Log-Linear Models

- We have sets $\mathcal{X}$ and $\mathcal{Y}$; we will assume that $\mathcal{Y}$ is a finite set. We have a feature-vector definition $\phi : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}^d$. We also assume a parameter vector $w \in \mathbb{R}^d$. Given these definitions,

$$p(y|x; w) = \frac{\exp \left( w \cdot \phi(x, y) \right)}{\sum_{y' \in \mathcal{Y}} \exp \left( w \cdot \phi(x, y') \right)}$$

This is the conditional probability of $y$ given $x$, under parameters $w$.

The Log-Likelihood Function

- To estimate the parameters, we assume we have a set of $n$ labeled examples, $\{(x_i, y_i)\}_{i=1}^n$. The log-likelihood function is

$$L(w) = \sum_{i=1}^n \log p(y_i|x_i; w)$$

We can think of $L(w)$ as being a function that for a given $w$ measures how well $w$ explains the labeled examples. A "good" value for $w$ will give a high value for $p(y_i|x_i; w)$ for all $i = 1 \ldots n$, and thus will have a high value for $L(w)$.
Maximunmp Likelihood Estimates

- The maximum-likelihood estimates are
  \[ w^* = \arg \max_{w \in \mathbb{R}^d} \sum_{i=1}^n \log p(y_i|x_i; w) \]

The maximum-likelihood estimates are thus the parameters that best fit the training set, under the criterion \( L(w) \). (In some cases this maximum will not be well-defined—we’ll come back to this point later—but for now we’ll assume that the maximum exists.)

Finding the Maximum-Likelihood Estimates

- Given a training set \( \{(x_i, y_i)\}_{i=1}^n \), how do we find the maximum-likelihood parameter estimates \( w^* \)?
- Unfortunately, closed-form solutions do not in general exist. Instead, gradient-based optimization methods are often used. For these we need the derivative of \( L(w) \) with respect to the parameters \( w_1, w_2, \ldots, w_d \). These derivatives take the form
  \[ \frac{\partial}{\partial w_j} L(w) = \sum_i \phi_j(x_i, y_i) - \sum_i \sum_y p(y|x_i; w) \phi_j(x_i, y) \]

Regularized Log-Likelihood

- In many cases, it is useful to add a regularization term that penalizes large parameter values. The new objective function is:
  \[ L(w) = \sum_{i=1}^n \log p(y_i|x_i; w) - \frac{\lambda}{2} ||w||^2 \]
  where \( \lambda > 0 \) is a constant.
- We again choose the optimal parameter values to be \( w^* = \arg \max_{w \in \mathbb{R}^d} L(w) \)
- In this case
  \[ \frac{\partial}{\partial w_j} L(w) = \sum_i \phi_j(x_i, y_i) - \sum_i \sum_y p(y|x_i; w) \phi_j(x_i, y) - \lambda w_j \]

Maximum-Entropy Markov Models (MEMMs)

- Goal: model the distribution
  \[ p(s_1, s_2 \ldots, s_m | x_1 \ldots x_m) \]
  where each \( x_i \) for \( i = 1 \ldots m \) is a word, and each \( s_i \) for \( i = 1 \ldots m \) is an underlying state (for example, a part-of-speech tag for the \( i \)’th word). We use \( S \) to refer to the set of possible states (each \( s_i \) can take any value in \( S \)). \( S \) is a finite set.

- In HMMs (last lecture), we had
  \[ p(x_1 \ldots x_m, s_1 \ldots s_m) = t(s_1) \prod_{j=2}^m t(s_j | s_{j-1}) \prod_{j=1}^m e(x_j | s_j) \]
  where \( t(s'|s) \) are the transition parameters, and \( e(x|s) \) are the emission parameters.
Independence Assumptions in MEMMs

- MEMMs use the following decomposition:
  \[ p(s_1, s_2, \ldots, s_m | x_1, \ldots, x_m) = \prod_{i=1}^{m} p(s_i | s_{1:i-1}, x_1, \ldots, x_n) \]

- The first step is exact by the chain rule.
- The second step follows from an independence assumption, i.e., that for all \( i \),
  \[ p(s_i | s_{1:i-1}, x_1, \ldots, x_m) = p(s_i | s_{i-1}, x_1, \ldots, x_m) \]

Using Log-Linear Models

- We then model each term using a log-linear model:
  \[ p(s_i | s_{i-1}, x_1, \ldots, x_m) = \frac{\exp \left( w \cdot \phi(x_1, \ldots, x_m, i, s_{i-1}, s_i) \right)}{\sum_{s' \in S} \exp \left( w \cdot \phi(x_1, \ldots, x_m, i, s_{i-1}, s') \right)} \]

- Here \( \phi(x_1, \ldots, x_m, i, s, s') \) is a feature vector where:
  - \( x_1, \ldots, x_m \) is the sequence of \( m \) words to be tagged
  - \( i \) is the position to be tagged (any value from 1 \ldots m)
  - \( s \) is the previous state
  - \( s' \) is the new state

Decoding with MEMMs

- Goal: for a given input sequence \( x_1, \ldots, x_m \), find
  \[ \arg \max_{s_1, \ldots, s_m} p(s_1, \ldots, s_m | x_1, \ldots, x_m) \]

- We can use the Viterbi algorithm again (see last lecture on HMMs). Basic data structure:
  \[ \pi[j, s] \]

  will be a table entry that stores the maximum probability for any state sequence ending in state \( s \) at position \( j \). More formally:
  \[ \pi[j, s] = \max_{s_1, \ldots, s_{j-1}} \left( p(s_j \mid s_{j-1}, x_1, \ldots, x_m) \prod_{k=1}^{j-1} p(s_k \mid s_{k-1}, x_1, \ldots, x_m) \right) \]

The Viterbi Algorithm

- Initialization: for \( s \in S \)
  \[ \pi[1, s] = p(s \mid s_0, x_1, \ldots, x_m) \]
  where \( s_0 \) is a special “initial” state.

- For \( j = 2, \ldots, m \), \( s = 1, \ldots, k \):
  \[ \pi[j, s] = \max_{s' \in S} \left[ \pi[j-1, s'] \times p(s \mid s', x_1, \ldots, x_m) \right] \]

- We then have
  \[ \max_{s_1, \ldots, s_m} p(s_1, \ldots, s_m \mid x_1, \ldots, x_m) = \max_{s} \pi[m, s] \]

- The algorithm runs in \( O(mk^2) \) time. As before (see HMM lecture slides), we can use backpointers to recover the most likely sequence of states.
Comparison between HMMs and MEMMs

- In MEMMs, each state transition has probability
  \[ p(s_i | s_{i-1}, x_1 \ldots x_n) = \frac{\exp \left( w \cdot \phi(x_1 \ldots x_n, i, s_{i-1}, s_i) \right)}{\sum_{s' \in S} \exp \left( w \cdot \phi(x_1 \ldots x_n, i, s_{i-1}, s') \right)} \]

- In HMMs, each state transition has probability
  \[ p(s_i | s_{i-1}) \]

- The introduction of feature vectors \( \phi \) allows much richer representations in MEMMs, for example:
  - Sensitivity to any word in the input sequence \( x_1 \ldots x_n \) (not just \( x_i \))
  - Sensitivity to spelling features (prefixes, suffixes etc.) of \( x_i \), or of surrounding words

- Parameter estimation in MEMMs is more expensive than in HMMs (but is still not prohibitive for most tasks)

CRFs

- We use \( \Phi(x, s) \in \mathbb{R}^d \) to refer to a feature vector for an entire state sequence.
- We then build a giant log-linear model,
  \[ p(s | x; w) = \frac{\exp \left( w \cdot \Phi(x, s) \right)}{\sum_{s' \in S^m} \exp \left( w \cdot \Phi(x, s') \right)} \]

- The model is “giant” in the sense that: 1) the space of possible values for \( s \), i.e., \( S^m \), is huge. 2) The normalization constant (denominator in the above expression) involves a sum over a huge number of possibilities (i.e., all members of \( S^m \)).

CRFs (continued)

- Notation: for convenience we’ll use \( x \) to refer to the sequence of input words, \( x_1 \ldots x_m \), and \( s \) to refer to a sequence of possible states, \( s_1 \ldots s_m \). The set of possible states is \( S \). We use \( S^m \) to refer to the set of all possible state sequences (we have \( |S^m| = |S|^m \)).
- We’re again going to build a model of
  \[ p(s_1 \ldots s_m | x_1 \ldots x_m) = p(s | x) \]

- How do we define \( \Phi(x, s) \)? Answer:
  \[ \Phi(x, s) = \sum_{j=1}^{m} \phi(x, j, s_{j-1}, s_j) \]
  where \( \phi(x, j, s_{j-1}, s_j) \) are the same as the feature vectors used in MEMMs.
Decoding with CRFs

▶ The decoding problem: find
\[
\arg \max_{s \in S^m} p(s|x; w) = \arg \max_{s \in S^m} \frac{\exp (w \cdot \Phi(x, s))}{\sum_{s' \in S^m} \exp (w \cdot \Phi(x, s'))}
\]
\[
= \arg \max_{s \in S^m} \exp (w \cdot \Phi(x, s))
\]
\[
= \arg \max_{s \in S^m} w \cdot \Phi(x, s)
\]
\[
= \arg \max_{s \in S^m} \sum_{j=1}^{m} w \cdot \phi(x, j, s_{j-1}, s_j)
\]
\[
= \arg \max_{s \in S^m} \sum_{j=1}^{m} w \cdot \phi(x, j, s_{j-1}, s_j)
\]

▶ Again, we can use the Viterbi algorithm...

Parameter Estimation in CRFs

▶ To estimate the parameters, we assume we have a set of \(n\) labeled examples, \(\{(x^i, s^i)\}_{i=1}^n\). Each \(x^i\) is an input sequence \(x_1^i, \ldots, x_m^i\), each \(s^i\) is a state sequence \(s_1^i, \ldots, s_m^i\).

▶ We then proceed in exactly the same way as for regular log-linear models

▶ The regularized log-likelihood function is
\[
L(w) = \sum_{i=1}^{n} \log \left( \frac{p(s^i|x^i; w)}{\sum_{s' \in S^m} p(s'|x^i; w) \Phi_k(x^i, s')} \right) - \frac{\lambda}{2} ||w||^2
\]

▶ Our parameter estimates are
\[
w^* = \arg \max_{w \in \mathbb{R}^d} \sum_{i=1}^{n} \log p(s^i|x^i; w) - \frac{\lambda}{2} ||w||^2
\]

Finding the Maximum-Likelihood Estimates

▶ We’ll again use gradient-based optimization methods to find \(w^*\)

▶ How can we compute the derivatives? As before,
\[
\frac{\partial}{\partial w_k} L(w) = \sum_{i} \Phi_k(x^i, s^i) - \sum_{i} \sum_{s \in S^m} p(s|x^i; w) \Phi_k(x^i, s) - \lambda w_k
\]

▶ The first term is easily computed, because
\[
\sum_{i} \Phi_k(x^i, s^i) = \sum_{i} \sum_{j=1}^{m} \phi_k(x^i, j, s_{j-1}^i, s_j^i)
\]

▶ The second term involves a sum over \(S^m\), and because of this looks nasty...
Calculating Derivatives using the Forward-Backward Algorithm

We now consider how to compute the second term:

\[
\sum_{s \in S} p(s|x_i; w) \Phi_k(x^i, s) = \sum_{s \in S} p(s|x_i; w) \sum_{j=1}^{m} \phi_k(x^i, j, s_{j-1}, s_j)
\]

\[
= \sum_{j=1}^{m} \sum_{a \in S, b \in S} q_j^i(a, b) \phi_k(x^i, j, a, b)
\]

where

\[
q_j^i(a, b) = \sum_{s \in S : s_{j-1}=a, s_j=b} p(s|x_i; w)
\]

(for the full derivation see the notes)

For a given \(i\), all \(q_j^i\) terms can be computed simultaneously in \(O(mk^2)\) time using the forward-backward algorithm, a dynamic programming algorithm that is closely related to Viterbi.

Why prefer CRFs over MEMMs?

1. We’ll soon see in the class that it’s easy to generalize CRFs to a wide range of structured prediction problems

2. The label bias problem. An example of a conditional distribution that MEMMs can’t capture:

   \[
   \begin{align*}
   a \ b \ c & \Rightarrow \ a/A \ b/B \ c/C \quad \text{with } p(A \ B \ C|a \ b \ c) = 1 \\
   a \ b \ e & \Rightarrow \ a/A \ b/B \ e/E \quad \text{with } p(A \ D \ E|a \ b \ e) = 1
   \end{align*}
   \]

   It’s impossible to find parameters that satisfy

   \[
   p(A|a)p(B|b, A)p(C|c, B) = 1
   \]

   \[
   p(A|a)p(D|b, A)p(E|c, e) = 1
   \]

   It’s easy to find parameters in a CRF that model this distribution correctly.