Deep Learning

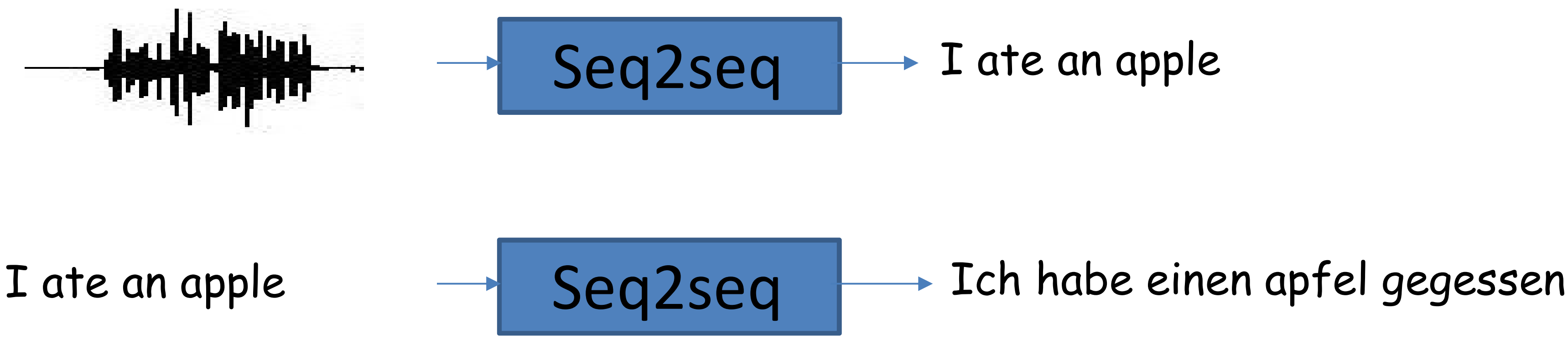
Sequence to Sequence models: Connectionist Temporal Classification

Sequence-to-sequence modelling

- Problem:
  - A sequence $X_1 \dots X_N$ goes in
  - A different sequence $Y_1 \dots Y_M$ 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

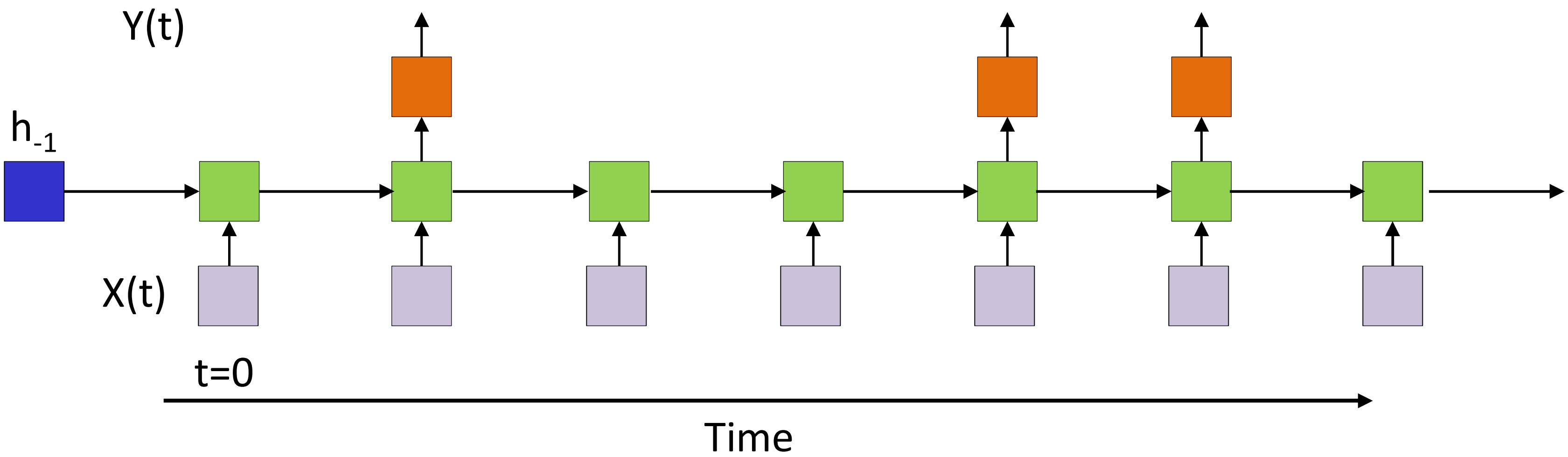
- In general $N \neq M$
  - No synchrony between X and Y.

Sequence to sequence



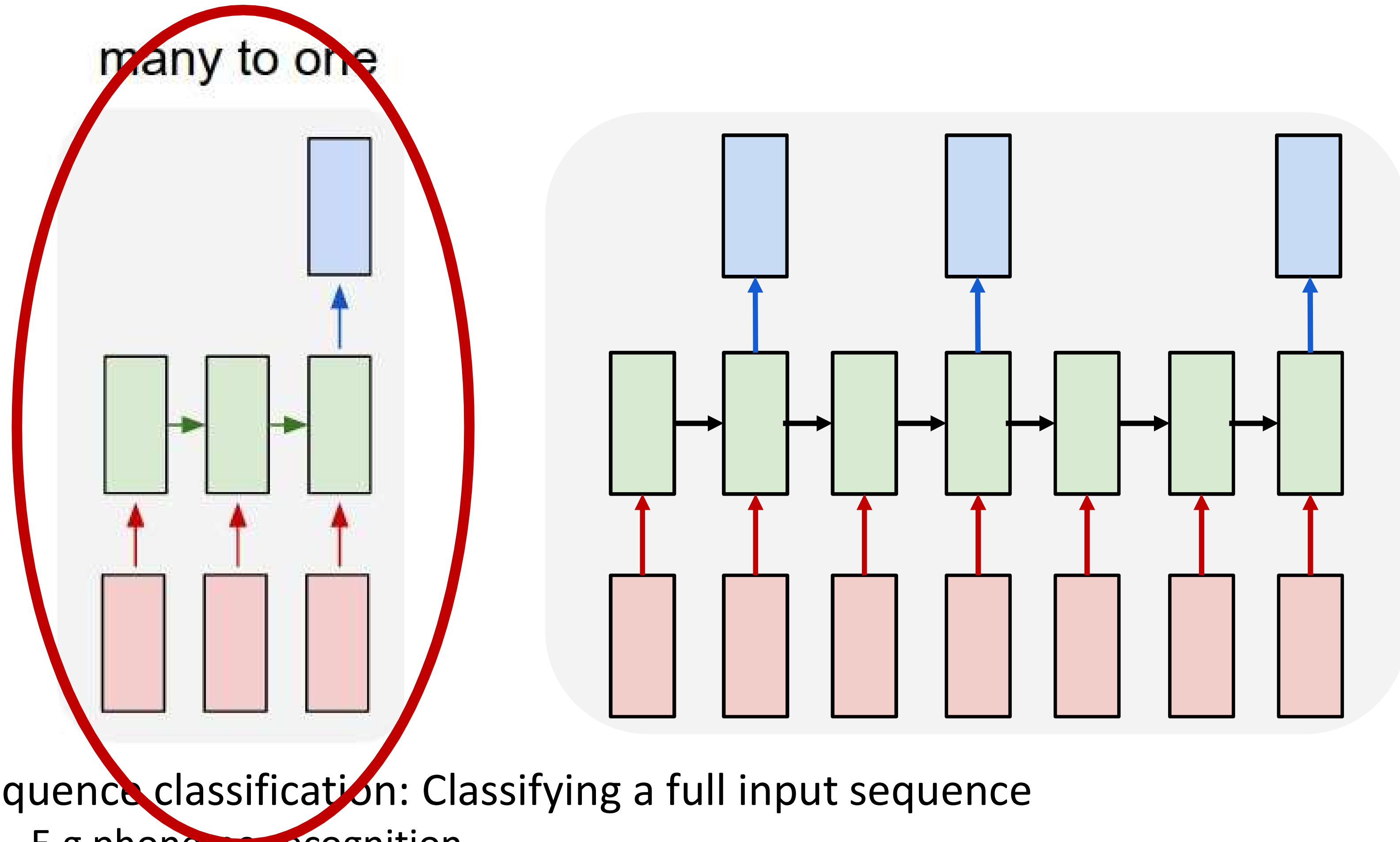
- Sequence goes in, sequence comes out
- No notion of "synchrony" between input and output
  - May even not have a notion of "alignment"
    - E.g. "I ate an apple" → "Ich habe einen apfel gegessen"

Case 1: With alignment



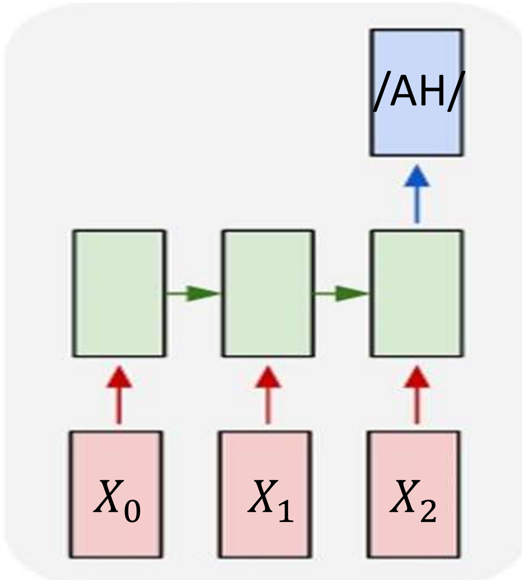
- The input and output sequences happen in the same order
  - Although they may be asynchronous
  - E.g. Speech recognition
    - The input speech corresponds to the phoneme sequence output

Variants on recurrent nets



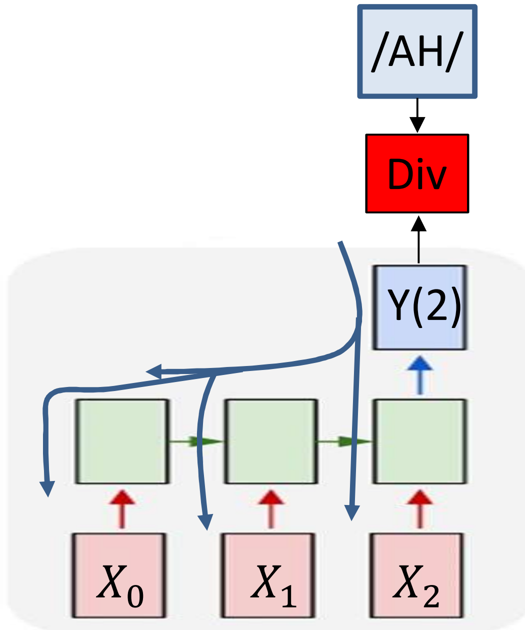
- Sequence classification: Classifying a full input sequence
  - E.g phoneme recognition
- Order synchronous , time asynchronous sequence-to-sequence generation
  - E.g. speech recognition
  - Exact location of output is unknown a priori

Basic model



- Sequence of inputs produces a single output

Training

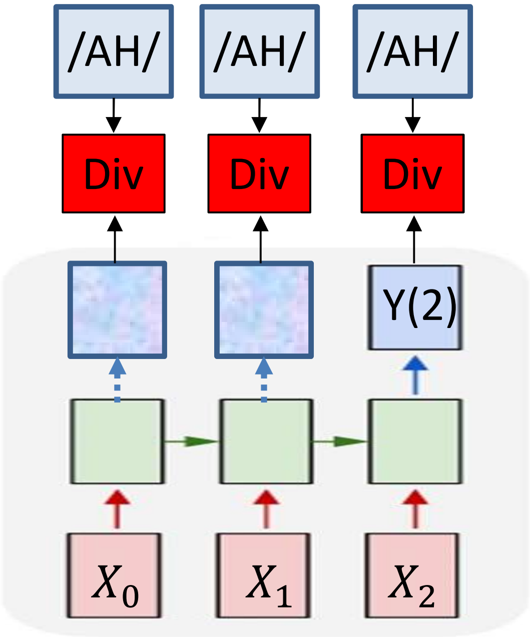


- The Divergence is only defined at the final input
  - $DIV(Y_{target}, Y) = Xent(Y(T), Phoneme)$
- This divergence must propagate through the net to update all parameters
- Ignores outputs at intermediate steps

Training

Fix: Use these outputs too.

These too must ideally point to the correct phoneme



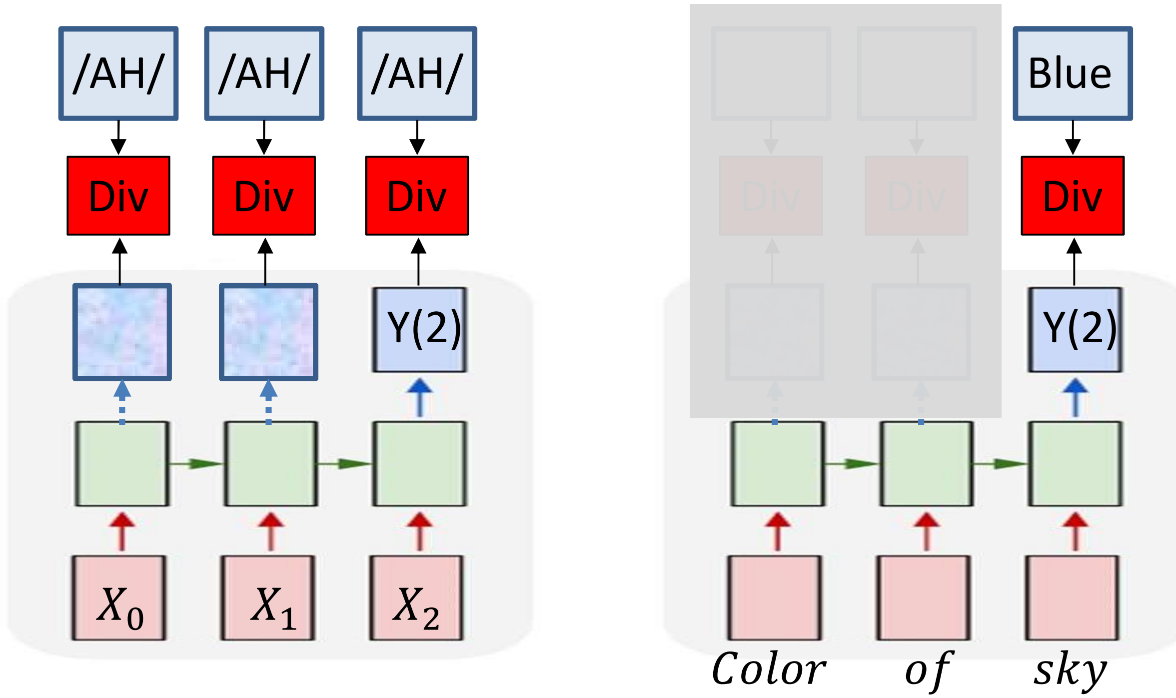
- Exploiting the untagged inputs: assume the same output for the entire input
- Define the divergence everywhere

$$DIV\left(Y_{target}, Y\right) = \sum_{t} w_t Xent(Y(t), Phoneme)$$

Training

Fix: Use these outputs too.

These too must ideally point to the correct phoneme

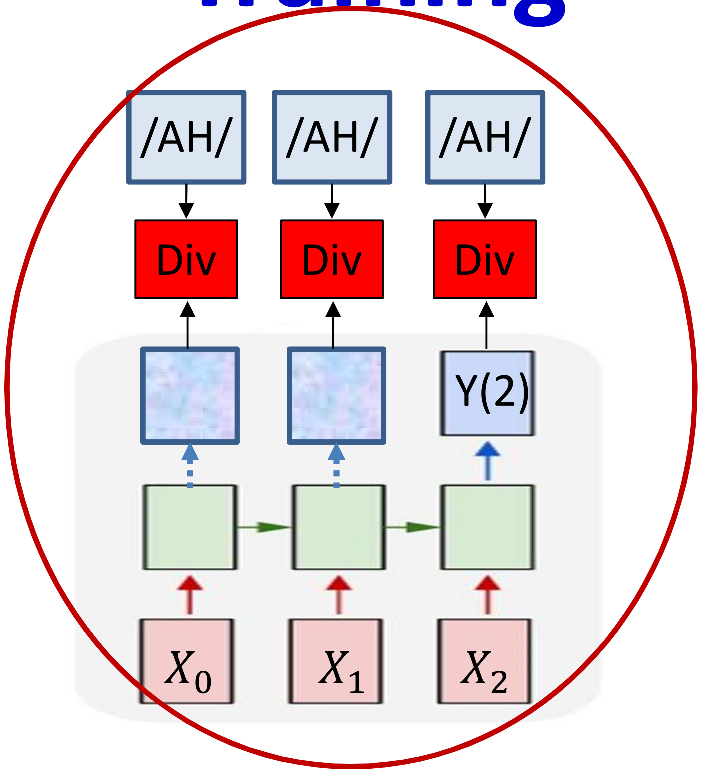


- Define the divergence everywhere

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

- 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

Training



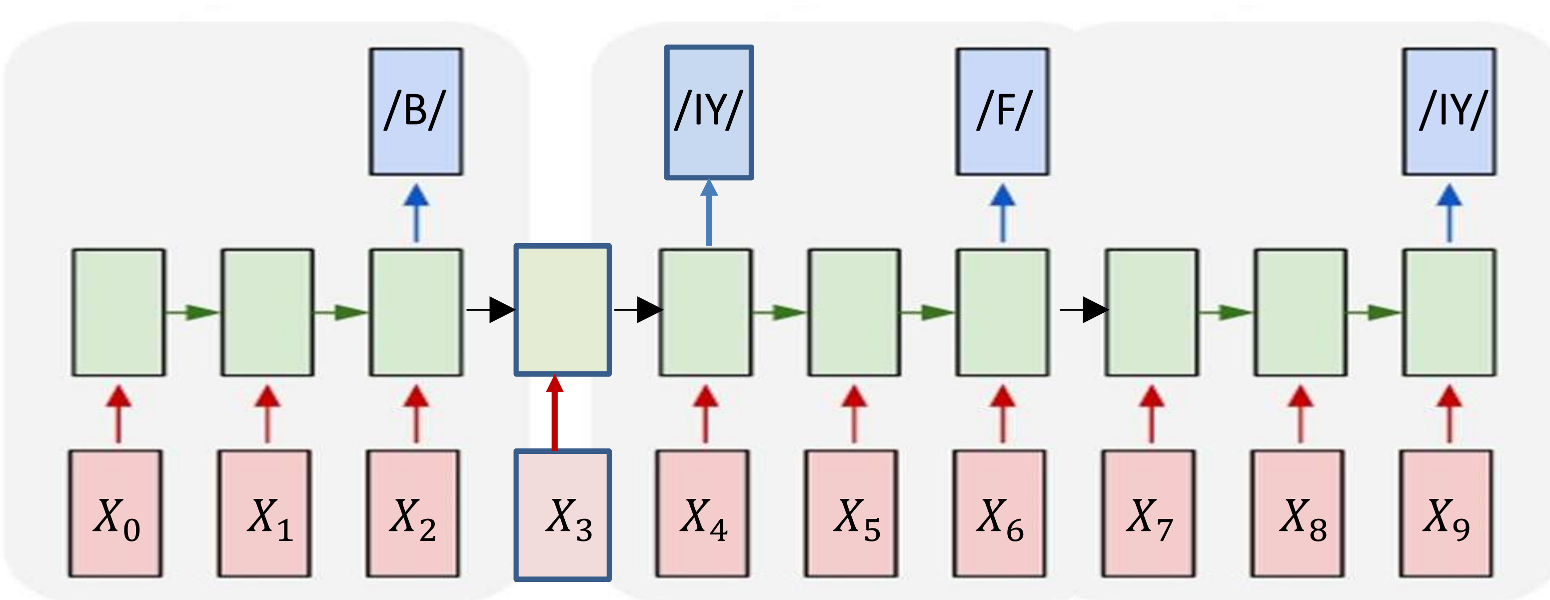
We will initially focus on the class of problem where uniform weights are reasonable (e.g speech recognition)

- Define the divergence everywhere

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

- 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

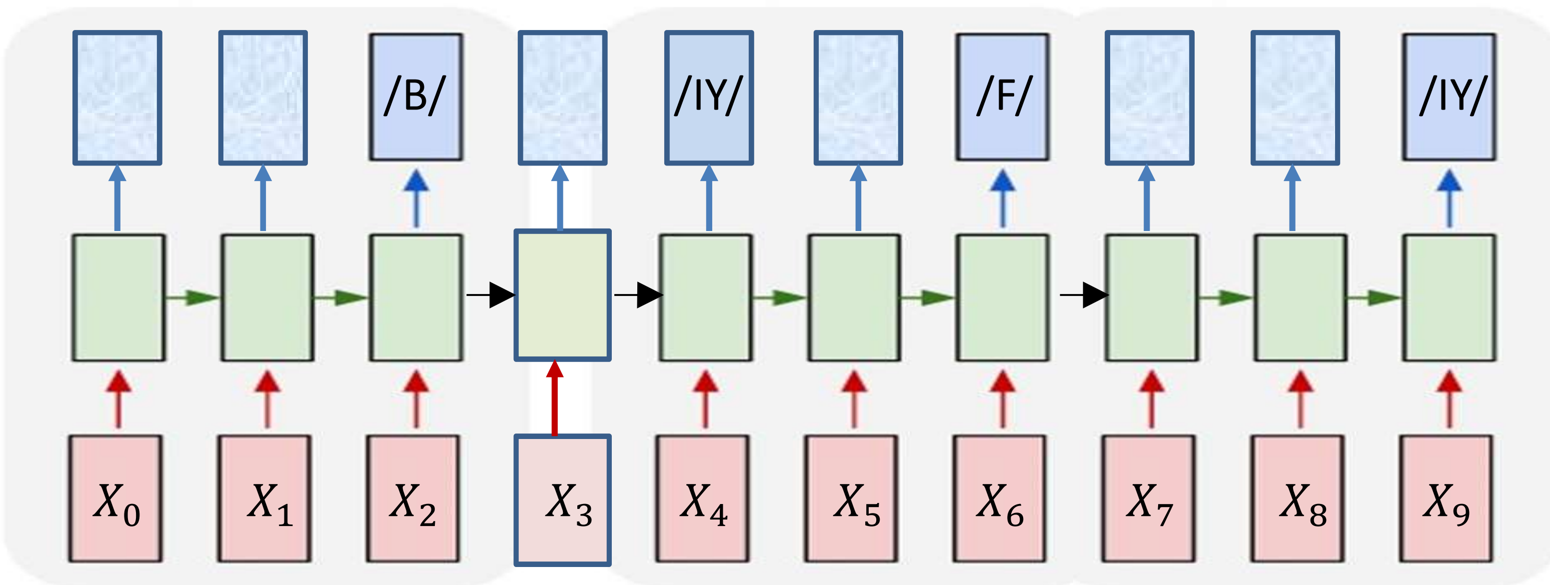
The more complex problem



- 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

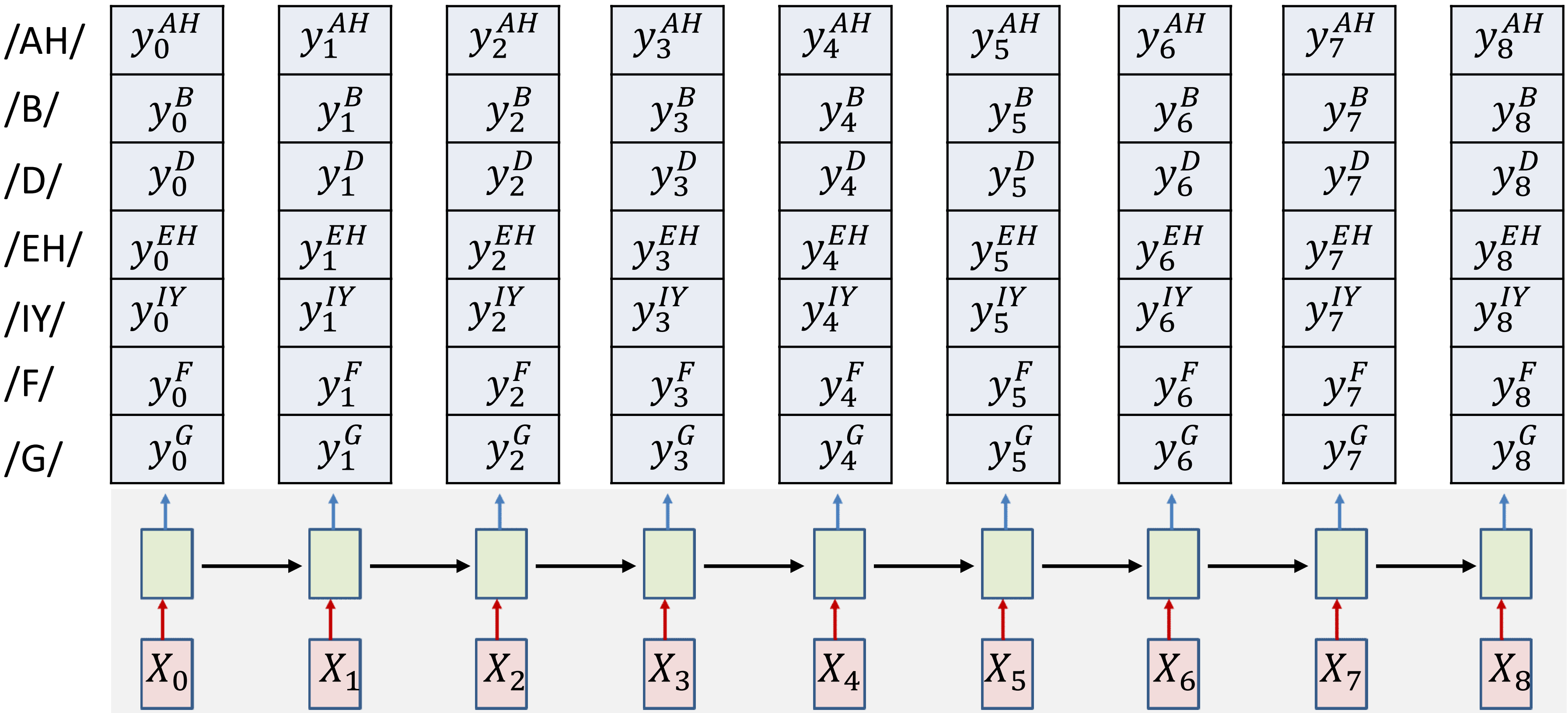
- But this simple extension complicates matters..

The *sequence-to-sequence* problem



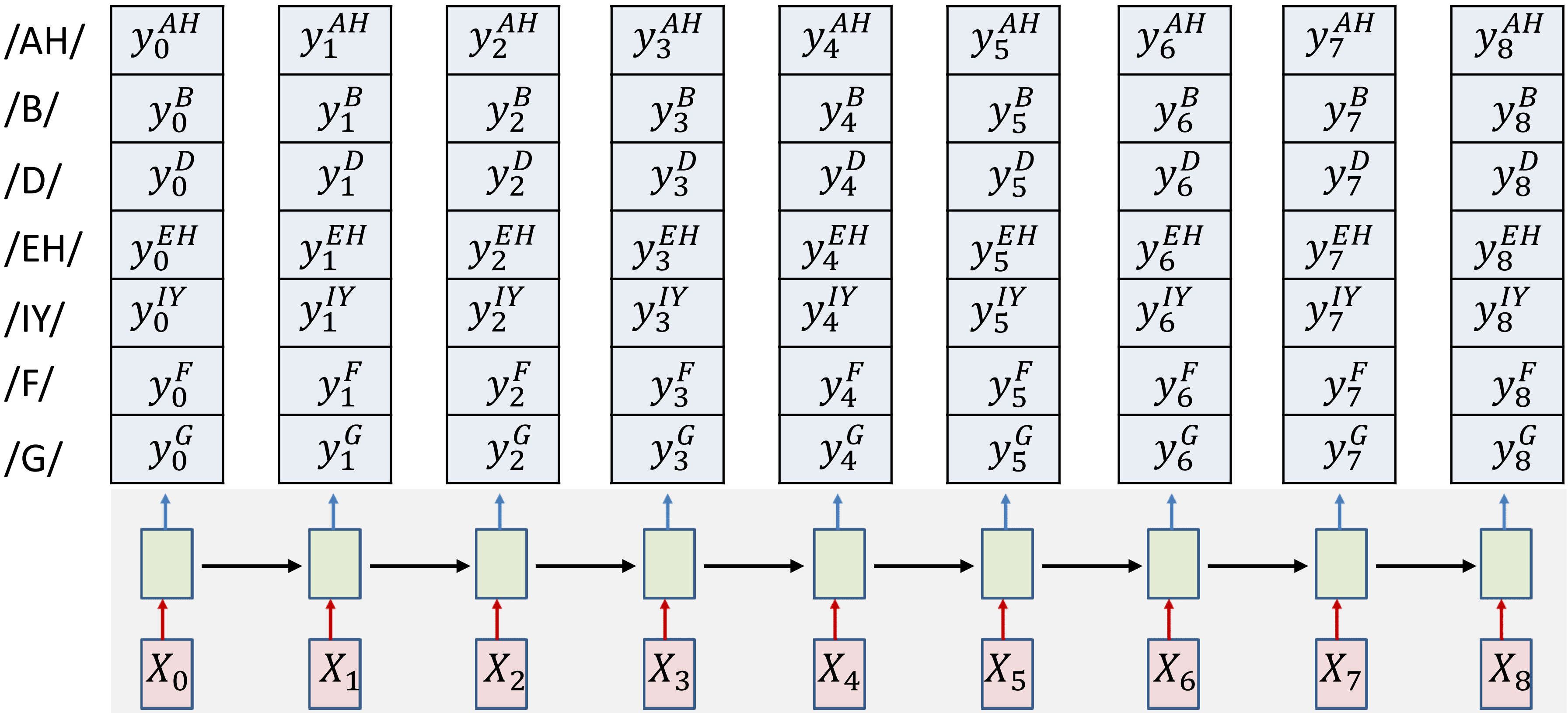
- How do we know *when* to output symbols
  - In fact, the network produces outputs at *every* time
  - *Which* of these are the *real* outputs?

The actual output of the network



- At each time the network outputs a probability for *each* output symbol given all inputs until that time
  - E.g. $y_4^D = prob(s_4 = D|X_0 \dots X_4)$

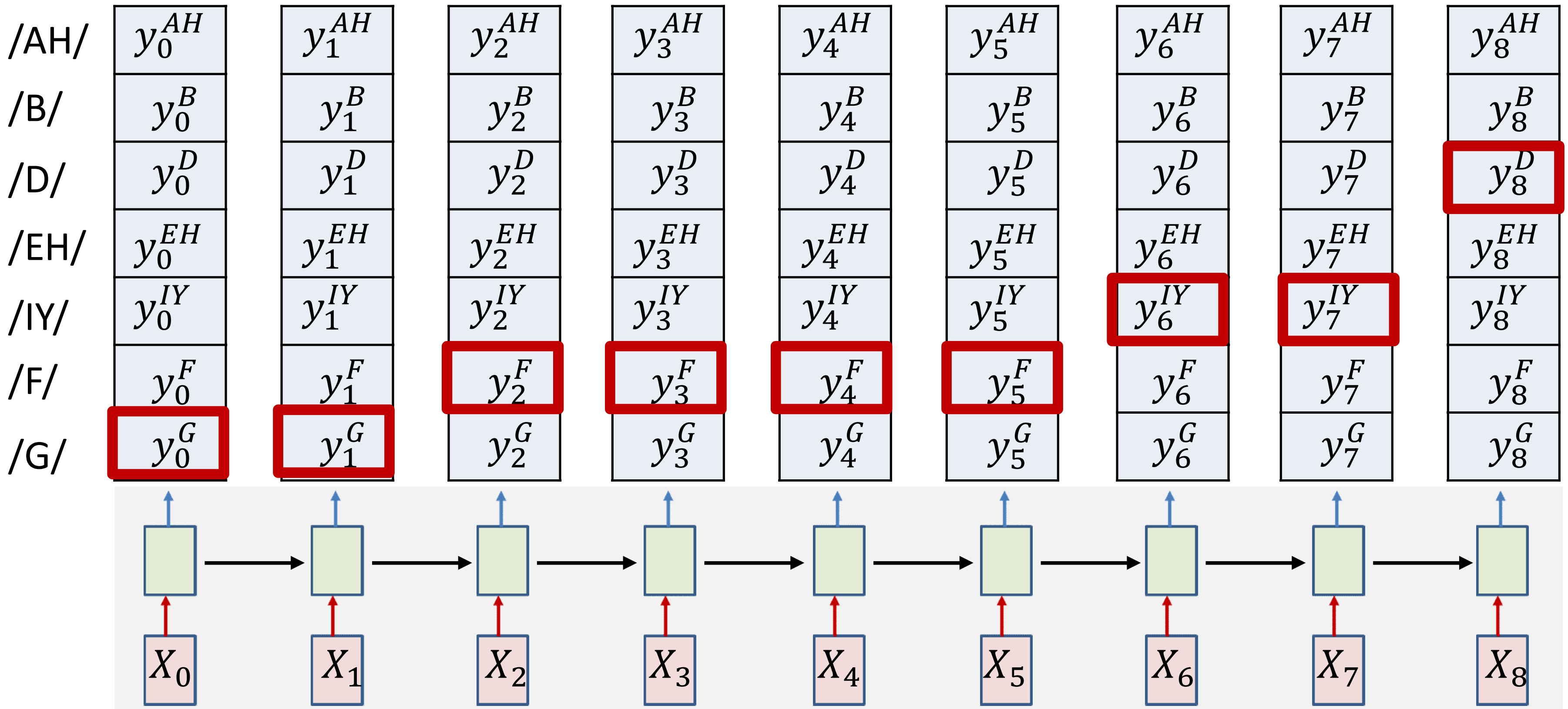
Overall objective



- Find most likely symbol sequence given inputs

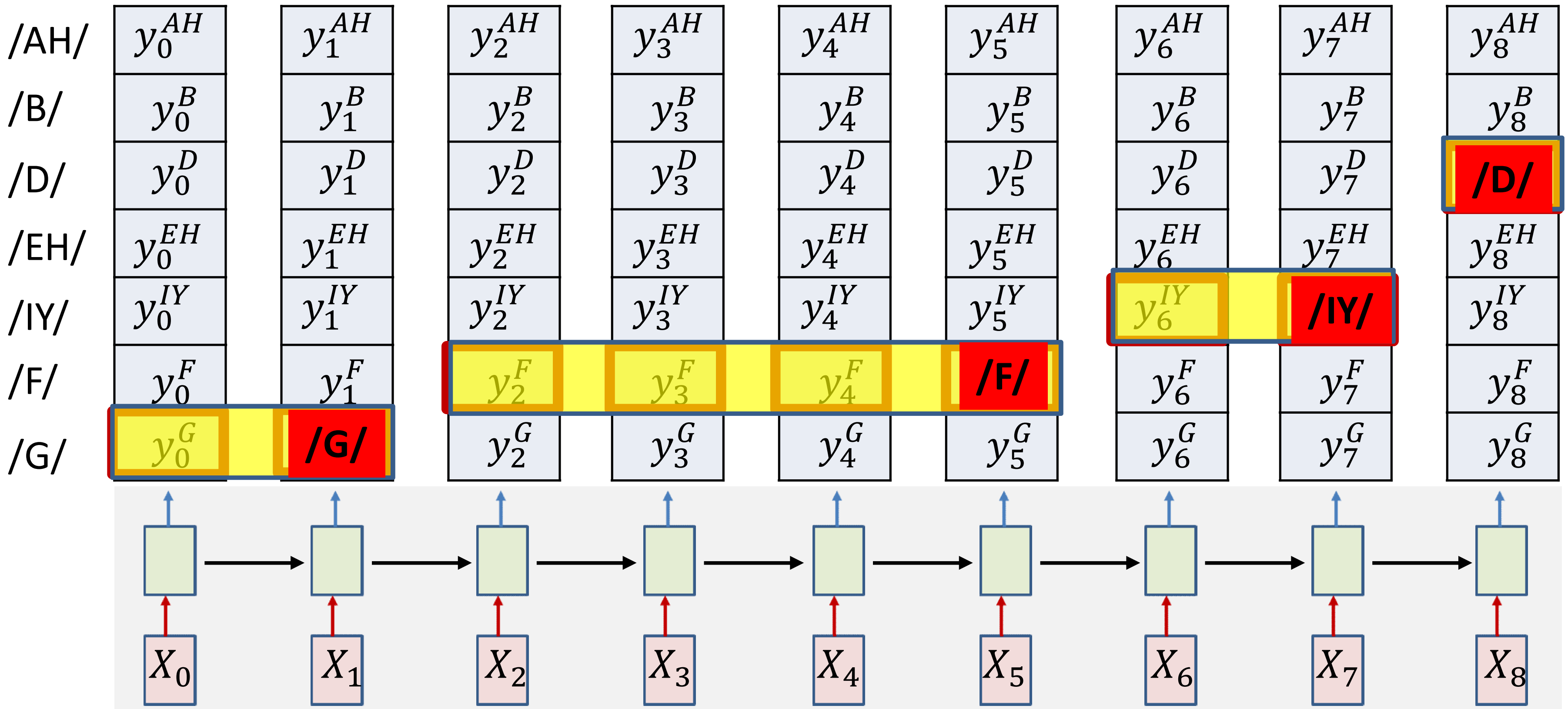
$$S_0 \ldots S_{K-1} = \underset{S'_0 \ldots S'_{K-1}}{\operatorname{argmax}} \; prob(S'_0 \ldots S'_{K-1} | X_0 \ldots X_{N-1})$$

Finding the best output



- Option 1: Simply select the most probable symbol at each time

Finding the best output



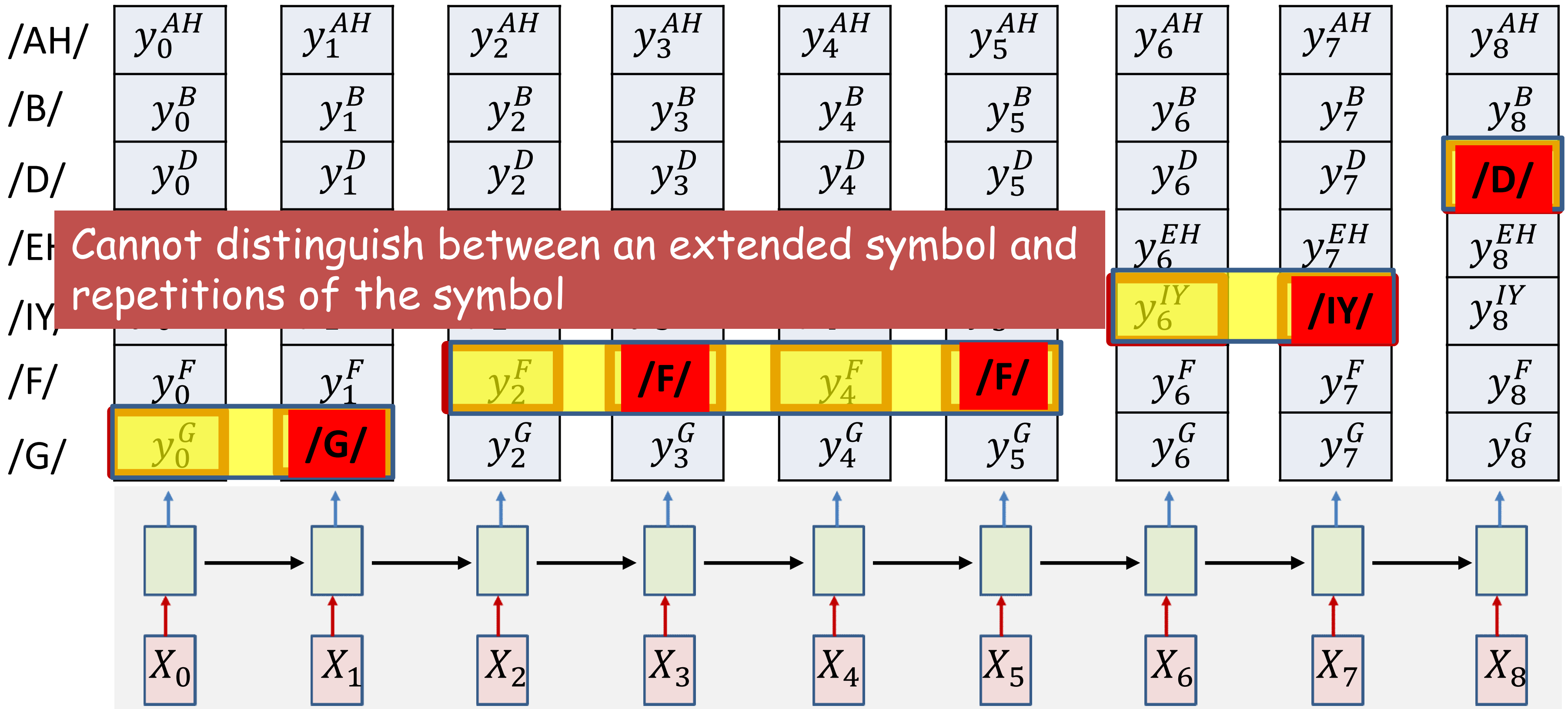
- Option 1: Simply select the most probable symbol at each time
  - *Merge* adjacent repeated symbols, and place the actual emission of the symbol in the final instant

Simple pseudocode

- Assuming $y(t, i), t = 1 \ldots T, i = 1 \ldots N$ is already computed using the underlying RNN

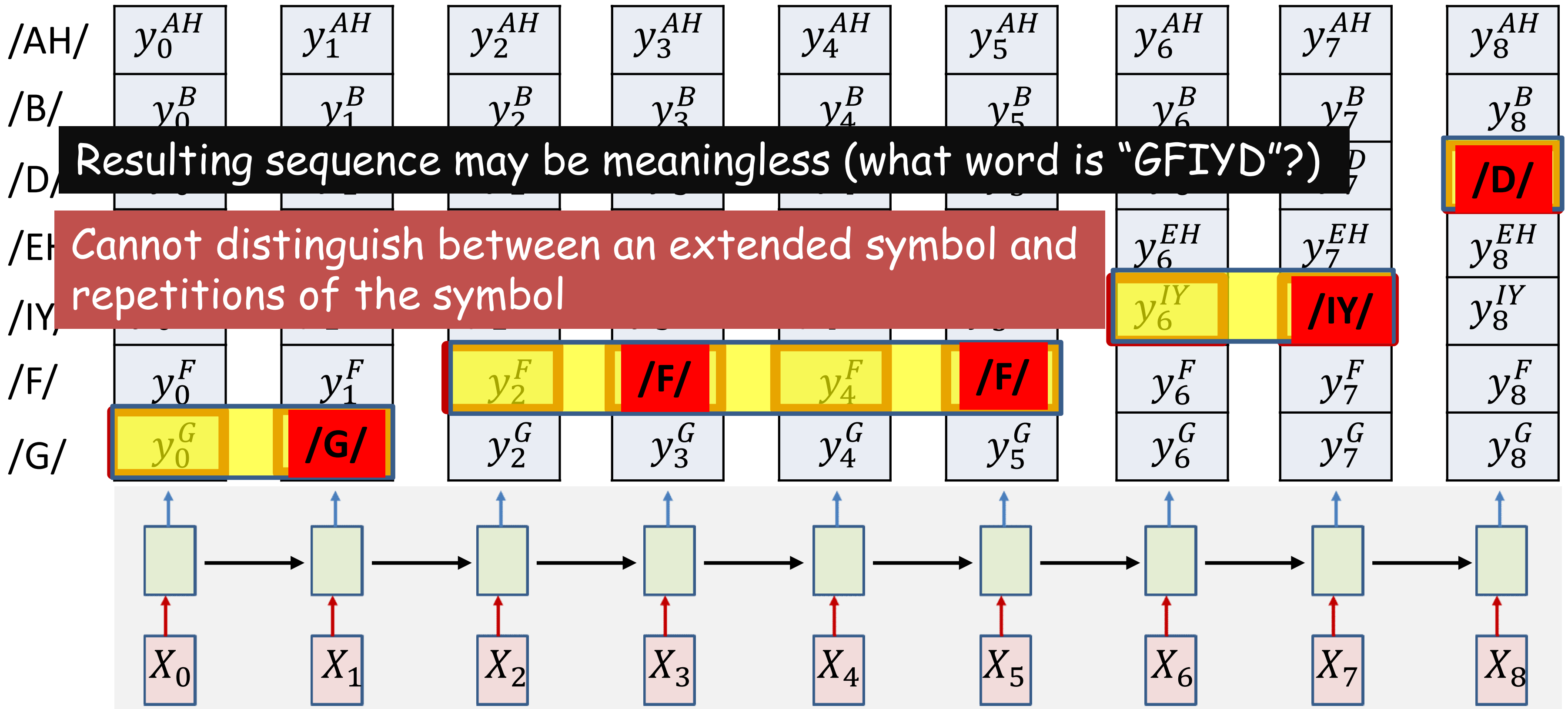
```
n = 1
best(1)= max_i(y(1,i))
for t = 1:T
    best(t)= max_i(y(t,i))
    if (best(t) != best(t-1))
        out(n) = best(t-1)
        time(n) = t-1
        n = n+1
```

The actual output of the network



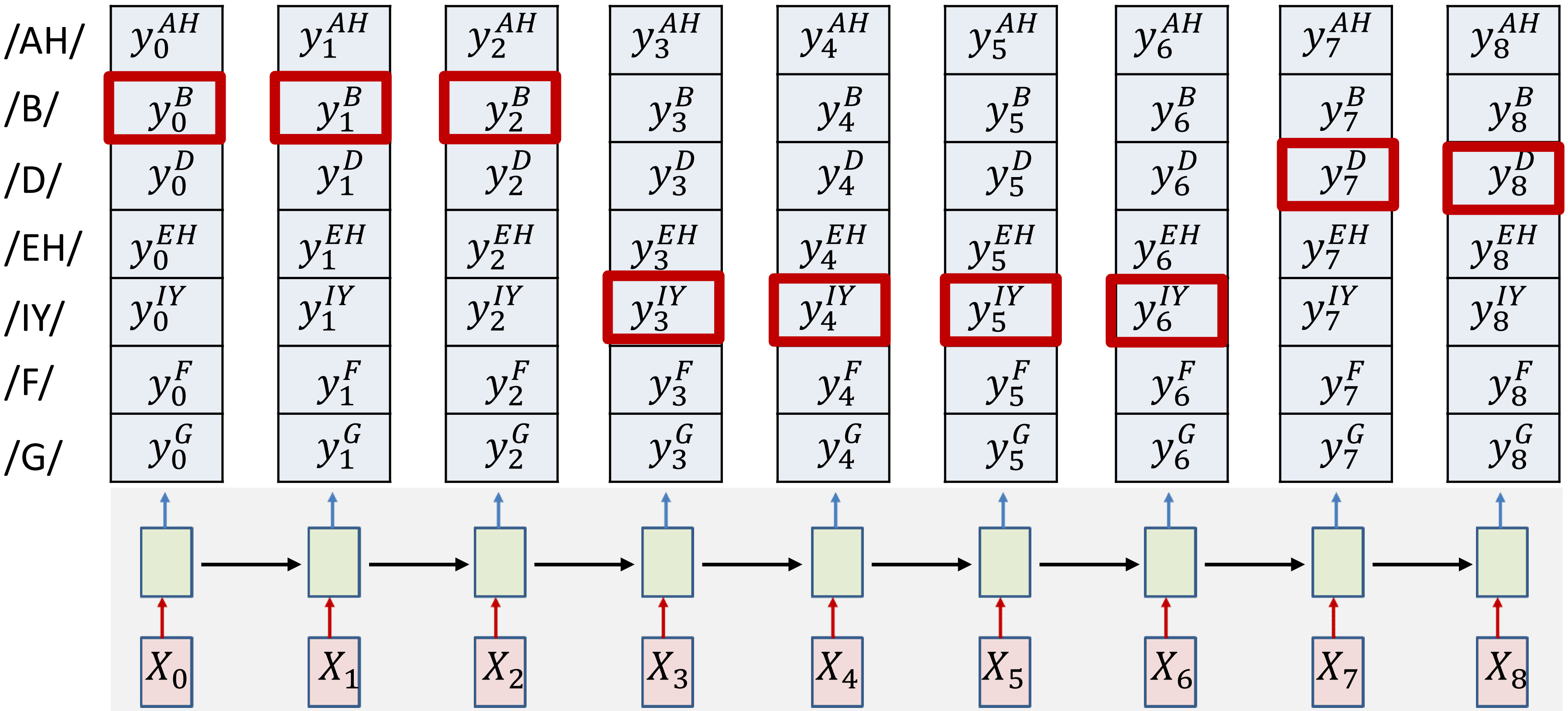
- Option 1: Simply select the most probable symbol at each time
  - *Merge* adjacent repeated symbols, and place the actual emission of the symbol in the final instant

The actual output of the network



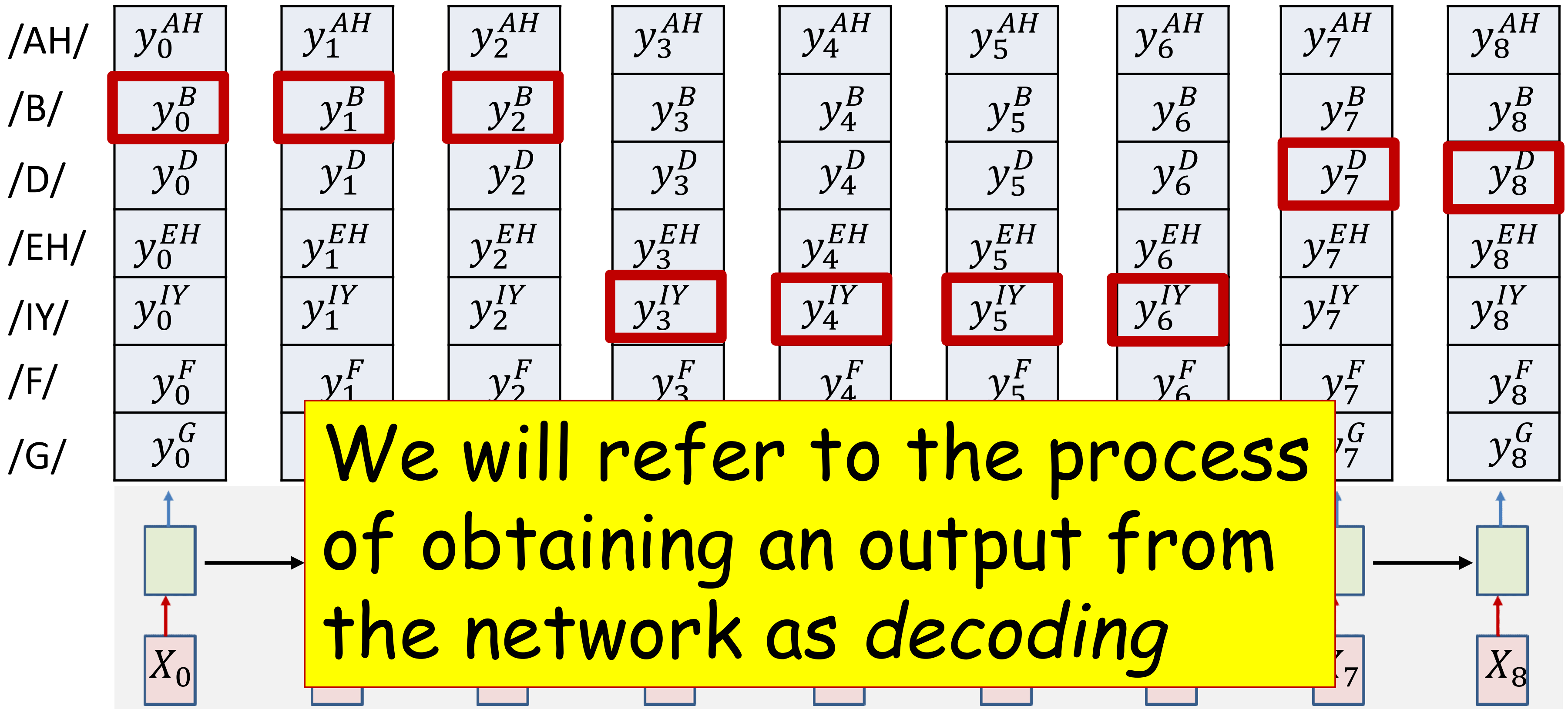
- Option 1: Simply select the most probable symbol at each time
  – *Merge* adjacent repeated symbols, and place the actual emission of the symbol in the final instant

The actual output of the network



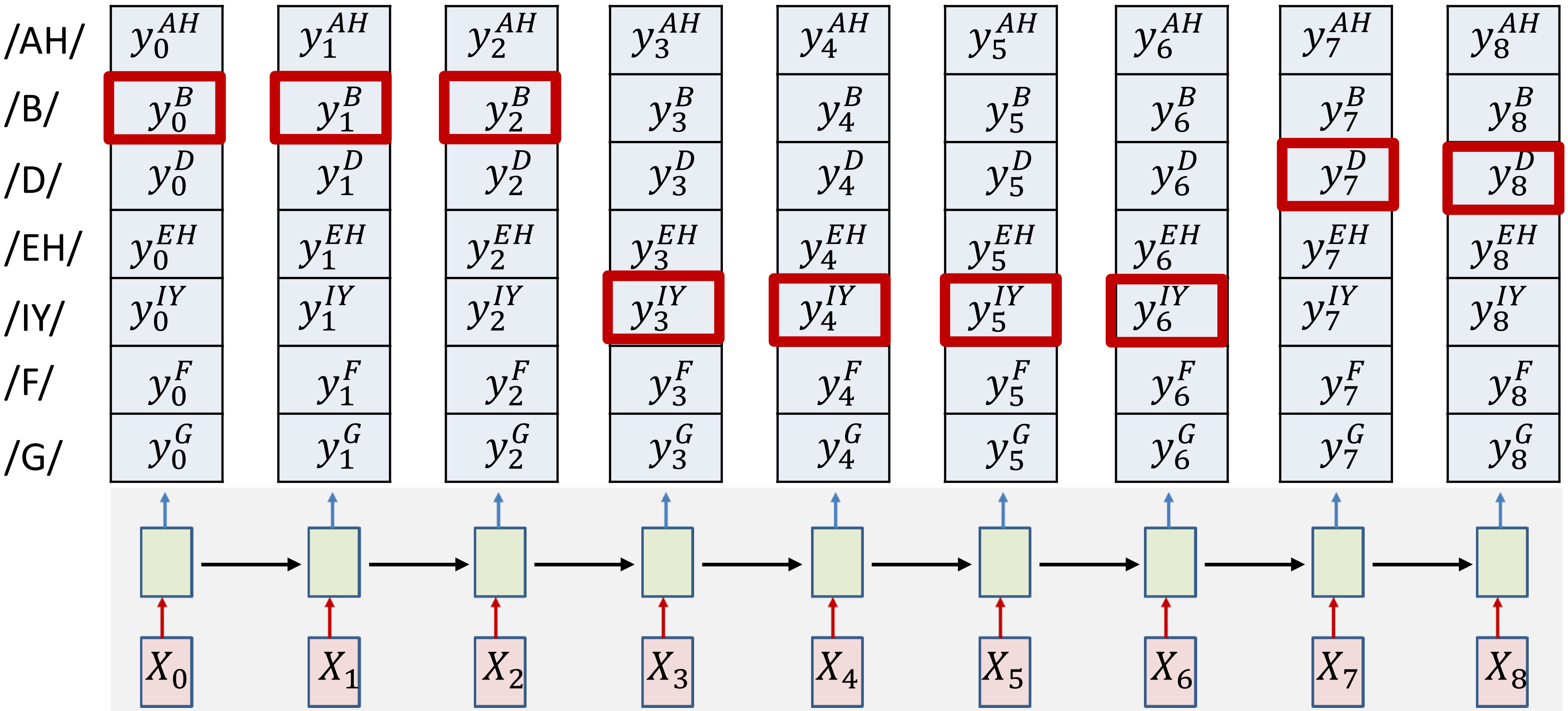
- Option 2: Impose external constraints on what sequences are allowed
  - *E.g.* only allow sequences corresponding to dictionary words
  - *E.g. Sub-symbol* units (like in HW1 – what were they?)

The actual output of the network



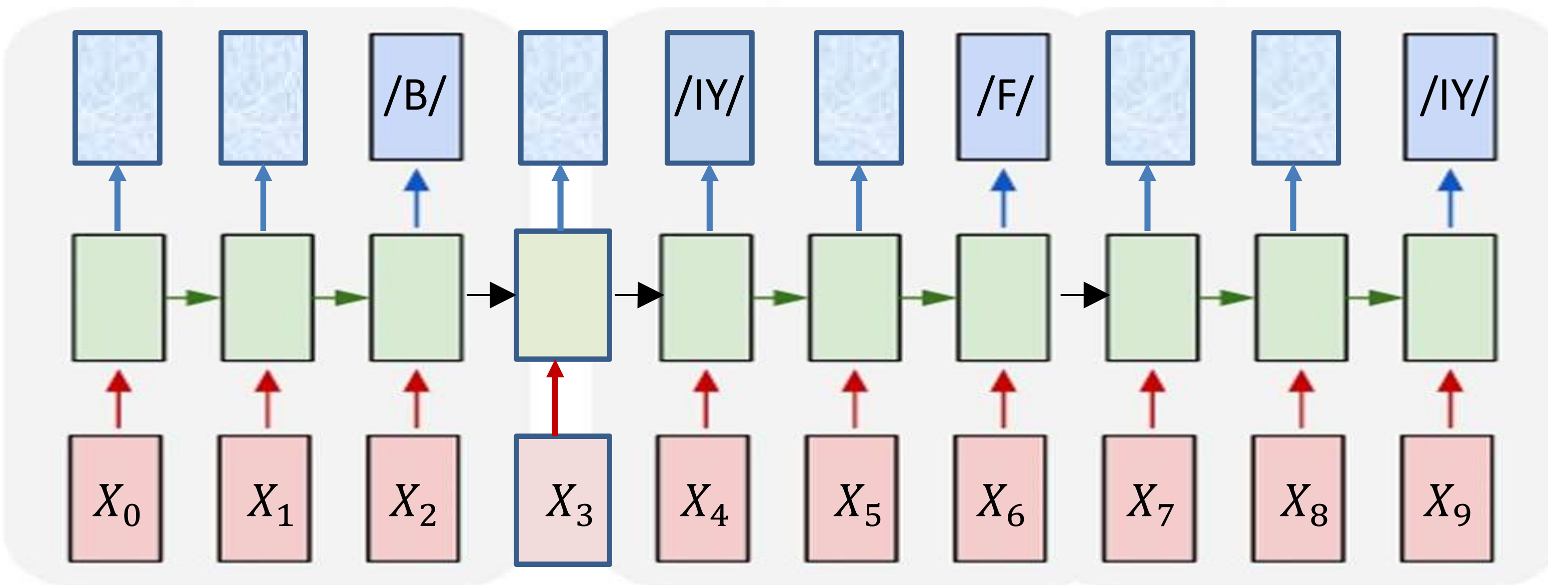
- Option 2: Impose external constraints on what sequences are allowed
  - *E.g.* only allow sequences corresponding to dictionary words
  - *E.g. Sub-symbol* units (like in HW1 – what were they?)

Decoding



- This is in fact a *suboptimal* decode that actually finds the most likely *synchronous* output sequence
  - Which is not necessarily the most likely *asynchronous* sequence
    - The “merging” heuristics do not guarantee optimal asynchronous sequences
  - We will return to this topic later

The *sequence-to-sequence* problem

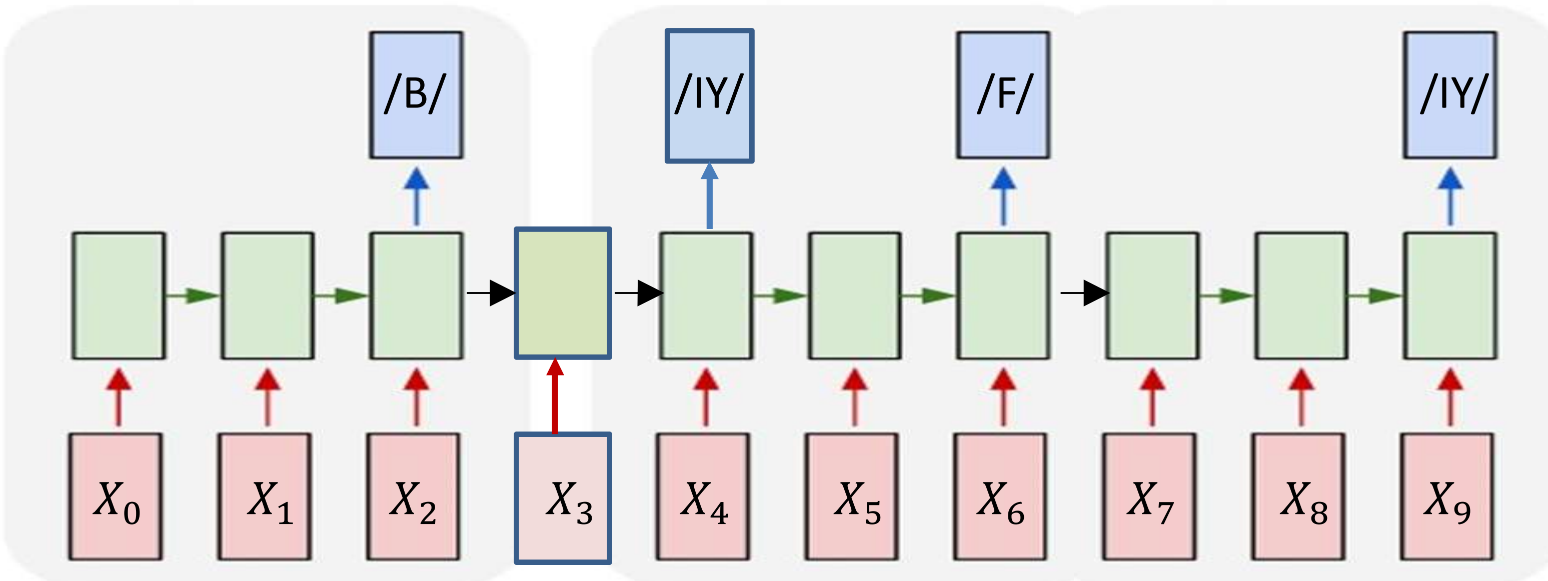


- How do we know *when* to output symbols
  - In fact, the network produces outputs at *every* time
  - *Which* of these are the *real* outputs

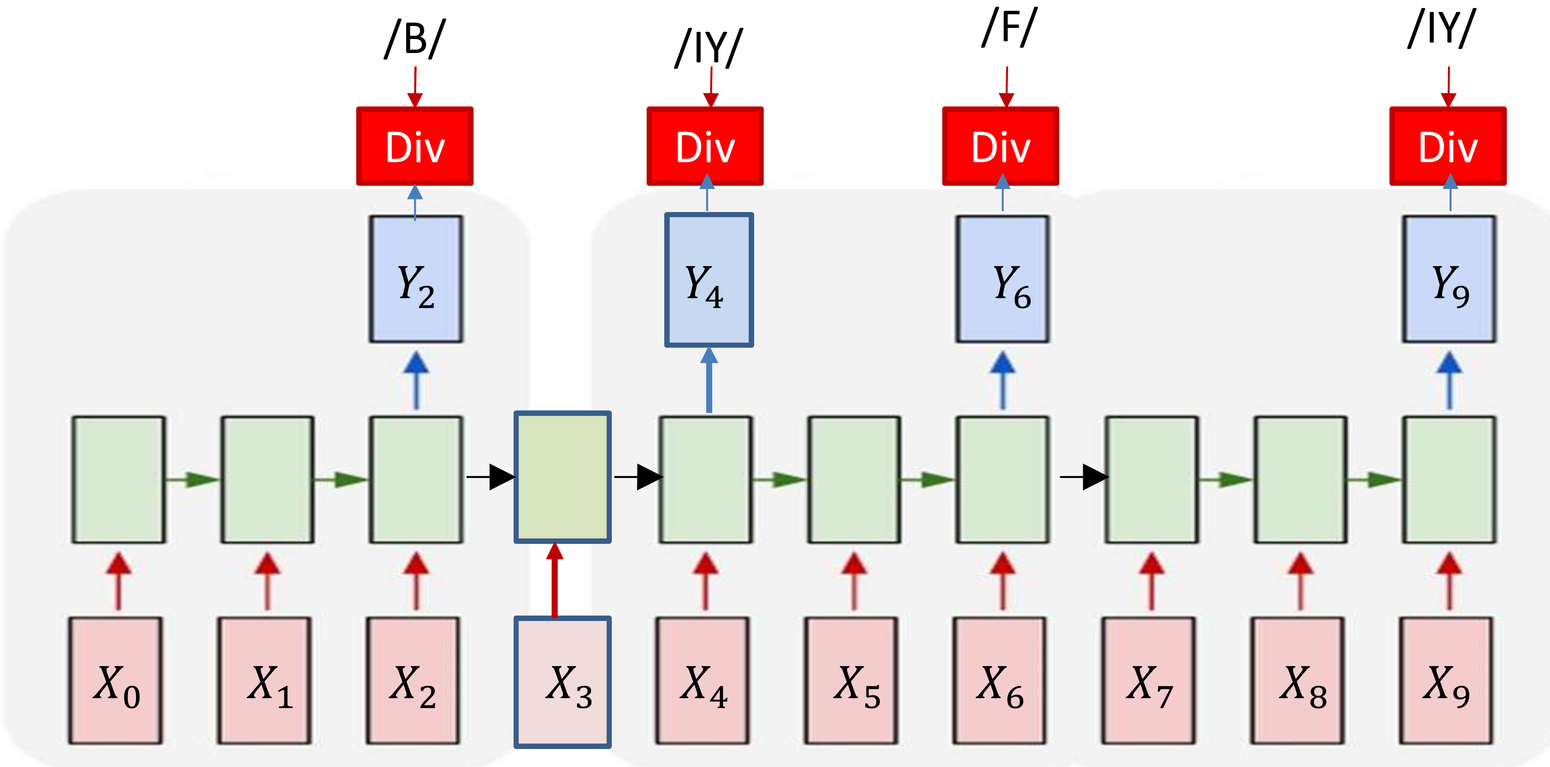
Addressed

- How do we *train* these models?

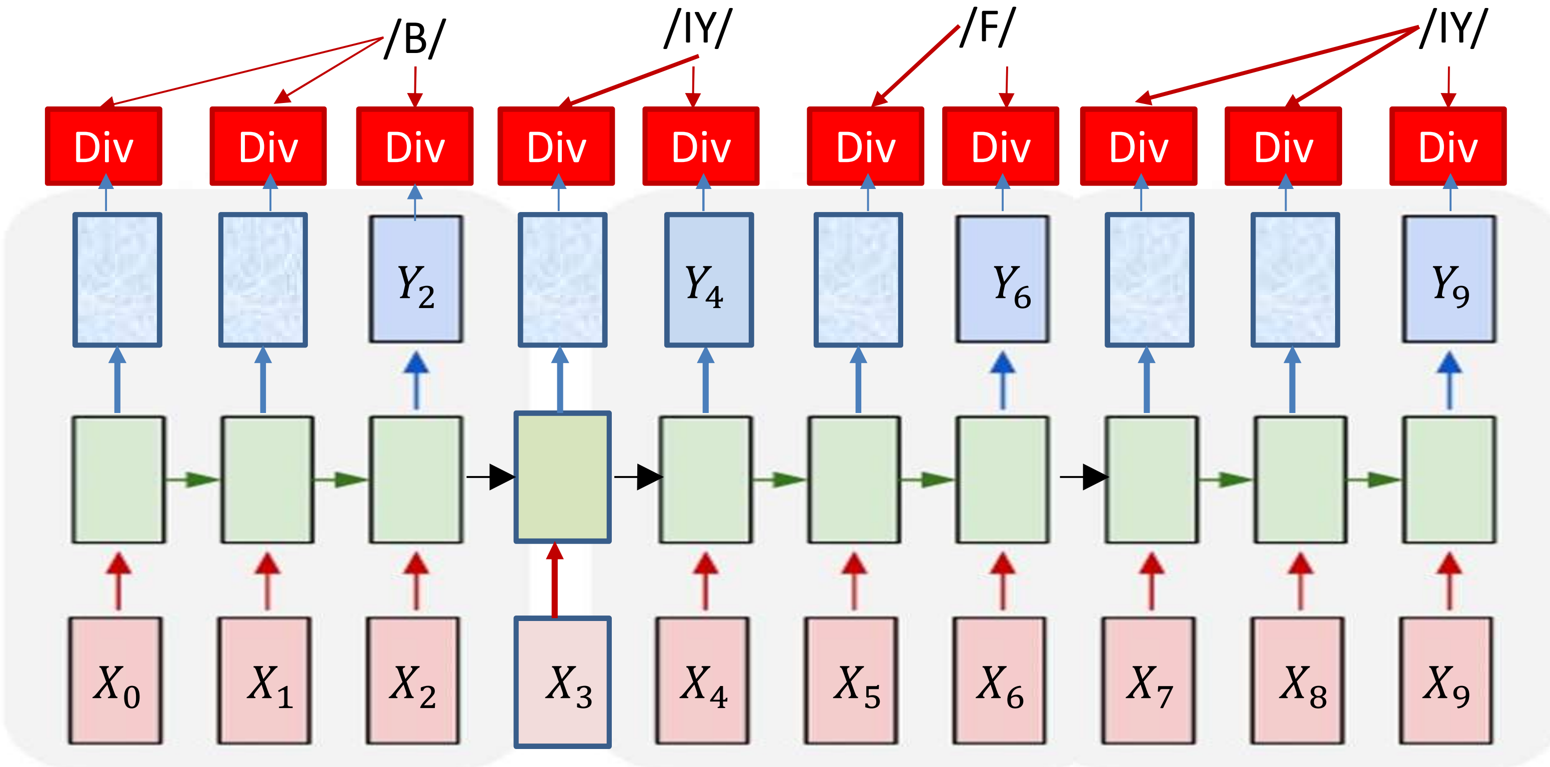
Training



- Given output symbols *at the right locations*
  - The phoneme /B/ ends at X_2, /IY/ at X_4, /F/ at X_6, /IY/ at X_9



- Either just define Divergence as:
  $DIV = Xent(Y_2, B) + Xent(Y_4, IY) + Xent(Y_6, F) + Xent(Y_9, IY)$
- Or..

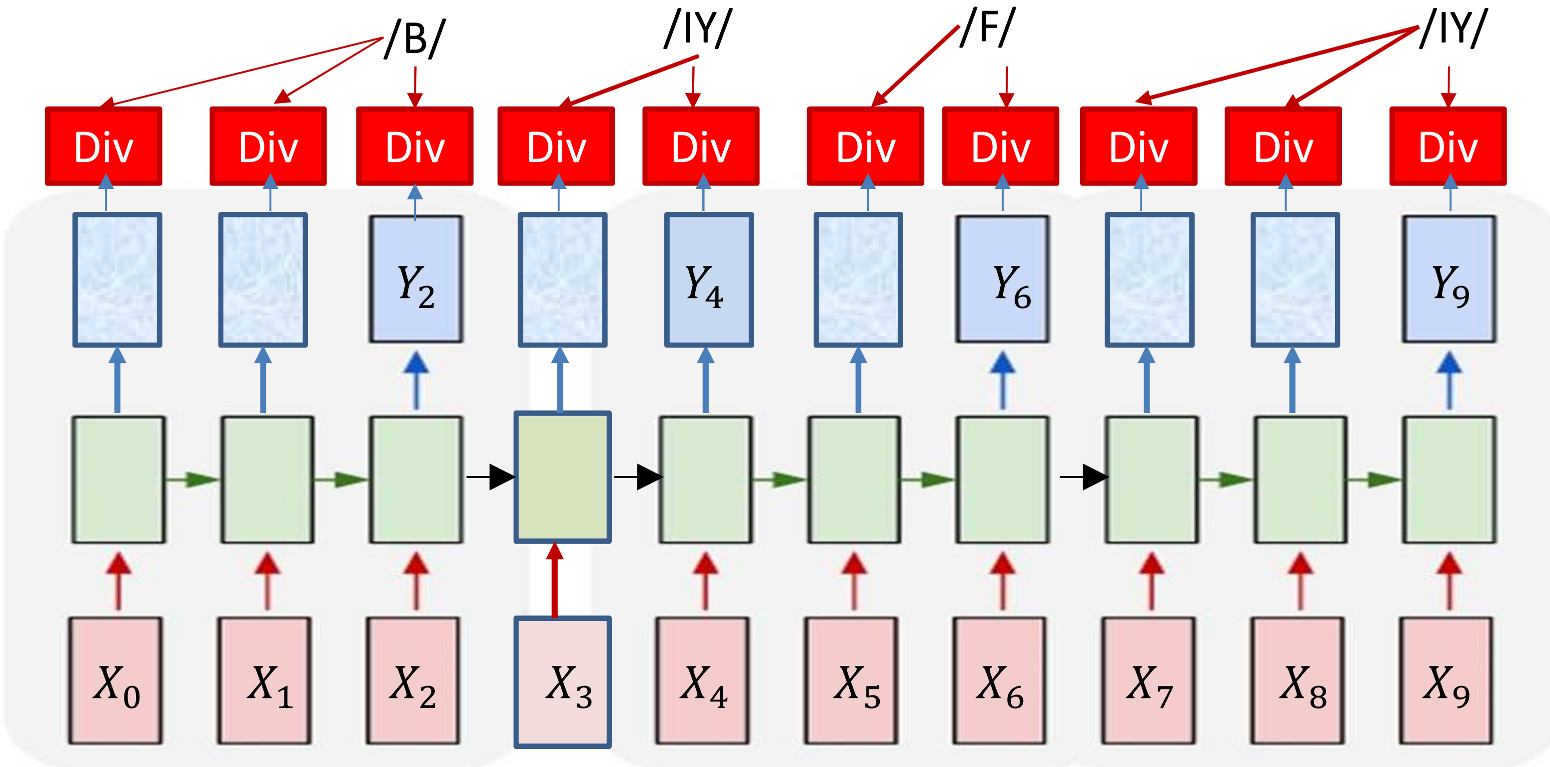


- Either just define Divergence as:

$$DIV = Xent(Y_2, B) + Xent(Y_4, IY) + Xent(Y_6, F) + Xent(Y_9, IY)$$

- Or *repeat the symbols over their duration*

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



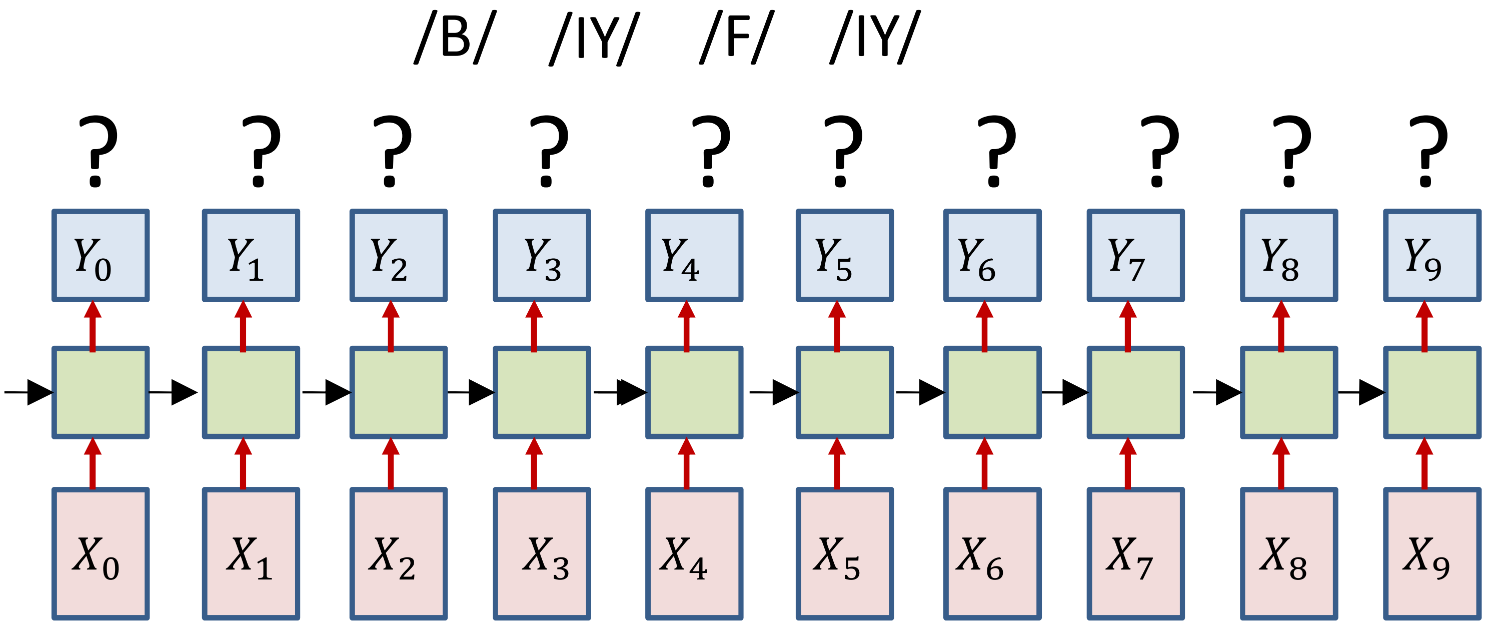
$$DIV = \sum_t Xent(Y_t, symbol_t) = -\sum_t \log Y(t, symbol_t)$$

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

$$\nabla_{Y_t} DIV = \begin{bmatrix} 0 & 0 & \dots & \dfrac{-1}{Y(t, symbol_t)} & 0 & \dots & 0 \end{bmatrix}$$

  – Zeros except at the component corresponding to the target

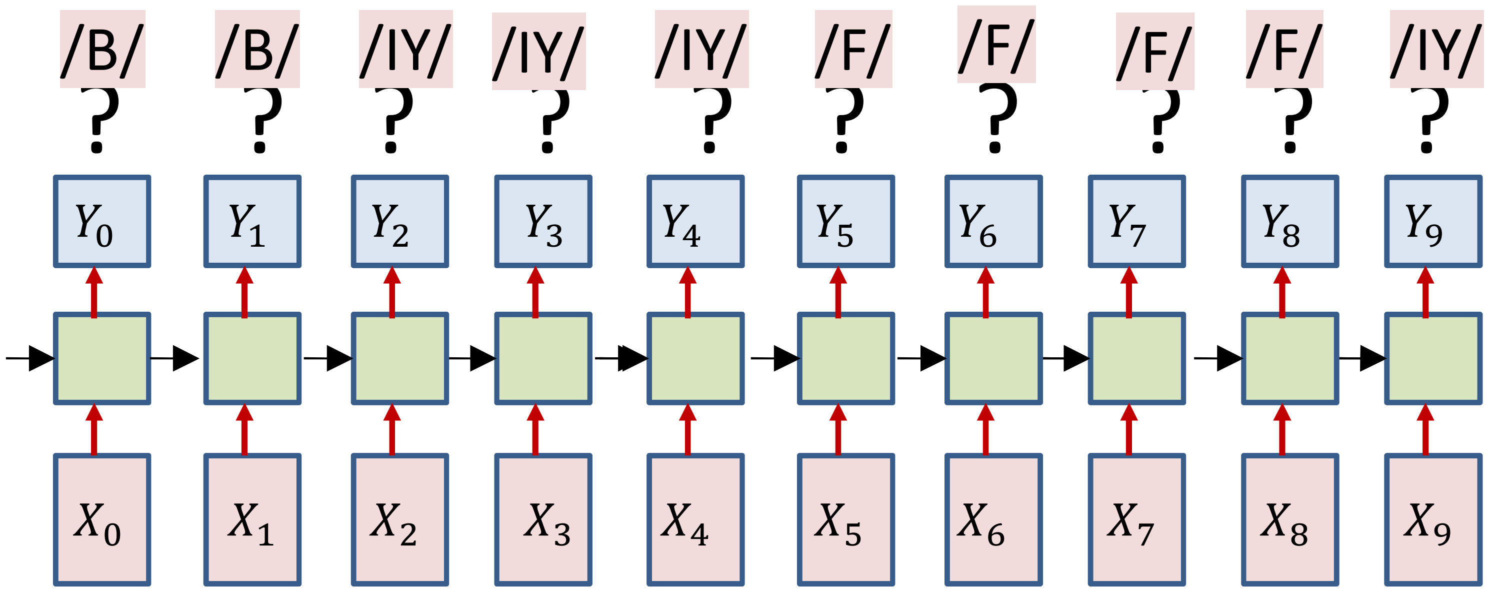
Problem: No timing information provided



- Only the sequence of output symbols is provided for the training data
  - But no indication of which one occurs where

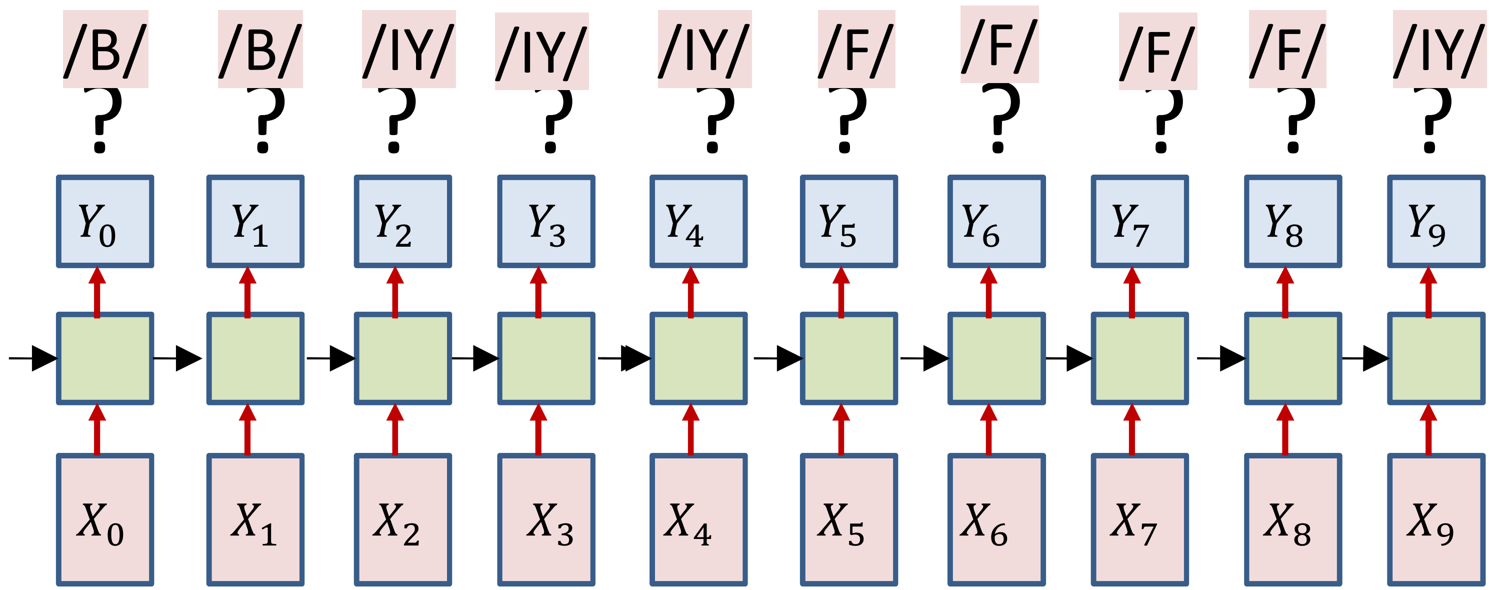
- How do we compute the divergence?
  - And how do we compute its gradient w.r.t. Y_t

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

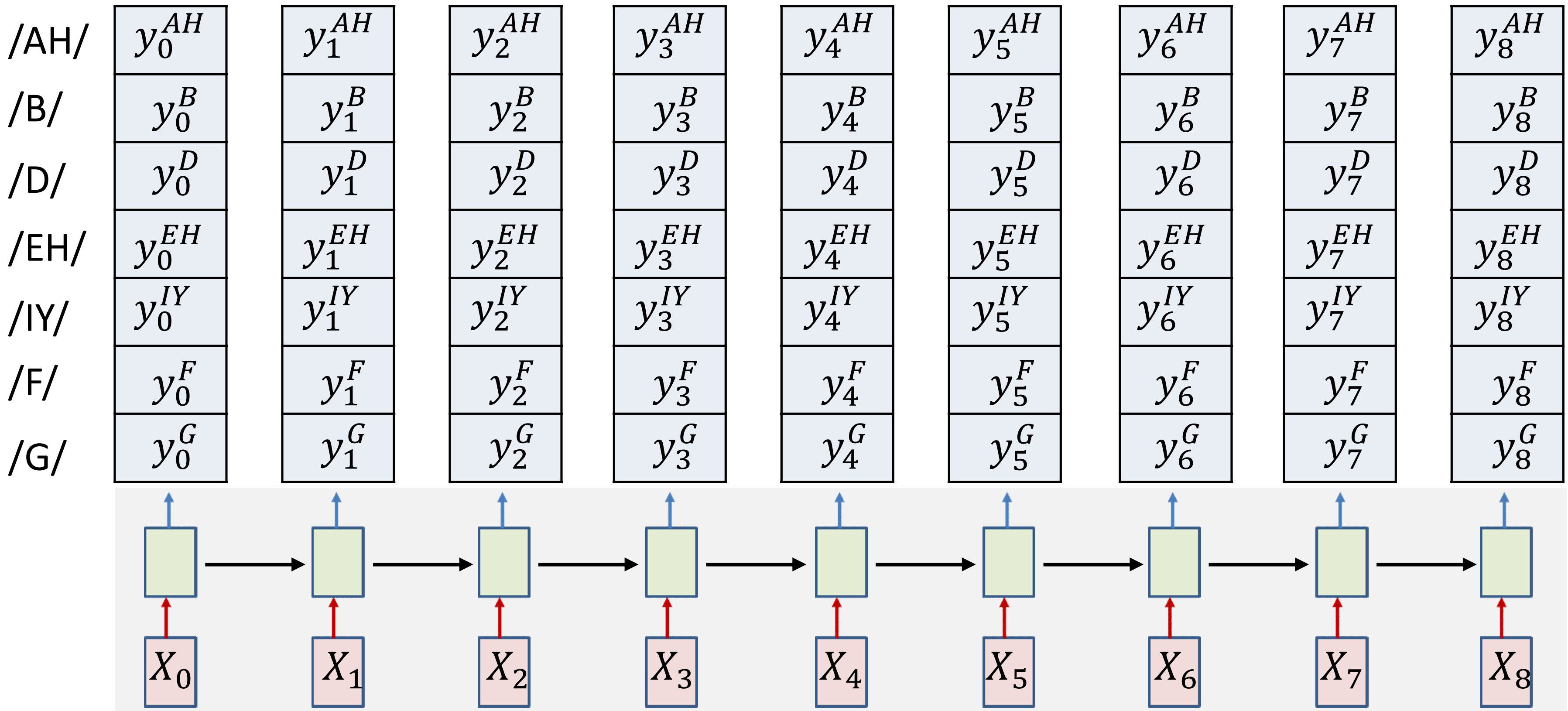
Estimating an alignment

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

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

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

Recall: The actual output of the network



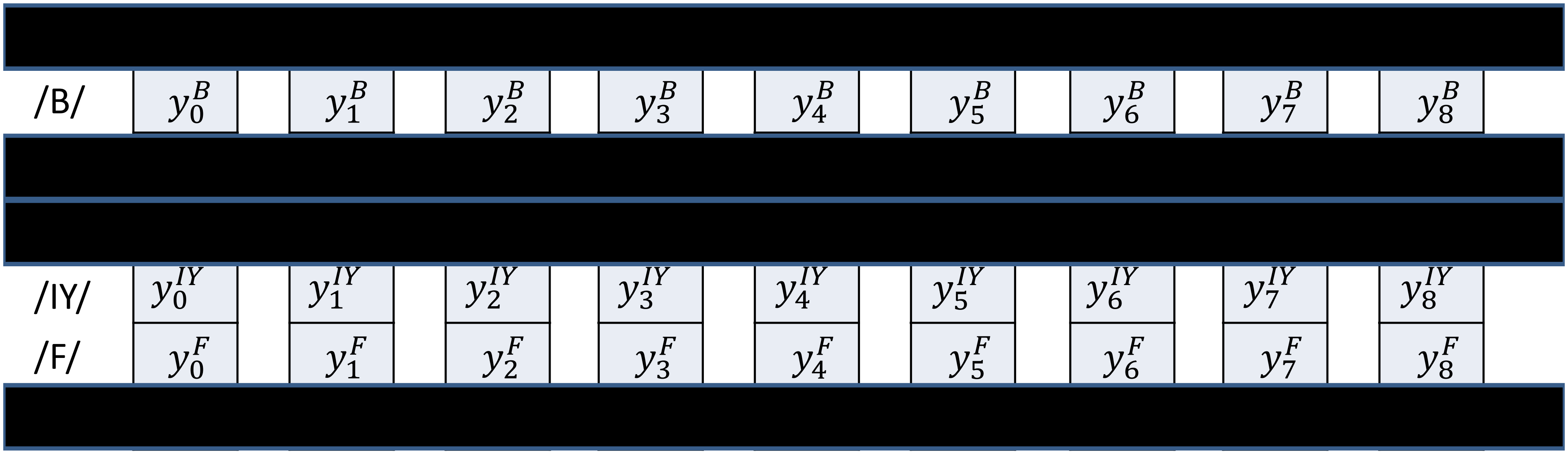
- At each time the network outputs a probability for *each* output symbol

Recall: unconstrained decoding

/AH/	$\boxed{y_0^{AH}}$	$\boxed{y_1^{AH}}$	$\boxed{y_2^{AH}}$	y_3^{AH}	y_4^{AH}	$\boxed{y_5^{AH}}$	y_6^{AH}	y_7^{AH}	y_8^{AH}
/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}
/D/	y_0^{D}	y_1^{D}	y_2^{D}	$\boxed{y_3^{D}}$	$\boxed{y_4^{D}}$	y_5^{D}	y_6^{D}	y_7^{D}	y_8^{D}
/EH/	y_0^{EH}	y_1^{EH}	y_2^{EH}	y_3^{EH}	y_4^{EH}	y_5^{EH}	y_6^{EH}	y_7^{EH}	y_8^{EH}
/IY/	y_0^{IY}	y_1^{IY}	y_2^{IY}	y_3^{IY}	y_4^{IY}	y_5^{IY}	y_6^{IY}	$\boxed{y_7^{IY}}$	$\boxed{y_8^{IY}}$
/F/	y_0^{F}	y_1^{F}	y_2^{F}	y_3^{F}	y_4^{F}	y_5^{F}	$\boxed{y_6^{F}}$	y_7^{F}	y_8^{F}
/G/	y_0^{G}	y_1^{G}	y_2^{G}	y_3^{G}	y_4^{G}	y_5^{G}	y_6^{G}	y_7^{G}	y_8^{G}

- We find the most likely sequence of symbols
  - (Conditioned on input $X_0 \dots X_{N-1}$)
- This may not correspond to an expansion of the desired symbol sequence
  - E.g. the unconstrained decode may be /AH//AH//AH//D//D//AH//F//IY//IY/
    - Contracts to /AH/ /D/ /AH/ /F/ /IY/
  - Whereas we want an expansion of /B//IY//F//IY/

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

Constraining the alignment: Try 1

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

- Only decode on reduced grid
  - We are now assured that only the appropriate symbols will be hypothesized
- Problem: This still doesn’t assure that the decode sequence correctly expands the target symbol sequence
  - E.g. the above decode is not an expansion of /B//IY//F//IY/
- Still needs additional constraints

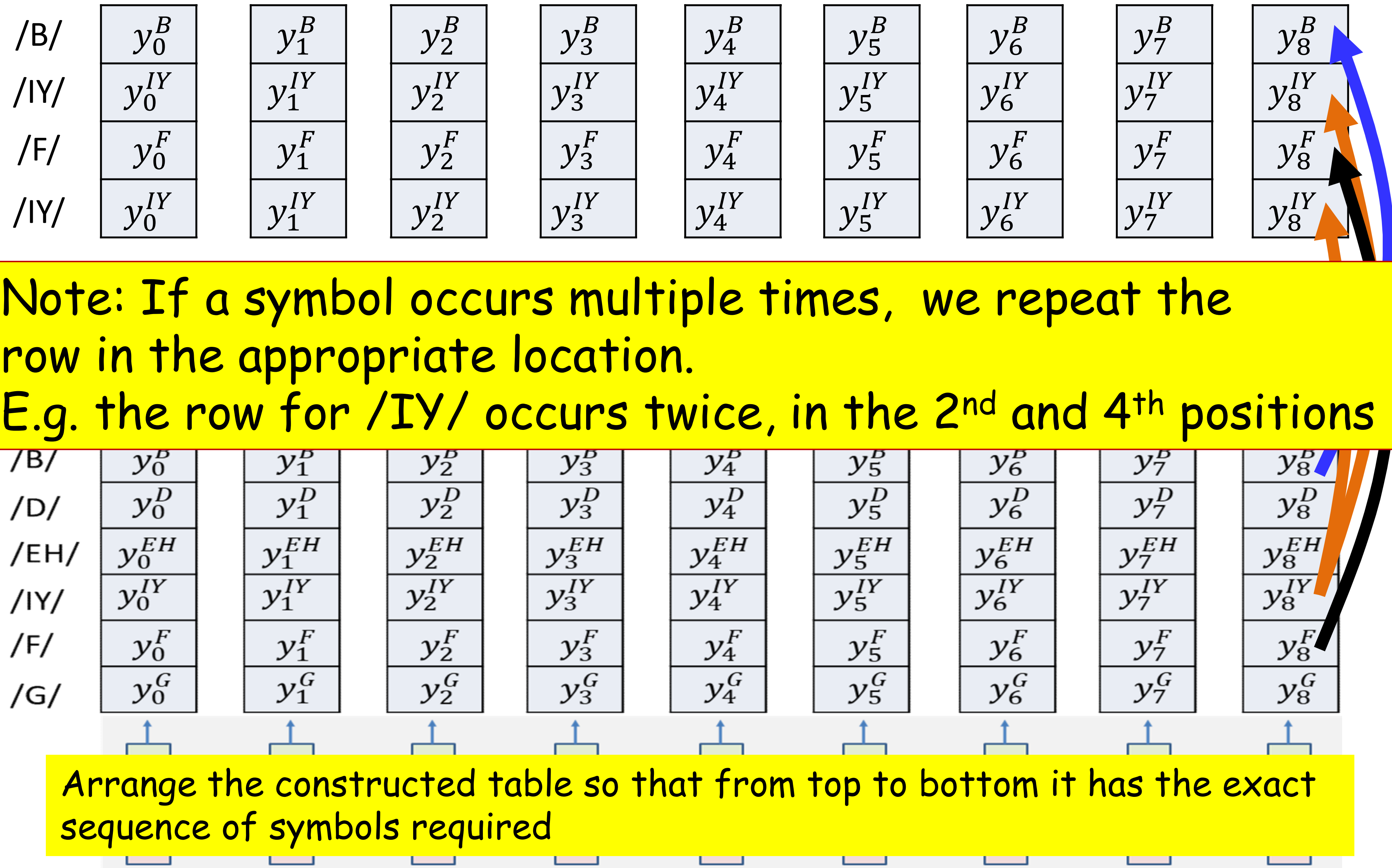
Try 2: Explicitly arrange the constructed table

/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
/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}

/AH/	y_0^{AH}	y_1^{AH}	y_2^{AH}	y_3^{AH}	y_4^{AH}	y_5^{AH}	y_6^{AH}	y_7^{AH}	y_8^{AH}
/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
/D/	y_0^D	y_1^D	y_2^D	y_3^D	y_4^D	y_5^D	y_6^D	y_7^D	y_8^D
/EH/	y_0^{EH}	y_1^{EH}	y_2^{EH}	y_3^{EH}	y_4^{EH}	y_5^{EH}	y_6^{EH}	y_7^{EH}	y_8^{EH}
/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
/G/	y_0^G	y_1^G	y_2^G	y_3^G	y_4^G	y_5^G	y_6^G	y_7^G	y_8^G

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

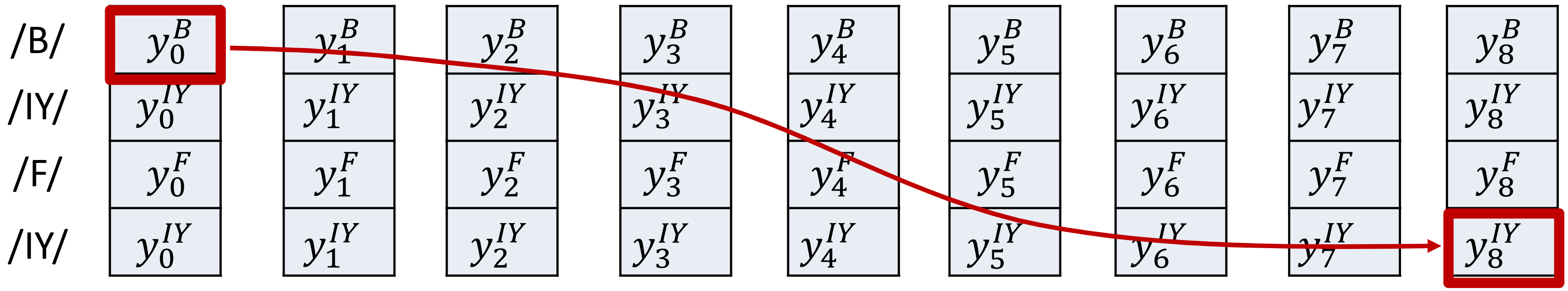
Try 2: Explicitly arrange the constructed table



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

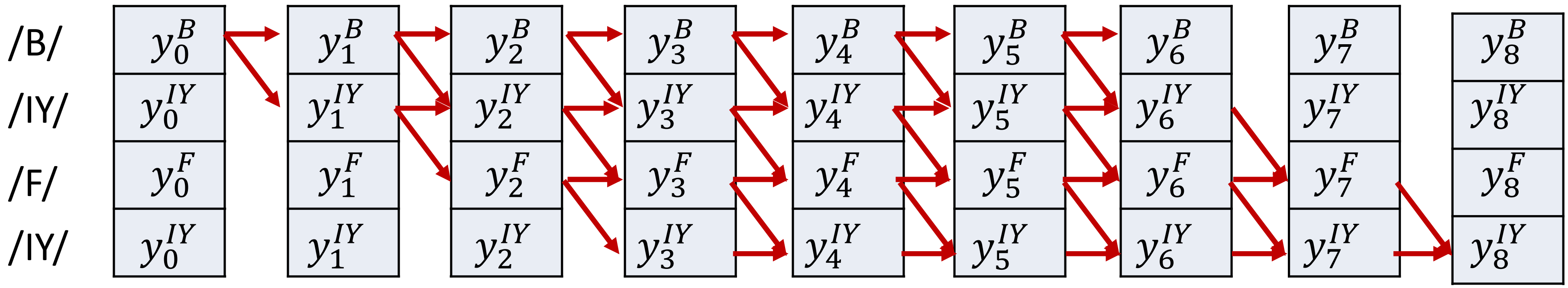
Arrange the constructed table so that from top to bottom it has the exact sequence of symbols required

Explicitly constrain alignment



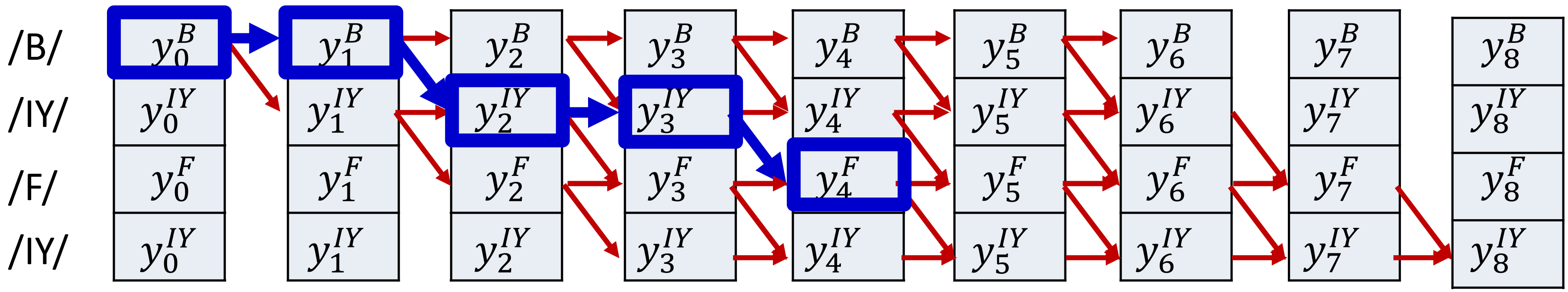
- Constrain that the first symbol in the decode *must* be the top left block
- The last symbol *must* be the bottom right
- The rest of the symbols must follow a sequence that *monotonically* travels down from top left to bottom right
  - I.e. symbol chosen at any time is at the same level or at the 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

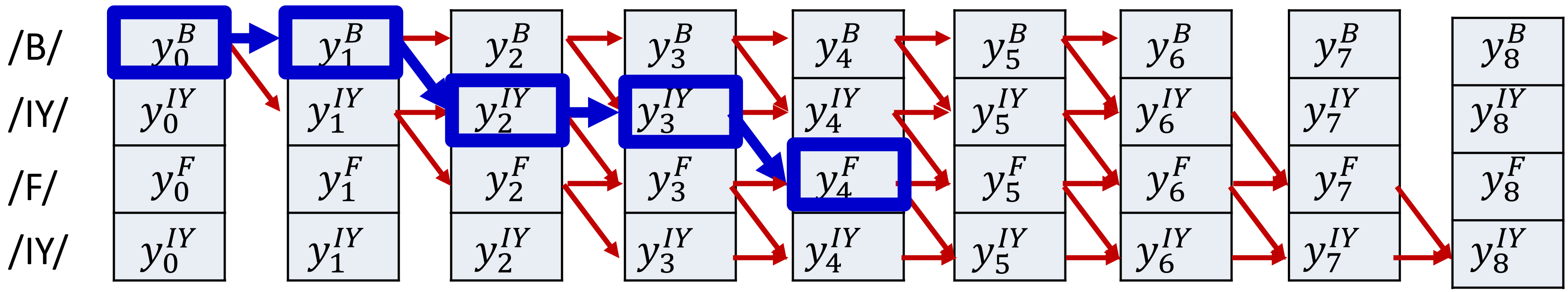
Path Score (probability)



- 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**

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

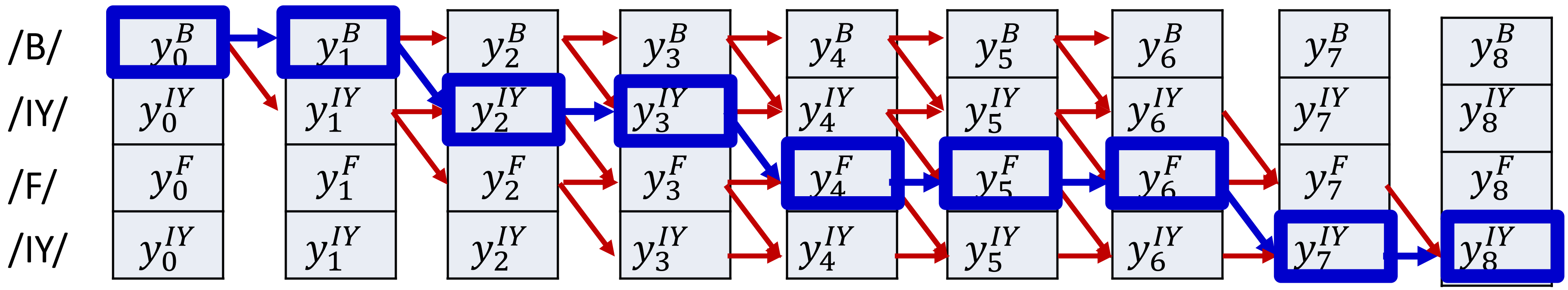
Path Score (probability)



- 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**

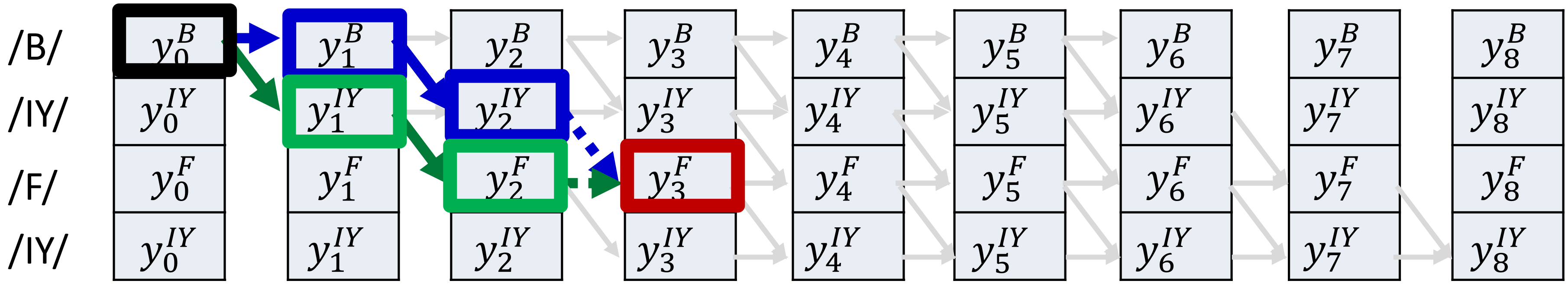
Figure shows a typical end-to-end path. There are an exponential number of such paths. Challenge: Find the path with the highest score (probability)

Explicitly constrain alignment



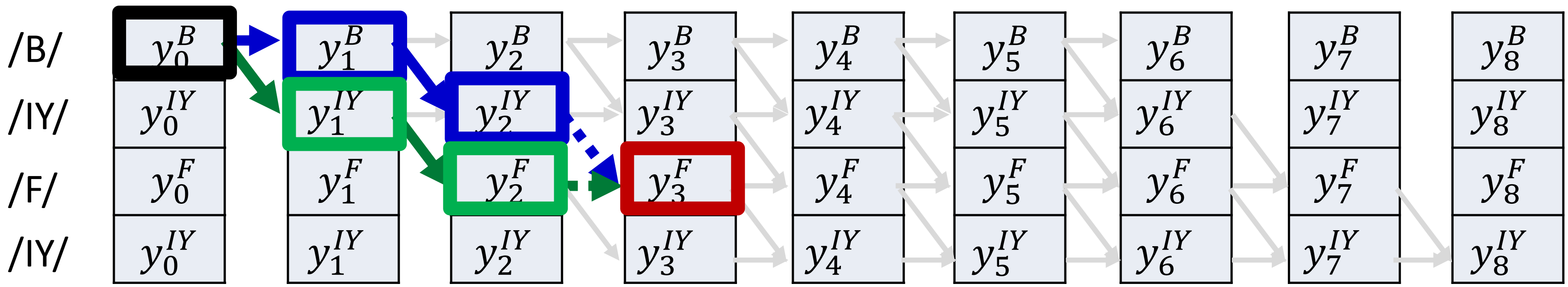
- Find the most probable path from source to sink using any dynamic programming algorithm
  - E.g. The Viterbi algorithm

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



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

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

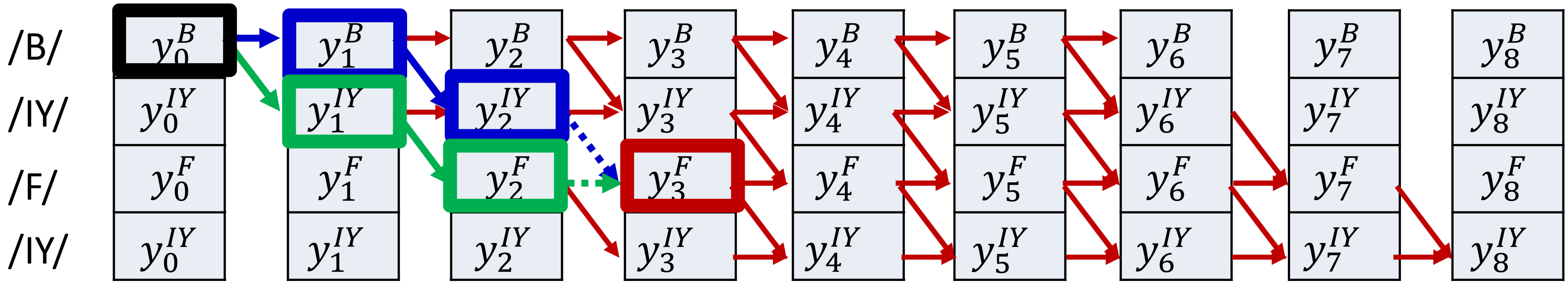
$$BestPath(y_0^B \rightarrow y_3^F) = Best_{Parent \in (y_2^{IY}, y_2^F)}\left(BestPath(y_0^B \rightarrow Parent)\right)y_3^F$$

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

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

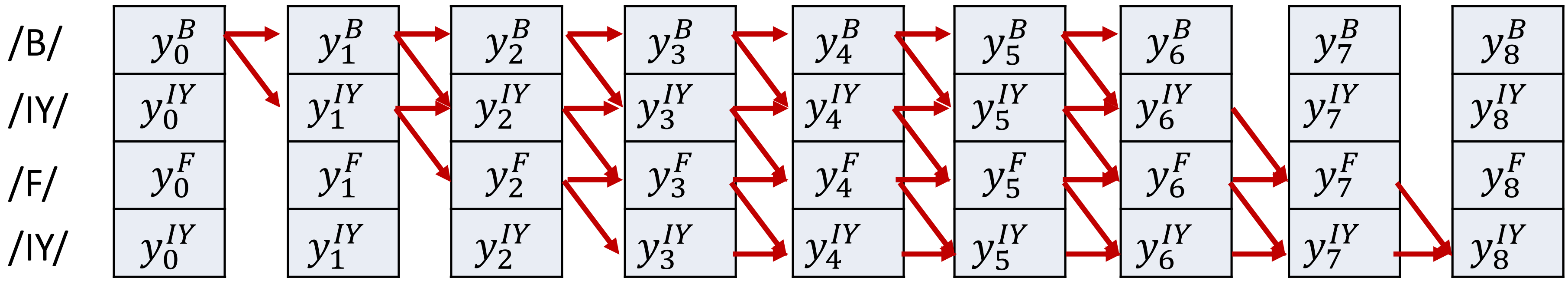
$$BestParent = argmax_{Parent \in (y_2^{IY}, y_2^F)}(Score(BestPath(y_0^B \rightarrow Parent)))$$

Viterbi algorithm



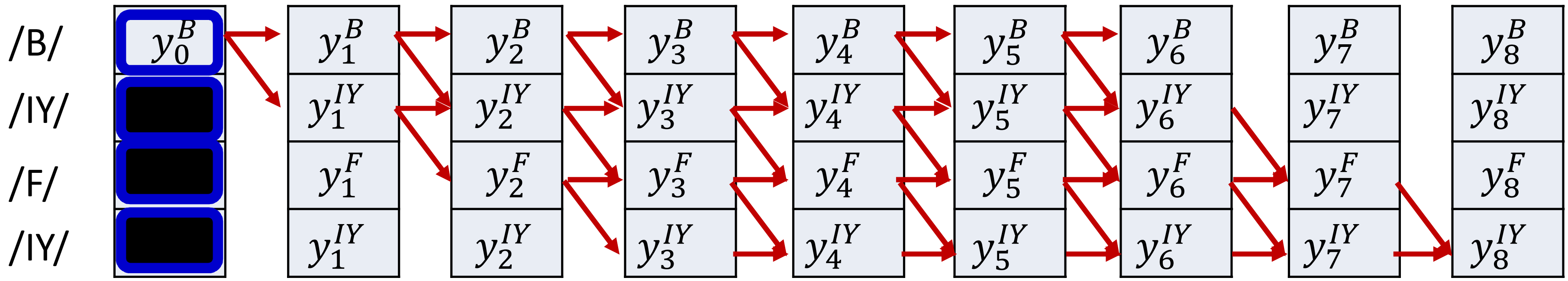
- Dynamically track the best path (and the score of the best path) from the source node to every node in the graph
  - At each node, keep track of
    - The best incoming parent edge
    - The score of the best path from the source to the node through this best parent edge
- Eventually compute the best 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 $X_0 \dots X_t$)
  - 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

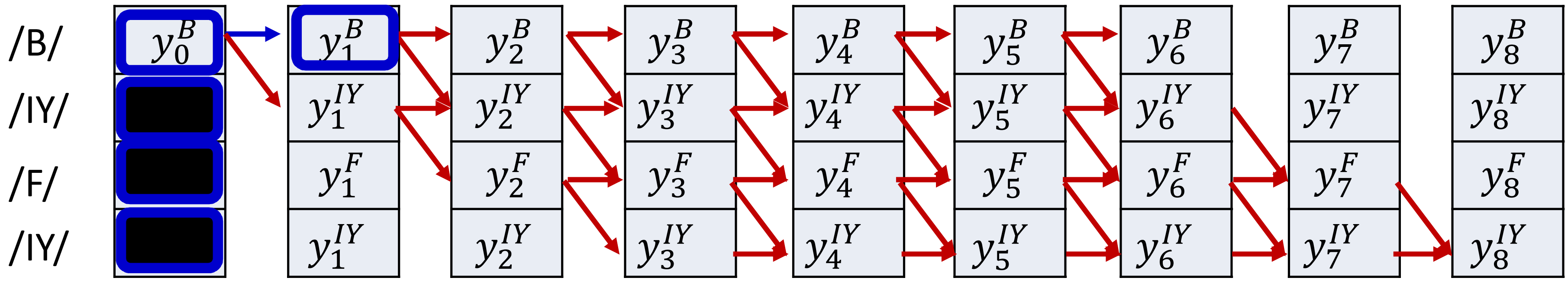


- Initialization:

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

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

Viterbi algorithm



- Initialization:

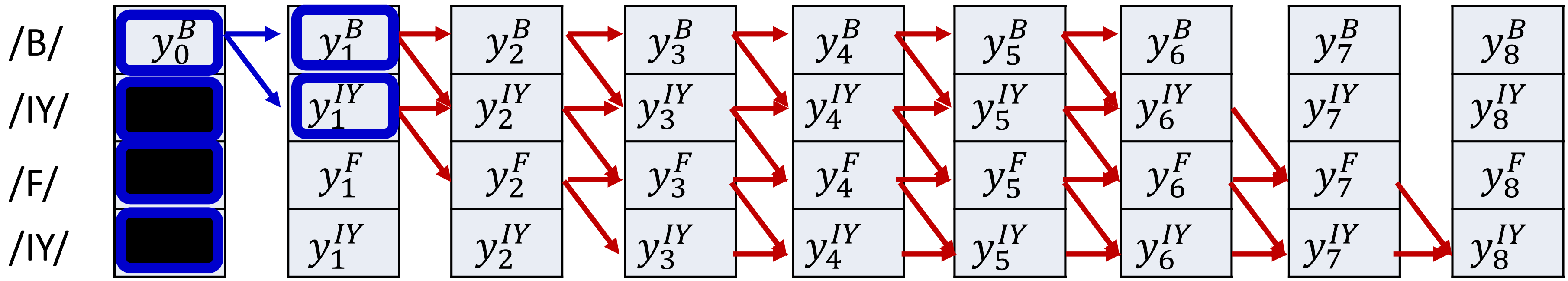
$$BP(0,i) = \ null,\ \ i = 0 \ldots K-1$$

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

- 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:

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

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

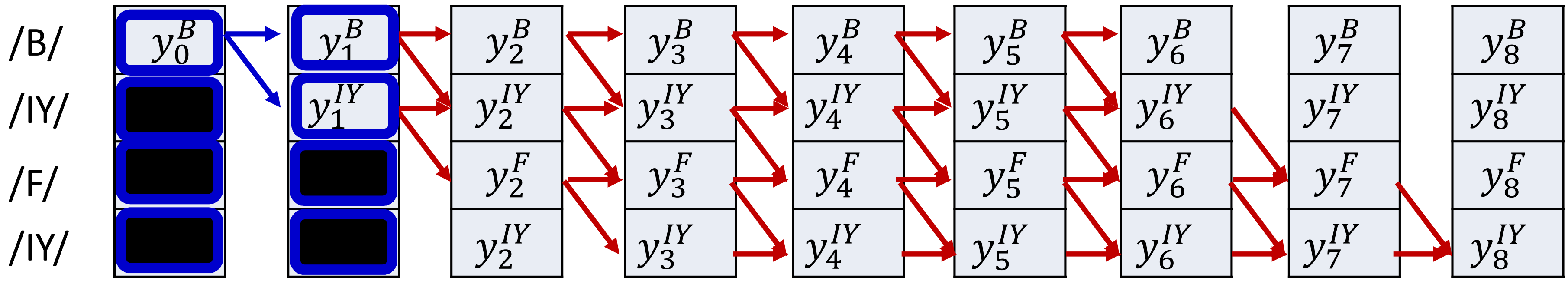
- for $t \;\; = \;\; 1 \dots T-1$

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

for $l \;\; = 1 \dots K-1$

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

Viterbi algorithm



- Initialization:

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

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

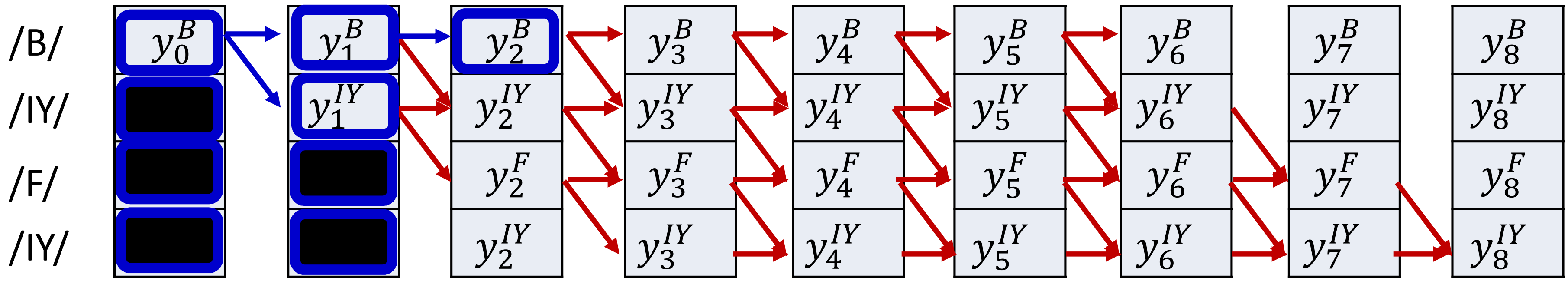
- for $t = 1 \ldots T-1$

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

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:

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

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

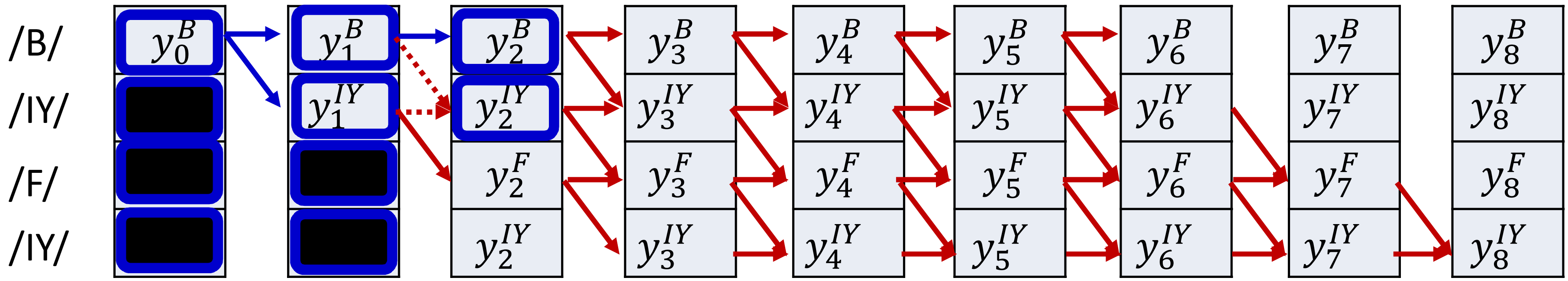
- for $t \ = \ 1 \dots T-1$

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

for $l \ = 1 \dots K-1$

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

Viterbi algorithm



- Initialization:

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

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

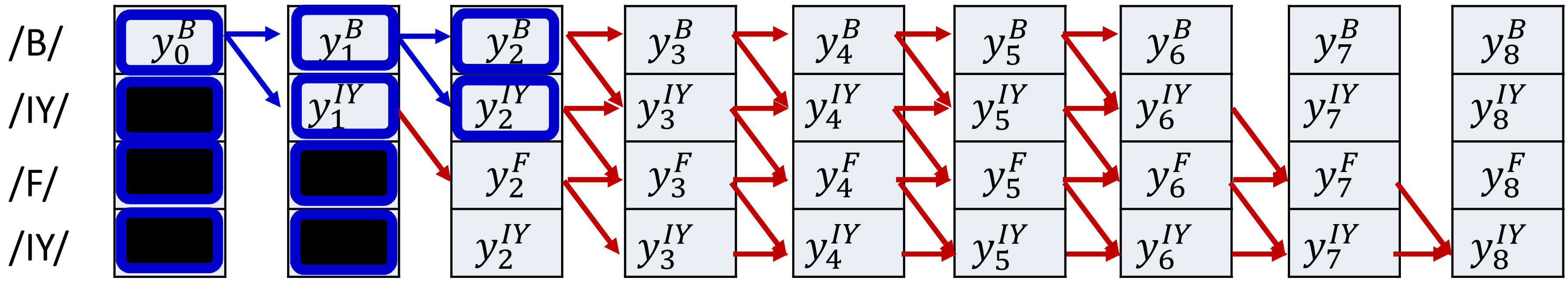
- for $t = 1 \ldots T-1$

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

for $l = 1 \ldots K-1$

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

Viterbi algorithm



- Initialization:

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

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

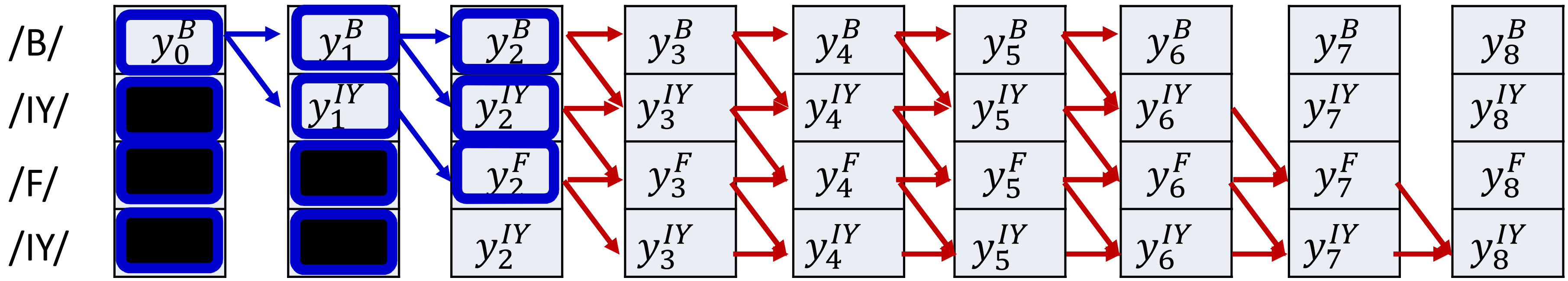
- for $t = 1 \ldots T-1$

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

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:

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

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

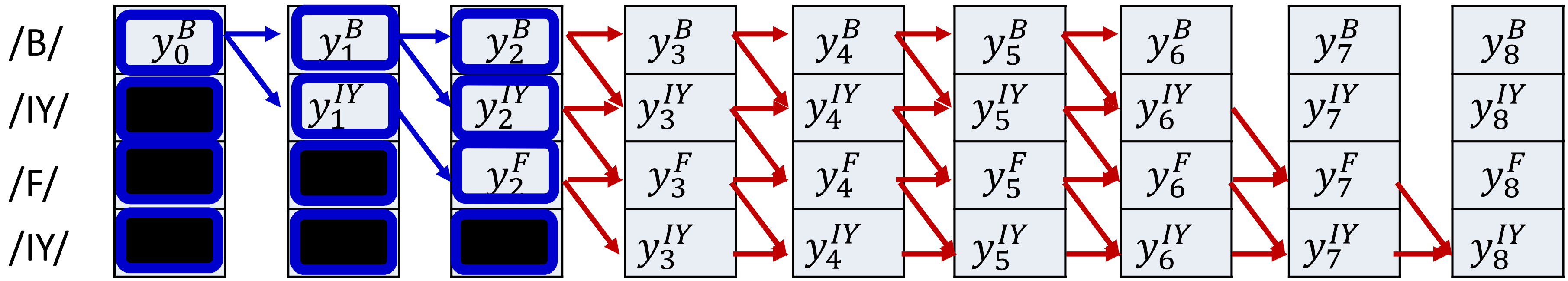
- for $t \ = \ 1 \dots T-1$

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

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:

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

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

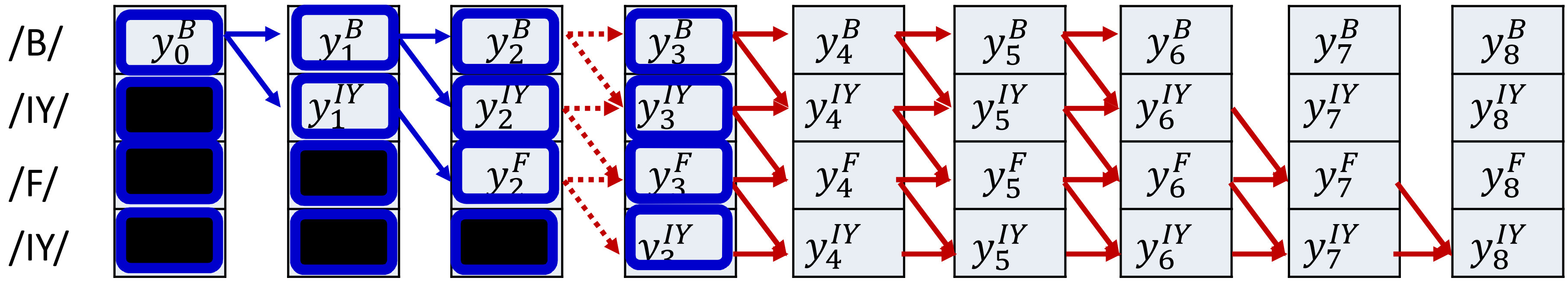
- for $t = 1 \dots T-1$

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

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:

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

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

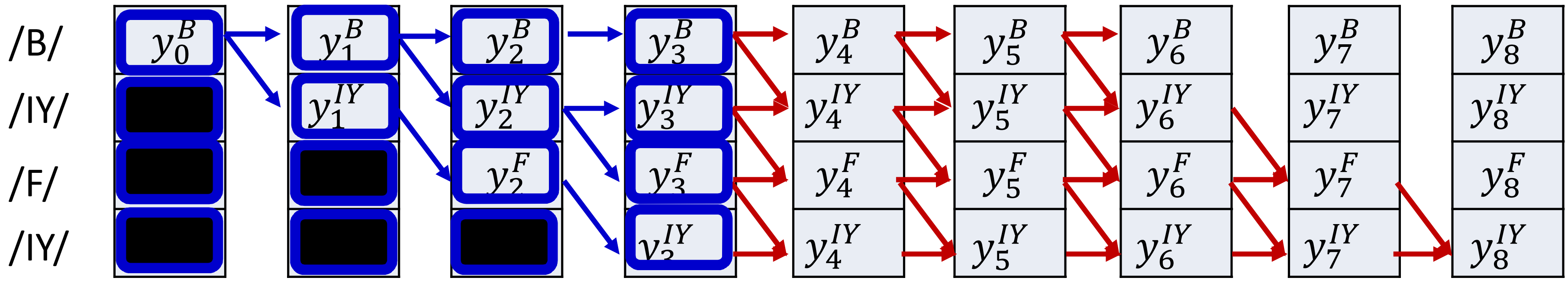
- for $t \; = \; 1 \ldots T-1$

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

for $l \; = 1 \ldots K-1$

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

Viterbi algorithm



- Initialization:

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

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

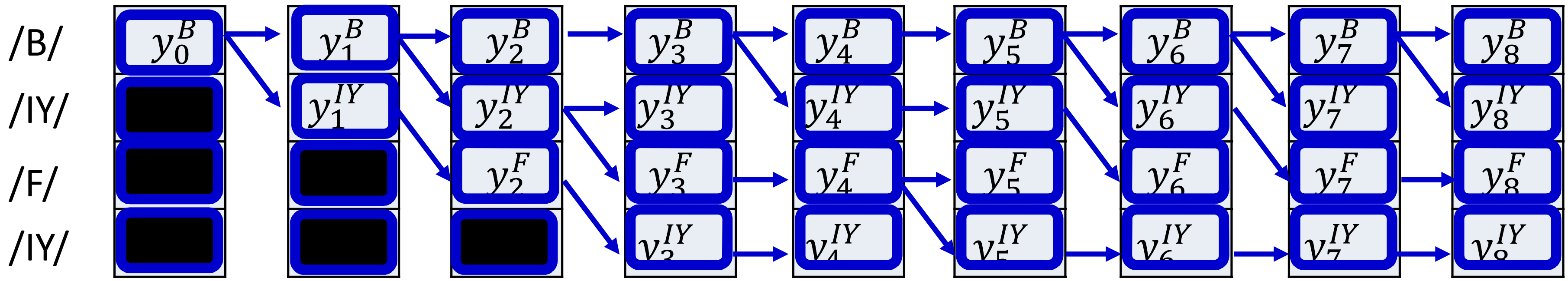
- for $t = 1 \ldots T-1$

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

for $l = 1 \ldots K-1$

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

Viterbi algorithm



- Initialization:

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

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

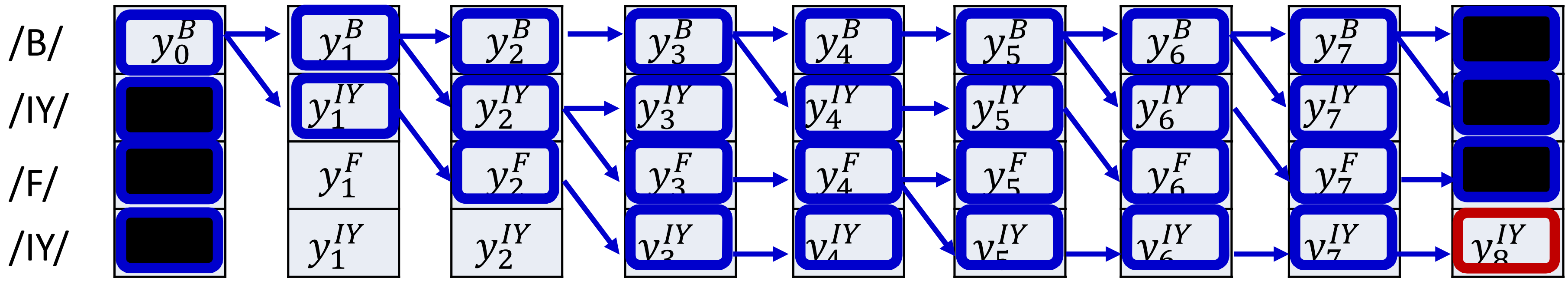
- for $t \ = \ 1 \ldots T-1$

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

for $l \ = 1 \ldots K-1$

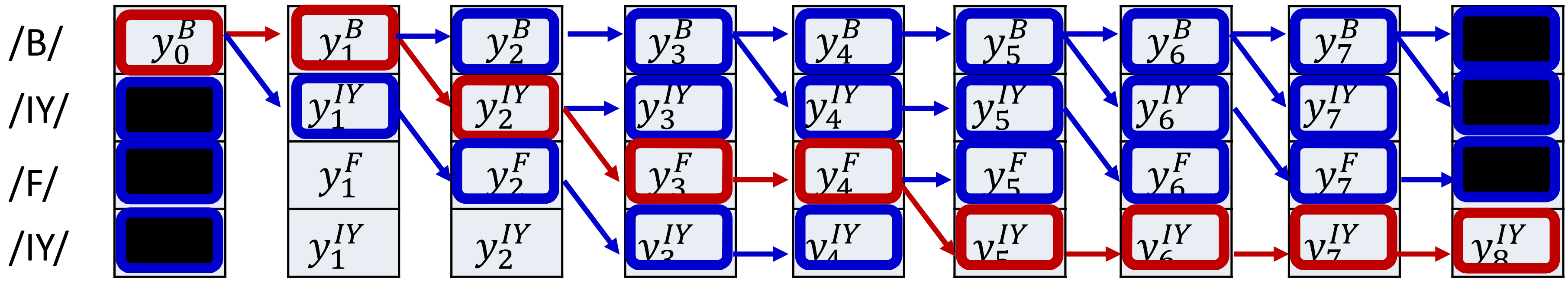
- $BP(t,l) = \begin{pmatrix} l-1: \ \ if \ \left(Bscr(t-1,l-1) > \ Bscr(t-1,l)\right) \ l-1; \\ l: else \end{pmatrix}$
- $Bscr(t,l) \ = \ Bscr(BP(t,l)) \ \times y_t^{S(l)}$

Viterbi algorithm



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

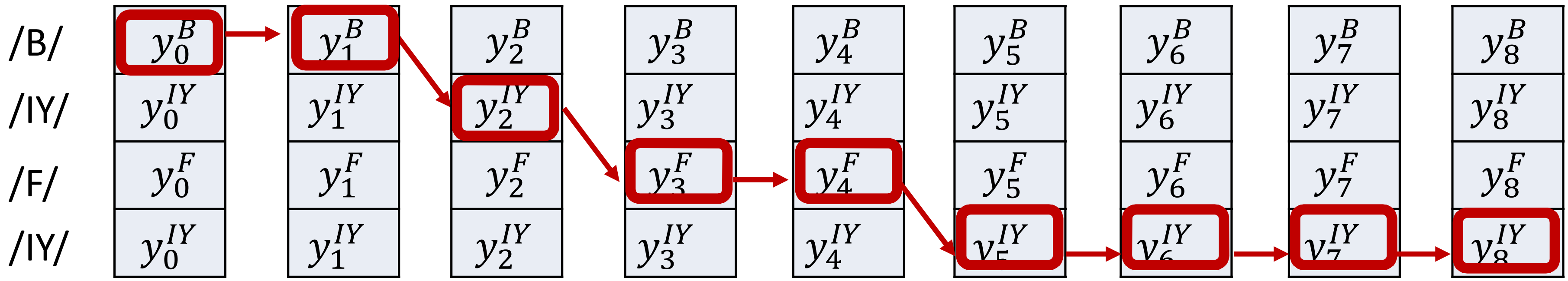
Viterbi algorithm



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

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

Viterbi algorithm



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

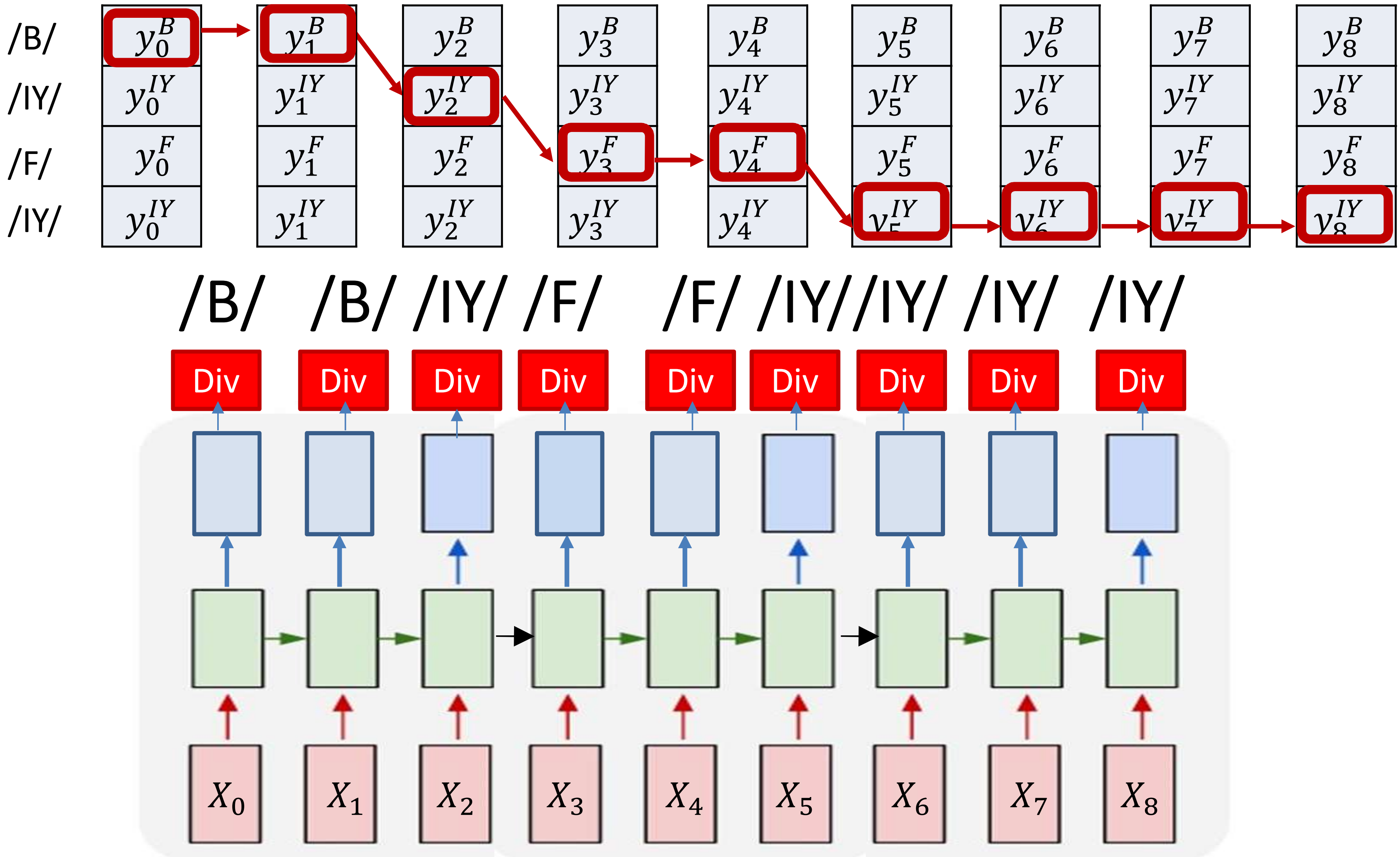
  $s(t-1) \;=\; BP(s(t))$

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

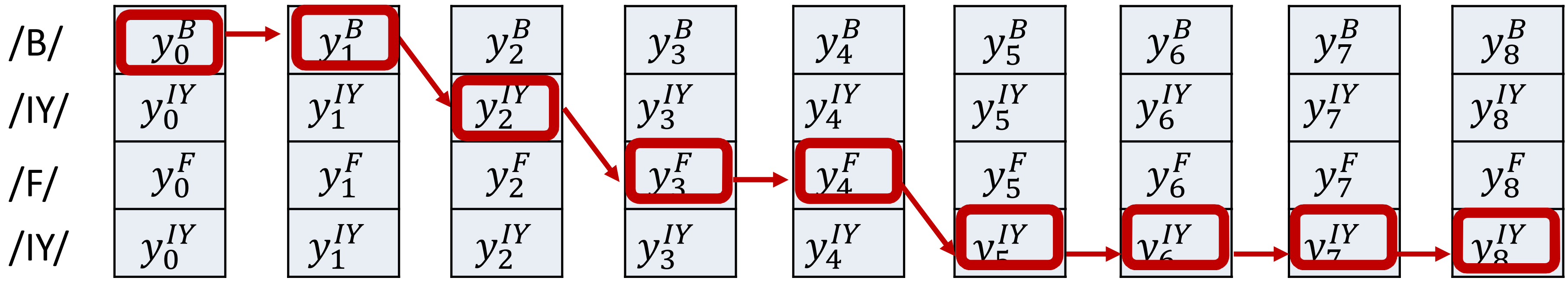
VITERBI

```
#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



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

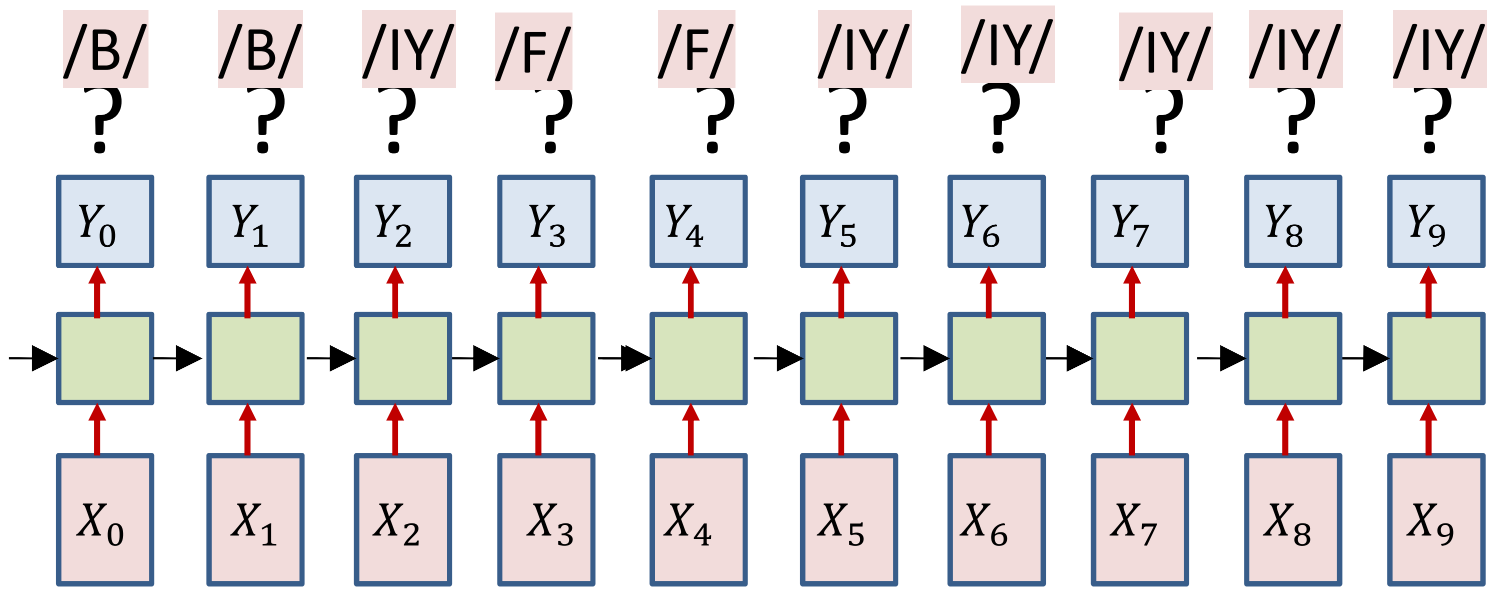
$$DIV = \sum_t Xent\left(Y_t, symbol_t^{bestpath}\right) = -\sum_t \log Y\left(t, symbol_t^{bestpath}\right)$$

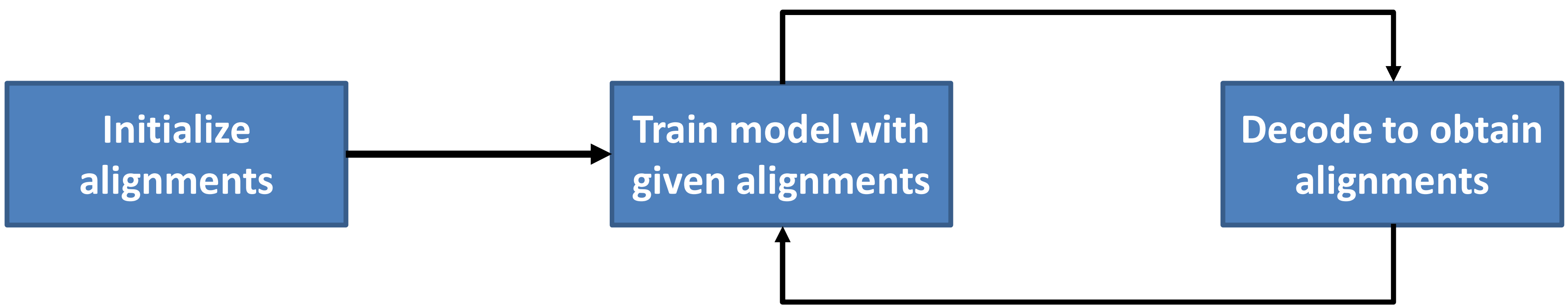
- The gradient w.r.t the t-th output vector Y_t

$$\nabla_{Y_t} DIV = \begin{bmatrix} 0 & 0 & \cdots & \dfrac{-1}{Y\left(t, symbol_t^{bestpath}\right)} & 0 & \cdots & 0 \end{bmatrix}$$

  – 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

- Option 1:
  - Determine alignments for every training instance
  - Train model (using SGD or your favorite approach) on the entire training set
  - Iterate

- Option 2:
  - During SGD, for each training instance, find the alignment during the forward pass
  - Use in backward pass

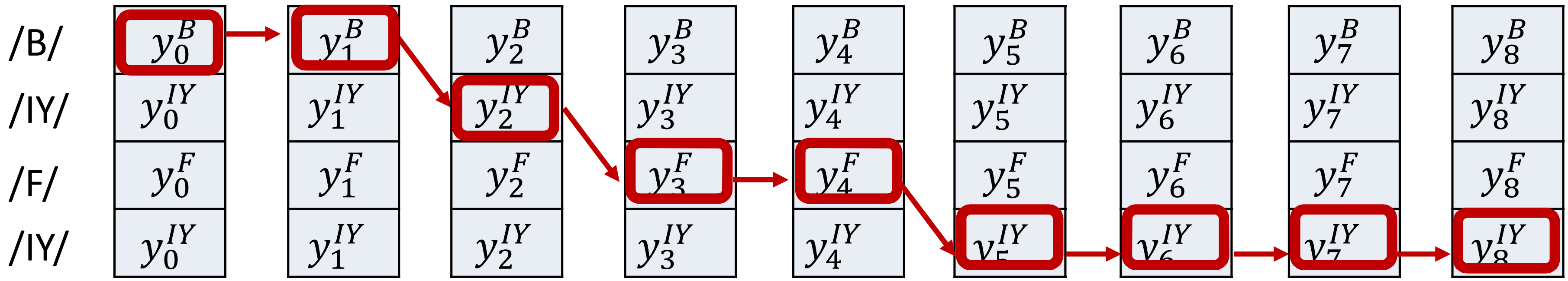
Iterative update: Problem

- 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

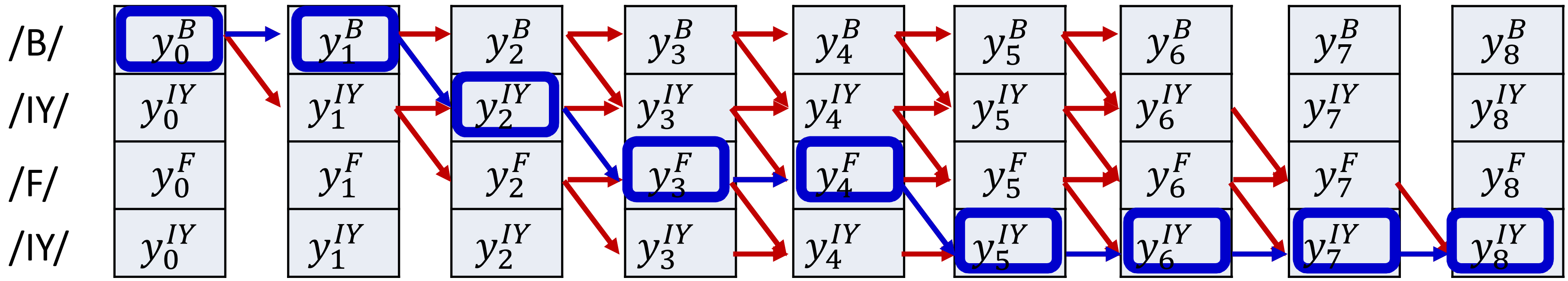


- We *commit* to the single "best" estimated alignment
  - The *most likely* alignment

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

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

The reason for suboptimality



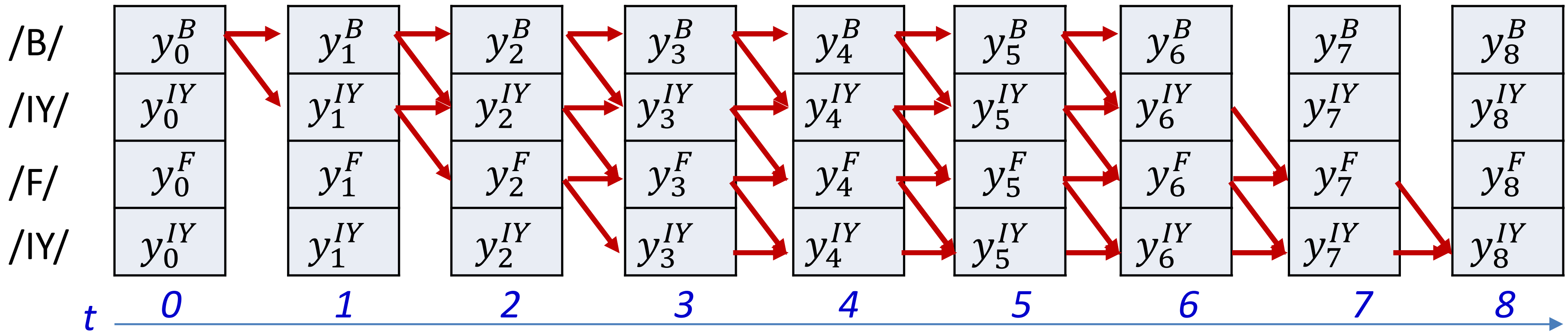
- We *commit* to the single "best" estimated alignment
  - The *most likely* alignment

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

  - 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

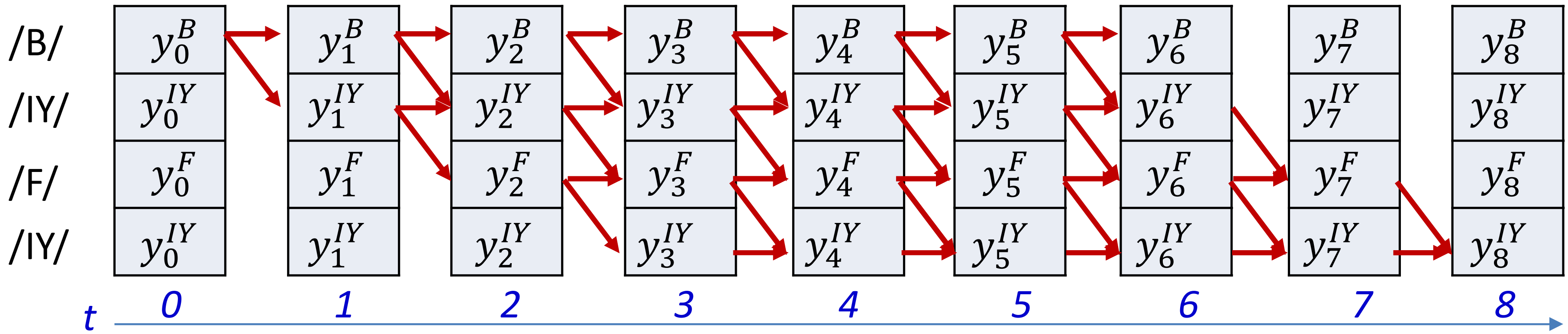


- Instead of only selecting the most likely alignment, use the statistical expectation over *all* possible alignments

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

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

The expectation over *all* alignments



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

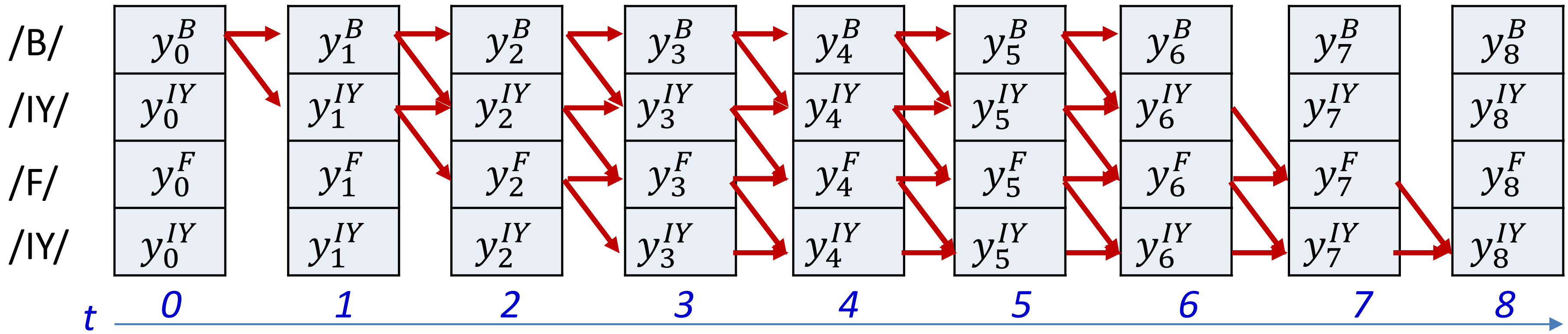
- Using the linearity of expectation

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

  – 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 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

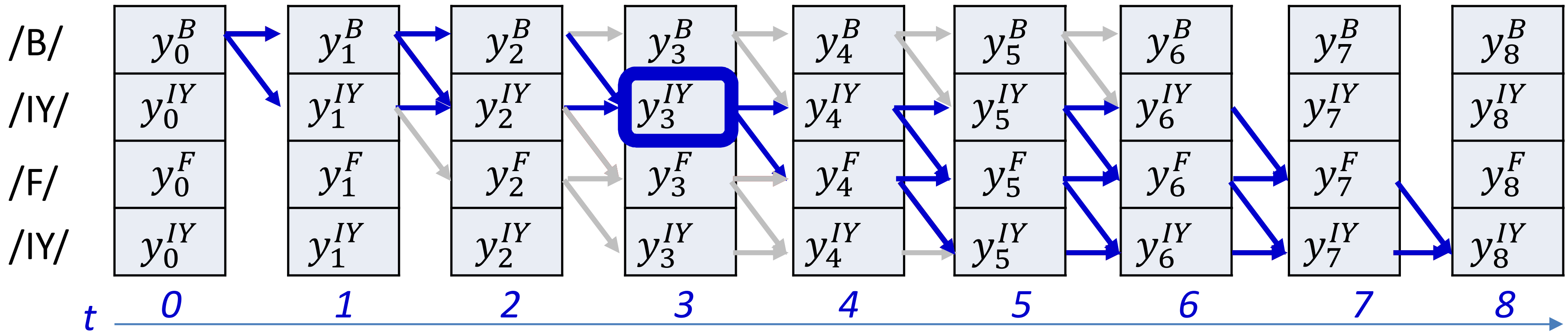
•

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

– 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

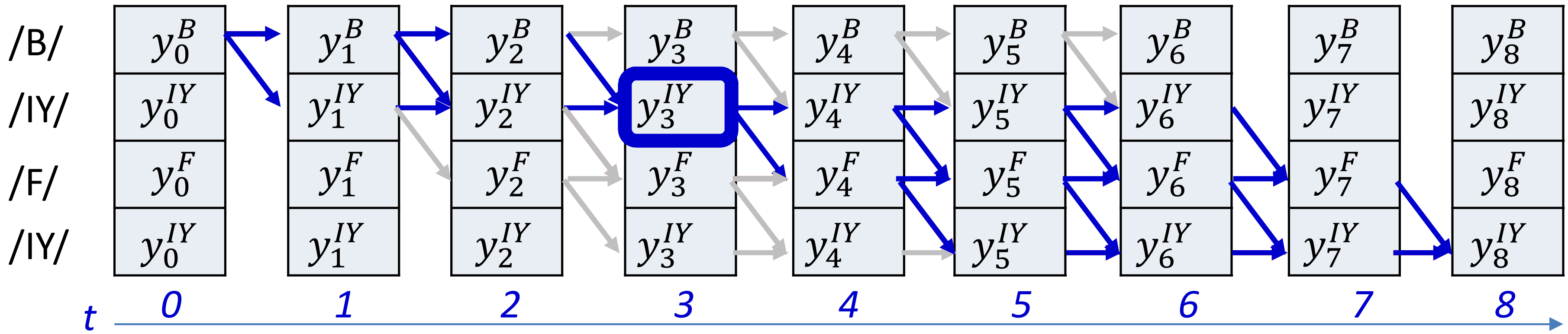


$$P(s_t = S_r|\mathbf{S},\mathbf{X}) \propto P(s_t = S_r,\mathbf{S}|\mathbf{X})$$

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

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

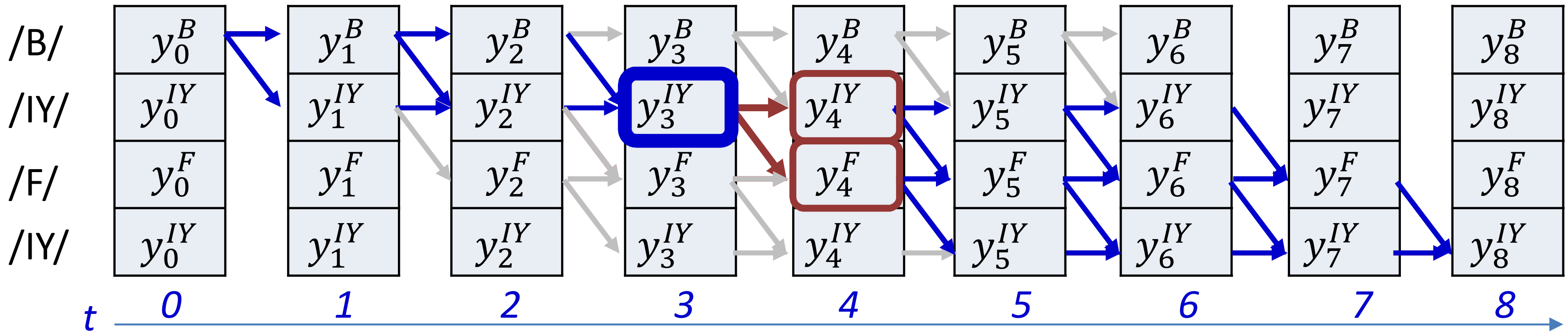
A posteriori probabilities of symbols



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



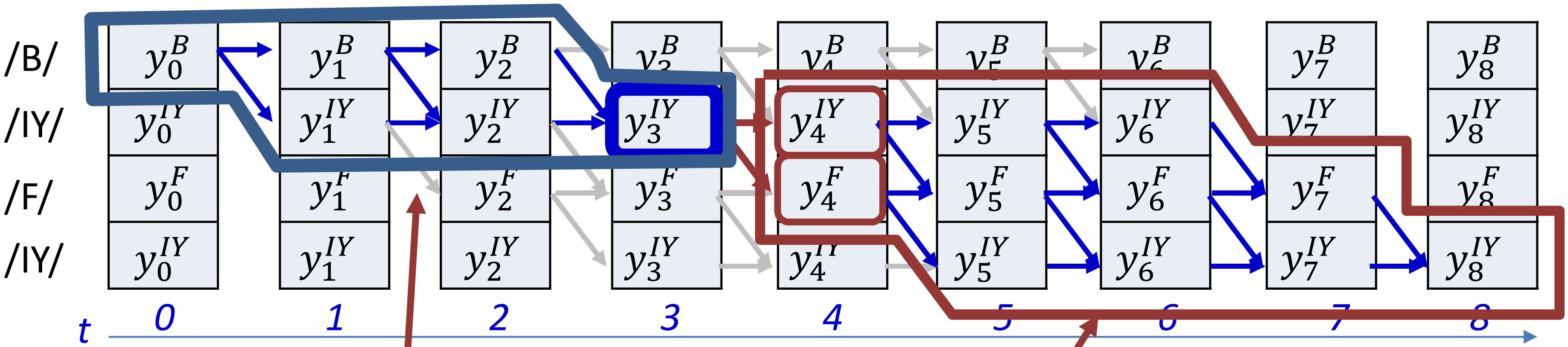
- $P(s_t = S_r, \mathbf{S}|\mathbf{X})$ can be decomposed as

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

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

- 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



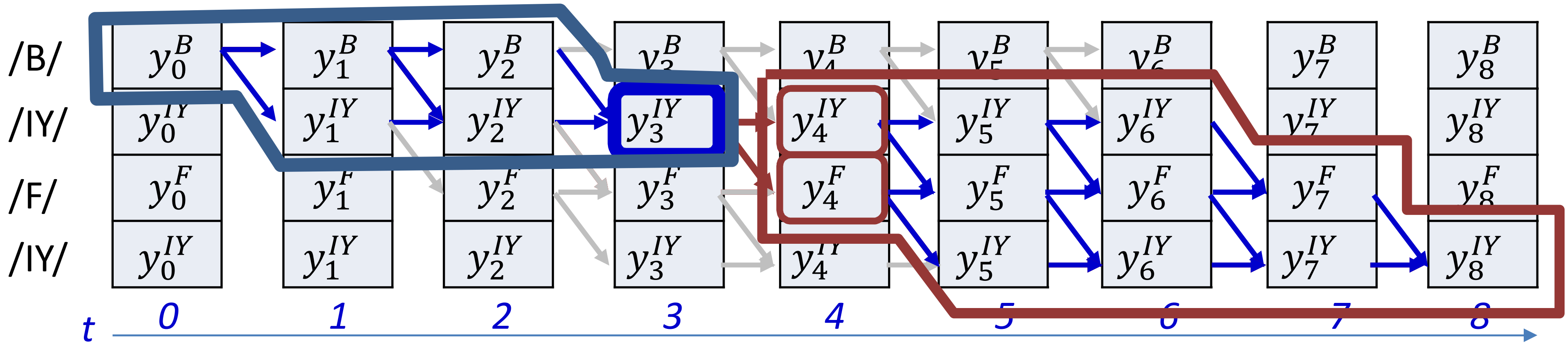
- $P(s_t = S_r, \mathbf{S}|\mathbf{X})$ can be decomposed as

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

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

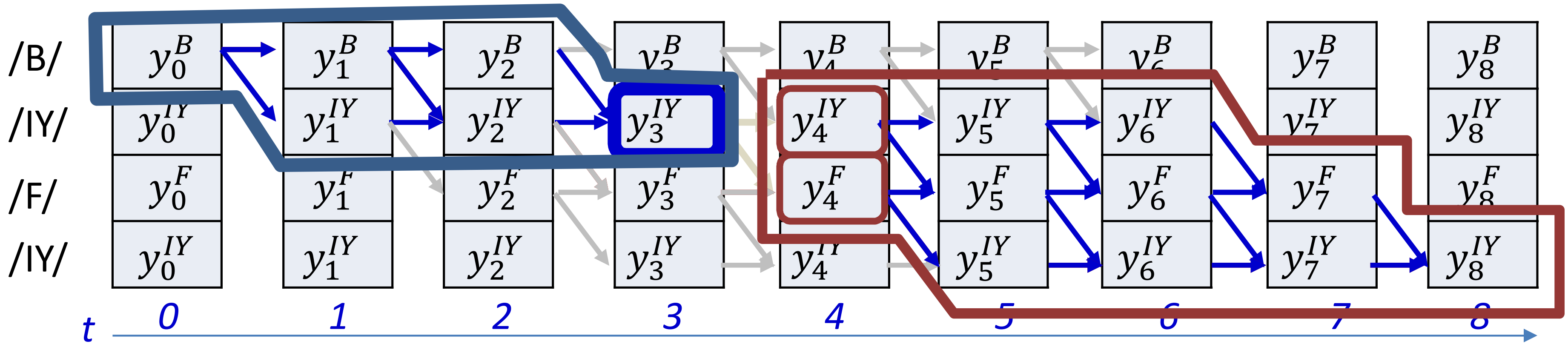
- Where $succ(S_r)$ is a symbol that can follow S_r in a sequence
  - Here it is either S_r or S_{r+1} (red blocks in figure)
  - The equation literally says that after the blue block, either of the two red arrows may be followed

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

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

- Using Bayes Rule

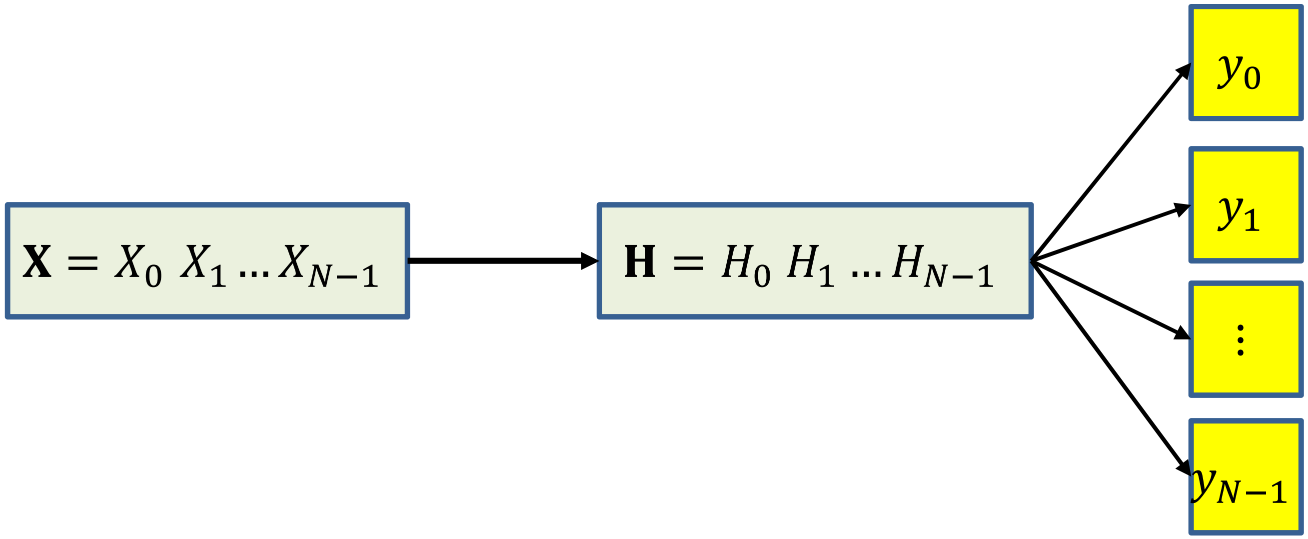
$$= P(S_1 \ldots S_r, s_t = S_r\ |\mathbf{X}) P(s_{t+1} \in succ(S_r), succ(S_r), \ldots, S_K\ |S_1 \ldots S_r, s_t = S_r \mathbf{X})$$

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

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

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

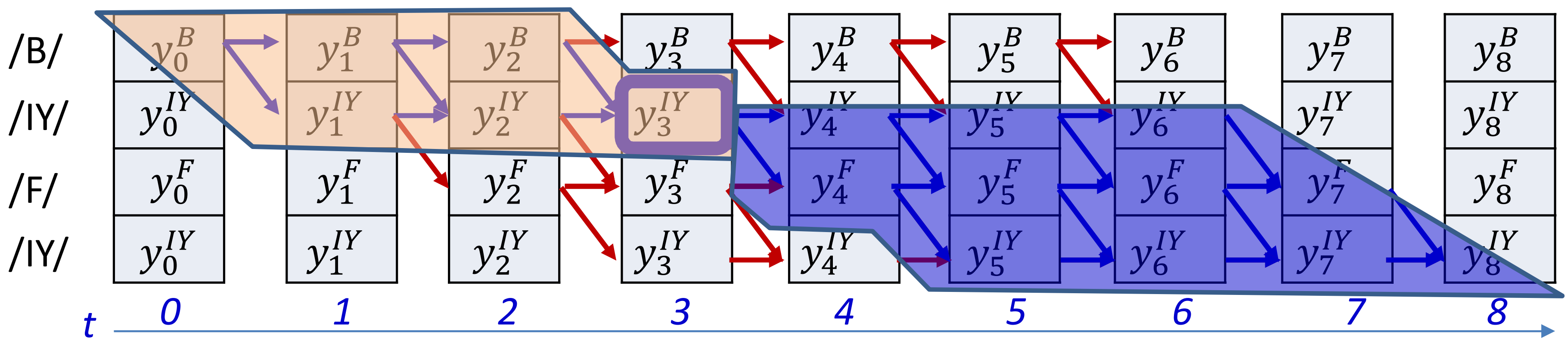
Conditional independence



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

- Since $\mathbf{H}$ is deterministically derived from $\mathbf{X}$, $y_0, y_1, \dots y_{N-1}$ 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

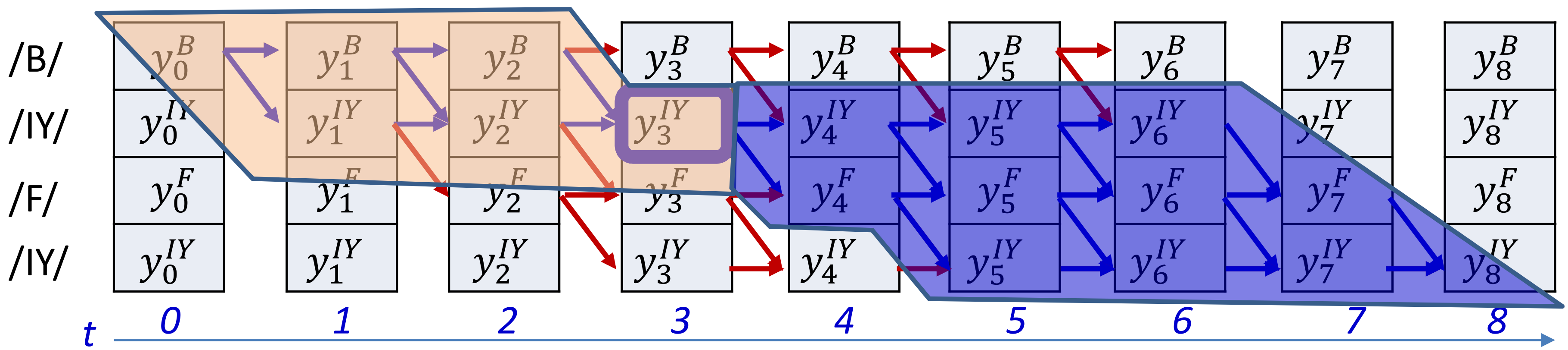
A posteriori symbol probability



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

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

A posteriori symbol probability

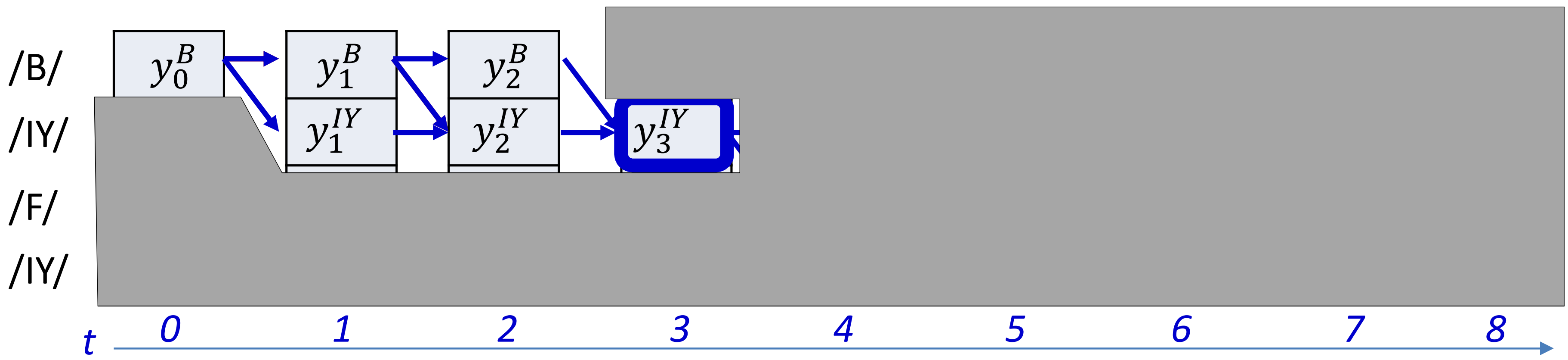


$P(s_t = S_r, \mathbf{S}|\mathbf{X})$

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

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

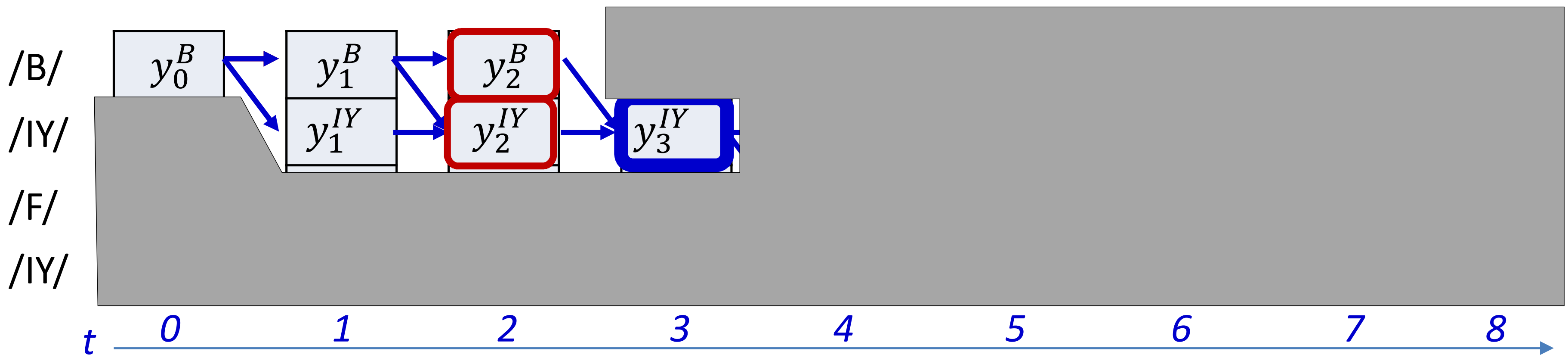
Computing $\alpha(t,r)$: Forward algorithm



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

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

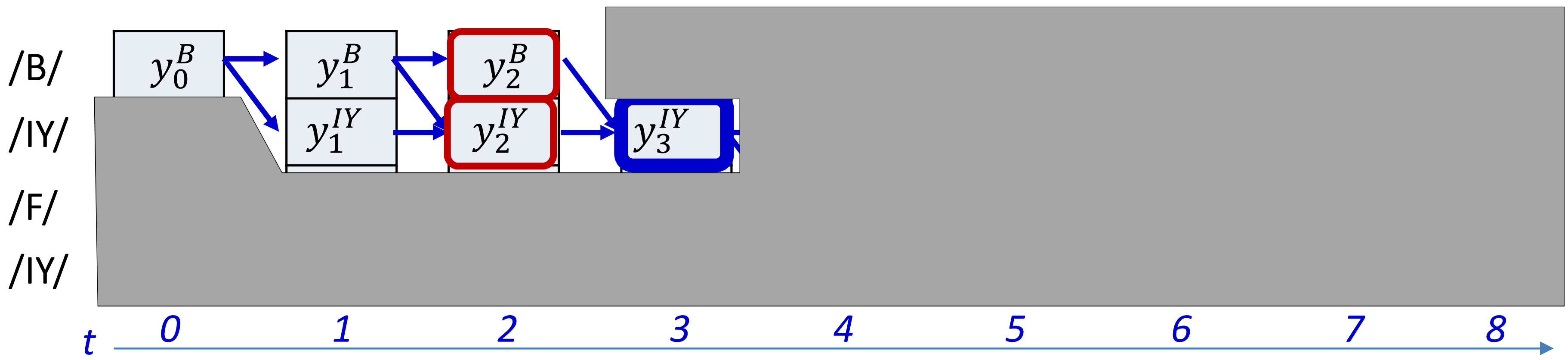
Computing $\alpha(t, r)$: Forward algorithm

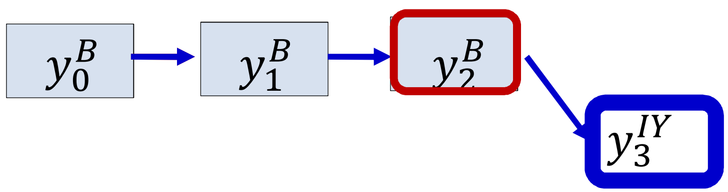


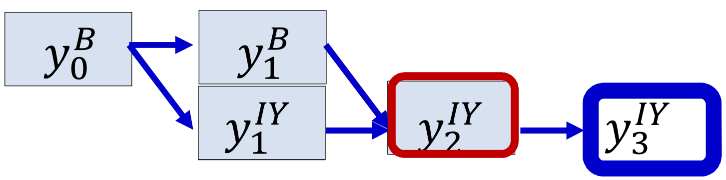
$$\alpha(3, IY) = P(S_1..S_r, s_t = S_r|\mathbf{X})$$

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

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





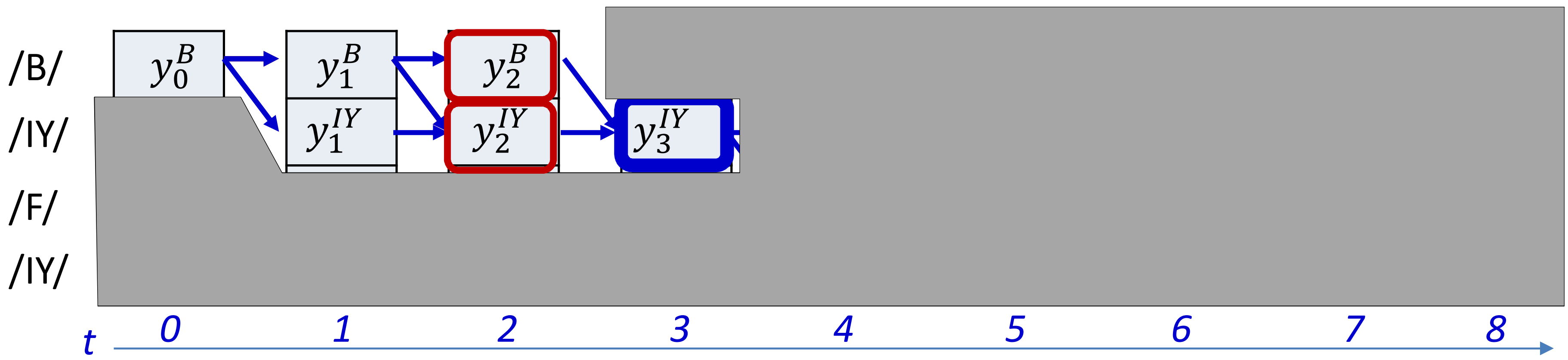


$$\alpha(t,r) = P(S_1..S_r, s_t = S_r|\mathbf{X})$$

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

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

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



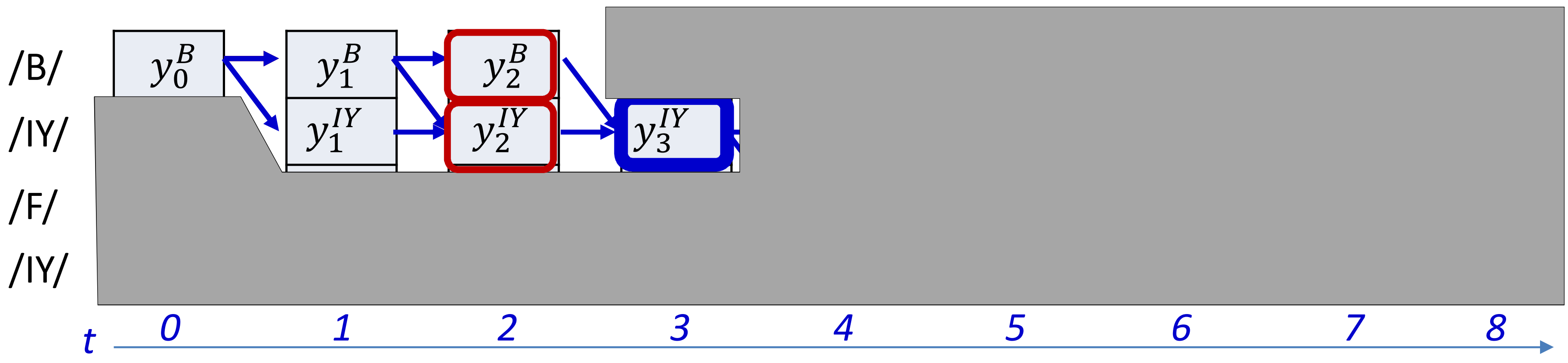
$$\alpha(t, r) = P(S_1..S_r, s_t = S_r|\mathbf{X})$$

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

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

  – 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



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

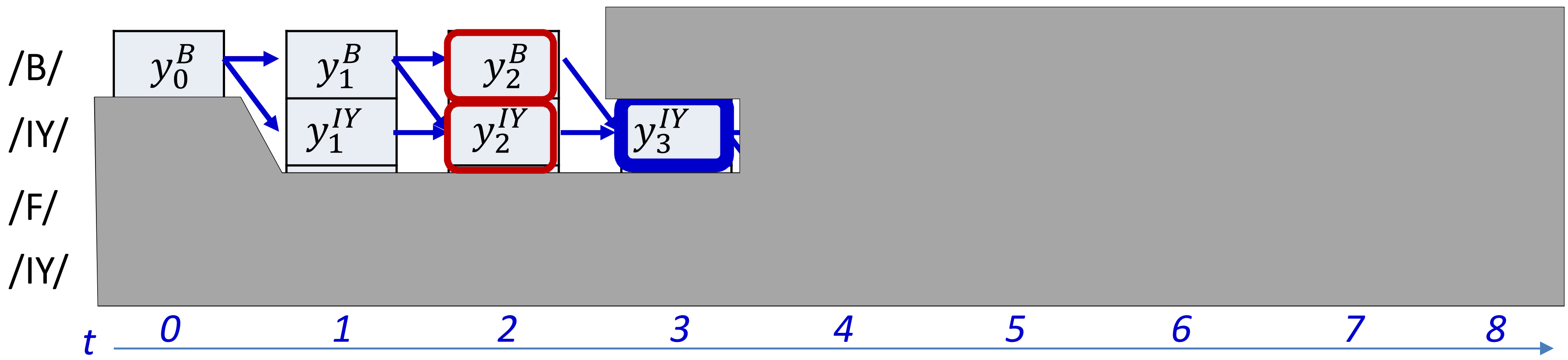
- The $\alpha(t,r)$ is the total probability of the subgraph shown
- We can marginalize out the symbol at time t-1

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

- Using the conditional independence assumed

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

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



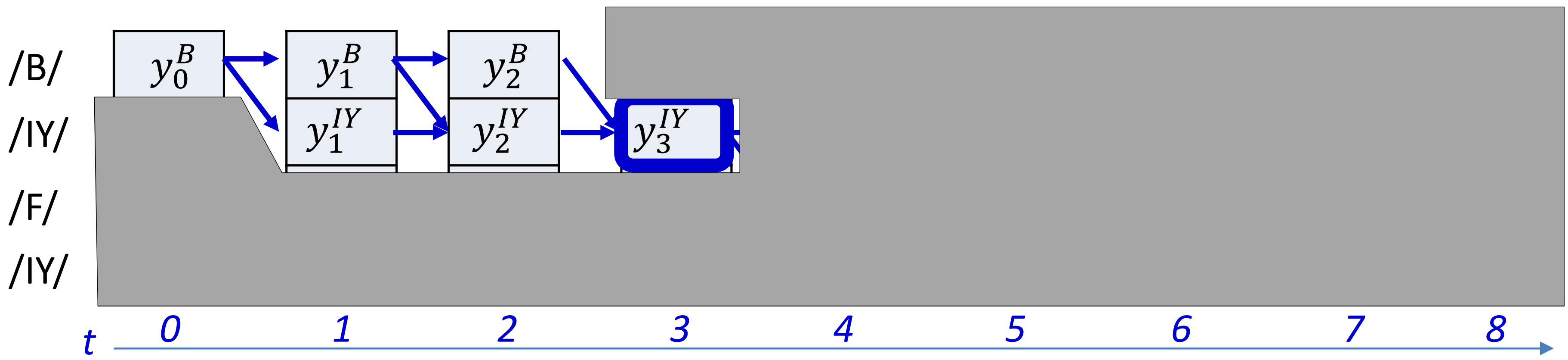
$$\alpha(t,r) = P(S_1..S_r, s_t = S_r|\mathbf{X})$$

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

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

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

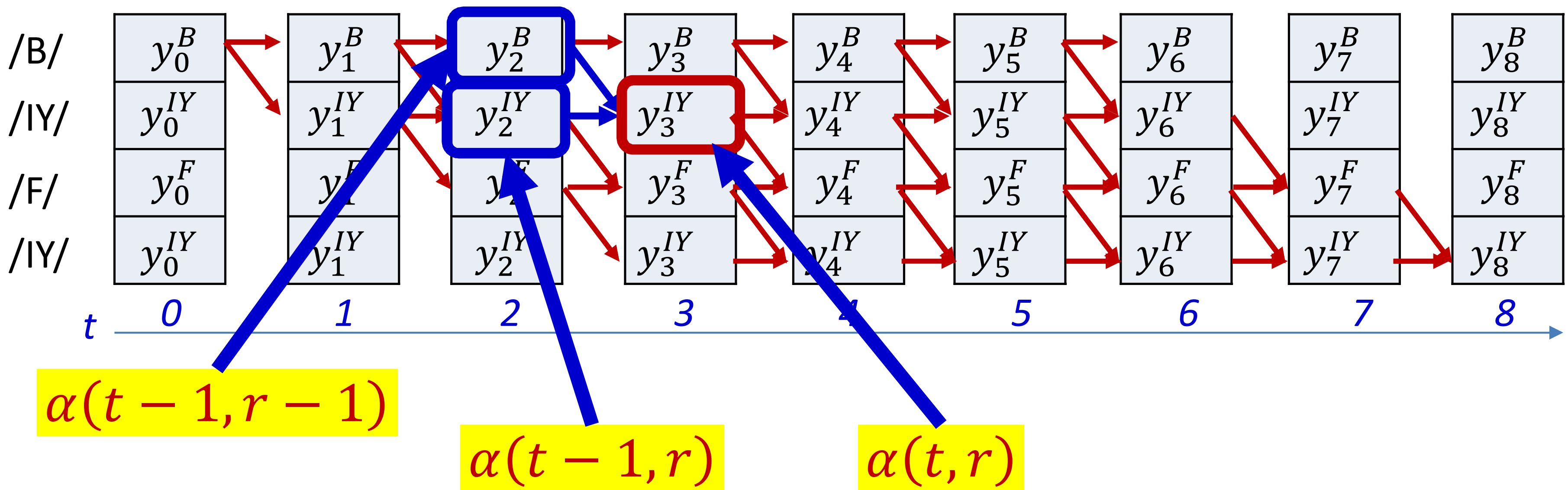
Forward algorithm



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

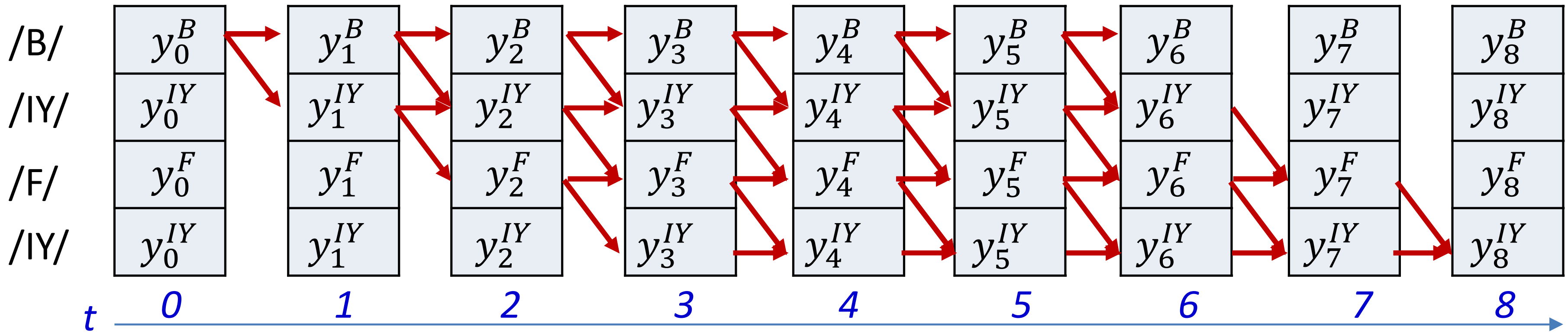
- The $\alpha(t,r)$ is the total probability of the subgraph shown

Forward algorithm



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

Forward algorithm



- Initialization:

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

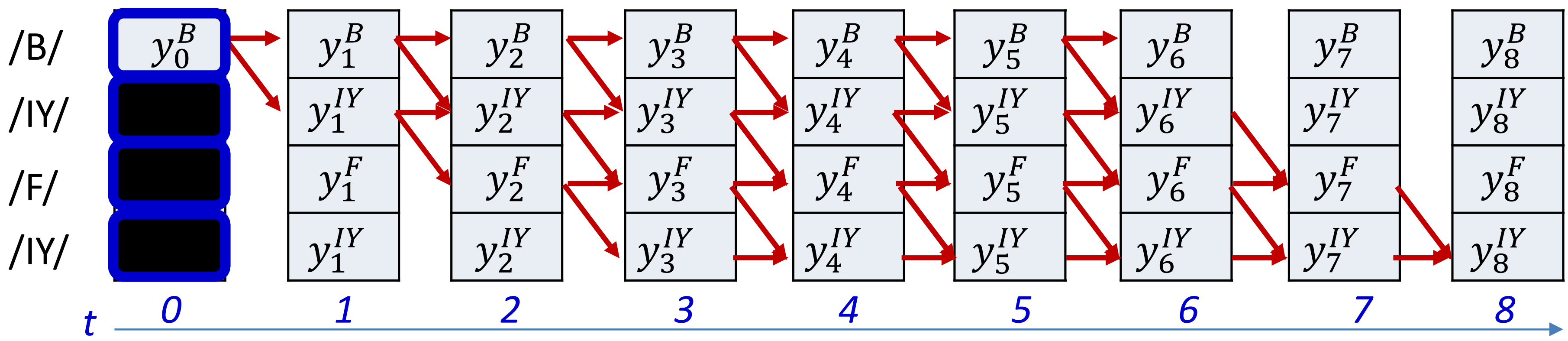
- for $t = 1 \dots T-1$

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

for $l = 2 \dots K$

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

Forward algorithm



- Initialization:

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

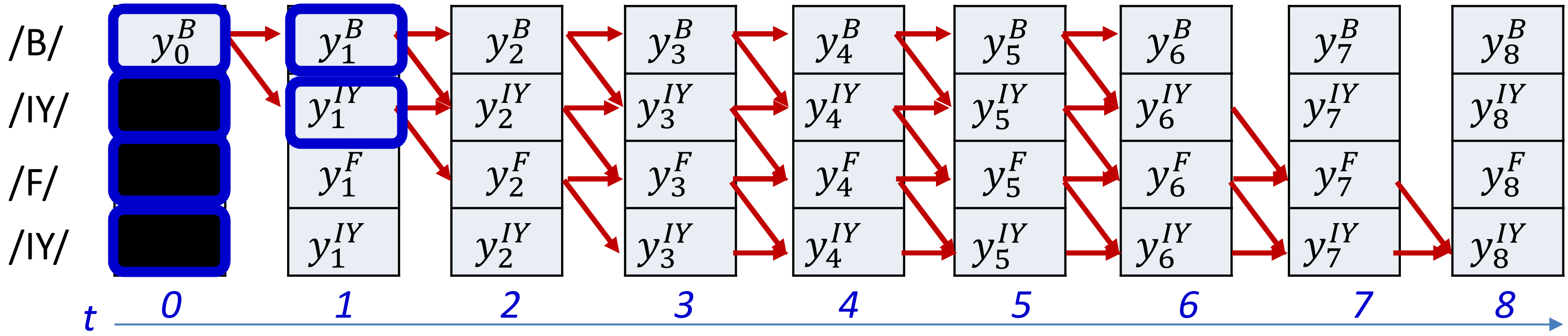
- for $t = 1 \ldots T-1$

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

for $l = 2 \ldots K$

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

Forward algorithm



- Initialization:

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

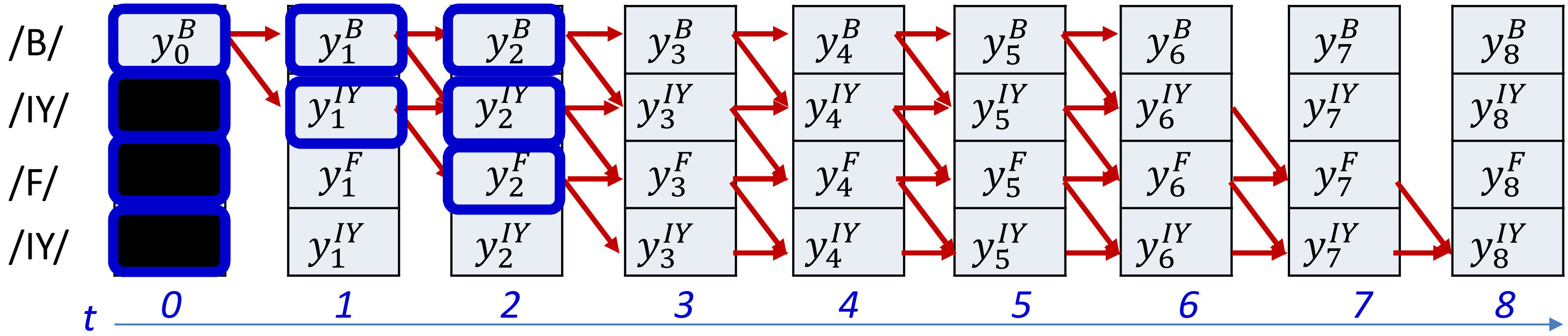
- for $t = 1 \ldots T-1$

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

for $l = 2 \ldots K$

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

Forward algorithm



- Initialization:

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

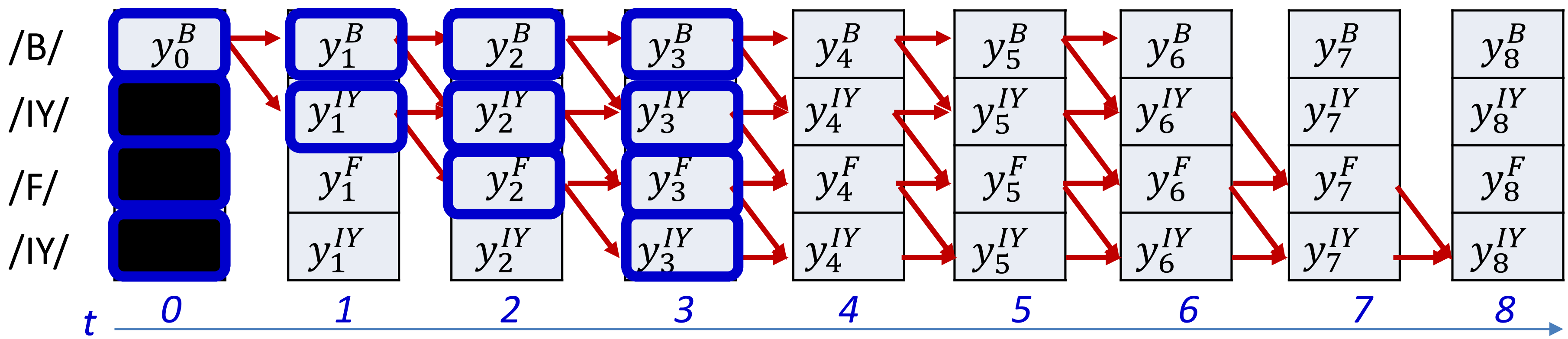
- for $t = 1 \ldots T-1$

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

for $l = 2 \ldots K$

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

Forward algorithm



- Initialization:

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

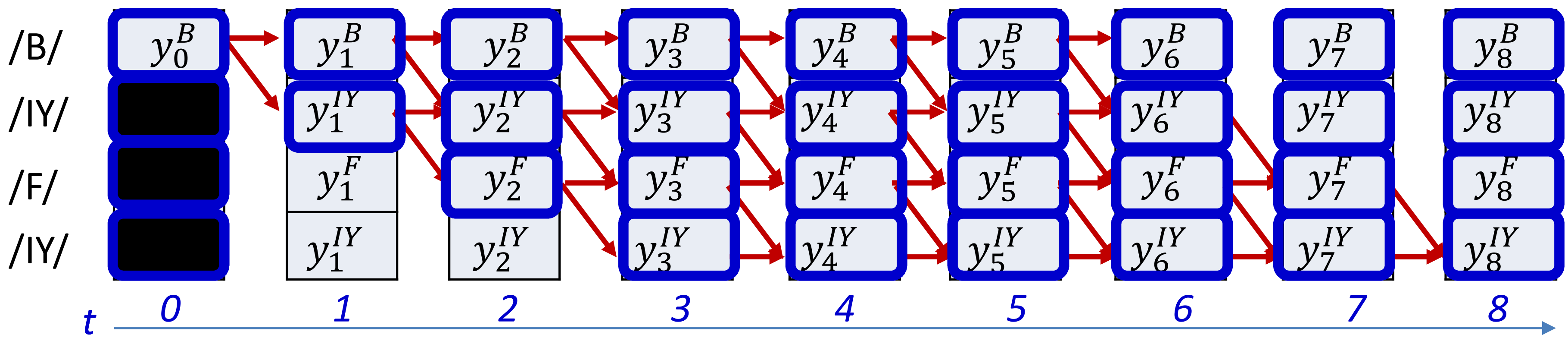
- for $t = 1 \ldots T-1$

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

  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 $l = 2 \ldots K$

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

In practice..

- The recursion

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

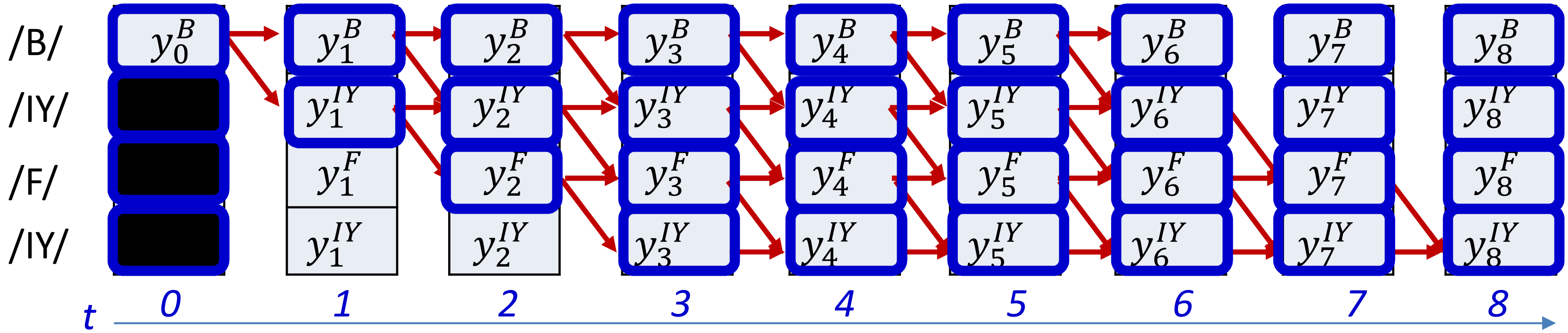
  will generally underflow

- Instead we can do it in the *log* domain

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

  – This can be computed entirely without underflow

Forward algorithm: Alternate statement



- The algorithm can also be stated as follows which separates the graph probability from the observation probability. This is needed to compute derivatives
- Initialization:

$$\hat{\alpha}(0,1) = 1, \quad \hat{\alpha}(0,r) = 0, \ r > 1$$

$$\alpha(0,r) = \hat{\alpha}(0,r) y_0^{S(r)}, \qquad 1 \le r \le K$$

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

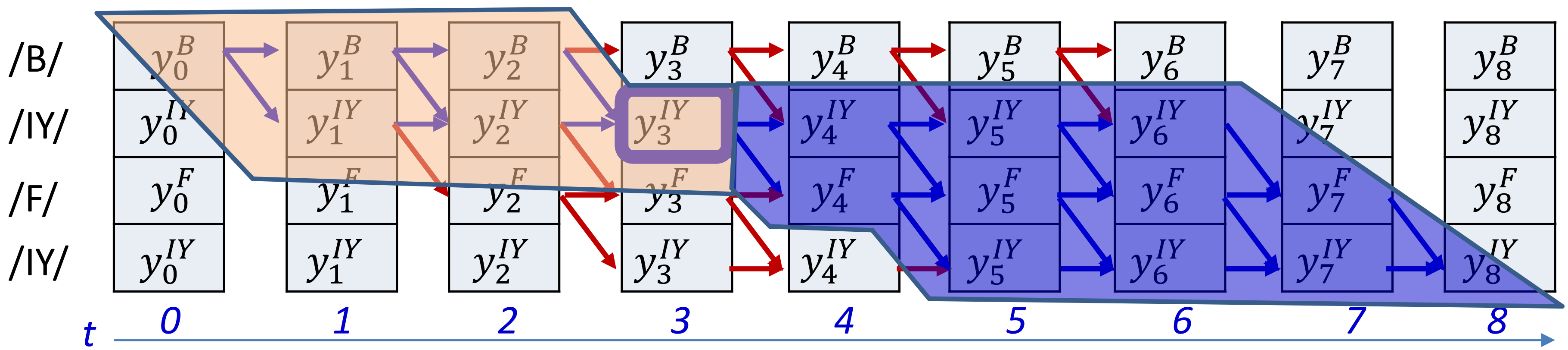
  $\hat{\alpha}(t,1) \ = \alpha(t-1,1)$

  for $l \ = 2 \ldots K$

    - $\hat{\alpha}(t,l) \ = \alpha(t-1,l) + \alpha(t-1,l-1)$

  $$\alpha(t,r) = \hat{\alpha}(t,r) y_t^{S(r)}, \qquad 1 \le r \le K$$

A posteriori symbol probability



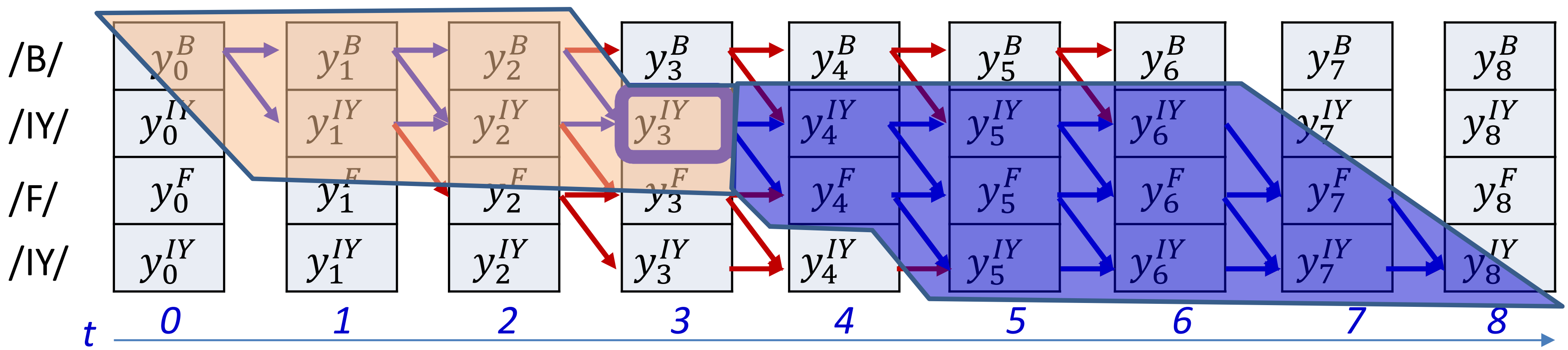
$P(s_t = S_r, \mathbf{S}|\mathbf{X})$

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

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

We have seen how to compute this

A posteriori symbol probability

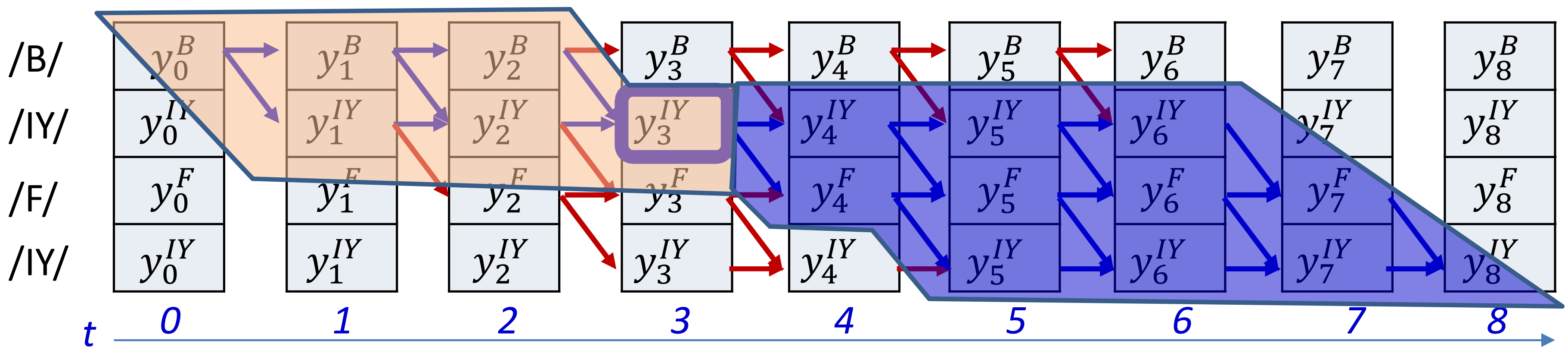


$$P(s_t = S_r, \mathbf{S}|\mathbf{X}) = \boxed{\alpha(t,r)} P(s_{t+1} \in succ(S_r), succ(S_r), \ldots, S_K |\mathbf{X})$$

- 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

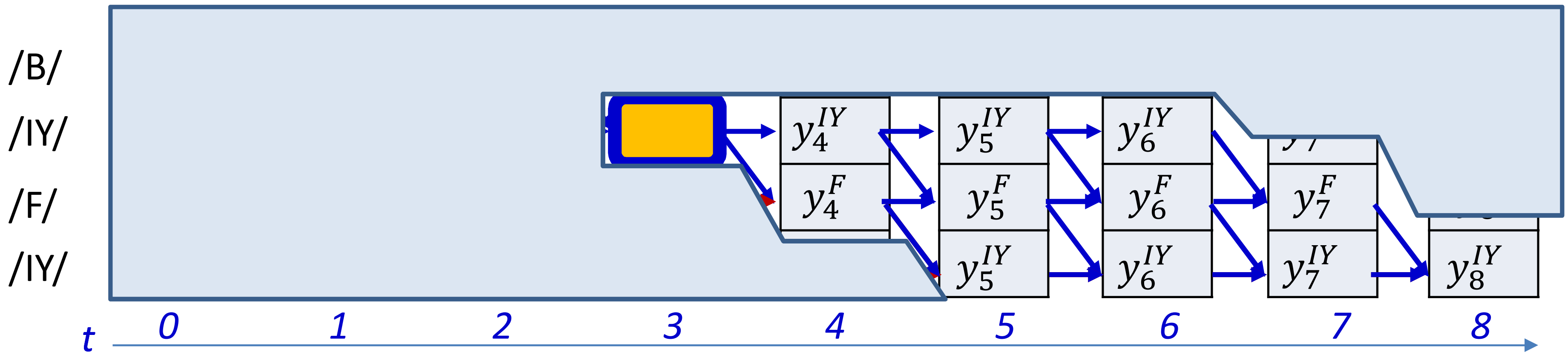


$$P(s_t = S_r, \mathbf{S}|\mathbf{X}) = \alpha(t,r)P(s_{t+1} \in succ(S_r), succ(S_r), \ldots, S_K\ |\mathbf{X})$$

- 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

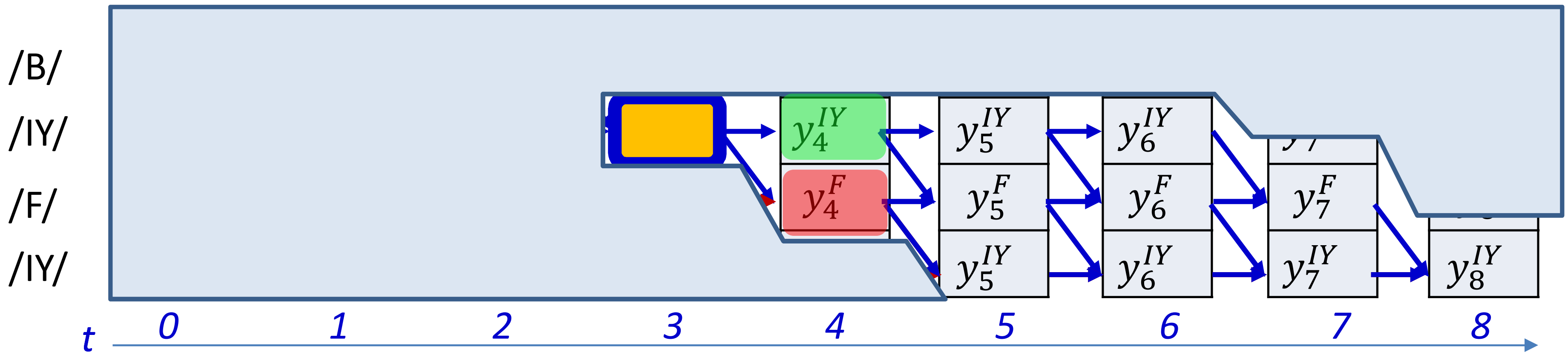
A posteriori symbol probability



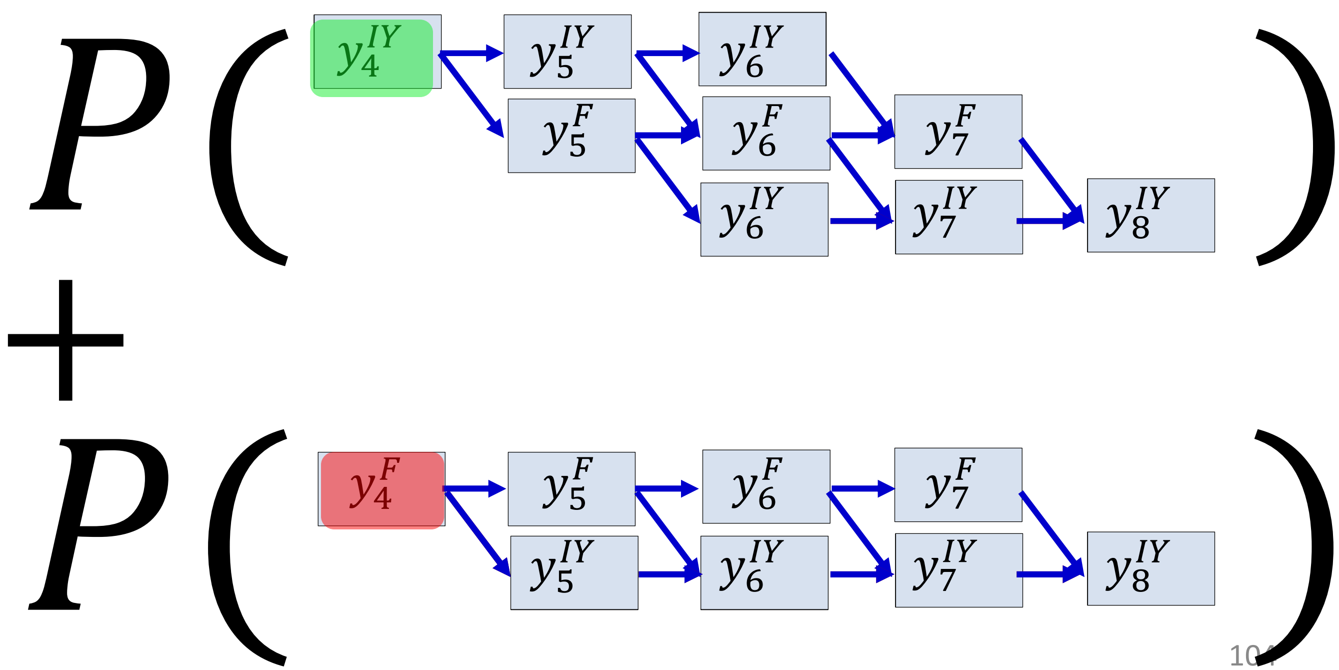
$$\beta(t,r) = P(s_{t+1} \in succ(S_r), succ(S_r), \dots, S_K \,|\mathbf{X})$$

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

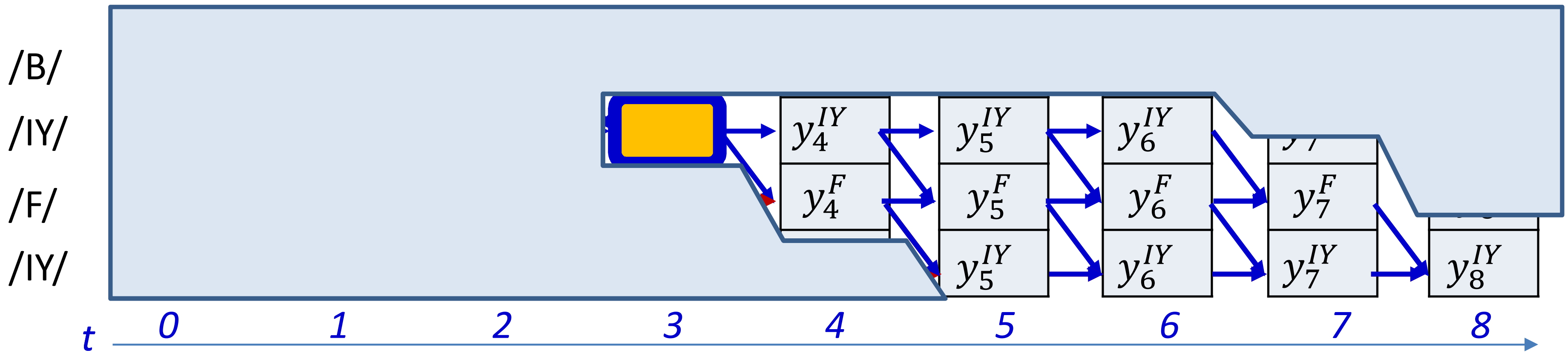
A posteriori symbol probability

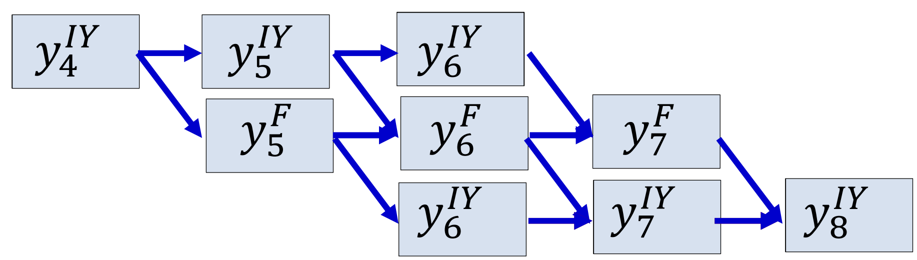


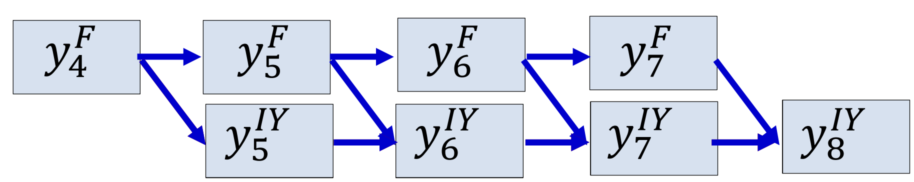
$\beta(3,1) =$



A posteriori symbol probability

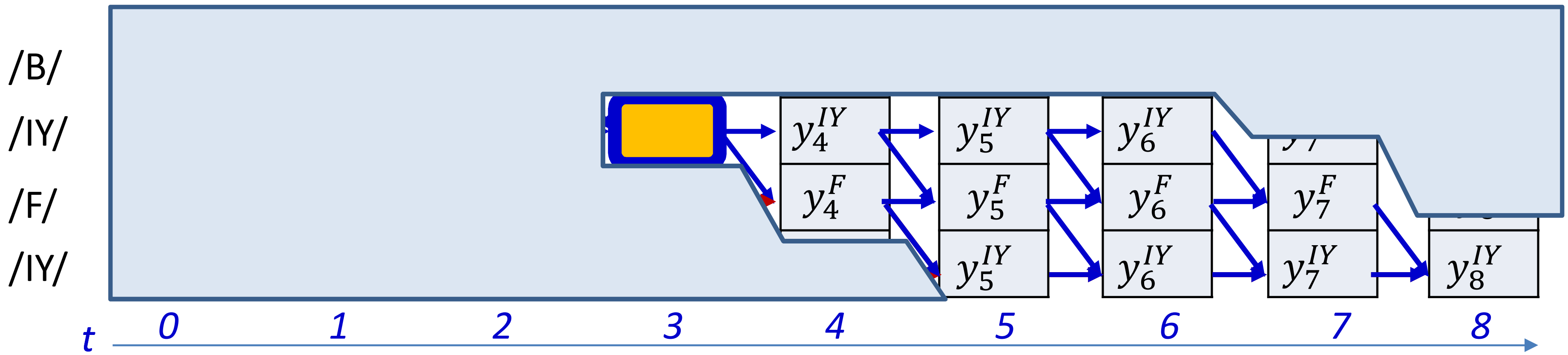


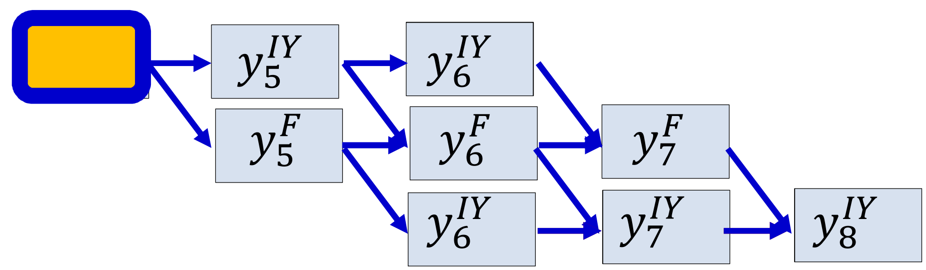
$\beta(3,1) =$ $P($ 

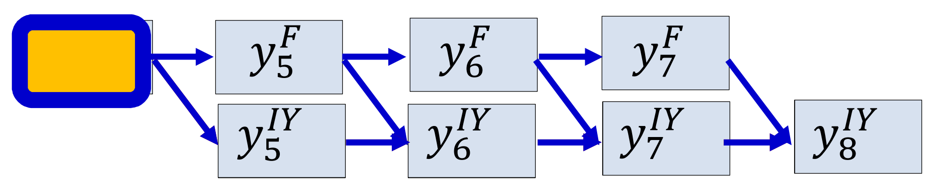
$)$ $+$ $P($ 

$)$

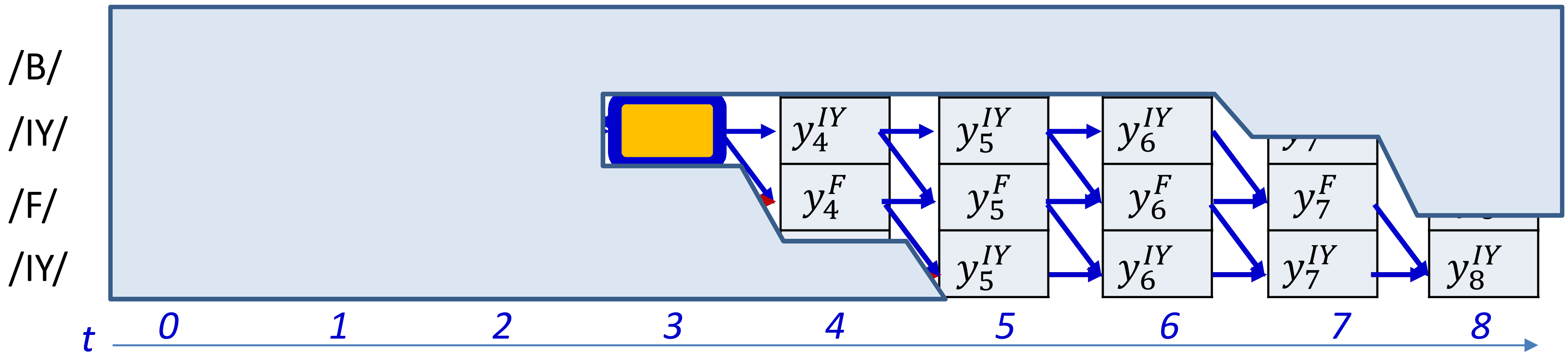
A posteriori symbol probability



$\beta(3,1) =$ $y_4^{IY} P\big($  $\big)$

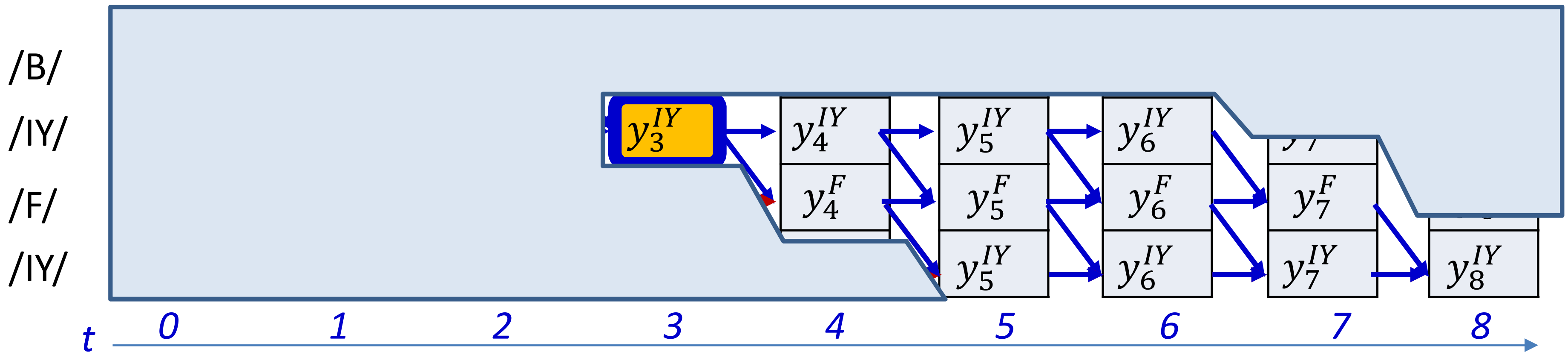
$+$ $y_4^F P\big($  $\big)$

A posteriori symbol probability



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

A posteriori symbol probability

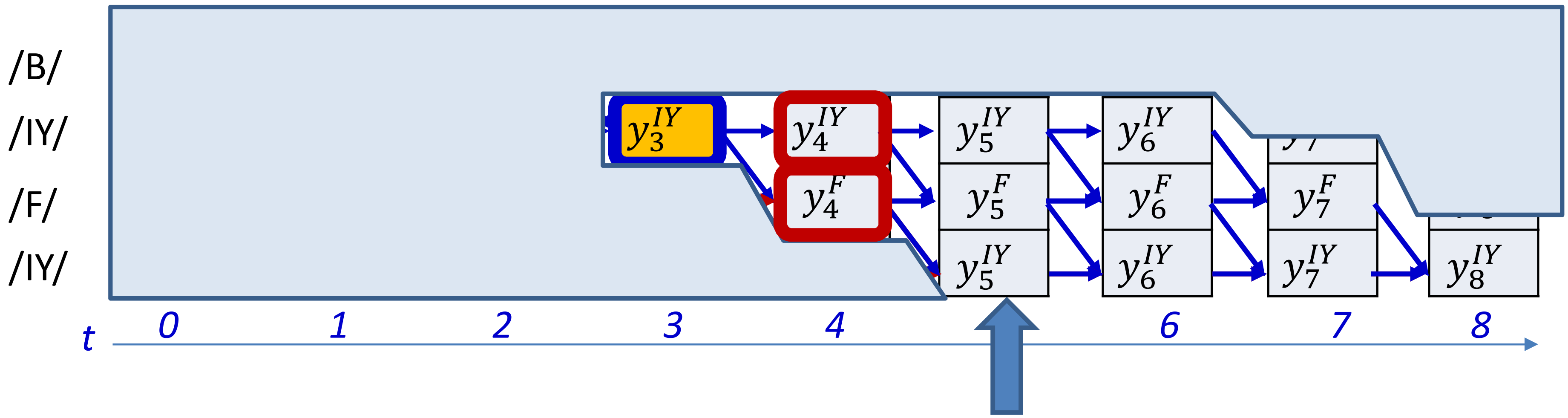


$$\beta(t,r) = P(s_{t+1} \in succ(S_r), succ(S_r), \dots, S_K \,|\mathbf{X})$$

- $\beta(t,r)$ is the probability of the exposed subgraph, not including the orange shaded box
- Note that RHS includes *all* potential successors of S_r. Lets expand this out by explicitly summing over all potential successors

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

A posteriori symbol probability

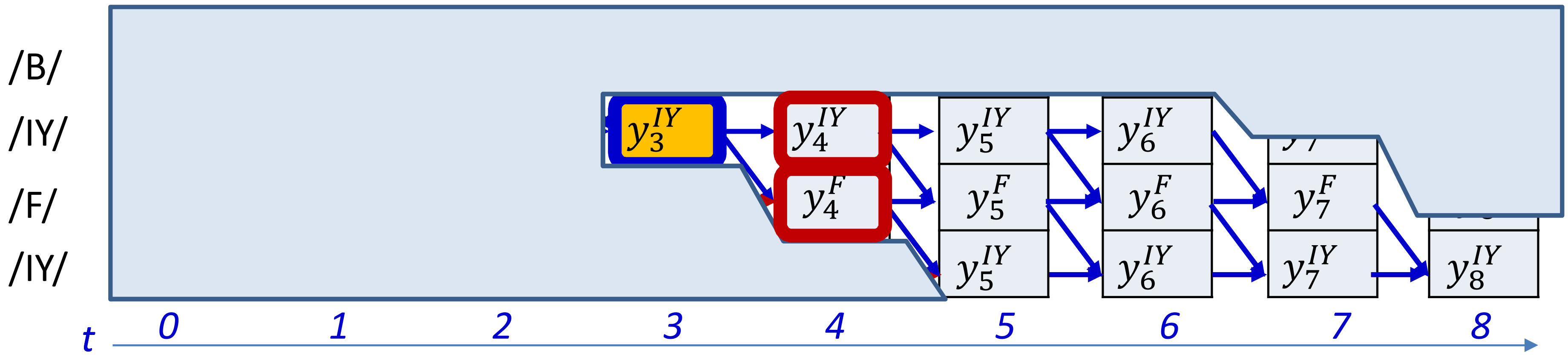


$$\beta(t,r) = \sum_{q:S_q \in succ(S_r)} P\left(s_{t+1} = S_q, S_q, \dots, S_K | \mathbf{X}\right)$$

- 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)

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

A posteriori symbol probability



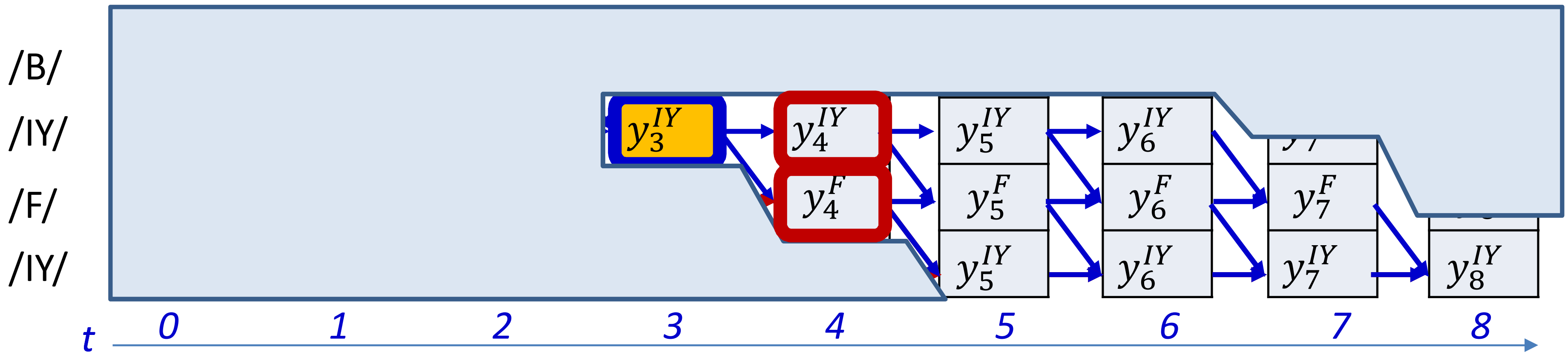
- Expressing $\beta(t,r)$ in terms of the successors of S_q

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

- Using our assumption of conditional independence

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

A posteriori symbol probability



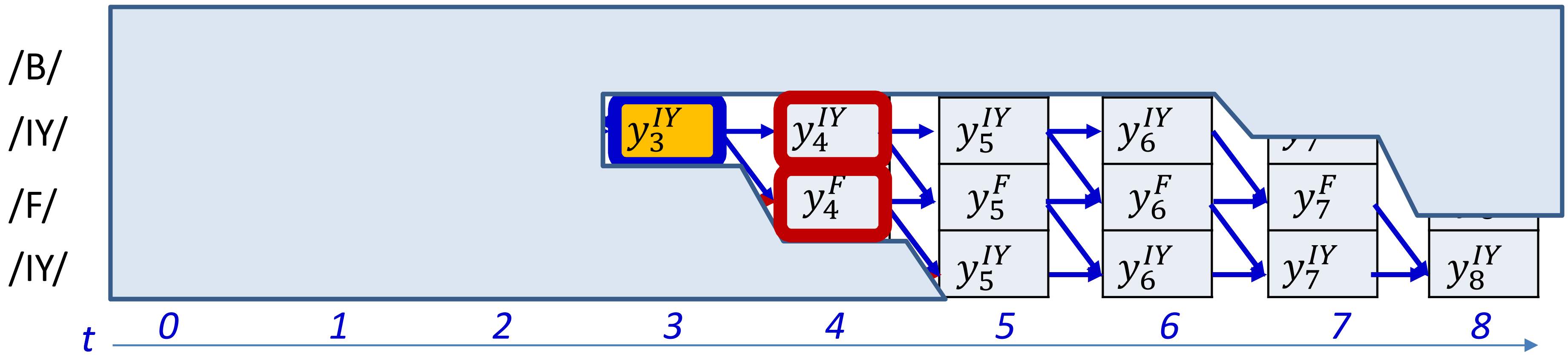
- Expressing $\beta(t,r)$ in terms of the successors of S_q

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

- Using our assumption of conditional independence

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

A posteriori symbol probability



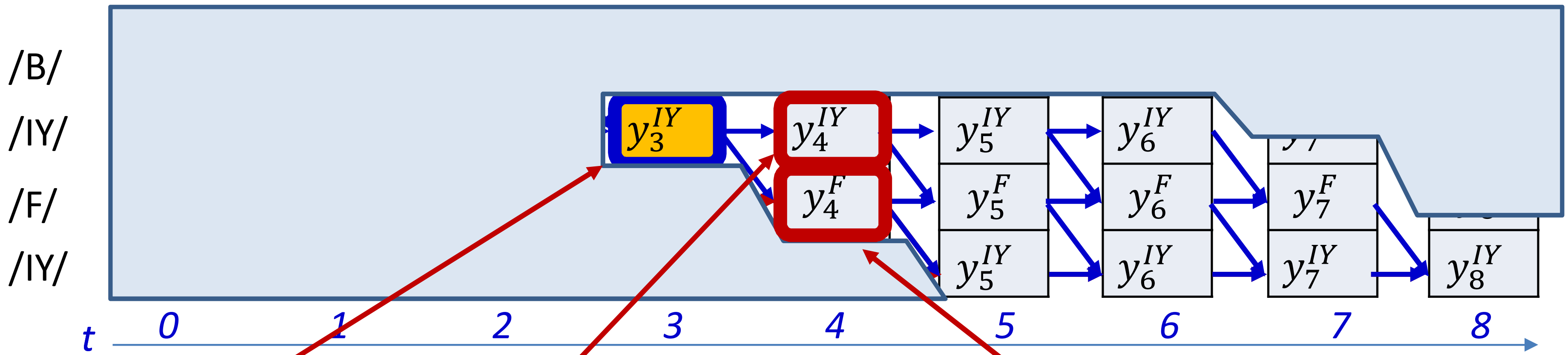
- Expressing $\beta(t,r)$ in terms of the successors of S_q

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

- Using our assumption of conditional independence

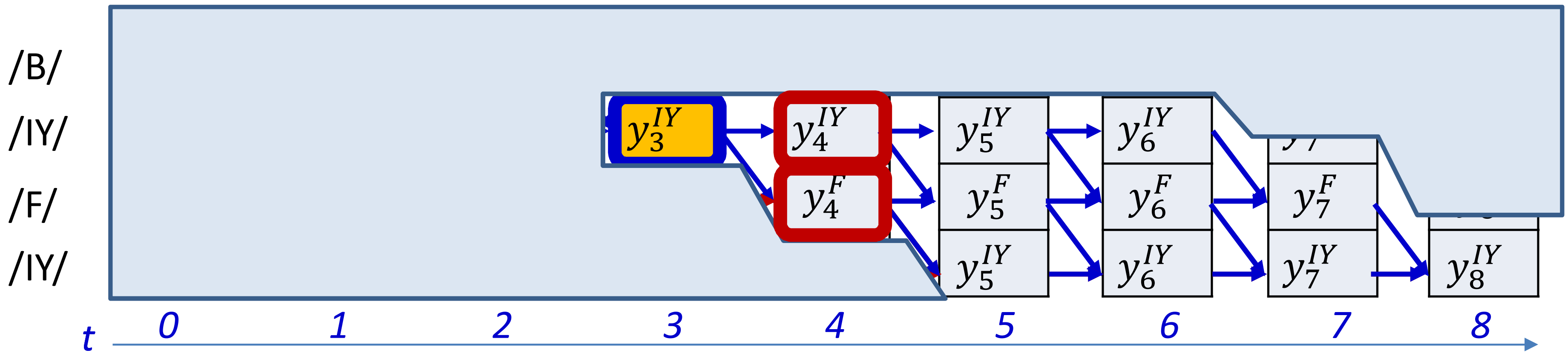
$$\beta(t,r) = \sum_{q:S_q \in succ(S_r)} y_{t+1}^{S_q} \beta(t+1,q)$$

Backward algorithm



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

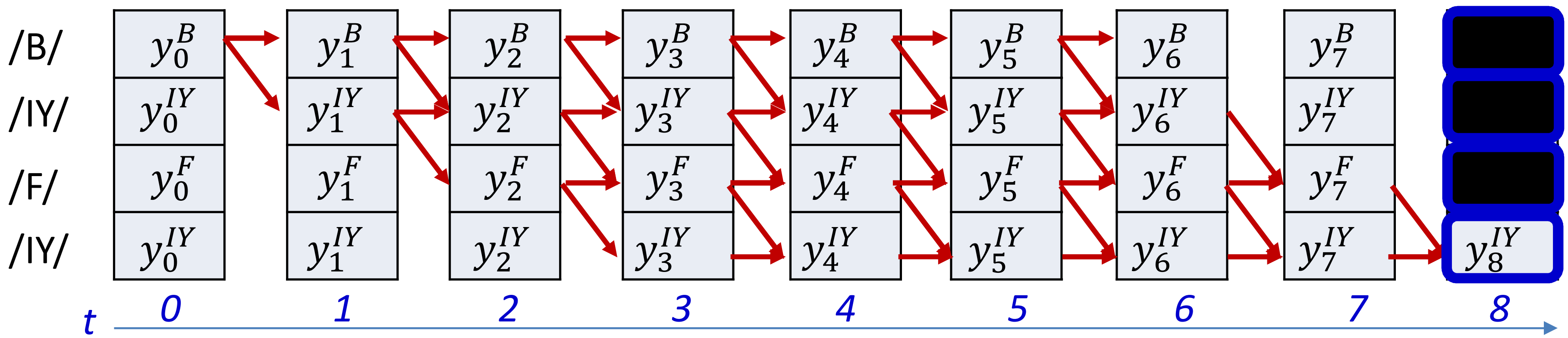
Backward algorithm



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

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

Backward algorithm



- Initialization:

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

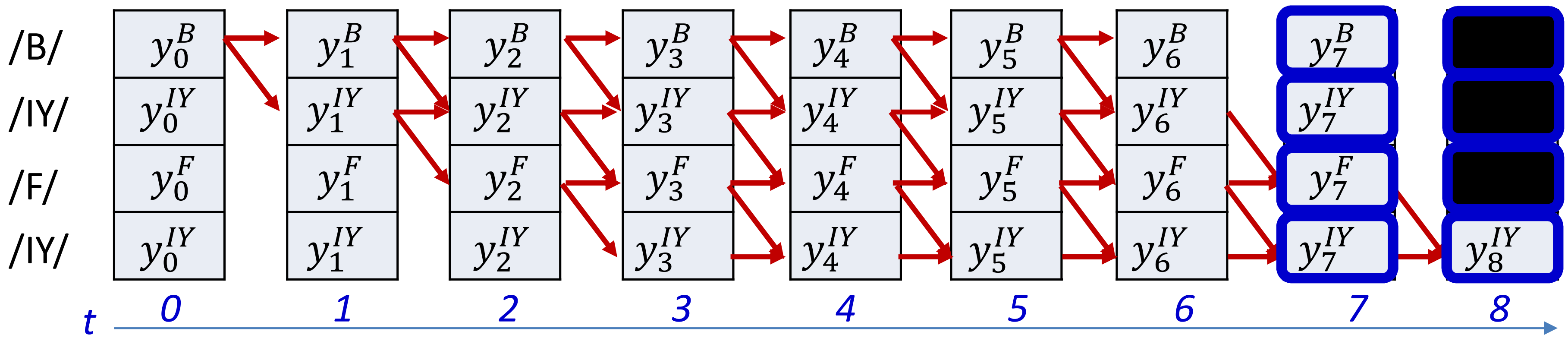
- for $t \; = T - 2 \; downto \; 0$

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

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:

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

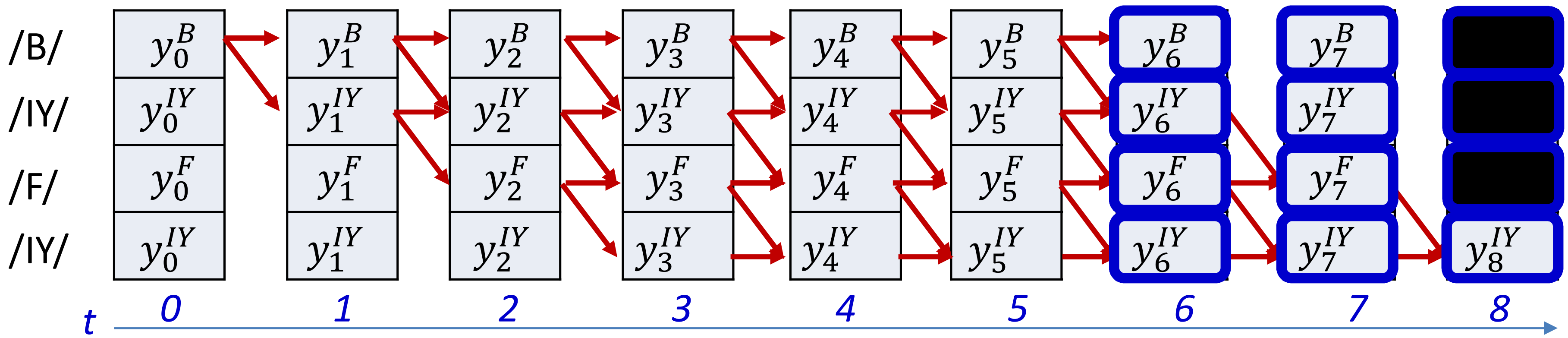
- for $t\ = T - 2\ downto\ 0$

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

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:

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

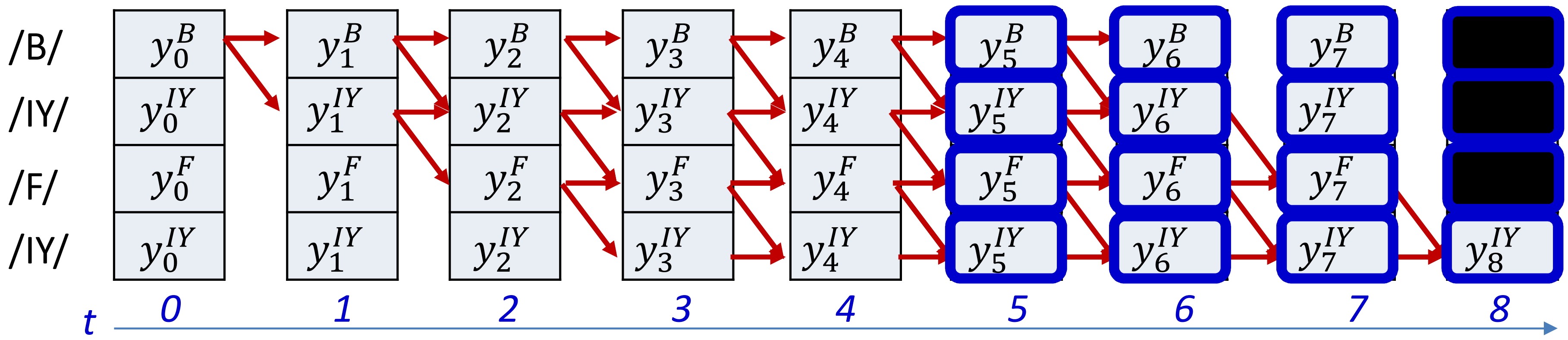
- for $t\ = T - 2\ downto\ 0$

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

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:

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

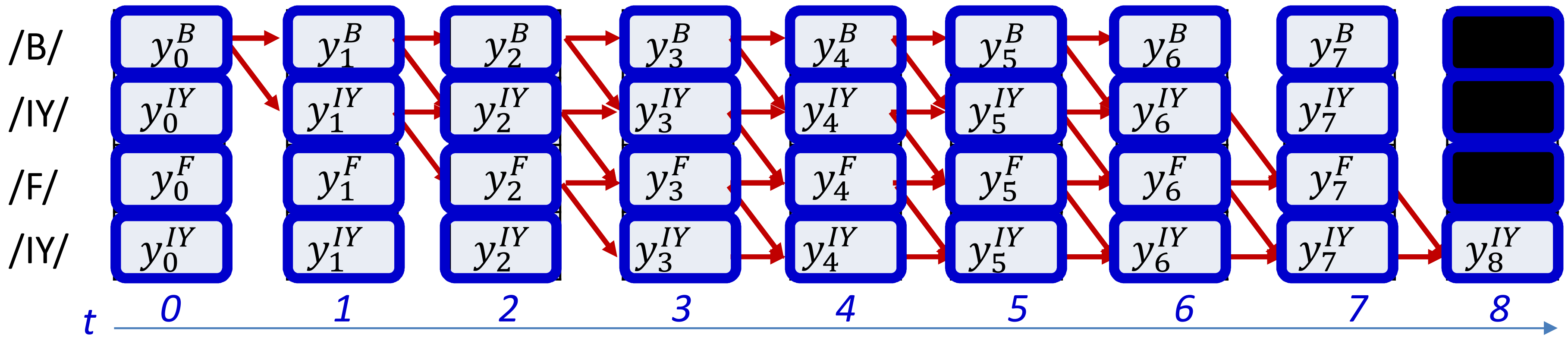
- for $t\ = T - 2\ downto\ 0$

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

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:

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

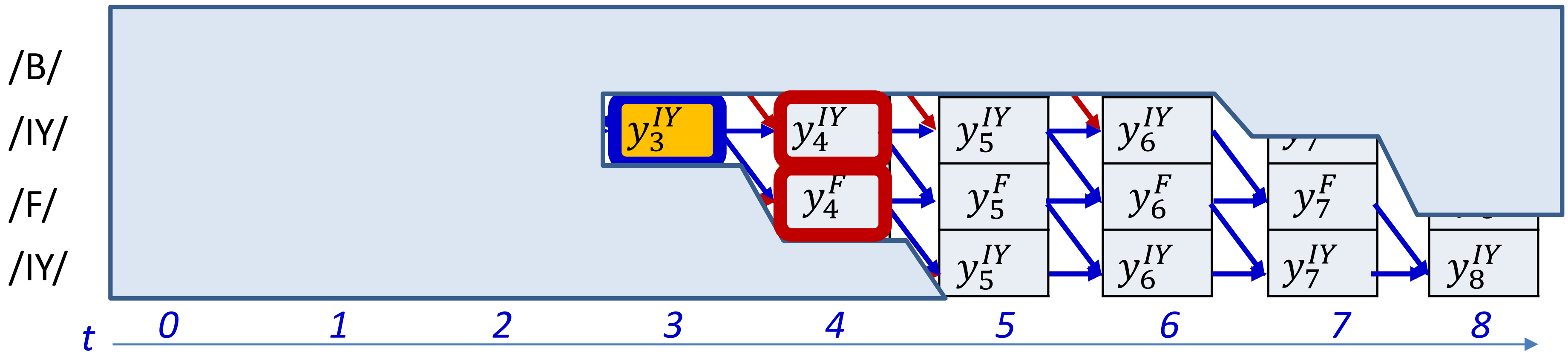
- for $t \; = T - 2 \; downto \; 0$

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

for $l \; = K - 1 \ldots 1$

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

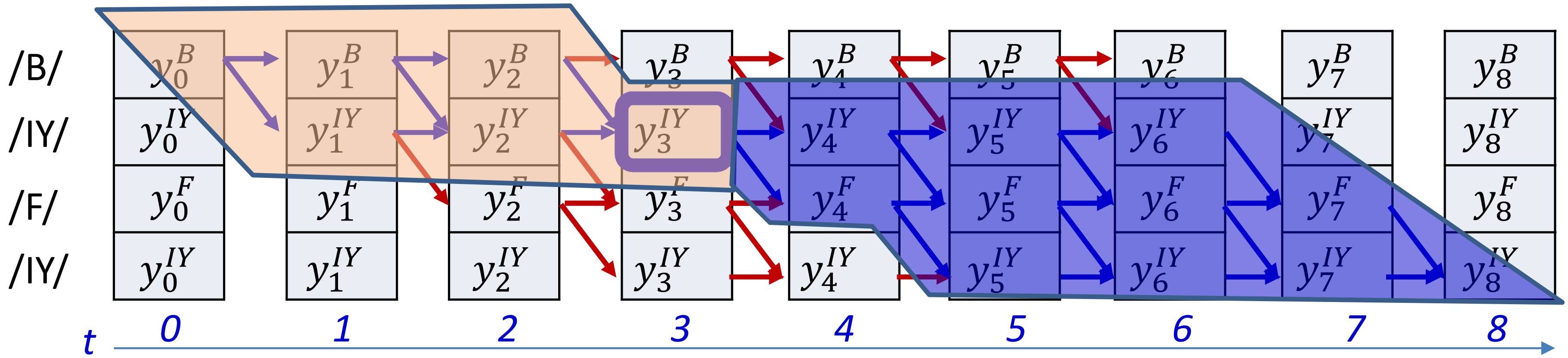
Alternate Backward algorithm



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

- Some implementations of the backward algorithm will use the above formula
- Note that here the probability of the observation at t is also factored into beta
- It will have to be unfactored later (we'll see how)

The joint probability

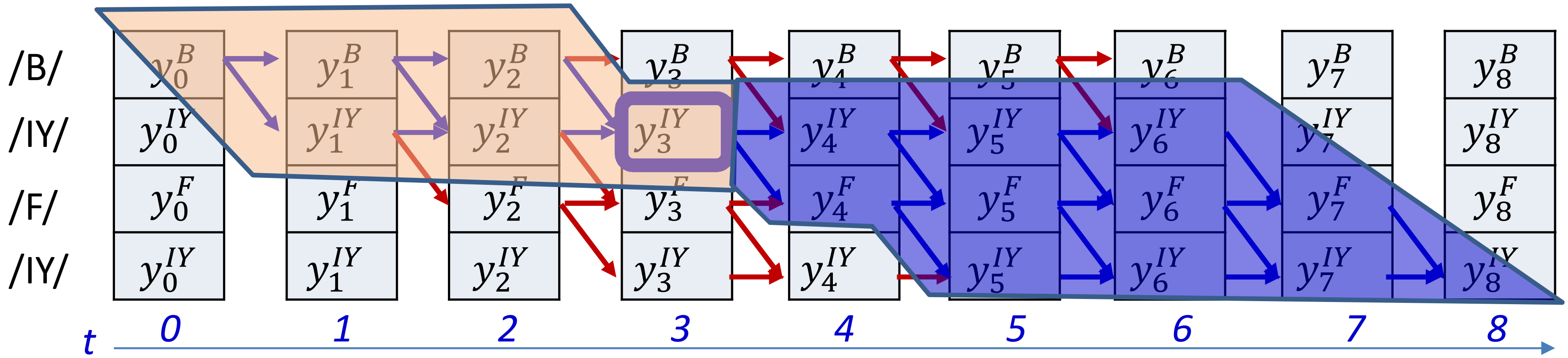


$$P(s_t = S_r, \mathbf{S}|\mathbf{X}) = \alpha(t,r)P(s_{t+1} \in succ(S_r), succ(S_r), \dots, S_K\, |\mathbf{X})$$

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

We now can compute this

The joint probability



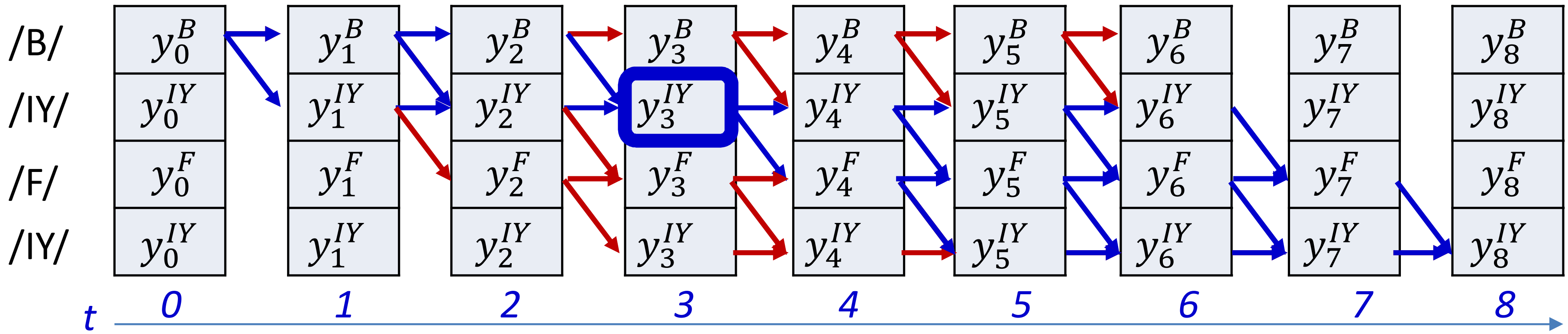
$$P(s_t = S_r, \mathbf{S}|\mathbf{X}) = \boxed{\alpha(t,r)}\boxed{\beta(t,r)}$$

- 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

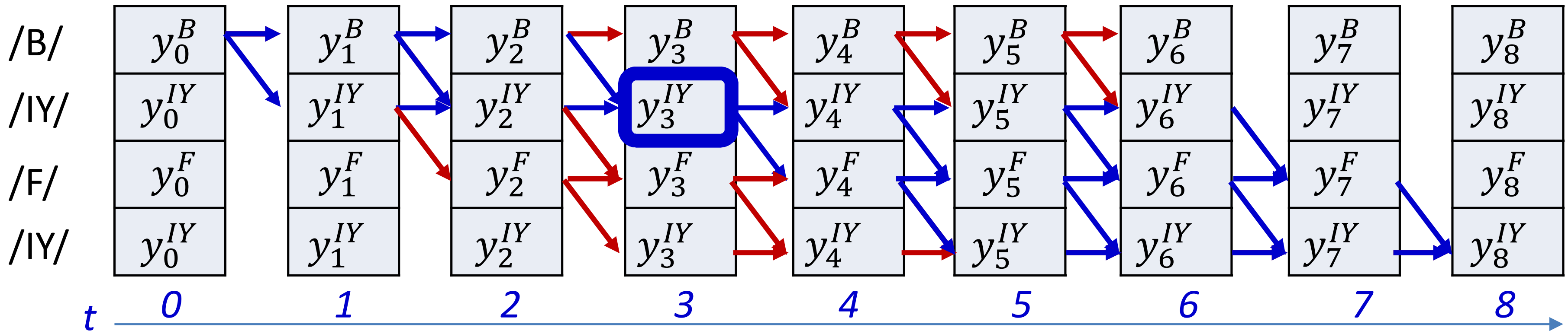


$$P(s_t = S_r, \mathbf{S}|\mathbf{X}) = \alpha(t,r)\beta(t,r)$$

- The *posterior* is given by

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

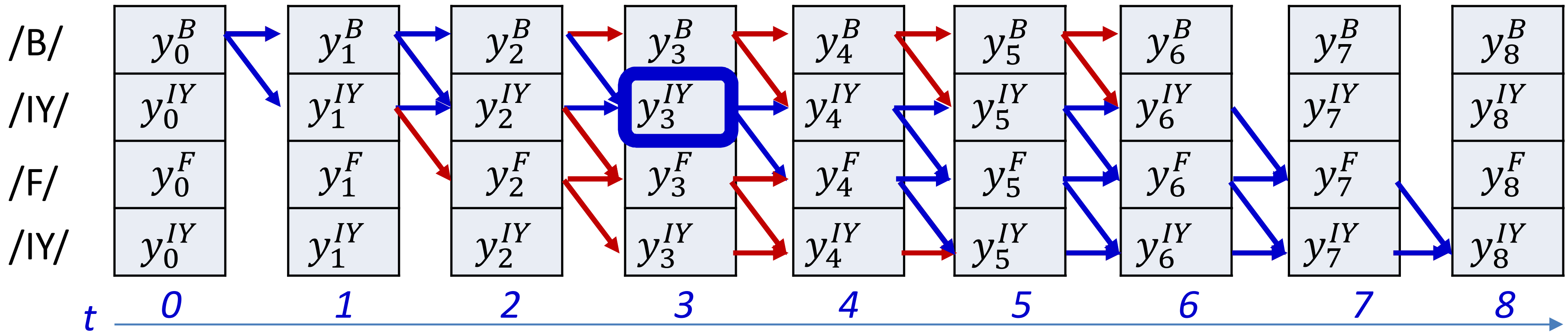
The posterior probability



- 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 posterior probability



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

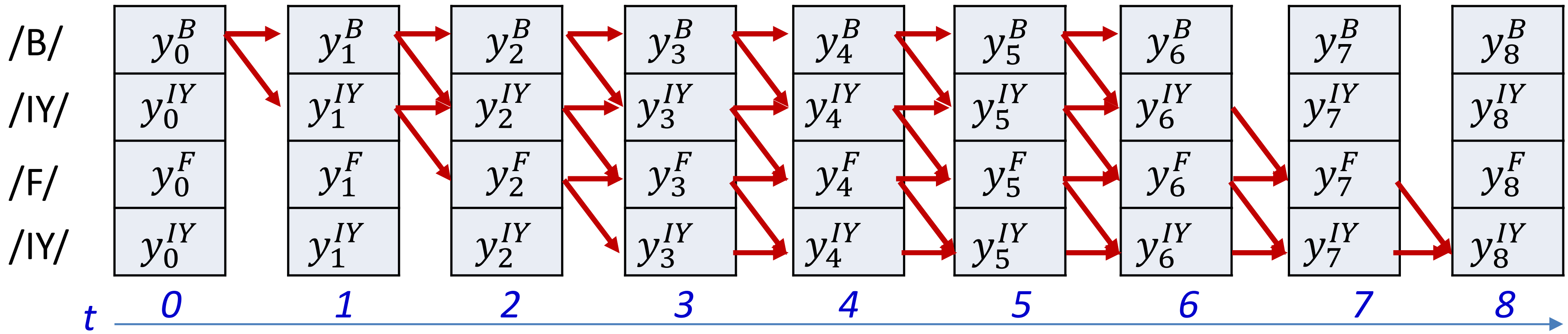
- The *posterior* is given by

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

- We can also write this using the modified beta formula as (you will see this in papers)

$$\gamma(t,r) = \frac{\frac{1}{y_t^{S(r)}}\alpha(t,r)\hat{\beta}(t,r)}{\sum_{r'} \frac{1}{y_t^{S(r)}}\alpha(t,r)\hat{\beta}(t,r)}$$

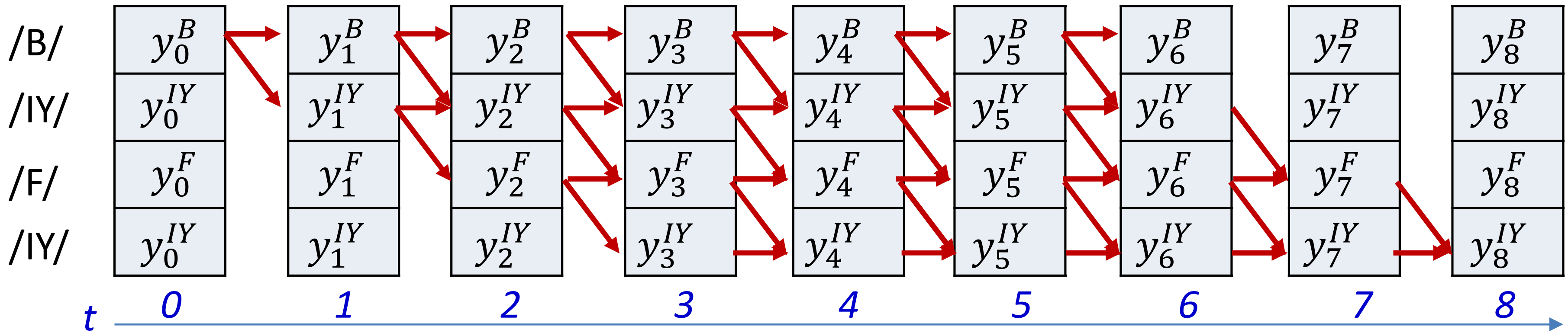
The expected divergence



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

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

The expected divergence



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

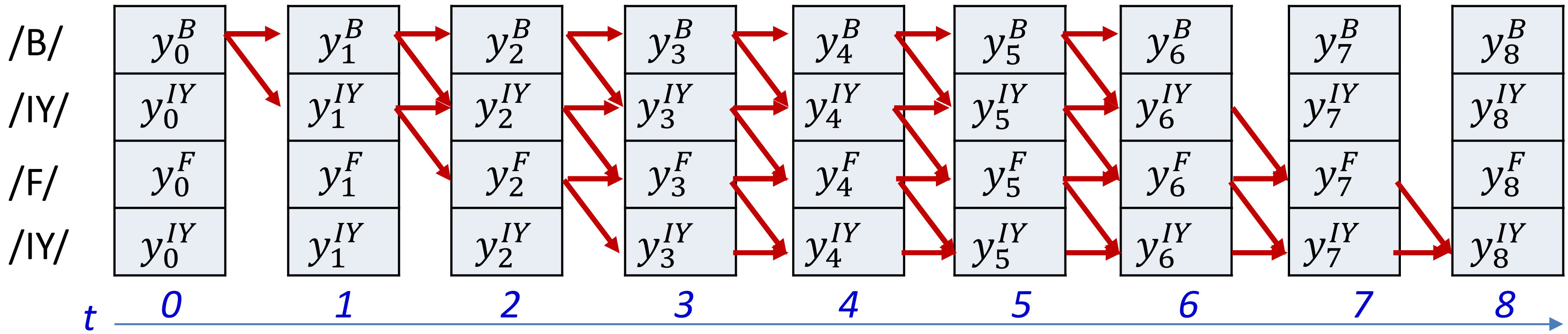
$$DIV = -\sum_t \sum_r \gamma(t, r) \log y_t^{S(r)}$$

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

$$\nabla_{Y_t} DIV = \left[\frac{dDIV}{dy_t^{S_1}} \quad \frac{dDIV}{dy_t^{S_2}} \quad \cdots \quad \frac{dDIV}{dy_t^{S_L}} \right]$$

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

The expected divergence



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

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

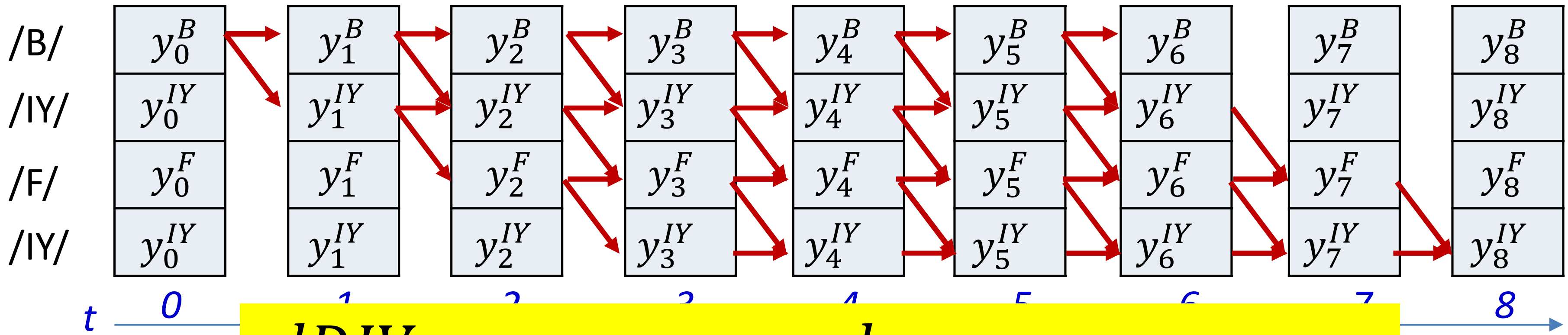
- The derivative of the divergence w.r.t the output Y_t of the net at any time:

$$\nabla_{Y_t} DIV = \left[\frac{dDIV}{dy_t^{S_1}} \quad \frac{dDIV}{dy_t^{S_2}} \quad \cdots \right]$$

Must compute these terms from here

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

The expected divergence



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

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

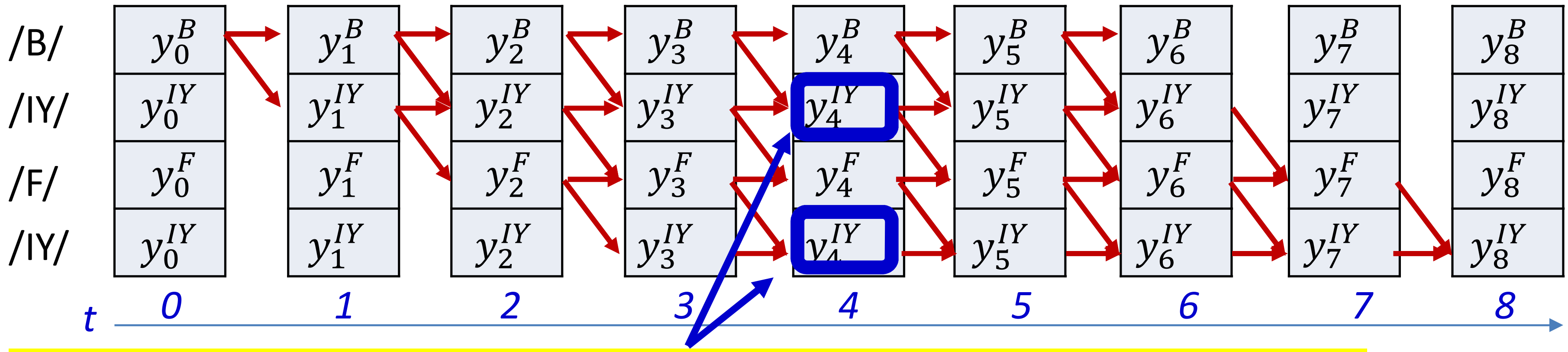
- The derivative of the divergence w.r.t the output Y_t of the net at any time:

$$\nabla_{Y_t} DIV = \left[\frac{dDIV}{dy_t^{S_1}} \quad \frac{dDIV}{dy_t^{S_2}} \quad \cdots \right]$$

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 $\frac{dDIV}{dy_4^{IY}}$

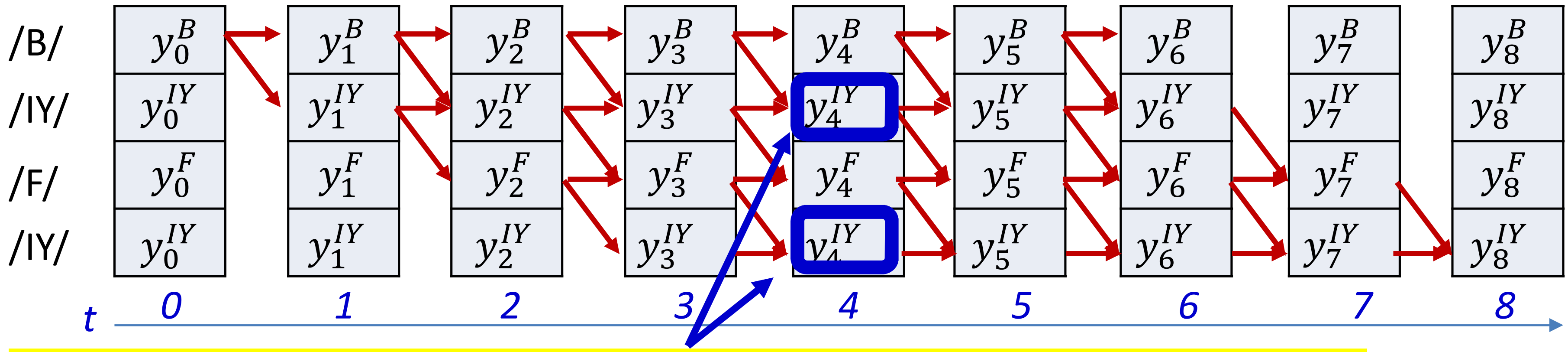
$$\frac{dDIV}{dy_t^l} = -\sum_{r\,:S(r)=l} \frac{d}{dy_t^{S(r)}} \gamma(t,r) \log y_t^{S(r)}$$

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

$$\nabla_{Y_t} DIV = \left[\frac{dDIV}{dy_t^{S_1}} \quad \frac{dDIV}{dy_t^{S_2}} \quad \cdots \quad \frac{dDIV}{dy_t^{S_L}} \right]$$

  – 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 $\frac{dDIV}{dy_4^{IY}}$

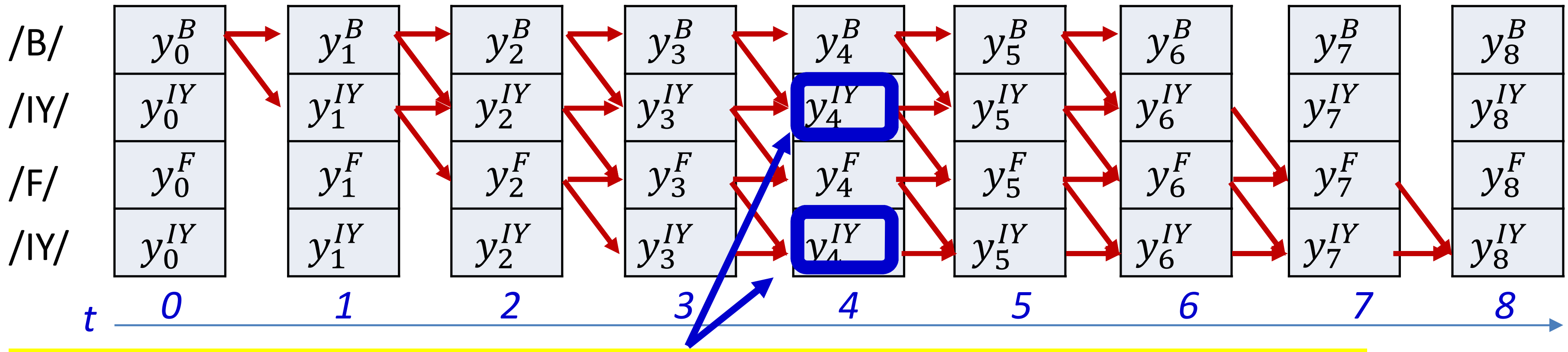
$$\frac{dDIV}{dy_t^l} = - \sum_{r\,:S(r)=l} \frac{d}{dy_t^{S(r)}} \gamma(t,r) \log y_t^{S(r)}$$

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

$$\nabla_{Y_t} DIV = \left[\frac{dDIV}{dy_t^{S_1}} \quad \frac{dDIV}{dy_t^{S_2}} \quad \cdots \quad \frac{dDIV}{dy_t^{S_L}} \right]$$

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

The expected divergence



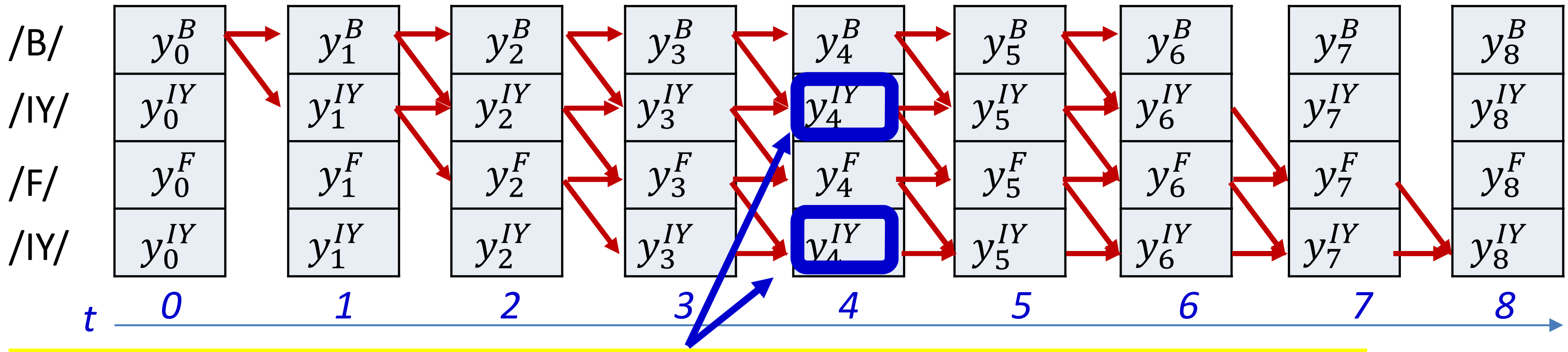
The derivatives at both these locations must be summed to get $\frac{dDIV}{dy_4^{IY}}$

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

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

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

The expected divergence



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

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

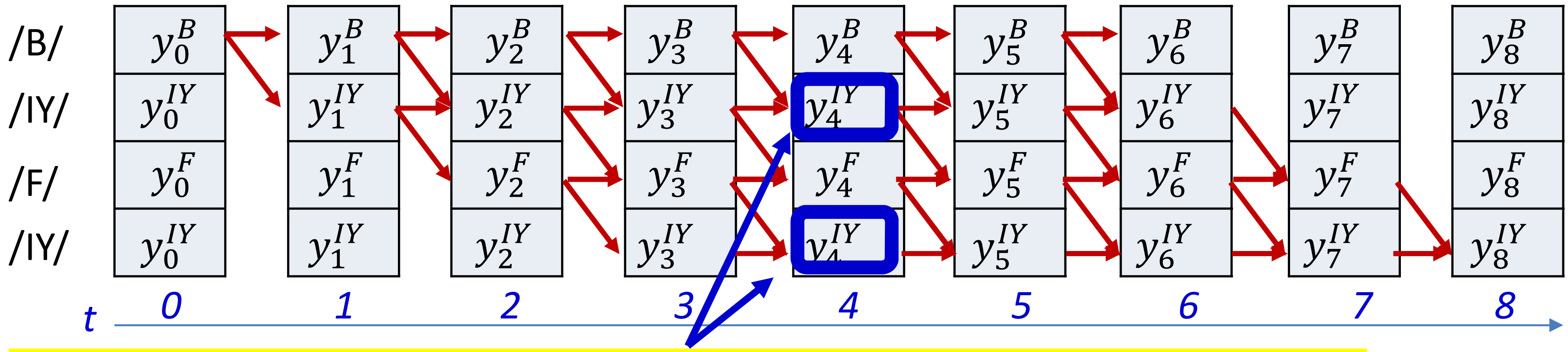
- The derivativ

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

at any time:

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

The expected divergence



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

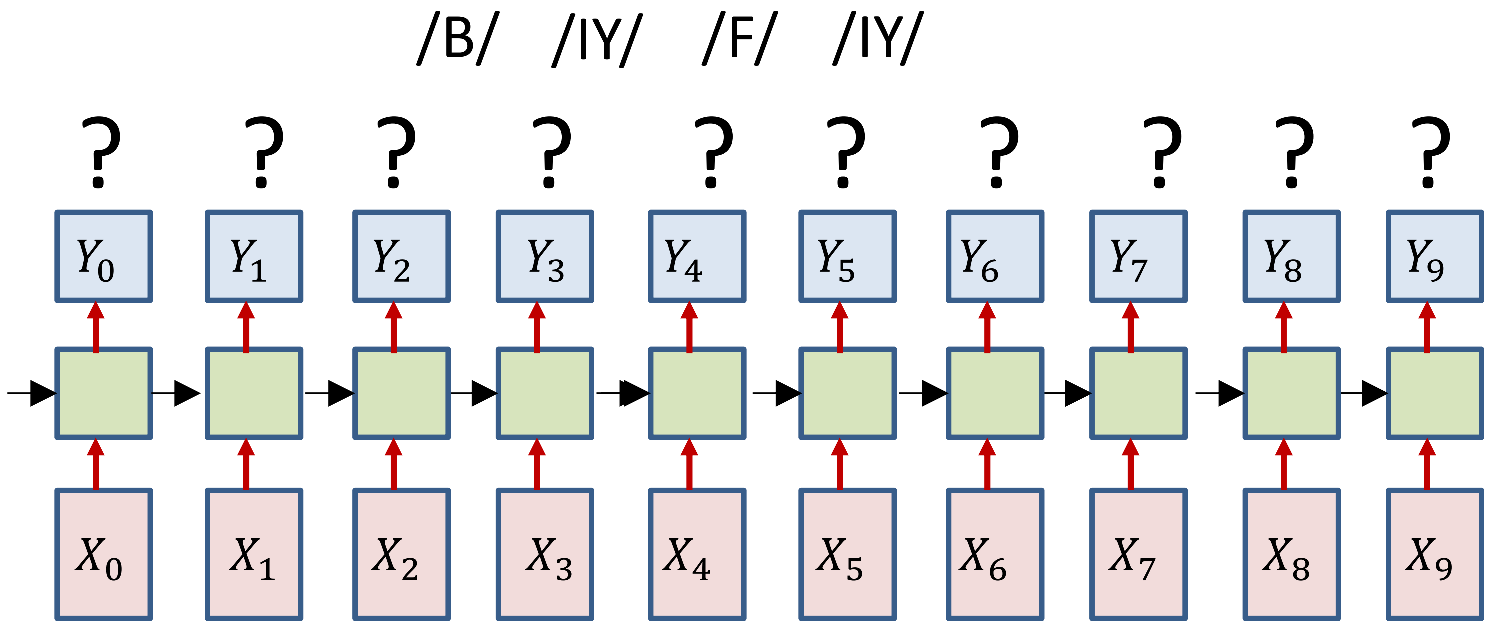
$$DIV = -\sum_t \sum_r \gamma(t,r) \log y_t^{S(r)}$$

- 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

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

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

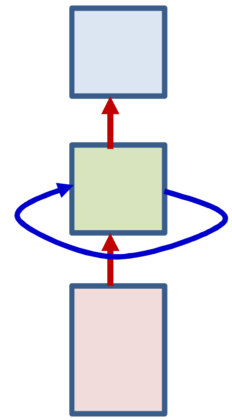
Overall training procedure for Seq2Seq case 1



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

Overall training procedure for Seq2Seq case 1

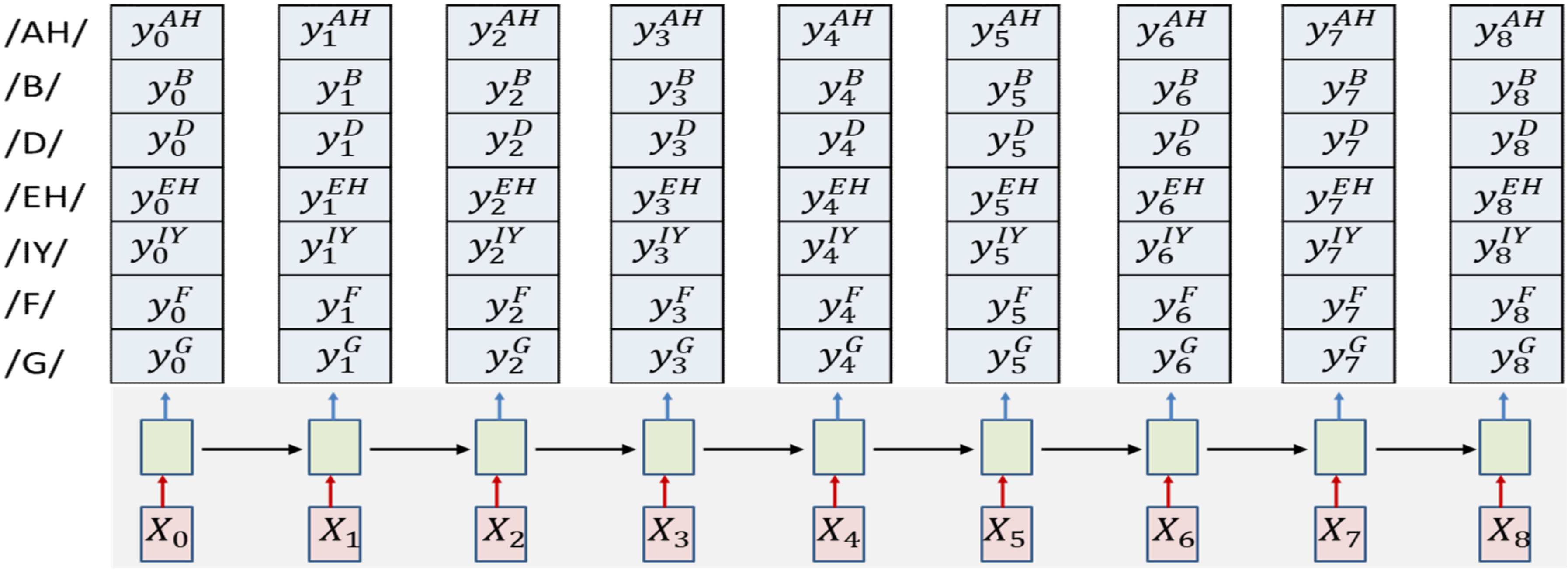
- **Step 1**: Setup the network
  - Typically many-layered LSTM



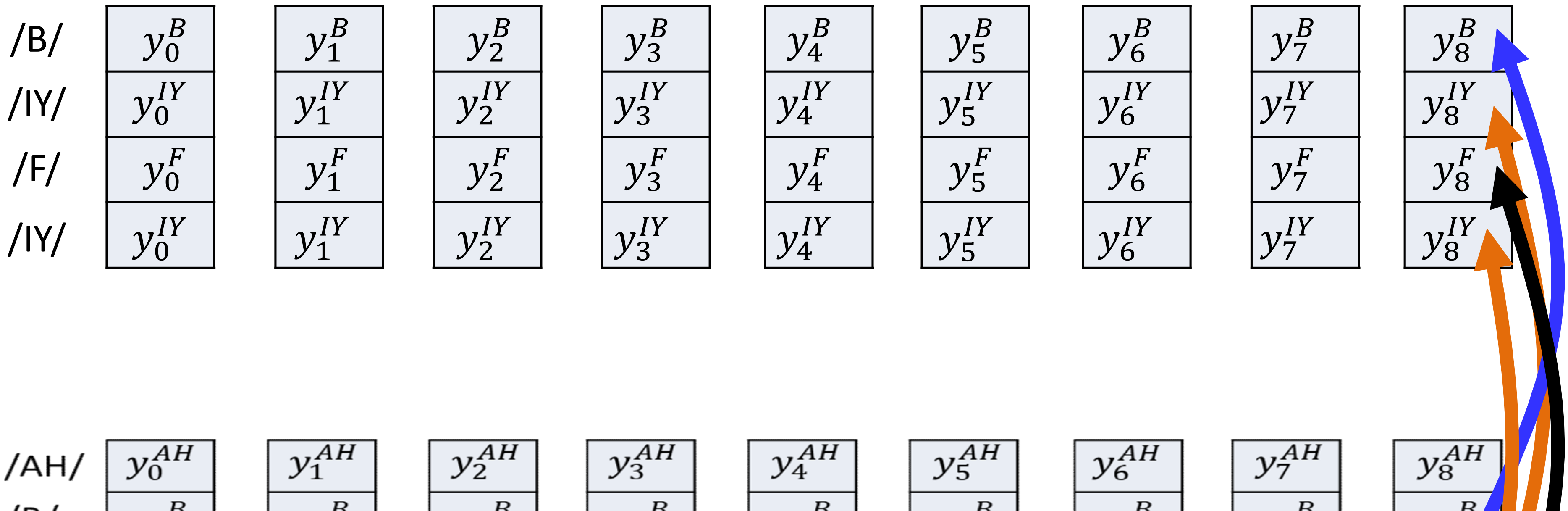
- **Step 2**: Initialize all parameters of the network

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

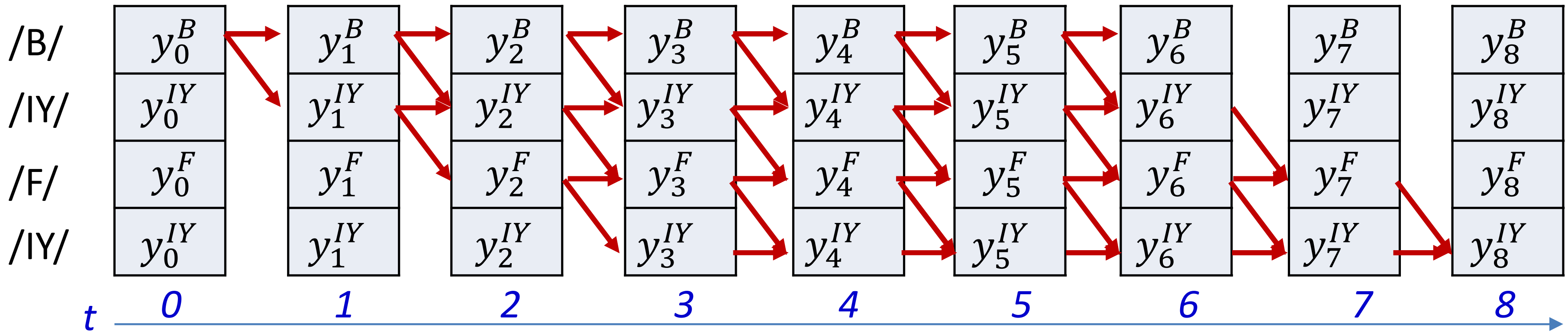


Overall training: Backward pass



- 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
  - **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

$$\nabla_{Y_t} DIV = \left[\frac{dDIV}{dy_t^1} \quad \frac{dDIV}{dy_t^2} \quad \cdots \quad \frac{dDIV}{dy_t^L} \right]$$

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

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

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

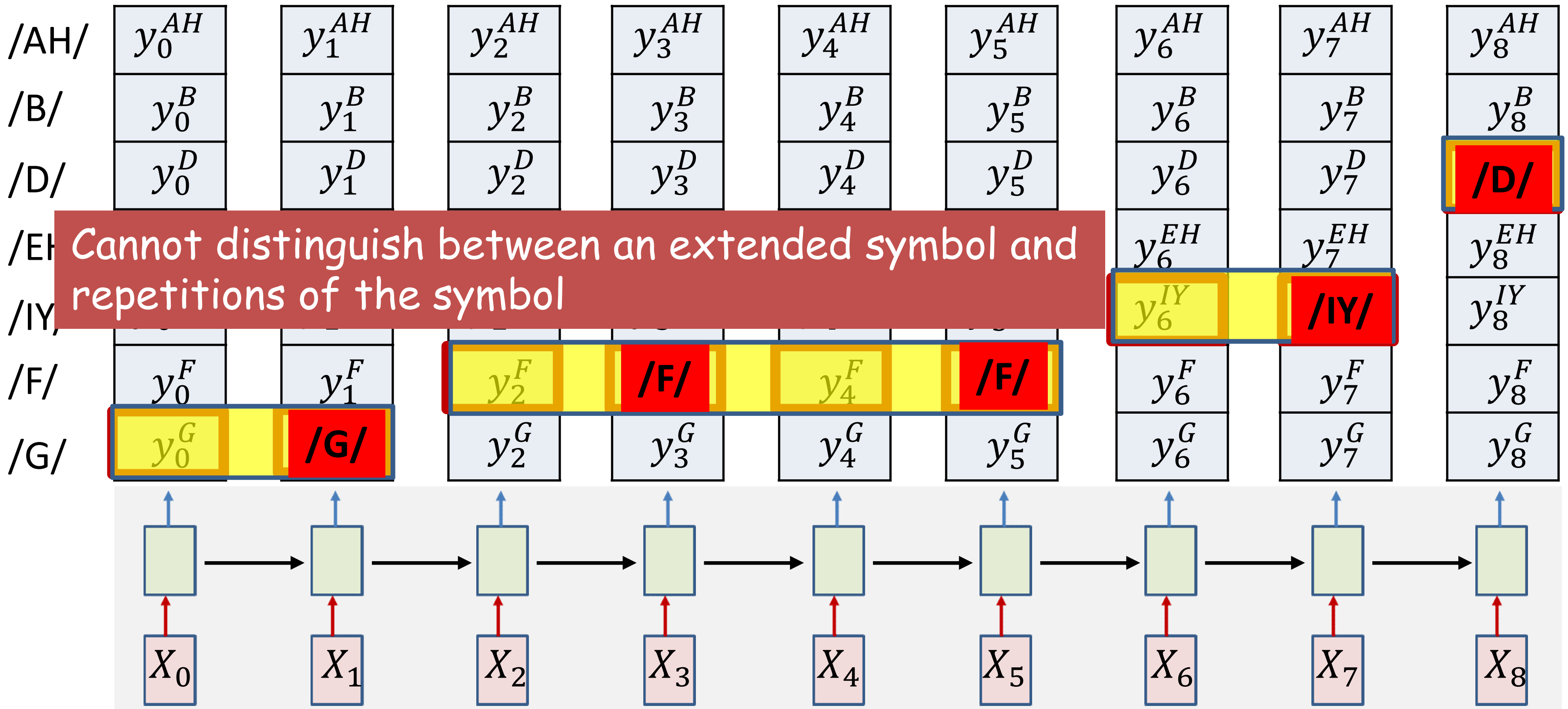
- Training can be performed by iteratively estimating the alignment by Viterbi-decoding and time-synchronous training

- 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 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

- 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 symbol sequence ROD or ROOD?

- Note: This problem does not always occur, e.g. if repetition is not permitted by the structure of the problem
  - E.g. when symbols have sub symbols
    - E.g. If O is produced as O1 and O2 and must always begin with O1 and terminate with O2
      - A single O would be of the form O1 O1 .. O2 → O
      - Multiple Os would have the decode O1 .. O2.. O1..O2.. → OO

A key *decoding* problem

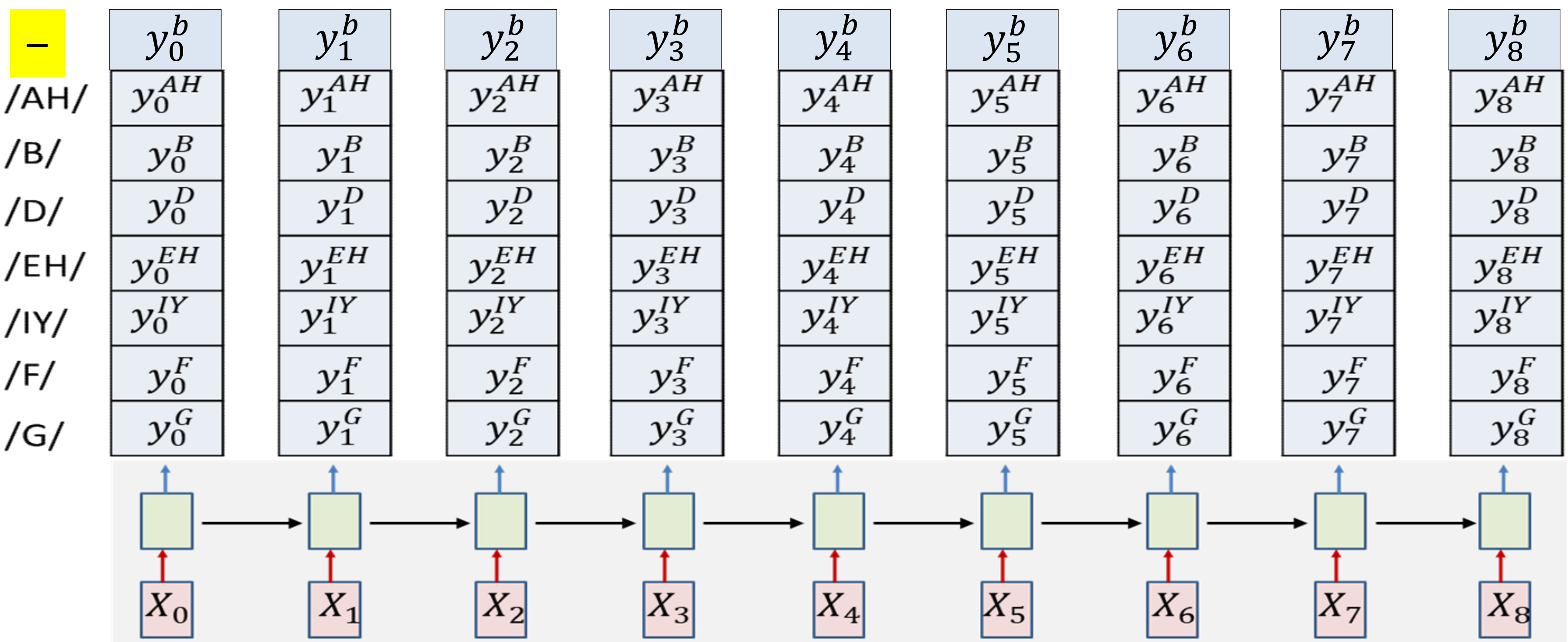
- We have a decode: R R R O O O O O D
- Is this the symbol sequence ROD or ROOD?

- 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

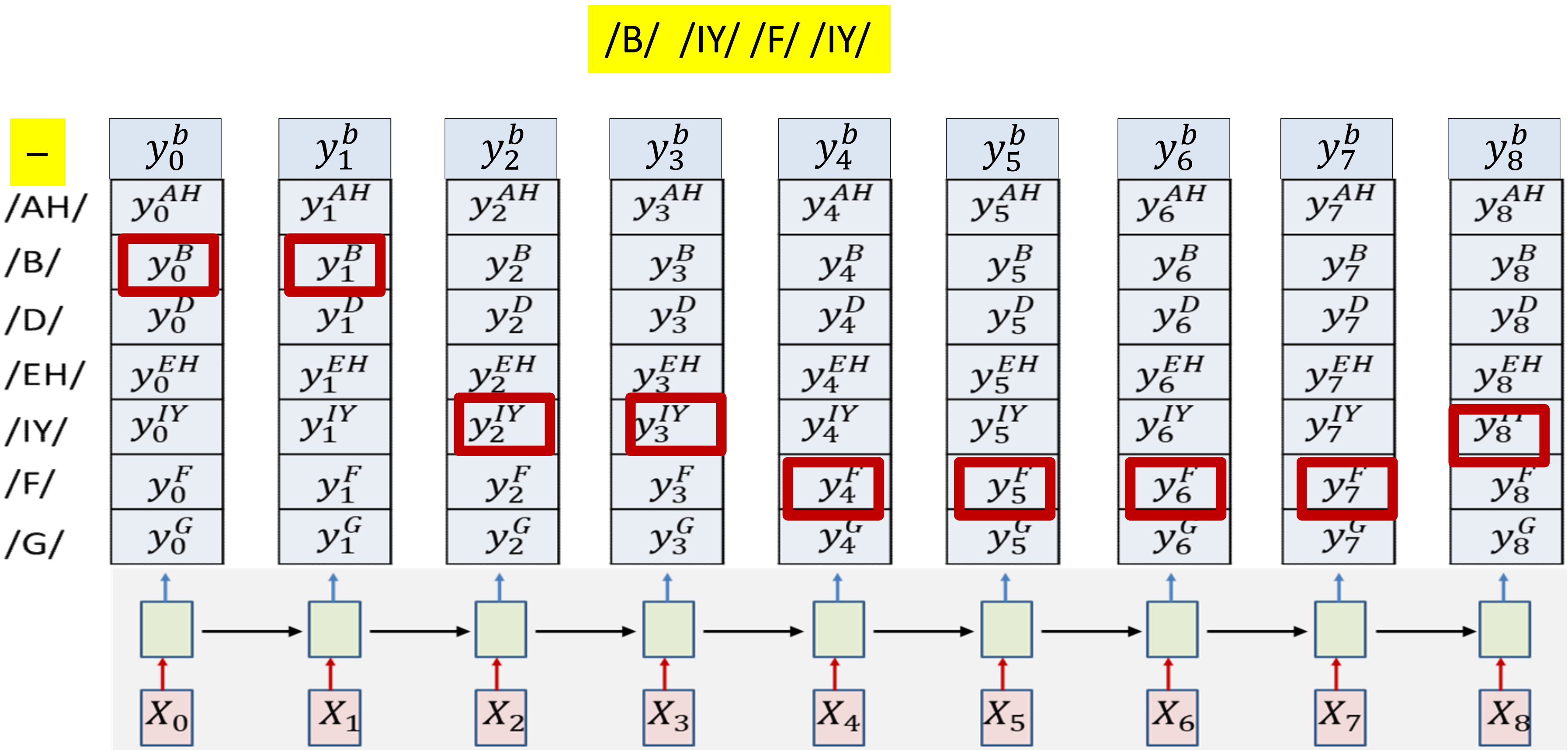
The modified forward output

- Note the extra “blank” at the output



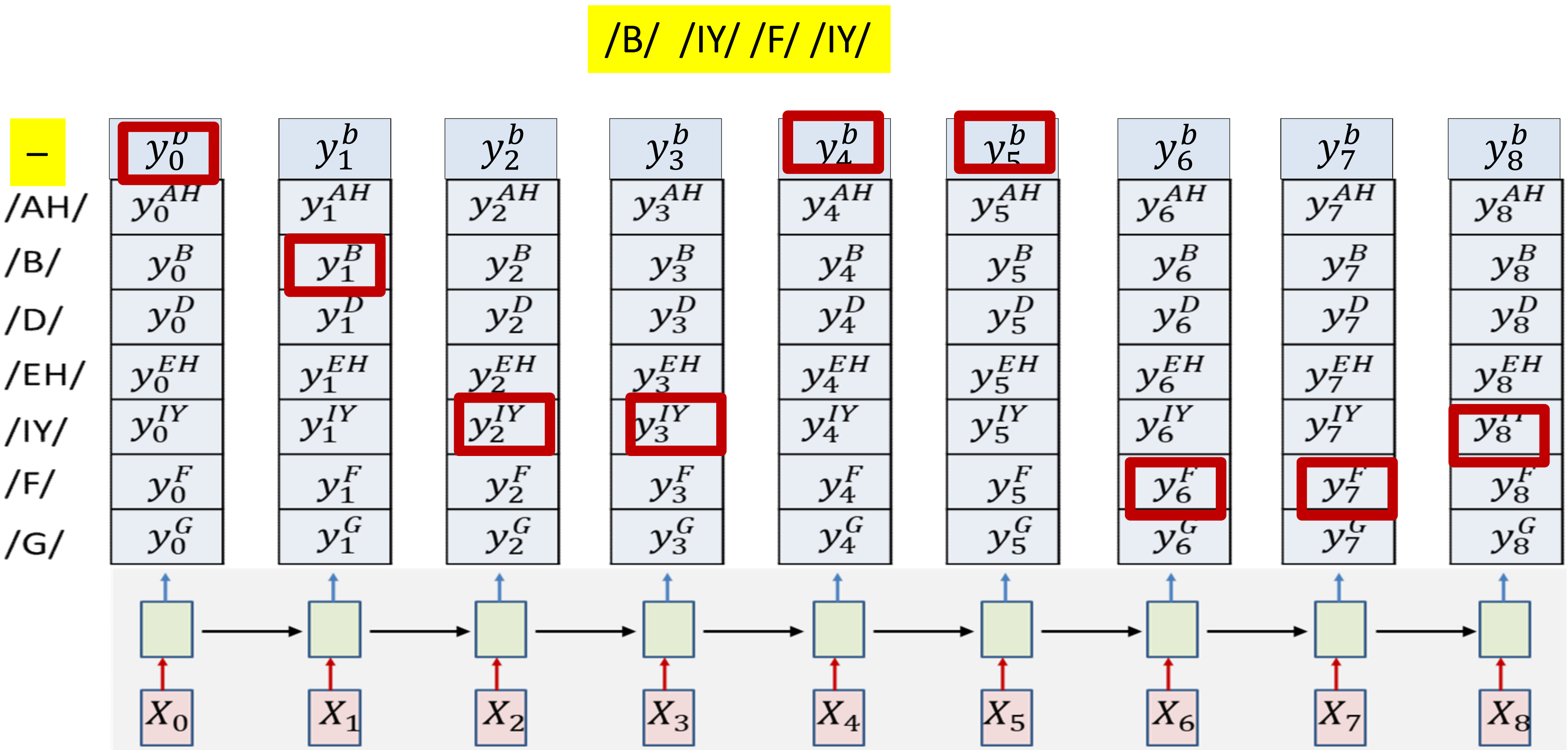
The modified forward output

- Note the extra "blank" at the output



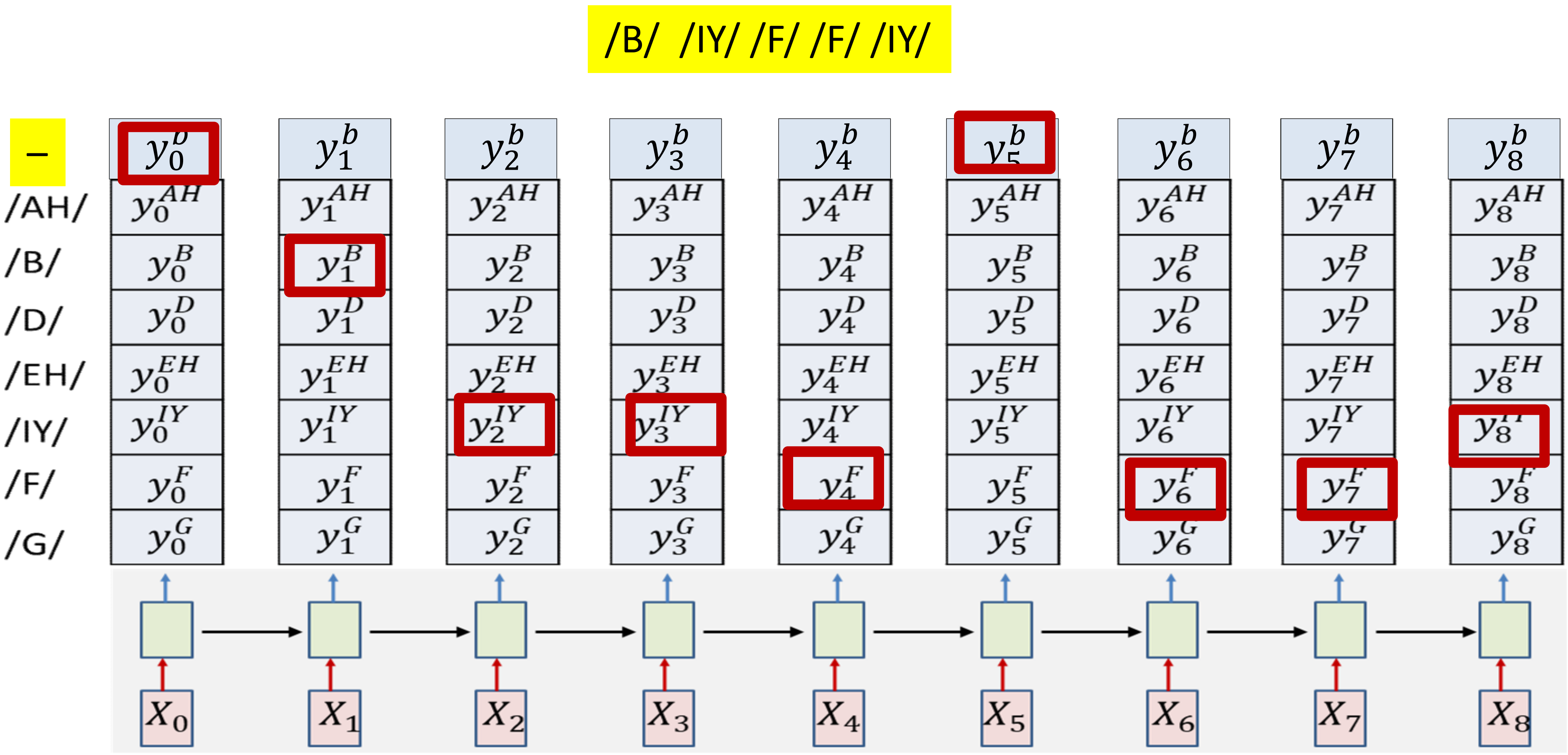
The modified forward output

- Note the extra "blank" at the output

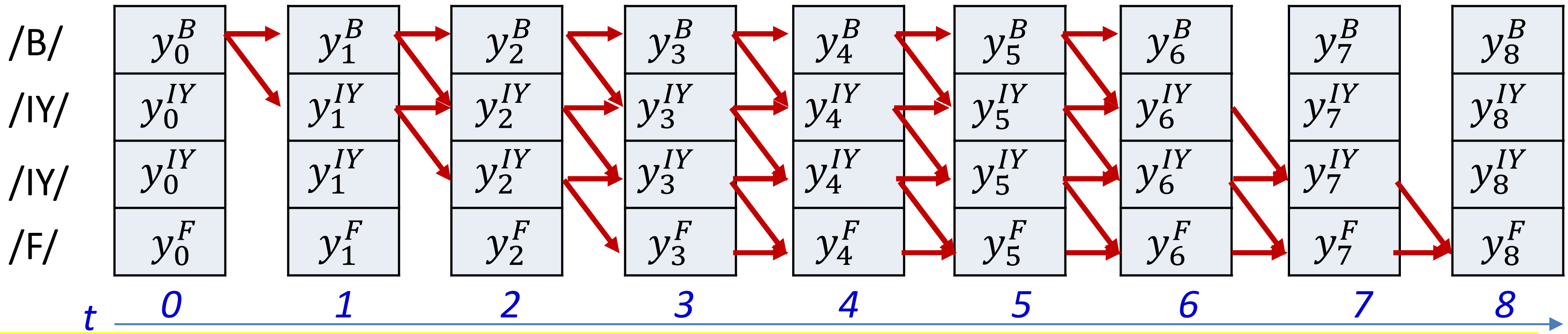


The modified forward output

- Note the extra "blank" at the output



Composing the graph for training



- The original method without blanks

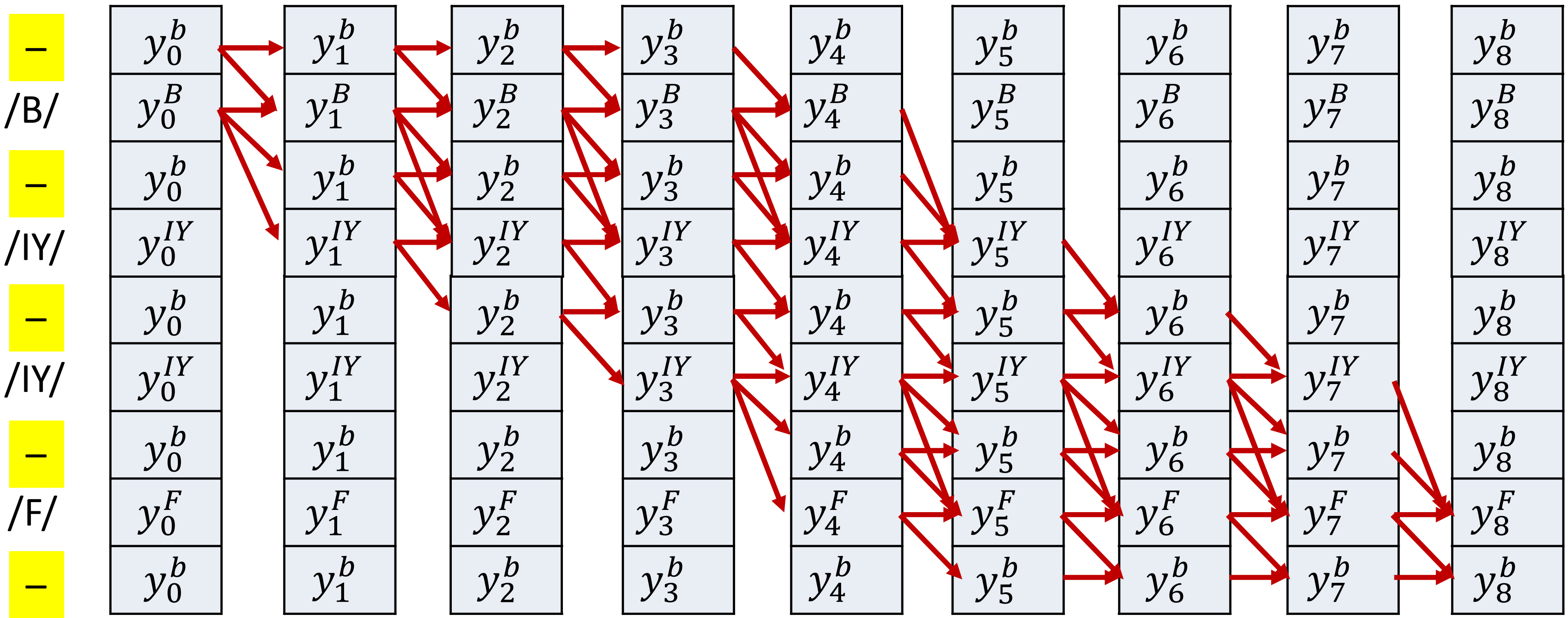
- **Changing the example to /B/ /IY/ /IY/ /F/ from /B/ /IY/ /F/ /IY/ for illustration**

Composing the graph for training

–	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
/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
–	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}
–	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}
–	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
/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
–	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

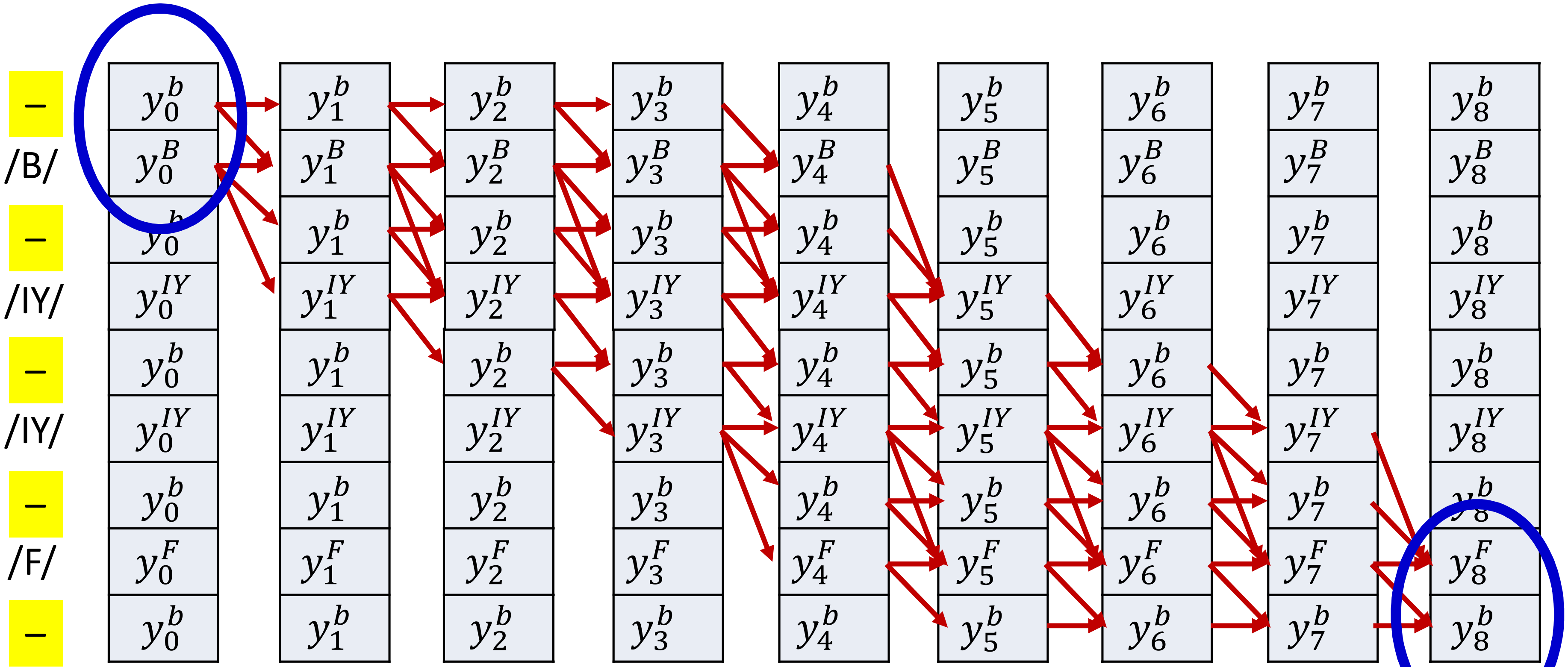
- With blanks
- Note: a row of blanks between any two symbols
- Also blanks at the very beginning and the very end

Composing the graph for training



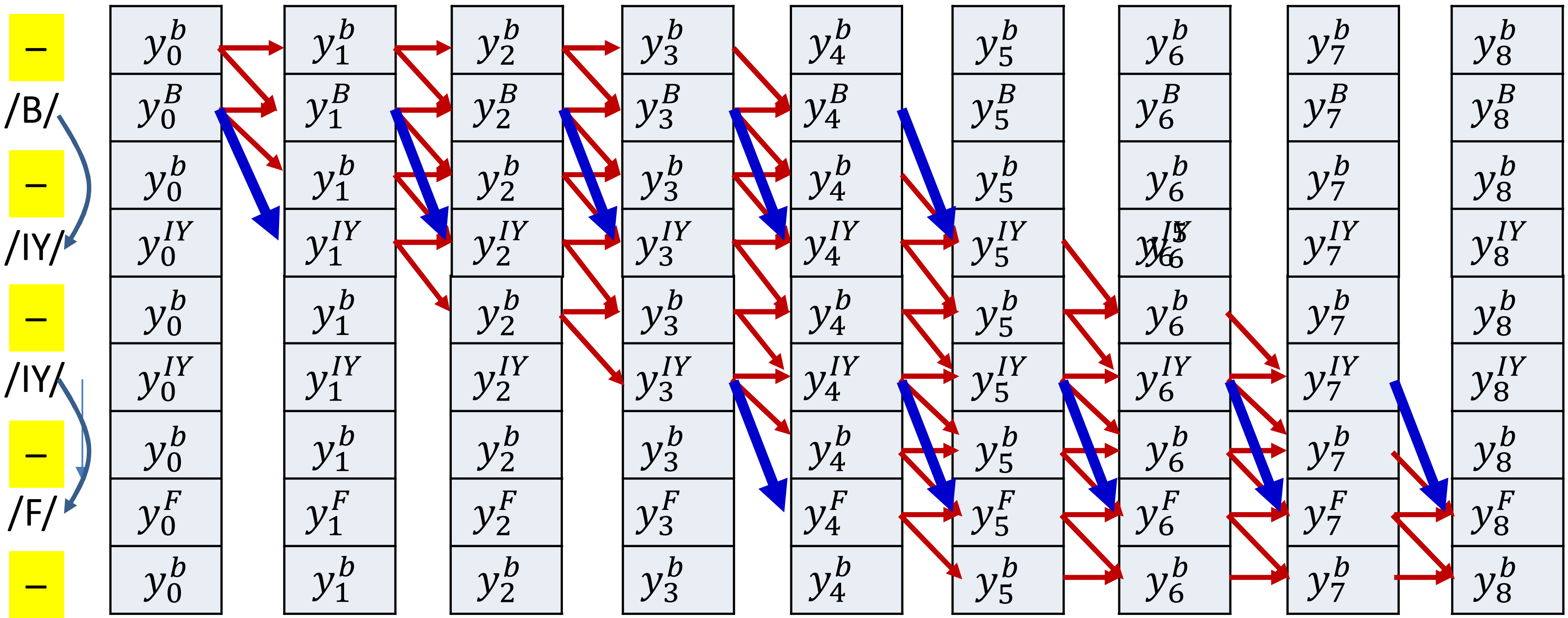
- Add edges such that all paths from initial node(s) to final node(s) unambiguously represent the target symbol sequence

Composing the graph for training



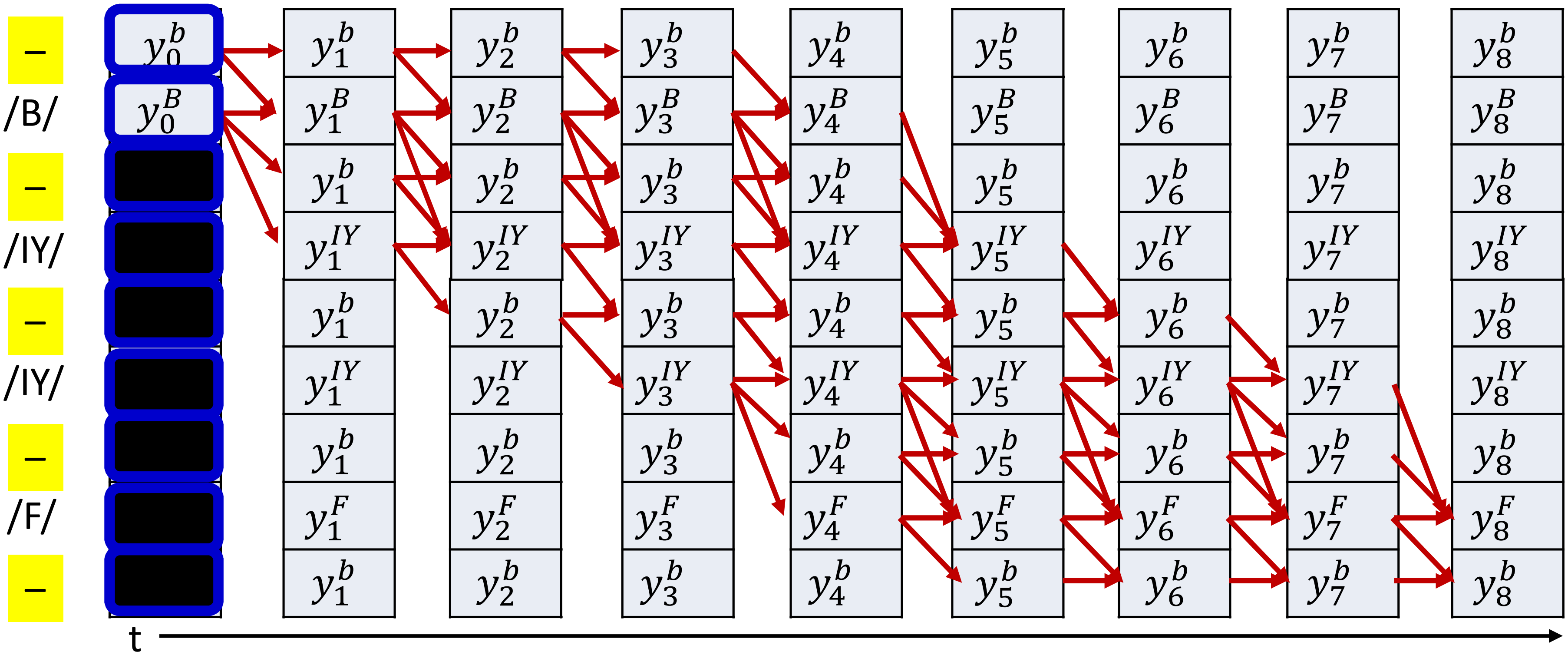
- The first and last column are allowed to also end at initial and final blanks

Composing the graph for training



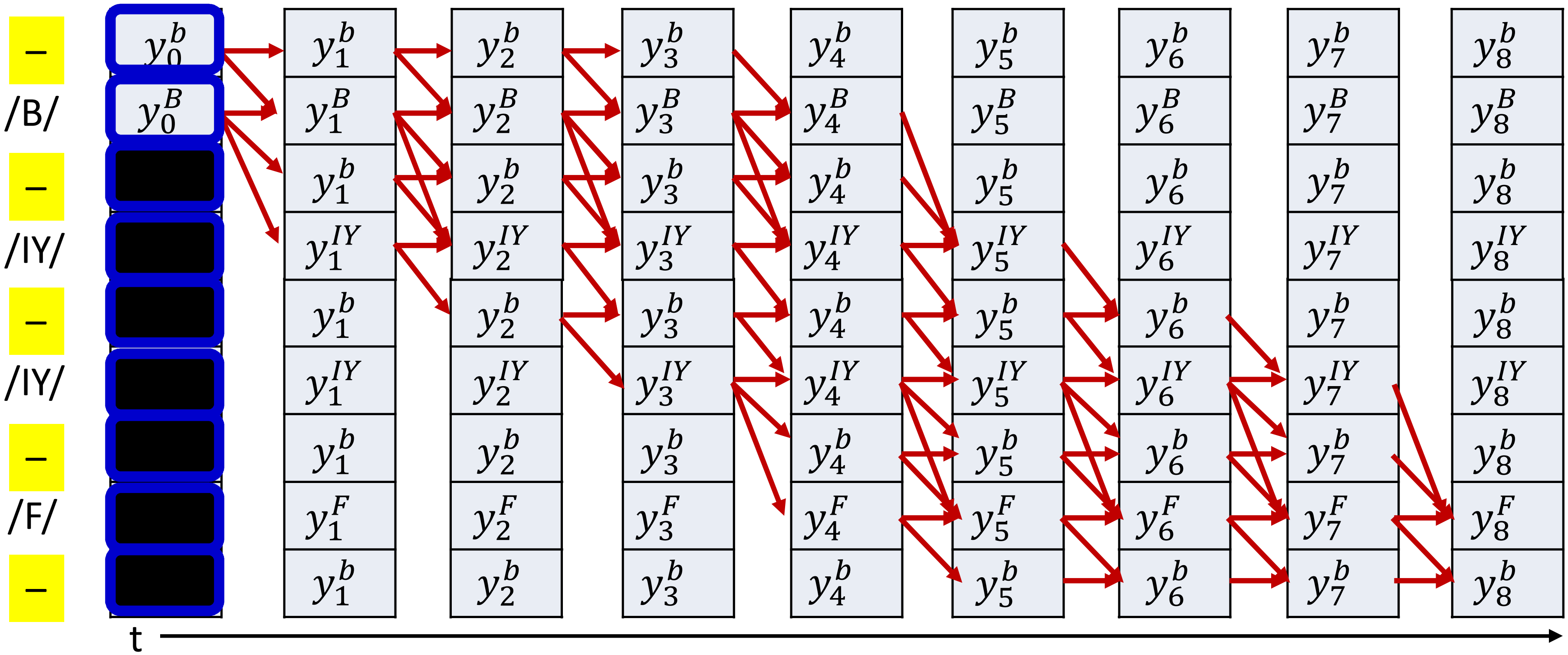
- The first and last column are allowed to also end at initial and final blanks
- Skips are permitted across a blank, but only if the symbols on either side are different
  - Because a blank is *mandatory between repetitions of a symbol* but *not required between distinct symbols*

Modified Forward Algorithm



- Initialization:
  - $\alpha(0,0) = y_0^b, \alpha(0,1) = y_0^b, \alpha(0,r) = 0 \quad r > 1$

Modified Forward Algorithm



- Iteration:

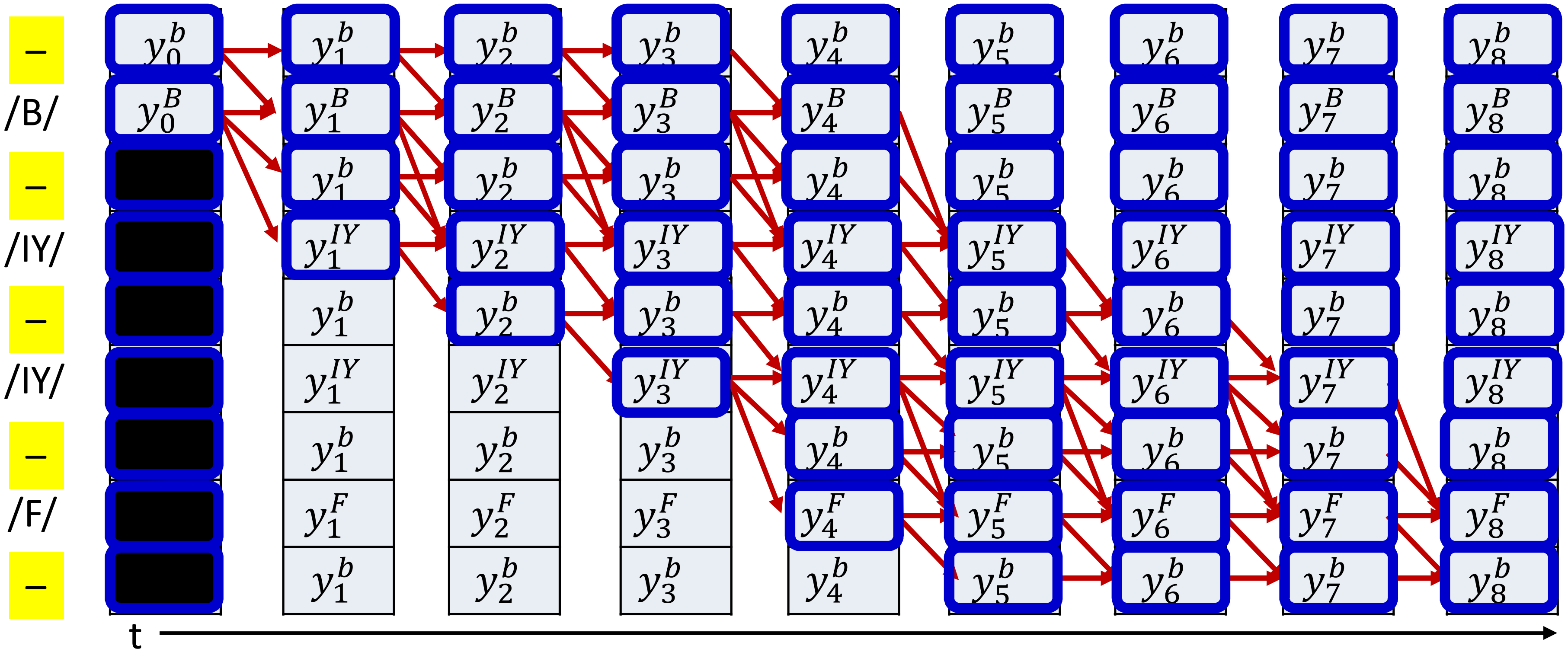
$$\alpha(t,r) = \left(\alpha(t-1,r) + \alpha(t-1,r-1)\right)y_t^{S(r)}$$

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

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

  - Otherwise

Modified Forward Algorithm



- Iteration:

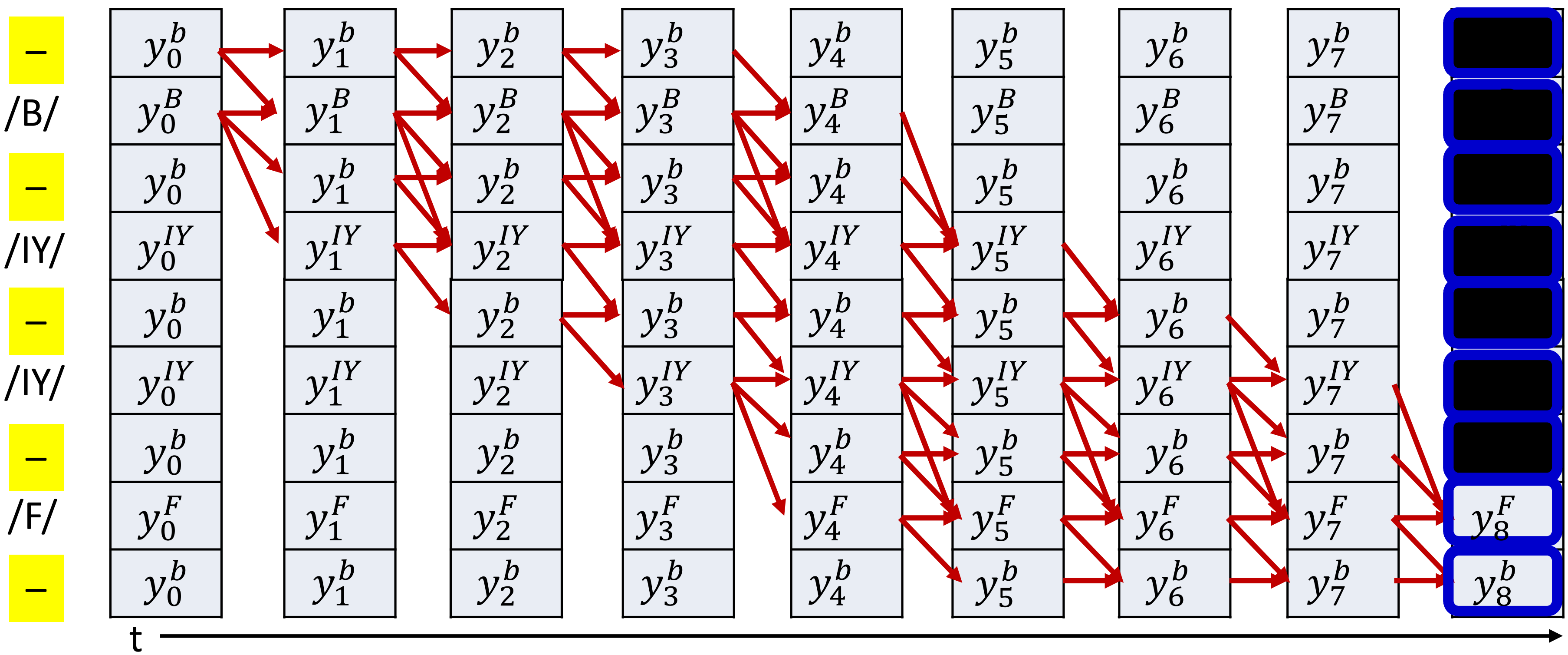
$$\alpha(t,r) = \big(\alpha(t-1,r) + \alpha(t-1,r-1)\big)y_t^{S(r)}$$

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

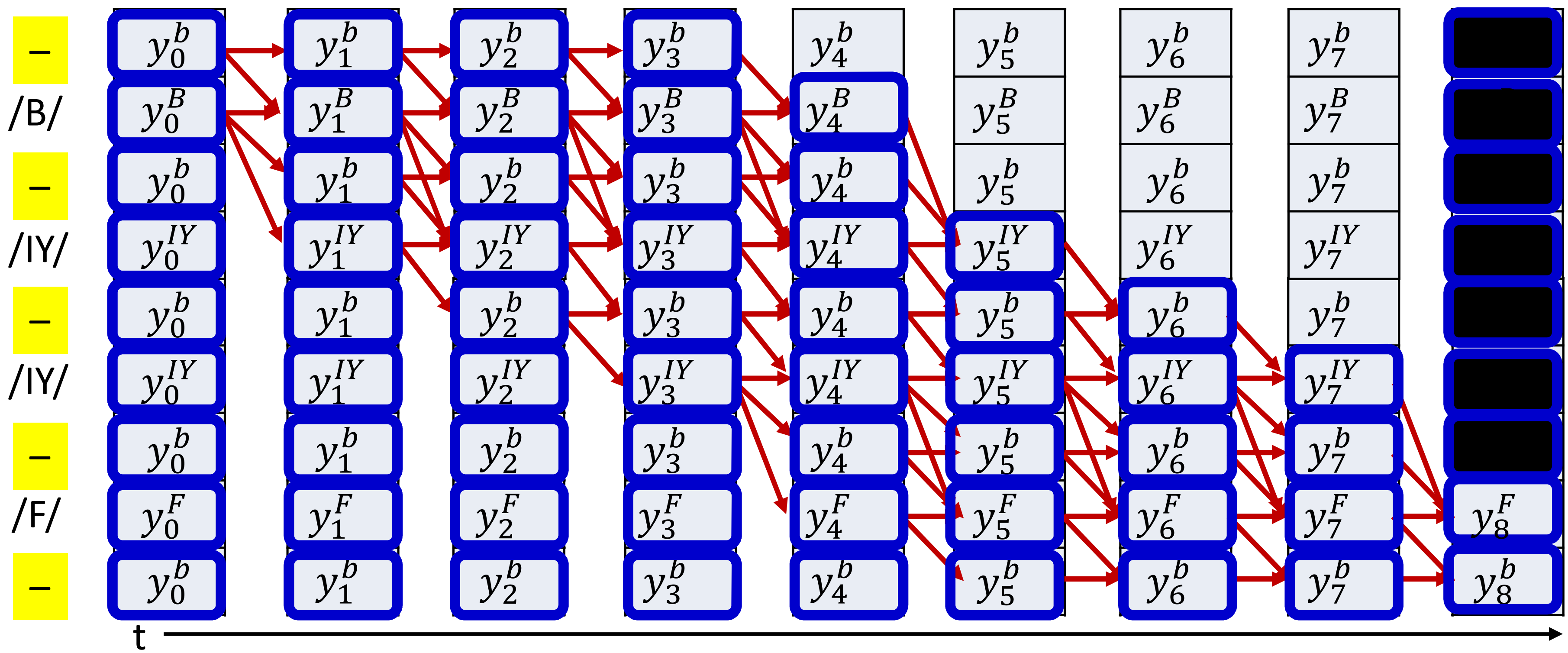
Modified Backward Algorithm



- Initialization:

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

Modified Backward Algorithm



- Iteration:

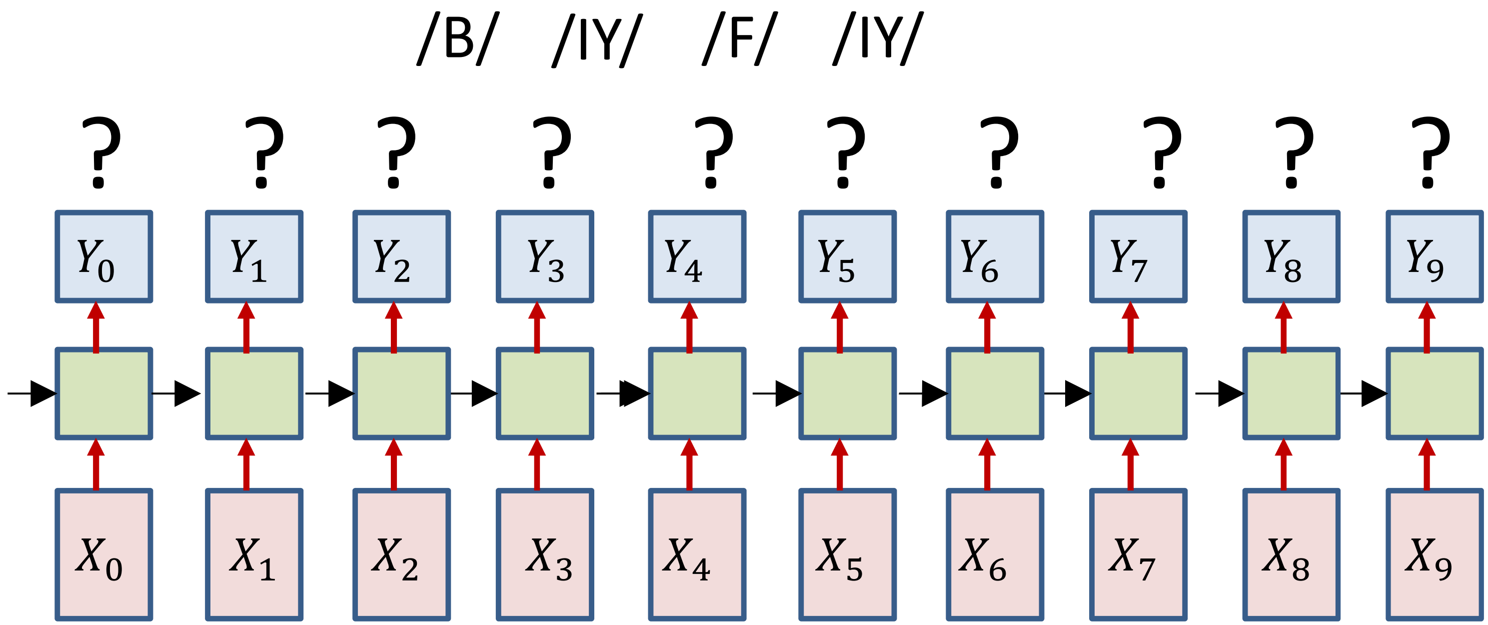
$$\beta(t,r) = \beta(t+1,r)y_t^{S(r)} \; + \beta(t+1,r+1)y_t^{S(r+1)}$$

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

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

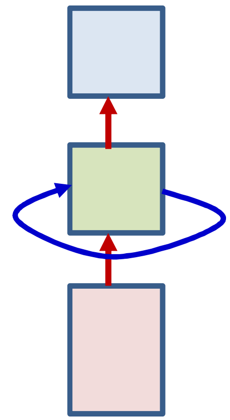
  - Otherwise

Overall training procedure for Seq2Seq with blanks



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

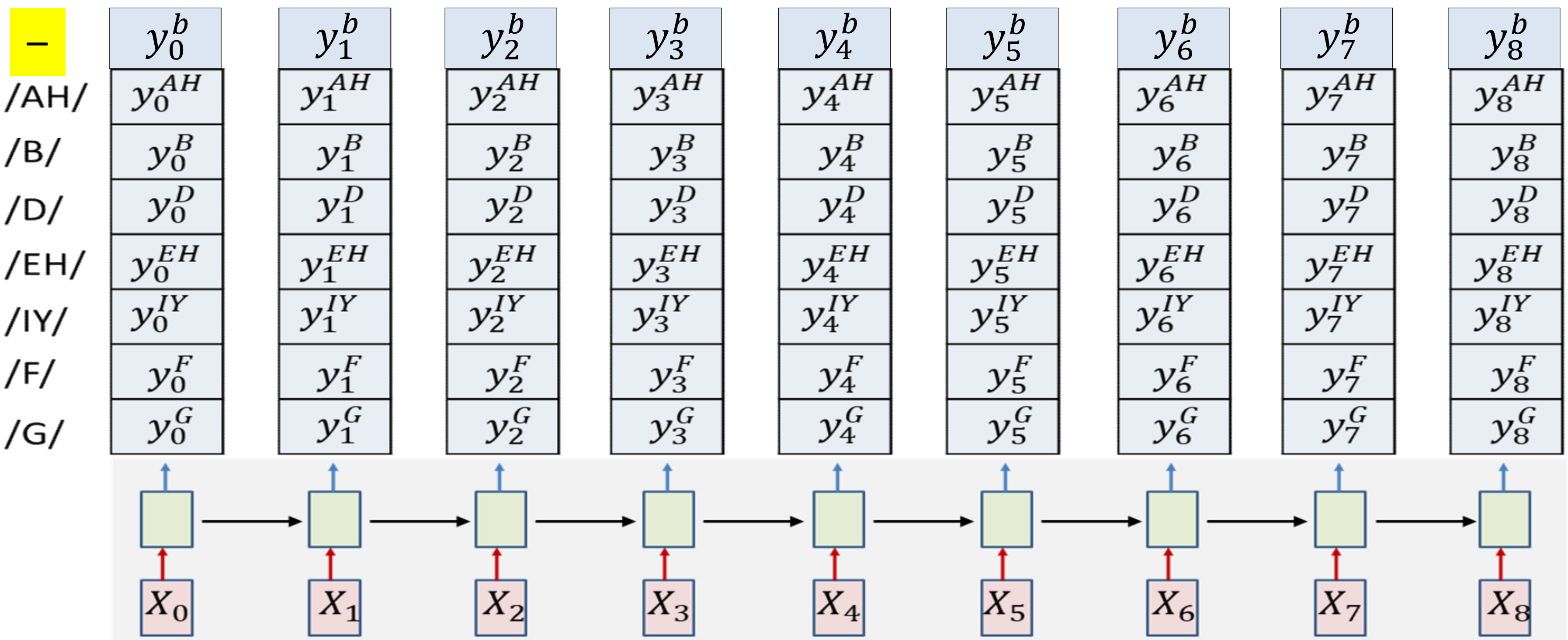
Overall training procedure



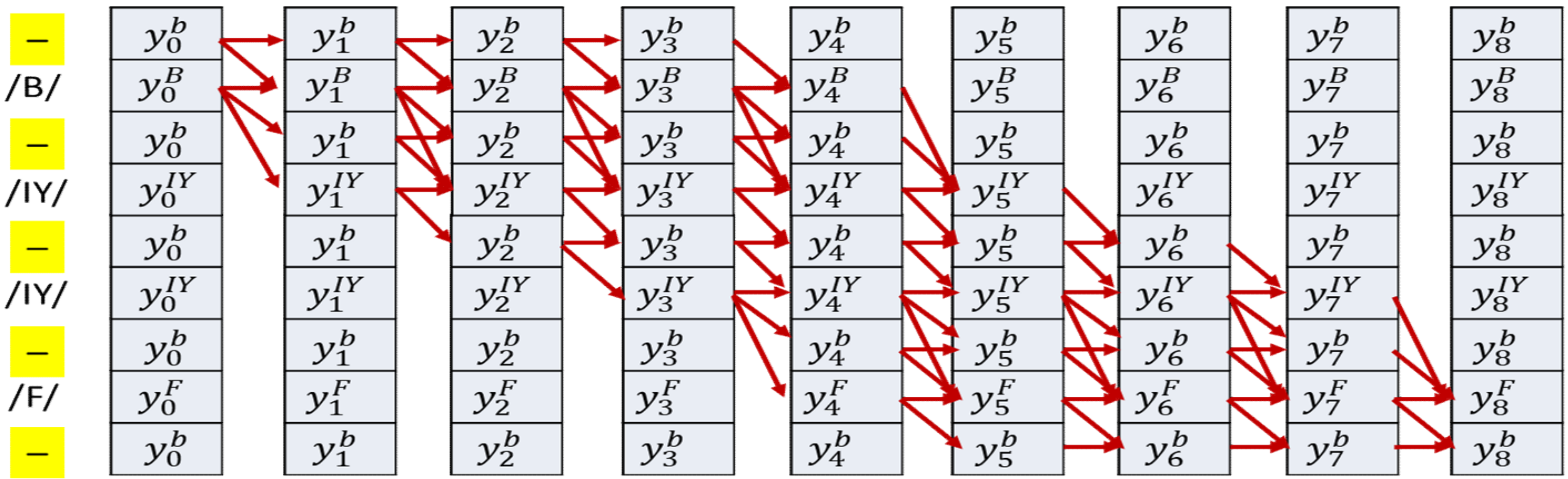
- **Step 1**: Setup the network
  - Typically many-layered LSTM
- **Step 2**: Initialize all parameters of the network
  - Include a “blank” symbol in vocabulary

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

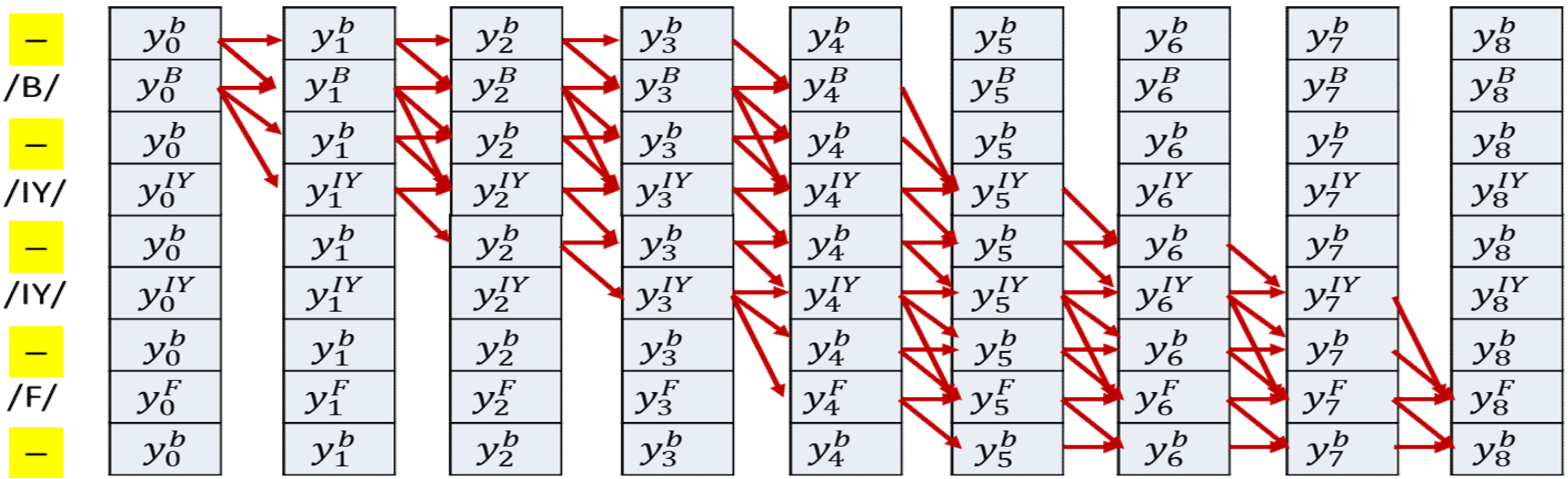


Overall training: Backward pass



- 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
  - **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

$$\nabla_{Y_t} DIV = \left[\frac{dDIV}{dy_t^1} \quad \frac{dDIV}{dy_t^2} \quad \cdots \quad \frac{dDIV}{dy_t^L}\right]$$

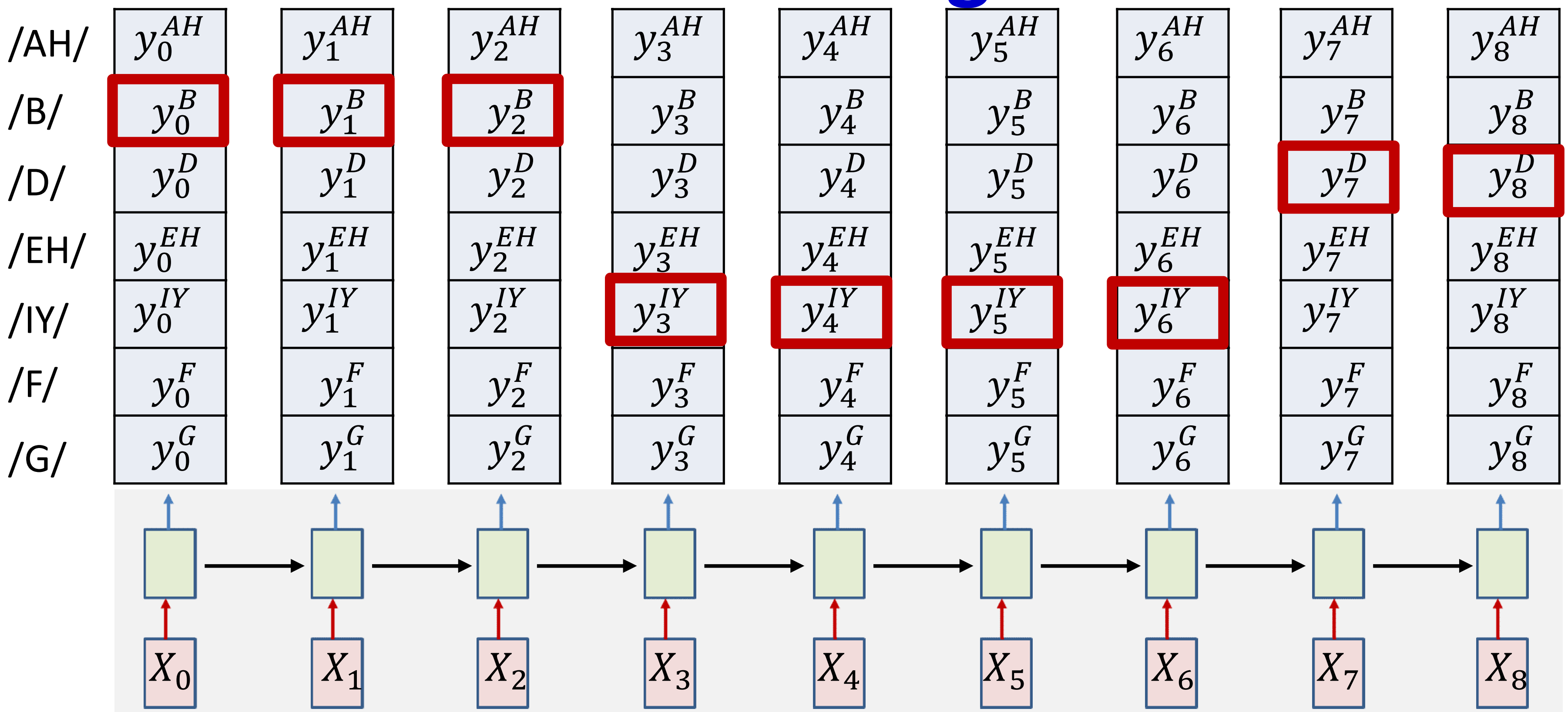
$$\frac{dDIV}{dy_t^l} = -\sum_{r\,:S(r)=l} \frac{\gamma(t,r)}{y_t^{S(r)}}$$

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

CTC: Connectionist Temporal Classification

- The overall framework we saw is referred to as CTC
  – Applies when “duplicating” labels at the output is considered acceptable, and when output sequence length < input sequence length

Returning to an old problem: Decoding



- Using a trained network, we can "decode" the symbol sequence for an input by tracing the most likely symbol at each frame and merging
- This is in fact a *suboptimal* decode that actually finds the most likely *synchronous* output sequence
  - Which is not necessarily the most likely *asynchronous* sequence
- **Nevertheless it is effective with CTC models that have blanks**

What a CTC system outputs

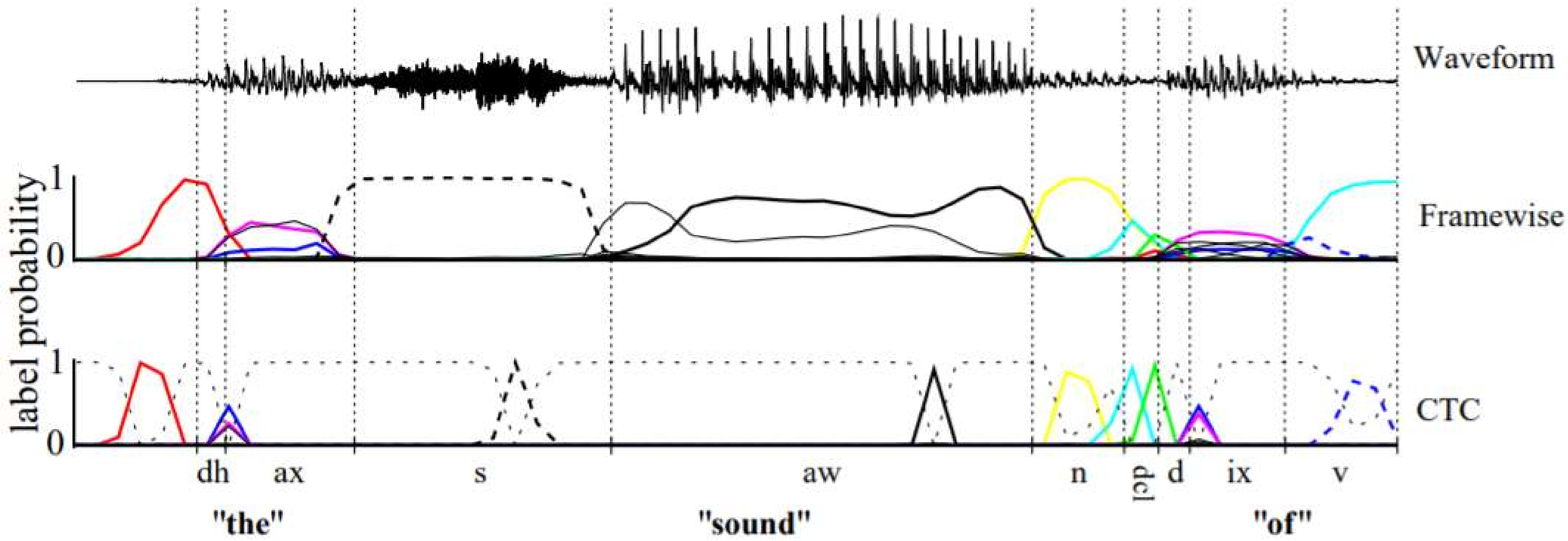


Figure 1. **Framewise and CTC networks classifying a speech signal.** The shaded lines are the output activations, corresponding to the probabilities of observing phonemes at particular times. The CTC network predicts only the sequence of phonemes (typically as a series of spikes, separated by 'blanks', 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
  - But are smeared..
- Better ability to find most likely symbol sequence, and can handle repetitions
- But this is still suboptimal..

Actual objective of decoding

- Want to find most likely asynchronous symbol sequence
  - /R/ /EH/ /D/

- What Viterbi finds: most likely synchronous symbol sequence
  - /R/ /R/ /R//EH//EH//EH//D/
  - Which must be compressed

- The likelihood of an asynchronous symbol sequence $\boldsymbol{S} = S_1, \ldots, S_K$, given an input $\boldsymbol{X} = X_1, \ldots, X_T$, is given by the forward algorithm

$$P(\boldsymbol{S}|\boldsymbol{X}) = P(\boldsymbol{S}, s_T = S_K|\mathbf{X}) = \alpha_{\mathbf{S}}(S_K, T)$$

Actual decoding objective

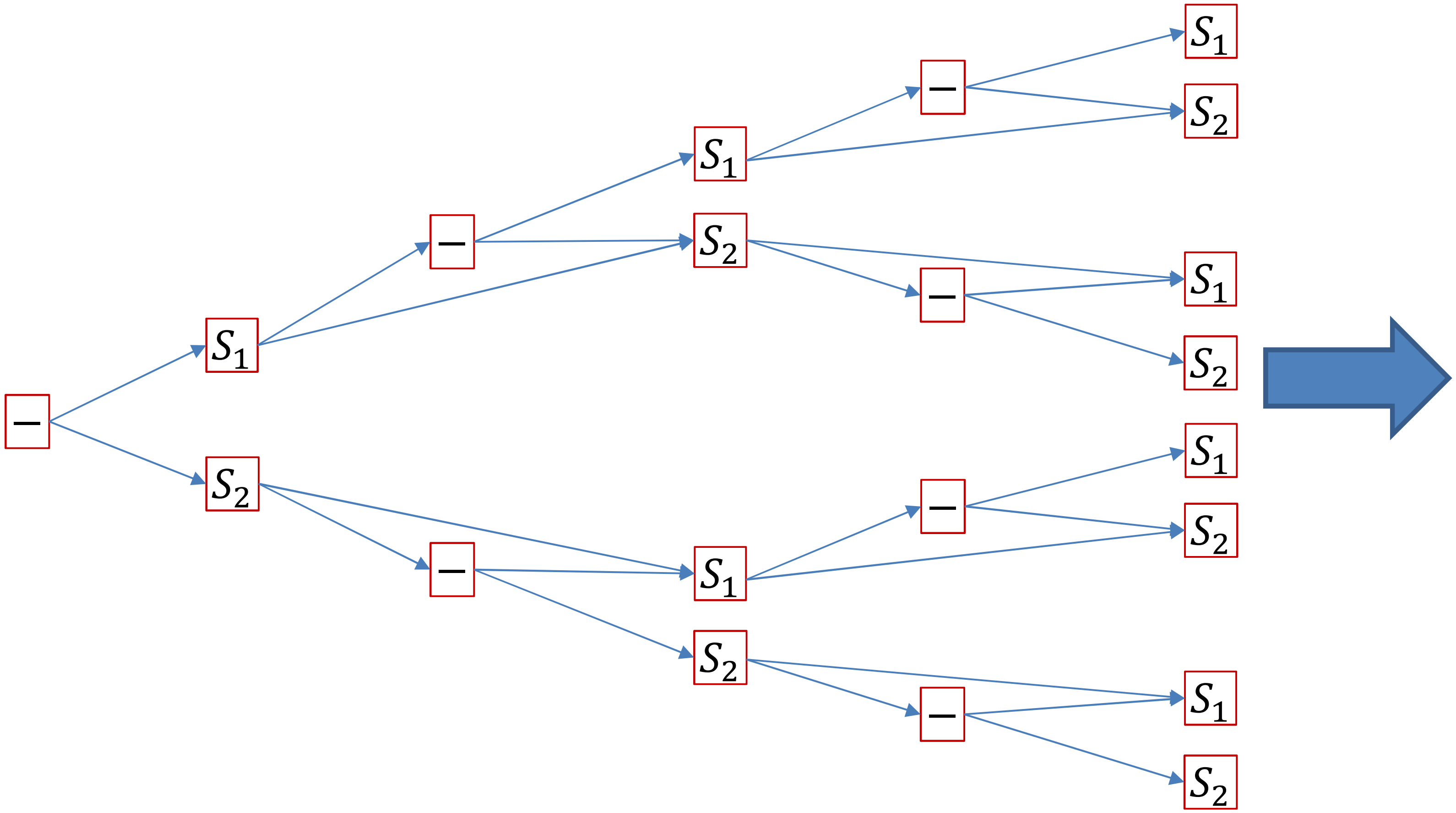
- Find the most likely (asynchronous) symbol sequence

$$\hat{\mathbf{S}} = \underset{\mathbf{S}}{\operatorname{argmax}}\, \alpha_{\mathbf{S}}(S_K, T)$$

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

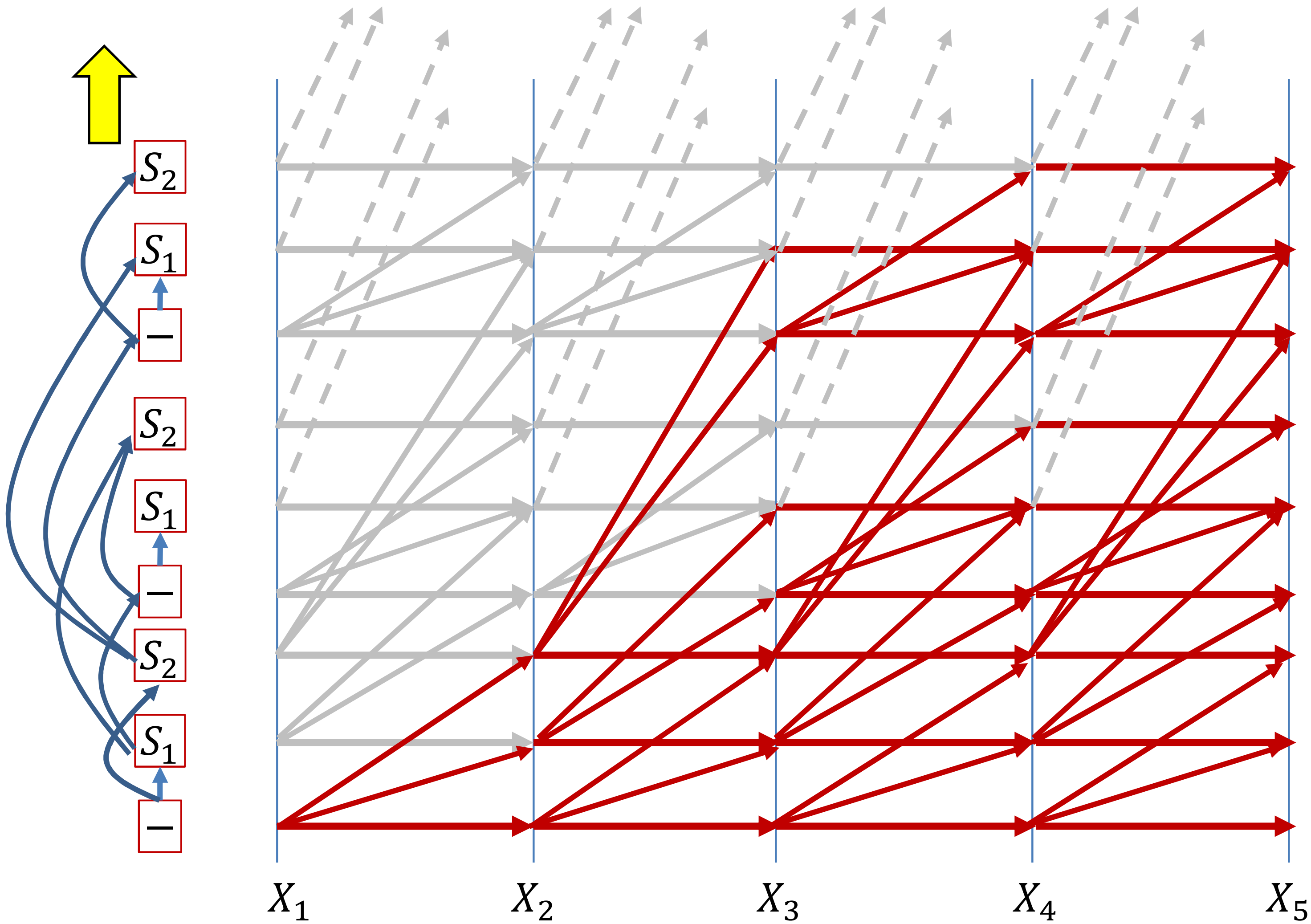
- Solution: Organize all possible symbol sequences as a (semi)tree

Hypothesis semi-tree



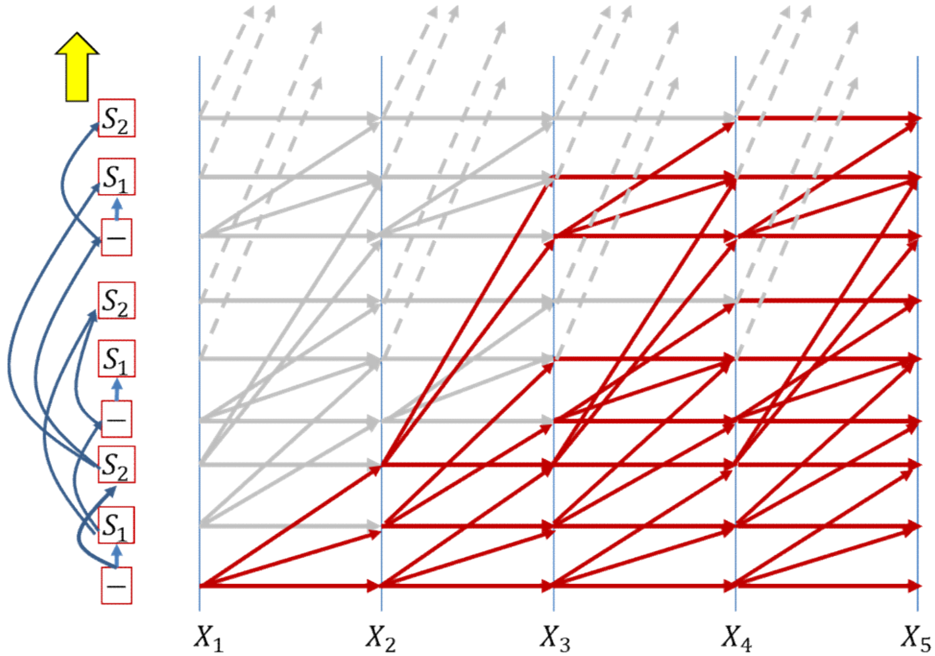
- The semi tree of hypotheses (assuming only 3 symbols in the vocabulary)
- Every symbol connects to every symbol other than itself
  - It also connects to a blank, which connects to every symbol including itself
- The simple structure repeats recursively
- Each node represents a unique (partial) symbol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

The decoding graph for the tree



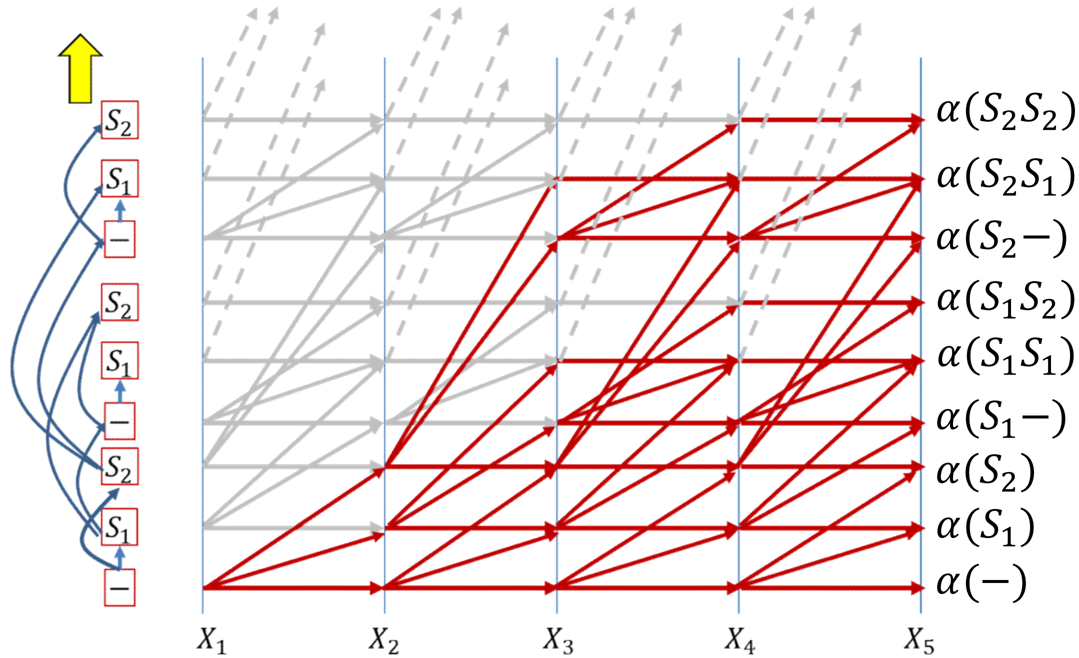
- Graph with more than 2 symbols will be similar but much more cluttered and complicated

The decoding graph for the tree



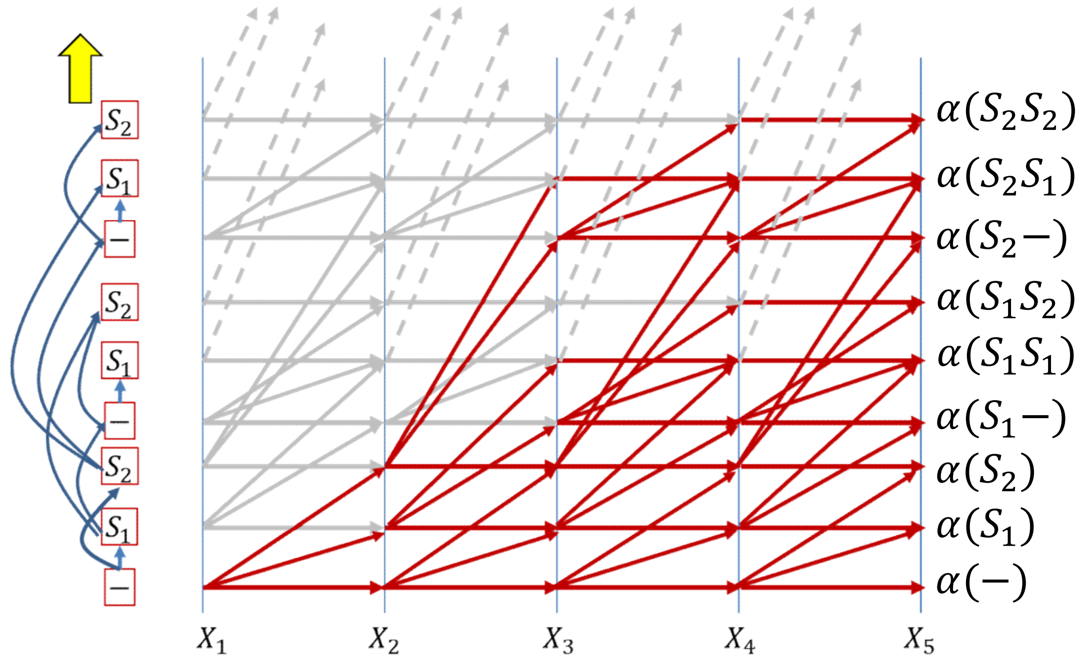
- The figure to the left is the tree, drawn in a vertical line
- The graph is just the tree unrolled over time
  - For a vocabulary of V symbols, every node connects out to V other nodes at the next time
- Every node in the graph represents a unique symbol sequence

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
  - Some sequences may have 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

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

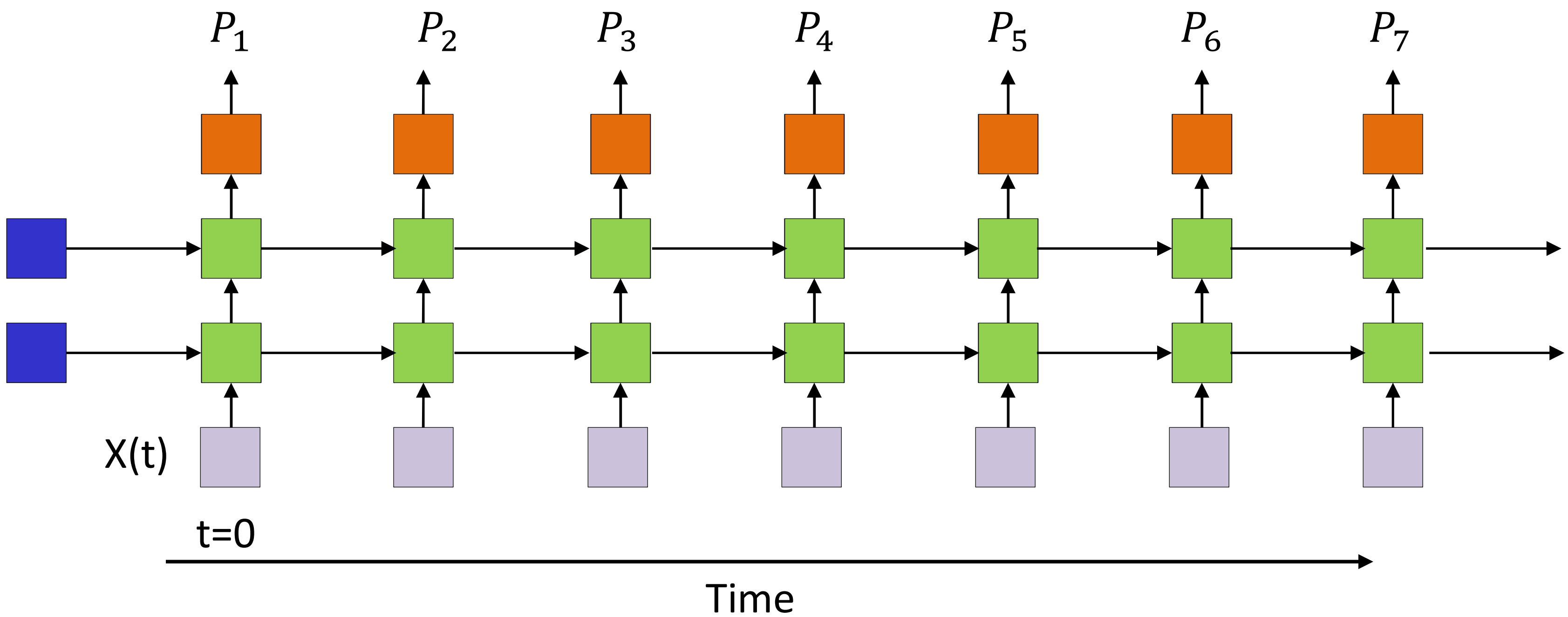
- The "blank" structure (with concurrent modifications to the forward-backward equations) is only one way to deal with the problem of repeating symbols
- Possible variants:
  – Symbols partitioned into two or more sequential subunits
    - No blanks are required, since subunits must be visited in order
  – 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)

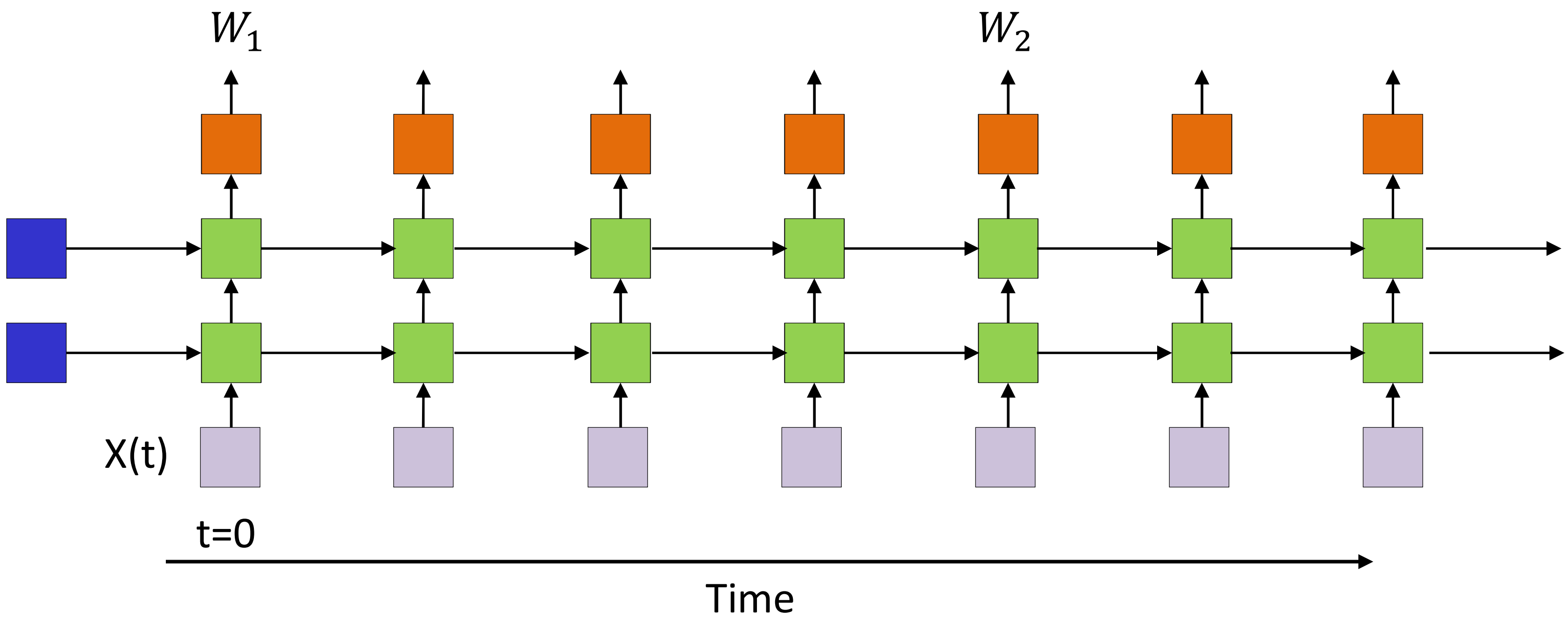
- Handwriting recognition

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

CNN-LSTM-DNN for speech recognition

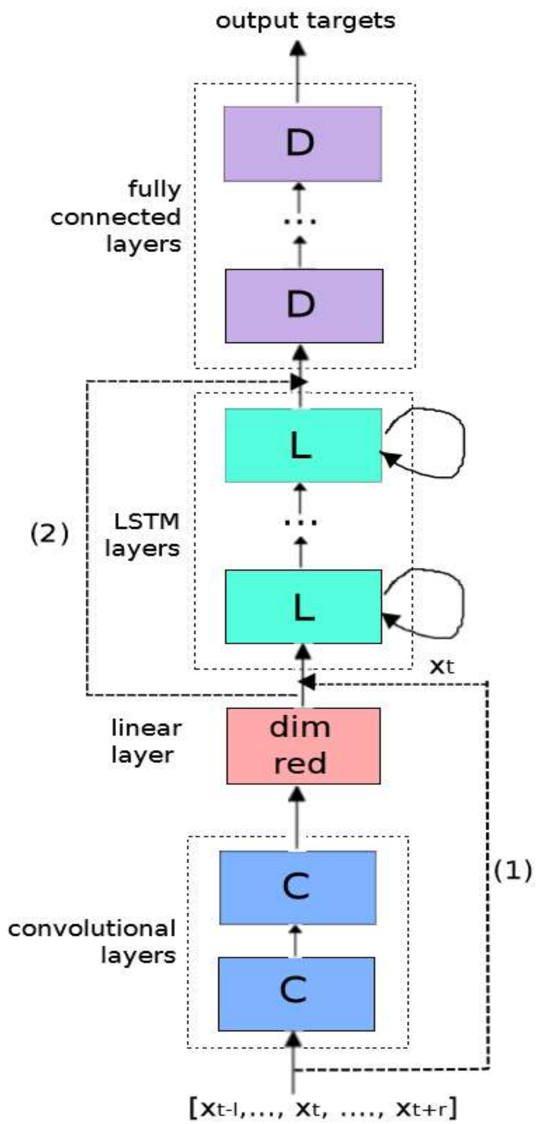
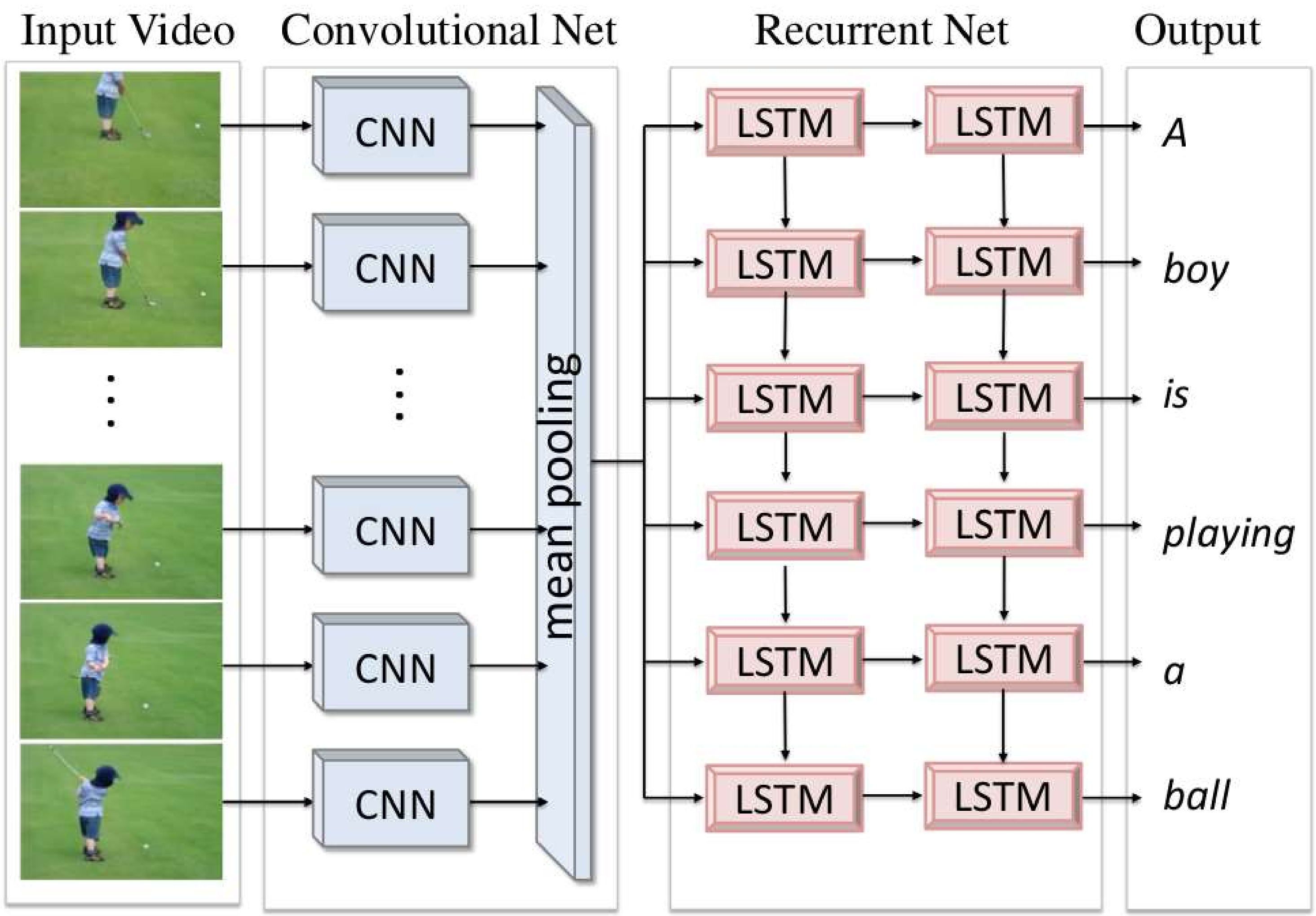


Fig. 1. CLDNN Architecture

Ensembles of RNN/LSTM, DNN, & Conv Nets (CNN) :

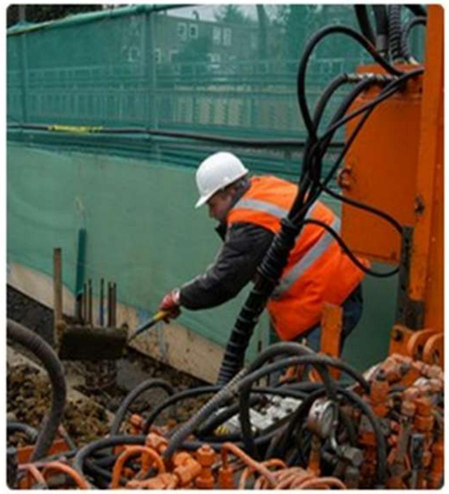
T. Sainath, O. Vinyals, A. Senior, H. Sak. "Convolutional, Long Short-Term Memory, Fully Connected Deep Neural Networks," ICASSP 2015.

Translating Videos to Natural Language Using Deep Recurrent Neural Networks



Translating Videos to Natural Language Using Deep Recurrent Neural Networks
Subhashini Venugopalan, Huijun Xu, Jeff Donahue, Marcus Rohrbach, Raymond Mooney, Kate Saenko
North American Chapter of the Association for Computational Linguistics, Denver, Colorado, June 2015.

"man in black shirt is playing guitar."



"construction worker in orange safety vest is working on road."

"two young girls are playing with lego toy."

"boy is doing backflip on wakeboard."

"a young boy is holding a baseball bat."

"a cat is sitting on a couch with a remote control."

"a woman holding a teddy bear in front of a mirror."

"a horse is standing in the middle of a road."

Not explained

- Can be combined with CNNs
  - Lower-layer CNNs to extract features for RNN

- Can be used in tracking
  - Incremental prediction