Hidden Markov Models
Implementing the forward-, backward- and Viterbi-algorithms using log-space and scaling
The Viterbi Algorithm

\( \omega(z_n) \) is the probability of the most likely sequence of states \( z_1, \ldots, z_n \) generating the observations \( x_1, \ldots, x_n \)

\[
\omega(z_n) = \max_{z_1, \ldots, z_{n-1}} p(x_1, \ldots, x_n, z_1, \ldots, z_n)
\]

Recursion:

\[
\omega(z_n) = p(x_n | z_n) \max_{z_{n-1}} \omega(z_{n-1}) p(z_n | z_{n-1})
\]

Basis:

\[
\omega(z_1) = p(x_1, z_1) = p(z_1)p(x_1 | z_1) = \prod_{k=1}^{K} \left\{ \pi_k p(x_1 | \phi_k) \right\}^\{z_{1k}\}
\]

Takes time \( O(K^2N) \) and space \( O(KN) \) using memorization
The Viterbi Algorithm

\( \omega(z_n) \) is the probability of the most likely sequence of states \( z_1, \ldots, z_n \) generating the observations \( x_1, \ldots, x_n \).

Recursion:

\[
\omega(z_n) = p(x_n | z_n) \max_{z_{n-1}} \omega(z_{n-1}) p(z_n | z_{n-1})
\]

Basis:

\[
\omega(z_1) = p(x_1, z_1) = p(z_1) p(x_1 | z_1) = \prod_{k=1}^{K} \{ \pi_k p(x_1 | \phi_k) \}^{z_1^k}
\]

Takes time \( O(K^2N) \) and space \( O(KN) \) using memorization.
The Viterbi Algorithm

ω(z_n) is the probability of the most likely sequence of states z_1,...,z_n generating the observations x_1,...,x_n.

ω(z_n) ≡ \max_{z_1,...,z_{n-1}} ω(z_n|z_{n-1})

Recursion:

ω(z_n) = p(x_n|z_n) \max_{z_{n-1}} ω(z_{n-1})p(z_n|z_{n-1})

Basis:

Problem: The values ω(z_{nk}) can come very close to zero, by multiplying them we potentially exceed the precision of double precision floating points

Takes time O(K^2N) and space O(KN) using memorization
The Viterbi Algorithm

$\omega(z_n)$ is the probability of the most likely sequence of states $z_1, \ldots, z_n$ generating the observations $x_1, \ldots, x_n$.

Recursion:

$$\omega(z_n) = p(x_n | z_n) \max_{z_{n-1}} \omega(z_{n-1}) p(z_n | z_{n-1})$$

Basis:

**Problem:** The values $\omega(z_{nk})$ can come very close to zero, by multiplying them we potentially exceed the precision of double precision floating points.

**Solution:** Because $\log (\max f) = \max \log f$, we can work in “log-space” which turns multiplications into additions ...

The Viterbi Algorithm in log-space

ω(\(z_n\)) is the probability of the most likely sequence of states \(z_1, \ldots, z_n\) generating the observations \(x_1, \ldots, x_n\)

\[
\log \omega(z_n) = \max_{z_1, \ldots, z_{n-1}} \log p(x_1, \ldots, x_n, z_1, \ldots, z_n)
\]

Recursion:

\[
\log \omega(z_n) = \log p(x_n | z_n) + \max_{z_{n-1}} \left( \log \omega(z_{n-1}) + \log p(z_n | z_{n-1}) \right)
\]

\[
\hat{\omega}(z_n) = \log p(x_n | z_n) + \max_{z_{n-1}} \left( \hat{\omega}(z_{n-1}) + \log p(z_n | z_{n-1}) \right)
\]

Basis:

\[
\hat{\omega}(z_1) = \log \prod_{k=1}^{K} \left\{ \pi_k p(x_1 | \phi_k) \right\}^{z_{1k}} = \sum_{k=1}^{K} z_{1k} \left( \log \pi_k + \log p(x_1 | \phi_k) \right)
\]
The Viterbi Algorithm in log-space

\( \omega(\mathbf{z}_n) \) is the probability of the most likely sequence of states \( \mathbf{z}_1, \ldots, \mathbf{z}_n \) generating the observations \( \mathbf{x}_1, \ldots, \mathbf{x}_n \)

Recursion:

\[
\log \omega(\mathbf{z}_n) = \log p(\mathbf{x}_n | \mathbf{z}_n) + \max_{\mathbf{z}_{n-1}} (\log \omega(\mathbf{z}_{n-1}) + \log p(\mathbf{z}_n | \mathbf{z}_{n-1}))
\]

\( \hat{\omega}(\mathbf{z}_n) = \log p(\mathbf{x}_n | \mathbf{z}_n) + \max_{\mathbf{z}_{n-1}} (\hat{\omega}(\mathbf{z}_{n-1}) + \log p(\mathbf{z}_n | \mathbf{z}_{n-1})) \)

Basis:

\( \hat{\omega}(\mathbf{z}_1) = \log \prod_{k=1}^{K} \{ \pi_k p(\mathbf{x}_1 | \phi_k) \}^{z_1k} = \sum_{k=1}^{K} z_1k (\log \pi_k + \log p(\mathbf{x}_1 | \phi_k)) \)
The Viterbi Algorithm in log-space

$\omega(z_n)$ is the probability of generating the observations $x_1, \ldots, x_n$.

$$\log \omega(z_n) = \log p(x_n | z_n) + \max_{z_{n-1}} (\log \omega(z_{n-1}) + \log p(z_n | z_{n-1}))$$

Recursion:

$$\hat{\omega}(z_n) = \log p(x_n | z_n) + \max_{z_{n-1}} (\hat{\omega}(z_{n-1}) + \log p(z_n | z_{n-1}))$$

Basis:

$$\hat{\omega}(z_1) = \log \prod_{k=1}^{K} (\pi_k p(x_1 | \phi_k)) + \sum_{k=1}^{K} \gamma_1 (\log \pi_k + \log p(x_1 | \phi_k))$$

Still takes time $O(K^2N)$ and space $O(KN)$ using memorization, and the most likely sequence of states can be found by backtracking.
Backtracking

Pseudocode for backtracking not using log-space:

\[
\begin{align*}
z[1..N] &= \text{undef} \\
z[N] &= \arg \max_k \omega[k][N] \\
\text{for } n = N-1 \text{ to } 1: \\
& \quad z[n] = \arg \max_k ( p(x[n+1] \mid z[n+1]) \cdot \omega[k][n] \cdot p(z[n+1] \mid k) ) \\
\text{print } z[1..N]
\end{align*}
\]

Pseudocode for backtracking using log-space:

\[
\begin{align*}
z[1..N] &= \text{undef} \\
z[N] &= \arg \max_k \omega^*[k][N] \\
\text{for } n = N-1 \text{ to } 1: \\
& \quad z[n] = \arg \max_k ( \log p(x[n+1] \mid z[n+1]) + \omega^*[k][n] + \log p(z[n+1] \mid k) ) \\
\text{print } z[1..N]
\end{align*}
\]

Takes time $O(NK)$ but requires the entire $\omega$- or $\omega^*$-table in memory.
A problem with “log-space”?

\[
\omega(z_n) = p(x_n|z_n) \max_{z_{n-1}} \omega(z_{n-1}) p(z_n|z_{n-1})
\]

\[
\hat{\omega}(z_n) = \log p(x_n|z_n) + \max_{z_{n-1}} (\hat{\omega}(z_{n-1}) + \log p(z_n|z_{n-1}))
\]

What if \( p(x_n|z_n) \) or \( p(z_n|z_{n-1}) \) is 0? Then the product of probabilities becomes 0, but what should it be in log-space?
A problem with “log-space”?

\[
\omega(z_n) = p(x_n | z_n) \max_{z_{n-1}} \omega(z_{n-1}) p(z_n | z_{n-1})
\]

\[
\hat{\omega}(z_n) = \log p(x_n | z_n) + \max_{z_{n-1}} (\hat{\omega}(z_{n-1}) + \log p(z_n | z_{n-1}))
\]

What if \(p(x_n | z_n)\) or \(p(z_n | z_{n-1})\) is 0? Then the product of probabilities becomes 0, but what should it be in log-space?

It should be some representation of “minus infinity”

// Pseudo code for computing w^[k][n] for some n>1

\[
w^[n][k] = \text{undef}
\]

if \(p(x[n] | k) \neq 0\):

for \(j = 1\) to \(K\):

if \(p(k | j) \neq 0\):

\[
w^[n][k] = \max(w^[k][n], \log(p(x[n] | k)) + w^[j][n-1] + \log(p(k | j)))
\]
The Forward Algorithm

\( \alpha(z_n) \) is the joint probability of observing \( x_1, \ldots, x_n \) and being in state \( z_n \)

\[
\alpha(z_n) \equiv p(x_1, \ldots, x_n, z_n)
\]

**Recursion:**

\[
\alpha(z_n) = p(x_n | z_n) \sum_{z_{n-1}} \alpha(z_{n-1}) p(z_n | z_{n-1})
\]

**Basis:**

\[
\alpha(z_1) = p(x_1, z_1) = p(z_1) p(x_1 | z_1) = \prod_{k=1}^{K} \left\{ \pi_k p(x_1 | \phi_k) \right\}^{z_{1k}}
\]

Takes time \( O(K^2N) \) and space \( O(KN) \) using memorization
The Forward Algorithm

\( \alpha(z_n) \) is the joint probability of observing \( x_1, \ldots, x_n \) and being in state \( z_n \)

\[ \alpha(z_n) \equiv p(x_1, \ldots, x_n) \]

Recursion:

\[ \alpha(z_n) = p(x_n|z_n) \sum_{z_{n-1}} \alpha(z_{n-1}) p(z_n|z_{n-1}) \]

Basis:

\[ \alpha(z_1) = p(x_1, z_1) = p(z_1) p(x_1|z_1) = \prod_{k=1}^{K} \{ \pi_k p(x_1|\phi_k) \}^{z_{1k}} \]

Takes time \( O(K^2N) \) and space \( O(KN) \) using memorization.
The **Forward Algorithm**

The **Forward Algorithm** is the joint probability of observing \( x_1, \ldots, x_n \) and being in state \( z_n \):

\[
\alpha(z_n) \equiv p(x_1, \ldots, x_n) 
\]

**Recursion:**

\[
\alpha(z_n) = p(x_n | z_n) \sum_{z_{n-1}} \alpha(z_{n-1}) p(z_n | z_{n-1}) 
\]

**Basis:**

\[
\alpha(z_n) 
\]

**Problem:** The values \( \alpha(z_{nk}) \) can come very close to zero, by multiplying them we potentially exceed the precision of double precision floating points.

**Another problem:** Because \( \log (\Sigma f) \neq \Sigma (\log f) \), we cannot use the “log-space” trick ...
Forward algorithm using scaled values

\( \alpha(z_n) \) is the joint probability of observing \( x_1, \ldots, x_n \) and being in state \( z_n \)

\[
\alpha(z_n) = p(x_1, \ldots, x_n, z_n) = p(x_1, \ldots, x_n) p(z_n | x_1, \ldots, x_n)
\]

\[
\hat{\alpha}(z_n) = p(z_n | x_1, \ldots, x_n) = \frac{\alpha(z_n)}{p(x_1, \ldots, x_n)} = \frac{\alpha(z_n)}{\prod_{m=1}^{n} c_m}
\]

\[
c_n = p(x_n | x_1, \ldots, x_{n-1})
\]

\[
p(x_1, \ldots, x_n) = \prod_{m=1}^{n} c_m
\]

This “normalized version” \( \alpha(z_n) \) is a probability distribution over \( K \) variables, and we expect it to “behave numerically well” because

\[
\sum_{k=1}^{K} \hat{\alpha}(z_{nk}) = 1
\]

The normalized values can not all become arbitrary small ...
Forward algorithm using scaled values

We can modify the forward-recursion to use scaled values

\[
\alpha(z_n) = p(x_n|z_n) \sum_{z_{n-1}} \alpha(z_{n-1}) p(z_n|z_{n-1}) \Leftrightarrow
\]

\[
\left( \prod_{m=1}^{n} c_m \right) \hat{\alpha}(z_n) = p(x_n|z_n) \sum_{z_{n-1}} \left( \prod_{m=1}^{n-1} c_m \right) \hat{\alpha}(z_{n-1}) p(z_n|z_{n-1}) \Leftrightarrow
\]

\[
c_n \hat{\alpha}(z_n) = p(x_n|z_n) \sum_{z_{n-1}} \hat{\alpha}(z_{n-1}) p(z_n|z_{n-1})
\]

\[
\alpha(z_n) = \left( \prod_{m=1}^{n} c_m \right) \hat{\alpha}(z_n)
\]
Forward algorithm using scaled values

We can modify the forward-recursion to use scaled values

\[
\alpha(z_n) = p(x_n|z_n) \sum_{z_{n-1}} \alpha(z_{n-1}) p(z_n|z_{n-1}) \Leftrightarrow
\]

\[
\left( \prod_{m=1}^{n} c_m \right) \hat{\alpha}(z_n) = p(x_n|z_n) \sum_{z_{n-1}} \left( \prod_{m=1}^{n-1} c_m \right) \hat{\alpha}(z_{n-1}) p(z_n|z_{n-1}) \Leftrightarrow
\]

\[
c_n \hat{\alpha}(z_n) = p(x_n|z_n) \sum_{z_{n-1}} \hat{\alpha}(z_{n-1}) p(z_n|z_{n-1})
\]

If we know \( c_n \) then we have a recursion using the normalized values

\[
\alpha(z_n) = \left( \prod_{m=1}^{n} c_m \right) \hat{\alpha}(z_n)
\]
Forward algorithm using scaled values

We can modify the forward-recursion to use scaled values.

\[
\alpha(\mathbf{z}_n) = p(\mathbf{x}_n | \mathbf{z}_n) \sum_{\mathbf{z}_{n-1}} \alpha(\mathbf{z}_{n-1}) p(\mathbf{z}_n | \mathbf{z}_{n-1}) \leftrightarrow
\]

\[
\left( \prod_{m=1}^{n} c_m \right) \hat{\alpha}(\mathbf{z}_n) = p(\mathbf{x}_n | \mathbf{z}_n) \sum_{\mathbf{z}_{n-1}} \left( \prod_{m=1}^{n-1} c_m \right) \hat{\alpha}(\mathbf{z}_{n-1}) p(\mathbf{z}_n | \mathbf{z}_{n-1}) \leftrightarrow
\]

\[
c_n \hat{\alpha}(\mathbf{z}_n) = p(\mathbf{x}_n | \mathbf{z}_n) \sum_{\mathbf{z}_{n-1}} \hat{\alpha}(\mathbf{z}_{n-1}) p(\mathbf{z}_n | \mathbf{z}_{n-1})
\]

If we know \(c_n\) then we have a recursion using the normalized values.

\[
\sum_{k=1}^{K} c_n \hat{\alpha}(\mathbf{z}_{nk}) = c_n \sum_{k=1}^{K} \hat{\alpha}(\mathbf{z}_{nk}) = c_n \cdot 1
\]

\[
\alpha(\mathbf{z}_n) = \left( \prod_{m=1}^{n} c_m \right) \hat{\alpha}(\mathbf{z}_n)
\]
Forward algorithm using scaled values

We can modify the forward-recursion to use scaled values.

**Recursion:**

In step $n$ compute and store temporarily the $K$ values $\delta(z_{n1}), \ldots, \delta(z_{nK})$:

\[
\delta(z_n) = c_n \hat{\alpha}(z_n) = p(x_n|z_n) \sum_{z_{n-1}} \hat{\alpha}(z_{n-1}) p(z_n|z_{n-1})
\]

Compute and store $c_n$ as

\[
\sum_{k=1}^{K} \delta(z_{nk}) = \sum_{k=1}^{K} c_n \hat{\alpha}(z_{nk}) = c_n \sum_{k=1}^{K} \hat{\alpha}(z_{nk}) = c_n
\]

Compute and store $\hat{\alpha}(z_{nk}) = \delta(z_{nk})/c_n$
Forward algorithm using scaled values

We can modify the forward-recursion to use scaled values.

Recursion:
In step \( n \) compute and store temporarily

\[
\delta(z_n) = c_n \hat{\alpha}(z_n) = p(x_n | z_n) \sum_{z_{n-1}} \hat{\alpha}(z_{n-1}) p(z_n | z_{n-1})
\]

Compute and store \( c_n \) as

\[
\sum_{k=1}^{K} \delta(z_{nk}) = \sum_{k=1}^{K} c_n \hat{\alpha}(z_{nk}) = c_n \sum_{k=1}^{K} \hat{\alpha}(z_{nk}) = c_n
\]

Compute and store \( \hat{\alpha}(z_{nk}) = \delta(z_{nk})/c_n \)

Basis:

\[
\hat{\alpha}(z_1) = \frac{\alpha(z_1)}{c_1} \quad c_1 = p(x_1) = \sum_{z_1} p(z_1)p(x_1 | z_1) = \sum_{k=1}^{K} \pi_k p(x_1 | \phi_k)
\]
Forward algorithm using scaled values

We can modify the forward-recursion to use scaled values

Recursion:

In step $n$ compute and store temporarily

$$\delta(z_n) = c_n \hat{\alpha}(z_n) = p(x_n | z_n) \sum_{z_{n-1}} \hat{\alpha}(z_{n-1}) p(z_n | z_{n-1})$$

Compute and store $c_n$ as

$$\sum_{k=1}^{K} \delta(z_{nk}) = \sum_{k=1}^{K} c_n \hat{\alpha}(z_{nk}) = c_n \sum_{k=1}^{K} \hat{\alpha}(z_{nk}) = c_n$$

Compute and store $\hat{\alpha}(z_{nk}) = \delta(z_{nk}) / c_n$

Basis:

$$\hat{\alpha}(z_1) = \frac{\alpha(z_1)}{c_1} \quad c_1 = p(x_1) = \sum_{z_1} p(z_1)p(x_1 | z_1) = \sum_{k=1}^{K} \pi_k p(x_1 | \phi_k)$$

Takes time $O(K^2N)$ and space $O(KN)$ using memorization
The Backward Algorithm

$\beta(z_n)$ is the conditional probability of future observation $x_{n+1}, \ldots, x_N$ assuming being in state $z_n$

$$\beta(z_n) \equiv p(x_{n+1}, \ldots, x_N | z_n)$$

Recursion:

$$\beta(z_n) = \sum_{z_{n+1}} \beta(z_{n+1}) p(x_{n+1} | z_{n+1}) p(z_{n+1} | z_n)$$

Basis:

$$\beta(z_N) = 1$$

Takes time $O(K^2 N)$ and space $O(KN)$ using memorization
Backward algorithm using scaled values

We can modify the backward-recursion to use scaled values

Recursion:
In step $n$ compute and store temporarily the $K$ values $\varepsilon(z_{n1}), ..., \varepsilon(z_{nk})$

$$\varepsilon(z_n) = c_{n+1} \hat{\beta}(z_n) = \sum_{z_{n+1}} \hat{\beta}(z_{n+1}) p(x_{n+1}|z_{n+1}) p(z_{n+1}|z_n)$$

Using $c_{n+1}$ computed during the forward-recursion, compute and store

$$\hat{\beta}(z_{nk}) = \varepsilon(z_{nk}) / c_{n+1}$$

Basis:

$$\hat{\beta}(z_N) = 1$$
Backward algorithm using scaled values

We can modify the backward-recursion to use scaled values.

**Recursion:**

In step $n$ compute and store temporarily

$$\epsilon(z_n) = c_{n+1} \hat{\beta}(z_n) = \sum_{z_{n+1}} \hat{\beta}(z_{n+1}) p(x_{n+1}|z_{n+1}) p(z_{n+1}|z_n)$$

Using $c_{n+1}$ computed during the forward-recursion, compute and store

$$\hat{\beta}(z_{nk}) = \epsilon(z_{nk}) / c_{n+1}$$

**Basis:**

$$\hat{\beta}(z_N) = 1$$

Takes time $O(K^2N)$ and space $O(KN)$ using memorization