Artificial Intelligence

wu

August 2, 2019

Contents

1 solving problems by search

problem-solving agent

- **goal**
- **goal information** is the 1st step in problem-solving, based on the current situation and the agents performance measure
- **problem formulation** is the process of deciding what actions and states to consider, given a goal
- **search**
- **execution phase**

formulatesearchexecution
type of search

- **uninformed search algorithms**

  algorithms that are given no information about the problem other than its definition. Although some of these algorithms can solve any solvable problem, none of them can do so efficiently

- **informed search algorithms**

The types of Problem-solving by Search

- Deterministic, fully observable

  Agent knows exactly which state it will be in

  solution is a sequence

- non-observable

  Agent may have no idea where it is

  solution (if any) is a sequence

- Nondeterministic and/or partially observable

  percepts provide new information about current state

  solution is a tree or policy

  often interleave search, execution

- Unknown state space

Some assumptions about environment

- **observable**
- **discrete**: the environment is discrete
- **known**: the agent knows which states are reached by each action
- **deterministic**: each action has exactly one outcome

Problem definition

1. **Initial state**
2. **actions**
3. **Transition model**
4. **goal test**: determines whether a given state is a goal state
5. **path cost**: a function that assigns a numeric cost to each path

A solution is an **action sequence**, so search algorithms work by considering various possible action sequences.

Given a search tree, the set of all leaf nodes available for expansion at any given point is called the **frontier(open list)**. **Search strategy**

queues: FIFO queue, LIFO queue (stack), priority queue

Measuring problem-solving performance:

- completeness: Does it always find a solution
- optimality: How long does it take?
- time complexity
- space complexity

uninformd search: Breadth-first search, Depth-first search

Strategies that know whether one non-goal state is more promising than another are called **informed search** or **heuristic search** strategies

1.1 uninformed search

Uniform-cost search: Instead of expanding the shallowest node, uniform-cost search expands the node n with the lowest path cost g(n). This is done by storing the frontier as a priority queue ordered by g(n)

DFS stack LIFO

Depth-limited search:

Iterative deepening depth-first search: for depth = 0 to ∞ do

1.2 Informed search strategies

best-first search

- Best-first search is an instance of the general TREE-SEARCH or GRAPH-SEARCH algorithm in which a node is selected for expansion based on an **evaluation function** f(n)
- The evaluation function is construed as a cost estimate, so the node with the **lowest evaluation** is expanded first

evaluation function *f*

- Most best-first algorithms include as a component of f a **heuristic function**, denoted h(n): *estimated cost of the cheapest path from the state at node n to a goal state*
- For now, we consider h(n) to be **arbitrary, nonnegative, problem-specific** functions, with one constraint: if n is a goal node, then h(n)=0
- **Greedy best-first search** f(n) = h(n)

A* search: f(n)=g(n)+h(n), h(n) the cost to get from the node to the goal
Conditions for optimality: **Admissibility** and Consistency

- f(n) = g(n) + h(n)
- g(n) is the actual cost to reach n along the current path
- h(n) is an **admissible heuristic** function: it never overestimates the cost to reach the goal
- So, f(n) never overestimates the true cost of a solution along the current path through n

Conditions for optimality: Admissibility and **Consistency**

- h(n) c(n, a, n') + h(n')
- for every node n and every successor n of n generated by any action a, the estimated cost of reaching the goal from n is no greater than the step cost of getting to n plus the estimated cost of reaching the goal from n

Properties of A* search

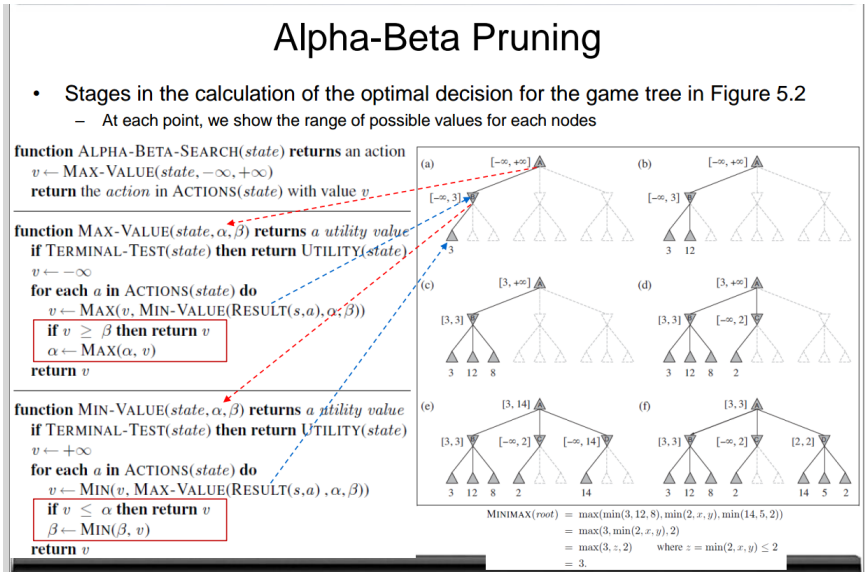
- The tree-search version of A* is optimal if h(n) is admissible
- The graph-search version of A* is optimal if h(n) is consistent

2 Adversarial search

2.1 Minimax search

2.2 evaluation function

2.3 Alpha-Beta Pruning Search



2.4 Monte-Carlo Tree Search

3 Inference and Reasoning

3.1 Propositional logic

3.2 Predicate logic

3.3 First Order Inductive Learner

knowledge graph: node = entity, edge = relation. triplet (head entity, relation, tail entity)

4 Statistical learning and modeling

4.1 Machine Learning: the concept

4.1.1 Example and concept

Supervised learning problems applications in which the **training data** comprises examples of the input vectors along with their corresponding **target vectors** are known

classification and regression

Unsupervised learning problems the training data consists of a set of input vectors X **without any corresponding target values**

density estimation, clustering, hidden markov models

Reinforcement learning problem finding suitable actions to take in a given situation in order to maximize a reward. Here the learning algorithm is not given examples of optimal outputs, in contrast to supervised learning, but must instead discover them by a process of trial and error. A general feature of reinforcement learning is the trade-off between exploration and exploitation

types of machine learning

- supervised learning
  - classification: the output is categorical or nominal variable
  - regression: the output is read-valued variable
- unsupervised learning
- semi-supervised learning
- reinforcement learning
- deep learning

4.1.2 supervised learning: important concepts

- Data: labeled instances $< \boldsymbol{x}_i, \boldsymbol{y} >$
- features: attribute-value pairs which characterize each $\boldsymbol{x}$
- learning a discrete function: **classification**

- learning a continuous function: **regression**

Classification - A two-step process

- **model construction**
- **model usage**

regression

- Example: price of a used car

  $\boldsymbol{x}$: car attributes. $\boldsymbol{y} = g(\boldsymbol{x} \mid \boldsymbol{\theta})$: price. g: model. θ parameter set.

4.2 example: polynomial curve fitting

cross validation

SSE error(sum-of-square) $E(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \boldsymbol{w}) - t_n\}^2$
RMS(root-mean-square) error $E_{RMS} = \sqrt{2E(\boldsymbol{w}^*)/N}$
high variance -> overfitting
How to control over-fitting

1. more train data
2. regularization
3. bayesian approach
4. cross-validation

curse of dimensionality

- Extend polynomial curve fitting approach to deal with input spaces having several variables. If we have D input variables, then a general polynomial with coefficients up to order 3 would take the form:

$$y(\boldsymbol{x}, \boldsymbol{w}) = w_0 + \sum_{i=1}^{D} w_i x_i + \sum_{i=1}^{D}\sum_{j=1}^{D} w_{ij} x_i x_j + \sum_{i=1}^{D}\sum_{j=1}^{D}\sum_{k=1}^{D} w_{ijk} x_i x_j x_k$$

4.3 probability theory review and notation

rules of probability

- **sum rule** $p(X) = \sum_Y p(X, Y)$
- **product rule** $p(X, Y) = p(Y|X)p(X)$

Bayes' Theorem: $p(Y|X) = \frac{p(X|Y)p(Y)}{p(X)}$. Using sum rule $p(X) = \sum_Y p(X|Y)p(Y)$

probability densities.

$$p(x \in (a, b)) = \int_a^b p(x)dx$$
$$P(z) = \int_{-\infty}^{z} p(x)dx$$
$$\int_{-\infty}^{\infty} p(x)dx = 1 \quad p(x) \leq 0$$

$p(x)$ must satisfy two conditions

$$p(x) \leq 0$$
$$\int_{-\infty}^{\infty} p(x)dx = 1$$

expectation $\mathbb{E}[f] = \begin{cases} \sum_x p(x)f(x) & \text{discrete variables} \\ \int p(x)f(x)dx & \text{continuous variables} \end{cases}$. In either cases, $\mathbb{E}[f] \approx \frac{1}{N}\sum_{n=1}^{N} f(x_n)$. **conditional expectation**: $\mathbb{E}_x[f|y] = \sum_x p(x|y)f(x)$.

The **variance** of $f(x)$ is

$$\begin{aligned} var[f] &= \mathbb{E}[(f(x) - \mathbb{E}[f(x)])^2] \\ &= \mathbb{E}[f(x)^2 - 2f(x)\mathbb{E}[f(x)] + \mathbb{E}[f(x)]^2] \\ &= \mathbb{E}[f(x)^2] - \mathbb{E}[f(x)]^2 \end{aligned}$$

The **covariance** is

$$
\begin{aligned}
cov[x, y] &= \mathbb{E}_{x,y}[(x - \mathbb{E}[x])(y - \mathbb{E}[y])] \\
&= \mathbb{E}_{x,y}[xy] - \mathbb{E}[x]\mathbb{E}[y]
\end{aligned}
$$

$$
\mathbb{V}[X] = \sigma_X^2 = \mathbb{E}[(X - \mathbb{E}[X])^2] = \mathbb{E}[X^2] - \mathbb{E}[X]^2
$$

$$
\mathbb{V}[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} \mathbb{V}[X_i] + \sum_{i \neq j} \text{Cov}[X_i, X_j]
$$

$$
\begin{aligned}
&\text{Cov}[X, X] = \mathbb{V}[X] \\
&\text{Cov}[aX, bY] = ab\text{Cov}[X, Y] \\
&\text{Cov}[X + a, Y + b] = \text{Cov}[X, Y]
\end{aligned}
$$

the variance of the sum of two independent random variables is the sum of variance. Given

X	probability
x_1	p_1
$\dots$	$\dots$
x_n	p_n

Y	probability
y_1	q_1
$\dots$	$\dots$
y_m	q_m

$$
var(X + Y) = var(X) + var(Y)
$$

In case of two vectors of random variables $\boldsymbol{x}$ and $\boldsymbol{y}$, the covariance is a matrix

$$
\begin{aligned}
cov[\boldsymbol{x}, \boldsymbol{y}] &= \mathbb{E}_{\boldsymbol{x},\boldsymbol{y}}[(\boldsymbol{x} - \mathbb{E}[\boldsymbol{x}])(\boldsymbol{y}^T - \mathbb{E}[\boldsymbol{y}^T])] \\
&= \mathbb{E}_{\boldsymbol{x},\boldsymbol{y}}[\boldsymbol{x}\boldsymbol{y}^T] - \mathbb{E}[\boldsymbol{x}]\mathbb{E}[\boldsymbol{y}^T]
\end{aligned}
$$

Bayesian probabilities: $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$, $p(\mathcal{D}) = \int p(\mathcal{D}|\boldsymbol{w})p(\boldsymbol{w})\mathrm{d}\boldsymbol{w}$. For a data set $\mathcal{D} = \{t_1, \dots, t_n\}$ and assumption w, $p(w|\mathcal{D}) = \frac{p(\mathcal{D}|w)p(w)}{p(\mathcal{D})}$.

$p(w)$ is **prior probability**, $p(\mathcal{D}|w)$ is **likelihood** (the probability $\mathcal{D}$ happens). Hence

$$\text{posterior} \propto \text{likelihood} \times \text{prior}$$

Gaussian distribution.

$$\mathcal{N}(x|\mu, \sigma^2) = \frac{1}{(2\pi\sigma^2)^{1/2}} \exp\left\{-\frac{1}{2\sigma^2}(x-\mu)^2\right\}$$

μ is called **mean**, σ^2 is called **variance**, σ **standard deviation**, $\beta = 1/\sigma^2$ **precision**

$$\begin{aligned}
\mathbb{E}[x] &= \int_{-\infty}^{\infty} \mathcal{N}(x|\mu, \sigma^2) x dx = \mu \\
\mathbb{E}[x^2] &= \int_{-\infty}^{\infty} \mathcal{N}(x|\mu, \sigma^2) x^2 dx = \mu^2 + \sigma^2 \\
var[x] &= \mathbb{E}[x^2] - \mathbb{E}[x]^2 = \sigma^2
\end{aligned}$$

For D-dimensional vector $\boldsymbol{x}$ of continuous variables

$$\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\boldsymbol{\Sigma}|^{1/2}} \exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma^{-1}} (\boldsymbol{x}-\boldsymbol{\mu})\right\}$$

To determine values for the unknown parameters given μ and σ^2 by maximizing the likelihood function. Use log.

$$\begin{aligned}
P(\boldsymbol{X}|\mu, \sigma^2) &= \prod_{n=1}^{N} \mathcal{N}(x_n|\mu, \sigma^2) \\
\Rightarrow \ln P(\boldsymbol{X}|\mu, \sigma^2) &= -\frac{1}{2\sigma^2} \sum_{n=1}^{N} (x_n - \mu)^2 - \frac{N}{2} \ln \sigma^2 - \frac{N}{2} \ln(2\pi)
\end{aligned}$$

Hence $\mu_{ML} = \frac{1}{N} \sum_{n=1}^{N} x_n$, $\sigma^2_{ML} = \frac{1}{N} \sum_{n=1}^{N} (x_n - \mu_{ML})^2$ by partial derivative. $\mathbb{E}[\sigma^2_{ML}] = (\frac{N-1}{N})\sigma^2$

Maximum likelihood estimator for mean is unbiased, that is, $\mathbb{E}(\mu_{ML}) = \mu$. Maximum likelihood estimator for variance is biased. $\mathbb{E}(\sigma^2_{ML}) = \mathbb{E}(x^2) - \mathbb{E}(\mu^2_{ML}) = \frac{N-1}{N}\sigma^2_x$

4.4 information theory

entropy: measuring uncertainty of a random variable X. $H(X) = H(p) = -\sum_{x\in\Omega} p(x)\log p(x)$ where Ω is all possible values and define $0\log 0 = 0$, $\log = \log_2$

$H(X) = \sum_{x\in\Omega} p(x)\log_2 \frac{1}{p(x)} = E(\log_2 \frac{1}{p(x)})$. And "information of x"="#bits to code x"$=-\log p(x)$

Kullback-Leibler divergence: comparing two distributions $D_{KL}(p||q) = H(p,q) - H(p) = -\int p(\boldsymbol{x}) \ln\left\{\frac{q(\boldsymbol{x})}{p(\boldsymbol{x})}\right\} d\boldsymbol{x}$

`https://www.youtube.com/watch?v=ErfnhcEV108`

mutual information $I[\boldsymbol{x},\boldsymbol{y}] = \mathrm{KL}(p(\boldsymbol{x},\boldsymbol{y})||p(\boldsymbol{x})p(\boldsymbol{y})) = H(\boldsymbol{y}) - H[\boldsymbol{y}|\boldsymbol{x}]$

4.5 The gaussian distribution

$$\begin{aligned}\Delta^2 &= (x-\mu)^T\Sigma^{-1}(x-\mu)\\ &= (x-\mu)^T U\Lambda^{-1}U^T(x-\mu)\\ &= (U^T(x-\mu))^T\Lambda^{-1}(U^T(x-\mu)) = y^T\Lambda^{-1}y\end{aligned}$$

$\Sigma u_i = \lambda_i u_i$ where $i = i,\ldots,D$.

$$\Sigma U = \Sigma(u_1,\ldots,u_D) = (u_1,\ldots,u_D)\begin{pmatrix}\lambda_1 & \ldots & 0\\ \vdots & \ddots & \vdots\\ 0 & \ldots & \lambda_D\end{pmatrix} = U\Lambda$$

$\forall i,j \in \{1,\ldots,D\}$,

$$u_i^T u_j = I_{ij} = \begin{cases}1 & \text{if } i=j\\ 0\end{cases}$$

$$U^T U = I$$

So U is orthogonal, $\Sigma UU^T = U\Lambda U^T = \sum_{i=1}^{D}\lambda_i u_i u_i^T$, and $\Sigma^T = U\Lambda^{-1}U^T$

$$\Delta^2 = \boldsymbol{y}^T\Lambda^{-1}\boldsymbol{y} \xrightarrow{y_i=\boldsymbol{u}_i^T(\boldsymbol{x}-\boldsymbol{\mu})} \sum_{i=1}^{D}\frac{y_i^2}{\lambda_i}$$

GIven a square matrix $A \in \mathbb{R}^{n\times n}, x \in \mathbb{R}^n$, x^TAx is called a **quadratic form**

$$x^TAx = \sum_{i=1}^{n} x_i(Ax)_i = \sum_{i=1}^{n} x_i(\sum_{j=1}^{n} A_{ij}x_j) = \sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij}x_ix_j$$

$$x^TAx = (x^TAx)^T = x^T(1/2A + 1/2A^T)x$$

Let $A = \Sigma^{-1}$, if A is not symmetric, let $A^* = (A + A^T)/2$, then it's symmetric

$$\text{cov}[\boldsymbol{x}] = \mathbb{E}[\boldsymbol{x}\boldsymbol{x}^T] - (\mathbb{E}[\boldsymbol{x}])^2 = \boldsymbol{\mu}\boldsymbol{\mu}^T - \boldsymbol{\Sigma} - \boldsymbol{\mu}^2 = \boldsymbol{\Sigma}$$

$$p(\boldsymbol{X}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \prod_{n=1}^{N} \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

$$\ln p(\boldsymbol{X}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = -\frac{ND}{2}\ln(2\pi) - \frac{N}{2}\ln|\boldsymbol{\Sigma}| - \frac{1}{2}\sum_{n=1}^{N}(\boldsymbol{x}_n - \boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}_n - \boldsymbol{\mu})$$

$$\frac{\partial}{\partial\boldsymbol{\Sigma}}\ln p(\boldsymbol{X}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = -\frac{N}{2}\frac{\partial}{\partial\boldsymbol{\Sigma}}\left\{|\boldsymbol{\Sigma}| - \frac{\partial}{2\partial\boldsymbol{\Sigma}}\sum_{n=1}^{N}(\boldsymbol{x}_n - \boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}_n - \boldsymbol{\mu})\right\}$$

Since $\dfrac{\partial \boldsymbol{a}^T\boldsymbol{X}^{-1}\boldsymbol{b}}{\partial\boldsymbol{X}} = -\boldsymbol{X}^{-1}\boldsymbol{a}\boldsymbol{b}^T\boldsymbol{X}^{-1}, \quad \dfrac{\partial}{\partial\boldsymbol{A}}\ln|\boldsymbol{A}| = (\boldsymbol{A}^{-1})^T$

$$-\frac{N}{2}\boldsymbol{\Sigma}^{-1} + \frac{N}{2}\boldsymbol{\Sigma}^{-1}\boldsymbol{S}\boldsymbol{\Sigma}^{-1} = 0, \quad \boldsymbol{S} = \frac{1}{N}\sum_{n=1}^{N}(\boldsymbol{x}_n - \boldsymbol{\mu})(\boldsymbol{x}_n - \boldsymbol{\mu})^T$$

$$\boldsymbol{\Sigma} = \boldsymbol{S} = \boldsymbol{\Sigma}_{ML}$$

$$\begin{aligned}
\mathbb{E}[\boldsymbol{\Sigma}_{ML}] &= \frac{1}{N}\sum_{n=1}^{N}\mathbb{E}[\left\{(\boldsymbol{x}_n - \frac{1}{N}\sum_{m=1}^{N}\boldsymbol{x}_m)(\boldsymbol{x}_n^T - \frac{1}{N}\sum_{l=1}^{N}\boldsymbol{x}_l^T)\right\}] \\
&= \frac{1}{N}\sum_{n=1}^{N}\mathbb{E}[\boldsymbol{x}_n\boldsymbol{x}_n^T - \frac{2}{N}\boldsymbol{x}_n\sum_{m=1}^{N}\boldsymbol{x}_m^T + \frac{1}{N^2}\sum_{m=1}^{N}\sum_{l=1}^{N}\boldsymbol{x}_m\boldsymbol{x}_l^T] \\
&= \boldsymbol{\mu}\boldsymbol{\mu}^T + \boldsymbol{\Sigma} - 2(\boldsymbol{\mu}\boldsymbol{\mu}^T + \frac{1}{N}\boldsymbol{\Sigma}) + \boldsymbol{\mu}\boldsymbol{\mu}^T + \frac{1}{N}\boldsymbol{\Sigma} \\
&= (\frac{N-1}{N})\boldsymbol{\Sigma}
\end{aligned}$$

4.6 Nonparametric methods

How to estimate unknown probability densituy $p(x)$:

- Assume we have collected a data set comprising N observations drawn from p(x). Consider some small region R containing x, the probability mass associated with this region is given by

$$P = \int_{\mathcal{R}} p(\boldsymbol{x}) d\boldsymbol{x} \quad \Rightarrow \quad p(\boldsymbol{x}) = \frac{K}{NV}$$

- V is the volumn of R
- K is the total number of points that lie inside R

1. **Kernel density estimator**

   Fix V, determine K from the data

2. **KNN density estimator**

   Fix K, determine the value of V from the data

4.6.1 Kernel density estimators

Parzen window

$$k(\boldsymbol{u}) = \begin{cases} 1 & |u_i| \leq 1/2, i = 1, \ldots, D \\ 0 & \text{otherwise} \end{cases}$$

- The total number of data points lying inside this cube:

$$K = \sum_{n=1}^{N} k(\frac{\boldsymbol{x} - \boldsymbol{x}_n}{h})$$

- The estimated density at x:

$$p(\boldsymbol{x}) = \frac{1}{N} \sum_{n=1}^{N} \frac{1}{h^D} k(\boldsymbol{x} - \boldsymbol{x}_n) h$$

4.6.2 Nearest-neighbour methods

Fix K, determine the value of V from the data

KNN classifier

$$p(\boldsymbol{x}|\mathcal{C}_k) = \frac{K_k}{N_k V}$$
$$p(\mathcal{C}_k) = \frac{N_k}{N}$$
$$p(\mathcal{C}_k|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{p(\boldsymbol{x})} = \frac{K_k}{K}$$
$$p(\boldsymbol{x}) = \frac{K}{NV}$$

- **training phase**

  storing the d-Dim feature vectors and class labels of the training samples

- **testing phase**

  An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k = 1,3,5,)

4.7 Linear model for classification

Regression: given a training data set comprising N observations $\{\boldsymbol{x}_n\}$, where $n = 1, \ldots, N$ together with corresponding target values $\{t_n\}$, the goal is to predict the value of t for a new value of $\boldsymbol{x}$

linear regression: $y(\boldsymbol{x}, \boldsymbol{w}) = w_0 + w_1 x_1 + \cdots + w_D x_D = \boldsymbol{w}^T \boldsymbol{x}$ where $\boldsymbol{x} = (x_1, \ldots, x_D)^T$

linear basis function model: Linear combinations of fixed nonlinear functions of the input variables

$$y(\boldsymbol{x}, \boldsymbol{w}) = w_0 + \sum_{j=1}^{M-1} w_j \phi_j(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x})$$

where $\phi_0(\boldsymbol{x}) = 1$, $\boldsymbol{\phi} = (\phi_0, \ldots, \phi_{M-1})^T$, $\boldsymbol{w} = (w_0, \ldots, w_{M-1})$

1. polynomial basis function $\phi_j(x) = x^j$
2. Gaussian basis function: $\phi_j(x) = \exp\left\{-\frac{(x-\mu_j)^2}{2s^2}\right\}$
3. sigmoid basis function: $\phi_j(x) = \sigma(\frac{x-\mu_j}{s})$, $\sigma(a) = \frac{1}{1+\exp(-a)}$

4.7.1 Maximum likelihood and least squares

Assume target variable t is given by a deterministic function $y(\boldsymbol{x}, \boldsymbol{w})$ with additive Gaussian noice so that $t = y(\boldsymbol{x}, \boldsymbol{w}) + \epsilon$ where ϵ is a zero mean Gaussian random variable with precision β, hence we can write

$$p(t|\boldsymbol{x}, \boldsymbol{w}, \beta) = \mathcal{N}(t|y(\boldsymbol{x}, \boldsymbol{w}), \beta^{-1})$$

and $\mathbb{E}(t|\boldsymbol{x}) = \int tp(t|\boldsymbol{x})dt = y(\boldsymbol{x}, \boldsymbol{w})$

For data set $\boldsymbol{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}, \boldsymbol{t} = (t_1, \ldots, t_n)^T$, $p(\boldsymbol{t}|\boldsymbol{X}, \boldsymbol{w}, \beta) = \prod_{n=1}^{N} \mathcal{N}(t_n|\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n), \beta^{-1})$

$$\ln p(\boldsymbol{t}|\boldsymbol{w}, \beta) = \sum_{n=1}^{N} \ln \mathcal{N}(t_n|\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n), \beta^{-1}) = \frac{N}{2}\ln\beta - \frac{N}{2}\ln(2\pi) - \beta\boxed{E_D(\boldsymbol{w})}$$

$E_D(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}\left\{t_n - \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n)\right\}^2 = \frac{1}{2}\|\boldsymbol{t} - \Phi\boldsymbol{w}\|$ is sum-of-squares error function

solve $\boldsymbol{w}$ by maximum likelihood.

$$\nabla \ln p(\boldsymbol{t}|\boldsymbol{w}, \beta) = \sum_{n=1}^{N}\left\{t_n - \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n)\right\}\phi(\boldsymbol{x}_n)^T$$

$$0 = \sum_{n=1}^{N} t_n\boldsymbol{\phi}(\boldsymbol{x}_n)^T - \boldsymbol{w}^T(\sum_{n=1}^{N}\boldsymbol{\phi}(\boldsymbol{x}_n)\boldsymbol{\phi}(\boldsymbol{x}_n)^T)$$

Hence we get

$$\boldsymbol{w}_{ML} = (\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\boldsymbol{t}$$

Φ is **design matrix**.

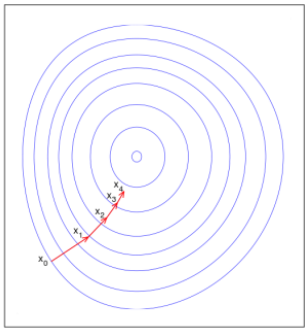
$$\Phi = \begin{pmatrix} \phi_0(\boldsymbol{x}_1) & \phi_1(\boldsymbol{x}_1) & \ldots & \phi_{M-1}(\boldsymbol{x}_1) \\ \phi_0(\boldsymbol{x}_2) & \phi_1(\boldsymbol{x}_2) & \ldots & \phi_{M-1}(\boldsymbol{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(\boldsymbol{x}_N) & \phi_1(\boldsymbol{x}_N) & \ldots & \phi_{M-1}(\boldsymbol{x}_N) \end{pmatrix}$$

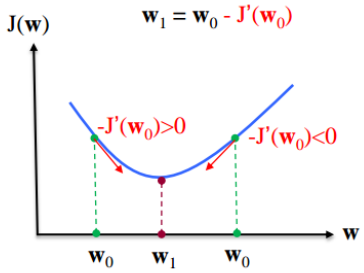
For bias parameter w_0. $E_D(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}\{t_n - w_0 - \sum_{j=1}^{M-1} w_j\phi_j(\boldsymbol{x}_n)\}^2$. Hence $w_0 = \bar{t} - \sum_{j=1}^{M-1} w_j\bar{\phi_j}$, $\bar{t} = \frac{1}{N}\sum_{n=1}^{N} t_n$, $\bar{\phi_j} = \frac{1}{N}\sum_{n=1}^{N}\phi_j(\boldsymbol{x}_n)$.

Solving the noise precision parameter β by ML $\frac{N}{2\beta} = E_D(\boldsymbol{w})$. $\frac{1}{\beta_{ML}} = \frac{1}{N}\sum_{n=1}^{N}\left\{t_n - \boldsymbol{w}_{ML}^T\boldsymbol{\phi}(\boldsymbol{x}_n)\right\}^2$

4.7.2 sequential learning

Gradient descent: Gradient descent is based on the observation that if the multivariable function $J(\boldsymbol{w})$ is defined and differentiable in a neighborhood of a point $\boldsymbol{w}_0$, then $J(\boldsymbol{w})$ decreases **fastest** if one goes from $\boldsymbol{w}_0$ in the direction of the negative gradient of J(.) at $\boldsymbol{w}_0 - J'(\boldsymbol{w}_0)$





batch gradient descent

$$\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} - \eta\nabla E_D(\boldsymbol{w})$$

$$E_D(\boldsymbol{w}) = \frac{1}{2}\sum_{n=1}^{N}\left\{t_n - \boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n)\right\}^2 = \frac{1}{2}\|\boldsymbol{t} - \boldsymbol{\Phi}\boldsymbol{w}\|^2$$

$$\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} + \eta\boldsymbol{\Phi}^T(\boldsymbol{t} - \boldsymbol{\Phi}\boldsymbol{w}^{(\tau)})$$

if $\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)}$, then $\boldsymbol{w}^{(\tau)} = (\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\boldsymbol{t}$

stochastic gradient descent

$$\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} - \eta\nabla E_n$$

4.7.3 Regularized least squares

Error function with regularization term

$$E_D(\boldsymbol{w}) + \lambda E_W(\boldsymbol{w}) = \frac{1}{2}\|\boldsymbol{t} - \boldsymbol{\Phi}\boldsymbol{w}\| + \frac{\lambda}{2}\boldsymbol{w}^T\boldsymbol{w}$$

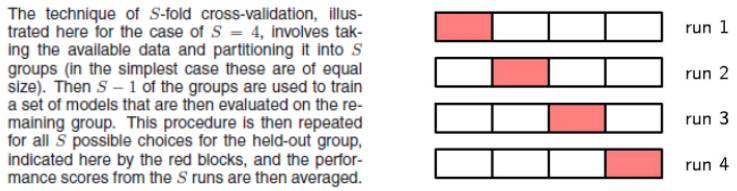
$$\boldsymbol{w} = (\lambda\boldsymbol{I} + \boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^T\boldsymbol{t}$$

4.7.4 multiple outputs

$\boldsymbol{y}(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{W}^T \boldsymbol{\phi}(\boldsymbol{x})$, $\boldsymbol{W} = (\boldsymbol{w}_1, \ldots, \boldsymbol{w}_K)_{M \times K}$

$\boldsymbol{W}_{ML} = (\boldsymbol{\Phi}^T \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^T \boldsymbol{T}$

4.8 model selection



cross-validation

split training data into **training set** and **validation set**. Train different models on training set and choose model with minimum error on validation set.

4.9 decision theory

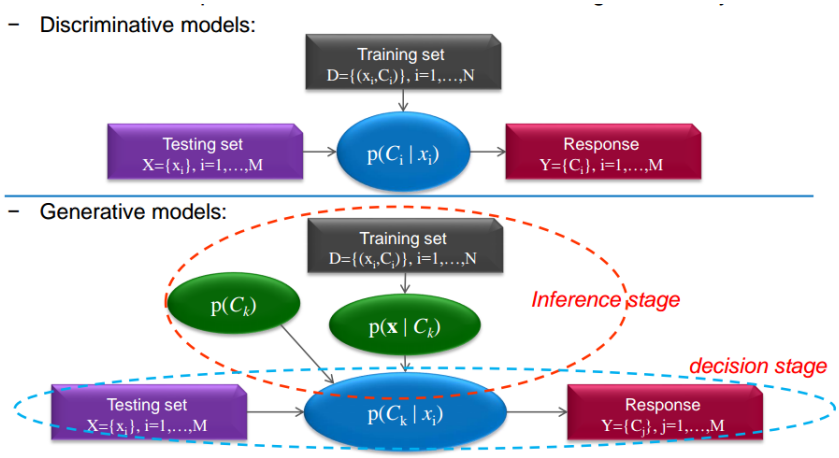
Suppose we have an input vector $\boldsymbol{x}$ together with a corresponding vector $\boldsymbol{t}$ of target variables and our goal is to predict $\boldsymbol{t}$ given new value for $\boldsymbol{x}$. The joint probability distribution $p(\boldsymbol{x}, \boldsymbol{t})$ provides a complete summary of the uncertainty with these variables

misclassification rate

$$\begin{aligned} p(\text{mistake}) &= p(\boldsymbol{x} \in \mathcal{R}_1, \mathcal{C}_2) + p(\boldsymbol{x} \in \mathcal{R}_2, \mathcal{C}_1) \\ &= \int_{\mathcal{R}_1} p(\boldsymbol{x}, \mathcal{C}_2) d\boldsymbol{x} + \int_{\mathcal{C}_2} p(\boldsymbol{x}, \mathcal{C}_1) d\boldsymbol{x} \end{aligned}$$

msupose loss matrix L

average loss $\mathbb{E}[L] = \sum_k \sum_j \int_{\mathcal{R}_j} L_{kj} p(\boldsymbol{x}, \mathcal{C}_k) d\boldsymbol{x}$



5 Statistical learning and modeling - Supervised learning

5.1 Basic concepts

- **Linearly separable**
  - decision regions:
    input space is divided into several regions
  - decision boundaries:
    - under linear models, it's a linear function
    - (D-1)-dimensional hyper-plane within the D-dimensional input space
- **representation of class labels**
  - Two classes K = 2
  - K classes
    - 1-of-K coding scheme $\boldsymbol{t} = (0, 0, 1, 0, 0)^T$
  - Predict discrete class labels
    - linear model prediction $y(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + w_0$
      w: weight vector, w_0 bias/threshold
    - nonlinear function $f(.) : R \to (0, 1)$

    - generalized linear models
      $y(\boldsymbol{x}) = f(\boldsymbol{w}^T\boldsymbol{x} + w_0)$
      f:activation function
    - dicision surface
      $y(\boldsymbol{x}) = \text{constant} \rightarrow \boldsymbol{w}^T\boldsymbol{x} + w_0 = \text{constant}$

- **Three classification approaches**
  - discriminant function
    - least squares approach
    - fisher's linear discriminant
    - the perceptron algorithm of rosenblatt
  - use discriminant functions directly and don't compute probabilities

    Given discriminant functions $f_1(\boldsymbol{x}), \ldots, f_K(\boldsymbol{x})$. Classify $\boldsymbol{x}$ as class $\mathcal{C}_k$ iff $f_k(\boldsymbol{x}) > f_j(\boldsymbol{x}), \forall j \neq k$
    - **least-squares approach**: making the model predictions as close as possible to a set of target values
    - **fisher's linear discriminant**: maximum class separation in the ouput space
    - **the perceptron algorithm of rosenblatt**
  - generative approach
    - model the class-conditional densities and the class priors
    - compute posterior probabilities through Bayes's theorem

$$\underbrace{p(\mathcal{C}_k|\boldsymbol{x})}_{\text{posterior for class}} = \frac{\overbrace{p(\boldsymbol{x}|\mathcal{C}_k)}^{\text{class conditional density}} \overbrace{p(\mathcal{C}_k)}^{\text{class prior}}}{p(\boldsymbol{x})} = \frac{p(\boldsymbol{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{\sum_j p(\boldsymbol{x}|\mathcal{C}_j)p(\mathcal{C}_j)}$$

5.2 discriminant functions

linear classification $\boldsymbol{y} = y(\boldsymbol{x}) = W^T\boldsymbol{x} + \boldsymbol{w}_0$

Hinge loss(c is the true label)

$$L_i = \sum_{j \in \{1,\ldots,C\}, j \neq c} \max(0, y_i^j - y_i^c + \epsilon)$$

loss function with regularization $L = \frac{1}{N} \sum_{i=1,\ldots,N} L_i + \alpha\varnothing(W)$

dimensionality reduction

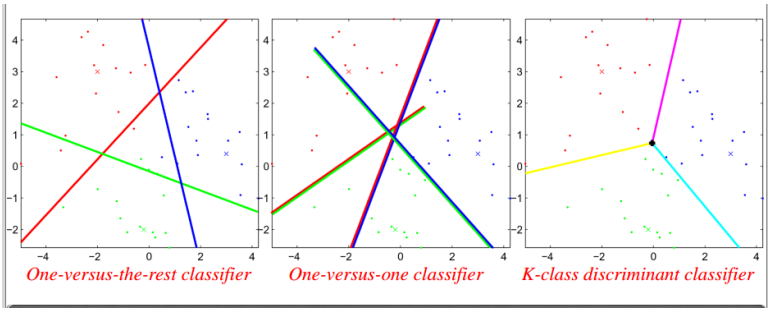
1. PCA - principal component analysis
2. SVD - singular value decomposition

5.2.1 Two classes

- Linear discriminant function $y(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + w_0$
  - Dicision surface $\Omega : y(\boldsymbol{x}) = 0$
  - the normal distant from the origin to the dicision surface $\frac{\boldsymbol{w}^T\boldsymbol{x}}{\|\boldsymbol{w}\|} = -\frac{w_0}{\|\boldsymbol{w}\|}$
  - if x_A, x_B lie on the decision surface $y(\boldsymbol{x}_A) = y(\boldsymbol{x}_B) = 0$, then $\boldsymbol{w}^T(\boldsymbol{x}_A - \boldsymbol{x}_B) = 0$. hence w is orthogonal to every vector lying within . $\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}$ is the normal vector of
  - $\boldsymbol{x} = \boldsymbol{x}_\perp + r\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}$ hence $r = \frac{y(\boldsymbol{x})}{\|\boldsymbol{w}\|}$. $y(\boldsymbol{x}_\perp) = 0 \rightarrow \boldsymbol{w}^T\boldsymbol{x} = -w_0 + r\frac{\boldsymbol{w}^T\boldsymbol{w}}{\|\boldsymbol{w}\|}$
  - $\tilde{\boldsymbol{w}} = (w_0, \boldsymbol{w}), \tilde{\boldsymbol{x}} = (x_0, \boldsymbol{x}), y(\boldsymbol{x}) = \tilde{\boldsymbol{w}}^T\tilde{\boldsymbol{x}}$

5.2.2 K-class

- One-versus-the-rest classifier K - 1 classifiers each of which solves a two-class problem
- One-versus-one classifier K(K-1)/2 binary discriminant functions
- K-class discriminant classifier



- single K-class discriminant comprising K linear functions $y_k(\boldsymbol{x}) = \boldsymbol{w}_k^T\boldsymbol{x} + w_{k_0}$
  - assigning a point x to class $\mathcal{C}_k$ if $y_k(\boldsymbol{x} > y_j(\boldsymbol{x}))$ for all $j \neq k$

- dicision boundary between class $\mathcal{C}_k, \mathcal{C}_j$ is given $y_k(\boldsymbol{x}) = y_j(\boldsymbol{x}) \rightarrow (\boldsymbol{w}_k - \boldsymbol{w}_j)^T \boldsymbol{x} + (w_{k_0} - w_{j_0}) = 0$
- $\mathcal{R}_k$ is singly connected convex
- $\hat{\boldsymbol{x}} = \lambda \boldsymbol{x}_A + (1-\lambda)\boldsymbol{x}_B$ where $0 \leq \lambda \leq 1$, $y_k(\hat{\boldsymbol{x}}) = \lambda y_k(\boldsymbol{x}_A) + (1-\lambda) y_k(\boldsymbol{x}_B)$ and hence $\hat{x}$ also lies inside $\mathcal{R}_k$

5.2.3 Learning the parameters of linear discriminant functions

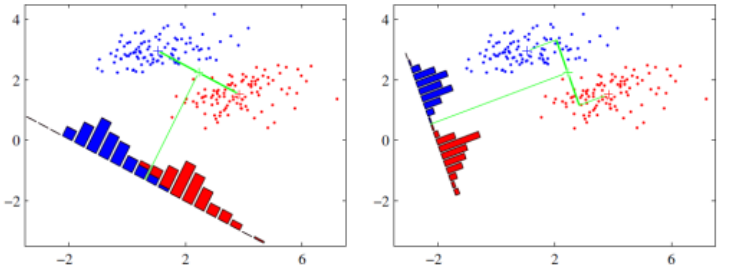
1. Least-squares approach

   - Problem
     - Each class $\mathcal{C}_k$ is described by its own linear model $y_k(\boldsymbol{x}) = \boldsymbol{w}_k^T \boldsymbol{x} + w_{k0}$
     - group together: $y(\boldsymbol{x}) = \widetilde{\boldsymbol{W}}^T \tilde{\boldsymbol{x}}$, $\tilde{\boldsymbol{w}}_k = (w_{k0}, \boldsymbol{w}_k^T)^T$, $\tilde{\boldsymbol{x}} = (1, \boldsymbol{x}^T)^T$
   - Learning $\widetilde{\boldsymbol{W}}$ with training set $\{\boldsymbol{x}_n, \boldsymbol{t}_n\}$
     - minimizing SSE function sum-of-squares $SSE = \sum_{i=1}^{n} (y_i - f(x_i))^2$

     $E_D(\widetilde{\boldsymbol{W}}) = 1/2 \mathrm{Tr}\{(\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{W}} - \boldsymbol{T})^T (\widetilde{\boldsymbol{X}}\widetilde{\boldsymbol{W}} - \boldsymbol{T})\}$

     $\widetilde{\boldsymbol{W}} = (\widetilde{\boldsymbol{X}}^T \widetilde{\boldsymbol{X}})^{-1} \widetilde{\boldsymbol{X}}^T \boldsymbol{T}$

2. fisher's linear discriminant



$y \geqslant -w_0$ as class $\mathcal{C}_1$ belonging to $\mathcal{C}_1$ if $y(\mathbf{x}) \geqslant y_0$

from the view of dimensionality reduction $y \geq -w_0$ as class $\mathcal{C}_1$

$$m_1 = \frac{1}{N_1} \sum_{n \in \mathcal{C}_1} \boldsymbol{x}_n, m_2 = \frac{1}{N_2} \sum_{n \in \mathcal{C}_2} \boldsymbol{x}_n \xrightarrow{y = \boldsymbol{w}^T \boldsymbol{x}} m_2 - m_1 = \boldsymbol{w}^T (\boldsymbol{m}_2 - \boldsymbol{m}_1)$$

Fisher's criterion: maximize the separation between the projected class means as well as the inverse of the total within-class variance.

within-class variance $s_k^2 = \sum_{n \in \mathcal{C}_k} (y_n - m_k)^2$, $y = \boldsymbol{w}^T \boldsymbol{x}, m_k = \boldsymbol{w}^T \boldsymbol{m}_k$

generalized rayleigh quotient

$$J(\boldsymbol{w}) = \frac{(m_2 - m_1)^2}{s_1^2 + s_2^2} = \frac{\boldsymbol{w}^T \boldsymbol{S}_B \boldsymbol{w}}{\boldsymbol{w}^T \boldsymbol{S}_W \boldsymbol{w}}$$

between-class covariance matrix $\boldsymbol{S}_B = (\boldsymbol{m}_2 - \boldsymbol{m}_1)(\boldsymbol{m}_2 - \boldsymbol{m}_1)^T$

within-class covariance matrix $\boldsymbol{S}_W = \sum_{n \in \mathcal{C}_1} (\boldsymbol{x}_n - \boldsymbol{m}_1)(\boldsymbol{x}_n - \boldsymbol{m}_1)^T + \sum_{n \in \mathcal{C}_2} (\boldsymbol{x}_n - \boldsymbol{m}_2)(\boldsymbol{x}_n - \boldsymbol{m}_2)^T$

Fisher's linear discriminant: $\nabla J(\boldsymbol{w}) = 0 \Rightarrow (\boldsymbol{w}^T \boldsymbol{S}_B \boldsymbol{w}) \boldsymbol{S}_W \boldsymbol{w} = (\boldsymbol{w}^T \boldsymbol{S}_w \boldsymbol{w}) \boldsymbol{S}_B \boldsymbol{w}$

hence

$$\boldsymbol{w} \propto \boldsymbol{S}_W^{-1} (\boldsymbol{m}_2 - \boldsymbol{m}_1)$$

3. the perceptron algorithm of rosenblatt

   construct a generalized linear model

$$y(\boldsymbol{x}) = f(\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x})) \quad f(a) = \begin{cases} +1 & a \geq 0 \\ -1 & a < 0 \end{cases}$$

   $\boldsymbol{\phi}_n = \boldsymbol{\phi}(\boldsymbol{x}_n)$

   perceptron criterion(minimize): $E_P(\boldsymbol{w}) = -\sum_{n \in \mathcal{M}} \boldsymbol{w}^T \boldsymbol{\phi}_n t_n, t \in \{+1, -1\}$

$$\boldsymbol{w}^{(\tau+1)} = \boldsymbol{w}^{(\tau)} - \eta \nabla E_P(\boldsymbol{w}) = \boldsymbol{w}^{(\tau)} + \eta \boldsymbol{\phi}_n t_n$$

   Perceptron convergence theorem: If there exists an exact solution (in other words, if the training data set is linearly separable), then the perceptron learning algorithm is guaranteed to find an exact solution in a finite number of steps

5.3 probalibilistic generative models

Determine the class conditional densities and class-specifix priors, and then use Bayes' rule to obtain the posterior probabilites

A probabilistic view of classification from simple assumptions about the distribution of the data

$$\begin{aligned} p(\mathcal{C}_1|\boldsymbol{x}) &= \frac{p(\boldsymbol{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\boldsymbol{x}|\mathcal{C}_1)p(\mathcal{C}_1) + p(\boldsymbol{x}|\mathcal{C}_2)p(\mathcal{C}_2)} \\ &= \frac{1}{1+\exp(-a)} = \sigma(a) \end{aligned}$$

where

$$a = \ln \frac{p(\boldsymbol{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\boldsymbol{x}|\mathcal{C}_2)p(\mathcal{C}_2)}$$

and $\sigma(a)$ is the **logistic sigmoid** function defined by

$$\sigma(a) = \frac{1}{1+\exp(-a)}$$

and $\sigma(-a) = 1 - \sigma(a)$, its inverse is **logit** function

$$a = \ln(\frac{\sigma}{1-\sigma})$$

For case of $K > 2$ classes, we have the following **multi-class generalization**

$$p(\mathcal{C}_k|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{\sum_j p(\boldsymbol{x}|\mathcal{C}_j)p(\mathcal{C}_j)} = \frac{\exp(a_k)}{\sum_j \exp(a_j)}, a_k = \ln\left[p(\boldsymbol{x}|\mathcal{C}_k)p(\mathcal{C}_k)\right]$$

The **normalized exponential** is known as the **softmax function** as it represents a *smoothed version of the max function*

$$\text{if } a_k \ll a_j, \forall j \neq k, \text{then } p(\mathcal{C}_k|\boldsymbol{x}) \approx 1, p(\mathcal{C}_j|\boldsymbol{x}) \approx 0$$

For **continuous inputs**, assume

$$p(\boldsymbol{x}|\mathcal{C}_k) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\boldsymbol{\Sigma}|^{1/2}} \exp\left\{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu}_k)^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}_k)\right\}$$

1. 2 classes

$$
\begin{aligned}
p(\mathcal{C}_1|\boldsymbol{x}) &= \sigma(\boldsymbol{w}^T\boldsymbol{x} + w_0) \\
\boldsymbol{w} &= \boldsymbol{\Sigma}^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2) \\
w_0 &= -\frac{1}{2}\boldsymbol{\mu}_1^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_1 + \frac{1}{2}\boldsymbol{\mu}_2^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_2 + \ln\frac{p(\mathcal{C}_1)}{p(\mathcal{C}_2)}
\end{aligned}
$$

2. K classes

$$
\begin{aligned}
a_k(\boldsymbol{x}) &= \boldsymbol{w}_k^T\boldsymbol{x} + w_{k0} \\
\boldsymbol{w}_k &= \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k \\
w_{k0} &= -\frac{1}{2}\boldsymbol{\mu}_k^T\boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}_k + \ln p(\mathcal{C}_k)
\end{aligned}
$$

Maximum likelihood solution for two classes. Assume $p(\boldsymbol{x}_n|\mathcal{C}_n) = \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma})$ and $p(\mathcal{C}_1) = \pi, p(\mathcal{C}_2) = 1 - \pi$. We have $\{x_n, t_n\}$ where $t_n = 1$ denotes class $\mathcal{C}_1$

$$
p(\boldsymbol{t}|\pi, \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\Sigma}) = \prod_{n=1}^{N}[\pi\mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_1, \boldsymbol{\Sigma})]^{t_n}[(1-\pi)\mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_2, \boldsymbol{\Sigma})]^{1-t_n}
$$

$$
\ln p(\boldsymbol{t}|\pi, \boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \boldsymbol{\Sigma}) = \sum_{n=1}^{N}\left\{t_n\ln\pi + (1-t_n)\ln(1-\pi) + t_n\ln\mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_1, \boldsymbol{\Sigma}) + (1-t_n)\ln\mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_2, \boldsymbol{\Sigma})\right\}
$$

1. solve π

   $\pi = \frac{N_1}{N_1+N_2}$

2. solve $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2$

$$
\sum_{n=1}^{N} t_n\ln\mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_1, \boldsymbol{\Sigma}) = -\frac{1}{2}\sum_{n=1}^{N} t_n(\boldsymbol{x}_n - \boldsymbol{\mu}_1)^T\boldsymbol{\Sigma}^{-1}(\boldsymbol{x}_n - \boldsymbol{\mu}_1) + \text{const}
$$

$$
\boldsymbol{\mu}_1 = \frac{1}{N_1}\sum_{n=1}^{N} t_n\boldsymbol{x}_n, \boldsymbol{\mu}_2 = \frac{1}{N_2}\sum_{n=1}^{N}(1-t_n)\boldsymbol{x}_n
$$

3. solve $\boldsymbol{\Sigma}$

$$\boldsymbol{S} = \frac{N_1}{N}\underbrace{\boxed{\frac{1}{N_1}\sum_{n\in\mathcal{C}_1}(\boldsymbol{x}_n-\boldsymbol{\mu}_1)(\boldsymbol{x}_n-\boldsymbol{\mu}_1)^T}}_{S_1} + \frac{N_2}{N}\underbrace{\boxed{\frac{1}{N_2}\sum_{n\in\mathcal{C}_2}(\boldsymbol{x}_n-\boldsymbol{\mu}_2)(\boldsymbol{x}_n-\boldsymbol{\mu}_2)^T}}_{S_2} \quad x$$

$\boldsymbol{\Sigma}_{ML} = \boldsymbol{S}$

5.4 probabilistic discriminative models

train all of the model parameters to maximize the probability of getting the label right. Model $p(\mathcal{C}_k|\boldsymbol{x})$ directly

logistic sigmoid function

$$p(\mathcal{C}_1|\boldsymbol{x}) = \frac{1}{1+\exp(-\boldsymbol{w}^T\boldsymbol{x})} = \sigma(\boldsymbol{w}^T\boldsymbol{x})$$

training dataset $\{\boldsymbol{x}_n, t_n\}$, $t_n \in \{0,1\}$

maximize the probability of getting the label right, so

$$p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{w}) = \prod_{n=1}^{N}\left[y_n^{t_n}(1-y_n)^{1-t_n}\right], \quad y_n = \sigma(\boldsymbol{w}^T\boldsymbol{x}_n)$$

cross-entropy error function

$$E(\boldsymbol{w}) = -\ln p(\boldsymbol{t}|\boldsymbol{X},\boldsymbol{w}) = -\sum_{n=1}^{N}\left[t_n\ln y_n + (1-t_n)\ln(1-y_n)\right] = \sum_{n=1}^{N}E_n = \sum_{n=1}^{N}H(p,q)$$

hence

$$\nabla E(\boldsymbol{w}) = \sum_{n=1}^{N}(y_n-t_n)\boldsymbol{x}_n$$

the same form as the gradient of the sum-of-squares error function

logistic regression model: $p(\mathcal{C}_1|\phi) = y(\phi) = \sigma(\boldsymbol{w}^T\boldsymbol{\phi})$. Only **M** parameters need to be estimated

$$\nabla E(\boldsymbol{w}) = \sum_{n=1}^{N}(y_n-t_n)\phi_n$$

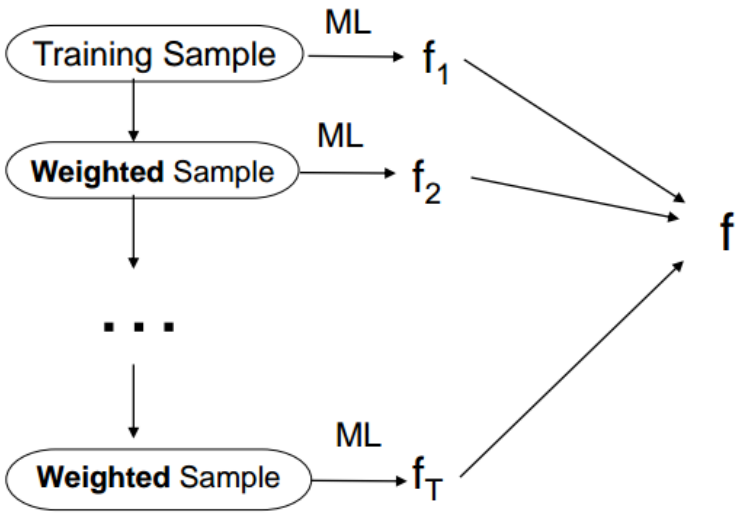
Newton-raphson iterative optmization scheme

$$\boldsymbol{w}^{\text{new}} = \boldsymbol{w}^{\text{old}} - \boldsymbol{H}^{-1}\nabla E(\boldsymbol{w})$$

5.5 Boosting

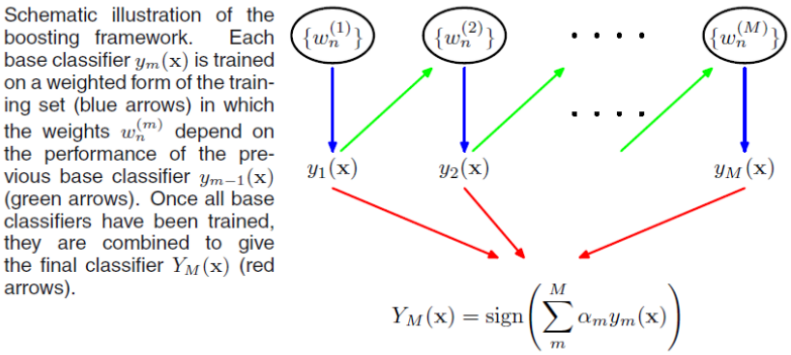
Originally designed for classification problems.

Motivation: a procedure that combines the outputs of many "weak" classifiers to produce a strong/accurate classifier



5.5.1 AdaBoost

adaptive boosting



$t_n \in \{-1, 1\}, y(\boldsymbol{x}) \in \{-1, 1\}$ .algorithm

1. initialize $\{w_n\}$ by $w_n^{(1)} = 1/N$ for $n = 1, \ldots, N$

2. for $m = 1, \ldots, M$

   (a) find a classifier $y_m(\boldsymbol{x})$ by minimizing

   $$J_m = \sum_{n=1}^{N} w_n^{(w)} I(y_m(\boldsymbol{x}_n) \neq t_n)$$

   where $I = 1$ if $y_m(\boldsymbol{x}_n) \neq t_n$

   (b) evaluate

   $$\epsilon_m = J_m / \sum_{n=1}^{N} w_n^{(m)}$$

   then

   $$\alpha_m = \ln \left\{ \frac{1 - \epsilon_m}{\epsilon_m} \right\}$$

   (c) update

   $$w_n^{(m+1)} = w_n^{(m)} \exp \left\{ \alpha_m I(y_m(\boldsymbol{x}_n) \neq t_n) \right\}$$

3. make prediction

$$Y_M(\boldsymbol{x}) = \text{sign} \left(\sum_{m=1}^{M} \alpha_m y_m(\boldsymbol{x}) \right)$$

6 unsupervised learning - clustering em and PCA

6.1 K-means clustering

use $\boldsymbol{\mu}_k$ as a prototype associated with the k^{th} cluster, Distortion measure(responsibilities) $J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \|\boldsymbol{x}_n - \boldsymbol{\mu}_k\|^2$.

$$\frac{\partial J}{\partial \boldsymbol{\mu}_k} = 2 \sum_{n=1}^{N} r_{nk} (\boldsymbol{x}_n - \boldsymbol{\mu}_k) = 0$$

$$\boldsymbol{\mu}_k = \frac{\sum_n r_{nk} \boldsymbol{x}_n}{\sum_n r_{nk}}$$

$$r_{nk} = \begin{cases} 1 & \text{if } k = \arg\min_j \left\| \boldsymbol{x}_n - \boldsymbol{\mu}_j \right\|^2 \\ 0 & \text{otherwise} \end{cases}$$

example: 5 data points and 3 clusters

$$r_{n,k} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

K-means algorithm (batch version):

1. Pick number of clusters k
2. Randomly scatter k cluster centers in data space
3. Repeat: a. Assign each data point to its closest cluster center b. Move each cluster center to the mean of the points assigned to it

online k-means algorithm(sequential k-means)

$$\boldsymbol{\mu}_k^{new} = \boldsymbol{\mu}_k^{old} + \eta_n(\boldsymbol{x}_n - \boldsymbol{\mu}_k^{old})$$

k-medoids algorithm

- choose input data points as center
- Works with an arbitrary matrix of distances between data points instead of Euclidean distance
  - E.g. Manhattan distance or Minkowski distance

$$J = \sum_{n=1}^{N}\sum_{k=1}^{K} r_{nk}\|\boldsymbol{x}_n - \boldsymbol{\mu}_k\|^2 \Rightarrow \widetilde{J} = \sum_{n=1}^{N}\sum_{k=1}^{K} r_{nk}\mathcal{V}(\boldsymbol{x}_n, \boldsymbol{\mu}_k)$$

The limitation of K-means clustering

1. The K-means algorithm often convergence to a local minimum
2. The K-means algorithm adopts the hard assignment and doesnt consider the data density and probabilistic distribution.

6.2 Mixtures of Gaussians

- Definition:

$$p(\boldsymbol{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \quad \sum_{k=1}^{k} \pi_k = 1 \quad 0 \leq \pi_k \leq 1$$

- introduce a K-dimensional binary random variable $\boldsymbol{z} = (z_1, \ldots, z_k)^T$

$$z_k \in \{0, 1\} \quad \sum_k z_k = 1 \quad p(z_k = 1) = \pi_k$$

Hence $p(\boldsymbol{z}) = \prod_{k=1}^{K} \pi_k^{z_k}$, $\boldsymbol{z}$ is **latent variable** (inferred from other observed variables)

If $p(\boldsymbol{x}|z_k = 1) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma})$, then $p(\boldsymbol{x}|\boldsymbol{z}) = \prod_{k=1}^{K} \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)^{z_k}$

- **equivalent formulation** of the Gaussian mixture.

$$\boxed{p(\boldsymbol{x}) = \sum_{\boldsymbol{z}} p(\boldsymbol{x}|\boldsymbol{z}) p(\boldsymbol{z}) = \sum_{\boldsymbol{z}} \prod_{k=1}^{K} \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)^{z_k}}$$

$$= \sum_{j=1}^{K} \prod_{k=1}^{K} \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)^{I_{kj}} \quad I_{kj} = \begin{cases} 1 & \text{if } k = j \\ 0 & \text{otherwise} \end{cases}$$

$$= \sum_{j=1}^{K} \pi_j \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$$

responsibility:

$$\gamma(z_k) = p(z_k = 1|\boldsymbol{x}) = \frac{p(z_k = 1) p(\boldsymbol{x}|z_k = 1)}{\Sigma_{j=1}^{K} p(z_j = 1) p(\boldsymbol{x}|z_j = 1)} = \frac{\pi_k \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma})}{\Sigma_{j=1}^{K} \pi_j \mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}_j \boldsymbol{\Sigma}_j)}$$

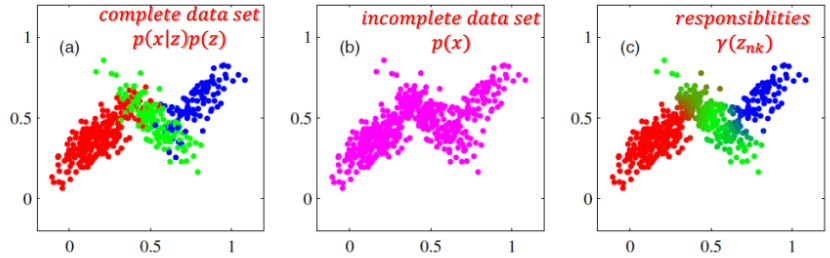


Figure 9.5 Example of 500 points drawn from the mixture of 3 Gaussians shown in Figure 2.23. (a) Samples from the joint distribution $p(\mathbf{z})p(\mathbf{x}|\mathbf{z})$ in which the three states of $\mathbf{z}$, corresponding to the three components of the mixture, are depicted in red, green, and blue, and (b) the corresponding samples from the marginal distribution $p(\mathbf{x})$, which is obtained by simply ignoring the values of $\mathbf{z}$ and just plotting the $\mathbf{x}$ values. The data set in (a) is said to be *complete*, whereas that in (b) is *incomplete*. (c) The same samples in which the colours represent the value of the responsibilities $\gamma(z_{nk})$ associated with data point $\mathbf{x}_n$, obtained by plotting the corresponding point using proportions of red, blue, and green ink given by $\gamma(z_{nk})$ for $k = 1, 2, 3$, respectively

$$\ln p(\boldsymbol{X}|\pi, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^{N} \ln \left\{ \sum_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\}$$

The difficulty of estimating parameters in GMM by ML

1. singularities

   Collapses onto a specific data point

2. identifiability

   Total K! equivalent solutions

3. no closed form solution

   The derivatives of the log likelihood are complex

Expectation-Maximization algorithm for GMM.

$p(\boldsymbol{X}|) = \prod p(\boldsymbol{x})$

$\ln p(\boldsymbol{X}|\pi, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \Sigma_{n=1}^{N} \ln \left\{ \Sigma_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\}$

1. E step

$$\gamma(z_{nk}) = \frac{\pi_k \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\Sigma_j \pi_j \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}$$

2. M step

- solve $\boldsymbol{\mu}_k$

$$\frac{\partial \ln p(\boldsymbol{X}|\pi, \boldsymbol{\mu}, \boldsymbol{\Sigma})}{\partial \boldsymbol{\mu}_k} = 0$$
$$0 = -\sum_{n=1}^{N} \frac{\pi_k \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\Sigma_j \pi_j \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} \boldsymbol{\Sigma}_k^{-1}(\boldsymbol{x}_n - \boldsymbol{\mu}_k)$$
$$\boldsymbol{\mu}_k = \frac{1}{N_k} \Sigma_{n=1}^{N} \gamma(z_{nk}) \boldsymbol{x}_n$$
$$N_k = \Sigma_{n=1}^{N} \gamma(z_{nk})$$

- solve $\boldsymbol{\Sigma}_k$

$$\frac{\partial \ln p(\boldsymbol{X}|\pi, \boldsymbol{\mu}, \boldsymbol{\Sigma})}{\partial \boldsymbol{\Sigma}_k} = 0$$
$$\boldsymbol{\Sigma}_k = \frac{1}{N_k} \Sigma_{n=1}^{N} \gamma(z_{nk})(\boldsymbol{x}_n - \boldsymbol{\mu}_k)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^T$$

- solve π_k

$$\frac{\partial}{\partial \pi_k} \left\{ \ln p(\boldsymbol{X}|\pi, \boldsymbol{\mu}, \boldsymbol{\Sigma}) + \lambda \left(\sum_{k=1}^{K} \pi_k - 1 \right) \right\} = 0$$
$$0 = \sum_{n=1}^{N} \frac{\mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_j \pi_j \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} + \lambda$$
$$\pi_k = \frac{N_k}{N}$$

EM for Gaussian Mixtures

1. initialize the means $\boldsymbol{\mu}_k$, covariances $\boldsymbol{\Sigma}_k$ and mixing coefficients π_k
2. E step

   find the posterior probability of latent variable

$$\gamma(z_{nk}) = \frac{\pi_k \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\Sigma_j \pi_j \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}$$

3. M step

$$\boldsymbol{\mu}_k^{new} = \frac{1}{N_k}\Sigma_{n=1}^{N}\gamma(z_{nk})\boldsymbol{x}_n$$
$$\boldsymbol{\Sigma}_k^{new} = \frac{1}{N_k}\Sigma_{n=1}^{N}\gamma(z_{nk})(\boldsymbol{x}_n - \boldsymbol{\mu}_k^{new})(\boldsymbol{x}_n - \boldsymbol{\mu}_k^{new})^T$$
$$\pi_k^{new} = \frac{N_k}{N} \quad \text{where } N_k = \sum_{n=1}^{N}\gamma(z_{nk})$$

4. evaluate the log likelihood

$$\ln p(\boldsymbol{X}|\pi, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \Sigma_{n=1}^{N} \ln \left\{ \Sigma_{k=1}^{K} \pi_k \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\}$$

and check for convergence of either the parameters or the log likelihood. If the convergence criterion is not satisfied return to step 2

6.3 An alternative view of EM

6.3.1 the general EM algorithm

Data $\boldsymbol{X}$, observation $\boldsymbol{\theta}$ The log likelihood of a discrete latent variables model

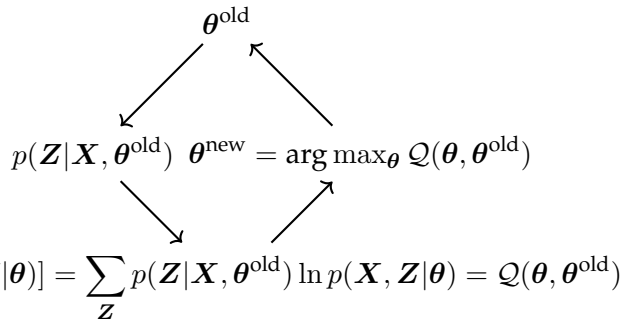
$$\ln p(\boldsymbol{X}|\boldsymbol{\theta}) = \ln \left\{ \sum_{\boldsymbol{Z}} p(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\theta}) \right\}$$

the goal of EM algorithm is to find maximum likelihood solution for models having latent variables

For the complete data set $\{\boldsymbol{X}, \boldsymbol{Z}\}$, the likelihood function

$$\ln p(\boldsymbol{X}|\boldsymbol{\theta}) \Longrightarrow \ln p(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\theta})$$

For the incomplete data set $\{\boldsymbol{X}\}$, we adopt the following steps to find maximum likelihood solution



the general EM algorithm

Given a joint distribution $p(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\theta})$ over observed variables $\boldsymbol{X}$ and latent variables $\boldsymbol{Z}$, govened by parameters $\boldsymbol{\theta}$, the goal is to maximize the likelihood function $p(\boldsymbol{X}|\boldsymbol{\theta})$

1. choose an initial setting for the parameters $\boldsymbol{\theta}^{old}$
2. **E step** evaluate $p(\boldsymbol{Z}|\boldsymbol{X}, \boldsymbol{\theta}^{old})$
3. **M step**

$$\boldsymbol{\theta}^{\text{new}} = \arg\max_{\boldsymbol{\theta}} \mathcal{Q}(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}})$$
$$\mathcal{Q}(\boldsymbol{\theta}, \boldsymbol{\theta}^{\text{old}}) = \sum_{\boldsymbol{Z}} p(\boldsymbol{Z}|\boldsymbol{X}, \boldsymbol{\theta}^{\text{old}}) \ln p(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\theta})$$
$$\mathcal{Q}(\boldsymbol{\theta}, \boldsymbol{\theta}^{old} + \ln p(\boldsymbol{\theta}))$$

4. check for convergence of either the log likelihood or the parameter values. If the convergence criterion is not satisfied, then let

$$\boldsymbol{\theta}^{old} \leftarrow \boldsymbol{\theta}^{new}$$

6.3.2 Gaussian mixtures revisited

$$p(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\mu}, \boldsymbol{\Sigma}, \pi) = \prod_{n=1}^{N} \prod_{k=1}^{K} \pi_k^{z_{nk}} \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)^{z_{nk}}$$
$$\ln p(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\mu}, \boldsymbol{\Sigma}, \pi) = \sum_{n=1}^{N} \sum_{k=1}^{K} z_{nk} \left\{ \ln \pi_k + \ln \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\}$$
$$\pi_k = \frac{1}{N} \sum_{n=1}^{N} z_{nk}$$
$$\mathbb{E}_{\boldsymbol{Z}}[\ln p(\boldsymbol{X}, \boldsymbol{Z}|\boldsymbol{\mu}, \boldsymbol{\Sigma}, \pi)] = \sum_{n=1}^{N} \sum_{k=1}^{K} \gamma(z_{nk}) \left\{ \ln \pi_k + \ln \mathcal{N}(\boldsymbol{x}_n|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\}$$

relation to K-means

6.4 The EM in general

6.5 PCA

dimensionality reduction: The result of Dimensionality reduction should keep the original data structure

$$\text{Var}(X) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

$$\text{cov}X, Y = \frac{1}{n}\sum_{i=1}^{n}(x_i - E(X))(y_i - E(Y))$$

$$\text{corr(X,Y)} = \frac{\text{cov}(X, Y)}{\sqrt{\text{Var}(X)\text{Var}(Y)}} = \frac{\text{Cov}(X, Y)}{\sigma_x \sigma_y}$$

Pearson correlation coefficients

1. $\left|corr(X, Y)\right| \leq 1$
2. $corr(X, Y) = 1 \leftrightarrow \exists a, b, Y = aX + b$
3. Pearson Correlation coefficient measures the degree of linear correlation between variable X and Y
4. Positive correlation means as X increases, so does Y. Negative correlation means as X increases, Y goes down

motivation:

1. In dimension reduction, data should be projected to the direction with the largest variance as far as possible, by this way the information contained in the data is preserved and the personality is highlighted.
2. The motivation of PCA is to project d-dimensional data to l-dimensional space($d \gg l$) and remove the redundancy between data(by removing correlation between data).

Given $D = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}, \boldsymbol{x}_i \in \mathbb{R}^d$, then

$$\boldsymbol{Y}_{n\times l} = \boldsymbol{X}_{n\times d}\boldsymbol{W}_{d\times l}$$

$$var(\boldsymbol{Y}) = \frac{1}{n} trace(\boldsymbol{Y}^T\boldsymbol{Y}) = \frac{1}{n} trace(\boldsymbol{W}^T\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{W}) = trace(\boldsymbol{W}\frac{1}{n}\boldsymbol{X}^T\boldsymbol{X}\boldsymbol{W})$$
$$\Sigma = \frac{1}{n}\boldsymbol{X}^T\boldsymbol{X}$$
$$\max_{\boldsymbol{W}} trace(\boldsymbol{W}^T\boldsymbol{\Sigma}\boldsymbol{W}), \boldsymbol{w}_i^T\boldsymbol{w}_i = 1, i \in \{1, \ldots, l\}$$

Use Lagrangian multiplier

$$L(\boldsymbol{W}, \boldsymbol{\lambda}) = trace(\boldsymbol{W}^T\boldsymbol{\Sigma}\boldsymbol{W}) - \sum_{i=1}^{l} \lambda_i(\boldsymbol{w}_i^T\boldsymbol{w}_i^T - 1)$$

by partial derivative, we get

$$\boldsymbol{\Sigma}\boldsymbol{w}_i = \lambda_i\boldsymbol{w}_i$$

$$trace(\boldsymbol{W}^T\boldsymbol{\Sigma}\boldsymbol{W}) = \sum_{i=1}^{l} \lambda_i$$

algorithm

1. centralization $\boldsymbol{x}_i = \boldsymbol{x}_i - \bar{\boldsymbol{x}}$
2. $\boldsymbol{\Sigma} = \frac{1}{n}\boldsymbol{X}^T\boldsymbol{X}$
3. get the eigenvalues of $\boldsymbol{\Sigma}$ and sort

$$\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_1$$

4. Select the eigenvectors corresponding to top l biggest eigenvalues to form mapping matrix $\boldsymbol{W}$
5. Reduce the dimension of every sample $\boldsymbol{x}_i$

$$(\boldsymbol{x}_i)_{1\times d}\boldsymbol{W}_{d\times l} = 1 \times l$$

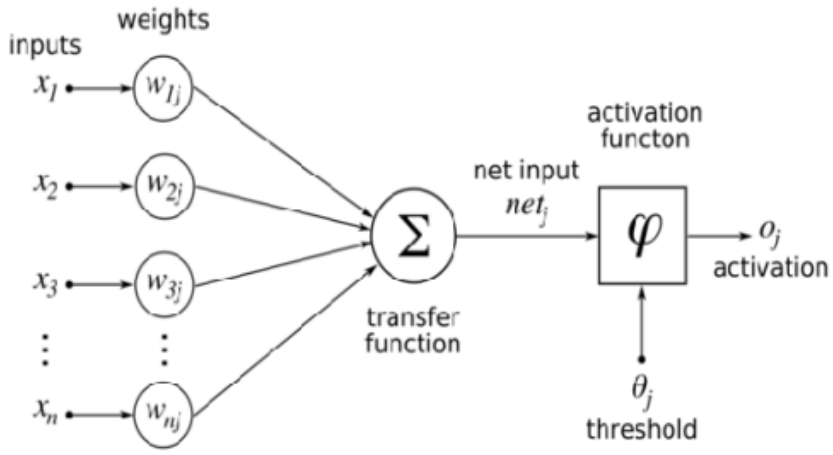
7 deep learning

7.1 Neural networks

7.1.1 biological inspiration

activation functions(non-linear functions)

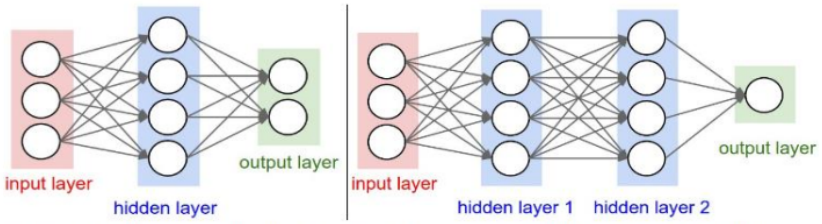
- sigmoid/logistic, tanh, rectified linear unit(ReLu)



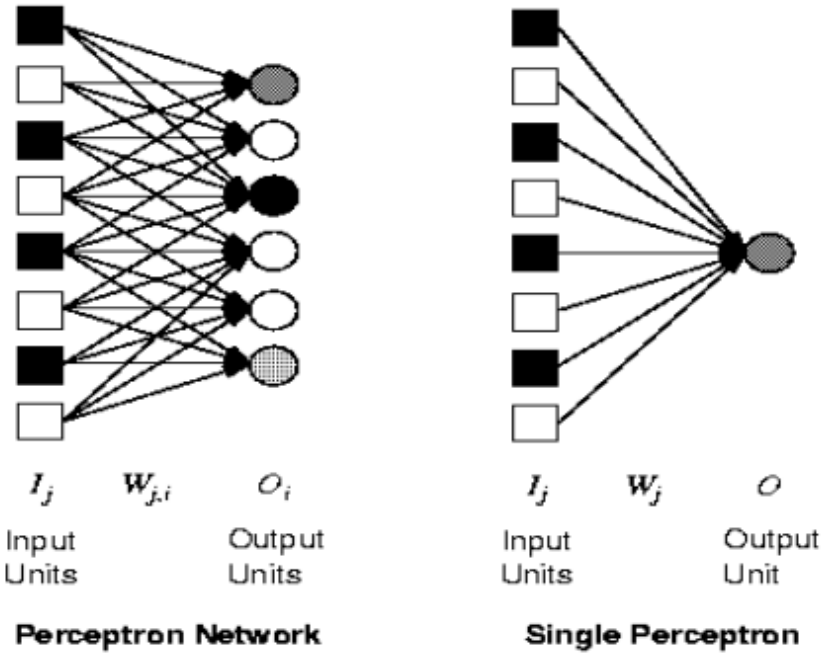
Name	Plot	Equation	Derivative (with respect to x)	Range	Order of continuity	Monotonic
Identity		$f(x) = x$	$f'(x) = 1$	$(-\infty, \infty)$	C^∞	Yes
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$	$\{0, 1\}$	C^{-1}	Yes
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$	$(0, 1)$	C^∞	Yes
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$	$(-1, 1)$	C^∞	Yes
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$	$\left(-\frac{\pi}{2}, \frac{\pi}{2}\right)$	C^∞	Yes
Softsign [7][8]		$f(x) = \frac{x}{1+\lvert x \rvert}$	$f'(x) = \frac{1}{(1+\lvert x \rvert)^2}$	$(-1, 1)$	C^1	Yes
Rectifier (ReLU)[9]		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$[0, \infty)$	C^0	Yes
Parameteric Rectified Linear Unit (PReLU)[10]		$f(\alpha, x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(\alpha, x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$(-\infty, \infty)$	C^0	Yes iff $\alpha \geq 0$

7.1.2 feedforward NN

fully-connected layer: neurons between two adjacent layers are fully pairwise connected



perception networks: Single-layer feed-forward neural networks (no hidden units)



Given any continuous function $f(x)$ and some $\epsilon < 0$, there exists a Neural network $g(x)$ with one hidden layer s.t.

$$\forall x, \left| f(x) - g(x) \right| < \epsilon$$

We increase the size and number of layers in a Neural Network, the **capacity** of the network increases. That is, the space of representable functions grows since the neurons can collaborate to express many different functions

7.2 optimization and gradient descent

Problem: how to lean the best W of a classifier

- dataset (x, y)
- score function $s = f(x; W) = Wx$
- loss function
  - softmax $L_i = -\log(\frac{e^{s_{y_i}}}{\sum_j e^{s_j}})$
  - SVM $L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1)$
  - full loss $L = \frac{1}{N} \sum_{i=1}^{N} L_i + R(W)$

7.2.1 gradient descent

7.2.2 stocahstic gradient descent

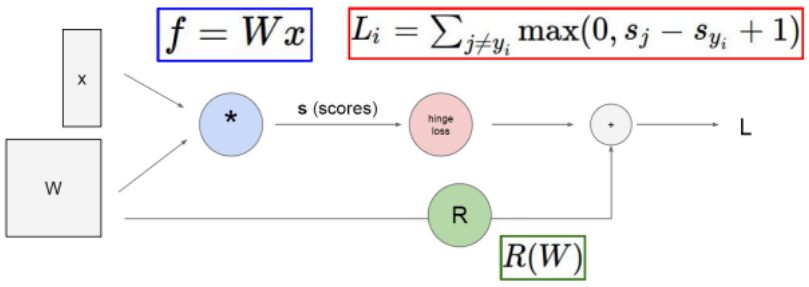
$$L(W) = \frac{1}{N} \sum_{i=1}^{N} L_i(x_i, y_i, W) + \lambda R(W)$$

$$\nabla_W L(W) = \frac{1}{N} \sum_{i=1}^{N} \nabla_W L_i(x_i, y_i, W) + \lambda \nabla_W R(W)$$

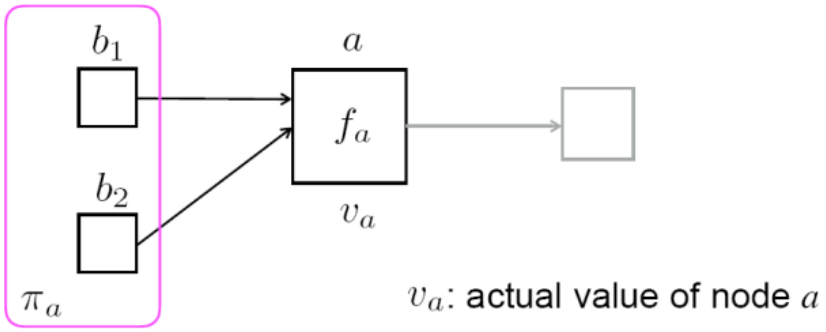
Approximate sum using a **minibatch** of examples

7.2.3 backpropagation

computation graph

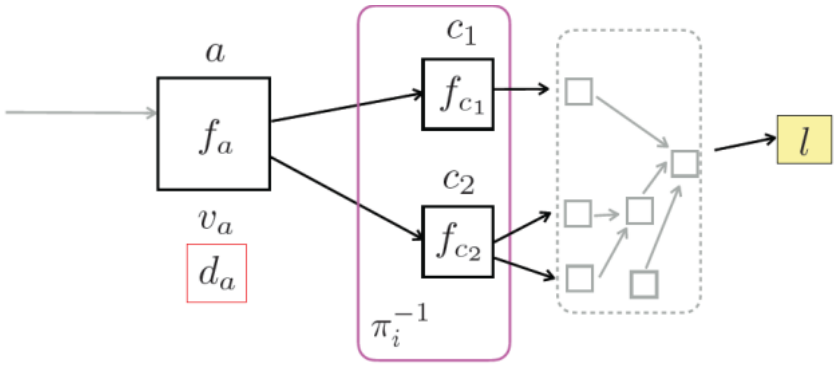


Forward computation

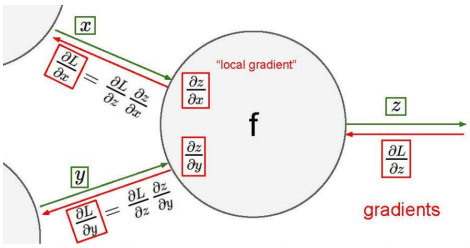


- Each node represented as a variable a
- $v_a = f_a(v_{b_1}, \ldots, v_{b_m})$

Backward computation



- $d_a = \frac{\partial l}{\partial a}$
- In general, $d_a = \sum_{c_i \in \pi_a^{-1}} d_{c_i} \cdot \frac{\partial f_{c_i}}{\partial a}$
- In this case, $d_a = d_{c_1} \frac{\partial f_{c_1}}{\partial a} + d_{c_2} \frac{\partial f_{c_2}}{\partial a}$

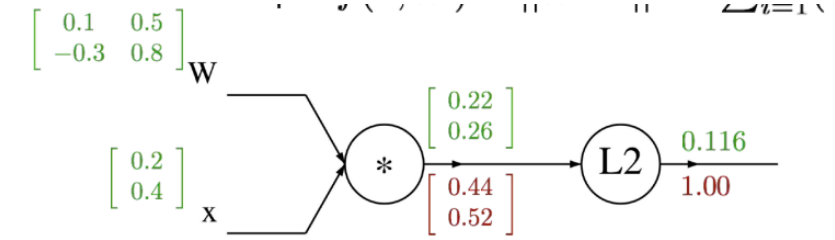


$$\frac{d\sigma(x)}{dx} = (1 - \sigma(x))\sigma(x)$$

patterns in backward flow

- **add** gate: gradient distributor
- **max** gate: gradient router
- **mul** gate: gradient switcher

For gradients for vectorized code, $\frac{\partial z}{\partial x}$ is **Jacobian matrix**. For example, $f(x, W) = \|W \cdot\|^2 = \sum_{i=1}^{n} (W \cdot x)_i^2$

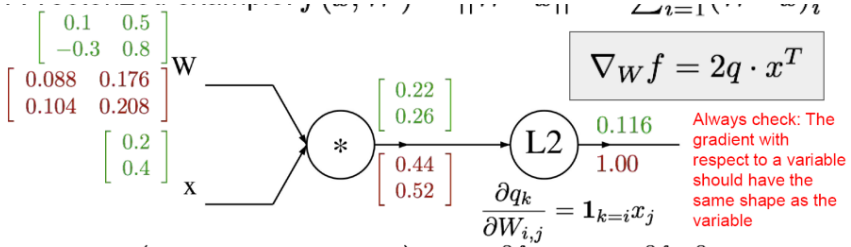


$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

$$\frac{\partial f}{\partial q_i} = 2q_i$$

$$\boxed{\nabla_q f = 2q}$$



$$\boxed{\nabla_W f = 2q \cdot x^T}$$

Always check: The gradient with respect to a variable should have the same shape as the variable

$$q = W \cdot x = \begin{pmatrix} W_{1,1}x_1 + \cdots + W_{1,n}x_n \\ \vdots \\ W_{n,1}x_1 + \cdots + W_{n,n}x_n \end{pmatrix}$$

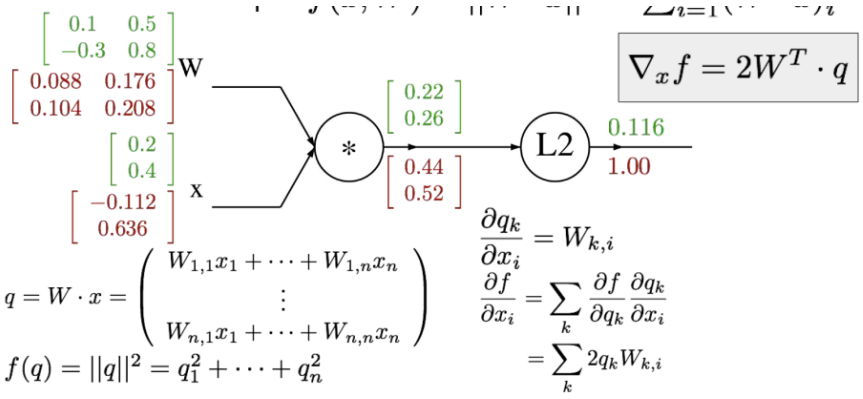
$$f(q) = ||q||^2 = q_1^2 + \cdots + q_n^2$$

$$\frac{\partial q_k}{\partial W_{i,j}} = \mathbf{1}_{k=i} x_j$$

$$\frac{\partial f}{\partial W_{i,j}} = \sum_k \frac{\partial f}{\partial q_k} \frac{\partial q_k}{\partial W_{i,j}}$$

$$= \sum_k (2q_k)(\mathbf{1}_{k=i} x_j)$$

$$= 2q_i x_j$$



7.3 convolutional neural network

7.3.1 basic concepts

summary:

- accepts a volume of size $W_1 \times H_1 \times D_1$
- four hyperparameters
  - number of filters K
  - their spatial extent F
  - the stride S
  - the amount of zero padding P
- produces a volume of size $W_2 \times H_2 \times D_2$
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$
  - $D_2 = K$
- parameter: $F^2 \times D_1 \times K + K$

Pooling layer

- Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting
- The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the MAX operation

Normalization layer

- Many types of normalization layers have been proposed for use in ConvNet architectures, sometimes with the intentions of implementing inhibition schemes observed in the biological brain. However, these layers have recently fallen out of favor because in practice their contribution has been shown to be minimal, if any.

Fully-connected layer

- Neurons in a fully connected layer have full connections to all activations in the previous layer, as seen in regular Neural Networks. Their activations can hence be computed with a matrix multiplication followed by a bias offset.
- Any FC layer can be converted to a CONV layer
- By converting FC layers to CONV layers, we can build a Fully Convolutional Networks

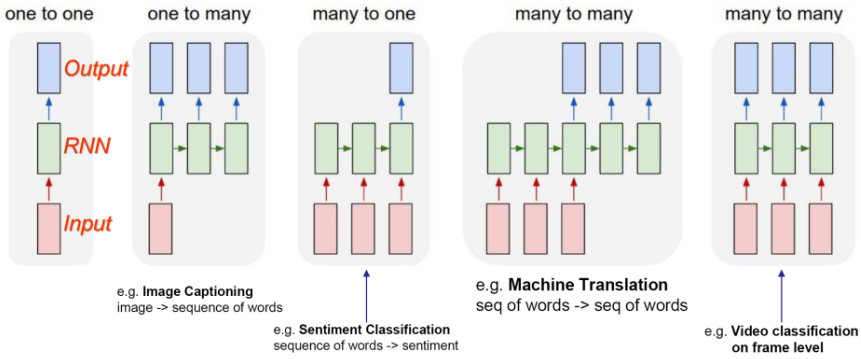
7.3.2 case study: AlexNet, GoogLeNet, VGG

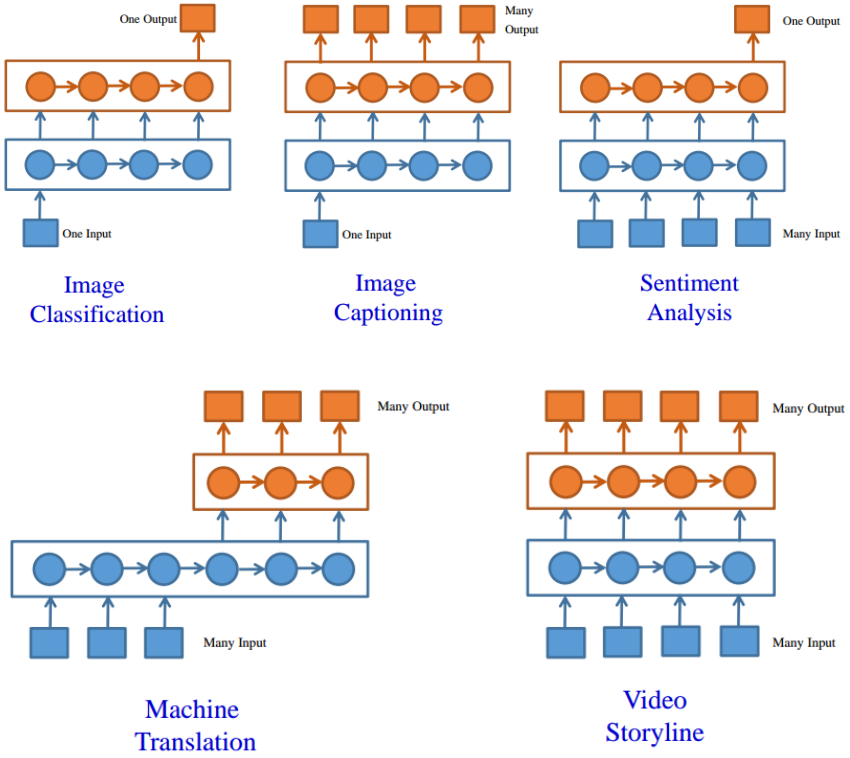
7.4 Appication of deep learning

7.5 Recurrent Neural Network(RNN)

7.5.1 RNN

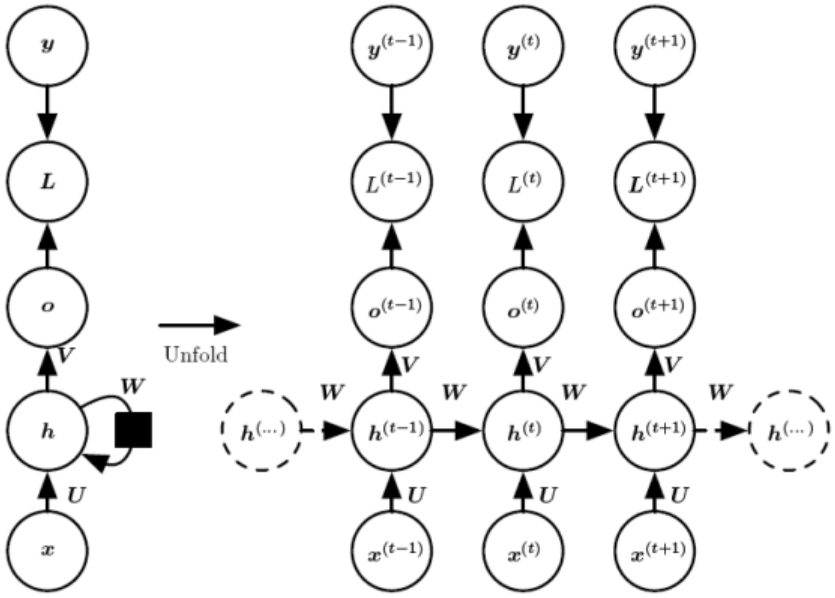
A family of neural networks for **processing sequential data** $x^{(1)}, x^{(2)}, \ldots, x^{(\tau)}$





Vanilla RNN

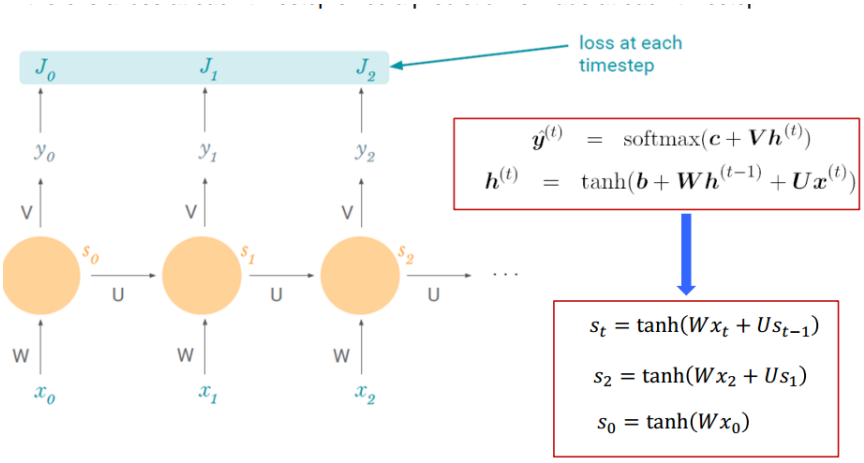
$$h_t = f_W(h_{t-1}, x_t)$$
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$
$$y_t = W_{hy}h_t$$
$$y_t = \text{softmax}(W_{hy}h_t)$$



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

$$
L(\left\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(\tau)}\right\}, \left\{\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(\tau)}\right\}) = \sum_t l^{(t)} = -\sum_t \log p_{model}(y^{(t)} | \left\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(t)}\right\})
$$

Training a RNN with gradient flow



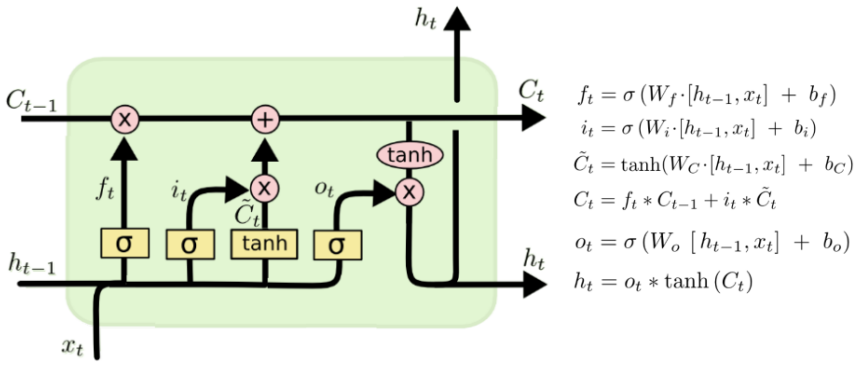
loss at time $t = J_t(\Theta)$, total loss $J(\Theta) = \sum_t J_t(\Theta)$ hence

$$\frac{\partial J}{\partial P} = \sum_t \frac{\partial J_t}{\partial P}$$

$$\frac{\partial J_2}{\partial W} = \frac{\partial J_2}{\partial y_2}\frac{\partial y_2}{\partial s_2}\frac{\partial s_2}{\partial W}$$
$$s_2 = \tanh(Us_1 + Wx_2)$$
$$\frac{\partial J_t}{\partial W} = \sum_{k=0}^{t} \frac{\partial J_t}{\partial y_t}\frac{\partial y_t}{\partial s_t}\frac{\partial s_t}{\partial s_k}\frac{\partial s_k}{\partial W}$$

Hard to train

7.5.2 long short-term memory and other gated RNNs



8 reinforcement learning

8.1 About RL

RL addresses the question of how an autonomous agent that sense and acts in its environment can learn an optimal control strategy, or policy, for choosing actions to achieve its goals (maximize cumulative reward).

A **reward** R_t is a scalar feedback signal. The agents job is to maximize cumulative reward

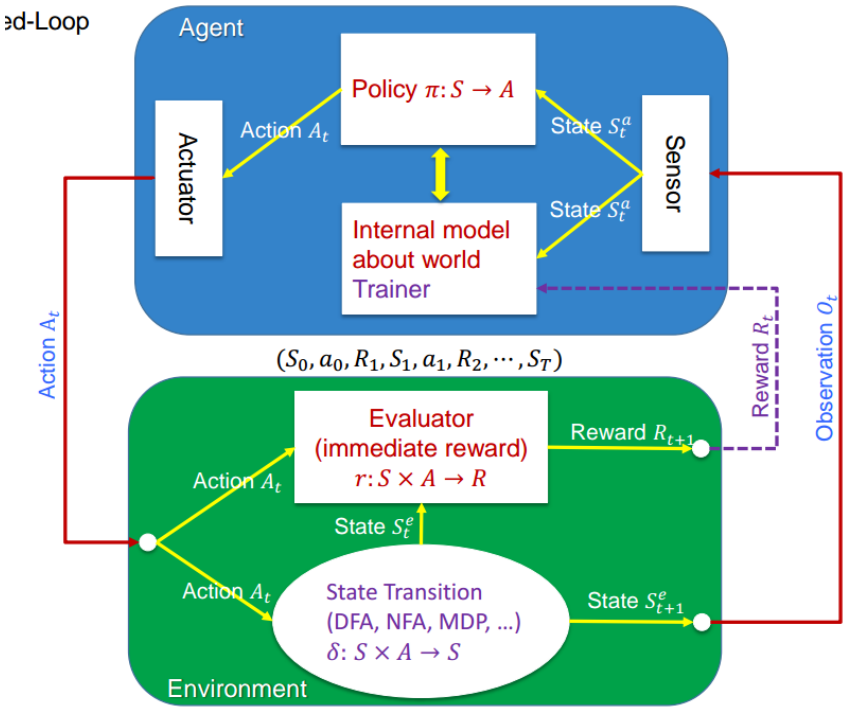
Reward hypothesis: All goals can be described by the maximization of expected cumulative reward

At each step t the agent

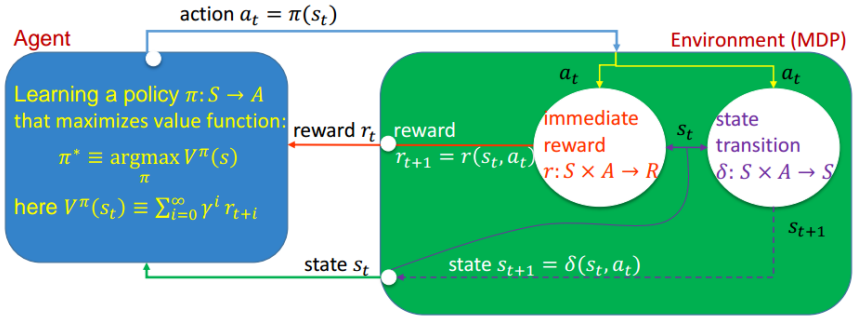
- Receives observation O_t
- Executes action A_t
- Receives scalar reward R_t

the environment:

- Receives action A_t
- Emits scalar reward R_{t+1}
- Emits observation O_{t+1}



policy: agent's behaviour function.
value function: how good is each state
model: agent's representation of the environment



History is the sequence of observations, actions, rewards

$$H_t = O_1, A_1, R_1, \ldots, O_t, A_t, R_t$$

8.2 Markov Decision processes

Definition 8.1. A state S_t is **Markov** if and only if

$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, \ldots, S_t]$$

Definition 8.2. A **Markov Process** is a tuple $< \mathcal{S}, \mathcal{P} >$

- $\mathcal{S}$ is a (finite) set of states
- $\mathcal{P}$ is a state transition probability matrix

$$\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s'|S_t = s]$$

Definition 8.3. A **Markov Reward Process** is a tuple $< \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma >$

- $\mathcal{S}$ is a (finite) set of states
- $\mathcal{P}$ is a state transition probability matrix

$$\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s'|S_t = s]$$

- $\mathcal{R}$ is a reward function, $\mathcal{R}_s = \mathbb{E}[R_{t+1}|S_t = s]$
- γ is a discount factor, $\gamma \in [0, 1]$

Definition 8.4. The **return** G_t is the total discounted reward from time-step t

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

The **discount factor** γ is the present value of future rewards. The value of receiving reward R after $k + 1$ time-steps is $\gamma^k R$

Definition 8.5. A **Markov Decision Process** is a tuple $< \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma >$

- $\mathcal{S}$ is a (finite) set of states
- $\mathcal{P}$ is a state transition probability matrix
- $\mathcal{A}$ is a finite set of actions

$$\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s'|S_t = s]$$

- $\mathcal{R}$ is a reward function, $\mathcal{R}_s = \mathbb{E}[R_{t+1}|S_t = s]$

- γ is a discount factor, $\gamma \in [0, 1]$

policy function $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$

- where the value of $\pi(s, a)$ represents the probability of taking action a under s
  - **stochastic** policy: $\pi(a|s) = \mathbb{P}[A_t = a|S_t = s] = \mathbb{P}[a|s]$
  - **deterministic** policy: $a = \pi(s), a \in \mathcal{A}$

the problem with the terminal state is **episodic**, otherwise **continuing**
In order to evaluate policy function π, define:

1. **value function** $V : \mathcal{S} \to \mathbb{R}$ where

$$\begin{aligned} V_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \dots |S_t = s] \\ &= \mathbb{E}_{a\sim\pi(s,\cdot)}[\mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \dots |S_t = s, A_t = a]] \\ &= \sum_{a\in A} \pi(s, a) q_\pi(s, a) \end{aligned}$$

2. **Q-value function/ Action-value Function** $q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, where

$$\begin{aligned} q_\pi(s, a) &= \mathbb{E}_\pi[G_t|S_t = s, A_t = a] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \dots |S_t = s, A_t = a] \\ &= \mathbb{E}_{s'\sim Pr(\cdot|s,a)}[R(s, a, s') + \gamma \mathbb{E}_\pi[R_{t+2} + \gamma R_{t+3} + \dots |S_{t+1} = s]] \\ &= \sum_{s'\in S} Pr(s'|s, a)[R(s, a, s') + \gamma V_\pi(s')] \end{aligned}$$

8.3 policy improvement and policy evaluation

RL problem: given Markov decision process $MDP = \{\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma\}$, find an optimal policy π^* which maximizes the value of $V_{\pi^*}(s)$ for any $s \in \mathcal{S}$

8.3.1 value-based solution method

1. policy improvement

   let π, π' be any deterministic policies, for all $s \in \mathcal{S}$:

$$q_\pi(s, \pi'(s)) \geq q_\pi(s, \pi(s))$$

that is for all states $s \in \mathcal{S}$

$$V_{\pi'}(s) \geq V_{\pi}(s)$$

then π' must be as goog as, or better than π

$$\pi'(s) = \arg\max_{a} q_{\pi}(s, a) \quad \text{for all } s \in \mathcal{S}$$

2. policy evaluation

   iterative calculation of bellman equation

   (a) dynamic programming

   $$V_{\pi}(s) \leftarrow \sum_{a \in A} \pi(s, a) q_{\pi}(s, a)$$

   (b) Monte Carlo sampling

   Select different initial states, sample several tracks according to the current strategy π,and record their set as D.

   calculate the corresponding reward $G_1, \ldots, G_k$ for each occurence of s in D

   $$V_{\pi}(s) \leftarrow \frac{1}{k} \sum_{i=1}^{k} G_i$$

8.4 q-learning for RL