CS 4650/7650, Lecture 11
Discriminative and conditional models for sequence labeling

Jacob Eisenstein

September 23, 2013

1 Review of Hidden Markov Models

Hidden Markov Models (HMMs) are a special case of a weighted (probabilistic) finite-state transducer, which transduce from tags to words. The probability model is

$$P(\boldsymbol{x}, \boldsymbol{y}) = \prod_n P(x_n|y_n;\phi)P(y_n|y_{n-1};\theta)$$

Estimation in HMMs is easy. We can use smoothed relative frequency estimation to compute the parameters as ratios of counts, e.g.

$$\phi_{\text{fish}|\text{NN}} = \frac{\text{count}(\text{fish}, \text{NN})}{\text{count}(\text{NN})}$$
$$\theta_{\text{NN}|\text{DT}} = \frac{\text{count}(\text{DT}, \text{NN})}{\text{count}(\text{DT})}$$

Decoding in HMMs is more interesting. We want to find $\hat{\boldsymbol{y}} = \arg\max_{\boldsymbol{y}} P(\boldsymbol{y}, \boldsymbol{x})$, but that involves searching $\mathcal{O}(K^N)$ tag sequences.

The **Viterbi Algorithm** is based on a recurrence relation. Let's define

$$v_n(k) = \max_{\boldsymbol{y}_{1:n-1}} P(\boldsymbol{x}_{1:n}, y_n = k, \boldsymbol{y}_{1:n-1}), \quad (1)$$

which is the joint probability of the best tag sequence ending in $y_n = k$. Then we can apply the chain rule,

$$v_n(k) = P(x_n|y_n = k) \max_{\boldsymbol{y}_{1:n-1}} P(y_n = k|y_{n-1})P(\boldsymbol{x}_{1:n-1}, \boldsymbol{y}_{1:n-1}) \tag{2}$$

Now let's say that y_{n-1} takes the value k'. Then,

$$v_n(k) = P(x_n|y_n = k) \max_{k'} P(y_n = k|y_{n-1} = k')v_{n-1}(k'). \tag{3}$$

The Viterbi algorithm directly implements this recurrence relation, in the form of a lattice. I made some nice slides about this. Check them out. In the slides, we add log-probabilities rather than multiply probabilities. This is crucial for long sequences so that the probabilities don't underflow.

The Forward Algorithm computes the marginal probability $P(\boldsymbol{x}_{1:N}) = \sum_{\boldsymbol{y}_{1:N}} P(\boldsymbol{x}_{1:N}, \boldsymbol{y}_{1:N})$. It's almost the same exact thing as the Viterbi algorithm, but replace max with addition. Let's define

$$\begin{aligned}
\alpha_n(k) &= \sum_{\boldsymbol{y}_{1:n-1}} P(\boldsymbol{x}_{1:n}, y_n = k, \boldsymbol{y}_{1:n-1}) \\
&= P(x_n|y_n = k) \sum_{\boldsymbol{y}_{1:n-1}} P(y_n = k|y_{n-1})P(\boldsymbol{x}_{1:n-1}, \boldsymbol{y}_{1:n-1}) \\
&= P(x_n|y_n = k) \sum_{k'} P(y_n = k|y_{n-1} = k')\alpha_{n-1}(k').
\end{aligned}$$

If we're in the log domain, we have to replace addition with log addition, $\oplus(a, b) = \log(e^a + e^b)$. So,

$$\alpha_n(k) = \log P(x_n|y_n = k) + \log \sum_{k'} \exp\left(\log P(y_n = k|y_{n-1} = k') + \alpha_{n-1}(k')\right) \tag{4}$$

You can use a numerical library like `scipy.misc.logsumexp` to avoid underflow in the computation of the exponent.

2 Tagging and features

The HMM can incorporate two sources of information:

- Word-tag probabilities, via $P(x_n|y_n)$
- Local context, via $P(y_n|y_{n-1})$.

Note that there are a lot of things that we're missing here:

- **Morphology**. *Slithy toves* just kind of looks like JJ NNS, because of the apparent suffixes.
- **Capitalization**. This is especially relevant for named entity recognition, e.g., *I bought an apple* and *I bought an Apple phone*.
- **Word-specific context**. In the PTB, *this* and *these* are both tagged DT. But *this* is likely to be followed by a singular noun NN, and *these* is likely to be followed by a plural noun NNS.

In addition, we're learning to maximize the joint probability $P(\boldsymbol{x}, \boldsymbol{y})$, rather than to maximize the conditional probability $P(\boldsymbol{y}|\boldsymbol{x})$, or to minimize the error (or sum of hinge losses). Today we'll talk about methods that address these limitations.

3 Structured perceptron

Remember the perceptron update:

$$\hat{y} = \arg\max_y \boldsymbol{w}^\top \boldsymbol{f}(\boldsymbol{x}, y) \tag{5}$$

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} + \boldsymbol{f}(\boldsymbol{x}, y) - \boldsymbol{f}(\boldsymbol{x}, \hat{y}) \tag{6}$$

In sequence labeling, we have a **structured output** $\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x})$. Can we still apply the perceptron rule?

$$\hat{\boldsymbol{y}} = \arg\max_{\boldsymbol{y}} \boldsymbol{w}^\top \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}) \tag{7}$$

$$\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} + \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}) - \boldsymbol{f}(\boldsymbol{x}, \hat{\boldsymbol{y}}) \tag{8}$$

This is called structured perceptron, because it learns to predict structured output $\boldsymbol{y}$. The only difference from standard perceptron is that the set of possible outputs $\mathcal{Y}(\boldsymbol{x})$ is a function of the input: in the case of sequence labeling, it is the set of tag sequences with the same length as the input.

Suppose we want to apply this to POS tagging? What features might we want? Here are three:

- Word-tag features, e.g. $\langle W : \text{slithy}, \text{JJ} \rangle$
- Adjacent tag-tag features, e.g. $\langle T : \text{JJ}, \text{NNS} \rangle$
- Suffix-tag features, e.g., $\langle M : \text{-es}, \text{NNS} \rangle$

Then we can characterize the tagging DT JJ NNS of the text the slithy toves (from Jabberwocky) in terms of the following features:

$$\begin{aligned}\boldsymbol{f}(\text{the slithy toves}, \text{DT JJ NNS}) = \{&\langle W : \text{the}, \text{DT} \rangle, \langle M : \varnothing, \text{DT} \rangle, \langle T : \star, \text{DT} \rangle \\ &\langle W : \text{slithy}, \text{JJ} \rangle, \langle M : \text{-thy}, \text{JJ} \rangle, \langle T : \text{DT}, \text{JJ} \rangle \\ &\langle W : \text{toves}, \text{NNS} \rangle, \langle M : \text{-es}, \text{NNS} \rangle, \langle T : \text{JJ}, \text{NNS} \rangle \\ &\langle T : \text{NNS}, \star \rangle\}\end{aligned}$$

If this is the correct tagging, then we hope to learn a set of weights $\boldsymbol{w}$ such that $\boldsymbol{w}^\mathsf{T} \boldsymbol{f}(\text{the slithy toves}, \text{DT JJ NNS}) > \boldsymbol{w}^\mathsf{T} \boldsymbol{f}(\text{the slithy toves}, \text{DT NN VBZ})$ (for example).

3.1 Decoding

Recall that perceptron learning requires computing $\arg\max_{\boldsymbol{y}} \boldsymbol{w}^\mathsf{T} \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y})$. The argmax is over exponentially many sequences. What can we do?

Suppose we restrict the feature function to the form,

$$\boldsymbol{f}(\boldsymbol{x}_{1:N}, \boldsymbol{y}_{1:N}) = \sum_n \boldsymbol{f}(\boldsymbol{x}_{1:N}, y_n, y_{n-1}, n) \tag{9}$$

Let's think about what this restriction says:

- We can incorporate the HMM "features" easily: the transition probabilities are covered by $\langle y_n, y_{n-1} \rangle$, and the emission probabilities are covered by $\langle y_n, x_n \rangle$. Note that the feature function needs to be parametrized by n for the emission feature.
- We can consider any part of the input $\boldsymbol{x}_{1:N}$. For example, we can have a feature that fires on $y_n = \text{JJ}$ and $x_{n-15} = \textit{barnacle}$. We can also do absolute indexing on x, for example, firing if $x_1 = \textit{slithy}$.
- We can have features that consider morphological or character-level ("word shape") features of x_n; any function that doesn't consider tags other than y_n or y_{n-1} is okay.

- What we **cannot** do is consider arbitrary parts of $\boldsymbol{y}_{1:N}$, such as $\langle y_1 = \text{CC}\rangle$ (e.g. penalizing sentences that start with conjunctions), or $\langle y_{n-3} = \text{DT}, y_n = \text{CC}\rangle$.

Given this restriction, we have

$$\boldsymbol{w}^\mathsf{T}\boldsymbol{f}(\boldsymbol{x}_{1:N}, \boldsymbol{y}_{1:N}) = \sum_n \boldsymbol{w}^\mathsf{T}\boldsymbol{f}(\boldsymbol{x}_{1:N}, y_n, y_{n-1}, n) \tag{10}$$

We can now apply the Viterbi algorithm to do **decoding**, with a very similar recurrence relation to what we saw before:

$$v_n(k) = \max_{\boldsymbol{y}_{1:n-1}} \boldsymbol{w}^\mathsf{T}\boldsymbol{f}(\boldsymbol{x}_{1:N}, \boldsymbol{y}_{1:n-1}, y_n = k) \tag{11}$$

$$= \max_{\boldsymbol{y}_{1:n}:y_n=k} \sum_m \boldsymbol{w}^\mathsf{T}\boldsymbol{f}(\boldsymbol{x}_{1:N}, y_m, y_{m-1}, m) \tag{12}$$

$$= \max_{y_{n-1}=k'} \boldsymbol{w}^\mathsf{T}\boldsymbol{f}(\boldsymbol{x}_{1:N}, y_n = k, y_{n-1} = k', n) + \sum_m^{n-1} \boldsymbol{w}^\mathsf{T}\boldsymbol{f}(\boldsymbol{x}_{1:N}, y_m = k', y_{m-1}, m) \tag{13}$$

$$= \max_{y_{n-1}=k'} \boldsymbol{w}^\mathsf{T}\boldsymbol{f}(\boldsymbol{x}_{1:N}, y_n = k, y_{n-1} = k', n) + v_{n-1}(k') \tag{14}$$

So to compute $v_n(k)$, we have to iterate over all $y_{n-1} = k'$,

- find the features $\boldsymbol{f}(\boldsymbol{x}_{1:N}, y_n = k, y_{n-1} = k', n)$,
- compute the inner product $\boldsymbol{w}^\mathsf{T}\boldsymbol{f}(\boldsymbol{x}_{1:N}, y_n = k, y_{n-1} = k', n)$,
- add it to $v_{n-1}(k')$
- take the max over all k'

This only works because of the assumption that the feature function decomposes over local parts of the sequence! If we wanted a feature that considered arbitrary parts of tag sequence, there would be no way to incorporate it into the recurrence relation.

3.2 Example

In our example, we have

$$
\begin{aligned}
v_0(\text{DT}) =& \boldsymbol{w}^\mathsf{T}\{\langle T : \star, \text{DT}\rangle, \langle W : \text{the}, \text{DT}\rangle, \langle M : \varnothing, \text{DT}\rangle\} \\
v_0(\text{JJ}) =& \boldsymbol{w}^\mathsf{T}\{\langle T : \star, \text{JJ}\rangle, \langle W : \text{the}, \text{JJ}\rangle, \langle M : \varnothing, \text{JJ}\rangle\} \\
& \ldots \\
v_1(\text{JJ}) = \max & \quad \boldsymbol{w}^\mathsf{T}\{\langle T : \text{DT}, \text{JJ}\rangle, \langle W : \text{slithy}, \text{JJ}\rangle, \langle M : \text{-thy}, \text{JJ}\rangle\} + v_0(\text{DT}), \\
& \quad \boldsymbol{w}^\mathsf{T}\{\langle T : \text{JJ}, \text{JJ}\rangle, \langle W : \text{slithy}, \text{JJ}\rangle, \langle M : \text{-thy}, \text{JJ}\rangle\} + v_0(\text{JJ}), \\
& \quad \ldots
\end{aligned}
$$

3.3 Learning

If we define $\boldsymbol{f}(\boldsymbol{x}_{1:N}, y_n, y_{n-1}, n) = \{\langle W : x_n, y_n\rangle, \langle T : y_{n-1}, y_n\rangle\}$, then our model is identical to the HMM. If we set the weights of these features to the log of their maximum-likelihood estimates,

$$
\begin{aligned}
w_{\langle W:x_n,y_n\rangle} &= \log \text{count}(x_n, y_n) - \log \text{count}(y_n) \\
w_{\langle W:y_{n-1},y_n\rangle} &= \log \text{count}(y_{n-1}, y_n) - \log \text{count}(y_{n-1}),
\end{aligned}
$$

then we exactly recover the HMM.

But to use more overlapping features and to get the advantages of error-driven learning, we're going to do perceptron updates. It's exactly the same as the non-structured perceptron:

- $\boldsymbol{w}^{(t+1)} \leftarrow \boldsymbol{w}^{(t)} + \boldsymbol{f}(\boldsymbol{x}, \boldsymbol{y}) - \boldsymbol{f}(\boldsymbol{x}, \hat{\boldsymbol{y}})$ is the standard update, using Viterbi to find $\hat{\boldsymbol{y}}$.
- As before, we can use MIRA to do large-margin training.

4 Conditional random fields

Structured perceptron works well in practice, and you will implement in your project 2, where it should work much much better than the Hidden Markov Model.

- But sometimes we need probabilities, and SP doesn't give us that.

- The Conditional Random Field (CRF) is a probabilistic conditional model for sequence labeling.
- Just as structured perceptron is built on the perceptron classifier, conditional random fields are built on the logistic regression classifier.

$$P(y|x) = \frac{e^{\boldsymbol{w}^\top \boldsymbol{f}(y,x)}}{\sum_{y' \in \mathcal{Y}} e^{\boldsymbol{w}^\top \boldsymbol{f}(y',x)}}$$

We can again move to structured prediction, assuming the same restriction on the scoring function:

$$P(\boldsymbol{y}|\boldsymbol{x}) = \frac{e^{\boldsymbol{w}^\top \boldsymbol{f}(\boldsymbol{y},\boldsymbol{x})}}{\sum_{\boldsymbol{y}' \in \mathcal{Y}(\boldsymbol{x})} e^{\boldsymbol{w}^\top \boldsymbol{f}(\boldsymbol{y}',\boldsymbol{x})}}$$

This is called a conditional random field because it models the sequence labeling task as a Markov random field, and estimates the probability of a set of variables **conditioned** on the others (as opposed to jointly, which the HMM does).

4.1 Decoding in CRFs

Decoding in the CRF does not depend on the denominator.

$$\begin{aligned}
\hat{\boldsymbol{y}} =& \arg\max_{\boldsymbol{y}} P(\boldsymbol{y}|\boldsymbol{x}) \\
=& \arg\max_{\boldsymbol{y}} \log P(\boldsymbol{y}|\boldsymbol{x}) \\
=& \arg\max_{\boldsymbol{y}} \boldsymbol{w}^\top \boldsymbol{f}(\boldsymbol{y},\boldsymbol{x}) - \log \sum_{\boldsymbol{y}' \in \mathcal{Y}(\boldsymbol{x})} e^{\boldsymbol{w}^\top \boldsymbol{f}(\boldsymbol{y}',\boldsymbol{x})} \\
=& \arg\max_{\boldsymbol{y}} \boldsymbol{w}^\top \boldsymbol{f}(\boldsymbol{y},\boldsymbol{x}) \\
=& \arg\max_{\boldsymbol{y}} \sum_{n}^{N} \boldsymbol{w}^\top \boldsymbol{f}_n(y_n, y_{n-1}, \boldsymbol{x}_{1:N}, n)
\end{aligned}$$

So we can just apply Viterbi directly, as defined in the structured perceptron. At each n, we have,

$$\begin{aligned}
v_n(k) =& \max_{k'} v_{n-1}(k') + \boldsymbol{w}^\top \boldsymbol{f}(y_n = k, y_{n-1} = k', \boldsymbol{x}, n) \\
=& \bigoplus_{k'} v_{n-1}(k') \otimes \boldsymbol{w}^\top \boldsymbol{f}(y_n = k, y_{n-1} = k', \boldsymbol{x}, n)
\end{aligned}$$

4.2 Learning in CRFs

Learning is a little more complicated. As with logistic regression, we need to learn weights to maximize the conditional log probability,

$$\begin{aligned}\ell &= \sum_{i}^{\#\text{instances}} \log P(\boldsymbol{y}_i|\boldsymbol{x}_i), \\ &= \sum_i \boldsymbol{w}^\top \boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{y}_i) - \log \sum_{\boldsymbol{y}' \in \mathcal{Y}(\boldsymbol{x}_i)} \exp\left(\boldsymbol{w}^\top \boldsymbol{f}(\boldsymbol{x}_i, \boldsymbol{y}')\right)\end{aligned}$$

As in logistic regression, the derivative is a difference between observed and expected counts:

$$\begin{aligned}\frac{d\ell}{dw_j} &= \sum_i \text{count}(\boldsymbol{x}_i, \boldsymbol{y}_i)_j - E_{\boldsymbol{y}|\boldsymbol{x}_i;\boldsymbol{w}}[\text{count}(\boldsymbol{x}_i, \boldsymbol{y})_j] \\ \text{count}(\boldsymbol{x}_i, \boldsymbol{y}_i)_j &= \sum_n^N f_{n,j}(\boldsymbol{x}_i, y_{i,n}, y_{i,n-1}, n)\end{aligned}$$

For example:

- If feature j is $\langle CC, DT \rangle$, then $c_j(\boldsymbol{x}_n, \boldsymbol{y}_n)$ is the count of times DT follows CC in the sequence $\boldsymbol{y}_n$.
- If feature j is $\langle M : \textit{-thy}, \text{JJ} \rangle$, then $\text{count}(\boldsymbol{x}_n, \boldsymbol{y}_n)_j$ is the count of words ending in *-thy* in $\boldsymbol{x}_n$ that are tagged JJ.

The expected feature counts are more complex.

- $E_{\boldsymbol{y}|\boldsymbol{x};\boldsymbol{w}}[\text{count}(\boldsymbol{x}_i, \boldsymbol{y})_j] = \sum_{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x}_i)} P(\boldsymbol{y}|\boldsymbol{x}_i; \boldsymbol{w}) f_j(\boldsymbol{x}, \boldsymbol{y})$
- This looks bad: we have to sum over an exponential number of labelings again.

- But remember that the feature function decomposes $f_j(\boldsymbol{x}, \boldsymbol{y}) = \sum_n f_j(\boldsymbol{x}, y_n, y_{n-1}, n)$.

$$
\begin{aligned}
E_{\boldsymbol{y}|\boldsymbol{x};\boldsymbol{w}}[\text{count}(\boldsymbol{x}, \boldsymbol{y})_j] &= \sum_{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x})} P(\boldsymbol{y}|\boldsymbol{x}; \boldsymbol{w}) f_j(\boldsymbol{x}, \boldsymbol{y}) \\
&= \sum_{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x})} P(\boldsymbol{y}|\boldsymbol{x}; \boldsymbol{w}) \sum_n^N f_j(\boldsymbol{x}, y_{n,i}, y_{n,i-1}, n) \\
&= \sum_n^N \sum_{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x})} P(\boldsymbol{y}|\boldsymbol{x}; \boldsymbol{w}) f_j(\boldsymbol{x}, y_{n,i}, y_{n,i-1}, n) \\
&= \sum_n^N \sum_{j,k \in \mathcal{Y}} \sum_{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x}): y_{n-1}=j, y_n=k} P(\boldsymbol{y}|\boldsymbol{x}; \boldsymbol{w}) f_j(\boldsymbol{x}, y_{n,i}, y_{n,i-1}, n) \\
&= \sum_n^N \sum_{j,k \in \mathcal{Y}} f_j(\boldsymbol{x}, y_{n,i}, y_{n,i-1}, n) \sum_{\boldsymbol{y} \in \mathcal{Y}(\boldsymbol{x}): y_{n-1}=j, y_n=k} P(\boldsymbol{y}|\boldsymbol{x}; \boldsymbol{w}) \\
&= \sum_n^N \sum_{j,k \in \mathcal{Y}} f_j(\boldsymbol{x}, y_{n,i}, y_{n,i-1}, n) P(y_{n-1} = j, y_n = k|\boldsymbol{x}; \boldsymbol{w})
\end{aligned}
$$

- The expected feature counts can be computed efficiently if we know the **marginal** probabilities $P(y_n, y_{n-1}|\boldsymbol{x}; \boldsymbol{w})$.
- This is the probability of traversing the edge $y_{n-1} \rightarrow y_n$, conditioned on the entire observation $\boldsymbol{x}_{1:N}$. [Draw this in trellis]
- To compute this marginal probability, we will apply the forward-backward algorithm.

5 The forward-backward algorithm

Here we require the marginal probability, e.g. $P(y_n = \text{NNP}, y_{n-1} = \text{DET}|x_{1:N})$.

We can rewrite the marginal probability of a single tag transition $P(y_n, y_{n-1}|x_{1:N})$ as a ratio:

$$P(y_n, y_{n-1}|x_{1:N}) = \frac{P(y_n, y_{n-1}, x_{1:N})}{P(x_{1:N})} \tag{15}$$

Recall the Forward Algorithm, which defines the forward probabilities $\alpha_n(k) \triangleq P(y_n, x_{1:n})$. This allows us to compute the denominator $P(x_{1:N}) = \sum_k \alpha_N(k)$.

5.1 Forward algorithm

First we need to show how to use the forward algorithm in a CRF. We're going to switch to a semiring in which we work with probabilities rather than log-probabilities, so that we can use simple addition rather than log-addition. But in practice we'd work in a log-probability semiring, where $a \oplus b = \log(e^a + e^b)$. For a practical implementation, see `numpy.logaddexp` and `scipy.misc.logsumexp`.

$$\alpha_n(k) \triangleq P(y_n = k, x_{1:n}) = \sum_{y_{1:n}: y_n = k} P(\boldsymbol{y}_{1:n}, \boldsymbol{x}_{1:n}) \tag{16}$$

$$= \sum_{\boldsymbol{y}_{1:n}: y_n = k} \exp(\sum_{m}^{n} \boldsymbol{w}^\mathsf{T} \boldsymbol{f}(y_m, y_{m-1}, \boldsymbol{x}, m)) \tag{17}$$

$$= \sum_{\boldsymbol{y}_{1:n}} \exp(\boldsymbol{w}^\mathsf{T} \boldsymbol{f}(y_n = k, y_{n-1}, \boldsymbol{x}, n)) \exp(\sum_{m}^{n-1} \boldsymbol{w}^\mathsf{T} \boldsymbol{f}(y_m, y_{m-1}, \boldsymbol{x}, m)) \tag{18}$$

$$= \sum_{y_{n-1} = k'} \exp(\boldsymbol{w}^\mathsf{T} \boldsymbol{f}(y_n = k, y_{n-1} = k', \boldsymbol{x}, n)) \tag{19}$$

$$\times \sum_{\boldsymbol{y}_{1:n-1}: y_{n-1} = k'} \exp(\sum_{m}^{n-1} \boldsymbol{w}^\mathsf{T} \boldsymbol{f}(y_m, y_{m-1}, \boldsymbol{x}, m)) \tag{20}$$

$$= \sum_{y_{n-1} = k'} \exp\left(\boldsymbol{w}^\mathsf{T} \boldsymbol{f}(y_n = k, y_{n-1} = k', \boldsymbol{x}, n)\right) \alpha_{n-1}(k') \tag{21}$$

$$P(\boldsymbol{x}_{1:N}) = \sum_{k} \alpha_N(y_N = k) \tag{22}$$

We can see an example of this in the slides. In this example, we have only emission and transition features. The feature weights are shown in the

slide. So to compute the forward variable for *can* tagged as N, we have,

$$\begin{aligned}\alpha_2(\text{N}) =& (-4 \otimes -3 \otimes -3) \oplus (-5 \otimes -1 \otimes -3) && (23)\\ =& (-10) \oplus (-9) && (24)\\ =& \log\left(e^{-10} + e^{-9}\right) && (25)\\ \approx& -8.7 && (26)\end{aligned}$$

Now we return to the numerator of $P(y_n, y_{n-1}|x_{1:N}) = \frac{P(y_n, y_{n-1}, x_{1:N})}{P(x_{1:N})}$. We can factor,

$$\begin{aligned}P(y_n, y_{n-1}, x_{1:N}) &= P(\boldsymbol{x}_{n+1:N}|y_n, y_{n-1}, x_{1:N})P(y_n, y_{n-1}, x_{1:N})\\ &= P(\boldsymbol{x}_{n+1:N}|y_n)P(y_n, x_n|y_{n-1})P(y_{n-1}, x_{1:n-1})\\ &\propto P(\boldsymbol{x}_{n+1:N}|y_n)\exp\left(\boldsymbol{w}^\mathsf{T}\boldsymbol{f}(y_n, y_{n-1}, \boldsymbol{x}, n)\right)\alpha_{n-1}(y_{n-1}),\end{aligned}$$

where we have plugged in the forward variable $\alpha_n(y_{n-1}) = P(y_{n-1}, \boldsymbol{x}_{1:n-1})$. Now we define a **backward variable**, $b_n(y_n) = P(\boldsymbol{x}_{n+1:N}|y_n)$, yielding

$$P(y_n, y_{n-1}|x_{1:N}) \propto b_n(y_n)\exp\left(\boldsymbol{w}^\mathsf{T}\boldsymbol{f}(y_n, y_{n-1}, \boldsymbol{x}, n)\right)\alpha_{n-1}(y_{n-1}).$$

If we want to compute the marginal for a single node, we have

$$P(y_n|\boldsymbol{x}_{1:N}) \propto b_n(y_n)\alpha_n(y_n) = P(\boldsymbol{x}_{n+1:N}|y_n)P(y_n, x_{1:n}) = P(y_n, \boldsymbol{x}_{1:N}) \quad (27)$$

This is useful for applying expectation-maximization to unsupervised learning of HMMs.

5.2 The backwards algorithm

We will need another recursive algorithm to compute the backwards variables. Fortunately, the backwards probability also decomposes into a recurrence relation,

$$\begin{aligned}b_n(y_n) \triangleq& P(x_{n+1:N}|y_n)\\ =& \sum_{y_{n+1}} P(x_{n+1:N}, y_{n+1}|y_n)\\ =& \sum_{y_{n+1}} P(x_{n+2:N}|y_{n+1}, x_{n+1})P(x_{n+1}, y_{n+1}|y_n)\\ \propto& \sum_{y_{n+1}} b_{n+1}(y_{n+1})\exp\left(\boldsymbol{w}^\mathsf{T}\boldsymbol{f}(y_{n+1}, y_n, \boldsymbol{x}_{1:N}, n)\right)\end{aligned}$$

$$b_N(y_N) = P(\text{END}|y_N)$$

The steps are as follows:

- introduce y_{n+1} and sum over all possible values
- apply the chain rule to extract y_{n+1} and x_{n+1}
- use the definition of the backwards probability and the model assumptions

The backwards algorithm moves from right to left, starting with $B_N(y_N) = P(\text{END}|y_N)$, the score for transitioning to the end state given y_N.

We can see an example in the slides.

5.3 Learning in CRFs: wrapup

The overall procedure looks just like logistic regression:

- Use forward backward to compute expected feature counts under $P(\boldsymbol{y}|\boldsymbol{x}; \boldsymbol{w})$
- Compute gradients as difference between feature counts and expected counts
- Update $\boldsymbol{w}$ (using quasinewton optimization)
- Iterate