Markov Model, HMM

Special case of Bayes Net? Yes and No. Joint distribution $P(X_0, \dots, X_T) = P(X_0) \prod_t P(X_t \mid X_{t-1})$

k-th model allow dependencies on k earlier steps.

Forward Algorithm: $P(X_t) = \sum_{x_{t-1}} P(X_t, X_{t-1}=x_{t-1})$
$= \sum_{x_{t-1}} P(X_{t-1}=x_{t-1})\, P(X_t \mid X_{t-1}=x_{t-1})$

Stationary Distributions
$P_\infty(X) = P_{\infty+1}(X)$ → Gibbs Sampling
Web link analysis

$P(X_0, X_1, E_1, \dots, X_T, E_T) = P(X_0) \prod_{t=1:T} P(X_t \mid X_{t-1})\, P(E_t \mid X_t)$

$B(X_t) = P(X_t \mid e_{1:t}) \Rightarrow P(X_{t+1} \mid e_{1:t}) = \sum_{x_t} P(X_{t+1} \mid x_t)\, B(X_t)$ Elapse time

$B'(X_{t+1}) = P(X_{t+1} \mid e_{1:t}) \Rightarrow P(X_{t+1} \mid e_{1:t+1}) \propto_{X_{t+1}} P(X_{t+1}, e_{t+1} \mid e_{1:t}) = P(e_{t+1} \mid X_{t+1})\, B'(X_{t+1})$ observe

Cost per time step: $O(|X|^2)$, X is the number of states Filtering (Forward)

Viterbi Algorithm (max)
For each state at time t, keep track of the (unnormalized) ***maximum probability of any path*** to it:
$m_{1:t}(x_t) = \max_{x_{1:t-1}} P(x_{1:t} \mid e_{1:t})$

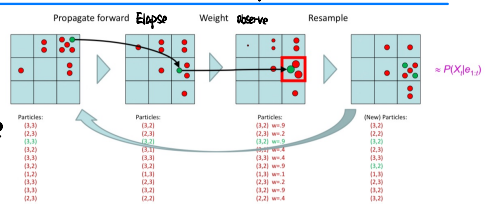
$m_{1:t+1} = \text{VITERBI}(m_{1:t}, e_{t+1})$
$= P(e_{t+1} \mid X_{t+1}) \max_{x_t} P(X_{t+1} \mid x_t)\, m_{1:t}[x_t]$

Forward Algorithm (sum)
For each state at time t, keep track of the ***total probability of all paths*** to it:
$f_{1:t}(x_t) = P(x_t \mid e_{1:t})$
$= \sum_{x_{1:t-1}} P(x_{1:t} \mid e_{1:t})$

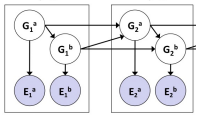
$f_{1:t+1} = \text{FORWARD}(f_{1:t}, e_{t+1})$
$= \alpha\, P(e_{t+1} \mid X_{t+1}) \sum_{x_t} P(X_{t+1} \mid x_t)\, f_{1:t}[x_t]$

Most likely explanation = most probable path

Time: $O(|X|^2 T)$ Space: $O(|X|T)$



DBN: repeat a fixed Bayes Net structure at each time
Inference: variable elimination. "unroll" the network



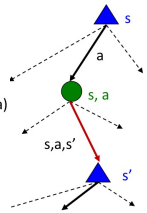
Sparse dependencies
⇒ exponentially fewer parameters

|X| is too big to exact inference. particle filtering
$w(x) = P(e|x)$. $B(X) \propto P(e|X) \cdot B'(X)$

MDPs

optimal policy: $\pi^*: S \to A$. given action from state. maximize except utility.

An MDP is defined by:
- A set of states $s \in S$
- A set of actions $a \in A$
- A transition function T(s, a, s')
  - Probability that a from s leads to s', i.e., P(s' | s, a)
  - Also called the model or the dynamics
- A reward function R(s, a, s')
  - Sometimes just R(s) or R(s')
- A start state
- Maybe a terminal state



s is a *state* — $V^*(s)$ = expected utility starting in s and acting optimally

(s, a) is a *q-state* — $Q^*(s,a)$ = expected utility starting out having taken action a from state s and (thereafter) acting optimally

(s,a,s') is a *transition* — $T(s,a,s') = P(s' \mid s,a)$, $R(s,a,s')$

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

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

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

Value Iteration $O(|S^2|A)$

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

Policy extraction

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

Policy Iteration

Step 1: Policy evaluation: calculate utilities for some fixed (not optimal) policy
Step 2: Policy improvement: update policy using one-step look-ahead with resulting converged (but not optimal!) utilities as future values

In value iteration:
- Every iteration updates both the values and (implicitly) the policy
- We don't track the policy, but taking the max over actions implicitly recomputes it

In policy iteration:
- We do several passes that update utilities with fixed policy (each pass is fast because we consider only one action, not all of them)
- After the policy is evaluated, a new policy is chosen (slow like a value iteration pass)
- May converge faster

Idea 1: Iterative updates (like value iteration) Evaluation
- Start with $V_0^\pi(s) = 0$
- Given V_k^π, calculate the depth k+1 values for all states:

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

s, π(s)

- Repeat until convergence
- Efficiency: $O(S^2)$ per iteration

Idea 2: Without the maxes, the Bellman equations are just a linear system

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

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

All use one-step lookahead expectimax fragments.

Reinforcement Learning

a MDP, don't know T or R

Model-Based: learn empirical MDP model, solve it.

Without P(A), instead collect samples $[a_1, a_2, \dots a_N]$

Unknown P(A): "Model Based"

Why does this work? Because eventually you learn the right model.

$\hat{P}(a) = \frac{\text{num}(a)}{N}$

$E[A] \approx \sum_a \hat{P}(a) \cdot a$

Unknown P(A): "Model Free"

$E[A] \approx \frac{1}{N} \sum_i a_i$

Why does this work? Because samples appear with the right frequencies.

Model-Free Passive

Direct Evaluation: Average together observed sampled value. Temporal Difference Learning

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

Update to V(s): $V^\pi(s) \leftarrow (1-\alpha) V^\pi(s) + (\alpha)\, sample$

Same update: $V^\pi(s) \leftarrow V^\pi(s) + \alpha(sample - V^\pi(s))$

Exponential moving average
- The running interpolation update: $\bar{x}_n = (1-\alpha) \cdot \bar{x}_{n-1} + \alpha \cdot x_n$

$$\bar{x}_n = \frac{x_n + (1-\alpha) \cdot x_{n-1} + (1-\alpha)^2 \cdot x_{n-2} + \dots}{1 + (1-\alpha) + (1-\alpha)^2 + \dots}$$

- Makes recent samples more important
- Forgets about the past (distant past values were wrong anyway)

Active Q-learning

$sample = R(s,a,s') + \gamma \max_{a'} Q(s',a')$

$Q(s,a) \leftarrow (1-\alpha) Q(s,a) + (\alpha)\, [sample]$

Explore Idea: select actions based on modified Q-value
- Exploration function: takes a Q-value estimate u and a visit count n, and returns an optimistic utility, e.g. $f(u,n) = u + k/n$

transition $= (s,a,r,s')$

difference $= \left[r + \gamma \max_{a'} Q(s',a')\right] - Q(s,a)$

$Q(s,a) \leftarrow Q(s,a) + \alpha\, [\text{difference}]$ Exact Q's

$w_i \leftarrow w_i + \alpha\, [\text{difference}]\, f_i(s,a)$ Approximate Q's

Feature-Based Representation

$V(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$

$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \dots + w_n f_n(s,a)$

Policy Search: start with an OK solution.
fine-tune feature weights to find better policy

Supervised Machine Learning

learn an unknown target function f

Overfit Sol: more training data (less sampling variance, training more like test)
limit the complexity of hypotheses (regularization or small hypothesis space)

$P(Y, W_1 \dots W_n) = P(Y) \prod_i P(W_i \mid Y)$

Word at position i, not i^th word in the dictionary!

Usually, each variable gets its own conditional probability distribution P(F|Y)
Here
- Each position is identically distributed
- All positions share the same conditional probabilities P(W|Y)

This is called "bag-of-words" because model is insensitive to word order or reordering

To generalize better: smooth or regularize the estimate

empirical rate $P_{ML}(x) = \frac{\text{count}(x)}{\text{total samples}}$ ⇔ Maximum Likelihood $\theta_{ML} = \arg\max_\theta P(X|\theta) = \arg\max_\theta \prod_i P_\theta(X_i)$

most likely parameter $\theta_{MAP} = \arg\max_\theta P(\theta|X) = \arg\max_\theta P(X|\theta) P(\theta)$

Laplace Smoothing

$P_{LAP}(x) = \frac{c(x)+1}{\sum_x [c(x)+1]} = \frac{c(x)+1}{N+|X|}$

k times → $P_{LAP,k}(x) = \frac{c(x)+k}{N+k|X|}$

k is strength of prior. k↑, fit ↓

$P_{LAP,k}(x|y) = \frac{c(x,y)+k}{c(y)+k|X|}$ Condition

Another option: linear interpolation
- Also get the empirical P(X) from the data
- Make sure the estimate of P(X|Y) isn't too different from the empirical P(X)

$P_{LIN}(x|y) = \alpha \hat{P}(x|y) + (1.0 - \alpha) \hat{P}(x)$

Linear Classifiers

$\text{activation}_w(x) = \sum_i w_i \cdot f_i(x) = w \cdot f(x)$ (+ bias = constant feature) adjust: $w' = w + y^* \cdot f$

Probabilistic Decision → $\phi(z) = \frac{1}{1+e^{-z}}$

Logistic regression

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

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

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

$$z_1, z_2, z_3 \rightarrow \frac{e^{z_1}}{e^{z_1}+e^{z_2}+e^{z_3}}, \frac{e^{z_2}}{e^{z_1}+e^{z_2}+e^{z_3}}, \frac{e^{z_3}}{e^{z_1}+e^{z_2}+e^{z_3}}$$

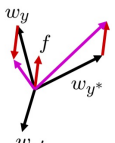
original activations → softmax activations

Multiclass

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

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

Multiclass Perceptron



Start with all weights = 0
Pick up training examples one by one
Predict with current weights

$$y = \arg\max_y \; w_y \cdot f(x)$$

If correct, no change!
If wrong: lower score of wrong answer, raise score of right answer

$$w_y = w_y - f(x)$$

$$w_{y^*} = w_{y^*} + f(x)$$

Gradient Ascent

consider: $g(w_1, w_2)$

- Updates: (learning rate)

$$w_1 \leftarrow w_1 + \alpha * \frac{\partial g}{\partial w_1}(w_1, w_2)$$

$$w_2 \leftarrow w_2 + \alpha * \frac{\partial g}{\partial w_2}(w_1, w_2)$$

- Updates in vector notation:

$$w \leftarrow w + \alpha * \nabla_w g(w)$$

with: $\nabla_w g(w) = \begin{bmatrix} \frac{\partial g}{\partial w_1}(w) \\ \frac{\partial g}{\partial w_2}(w) \end{bmatrix}$ = gradient

Batch

$$w \leftarrow w + \alpha * \sum_i \nabla \log P(y^{(i)}|x^{(i)}; w)$$

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

$$\nabla w_{y^{(i)}} f(x^{(i)}) - \nabla \log \sum_y e^{w_y f(x^{(i)})}$$

add f to weights of the correct class → Increase the score of the correct class

for y' weights: $\frac{1}{\sum_y e^{w_y f(x^{(i)})}} e^{w_{y'} f(x^{(i)})} f(x^{(i)}) = P(y'|x^{(i)}; w) f(x^{(i)})$

subtract f from y' weights in proportion to the current probability of y' → Decrease the scores of all the classes

Stochastic

Idea: once gradient on one training example has been computed, might as well update before computing next one

- init w
- for iter = 1, 2, …
  - pick random j

$$w \leftarrow w + \alpha * \nabla \log P(y^{(j)}|x^{(j)}; w)$$

Mini-Batch

Idea: gradient over a small set of training examples (=mini-batch) can be computed in parallel, might as well do that instead of a single one

- init w
- for iter = 1, 2, …
  - pick random subset of training examples J

$$w \leftarrow w + \alpha * \sum_{j \in J} \nabla \log P(y^{(j)}|x^{(j)}; w)$$

Batch GD
- Slowest
- Perfect gradient

Stochastic GD
- Fastest
- Rough-estimate grad

Mini-batch GD
- Compromise

- L2 loss function: sum of squared errors over all examples

$$L(\boldsymbol{w}) = \sum_i (y_i - h_w(\boldsymbol{x}_i))^2 = \sum_i (y_i - \boldsymbol{w}^T \boldsymbol{x}_i)^2$$

- We want the weights w* that minimize loss
- Analytical solution: at w* the derivative of loss w.r.t. each weight is zero
  - $\boldsymbol{w}^* = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$
  - **X** is the data matrix (all the data, one example per row); **y** is the vector of output values

Overfitting is also possible in regression
Regularization can be used to alleviate overfitting

LASSO (Least Absolute Shrinkage and Selection Operator)

$$L(\boldsymbol{w}) = \sum_i (y_i - \boldsymbol{w}^T \boldsymbol{x}_i)^2 + \lambda \sum_k |w_k|$$

Ridge Regression

$$L(\boldsymbol{w}) = \sum_i (y_i - \boldsymbol{w}^T \boldsymbol{x}_i)^2 + \lambda \sum_k w_k^2$$

Unsupervised

k-means — non-deterministic — update assignments or means — cannot increase ϕ

$$\phi(\{x_i\}, \{a_i\}, \{c_k\}) = \sum_i \text{dist}(x_i, c_{a_i})$$

points, assignments, means, squared Euclidean distance

Expectation Maximization

Gaussians Mixture model

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

Component; Mixing coefficient

$$\forall k : \pi_k \geqslant 0 \qquad \sum_{k=1}^{K} \pi_k = 1$$

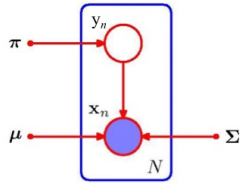
$$P(x|\mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-|x-\mu|^2}{2\sigma^2}}$$

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

P(X|Y): Each component generates data from a **multivariate Gaussian** with mean μ_i and covariance matrix Σ_i

Each data point is sampled from a ***generative process***:

1. Choose component *i* with probability π_i
2. Generate data point from $N(\mathbf{x}|\mu_i, \Sigma_i)$



[E step] Compute probability of each instance having each possible label

[M step] Treating each instance as fractionally having both labels, compute the new parameter values

EM degrades to k-means if we assume

- All the Gaussians are spherical and have identical weights and covariances
  - i.e., the only parameters are the means
- The label distributions computed at E-step are point-estimations
  - i.e., hard-assignments of data points to Gaussians
  - Alternatively, assume the variances are close to zero