Recurrent Neural Networks: Training of Variants (CTC)

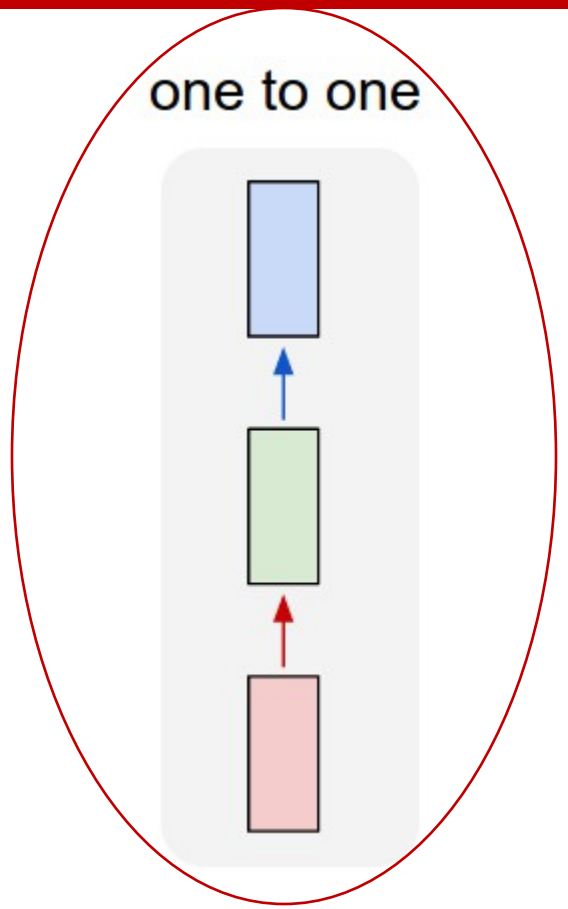
M. Soleymani
Sharif University of Technology
Spring 2020

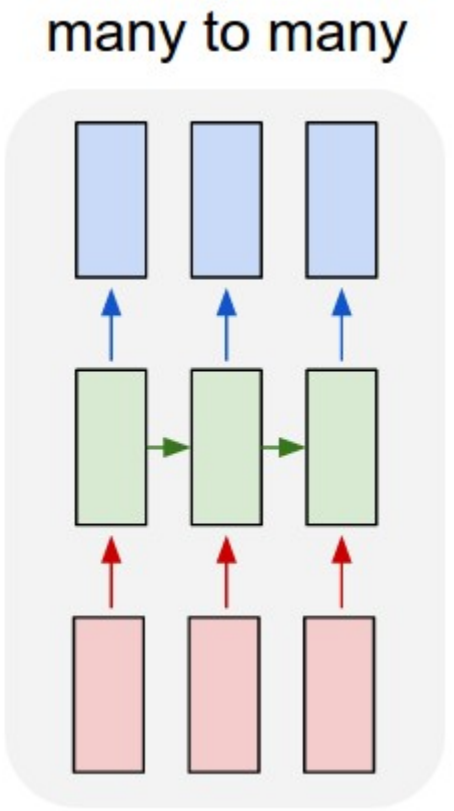
Most slides have been adopted from Bhiksha Raj, 11-785, CMU 2019.

What follows in this series on recurrent nets

- Architectures: How to train recurrent networks of different architectures
- Synchrony: How to train recurrent networks when
  - The target output is time-synchronous with the input
  - The target output is order-synchronous, but not time synchronous
  - Applies to only some types of nets
- How to make predictions/inference with such networks

Variants on recurrent nets

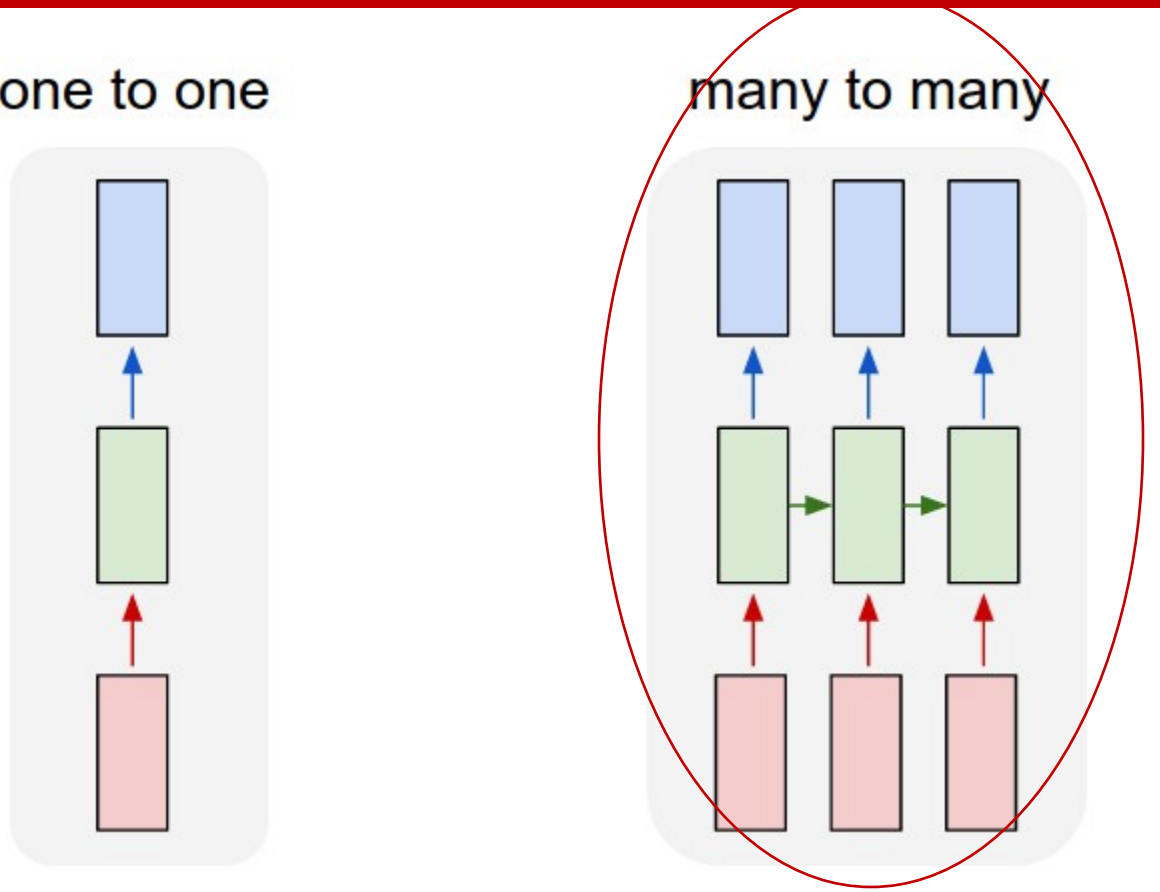




Images from Karpathy

- Conventional MLP
- Time-synchronous outputs
  - E.g. part of speech tagging

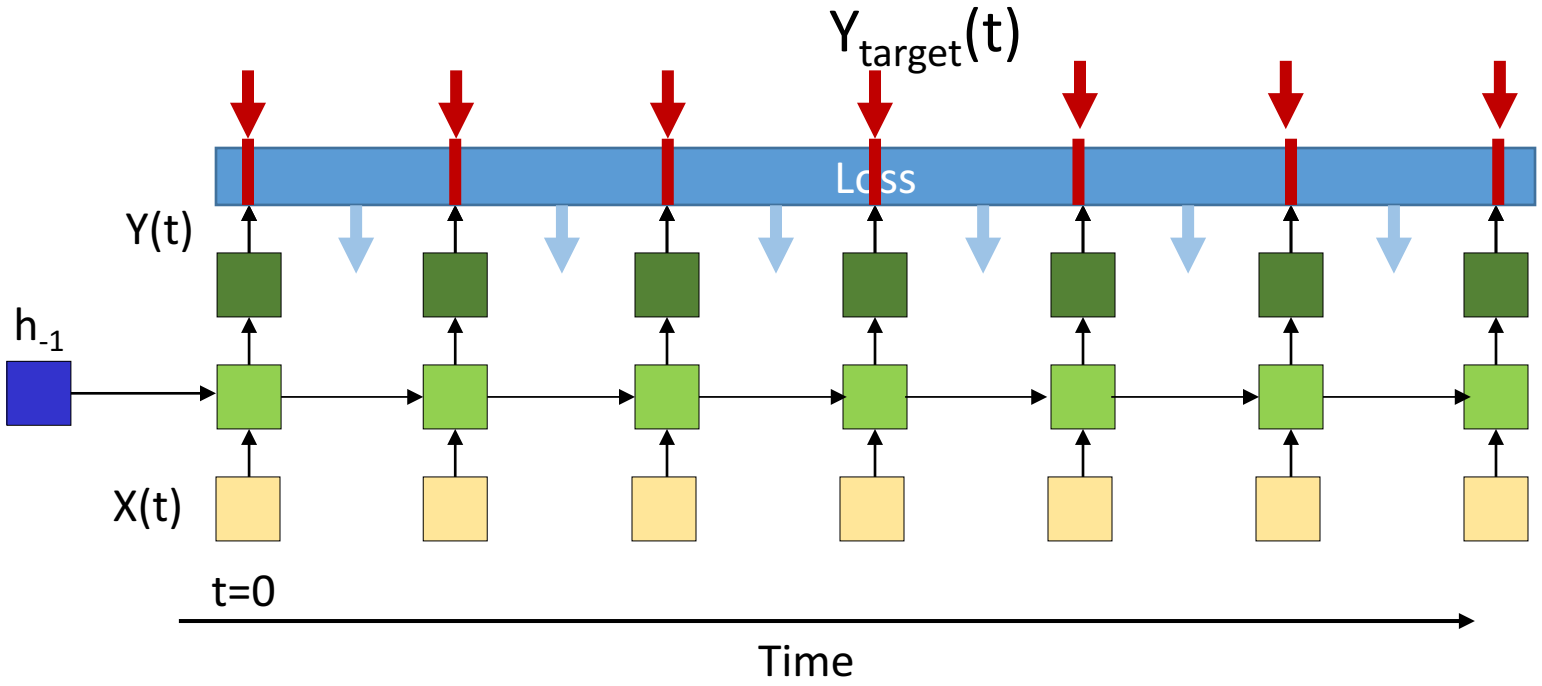
Variants on recurrent nets



Images from Karpathy

- Conventional MLP
- Time-synchronous outputs
  - E.g. part of speech tagging

Time-synchronous recurrence

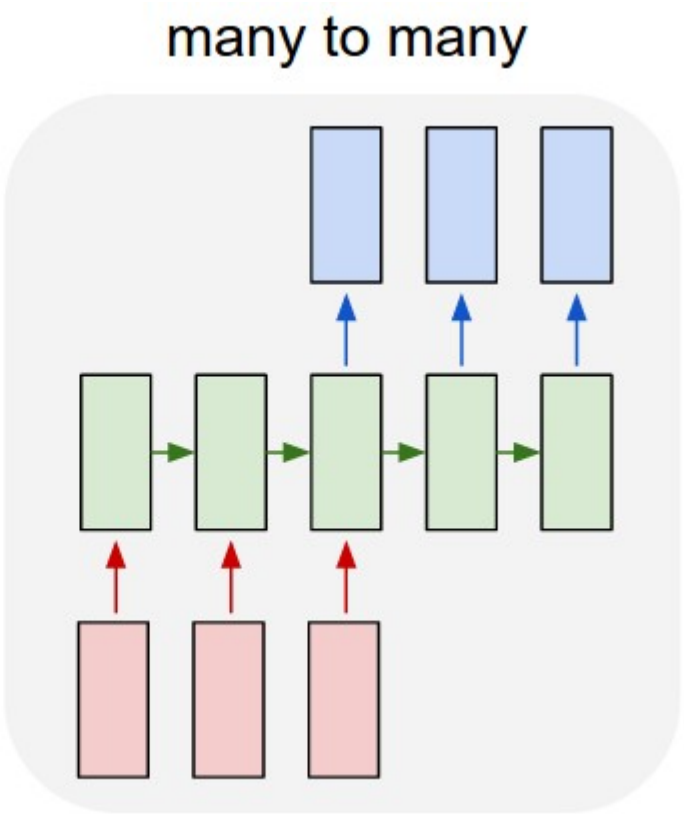


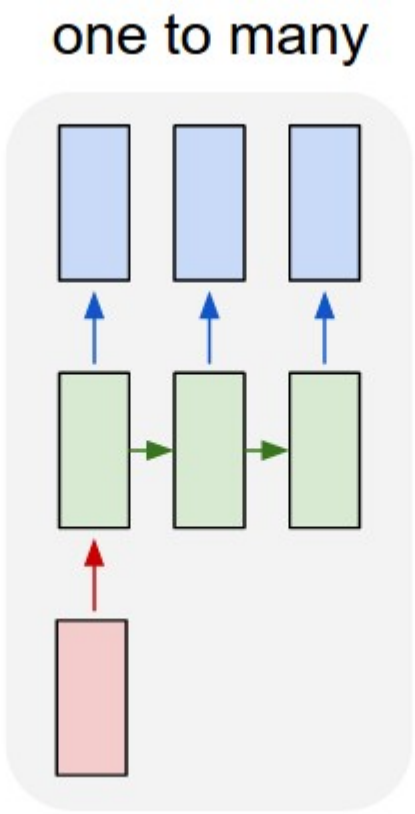
- Usual assumption: ***Sequence loss is the sum of the loss at individual instants***

$$Loss(Y^{target}(1 \ldots T), Y(1 \ldots T)) = \sum_t Loss(Y^{target}, Y(t))$$

$$\nabla_{Y(t)} Loss(Y^{target}(1 \ldots T), Y(1 \ldots T)) = \nabla_{Y(t)} Loss(Y^{target}(t), Y(t))$$

Variants

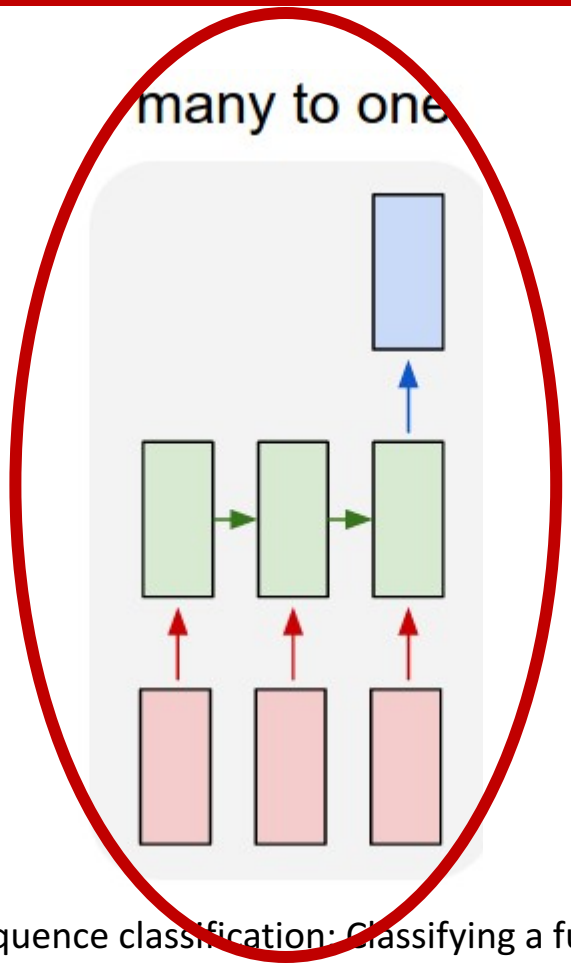


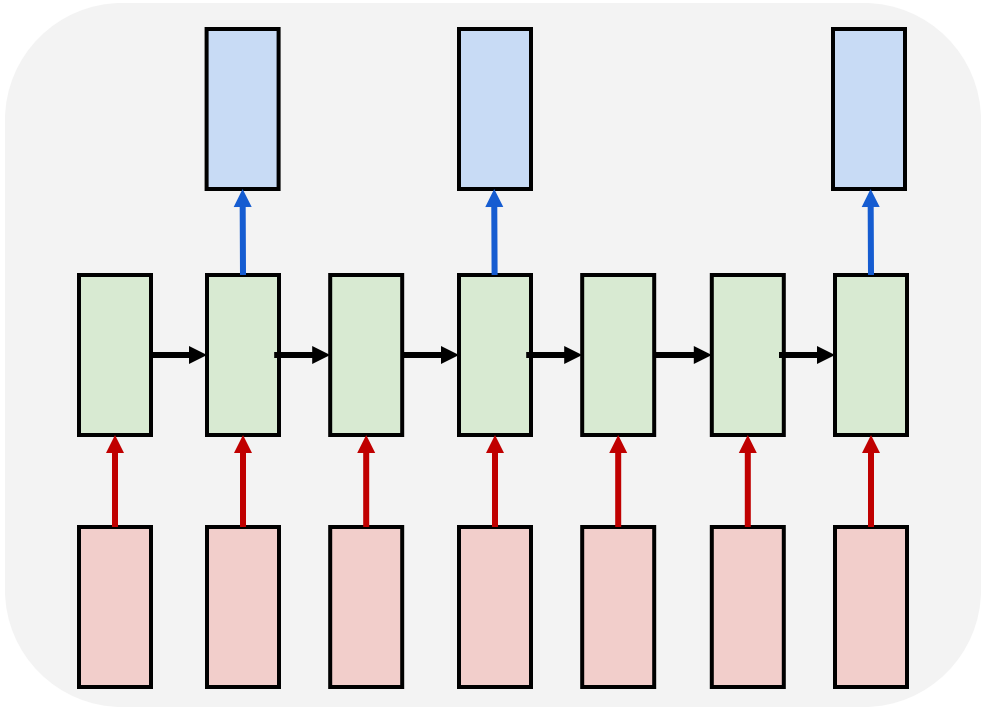


Images from Karpathy

- A posteriori sequence to sequence: Generate output sequence after processing input
  - E.g. language translation
- Single-input a posteriori sequence generation
  - E.g. captioning an image

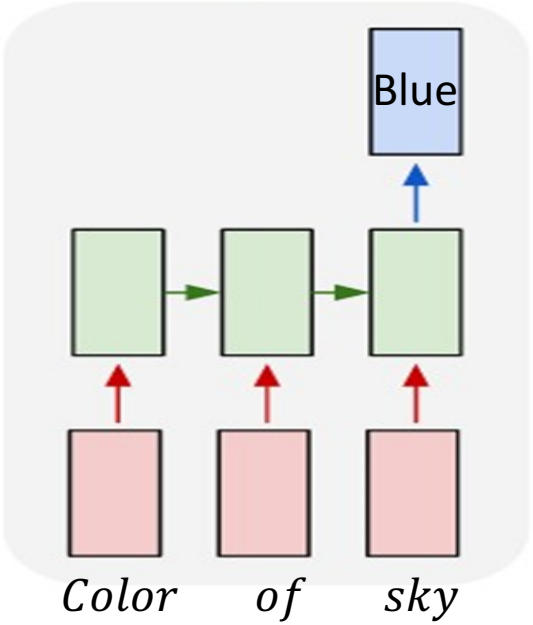
Variants on recurrent nets





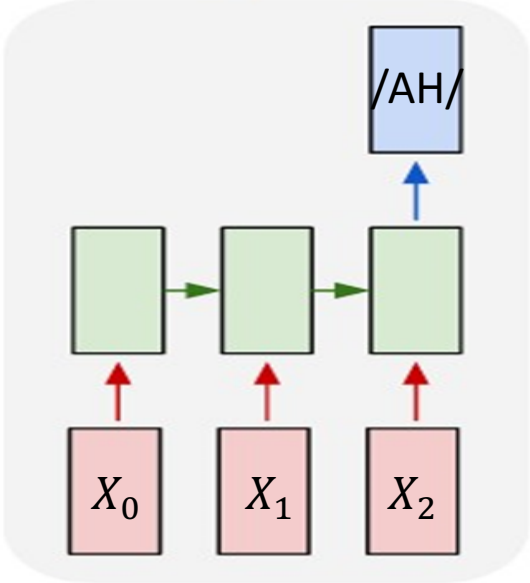
- Sequence classification: Classifying a full input sequence
  - E.g phoneme recognition
- Order synchronous , time asynchronous sequence-to-sequence generation
  - E.g. speech recognition
  - Exact location of output is unknown a priori

Example..



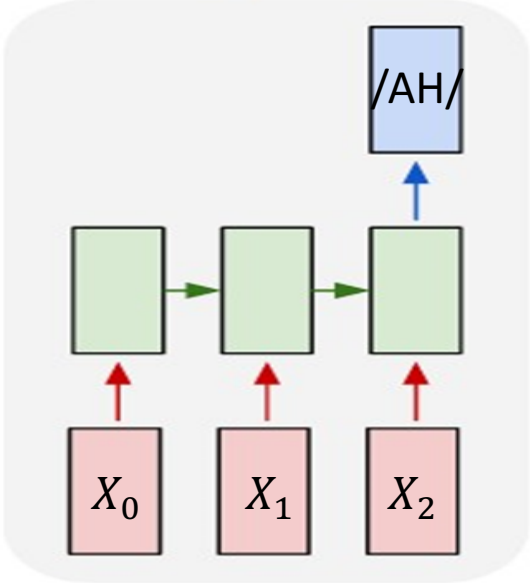
- Question answering
- Input : Sequence of words
- Output: Answer at the end of the question

Example..



- Speech recognition
- Input : Sequence of feature vectors (e.g. Mel spectra)
- Output: Phoneme ID at the end of the sequence
  - Represented as an N-dimensional output probability vector, where N is the number of phonemes

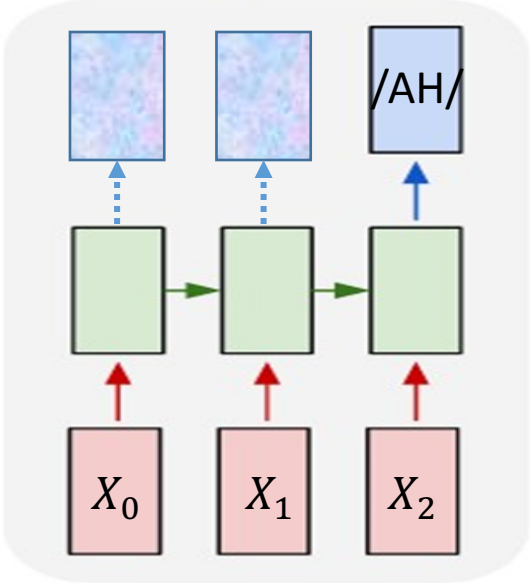
Inference: Forward pass



- Exact input sequence provided
  - Output generated when the last vector is processed
    - Output is a probability distribution over phonemes

- But what about at *intermediate stages?*

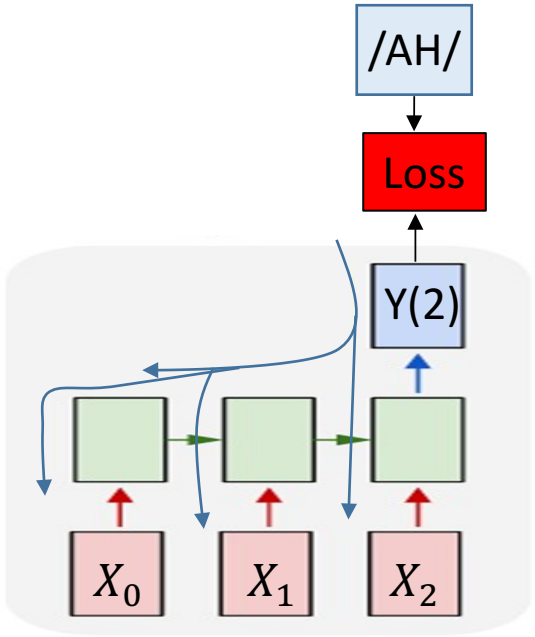
Forward pass



- Exact input sequence provided
  - Output generated when the last vector is processed
    - Output is a probability distribution over phonemes

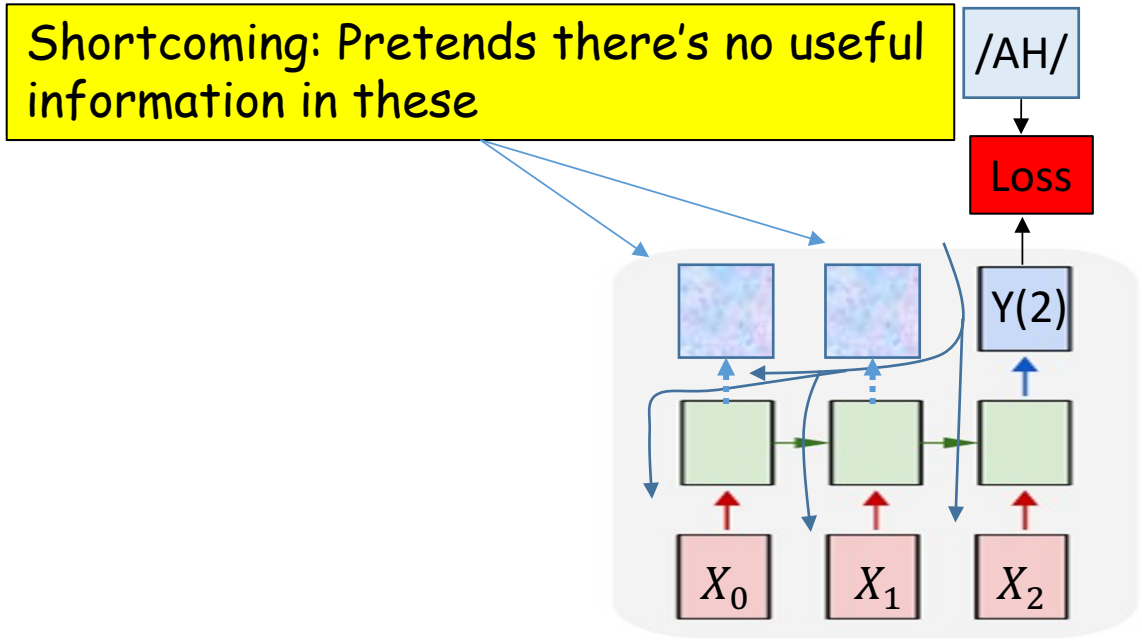
- Outputs are actually produced for *every* input
  - We only *read* it at the end of the sequence

Training



- The Loss is only defined at the final input
  - $Loss(Y^{Target}, Y) = Xent(Y(T), Phoneme)$
- This loss must propagate through the net to update all parameters

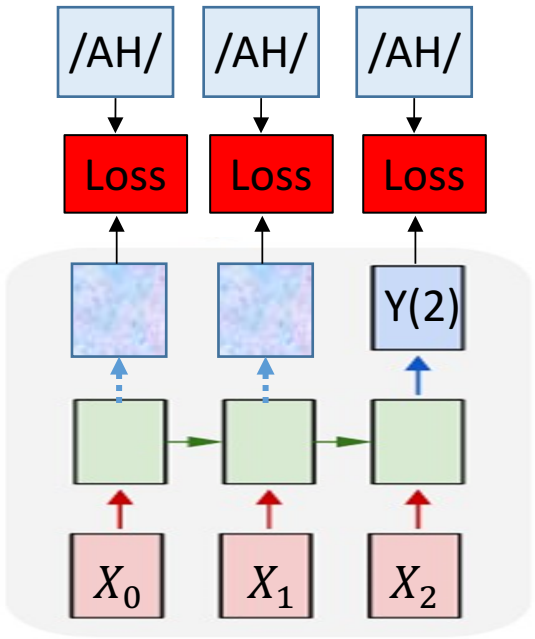
Training


Shortcoming: Pretends there's no useful information in these
/AH/
Loss
Y(2)
X_0
X_1
X_2

Training

Fix: Use these outputs too.

These too must ideally point to the correct phoneme



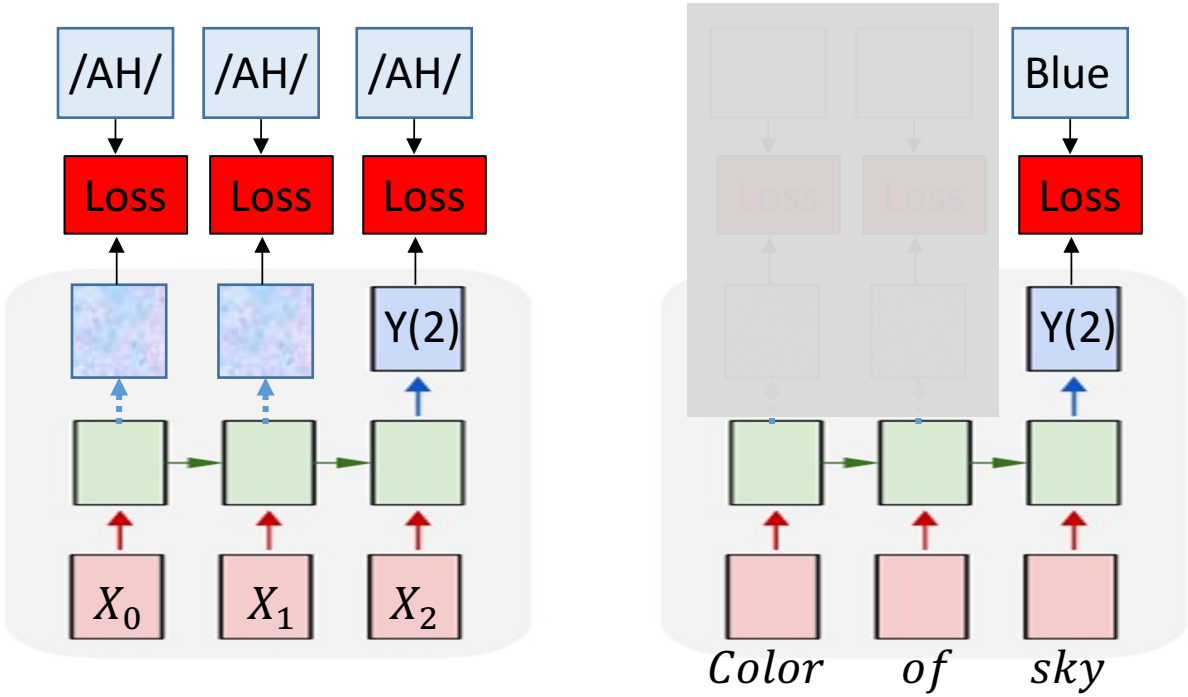
- Exploiting the untagged inputs: assume the same output for the entire input
- Define the Loss everywhere

$$Loss(Y_{target}, Y) = \sum_t w_t Xent(Y(t), Phoneme)$$

Training

Fix: Use these outputs too.

These too must ideally point to the correct phoneme

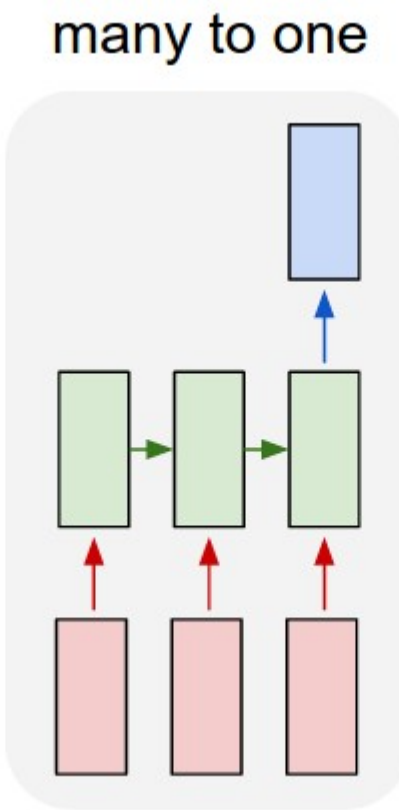


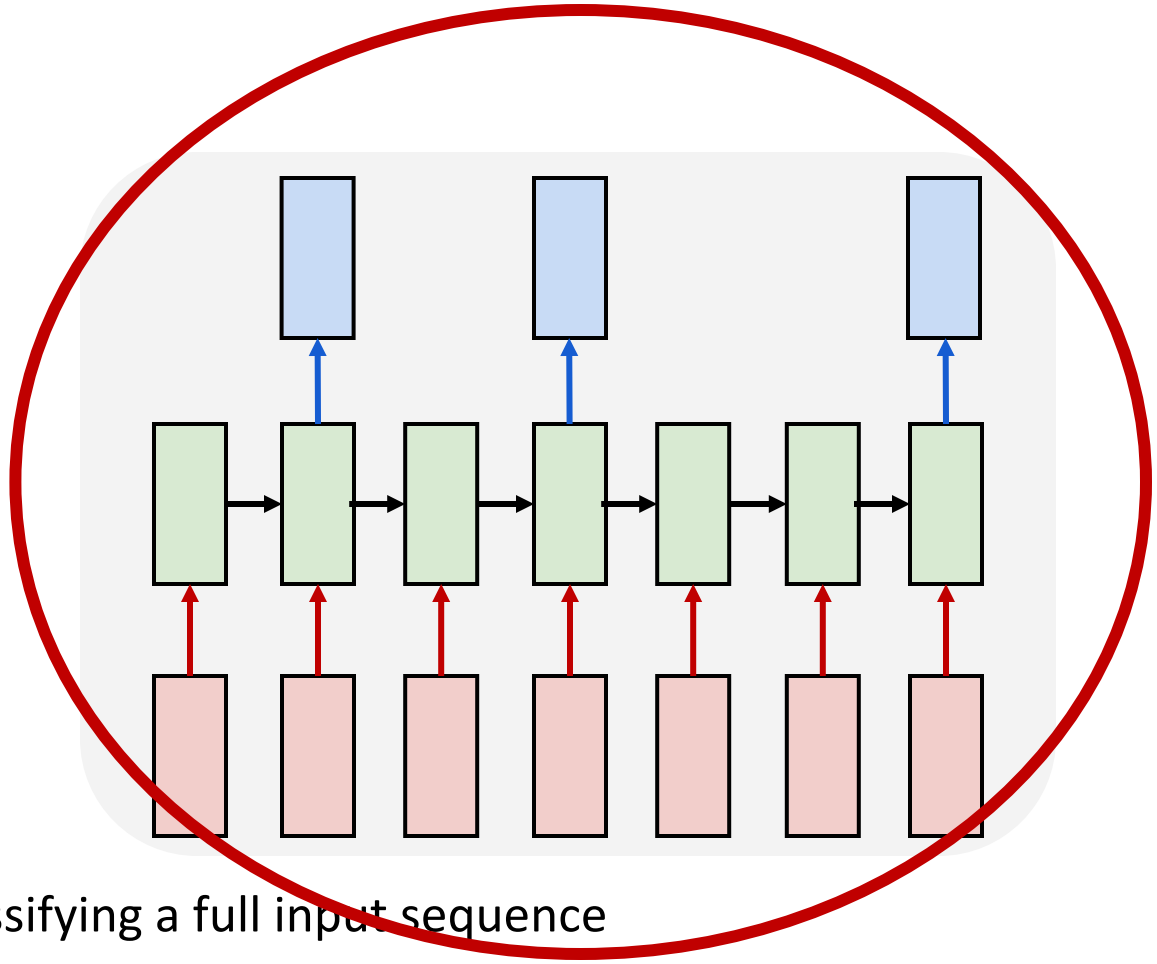
- Define the loss everywhere

$$Loss(Y^{Target}, Y) = \sum_t w_t Xent(Y(t), Phoneme)$$

- Typical weighting scheme for speech: all are equally important
- Problem like question answering: answer only expected after the question ends
  - Only w_T is high, other weights are 0 or low

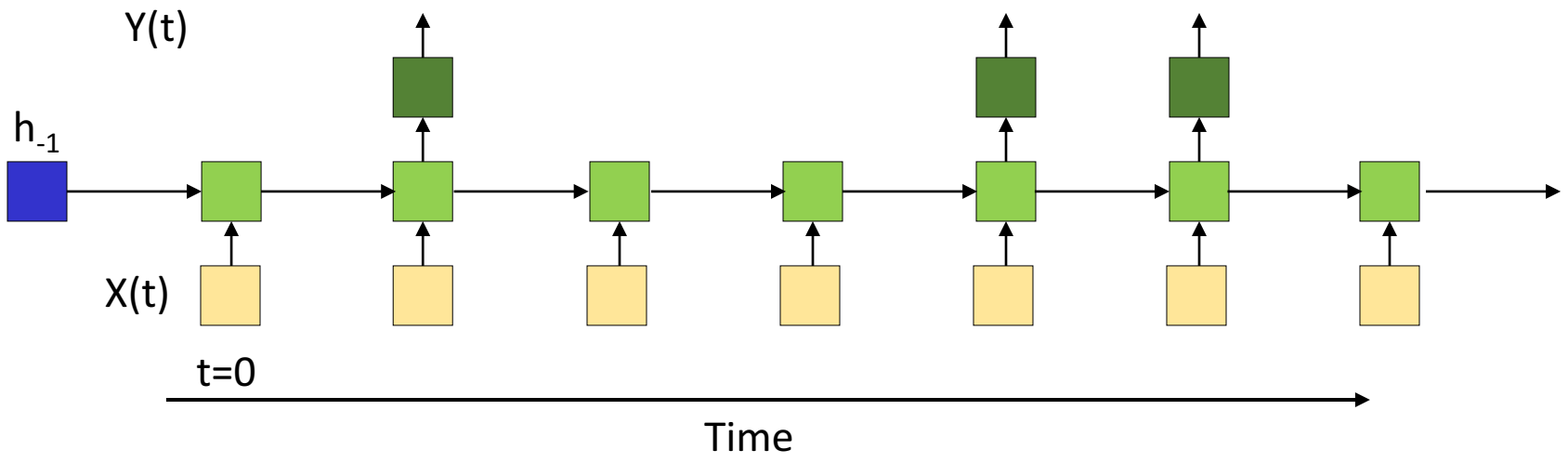
Variants on recurrent nets





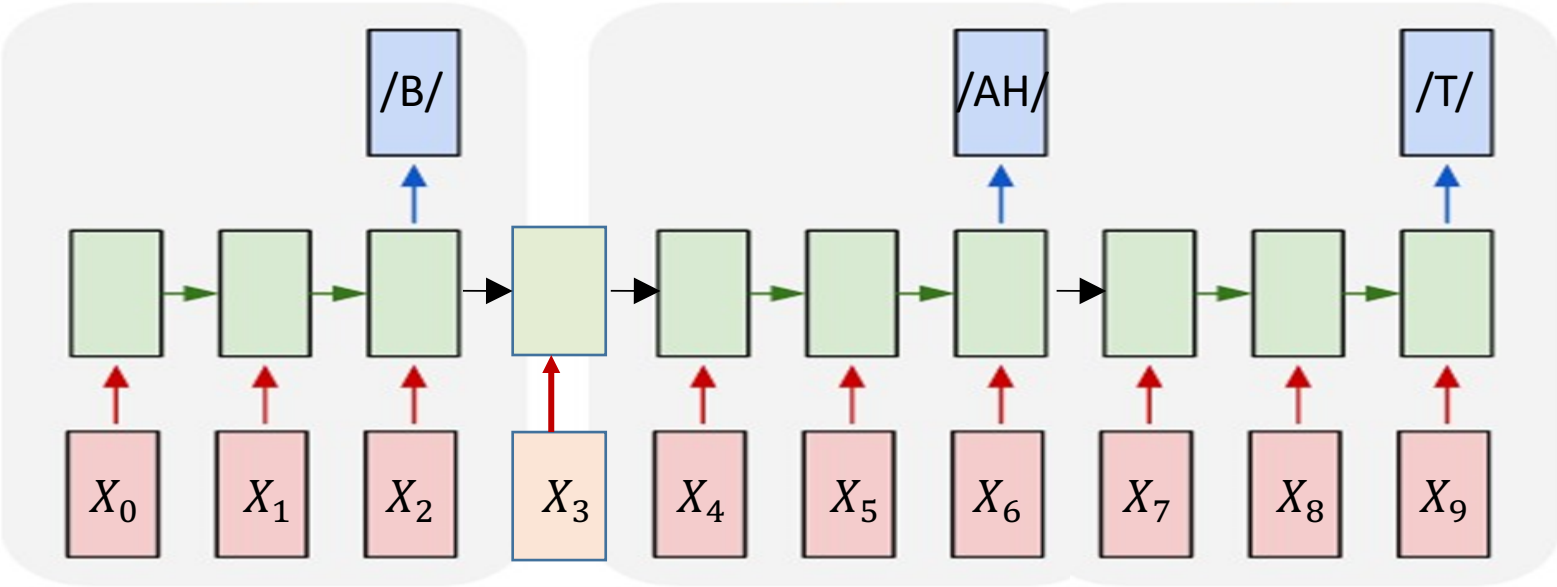
- Sequence classification: Classifying a full input sequence
  - E.g phoneme recognition
- Order synchronous , time asynchronous sequence-to-sequence generation
  - E.g. speech recognition
  - Exact location of output is unknown a priori

Case 1: With alignment



- The input and output sequences happen in the same order
  - Although they may be asynchronous
  - E.g. Speech recognition
    - The input speech corresponds to the phoneme sequence output

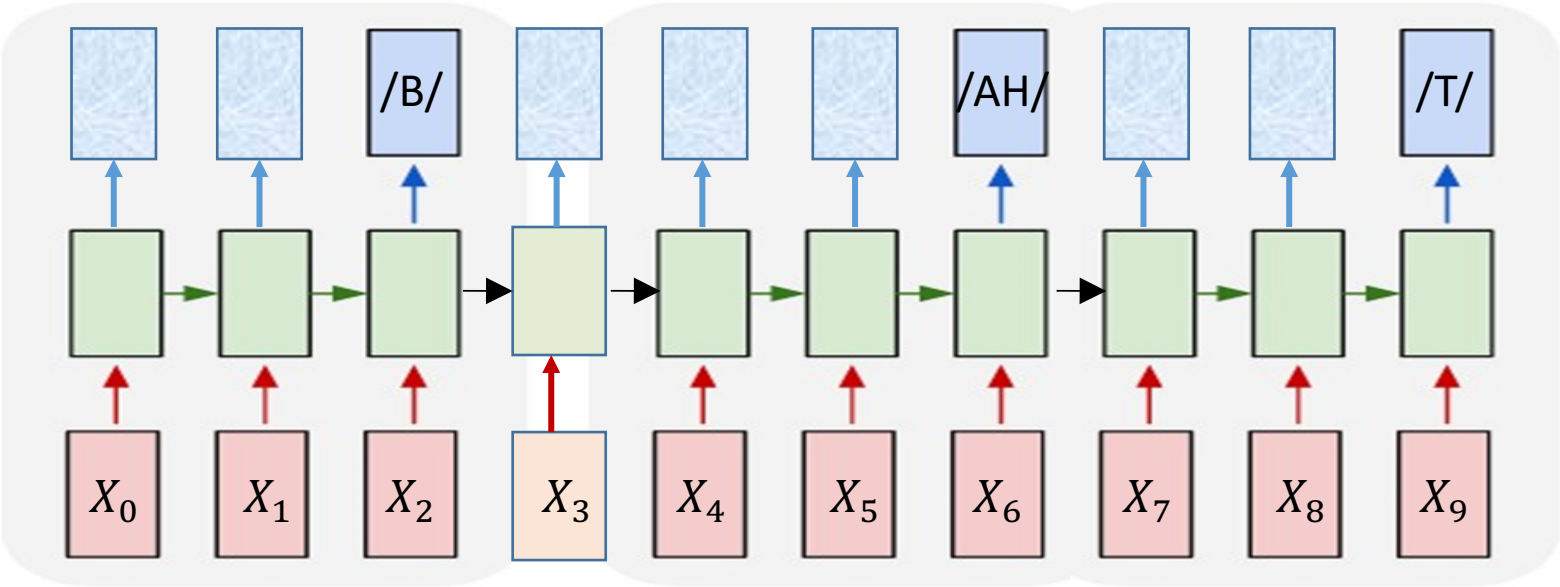
A more complex problem



- Objective: Given a sequence of inputs, asynchronously output a sequence of symbols
  - This is just a simple concatenation of many copies of the simple "output at the end of the input sequence" model we just saw

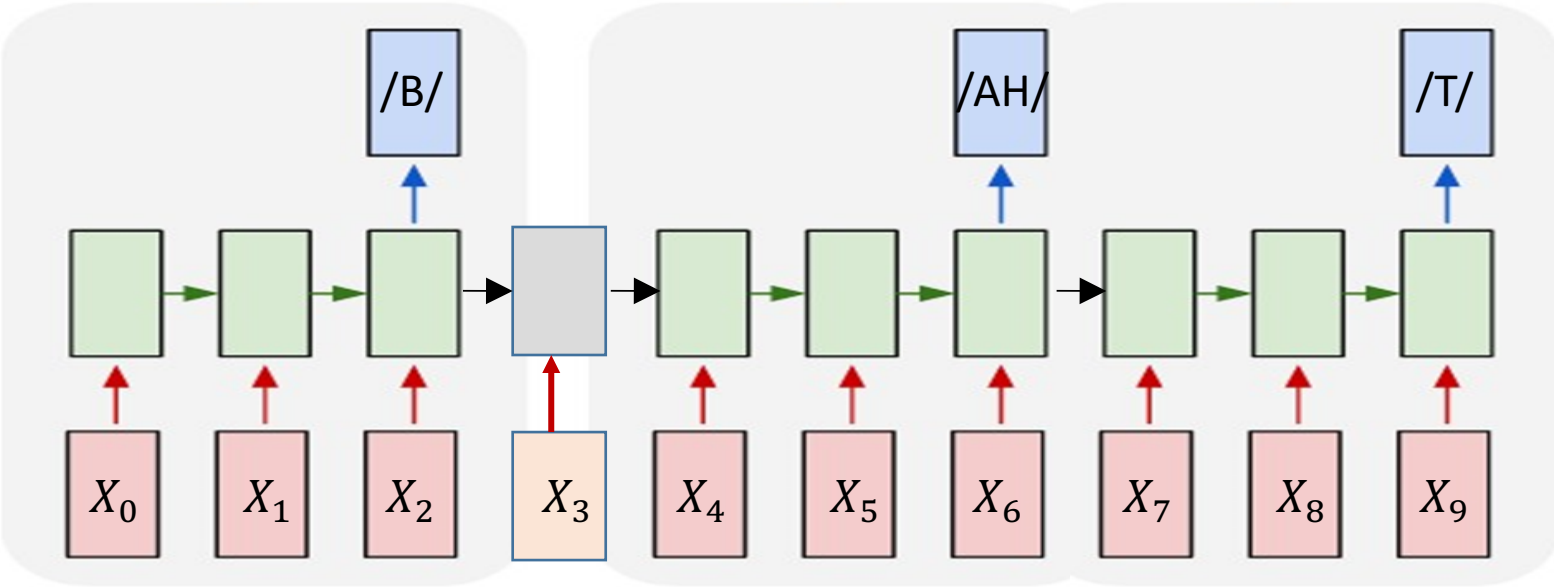
- But this simple extension complicates matters..

The *sequence-to-sequence* problem



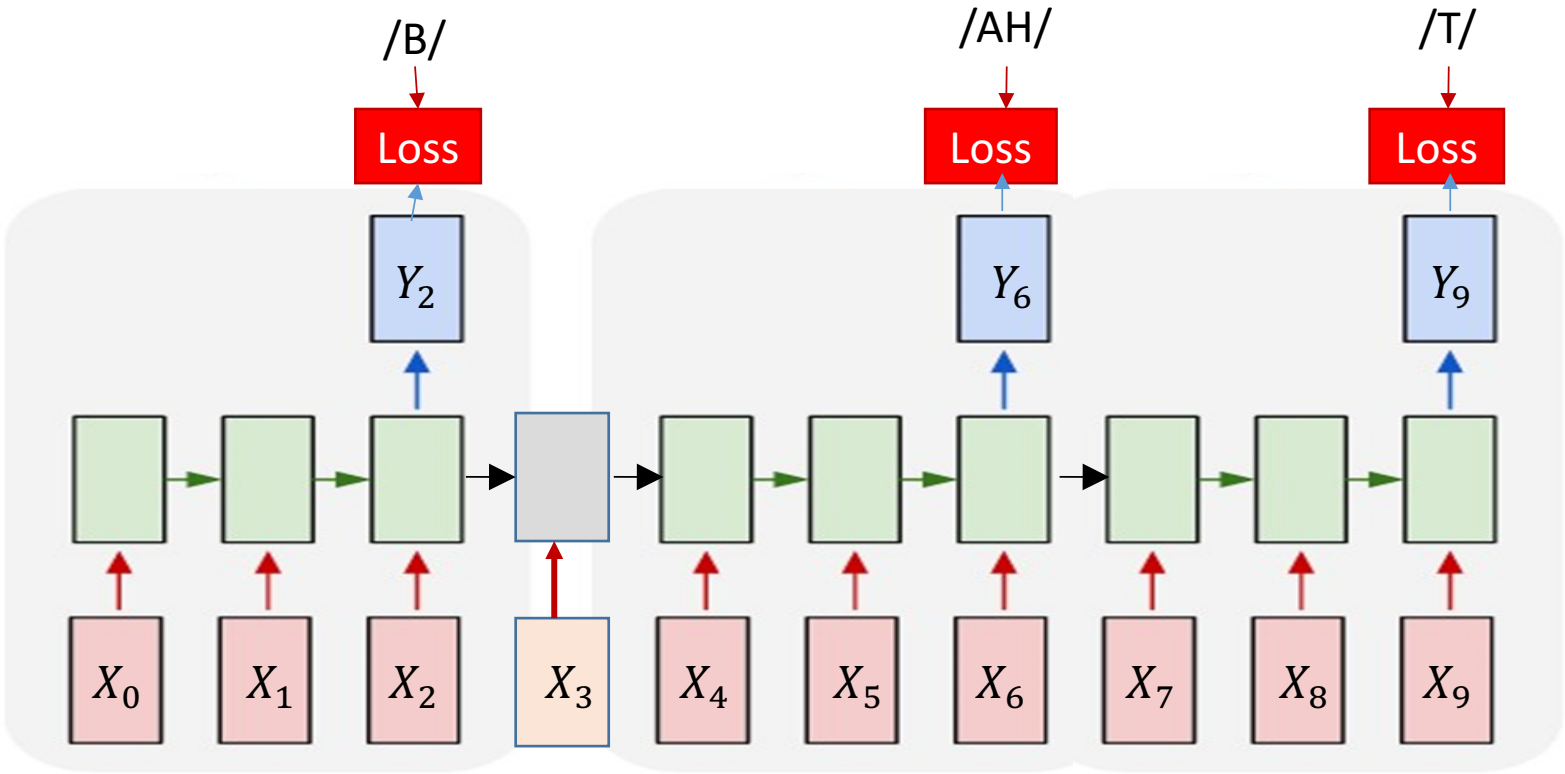
- How do we know *when* to output symbols
  - In fact, the network produces outputs at *every* time
  - *Which* of these are the *real* outputs
    - Outputs that represent the definitive occurrence of a symbol

Training



- Given output symbols *at the right locations*
  - The phoneme /B/ ends at X_2, /AH/ at X_6, /T/ at X_9

Training

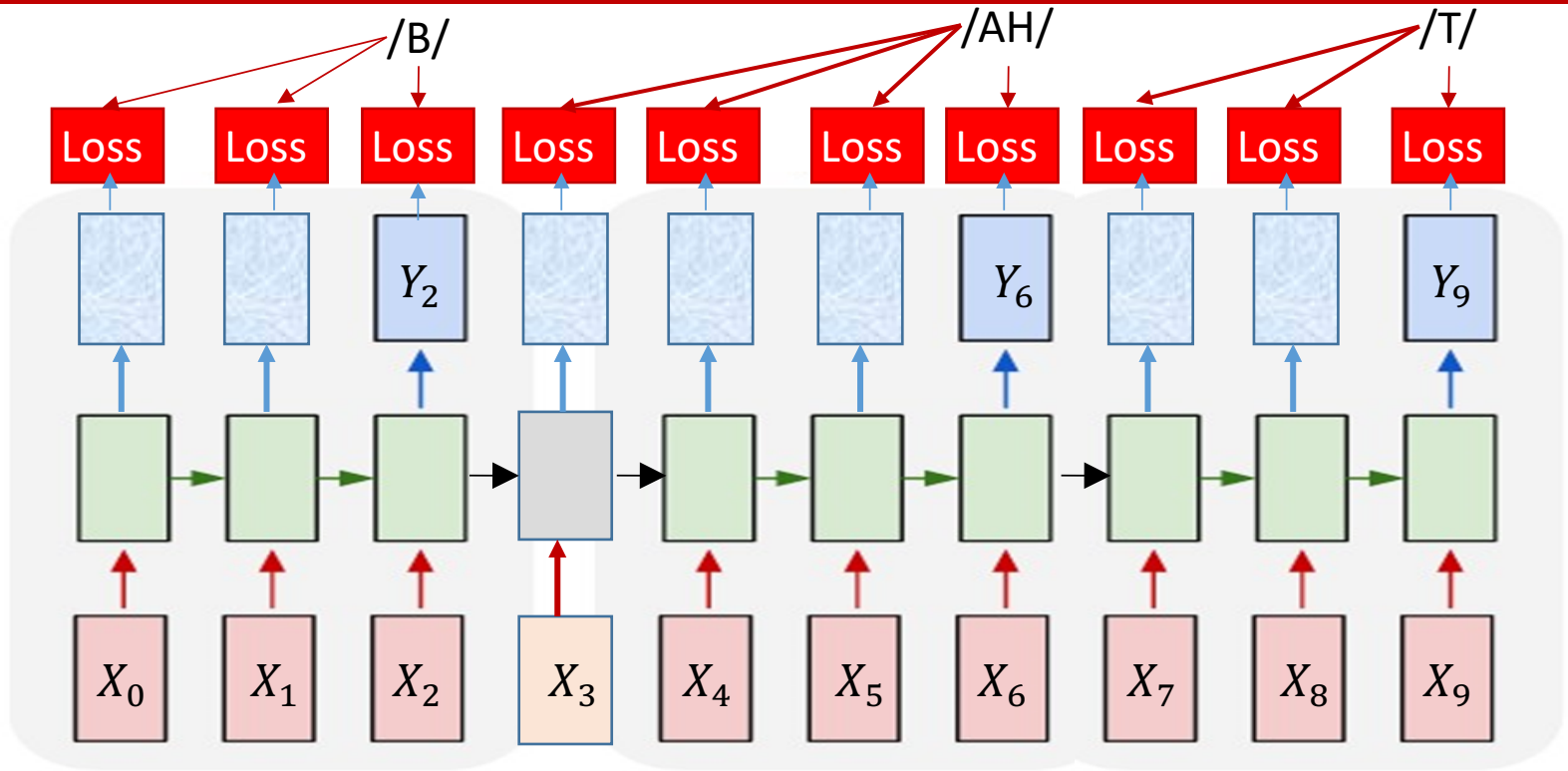


- Either just define Loss as:

$$DIV = Xent(Y_2, B) + Xent(Y_6, AH) + Xent(Y_9, T)$$

- Or..

Training

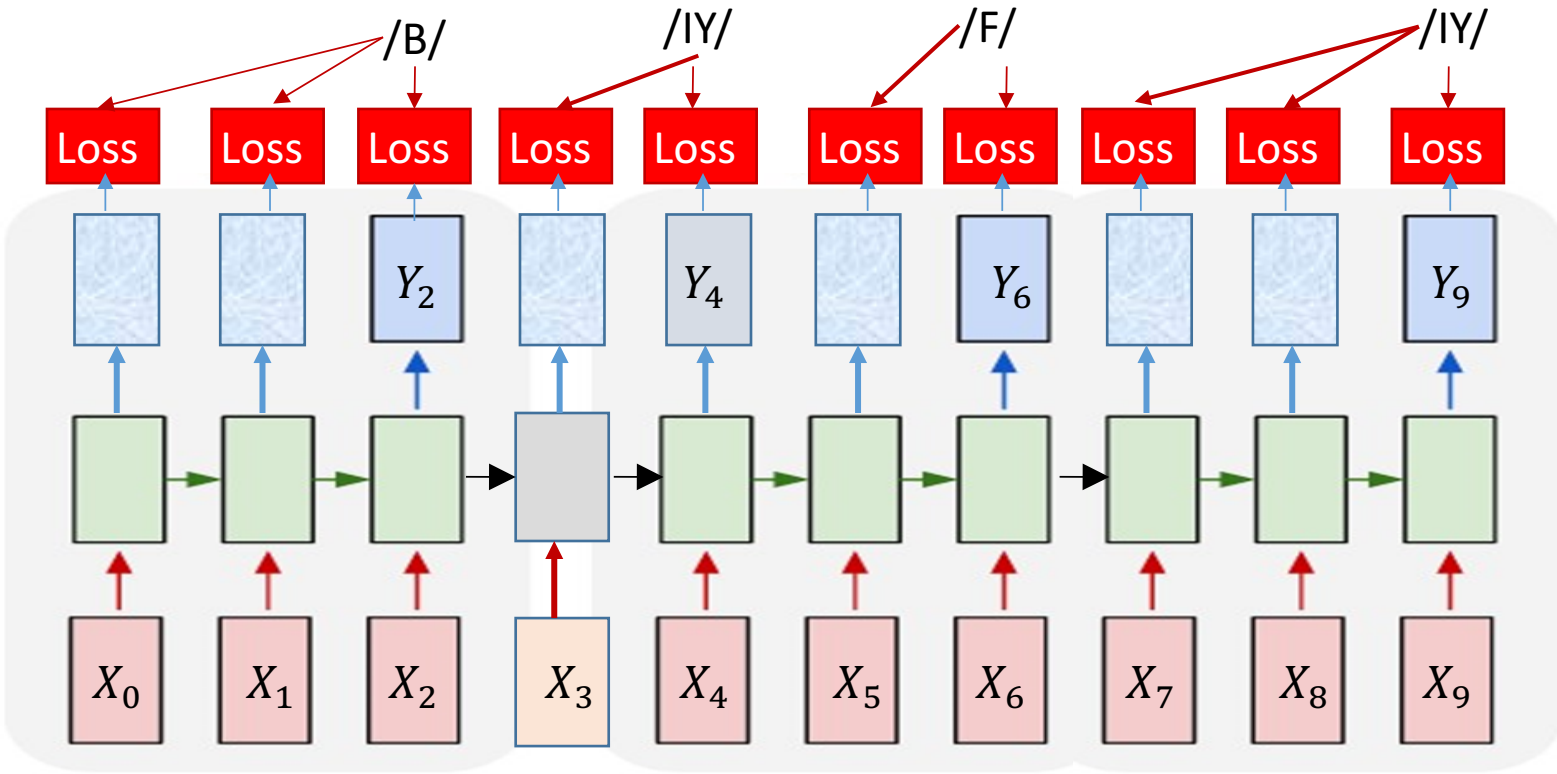


- Either just define Loss as:

$$Loss = Xent(Y_2, B) + Xent(Y_6, AH) + Xent(Y_9, T)$$

- Or *repeat the symbols over their duration*

$$Loss = \sum_t Xent(Y_t, symbol_t) = -\sum_t \log Y(t, symbol_t)$$



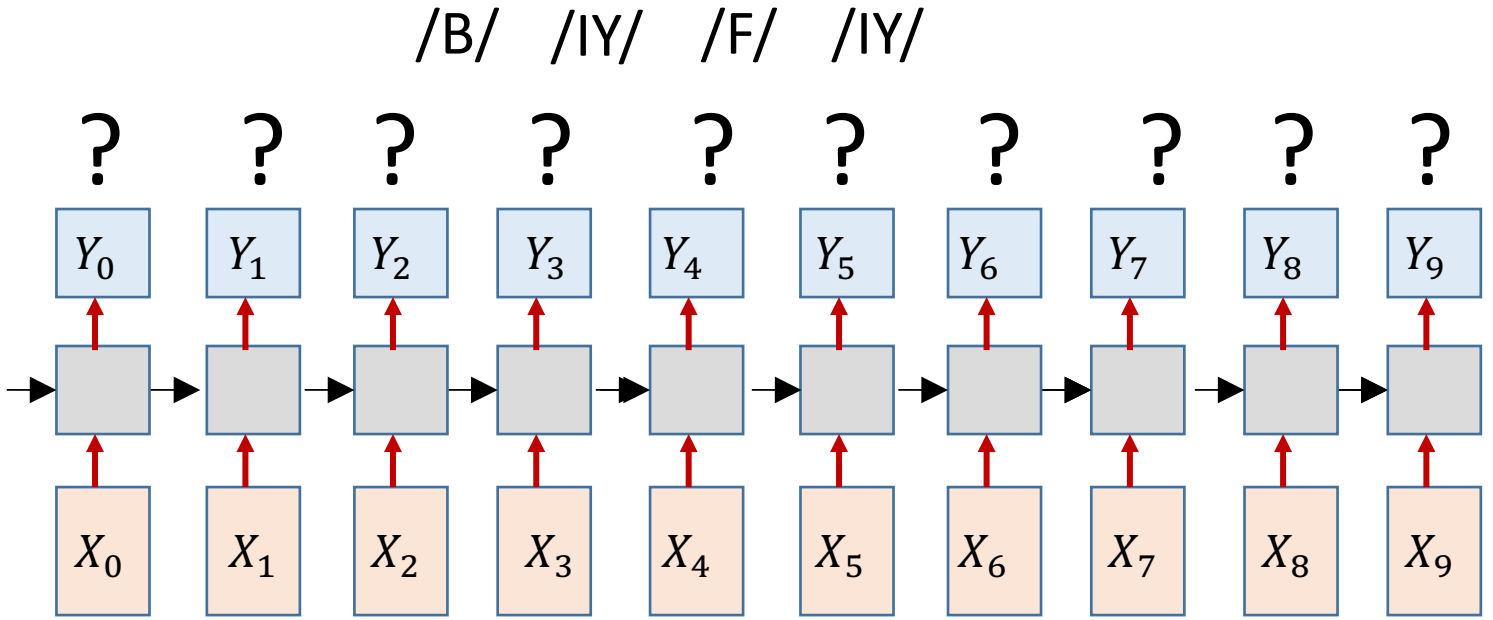
$$Loss = \sum_t Xent(Y_t, symbol_t) = -\sum_t \log Y(t, symbol_t)$$

- The gradient w.r.t the t-th output vector Y_t

$$\nabla_{Y_t} Loss = \left[0 \quad 0 \quad \dots \quad \frac{-1}{Y(t, symbol_t)} \quad 0 \quad \dots \quad 0\right]$$

- Zeros except at the component corresponding to the target

Problem: No timing information provided



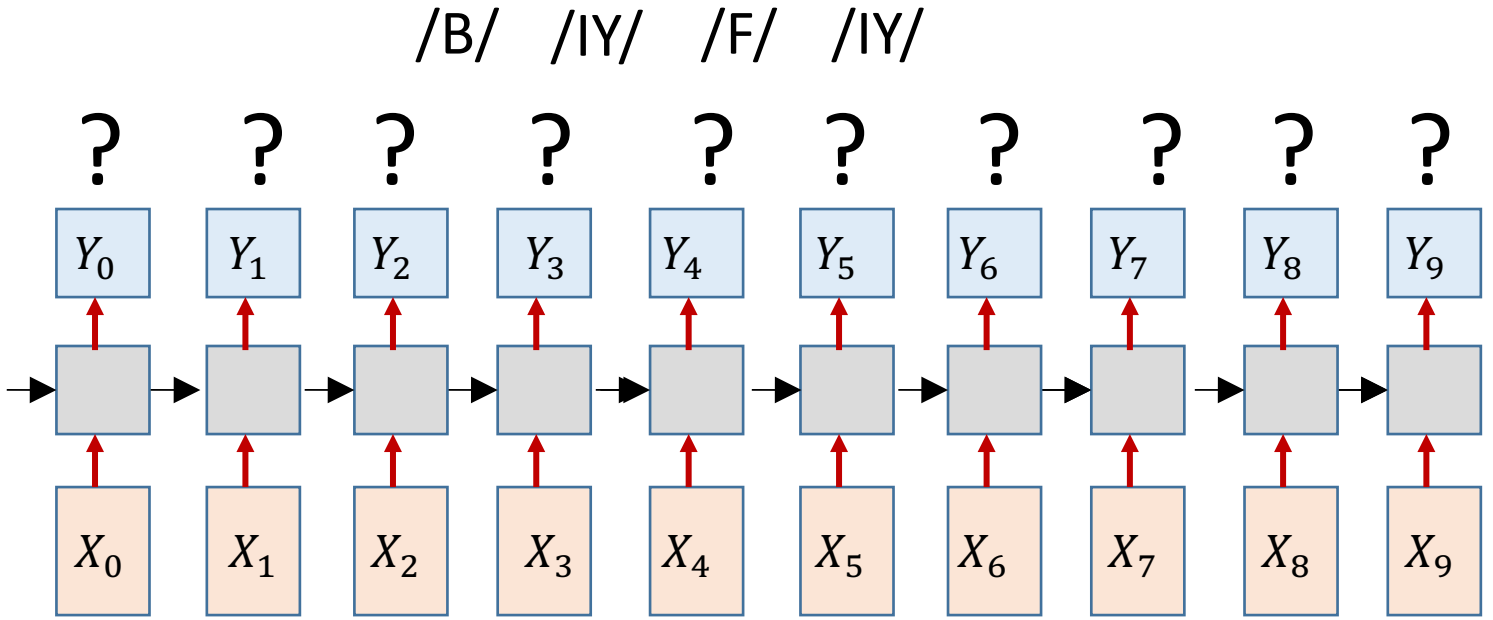
- Only the sequence of output symbols is provided for the training data
  - But no indication of which one occurs where
- How do we compute the Loss?
  - And how do we compute its gradient w.r.t. Y_t

Connectionist Temporal Classification (CTC)

Labelling Unsegmented Sequence Data with Recurrent Neural Networks

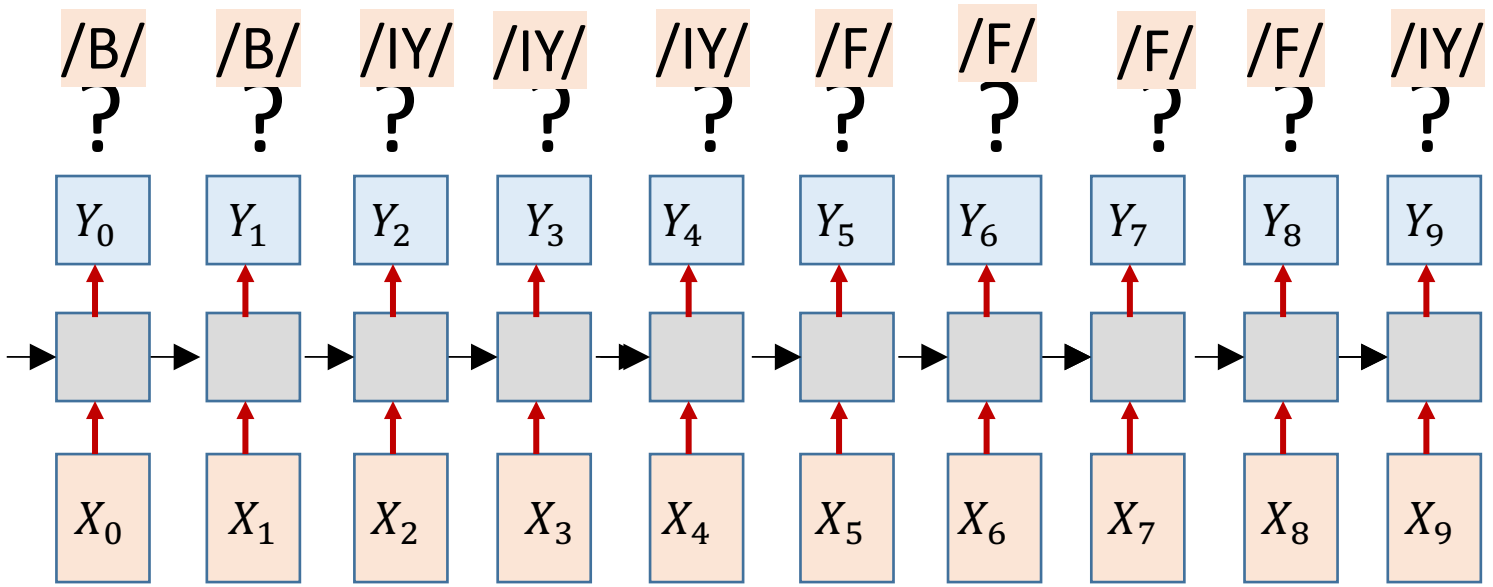
Alex Graves et al., ICML 2006.

Problem: No timing information provided



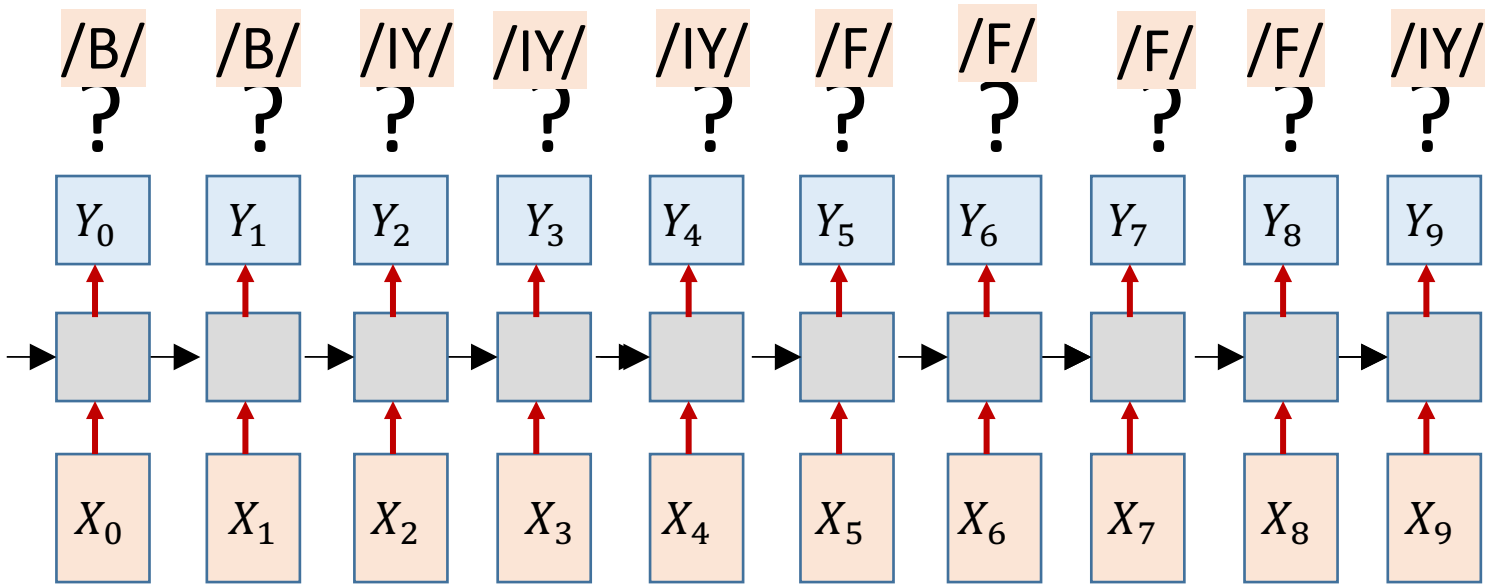
- Only the sequence of output symbols is provided for the training data
  - But no indication of which one occurs where
- How do we compute the Loss?
  - And how do we compute its gradient w.r.t. Y_t

Solution 1: *Guess the alignment*



- Initialize: Assign an initial alignment
  - Either randomly, based on some heuristic, or any other rationale
- Iterate:
  - Train the network using the current alignment
  - *Reestimate* the alignment for each training instance
    - Using the decoding methods already discussed

Solution 1: *Guess the alignment*



- Initialize: Assign an initial alignment
  - Either randomly, based on some heuristic, or any other rationale
- Iterate:
  - Train the network using the current alignment
  - *Reestimate* the alignment for each training instance
    - Using the decoding methods already discussed

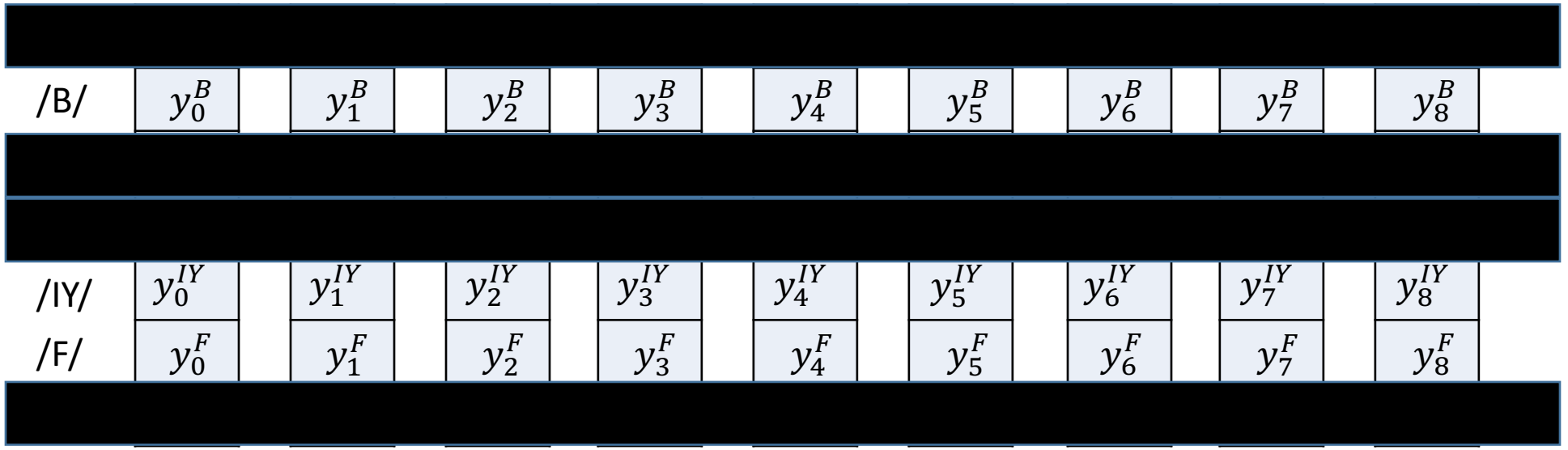
Estimating an alignment

- Given:
  - The unaligned K-length symbol sequence $S = S_0 \ldots S_{K-1}$ (e.g. /B/ /IY/ /F/ /IY/)
  - An N-length input ($N \geq K$)
  - And a (trained) recurrent network

- Find:
  - An N-length expansion $s_0 \ldots s_{N-1}$ comprising the symbols in S in strict order
    - e.g. $S_0 S_1 S_1 S_2 S_3 S_3 \ldots S_{K-1}$
      - i.e. $s_0 = S_0, s_2 = S_1, S_3 = S_1, s_4 = S_2, s_5 = S_3, \ldots s_{N-1} = S_{K-1}$
    - E.g. /B/ /B/ /IY/ /IY/ /IY/ /F/ /F/ /F/ /F/ /IY/ ..

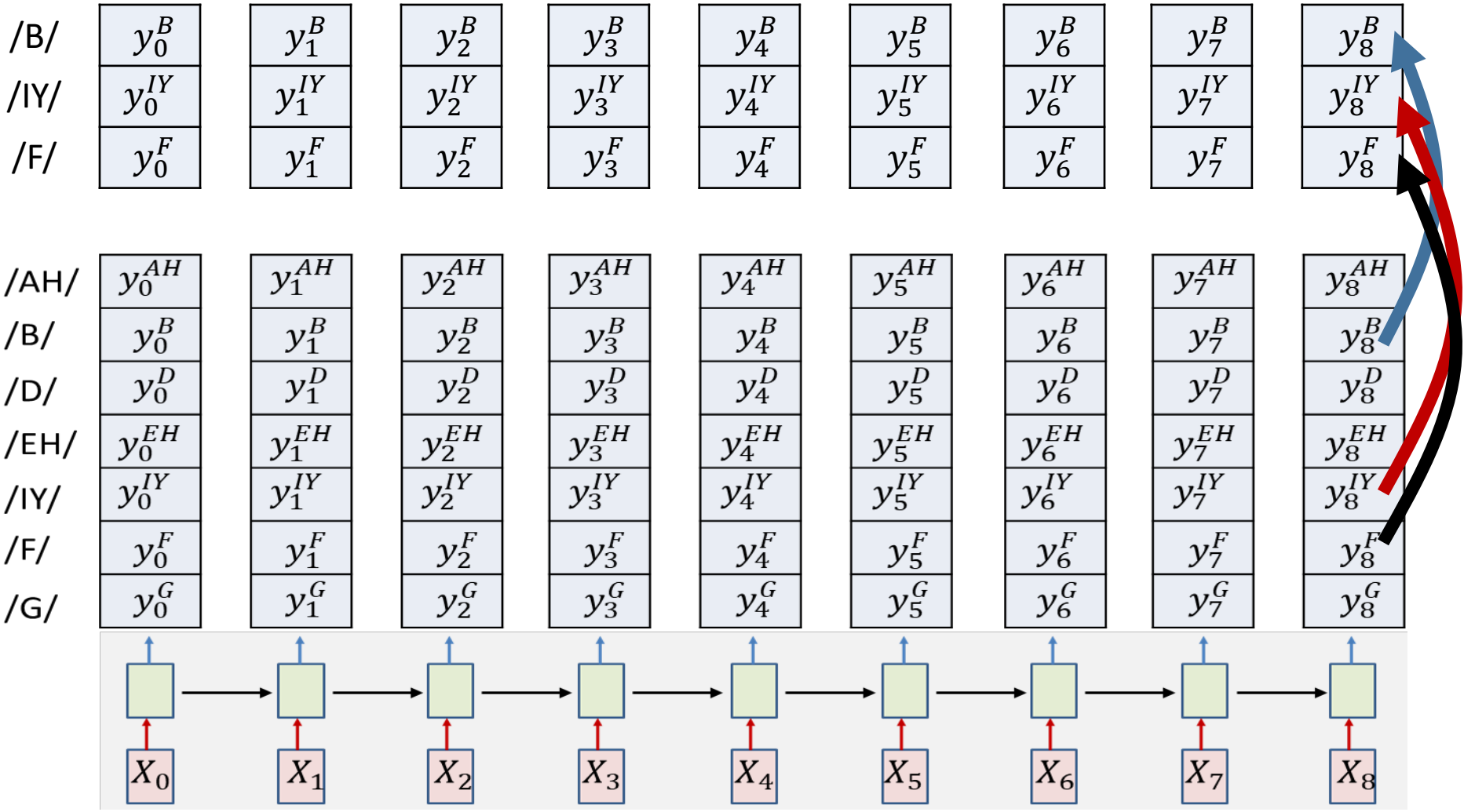
- Outcome: an *alignment* of the target symbol sequence $S_0 \ldots S_{K-1}$ to the input $X_0 \ldots X_{N-1}$

Constraining the alignment: Try 1



- Block out all rows that do not include symbols from the target sequence
  - E.g. Block out rows that are not /B/ /IY/ or /F/

Blocking out unnecessary outputs



Compute the entire output (for all symbols)

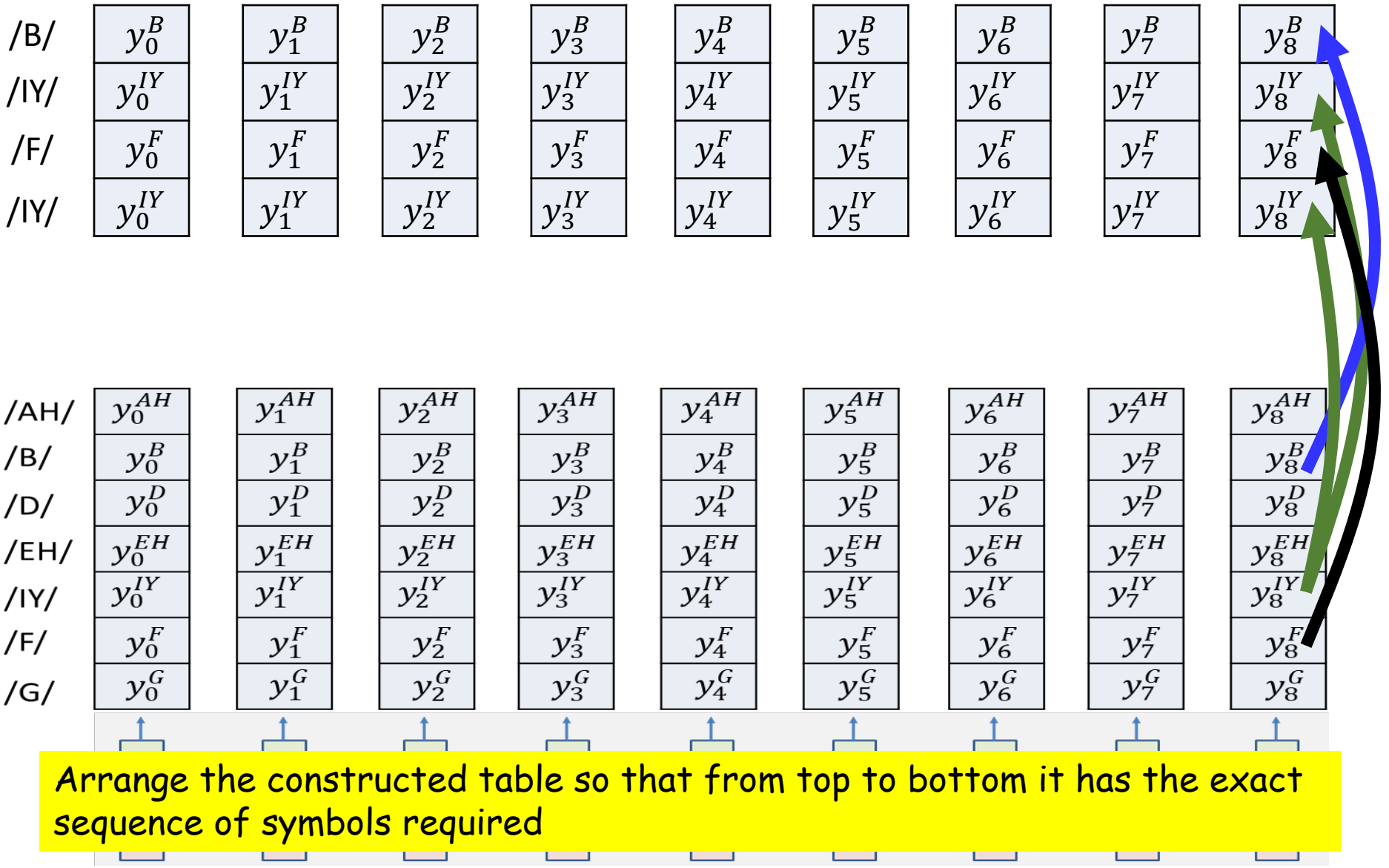
Copy the output values for the target symbols into the secondary reduced structure

Constraining the alignment: Try 1

/B/	y_0^B	y_1^B	y_2^B	y_3^B	y_4^B	y_5^B	y_6^B	y_7^B	y_8^B
/IY/	y_0^{IY}	y_1^{IY}	y_2^{IY}	y_3^{IY}	y_4^{IY}	y_5^{IY}	y_6^{IY}	y_7^{IY}	y_8^{IY}
/F/	y_0^F	y_1^F	y_2^F	y_3^F	y_4^F	y_5^F	y_6^F	y_7^F	y_8^F

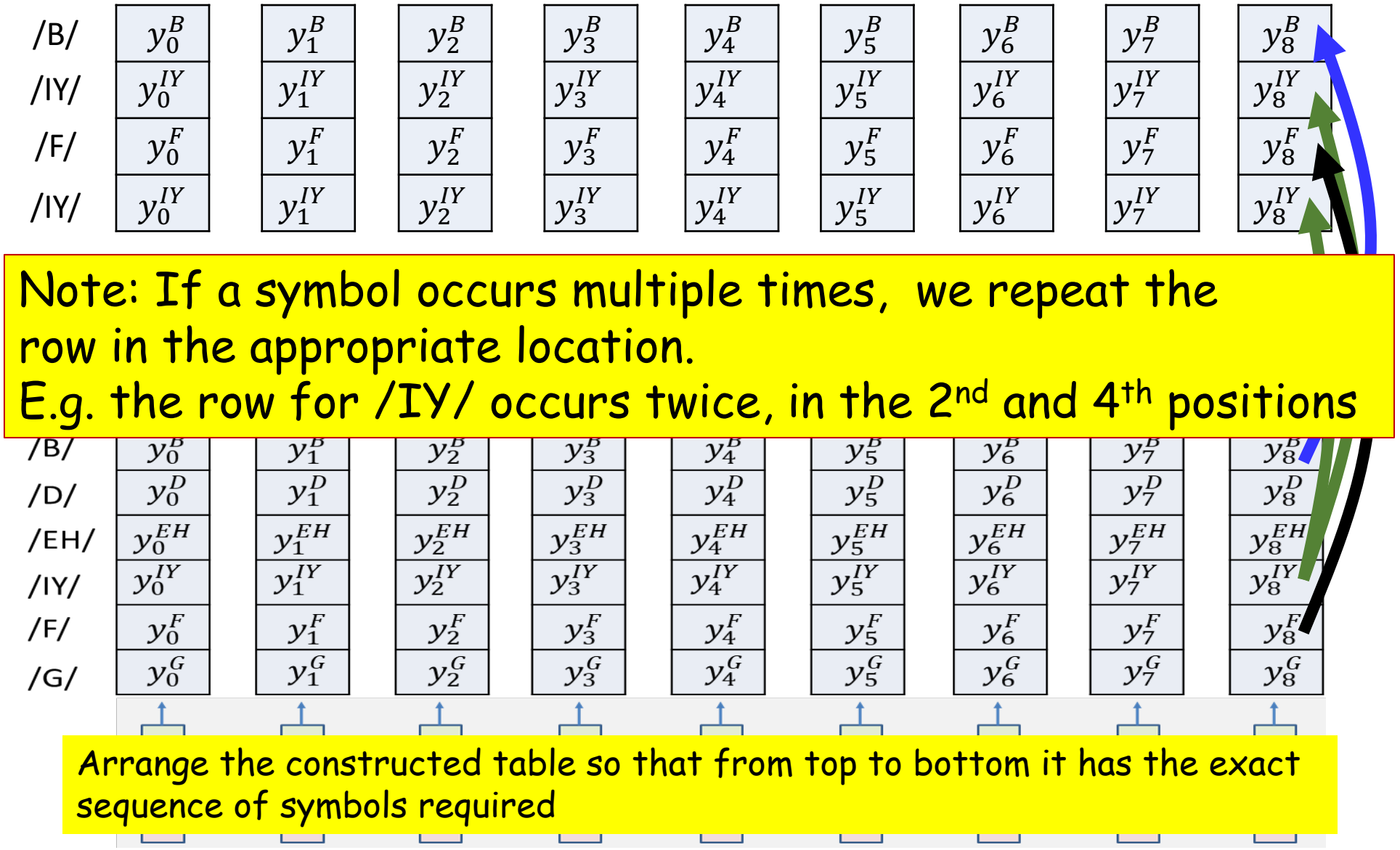
- Only decode on reduced grid
  - We are now assured that only the appropriate symbols will be hypothesized

Try 2: Explicitly arrange the constructed table



Arrange the constructed table so that from top to bottom it has the exact sequence of symbols required

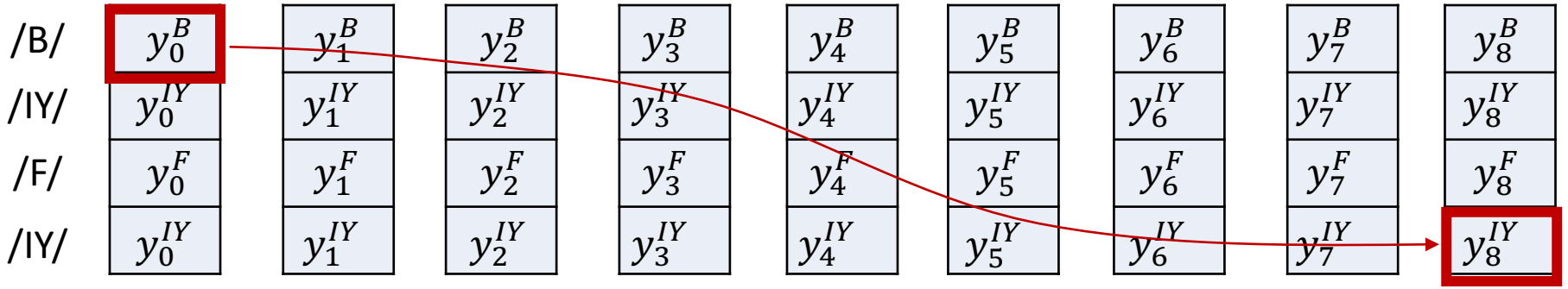
Try 2: Explicitly arrange the constructed table



Note: If a symbol occurs multiple times, we repeat the row in the appropriate location.
E.g. the row for /IY/ occurs twice, in the 2nd and 4th positions

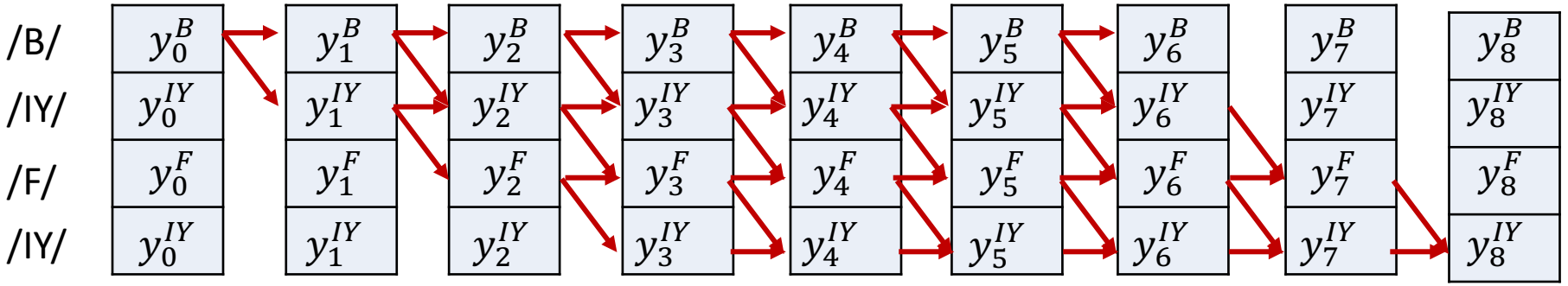
Arrange the constructed table so that from top to bottom it has the exact sequence of symbols required

Explicitly constrain alignment



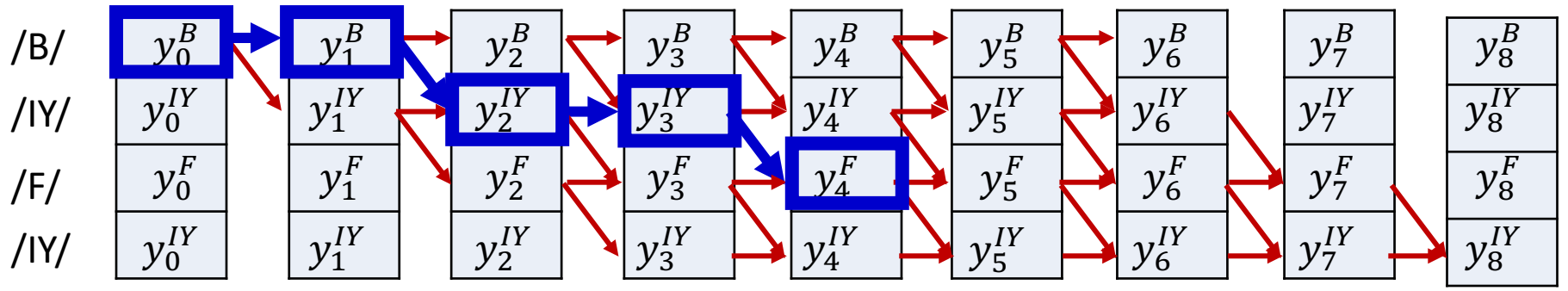
- Constrain that the first symbol in the decode *must* be the top left block
- The last symbol *must* be the bottom right
- The rest of the symbols must follow a sequence that *monotonically* travels down from top left to bottom right
  - I.e. symbol chosen at any time is at the same level or at the next level to the symbol at the previous time
- This guarantees that the sequence *is* an expansion of the target sequence
  - /B/ /IY/ /F/ /IY/ in this case

Explicitly constrain alignment



- Constrain that the first symbol in the decode *must* be the top left block
- The last symbol *must* be the bottom right
- The rest of the symbols must follow a sequence that *monotonically* travels down from top left to bottom right
  - I.e. symbol chosen at any time is at the same level or at the next level to the symbol at the previous time
- This guarantees that the sequence *is* an expansion of the target sequence
  - /B/ /IY/ /F/ /IY/ in this case

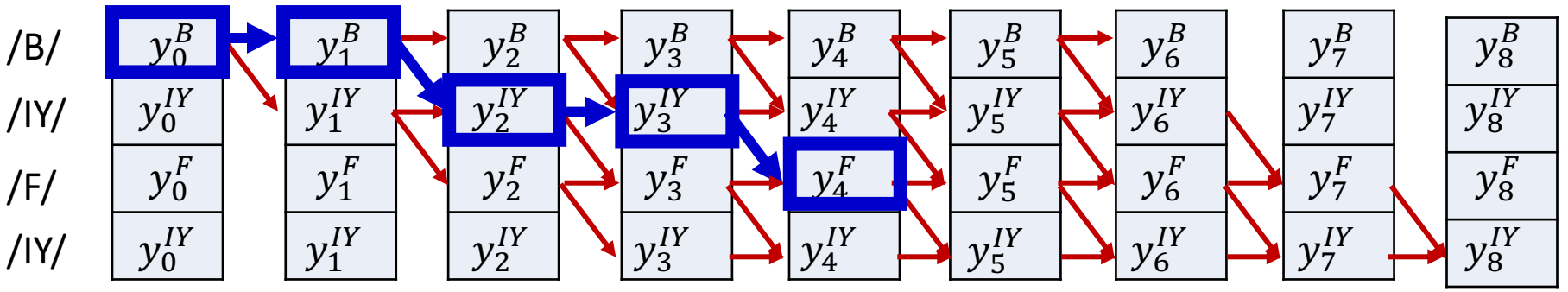
Path Score (probability)



- Compose a graph such that every path in the graph from source to sink represents a valid alignment
  - Which maps on to the target symbol sequence (/B//AH//T/)
- Edge scores are 1
- Node scores are the probabilities assigned to the symbols by the neural network
- **The "score" of a path is the product of the probabilities of all nodes along the path**
- **E.g. the probability of the marked path is**

$$Scr(Path) = y_0^B y_1^B y_2^{IY} y_3^{IY} y_4^F$$

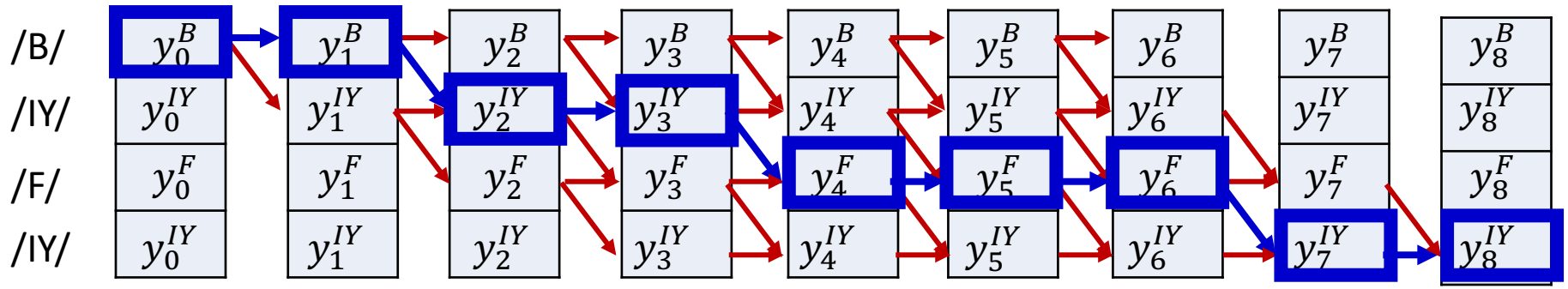
Path Score (probability)



- Compose a graph such that every path in the graph from source to sink represents a valid alignment
  - Which maps on to the target symbol sequence (/B//AH//T/)
- Edge scores are 1
- Node scores are the probabilities assigned to the symbols by the neural network
- **The "score" of a path is the product of the probabilities of all nodes along the path**
- **E.g. the probability of the marked path is**

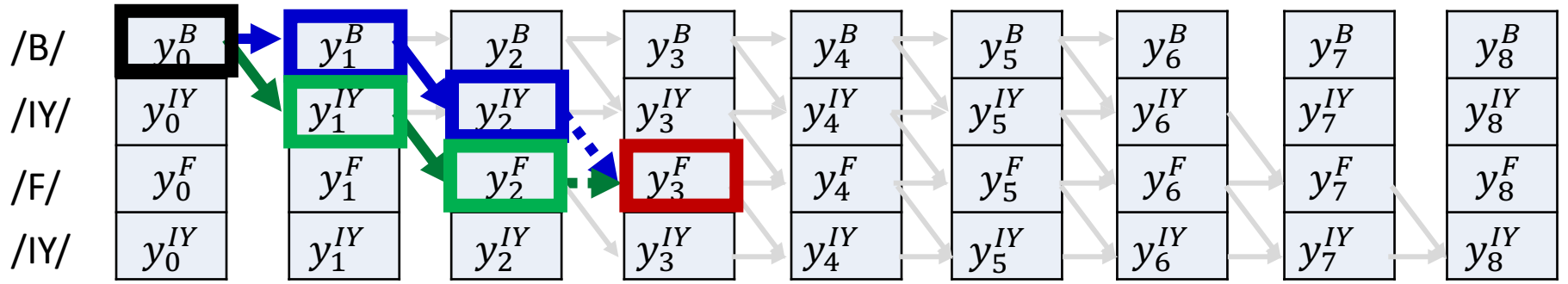
Figure shows a typical end-to-end path. There are an exponential number of such paths. Challenge: Find the path with the highest score (probability)

Explicitly constrain alignment



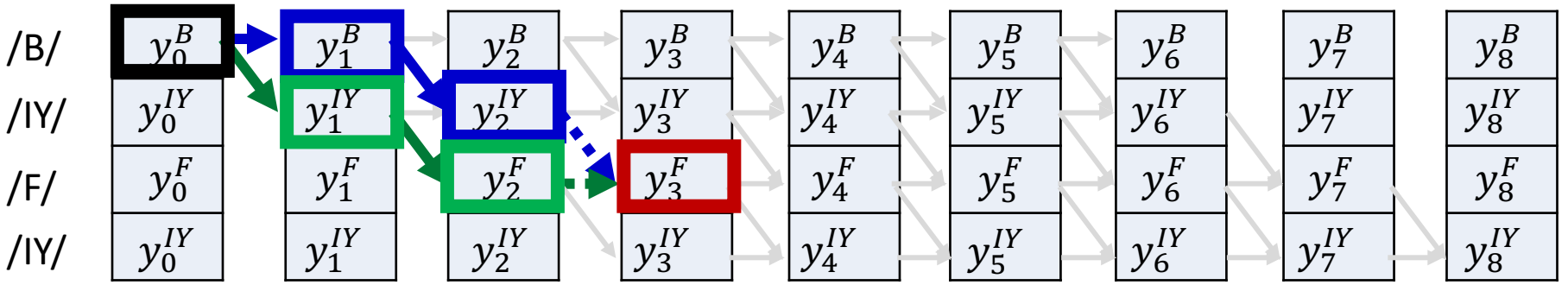
- Find the most probable path from source to sink using any dynamic programming algorithm
  - E.g. The Viterbi algorithm

Viterbi algorithm Basic idea



- The best path to any node *must* be an extension of the best path to one of its parent nodes
  - Any other path would necessarily have a lower probability
- The best parent is simply the parent with the best-scoring best path

Viterbi algorithm Basic idea



$$BestPath(y_0^B \rightarrow y_3^F) = BestPath(y_0^B \rightarrow y_2^{IY})y_3^F$$

$$or \quad BestPath(y_0^B \rightarrow y_2^F)y_3^F$$

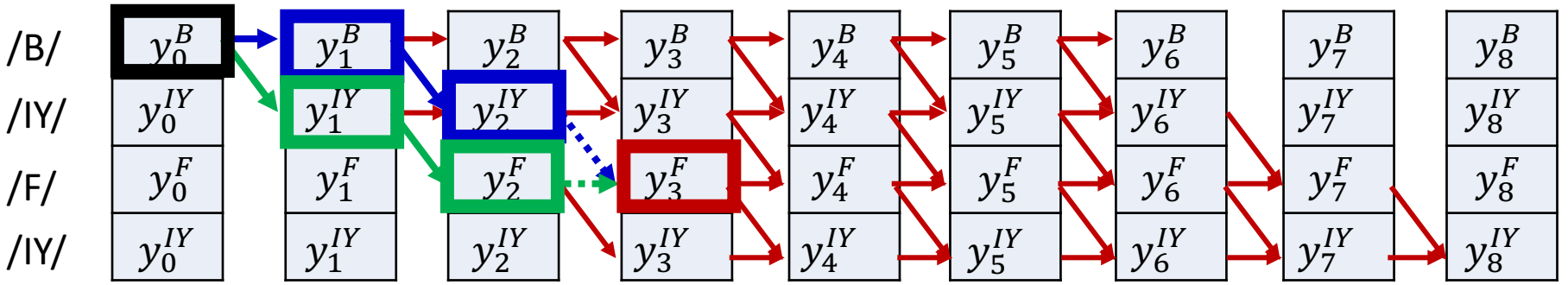
$$BestPath(y_0^B \rightarrow y_3^F) = Best_{Parent \in (y_2^{IY}, y_2^F)} \left(BestPath(y_0^B \rightarrow Parent)\right) y_3^F$$

$$BestPath(y_0^B \rightarrow y_3^F) = BestPath(y_0^B \rightarrow BestParent)y_3^F$$

- The best parent is simply the parent with the best-scoring best path

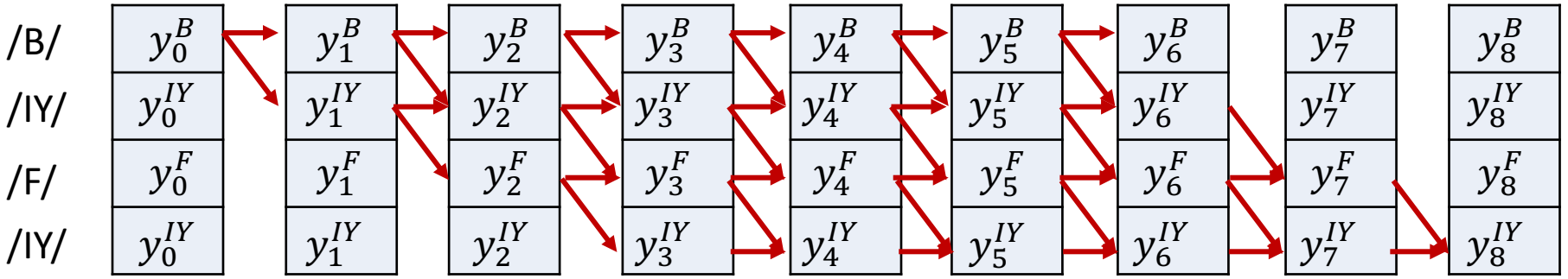
$$BestParent = argmax_{Parent \in (y_2^{IY}, y_2^F)}(Score(BestPath(y_0^B \rightarrow Parent)))$$

Viterbi algorithm



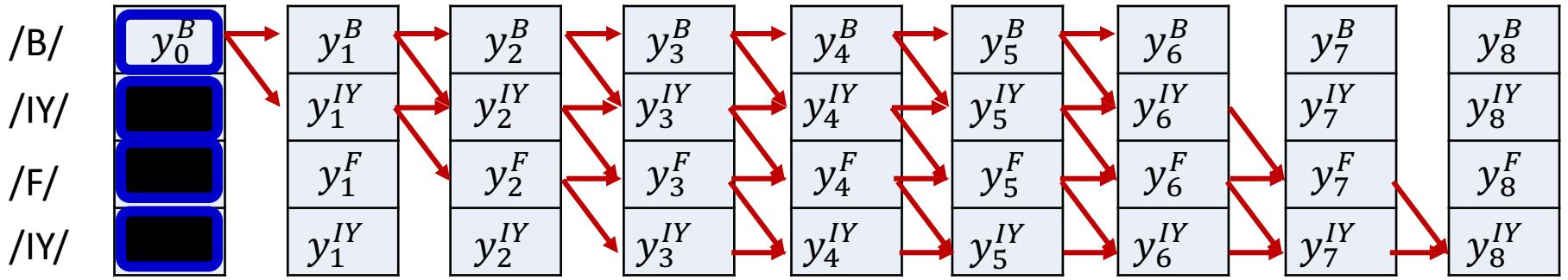
- Dynamically track the best path (and the score of the best path) from the source node to every node in the graph
  - At each node, keep track of
    - The best incoming parent edge
    - The score of the best path from the source to the node through this best parent edge
- Eventually compute the best path from source to sink

Viterbi algorithm



- First, some notation:
- $y_t^{S(r)}$ is the probability of the target symbol assigned to the r-th row in the t-th time (given inputs $X_0 \ldots X_t$)
  - E.g., S(0) = /B/
    - The scores in the 0th row have the form y_t^B
  - E.g. S(1) = S(3) = /IY/
    - The scores in the 1st and 3rd rows have the form y_t^{IY}
  - E.g. S(2) = /F/
    - The scores in the 2nd row have the form y_t^F

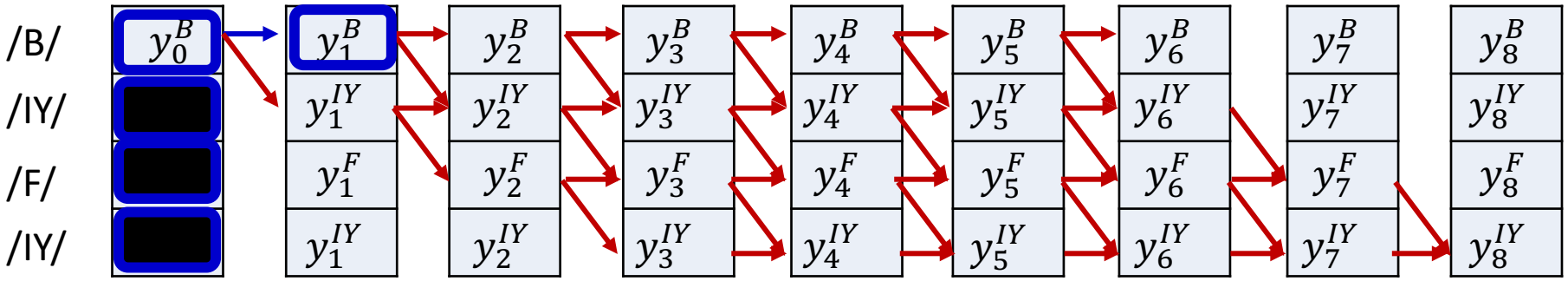
Viterbi algorithm



- Initialization:

$$Bscr(0,0) = y_0^{S(0)},\ Bscr(0,i) = -\infty,\ \ i = 1 \ldots K-1$$

Viterbi algorithm



- Initialization:

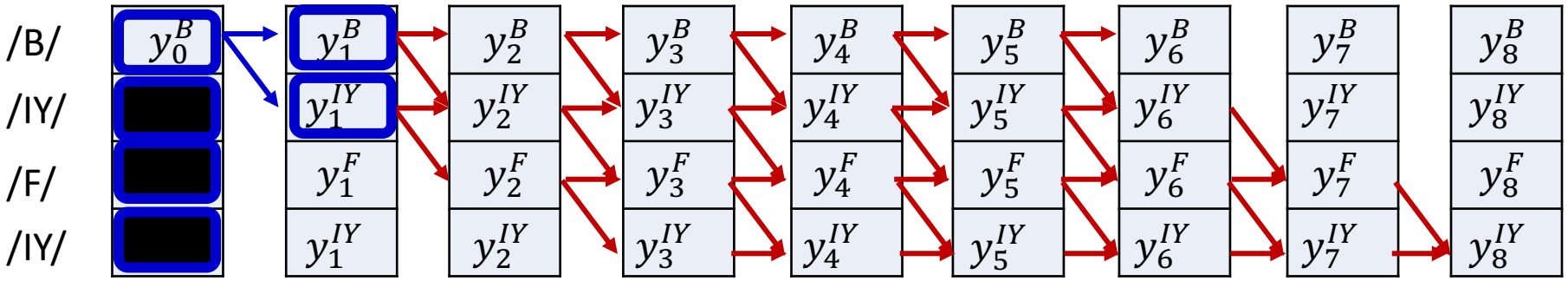
$$BP(0,i) = null,\ i = 0 \ldots K-1$$

$$Bscr(0,0) = y_0^{S(0)},\ Bscr(0,i) = -\infty,\ i = 1 \ldots K-1$$

- for $t = 1 \ldots T-1$

$$BP(t,0) = 0;\ Bscr(t,0) = Bscr(t-1,0) \times y_t^{S(0)}$$

Viterbi algorithm



- Initialization:

$$BP(0,i) = null,\ i = 0 \ldots K-1$$

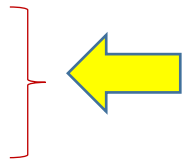
$$Bscr(0,0) = y_0^{S(0)},\ Bscr(0,i) = -\infty,\ i = 1 \ldots K-1$$

- for $t = 1 \ldots T-1$

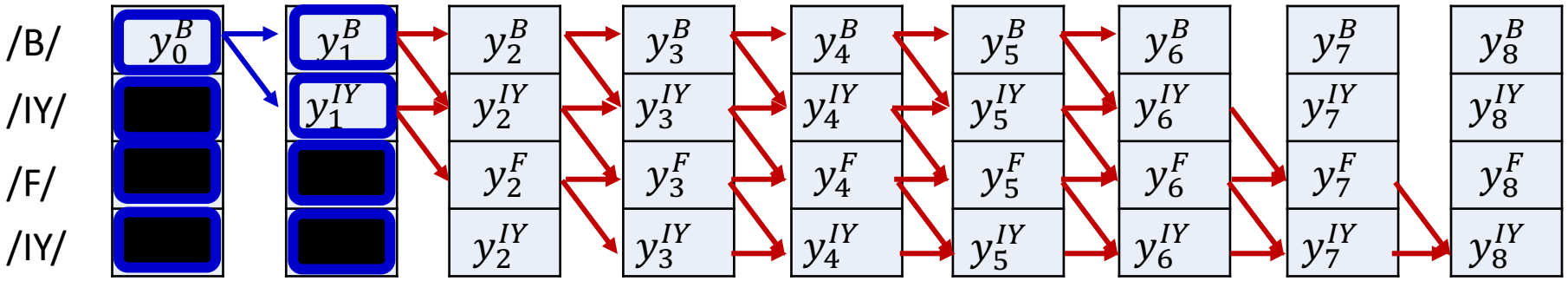
$$BP(t,0) = 0;\ Bscr(t,0) = Bscr(t-1,0) \times y_t^{S(0)}$$

for $l = 1 \ldots K-1$

- $BP(t,l) = \begin{pmatrix} l-1: & if\ (Bscr(t-1,l-1) > Bscr(t-1,l)); \\ l: & else \end{pmatrix}$
- $Bscr(t,l) = Bscr(BP(t,l)) \times y_t^{S(l)}$



Viterbi algorithm



- Initialization:

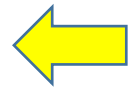
$$Bscr(0,0) = y_0^{S(0)},\ Bscr(0,i) = -\infty,\ \ i = 1 \ldots K-1$$

- for $t = 1 \ldots T-1$

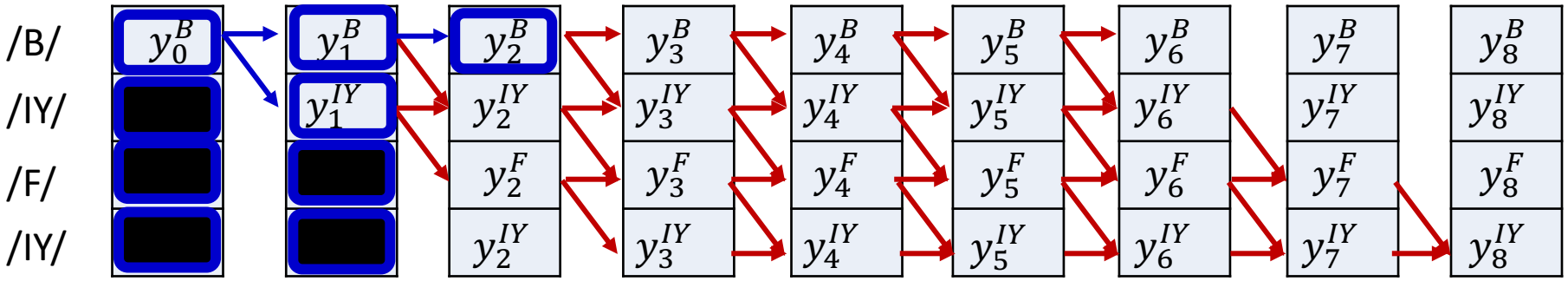
$$BP(t,0) = 0;\ Bscr(t,0) = Bscr(t-1,0) \times y_t^{S(0)}$$

for $l = 1 \ldots K-1$

- $BP(t,l) = \begin{pmatrix} l-1: & if\ \left(Bscr(t-1,l-1) > Bscr(t-1,l)\right); \\ l: & else \end{pmatrix}$
- $Bscr(t,l) = Bscr(BP(t,l)) \times y_t^{S(l)}$



Viterbi algorithm



- Initialization:

$$BP(0,i) = null,\ i = 0 \ldots K-1$$

$$Bscr(0,0) = y_0^{S(0)},\ Bscr(0,i) = -\infty,\ i = 1 \ldots K-1$$

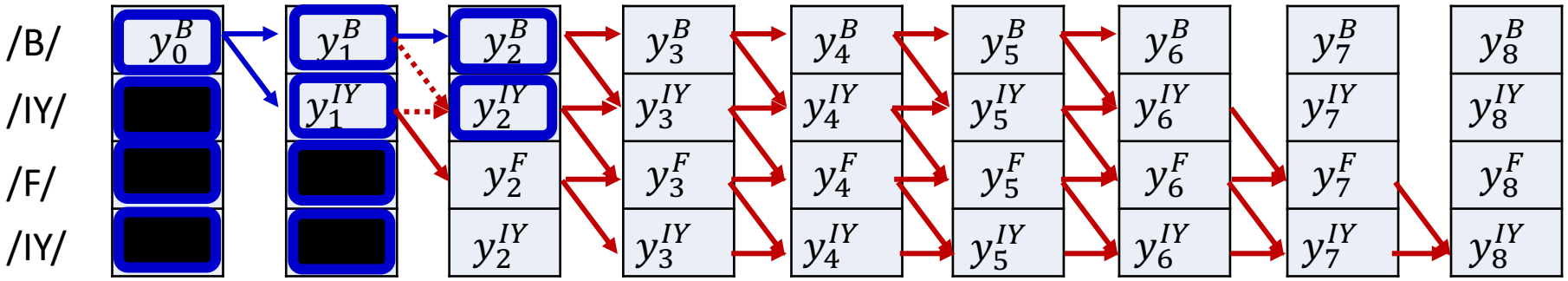
- for $t = 1 \ldots T-1$

$$BP(t,0) = 0; Bscr(t,0) = Bscr(t-1,0) \times y_t^{S(0)}$$

for $l = 1 \ldots K-1$

  - $BP(t,l) = \begin{pmatrix} l-1: & if\ \left(Bscr(t-1,l-1) > Bscr(t-1,l)\right); \\ l: & else \end{pmatrix}$
  - $Bscr(t,l) = Bscr(BP(t,l)) \times y_t^{S(l)}$

Viterbi algorithm



- Initialization:

$$BP(0,i) = null,\ i = 0 \ldots K-1$$

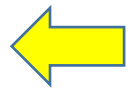
$$Bscr(0,0) = y_0^{S(0)},\ Bscr(0,i) = -\infty,\ i = 1 \ldots K-1$$

- for $t = 1 \ldots T-1$

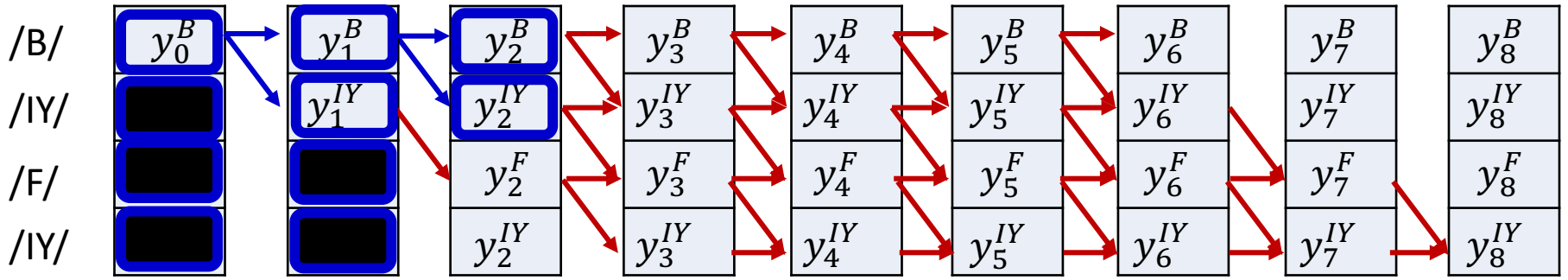
$$BP(t,0) = 0;\ Bscr(t,0) = Bscr(t-1,0) \times y_t^{S(0)}$$

for $l = 1 \ldots K-1$

- $BP(t,l) = \left(if\ \left(Bscr(t-1,l-1) > Bscr(t-1,l)\right)\ l-1;\ else\ l\right)$
- $Bscr(t,l) = Bscr(BP(t,l)) \times y_t^{S(l)}$



Viterbi algorithm



- Initialization:

$$BP(0,i) = null,\ i = 0 \ldots K-1$$

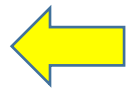
$$Bscr(0,0) = y_0^{S(0)},\ Bscr(0,i) = -\infty,\ i = 1 \ldots K-1$$

- for $t = 1 \ldots T-1$

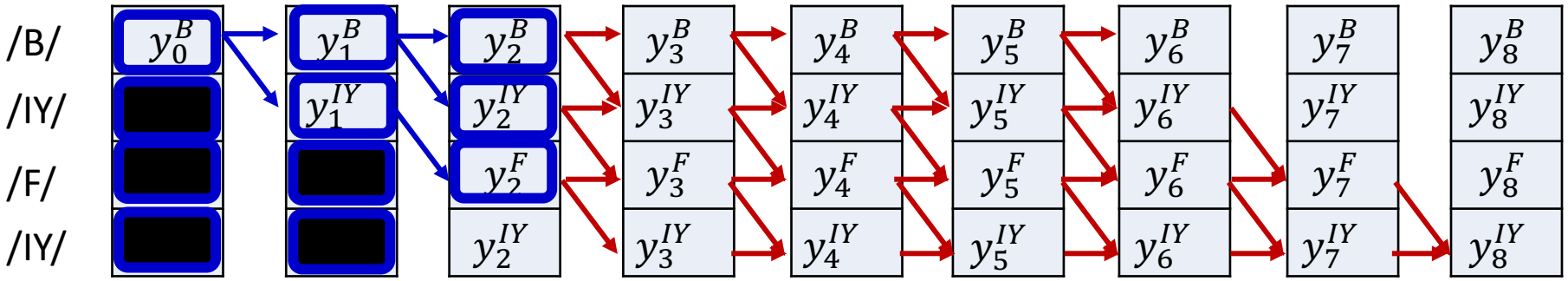
$$BP(t,0) = 0;\ Bscr(t,0) = Bscr(t-1,0) \times y_t^{S(0)}$$

for $l = 1 \ldots K-1$

- $BP(t,l) = \left(if\ \left(Bscr(t-1,l-1) > Bscr(t-1,l)\right)\ l-1;\ else\ l\right)$
- $Bscr(t,l) = Bscr(BP(t,l)) \times y_t^{S(l)}$



Viterbi algorithm



- Initialization:

$$BP(0,i) = null,\ i = 0 \dots K-1$$

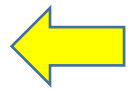
$$Bscr(0,0) = y_0^{S(0)},\ Bscr(0,i) = -\infty,\ i = 1 \dots K-1$$

- for $t = 1 \dots T-1$

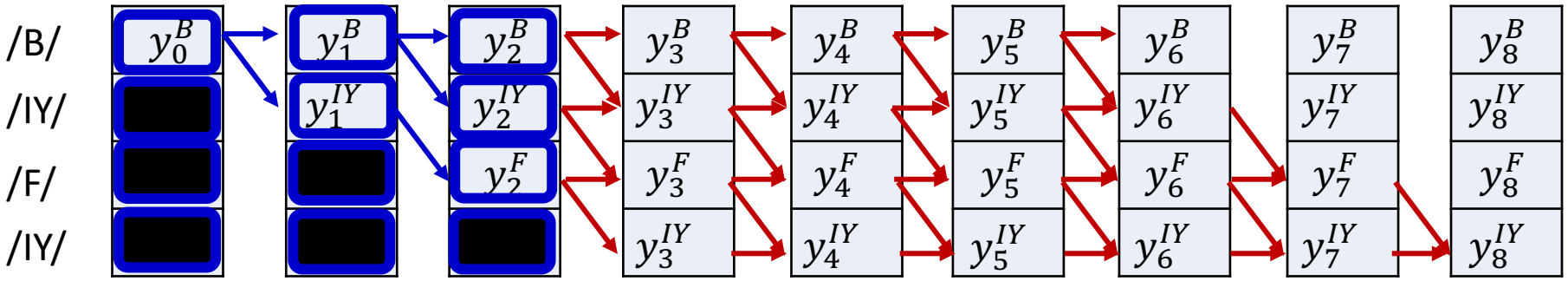
$$BP(t,0) = 0;\ Bscr(t,0) = Bscr(t-1,0) \times y_t^{S(0)}$$

for $l = 1 \dots K-1$

- $BP(t,l) = \left(if\ \left(Bscr(t-1,l-1) > Bscr(t-1,l)\right)\ l-1;\ else\ l\right)$
- $Bscr(t,l) = Bscr(BP(t,l)) \times y_t^{S(l)}$



Viterbi algorithm



- Initialization:

$$BP(0,i) = null,\ i = 0 \ldots K-1$$

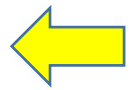
$$Bscr(0,0) = y_0^{S(0)},\ Bscr(0,i) = -\infty,\ i = 1 \ldots K-1$$

- for $t = 1 \ldots T-1$

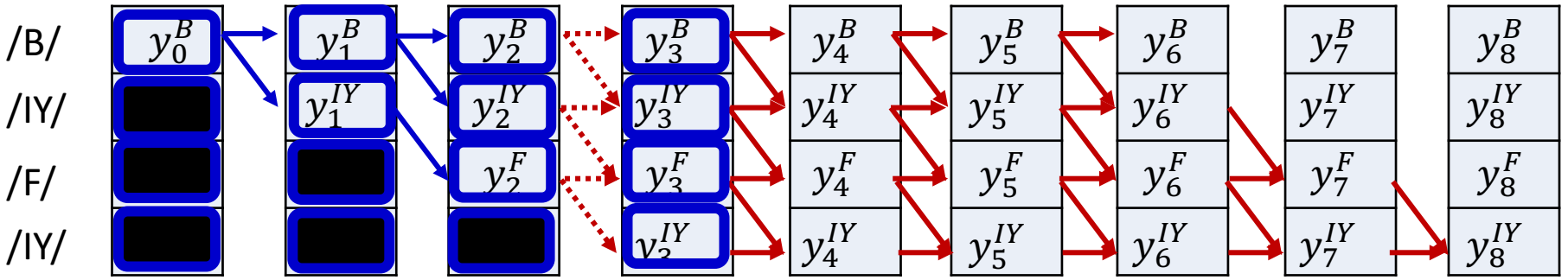
$$BP(t,0) = 0;\ Bscr(t,0) = Bscr(t-1,0) \times y_t^{S(0)}$$

for $l = 1 \ldots K-1$

- $BP(t,l) = \left(if\ \left(Bscr(t-1,l-1) > Bscr(t-1,l)\right)\ l-1;\ else\ l\right)$
- $Bscr(t,l) = Bscr(BP(t,l)) \times y_t^{S(l)}$



Viterbi algorithm



- Initialization:

$$BP(0,i) = null,\ i = 0 \ldots K-1$$

$$Bscr(0,0) = y_0^{S(0)},\ Bscr(0,i) = -\infty,\ i = 1 \ldots K-1$$

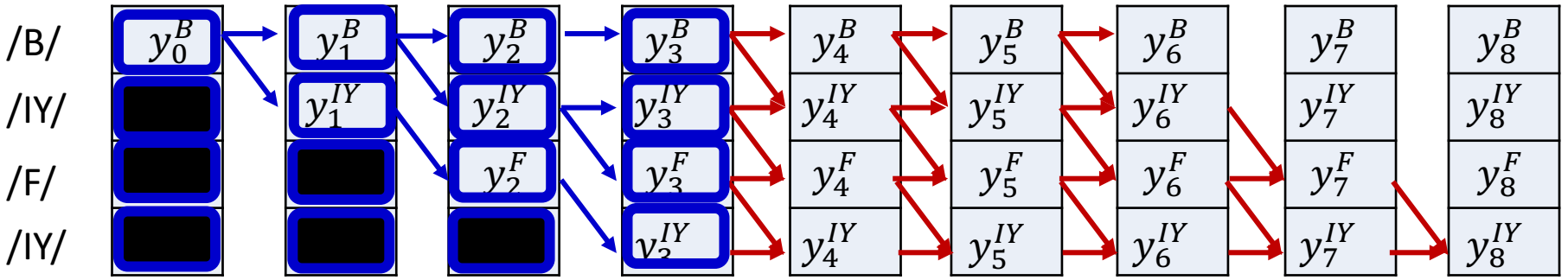
- for $t = 1 \ldots T-1$

$$BP(t,0) = 0;\ Bscr(t,0) = Bscr(t-1,0) \times y_t^{S(0)}$$

for $l = 1 \ldots K-1$

- $BP(t,l) = \begin{pmatrix} l-1: & if\ \left(Bscr(t-1,l-1) > Bscr(t-1,l)\right)\ l-1; \\ & l: else \end{pmatrix}$
- $Bscr(t,l) = Bscr(BP(t,l)) \times y_t^{S(l)}$

Viterbi algorithm



- Initialization:

$$BP(0,i) = null,\ i = 0 \ldots K-1$$

$$Bscr(0,0) = y_0^{S(0)},\ Bscr(0,i) = -\infty,\ i = 1 \ldots K-1$$

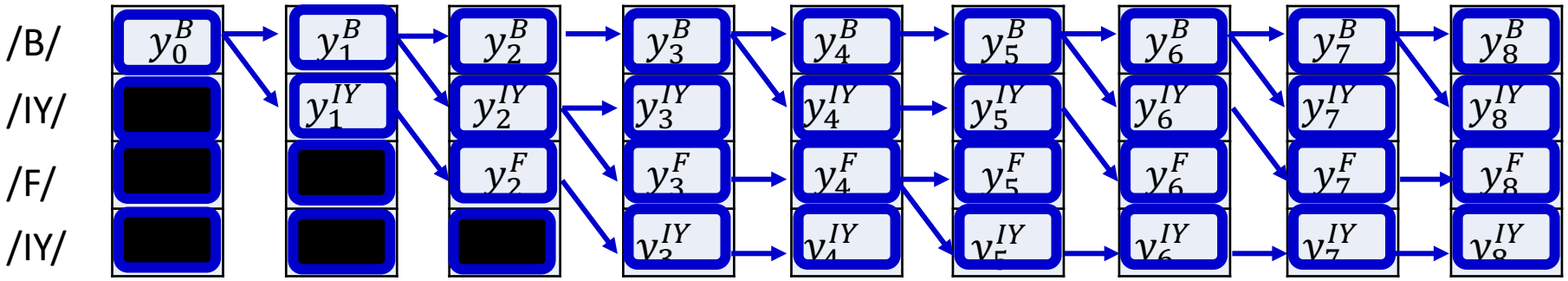
- for $t = 1 \ldots T-1$

$$BP(t,0) = 0;\ Bscr(t,0) = Bscr(t-1,0) \times y_t^{S(0)}$$

for $l = 1 \ldots K-1$

- $BP(t,l) = \begin{pmatrix} l-1: & if\ \left(Bscr(t-1,l-1) > Bscr(t-1,l)\right)\ l-1; \\ l: & else \end{pmatrix}$
- $Bscr(t,l) = Bscr(BP(t,l)) \times y_t^{S(l)}$

Viterbi algorithm



- Initialization:

$$BP(0,i) = null,\ i = 0 \ldots K-1$$

$$Bscr(0,0) = y_0^{S(0)},\ Bscr(0,i) = -\infty,\ i = 1 \ldots K-1$$

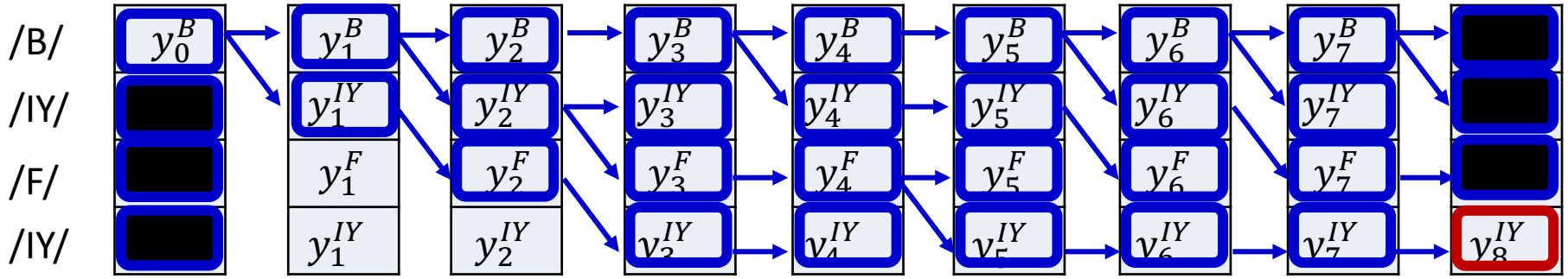
- for $t = 1 \ldots T-1$

$$BP(t,0) = 0;\ Bscr(t,0) = Bscr(t-1,0) \times y_t^{S(0)}$$

for $l = 1 \ldots K-1$

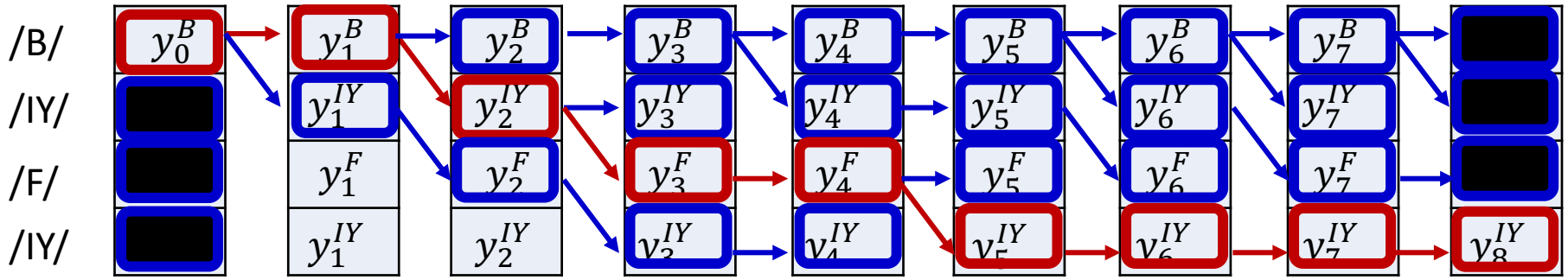
- $BP(t,l) = \begin{pmatrix} l-1: & if\ \left(Bscr(t-1,l-1) > Bscr(t-1,l)\right)\ l-1; \\ l: & else \end{pmatrix}$
- $Bscr(t,l) = Bscr(BP(t,l)) \times y_t^{S(l)}$

Viterbi algorithm



- $s(T-1) = S(K-1)$

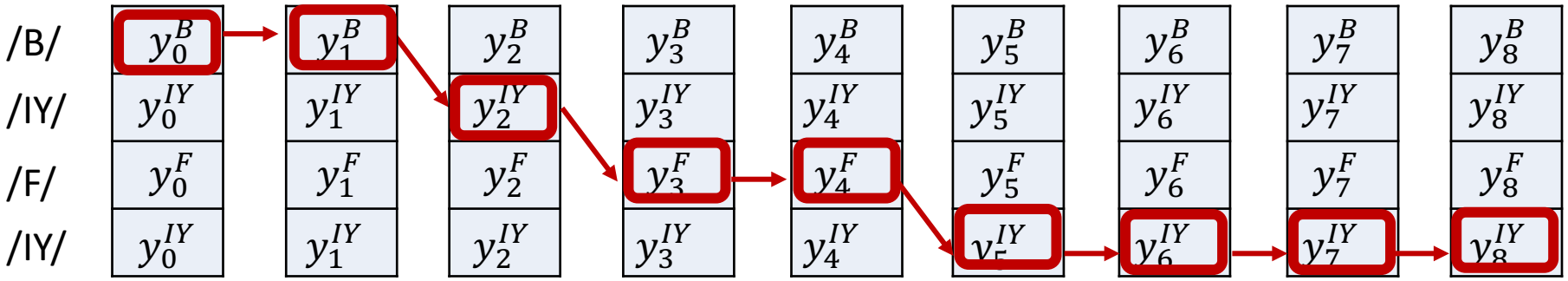
Viterbi algorithm



- $s(T-1) = S(K-1)$
- for $t = T-1\ downto\ 1$

$$s(t-1) = BP(s(t))$$

Viterbi algorithm



- $s(T-1) = S(K-1)$
- for $t = T-1 \; downto \; 1$

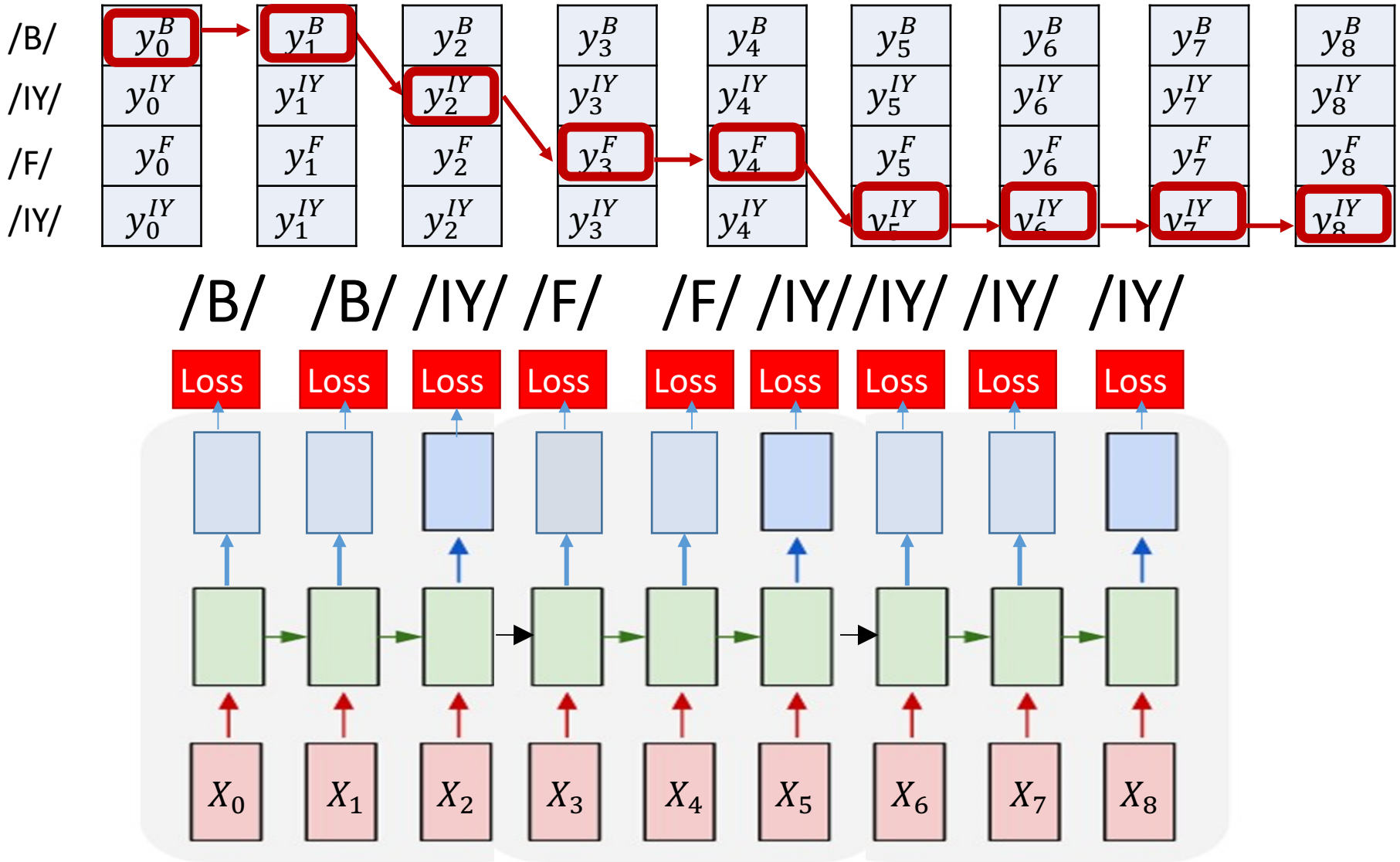
$$s(t-1) = BP(s(t))$$

/B/ /B/ /IY/ /F/ /F/ /IY/ /IY/ /IY/ /IY/

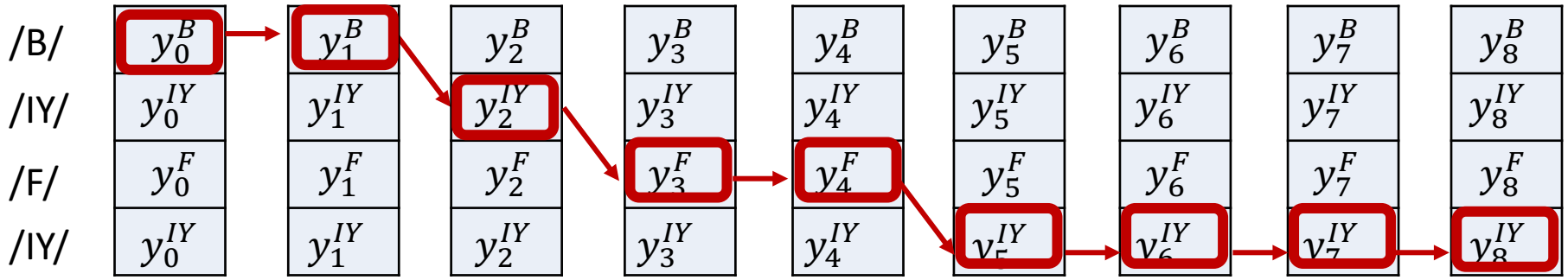
VITERBI

```
#N is the number of symbols in the target output
#S(i) is the ith symbol in target output
#T = length of input

#First create output table
For i = 1:N
    s(1:T,i) = y(1:T, S(i))

#Now run the Viterbi algorithm
# First, at t = 1
BP(1,1) = -1
Bscr(1,1) = s(1,1)
Bscr(1,2:N) = -infty
for t = 2:T
    BP(t,1) = BP(t-1,1);
    Bscr(t,1) = Bscr(t-1,1)s(t,1)
    for i = 1:min(t,N)
          BP(t,i) = Bscr(t-1,i) > Bscr(t-1,i-1) ? i : i-1
          Bscr(t,i) = Bscr(t-1,BP(t,i))s(t,i)

# Backtrace
AlignedSymbol(T) = N
for t = T downto 1
    AlignedSymbol(t-1) = BP(t,AlignedSymbol(T))
```

Assumed targets for training with the Viterbi algorithm



Gradients from the alignment



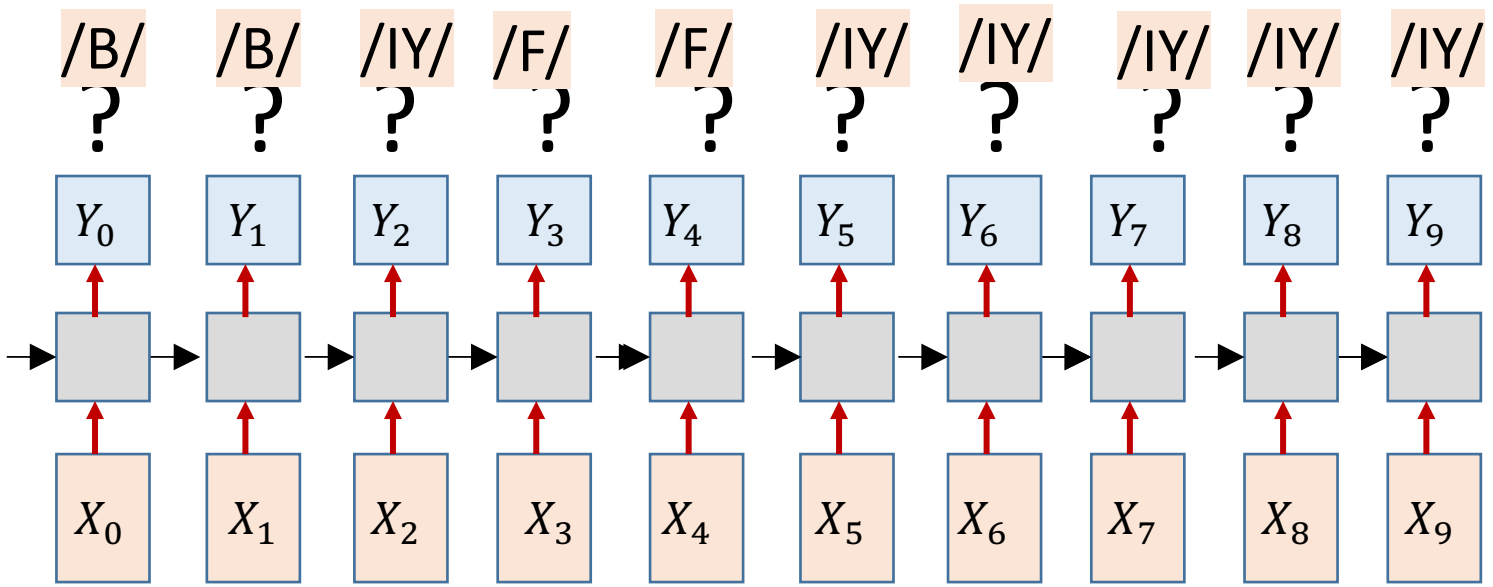
$$Loss = \sum_t Xent\left(Y_t, symbol_t^{bestpath}\right) = -\sum_t \log Y\left(t, symbol_t^{bestpath}\right)$$

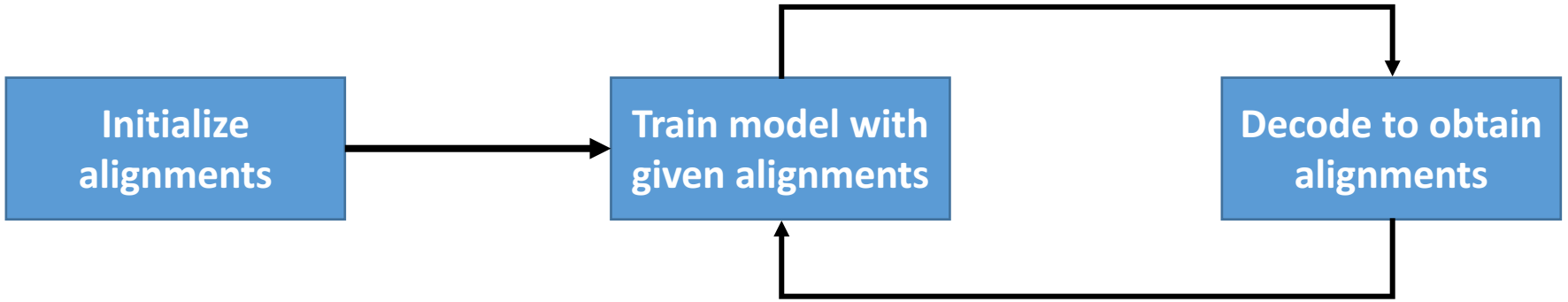
- The gradient w.r.t the t-th output vector Y_t

$$\nabla_{Y_t} Loss = \left[0 \quad 0 \quad \dots \quad \frac{-1}{Y\left(t, symbol_t^{bestpath}\right)} \quad 0 \quad \dots \quad 0\right]$$

  - Zeros except at the component corresponding to the target *in the estimated alignment*

Iterative Estimate and Training



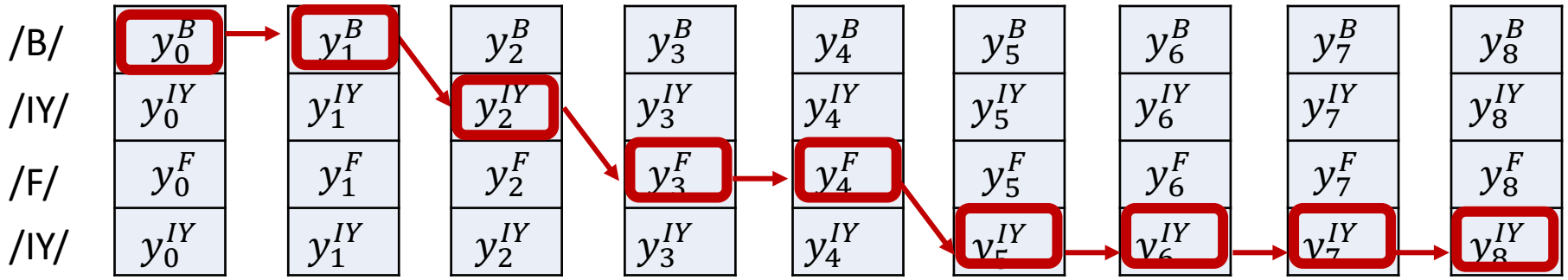


The "decode" and "train" steps may be combine into a single "decode, find alignment, compute derivatives" step for SGD and mini-batch updates

Iterative update: Problem

- Approach heavily dependent on initial alignment
- Prone to poor local optima
- Alternate solution: Do not commit to an alignment during any pass..

The reason for suboptimality

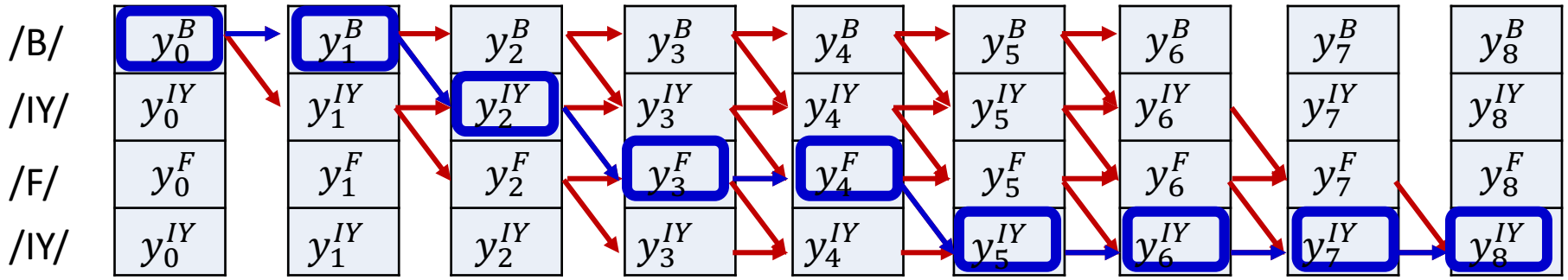


- We *commit* to the single "best" estimated alignment
  - The *most likely* alignment

$$DIV = -\sum_{t} \log Y\left(t, symbol_t^{bestpath}\right)$$

  - This can be way off, particularly in early iterations, or if the model is poorly initialized

The reason for suboptimality



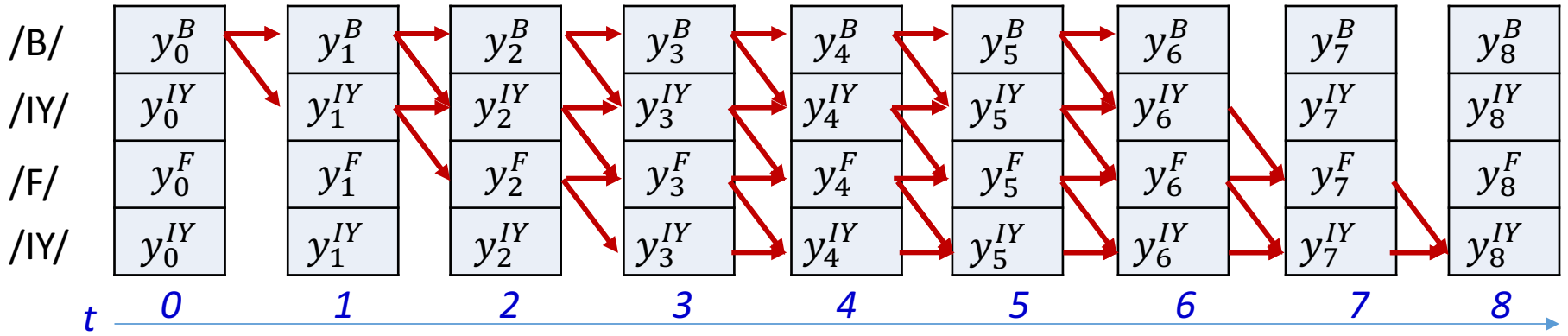
- We *commit* to the single "best" estimated alignment
  - The *most likely* alignment

$$Loss = -\sum_t \log Y\left(t, symbol_t^{bestpath}\right)$$

  - This can be way off, particularly in early iterations, or if the model is poorly initialized

- **Alternate view:** there is a probability distribution over alignments of the target Symbol sequence (to the input)
  - *Selecting a single alignment is the same as drawing a single sample from it*
  - Selecting the most likely alignment is the same as deterministically always drawing the most probable value from the distribution

Averaging over *all* alignments

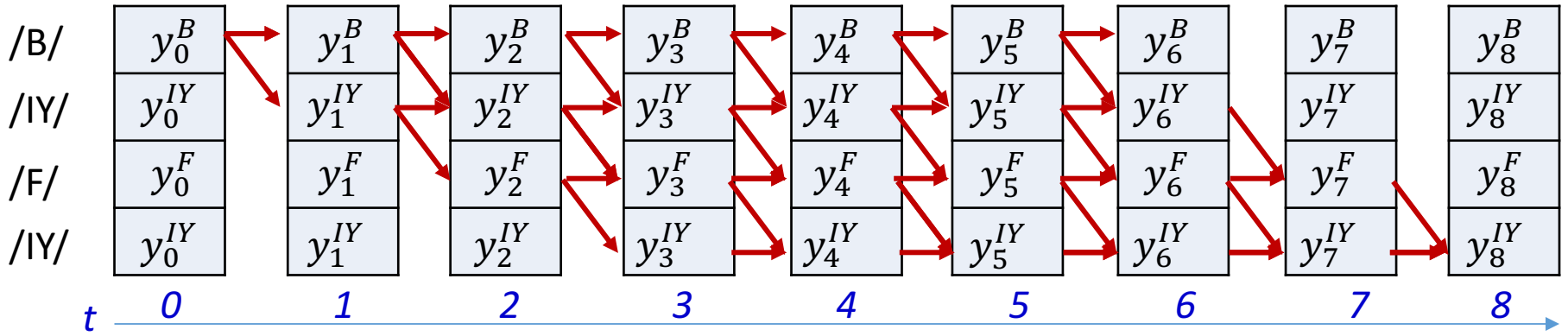


- Instead of only selecting the most likely alignment, use the statistical expectation over *all* possible alignments

$$Loss = E\left[-\sum_t \log Y(t, s_t)\right]$$

  - Use the *entire distribution of alignments*
  - This will mitigate the issue of suboptimal selection of alignment

The expectation over *all* alignments



$$Loss = E\left[-\sum_t \log Y(t, s_t)\right]$$

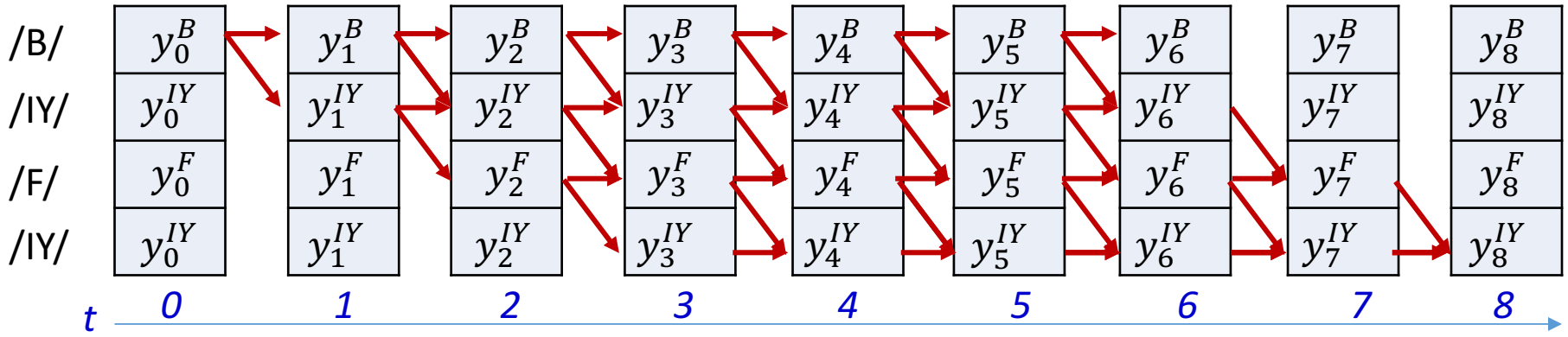
- Using the linearity of expectation

$$Loss = -\sum_t E[\log Y(t, s_t)]$$

  - This reduces to finding the expected Loss *at each input*

$$Loss = -\sum_t \sum_{S \in S_1 \ldots S_K} P(s_t = S|\mathbf{S}, \mathbf{X}) \log Y(t, s_t = S)$$

The expectation over *all* alignments



The probability of seeing the specific symbol s at time t, *given* that the symbol sequence is an expansion of $\mathbf{S} = S_0 \ldots S_{K-1}$ *and given* the input sequence $\mathbf{X} = X_0 \ldots X_{N-1}$
We need to be able to compute this

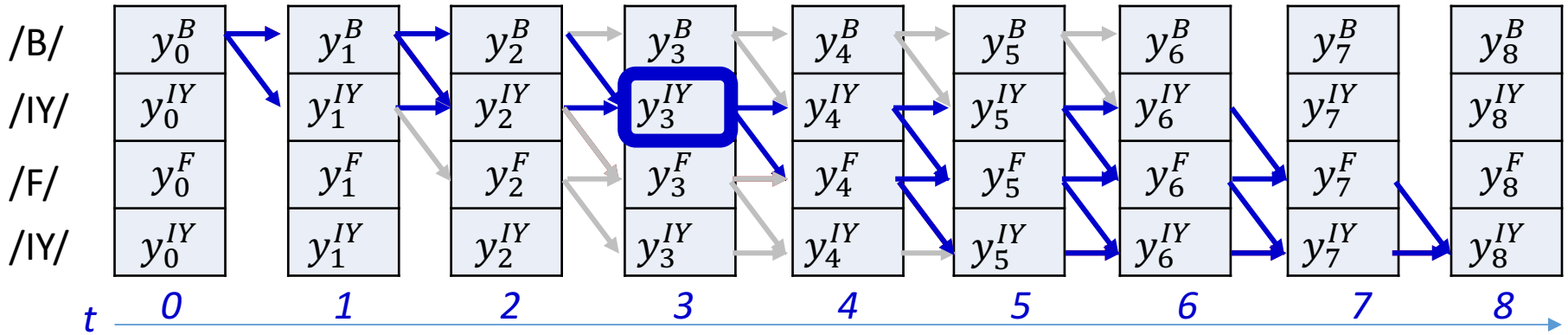
- Using th

$$Loss = -\sum_t E[\log Y(t, s_t)]$$

  - This reduces to finding the expected Loss *at each input*

$$Loss = -\sum_t \sum_{S \in S_1 \ldots S_K} P(s_t = S|\mathbf{S}, \mathbf{X}) \log Y(t, s_t = S)$$

A posteriori probabilities of symbols

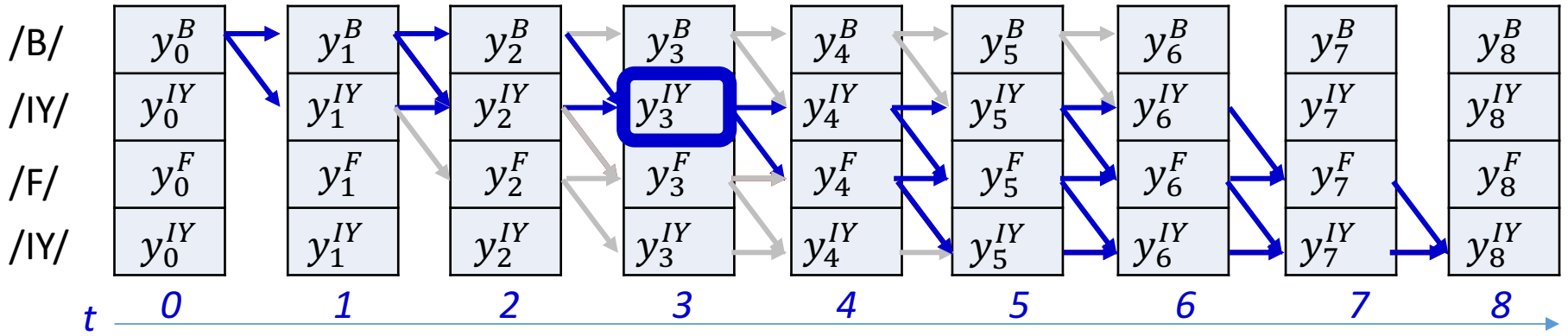


$$P(s_t = S_r|\mathbf{S},\mathbf{X}) \propto P(s_t = S_r,\mathbf{S}|\mathbf{X})$$

- $P(s_t = S_r,\mathbf{S}|\mathbf{X})$ is the total probability of all valid paths *in the graph for target sequence* $\mathbf{S}$ that go through the symbol S_r (the r^{th} symbol in the sequence $S_1 \ldots S_K$) at time t

- We will compute this using the "forward-backward" algorithm

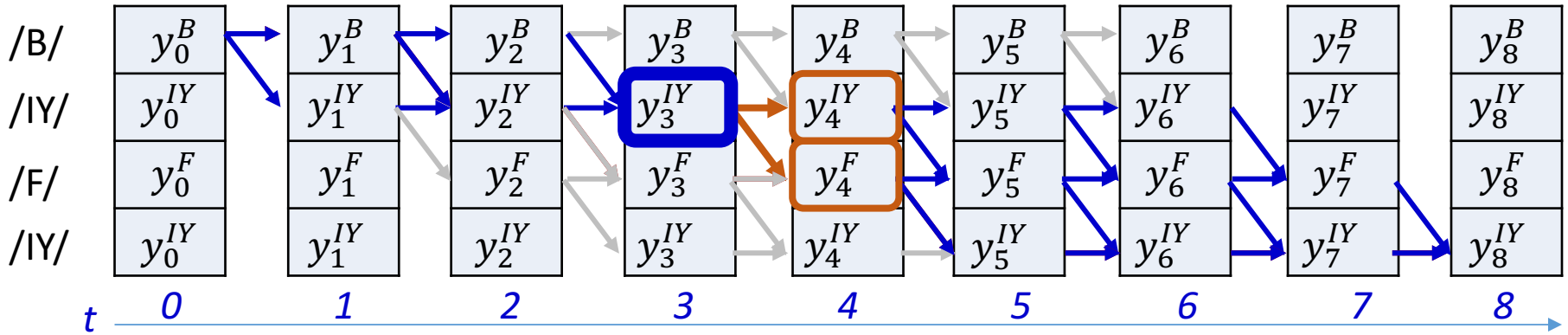
A posteriori probabilities of symbols



- $P(s_t = S_r, \mathbf{S}|\mathbf{X})$ can be decomposed as

$$P(s_t = S_r, \mathbf{S}|\mathbf{X}) = P(S_1, \dots, S_K, s_t = S_r|\mathbf{X})$$

A posteriori probabilities of symbols

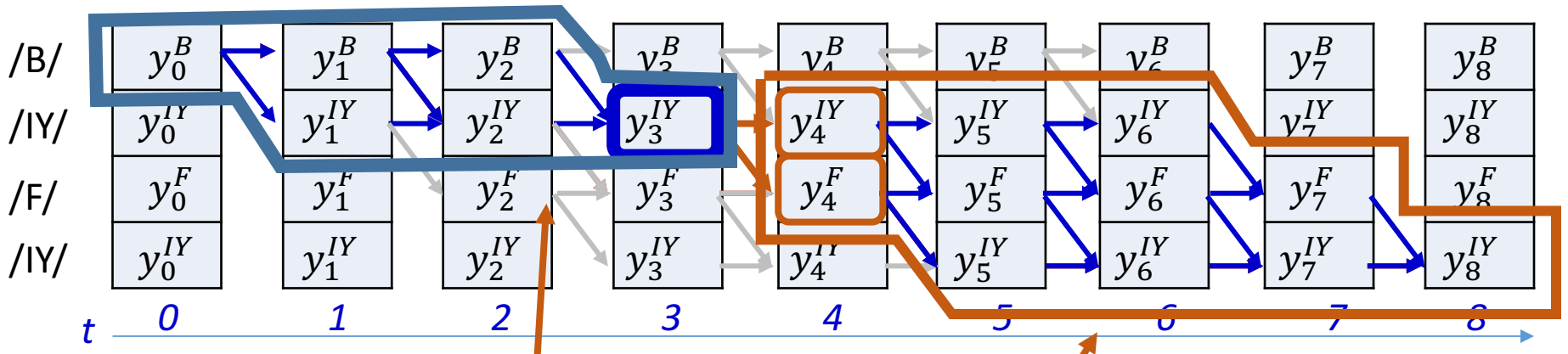


- $P(s_t = S_r, \mathbf{S}|\mathbf{X})$ can be decomposed as

$$P(s_t = S_r, \mathbf{S}|\mathbf{X}) = P(S_1, \dots, S_r, \dots, S_K, s_t = S_r|\mathbf{X})$$
$$= P(S_1 \dots S_r, s_t = S_r, s_{t+1} \in succ(S_r), \dots, S_K \,|\mathbf{X})$$

- Where $succ(S_r)$ is a symbol that can follow S_r in a sequence
  - Here it is either S_r or S_{r+1} (red blocks in figure)
  - The equation literally says that after the blue block, either of the two red arrows may be followed

A posteriori probabilities of symbols

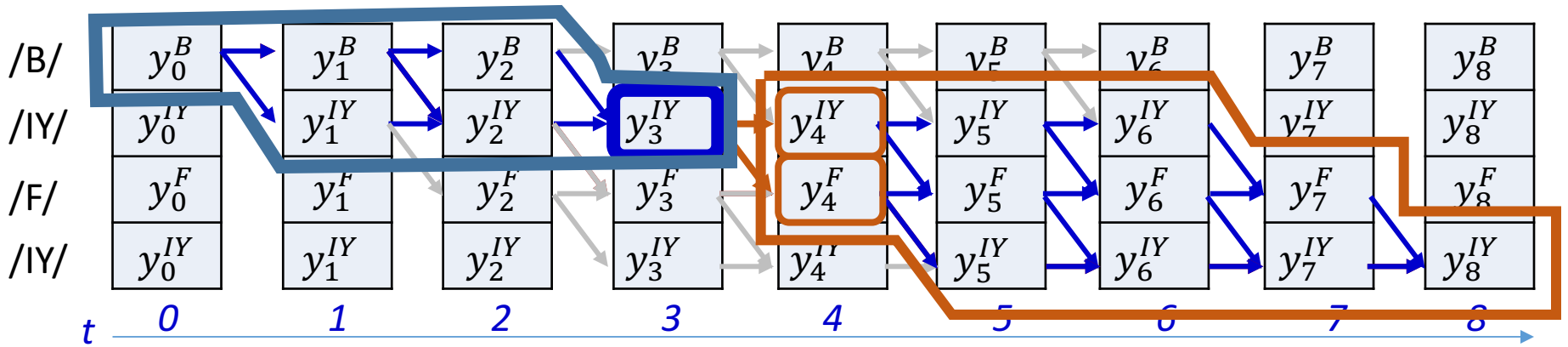


- $P(s_t = S_r, \mathbf{S}|\mathbf{X})$ can be decomposed as

$$P(s_t = S_r, \mathbf{S}|\mathbf{X}) = P(S_1, \dots, S_r, \dots, S_K, s_t = S_r|\mathbf{X})$$
$$= P(S_1 \dots S_r, s_t = S_r, s_{t+1} \in succ(S_r), \dots, S_K|\mathbf{X})$$

- Where $succ(S_r)$ is a symbol that can follow S_r in a sequence
  - Here it is either S_r or S_{r+1} (red blocks in figure)
  - The equation literally says that after the blue block, either of the two red arrows may be followed

A posteriori probabilities of symbols



- $P(s_t = S_r, \mathbf{S}|\mathbf{X})$ can be decomposed as

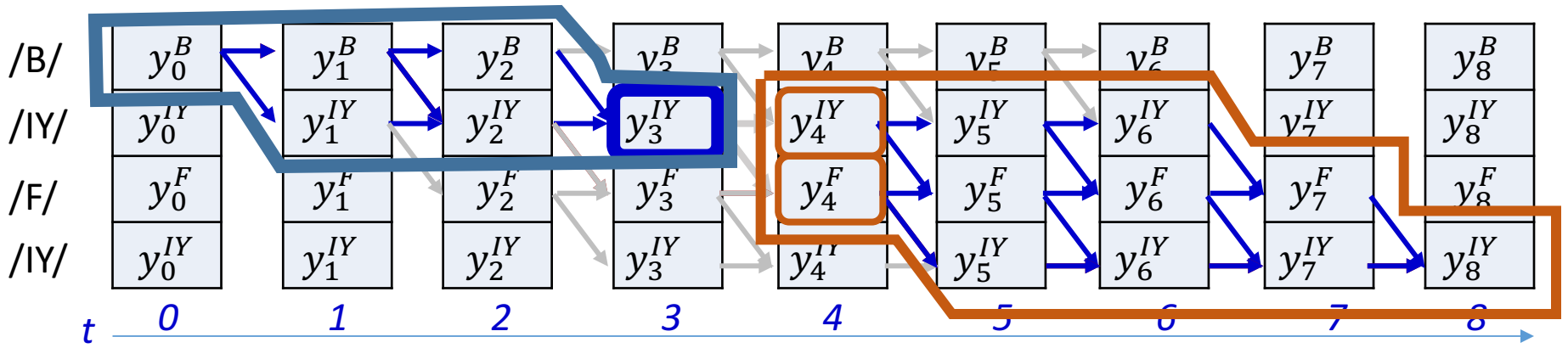
$$P(s_t = S_r, \mathbf{S}|\mathbf{X}) = P(S_1, \dots, S_r, \dots, S_K, s_t = S_r|\mathbf{X})$$
$$= P(S_1 \dots S_r, s_t = S_r, s_{t+1} \in succ(S_r), succ(S_r), \dots, S_K \ |\mathbf{X})$$

- Using Bayes Rule

$$= P(S_1 \dots S_r, s_t = S_r \ |\mathbf{X}) P(s_{t+1} \in succ(S_r), succ(S_r), \dots, S_K \ |S_1 \dots S_r, s_t = S_r \mathbf{X})$$

- The probability of the subgraph in the blue outline, times the conditional probability of the red-encircled subgraph, given the blue subgraph

A posteriori probabilities of symbols



- $P(s_t = S_r, \mathbf{S}|\mathbf{X})$ can be decomposed as

$$P(s_t = S_r, \mathbf{S}|\mathbf{X}) = P(S_1, \dots, S_r, \dots, S_K, s_t = S_r|\mathbf{X})$$
$$= P(S_1 \dots S_r, s_t = S_r, s_{t+1} \in succ(S_r), succ(S_r), \dots, S_K \,|\mathbf{X})$$

- Using Bayes Rule

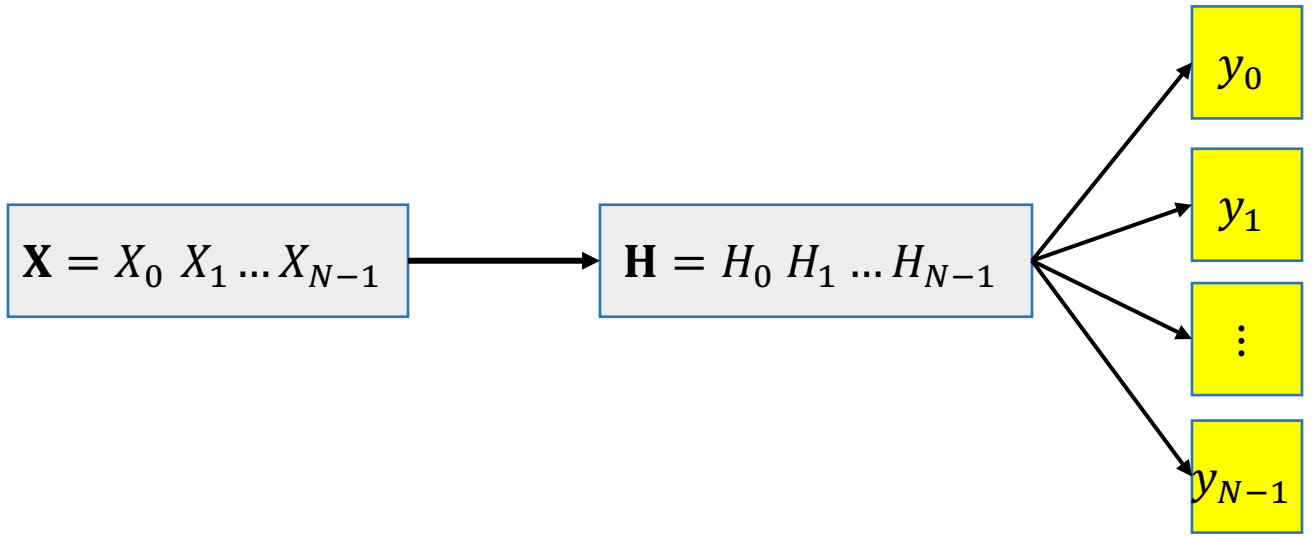
$$= P(S_1 \dots S_r, s_t = S_r \,|\mathbf{X}) P(s_{t+1} \in succ(S_r), succ(S_r), \dots, S_K \,|S_1 \dots S_r, s_t = S_r \mathbf{X})$$

- For a recurrent network without feedback from the output we can make the conditional independence assumption:

$$P(s_t = S_r, \mathbf{S}|\mathbf{X}) = P(S_1 \dots S_r, s_t = S_r \,|\mathbf{X}) P(s_{t+1} \in succ(S_r), succ(S_r), \dots, S_K \,|\mathbf{X})$$

Assuming past output symbols do not directly feed back into the net

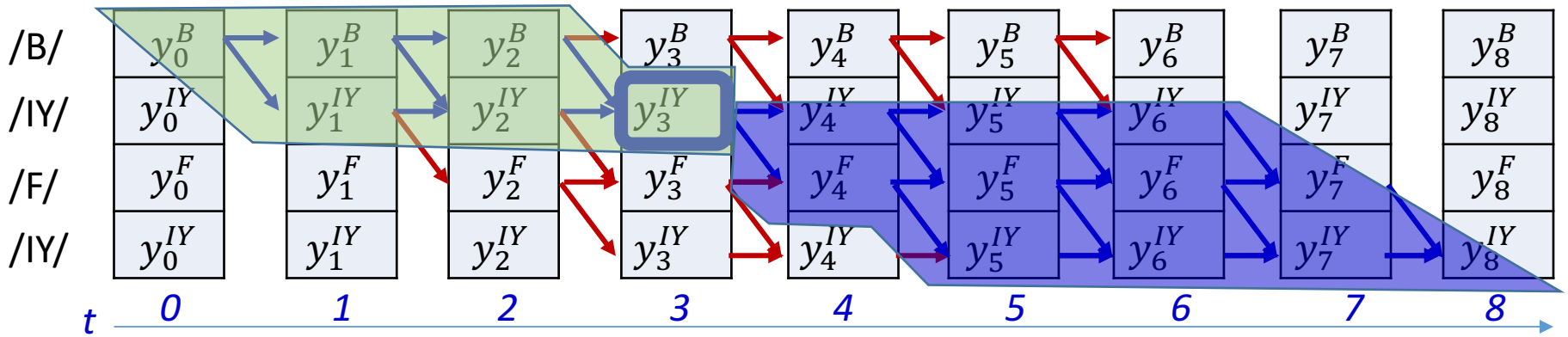
Conditional independence



- ***Dependency graph:*** Input sequence $\mathbf{X} = X_0\ X_1 \ldots X_{N-1}$ governs hidden variables $\mathbf{H} = H_0\ H_1 \ldots H_{N-1}$
- Hidden variables govern output predictions $y_0, y_1, \ldots y_{N-1}$ individually
- $y_0, y_1, \ldots y_{N-1}$ are conditionally independent given $\mathbf{H}$

- Since $\mathbf{H}$ is deterministically derived from $\mathbf{X}$, $y_0, y_1, \ldots y_{N-1}$ are also conditionally independent given $\mathbf{X}$
  - This wouldn't be true if the relation between $\mathbf{X}$ and $\mathbf{H}$ were not deterministic or if $\mathbf{X}$ is unknown, or if there were direct connections between the ys

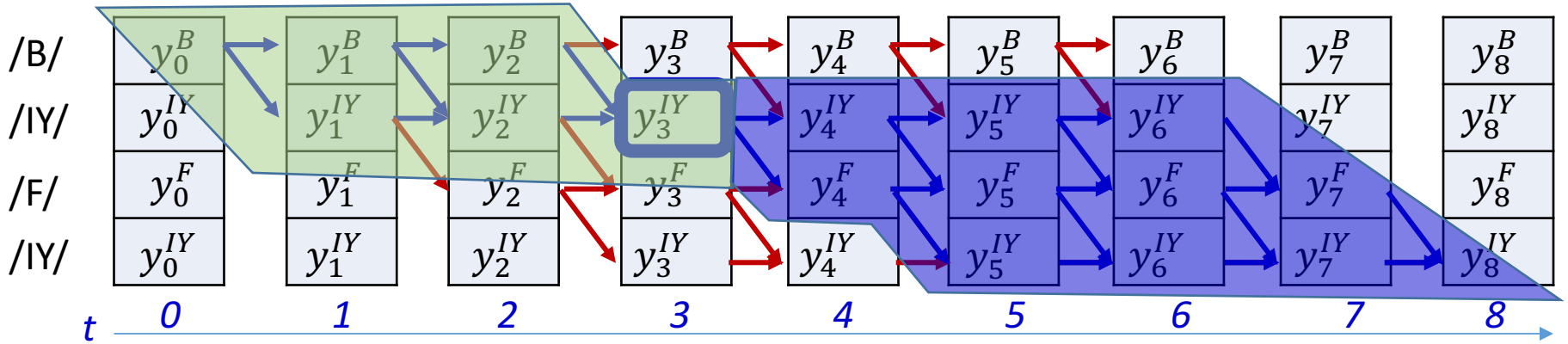
A posteriori symbol probability



$$P(s_t = S_r, \mathbf{S}|\mathbf{X})$$
$$= P(S_1 \dots S_r, s_t = S_r \,|\mathbf{X}) P(s_{t+1} \in succ(S_r), succ(S_r), \dots, S_K \,|\mathbf{X})$$

- We will call the first term the *forward probability* $\alpha(t, r)$
- We will call the second term the *backward* probability $\beta(t, r)$

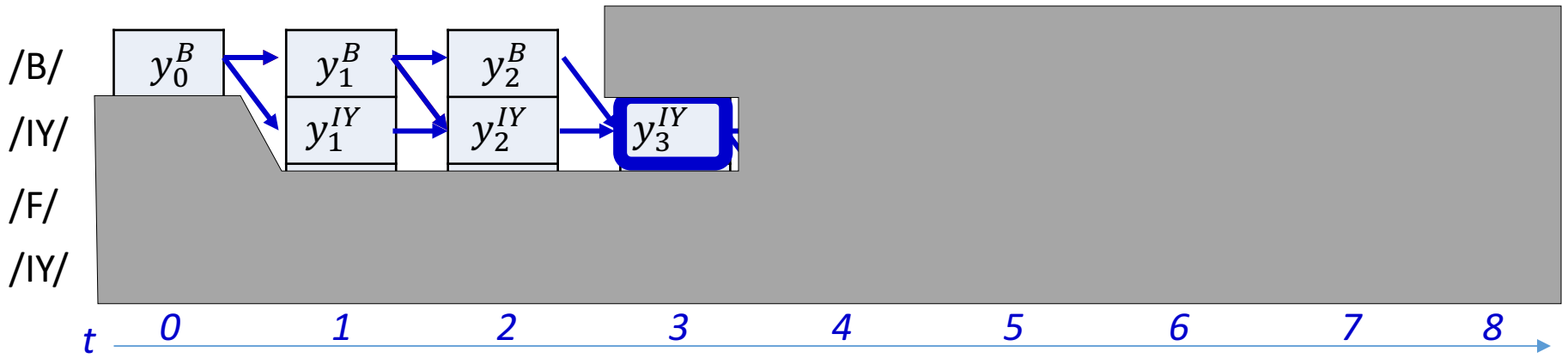
A posteriori symbol probability



$$P(s_t = S_r, \mathbf{S}|\mathbf{X})$$
$$= P(S_1 \dots S_r, s_t = S_r \,|\mathbf{X}) P(s_{t+1} \in succ(S_r), succ(S_r), \dots, S_K \,|\mathbf{X})$$

- We will call the first term the *forward probability* $\alpha(t, r)$
- We will call the second term the *backward* probability $\beta(t, r)$

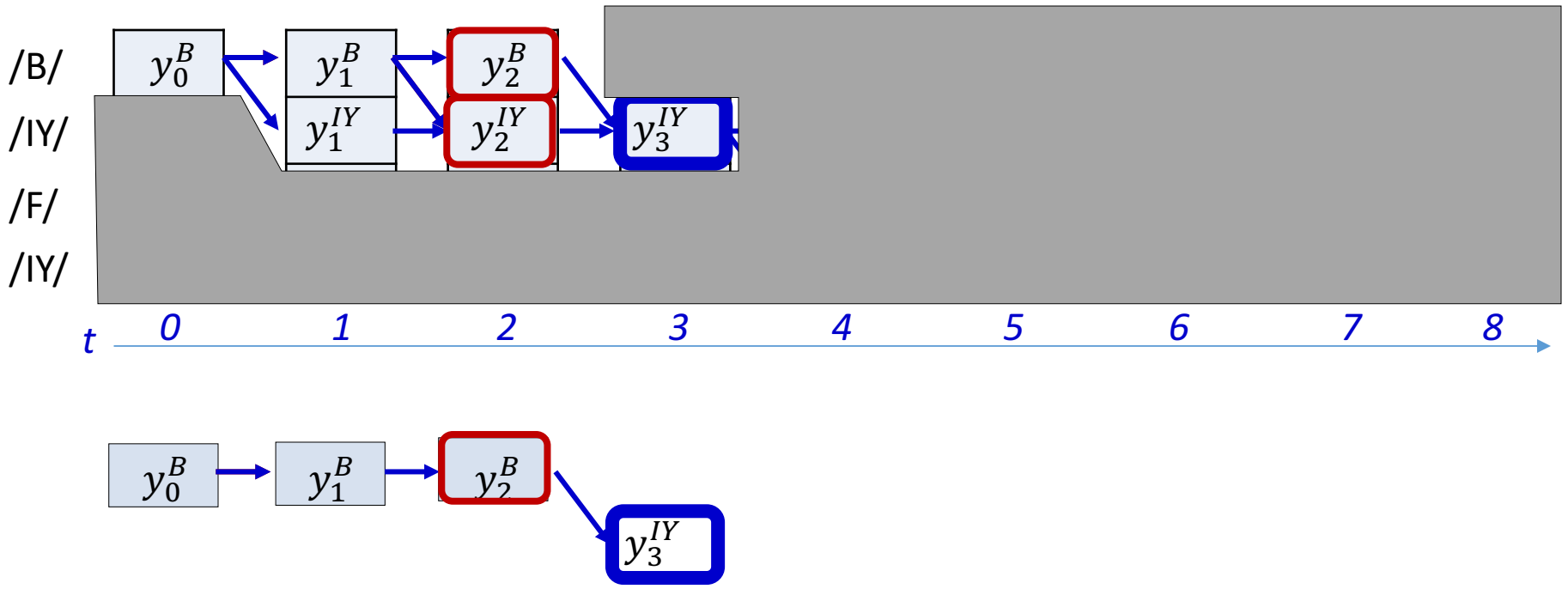
Computing $\alpha(t,r)$: Forward algorithm



$$\alpha(t,r) = P(S_1..S_r, s_t = S_r|\mathbf{X})$$

- The $\alpha(t,r)$ is the total probability of the subgraph shown

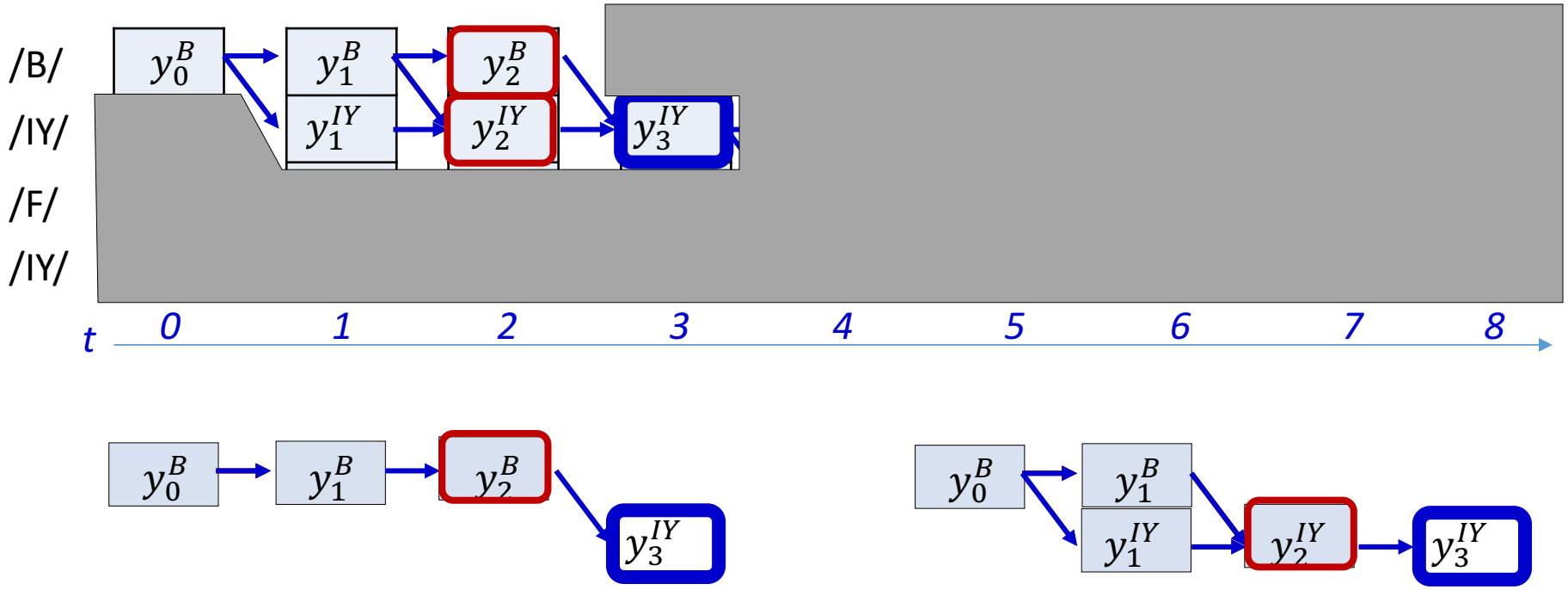
Computing $\alpha(t, r)$: Forward algorithm



$$\alpha(3, IY) = P(S_1..S_r, s_t = S_r|\mathbf{X})$$

$$\alpha(3, IY)$$
$$= P(subgraph\ ending\ at(2,B))y_3^{IY} + P(subgraph\ ending\ at(2,IY))y_3^{IY}$$

Computing $\alpha(t,r)$: Forward algorithm

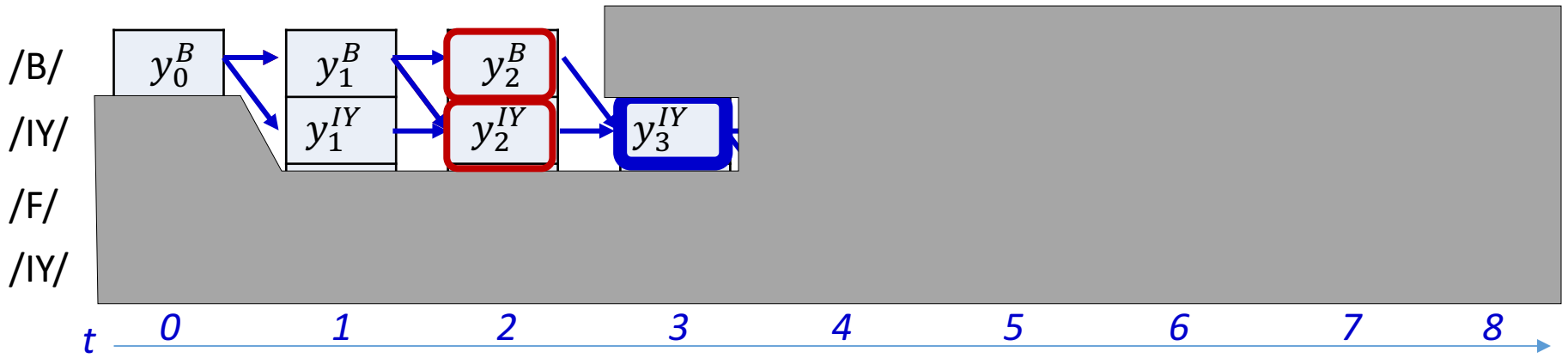


$$\alpha(t,r) = P(S_1..S_r, s_t = S_r|\mathbf{X})$$

$$\alpha(3,IY) = P(subgraph\ ending\ at\ (2,\mathrm{B})\ )y_3^{IY} + P(subgraph\ ending\ at\ (2,\mathrm{IY}))y_3^{IY}$$

$$\alpha(3,IY) = \alpha(2,B)y_3^{IY} + \alpha(2,IY)y_3^{IY}$$

Computing $\alpha(t,r)$: Forward algorithm



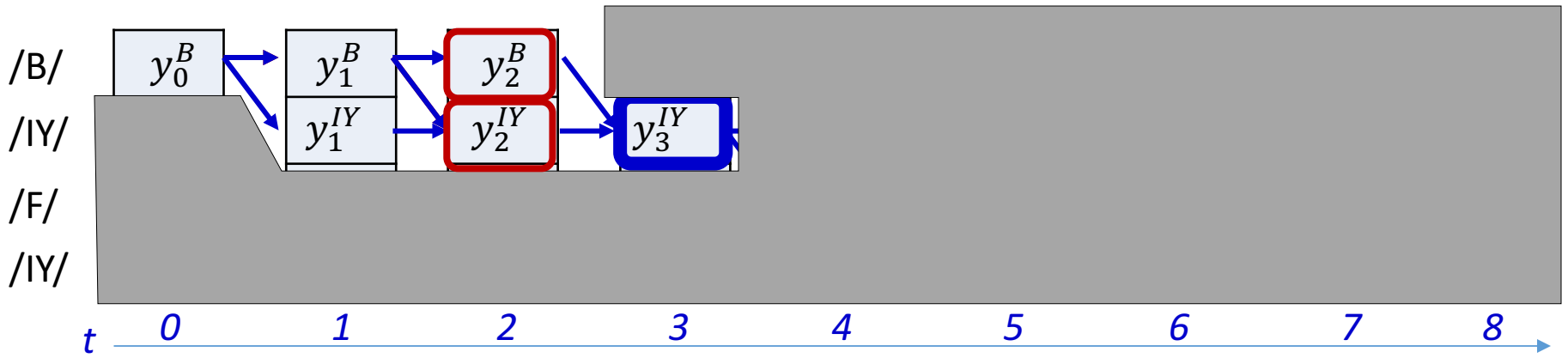
$$\alpha(t,r) = P(S_1..S_r, s_t = S_r|\mathbf{X})$$

- The $\alpha(t,r)$ is the total probability of the subgraph shown
- We can marginalize the symbol at time t-1

$$\alpha(t,r) = \sum_{q:S_q \in pred(S_r)} P(S_1\ldots,S_q\,, s_{t-1} = S_q, s_t = S_r|\mathbf{X})$$

  - Where $pred(S_r)$ is any symbol that is permitted to come before an S_r and may include S_r
  - q is its row index, and can take values r and $r-1$ in this example

Computing $\alpha(t,r)$: Forward algorithm



$$\alpha(t,r) = P(S_1..S_r, s_t = S_r|\mathbf{X})$$

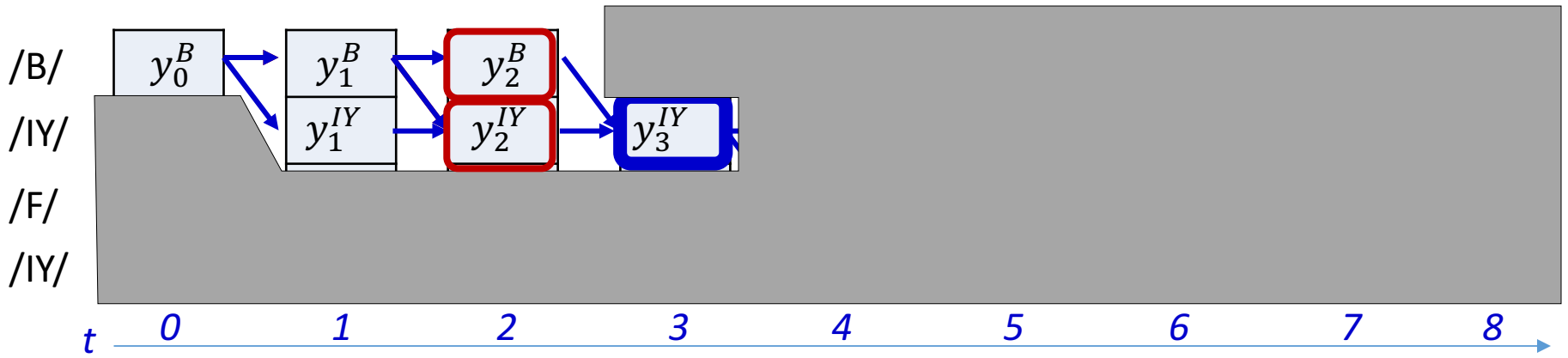
- The $\alpha(t,r)$ is the total probability of the subgraph shown
- We can marginalize out the symbol at time t-1

$$\alpha(t,r) = \sum_{q:S_q \in pred(S_r)} P(S_1\ldots,S_q\,, s_{t-1} = S_q, s_t = S_r|\mathbf{X})$$

- Using the conditional independence assumed

$$\alpha(t,r) = \sum_{q:S_q \in pred(S_r)} P(S_1\ldots,S_q\,, s_{t-1} = S_q|\mathbf{X})P(s_t = S_r|\mathbf{X})$$

Computing $\alpha(t,r)$: Forward algorithm



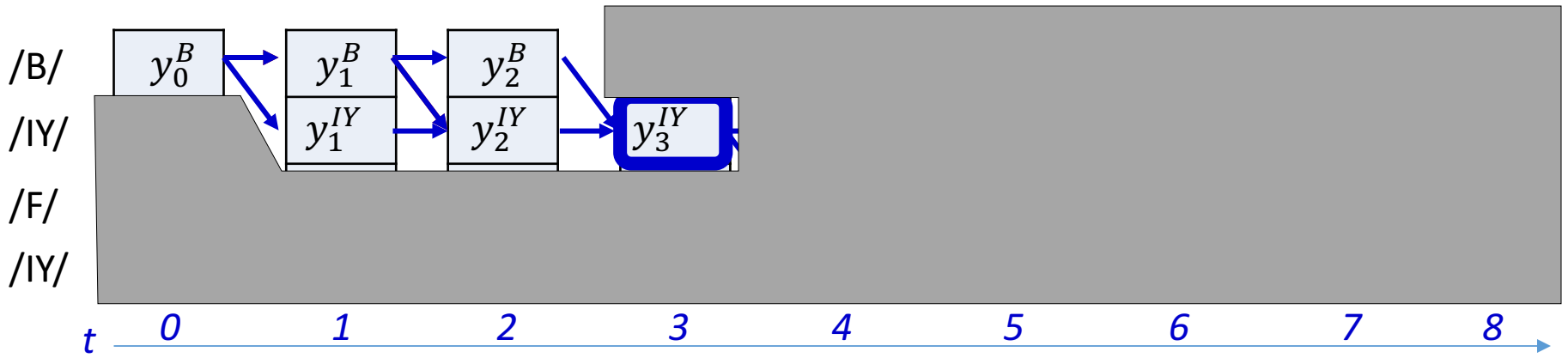
$$\alpha(t,r) = P(S_1..S_r, s_t = S_r|\mathbf{X})$$

- The $\alpha(t,r)$ is the total probability of the subgraph shown

$$\alpha(t,r) = \sum_{q:S_q \in pred(S_r)} P(S_1\ldots,S_q\,, s_{t-1} = S_q|\mathbf{X})P(s_t = S_r|\mathbf{X})$$

$$= \sum_{q:S_q \in pred(S_r)} \alpha(t-1,q)y_t^{S_r}$$

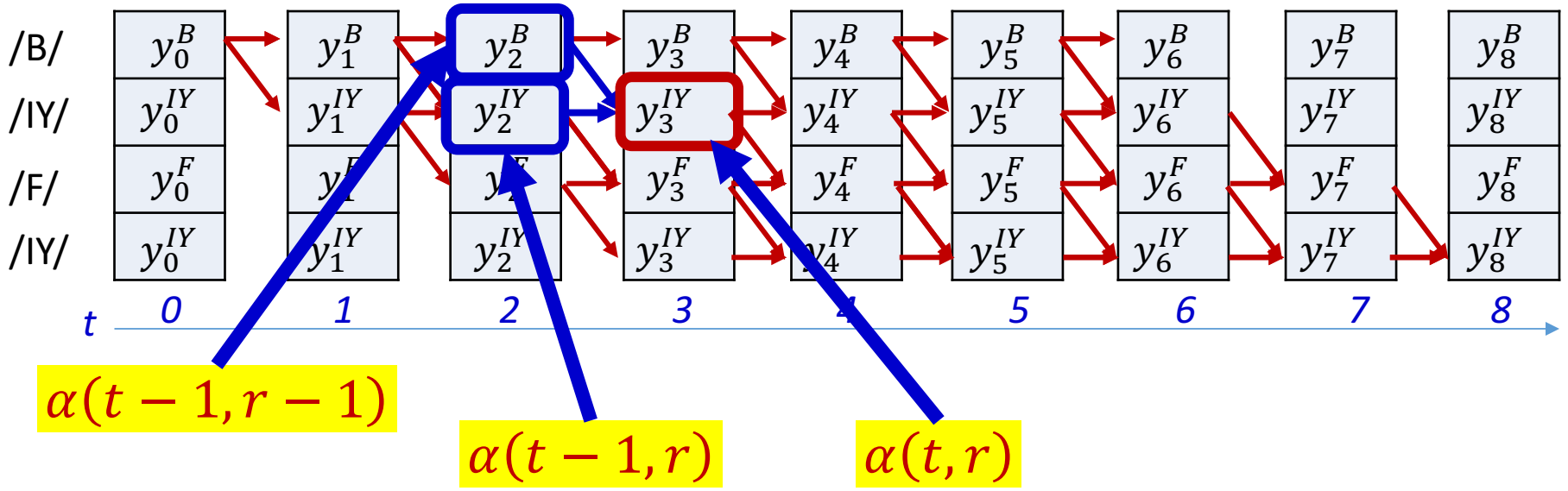
Forward algorithm



$$\alpha(t,r) = \sum_{q:S_q \in pred(S_r)} \alpha(t-1,q) y_t^{S_r}$$

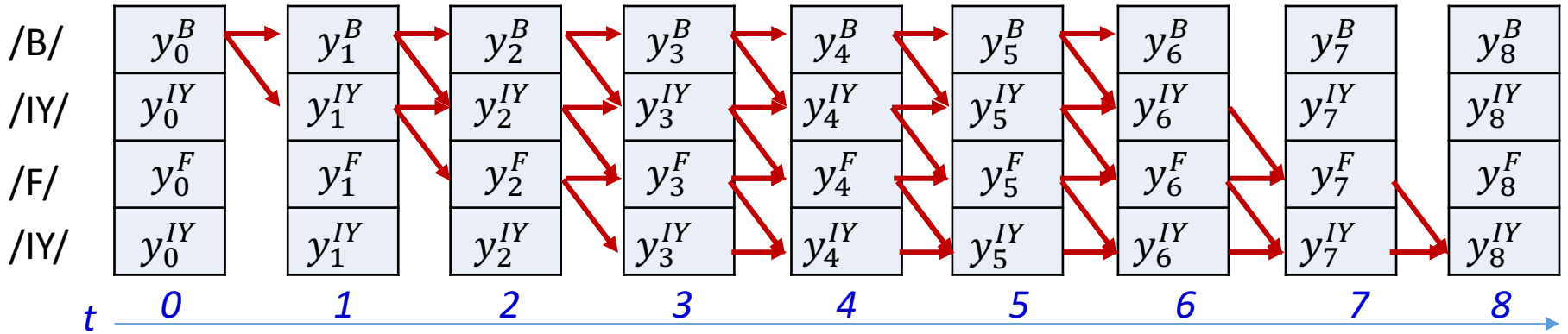
- The $\alpha(t,r)$ is the total probability of the subgraph shown

Forward algorithm



$$\alpha(t,r) = \left(\alpha(t-1,r) + \alpha(t-1,r-1)\right) y_t^{S(r)}$$

Forward algorithm



- Initialization:

$$\alpha(0,1) = y_0^{S(1)}, \quad \alpha(0,r) = 0, \ r > 1$$

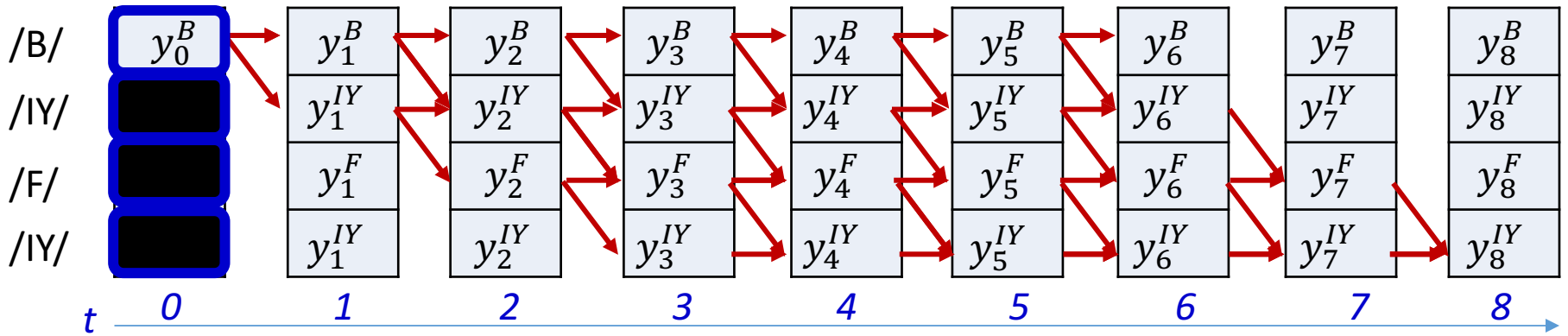
- for $t = 1 \ldots T-1$

$$\alpha(t,1) = \alpha(t-1,1) y_t^{S(1)}$$

for $l = 2 \ldots K$

$$\alpha(t,l) = (\alpha(t-1,l) + \alpha(t-1,l-1)) y_t^{S(l)}$$

Forward algorithm



- Initialization:

$$\alpha(0,1) = y_0^{S(1)}, \quad \alpha(0,r) = 0, \ r > 1$$

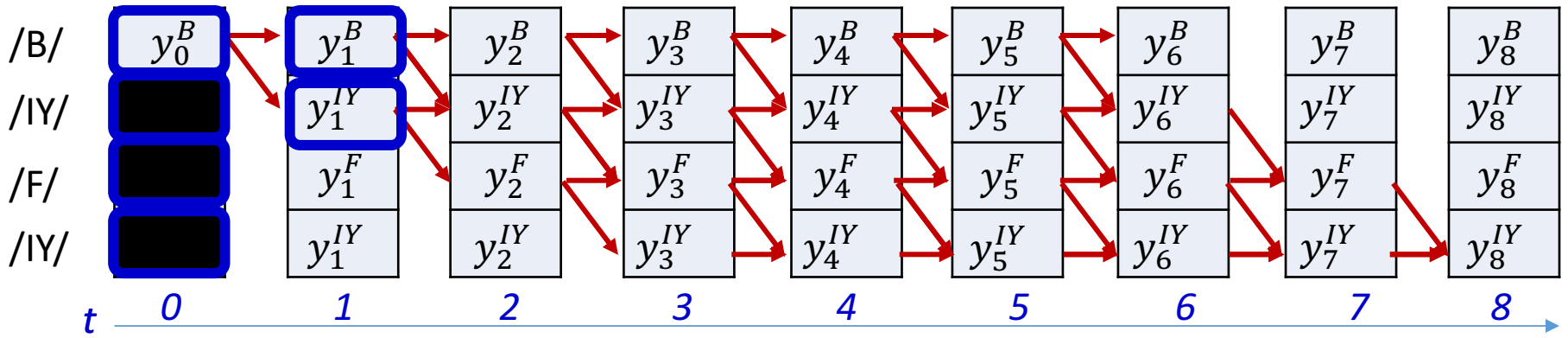
- for $t = 1 \ldots T-1$

$$\alpha(t,1) = \alpha(t-1,1) y_t^{S(1)}$$

for $l = 2 \ldots K$

- $\alpha(t,l) = (\alpha(t-1,l) + \alpha(t-1,l-1)) y_t^{S(l)}$

Forward algorithm



- Initialization:

$$\alpha(0,1) = y_0^{S(1)}, \quad \alpha(0,r) = 0, \; r > 1$$

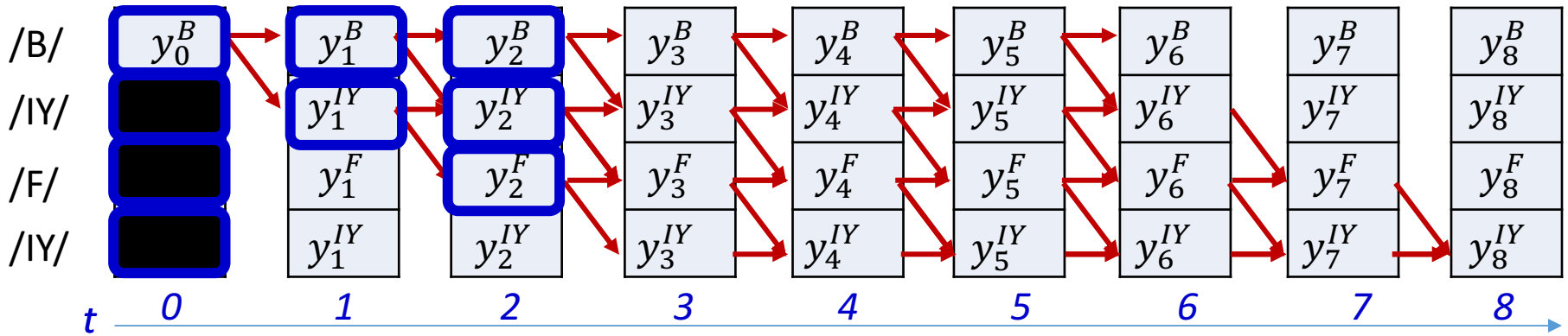
- for $t = 1 \ldots T-1$

$$\alpha(t,1) = \alpha(t-1,1) y_t^{S(1)}$$

for $l = 2 \ldots K$

- $\alpha(t,l) = (\alpha(t-1,l) + \alpha(t-1,l-1)) y_t^{S(l)}$

Forward algorithm



- Initialization:

$$\alpha(0,1) = y_0^{S(1)}, \quad \alpha(0,r) = 0, \ r > 1$$

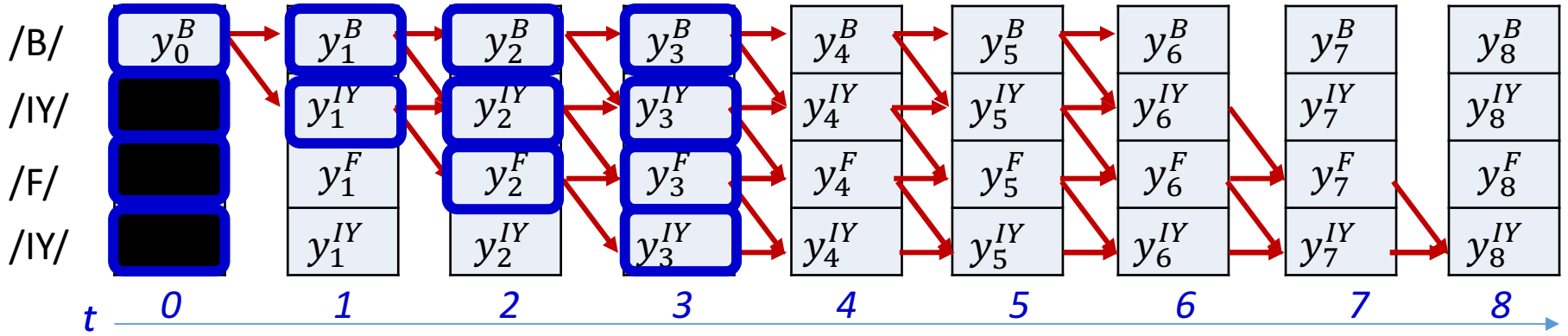
- for $t = 1 \ldots T-1$

$$\alpha(t,1) = \alpha(t-1,1) y_t^{S(1)}$$

for $l = 2 \ldots K$

- $\alpha(t,l) = (\alpha(t-1,l) + \alpha(t-1,l-1)) y_t^{S(l)}$

Forward algorithm



- Initialization:

$$\alpha(0,1) = y_0^{S(1)}, \quad \alpha(0,r) = 0, \; r > 1$$

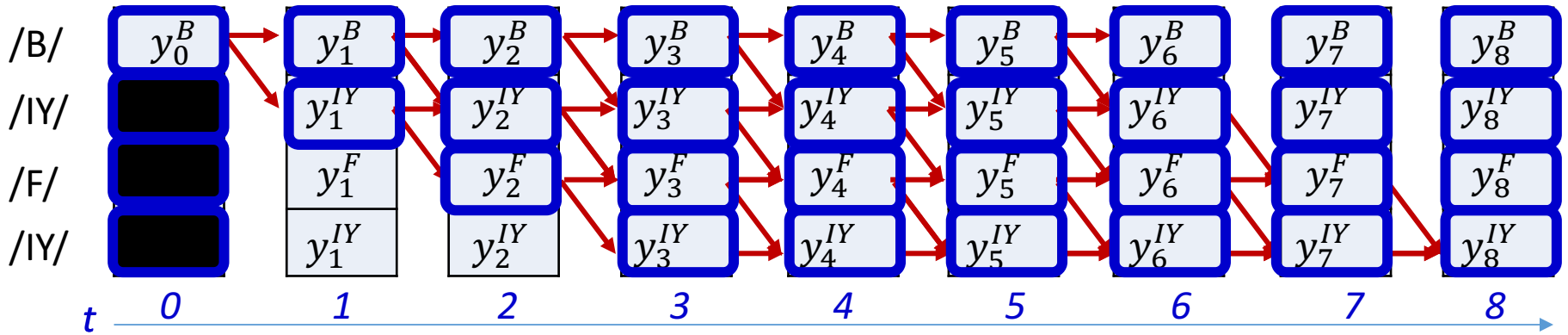
- for $t = 1 \ldots T-1$

$$\alpha(t,1) = \alpha(t-1,1) y_t^{S(1)}$$

for $l = 2 \ldots K$

- $\alpha(t,l) = (\alpha(t-1,l) + \alpha(t-1,l-1)) y_t^{S(l)}$

Forward algorithm



- Initialization:

$$\alpha(0,1) = y_0^{S(1)}, \quad \alpha(0,r) = 0, \; r > 1$$

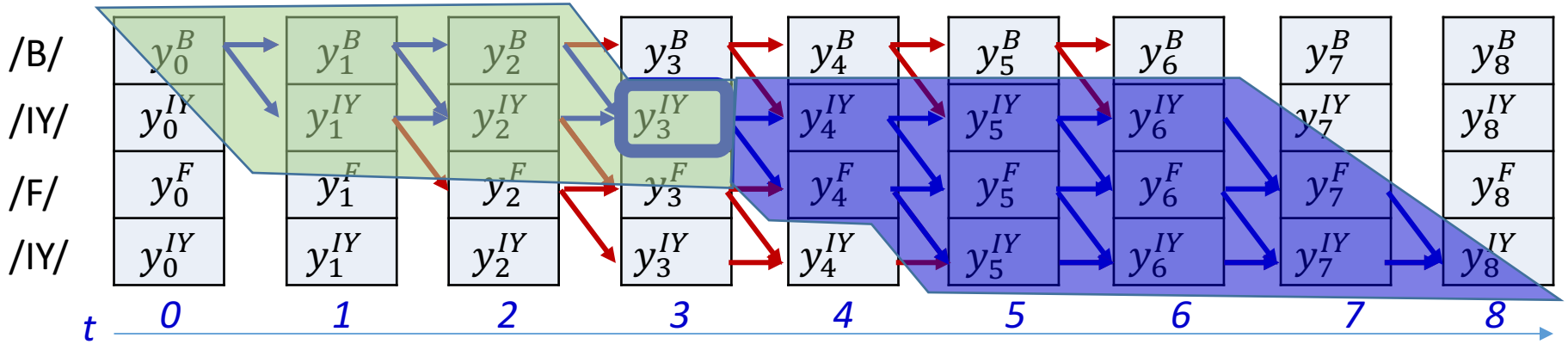
- for $t = 1 \ldots T-1$

$$\alpha(t,1) = \alpha(t-1,1) y_t^{S(1)}$$

for $l = 2 \ldots K$

- $\alpha(t,l) = (\alpha(t-1,l) + \alpha(t-1,l-1)) y_t^{S(l)}$

A posteriori symbol probability

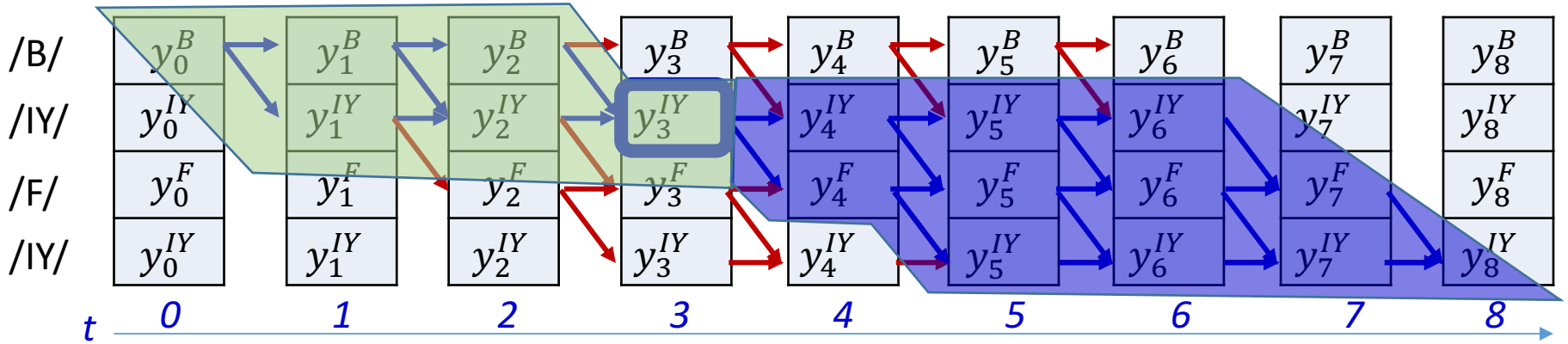


$$P(s_t = S_r, \mathbf{S}|\mathbf{X})$$
$$= P(S_1 \ldots S_r, s_t = S_r \,|\mathbf{X})\, P(s_{t+1} \in succ(S_r), succ(S_r), \ldots, S_K \,|\mathbf{X})$$

- We will call the first term the *forward probability* $\alpha(t,r)$
- We will call the second term the *backward* probability $\beta(t,r)$

We have seen how to compute this

A posteriori symbol probability

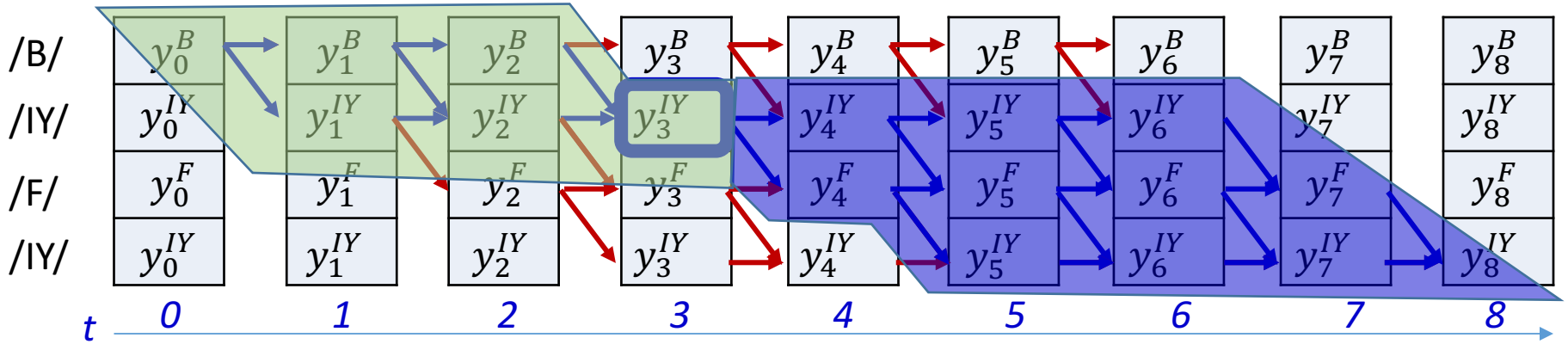


$$P(s_t = S_r, \mathbf{S}|\mathbf{X})$$
$$= \alpha(t,r) P(s_{t+1} \in succ(S_r), succ(S_r), \ldots, S_K \,|\mathbf{X})$$

- We will call the first term the *forward probability* $\alpha(t,r)$
- We will call the second term the *backward* probability $\beta(t,r)$

We have seen how to compute this

A posteriori symbol probability

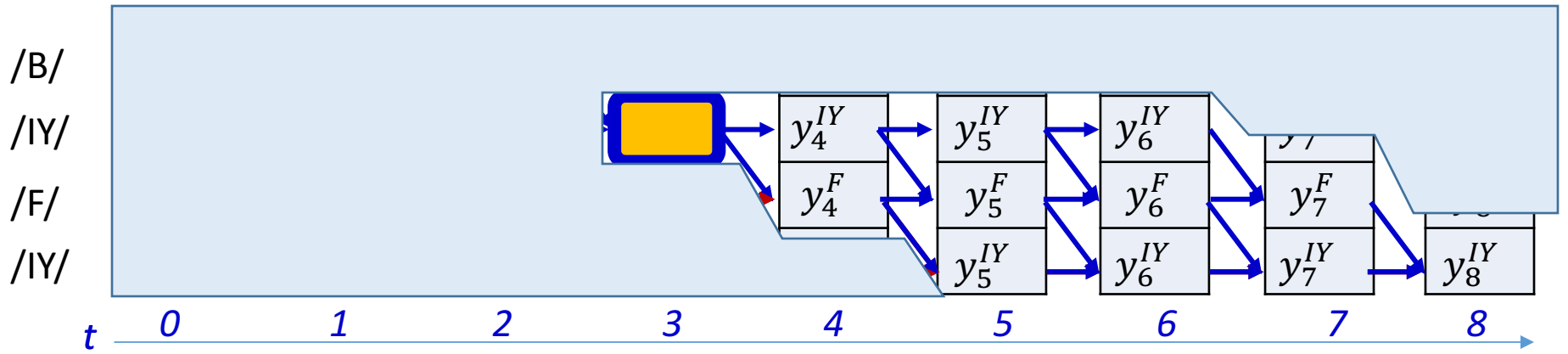


$$P(s_t = S_r, \mathbf{S}|\mathbf{X})$$
$$= \alpha(t,r) P(s_{t+1} \in succ(S_r), succ(S_r), \ldots, S_K \,|\mathbf{X})$$

- We will call the first term the *forward probability* $\alpha(t,r)$
- We will call the second term the *backward* probability $\beta(t,r)$

Lets look at this

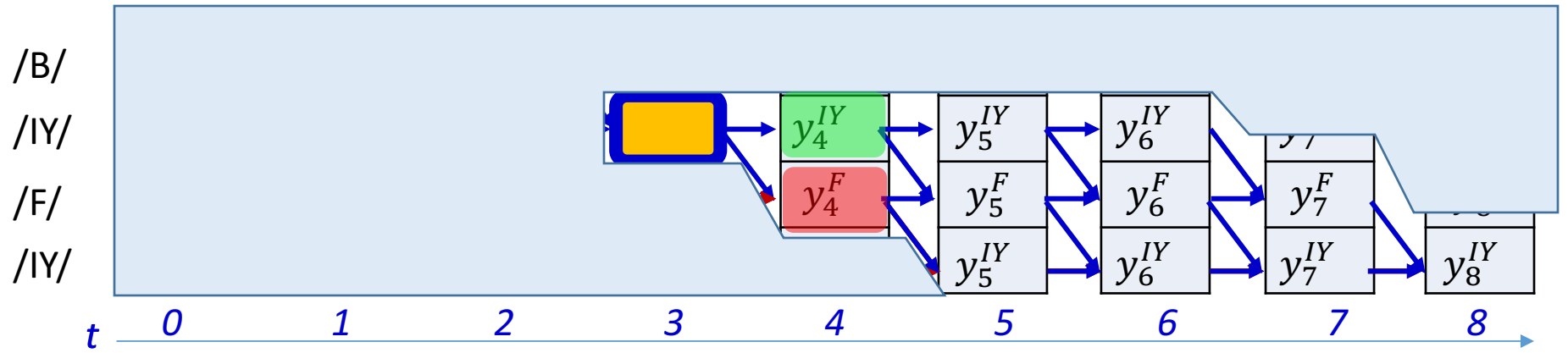
A posteriori symbol probability



$$\beta(t,r) = P(s_{t+1} \in succ(S_r), succ(S_r), \ldots, S_K \,|\mathbf{X})$$

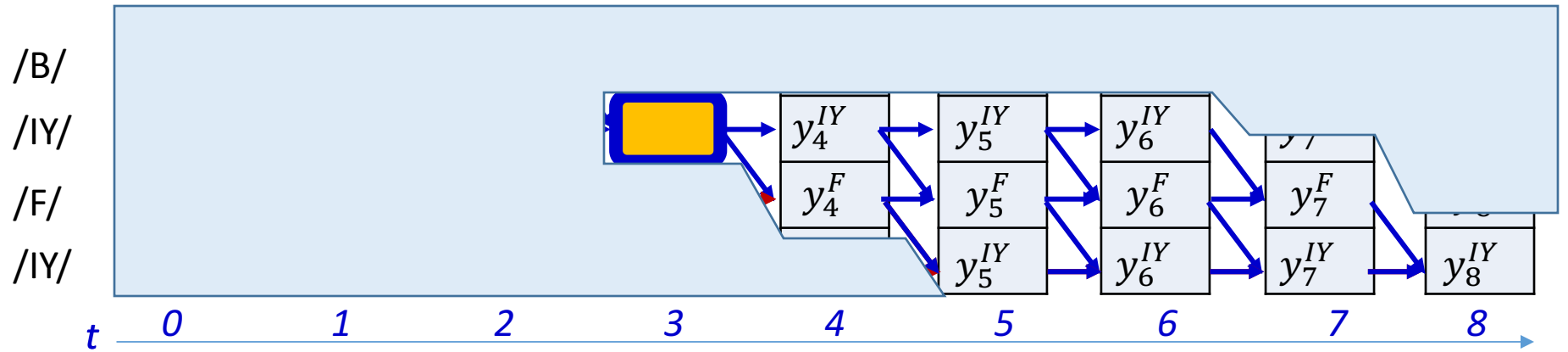
- $\beta(t,r)$ is the probability of the exposed subgraph, not including the orange shaded box

A posteriori symbol probability



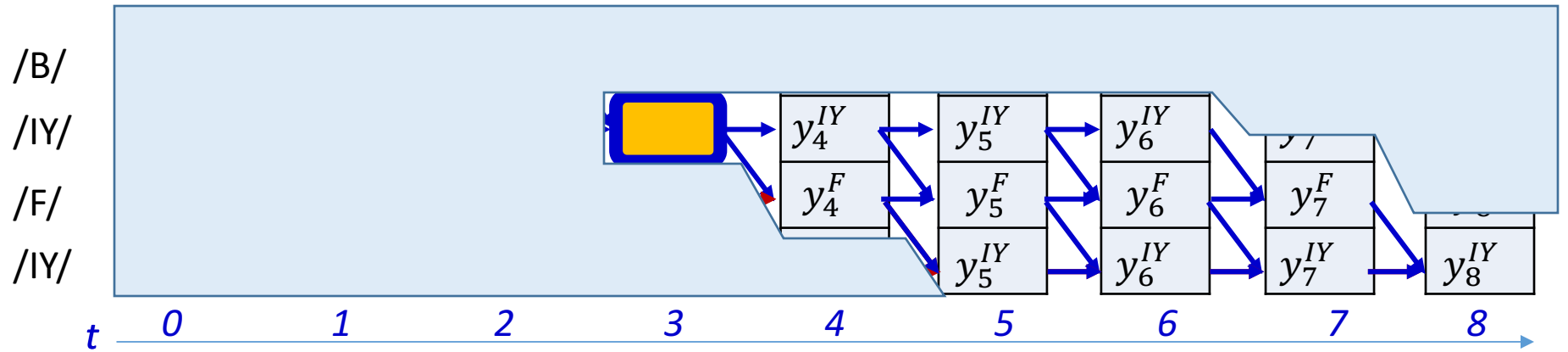
$$\beta(3,1) = P\left(\right) + P\left(\right)$$

A posteriori symbol probability



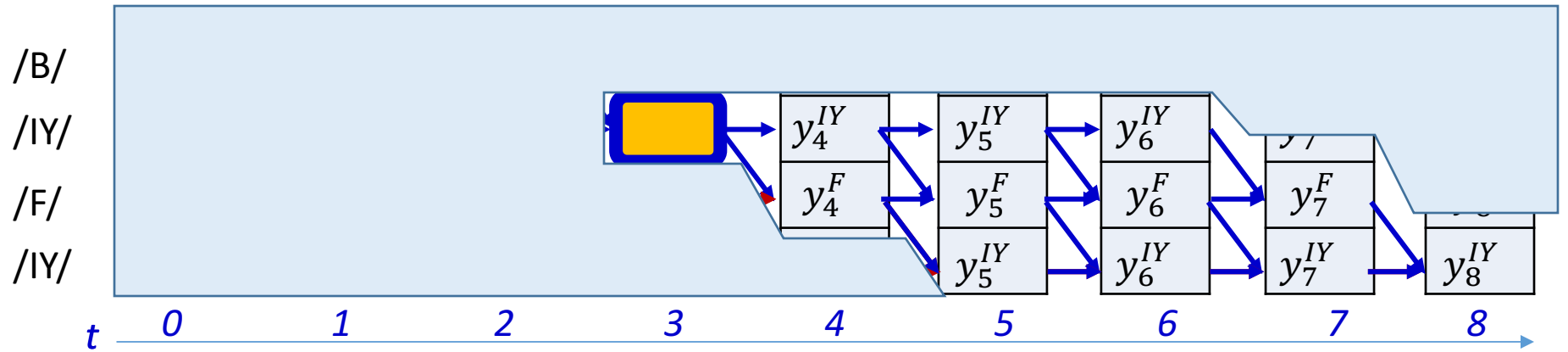
$$\beta(3,1) = P\left(\right) + P\left(\right)$$

A posteriori symbol probability



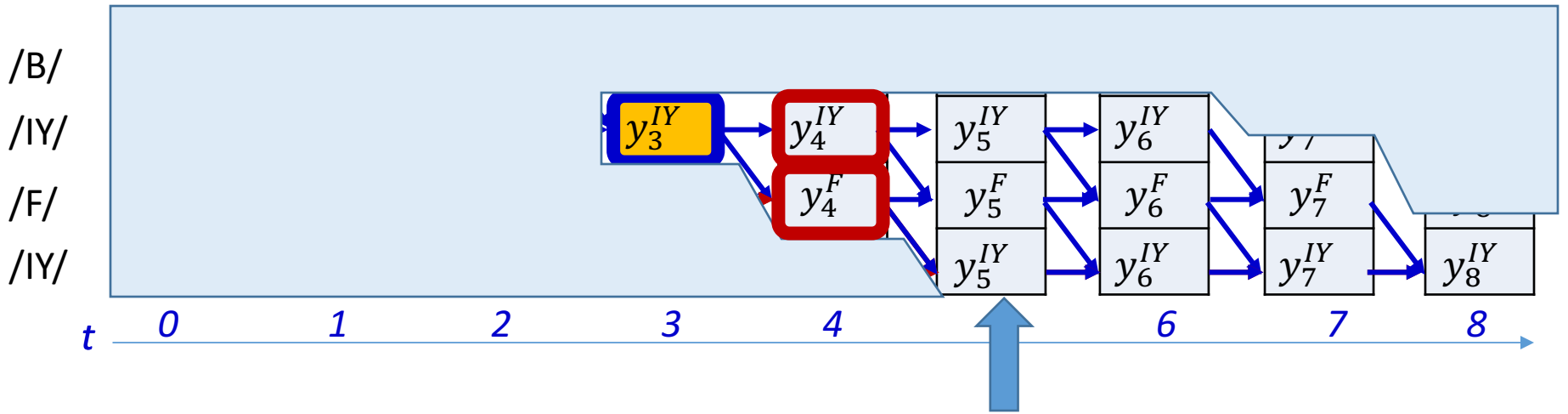
$$\beta(3,1) = y_4^{IY} P\left(\right) + y_4^{F} P\left(\right)$$

A posteriori symbol probability



$$\beta(3,1) = y_4^{IY}\beta(4,1) + y_4^F\beta(4,2)$$

A posteriori symbol probability

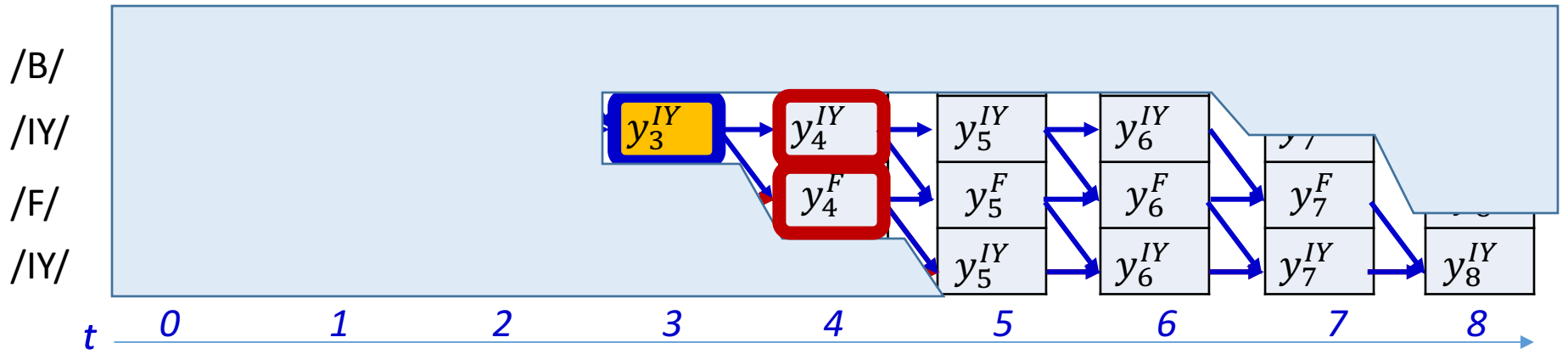


$$\beta(t,r) = \sum_{q:S_q \in succ(S_r)} P\left(s_{t+1} = S_q, S_q, \ldots, S_K | \mathbf{X}\right)$$

- Lets expand this out in terms of the *successors of* S_q at t+2
  - Explicitly consider all possible successors, to cover all possibilities (the two red boxes)

$$\beta(t,r) = \sum_{q:S_q \in succ(S_r)} P\left(s_{t+1} = S_q, s_{t+2} \in succ(S_q), succ(S_q), \ldots, S_K | \mathbf{X}\right)$$

A posteriori symbol probability



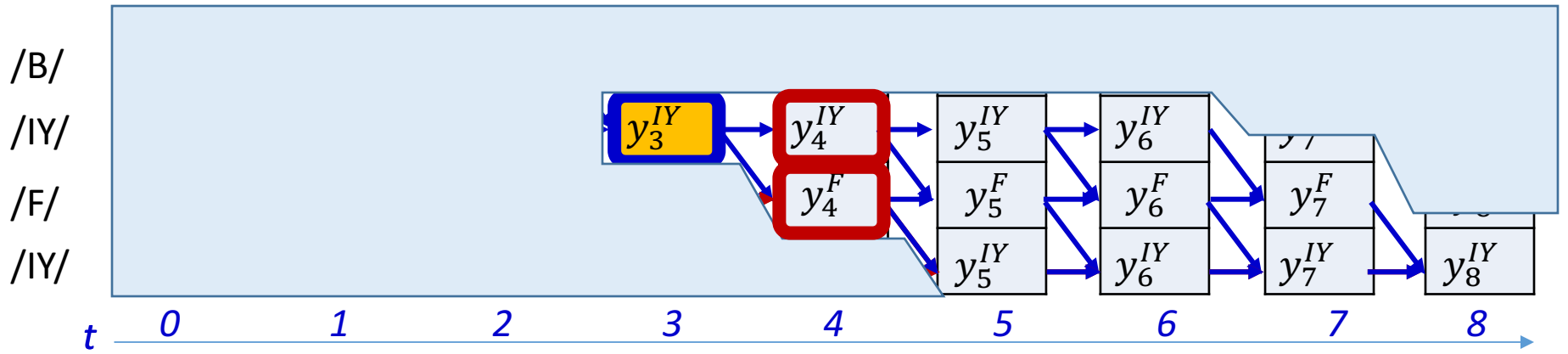
- Expressing $\beta(t,r)$ in terms of the successors of S_q

$$\beta(t,r) = \sum_{q:S_q \in succ(S_r)} P\left(s_{t+1} = S_q, s_{t+2} \in succ(S_q), succ(S_q), \dots, S_K | \mathbf{X}\right)$$

- Using our assumption of conditional independence

$$\beta(t,r) = \sum_{q:S_q \in succ(S_r)} P\left(s_{t+1} = S_q | \mathbf{X}\right) P\left(s_{t+2} \in succ(S_q), succ(S_q), \dots, S_K | \mathbf{X}\right)$$

A posteriori symbol probability



- Expressing $\beta(t,r)$ in terms of the successors of S_q

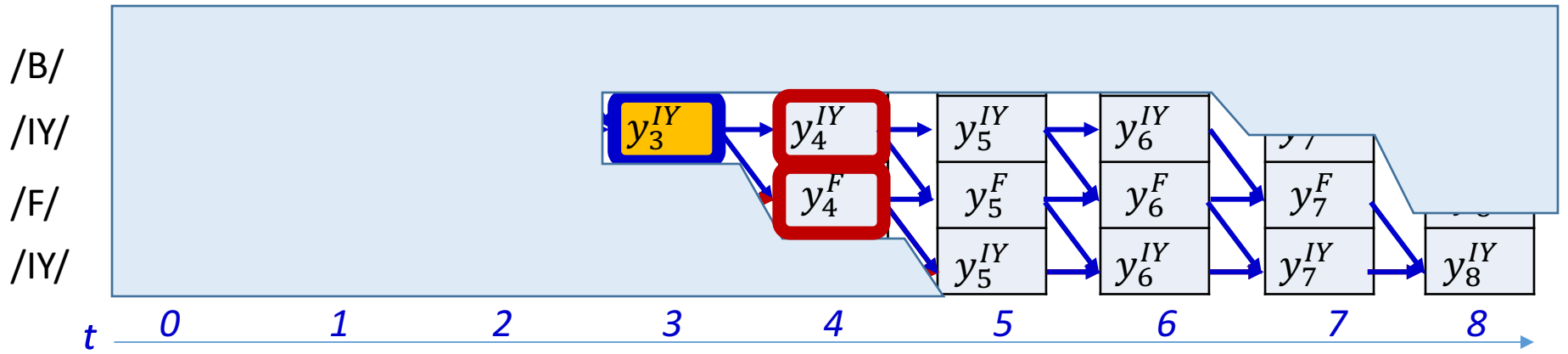
$$\beta(t,r) = \sum_{q:S_q \in succ(S_r)} P\left(s_{t+1} = S_q, s_{t+2} \in succ(S_q), succ(S_q), \dots, S_K | \mathbf{X}\right)$$

- Using our assumption of conditional independence

$$\beta(t,r) = \sum_{q:S_q \in succ(S_r)} P\left(s_{t+1} = S_q | \mathbf{X}\right) P\left(s_{t+2} \in succ(S_q), succ(S_q), \dots, S_K | \mathbf{X}\right)$$

$y_{t+1}^{S_q}$ $\qquad$ $\beta(t+1,q)$

A posteriori symbol probability



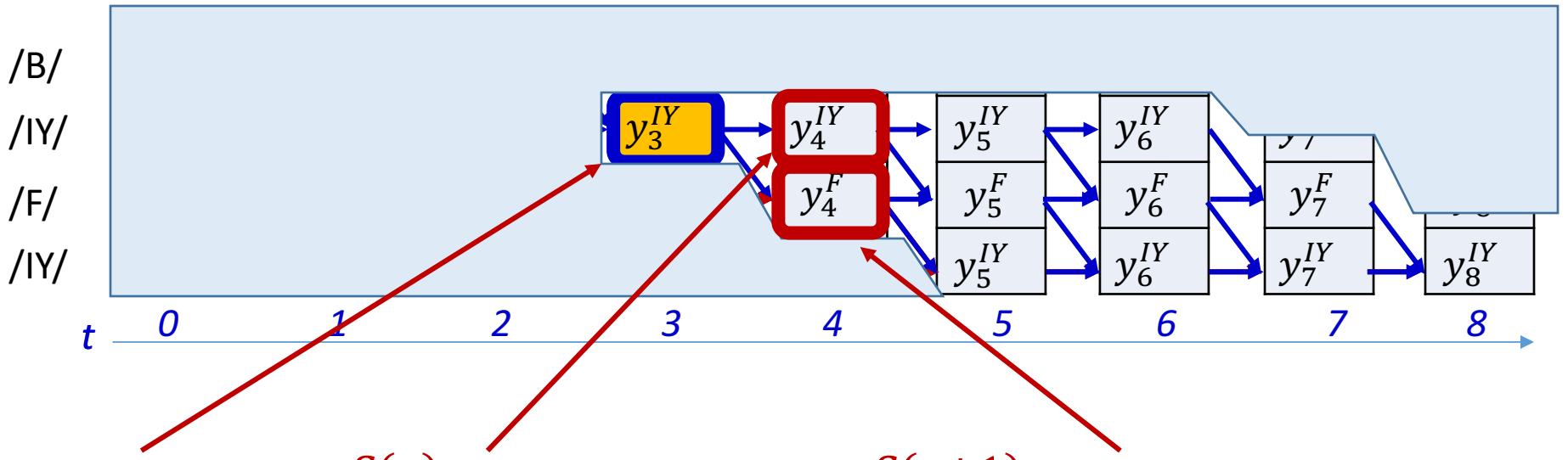
- Expressing $\beta(t,r)$ in terms of the successors of S_q

$$\beta(t,r) = \sum_{q:S_q \in succ(S_r)} P\left(s_{t+1} = S_q, s_{t+2} \in succ(S_q), succ(S_q), \ldots, S_K | \mathbf{X}\right)$$

- Using our assumption of conditional independence

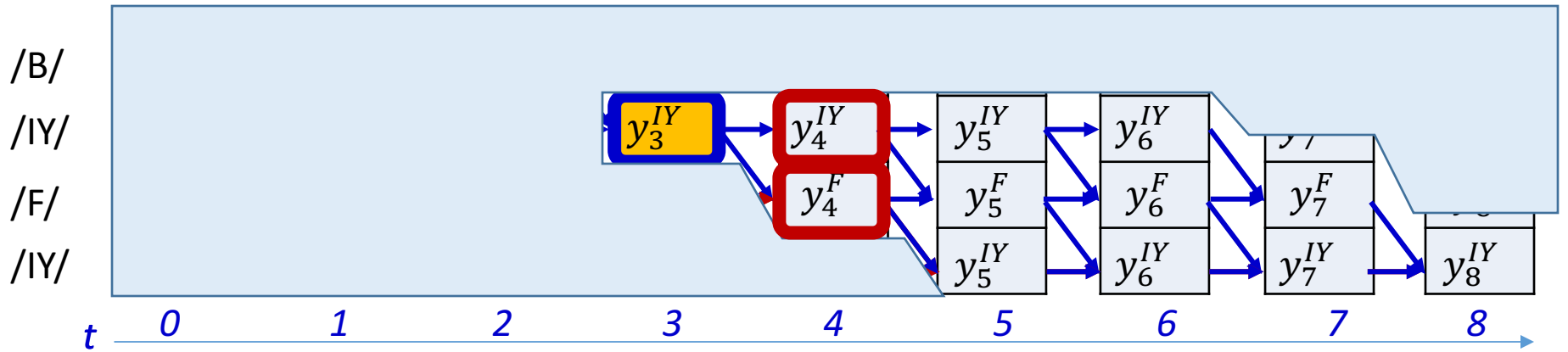
$$\beta(t,r) = \sum_{q:S_q \in succ(S_r)} y_{t+1}^{S_q} \beta(t+1,q)$$

Backward algorithm



$$\beta(t,r) = y_{t+1}^{S(r)}\beta(t+1,r) + y_{t+1}^{S(r+1)}\beta(t+1,r+1)$$

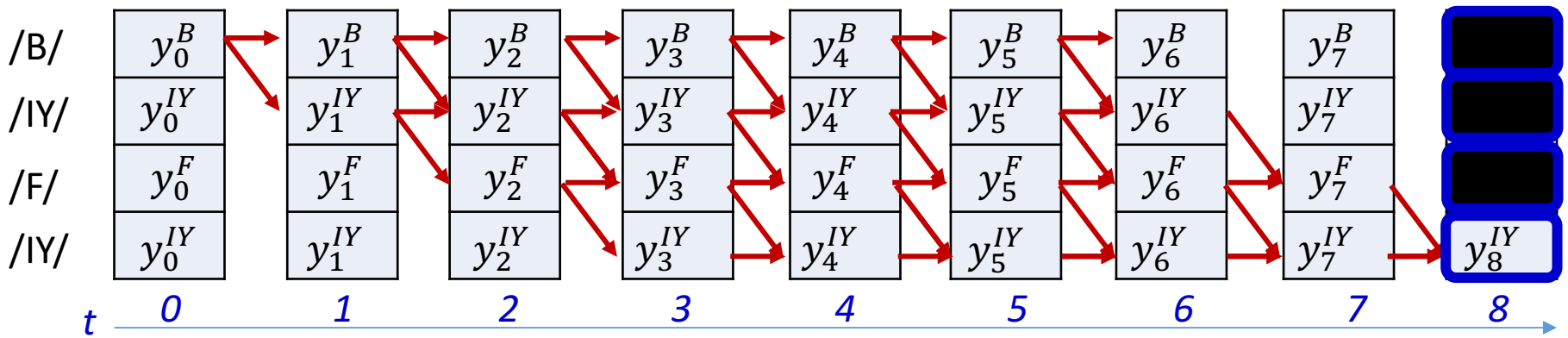
Backward algorithm



$$\beta(t,r) = \sum_{q:S_q \in succ(S_r)} \beta(t+1,q) y_{t+1}^{S_q}$$

- The $\beta(t,r)$ is the total probability of the subgraph shown

Backward algorithm



- Initialization:

$$\beta(T-1,K)=1,\ \beta(T-1,r)=0,\ r<K$$

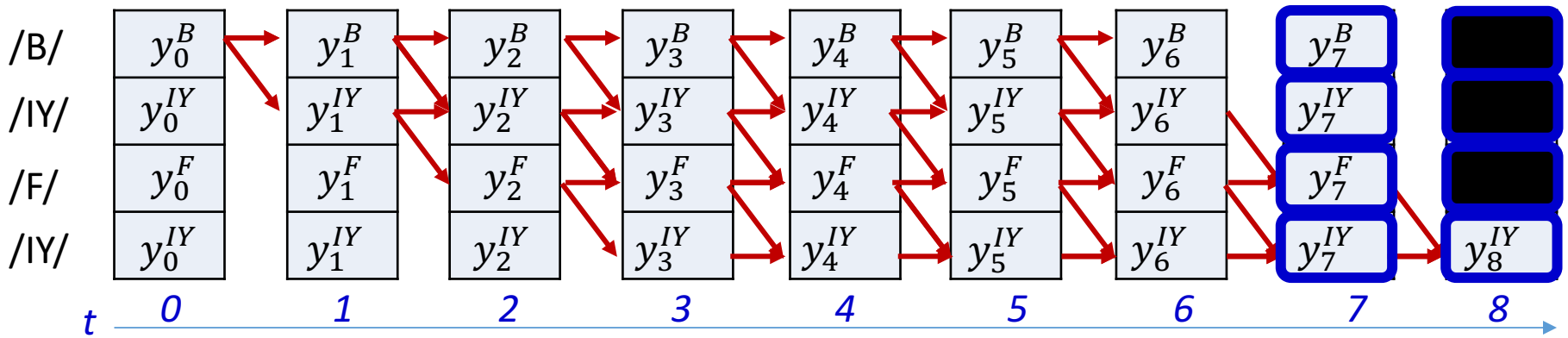
- for $t = T-2\ downto\ 0$

$$\beta(t,K) = \beta(t+1,K)y_{t+1}^{S(K)}$$

for $l = K-1 \ldots 1$

- $\beta(t,r) = y_{t+1}^{S(l)}\beta(t+1,r) + y_{t+1}^{S(r+1)}\beta(t+1,r+1)$

Backward algorithm



- Initialization:

$$\beta(T-1,K)=1,\ \beta(T-1,r)=0,\ r<K$$

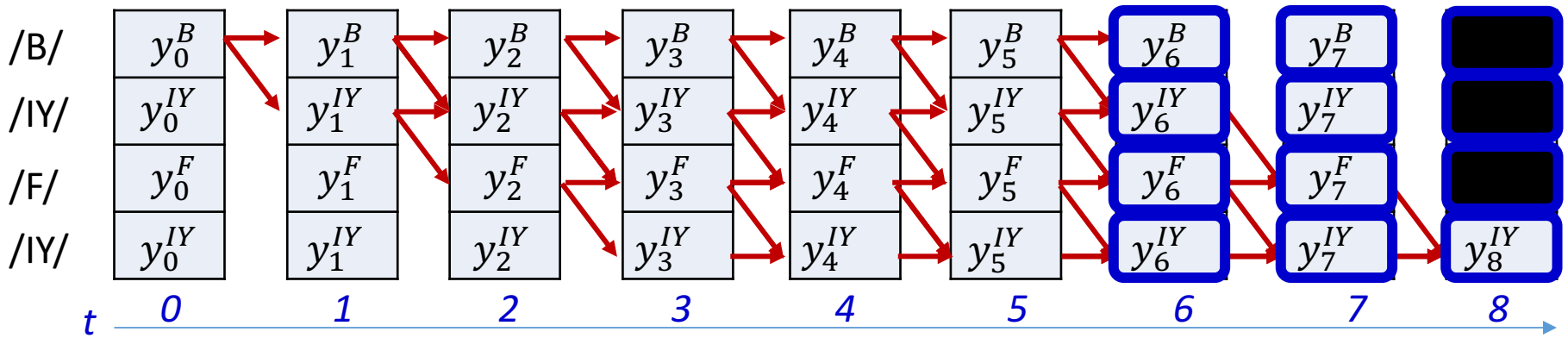
- for $t = T-2\ downto\ 0$

$$\beta(t,K) = \beta(t+1,K)y_{t+1}^{S(K)}$$

for $l = K-1 \ldots 1$

- $\beta(t,r) = y_{t+1}^{S(l)}\beta(t+1,r) + y_{t+1}^{S(r+1)}\beta(t+1,r+1)$

Backward algorithm



- Initialization:

$$\beta(T-1,K)=1,\ \beta(T-1,r)=0,\ r<K$$

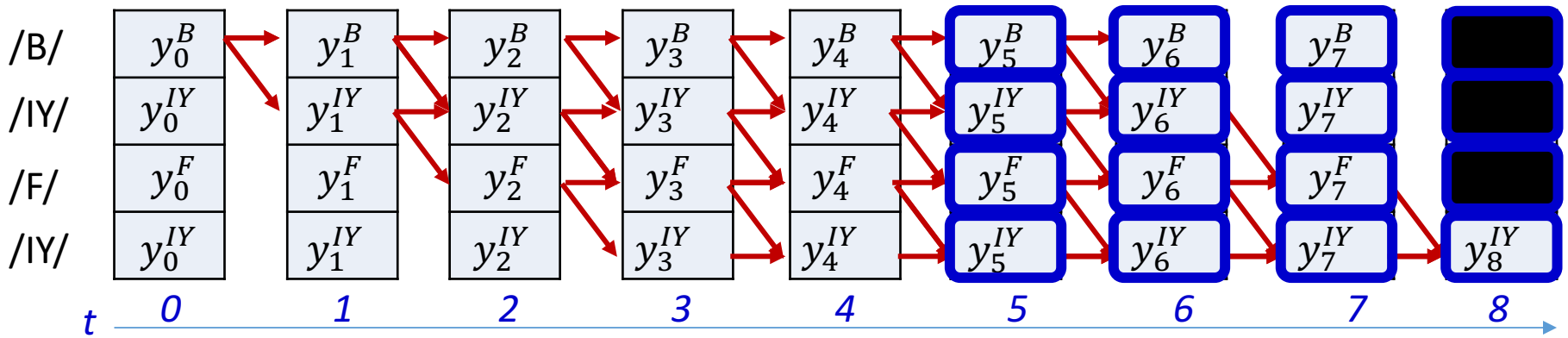
- for $t = T-2\ downto\ 0$

$$\beta(t,K) = \beta(t+1,K)y_{t+1}^{S(K)}$$

for $l = K-1 \ldots 1$

- $\beta(t,r) = y_{t+1}^{S(l)}\beta(t+1,r) + y_{t+1}^{S(r+1)}\beta(t+1,r+1)$

Backward algorithm



- Initialization:

$$\beta(T-1,K)=1,\ \beta(T-1,r)=0,\ r<K$$

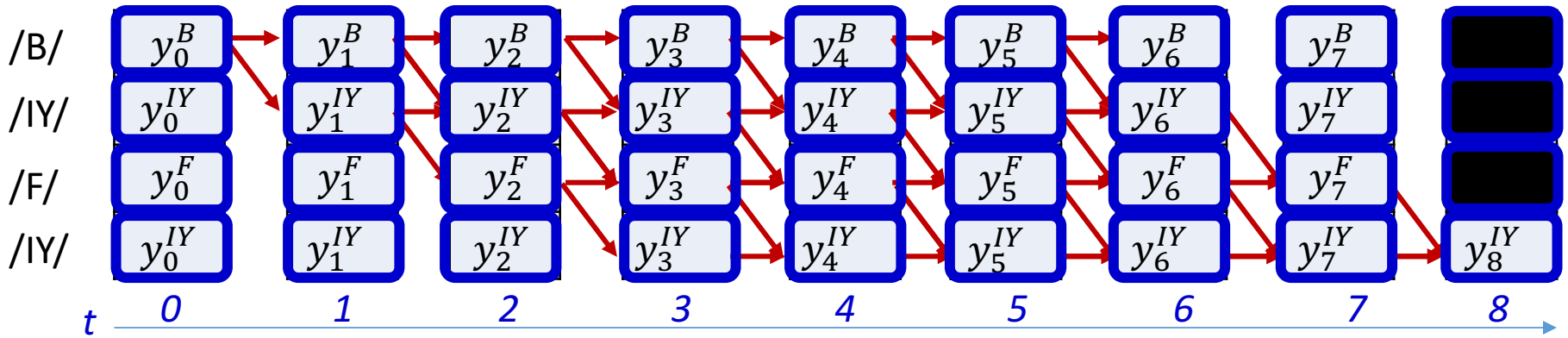
- for $t = T-2\ downto\ 0$

$$\beta(t,K) = \beta(t+1,K)y_{t+1}^{S(K)}$$

for $l = K-1 \ldots 1$

- $\beta(t,r) = y_{t+1}^{S(l)}\beta(t+1,r) + y_{t+1}^{S(r+1)}\beta(t+1,r+1)$

Backward algorithm



- Initialization:

$$\beta(T-1,K)=1,\ \beta(T-1,r)=0,\ r<K$$

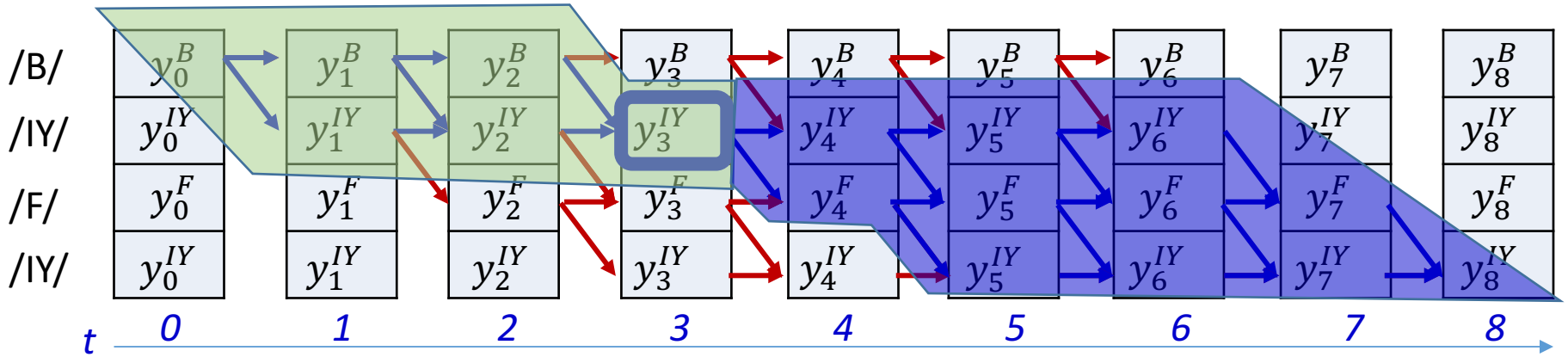
- for $t = T-2\ downto\ 0$

$$\beta(t,K) = \beta(t+1,K)y_{t+1}^{S(K)}$$

for $l = K-1 \ldots 1$

- $\beta(t,r) = y_{t+1}^{S(l)}\beta(t+1,r) + y_{t+1}^{S(r+1)}\beta(t+1,r+1)$

The joint probability

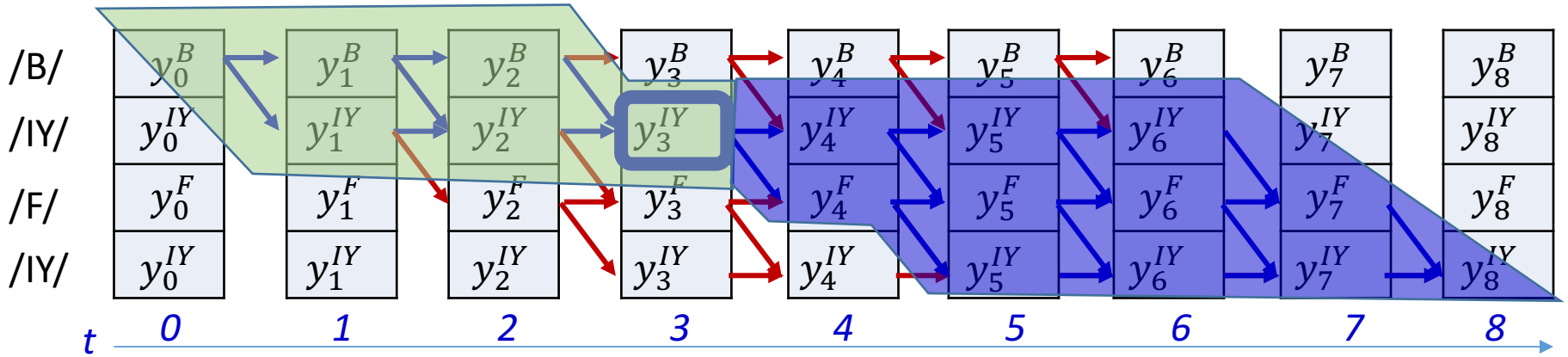


$$P(s_t = S_r, \mathbf{S}|\mathbf{X}) = \alpha(t,r)P(s_{t+1} \in succ(S_r), succ(S_r), \ldots, S_K\ |\mathbf{X})$$

- We will call the first term the *forward probability* $\alpha(t,r)$
- We will call the second term the *backward* probability $\beta(t,r)$

We now can compute this

The joint probability



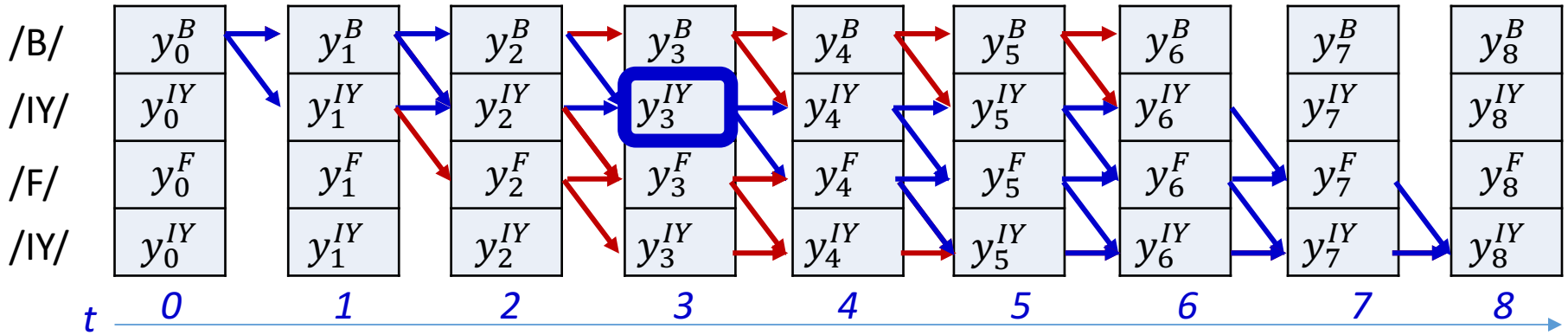
$$P(s_t = S_r, \mathbf{S}|\mathbf{X}) = \alpha(t,r)\beta(t,r)$$

- We will call the first term the *forward probability* $\alpha(t,r)$
- We will call the second term the *backward probability* $\beta(t,r)$

Forward algo

Backward algo

The posterior probability

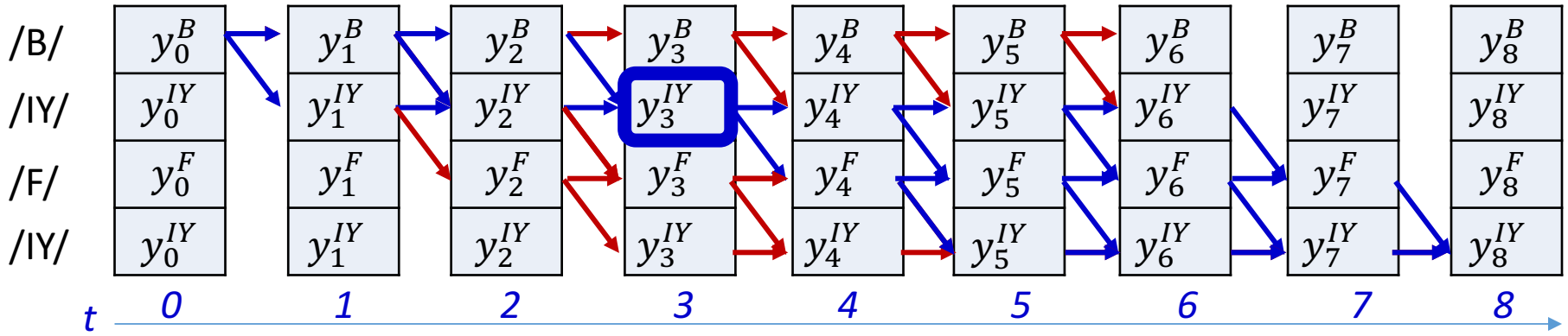


$$P(s_t = S_r, \mathbf{S}|\mathbf{X}) = \alpha(t,r)\beta(t,r)$$

- The *posterior* is given by

$$P(s_t = S_r|\mathbf{S},\mathbf{X}) = \frac{P(s_t = S_r, \mathbf{S}|\mathbf{X})}{\sum_{S_r'} P(s_t = S_r', \mathbf{S}|\mathbf{X})} = \frac{\alpha(t,r)\beta(t,r)}{\sum_{r'} \alpha(t,r')\beta(t,r')}$$

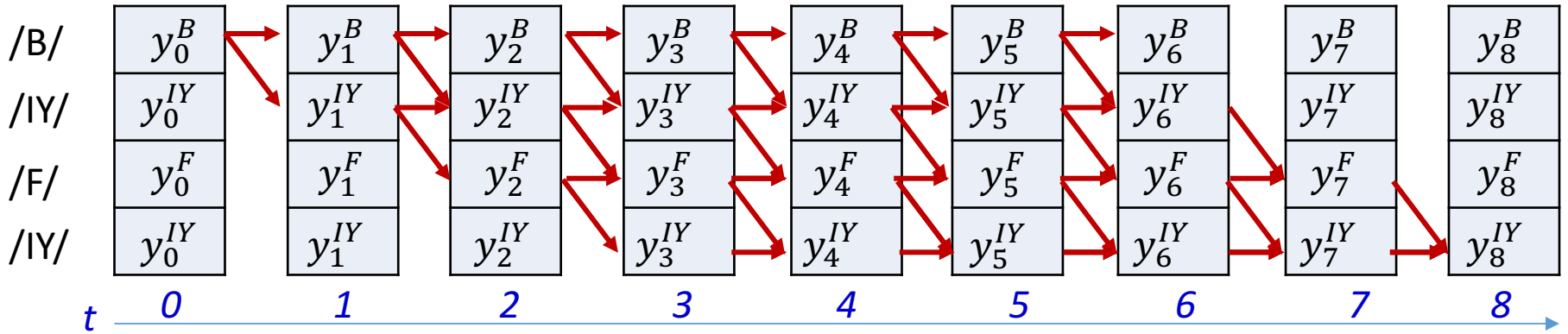
The posterior probability



- Let the posterior $P(s_t = S_r|\mathbf{S}, \mathbf{X})$ be represented by $\gamma(t, r)$

$$\gamma(t,r) = \frac{\alpha(t,r)\beta(t,r)}{\sum_{r'} \alpha(t,r')\beta(t,r')}$$

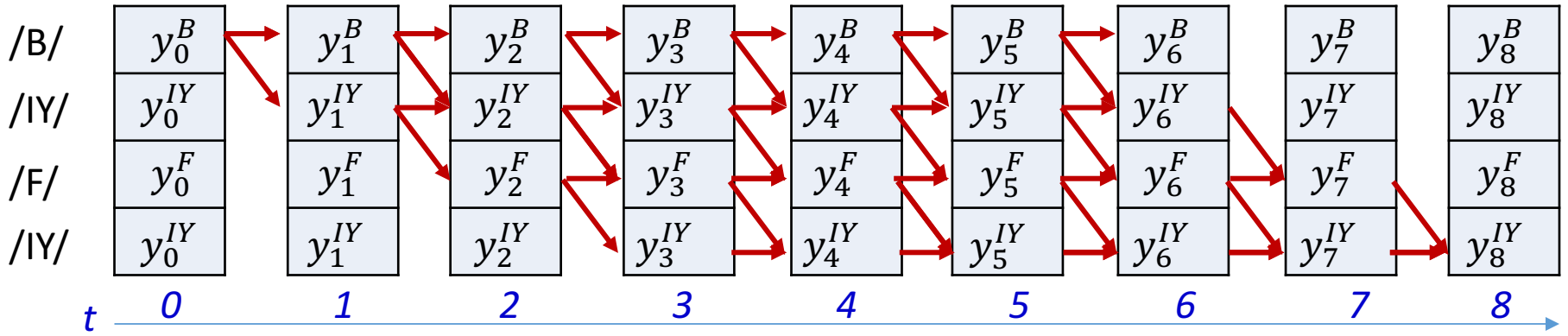
The expected Loss



$$Loss = -\sum_{t}\sum_{s \in S_1 \ldots S_K} P(s_t = s|\mathbf{S}, \mathbf{X}) \log Y(t, s_t = s)$$

$$Loss = -\sum_{t}\sum_{r} \gamma(t, r) \log y_t^{S(r)}$$

The expected Loss



$$Loss = -\sum_{t}\sum_{s \in S_1 \ldots S_K} P(s_t = s|\mathbf{S}, \mathbf{X}) \log Y(t, s_t = s)$$

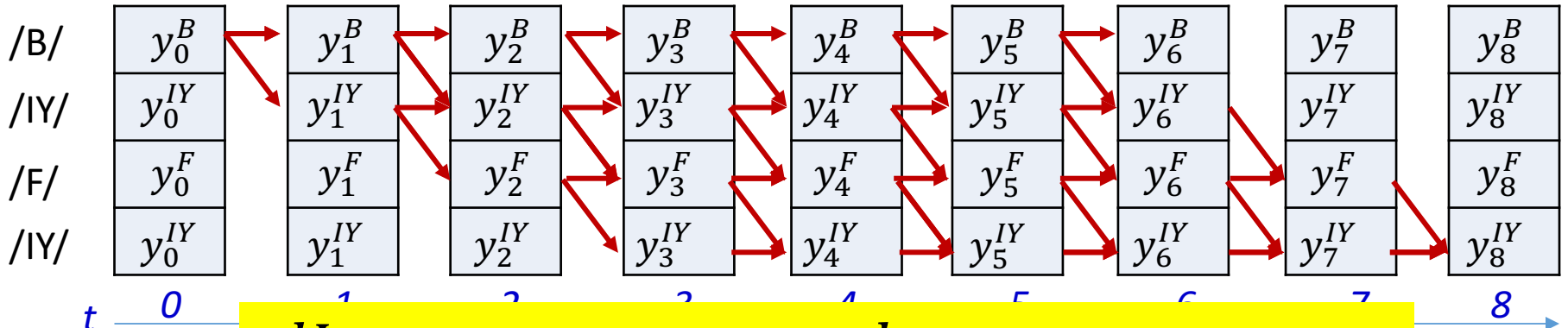
$$Loss = -\sum_{t}\sum_{r} \gamma(t,r) \log y_t^{S(r)}$$

- The derivative of the Loss w.r.t the output Y_t of the net at any time:

$$\nabla_{Y_t} Loss = \left[\frac{dLoss}{dy_t^{s_1}} \quad \frac{dLoss}{dy_t^{s_2}} \quad \ldots \quad \frac{dLoss}{dy_t^{s_L}}\right]$$

  - Components will be non-zero only for symbols that occur in the training instance

The expected Loss



$$\frac{dLoss}{dy_t^l} = -\sum_{r\,:S(r)=l} \frac{d}{dy_t^{S(r)}} \gamma(t,r) \log y_t^{S(r)}$$

$$Loss = -\sum_t \sum_r \gamma(t,r) \log y_t^{S(r)}$$

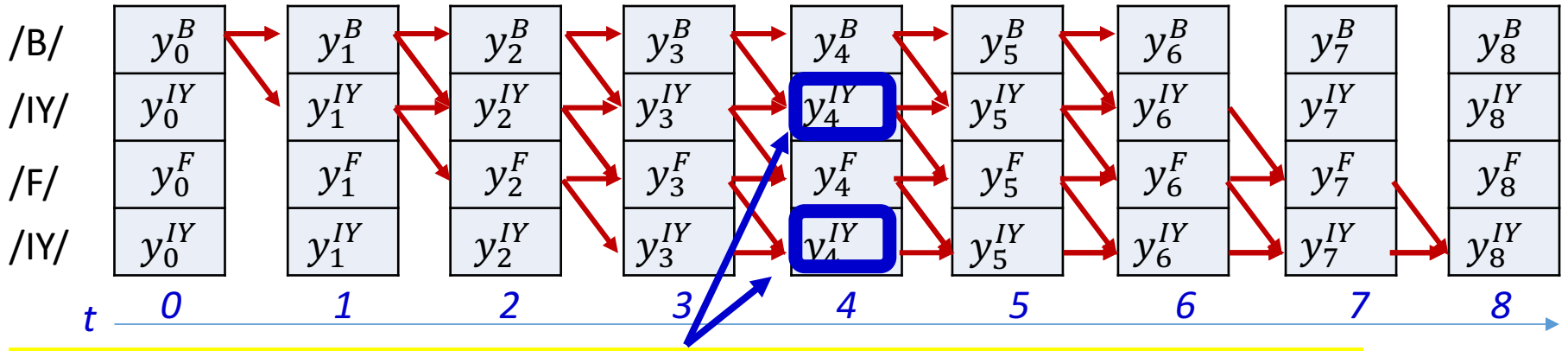
- The derivative of the Loss w.r.t the output Y_t of the net at any time:

$$\nabla_{Y_t} Loss = \left[\frac{dLoss}{dy_t^{s_1}} \quad \frac{dLoss}{dy_t^{s_2}} \quad \dots \right]$$

Must compute these terms from here

  - Components will be non-zero only for symbols that occur in the training instance

The expected Loss



The derivatives at both these locations must be summed to get $\frac{dLoss}{dy_4^{IY}}$

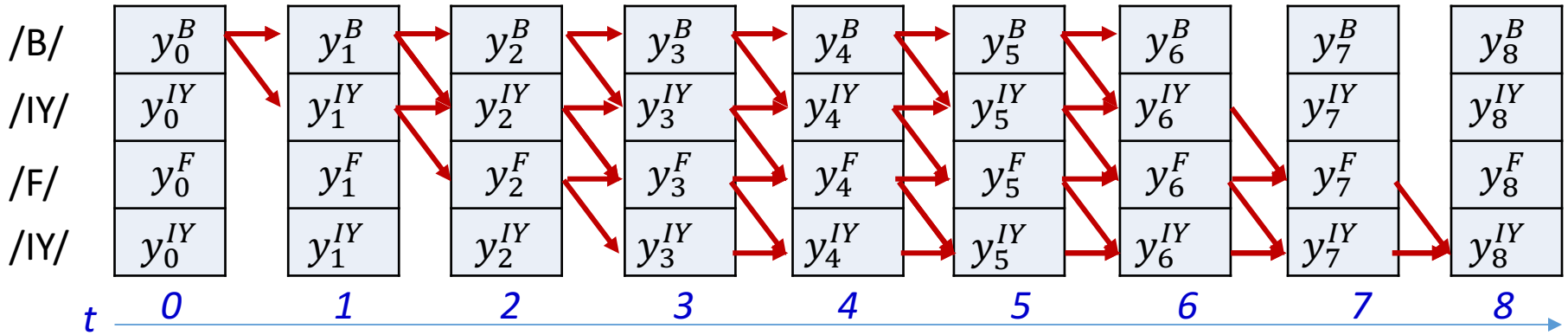
$$\frac{dLoss}{dy_t^l} = -\sum_{r:S(r)=l} \frac{d}{dy_t^{S(r)}} \gamma(t,r) \log y_t^{S(r)}$$

- The derivative of the Loss w.r.t the output Y_t of the net at any time:

$$\nabla_{Y_t} Loss = \left[\frac{dLoss}{dy_t^{s_1}} \quad \frac{dLoss}{dy_t^{s_2}} \quad \dots \quad \frac{dLoss}{dy_t^{s_L}} \right]$$

  - Components will be non-zero only for symbols that occur in the training instance

The expected Loss



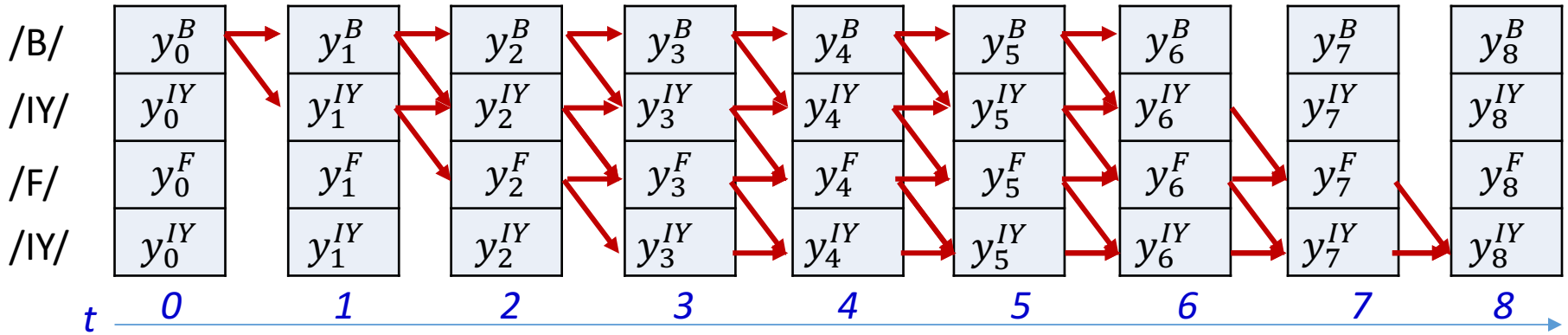
$$Loss = -\sum_{t}\sum_{s} P(s_t = s|\mathbf{S},\mathbf{X}) \log Y(t, s_t = s)$$

$$\frac{dLoss}{dy_t^l} = -\sum_{r:S(r)=l} \frac{d}{dy_t^{S(r)}} \gamma(t,r) \log y_t^{S(r)}$$

- $$\frac{d}{dy_t^{S(r)}} \gamma(t,r) \log y_t^{S(r)} = \frac{\gamma(t,r)}{y_t^{S(r)}} + \frac{d\gamma(t,r)}{dy_t^{S(r)}} \log y_t^{S(r)}$$

  - Components will be non-zero only for symbols that occur in the training instance

The expected Loss



$$Loss = -\sum_{t}\sum_{s} P(s_t = s|\mathbf{S},\mathbf{X}) \log Y(t, s_t = s)$$

$$\frac{dLoss}{dy_t^l} = -\sum_{r:S(r)=l} \frac{d}{dy_t^{S(r)}} \gamma(t,r) \log y_t^{S(r)}$$

- The derivative

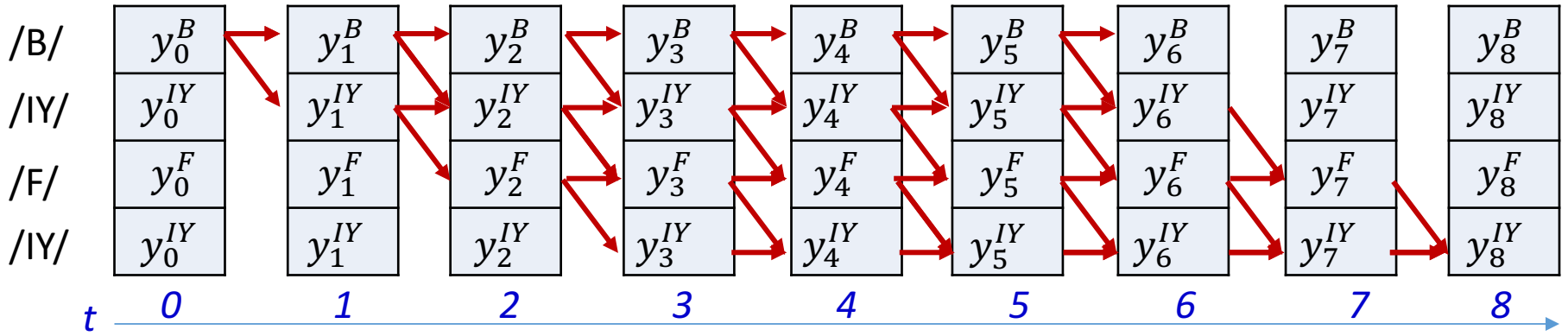
$$\frac{d}{dy_t^{S(r)}} \gamma(t,r) \log y_t^{S(r)} \approx \frac{\gamma(t,r)}{y_t^{S(r)}}$$

ime:

The approximation is exact if we think of this as a maximum-likelihood estimate

– Components will be non-zero only for symbols that occur in the training instance

The expected Loss



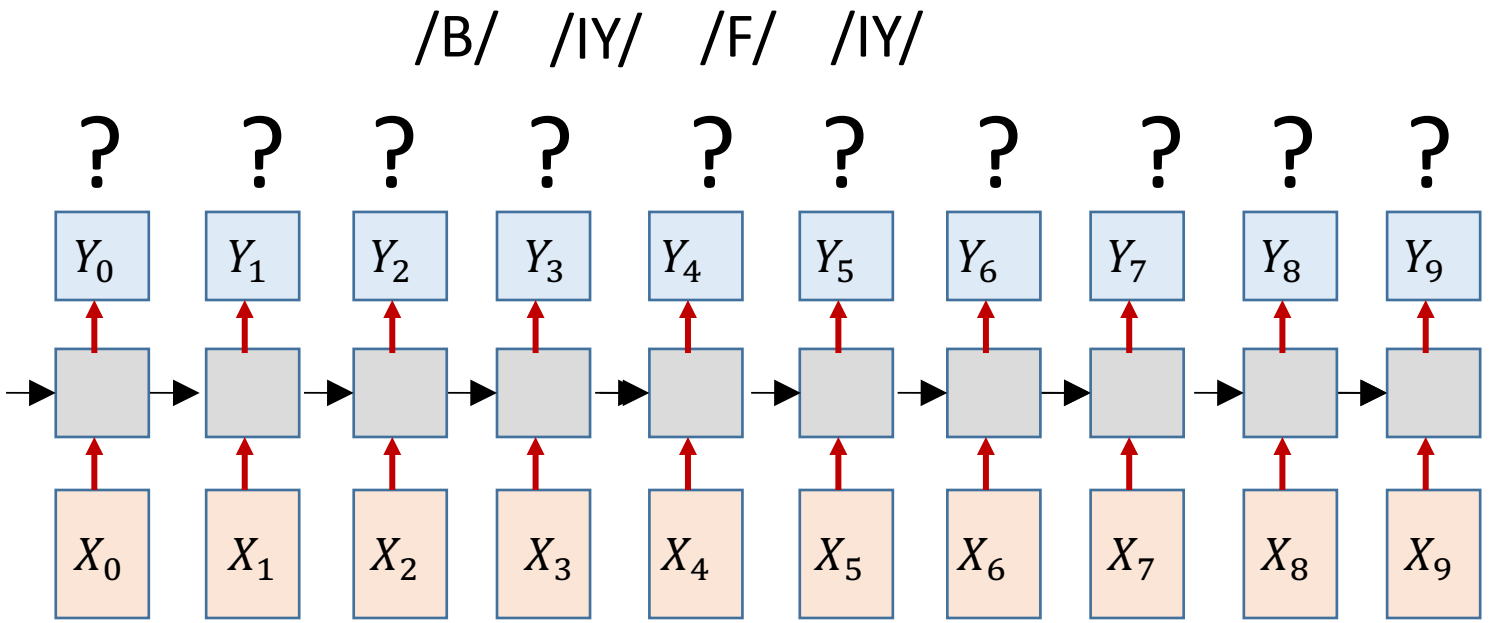
$$Loss = -\sum_t \sum_r \gamma(t,r) \log y_t^{S(r)}$$

- The derivative of the Loss w.r.t any particular output of the network must sum over all instances of that symbol in the target sequence

$$\frac{dLoss}{dy_t^l} = -\sum_{r\,:S(r)=l} \frac{\gamma(t,r)}{y_t^{S(r)}}$$

  - E.g. the derivative w.r.t y_t^{IY} will sum over both rows representing /IY/ in the above figure

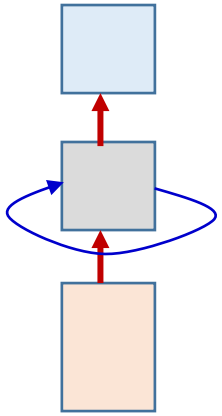
Overall training procedure for Seq2Seq case 1



- Problem: Given input and output sequences without alignment, train models

Overall training procedure for Seq2Seq

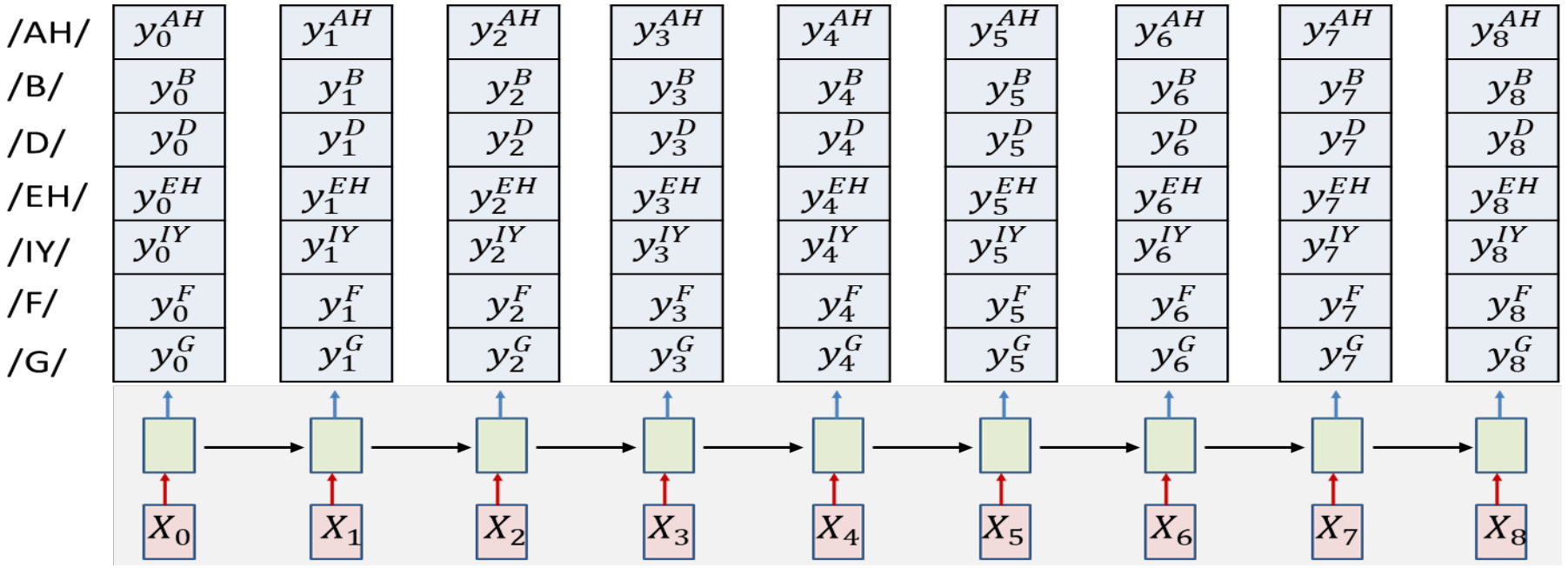
- **Step 1**: Setup the network
  - Typically many-layered LSTM



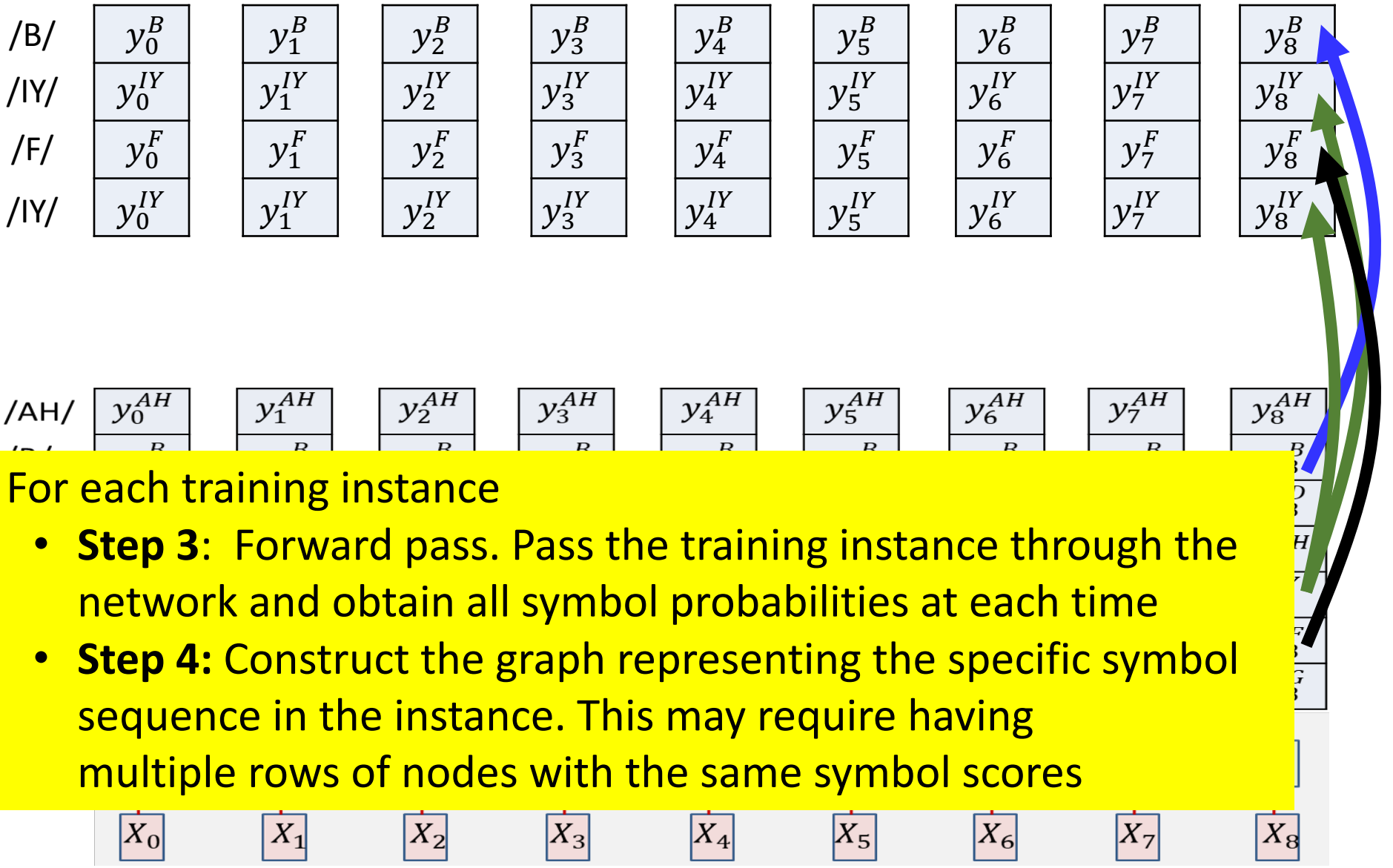
- **Step 2**: Initialize all parameters of the network

Overall Training: Forward pass

- For each training instance
  - **Step 3**: Forward pass. Pass the training instance through the network and obtain all symbol probabilities at each time

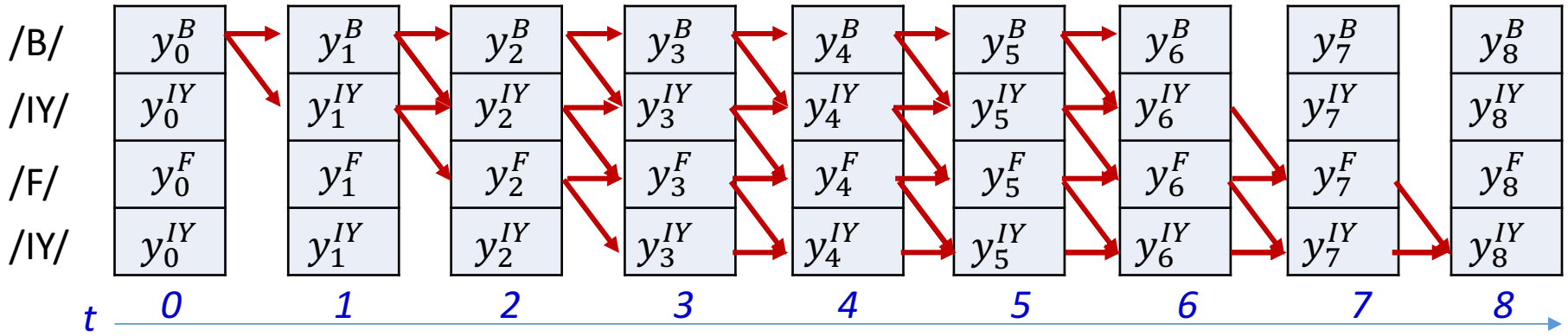


Overall training: Backward pass



- For each training instance
  - **Step 3**: Forward pass. Pass the training instance through the network and obtain all symbol probabilities at each time
  - **Step 4:** Construct the graph representing the specific symbol sequence in the instance. This may require having multiple rows of nodes with the same symbol scores

Overall training: Backward pass



- Foreach training instance:
  - **Step 5:** Perform the forward backward algorithm to compute $\alpha(t,r)$ and $\beta(t,r)$ at each time, for each row of nodes in the graph
  - **Step 6:** Compute derivative of Loss $\nabla_{Y_t} Loss$ for each Y_t

Overall training: Backward pass

- Foreach instance
  - **Step 6:** Compute derivative of Loss $\nabla_{Y_t} Loss$ for each Y_t

$$\nabla_{Y_t} Loss = \left[\frac{dLoss}{dy_t^1} \quad \frac{dLoss}{dy_t^2} \quad \dots \quad \frac{dLoss}{dy_t^L} \right]$$

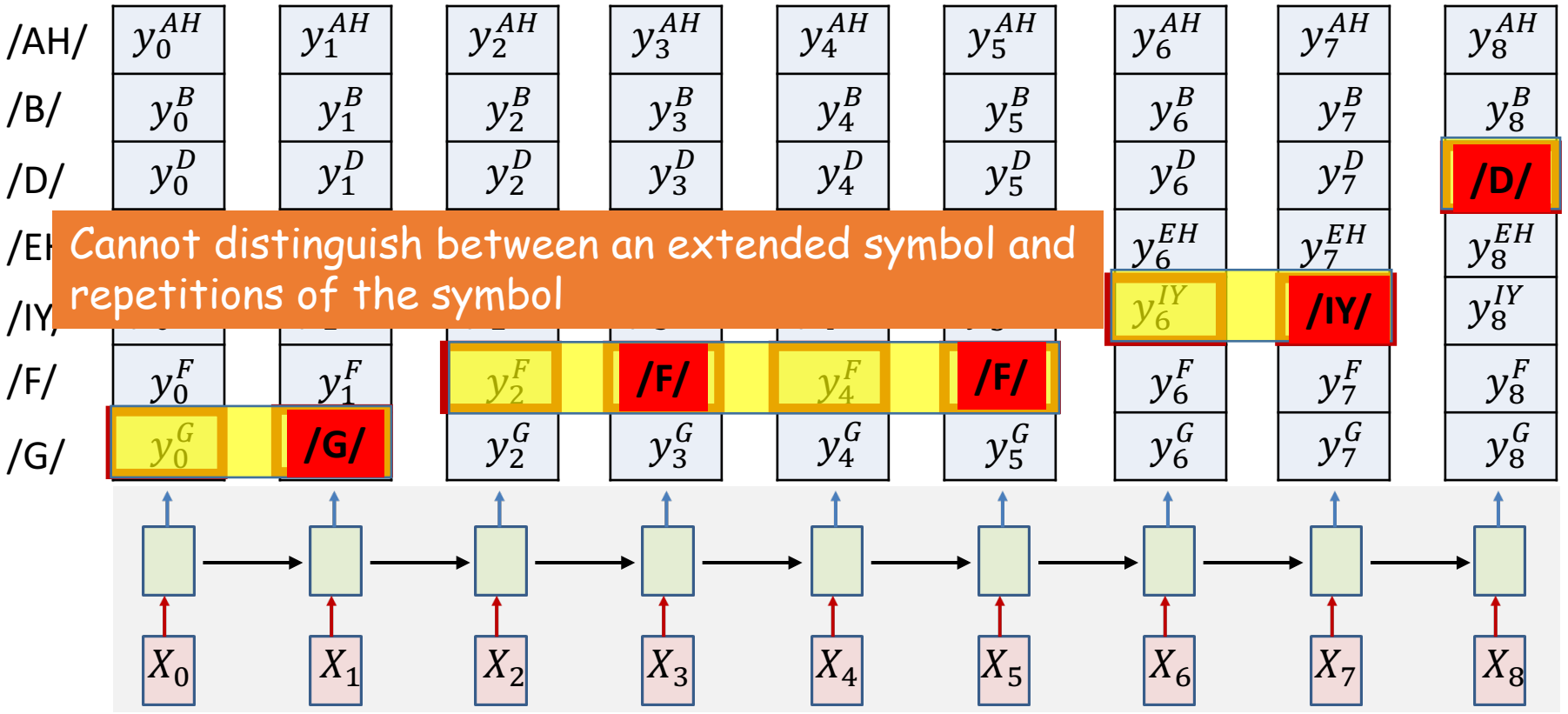
$$\frac{dLoss}{dy_t^l} = -\sum_{r\,:S(r)=l} \frac{\gamma(t,r)}{y_t^{S(r)}}$$

- **Step 7**: Aggregate derivatives over minibatch and update parameters

A key *decoding* problem

- Consider a problem where the output symbols are characters
- We have a decode: R R R O O O O O D
- Is this the merged symbol sequence ROD or ROOD?

We've seen this before



- /G/ /F/ /F/ /IY/ /D/ or /G/ /F/ /IY/ /D/ ?

A key *decoding* problem

- We have a decode: R R R O O O O O D
- Is this the symbol sequence ROD or ROOD?

- A solution: Introduce an explicit extra symbol which serves to separate discrete versions of a symbol
  - A “blank” (represented by “-”)
  - RRR---OO---DDD = ROD
  - RR-R---OO---D-DD = RRODD
  - R-R-R---O-ODD-DDDD-D = RRROODDD
    - The next symbol at the end of a sequence of blanks is always a new character
    - When a symbol repeats, there must be at least one blank between the repetitions

- The symbol set recognized by the network must now include the extra blank symbol
  - Which too must be trained

Most common CTC applications

- Speech recognition
  - Speech in, phoneme sequence out
  - Speech in, character sequence (spelling out)

- Handwriting recognition

Story so far: CTC models

- Sequence-to-sequence networks which irregularly output symbols can be “decoded” by Viterbi decoding
  - Which assumes that a symbol is output at each time and ***merges*** adjacent symbols
- They require alignment of the output to the symbol sequence for training
  - This alignment is generally not given
- Training can be performed by iteratively estimating the alignment by Viterbi-decoding and time-synchronous training
- Alternately, it can be performed by optimizing the expected error over *all* possible alignments
  - Posterior probabilities for the expectation can be computed using the forward backward algorithm

Most common CTC applications

- Speech recognition
  - Speech in, phoneme sequence out
  - Speech in, character sequence (spelling out)

- Handwriting recognition