**MAP algorithm**

Kalle Ruttik  
Communications Laboratory  
Helsinki University of Technology  
May 2, 2007

---

**Soft decoder performance**

**Need for a soft decoder**

![Diagram of decoder with Viterbi Equalizer and Decoder]

- Viterbi Equalizer provides only ML bit sequence $\hat{x}$
- ML sequence $\hat{x}$ contains hard bits
- The decoder following the Viterbi equalizer has to operate on hard bits

---

**Soft bit calculation**

- In order to have soft values at the input of the decoder the equalizer should generate a soft output.
- Viterbi algorithm generates at the output only ML sequence → hard bits.
- The soft information can be expressed as *aposteriori* probability of a bit.
  - In the input to the equalizer the soft information is described as the probability of the input symbol.
  - At the output the soft information can be expressed as *aposteriori* probability of the symbol.
  - In case of binary bits the soft information can be expressed as loglikelihood ratio (llr).

---

**Performance**

- The performance of the decoder can be improved if the decoder operates on the soft values.
  - Soft value contains not only the bit value but also the reliability information.
  - For example soft value could be the bit probability.

---

Comparison of soft and hard Viterbi decoder performance.
Soft bit as loglikelihood ratio

- For deciding a binary bit value accordingly to Bayesian decision approach we have to calculate llr for a bit and compare it to the decision threshold
  - Loglikelihood ratio is a sufficient statistics - it contains all information for making optimal decision
  - In our setup (equalizer decoder) at the output of the equalizer we do not want to make decision (yet) but postpone it to the decoder stage
  - We provide into the decoder input sufficient statistics
- For bit sequence the llr for a bit has to be calculated by first marginalising the bit probabilities
  - Marginalization meas integrating (summing) over the values of nonrelevant variables

Marginalisation

- Marginalisation of a distribution of a variable is a removal of the impact of the nonrelevant variables
- For example marginal distribution of a variable \( x_1 \) from the distribution of three variables \( x_1, x_2, x_3 \) is
  \[
  p(x_1) = \int \int p(x_1, x_2, x_3) dx_1 dx_2
  \]
- for a discrete variables we could replace the integral over distribution by a sum over possible values
  \[
  p(x_1) = \sum_{x_2} \sum_{x_3} p(x_1, x_2, x_3)
  \]

Marginalisation ...

- Further simplification: if the variables are independent the probability can be split
  \[
  p(x_1)p(x_2)p(x_3)
  \]
- The marginal probability of \( x_1 \)
  \[
  p(x_1) = p(x_1) \left( \sum_{x_2} p(x_2) \right) \left( \sum_{x_3} p(x_3) \right)
  \]
- Because we sum over probabilities: the sum over all possible values of \( x_2 \) and \( x_3 \) are 1

Soft bit

- LLR contain sufficient statistics for making optimal decision about the bit
  \[
  \log \frac{p(x = 1 | y)}{p(x = 0 | y)}
  \]
- replace both of the conditional probabilities by using Bayes formula
  \[
  p(x = 0 | y) = \frac{p(y | x = 0)p(x)}{p(y | x = 0) + p(y | x = 1)} = \frac{p(y | x = 0)p(x)}{p(y)}
  \]
  \[
  p(x = 1 | y) = \frac{p(y | x = 1)p(x)}{p(y | x = 0) + p(y | x = 1)} = \frac{p(y | x = 1)p(x)}{p(y)}
  \]
- If the symbols are \textit{apriori} equally probable we can write
  \[
  \log \frac{p(x = 1 | y)}{p(x = 0 | y)} = \log (p(x = 1 | y)) - \log (p(x = 1 | y))
Soft information from a symbol sequence

- Assume that the symbols are observed in the additive white noise
- For a symbol sequence we have from marginalisation
  \[
  \log \left( \frac{\sum_{x_k=1} p(x_1, x_2, \ldots, x_k = 1, \ldots, x_N | y_1, y_2, \ldots, y_N)}{\sum_{x_k=0} p(x_1, x_2, \ldots, x_k = 0, \ldots, x_N | y_1, y_2, \ldots, y_N)} \right)
  \]
- Based on Bayes formula
  \[
  p(x_1, x_2, \ldots, x_k = 1, \ldots, x_N | y_1, y_2, \ldots, y_N) = \frac{p(y_1, y_2, \ldots, y_N | x_1, x_2, \ldots, x_k = 1, \ldots, x_N)p(x_1, x_2, \ldots, x_k = 1)}{p(y_1, y_2, \ldots, y_N)}
  \]
  \[
  p(x_1, x_2, \ldots, x_k = 0, \ldots, x_N | y_1, y_2, \ldots, y_N) = \frac{p(y_1, y_2, \ldots, y_N | x_1, x_2, \ldots, x_k = 0, \ldots, x_N)p(x_1, x_2, \ldots, x_k = 0)}{p(y_1, y_2, \ldots, y_N)}
  \]

MAP algorithm

- We can simplify
  \[
  \log \frac{\sum_{x_k=1} p(y_1, y_2, \ldots, y_N | x_1, x_2, \ldots, x_k = 1, \ldots, x_N)}{\sum_{x_k=0} p(y_1, y_2, \ldots, y_N | x_1, x_2, \ldots, x_k = 0, \ldots, x_N)}
  \]
  \[
  = \log \frac{\sum_{x_k=1} \prod_{k_i} p(y_{k_i} | x_1, x_2, \ldots, x_k = 1, \ldots, x_N)}{\sum_{x_k=0} \prod_{k_i} p(y_{k_i} | x_1, x_2, \ldots, x_k = 0, \ldots, x_N)}
  \]
  \[
  = \log \frac{\sum_{x_k=1} \prod_{k_i} p(y_{k_i} | x_1, x_2, \ldots, x_k = 1)}{\sum_{x_k=0} \prod_{k_i} p(y_{k_i} | x_1, x_2, \ldots, x_k = 0)}
  \]

MAP algorithm

- The last equation is the general form for the Maximum Aposteriori Probability (MAP) estimation algorithm
- The optimal algorithm for estimating not only bit values but also their reliability is maximum aposteriori probability (MAP) estimation for each bit in the received codeword
- For calculating the loglikelihood ratio from the aposteriori probabilities we have to sum over all possible bit sequences
  - Sum of the probabilities of the bit sequences where the bit \( x_k = 1 \) and where the bit \( x_k = 0 \)
  - Take the logarithm of the ratio of these probabilities
- This is different compared to the Viterbi algorithm where we selected only one ML path
- MAP algorithm sums over all possible codewords (paths in the trellis)
Comparison of MAP and Viterbi algorithms

- Viterbi algorithm estimates the whole sequence, MAP algorithm estimates each bit separately
- Minimization of sequence (codeword) error does not mean that the bit error is minimized
- Viterbi algorithm estimates ML sequence
  - Estimated $\hat{x}$ is a vector of hard bits corresponding to ML sequence
  - The error probability is $p(\hat{x} \neq x | y)$
- Viterbi algorithm minimizes block error ratio (BLER)
- No information about the reliability of the estimation

The loglikelihood ratio sign shows the bit value the amplitude describes reliability - how probable the decision is

MAP algorithm

Comparison of MAP and Viterbi algorithms...

- MAP for each bit estimates the \textit{a-posteriori} probability for each bit and minimizes bit error probability
  - $\hat{x}$ is a vector of likelihood ratios for each bit
  - The error probability is: $p(\hat{x} \neq x | y)$
  - The loglikelihood ratio sign shows the bit value the amplitude describes reliability - how probable the decision is

BCJR

- Implementation of MAP algorithm for decoding a linear code was proposed by Bahl, Cocke, Jelinek, and Raviv (BCJR) in 1974
  - For general trellis the algorithm is also known as forward-backward algorithm or Baum-Welch algorithm
  - The algorithm is more complex than Viterbi algorithm
  - When information bits are not equally likely the MAP decoder performs much better than Viterbi decoder
    - Used in iterative processing (turbo codes)
    - For turbo processing BCJR algorithm is slightly modified

Derivation of MAP decoding algorithm in trellis

- BCJR (MAP) algorithm for binary transmission finds the marginal probability that the received bit was 1 or 0
- Since the bit 1 (or 0) could occur in many different code words, we have to sum over the probabilities of all these code words
  - The decision is made by using the likelihood ratio of these marginal distributions for $x = 1$ and $x = 0$
  - The calculation can be structured by using trellis diagram
    - For every state sequence there is an unique path through the trellis
    - Codewords sharing common state have common bit sequence.
    - This sequence can be computed only once.
- The objective of the decoder is to examine states $s$ and compute APPs associated with the state transitions
Marginal probability

- The probability of the code words is visualised in the code tree
- For independent bits the probability of one codeword is multiplication of probabilities of the individual bits in the codeword
- The marginal probability from the code tree for some particular bit being 1 or 0 corresponds to the sum of probabilities over all the codewords where this bit is 1 or 0
- A structured way for marginal probability calculation is to use trellis

Example: calculation of marginal probabilities

\[ p(c_1, c_2, c_3) \]

\[ p(0, 0, 0) \]
\[ p(0, 0, 1) \]
\[ p(0, 1, 0) \]
\[ p(0, 1, 1) \]
\[ p(1, 0, 0) \]
\[ p(1, 0, 1) \]
\[ p(1, 1, 0) \]
\[ p(1, 1, 1) \]

\[ p^{\text{post}}(c_2 = 0) = \frac{\sum_{c_3=0} p(c_1, c_2, c_3) p(c_2) p(c_3 | c_1, c_2)}{\sum_{c_2=0} \sum_{c_3=0} p(c_1 | c_2) p(c_2) p(c_3 | c_1, c_2) + \sum_{c_2=1} \sum_{c_3=0} p(c_1 | c_2) p(c_2) p(c_3 | c_1, c_2)} \]
\[ = \frac{\sum_{c_2=0} p(c_1) p(c_2) p(c_3)}{\sum_{c_2=0} p(c_1) p(c_2) p(c_3) + \sum_{c_2=1} p(c_1) p(c_2) p(c_3)} \]
\[ = \frac{p(c_2 = 0) \left( \sum_{0} p(c_1) \right) \left( \sum_{0} p(c_3) \right)}{\sum_{c_1, c_3} p(c_1) p(c_3) \left( \sum_{0} p(c_1) + \sum_{1} p(c_2) \right)} \]

\[ p^{\text{post}}(c_2 = 1) = \frac{\sum_{c_2=1} p(c_1, c_2, c_3) p(c_2) p(c_3 | c_1, c_2)}{\sum_{c_2=0} \sum_{c_3=0} p(c_1 | c_2) p(c_2) p(c_3 | c_1, c_2) + \sum_{c_2=1} \sum_{c_3=0} p(c_1 | c_2) p(c_2) p(c_3 | c_1, c_2)} \]
\[ = \frac{\sum_{c_2=1} p(c_1) p(c_2) p(c_3)}{\sum_{c_2=0} p(c_1) p(c_2) p(c_3) + \sum_{c_2=1} p(c_1) p(c_2) p(c_3)} \]
\[ = \frac{p(c_2 = 1) \left( \sum_{0} p(c_1) \right) \left( \sum_{0} p(c_3) \right)}{p(c_2 = 0) \left( \sum_{0} p(c_1) + \sum_{1} p(c_3) \right) + \sum_{1} p(c_1) p(c_2) p(c_3) + \sum_{0} p(c_1) p(c_3) + \sum_{0} p(c_2) p(c_3)} \]

Example: a-posteriori probability

For the independent samples we can separate

\[ p^{\text{post}}(c_2 = 0) = \frac{\sum_{c_3=0} p(c_1 | c_2) p(c_2) p(c_3 | c_1, c_2)}{\sum_{c_3=0} p(c_1 | c_2) p(c_2) p(c_3 | c_1, c_2) + \sum_{c_3=1} p(c_1 | c_2) p(c_2) p(c_3 | c_1, c_2)} \]
\[ = \frac{\sum_{c_3=0} p(c_1) p(c_2) p(c_3)}{\sum_{c_3=0} p(c_1) p(c_2) p(c_3) + \sum_{c_3=1} p(c_1) p(c_2) p(c_3)} \]
\[ = \frac{p(c_2 = 0) \left( \sum_{c_1} p(c_1) \right) \left( \sum_{c_3} p(c_3) \right)}{\sum_{c_1, c_3} p(c_1) p(c_3) \left( \sum_{c_1} p(c_1) + \sum_{c_2} p(c_2) \right)} \]

Example: a-posteriori probability...

- The likelihood ratio becomes

\[ \frac{p^{\text{post}}(c_2 = 1)}{p^{\text{post}}(c_2 = 0)} = \frac{p(c_2 = 1) \left( \sum_{c_1} p(c_1) \right) \left( \sum_{c_3} p(c_3) \right)}{p(c_2 = 0) \left( \sum_{c_1} p(c_1) \right) \left( \sum_{c_3} p(c_3) \right)} \]
\[ = \frac{p(c_2 = 1) \cdot (p(c_1 = 0) + p(c_1 = 1)) \cdot (p(c_3) + p(c_3 = 0))}{p(c_2 = 0) \cdot (p(c_1 = 0) + p(c_1 = 1)) \cdot (p(c_3) + p(c_3 = 0))} \]

- The computation can be simplified by summing over all possible beginnings and ends of the codewords separately
- In this simple example the sums can be reduced to 1, for a general code this is not the case
Derivation of the forward-backward algorithm

- For a general HMM the marginalisation can be simplified by grouping together the codewords that have a common terms.

\[
\sum_{y_k} \left( \prod_{y_k} p(y_{k} | s_{k-1}, x_{k}, x_{k} = 1) \right)
\]

\[
\prod_{y_k} p(y_{k} | s_{k-1}, x_{k}, x_{k} = 0)
\]

- Let denote transition at stage \( k \) from state \( s_{k_1} = S_i \) to state \( s_{k_1+1} = S_j \) in the next stage as

\[
M_{k_1,i,j} = p(y_{k_1} | s_{k_1} = S_i, s_{k_1+1} = S_j)
\]

Derivation of the forward-backward algorithm

\[
\sum_{x_k = 1} \left( \prod_{y_k} p(y_{k} | s_{k}, x_{k}, x_{k} = 1) \right) = \sum_{x_k = 1} \left( \prod_{y_k} p(y_{k} | s_{k}, s_{k+1}, x_{k} = 1) \right)
\]

sum over all the symbol sequences where \( x_k = 1 \)

- Multiplication over all the state transition along a path corresponding to some particular codeword

- We can regroup:

  - all the codewords contain one common transition

\[
p(y_{k} | s_{k}, s_{k+1}, x_{k} = 1)
\]

Calculation of metrics

- \( A \) and \( B \) can be calculated recursively

\[
A_{k_1,i} = \sum_{s_{k_1} = S_i} p(y_{k_1 - 1}, y_{k_1 - 2}, \ldots, y_1 | s_{k_1} = S_i, s_{k_1 - 1}, \ldots, s_1)
\]

\[
A_{k_1,i} = \sum_{s_{k_1} = S_i} \left( \prod_{k_1 < k} p(y_{k_1} | s_{k_1}, s_{k_1+1}) \right) \prod_{k_1 > k} p(y_{k_1} | s_{k_1}, s_{k_1+1})
\]

- \( A_{k_1,i} \) is called forward metrics

\[
B_{k_1,j} = \sum_{s_{k_1} = S_i} \left( \prod_{k_1 > k} p(y_{k_1} | s_{k_1}, s_{k_1+1}) \right) \prod_{k_1 < k} p(y_{k_1} | s_{k_1}, s_{k_1+1})
\]

- \( B_{k_1,j} \) is called backward metrics

\[
\alpha_{k,i} = \log(A_{k,i}) \text{ is sum of the probabilities along all the paths that when starting from the beginning of the trellis at the stage } k \text{ will merge to state } i
\]

\[
\beta_{k,j} = \log(B_{k,j}) \text{ is sum of the probabilities along all the paths that when starting from the end of the trellis at the stage } k + 1 \text{ merge to state } j
\]
Illustration: froward metrics

\[
A_{k,i} = \sum_{i} M_{k,i,i} \cdot A_{k-1,i} \\
= M_{k,1,1} \cdot A_{k-1,1} + M_{k,1,2} \cdot A_{k-1,2}
\]

Illustration: bacward metrics

\[
B_{k,i} = \sum_{i} M_{k+1,i,i} \cdot B_{k+1,i} \\
= M_{k+1,1,1} \cdot B_{k+1,1} + M_{k+1,1,2} \cdot B_{k+1,2}
\]

Calculation in log domain

- For numerical stability better to compute in logarithmic domain

\[
\log (A_{k,i} \cdot M_{k,i,j} \cdot B_{k,j}) = \log (A_{k,i}) + \log (M_{k,i,j}) + \log (B_{k,j})
\]

\[
= \alpha_{k,i} + \mu_{k,i,j} + \beta_{k,j}
\]

\[
\alpha_{k,i} = \log (A_{k,i}) \quad \beta_{k,j} = \log (B_{k,j}) \quad \mu_{k,i,j} = \log (M_{k,i,j})
\]

\[
A_{k,i} = \sum_{i} M_{k,i,i} \cdot A_{k-1,i} \quad \Rightarrow \quad \log \left( \sum_{i} e^{\mu_{k,i,i} + \alpha_{k-1,i}} \right)
\]

\[
B_{k,i} = \sum_{j} M_{k+1,i,j} \cdot B_{k+1,i} \quad \Rightarrow \quad \log \left( \sum_{j} e^{\mu_{k+1,j,j} + \beta_{k+1,j}} \right)
\]

Metric for AWGN channel

- Probability calculation simplification for the equalizer in AWGN channel

\[
\frac{1}{\sqrt{2\pi \sigma^2_N}} e^{-\frac{(y_k - f(x_k, s_k, h_{ch}))^2}{2\sigma^2_N}} \Rightarrow - \log \sqrt{2\pi \sigma^2_N} - \frac{(y_k - f(x_k, s_k, h_{ch}))^2}{2\sigma^2_N}
\]

\[
\Rightarrow - \frac{1}{2\sigma^2_N} \left( y_k^2 - 2y_k f(x_k, s_k, h_{ch}) + f(x_k, s_k, h_{ch})^2 \right)
\]

\[
\Rightarrow \frac{1}{2\sigma^2_N} y_k f(x_k, s_k, h_{ch}) - \frac{1}{2\sigma^2_N} f(x_k, s_k, h_{ch})^2
\]
**Initialization**

- Suppose the decoder starts and ends with known states.
  
  \[
  A_{0,0} = \begin{cases} 
  1, & s = S_0 \\
  0, & \text{otherwise}
  \end{cases}
  \]

- If the final state is known
  
  \[
  B_{N,0} = \begin{cases} 
  1, & s_N = S_0 \\
  0, & \text{otherwise}
  \end{cases}
  \]

- If the final state of the trellis is unknown
  
  \[
  B_{N,j} = \frac{1}{2^m}, \forall s_N
  \]

**Metric calculation**

Map algorithm: Summing over all the codewords where the symbol \( x_k = 0 \) and where it is \( x_k = 1 \)

- at the node we combine the probabilities merging into this node
  
  \[
  \alpha_k(s_k = S_i) = \sum_{s_{k-1} = S_i} p(y_{k-1}, y_{k-2}, \ldots, y_1 | s_k = S_i, s_{k-1}, \ldots, s_1)
  \]

- We do not reject other paths but sum the probabilities together

- the probability of the part of the codeword continuing from the trellis state where they merge will be same for both codewords

- Similarly we can calculate for backward metric \( B \) by starting at the end of the trellis

**Algorithm**

- Initialize the forward and backward metrics \( \alpha_0^+(k) \)
- Compute the branch metrics \( \mu_{k,i,j} \)
- Compute the forward metrics \( \alpha_{k,i} \)
- Compute the backward metrics \( \beta_{k,j} \)
- Compute APP L-values \( L(b) \)
- (optional) Compute hard decisions

**Example: MAP equalizer**

- Use MAP bit estimation for estimating the transmitted bits if the channel is \( h_{ch} = 1 \cdot \delta(0) + 0.3 \cdot \delta(1) \).
- The received signal is \( y_{rec} = [1.3, 0.1, 1, -1.4, -0.9] \)
- The input is a sequence of binary bits \( b \) modulated as \( x \)
  
  \[
  \begin{align*}
  1 & \rightarrow 1 \\
  0 & \rightarrow -1
  \end{align*}
  \]
- The \( EbN0 = 2dB \)
- Estimate the most likely transmitted bits
- (Notice that we are using the same values as in the example about Viterbi equalizer.)
Metric assignment

- Metric is calculated based on the received value and the value in trellis branch.

\[
\frac{1}{||h_{ch}||^2} \left( \frac{1}{2\sigma^2_N} 2y_k f(x_k, s_k, h_{ch}) - \frac{1}{2\sigma^2_N} f(x_k, s_k, h_{ch})^2 \right)
\]

where

\[f(x_k, s_k, h_{ch}) = h_1 \cdot x_k + h_2 \cdot x_{k-1} + \cdots + h_L \cdot x_{k-(L-1)}\]

is the noiseless (mean) value of the corresponding trellis branch. This metric is normalized by the channel total power \(\frac{1}{||h_{ch}||^2}\).

Example: Backward calculation, Backward trellis

- Start from the end of the trellis and calculate the sum of the probabilities.
- Init the last stage probability to 1 (in log domain to 0): \(\beta_{S,0} = 0\).
  1. By using the state probabilities \(k+1\) and the probabilities on the transitions calculate the probabilities for the states \(k\)
  \[
  \beta_{k,i} = \log \left( \sum_j e^{\beta_{k+1,j} + \beta_{k+1,j}} \right)
  \]
  (If the values are in the log domain for the sum you have to convert them back to the probability domain.)
  2. Normalize (needed for the numerical stability)
  \[
  \beta_{k,#} = \beta_{k,#} - \max(\beta_{k,#})
  \]
  \[
  \beta_{\text{stage.state}} = \log \left( \sum_j e^{\beta_{k+1,j} + \beta_{k+1,j}} \right)
  \]
Example: Backward calculation: Stage 5 to 4

1. The new state probabilities
   \[ \beta_{4,0} = \mu_{5,0,0} + \mu_{5,0} = 0.80 + 0 \]
   \[ \beta_{4,1} = \mu_{5,1,0} + \mu_{5,0} = -1.17 + 0 \]
2. \[ \beta_{4,0} = 0.80 - (0.80) = 0, \beta_{4,1} = -1.17 - (0.80) = -1.97 \]

Example: Backward calculation: Stage 4 to 3

1. States backward probabilities
   \[ \beta_{3,0} = \log (e^{3.1} + e^{-4.54}) = 3.1 \]
   \[ \beta_{3,1} = \log (e^{3.1} + e^{-4.54}) = 2.53 \]
2. \[ \beta_{4,0} = 3.10 - (3.10) = 0, \beta_{4,1} = 2.53 - (3.10) = -0.57 \]
Example: Backward calculation: Stage 4 to 3

\[ \beta_{4,0} = 3.10 - (3.10) = 0, \beta_{4,1} = 2.53 - (3.10) = -0.57 \]

Example: Backward calculation: Stage 3 to 2

\[ \beta_{2,0} = 0.91 - (0.91) = 0, \beta_{2,1} = 0.59 - (0.91) = -0.32 \]
Example: Backward calculation: Stage 3 to 2

1. States backward probabilities
   \[ \beta_{2,0} = \log \left( e^{\beta_{2,0} + \beta_{3,0}} + e^{\beta_{3,1}} \right) = \log(e^{-8.08} + e^{4.86 + (-0.57)}) = 0.91 \]
   \[ \beta_{2,1} = \log \left( e^{\beta_{2,1} + \beta_{3,0}} + e^{\beta_{3,1} + \beta_{3,1}} \right) = \log(e^{-3.49} + e^{1.14 + (-0.57)}) = 0.59 \]

2. \[ \beta_{2,0} = 0.91 - (0.91) = 0, \quad \beta_{2,1} = 0.59 - (0.91) = -0.32 \]

Example: Backward calculation: Stage 3 to 2

1. States backward probabilities
   \[ \beta_{1,0} = \log \left( e^{\beta_{1,0} + \beta_{2,0}} + e^{\beta_{2,1} + \beta_{2,1}} \right) = \log(e^{-4.24} + e^{-0.59 + (-0.32)}) = -0.88 \]
   \[ \beta_{1,1} = \log \left( e^{\beta_{1,1} + \beta_{2,0}} + e^{\beta_{2,1} + \beta_{2,1}} \right) = \log(e^{-1.42} + e^{-2.7 + (-0.32)}) = -1.24 \]

2. \[ \beta_{1,0} = -0.88 - (-0.88) = 0, \quad \beta_{1,1} = -1.24 - (-0.88) = -0.36 \]
Example: Forward calculation

- **start from the beginning of the trellis and calculate the sum of the probabilities**
- **Init the probability of the first state at the stage to 1 (in log domain to 0):** \( \alpha_{0,0} = 0 \)
  
  1. By using the state probabilities \( k \) and the probabilities on the transitions calculate the probabilities for the states \( k+1 \)
     \[ \alpha_{k,i} = \log \left( \sum_j e^{\mu_{k,j} + \alpha_{k-1,j}} \right) \]
     (If the values are in the log domain for the sum you have to convert them back to the probability domain.)
  2. Normalize (needed for the numerical stability)
     \[ \alpha_{k,\#} = \alpha_{k,\#} - \max(\alpha_{k,\#}) \]

Example: Forward calculation: Stage 0 to 1

- **The new state probabilities**
  \[ \alpha_{1,0} = \mu_{1,0} + \alpha_{0,0} = -6.64 \]
  \[ \alpha_{1,1} = \mu_{1,1} + \alpha_{0,0} = 2.53 \]
  \[ \alpha_{1,0} = -6.64 - 2.53 = -9.17, \quad \alpha_{1,1} = 2.53 - 2.53 = 0 \]
Example: Forward calculation: Stage 0 to 1

1. The new state probabilities
   \[ \alpha_{1,0} = \mu_{1,0} + \alpha_{0,0} = -6.64 \]
   \[ \alpha_{1,1} = \mu_{1,1} + \alpha_{0,0} = 2.53 \]

2. \[ \alpha_{1,0} = -6.64 - 2.53 = -9.17, \alpha_{1,1} = 2.53 - 2.53 = 0 \]

Example: Forward calculation: Stage 1 to 2

1. \[ \alpha_{2,0} = \log(e^{\mu_{2,0} + \alpha_{1,0}} + e^{\mu_{2,1} + \alpha_{1,1}}) \]
   \[ = \log(e^{-4.24 + (-9.17)} + e^{-1.42 + 0}) = -1.42 \]
   \[ \alpha_{2,1} = \log(e^{\mu_{2,1} + \alpha_{1,0}} + e^{\mu_{2,1} + \alpha_{1,1}}) \]
   \[ = \log(e^{1.48 + (-9.17)} + e^{-1.14 + 0}) = -2.70 \]

2. \[ \alpha_{2,0} = -1.42 - (-1.42) = 0, \alpha_{2,1} = -2.70 - (-1.42) = -1.28 \]
Example: Forward calculation: Stage 2 to 3

\[ \alpha_{3,0} = \log \left( e^{H_{3,0} + \alpha_{2,0}} + e^{H_{3,0,1} + \alpha_{2,1}} \right) = \log(e^{-8.08+0} + e^{-3.49+(-1.28)}) = -4.73 \]
\[ \alpha_{3,1} = \log \left( e^{H_{3,1,0} + \alpha_{2,0}} + e^{H_{3,1,1} + \alpha_{2,1}} \right) = \log(e^{-4.54+0} + e^{-10.04+(-1.28)}) = 1.66 \]
\[ \alpha_{3,0} = -4.73 - 1.66 = -6.40, \alpha_{3,1} = 1.66 - 1.66 = 0 \]
Bite a-posteriori $P_b$

- The bit a-posteriori probability is calculated by combining forward probability is states $k$ backward probability in states $k+1$ and transitions probabilities.

$$
\log p(y_k|x_k=1,x) \\
\sum_{x_k=1} e^{\alpha_k,j+\mu_k,j+\beta_k+1,j} \\
\sum_{x_k=0} e^{\alpha_k,j+\mu_k,j+\beta_k+1,j}
$$

Example: a-posteriori for the bits: bit 1

$$
\log \left( \frac{e^{\alpha_0,0+\mu_1,0,1+\beta_1,1}}{e^{\alpha_0,0+\mu_1,0,0+\beta_1,0}} \right) = \log \left( \frac{e^{0-2.53+(-0.36)}}{e^{0-6.63+0}} \right) = 2.17 - (-6.64) = 8.81
$$
Example: a-posteriori for the bits: bit 2

\[
\log \left( \frac{e^{\alpha_{1.0}+\mu_{2.0,1}+\beta_{2.1}} + e^{\alpha_{1.1}+\mu_{1.1,1}+\beta_{2.1}}}{e^{\alpha_{1.0}+\mu_{2.0,0}+\beta_{2.0}} + e^{\alpha_{1.1}+\mu_{2.0,1}+\beta_{2.0}}} \right)
= \log \left( e^{-9.17-4.24+0} + e^{0-1.42+0} \right) - \log \left( e^{-9.17-0.59-0.32} + e^{0-2.70-0.32} \right)
= -3.14
\]

Example: a-posteriori for the bits: bit 3

\[
\log \left( \frac{e^{\alpha_{2.0}+\mu_{3.0,1}+\beta_{3.1}} + e^{\alpha_{2.1}+\mu_{3.1,1}+\beta_{3.1}}}{e^{\alpha_{2.0}+\mu_{3.0,0}+\beta_{3.0}} + e^{\alpha_{2.1}+\mu_{3.1,0}+\beta_{3.0}}} \right)
= \log \left( e^{0+8.08+0} + e^{-1.28-3.49-0} \right) - \log \left( e^{0+1.47-0.57} + e^{-1.28-1.14-0.57} \right)
= 5.64
\]

Example: a-posteriori for the bits: bit 3

\[
\log \left( \frac{e^{\alpha_{3.0}+\mu_{4.0,1}+\beta_{4.1}} + e^{\alpha_{3.1}+\mu_{4.1,1}+\beta_{4.1}}}{e^{\alpha_{3.0}+\mu_{4.0,0}+\beta_{4.0}} + e^{\alpha_{3.1}+\mu_{4.1,0}+\beta_{4.1}}} \right)
= \log \left( e^{-6.40+3.10+0} + e^{0+2.53+1.97} \right) - \log \left( e^{-6.40-4.54+0} + e^{0-10.04-1.97} \right)
= -1.41
\]
Example: a-posteriori for the bits: bit 4

\[
\log \left( \frac{e^{\alpha_{3,0} + \mu_{4,0,1} + \beta_{4,1}} + e^{\alpha_{3,1} + \mu_{4,1,1} + \beta_{4,1}}}{e^{\alpha_{3,0} + \mu_{4,0} + \beta_{4,0}} + e^{\alpha_{3,1} + \mu_{4,1,0} + \beta_{4,0}}} \right) \\
= \log \left( e^{-6.40 + 3.10 + 0} + e^{0 + 2.53 + 0} \right) - \log \left( e^{-6.40 + 4.54 + 0} + e^{0 + 10.04 + 0} \right) \\
= -1.41
\]