1 Introduction

Let $\mathbf{x} = (x^1, \ldots, x^M)$ denote a sequence of inputs (e.g., a sequence fo words), and let $\mathbf{y} = (y^1, \ldots, y^M)$ denote the corresponding sequence of outputs (e.g., a sequence of part of speech tags. We use x and y denote a single input and output token, respectively.

In general, we use M to denote the length of a sequence. For simplicity, we assume that $\mathbf{x}$ is encoded using D words, so there are D^M possible length-M sequences $\mathbf{x}$. We also assume that there are are L possible labels for each y^j, so there are L^M possible length-M sequences of $\mathbf{y}$.

We are given a training set of N training examples:

$$S = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N,$$

and our goal is to learn a good mapping from $\mathbf{x}$ to $\mathbf{y}$ using S. We will do so by fitting a 1st order sequential Conditional Random Field (CRF) to S.

We use capital notation $\mathbf{Y} = (Y^1, \ldots, Y^M)$ to denote random variables, i.e., variables that can take on multiple values according to some probability distribution. A CRF is a "log-linear" conditional probabilistic model that models the distribution of $\mathbf{Y}$ conditional on a specific $\mathbf{x}$:

$$P_w(\mathbf{Y} = \mathbf{y}|\mathbf{x}) = \frac{1}{Z_w(\mathbf{x})} \exp\left\{\sum_{j=1}^M w^\top \phi_j(y^j, y^{j-1}|\mathbf{x})\right\} \equiv \frac{1}{Z_w(\mathbf{x})} \exp\left\{F_w(\mathbf{y}, \mathbf{x})\right\}, \tag{1}$$

where $F_w(\mathbf{y}, \mathbf{x})$ is often referred to as the joint discriminant function:

$$F_w(\mathbf{y}, \mathbf{x}) \equiv \sum_{j=1}^M w^\top \phi_j(y^j, y^{j-1}|\mathbf{x}),$$

and the model is parameterized by a weight vector w. $Z_w(\mathbf{x})$ is the so-called partition function:

$$Z_w(\mathbf{x}) = \sum_{\mathbf{y}'} \exp\left\{F_w(\mathbf{y}', \mathbf{x})\right\}, \tag{2}$$

which sums over the scores of all possible $\mathbf{y}'$, and acts as a normalizer so that (1) yields a valid probability.

In this tutorial, we will see how to:

- Instantiate the feature map $\phi^j(y, y'|\mathbf{x})$ to correspond to a 1st order sequence labeling model.
- Compute the conditional probability $P_w(\mathbf{Y} = \mathbf{y}|\mathbf{x})$ of a trained model w and a give $(\mathbf{x}, \mathbf{y})$.
- Make predictions given an input sequence $\mathbf{x}$: $\text{argmax}_{\mathbf{y}} P(\mathbf{y}|\mathbf{x})$.
- Compute gradients of the negative log likelihood of the training set S: $\partial_w - \sum_{i=1}^N \log P_w(\mathbf{Y} = \mathbf{y}_i|\mathbf{x}_i)$

2 Feature Representation

In (1), $\phi^j(y, y'|\mathbf{x})$ is a feature mapping function that takes:

- a location in the sequence j,
- a pair of adjacent labels (y, y'),

- and the input sequence $\mathbf{x}$,

and returns a feature vector. We now show how to instantiate $\phi^j(y, y'|\mathbf{x})$ to correspond to a 1st order sequence labeling model.

Simplest Case. In the simplest case, the features in $\phi^j(y, y'|\mathbf{x})$ encode two types of information, and can thus be broken down into two groups.

Input/Output Features. The first group, denoted $\phi_1^j(y|\mathbf{x})$, are input/output indicator features:

$$\phi_1^j(y, y'|\mathbf{x}) = \left\{\mathbf{1}_{[(y=a)\wedge(x^j=z)]}\right\}_{a=1,z=1}^{a=L,z=D}.$$

For example, for $L = 2$ and $D = 3$, there are six such indicator features:

$$\phi_1^j(y|\mathbf{x}) = \begin{bmatrix} \mathbf{1}_{[(y=1)\wedge(x^j=1)]} \\ \mathbf{1}_{[(y=1)\wedge(x^j=2)]} \\ \mathbf{1}_{[(y=1)\wedge(x^j=3)]} \\ \mathbf{1}_{[(y=2)\wedge(x^j=1)]} \\ \mathbf{1}_{[(y=2)\wedge(x^j=2)]} \\ \mathbf{1}_{[(y=2)\wedge(x^j=3)]} \end{bmatrix}. \tag{3}$$

Output/Output Transition Features. The second group, denoted $\phi_2(y, y')$, are output/output transition indicator features:

$$\phi_2(y, y') = \left\{\mathbf{1}_{[(y=a)\wedge(y'=b)]}\right\}_{a=1,b=1}^{a=L,b=L}.$$

For example, for $L = 2$, there are six such indicator features:

$$\phi_2(y, y') = \begin{bmatrix} \mathbf{1}_{[(y=1)\wedge(y'=0)]} \\ \mathbf{1}_{[(y=1)\wedge(y'=1)]} \\ \mathbf{1}_{[(y=1)\wedge(y'=2)]} \\ \mathbf{1}_{[(y=2)\wedge(y'=0)]} \\ \mathbf{1}_{[(y=2)\wedge(y'=1)]} \\ \mathbf{1}_{[(y=2)\wedge(y'=2)]} \end{bmatrix}. \tag{4}$$

Here we use $y' = 0$ to refer to a transition from the special "Start" state to another output/state.

Overall Feature Representation. We can then write the overall feature mapping $\phi^j(y, y'|\mathbf{x})$ as the vector concatenation:

$$\phi^j(y, y'|\mathbf{x}) = \begin{bmatrix} \phi_1^j(y|\mathbf{x}) \\ \phi_2(y, y') \end{bmatrix}.$$

The first group $\phi_1^j(y|\mathbf{x})$ encodes the standard multi-task logistic regression features for each token y'^j. The second group $\phi_2(y, y')$ encodes the 1-st order transition features. Because of this common demarcation of two feature groups, one often writes the weight vector correspondingly as a concatenation of two sub-vectors:

$$w = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}. \tag{5}$$

Thus we can rewrite $w^\top \phi^j(y, y'|\mathbf{x})$ equivalently as

$$w^\top \phi^j(y, y'|\mathbf{x}) = w_1^\top \phi_1^j(y|\mathbf{x}) + w_2^\top \phi_2(y, y').$$

2.1 Example

We use the simple example defined in (3) and (4) to show how to use Conditional Random Fields. We will also use the decomposition of the weight vector defined in (5) and instantiate w as:

$$w_1 = \begin{bmatrix} 1 \\ 2 \\ -3 \\ 0 \\ 0 \\ 1 \end{bmatrix}, \qquad w_2 = \begin{bmatrix} 1 \\ -2 \\ 2 \\ -1 \\ -1 \\ 0 \end{bmatrix}. \tag{6}$$

In this example, we wish to compute $P_w(\mathbf{Y}|\mathbf{x} = (1,2))$. Recall that:

$$P_w(\mathbf{Y} = \mathbf{y}|\mathbf{x} = (1,2)) = \frac{1}{Z_w(\mathbf{x} = (1,2))} \exp\left\{F(\mathbf{y}, \mathbf{x} = (1,2))\right\}, \tag{7}$$

and that:

$$Z_w(\mathbf{x} = (1,2)) = \sum_{\mathbf{y}'} \exp\{F(\mathbf{y}', \mathbf{x} = (1,2))\}.$$

Thus, we just need to compute $F(\mathbf{y}, \mathbf{x} = (1,2))$ for all possible $\mathbf{y}$ in order to compute $P_w(\mathbf{Y} = \mathbf{y}|\mathbf{x} = (1,2)$ for all possible $\mathbf{y}$. Using the feature maps in (3) and (4), we can write $F(\mathbf{y}|\mathbf{x} = (1,2))$ as:

$$F(\mathbf{y}, \mathbf{x} = (1,2)) = w_1^\top \phi_1^1(y^1|\mathbf{x} = (1,2)) + w_2^\top \phi_2(y^1, y^0 = 0) + w_1^\top \phi_1^2(y^2|\mathbf{x} = (1,2)) + w_2^\top \phi_2(y^2, y^1). \tag{8}$$

For completeness, we show this computation in Table 1 below:

Table 1: Computation of the simple example based on feature maps (3) and (4) and weight vectors in (6)

y^1	y^2	$F(\mathbf{y}, \mathbf{x} = (1,2))$	$\exp\{F(\mathbf{y}, \mathbf{x} = (1,2))\}$
1	1	$w_{1,1} + w_{2,1} + w_{1,2} + w_{2,2} = 1 + 1 + 2 - 2 = 0$	$\exp\{0\}$
1	2	$w_{1,1} + w_{2,1} + w_{1,5} + w_{2,5} = 1 + 1 + 0 - 1 = 1$	$\exp\{1\}$
2	1	$w_{1,4} + w_{2,4} + w_{1,2} + w_{1,3} = 0 - 1 + 2 + 2 = 3$	$\exp\{3\}$
2	2	$w_{1,4} + w_{2,4} + w_{1,5} + w_{2,6} = 1 + 1 + 0 + 0 = 2$	$\exp\{2\}$

We can also see that the optimal prediction that has maximal conditional probability is $\mathbf{y} = (2,1)$:

$$(2,1) = \text{argmax}_{\mathbf{y}} P_w(\mathbf{Y} = \mathbf{y}|\mathbf{x} = (1,2)) = \text{argmax}_{\mathbf{y}} F(\mathbf{y}|\mathbf{x} = (1,2)).$$

Our simple example was small enough to allow for exhaustive enumeration of all possible $\mathbf{y}$. In general, the number of possible $\mathbf{y}$ scales exponentially w.r.t. the sequence length $M = |\mathbf{x}|$. The rest of this tutorial describes how to efficiently compute conditional probabilities $P_w(\mathbf{Y} = \mathbf{y}|\mathbf{x})$, compute the optimal predictions $\text{argmax}_{\mathbf{y}} P_w(\mathbf{Y} = \mathbf{y}|\mathbf{x})$, and compute gradients for gradient descent training.

2.2 Reduction to Independent Logistic Regression

If ϕ_1^j were the only features in our feature map, then we can rewrite the discriminant as

$$F_w(\mathbf{y}, \mathbf{x}) = \sum_{j=1}^{M} w^\top \phi_1^j(y^j|\mathbf{x}),$$

which computes a completely disjoint score for each output token y^j. In that case, the CRF model decomposes to

$$P_w(\mathbf{Y} = \mathbf{y}|\mathbf{x}) = \frac{1}{Z_w(\mathbf{x})} \exp\left\{\sum_{j=1}^{M} w^\top \phi_1^j(y^j|\mathbf{x})\right\} = \frac{1}{Z_w(\mathbf{x})} \prod_{j=1}^{M} \exp\left\{w^\top \phi_1^j(y^j|\mathbf{x})\right\}. \tag{9}$$

We can also write $Z_w(\mathbf{x})$ as

$$Z_w(\mathbf{x}) = \sum_{\mathbf{y}'} \exp\left\{F_w(\mathbf{y}', \mathbf{x})\right\} = \sum_{\mathbf{y}'} \prod_{j=1}^{M} \exp\left\{w^\top \phi_1^j(y'^j|\mathbf{x})\right\} = \prod_{j=1}^{M} \sum_{y'^j} \exp\left\{w^\top \phi_1^j(y'^j|\mathbf{x})\right\}. \tag{10}$$

Using (10), we can re-write (9) as

$$\prod_{j=1}^{M} \frac{\exp\left\{w^\top \phi_1^j(y^j|\mathbf{x})\right\}}{\sum_{y'^j} \exp\left\{w^\top \phi_1^j(y'^j|\mathbf{x})\right\}} = \prod_{j=1}^{N} P_w(Y^j = y'^j|\mathbf{x}),$$

which is the independent conditional probability of each output token y'^j. Note that the independent conditional probability of a single output token is exactly logistic regression. So without the pairwise features $\phi_2(y, y'|\mathbf{x})$, the CRF reduces to a independent logistic regression per output token.

3 Computing Conditional Probability

In this section, we show how to efficiently compute the conditional probability $P_w(\mathbf{Y} = \mathbf{y}|\mathbf{x})$ for any $(\mathbf{x}, \mathbf{y})$ and model w. Since we restrict ourselves to a single $(\mathbf{x}, \mathbf{y})$ in this section, for brevity we suppress the use of $(\mathbf{x}, \mathbf{y})$ in the notation wherever possible.

Define a matrix G^j as

$$G^j = \left[\exp\left\{w^\top \phi^j(a, b|\mathbf{x})\right\}\right]_{ab}, \tag{11}$$

which encodes the un-normalized probability of transitioning from $y^{j-1} = b$ to $y^j = a$ in the j-th token of the sequence.

For $j = 1$, all transitions must come from the "Start" state, so we can write G^1 as a vector:

$$G^1 = \left[\exp\left\{w^\top \phi^1(a, 0|\mathbf{x})\right\}\right]_a.$$

For length-1 sequences, we can write the partition function $Z_w(x)$ from (2) as:

$$Z_w(x) = \sum_a \exp\left\{w^\top \phi^1(a, 0|x)\right\} = \sum_a G^1_a,$$

which is simply the sum of the vector G^1.

For length-2 sequences, we can write the partition function as:

$$Z_w(\mathbf{x}) = \sum_{b,a} \exp\left\{w^\top \phi^2(a, b|\mathbf{x}) + w^\top \phi^1(b, 0|\mathbf{x})\right\} \tag{12}$$

$$= \sum_{b,a} \exp\left\{w^\top \phi^2(a, b|\mathbf{x})\right\} \exp\left\{w^\top \phi^1(b, 0|\mathbf{x})\right\} \tag{13}$$

$$= \sum_a \left[G^2 G^1\right]_a, \tag{14}$$

where (14) is simply the sum of the vector $G^2 G^1$. We now introduce the notation

$$G^{i:j} = G^j G^{j-1} \dots G^{i+1} G^i. \tag{15}$$

Note that $G^{i:j}$ is a vector when $i = 1$ and a matrix otherwise. We can now rewrite (14) as the sum of the vector $G^{1:2}$, and in general:

$$G^{1:j} = G^j G^{1:j-1}. \tag{16}$$

We can extend the computation of $Z_w(\mathbf{x})$ to arbitary length-M sequences via:

$$Z_w(\mathbf{x}) = \sum_a G_a^{1:M}. \tag{17}$$

The running time of computing (17) is $\mathcal{O}(ML^2)$, where M is the length of the sequence and L the number of output labels per token. We can now efficiently compute $P_w(\mathbf{Y} = \mathbf{y}|\mathbf{x}) = \exp\{F_w(\mathbf{y}, \mathbf{x})\}/Z_w(\mathbf{x})$.

Dealing with Numerical Instability. In practice, we often compute $\log(Z_w(\mathbf{x}))$ instead for numerical stability reasons:

$$\log\left(Z_w(\mathbf{x})\right) = \log\left(\sum_a G_a^{1:M}\right). \tag{18}$$

One way to compute the log-sum in the RHS of (18) is to keep track of a multiplicative factor for each computation of (16):

$$\tilde{G}^{1:j} = \frac{1}{C_G^j}\left(G^j \tilde{G}^{1:j-1}\right), \tag{19}$$

where C_G^j is a multiplicative factor that keeps $G^{1:j}$ from blowing up. For example, a common choice is

$$C_G^j = \sum_a \left[G^j \tilde{G}^{1:j-1}\right]_a,$$

which implies that the entries of $\tilde{G}^{1:j}$ in (19) sum to 1. Note that:

$$G^{1:j} = \tilde{G}^{1:j} \prod_{j'=1}^{j} C_G^{j'}.$$

One can thus compute $\log(Z_w(\mathbf{x}))$ in (18) as:

$$\log\left(Z_w(\mathbf{x})\right) = \log\left(\sum_a \tilde{G}_a^{1:M}\right) + \sum_{j=1}^{M} \log\left(C_G^j\right).$$

One can then keep track of the log conditional probabilities as:

$$\log\left(P_w(\mathbf{y}|\mathbf{x})\right) = F_w(\mathbf{y}, \mathbf{x}) - \log\left(Z_w(\mathbf{x})\right).$$

4 Viterbi Algorithm for Making Predictions

Given a test instance $\mathbf{x}$, we make predictions by selecting the $\mathbf{y}$ that maximizes $P_w(\mathbf{Y} = \mathbf{y}|\mathbf{x})$, i.e.,:

$$\text{argmax}_{\mathbf{y}} \, P_w(\mathbf{Y} = \mathbf{y}|\mathbf{x}) = \text{argmax}_{\mathbf{y}} \, \frac{1}{Z_w(\mathbf{x})} \exp\left\{F_w(\mathbf{y}, \mathbf{x})\right\} = \text{argmax}_{\mathbf{y}} \, F_w(\mathbf{y}, \mathbf{x}). \tag{20}$$

Naively iterating over all possible $\mathbf{y}$ takes exponential time (w.r.t. the length $|\mathbf{x}| = M$). We show here how to use a dynamic programming approach known as the Viterbi algorithm to efficiently compute (20).

Let $\hat{\mathbf{y}}_a^j$ denote the best length-j prefix solution that ends in $y^j = a$:

$$\hat{\mathbf{y}}_a^j = \left(\text{argmax}_{\mathbf{y}^{1:j-1} = (y^1, \ldots, y^{j-1})} \, F_w(\mathbf{y}^{1:j-1} \oplus a, \mathbf{x})\right) \oplus a, \tag{21}$$

where $\oplus$ denotes sequence concatenation or appending (should be clear from context which one it is).

If one had already computed the length-M prefix solutions $\hat{\mathbf{y}}_a^M$, then one can predict by simply selecting the $\hat{\mathbf{y}}_a^M$ with highest score $F_w(\hat{\mathbf{y}}_a^M, \mathbf{x})$. For $j = 1$, computing each $\hat{\mathbf{y}}_a^1$ is trivial since by definition $\hat{\mathbf{y}}_a^1 = a$. Via the Viterbi algorithm, one can compute $\hat{\mathbf{y}}_a^j$ for $j > 1$ by looping through all the $\hat{\mathbf{y}}_b^{j-1}$ and choosing the best one:

$$\hat{\mathbf{y}}_a^j = \left(\text{argmax}_{\mathbf{y}^{1:j-1} \in \left\{\hat{\mathbf{y}}_1^{j-1}, \ldots, \hat{\mathbf{y}}_L^{j-1}\right\}} \, F(\mathbf{y}^{1:j-1} \oplus a, \mathbf{x})\right) \oplus a.$$

The running time of this approach is $\mathcal{O}(L^2 M)$.

5 Computing Gradients

Conditional Random Fields are typically trained using gradient descent to minimize the negative log loss over a training set $S = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$:

$$\text{argmin}_w - \sum_{i=1}^N \log P_w(\mathbf{Y} = \mathbf{y}_i|\mathbf{x}_i) = \text{argmin}_w \sum_{i=1}^N \left[-F_w(\mathbf{y}_i, \mathbf{x}_i) + \log\left(Z_w(\mathbf{x}_i)\right)\right]. \tag{22}$$

For clarity, we focus on deriving the gradient w.r.t. a single training example $(\mathbf{x}, \mathbf{y})$. The derivation extends trivially to the gradient over an entire training set by linearity of differentiation.

We can compute the gradient with respect to both the discriminant $F_w(\mathbf{y}, \mathbf{x})$ and the log partition $\log Z_w(\mathbf{x})$. The derivation of $\nabla_w F(\mathbf{y}, \mathbf{x})$ is fairly straightforward:

$$\nabla_w F_w(\mathbf{y}, \mathbf{x}) = \sum_{j=1}^M \nabla_w \left(w^\top \phi^j(y^j, y^{j-1}|\mathbf{x})\right) = \sum_{j=1}^M \phi^j(y^j, y^{j-1}|\mathbf{x}). \tag{23}$$

The derivation of $\nabla_w \log Z_w(\mathbf{x})$ is more involved:

$$\nabla_w \log Z_w(\mathbf{x}) = \frac{1}{Z_w(\mathbf{x})} \nabla_w Z_w(\mathbf{x}) \tag{24}$$

$$= \frac{1}{Z_w(\mathbf{x})} \nabla_w \sum_{\mathbf{y}'} \exp\left\{F_w(\mathbf{y}', \mathbf{x})\right\} \tag{25}$$

$$= \frac{1}{Z_w(\mathbf{x})} \sum_{\mathbf{y}'} \nabla_w \exp\left\{F_w(\mathbf{y}', \mathbf{x})\right\} \tag{26}$$

$$= \frac{1}{Z_w(\mathbf{x})} \sum_{\mathbf{y}'} \exp\left\{F_w(\mathbf{y}', \mathbf{x})\right\} \nabla_w F_w(\mathbf{y}', \mathbf{x}) \tag{27}$$

$$= \frac{1}{Z_w(\mathbf{x})} \sum_{\mathbf{y}'} \exp\left\{F_w(\mathbf{y}', \mathbf{x})\right\} \nabla_w \left(\sum_{j=1}^{M} w^\top \phi^j(y'^j, y'^{j-1}|\mathbf{x} \right) \tag{28}$$

$$= \frac{1}{Z_w(\mathbf{x})} \sum_{\mathbf{y}'} \exp\left\{F_w(\mathbf{y}', \mathbf{x})\right\} \sum_{j=1}^{M} \nabla_w \left(w^\top \phi^j(y'^j, y'^{j-1}|\mathbf{x}) \right) \tag{29}$$

$$= \frac{1}{Z_w(\mathbf{x})} \sum_{\mathbf{y}'} \exp\left\{F_w(\mathbf{y}', \mathbf{x})\right\} \sum_{j=1}^{M} \phi^j(y'^j, y'^{j-1}|\mathbf{x}) \tag{30}$$

$$= \sum_{\mathbf{y}'} \frac{\exp\left\{F_w(\mathbf{y}', \mathbf{x})\right\}}{Z_w(\mathbf{x})} \sum_{j=1}^{M} \phi^j(y'^j, y'^{j-1}|\mathbf{x}) \tag{31}$$

$$= \sum_{\mathbf{y}'} P_w(\mathbf{Y} = \mathbf{y}'|\mathbf{x}) \sum_{j=1}^{M} \phi^j(y'^j, y'^{j-1}|\mathbf{x}) \tag{32}$$

$$= \sum_{j=1}^{M} \sum_{\mathbf{y}'} P_w(\mathbf{Y} = \mathbf{y}'|\mathbf{x}) \phi^j(y'^j, y'^{j-1}|\mathbf{x}) \tag{33}$$

$$= \sum_{j=1}^{M} E_{\mathbf{y}' \sim P_w(\mathbf{Y}|\mathbf{x})}[\phi^j(y'^j, y'^{j-1}|\mathbf{x})]. \tag{34}$$

Thus the gradient of $Z_w(\mathbf{x})$, $\nabla_w Z_w(\mathbf{x})$, is equal the (sum of the) expected value of each feature $\phi^j(y'^j, y'^{j-1})|\mathbf{x})$ under the distribution induced by the current model $P_w(\mathbf{Y} = \mathbf{y}'|\mathbf{x})$.

The gradient of (22) can thus be written as:

$$\nabla_w = -\sum_{j=1}^{M} \sum_{i=1}^{N} \phi^j(y_i^j, y_i^{j-1}|\mathbf{x}_i) + \sum_{j=1}^{M} \sum_{i=1}^{N} E_{\mathbf{y}' \sim P_w(\mathbf{Y}|\mathbf{x}_i)}[\phi^j(y'^j, y'^{j-1}|\mathbf{x}_i)]. \tag{35}$$

$$= \sum_{j=1}^{M} \sum_{i=1}^{N} \left(-\phi^j(y_i^j, y_i^{j-1}|\mathbf{x}_i) + E_{\mathbf{y}' \sim P_w(\mathbf{Y}|\mathbf{x}_i)}[\phi^j(y'^j, y'^{j-1}|\mathbf{x}_i)] \right). \tag{36}$$

Implication of Optimality Condition. The standard optimality condition of (22) requires that the gradient (35) be 0 for the optimal w, which implies that:

$$\sum_{j=1}^{M} \sum_{i=1}^{N} \phi^j(y_i^j, y_i^{j-1}|\mathbf{x}_i) = \sum_{j=1}^{M} \sum_{i=1}^{N} E_{\mathbf{y}' \sim P_w(\mathbf{Y}|\mathbf{x}_i)}[\phi^j(y'^j, y'^{j-1}|\mathbf{x}_i)]. \tag{37}$$

In other words, for the optimal w, the expected value of each feature $\phi^j(y'^j, y'j-1|\mathbf{x})$ over the conditional probability $P_w(\mathbf{Y} = \mathbf{y}'|\mathbf{x})$ should equal the frequency counts of that feature over the training set. In the case where each $\phi^j(y'^j, y'^{j-1}|\mathbf{x})$ occupies a disjoint region of the total feature vector, then (37) can be decomposed to:

$$\forall j \in \{1, \ldots, M\} : \sum_{i=1}^{N} \phi^j(y_i^j, y_i^{j-1}|\mathbf{x}_i) = \sum_{i=1}^{N} E_{\mathbf{y}' \sim P_w(\mathbf{Y}|\mathbf{x}_i)}[\phi^j(y'^j, y'^{j-1}|\mathbf{x}_i)]. \tag{38}$$

5.1 Forward-Backward Algorithm to Compute Conditional Probabilities

The expectation in (34) can be rewritten as

$$E_{y \sim P_w(\mathbf{Y}|\mathbf{x})}\phi^j(y^j, y^{j-1}|\mathbf{x}) \equiv \sum_{y} P_w(\mathbf{Y} = \mathbf{y}|\mathbf{x})\phi^j(y^j, y^{j-1}|\mathbf{x}) \tag{39}$$

$$\equiv \sum_{y^1,\ldots,y^M} P_w(\mathbf{Y} = (y^1, \ldots, y^M|\mathbf{x})\phi^j(y^j, y^{j-1}|\mathbf{x}) \tag{40}$$

$$= \sum_{a,b} P_w(Y^j = a, Y^{j-1} = b|\mathbf{x})\phi^j(a, b|\mathbf{x}) \tag{41}$$

We can further observe that for any specific transition $Y^{j-1} = b$ to $Y^j = a$:

$$P_w(Y^j = a, Y^{j-1} = b|\mathbf{x}) = \sum_{y^1,\ldots,y^{j-2}} \sum_{y^{j+1},\ldots,y^M} P_w(\mathbf{Y} = (y^1, \ldots y^{j-2}, b, a, y^{j+1}, \ldots, y^M)|\mathbf{x}). \tag{42}$$

Naively summing over all possible $y^1, \ldots, y^{j-2}$ and $y^{j+1}, \ldots, y^M$ will take exponential time (w.r.t. M). We will show how to use a dynamic programming approach called the Forward-Backward algorithm to efficiently compute the two summations in (42).

We define a sequence of vectors α^j which will correspond to the unnormalized probability of:

$$\alpha^j(a) \propto \sum_{y^1,\ldots,y^{j-1}} P_w(\mathbf{Y}^{1:j} = (y^1, \ldots, y^{j-1}, a)|\mathbf{x}).$$

We similarly define a sequence of vectors β^j which will correspond to the unnormalized probability of:

$$\beta^j(a) \propto \sum_{y^{j+1},\ldots,y^M} P_w(\mathbf{Y}^{j:M} = (b, y^{j+1}, \ldots, y^M)|\mathbf{x}).$$

After computing the α^j and β^j vectors, we can thus write (42) as

$$P_w(Y^{j-1} = b, Y^j = a|\mathbf{x}) = \frac{\alpha^{j-1}(a)G^j(b, a)\beta^j(b)}{Z_w(\mathbf{x})}. \tag{43}$$

It remains to show how to compute each α^j and β^j efficiently.

We will compute α^j recursively using α^{j-1} (i.e., the Forward Algorithm), and β^j recursively using β^{j+1} (i.e., the Backward Algorithm).

We inialize α^0 and β^M as:

$$\alpha^0(a) = \begin{cases} 1 & \text{if } a = \texttt{Start} \\ 0 & \text{otherwise} \end{cases}, \qquad \beta^M(b) = 1.$$

Similar to the Viterbi algorithm, we recursively define each α^j as:

$$\alpha^j(a) = \sum_b \alpha^{j-1}(b) G^j(a,b), \tag{44}$$

which can be simplified in matrix notation as:

$$\alpha^j = G^j \alpha^{j-1}. \tag{45}$$

Recall that G^j is defined in (11).

We can also recursively define each β^j as:

$$\beta^j(b) = \sum_a \beta^{j+1}(a) G^{j+1}(a,b), \tag{46}$$

which can be simplified in matrix notation as:

$$\beta^j = G^{(j+1)\top} \beta^{j+1}. \tag{47}$$

Dealing With Numerical Instability. Although (43) is mathematically elegant, in practice directly implementing (45) and (47) leads to numerical instability. For instance, α, β, and $Z_w(\mathbf{x})$ can all overflow or underflow if we compute them as is. The first observation we make from the definition of $Z_w(\mathbf{x})$ in (17) and (18) is that:

$$Z_w(\mathbf{x}) = \sum_{a,b} \alpha^{j-1}(a) G^j(b,a) \beta^j(b).$$

Thus, we do not need to actually compute $Z_w(\mathbf{x})$ in order to compute (43). All we need to do is to renormalize each α^j and β^j after each step in (45) and (47), i.e.:

$$\tilde{\alpha}^j = \frac{1}{C_\alpha^j} \left(G^j \tilde{\alpha}^{j-1} \right), \tag{48}$$

and

$$\tilde{\beta}^j = \frac{1}{C_\beta^j} \left(G^{(j+1)\top} \tilde{\beta}^{j+1} \right), \tag{49}$$

for some choice of C_α^j and C_β^j such that overflow and underflow do not happen. Afterwards, we can compute (43) as

$$P_w(Y^{j-1} = a, Y^j = b | \mathbf{x}) = \frac{\tilde{\alpha}^{j-1}(a) G^j(b,a) \tilde{\beta}^j(b)}{\sum_{a',b'} \tilde{\alpha}^{j-1}(a') G^j(b',a') \tilde{\beta}^j(b')}. \tag{50}$$

Relationship to Viterbi. The main difference between the Forward Algorithm described in (44) and the Viterbi Algorithm is that the Forward Algorithm sums over all possibilities whereas the Viterbi Algorithm only keeps the most likely possibility. Indeed the Forward-Backward Algorithm is known as a special case of the more general Sum-Product Algorithm, whereas the Viterbi Algorithm is a special case of the more general Max-Product Algorithm. The special case here is due to the fact that we are only considering linear pairwise chains of variables, whereas the general Sum-Product and Max-Product algorithms can handle more complex structural dependencies between the random variables.