Tagging with Hidden Markov Models

Michael Collins

1 Tagging Problems

In many NLP problems, we would like to model *pairs* of sequences. Part-of-speech (POS) tagging is perhaps the earliest, and most famous, example of this type of problem. In POS tagging our goal is to build a model whose input is a *sentence*, for example

the dog saw a cat

and whose output is a *tag sequence*, for example

D N V D N (1)

(here we use D for a determiner, N for noun, and V for verb). The tag sequence is the same length as the input sentence, and therefore specifies a single tag for each word in the sentence (in this example D for *the*, N for *dog*, V for *saw*, and so on).

We will use $x_1 \dots x_n$ to denote the input to the tagging model: we will often refer to this as a *sentence*. In the above example we have the length $n = 5$, and $x_1 =$ *the*, $x_2 =$ *dog*, $x_3 =$ *saw*, $x_4 =$ *the*, $x_5 =$ *cat*. We will use $y_1 \dots y_n$ to denote the output of the tagging model: we will often refer to this as the *state sequence* or *tag sequence*. In the above example we have $y_1 =$ D, $y_2 =$ N, $y_3 =$ V, and so on.

This type of problem, where the task is to map a sentence $x_1 \dots x_n$ to a tag sequence $y_1 \dots y_n$, is often referred to as a **sequence labeling problem**, or a **tagging problem**.

We will assume that we have a set of *training examples*, $(x^{(i)}, y^{(i)})$ for $i = 1 \dots m$, where each $x^{(i)}$ is a sentence $x_1^{(i)} \dots x_{n_i}^{(i)}$, and each $y^{(i)}$ is a tag sequence $y_1^{(i)} \dots y_{n_i}^{(i)}$ (we assume that the i'th example is of length n_i). Hence $x_j^{(i)}$ is the j'th word in the i'th training example, and $y_j^{(i)}$ is the tag for that word. Our task is to learn a function that maps sentences to tag sequences from these training examples.

2 Generative Models, and The Noisy Channel Model

Supervised problems in machine learning are defined as follows. We assume training examples $(x^{(1)}, y^{(1)}) \dots (x^{(m)}, y^{(m)})$, where each example consists of an input $x^{(i)}$ paired with a label $y^{(i)}$. We use $\mathcal{X}$ to refer to the set of possible inputs, and $\mathcal{Y}$ to refer to the set of possible labels. Our task is to learn a function $f : \mathcal{X} \rightarrow \mathcal{Y}$ that maps any input x to a label $f(x)$.

Many problems in natural language processing are supervised learning problems. For example, in tagging problems each $x^{(i)}$ would be a sequence of words $x_1^{(i)} \dots x_{n_i}^{(i)}$, and each $y^{(i)}$ would be a sequence of tags $y_1^{(i)} \dots y_{n_i}^{(i)}$ (we use n_i to refer to the length of the i'th training example). $\mathcal{X}$ would refer to the set of all sequences $x_1 \dots x_n$, and $\mathcal{Y}$ would be the set of all tag sequences $y_1 \dots y_n$. Our task would be to learn a function $f : \mathcal{X} \rightarrow \mathcal{Y}$ that maps sentences to tag sequences. In machine translation, each input x would be a sentence in the source language (e.g., Chinese), and each "label" would be a sentence in the target language (e.g., English). In speech recognition each input would be the recording of some utterance (perhaps pre-processed using a Fourier transform, for example), and each label is an entire sentence. Our task in all of these examples is to learn a function from inputs x to labels y, using our training examples $(x^{(i)}, y^{(i)})$ for $i = 1 \dots n$ as evidence.

One way to define the function $f(x)$ is through a *conditional model*. In this approach we define a model that defines the conditional probability

$$p(y|x)$$

for any x, y pair. The parameters of the model are estimated from the training examples. Given a new test example x, the output from the model is

$$f(x) = \arg\max_{y \in \mathcal{Y}} p(y|x)$$

Thus we simply take the most likely label y as the output from the model. If our model $p(y|x)$ is close to the true conditional distribution of labels given inputs, the function $f(x)$ will be close to optimal.

An alternative approach, which is often used in machine learning and natural language processing, is to define a *generative model*. Rather than directly estimating the conditional distribution $p(y|x)$, in generative models we instead model the *joint* probability

$$p(x, y)$$

over (x, y) pairs. The parameters of the model $p(x, y)$ are again estimated from the training examples $(x^{(i)}, y^{(i)})$ for $i = 1 \dots n$. In many cases we further decompose

the probability $p(x, y)$ as follows:

$$p(x, y) = p(y)p(x|y) \tag{2}$$

and then estimate the models for $p(y)$ and $p(x|y)$ separately. These two model components have the following interpretations:

- $p(y)$ is a *prior* probability distribution over labels y.
- $p(x|y)$ is the probability of generating the input x, given that the underlying label is y.

We will see that in many cases it is very convenient to decompose models in this way; for example, the classical approach to speech recognition is based on this type of decomposition.

Given a generative model, we can use Bayes rule to derive the conditional probability $p(y|x)$ for any (x, y) pair:

$$p(y|x) = \frac{p(y)p(x|y)}{p(x)}$$

where

$$p(x) = \sum_{y \in \mathcal{Y}} p(x, y) = \sum_{y \in \mathcal{Y}} p(y)p(x|y)$$

Thus the joint model is quite versatile, in that we can also derive the probabilities $p(x)$ and $p(y|x)$.

We use Bayes rule directly in applying the joint model to a new test example. Given an input x, the output of our model, $f(x)$, can be derived as follows:

$$\begin{aligned} f(x) &= \arg\max_y p(y|x) \\ &= \arg\max_y \frac{p(y)p(x|y)}{p(x)} \qquad (3) \\ &= \arg\max_y p(y)p(x|y) \qquad (4) \end{aligned}$$

Eq. 3 follows by Bayes rule. Eq. 4 follows because the denominator, $p(x)$, does not depend on y, and hence does not affect the $\arg\max$. This is convenient, because it means that we do not need to calculate $p(x)$, which can be an expensive operation.

Models that decompose a joint probability into into terms $p(y)$ and $p(x|y)$ are often called *noisy-channel* models. Intuitively, when we see a test example x, we assume that has been generated in two steps: first, a label y has been chosen with probability $p(y)$; second, the example x has been generated from the distribution

$p(x|y)$. The model $p(x|y)$ can be interpreted as a "channel" which takes a label y as its input, and corrupts it to produce x as its output. Our task is to find the most likely label y, given that we observe x.

In summary:

- Our task is to learn a function from inputs x to labels $y = f(x)$. We assume training examples $(x^{(i)}, y^{(i)})$ for $i = 1 \ldots n$.
- In the noisy channel approach, we use the training examples to estimate models $p(y)$ and $p(x|y)$. These models define a joint (generative) model

$$p(x, y) = p(y)p(x|y)$$

- Given a new test example x, we predict the label

$$f(x) = \arg\max_{y \in \mathcal{Y}} p(y)p(x|y)$$

Finding the output $f(x)$ for an input x is often referred to as the *decoding* problem.

3 Generative Tagging Models

We now see how generative models can be applied to the tagging problem. We assume that we have a finite vocabulary $\mathcal{V}$, for example $\mathcal{V}$ might be the set of words seen in English, e.g., $\mathcal{V} = \{$*the, dog, saw, cat, laughs*$, \ldots\}$. We use $\mathcal{K}$ to denote the set of possible tags; again, we assume that this set is finite. We then give the following definition:

Definition 1 (Generative Tagging Models) *Assume a finite set of words $\mathcal{V}$, and a finite set of tags $\mathcal{K}$. Define $\mathcal{S}$ to be the set of all sequence/tag-sequence pairs $\langle x_1 \ldots x_n, y_1 \ldots y_n \rangle$ such that $n \geq 0$, $x_i \in \mathcal{V}$ for $i = 1 \ldots n$ and $y_i \in \mathcal{K}$ for $i = 1 \ldots n$. A* generative tagging model *is then a function p such that:*

1. *For any $\langle x_1 \ldots x_n, y_1 \ldots y_n \rangle \in \mathcal{S}$,*

$$p(x_1 \ldots x_n, y_1 \ldots y_n) \geq 0$$

2. *In addition,*

$$\sum_{\langle x_1 \ldots x_n, y_1 \ldots y_n \rangle \in \mathcal{S}} p(x_1 \ldots x_n, y_1 \ldots y_n) = 1$$

Hence $p(x_1 \ldots x_n, y_1 \ldots y_n)$ is a probability distribution over pairs of sequences (i.e., a probability distribution over the set $\mathcal{S}$).

Given a generative tagging model, the function from sentences $x_1 \ldots x_n$ to tag sequences $y_1 \ldots y_n$ is defined as

$$f(x_1 \ldots x_n) = \arg\max_{y_1 \ldots y_n} p(x_1 \ldots x_n, y_1 \ldots y_n)$$

Thus for any input $x_1 \ldots x_n$, we take the highest probability tag sequence as the output from the model. □

Having introduced generative tagging models, there are three critical questions:

- How we define a generative tagging model $p(x_1 \ldots x_n, y_1 \ldots y_n)$?
- How do we estimate the parameters of the model from training examples?
- How do we efficiently find

$$\arg\max_{y_1 \ldots y_n} p(x_1 \ldots x_n, y_1 \ldots y_n)$$

  for any input $x_1 \ldots x_n$?

The next section describes how trigram hidden Markov models can be used to answer these three questions.

4 Trigram Hidden Markov Models (Trigram HMMs)

In this section we describe an important type of generative tagging model, a *trigram hidden Markov model*, describe how the parameters of the model can be estimated from training examples, and describe how the most likely sequence of tags can be found for any sentence.

4.1 Definition of Trigram HMMs

We now give a formal definition of trigram hidden Markov models (trigram HMMs). The next section shows how this model form is derived, and gives some intuition behind the model.

Definition 2 (Trigram Hidden Markov Model (Trigram HMM)) *A trigram HMM consists of a finite set $\mathcal{V}$ of possible words, and a finite set $\mathcal{K}$ of possible tags, together with the following parameters:*

- *A parameter*

$$q(s|u,v)$$

for any trigram (u,v,s) *such that* $s \in \mathcal{K} \cup \{\textit{STOP}\}$*, and* $u,v \in \mathcal{V} \cup \{*\}$*. The value for* $q(s|u,v)$ *can be interpreted as the probability of seeing the tag* s *immediately after the bigram of tags* (u,v)*.*

- *A parameter*

$$e(x|s)$$

for any $x \in \mathcal{V}$*,* $s \in \mathcal{K}$*. The value for* $e(x|s)$ *can be interpreted as the probability of seeing observation* x *paired with state* s*.*

Define $\mathcal{S}$ *to be the set of all sequence/tag-sequence pairs* $\langle x_1 \ldots x_n, y_1 \ldots y_{n+1} \rangle$ *such that* $n \geq 0$*,* $x_i \in \mathcal{V}$ *for* $i = 1 \ldots n$*,* $y_i \in \mathcal{K}$ *for* $i = 1 \ldots n$*, and* $y_{n+1} = \textit{STOP}$*.*

We then define the probability for any $\langle x_1 \ldots x_n, y_1 \ldots y_{n+1} \rangle \in \mathcal{S}$ *as*

$$p(x_1 \ldots x_n, y_1 \ldots y_{n+1}) = \prod_{i=1}^{n+1} q(y_i|y_{i-2}, y_{i-1}) \prod_{i=1}^{n} e(x_i|y_i)$$

where we have assumed that $y_0 = y_{-1} = *$*.* $\square$

As one example, if we have $n = 3$, $x_1 \ldots x_3$ equal to the sentence *the dog laughs*, and $y_1 \ldots y_4$ equal to the tag sequence `D N V STOP`, then

$$\begin{aligned} p(x_1 \ldots x_n, y_1 \ldots y_{n+1}) \quad = \quad & q(\texttt{D}|*,*) \times q(\texttt{N}|*,\texttt{D}) \times q(\texttt{V}|\texttt{D},\texttt{N}) \times q(\texttt{STOP}|\texttt{N},\texttt{V}) \\ & \times e(\textit{the}|\texttt{D}) \times e(\textit{dog}|\texttt{N}) \times e(\textit{laughs}|\texttt{V}) \end{aligned}$$

Note that this model form is a noisy-channel model. The quantity

$$q(\texttt{D}|*,*) \times q(\texttt{N}|*,\texttt{D}) \times q(\texttt{V}|\texttt{D},\texttt{N}) \times q(\texttt{STOP}|\texttt{N},\texttt{V})$$

is the prior probability of seeing the tag sequence `D N V STOP`, where we have used a second-order Markov model (a trigram model), very similar to the language models we derived in the previous lecture. The quantity

$$e(\textit{the}|\texttt{D}) \times e(\textit{dog}|\texttt{N}) \times e(\textit{laughs}|\texttt{V})$$

can be interpreted as the conditional probability $p(\textit{the dog laughs}|\texttt{D N V STOP})$: that is, the conditional probability $p(x|y)$ where x is the sentence *the dog laughs*, and y is the tag sequence `D N V STOP`.

4.2 Independence Assumptions in Trigram HMMs

We now describe how the form for trigram HMMs can be derived: in particular, we describe the independence assumptions that are made in the model. Consider a pair of sequences of random variables $X_1 \dots X_n$, and $Y_1 \dots Y_n$, where n is the length of the sequences. We assume that each X_i can take any value in a finite set $\mathcal{V}$ of *words*. For example, $\mathcal{V}$ might be a set of possible words in English, for example $\mathcal{V} = \{$*the, dog, saw, cat, laughs*$, \dots\}$. Each Y_i can take any value in a finite set $\mathcal{K}$ of possible *tags*. For example, $\mathcal{K}$ might be the set of possible part-of-speech tags for English, e.g. $\mathcal{K} = \{\texttt{D}, \texttt{N}, \texttt{V}, \dots\}$.

The length n is itself a random variable—it can vary across different sentences—but we will use a similar technique to the method used for modeling variable-length Markov processes (see the previous lecture notes).

Our task will be to model the joint probability

$$P(X_1 = x_1 \dots X_n = x_n, Y_1 = y_1 \dots Y_n = y_n)$$

for any observation sequence $x_1 \dots x_n$ paired with a state sequence $y_1 \dots y_n$, where each x_i is a member of $\mathcal{V}$, and each y_i is a member of $\mathcal{K}$.

We will find it convenient to define one additional random variable Y_{n+1}, which always takes the value STOP. This will play a similar role to the STOP symbol seen for variable-length Markov sequences, as described in the previous lecture notes.

The key idea in hidden Markov models is the following definition:

$$\begin{aligned} & P(X_1 = x_1 \dots X_n = x_n, Y_1 = y_1 \dots Y_{n+1} = y_{n+1}) \\ = & \prod_{i=1}^{n+1} P(Y_i = y_i | Y_{i-2} = y_{i-2}, Y_{i-1} = y_{i-1}) \prod_{i=1}^{n} P(X_i = x_i | Y_i = y_i) \end{aligned} \quad (5)$$

where we have assumed that $y_0 = y_{-1} =$ *, where * is a special start symbol.

Note the similarity to our definition of trigram HMMs. In trigram HMMs we have made the assumption that the joint probability factorizes as in Eq. 5, and in addition we have assumed that for any i, for any values of y_{i-2}, y_{i-1}, y_i,

$$P(Y_i = y_i | Y_{i-2} = y_{i-2}, Y_{i-1} = y_{i-1}) = q(y_i | y_{i-2}, y_{i-1})$$

and that for any value of i, for any values of x_i and y_i,

$$P(X_i = x_i | Y_i = y_i) = e(x_i | y_i)$$

Eq. 5 can be derived as follows. First, we can write

$$\begin{aligned} & P(X_1 = x_1 \dots X_n = x_n, Y_1 = y_1 \dots Y_{n+1} = y_{n+1}) \\ = & P(Y_1 = y_1 \dots Y_{n+1} = y_{n+1}) P(X_1 = x_1 \dots X_n = x_n | Y_1 = y_1 \dots Y_{n+1} = y_{n+1}) \end{aligned} \quad (6)$$

This step is exact, by the chain rule of probabilities. Thus we have decomposed the joint probability into two terms: first, the probability of choosing tag sequence $y_1 \ldots y_{n+1}$; second, the probability of choosing the word sequence $x_1 \ldots x_n$, conditioned on the choice of tag sequence. Note that this is exactly the same type of decomposition as seen in noisy channel models.

Now consider the probability of seeing the tag sequence $y_1 \ldots y_{n+1}$. We make independence assumptions as follows: we assume that for any sequence $y_1 \ldots y_{n+1}$,

$$P(Y_1 = y_1 \ldots Y_{n+1} = y_{n+1}) = \prod_{i=1}^{n+1} P(Y_i = y_i | Y_{i-2} = y_{i-2}, Y_{i-1} = y_{i-1})$$

That is, we have assumed that the sequence $Y_1 \ldots Y_{n+1}$ is a second-order Markov sequence, where each state depends only on the previous two states in the sequence.

Next, consider the probability of the word sequence $x_1 \ldots x_n$, conditioned on the choice of tag sequence, $y_1 \ldots y_{n+1}$. We make the following assumption:

$$\begin{aligned} & P(X_1 = x_1 \ldots X_n = x_n | Y_1 = y_1 \ldots Y_{n+1} = y_{n+1}) \\ = & \prod_{i=1}^{n} P(X_i = x_i | X_1 = x_1 \ldots X_{i-1} = x_{i-1}, Y_1 = y_1 \ldots Y_{n+1} = y_{n+1}) \\ = & \prod_{i=1}^{n} P(X_i = x_i | Y_i = y_i) \end{aligned} \tag{7}$$

The first step of this derivation is exact, by the chain rule. The second step involves an independence assumption, namely that for $i = 1 \ldots n$,

$$P(X_i = x_i | X_1 = x_1 \ldots X_{i-1} = x_{i-1}, Y_1 = y_1 \ldots Y_{n+1} = y_{n+1}) = P(X_i = x_i | Y_i = y_i)$$

Hence we have assumed that the value for the random variable X_i depends only on the value of Y_i. More formally, the value for X_i is conditionally independent of the previous observations $X_1 \ldots X_{i-1}$, and the other state values $Y_1 \ldots Y_{i-1}, Y_{i+1} \ldots Y_{n+1}$, given the value of Y_i.

One useful way of thinking of this model is to consider the following stochastic process, which generates sequence pairs $y_1 \ldots y_{n+1}, x_1 \ldots x_n$:

1. Initialize $i = 1$ and $y_0 = y_{-1} =$ *.
2. Generate y_i from the distribution
$$q(y_i | y_{i-2}, y_{i-1})$$
3. If $y_i =$ STOP then return $y_1 \ldots y_i, x_1 \ldots x_{i-1}$. Otherwise, generate x_i from the distribution
$$e(x_i | y_i),$$
set $i = i + 1$, and return to step 2.

4.3 Estimating the Parameters of a Trigram HMM

We will assume that we have access to some training data. The training data consists of a set of examples where each example is a sentence $x_1 \dots x_n$ paired with a tag sequence $y_1 \dots y_n$. Given this data, how do we estimate the parameters of the model? We will see that there is a simple and very intuitive answer to this question.

Define $c(u, v, s)$ to be the number of times the sequence of three states (u, v, s) is seen in training data: for example, $c(\text{V, D, N})$ would be the number of times the sequence of three tags V, D, N is seen in the training corpus. Similarly, define $c(u, v)$ to be the number of times the tag bigram (u, v) is seen. Define $c(s)$ to be the number of times that the state s is seen in the corpus. Finally, define $c(s \rightsquigarrow x)$ to be the number of times state s is seen paired sith observation x in the corpus: for example, $c(\text{N} \rightsquigarrow \text{dog})$ would be the number of times the word *dog* is seen paired with the tag N.

Given these definitions, the *maximum-likelihood* estimates are

$$q(s|u, v) = \frac{c(u, v, s)}{c(u, v)}$$

and

$$e(x|s) = \frac{c(s \rightsquigarrow x)}{c(s)}$$

For example, we would have the estimates

$$q(\texttt{N}|\texttt{V}, \texttt{D}) = \frac{c(\texttt{V}, \texttt{D}, \texttt{N})}{c(\texttt{V}, \texttt{D})}$$

and

$$e(\textit{dog}|\texttt{N}) = \frac{c(\texttt{N} \rightsquigarrow \textit{dog})}{c(\texttt{N})}$$

Thus estimating the parameters of the model is simple: we just read off counts from the training corpus, and then compute the maximum-likelihood estimates as described above.

4.4 Decoding with HMMs: the Viterbi Algorithm

We now turn to the problem of finding the most likely tag sequence for an input sentence $x_1 \dots x_n$. This is the problem of finding

$$\arg \max_{y_1 \dots y_{n+1}} p(x_1 \dots x_n, y_1 \dots y_{n+1})$$

where the $\arg\max$ is taken over all sequences $y_1 \dots y_{n+1}$ such that $y_i \in \mathcal{K}$ for $i = 1 \dots n$, and $y_{n+1} = \text{STOP}$. We assume that p again takes the form

$$p(x_1 \dots x_n, y_1 \dots y_{n+1}) = \prod_{i=1}^{n+1} q(y_i | y_{i-2}, y_{i-1}) \prod_{i=1}^{n} e(x_i | y_i) \tag{8}$$

Recall that we have assumed in this definition that $y_0 = y_{-1} = *$, and $y_{n+1} = \text{STOP}$.

The naive, brute force method would be to simply enumerate all possible tag sequences $y_1 \dots y_{n+1}$, score them under the function p, and take the highest scoring sequence. For example, given the input sentence

the dog barks

and assuming that the set of possible tags is $\mathcal{K} = \{\texttt{D}, \texttt{N}, \texttt{V}\}$, we would consider all possible tag sequences:

D	D	D	STOP
D	D	N	STOP
D	D	V	STOP
D	N	D	STOP
D	N	N	STOP
D	N	V	STOP
...			

and so on. There are $3^3 = 27$ possible sequences in this case.

For longer sentences, however, this method will be hopelessly inefficient. For an input sentence of length n, there are $|\mathcal{K}|^n$ possible tag sequences. The exponential growth with respect to the length n means that for any reasonable length sentence, brute-force search will not be tractable.

4.4.1 The Basic Algorithm

Instead, we will see that we can efficiently find the highest probability tag sequence, using a dynamic programming algorithm that is often called *the Viterbi algorithm*. The input to the algorithm is a sentence $x_1 \dots x_n$. Given this sentence, for any $k \in \{1 \dots n\}$, for any sequence $y_1 \dots y_k$ such that $y_i \in \mathcal{K}$ for $i = 1 \dots k$ we define the function

$$r(y_1 \dots y_k) = \prod_{i=1}^{k} q(y_i | y_{i-2}, y_{i-1}) \prod_{i=1}^{k} e(x_i | y_i) \tag{9}$$

This is simply a truncated version of the definition of p in Eq. 8, where we just consider the first k terms. In particular, note that

$$\begin{aligned} p(x_1 \dots x_n, y_1 \dots y_{n+1}) &= r(y_1 \dots y_n) \times q(y_{n+1}|y_{n-1}, y_n) \\ &= r(y_1 \dots y_n) \times q(\text{STOP}|y_{n-1}, y_n) \end{aligned} \quad (10)$$

Next, for any any $k \in \{1 \dots n\}$, for any $u \in \mathcal{K}$, $v \in \mathcal{K}$, define $S(k, u, v)$ to be the set of sequences $y_1 \dots y_k$ such that $y_{k-1} = u$, $y_k = v$, and $y_i \in \mathcal{K}$ for $i = 1 \dots k$. Thus $S(k, u, v)$ is the set of all tag sequences of length k, which end in the tag bigram (u, v). Define

$$\pi(k, u, v) = \max_{\langle y_1 \dots y_k \rangle \in S(k,u,v)} r(y_1 \dots y_k) \quad (11)$$

We now observe that we can calculate the $\pi(k, u, v)$ values for all (k, u, v) efficiently, as follows. First, as a base case define

$$\pi(0, *, *) = 1$$

and

$$\pi(0, u, v) = 0$$

if $u \neq *$ or $v \neq *$. These definitions just reflect the fact that we always assume that $y_0 = y_{-1} = *$.

Next, we give the recursive definition.

Proposition 1 *For any $k \in \{1 \dots n\}$, for any $u \in \mathcal{K}$ and $v \in \mathcal{K}$, we can use the following recursive definition:*

$$\pi(k, u, v) = \max_{w \in \mathcal{K}} (\pi(k-1, w, u) \times q(v|w, u) \times e(x_k|v)) \quad (12)$$

This definition is recursive because the definition makes use of the $\pi(k-1, w, v)$ values computed for shorter sequences. This definition will be key to our dynamic programming algorithm.

How can we justify this recurrence? Recall that $\pi(k, u, v)$ is the highest probability for any sequence $y_1 \dots y_k$ ending in the bigram (u, v). Any such sequence must have $y_{k-2} = w$ for some state w. The highest probability for any sequence of length $k-1$ ending in the bigram (w, u) is $\pi(k-1, w, u)$, hence the highest probability for any sequence of length k ending in the trigram (w, u, v) must be

$$\pi(k-1, w, u) \times q(v|w, u) \times e(x_i|v)$$

In Eq. 12 we simply search over all possible values for w, and return the maximum.

Our second claim is the following:

Input: a sentence $x_1 \dots x_n$, parameters $q(s|u,v)$ and $e(x|s)$.
Initialization: Set $\pi(0,*,*) = 1$, and $\pi(0,u,v) = 0$ for all (u,v) such that $u \neq *$ or $v \neq *$.
Algorithm:

- For $k = 1 \dots n$,
  - For $u \in \mathcal{K}$, $v \in \mathcal{K}$,

$$\pi(k,u,v) = \max_{w \in \mathcal{K}} (\pi(k-1,w,u) \times q(v|w,u) \times e(x_k|v))$$

- **Return** $\max_{u \in \mathcal{K}, v \in \mathcal{K}} (\pi(n,u,v) \times q(\text{STOP}|u,v))$

Figure 1: The basic Viterbi Algorithm.

Proposition 2

$$\max_{y_1 \dots y_{n+1}} p(x_1 \dots x_n, y_1 \dots y_{n+1}) = \max_{u \in \mathcal{K}, v \in \mathcal{K}} (\pi(n,u,v) \times q(\textit{STOP}|u,v)) \quad (13)$$

This follows directly from the identity in Eq. 10.

Figure 1 shows an algorithm that puts these ideas together. The algorithm takes a sentence $x_1 \dots x_n$ as input, and returns

$$\max_{y_1 \dots y_{n+1}} p(x_1 \dots x_n, y_1 \dots y_{n+1})$$

as its output. The algorithm first fills in the $\pi(k,u,v)$ values in using the recursive definition. It then uses the identity in Eq. 13 to calculate the highest probability for any sequence.

The running time for the algorithm is $O(n|\mathcal{K}|^3)$, hence it is linear in the length of the sequence, and cubic in the number of tags.

4.4.2 The Viterbi Algorithm with Backpointers

The algorithm we have just described takes a sentence $x_1 \dots x_n$ as input, and returns

$$\max_{y_1 \dots y_{n+1}} p(x_1 \dots x_n, y_1 \dots y_{n+1})$$

as its output. However we'd really like an algorithm that returned the highest probability sequence, that is, an algorithm that returns

$$\arg \max_{y_1 \dots y_{n+1}} p(x_1 \dots x_n, y_1 \dots y_{n+1})$$

Input: a sentence $x_1 \ldots x_n$, parameters $q(s|u,v)$ and $e(x|s)$.
Initialization: Set $\pi(0,*,*) = 1$, and $\pi(0,u,v) = 0$ for all (u,v) such that $u \neq *$ or $v \neq *$.
Algorithm:

- For $k = 1 \ldots n$,
  - For $u \in \mathcal{K}$, $v \in \mathcal{K}$,

$$\begin{aligned} \pi(k,u,v) &= \max_{w \in \mathcal{K}} (\pi(k-1,w,u) \times q(v|w,u) \times e(x_k|v)) \\ bp(k,u,v) &= \arg\max_{w \in \mathcal{K}} (\pi(k-1,w,u) \times q(v|w,u) \times e(x_k|v)) \end{aligned}$$

- Set $(y_{n-1}, y_n) = \arg\max_{(u,v)} (\pi(n,u,v) \times q(\text{STOP}|u,v))$
- For $k = (n-2) \ldots 1$,

$$y_k = bp(k+2, y_{k+1}, y_{k+2})$$

- **Return** the tag sequence $y_1 \ldots y_n$

Figure 2: The Viterbi Algorithm with backpointers.

for any input sentence $x_1 \ldots x_n$.

Figure 2 shows a modified algorithm that achieves this goal. The key step is to store backpointer values $bp(k,u,v)$ at each step, which record the previous state w which leads to the highest scoring sequence ending in (u,v) at position k (the use of backpointers such as these is very common in dynamic programming methods). At the end of the algorithm, we unravel the backpointers to find the highest probability sequence, and then return this sequence. The algorithm again runs in $O(n|\mathcal{K}|^3)$ time.

5 Summary

We've covered a fair amount of material in this note, but the end result is fairly straightforward: we have derived a complete method for learning a tagger from a training corpus, and for applying it to new sentences. The main points were as follows:

- A trigram HMM has parameters $q(s|u,v)$ and $e(x|s)$, and defines the joint

probability of any sentence $x_1 \dots x_n$ paired with a tag sequence $y_1 \dots y_{n+1}$ (where $y_{n+1} = \text{STOP}$) as

$$p(x_1 \dots x_n, y_1 \dots y_{n+1}) = \prod_{i=1}^{n+1} q(y_i | y_{i-2}, y_{i-1}) \prod_{i=1}^{n} e(x_i | y_i)$$

- Given a training corpus from which we can derive counts, the maximum-likelihood estimates for the parameters are

$$q(s|u,v) = \frac{c(u,v,s)}{c(u,v)}$$

and

$$e(x|s) = \frac{c(s \rightsquigarrow x)}{c(s)}$$

- Given a new sentence $x_1 \dots x_n$, and parameters q and e that we have estimated from a training corpus, we can find the highest probability tag sequence for $x_1 \dots x_n$ using the algorithm in figure 2 (the Viterbi algorithm).