**10-708: Probabilistic Graphical Models, Spring 2015**

# 9 : Conditional Random Fields & Case Study I: Image Segmentation

*Lecturer: Bin Zhao*

*Scribes: Emmanouil Antonios Platanios*
*Jeya Balaji Balasubramanian*
*Mariya Toneva*

## 1 Intuition

Before we present a rigorous treatment of three popular models for partially observed data - the hidden Markov model (HMM), the maximum entropy Markov model (MEMM), and the conditional random field (CRF) - we introduce some intuition behind the evolution of these models.

### 1.1 Hidden Markov Model

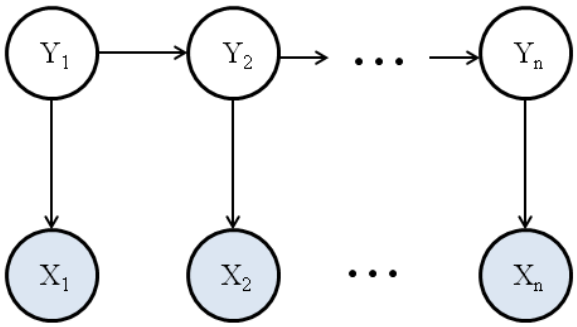


Figure 1: HMM with $n$ states and shaded nodes corresponding to the observed variables.

As introduced in Lecture 2, HMMs can be beneficial when modeling partially observed data. However, they also suffer from several important limitations.

One problem with the HMM framework is the model's strong independence assumption, known as the Markov property. The Markov property states $P(X_i|X_{i-1}, X_{i-2}, ..., X_1) = P(X_i|X_{i-1})$. While this property may simplify the model, it is not always beneficial. By restricting the conditional dependencies to only those between parents and children, the HMM allows for only local features. In many applications, such as the one discussed in section 3, it is advantageous to incorporate global features.

Another issue is that the HMM is a generative model, which computes the joint probability $P(X, Y)$. However, frequently we are mostly interested in computing $P(Y|X)$ - the probability of a state assignment given the observed data. To calculate $P(Y|X)$, the HMM spends resources computing $P(X)$ and $P(X, Y)$, even though $P(X)$ is rarely used.

To circumvent both of these problems, researchers looked to maximum entropy Markov models (MEMMs). Even though MEMMs suffer from their own set of problems, they enable the inclusion of global features and offer a discriminative framework.

## 1.2 Maximum Entropy Markov Model

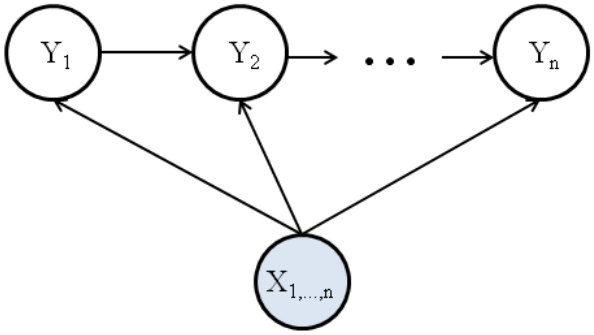


Figure 2: MEMM with $n$ states and shaded nodes corresponding to the observed variables.

The maximum entropy Markov model compensates for the HMM's locality of features by introducing a global dependence between each observed and hidden variable. With the introduction of this new dependence, MEMMs become more expressive than HMMs as they allow explicit dependence between each state and the full observation sequence.

In addition, the reversal of the arrow between the observed and hidden nodes formulates a discriminative model, which saves the resources necessary for modeling $P(X)$.

The predictive function now becomes: $P(Y_{1:n}|X_{1:n}) = \prod_{i=1}^{n} P(Y_i|Y_{i-1}, X_{1:n}) = \prod_{i=1}^{n} \frac{exp(w^T \times f(Y_i, Y_{i-1}, X_{1:n})}{Z(Y_{i-1}, X_{1:n})}$, where $w$ is some weight vector and $Z$ is a normalizaton function. While this formulation is consistent with the new objective function, it introduces a local normalization function, $Z$. This local normalization leads to a problem known as 'label bias'. The label bias results in preferences of states with lower number of children over the rest of the states.

The label biasing problem can be solved by removing the local normalization in favor of a global one. To enable global normalization, the directed edges between the hidden variables must be replaced by undirected ones.

## 1.3 Conditional Random Fields

By replacing the directed edges between the hidden nodes in MEMM by undirected ones, the model overcomes the label biasing problem while maintaining its global dependence and discriminative framework.

This new model is the conditional random field (CRF) and its predictive function can be written as $P(Y_{1:n}|X_{1:n}) = \frac{1}{Z(X_{1:n}, w)} \times \prod_{i=1}^{n} exp(w^T \times f(Y_i, Y_{i-1}, X_{1:n})$. Note that the normalizer $Z$ is now global.

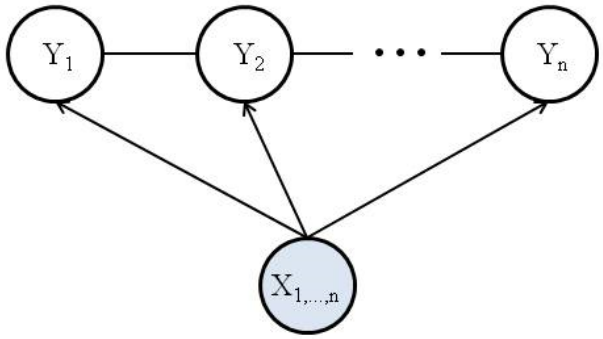


Figure 3: CRF with $n$ states and shaded nodes corresponding to the observed variables.

# 2 Decoding, Inference & Learning

## 2.1 Hidden Markov Models

The following basic elements are needed to define an HMM:

- $N$: Number of states in the model (we denote the individual states as $S = \{S_1, S_2, ..., S_N\}$ and the state at time $t$ as $q_t$).
- $M$: Number of distinct observation symbols per state (we denote the individual symbols as $V = \{v_1, v_2, ..., v_M\}$).
- $\mathrm{A} = \{a_{ij}\}_{i,j=1}^{N}$: The state transition probabilities matrix, where:

$$a_{ij} = P\left(q_{t+1} = S_j | q_t = S_i\right), \text{ for } 1 \leq i, j \leq N$$

- $\mathrm{B} = \{b_j\left(k\right)\}_{j,k=1}^{N,M}$: The observation symbol emission probabilities, where:

$$b_j\left(k\right) = P\left(v_k \text{ at } t | q_t = S_j\right), \text{ for } 1 \leq j \leq N \text{ and } 1 \leq k \leq M$$

- $\boldsymbol{\pi} = \{\pi_i\}_{i=1}^{N}$: The initial state probabilities, where:

$$\pi_i = P\left(q_1 = S_i\right), \text{ for } 1 \leq i \leq N$$

Given the appropriate values of $N$, $M$, A, B, and $\boldsymbol{\pi}$, the HMM can be used to generate an observation sequence, $\boldsymbol{O} = \{O_1, O_2, ..., O_T\}$, where each observation is one of the symbols from $V$ and $T$ is the total number of observations in the sequence. The procedure used to generate the observation sequence is the following:

1. Choose an initial state $q_1$ according to the initial state probabilities $\boldsymbol{\pi}$ and set $t = 1$.
2. Choose $O_t$ according to the emission probabilities B and the current state, $q_t$.
3. Move to a new state according to the transition probabilities A and the current state, $q_t$.
4. Set $t = t + 1$ and if $t < T$ return to step 2; otherwise terminate the procedure.

This procedure can be used to generate an observation sequence as well as to determine how a given sequence was generated from a given HMM. For convenience from now on we will use the notation $\lambda = (\mathrm{A}, \mathrm{B}, \boldsymbol{\pi})$ to indicate the complete parameter set of an HMM.

The following three formulations respectively correspond to the three basic problems: decoding, inference, and learning.

1. Given the observation sequence $\boldsymbol{O} = \{O_1, O_2, ..., O_T\}$ and the model $\lambda = (\mathrm{A}, \mathrm{B}, \boldsymbol{\pi})$, how do we efficiently compute $P(\boldsymbol{O}|\lambda)$, the probability of the observation sequence given the model?
2. Given the observation sequence $\boldsymbol{O} = \{O_1, O_2, ..., O_T\}$ and the model $\lambda = (\mathrm{A}, \mathrm{B}, \boldsymbol{\pi})$, how do we choose a corresponding state sequence $\boldsymbol{Q} = \{q_1, q_2, ..., q_T\}$ which is optimal in some meaningful sense (i.e. which best "explains" the observations)?
3. How do we adjust the model parameters, $\lambda = (\mathrm{A}, \mathrm{B}, \boldsymbol{\pi})$ so that $P(\boldsymbol{O}|\lambda)$ is maximised?

### 2.1.1 Decoding: The Forward-Backward Algorithm

To address the decoding problem, we want to calculate the probability of the observation sequence $\boldsymbol{O} = \{O_1, O_2, ..., O_T\}$ given the model $\lambda = (\mathrm{A}, \mathrm{B}, \boldsymbol{\pi})$. We could do that by enumerating every possible state sequence of length $T$, the number of observations, computing the probability of the observation sequence for each one of those state sequences and aggregating those probabilities using the probabilities of the state sequences themselves. In order to do that we initially consider one such state sequence of length $T$, $\boldsymbol{Q} = \{q_1, q_2, ..., q_T\}$, where $q_1$ is the initial state. The probability of the observation sequence $\boldsymbol{O}$ given this state sequence is given by[1]:

$$P(\boldsymbol{O}|\boldsymbol{Q}, \lambda) = \prod_{t=1}^{T} P(O_t|q_t, \lambda) = \prod_{t=1}^{T} b_{q_t}(O_t) \tag{1}$$

The probability of the state sequence $\boldsymbol{Q}$ is itself given by:

$$P(\boldsymbol{Q}|\lambda) = \pi_{q_1} a_{q_1 q_2} a_{q_2 q_3} ... a_{q_{T-1} q_T} = \pi_{q_1} \prod_{t=2}^{T} a_{q_{t-1} q_t} \tag{2}$$

Therefore, we have that:

$$P(\boldsymbol{O}, \boldsymbol{Q}|\lambda) = P(\boldsymbol{O}|\boldsymbol{Q}, \lambda) P(\boldsymbol{Q}|\lambda) \Rightarrow$$
$$P(\boldsymbol{O}, \boldsymbol{Q}|\lambda) = \left[\prod_{t=1}^{T} b_{q_t}(O_t)\right] \pi_{q_1} \left(\prod_{t=2}^{T} a_{q_{t-1} q_t}\right) \tag{3}$$

We can now obtain $P(\boldsymbol{O}|\lambda)$ by marginalising out $\boldsymbol{Q}$[2]. The result we get by doing this is the following:

$$P(\boldsymbol{O}|\lambda) = \sum_{q_1, q_2, ..., q_T} P(\boldsymbol{O}, \boldsymbol{Q}|\lambda) = \sum_{q_1, q_2, ..., q_T} \left[\prod_{t=1}^{T} b_{q_t}(O_t)\right] \pi_{q_1} \left(\prod_{t=2}^{T} a_{q_{t-1} q_t}\right) \tag{4}$$

Using this expression, the computation of $P(\boldsymbol{O}|\lambda)$ requires a total of $(2T-1) N^T + N^T - 1 \approx 2TN^T$ calculations which can be prohibitively expensive to compute even for small values of $N$ and $T$[3]. We thus need a more efficient way to compute that probability. Fortunately a more efficient method exists and it is called the forward-backward algorithm.

**The Forward-Backward Algorithm:** We consider the forward variables $\alpha_t(i)$, defined as:

$$\alpha_t(i) = P(\{O_1, O_2, ..., O_t\}, q_t = S_i|\lambda), \text{ for } 1 \leq t \leq T \text{ and } 1 \leq i \leq N \tag{5}$$

We can define $\alpha_t(i)$ inductively, so that it can be computed efficiently, in the following way:

[1] We have assumed statistical independence of the observations.
[2] This means summing $P(\boldsymbol{O}, \boldsymbol{Q}|\lambda)$ over all possible state sequences $\boldsymbol{Q}$.
[3] E.g. for $N = 5$ and $T = 100$ there are in the order of $2 \times 100 \times 5^{100} \approx 10^{72}$ calculations required.

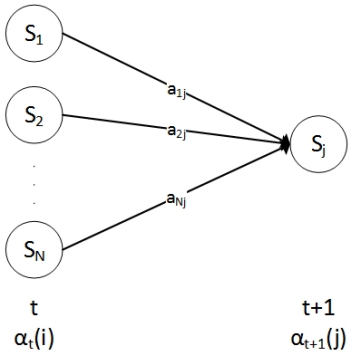


Figure 4: A diagram showing the inductive computation procedure of the forward variables.

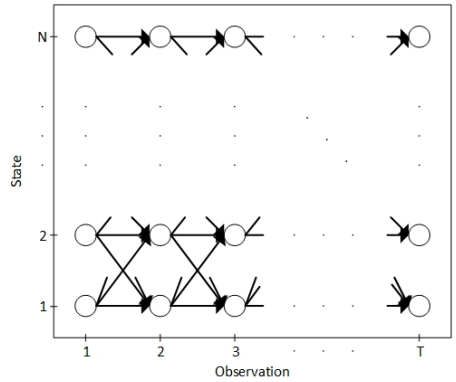


Figure 5: A diagram showing the lattice structure of the inductive computation procedure of the forward variables.

1. Initialisation:

$$\alpha_1(i) = \pi_i b_i(O_1), \text{ for } 1 \leq i \leq N \tag{6}$$

2. Induction:

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_{ij}\right] b_j(O_{t+1}), \text{ for } 1 \leq t \leq T-1 \text{ and } 1 \leq j \leq N \tag{7}$$

3. Termination:

$$P(\boldsymbol{O}|\lambda) = \sum_{i=1}^{N} \alpha_T(i) \tag{8}$$

The computation of the forward variables requires only $N(N+1)(T-1)+N+N(N-1)(T-1) \approx 2N^2T$ calculations and is thus easily computable even for large values of $N$ and $T$[4]. For solving the other two fundamental problems of HMMs we now also define a set of backward variables, $\beta_t(i)$, in a similar manner to that used for defining the forward variables, $\alpha_t(i)$:

$$\beta_t(i) = P(\{O_{t+1}, O_{t+2}, ..., O_T\} | q_t = S_i, \lambda), \text{ for } 1 \leq t \leq T \text{ and } 1 \leq i \leq N \tag{9}$$

We can also define $\beta_t(i)$ inductively, so that it can be computed efficiently, as follows:

1. Initialisation[5]:

$$\beta_T(i) = 1, \text{ for } 1 \leq i \leq N \tag{10}$$

2. Induction:

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j), \text{ for } t = T-1, T-2, ..., 1 \text{ and } 1 \leq i \leq N \tag{11}$$

[4]E.g. for $N = 5$ and $T = 100$ there are in the order of about 5000 calculations required which are much less than the about $10^{72}$ required with the straightforward calculation given in equation 4.

[5]This is an arbitrary definition that helps us derive this inductive definition for the backward variables.

The computation of the backward variables requires a similar number of calculations to the computation of the forward variables and so is easily computable even for large values of $N$ and $T$.

### 2.1.2 Inference: The Viterbi Algorithm

There are several ways of solving problem 2. We want to find the "optimal" state sequence associated with a given observation sequence but there are several possible definitions for optimality giving many possible solutions to this problem. One possible optimality criterion would be to choose the states $q_t$ which are individually most likely. This optimality criterion maximises the expected number of correct individual states. To implement the solution to problem 2 using this optimality criterion we define the following variable[6]:

$$\gamma_t(i) = P(q_t = S_i|\boldsymbol{O}, \lambda) = \frac{\alpha_t(i)\beta_t(i)}{P(\boldsymbol{O}|\lambda)} = \frac{\alpha_t(i)\beta_t(i)}{\sum_{j=1}^{N}\alpha_t(j)\beta_t(j)}, \quad (12)$$

for $1 \leq t \leq T$ and $1 \leq i \leq N$. Using the $\gamma_t(i)$ variables we can solve for the individually most likely state $q_t$ at time $t$, as:

$$q_t = \underset{i\in\{1,...,N\}}{\arg\max}\ \gamma_t(i),, \text{ for } 1 \leq t \leq T \quad (13)$$

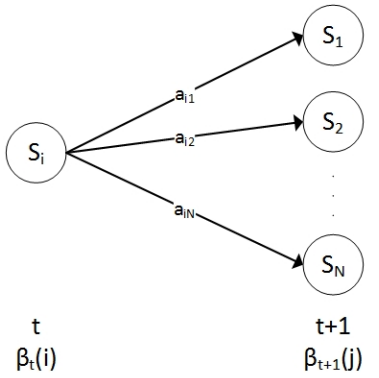


Figure 6: A diagram showing the inductive computation procedure of the backward variables.

There could however be some problems. Suppose that the HMM is not ergodic and that $a_{ij} = 0$ for some $i$ and some $j$. The above solution does not consider the fact that some state may not be accessible from some other state and could therefore result in an invalid sequence. One could solve for the state sequence that maximises the expected number of correct pairs of states $(q_t, q_{t+1})$ or even triples of states $(q_t, q_{t+1}, q_{t+2})$, etc., but there could still be issues. The most widely used criterion is to solve for the single best state sequence (i.e. maximise $P(\boldsymbol{Q}|\boldsymbol{O}, \lambda)$, which is equivalent to maximising $P(\boldsymbol{Q}, \boldsymbol{O}|\lambda)$). To do that we can use the so-called Viterbi algorithm.

**The Viterbi Algorithm:** To find the single best state sequence, $\boldsymbol{Q} = \{q_1, q_2, ..., q_T\}$, for the given observation sequence, $\boldsymbol{O} = \{O_1, O_2, ..., O_T\}$, we need to define the following set of variables:

$$\delta_t(i) = \max_{q_1,q_2,...,q_{t-1}} P(\{q_1, q_2, ..., q_{t-1}\}, q_t = S_i, \{O_1, O_2, ..., O_t\}|\lambda), \text{ for } 1 \leq t \leq T \text{ and } 1 \leq i \leq N \quad (14)$$

By induction we have that:

$$\delta_{t+1}(j) = \left[\max_{i\in\{1,...,N\}} \delta_t(i)\, a_{ij}\right] b_j(O_{t+1}), \text{ for } 1 \leq t \leq T-1 \text{ and } 1 \leq j \leq N \quad (15)$$

In order to retrieve the optimal state sequence we need to keep track of the argument which maximised $\delta_t(i)\, a_{ij}$ for each $t$ and for each $j$. We do this defining a new set of variables, the $\psi_t(j)$, which store that argument. The complete recursive procedure for finding the best state sequence can now be defined as:

1. Initialisation:

$$\delta_1(i) = \pi_i b_i(O_1), \text{ for } 1 \leq i \leq N \quad (16)$$

$$\psi_1(i) = 0, \text{ for } 1 \leq i \leq N \quad (17)$$

[6]The quantity in the denominator of the fraction in this equation is simply a normalisation factor used to make the $\gamma_t(i)$ proper probability measures.

2. Recursion:

$$\delta_t(j) = \max_{i\in\{1,\ldots,N\}} \left[\delta_{t-1}(i)\, a_{ij}\right] b_j(O_t), \text{ for } 2 \le t \le T \text{ and } 1 \le j \le N \tag{18}$$

$$\psi_t(j) = \arg\max_{i\in\{1,\ldots,N\}} \left[\delta_{t-1}(i)\, a_{ij}\right], \text{ for } 2 \le t \le T \text{ and } 1 \le j \le N \tag{19}$$

3. Termination:

$$P^* = \max_{i\in\{1,\ldots,N\}} \delta_T(i) \tag{20}$$

$$q_T^* = \arg\max_{i\in\{1,\ldots,N\}} \delta_T(i) \tag{21}$$

4. State sequence backtracking:

$$q_t^* = \psi_{t+1}\left(q_{t+1}^*\right), \text{ for } t = T-1, T-2, \ldots, 1 \tag{22}$$

### 2.1.3 Learning: The Baum-Welch Algorithm

There is no known way to analytically solve for the model which maximises the probability of the observation sequence. In fact, given any finite observation sequence as training data there is no optimal way of estimating the model parameters. We can, however, choose $\lambda = (\mathrm{A}, \mathrm{B}, \boldsymbol{\pi})$ such that $P(\boldsymbol{O}|\lambda)$ is locally maximised by using an iterative procedure such as the Baum-Welch algorithm or by using gradient techniques.

**The Baum-Welch Algorithm:** We firstly define the following set of variables[7]:

$$\xi_t(i,j) = P(q_t = S_i, q_{t+1} = S_j|\boldsymbol{O}, \lambda) = \frac{\alpha_t(i)\, a_{ij} b_j(O_{t+1})\, \beta_{t+1}(j)}{P(\boldsymbol{O}|\lambda)} \Rightarrow$$

$$\xi_t(i,j) = \frac{\alpha_t(i)\, a_{ij} b_j(O_{t+1})\, \beta_{t+1}(j)}{\sum_{k=1}^{N}\sum_{l=1}^{N} \alpha_t(k)\, a_{kl} b_l(O_{t+1})\, \beta_{t+1}(l)}, \text{ for } 1 \le t \le T \text{ and } 1 \le i, j \le N \tag{23}$$

We can now also write that:

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j), \text{ for } 1 \le t \le T \text{ and } 1 \le i \le N \tag{24}$$

and we can see that:

$$\sum_{t=1}^{T-1} \gamma_t(i) = \text{expected number of transitions from } S_i \tag{25}$$

and that:

$$\sum_{t=1}^{T-1} \xi_t(i,j) = \text{expected number of transitions from } S_i \text{ to } S_j \tag{26}$$

Using the above formulas and the concept of counting event occurrences we can give a method for the estimation of the parameters, $\lambda$, of a HMM. A set of reasonable re-estimation formulas for $\boldsymbol{\pi}$, A and B are the following:

$$\bar{\pi}_i = \text{expected number of times in } S_i \text{ at time } (t=1) = \gamma_1(i), \text{ for } 1 \le i \le N \tag{27}$$

[7]The quantity in the denominator of the fraction in this equation is simply a normalisation factor used to make the $\xi_t(i,j)$ proper probability measures.

$$\bar{a}_{ij} = \frac{\text{expected number of transitions from } S_i \text{ to } S_j}{\text{expected number of transitions from } S_i} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}, \text{ for } 1 \leq i,j \leq N \tag{28}$$

$$\bar{b}_j(k) = \frac{\text{expected number of times in } S_j \text{ and observing } v_k}{\text{expected number of times in } S_j} \Rightarrow$$

$$\bar{b}_j(k) = \frac{\sum_{t=1 \text{ s.t. } O_t = v_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(j)}, \text{ for } 1 \leq j \leq N \text{ and } 1 \leq k \leq M \tag{29}$$

It has been proven by Baum and his colleagues that either:

- The initial model $\lambda$ defines a critical point of the likelihood function, in which case $\bar{\lambda} = \left(\bar{\mathrm{A}}, \bar{\mathrm{B}}, \bar{\boldsymbol{\pi}}\right) = \lambda$, or:
- Model $\bar{\lambda}$ is more likely than model $\lambda$ in the sense that $P\left(\boldsymbol{O}|\bar{\lambda}\right) > P\left(\boldsymbol{O}|\lambda\right)$ (i.e. we have found a new model, $\bar{\lambda}$, from which the observation sequence, $\boldsymbol{O}$, is more likely to have been produced).

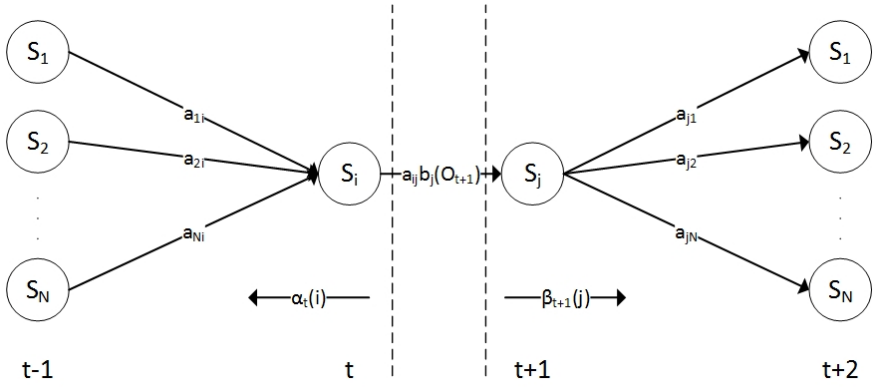


Figure 7: A diagram showing the computation procedure of the Baum-Welch algorithm variables.

The final result of this iterative re-estimation procedure is called a maximum likelihood estimate of the HMM. It should be pointed out that the forward-backward algorithm leads to local maxima only and that in most problems of interest the optimisation surface is very complex and has many local maxima. However, it has been shown that these local maxima are good enough for getting satisfactory results. The above re-estimation formulas can be derived directly by maximising (using standard constrained optimisation techniques) Baum's auxiliary function:

$$Q\left(\lambda, \bar{\lambda}\right) = \sum_{\boldsymbol{Q}} P\left(\boldsymbol{Q}|\boldsymbol{O}, \lambda\right) \log P\left(\boldsymbol{O}, \boldsymbol{Q}|\bar{\lambda}\right) \tag{30}$$

Eventually the likelihood function converges to a local maximum point. The re-estimation formulas can be readily interpreted as an implementation of the Expectation-Maximisation (EM) algorithm from the field of statistics that was also described in the GMM and the QGMM sections of this report. The expectation step consists of the calculation of the auxiliary function $Q\left(\lambda, \bar{\lambda}\right)$ and the maximisation step consists of the maximisation over $\bar{\lambda}$. The following stochastic constraints are automatically satisfied at each iteration:

$$\sum_{i=1}^{N} \bar{\pi}_i = 1 \tag{31}$$

$$\sum_{j=1}^{N} \bar{a}_{ij} = 1, \text{ for } 1 \leq i \leq N \tag{32}$$

and:

$$\sum_{k=1}^{M} \bar{b}_j(k) = 1, \text{ for } 1 \leq j \leq N \tag{33}$$

Based on setting up a standard Lagrange optimisation problem by using Lagrange multipliers it can be shown that $P(\boldsymbol{O}|\lambda)$ is maximised when the following conditions are met:

$$\pi_i = \frac{\pi_i \frac{\partial P}{\partial \pi_i}}{\sum_{k=1}^{N} \pi_k \frac{\partial P}{\partial \pi_k}}, \text{ for } 1 \leq i \leq N \tag{34}$$

$$a_{ij} = \frac{a_{ij} \frac{\partial P}{\partial a_{ij}}}{\sum_{k=1}^{N} a_{ik} \frac{\partial P}{\partial a_{ik}}}, \text{ for } 1 \leq i, j \leq N \tag{35}$$

and:

$$b_j(k) = \frac{b_j(k) \frac{\partial P}{\partial b_j(k)}}{\sum_{l=1}^{M} b_j(l) \frac{\partial P}{\partial b_j(l)}}, \text{ for } 1 \leq j \leq N \text{ and } 1 \leq k \leq M \tag{36}$$

These formulas can be easily shown to be the same as the ones we derived earlier and which are shown in equations (27), (28), and (29).

Finally we note that, since the entire problem can be set up as an optimisation problem, standard gradient techniques can be used to solve for the optimal values of the model parameters. Such techniques have been tried and have been shown to yield solutions compatible with those of the standard re-estimation procedures presented here.

## 2.2 Conditional Random Fields

The general formulation for CRFs is shown below:

$$P_\lambda(y|X) = \frac{exp(\lambda \cdot F(y, X))}{Z_\lambda(X)}, \text{ where } Z_\lambda(X) = \sum_y exp(\lambda \cdot F(Y, X)) \tag{37}$$

It can be seen that the numerator is in the exponential family, and the denominator is the normalization factor which sums over all the state sequence, $y$, over the exponential family.

### 2.2.1 Inference

The task in inference is a very similar problem as in HMMs, which is to find the most likely state sequence $y$, given observation $X$. Since the normalization term $Z_\lambda$ does not depend on the state sequence, we can just do the maximization over the term with $\lambda$ and the feature vector (which depends upon the state sequence and observation sequence).

$$\hat{y} = \arg\max_y P_\lambda(y|X) = \arg\max_y \lambda \cdot F(y, X) \tag{38}$$

This problem can be solved using a dynamic programming algorithm similar to the Viterbi algorithm for HMMs.

### 2.2.2 Learning

The learning task is to learn a model $\lambda$, given a set of $N$ training sequences $\{(X_k, y_k)\}$, where $k = 1, \ldots, N$; $X_k$ is the $k$-th observed sequence, and $y_k$ is the $k$-th hidden sequence.

We use the maximum likelihood estimation framework here. We want to find the log-likelihood. It is defined as a summation over all of the given training sequences $k$:

$$L_\lambda = \sum_y log P_\lambda(y|X) \tag{39}$$

This likelihood factorizes into two parts: the first part is the exponential family, and the second is the normalization factor:

$$L_\lambda = \sum_y [\lambda \cdot F(X_k) - log Z_\lambda(X_k)] \tag{40}$$

Next, we need a gradient to perform the optimization. We take the gradient of the log-likelihood with respect to $\lambda$:

$$\Delta L_\lambda = \frac{\partial L_\lambda}{\partial \lambda} \tag{41}$$

Taking the derivative of the gradient with respect to lambda results in the following:

$$\Delta L_\lambda = F(X_k, y_k) - \frac{1}{Z_\lambda(X_k)} \tag{42}$$

First term of the gradient is the feature we will use. The second term is the complex intractable summation over the hidden states: $Z_\lambda(X_k) = \sum_y exp(F(X_k, y_k) \cdot \lambda \cdot F(y, X))$.

We can resolve this using procedures very similar to the forward-backward procedure. The summation over the hidden state variables is the expectation of $y$ given $x_k$ of the features. The model expectations are computed below:

$$\sum_y P_\lambda(y|x) \cdot F(y|X_k) = \mathbb{E}_{P_\lambda(y|X)} \cdot F(y|X_k) \tag{43}$$

We can expand this term because we assume a chain structure for the CRFs:

$$\sum_y P_\lambda(y|X) \cdot \sum_{i=1}^{n} f(y_{i-1}, y_i, X, i) \tag{44}$$

Taking the summation to the front, we get:

$$\sum_{i=1}^{n} \left[ \sum_y P_\lambda(y|X) \cdot f(y_{i-1}, y_i, X, i) \right] \tag{45}$$

Here we consider the states of the sequence between state 1 and state $i$. If say, we want to know the probability that the state $y_{i-1}$ is $y$, and the state of $y_i$ is $y'$, given observation $X$ and the feature term, we can write this expression as:

$$\sum_{i=1}^{n} \left[ \sum_y P_\lambda(y_{i-1} = y, y_i = y'|X) \cdot f(y_{i-1}, y_i, X, i) \right] \tag{46}$$

We need to know how to compute the probability that state at $i-1$ is $y$, and at $i$ is $y'$. This was $\psi$ in the previous HMM derivation.

Analogous to HMMs, we define $\alpha$ and $\beta$ for CRFs, and also a transition matrix $M[y, y']$ at a time for the probability of a transition from state $y$ to $y'$. This quantity is just $exp[(\lambda \cdot f(y, y', X, i)]$. This $\alpha$, $\beta$, transition matrix $M[y, y']$, and the feature matrix terms give the expectation derivation.

From the definition of $\alpha$, $\beta$, we can derive this final equation:

$$\sum_{i=1}^{N} \frac{1}{Z_\lambda(X)} \alpha_{i-1}^T (M \cdot f[i] \cdot \beta_i) \tag{47}$$

Using these terms we can compute the gradient of the log-likelihood for our model.

Since we can now compute our gradient, and our goal is to maximize this using optimization techniques, we can use methods like gradient descent and conjugate gradient.

An important challenge in the learning stage is over-fitting. The optimization needs to be penalized for complexity. So, instead of just maximizing the log-likelihood, we can also do a very simple regularization, as it is done in SVM. The regularization term,$-\frac{\|\lambda\|_2^2}{\sigma^2}$, contains the outer bound of the parameters that we try to minimize. The term to optimize now becomes

$$\sum_k log P_\lambda(y|x) - \frac{\|\lambda\|_2^2}{\sigma^2} \tag{48}$$

# 3 Application of CRFs in Computer Vision

One application of CRF in computer vision is image segmentation, where the goal is to segment an image into objects. This can be reduced to a labeling problem, and in a simple case, a binary classification problem. The classification task is to differentiate the objects we are interested in (foreground) from the rest of the pixels (background). The foreground object receives a label (+), and background receives another label (-). The classification task assigns an independently learned label on each patch but does not smooth across the neighboring patches, even though they can be related.

Using CRFs, we can overcome this shortcoming with two terms. The first term (state): $f(y_i, X, \lambda)$, depends on the features of the entire image, and the label for its own patch. The state term is a logistic regression function that tries to classify whether each of the pixels/patches belongs to the foreground or background. The state term suffers from the lack of smoothness. So, we add a second term (smoothness regularization) in the CRF: $g(y_i, y_j, X, \lambda)$, which accounts for all labels from neighboring patches. This term constrains all patches that either appear spatially close to each other or share similar features and can have similar labels.

To summarize, image segmentation involves feature representation using appropriate image filters. This filtered image is subject to a classification task using a logistic regression on the state term that assigns a classification score (for each possible label) to each patch in the image. Next, smoothing is performed using an association score that considers the similarity between labels that are spatially close or are similar in appearance. These steps result in proper image segmentation instead of obtaining random patches.

An extension of such CRFs is one that addresses the issue of the size of the patch. If the patch is too large, it may include multiple objects; if it were too small, it would only include parts of an image. These errors can make optimization difficult. A possible solution using CRFs is a multi-level CRF, which contains 2 levels. The first level uses small patch sizes to learn a local appearance CRF. The second level uses large patches for global coherence CRFs. The scores from these two terms are combined to return better performing image classification tasks.