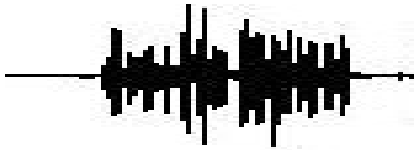
Deep Learning
Sequence to Sequence models: Connectionist Temporal Classification

Sequence-to-sequence modelling

- Problem:
  - A sequence $X_1 \ldots X_N$ goes in
  - A different sequence $Y_1 \ldots Y_M$ comes out
- E.g.
  - Speech recognition: Speech goes in, a word sequence comes out
    - Alternately output may be phoneme or character sequence
  - Machine translation: Word sequence goes in, word sequence comes out
  - Dialog : User statement goes in, system response comes out
  - Question answering : Question comes in, answer goes out
- In general $N \neq M$
  - No synchrony between X and Y.

Sequence to sequence



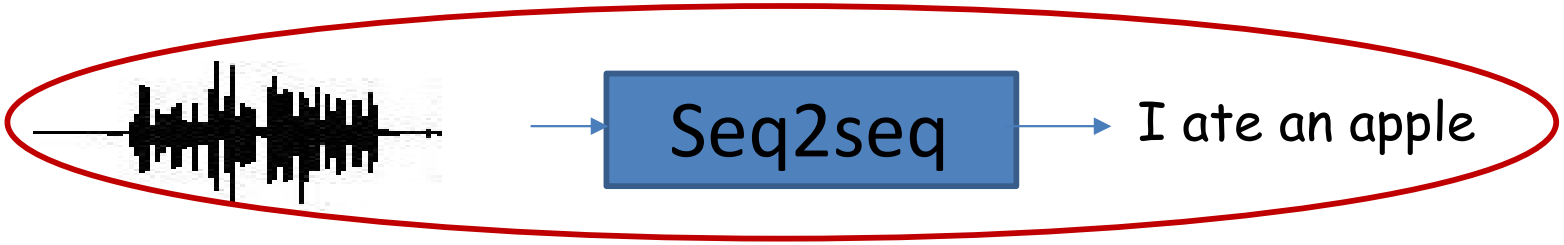
Seq2seq → I ate an apple

I ate an apple → Seq2seq → Ich habe einen apfel gegessen

- Sequence goes in, sequence comes out
- No notion of "time synchrony" between input and output
  - May even not even maintain order of symbols
    - E.g. "I ate an apple" → "Ich habe einen apfel gegessen"

  - Or even seem related to the input
    - E.g. "My screen is blank" → "Please check if your computer is plugged in."

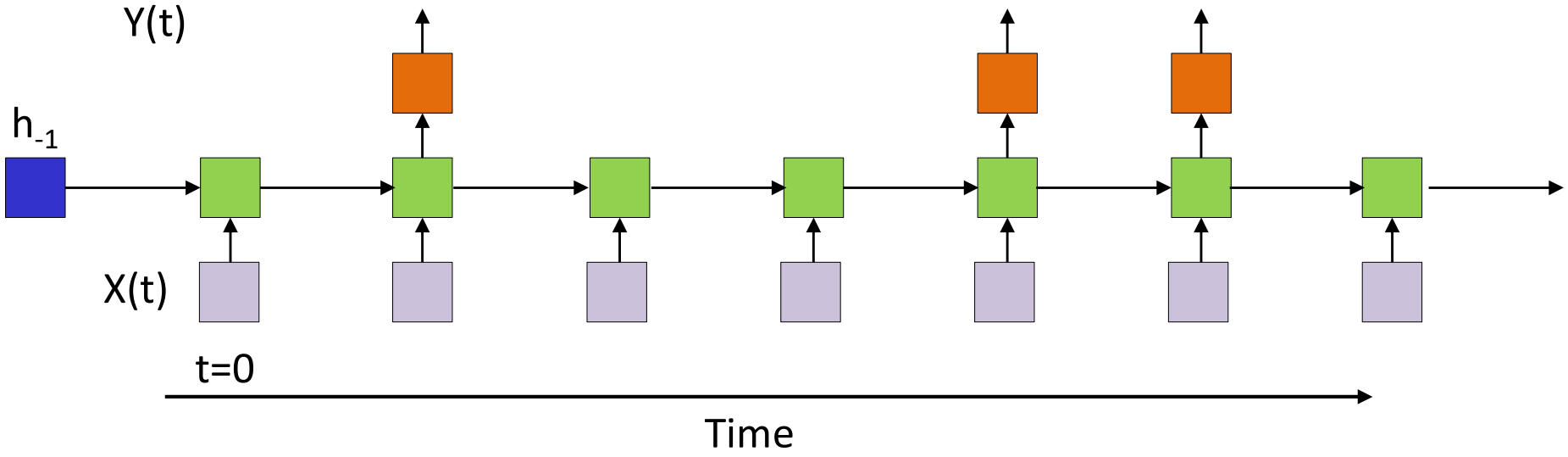
Sequence to sequence



- Sequence goes in, sequence comes out
- No notion of "time synchrony" between input and output
  - May even not even maintain order of symbols
    - E.g. "I ate an apple" → "Ich habe einen apfel gegessen"

  - Or even seem related to the input
    - E.g. "My screen is blank" → "Can you check if your computer is plugged in?"

Case 1: Order-aligned but not time synchronous



- The input and output sequences happen in the same order
  - Although they may not be *time synchronous*, they can be "aligned" against one another
  - E.g. Speech recognition
    - The input speech can be aligned to the phoneme sequence output

Problems

- How do we perform *inference* on such a model
  - How to output time-asynchronous sequences
- How do we *train* such models

Problems

- How do we perform *inference* on such a model Partially addressed
  - How to output time-asynchronous sequences

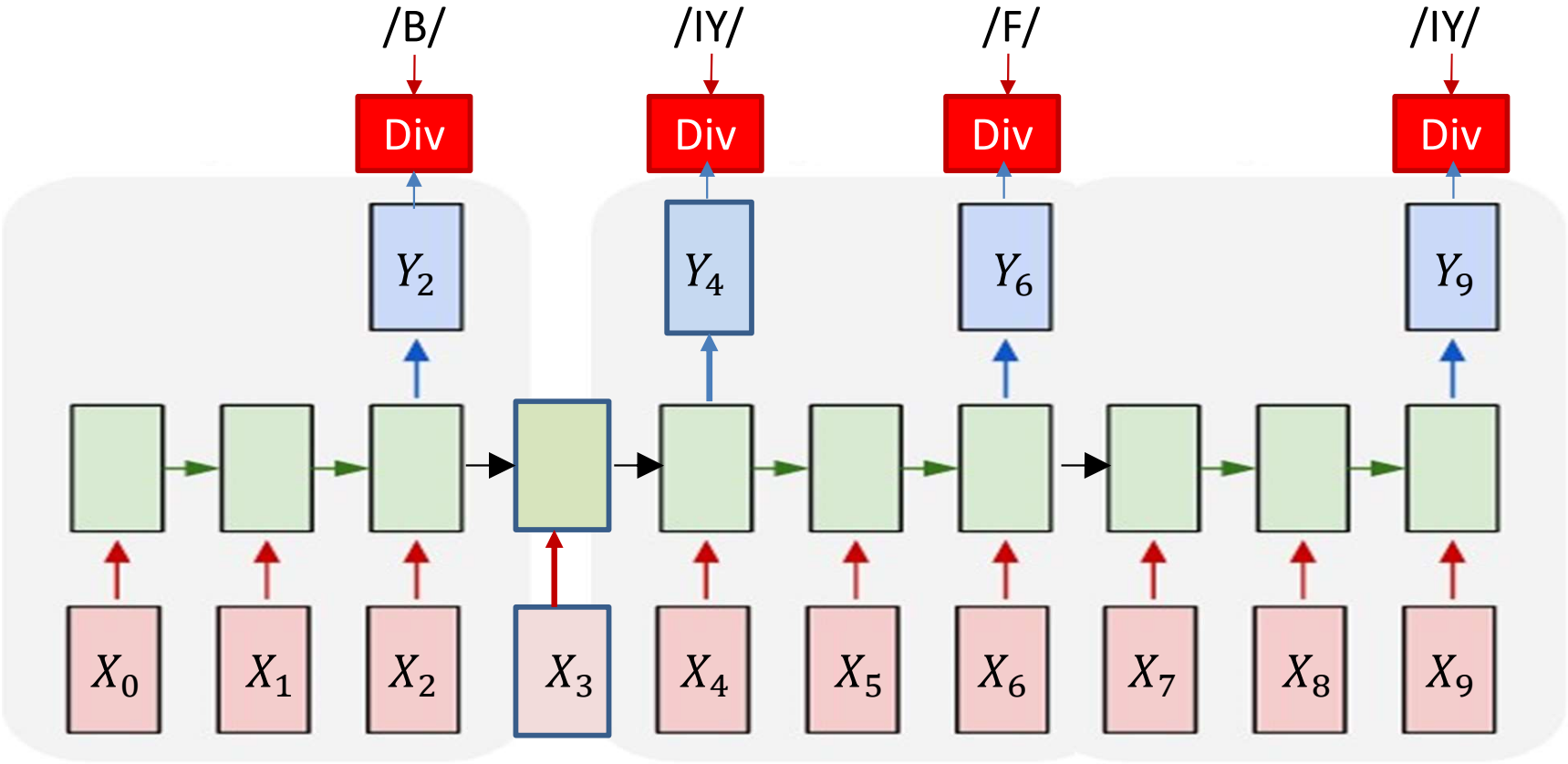
- How do we *train* such models

Problems

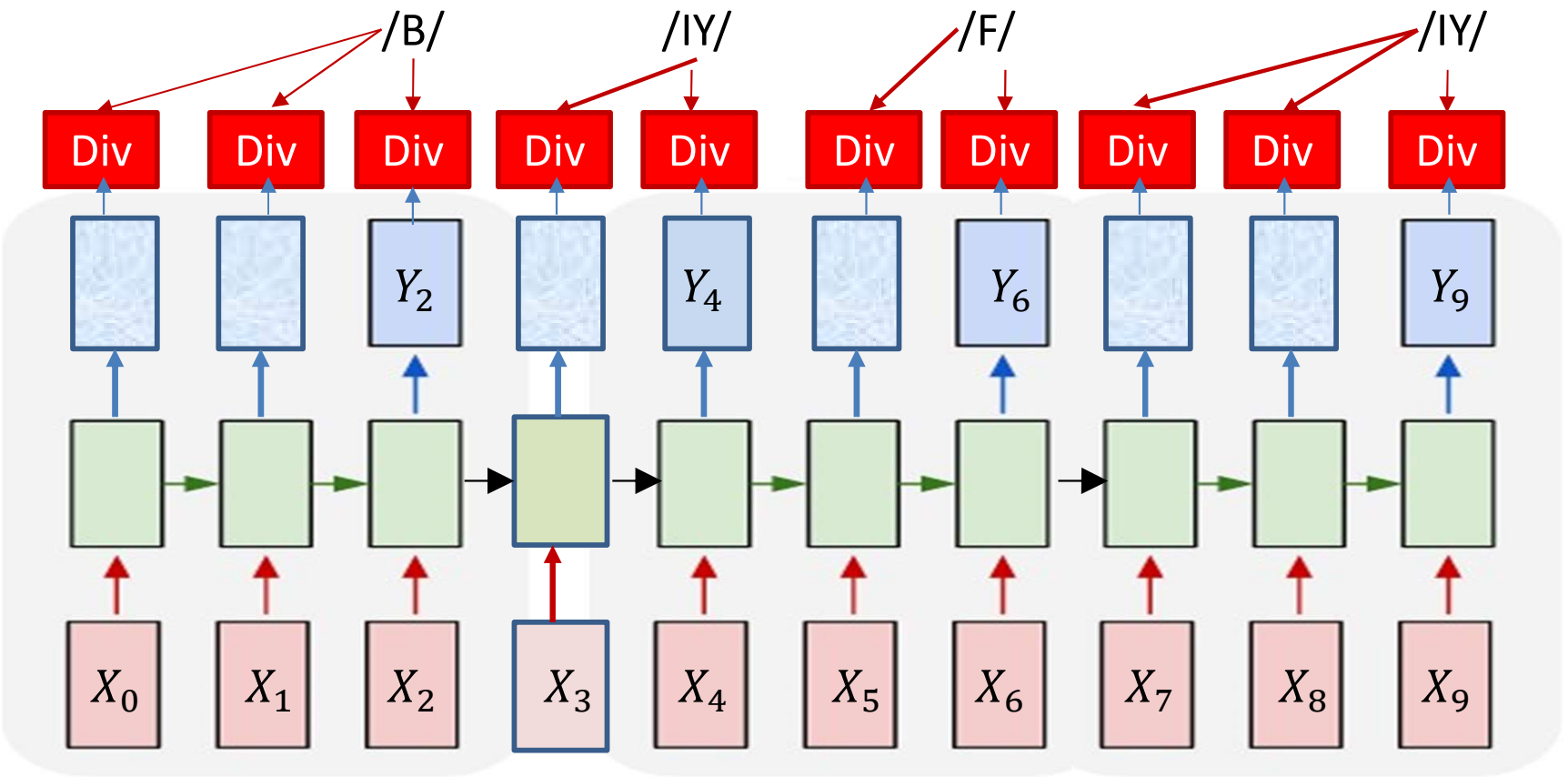
- How do we perform *inference* on such a model
  - How to output time-asynchronous sequences

- How do we *train* such models

Recap: Training with alignment

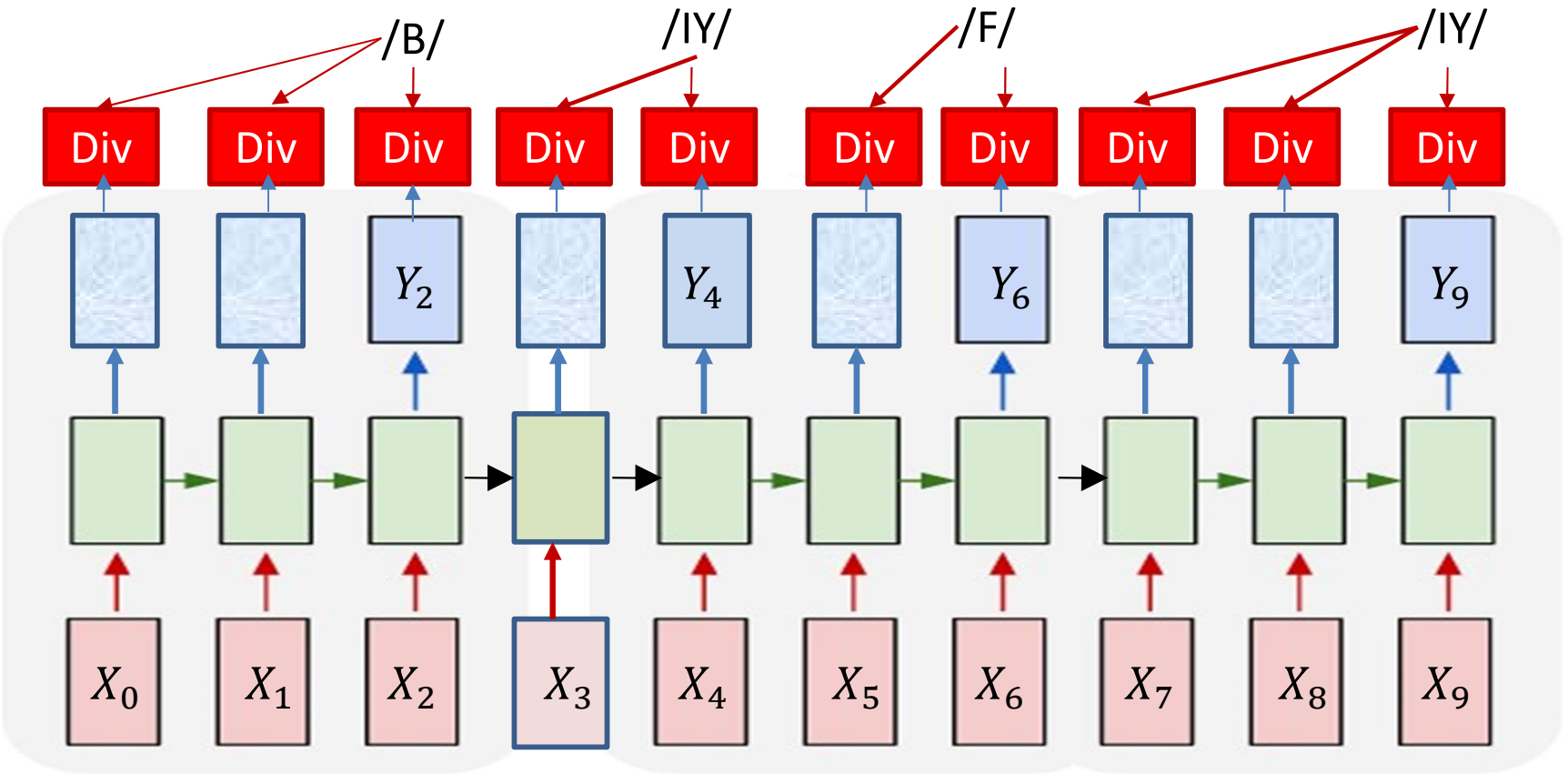


- Given the order-aligned output sequence with timing



- Given the order aligned output sequence with timing
  - Convert it to a time-synchronous alignment by repeating symbols
- Compute the divergence from the time-aligned sequence

$$DIV = \sum_t KL(Y_t, symbol_t) = -\sum_t \log Y(t, symbol_t)$$



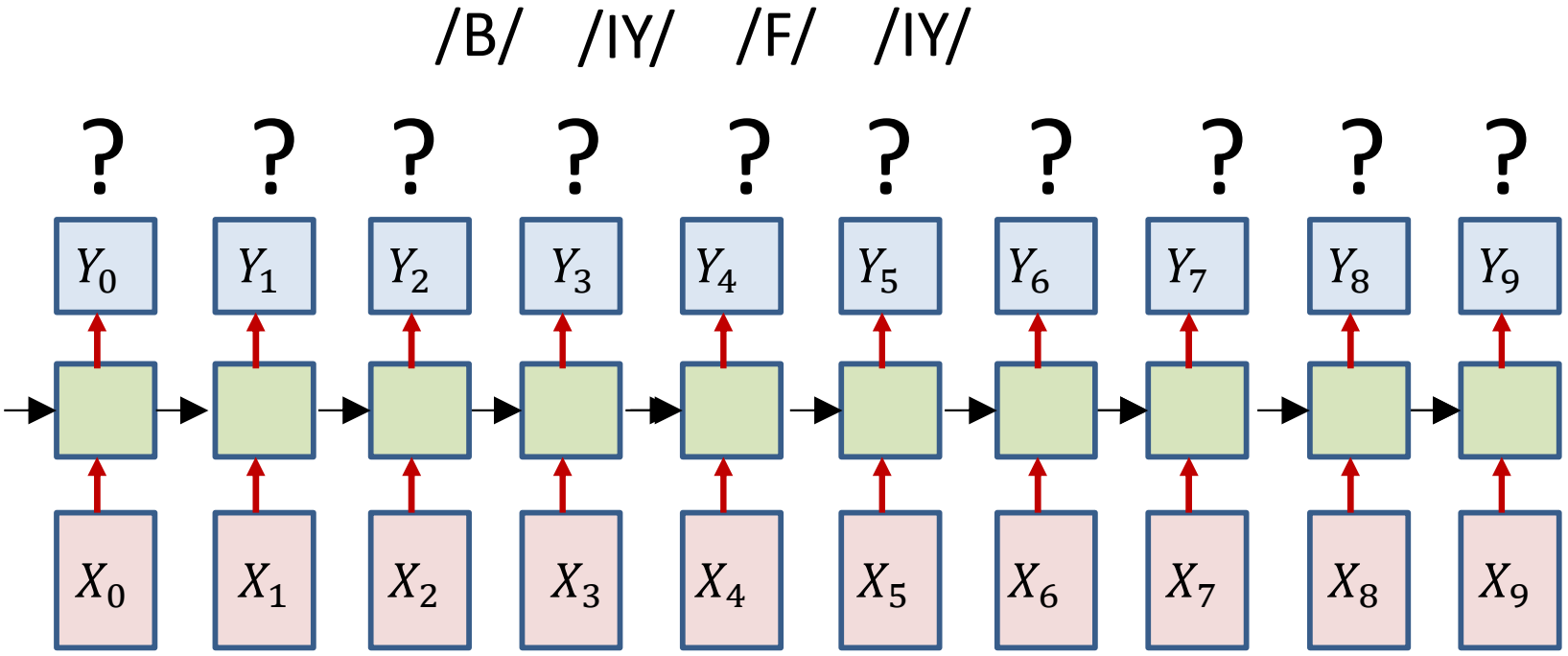
$$DIV = \sum_t KL(Y_t, symbol_t) = -\sum_t \log Y(t, symbol_t)$$

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

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

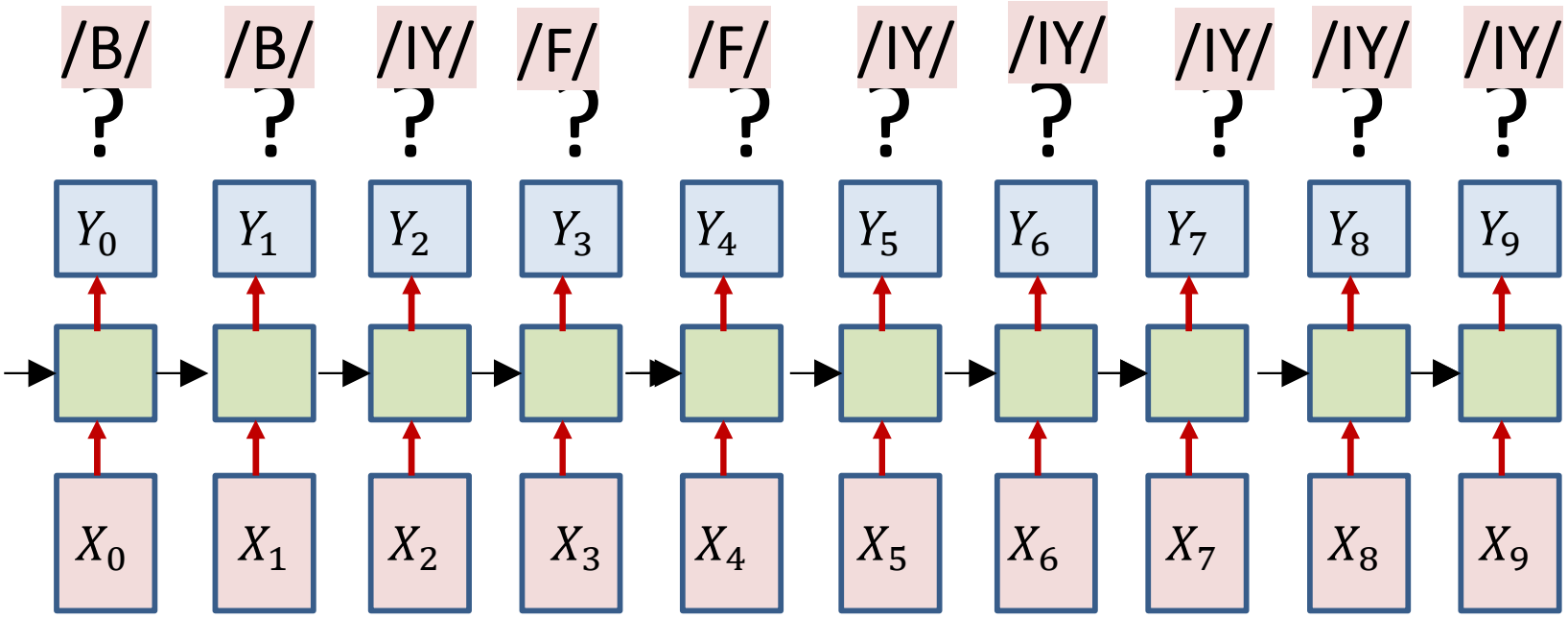
  - Zeros except at the component corresponding to the target aligned to that time

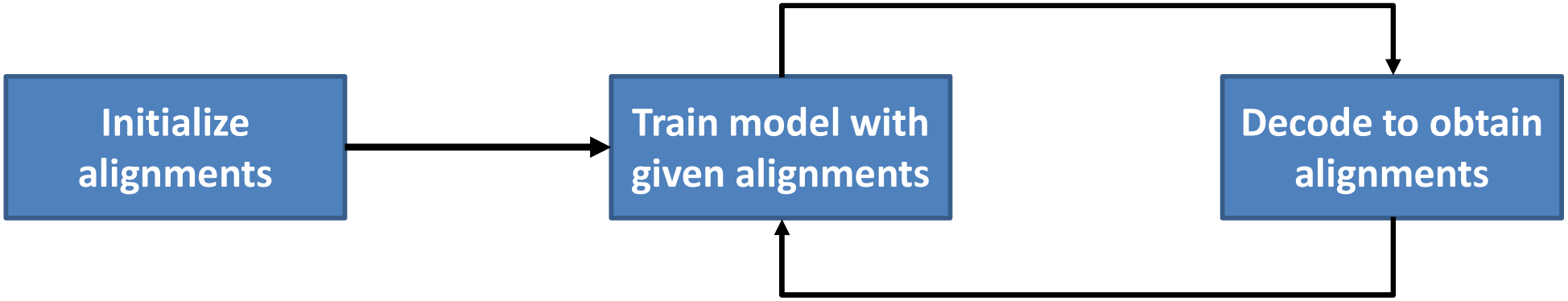
Problem: Alignment not provided



- Only the sequence of output symbols is provided for the training data
  - But no timing information

Solution 1: *Guess the alignment*





Poll 1

- @, @

Poll 1

Viterbi training explicitly estimates the alignment of each training instance and computes the divergence for the estimated alignment (T/F)

- **True**
- False

Viterbi training requires reestimation of alignments in every iteration (T/F)

- **True**
- False

Iterative update: Problem

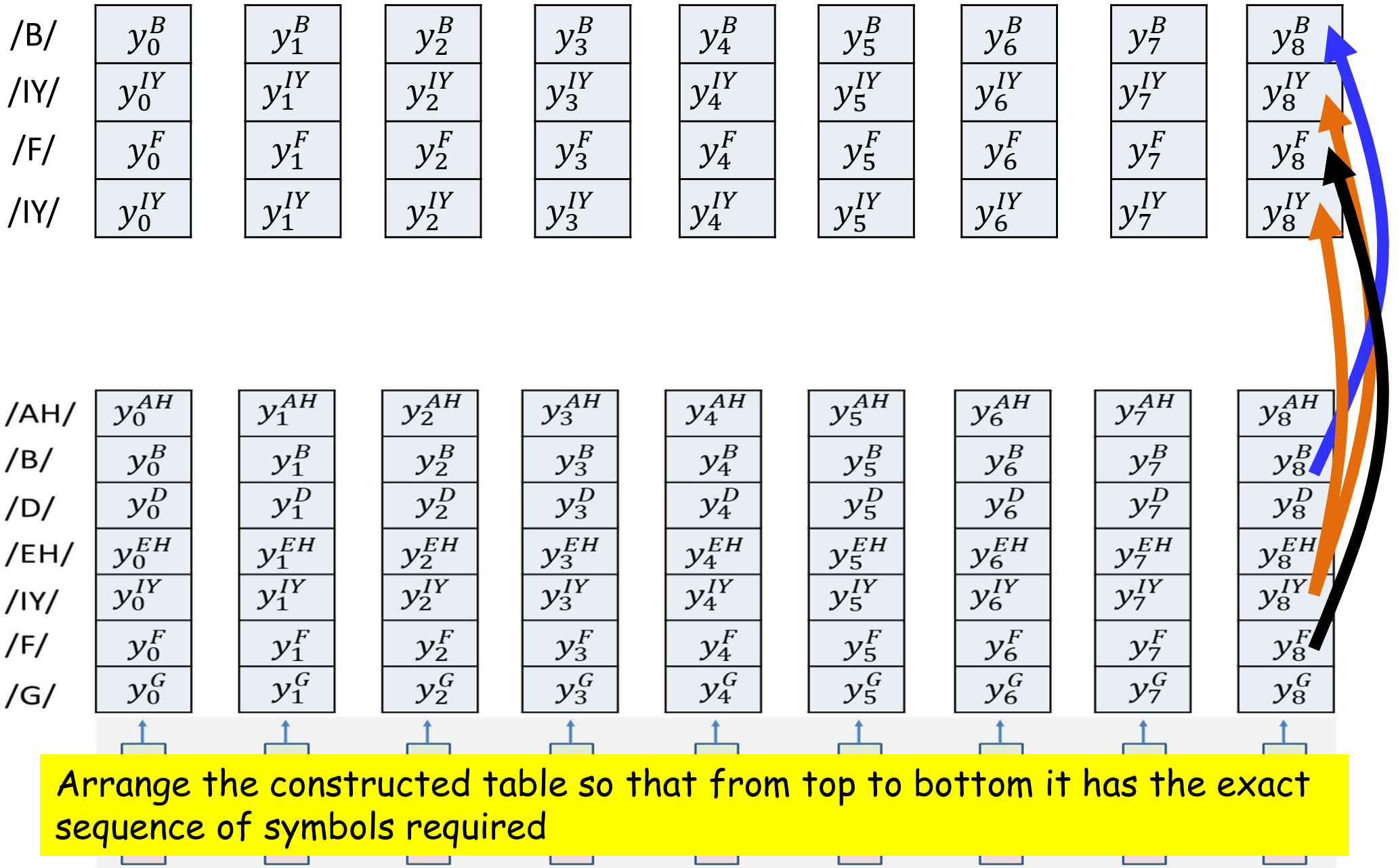
- Approach heavily dependent on initial alignment
- Prone to poor local optima
- Alternate solution: Do not commit to an alignment during any pass..

Recap: Training *without* alignment

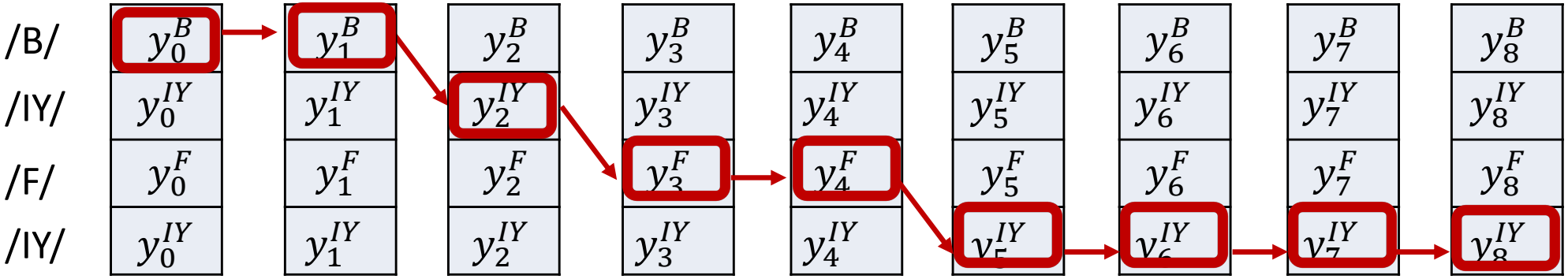
- We know how to train if the alignment is provided
- Problem: Alignment is *not* provided

- Solution:
  1. *Guess* the alignment
  2. Consider *all possible* alignments

Recap: The “aligned” table



The reason for suboptimality

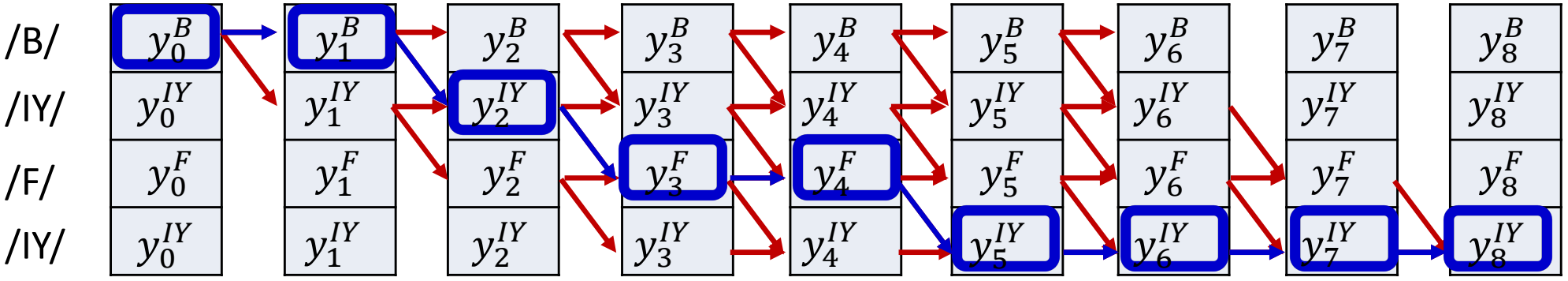


- We *commit* to the single "best" estimated alignment
  - The *most likely* alignment

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

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

The reason for suboptimality



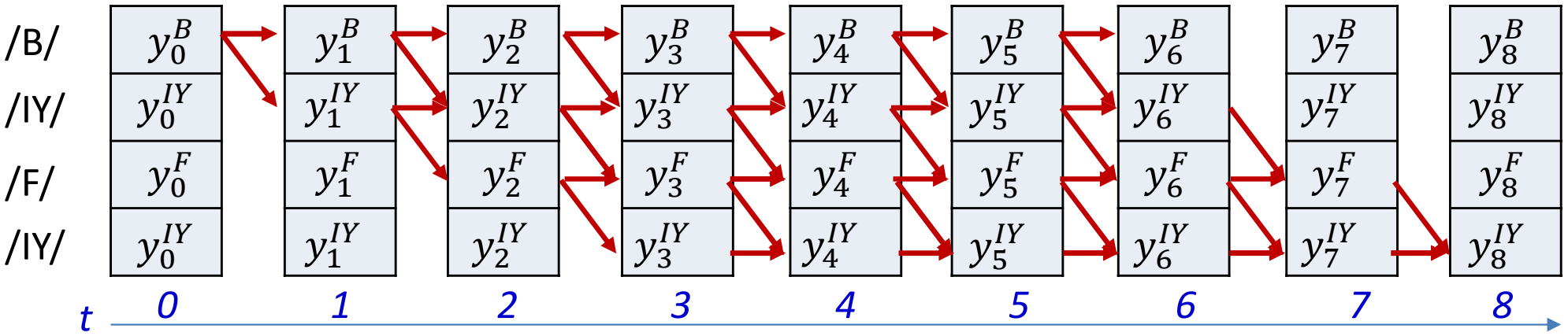
- We *commit* to the single "best" estimated alignment
  - The *most likely* alignment

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

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

- **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

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

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

Poll 2

- @, @

Poll 2

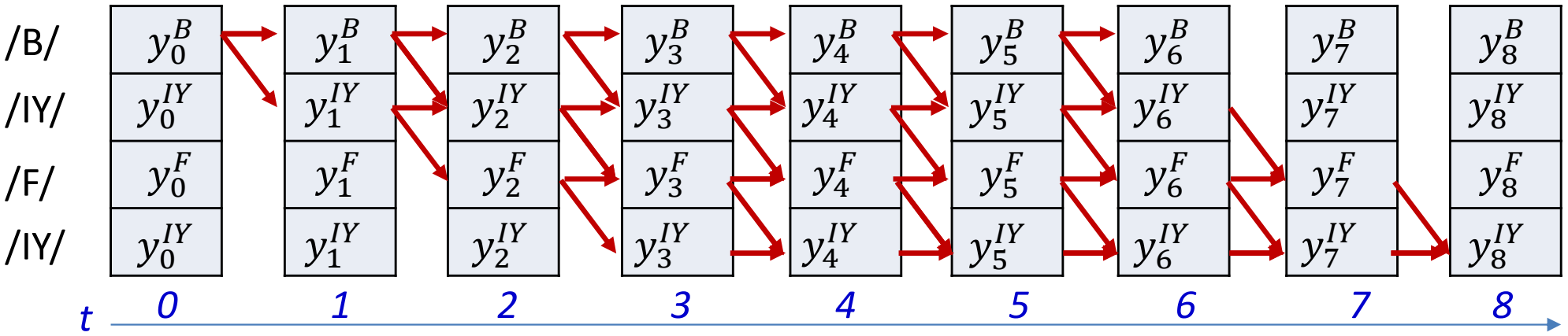
The "training-without-alignment" procedure computes the average divergence over *all possible* alignments of the label sequence to the input (T/F)

- **True**
- False

The "training-without-alignment" requires explicit estimation of the alignment of the label sequence to the input

- True
- **False**

The expectation over *all* alignments



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

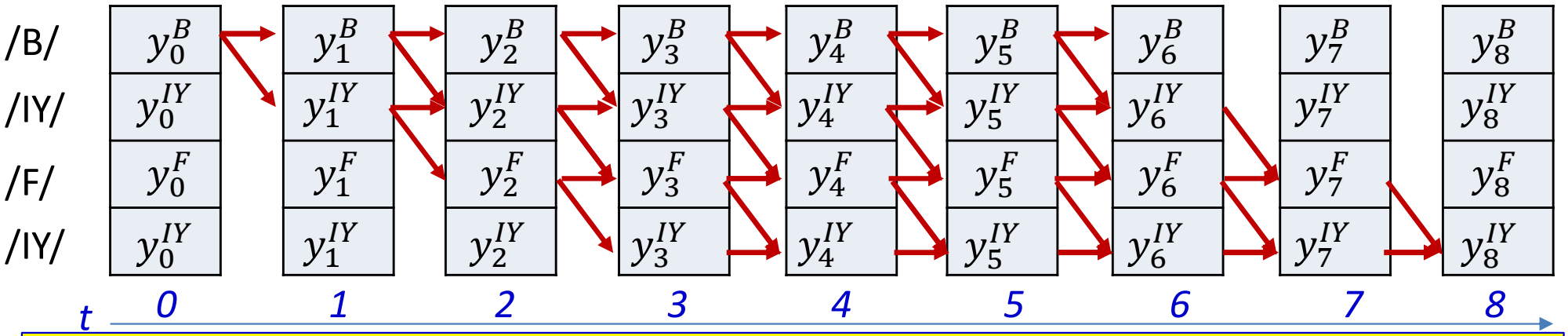
- Using the linearity of expectation

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

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

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

The expectation over *all* alignments



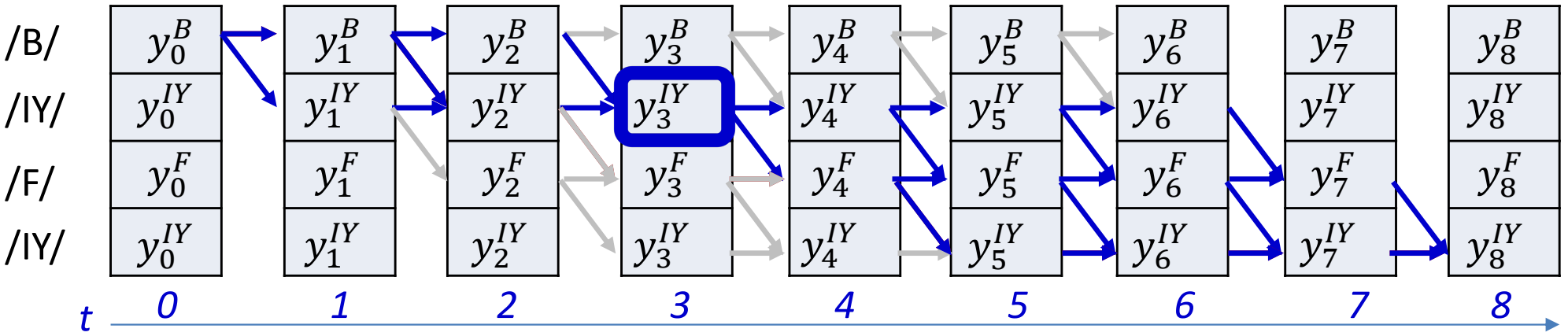
- The probability of aligning the specific symbol s at time t, *given* that unaligned sequence $\mathbf{S} = S_0 \ldots S_{K-1}$ *and given* the input sequence $\mathbf{X} = X_0 \ldots X_{N-1}$
- We need to be able to compute this

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

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

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

A posteriori probabilities of symbols

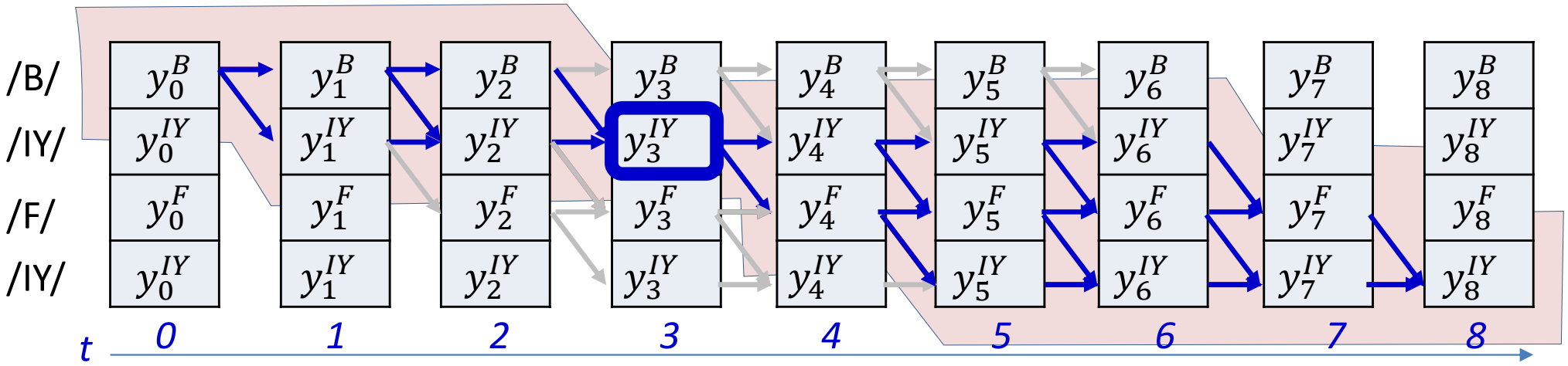


$$P(s_t = S_r|\mathbf{S}, \mathbf{X}) \propto P(s_t = S_r, \mathbf{S}|\mathbf{X})$$

- $P(s_t = S_r, \mathbf{S}|\mathbf{X})$ 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 $S_0 \ldots S_{K-1}$) at time t

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

A posteriori probabilities of symbols

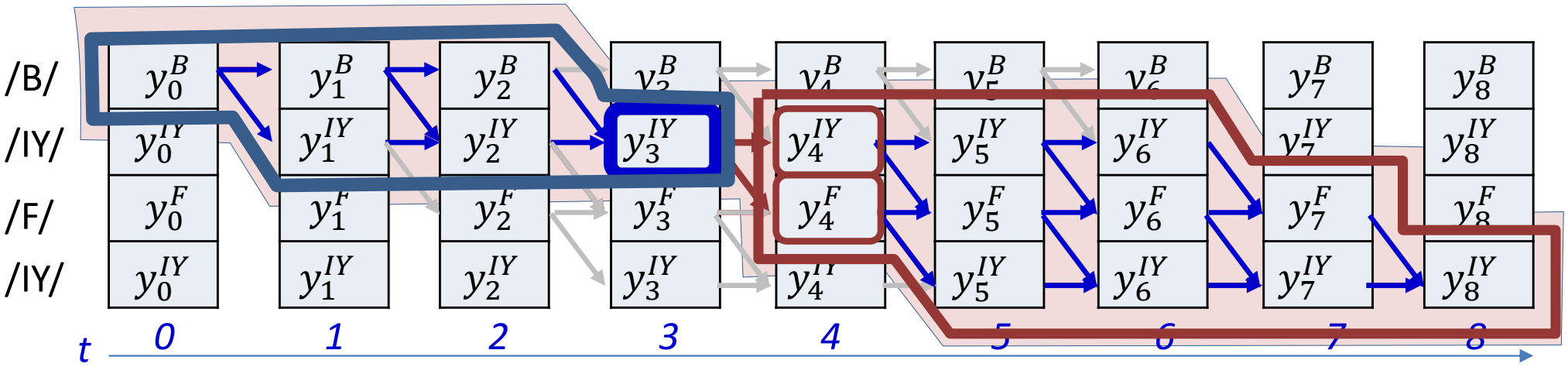


$$P(s_t = S_r|\mathbf{S},\mathbf{X}) \propto P(s_t = S_r,\mathbf{S}|\mathbf{X})$$

- $P(s_t = S_r,\mathbf{S}|\mathbf{X})$ 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 $S_0 \ldots S_{K-1}$) 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

$$P(s_t = S_r, \mathbf{S}|\mathbf{X}) = P(S_0, \ldots, S_r, \ldots, S_{K-1}, s_t = S_r|\mathbf{X})$$

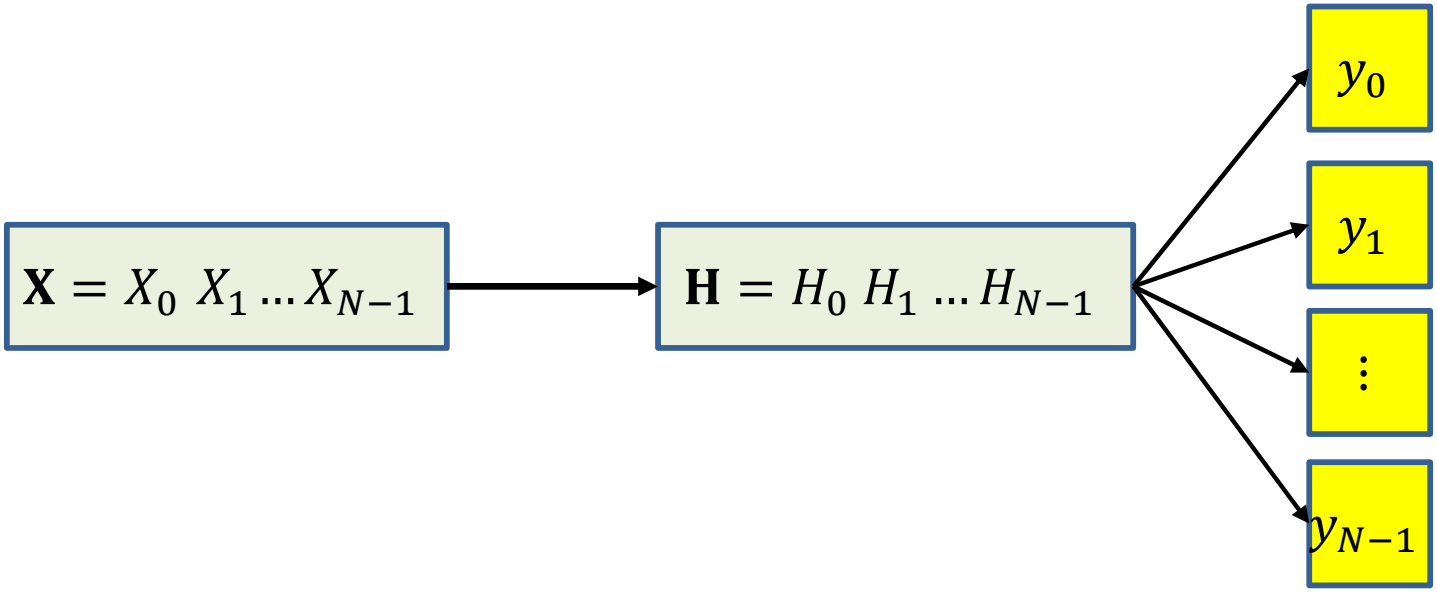
$$= P(S_0 \ldots S_r, s_t = S_r, s_{t+1} \in succ(S_r), succ(S_r), \ldots, S_{K-1}|\mathbf{X})$$

- Using Bayes Rule

$$= P(S_0 \ldots S_r, s_t = S_r|\mathbf{X}) P(s_{t+1} \in succ(S_r), succ(S_r), \ldots, S_{K-1}|S_0 \ldots S_r, s_t = S_r\mathbf{X})$$

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

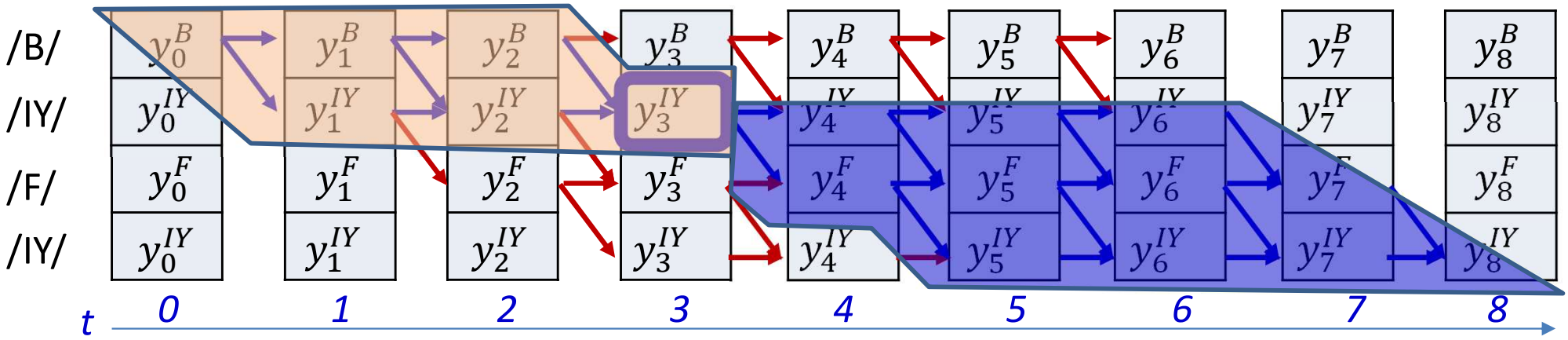
Conditional independence



- ***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 the ys at any time went back into the net as inputs

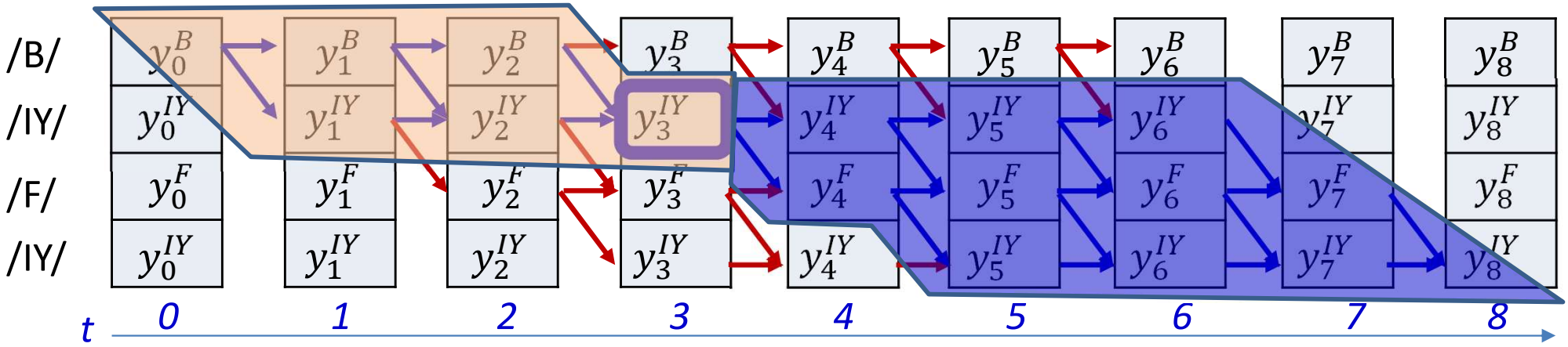
A posteriori symbol probability



$$P(s_t = S_r, \mathbf{S}|\mathbf{X})$$
$$= P(S_0 \ldots S_r, s_t = S_r \,|\mathbf{X}) P(s_{t+1} \in succ(S_r), succ(succ(S_r)), \ldots, S_{K-1} \,|\mathbf{X})$$

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

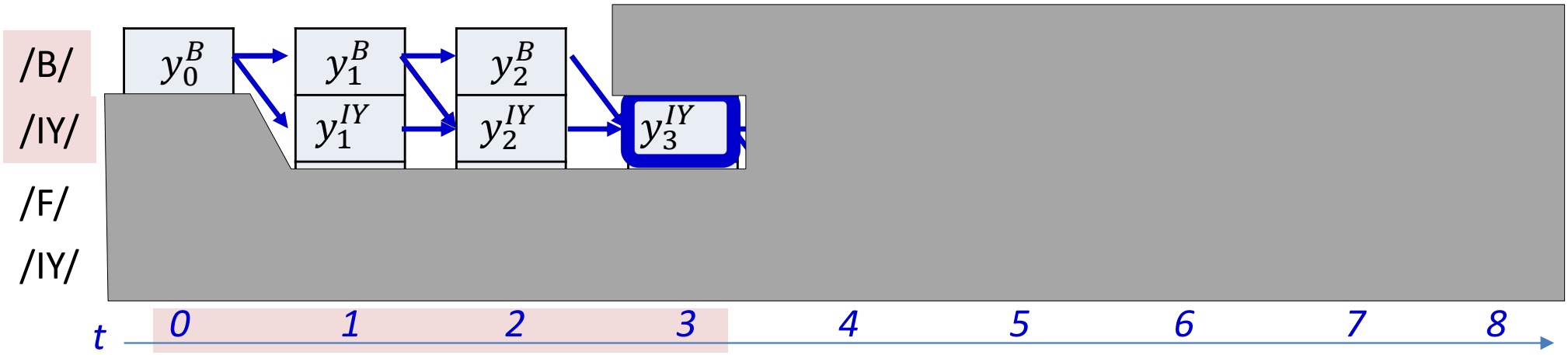
A posteriori symbol probability



$$P(s_t = S_r, \mathbf{S}|\mathbf{X})$$
$$= P(S_0 \ldots S_r, s_t = S_r \,|\mathbf{X}) P(s_{t+1} \in succ(S_r), succ(S_r), \ldots, S_{K-1} \,|\mathbf{X})$$

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

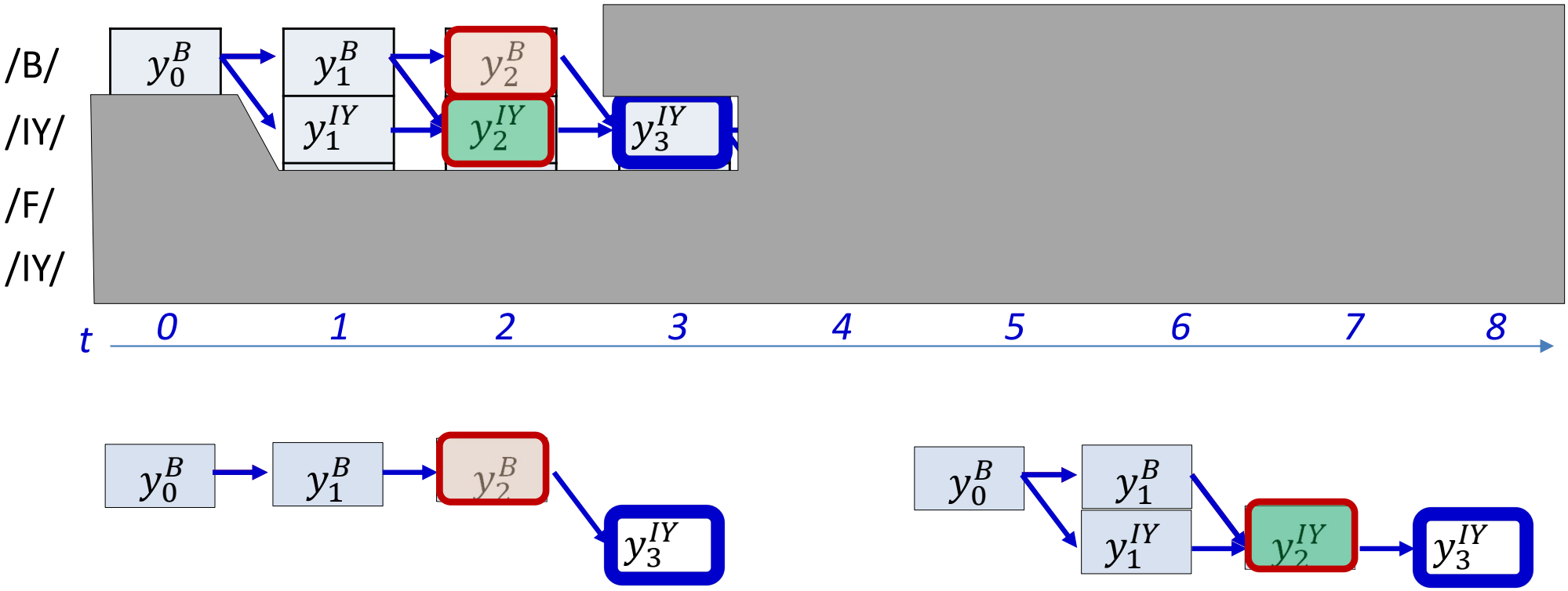
Computing $\alpha(t,r)$: Forward algorithm



$$\alpha(t,r) = P(S_0..S_r, s_t = S_r|\mathbf{X})$$

- The $\alpha(t,r)$ is the total probability of the subgraph shown
  - The total probability of all paths leading to the alignment of S_r to time t

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



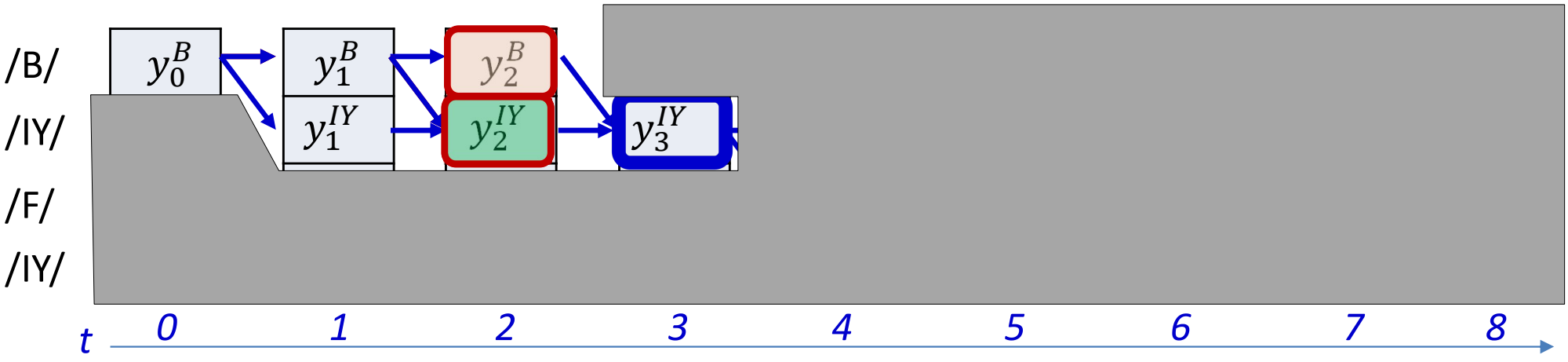
$$\alpha(3, IY) = P(S_0..S_r, s_t = S_r|\mathbf{X})$$

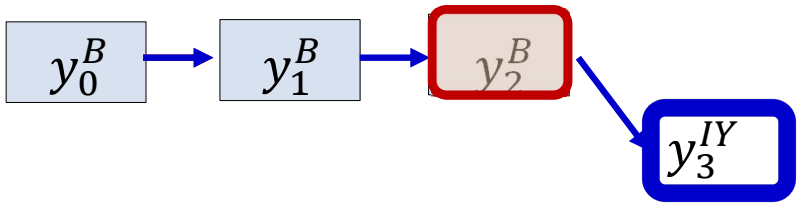
$$\alpha(3, IY) = P(subgraph\ ending\ at\ (2,B))y_3^{IY} + P(subgraph\ ending\ at\ (2,IY))y_3^{IY}$$

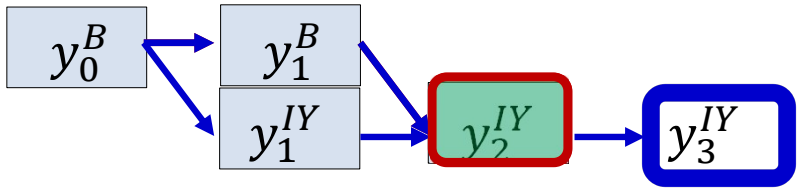
$$\alpha(t,r) = \sum_{q:S_q \in pred(S_r)} P(subgraph\ ending\ at\ (t-1,q))Y_t^{S(r)}$$

- 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







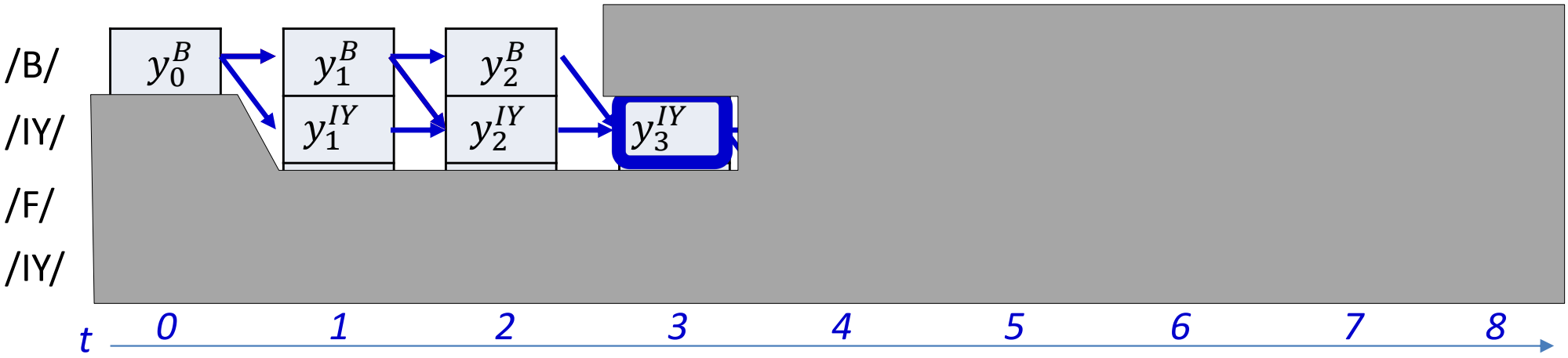
$$\alpha(t,r) = P(S_0..S_r, s_t = S_r|\mathbf{X})$$

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

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

- 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

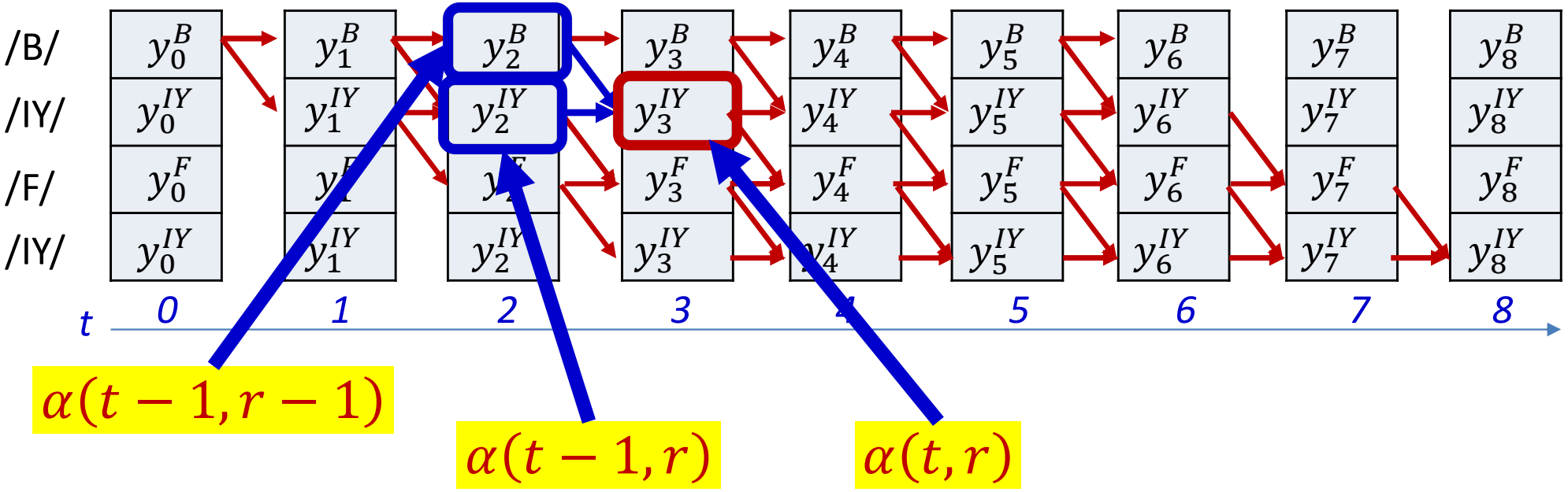
Forward algorithm



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

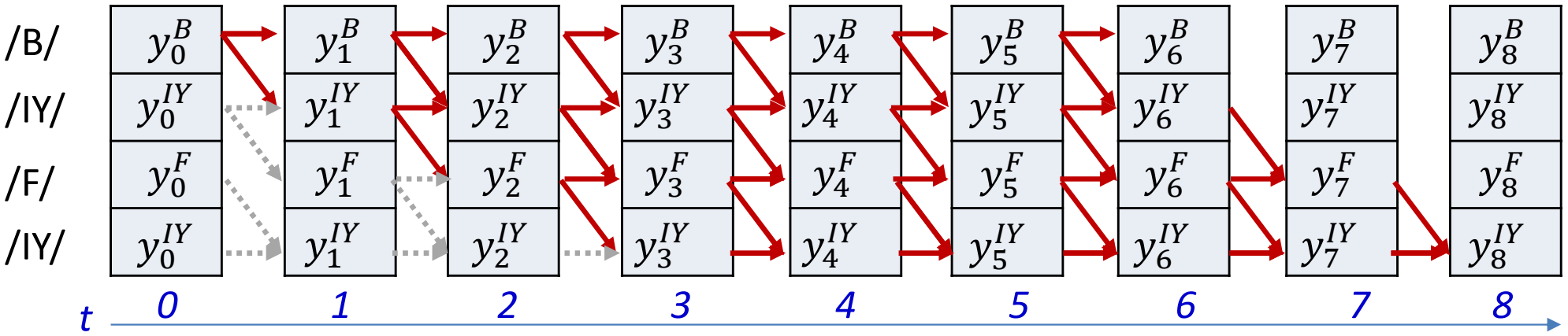
- The $\alpha(t,r)$ is the total probability of the subgraph shown

Forward algorithm



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

Forward algorithm



- Initialization:

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

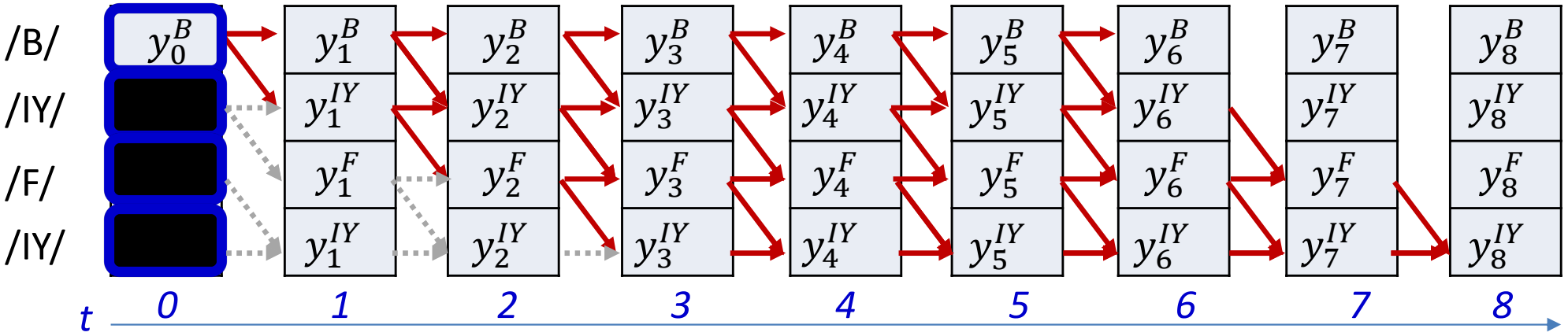
- for $t = 1 \ldots T-1$

$$\alpha(t,0) = \alpha(t-1,0) y_t^{S(0)}$$

for $l = 1 \ldots K-1$

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

Forward algorithm



- Initialization:

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

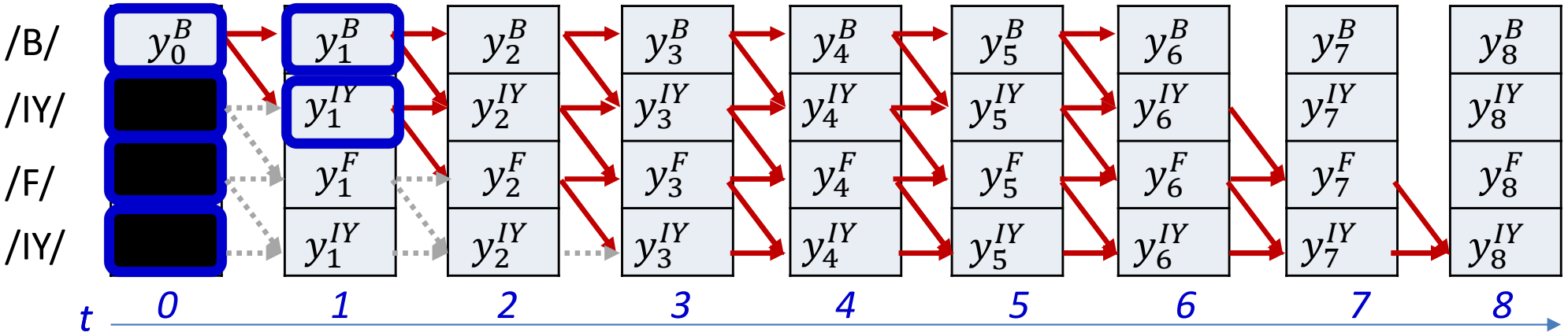
- for $t = 1 \ldots T-1$

$$\alpha(t,0) = \alpha(t-1,0) y_t^{S(0)}$$

for $l = 1 \ldots K-1$

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

Forward algorithm



- Initialization:

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

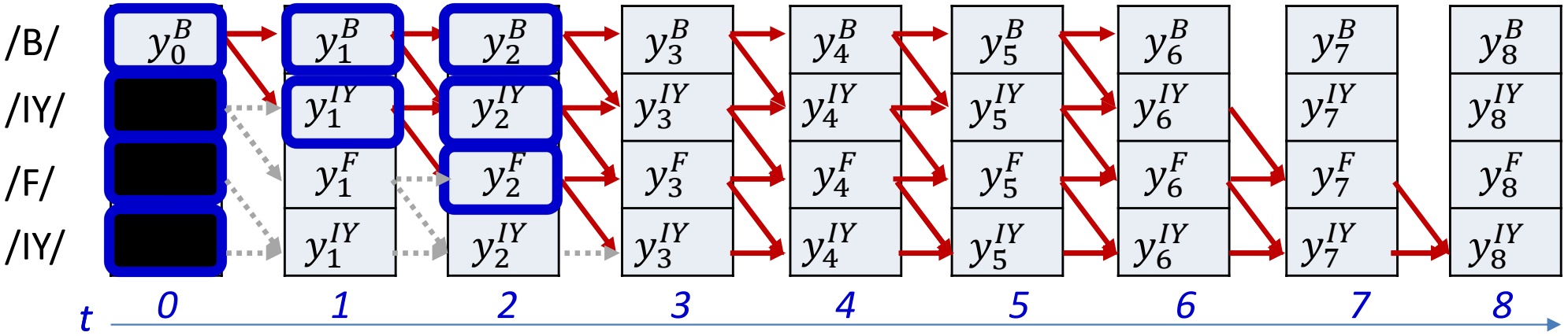
- for $t = 1 \ldots T-1$

$$\alpha(t,0) = \alpha(t-1,0) y_t^{S(0)}$$

for $l = 1 \ldots K-1$

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

Forward algorithm



- Initialization:

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

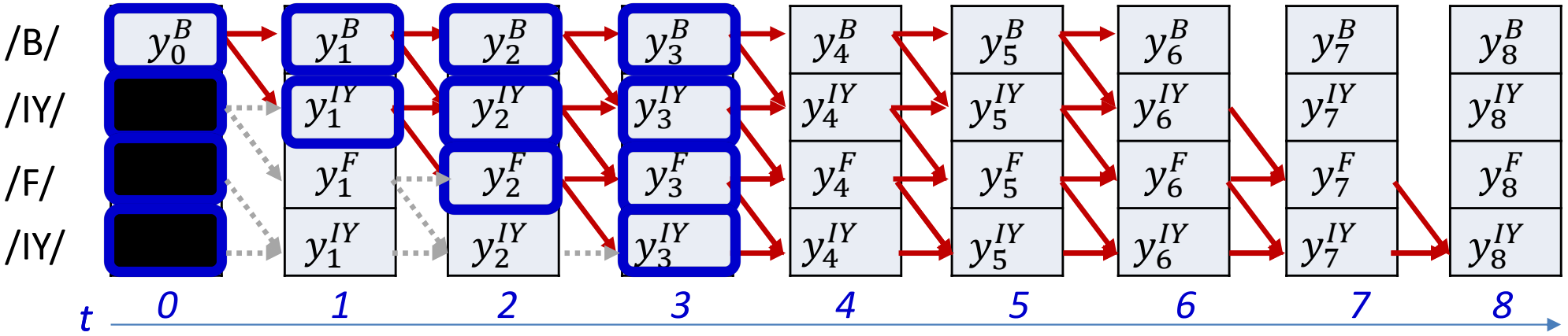
- for $t = 1 \ldots T-1$

$$\alpha(t,0) = \alpha(t-1,0) y_t^{S(0)}$$

for $l = 1 \ldots K-1$

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

Forward algorithm



- Initialization:

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

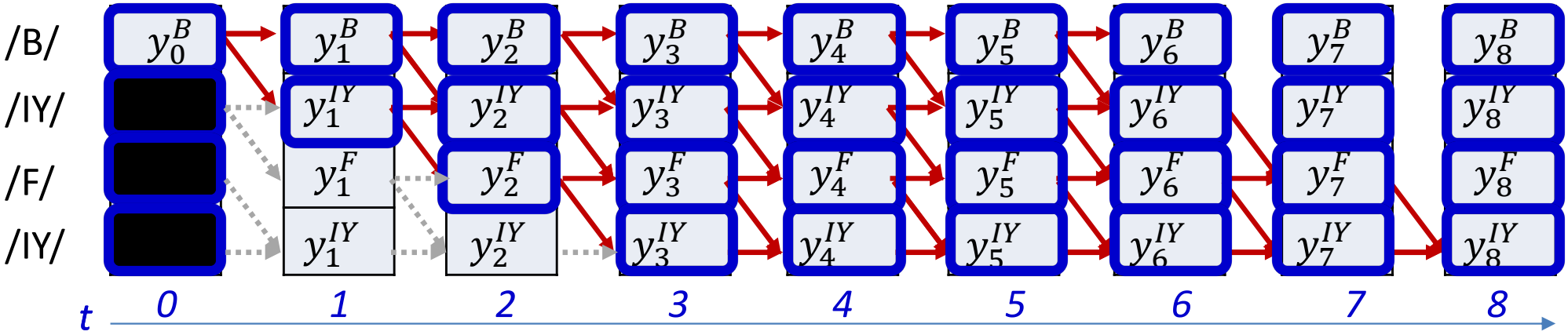
- for $t = 1 \ldots T-1$

$$\alpha(t,0) = \alpha(t-1,0) y_t^{S(0)}$$

for $l = 1 \ldots K-1$

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

Forward algorithm



- Initialization:

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

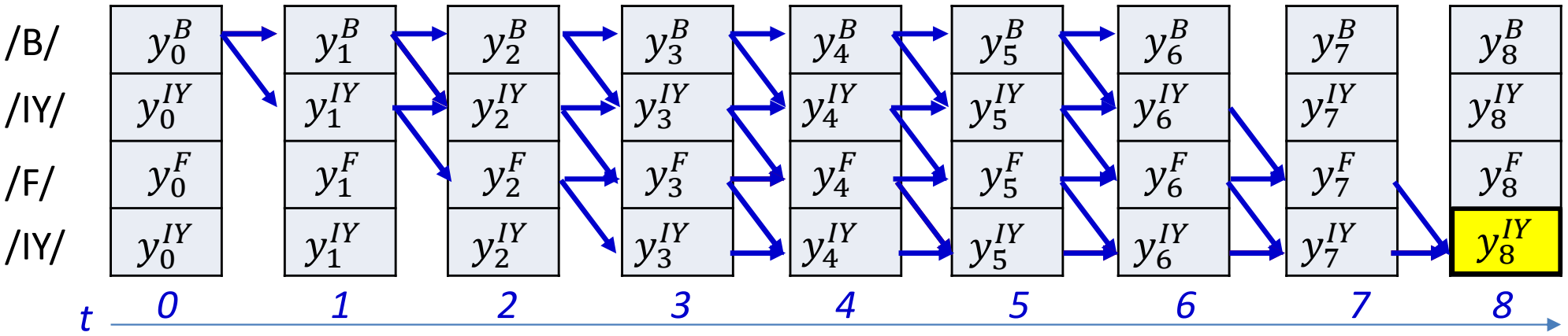
- for $t = 1 \ldots T-1$

$$\alpha(t,0) = \alpha(t-1,0) y_t^{S(0)}$$

for $l = 1 \ldots K-1$

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

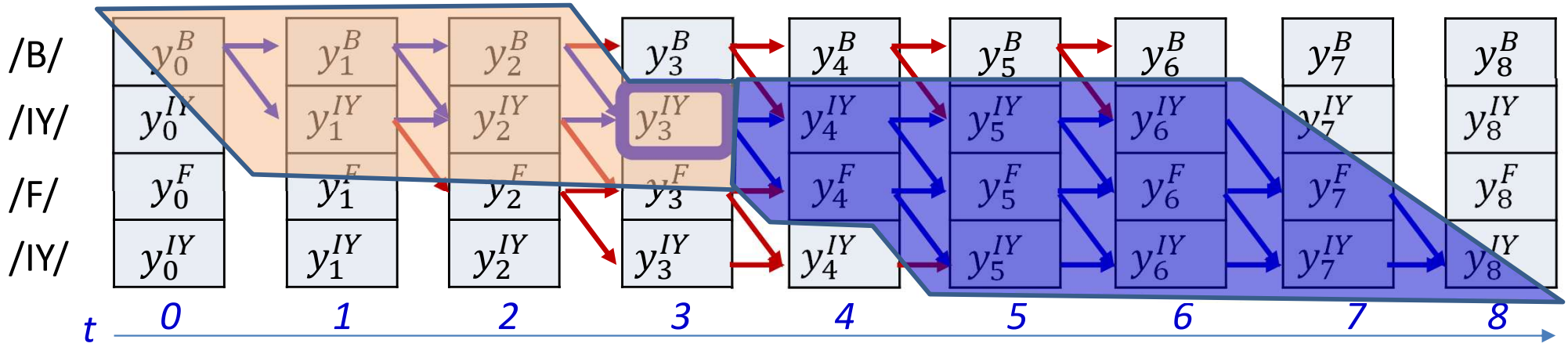
The final forward probability $\alpha(t, r)$



$$\alpha(T-1, K-1) = P(S_0..S_{K-1}|\mathbf{X})$$

- The probability of the entire symbol sequence is the alpha at the bottom right node

A posteriori symbol probability

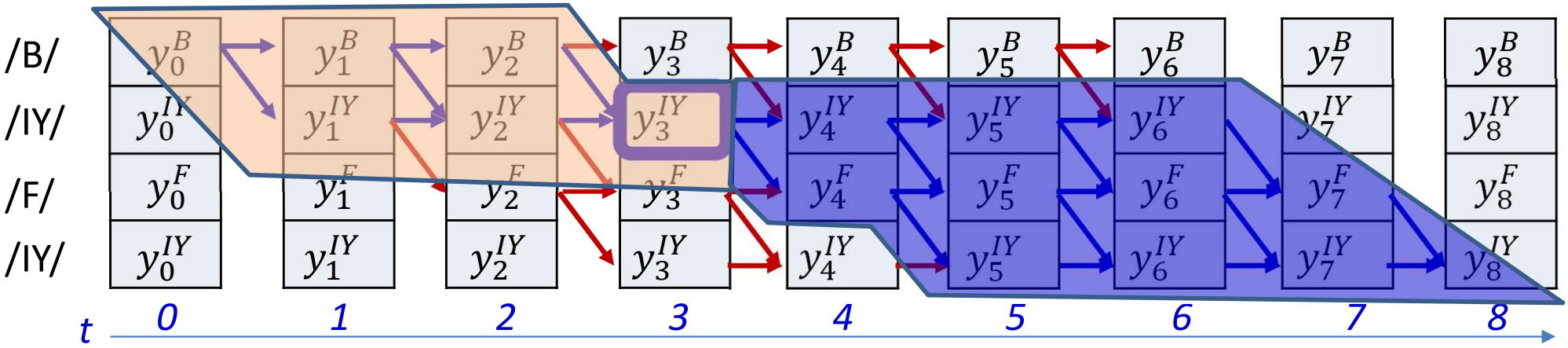


$$P(s_t = S_r, \mathbf{S}|\mathbf{X})$$
$$= P(S_0 \ldots S_r, s_t = S_r \,|\mathbf{X})\, P(s_{t+1} \in succ(S_r), succ(S_r), \ldots, S_{K-1} \,|\mathbf{X})$$

- 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

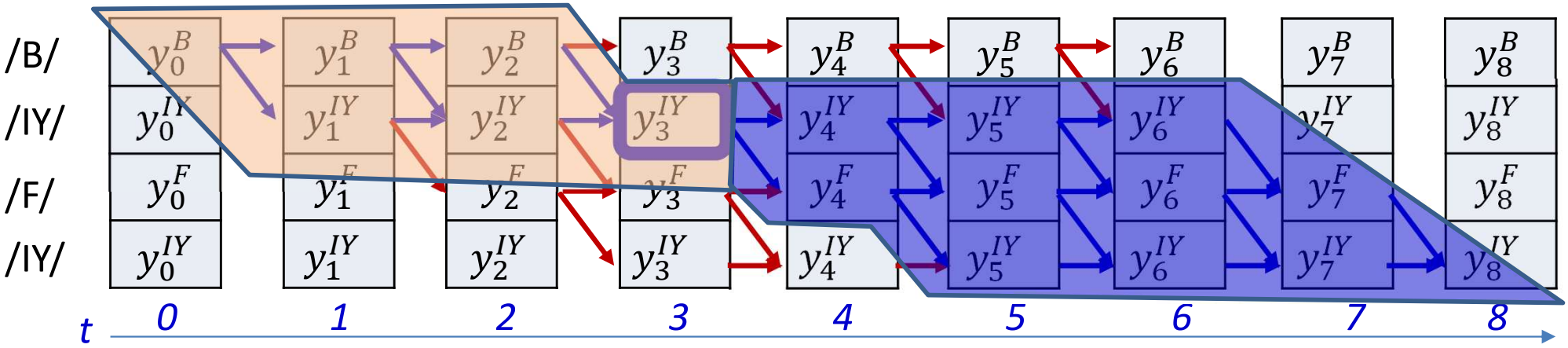


$$P(s_t = S_r, \mathbf{S}|\mathbf{X}) = \alpha(t,r) P(s_{t+1} \in succ(S_r), succ(S_r), \ldots, S_{K-1} |\mathbf{X})$$

- 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

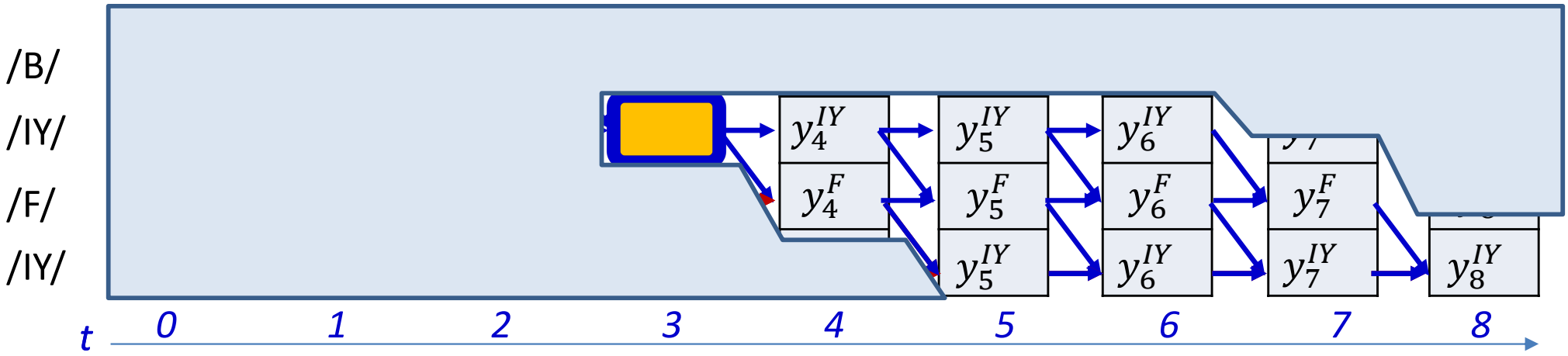


$$P(s_t = S_r, \mathbf{S}|\mathbf{X}) = \alpha(t,r) P(s_{t+1} \in succ(S_r), succ(S_r), \ldots, S_{K-1} |\mathbf{X})$$

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

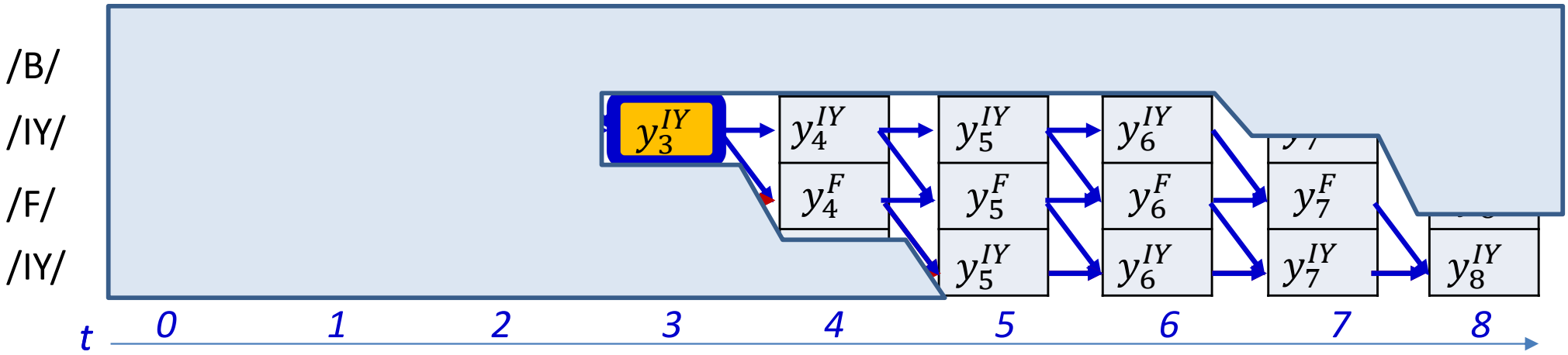
Lets look at this

Backward probability



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

Backward probability



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

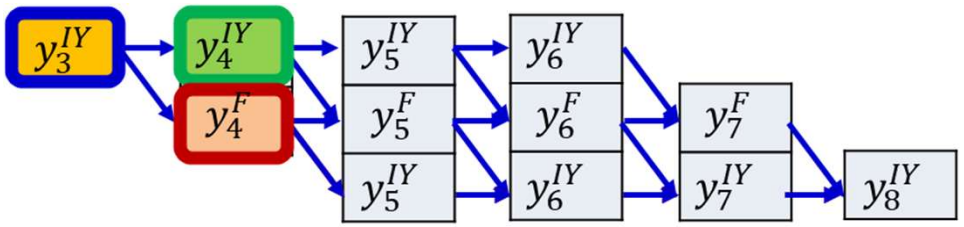
- For convenience, let us include the box in the graph, and factor it out later

  $\hat{\beta}(t,r)$ = probability of graph including node at (t,r)

$$\beta(t,r) = \frac{1}{y_t^{S_r}} \hat{\beta}(t,r)$$

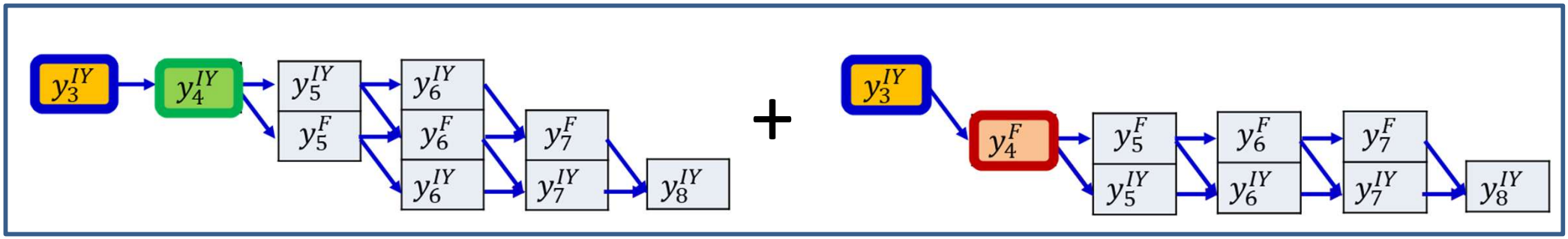
- We will develop an algorithm to compute $\hat{\beta}(t,r)$ and compute $\beta(t,r)$ from it by dividing out $y_t^{S_r}$ later

Backward probability



=

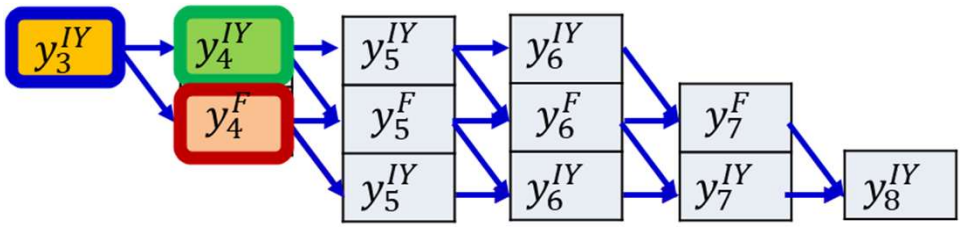
B IY D IY



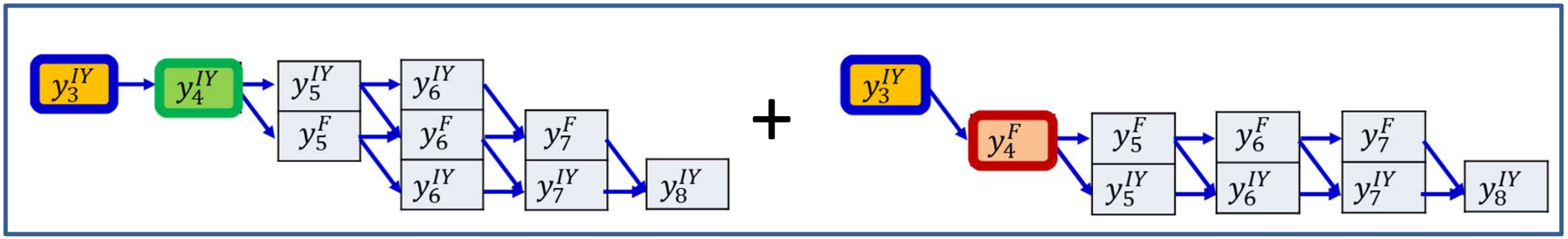
- Using the same logic as in the forward algorithm:

$$\hat{\beta}(3, IY)$$
$$= y_3^{IY} P(subgraph\ starting\ at\ (4, IY)) + y_3^{IY} P(subgraph\ ending\ at\ (4, F))$$

Backward probability



=



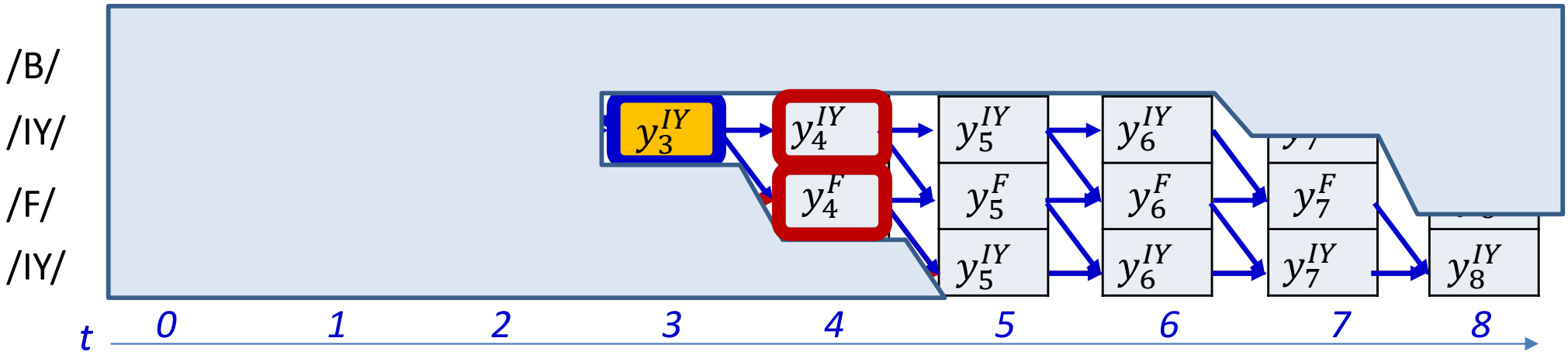
- Using the same logic as in the forward algorithm:

$$\hat{\beta}(3, IY) = y_3^{IY} P(\textit{subgraph starting at } (4, IY)) + y_3^{IY} P(\textit{subgraph ending at } (4, F))$$

- We recognize these terms:

$$\hat{\beta}(3, IY) = y_3^{IY} \left(\hat{\beta}(3, IY) + \hat{\beta}(3, F) \right)$$

Backward algorithm

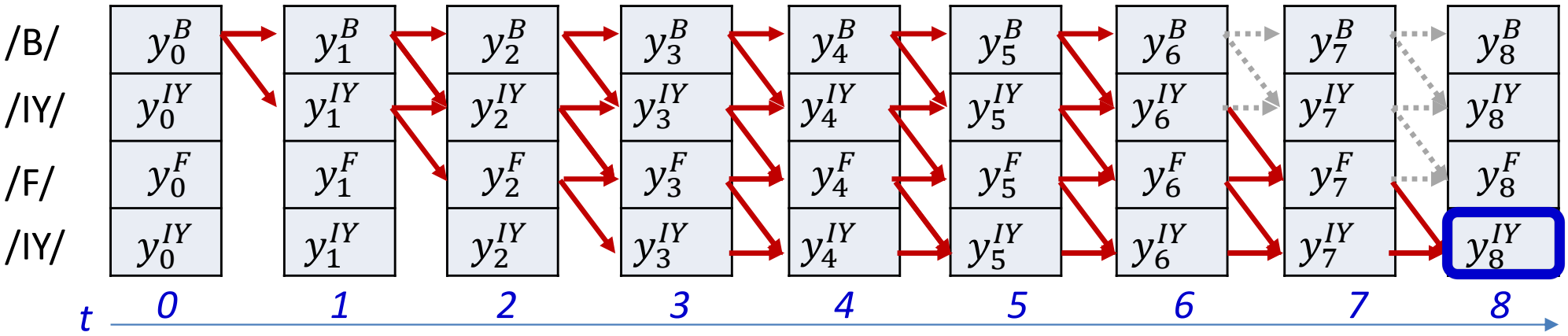


$$\hat{\beta}(t,r) = y_t^{S_r} \sum_{q \in succ(r)} \hat{\beta}(t+1,q)$$

- The $\hat{\beta}(t,r)$ is the total probability of the subgraph shown
  - *Including* the node at (t,r)

- The $\hat{\beta}(t,r)$ terms at any time t are defined recursively in terms of the $\hat{\beta}(t+1,q)$ terms at the next time

Backward algorithm



- Entire backward algorithm:
  - Note : some nodes (bottom row) have more successors than others

- Initialization:

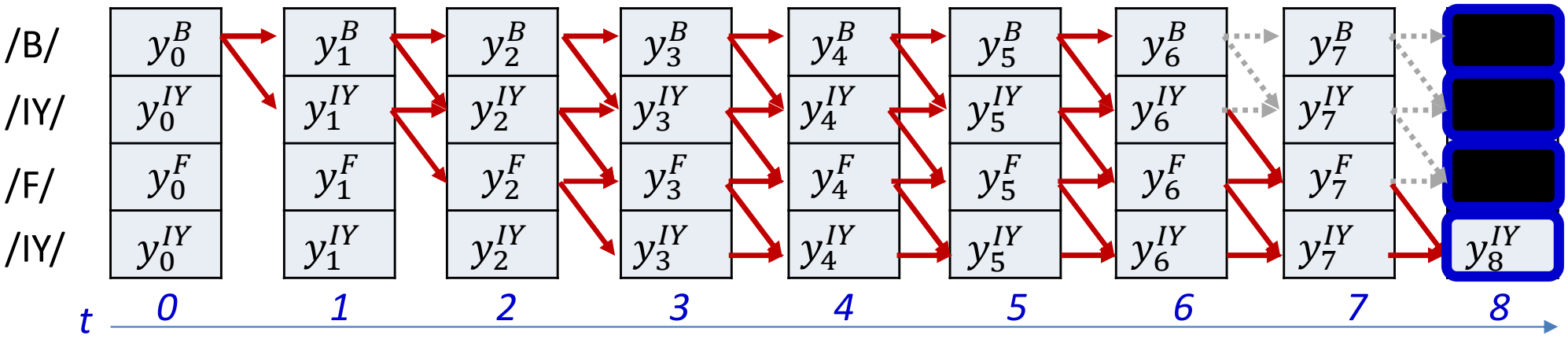
$$\hat{\beta}(T-1, K-1) = y_{T-1}^{S(K-1)}, \quad \hat{\beta}(T-1, r) = 0, \; r < K-1$$

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

  for $r = K-1 \ldots 0$

$$\hat{\beta}(t, r) = y_t^{S(r)} \sum_{q \in succ(r)} \hat{\beta}(t+1, q)$$

Backward algorithm



- Initialization:

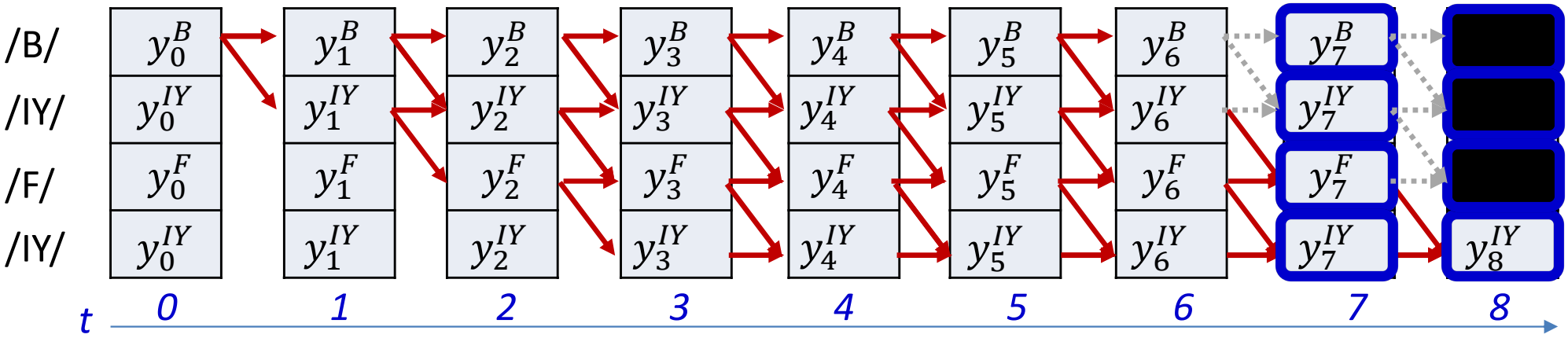
$$\hat{\beta}(T-1, K-1) = y_{T-1}^{S(K-1)}, \;\; \hat{\beta}(T-1, r) = 0, \;\; r < K-1$$

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

  for $r = K-1 \ldots 0$

$$\hat{\beta}(t, r) = y_t^{S(r)} \sum_{q \in succ(r)} \hat{\beta}(t+1, q)$$

Backward algorithm



- Initialization:

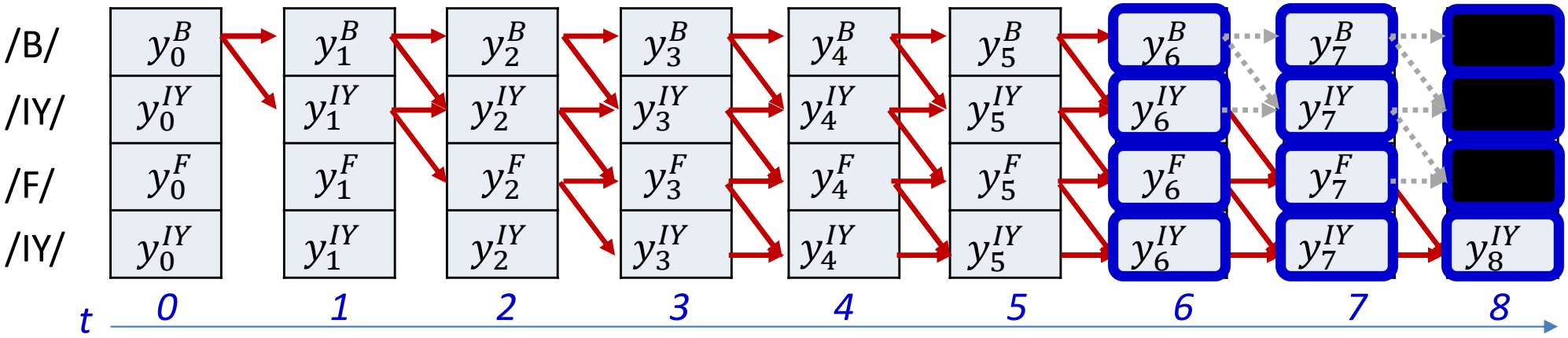
$$\hat{\beta}(T-1, K-1) = y_{t+1}^{S(K-1)}, \hat{\beta}(T-1, r) = 0, \; r < K-1$$

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

  for $r = K-1 \ldots 0$

$$\hat{\beta}(t, r) = y_t^{S(r)} \sum_{q \in succ(r)} \hat{\beta}(t+1, q)$$

Backward algorithm



- Initialization:

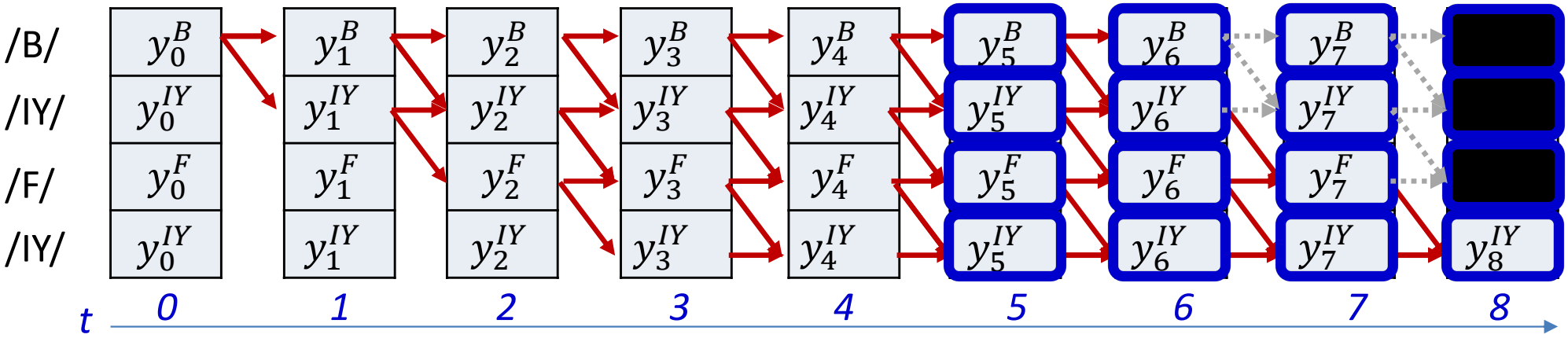
$$\hat{\beta}(T-1, K-1) = y_{t+1}^{S(K-1)}, \hat{\beta}(T-1, r) = 0, \; r < K-1$$

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

  for $r = K-1 \ldots 0$

$$\hat{\beta}(t, r) = y_t^{S(r)} \sum_{q \in succ(r)} \hat{\beta}(t+1, q)$$

Backward algorithm



- Initialization:

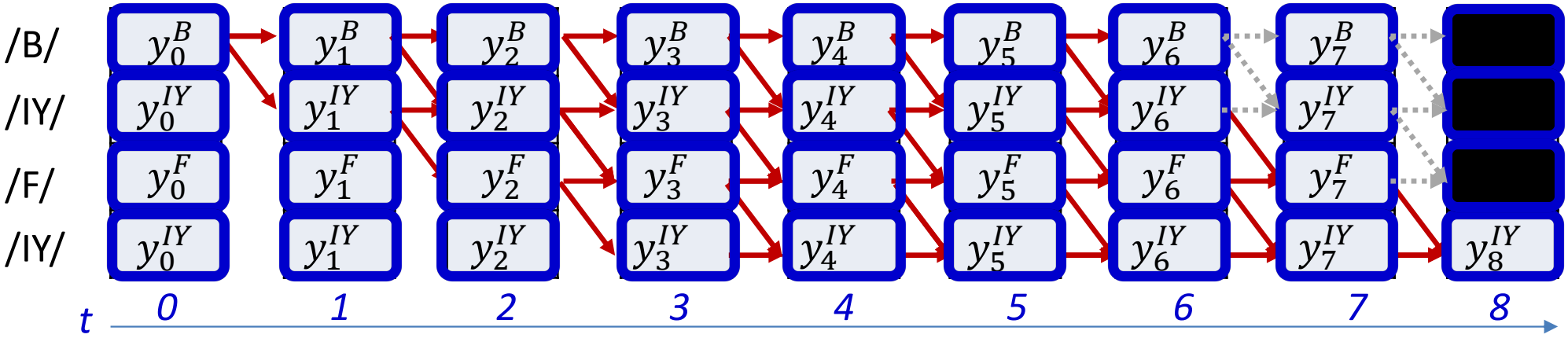
$$\hat{\beta}(T-1, K-1) = y_{t+1}^{S(K-1)}, \hat{\beta}(T-1, r) = 0, \; r < K-1$$

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

  for $r = K-1 \ldots 0$

$$\hat{\beta}(t, r) = y_t^{S(r)} \sum_{q \in succ(r)} \hat{\beta}(t+1, q)$$

Backward algorithm



- Initialization:

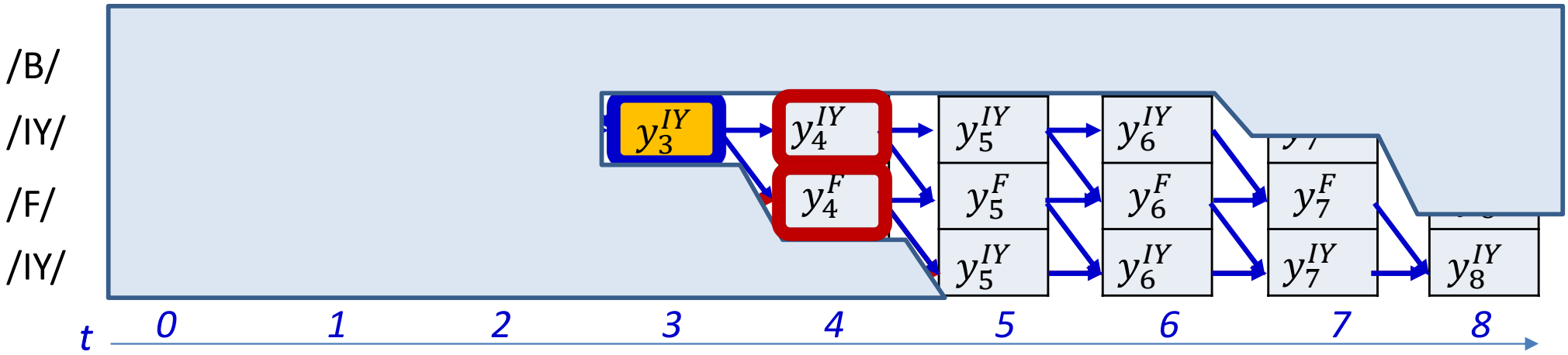
$$\hat{\beta}(T-1, K-1) = y_{t+1}^{S(K-1)}, \hat{\beta}(T-1, r) = 0, \; r < K-1$$

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

  for $r = K-1 \ldots 0$

$$\hat{\beta}(t, r) = y_t^{S(r)} \sum_{q \in succ(r)} \hat{\beta}(t+1, q)$$

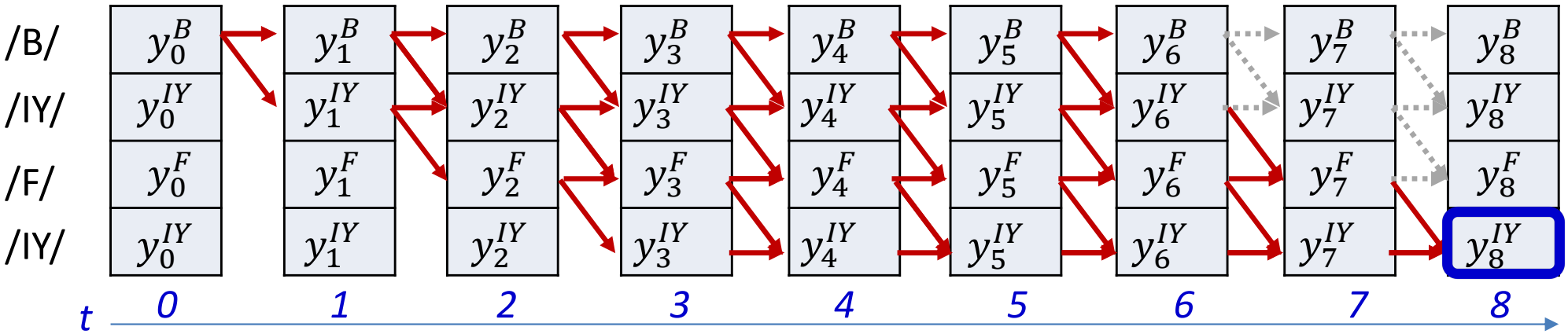
Backward algorithm



- This recursion gives us $\hat{\beta}(t,r)$ which includes the node at (t,r)
- The actual backward probability is obtained as

$$\beta(t,r) = \frac{1}{y_t^{s_r}}\hat{\beta}(t,r)$$

Backward algorithm



- Initialization:

$$\hat{\beta}(T-1, K-1) = y_{T-1}^{S(K-1)}, \;\; \hat{\beta}(T-1, r) = 0, \;\; r < K-1$$

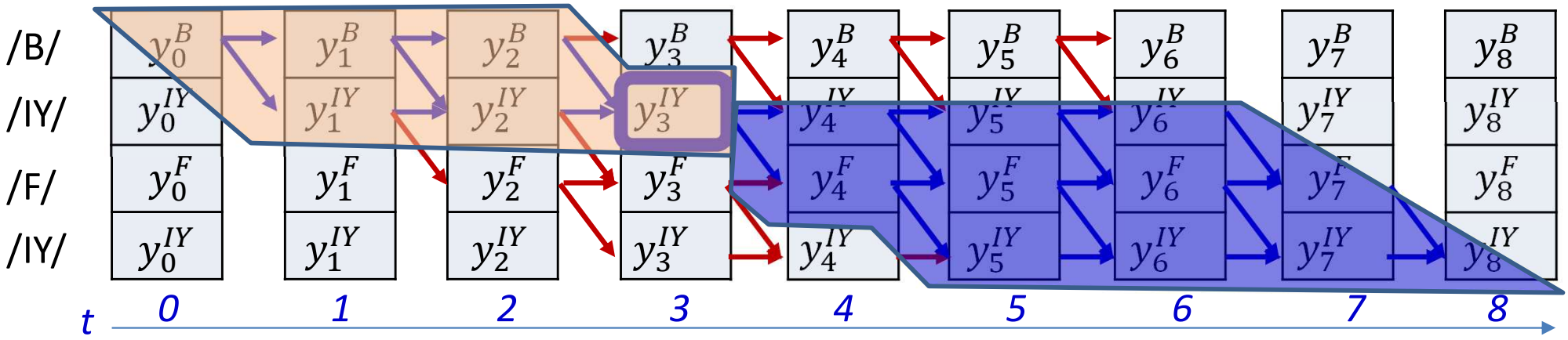
- for $t = T-2$ *downto* 0

  for $r = K-1 \ldots 0$

$$\hat{\beta}(t, r) = y_t^{S(r)} \sum_{q \in succ(r)} \hat{\beta}(t+1, q)$$

$$\boldsymbol{\beta(t,r) = \frac{1}{y_t^{S(r)}} \hat{\beta}(t,r)}$$

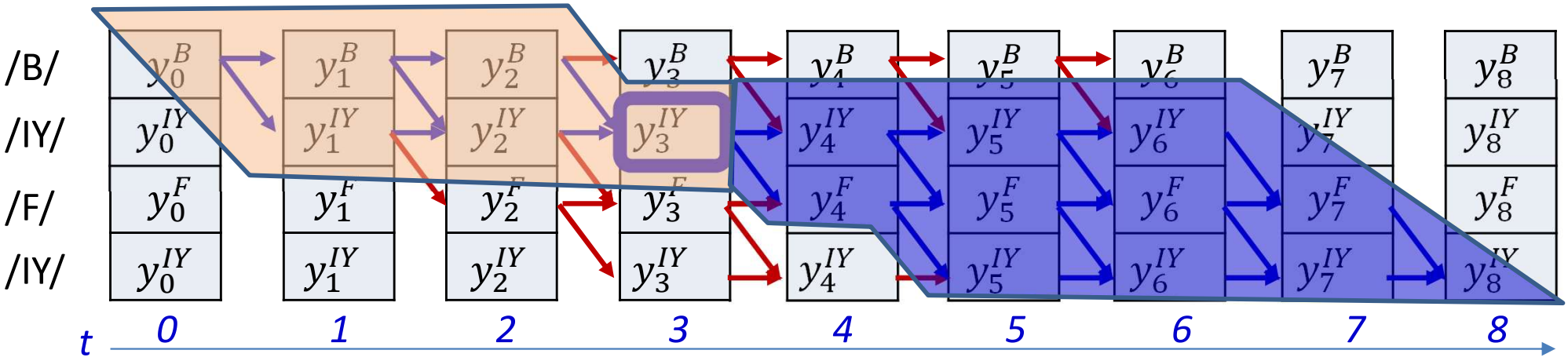
A posteriori symbol probability



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

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

The joint probability



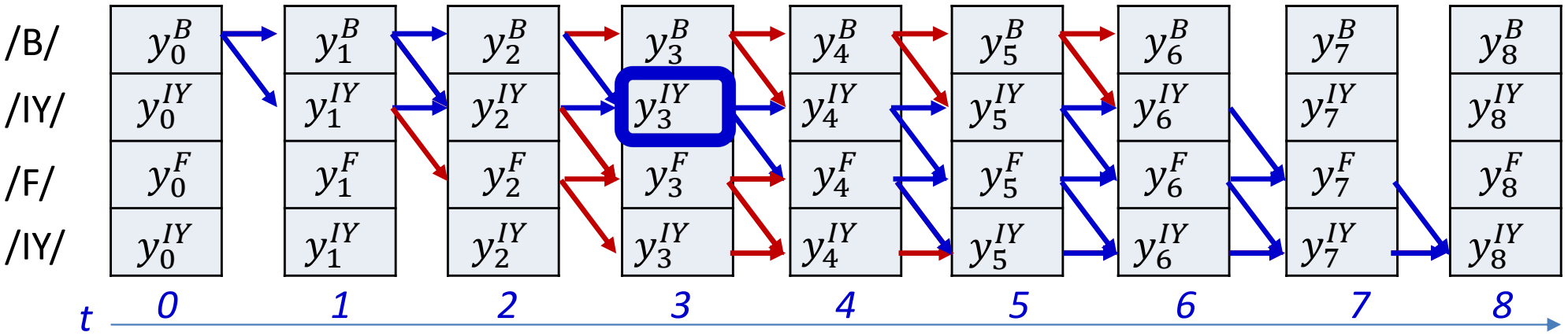
$$P(s_t = S_r, \mathbf{S}|\mathbf{X}) = \alpha(t,r)\beta(t,r)$$

- 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

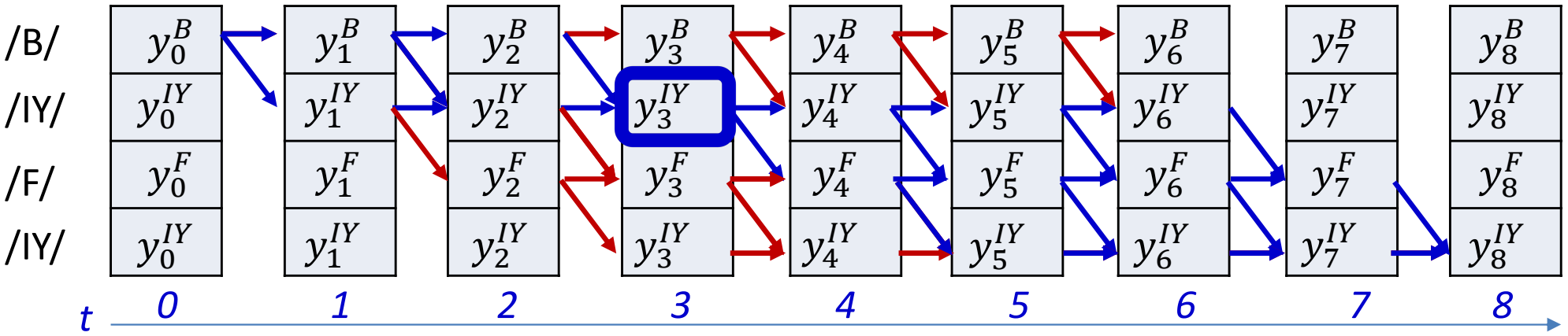


$$P(s_t = S_r, \mathbf{S}|\mathbf{X}) = \alpha(t,r)\beta(t,r)$$

- The *posterior* is given by

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

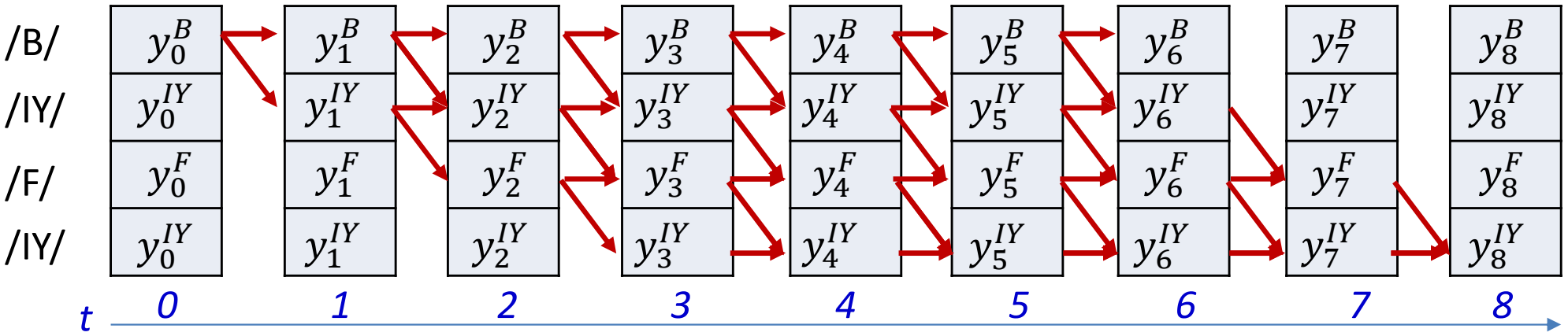
The posterior probability



- 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 divergence



$$DIV = -\sum_t \sum_{s \in S_0 \ldots S_{K-1}} P(s_t = s|\mathbf{S}, \mathbf{X}) \log Y(t, s_t = s)$$

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

Poll 3

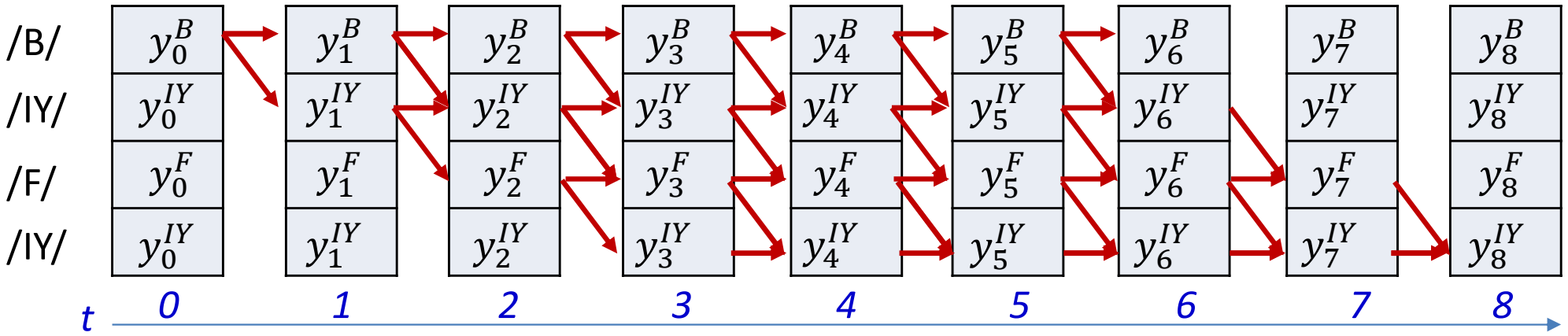
- @

Poll 3

Select all that are true

- **The forward-backward algorithm is used to compute the a posteriori probability of aligning each symbol in the compressed sequence to each input**
- **These probabilities are required to compute the expected divergence across all alignments of the compressed symbol sequence to the input**

The expected divergence



$$DIV = -\sum_{t} \sum_{s \in S_0 \ldots S_{K-1}} P(s_t = s|\mathbf{S}, \mathbf{X}) \log Y(t, s_t = s)$$

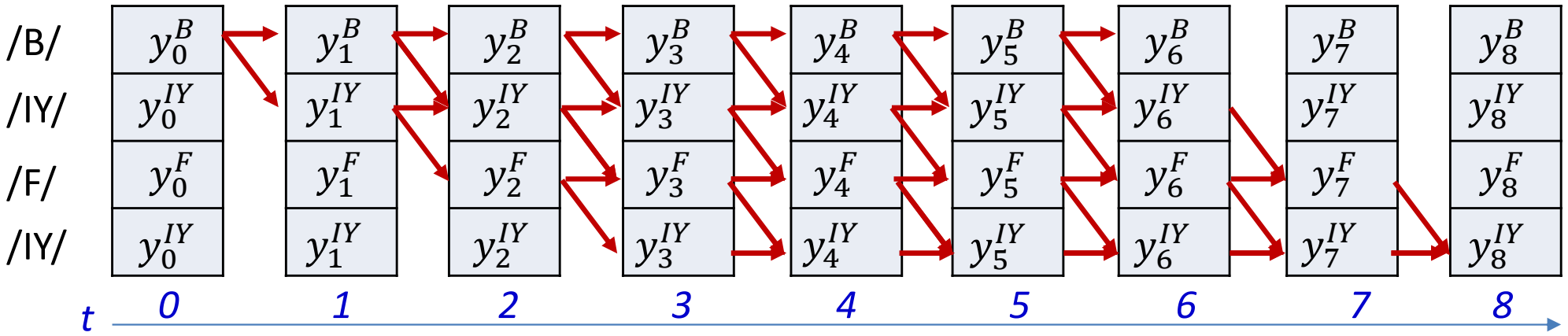
$$DIV = -\sum_{t} \sum_{r} \gamma(t, r) \log y_t^{S(r)}$$

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

$$\nabla_{Y_t} DIV = \left[\frac{dDIV}{dy_t^{S_0}} \quad \frac{dDIV}{dy_t^{S_1}} \quad \ldots \quad \frac{dDIV}{dy_t^{S_{L-1}}} \right]$$

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

The expected divergence



$$DIV = -\sum_{t} \sum_{s \in S_0 \ldots S_{K-1}} P(s_t = s|\mathbf{S}, \mathbf{X}) \log Y(t, s_t = s)$$

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

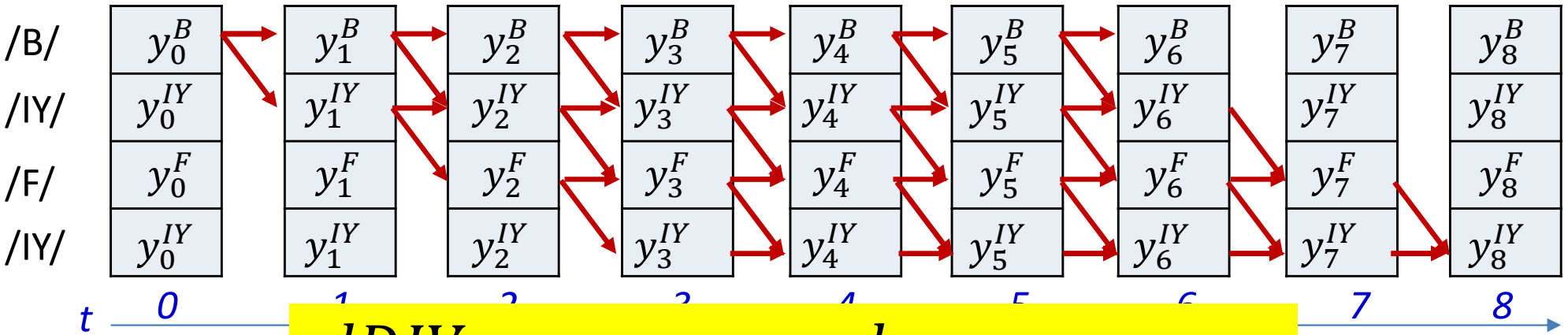
- The derivative of the divergence w.r.t the output Y_t of the net at any time:

$$\nabla_{Y_t} DIV = \left[\frac{dDIV}{dy_t^{s_0}} \quad \frac{dDIV}{dy_t^{s_1}} \quad \ldots \quad \frac{d}{d} \right.$$

Must compute these terms from here

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

The expected divergence



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

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

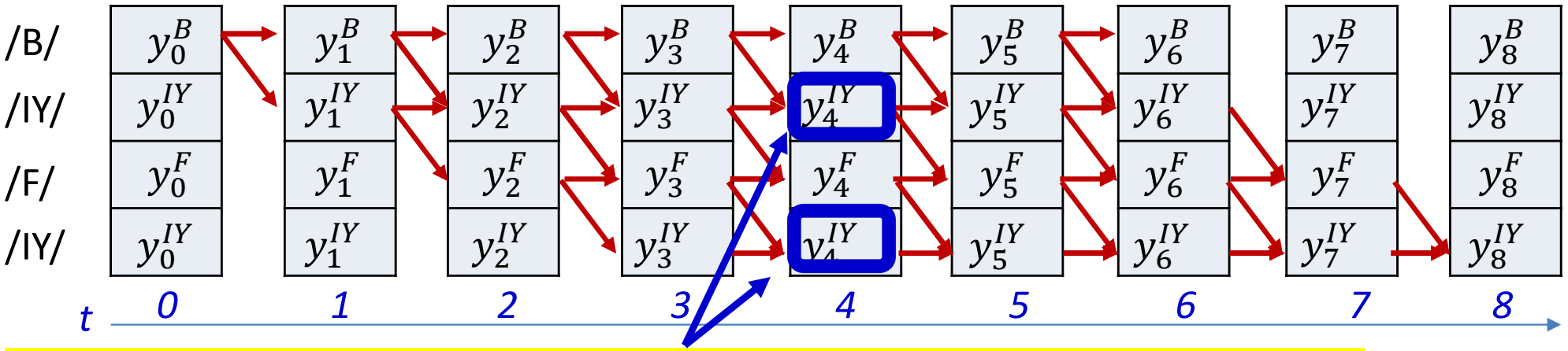
- The derivative of the divergence w.r.t the output Y_t of the net at any time:

$$\nabla_{Y_t} DIV = \left[\frac{dDIV}{dy_t^{S_0}} \quad \frac{dDIV}{dy_t^{S_1}} \quad \ldots \quad \frac{d}{d}\right.$$

Must compute these terms from here

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

The expected divergence



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

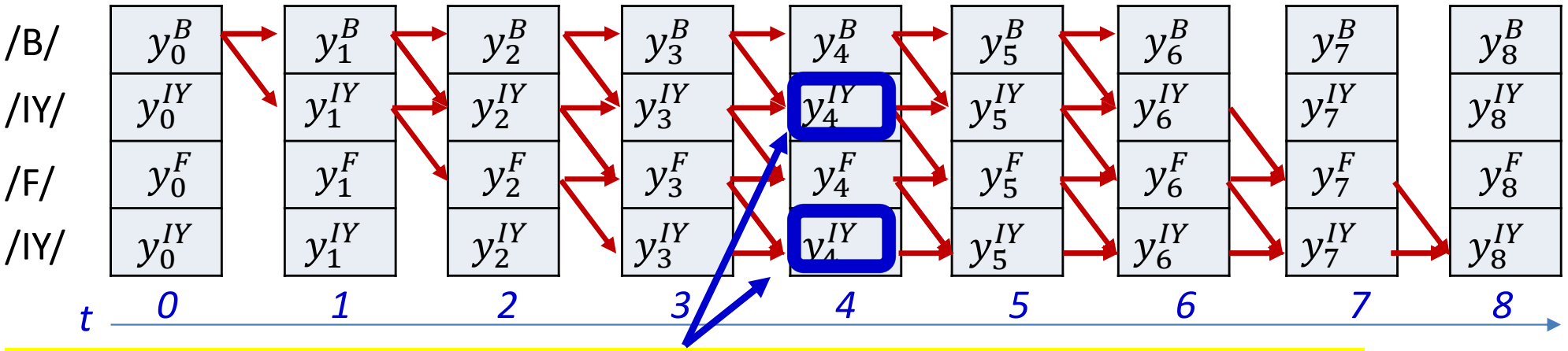
$$\frac{dDIV}{dy_t^l} = -\sum_{r:S(r)=l} \frac{d}{dy_t^l} \gamma(t,r) \log y_t^l$$

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

$$\nabla_{Y_t} DIV = \left[\frac{dDIV}{dy_t^{s_0}} \quad \frac{dDIV}{dy_t^{s_1}} \quad \cdots \quad \frac{dDIV}{dy_t^{s_{L-1}}} \right]$$

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

The expected divergence



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

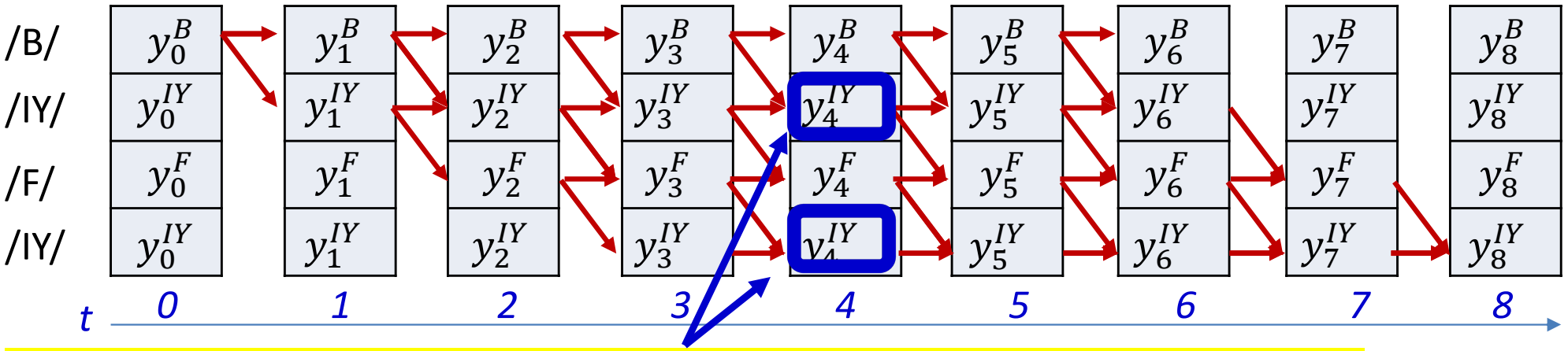
$$\frac{dDIV}{dy_t^l} = -\sum_{r:S(r)=l} \frac{d}{dy_t^l}\gamma(t,r)\log y_t^l$$

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

$$\nabla_{Y_t} DIV = \left[\frac{dDIV}{dy_t^{s_0}} \quad \frac{dDIV}{dy_t^{s_1}} \quad \cdots \quad \frac{dDIV}{dy_t^{s_{L-1}}}\right]$$

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

The expected divergence

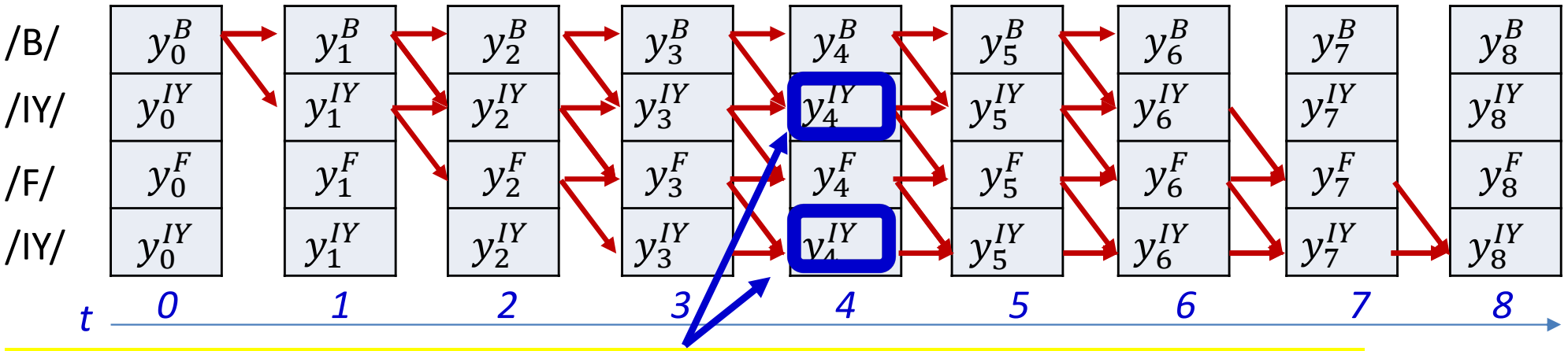


The derivatives at both these locations must be summed to get $\frac{dDIV}{dy_4^{IY}}$

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

- $$\frac{d}{dy_t^l}\gamma(t,r)\log y_t^l = \frac{\gamma(t,r)}{y_t^l} + \frac{d\gamma(t,r)}{dy_t^l}\log y_t^l$$ any time:
  - Components will be non-zero only for symbols that occur in the training instance

The expected divergence



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

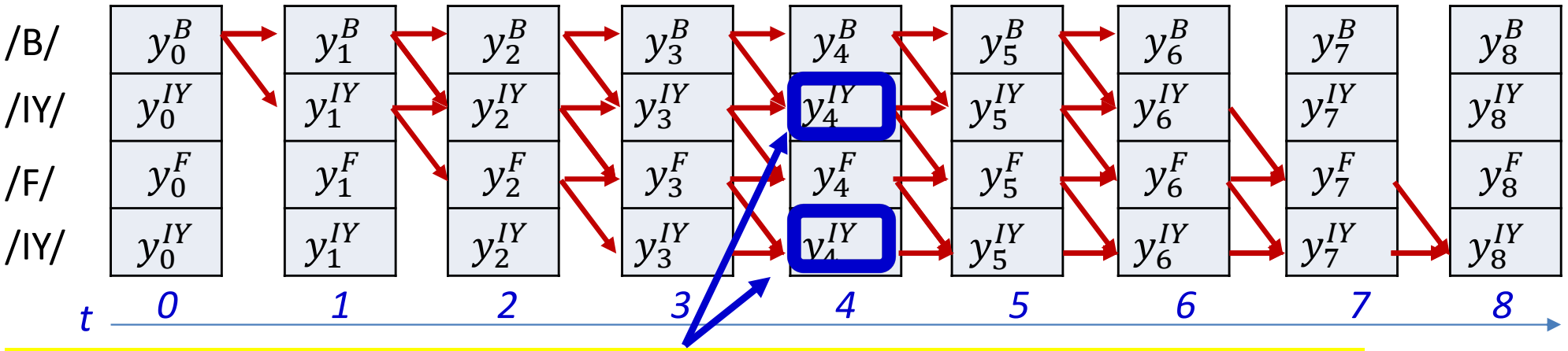
$$\frac{dDIV}{dy_t^l} = -\sum_{r:S(r)=l} \frac{d}{dy_t^l}\gamma(t,r)\log y_t^l$$

- The derivative of t … net at any time:

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

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

Derivative of the expected divergence



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

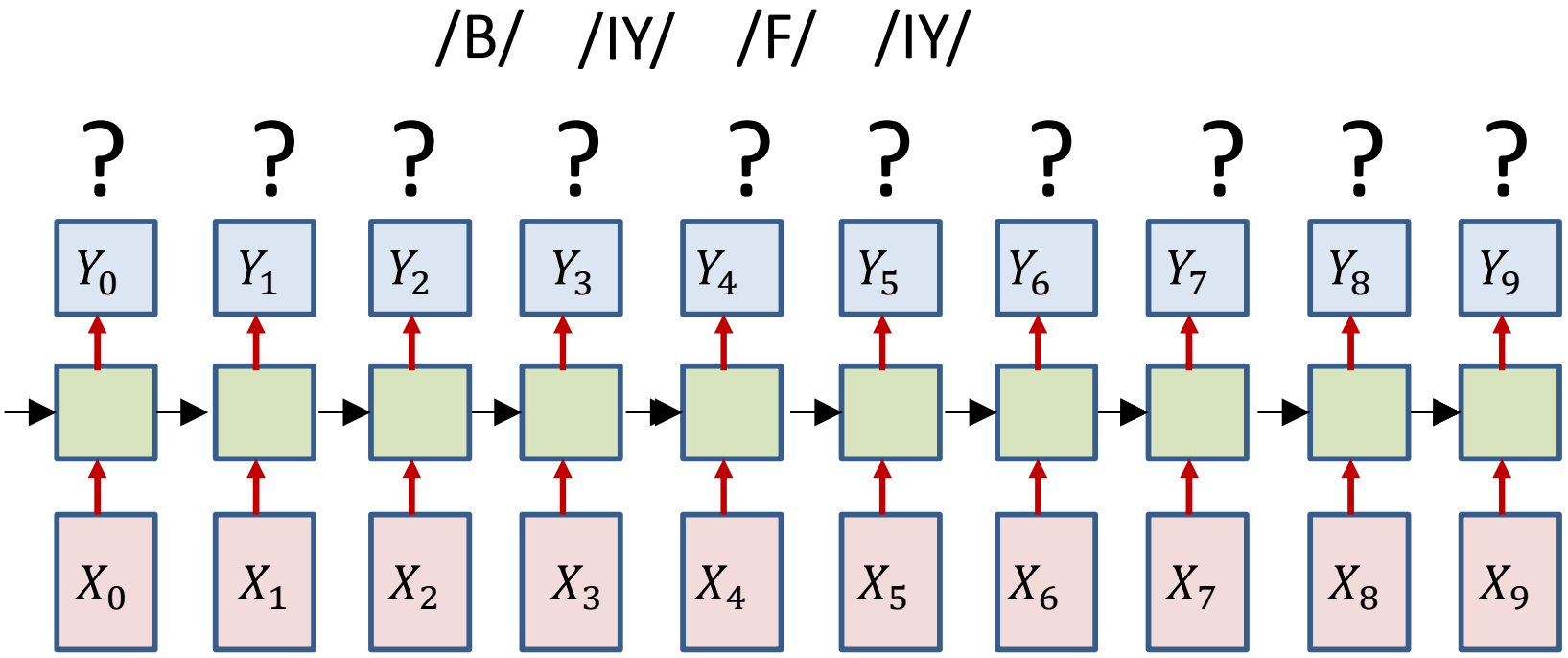
$$DIV = -\sum_t \sum_r \gamma(t,r) \log y_t^{S(r)}$$

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

$$\frac{dDIV}{dy_t^l} = -\frac{1}{y_t^l} \sum_{r\,:S(r)=l} \gamma(t,r)$$

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

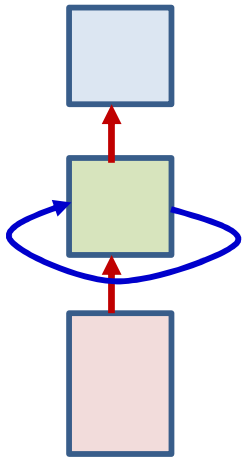
Overall training procedure for Seq2Seq case 1



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

Overall training procedure for Seq2Seq case 1

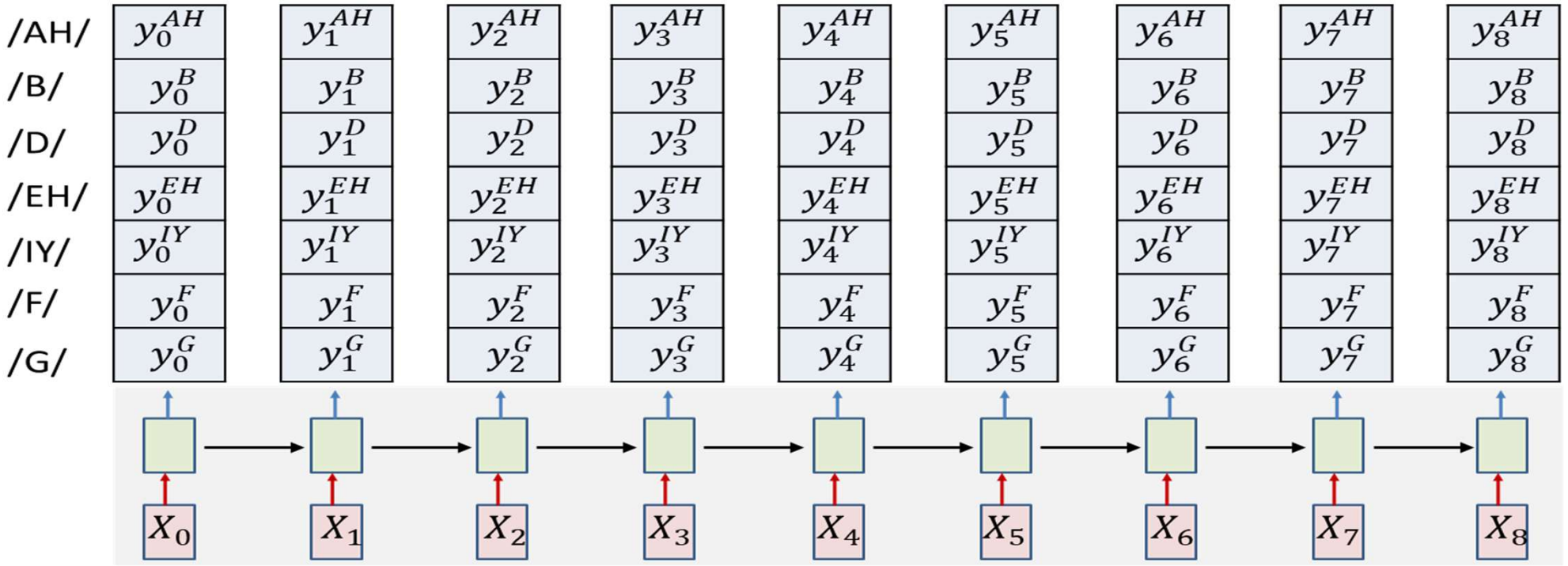
- **Step 1**: Setup the network
  - Typically many-layered LSTM



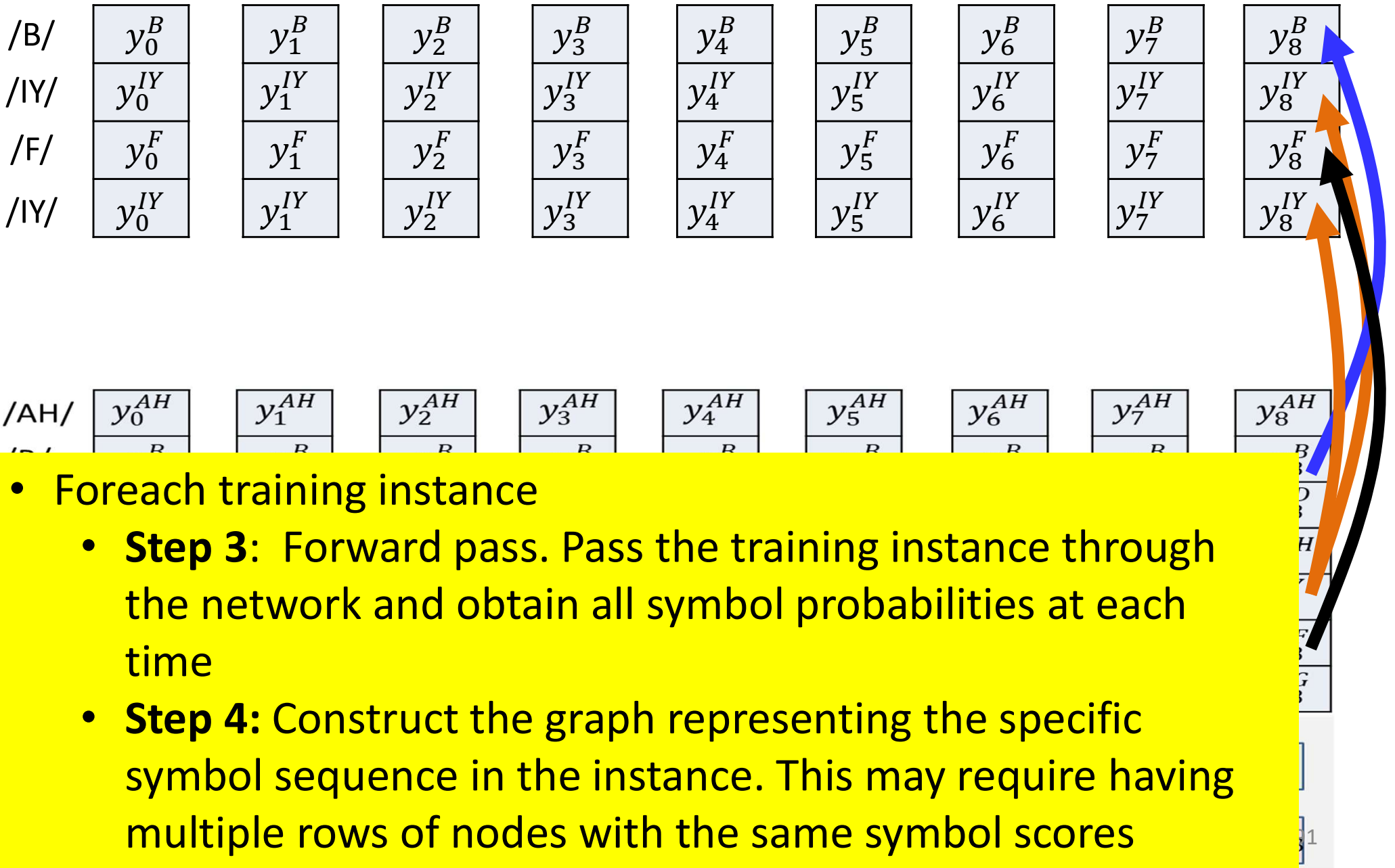
- **Step 2**: Initialize all parameters of the network

Overall Training: Forward pass

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

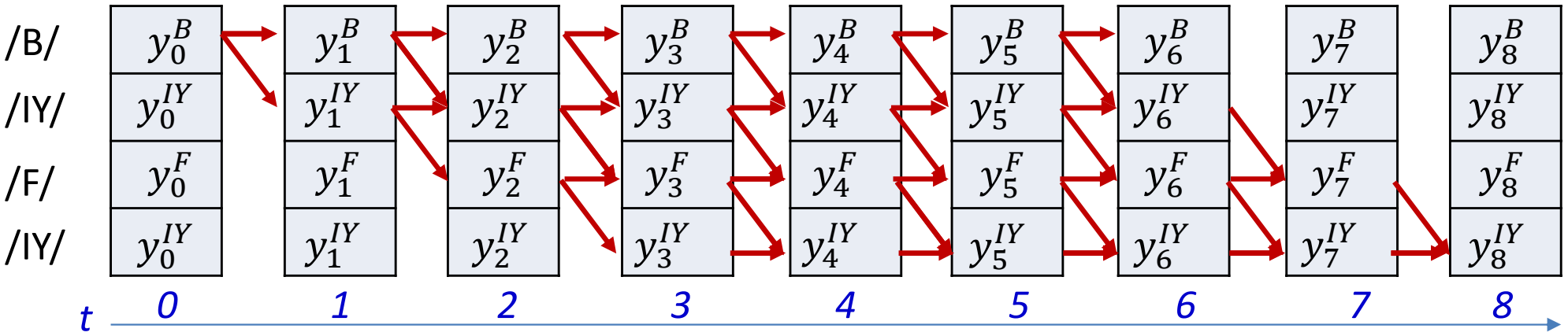


Overall training: Backward pass



- Foreach 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. Compute $\gamma(t,r)$.
  - **Step 6:** Compute derivative of divergence $\nabla_{Y_t} DIV$ for each Y_t

Overall training: Backward pass

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

$$\nabla_{Y_t} DIV = \left[\frac{dDIV}{dy_t^0} \quad \frac{dDIV}{dy_t^1} \quad \ldots \quad \frac{dDIV}{dy_t^{L-1}} \right]$$

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

- **Step 7**: Backpropagate $\frac{dDIV}{dy_t^l}$ and aggregate derivatives over minibatch and update parameters

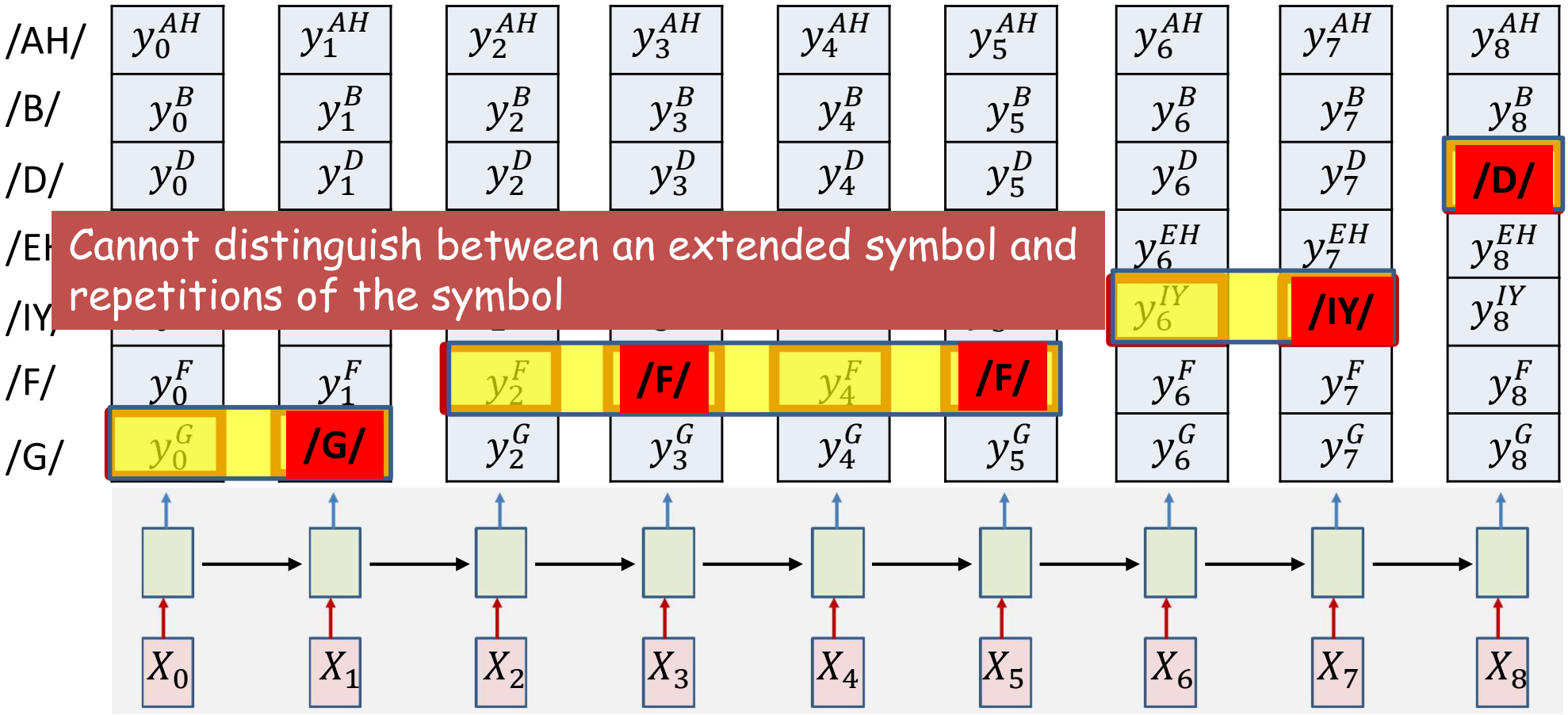
Story so far: CTC models

- 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

A key *decoding* problem

- Consider a problem where the output symbols are characters
- We have a decode: R R R E E E E D
- Is this the compressed symbol sequence RED or REED?

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 E E E E E D
- Is this the symbol sequence RED or REED?

- Solution: Introduce an explicit extra symbol which serves to separate discrete versions of a symbol
  - A “blank” (represented by “-”)
  - RRR---EE---DDD = RED
  - RR-E--EED = REED
  - RR-R---EE---D-DD = RREDD
  - R-R-R---E-EDD-DDDD-D = [redacted]
    - 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

A key *decoding* problem

- We have a decode: R R R E E E E E D
- Is this the symbol sequence RED or REED?

- Solution: Introduce an explicit extra symbol which serves to separate discrete versions of a symbol
  - A “blank” (represented by “-”)
  - RRR---EE---DDD = RED
  - RR-E--EED = REED
  - RR-R---EE---D-DD = RREDD
  - R-R-R---E-EDD-DDDD-D = RRREEDDD
    - 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

Poll 4

- @

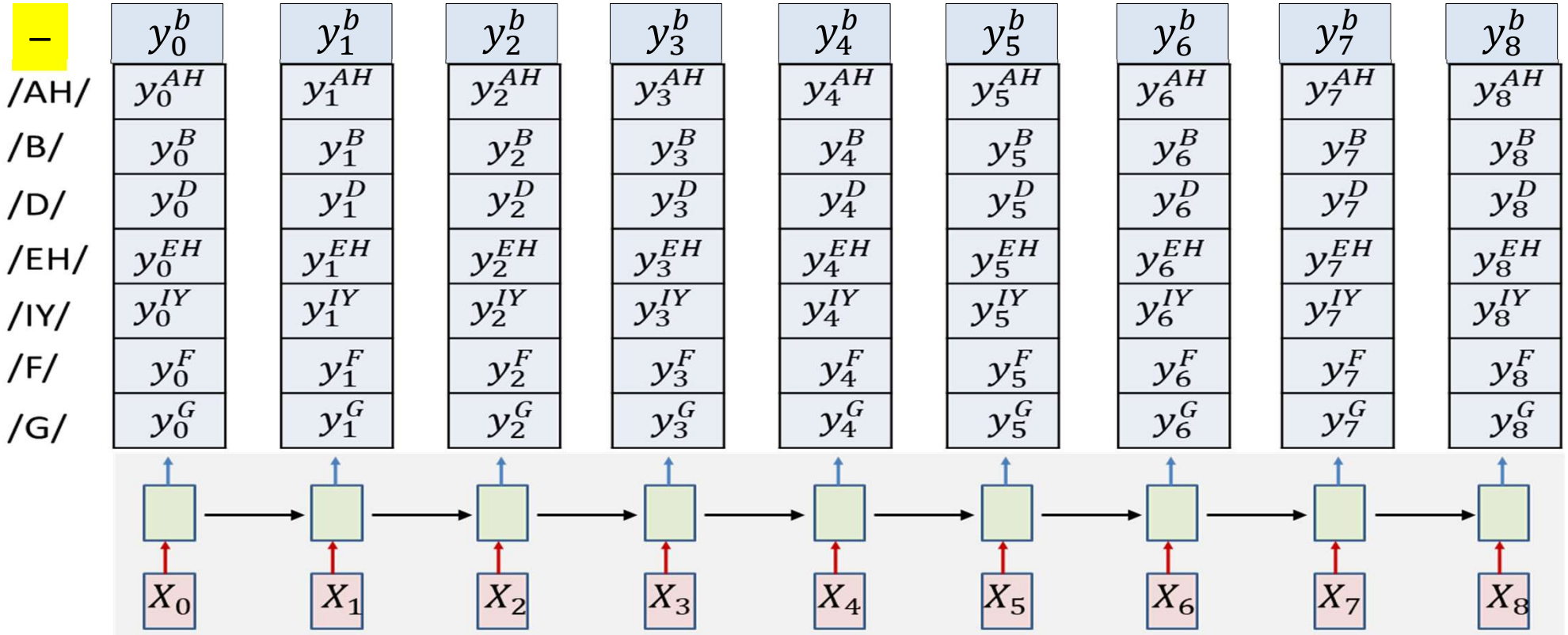
Poll 4

Which of the following are valid expansions of the character string "BILLY"?

- B B I I L L Y
- B – B I L – L Y
- B – I – L L Y
- **B – I – L – L Y Y**

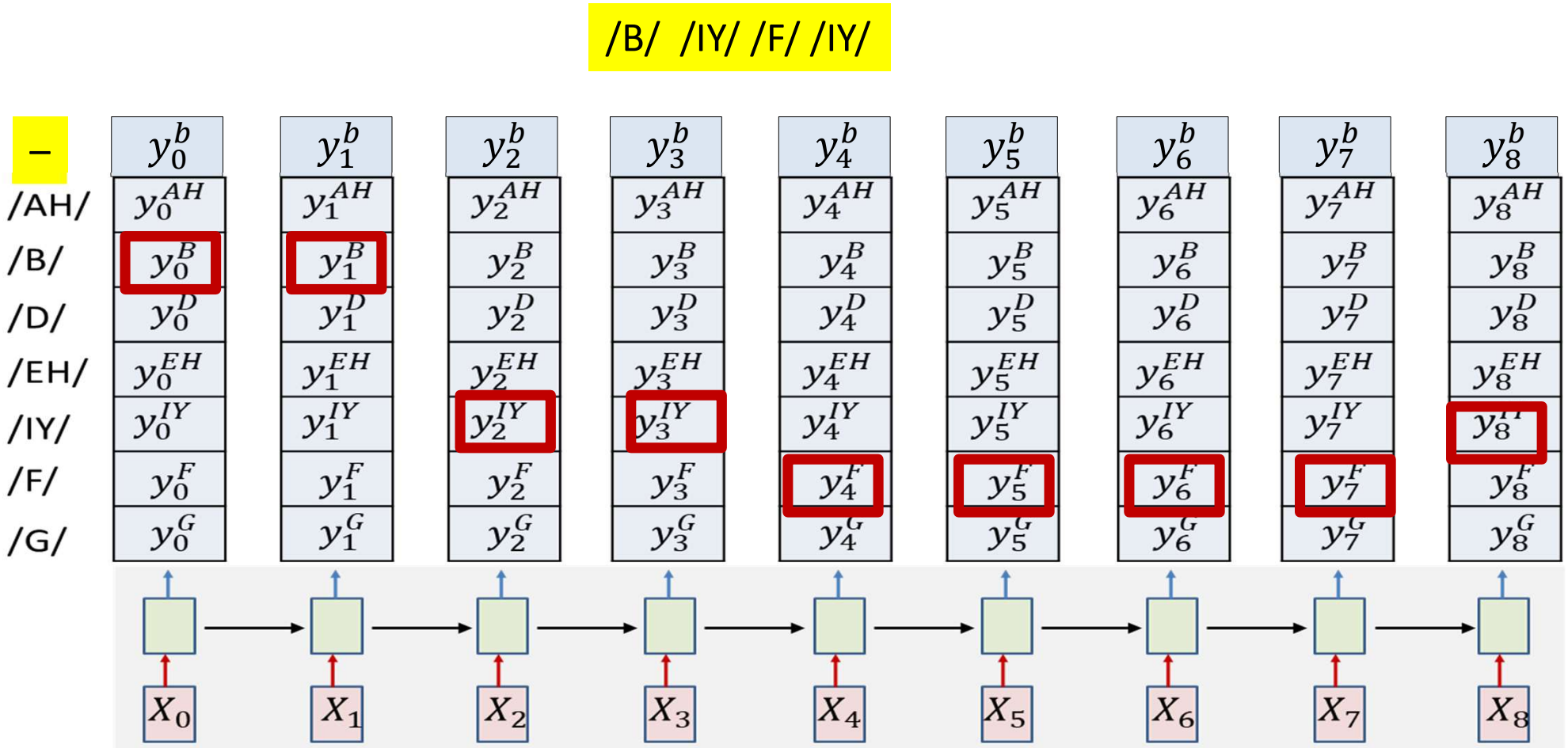
The modified forward output

- Note the extra “blank” at the output



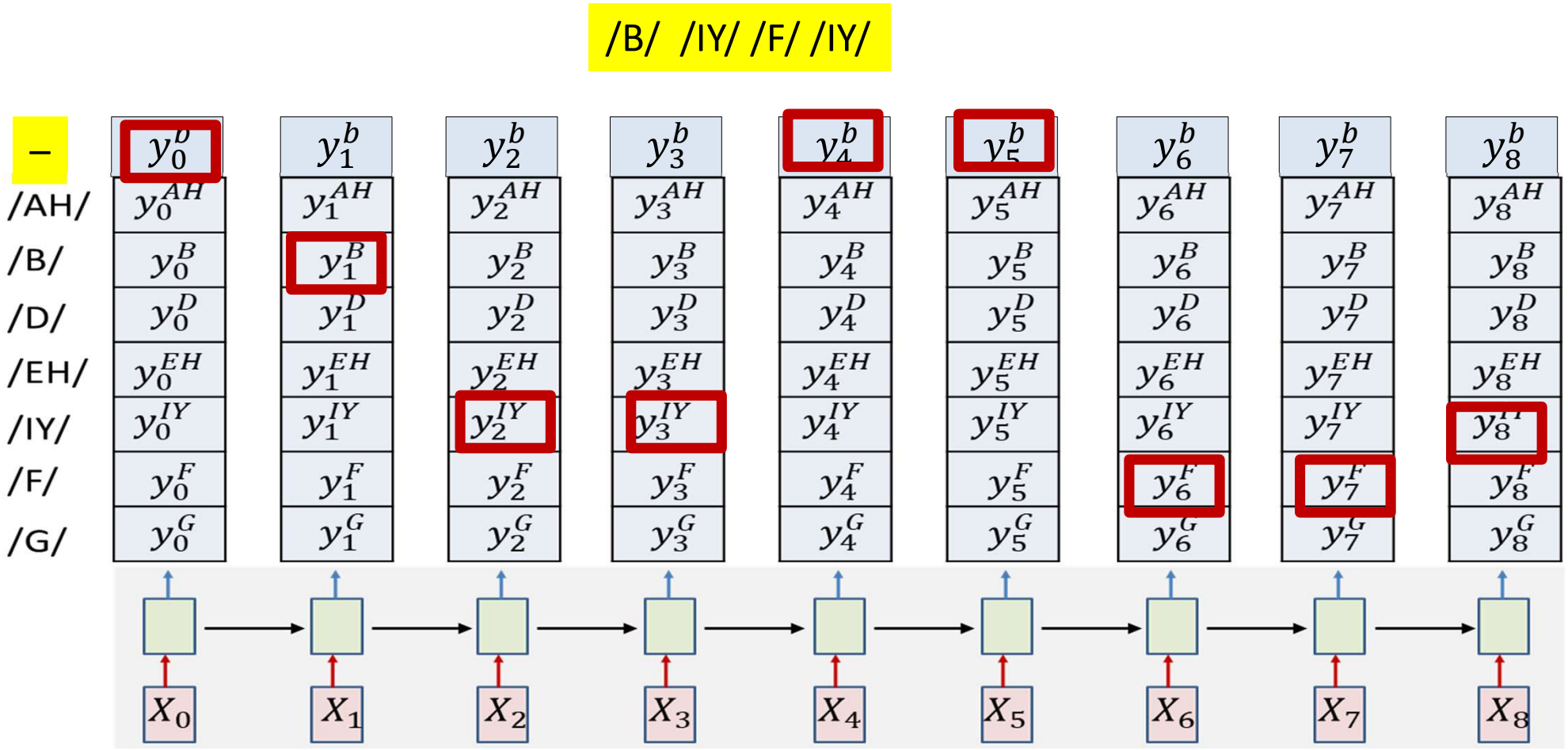
The modified forward output

- Note the extra “blank” at the output



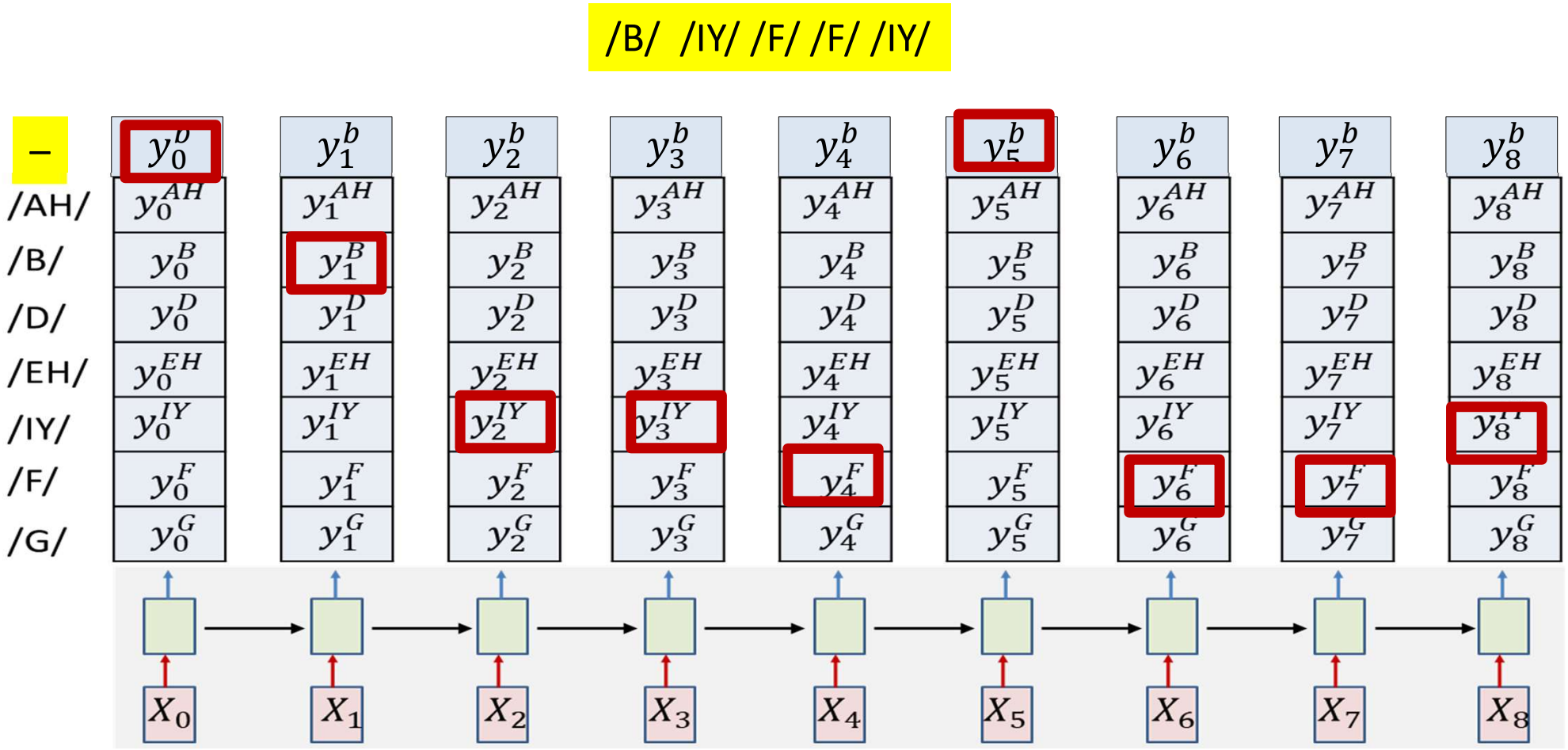
The modified forward output

- Note the extra “blank” at the output

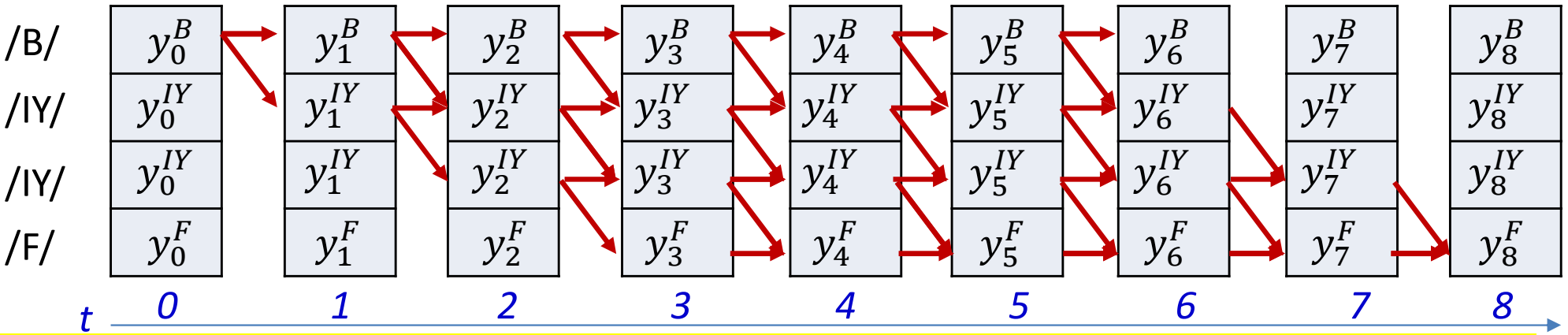


The modified forward output

- Note the extra “blank” at the output

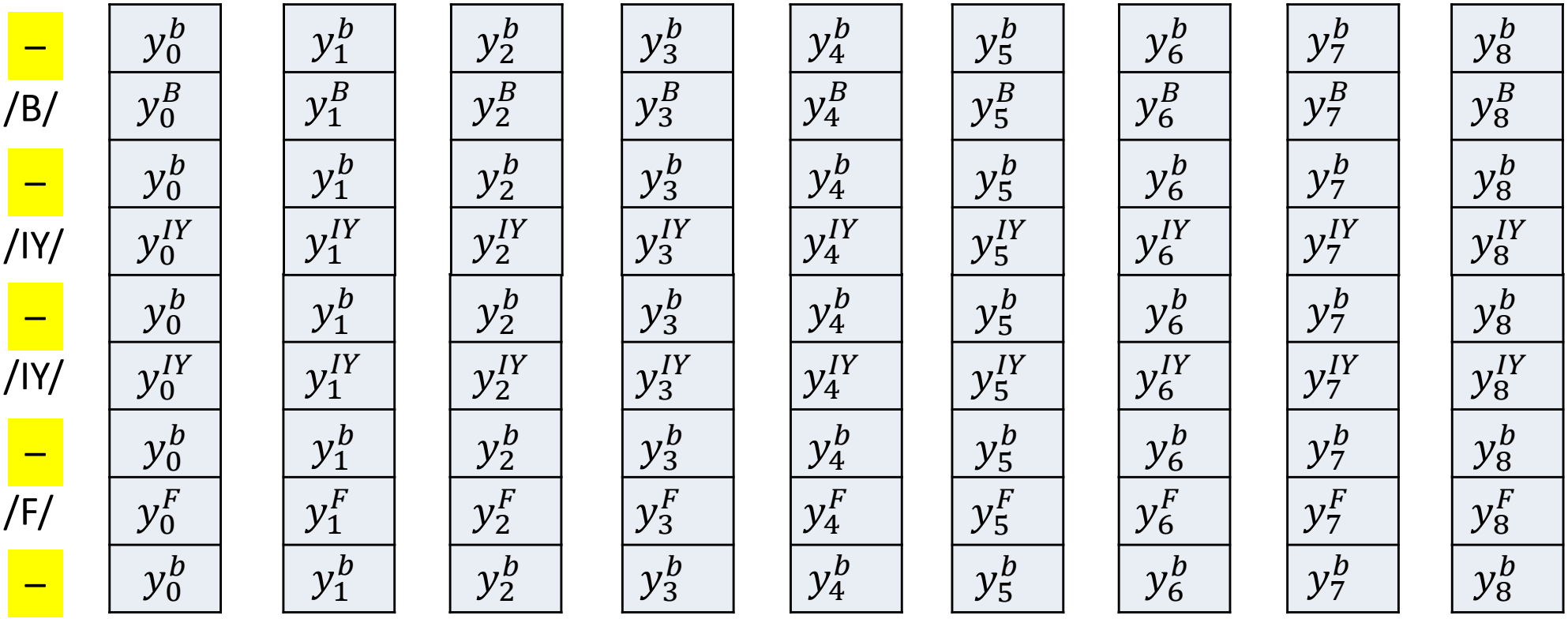


Composing the graph for training



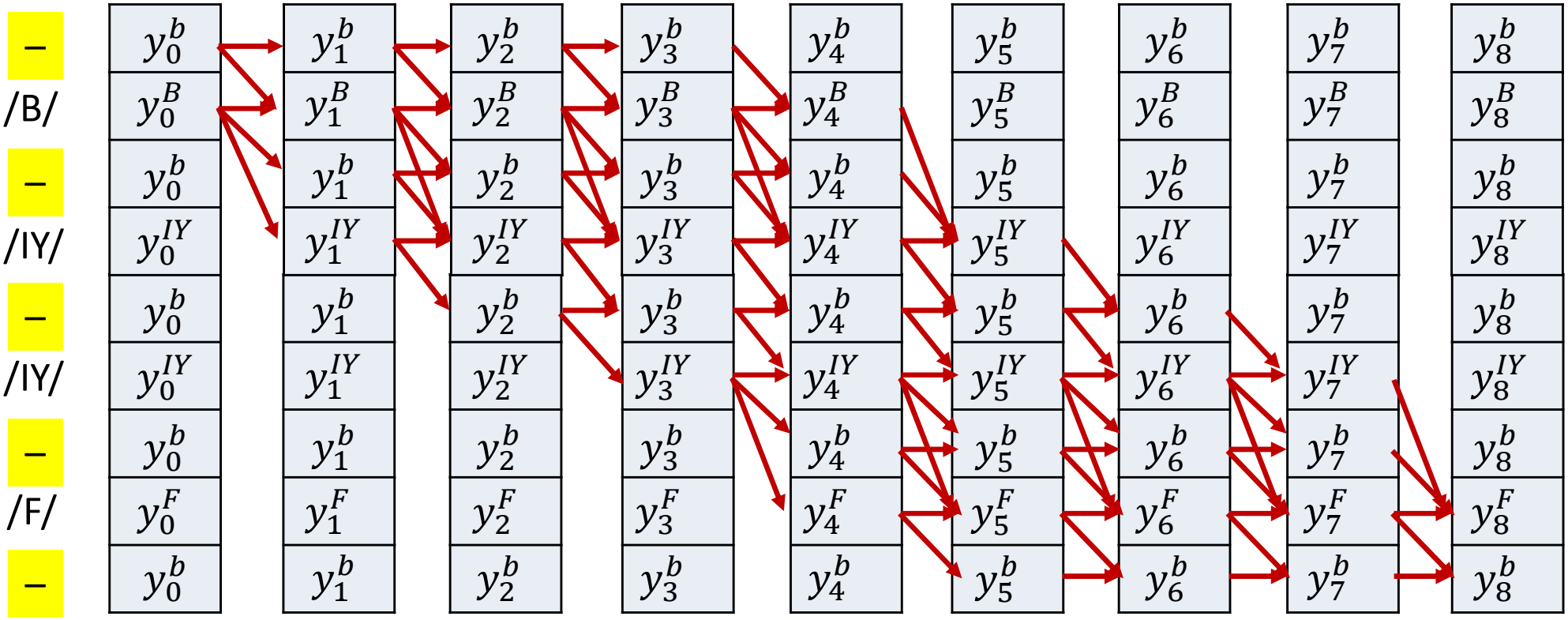
- The original method without blanks
- **Changing the example to /B/ /IY/ /IY/ /F/ from /B/ /IY/ /F/ /IY/ for illustration**

Composing the graph for training



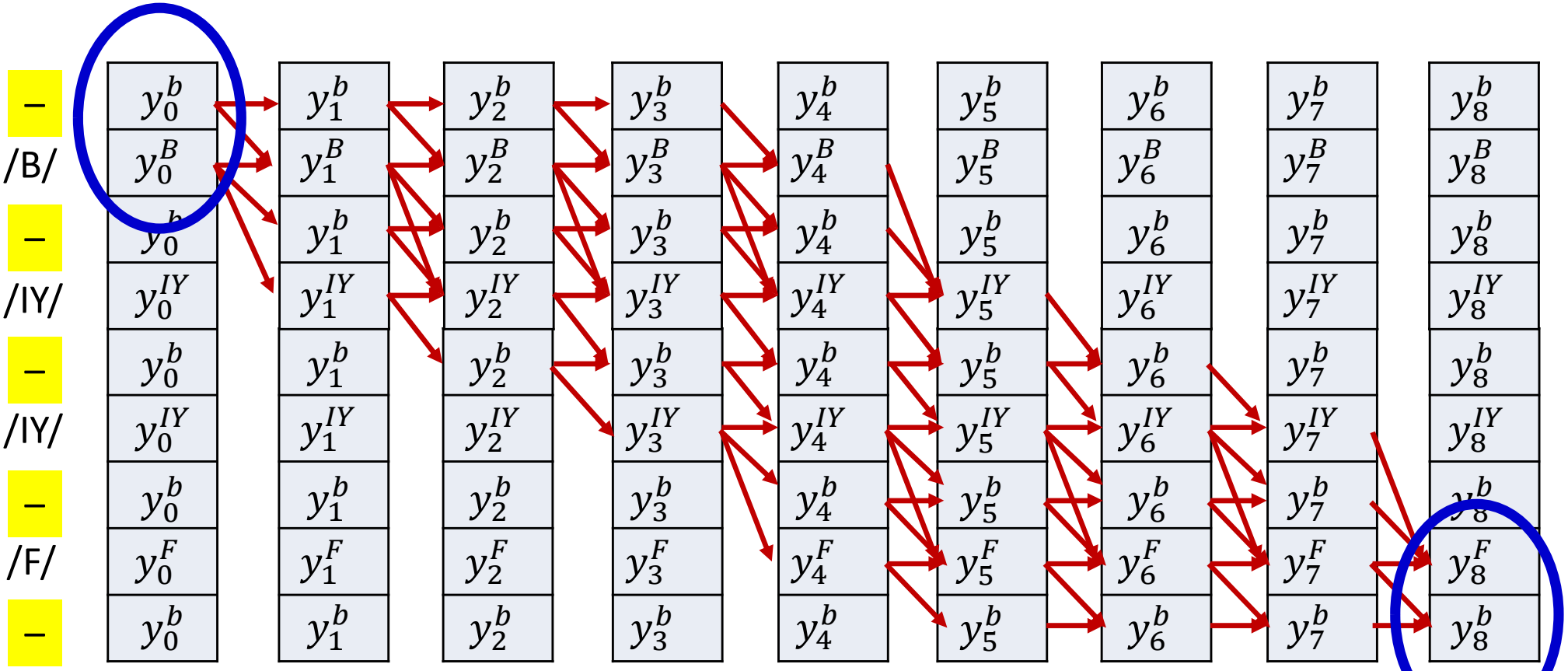
- With blanks
- Note: a row of blanks between any two symbols
- Also blanks at the very beginning and the very end

Composing the graph for training



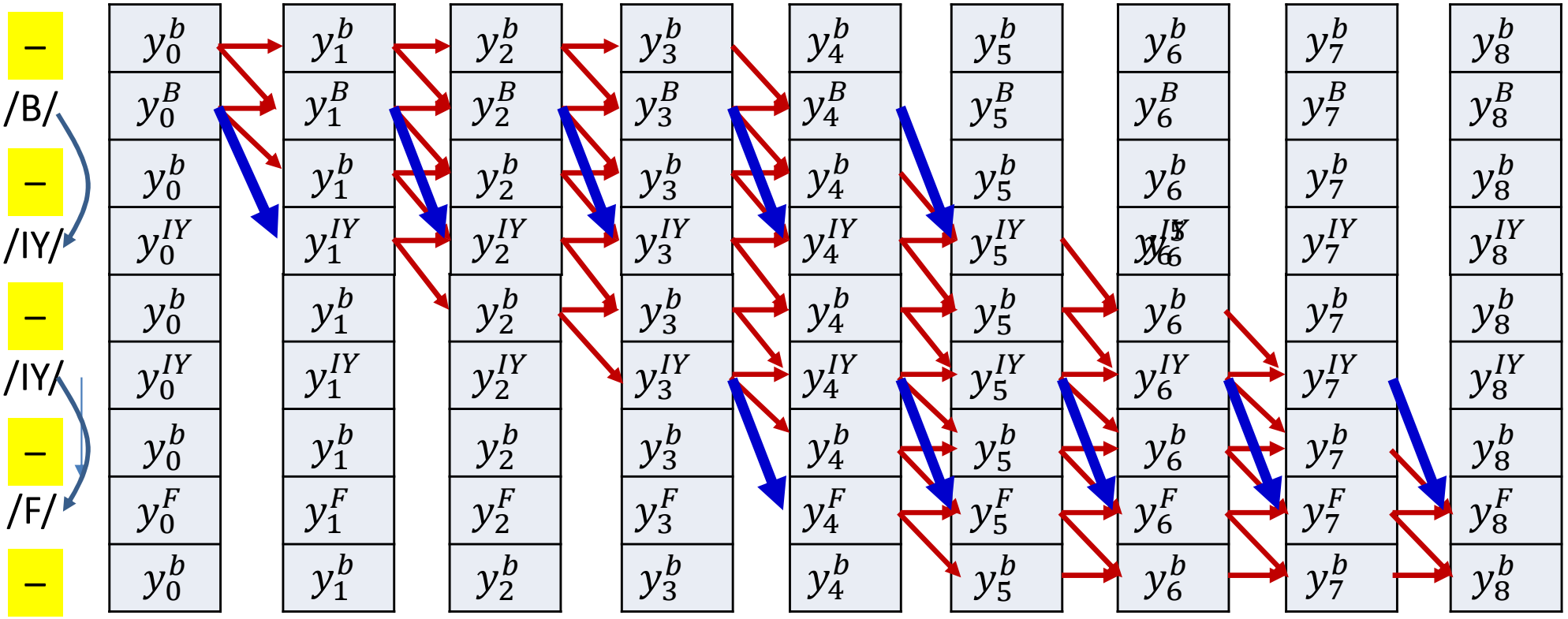
- Add edges such that all paths from initial node(s) to final node(s) unambiguously represent the target symbol sequence

Composing the graph for training



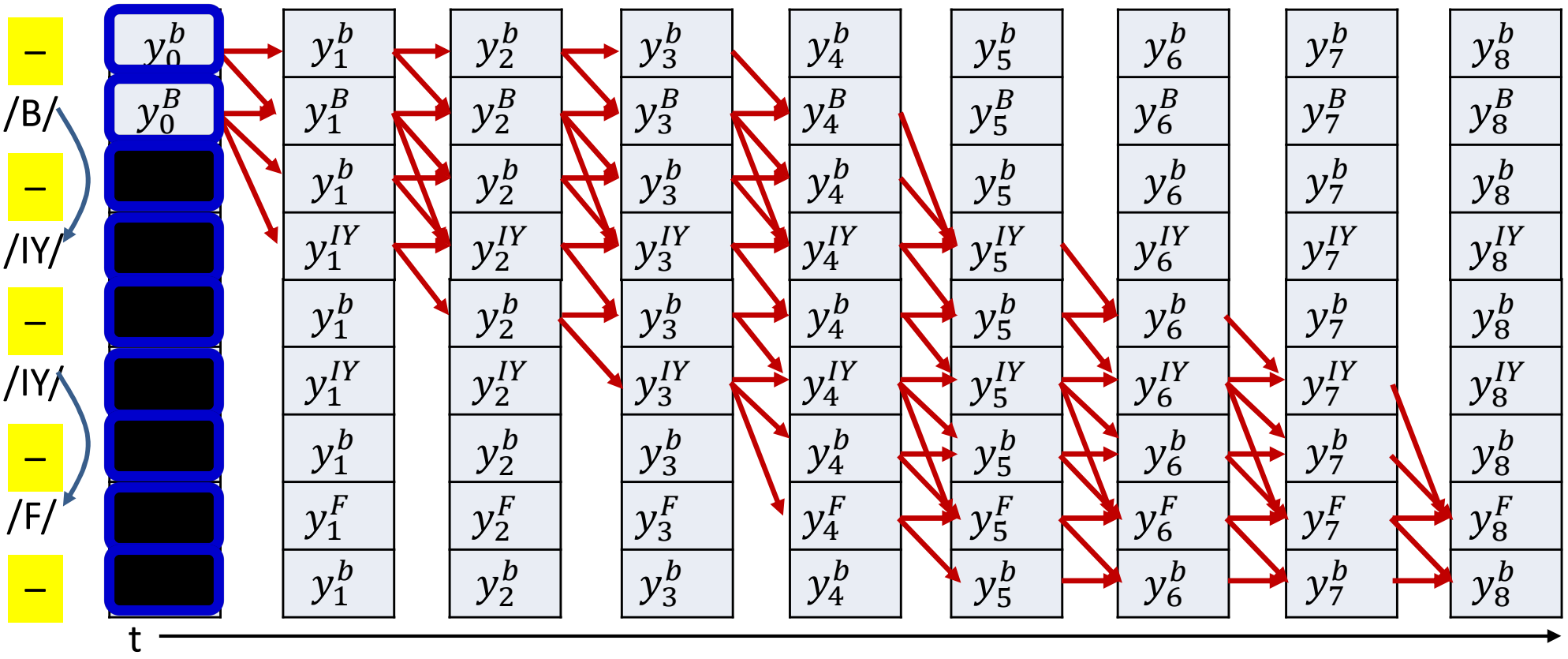
- The first and last column are allowed to also end at initial and final blanks

Composing the graph for training



- The first and last column are allowed to also end at initial and final blanks
- Skips are permitted across a blank, but only if the symbols on either side are different
  - Because a blank is *mandatory between repetitions of a symbol* but *not required between distinct symbols*

Modified Forward Algorithm

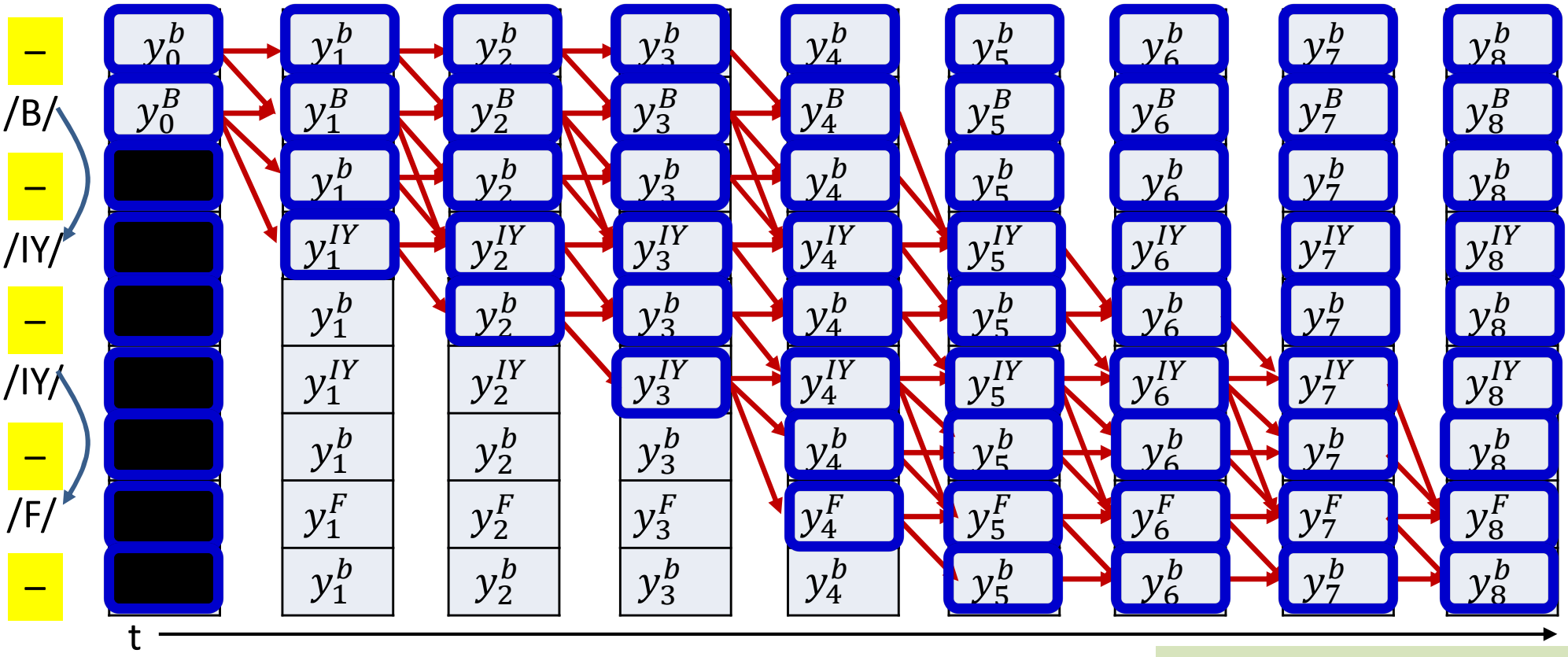


- Initialization:

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

$S(k)$ refers to the *extended* sequence with blanks included

Modified Forward Algorithm



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

- Iteration t = 1:N:

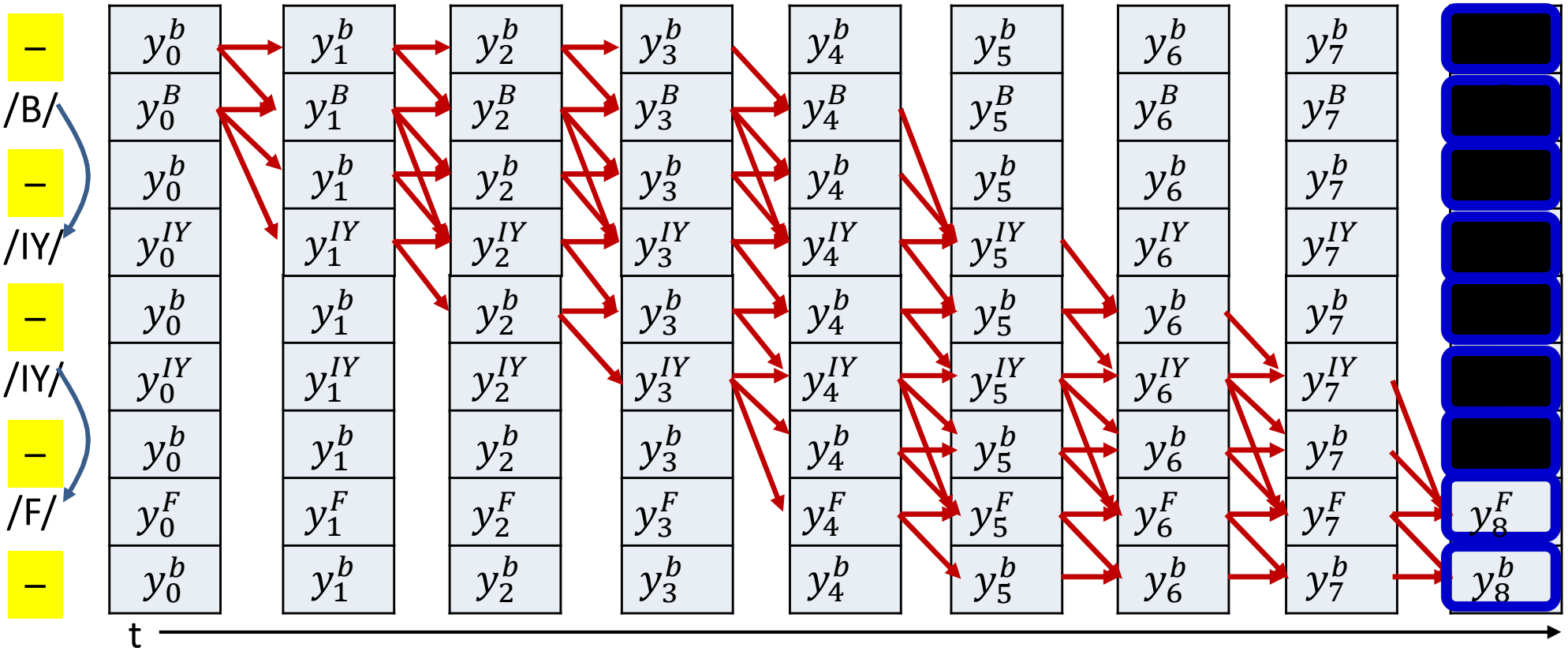
$$\alpha(t,r) = \big(\alpha(t-1,r) + \alpha(t-1,r-1)\big) y_t^{S(r)}$$

  - If $S(r) = \text{" – "}$ or $S(r) = S(r-2)$

$$\alpha(t,r) = \big(\alpha(t-1,r) + \alpha(t-1,r-1) + \alpha(t-1,r-2)\big) y_t^{S(r)}$$

  - Otherwise

Modified Backward Algorithm



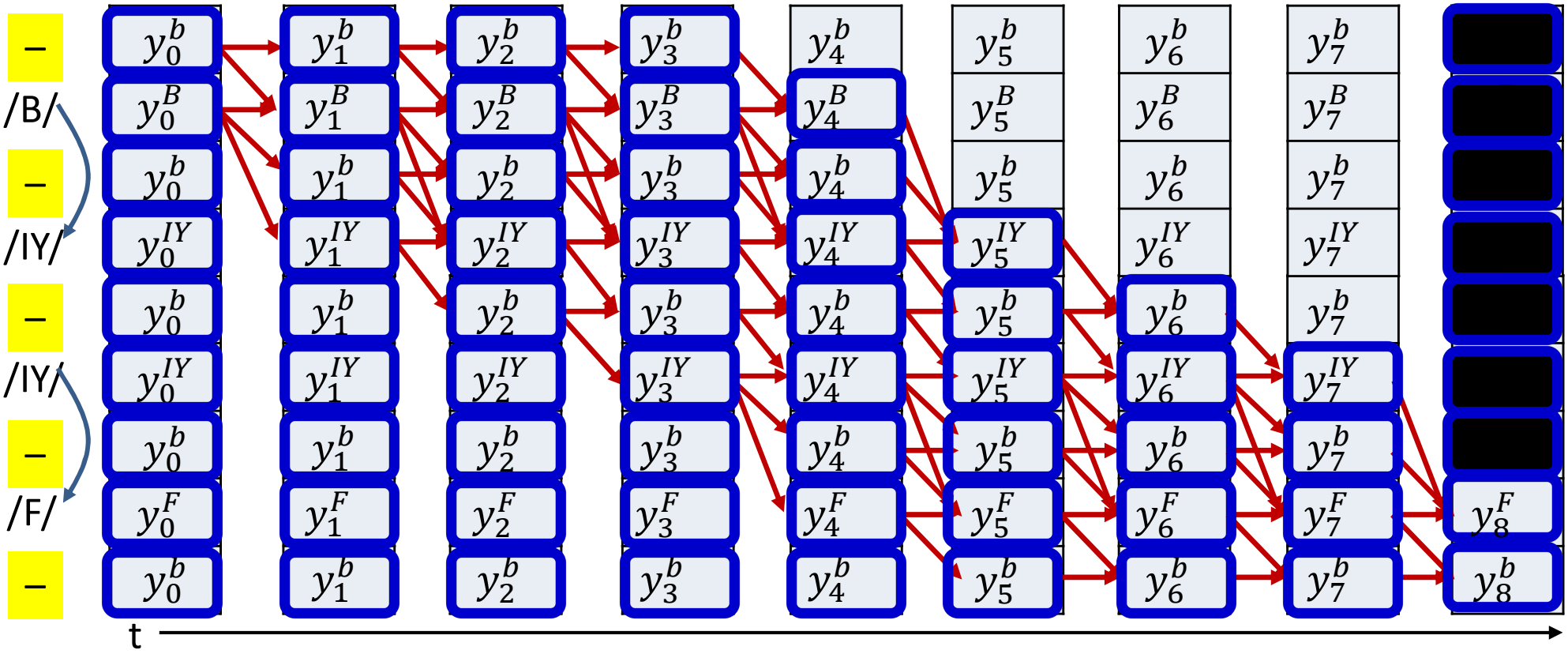
- Initialization:

$$\hat{\beta}(T-1,2K-1) = y_{T-1}^{b}; \hat{\beta}(T-1,2K-2) = y_{T-1}^{S(2K-1)}$$

$$\hat{\beta}(T-1,r) = 0 \quad r < 2K-2$$

$S(k)$ refers to the *extended* sequence with blanks included

Modified Backward Algorithm



- Iteration:

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

  - If $S(r) = " - "$ or $S(r) = S(r+2)$

$$\hat{\beta}(t,r) = y_t^{S(r)}\left(\hat{\beta}(t+1,r) + \hat{\beta}(t+1,r+1) + \hat{\beta}(t+1,r+2)\right)$$

  - Otherwise

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

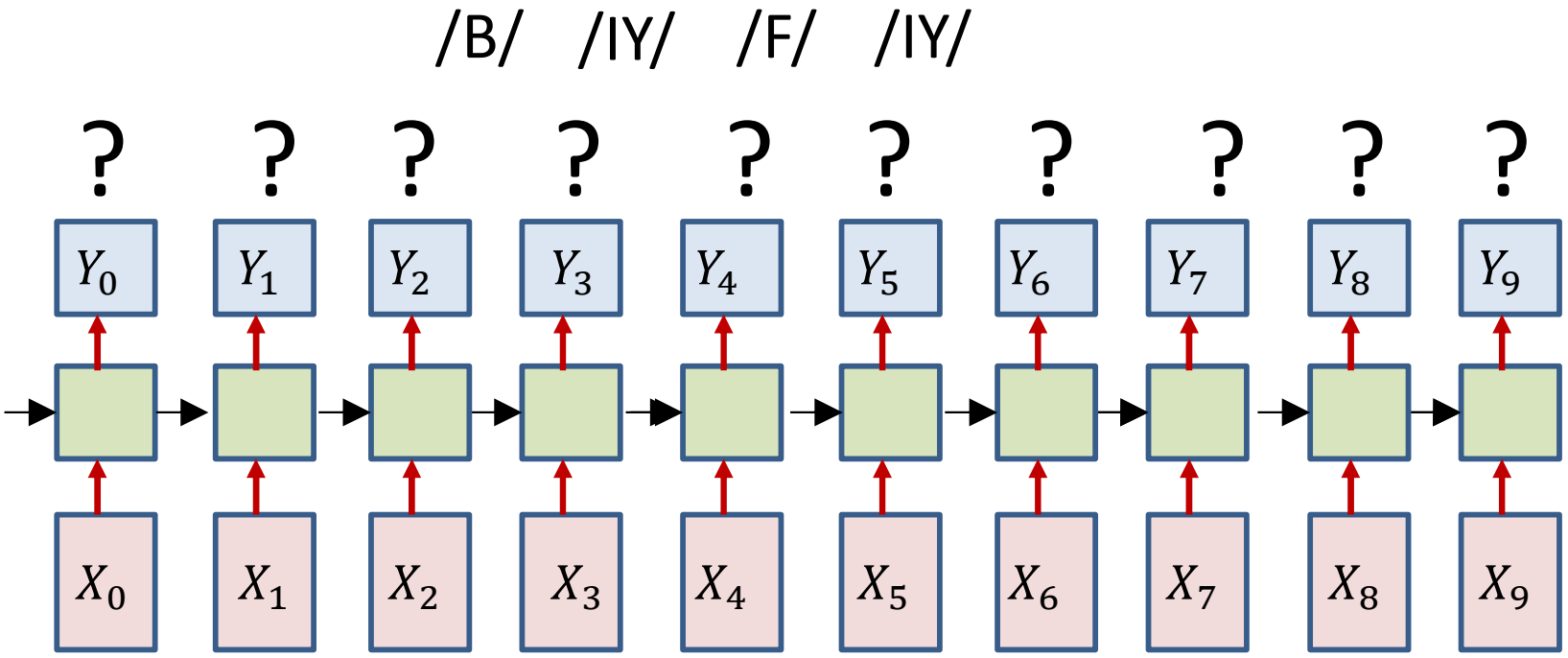
- $\beta(t,r) = \hat{\beta}(t,r)/y_t^{S(r)}$

The rest of the computation

- Posteriors and derivatives are computed exactly as before

- But using the extended graphs with blanks

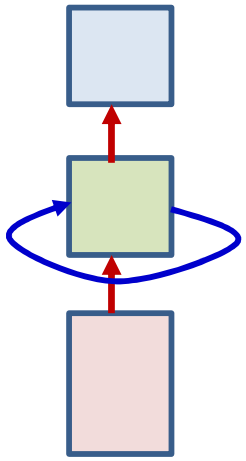
Overall training procedure for Seq2Seq with blanks



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

Overall training procedure

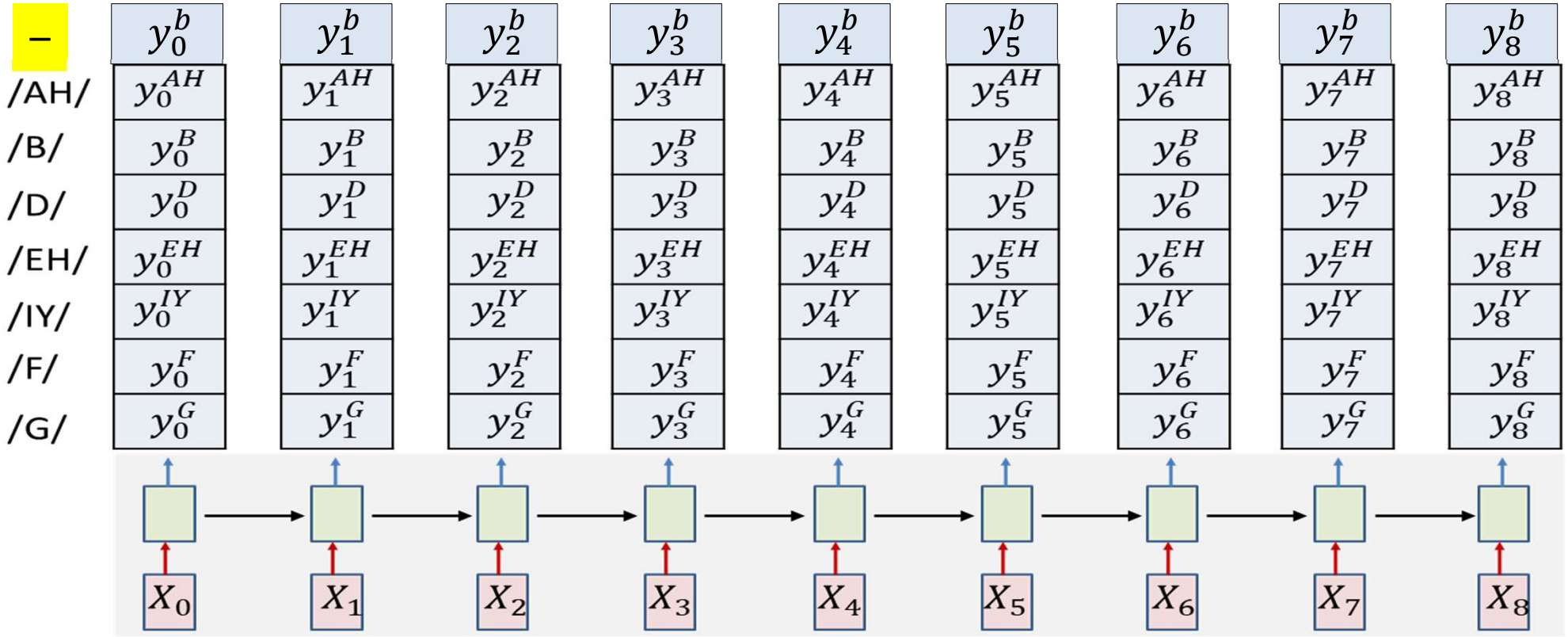
- **Step 1**: Setup the network
  - Typically many-layered LSTM



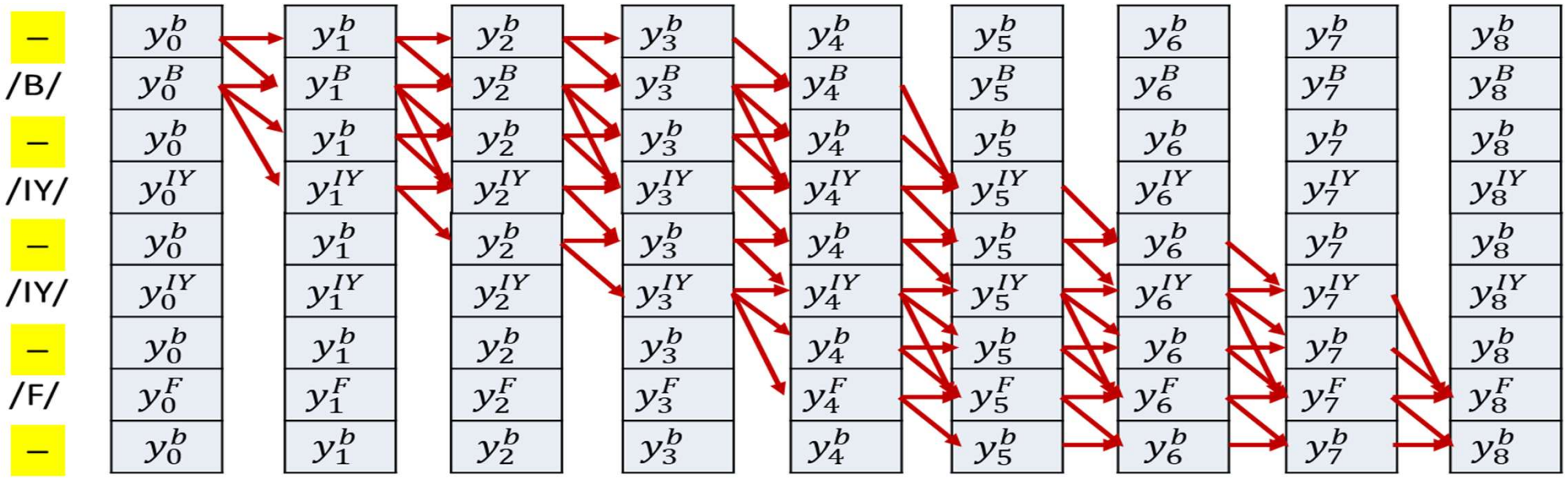
- **Step 2**: Initialize all parameters of the network
  - Include a “blank” symbol in vocabulary

Overall Training: Forward pass

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

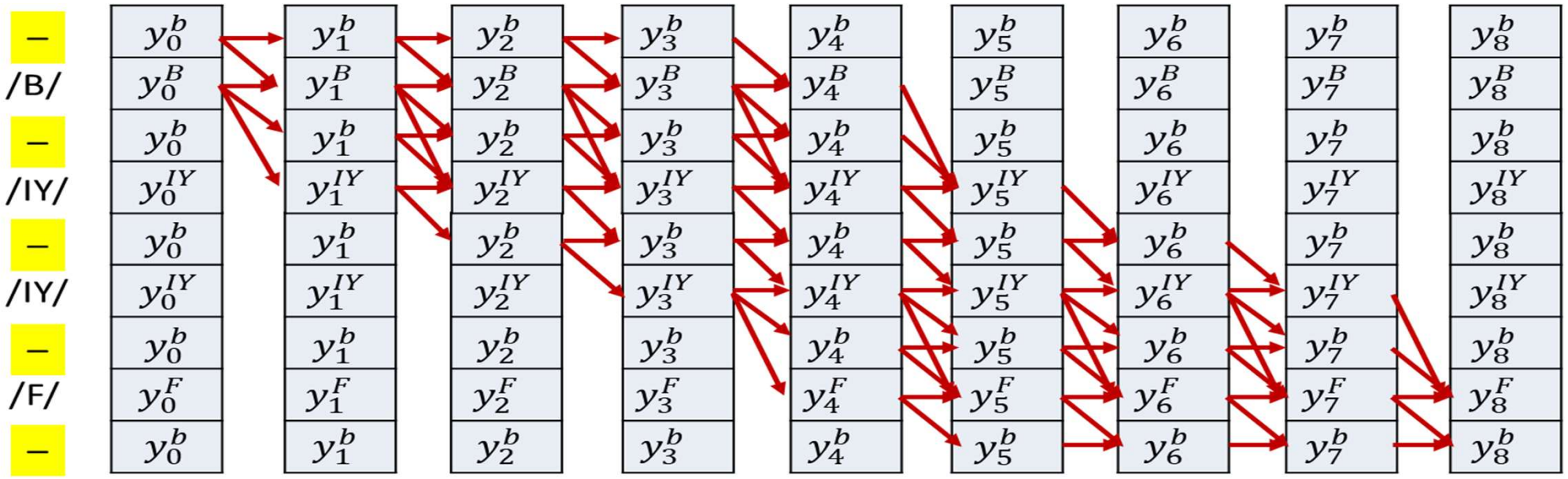


Overall training: Backward pass



- Foreach 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. Use appropriate connections if blanks are included

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 using the modified forward-backward equations. Compute a posteriori probabilities $\gamma(t,r)$ from them
  - **Step 6:** Compute derivative of divergence $\nabla_{Y_t} DIV$ for each Y_t

Overall training: Backward pass

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

$$\nabla_{Y_t} DIV = \left[\frac{dDIV}{dy_t^0} \quad \frac{dDIV}{dy_t^1} \quad \ldots \quad \frac{dDIV}{dy_t^{L-1}}\right]$$

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

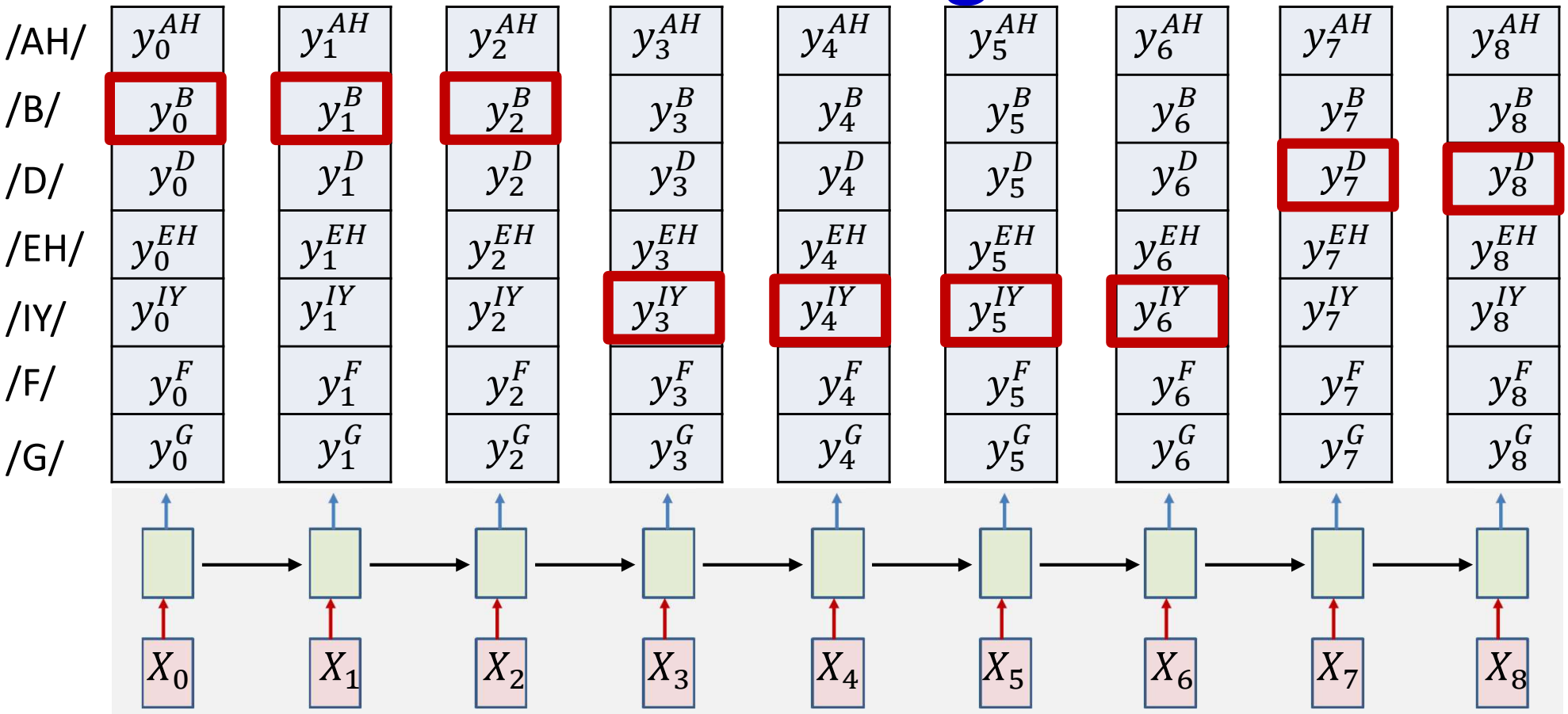
- **Step 7**: Backpropagate $\frac{dDIV}{dy_t^l}$ and aggregate derivatives over minibatch and update parameters

CTC: Connectionist Temporal Classification

- The overall framework we saw is referred to as CTC

- Applies to models that output order-aligned, but time-asynchronous outputs

Returning to an old problem: Decoding



- The greedy decode computes its output by finding the most likely symbol at each time and merging repetitions in the sequence
- This is in fact a *suboptimal* decode that actually finds the most likely *time-synchronous* output sequence
  - Which is not necessarily the most likely *order-synchronous* sequence

Greedy decodes are suboptimal

- Consider the following candidate decodes
  - R R – E E D (RED, 0.7)
  - R R – – E D (RED, 0.68)
  - R R E E E D (RED, 0.69)
  - T T E E E D (TED, 0.71)
  - T T – E E D (TED, 0.3)
  - T T – – E D (TED, 0.29)
- A greedy decode picks the most likely output: TED
- A decode that considers the sum of all alignments of the same final output will select RED
- Which is more reasonable?

Greedy decodes are suboptimal

- Consider the following candidate decodes
  - R R – E E D (RED, 0.7)
  - R R – – E D (RED, 0.68)
  - R R E E E D (RED, 0.69)
  - T T E E E D (TED, 0.71)
  - T T – E E D (TED, 0.3)
  - T T – – E D (TED, 0.29)
- A greedy decode picks the most likely output: TED
- A decode that considers the sum of all alignments of the same final output will select RED
- Which is more reasonable?
- *And yet, remarkably, greedy decoding can be surprisingly effective, when using decoding with blanks*

What a CTC system outputs

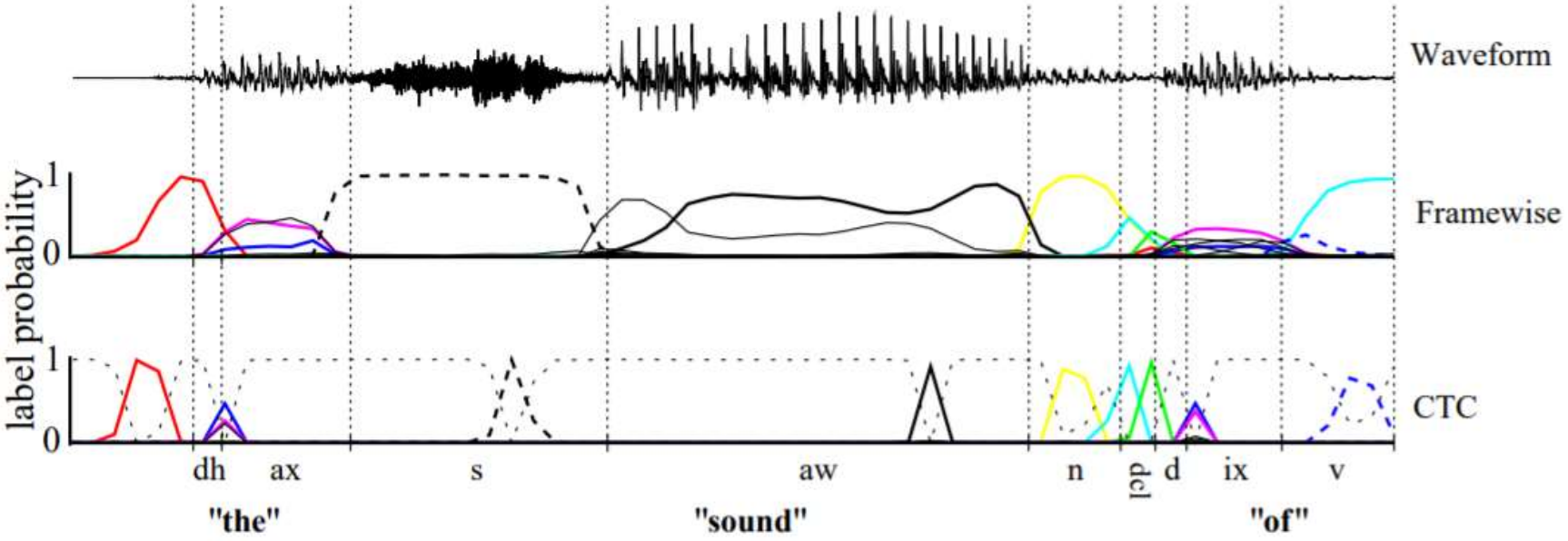


Figure 1. **Framewise and CTC networks classifying a speech signal**. The shaded lines are the output activations, corresponding to the probabilities of observing phonemes at particular times. The CTC network predicts only the sequence of phonemes (typically as a series of spikes, separated by 'blanks', or null predictions), while the framewise network attempts to align them with the manual segmentation (vertical lines). The framewise network receives an error for misaligning the segment boundaries, even if it predicts the correct phoneme (e.g. 'dh'). When one phoneme always occurs beside another (e.g. the closure 'dcl' with the stop 'd'), CTC tends to predict them together in a double spike. The choice of labelling can be read directly from the CTC outputs (follow the spikes), whereas the predictions of the framewise network must be post-processed before use.

- Ref: Graves
- Symbol outputs peak at the ends of the sounds
  - Typical output: - - R - - - E - - -D
  - Model output naturally eliminates alignment ambiguities
- But this is still suboptimal..

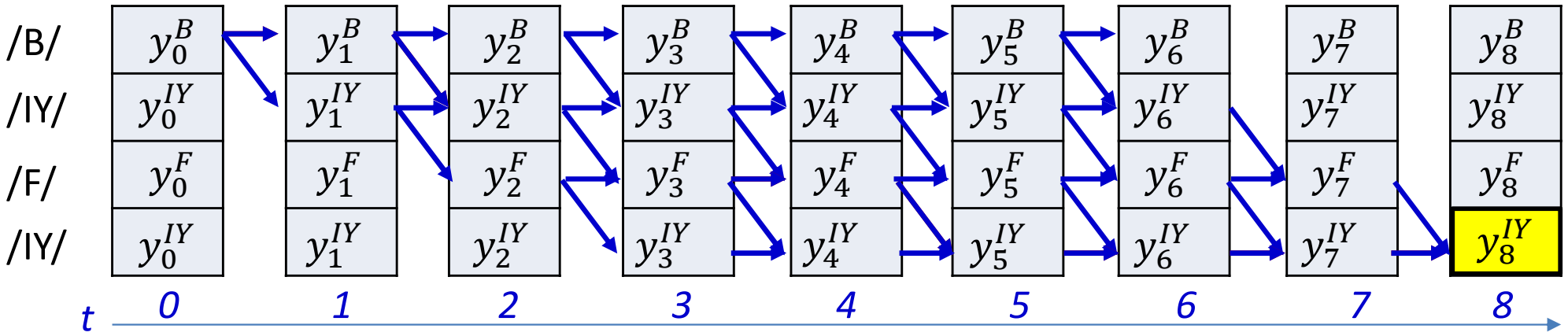
Actual objective of decoding

- Want to find most likely order-aligned symbol sequence
  - **R E D**
  - What greedy decode finds: most likely time synchronous symbol sequence
    - **– /R/ /R/ – – /EH//EH//D/**
    - Which must be compressed

- Find the order-aligned symbol sequence $\boldsymbol{S} = S_0, \ldots, S_{K-1}$, given an input $\boldsymbol{X} = X_0, \ldots, X_{T-1}$, that is most likely

$$= \underset{\boldsymbol{S}}{\operatorname{argmax}} P(S_0, \ldots, S_{K-1} | \boldsymbol{X})$$

Recall: The forward probability $\alpha(t, r)$



$$\alpha_{S_0..S_{K-1}}(T-1, K-1) = P(S_0..S_{K-1}|\mathbf{X})$$

- The probability of the entire symbol sequence is the alpha at the bottom right node

Actual decoding objective

- Find the most likely (asynchronous) symbol sequence

$$\hat{\mathbf{S}} = \underset{\mathbf{S}}{\operatorname{argmax}}\, \alpha_{\mathbf{S}}(S_{K-1}, T-1)$$

Poll 5

- @1412, @1413

Poll 5

The actual objective of decoding is to identify the compressed/unaligned sequence that has the highest probability given the input

- **True**
- False

This is the same as finding the compressed sequence with the highest forward probability (alpha) for aligning the final symbol in the sequence to the final input

- **True**
- False

Actual decoding objective

- Find the most likely (asynchronous) symbol sequence

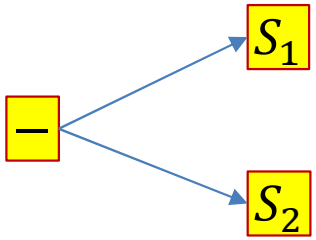
$$\hat{\mathbf{S}} = \underset{\mathbf{S}}{\operatorname{argmax}}\, \alpha_{\mathbf{S}}(S_{K-1}, T-1)$$

- Unfortunately, explicit computation of this will require evaluate of an exponential number of symbol sequences

- Solution: Organize all possible symbol sequences as a (semi)tree

Hypothesis semi-tree

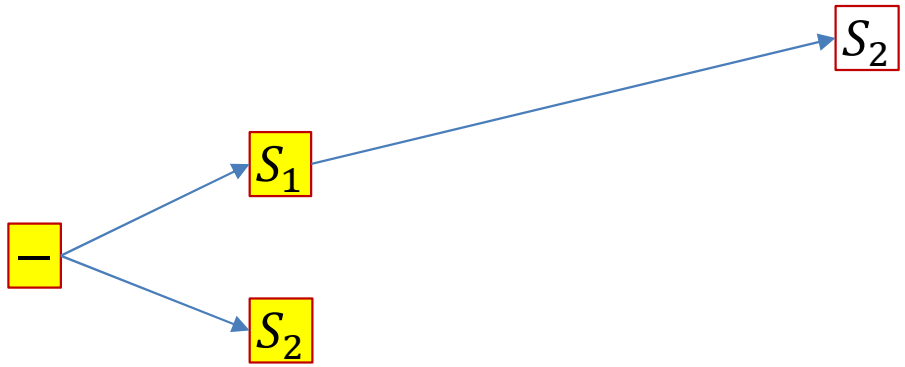
Highlighted boxes represent possible symbols for first frame



- The semi tree of hypotheses (assuming only 3 symbols in the vocabulary)
- Every symbol connects to every symbol other than itself
  - It also connects to a blank, which connects to every symbol including itself
- The simple structure repeats recursively
- Each node represents a unique (partial) symbol sequence!

Hypothesis semi-tree

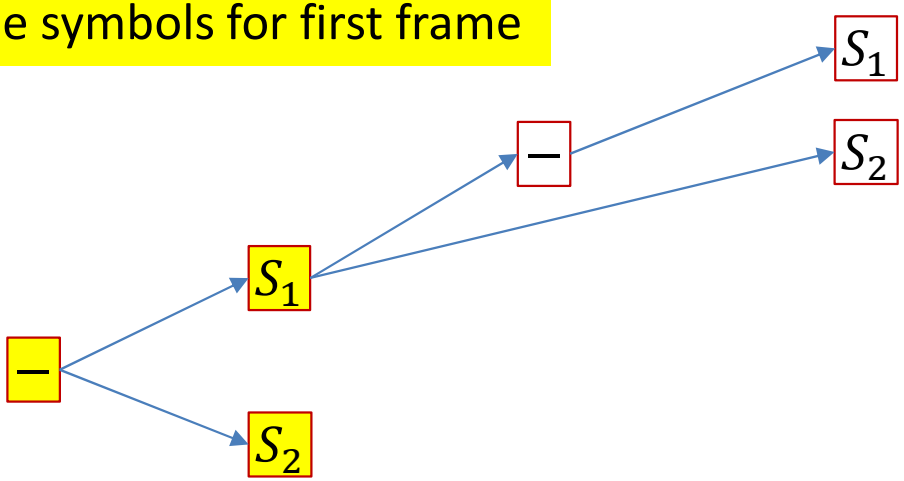
Highlighted boxes represent possible symbols for first frame



- The semi tree of hypotheses (assuming only 3 symbols in the vocabulary)
- Every symbol connects to every symbol other than itself
  - It also connects to a blank, which connects to every symbol including itself
- The simple structure repeats recursively
- Each node represents a unique (partial) symbol sequence!

Hypothesis semi-tree

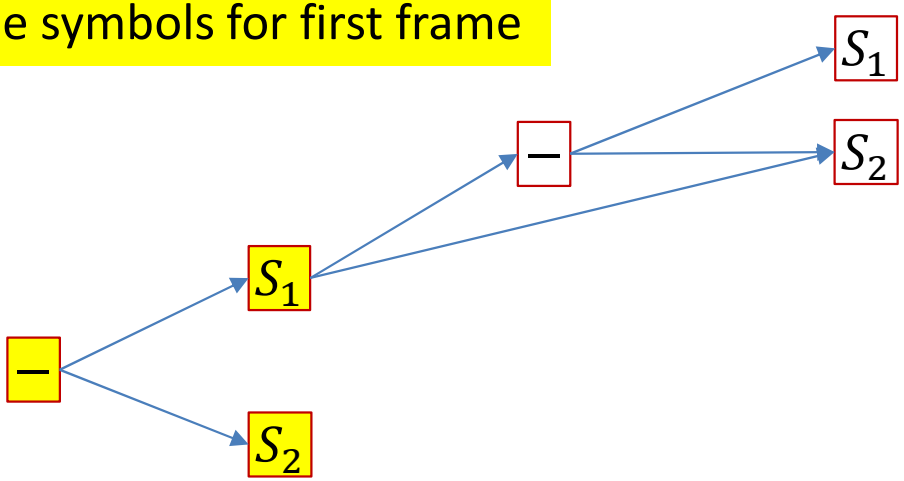
Highlighted boxes represent possible symbols for first frame



- The semi tree of hypotheses (assuming only 3 symbols in the vocabulary)
- Every symbol connects to every symbol other than itself
  - It also connects to a blank, which connects to every symbol including itself
- The simple structure repeats recursively
- Each node represents a unique (partial) symbol sequence!

Hypothesis semi-tree

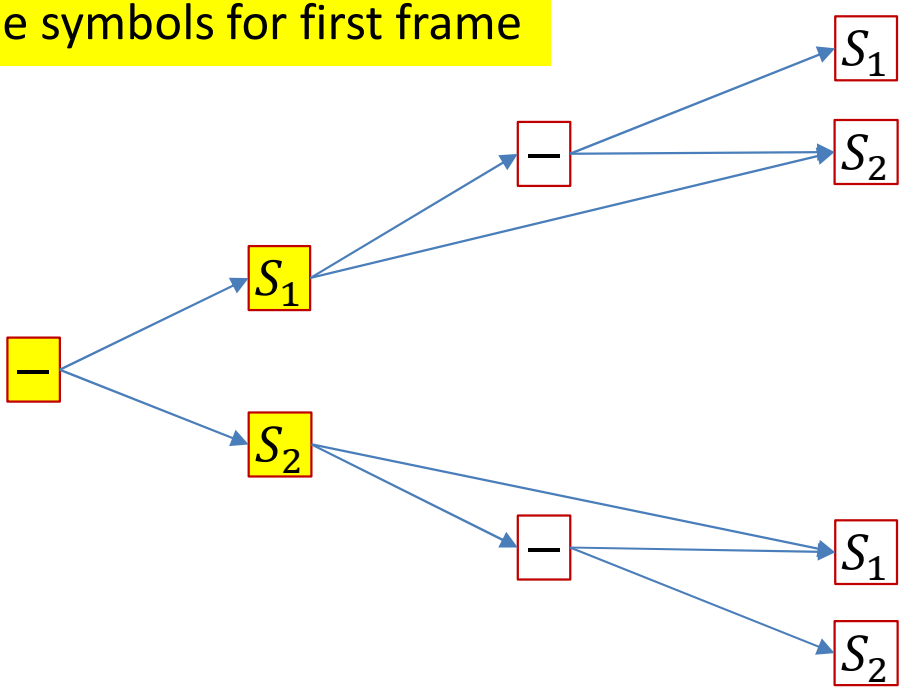
Highlighted boxes represent possible symbols for first frame



- The semi tree of hypotheses (assuming only 3 symbols in the vocabulary)
- Every symbol connects to every symbol other than itself
  - It also connects to a blank, which connects to every symbol including itself
- The simple structure repeats recursively
- Each node represents a unique (partial) symbol sequence!

Hypothesis semi-tree

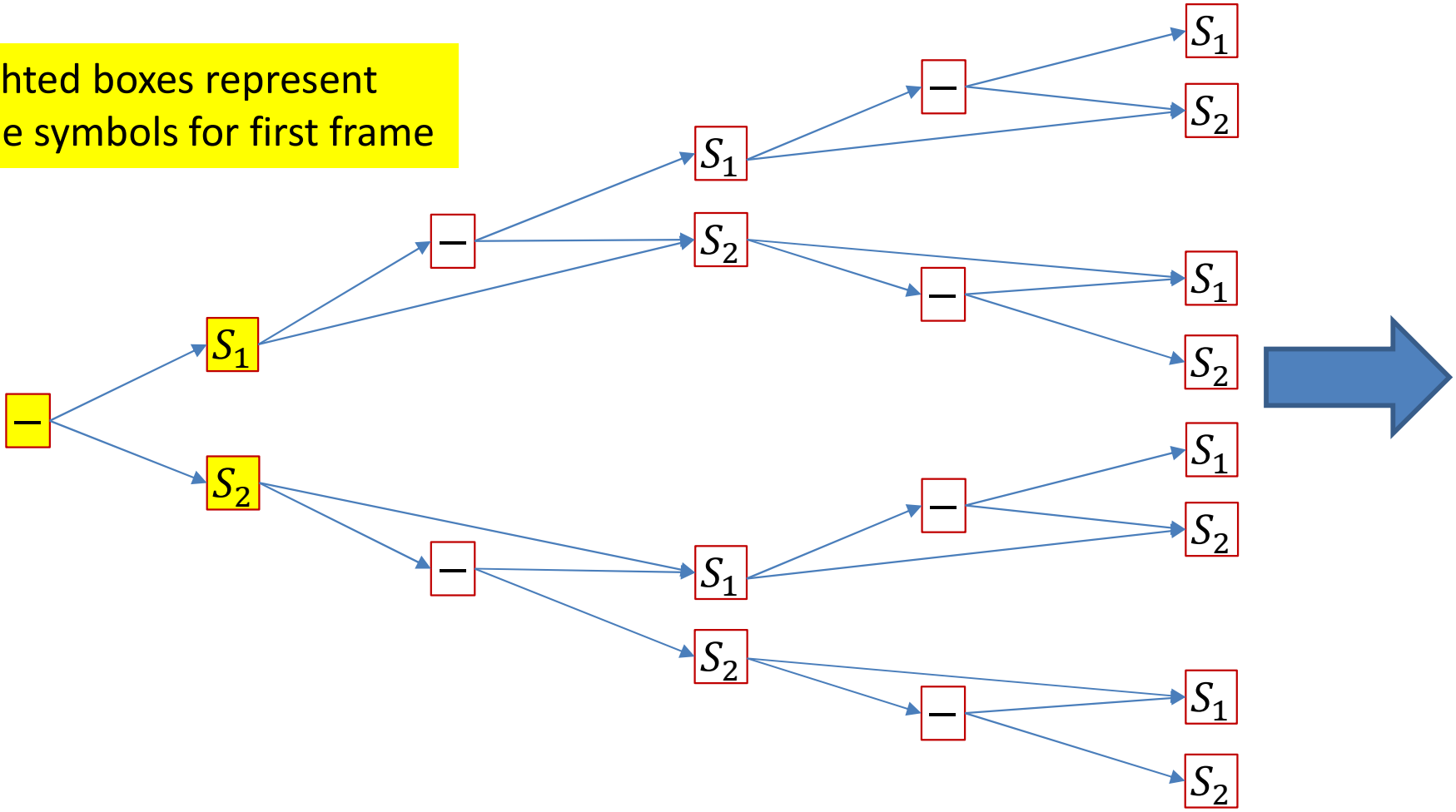
Highlighted boxes represent possible symbols for first frame



- The semi tree of hypotheses (assuming only 3 symbols in the vocabulary)
- Every symbol connects to every symbol other than itself
  - It also connects to a blank, which connects to every symbol including itself
- The simple structure repeats recursively
- Each node represents a unique (partial) symbol sequence!

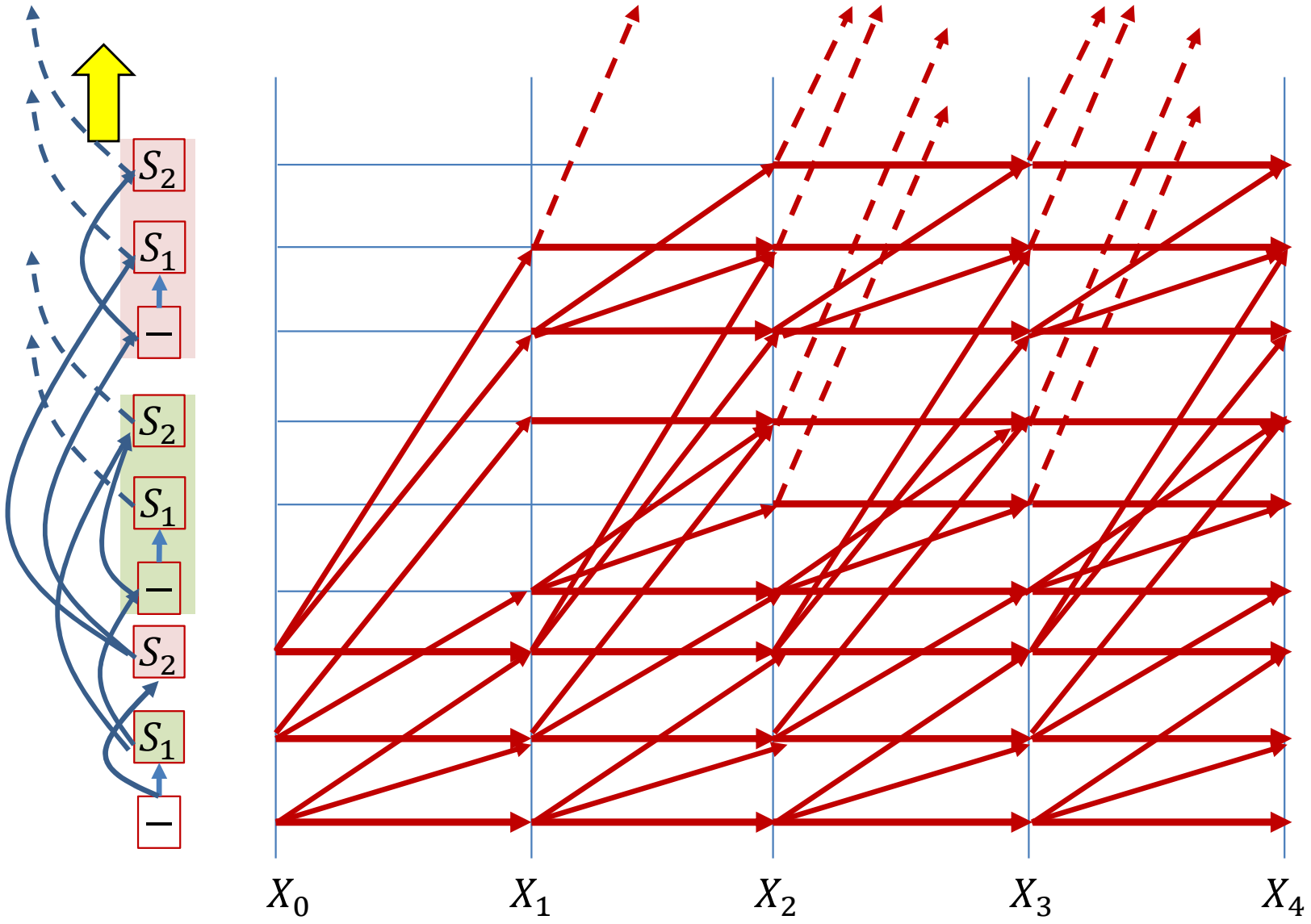
Hypothesis semi-tree

Highlighted boxes represent possible symbols for first frame



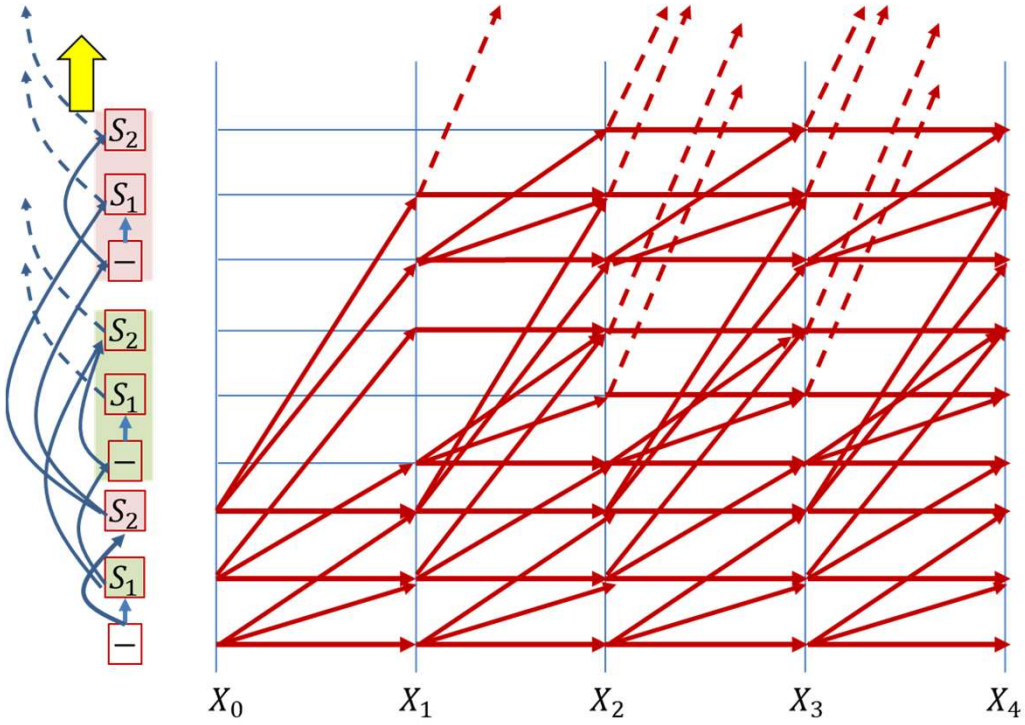
- The semi tree of hypotheses (assuming only 3 symbols in the vocabulary)
- Every symbol connects to every symbol other than itself
  - It also connects to a blank, which connects to every symbol including itself
- The simple structure repeats recursively
- Each node represents a unique (partial) symbol sequence!

The decoding graph for the tree



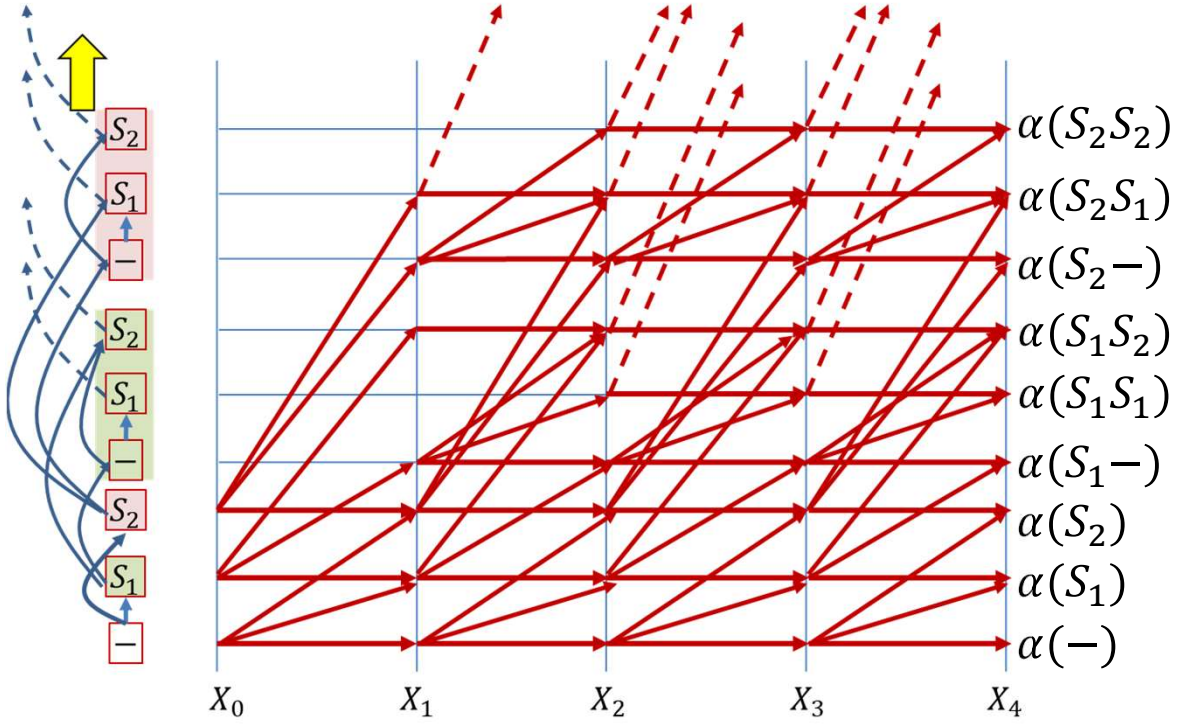
- Graph with more than 2 symbols will be similar but much more cluttered and complicated

The decoding graph for the tree



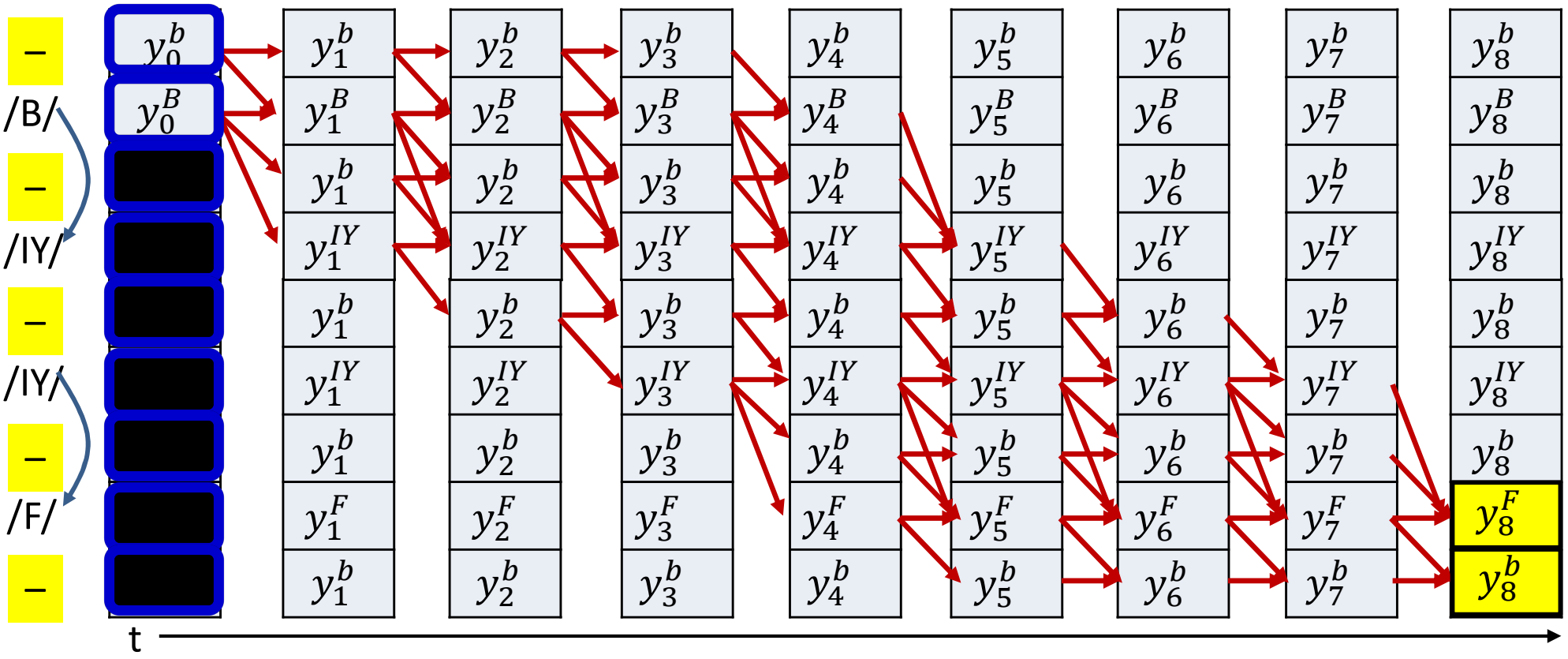
- The figure to the left is the tree, drawn in a vertical line
- The graph is just the tree unrolled over time
  - For a vocabulary of V symbols, every node connects out to V other nodes at the next time
- Every node in the graph represents a unique symbol sequence

The decoding graph for the tree



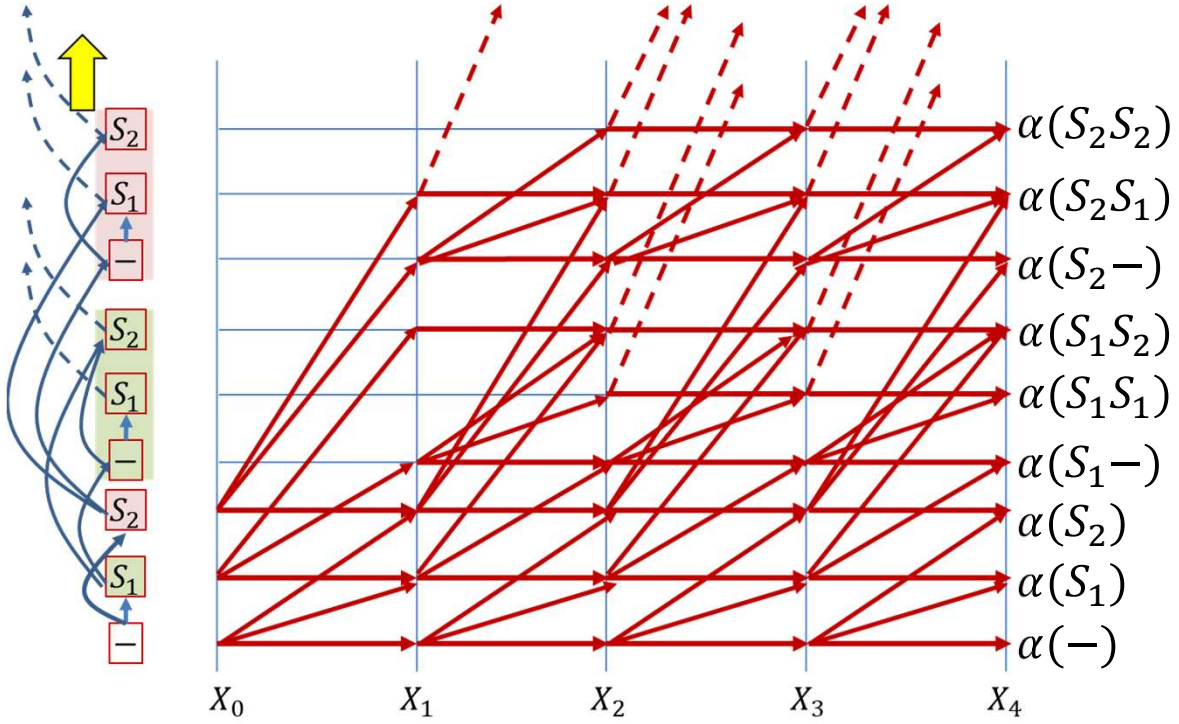
- The forward score $\alpha(r, T)$ at the final time represents the full forward score for a unique symbol sequence (including sequences terminating in blanks)
- Select the symbol sequence with the largest alpha at the final time

Recall: Forward Algorithm



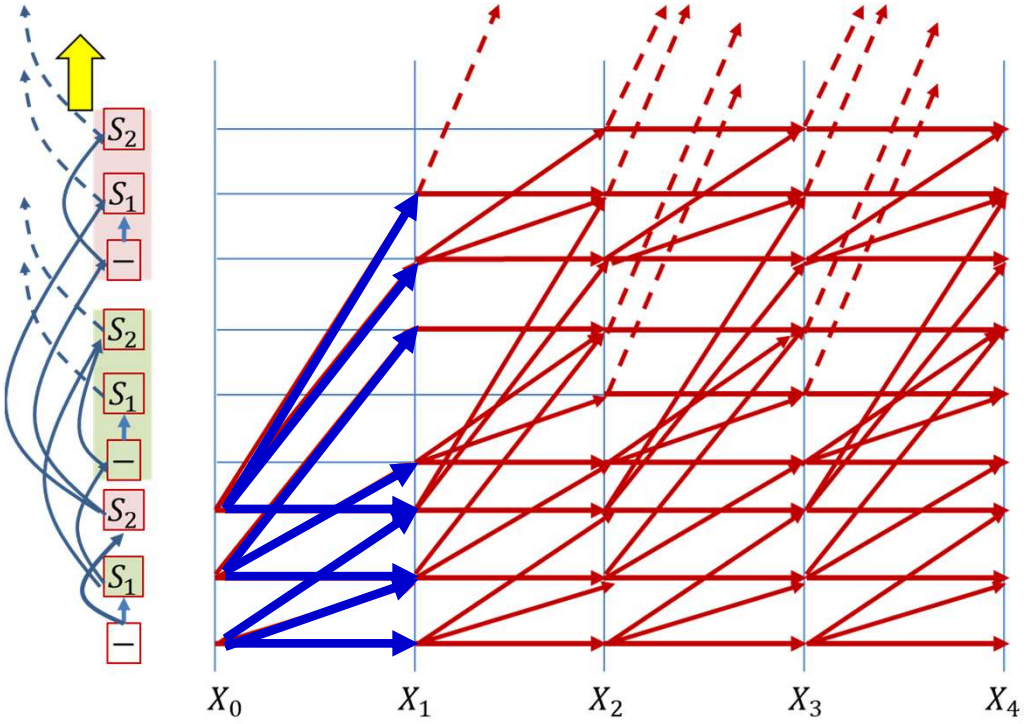
- $P(S_0, \ldots, S_{K-1}|\boldsymbol{X}) = \alpha(T-1, 2K) + \alpha(T-1, 2K+1)$

The decoding graph for the tree



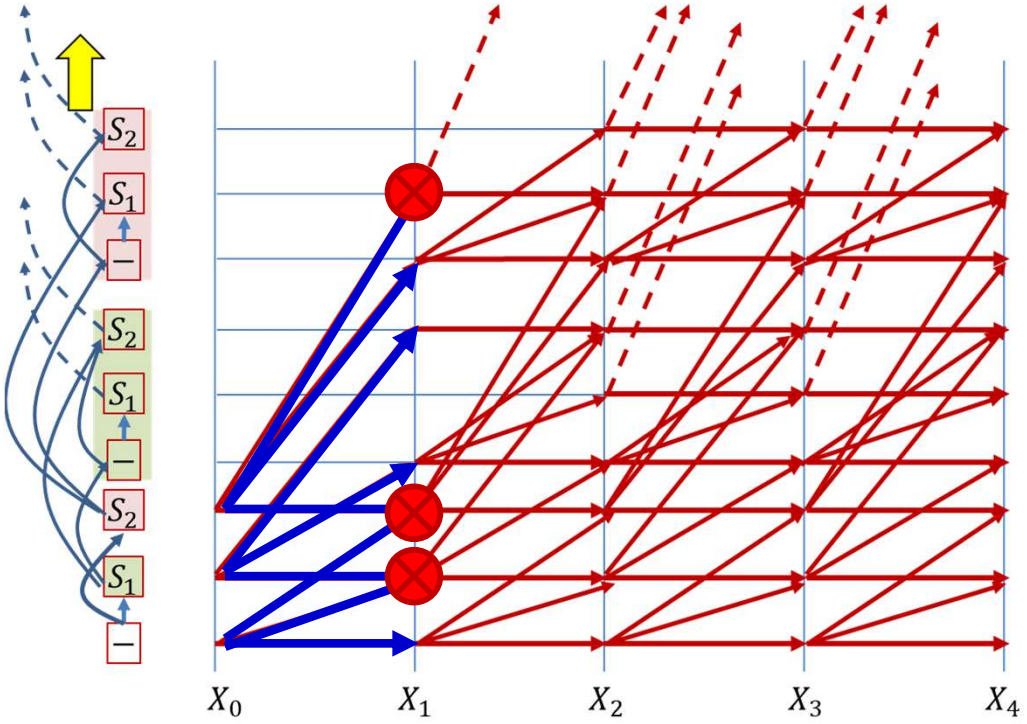
- The forward score $\alpha(r, T)$ at the final time represents the full forward score for a unique symbol sequence (including sequences terminating in blanks)
- Select the symbol sequence with the largest alpha
  - Sequences may two alphas, one for the sequence itself, one for the sequence followed by a blank
  - Add the alphas before selecting the most likely

CTC decoding



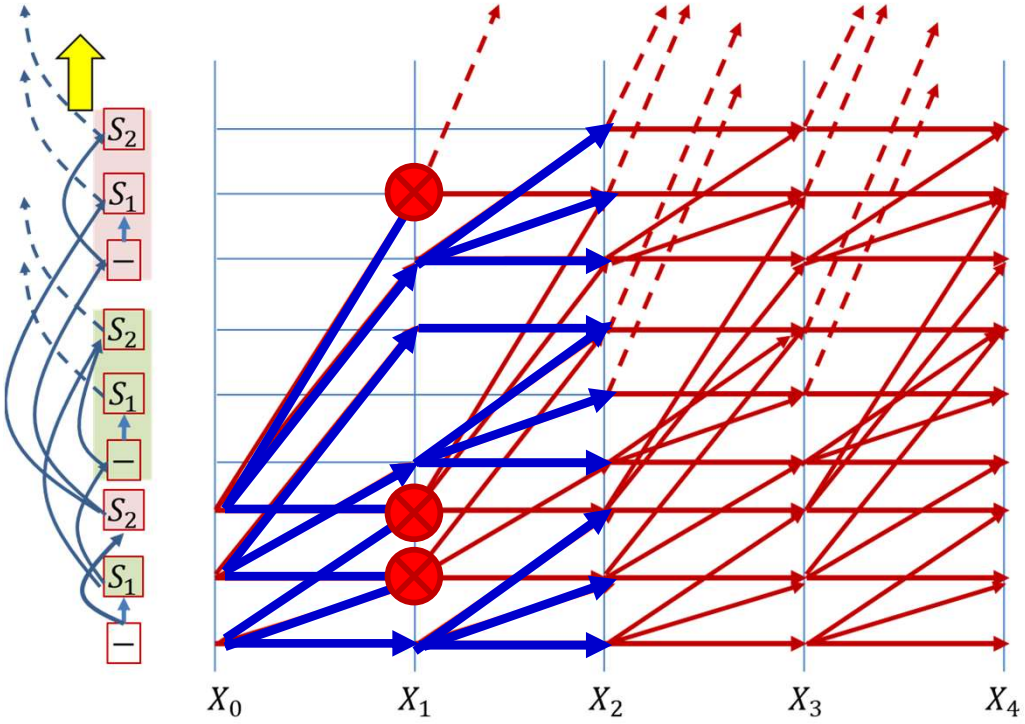
- This is the “theoretically correct” CTC decoder
- In practice, the graph gets exponentially large very quickly
- To prevent this pruning strategies are employed to keep the graph (and computation) manageable
  - This may cause suboptimal decodes, however
  - The fact that CTC scores peak at symbol terminations minimizes the damage due to pruning

CTC decoding



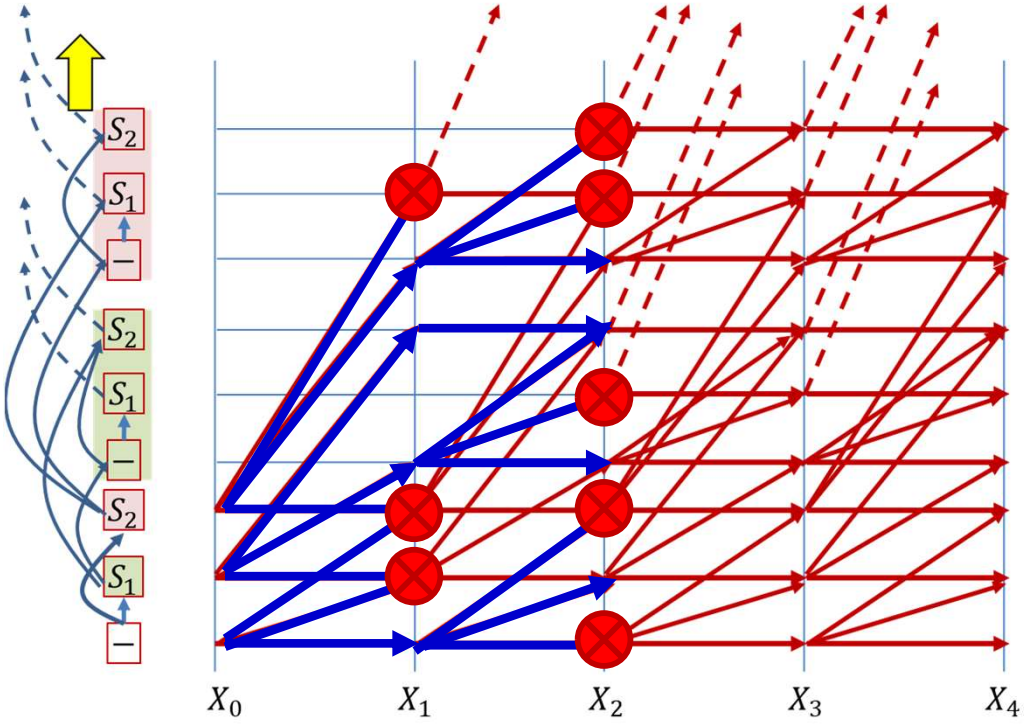
- This is the “theoretically correct” CTC decoder
- In practice, the graph gets exponentially large very quickly
- To prevent this pruning strategies are employed to keep the graph (and computation) manageable
  - This may cause suboptimal decodes, however
  - The fact that CTC scores peak at symbol terminations minimizes the damage due to pruning

CTC decoding



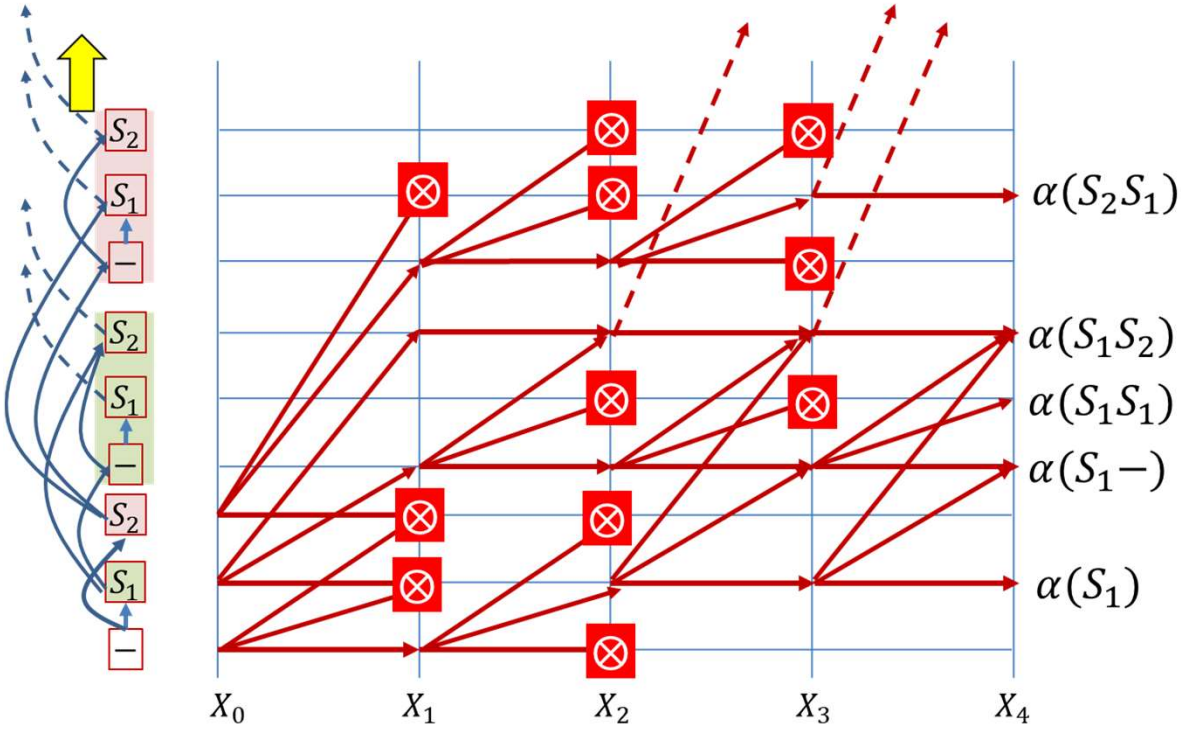
- This is the “theoretically correct” CTC decoder
- In practice, the graph gets exponentially large very quickly
- To prevent this pruning strategies are employed to keep the graph (and computation) manageable
  - This may cause suboptimal decodes, however
  - The fact that CTC scores peak at symbol terminations minimizes the damage due to pruning

CTC decoding



- This is the “theoretically correct” CTC decoder
- In practice, the graph gets exponentially large very quickly
- To prevent this pruning strategies are employed to keep the graph (and computation) manageable
  - This may cause suboptimal decodes, however
  - The fact that CTC scores peak at symbol terminations minimizes the damage due to pruning

CTC decoding



- This is the "theoretically correct" CTC decoder
- In practice, the graph gets exponentially large very quickly
- To prevent this pruning strategies are employed to keep the graph (and computation) manageable
  - This may cause suboptimal decodes, however
  - The fact that CTC scores peak at symbol terminations minimizes the damage due to pruning

Beamsearch Pseudocode Notes

- Retaining separate lists of paths and pathscores for paths terminating in blanks, and those terminating in valid symbols
  - Since blanks are special
  - Do not explicitly represent blanks in the partial decode strings

- Pseudocode takes liberties (particularly w.r.t null strings)
  - I.e. you must be careful if you convert this to code

- Key
  - **`PathScore`** : array of scores for paths ending with symbols
  - **`BlankPathScore`** : array of scores for paths ending with blanks
  - **`SymbolSet`** : A list of symbols *not* including the blank

Story so far: CTC models

- Sequence-to-sequence networks which irregularly produce output symbols can be trained by
  - Iteratively aligning the target output to the input and time-synchronous training
  - Optimizing the expected error over *all* possible alignments: CTC training
- Distinct repetition of symbols can be disambiguated from repetitions representing the extended output of a single symbol by the introduction of blanks
- Decoding the models can be performed by
  - Best-path decoding, i.e. Viterbi decoding
  - Optimal CTC decoding based on the application of the forward algorithm to a tree-structured representation of all possible output strings

CTC caveats

- The “blank” structure (with concurrent modifications to the forward-backward equations) is only one way to deal with the problem of repeating symbols

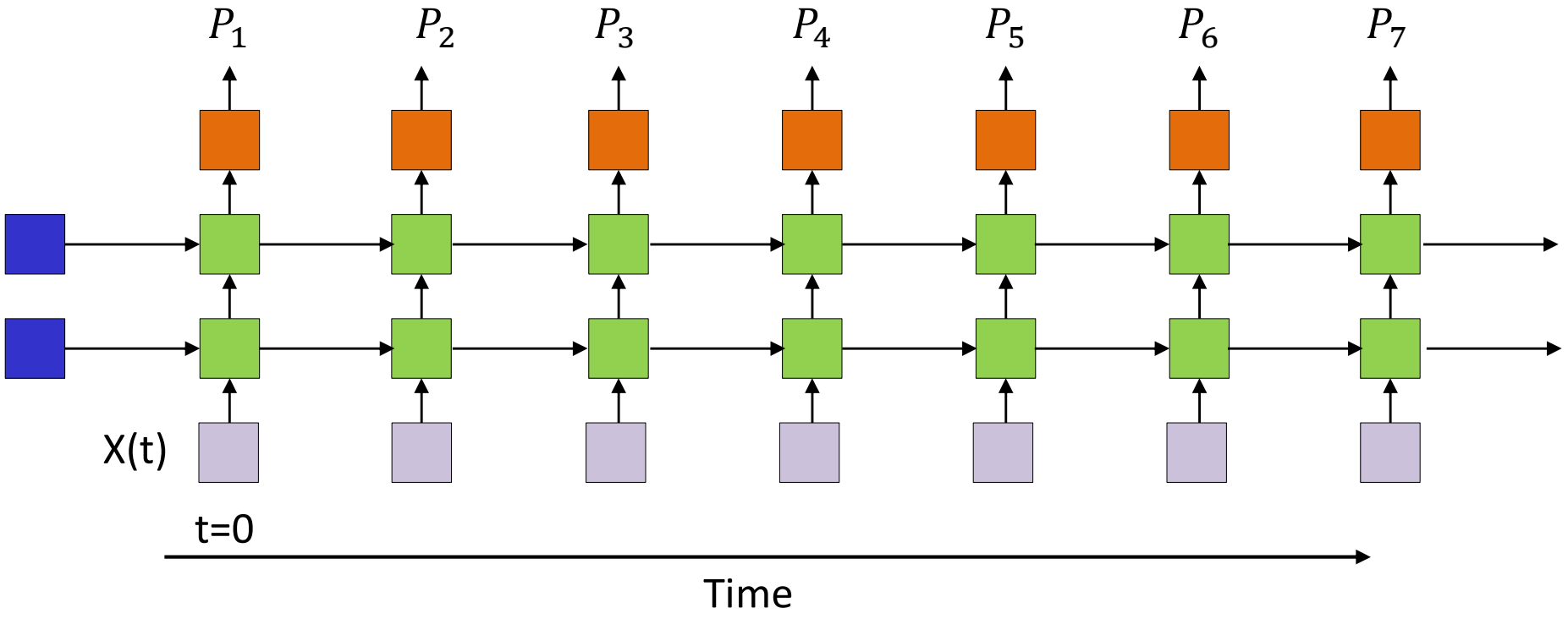
- Possible variants:
  - Symbols partitioned into two or more sequential subunits
    - No blanks are required, since subunits must be visited in order
  - Symbol-specific blanks
    - Doubles the “vocabulary”
  - CTC can use *bidirectional* recurrent nets
    - And frequently does
  - Other variants possible..

Most common CTC applications

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

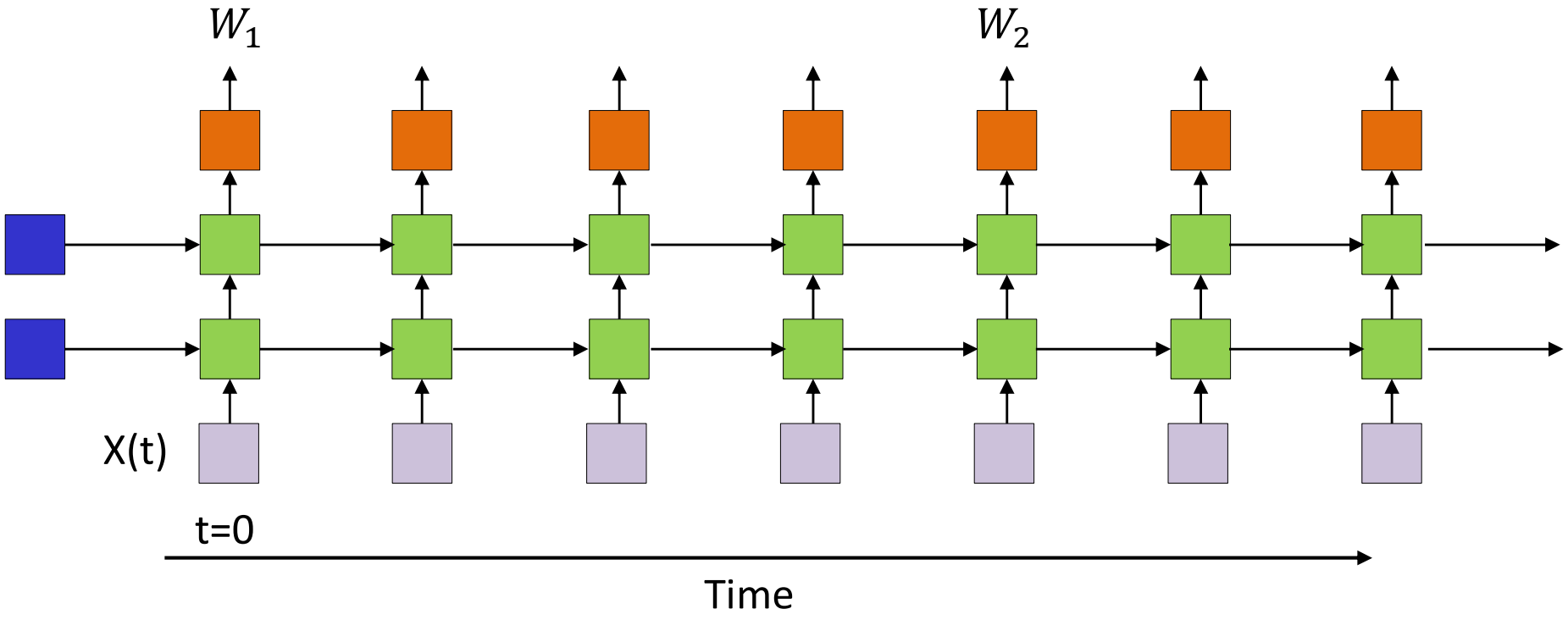
- Handwriting recognition

Speech recognition using Recurrent Nets



- Recurrent neural networks (with LSTMs) can be used to perform speech recognition
  - Input: Sequences of audio feature vectors
  - Output: Phonetic label of each vector

Speech recognition using Recurrent Nets



- Alternative: Directly output phoneme, character or word sequence

Next up: Attention models