Machine Learning

1. Supervised Learning

- Feature normalization: $\forall x_{ij} \in X, x_{ij} = \frac{x_{ij}-\mu_j}{\sigma_j^2}$, $X : [instance][feature]$, without $[1...1]^T$ in 1st column
  - $X = [x_1, x_2, \ldots, x_m]$, m instances in total
- Regularization: add penalty for $\boldsymbol{\theta}$ being large into cost function
  - $J(\theta) = \ \ldots + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$, θ_0 **shouldn't be penalized**
- Matlab/Octave tips:
  - Unrolling parameters: deltaVector = [D1(:) ; D2(:)] // *matrix D1,D2 => column vector*
  - Reshape: D1 = reshape (deltaVector(1, m*n), m, n) // *column vector => m*n matrix*

Linear Regression

- Assumption: $y = h_\theta^\star(x) + \epsilon$, where $h_\theta^\star(x)$ is the theoretical (perfect) hypothesis, ϵ is gaussian noise
- General Hypothesis Function: $h_\theta(x) = \phi(x)\theta$, where $\phi(x)$ is basis function
- Log Likelihood leads to least square cost function $J(\theta)$
  - $P(y|x,\theta,\beta) = \mathcal{N}(x|h_\theta(x), \beta^{-1})$ - likelihood
    - based on assumption: $y = h_\theta^\star(x) + \epsilon$
      where $h_\theta^\star(x)$ is the theoretical (perfect) hypothesis, ϵ is gaussian noise
  - $P(\boldsymbol{y}|X,\theta,\beta) = \prod_{i=1}^{m}\mathcal{N}(x^i|h_\theta(x^i), \beta^{-1})$
  - $\ln P(\boldsymbol{y}|X,\theta,\beta) = \frac{N}{2}\ln\beta - \frac{N}{2}\ln(2\pi) - \beta\frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^i) - y^i)^2$
- Log Posterior leads to regularization
  - Maximizing the likelihood function => (always) excessively complex models & over-fitting
  - Regularization term comes from the Prior:
    - assume Prior $p(\theta) = \mathcal{N}(\theta|0, \alpha^{-1}I)$, so that Posterior & Prior are of the same distribution
      to maximize log Posterior :
      $\Rightarrow \ln p(\theta|X) \propto -\frac{\beta}{2}\sum_{i=1}^{n}(y^i - h_\theta(x))^2 - \frac{\alpha}{2}\theta^T\theta + const$
    - If $\alpha \to 0$ (Prior is most useless), maximize Posterior is equivalent to maximizing likelihood
    - Maximize Posterior $\Leftrightarrow$ Minimize cost function with regularization, where $\lambda = \alpha/\beta$
- Predictive Distribution: $p(y|x,X,Y)$
  - $p(y|x,X,Y) = \int p(y,\theta|x,X,Y)d\theta = \int p(y|\theta,x,X,Y)p(\theta|x,X,Y)d\theta$
  - $p(y|\theta,x,X,Y) = p(y|\theta,x) = \mathcal{N}(y|h(x,\theta), \beta^{-1})$
    based on assumption: $y = y(x,\theta) + \epsilon$, where ϵ is Guassian noise
    $p(\theta|x,X,Y) = p(\theta|X,Y) = \text{posterior}$
  - $\Rightarrow p(y|x,X,Y) = \int p(y|\theta,x)p(\theta|X,Y)d\theta$

- $Expected\ Lost = (bias)^2 + variance + noise$
  - Notation:
    - $t = y(x, w) + \epsilon,\ \text{where } \epsilon \text{ is Gaussian noise}$
    - $\hat{y}$ is prediction function to approximate $y = y(x, w)$
  - Procedure:
    - $\mathbb{E}[(t - \hat{y})^2] = \mathbb{E}[t^2 - 2t\hat{y} + \hat{y}^2]$
      $= \mathbb{E}[t^2] + \mathbb{E}[\hat{y}^2] - \mathbb{E}[2t\hat{y}]$
      $= \mathrm{Var}[t] + \mathbb{E}[t]^2 + \mathrm{Var}[\hat{y}] + \mathbb{E}[\hat{y}]^2 - 2y\mathbb{E}[\hat{y}]$
      $= \mathrm{Var}[t] + \mathrm{Var}[\hat{y}] + (y^2 - 2y\mathbb{E}[\hat{y}] + \mathbb{E}[\hat{y}]^2)$
      $= \mathrm{Var}[t] + \mathrm{Var}[\hat{y}] + (y - \mathbb{E}[\hat{y}])^2$
      $= \mathrm{Var}[t] + \mathrm{Var}[\hat{y}] + \mathbb{E}[t - \hat{y}]^2$
      $= \sigma^2 + \mathrm{Var}[\hat{y}] + \mathrm{Bias}[\hat{y}]^2$

      $\text{where } \sigma^2 = \mathrm{Var}[\epsilon] \text{ is the noise}$

      (formula used: $\mathrm{Var}[x] = \mathbb{E}[x^2] - \mathbb{E}[x]^2 \Leftrightarrow \mathbb{E}[x^2] = \mathrm{Var}[x] + \mathbb{E}[x]^2$)
- Matrix inverse can be evil & avoid inverse operation:

  $A = U\Lambda U^T, \text{ where } \Lambda \text{ is diagonal matirx}$
  $=> A^{-1} = U\Lambda^{-1}U^T$
  $\text{but number on the diagonal line of } \Lambda \text{ can be small} => \text{maybe 0 depending on accuracy of computer}$

1. Bayesian Regression:
   - Assumption:
     - $t = y(x, w) + \epsilon,\ \text{where } \epsilon \text{ is Gaussian noise; } y(x, w) \text{ approximated by } \phi(x)w$
   - Bayesian view:
     - Gaussian $\text{Prior}: p(w) = \mathcal{N}(w|m_0, S_0)$

       $\text{Reason: to be conjugate}$
     - $\text{Likelihood}: p(\boldsymbol{t}|w) = \prod_{n=1}^{N} \mathcal{N}(t_n|w^T\phi(x_n), \beta^{-1}) = \mathcal{N}(\boldsymbol{t}|\Phi w, \beta^{-1}I)$
     - $\Rightarrow \text{Posterior}: p(w|\boldsymbol{t}) = \mathcal{N}(w|m_N, S_N)$

       $\text{where } m_N = S_N(S_0^{-1}m_0 + \beta\Phi^T\boldsymbol{t}),\ S_N^{-1} = S_0^{-1} + \beta\Phi^T\Phi$
   - Maximum Likelihood:
     - $\text{Likelihood}: p(\boldsymbol{t}|w) = \prod_{n=1}^{N} \mathcal{N}(t_n|\phi(x_n)w, \beta^{-1})$
     - $\ln \text{Likelihood} = \sum_{n=1}^{N}[-\ln\frac{\beta}{\sqrt{2\pi}} - \frac{\beta}{2}(t_n - \phi(x)w)^2]$
     - $\frac{\partial}{\partial w}\ln \text{Likelihood} = \beta\Phi^T(\boldsymbol{t} - \Phi w)$

       let $\frac{\partial}{\partial w}\ln \text{Likelihood} = 0$

       $\Rightarrow w_{ML} = (\Phi^T\Phi)^{-1}\Phi^T\boldsymbol{t}$
     - $\frac{\partial}{\partial \beta}\ln \text{Likelihood} = -N\beta^{\frac{1}{2}} + \beta^{\frac{3}{2}}(\boldsymbol{t} - \Phi w)^T(\boldsymbol{t} - \Phi w)$

       let $\frac{\partial}{\partial \beta}\ln \text{Likelihood} = 0$

$\Rightarrow \beta^{-1} = \frac{1}{N}(\boldsymbol{t} - \Phi w)^T(\boldsymbol{t} - \Phi w)$

$\text{Note: solve } w = w_{ML} \text{ first}$

- Maximum Posterior:
  - $\text{Posterior} = p(w|\boldsymbol{t}), \text{Prior} = p(w), \text{Likelihood} = p(\boldsymbol{t}|w), \text{Normalization} = p(t)$
    $\Rightarrow \text{Posterior} = \frac{\text{Likelihood*Prior}}{\text{Normalization}}$
    $\Rightarrow \text{Posterior} \propto \text{Likelihood*Prior}$
  - $\text{assume Prior } p(w) = \mathcal{N}(w|m_0, S_0),$
    $\text{so that Prior \& Likelihood are conjugate} \Rightarrow \text{Gaussian Posterior}$
  - $\text{Likelihood } p(\boldsymbol{t}|w) = \prod_{n=1}^{N} \mathcal{N}(t_n|\phi(x_n)w, \beta^{-1}) = \mathcal{N}(\boldsymbol{t}|\Phi w, \beta^{-1}I)$
  - $\Rightarrow \text{Posterior } p(w|\boldsymbol{t}) = \mathcal{N}(w|m_N, S_N),$
    $\text{where } m_N = S_N(S_0^{-1}m_0 + \beta\Phi^T\boldsymbol{t}),\ S_N^{-1} = S_0^{-1} + \beta\Phi^T\Phi$
    $\Rightarrow w_{MAP} = \text{mean of the Gaussian} = m_N$
    $\text{Note: can also get } w_{MAP} \text{ from taking gradient}$
  - Simple Prior:
    $\text{Prior } p(w) = \mathcal{N}(w|0, \alpha^{-1}I)$
    $\Rightarrow \text{Posterior } p(w|\boldsymbol{t}) = \mathcal{N}(w|m_N, S_N),$
    $\text{where } m_N = \beta(\alpha I + \beta\Phi^T\Phi)\Phi^T\boldsymbol{t},\ S_N^{-1} = \alpha I + \beta\Phi^T\Phi$
    $w_{MAP} \to w_{ML},\ \text{when } \alpha \to 0 \text{ (most useless Prior)}$
  - Maximize Posterior ⇔ Minimize $\text{cost function with regularization}$:
    $\text{Simple Prior} \Rightarrow \ln p(w|\boldsymbol{t}) = -\frac{\beta}{2}(\boldsymbol{t} - \Phi w)^T(\boldsymbol{t} - \Phi w) - \frac{\alpha}{2}w^Tw + const$
    - If $\alpha \to 0$ $(\text{Prior is most useless})$, maximize Posterior is equivalent to maximizing likelihood
    - Maximize Posterior equal to minimize sum-of-squares error function with the addition of a quadratic regularization term with $\lambda = \alpha/\beta$
    - Regularization term comes from the Prior
- Predictive Distribution:
  - Assume: $\text{Prior}: p(x|\alpha) = \mathcal{N}(x|0, \alpha^{-1}I),\ \ (m_0 = 0, S_0 = \alpha^{-1}I)$
  - $p(t|x, X, \boldsymbol{t}) = \int p(t|w, x)p(w|X, \boldsymbol{t})dw$
  - $\Rightarrow p(t|x, X, \boldsymbol{t}) = \mathcal{N}(t|m_N^T\phi(x), \sigma_N^2(x))$
    $\text{where } \sigma_N^2(x) = \frac{1}{\beta} + \phi(x)^TS_N\phi(x);\ m_N, S_N \text{ from Posterior}(m_N = w_{MAP})$
- Sequential data:
  - Posterior from previous data ⇔ the Prior for the arriving data
  - Sequential data and data in one go is equivalent when finfding the Porsterior

2. Gradient descent
   - Hypothesis function: $h_\theta(x) = x\theta$, $\theta = [\theta_0, \theta_1, \ldots, \theta_n]^T$, $x = [x_0, x_1, \ldots, x_n], x_0 = 1$
     - x is one instance
   - Cost function: $J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}(h_\theta(x^i) - y^i)^2 + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$

- Update rule: $\forall \theta_j \in \theta, \theta_j := \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j}$, $\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}[(h_\theta(x^i) - y^i)x_j^i] + \frac{\lambda}{m}\theta_j$
  - $\frac{d}{d\theta}J(\theta) = \frac{1}{m}((X\theta - y)^T X)^T + \frac{\lambda}{m}[0, \theta_1, \ldots, \theta_m]^T$ (θ_0 shouldn't be penalized)
  - simultaneously for all $\theta_j \in \theta$
- Normal equation (mathmetical solution)
  - $\theta = (X^T X)^{-1} X^T y$

Logistic Regression (Classification)

- Decision Theory:
  - classes $C_1, \ldots, C_K$, decision regions $\mathcal{R}_1, \ldots, \mathcal{R}_K$
    - Minimze $p(mistake) = \sum_{k=1}^{K}(\int_{\mathcal{R}_k} \sum_{i \neq k} p(x, C_i)dx)$ $\text{(can have weight on each misclassfication though)}$
    - Maximize $p(correct) = \sum_{k=1}^{K} \int_{\mathcal{R}_k} p(x, C_k)dx$
- Models for Decision Problems:
  - Find a discriminant function
  - Discriminative Models: $\text{less powerful, yet less parameter} => \text{easier to learn}$
    - Infer **posterior** $p(C_k|x)$, C_k: $x \in C_k$, x $\text{is examples in trainning set}$
    - Use decision theory to assign a new x
  - Generative Models: $\text{more powerful, but computationally expensive}$
    - Infer conditional probabilities $p(x|C_k)$
    - Infer prior $p(C_k)$
    - Find **either posterior** $p(C_k|x)$, **or joint distribution** $p(x, C_k)$ (using Bayes' theorem)
    - Use decision theory to assign a new x
    - **=> Able to create synthetic data using** $p(x)$
- Naive Bayes on Discrete Features:
  - Assumption:
    - Discrete Features: data point $x \in \{0, 1\}^D$
    - Naive Bayes: all features conditioned on class C_k are independent with each other

      $\Rightarrow p(x|C_k) = \prod_{i=1}^{D} \mu_{ki}^{x_i} \ (1 - \mu_{ki})^{1-x_i}$

1. Linear Discriminant (Least Squares Approach)
   - Prediction:
     - $y(x) = xw + w_0, \text{with bias} = w_0, \text{where } w = [w_1, \ldots, w_n]^T, x = [x_1, \ldots, x_n]$
     - $y(x) > 0$: positive confidence to assign x to current class
     - $-w_0$ called threshold sometimes
   - Decision Boundary $y(x) = w^T x + w_0 = 0$:
     - w is orthogonal to vectors on the boundary:

       $\text{assume } x_1, x_2 \text{ on the boundary}$
       $\Rightarrow 0 = y(x_1) - y_(x_2) = (x_1 - x_2)w$
     - Distance from origin to boundary is $-\frac{w_0}{\|w\|}$:

$\text{assume distance is } k$
$\Rightarrow k\frac{w}{\|w\|} \text{on boundary, thus } k\frac{w}{\|w\|}w + w_0 = 0$
$\Rightarrow k = -\frac{w_0}{\|w\|}$

- $y(x)$ is a signed measure of distance from point x to boundary:

  $\text{assume distance is } r$
  $\Rightarrow y(x) = \overbrace{(x_\perp + r\frac{w}{\|w\|})}^{x}w + w_0 = \overbrace{x_\perp w + w_0}^{0} + r\|w\| = r\|w\|$
  $\Rightarrow r = \frac{y(x)}{\|w\|}$

- Multi-class (k-classes):
  - prediction: x is of class C_k if $\forall j \neq k, y_k(x) > y_j(x)$

    $\Rightarrow y(x) = xW, \text{where } W = [w_1, \ldots, w_k], \forall x_i \in X, x_{i0} = 1 \text{ (bias)}, y(x) \text{ is 1-of-k coding}$
  - sum-of-squares error: $E_D(W) = \frac{1}{2}\text{tr}\{(XW - T)(XW - T)^T\}$

    $\Rightarrow \text{optimal } W = (X^T X)^{-1} X^T T$

    $\text{note that } tr\{AB\} = A^T B^T$

2. Fisher's Linear Discriminant
   - Basic idea:
     - Take linear classification $y = w^T x$ as dimensionality reduction (projection onto 1-D)
     - => find a projection (denoted by vector w) which maximally preserves the class separation
     - => if $y > -w_0$ then class C_1, otherwise C_2
   - Goal:
     - Distance within one class is small
     - Distance between classes is large
   - Mean & Variance of Projected Data:
     - Mean: $\widetilde{m}_k = w^T m_k, \text{ where } m_k = \frac{1}{N_k}\sum_{x \in C_k} x$
     - Variance: $\tilde{s}_k = \sum_{x \in C_k}(w^T x - w^T m_k)^2 = w^T[\sum_{x \in C_k}(x - m_k)(x - m_k)^T]w$
   - 2-Classes to 1-D line:
     - Maximize Fisher criterion: $J(w) = \frac{|\widetilde{m}_1 - \widetilde{m}_2|^2}{\tilde{s}_1{}^2 + \tilde{s}_2{}^2}$
     - Between-class covariance: $S_B = (m_1 - m_2)(m_1 - m_2)^T$
     - Within-class covariance: $S_k = \sum_{n \in C_k}(x_n - m_k)(x_n - m_k)^T$

       $\Rightarrow J(w) = \frac{w^T S_B w}{w^T S_W w}$
     - Lagrangian: $L(w, \lambda) = w^T S_B w + \lambda(1 - w^T S_W w)$

       $\text{fix } w^T S_W w \text{ to 1 to avoid infinite solution (any multiple of a solution is a solution)}$
       $\Rightarrow \frac{\partial}{\partial w}L = 2S_B w - 2\lambda S_W w = 0$
       $\Rightarrow S_B w = \lambda S_W w$
       $\Rightarrow (S_W{}^{-1} S_B)w = \lambda w$
       $\text{To maximize } J(w), w \text{ is the largest eigenvector of } S_W{}^{-1} S_B \text{ if } S_W \text{ invertible}$

- K-classes to a d-D subspace: N_k **is num in class k,** N **is the total example num**
  - Between-class covariance: $S_B = \sum_{k=1}^{K} N_k (m_k - m)(m_k - m)^T$, **where** $m = \frac{1}{N}\sum_{n=1}^{N} x_n$

    reduce to $(m_1 - m_2)(m_1 - m_2)^T$ **when K=2 (constant ignored)**
  - Within-class covariance: $S_W = \sum_{k=1}^{K} S_k$, **where** $S_k = \sum_{n \in C_k} (x_n - m_k)(x_n - m_k)^T, m_k = \frac{1}{N_k}\sum_{n \in C_k} x_n$
  - Maximize Fisher criterion: $J(w) = \frac{tr\{W^T S_B W\}}{tr\{W^T S_W W\}}$
  - Lagrangian:

    Solve for each $w_i \in W \Rightarrow ({S_W}^{-1} S_B) w_i = \lambda_i w_i$

    $\Rightarrow W$ **conosists of the largest d eigenvectors**

    ${S_W}^{-1} S_B$ **is not guaranteed to be symmetric** $\Rightarrow W$ **might not be orthogonal**

    Need to minimize the whole covariance matrix ($J(w)$ **as a matrix**) $\Rightarrow$ **not choosing same eigenvectors twice**
  - Maximum Possible Projection Directions = $K - 1$:

    $r({S_W}^{-1} S_B) \leq \min(r({S_W}^{-1}), r(S_B)) \leq r(S_B)$

    $r(S_B) \leq \sum_{K} r((m_k - m)(m_k - m)^T) = K$, **as** $r(m_k - m) = 1$

    $\sum_{K} m_k = m \Rightarrow r(m_1 - m, \ldots, m_K - m) = K - 1$

    $\Rightarrow r(S_B) \leq K - 1$

    $\Rightarrow r({S_W}^{-1} S_B) \leq K - 1$

3. Perceptron Algorithm
   - Generalised linear model $y = f(w^T \phi(x))$, where $\phi(x)$ is basis function; $\phi_0(x) = 1$
   - Nonlinear activation funtion: $f(a) = \begin{cases} 1, & a \geq 0 \\ -1, & a < 0 \end{cases}$
   - Target coding: $t = \begin{cases} 1, & \text{if } C_1 \\ -1, & \text{if } C_2 \end{cases}$
   - Cost function:
     - All correctly classified patterns: $w^T \phi(x_n) t_n > 0$
     - Add the errors for all misclassified patterns (denoted as set $\mathcal{M}$):

       $\Rightarrow E_P(w) = -\sum_{n \in \mathcal{M}} w^T \phi(x_n) t_n$
   - Algorithm: (Aim: minimize total num of misclassified patterns)
     - loop

       choose a traning pair (x_n, t_n)

       update the weight vector w: $w = w - \eta \nabla E_p(w) = w + \phi_n t_n$

       where η**=1 because** $y = f(\cdot)$ **does not depend on** $\|w\|$
   - Perceptron Convergence Theorem:
     - If the training set is linearly separable, the perceptron algorithm is guaranteed to find a solution in a finite number of steps

       (Also is the algorithm to find whether the set is linearly separable => Halting Problem)

4. Maximum Likelihood
   - Assumption:

- $p(x|C_k) \sim \mathcal{N}(\mu_k, \Sigma)$, and all $p(x|C_k)$ share the same Σ
- $p(C_1) = \pi,\ p(C_2) = 1 - \pi,\ \pi$ unknown

- Likelihood of whole data set $\boldsymbol{X}, \boldsymbol{t}$, N is the num of data
  - $p(\boldsymbol{X}, \boldsymbol{t}|\pi, \mu_1, \mu_2, \Sigma) = \prod_{n=1}^{N} [\pi \mathcal{N}(x_n|\mu_1, \Sigma)]^{t_n}\ [(1-\pi)\mathcal{N}(x_n|\mu_2, \Sigma)^{1-t_n}]$

    $\rightarrow$ when info of label t lost: mixture of Gaussian
  - $\ln(\text{Likelihood}) = \sum_{n=1}^{N} [t_n(\ln \pi + \ln \mathcal{N}(x_n|\mu_1, \Sigma)) + (1-t_n)(\ln(1-\pi) + \ln \mathcal{N}(x_n|\mu_2, \Sigma))]$

- Parameters when maximum reached:
  - $\pi = \dfrac{N_1}{N_1 + N_2}$, N_1 is the num of class C_1
  - $\mu_1 = \dfrac{1}{N_1} \sum_{n=1}^{N} t_n x_n,\ \mu_2 = \dfrac{1}{N_2} \sum_{n=1}^{N} (1-t_n) x_n$, (mean of each class)
  - $\Sigma = \dfrac{N_1}{N} S_1 + \dfrac{N_2}{N} S_2$, where $S_k = \dfrac{1}{N_k} \sum_{n \in C_k} (x_n - \mu_k)(x_n - \mu_k)^T$

5. Logistic Regression

- Sigmoid function: $\sigma(a) = \dfrac{1}{1 + e^{-a}}$
- $p(x|C_k) \sim \mathcal{N} \implies p(C_k|x) = \sigma(w^T x + w_0)$ (2-classes) (Generative model)
  - Assumption:

    $p(x|C_k) = \mathcal{N}(\mu_k, \Sigma)$ (can also be a number of other distributions)

    $\forall k, p(x|C_k)$ shares the same Σ
  - $p(C_1|x) = \dfrac{p(x|C_1)p(C_1)}{p(x|C_1)p(C_1) + p(x|C_2)p(C_2)} = \sigma(a)$,

    where $a = \ln \dfrac{p(x, C_1)}{p(x, C_2)}$

    $= \ln \dfrac{p(x|C_1)p(C_1)}{p(x|C_2)p(C_2)}$

    $= \ldots$ (assumption applied)

    $= \ln \dfrac{\exp(\mu_1^T \Sigma^{-1} x - \frac{1}{2}\mu_1^T \Sigma^{-1} \mu_1)}{\exp(\mu_2^T \Sigma^{-1} x - \frac{1}{2}\mu_2^T \Sigma^{-1} \mu_2)} + \ln \dfrac{p(C_1)}{p(C_2)}$

    $\implies a = w^T x + w_0$ where,

    $w = \Sigma^{-1}(\mu_1 - \mu_2)$

    $w_0 = -\dfrac{1}{2}\mu_1^T \Sigma^{-1} \mu_1 + \dfrac{1}{2}\mu_2^T \Sigma^{-1} \mu_2 + \ln \dfrac{p(C_1)}{p(C_2)}$
  - $\implies p(C_1|x) = \sigma(w^T x + w_0)$

    $\Rightarrow$ Find parameters in Gaussian model using Maximal Likelihood Sulotion
  - Generalize to K-classes:

    $a_k(x) = \ln[p(x|C_k)p(C_k)],\ p(C_k|x) = \dfrac{\exp(a_k)}{\sum_i \exp(a_i)}$

    $\Rightarrow a_k(x) = w_k^T x + w_{k0}$, where $w_k = \Sigma^{-1}\mu_k;\ w_{k0} = -\dfrac{1}{2}\mu_k^T \Sigma^{-1} \mu_k + p(C_k)$
- Assume directly $p(C_k|x) = \sigma(w^T x + w_0)$ (2-classes) (Discriminative model)
  - Assume directly: $p(C_1|w, x) = \sigma(w^T x),\ x_0 = 1$

$\Rightarrow$ less parameters to fit (compared to Gaussian)

- Likelihood function:

  $$p(\boldsymbol{t}|w, X) = \prod_{n=1}^{N} p_n^{t_n}(1-p_n)^{1-t_n}, \text{ where, } p_n = p(C_1|x_n),\ t_n \text{ is the class of } x_n$$

  Define error function :

  $$E(w) = -\ln(Likelihood) = -\sum_{n=1}^{N}[t_n \ln p_n + (1-t_n)\ln(1-p_n)]$$

  $$\Rightarrow \nabla E(w) = \sum_{n=1}^{N}(p_n - t_n)x_n$$

- Find Posterior $p(w|\boldsymbol{t})$:

  $\textbf{Likelihood}$ is product of sigmoid

  Conjugate $\textbf{Prior}$ for "sigmoid distribution" is unknown

  $\Rightarrow$ Assume $\textbf{Prior}\ p(w) = \mathcal{N}(w|m_0, S_0)$

  $$\Rightarrow \ln p(w|\boldsymbol{t}) \propto -\frac{1}{2}(w-m_0)^T S_0^{-1}(w-m_0) + \sum_{n=1}^{N}[t_n \ln p_n + (1-t_n)\ln(1-p_n)]$$

  find w_{MAP}, calculate $S_N = -\nabla\nabla \ln p(w|\boldsymbol{t}) = S_o^{-1} + \sum_{n=1}^{N} p_n(1-p_n)\phi_n\phi_n^T$

  $\Rightarrow p(w|\boldsymbol{t}) \simeq \mathcal{N}(w|w_{MAP}, S_N)$, via Laplace Approximation

- Laplace Approximation:
  - Fit a guassian to $p(z)$ at its **mode** (mode of $p(z)$: point where $p'(z) = 0$)
  - Assume $p(z) = \frac{1}{Z}f(z)$, with normalization $Z = \int f(z)dz$

    Taylor expansion of $\ln f(z)$ at z_0: $\ln f(z) \simeq \ln f(z_0) - \frac{1}{2}A(z-z_0)^2$,

    where $f'(z_0) = 0, A = -\frac{d^2}{dz^2}\ln f(z)|_{z=z_0}$

    Take its exponentiating: $f(z) \simeq f(z_0)\exp\{-\frac{A}{2}(z-z_0)^2\}$

    $$\Rightarrow \text{Laplace Approximation} = (\frac{A}{2\pi})^{1/2}\exp\{-\frac{A}{2}(z-z_0)^2\}, \text{ where } A = \frac{1}{\sigma^2}$$

  - Requirement:

    $f(z)$ differentiable to find a critical point

    $f''(z_0) < 0$ to have a maximum & so that $\nabla\nabla \ln f(z_0) = A > 0$ as $A = \frac{1}{\sigma^2}$

  - In Vector Space: approximate $p(z)$ for $z \in \mathcal{R}^M$

    Assume $p(z) = \frac{1}{Z}f(z)$

    Taylor expansion: $\ln f(z) \simeq \ln f(z_0) - \frac{1}{2}(z-z_0)^T A(z-z_0)$,

    Hessian $A = -\nabla\nabla \ln f(z)|_{z=z_0}$

    $$\begin{aligned}\Rightarrow \text{Laplace approximation} &= \frac{|A|^{1/2}}{(2\pi^{M/2})}\exp\{-\frac{1}{2}(z-z_0)^T A(z-z_0)\} \\ &= \mathcal{N}(z|z_0, A^{-1})\end{aligned}$$

- Gradient descent:
  - Hypothesis function: $h_\theta(x) = \sigma(x\theta) = \frac{1}{1+e^{-x\theta}}$
  - Cost function:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}[-y^i \ln(h_\theta(x^i)) - (1-y^i)\ln(1-h_\theta(x^i))] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

- Update rule: $\forall\theta_j \in \theta, \theta_j := \theta_j - \alpha\frac{\partial J(\theta)}{\partial\theta_j}$, $\frac{\partial J(\theta)}{\partial\theta_j} = \frac{1}{m}\sum_{i=1}^{m}[(h_\theta(x^i) - y^i)x_j^i] + \frac{\lambda}{m}\theta_j$

Neural Networks

1. Goal:
   - Learn basis function(s) rather than manually choose
     - Generalized Linear Model: $h_\theta(x) = f(\phi(x)\theta),\ \text{where } \phi \text{ is basis function, f}(\cdot) \text{ is activation function}$
     - Let $\phi(x)$ depend on parameters, and then adjust parameters together with θ
   - Complex non-linear hypotheses
     - hard to train: too many features with their polynomial terms (computer vision)
2. Problem in Neural Networks:
   - Weight-space Symmetries (n units in one hiden layer)
     - Occure when activation function for hidden units has symmetries $\Rightarrow \mathcal{O}(2^n)$

       $\text{Ex. } \arctan(-a) = -\arctan(a)\text{: changing sign of all input \& output units has the same mapping}$
     - In one hidden layer, exchange unit with each other (along with their inout & output weights).

       Mapping stay the same $\Rightarrow \mathcal{O}(n!)$
     - $\Rightarrow \mathcal{O}(n!2^n)$ overall weight-space symmetries
   - Error function not convex
     - At least $\mathcal{O}(n!2^n)$ critical points ($\nabla E(w) = 0,\ \text{where } E(w) \text{ is Error function}$)
     - $\Rightarrow$ Can be expensive in finding even local minimum with Gradient Descent

       $\mathcal{O}(n^3)$, using Laplace Approximation
   - Deep networks result in gradient vanishing
     - But functions that can be compactly represented by a depth k architecture might require an exponential number of computational nodes using a depth k-1 architecture
     - Potential solution: Xavier initialization, ReLU Activations $h(x) = \max(x, 0)$
3. Representation:
   - input layer (layer 1), hidden layer(s), output layer
     - each layer has $x_0 = 1$ as bias unit $(\text{except for output layer})$
   - s_l = num of units in layer l
   - z_i^j = output of unit i in layer j ($\text{represents parametrised basis}$)
   - Θ^j = matrix of weights controlling function mapping from layer j to layer $j+1$
     - with $s_j$units in layer j and s_{j+1} units in layer $j+1$, Θ^j is of dimension $s_{j+1} \times (s_j + 1)$
4. Forward propagation:
   - Activation $a^{j+1} = \Theta^j \cdot [z_0^j, z_1^j, \ldots, z_{s_j}^j]^T,\ z_0^j = 1$
   - Activation function $h(\cdot)$ to get output of a hidden units
     - Need to be nonlinear
     - **ReLU activation**: $h(x) = \max(x, 0) \Rightarrow$ help avoiding saturation (gradient vanishing)

- $z^{j+1} = h(a^{j+1}) = [z_1^{j+1}, z_2^{j+1}, \ldots, z_{s_{j+1}}^{j+1}]^T$

5. **Backward propagation**:
   - $J(\Theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}[y_k^i log((h_\Theta(x^i))_k) + (1-y_k^i)log(1-(h_\Theta(x^i))_k)] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_{l+1}}\sum_{j=1}^{s_l}(\Theta_{i,j}^l)^2$
     - sum the cost for all k categories
     - s_l: num of units in lth layer; Θ^l: $s_{l+1} \times (s_i + 1)$ => in $\Theta_{i,j},\ i \neq 0$
     - can introduce regularised error $\tilde{J} = J(\Theta) + \frac{\lambda}{2}\Theta^T\Theta$
   - $\frac{\partial J(\Theta)}{\partial \Theta_{i,j}^l} = \frac{\partial J(\Theta)}{\partial a_i^{l+1}}\frac{\partial a_i^{l+1}}{\partial \Theta_{i,j}^l} = \delta_i^{l+1} z_j^l$, $\textbf{where } \delta_i^l = \frac{\partial J(\Theta)}{\partial a_i^l}$
     - $\delta_i^{l+1} = h'(a_i^{l+1})\sum_{k=1}^{s_{l+1}}\delta_k^{l+2}\Theta_{ki}^{l+1}$
     - Intuition: $\textbf{Gradient=output error } \delta \times \textbf{ input associatd with } \Theta_{i,j}$
   - Efficiency: $\mathcal{O}(n), \textbf{where } n \textbf{ is the num of parameters in all } \Theta$
     - whereas numerical differentiation ($\frac{J(\Theta_{i,j}+\epsilon)-J(\Theta_{i,j}-\epsilon)}{2\epsilon}$) need $\mathcal{O}(n^2)$
   - **Algorithm**:
     - $\forall l, i, j,\ \Delta_{ij}^l := 0$
     - For $i = 1$ to m

       $a^1 := x^i$, compute $a^l,\ l = 2, \ldots, L$ (forward)

       $\delta^L := a^L - y^i$, compute $\delta^{L-1}, \ldots, \delta^2$ (backward, a, y^i are column vectors)

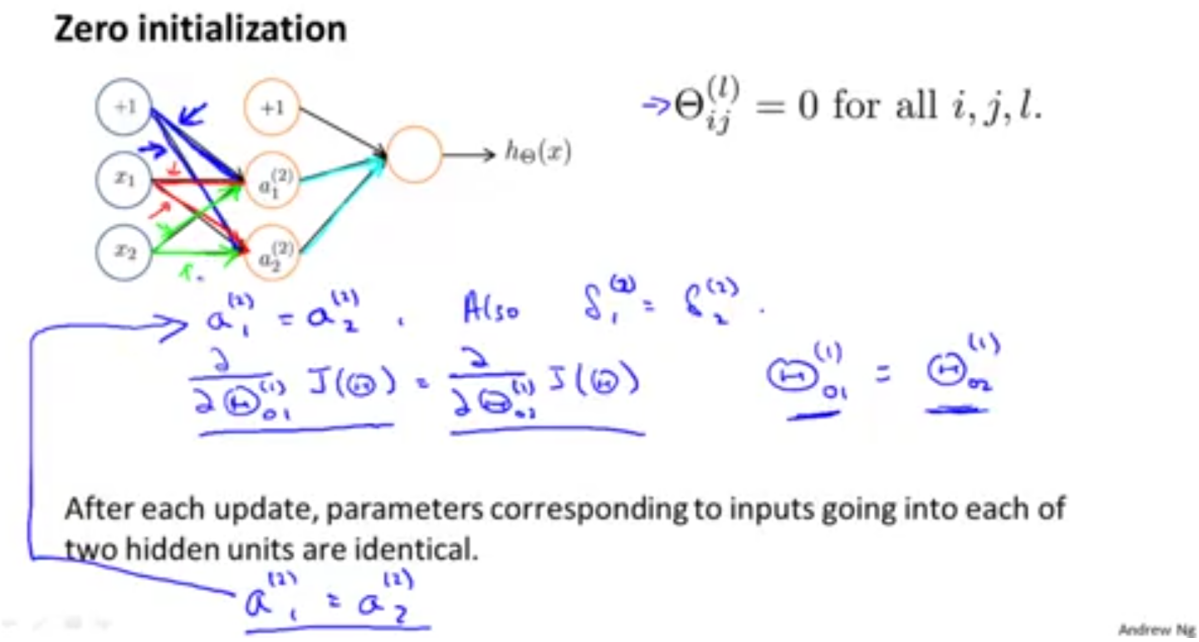
       $\Delta_{ij}^l += a_j^l \delta_i^{l+1}$ (vectorization: $\Delta^l += \delta^{l+1}(a^l)^T$)

       $D_{i,j}^l = \begin{cases} \frac{1}{m}(\Delta_{i,j}^l + \lambda\Theta_{i,j}^l) & j \neq 0, \\ \frac{1}{m}(\Delta_{i,j}^l & j = 0. \end{cases}$
     - $D_{i,j}^l = \frac{\partial J(\Theta)}{\partial \Theta_{i,j}^l}$

6. Gradient checking: $D_{i,j} \approx \frac{J(\Theta_{i,j}+\epsilon)-J(\Theta_{i,j}-\epsilon)}{2\epsilon}$
   - turn checking off when running the backpropagation algorithm

7. Initialization:

- Symmetry problem:



  - so that units in the same layer are computing the same features
- Random initialization: $\Theta_{i,j}^{l} := random[-\epsilon, \epsilon]$
  - Default choice: $\epsilon = \dfrac{\sqrt{6}}{\sqrt{s_l + s_{l+1}}}$, s_l is num of units in layer l
  - Matlab/Octave: $\Theta = rand(m, n) * 2\epsilon - \epsilon$

    $rand(m, n)$ => m*n matrix with randome num between 0,1
- **Xavier Initialization**:
  - Avoid saturation $\textbf{(gradient vanishing)}$
  - For each layer, assume: Input $x_i \sim \mathcal{N}(0, 1)$, weights $w_i \sim \mathcal{N}(0, \sigma^2)$ and activation $a = \sum x_i w_i$

    $\Rightarrow Var(a) = \mathbb{E}(a - \mathbb{E}(a))^2 = \mathbb{E}(a^2) = \sum \mathbb{E}((x_i w_i)^2) = \sum \mathbb{E}(x_i^2)\mathbb{E}(w_i^2) = m\sigma^2$

    $\Rightarrow$ $\textbf{Set } \sigma^2 = \frac{1}{m}$

    $\textbf{, where } m \textbf{ is total num of input } x_i$

8. Network architecture
   - Num of input units: dimension of eatures in x^i
   - Num of output units: num of classes (in Classification)
   - Num of hidden layer:
     - default: 1
     - if >1: same num of units in each hidden layer
   - Num of units in layer: comparable to num of layer
9. Bayesian Neural Networks
   - Assumption:
     - $\textbf{Conditional distribution: } p(t|x, w) = \mathcal{N}(t|h(x, w), \beta^{-1})$
     - $\textbf{Prior: } p(w) = \mathcal{N}(w|0, \alpha^{-1}I)$
   - Likelihood:

- $p(\boldsymbol{t}|w) = \prod_{n=1}^{N} \mathcal{N}(t_n|h(x_n, w), \beta^{-1})$

- Posterior:
  - $p(w|\boldsymbol{t}) \propto p(\boldsymbol{t}|w)p(w)$, Non-Guassian because of non-linear $h(x_n, w)$
  - Apply Laplace Approximation:

    $\Rightarrow$ find w_{MAP} for $\ln p(w|\boldsymbol{t}) = -\frac{\beta}{2}\sum_{n=1}^{N}(h(x,w) - t_n)^2 - \frac{\alpha}{2}w^T w + \text{const}$,

    * w_{MAP} is local maximal due to non-convex $\ln p(x|\boldsymbol{t})$, which due to non-linear $h(x, w)$

    $\Rightarrow \boldsymbol{A} = -\nabla\nabla \ln p(w|\boldsymbol{t}) = \alpha I + \beta \boldsymbol{H}$,

    where $\boldsymbol{H}$ is Hessian matrix of sum-of-square error function with respect to w
  - $p(w|\boldsymbol{t}) \simeq \mathcal{N}(w|w_{MAP}, \boldsymbol{A}^{-1})$
- Predictive Distribution:
  - $p(t|x, \boldsymbol{t}) = \int p(t|x, w)p(w|\boldsymbol{t})dw$, still analytically intractable due to non-linear $h(\cdot)$ in $p(t|x, w)$
  - Taylor Expansion: $h(x, w) \simeq h(x, w_{MAP}) + \boldsymbol{g}^T(w - w_{MAP})$, where $\boldsymbol{g} = \nabla h(x, w)|_{w=w_{MAP}}$

    $\Rightarrow \textbf{Conditional distribution: } p(t|x, w) \simeq \mathcal{N}(t|h(x, w_{MAP}) + \boldsymbol{g}^T(w - w_{MAP}), \beta^{-1})$
  - $p(t|x, \boldsymbol{t}) = \mathcal{N}(t|h(x, w_{MAP}), \sigma^2(x)),\ \text{where } \sigma^2(x) = \beta^{-1} + \boldsymbol{g}^T\boldsymbol{A}^{-1}\boldsymbol{g}$

2. Kernel Methods

- Basic Ideas
  - Use all training data to make prediction
- Assumptions:
  - Continuity: Most targets con't change abruptly
  - Similarity: Each traning pair shows something about the posible targets in its neighbourhood

Kernel Density Estimation

- Assumption:
  - data are drawn independently from distribution $p(x)$
- Goal:
  - Estimate $p(x)$ from data

1. Nonparametric Density Estimation – Histogram
   - Procedure:
     - Partition soace into bins of width Δ_i
     - Compute density $p_i = \frac{n_i}{N\Delta_i}$, $\text{where } n_i \text{ is num of sample in bin } i,\ N \text{ is total num}$
   - Pros:
     - Data can be discarded after calculating p_i=> apply to sequential learning
   - Cons:
     - Dependent on bin width Δ_i
     - Discontinuities due to bin edges
     - Exponential scaling with dimension of data (M^D bins for D dimensions, M bins per dimension)
2. Nonparametric Density Estimation – Refined

- Probability of points found In region $\mathcal{R}$: $P = \int_{\mathcal{R}} p(x)dx$
  - In N observation from $p(x)$, K points found inside region $\mathcal{R}$
    $\Rightarrow \text{Prob}(K|N,P) = \frac{N!}{K!(N-K)!}P^K(1-P^{N-K})$
  - with large N: $P \approx \frac{K}{N}$
  - with small $\mathcal{R}$: $P \approx p(x)V, \text{where } V \text{ is volume of } \mathcal{R}$
- Assume $\mathcal{R}$:
  - small enough for $p(x)$ to be constant
  - big enough for enough K
    $\Rightarrow\ p(x) \approx \frac{K}{NV}$
- K-nearest-neighbours density estimation:
  - Fix K : find a small region around x with exactly K points in it
  - $p(x) \approx \frac{K}{NV}$
- Parzen window (kernel density estimation):
  - Fix V: define hypercube region $\mathcal{R}$ around x, with length h
  - Parzen window (kernel function): $k(u) = \begin{cases} 1, & |u_i| \leq \frac{1}{2}\ (\text{threshold}) \\ 0, & \text{otherwise} \end{cases}$
  - num of point inside: $K = \sum_{n=1}^{N} k(\frac{x-x_n}{h})$
  - $p(x) \approx \frac{K}{NV} = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{h^D}k(\frac{x-x_n}{h})$

    Interpretation: sum over N cube centered at each of the x_n
- Remaining Problem:
  - Discontinuitiy due to hybercube (in\out)
  - Choose soomther kernel function

    $\Rightarrow \text{Gaussian kernel: } p(x) = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{(2\pi h^2)^{D/2}}\exp\{-\frac{||x-x_n||^2}{2h^2}\}$

    or any kernel function where $k(u) \geq 0,\ \int k(u)du$=1
  - h controls the trade-off between <u>sensitivity</u> and <u>soomthing</u>

3. **Classification**
   - Role of training data:
     - Prediction is based on linear combination of **kernel functions** evaluated at data
     - Traning data: $\Phi = \begin{bmatrix} \phi(x_1) \\ \phi(x_2) \\ \vdots \\ \phi(x_N) \end{bmatrix}$, $\text{where } \phi \text{ is basis function, } \Phi \text{ is of } N \times M$
   - Intuition: (linear regression)
     - optimal (regularized) $w = (\lambda I + \Phi^T\Phi)^{-1}\Phi^T t$

- $$\begin{aligned}\textbf{prediction: } y(x) = \phi(x)w &= \phi(x)\ (\lambda I + \Phi^T\Phi)^{-1}\Phi^T\boldsymbol{t} \qquad (\lambda I + \Phi^T\Phi)^{-1} = M \times M \\ &= \phi(x)\ \Phi^T(\lambda I + \Phi\Phi^T)^{-1}\boldsymbol{t} \qquad (\lambda I + \Phi\Phi^T)^{-1} = N \times N\end{aligned}$$

  $\Phi\Phi^T$ is considered as **kernel matrix**, which measures similarity of pair of data

  $\textbf{as inner product of 2 points is a measure of similarity (similar to cosine similarity)}$

- Kernel Function:
  - $k(x, x') = \phi(x)\phi(x')^T\textbf{: inner product of two vectors; symmetric}$
- Kernel Matrix $K = \Phi\Phi^T$: Gram matrix over Φ
  - $K_{i,j} = \phi(x_i)\phi(x_j)^T = k(x_i, x_j)$
  - positve-semidefinite (半正定):

    $$\begin{aligned}v^TKv &= v^T\Phi\Phi^Tv \qquad \Leftrightarrow \forall\lambda \textbf{ is eigenvalue of } K, \lambda \geq 0 \quad \Leftrightarrow \sum_{i,j=1}^{N} K_{i,j} \cdot v_iv_j \geq 0 \\ &= (v^T\Phi)(v^T\Phi)^T \\ &= \|v^T\Phi\|^2 \geq 0 \\ &\textbf{where } v \textbf{ is eigenvector of } K\end{aligned}$$

  - symmetry: $K^T = K$
- Dual Representation:
  - Cost function: $J(w) = \frac{1}{2}(\boldsymbol{t} - \Phi w)^T(\boldsymbol{t} - \Phi w) + \frac{\lambda}{2}w^Tw$
  - optimal (regularized) $w = (\lambda I + \Phi^T\Phi)^{-1}\Phi^T\boldsymbol{t}$

    $\Rightarrow w = \frac{1}{\lambda}\Phi^T(\boldsymbol{t} - \Phi w)$

    $\Rightarrow \textbf{let } w = \Phi^Ta, \textbf{ where } a = -\frac{1}{\lambda}(\Phi w - \boldsymbol{t})$

    $\Rightarrow J(a) = \frac{1}{2}a^T\Phi\Phi^T\Phi\Phi^Ta - a^T\Phi\Phi^T\boldsymbol{t} + \frac{1}{2}\boldsymbol{t}^T\boldsymbol{t} + \frac{\lambda}{2}a^T\Phi\Phi^Ta$

    $\Rightarrow J(a) = \dfrac{1}{2}a^TKKa - a^TK\boldsymbol{t} + \dfrac{1}{2}\boldsymbol{t}^T\boldsymbol{t} + \dfrac{\lambda}{2}a^TKa$

    $\Rightarrow \textbf{let } \mathcal{D}J(a)(\xi) = \xi^TKKa - \xi^TK\boldsymbol{t} + \lambda\xi^TKa = 0$

    $\Rightarrow K(Ka - \boldsymbol{t} + \lambda a) = 0, \textbf{ while } K \neq 0 \textbf{ in most cases}$

    $\Rightarrow a = (K + \lambda I)^{-1}\boldsymbol{t} \textbf{ minimizes } J(a)$

  - Gain:

    K is $N \times N$ matrix $\Rightarrow$ can use infinite num of basis functions: $\phi(x) = [\phi_0(x), \ldots, \phi_\infty(x)]$

    Can construct new valid kernel from given ones (don't need to think of basis functions)

    Can define kernel over graphs, sets, documnets...
- Prediction:
  - $$\begin{aligned}y(x) &= \phi(x)w = \phi(x)\Phi^Ta \\ &= k(x)(K + \lambda I)^{-1}\boldsymbol{t} \qquad \textbf{, where } k(x) = \phi(x)\Phi,\ k_n(x) = k(x, x_n) = \phi(x)\phi(x_n)^T\end{aligned}$$
  - $y(x)$ is based on **kernel function** $k(x_{\text{new}}, x_{\text{traning}})$, which measures **similarity** between a pair of data
- Kernel Construction:
  - Kernels from Basis Function: $k(x, x') = \phi(x)\phi(x')^T$

    Ex. Polynomial, Guassian, Logistic Sigmoid
  - Kernel from Theory:

    choose $k(x, x')$ from intuition and then check validation

K is Gram matrix $(K = \Phi\Phi^T) \Leftrightarrow K$ is positive semidefinite and symmetric:

$K = \Phi\Phi^T \Rightarrow \ldots \Rightarrow K$ positive semidefinite and symmetric

K is positive semidefinite and symmetric :

(Real) Symmetric: $K = Q\Lambda Q^T$, where Λ is diagonal matrix with $\Lambda_{ii} = \lambda_i$, Q 正交

positive semidefinite: $\lambda \geq 0 \Rightarrow \Lambda = DD$, where D is diagnal matrix with $D_{ii} = \sqrt{\lambda_i}$

$\Rightarrow K = \Phi\Phi^T$, where $\Phi = QD$

$\Rightarrow k(x, x')$ is valid $\Leftrightarrow k(x, x') = \phi(x)\phi(x')^T$

$\Leftrightarrow$ corresponding K is positive semidefinite and symmetric

- Kernel from other kernels: operations that remain positive semidefinite and symmetric
- Kernel over Graphs, Sets, Texts: only need a valid **similarity measure** $k(x, x')$
- Kernels from Probabilistic Generative Models:

  Given $p(x)$ define $k(x, x') = p(x)p(x')$, x and x' are similar if they are both of high probabilities

  Include a weighting function $p(i)$: $k(x, x') = \sum_i p(x|i)p(x'|i)p(i)$

  For a continuous variable z: $k(x, x') = \int p(x|z)p(x'|z)p(z)dz$ (Hidden Markov Model)

Support Vector Machine (SVM)

- Do perform feature scaling **before** using Guassian Kernel
- Sparse Kernel Machine
  - prediction evaluated on a subset of data (support vectors)
  - most data can be discarded after training
- Choose the decision boundary with Maximum margin ⇒ Maximum Margin Classifier
  - Margin: smallest distance between decision boudary and any of the sample
- **Maximum Margin Solution** on linear sperable data from 2 classes:
  - Assume:

    linear sperable data from two classes

    using linear discriminant $y(x) = \phi(x)w + b$

    predict based on sign of $y(x) : t \in \{-1, +1\}$
  - Distance of x_n from boundary:

    $\text{dist} = \frac{|y(x_n)|}{\|w\|} = \frac{t_n(w^T\phi(x_n) + b)}{\|w\|}$ (for perfect classification $\forall i, t_i y(x_i) > 0$)
  - Goal: maximize the smallest distance to samples in both classes
    - $\arg\max_{w,b}\{\frac{1}{\|w\|}\min_n[t_n(w^T\phi(x_n) + b)]\}$, s.t. $t_n y(x_n) > 0$

      $\Leftrightarrow \arg\max_{w,b,\gamma}\{\frac{\gamma}{\|w\|}\}$ s.t. $\forall n \in [1, N], t_n(\phi(x_n)w + b) \geq \gamma$, and then resacle by γ
  - Canonical representation: $\arg\max_{x,b}\{\frac{1}{\|w\|}\}$, s.t. $\forall n, t_n(\phi(x_n)w + b) \geq 1$

    $\Leftrightarrow \arg\min_{w,b}\{\frac{1}{2}\|w\|^2\}$ s.t. $\forall n, t_n(\phi(x_n)w + b) - 1 \geq 0$
  - From KKT conditions:

$$\begin{cases} w^T = \sum_{n=1}^{N} a_n t_n \phi(x_n) & ① \\ \sum_{n=1}^{N} a_n t_n = 0 & ② \\ \forall n,\ a_n[t_n(\phi(x_n)w+b)-1]=0 & ③ \\ \forall n,\ t_n(\phi(x_n)w+b)-1 \geq 0 & ④ \\ \forall n,\ a_n \geq 0 & ⑤ \end{cases}$$

- Lagrangian:

  $$L(w,b,a) = \frac{1}{2}\|w\|^2 - \sum_{n=1}^{N} a_n\{t_n[\phi(x_n)w+b]-1\},\ a_n \text{ is Lagrange multiplier}$$

  $$(\text{minimize } L \text{ with respect to } w, b; \text{ maximize } L \text{ with respect to } a)$$

  Eliminate w, b using KKT①② $+$ dual representation:

  $$L(a) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n,m=1}^{N} a_n a_m t_n t_m k(x_n, x_m) \text{ s.t. } \forall n, \sum_{n=1}^{N} a_n t_n = 0 \ \wedge\ a_n \geq 0$$

  Equivlant problem: $\arg\max_a L(a)$ s.t. $\forall n,\ \sum_{n=1}^{N} a_n t_n = 0 \ \wedge\ a_n \geq 0$

  $$\Rightarrow \text{Maximize Obect: } L(a) = a - \frac{1}{2}a^T Q a, \text{ s.t. } a^T t = 0 \wedge \forall n, a_n \geq 0$$
  $$\text{where } Q_{nm} = t_n t_m k(x_n, x_m)$$

- Support Vectors:

  $$y(x) = \phi(x)w + b = \sum_{n=1}^{N} a_n t_n k(x, x_n) + b, \text{ using ①}$$

  $$\text{From KKT③④⑤ :}$$
  $$\text{if } a_n = 0 : \text{inactive constraint}, \qquad x_n \text{ not contributing to } y(x)$$
  $$\text{iff } a_n > 0 : \text{active constraint } (t_n y(x_n) = 1), \qquad x_n \text{ contributing to } y(x)$$
  $$\Rightarrow \text{Support Vectors: } \forall x_n,\ t_n y(x_n) = 1 \text{ (all } x_n \text{ on the margin boundaries)}$$

- Solving bias b:

  $$\forall \text{support vector } x_n,\ t_n y(x_n) = 1$$
  $$\Rightarrow t_n(\sum_{m\in\mathcal{S}} a_m t_m k(x_n, x_m) + b) = 1, \text{ where } \mathcal{S} \text{ is set of support vectors}$$

  For more numerical stability: use $t_n^2 = 1$ and average over all support vectors

  $$\Rightarrow b = \frac{1}{N_{\mathcal{S}}}\sum_{n\in\mathcal{S}}(t_n - \sum_{m\in\mathcal{S}} a_m t_m k(x_n, x_m))$$

- Prediction:
  - $y(x) = \sum_{n\in\mathcal{S}} a_n t_n k(x, x_n) + b$, where $\mathcal{S}$ is set of support vectors

    $\Rightarrow$ kernels are only evaluated at a subset of training data (support vectors)

- **Soft Margin SVM** on non-seperable data:
  - Basic idea:

    Allow for overlapping class distributions - allow some points on "wrong side"

    Incrasing penalty linearly with distance

  - Slack variable $\forall n, \xi_n \geq 0$:

    $$\xi_n \begin{cases} = 0, & \text{correctly classified (on margin boundary or beyond)} \\ < 1, & \text{in correct margin} \\ = 1, & \text{on decision boundary} \\ > 1, & \text{misclassified} \end{cases}$$

- Constraint:

  $t_n y(x_n) \geq 1 - \xi_n$ (instead of $t_n y(x_n) \geq 1$)

  $\Rightarrow$ In all $\begin{cases} \xi_n \geq 0 \\ \xi_n \geq 1 - t_n y(x_n) \end{cases}$

- Goal: $\min_{w,\xi} C \sum_{n=1}^{N} \xi_n + \frac{1}{2}\|w\|^2$ s.t. $\xi_n \geq 0 \wedge t_n y(x_n) - 1 + \xi_n \geq 0$,

  where $C > 0$ controls the trade-off between slack variable penalty and the margin

  $\sum_n \xi_n$ as upper bound on the num of misclassified points ($\xi_n > 1$ if misclassified)

  $\Rightarrow C$ is analogous to (the inverse of) a regularization coefficient

  with $C \to \infty$, turn into SVM for linear sperable data

- From KKT Condition:

$$\begin{cases} w = \sum_{n=1}^{N} a_n t_n \phi(x_n) & \Leftrightarrow \frac{\partial L}{\partial w} = 0 \\ \sum_{n=1}^{N} a_n t_n = 0 & \Leftrightarrow \frac{\partial L}{\partial b} = 0 \\ a_n = C - \mu_n & \Leftrightarrow \frac{\partial L}{\partial \xi_n} = 0 \end{cases}, \quad \begin{cases} a_n(t_n y(x_n) - 1 + \xi_n) = 0 \\ t_n y(x_n) - 1 + \xi_n \geq 0 \\ a_n \geq 0 \\ \mu_n \xi_n = 0 \\ \xi_n \geq 0 \\ \mu_n \geq 0 \end{cases}$$

- Lagrangian:

  $L(w, b, a, \xi, \mu) = \frac{1}{2}\|w\|^2 + C\sum_{n=1}^{N} \xi_n - \sum_{n=1}^{N} a_n\{t_n y(x_n) - 1 + \xi_n\} - \sum_{n=1}^{N} \mu_n \xi_n$, where a_n, μ_n Lagrange multipliers

  Eliminate w, b, ξ_n, using KKT + dual representation:

  $$L(a) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n,m=1}^{N} a_n a_m t_n t_m k(x_n, x_m) \text{ s.t. } \forall n, \boldsymbol{a_n} \in [\mathbf{0}, \boldsymbol{C}] \ \wedge \ \sum_{n=1}^{N} a_n t_n = 0$$

  Equivlant problem: $\arg\max_a L(a)$, with ***box constraint***: $a_n \in [0, C]$ and constraint $\sum_{n=1}^{N} a_n t_n = 0$

- Support Vectors:

  $\forall n \in \mathcal{S}, a_n > 0 \Rightarrow t_n y(x_n) = 1 - \xi_n$

  $\begin{cases} \text{if } a_n < C \Rightarrow \mu_n > 0, \text{ hence } \xi_n = 0 & \Rightarrow t_n y(x_n) = 1 \text{ (point on margin boundary)} \\ \text{if } a_n = C \Rightarrow \mu_0 = 0, \text{ hence no constraint on } \xi_n & \Rightarrow \text{ point either correctly classfied or misclassified} \end{cases}$

- Solving bias b:

  $t_n(\sum_{m \in \mathcal{M}} a_m t_m k(x_n, x_m) + b) = 1$, where $\mathcal{M}$ is set of support vectors with $a < C$

  For numerical stability:

  $$b = \frac{1}{N_{\mathcal{M}}} \sum_{n \in \mathcal{M}} (t_n - \sum_{m \in \mathcal{S}} a_m t_m k(x_n, x_m))$$

- Prediction:

  $$y(x) = \sum_{n \in \mathcal{S}} a_n t_n k(x, x_n) + b$$

- Limitations:

  Complexity parameter C: to be found (e.g. via cross-validation)

  Problematic in scaling to k-classes

  Kernel matrix need to be **positive definite**

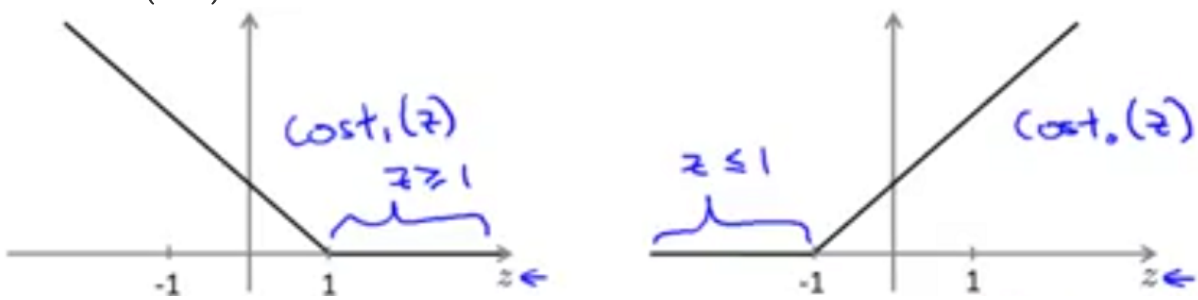
Prediction is linear combinations of kernel functions

Output is decision, not posterior probability

1. Hypothesis function: $h_\theta(x) = \begin{cases} 1 & \theta^T x \geq 0, \\ 0 & \text{otherwise.} \end{cases}$

2. Cost Function:

   - $J(\theta) = C\sum_{i=1}^{m}[y^i cost_1(\theta^T x^i) + (1-y^i)cost_0(\theta^T x^i)] + \frac{1}{2}\sum_{i=1}^{n}\theta_j^2$
   - Curve of $cost(\theta^T x^i)$



3. SVM Decision Boundary (Large margin classification)

   - $min_\theta \frac{1}{2}\sum_{j=1}^{n}\theta_j^2 = min_\theta \frac{1}{2}||\theta||^2$
     - vector θ vertical to the decision boundary
     - samll margin => small projection to θ => large θ due to $h_\theta(x)$ => contradiction **margin is actually the projection from data to θ**

4. **Kernel**:

   - Similarity function (a way to measure distance between examples):
     - $f_i = similarity(x, l^i) = e^{-\frac{||x_j - l_j^i||^2}{2\sigma^2}}$ (Guassian Kernel)
   - Hypothesis function: $h_\theta(x) = \begin{cases} 1 & \theta^T f \geq 0, \\ 0 & \text{otherwise.} \end{cases}$
     - $f = [f_0, f_1, f_2, \ldots, f_n],\ f_i = similarity(x, l^i), f_0 = 1$
   - Choosing landmark l:
     - Apply feature scaling
     - Initialize m landmarks to training examples
     - map x -> f, with $f_0 = 1$
     - Traning SVM: solving minimization poblem of $J(\theta)$ using (f, y) instead of (x, y))

5. SVM parameters:

   - $C(= \frac{1}{\lambda})$:
     - Large C: lower bias, higher variance (small λ)
     - Small C: higher bias, lower variance (large λ)
   - σ^2 in Guassian Kernel:
     - Large σ^2: features f_i vary more smoothly, higher bias, lower variance
     - Small σ^2: features f_i vary less smoothly, lower bias, higher variance
   - Similarity function (Kernel):
     - Linear kernel (no kernel): $f = x$
     - Guassian kernel

        - Polynomial kernel: $f = similarity(x, l) = (x^T l + a)^b, a, b \in N, x, l$ (usually) $\in R^+$
        - String kernel, Chi-square kernel, Histogram kernel, Intersection kernel...
6. Comparison between logistic regression: n - num of features, m - num of traning data
    - $n >> m$ ($n \geq 10m$): Logistic regression or SVM with Linear kernel

      (large n => linear function is enough & small m => not enough data to train complicated hypothesis function)
    - small n with intermidiate m ($m \approx 10n$): SVM with Guassian kernel
    - $n << m$ ($10n \leq m$): **add more features** and then Logistic regression or SVM with Linear kernel

  (Logistic regression and SVM with Linear kernel are similar)

Relevance Vector Machine

1. Assumption:
    - $p(t|x, w, \beta) = \mathcal{N}(t|y(x), \beta^{-1})$
    - $y(x) = \sum_{n=1}^{N} w_n k(x, x_n) + b = w^T \Phi \phi(x) + b$
    - $\text{Prior: } p(w|\alpha) = \prod_{i=1}^{M} \mathcal{N}(w_i|0, \alpha_i^{-1}), \text{ with different precision } \alpha_i$
2. Maximize Posterior:
    - Iteratively calculates α_i, β from training data by optimising a nonconvex function
    - When maximizing with respect to a_i, a significant portion of $a,\ a_i \to \infty$

      $\Rightarrow$ corresponding w_i not contributin to prediction
3. Advantages $vs.$ Disadvantages:
    - Adv:
        - Parameters governing complexity and noise are determined from data

          (compared to SVM, which needs cross-validation)
        - Num of relevance vetors is much more smaller than suppot vectors in SVM

          $\Rightarrow$ faster prediction
        - Scaling well to k classes
    - Dis:
        - Non-convex function $\Rightarrow$ may result in local minimum
        - Longer training time than SVM

3. Diagnose Machine Learning

- Size of data set
- value of λ
- Num of feature
- Degree of polynomial
- Build a quick implmentation

Evaluating hypothesis

1. Data set => training set (60%), cross validation set (20%), test set (20%)
    - randomly split

2. Cross Validation:
    - estimation of the generalization error, inaccurate because:
        - finite trainning set
        - finite corss validation set
    - S-fold Cross Validation:
        - devide whole set of data into S sets
        - for $i \in (1, S)$

          choose the ith set as cross validation (or test set in some cases)

          rest of the sets are called traning set

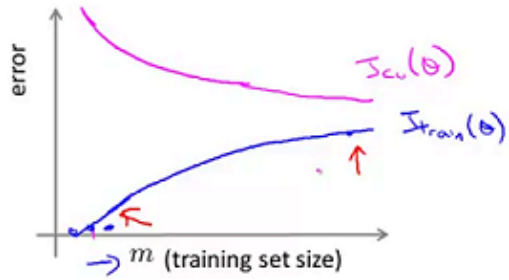
          run the procedure (mentioned later) on the traning set

          estimate generalization in CV set
3. Test set error
    - Linear regression: $J_{test}(\theta) = \frac{1}{2m_{test}} \sum_{i=1}^{m_{test}} (h_\theta(x_{test}^i) - y_{test}^i)^2$
    - Logistic regression: $J_{test}(\theta) = -\frac{1}{m_{test}} \sum_{i=1}^{m_{test}} (y_{test}^i log\ h_\theta(x_{test}^i) + (1 - y_{test}^i) log\ h_\theta(x_{test}^i))$
    - Misclassfication error: $J_{test}(\theta) = \frac{1}{m_{test}} \sum_{i=1}^{m_{test}} err(h_\theta(x_{test}^i),\ y_{test}^i)$
        - $err(h_\theta(x),\ y) = \begin{cases} 1 & \text{if } h_\theta(x) \geq 0.5, y = 0 \\ & \text{or } h_\theta(x) < 0.5, y = 1 \\ 0 & \text{otherwise} \end{cases}$
4. Choosing procedure:
    - Minimize traning error $J_{train}(\theta)$
    - Select a model with lowest $J_{cv}(\theta)$
    - Estimate generalization error as $J_{test}(\theta)$

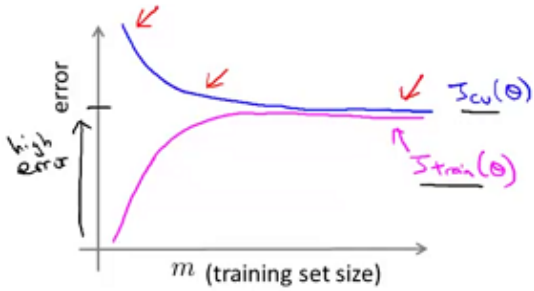
Bias / Variance

1. Problems:
    - High bias: underfitting
        - high $J_{train}(\theta)$ and high $J_{cv}(\theta)$
    - High variance: overfitting
        - low $J_{train}(\theta)$ but high $J_{cv}(\theta)$
2. Interaction with regularization:
    - Improper λ:
        - large λ => high bias
        - small λ => high variance
    - Choosing λ:
        - try $\lambda = 0, 0.01, 0.02, 0.04, \ldots, 10$
        - select the model with lowest $J_{cv}(\theta)$ without regularization term
3. Interaction with training set size:

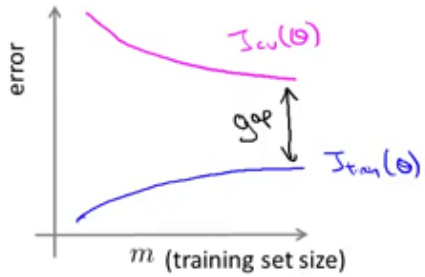
- Normal Learning curve:



- Learning curve with high bias:
  - where getting more training data **doesn't** help



- Learning curve with high variance:
  - where getting more training data **helps**



4. Ways to fix:
   - High bias:
     - more features / more polunomial terms of features
     - decreasing λ

- High variance:
  - larger data set
  - fewer features
  - increasing λ
- **In neural network:**
  - High bias => larger neural networks (more hidden layers / more units in one layer)
  - High variance => smaller neural networks

    Larger network with regularization (λ) is more powerful

Error Analysis

1. Procedure:
   - Algorithm (trained) misclassifies n data in cross validation set
   - Classify these n data and rank them

- Maybe more features are found

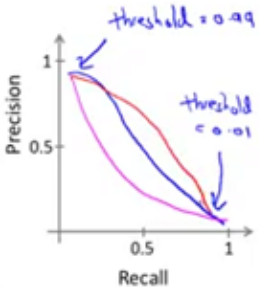
2. Feature selection => Numerical evaluation
   - => test algorithm with / without this feature on **CV set** (compare error rate)

Skewed classes

1. Precision / Recall
   -

	Actual 1	**Actual 0**
Predicted 1	True positive	False positive
Predicted 0	False negative	True negative

   - **Precision** = $\dfrac{\text{True positive}}{\#\text{Predicted positive}} = \dfrac{\text{True positive}}{\text{True pos} + \text{False pos}}$
   - **Recall** = $\dfrac{\text{True positive}}{\#\text{Actual positive}} = \dfrac{\text{True positive}}{\text{True pos} + \text{False neg}}$
2. Evaluation with precision/recall
   - Predict 1 if $h_\theta(x) \geq \epsilon$, 0 if $h_\theta(x) < \epsilon$
     - larger ϵ => higher precision, lower recall (more confident)
     - smaller ϵ => lower pecision, higher recall (avoid missing)



3. Compare precision/recall num
   - $\mathrm{F}_1\ Score = 2\dfrac{PR}{P+R}$, P as precision, R as recall
     - higher better, on cross validation set
4. High precision & high recall:
   - **large num of features** (low bias) **+ large sets of data** (low variance)

4. Latent Variable Models

- Kullback-Leibler divergence
  - Distance between two distributions $p(x)$ and $q(x)$
    - $\text{discret: } KL(q\|p) = \sum_x q(x) \ln \dfrac{q(x)}{p(x)} = -\sum_x q(x) \ln \dfrac{p(x)}{q(x)}$
    - $\text{continuous: } KL(q\|p) = \int q(x) \ln \dfrac{q(x)}{p(x)} dx = -\int q(x) \ln \dfrac{p(x)}{q(x)} dx$
  - Properties
    - $KL(q\|p) \geq 0$
    - not symmetric: $KL(q\|p) \neq KL(p\|q)$
    - $KL(q\|p) = 0 \Leftrightarrow q = p$

        - invariant under parameter tansformations

- **General Expectation Maximization Algorithm**
    - Goal:
        - Find maximum likelihood solution for models with latent variables
    - Variables:
        - X: observed variables
        - Z: latent variables
        - θ: model parameters
    - Log Likelihood:
        - $\ln p(X|\theta) = \ln\{\sum_Z p(X, Z|\theta)\}$

          $\Rightarrow$ problem: $\ln$ out of sum $\Rightarrow$ no clean math solution

          only $\{X\}$ observed $\Rightarrow$ cannot maximize $p(X, Z)$ directly, but have $p(Z|X, \theta)$
        - Introduce arbitrary $q(Z)$ and then sum over Z, $\text{where } q(Z) \text{ is distribution} \Rightarrow \sum_Z q(Z) = 1$

          $$\Rightarrow \sum_Z q(Z) \ln p(X, Z|\theta) = \sum_Z q(Z) \ln p(Z|X, \theta) + \ln p(X|\theta)$$

          $$\Rightarrow \ln p(X|\theta) = \underbrace{\sum_Z q(Z) \ln \frac{p(X, Z|\theta)}{q(Z)}}_{\mathcal{L}(q,\theta)} + \underbrace{(-\sum_Z q(Z) \ln \frac{p(Z|X, \theta)}{q(Z)})}_{KL(q\|p)},$$

          $\text{where } KL(q\|p) \text{ is Kullback-Leibler divergence}$
    - **Algorithm**:
        - E step: fix θ^{old}, maximize lower bound $\mathcal{L}(q, \theta^{\text{old}})$

          $\Rightarrow \mathcal{L}(q, \theta^{\text{old}}) \text{ maximize when } KL(q\|p) = 0$ ($\ln p(X|\theta)$ $\text{independent from } q(\cdot)$)

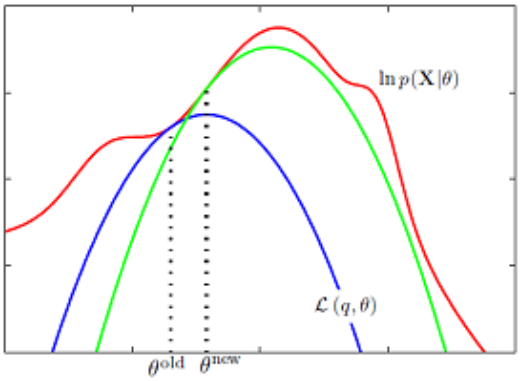
          $\Rightarrow q(Z) = p(Z|X, \theta^{\text{old}})$

          $\text{where } p(Z|X, \theta^{\text{old}}) \text{ is posterior}$
        - M step: fix $q(Z) = p(Z|X, \theta^{\text{old}})$, maximize lower bound $\mathcal{L}(q^{\text{old}}, \theta)$

          $$\Rightarrow \theta = \arg\max_\theta \ln p(X|\theta)$$
          $$= \arg\max_\theta \mathcal{L}(q^{\text{old}}, \theta)$$

          $\text{taking gradient w.r.t. } \theta \text{ will contain } q(Z) = p(Z|X, \theta^{\text{old}})$
    - Convergence:
        - Start by initializing θ
        - After $\text{M step}: \mathcal{L}(q^{\text{old}}, \theta) > \mathcal{L}(q^{\text{old}}, \theta^{\text{old}}) \ \wedge \ \theta \neq \theta^{\text{old}}$ unless converged

          $\Rightarrow KL(q\|p) > 0$
          $\Rightarrow \text{E step}$
    - Understanding:
        - $q(\cdot)$ **:** model to fit $\ln p(X|\theta)$
        - θ **:** model parameter for $\mathcal{L}(q, \theta)$
        - blue curve is one iteration before green curve



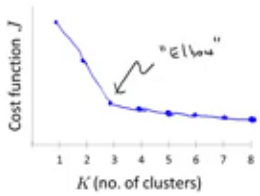
Clustering - Naive K-means

1. Input:
   - K (num of clusters)
   - Training set $\{x^1, x^2, \ldots, x^m\}$, $x^i \in R^n$ (drop $x_0 = 1$ convention), $K < m$
2. Cost function:
   - $J(c^1, \ldots, c^m, \mu_1, \ldots, \mu_K) = \frac{1}{m}\sum_{i=1}^{m}||x^i - \mu_{c^i}||^2$
     - $c^i = \text{index of cluster } \{1,2,\ldots,\text{K}\} \text{ assigned to } x^i, \text{K} = \text{num of cluster}$
     - $\mu_k = \text{cluster centroid } k$, $\mu_k \in R^n$
     - $\mu_{c^i} = \text{cluster centroid assigned to } x^i$
   - With $\text{1-of-}K$ coding scheme: $J = \frac{1}{m}\sum_{i=1}^{m}(\sum_{k=1}^{K} r_k^i \|x^i - \mu_k\|)$
     - $r^i = \text{1-of-K vector to denote cluster assigned to } x^i, r^i \in \{0,1\}^K$
     - $\mu_k = \text{cluster centroid } k$, $\mu_k \in R^n$
3. Goal:
   - Find r^i, μ_k to minimize J
     - Problem: cyclic dependence between $r^i, \mu_k \Rightarrow \text{EM algorithm}$
4. Algorithm:
   - Initialize K cluster centroid $\mu_1, \mu_2, \ldots, \mu_K \in R^n$
   - Repeat
     - for $i = 1$ to m $(\text{cluster assignment - E step})$

       $c^i := \text{closest cluster centroid to } x^i$
     - for $k = 1$ to K $(\text{move centroid - M step})$

       $\mu_k := \text{mean of points assigned to cluster } k$
5. Initialization: $(\text{crucial for convergence})$
   - **Randomly** pick K **distinct** examples and set $\mu_1, \ldots, \mu_K$ to them
     - random algorithm may need to repeat multiple times to avoid local optimum
6. Understanding:
   - Cluster assignment: adjust $c^1, \ldots, c^m$ to minimize J, holding $\mu_1, \ldots, \mu_K$
   - Move centroid: adjust $\mu_1, \ldots \mu_K$ to minimize J, holding $c^1, \ldots, c^m$
   - J is **not** possible to increase if algorithm is implemented correctly
7. Parameter K:

- Elbow method: plot $J - K$ graph:



- Consider downstream purpose

8. Possible Improvement:
   - Use triangle inequality $\Rightarrow$ Triangle K-means
   - Use Non-Euclidean distance $\Rightarrow$ Kernel K-means (K-medoids)
   - Build K-D tree

Mixture Models - Generative K-means

1. Gaussian Mixture Model Distribution:
   - $p(x) = \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k)$
     - $\int p(x)dx = 1 \Rightarrow \sum_{k=1}^{K} \pi_k = 1$
     - Need to find π, μ, Σ
2. Inroduce Latent Variable $\mathbf{z}$:
   - $\mathbf{z} \in \{0,1\}^K$: 1-of-K coding scheme to denote the distribution from which x is drawn
   - Distribtion:
     - $p(\mathbf{z}_k = 1) = \pi_k$

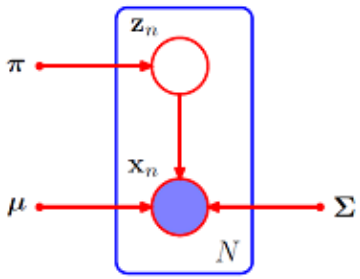
       $\Rightarrow p(\mathbf{z}) = \prod_{k=1}^{K} \pi_k^{z_k}$
     - $p(x|\mathbf{z}_k = 1) = \mathcal{N}(x|\mu_k, \Sigma_k)$

       $\Rightarrow p(x|\mathbf{z}) = \prod_{k=1}^{K} \mathcal{N}(x|\mu_k, \Sigma_k)^{z_k}$
     - $p(x, \mathbf{z}) = \prod_{k=1}^{K} [\pi_k \mathcal{N}(x|\mu_k, \Sigma_k)]^{z_k}$
     - $p(z_k = 1|x) = \dfrac{p(z_k = 1)p(x|z_k = 1)}{\sum_{i=1}^{K} p(z_i = 1)p(x|z_i = 1)} = \dfrac{\pi_k \mathcal{N}(x|\mu_k, \Sigma_k)}{\sum_{i=1}^{K} \pi_i \mathcal{N}(x|\mu_i, \Sigma_i)}$

       denoted as $\gamma(z_k)$: responsibility of z_k to explain the observed x
   - Recover $p(x)$: (to assert that we are on the right track)
     - $p(x) = \sum_{\mathbf{z}} p(\mathbf{z})p(x|\mathbf{z})$

       $= \sum_{\mathbf{z}} \prod_{k=1}^{K} \pi_k^{z_k} \prod_{k=1}^{K} \mathcal{N}(x|\mu_k, \Sigma_k)^{z_k}$

       $= \sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k)$
   - N data drawn i.i.d.



3. Likelihood:
   - $\ln p(X|\pi,\mu,\Sigma) = \sum_{n=1}^{N} \ln[\sum_{k=1}^{K} \pi_k \mathcal{N}(x|\mu_k,\Sigma_k)]$
     - $P(X) = \sum_{Z} P(X,Z)$, where full combination $\sum_{Z} \prod_{\text{each combination of } z_{1-n}} = \prod_{X} \sum_{\text{all possibility of each } x}$
   - Goal:
     - Find π, μ, Σ that maximize ($\ln$) likelihood
   - Problem:
     - $\ln$ does **not** apply on $\mathcal{N} \Rightarrow$ does **not** have neat direct math solution
     - $\Rightarrow$ apply $\mathbf{EM}$
4. EM algorithm:
   - Critical points:
     - $$\frac{\partial}{\partial \mu_k} \ln p = 0 \Rightarrow 0 = \sum_{n=1}^{N} \underbrace{\frac{\pi_k \mathcal{N}(x_n|\mu_k,\Sigma_k)}{\sum_{i=1}^{K} \pi_i \mathcal{N}(x_n|\mu_i,\Sigma_i)}}_{\gamma(z_k)} \Sigma^{-1}(x_n - \mu_k)$$
       $$\Rightarrow \mu_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_k) x_n$$
     - $$\frac{\partial}{\partial \Sigma_k} \ln p = 0$$
       $$\Rightarrow \Sigma_k = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_k)(x_n - \mu_k)(x_n - \mu_k)^T$$
     - $$\frac{\partial}{\partial \pi_k} \ln p = 0$$
       $$\Rightarrow \pi_k = \frac{N_k}{N}$$
       $\text{where } N_k \text{ is the effective num of data point assigned to } k : N_k = \sum_{n=1}^{N} \gamma(z_k)$
   - $\mathbf{E\ step}$: evaluate $q(Z) = \gamma(z_k)$ using current parameters μ, Σ, π $\quad$ $\text{(soft assign point to Gaussian)}$
     - $$\gamma(z_k) = \frac{\pi_k \mathcal{N}(x|\mu_k,\Sigma_k)}{\sum_{i=1}^{K} \pi_i \mathcal{N}(x|\mu_i,\Sigma_i)}$$
   - $\mathbf{M\ step}$: re-estimate parametes $\theta = \{\mu, \Sigma, \pi\}$ using current $\gamma(z_k)$ $\quad$ $\text{(maximize likelihood)}$
     - $$\mu_k^{\text{new}} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) x_n$$
     - $$\Sigma_k^{\text{new}} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk})(x_n - \mu_k^{\text{new}})(x_n - \mu_k^{\text{new}})^T$$

- $\pi_k^{\text{new}} = \frac{N_k}{N}$

5. Relation to naive K-means:
   - Setting $\Sigma_k = \epsilon, \epsilon \to 0$
     - $\gamma(z_{nk})$ with smallest $\|x_n - \mu_k\|^2$ goes to 0 most slowly
     - $\Rightarrow \gamma(z_{nk}) = \begin{cases} 1 & \text{if } \|x_n - \mu_k\| < \|x_n - \mu_i\| \forall i \neq k \\ 0 & \text{otherwise} \end{cases}$
   - Reduce to naive K-means when $\Sigma_k = \epsilon, \epsilon \to 0$
     - $\lim_{\epsilon \to 0} \gamma(z_{nk}) = r_{nk}$ $\Rightarrow$ becomes hard assignment
6. Broad view:
   - Generalized to other distribution:
     - Mixture of Bernoulli Distributions
     - ...
   - Reducing to Naive K-means:
     - $\gamma(z_{nk}) = \begin{cases} 1 & \text{if } \|x_n - \mu_k\| < \|x_n - \mu_i\| \quad \forall i \neq k \\ 0 & \text{otherwise} \end{cases}$

Principal Component Analysis (PCA)

1. Motivation:
   - Data compression (reduce highly related features)
   - Data visualization
2. Assumption:
   - Gaussian distributions for both the latent and observed variables
3. Two Equivalent Definition of PCA:
   - Linear projection of data onto lower dimensional linear space (principal subspace) such that:

     $\Rightarrow$ **variance of projected data is maximized**
     $\Rightarrow$ **distortion error from projection is minimized**
4. Maximum Variance Formulation
   - Goal:
     - project data from D dimension to M while maximizing the variance of projected data
   - Eigenvalues λ of covariance matrix S express the variance of data set X in direction of corresponding eigenvectors
   - Projection Vectors:
     - $U = (u_1, \ldots, u_M),\ \text{where } \forall i \in \{1, .., M\}, u_i \in \mathbb{R}^D \text{ s.t. } u_i^T u_i = 1 \text{ (only consider direction)}$
   - Projected Data:
     - $\text{Mean} = \bar{x}^T U, \text{ where } \bar{x} = \frac{1}{N} \sum_{i=1}^{N} x^i$
     - $\text{Variance} = tr\{U^T S U\}, \text{ where } S = \sum_{i=1}^{N} (x^i - \bar{x})(x^i - \bar{x})^T \text{ (outer product)}$
   - Lagrangian to maximize **Variance**:
     - $L(U, \lambda) = tr\{U^T S U\} + tr\{(I - U^T U)\lambda\}$

$\text{constraint } u_i^T u_i = 1 \text{ to prevent } u_i \to +\infty$

$\text{For each } u_i \in U,\ \dfrac{\partial}{\partial u_i} L = 2Su_i - 2\lambda_i u_i = 0$

$\Rightarrow Su_i = \lambda_i u_i$

$\Rightarrow U \text{ consists of eigenvectors corresponding to the first M large eigenvalue of } S$

$(S \text{ symmetric} \Rightarrow U \text{ orthogonal})$

5. Minimum Error Formulation:
   - Introduce Orthogonal Basis Vector for D dimension:
     - $U = (u_1, \ldots, u_D)$
   - Data representation:
     - Original: $x^n = \sum_{i=1}^{D} \alpha_i^n u_i$
     - Projected: $\widetilde{x^n} = \sum_{i=1}^{M} z_i^n u_i + \sum_{i=M+1}^{D} b_i u_i$

       $(z_1^n, \ldots, z_M^n) \text{ is different for different } x^n,\ (b_{M+1}, \ldots, b_D) \text{ is the same for all } x^n$
   - Cost function: $J = \dfrac{1}{N} \sum_{n=1}^{N} \|x^n - \widetilde{x^n}\|^2, \text{where } \widetilde{x^n} = \sum_{i=1}^{M} z_i^n u_i + \sum_{i=M+1}^{D} b_i u_i$
     - $\text{Let } \begin{cases} \dfrac{\partial}{\partial z_j^n} J = 0 \\ \dfrac{\partial}{\partial b_j} J = 0 \end{cases} \Rightarrow \begin{cases} \dfrac{1}{N} 2(x^n - \widetilde{x^n})^T(-u_j) = \dfrac{2}{N}(z_j - (x^n)^T u_j) = 0 \\ \dfrac{1}{N} \sum_{n=1}^{N} 2(x^n - \widetilde{x^n})^T(-u_j) = \dfrac{2}{N} \sum_{n=1}^{N} (b_j - (x^n)^T u_j) = 0 \end{cases}$

       $\Rightarrow \begin{cases} z_j = (x^n)^T u_j & j \in \{1, \ldots, M\} \\ b_j = \overline{x}^T u_j & j \in \{M+1, \ldots, D\} \end{cases}$

       Noticing $(x^n)^T u_j = (\sum_{i=1}^{D} \alpha_i^n u_i^T) u_j = a_j \Rightarrow a_j = (x^n)^T u_j$

       $\Rightarrow x^n - \widetilde{x^n} = \sum_{i=M+1}^{D} [(x^n - \overline{x})^T u_i] u_i$
     - $$\begin{aligned} \Rightarrow J &= \frac{1}{N} \sum_{n=1}^{N} \left(\sum_{i=M+1}^{D} [(x^n - \overline{x})^T u_i] u_i \right)^T \left(\sum_{i=M+1}^{D} [(x^n - \overline{x})^T u_i] u_i \right) \\ &= \frac{1}{N} \sum_{n=1}^{N} \left(\sum_{i=M+1}^{D} u_i^T ((x^n - \overline{x})^T u_i) \right) \left(\sum_{i=M+1}^{D} ((x^n - \overline{x})^T u_i) u_i \right) \\ &= \frac{1}{N} \sum_{n=1}^{N} \sum_{i=M+1}^{D} u_i^T (x^n - \overline{x})^T u_i u_i^T (x^n - \overline{x}) u_i && u_i \text{ orthogonal to each other} \\ &= \sum_{i=M+1}^{D} u_i^T \left(\frac{1}{N} \sum_{n=1}^{N} (x^n - \overline{x})^T (x^n - \overline{x}) \right) u_i && \|u_i\| = 1 \end{aligned}$$

       $\Rightarrow J = \sum_{i=M+1}^{D} u_i^T S u_i, \text{ where } S = \dfrac{1}{N} \sum_{n=1}^{N} (x^n - \overline{x})^T (x^n - \overline{x})$
   - Lagrangian to Minimize J:
     - $L(u_{M+1}, \ldots, u_D, \lambda_{M+1}, \ldots, \lambda_D) = \sum_{i=M+1}^{D} u_i^T S u_i + \sum_{i=M_1}^{D} \lambda_i (1 - u_i^T u_i)$

       $\text{constraint } \|u_i\| = 1 \text{ to prevent } u_i = 0$

$\text{For each } u_i, \frac{\partial}{\partial u_i} L = 2Su_i - 2\lambda_i u_i = 0$

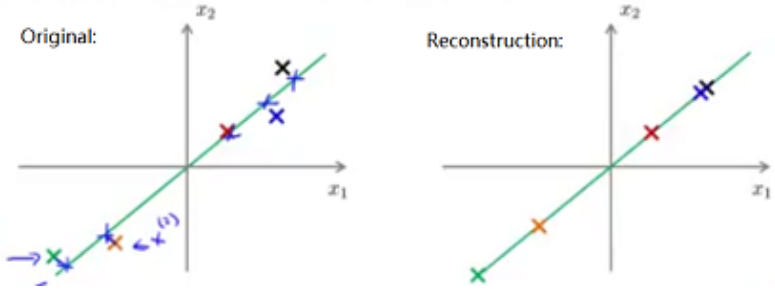
$\Rightarrow Su_i = \lambda_i u_i$

$\Rightarrow \text{To minmize } J, \text{ take eigenvectors with the first } (D-M) \text{ small eigenvalue orthogonal to (out of) subspace}$

$\Leftrightarrow \text{define subspace with eigenvectors with the first } M \text{ large eigenvalue}$

- $$\text{Intuition: } \widetilde{x_n} = \sum_{i=1}^{M}((x^n)^T u_i)u_i + \sum_{i=M+!}^{D}(\overline{x}^T u_i)u_i$$
  $$= \overline{x} + \sum_{i=1}^{M}[(x^n - \overline{x})^T u_i]u_i$$

6. Singular Value Decomposition - SVD:
   - Intorduce matrix $A_{m\times n}$
     - $(A^T A)_{n\times n}$ symmetric matrix (**actually, Gram matrix $\rightarrow$ semi-definite**)
     - **eigenvalue decomposition:**
       $A^T A = VDV^T, V \text{ is normalized } (v_i^T v_i = 1) \text{ with column as eigenvector}$
   - $AV = (Av_1, \ldots, Av_n)_{m\times n}$
     - Let $r(A) = r$
       $\Rightarrow r(A^T A) = r(A) = r$
       $r(AV) = \min\{r(A), r(V)\} = \min\{r, n\} = r$
     - Reduce AV to basis $(Av_1, \ldots, Av_r)$
   - Let $U = (u_1, \ldots, u_r) = (\frac{Av_1}{\sqrt{\lambda_1}}, \ldots, \frac{Av_r}{\sqrt{\lambda_r}})$, $\lambda_i \text{ is } i\text{-thh eigenvalue of } A^T A$
     - Orthogonal: $\forall i \neq j, u_i^T u_j = \frac{1}{\sqrt{\lambda_i \lambda_j}} v_i^T A^T A v_j = \frac{\lambda_j}{\sqrt{\lambda_i \lambda_j}} v_i^T v_j = 0$
     - Unit: $\|u_i\| = \frac{\|Av_i\|}{\sqrt{\lambda_i}} = \frac{\sqrt{<Av_i, Av_i>}}{\sqrt{\lambda_i}} = 1$
       $\Rightarrow U \text{ is standard orthogonal (orthonormal) basis}$
     - $AV = U\Sigma, \text{ where } \Sigma = D^{\frac{1}{2}}$
   - Expand U to orthonormal in $\mathbb{R}^m : (u_i, \ldots, u_m)$
     - Epand corresponding part in Σ with 0
   - $A = U\Sigma V^T, \text{ with singular value in } \Sigma \text{ in decreasing order}$
7. SVD with PCA:
   - X is data matrix in row (**centered - zero mean**)
   - Eigenvectors of corvariance matrix $S = X^T X$ are in $V, \text{ where } X = U\Sigma V^T$
     - When using $S = U\Sigma V^T \Rightarrow U = V \wedge S = V\Sigma V^T$
       reduced to eigenvalue decomposition
     - $S = VDV^T$ **with V orthonormal**:
       Eigenvalues λ of covariance matrix S express the variance of data set X in direction of corresponding eigenvectors
   - Projection:
     - $\widetilde{X} = XV_M, \text{ where } V_M \text{ contains first M-large eigenvectors}$
     - Projection direction is **not** unique
8. Reconstruction (approximate):
   - Data is projected onto k dimension using SVD with $S = U\Sigma V^T$

- $x_{approx} = U_{reduce} \cdot z$, U_{reduce} is n*k matrix, z is k*1 vector
- 

9. Choosing k (num of principal components):
   - choose the **smallest** k making $\frac{J}{V} \leq 0.01$ => 99% of variance is retained
   - $[U, S, V]$ = svd(Sigma) => $\frac{J}{V} = 1 - \frac{\sum_{i=1}^{k} S_{ii}}{\sum_{i=1}^{n} S_{ii}}$, S is diagonal matrix

     => check $\frac{J}{V}$ before compress data
10. Data Preprocessing:
    - PCA $vs.$ Normalization:
      - Normalization: Individually normalized but still correlated
      - PCA: create decorrelated data $-$ $\textbf{whitening}$
    - Whitening: $\textbf{projection with normalization}$
      - $S = VDV^T$, where S is Gram matrix over X^T
      - $\forall n, y_n = D^{-\frac{1}{2}} V^T (x^n - \overline{x})$, where $\overline{x}$ is the mean of X

        $\Rightarrow \{y^n\}$ has zero mean

        $$cov(\{y^n\}) = \frac{1}{N} \sum_{n=1}^{N} y_n y_n^T = D^{\frac{-1}{2}} V^T S V D^{\frac{-1}{2}} = I$$
11. Tips for PCA:
    - **Don't** use PCA to prevent overfitting, use regularization instead
    - Try original data before implement PCA
    - Train PCA only on trainning set

Independent Component Analysis (ICA)

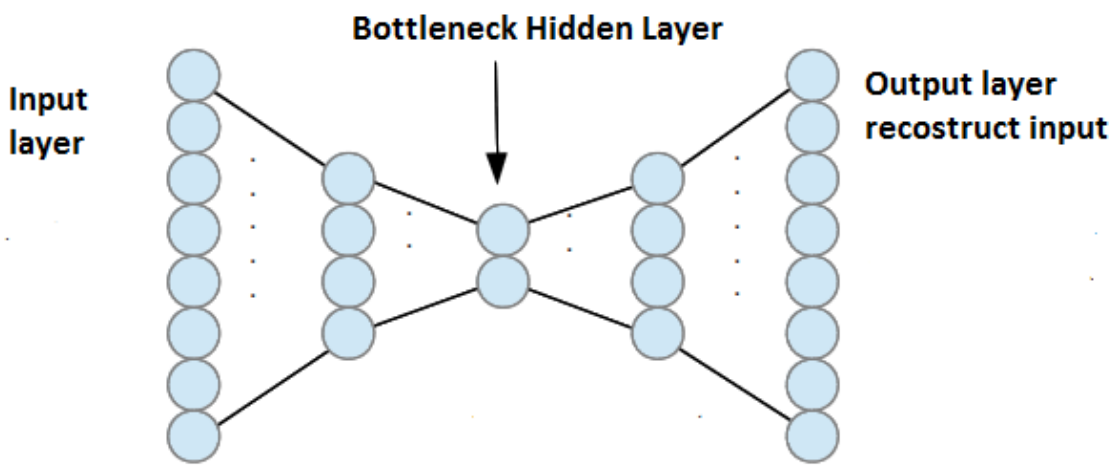
1. Goal:
   - Recover original signals from a mixed observed data
     - Source signal $S \in \mathbb{R}^{N \times K}$; mixing matrix A; Observed data $X = SA$
   - Maximizes statistical independence
     - Find A^{-1} to maximizes independence of columns of S
2. Assumption:
   - At most one signal is Gaussian distributed
   - Ignorde amplitude and order of recovered signals
   - Have at least as many observed mixtures as signals
   - A invertible
3. Independence $vs.$ Uncorrelatedness
   - Independence $\Rightarrow$ Uncorrelatedness
     - $p(x_1, x_2) = p(x_1)p(x_2) \Rightarrow \mathbb{E}(x_1 x_2) - \mathbb{E}(x_1)\mathbb{E}(x_2) = 0$

4. Central Limit Theorem
5. FastICA algorithm

Autoencoder - Neural Network

1. Problem:
   - Functions that can be compactly represented by a depth k architecture might require an exponential number of computational nodes using a depth k-1 architecture
     - $\Rightarrow$ deep network
   - Problem in Deep network:
     - More local minimum, plateaus and saddle points $\Rightarrow$ hard to generalize well
2. Goal:
   - Unsupervised pre-training to find distributed representation
   - Find good lossy compression of data
3. Undercomplete Architecture:
   - Input: data $\boldsymbol{X}$
   - Target output: $\boldsymbol{X}$ (input itself)
   - Reconstruction error: $J = \sum_{n=1}^{N} \ell(x^n, f(c(x^n)))$
     - $c(x)$ **is encoder;** $f(x)$ **is decoder;** $f(c(x))$ **is reconstruction from network**
     - Minimize negative log likelihood: $-\ln P(x|c(x))$

       $\textbf{if } P(x|c(x)) \textbf{ Guassian} \Rightarrow \ell \textbf{ is squared error}$
       $\textbf{if } x_i \textbf{ obey binomial} \Rightarrow -\ln P(x|c(x) = -x_i \ln f_i(c(x)) + (1 - x_i)\ln(1 - f_i(c(x))\ )$

       f_i **is the** i**-th component of decoder**
   - Undercomplete:
     - Small num of hidden units $\Rightarrow$ $c(x)$ as a lossy compression of x
   - Stacking Encoders:
     - 

     - chain non-linear encoder $c_i(x)$

       Compared to PCA: chain linear transformations $\Leftrightarrow$ **single linear transformation**
     - Discard decoding layer $\Rightarrow$ use $z_i = c_i(\cdots c_1(x)\cdots)$ as features for subsequent learning
4. Higher Dimensional Hidden Layer:

- Reason:
  - Want many features wider than input
- Constraint to force compression: $\textbf{(avoid learning identity function - "perfect reconstruction")}$
  - Regularization
  - Early stopping of stochastic gradient descent
  - Adding noise into input
  - Sparsity constraint on encoding $c(x)$

5. Denoising Autoencoder:
   - Idea:
     - Add noise to input & noise free data as target output
   - Goal:
     - preserve information of input
     - undo corruption process
   - Reconstruction likelihood:
     - $P(x|c(\hat{x})),\ \textbf{where } x \textbf{ noise free, } \hat{x} \textbf{ corrupted}$
6. Sparse Representations
   - Idea:
     - Some basis functions for a partcular x may be of minor variation

       $\Rightarrow$ many hidden nodes (basis functions collected in Φ) but a few active for a given code $c(x)$

       $\textbf{use few element from } \Phi \textbf{ to represent } x$
   - ℓ_0 Norm Penalty:
     - $\min_w = \sum_{n=1}^{N} \frac{1}{2} \|x^n - \Phi w^n\|_2^2 + \lambda \|w\|_0$
     - Sparse: small num of non-zero w_i $(\|w\|_0 \textbf{ is small for } x \approx \Phi w)$
     - Problem: $\|w\|_0$ non-convex

       $\Rightarrow$ relax to $\|w\|_1$
   - ℓ_1 Norm Penalty:
     - $\min_w = \sum_{n=1}^{N} \frac{1}{2} \|x^n - \Phi w^n\|_2^2 + \lambda \|w\|_1$
     - Let $K = \Phi\Phi^T$ be Gram matrix over Φ, $(\Phi \textbf{ normalized})$

       $\textbf{Mutual Coherence: } M = M(\Phi) = \max_{i \neq j} |K_{i,j}|$

       $\Rightarrow \textbf{similar rows, then } M \approx 1$
     - Therom:

       $\textbf{For } w^\star \textbf{ minimize } \ell_0 \textbf{ problem}$

       $\textbf{If } \|w^\star\|_0 < \frac{1}{M} \Rightarrow w^\star \textbf{ is the unique sparsest solution}$

       $\textbf{If } \|w^\star\|_0 < \frac{1}{2}(1 + \frac{1}{M}) \Rightarrow \textbf{minimizer of } \ell_1 \textbf{ problem has the same sparsity pattern as } w^\star$

       $\textbf{(has 0 in the same places)}$
   - Probabilistic View of ℓ_1:

- Maximize Joint distribution: $p(x,w) = p(w)p(x|w)$,
  $\textbf{gives rise to } \|\cdot\|_2, \textbf{where } w \textbf{ can be considered as latent variable}$
- Laplace Prior: $p(w_i) = \frac{\lambda}{2}e^{-\lambda|w_i|}$, $\textbf{gives rise to } \ell_1 \textbf{ regularization}$

Restricted Boltzmann Machine

Anomaly Detection

1. Problem to solve:
   - Given dataset $\{x^1, x^2, \ldots, x^m\}$, build density estimation model $p(x)$
   - $p(x^{test} < \epsilon)$ => x^{test} anomaly
2. Hypothesis function:
   - $p(x) = \prod_{i=1}^{n} p(x_i),\ x \in R^n, \forall i \in [1,n],\ x_i \sim N(\mu_i, \sigma_i^2)$
     - $\mu = \frac{1}{m}\sum_{i=1}^{m} x^i,\ \sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(x^i - \mu)^2$
     - assume $x_1, \ldots, x_n$ independent from each other
3. Multivariate Gaussian:
   - $p(x;\mu,\Sigma) = \frac{1}{(2\pi)^{\frac{n}{2}}|\Sigma|^{\frac{1}{2}}} exp(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)),$

     $x \in R^n, \mu \in R^n, \Sigma \in R^{n\times n}$, where Σ is covariance matrix
     - $\mu = \frac{1}{m}\sum_{i=1}^{m} x^i,\ \Sigma = \frac{1}{m}\sum_{i=1}^{m}(x^i - \mu)(x^i - \mu)^T$
     - $x_1, \ldots x_n$ can be correlated but **not** linearly dependent
     - need $m > n\ (m \geq 10n\ suggested)$ or elas Σ non-invertible
4. Algorithm:
   - choose features
   - compute μ, σ
   - compute $p(x)$ for new example, anomaly if $p(x) < \epsilon$
5. Evaluation (real-number):
   - Labeled data into normal/anomalous set

     (okay if some anomalies slip into normal set)
     - training set: unlabeled data from normal set (60%)
     - CV set: labeled data from normal (20%) & anomalous (50%) set
     - test set: labeled data from normal (20%) & anomalous (50%) set
   - Use evaluation metrics (skewed data)
6. When to use:
   - Anomaly detection:
     - Very small num of positive data (0-20 commonly); Large num of negative data
     - Difficult to learn from positive data (not enough data, too many features...)
     - Future anomalies may look nothing like given data
   - Supervised Learning:
     - Larger num of positive & negative data
     - Enough positive data for algorithm to learn

        - Future positive example is likely to be similar to given data
7. Example:
    - Anomaly detection:
    - Fraud detection, Manufacturing, Monitoring machines in data center...
    - Supervised learning:
        - Email spam classification (enough data), Weather prediction (sunny/rainy/etc), Cancer classification...
8. Tips:
    - Non-guassian feature: transformation / using other distribution
    - Choosing features: compare anomaly data with normal data

Recommender System

1. Problem Formulation:
    - $r_{i,j} = 1$ if item i is rated by user j
    - $y_{i,j}$ = rating of item i given by user j
    - θ^j = parameter vector for user j
    - x^i = feature vector for movie i

      => for user j, movie i, ($r_{i,j} = 0$), predict rating $x^i\theta^j$
2. Content Based Recommendations:
    - Treat each user as a seperate linear regression problem with the feature vectors of its rated items as traning set

      Assume features for each items (x^i) are available and known

      => given X estimate Θ
    - Cost Function for θ_j:

      $$J(\theta^j) = \frac{1}{2}\sum_{i:r_{i,j}=1}(x^i\theta^j - y_{i,j})^2 + \frac{\lambda}{2}\sum_{k=1}^{n}(\theta_k^j)^2, \theta^j \in R^{n+1}(\theta_0 \text{ not regularized})$$
    - Cost Function for Θ:

      $$J(\Theta) = \frac{1}{2}\sum_{j=1}^{n_u}\sum_{i:r_{i,j}=1}(x^i\theta^j - y_{i,j})^2 + \frac{\lambda}{2}\sum_{j=1}^{n_u}\sum_{k=1}^{n}(\theta_k^j)^2,$$

      $$\theta^j \in R^{n+1}(\theta_0 \text{ not regularized}),\ n_u \text{ is num of users}$$
    - Update Rule: $\forall\theta_k^j \in \theta^j, \theta_k^j := \theta_k^j - \alpha\frac{\partial J(\Theta)}{\partial\theta_k^j}$,

      $$\frac{\partial J(\Theta)}{\partial\theta_k^j} = \sum_{i:r_{i,j}=1}(x^i\theta^j - y_{i,j})x_k^i + \lambda\theta_k^j, \text{ for } k \neq 0\ (\theta^j \in R^{n+1})$$
3. Collaborative Filtering
    - **Assume preference of each users (θ^j) are available and known**

      => given Θ estimate X
        - Cost Function for x^i: $J(x^i) = \frac{1}{2}\sum_{j:r_{i,j}=1}(x^i\theta^j - y_{i,j})^2 + \frac{\lambda}{2}\sum_{k=1}^{n}(x_k^i)^2$

- Cost Function for X: $J(X) = \frac{1}{2}\sum_{i=1}^{n_m}\sum_{j:r_{i,j}=1}(x^i\theta^j - y_{i,j})^2 + \frac{\lambda}{2}\sum_{i=1}^{n_m}\sum_{k=1}^{n}(x_k^i)^2$

  $$x^j \in R^{n+1}(x_0 \text{ not regularized}),\ n_m \text{ is num of items}$$

- Update Rule: $\forall x_k^i \in x^i, x_k^i := x_k^i - \alpha\frac{\partial J(X)}{\partial x_k^i}$,

  $$\frac{\partial J(X)}{\partial x_k^i} = \sum_{j:r_{i,j}=1}(\theta^j x^i - y_{i,j})\theta_k^j + \lambda x_k^i,\ \text{for } k \neq 0\ (x^i \in R^{n+1})$$

- Basic Idea:
  - Randomly initialize Θ
  - loop:

    Estimate X

    Estimate Θ
- Cost Function:

  $$J(X,\Theta) = \frac{1}{2}\sum_{(i,j):r_{i,j}=1}(x^i\theta^j - y_{i,j})^2 + \frac{\lambda}{2}\sum_{i=1}^{n_m}\sum_{k=1}^{n}(x_k^i)^2 + \frac{\lambda}{2}\sum_{j=1}^{n_u}\sum_{k=1}^{n}(\theta_k^j)^2,\ x \in R^n,\ \theta \in R^n$$

  (the sum term in $J(\Theta)$, $J(X)$, and $J(X,\Theta)$ is the same)
- Update Rule:
  - $\forall x_k^i \in x^i, x_k^i := x_k^i - \alpha\frac{\partial J(X,\Theta)}{\partial x_k^i}$, $\frac{\partial J(X,\Theta)}{\partial x_k^i} = \frac{\partial J(X)}{\partial x_k^i} = \sum_{j:r_{i,j}=1}(\theta^j x^i - y_{i,j})\theta_k^j + \lambda x_k^i,\ x^i \in R^n$
  - $\forall \theta_k^j \in \theta^j, \theta_k^j := \theta_k^j - \alpha\frac{\partial J(X,\Theta)}{\partial \theta_k^j}$, $\frac{\partial J(X,\Theta)}{\partial \theta_k^j} = \frac{\partial J(\Theta)}{\partial \theta_k^j} = \sum_{i:r_{i,j}=1}(\theta^j x^i - y_{i,j})x_k^i + \lambda\theta_k^j,\ \theta^j \in R^n$
- **Algorithm**:
  - Initialize X, Θ to **small random values**

    => for symmetry breaking (similar to random initialization in neural network)

    => so that algorithm learns features $x^1, \ldots, x^{n_m}$ that are different from each other
  - Minimize $J(X,\Theta)$
  - Predict $y_{i,j} = x^i\theta^j$ ($Y = X\Theta$)
- Finding Related Item to Recommend
  - $||x^i - x^j||$ is samll => item i and j is similar
- Mean Normalization:
  - Problem: if user j hasn't rated any movie, $\theta^j = [0, \ldots, 0]$

    => predicted rating of user j on all item $= 0$

    => useless prediction
  - Algorithm (row version):

    compute vector μ, $\forall \mu_i \in \mu, \mu_i = \text{mean of } Y_i, \text{ where } Y_i \text{ is the } i^{th} \text{ row in } Y$

    manipulate Y: $\forall y_{i,j} \in Y \wedge r_{i,j} = 1,\ y_{i,j} -= \mu_i$ => the mean of each row in Y is 0

    predict rating for user j on item $i = x^i\theta^j + \mu_i$
  - For item i with no rating

    => apply column version of mean normalization

    (but user with no rating is generally more important)

5. Large Scale Machine Learning

Gradient Descent with Large Dataset

1. Stochastic Gradient Descent
    - Problem in Big Data:
        - Updating θ becomes computationally expensive in batch gradient decent

- Cost Function:
    - Cost function on single data: $cost(\theta, (x^i, y^i)) = \frac{1}{2}(h_\theta(x^i) - y^i)^2$
    - Overall Cost Function: $J_{train}(\theta) = \dfrac{1}{m}\sum_{i=1}^{m} cost(\theta, (x^i, y^i))$

- Procedure:
    - Randomly shuffle dataset
    - Repeat

      for $i \in [1, m]$

      $$\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} cost(\theta, (x^i, y^i))$$
      $$= \theta_j - \alpha\,(h_\theta(x^i) - y^i) \cdot x_j^i$$
      $$(\text{for } j = 0, \ldots, n)$$
      $$\text{=> make progress with each single data}$$

- Convergence:
    - Wanting θ to converge => slowly decrease α over time (but more parameters)

      $$(\text{E.g } \alpha = \frac{\text{const_1}}{\text{iteration\#} + \text{const_2}})$$
    - Compute $cost(\theta, (x^i, y^i))$ before updating

      For every k update iterations, plot average $cost(\theta, (x^i, y^i))$ over the last k examples
    - Checking curves:

      Increasing k result in smoother line and less noise, but the result is more delayed

      Use smaller learning rate α will generally have slight benefit

      Curve goes up => smaller α

- vs Batch Gradient Descent:
    - use 1 example un each update iteration => make progress earlier => faster
    - Result may not be the optimal but in its neighbourhood

2. Mini-batch Gradient Descent
    - Use b examples in each update iteration
    - vs Batch Gradient Descent:
        - start to make progress earlier => faster
        - Result may not be the optimal but in its neighbourhood
    - vs Stochatistic Gradient Descent:t
        - can partially parallelize computation over b examples => faster under a good vectorized implementation & appropriate b
        - introduce extra parameter b

Online Learning

1. Situation:
    - Has too many data (can be considered as infinite)
    - When data comes in as a continuous stream
    - Can adapt to changing user preference
2. Procedure:
    - Use one example only once (Similar to stochastic gradient decent in this sense

Map-reduce

1. In Batch Gradient Descent:
    - Update rule $\theta_j = \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^i) - y^i) x_j^i$
    - Parallelize the computation of $\sum_{i=1}^{m} (h_\theta(x^i) - y^i) x_j^i$ by dividing the data set into multiple sections
2. Ability to reduce:
    - Contain operation over the whole data set

      (Neural Network can be map-reduced)

6. Building Machine Learning System

- Under the example of Photo OCR (Optical Character Recognition)

Pipeline

1. Break ML system into modules
2. Example:
    - Image -> Text detection -> Character segmentation -> Character recognition
    - Text Detection:
        - Sliding window detection:

          set different sizes of the window (mostly rectangle), for each size:

          take a image patch

          resize the patch into desired size

          run ML algorithm on the small patch

          slide the window by $\mathbf{step_size}$ (eventually through the image)
        - Expansion: expand the related region to create a bigger region
    - Chraracter Segmentation:
        - 1-D sliding window
    - Character Recognition

Getting More Data

1. **Artificial Data Synthesis**
    - Creating New Data: Use available resource and combine them
        - Example (in Character Recognition): Paste different fonts in the randomly chosen backgrounds

- Amplify Data Set: Intorduce distortions to the original data set
  - Need to identify the appropriate distortion
  - Usually adding purely/random/meanless noise

- Prerequisite:
  - Having a low bias/high variance hypothesis is

2. Collect/Label Data Manually
   - Usually a surprise to find how little time it needs to get 10,000 data
     - Caculate the time it needs before decide to/not to collect the data

Ceilling Analysis

1. Aim:
   - Decide which modules might be the best use of time to improve
2. Procedure:
   - Draw a table with 2 column (Component - Accuracy)
     - Component: the modules simulated to be perfect (100% accuracy)
     - Accuracy: the accuracy of the entire system on the test set (define by chosen evaluation matrix)
   -

Perfect Component	Accuracy
none	f
module 1	$f+\epsilon_1$
module $1,2$	$f+\epsilon_1+\epsilon_2$
$\vdots$	$\vdots$
module $1,2,\ldots,n$	$f+\epsilon_1+\ldots+\epsilon_n=100\%$

   - => Improving module x will gain at most ϵ_x improvement in the overall performance
   - Choose the module with most significant ϵ to improve

7. Graphical Models

- Diagrammatic representations of probability distributions:
  - A simple way to visualize the structure of a probabilistic model and can be used to design and motivate new models
  - Insights into the properties of the model (e.g. conditional independence) can be obtained by inspection of the graph.
  - Graphical manipulations to expresse complex computations

Bayesian Networks

1. Structure:
   - Joint Distribution $\rightarrow$ Graph:
     - A **node** for each random variable in the distribution
     - An **directed edge** from a random variable a to all other random variables conditioned on a

- Graph $\rightarrow$ Joint Distribution:
  - $p(\mathbf{x}) = \prod_{k=1}^{K} p(x_k|pa(x_k))$, where $pa(x) =$ parents of x

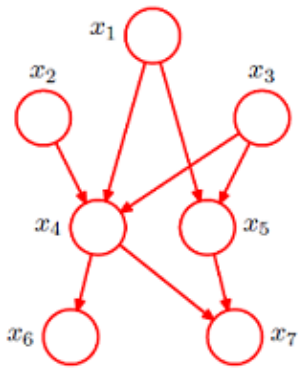
    A **product** of distributions, one for each associated random variable $\textbf{(over all the nodes)}$

    For each node, add all **parents** to its conditioning variables list $\textbf{(conditioned on parents)}$
  - Must be **directed acyclic graph**
  - Normalization proof:

    $\textbf{Start with the node with no outgoing edges, denoted as } x_i$

    $$\begin{aligned}\sum_{\mathbf{x}} p(\mathbf{x}) &= \sum_{x_1,\ldots,x_K} \prod_{k=1}^{K} p(x_k|pa(x_k)) \\ &= \sum_{x_1,\ldots,x_{i-1},x_{i+1}\ldots,x_K} \left(\sum_{x_i} \prod_{k=1}^{K} p(x_k|pa(x_k)) \right) \\ &= \sum_{x_1,\ldots,x_{i-1},x_{i+1}\ldots,x_K} \left(\prod_{k=1 \wedge k \neq i}^{K} p(x_k|pa(x_k)) \underbrace{\sum_{x_i} p(x_i|pa(x_i))}_{=1} \right)\end{aligned}$$

    $\textbf{Repeat until no node left}$
- Example:
  - Distribution: $p(x_1)p(x_2)p(x_3)p(x_4|x_1,x_2,x_3)p(x_5|x_1,x_3)p(x_6|x_4)p(x_7|x_4,x_5)$
  - Graph:

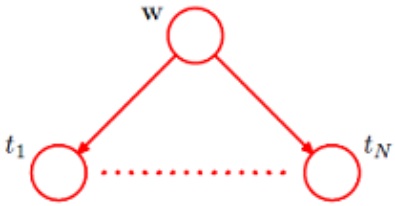
    

    $\Rightarrow$ $\textbf{A fully connected graph for general distribution } p(x_1,\ldots,x_K) = p(x_K|x_1,\ldots,x_{K-1})\cdots p(x_2|x_1)p(x_1)$
    $\textbf{(if reduced to undirected graph)}$

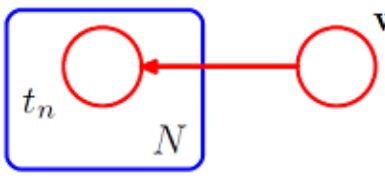
    $\Rightarrow$ $\textbf{Absence of edges carries important indormation}$
- Extension:
  - A node can be vectors or sets of variables

2. Notation:
   - Plate Notation:

- $p(\mathbf{t}, \mathbf{w}) = p(\mathbf{w}) \prod_{n=1}^{N} p(t_n|\mathbf{w})$:

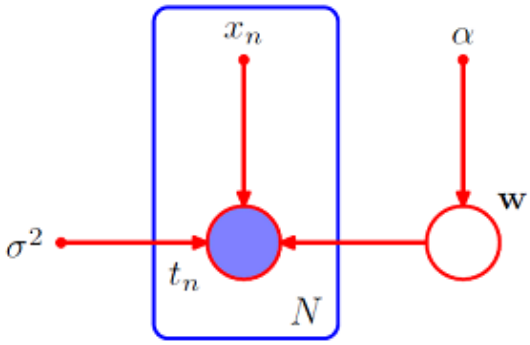


$\Leftrightarrow$

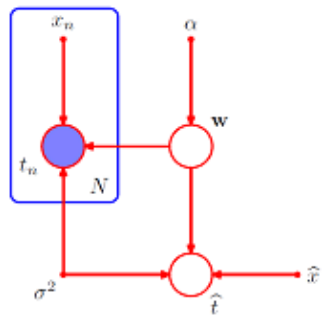


A plate.

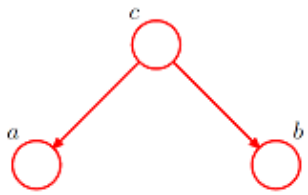
- Parameters of model:
  - Observed random variables = open circles
  - Unobserved random variables = shaded circles
  - Deterministic variable = smaller solid circles
  - $p(\mathbf{t}, \mathbf{w}|\mathbf{X}, \alpha, \sigma^2) = p(\mathbf{w}|\alpha) \prod_{n=1}^{N} p(t_n|\mathbf{w}, x_n, \sigma^2)$:

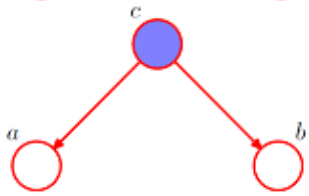


- Prediction:
  - $p(\hat{t} \mid \hat{x}, \mathbf{x}.\mathbf{t}, \alpha, \sigma^2) \propto \int p(\hat{t}, \mathbf{t}, \mathbf{w} \mid \hat{x}, \mathbf{x}, \alpha, \sigma^2) d\mathbf{w}$
  - $p(\hat{t}, \mathbf{t}, \mathbf{w}|\hat{x}, \mathbf{X}, \alpha, \sigma^2) = \left[\prod_{n=1}^{N} p(t_n|x_n, \mathbf{w}, \sigma^2)\right] p(\mathbf{w}|\alpha) p(\hat{t}|\hat{x}, \mathbf{w}, \sigma^2)$:
    (assume $y = y(x, \theta) + \epsilon$; $\hat{x}$ is new data; $\hat{t}$ is prediction)



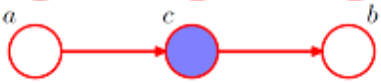
3. Conditional Independence:
   - Definition:
     - $p(a, b|c) = p(a|c)p(b|c) \Leftrightarrow a$ is conditionally independent of b given c
     - same as $p(a|b, c) = p(a|c)$
   - Notation: $a \perp\!\!\!\perp b \mid c$
   - Conditional Independence $vs.$ Independence:
     - Neither of them implies the other
     - $p(a, b|c) = p(a|c)p(b|c)$ reduces to independence when $c = \varnothing$ ($a \perp\!\!\!\perp b \mid \varnothing \Leftrightarrow a, b$ independent)
4. Conditional Independence on graph:
   - The presence of TT-node: $p(a, b, c) = p(a|c)p(b|c)p(c)$

- $a \not\perp\!\!\!\perp b \mid \varnothing$: 

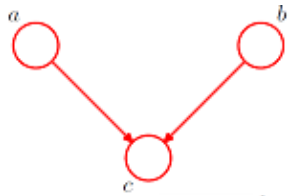
$$p(a,b) = \sum_c p(a|c)p(b|c)p(c) \neq p(a)p(b)$$
- $a \perp\!\!\!\perp b \mid c$: 

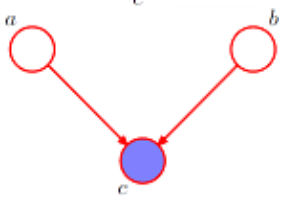
$$p(a,b|c) = \frac{p(a,b,c)}{p(c)} = p(a|c)p(b|c)$$
- Conditioning on c **blocks** the path from a to $b \Rightarrow a, b$ conditionally independent on c

- The presence of **HT-node**: $p(a,b,c) = p(a)p(b|c)p(c|a)$
  - $a \not\perp\!\!\!\perp b \mid \varnothing$:

$$p(a,b) = \sum_c p(a)p(b|c)p(c|a) \neq p(a)p(b)$$
  - $a \perp\!\!\!\perp b \mid c$: 

$$p(a,b|c) = \frac{p(a)p(b|c)p(c|a)}{p(c)} = p(a|c)p(b|c)$$
  - Conditioning on c **blocks** the path from a to $b \Rightarrow a, b$ conditionally independent on c

- The absence of **HH-node**: p(a,b,c)=p(a)p(b)p(c|a,b)
  - $a \perp\!\!\!\perp b \mid \varnothing$: 

$$p(a,b) = \sum_c p(a)p(b)p(c|a,b) = p(a)p(b)$$
  - $a \not\perp\!\!\!\perp b \mid c$: 

$$p(a,b|c) = \frac{p(a)p(b)p(c|a,b)}{p(c)} \neq p(a|c)p(b|c)$$
  - Conditioning on c **unblocks** the path $\Rightarrow a, b$ dependent **(absence of HH-node blocks the path)**
  - Extension: $\exists x$ **is descendant of HH-node** $\Rightarrow$ **conditioning on** x **unblocks the path**

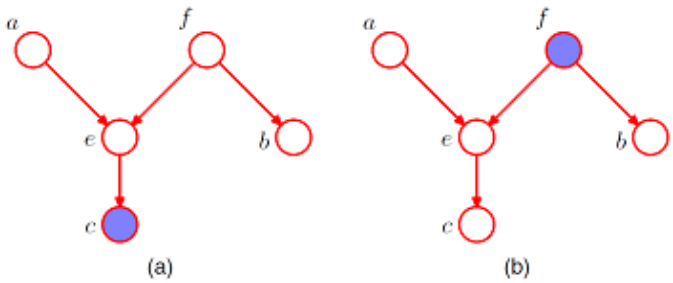
5. Blocked Path Definition:
   - A path which contains
     - an observed **TT-node** or **HT-node**, or
     - an unobserved **HH-node** whose **descendants** are **all** unobserved

       (need to condition on any observed node in the path when considering blocked/unblocked)

6. D-Separation
   - Definition:
     - A, B, C are non-intersecting sets of nodes in a Bayesian network
     - If all possible paths are blocked by nodes in C, then A is D-separated from B by C
     - $\Rightarrow$ Joint distribution of the graph satisfies $A \perp\!\!\!\perp B \mid C$

       (conditioning on observed nodes in C; All nodes outside C considered unobserved)

       (Note: D-separation stands for 'directional' separation)
   - Example:

- Graph: 

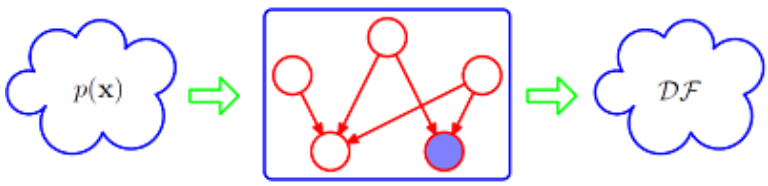
  - path $a \to b$ in (a) not blocked by either e or f

    path $a \to b$ in (b) is blocked by e and f
- Conditional Independence ⇔ Factorisation:
  - Factorisation ⇒ Conditional Independence:

    If probability distribution factorises according to a directed acyclic graph, and disjoint subsets of nodes A, B, C satisfy that A is D-separated from B by C ⇒ Distribution satisfies $A \not\perp\!\!\!\perp B \mid C$
  - Conditional Independence ⇒ Factorisation:

    If probability distribution satisfies conditional independence statements implied by D-separation over a directed graph ⇒ it can factorise according to the graph

7. Understanding:
   - Conditional Independence ⇒ Factorisation:
     - provides factorised computational model $p(x)$ from Conditional Independence statements given by domain experts
   - All Conditional Independence statements can be shown as D-separation in a DAG
     - But there are sets of statements **cannot** be satisfied by any Bayesian Network
   - Bayesian Network as filter:
     - Only let $p(x)$ satifying factorisation property through
     - Only let $p(x)$ respecting conditional independence $(\text{implied by D-separation})$ through



Markov Random Fields (MRFs)

- Undirected graphical model
- Different conditional independence statements than Bayesian networks

1. Conditional Independence
   - Graph Separation:
     - In undirected graph G, with A, B, C disjoint subsets of nodes, if every path from A to B includes at least one node from C, then C is said to separate A from B in G
   - In undirected graph G, with A, B, C disjoint subsets of nodes:
     - $A \perp\!\!\!\perp B \mid C$ iff C separates A from B in G
2. Probability Distribution:
   - A MRFs is a **set** of probability distributions such that, for each distribution:

    - there exisits an undirected graph G with disjoint subsets of nodes A, B, C
    - in which $A \perp\!\!\!\perp B \mid C$ whenever C seperates A from B in G
  - "MRF is such a graph" means:
    - MRF represents set of the probability distributions
    - whose conditional independence are given by the graph

3. Factorisation in MRFs:
  - 2 not directly linked nodes conditionally independent on all other nodes
    - $x_i \perp\!\!\!\perp x_j \mid (\mathbf{x} - \{x_i, x_j\}),\ \text{where } x_i, x_j \text{ not on the same edge}$
  - Cliques:
    - complete sub-graph
    - $\Rightarrow$ Maximal clique: a clique that is not a subset of other clique
  - Factorisation with (maximal) cliques:
    - For maixmal clique $C \in \mathcal{C}$ in undirected graph G ,$\mathbf{x}_C$ are nodes in C
    - $\Rightarrow \text{Probability distribution } p(x) \text{ factorises with respect to } G \text{ if } p(x) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_C(\mathbf{x}_C)$

      $\text{where } \mathcal{C} \text{ is the set of maximal cliques of } G\text{, potential functions } \psi_C(\mathbf{x}_C) \geq 0,\ Z = \sum_{\mathbf{x}} p(\mathbf{x}) \text{ for normalization}$
  - Conditional Independence $\Leftrightarrow$ Factorisation:
    - Factorisation $\Rightarrow$ Conditional Independence:

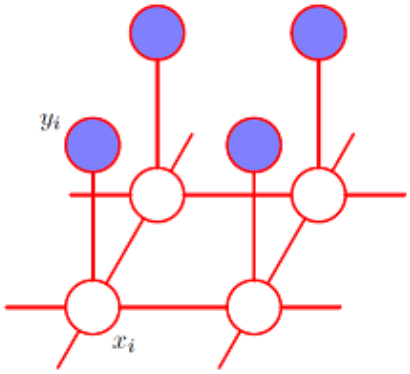
      Probability distribution factorises according to undirected graph G; A, B, C are disjoint subsets of nodes such that C separates A from B in G

      $\Rightarrow$ In distribution: $A \perp\!\!\!\perp B \mid C$
    - Conditional Independence $\Rightarrow$ Factorisation: $\text{(Hammersley-Clifford Theorem)}$

      Strictly positive probability distribution $\forall x, p(x) > 0$ satisfies conditional independence statements implied by graph separation over a particular undirected graph G

      $\Rightarrow$ It factorises according to G
  - Factorisation in **Strictly Positive** Potential Functions:
    - Let potential function $\psi_C(\mathbf{x}_C) = e^{-E(\mathbf{x}_C)},\ E(\mathbf{x}_C) \text{ called energy function}$
    - $\Rightarrow p(\mathbf{x}) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_C(\mathbf{x}_C) = \frac{1}{Z} e^{-\sum_{C \in \mathcal{C}} E(\mathbf{x}_C)}$

4. Example of Image Denoising:
  - Problem & Goal:
    - An unknown noise-free image described by binary pixels $x_i \in \{-1, 1\}$ is added with noise
    - Noise: Randomly flip sign of each pixel with small probability and denoted by $y_i \in \{-1, 1\}$
    - $\Rightarrow$ Recover the noise free image
  - Prior Knowledge from problem:
    - Noise level is small $\Rightarrow$ strong correlation between original x_i and observed y_i
    - Neighbouring pixels x_i, x_j are strongly correlated
  - MRF Capturing Prior:

- 

  - Cliques $\{x_i, y_i\}$:

    Energy function: $E(x_i, y_i) = -\eta x_i y_i, \eta > 0 \Rightarrow \textbf{Potential function: } \psi = \mathbf{e}^{\eta x_i y_i}$

    $\Rightarrow$ Favors equal signs for x_i, y_i
  - Cliques of neighbouring $\{x_i, x_j\}$:

    Energy function: $E(x_i, x_j) = -\beta x_i x_j, \beta > 0 \Rightarrow \textbf{Potential function: } \psi = e^{\beta x_i x_j}$

    $\Rightarrow$ Favors equal signs for x_i, x_j
  - Fact that one kind of pixels might appears more often

    $\Rightarrow$ adding term hx_i

    $\textbf{Reason: Potential function is an arbitrary, non-negative function over maximal cliques}$

    $\Rightarrow \textbf{allowed to multiply it by any nonnegative function of subsets of the clique}$
- Joint Distribution:
  - Total Energy Function: $E(\mathbf{x}, \mathbf{y}) = h\sum_i x_i - \beta\sum_{i,j} x_i x_j - \eta\sum_i x_i y_i$
  - $\Rightarrow p(\mathbf{x}, \mathbf{y}) = \frac{1}{Z}\mathbf{e}^{-E(\mathbf{x},\mathbf{y})}$
- Iterated Conditional Mode: $(\textbf{cannot avoid local minimum})$
  - Fix $\mathbf{y}$ $(\textbf{observed}) \Rightarrow$ implicitly defines $p(\mathbf{x} \mid \mathbf{y})$
  - Initialize $\forall i, x_i := y_i$
  - For each node x_i, evaluate total energy for $x_i = -1, 1$ with other nodes fixed ($\forall j \neq i, x_j = y_j$)
  - Set x_i to the lower energy state $(\textbf{Lower energy} \Rightarrow \textbf{Larger probability})$

5. Comparison with Bayesian Networks:
   - Chain:
     - Bayesian Network:

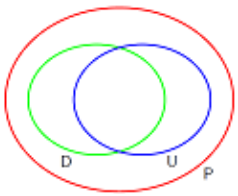
       $p(\mathbf{x}) = p(x_1)p(x_2|x_1)p(x_3|x_2)\cdots p(x_N|x_{N-1})$
     - MRFs:

       $p(\mathbf{x}) = \frac{1}{Z}\psi_{1,2}(x_1, x_2)\psi_{2,3}(x_2, x_3)\cdots\psi_{N-1,N}(x_{N-1}, x_N)$
     - Corresponding term:

$$
\begin{aligned}
\psi_{1,2}(x_1, x_2) &= p(x_1)p(x_2|x_1) \\
\psi_{2,3}(x_2, x_3) &= p(x_3|x_2) \\
&\vdots \\
\psi_{N-1,N}(x_{N-1}, x_N) &= p(x_N|x_{N-1}) \\
Z &= 1 \text{ for MRFs}
\end{aligned}
$$

- Bayesian Networks in MRFs:
  - Moralisation: For every nodes in Bayesian Networks, fully connect its parents in MRFs

    $\textbf{(Simulation instead of expressing the same distribution)}$
  - Expressing Distribution in Graph:

    P : set of probability distribution

    D : set of distribution expressed by $\textbf{directed graph (Bayesian Networks)}$

    U : set of distribution expressed by $\textbf{undirected graph (Markov Random Fields)}$
  - In both: $\textbf{e.g. chain}$
  - Only in Bayesian Networks: $\textbf{HH-node}$
  - Only in MRFs: $\textbf{4-nodes clique}$

6. Common betweeb two graphs:
   - Conditional independence statements are related to concepts of $\textbf{separation}$ in the graph.
   - Simplified algebraic structure $\textbf{(factorisation of p(x))}$ is related to 'local pieces' of the graph
     - BNs: child with its parents
     - MRFs: Cliques

Factor Graphs

1. Definition:
   - Nodes:
     - Variable nodes: variables $\textbf{(circle)}$
     - Factor nodes: function taking its neighbour as variables $\textbf{(solid square)}$
   - Bipartite Graph over set of $\textbf{variable nodes}$ and set of $\textbf{factor nodes}$
2. Expressing Distribution:
   - $p(\mathbf{x}) = \prod_s f_s(\mathbf{x}_s), \text{ where } \mathbf{x}_s \text{ a subset of variables, } f_s(\cdot) \text{ factor connecting to that subset}$
3. Bayesian Networks & MRFs in Factor Graphs:
   - Bayesian Network $\rightarrow$ Factor Graph:
     - Variable node for each node in BNs
     - Factor node for corresponding (conditional) distributions

       $\textbf{(may be several factor graph for one directed graph)}$
   - MRFs $\rightarrow$ Factor Graph:
     - Variable node for each node in MRFs

- Factor node for corresponding maximal clique
- Factor $f_S(\mathbf{x}_S)$ for potential function of maximal clique $\mathbf{x}_S$

  $\text{(may be several factor graph for one undirected graph)}$

4. **Sum-Product Algorithm**:
   - Goal:
     - Calculate marginal $p(x) = \sum_{\mathbf{x}-\{x\}} p(\mathbf{x})$, in P time
   - Math Underneath:
     - Swap $\sum\prod$ to $\prod\sum$ using independence

       $$\text{Ex. } p(\mathbf{x}) = \sum_{x_1,x_3,x_4} f_a(x_1,x_2)f_b(x_2,x_3)f_c(x_2,x_4)$$
       $$= \left(\sum_{x_1} f_a(x_1,x_2)\right)\left(\sum_{x_3} f_b(x_2,x_3)\right)\left(\sum_{x_4} f_c(x_2,x_4)\right)$$
   - Assumption:
     - Tree-structured factor graph $\Rightarrow$ to use independence (expressing solution space of DP)
   - Messages:
     - Variable nodes $\rightarrow$ Factor node:

       $$\text{Leaves: } \mu_{x\to f}(x) = 1$$
       $$\text{Other: } \mu_{x\to f}(x) = \prod_{l\in\{ne(x)-f\}} \mu_{f_l\to x}(x), \quad \text{(get a function of x)}$$

       $\text{where } ne(x) \text{ is neighbour of } x$
     - Factor nodes $\rightarrow$ Variable node:

       $$\text{Leaves: } \mu_{f\to x}(x) = f(x)$$
       $$\text{Other: } \mu_{f_S\to x}(x) = \sum_{X_S}\left(f_S(x,X_S)\prod_{m\in\{ne(f_S)-x\}} \mu_{x_m\to f_S}\right), \quad \text{(get a fuction of } x\text{)}$$

       $\text{where } X_S \text{ is the set of nodes connected to factor } f_S(\cdot) \text{ except for } x \ (X_S = \{ne(f_S)-x\})$
   - Distribution with Messages:
     - $p(x) = \prod_{l\in ne(x)} \mu_{f_l\to x}(x)$

       $\text{Product over all the incoming messages}$

       $\Rightarrow \text{only depends on its neighbour}$

       $\text{(as info of sub-graph is stored in its beighbour)}$
   - Algorithm:
     - Arbitrarily choose one root
     - Propagate messages $\text{Leaves} \rightarrow \text{Root}$
     - Propagate messages back $\text{Root} \rightarrow \text{Leaves}$
     - $\Rightarrow$ With all message stored, marginal distribution computed easily
   - Normalization:
     - $Z = \sum_x p(x)$, $\text{where } p(x) \text{ is calculated by message propagating}$

- Extension on not tree-structured graph (**Loopy Belief Propagation**):
  - Assume a unit message sent over each link in each direction

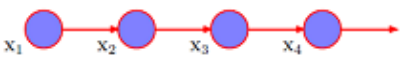
    (extract out the cyclic sub-graph, substituted with one node)
  - Try algorithm on cyclic graph
  - Problem: Convergence is **NOT** guaranteed

8. Sequential Data

- Problem to solve:
  - Current data depends on previous data $\Rightarrow$ **i.i.d.** assumption **not** hold
    - Time series
    - Natrual language
    - ...
  - Changed distribution while drawing data
  - Both of the above
- Category:
  - Stationary case:
    - data evolves in time; generative distribution stays the same
  - Nonstationary case:
    - generative distribution changes along the time

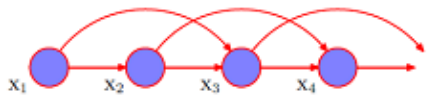
Markov Model

1. First-order Markov Chain:
   - Assumption:
     - Each data only depends on the most recent
- Joint distribution:
  - $p(x_1, \ldots, x_N) = p(x_1) \prod_{n=2}^{N} p(x_n | x_{n-1})$
  - BNs:

    $\Rightarrow p(x_n | x_1, \ldots, x_{n-1}) = \cdots = p(x_n | x_{n-1})$
2. Second-order Markov Chain:
   - Assumption:
     - The trend in previous observations is important

       $\Rightarrow$ have at least 2 observations for a trend
   - Joint distribution:
     - $p(x_1, \ldots, x_N) = p(x_1) p(x_2 | x_1) \prod_{n=3}^{N} p(x_n | x_{n-1}, x_{n-2})$
     - BNs:



3. M^{th}-order Markov Chain:
   - Joint distribution:
     - $p(x_1, \ldots, x_N) = p(x_1)p(x_2|x_1)\cdots p(x_M|x_1, \ldots, x_{M-1}) \times \prod_{n=M+1}^{N} p(x_n|x_{n-1}, \ldots, x_{n-M})$
   - Analysis of M: $\textbf{(assume } K \textbf{ states for an observation)}$
     - $M = 0$ **:** no Markov parameter, data drawn $\textbf{i.i.d.}$