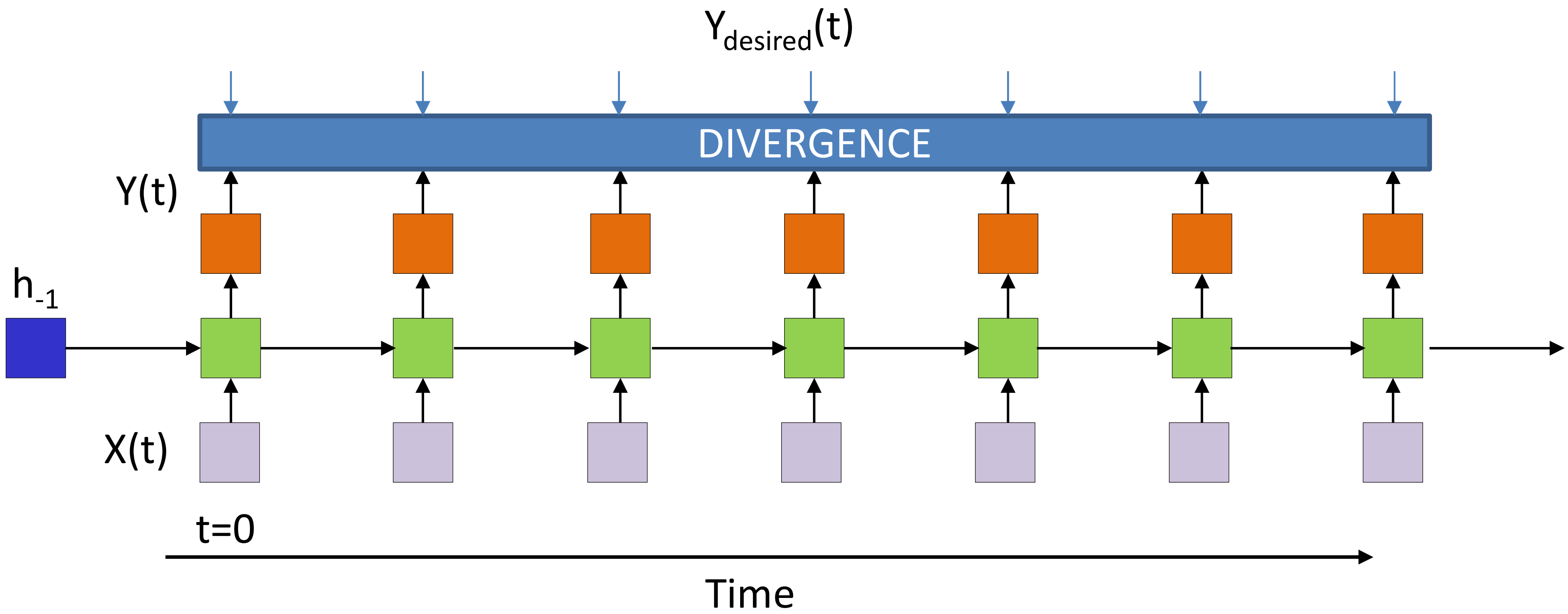
Deep Learning

Recurrent Networks: Part 3

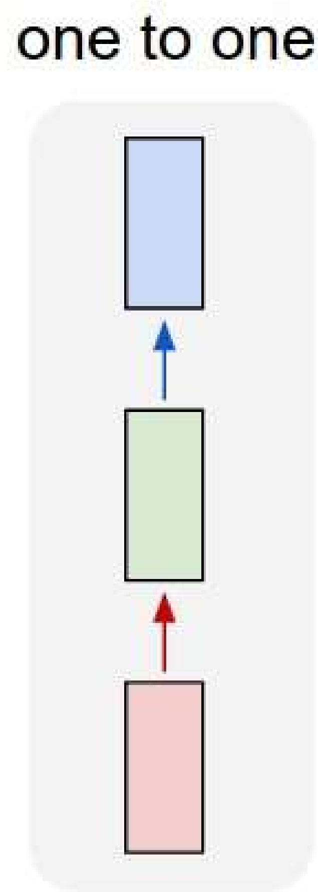
Fall 2022

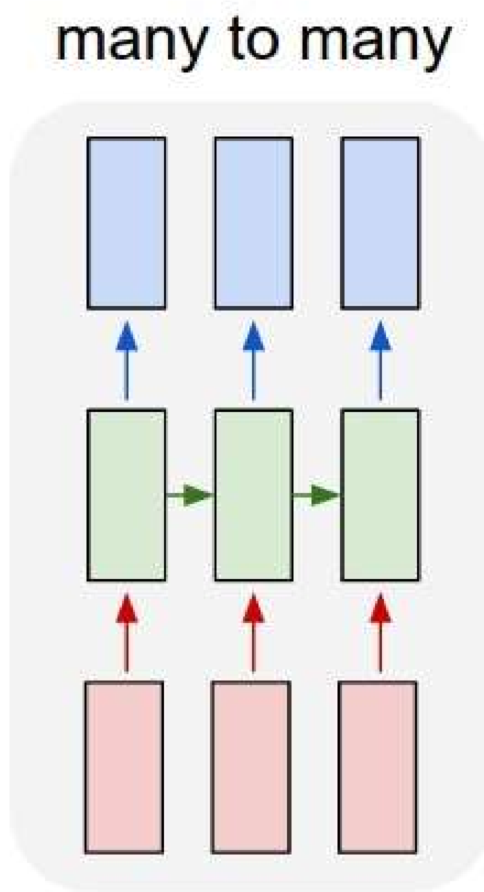
Story so far



- Recurrent structures can be trained by minimizing the divergence between the ***sequence*** of outputs and the ***sequence*** of desired outputs
  - Through gradient descent and backpropagation
- The challenge: Defining this divergence
  - Inputs and outputs may not be time aligned or even synchronous

Variants of recurrent nets

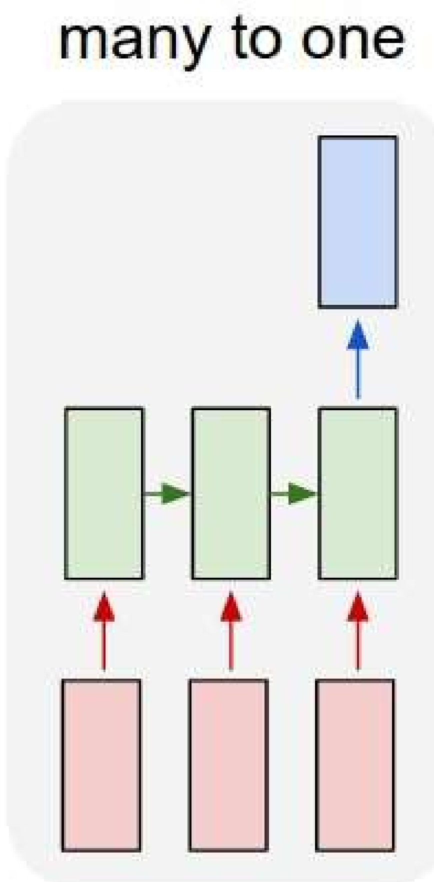


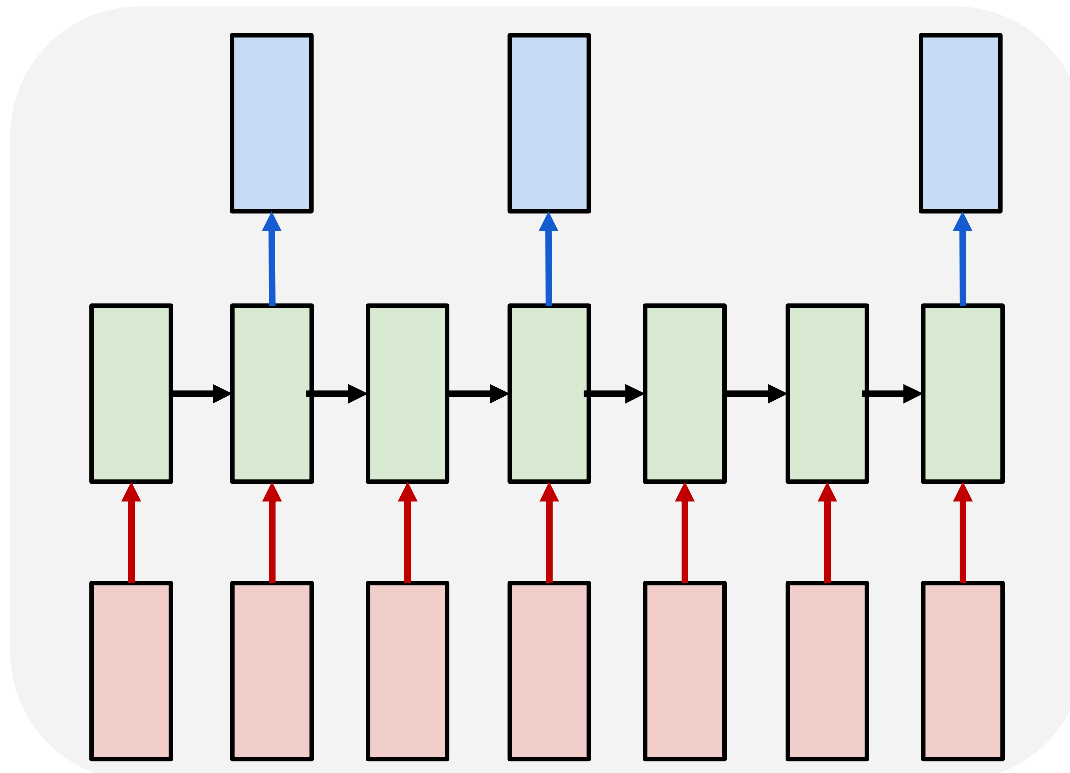


Images from Karpathy

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

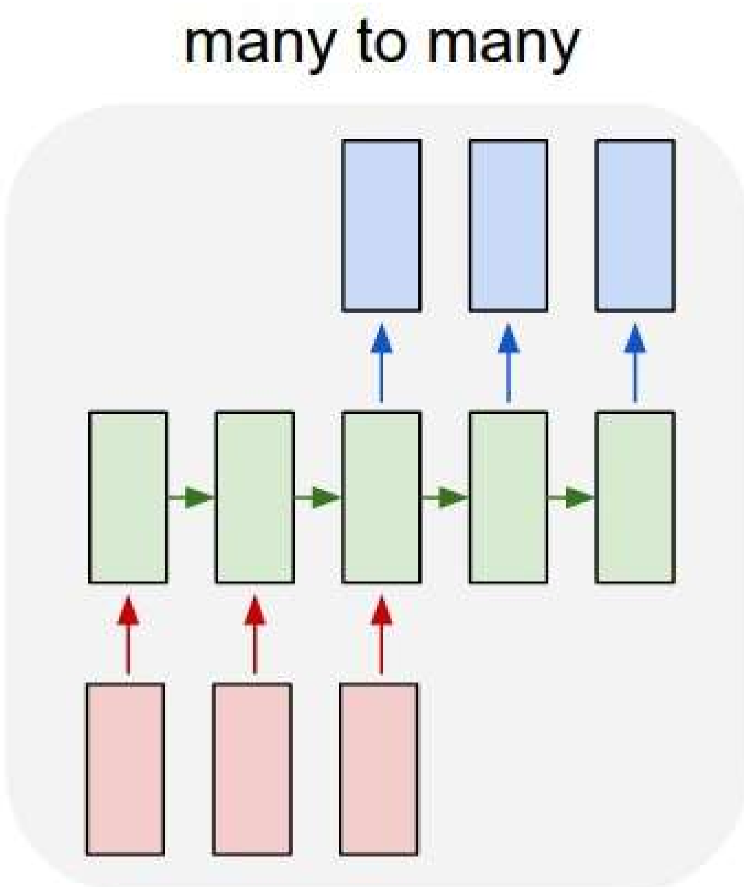
Variants of recurrent nets

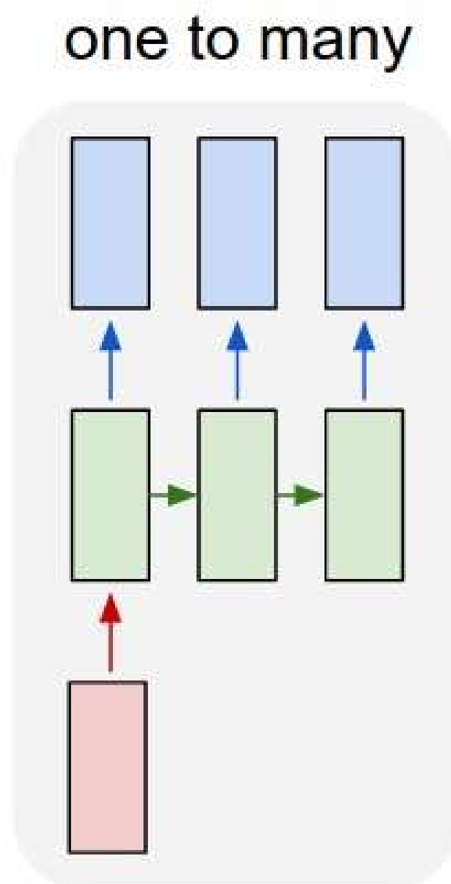




- Sequence classification: Classifying a full input sequence
  - E.g isolated word/phrase recognition
- Order synchronous , time asynchronous sequence-to-sequence generation
  - E.g. speech recognition
  - Exact location of output is unknown a priori

More variants

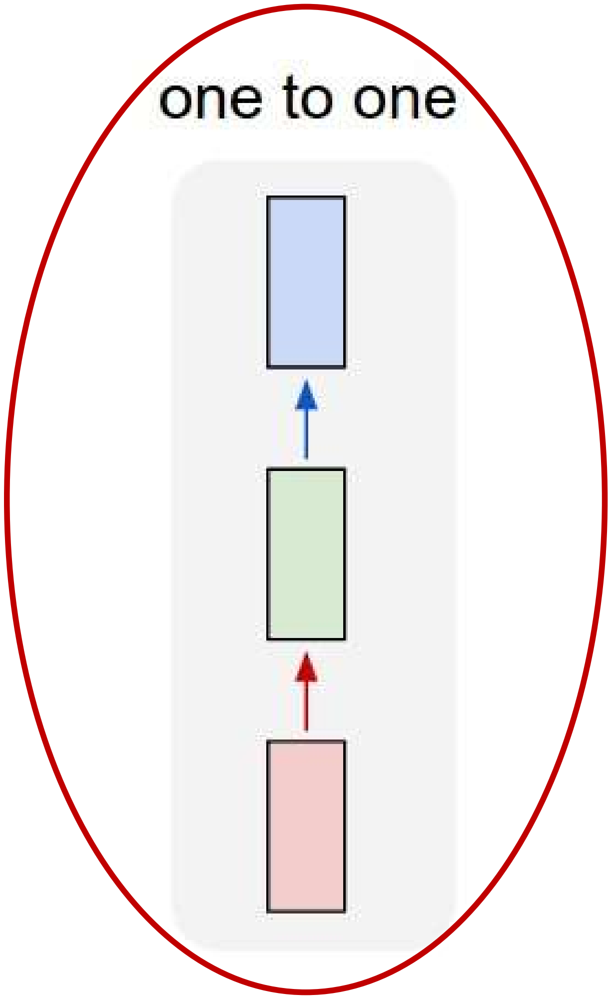


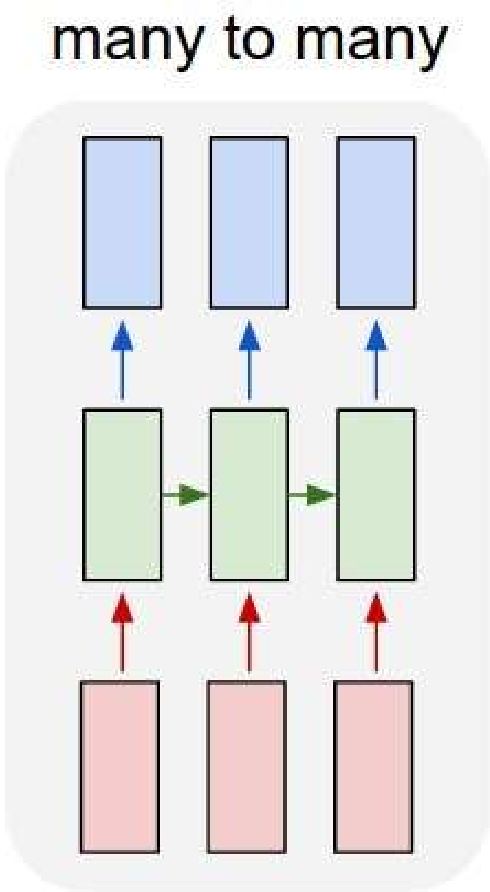


Images from Karpathy

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

Variants of recurrent nets

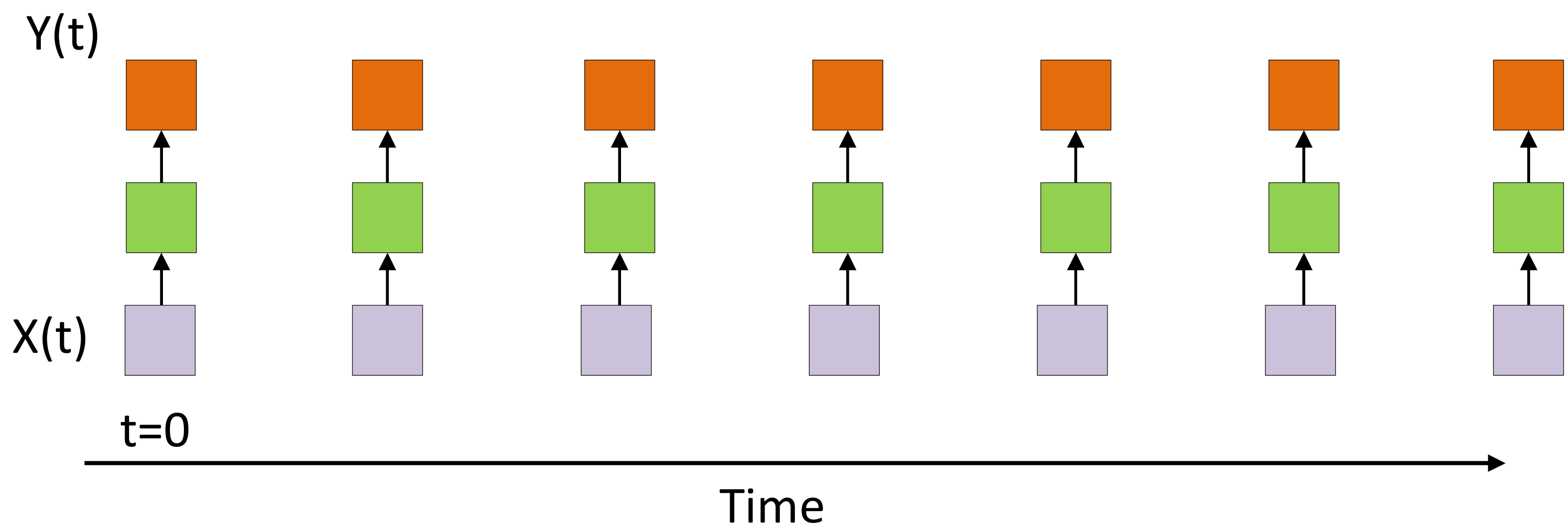




Images from Karpathy

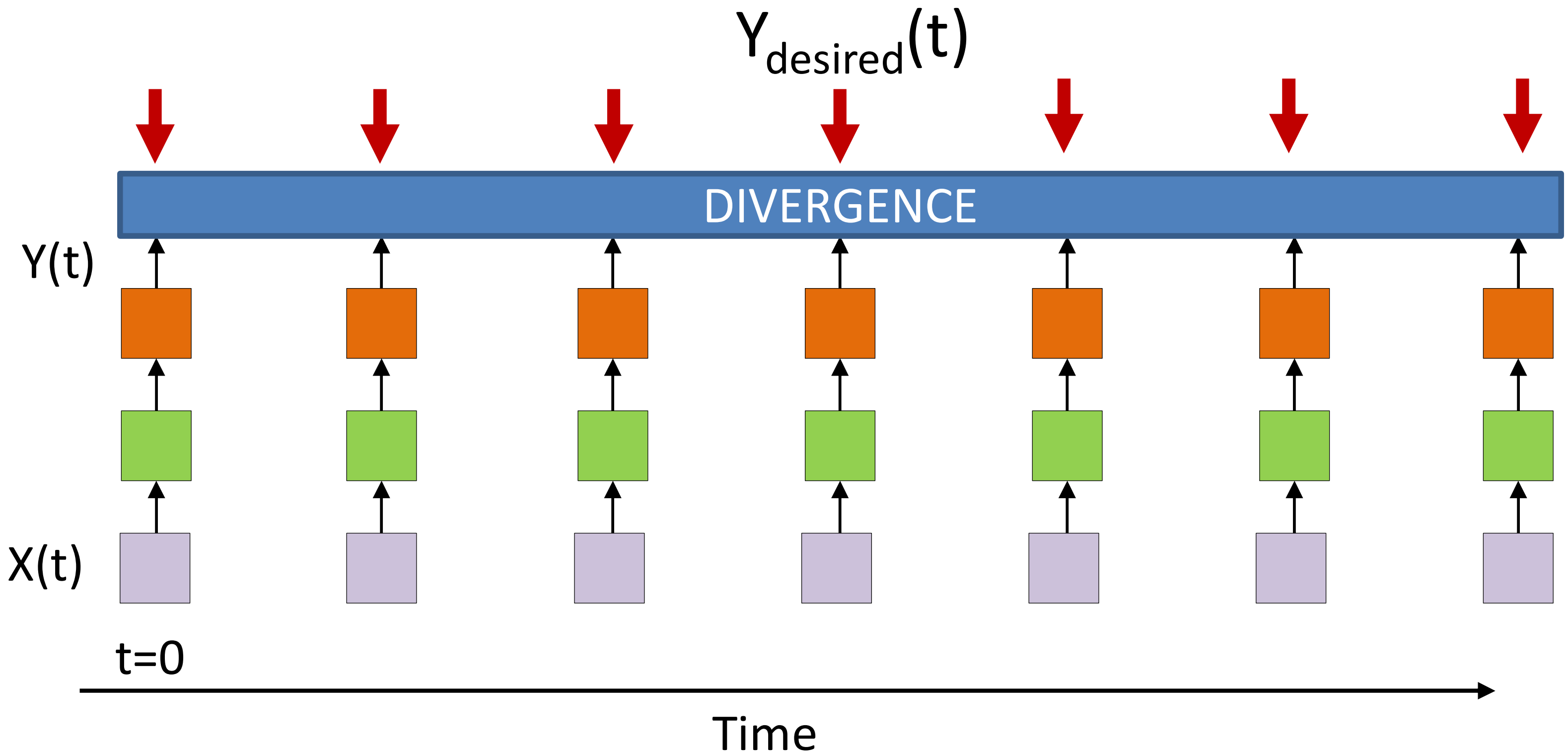
- Conventional MLP
- Time-synchronous outputs
  - E.g. part of speech tagging

This is a regular MLP



- No recurrence
  - Exactly as many outputs as inputs
  - The output at time t is unrelated to the output at $t' \neq t$.

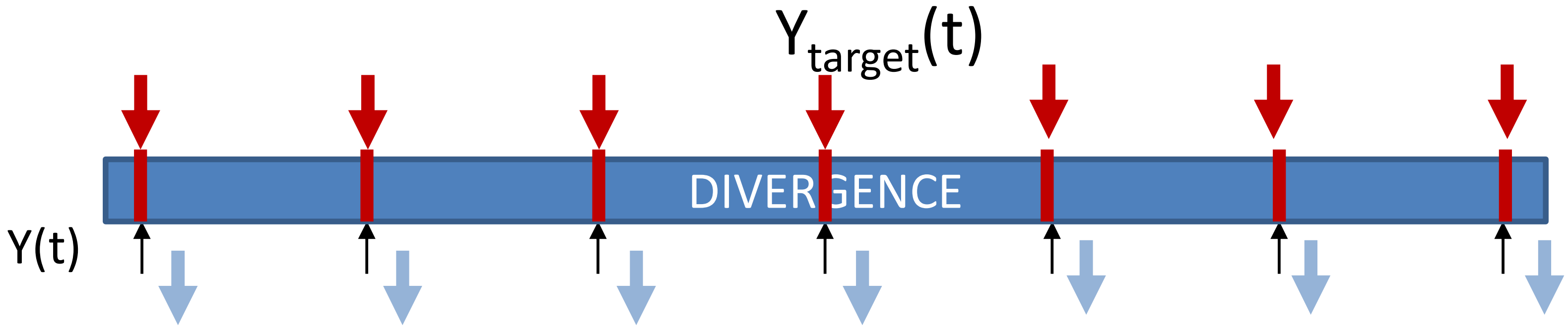
Learning in a regular MLP for series



- In the context of analyzing time series, the divergence to minimize is still the divergence between two series
  - Must be differentiable w.r.t every Y(t)

- In this setting: One-to-one correspondence between actual and target outputs
- Common assumption: Total divergence is the sum of *local* divergences at individual times
  - Simplifies model and maths

"Series MLP" as a regular MLP



- Gradient backpropagated at each time

$$\nabla_{Y(t)} Div\big(Y_{target}(1 \dots T), Y(1 \dots T)\big)$$

- Common assumption: One-to-one correspondence

$$Div\big(Y_{target}(1 \dots T), Y(1 \dots T)\big) = \sum_t Div\big(Y_{target}(t), Y(t)\big)$$

$$\nabla_{Y(t)} Div\big(Y_{target}(1 \dots T), Y(1 \dots T)\big) = \nabla_{Y(t)} Div\big(Y_{target}(t), Y(t)\big)$$

  – This is further backpropagated to update weights etc

Typical Divergence for classification: $Div\big(Y_{target}(t), Y(t)\big) = KL(Y_{target}(t), Y(t))$

Poll 1

- @, @

Conventional MLPs too can be used to model sequences, True or false

- True
- False

When we use conventional MLPs to model sequences, the sequence nature of the problem is captured through the divergence, which is now computed between the output sequence and the desired output sequence, true or false

- True
- False

Poll 1

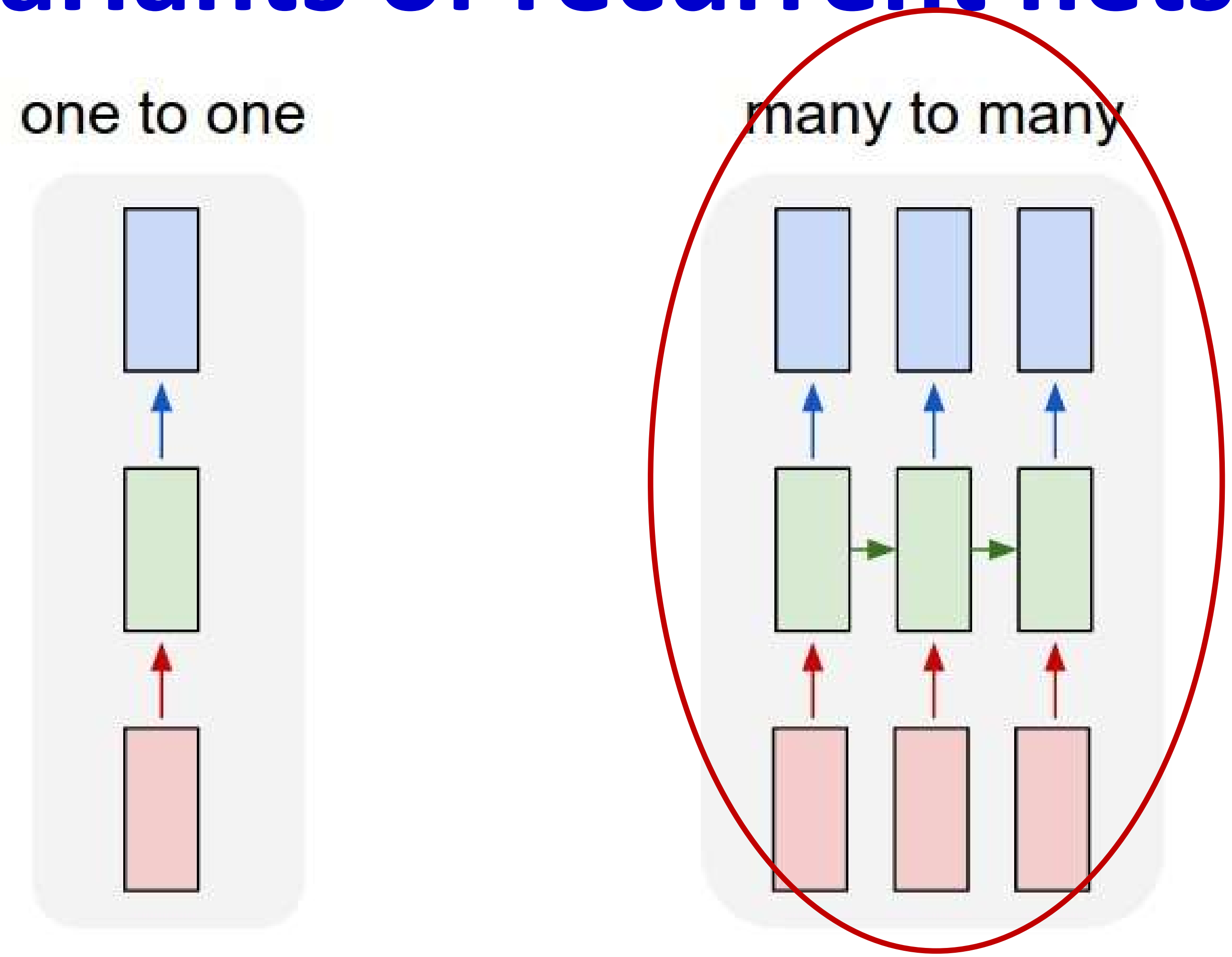
Conventional MLPs too can be used to model sequences, True or false

- **True**
- False

When we use conventional MLPs to model sequences, the sequence nature of the problem is captured through the divergence, which is now computed between the output *sequence* and the desired output *sequence*, true or false

- **True**
- False

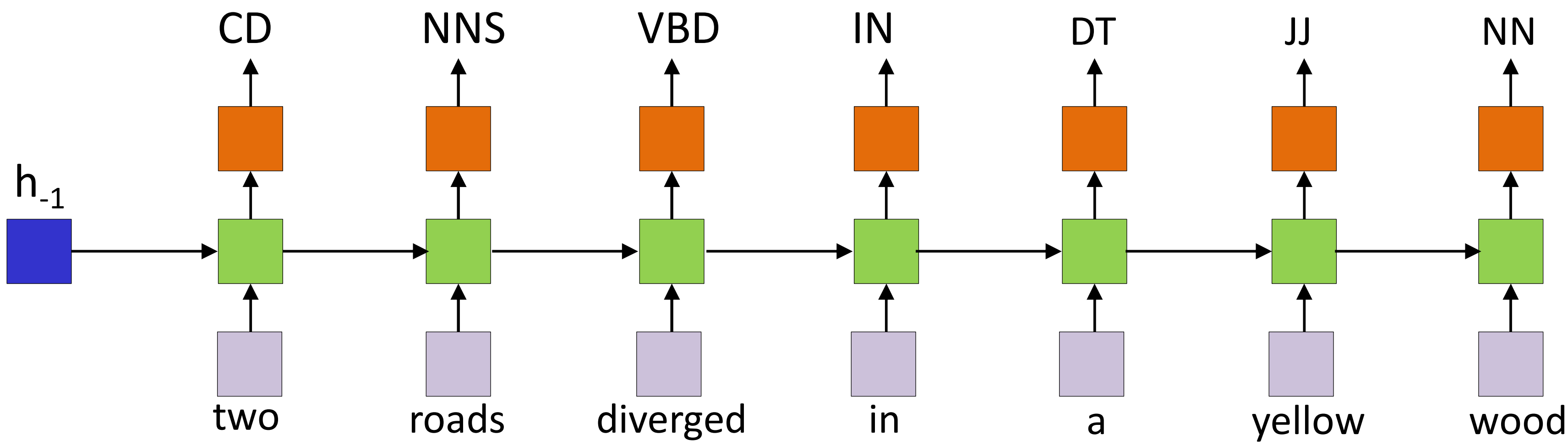
Variants of recurrent nets



Images from Karpathy

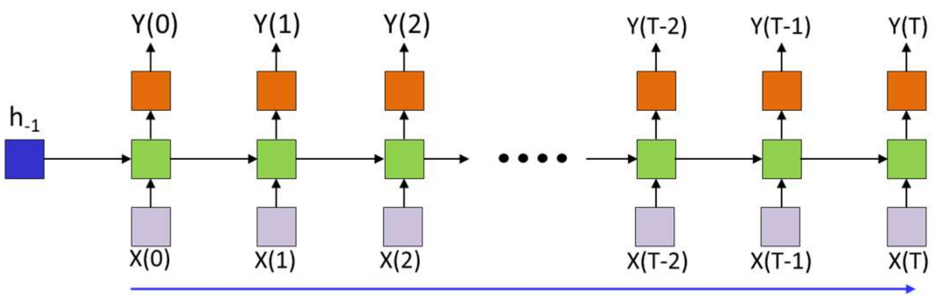
- Conventional MLP
- Time-synchronous outputs
  - E.g. part of speech tagging

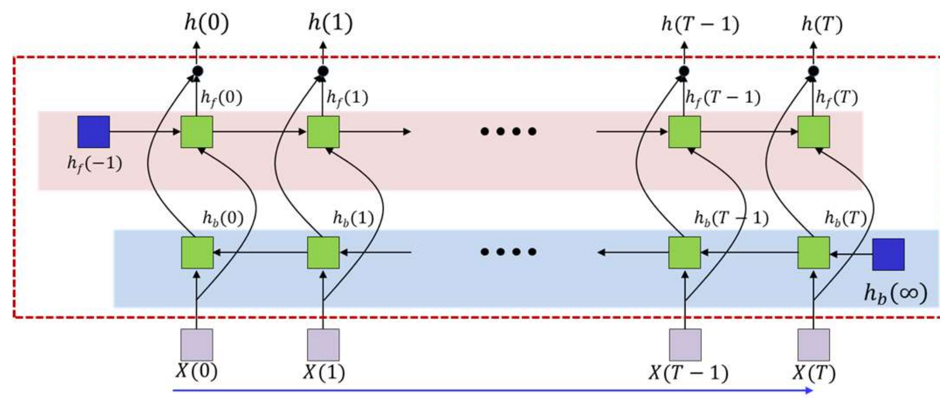
Time synchronous network



- Network produces one output for each input
  - With one-to-one correspondence
  - E.g. Assigning grammar tags to words
    - May require a bidirectional network to consider both past and future words in the sentence

Time-synchronous networks: Inference



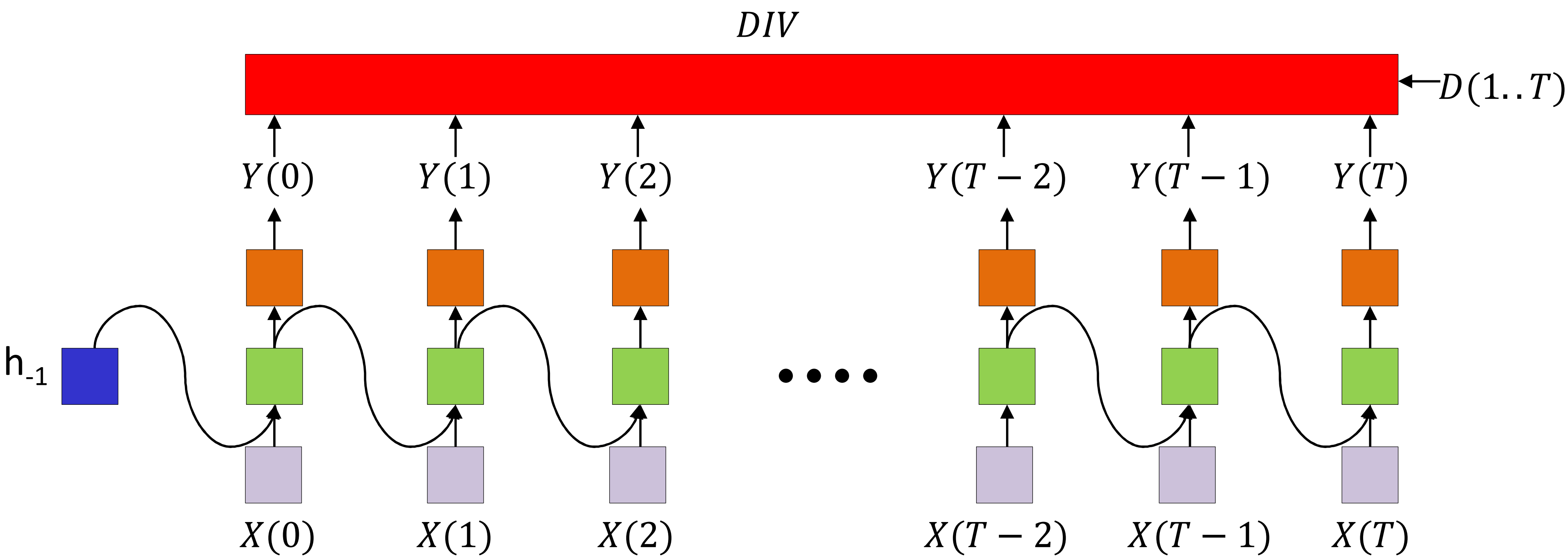


- One sided network: Process input left to right and produce output after each input

- Bi-directional network: Process input in both directions

- In all cases, there is an output for every input with exact one-to-one time-synchronous correspondence
  - Will continue to assume unidirectional models for explanations

Back Propagation Through Time

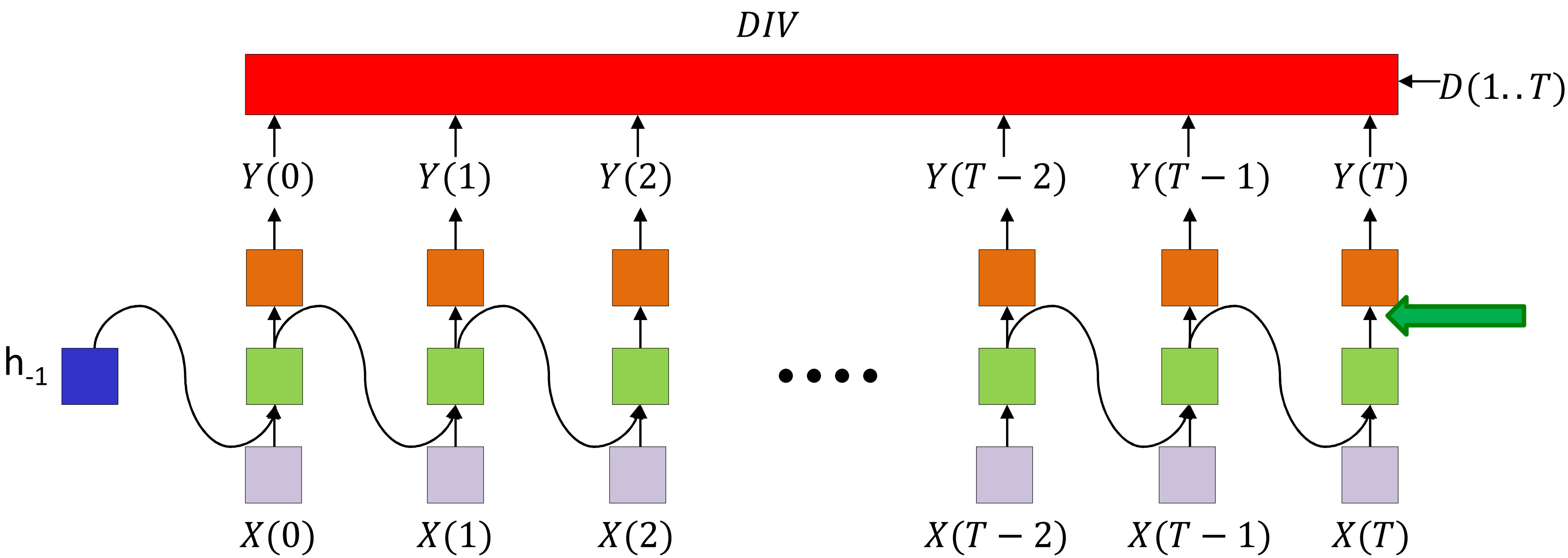


- Train given a set of input-target output pairs that are time synchronous
  - $(\mathbf{X}_i, \mathbf{D}_i)$, where $\mathbf{X}_i = X_{i,0}, \ldots, X_{i,T}$, $\mathbf{D}_i = D_{i,0}, \ldots, D_{i,T}$

- The divergence computed is between the *sequence of outputs* by the network and the *desired sequence of outputs*

$$Div\left(Y_{target}(1 \ldots T), Y(1 \ldots T)\right)$$

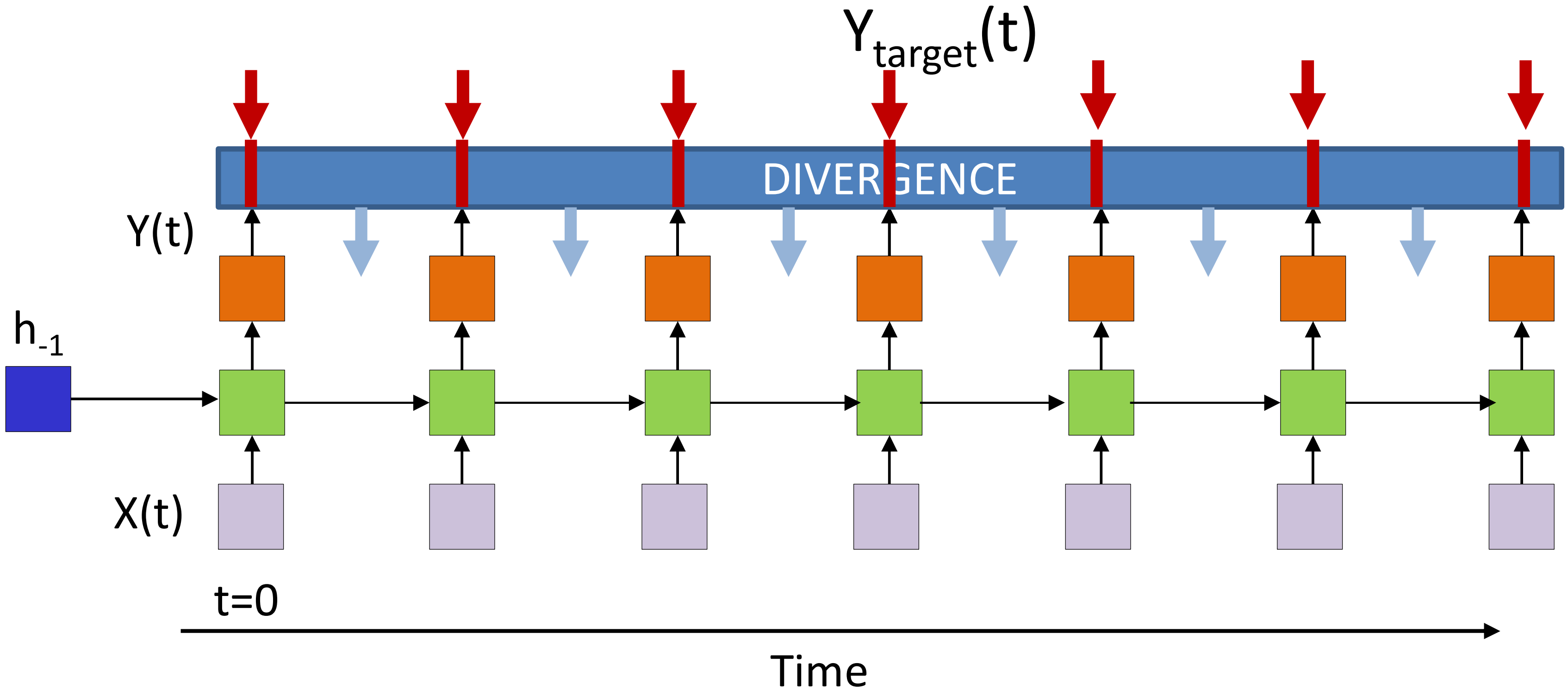
Back Propagation Through Time



First step of backprop: Compute $\nabla_{Y(t)} DIV$ for all t

- The key component is the computation of this derivative!!
- This depends on the definition of "DIV"

BPTT: Time-synchronous recurrence



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

$$Div(Y_{target}(1 \ldots T), Y(1 \ldots T)) = \sum_t Div(Y_{target}(t), Y(t))$$

$$\nabla_{Y(t)} Div(Y_{target}(1 \ldots T), Y(1 \ldots T)) = \nabla_{Y(t)} Div(Y_{target}(t), Y(t))$$

Typical Divergence for classification: $Div(Y_{target}(t), Y(t)) = KL(Y_{target}(t), Y(t))$

Poll 2

- @

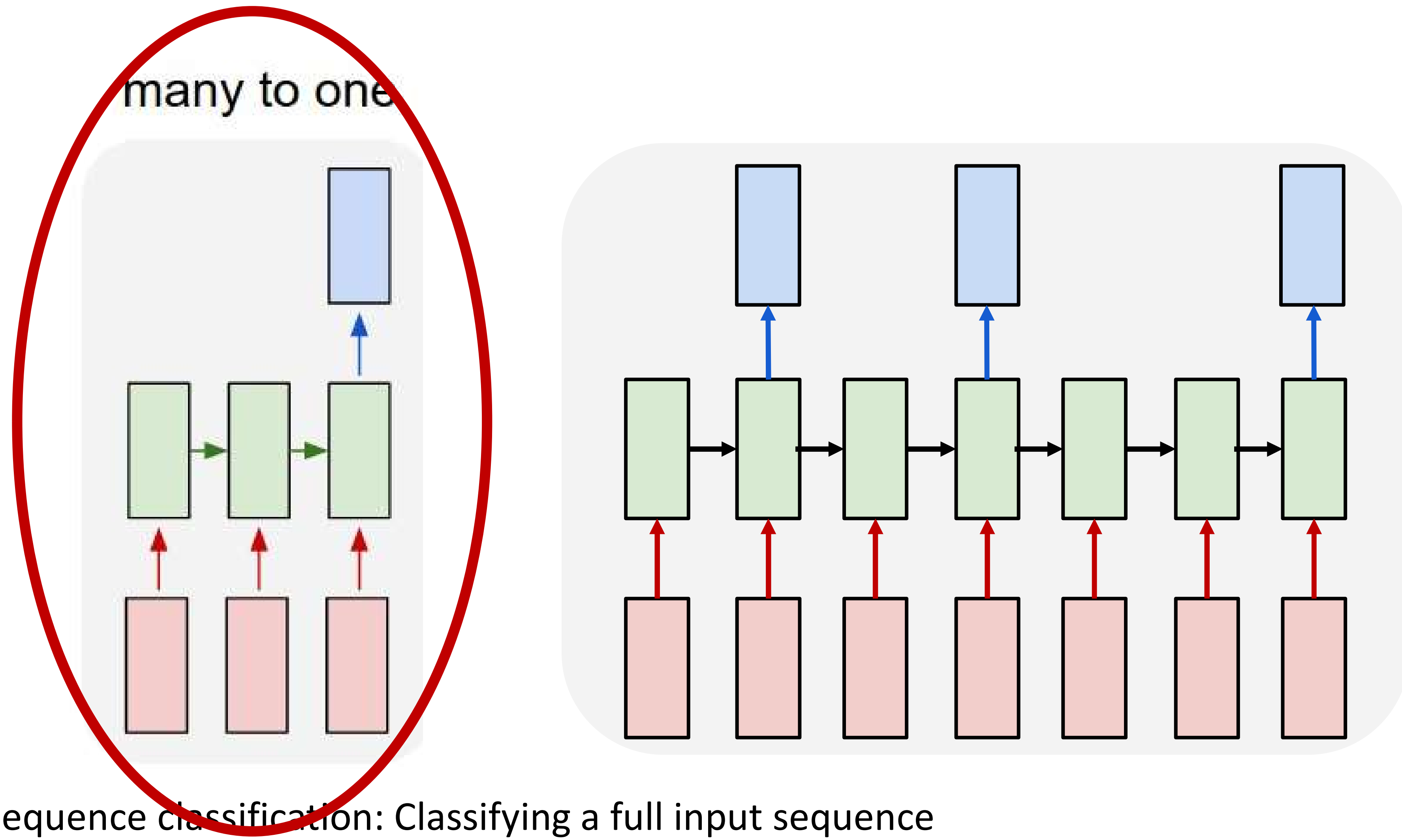
Select all that are true about time-synchronous RNNs
- There is one output corresponding to every input
- They can only be unidirectional, i.e. either forward recursion or backward recursion, but not both.
- The divergence between true and desired outputs can have an additive contribution from the output at each time.

Poll 2

Select all that are true about time-synchronous RNNs

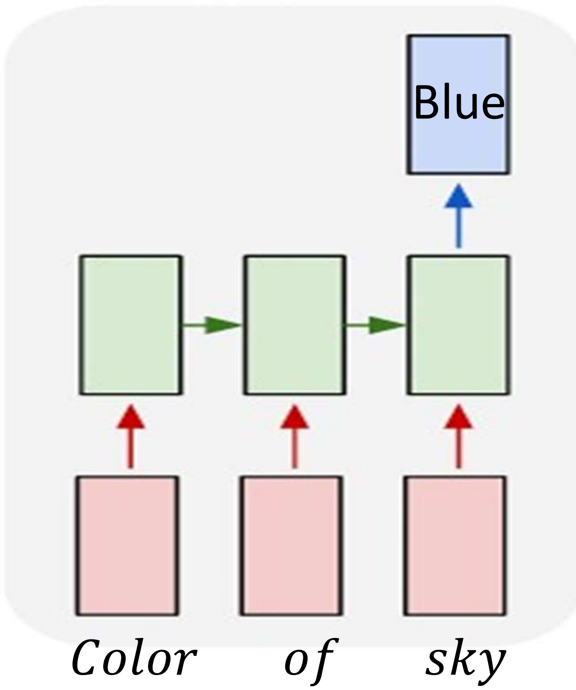
- **There is one output corresponding to every input**
- They can only be unidirectional, i.e. either forward recursion or backward recursion, but not both.
- **The divergence between true and desired outputs can have an additive contribution from the output at each time.**

Variants of recurrent nets



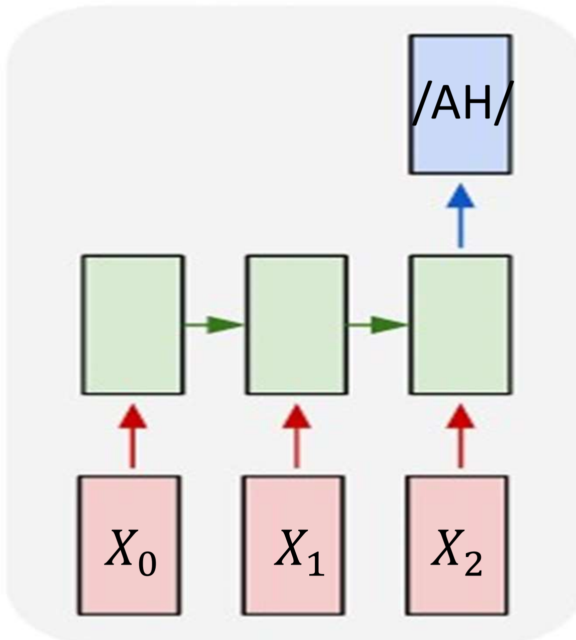
- Sequence classification: Classifying a full input sequence
  - E.g phoneme recognition
- Order synchronous , time asynchronous sequence-to-sequence generation
  - E.g. speech recognition
  - Exact location of output is unknown a priori

Example..



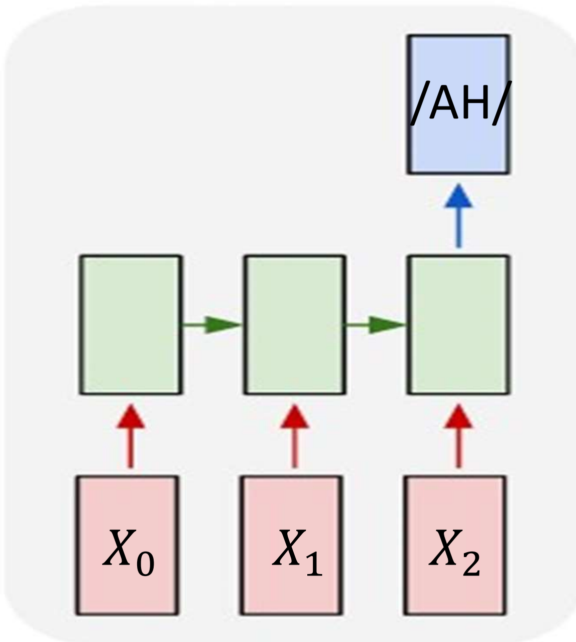
- Question answering
- Input : Sequence of words
- Output: Answer at the end of the question

Example..



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

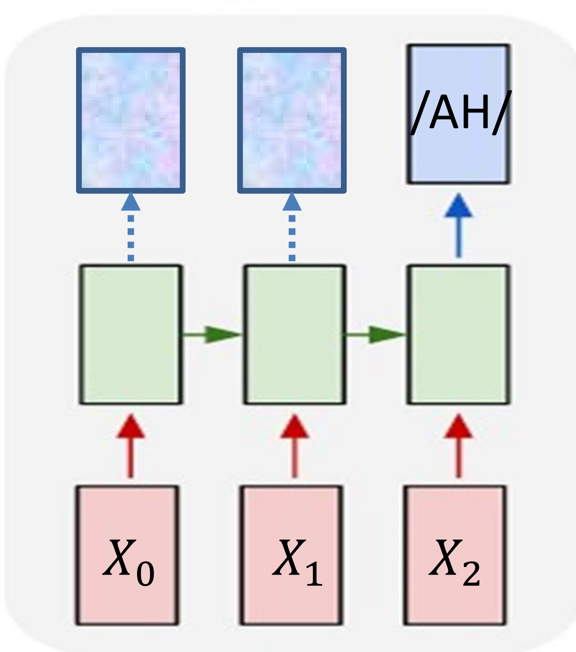
Inference: Forward pass



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

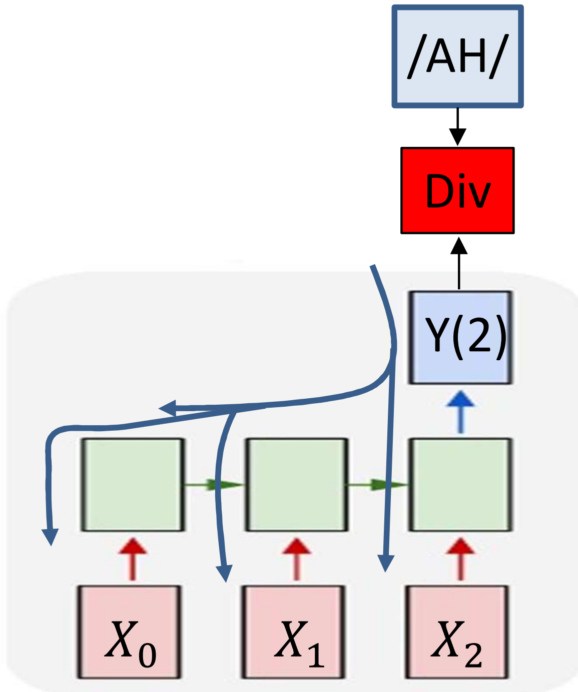
- But what about at *intermediate stages?*

Forward pass



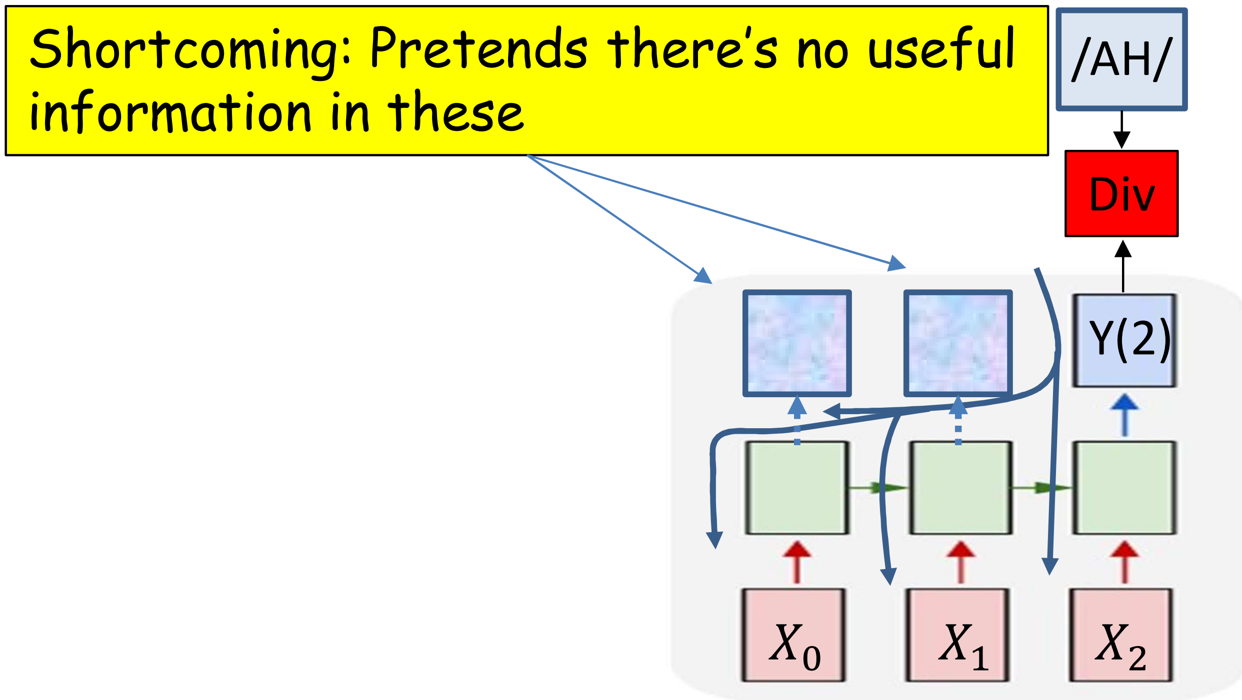
- Exact input sequence provided
  - Output generated when the last vector is processed
    - Output is a probability distribution over phonemes
- Outputs are actually produced for *every* input
  - We only *read* it at the end of the sequence

Training



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

Training

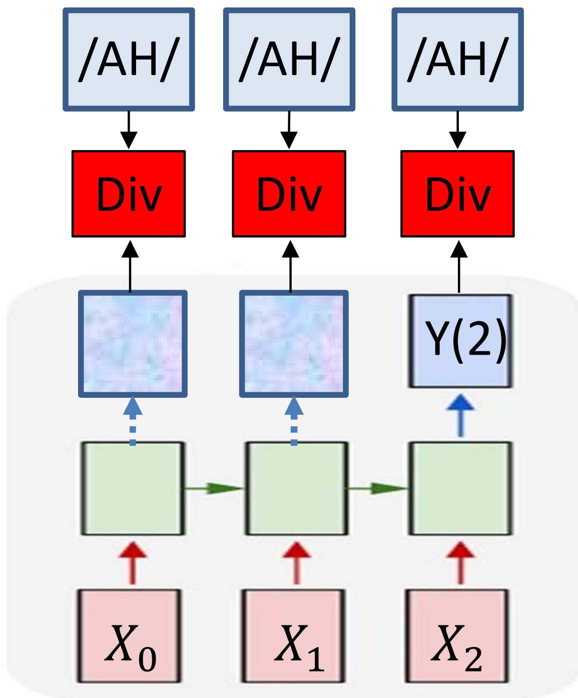


- The Divergence is only defined at the final input
  - $DIV(Y_{target}, Y) = KL(Y(T), Phoneme)$
- This divergence must propagate through the net to update all parameters

Training

Fix: Use these outputs too.

These too must ideally point to the correct phoneme



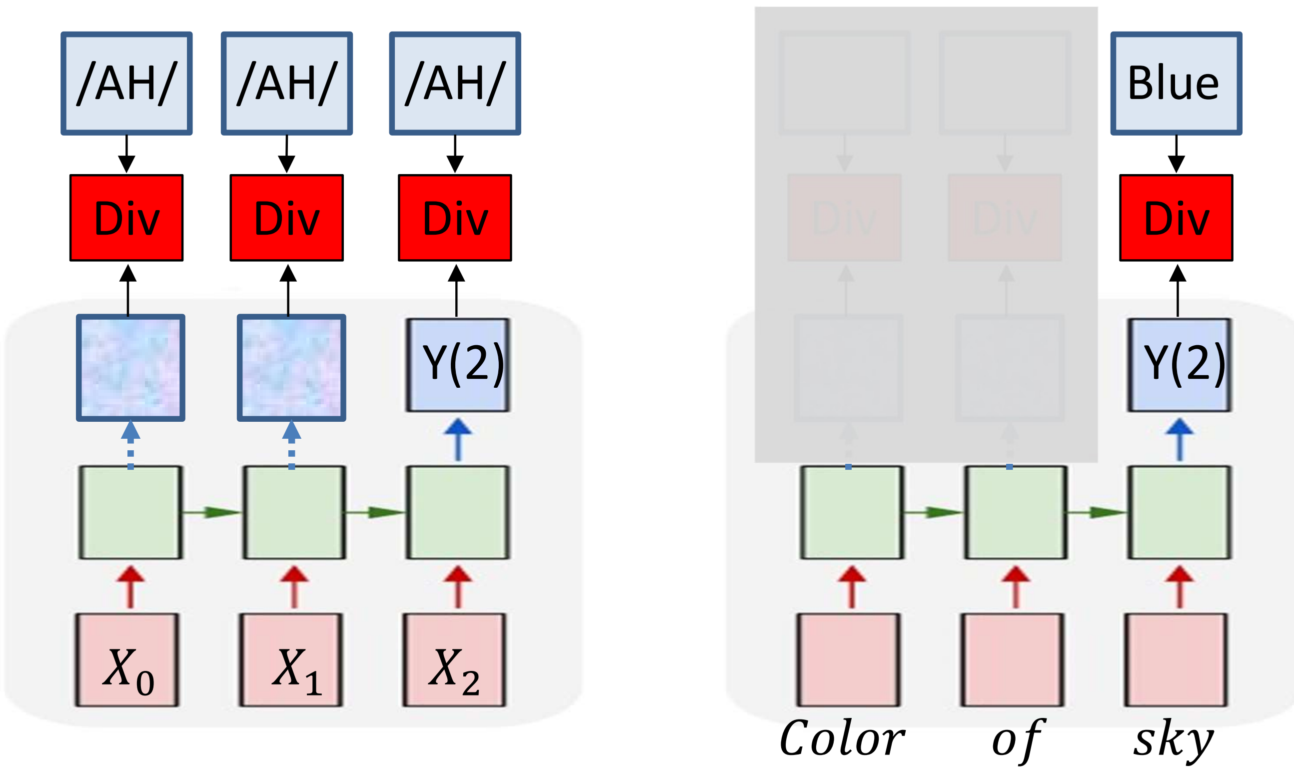
- Exploiting the untagged inputs: assume the same output for the entire input
- Define the divergence everywhere

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

Training

Fix: Use these outputs too.

These too must ideally point to the correct phoneme

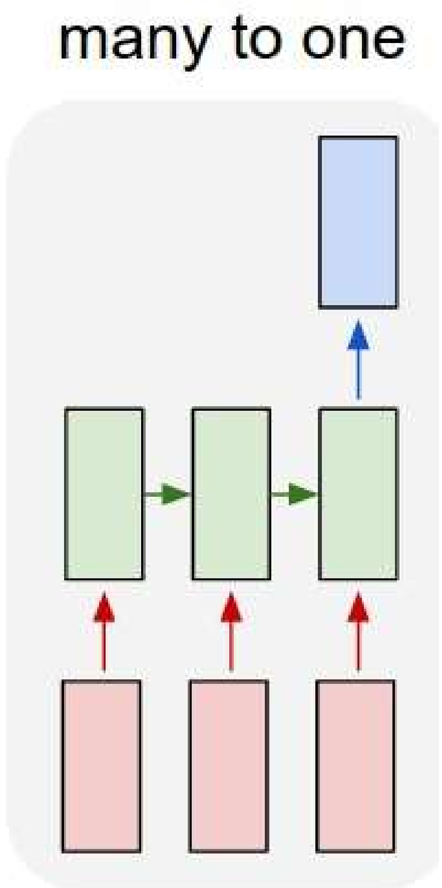


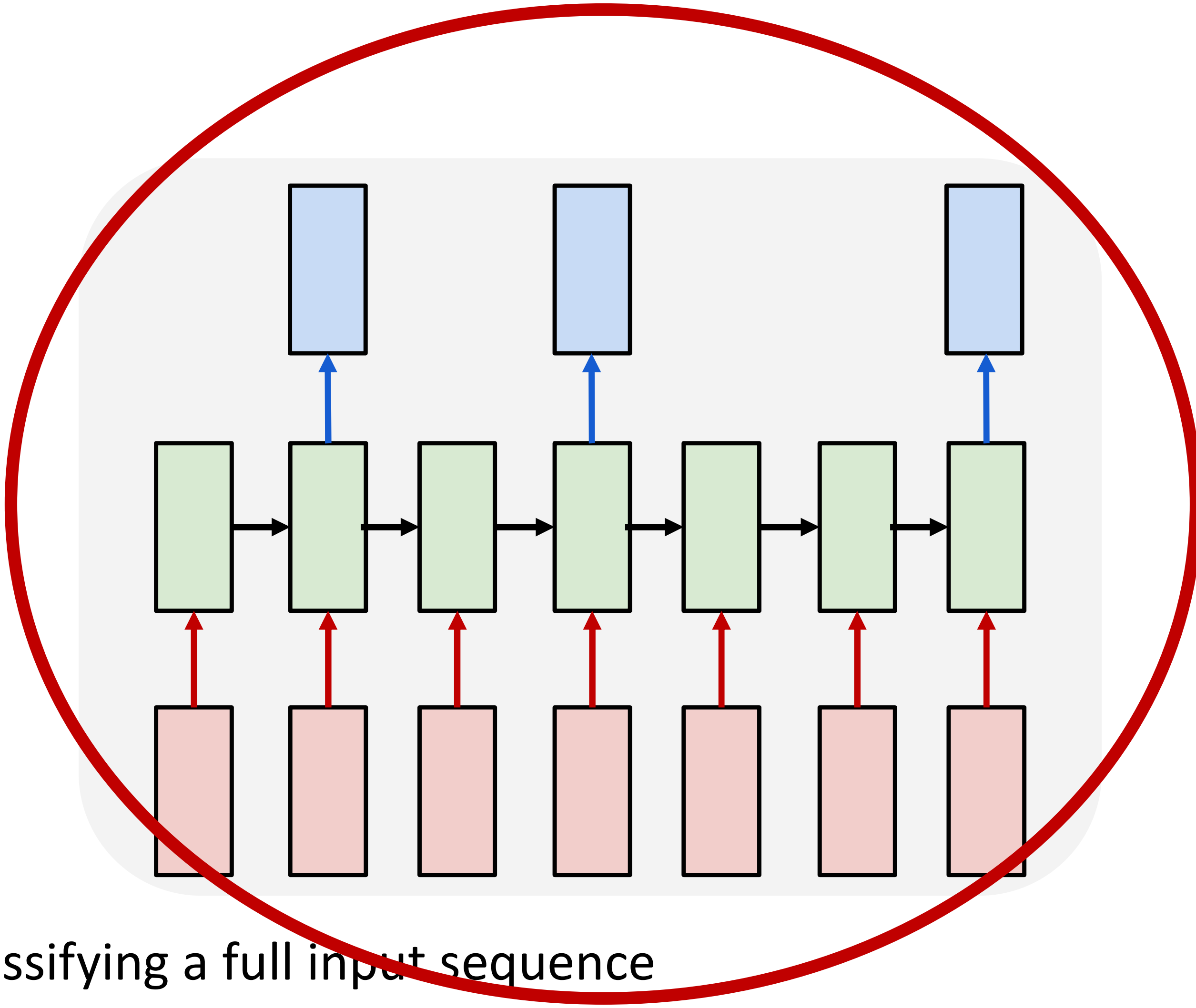
- Define the divergence everywhere

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

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

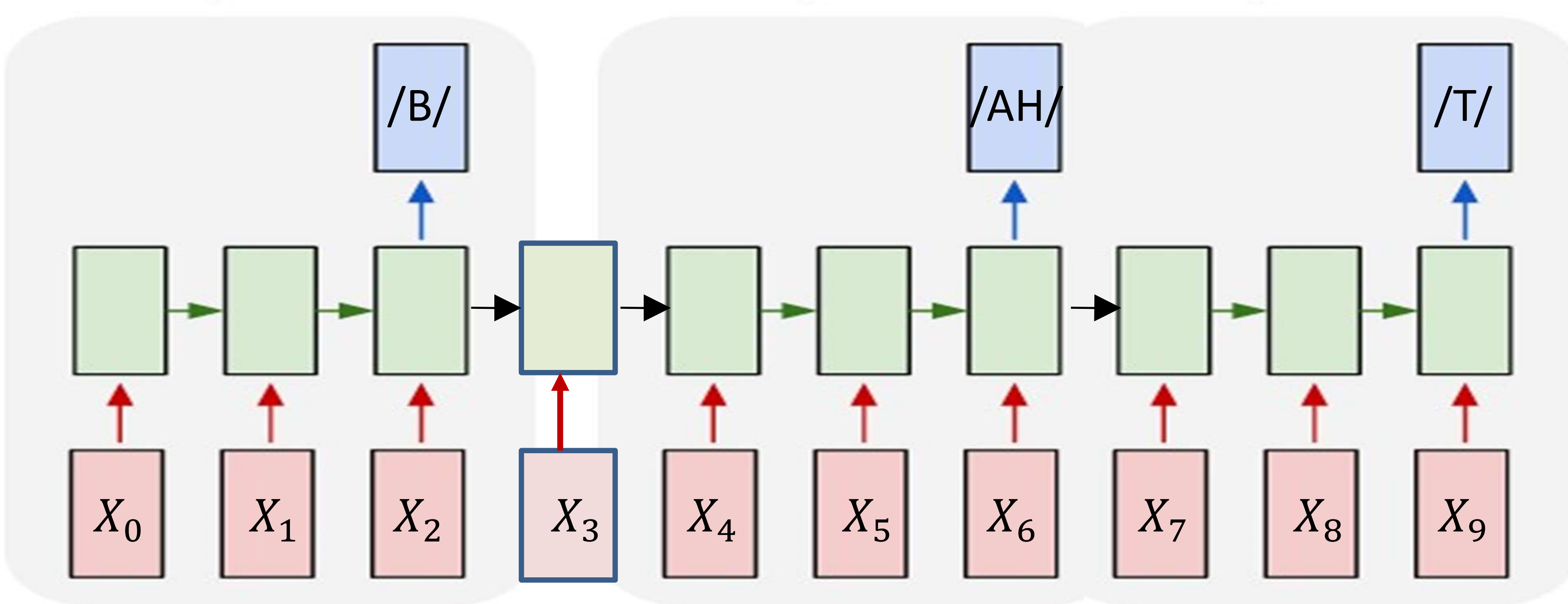
Variants on recurrent nets





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

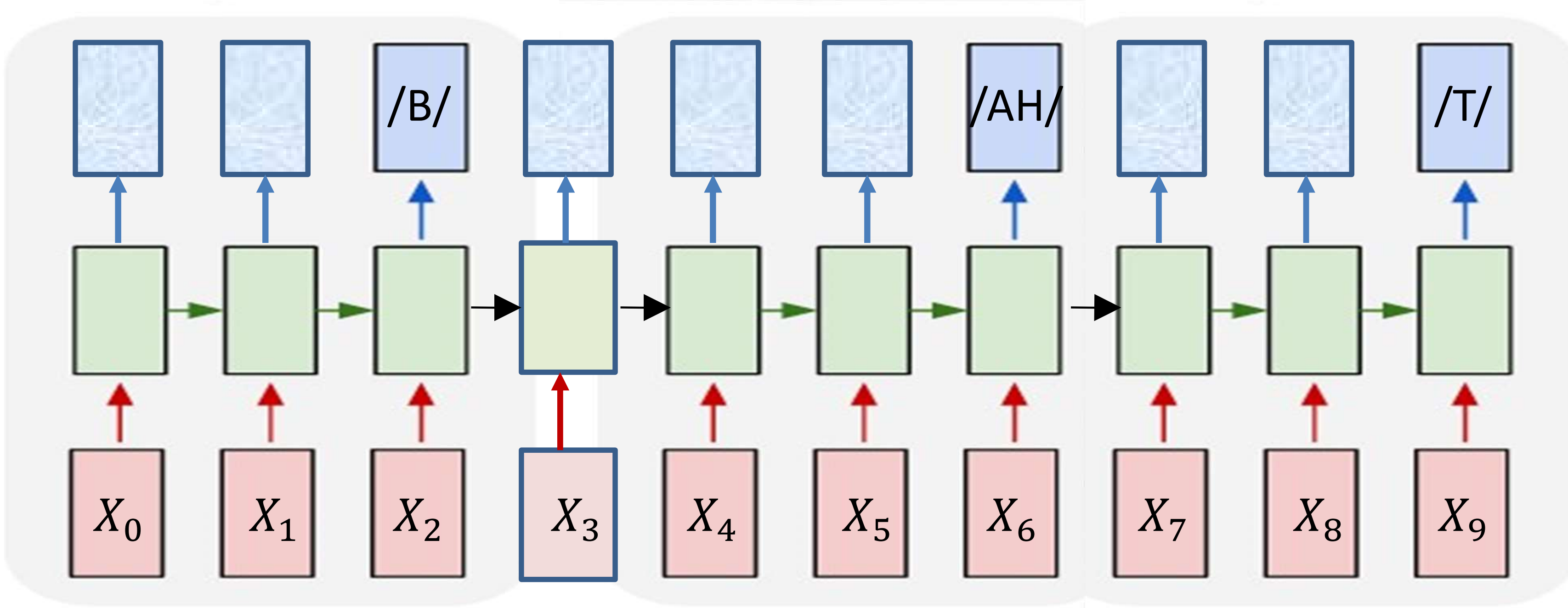
A more complex problem



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

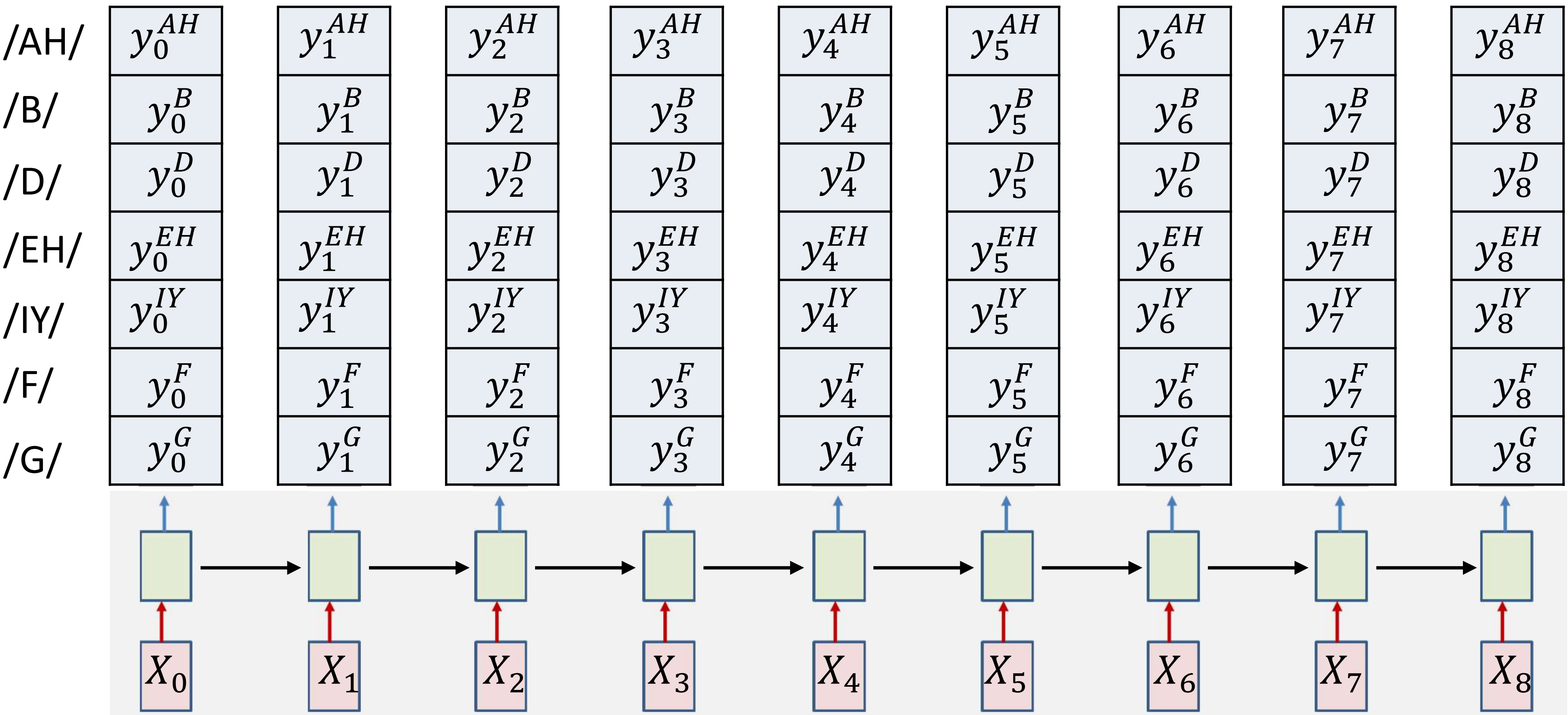
- But this simple extension complicates matters..

The *sequence-to-sequence* problem



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

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

Recap: The output of a network

- Any neural network with a softmax (or logistic) output is actually outputting an estimate of the *a posteriori* probability of the classes given the output

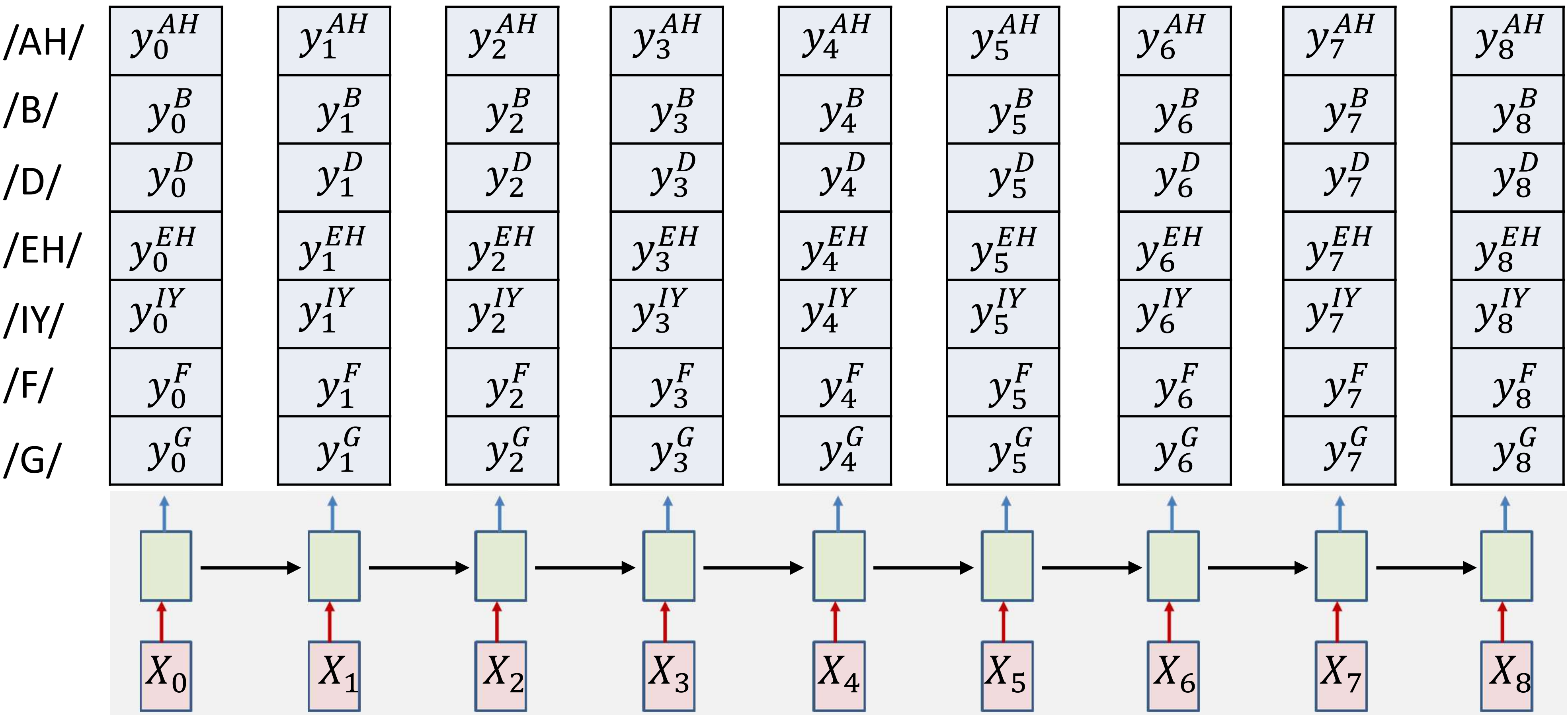
$$[P(c_1|X), P(c_2|X), \ldots, P(c_K|X)]$$

- Selecting the class with the highest probability results in *maximum a posteriori probability* classification

$$Class = \underset{i}{\operatorname{argmax}} P(Y_i|X)$$

- We use the same principle here

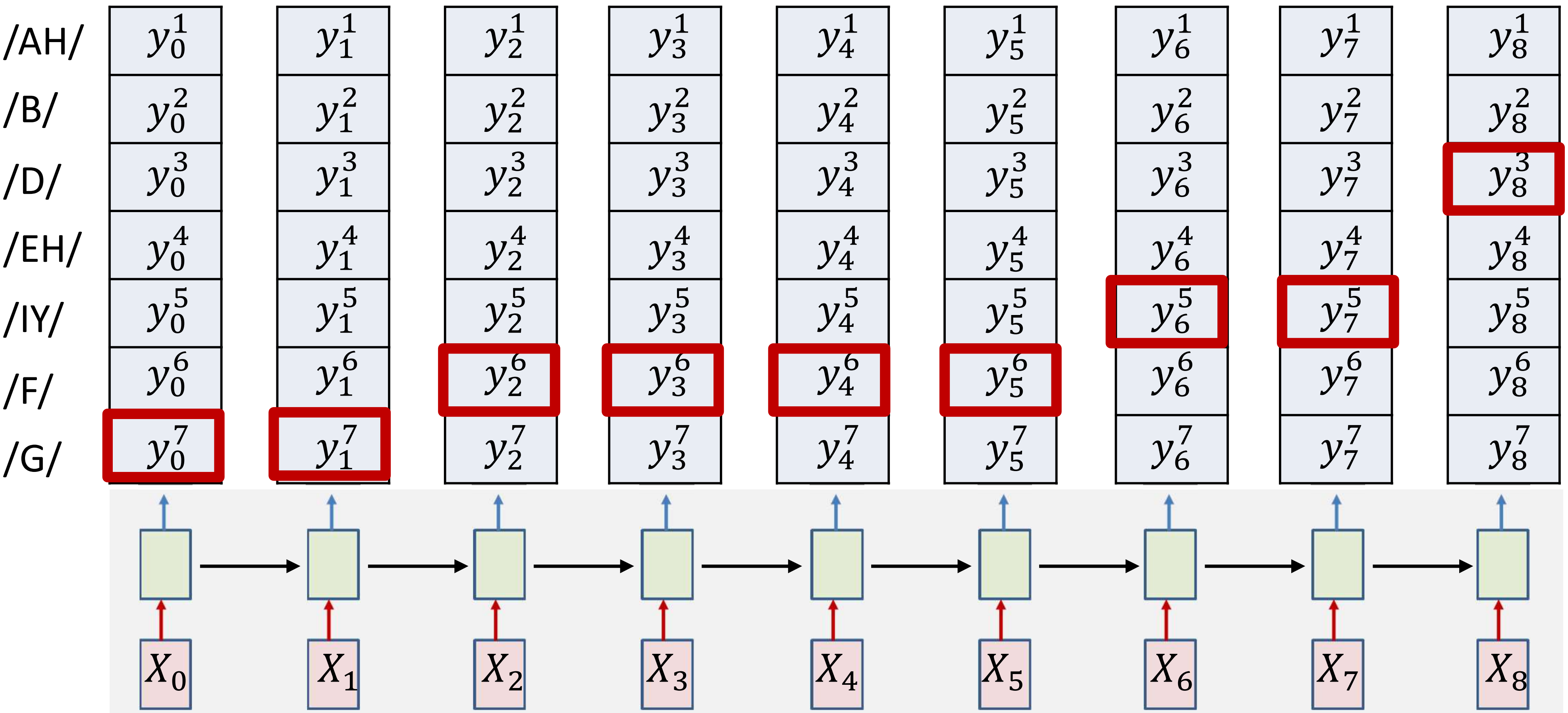
Overall objective



- Find most likely symbol sequence given inputs

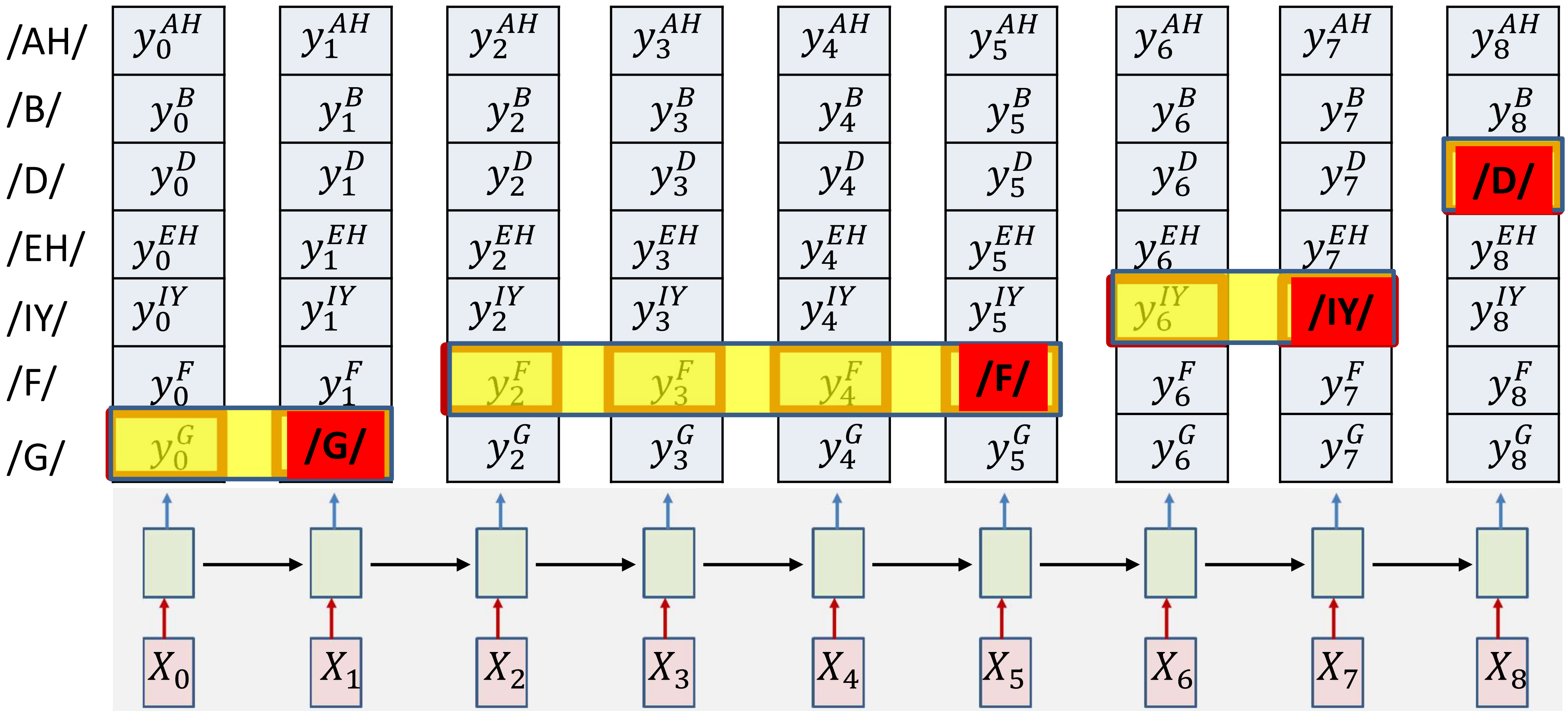
$$S_0 \dots S_{K-1} = \underset{S'_0 \dots S'_{K-1}}{\operatorname{argmax}} \; prob(S'_0 \dots S'_{K-1} | X_0 \dots X_{N-1})$$

Finding the best output



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

Finding the best output



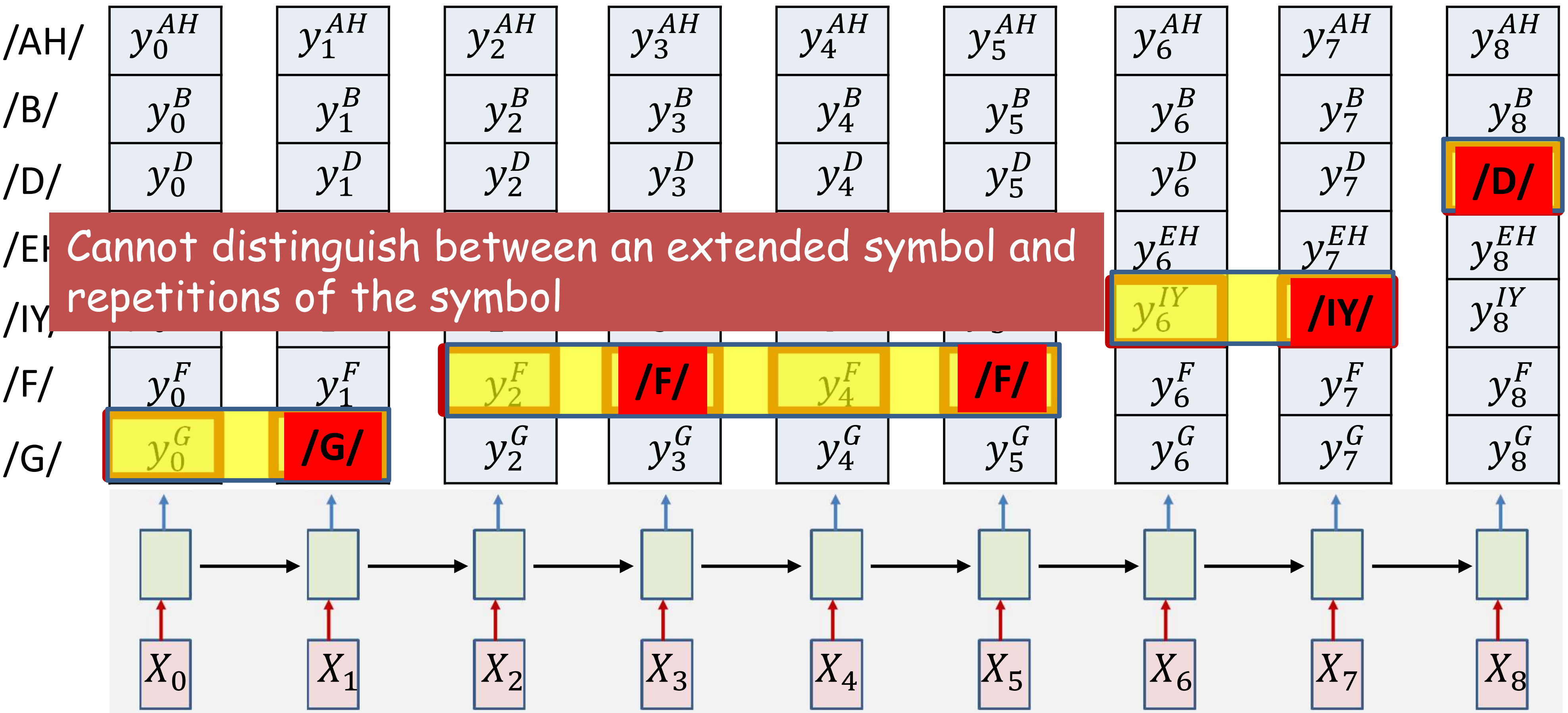
- Option 1: Simply select the most probable symbol at each time
  - *Merge* adjacent repeated symbols, and place the actual emission of the symbol in the final instant

Simple pseudocode

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

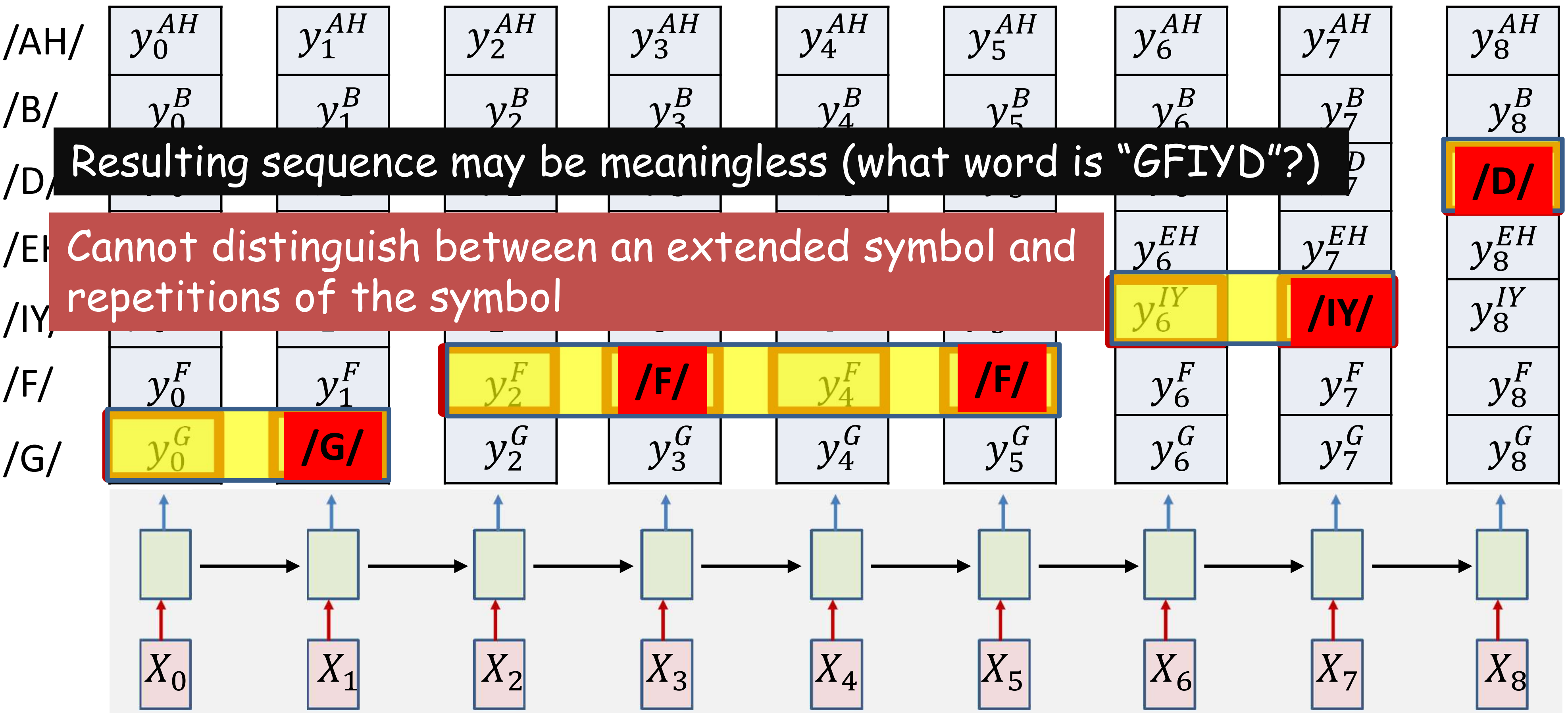
```
n = 1
best(1)= argmax_i(y(1,i))
for t = 1:T
    best(t)= argmax_i(y(t,i))
    if (best(t) != best(t-1))
        out(n) = best(t-1)
        time(n) = t-1
        n = n+1
```

Finding the best output



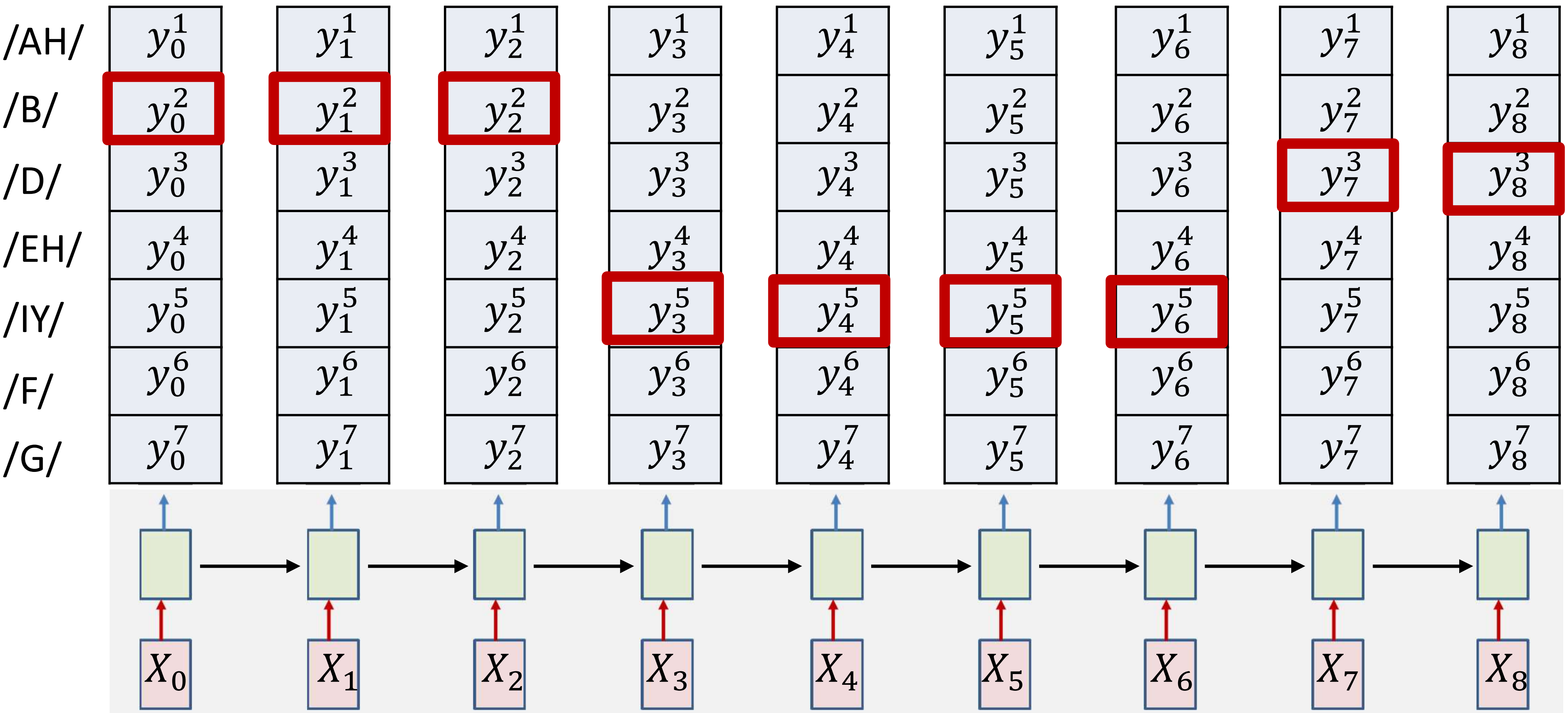
- Option 1: Simply select the most probable symbol at each time
  - *Merge* adjacent repeated symbols, and place the actual emission of the symbol in the final instant

Finding the best output



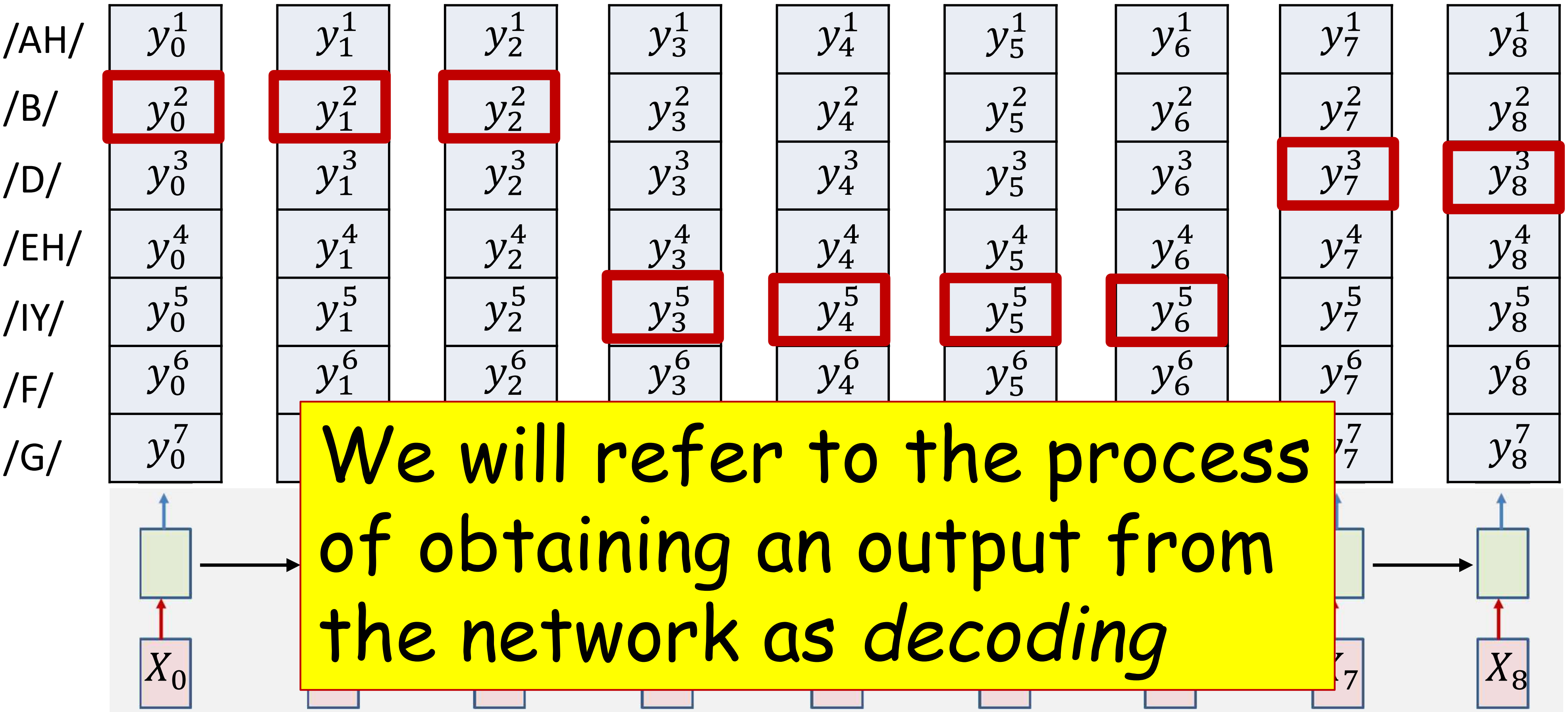
- Option 1: Simply select the most probable symbol at each time
  - *Merge* adjacent repeated symbols, and place the actual emission of the symbol in the final instant

Finding the best output



- Option 2: Impose external constraints on what sequences are allowed
  - *E.g.* only allow sequences corresponding to dictionary words
  - *E.g.* using special “separating” symbols to separate repetitions

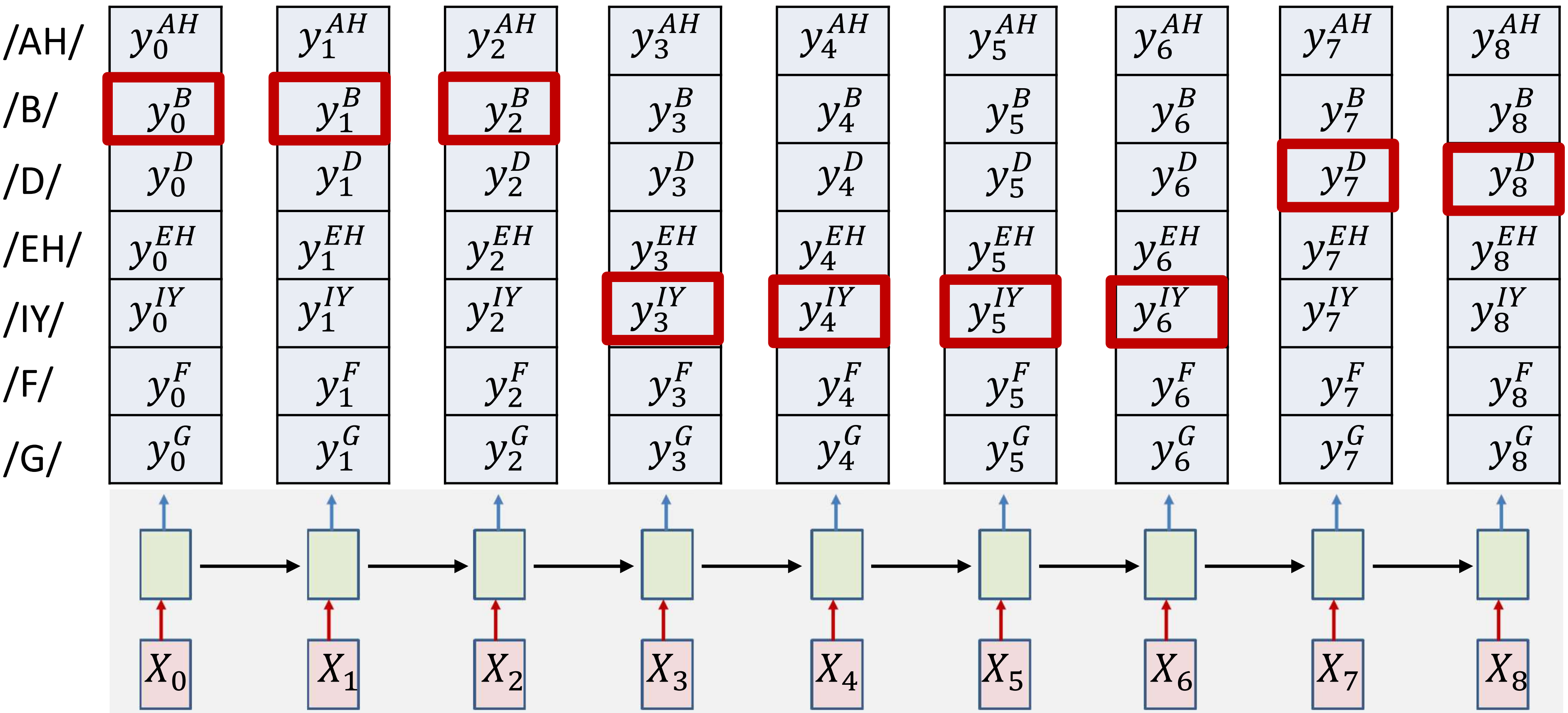
Finding the best output



We will refer to the process of obtaining an output from the network as *decoding*

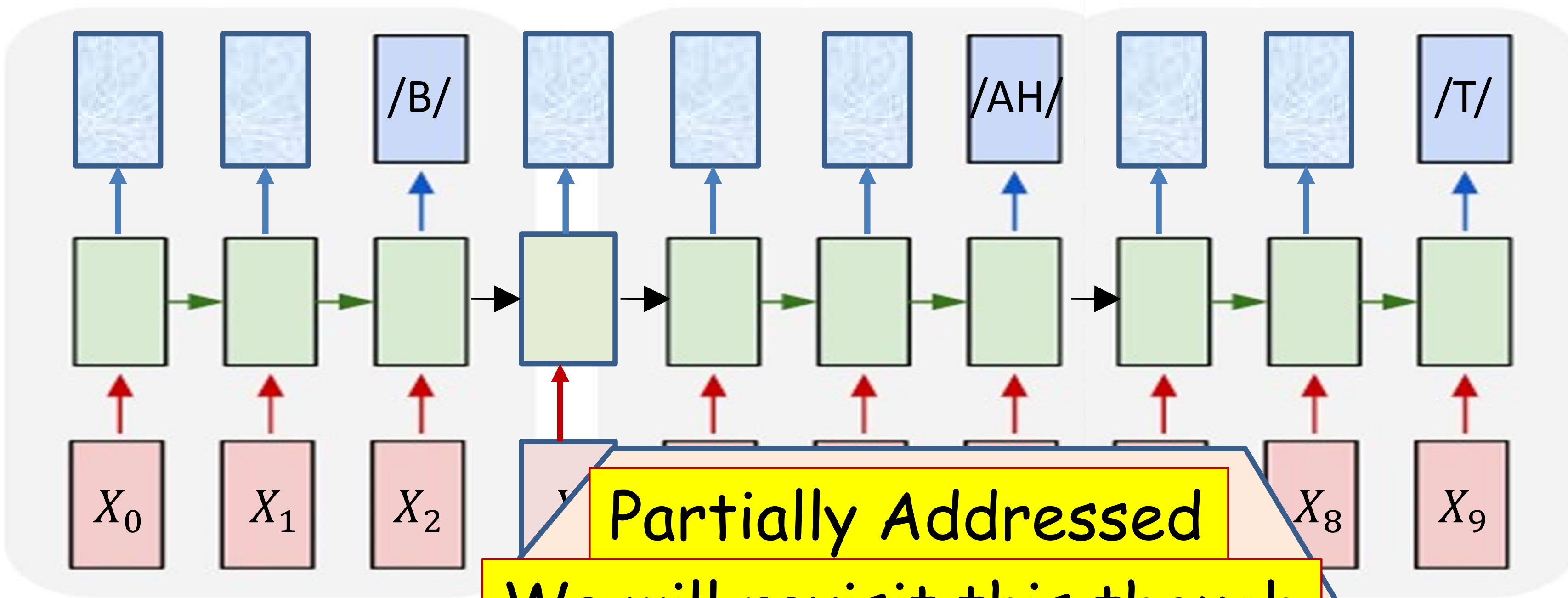
- Option 2: Impose external constraints on what sequences are allowed
  - *E.g.* only allow sequences corresponding to dictionary words
  - *E.g.* using special "separating" symbols to separate repetitions

Decoding



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

The *sequence-to-sequence* problem



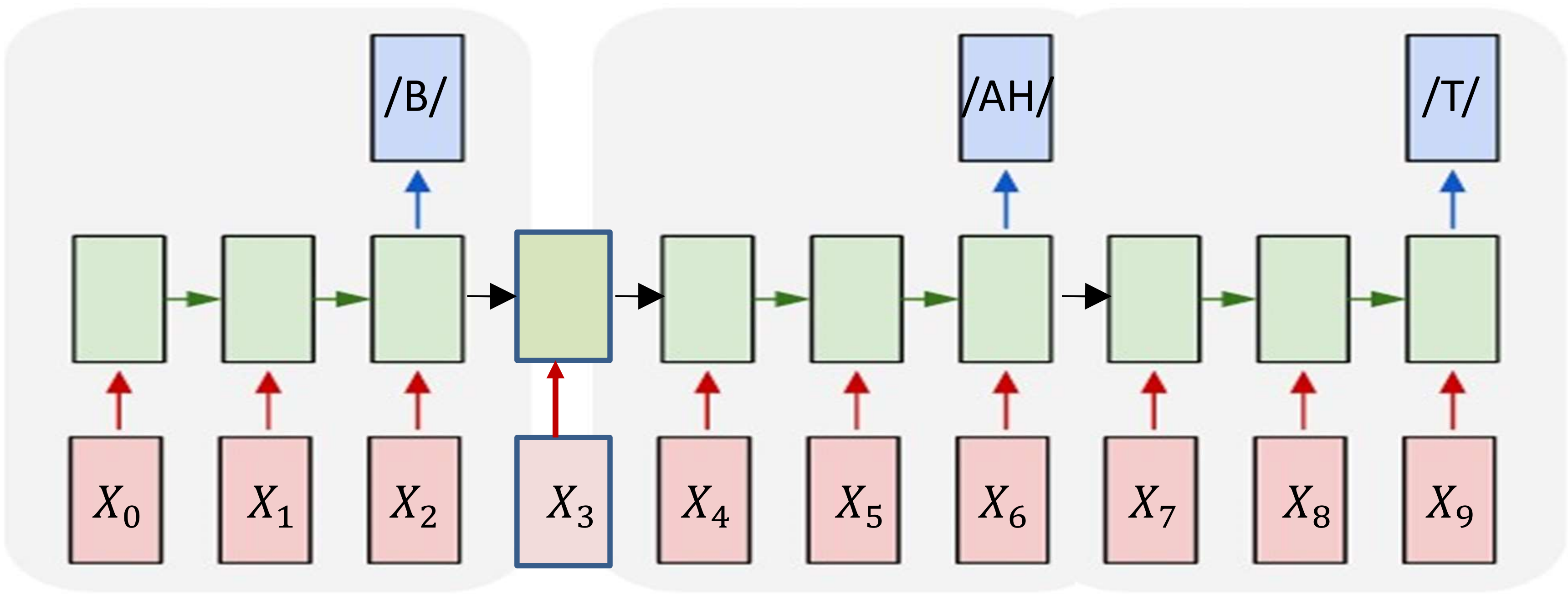
Partially Addressed

We will revisit this though

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

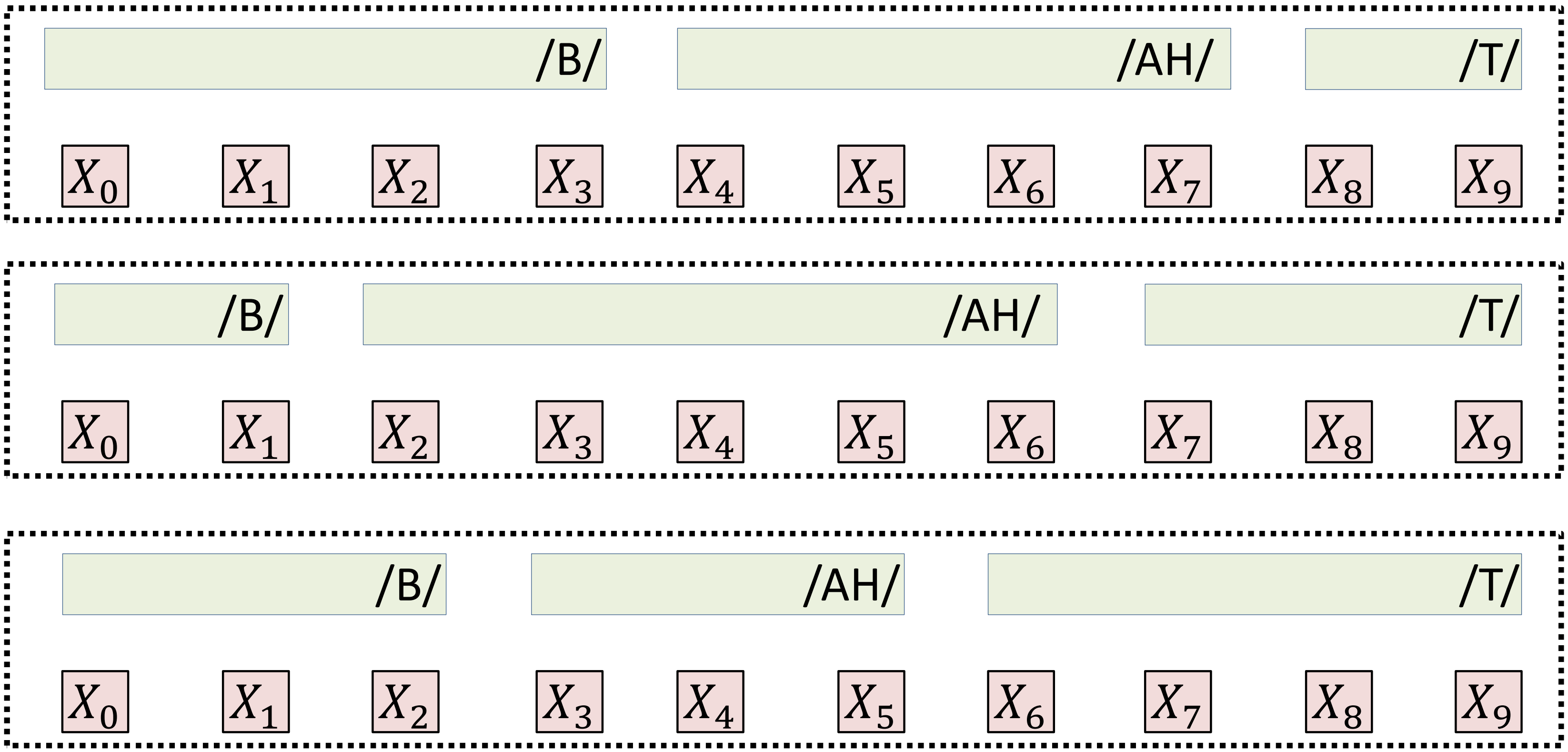
- How do we *train* these models?

Training



- Training data: input sequence + output sequence
  - Output sequence length <= input sequence length
- Given output symbols *at the right locations*
  - The phoneme /B/ ends at X_2, /AH/ at X_6, /T/ at X_9

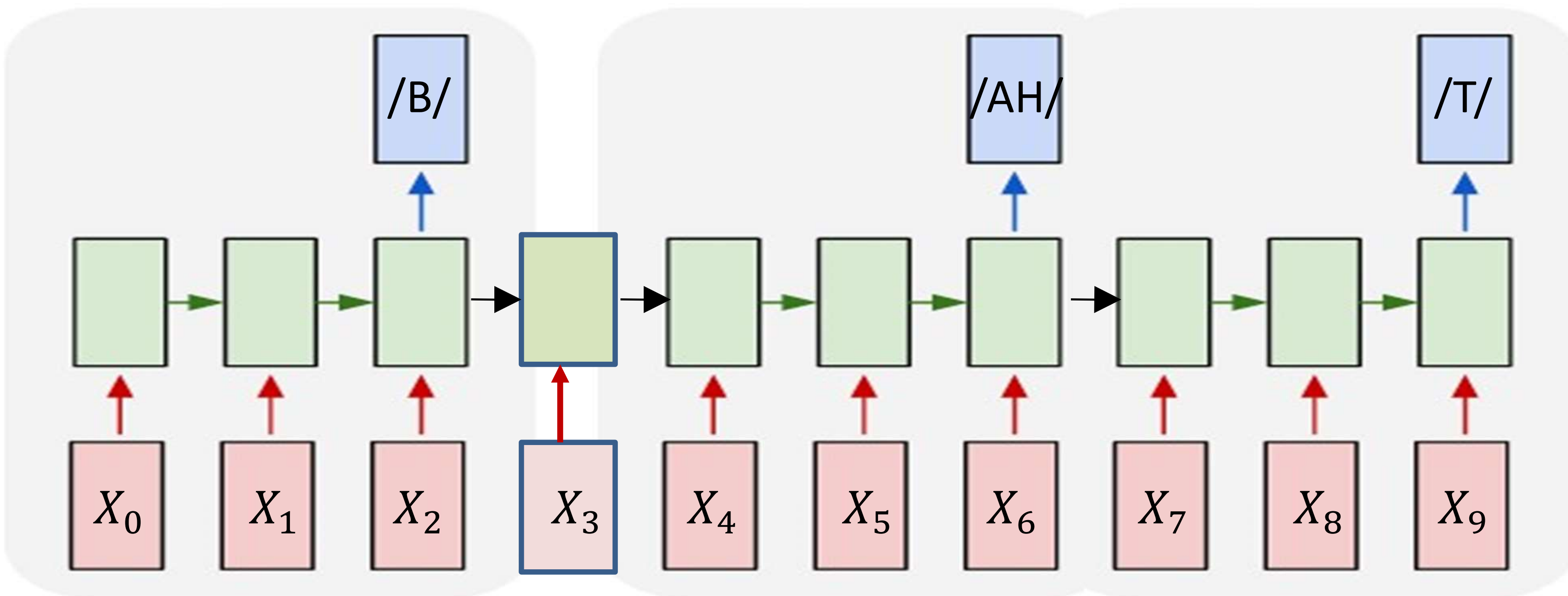
The “alignment” of labels



- The time-stamps of the output symbols give us the “alignment” of the output sequence to the input sequence
  - Which portion of the input aligns to what symbol

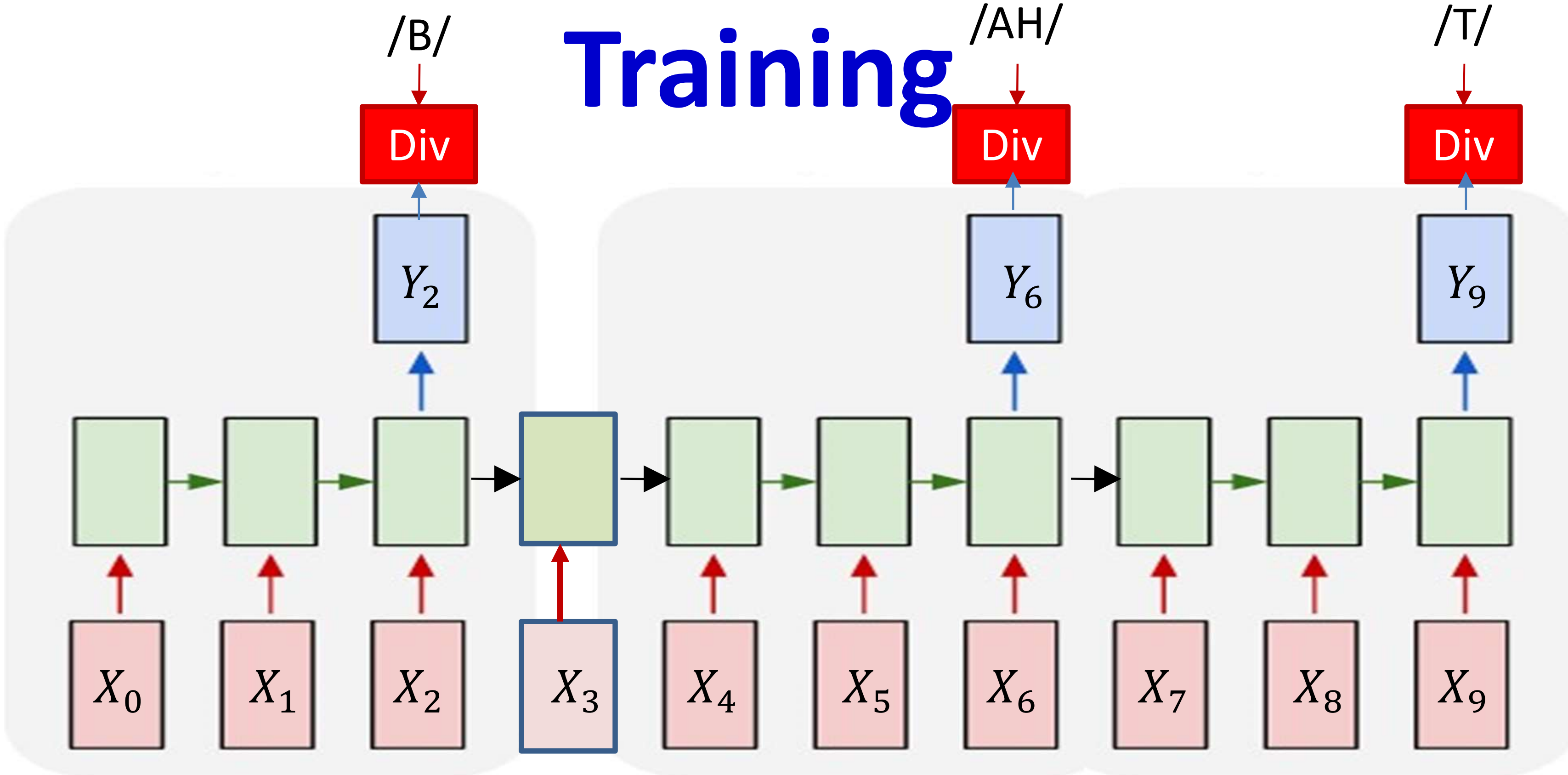
- Simply knowing the output sequence does not provide us the alignment
  - This is extra information

Training with alignment



- Training data: input sequence + output sequence
  - Output sequence length <= input sequence length
- Given the *alignment* of the output to the input
  - The phoneme /B/ ends at X_2, /AH/ at X_6, /T/ at X_9

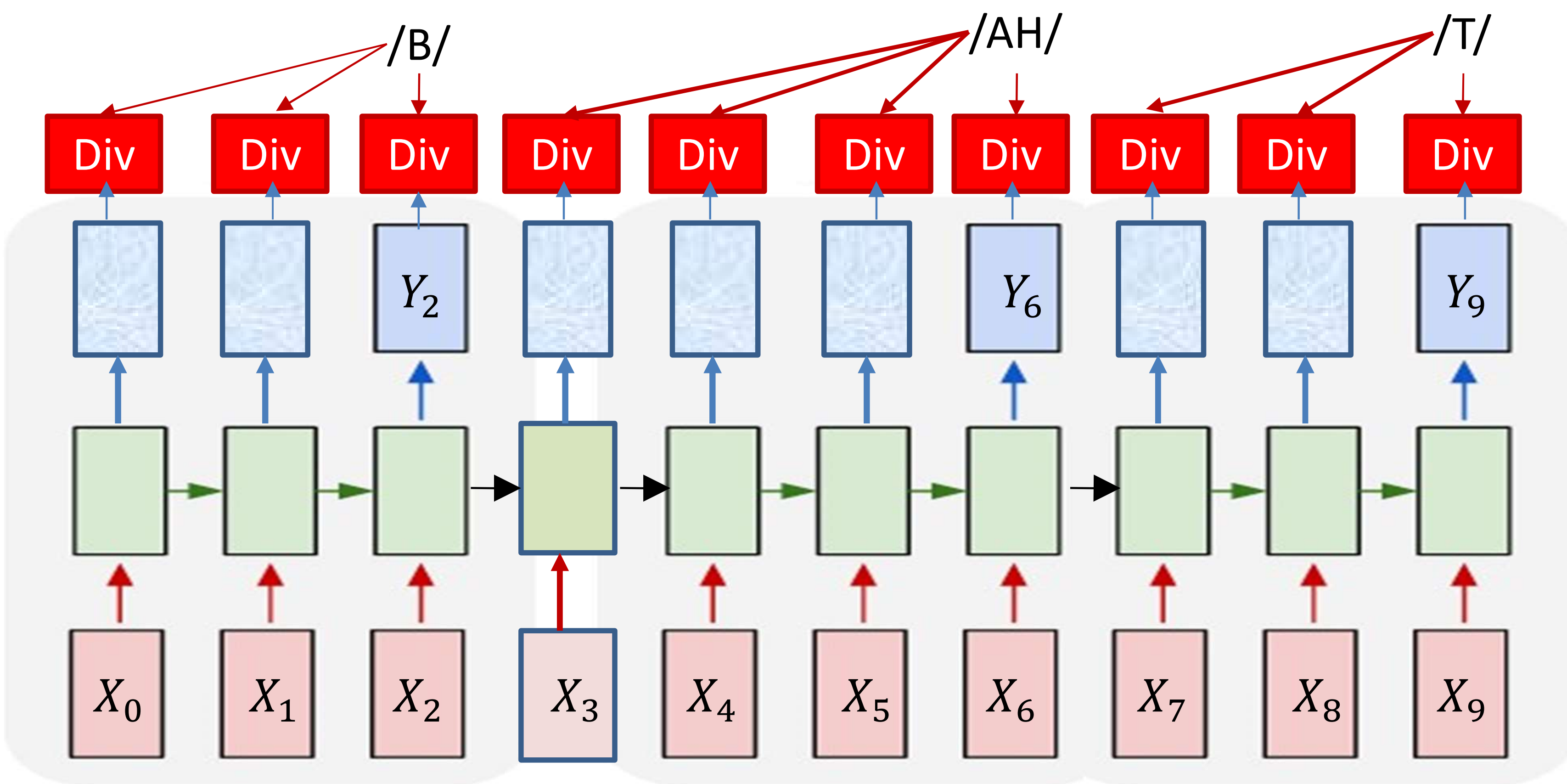
Training



- Either just define Divergence as:

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

- Or..



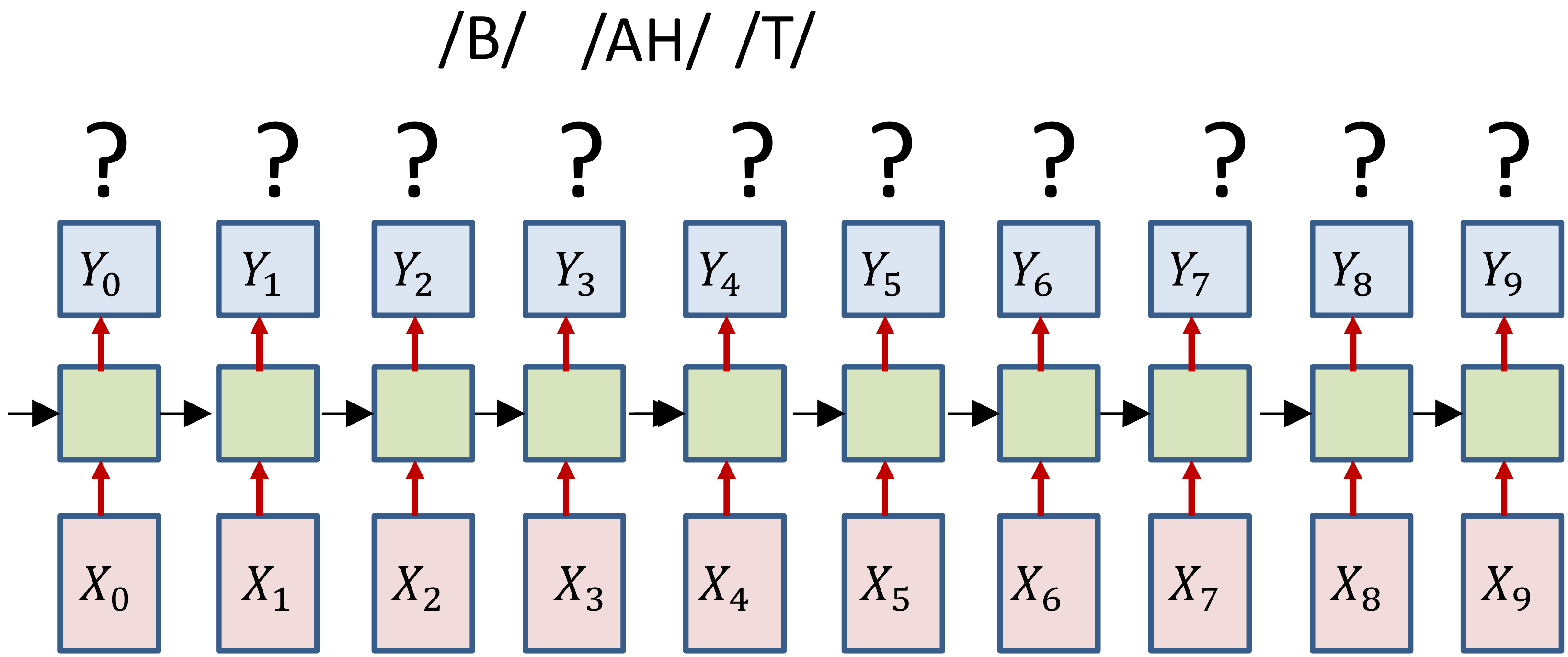
- Either just define Divergence as:

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

- Or *repeat the symbols over their duration*

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

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

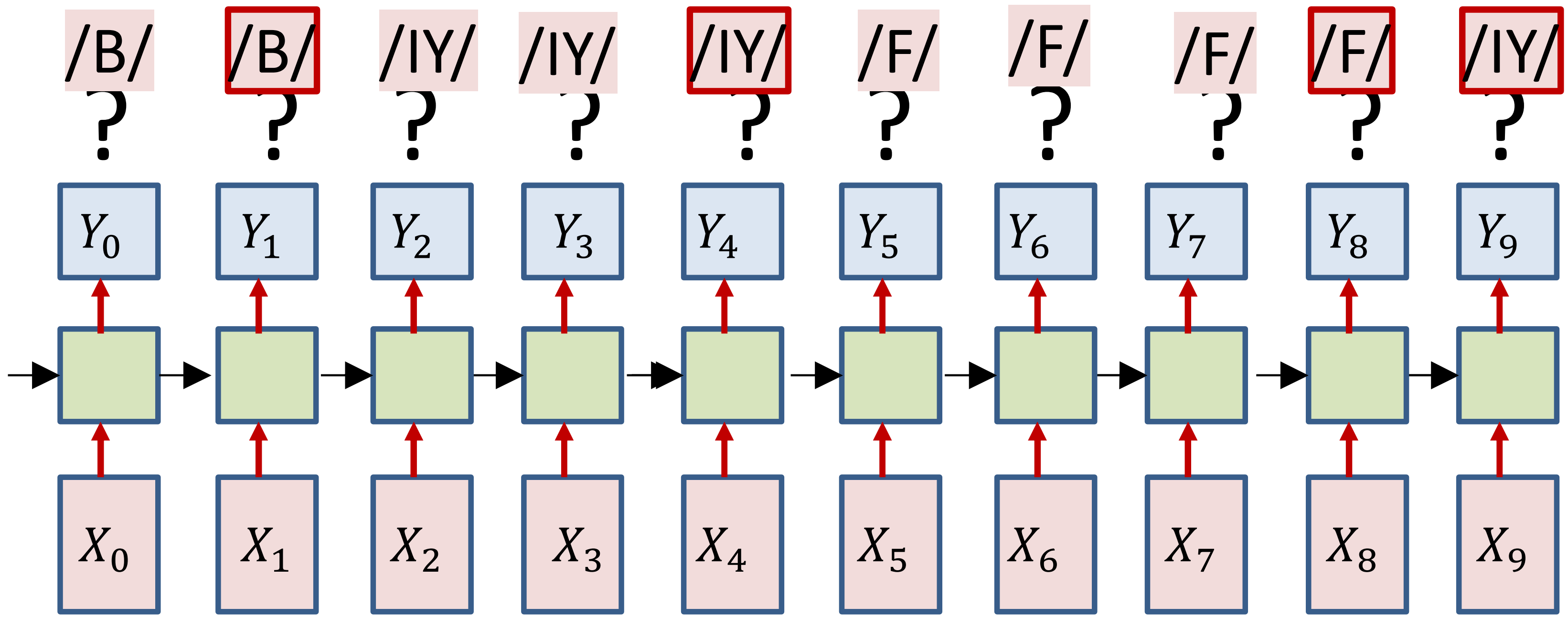
- How do we compute the divergence?
  - And how do we compute its gradient w.r.t. Y_t

Training *without* alignment

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

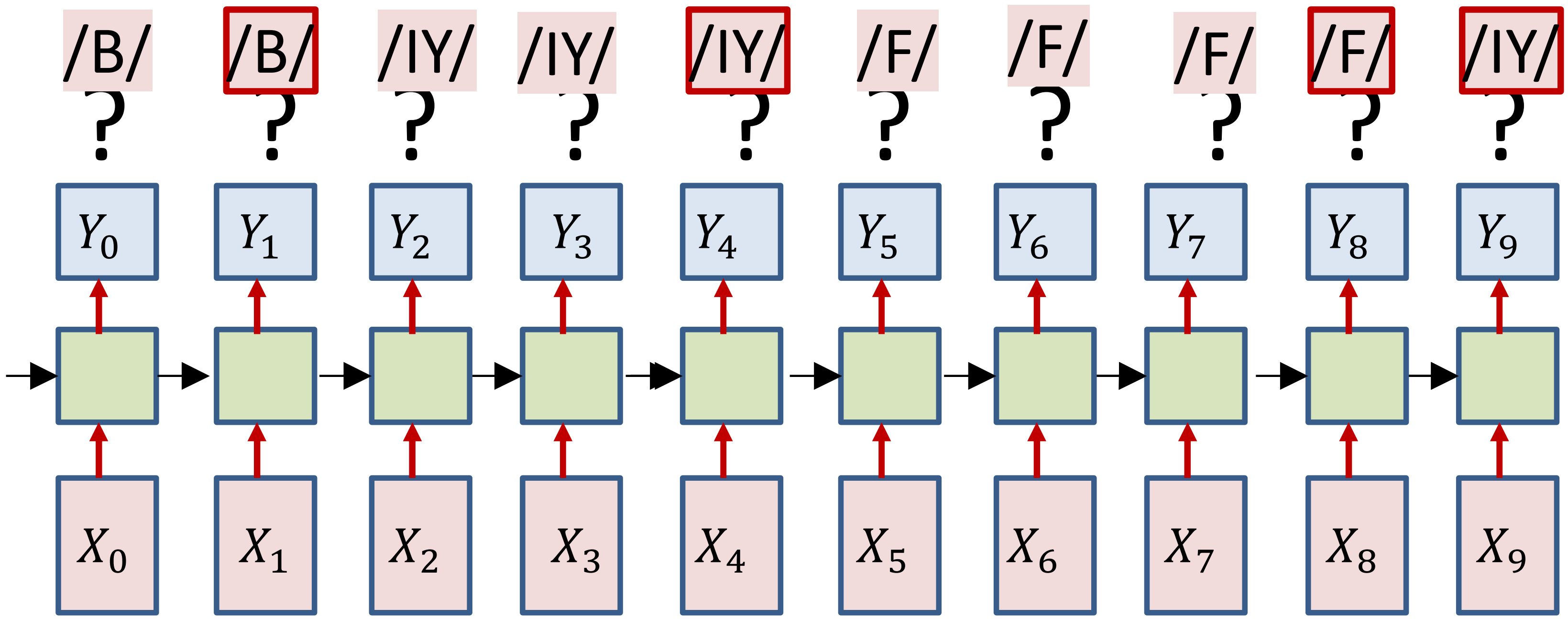
- Solution:
  1. *Guess* the alignment
  2. Consider *all possible* alignments

Solution 1: *Guess the alignment*



- Guess an initial alignment and iteratively refine it as the model improves
- 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

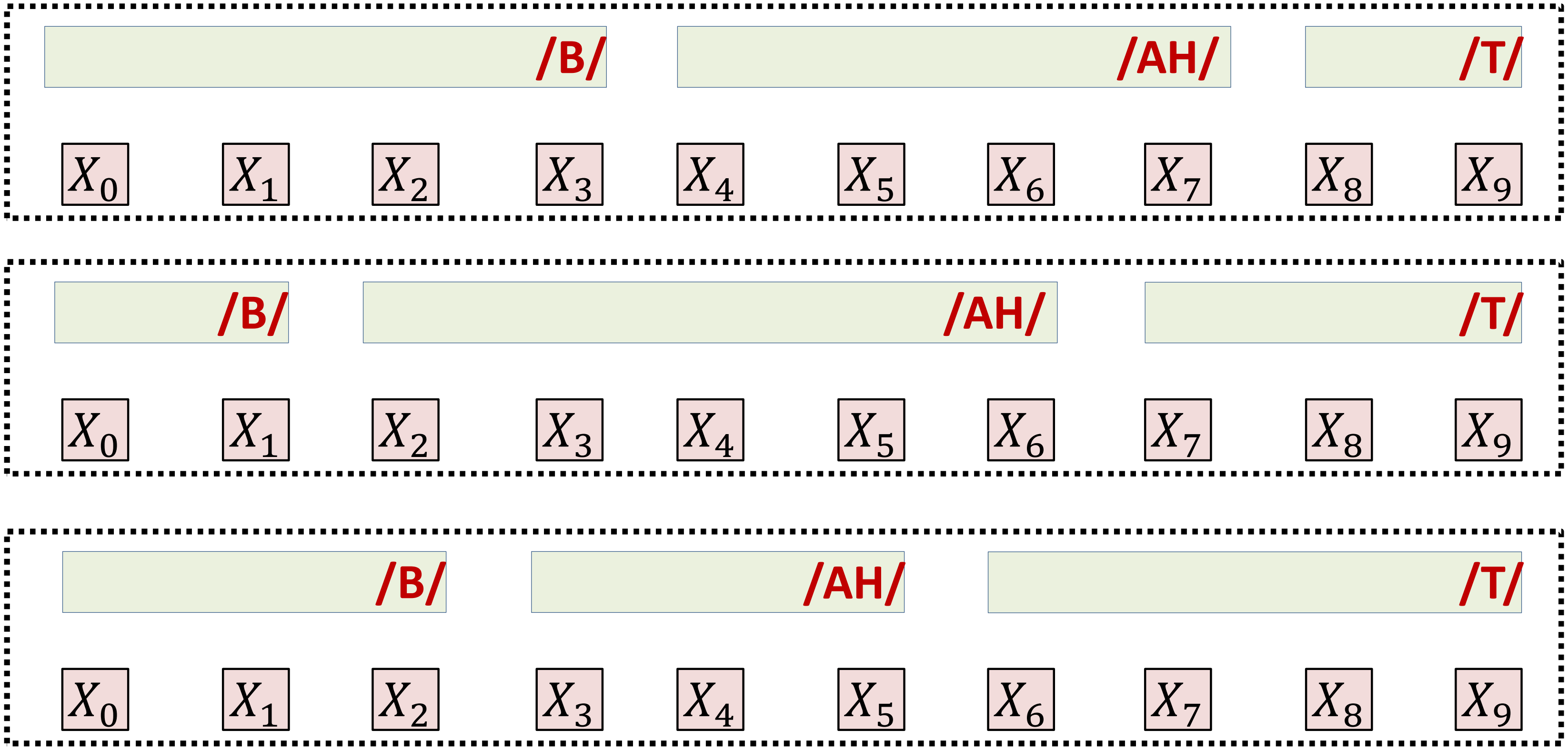
Solution 1: *Guess the alignment*



- Guess an initial alignment and iteratively refine it as the model improves

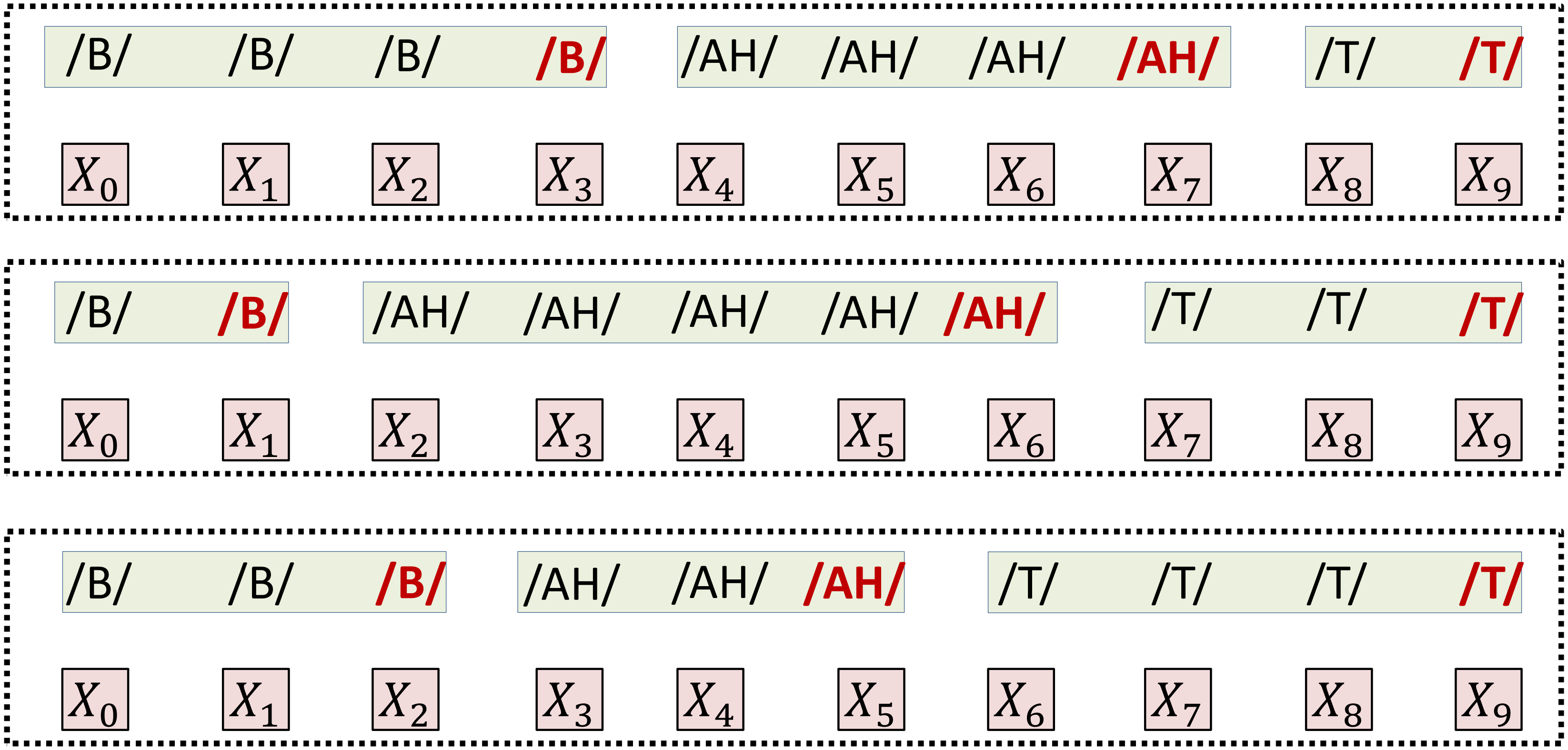
- Initialize: Assign an initial alignment
  - Either randomly, based on some heuristic, or any other rationale
- Iterate:
  - Train the network using the current alignment
  - *Reestimate* the alignment for each training instance

Characterizing the alignment



- The "alignment" tells us which portion of the input aligns to what symbol in the sequence
  - Examples show different alignments of /B/ /AH/ /T/ to $X_0 \ldots X_9$

Characterizing the alignment



- The "alignment" tells us which portion of the input aligns to what symbol in the sequence
  - Examples show different alignments of /B/ /AH/ /T/ to $X_0 \ldots X_9$
- An alignment can be represented as a repetition of symbols
  - The "**expansion**" of the "**compressed**" sequence to the length of the input

Expansion and Compression

/B/ /AH/ /T/ ⇨

/B/ /B/ /B/ **/B/**	/AH/ /AH/ /AH/ **/AH/**	/T/ **/T/**
/B/ **/B/**	/AH/ /AH/ /AH/ /AH/ **/AH/**	/T/ /T/ **/T/**
/B/ /B/ **/B/**	/AH/ /AH/ **/AH/**	/T/ /T/ /T/ **/T/**

- The same asynchronous "compressed" sequence can be "expanded" in many different ways to align it to an input

/B/ /B/ /B/ **/B/**	/AH/ /AH/ /AH/ **/AH/**	/T/ **/T/**
/B/ **/B/**	/AH/ /AH/ /AH/ /AH/ **/AH/**	/T/ /T/ **/T/**
/B/ /B/ **/B/**	/AH/ /AH/ **/AH/**	/T/ /T/ /T/ **/T/**

⇨ /B/ /AH/ /T/

- Many different alignments for an input can compress to the same unaligned "compressed" sequence
- The problem of finding the alignment: find the best expansion of a compressed sequence, for a given input, *given a model*

Estimating an alignment

- Alignment problem: Given
  - The unaligned K-length compressed symbol sequence $S = S_0 \ldots S_{K-1}$
    - E.g. /B/ /IY/ /F/ /IY/
  - An N-length input ($N \geq K$)
    - E.g. input $X_0, X_1, \ldots, X_9$
  - And a (trained) recurrent network

- Find the most likely alignment:

$$argmax\ P(s_0, s_1, \ldots, s_{N-1} | S_0, S_1, \ldots, S_K, X_0, X_1, \ldots, X_{N-1})$$

  - Such that

$$compress(s_0, s_1, \ldots, s_{N-1}) \equiv S_0, S_1, \ldots, S_K$$

  - $compress()$ is the operation of compressing repetitions into one

Poll 3

- @

Select all that are true about alignments, time-synchronous sequences, order-synchronous sequences, compression, and compressed sequences

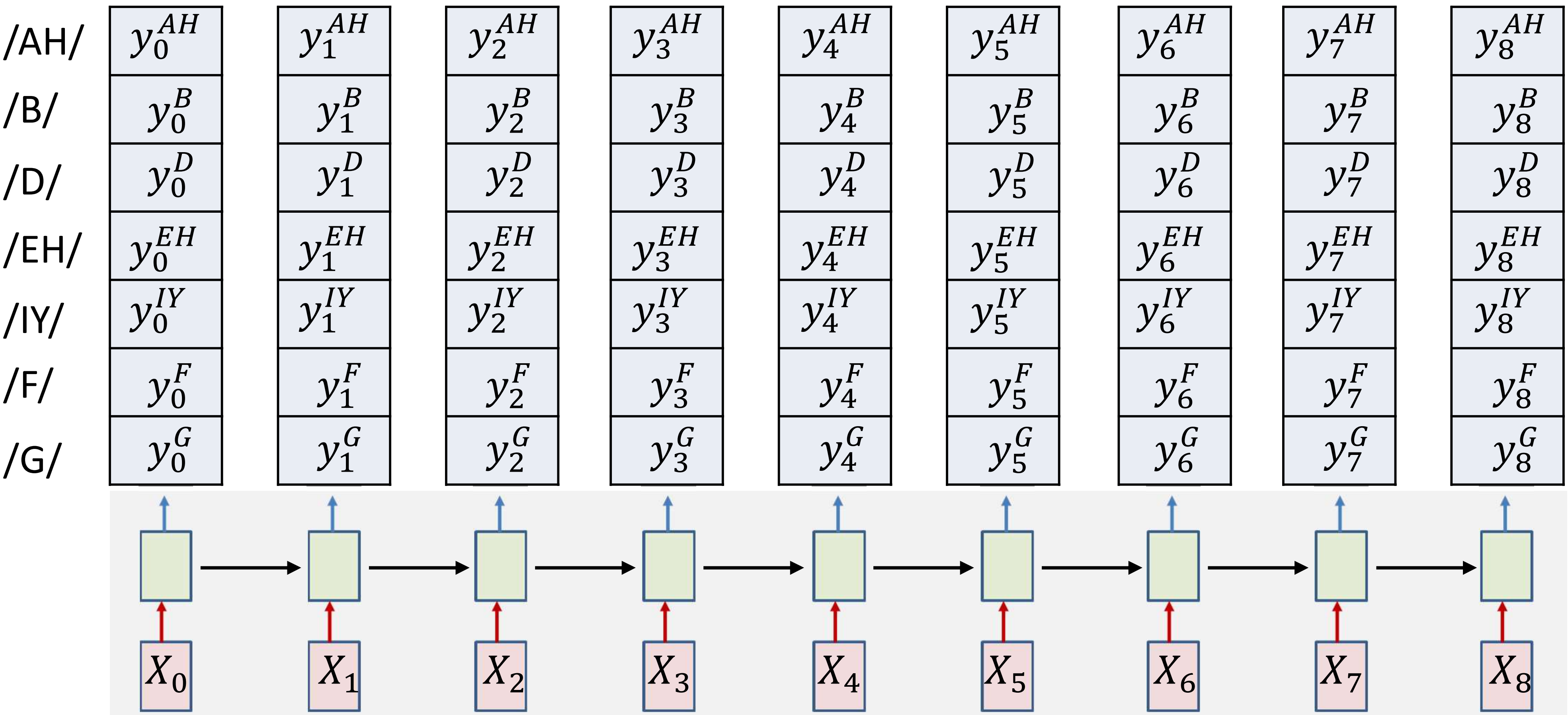
- An order-synchronous symbol sequence that is shorter than the input can be “aligned” to the input by repeating symbols until the expanded sequence is exactly as long as the input
- The “alignment” of an order-synchronous symbol sequence to an input is a time-synchronous symbol sequence
- A symbol sequence that is time-synchronous with an input can be compressed to a shorter order-synchronous input by eliminating repetitions of symbols
- Order-synchronous symbol sequences that are shorter than the input are compressed symbol sequences
- There is only one way of generating an alignment of a compressed symbol sequence to an input

Poll 3

Select all that are true about alignments, time-synchronous sequences, order-synchronous sequences, compression, and compressed sequences

- **An order-synchronous symbol sequence that is shorter than the input can be “aligned” to the input by repeating symbols until the expanded sequence is exactly as long as the input**
- **The “alignment” of an order-synchronous symbol sequence to an input is a time-synchronous symbol sequence**
- **A symbol sequence that is time-synchronous with an input can be compressed to a shorter order-synchronous input by eliminating repetitions of symbols**
- **Order-synchronous symbol sequences that are shorter than the input are compressed symbol sequences**
- There is only one way of generating an alignment of a compressed symbol sequence to an input

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

[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

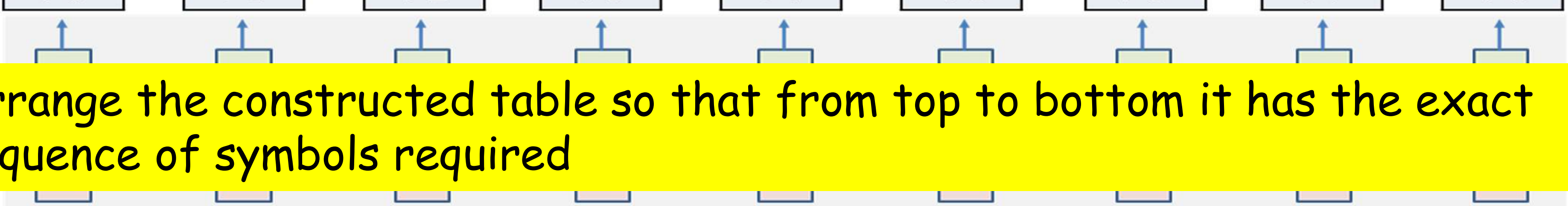
- Problem: This still doesn’t assure that the decode sequence correctly expands the target symbol sequence
  - E.g. the above decode is not an expansion of /B//IY//F//IY/

- Still needs additional constraints

Try 2: Explicitly arrange the constructed table

/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

/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

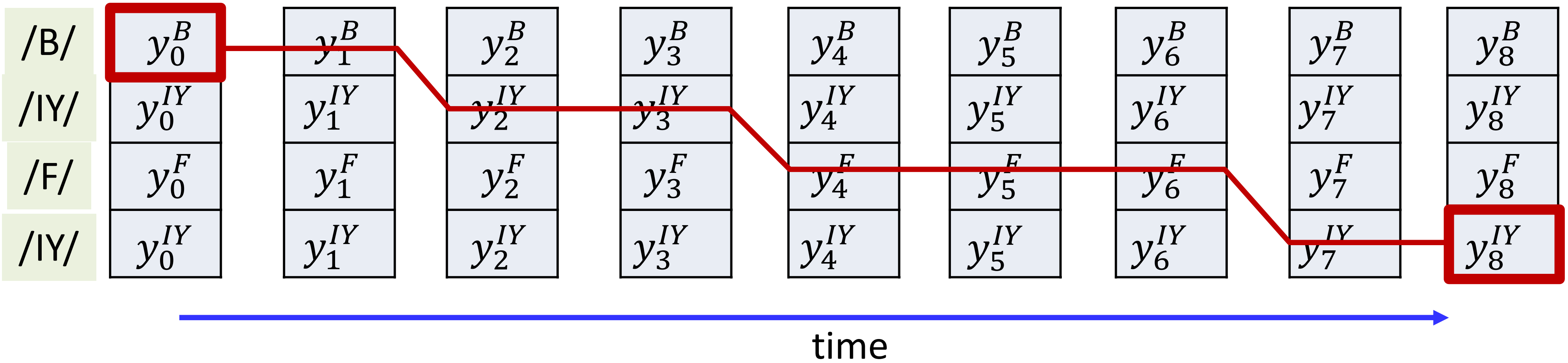
Composing the graph

```
#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))
```

Using 1..N and 1..T indexing, instead of 0..N-1, 0..T-1, for convenience of notation

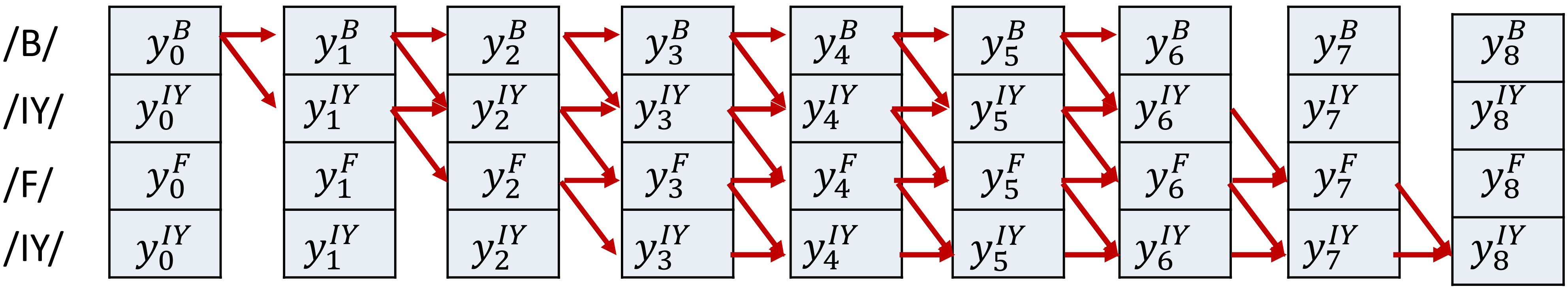
/B/	y_0^B		y_1^B		y_2^B		y_3^B		y_4^B		y_5^B		y_6^B		y_7^B		y_8^B
/IY/	y_0^{IY}		y_1^{IY}		y_2^{IY}		y_3^{IY}		y_4^{IY}		y_5^{IY}		y_6^{IY}		y_7^{IY}		y_8^{IY}
/F/	y_0^F		y_1^F		y_2^F		y_3^F		y_4^F		y_5^F		y_6^F		y_7^F		y_8^F
/IY/	y_0^{IY}		y_1^{IY}		y_2^{IY}		y_3^{IY}		y_4^{IY}		y_5^{IY}		y_6^{IY}	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

Explicitly constrain alignment



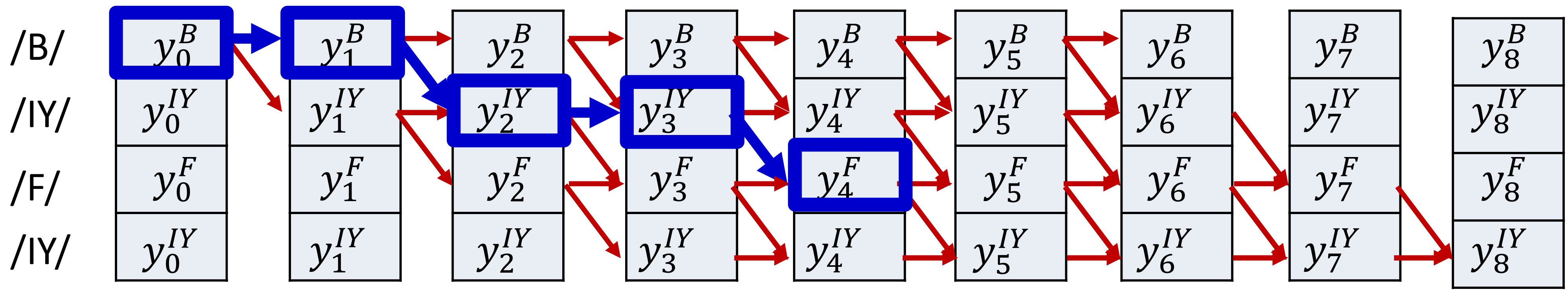
- Constrain that the first symbol in the decode *must* be the top left block
- The last symbol *must* be the bottom right
- The rest of the symbols must follow a sequence that *monotonically* travels down from top left to bottom right
  - I.e. symbol chosen at any time is at the same level or at the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



- 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//IY//F//IY/)
- Edge scores are 1
- Node scores are the probabilities assigned to the symbols by the neural network

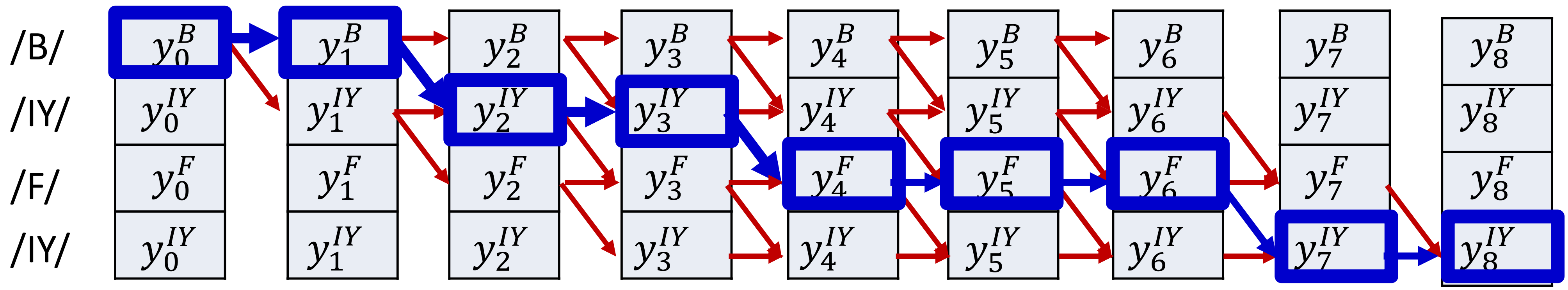
Path Score (probability)



- 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//IY//F//IY/)
- 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**

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

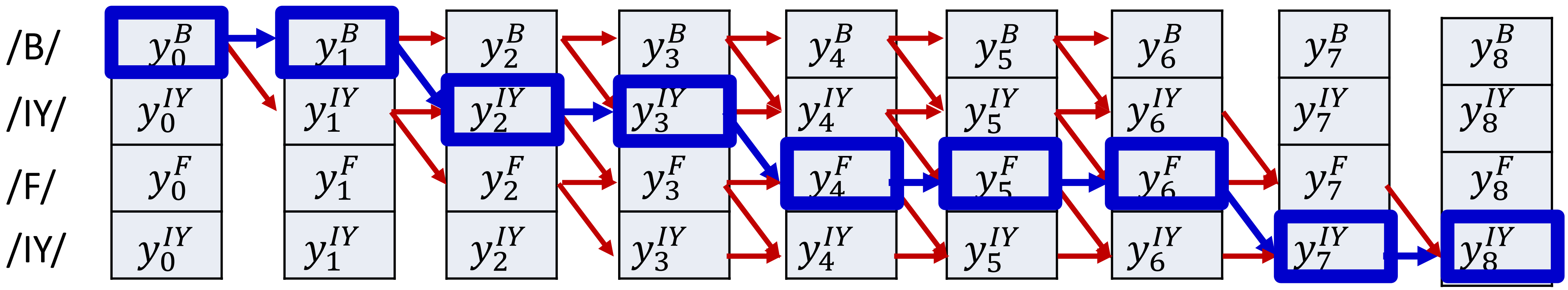
Path Score (probability)



- 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//IY//F//IY/)
- 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**

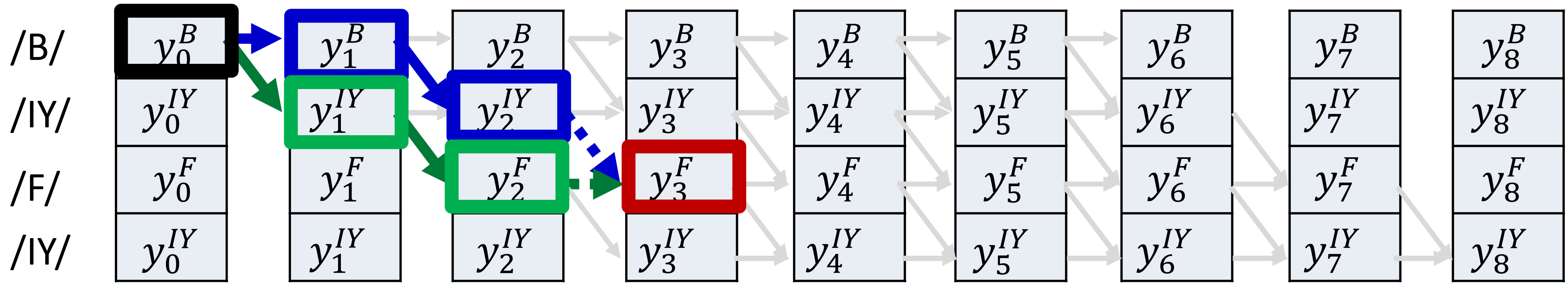
Figure shows a typical end-to-end path. There is an exponential number of such paths. Challenge: Find the path with the highest score (probability)

Explicitly constrain alignment



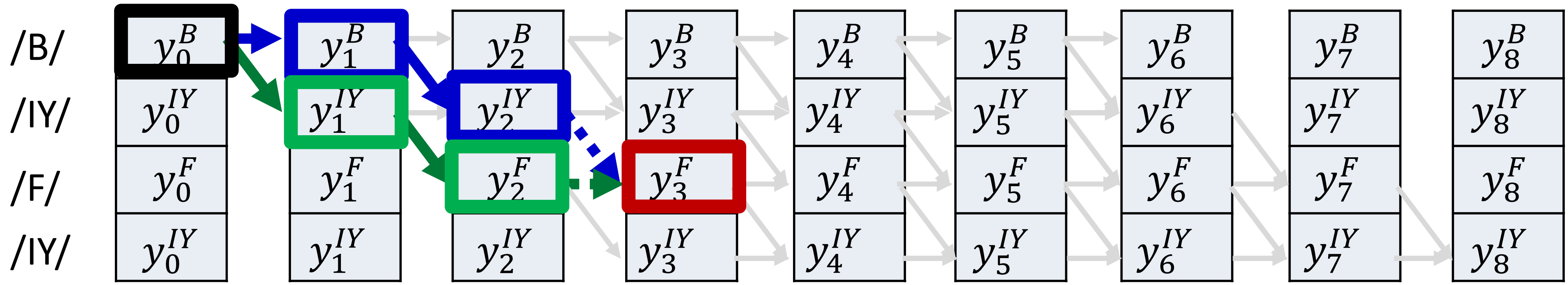
- Find the ***most probable path*** from source to sink using any dynamic programming algorithm
  - E.g. The Viterbi algorithm

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



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

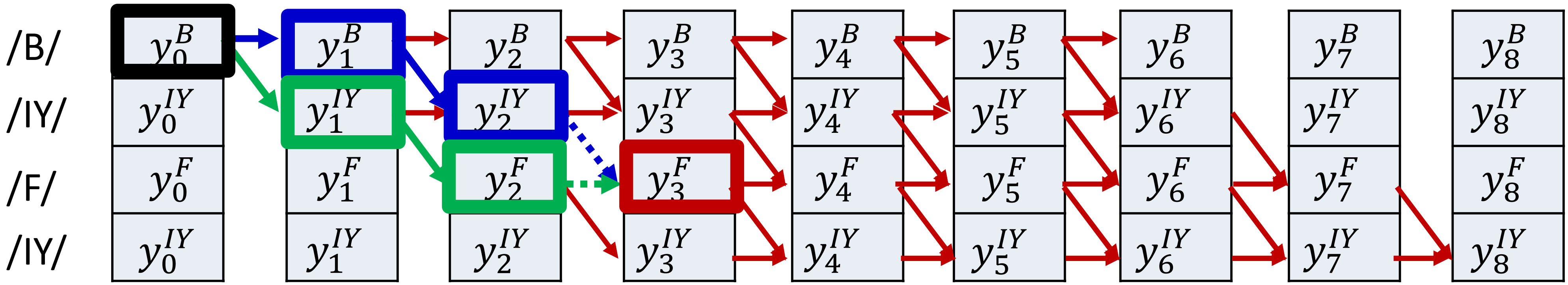
$$or \quad BestPath(y_0^B \to y_2^F)y_3^F$$

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

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

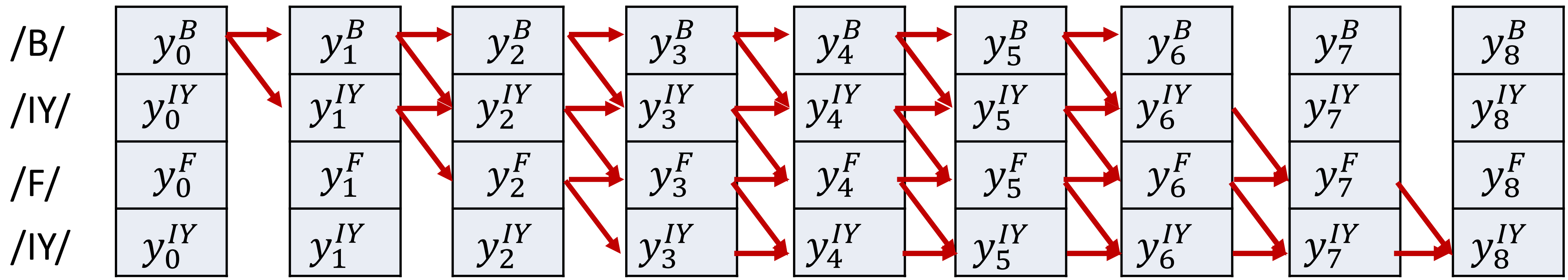
$$BestParent$$
$$= argmax_{Parent\,\in(y_2^{IY},y_2^F)}(Score(BestPath(y_0^B \to Parent)))$$

Viterbi algorithm



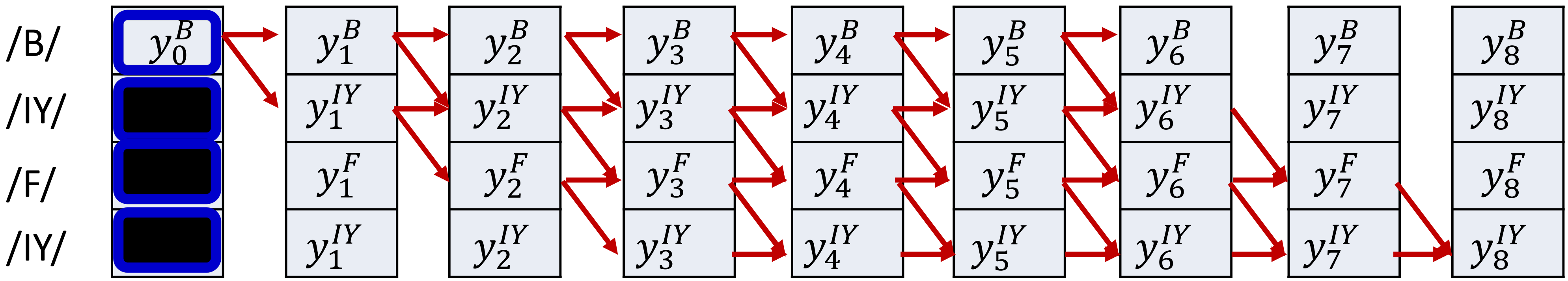
- Dynamically track the best path (and the score of the best path) from the source node to every node in the graph
  - At each node, keep track of
    - The best incoming parent edge
    - The score of the best path from the source to the node through this best parent edge
- Eventually compute the best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

Viterbi algorithm



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

Viterbi algorithm



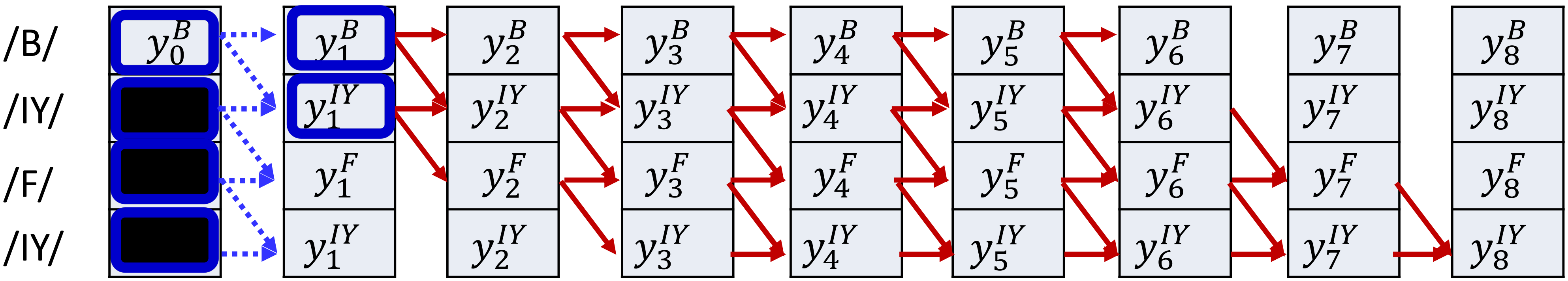
BP := Best Parent
Bscr := Bestpath Score to node

- Initialization:

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

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

Viterbi algorithm



- Initialization:

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

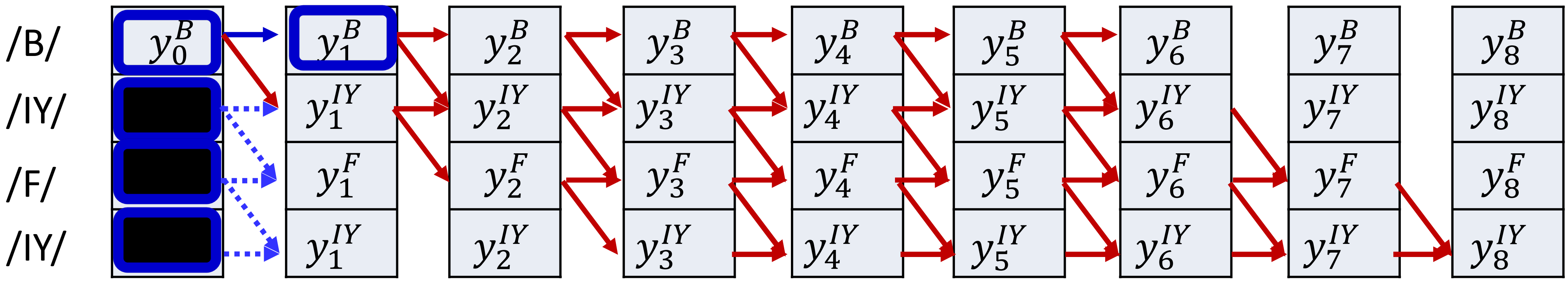
$$Bscr(0,0) = \ y_0^{S(0)}, \ Bscr(0,i) = 0 \ for \ i \ = \ 1 \ldots K-1$$

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

  for $l \ = 0 \ldots K-1$

  - $BP(t,l) = \underset{p \in parents(l)}{\operatorname{argmax}} \ Bscr(t-1,p)$
  - $Bscr(t,l) \ = \ Bscr(BP(t,l)) \ \times y_t^{S(l)}$

Viterbi algorithm



- Initialization:

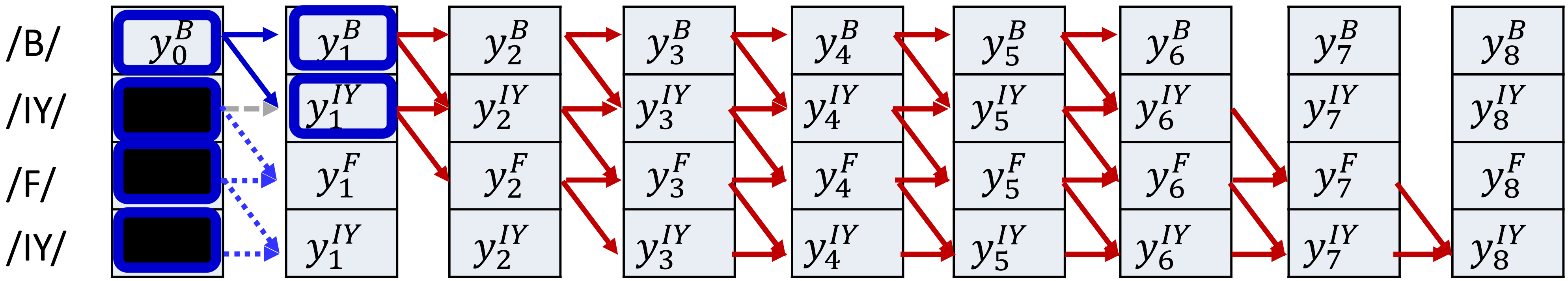
$$BP(0,i) = null, \; i = 0 \dots K-1$$

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

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

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

Viterbi algorithm



- Initialization:

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

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

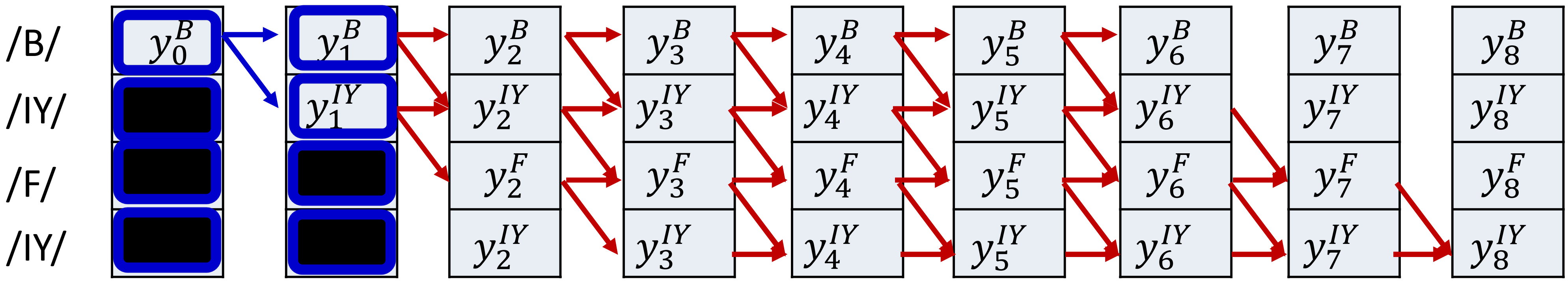
- for $t \ = \ 1 \ldots T-1$

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

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

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

Viterbi algorithm



- Initialization:

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

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

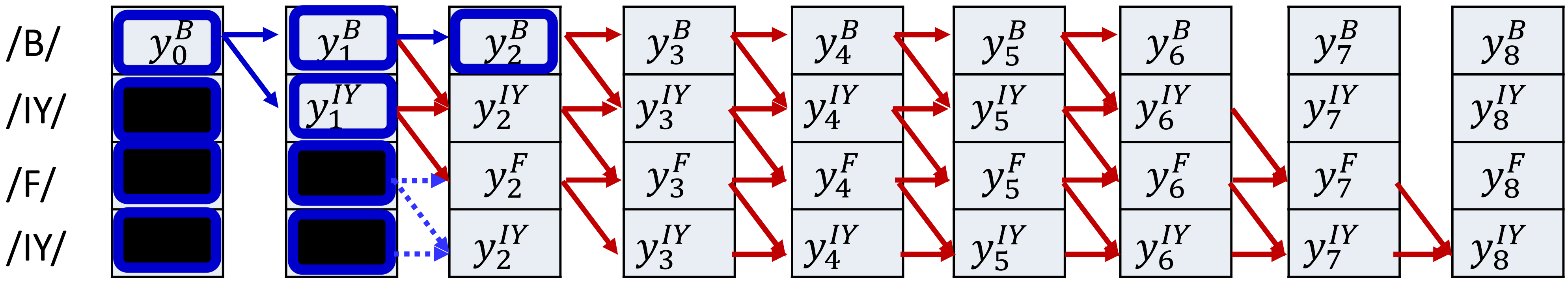
- for $t \ = \ 1 \ldots T-1$

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

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:

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

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

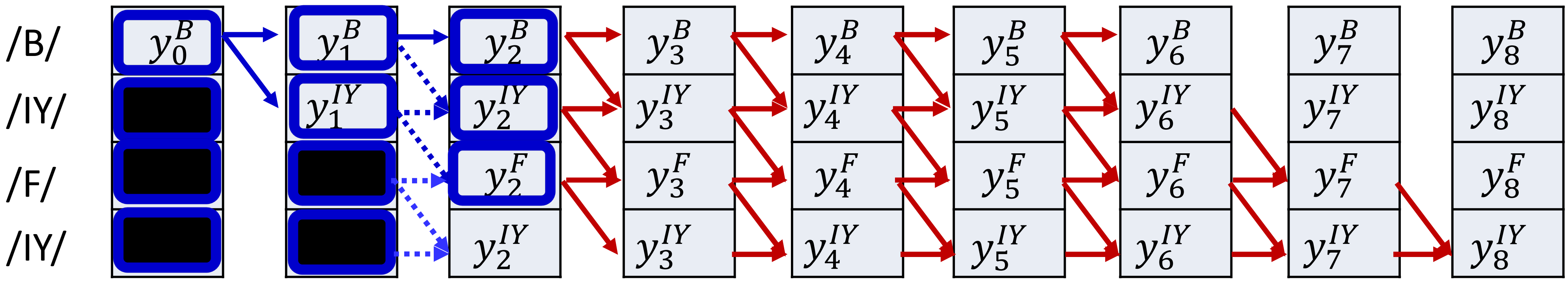
- for $t \ = \ 1 \dots T-1$

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

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

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

Viterbi algorithm



- Initialization:

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

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

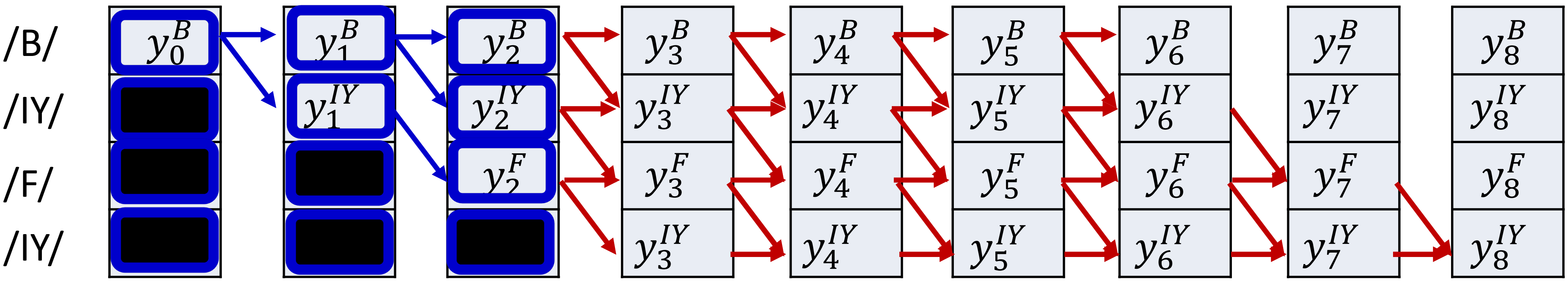
- for $t = 1 \ldots T-1$

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

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:

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

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

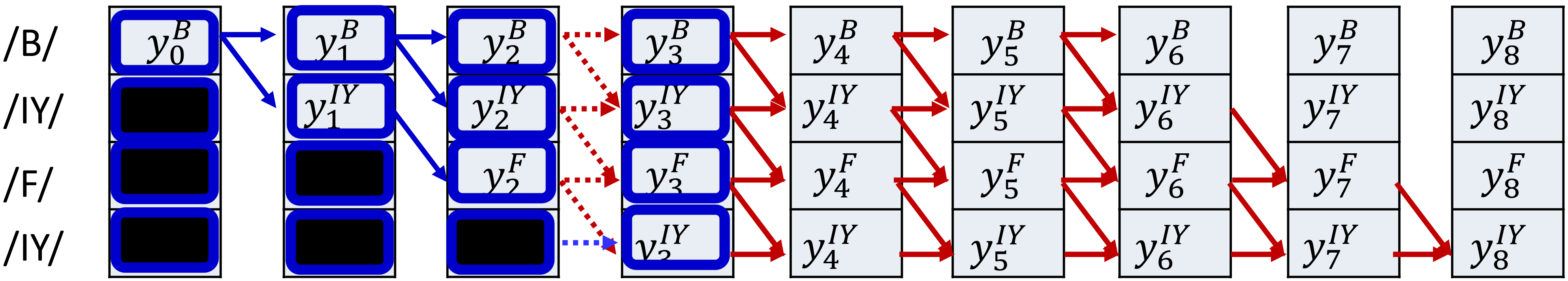
- for $t \ = \ 1 \ldots T-1$

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

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:

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

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

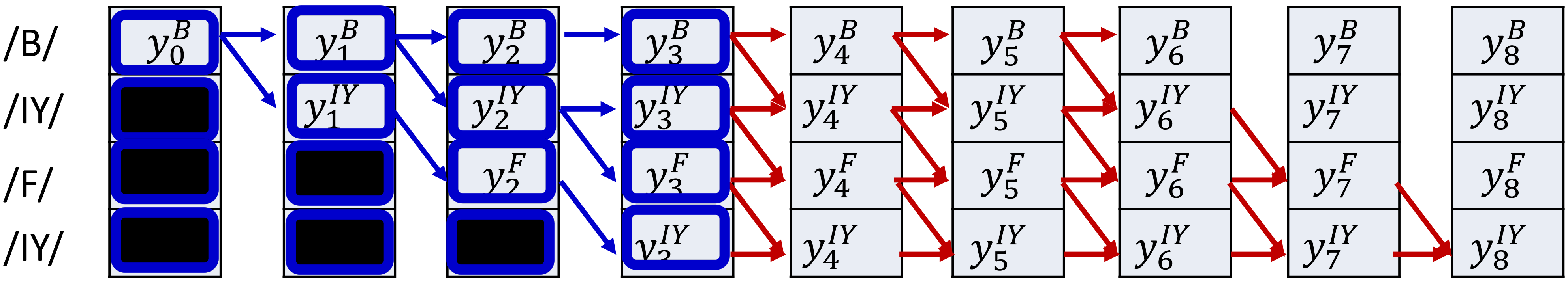
- for $t = 1 \dots T-1$

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

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:

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

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

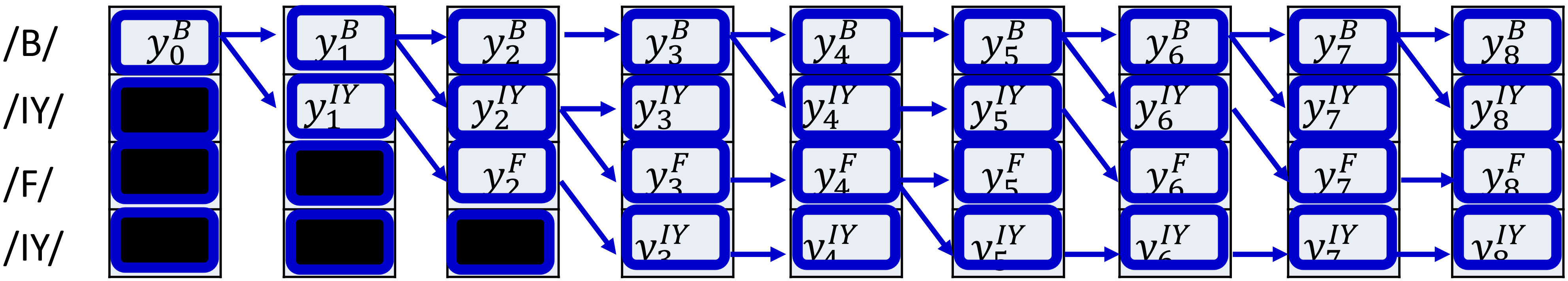
- for $t \ = \ 1 \dots T-1$

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

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:

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

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

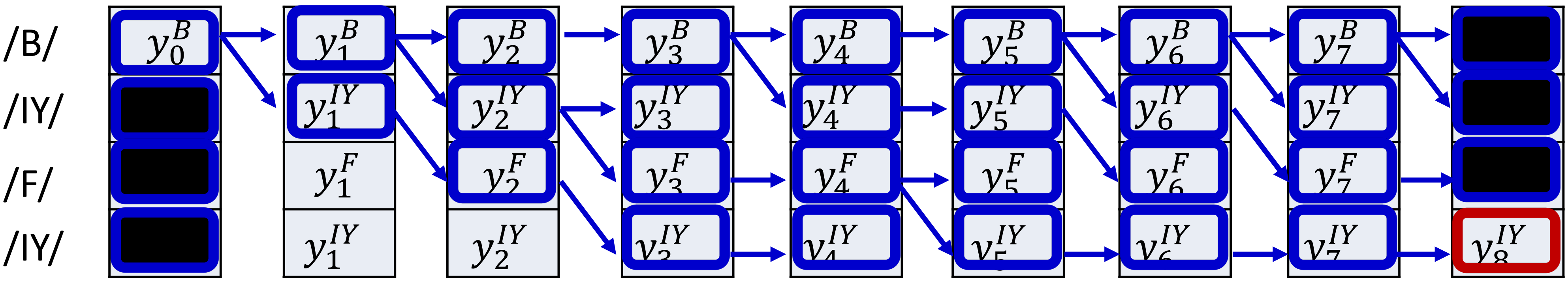
- for $t \; = \; 1 \ldots T-1$

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

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

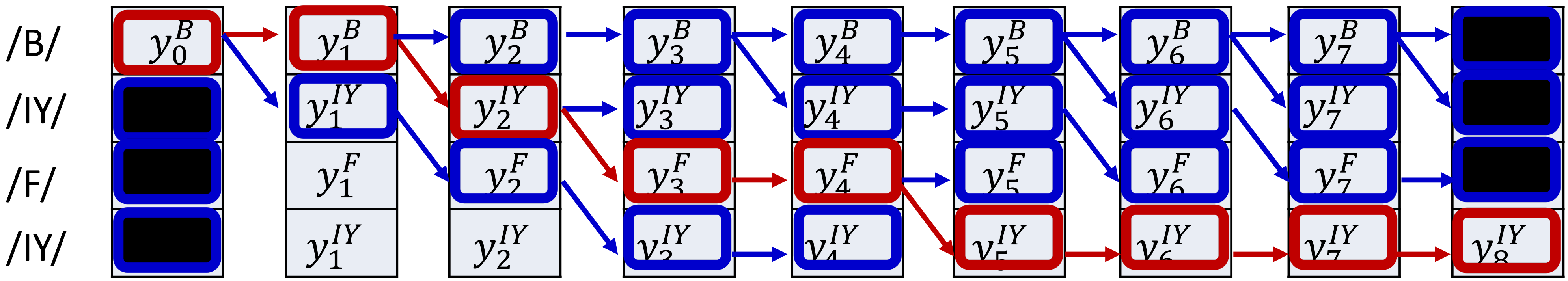
- $BP(t,l) = \begin{pmatrix} l-1: \;\; if \; \big(Bscr(t-1,l-1) > \; Bscr(t-1,l)\big) \; l-1; \\ l : else \end{pmatrix}$
- $Bscr(t,l) \; = \; Bscr(BP(t,l)) \; \times y_t^{S(l)}$

Viterbi algorithm



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

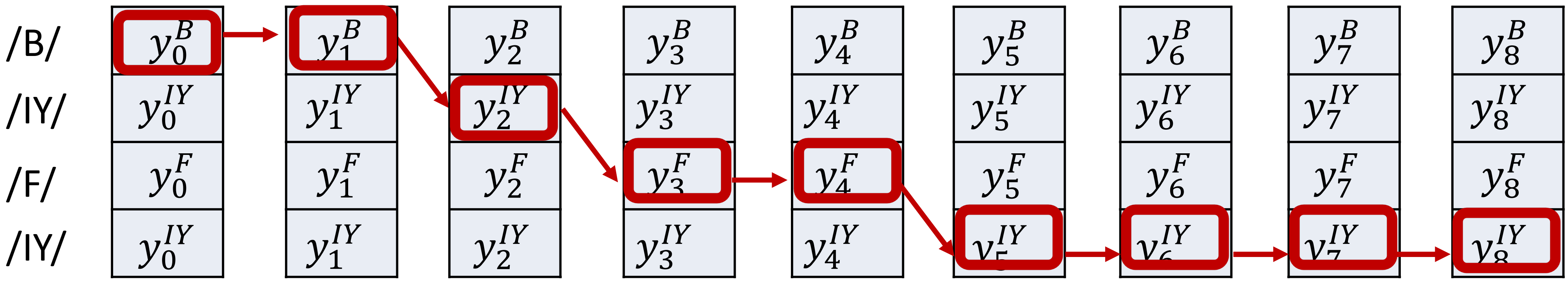
Viterbi algorithm



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

Poll 4

- @

Select all that are true about Viterbi decoding

- It finds the most probable alignment of a compressed (order-synchronous) sequence to an input
- Viterbi decoding is run on a table of probabilities constructed for the compressed sequence, with one row for each symbol in the sequence, derived from the probability table generated by from the output of the recurrent network
- Viterbi decoding selects the most probable symbol from each column of the table

Poll 4

Select all that are true about Viterbi decoding

- **It finds the most probable alignment of a compressed (order-synchronous) sequence to an input**
- **Viterbi decoding is run on a table of probabilities constructed for the compressed sequence, with one row for each symbol in the sequence, derived from the probability table generated by from the output of the recurrent network**
- Viterbi decoding selects the most probable symbol from each column of the table

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) = 0
for t = 2:T
    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 2
    AlignedSymbol(t-1) = BP(t,AlignedSymbol(t))
```

Using 1..N and 1..T indexing, instead of 0..N-1, 0..T-1, for convenience of notation

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) = 0
for t = 2:T
    BP(t,1) = 1;
    Bscr(t,1) = Bscr(t-1,1)*s(t,1)
    for i = 2: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 2
    AlignedSymbol(t-1) = BP(t,AlignedSymbol(t))
```

Do not need explicit construction of output table

Information about order already in symbol sequence S(i), so we can use y(t,S(i)) instead of composing s(t,i) = y(t,S(i)) and using s(t,i)

Using 1..N and 1..T indexing, instead of 0..N-1, 0..T-1, for convenience of notation

VITERBI

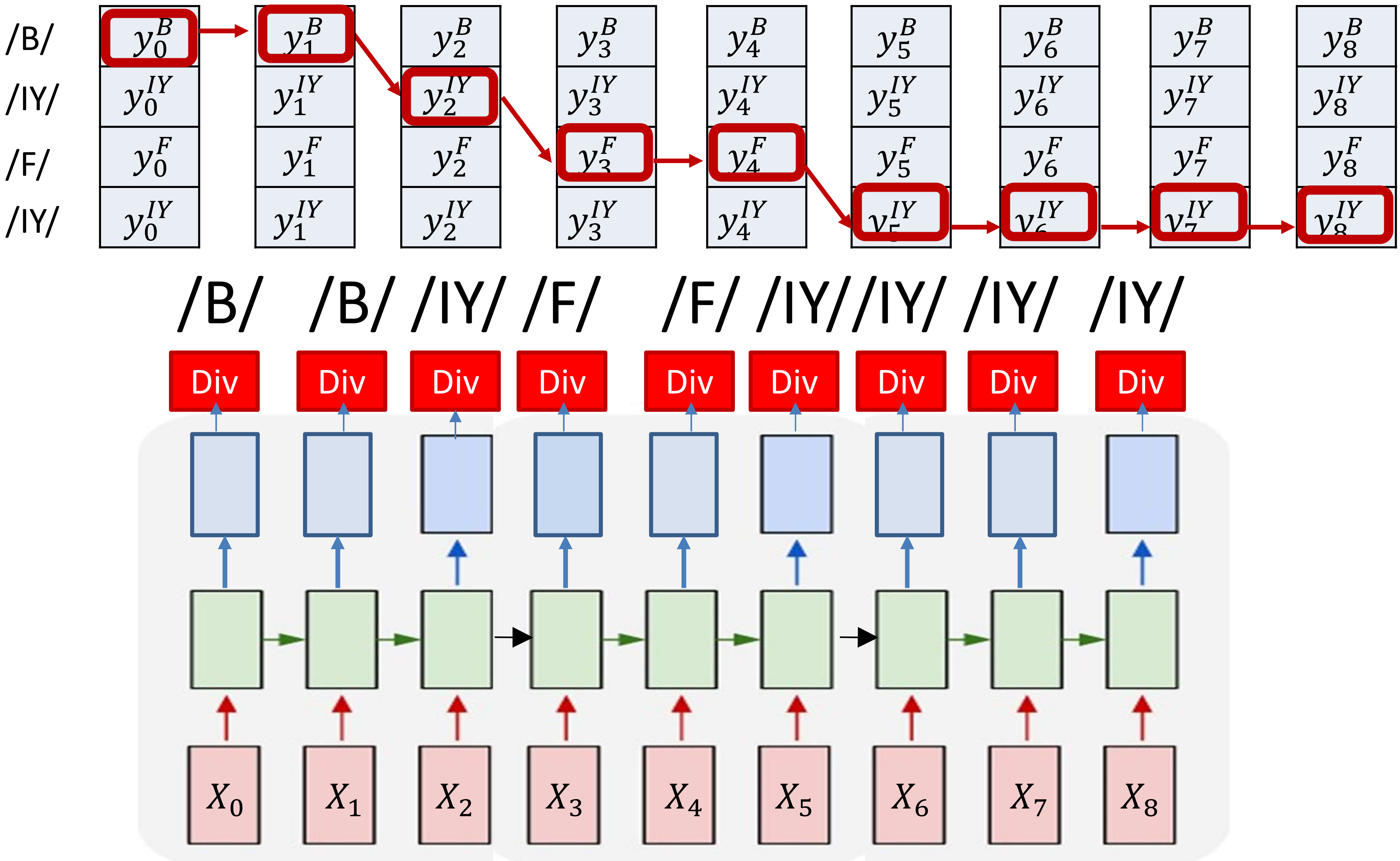
```
#N is the number of symbols in the target output
#S(i) is the ith symbol in target output
#T = length of input
```

Without explicit construction of output table

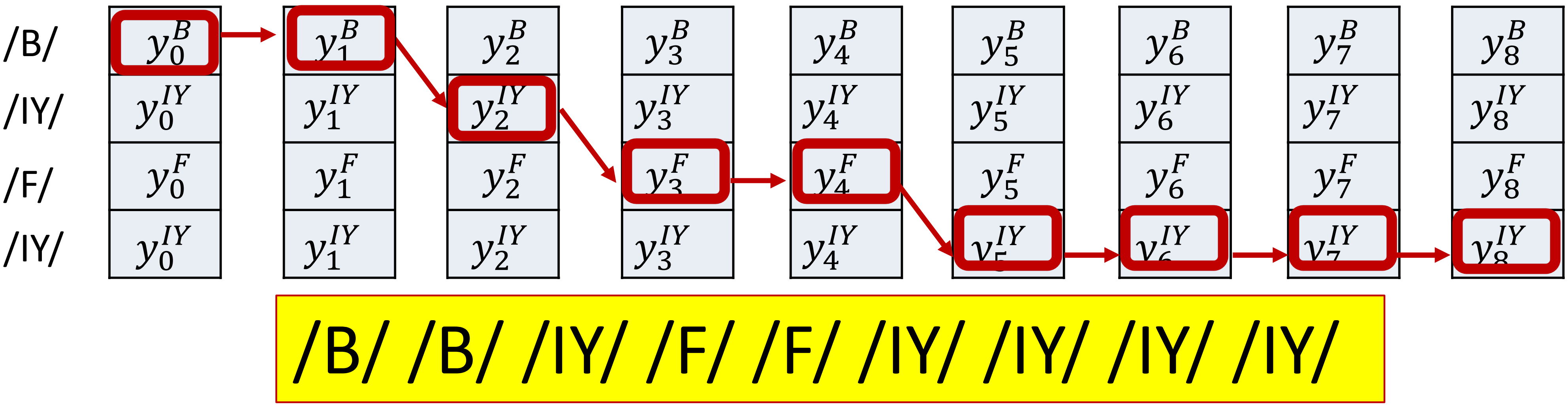
```
# First, at t = 1
BP(1,1) = -1
Bscr(1,1) = y(1,S(1))
Bscr(1,2:N) = 0
for t = 2:T
    BP(t,1) = 1;
    Bscr(t,1) = Bscr(t-1,1)*y(t,S(1))
    for i = 2:min(t,N)
        BP(t,i) = Bscr(t-1,i) > Bscr(t-1,i-1) ? i : i-1
        Bscr(t,i) = Bscr(t-1,BP(t,i))*y(t,S(i))

# Backtrace
AlignedSymbol(T) = N
for t = T downto 2
    AlignedSymbol(t-1) = BP(t,AlignedSymbol(t))
```

Using 1..N and 1..T indexing, instead of 0..N-1, 0..T-1, for convenience of notation

Assumed targets for training with the Viterbi algorithm



Gradients from the alignment



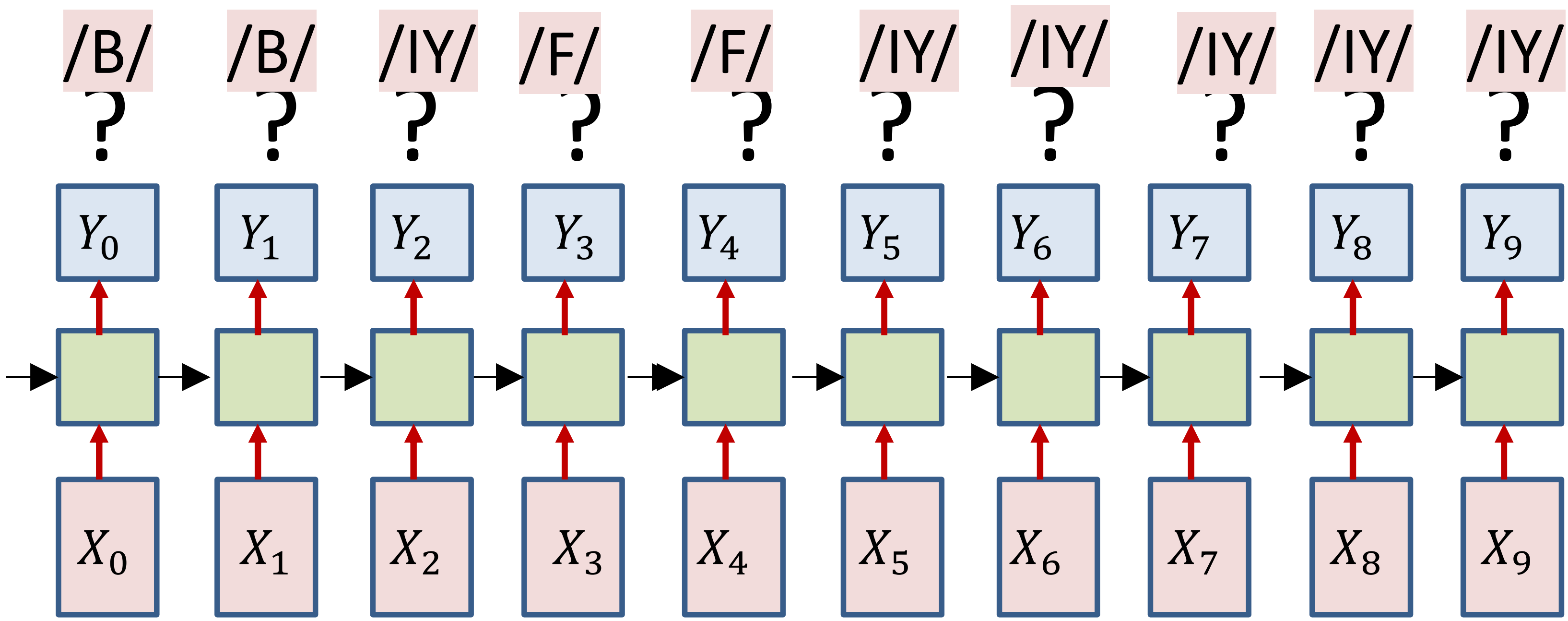
$$DIV = \sum_t KL\left(Y_t, symbol_t^{bestpath}\right) = -\sum_t \log Y\left(t, symbol_t^{bestpath}\right)$$

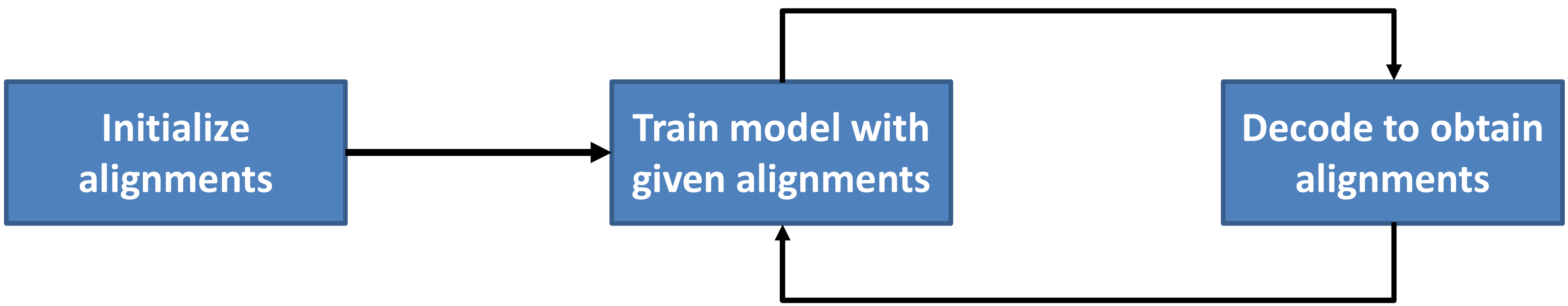
- The gradient w.r.t the t-th output vector Y_t

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

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

Iterative Estimate and Training





The "decode" and "train" steps may be combined 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..

Next Class

- Training without explicit alignment..
  - Connectionist Temporal Classification
  - Separating repeated symbols
- The CTC decoder..