Hidden Markov Model

An interesting application of deep learning is speech recognition. The traditional solution depends on the hidden Markov model (HMM). Until very recently, deep learning beats DNN+HMM by introducing the discriminative training criteria, e.g., minimizing the word error rate. These criteria was initially developed to work with HMM. So, to understand the traditional solution, and to understand discriminative learning of sequence models, we need to understand HMM.

Fifteen years ago, I read an excellect article about HMM, A Tutorial on hidden Markov models and selected applications. I haven't read any else better than it till now. This article addresses the three Problems with HMM:

1. Evaluation, or, to compute $P(O|\lambda)$.
2. Inference, or, to compute $Q = \arg\max_{Q^*} P(Q^* \mid O, \lambda)$
3. Training, or, to compute $\lambda = \arg\max_{\lambda^*} P(\lambda \mid O)$

Evaluation and the Forward Algorithm

The forward algorithm, a dynamic programming algorithm, can evaluate the likelihood of a sequence $O = \{o_1, \ldots, o_T\}$, given a model, λ. This algorithm defines and calculates

$$\alpha_t(i) = P(o_1, \ldots, o_t, q_t = i \mid \lambda)$$

where

$$\alpha_1(i) = \pi_i b_i(o_t)$$

$$\alpha_t(i) = \left[\sum_j \alpha_{t-1}(j) a_{j,i}\right] b_i(o_t)$$

$$P(O \mid \lambda) = \sum_i \alpha_T(i)$$

Inference and the Viterbi Algorithm

The Viterbi algorithm maximizes $P(Q, O \mid \lambda)$. Because

$$P(Q \mid O, \lambda) = \frac{P(Q, O \mid \lambda)}{P(O \mid \lambda)}$$

and during inference, given O, $P(O \mid \lambda)$ is a constant, the Viterbi algorithm is actually maximizing $P(Q \mid O, \lambda)$.

To simplify the representation, let us define

$$\delta_t(i) = \max_{q_1,\ldots,q_{t-1}} P(q_1, \ldots, q_t = i, o_1, \ldots, o_t \mid \lambda)$$

It is straightforward that

$$\delta_1(i) = \pi_i b_i(o_1)$$

$$\delta_t(i) = \max_j \left[\delta_{t-1}(j) a_{j,i}\right] b_i(o_t)$$

$$P(Q, O \mid \lambda) = \max_i \delta_T(i)$$

To trace the path, we define discrete function $\psi_t(i)$, the previous state that leads to the best choice of state i at time t. We denote the non-existing state before $t = 1$ by 0, so

$$\psi_1(i) = 0$$

$$\delta_t(i) = \arg\max_j \left[\delta_{t-1}(j) a_{j,i}\right] b_i(o_t)$$

Because $b_i(o_t)$ is a constant in the above equation, we can simplify it a little

$$\delta_t(i) = \arg\max_j \left[\delta_{t-1}(j) a_{j,i}\right]$$

The best state at $t = T$ is

$$q_T^* = \arg\max_i \delta_T(i)$$

$$q_{t-1}^* = \psi_t(q_t^*)$$

Training and the Baum-Welch Algorithm

The Baum-Welch algorithm is an EM algorithm. In order to compute the expectations of hidden variables that are necessary for the model update in the M-step. In order to update

$$\bar{\pi}_i = \bar{P}(q_1 = i) = \gamma_1(i)$$

we compute in the E-step

$$\gamma_t(i) = P(q_t = i \mid O, \lambda)$$

To compute $\gamma_t(i)$, we can use $\alpha_t(i) = P(o_1, \ldots, o_t, q_t = i \mid \lambda)$, but we need an additional variable

$$\beta_t(i) = P(q_t = i, o_{t+1}, \ldots, o_T \mid \lambda)$$

so that we can have

$$\alpha_t(i)\beta_t(i) = P(O, q_t = i \mid \lambda)$$

and

$$\gamma_t(i) = \frac{P(O, q_t = i \mid \lambda)}{P(O \mid \lambda)} = \frac{\alpha_t(i)\beta_t(i)}{\sum_i \alpha_t(i)\beta_t(i)}$$

We can have another dynamic programming algorithm for compute $\beta_t(i)$.

Suppose that the HMM emits discrete outputs in the range $[1, K]$,

$$\bar{b}_i(k) = \frac{\sum_{t \text{ s.t. } o_t = k} \gamma_t(i)}{\sum_t \gamma_t(i)}$$

To update $a_{i,j}$, we need

$$\xi_t(i,j) = P(q_t = i, q_{t+1} = j \mid O, \lambda)$$

$$= \frac{\alpha_t(i) a_{i,j} b_j(o_{t+1}) \beta_{t+1}(j)}{\sum_i \sum_j \alpha_t(i) a_{i,j} b_j(o_{t+1}) \beta_{t+1}(j)}$$

then we have

$$\bar{a}_{i,j} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}$$