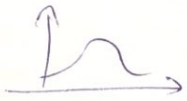
ML

Probabilities

+) if x is discrete $\sum_x p(x) = 1 \Rightarrow \forall p(x) \leq 1$

+) if x is continuous: theoretically $p(x) = 0 \Rightarrow \int p(x)\,dx = 1$ 

$\Rightarrow$ Là hàm mật độ xác suất – probability density function (pdf)

mật độ xác suất $p(x)$ có thể nhận giá trị dương bất kỳ miễn $\int p(x) = 1$

+) Definition < joint probability $p(x_i, y_i)$ | marginal probability $p(x_i)$ | conditional probability $p(y_i \mid x_i)$

(full: $p(X = x_i, Y = y_i)$, $p(X = x_i)$, $p(Y = y_i \mid X = x_i)$)

+) Bayes rule: $p(y_i) \cdot p(x_i \mid y_i) = p(x_i, y_i) = p(x_i) \cdot p(y_i \mid x_i) \Rightarrow p(y_i \mid x_i) = \frac{p(y_i) \cdot p(x_i \mid y_i)}{p(x_i)}$

posterior = likelihood × Prior / Normalization Factor $= \frac{p(x_i \mid y_i)\, p(y_i)}{\sum_y p(x_i \mid y_i)\, p(y_i)}$

+) Marginalization discrete: $p(x) = \sum_y p(x, y)$

continuous: $p(x) = \int p(x, y)\,dy$

Nếu 2 biến độc lập $\Rightarrow$ Independence $\Rightarrow \begin{cases} p(x \mid y) = p(x) \\ p(y \mid x) = p(y) \end{cases} \Leftrightarrow p(x, y) = p(x)\,p(y)$

- Expectation: $E[x] = \sum_x x\,p(x) \left(= \int x\,p(x)\,dx \right)$
- Với $f(\cdot) \Rightarrow E[f(x)] = \sum_x f(x)\,p(x) \left(= \int f(x)\,p(x)\,dx \right)$
- Measure variability $\Rightarrow$ < variance: $\mathrm{var}[f] = \mathbb{E}\left[(f(x) - \mathbb{E}[f(x)])^2\right] = \mathbb{E}[f(x)^2] - \mathbb{E}[f(x)]^2$
  covariance: $\mathrm{cov}[x, y] = \mathbb{E}_{x,y}[xy] - \mathbb{E}[x]\mathbb{E}[y]$ (matrix)
  $= \mathbb{E}_{x,y}[xy^T] - \mathbb{E}[x]\mathbb{E}[y^T]$
- Sum Rule: Sum of joint prob. = marginal prob.

  Product Rule: Product of marginal prob & conditional prob = joint prob

Probability distribution:

1) Bernoulli distribution: $x \in \{0, 1\}$ (2 classes) $\quad p(x) = Bern_x[\lambda]$

Tổng quát ⇓ $\lambda = [0, 1] \Rightarrow \begin{cases} p(x=1) = \lambda \\ p(x=0) = 1-\lambda \end{cases}$ hay $p(x) = \lambda^x (1-\lambda)^{1-x}$

Example = tổng xu

2) Categorical: $x \in \{e_1, e_2, \dots, e_k\}$ (k classes) ≈ one-hot vector

$p(x=k) = \lambda_k \qquad p(x) = Cat_x[\lambda]$

$$\Rightarrow p(x = e_k) = \prod_{j=1}^{k} \lambda_j^{x_j} = \lambda_k$$

với $x = [x_1, x_2, x_3 \dots, x_K] \Rightarrow (= \lambda_1^0 \lambda_2^0 \dots \lambda_k^1 \dots \lambda_K^0)$ vì $x_k = 1$ vì $x = e_k$

3) Univariate Normal Distribution: Gaussian

Tổng quát ⇓

$$p(x) = f(\mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right)$$

$$= Norm_x[\mu, \sigma^2]$$

4) Multivariate Normal Distribution: $\mu \in \mathbb{R}^p$, $\sigma^2 \Rightarrow \Sigma \in S_{++}^p$

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

$$= Norm_x[\mu, \Sigma]$$

5) Beta distribution ⇒ một tham số cho 1 distribution khác (VD: Bernoulli)

⇓ Dirichlet Distribution (VD một Categorical Distribution)

MAP: MLE

Maximum Likelihood Estimation

$$\theta = \arg\max_\theta p(x_1, \dots x_N | \theta)$$

Independence assumption $\theta = \arg\max_\theta \prod_{n=1}^{N} p(x_n | \theta)$

Maximum log-likelihood $\theta = \arg\max_\theta \sum_{n=1}^{N} \log(p(x_n | \theta))$

Minimum negative log-likelihood

MAP

Maximum a Posteriori (Bayes rule)

$$\theta = \arg\max_\theta \left[\prod_{i=1}^{N} p(x_i | \theta) p(\theta)\right]$$

Marginals of Gaussian are again Gaussian

Underestimation problem

$$\mathbb{E}[\mu_{ML}] = \mu$$

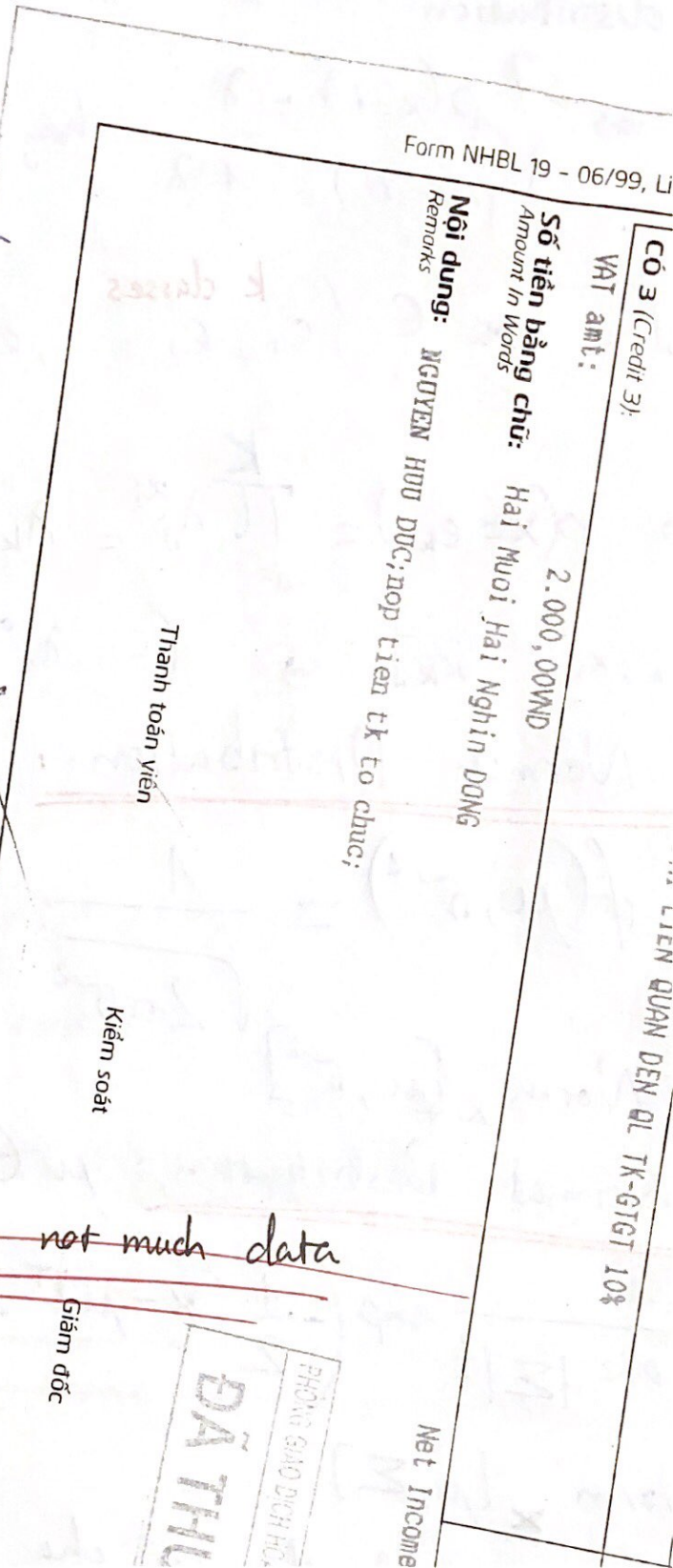
$$\mathbb{E}[\sigma^2_{ML}] = \left(\frac{N-1}{N}\right)\sigma^2$$

$$\Rightarrow \tilde{\sigma}^2 = \frac{N}{N-1}\sigma^2_{ML}$$

$$= \frac{1}{N-1}\sum_{n=1}^{N}(x_n - \hat{\mu})^2$$

MLE suffers when there is not much data

=> Solve by MAP



Naive Bayes Classifier

Independence assumption ⇒ Naive: ~~$p(x|c)$~~ =

$$c = \arg\max_{c \in \mathbb{C}} p(c|\mathbf{x})$$

$$= \arg\max_{c} p(\mathbf{x}|c)\,p(c)$$

x < feature vector ⇒ Multinomial
  binary ⇒ Bernoulli

brein when true ⇒ Gaussian Naive Bayes

⊛ Minimize the expected loss: $\mathbb{E}[L] = \sum_k \sum_j \int_{R_j} L_{kj}\, p(\mathbf{x}, C_k)\, d\mathbf{x}$

by choosing regions R_j such that: $\mathbb{E}[L] = \sum_k L_{kj}\, p(C_k|\mathbf{x})$

Views on the Decision Problem

+) **Generative Methods:** First determine the class-conditional densities & separately infer the prior class prob ⇒ Bayes Theorem ⇒ class membership

$p(\mathbf{x}|C_k)\,p(C_k) \Rightarrow y_k(\mathbf{x})$ Example: Mixture of Gaussians

+) **Discriminative Methods:** First solve the inference problem of determined the posterior class probabilities

Example:

2 Class C_1 C_2

2 Decision α_1 α_2

Loss: $L(\alpha_j | C_k) = L_{kj}$

Expected loss = Risk (R)

$$E_{\alpha_1}[L] = R(\alpha_1 | x)$$

$$= L_{11} p(C_1|x) + L_{21} p(C_2|x)$$

$$E_{\alpha_2}[L] = R(\alpha_2|x)$$

$$= L_{12} p(C_1|x) + L_{22} p(C_2|x)$$

Choose α_1 if

$$R(\alpha_1|x) < R(\alpha_2|x)$$

Form NHBL 01 - 06/99, Liên 1: Lưu, Liên 2: KH

quận Đống Đa, thành phố Hà Nội
Mã VAT: 0100112437006

Ngày (Date)....

So HD - Invoice No: 210818.

ĐỀ NGHỊ GHI CÓ TÀI KHOẢN (Please Credit account):

SỐ TIỀN

SỐ TK (A/C No): 00113715447217	Bằng số (In figures):
TÊN TK (A/C Name): [illegible]	Bằng chữ (In words):
ĐỊA CHỈ (Address): 29 [illegible]	
NGÂN HÀNG (With Bank): [illegible]	

NGƯỜI NỘP TIỀN (Depositor):

TRONG ĐÓ (In which:

Họ và tên (Full name): [illegible]

Địa chỉ (Address): [illegible]

Nội dung nộp (Details): [illegible]

KÝ TÊN (Sig

DÀNH CHO NGÂN HÀNG (For Bank's Use only) MÃ VAT:

002 HAN0704 21Aug1815:46:13 1100 CSD CA CSH DEP
7E49.0072 0-001-1-37-184471-7/ 04261
SO TIEN MAT NOP: USD **********250,00 23.210,00/ 23.200,00 Thành toán
TIEN GHI CO TK: USD **********250,00 23.300,00

non parametric

Probability Density Estimation

- Histograms: $p_i = \frac{n_i}{N \Delta_i}$ 

often $\Delta_i = \Delta$, D = dimension

no of bins: $\in \mathcal{O}(k^D)$

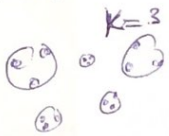
Δ is the smoothing factor

- $P = \int_{\mathcal{R}} p(y)\,dy \approx p(x)\,V$ when $\mathcal{R}$ is sufficiently small

$\Rightarrow p(x) \approx \frac{K}{NV}$ $\left(p = \frac{K}{N}\right)$

determine K, fixed V ← Kernel Methods

→ fixed K, determine V K-Nearest Neighbor



$p(x) \approx \frac{K}{NV^*}$

K is smoothing factor

Too much bias = too smooth

variance = too not smooth enough

⇒ including parametric methods mixture models

Parzen Window

$k(u) = \begin{cases} 1 & |u_i| \le \frac{1}{2}h,\ i = 1, \dots, D \\ 0 & \text{else} \end{cases}$

$K = \sum_{n=1}^{N} k(x - x_n)$; $V = \int k(u)\,du = h^D$

$\Rightarrow p(x) \approx \frac{K}{NV} = \frac{1}{Nh^D} \sum_{n=1}^{N} k(x - x_n)$

Gaussian kernel

$k(u) = \frac{1}{\sqrt{2\pi h^2}} \exp\left\{\frac{-u^2}{2h^2}\right\}$

$K = \sum_{n=1}^{N} k(x - x_n)$; $V = \int k(u)\,du = 1$

$\Rightarrow p(x) \approx \frac{1}{N} \sum_{n=1}^{N} \frac{1}{(2\pi)^{D/2} h} \exp\left\{\frac{-\|x - x_n\|^2}{2h^2}\right\}$

h is smoothing factor

General: $k(u) \ge 0$ $\int k(u)\,du = 1$

Size of the hypersphere is proportional to h^2

Mixture distributions = multi parametric models

Mixture of Gaussians (MoG)

$p(x|\theta) = \sum_{j=1}^{M} p(x|\theta_j) p(j)$

Generative Model

$p(x|\theta_j) = \frac{1}{\sqrt{2\pi}\sigma_j} \exp\left\{\frac{-(x-\mu_j)^2}{2\sigma_j^2}\right\}$; $p(j) = \pi_j$, $\sum \pi_j = 1$

$p(x|\theta_j) = \frac{1}{(2\pi)^{D/2} |\Sigma_j|^{1/2}} \exp\left\{-\frac{1}{2}(x-\mu_j)^T \Sigma_j^{-1} (x-\mu_j)\right\}$

$p(x|\theta_j)$: mixture component

$p(j) = \pi_j$: weight of ↑

- K-means clustering: 3 steps +) pick K centroids +) assign sample +) adjust centroids ↙ till no change

local optimum, depends on initialization

sensitive to outliers, detects spherical clusters only

⇒ Image compression

- EM Clustering (Expectation - Maximization)

E-Step: $\gamma_j(x_n) = \frac{\pi_j \mathcal{N}(x_n | \mu_j, \Sigma_j)}{\sum_{k=1}^{K} \pi_k \mathcal{N}(x_n | \mu_k, \Sigma_k)}$ ⇒ $\forall j = 1 \dots K, n = 1 \dots N$

responsibility of component j for x

N	datas
K	Gaussians

M-Step: $\hat{N}_j = \sum_{n=1}^{N} \gamma_j(x_n)$; $\hat{\mu}_j = \frac{1}{\hat{N}_j} \sum_{n=1}^{N} \gamma_j(x_n) x_n$

$\hat{\pi}_j = \frac{\hat{N}_j}{N}$ ⇒ w

$\hat{\Sigma}_j = \frac{1}{\hat{N}_j} \sum_{n=1}^{N} \gamma_j(x_n)(x_n - \hat{\mu}_j)(x_n - \hat{\mu}_j)^T$

Regularization: $\Sigma + \sigma_{min} I$

Init μ_j with K-Means

Hard-assignments: 1 point to 1 class ⇒ K-means

Soft ——— : Each point ⇒ prob to fall into many classes ⇒ EM

EM needs more iteration cause there are more parameters

Machine Learning

Linear Regression

$$w = [w_0, w_1, \dots w_n]^T$$

$$\bar{x} = [1, x_0, \dots, x_n] \quad (\text{x bar})$$

$$y \approx \hat{y} = \bar{x} . w \quad (\text{y hat})$$

$$\Rightarrow \frac{1}{2} e^2 = \frac{1}{2} (y - \bar{x} w)^2$$

$$\Rightarrow \mathcal{L}(w) = \frac{1}{2} \sum_{i=1}^{N} (y_i - \bar{x}_i w)^2 \quad (\text{loss function})$$

$$\Rightarrow w^* = \arg\min_w \mathcal{L}(w)$$

$$\mathcal{L}(w) = \frac{1}{2} \| y - \bar{X} w \|_2^2$$

$$\bar{X} = \begin{bmatrix} \bar{x}_1 \\ \bar{x}_2 \\ \vdots \\ \bar{x}_n \end{bmatrix} \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

Solution: $\frac{\partial \mathcal{L}(w)}{\partial w} = \bar{X}^T (\bar{X} w - y) = 0$

$$\Leftrightarrow \bar{X}^T \bar{X} w = \bar{X}^T y$$

$$\Leftrightarrow w = (\bar{X}^T \bar{X})^{\dagger} \bar{X}^T y \quad (\text{dagger})$$

vì chưa chắc khả nghịch nên dùng giả nghịch đảo pseudo inverse

(?)

$$A^{\dagger} = (A^T A)^{-1} A^T$$

(!) Sensitive to outliers $\Rightarrow$ pre-processing

Multivariable: $\theta_{t+1} = \theta_t - \eta \nabla_\theta f(\theta_t)$ or $\theta_{t+1} = \theta_t - \eta \nabla_\theta f(\theta)$

$$\mathcal{L}(w) = \frac{1}{2N} \| y - \bar{X} w \|_2^2 \Rightarrow \nabla_w \mathcal{L}(w) = \frac{1}{N} \bar{X}^T (\bar{X} w - y)$$

Check derivative: $f'(x) \approx \frac{f(x+\varepsilon) - f(x-\varepsilon)}{2\varepsilon}$ (numerical gradient)

- Linear Discriminant Funes:

$$y(x) = \tilde{W}^T \tilde{x}$$

$$E(W) = \frac{1}{2} \sum_{n=1}^{N} \sum_{k=1}^{K} (w_k^T x_n - t_{kn})^2$$

$$\tilde{W} = (\tilde{X}^T \tilde{X})^{-1} \tilde{X}^T . T$$

$$= \tilde{X}^+ . T$$

- Generalized Discriminants

$$y_k(x) = \sum_{j=1}^{M} w_{kj} \phi_j(x) + w_{k0} = \sum_{j=0}^{M} w_{kj} \phi_j(x), \quad \phi_0(x) = 1$$

Logistic Regression

$$f(x) = \theta(w^T x)$$

$$f(s) = \frac{1}{1+e^{-s}} \triangleq \sigma(s) \quad \text{sigmoid func}$$

$$s = \ln\left(\frac{\sigma}{1-\sigma}\right)$$

$$\sigma'(s) = \sigma(s)\,(1-\sigma(s))$$

$$J(w,x,y) = -\left(y_i \log z_i + (1-y_i)\log(1-z_i)\right)$$

with $z_i = f(w^T x_i)$ cross entropy error func

$$\Rightarrow \frac{\partial J}{\partial w} = (z_i - y_i)\,x_i$$

$$\Rightarrow w = w + \eta(y_i - z_i)\,x_i$$

- Require less parameters (only M: dimensions) compared to ... Gaussians .. $\frac{M(M+5)}{2}+1$

Softmax Regression

Multinomial Logistic Regression, Maximum Entropy Classifier

$$a_i = \frac{\exp(z_i)}{\sum_{j=1}^{C} \exp(z_j)}$$

so that $\begin{cases} a_i > 0 \\ \sum a_i = 1 \\ z_m > z_n \Leftrightarrow a_m > a_n \text{ (order)} \end{cases}$

- when z_i too big

$$\frac{\exp(z_i)}{\sum_{j=1}^{C} \exp(z_j)} = \frac{\exp(z_i - c)}{\sum_{j=1}^{C} \exp(z_j - c)}$$

$$c = \max_i (z_i)$$

$$J(w,x,y) = -\sum_{j=1}^{N}\sum_{j=1}^{C} y_{ji} \log(a_{ji})$$

$$J = -\sum_{n=1}^{N}\sum_{k=0}^{1}\left\{\mathbb{I}(t_n=k)\ln P(y_n=k \mid x_n; w)\right\} \Rightarrow E(w) = -\sum_{n=1}^{N}\sum_{k=1}^{K}\left\{\mathbb{I}(t_n=k)\ln\frac{\exp w_k^T x}{\sum_{j=1}^{K}(\exp w_j^T x)}\right\}$$

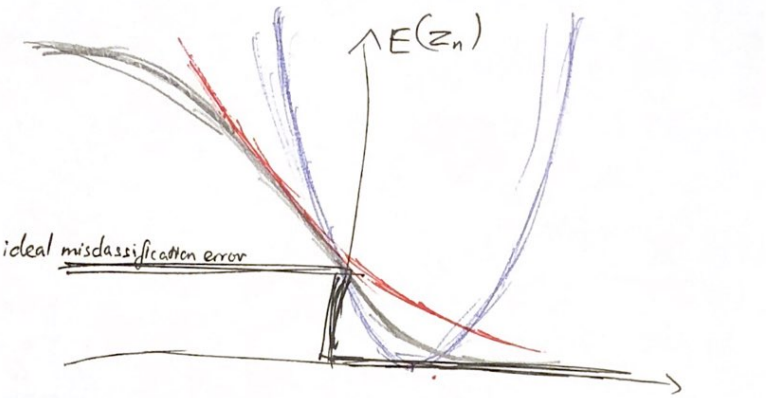
$$\frac{\partial J_i(w)}{\partial w} = x_i e_i^T = x_i\,(a_{ji} - y_{ji})^T$$

$$w = w + \eta\, x_i\,(y_i - a_i)^T$$

Error functions

+) Ideal misclassification error:
gradient = 0 ⇒ cannot use gradient descent
⇒ Count wrong-classified points

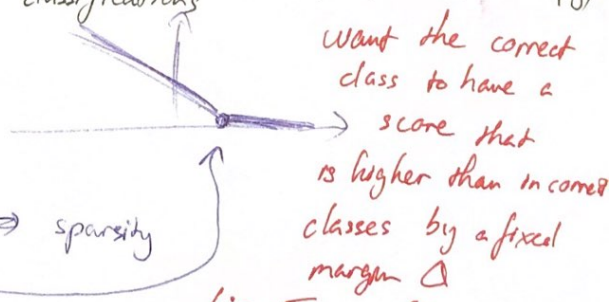
+) Squared error - L_2 loss
Leads to closed form solutions
sensitive to outliers
Penalize "too correct" data points



+) Cross entropy error
Concave function ⇒ unique minimum exists
Robust to outliers, error increases only roughly linear
No closed-form solution, requires iterative method

+) Squared error on sigmoid/tanh
No penalty for "too-correct" points
Zero gradient for confidently incorrect classifications
⇒ Do NOT use L_2 loss with sigmoid outputs (instead cross-entropy)

+) Hinge error $\sum_n [1 - t_n\, y(x_n)]$
robust to outliers
zero error for points outside margin ⇒ sparsity
not differentiable around $z_n = 1$



Want the correct class to have a score that is higher than incorrect classes by a fixed margin Δ

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)$$

s_j: other classes score; s_{y_i}: real class score

+) L_1, L_0 loss
median No wrong points

$$L_1 = \sum |t - y|$$

$$L_2 = \sum (t - y)^2$$

Support Vector Machine

ML

$$\text{margin} = \min_n \frac{y_n(w^T x_n + b)}{\|w\|_2}$$

$$\Rightarrow (w,b) = \arg\max_{w,b} \left\{ \min_n \frac{y_n(w^T x_n + b)}{\|w\|_2} \right\} = \arg\max_{w,b} \left\{ \frac{1}{\|w\|_2} \min_n y_n(w^T x_n + b) \right\}$$

$\Rightarrow$ **Problem:** $(w,b) = \arg\min_{w,b} \frac{1}{2}\|w\|_2^2$

subject to $1 - y_n(w^T x_n + b) \le 0, \forall n = 1, 2, \dots, N$

Primal formulation of SVMs:

$$\mathcal{L}(w,b,\lambda) = \frac{1}{2}\|w\|_2^2 + \sum_{n=1}^{N} \lambda_n (1 - y_n(w^T x_n + b)) \quad \lambda_n \ge 0 \ \forall n$$

$$\frac{\partial \mathcal{L}(w,b,\lambda)}{\partial w} = 0 \Rightarrow w = \sum_{n=1}^{N} \lambda_n y_n x_n$$

$$\frac{\partial \mathcal{L}(w,b,\lambda)}{\partial b} = 0 \Rightarrow \sum_{n=1}^{N} \lambda_n y_n = 0$$

$$\Rightarrow g(\lambda) = \sum_{n=1}^{N} \lambda_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} \lambda_n \lambda_m y_n y_m x_n^T x_m$$ (Dual formulation of SVM)

Set: $V = [y_1 x_1, y_2 x_2, \dots y_N x_N]$

$\Rightarrow g(\lambda) = -\frac{1}{2} \lambda^T V^T V \lambda + 1^T \lambda$ is concave with $\lambda \ge 0$, $\sum_{n=1}^{N} \lambda_n y_n = 0$

$\Rightarrow$ Find λ by solving Quadratic Programming

- Then use KKT to find w, b

$$S = \{n : \lambda_n \ne 0\} \Rightarrow \begin{cases} b = \frac{1}{N_S} \sum_{n \in S} \left(y_n - \sum_{m \in S} \lambda_m y_m x_m^T x_n \right) \\ w = \sum_{m \in S} \lambda_m y_m x_m \end{cases}$$

KKT conditions: $\begin{cases} \lambda_i \ge 0 \\ \lambda_i f(x) = 0 \\ f_i(x) \le 0 \end{cases}$ if we have $\mathcal{L}(\dots) = \dots + \lambda_i f_i(x)$

Hard Margin SVM: $(w,b) = \arg\min_{w,b} \frac{1}{2}\|w\|_2^2$

subject to: $y_n(w^T x_n + b) \geq 1 \quad \forall n$

Soft Margin SVM

$\frac{1}{2}\|w\|_2^2 + C\sum_{n=1}^{N} \xi_n$ — ξ_n: slack variable

"Soft" constraints: $y_n(w^T x_n + b) \geq 1 - \xi_n \Leftrightarrow 1 - \xi_n - y_n(w^T x_n + b) \leq 0 \quad \forall n$

(and $\xi_n \geq 0 \quad \forall n$)

C and margins are inproperttional: $C \uparrow \Rightarrow$ margin $\downarrow$; $C \downarrow \Rightarrow$ margin $\uparrow$

$$\Rightarrow (w,b,\xi) = \arg\min_{w,b,\xi} \frac{1}{2}\|w\|_2^2 + C\sum_{n=1}^{N} \xi_n$$

subject to $1 - \xi_n - y_n(w^T x_n + b) \leq 0 \quad \forall n$

$-\xi_n \leq 0 \quad \forall n$

$$\Rightarrow \mathcal{L}(w,b,\xi,\lambda,\mu) = \frac{1}{2}\|w\|_2^2 + C\sum_{n=1}^{N}\xi_n + \sum_{n=1}^{N}\lambda_n(1 - \xi_n - y_n(w^T x_n + b)) - \sum_{n=1}^{N}\mu_n\xi_n$$

với $\lambda \geq 0$ và $\mu \geq 0$

$$g(\lambda,\mu) = \inf_{w,b,\xi} \mathcal{L}(w,b,\xi,\lambda,\mu)$$

$\frac{\partial \mathcal{L}}{\partial w} = 0 \Leftrightarrow w = \sum_{n=1}^{N} \lambda_n y_n x_n$; $\frac{\partial \mathcal{L}}{\partial b} = 0 \Leftrightarrow \sum_{n=1}^{N} \lambda_n y_n = 0$; $\frac{\partial \mathcal{L}}{\partial \xi_n} = 0 \Leftrightarrow \lambda_n = C - \mu_n$

$$\Rightarrow g(\lambda,\mu) = \sum_{n=1}^{N}\lambda_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N}\lambda_n\lambda_m y_n y_m x_n^T x_m$$

$$\Rightarrow \lambda = \arg\max_{\lambda} g(\lambda) \text{ subject to } \sum_{n=1}^{N}\lambda_n y_n = 0$$

$0 \leq \lambda_n \leq C \quad \forall n$

$M = \{n : 0 < \lambda_n < C\} \Rightarrow b = \frac{1}{N_M}\sum_{n \in M}\left(y_n - \sum_{m \in S}\lambda_m y_m x_m^T x_n\right)$

$S = \{m : 0 < \lambda_m \leq C\} \Rightarrow w = \sum_{m \in S}\lambda_m y_m x_m$

- $\lambda_n = 0, \xi_n = 0 \Rightarrow$ safe points
- $0 < \lambda_n < C, \xi_n = 0 \Rightarrow$ marginal points
- $\lambda_n = C$: $\xi_n \leq 1$: still correct; $\xi_n > 1$: wrong

Kernel Support Vector Machine

/mở sổ/

Mercer's condition: kernel func must satisfy (theoretically)

$$\sum_{n=1}^{N}\sum_{m=1}^{N} k(x_m, x_n) c_n c_m \geq 0 \quad \forall c_i \in \mathbb{R}, i = 1, 2, \dots, N$$

Thực hành, nhiều kernel ko thỏa mãn, nhưng vẫn chấp nhận được

+) $k(x, z) = x^T z$ (linear)

+) $k(x, z) = (r + \gamma x^T z)^d$ polynomial

+) $k(x, z) = \exp(-\gamma \|x - z\|_2^2), \gamma > 0$ radial basis function

+) $k(x, z) = \tanh(\gamma x^T z + r)$ sigmoid

ML

The Lagrange dual function

Primal prob $x^* = \arg\min_x f_0(x)$ subject to $f_i(x) \le 0, i = 1, \dots, m$
$h_j(x) = 0, j = 1, \dots p$

$$\Rightarrow \mathcal{L}(x, \lambda, \nu) = f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{j=1}^{p} \nu_j h_j(x)$$

λ_i: dual variables

x^* thỏa mãn $\nabla_{x,\lambda} \mathcal{L}(x, \lambda) = 0$

Dual function: $g(\lambda, \nu) = \inf_{x \in D} \mathcal{L}(x, \lambda, \nu)$ infimum

With $\lambda_i \ge 0, \forall i, \nu$: $g(\lambda, \nu) \le p^*$ (optimal value of f_0)

$\Rightarrow$ The Lagrange Dual problem:

$$\lambda^*, \nu^* = \arg\max_{\lambda, \nu} g(\lambda, \nu) \text{ subject to } \lambda \succeq 0$$

↳ $g(\lambda, \nu)$ is concave $\Rightarrow$ Lagrange Dual Prob is a convex prob regardless of primal prob

↳ λ^* and ν^* are called dual optimal / optimal Lagrange multipliers

- Weak duality: $\max g(\lambda, \nu) = d^* \le p^* = \min f_0(x)$
  $p^* - d^*$: optimal duality gap
- Strong duality & Slater's constraint qualification
  if $p^* = d^* \Rightarrow$ strong duality

If $x = \arg\min_x f_0(x)$ subject to $f_i(x) \le 0$ $i = 1, \dots n$, $Ax = b$

$f_0, f_1 \dots f_n$ are convex $\Rightarrow$ usually (not always) have strong duality

⊛ Slater's condition:

+) A feasible point is strictly feasible if: $\begin{cases} f_i(x) < 0, & i = 1, \dots n \\ Ax = b \end{cases}$

⇒ Slater's condition / theorem: if a strictly feasible point exists and primal problem is convex then strong duality holds

when strong duality happens: $f_0(x^*) = g(\lambda^*, \nu^*)$

$$\Rightarrow \sum_{i=1}^{m} \lambda_i^* f_i(x^*) = 0 \Rightarrow \lambda_i f_i(x^*) = 0 \quad i = 1, \dots, m$$

Complementary slackness

⊛ Karush – Kuhn – Tucker conditions

f_0 is not necessary convex, strong duality holds, then x^*, λ^*, ν^* must hold:

$$f_i(x^*) \le 0, \quad i = 1, \dots, m$$
$$h_j(x^*) = 0, \quad j = 1, \dots, p$$
$$\lambda_i^* \ge 0, \quad i = 1, \dots, m$$
$$\lambda_i^* f_i(x^*) = 0, \quad i = 1, \dots, m$$
$$\nabla f_0(x^*) + \sum_{i=1}^{m} \lambda_i^* \nabla f_i(x^*) + \sum_{j=1}^{p} \nu_j^* \nabla h_j(x^*) = 0 \iff ⊛$$

These are necessary conditions

If f_0 convex, then KKT conditions are necessary and sufficient conditions for solutions

ML

Convexity

1) Convex sets

+) Def 1: line connect 2 points of the set lies within the set

+) Def 2: C is convex if for $\forall x_1, x_2 \in C$:

$$x_\theta = \theta x_1 + (1-\theta) x_2 \in C \quad \forall \; 0 \le \theta \le 1$$

+) Hyperplanes: $a_1 x_1 + a_2 x_2 + \dots + a_n x_n = a^T x = b$

$b, a_i \; (i \in \mathbb{N}) \in \mathbb{R}$

Hyperplane is also a convex set

+) Half space: $a_1 x_1 + a_2 x_2 + \dots + a_n x_n = a^T x \le b$

Half space is a convex set

+) Positive definite: $x^T A^{-1} x \ge 0 \quad \forall x \in \mathbb{R}^n$ | $A \succ 0$

(ii $\exists A^{-1} \Rightarrow \forall \lambda \neq 0$ tri riêng $\neq 0$)

Positive semi definite: $x^T B x \ge 0 \quad \forall x \in \mathbb{R}^n$ | $B \succeq 0$

+) Intersection of convex sets is a convex set

$\Rightarrow$ Intersection of halfspaces & hyperplanes is polyhedra (is convex)

+) Convex combination of $x_1, x_2 \dots, x_k$: x if

$x = \theta_1 x_1 + \theta_2 x_2 + \dots \theta_k x_k$, with $\theta_1 + \theta_2 + \dots + \theta_k = 1$

Convex hull of a set $(x_1, \dots x_k)$ is a set of all possible convex combination of that set

Convex hull of a set is the smallest convex set that contains that set

↳ 2 convex sets $\mathcal{C}$ and $\mathcal{D}$ are disjoint then exist a, b:

$$\begin{cases} a^T x \le b & \forall x \in \mathcal{C} \\ a^T x \ge b & \forall x \in \mathcal{D} \end{cases}$$

Set of all x that $a^T x - b = 0$ is a hyperplane that separate $\mathcal{C}$ and $\mathcal{D}$ ⇒ <u>separating hyperplane</u>

Convex function

<u>Definition</u> $f: \mathbb{R}^n \to \mathbb{R}$ is a convex func if $\text{dom} f$ is a convex set and

$$f(\theta x + (1-\theta)y) \le \theta f(x) + (1-\theta) f(y)$$

$\forall x, y \in \text{dom} f, \quad 0 \le \theta \le 1$

Tập xác định của $f(\cdot)$: $\text{dom} f$

↳ If $f(\theta x + (1-\theta)y) < \theta f(x) + (1-\theta) f(y)$ ⇒ strictly convex

Nếu có cực trị thì đó là duy nhất và cũng là global minimum

Affine function : $f(x) = a^T x + b$ is both { convex / concave

if variable is a matrix $X \Rightarrow f(X) = \text{trace}(A^T X) + b$

Quadratic form : $f(x) = x^T A x + b^T x + c$ convex if $A \succeq 0$

concave if $-A \succeq 0$

Hàm thỏa mãn 3 ĐK của Norm ⇒ <u>convex function</u>

α - sub level set of $f: \mathbb{R}^n \to \mathbb{R}$: $C_\alpha = \{x \in \text{dom} f \mid f(x) \le \alpha\}$

Check if f is convex:

1) First-order condition:

f differentiable with convex domain

iff (if and only if)

$$f(x) \geq f(x_0) + \nabla f(x_0)^T (x - x_0) \quad \forall x, x_0 \in \text{dom} f$$

2) Second-order condition: dùng phổ biến hơn:

Hessian: $\nabla^2 f(x) \succeq 0$

Optimization

$$x^* = \arg\min_x f_0(x) \quad \text{subject to} \begin{cases} f_i(x) \leq 0, & i = 1, 2, \ldots, m \\ h_j(x) = 0, & j = 1, 2, \ldots, p \end{cases}$$

$$\mathcal{D} = \bigcap_{i=0}^{m} \text{dom} f_i \cap \bigcap_{j=1}^{p} \text{dom} h_j$$

$\Rightarrow$ set of all x satisfying all (in)equality constraints $\Rightarrow$ feasible set

infeasible point $\not\in$ feasible point

⊗ Convex optimization problem:

$$x^* = \arg\min_x f_0(x) \quad \text{subject to} \begin{cases} f_i(x) \leq 0, & i = 1, 2, \ldots, m \\ a_j^T x - b_j = 0, & j = 1 \ldots, p \end{cases}$$

f_0 is convex func, f_i is convex func, h_j is affine func

$\Downarrow$

$f_i(x) \leq 0 \Rightarrow$ 0 sublevel set of f_i

all x: $h_j(x) = 0 \Rightarrow$ hyperplane

$\Rightarrow$ We optimize a convex function in a convex set domain

---

Linear Programming (Quy hoạch tuyến tính)

f_0, f_i are affine funcs

`cvxopt.solvers.lp`

$$x = \arg\min_x c^T x + d \quad \text{subject to} \begin{cases} Gx \preceq h \\ Ax = b \end{cases}$$

A Standard form LP: $x = \arg\min_x c^T x$ subject to $\begin{cases} Ax = b \\ x \geq 0 \end{cases}$

Quadratic Programming

cvxopt.solvers.qp

$$x = \arg\min_x \frac{1}{2} x^T P x + q^T x + r \quad \text{subject to} \quad \begin{matrix} Gx \leq h \\ Ax = b \end{matrix}$$

P nửa xác định dương $P \geq 0$

→ Khi $P = 0 \Rightarrow$ QP là LP

Geometric Programming

cvxopt.solvers.gp

+) Function $f: \mathbb{R}^n \to \mathbb{R}$ with $\text{dom} f = \mathbb{R}^n_{++}$ (all element > 0)

$$f(x) = c\, x_1^{a_1} x_2^{a_2} \dots x_n^{a_n} \qquad c > 0, \quad a_i \in \mathbb{R}$$

f is a monomial function

+) $$f(x) = \sum_{k=1}^{K} c_k x_1^{a_{1k}} x_2^{a_{2k}} \dots x_n^{a_{nk}} \qquad c_k > 0$$

f is a posynomial function

$$x = \arg\min_x f_0(x) \quad \text{subject to} \quad f_i(x) \leq 1,\ i = 1, 2 \dots m$$
$$h_j(x) = 1,\ j = 1, 2 \dots p$$

where f_0, f_i are posynomials and h_j are monomials

$\Rightarrow$ geometric programming ($x > 0$ is hidden)

Set $\begin{cases} x_i = e^{y_i} \\ y_i = \log(x_i) \end{cases} \Rightarrow f_0(x) = \exp(a^T y + b)$

MK

Ensemble of models

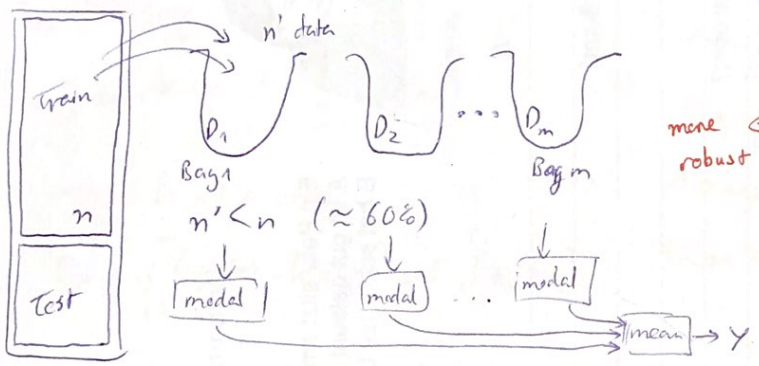
(also as additive model)

Bagging

Bootstrap aggregating

average ~ 63%

Data



random with replacement:

Bag i: can have multiple times data x_i

Simple, easy to implement, commonly used

Boosting

1) Ada boost ✪ (exponential error)
2) Gradient boosting
3) Xg boost (extreme gradient boosting)
4) Gentle Boost (cross entropy error) ⇐ more robust

Steps for Ada boost:

1) 1st weak classifier
2) Calculate error $J \Rightarrow \varepsilon$
   (in case of normalization w $\Rightarrow \sum w = 1 \Rightarrow \varepsilon = J$)
3) Calculate amount of say α from ε
4) Update w with α, (normalize w)
5) 2nd weak classifier with new weight w or sample on new w

$$-g(x_i) = -\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}$$

⇒ Regression model: h_i for $(x_i, -g(x_i))$

⇒ $F := F + \rho h$, $\rho = 1$

Ada boost ✪

$$H(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m h_m(x)\right)$$

$$J_m = \sum_{n=1}^{N} w_n^{(m)} I\left(h_m(x) \neq t_n\right)$$

$$\varepsilon_m = \frac{\sum_{n=1}^{N} w_n^{(m)} I\left(h_m(x) \neq t_n\right)}{\sum_{n=1}^{N} w_n^{(m)}}$$ ← weighted error

$$\alpha_m = \ln\left(\frac{1-\varepsilon_m}{\varepsilon_m}\right)$$

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

$\frac{\partial E}{\partial h} = 0$ ⇒ $E = \sum_{n=1}^{N} \exp\left\{-t_n f_m(x_n)\right\}$

$f(x) = \frac{1}{M}\sum_{l=1}^{m} \alpha_l h_l(x)$

$\frac{\partial E}{\partial \alpha} = 0$ ⇒

Gradient boosting (Regression)

(x_i, y_i) & first model $F_1(x)$

⇒ fit $F_2(x)$ to $\left(x_i, \underbrace{y_i - F_1(x_i)}_{\text{Residuals} = h(x_i)}\right)$

$J = \sum L(y_i, F(x_i))$

$\Rightarrow F_{i+1} = F_i + h_i$

$L = \frac{(y_i - F_i)^2}{2} \Rightarrow -g = y_i - F_i(x_i)$

$L = |y - F|$ absolute loss } less sensitive to outliers

$L = \begin{cases} \frac{1}{2}(y-F)^2 & |y-F| \leq \delta \\ \delta(|y-F| - \delta/2) & |y-F| > \delta \end{cases}$ Huber loss } less sensitive to outliers

Bagesian Model Averagng: H different models $h = 1, \dots, H$ with prior probability $p(h)$

$$p(X) = \sum_{h=1}^{H} p(X|h)\,p(h)$$

Error Reduction

Ex: baggmg

$$y_{com}(x) = \frac{1}{M}\sum_{m=1}^{M} y_m(x) \quad \Rightarrow$$

Average error of individual models

$$E_{AV} = \frac{1}{M}\sum_{m=1}^{M} E_x\left[\varepsilon_m(x)^2\right]$$

$$y(x) = h(x) + \varepsilon(x)$$

$$E_x = \left[\left\{y_m(x) - h(x)\right\}^2\right] = E_x\left[\varepsilon_m(x)^2\right]$$

Average error of committee

$$E_{COM} = E_x\left[\left\{\frac{1}{M}\sum_{m=1}^{M} y_m(x) - h(x)\right\}^2\right]$$

$$= E_x\left[\left\{\frac{1}{M}\sum_{m=1}^{M} \varepsilon_m(x)\right\}^2\right]$$

$\Rightarrow$ if { errors have 0 mean: $E_x[\varepsilon_m(x)] = 0$

errors are uncorrelated: $E_x[\varepsilon_m(x)\,\varepsilon_j(x)] = 0$ $\Rightarrow E_{COM} = \frac{1}{M} E_{AV}$

(?) → unrealistic

However in general $E_{COM} < E_{AV}$

⊗ The weak classifiers have to be a bit unstable algo like decision trees, neural net, ~~linear discriminant func~~ not nearest neighbor, SVM, linear regression

Decision Trees

All non-leaf nodes have 2 child nodes ⇒ binary decision tree.

Iterative Dichotomiser 3 (ID3) only for categorical attribute (discrete)

Classification & Regression Tree (CART) for both categorical & continuous

---

6 question

Bin / multi valued

when node → leaf

deal impure nodes?

how select query

pruned?

missing attributes?

Impurity Measures:

+) Mis classification: $i(s_j) = 1 - \max_k p(C_k | s_j)$

→) Information gain: C class với N_c là số phần tử mỗi class

$$H(S) = -\sum_{c=1}^{C} \frac{N_c}{N} \log\left(\frac{N_c}{N}\right)$$ entropy tại node đó

chọn thuộc tính x: ⇒ K child node $S_1, \dots, S_k$ với m_k phần tử

entropy sum with weight $H(x, S) = \sum_{k=1}^{K} \frac{m_k}{N} H(S_k)$ ⇒ $G(x, S) = H(S) - H(x, S)$ Information gain

$$x^* = \arg\max_x G(x, S) = \arg\min_x H(x, S)$$

Reduction in entropy = gain in information

+) Gini Index: $i(s_j) = \sum_{k \neq l} p(C_k | s_j) p(C_l | s_j)$ variance impurity at node s_j

$$= \frac{1}{2}\left(1 - \sum_k p^2(C_k | s_j)\right)$$

$$H(x, s_j) = \sum_{k=1}^{K} \frac{m_k}{N} i(s_j) \Rightarrow x^* = \min H(x, s_j)$$

+) Chi-square = $\sqrt{\frac{(actual - expected)^2}{expected}}$

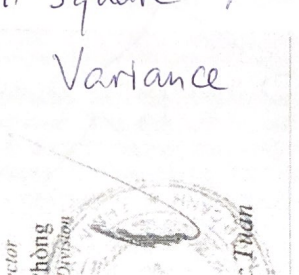
CHAID Chi-square Automatic Interaction Detector

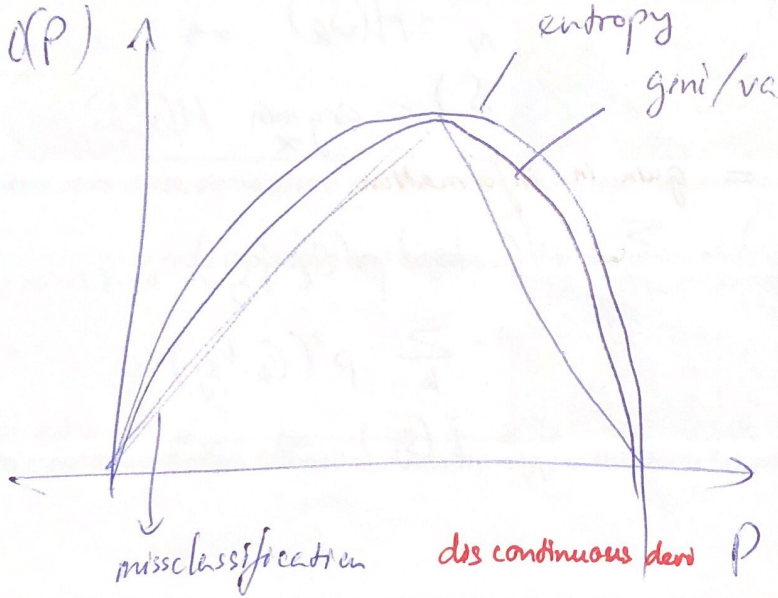
+) Reduction in Variance

$i(s_j) = \frac{\sum (x - \bar{x})^2}{n}$

arg max

arg min





} performs well
no big difference

⊛ Stopping & pruning
more
IMPORTANT

Stopping Conditions: (prevent over-fitting) = Pre pruning

- $H(S) = 0$: entropy $= 0 \Rightarrow \forall$ điểm $\in$ 1 class
- Số phần tử trong node $<$ 1 ngưỡng nào đó
  Class cho leaf node = class đa số
- Khoảng cách từ node → root $= c \Rightarrow$ hạn chế chiều sâu
- Tổng số leaf node vượt quá 1 ngưỡng
- Phân chia tiếp ko làm giảm entropy quá nhiều

+) Pruning: 1) Validation Set: Tỉa node nếu tăng Precision cho VS (Reduced error pruning method) — validation set

Post-pruning 2) Regularized loss function:

$$\mathcal{L} = \sum_{k=1}^{K} \frac{|S_k|}{|S|} H(S_k) + \lambda K$$

Minimum error	Pessimistic	Error complexity: $R(t) = r(t)\,p(t) + \alpha N_t$
Sensitive to no. classes least accurate in practice	most crude & quickest no need validation set but + caution needed	↙ error ↘ no. leafs Critical value: the value we choose to decide query at each nodes ⇒ depends on how the tree is created Reduced error: ... More Stable & accurate

Computational Complexity:

N data points
D dimensionality

⇒

Storage $O(N)$ — the worst case 1 data each leaf

Test runtime $O(\log N)$ — k step ⇒ $2^k = N$ ⇒ $k \in \Theta(\log N)$

Training time $O(DN^2 \log N)$

max $\Theta(DN \log N)$ times each node
max N nodes

Summary

Simple
Interpretable results
Resistance to over fitting

Memory consumption ⇒ suitable for problems with little data
Noisy weak classifiers
not generalized well
Sensitive outliers ☹
expensive learning step

Random forest

Handle missing values while maintain accuracy
Won't overfit

Not good with regression
Have little control to modify

Step: Sample from training set
Choose $m < M$ (input features). At each node, select m random data from M, to decide query attribute to split the node
Grow tree to largest, no pruning

Predict data: majority vote ⇒ classification
average ⇒ regression

⊗ Choose randomly K attributes.
Training time: $O(KN^2 \log N)$ $K \ll D$ $\left(K = \sqrt{N_s}\right)$

Typical: $\begin{cases} K = 10 & \text{root node} \\ K = 100d & \text{level-}d \text{ node} \end{cases}$

Adaboost

⊗ Ensembles of Classifiers: K classifiers, independent, errorprob < 0,5
⊗ Suitable for: unstable algorithms: decision trees, neural networks
⊗ good with stable methods: nearest neighbor, SVMs, linear regression

⊗ **Adaboost**

$$F(x) = \text{sign}\left(\sum_{m=1}^{M} \theta_m f_m(x)\right)$$

Initial weight for each data: $w(x_i, y_i) = \frac{1}{n}$

Classification error: $\varepsilon_m = E_{w_m}\left[1_{y \neq f(x)}\right]$

⇒ Update rule $\theta_m = \frac{1}{2} \ln\left(\frac{1-\varepsilon_m}{\varepsilon_m}\right)$

$$w_{m+1}(x_i, y_i) = \frac{w_m(x_i, y_i) \exp\left[-\theta_m y_i f_m(x_i)\right]}{Z_m}$$

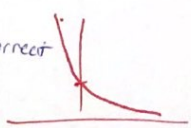
Z_m is normalization factor

Bayesian Model Averaging

$$p(X) = \sum_{h=1}^{H} p(X|h)\, p(h)$$

Exponential error used in AdaBoost

Fast convergence
No penalty for too correct
less robust to outliers



Weighted error function $J_m = \sum_{n=1}^{N} w_n^{(m)} I(h_m(x) \neq t_n)$

$$H(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m h_m(x)\right)$$

$$\varepsilon_m = \frac{\sum_{n=1}^{N} w_n^{(m)} I(h_m(x) \neq t_n)}{\sum_{n=1}^{N} w_n^{(m)}}$$

Normalize

$$E = \sum_{n=1}^{N} w_n^{(m)} \exp\left(-\frac{1}{2} t_n \alpha_m h_m(x_n)\right)$$

$$\Rightarrow w_n^{(m+1)} = w_n^{(m)} \exp\left[\alpha_m I(h_m(x_n) \neq t_n)\right]$$

Final classification

- **Bargeldauszahlungen mit der Debitkarte** [~~Deutsche Bank Card[4]~~] **~~an Geldautomaten~~ ausländischer Kooperationspartner[6] in Fremdwährung** - **Kontoauszüge im Online-Banking oder am Bankingterminal** In begrenzter Anzahl - **Ausgabe einer Debitkarte** [Deutsche Bank Card[4]] Über diese Anzahl hinausgehende Dienste werden getrennt in Rechnung gestellt.	
Zahlungen (ohne Karten)	
Überweisung [SEPA-Überweisung in EUR innerhalb des EWR[1]]	Je beleghafte Überweisung 1,50 EUR bzw. Überweisung über einen Mitarbeiter im telefonischen Kundenservice Je formlos erteilte Überweisung 4,00 EUR zzgl. Entgelt für beleghafte Überweisung zzgl. 1,50 EUR

amount of ... say

$$\alpha_m = \ln\left(\frac{1-\varepsilon_m}{\varepsilon_m}\right)$$

Gradient Descent

ML

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta f(\theta_t)$$

Check derivative $f'(x) \approx \frac{f(x+\varepsilon) - f(x-\varepsilon)}{2\varepsilon}$

+) **Momentum**:

- Init: $v_{dW_0} = 0$, $v_{db_0} = 0$
- Update: Calculate dW, db

$$\Rightarrow \begin{cases} v_{dW} = \beta v_{dW} + (1-\beta) dW \\ v_{db} = \beta v_{db} + (1-\beta) db \end{cases} \Rightarrow \begin{cases} W = W - \alpha v_{dW} \\ b = b - \alpha v_{db} \end{cases}$$

$\beta_1 = 0,9$

$\Rightarrow$ Nesterov accelerated gradient (NAG)

$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1})$

+) **RMSprop** (Root mean squared prop)

- Init $S_{dW_0} = 0$, $S_{db_0} = 0$
- Calculate dW, db
- Update: $\begin{cases} S_{dW} = \beta S_{dW} + (1-\beta) dW^2 \\ S_{db} = \beta S_{db} + (1-\beta) db^2 \end{cases} \Rightarrow \begin{cases} W = W - \alpha \frac{dW}{\sqrt{S_{dW}} + \varepsilon} \\ b = b - \alpha \frac{db}{\sqrt{S_{db}} + \varepsilon} \end{cases}$

$\beta_2 = 0,999$

$\varepsilon = 10^{-7}$

+) **Adam** (Adaptive moment estimation)

+ Init: $v_{dW}, S_{dW}, v_{db}, S_{db} = 0$
- Calculate dW, db
- $\begin{cases} v_{dW} = \beta_1 v_{dW} + (1-\beta_1) dW \\ v_{db} = \beta_1 v_{db} + (1-\beta_1) db \\ S_{dW} = \beta_2 S_{dW} + (1-\beta_2) dW \\ S_{db} = \beta_2 S_{db} + (1-\beta_2) db \end{cases} \Rightarrow \begin{cases} v_{dW}^{corrected} = \frac{v_{dW}}{1-\beta_1^t}, \; v_{db}^{cor.} = \frac{v_{db}}{1-\beta_1^t} \\ S_{dW}^{cor.} = \frac{S_{dW}}{1-\beta_2^t}, \; S_{db}^{cor.} = \frac{S_{db}}{1-\beta_2^t} \end{cases}$

$$\Rightarrow W := W - \alpha \frac{v_{dW}^{cor.}}{\sqrt{S_{dW}^{cor.}} + \varepsilon}; \; b := b - \alpha \frac{v_{db}^{cor.}}{\sqrt{S_{db}^{cor.}} + \varepsilon}$$

$\beta_1 = 0,9$

$\beta_2 = 0,999$

$\varepsilon = 10^{-7}$

7

Neural Network

Tips & tricks

- Shuffling: ✓
- Data augmentation: reshape, rescale, crops, zooming
  change color / color PCA

- Normalizing the inputs: ✓

  Convergence is fastest if: ±mean each input variable ≈ 0
  ± scale ⇒ same covariance

  Mean Cancellation ⇒ K-L expansion ⇒ covariance equalization
  (if possible)

- Leaky ReLU better a bit ReLU, ELU ✓

- Initialize Weights:

  Xavier Glorot: $W \sim U\left(0, \sqrt{\frac{6}{n_{in} + n_{out}}}\right)$ ✓

- Batch Norm(alization)

  Normalize after each layer (?)

  ⇒ Learn this moving average

- Drop out → Note: when applying, must multiply activation with the prob that it is set to 0

Neural Networks

(General stuffs)

Forward pass

$$y^{(0)} = x$$
$$z^{(k)} = W^{(k)} y^{(k-1)} \quad k = 1, \dots, \ell$$
$$y^{(k)} = g_k(z^{(k)})$$
$$y = y^{(\ell)}$$
$$E = L(t, y) + \lambda \Omega(W)$$

Backward Pass

$$h \leftarrow \frac{\partial E}{\partial y} = \frac{\partial}{\partial y} L(t, y) + \lambda \frac{\partial}{\partial y} \Omega$$

for $k = \ell \to 1$

$$h \leftarrow \frac{\partial E}{\partial z^{(k)}} = h \odot g'(y^{(k)})$$
$$\frac{\partial E}{\partial W^{(k)}} = h y^{(k-1)\top} + \lambda \frac{\partial \Omega}{\partial W^{(k)}}$$
$$h \leftarrow \frac{\partial E}{\partial y^{(k-1)}} = W^{(k)\top} h$$

Gradient Descent

$$w_{kj}^{(\tau+1)} = w_{kj}^{(\tau)} - \eta \left. \frac{\partial E(w)}{\partial w_{kj}} \right|_{w^{(\tau)}}$$

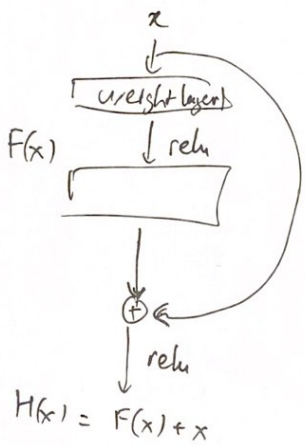
Momentum method

BPTT: Back propagation through time

$$\frac{\partial E_t}{\partial w_{ij}} = \sum_{1 \le k \le t} \left(\frac{\partial E_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial^+ h_k}{\partial w_{ij}} \right) ; \qquad E = \sum_{1 \le t \le T} E_t$$

$$\frac{\partial h_t}{\partial h_k} = \prod_{t \ge i \ge k} \frac{\partial h_i}{\partial h_{i-1}}$$

Residual



Word Embeddings

The idea of represent a word as a vector, but not a one-hot coding. In which similar words have similar vector values.

+) The trigram (n-gram method):
hugh amount of n-tuples of words ⇒ predict relative probabilities
Problems: scalability, partial observability

+) Word embedding: $x_{V\times 1} \longrightarrow W_{V\times d} \longrightarrow h_{D\times 1}$
1 of K encoding

+) word2vec : CBOW: only care which words occur
Syntactic (grammar) don't care about order of occurance
SKIP Gram: less weight to the more distance word
Semantic (meaning)

+) Hierachical softmax : organize word in binary search tree

CSiME

1, Search Algorithm

- Naive Bayes: $P(Y, f_1 \dots f_n)$

From data $\Rightarrow$ 1, $P(y_j, f_1, \dots f_n) = P(y_j)\, \pi_i\, P(f_i \mid y_j)$

2, $P(f_1 \dots f_n) = \sum_j P(y_j)\, \pi_i\, P(f_i \mid y_j)$

3, $P(Y \mid f_1 \dots f_n) = \dfrac{P(Y, f_1 \dots f_n)}{P(f_1 \dots f_n)}$

- Hinge loss: $s_j = f(x_i, W)_j$

$$L_i = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + \Delta)$$

- Regularization: L_1: $R(W) = \sum_k \sum_l |W_{k,l}|$

L_2 $R(W) = \sum_k \sum_l W_{k,l}^2$

Elastic Net $R(W) = \sum_k \sum_l \beta W_{k,l}^2 + |W_{k,l}|$

- Gradient descend $\hat{x} = x - \xi \nabla_x f(x)$

- Laplace smoothing: $P_{LAP,k}(x) = \dfrac{c(x) + k}{N + k|X|}$

($c(x)$: freq of class x; N: total no. of data; $|X|$: No of class)

CSiME

Graphs

Definition: Undirected Graph vs Directed Graph

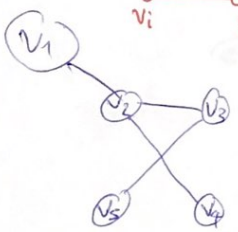
$G = (V, E)$ — nodes & edges

$D = (V, A)$ — nodes & arcs

$V = \{v_i\}$

$E = \{e_i\} = \{\{v_i, v_j\}\}$

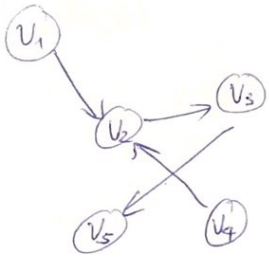
Adjacency Matrix



$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

symmetric

$$\begin{array}{c} \\ \text{from} \end{array} \begin{array}{c} \text{to} \\ \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \end{array}$$



A^k: $(i, j) \Rightarrow$ how many ways from node $i \to j$ with path length k

Incidence Matrix

$$\begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} \begin{array}{c} \begin{array}{cccc} e_1 & e_2 & e_3 & e_4 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

$$\begin{array}{c} \\ v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{array} \begin{array}{c} \begin{array}{cccc} a_1 & a_2 & a_3 & a_4 \end{array} \\ \begin{bmatrix} 1 & 0 & 0 & 0 \\ -1 & 1 & 0 & -1 \\ 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -1 & 0 \end{bmatrix} \end{array}$$

1: out (go from)
-1: in (go to)

Trees & Forest

+) Tree: [undirected graph: connected & acyclic (not forming loop)
directed graph: exists a unique walk from root to all leaf

+) Forest: trees

Search Algorithms

Properties: completeness, optimality, time complexity, space complexity

- completeness: guaranteed solution will be found if exists
- optimality: opt solution is always found
- time complexity: (running time)
- space complexity: (memory required)

Uninformed search

Depth-first search

go down until you can't

Breadth-first search

1 layer at a time

Minimum Spanning Tree (MST)

+) Subgraph of an undirected graph G
+) If edges have weights ⇒ MST can be found
+) Weight of tree := Σ weight of edges
+) MST: tree with minimal weight

MST heuristic: create MST for an undirected graph

Prim's Algorithm

+) Init at random node
+) In all available edges choose the minimum — expand from previous node
+) ↻ till all vertices are in the tree

Kruskal's Algorithm

+) Add the smallest edge, as long as no circle is created, till all vertices ∈ tree

Dijkstra's Algorithm

⊛ Only positive weights

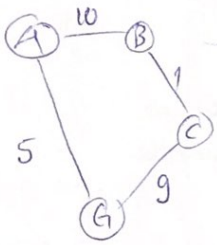
+) Init arbitrary

+) Follow expand at 1 vertex with closest distance

Till converge

Example:

Predecesser	Active	A	B	C	G
		0	∞	∞	∞
	A	0	10	0	5
A	G	0	10	14	5
A, G	B	0	10	11	5



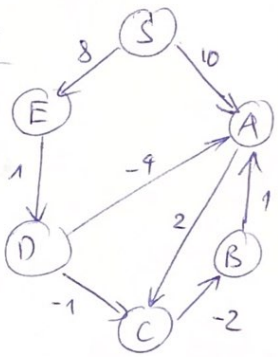
Bellman & Ford's Algorithm

Have table

+) Each step

+) At all nodes, but in order (left ⇒ right)

Init	S	A	B	C	D	E
	0	∞	∞	∞	∞	∞
Loop 1:	0	10	10	12	9	8
Loop 2:	0	5	10	8	9	8



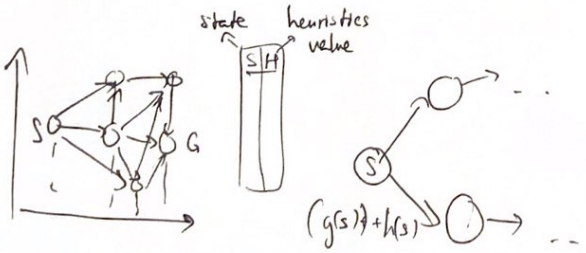
Use when we require to find the shortest path

⊛ A^*: $f(x) = g(x) + h(x)$

- $f(x)$: value of state x
- $g(x)$: cost to get to x
- $h(x)$: the heuristic from x to the final goal

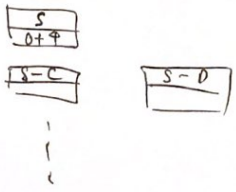
- Expand node with smallest $f(x)$, until every path is explored

A^* as search tree



Closed list = { S, ... }

A^* with priority queue



~~44~~

- Statically weighted A^*: $h_w(v) = h_{a(v)} \cdot \varepsilon \quad , \varepsilon > 1$
- Dynamically weighted A^*: $f(v) = g(v) + (1 + \varepsilon w(v)) h(v)$

$$w(v) = \begin{cases} 1 - \frac{d(v)}{N} & \text{if } d(v) \leq N \\ 0 & \text{else} \end{cases}$$

, N is the expected path steps needed

A^*: optimal solution
computational-expensive & high memory consumption

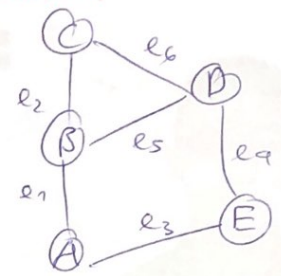
Statically weighted A^*: faster to a solution
loses solution quality

Dynamically weighted A^*: trade-off between speed & quality

CSiME

Ex 1: Search Algorithm

T1/ a) Adjacency matrix



$$A = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

b) Number of paths from E → C with length 3

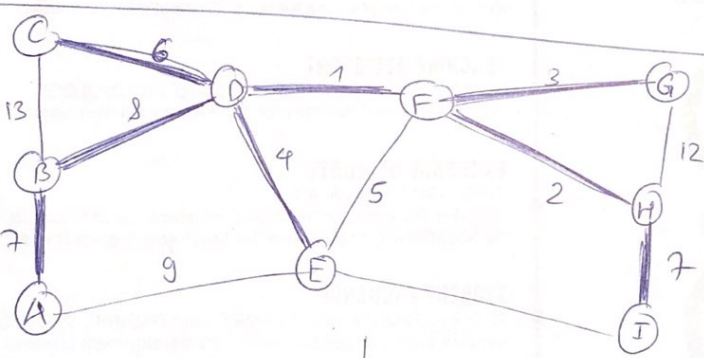
$$\Rightarrow A^3 = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \end{array} \begin{array}{c} \begin{array}{ccccc} A & B & C & D & E \end{array} \\ \begin{bmatrix} 0 & 5 & 2 & 1 & 4 \\ 5 & 2 & 4 & 6 & 1 \\ 2 & 4 & 2 & 4 & 2 \\ 1 & 6 & 4 & 2 & 5 \\ 4 & 1 & 2 & 5 & 0 \end{bmatrix} \end{array}$$

⇒ 2 possible path EABC, EDBC

c) $G = (V, E)$ where $V = \{A, B, C, D, E\}$

$E = \{e_1 \dots e_6\}$

T2/



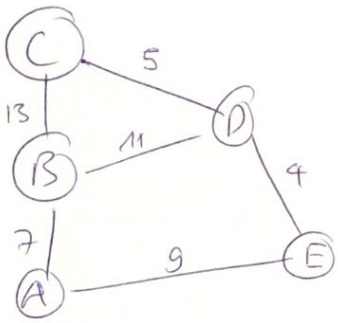
Prim (start node D)

Step	Nodes	Edges
1	D	—
2	D, F	e_{DF}
3	D, F, H	e_{DF}, e_{FH}
4	D, F, H, G	e_{DF}, e_{FH}, e_{FG}
5	D F H G E	e_{DF} e_{FH} e_{FG} e_{DE}
6	D F H G E C	e_{DF} e_{FH} e_{FG} e_{DE} e_{DC}
7	D F H G E C I	e_{DF} e_{FH} e_{FG} e_{DE} e_{DC} e_{HI}
8	D F H G E C I B	- - - e_{DB}
9	- - - A - - -	e_{BA}

Kruskal

Step	Nodes	Edges
1	D F	e_{DF}
2	D F H	e_{DF} e_{FH}
3	D F H G	e_{DF} e_{FH} e_{FG}
4	D F H G E	e_{DF} e_{FH} e_{FG} e_{DE}
5	D F H G E C	e_{DF} e_{FH} e_{FG} e_{DE} e_{DC}
6	D F H G E C I A B	e_{DF} e_{FH} e_{FG} e_{DE} e_{DC} e_{AB} e_{HI}
7	——— - - -	e_{BD}

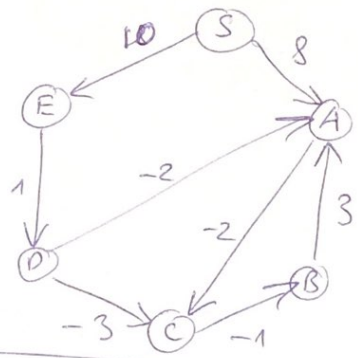
T3, Dijkstra from A



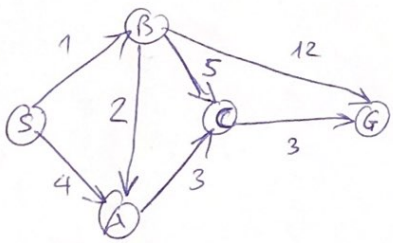
Predecessor	Active	A	B	C	D	E
		0	∞	∞	∞	∞
	A	0	7	∞	∞	9
A	B	0	7	20	18	9
AB	E	0	7	20	13	9
ABE	D	0	7	18	13	9
ABED	C	0	7	18	13	9

T4, Bellman and Ford algorithm

	S	A	B	C	D	E
Init	0	∞	∞	∞	∞	∞
Loop	0	8	5	~~6~~	11	10
	0	8	5	6	11	10

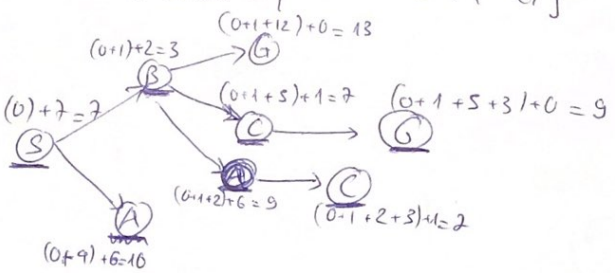


T5, A* algorithm with closed list

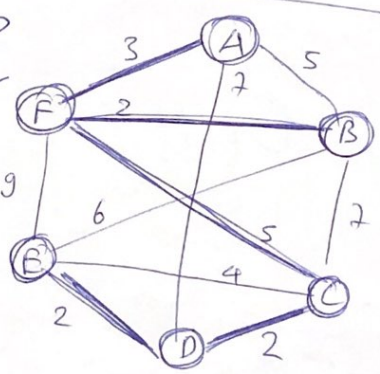


State	H
S	2
A	6
B	2
C	1
G	0

Closed list = { S B C A G }



T6 $W_{MST} \leq 2 W_{opt}$, $W_{MST} = 14$?



Euler Circle: FA F B F C D E D C F

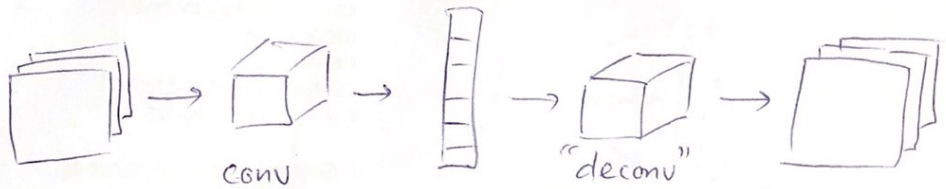
⇒ FA B C D EF = 27

⇒ Weigh of the ST = $\frac{27}{2} \approx 14$

CSME

Variational Auto encoders

- Normal: Auto encoder



tranposed convolutions
or fractionally strided convolutions

latent vector

Normal Auto Encoder	Variational Auto Encoder
- Learns hidden representation of input (⇒ learn latent vector)	- Learns to generate new data
- Can not generate new data	
what is optimized?	
- Reconstruction loss	Reconstruction loss + latent loss

Images are more blurry than GAN
(Generative Adversarial iNets)

The more dimensions in the latent vector → sharper, clearer outputs
→ needs more data

Application:

+) Denoising Autoencoder: add noise to inputting, try loss with no noise

+) Neural impainting: crop out a part, remove watermarks...

7

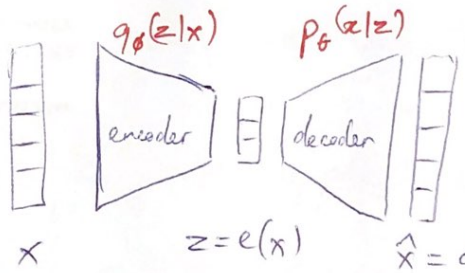
Normal AE

Input ⇒ fixed vector

VAE

Input ⇒ distribution < mean vector, standard deviation

- VAE loss:



Autoencoder loss $= \| x - \hat{x} \|^2 = \| x - d(e(x)) \|^2$

VAE:

$x \Rightarrow$ latent distribution $p(z|x)$ $N(\mu_x, \sigma_x)$ $\Longrightarrow$ sampling $z \sim p(z|x)$ $\Rightarrow$ decoding $d(z)$

loss $= \| x - d(z) \|^2 + \mathbb{KL}\left[N(\mu_x, \sigma_x), N(0,1) \right]$

$\mathcal{L}(\theta, \phi; x, z) = \mathbb{E}_{q_\phi(z|x)} \log p_\theta(x|z) - D_{KL}(q_\phi(z|x) \| p(z))$

(explaination behind, last page)

+) Reparametrized trick: $z = \mu + \sigma \cdot \varepsilon$ $(\varepsilon \sim \text{Normal}(0,1))$

⇒ to ensure gradients flow

+) Disentangled VAE: add β to D_{KL} loss

β too small ⇒ overfitting

β too big ⇒ loose high definition details ... hurt performance

Kullback-Leibler Divergence

Kind of a distance metric between 2 distributions

Entropy: $H = -\sum_{i=1}^{N} p(x_i) \log p(x_i)$

$$D_{KL}(p\|q) = \sum_{i=1}^{N} p(x_i)\left(\log p(x_i) - \log q(x_i)\right) = \sum_{i=1}^{N} p(x_i) \log \frac{p(x_i)}{q(x_i)}$$

$$= \mathbb{E}\left[\log p(x) - \log q(x)\right]$$

⇒ How many bits of info we expect to lose

⇒ A function that we can optimized

⇒ $H(p, q) = H(p) + D_{KL}(p\|q)$

cross entropy = entropy + KL Divergence

	State	Action model
Classical planning	Observable	Deterministic, exact
MDPs	Observable	Stochastic (Non-deterministic)
POMDPs	Partially observable	Stochastic

Markov Decision Processes

Partially observable Markov Decision Processes

KL Divergence between 2 normal distributions: $\mathcal{N}(\mu_1, \sigma_1)$ & $\mathcal{N}(\mu_2, \sigma_2)$

$$KL(p, q) = \log \frac{\sigma_2}{\sigma_1} + \frac{\sigma_1^2 + (\mu_1 - \mu_2)^2}{2\sigma_2^2} - \frac{1}{2}$$

$\mu_2 = 0$
$\sigma_2 = 1$

CODE:

```
μ, σ = encoder(x̂)
z    = μ + σ * random_normal(0, 1)
y    = decoder(z)
recon_loss = x · log(y) + (1-x) log(1-y)
KL_loss = 1/2 [squ(μ) + squ(σ) − log(squ(σ) + 1e^-8) − 1]
```



```
ELBO = recon_loss − KL_loss
loss = −ELBO
```

The math behind VAE

VAE: learn model parameters to maximize likelihood of training data:

$$p_\theta(x) = \int p_\theta(z)\, p_\theta(x|z)\, dz$$

$p_\theta(z)$: are chosen (simple Gaussian)

$p_\theta(x|z)$: Decoder neural network

$$\log p_\theta(x^{(i)}) = \mathbb{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right]$$

$$= \mathbb{E}_z\left[\log \frac{p_\theta(x^{(i)}|z)\,p_\theta(z)}{p_\theta(z|x^{(i)})} \cdot \frac{q_\phi(z|x^{(i)})}{q_\phi(z|x^{(i)})}\right]$$

$$= \mathbb{E}_z\left[\log p_\theta(x^{(i)}|z)\right] - \mathbb{E}_z\left[\log \frac{q_\phi(z|x^{(i)})}{p_\theta(z)}\right] + \mathbb{E}_z\left[\log \frac{q_\phi(z|x^{(i)})}{p_\theta(z|x^{(i)})}\right]$$

$$= \mathbb{E}_z\left[\log p_\theta(x^{(i)}|z)\right] - D_{KL}\left(q_\phi(z|x^{(i)}) \| p_\theta(z)\right) + D_{KL}\left(q_\phi(z|x^{(i)}) \| p_\theta(z|x^{(i)})\right)$$

Tractable lower bound ⇒ can take gradient and optimized

intractable, can't compute

But, we know it ≥ 0

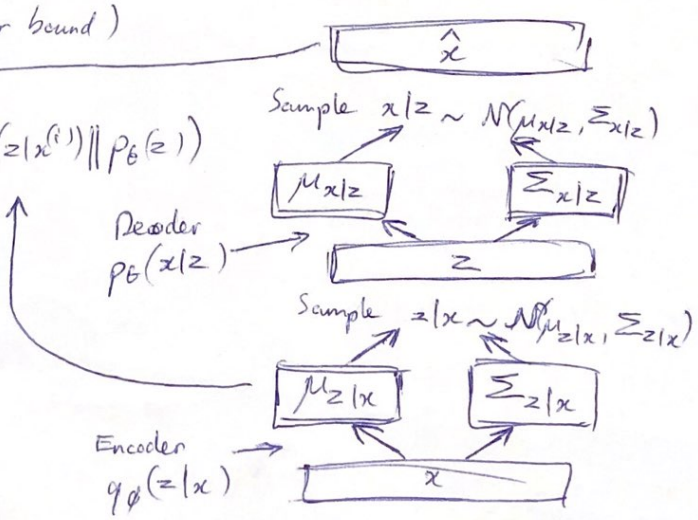
Intuition:

- Reconstruction: the higher the probability, the closer the output to input (similar to maximum likelihood)
- Make approximation posterior distribution close to prior

$$\log p_\theta(x^{(i)}) \geq \mathcal{L}(x^{(i)}, \theta, \phi) \quad \Rightarrow \quad \theta^*, \phi^* = \arg\max_{\theta,\phi} \sum_{i=1}^{N} \mathcal{L}(x^{(i)}, \theta, \phi)$$

Variational lower bound "ELBO" (evidence lower bound)

$$\mathbb{E}_z\left[\log p_\theta(x^{(i)}|z)\right] - D_{KL}\left(q_\phi(z|x^{(i)}) \| p_\theta(z)\right) = \mathcal{L}(x^{(i)}, \theta, \phi)$$



CS ME

MDP

MDP state Projects an expectimax-like search tree

(s, a, s') transition

$T(s, a, s') = P(s' | s, a)$

$R(s, a, s')$

Finite vs infinite Horizon

★ To solve infinite utilities: finite Horizon
discounting
absorbing state (like the freehold over heating)

MDP: [Policy: choice of action (at each state)
Utility: sum of (discounted) rewards

$V^*(s)$: value of a state: expected utility starting in s and acting optimally

$Q^*(s, a)$: expected utility starting in s and having taken a, then act optimally

$\pi^*(s)$: optimal action / policy from state s

$$V^*(s) = \max_a Q^*(s, a)$$

$$Q^*(s, a) = \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V^*(s') \right]$$

$$\Rightarrow V^*(s) = \max_a \sum_{s'} T(s, a, s') \left[R(s, a, s') + \gamma V^*(s') \right]$$

Bellman equations

$$V(s) = \mathbb{E}\left[G_t \mid S_t = s \right] = \mathbb{E}\left[R_{t+1} + \gamma V(S_{t+1}) \mid S_t = s \right] = R_s + \gamma \sum_{s'} P_{ss'} V(s')$$

Lecture 8

- Markov process: random process without memories

is a tuple $\langle S, P \rangle$

S: state

P: transition matrix

$$P = \begin{bmatrix} & \\ & P_{ss'} \\ & \end{bmatrix} \quad s \to s'$$

$$P_{ss'} = \mathbb{P}\left[S_{t+1} = s' \mid S_t = s\right]$$ probabilities from s to s'

- Markov reward process: tuple $\langle S, R, R, \gamma \rangle$

state, transition matrix, reward func, discount factor $\gamma \in [0, 1]$

$$R_s = \mathbb{E}\left[R_{t+1} \mid S_t = s\right]$$

- **Utility** of state sequences G_t

$$G_t = U([S_t, S_{t+1}, \dots, S_\infty]) = \sum_{k=0}^{\infty} \gamma^k R(S_{t+k}) \leq \frac{R_{max}}{1-\gamma} \quad (\gamma < 1)$$

(U: utility)

$\gamma \to 0 \Rightarrow$ short-sighted evaluation

$\gamma \to 1 \Rightarrow$ far-sighted

$\gamma = 1 \Rightarrow$ additive utility

The Stupid 5 axioms of rational decisions

order $(A \succ B) \vee (B \succ A) \vee (A \sim B)$

transitivity $(A \succ B) \wedge (B \succ C) \Rightarrow (A \succ C)$

continuity $A \succ B \succ C \Rightarrow \exists p\, [p, A;\ 1-p, C] \sim B$

substitution: $(A \sim B) \Rightarrow [p, A;\ 1-p, C] \sim [p, B;\ 1-p, C]$

monotony: $A \succ B \Rightarrow \left(p \geq q \Leftrightarrow [p, A;\ 1-p, B] \succ [q, A;\ 1-q, B]\right)$

- Value iteration: $V_{k+1}(s) \leftarrow \max_a \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma V_k(s')\right]$

Repeat until convergence: unique (fixed) optimal value

But, policy may converge long before values do

- When the policy is fixed, Bellman equation is a linear equation:

$$v = R + \gamma P v$$

$$\begin{bmatrix} v(1) \\ v(2) \\ \vdots \\ v(n) \end{bmatrix} = \begin{bmatrix} R_1 \\ \vdots \\ R_n \end{bmatrix} + \gamma \begin{bmatrix} P_{11} & \cdots & P_{1n} \\ \vdots & \ddots & \vdots \\ P_{n1} & \cdots & P_{nn} \end{bmatrix} \begin{bmatrix} V_1 \\ \vdots \\ V_n \end{bmatrix}$$

$\Rightarrow v = (I - \gamma P)^{-1} R$ (convergence)

- Dynamic Programming | Value iteration

Synchronous

for all $s \in S$

$v_{new}(s) \leftarrow \max_a \left(R_s^a + \gamma \sum_{s'} P_{ss'}^a v_{old}(s')\right)$

$v_{old} \leftarrow v_{new}$

calculate all expected value $v^*(s)$

Update all at once

Asynchronous

for all $s \in S$:

$v(s) \leftarrow \max_a \left(R_s^a + \gamma \sum_{s'} P_{ss'}^a v(s')\right)$

calculate each $v^*(s)$

then update immediately

Value iteration

1, Init: all expected utility = 0, except terminal state

2, Start at random state

3, Each state, try all actions

4, ...

⇒ 5, Draw out π^* for ~~that~~ the next iteration

6, Continue until convergence

POMDP

Partially observable MDP

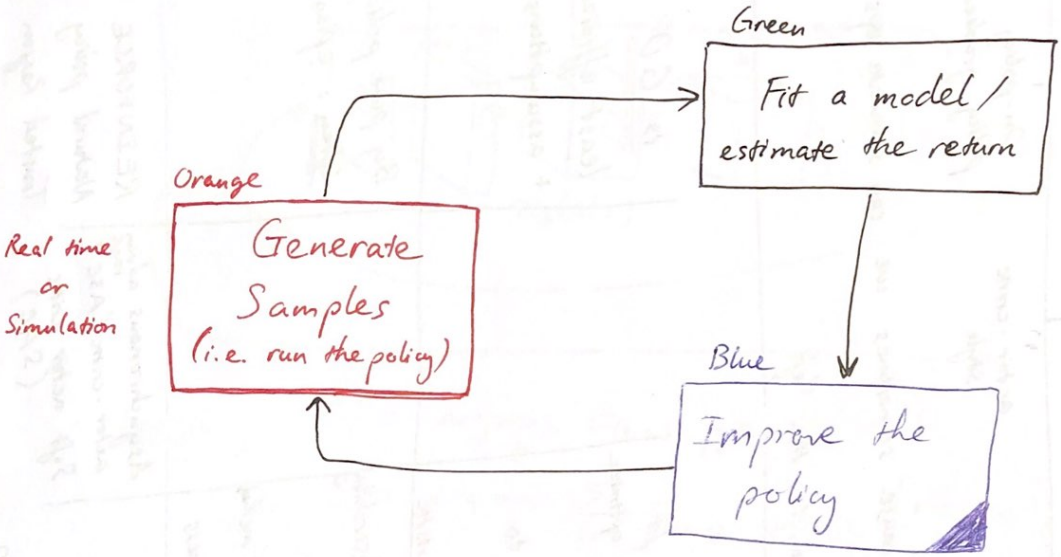
tuple $\langle S, A, O, P, R, Z, \gamma \rangle$

state, action, observation, transition matrix, reward, observation function, discount

$$Z^{a}_{s'o} = P\left[O_{t+1} = o \mid S_{t+1} = s', A_t = a\right]$$

$$\Rightarrow \begin{cases} v^*(b) \leftarrow \max_a \left(R^a_b + \gamma \sum_{s'} P^a_{bb'} v(b') \right) \\ \pi^*(b) = \arg\max_a \left(R^a_b + \gamma \sum_{s'} P^a_{bb'} v(b') \right) \end{cases}$$

Reinforcement - Learning



Structure of RL algorithms

Some definition:

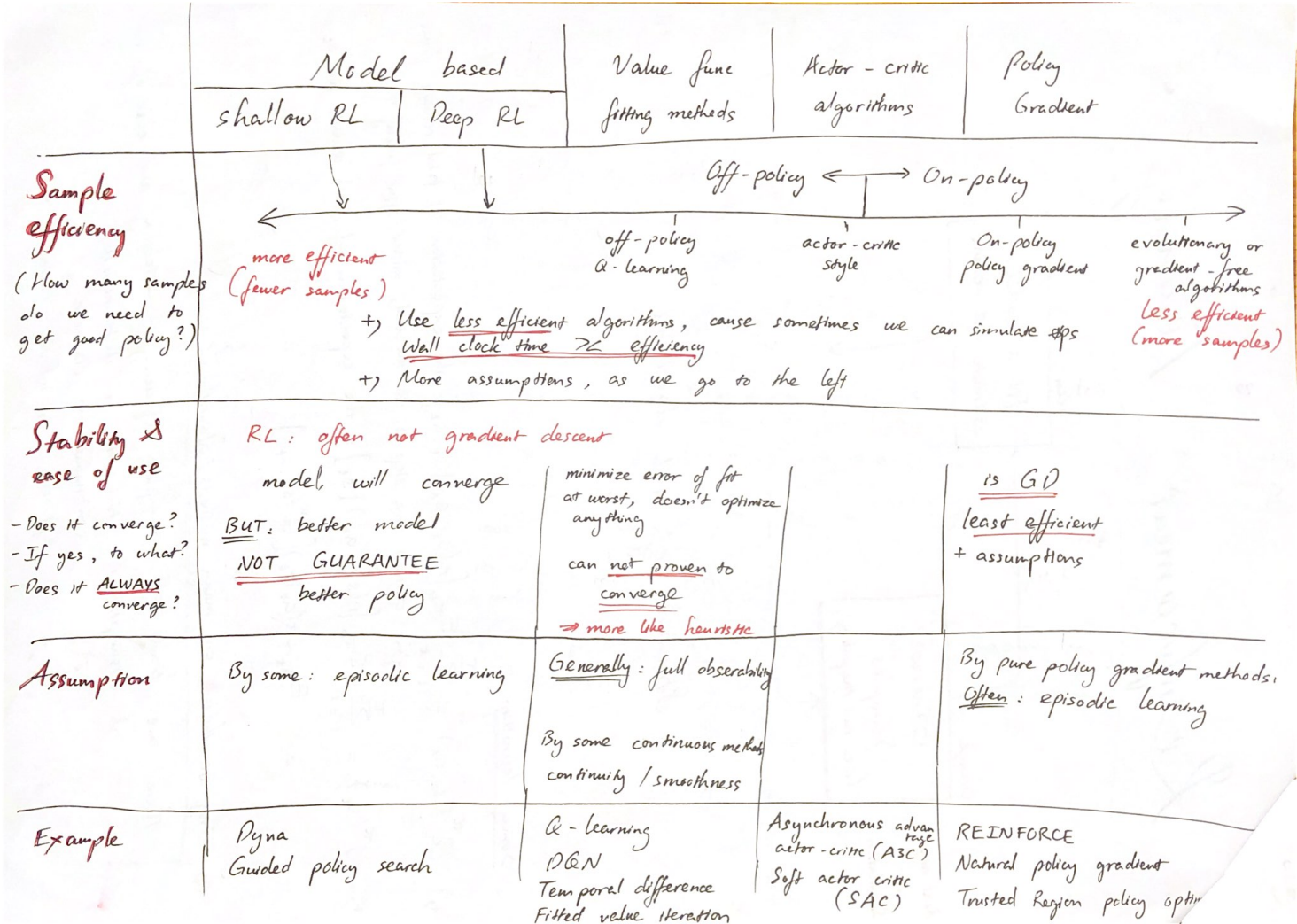
+, $Q^{\pi}(s_t, a_t) = \sum_{t'=t}^{T} E_{\pi_\theta}\left[r(s_{t'}, a_{t'}) \mid s_t, a_t\right]$: the expectation of total reward from your current time step t till the end, under your policy π_θ

+, $V^{\pi}(s_t) = \sum_{t'=t}^{T} E_{\pi_\theta}\left[r(s_{t'}, a_{t'}) \mid s_t\right]$: the expectation of total reward from s_t

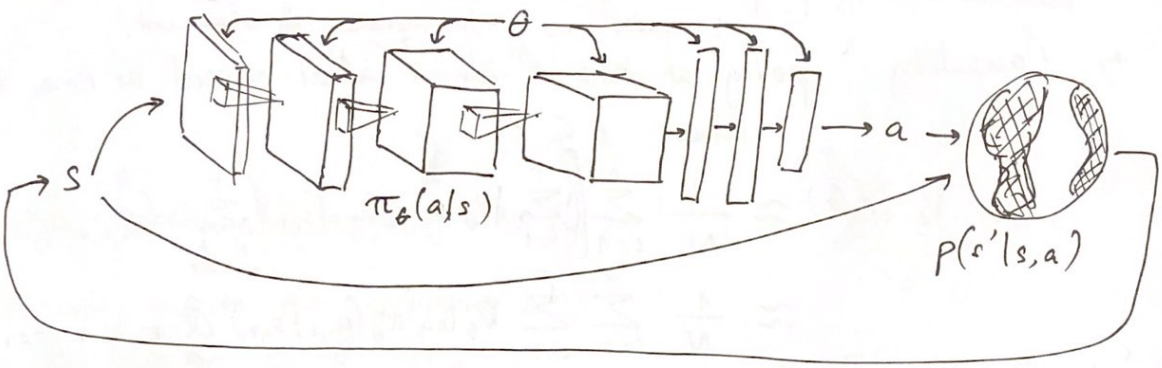
$= E_{a_t \sim \pi(a_t \mid s_t)}\left[Q^{\pi}(s_t, a_t)\right]$

⊗ Why are there so many methods / algorithms?

There are trade-off: sampling efficiency & stability and ease of use
& assumptions: stochastic or deterministic
continuous or discrete
episodic or infinite horizon


Model based
shallow RL
Deep RL
Value func fitting methods
Actor - critic algorithms
Policy Gradient
Sample efficiency
(How many samples do we need to get good policy?)
Off-policy ⟷ On-policy
more efficient (fewer samples)
off-policy Q-learning
actor-critic style
On-policy policy gradient
evolutionary or gradient-free algorithms
less efficient (more samples)
+) Use less efficient algorithms, cause sometimes we can simulate steps
Wall clock time ≠ efficiency
+) More assumptions, as we go to the left
Stability & ease of use
- Does it converge?
- If yes, to what?
- Does it ALWAYS converge?
RL: often not gradient descent
model will converge
BUT: better model
NOT GUARANTEE better policy
minimize error of fit
at worst, doesn't optimize anything
can not proven to converge
⇒ more like heuristic
is GD
least efficient
+ assumptions
Assumption
By some: episodic learning
Generally: full observability
By some continuous methods: continuity / smoothness
By pure policy gradient methods:
Often: episodic learning
Example
Dyna
Guided policy search
Q-learning
DQN
Temporal difference
Fitted value iteration
Asynchronous advantage actor-critic (A3C)
Soft actor critic (SAC)
REINFORCE
Natural policy gradient
Trusted Region policy optim

Policy Gradient



Goal: $\theta^* = \arg\max_\theta E_{(s,a)\sim p_\theta(s,a)}[r(s,a)]$

$= \arg\max_\theta \sum_{t=1}^{T} E_{(s_t,a_t)\sim p_\theta(s_t,a_t)}[r(s_t,a_t)]$ — T: infinite / finite] horizen case

⇒ maximize: $J(\theta) = E_{\tau\sim p_\theta(\tau)}[r(\tau)] = \int \pi_\theta(\tau) r(\tau)\, d\tau$

$\approx \frac{1}{N} \sum_i \sum_t r(s_{i,t}, a_{i,t})$

sum over samples ↗ ↖ sum over time steps

The Gradient: $\nabla_\theta J(\theta) = \int \nabla_\theta \pi_\theta(\tau) r(\tau) d\tau = \int \pi_\theta(\tau) \nabla_\theta \log \pi_\theta(\tau) r(\tau) d\tau$

$= E_{\tau\sim\pi_\theta(\tau)}[\nabla_\theta \log \pi_\theta(\tau) r(\tau)]$

$= E_{\tau\sim\pi_\theta(\tau)}\left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t)\right)\left(\sum_{t=1}^{T} r(s_t,a_t)\right)\right]$

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t})\right)\left(\sum_{t=1}^{T} r(s_{i,t},a_{i,t})\right)\right]$$

then $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

⊖ REMARKS: some what like MLE, makes good stuff happen more, bad stuff less

⊕ Partial observability:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left[\left(\sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|o_{i,t})\right)\left(\sum_{t=1}^{T} r(s_{i,t},a_{i,t})\right)\right]$$

$o_{i,t} \rightarrow$ network $\rightarrow a_{i,t}$

$\pi_\theta(a_{i,t}|o_{i,t})$

⊗ Problem of Policy Gradient: HIGH VARIANCE with $r(\tau)$

Different samples ⇒ different gradient estimate

For a small finite No. Samples ⇒ gradient noisy

(At the beginning, policy θ is not good ⇒ random action ⇒ the not-so-good action-result accumulate ⇒ high variance in the end)

+) Causality: policy at time t' cannot affect reward at time t when $t < t'$

$$\Rightarrow \nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\left(\left(\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t})\right)\left(\sum_{t'=t}^{T} r(s_{i,t'}, a_{i,t'})\right)\right)$$

$$\approx \frac{1}{N}\sum_{i=1}^{N}\sum_{t=1}^{T}\nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t})\,\hat{Q}_{i,t}$$

$\hat{Q}_{i,t}$: reward to-go ⊗

+) Baselines: $b = \frac{1}{N}\sum_{i=1}^{N} r(\tau)$ ⊗

$$\Rightarrow \nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\nabla_\theta \log \pi_\theta(\tau)\,[r(\tau) - b] \quad = \quad Q^{\pi}(s_t, a_t) - V^{\pi}(s_t)$$

REINFORCE algorithm:

1) Sample $\{\tau^i\}$ from $\pi_\theta(a_t|s_t)$

2) $\nabla_\theta J(\theta) \approx \frac{1}{N}\sum_{i=1}^{N}\nabla_\theta \log \pi_\theta(\tau)\left(\sum_{t'=t}^{T} r(s_{i,t'}, a_{i,t'}) - b\right)$ $\quad b = \frac{1}{N}\sum_{i=1}^{N}\sum_{t'=t}^{T} r(s_{t'}, a_{t'})$

3) $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Pseudo code: (with causality & baselines)

```
logits = policy.predictions(states)
negative_likelihoods = tf.nn.softmax_cross_entropy(labels=actions, logits)
weighted_negative_likelihoods = tf.multiply(negative_likelihoods, q_values)
loss = tf.reduce_mean(weighted_negative_likelihoods)
gradients = loss.gradients(loss, variables)
```

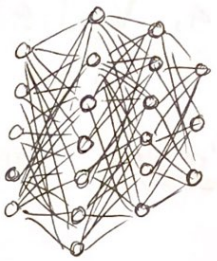
+) Can derive off-policy policy gradient with IMPORTANT SAMPLING

$$\because E_{x\sim p(x)}[f(x)] = E_{x\sim q(x)}\left[\frac{p(x)}{q(x)} f(x)\right] \Rightarrow \nabla_\theta J(\theta) = E_{\tau\sim \pi_{\theta_{old}}(\tau)}\left[\frac{\pi_\theta(\tau)}{\pi_{\theta_{old}}(\tau)}\nabla_\theta \ldots\right]$$

Actor - Critic

the policy - value function (a.k.a. policy evaluation)

An extra network:



$$Q^\pi(s_t, a_t) = \sum_{t'=t}^{T} E_{\pi_\theta}[r(s_{t'}, a_{t'}) \mid s_t, a_t]$$

$$V^\pi(s_t) = E_{a_t \sim \pi_\theta(a_t|s_t)}[Q^\pi(s_t, a_t)]$$

$$\Rightarrow A^\pi(s_t, a_t) = Q^\pi(s_t, a_t) - V^\pi(s_t)$$

- We can rewrite the gradient as:

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t} \mid s_{i,t}) A^\pi(s_{i,t} \mid a_{i,t})$$

As: $Q^\pi(s_t, a_t) = r(s_t, a_t) + E_{s_{t+1} \sim p(s_{t+1}|s_t, a_t)}[V^\pi(s_{t+1})]$

estimate as $\approx r(s_t, a_t) + V^\pi(s_{t+1})$ (instead of the average, take value of 1 sample)

- Then we have: $A^\pi(s_t, a_t) \approx r(s_t, a_t) + V^\pi(s_{t+1}) - V^\pi(s_t)$

- What to fit to:

$$y_{i,t} = \sum_{t'=t}^{T} E_{\pi_\theta}[r(s_{t'}, a_{t'} \mid s_{i,t})] \approx r(s_{i,t}, a_{i,t}) + V^\pi(s_{i,t+1}) \approx r(s_{i,t}, a_{i,t}) + \hat{V}^\pi_\phi(s_{i,t+1})$$

Bootstrap estimate

$$\mathcal{L}(\phi) = \frac{1}{2} \sum_i \| \hat{V}^\pi_\phi(s_i) - y_i \|^2$$

$$y_{i,t} = \sum_{t'=t}^{T} r(s_{i,t'}, a_{i,t'})$$ Monte-Carlo (do as before, estimate by running it in a real world $\Rightarrow$ a single sample estimate)

Batch Actor-Critic algorithm:

1) Sample $\{s_i, a_i\}$ from $\pi_\theta(a|s)$
2) Fit $V^\pi_\phi(s)$ to sampled reward sum (Monte Carlo / Bootstrap estimate) ... $+ \mathcal{L}(\phi) = \frac{1}{2} \dots$
3) Evaluate $\hat{A}^\pi(s_i, a_i) = r(s_i, a_i) + \hat{V}^\pi_\phi(s_i') - \hat{V}^\pi_\phi(s_i)$
4) $\nabla_\theta J(\theta) \approx \sum \nabla_\theta \log \pi_\theta(a_i|s_i) \hat{A}^\pi(s_i, a_i)$
5) $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

With discount factor:

3) $\hat{A}^\pi(s_i, a_i) = r(s_i, a_i) + \gamma \hat{V}^\pi_\phi(s_i') - \hat{V}^\pi_\phi(s_i)$

On line Actor-Critic Algorithm:

1, Take action $a \sim \pi_\theta(a|s)$ get (s, a, s', r)

2, Update $\hat{V}_\phi^\pi$ using target value $r + \gamma \hat{V}_\phi^\pi(s')$

3, Evaluate $\hat{A}^\pi(s,a) = r(s,a) + \gamma \hat{V}_\phi^\pi(s') - \hat{V}_\phi^\pi(s)$

4, $\nabla_\theta J(\theta) \approx \nabla_\theta \log \pi_\theta(a|s)\, \hat{A}^\pi(s,a)$

5, $\theta \leftarrow \theta + \alpha \nabla_\theta J(\theta)$

Problem: single batch $\Rightarrow$ need parallel actor-critic (synchronous / asynchronous)

+, Architecture design: 2 networks (simple & stable, no shared features) vs shared network design (two different gradients which need to tuned, balanced)

+, Critic as state-dependent baselines

$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_{i,t}|s_{i,t}) \left(\left(\sum_{t'=t}^{T} \gamma^{t'-t} r(s_{i,t'}, a_{i,t'}) \right) - \hat{V}_\phi^\pi(s_{i,t}) \right)$$

+, Control variates: action-dependent baselines

~~Set $b = \hat{V}_\phi^\pi(s_{i,t})$~~ $\Rightarrow$ { no bias, lower variance

+, Eligibility traces & n-step returns:

Actor-critic: $\hat{A}_C^\pi(s_t, a_t) = r(s_t, a_t) + \gamma \hat{V}_\phi^\pi(s_{t+1}) - \hat{V}_\phi^\pi(s_t)$ low variance but biased

Monte Carlo: $\hat{A}_{MC}^\pi(s_t, a_t) = \sum_{t'=t}^{\infty} \gamma^{t'-t} r(s_{t'}, a_{t'}) - \hat{V}_\phi^\pi(s_t)$ no bias but higher variance

$$\Rightarrow \hat{A}_n^\pi(s_t, a_t) = \sum_{t'=t}^{t+n} \gamma^{t'-t} r(s_{t'}, a_{t'}) - \hat{V}_\phi^\pi(s_t) + \gamma^n \hat{V}_\phi^\pi(s_{t+n})$$

+, Generalized advantage estimation

$$\hat{A}_{GAE}^\pi(s_t, a_t) = \sum_{n=1}^{\infty} \omega_n \hat{A}_n^\pi(s_t, a_t) \qquad \omega_n \propto \lambda^{n-1}$$

Policy iteration

Can we omit policy gradient completely?

High level idea - Algorithm:

1, Evaluate $A^{\pi}(s,a)$

2, Set $\pi \leftarrow \pi'$

Dynamic Programming

+, Assume: we know: $p(s'|s,a)$ — $V^{\pi}(s)$

s & a are both discrete & small $\Rightarrow$ can store V in table

$$\pi'(a_t|s_t) = \begin{cases} 1 & \text{if } a_t = \arg\max_{a_t} A^{\pi}(s_t,a_t) \\ 0 & \text{otherwise} \end{cases}$$

π': simply choose the current best action

Policy iteration with Dynamic Programming

1, $V^{\pi}(s) \leftarrow r(s,\pi(s)) + \gamma \mathbb{E}_{s' \sim p(s'|s,\pi(s))}[V^{\pi}(s')]$

2, Set $\pi \leftarrow \pi'$

Simpler ⇒

Value iteration Algorithm

Since: $\arg\max_{a_t'} A^{\pi}(s_t,a_t) = \arg\max_{a_t} Q^{\pi}(s_t,a_t)$

1, Set $Q^{\pi}(s,a) \leftarrow r(s,a) + \gamma \mathbb{E}[V^{\pi}(s')]$

2, Set $V^{\pi}(s) \leftarrow \max_a Q^{\pi}(s,a)$

REMARKS, Still have a big table

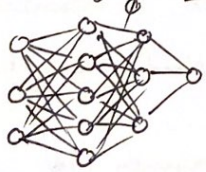
When facing the Curse of Dimension (of s)

⇒ Use neural network to evaluate $V(s)$, $Q(s,a)$

Fitted Value Iteration

1, $y_i \leftarrow \max_{a_i} r(s_i,a_i) + \gamma \mathbb{E}[V_{\phi}(s_i')]$

2, $\phi \leftarrow \arg\min_{\phi} \frac{1}{2} \sum_i \|V_{\phi}(s_i) - y_i\|^2$



BUT: Still need to know transition dynamic

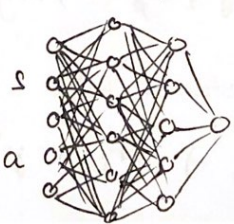
Fully Fitted Q Iteration

$\mathbb{E}[V(s_i')] \approx \max_{a'} Q_{\phi}(s_i', a_i')$

1, Collect: $\{(s_i, a_i, s_i', r_i)\}$ using some policy

[Kx 2, Set $y_i \leftarrow r(s_i,a_i) + \gamma \max_{a_i'} Q_{\phi}(s_i', a_i')$

3, $\phi \leftarrow \arg\min_{\phi} \frac{1}{2} \sum_i \|Q_{\phi}(s_i,a_i) - y_i\|^2$



- off-policy

- not really converge

Online Q iteration algorithm:

1, Take some action a_i & get (s_i, a_i, s_i', r_i) — 1 sample off policy

2, $y_i = r(s_i, a_i) + \gamma \max_{a'} Q_\phi(s_i', a_i')$

3, $\phi \leftarrow \phi - \alpha \frac{dQ_\phi}{d\phi}(s_i, a_i)\left(Q_\phi(s_i, a_i) - y_i\right)$ — 1 gradient step

+, How to choose action in step 1: Exploration vs Exploitation

⇒
- Epsilon greedy
- Boltzmann exploration (very large or continuous action-space)

Problems with Online Q-learning:

- Sequential states are strongly correlated ⇒ Replay buffer
- Target value is always changing ⇒ Target network

"Classic" Deep Q-learning (DQN) with replay buffer & target network

1, Save target network parameters $\phi' \leftarrow \phi$

N×

2, Collect dataset $\{(s_i, a_i, s_i', r_i)\}$ using some policy, add to buffer $\mathcal{B}$

K×

3, Sample a batch (s_i, a_i, s_i', r_i) from $\mathcal{B}$

4, $\phi \leftarrow \phi - \alpha \sum_i \frac{dQ_\phi}{d\phi}(s_i, a_i)\left(Q_\phi(s_i, a_i) - \left[r(s_i, a_i) + \gamma \max_{a'} Q_{\phi'}(s_i', a_i')\right]\right)$

+, Alternative: 5, Update $\phi' \leftarrow \tau\phi' + (1-\tau)\phi$ $\tau = 0,999$

+, Double Q-learning: helps a lot, solve over-estimate prob, no downside ⇒ should always use

$y = r + \gamma Q_{\phi'}(s', \arg\max_{a'} Q_\phi(s', a'))$ Instead of $\max_{a'} Q_{\phi'}(s_i', a_i')$

+, Mult-Step returns: helps a lot, have DOWNSIDE ⇒ frequently use

$y_{i,t} = \sum_{t'=t}^{t+N-1} \gamma^{t'-t} r_{i,t'} + \gamma^N \max_{a_{i,t+N}} Q_{\phi'}(s_{i,t+N}, a_{i,t+N})$ ONLY CORRECT when learning on-policy

Q learning with continuous action-space

Tips for Q-Learning:

- Large replay buffers help improve stability
- Start with high exploration ⇒ gradually reduce
- Bellman error gradients can be big ⇒ clip gradients / use Huber loss

Model-based RL

Optimal control

+) Deterministic case vs Stochastic

+) Open-loop case vs Closed-loop

Maximize objective through sequence of actions:

$$a_1, \dots, a_T = \underset{a_1 \dots a_T}{\operatorname{argmax}} J(a_1 \dots a_T) \qquad A = \underset{A}{\operatorname{argmax}} J(A)$$

Optimal control: discrete space

+) Guess & Check: Random Shooting Method

1) Pick $A_1 \dots A_N$ from some distribution (e.g. uniform)
2) Choose A_i based on $\operatorname{argmax}_i J(A_i)$

+) Cross-Entropy Method ~~CM~~ ~~CM~~ (CEM)

1) Sample $A_1, \dots, A_N$ from $P(A)$
2) Evaluate $J(A_1) \dots J(A_N)$
3) Pick elites $A_{i_1} \dots A_{i_M}$ with highest value $M < N$
4) Refit $P(A)$ to the elites $A_{i_1} \dots A_{i_M}$

+) Discrete case: Monte Carlo tree search (MCTS)

1) Find a leaf s_ℓ using Tree Policy(s_1)
2) Evaluate the leaf using Default Policy(s_ℓ)
3) Update all values in tree between s_1 and s_ℓ, take best action from s_1

UCT Tree Policy(s_t): $\text{Score}(s_t) = \dfrac{Q(s_t)}{N(s_t)} + 2C\sqrt{\dfrac{2\ln N(s_{t-1})}{N(s_t)}}$

All of these is what we do IF we already know the model

Derivatives are hard to come by, but SOMETIMES you CAN, through Physics equation

$$\min_{u_1,\dots,u_T} \sum_{t=1}^{T} c(x_t, u_t), \qquad x_t = f(x_{t-1}, u_{t-1})$$

$$= \min_{u_1,\dots,u_T} c(x_1, u_1) + c(f(x_1, u_1), u_2) + \dots + c(f(f(\dots)\dots), u_T)$$

Special case: LQR - Linear Quadratic Regulator

Assume: $f(.)$ has special structure:

$$f(x_t, u_t) = F_t \begin{bmatrix} x_t \\ u_t \end{bmatrix} + f_t \qquad \text{linear dynamics}$$

$$c(x_t, u_t) = \frac{1}{2} \begin{bmatrix} x_t \\ u_t \end{bmatrix}^T C_t \begin{bmatrix} x_t \\ u_t \end{bmatrix} + \begin{bmatrix} x_t \\ u_t \end{bmatrix}^T c_t \qquad \text{quadratic cost}$$

$$C_T = \begin{bmatrix} C_{x_T, x_T} & C_{x_T, u_T} \\ C_{u_T, x_T} & C_{u_T, u_T} \end{bmatrix}, \qquad c_T = \begin{bmatrix} c_{x_T} \\ c_{u_T} \end{bmatrix}$$

Solve for u_T only:

$$Q(x_T, u_T) = \text{const} + \frac{1}{2} \begin{bmatrix} x_T \\ u_T \end{bmatrix}^T C_T \begin{bmatrix} x_T \\ u_T \end{bmatrix} + \begin{bmatrix} x_T \\ u_T \end{bmatrix}^T c_T \qquad (2)$$

Set $\nabla_{u_T} Q(x_T, u_T) = 0 \Rightarrow \dots$

$$u_T = -C_{u_T, u_T}^{-1} (C_{u_T, x_T} \cdot x_T + c_{u_T})$$

$$= K_T x_T + k_T \qquad (1)$$

Replace (1) to (2) $\Rightarrow V(x_T)$

Minimize objective on last action u_T, based on x_T

u_{T-1} affects objective through linear dynamics func to x_T

Linear case: LQR (continue)

Backward recursion

for $t = T \to 1$:

$$Q_t = C_t + F_t^T V_{t+1} F_t$$

$$q_t = c_t + F_t^T V_{t+1} f_t + F_t^T v_{t+1}$$

$$Q(x_t, u_t) = \text{const} + \frac{1}{2}\begin{bmatrix} x_t \\ u_t \end{bmatrix}^T Q_t \begin{bmatrix} x_t \\ u_t \end{bmatrix} + \begin{bmatrix} x_t \\ u_t \end{bmatrix}^T q_t$$

$$u_t \leftarrow \arg\min_{u_t} Q(x_t, u_t) = K_t x_t + k_t$$

$$K_t = -Q_{u_t, u_t}^{-1} Q_{u_t, x_t}$$

$$k_t = -Q_{u_t, u_t}^{-1} q_{u_t}$$

$$V_t = Q_{x_t, x_t} + Q_{x_t, u_t} K_t + K_t^T Q_{u_t, x_t} + K_t^T Q_{u_t, u_t} K_t$$

$$v_t = q_{x_t} + Q_{x_t, u_t} k_t + K_t^T Q_{u_t} + K_t^T Q_{u_t, u_t} k_t$$

$$V(x_t) = \text{const} + \frac{1}{2} x_t^T V_t x_t + x_t^T v_t$$

Forward recursion:

for $t = 1 \to T$:

$$u_t = K_t x_t + k_t$$

$$x_{t+1} = f(x_t, u_t)$$

Non-linear case: DDP (Differential Dynamic Programming) or Iterative LQR

Every iteration: linearize local dynamic ~~at loca~~ with Taylor expansion

$$f(x_t, u_t) \approx f(\hat{x}_t, \hat{u}_t) + \nabla_{x_t, u_t} f(\hat{x}_t, \hat{u}_t) \begin{bmatrix} x_t - \hat{x}_t \\ u_t - \hat{u}_t \end{bmatrix}$$

$$c(x_t, u_t) \approx c(\hat{x}_t, \hat{u}_t) + \nabla_{x_t, u_t} c(\hat{x}_t, \hat{u}_t) \begin{bmatrix} x_t - \hat{x}_t \\ u_t - \hat{u}_t \end{bmatrix} + \frac{1}{2}\begin{bmatrix} x_t - \hat{x}_t \\ u_t - \hat{u}_t \end{bmatrix}^T \nabla^2_{x_t, u_t} c(\hat{x}_t, \hat{u}_t) \begin{bmatrix} x_t - \hat{x}_t \\ u_t - \hat{u}_t \end{bmatrix}$$

$$F_t = \nabla_{x_t, u_t} f(\hat{x}_t, \hat{u}_t)$$

$$c_t = \nabla_{x_t, u_t} c(\hat{x}_t, \hat{u}_t)$$

$$C_t = \nabla^2_{x_t, u_t} c(\hat{x}_t, \hat{u}_t)$$

Run LQR backward pass on state $\delta x_t = x_t - \hat{x}_t$, $\delta u_t = u_t - \hat{u}_t$

forward pass with real nonlinear dynamics and $u_t = K_t(x_t - \hat{x}_t) + k_t + \hat{u}_t$

Update $\hat{x}_t$ and $\hat{u}_t$

How to learn model

Model-based RL v.0.5:

1, Run based policy $\pi_0(a_t|s_t) \Rightarrow$ collect $\mathcal{D} = \{(s, a, s')_i\}$
2, Learn dynamic model $f(s,a)$ to minimize $\sum_i \| f(s_i, a_i) - s_i' \|^2$
3, Plan through $f(s,a)$ to choose actions

- Good based policy should be taken good care
- Particularly effective if can hand-engineer a dynamics representation with our knowledge of physics ... $\Rightarrow$ need to fit only a few parameters

+) Problems: might go beyond to new state distribution different to the one that colleted the data

Model-based RL v.1.0: (start to work with neural network)

1, Run base policy $\pi_0(a_t|s_t) \Rightarrow$ collect $\mathcal{D} = \{(s, a, s')_i\}$
2, Learn model $f(s,a)$ to minimice $\sum_i \| f(s_i, a_i) - s_i' \|^2$
3, Plan through $f(s,a) \Rightarrow$ choose actions
4, Execute these actions & add resulting data $\{(s, a, s')_j\}$ to $\mathcal{D}$

+, Problems: sometimes we make mistakes
$\Rightarrow$ learn faster if we correct the mistake more often

Model-based RL v.1.5:

1, Run base policy $\pi_0(a_t|s_t) \Rightarrow$ collect $\mathcal{D} = \{(s, a, s')_i\}$
2, Learn model $f(s,a)$ to minimize $\sum_i \| f(s_i, a_i) - s_i' \|^2$
3, Plan through $f(s,a) \Rightarrow$ choose actions
4, Execute the first planned action, observe resulting state s'
5, Append (s, a, s') to $\mathcal{D}$

2-step

⊗ Problems: overfitting early, especially with high-dimensional data

⇒ Introduce: **uncertainty estimation**

3, Take action with high expected reward

⊗ This goes against exploration

Depends on problems: use { expected / optimistic / pessimistic } value / planning

+) How to get uncertainty-aware model

- Not use output entropy: aleatoric / statistical uncertainty vs epistemic / model uncertainty
- Bayesian neural network: $p(\theta|\mathcal{D}) = \prod_i p(\theta_i|\mathcal{D})$, $p(\theta_i|\mathcal{D}) = \mathcal{N}(\mu_i, \sigma_i)$ — complicate
- Bootstrap ensembles: $p(\theta|\mathcal{D}) \approx \frac{1}{N}\sum_i \delta(\theta_i)$

  Sampled with replacement from $\mathcal{D}$, usually less than 10 models

⊗ How to plan with uncertainty?

In step 3, we do optimal control to choose action

In optimal control, for example LQR:

- Before $J(a_1, \dots a_H) = \sum_{t=1}^{H} r(s_t, a_t)$ where $s_{t+1} = f(s_t, a_t)$
- Now: $J(a_1, \dots, a_H) = \frac{1}{N}\sum_i^{\sim}\sum_t^{H} r(s_{t,i}, a_{ti})$ where $f_i(s_{t,i}, a_t) = s_{t+1,i}$

  distribution over deterministic model

1, Sample $\theta \sim p(\theta|\mathcal{D})$

2, At each time step t, sample $s_{t+1} \sim p(s_{t+1}|s_t, a_t, \theta)$

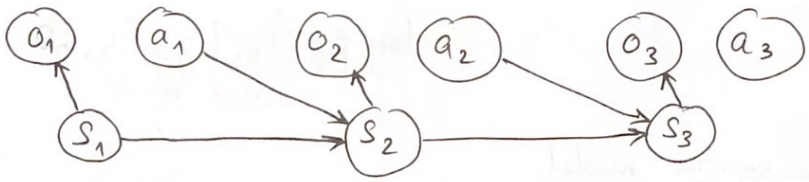
3, Calculate $R = \sum_t r(s_t, a_t)$

4, Repeat step 1 → 3 and accumulate average reward

? → (Generalized for stochastic model)

to deal with complex observations (images ...)

Model-based RL with state/latent space models



$p(o_t | s_t)$ observation model

$p(s_{t+1} | s_t, a_t)$ dynamics model

$p(r_t | s_t, a_t)$ reward model

+) State space (latent space) models:
Input is given observations, not states

+) Before: all model-based RL algorithm: $\max_\phi \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \log p_\phi(s_{t+1,i} | s_{t,i}, a_{t,i})$
Fully observable model

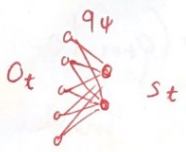
Now: latent space model $\max_\phi \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} E\left[\log p_\phi(s_{t+1,i} | s_{t,i}, a_{t,i}) + \log p_\phi(o_{t,i} | s_{t,i})\right]$

expectation with regards to $(s_t, s_{t+1}) \sim p(s_t, s_{t+1} | o_{1:T}, a_{1:T})$ very complicate

⇒ Can do it by learn approximate posterior $q_\psi(s_t | o_{1:t}, a_{1:t})$ "encoder"

+) Full smoothing posterior: $q_\psi(s_t, s_{t+1} | o_{1:T}, a_{1:T})$
look at all the observations & action in trajectory

+) Single-step encoder: $q_\psi(s_t | o_t)$
look at only 1 observation



↑ most accurate
↓ most complicate (big RNN ..)
↑ simplest (CNN ..)
↓ least accurate

(*) Simple special case: $q(s_t | o_t)$ is deterministic
$q_\psi(s_t | o_t) = \delta(s_t = g_\psi(o_t)) \Rightarrow s_t = g_\psi(o_t)$: deterministic encoder

⇒ Problems: to maximize likelihood:

$$\max_{\phi,\psi} \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \log p_\phi(g_\psi(o_{t+1,i}) | g_\psi(o_{t,i}), a_{t,i}) + \log p_\phi(o_{t,i} | g_\psi(o_{t,i}))$$

The whole recipe: model-based RL with latent space

$$\max_{\phi,\psi} \frac{1}{N} \sum_{i=1}^{N} \sum_{t=1}^{T} \log p_\phi\left(g_\psi(o_{t+1,i}) \mid g_\psi(o_{t,i}), a_{t,i}\right) + \log p_\phi\left(o_{t,i} \mid g_\psi(o_{t,i})\right) + \log p_\phi\left(r_{t,i} \mid g_\psi(o_{t,i})\right)$$

latent space dynamics — image reconstruction — reward model

This is deterministic encoder model

Model-based RL with latent space model v. 1.5:

1, Run base policy $\pi_0(a_t \mid o_t)$ to collect $\mathcal{D} = \{(o, a, o')_i\}$

every N steps:

2, Learn $p_\phi(s_{t+1} \mid s_t, a_t)$, $p_\phi(r_t, s_t)$, $p(o_t \mid s_t)$, $g_\psi(o_t)$ — Learn model

3, Plan through the model to choose actions (any, like MCTS, LQR, random shooting..)

4, Execute the first planned action, observing result o' (MPC)

5, Append (o, a, o') to dataset $\mathcal{D}$

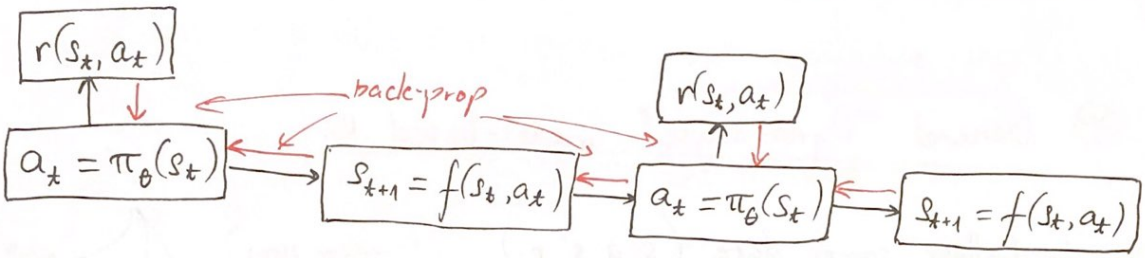
Also can learn directly in observation space:

Learn $p(o_{t+1} \mid o_t, a_t)$ (take in image → split out image)

Gigantic model: RNN..

⊗ Open-loop case: observe only initial state ⇒ formulate a complete plan
⇒ commit to that plan

Closed-loop case: get state feed-back ⇒ learn policy



⊗ +) Model-based RL v. 2.0:

1) Run base policy $\pi_0(a_t|s_t)$ ⇒ collect $\mathcal{D} = \{(s, a, s')_i\}$
2) Learn dynamics model $f(s,a)$ to minimize $\sum_i ||f(s_i, a_i) - s_i'||^2$
⊗ 3) Back-propagate through $f(s,a)$ into the policy to optimize $\pi_\theta(a_t|s_t)$
4) Run $\pi_\theta(a_t|s_t)$, appending the visited tuples (s,a,s') to $\mathcal{D}$

⊗ Problem: (like RNN) exploding / vanishing gradients

⇒ Solutions: +) Use derivative-free (model-free) algorithms with the model used to generate synthetic samples
Example: Policy gradients has high variance, which can be reduced with lots of data, which can be generated by learned model

+) Use simpler policies than neural nets
- LQR with learned models (LQR-FLM Fitted local model)
- Train local policies to solve simple tasks
- Combine into global policies via supervised learning

1) Use "model-free" RL algorithm

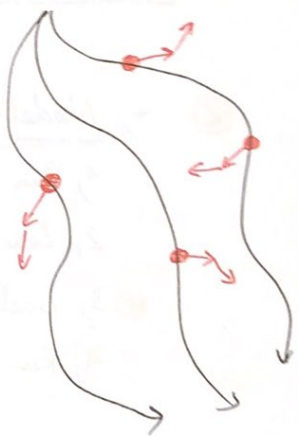
Problems: when generate longer & longer trajectories

If your model is inaccurate (which always is), the longer we roll-out the model, the more these errors compound

⇒ Not very nice for Policy Gradients, ..

⊛ General "Dyna-style" model-based RL:

1) Collect some data (s, a, s', r) (1-million steps)
2) Learn model $\hat{p}(s'|s,a)$ (& optionally $\hat{r}(s,a)$)
3) Repeat K times:
   4) Sample $s \sim \mathcal{B}$ from buffer
   5) Choose action a (from $\mathcal{B}$, π, or random)
   6) Simulate $s' \sim \hat{p}(s'|s,a)$ (and $r = \hat{r}(s,a)$)
   7) Train on (s, a, s', r) with model-free RL
   8) (Optional) take N more model-based steps



⊛ Note: ↑ Step 5: Classic Dyna choose action from Buffer

↑ only requires short roll-outs (as few as 1 step)
still see diverse states

⇒ Solve above problem

Use simpler policies than neural nets

(A) In order to use LQR, we need $\frac{df}{dx_t}$ & $\frac{df}{du_t}$

⇒ Idea: fit $\frac{df}{dx_t}$ & $\frac{df}{du_t}$ around current trajectory / policy

If continuous system sufficiently smooth & initial state distribution quite tight ⇒ do linear regression at every time step

(B) What controller to execute? how to choose u_t

+) Version 0.5: $p(u_t|x_t) = \delta(u_t = \hat{u}_t)$ doesn't correct deviations or drift

+) Version 1.0: $p(u_t|x_t) = \delta\, u_t = K_t(x_t - \hat{x}_t) + k_t + \hat{u}_t)$

Better, but a little too good

We need data to be a little bit cluster, but not too much — still need to be varied

⇒ +) Version 2.0: $p(u_t|x_t) = \mathcal{N}(K_t(x_t - \hat{x}_t) + k_t + \hat{u}_t, \Sigma_t)$

Set $\Sigma_t = Q^{-1}_{u_t, u_t}$

(C) How to fit the dynamics? $p(x_{t+1}|x_t, u_t)$

+) Version 1.0: At each time step using linear regression

$p(x_{t+1}|x_t, u_t) = \mathcal{N}(A_t x_t + B_t u_t + c, N_t)$; $A_t \approx \frac{df}{dx_t}$; $B_t \approx \frac{df}{du_t}$

Problems: linear regression requires number of samples that scale with dimensional states

⇒ +) Version 2.0: fit $p(x_{t+1}|x_t, u_t)$ using Bayesian linear regression

Use your favorite global model as prior

⇒ Can get away with fewer samples

(D) How to stay close to old controller? We want to stay close around local region of trajectories where we have linearize to approximate

Keep KL-Divergence small (between old & new trajectory)

By add that element to lost function

+) Guided policy search:

From LQR, we only get local policy
⇒ need to combine it to global policy

1) Have many

Guided policy search algorithm:

1) Optimize each local policy $\pi_{LQR,i}(u_t|x_t)$ on initial state $x_{0,i}$ w.r.t. $\tilde{c}_{k,i}(x_t, u_t)$ (modified cost to keep $\pi_{LQR,i}$ close to π_θ)

2) Use sample from (1) to train $\pi_\theta(u_t|x_t)$ to mimic each $\pi_{LQR,i}(u_t|x_t)$

3) Update cost function $\tilde{c}_{k+1,i}(x_t, u_t) = c(x_t, u_t) + \lambda_{k+1,i} \log \pi_\theta(u_t|x_t)$

CSME

State estimation

+) Bayes Filters:

$bel(x_t) = p(x_t | z_{1:t}, u_{1:t})$ belief over a state

$\overline{bel}(x_t) = p(x_t | z_{1:t-1}, u_{1:t})$ a posterior (before adapt to z_t)

⇒ Calculating $bel(x_t)$ from $\overline{bel}(x_t)$: correction / measurement update

Algorithm Bayes Filter:

2 for all x_t do:

3 $\overline{bel}(x_t) = \int p(x_t | u_t, x_{t-1}) bel(x_{t-1}) dx$ ⇒ prediction step

4 $bel(x_t) = \eta\, p(z_t | x_t) \overline{bel}(x_t)$ ⇒ update step

⇒ Can only be implemented: for very simple estimation problems
finite state space

⊛ Important Assumption: Markov property (each state is a complete summary of the past)

⇒ Problem: can not be implemented on digital computer

+) The Kalman Filter (KF)

- For continuous state space, not discrete or hybrid
- Assumption: posterior are Gaussians & Markov property

$p(x_t | u_t, x_{t-1})$ must be linear func (linear system dynamics)

$p(z_t | x_t)$ also linear

initial belief (x_0) must be Gaussian

$$x_t = A_t x_{t-1} + B_t u_t + \varepsilon_t$$

$$p(x_t | u_t, x_{t-1}) = \det(2\pi R_t)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(x_t - A_t x_{t-1} - B_t u_t)^T R_t^{-1} (x_t - A_t x_t - B_t u_t)\right\}$$

$$z_t = C_t x_t + \delta_t \quad (= y)$$

$$p(z_t | x_t) = \det(2\pi Q_t)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(z_t - C_t x_t)^T Q_t^{-1} (z_t - C_t x_t)\right\}$$

$$bel(x_0) = p(x_0) = \det(2\pi \Sigma_0)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(x_0 - \mu_0)^T \Sigma_0^{-1} (x_0 - \mu_0)\right\}$$

Algorithm Kalman Filter $(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$

2 $\bar{\mu}_t = A_t \mu_{t-1} + B_t u_t$

3 $\bar{\Sigma}_t = A_t \Sigma_{t-1} A_t^T + R_t$

} incorporate u_t - prediction step $O(n^2)$

4 $K_t = \bar{\Sigma}_t C_t^T (C_t \bar{\Sigma}_t C_t^T + Q_t)^{-1}$ Kalman gain

5 $\mu_t = \bar{\mu}_t + K_t (z_t - C_t \bar{\mu}_t)$

6 $\Sigma_t = (I - K_t C_t) \bar{\Sigma}_t$

} incorporate z_t - correction step $O(n^{2.8})$

7 return μ_t, Σ_t ⇒ belief at time t

⇒ quite computationally expensive, everything are Gaussian

+) **Extended Kalman Filter (EKF)**

overcome linearity assumption by only approximate by Gaussians

$$x_t = g(u_t, x_{t-1}) + \varepsilon_t$$
$$z_t = h(x_t) + \delta_t$$

$$g(u_t, x_{t-1}) \approx g(u_t, \mu_{t-1}) + g'(u_t, \mu_{t-1})(x_{t-1} - \mu_{t-1})$$
$$= g(u_t, \mu_{t-1}) + G_t (x_{t-1} - \mu_{t-1})$$

G_t: Jacobian of state, $n \times n$

$$p(x_t | u_t, x_{t-1}) \approx \det(2\pi R_t)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}[x_t - g(u_t, \mu_{t-1}) - G_t(x_{t-1} - \mu_{t-1})]^T R_t^{-1} [x_t - g(u_t, \mu_{t-1}) - G_t(x_{t-1} - \mu_{t-1})]\right\}$$

$$h(x_t) \approx h(\bar{\mu}_t) + h'(\mu_t)(x_t - \bar{\mu}_t)$$
$$= h(\bar{\mu}_t) + H_t (x_t - \bar{\mu}_t)$$

$$p(z_t | x_t) = \det(2\pi Q_t)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}[z_t - h(x_t)]^T Q_t^{-1} [z_t - h(x_t)]\right\}$$

Algorithm Extended Kalman Filter $(\mu_{t-1}, \Sigma_{t-1}, u_t, z_t)$

$$\bar{\mu}_t = g(u_t, \mu_{t-1})$$
$$\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + R_t$$
$$K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$$
$$\mu_t = \bar{\mu}_t + K_t (z_t - h(\bar{\mu}_t))$$
$$\Sigma_t = (I - K_t H_t) \bar{\Sigma}_t$$
return μ_t, Σ_t

+) Can extend EKF ⇒ multi-hypothesis (extended) Kalman filter (MHEKF)

+) EKF's performance depends on degree of nonlinearities & uncertainty

+) Unscented KF & moments matching KF are better

Gaussian
- moment representation μ & Σ
- canonical representation ξ & Ω
  - Information / precision matrix $\Omega = \Sigma^{-1}$; $\Sigma = \Omega^{-1}$
  - Information vector $\xi = \Sigma^{-1}\mu$; $\mu = \Omega^{-1}\xi$

$$p(x) = \det(2\pi\Sigma)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right\}$$
$$= \eta \exp\left\{-\frac{1}{2} x^T \Omega x + x^T \xi\right\}$$

Algorithm Information filter: (IF) $(\xi_{t-1}, \Omega_{t-1}, u_t, z_t)$

2 $\bar{\Omega}_t = (A_t \Omega_{t-1}^{-1} A_t^T + R_t)^{-1}$ } $O(n^{2.8})$

3 $\bar{\xi}_t = \bar{\Omega}_t (A_t \Omega_{t-1}^{-1} \xi_{t-1} + B_t u_t)$ }

4 $\Omega_t = C_t^T Q_t^{-1} C_t + \bar{\Omega}_t$ } $O(n^2)$

5 $\xi_t = C_t^T Q_t^{-1} z_t + \bar{\xi}_t$ }

Algorithm Extended Information filter

$$x_t = g(u_t, x_{t-1}) + \varepsilon_t$$
$$z_t = h(x_t) + \delta_t$$
$$G_t = g'(u_t, \mu_{t-1})$$
$$H_t = h'(\bar{\mu}_t)$$

2 $\mu_{t-1} = \Omega_{t-1}^{-1} \xi_{t-1}$

3 $\bar{\Omega}_t = (G_t \Omega_{t-1}^{-1} G_t^T + R_t)^{-1}$

4 $\bar{\xi}_t = \bar{\Omega}_t \, g(u_t, \mu_{t-1})$

5 $\bar{\mu}_t = g(u_t, \mu_{t-1})$

6 $\Omega_t = \bar{\Omega}_t + H_t^T Q_t^{-1} H_t$

7 $\xi_t = \bar{\xi}_t + H_t^T Q_t^{-1} [z_t - h(\bar{\mu}_t) - H_t \bar{\mu}_t]$

8 return ξ_t, Ω_t

- Global uncertainty: set $\Omega = 0$ better than set $|\Sigma| = \infty$
- IF tends to be numerically more stable than KF
- IF is better for multi-robot problems
- For high dimensional state, EKF is better computational than EIF

⊛ Major advantage of Gaussian filters ~~is~~ computational

—— disadvantage ———— is assumption on unimodel distribution

Measurements

⊗ Maps: $m = \{m_1 \dots m_N\}$

2 ways to represent a map

feature-based	location-based
m_n : properties of a feature & location of feature	a specific location
only the shape of the environment at the specific locations	volumetric: label for any location in the world
easy to adjust positions of objects	occupancy map
⇒ popular in the robotic mapping field	

+) Correct range with local measurement noise:

$$p_{hit}(z_t^k \mid x_t, m) = \begin{cases} \eta \mathcal{N}(z_t^k; z_t^{k*}, \sigma_{hit}^2) & \text{if } 0 \le z_t^k \le z_{max} \\ 0 & \text{otherwise} \end{cases}$$

+) Unexpected object:

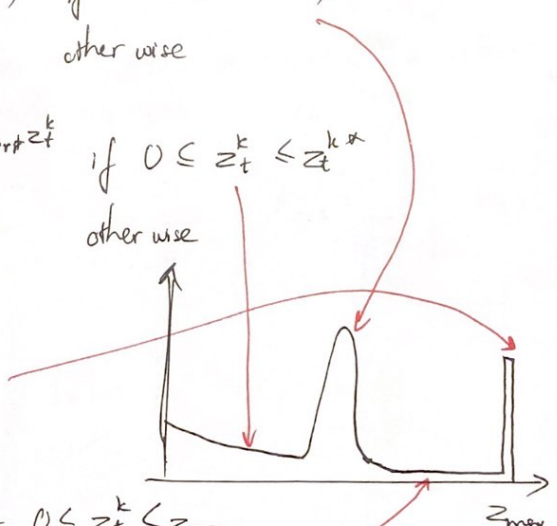
$$p_{short}(z_t^k \mid x_t, m) = \begin{cases} \eta \lambda_{short} e^{-\lambda_{short} z_t^k} & \text{if } 0 \le z_t^k \le z_t^{k*} \\ 0 & \text{otherwise} \end{cases}$$

+) Failures:

$$p_{max}(z_t^k \mid x_t, m) = I(z = z_{max})$$

+) Random measurements

$$p_{rand} \in z_t^k \mid x_t, m) = \begin{cases} \frac{1}{z_{max}} & \text{if } 0 \le z_t^k < z_{max} \\ 0 & \text{otherwise} \end{cases}$$



$$p(z_t^k | x_t, m) = \begin{bmatrix} z_{hit} \\ z_{short} \\ z_{max} \\ z_{rand} \end{bmatrix}^T \cdot \begin{bmatrix} p_{hit}(z_t^k | x_t, m) \\ p_{short}(z_t^k | x_t, m) \\ p_{max}(z_t^k | x_t, m) \\ p_{rand}(z_t^k | x_t, m) \end{bmatrix}$$

Robot motion

Pose $\begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$ } location / orientation

⊛ Motion model (Probabilistic kinematic model): $p(x_t \mid u_t, x_{t-1})$

Velocity commands | Odometry (distance traveled, angle turned..)

- more accurate
- but post-the-fact (not for motion planning

Probabilistic motion planning ⟵ Use for ⟶ estimation

Each has: closed form calculation & sampling algorithm

⊛ **Velocity motion model**: assumes we can control a robot through velocities: $u_t = \begin{bmatrix} v_t \\ \omega_t \end{bmatrix}$

$x_{t-1} = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$; $x_t = \begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix}$

Algorithm Motion model velocity (x_t, u_t, x_{t-1})

Invert the motion model:

$$\mu = \frac{1}{2}\frac{(x-x')c\theta + (y-y')s\theta}{(y-y')c\theta - (x-x')s\theta}$$

$$x^* = \frac{x+x'}{2} + \mu(y-y')$$

$$y^* = \frac{y+y'}{2} + \mu(x'-x)$$

$$r^* = \sqrt{(x-x^*)^2 + (y-y^*)^2}$$

$$\Delta\theta = \text{atan2}(y'-y^*, x'-x^*) - \text{atan2}(y-y^*, x-x^*)$$

compared actual with the commanded:

$$\hat{v} = \frac{\Delta\theta}{\Delta t} r^*$$

$$\hat{\omega} = \frac{\Delta\theta}{\Delta t}$$

$$\hat{\gamma} = \frac{\theta'-\theta}{\Delta t} - \hat{\omega}$$

return $\text{prob}(v-\hat{v}, \alpha_1|v| + \alpha_2|\omega|) \cdot \text{prob}(\omega-\hat{\omega}, \alpha_3|v| + \alpha_4|\omega|) \cdot \text{prob}(\hat{\gamma}, \alpha_5|v| + \alpha_6|\omega|)$

Algorithm Sample motion model velocity: (u_t, x_{t-1})

2 $\hat{v} = v + \text{sample}(\alpha_1 |v| + \alpha_2 |\omega|)$

3 $\hat{\omega} = \omega + \text{sample}(\alpha_3 |v| + \alpha_4 |\omega|)$

4 $\hat{\gamma} = \text{sample}(\alpha_5 |v| + \alpha_6 |\omega|)$

5 $x' = x - \frac{\hat{v}}{\hat{\omega}} s\theta + \frac{\hat{v}}{\hat{\omega}} s(\theta + \hat{\omega}\Delta t)$

6 $y' = y + \frac{\hat{v}}{\hat{\omega}} c\theta - \frac{\hat{v}}{\hat{\omega}} c(\theta + \hat{\omega}\Delta t)$

7 $\theta' = \theta + \hat{\omega}\Delta t + \hat{\gamma}\Delta t$

8 return $x_t = [x', y', \theta']^T$

Prob_normal_distribution (a, b): return $\frac{1}{\sqrt{2\pi b}} e^{-\frac{1}{2}\frac{a^2}{b}}$

Prob_triangular_distribution (a, b): ~~return~~ if $|a| > \sqrt{6b}$ return 0

else return $\frac{\sqrt{6b} - |a|}{6b}$

Sample_normal_distribution (b): return $\frac{b}{6} \sum_{i=1}^{12} \text{rand}(-1, 1)$

Sample_triangular_distribution (b): return $b \cdot \text{rand}(-1, 1) \cdot \text{rand}(-1, 1)$

⊗ Odometry motion model

only available after the robot has moved

⇒ only use for filter algorithm

not for accurate motion planning & control

Algorithm motion mode odometry (x_t, u_t, x_{t-1})

$\bar{x}_{t-1} = [\bar{x} \quad \bar{y} \quad \bar{\theta}]^T$

$\bar{x}_t = [\bar{x}' \quad \bar{y}' \quad \bar{\theta}']^T$

bar ⇔ measurements

hat ⇔ estimations

2 $\delta_{rot1} = \text{atan2}(\bar{y}' - \bar{y}, \bar{x}' - \bar{x}) - \bar{\theta}$

3 $\delta_{trans} = \sqrt{(\bar{x} - \bar{x}')^2 + (\bar{y} - \bar{y}')^2}$

4 $\delta_{rot2} = \bar{\theta}' - \bar{\theta} - \delta_{rot1}$

hypothesized final pose

5 $\hat{\delta}_{rot1} = \text{atan2}(y' - y, x' - x) - \theta$

6 $\hat{\delta}_{trans} = \sqrt{(x - x')^2 + (y - y')^2}$

7 $\hat{\delta}_{rot2} = \theta' - \theta - \hat{\delta}_{rot1}$

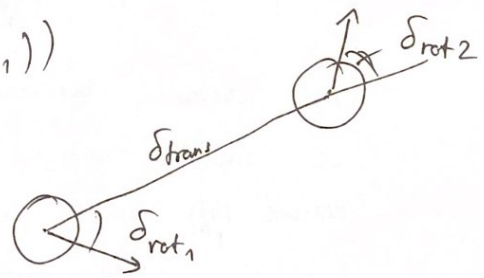
⇒ inverse motion model

8 $p_1 = \text{prob}(\delta_{rot1} - \hat{\delta}_{rot1}, \alpha_1 \hat{\delta}_{rot1} + \alpha_2 \hat{\delta}_{trans})$

9 $p_2 = \text{prob}(\delta_{trans} - \hat{\delta}_{trans}, \alpha_3 \hat{\delta}_{trans} + \alpha_4 (\hat{\delta}_{rot1} + \hat{\delta}_{rot2}))$

10 $p_3 = \text{prob}(\delta_{rot2} - \hat{\delta}_{rot2}, \alpha_1 \hat{\delta}_{rot2} + \alpha_2 \hat{\delta}_{trans})$

11 return $p_1 \cdot p_2 \cdot p_3 \quad (= p(x_t \mid u_t, x_{t-1}))$



Algorithm Sample motion model odometry (u_t, x_{t-1})

2 $\delta_{rot1} = \operatorname{atan2}(\bar{y}' - \bar{y}, \bar{x}' - \bar{x}) - \bar{\theta}$

3 $\delta_{trans} = \sqrt{(\bar{x} - \bar{x}')^2 + (\bar{y} - \bar{y}')^2}$

4 $\delta_{rot2} = \bar{\theta}' - \bar{\theta} - \delta_{rot1}$

5 $\hat{\delta}_{rot1} = \delta_{rot1} - \text{sample}(\alpha_1 \delta_{rot1} + \alpha_2 \delta_{trans})$

6 $\hat{\delta}_{trans} = \delta_{trans} - \text{sample}(\alpha_3 \delta_{trans} + \alpha_4(\delta_{rot1} + \delta_{rot2}))$

7 $\hat{\delta}_{rot2} = \delta_{rot2} - \text{sample}(\alpha_1 \delta_{rot2} + \alpha_2 \delta_{trans})$

8 $x' = x + \hat{\delta}_{trans}\, c(\theta + \hat{\delta}_{rot1})$

9 $y' = y + \hat{\delta}_{trans}\, s(\theta + \hat{\delta}_{rot1})$

10 $\theta' = \theta + \hat{\delta}_{rot1} + \hat{\delta}_{rot2}$

11 return $x_t = (x', y', \theta')^T$

(X) Map-based motion model $p(x_t \mid u_t, x_{t-1}, m)$

- Occupancy maps: $p(x_t \mid m) = 0$ iff the robot collides

- If distance between $x_{t-1} \to x_t$ is small enough ($<$ half robot's diameter) we can estimate: $p(x_t \mid u_t, x_{t-1}, m) \approx \eta\, p(x_t \mid u_t, x_{t-1})\, p(x_t \mid m)$
(discard info relating robot path to x_t)

Algorithm motion model with map (x_t, u_t, x_{t-1}, m)

return $p(x_t \mid u_t, x_{t-1}) \cdot p(x_t \mid m)$

Algorithm Sample motion model with map (u_t, x_{t-1}, m)

do
  $x_t = \text{sample\_motion\_model}(u_t, x_{t-1})$
  $\pi = p(x_t \mid m)$
until $\pi > 0$
return $\langle x_t, \pi \rangle$