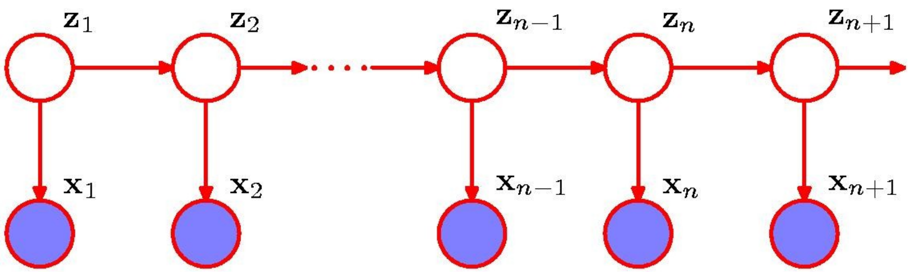
Hidden Markov Models

Implementing the forward-, backward- and Viterbi-algorithms



Viterbi

Recursion: $$\omega(\mathbf{z}_n) = p(\mathbf{x}_n|\mathbf{z}_n) \max_{\mathbf{z}_{n-1}} \omega(\mathbf{z}_{n-1})p(\mathbf{z}_n|\mathbf{z}_{n-1})$$

Basis: $$\omega(\mathbf{z}_1) = p(\mathbf{x}_1, \mathbf{z}_1) = p(\mathbf{z}_1)p(\mathbf{x}_1|\mathbf{z}_1)$$

Forward

Recursion: $$\alpha(\mathbf{z}_n) = p(\mathbf{x}_n|\mathbf{z}_n) \sum_{\mathbf{z}_{n-1}} \alpha(\mathbf{z}_{n-1})p(\mathbf{z}_n|\mathbf{z}_{n-1})$$

Basis: $$\alpha(\mathbf{z}_1) = p(\mathbf{x}_1, \mathbf{z}_1) = p(\mathbf{z}_1)p(\mathbf{x}_1|\mathbf{z}_1)$$

Backward

Recursion: $$\beta(\mathbf{z}_n) = \sum_{\mathbf{z}_{n+1}} \beta(\mathbf{z}_{n+1})p(\mathbf{x}_{n+1}|\mathbf{z}_{n+1})p(\mathbf{z}_{n+1}|\mathbf{z}_n)$$

Basis: $$\beta(\mathbf{z}_N) = 1$$

Problem: The values in the *ω*-, *α*-, and *β*-tables can come very close to zero, by multiplying them we potentially exceed the precision of double precision floating points and get underflow

Forward

Recursion: $$\alpha(\mathbf{z}_n) = p(\mathbf{x}_n|\mathbf{z}_n) \sum_{\mathbf{z}_{n-1}} \alpha(\mathbf{z}_{n-1}) p(\mathbf{z}_n|\mathbf{z}_{n-1})$$

Basis: $$\alpha(\mathbf{z}_1) = p(\mathbf{x}_1, \mathbf{z}_1) = p(\mathbf{z}_1)p(\mathbf{x}_1|\mathbf{z}_1)$$

Backward

Recursion: $$\beta(\mathbf{z}_n) = \sum_{\mathbf{z}_{n+1}} \beta(\mathbf{z}_{n+1}) p(\mathbf{x}_{n+1}|\mathbf{z}_{n+1}) p(\mathbf{z}_{n+1}|\mathbf{z}_n)$$

Basis: $$\beta(\mathbf{z}_N) = 1$$

The Viterbi algorithm

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

$$\omega(\mathbf{z}_n) \equiv \max_{\mathbf{z}_1,\ldots,\mathbf{z}_{n-1}} p(\mathbf{x}_1,\ldots,\mathbf{x}_n,\mathbf{z}_1,\ldots,\mathbf{z}_n)$$

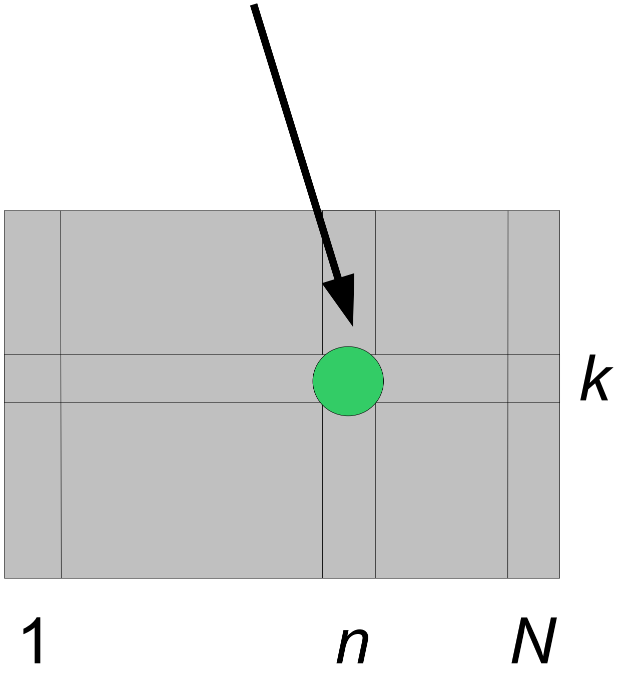
Recursion:

$\omega[k][n] = \omega(\mathbf{z}_n)$ if $\mathbf{z}_n$ is state k

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

Basis:

$$\omega(\mathbf{z}_1) = p(\mathbf{x}_1,\mathbf{z}_1) = p(\mathbf{z}_1)p(\mathbf{x}_1|\mathbf{z}_1)$$



Computing ω takes **time O(K^2N)** and **space O(KN)** using memorization

The Viterbi algorithm

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

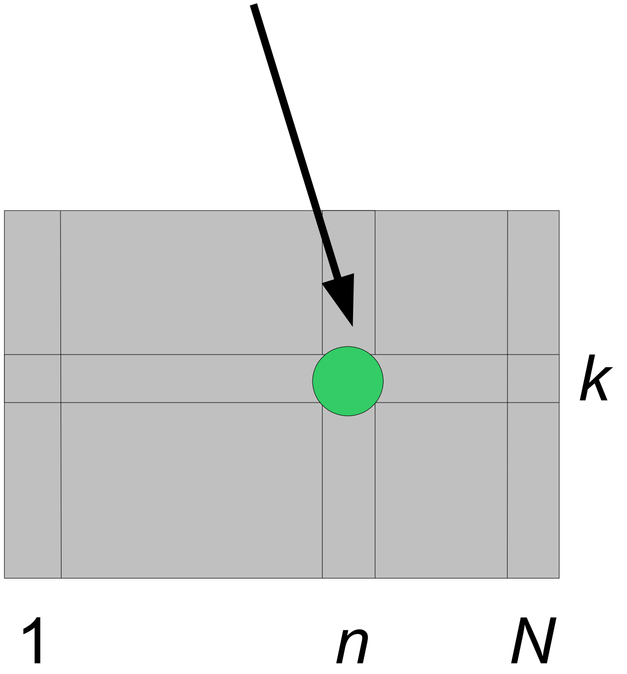
Solution to underflow-problem: Because **log max f = max log f**, we can work in "log-space" which turns multiplications into additions and thus avoids too small values

$\omega[k][n] = \omega(\mathbf{z}_n)$ if $\mathbf{z}_n$ is state k

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

Basis:

$$\omega(\mathbf{z}_1) = p(\mathbf{x}_1, \mathbf{z}_1) = p(\mathbf{z}_1)p(\mathbf{x}_1|\mathbf{z}_1)$$



Computing ω takes **time O(K^2N)** and **space O(KN)** using memorization

The Viterbi algorithm in "log-space"

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

$$\begin{aligned}
\log \omega(\mathbf{z}_n) &= \log \max_{\mathbf{z}_1,\ldots,\mathbf{z}_{n-1}} p(\mathbf{x}_1,\ldots,\mathbf{x}_n,\mathbf{z}_1,\ldots,\mathbf{z}_n) \\
&= \log(p(\mathbf{x}_n|\mathbf{z}_n) \max_{\mathbf{z}_{n-1}} \omega(\mathbf{z}_{n-1}) p(\mathbf{z}_n|\mathbf{z}_{n-1})) \\
&= \log p(\mathbf{x}_n|\mathbf{z}_n) + \log(\max_{\mathbf{z}_{n-1}} \omega(\mathbf{z}_{n-1}) p(\mathbf{z}_n|\mathbf{z}_{n-1})) \\
&= \log p(\mathbf{x}_n|\mathbf{z}_n) + \max_{\mathbf{z}_{n-1}} \log(\omega(\mathbf{z}_{n-1}) p(\mathbf{z}_n|\mathbf{z}_{n-1})) \\
&= \log p(\mathbf{x}_n|\mathbf{z}_n) + \max_{\mathbf{z}_{n-1}} (\log \omega(\mathbf{z}_{n-1}) + \log p(\mathbf{z}_n|\mathbf{z}_{n-1}))
\end{aligned}$$

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

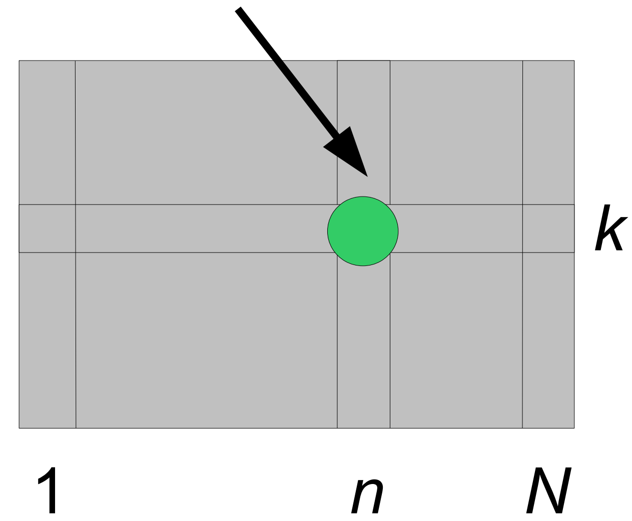
Basis: $\hat{\omega}(\mathbf{z}_1) = \log(p(\mathbf{z}_1) p(\mathbf{x}_1|\mathbf{z}_1)) = \log p(\mathbf{z}_1) + \log p(\mathbf{x}_1|\mathbf{z}_1)$

The Viterbi algorithm in "log-space"

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

ω^[k][n] = ω^($\mathbf{z}_n$) if $\mathbf{z}_n$ is state k

$$\begin{aligned}
\log\omega(\mathbf{z}_n) &= \log \max_{\mathbf{z}_1,\ldots,\mathbf{z}_{n-1}} p(\mathbf{x}_1,\ldots,\mathbf{x}_n,\mathbf{z}_1,\ldots \\
&= \log(p(\mathbf{x}_n|\mathbf{z}_n) \max_{\mathbf{z}_{n-1}} \omega(\mathbf{z}_{n-1}) p(\mathbf{z}_n| \\
&= \log p(\mathbf{x}_n|\mathbf{z}_n) + \log(\max_{\mathbf{z}_{n-1}} \omega(\mathbf{z}_{n-1} \\
&= \log p(\mathbf{x}_n|\mathbf{z}_n) + \max_{\mathbf{z}_{n-1}} \log(\omega(\mathbf{z}_{n-1} \\
&= \log p(\mathbf{x}_n|\mathbf{z}_n) + \max_{\mathbf{z}_{n-1}} (\log\omega(\mathbf{z}_{n-1}) + \log p(\mathbf{z}_n|\mathbf{z}_{n-1}))
\end{aligned}$$



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

Basis: $\hat{\omega}(\mathbf{z}_1) = \log(p(\mathbf{z}_1)p(\mathbf{x}_1|\mathbf{z}_1)) = \log p(\mathbf{z}_1) + \log p(\mathbf{x}_1|\mathbf{z}_1)$

The Viterbi algorithm in "log-space"

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

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

What if $p(\mathbf{x}_n|\mathbf{z}_n)$ or $p(\mathbf{z}_n|\mathbf{z}_{n-1})$ is 0? Then the product of probabilities becomes 0, but what should it be in "log-space"?

The Viterbi algorithm in “log-space”

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

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

What if $p(\mathbf{x}_n|\mathbf{z}_n)$ or $p(\mathbf{z}_n|\mathbf{z}_{n-1})$ is 0? Then the product of probabilities becomes 0, but what should it be in “log-space”?

“log 0” should be some representation of “minus infinity”

```
// Pseudo code for computing ω^[k][n] for some n>1
ω[k][n] = “minus infinity”
for j = 1 to K:
    ω^[k][n] = max( ω^[k][n], log(p(x[n] | k)) + ω^[ j ][n-1] + log(p( k | j)) )
```

The Viterbi algorithm in “log-space”

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

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

Still takes **time O(*K*^{2}*N*)** and **space O(*KN*)** using memorization, and the most likely sequence of states can be found be *backtracking*

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

ω[*k*][*n*] = “minus infinity”

for *j* = 1 to *K*:

    ω^[*k*][*n*] = max(*ω*^[*k*][*n*], log(*p*(x[*n*] | *k*)) + *ω*^[*j*][*n*-1] + log(*p*(*k* | *j*)))

Backtracking

Pseudocode for backtracking not using log-space:

```
z[1..N] = undef

z[N] = arg max_k ω[k][N]

for n = N-1 to 1:
      z[n] = arg max_k ( p(x[n+1] | z[n+1]) * ω[k][n] * p(z[n+1] | k ) )

print z[1..N]
```

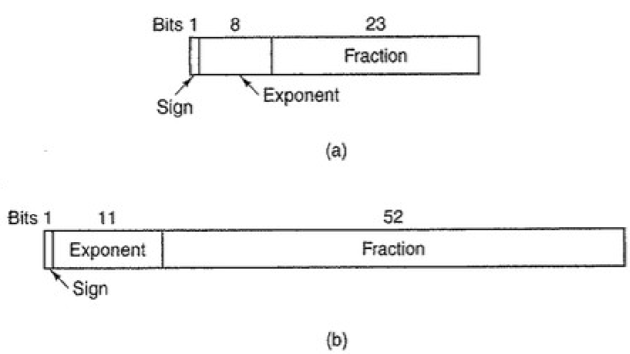
Pseudocode for backtracking using log-space:

```
z[1..N] = undef

z[N] = arg max_k ω^[k][N]

for n = N-1 to 1:
      z[n] = arg max_k ( log p(x[n+1] | z[n+1]) + ω^[k][n] + log p(z[n+1] | k ) )

print z[1..N]
```

Takes time $O(NK)$ but requires the entire ω- or ω^-table in memory

Why "log-space" helps

A floating point number *n* is represented as $n = f * 2^e$ cf. the IEEE-754 standard which specify the range of *f* and *e*



Item	Single precision	Double precision
Bits in sign	1	1
Bits in exponent	8	11
Bits in fraction	23	52
Bits, total	32	64
Exponent system	Excess 127	Excess 1023
Exponent range	−126 to +127	−1022 to +1023
Smallest normalized number	2^{-126}	2^{-1022}
Largest normalized number	approx. 2^{128}	approx. 2^{1024}
Decimal range	approx. 10^{-38} to 10^{38}	approx. 10^{-308} to 10^{308}
Smallest denormalized number	approx. 10^{-45}	approx. 10^{-324}

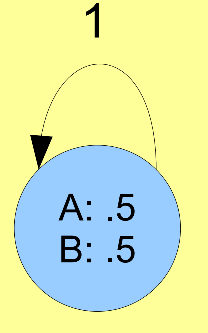
Figure B-5. Characteristics of IEEE floating-point numbers.

See e.g. Appendix B in Tanenbaum's Structured Computer Organization for further details.

Why “log-space” helps

The Viterbi-recursion for the HMM below yields:

$$\omega(\mathbf{z}_n) = p(\mathbf{z}_n|\mathbf{z}_{n-1})p(\mathbf{x}_n|\mathbf{z}_n)\omega(\mathbf{z}_{n-1}) = 1 \cdot \frac{1}{2} \cdot \omega(\mathbf{z}_{n-1}) = \left(\frac{1}{2}\right)^n = 2^{-n}$$



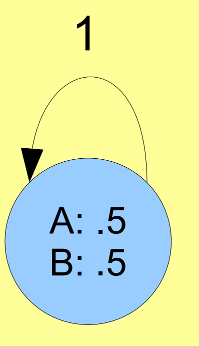
A simple HMM

Why "log-space" helps

The Viterbi-recursion for the HMM below yields:

$$\omega(\mathbf{z}_n) = p(\mathbf{z}_n|\mathbf{z}_{n-1})p(\mathbf{x}_n|\mathbf{z}_n)\omega(\mathbf{z}_{n-1}) = 1 \cdot \frac{1}{2} \cdot \omega(\mathbf{z}_{n-1}) = \left(\frac{1}{2}\right)^n = 2^{-n}$$

If n > 467 then 2^{-n} is smaller than 10^{-324}, i.e. cannot be represented



A simple HMM

Why "log-space" helps

The Viterbi-recursion for the HMM below yields:

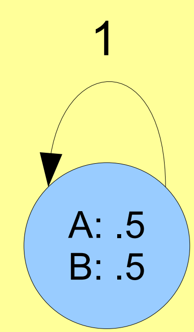
$$\omega(\mathbf{z}_n) = p(\mathbf{z}_n|\mathbf{z}_{n-1})p(\mathbf{x}_n|\mathbf{z}_n)\omega(\mathbf{z}_{n-1}) = 1 \cdot \frac{1}{2} \cdot \omega(\mathbf{z}_{n-1}) = \left(\frac{1}{2}\right)^n = 2^{-n}$$

If n > 467 then 2^{-n} is smaller than 10^{-324}, i.e. cannot be represented

The log-transformed Viterbi-recursion for the HMM below yields:

$$\begin{aligned}
\omega(\hat{\mathbf{z}}_n) &= \log p(\mathbf{z}_n|\mathbf{z}_{n-1}) + \log p(\mathbf{x}_n|\mathbf{z}_n) + \omega(\hat{\mathbf{z}}_{n-1}) \\
&= \log 1 + \log \frac{1}{2} + \omega(\mathbf{z}_{n-1}) = -1 + \omega(\mathbf{z}_{n-1}) \\
&= -n
\end{aligned}$$

No problem, as the decimal range is approx -10^{308} to 10^{308}

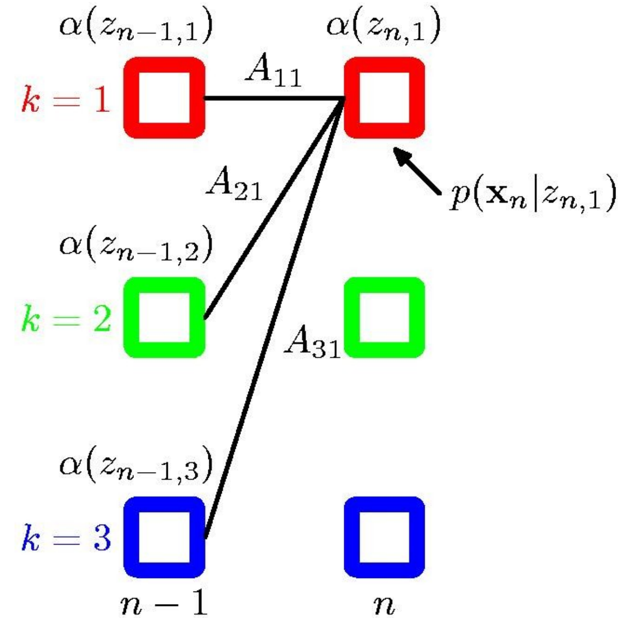


A simple HMM

The forward algorithm

$\alpha(\mathbf{z}_n)$ is the joint probability of observing $\mathbf{x}_1,\ldots,\mathbf{x}_n$ and being in state $\mathbf{z}_n$

$$\alpha(\mathbf{z}_n) \equiv p(\mathbf{x}_1, \ldots, \mathbf{x}_n, \mathbf{z}_n)$$



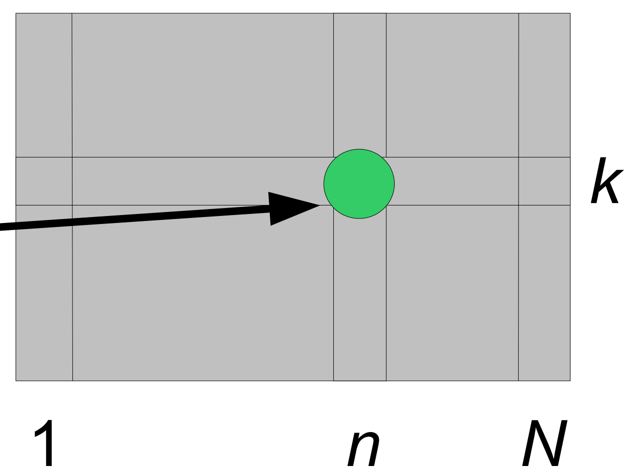
Recursion:

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

Basis:

$$\alpha(\mathbf{z}_1) = p(\mathbf{x}_1, \mathbf{z}_1) = p(\mathbf{z}_1)p(\mathbf{x}_1|\mathbf{z}_1)$$

$\alpha[k][n] = \alpha(\mathbf{z}_n)$ if $\mathbf{z}_n$ is state k

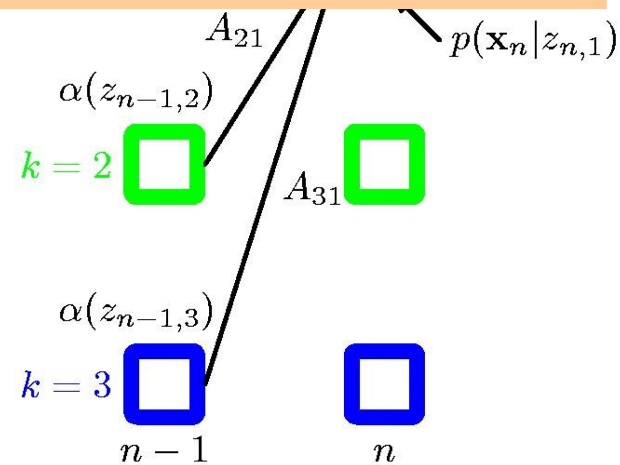


Takes time $O(K^2N)$ and space $O(KN)$ using memorization

The forward algorithm

$\alpha(\mathbf{z}_n)$ is the joint probability of observing $\mathbf{x}_1,\ldots,\mathbf{x}_n$ and being in state $\mathbf{z}_n$

Solution to underflow-problem: Since **log (Σ f) ≠ Σ (log f)**, we cannot (immediately) use the "log-space" trick



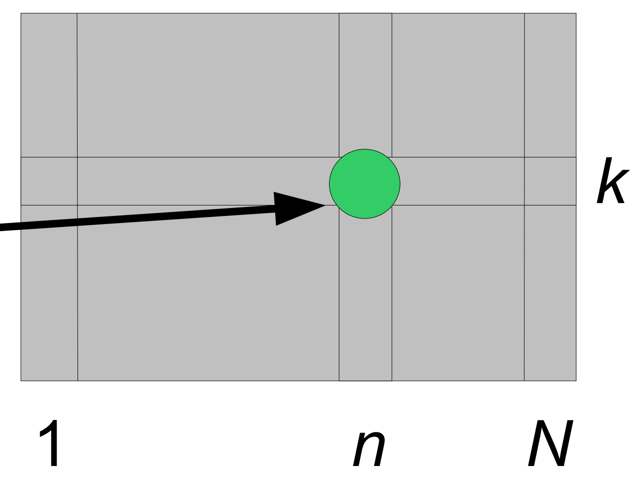
Recursion:

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

Basis:

$$\alpha(\mathbf{z}_1) = p(\mathbf{x}_1, \mathbf{z}_1) = p(\mathbf{z}_1) p(\mathbf{x}_1|\mathbf{z}_1)$$

$\alpha[k][n] = \alpha(\mathbf{z}_n)$ if $\mathbf{z}_n$ is state k



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

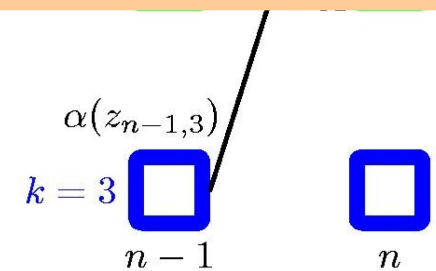
The forward algorithm

$\alpha(\mathbf{z}_n)$ is the joint probability of observing $\mathbf{x}_1,\ldots,\mathbf{x}_n$ and being in state $\mathbf{z}_n$

Solution to underflow-problem: Since **log (Σ f) ≠ Σ (log f)**, we cannot (immediately) use the "log-space" trick.

We instead use **scaling** such that values do not (all) become too small

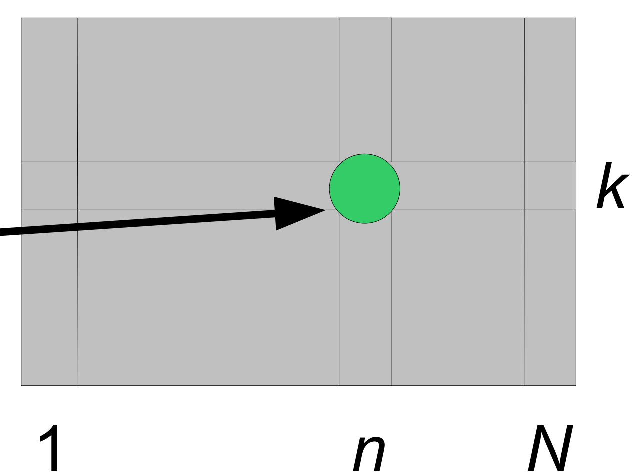
$$\alpha(\mathbf{z}_n) = p(\mathbf{x}_n|\mathbf{z}_n) \sum_{\mathbf{z}_{n-1}} \alpha(\mathbf{z}_{n-1}) p(\mathbf{z}_n|\mathbf{z}_{n-1})$$



Basis:

$$\alpha(\mathbf{z}_1) = p(\mathbf{x}_1, \mathbf{z}_1) = p(\mathbf{z}_1) p(\mathbf{x}_1|\mathbf{z}_1)$$

$\alpha[k][n] = \alpha(\mathbf{z}_n)$ if $\mathbf{z}_n$ is state k



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

Forward algorithm using scaled values

$\alpha(\mathbf{z}_n)$ is the joint probability of observing $\mathbf{x}_1,\ldots,\mathbf{x}_n$ and being in state $\mathbf{z}_n$

$$\alpha(\mathbf{z}_n) = p(\mathbf{x}_1, \ldots, \mathbf{x}_n, \mathbf{z}_n) = p(\mathbf{x}_1, \ldots, \mathbf{x}_n) p(\mathbf{z}_n | \mathbf{x}_1, \ldots, \mathbf{x}_n)$$

$$\hat{\alpha}(\mathbf{z}_n) = p(\mathbf{z}_n | \mathbf{x}_1, \ldots, \mathbf{x}_n) = \frac{\alpha(\mathbf{z}_n)}{p(\mathbf{x}_1, \ldots, \mathbf{x}_n)} = \frac{\alpha(\mathbf{z}_n)}{\prod_{m=1}^{n} c_m}$$

$$c_n = p(\mathbf{x}_n | \mathbf{x}_1, \ldots, \mathbf{x}_{n-1}) \qquad p(\mathbf{x}_1, \ldots, \mathbf{x}_n) = \prod_{m=1}^{n} c_m$$

Forward algorithm using scaled values

$\alpha(\mathbf{z}_n)$ is the joint probability of observing $\mathbf{x}_1,\ldots,\mathbf{x}_n$ and being in state $\mathbf{z}_n$

$$\alpha(\mathbf{z}_n) = p(\mathbf{x}_1, \ldots, \mathbf{x}_n, \mathbf{z}_n) = p(\mathbf{x}_1, \ldots, \mathbf{x}_n) p(\mathbf{z}_n | \mathbf{x}_1, \ldots, \mathbf{x}_n)$$

$$\hat{\alpha}(\mathbf{z}_n) = p(\mathbf{z}_n | \mathbf{x}_1, \ldots, \mathbf{x}_n) = \frac{\alpha(\mathbf{z}_n)}{p(\mathbf{x}_1, \ldots, \mathbf{x}_n)} = \frac{\alpha(\mathbf{z}_n)}{\prod_{m=1}^{n} c_m}$$

$$c_n = p(\mathbf{x}_n | \mathbf{x}_1, \ldots, \mathbf{x}_{n-1}) \qquad p(\mathbf{x}_1, \ldots, \mathbf{x}_n) = \prod_{m=1}^{n} c_m$$

This "normalized version" of $\alpha(\mathbf{z}_n)$ is a probability distribution over K outcomes, 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

$$\alpha(\mathbf{z}_n) = p(\mathbf{x}_n|\mathbf{z}_n) \sum_{\mathbf{z}_{n-1}} \alpha(\mathbf{z}_{n-1}) p(\mathbf{z}_n|\mathbf{z}_{n-1}) \Leftrightarrow$$

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

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

$$\alpha(\mathbf{z}_n) = \left(\prod_{m=1}^{n} c_m\right) \hat{\alpha}(\mathbf{z}_n)$$

Forward algorithm using scaled values

We can modify the forward-recursion to use scaled values

$$\alpha(\mathbf{z}_n) = p(\mathbf{x}_n|\mathbf{z}_n) \sum_{\mathbf{z}_{n-1}} \alpha(\mathbf{z}_{n-1}) p(\mathbf{z}_n|\mathbf{z}_{n-1}) \Leftrightarrow$$

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

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

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

$$\alpha(\mathbf{z}_n) = \left(\prod_{m=1}^{n} c_m\right) \hat{\alpha}(\mathbf{z}_n)$$

Forward algorithm using scaled values

We can modify the forward-recursion to use scaled values

$$\alpha(\mathbf{z}_n) = p(\mathbf{x}_n|\mathbf{z}_n) \sum_{\mathbf{z}_{n-1}} \alpha(\mathbf{z}_{n-1}) p(\mathbf{z}_n|\mathbf{z}_{n-1}) \Leftrightarrow$$

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

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

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

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

$$\alpha(\mathbf{z}_n) = \left(\prod_{m=1}^{n} c_m\right) \hat{\alpha}(\mathbf{z}_n)$$

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 $\delta(z_{n1})$, ..., $\delta(z_{nK})$

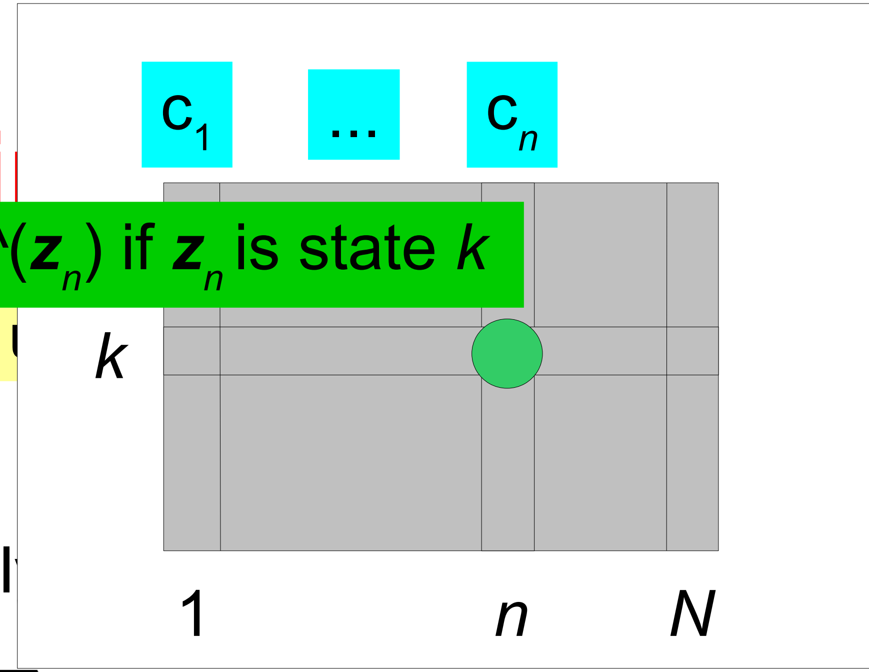
$$\delta(\mathbf{z}_n) = c_n \hat{\alpha}(\mathbf{z}_n) = p(\mathbf{x}_n|\mathbf{z}_n) \sum_{\mathbf{z}_{n-1}} \hat{\alpha}(\mathbf{z}_{n-1}) p(\mathbf{z}_n|\mathbf{z}_{n-1})$$

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 usi



We can modify the forward-recursion to u

Recursion:

In step n compute and store temporaril

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

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

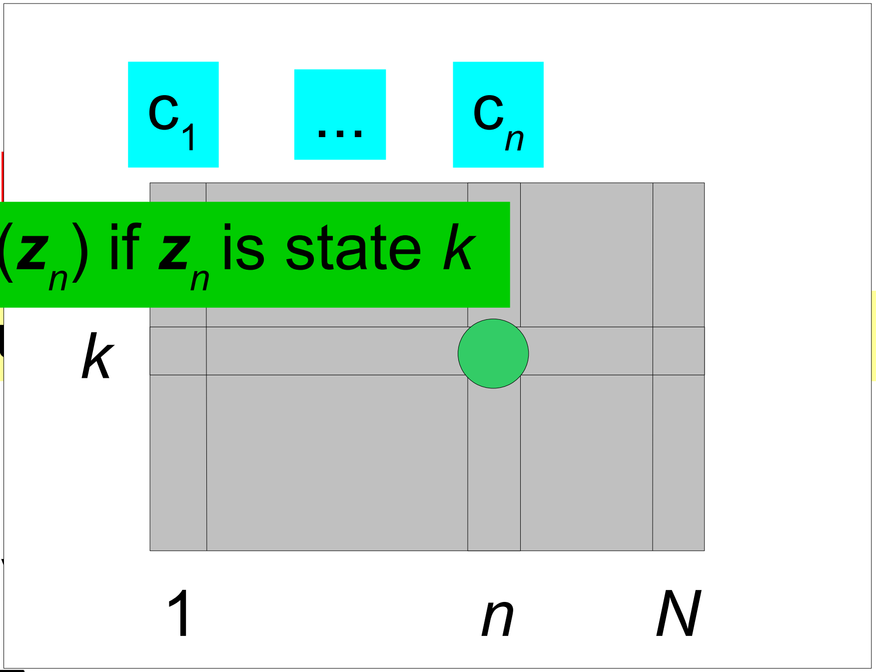
Basis:

$$\hat{\alpha}(\mathbf{z}_1) = \frac{\alpha(\mathbf{z}_1)}{c_1} \qquad c_1 = p(\mathbf{x}_1) = \sum_{\mathbf{z}_1} p(\mathbf{z}_1) p(\mathbf{x}_1|\mathbf{z}_1) = \sum_{k=1}^{K} \pi_k p(\mathbf{x}_1|\phi_k)$$

Forward algorithm usi

$\alpha\text{\^{}}[k][n] = \alpha\text{\^{}}(\mathbf{z}_n)$ if $\mathbf{z}_n$ is state k

We can modify the forward-recursion to u



Recursion:

In step n compute and store temporaril

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

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

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

Basis:

$$\hat{\alpha}(\mathbf{z}_1) = \frac{\alpha(\mathbf{z}_1)}{c_1} \qquad c_1 = p(\mathbf{x}_1) = \sum_{\mathbf{z}_1} p(\mathbf{z}_1) p(\mathbf{x}_1|\mathbf{z}_1) = \sum_{k=1}^{K} \pi_k p(\mathbf{x}_1|\phi_k)$$

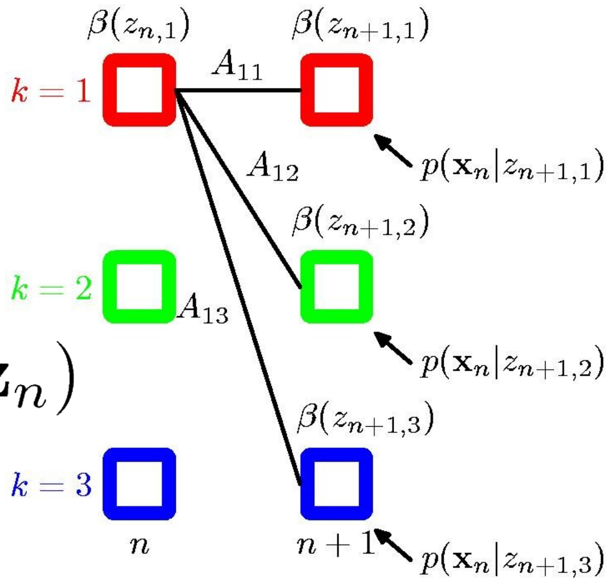
The Backward Algorithm

$\beta(\mathbf{z}_n)$ is the conditional probability of future observation $\mathbf{x}_{n+1},\ldots,\mathbf{x}_N$ assuming being in state $\mathbf{z}_n$

$$\beta(\mathbf{z}_n) \equiv p(\mathbf{x}_{n+1},\ldots,\mathbf{x}_N|\mathbf{z}_n)$$

Recursion:

$$\beta(\mathbf{z}_n) = \sum_{\mathbf{z}_{n+1}} \beta(\mathbf{z}_{n+1})p(\mathbf{x}_{n+1}|\mathbf{z}_{n+1})p(\mathbf{z}_{n+1}|\mathbf{z}_n)$$



Basis:

$$\beta(\mathbf{z}_N) = 1$$

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

Backward algorithm using scaled values

$$\hat{\beta}(\mathbf{z}_n) = \frac{\beta(\mathbf{z}_n)}{\prod_{m=n+1}^{N} c_m} = \frac{p(\mathbf{x}_{n+1},\ldots,\mathbf{x}_N|\mathbf{z}_n)}{p(\mathbf{x}_{n+1},\ldots,\mathbf{x}_N|\mathbf{x}_1,\ldots,\mathbf{x}_n)}$$

We can modify the backward-recursion to use scaled values

$$\begin{aligned}
\beta(\mathbf{z}_n) &= \sum_{\mathbf{z}_{n+1}} \beta(\mathbf{z}_{n+1}) p(\mathbf{x}_{n+1}|\mathbf{z}_{n+1}) p(\mathbf{z}_{n+1}|\mathbf{z}_n) \Leftrightarrow \\
\left(\prod_{m=n+1}^{N} c_m\right)\hat{\beta}(\mathbf{z}_n) &= \sum_{\mathbf{z}_{n+1}} \left(\prod_{m=n+2}^{N} c_m\right)\hat{\beta}(\mathbf{z}_{n+1}) p(\mathbf{x}_{n+1}|\mathbf{z}_{n+1}) p(\mathbf{z}_{n+1}|\mathbf{z}_n) \Leftrightarrow \\
c_{n+1}\hat{\beta}(\mathbf{z}_n) &= \sum_{\mathbf{z}_{n+1}} \hat{\beta}(\mathbf{z}_{n+1}) p(\mathbf{x}_{n+1}|\mathbf{z}_{n+1}) p(\mathbf{z}_{n+1}|\mathbf{z}_n)
\end{aligned}$$

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 $\epsilon(z_{n1})$, ..., $\epsilon(z_{nK})$

$$\epsilon(\mathbf{z}_n) = c_{n+1}\hat{\beta}(\mathbf{z}_n) = \sum_{\mathbf{z}_{n+1}} \hat{\beta}(\mathbf{z}_{n+1}) p(\mathbf{x}_{n+1}|\mathbf{z}_{n+1}) p(\mathbf{z}_{n+1}|\mathbf{z}_n)$$

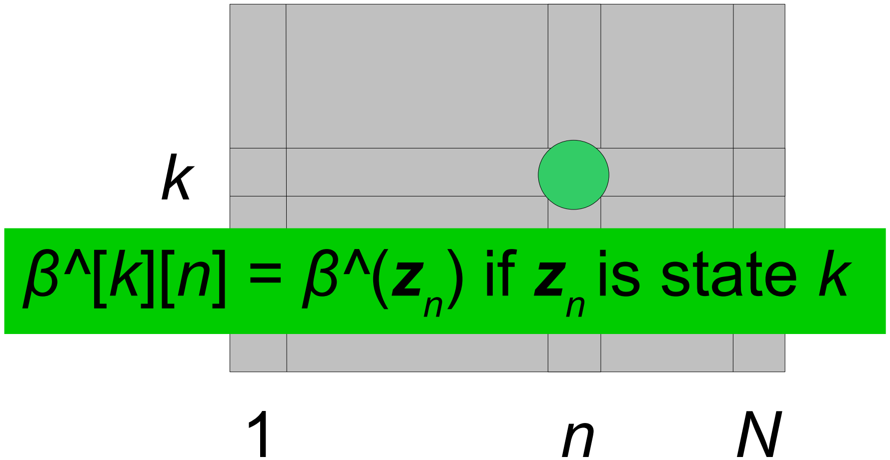
Using c_{n+1} computed during the forward-recursion, compute and store

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

Basis:

$$\hat{\beta}(\mathbf{z}_N) = 1$$

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



Posterior decoding - Revisited

Given **X**, find **Z***, where $\mathbf{z}_n^* = \arg\max_{\mathbf{z}_n} p(\mathbf{z}_n | \mathbf{x}_1, \ldots, \mathbf{x}_N)$

is the most likely state to be in the *n*'th step.

$$\begin{aligned} p(\mathbf{z}_n | \mathbf{x}_1, \ldots, \mathbf{x}_N) &= \frac{p(\mathbf{z}_n, \mathbf{x}_1, \ldots, \mathbf{x}_N)}{p(\mathbf{x}_1, \ldots, \mathbf{x}_N)} \\ &= \frac{p(\mathbf{x}_1, \ldots, \mathbf{x}_n, \mathbf{z}_n) p(\mathbf{x}_{n+1}, \ldots, \mathbf{x}_N | \mathbf{z}_n)}{p(\mathbf{x}_1, \ldots, \mathbf{x}_N)} \\ &= \frac{\alpha(\mathbf{z}_n)\beta(\mathbf{z}_n)}{p(\mathbf{X})} \end{aligned}$$

$$\mathbf{z}_n^* = \arg\max_{\mathbf{z}_n} p(\mathbf{z}_n | \mathbf{x}_1, \ldots, \mathbf{x}_N) = \arg\max_{\mathbf{z}_n} \alpha(\mathbf{z}_n)\beta(\mathbf{z}_n)/p(\mathbf{X})$$

Posterior decoding - Revisited

Given **X**, find **Z***, where $\mathbf{z}_n^* = \arg\max_{\mathbf{z}_n} p(\mathbf{z}_n | \mathbf{x}_1, \ldots, \mathbf{x}_N)$

is the most likely state to be in the *n*'th step.

$$\begin{aligned} p(\mathbf{z}_n | \mathbf{x}_1, \ldots, \mathbf{x}_N) &= \frac{\alpha(\mathbf{z}_n)\beta(\mathbf{z}_n)}{p(\mathbf{X})} \\ &= \frac{\hat{\alpha}(\mathbf{z}_n)\left(\prod_{m=1}^{n} c_m\right)\hat{\beta}(\mathbf{z}_n)\left(\prod_{m=n+1}^{N} c_m\right)}{\left(\prod_{m=1}^{N} c_m\right)} \\ &= \hat{\alpha}(\mathbf{z}_n)\hat{\beta}(\mathbf{z}_n) \end{aligned}$$

$$\mathbf{z}_n^* = \arg\max_{\mathbf{z}_n} p(\mathbf{z}_n | \mathbf{x}_1, \ldots, \mathbf{x}_N) = \arg\max_{\mathbf{z}_n} \hat{\alpha}(\mathbf{z}_n)\hat{\beta}(\mathbf{z}_n)$$

Summary

- Implementing the **Viterbi-** and **Posterior decoding** in a "numerically" sound manner.

- **Next:** How to "build" an HMM, i.e. determining the number of observables (D), the number of hidden states (K) and the transition- and emission-probabilities.