HMM

Sequence labeling problem – map sequence of observations to sequence of tags
– $x_1, x_2 \dots x_n \rightarrow y_1, y_2 \dots y_n$
– needs to take into account previous tags

Joint probability distribution
$p(x_1 \dots x_n, y_0, y_1 \dots y_{n+1}) = \prod_{i=1}^{n+1} p(y_i|y_{i-1}) p(y_i|x_i)$ $y_0 = START$, $y_{n+1} = STOP$

$\mathcal{T}$ is set of states including START and STOP states
O is set of observation symbol
$\theta = \langle a, b \rangle$

$a_{u,v} \equiv p(y_{next} = v | y_{curr} = u) = \frac{count(u,v)}{count(u)}$
$b_u(o) \equiv p(x = o | y = u) = \frac{count(u \rightarrow o)}{count(u)}$

Therefore, probability for each possible (x,y) pair is given by:
$p_\theta(x,y) = p(x,y;\theta)$
$= a_{START,y_1} \cdot a_{y_n,STOP} \cdot \prod_{i=1}^{n-1} a_{y_i,y_{i+1}} \cdot \prod_{i=1}^{n} b_{y_i}(x_i)$

Viterbi Algorithm – dynamic programming algorithm

Input: $x = x_1 \dots x_n$ and θ
Output: $\underset{y_1 \dots y_n}{\operatorname{argmax}}\, p(x_1 \dots x_n, y_0, y_1 \dots y_{n+1}; \theta)$
Joint probability for the first k tags:
$r(y_1, \dots, y_k) = \prod_{i=1}^{k} a_{y_{i-1},y_i} \prod_{i=1}^{k} b_{y_i}(x_i)$
$p(x_1 \dots x_n, y_0, y_1 \dots y_{n+1}) = r(y_1 \dots y_n) \cdot a_{y_n, y_{n+1}} = r(y_1 \dots y_n) \cdot a_{y_n, stop}$

Let $S(k,v)$ be the set of tag sequences $y_1 \dots y_k$ such that $y_k = v$
$\Rightarrow S(k,v)$ is set of all sequences of length k whose last tag is v
The dynamic programming algorithm will calculate the following recursively (forward)
$\pi(k,v) = \max_{(y_1 \dots y_k) \in S(k,v)} r(y_1 \dots y_k)$
$\Rightarrow$ solving the maximization problem partially over the tags $y_1 \dots y_{k-1}$ with constraint that we use tag v for y_k

Base Case:
$\pi(0,u) = \begin{cases} 1 & \text{if } u = START \\ 0 & \text{otherwise} \end{cases}$

Moving forward recursively: for any $k \in \{1, \dots, n\}$
$\pi(k,v) = \max_{u \in \mathcal{T}} \{ \pi(k-1,u) \cdot a_{u,v} \cdot b_v(x_k) \}$

Transition from y_n to STOP:
$\max_{y_1 \dots y_n} p(x_1 \dots x_n, y_0, y_1 \dots y_n, y_{n+1}) = \max_{v \in \mathcal{T}} \{ \pi(n,v) \cdot a_{v,STOP} \}$

$\Rightarrow$ time complexity $\rightarrow O(n|\mathcal{T}|^2)$, n = length, $\mathcal{T}$ = # of tags
$y_n^* = \operatorname{argmax} \{ \pi(n,v) \cdot a_{v,STOP} \}$
$y_{n-1}^* = \underset{u}{\operatorname{argmax}} \{ \pi(n-1,u) \cdot a_{u,y_n^*} \}$ and so on

Unsupervised Learning $\rightarrow$ given unlabelled observation sequences xs and a collection of tags but not the tag sequences ys
eg. $\mathcal{T} = \{A, B\}$, $O = \{a, b, c\}$
output sequences: [(a,b), (a,c), (a,b)]

HARD EM (similar to k-means for clustering)
E-step:
each data point x is strictly assigned to a single y
$\rightarrow$ each observation sequence strictly assigned to a single state sequence
given model params and observation sequences: find corresponding state sequences.
1. Run Viterbi algorithm to find most probable state sequence
M-step:
given observation sequences and state sequences, find model parameters
1. Parameter estimation using MLE
$a_{u,v} = \frac{count(u,v)}{count(u)}$ $b_u(o) = \frac{count(u \rightarrow o)}{count(u)}$

SOFT EM:
E-step:
evaluate posterior probability over possible tag sequences
$Count(x^i, y, u \rightarrow v)$: # of times a transition from state u to v occurs in tag sequence y given obs $x^{(i)}$
1. Reassign partial memberships for each $x^{(i)}$
$Count(u;v) = \sum_{i=1}^{m} \sum_{y} p(y|x^{(i)})\, Count(x^i, y, u \rightarrow v)$
M-step:
1. Re-estimate transition probabilites based on expected counts
$a_{u,v} = \frac{Count(u;v)}{\sum_{v'} Count(u;v')}$
where $\sum_{v'=1}^{N+1} a_{u,v'} = 1$

Problem $\rightarrow$ calculating sum over possible tag sequences in E-step has exponential hidden state sequences y.
Suppose we can efficiently calculate marginal probabilities:
$Count(u;v) = \sum_{i=1}^{m} \sum_{y} p(y|x^{(i)})\, Count(x^i, y, u \rightarrow v) = \sum_{i=1}^{m} \sum_{j=0}^{n} p(y_j = u, y_{j+1} = v | x^{(i)}; \theta)$

Inference

How to efficiently compute a simpler marginal posterior probability
$p(y_j = u | x; \theta) = \sum_{y: y_j = u} p(y|x;\theta) = \frac{p(y_j = u, x; \theta)}{p(x;\theta)} = \frac{p(x_1, x_2 \dots, x_{j-1}, y_j = u, x_j, x_{j+1}, \dots, x_n; \theta)}{p(x_1, x_2 \dots x_n; \theta)}$
Numerator can be written as:
$p(x_1, x_2 \dots x_{j-1}, y_j = u, x_j, \dots x_n; \theta) = p(x_1, x_2 \dots x_{j-1}, y_j = u) \cdot p(x_j, x_{j+1}, \dots, x_n | x_1, x_2 \dots x_{j-1}, y_j = u; \theta)$
in HMM, given y_j, $x_j \dots x_n$ are conditionally independent of $x_1 \dots x_{j-1}$. $\therefore$
$p(x_1, x_2 \dots x_{j-1}, y_j = u, x_j, \dots x_n; \theta) = p(x_1, x_2 \dots x_{j-1}, y_j = u) \cdot p(x_j, x_{j+1}, \dots, x_n | y_j = u; \theta)$

Forward Probability ($\alpha_u(j)$)
$\alpha_u(j) = p(x_1, x_2 \dots x_{j-1}, y_j = u; \theta)$
$\Rightarrow$ probability of emitting the symbols $x_1 \dots x_{j-1}$ and ending in state u with position j without yet emitting the corresponding output symbol
Backward Probability ($\beta_u(j)$)
$\beta_u(j) = p(x_j, x_{j+1} \dots x_n | y_j = u; \theta)$
$\Rightarrow$ probability of emitting symbols $x_j \dots x_n$ and final STOP state given we began in state u in position j.

$p(x;\theta) = p(x_1, x_2 \dots x_n; \theta) = \sum_{u} p(x_1, x_2 \dots, x_{i-1}, y_i = u, x_i, \dots, x_n; \theta)$
$= \sum_{u'} p(x_1, x_2 \dots, x_{i-1}, y_i = u'; \theta) \cdot p(x_i, x_{i+1} \dots, x_n | y_i = u'; \theta) = \sum_{u'} \alpha_{u'}(i) \beta_{u'}(i)$

These give us:
$p(y_j = u, y_{j+1} = v | x; \theta) = \frac{p(x_1, x_2 \dots, x_{j-1}, y_j = u; \theta) \cdot p(y_{j+1} = v | y_j = u) \cdot p(x_j | y_j = u) \cdot p(x_{j+1}, \dots, x_n | y_{j+1} = v; \theta)}{p(x;\theta)}$
$= \frac{\alpha_u(j) \cdot a_{u,v} \cdot b_u(x_j) \cdot \beta_v(j+1)}{\sum_{u'} \alpha_{u'}(i) \beta_{u'}(i)}$

Sum over all possible tag sequences that include the transition $y_j = u$ and $y_{j+1} = v$ and generates the observations, divided by sum of all sequences that generate the observations
$\rightarrow$ obtain relative probability of transition, relative to all the alternatives given the observations (posterior)

Forward Backward Algorithm
For a state sequence $y_1, y_2 \dots y_n$ there is:
- a path through the graph that has sequence of states START, $\langle 1, y_1 \rangle, \dots \langle n, y_n \rangle$, STOP
- path associated with state sequence $y = y_1 \dots y_n$ has score: $p(x, y; \theta)$
- $\alpha_u(j)$ is sum of scores of all paths from START to state $\langle j, u \rangle$
- $\beta_u(j)$ is sum of scores of all paths from state $\langle j, u \rangle$ to STOP
- Given input sequence $x_1, x_2 \dots x_n$ for $u \in 1, \dots N$, $j \in 1, \dots n$, the forward and backward probability can be calculated recursively.

Forward probability
$\alpha_u(j) = p(x_1, \dots x_{j-1}, y_j = u)$
Base case:
$\alpha_u(1) = a_{START,u}$ $\forall u \in$
Recursive case:
$\alpha_u(j+1) = \sum_{v} \alpha_v(j)\, a_{v,u}\, b_v(x_j)$ $\forall u \in 1, \dots, N$, $j = 1, \dots, n-1$
Backward probability
$\beta_u(j) = p(x_j, \dots, x_n | y_j = u; \theta)$
Base case:
$\beta_u(n) = a_{u,stop}\, b_u(x_n)$ $\forall u \in 1, \dots, N-1$
Recursive case:
$\beta_u(j) = \sum_{v} a_{u,v}\, b_u(x_j)\, \beta_v(j+1)$ $\forall u \in 1, \dots, N-1$, $j = n-1, \dots, 1$

Transition Parameters
$Count(u;v) = \sum_{i=1}^{m} Count^{(i)}(u;v)$, $Count(u) = \sum_{i=1}^{m} Count^{(i)}(u)$
$Count^{(i)}(u;v) = \sum_{j=1}^{n} P(y_j = u, y_{j+1} = v | x^{(i)}; \theta^{t-1})$
$= \sum_{j=1}^{n} \frac{1}{Z} \alpha_u(j) b_u(x_j^{(i)}) a_{u,v} \beta_v(j+1)$, $Z = \sum_{u'} \alpha_{u'}(j) \beta_{u'}(j)$
$Count^{(i)}(u) = \sum_{j=1}^{n} p(y_j = u | x^{(i)}; \theta^{t-1})$
$= \sum_{j=1}^{n} \frac{1}{Z} \alpha_u(j) \beta_u(j)$, $Z = \sum_{u'} \alpha_{u'}(j) \beta_{u'}(j)$

Emission Parameters
$Count(u \rightarrow o) = \sum_{i=1}^{m} Count^{(i)}(u \rightarrow o)$
$Count^{(i)}(u \rightarrow o) = \sum_{j=1}^{n} p(y_j = u, x_j^{(i)} = o | x^{(i)}; \theta^{t-1})$
$= \sum_{j=1 \text{ s.t. } x_j^{(i)} = o}^{n} \frac{1}{Z} \alpha_u(j) \beta_u(j)$, $Z = \sum_{u'} \alpha_{u'}(j) \beta_{u'}(j)$
$\Rightarrow$ can compute E-step of soft EM

Max-marginal Decoding
$p(y_i = u | x_1, \dots, x_n; \theta) = \frac{p(x_1, \dots, x_{i-1}, y_i, x_i, \dots, x_n)}{p(x_1, \dots x_n; \theta)}$
$= \frac{\alpha_u(i) \beta_u(i)}{\sum_{v} \alpha_v(i) \beta_v(i)}$
$y_i^* = \underset{u}{\operatorname{arg\,max}}\, p(y_i = u | x_1 \dots x_n; \theta) = \underset{u}{\operatorname{argmax}} \frac{\alpha_u(i) \beta_u(i)}{\sum_v \alpha_v(i) \beta_v(i)}$
$= \underset{u}{\operatorname{arg\,max}}\, \alpha_u(i) \beta_u(i)$

$\rightarrow$ Viterbi is often better

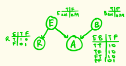
BN

Joint distribution $\rightarrow$ product of conditional probabilities given its parents
$\rightarrow$ if variable has no parents (eg. X_1), then probability is just $P(X_1 = x_1)$

Marginal Independence
$P(X_1 = x_1, X_2 = x_2) = \sum_{x_3} P(X_1 = x_1) P(X_2 = x_2) P(X_3 = x_3 | X_1 = x_1, X_2 = x_2)$
$= P(X_1 = x_1) P(X_2 = x_2)$
X_1 and X_2 are marginally independent of each other: if we don't know the value of X_3, there is nothing that ties the coin flips together

Induced dependence
However, if we know the value of X_3, X_1 and X_2 become dependent (possible for them to still be independent? given its parents (based on probability)

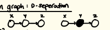
Explaining away
E: Earthquake
B: Burglary
A: Alarm
R: Report

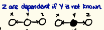


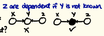
$P(E = e, B = b, A = a, R = r) = P(E = e) P(B = b) P(A = a | E = e, B = b) P(E = r | E = e)$ [product of variable given its parents]
If we observe alarm went off – at least one of E=T or B=T should have occurred (most likely not both)
To calculate $P(B = T | A = T)$, first find $P(B = b, A = T)$
$P(B = b, A = T) = \sum_{e \in \{T,F\}} \sum_{r \in \{T,F\}} P(E = e) P(B = b) P(A = a | E = e, B = b) P(R = r | E = e)$
$= \sum_{e \in \{T,F\}} P(E = e) P(B = b) P(A = a | E = e, B = b) \underbrace{\sum_{r \in \{T,F\}} P(R = r | E = e)}_{= 1}$
$= P(B = b) \sum_{e \in \{T,F\}} P(E = e) P(A = a | E = e, B = b)$
$P(B = T | A = T) = \frac{P(B = T, A = T)}{\sum_{b \in \{T,F\}} P(B = b, A = T)}$
$= \frac{0.01 \cdot (\dots)}{0.01 \cdot (\dots) + (\dots)}$ = slightly above 0.5 $\Rightarrow$ because of chance of both B=T and E=T.

If we now hear a report about earthquake $\rightarrow$ we now know that E=T, removing any evidence about B=T
$\therefore P(B = T | A = T, R = T) = P(B = T) = 0.01$
$P(E = T | A = T, R = T) = 1$

$\Rightarrow$ R and B are dependent given A=T (induced dependence)

Independence from graph: D-seperation
Chain:

X and Z dependent?
$P(Z | X, Y) = \frac{P(X,Y,Z)}{P(X,Y)} = \frac{P(X) P(Y|X) P(Z|Y)}{P(X) P(Y|X)} = P(Z|Y)$
$\Rightarrow$ shaded means value of variable is known
$\Rightarrow$ X and Z are independent given Y
$\Rightarrow$ X and Z are dependent if Y is not known

Common Cause:

X and Z dependent?
$P(Z | X, Y) = \frac{P(X,Y,Z)}{P(X,Y)} = \frac{P(Y) P(X|Y) P(Z|Y)}{P(Y) P(X|Y)} = P(Z|Y)$
$\Rightarrow$ X and Z are independent given Y
$\Rightarrow$ X and Z are dependent if Y is not known

Explaining away:

X and Z dependent?
$P(X, Z) = \sum_{Y} P(X) P(Z) P(Y | X, Z) = P(X) P(Z) \sum_{Y} P(Y|X,Z) = P(X) P(Z)$
$\Rightarrow$ X and Z are dependent given Y
$\Rightarrow$ X and Z are independent without knowing Y

Joint Distribution:
$P(X_1 = x_1, \dots X_d = x_d) = \prod_{i=1}^{d} P(X_i = x_i | X_{pa_i} = x_{pa_i}) = \prod_{i=1}^{d} \theta_i(x_i | x_{pa_i})$
$\Rightarrow \theta_i(x_i | x_{pa_i})$ is the probability table for variable i
Number of free parameters:
$dim(G) = \sum_{i=1}^{d} (r_i - 1) \prod_{j \in pa_i} r_j$
Log-likelihood of data D for a graph G (ie how well does graph G explain data D)
$\ell(D; G) = \sum_{t=1}^{n} \log \left[\prod_{i=1}^{d} \theta_i(x_i^{(t)} | x_{pa_i}^{(t)}) \right] = \sum_{t=1}^{n} \sum_{i=1}^{d} \log \theta_i(x_i^{(t)} | x_{pa_i}^{(t)}) = \sum_{i=1}^{d} \left[\sum_{t=1}^{n} \log \theta_i(x_i^{(t)} | x_{pa_i}^{(t)}) \right]$
Using values from the table $\theta_i(x_i | x_{pa_i})$
$\sum_{t=1}^{n} \log \theta_i(x_i^{(t)} | x_{pa_i}^{(t)}) = \sum_{x_i, x_{pa_i}} count(x_i, x_{pa_i} \text{ in } D) \cdot \log \theta_i(x_i | x_{pa_i})$

Therefore the corresponding maximum likelihood parameter estimate for $\theta_i(x_i | x_{pa_i})$:
$\hat{\theta}_i(x_i | x_{pa_i}) = \frac{count((x_i, x_{pa_i}) \text{ in } D)}{count((x_{pa_i}) \text{ in } D)}$, $x_i \in \{1, \dots, r_i\}$
$\rightarrow \sum_{x_i'} count((x_i', x_{pa_i}) \text{ in } D)$

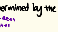
Learning the graph structure
We cannot use only the maximum likelihood to determine the graph structure
$\because$ it favours graphs that have more parameters to fit the data
Therefore, we use the Bayesian Information Criterion (BIC)
$BIC(D; \hat{\theta}, G) = \ell(D; \hat{\theta}, G) - \frac{dim(G)}{2} \log(n)$
$dim(G) = \sum_{i=1}^{d} (r_i - 1) \prod_{j \in pa_i} r_j$ $\rightarrow$ # of free parameters

We can also write it as:
$Score(i | pa_i; D) = \sum_{t=1}^{n} \log \hat{\theta}_i(x_i^{(t)} | x_{pa_i}^{(t)}) - \frac{1}{2} \left[(r_i - 1) \prod_{j \in pa_i} r_j \right] \log(n)$
$BIC(D; G) = \sum_{i=1}^{d} score(i | pa_i; D)$

$\Rightarrow$ to maximize BIC score: local search / dynamic programming
complete data $\rightarrow$ cond. prob estimates for each variable $\rightarrow$ parent selection scores for each variable $\rightarrow$ decomposable scoring func. for graphs $\rightarrow$ highest scoring acyclic graph

RL

HMM – use states y_t to encode position of the robot and observations x_t to capture sensor readings
process is markov as transition to the next state is only determined by the current state.



$T(i,j) = p(y_{t+1} = j | y_t = i)$
$T(i,k,j) = P(y_{t+1} = j | y_t = i, a_t = k)$, $i \rightarrow j$ with action k
Rewards: associated with states: $R(s_t)$
$\rightarrow$ can also take into account action that brings robot to next state: $R(s_{t-1}, a_{t-1}, s_t)$
Utility function: long term reward for a robot when it takes actions
To prevent infinite rewards (loops):
1. Assume robot has finite horizon (after N steps utility value does not change)
$U([s_0, s_1 \dots s_N \dots s_{N+k}]) = U([s_0, s_1 \dots s_N])$ $\forall k \geq 1$
2. Discounted Rewards (Has finite value even for infinite sequences) ✓
$U([s_0, s_1 \dots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots = \sum_{t=0}^{\infty} \gamma^t R(s_t)$
$U([s_0, s_1 \dots]) \leq \sum_{t=0}^{\infty} \gamma^t R_{max} = \frac{R_{max}}{1-\gamma}$ $\Big\}$ $0 \leq \gamma < 1$
$\rightarrow$ larger γ means longer sequences
Policy π: function that maps each state to an action
goal $\rightarrow$ find optimal policy that maximizes utility $\rightarrow$ action chosen to maximize discounted rewards

-0.01	-0.01	-0.01	+1
-0.01		-0.01	-1
-0.01	-0.01	-0.01	-0.01

→	→	→	+1
↑		←	-1
↑	←	←	↓

$R(s) = -0.01$

-2	-2	-2	+1
-2		-2	-1
-2	-2	-2	-2

→	→	→	+1
↑		→	-1
↑	→	→	↑

$R(s) = -2$

Markov Decision Process $\rightarrow$ provided with:
- set of states S
- set of actions A
- transition probability function $T(s,a,s') = p(s'|s,a)$
- Reward function $R(s,a,s')$ or $R(s)$

Reinforcement Learning $\rightarrow$ provided with:
- set of states S
- set of actions A

Value Iteration – find an optimal policy using MDP.
$\pi(s)$ – particular policy that specifies action we should take at state s.
$V^\pi(s)$ – value of state s under policy π
$Q^\pi(s,a)$ – value of state s and action a under policy π
$\pi^*(s)$ – action to take in state s that gives highest utility
$V^*(s)$ – value of state s under optimal policy $\pi^*(s)$
$Q^*(s,a)$ – value of state s and action a under optimal policy π^*

value: $\rightarrow$ value of rewards for acting optimally from that state

These values are related by:
$V^*(s) = \max_a Q^*(s,a) = Q^*(s, \pi^*(s))$
$Q^*(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$
$V^*(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$
$= \sum_{s'} T(s, \pi^*(s), s')[R(s, \pi^*(s), s') + \underbrace{\gamma V^*(s')}_{\text{recursive}}]$

Value Iteration Algorithm
- Start $V_0^*(s) = 0$, for all $s \in S$
- Given V_i^*, calculate values for all states $s \in S$ (depth $i+1$)
$V_{i+1}^* \leftarrow \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_i^*(s')]$
- Repeat until convergence (V_{i+1}^* is almost V_i^* for all states)
- Optimal Policy
$Q^*(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$
$\pi^*(s) = \underset{a}{\operatorname{arg\,max}}\, Q^*(s,a)$

Reinforcement Learning (II)
$V^*(s) = \max_a Q^*(s,a) = Q^*(s, \pi^*(s))$ – ①
$Q^*(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$ – ② ①→②
$Q^*(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma \max_a Q^*(s,a)]$

Q-value iteration algorithm
- Start with $Q_0^*(s,a) = 0$ for all $s \in S$, $a \in A$
- Given $Q_i^*(s,a)$, calculate the Q-values for all states (depth $i+1$) and for all actions a
$Q_{i+1}^*(s,a) \leftarrow \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma \max_{a'} Q_i^*(s',a')]$
- Repeat until convergence
- Optimal policy
$\pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q^*(s,a)$

Reinforcement Learning – rewards and transitions are not known prior
Two options: 1. Try to learn model, transition probabilities, rewards and solve for optimal policy using th MDP algorithms
2. Learn Q-values directly

Model based Learning:
– assume we can collect information about transitions involving any state s and action a.
$\rightarrow$ Learn T and R through experience
$T(s,a,s') = \frac{count(s,a,s')}{count(s,a)}$
$R(s,a,s') = \frac{\sum_t R_t(s,a,s')}{count(s,a,s')}$ $\rightarrow$ reward observed for the t^{th} time when $s \rightarrow s'$ with a } might differ due to noisy rewards $\because$ avg

Model free Learning:
$\pi^*(s) = \arg\max_a Q^*(s,a) \rightarrow$ Q-values can be learned recursively (assume we know T & R)
Compute the expected value of function $f(x)$: $E[f(x)] = \sum_x p(x) f(x)$
- Model based computation – estimate $p(x)$ from samples and then compute expectation
$x_i \approx p(x)$, $i = 1, \dots, k$
$\hat{p}(x) = \frac{count(x)}{k}$
$E[f(x)] \approx \sum_x \hat{p}(x) f(x)$
- Model-free estimation – estimate expectation directly from samples
$x_i \approx p(x)$, $i = 1, \dots, k$
$E[f(x)] \approx \frac{1}{k} \sum_{i=1}^{k} f(x_i)$

$\rightarrow$ apply model free learning approach to the estimation of Q-values
$Q_{i+1}^*(s,a) \leftarrow \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma \max_{a'} Q_i^*(s',a')]$
$\rightarrow$ repeatedly take one action at a time, observe reward for next state
Sample 1: $R(s,a,s_1') + \gamma \max_{a'} Q_i(s_1', a')$
⋮
Sample k: $R(s,a,s_k') + \gamma \max_{a'} Q_i(s_k', a')$
Average the samples to obtain the Q-value estimate.
$Q_{i+1}(s,a) \leftarrow \frac{1}{k} \sum_{k=1}^{k} [R(s,a,s_k') + \gamma \max_{a'} Q_i(s_k', a')]$
In practice, we observe states only when we move
Update function: (observed $k-1$ samples so far, now updating for k^{th} sample
$Q_{new}(s,a) \leftarrow \frac{(k-1) Q_{old}(s,a) + R(s,a,s_k') + \gamma \max_{a'} Q_{old}(s_k', a')}{k}$
$Q(s,a) \leftarrow Q(s,a) + \frac{1}{k} [R(s,a,s_k') + \gamma \max_{a'} Q(s_k', a') - Q(s,a)]$

Q Learning Algorithm
- Collect a sample: s, a, s' and $R(s,a,s')$
- Update Q values – incoporate new sample to running average over samples
$Q(s,a) \leftarrow (1-\alpha) Q(s,a) + \alpha[R(s,a,s') + \gamma \max_{a'} Q(s',a')]$
$Q(s,a) \leftarrow Q(s,a) + \alpha[R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)]$ } learning rate $\alpha = \frac{1}{k}$

Update has some convergence conditions as the gradient ascent algorithm.
(s,a): each sample, can assign seperate learning rate for each case, $\alpha = \alpha_k(s,a)$, k is # of (s,a) so far.
$\sum_k \alpha_k(s,a) \rightarrow \infty$ and $\sum_k \alpha_k^2(s,a) < \infty$

HMM

Sequence labeling problem - map sequence of observations to sequence of tags

- $x_1, x_2 \dots x_n \to y_1, y_2 \dots y_n$
- needs to take into account previous tags

Joint probability distribution

$p(x_1 \dots x_n, y_0, y_1 \dots y_{n+1}) = \prod_{i=1}^{n+1} p(y_i|y_{i-1}) \prod_{i=1}^{n} p(x_i|y_i)$ $y_0 = START$, $y_{n+1} = STOP$

$\mathcal{T}$ is set of states including START and STOP states

$\mathcal{O}$ is set of observation symbol

$\theta = \langle a, b \rangle$

$a_{u,v} \equiv p(y_{next}=v|y_{curr}=u) = \frac{Count(u;v)}{Count(u)}$

$b_u(o) \equiv p(x=o|y=u) = \frac{Count(u \to o)}{Count(u)}$

Therefore, probability for each possible (x, y) pair is given by:

$p_\theta(x,y) = p(x,y;\theta)$

$= a_{START,y_1} \cdot a_{y_n,STOP} \cdot \prod_{i=1}^{n-1} a_{y_i,y_{i+1}} \cdot \prod_{i=1}^{n} b_{y_i}(x_i)$

Viterbi Algorithm - dynamic programming algorithm

Input: $x = x_1 \dots x_n$ and θ

Output: $\arg\max_{y_1 \dots y_n} p(x_1 \dots x_n, y_0, y_1, \dots y_{n+1}; \theta)$

Joint probability for the first k tags:

$r(y_1, \dots y_k) = \prod_{i=1}^{k} a_{y_{i-1},y_i} \prod_{i=1}^{k} b_{y_i}(x_i)$

$p(x_1 \dots x_n, y_0, y_1 \dots y_{n+1}) = r(y_1, \dots y_n) \cdot a_{y_n,y_{n+1}} = r(y_1, \dots y_n) \cdot a_{y_n,STOP}$

Let $S(k,v)$ be the set of tag sequences $y_1 \dots y_k$ such that $y_k = v$

⇒ $S(k,v)$ is set of all sequences of length k whose last tag is v

The dynamic programming algorithm will calculate the following recursively (forward)

$\pi(k,v) = \max_{(y_1 \dots y_k) \in S(k,v)} r(y_1, \dots y_k)$

⇒ solving the maximization problem partially over the tags $y_1 \dots y_{k-1}$ with constraint that we use tag v for y_k

Base Case:

$$\pi(0,u) = \begin{cases} 1 & \text{if } u = START \\ 0 & \text{otherwise} \end{cases}$$

Moving forward recursively: for any $k \in \{1, \dots, n\}$

$$\pi(k,v) = \max_{u \in \mathcal{T}} \{\pi(k-1,u) \cdot a_{u,v} \cdot b_v(x_k)\}$$

Transition from y_n to STOP:

$$\max_{y_1 \dots y_n} p(x_1 \dots x_n, y_0, y_1 \dots y_n, y_{n+1}) = \max_{v \in \mathcal{T}} \{\pi(n,v) \cdot a_{v,STOP}\}$$

⇒ time complexity → $O(n|\mathcal{T}|^2)$, n = length, $\mathcal{T}$ = # of tags

$y_n^* = \arg\max_v \{\pi(n,v) \cdot a_{v,STOP}\}$

$y_{n-1}^* = \arg\max_u \{\pi(n-1,u) \cdot a_{u,y_n^*}\}$ and so on

Unsupervised Learning → given unlabelled observation sequences xs and a collection of tags but not the tag sequences ys

eg. $\mathcal{T} = \{A, B\}$, $\mathcal{O} = \{a, b, c\}$

output sequences: [(a,b), (a,c), (a,b)]

HARD EM (similar to k-means for clustering)

E-step:

each data point x is strictly assigned to a single y

→ each observation sequence strictly assigned to a single state sequence

given model params and observation sequences: find corresponding state sequences.

1. Run Viterbi algorithm to find most probable state sequence

M-step:

given observation sequences and state sequences, find model parameters

1. Parameter estimation using MLE

$a_{u,v} = \frac{Count(u;v)}{Count(u)}$ $b_u(o) = \frac{Count(u \to o)}{Count(u)}$

SOFT EM:

E-step:

evaluate posterior probability over possible tag sequences

$Count(x^i, y, u \to v)$: # of times a transition from state u to v occurs in tag sequence y given obs $x^{(i)}$

1. Reassign partial memberships for each $x^{(i)}$

$Count(u;v) = \sum_i \sum_y p(y|x^{(i)}) Count(x^i, y, u \to v)$

M-step:

1. Re-estimate transition probabilities based on expected counts

$a_{u,v} = \frac{Count(u;v)}{\sum_{v'} Count(u;v')}$

where $\sum_{v'} a_{u,v'} = 1$

Problem → calculating sum over possible tag sequences in E-step has exponential hidden state sequences y.

Suppose we can efficiently calculate marginal probabilities:

$Count(u;v) = \sum_i \sum_y p(y|x^{(i)}) Count(x^i, y, u \to v) = \sum_i \sum_{j=0}^{n} p(y_j=u, y_{j+1}=v|x^{(i)};\theta)$

Inference

How to efficiently compute a simpler marginal posterior probability

$p(y_i=u|x_i;\theta) = \sum_{y: y_i=u} p(y|x;\theta) = \frac{p(x_1 \dots x_n, y_i=u;\theta)}{p(x;\theta)} = \frac{p(x_1, \dots x_{i-1}, y_i=u, x_i \dots x_n;\theta)}{p(x_1 \dots x_n;\theta)}$

Numerator can be written as:

$p(x_1, x_2 \dots, y_j=u, x_j, \dots x_n;\theta) = p(x_1, x_2 \dots x_{j-1}, y_j=u;\theta) \cdot p(x_j, x_{j+1}, \dots x_n|x_1, x_2 \dots x_{j-1}, y_j=u;\theta)$

in HMM, given y_j, $x_j \dots x_n$ are conditionally independent of $x_1 \dots x_{j-1}$.

$p(x_1, x_2 \dots, y_j=u, x_j \dots x_n;\theta) = p(x_1, x_2, \dots x_{j-1}, y_j=u;\theta) \cdot p(x_j, x_{j+1}, \dots, x_n|y_j=u;\theta)$

Forward Probability ($\alpha_u(j)$)

$\alpha_u(j) = p(x_1, x_2, \dots x_{j-1}, y_j=u;\theta)$

⇒ probability of emitting the symbols $x_1 \dots x_{j-1}$ and ending in state u with position j without yet emitting the corresponding output symbol

Backward Probability ($\beta_u(j)$)

$\beta_u(j) = p(x_j, x_{j+1} \dots x_n|y_j=u;\theta)$

⇒ probability of emitting symbols $x_j \dots x_n$ and final STOP state given we began in state u in position j.

$p(x;\theta) = p(x_1, x_2 \dots x_n;\theta) = \sum_u p(x_1, x_2 \dots x_{j-1}, y_j=u, x_j \dots x_n;\theta) = \sum_u \alpha_u(j)\beta_u(j)$

These give us:

$$p(y_j=u, y_{j+1}=v|x;\theta) = \frac{p(x_1 \dots x_{j-1}, y_j=u) \cdot p(x_j|y_j=u) \cdot p(y_{j+1}=v|y_j=u) \cdot p(x_{j+1} \dots x_n|y_{j+1}=v)}{p(x;\theta)} = \frac{\alpha_u(j) \cdot a_{u,v} \cdot b_u(x_j) \cdot \beta_v(j+1)}{\sum_{u'} \alpha_{u'}(j)\beta_{u'}(j)}$$

Sum over all possible tag sequences that include the transition $y_j=u$ and $y_{j+1}=v$ and generates the observations, divided by sum of all sequences that generate the observations

→ obtain relative probability of transition, relative to all the alternatives given the observations (posterior)

Forward Backward Algorithm

For a state sequence $y_1, y_2, \dots y_n$ there is:

- a path through the graph that has sequence of states START, $\langle 1, y_1 \rangle, \dots, \langle n, y_n \rangle$, STOP
- path associated with state sequence $y = y_1 \dots y_n$ has score: $p(x,y;\theta)$
- $\alpha_u(j)$ is sum of scores of all paths from START to state $\langle j, u \rangle$
- $\beta_u(j)$ is sum of scores of all paths from state $\langle j, u \rangle$ to STOP
- Given input sequence $x_1, \dots, x_n$ for $u \in 1, \dots, N$, $j \in 1, \dots, n$, the forward and backward probability can be calculated recursively.

Forward probability

$\alpha_u(j) = p(x_1, \dots x_{j-1}, y_j=u)$

Base case:

$a_u(1) = a_{START,u}$ $\forall u \in$

Recursive case:

$a_u(j+1) = \sum_v \alpha_v(j) a_{v,u} b_v(x_j)$ $\forall u \in 1, \dots, N-1, j = 1, \dots, n-1$

Backward probability

$\beta_u(j) = p(x_j, \dots, x_n|y_j=u;\theta)$

Base case:

$\beta_u(n) = a_{u,stop} b_u(x_n)$ $\forall u \in 1, \dots, N-1$

Recursive case:

$\beta_u(j) = \sum_v a_{u,v} b_u(x_j) \beta_v(j+1)$ $\forall u \in 1, \dots, N-1, j = n-1, \dots, 1$

Transition Parameters

$Count(u;v) = \sum_{i=1}^{m} Count^{(i)}(u;v)$, $Count(u) = \sum_{i=1}^{m} Count^{(i)}(u)$

$Count^{(i)}(u;v) = \sum_{j=0}^{n} P(y_j=u, y_{j+1}=v|x^{(i)};\theta^{t-1})$

$= \sum_{j=0}^{n} \frac{1}{Z} \alpha_u(j) b_u(x_j^{(i)}) a_{u,v} \beta_v(j+1)$, $Z = \sum_{u'} a_{u'}(j)\beta_{u'}(j)$

$Count^{(i)}(u) = \sum_{j=1}^{n} p(y_j=u|x^{(i)};\theta^{t-1})$

$= \sum_{j=1}^{n} \frac{1}{Z} \alpha_u(j)\beta_u(j)$, $Z = \sum_{u'} \alpha_{u'}(j)\beta_{u'}(j)$

Emisson Parameters

$Count(u \to o) = \sum_{i=1}^{m} Count^{(i)}(u \to o)$

$Count^{(i)}(u \to o) = \sum_{j=1}^{n} p(y_j=u, x_j^{(i)}=o|x^{(i)};\theta^{t-1})$

$= \sum_{j \text{ s.t. } x_j^{(i)}=o} \frac{1}{Z} \alpha_u(j)\beta_u(j)$, $Z = \sum_{u'} \alpha_{u'}(j)\beta_{u'}(j)$

⇒ can compute E-step of soft EM

Max-marginal Decoding

$p(y_i=u|x_1, \dots, x_n;\theta) = \frac{p(x_1, \dots, x_{i-1}, y_i, x_i \dots, x_n)}{p(x_1, \dots x_n;\theta)}$

$= \frac{\alpha_u(i)\beta_u(i)}{\sum_v \alpha_v(j)\beta_v(j)}$

$y_i^* = \arg\max_u p(y_i=u|x_1 \dots x_n;\theta) = \arg\max_u \frac{\alpha_u(i)\beta_u(i)}{\sum_v a_v(j)\beta_v(j)}$

$= \arg\max_u \alpha_u(i)\beta_u(i)$

→ Viterbi is often better

BN

Joint distribution → Product of conditional probabilities given its parents

→ if variable has no parents (eg. X_1), then probability is just $P(X_1 = x_1)$

Marginal Independence

$P(X_1=x_1, X_2=x_2) = \sum_{x_3} P(X_1=x_1)P(X_2=x_2)P(X_3=x_3|X_1=x_1, X_2=x_2)$

$= P(X_1=x_1)P(X_2=x_2)$

X_1 and X_2 are marginally independent of each other: if we don't know the value of X_3, there is nothing that ties the coin flips together

Induced dependence

However, if we know the value of X_3, X_1 and X_2 become dependent

possible for them to still be independent? (based on probability)

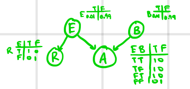
Explaining away

E: Earthquake

B: Burglary

A: Alarm

R: Report



$P(E=e, B=b, A=a, R=r) = P(E=e)P(B=b)P(A=a|E=e, B=b)P(R=r|E=e)$ [product of variable given its parents]

If we observe alarm went off - at least one of E=T or B=T should have occurred (most likely not both)

To calculate $P(B=T|A=T)$, first find $P(B=b, A=T)$

$P(B=b, A=T) = \sum_{e \in \{T,F\}} \sum_{r \in \{T,F\}} P(E=e)P(B=b)P(R=r|E=e)P(A=a|E=e, B=b)$

$= \sum_{e \in \{T,F\}} P(E=e)P(B=b)P(A=a|E=e, B=b) \underbrace{\sum_{r \in \{T,F\}} P(R=r|E=e)}_{=1}$

$= P(B=b) \sum_{e \in \{T,F\}} P(E=e)P(A=a|E=e, B=b)$

$P(B=T|A=T) = \frac{P(B=T, A=T)}{\sum_{b \in T,F} P(B=b, A=T)}$

$= \frac{0.01 \cdot [0.01 \cdot 1 + 0.99 \cdot 1]}{0.01 + [0.01 \cdot 0 + 0.99 \cdot 1]}$ = slightly above 0.5 ⇒ because of chance of both B=T and E=T.

If we now hear a report about earthquake → we now know that E=T, removing any evidence about B=T

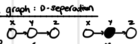
∴ $P(B=T|A=T, R=T) = P(B=T) = 0.01$

$P(E=T|A=T, R=T) = 1$

⇒ R and B are dependent given A=T (induced dependence)

Indepence from graph: D-seperation

Chain:



X and Z dependent? ✓ ✗

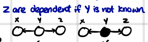
⇒ shaded means value of variable is known

$P(Z|X,Y) = \frac{P(X,Y,Z)}{P(X,Y)} = \frac{P(X)P(Y|X)P(Z|Y)}{P(X)P(Y|X)} = P(Z|Y)$

⇒ X and Z are independent given Y

⇒ X and Z are dependent if Y is not known

Common Cause:



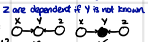
X and Z dependent? ✓ ✗

$P(Z|X,Y) = \frac{P(X,Y,Z)}{P(X,Y)} = \frac{P(Y)P(X|Y)P(Z|Y)}{P(Y)P(X|Y)} = P(Z|Y)$

⇒ X and Z are independent given Y

⇒ X and Z are dependent if Y is not known

Explaining away:



X and Z dependent? ✗ ✓

$P(X,Z) = \sum_Y P(X)P(Z)P(Y|X,Z) = P(X)P(Z)\sum_Y P(Y|X,Z) = P(X)P(Z)$

⇒ X and Z are dependent given Y

⇒ X and Z are independent without knowing Y

Joint Distribution:

$$P(X_1=x_1, \dots X_d=x_d) = \prod_{i=1}^{d} P(X_i=x_i|X_{pa_i}=x_{pa_i}) = \prod_{i=1}^{d} \theta_i(x_i|x_{pa_i})$$

⇒ $\theta_i(x_i|x_{pa_i})$ is the probability table for variable i

Number of free parameters:

$dim(G) = \sum_{i=1}^{d} (r_i - 1) \prod_{j \in pa_i} r_j$

Log-likelihood of data D for a graph G (i.e how well does graph G explain data D)

$\ell(D;\theta;G) = \sum_{t=1}^{m} \log[\prod_{i=1}^{d} \theta_i(x_i^{(t)}|x_{pa_i}^{(t)})] = \sum_{t=1}^{m}\sum_{i=1}^{d} \log\theta_i(x_i^{(t)}|x_{pa_i}^{(t)}) = \sum_{i=1}^{d}[\sum_{t=1}^{m} \log\theta_i(x_i^{(t)}|x_{pa_i}^{(t)})]$

Using values from the table $\theta_i(x_i|x_{pa_i})$

$\sum_{t=1}^{m} \log\theta_i(x_i^{(t)}|x_{pa_i}^{(t)}) = \sum_{x_i, x_{pa_i}} \underbrace{Count(x_i, x_{pa_i} \text{ in } D)}_{\text{\# of observation in Data D for which } X_i=x_i \text{ and } X_{pa_i}=x_{pa_i}} \cdot \log\theta_i(x_i|x_{pa_i})$

Therefore the corresponding maximum likelihood parameter estimate for $\theta_i(x_i|x_{pa_i})$:

$\hat{\theta}_i(x_i|x_{pa_i}) = \frac{Count((x_i, x_{pa_i}) \text{ in } D)}{Count((x_{pa_i}) \text{ in } D)}$, $x_i \in \{1, \dots r_i\}$

$Count((x_{pa_i}) \text{ in } D) \to \sum_{x_i'} Count((x_i', x_{pa_i}) \text{ in } D)$

Learning the graph structure

We cannot use only the maximum likelihood to determine the graph structure

∵ it favours graphs that have more parameters to fit the data

Therefore, we use the Bayesian Information Criterion (BIC)

$BIC(D;\hat{\theta}, G) = \ell(D;\hat{\theta}, G) - \frac{dim(G)}{2}\log(m)$

$dim(G) = \sum_{i=1}^{d}(r_i - 1)\prod_{j \in pa_i} r_j$ → # of free parameters

We can also write it as:

$Score(i|pa_i;D) = \sum_{t=1}^{m} \log\hat{\theta}_i(x_i^{(t)}|x_{pa_i}^{(t)}) - \frac{1}{2}[(r_i - 1)\prod_{j \in pa_i} r_j]\log(m)$

$BIC(D;\hat{\theta};G) = \sum_{i=1}^{d} score(i|pa_i;D)$

⇒ to maximise BIC score: local search / dynamic programming

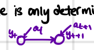
complete data → cond. prob estimates for each variable, parent selection scores for each variable → decomposable scoring func. for graphs → highest scoring acyclic graph

RL

HMM - use states y_t to encode position of the robot and observations x_t to capture sensor readings

process is markov as transition to the next state is only determined by the current state.

$T(i,j) = p(y_{t+1}=j|y_t=i)$



$T(i,k,j) = P(y_{t+1}=j|y_t=i, a_t=k)$, $i \to j$ with action k

Rewards: associated with states: $R(s_t)$

→ can also take into account action that brings robot to next state: $R(s_{t-1}, a_{t-1}, s_t)$

Utility function: long term reward for a robot when it takes actions

To prevent infinite rewards (loops):

1. Assume robot has finite horizon (after N steps utility value does not change)

$U([s_0, s_1 \dots s_N \dots s_{N+k}]) = U([s_0, s_1 \dots s_N])$ $\forall k \geq 1$

2. Discounted Rewards (Has finite value even for infinite sequences) ✓

$U([s_0, s_1 \dots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \dots = \sum_{t=0}^{\infty} \gamma^t R(s_t)$

$U([s_0, s_1 \dots]) \leq \sum_{t=0}^{\infty} \gamma^t R_{max} = \frac{R_{max}}{1-\gamma}$ } $0 \leq \gamma < 1$

→ larger γ means longer sequences

Policy π: function that maps each state to an action

goal → find optimal policy that maximizes utility → action chosen to maximize discounted rewards

-0.01	-0.01	-0.01	+1
-0.01		-0.01	-1
-0.01	-0.01	-0.01	-0.01

→	→	→	+1
↑		←	-1
↑	←	←	↓

$R(s) = -0.01$

-2	-2	-2	+1
-2		-2	-1
-2	-2	-2	-2

→	→	→	+1
↑		→	-1
↑	→	→	↑

$R(s) = -2$

Markov Decision Process → provided with:

- set of states S
- set of actions A
- transition probability function $T(s,a,s') = p(s'|s,a)$
- Reward function $R(s,a,s')$ or $R(s)$

Reinforcement Learning → provided with:

- set of states S
- set of actions A

Value Iteration - find an optimal policy using MDP.

$\pi(s)$ - particular policy that specifies action we should take at state s.

$V^\pi(s)$ - value of state s under policy π

$Q^\pi(s,a)$ - value of state s and action a under policy π

$\pi^*(s)$ - action to take in state s that gives highest utility

$V^*(s)$ - value of state s under optimal policy $\pi^*(s)$

$Q^*(s,a)$ - value of state s and action a under optimal policy π^*

value:

→ value of rewards for acting optimally from that state

These values are related by:

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

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

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

$= \sum_{s'} T(s, \pi^*(s), s')[R(s, \pi^*(s), s') + \underbrace{\gamma V^*(s')}_{\text{recursive}}]$

Value Iteration Algorithm

- Start $V_0^*(s) = 0$, for all $s \in S$
- Given V_i^*, calculate values for all states $s \in S$ (depth i+1)

$V_{i+1}^* \leftarrow \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_i^*(s')]$

- Repeat until convergence (V_{i+1}^* is almost V_i^* for all states)
- Optimal Policy

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

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

Reinforcement Learning (II)

$V^*(s) = \max_a Q^*(s,a) = Q^*(s, \pi^*(s))$ -①

$Q^*(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$ -② ①→②

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

Q-value iteration algorithm

- Start with $Q_0^*(s,a) = 0$ for all $s \in S$, $a \in A$
- Given $Q_i^*(s,a)$, calculate the Q-values for all states (depth i+1) and for all actions a

$Q_{i+1}^*(s,a) \leftarrow \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma \max_{a'} Q_i^*(s',a')]$

- Repeat until convergence
- Optimal policy

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

Reinforcement Learning - rewards and transitions are not known prior

Two options: 1. Try to learn model, transition probabilities, rewards and solve for optimal policy using the MDP algorithms

2. Learn Q-values directly

Model based Learning:

- assume we can collect information about transitions involving any state s and action a.

→ Learn T and R through experience

$T(s,a,s') = \frac{Count(s,a,s')}{Count(s,a)}$

$R(s,a,s') = \frac{\sum_t R_t(s,a,s')}{Count(s,a,s')}$ → reward observed for the t^{th} time when $s \to s'$ with a } might differ due to noisy rewards ∵ avg

Model free Learning:

$\pi^*(s) = \arg\max_a Q^*(s,a)$ → Q-values can be learned recursively (assume we know T & R)

Compute the expected value of function $f(x)$: $E[f(x)] = \sum_x p(x)f(x)$

- Model based computation - estimate $p(x)$ from samples and then compute expectation

$x_i \approx p(x)$, $i = 1, \dots, k$

$\hat{p}(x) = \frac{Count(x)}{k}$

$E[f(x)] \approx \sum_x \hat{p}(x)f(x)$

- Model-free estimation - estimate expectation directly from samples

$x_i \approx p(x)$, $i = 1, \dots, k$

$E[f(x)] \approx \frac{1}{k}\sum_{i=1}^{k} f(x_i)$

→ apply model free learning approach to the estimation of Q-values

$Q_{i+1}^*(s,a) \leftarrow \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma \max_{a'} Q_i^*(s',a')]$

→ repeatedly take one action at a time, observe reward for next state

Sample 1: $R(s,a,s_1') + \gamma \max_{a'} Q_i(s_1',a')$

⋮

Sample k: $R(s,a,s_k') + \gamma \max_{a'} Q_i(s_k',a')$

Average the samples to obtain the Q-value estimate.

$Q_{i+1}(s,a) \leftarrow \frac{1}{k}\sum_{k=1}^{k}[R(s,a,s_k') + \gamma \max_{a'} Q_i(s_k',a')]$

In practice, we observe states only when we move

Update function: (observed k-1 samples so far, now updating for k^{th} sample

$Q_{new}(s,a) \leftarrow \frac{(k-1)Q_{old}(s,a) + R(s,a,s_k') + \gamma \max_{a'} Q_{old}(s_k',a')}{k}$

$Q(s,a) \leftarrow Q(s,a) + \frac{1}{k}[R(s,a,s_k') + \gamma \max_{a'} Q(s_k',a') - Q(s,a)]$

Q Learning Algorithm

- Collect a sample: s, a, s' and $R(s,a,s')$
- Update Q values - incoporate new sample to running average over samples

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

$Q(s,a) \leftarrow Q(s,a) + \alpha[R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)]$ } learning rate $\alpha = \frac{1}{k}$

Update has same convergence conditions as the gradient ascent algorithm.

(s,a): each sample, can assign seperate learning rate for each case, $\alpha = \alpha_k(s,a)$, k is # of (s,a) so far.

$\sum_k \alpha_k(s,a) \to \infty$ and $\sum_k \alpha_k^2(s,a) < \infty$

→ $\alpha_k(s,a) = 1/k$ satisfies both conditions

HMM

Sequence labeling problem - map sequence of observations to sequence of tags
- $x_1, x_2, \ldots x_n \to y_1, y_2, \ldots y_n$
- needs to take into account previous tags

Joint probability distribution
$p(x_1,\ldots,x_n, y_0, y_1, \ldots, y_{n+1}) = \prod_{i=1}^{n+1} p(y_i|y_{i-1})\, p(x_i|y_i)$ $y_0 = START$, $y_{n+1} = STOP$

$\mathcal{T}$ is set of states including START and STOP states
O is set of observation symbol
$\theta = \langle a, b \rangle$

$a_{u,v} = p(y_{next}=v|y_{curr}=u) = \frac{count(u,v)}{count(u)}$
$b_u(o) = p(x=o|y=u) = \frac{count(u \to o)}{count(u)}$

Therefore, probability for each possible (x,y) pair is given by:
$p_\theta(x,y) = p(x,y;\theta)$
$= a_{START,y_1} \cdot a_{y_n,STOP} \cdot \prod_{i=2}^{n} a_{y_{i-1},y_i} \cdot \prod_{i=1}^{n} b_{y_i}(x_i)$

Supervised Learng

	State sequence		Observation sequence
START	(A, B, C, A, B)	STOP	(y, z, x, y, z)
START	(B, C, A, C)	STOP	(y, z, x, y)
START	(C, A, B)	STOP	(x, y, x)
START	(A, A, B)	STOP	(z, y, x)

6A
5B
4C

HMM

Clearly state what are the parameters associated with the HMM. Under the maximum likelihood estimation (MLE), what would be the optimal model parameters? Fill up the following transition and emission probability tables. Use the space below to clearly show how each **emission parameter** is estimated exactly. *(10 points)*

$a_{u,v}$ $u \backslash v$	A	B	C	STOP
START	2/4 = 0.5	1/4 = 0.25	1/4 = 0.25	0/4 = 0
A	1/6	4/6	1/6	0/6 = 0
B	0/5 = 0	0/5 = 0	2/5	3/5
C	3/4 = 0.75	0/4 = 0	0/4 = 0	1/4 = 0.25

$b_u(o)$ $u \backslash o$	x	y	z
A	1/6	4/6 = 2/3	1/6
B	2/5 = 0.4	1/5 = 0.2	2/5 = 0.4
C	2/4 = 0.5	1/4 = 0.25	1/4 = 0.25

Viterbi Algorithm - dynamic programming algorithm (Decoding, Tagg)

Input: $x = x_1 \ldots x_n$ and θ
Output: $\arg\max_{y_1 \ldots y_n} p(x_1,\ldots,x_n, y_0, y_1, \ldots, y_{n+1};\theta)$ $r(y_1,\ldots,y_k) = \prod_{i=1}^{k} a_{y_{i-1},y_i} \cdot \prod_{i=1}^{k} b_{y_i}(x_i)$
Joint probability for the first k tags: $p(x_1,\ldots,x_n,y_0,y_1,\ldots,y_{n+1}) = r(y_1,\ldots,y_n)\cdot a_{y_{n+1},y_n} = r(y_1,\ldots,y_n)\cdot a_{y_n,STOP}$

Let $S(k,v)$ be the set of tag sequences $y_1,\ldots,y_k$ such that $y_k = v$
⇒ $S(k,v)$ is set of all sequences of length k whose last tag is v

The dynamic programming algorithm will calculate the following recursively (forward)
$\pi(k,v) = \max_{(y_1 \ldots y_k) \in S(k,v)} r(y_1 \ldots y_k)$
⇒ solving the maximization problem partially over the tags $y_1 \ldots y_{k-1}$ with constraint that we use tag v for y_k

Base Case: $\pi(0,u) = \begin{cases} 1 & \text{if } u = START \\ 0 & \text{otherwise} \end{cases}$

Moving forward recursively: for any $k \in \{1,\ldots,n\}$ $\pi(k,v) = \max_{u \in \mathcal{T}} \{\pi(k-1,u)\cdot a_{u,v} \cdot b_v(x_k)\}$

Transition from y_n to STOP: $\max_{y_1 \ldots y_n} p(x_1,\ldots,x_n,y_0,y_1,\ldots,y_n,y_{n+1}) = \max_{v \in \mathcal{T}} \{\pi(n,v)\cdot a_{v,STOP}\}$

⇒ time complexity → $O(n|\mathcal{T}|^2)$, n = length, $\mathcal{T}$ = # of tags

$y_n^* = \arg\max_v \{\pi(n,v)\cdot a_{v,STOP}\}$ $y_{n-1}^* = \arg\max_u \{\pi(n-1,u)\cdot a_{u,y_n^*}\}$

Now, consider in the test phase, you are given the following new observation sequence and the following parameters, find the most probable state sequence using the Viterbi algorithm discussed in class. Clearly present the steps that lead to your final answer. *(5 points)*

	A	B	C	STOP
START	0.2	0.3	0.5	0.0
A	0.2	0.2	0.4	0.2
B	0.2	0.1	0.2	0.5
C	0.4	0.3	0.1	0.2

	x	y	z
A	0.4	0.5	0.1
B	0.4	0.1	0.5
C	0.2	0.6	0.2

State sequence	Observation sequence
(?, ?)	(y, z)

RECURSIVE CASE   BASE CASE: $\pi(0, START) = 1$
$\pi(1,A) = \pi(0,START)\cdot a_{START,A}\cdot b_A(y) = 1 \cdot 0.2 \cdot 0.5 = 0.1$
$\pi(2,A) = \max_{u\in\mathcal{T}}\{\pi(1,u)\cdot a_{u,A}\cdot b_A(z)\}$
$= \max\{(0.1)(0.2)(0.1), (0.03)(0.2)(0.1), (0.3)(0.4)(0.1)\} = 0.012$
$\pi(3,STOP) = \max_{u\in\mathcal{T}}\{\pi(2,v)\cdot a_{v,STOP}\}$
$= \max\{(0.012)(0.2), (0.045)(0.5)^{B}, (0.008)(0.2)\} = 0.0225$
Backtracking: $y_2^* = B$, $y_1^* = C$ → optimal sequence: C, B

Unsupervised Learning → given unlabelled observation sequences xs and a collection of tags but not the tag sequences ys
eg. $\mathcal{T} = \{A, B\}$, $O = \{a,b,c\}$
output sequences: [(a,b), (a,c), (a,b)]

HARD EM (similar to k-means for clustering)
E-step: each data point x is strictly assigned to a single y observation sequence → a single state sequence given model params and observation sequences: find corresponding state sequences.
1. Run Viterbi algorithm to find most probable state sequence
M-step: given observation sequences and state sequences, find model parameters
1. Parameter estimation using MLE $a_{u,v} = \frac{count(u,v)}{count(u)}$ $b_u(o) = \frac{count(u \to o)}{count(u)}$

SOFT EM:
E-step: Evaluate posterior probability over possible tag sequences
Count($x^{(i)}$, y, u→v): # of times a transition from state u to v occurs in tag sequence y given obs $x^{(i)}$
1. Reassign partial membership for each $x^{(i)}$ $Count(u,v) = \sum_i \sum_y p(y|x^{(i)})\, Count(x^{(i)}, y, u \to v)$
M-step: 1. Re-estimate transition probabilities based on expected counts $a_{u,v} = \frac{Count(u,v)}{\sum_{v'} Count(u,v')}$ where $\sum_{v'} a_{u,v'} = 1$
Problem → calculating sum over possible tag sequences in E-step has exponential hidden state sequences y.
Suppose we can efficiently calculate marginal probabilities:
$Count(u;v) = \sum_i \sum_y p(y|x^{(i)})\, Count(x^{(i)}, y, u \to v) = \sum_i \sum_j p(y_j=u, y_{j+1}=v|x^{(i)};\theta)$

Inference
How to efficiently compute a simpler marginal posterior probability
$p(y_j=u|x;\theta) = \sum_{y: y_j = u} p(y|x;\theta) = \frac{p(x, y_j=u;\theta)}{p(x;\theta)} = \frac{p(x_1,\ldots,x_n, y_j = u)}{p(x_1,\ldots,x_n)}$
Numerator can be written as: $p(x_1,x_2,\ldots,x_{j-1},y_j=u)\, p(x_j,\ldots,x_n|y_j=u)$
in HMM, given y_j, $x_j \ldots x_n$ are conditionally independent of $x_1 \ldots x_{j-1}$ → D-Separation.
Forward Probability ($\alpha_u(j)$) $\alpha_u(j) = p(x_1,\ldots,x_{j-1}, y_j = u;\theta)$
⇒ probability of emitting the symbols $x_1 \ldots x_{j-1}$ and ending in state u with position j without yet emitting the corresponding output symbol
Backward Probability ($\beta_u(j)$) $\beta_u(j) = p(x_j, x_{j+1}, \ldots, x_n|y_j = u;\theta)$
⇒ probability of emitting symbols $x_j \ldots x_n$ and final STOP state given we began in state u in position j.
$p(x;\theta) = p(x_1,\ldots,x_n;\theta) = \sum_u p(x_1,\ldots,x_{j-1}, y_j=u, x_j,\ldots,x_n;\theta)$
$= \sum_u p(x_1,\ldots,x_{j-1},y_j=u;\theta)\cdot p(x_j,\ldots,x_n|y_j=u;\theta) = \sum_u \alpha_u(j)\beta_u(j)$ (denominator)
These give us:
$p(y_j=u, y_{j+1}=v|x;\theta) = \frac{p(x_1,\ldots,x_{j-1},y_j=u)\cdot p(x_j|y_j=u)\cdot p(y_{j+1}=v|y_j=u)\cdot p(x_{j+1},\ldots,x_n|y_{j+1}=v)}{p(x;\theta)} = \frac{\alpha_u(j)\cdot a_{u,v}\cdot b_u(x_j)\cdot \beta_v(j+1)}{\sum_{u'}\alpha_{u'}(j)\beta_{u'}(j)}$
Sum over all possible tag sequences that include the transition $y_j=u$ and $y_{j+1}=v$ and generates the observations, divided by sum of all sequences that generate the observations
→ obtain relative probability of transition, relative to all the alternatives given the observations (posterior)

Forward Backward Algorithm
For a state sequence $y_1, y_2, \ldots y_n$ there is:
- a path through the graph that has sequence of states START, <1,y1>, ..., <n,yn>, STOP
- path associated with state sequence $y = y_1 \ldots y_n$ has score: $r(x,y;\theta)$
- $\alpha_u(j)$ is sum of scores of all paths from START to state <j,u>
- $\beta_u(j)$ is sum of scores of all paths from state <j,u> to STOP
- Given input sequence $x_1,\ldots,x_n$ for $u \in 1,\ldots,N$, $j \in 1,\ldots,n$, the forward and backward probability be calculated recursively

Forward probability $\alpha_u(j) = p(x_1,\ldots,x_{j-1}, y_j = u)$ Base case: $\alpha_u(1) = a_{START,u}\ \forall u \in$
Recursive case: $\alpha_u(j+1) = \sum_v \alpha_v(j)\, a_{v,u}\, b_v(x_j)\ \forall u \in 1,\ldots,N-1, j = 1,\ldots,n-1$
Backward probability $\beta_u(j) = p(x_j,\ldots,x_n|y_j=u)$ Base case: $\beta_u(n) = a_{u,STOP}\, b_u(x_n)\ \forall u \in 1,\ldots,N-1$
Recursive case: $\beta_u(j) = \sum_v a_{u,v}\, b_u(x_j)\, \beta_v(j+1)\ \forall u \in 1,\ldots,N-1, j = n-1,\ldots,1$

Transition Parameters
$Count(u;v) = \sum_i Count^{(i)}(u;v)$, $Count(u) = \sum_i Count^{(i)}(u)$
$Count^{(i)}(u;v) = \sum_{j=1}^{n} P(y_j=u, y_{j+1}=v|x^{(i)};\theta^{t-1})$
$= \sum_{j=1}^{n} \frac{1}{Z}\alpha_u(j)\, b_u(x_j^{(i)})\, a_{u,v}\, \beta_v(j+1)$, $Z = \sum_u \alpha_u(j)\beta_u(j)$
$Count^{(i)}(u) = \sum_{j=1}^{n} P(y_j=u|x^{(i)};\theta^{t-1}) = \sum_{j=1}^{n}\frac{1}{Z}\alpha_u(j)\beta_u(j)$, $Z = \sum_u \alpha_u(j)\beta_u(j)$

Emission Parameters
$Count(u \to o) = \sum_i Count^{(i)}(u \to o)$
$Count^{(i)}(u \to o) = \sum_{j=1}^{n} P(y_j = u, x_j^{(i)} = o|x^{(i)};\theta^{t-1})$
$= \sum_{j \in \{j: x_j^{(i)} = o\}} \frac{1}{Z}\alpha_u(j)\beta_u(j)$, $Z = \sum_u \alpha_u(j)\beta_u(j)$ ⇒ can compute E-step of soft EM

Max-marginal Decoding
$p(y_i=u|x_1,\ldots,x_n;\theta) = \frac{p(x_1,\ldots,x_{i-1},y_i=u,x_i,\ldots,x_n)}{p(x_1,\ldots,x_n;\theta)} = \frac{\alpha_u(i)\beta_u(i)}{\sum_v \alpha_v(i)\beta_v(i)}$ → Viterbi is often better
$y_i^* = \arg\max_u p(y_i=u|x_1,\ldots,x_n;\theta) = \arg\max_u \frac{\alpha_u(i)\beta_u(i)}{\sum_v \alpha_v(i)\beta_v(i)}$
$= \arg\max_u \alpha_u(i)\beta_u(i)$

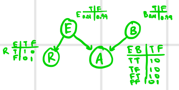
Bayesian Networks

Joint distribution → Product of conditional probabilities given its parents
→ if variable has no parents (eg. X_1), then probability is just $P(X_1 = x_1)$

Marginal Independence
$P(X_1=x_1, X_2=x_2) = \sum_{x_3} P(X_1=x_1)P(X_2=x_2)P(X_3=x_3|X_1=x_1,X_2=x_2) = P(X_1=x_1)P(X_2=x_2)$
X_1 and X_2 are marginally independent of each other: if we don't know the value of X_3, there is nothing that ties the coin flips together

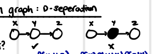
Induced dependence
However, if we know the value of X_3, X_1 and X_2 become dependent   possible for them to still be independent? (based on probability)

Explaining away
E: Earthquake
B: Burglary
A: Alarm
R: Report



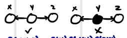
$P(E=e, B=b, A=a, R=r) = P(E=e)P(B=b)P(A=a|E=e,B=b)P(R=r|E=e)$ [product of variable given its parents]
If we observe alarm went off - at least one of E=T or B=T should have occurred (most likely not both)
To calculate $P(B=T|A=T)$, first find $P(B=b, A=T)$
$P(B=b, A=T) = \sum_{e\in\{T,F\}}\sum_{r\in\{T,F\}} P(E=e)P(B=b)P(A=a|E=e,B=b)P(R=r|E=e)$
$= \sum_{e\in\{T,F\}} P(E=e)P(B=b)P(A=a|E=e,B=b)\underbrace{\sum_{r\in\{T,F\}}P(R=r|E=e)}_{=1}$
$= P(B=b)\sum_{e} P(E=e)P(A=a|E=e,B=b)$
$P(B=T|A=T) = \frac{P(B=T, A=T)}{\sum_b P(B=b, A=T)} = \frac{0.01 \cdot (\ldots)}{0.01 \cdot (\ldots) + \ldots}$ = slightly above 0.5 ⇒ because of chance of both B=T and E=T.
If we now hear a report about earthquake → we now know that E=T, removing any evidence about B=T
∴ $P(B=T|A=T, R=T) = P(B=T) = 0.01$
$P(E=T|A=T, R=T) = 1$
⇒ R and B are dependent given A=T (induced dependence)

Independence from graph: D-separation
Chain: X and Z dependent? ✓ ✗



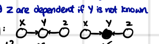
$P(Z|X,Y) = \frac{P(X,Y,Z)}{P(X,Y)} = \frac{P(X)P(Y|X)P(Z|Y)}{P(X)P(Y|X)} = P(Z|Y)$
⇒ X and Z are independent given Y
⇒ X and Z are dependent if Y is not known

Common Cause: X and Z dependent?



$P(Z|X,Y) = \frac{P(X,Y,Z)}{P(X,Y)} = \frac{P(Y)P(X|Y)P(Z|Y)}{P(Y)P(X|Y)} = P(Z|Y)$
⇒ X and Z are independent given Y
⇒ X and Z are dependent if Y is not known

Explaining away: X and Z dependent? ✗ ✓



$P(X,Z) = \sum_Y P(X)P(Z)P(Y|X,Z) = P(X)P(Z)\sum_Y P(Y|X,Z) = P(X)P(Z)$
⇒ X and Z are dependent given Y
⇒ X and Z are independent without knowing Y

Number of free parameters:
$dim(G) = \sum_i (r_i - 1)\prod_{j \in pa_i} r_j$

Joint Distribution:
$P(X_1=x_1,\ldots,X_d=x_d) = \prod_{i=1}^{d} P(X_i = x_i|X_{pa_i} = x_{pa_i}) = \prod_{i=1}^{d}\theta_i(x_i|x_{pa_i})$
⇒ $\theta_i(x_i|x_{pa_i})$ is the probability table for variable i

Log-likelihood of data D for a graph G (i.e how well does graph G explain data D)
$\ell(D;\theta,G) = \sum_{t=1}^{n}\log\left[\prod_{i=1}^{d}\theta_i(x_i^{(t)}|x_{pa_i}^{(t)})\right] = \sum_{t=1}^{n}\sum_{i=1}^{d}\log\theta_i(x_i^{(t)}|x_{pa_i}^{(t)}) = \sum_{i=1}^{d}\left[\sum_{t=1}^{n}\log\theta_i(x_i^{(t)}|x_{pa_i}^{(t)})\right]$
Using values from the table $\theta_i(x_i|x_{pa_i})$
$\sum_{i} \sum_{x_i, x_{pa_i}} count((x_i, x_{pa_i}) \text{ in } D)\cdot \log\theta_i(x_i|x_{pa_i})$ (# of occurences in table in D)

Therefore the corresponding maximum likelihood parameter estimate for $\theta_i(x_i|x_{pa_i})$:
$\hat{\theta}_i(x_i|x_{pa_i}) = \frac{count((x_i, x_{pa_i}) \text{ in } D)}{count((x_{pa_i}) \text{ in } D)}$, $x_i \in \{1,\ldots,r_i\}$ → $\sum_{x_i} count((x_i, x_{pa_i}) \text{ in } D)$

$P(X_3=2|X_4=1) = \frac{P(X_3=2, X_4=1)}{P(X_4=1)}$ = [illegible]
$= 0.92$
$= \frac{P(X_4=1|X_3=2)P(X_3=2)}{P(X_4=1|X_3=1)P(X_3=1) + P(X_3=2)P(X_4=1|X_3=2)} = \frac{(0.3)(0.5)}{(0.3)(0.1)+(0.7)(0.5)}$
[illegible] $= 0.41$

Learning the graph structure
We cannot use only the maximum likelihood to determine the graph structure
∵ it favours graphs that have more parameters to fit the data
Therefore, we use the Bayesian Information Criterion (BIC)
$BIC(D;\hat\theta,G) = \ell(D;\hat\theta,G) - \frac{dim(G)}{2}\log(n)$
$dim(G) = \sum_i (r_i - 1)\prod_{j\in pa_i} r_j$ → # of free parameters ↑ → ↓ BIC
We can also write it as:
$Score(i|pa_i;D) = \sum_t \log\hat\theta_i(x_i^{(t)}|x_{pa_i}^{(t)}) - \frac{1}{2}\left[(r_i-1)\prod_{j\in pa_i} r_j\right]\log(n)$
$BIC(D;\hat\theta,G) = \sum_i Score(i|pa_i;D)$
⇒ to maximise BIC score: local search / dynamic programming
complete data → cond. prob estimated for each variable, parent selection occurs for each variable → decomposable scoring for graphs → highest scoring acyclic graph

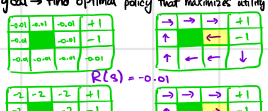
Reinforcement Learning

HMM - use states y_t to encode position of the robot and observations x_t to capture sensor readings

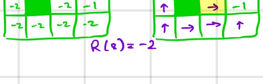
process is markov as transition to the next state is only determined by the current state
$T(i,j) = p(y_{t+1} = j|y_t = i)$
$T(i,k,j) = P(y_{t+1}=j|y_t=i, a_t=k)$, $i \to j$ with action k
Rewards: associated with states: $R(s_t)$
→ can also take into account action that brings robot to next state: $R(s_{t-1}, a_{t-1}, s_t)$

Utility function: long term reward for a robot when it takes actions
To prevent infinite rewards (loops):
1. Assume robot has finite horizon (after N steps utility value does not change)
$U([s_0,s_1,\ldots,s_{N+k}]) = U([s_0,s_1,\ldots,s_N])\ \forall k \geq 1$
2. Discounted Rewards (Has finite value even for infinite sequences) ✓
$U([s_0,s_1,\ldots]) = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + \ldots = \sum_{t=0}^{\infty}\gamma^t R(s_t)$
$U([s_0,s_1,\ldots]) \leq \sum_{t=0}^{\infty}\gamma^t R_{max} = \frac{R_{max}}{1-\gamma}$ $0 \leq \gamma < 1$
→ larger γ means longer sequences

Policy π: function that maps each state to an action
goal → find optimal policy that maximizes utility → action chosen to maximize discounted rewards



$R(s) = -0.01$



$R(s) = -2$

Markov Decision Process → provided with:
- set of states S
- set of actions A $T(s,a,s') = p(s'|s,a)$
- transition probability function
- Reward function $R(s,a,s')$ or $R(s)$

Reinforcement Learning → provided with:
- set of states S
- set of actions A

$T(s,A,s')$ $s \backslash s'$	0	1	2
0	1.0	0.0	0.0
1	0.5	0.5	0.0
2	0.5	0.0	0.5

$T(s,B,s')$ $s \backslash s'$	0	1	2
0	1.0	0.0	0.0
1	1.0	0.0	0.0
2	0.0	1.0	0.0

$0.5 \times 0^2 + 0.5 \times 1^2 + 0.0 \times 2^2 = 0.5$
$1.0 \times 0^2 + 0.0 \times 1^2 + 0.0 \times 2^2 = 0$

$R(s,a,s') = s'^2$.

Value Iteration - find an optimal policy using MDP.
$\pi(s)$ - particular policy that specifies action we should take at state s.
$V^\pi(s)$ - value of state s under policy π
$Q^\pi(s,a)$ - value of state s and action a under policy π
$\pi^*(s)$ - action to take in state s that gives highest utility
$V^*(s)$ - value of state s under optimal policy $\pi^*(s)$
$Q^*(s,a)$ - value of state s and action a under optimal policy π^*

value:
→ value of rewards for acting optimally from that state

These values are related by:
$V^*(s) = \max_a Q^*(s,a) = Q^*(s,\pi^*(s))$ (1)
$Q^*(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$ (2)
$V^*(s) = \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$ Sub (2) into (1)
$= \sum_{s'} T(s,\pi^*(s),s')[R(s,\pi^*(s),s') + \underbrace{\gamma V^*(s')}_{\text{recursive}}]$

Value Iteration Algorithm
- Start $V_0^*(s) = 0$, for all $s \in S$
- Given V_i^*, calculate values for all states $s \in S$ (depth i+1)
$V_{i+1}^* \leftarrow \max_a \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V_i^*(s')]$
- Repeat until convergence (V_{i+1}^* is almost V_i^* for all states)
- Optimal Policy
$Q^*(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$
$\pi^*(s) = \arg\max_a Q^*(s,a)$

Reinforcement Learning (II)
$V^*(s) = \max_a Q^*(s,a) = Q^*(s,\pi^*(s))$ - ①   Sub (1) into (2)
$Q^*(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma V^*(s')]$ - ②
$Q^*(s,a) = \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma \max_a Q^*(s,a)]$

Q-value iteration algorithm
- Start with $Q_0^*(s,a) = 0$ for all $s \in S, a \in A$
- Given $Q_i^*(s,a)$, calculate the Q-values for all states (depth i+1) and for all actions a
$Q_{i+1}^*(s,a) \leftarrow \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma \max_{a'} Q_i^*(s',a')]$
- Repeat until convergence
- Optimal policy $\pi^*(s) = \arg\max_{a'} Q^*(s,a)$

Reinforcement Learning - rewards and transitions unknown
2 options: 1. Try to learn model, transition probabilities, rewards and solve for optimal policy MDP algorithms   2. Learn Q-values directly

Model based Learning:
- assume we can collect information about transitions involving any state s and action a.
→ Learn T and R through experience
$T(s,a,s') = \frac{count(s,a,s')}{count(s,a)}$
$R(s,a,s') = \frac{\sum_t R_t(s,a,s')}{count(s,a,s')}$ → reward observed for the t^{th} time when $s \to s'$ with a } might differ due to noisy rewards ∵ avg

Model free Learning:
$\pi^*(s) = \arg\max_a Q^*(s,a)$ → Q-values can be learned recursively (assume we know T & R)
Compute the expected value of function $f(x)$: $E[f(x)] = \sum_x p(x)f(x)$
- Model based computation - estimate $p(x)$ from samples and then compute expectation
$x_i \approx p(x), i = 1,\ldots,k$
$\hat{p}(x) = \frac{count(x)}{k}$
$E[f(x)] \approx \sum_x \hat{p}(x)f(x)$
- Model-free estimation - estimate expectation directly from samples
$x_i \approx p(x), i = 1,\ldots,k$
$E[f(x)] \approx \frac{1}{k}\sum_{i=1}^{k} f(x_i)$

- → apply model free learning approach to the estimation of Q-values
$Q_{i+1}^*(s,a) \leftarrow \sum_{s'} T(s,a,s')[R(s,a,s') + \gamma \max_{a'} Q_i^*(s',a')]$
→ repeatedly take one action at a time, observe reward for next state
Sample 1: $R(s,a,s_1') + \gamma \max_{a'} Q_i(s_1',a')$
⋮
Sample k: $R(s,a,s_k') + \gamma \max_{a'} Q_i(s_k',a')$
Average the samples to obtain the Q-value estimate.
$Q_{i+1}(s,a) \leftarrow \frac{1}{k}\sum_{k=1}^{k}[R(s,a,s_k') + \gamma \max_{a'} Q_i(s_k',a')]$
In practice, we observe states only when we move
Update function: (observed k-1 samples so far, now updating for k^{th} sample
$Q_{new}(s,a) \leftarrow \frac{(k-1)Q_{old}(s,a) + R(s,a,s_k') + \gamma\max_{a'} Q_{old}(s_k',a')}{k}$
$Q(s,a) \leftarrow Q(s,a) + \frac{1}{k}[R(s,a,s_k') + \gamma \max_{a'} Q(s_k',a') - Q(s,a)]$

Q Learning Algorithm
- Collect a sample: s, a, s' and $R(s,a,s')$
- Update Q values - incorporate new sample to running average over samples
$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha[R(s,a,s') + \gamma \max_{a'} Q(s',a')]$
$Q(s,a) \leftarrow Q(s,a) + \alpha[R(s,a,s') + \gamma \max_{a'} Q(s',a') - Q(s,a)]$ } learning rate $\alpha = \frac{1}{k}$

Update has same convergence conditions as the gradient ascent algorithm.
(s,a): each sample, can assign seperate learning rate for each case, $\alpha = \alpha_k(s,a)$, k is # of (s,a) so far.
$\sum_k \alpha_k(s,a) \to \infty$ and $\sum_k \alpha_k^2(s,a) < \infty$
→ $\alpha_k(s,a) = 1/k$ satisfies both conditions

$a_{u,v}$ $u \backslash v$	X	Y	Z	STOP
START	0.2	0.3	0.5	0.0
X	0.2	0.2	0.4	0.2
Y	0.2	0.1	0.2	0.5
Z	0.4	0.3	0.1	0.2

$b_u(o)$ $u \backslash o$	a	b	c
X	0.4	0.5	0.1
Y	0.4	0.1	0.5
Z	0.2	0.6	0.2

State sequence	Observation sequence
(?, ?)	(**a**, **c**)

BASE CASE: $\pi(3, START) = 1$ otherwise $\pi(3,v) = 0$ if $v \neq START$.
BACKWARD RECURSIVE CASE
$\pi(2,X) = a_{START,X}\cdot b_X(c) = (0.2)(0.1) = 0.02$
$\pi(2,Y) = a_{START,Y}\cdot b_Y(c) = (0.3)(0.5) = 0.15$
$\pi(2,Z) = a_{START,Z}\cdot b_Z(c) = (0.5)(0.2) = 0.1$
$\pi(1,X) = \max_{u\in\mathcal{T}}\{\pi(2,u)\cdot a_{u,X}\cdot b_X(a)\}$
$= \max\{(0.02)(0.2)(0.4), (0.15)(0.2)(0.4), (0.1)(0.4)(0.4)\}$
$= 0.016$
$\pi(1,Y) = \max_{u\in\mathcal{T}}\{\pi(2,u)\cdot a_{u,Y}\cdot b_Y(a)\}$
$= \max\{(0.02)(0.2)(0.4), (0.15)(0.1)(0.4), (0.1)(0.3)(0.4)\}$
$= 0.012$
$\pi(1,Z) = \max_{u\in\mathcal{T}}\{\pi(2,u)\, a_{u,Z}\cdot b_Z(a)\}$
$= \max\{(0.02)(0.4)(0.2), (0.15)(0.2)(0.2), (0.1)(0.1)(0.2)\}$
$= 0.006$
$\pi(0,STOP) = \max_{v\in\mathcal{T}}\{\pi(1,v)\cdot a_{v,STOP}\}$
$= \max\{0.016\cdot 0.2, 0.012\cdot 0.5, 0.006\cdot 0.2\}$
$= 0.006$
Backtracking: $y_2^* = \arg\max_{v\in\mathcal{T}}\{\pi(2,v)\cdot a_{v,Y}\} = Z$
$y_1^* = \arg\max_{v\in\mathcal{T}}\{\pi(1,v)\cdot a_{v,STOP}\} = Y$
Optimal Sequence is Y, Z

Answer
(a) During training, simply pass the reversed state sequences and observation sequences to the implemented training procedure.
(b) During decoding, first reverse the given input sentence and pass it to implemented decoder for decoding (using the learned model parameters above), and then output the reversed output state sequence.

In the Viterbi algorithm we are interested in finding the optimal state sequence, or most probable state sequence using the following formula:

$$(s_1^*, s_2^*) = \arg\max_{s_1,s_2} P(s_1, s_2|o_1 = \mathbf{y}, o_2 = \mathbf{z}) \quad (1)$$

Sometimes we are also interested in finding the "worst" state sequence – the least probable state sequence. In other words, we are interested in:

$$(s_1^*, s_2^*) = \arg\min_{s_1,s_2} P(s_1, s_2|o_1 = \mathbf{y}, o_2 = \mathbf{z}) \quad (2)$$

This procedure is useful in some advanced structured prediction problems. Briefly and clearly explain how to perform such decoding efficiently by modifying the Viterbi algorithm. *(5 points)*

using Dynamic Programming
BASE CASE: $\pi(0,u) = \begin{cases} 1; & \text{if } u = START \\ 0; & \text{otherwise} \end{cases}$
RECURSIVE CASE: $\pi(k,v) = \min_{u\in\mathcal{T}}\{\pi(k-1,u)\cdot a_{u,v}\cdot b_v(k)\}$
TRANSITION: $\min_{v\in\mathcal{T}}\{\pi(n,v)\cdot a_{v,STOP}\}$
$(s_1^*, s_2^*) = \arg\min_{s_1,s_2} a_{START,s_1}\cdot b_{s_1}(y)\cdot a_{s_1,s_2}\cdot b_{s_2}(z)\cdot a_{s_2,STOP}$

Training
↑
Hard EM ⟷ use (decoding)
Viterbi
— Test

Soft EM ⟷ use
inference

↓

Training in Unsupervised

B4 – MLE Supervised
Learning
↓
Counting