MDP

Markov Model: discrete variables are share the same infinite domain

$x_0 \to x_1 \to x_2 \to \cdots \to x_n \to \cdots$

$P(x_0)$ $P(X_t|X_{t-1})$

Transition Model: $P(X_t|X_{t-1})$ → how state evolves

Stationarity assumption: same transition probabilities at all time steps

Joint distribution: $P(X_0,\cdots,X_n) = P(X_0)\prod_{t=1}^{n} P(X_t|X_{t-1})$

Markov assumption: X_{t+1} are independent of $X_0,\cdots,X_{t-1}$ given X_t.

⇒ Each time step only depends on previous

k-th order markov model: depends on k earlier steps.

Markov Random walk

State: location on the unbounded integer line

Initial probability: starts at 0

Transition model: $P(X_t = k|X_{t-1} = k\pm1) = 0.5$

Applications: partical motion in crystal, stock price, gambling, ...

Eg. whether state {rain, sun}

$P(X_0)$ { sun 0.5, rain 0.5 }

$P(X_t|X_{t-1})$:

X_{t-1} \ X_t	sun	rain
sun	0.9	0.1
rain	0.3	0.7

predict: time 0: <0.5, 0.5>

$P(X_1) = \Sigma_{x_0} P(X_1, X_0 = x_0) = \Sigma_{x_0} P(x_0) P(X_1|X_0 = x_0)$

$= 0.5<0.9, 0.1> + 0.5<0.3, 0.7>$

$= <0.6, 0.4>$

$P(X_2) = \Sigma_{x_1} P(x_1) P(X_2|X_1 = x_1)$

$= 0.6<0.9, 0.1> + 0.4<0.3, 0.7>$

$= <0.66, 0.34>$

State at time step t:

$P(X_t) = \Sigma_{x_{t-1}} P(X_{t-1} = x_{t-1}) P(X_t|X_{t-1} = x_{t-1})$

Iterate from $t = 0$

Stationary distribution

The distribution end up with is called the stationary distribution P_∞ of the chain.

⇒ $P_\infty(X) = P_{\infty+1}(X) = \Sigma_x P(X|x) P_\infty(x)$

eg. Rain and sun

$P_\infty(sun) = 0.9 P_\infty(sun) + 0.3 P_\infty(rain)$

$P_\infty(rain) = 0.1 P_\infty(sun) + 0.7 P_\infty(rain)$

Also, $P_\infty(rain) + P_\infty(sun) = 1$

⇒ $P_\infty(sun) = \frac{3}{4}$, $P_\infty(rain) = \frac{1}{4}$

or by matrix: $P_\infty = P_{\infty+1} = T^T P_\infty$

$\begin{bmatrix}0.9 & 0.3\\0.1 & 0.7\end{bmatrix}\begin{bmatrix}P\\1-P\end{bmatrix} = \begin{bmatrix}P\\1-P\end{bmatrix} \Rightarrow P = 0.75$

HMM Hidden Markov Model

Joint distribution for MM: $P(X_0,\cdots,X_T) = P(X_0)\prod_{t=1}^{T} P(X_t|X_{t-1})$

Joint distribution for HMM:

$P(X_0, X_1, E_1, \cdots X_T, E_T) = P(X_0)\prod_{t=1}^{T} P(X_t|X_{t-1}) P(E_t|X_t)$

E_t: obversed variables. X_t: Hidden variables.

Filtering: $P(X_t|e_{1:t})$ posterior distribution over the most recent state given all evidence.

Prediction: $P(X_{t+k}|e_{1:t})$ for $k>0$ posterior distribution over a future state given all evidence

Smoothing: $P(X_k|e_{1:t})$ for $0<k<t$ posterior distribution over a past state given all evidence

Most likely explanation: $\arg\max_{x_{0:t}} P(x_{0:t}|e_{1:t})$

Assumption: 1. $P(X_{t+1}|x_{0:t}, e_{1:t}) = P(X_{t+1}|x_t)$

2. $P(E_t|X_{0:t}, E_{1:t-1}) = P(E_t|X_t)$

Filtering: recursive filtering

$P(X_{t+1}|e_{1:t+1}) = g(e_{t+1}, P(X_t|e_{1:t})) = P(X_{t+1}|e_{1:t}, e_{t+1})$

$= \alpha P(e_{t+1}|X_{t+1}) \Sigma_{x_t} P(x_t|e_{1:t}) P(X_{t+1}|x_t)$

Normalize ($\frac{1}{P(e_{t+1}|e_{1:t})}$), update, predict

⇒ $f_{1:t+1} = \text{Forward}(f_{1:t}, e_{t+1})$, $= \sum_{X_{t+1}} P(b_{t+1}|e_{1:t})$, $f_{1:0} = P(X_0)$

Time complexity: $O(|X|^2)$ where $|X|$ is the number of states.

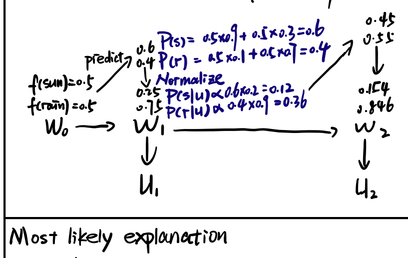
Example: Weather & Umbrella

$P(W_t|W_{t-1})$:

W_{t-1} \ W_t	sun	rain
sun	0.9	0.1
rain	0.3	0.7

$P(U_t|W_t)$:

W_t	true	false
sun	0.2	0.8
rain	0.1	0.9



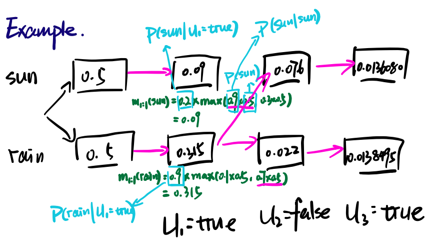
Most likely explanation

Viterbi Algorithm: $\arg\max_{x_{1:t}} P(x_{0:t}|e_{1:t})$

$m_{1:t+1} = P(e_{t+1}|X_{t+1}) \max_{x_t} P(X_{t+1}|x_t) m_{1:t}[x_t]$

$m_{1:t}[x_t] = \max_{x_{1:t-1}} P(x_{1:t}|e_{1:t})$

Example.



DBN. Dynamic Bayes Network

Repeat a fixed Bayes net structure at each time.

Variables from time t can condition on those from t-1

Each HMM is Cartesian product of DBN state variable

⇒ eg. 3 binary state variables ⇒ one state var with 2^3 domain

Advantage DBN v.s. HMM:

sparse dependencies ⇒ exponentially fewer parameters.

E.g. 20 binary state variables, 2 parent each

⇒ DBN has $20 \times 2^{2+1} = 160$ parameters (→ 2 parents with self)

HMM has $2^{20} \times 2^{20} \approx 10^{12}$ parameters.

Partical Filtering

1. Sample N partical by distribution $P(X_0)$
2. transsion by $P(X_{t+1}|X_t)$
3. give weight $w = P(e_{t+1}|X_{t+1})$ for each particle.
4. resample from weight and particle position.

(将权重之和视作概率，再从此后作为分布采样).

Markov Decision Process

Define: 1. state $s \in S$ 2. action $a \in A$

3. transition function $T(s,a,s')$

4. reward function $R(s,a,s')$

5. Start state s_0 6. Maybe terminal state.

Discounting: γ. values of rewards decay exponentially

Infinite Utilities:

finite horizon: 1. Terminate episodes after a fixed T steps.

2. Given nonstationary policies.

Discounting: use $0<\gamma<1$. $U([r_0,\cdots,r_\infty]) = \sum_{t=0}^{\infty}\gamma^t r_t \le \frac{R_{max}}{1-\gamma}$

Qstate: value with action

Value Iteration

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

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

Bellman equation

Value Iteration

$V_{k+1} \leftarrow \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_k(s')]$

Time complexity of each iteration: $O(S^2A)$

Theorem: will converge to unique optimal value.

⇒ Definition: Bellman eqution $U_{i+1} \leftarrow BU_i$, $\|U_i\| = \max_s |U(s)|$

proof: $\|BU_i - BU_i'\| \le \gamma \|U_i - U_i'\|$

$B[V^*] = V$

$U_{i+1} = T[U_i]$ converges to V^*

Compute Action from Values.

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

or use Q-value: $\pi^*(s) = \arg\max_a Q^*(s,a)$

Q-Value Iteration

$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma \max_{a'} Q_k(s',a')]$

Policy Iteration

1. Policy Evaluation

$V_{k+1}^{\pi}(s) \leftarrow \sum_{s'} T(s,\pi(s),s')[R(s,\pi(s),s') + \gamma V_k^{\pi}(s')]$

Time complexity: $O(S^2)$, start with $V_0^\pi(s) = 0$

2. Policy Improvement

$\pi_{i+1}(s) = \arg\max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^{\pi_i}(s')]$

Reinforcement learning

Don't know Transition or Reward.

Basic Idea. At time step t, agent following a policy $\pi_\theta(S_t)$

⇒ obtain an observation S_t of the surrounding environment

⇒ produce action a_t

⇒ then, environment will transmit to S_{t+1}, and agent will receive reward r_t

The goal is to maximize the expected cumulative reward as:

$R = E_\pi[\sum_{t=0}^{T-1} r_t(s_t,a_t)]$ $\{s_0,a_0,r_0, s_1,a_1,r_1, \cdots s_t,a_t,r_t, \cdots, s_{T-1},a_{T-1},r_{T-1}\}$ (Transition)

Model Based

1. Learn empirical MDP model

count outcomes s' for each s, a.

directly estimate each entry in $T(s,a,s')$ from counts

discover each $R(s,a,s')$ when we experience the transition.

2. Solve the learned MDP

Use value iteration as before.

Pros: Make efficient use of experiences.

Cons: May not scale to large state space.

RL feed back loop tends to magnify small model errors

Much harder when the environment is partially observable.

Model Free

Use samples instead probability model.

Passive Reinforcement Learning

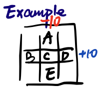
Input: fixed policy $\pi(s)$ Goal: learn state value $V^\pi(s)$

根据 policy $\pi(s)$ 运行一次游戏，然后得到最终的 reward.

然后将 reward 回传给 episode 中每一个 state.

多次运行，将每个 state 的 reward 求平均.

Example



episode 1: B, east, C -1; C, east, D -1; D, exit, x, +10

2: E, north, C -1; C, east, D -1; D, exit, x +10

3: E, north, C, -1; C, east, A, -1; A, exit, x, -10

D: 10 C: 9 B: 8 | D: 10 C: 9 E: 8 | A: -10 C: -11 E: -12

⇒ D: $\frac{10+10}{2} = 10$ C: $\frac{9+9-11}{3} = \frac{7}{3}$ B: 8 A: -10 E: $\frac{8-12}{2} = -2$

Pros: 1. easy to understand 2. Doesn't require T or R

3. It converges to right answer in the limit

Cons: 1. Each state need to be learned seperately

2. Ignore information about state connection

⇒ take long time to learn.

Temporal Difference learning

Sample Based:

Sample $= R(s,\pi(s),s') + \gamma V^\pi(s')$

$V^\pi(s) = (1-\alpha)V^\pi(s) + \alpha \cdot \text{sample} = V^\pi(s) + \alpha(\text{sample} - V^\pi(s))$ (α: learning rate; sample − V: TD error)

Q learning iteration: $Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha[R(s,a,s') + \gamma \max_{a'} Q(s',a')]$

ε-greedy

each step. with small probability ε, act randomly

with large probability 1-ε, act on current policy

Does a lot of stupid things ⇒ Jumping off a cliff lots of time

keeps doing stupid things forever ⇒ decay ε towards 0

Optimistic Exploration Functions

Take value estimate u and a visit count n, and returns an optimistic utility, eg. $f(u,n) = u + \frac{k}{\sqrt{n}}$.

$\Rightarrow$ Modified Q-update:

$$Q(s,a) \leftarrow (1-\alpha)\,Q(s,a) + \alpha\left[R(s,a,s') + \gamma \max_{a'} f(Q(s',a'), n(s',a'))\right]$$

Feature Based Representation

$V_w(s) = w_1 f_1(s) + w_2 f_2(s) + \cdots + w_n f_n(s)$

$Q_w(s,a) = w_1 f_1(s,a) + \cdots + w_n f_n(s,a)$

Update: $L = \frac{1}{2}(\text{diff})^2 \Rightarrow w_i \leftarrow w_i - \alpha \frac{\partial L}{\partial w_i}$

$\Rightarrow w_i \leftarrow w_i + \alpha\left[R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)\right] \frac{\partial Q_w(s,a)}{\partial w}$

$\Rightarrow w_i \leftarrow w_i + \alpha\left[R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)\right] f_i(s,a)$

total error: $\sum_i (y_i - \hat{y}_i)^2 = \sum_i \left(y_i - \sum_k w_k f_k(x_i)\right)^2$

minimize error: $w_m \leftarrow w_m + \alpha\left[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right] f_m(s,a)$

(target: $r + \gamma \max_{a'} Q(s',a')$; prediction: $Q(s,a)$)

Policy Search

Start with an ok solution (eg, Q-learning), then fine tune by hill climbing or gradient ascent on feature weights.

Pros: Works well for partial observability / stochastic policy

Cons: 1. How do we tell the policy got better?
2. Need to run many sample episodes.
3. If there are a lot of features, this can be impractical

Supervised Machine Learning

Naive Bayes $\Rightarrow$ calculate by sample.

Inference: $P(Y, f_1, f_2 \cdots) = \left[\begin{matrix} P(y_1, f_1, \cdots) \\ \cdots \\ P(y_n, f_1, \cdots) \end{matrix}\right] \Rightarrow \left[\begin{matrix} P(y_1)\prod_i P(f_i|y_1) \\ \cdots \\ P(y_n)\prod_i P(f_i|y_n) \end{matrix}\right]$

Parameter Estimation

Example

$P(\text{observation}|\theta) = P(2\text{red}, 1\text{blue}|\theta)$

$= P(\text{red}|\theta)\,P(\text{red}|\theta)\,P(\text{blue}|\theta) = \theta^2(1-\theta)$

$\theta = \arg\max_\theta \theta^2(1-\theta)$ (derivative to 0)

Overfit / Generalize. pass

Smoothing.

Laplace Smoothing: $P_{LAP}(x) = \frac{c(x)+1}{\sum_x (c(x)+1)} = \frac{1+c(x)}{k+\sum_x c(x)}$

Maximum likelihood: $\theta = \arg\max_\theta \prod_i P(X_i|\theta)$

Laplace: $\theta_{MAP} = \arg\max_\theta P(X|\theta)P(\theta)$

$P(\theta)$ is Dirichlet distribute parameterized by k

Precision: $\frac{t}{k}$ Recall: $\frac{t}{r}$

t: current true r: total true k: current sample

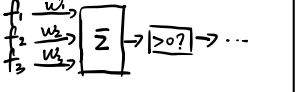
Break-even point: Precision = Recall

F-measure: $F_1 = \frac{2}{\frac{1}{\text{Precision}} + \frac{1}{\text{Recall}}}$

Neural Network

Positive: output +1

Negative: output -1



bias: $y = wf + b$

Decision Boundary

$w = w + y^* \cdot f$ $\quad y^* = \begin{cases} +1 \\ -1 \end{cases}$ > 0

Before: $w \cdot f(x)$ After: $w \cdot f(x) + y^* \cdot f(x) \cdot f(x)$

Perceptron

Separability: true if some parameters get the same training set perfectly correct

Convergence: if the training set is separable, perceptron will eventually converge.

Sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$ (binary case)

Logistic Regression

$\max_w ll(w) = \max_w \sum \log P(y^{(i)}|x^{(i)}; w)$

with $P(y^i=1|x^i;w) = \frac{1}{1+e^{-w\cdot f(x^i)}}$

$P(y^i=-1|x^i;w) = 1 - \frac{1}{1+e^{-w\cdot f(x^i)}}$

multi-class: $P(y^i|x^i;w) = \frac{e^{w_{y^i}\cdot f(x^i)}}{\sum_y e^{w_y \cdot f(x^i)}}$

Unsupervised Learning

Kmeans pass.

optimal: $\phi(\{x_i\}, \{a_i\}, \{c_k\}) = \sum_i \text{dist}(x_i, a_i)$ (points: assignments; means)

GMM

$P(X=x) = \frac{1}{(2\pi)^{\frac{M}{2}}\|\Sigma\|^{\frac{1}{2}}} \exp\left[-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right]$ (Gaussian)

$p(x) = \sum_{k=1}^{N} \pi_k \mathcal{N}(x|\mu_k, \Sigma_k)$, s.t. $\forall k, \pi_k \geq 0, \sum_{k=1}^{N} \pi_k = 1$

$P(Y)$: Distribution over k components (clusters)

$P(X|Y)$: generate data from each μ_i and Σ_i

for m datapoints, for component i, suppose we have n data points with label i:

$\mu_i = \frac{1}{n}\sum_{j=1}^{m} x_j \quad \Sigma_i = \frac{1}{n}\sum_{j=1}^{n}(x_j-\mu_i)(x_j-\mu_i)^T \quad \pi_i = \frac{n}{m}$

Objective: Maximum the likelihood:

$\prod_j P(y_j = i, x_j) = \prod_j \pi_i \mathcal{N}(x_j|\mu_i, \Sigma_i)$

$= \prod_j \sum_i \pi_i \mathcal{N}(x_j|\mu_i, \Sigma_i)$

Use EM Algorithm

E-step: Compute probability of each instance having each possible label

$P(y_j = i|x_j, \theta^t) \propto \pi_i^t \mathcal{N}(x_j|\mu_i^t, \Sigma_i^t)$

M-step: Treating each instance as fractionally having both labels, compute the new parameters.

$\mu_i^{t+1} = \frac{\sum_j P(y_j=i|x_j,\theta^t)x_j}{\sum_j P(y_j=i|x_j,\theta^t)}$

$\Sigma_i^{t+1} = \frac{\sum_j P(y_j=i|x_j,\theta^t)[x_j-\mu_i^{t+1}][x_j-\mu_i^{t+1}]^T}{\sum_t P(y_j=i|x_j,\theta)}$

$\pi_i^{t+1} = \frac{\sum_j P(y_j=i|x_j,\theta)}{m}$

Math Behind EM

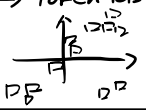
$l(\theta;D) \geq F(\theta,Q) = \sum_{j=1}^{m}\sum_z Q(z|x_j)\log\frac{P(z,x_j|\theta)}{Q(z|x_j)}$

Jensen's inequality

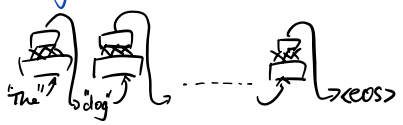
LLM

text $\rightarrow$ tokenize $\rightarrow$ token ids.

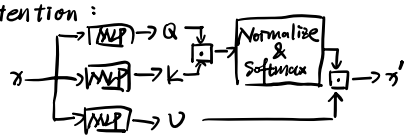
word embeddings:



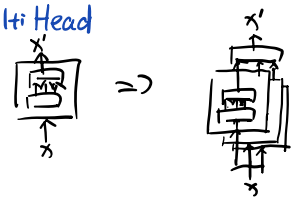
Autoregressive Models



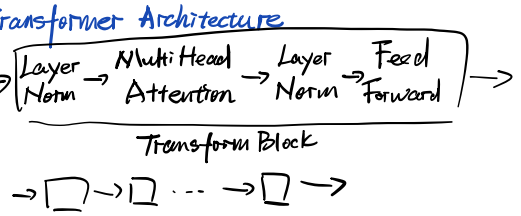
Attention:



Multi Head



Transformer Architecture



pretrain: train a large model with lot of data on self-supervised task

finetune: continue training the same model on task you care about

Policy Search

1. Initialize policy π_θ somehow
2. Estimate policy performance: $J(\theta) = V^{\pi_\theta}(s)$
3. Improve policy:

Hill Climbing: Change θ, evaluate new policy. keep if better

Gradient ascent: Estimate $\nabla_\theta J(\theta)$, change θ to ascent gradient: $\theta_{k+1} = \theta_k + \alpha \nabla_\theta J(\theta_k)$

$J(\theta) = \mathbb{E}_{\tau\sim P_\theta(\tau)}\left[\sum_t r(s_t, a_t)\right] \approx \frac{1}{N}\sum_i \sum_t r(s_{i,t}, a_{i,t})$ (sum over samples from π_θ)

$\theta^* = \arg\max_\theta J(\theta)$

$J(\theta) = \mathbb{E}_{\tau\sim P_\theta(\tau)}[r(\tau)] = \int P_\theta(\tau) r(\tau)\, d\tau$

$\nabla_\theta J(\theta) = \int \nabla_\theta P_\theta(\tau) r(\tau)\, d\tau = \int P_\theta(\tau)\nabla_\theta \log P_\theta(\tau) r(\tau)\, d\tau$

$= \mathbb{E}_{\tau\sim P_\theta(\tau)}[\nabla_\theta \log P_\theta(\tau) r(\tau)]$

Define Advantage functions: $A^\pi(s,a) = Q^\pi(s,a) - V^\pi(s)$

TD error $\mathbb{E}_\pi[\delta_t] = \mathbb{E}_\pi[Q^\pi(s_t,a_t) - V^\pi(s_t)]$

Let t denote a trajectory from an arbitrary episode:

$\nabla_\theta J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{|\tau|} A^\pi(s_t,a_t)\nabla_\theta \log \pi_\theta(a_t|s_t)\right]$

$\approx \frac{1}{N}\sum_i \sum_t [r_t + \gamma V^\pi(s_{t+1}) - V^\pi(s_t)]\nabla_\theta \log \pi_\theta(a_t|s_t)$