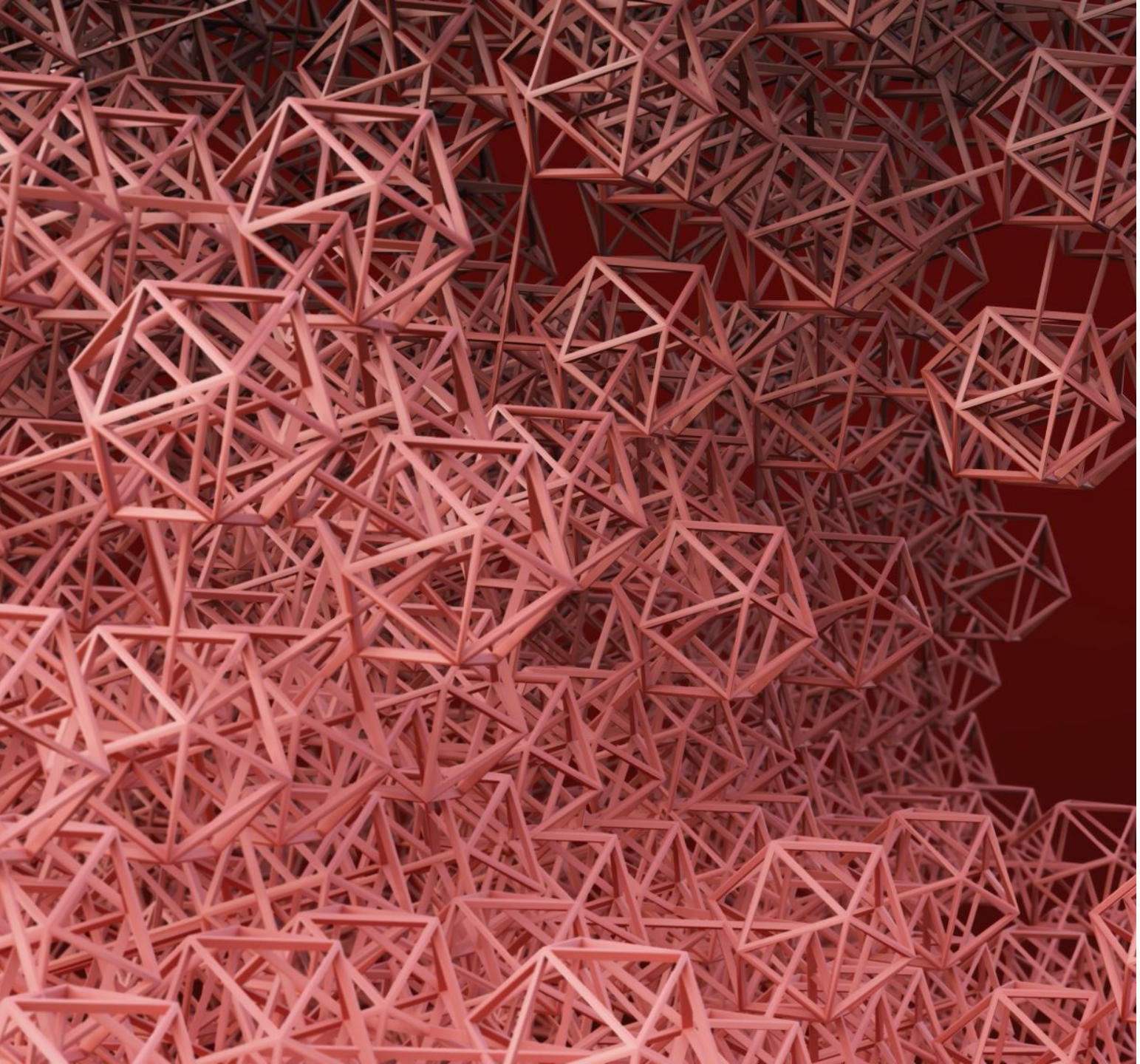
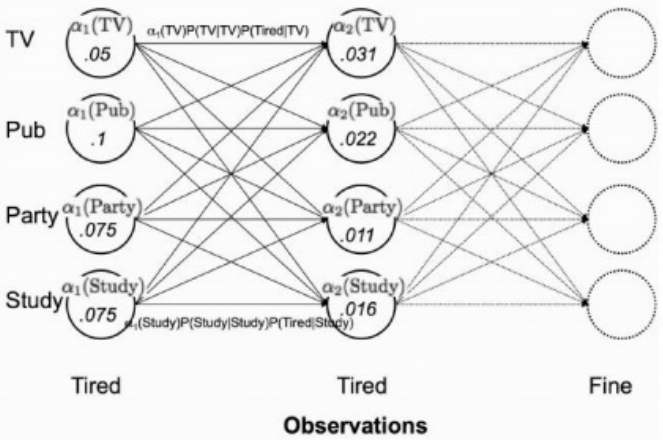


CHAPTER 16

16.3.2 THE VITERBI ALGORITHM

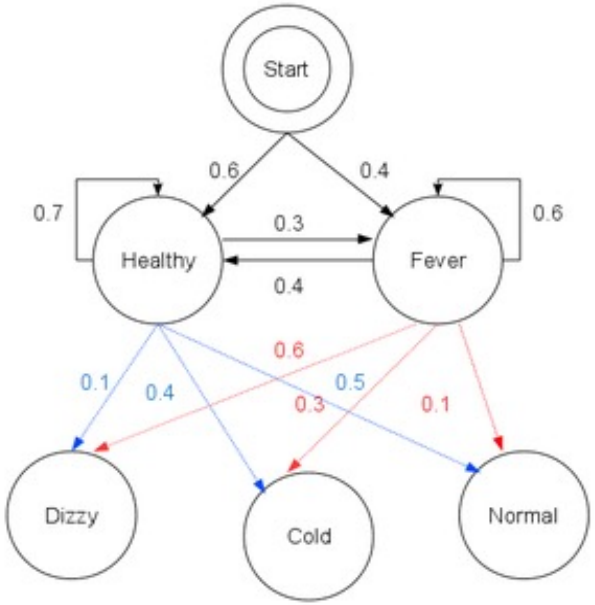
- The Viterbi algorithm is a dynamic programming algorithm for obtaining the maximum a posteriori probability estimate of the most likely sequence of hidden states—called the Viterbi path—that results in a sequence of observed events, especially in the context of Markov information sources and hidden Markov models (HMM). (2)
- The Viterbi algorithm (VA) is a recursive optimal solution to the problem of estimating the state sequence of a discrete time (1)



1. Viterbi_Forney.pdf (gold.ac.uk)
2. Viterbi algorithm – Wikipedia

THE VITERBI ALGORITHM

```
obs = ("normal", "cold", "dizzy")
states = ("Healthy", "Fever")
start_p = {"Healthy": 0.6, "Fever": 0.4}
trans_p = { "Healthy": {"Healthy": 0.7, "Fever": 0.3}, "Fever": {"Healthy": 0.4, "Fever": 0.6}, }
emit_p = { "Healthy": {"normal": 0.5, "cold": 0.4, "dizzy": 0.1}, "Fever": {"normal": 0.1, "cold": 0.
```




```
Healthy: 0.30000 0.08400 0.00588 Fever: 0.04000 0.02700 0.01512
The steps of states are Healthy Healthy Fever with highest probability of 0.01512
```

This reveals that the observations `['normal', 'cold', 'dizzy']`
were most likely generated by states `['Healthy', 'Healthy', 'Fever']`.
In other words, given the observed activities, the patient was most likely to have been healthy both on the first day when he felt normal as well as on the second day when he felt cold, and then he contracted a fever the third day.

1. Viterbi algorithm – Wikipedia

THE VITERBI ALGORITHM

The HMM Viterbi Algorithm

- Start by initialising $\delta_{i,0}$ by $\pi_i bi(o_0)$ for each state i, $\phi_0 = 0$
  - run forward in time t:
    - for each possible state s:
      - $\delta_{s,t} = \max_i(\delta_{i,t-1}a_{i,s})b_s(o_t)$
      - $\phi_{s,t} = \arg\max_i(\delta_{i,t-1}a_{i,s})$
  - set q_T^*, the most likely end hidden state to be $q_T^* = \arg\max_i \delta_{i,T}$
  - run backwards in time computing:
    - $q_{t-1}^* = \phi_{q_t^*,t}$

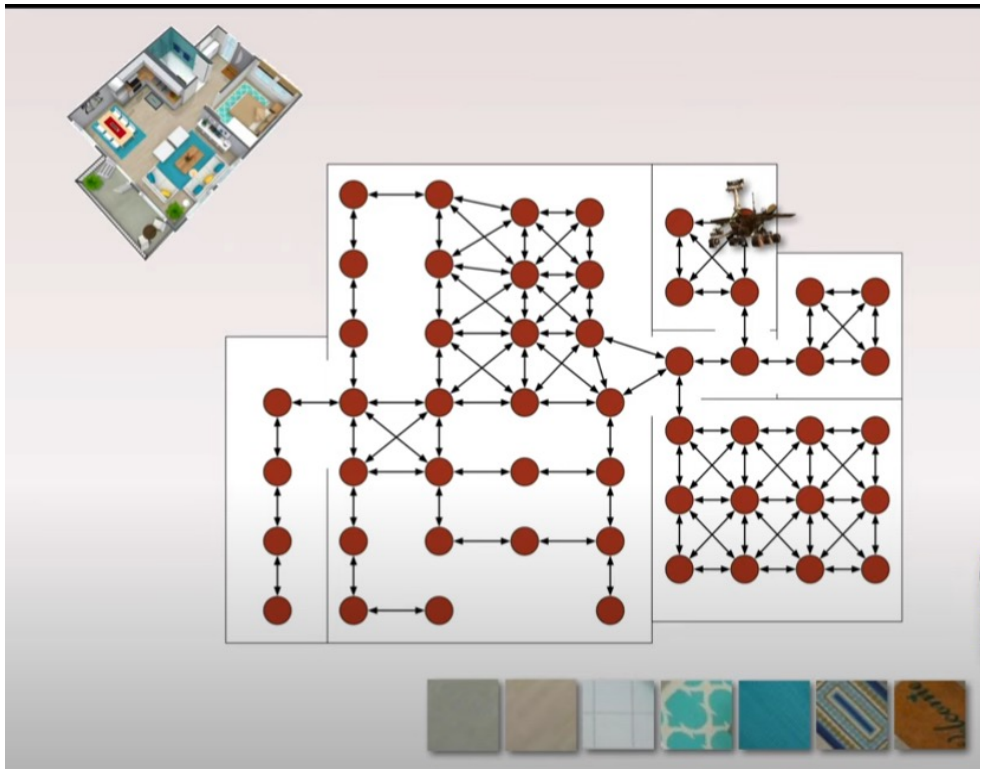
16.3.3 THE BAUM–WELCH OR FORWARD–BACKWARD ALGORITHM

- The Baum–Welch algorithm is a special case of the EM algorithm used to find the unknown parameters of a hidden Markov model (HMM). It makes use of the forward-backward algorithm to compute the statistics for the expectation step. (1)
- A hidden Markov model describes the joint probability of a collection of "hidden" and observed discrete random variables. It relies on the assumption that the i-th hidden variable given the (i □ 1)-th hidden variable is independent of previous hidden variables, and the current observation variables depend only on the current hidden state.

(1) Baum–Welch algorithm – Wikipedia

THE BAUM–WELCH OR FORWARD–BACKWARD ALGORITHM

- HMM = {N,M,A,B,π}
  - N : Number of states we can be in
  - M : Number of different observation a robot can make in given t
  - A : Two dimensions matrix contains probabilities between state.
  - B : variable store probability of an observation depending on what state you are
  - π : what is probability of seeing a state from t0 to t
- O = {O1 , O2, O3,O…}
  - observations



1. Hidden Markov Models 12: the Baum-Welch algorithm – YouTube

THE BAUM–WELCH OR FORWARD–BACKWARD ALGORITHM

- Key problem
- If $\lambda = (A,B, \pi)$
  - $P(O \mid \lambda)$ = what is probability of given ∂ seeing O
  - Path = What sequence of state explain sequence of observations. Q = (Q1,Q2 ,..)
  - Given O what is λ ?

THE BAUM–WELCH OR FORWARD–BACKWARD ALGORITHM

- $\alpha_t(i) : P(o_1,o_2,o_3,..,o_t, q_t = S_i \mid \lambda)$
  - Forward : Probability in given state S_i in time t given all observation before
  - You robot go through time, give observation and alpha says what is probabilities of end in state Si
- $\beta_t(i) : P(o_{t+1},o_{t+2},o_{t+3},..,o_t, q_t = S_i \mid \lambda)$
  - Backward : Probability of we would be in state i knowing what is coming in future
- $\gamma_t(i) : P(q_t = S_i \mid O , \lambda)$
  - Answers what is probability of time t we are goring to be in S_i, knowing everything before and after
  - using α times β and normalize it
- $\varepsilon_t(i , j) = P(q_t = S_i, q_{t+1} = s_j \mid O , \lambda)$
  - Capture the probability in time t we are in S_i and t+1 in S_j
  - Using α to find $P(S_i)$ and $\beta(S_j) + a_{ij}\, b_j(O_{t+1})$

THE BAUM–WELCH OR FORWARD–BACKWARD ALGORITHM

The HMM Baum–Welch (Forward–Backward) Algorithm

- Initialise π to be equal probabilities for all states, and a, b randomly u
  prior knowledge
- While updates have not converged:
  - **E-step:**
  - use forward and backward algorithms to get α and β
  - for each observation in the sequence $o_t, t = 1 \ldots T$
    - for each state i:
      - for each state j:
      - compute ξ using Equation (16.20)
  - **M-step:**
  - for each state i:
    - compute $\hat{\pi}_i$ using Equation (16.17)
    - for each state j:
      - compute $\hat{a}_{i,j}$ using Equation (16.18)
  - for each different possible observation o:
    - compute $\hat{b}_i(o)$ using Equation (16.19)

THE BAUM–WELCH OR FORWARD–BACKWARD ALGORITHM

- Iterative algorithm
- Find a local optimum
- Dividing α on each step over sum of α will help avoiding probabilities become zero

16.4 TRACKING METHODS

- Keeping track of where predators were and whether they were coming towards you could keep you alive
  - The Kalman Filter
    - Uses a series of measurements observed over time, containing statistical noise and other inaccuracies, and produces estimates of unknown variables by estimating a joint probability distribution over the variables for each timeframe.
  - The Particle Filter
    - Estimating the internal states in dynamical systems when partial observations are made, and random perturbations are present in the sensors as well as in the dynamical system

16.4.1 THE KALMAN FILTER

- Calculate the Kalman gain
  - Error in estimate
  - Error in data (measurement)
  - Larger weights for smaller error
- Calculate current estimate
  - Previous estimate (random value in initial time)
  - Measured value
  - Kalman gain
  - Updated estimate as out put
- Calculate new error in estimate
  - To use for next iteration

THE KALMAN FILTER

- Kalman gain
  - Error in estimate = $E\ est$
  - error in measurement = $E\ mea$
  - Kalman Gain = $KG = \frac{E\ est}{E\ est + E\ mea}$
  - $Est_t = Est_{t-1} + KG[mea - Est_{t-1}]$
  - If Kg is large : $E\ mea$ is small, we want update our estimate more
  - If Kg is small : $E\ mea$ is large, we want reduce learning rate and we are stable with less error

THE KALMAN FILTER

- Error in estimate = $E\ est$, error in measurement = $E\ mea$, Kalman Gain = KG
- $KG = \frac{E\ est}{E\ est + E\ mea}$
  - If Kg is large : $E\ mea$ is small, we want update our estimate more
  - If Kg is small : $E\ mea$ is large, we want reduce learning rate and we are stable with less error
- $est_t = est_{t-1} + KG[mea - Est_{t-1}]$
- $E_{est_t} = \frac{(E_{mea})(E_{est_{t-1}})}{(E_{mea}) + (E_{est_{t-1}})}$ = [1 – KG] $(E_{est_{t-1}})$

16.4.2 THE PARTICLE FILTER

- Relax assumption that distribution
- Keep track of the state of the probability distribution
- Prior history can be useful (No Markov assumption)

THE PARTICLE FILTER

The Particle Filter Algorithm

- Sample $\mathbf{x}_0^{(i)}$ from $p(\mathbf{x}_0$ for $i = 1 \dots N$
- For each timestep:
  - **importance sample**
  - for each datapoint:
    - sample $\hat{\mathbf{x}}_t^{(i)}$ from $q(\mathbf{x}_t^{(i)} | \mathbf{x}_{0.t-1}^{(i)}, \mathbf{y}_{1:t})$
    - add $\hat{\mathbf{x}}_t^{(i)}$ onto the list of samples to get $\mathbf{x}_{0:t}^{(i)}$ from $\mathbf{x}_{0:t-1}^{(i)}$
    - compute the importance weights:

$$w_t^{(i)} = w_{t-1}^{(i)} \frac{p(\mathbf{y}_t | \hat{\mathbf{x}}_t^{(i)}) p(\mathbf{x}_t^{(i)} | \hat{\mathbf{x}}_{t-1}^{(i)})}{q(\mathbf{x}_t^{(i)} | \mathbf{x}_{0:t-1}^{(i)}, \mathbf{y}_{1:t})} \tag{16.38}$$

  - normalise the importance weights by dividing by their sum
  - **resample the particles**
    - retain particles according to their importance weights, so that there might be several copies of some particles, and none of others to get the same number of particles approximately sampled from $p(\mathbf{x}_{0:t}^{(i)} | \mathbf{y}_{1:t})$

THE PARTICLE FILTER

- Create N samples (samples have some noise)
- Weights are equal at initial state
- Move and Observe some input like distance to a landmarks and orientation
- Update weights depending on distance to a landmarks and previous weight

- سپاس
- Dioch
- تشکر
- Thank you
- Danke