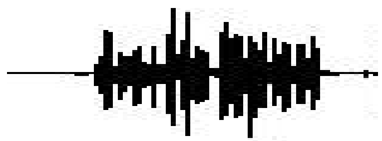
Deep Learning
Sequence to Sequence models: Connectionist Temporal Classification

5 March 2018

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

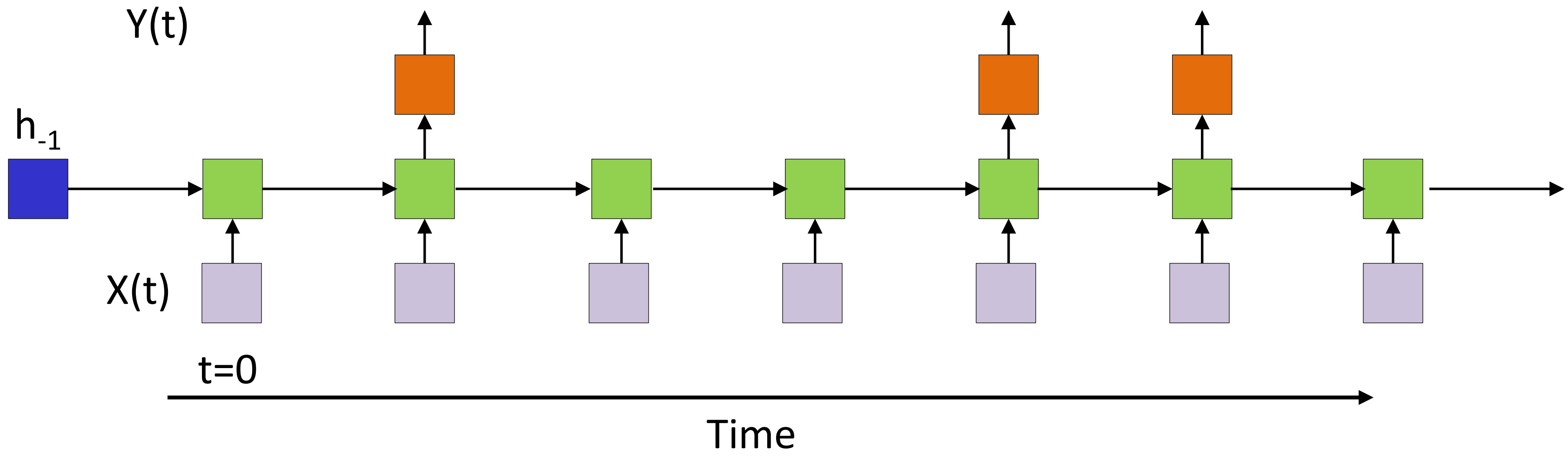
Sequence to sequence

 → Seq2seq → I ate an apple

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

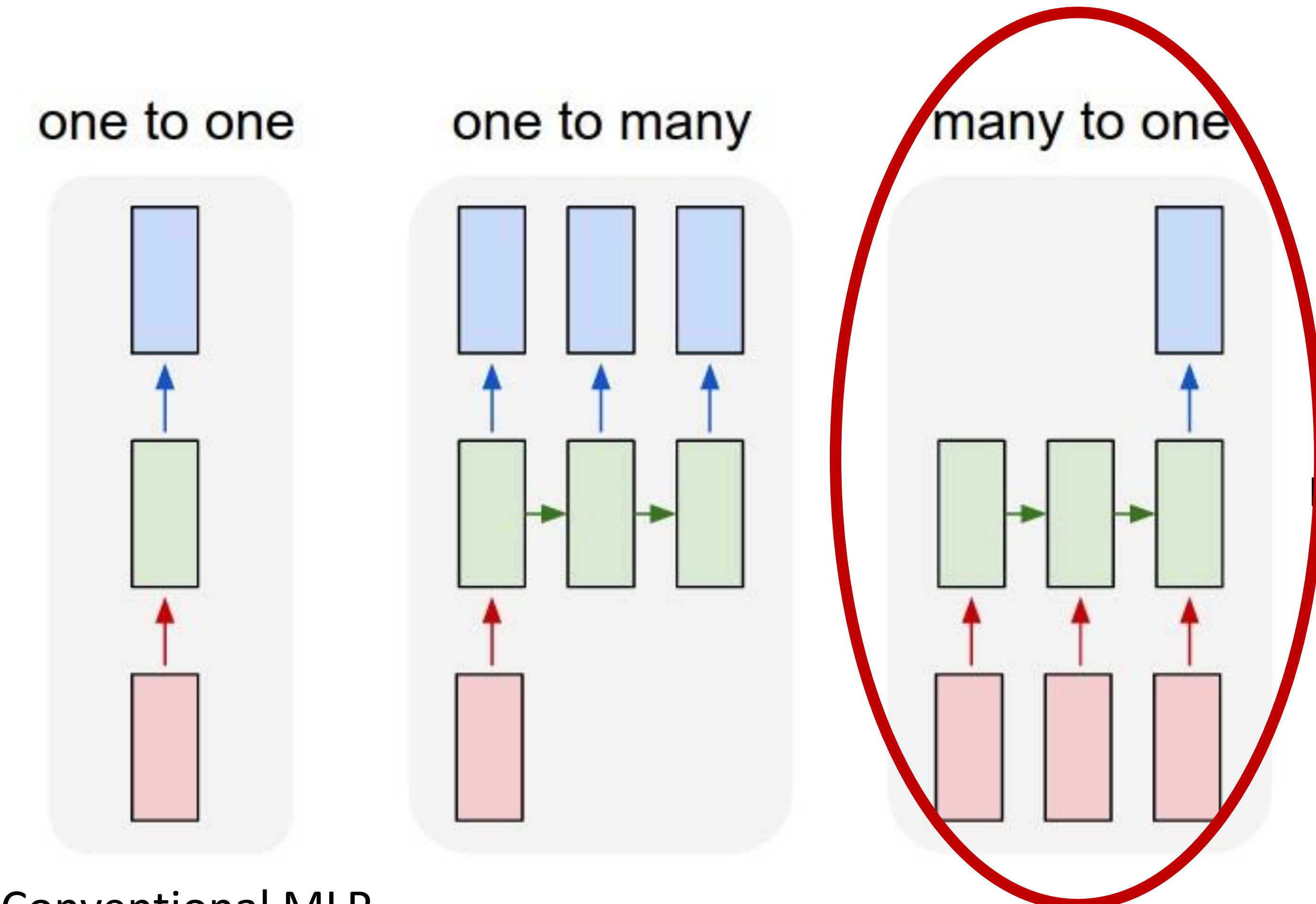
- Sequence goes in, sequence comes out
- No notion of “synchrony” between input and output
  - May even not have a notion of “alignment”
    - E.g.   “I ate an apple” → “Ich habe einen apfel gegessen”

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

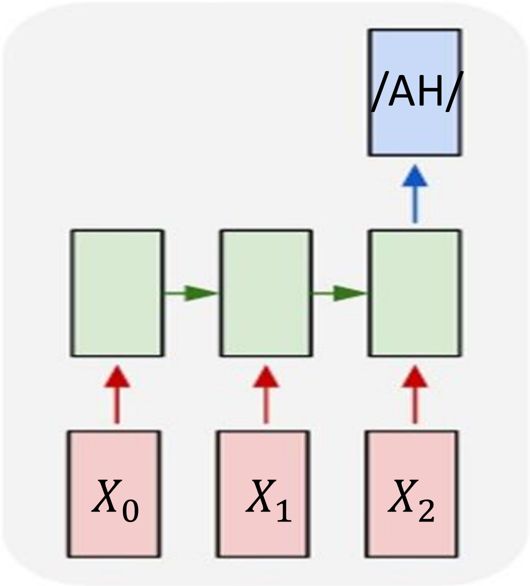
Variants on recurrent nets



Images from Karpathy

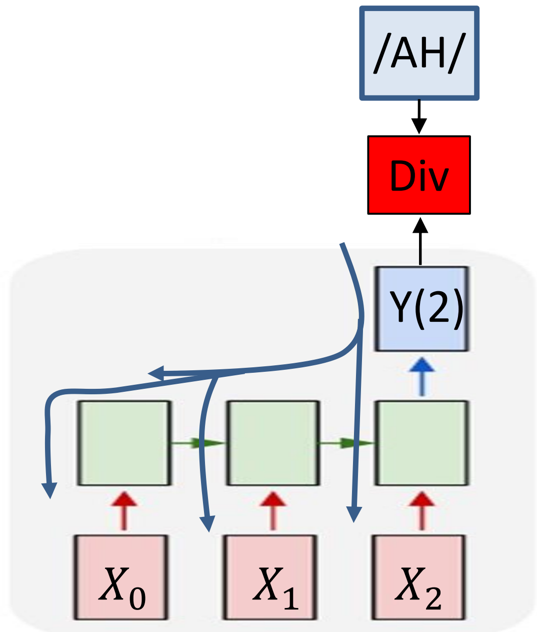
- 1: Conventional MLP
- 2: Sequence *generation*, e.g. image to caption
- 3: Sequence based *prediction or classification,* e.g. Speech recognition, text classification

Basic model



- Sequence of inputs produces a single output

Training

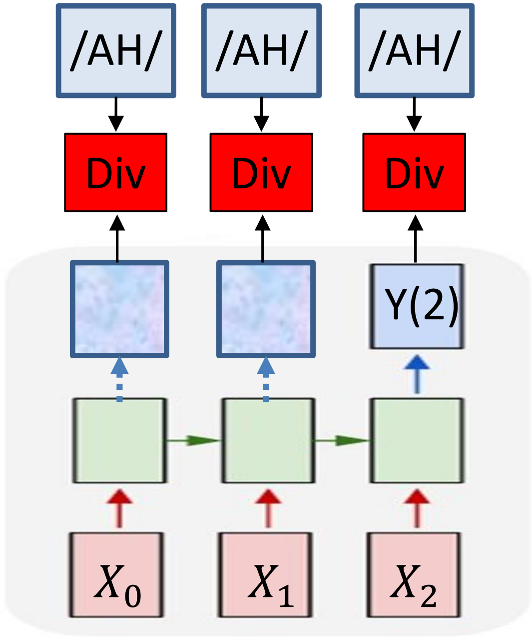


- The Divergence is only defined at the final input
  - $DIV\left(Y_{target}, Y\right) = Xent(Y(T), Phoneme)$
- This divergence must propagate through the net to update all parameters
- Ignores outputs at intermediate steps

Training

Fix: Use these outputs too.

These too must ideally point to the correct phoneme



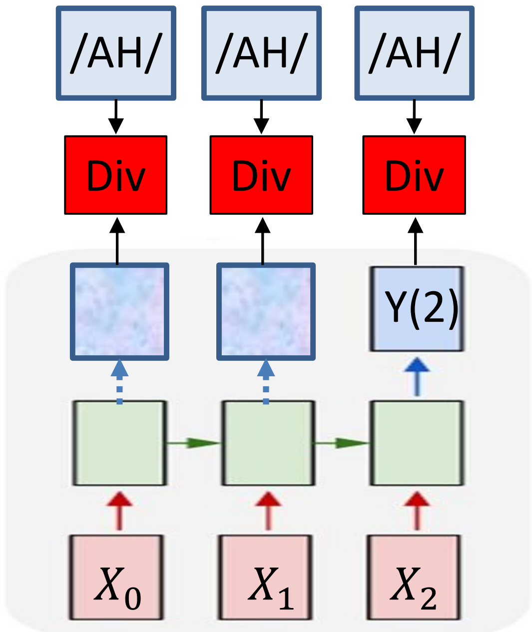
- Exploiting the untagged inputs: assume the same output for the entire input
- Define the divergence everywhere

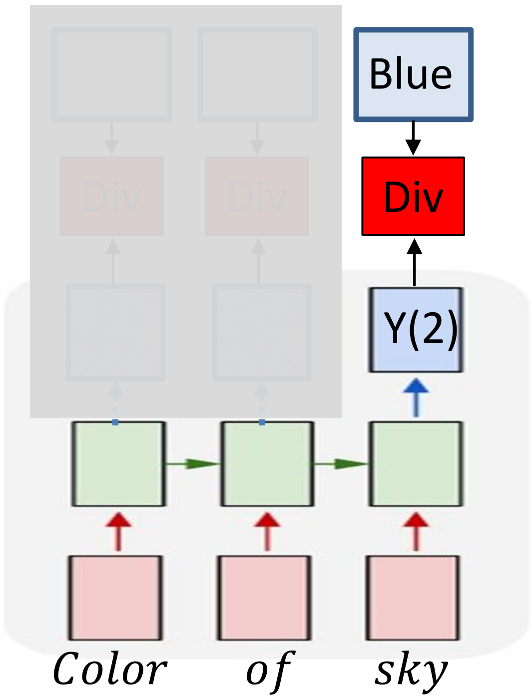
$$DIV\left(Y_{target}, Y\right) = \sum_t w_t Xent(Y(t), Phoneme)$$

Training

Fix: Use these outputs too.

These too must ideally point to the correct phoneme



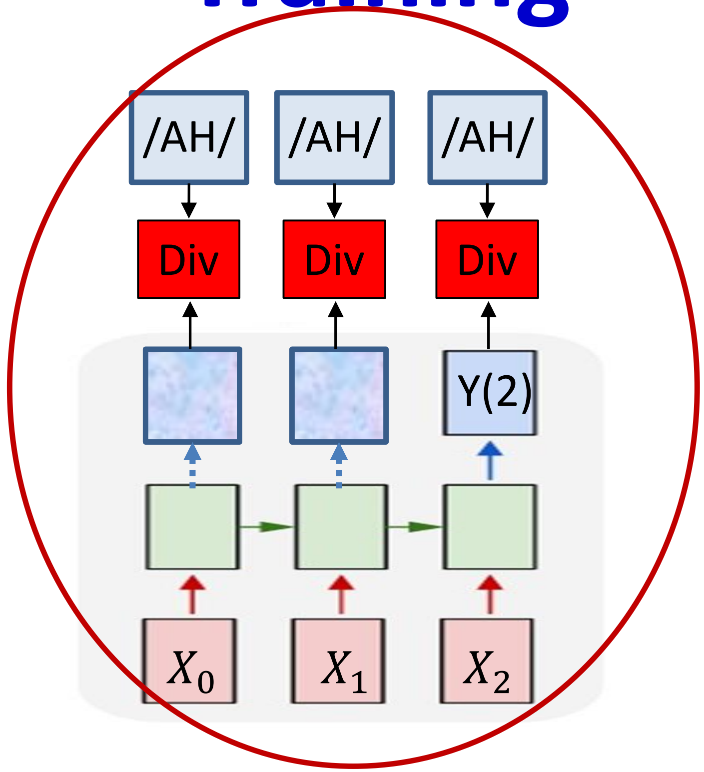


- Define the divergence everywhere

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

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

Training



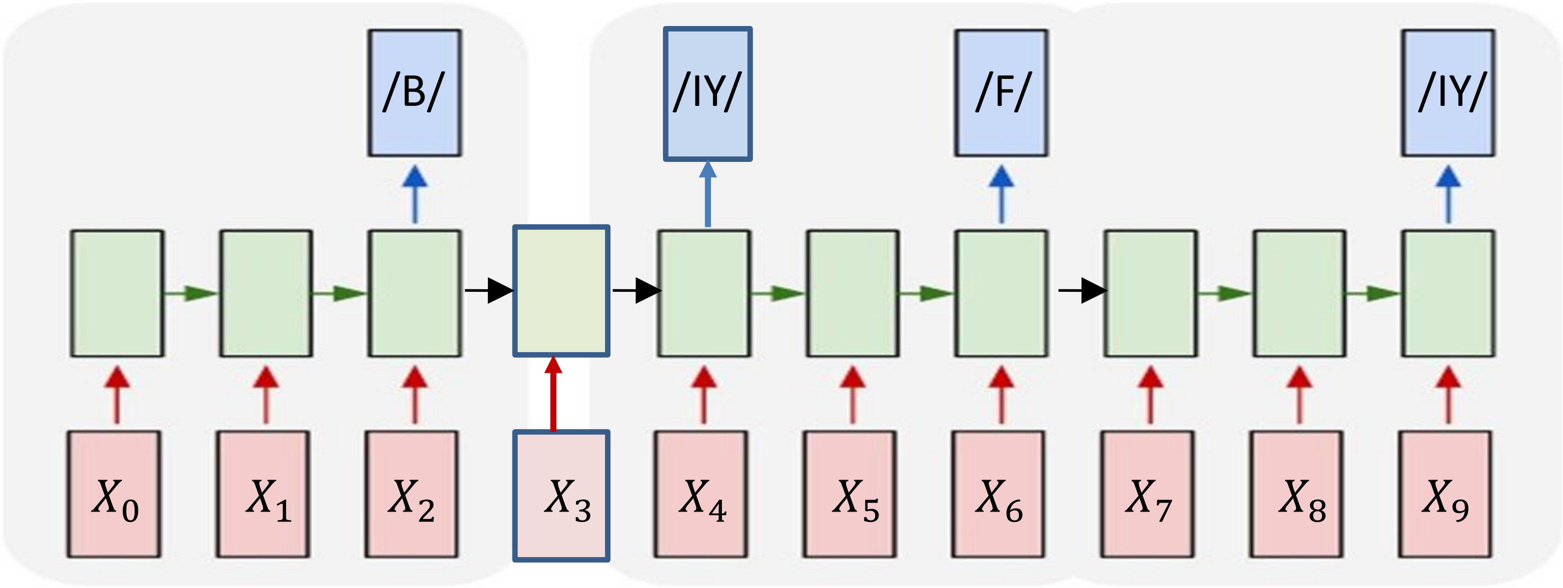
We will initially focus on the class of problem where uniform weights are reasonable (e.g speech recognition)

- Define the divergence everywhere

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

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

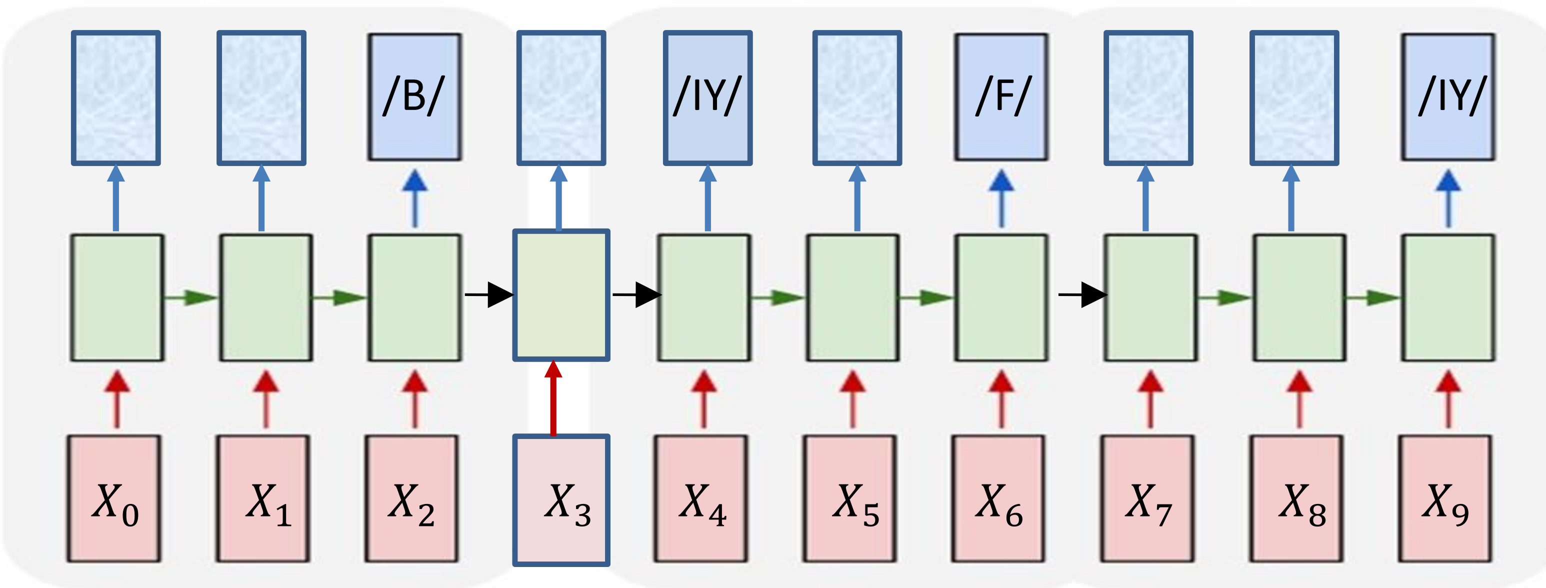
The more complex problem



- 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

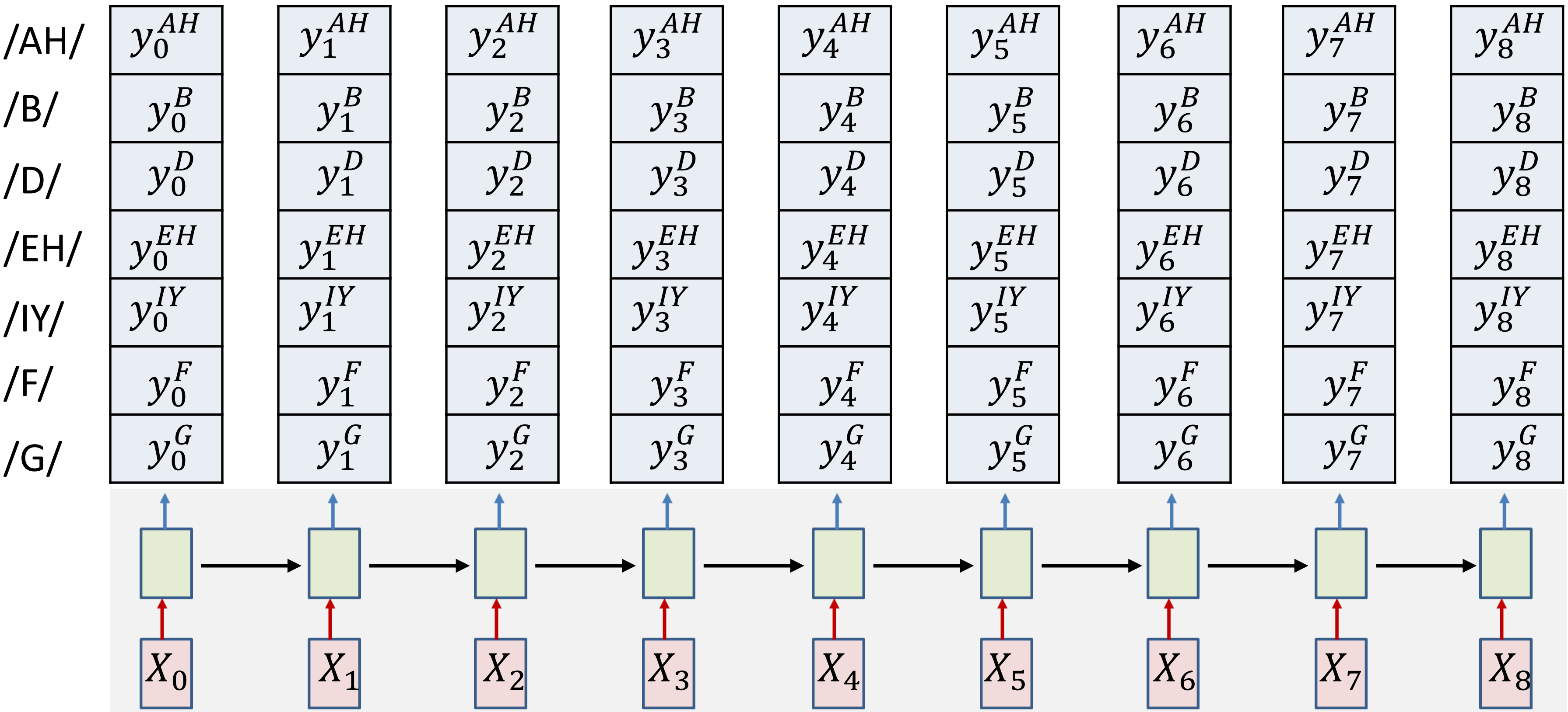
- But this simple extension complicates matters..

The *sequence-to-sequence* problem



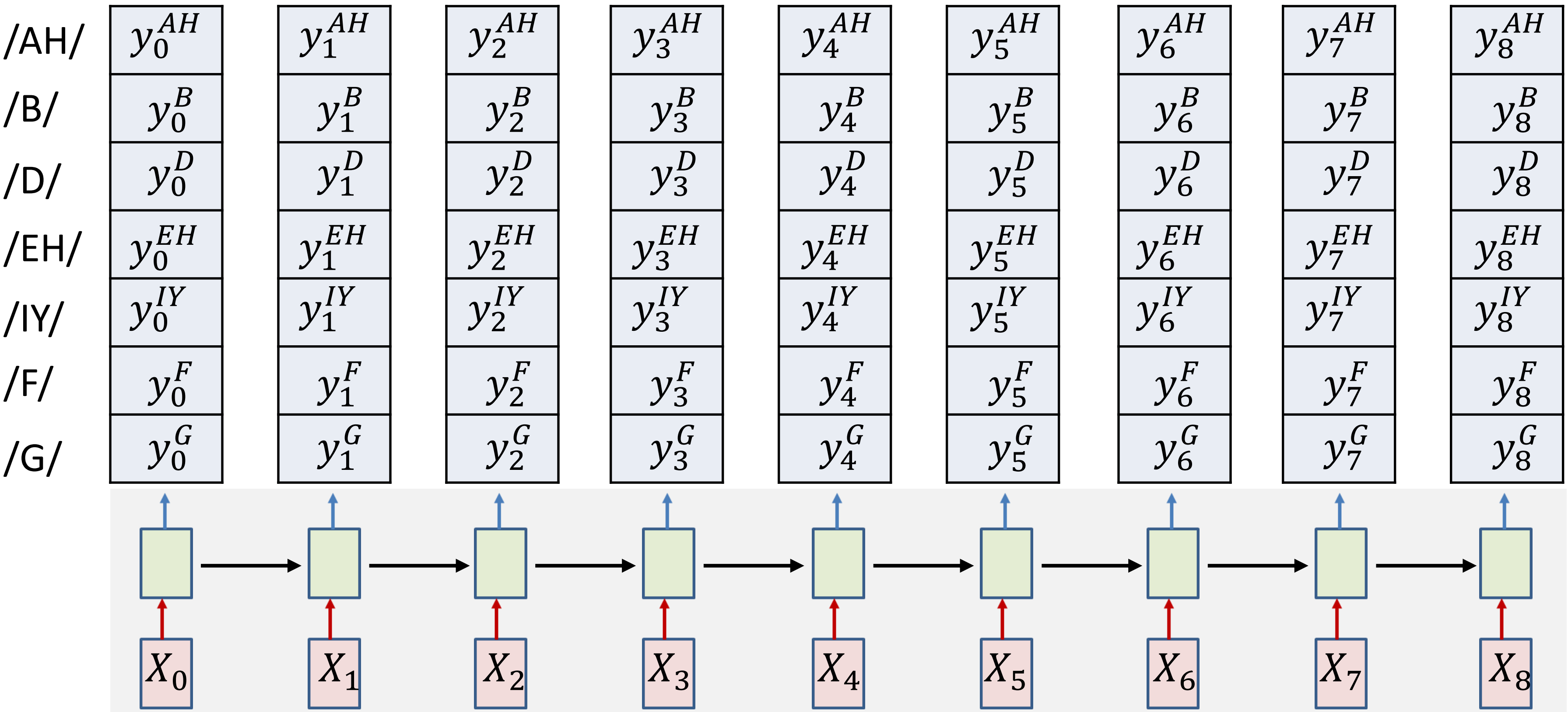
- How do we know *when* to output symbols
  - In fact, the network produces outputs at *every* time
  - *Which* of these are the *real* outputs?

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. $y_4^D = prob(s_4 = D|X_0 \dots X_4)$

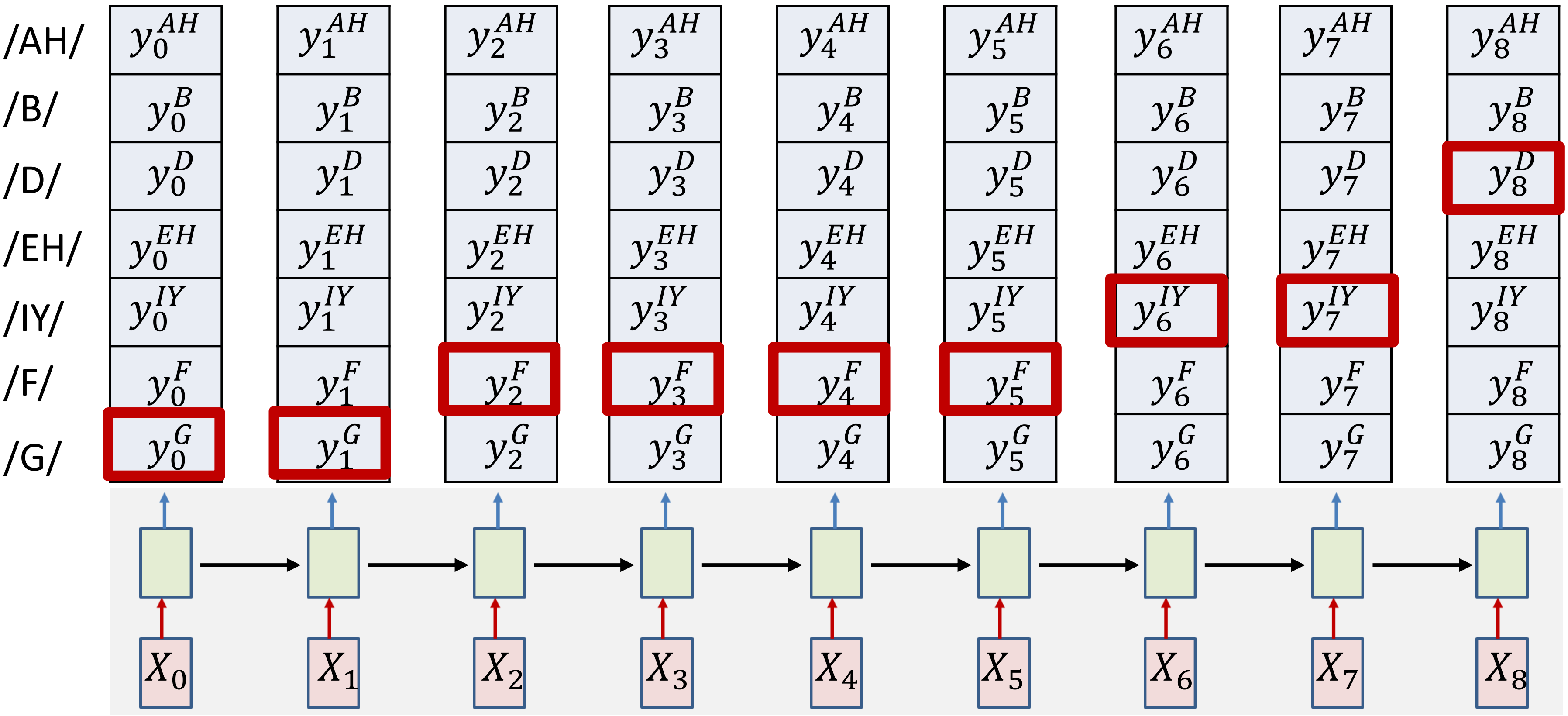
Overall objective



- Find most likely symbol sequence given inputs

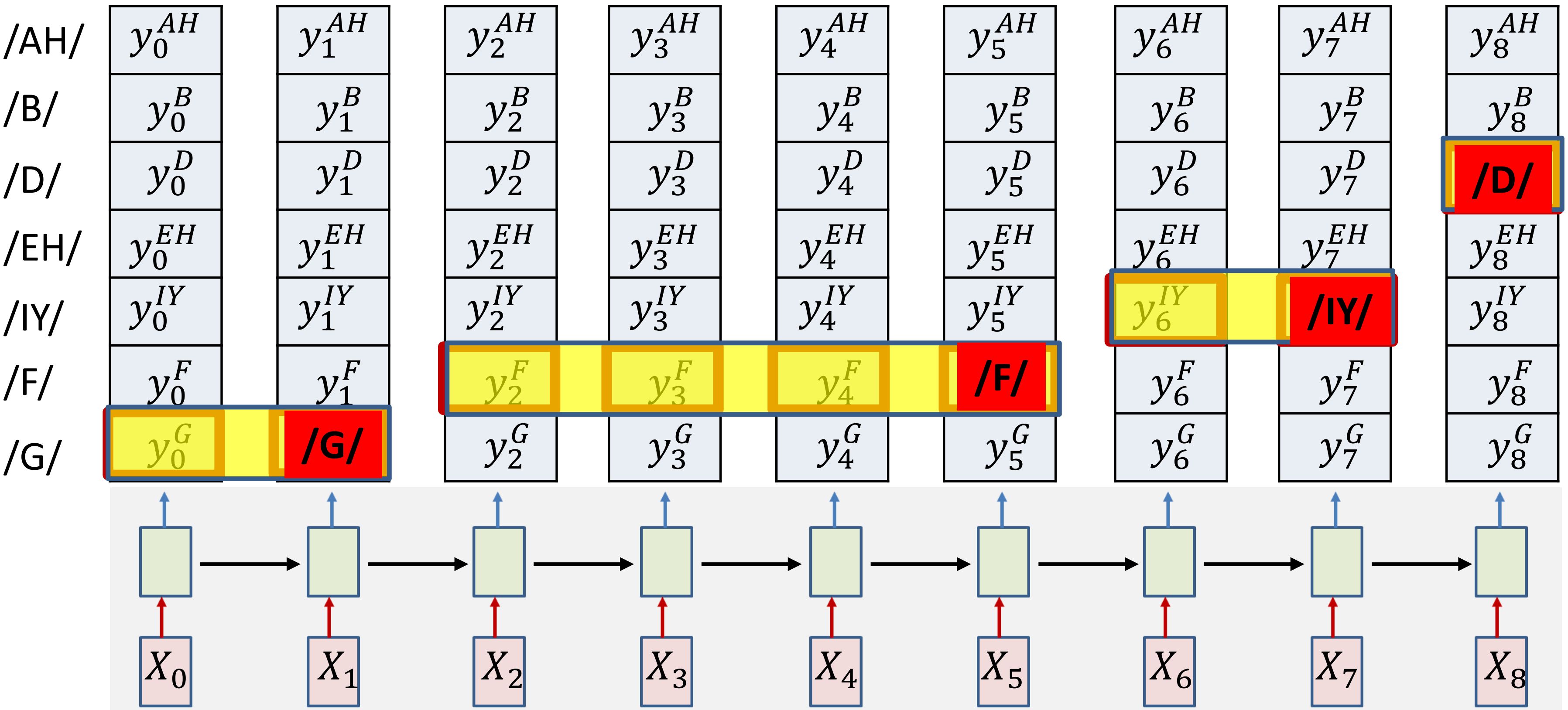
$$S_0 \ldots S_{K-1} = \underset{S'_0 \ldots S'_{K-1}}{\operatorname{argmax}} \; prob(S'_0 \ldots S'_{K-1} | X_0 \ldots X_{N-1})$$

Finding the best output



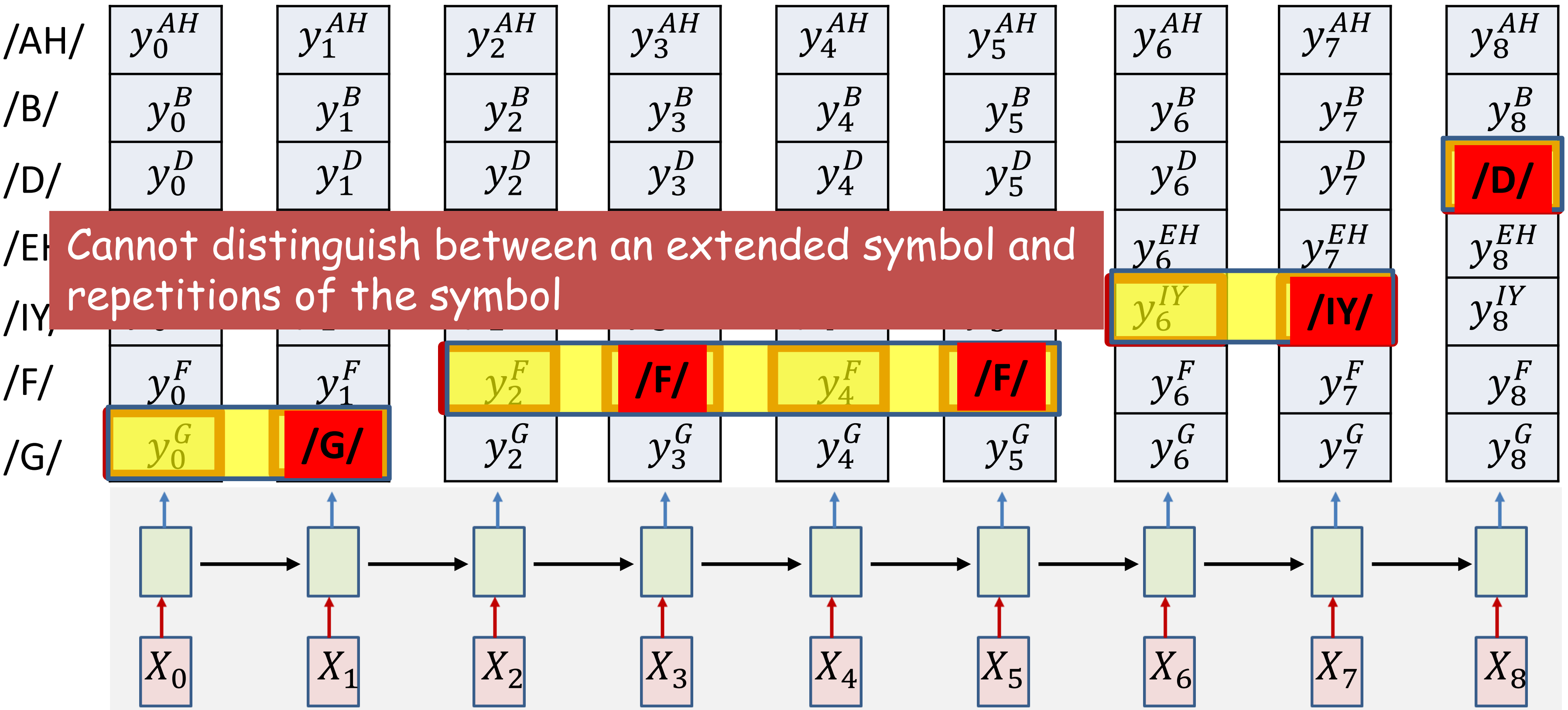
- Option 1: Simply select the most probable symbol at each time

Finding the best output



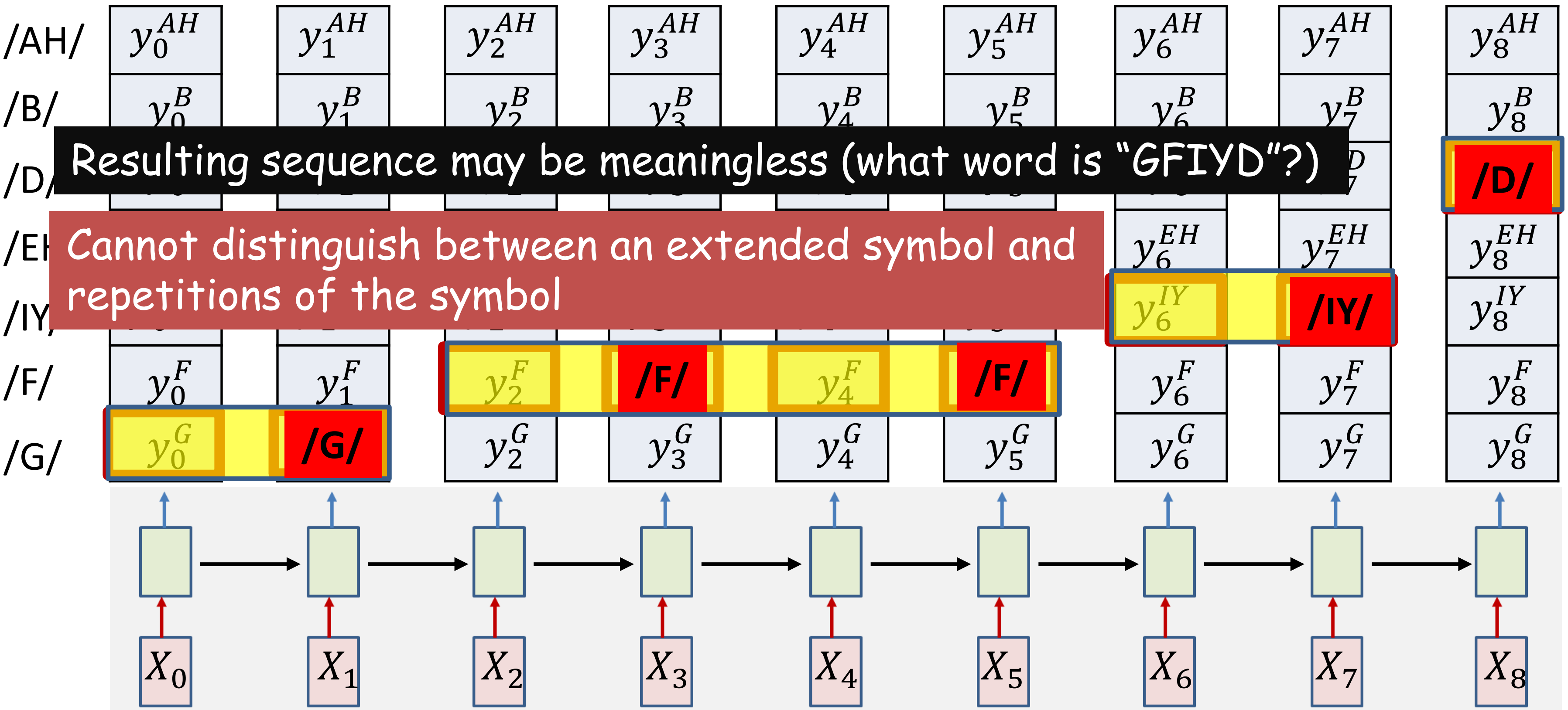
- Option 1: Simply select the most probable symbol at each time
  - *Merge* adjacent repeated symbols, and place the actual emission of the symbol in the final instant

The actual output of the network



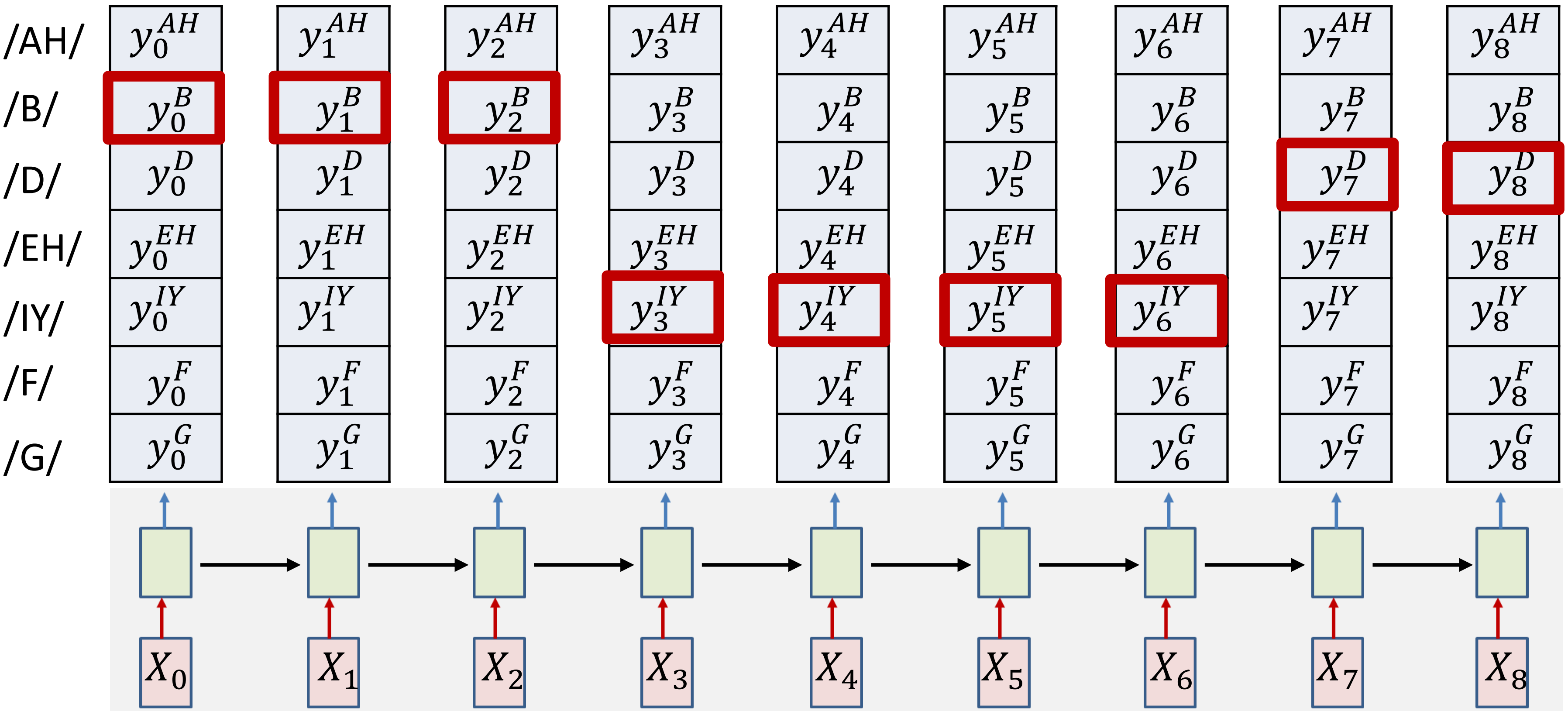
- Option 1: Simply select the most probable symbol at each time
  - *Merge* adjacent repeated symbols, and place the actual emission of the symbol in the final instant

The actual output of the network



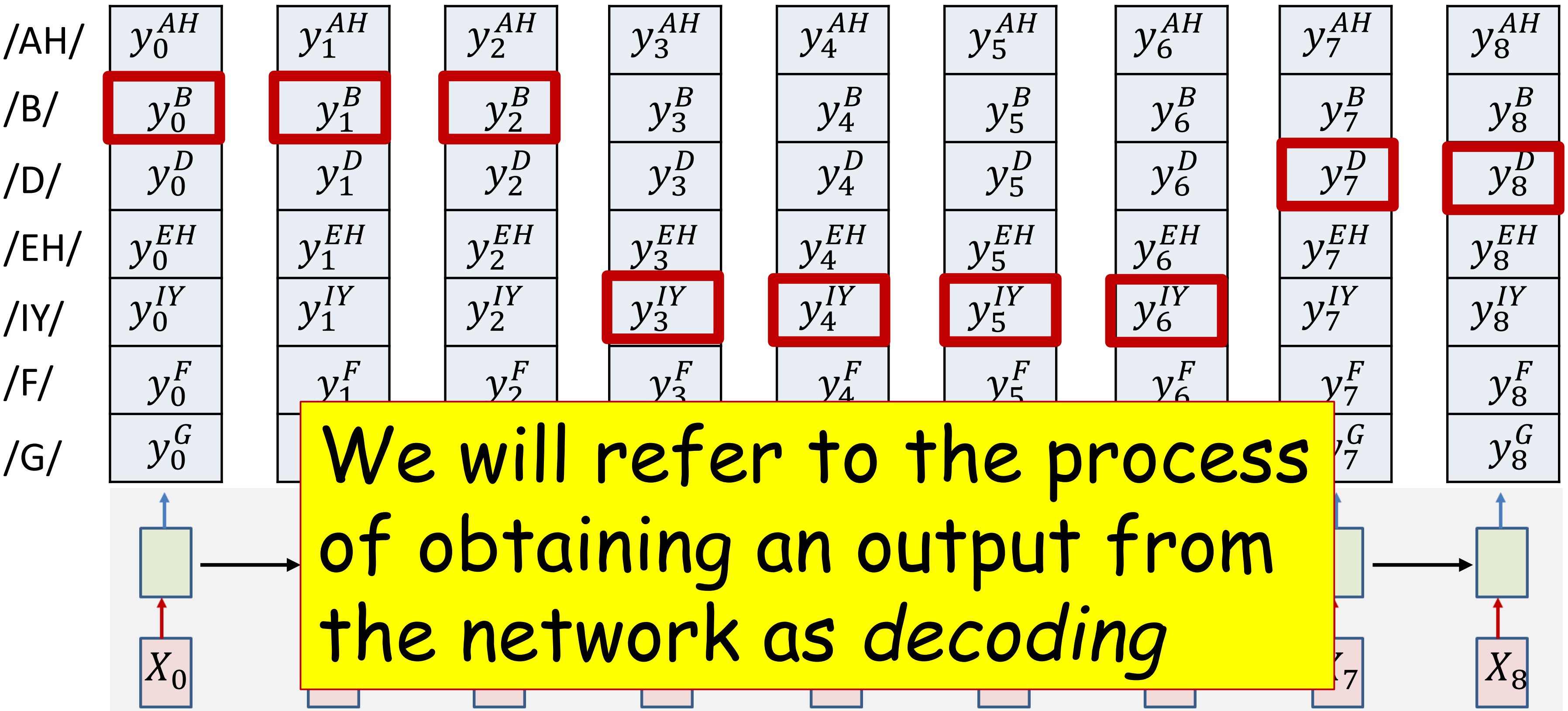
- Option 1: Simply select the most probable symbol at each time
  - *Merge* adjacent repeated symbols, and place the actual emission of the symbol in the final instant

The actual output of the network



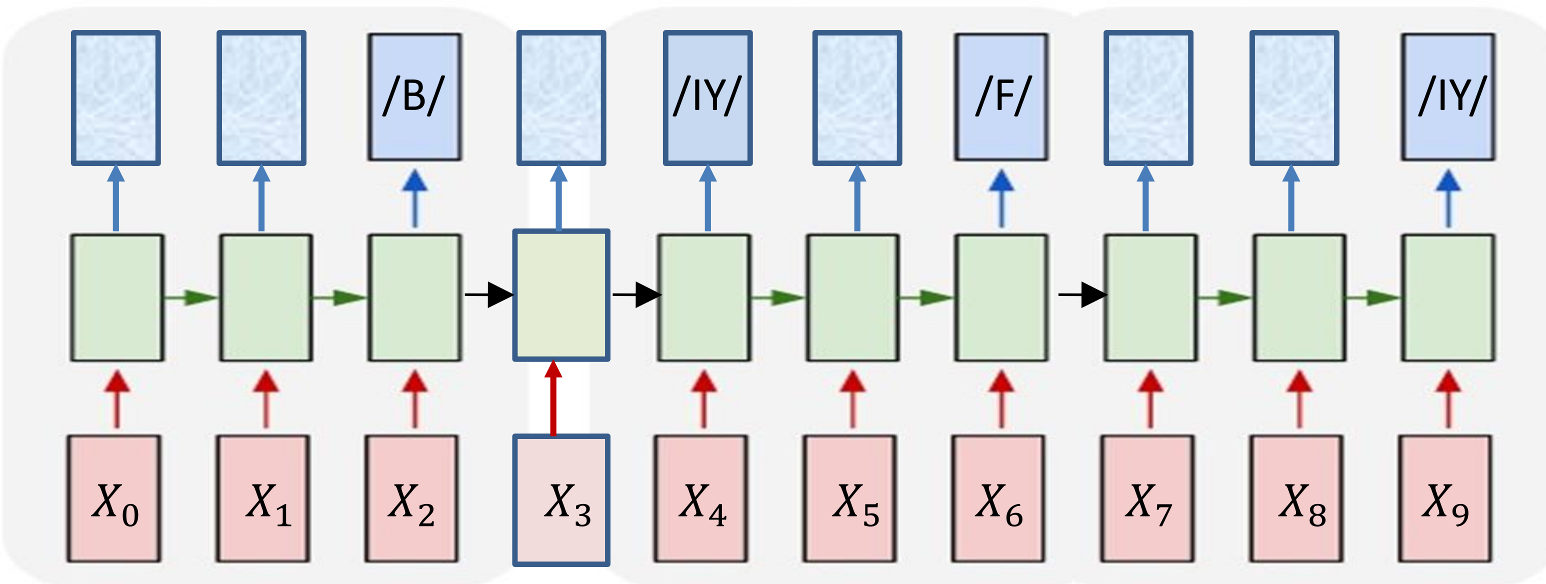
- Option 2: Impose external constraints on what sequences are allowed
  – *E.g.* only allow sequences corresponding to dictionary words
  – *E.g. Sub-symbol* units (like in HW1 – what were they?)

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?)

The *sequence-to-sequence* problem

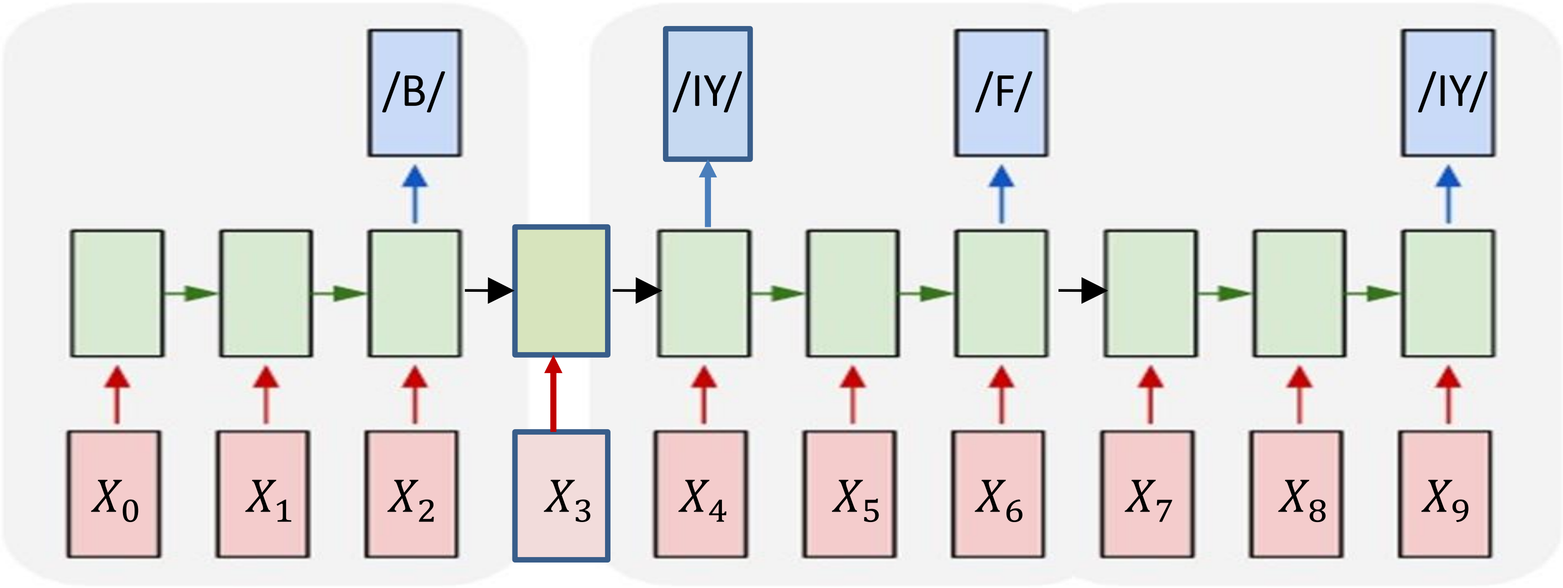


- How do we know *when* to output symbols
  - In fact, the network produces outputs at *every* time
  - *Which* of these are the *real* outputs

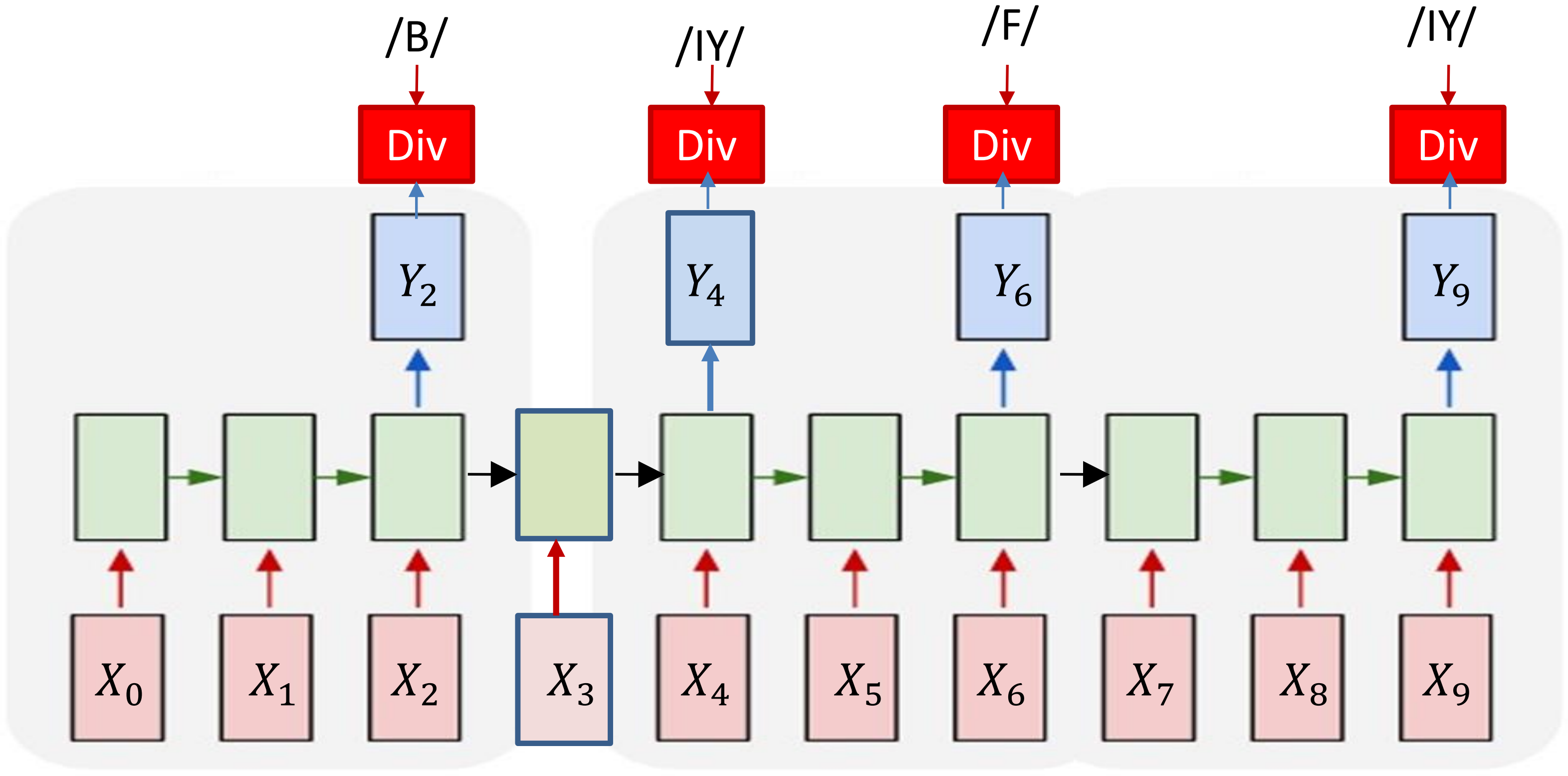
Addressed

- How do we *train* these models?

Training



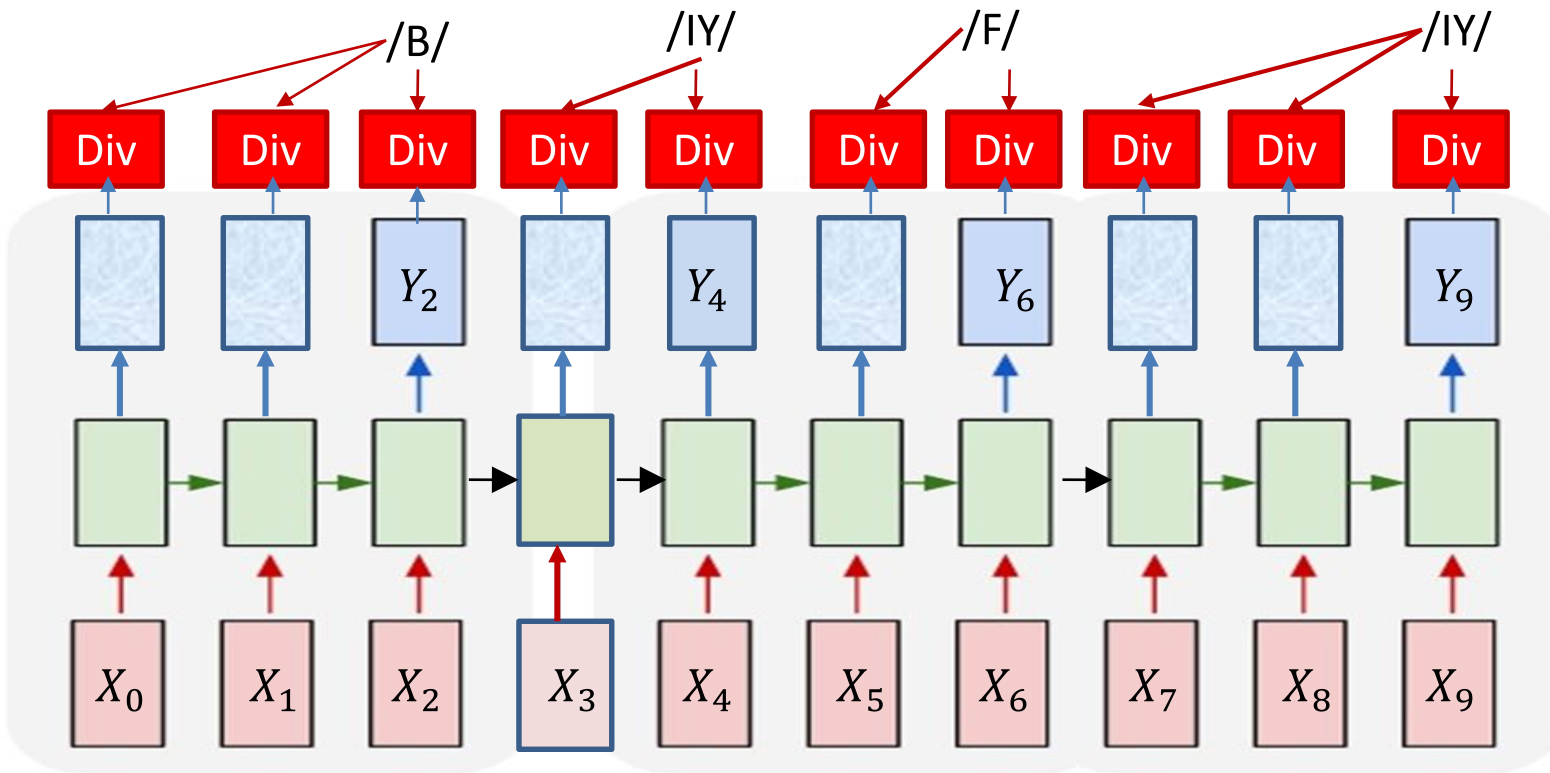
- Given output symbols *at the right locations*
  - The phoneme /B/ ends at X_2, /IY/ at X_4, /F/ at X_6, /IY/ at X_9



- Either just define Divergence as:

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

- Or..

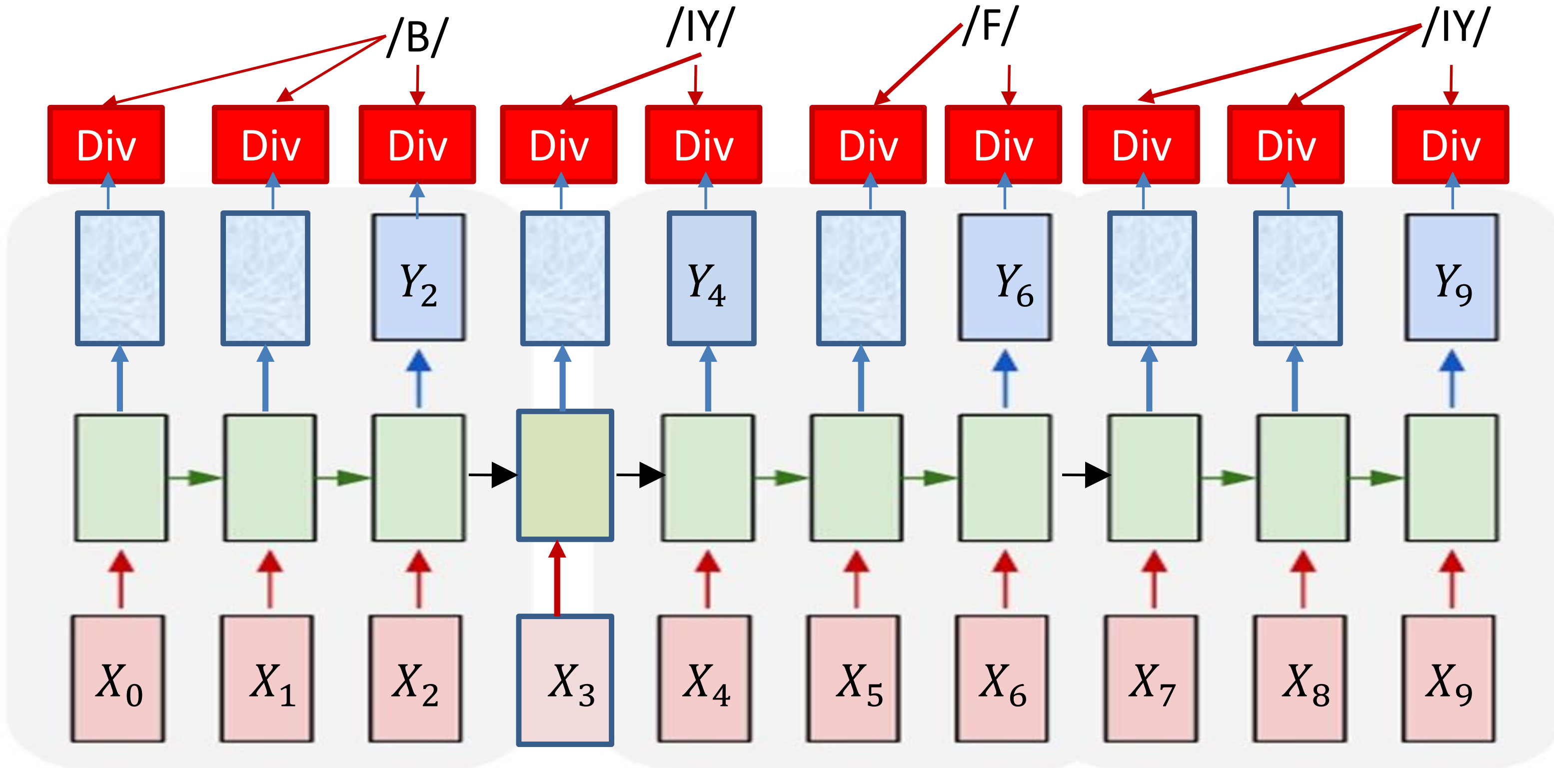


- Either just define Divergence as:

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

- Or *repeat the symbols over their duration*

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



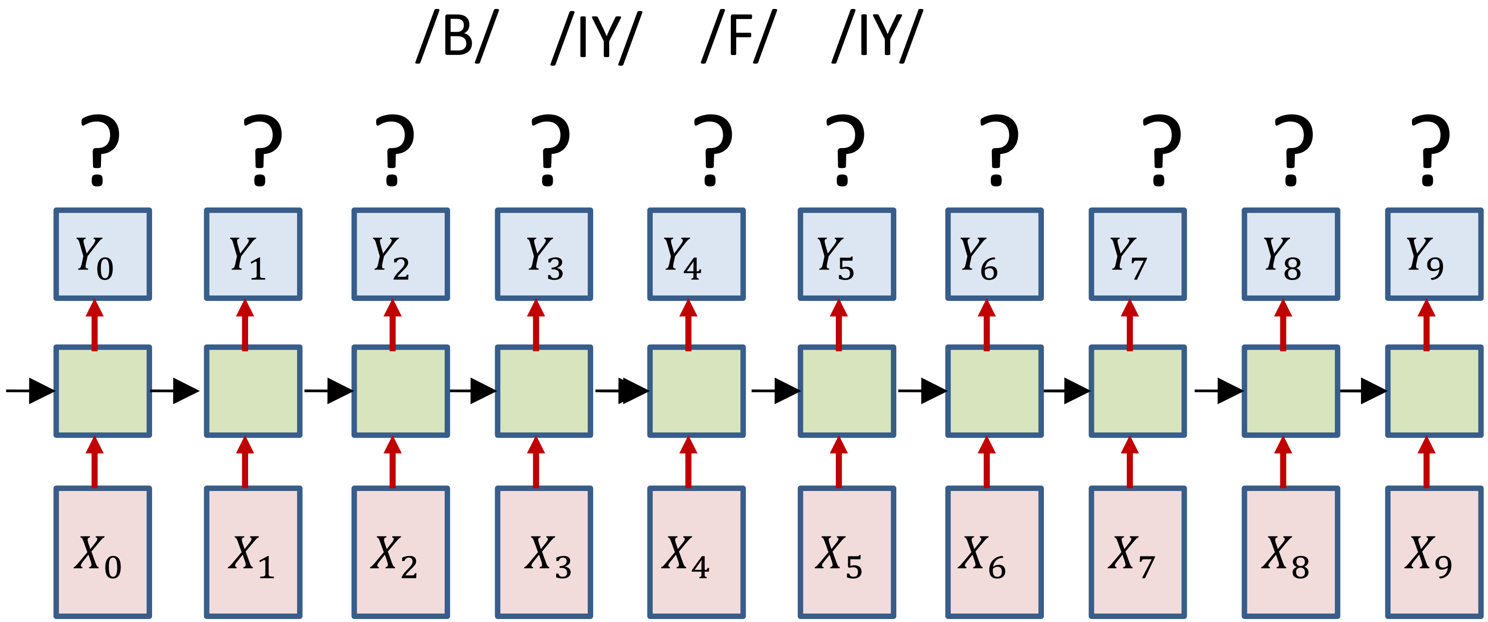
$$DIV = \sum_t Xent(Y_t, symbol_t) = -\sum_t \log Y(t, symbol_t)$$

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

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

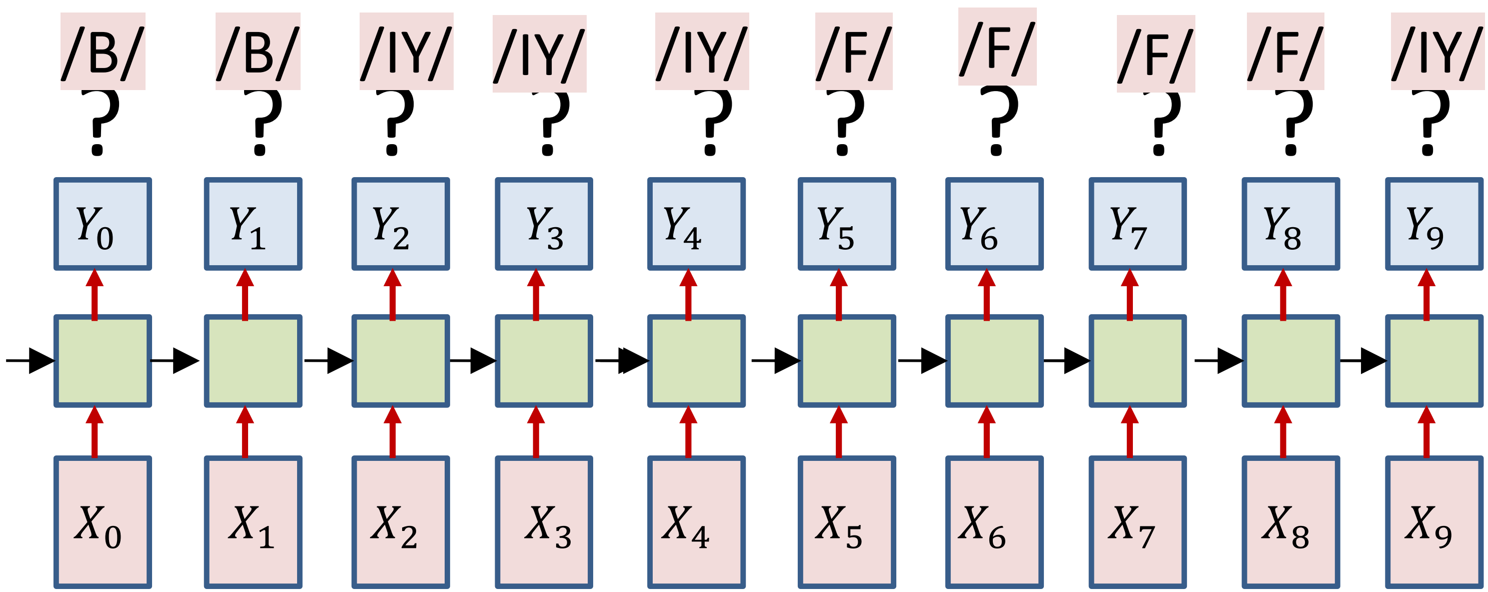
  – Zeros except at the component corresponding to the target

Problem: No timing information provided



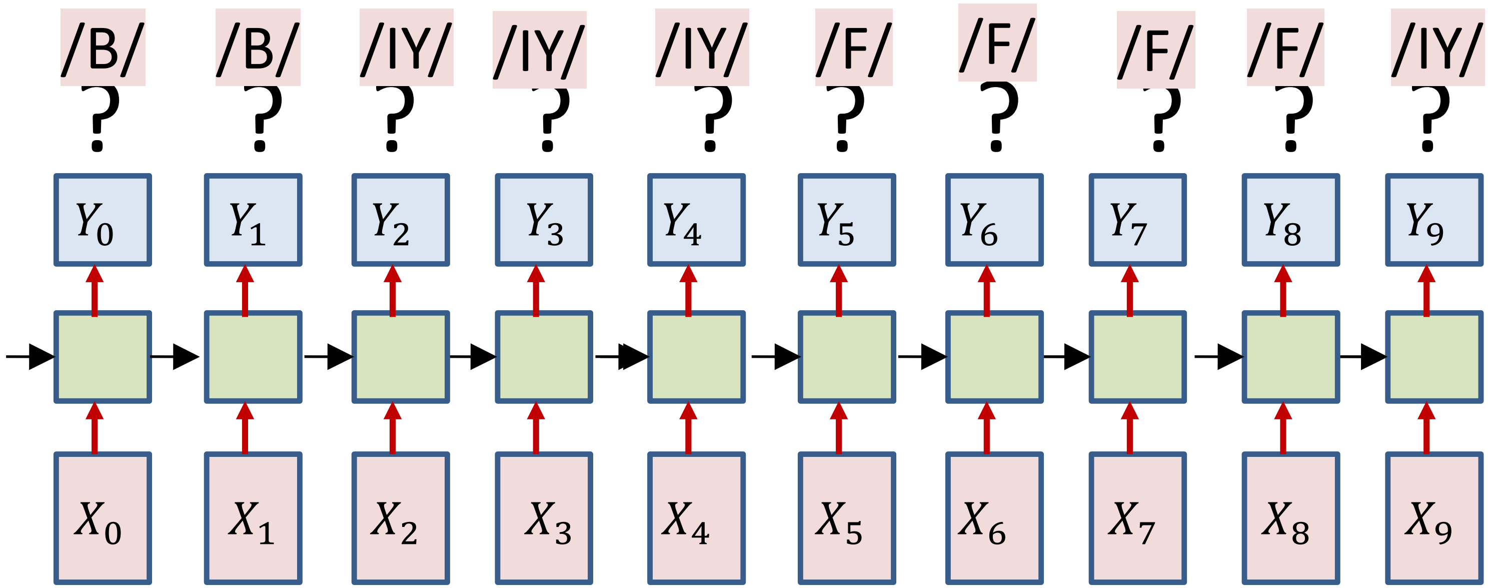
- Only the sequence of output symbols is provided for the training data
  - But no indication of which one occurs where
- How do we compute the divergence?
  - And how do we compute its gradient w.r.t. Y_t

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*



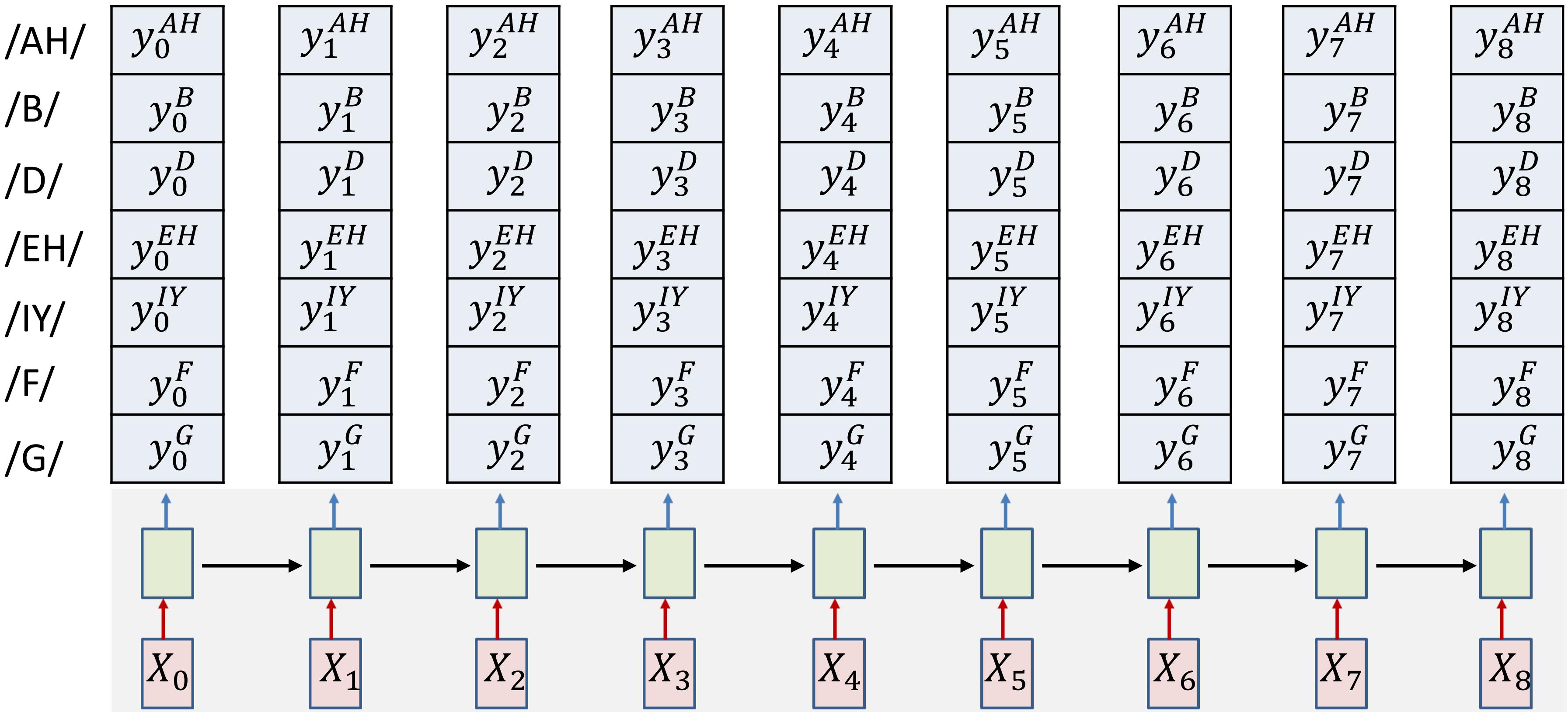
- Initialize: Assign an initial alignment
  - Either randomly, based on some heuristic, or any other rationale
- Iterate:
  - Train the network using the current alignment
  - *Reestimate* the alignment for each training instance
    - Using the decoding methods already discussed

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/ ..
  - $s_i = S_k \Rightarrow i \geq k$
  - $s_i = S_k,\ s_j = S_l,\quad i < j \Rightarrow k \leq l$
- 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}

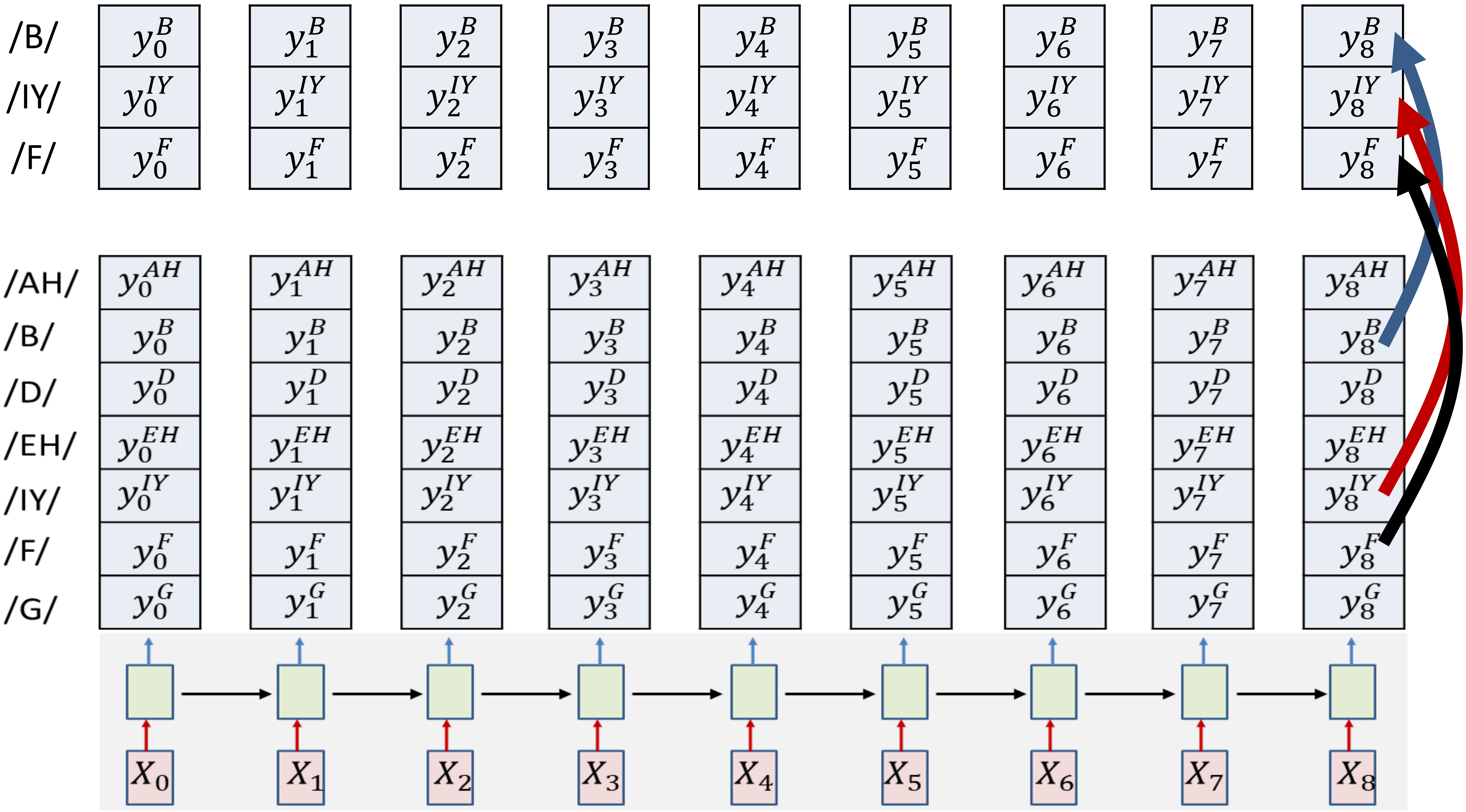
- We find the most likely sequence of symbols
  - (Conditioned on input $X_0 \ldots X_{N-1}$)
- This may not correspond to an expansion of the desired symbol sequence
  - E.g. the unconstrained decode may be /AH//AH//AH//D//D//AH//F//IY//IY/
    - Contracts to /AH/ /D/ /AH/ /F/ /IY/
  - Whereas we want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

Constraining the alignment: Try 1

[redacted]									
/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
[redacted]									
/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
[redacted]									

- 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

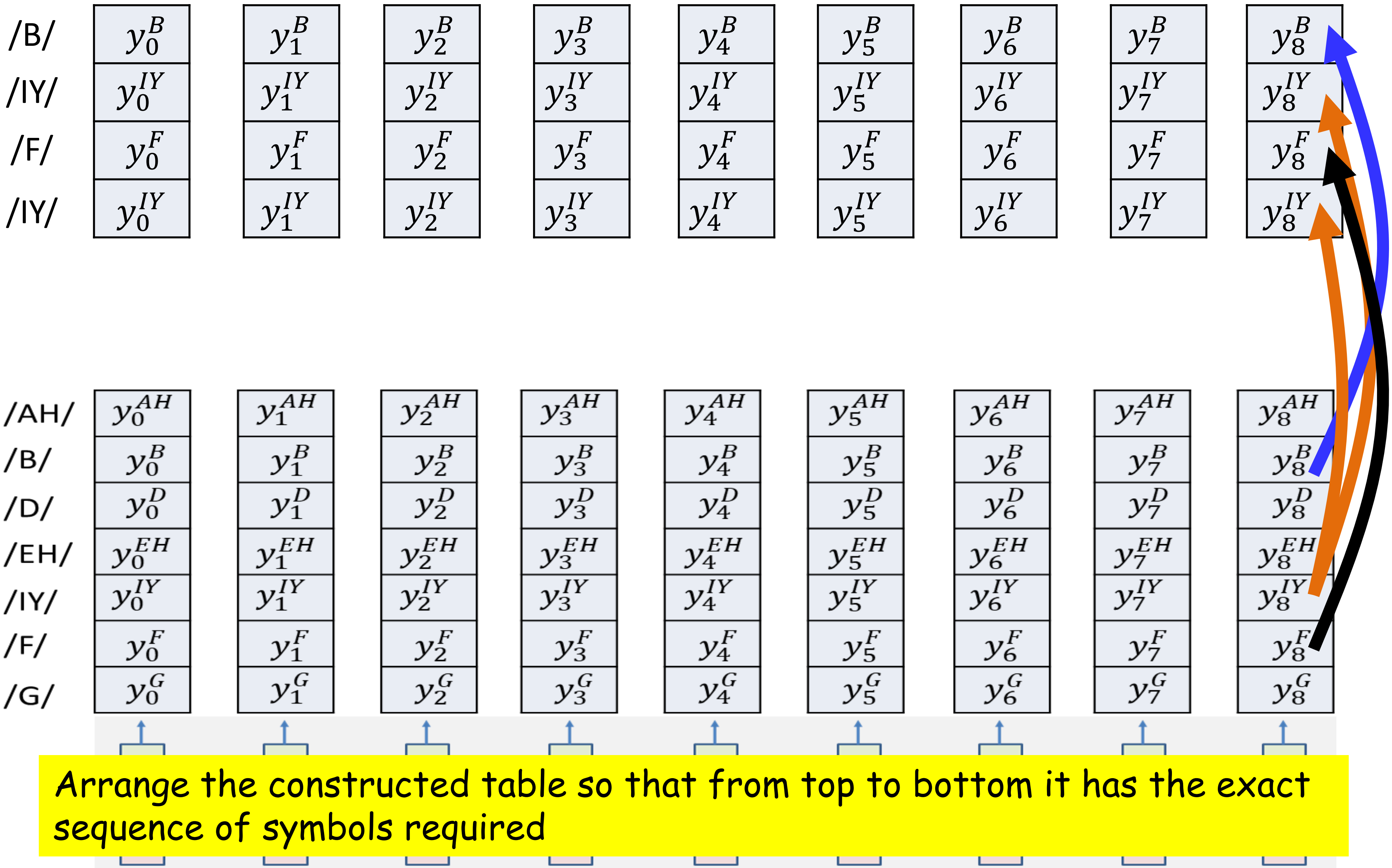
/B/	$\boxed{y_0^B}$	$\boxed{y_1^B}$	y_2^B	y_3^B	$\boxed{y_4^B}$	$\boxed{y_5^B}$	y_6^B	y_7^B	y_8^B
/IY/	y_0^{IY}	y_1^{IY}	$\boxed{y_2^{IY}}$	y_3^{IY}	y_4^{IY}	y_5^{IY}	y_6^{IY}	y_7^{IY}	$\boxed{y_8^{IY}}$
/F/	y_0^F	y_1^F	y_2^F	$\boxed{y_3^F}$	y_4^F	y_5^F	$\boxed{y_6^F}$	$\boxed{y_7^F}$	y_8^F

- 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



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

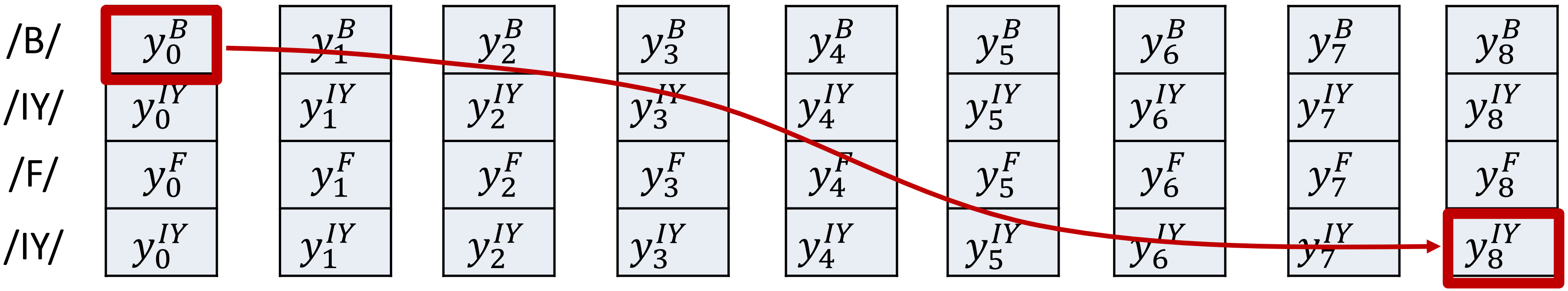
/B/	y_0^B	y_1^B	y_2^B	y_3^B	y_4^B	y_5^B	y_6^B	y_7^B	y_8^B
/IY/	y_0^{IY}	y_1^{IY}	y_2^{IY}	y_3^{IY}	y_4^{IY}	y_5^{IY}	y_6^{IY}	y_7^{IY}	y_8^{IY}
/F/	y_0^F	y_1^F	y_2^F	y_3^F	y_4^F	y_5^F	y_6^F	y_7^F	y_8^F
/IY/	y_0^{IY}	y_1^{IY}	y_2^{IY}	y_3^{IY}	y_4^{IY}	y_5^{IY}	y_6^{IY}	y_7^{IY}	y_8^{IY}

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

/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

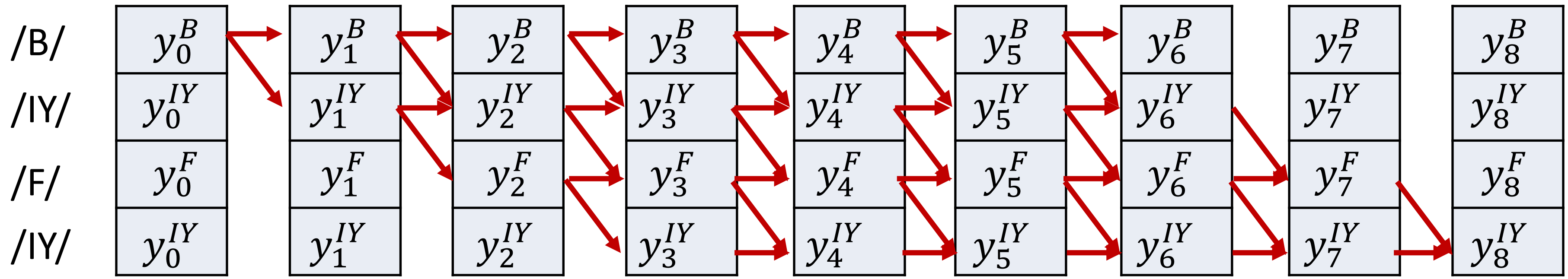
Arrange the constructed table so that from top to bottom it has the exact sequence of symbols required

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. never goes up
- This guarantees that the sequence *is* an expansion of the target sequence
  - /B/ /IY/ /F/ /IY/ in this case

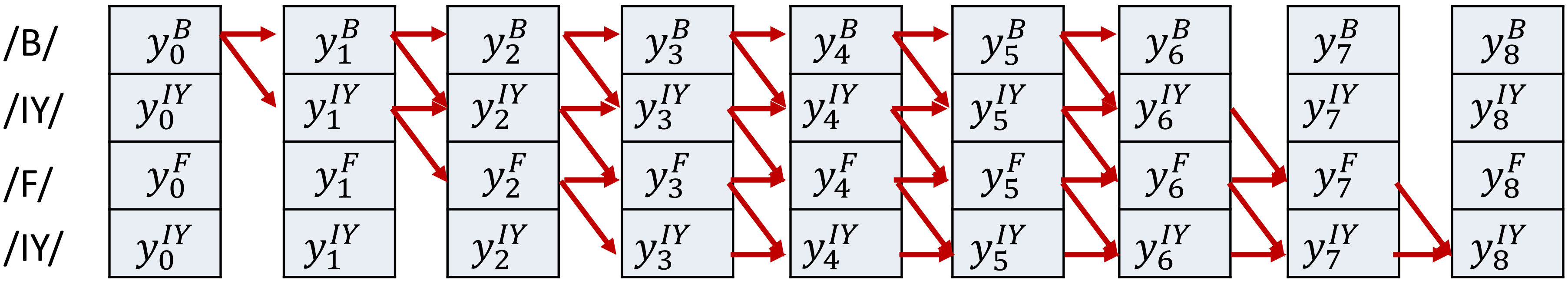
Explicitly constrain alignment



- 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

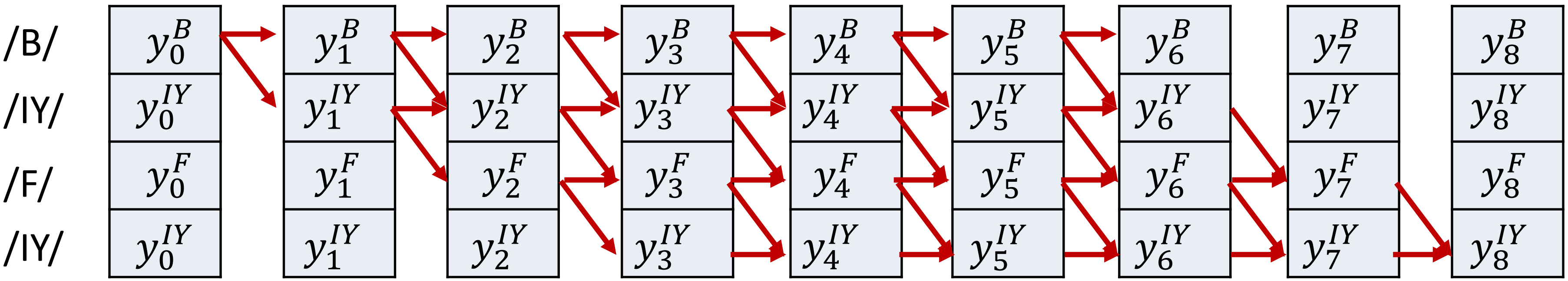
- Find the most probable path from source to sink using any dynamic programming algorithm
  - E.g. The Viterbi algorithm

Viterbi algorithm



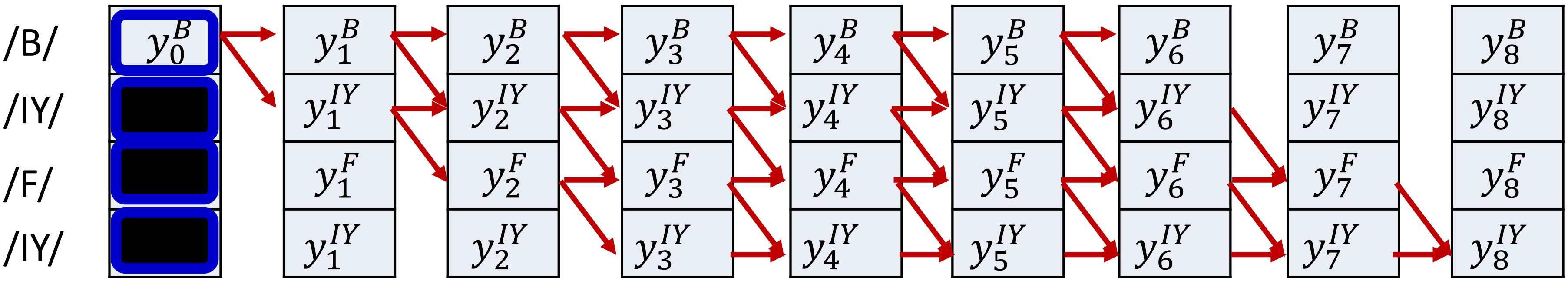
- At each node, keep track of
  - The best incoming edge
  - The score of the best path from the source to the node
- Dynamically compute the best path from source to sink

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

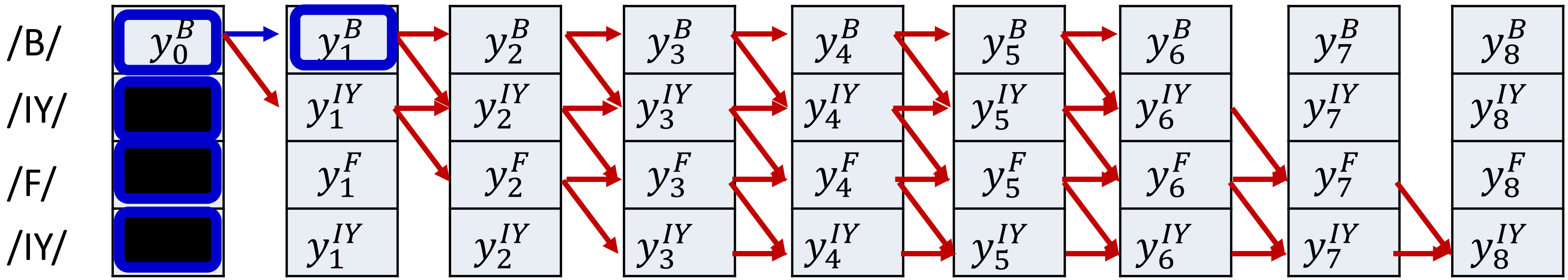


- Initialization:

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

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

Viterbi algorithm



- Initialization:

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

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

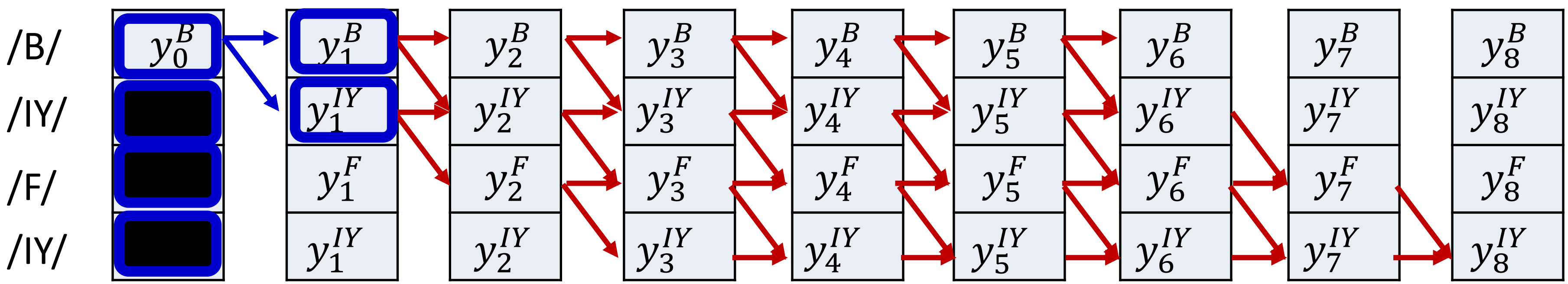
- for $t \ = \ 1 \dots T-1$

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

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

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

Viterbi algorithm



- Initialization:

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

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

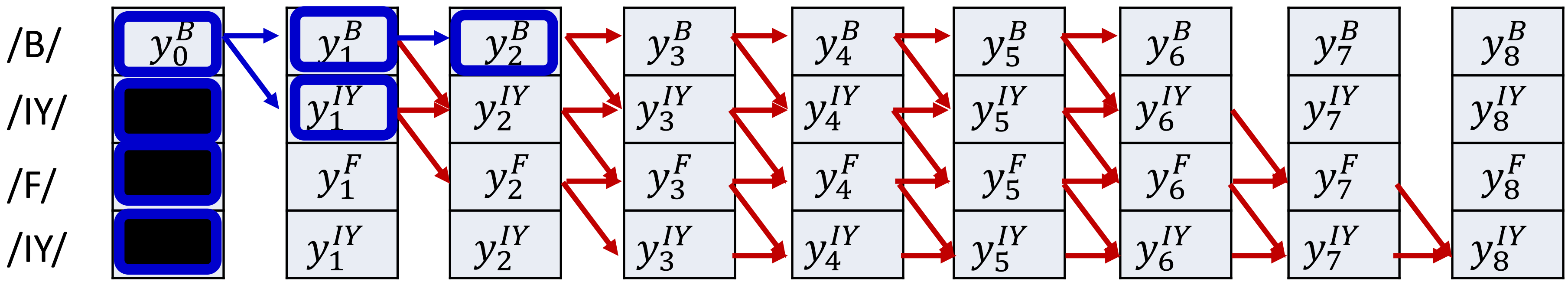
- for $t = 1 \ldots T-1$

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

for $l = 1 \ldots K-1$

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

Viterbi algorithm



- Initialization:

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

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

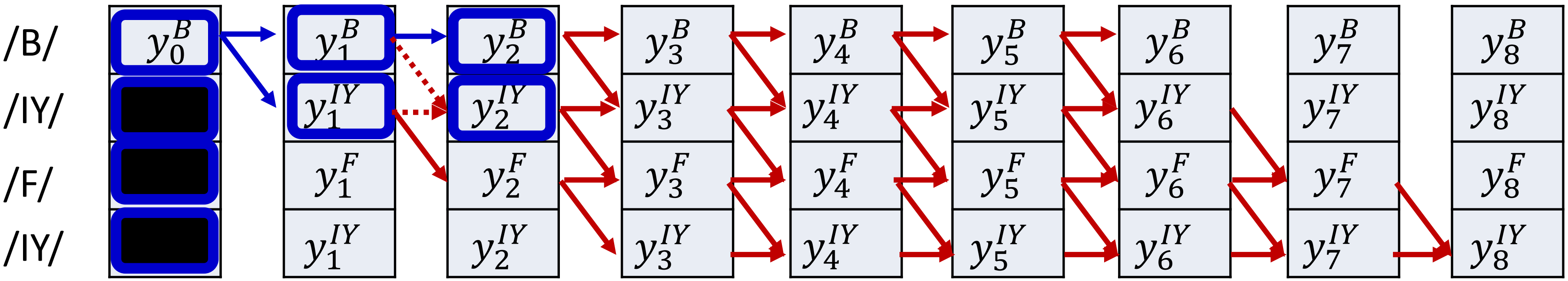
- for $t = 1 \ldots T-1$

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

for $l = 1 \ldots K-1$

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

Viterbi algorithm



- Initialization:

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

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

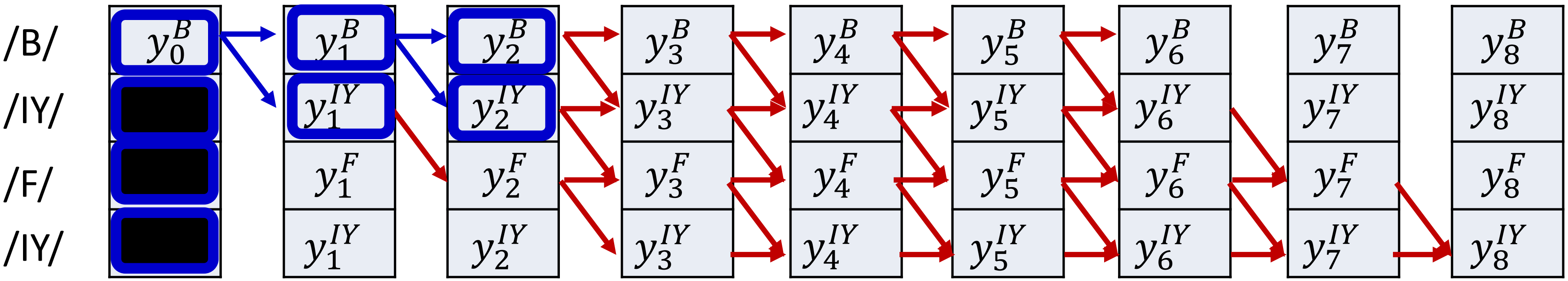
- for $t = 1 \dots T-1$

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

for $l = 1 \dots K-1$

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

Viterbi algorithm



- Initialization:

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

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

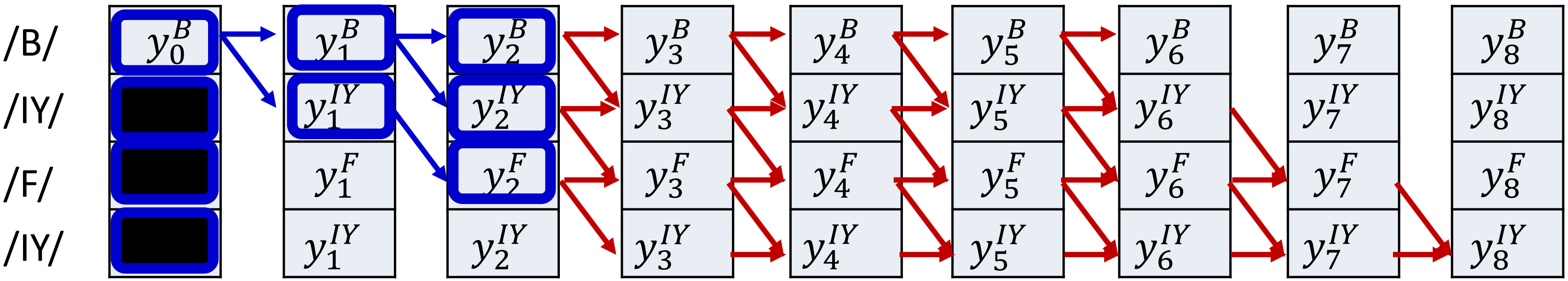
- for $t \ = \ 1 \dots T-1$

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

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

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

Viterbi algorithm



- Initialization:

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

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

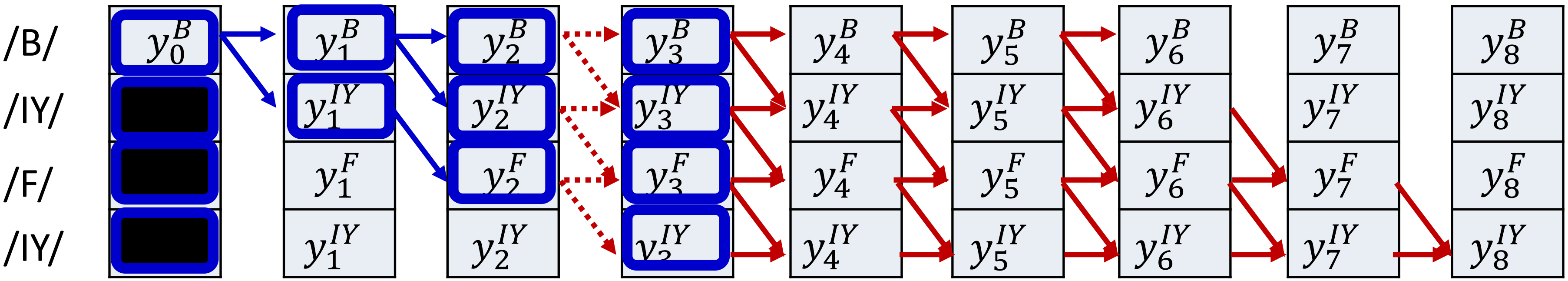
- for $t = 1 \ldots T-1$

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

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:

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

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

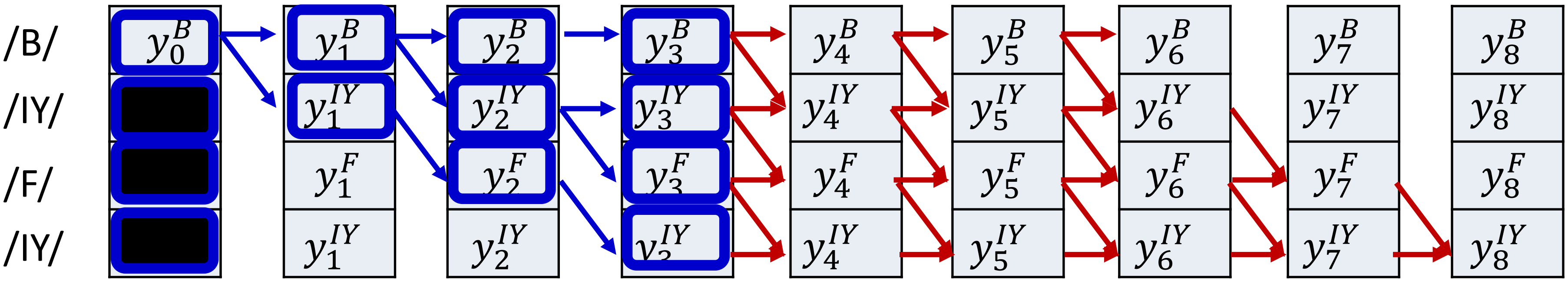
- for $t = 1 \ldots T-1$

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

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:

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

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

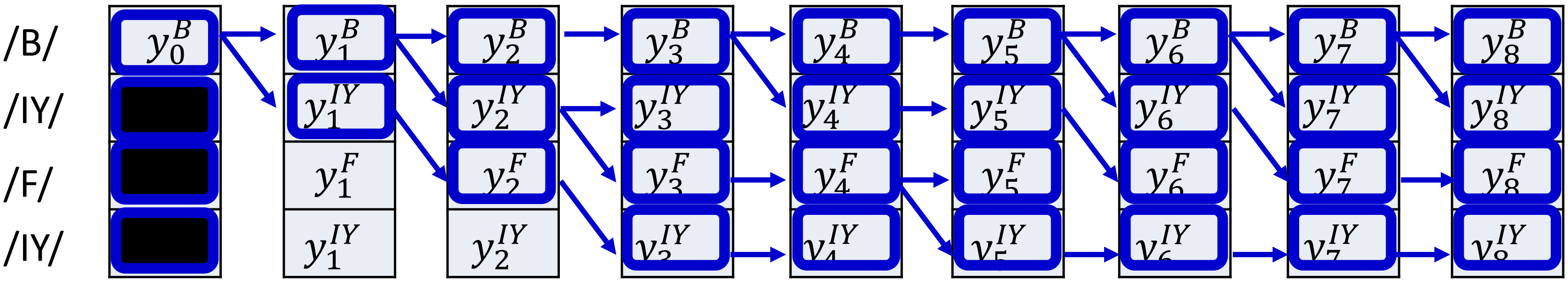
- for $t = 1 \ldots T-1$

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

for $l = 1 \ldots K-1$

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

Viterbi algorithm



- Initialization:

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

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

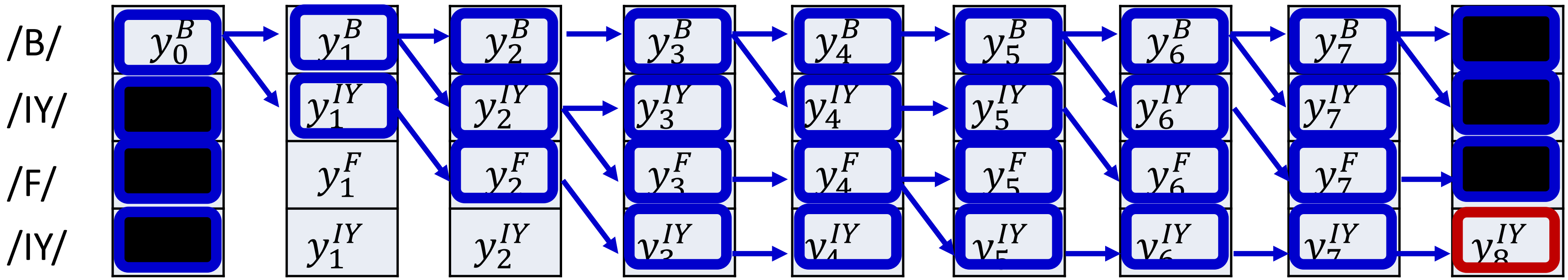
- for $t \ = \ 1 \ldots T - 1$

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

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

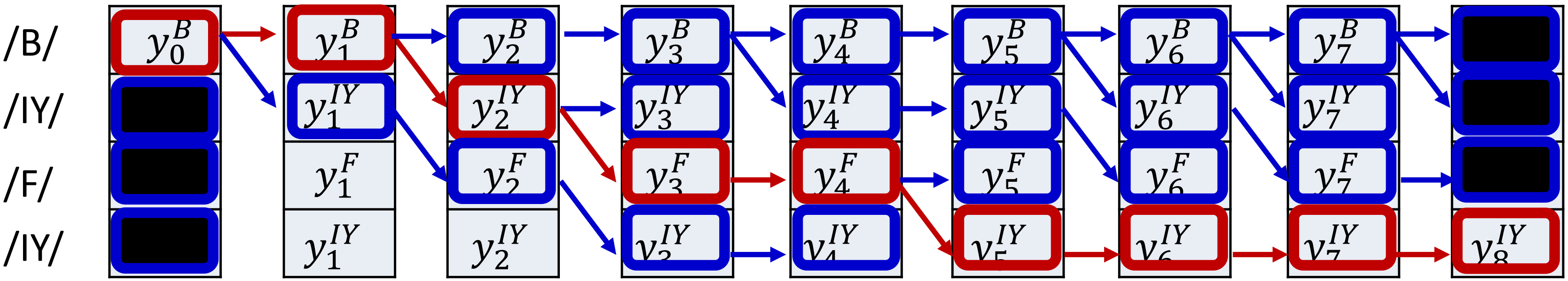
  - $BP(t, l) = \big(if \ \big(Bscr(t-1, l-1) > \ Bscr(t-1, l)\big) \ l - 1; \ \ else \ l\big)$
  - $Bscr(t, l) \ = \ Bscr(BP(t, l)) \ \times y_t^{S(l)}$

Viterbi algorithm



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

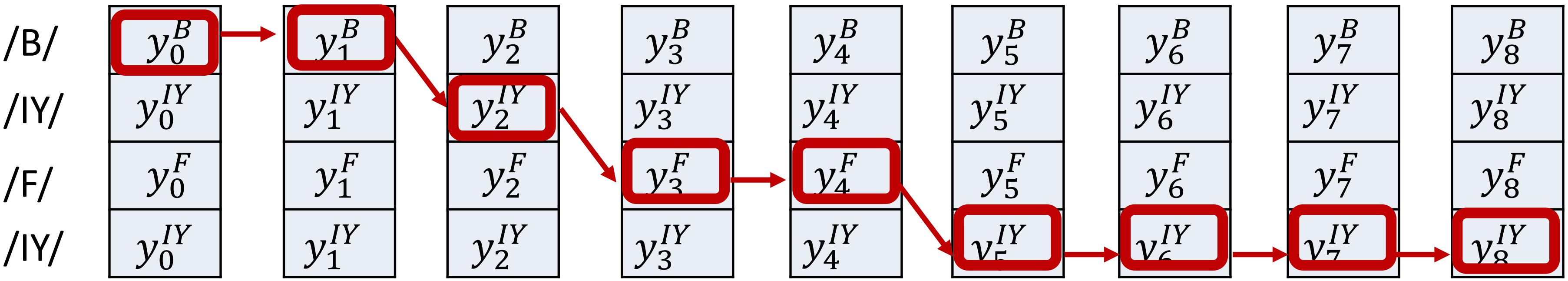
Viterbi algorithm



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

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

Viterbi algorithm

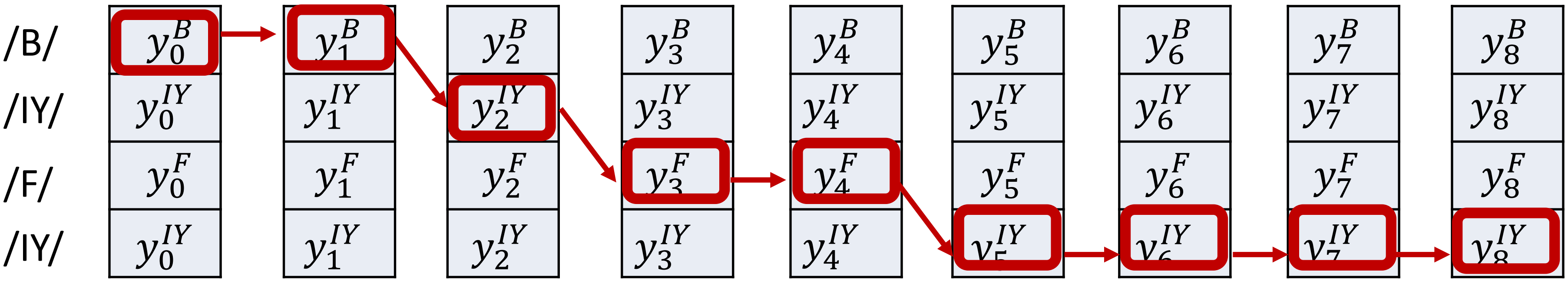


- $s(T-1) = S(K-1)$
- for $t = T-1 \; downto \; 1$

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

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

Gradients from the alignment



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

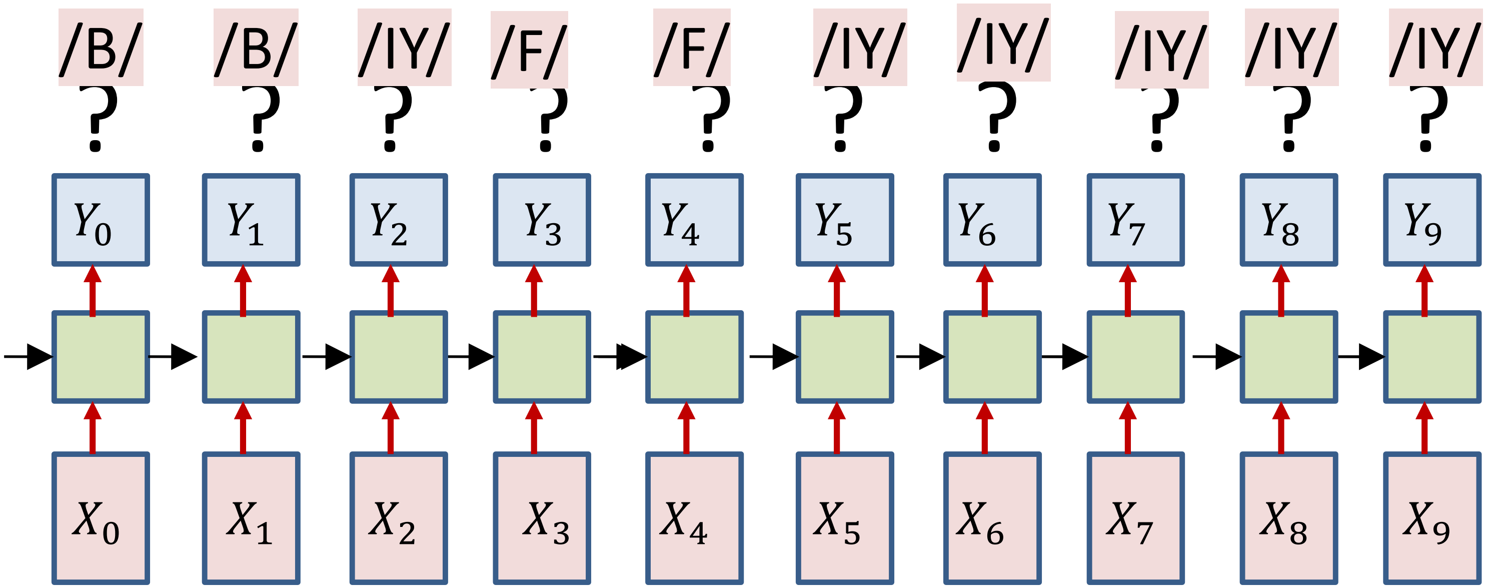
$$DIV = \sum_t Xent\left(Y_t, symbol_t^{bestpath}\right) = -\sum_t \log Y\left(t, symbol_t^{bestpath}\right)$$

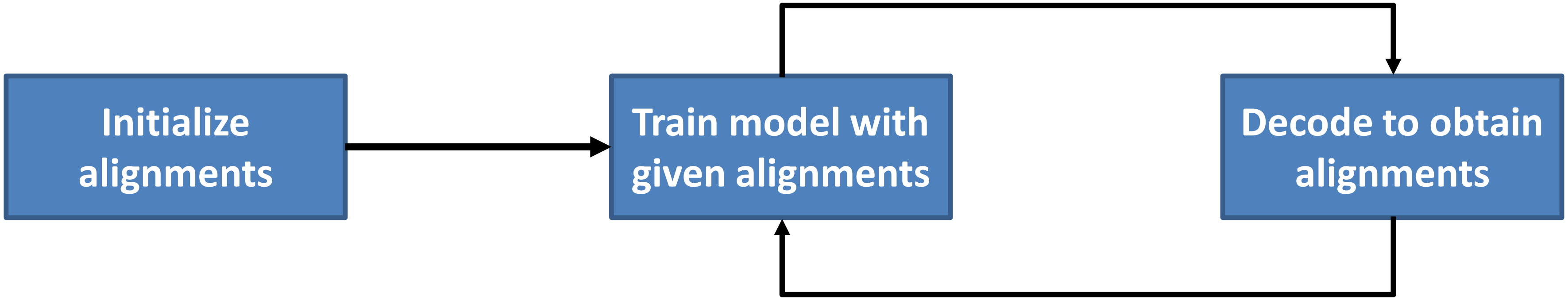
- The gradient w.r.t the t-th output vector Y_t

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

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

Iterative Estimate and Training





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

Iterative update

- 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

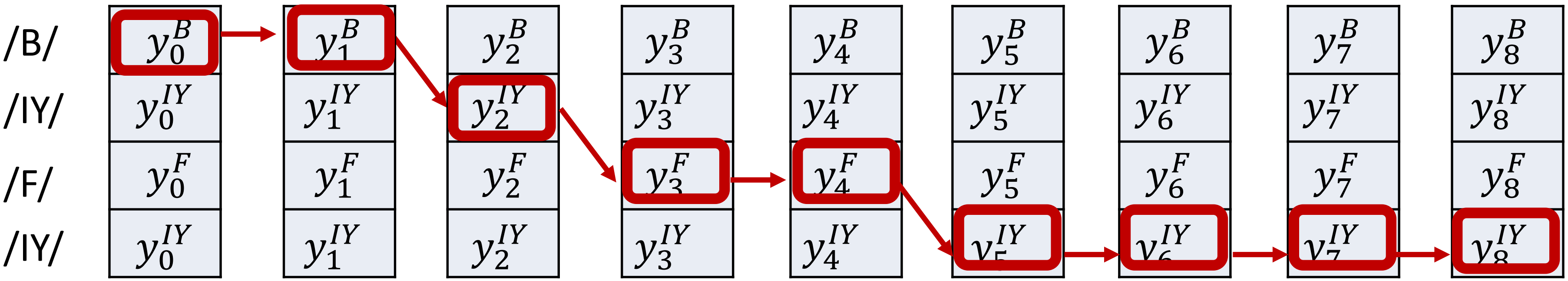
Iterative update: Problem

- 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

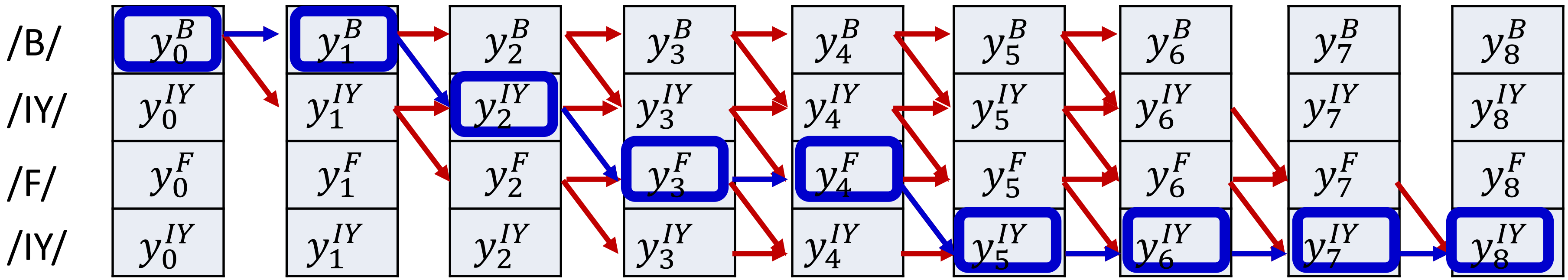


- We *commit* to the single "best" estimated alignment
  - The *most likely* alignment

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

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

The reason for suboptimality



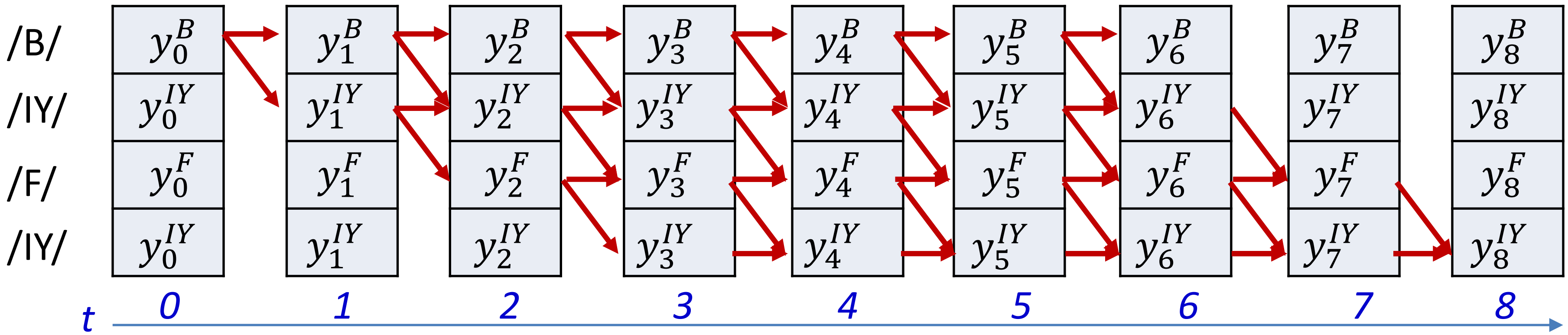
- We *commit* to the single “best” estimated alignment
  - The *most likely* alignment

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

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

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

Averaging over *all* alignments

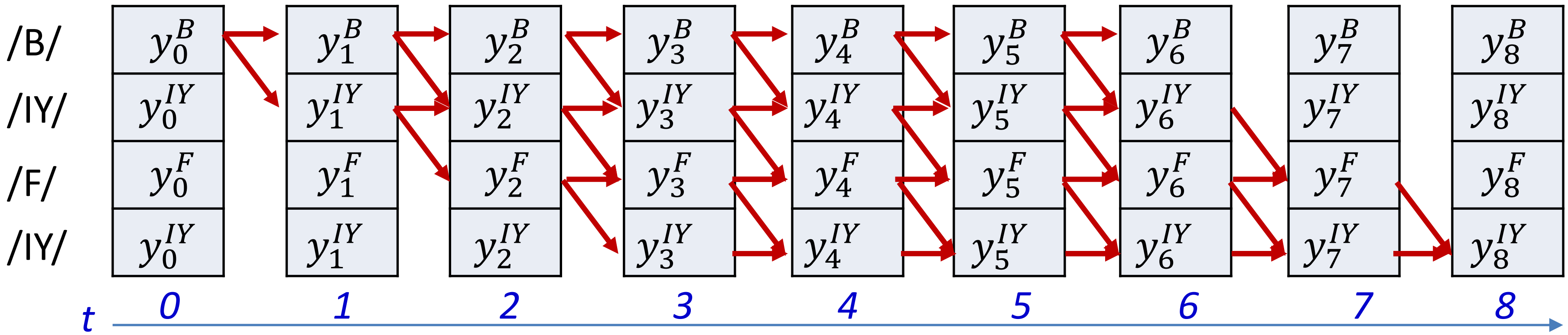


- Instead of only selecting the most likely alignment, use the statistical expectation over *all* possible alignments

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

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

The expectation over *all* alignments



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

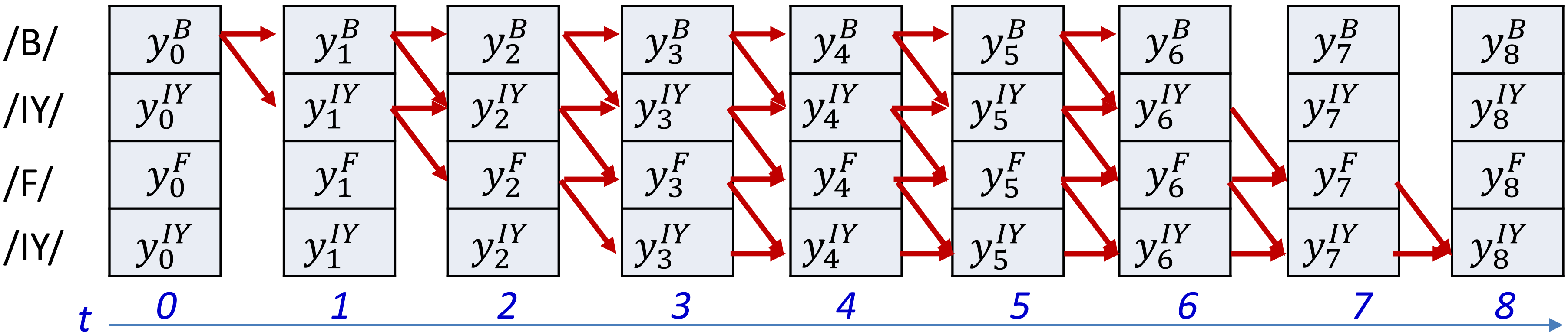
- Using the linearity of expectation

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

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

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

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

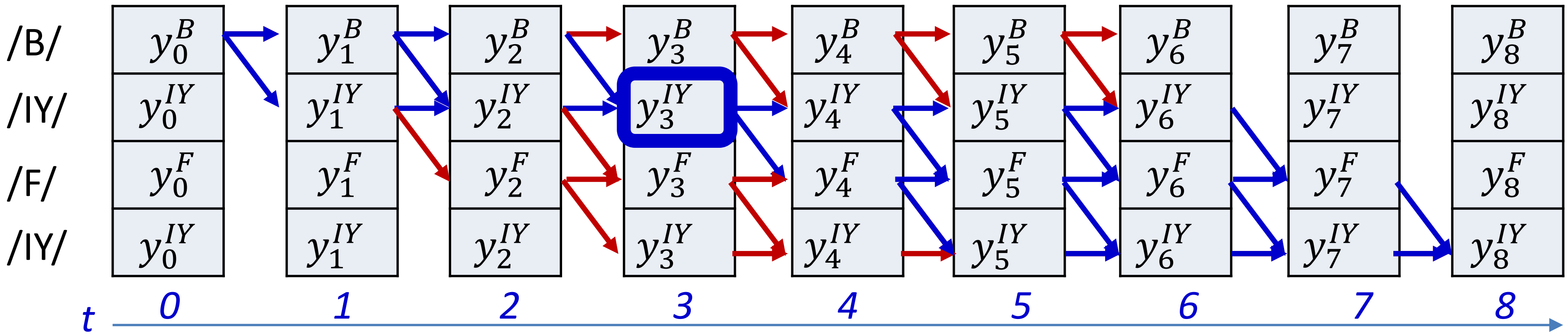
•

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

– 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

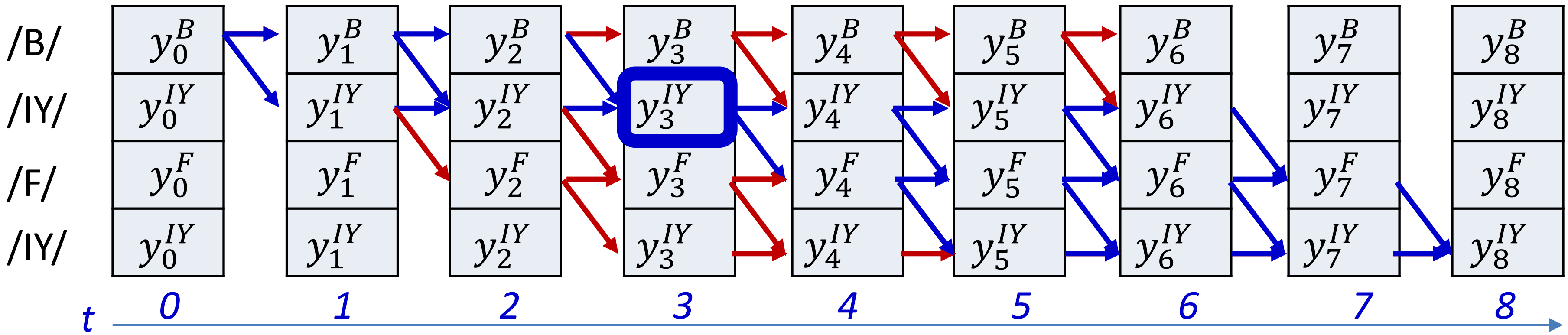


$$P(s_t = S_r|\mathbf{S}, \mathbf{X}) \propto P(s_t = S_r, \mathbf{S}|\mathbf{X})$$

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



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

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

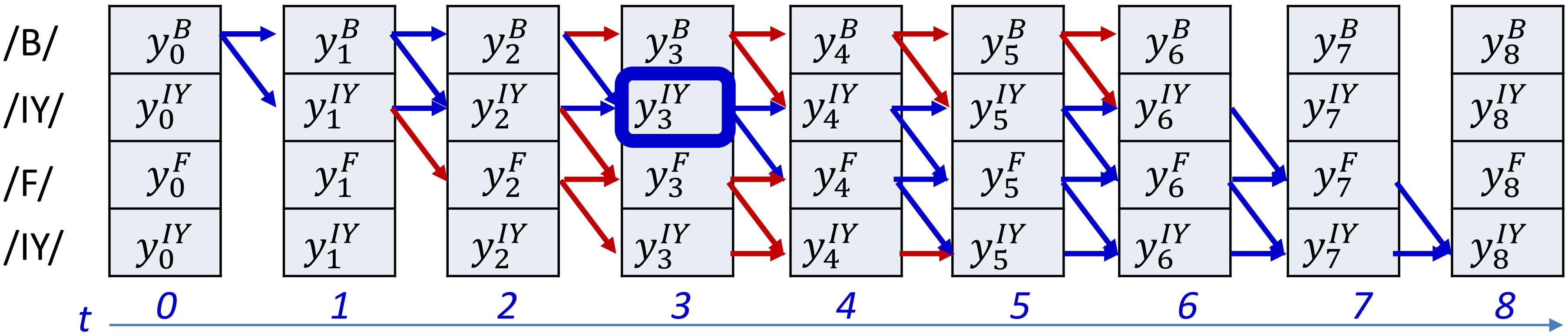
$$= \sum_{s_0 \dots s_{t-1} \rightarrow S_1 \dots [S_{r-}]} \quad \sum_{s_{t+1} \dots s_{N-1} \rightarrow [S_{r+}] \dots S_K} P(s_0 \dots s_{t-1}, s_t = S_r, s_{t+1} \dots s_{N-1}, \mathbf{S}|\mathbf{X})$$

- $[S_{r+}]$ indicates that s_{t+1} might either be S_r or S_{r+1}
- $[S_{r-}]$ indicates that s_{t-1} might be either S_r or S_{r-1}

$$= \sum_{s_0 \dots s_{t-1} \rightarrow S_1 \dots [S_{r-}]} \quad \sum_{s_{t+1} \dots s_{N-1} \rightarrow [S_{r+}] \dots S_K} P(s_0 \dots s_{t-1}, s_t = S_r, s_{t+1} \dots s_{N-1} |\mathbf{X})$$

  - Because the target symbol sequence $\mathbf{S}$ is implicit in the synchronized sequences $s_0 \dots s_{N-1}$ which are constrained to be expansions of $\mathbf{S}$

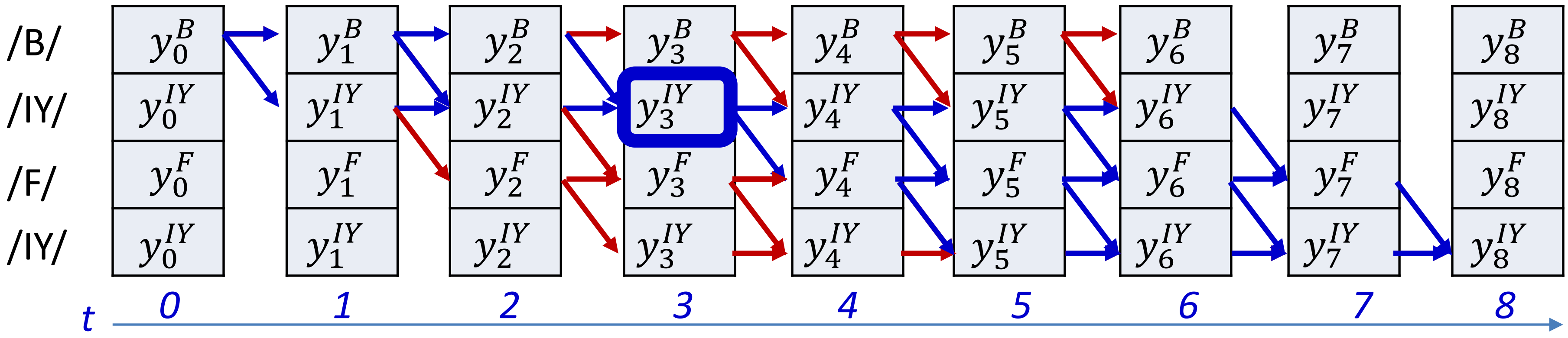
A posteriori probabilities of symbols



$$P(s_t = S_r, \mathbf{S}|\mathbf{X}) = \sum_{s_0 \dots s_{t-1} \to S_1 \dots [S_{r-}]} \; \sum_{s_{t+1} \dots s_{N-1} \to [S_{r+}] \dots S_K} P(s_0 \dots s_{t-1}, s_t = S_r, s_{t+1} \dots s_{N-1} \,|\mathbf{X})$$

$$= \sum_{s_0 \dots s_{t-1} \to S_1 \dots [S_{r-}]} \; \sum_{s_{t+1} \dots s_{N-1} \to [S_{r+}] \dots S_K} P(s_0 \dots s_{t-1}, s_t = S_r \,|\mathbf{X}) P(s_{t+1} \dots s_{N-1} \,|s_0 \dots s_{t-1}, s_t = S_r, \mathbf{X})$$

A posteriori probabilities of symbols



$$P(s_t = S_r, \mathbf{S}|\mathbf{X}) = \sum_{s_0 \ldots s_{t-1} \rightarrow S_1 \ldots [S_{r-}]} \sum_{s_{t+1} \ldots s_{N-1} \rightarrow [S_{r+}] \ldots S_K} P(s_0 \ldots s_{t-1}, s_t = S_r, s_{t+1} \ldots s_{N-1} |\mathbf{X})$$

$$= \sum_{s_0 \ldots s_{t-1} \rightarrow S_1 \ldots [S_{r-}]} \sum_{s_{t+1} \ldots s_{N-1} \rightarrow [S_{r+}] \ldots S_K} P(s_0 \ldots s_{t-1}, s_t = S_r |\mathbf{X}) P(s_{t+1} \ldots s_{N-1} | s_0 \ldots s_{t-1}, s_t = S_r, \mathbf{X})$$

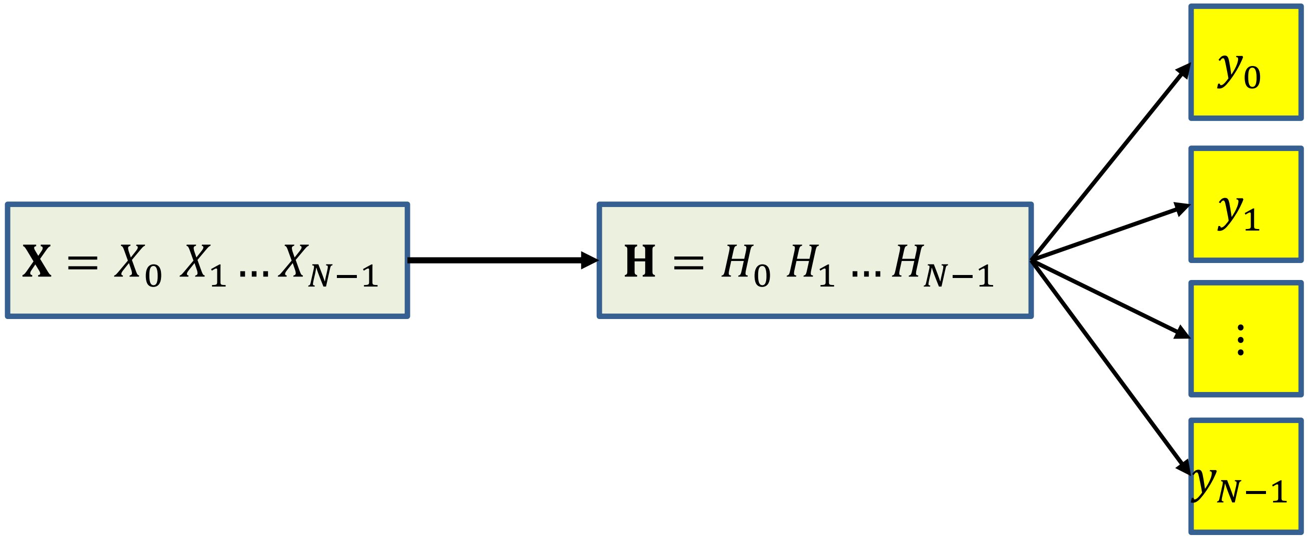
- For a recurrent network without feedback from the output we can make the conditional independence assumption:

$$P(s_{t+1} \ldots | s_0 \ldots s_t, \mathbf{X}) = P(s_{t+1} \ldots |\mathbf{X})$$

$$\boxed{P(s_t = S_r, \mathbf{S}|\mathbf{X}) = \sum_{s_0 \ldots s_{t-1} \rightarrow S_1 \ldots [S_{r-}]} \sum_{s_{t+1} \ldots s_{N-1} \rightarrow [S_{r+}] \ldots S_K} P(s_0 \ldots s_{t-1}, s_t = S_r |\mathbf{X}) P(s_{t+1} \ldots s_{N-1} | s_t = S_r, \mathbf{X})}$$

Note: in reality, this assumption is not valid if the hidden states are unknown, but we will make it anyway

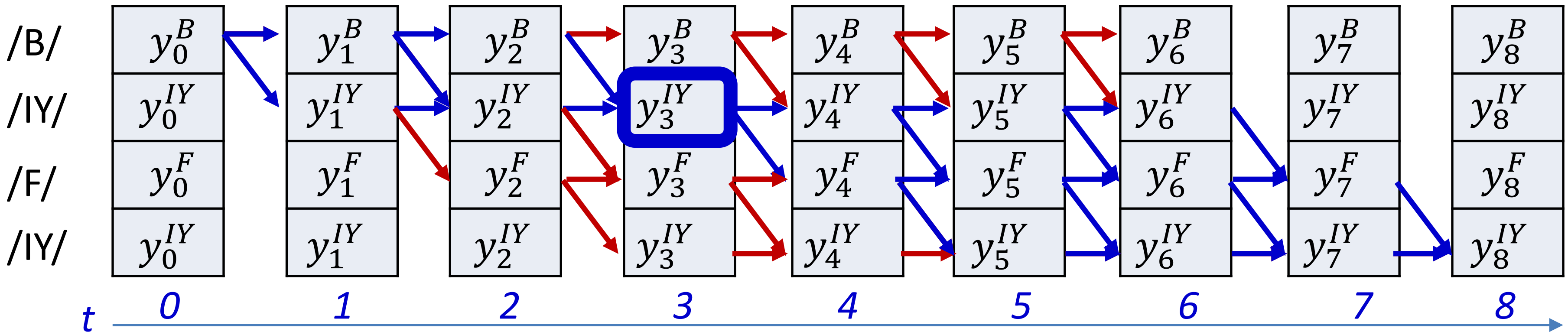
Conditional independence



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

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

A posteriori probabilities of symbols

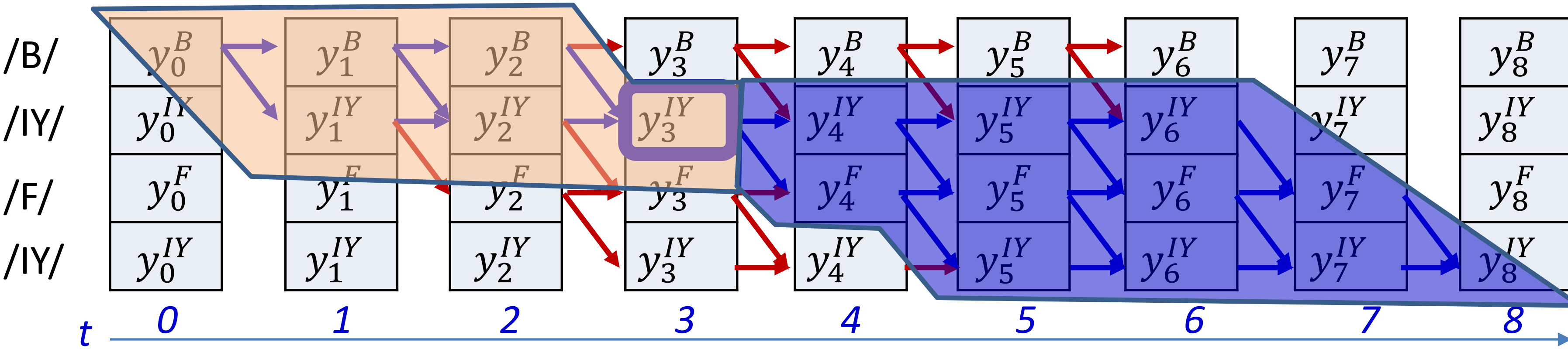


$$P(s_t = S_r, \mathbf{S}|\mathbf{X})$$

$$= \sum_{s_0 \ldots s_{t-1} \to S_1 \ldots [S_{r-}]} \sum_{s_{t+1} \ldots s_{N-1} \to [S_{r+}] \ldots S_K} P(s_0 \ldots s_{t-1}, s_t = S_r\, |\mathbf{X}) P(s_{t+1} \ldots s_{N-1}\, |\mathbf{X})$$

$$= \sum_{s_0 \ldots s_{t-1} \to S_1 \ldots [S_{r-}]} P(s_0 \ldots s_{t-1}, s_t = S_r\, |\mathbf{X}) \sum_{s_{t+1} \ldots s_{N-1} \to [S_{r+}] \ldots S_K} P(s_{t+1} \ldots s_{N-1}\, |\mathbf{X})$$

A posteriori probabilities of symbols

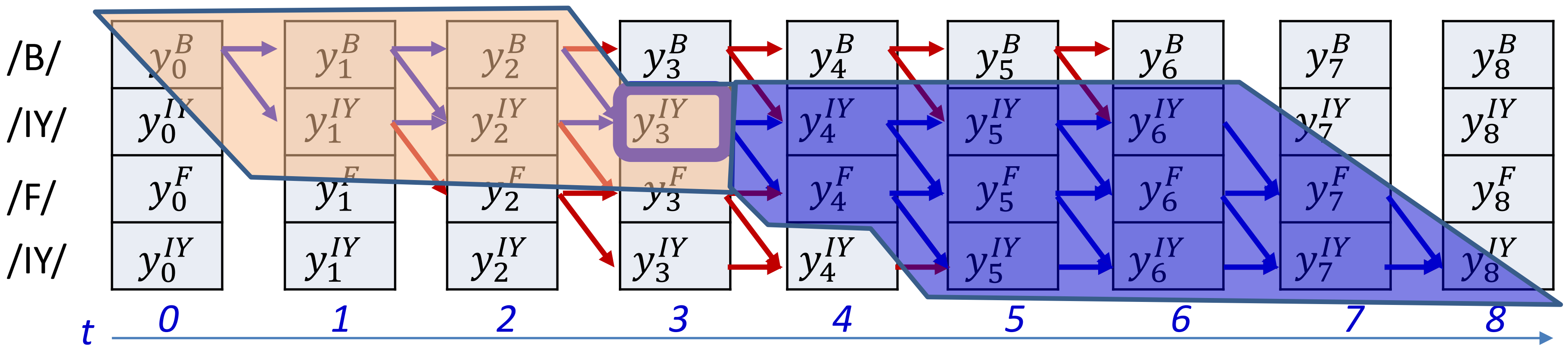


$P(s_t = S_r, \mathbf{S}|\mathbf{X})$

$$= \sum_{s_0 \ldots s_{t-1} \to S_1 \ldots [S_{r-}]} \sum_{s_{t+1} \ldots s_{N-1} \to [S_{r+}] \ldots S_K} P(s_0 \ldots s_{t-1}, s_t = S_r\, |\mathbf{X}) P(s_{t+1} \ldots s_{N-1}\, |\mathbf{X})$$

$$= \boxed{\sum_{s_0 \ldots s_{t-1} \to S_1 \ldots [S_{r-}]} P(s_0 \ldots s_{t-1}, s_t = S_r\, |\mathbf{X})} \boxed{\sum_{s_{t+1} \ldots s_{N-1} \to [S_{r+}] \ldots S_K} P(s_{t+1} \ldots s_{N-1}\, |\mathbf{X})}$$

The expectation over *all* alignments

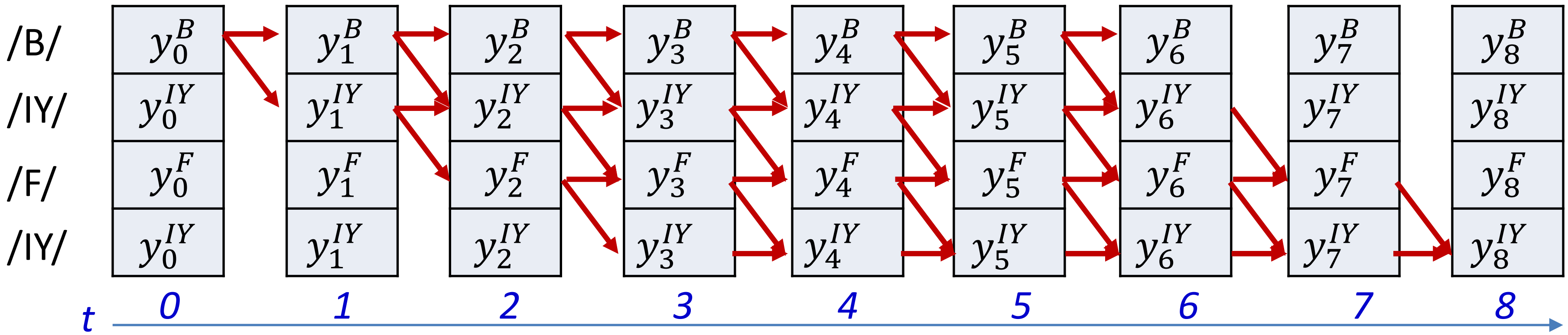


$$P(s_t = S_r, \mathbf{S}|\mathbf{X})$$

$$= \sum_{s_0 \ldots s_{t-1} \to S_1 \ldots [S_{r-}]} P(s_0 \ldots s_{t-1}, s_t = S_r \,|\mathbf{X}) \sum_{s_{t+1} \ldots s_{N-1} \to [S_{r+}] \ldots S_K} P(s_{t+1} \ldots s_{N-1} \,|\mathbf{X})$$

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

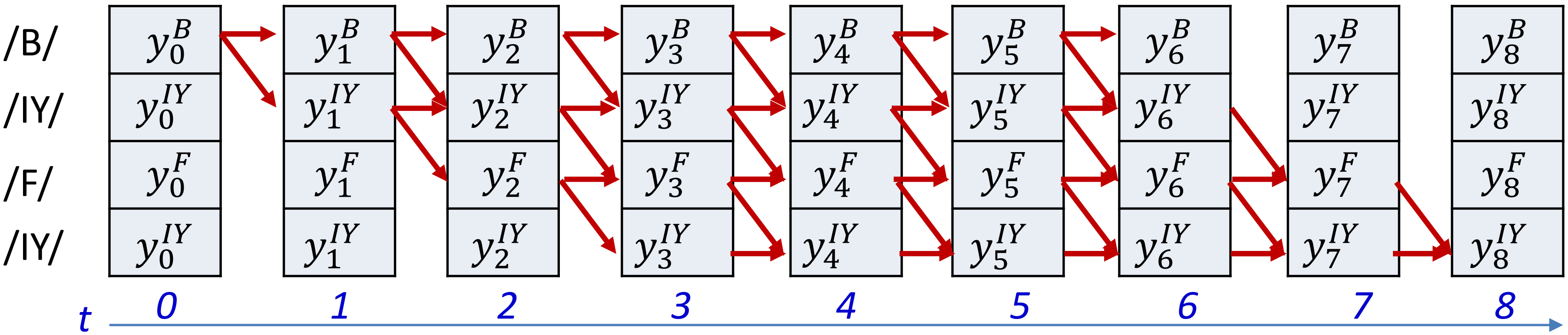
Forward algorithm



$$\alpha(t,r) = \sum_{s_0 \ldots s_{t-1} \to S_1 \ldots [S_{r-}]} P(s_0 \ldots s_{t-1}, s_t = S_r \,|\mathbf{X})$$

$$= \sum_{s_0 \ldots s_{t-1} \to S_1 \ldots [S_{r-}]} P(s_0 \ldots s_{t-1} \,|\mathbf{X}) P(s_t = S_r \,|s_0 \ldots s_{t-1}, \mathbf{X})$$

Forward algorithm

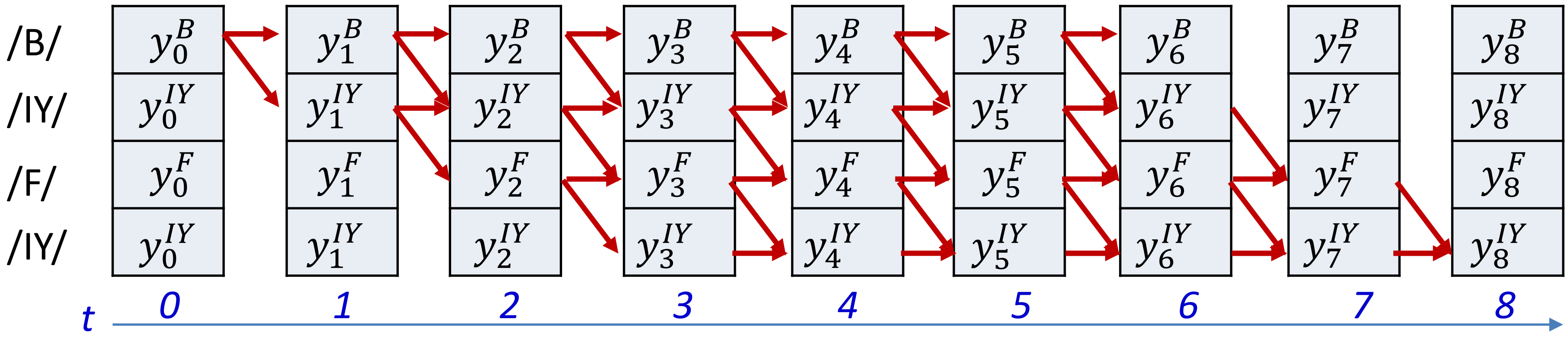


$$\alpha(t,r) = \sum_{s_0 \dots s_{t-1} \to S_1 \dots [S_{r-}]} P(s_0 \dots s_{t-1}, s_t = S_r \,|\mathbf{X})$$

$$= \sum_{s_0 \dots s_{t-1} \to S_1 \dots [S_{r-}]} P(s_0 \dots s_{t-1} \,|\mathbf{X}) P(s_t = S_r \,|s_0 \dots s_{t-1}, \mathbf{X})$$

$$= \sum_{s_0 \dots s_{t-1} \to S_1 \dots [S_{r-}]} P(s_0 \dots s_{t-1} \,|\mathbf{X}) P(s_t = S_r \,|\mathbf{X})$$

Forward algorithm



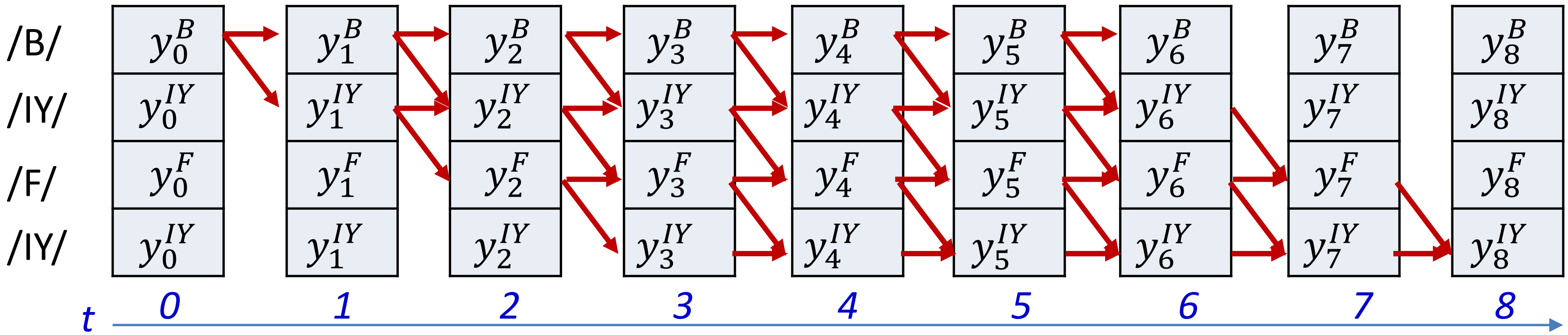
$$\alpha(t,r) = \sum_{s_0 \ldots s_{t-1} \to S_1 \ldots [S_{r-}]} P(s_0 \ldots s_{t-1}, s_t = S_r \,|\mathbf{X})$$

$$= \sum_{s_0 \ldots s_{t-1} \to S_1 \ldots [S_{r-}]} P(s_0 \ldots s_{t-1} \,|\mathbf{X}) P(s_t = S_r \,|s_0 \ldots s_{t-1}, \mathbf{X})$$

$$= \sum_{s_0 \ldots s_{t-1} \to S_1 \ldots [S_{r-}]} P(s_0 \ldots s_{t-1} \,|\mathbf{X}) P(s_t = S_r \,|\mathbf{X})$$

$$= \left(\sum_{s_0 \ldots s_{t-2} \to S_1 \ldots [S_{r-}]} P(s_0 \ldots s_{t-2}, s_{t-1} = S_r |\mathbf{X}) + \sum_{s_0 \ldots s_{t-2} \to S_1 \ldots [S_{(r-1)-}]} P(s_0 \ldots s_{t-2}, s_{t-1} = S_{r-1} |\mathbf{X}) \right) P(s_t = S_r \,|\mathbf{X})$$

Forward algorithm



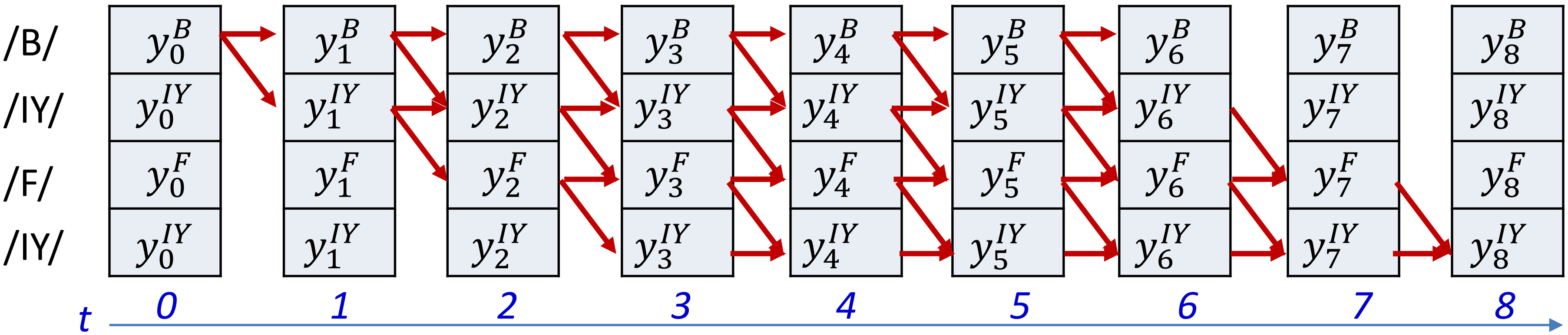
$$\alpha(t,r) = \sum_{s_0 \ldots s_{t-1} \to S_1 \ldots [S_{r-}]} P(s_0 \ldots s_{t-1}, s_t = S_r \,|\mathbf{X})$$

$$= \sum_{s_0 \ldots s_{t-1} \to S_1 \ldots [S_{r-}]} P(s_0 \ldots s_{t-1} \,|\mathbf{X}) P(s_t = S_r \,|s_0 \ldots s_{t-1}, \mathbf{X})$$

$$= \sum_{s_0 \ldots s_{t-1} \to S_1 \ldots [S_{r-}]} P(s_0 \ldots s_{t-1} \,|\mathbf{X}) P(s_t = S_r \,|\mathbf{X})$$

$$= \left(\underbrace{\sum_{s_0 \ldots s_{t-2} \to S_1 \ldots [S_{r-}]} P(s_0 \ldots s_{t-2}, s_{t-1} = S_r|\mathbf{X})}_{\alpha(t-1,r)} + \underbrace{\sum_{s_0 \ldots s_{t-2} \to S_1 \ldots [S_{(r-1)-}]} P(s_0 \ldots s_{t-2}, s_{t-1} = S_{r-1}|\mathbf{X})}_{\alpha(t-1,r-1)} \right) \underbrace{P(s_t = S_r \,|\mathbf{X})}_{y_t^{S(r)}}$$

Forward algorithm



$$\alpha(t,r) = \sum_{s_0 \ldots s_{t-1} \rightarrow S_1 \ldots [S_{r-}]} P(s_0 \ldots s_{t-1}, s_t = S_r \,|\mathbf{X})$$

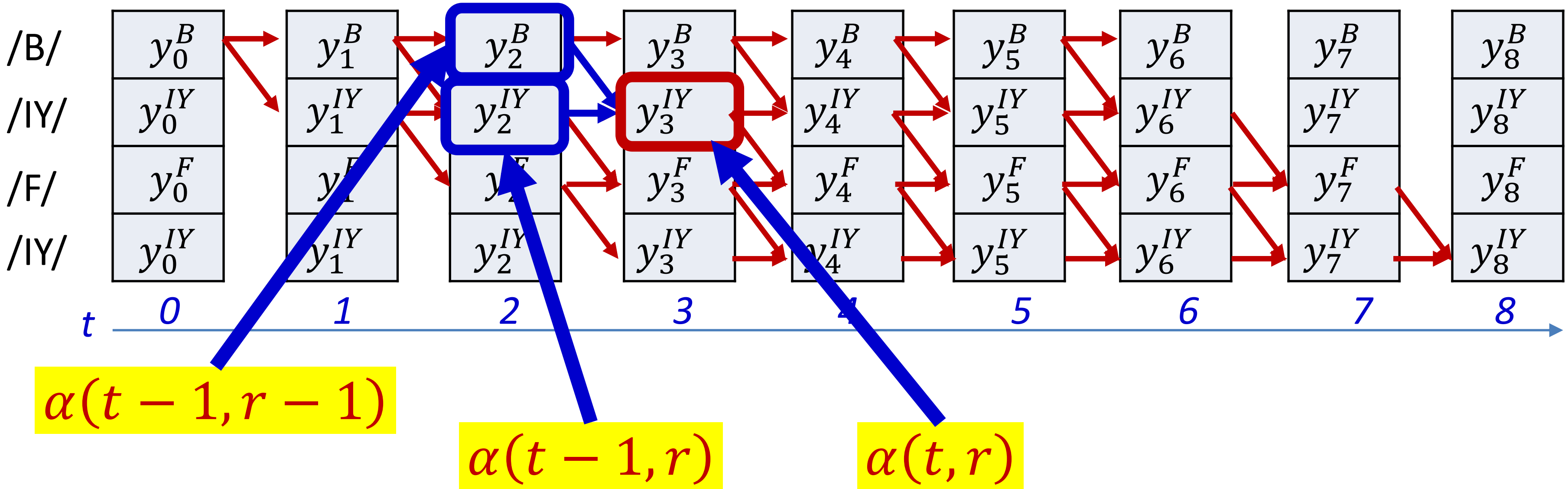
$$= \sum_{s_0 \ldots s_{t-1} \rightarrow S_1 \ldots [S_{r-}]} P(s_0 \ldots s_{t-1} \,|\mathbf{X}) P(s_t = S_r \,|s_0 \ldots s_{t-1}, \mathbf{X})$$

$$= \sum_{s_0 \ldots s_{t-1} \rightarrow S_1 \ldots [S_{r-}]} P(s_0 \ldots s_{t-1} \,|\mathbf{X}) P(s_t = S_r \,|\mathbf{X})$$

$$= \left(\sum_{s_0 \ldots s_{t-2} \rightarrow S_1 \ldots [S_{r-}]} P(s_0 \ldots s_{t-2}, s_{t-1} = S_r |\mathbf{X}) + \sum_{s_0 \ldots s_{t-2} \rightarrow S_1 \ldots [S_{(r-1)-}]} P(s_0 \ldots s_{t-2}, s_{t-1} = S_{r-1} |\mathbf{X}) \right) P(s_t = S_r \,|\mathbf{X})$$

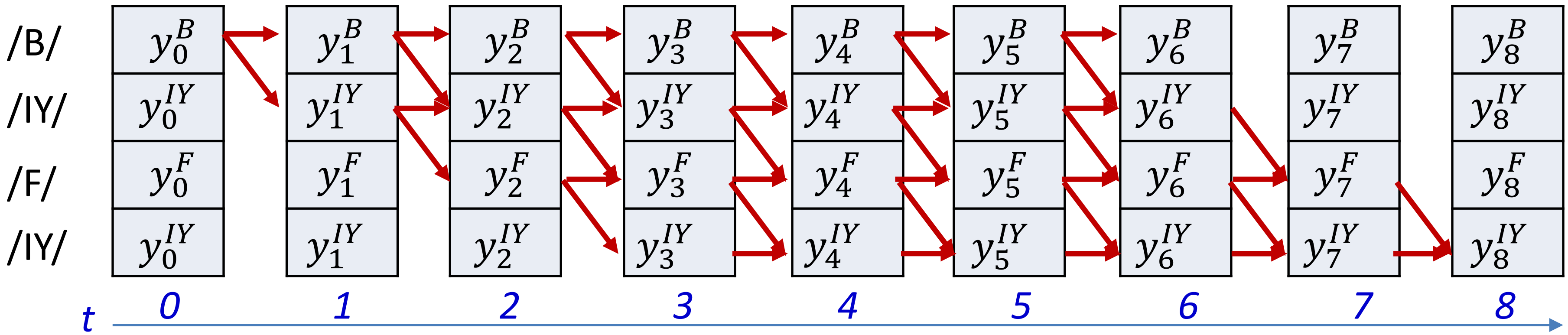
$$\alpha(t,r) = \big(\alpha(t-1,r) + \alpha(t-1,r-1)\big) y_t^{S(r)}$$

Forward algorithm



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

Forward algorithm



- Initialization:

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

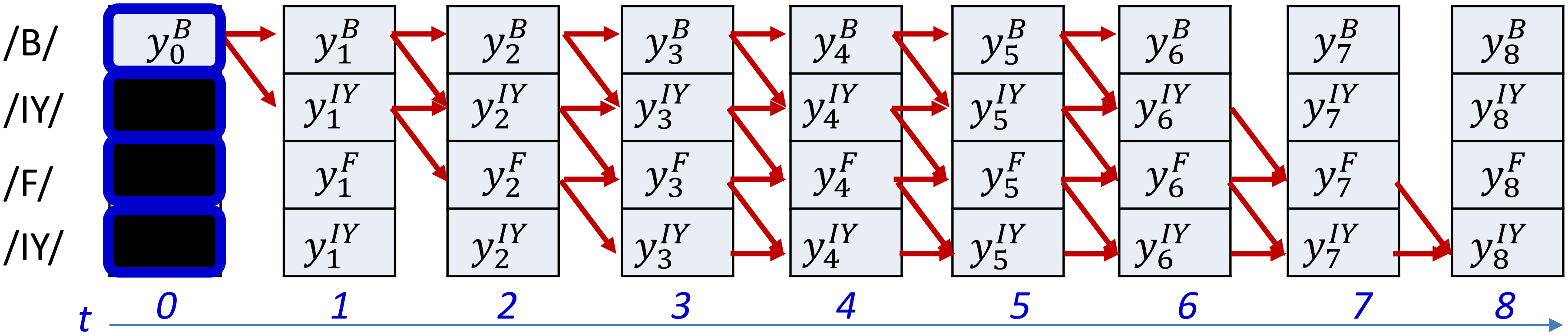
- for $t = 1 \dots T-1$

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

for $l = 2 \dots K$

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

Forward algorithm



- Initialization:

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

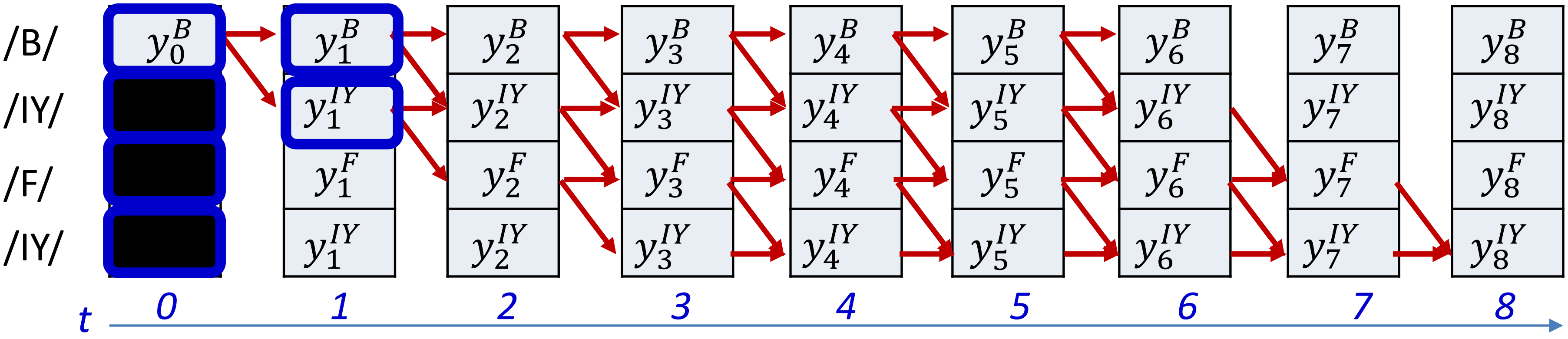
- for $t = 1 \dots T-1$

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

for $l = 2 \dots K$

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

Forward algorithm



- Initialization:

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

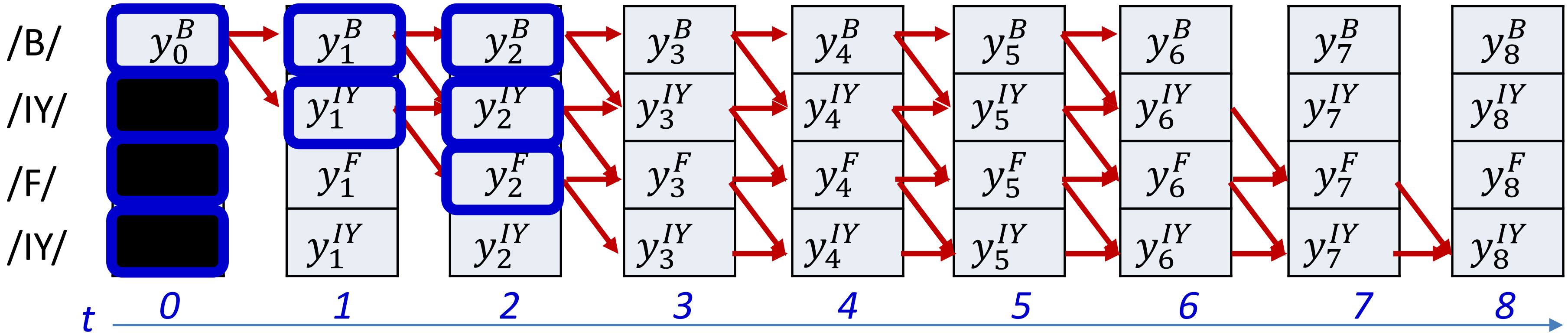
- for $t \ = \ 1 \dots T-1$

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

for $l \ = 2 \dots K$

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

Forward algorithm



- Initialization:

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

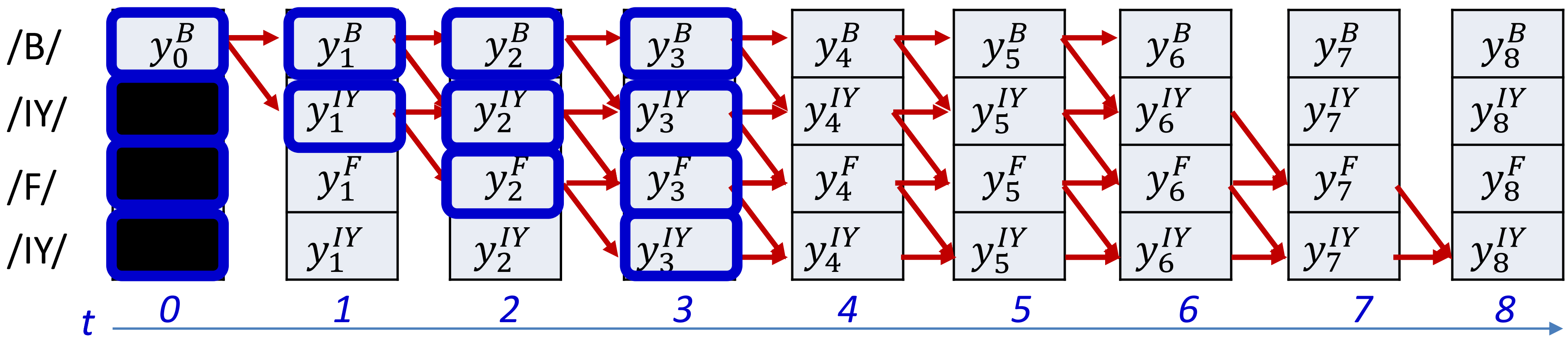
- for $t = 1 \ldots T-1$

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

for $l = 2 \ldots K$

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

Forward algorithm



- Initialization:

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

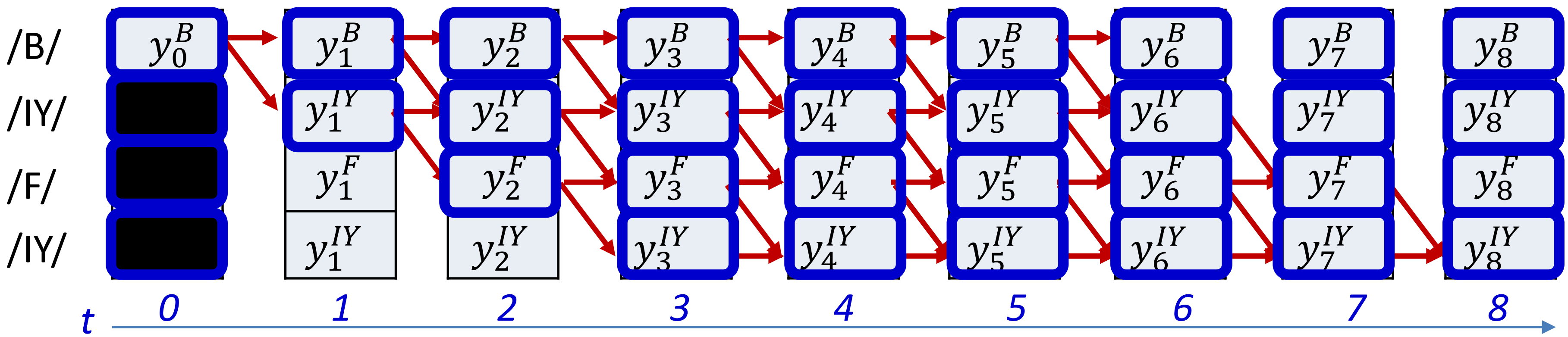
- for $t \;=\; 1 \dots T-1$

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

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 $t = 1 \dots T-1$

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

for $l = 2 \dots K$

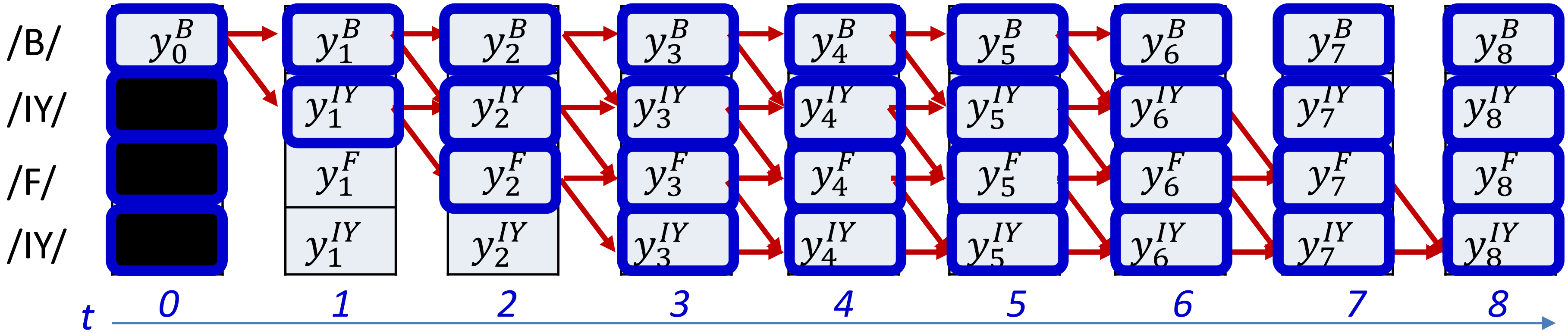
- $\alpha(t,l) = (\alpha(t-1,l) + \alpha(t-1,l-1)) y_t^{S(l)}$

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



- Initialization:

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

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

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

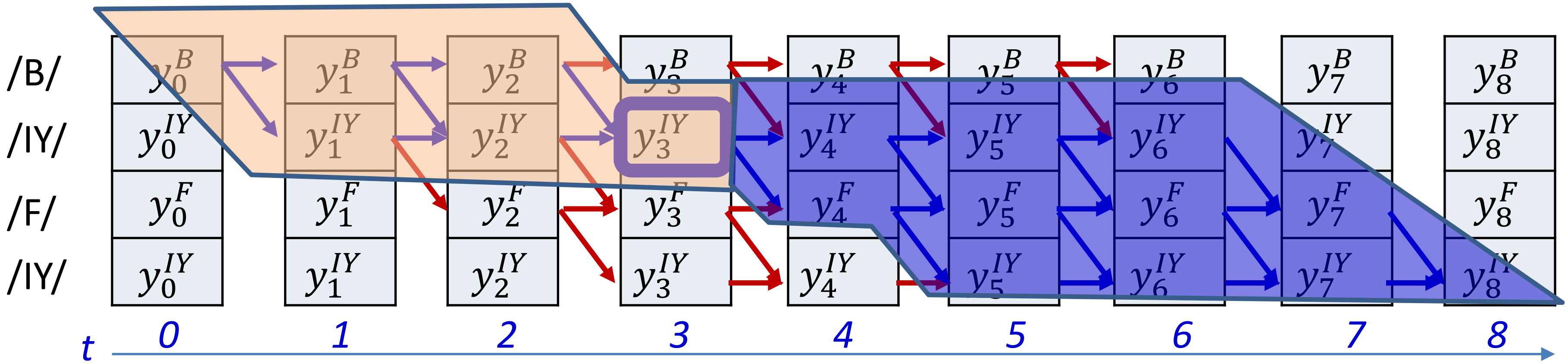
  $\hat{\alpha}(t,1) \;= \alpha(t-1,1)$

  for $l \;= 2 \ldots K$

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

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

The forward probability



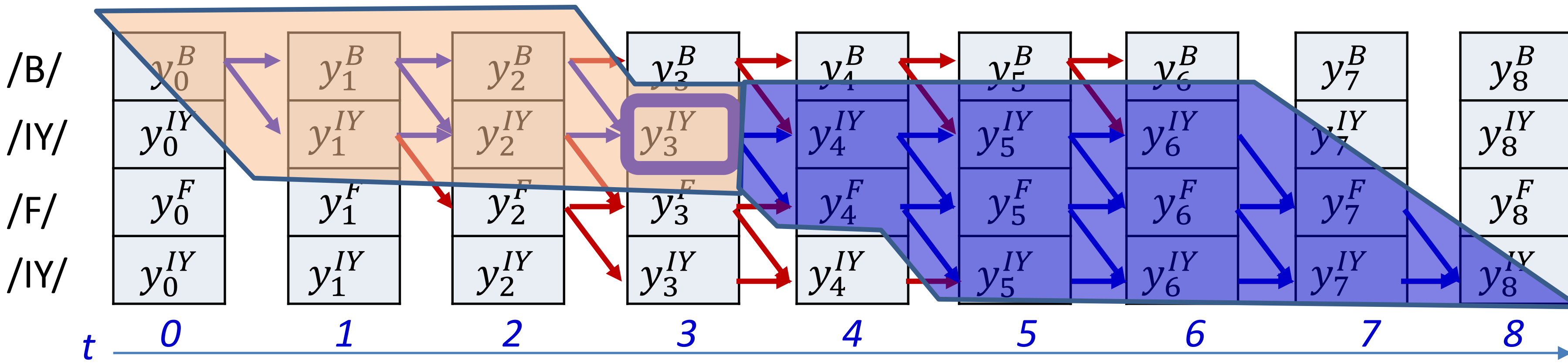
$$P(s_t = S_r, \mathbf{S}|\mathbf{X})$$

$$= \sum_{s_0 \ldots s_{t-1} \to S_1 \ldots [S_{r-}]} P(s_0 \ldots s_{t-1}, s_t = S_r\ |\mathbf{X}) \sum_{s_{t+1} \ldots s_{N-1} \to [S_{r+}] \ldots S_K} P(s_{t+1} \ldots s_{N-1}\ |\mathbf{X})$$

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

We have seen how to compute this $\alpha(t, r)$

The forward probability

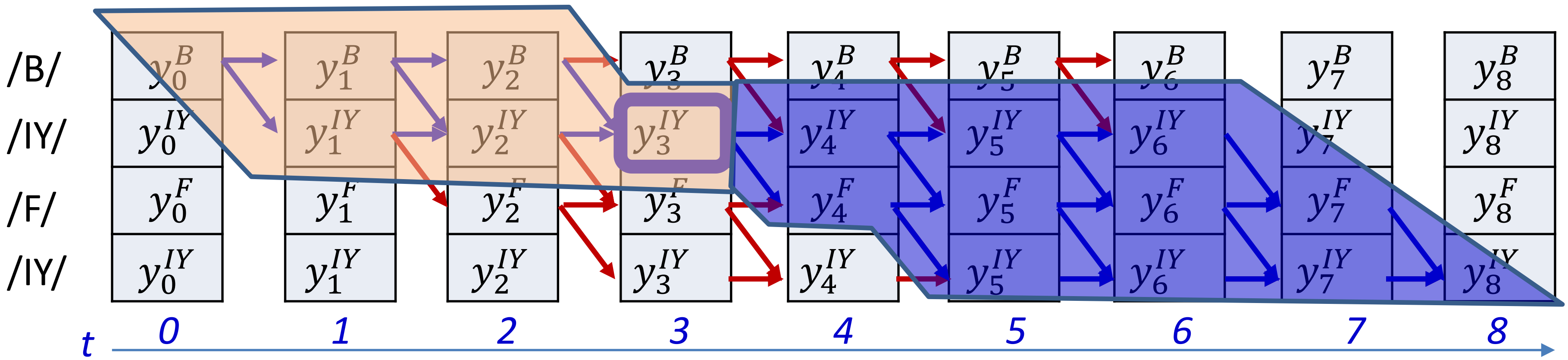


$$P(s_t = S_r, \mathbf{S}|\mathbf{X}) = \alpha(t,r) \sum_{s_{t+1} \ldots s_{N-1} \rightarrow [S_{r+}] \ldots S_K} P(s_{t+1} \ldots s_{N-1} | \mathbf{X})$$

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

We have seen how to compute this

The forward probability

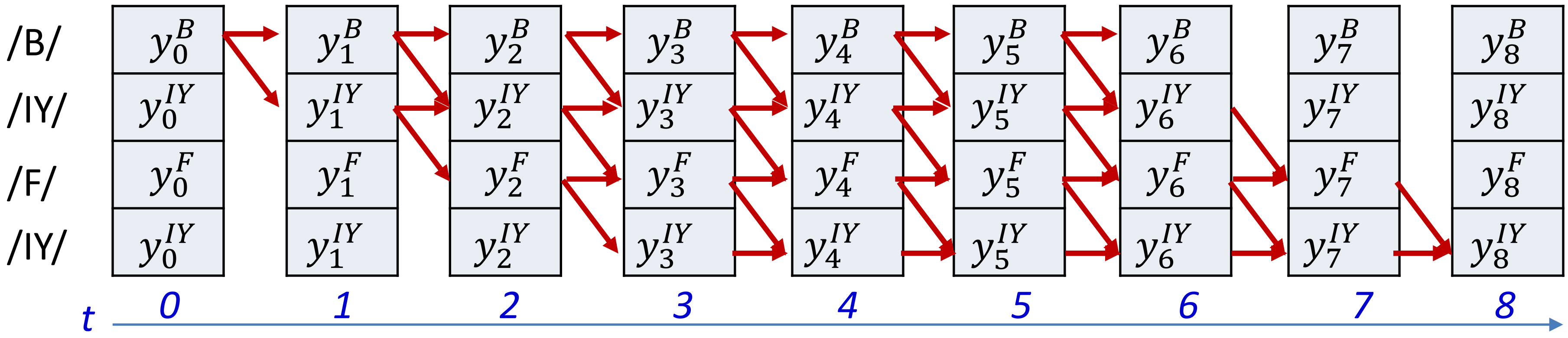


$$P(s_t = S_r, \mathbf{S}|\mathbf{X}) = \alpha(t,r) \sum_{s_{t+1} \ldots s_{N-1} \rightarrow [S_{r+}] \ldots S_K} P(s_{t+1} \ldots s_{N-1} \,|\mathbf{X})$$

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

$\beta(t,r)$ Lets look at this

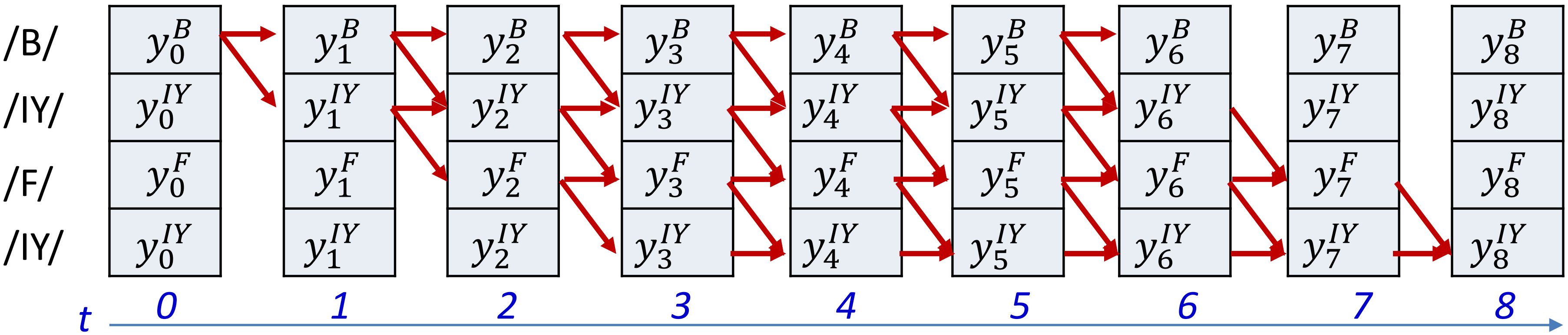
Backward algorithm



$$\beta(t,r) = \sum_{s_{t+1}\dots s_{N-1} \to [S_{r+}]\dots S_K} P(s_{t+1} \dots s_{N-1} \,|\mathbf{X})$$

$$= \sum_{s_{t+2}\dots s_{N-1} \to [S_{r+}]\dots S_K} P(s_{t+1} = S_r, s_{t+2} \dots s_{N-1} \,|\mathbf{X}) + \sum_{s_{t+2}\dots s_{N-1} \to [S_{(r+1)+}]\dots S_K} P(s_{t+1} = S_{r+1}, s_{t+2} \dots s_{N-1} \,|\mathbf{X})$$

Backward algorithm



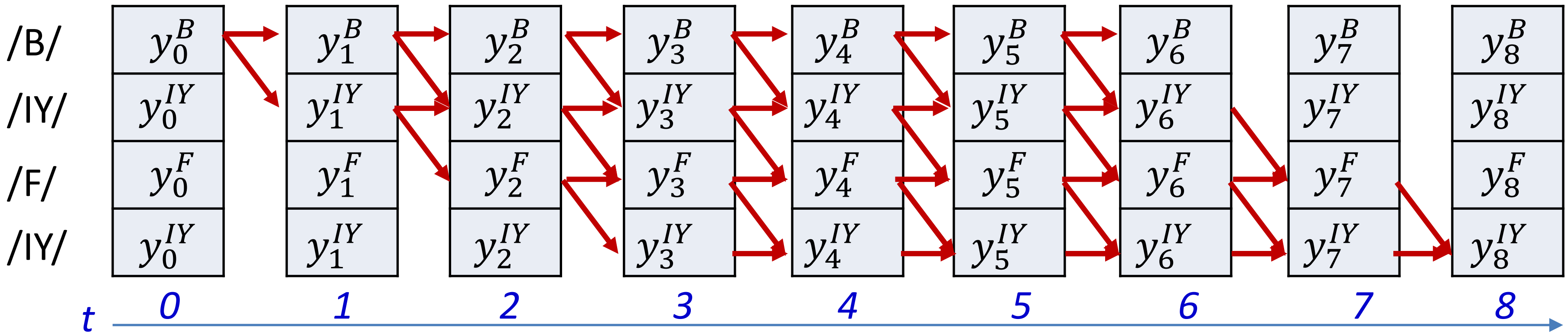
$$\beta(t,r) = \sum_{s_{t+1} \ldots s_{N-1} \rightarrow [S_{r+}] \ldots S_K} P(s_{t+1} \ldots s_{N-1} \,|\mathbf{X})$$

$$= \sum_{s_{t+2} \ldots s_{N-1} \rightarrow [S_{r+}] \ldots S_K} P(s_{t+1} = S_r, s_{t+2} \ldots s_{N-1} \,|\mathbf{X}) + \sum_{s_{t+2} \ldots s_{N-1} \rightarrow [S_{(r+1)+}] \ldots S_K} P(s_{t+1} = S_{r+1}, s_{t+2} \ldots s_{N-1} \,|\mathbf{X})$$

$$= P(s_{t+1} = S_r \,|\mathbf{X}) \sum_{s_{t+2} \ldots s_{N-1} \rightarrow [S_{r+}] \ldots S_K} P(s_{t+2} \ldots s_{N-1} \,|s_{t+1} = S_r, \mathbf{X})$$

$$+ P(s_{t+1} = S_{r+1} \,|\mathbf{X}) \sum_{s_{t+2} \ldots s_{N-1} \rightarrow [S_{(r+1)+}] \ldots S_K} P(s_{t+2} \ldots s_{N-1} \,|s_{t+1} = S_{r+1}, \mathbf{X})$$

Backward algorithm



$$\beta(t,r) = \sum_{s_{t+1}\dots s_{N-1}\to[S_{r+}]\dots S_K} P(s_{t+1}\dots s_{N-1}\,|\mathbf{X})$$

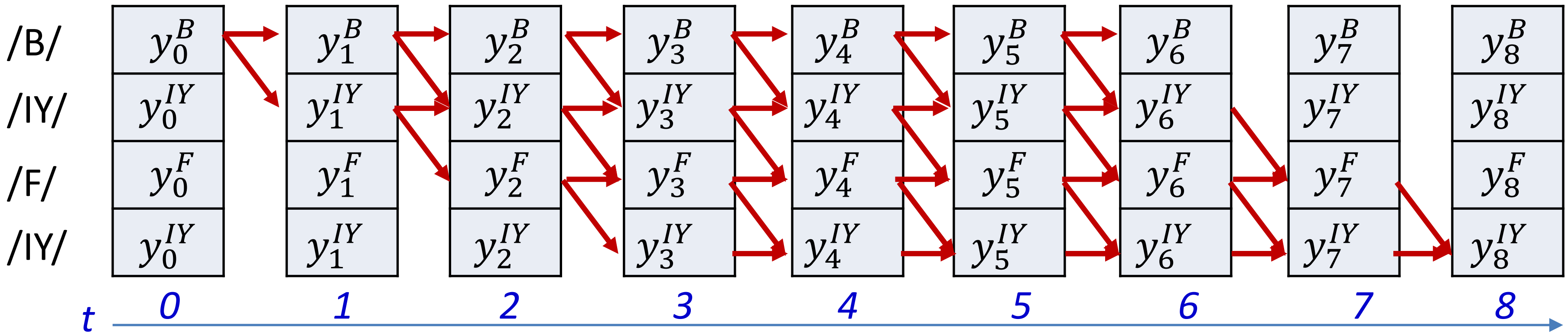
$$= \sum_{s_{t+2}\dots s_{N-1}\to[S_{r+}]\dots S_K} P(s_{t+1}=S_r, s_{t+2}\dots s_{N-1}\,|\mathbf{X}) + \sum_{s_{t+2}\dots s_{N-1}\to[S_{(r+1)+}]\dots S_K} P(s_{t+1}=S_{r+1}, s_{t+2}\dots s_{N-1}\,|\mathbf{X})$$

$$= P(s_{t+1}=S_r\,|\mathbf{X}) \sum_{s_{t+2}\dots s_{N-1}\to[S_{r+}]\dots S_K} P(s_{t+2}\dots s_{N-1}\,|s_{t+1}=S_r, \mathbf{X})$$

$$+ P(s_{t+1}=S_{r+1}\,|\mathbf{X}) \sum_{s_{t+2}\dots s_{N-1}\to[S_{(r+1)+}]\dots S_K} P(s_{t+2}\dots s_{N-1}\,|s_{t+1}=S_{r+1}, \mathbf{X})$$

$$= P(s_{t+1}=S_r\,|\mathbf{X}) \sum_{s_{t+2}\dots s_{N-1}\to[S_{r+}]\dots S_K} P(s_{t+2}\dots s_{N-1}\,|\mathbf{X}) \;+\; P(s_{t+1}=S_{r+1}\,|\mathbf{X}) \sum_{s_{t+2}\dots s_{N-1}\to[S_{(r+1)+}]\dots S_K} P(s_{t+2}\dots s_{N-1}\,|\mathbf{X})$$

Backward algorithm



$$\beta(t,r) = \sum_{s_{t+1}\ldots s_{N-1}\to[S_{r+}]\ldots S_K} P(s_{t+1} \ldots s_{N-1} \,|\mathbf{X})$$

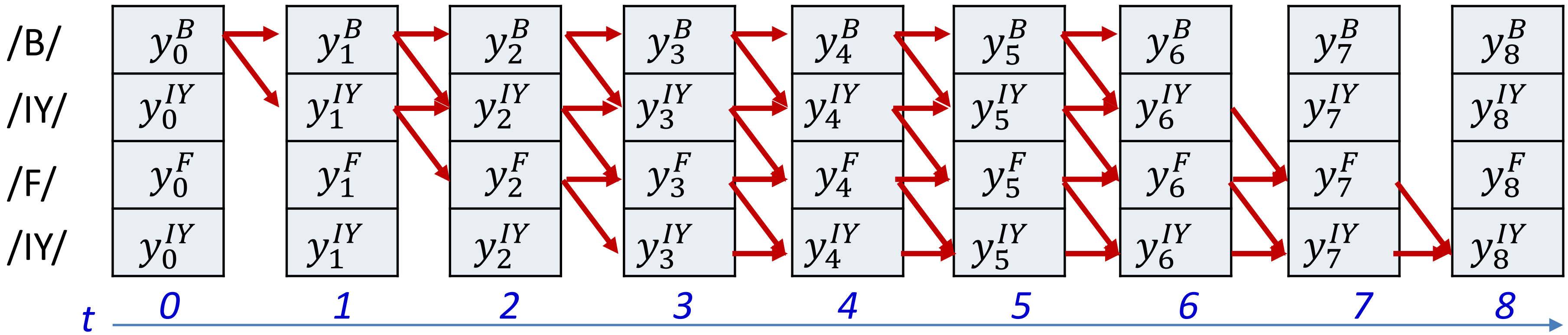
$$= \sum_{s_{t+2}\ldots s_{N-1}\to[S_{r+}]\ldots S_K} P(s_{t+1} = S_r, s_{t+2} \ldots s_{N-1} \,|\mathbf{X}) + \sum_{s_{t+2}\ldots s_{N-1}\to[S_{(r+1)+}]\ldots S_K} P(s_{t+1} = S_{r+1}, s_{t+2} \ldots s_{N-1} \,|\mathbf{X})$$

$$= P(s_{t+1} = S_r \,|\mathbf{X}) \sum_{s_{t+2}\ldots s_{N-1}\to[S_{r+}]\ldots S_K} P(s_{t+2} \ldots s_{N-1} \,|s_{t+1} = S_r, \mathbf{X})$$

$$+ P(s_{t+1} = S_{r+1} \,|\mathbf{X}) \sum_{s_{t+2}\ldots s_{N-1}\to[S_{(r+1)+}]\ldots S_K} P(s_{t+2} \ldots s_{N-1} \,|s_{t+1} = S_{r+1}, \mathbf{X})$$

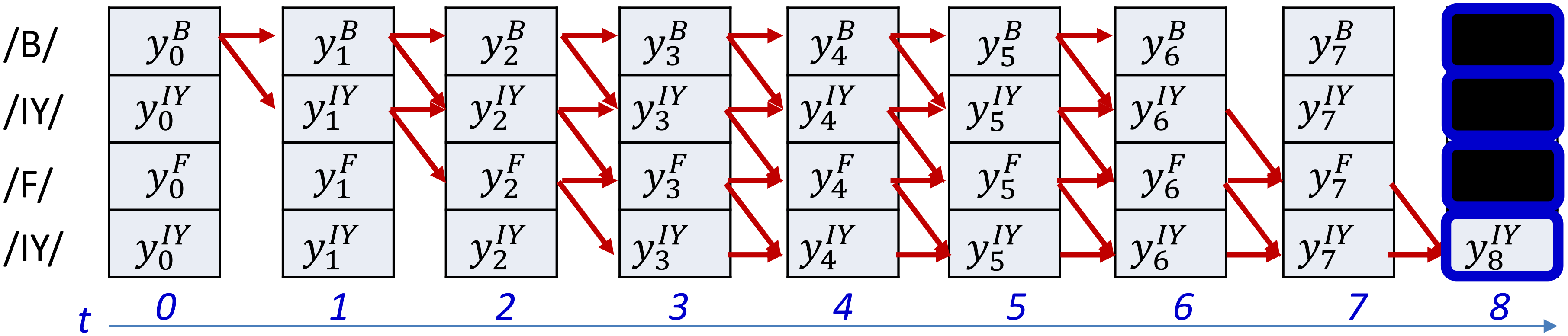
$$= \underbrace{P(s_{t+1} = S_r \,|\mathbf{X})}_{y_{t+1}^{S(r)}} \underbrace{\sum_{s_{t+2}\ldots s_{N-1}\to[S_{r+}]\ldots S_K} P(s_{t+2} \ldots s_{N-1} \,|\mathbf{X})}_{\beta(t+1,r)} + \underbrace{P(s_{t+1} = S_{r+1} \,|\mathbf{X})}_{y_{t+1}^{S(r+1)}} \underbrace{\sum_{s_{t+2}\ldots s_{N-1}\to[S_{(r+1)+}]\ldots S_K} P(s_{t+2} \ldots s_{N-1} \,|\mathbf{X})}_{\beta(t+1,r+1)}$$

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



- Initialization:

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

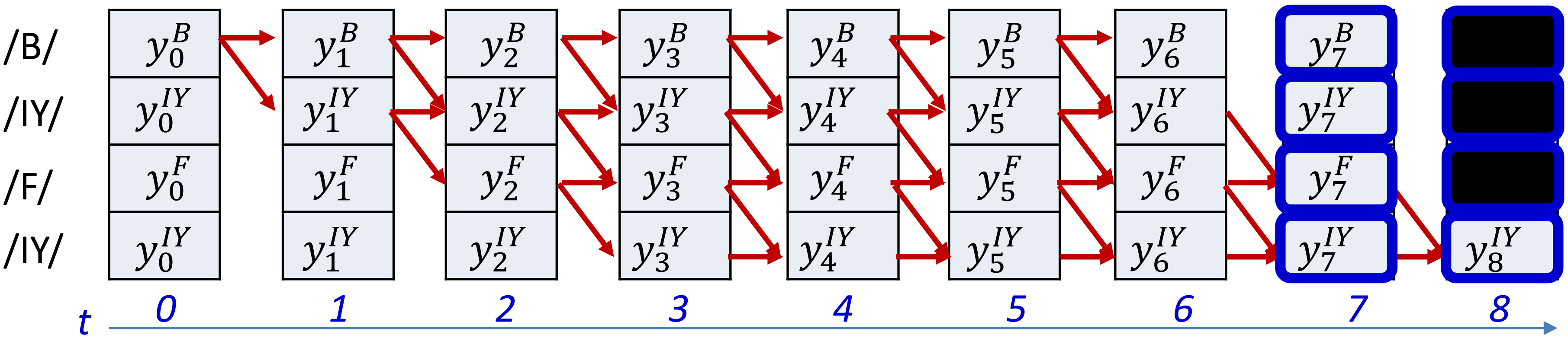
- for $t = T - 2\ downto\ 0$

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

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:

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

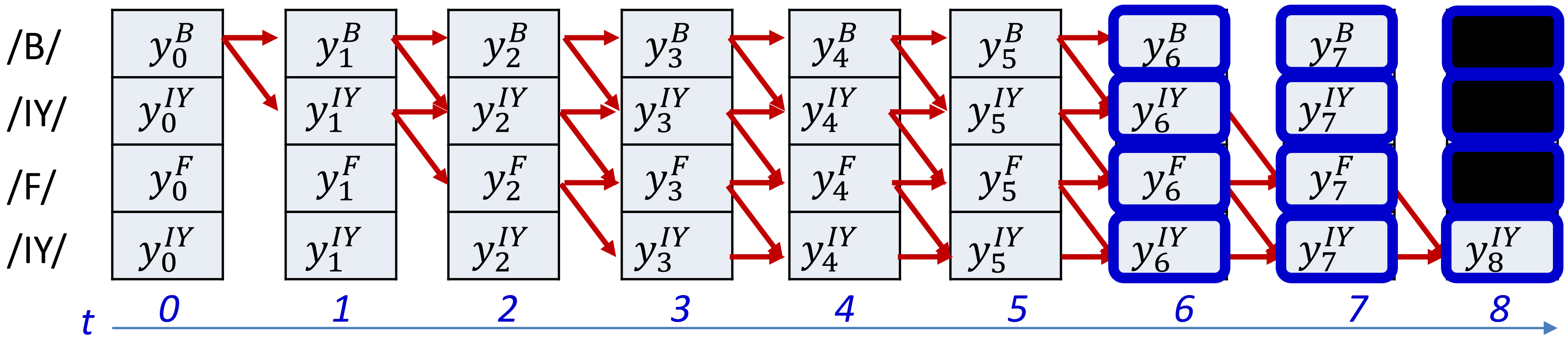
- for $t\ = T-2\ downto\ 0$

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

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:

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

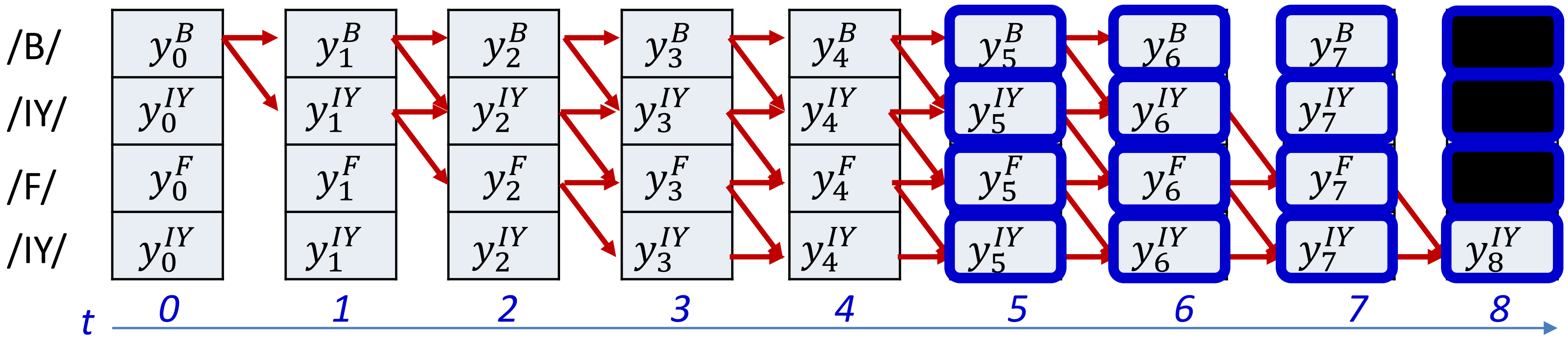
- for $t \; = T - 2 \; downto \; 0$

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

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:

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

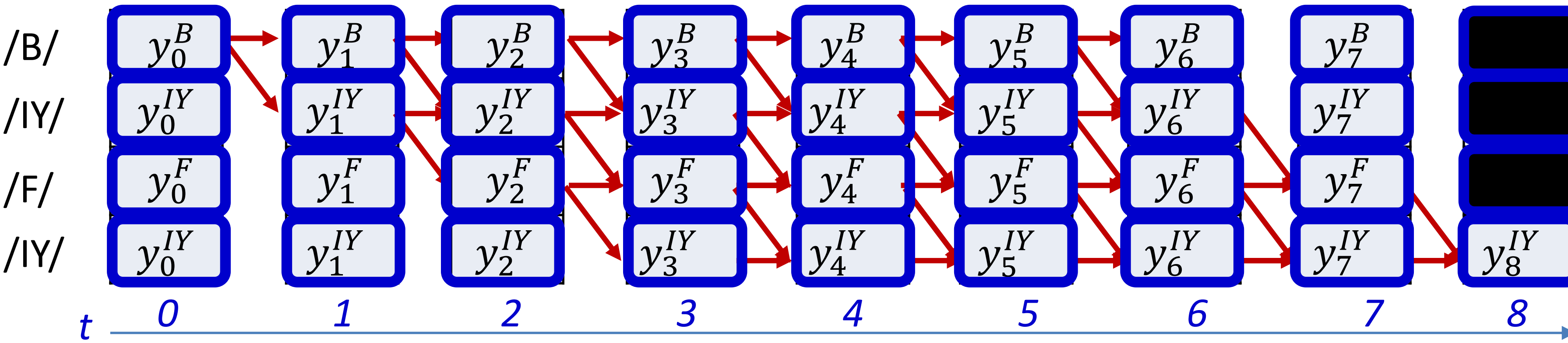
- for $t\ = T-2\ downto\ 0$

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

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

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

Backward algorithm



- Initialization:

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

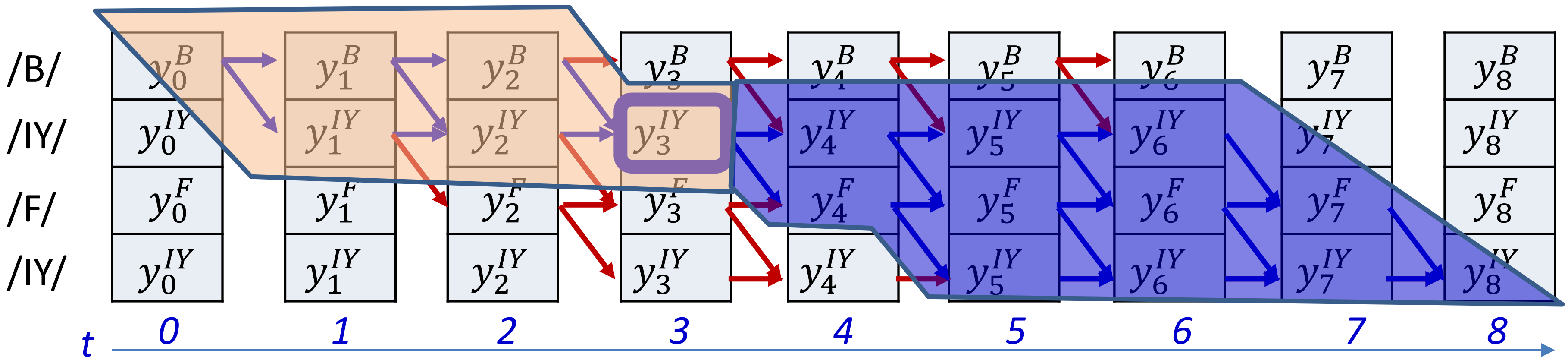
- for $t\ = T-2\ downto\ 0$

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

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

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

The joint probability

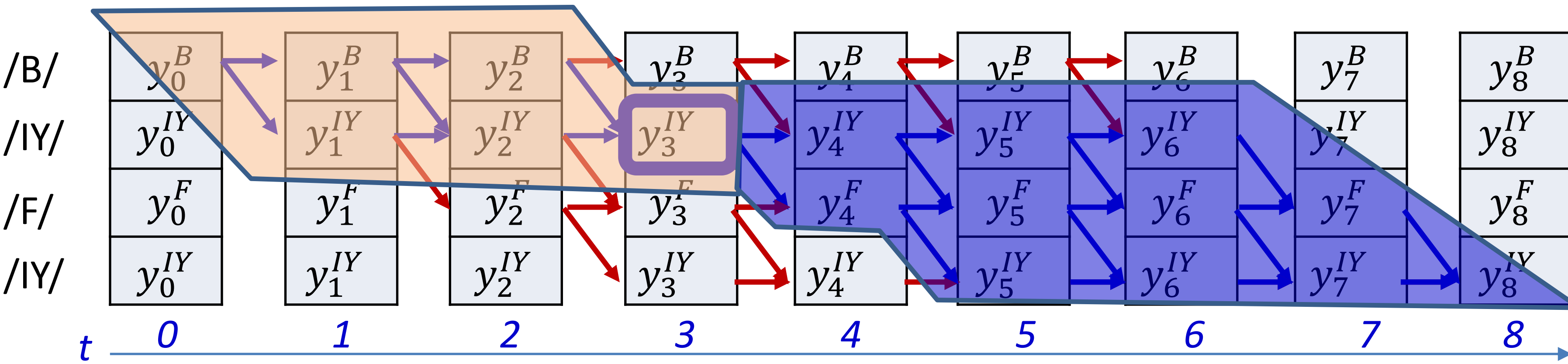


$$P(s_t = S_r, \mathbf{S}|\mathbf{X}) = \alpha(t,r) \sum_{s_{t+1}\ldots s_{N-1} \rightarrow [S_{r+}]\ldots S_K} P(s_{t+1} \ldots s_{N-1} \,|\mathbf{X})$$

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

$\beta(t,r)$ We now can compute this

The joint probability



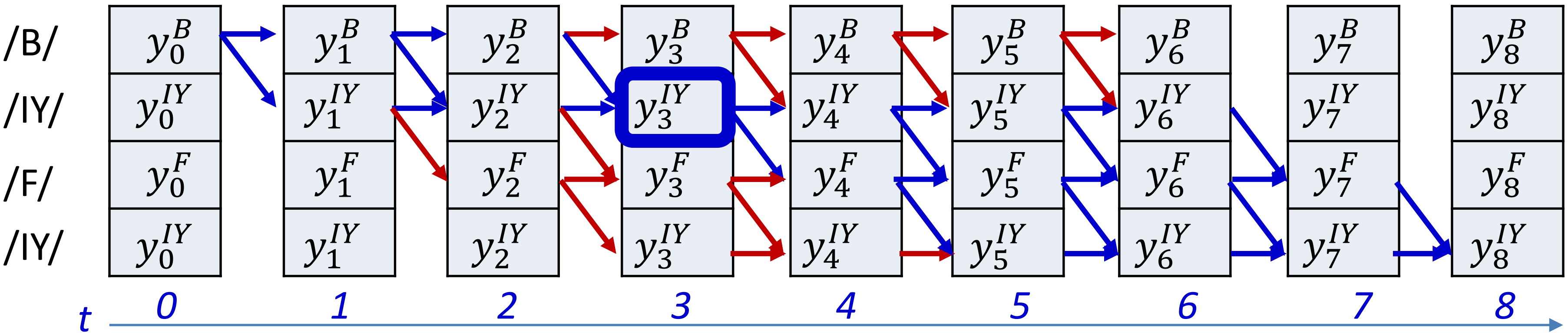
$$P(s_t = S_r, \mathbf{S}|\mathbf{X}) = \alpha(t,r)\beta(t,r)$$

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

Forward algo

Backward algo

The posterior probability



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

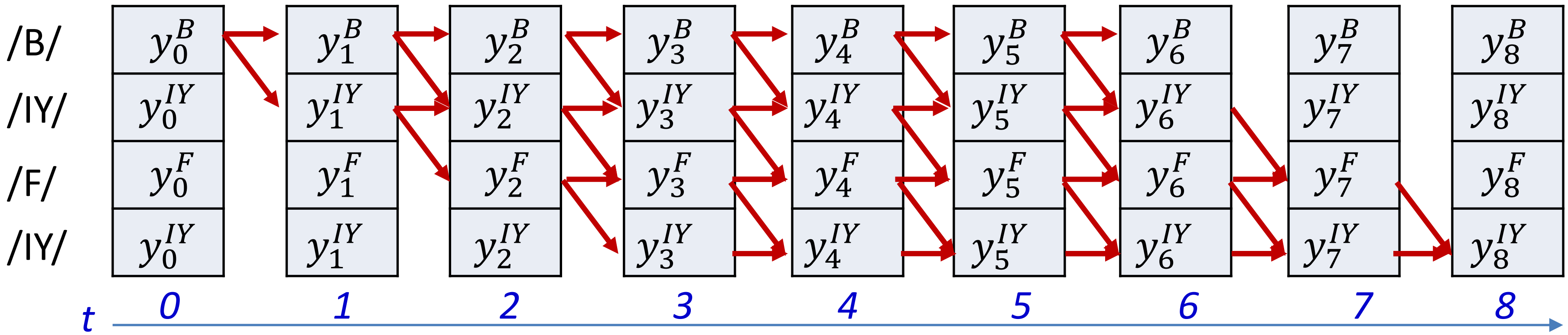
- The *posterior* is given by

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

- We can also write this as

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

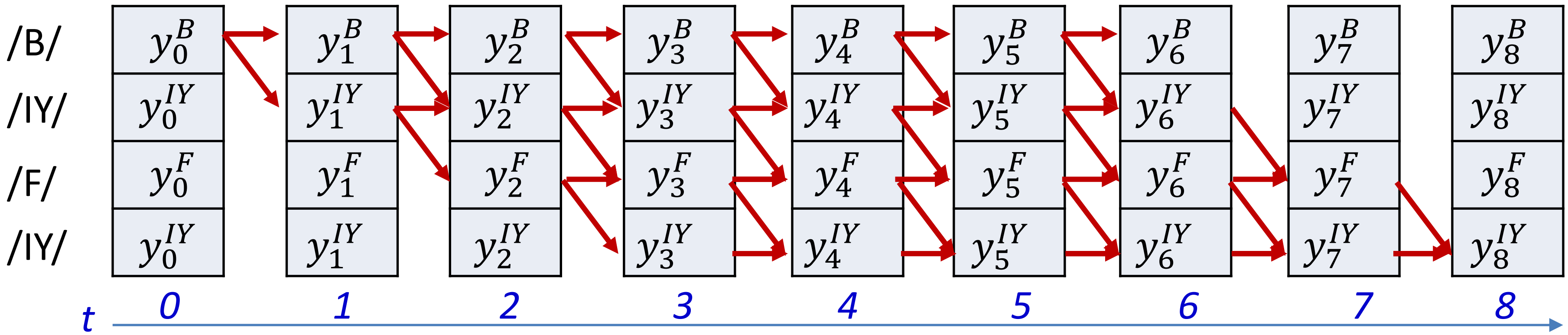
The expected divergence



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

$$DIV = -\sum_t \sum_r \frac{\alpha(t,r)\beta(t,r)}{\sum_{r'} \alpha(t,r')\beta(t,r')} \log y_t^{S(r)}$$

The expected divergence



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

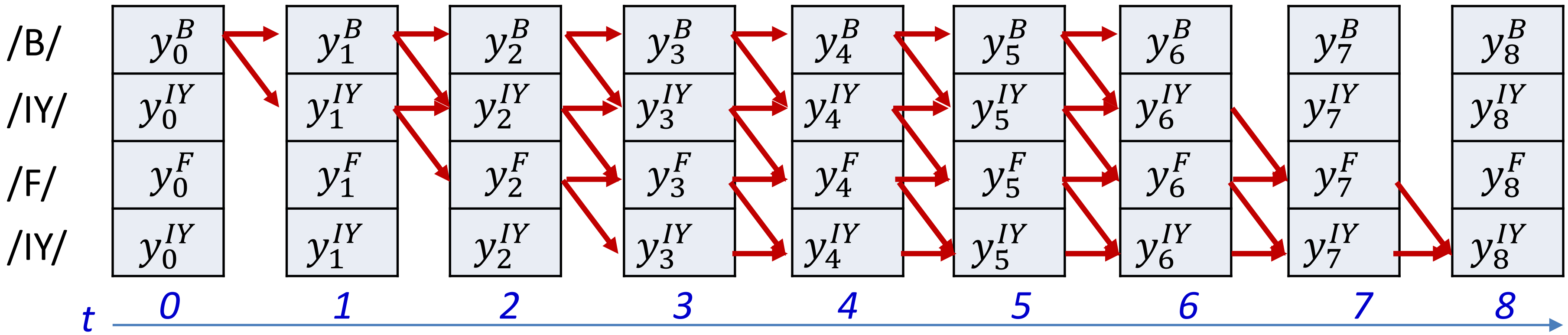
$$DIV = -\sum_{t} \sum_{r} \frac{\alpha(t,r)\beta(t,r)}{\sum_{r'} \alpha(t,r')\beta(t,r')} \log y_t^{S(r)}$$

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

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

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

The expected divergence



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

$$DIV = -\sum_t \sum_r \frac{\alpha(t,r)\beta(t,r)}{\sum_{r'} \alpha(t,r')\beta(t,r')} \log y_t^{S(r)}$$

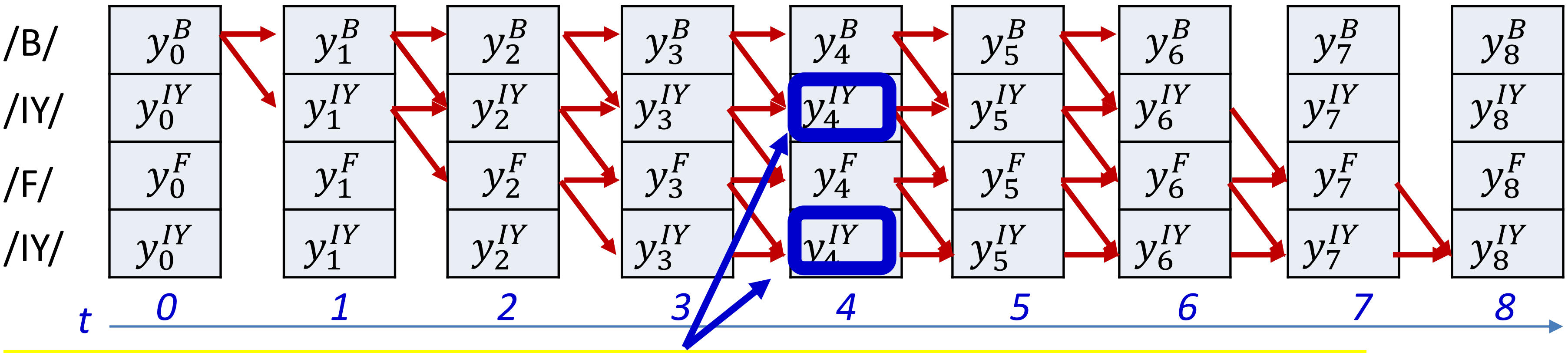
- The derivative of the divergence w.r.t the output Yt of the net at any time:

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

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 $\frac{dDIV}{dy_4^5}$

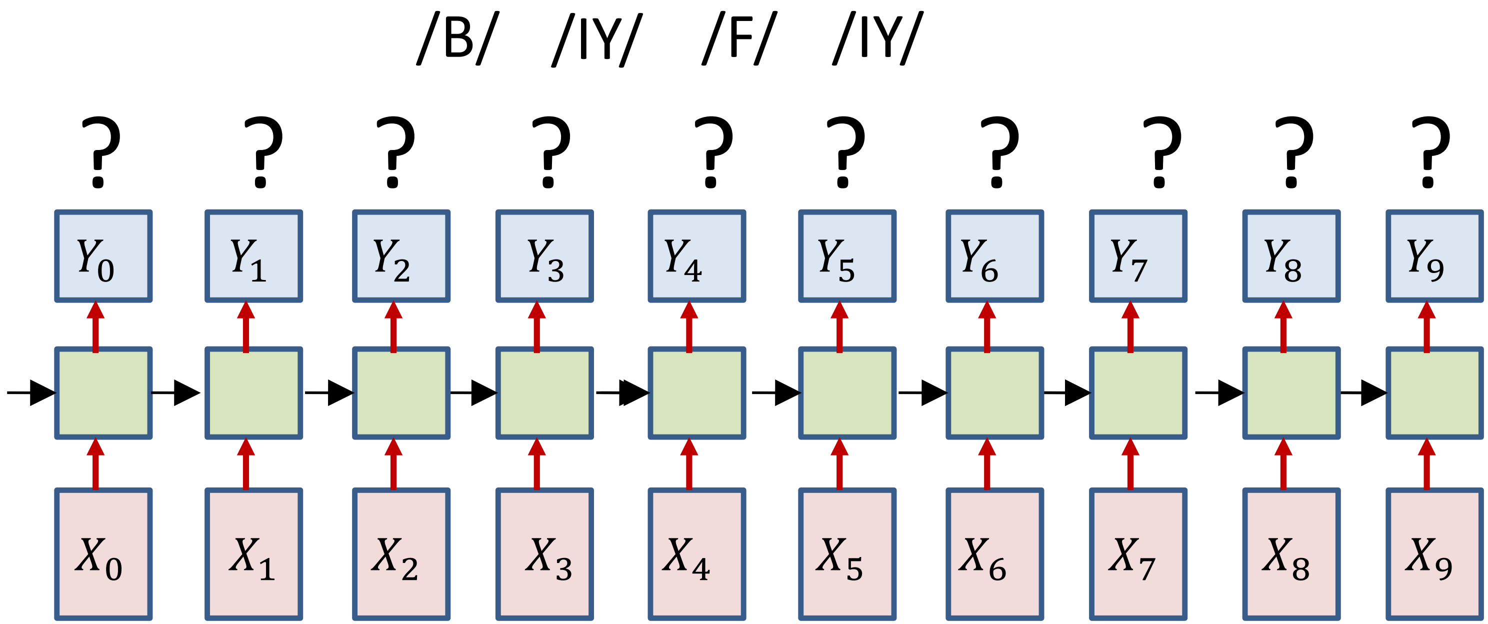
$$DIV = -\sum_t \sum_r \frac{\alpha(t,r)\beta(t,r)}{\sum_{r'} \alpha(t,r')\beta(t,r')} \log y_t^{S(r)}$$

- 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

$$\frac{dDIV}{dy_t^l} = -\sum_{r\,:S(r)=l} \frac{d}{dy_t^{S(r)}} \left(\frac{\alpha(t,r)\beta(t,r)}{\sum_{r'} \alpha(t,r')\beta(t,r')} \log y_t^{S(r)} \right)$$

  - E.g. the derivative w.r.t y_t^5 will sum over both rows representing /IY/ in the above figure

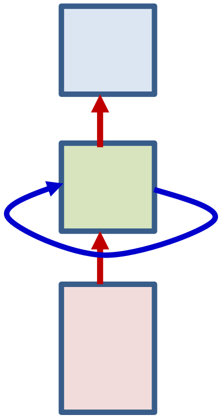
Overall training procedure for Seq2Seq case 1



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

Overall training procedure for Seq2Seq case 1

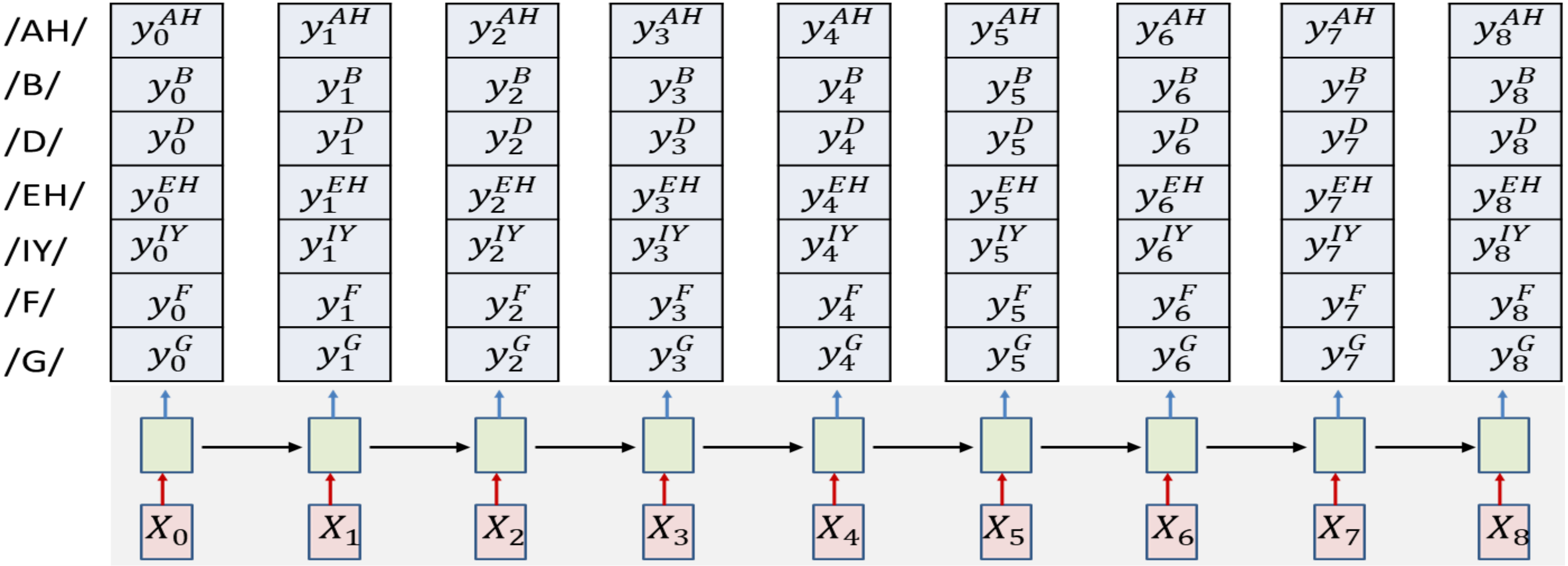
- **Step 1**: Setup the network
  – Typically many-layered LSTM



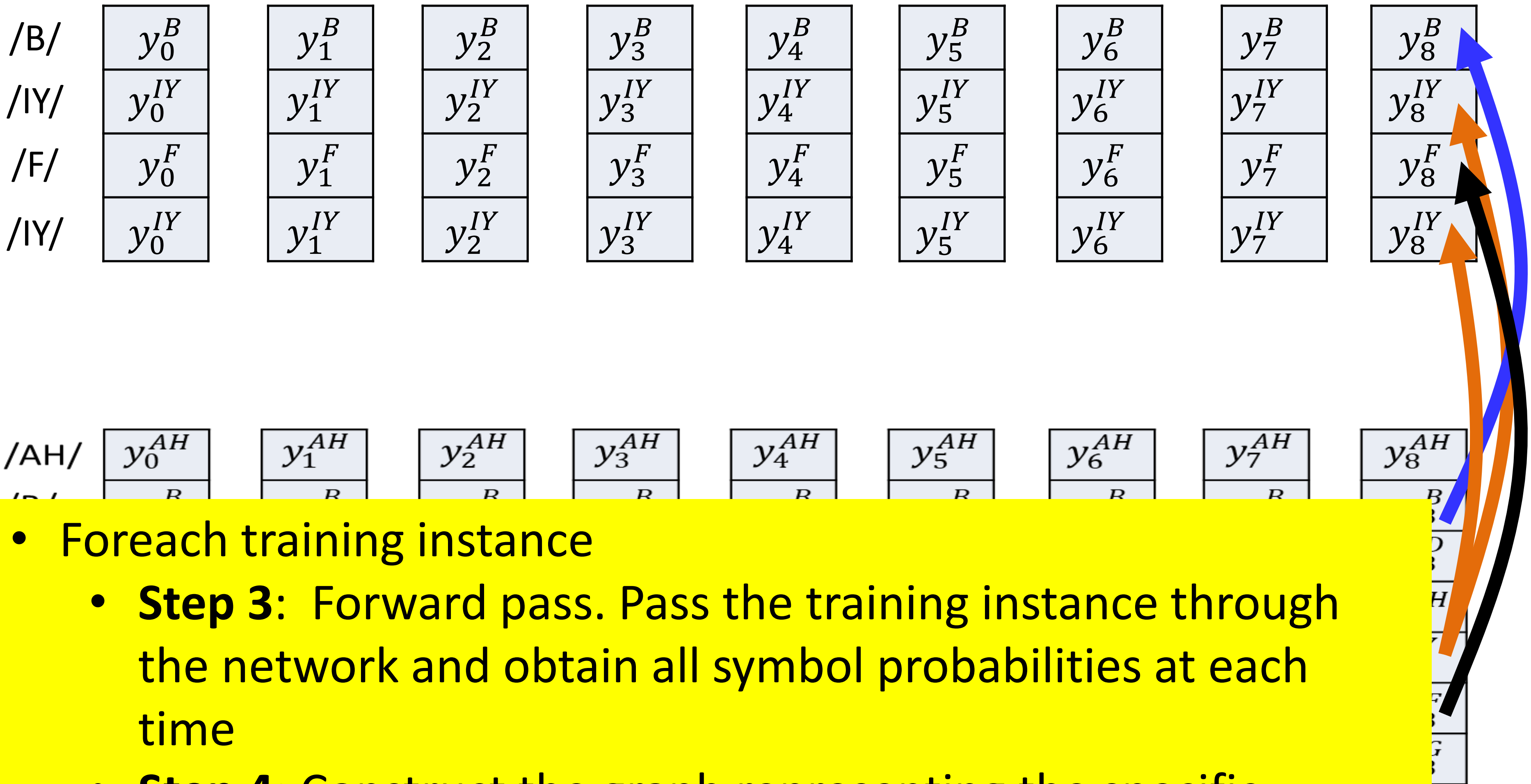
- **Step 2**: Initialize all parameters of the network

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

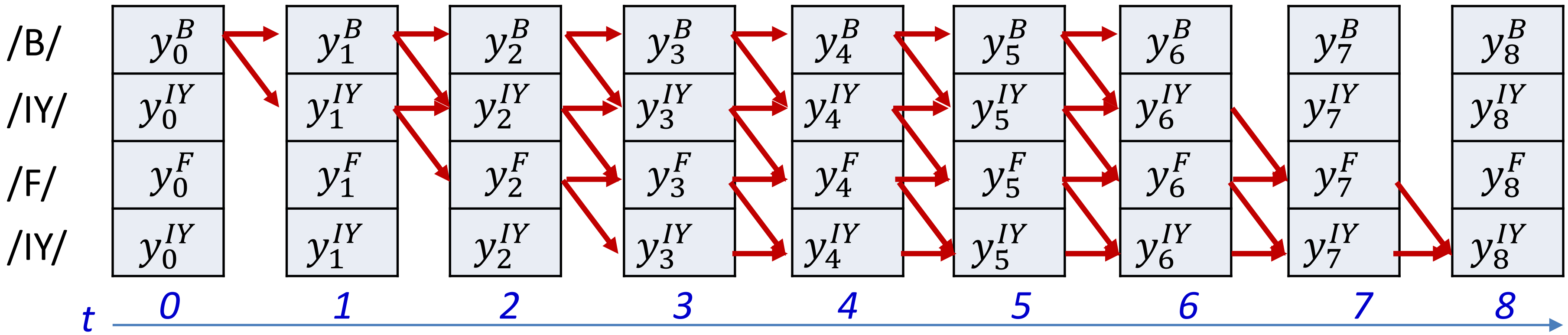


Overall training: Backward pass



- 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

$$\nabla_{Y_t} DIV = \begin{bmatrix} \frac{dDIV}{dy_t^1} & \frac{dDIV}{dy_t^2} & \cdots & \frac{dDIV}{dy_t^L} \end{bmatrix}$$

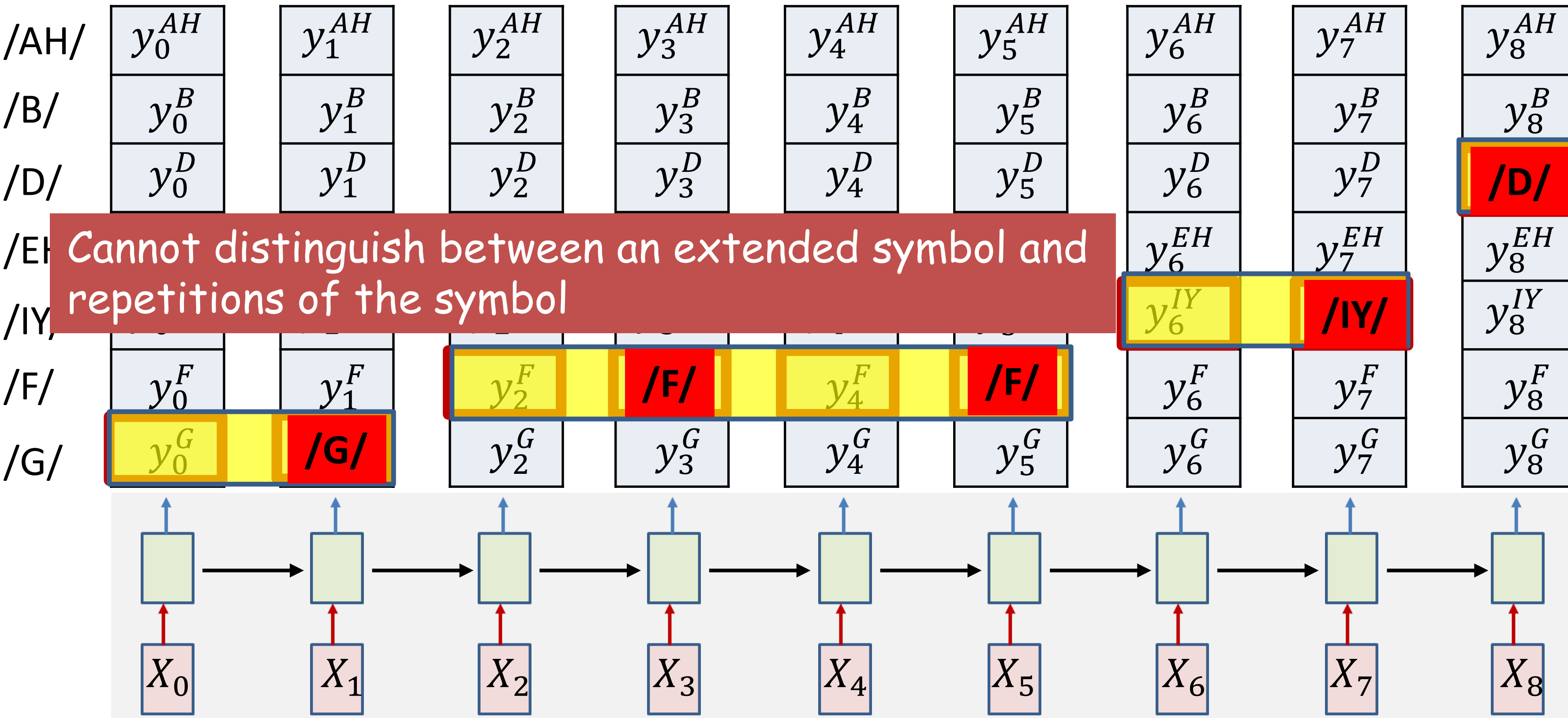
$$\frac{dDIV}{dy_t^l} = -\sum_{r\,:S(r)=l} \frac{d}{dy_t^{S(r)}} \left(\frac{\alpha(t,r)\beta(t,r)}{\sum_{r'} \alpha(t,r')\beta(t,r')} \log y_t^{S(r)} \right)$$

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

A key *decoding* problem

- Consider a problem where the output symbols are characters
- We have a decode: R R R O O O O O D
- Is this the 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. when symbols have sub symbols
  - E.g. If O is produced as O1 and 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

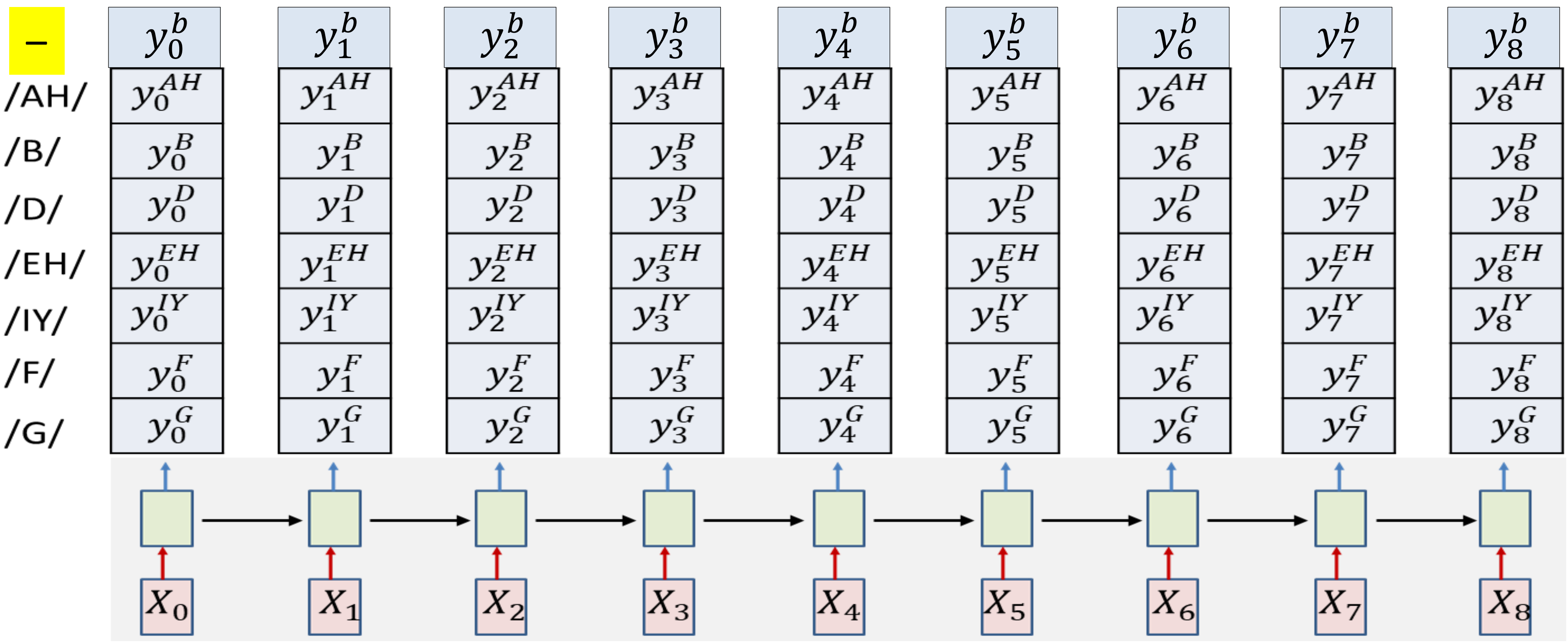
- We have a decode: R R R O O O O O D
- Is this the symbol sequence ROD or ROOD?

- 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

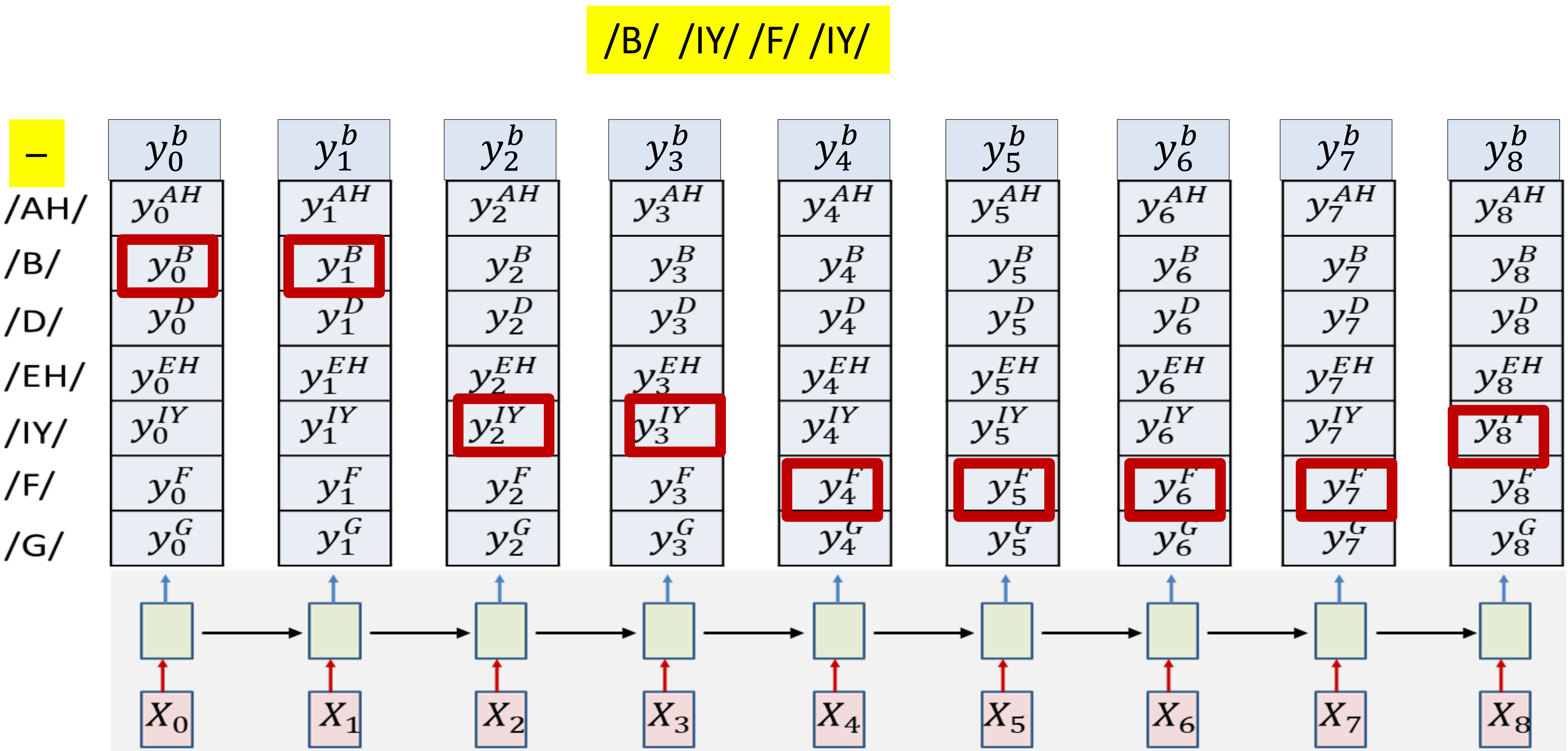
The modified forward output

- Note the extra "blank" at the output



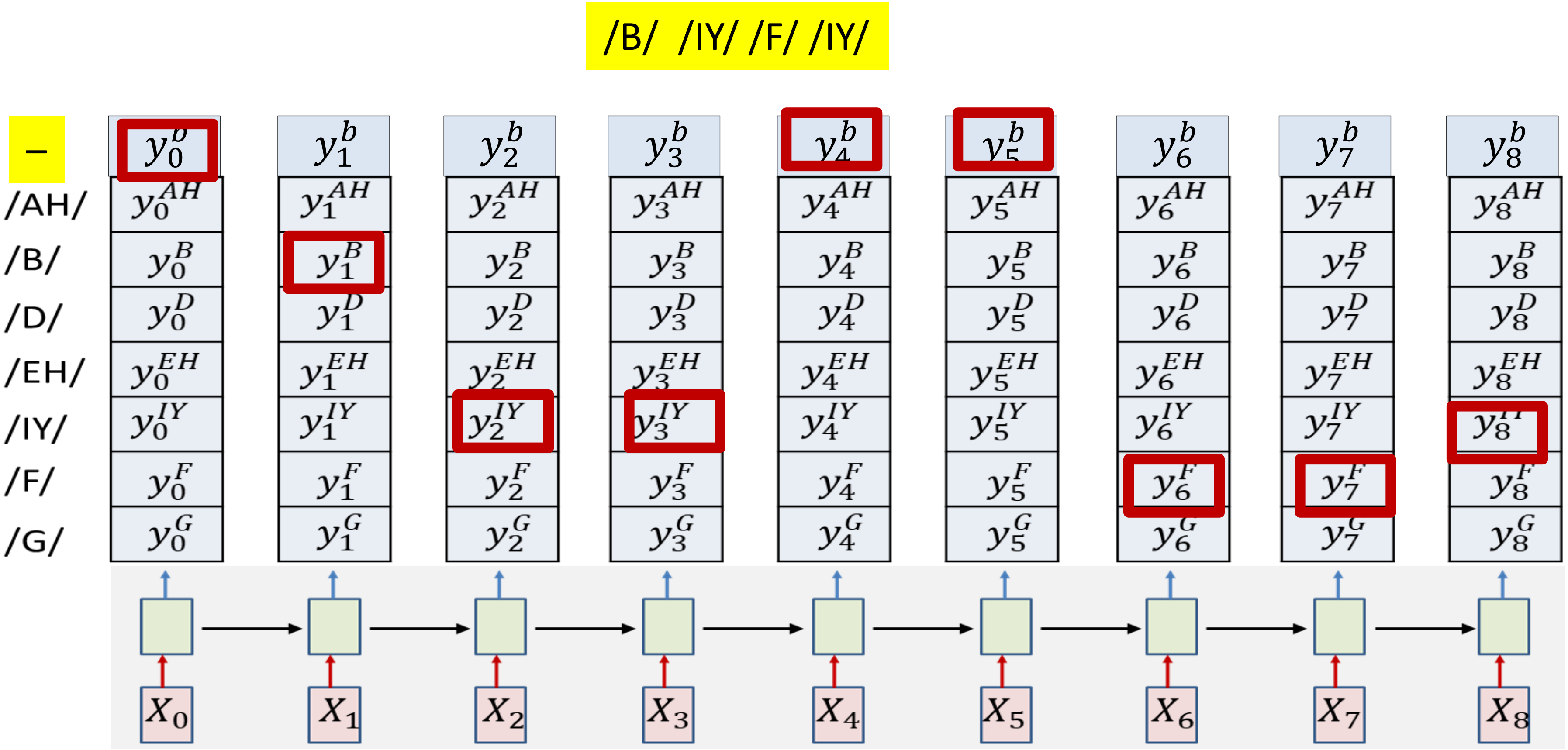
The modified forward output

- Note the extra "blank" at the output



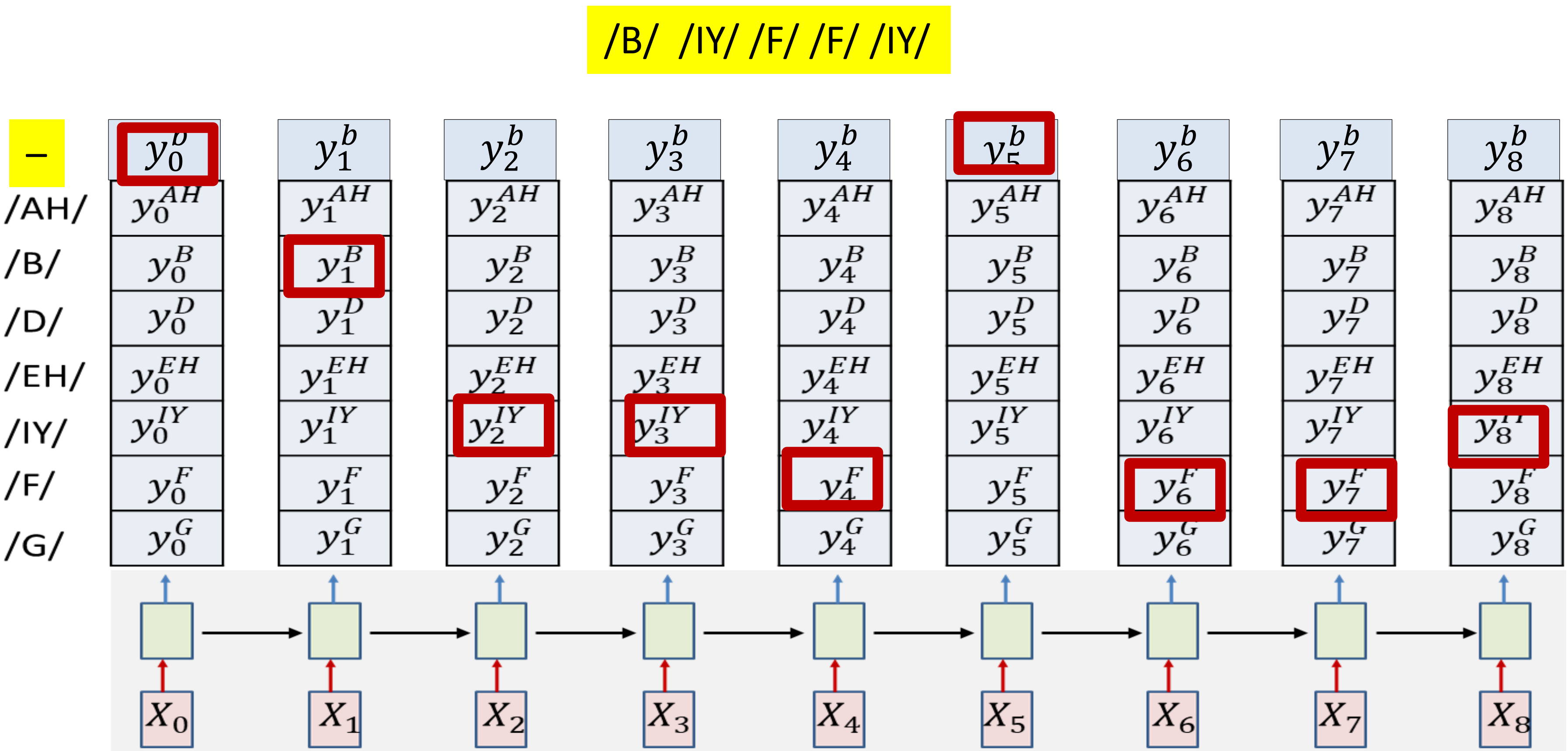
The modified forward output

- Note the extra "blank" at the output

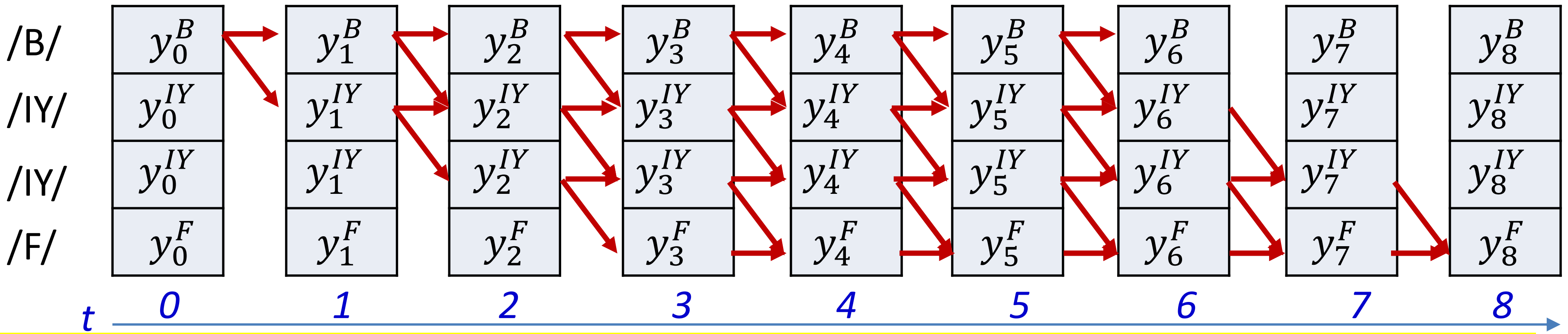


The modified forward output

- Note the extra "blank" at the output



Composing the graph for training



- The original method without blanks

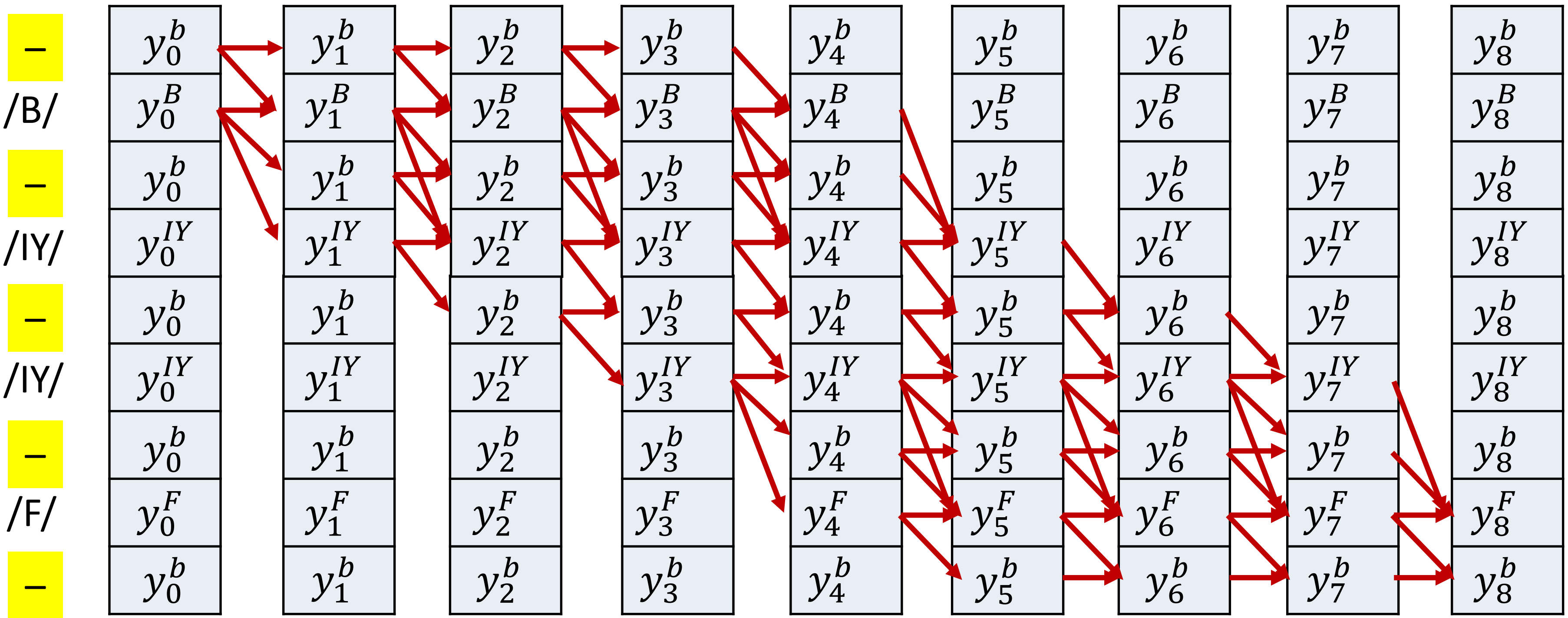
- **Changing the example to /B/ /IY/ /IY/ /F/ from /B/ /IY/ /F/ /IY/ for illustration**

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

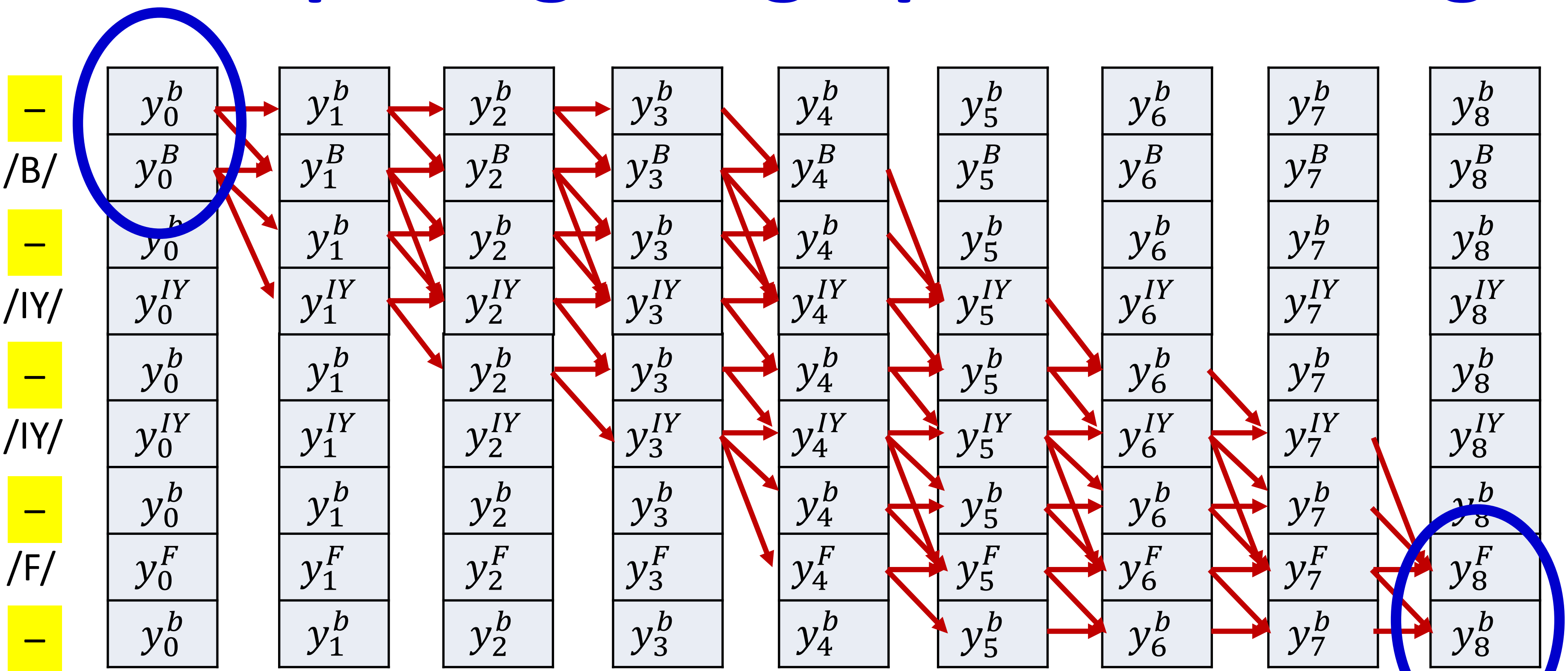
- With blanks
- Note: a row of blanks between any two symbols
- Also blanks at the very beginning and the very end

Composing the graph for training



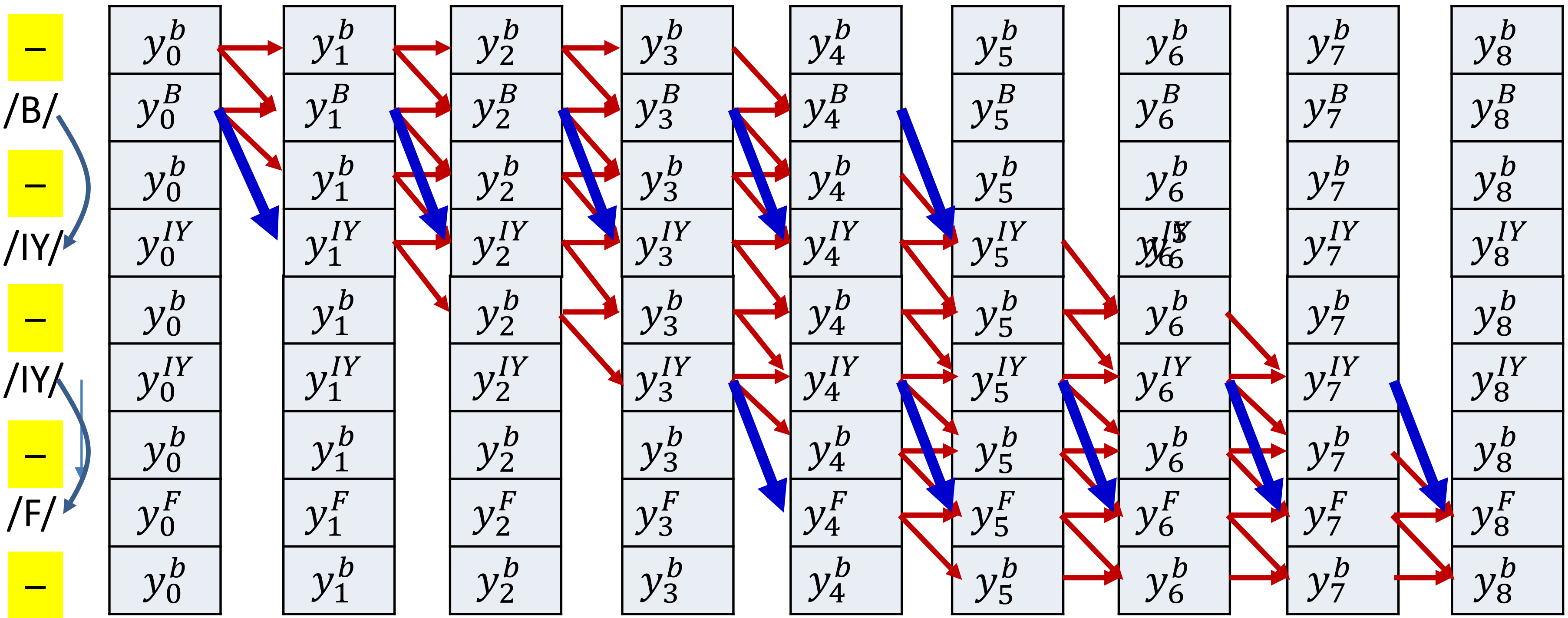
- Add edges such that all paths from initial node(s) to final node(s) unambiguously represent the target symbol sequence

Composing the graph for training



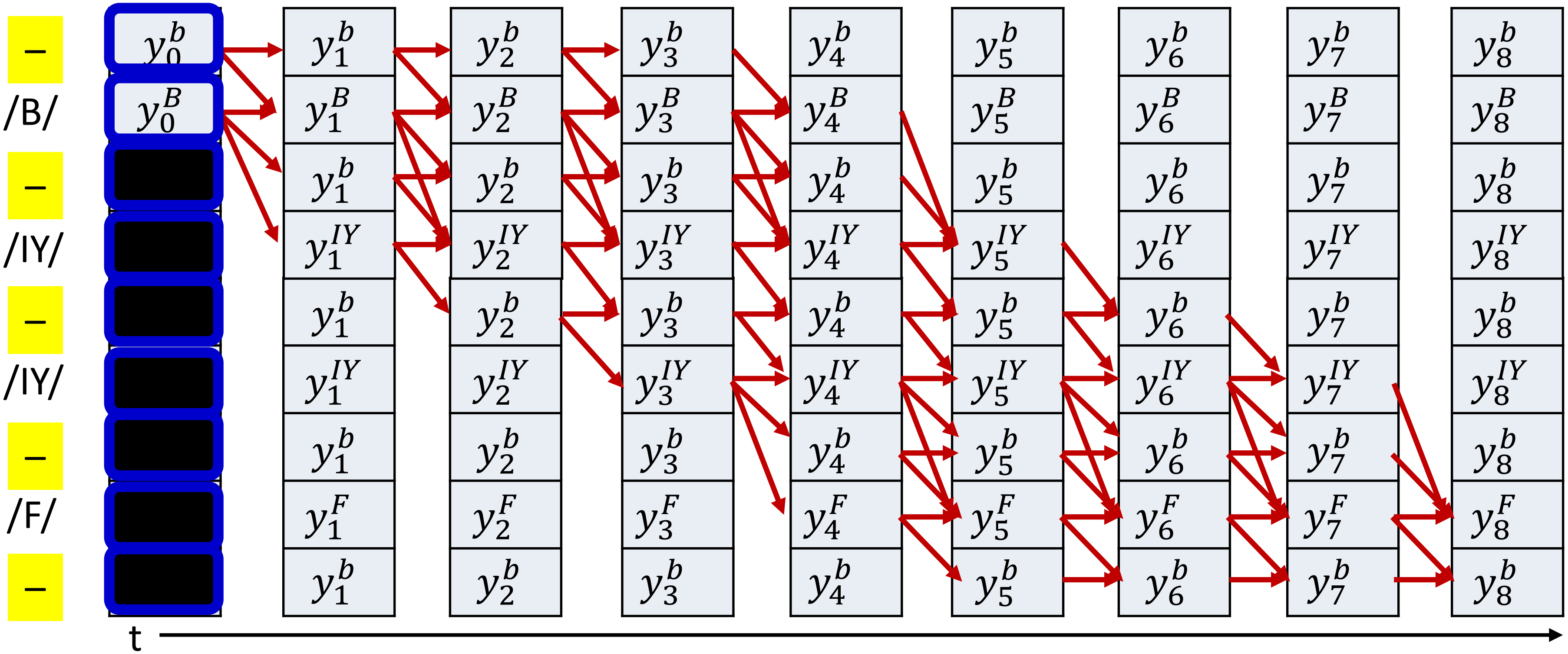
- The first and last column are allowed to also end at initial and final blanks

Composing the graph for training



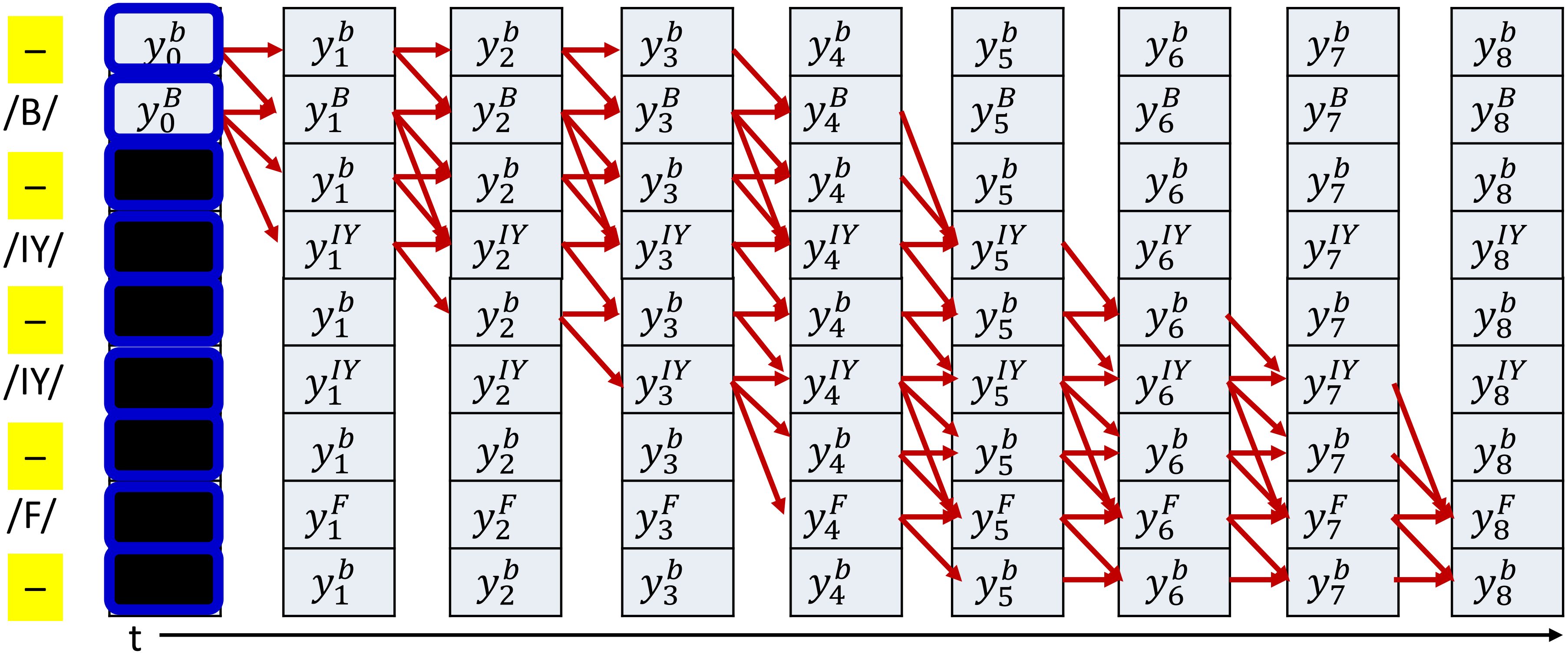
- The first and last column are allowed to also end at initial and final blanks
- Skips are permitted across a blank, but only if the symbols on either side are different
  - Because a blank is *mandatory between repetitions of a symbol* but *not required between distinct symbols*

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:

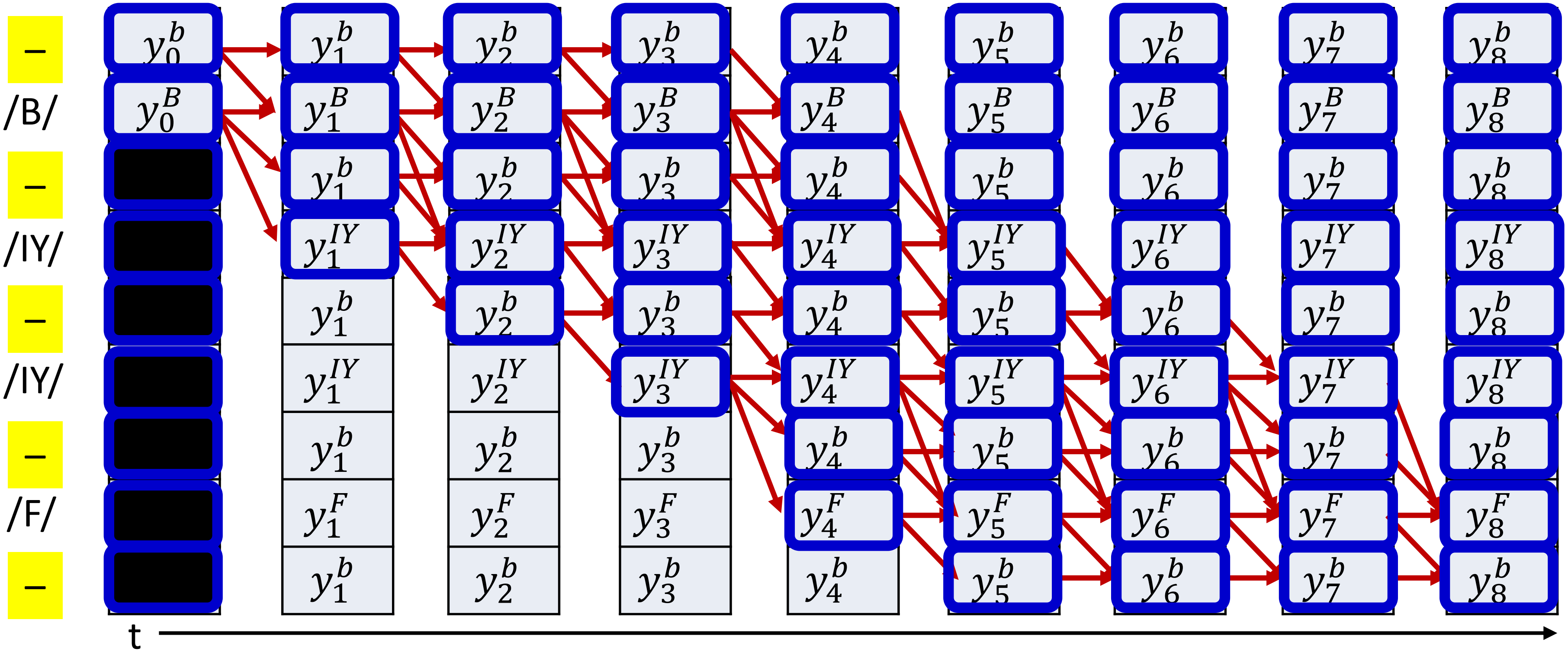
$$\alpha(t,r) = \big(\alpha(t-1,r) + \alpha(t-1,r-1)\big)y_t^{S(r)}$$

  - If $S(r) = " - "$ 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 Forward Algorithm



- Iteration:

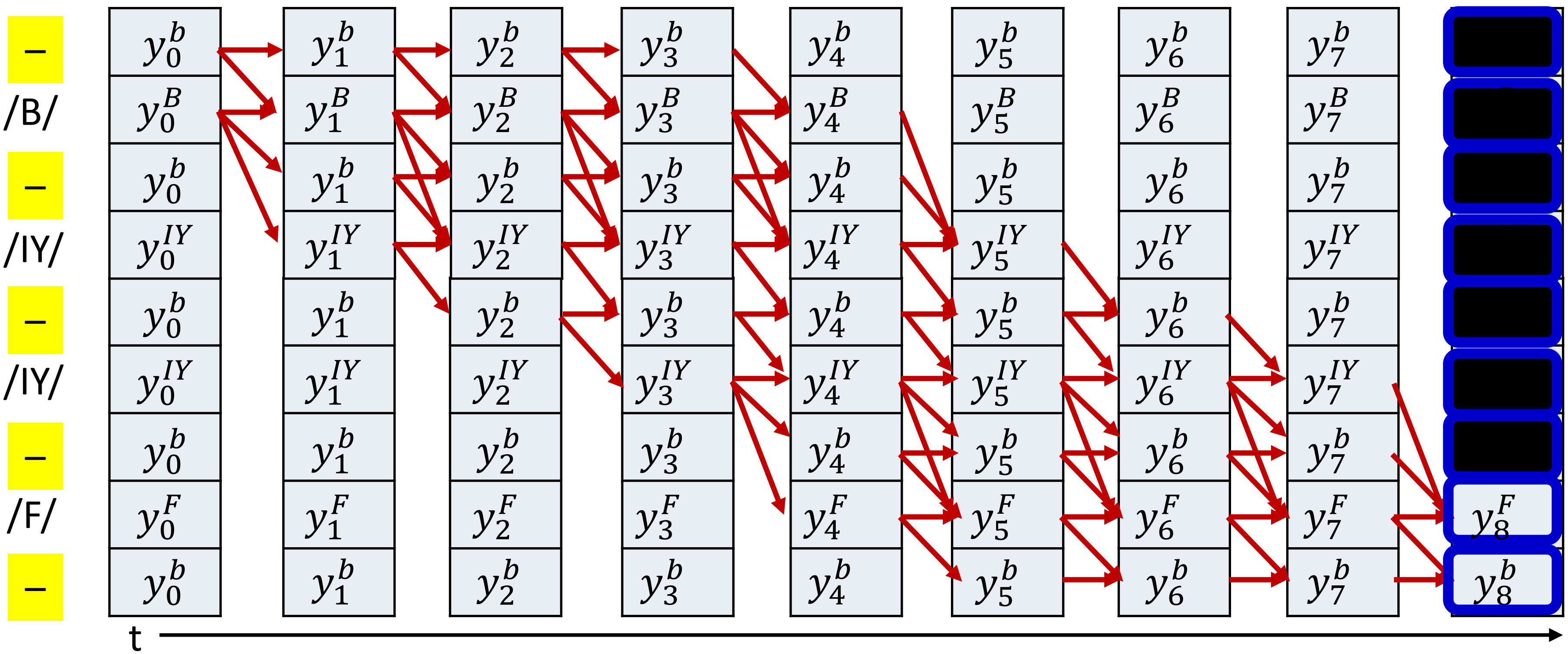
$$\alpha(t,r) = \big(\alpha(t-1,r) + \alpha(t-1,r-1)\big)y_t^{S(r)}$$

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

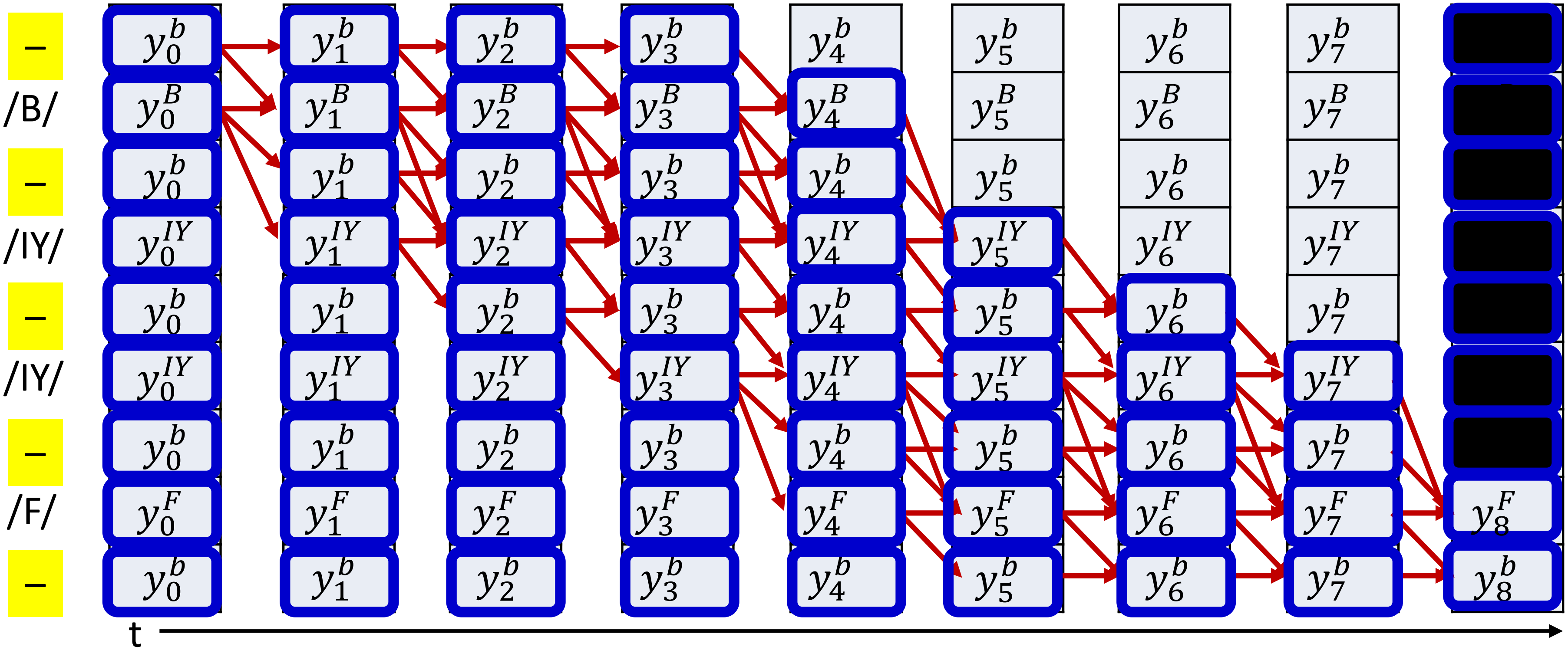
Modified Backward Algorithm



- Initialization:

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

Modified Backward Algorithm



- Iteration:

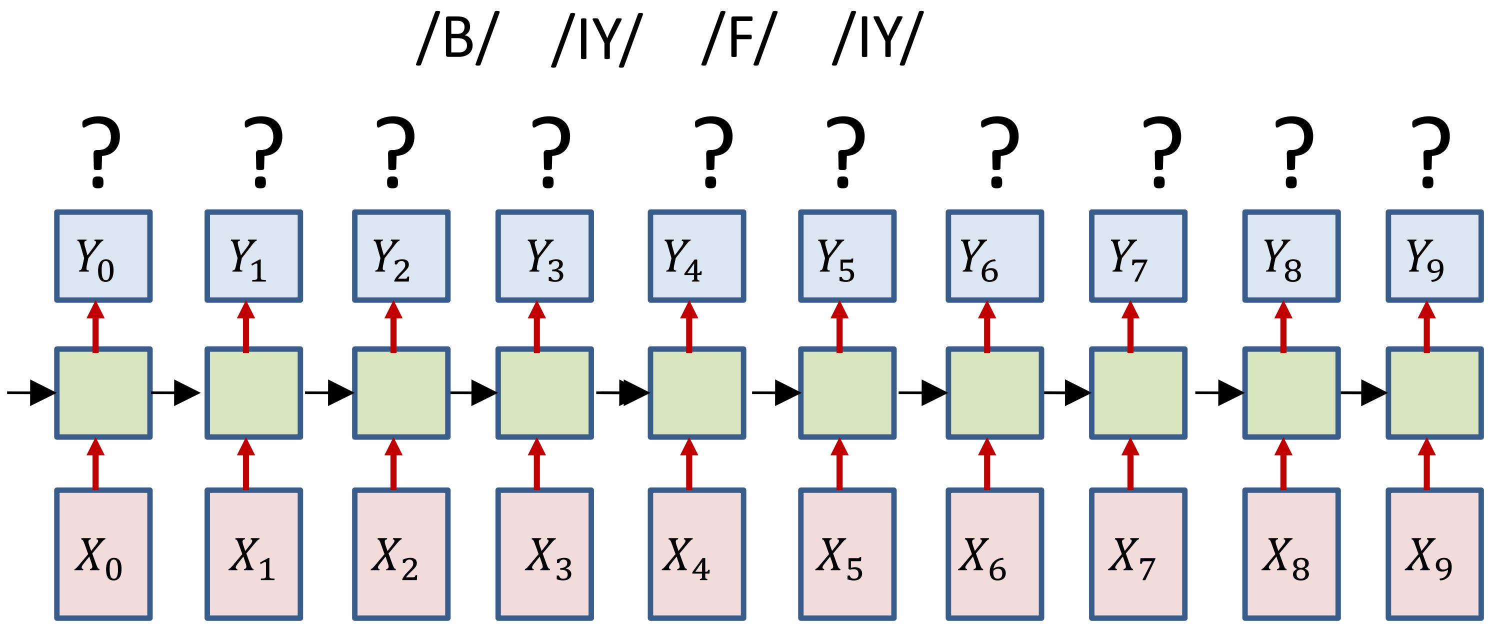
$$\beta(t,r) = \beta(t+1,r)y_t^{S(r)} + \beta(t+1,r+1)y_t^{S(r+1)}$$

  - 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

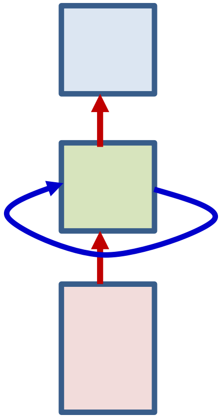
Overall training procedure for Seq2Seq with blanks



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

Overall training procedure

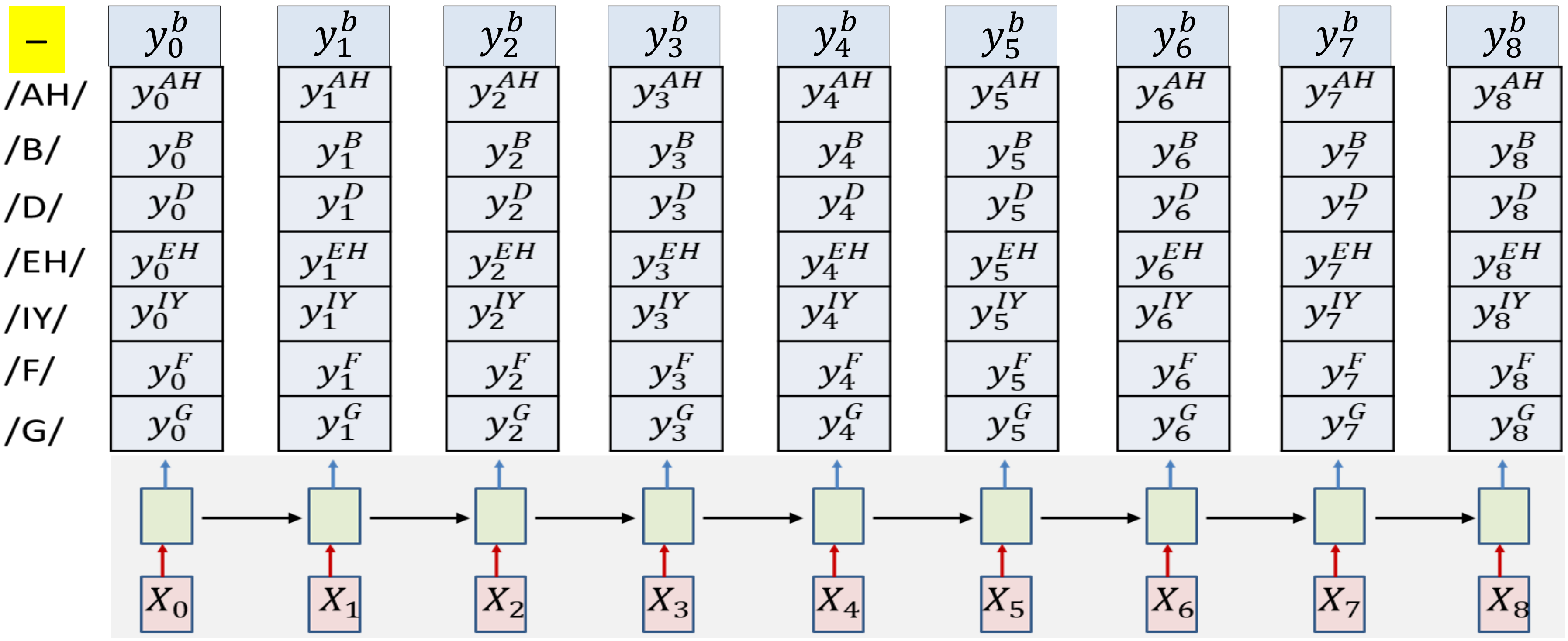
- **Step 1**: Setup the network
  - Typically many-layered LSTM



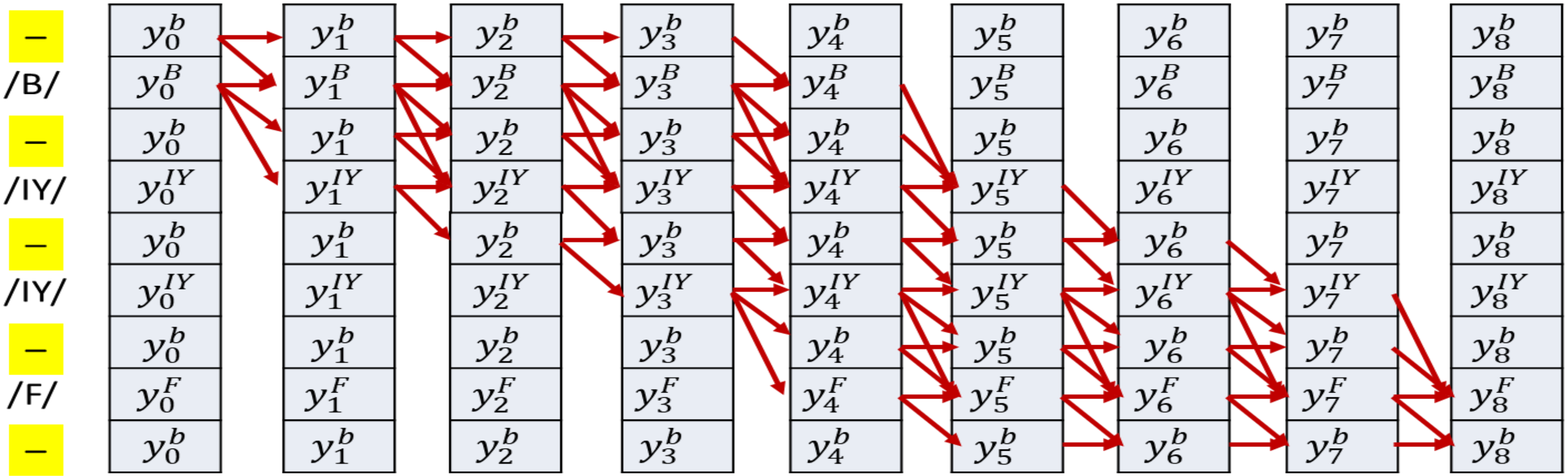
- **Step 2**: Initialize all parameters of the network
  - Include a “blank” symbol in vocabulary

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

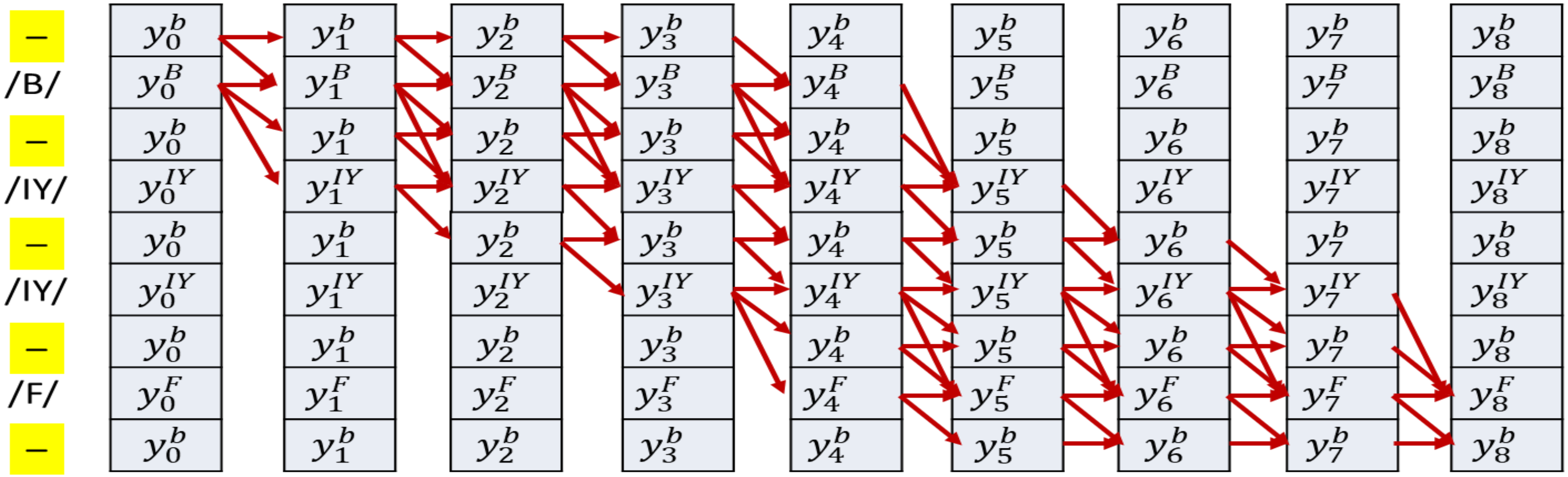


Overall training: Backward pass



- 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

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

$$\frac{dDIV}{dy_t^l} = -\sum_{r\,:S(r)=l} \frac{d}{dy_t^{S(r)}} \left(\frac{\alpha(t,r)\beta(t,r)}{\sum_{r'} \alpha(t,r')\beta(t,r')} \log y_t^{S(r)} \right)$$

- **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

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

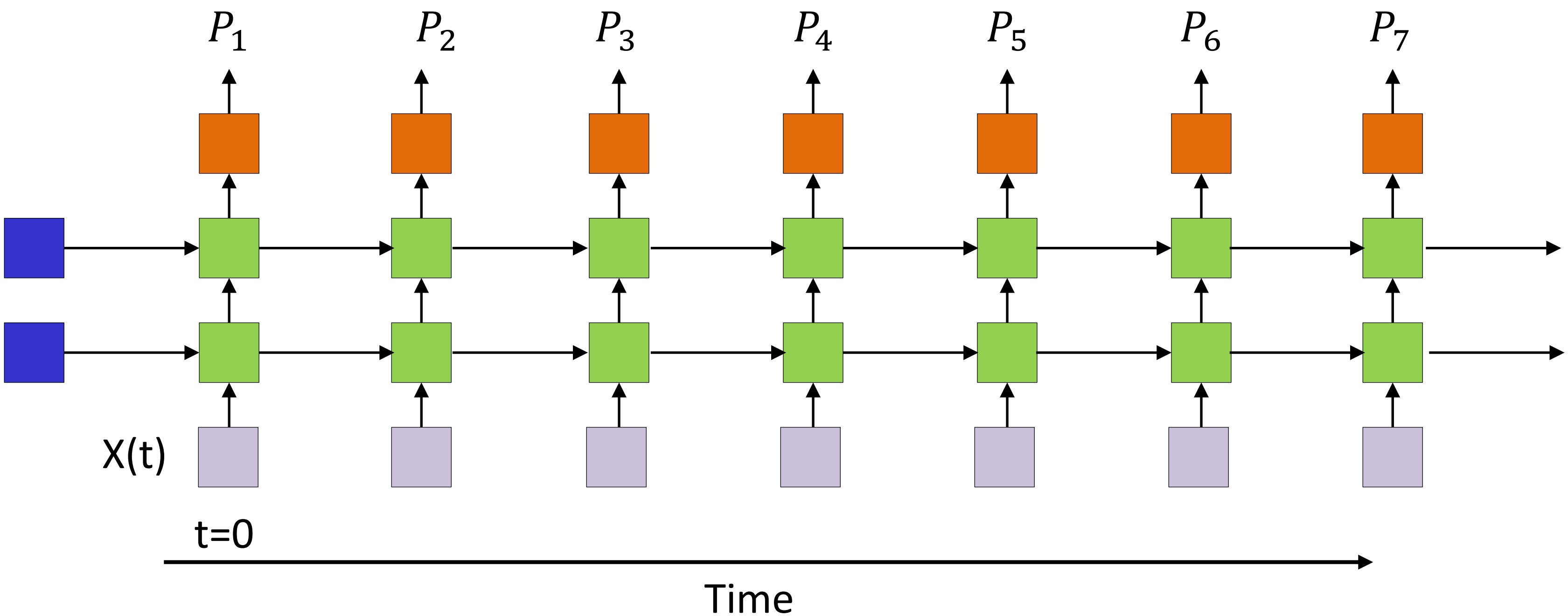
- Possible variants:
  - Symbols partitioned into two or more sequential subunits
    - No blanks are required, since subunits must be visited in order
  - Symbol-specific blanks
    - Doubles the “vocabulary”
  - CTC can use *bidirectional* recurrent nets
    - And frequently does
  - Other variants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

Most common CTC applications

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

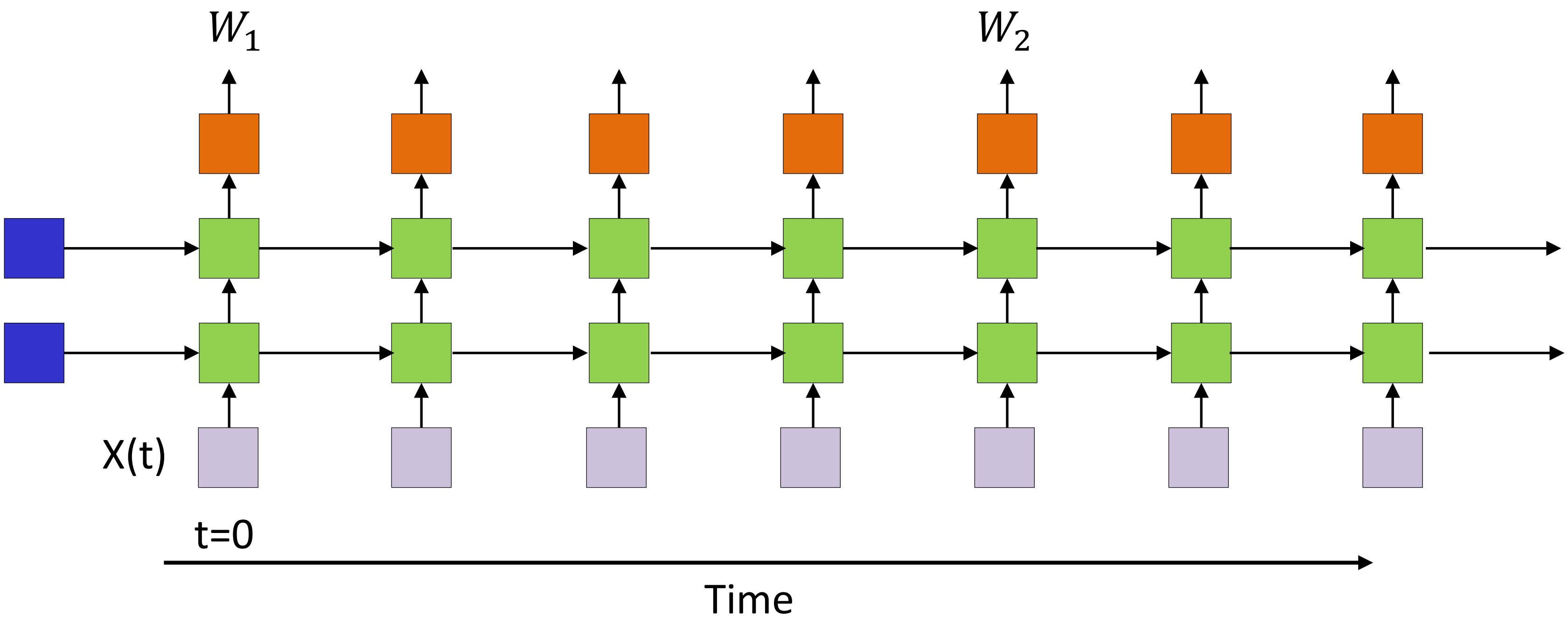
- Handwriting recognition

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

- Will cover on Friday!