Lecture 14: A tutorial on hidden Markov models and selected applications in speech recognition, part 1

Mark Hasegawa-Johnson

ECE 537, Fall 2022

Outline

Notation: Inputs and Outputs

- The **observation sequence** is a sequence of short-time spectra, or other observation vectors, $O = [\vec{o}_1, \ldots, \vec{o}_T]$.
- The **model parameters**, λ_y, are a set of numbers that describe the probability of observing O, given that word y was produced.
- **Isolated word recognition** is the problem of figuring out which word has the maximum probability, i.e., finding

$$\underset{y}{\operatorname{argmax}}\, p(O|\lambda_y)p(y)$$

Review: Automatic Speech Recognition

Remember that Velichko & Zagoruyko broke down the problem of ASR into two key subproblems:

- **Variable acoustics:** V&Z solved this problem by calculating Euclidean distance using a perceptually-motivated feature vector.
- **Variable duration:** V&Z solved this problem using dynamic time warping.

Hidden Markov Model

The hidden Markov model (HMM) solves the same problems, in a more scalable fashion:

- **Variable acoustics:** Instead of storing training examples, the HMM stores a model, λ_y, specifying the probability density function of the short-time spectrum, $\vec{o}_t$, given the index of the speech sound being produced at that instant, q_t:

$$p(\vec{o}_t|q_t, \lambda_y)$$

- **Variable duration:** The HMM solves this problem by imagining that each word is composed of a sequence of speech sounds, or "states:" $Q = [q_1, \ldots, q_T]$, and that the likelihood of the word is

$$p(O|\lambda_y) = \sum_Q p(Q, O|\lambda_y)$$

HMM: Key Concepts

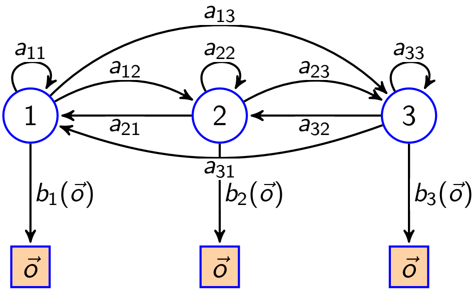
An HMM is a "generative model," meaning that it models the joint probability $p(Q, O|\lambda)$ using a model of the way in which those data might have been generated. An HMM pretends the following generative process:

1. Start in state $q_t = i$ with pmf $\pi_i = p(q_1 = i)$.
2. Generate an observation, $\vec{o}$, with pdf $b_i(\vec{o}) = p(\vec{o}|q_t = i)$.
3. Transition to a new state, $q_{t+1} = j$, according to pmf $a_{ij} = p(q_{t+1} = j|q_t = i)$.
4. Repeat.

The model parameters that define any particular word are thus

$$\lambda_y = \{\Pi, A, B\}$$

HMM: Finite State Diagram



1. Start in state $q_t = i$, for some $1 \leq i \leq N$.
2. Generate an observation, $\vec{o}$, with pdf $b_i(\vec{o})$.
3. Transition to a new state, $q_{t+1} = j$, according to pmf a_{ij}.
4. Repeat steps #2 and #3, T times each.

Notation: Model Parameters

Solving an HMM is possible if you **carefully keep track of notation**. Here's standard notation for the parameters:

- $\pi_i = p(q_1 = i)$ is called the **initial state probability**. Let N be the number of different states, so that $1 \le i \le N$.
- $a_{ij} = p(q_t = j | q_{t-1} = i)$ is called the **transition probability**, $1 \le i, j \le N$.
- $b_j(\vec{o}) = p(\vec{o}_t = \vec{o} | q_t = j)$ is called the **observation probability**. It is usually estimated by a neural network, though Gaussians, GMMs, and even lookup tables are possible.
- λ is the complete set of **model parameters**, including all the π_i's and a_{ij}'s, and the Gaussian, GMM, or neural net parameters necessary to compute $b_j(\vec{o})$.

The Three Problems for an HMM

1. **Recognition:** Given two different HMMs, λ_1 and λ_2, and an observation sequence O. Which HMM was more likely to have produced O? In other words, is $p(O|\lambda_1) > p(O|\lambda_2)$?
2. **Segmentation:** What is $p(q_t = i|O, \lambda)$?
3. **Training:** Given an initial HMM λ, and an observation sequence O, can we find $\bar{\lambda}$ such that $p(O|\bar{\lambda}) > p(O|\lambda)$?

Outline

The HMM Recognition Problem

- Given
  - $O = [\vec{o}_1, \ldots, \vec{o}_T]$ and
  - $\lambda = \{\pi_i, a_{ij}, b_j(\vec{o}) \forall i, j\}$,

  what is $p(O|\lambda)$?
- Let's solve a simpler problem first:
- Given
  - $O = [\vec{o}_1, \ldots, \vec{o}_T]$ and
  - $Q = [q_1, \ldots, q_T]$ and
  - $\lambda = \{\pi_i, a_{ij}, b_j(\vec{o}) \forall i, j\}$,

  what is $p(O, Q|\lambda)$?

Joint Probability of State Sequence and Observation Sequence

The joint probability of the state sequence and the observation sequence is calculated iteratively, from beginning to end:

- The probability that $q_1 = q_1$ is π_{q_1}.
- Given q_1, the probability of $\vec{o}_1$ is $b_{q_1}(\vec{o}_1)$.
- Given q_1, the probability of q_2 is $a_{q_1 q_2}$.
- . . . and so on. . .

$$p(Q, O|\lambda) = \pi_{q_1} b_{q_1}(\vec{o}_1) \prod_{t=2}^{T} a_{q_{t-1} q_t} b_{q_t}(\vec{o}_t)$$

Probability of the Observation Sequence

The probability of the observation sequence, alone, is somewhat harder, because we have to solve this sum:

$$
\begin{aligned}
p(O|\lambda) &= \sum_{Q} p(Q, O|\lambda) \\
&= \sum_{q_T=1}^{N} \cdots \sum_{q_1=1}^{N} p(Q, O|\lambda)
\end{aligned}
$$

On the face of it, this calculation seems to have complexity $\mathcal{O}\left\{N^T\right\}$. So for a very small 100-frame utterance, with only 10 states, we have a complexity of $\mathcal{O}\left\{10^{100}\right\}$ =one google.

The Forward Algorithm

The solution is to use a kind of dynamic programming algorithm, called "the forward algorithm." The forward probability is defined as follows:

$$\alpha_t(i) \equiv p(\vec{o}_1, \ldots, \vec{o}_t, q_t = i|\lambda)$$

Obviously, if we can find $\alpha_t(i)$ for all i and all t, we will have solved the recognition problem, because

$$\begin{aligned} p(O|\lambda) &= p(\vec{o}_1, \ldots, \vec{o}_T|\lambda) \\ &= \sum_{i=1}^{N} p(\vec{o}_1, \ldots, \vec{o}_T, q_T = i|\lambda) \\ &= \sum_{i=1}^{N} \alpha_T(i) \end{aligned}$$

The Forward Algorithm

So, working with the definition $\alpha_t(i) \equiv p(\vec{o}_1, \ldots, \vec{o}_t, q_t = i|\lambda)$, let's see how we can actually calculate $\alpha_t(i)$.

1. **Initialize:**

$$\begin{aligned}
\alpha_1(i) &= p(q_1 = i, \vec{o}_1|\lambda) \\
&= p(q_1 = i|\lambda)p(\vec{o}_1|q_1 = i, \lambda) \\
&= \pi_i b_i(\vec{o}_1)
\end{aligned}$$

The Forward Algorithm

Definition: $\alpha_t(i) \equiv p(\vec{o}_1, \ldots, \vec{o}_t, q_t = i|\lambda)$.

1. **Initialize:**

$$\alpha_1(i) = \pi_i b_i(\vec{o}_1), \quad 1 \le i \le N$$

2. **Iterate:**

$$\begin{aligned}
\alpha_t(j) &= p(\vec{o}_1, \ldots, \vec{o}_t, q_t = j|\lambda) \\
&= \sum_{i=1}^{N} p(\vec{o}_1, \ldots, \vec{o}_{t-1}, q_{t-1} = i) p(q_t = j|q_{t-1} = i) p(\vec{o}_t|q_t = j) \\
&= \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(\vec{o}_t)
\end{aligned}$$

The Forward Algorithm

So, working with the definition $\alpha_t(i) \equiv p(\vec{o}_1, \ldots, \vec{o}_t, q_t = i|\lambda)$, let's see how we can actually calculate $\alpha_t(i)$.

1. **Initialize:**

$$\alpha_1(i) = \pi_i b_i(\vec{o}_1), \quad 1 \le i \le N$$

2. **Iterate:**

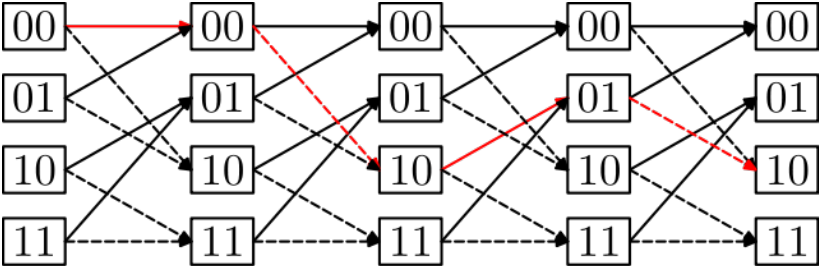
$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(\vec{o}_t), \;\; 1 \le j \le N, \; 2 \le t \le T$$

3. **Terminate:**

$$p(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$$

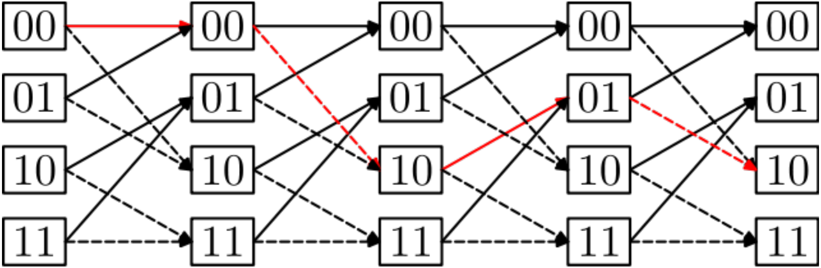
Visualizing the Forward Algorithm using a Trellis

One way to think about the forward algorithm is by way of a **trellis**. A trellis is a matrix in which each time step is a column, and each row shows a different state. For example, here's a trellis with $N = 4$ states, and $T = 5$ frames:



Public domain image by Qef, 2009

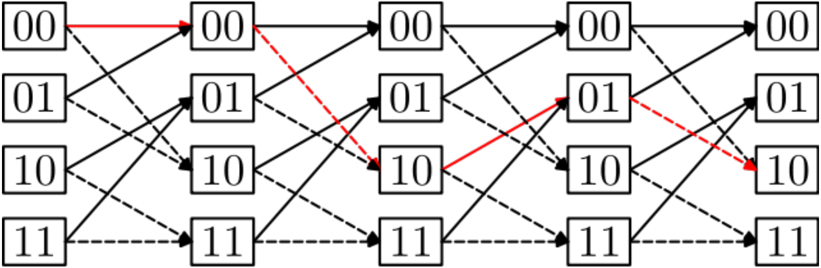
Visualizing the Forward Algorithm using a Trellis



Using a trellis, the **initialize** step computes probabilities for the first column of the trellis:

$$\alpha_1(i) = \pi_i b_i(\vec{o}_1), \quad 1 \leq i \leq N$$

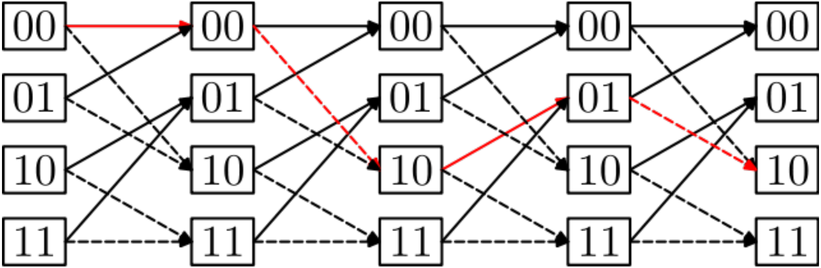
Visualizing the Forward Algorithm using a Trellis



The **iterate** step then computes the probabilities in the t^{th} column by adding up the probabilities in the $(t-1)^{\text{st}}$ column, each multiplied by the corresponding transition probability:

$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(\vec{o}_t), \quad 1 \le j \le N, \; 2 \le t \le T$$

Visualizing the Forward Algorithm using a Trellis



The **terminate** step then computes the likelihood of the model by adding the probabilities in the last column:

$$p(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$$

The Forward Algorithm: Computational Complexity

Most of the computational complexity is in this step:

- **Iterate:**

$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(\vec{o}_t), \quad 1 \le i, j \le N, \; 2 \le t \le T$$

Its complexity is:

- For each of $T-1$ time steps, $2 \le t \le T$,...
- we need to calculate N different alpha-variables, $\alpha_t(j)$, for $1 \le j \le N$,...
- each of which requires a summation with N terms.

So the total complexity is $\mathcal{O}\left\{TN^2\right\}$. For example, with $N = 10$ and $T = 100$, the complexity is only $TN^2 = 10,000$ multiplies (much, much less than N^T!!)

Outline

The Segmentation Problem

There are different ways to define the segmentation problem. Let's define it this way:

- We want to find the most likely state, $q_t = i$, at time $t, \ldots$
- given knowledge of the *entire* sequence $O = [\vec{o}_1, \ldots, \vec{o}_T]$, not just the current observation. So for example, we don't want to recognize state i at time t if the surrounding observations, $\vec{o}_{t-1}$ and $\vec{o}_{t+1}$, make it obvious that this choice is impossible. Also,...
- given knowledge of the HMM that produced this sequence, λ.

In other words, we want to find the **state posterior probability**, $p(q_t = i|O, \lambda)$. Let's define some more notation for the state posterior probability, let's call it

$$\gamma_t(i) = p(q_t = i|O, \lambda)$$

Use Bayes' Rule

Suppose we already knew the **joint probability,** $p(O, q_t = i|\lambda)$. Then we could find the state posterior using Bayes' rule:

$$\gamma_t(i) = p(q_t = i|O, \lambda) = \frac{p(O, q_t = i|\lambda)}{\sum_{j=1}^{N} p(O, q_t = j|\lambda)}$$

Use the Forward Algorithm

Let's expand this:

$$p(O, q_t = i|\lambda) = p(q_t = i, \vec{o}_1, \ldots, \vec{o}_T|\lambda)$$

We already know about half of that:
$\alpha_t(i) = p(q_t = i, \vec{o}_1, \ldots, \vec{o}_t|\lambda)$. We're only missing this part:

$$p(O, q_t = i|\lambda) = \alpha_t(i)p(\vec{o}_{t+1}, \ldots, \vec{o}_T|q_t = i, \lambda)$$

Again, let's try the trick of "solve the problem by inventing new notation." Let's define

$$\beta_t(i) \equiv p(\vec{o}_{t+1}, \ldots, \vec{o}_T|q_t = i, \lambda)$$

The Backward Algorithm

Now let's use the definition $\beta_t(i) \equiv p(\vec{o}_{t+1}, \ldots, \vec{o}_T | q_t = i, \lambda)$, and see how we can compute that.

1. **Initialize:**

$$\beta_T(i) = 1, \quad 1 \leq i \leq N$$

   This might not seem immediately obvious, but think about it. Given that there are no more $\vec{o}$ vectors after time T, what is the probability that there are no more $\vec{o}$ vectors after time T? Well, 1, obviously.

The Backward Algorithm

Now let's use the definition $\beta_t(i) \equiv p(\vec{o}_{t+1}, \ldots, \vec{o}_T | q_t = i, \lambda)$, and see how we can compute that.

1. **Initialize:**

$$\beta_T(i) = 1, \quad 1 \leq i \leq N$$

2. **Iterate:**

$$\begin{aligned}
\beta_t(i) &= p(\vec{o}_{t+1}, \ldots, \vec{o}_T | q_t = i, \lambda) \\
&= \sum_{j=1}^{N} p(q_{t+1} = j | q_t = i) p(\vec{o}_{t+1} | q_{t+1} = j) p(\vec{o}_{t+2}, \ldots, \vec{o}_T | q_{t+1} = j) \\
&= \sum_{j=1}^{N} a_{ij} b_j(\vec{o}_{t+1}) \beta_{t+1}(j)
\end{aligned}$$

The Backward Algorithm

Now let's use the definition $\beta_t(i) \equiv p(\vec{o}_{t+1}, \ldots, \vec{o}_T | q_t = i, \lambda)$, and see how we can compute that.

1. **Initialize:**

$$\beta_T(i) = 1, \quad 1 \leq i \leq N$$

2. **Iterate:**

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(\vec{o}_{t+1}) \beta_{t+1}(j), \quad 1 \leq i \leq N, \; 1 \leq t \leq T-1$$

3. **Terminate:**

$$p(O|\lambda) = \sum_{i=1}^{N} \pi_i b_i(\vec{o}_1) \beta_1(i)$$

The Backward Algorithm: Computational Complexity

Most of the computational complexity is in this step:

- **Iterate:**

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(\vec{o}_{t+1}) \beta_{t+1}(j), \quad 1 \leq i \leq N, \; 2 \leq t \leq T$$

Its complexity is:

- For each of $T-1$ time steps, $1 \leq t \leq T-1, \ldots$
- we need to calculate N different beta-variables, $\beta_t(i)$, for $1 \leq i \leq N, \ldots$
- each of which requires a summation with N terms.

So the total complexity is $\mathcal{O}\left\{TN^2\right\}$.

Use Bayes' Rule

The segmentation probability is then

$$\begin{aligned}
\gamma_t(i) &= \frac{p(O, q_t = i|\lambda)}{\sum_{k=1}^{N} p(O, q_t = k|\lambda)} \\
&= \frac{p(\vec{o}_1, \ldots, \vec{o}_t, q_t = i|\lambda) p(\vec{o}_{t+1}, \ldots, \vec{o}_T | q_t = i, \lambda)}{\sum_{k=1}^{N} p(\vec{o}_1, \ldots, \vec{o}_t, q_t = k|\lambda) p(\vec{o}_{t+1}, \ldots, \vec{o}_T | q_t = k, \lambda)} \\
&= \frac{\alpha_t(i)\beta_t(i)}{\sum_{k=1}^{N} \alpha_t(k)\beta_t(k)}
\end{aligned}$$

Segmentation: The Backward Algorithm

In summary, we now have three new probabilities, all of which can be computed in $\mathcal{O}\left\{TN^2\right\}$ time:

1. **The Backward Probability:**

$$\beta_t(i) = p(\vec{o}_{t+1}, \ldots, \vec{o}_T | q_t = i, \lambda)$$

2. **The State Posterior:**

$$\gamma_t(i) = p(q_t = i | O, \lambda) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{k=1}^N \alpha_t(k)\beta_t(k)}$$

3. **The Segment Posterior:**

$$\xi_t(i,j) = p(q_t = i, q_{t+1} = j | O, \lambda)$$
$$= \frac{\alpha_t(i) a_{ij} b_j(\vec{o}_{t+1})\beta_{t+1}(j)}{\sum_{k=1}^N \sum_{\ell=1}^N \alpha_t(k) a_{k\ell} b_\ell(\vec{o}_{t+1})\beta_{t+1}(\ell)}$$

Outline

Segmentation Problem: A Different Version

- Using the forward-backward algorithm, we can find $p(q_t = i|O, \lambda)$.
- Suppose we want to know **all** of the states, $Q = [q_1, \ldots, q_T]$. Notice that

$$p(q_1, \ldots, q_T|O, \Lambda) \neq \prod_{t=1}^{T} p(q_t|O, \Lambda)$$

  For example, the maximizer of the RHS might be an impossible state sequence: $q_t = i$ and $q_{t+1} = j$ might be individually likely, but $p(q_{t+1} = j|q_t = i)$ might be 0!
- In order to find $p(q_1, \ldots, q_T|O, \lambda)$, we need a different algorithm.

Viterbi Algorithm

Since the method of "solve a problem by defining new variables" is working so well for us, let's try it again. Define

$$\delta_t(i) \equiv \max_{q_1,\ldots,q_{t-1}} p(q_1, \vec{o}_1, \ldots, q_t = i, \vec{o}_t|\lambda)$$

$$\psi_t(i) \equiv \underset{q_{t-1}}{\mathrm{argmax}} \max_{q_1,\ldots,q_{t-2}} p(q_1, \vec{o}_1, \ldots, q_t = i, \vec{o}_t|\lambda)$$

The second term, $\psi_t(i)$, is called a **back-pointer**. It tells us:

- If you find yourself in state i at time t,
- . . . what was the most likely previuos state, q_{t-1}?

The Viterbi Algorithm

So, working with the definition $\delta_t(i) \equiv \max_{q_1,\ldots,q_{t-1}} p(q_1, \vec{o}_1, \ldots, q_t = i, \vec{o}_t|\lambda)$, let's see how we can actually calculate $\delta_t(i)$.

1. **Initialize:**

$$
\begin{aligned}
\delta_1(i) &= p(q_1 = i, \vec{o}_1|\lambda) \\
&= p(q_1 = i|\lambda)p(\vec{o}_1|q_1 = i, \lambda) \\
&= \pi_i b_i(\vec{o}_1) \\
\psi_t(i) &= \text{undefined}
\end{aligned}
$$

The Viterbi Algorithm

$$\delta_t(i) \equiv \max_{q_1,\ldots,q_{t-1}} p(q_1, \vec{o}_1, \ldots, q_t = i, \vec{o}_t|\lambda)$$

1. **Initialize:**

$$\delta_1(i) = \pi_i b_i(\vec{o}_1), \quad 1 \leq i \leq N$$

2. **Iterate:**

$$\begin{aligned}
\delta_t(j) &= \max_{q_{t-t}} \left(\max_{q_1,\ldots,q_{t-1}} (p(q_1, \vec{o}_1, \ldots, q_{t-1}, \vec{o}_{t-1}|\lambda) \times \right. \\
&\qquad p(q_t = j|q_{t-1} = i)p(\vec{o}_t|q_t = j))) \\
&= \max_{i=1}^{N} \delta_{t-1}(i) a_{ij} b_j(\vec{o}_t) \\
\psi_t(j) &= \operatorname*{argmax}_{i=1}^{N} \delta_{t-1}(i) a_{ij} b_j(\vec{o}_t)
\end{aligned}$$

The Viterbi Algorithm

$$\delta_t(i) \equiv \max_{q_1,\ldots,q_{t-1}} p(q_1, \vec{o}_1, \ldots, q_t = i, \vec{o}_t | \lambda)$$

1. **Initialize:**
$$\delta_1(i) = \pi_i b_i(\vec{o}_1), \quad 1 \le i \le N$$

2. **Iterate:**
$$\delta_t(j) = \max_{i=1}^{N} \delta_{t-1}(i) a_{ij} b_j(\vec{o}_t), \;\; 1 \le j \le N, \;\; 2 \le t \le T$$
$$\psi_t(j) = \operatorname*{argmax}_{i=1}^{N} \delta_{t-1}(i) a_{ij} b_j(\vec{o}_t), \;\; 1 \le j \le N, \;\; 2 \le t \le T$$

3. **Terminate:**
$$\max_{Q} p(O, Q|\lambda) = \max_{i=1}^{N} \delta_T(i)$$

Back-Tracing

Now that we have $\max_Q p(O, Q|\lambda)$, now we need to find

$$[q_1^*, \ldots, q_T^*] \equiv \underset{Q}{\operatorname{argmax}}\, p(O, Q|\lambda)$$

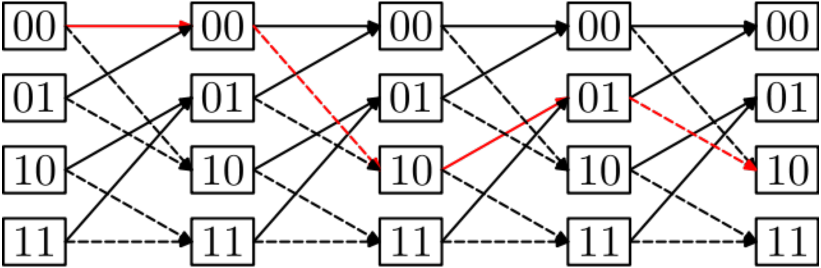
The algorithm is called "back-tracing." We start by finding the most likely final state:

$$q_T^* = \underset{i}{\operatorname{argmax}}\, \delta_T(i)$$

...and then we just follow the backpointers from there:

$$q_{t-1}^* = \psi_t(q_t^*), \quad T \geq t \geq 2$$

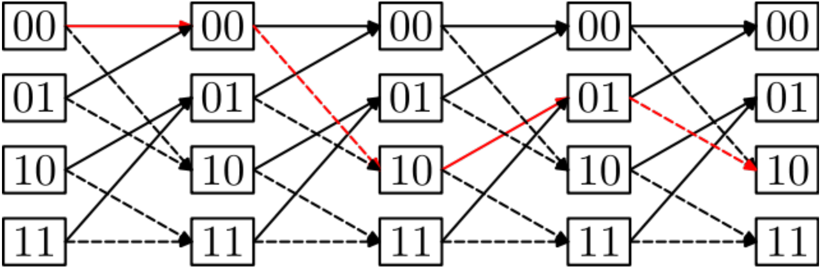
Visualizing the Viterbi Algorithm using a Trellis



Using a trellis, the **initialize** step computes probabilities for the first column of the trellis:

$$\delta_1(i) = \pi_i b_i(\vec{o}_1), \quad 1 \le i \le N$$

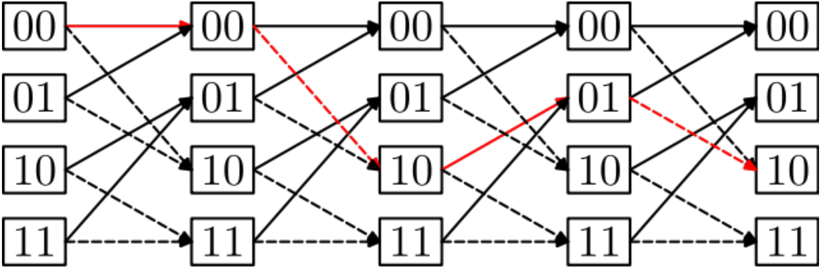
Visualizing the Viterbi Algorithm using a Trellis



The **iterate** step then computes the probability of the **best path** to **each state** in the t^{th} column:

$$\delta_t(j) = \max_{i=1}^{N} \delta_{t-1}(i) a_{ij} b_j(\vec{o}_t), \quad 1 \leq j \leq N, \; 2 \leq t \leq T$$

Visualizing the Viterbi Algorithm using a Trellis



Back-tracing then finds the most likely final state, and traces backward, from there, to find the most likely sequence over all:

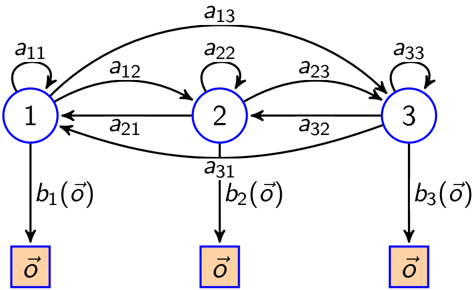
$$q_T^* = \underset{i}{\operatorname{argmax}} \, \delta_T(i)$$

$$q_{t-1}^* = \psi_t(q_t^*), \quad T \geq t \geq 2$$

Outline

1. Hidden Markov Models
2. Recognition: the Forward Algorithm
3. Segmentation: the Backward Algorithm
4. Segmentation: the Viterbi Algorithm
5. Summary

Hidden Markov Model



1. Start in state $q_t = i$ with pmf π_i.
2. Generate an observation, $\vec{o}$, with pdf $b_i(\vec{o})$.
3. Transition to a new state, $q_{t+1} = j$, according to pmf a_{ij}.
4. Repeat.

The Forward Algorithm

Definition: $\alpha_t(i) \equiv p(\vec{o}_1, \ldots, \vec{o}_t, q_t = i|\lambda)$. Computation:

1. **Initialize:**

$$\alpha_1(i) = \pi_i b_i(\vec{o}_1), \quad 1 \le i \le N$$

2. **Iterate:**

$$\alpha_t(j) = \sum_{i=1}^{N} \alpha_{t-1}(i) a_{ij} b_j(\vec{o}_t), \quad 1 \le j \le N, \; 2 \le t \le T$$

3. **Terminate:**

$$p(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$$

The Backward Algorithm

Definition: $\beta_t(i) \equiv p(\vec{o}_{t+1}, \ldots, \vec{o}_T | q_t = i, \lambda)$. Computation:

1. **Initialize:**
$$\beta_T(i) = 1, \quad 1 \le i \le N$$
2. **Iterate:**
$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(\vec{o}_{t+1}) \beta_{t+1}(j), \quad 1 \le i \le N, \; 1 \le t \le T-1$$
3. **Terminate:**
$$p(O|\lambda) = \sum_{i=1}^{N} \pi_i b_i(\vec{o}_1) \beta_1(i)$$

The Viterbi Algorithm

1. **Initialize:**

$$\delta_1(i) = \pi_i b_i(\vec{o}_1), \quad 1 \leq i \leq N$$

2. **Iterate:**

$$\delta_t(j) = \max_{i=1}^{N} \delta_{t-1}(i) a_{ij} b_j(\vec{o}_t), \quad 1 \leq j \leq N, \; 2 \leq t \leq T$$

$$\psi_t(j) = \operatorname*{argmax}_{i=1}^{N} \delta_{t-1}(i) a_{ij} b_j(\vec{o}_t), \quad 1 \leq j \leq N, \; 2 \leq t \leq T$$

3. **Back-trace:**

$$q_T^* = \operatorname*{argmax}_{i} \delta_T(i)$$

$$q_{t-1}^* = \psi_t(q_t^*), \quad T \geq t \geq 2$$