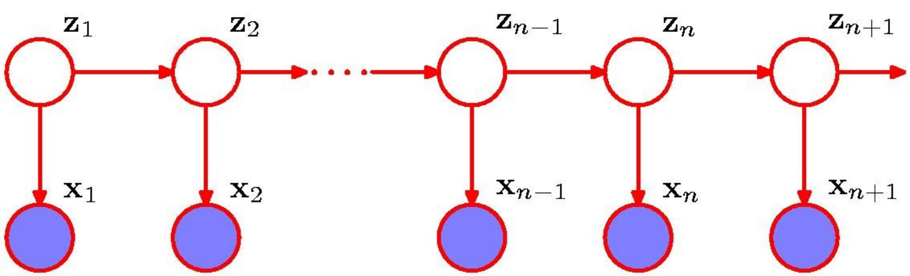
Hidden Markov Models

Implementing the forward-, backward- and Viterbi-algorithms



Christian Nørgaard Storm Pedersen

cstorm@birc.au.dk

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 $\mathbf{x}_1,\ldots,\mathbf{x}_n$

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

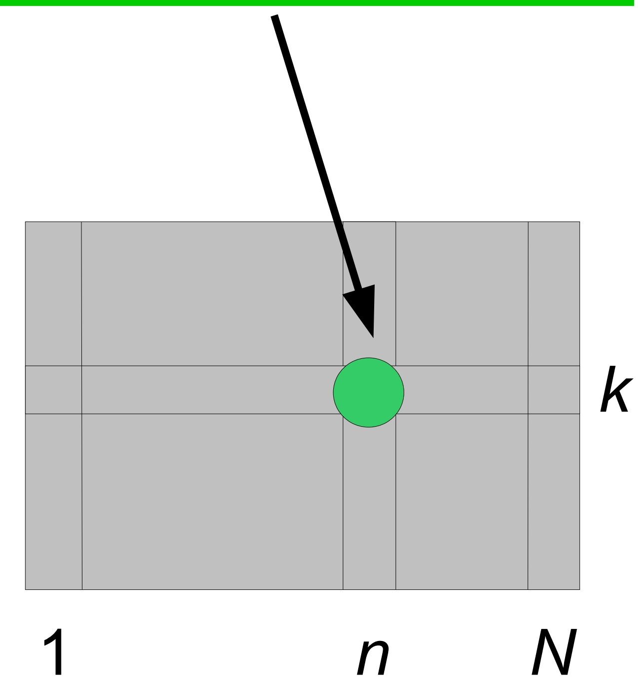
Recursion:

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

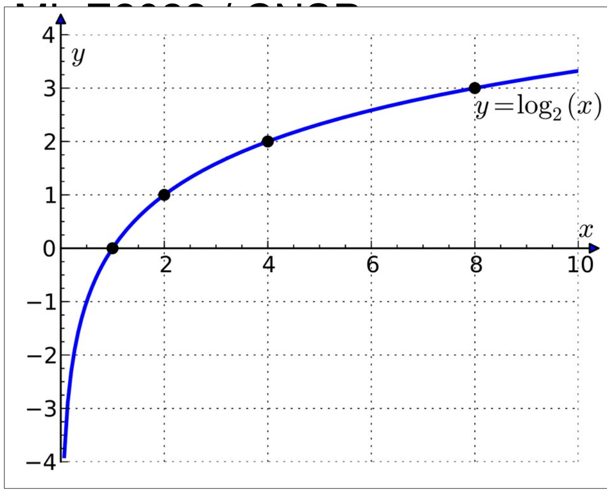
$$\omega(\mathbf{z}_n) = p(\mathbf{x}_n|\mathbf{z}_n)\max_{\mathbf{z}_{n-1}}\omega(\mathbf{z}_{n-1})p(\mathbf{z}_n|\mathbf{z}_{n-1})$$

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



he Viterbi algorithm

ity of the most likely sequence of states $\mathbf{z}_1,\ldots,\mathbf{z}_n$
ing the observations $\mathbf{x}_1,\ldots,\mathbf{x}_n$

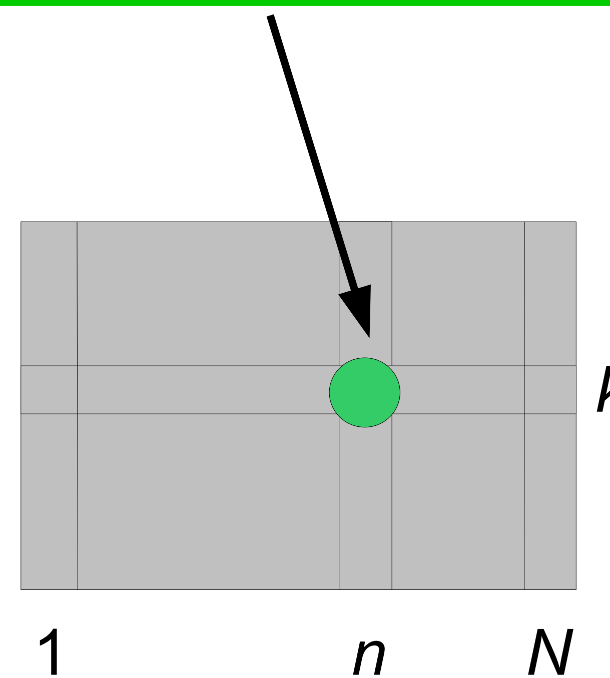
Solution to underflow-problem: Because **log max f = max log f**, we can "log-transform" which turns multiplications into additions and thus avoids too small values

$\omega[k][n] = \omega(\mathbf{z}_n)$ 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 $\mathbf{x}_1,\ldots,\mathbf{x}_n$

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

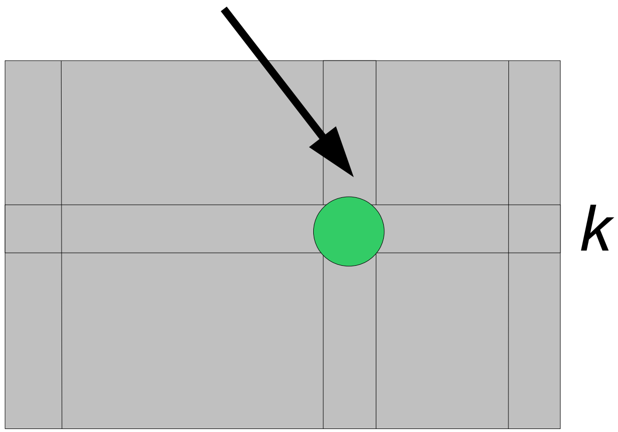
Recursion: $\hat{\omega}(\mathbf{z}_n) = \log p(\mathbf{x}_n|\mathbf{z}_n) + \max_{\mathbf{z}_{n-1}}\left(\hat{\omega}(\mathbf{z}_{n-1}) + \log p(\mathbf{z}_n|\mathbf{z}_{n-1})\right)$

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 with log-transform?

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 with log-transform?

"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^2N)** 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$_k$ $\omega[k][N]$

for n = N-1 to 1:
    z[n] = arg max$_k$ (p(x[n+1] | z[n+1]) * $\omega[k][n]$ * 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]

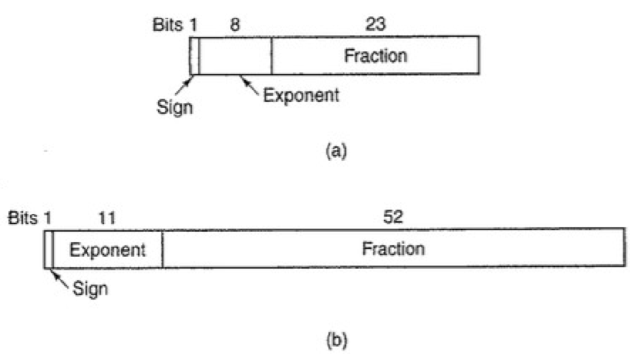
for n = N-1 to 1:
    z[n] = arg max$_k$ (log p(x[n+1] | z[n+1]) + ω^[k][n] + log p(z[n+1] | k))

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}

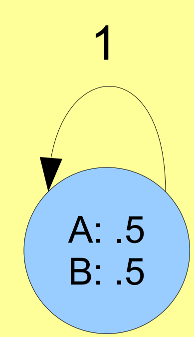
Figure B-5. Characteristics of IEEE floating-point numbers.

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:

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



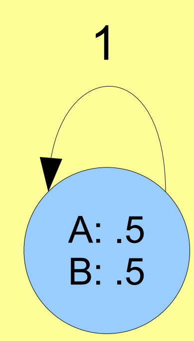
A simple HMM

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:

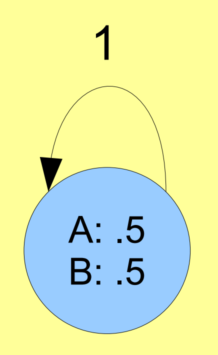
$$\omega(\mathbf{z}_n) = p(\mathbf{z}_n|\mathbf{z}_{n-1})p(\mathbf{x}_n|\mathbf{z}_n)\omega(\mathbf{z}_{n-1}) = 1 \cdot \frac{1}{2} \cdot \omega(\mathbf{z}_{n-1}) = \left(\frac{1}{2}\right)^n = 2^{-n}$$

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:

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

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

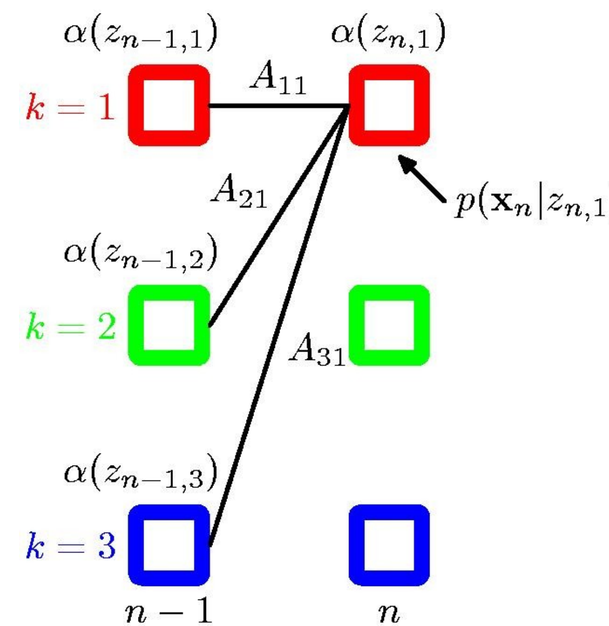


A simple HMM

The forward algorithm

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

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



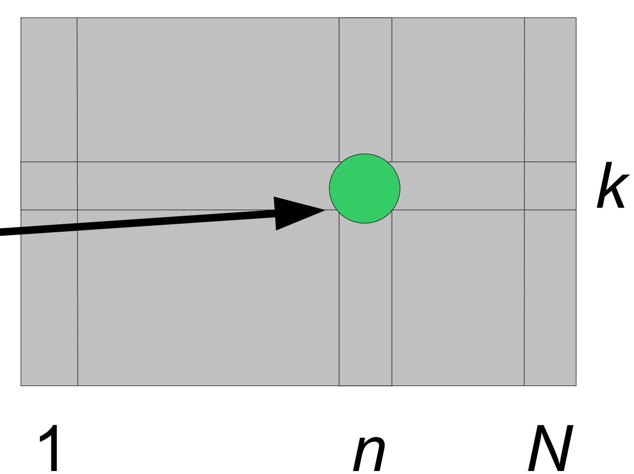
Recursion:

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

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

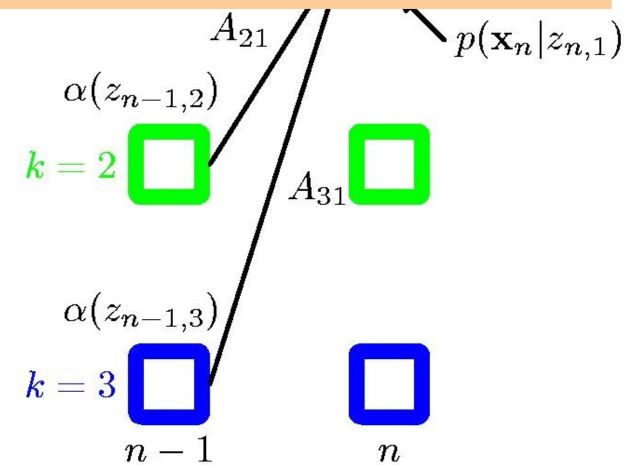


Takes time $O(K^2N)$ and space $O(KN)$ using memorization

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-transform trick.



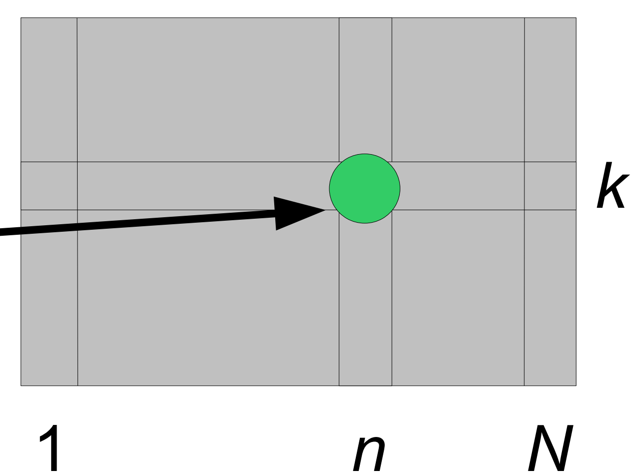
Recursion:

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

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

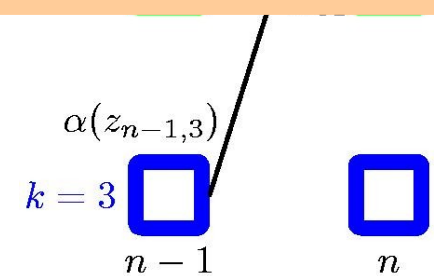
The forward algorithm

$\alpha(\mathbf{z}_n)$ 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-transform trick.

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

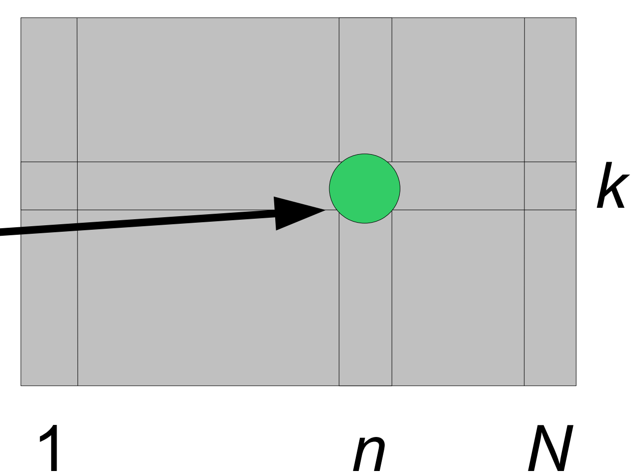
$$\alpha(\mathbf{z}_n) = p(\mathbf{x}_n|\mathbf{z}_n) \sum_{\mathbf{z}_{n-1}} \alpha(\mathbf{z}_{n-1}) p(\mathbf{z}_n|\mathbf{z}_{n-1})$$



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 $\mathbf{z}_n$

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

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

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

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 $\mathbf{z}_n$

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

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

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

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

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

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

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

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

Forward algorithm using scaled values

We can modify the forward-recursion to use scaled values

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

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

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

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

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

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

Forward algorithm using scaled values

We can modify the forward-recursion to use scaled values

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

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

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

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

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

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

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

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

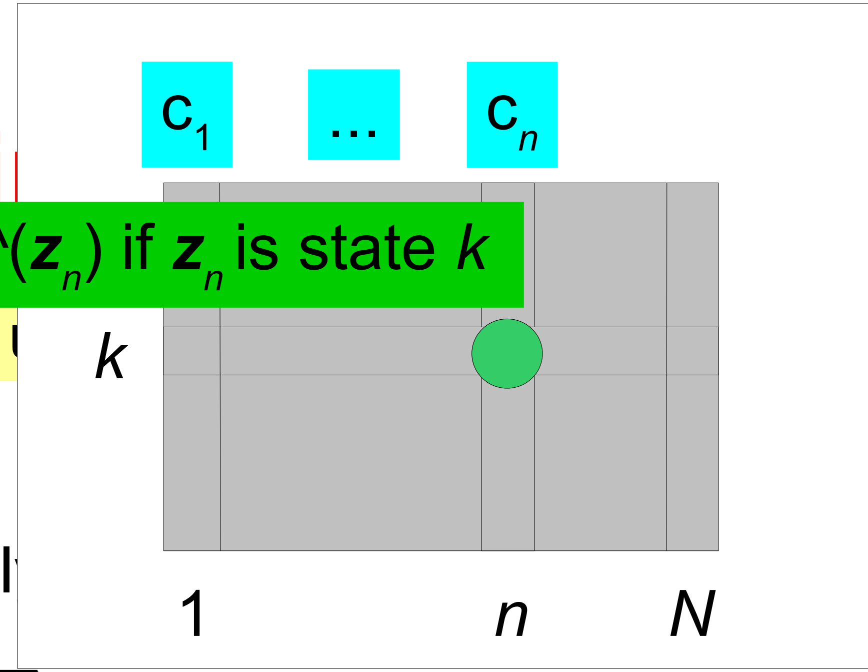
$$\delta(\mathbf{z}_n) = c_n \hat{\alpha}(\mathbf{z}_n) = p(\mathbf{x}_n|\mathbf{z}_n) \sum_{\mathbf{z}_{n-1}} \hat{\alpha}(\mathbf{z}_{n-1}) p(\mathbf{z}_n|\mathbf{z}_{n-1})$$

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

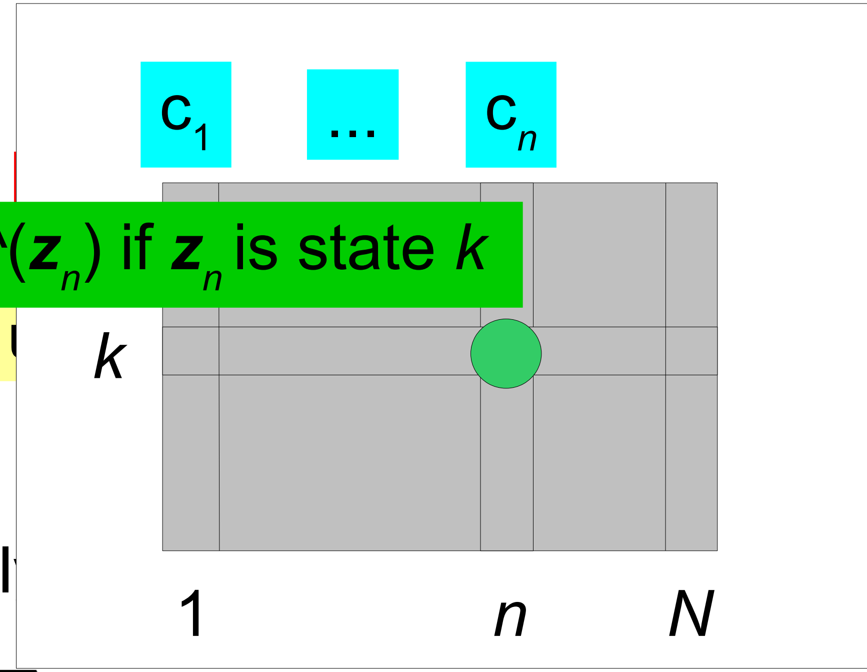
$$\sum_{k=1}^{K} \delta(z_{nk}) = \sum_{k=1}^{K} c_n \hat{\alpha}(z_{nk}) = c_n \sum_{k=1}^{K} \hat{\alpha}(z_{nk}) = c_n$$

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



α^[k][n] = α^($\mathbf{z}_n$) 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

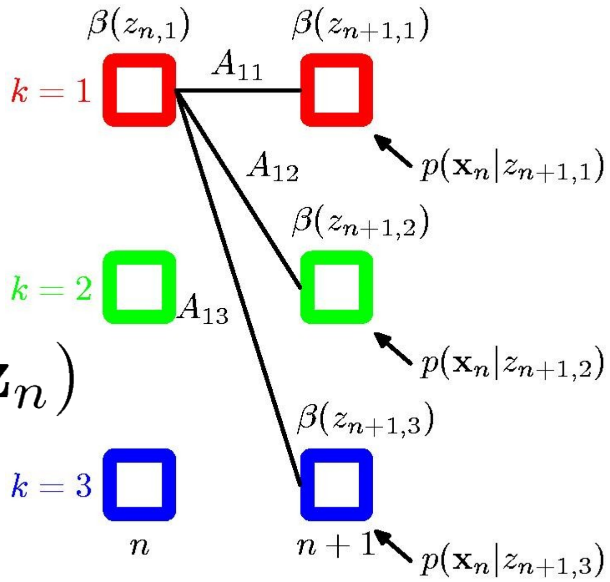
Basis:

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

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

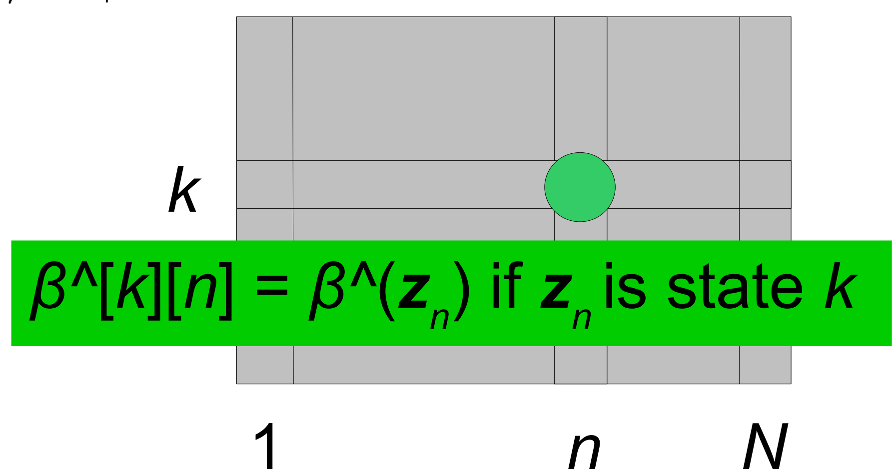
$$\epsilon(\mathbf{z}_n) = c_{n+1}\hat{\beta}(\mathbf{z}_n) = \sum_{\mathbf{z}_{n+1}} \hat{\beta}(\mathbf{z}_{n+1}) p(\mathbf{x}_{n+1}|\mathbf{z}_{n+1}) p(\mathbf{z}_{n+1}|\mathbf{z}_n)$$

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.

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

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

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.

Speeding up Viterbi decoding?

Recall: The Viterbi algorithm

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

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

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*)))

Recall: The Viterbi algorithm

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

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

Takes **time O(K^2N)** 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*)))

Observation: If $p(\mathbf{x}_n \mid \mathbf{z}_n)$ or $p(\mathbf{z}_n \mid \mathbf{z}_{n-1})$ is 0, then $\omega(\mathbf{z}_n)$ is 0 (ω^$(\mathbf{z}_n)$ is -inf)

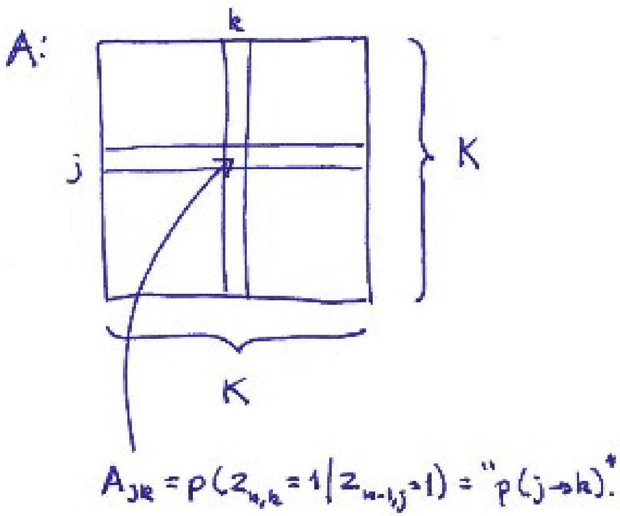
Recall: The Viterbi algorithm

```
// 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)) )
```

```
// Modified pseudo code for computing ω^[k][n] for some n>1
ω^[k][n] = "minus infinity"
if log(p(x[n] | k)) != "minus infinity" then
    for j = 1 to K:
        if log(p( k | j)) != "minus infinity" then
            ω^[k][n] = max( ω^[k][n], log(p(x[n] | k)) + ω^[ j ][n-1] + log(p( k | j)) )
```

Still takes **time O(K^2N)** and **space O(KN)** using memorization, but we might avoid some costly table lookups. What if we could avoid considering predecessors (j), where $p(k \mid j) == 0$?

Organizing transition probabilities



The transition matrix, A, is an *incidence matrix*, where $p(k \mid j) = A_{jk}$ is the weight of the edge from state j to state k in the transition diagram.

Idea: If we keep track of the transition diagram using *adjacency lists*, i.e. **Adj[*k*] = [(*j*, A_{jk}) for all states *j* where *p*(*k* | *j*) != 0]**, then we can write:

```
// Modified pseudo code for computing ω^[k][n] for some n>1
ω^[k][n] = "minus infinity"
if log(p(x[n] | k)) != "minus infinity" then
    for j in Adj[k]:
        ω^[k][n] = max( ω^[k][n], log(p(x[n] | k)) + ω^[ j ][n-1] + log(p( k | j)) )
```

Now takes **time O(*"edges in transition diagram"* x *N*)** and **space O(*KN*)** using memorization, where "edges" ≤ K^2. Gives a significant speedup for many models in practice. The idea can also be applied to implementations of the forward- and backward-algorithm.

Example – 7-state HMM

Observable: {A, C, G, T}, States: {0,1, 2, 3, 4, 5, 6}

$$A \quad \begin{matrix} 0.00 & 0.00 & 0.90 & 0.10 & 0.00 & 0.00 & 0.00 \\ 1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.05 & 0.90 & 0.05 & 0.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 1.00 & 0.00 \\ 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 1.00 \\ 0.00 & 0.00 & 0.00 & 0.10 & 0.90 & 0.00 & 0.00 \end{matrix} \qquad \pi \quad \begin{matrix} 0.00 \\ 0.00 \\ 0.00 \\ 1.00 \\ 0.00 \\ 0.00 \\ 0.00 \end{matrix} \qquad \varphi \quad \begin{matrix} 0.30 & 0.25 & 0.25 & 0.20 \\ 0.20 & 0.35 & 0.15 & 0.30 \\ 0.40 & 0.15 & 0.20 & 0.25 \\ 0.25 & 0.25 & 0.25 & 0.25 \\ 0.20 & 0.40 & 0.30 & 0.10 \\ 0.30 & 0.20 & 0.30 & 0.20 \\ 0.15 & 0.30 & 0.20 & 0.35 \end{matrix}$$

K = 7, K^2 = 49, “edges” = 11