CMPT 825
Natural Language Processing

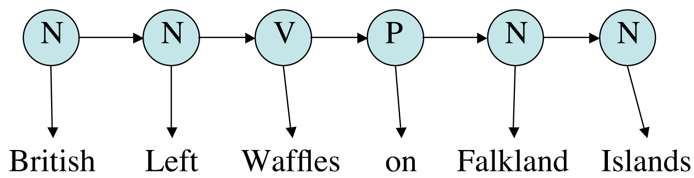
Anoop Sarkar
http://www.cs.sfu.ca/~anoop

Sequence Learning

- British Left Waffles on Falkland Islands
  - (N, N, V, P, N, N)
  - (N, V, N, P, N, N)
- Segmentation 中国十四个边境开放城市经济建设成就显著
  - (b, i, b, i, b, b, i, b, i, b, i, b, i, b, i, b, i, b, i)

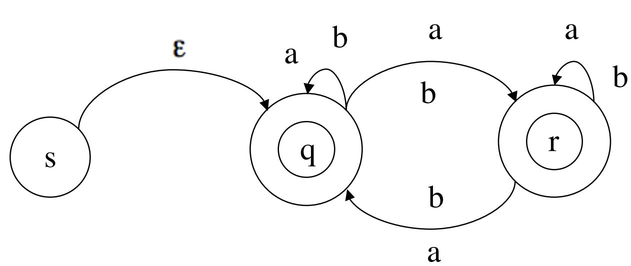
中国		十四	个	边境	开放	城市	经济	建设	成就	显著
China	's	14		open	border	cities			marked	economic achievements

Sequence Learning



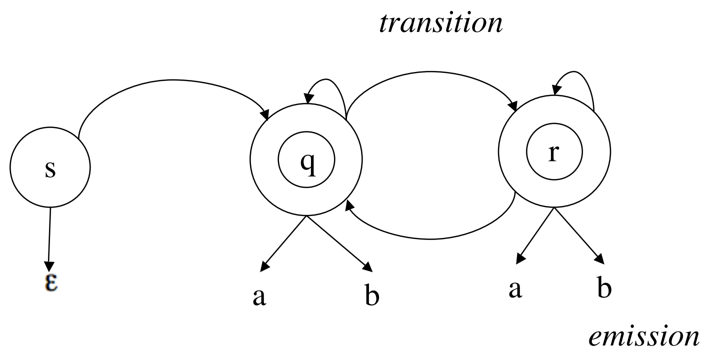
3 states: N, V, P
Observation sequence: $(o_1, \dots o_6)$
State sequence (6+1): *(Start, N, N, V, P, N, N)*

Finite State Machines



Mealy Machine

Finite State Machines

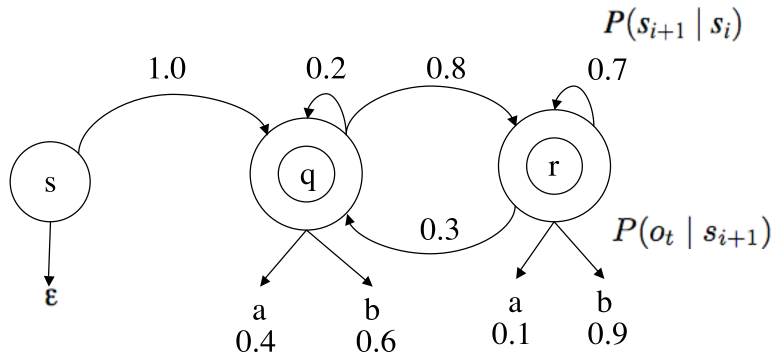


Moore Machine

Probabilistic FSMs

- Each transition is associated with a *transition probability*
- Each emission is associated with an *emission probability*
- Two conditions:
  – All outgoing transition arcs from a state must sum to 1
  – All emission arcs from a state must sum to 1

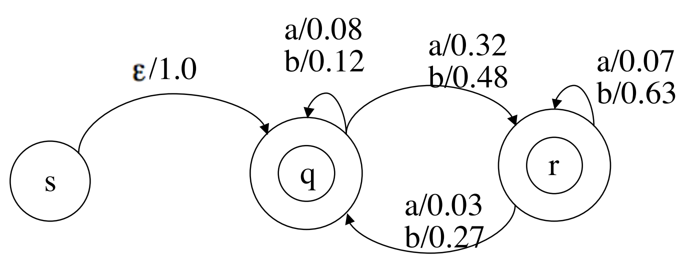
Probabilistic FSMs



$$\sum_x P(q \to x) = P(q \to r) + P(q \to q) = 1.0$$

$$\sum_x P(\mathsf{emit}(q,x)) = P(\mathsf{emit}(q,a)) + P(\mathsf{emit}(q,b)) = 1.0$$

Probabilistic FSMs



Hidden Markov Models

- There are n states $s_1, \ldots, s_i, \ldots, s_n$
- The emissions are observed (input data)
- Observation sequence $\mathbf{O}=(o_1, \ldots, o_t, \ldots, o_T)$
- The states are not directly observed (hidden)
- Data does not directly tell us which state X_t is linked with observation o_t

$$X_t \in \{s_1, \ldots, s_n\}$$

Properties of HMMs

- Markov assumption

$$P(X_t = s_i \mid \ldots, X_{t-1} = s_j)$$

- Stationary distribution

$$P(X_t = s_i \mid X_{t-1} = s_j) = P(X_{t+l} = s_i \mid X_{t+l-1} = s_j)$$

HMM Algorithms

- HMM as language model: compute probability of given observation sequence
- HMM as parser: compute the best sequence of states for a given observation sequence
- HMM as learner: given a set of observation sequences, learn its distribution, i.e. learn the transition and emission probabilities

HMM Algorithms

- HMM as language model: compute probability of given observation sequence
- Compute $P(o_1, \ldots, o_T)$ from the probability $P(X_1, \ldots, X_{T+1}, o_1, \ldots, o_T)$

$$= \prod_{t=1}^{T} P(X_{t+1} = s_j \mid X_t = s_i) \times P(o_t = k \mid X_{t+1} = s_j)$$

$$P(o_1, \ldots, o_T) = \sum_{X_1,\ldots,X_{T+1}} P(X_1, \ldots, X_{T+1}, o_1, \ldots, o_T)$$

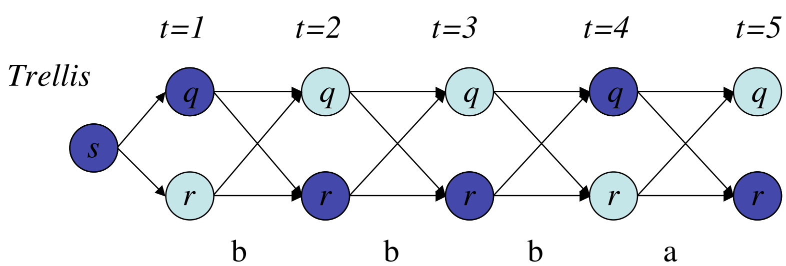
HMM Algorithms

- HMM as parser: compute the best sequence of states for a given observation sequence
- Compute best path $X_1, \ldots, X_{T+1}$ from the probability $P(X_1, \ldots, X_{T+1}, o_1, \ldots, o_T)$

Best state sequence $X^*_1, \ldots, X^*_{T+1}$

$$= \underset{X_1,\ldots,X_{T+1}}{\operatorname{argmax}} P(X_1,\ldots,X_{T+1},o_1,\ldots,o_T)$$

Best Path (Viterbi) Algorithm



- Key Idea 1: storing just the best path doesn't work
- Key Idea 2: store the best path upto *each* state

Viterbi Algorithm

```
function viterbi (edges, input, obs): returns best path
edges = transition probability
input = emission probability
T = length of obs, the observation sequence
num-states = number of states in the HMM
Create a path-matrix: viterbi[num-states+1, T+1] # init to all 0s
for each state s: viterbi[s, 0] = π[s]
for each time step t from 0 to T:
  for each state s from 0 to num-states:
    for each s' where edges[s,s'] is a transition probability:
      new-score = viterbi[s,t] * edges[s,s'] * input[s',obs[t]]
      if (viterbi[s',t+1] == 0) or (new-score > viterbi[s', t+1]):
        viterbi[s', t+1] = new-score
        back-pointer[s',t+1] = s
```


Viterbi Algorithm

```
# finding the best path
best-final-score = best-final-state = 0
for each state s from 0 to num-states:
  if (viterbi[s,T+1] > best-final-score):
    best-final-state = s
    best-final-score = viterbi[s,T+1]
# start with the last state in the sequence
x = best-final-state
state-sequence.push(x)
for t from T+1 downto 0:
  state-sequence.push(back-pointer[x,t])
  x = back-pointer[x,t]
return state-sequence
```

Forward-Backward Algorithm

- Algorithm that finds the transition and emission probabilities using training data that *does not have* hidden states provided
- Set the probabilities (for all parameters in the HMM) so that the training data T is assigned highest P(T) value (or lowest H(T), entropy value)
- This is called the maximum likelihood value over all possible hidden state sequences for the training data
- Exploits the fact that some transitions and resulting observations will occur more frequently than others in the training data

Forward-backward Algorithm

- Consider input $o_1,..., o_t,..., o_T$ where each o_t is from a set of symbols V = $\{1,..k,..K\}$
- Let π_i be the probability of state *i* being a start state (for simplicity, π_i is not discussed further)
- Let $a_{i,j}$ be the transition probability:

  $P(X_{t+1} = s_j \mid X_t = s_i)$ $|S|^2$ distinct $a_{i,j}$ values
- Let $b_{j,k}$ be the emission probability:

  $P(o_t = k \mid X_{t+1} = s_j)$ $|S|\times|V|$ distinct $b_{j,k}$ values
- Probability of going from state s_i to state s_j while observing input o_t is simply $a_{i,j} \times b_{j,k}$

Forward-backward Algorithm

- The algorithm starts with an initial setting for the probabilities in *a* and *b*
- We are provided with training data which consists of observation sequence(s): $o_1, \ldots, o_t, \ldots, o_T$
- The probability $P(o_1, \ldots, o_T)$ depends on the values in *a* and *b*
- For given observation sequence(s), different transitions/emissions will be visited with different frequencies

Forward-backward Algorithm

- For every path through the HMM, we count how many transitions occurred from state *i* to state *j* on observation o_t
- Then (loosely speaking) we reward those transitions (and emissions) which have high *expected* frequency and penalize the competing transitions
- Expected frequency means we multiply the frequency with the current probability (taken from *a* and *b*)

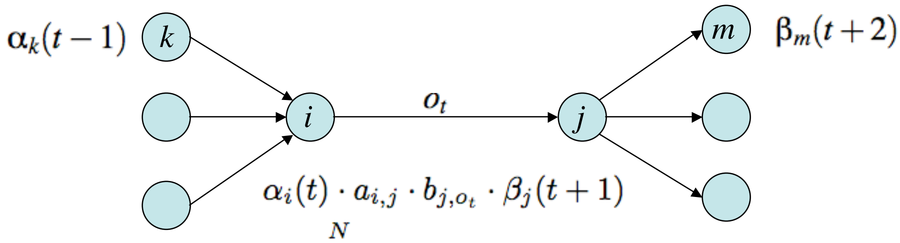
Forward-backward Algorithm

- $P(o_1,...,o_T)$ is the expected frequency of visiting all transitions and so the new frequency is the expected occurrence of a transition divided by $P(o_1,...,o_T)$
- This gives us new values for all probabilities: *a'* and *b'* and we set *a* and *b* to these new values
- Compute $P(o_1,...,o_T)$. If the value is unchanged from before iteration then stop (convergence)
- Otherwise iterate (the entire procedure) with new values for *a* and *b*

Forward-backward Algorithm

- How to compute expected frequency over all paths efficiently (*reuse dynamic programming idea from Viterbi algorithm*)
- For input $o_1,..., o_t,..., o_T$ where $o_t \in V = \{1,..k,..K\}$
- For every path from a start state to state *i* we can compute the probability of observing $o_1,..., o_{t-1}$
- Let $\alpha_i(t)$ be the sum of all these probabilities
- For every path from state *j* to a final state we can compute the probability of observing $o_{t+1},..., o_T$
- Let $\beta_j(t+1)$ be the sum of all these probabilities

Forward-Backward Algorithm

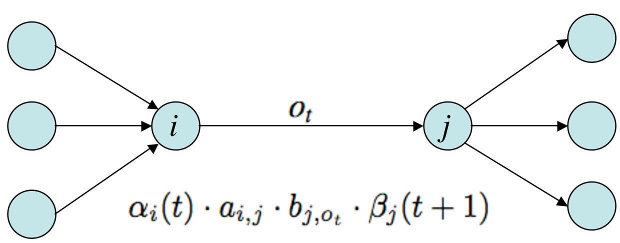


$$\alpha_i(t) = \sum_{k=1}^{N} a_{k,i} \cdot b_{i,o_{t-1}} \cdot \alpha_k(t-1)$$

$$\beta_j(t+1) = \sum_{m=1}^{N} a_{j,m} \cdot b_{m,o_{t+1}} \cdot \beta_m(t+2)$$

$$P(o_1, \ldots, o_T) = \sum_{i=1}^{N} \alpha_i(T+1) = \sum_{i=1}^{N} \pi_i \cdot \beta_i(1)$$

Forward-Backward Algorithm



$$\hat{f}(i,j,o_t) = \frac{\alpha_i(t) \cdot a_{i,j} \cdot b_{j,o_t} \cdot \beta_j(t+1)}{P(o_1, \ldots, o_T)} \qquad \hat{f}(i,j) = \sum_{t=1}^{T} \hat{f}(i,j,o_t)$$

$$a'_{i,j} = \frac{\hat{f}(i,j)}{\sum_{j=1}^{N} \hat{f}(i,j)} \qquad b'_{j,k} = \frac{\sum_{i=1}^{N} \sum_{t=1}^{T} \hat{f}(i,j,o_t = k)}{\sum_{i=1}^{N} \sum_{j=1}^{N} \hat{f}(i,j)}$$

Forward-Backward Algorithm

- Each iteration provides new values for all the *parameters*
- But are the new parameters any better? How can we tell?
- Compute probability of the training data
- For HMMs, Baum 1977 shows that the probability will always be non-decreasing (later generalized to the more general EM algorithm)
- Same as cross-entropy is non-increasing

$$KL(\mu_{i+1} \,||\, D) \leq KL(\mu_i \,||\, D)$$