial state distribution; $\sum_i \pi_i = 1$

- HMM Evaluation: Given HMM parameter μ and observed sequence $\boldsymbol{o}$, determine the likelihood $P(\boldsymbol{o}|\mu)$. We can marginalize by either backward or forward algorithm.

$$P(\mathbf{o}|\mu) = \sum_{q_1} P(q_1)P(o_1|q_1) \sum_{q_2} P(q_2|q_1)P(o_2|q_2) \dots \sum_{q_T} P(q_T|q_{T-1})P(o_T|q_T)$$

Eliminate q_T — $m_{T \to T-1}(q_{T-1})$

...

Eliminate q_2 — $m_{2 \to 1}(q_1)$

"Eliminate" q_1

$$P(\mathbf{o}|\mu) = \sum_{q_1} P(q_1)P(o_1|q_1)m_{2 \to 1}(q_1)$$

$$P(\mathbf{o}|\mu) = \sum_{q_T} P(o_T|q_T) \sum_{q_{T-1}} P(q_T|q_{T-1})P(o_T|q_T) \dots \sum_{q_1} P(q_2|q_1)P(q_1)P(o_1|q_1)$$

Eliminate q_1 — $m_{1 \to 2}(q_2)$

...

Eliminate q_{T-1} — $m_{T-1 \to T}(q_T)$

"Eliminate" q_T

$$P(\mathbf{o}|\mu) = \sum_{q_1} P(o_T|q_T)m_{T-1 \to T}(q_T)$$

Both have $O(TL^2)$, where T is the sequence size and L is the label set size.

- Decoding: Given HMM parameters μ and observed sequence $\boldsymbol{o}$, determine the most probable hidden state sequence $\boldsymbol{q}$. We can implement it using Viterbi algorithm by replacing summation with argmax operation in forward or backward algorithm.
- Learning: Given an observed sequence $\boldsymbol{o}$, and set of states, learn HMM parameters μ. Use EM to approximate MLE, i.e., $\arg\max_{\boldsymbol{\mu}} P(q_t \mid \boldsymbol{o}, \boldsymbol{\mu})$.

1. initialise $\boldsymbol{\mu}^1$, let *j=1*
2. compute expected marginal distributions $P(q_t|\mathbf{o}, \boldsymbol{\mu}^j)$ for all *t*; and $P(q_{t-1},q_t|\mathbf{o}, \boldsymbol{\mu}^j)$ for *t*=2..*T*
3. fit model $\boldsymbol{\mu}^{j+1}$ based on expectations
4. repeat from step 2, with *j=j+1*

Unsupervised Learning: There is no labels in an instance, so the objective is to explore the structures of the data. Unsupervised learning can be used as clustering, dimensionality reduction and learning parameters of probabilistic models.

- K-means Algorithm:

1. Initialisation: choose k cluster centroids randomly
2. Update:
   a) Assign points to the nearest* centroid
   b) Compute centroids under the current assignment
3. Termination: if no change then stop
4. Go to Step 2

- Gaussian Mixture Model: Instead assigning each point to an exact one cluster 100%, GMM assigns each data point to each cluster with some probability to express the uncertainty.

- Like *k*-means, a probabilistic mixture model requires the user to choose the number of clusters in advance
- Unlike *k*-means, the probabilistic model gives us a power to express uncertainly about the origin of each point
  - Each point originates from cluster c with probability w_c, $c = 1, \dots, k$
- That is, each point still originates from one particular cluster (aka component), but we are not sure from which one

The probability of each cluster is modeled as the categorical distribution:

$$P(C_j) = w_j$$

Each data within a particular cluster is modeled as a Gaussian distribution:

$$P(\boldsymbol{X}|C_j) = \mathcal{N}(\boldsymbol{\mu_j}, \boldsymbol{\Sigma_j})$$

The mixture of Gaussian distribution:

$$p(\boldsymbol{x}) = \sum_{j=1}^{k} w_j \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu_j}, \boldsymbol{\Sigma_j})$$

Data likelihood:

$$P(\boldsymbol{x_1}, \boldsymbol{x_2}, \dots, \boldsymbol{x_n}) = \prod_{i=1}^{n} \sum_{j=1}^{k} P(C_j) P(\boldsymbol{x_i}|C_j)$$

Taking logarithm:

$$\log P(\boldsymbol{x_1}, \boldsymbol{x_2}, \dots, \boldsymbol{x_n}) = \sum_{i=1}^{n} \log \sum_{j=1}^{k} P(C_j) P(\boldsymbol{x_i}|C_j)$$

However, finding the maximum value analytically the above is hard, so we consider the EM algorithm the approximate the maximum value.

Expectation-Maximization Algorithm:

- Initialisation Step:
  - Initialize *K* clusters: $C_1, \dots, C_K$
    (μ_j, Σ_j) and $P(C_j)$ for each cluster *j*.
- Iteration Step:
  - Estimate the cluster of each datum
    $p(C_j \mid x_i)$ ➡ Expectation
  - Re-estimate the cluster parameters
    $(\mu_j, \Sigma_j), p(C_j)$ for each cluster *j* ➡ Maximisation

- Coordinate Ascent:

* Coordinate ascent on a lower bound on the log-likelihood
  - M-step: ascent in modeled parameters **θ**
  - E-step: ascent in the marginal likelihood P(**Z**)
* Each step moves towards a ***local*** optimum
* Can get stuck, can need random restarts

1. Initialisation: choose (random) initial values of $\boldsymbol{\theta}^{(1)}$
2. Update:
   * E-step: compute $Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t)}) \equiv \mathbb{E}_{\boldsymbol{Z}|\boldsymbol{X},\boldsymbol{\theta}^{(t)}}[\log p(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\theta})]$
   * M-step: $\boldsymbol{\theta}^{(t+1)} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\, Q(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t)})$
3. Termination: if no change then stop
4. Go to Step 2

This algorithm will eventually stop (converge), but the resulting estimate can be only a local maximum

- Maximize Lower Bound:

$\log p(\boldsymbol{X}|\boldsymbol{\theta}) = \log \sum_{\boldsymbol{Z}} p(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\theta})$ ← Marginalisation (here $\sum_{\boldsymbol{Z}}$ ... iterates over all possible values of $\boldsymbol{Z}$)

$= \log \sum_{\boldsymbol{Z}} \left(p(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\theta}) \frac{p(\boldsymbol{Z})}{p(\boldsymbol{Z})} \right)$ ← Need $\boldsymbol{Z}$ to have non-zero marginal

$= \log \sum_{\boldsymbol{Z}} \left(p(\boldsymbol{Z}) \frac{p(\boldsymbol{X},\boldsymbol{Z}|\boldsymbol{\theta})}{p(\boldsymbol{Z})} \right)$

$= \log \mathbb{E}_{\boldsymbol{Z}} \left[\frac{p(\boldsymbol{X},\boldsymbol{Z}|\boldsymbol{\theta})}{p(\boldsymbol{Z})} \right]$

$\geq \mathbb{E}_{\boldsymbol{Z}} \left[\log \frac{p(\boldsymbol{X},\boldsymbol{Z}|\boldsymbol{\theta})}{p(\boldsymbol{Z})} \right]$ ← Jensen's inequality holds in this direction since log(...) is a concave function

$= \mathbb{E}_{\boldsymbol{Z}}[\log p(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\theta})] - \mathbb{E}_{\boldsymbol{Z}}[\log p(\boldsymbol{Z})]$

- Make the Lower Bound Tight:

- $\log p(\boldsymbol{X}|\boldsymbol{\theta}) \geq \mathbb{E}_{\boldsymbol{Z}} \left[\log \frac{p(\boldsymbol{X},\boldsymbol{Z}|\boldsymbol{\theta})}{p(\boldsymbol{Z})} \right]$

$= \mathbb{E}_{\boldsymbol{Z}} \left[\log \frac{p(\boldsymbol{Z}|\boldsymbol{X},\boldsymbol{\theta}) p(\boldsymbol{X}|\boldsymbol{\theta})}{p(\boldsymbol{Z})} \right]$ ← Chain rule of probability

$= \mathbb{E}_{\boldsymbol{Z}} \left[\log \frac{p(\boldsymbol{Z}|\boldsymbol{X},\boldsymbol{\theta})}{p(\boldsymbol{Z})} + \log p(\boldsymbol{X}|\boldsymbol{\theta}) \right]$

$= \mathbb{E}_{\boldsymbol{Z}} \left[\log \frac{p(\boldsymbol{Z}|\boldsymbol{X},\boldsymbol{\theta})}{p(\boldsymbol{Z})} \right] + \mathbb{E}_{\boldsymbol{Z}}[\log p(\boldsymbol{X}|\boldsymbol{\theta})]$ ← Linearity of $\mathbb{E}[.]$

$= \mathbb{E}_{\boldsymbol{Z}} \left[\log \frac{p(\boldsymbol{Z}|\boldsymbol{X},\boldsymbol{\theta})}{p(\boldsymbol{Z})} \right] + \log p(\boldsymbol{X}|\boldsymbol{\theta})$ ← $\mathbb{E}[.]$ of a constant

- $\log p(\boldsymbol{X}|\boldsymbol{\theta}) \geq \mathbb{E}_{\boldsymbol{Z}} \left[\log \frac{p(\boldsymbol{Z}|\boldsymbol{X},\boldsymbol{\theta})}{p(\boldsymbol{Z})} \right] + \log p(\boldsymbol{X}|\boldsymbol{\theta})$

When $p(\boldsymbol{Z}) = p(\boldsymbol{Z}|\boldsymbol{X}, \boldsymbol{\theta})$, the lower bound of $p(\boldsymbol{X}|\boldsymbol{\theta})$ tight.

- EM in GMM:

  E-step:

- Setting latent Z as originating cluster, yields (via Bayes rule)

$$P\left(z_i = c|\boldsymbol{x}_i, \boldsymbol{\theta}^{(t)}\right) = \frac{w_c \mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)}{\sum_{l=1}^{k} w_l \mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)}$$

- This probability is called *responsibility* that cluster c takes for data point i

$$r_{ic} \equiv P\left(z_i = c|\boldsymbol{x}_i, \boldsymbol{\theta}^{(t)}\right)$$

$$Q\left(\boldsymbol{\theta}, \boldsymbol{\theta}^{(t)}\right) \equiv \mathbb{E}_{\boldsymbol{z}|\boldsymbol{X},\boldsymbol{\theta}^{(t)}}[\log p(\boldsymbol{X}, \boldsymbol{z}|\boldsymbol{\theta})]$$

$$= \sum_{\boldsymbol{z}} p\left(\boldsymbol{z}|\boldsymbol{X}, \boldsymbol{\theta}^{(t)}\right) \log p(\boldsymbol{X}, \boldsymbol{z}|\boldsymbol{\theta})$$

$$= \sum_{\boldsymbol{z}} p\left(\boldsymbol{z}|\boldsymbol{X}, \boldsymbol{\theta}^{(t)}\right) \sum_{i=1}^{n} \log w_{z_i} \mathcal{N}\left(\boldsymbol{x}_i|\boldsymbol{\mu}_{z_i}, \boldsymbol{\Sigma}_{z_i}\right)$$

$$= \sum_{i=1}^{n} \sum_{\boldsymbol{z}} p\left(\boldsymbol{z}|\boldsymbol{X}, \boldsymbol{\theta}^{(t)}\right) \log w_{z_i} \mathcal{N}\left(\boldsymbol{x}_i|\boldsymbol{\mu}_{z_i}, \boldsymbol{\Sigma}_{z_i}\right)$$

$$= \sum_{i=1}^{n} \sum_{c=1}^{k} r_{ic} \log w_{z_i} \mathcal{N}\left(\boldsymbol{x}_i|\boldsymbol{\mu}_{z_i}, \boldsymbol{\Sigma}_{z_i}\right)$$

$$= \sum_{i=1}^{n} \sum_{c=1}^{k} r_{ic} \log w_{z_i}$$

$$+ \sum_{i=1}^{n} \sum_{c=1}^{k} r_{ic} \log \mathcal{N}\left(\boldsymbol{x}_i|\boldsymbol{\mu}_{z_i}, \boldsymbol{\Sigma}_{z_i}\right)$$

M-step:

$$w_c^{(t+1)} = \frac{1}{n} \sum_{i=1}^{n} r_{ic}$$

$$\boldsymbol{\mu}_c^{(t+1)} = \frac{\sum_{i=1}^{n} r_{ic} \boldsymbol{x}_i}{r_c}$$

* Here $r_c \equiv \sum_{i=1}^{n} r_{ic}$

$$\boldsymbol{\Sigma}_c^{(t+1)} = \frac{\sum_{i=1}^{n} r_{ic} \boldsymbol{x}_i \boldsymbol{x}_i'}{r_c} - \boldsymbol{\mu}_c^{(t)} \left(\boldsymbol{\mu}_c^{(t)}\right)'$$