Natural Language Processing

Stateless RNN

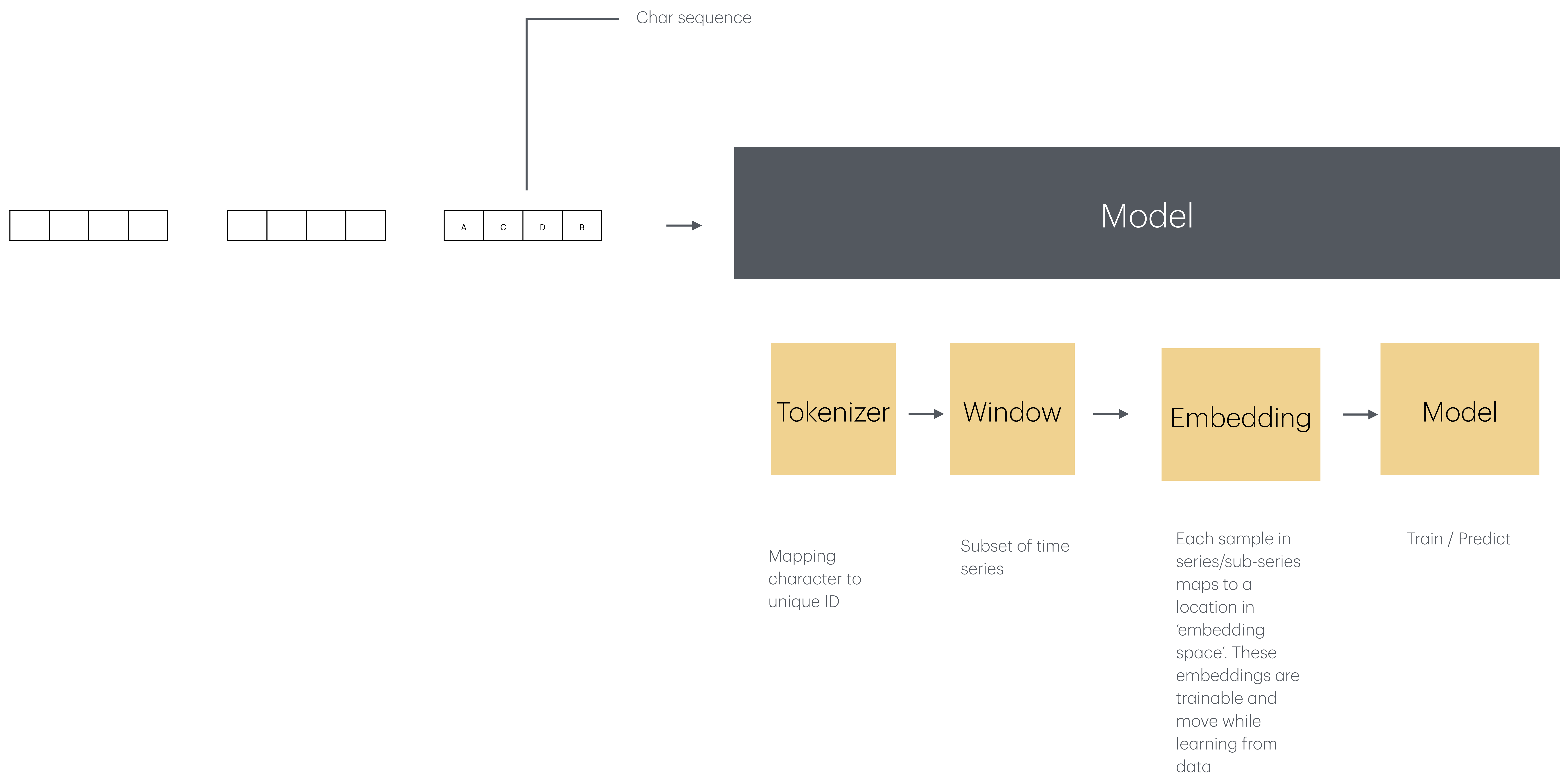
Portions of text randomly learned without info about text.
Each step starts learning from scratch

Stateful RNN

Portions of text patterns stored in memory and model continues to learn from those learning points. Learned history is preserved between training steps.

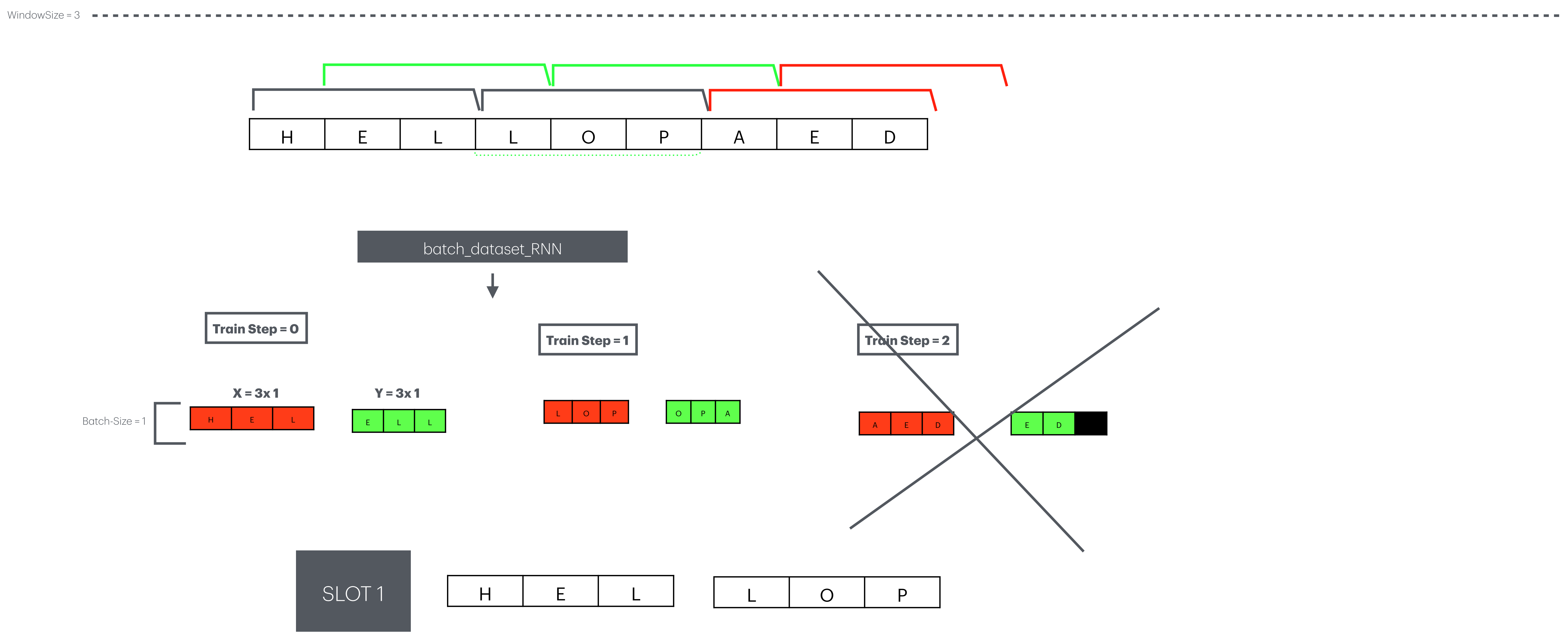
Learning from past experience to support future skills learned from fighting bosses

CHAR RNN



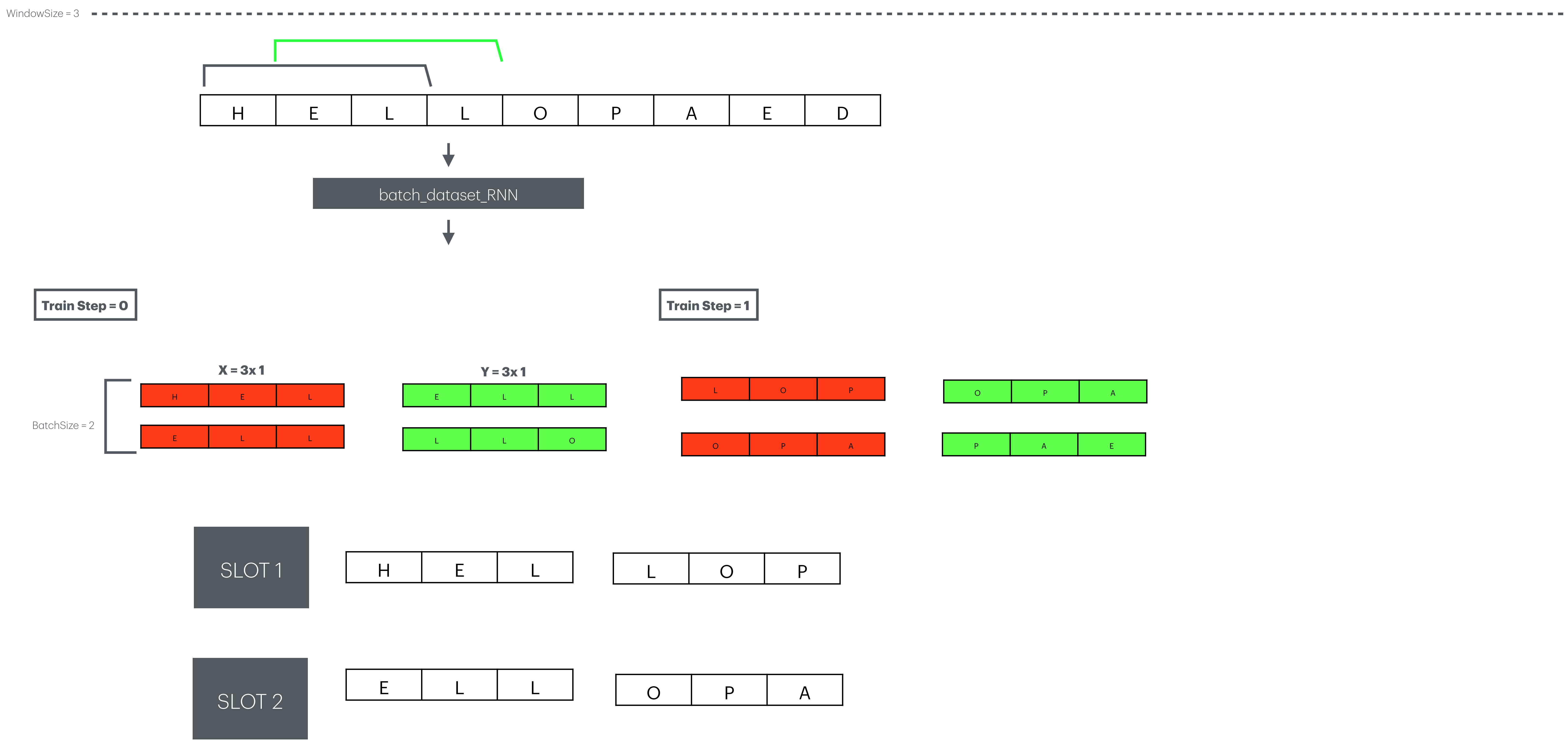
Char sequence
A
C
D
B
Model
Tokenizer
Window
Embedding
Model
Mapping character to unique ID
Subset of time series
Each sample in series/sub-series maps to a location in 'embedding space'. These embeddings are trainable and move while learning from data
Train / Predict

Stateful/Stateless RNN Preprocessing



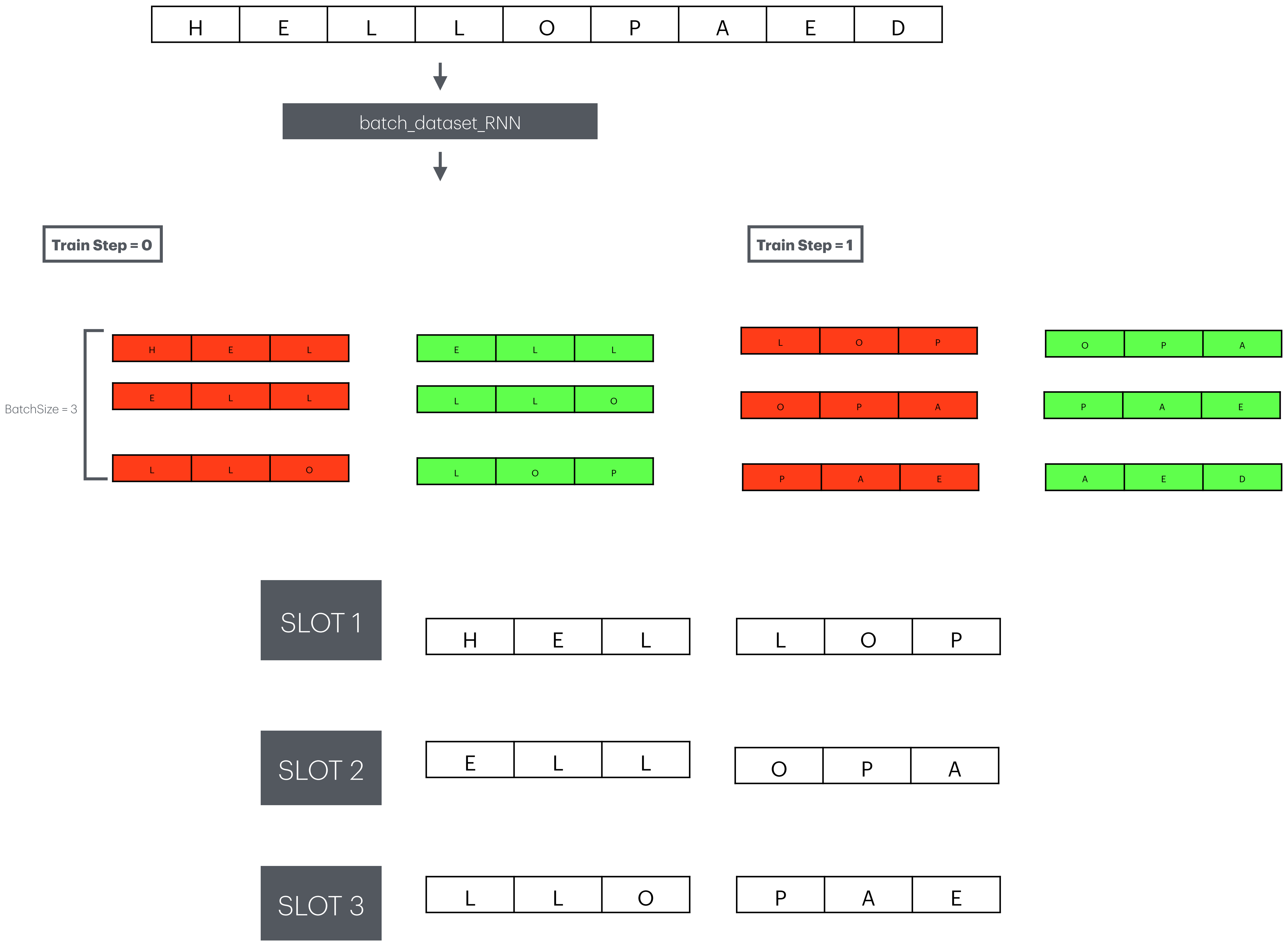
Two train steps ONLY, remaining or remainder is dropped.
The remainder is caused by the structure of the future batched data.
Each batch subset must be samples following/after the previous time step batch.
From the perspective of a batch slot, each batch slot's samples in a contiguous flow of the input data series

Stateful/Stateless RNN Preprocessing



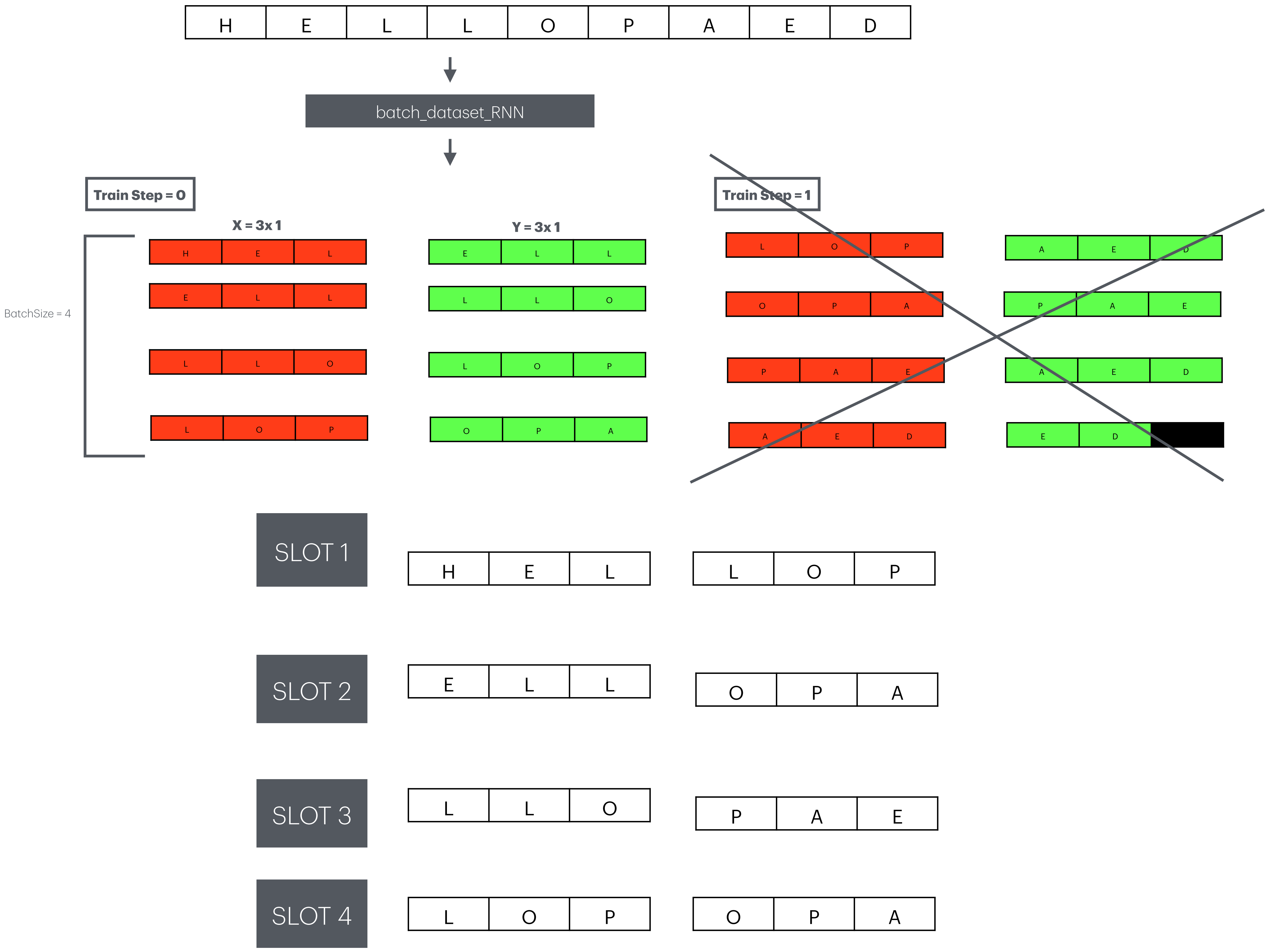
Two train steps ONLY, remaining or remainder is dropped.

Stateful/Stateless RNN Preprocessing



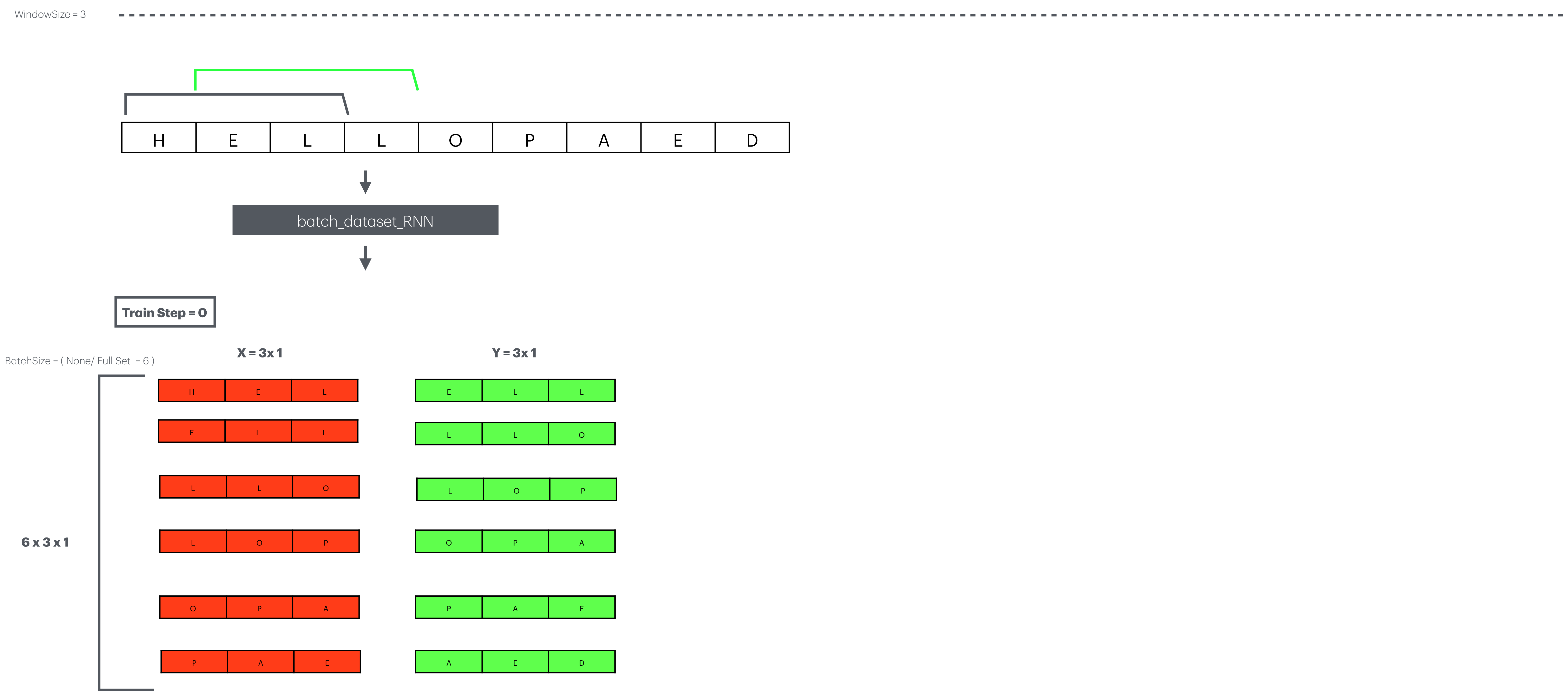
Two steps ONLY, remaining or remainder is dropped.
The remainder is caused by the structure of the future batched data.

Stateful/Stateless RNN Preprocessing



One step ONLY, remaining or remainder is dropped.

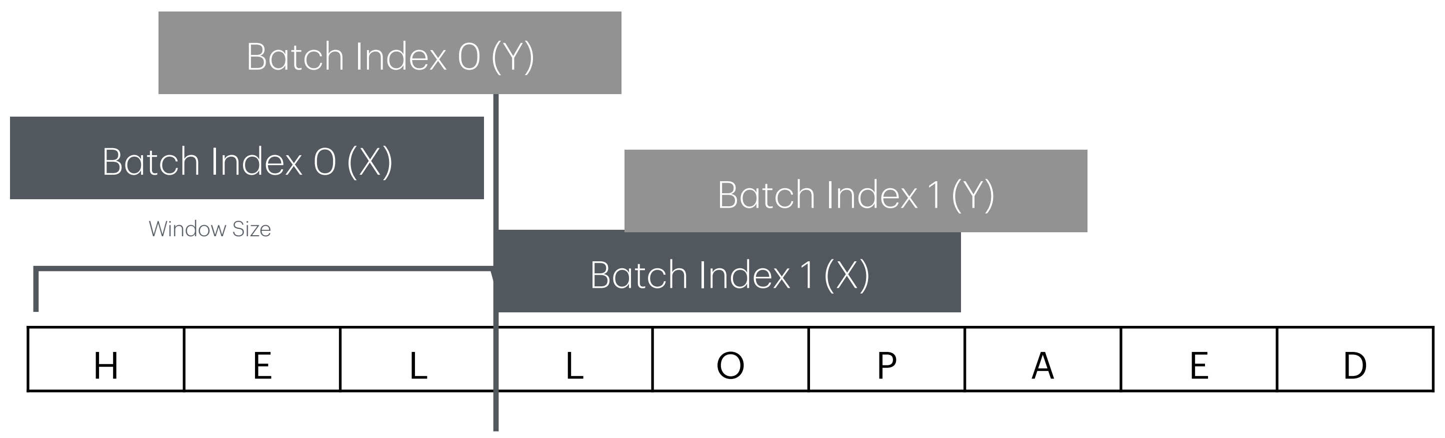
Stateful/Stateless RNN Preprocessing



Processing FullSet is not common, as it is greedy to RAM

One step ONLY, remaining or remainder is dropped.

Stateful/Stateless RNN Preprocessing



Batch Size = 1

[0 1 2]   [3 4 5]

Field View = 5 Neurons
Required for each batch

$$N_{steps} = 1 + \frac{text_{length} - window_{length}}{f_{view} + 1}$$

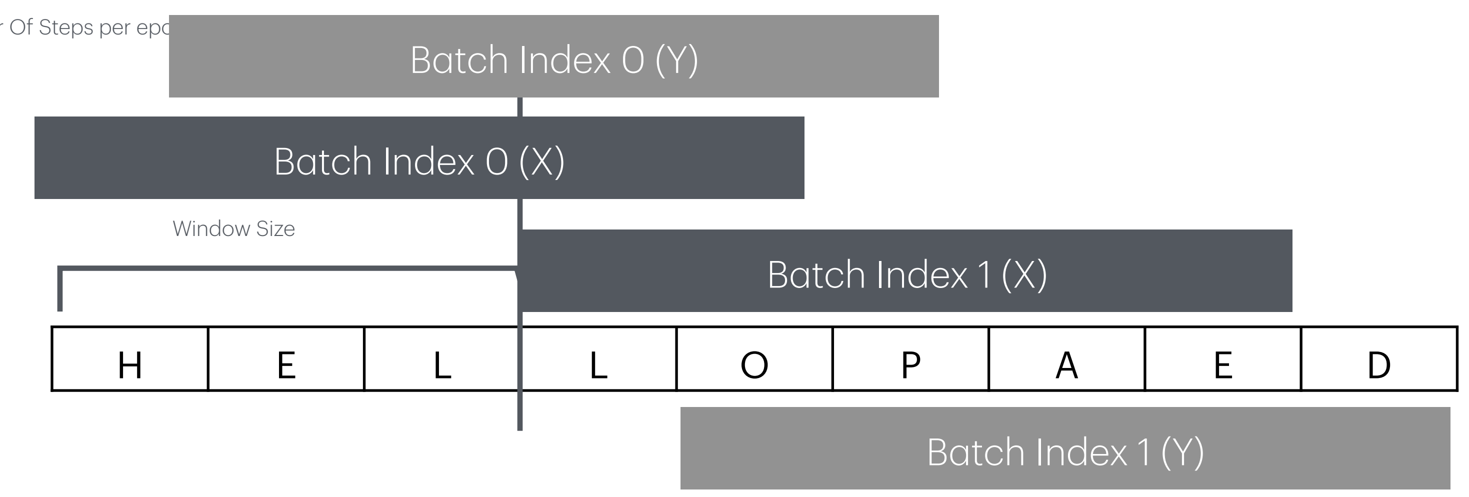
$$N_{steps} = 1 + \frac{9 - 3}{3 + 1}$$

$$N_{steps} = 1 + floor(\frac{9 - 3}{3 + 1})$$

$$N_{steps} = 1 + 1 = 2$$

Stateful/Stateless RNN

Calculate Number Of Steps per epo



Batch Size = 3

$$\begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 3 \\ 2 & 3 & 4 \end{bmatrix} \qquad \begin{bmatrix} 3 & 4 & 5 \\ 4 & 5 & 6 \\ 5 & 6 & 7 \end{bmatrix}$$

Field View = 5 Neurons
Required for each batch

$$N_{steps} = 1 + \frac{text_{length} - window_{length}}{f_{view} + 1}$$

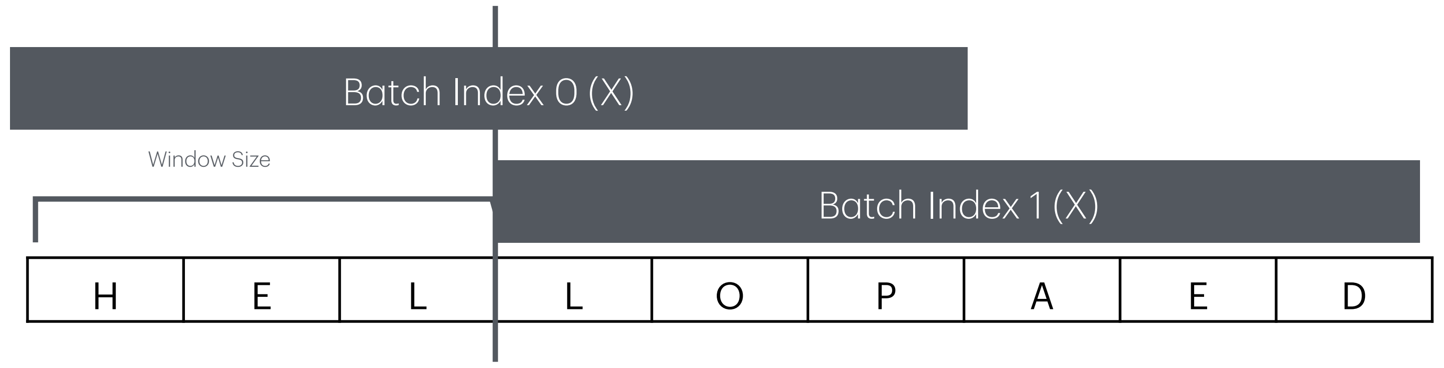
$$N_{steps} = 1 + \frac{9 - 3}{5 + 1}$$

$$N_{steps} = 1 + floor(\frac{9 - 3}{5 + 1})$$

$$N_{steps} = 1 + 1 = 2$$

Stateful/Stateless RNN

Calculate Number Of Steps per epoch



Batch Size = 4

$$\begin{bmatrix} 0 & 1 & 2 \\ 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \end{bmatrix} \qquad \begin{bmatrix} 3 & 4 & 5 \\ 4 & 5 & 6 \\ 5 & 6 & 7 \\ 6 & 7 & 8 \end{bmatrix}$$

Field View = 6 Neurons
Required for each batch

$$N_{steps} = 1 + \frac{text_{length} - window_{length}}{f_{view} + 1}$$

Fview + 1 = Total field view of X and Y sub samples

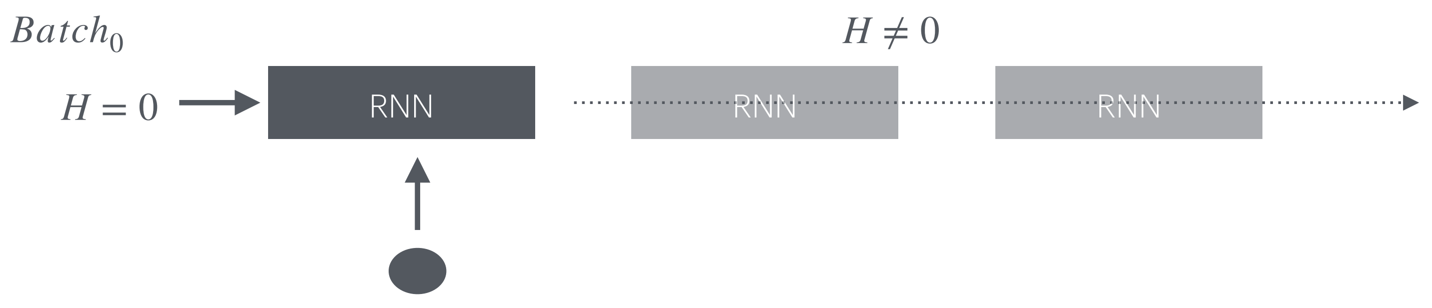
$$N_{steps} = 1 + \frac{9 - 3}{6 + 1}$$

$$N_{steps} = 1 + floor(\frac{9 - 3}{6 + 1})$$

$$N_{steps} = 1 + 0 = 1$$

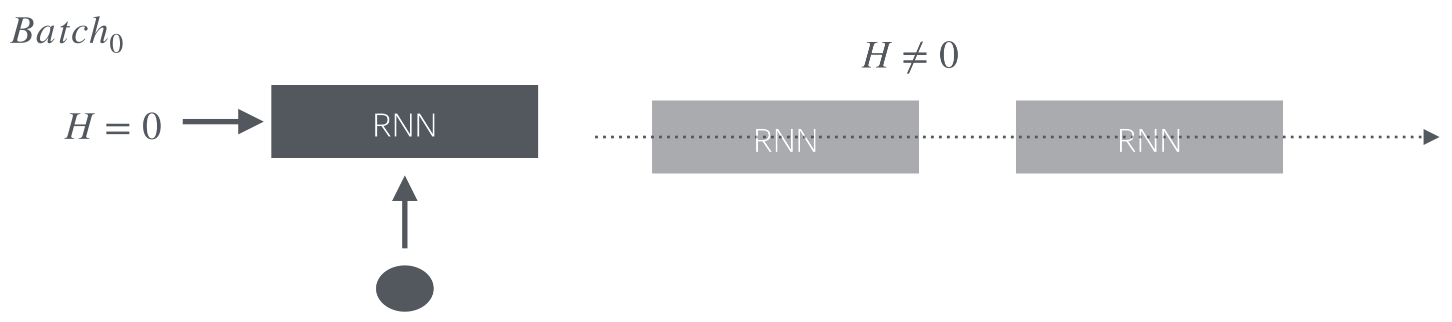
Stateful/Stateless RNN

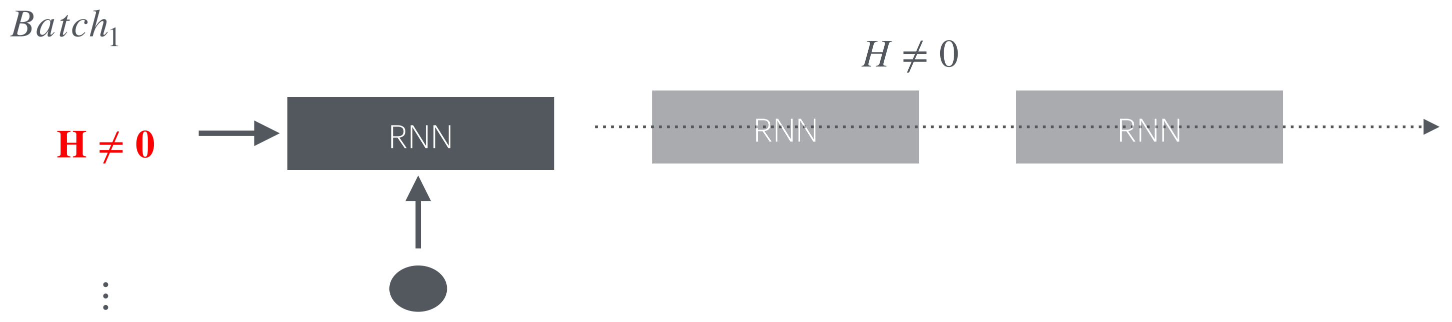
Stateless



After an iteration (unraveling of batch) the hidden states are zeroed

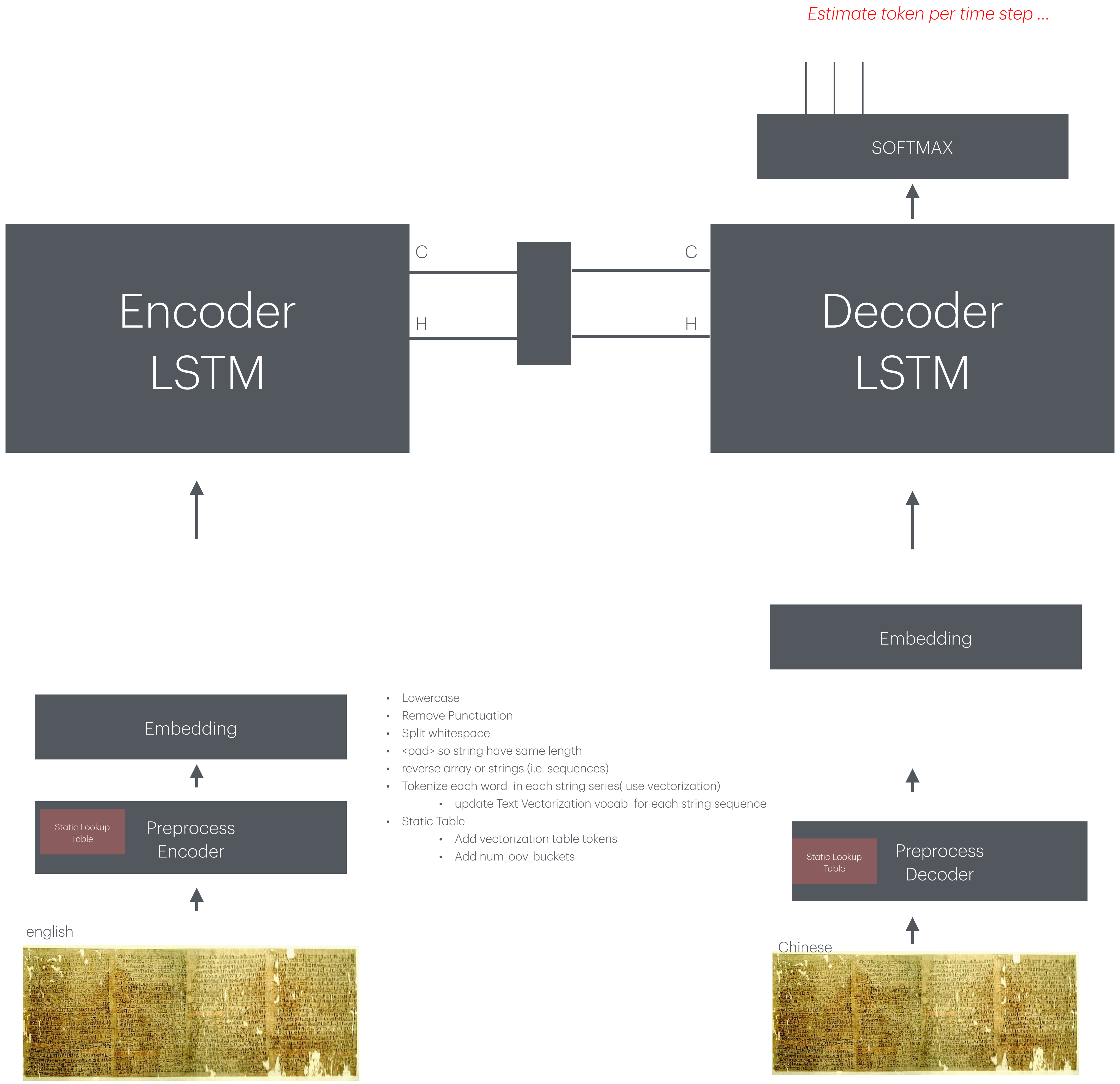
Stateful





After an iteration (unraveling of batch) the hidden states are preserved

Encoder Decoder

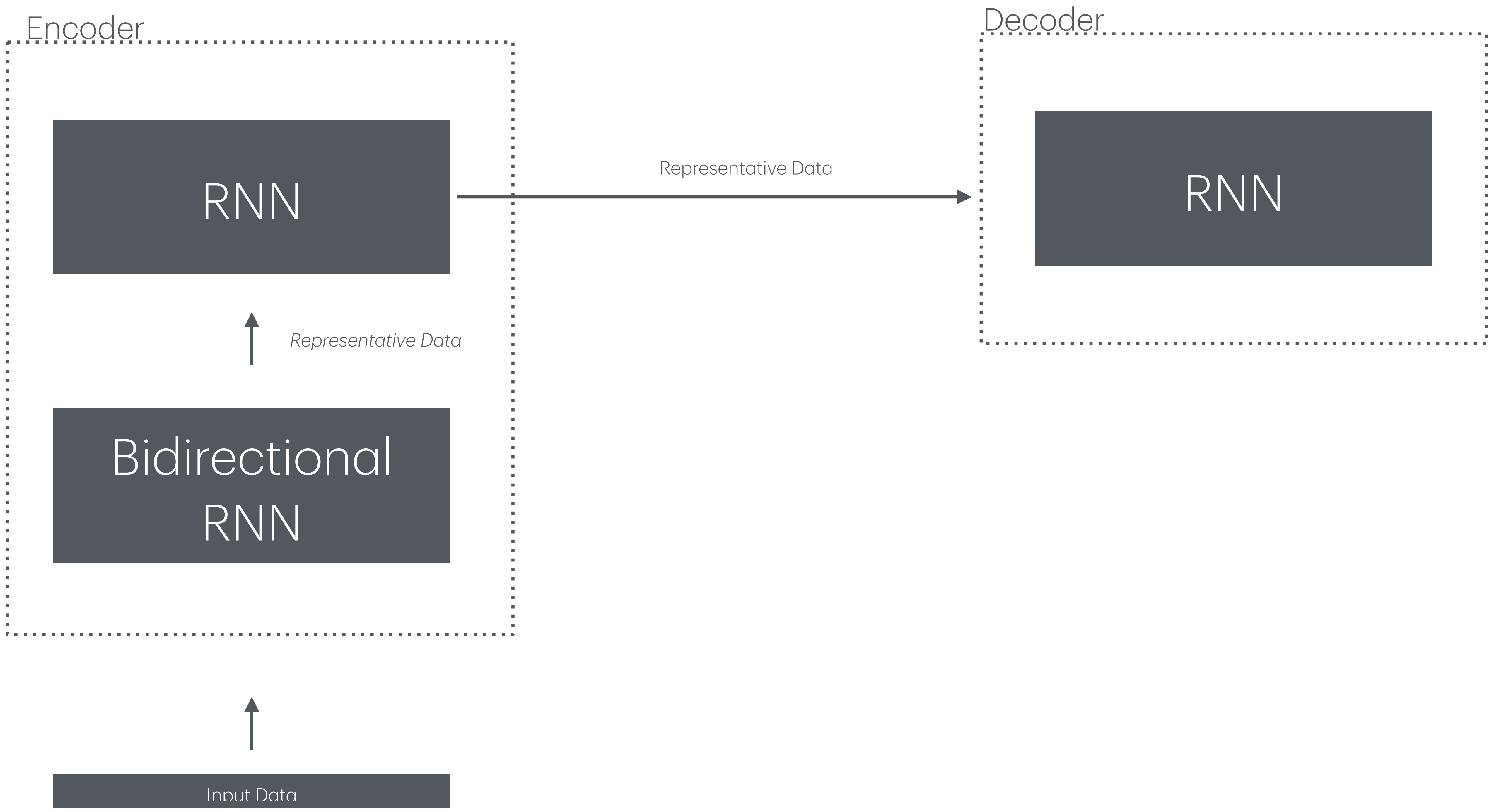


- Lowercase
- Remove Punctuation
- Split whitespace
- <pad> so string have same length
- reverse array or strings (i.e. sequences)
- Tokenize each word in each string series(use vectorization)
    - update Text Vectorization vocab for each string sequence
- Static Table
    - Add vectorization table tokens
    - Add num_oov_buckets

- Lowercase
- Remove Punctuation
- Split whitespace
- <pad> so string have same length
- Head each string series with <sos>
- Tail each string series with <eos>
- Tokenize each word in each string series (use vectorization)
    - update Text Vectorization vocab for each string sequence
- Static Table
    - Add vectorization table tokens
    - Add num_oov_buckets
- Split X, Y (i.e. data and target)
- .....

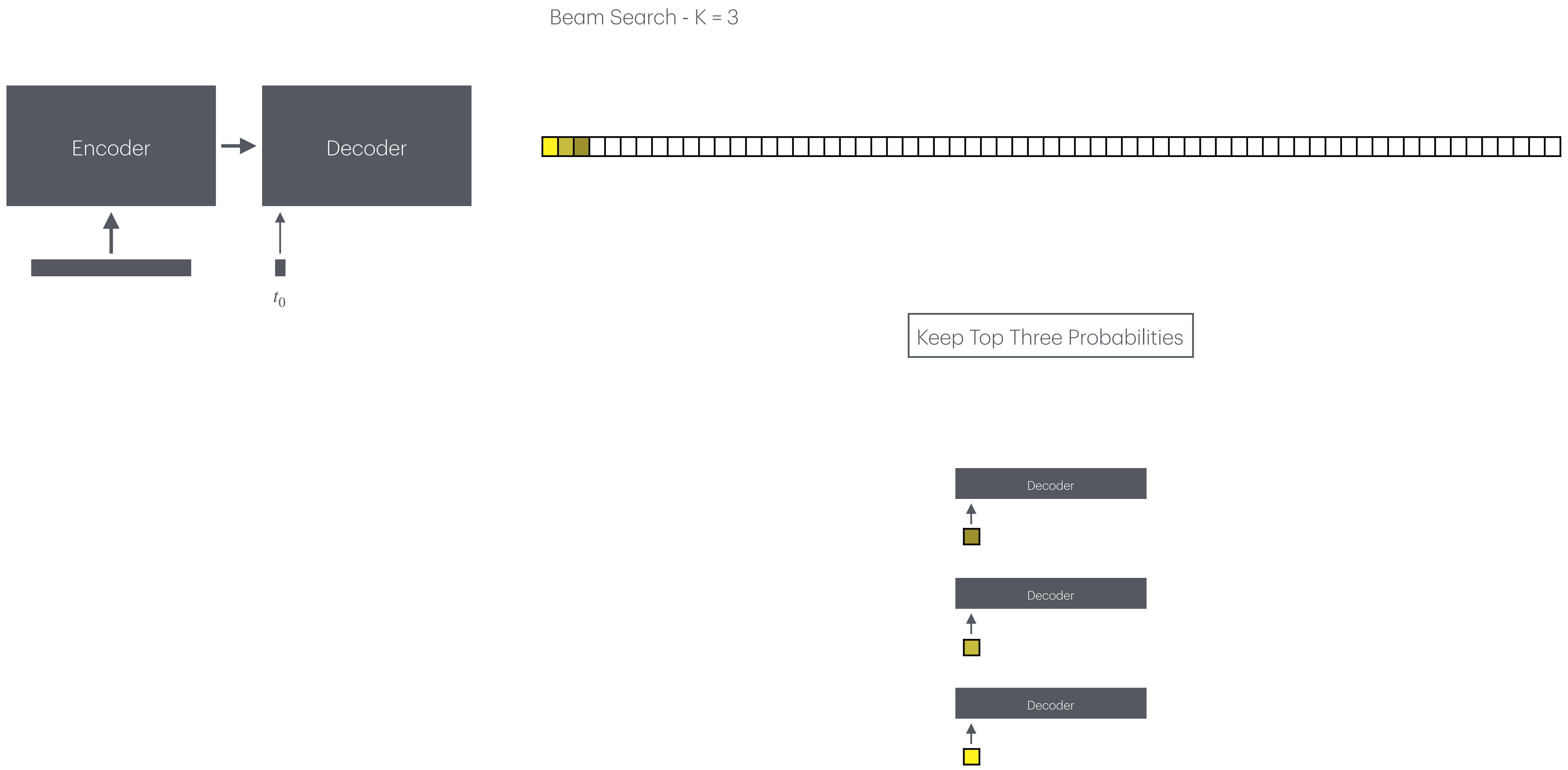
Encoder Decoder

Boost encoding using Bidirectional RNN

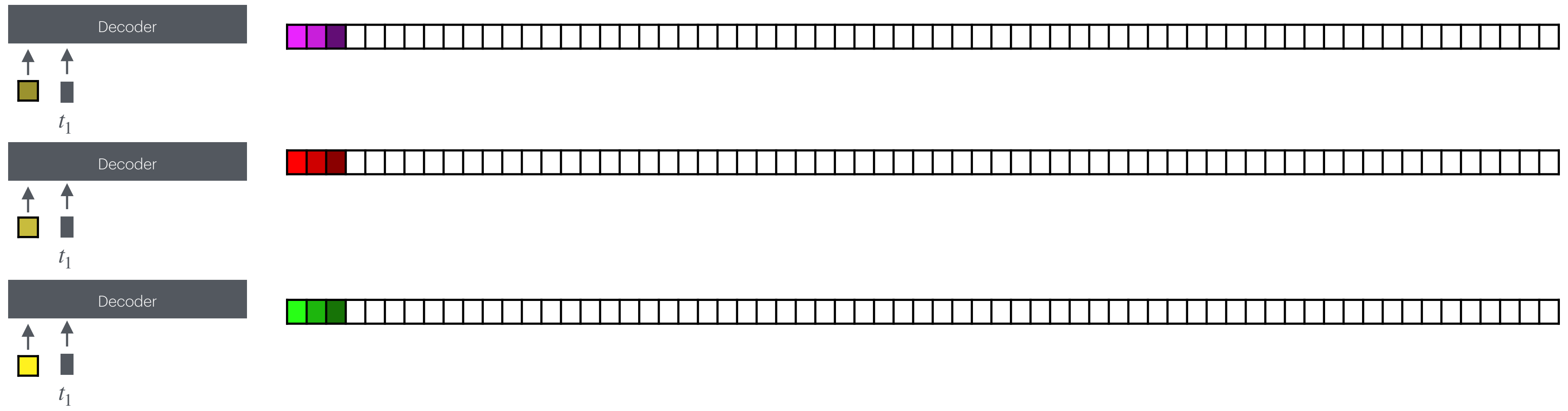


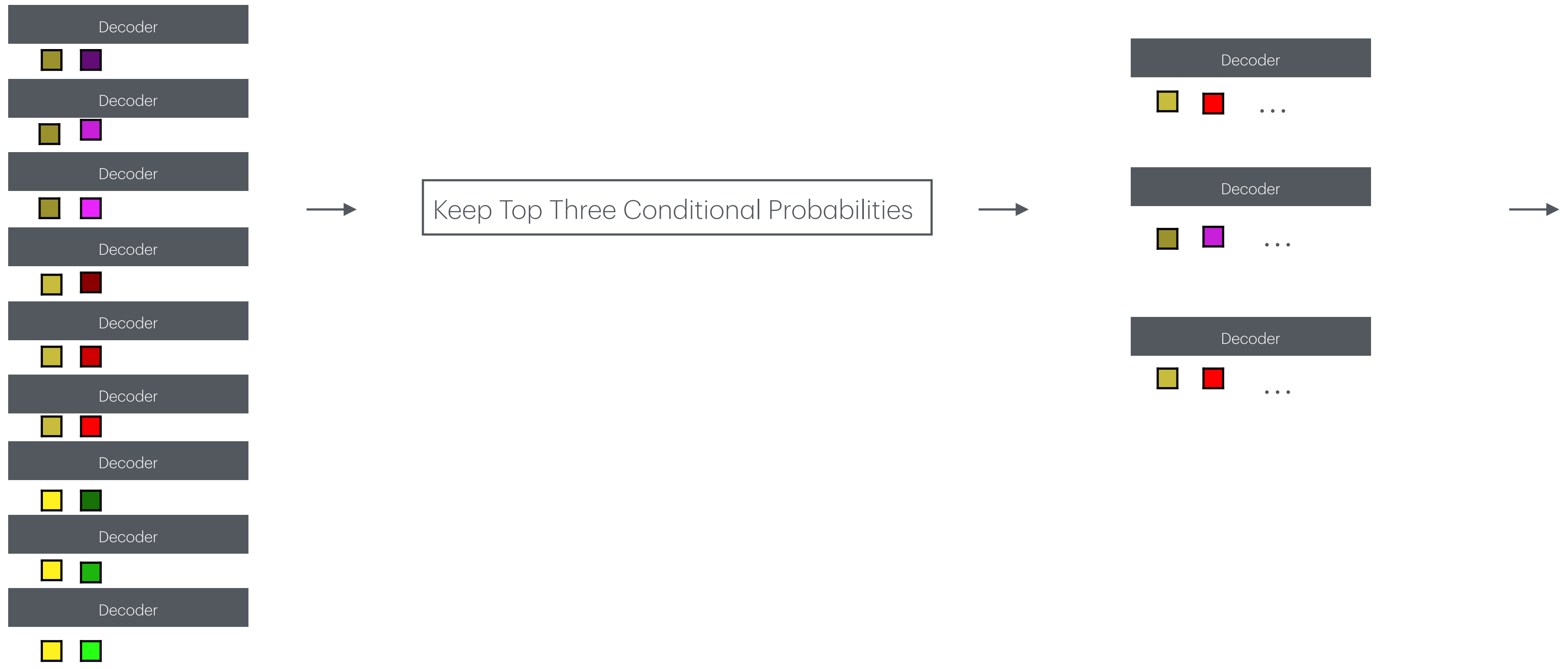
Beam Search

Supports model in finding optimal translation (Boosting model performance)



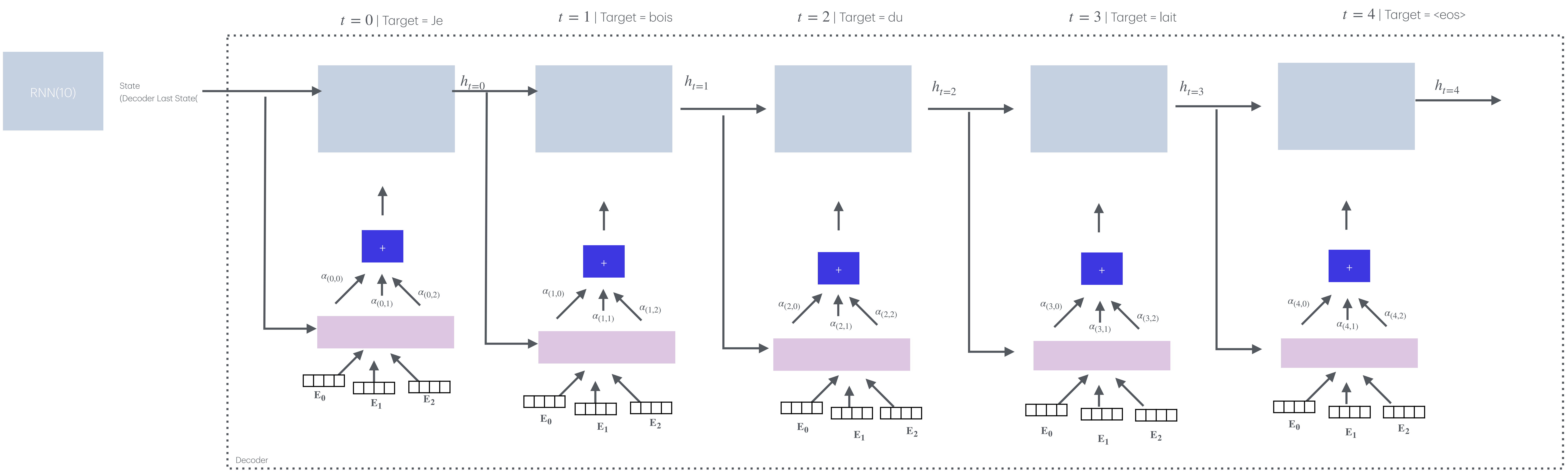
Beam Search

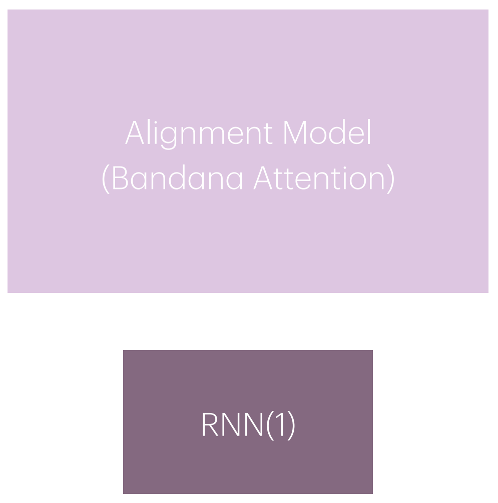


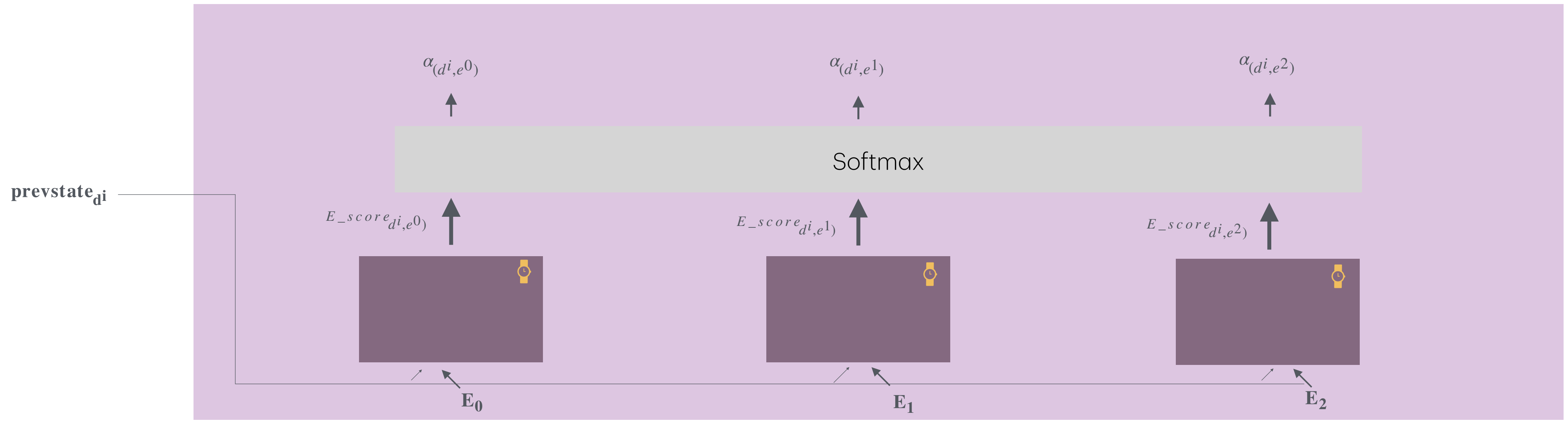


Continue and evaluate until an <EOS> is output. Using Bidirectional encoder and increase the size of decoder RNN neurons to boost predictive performance

Attention Mechanisms (Train - Alignment Method 1)

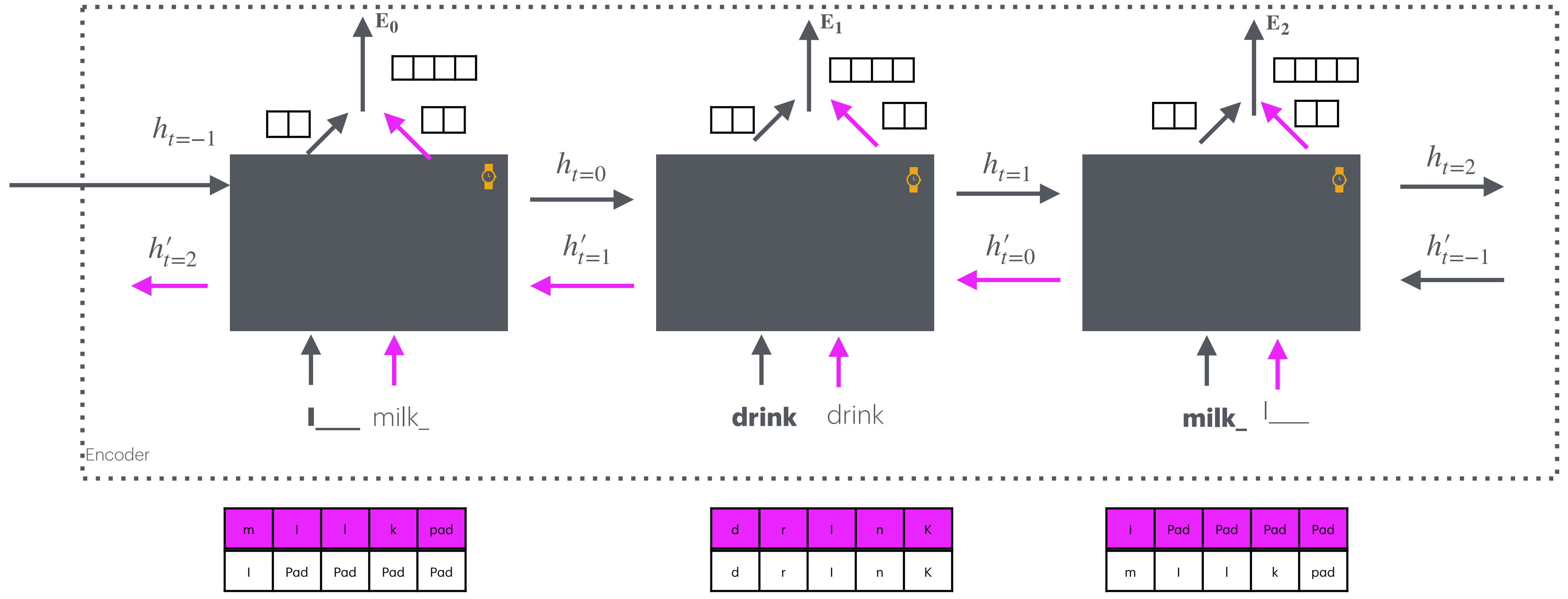






Measure encoder output 'alignment' with previous hidden state

Note: Each time step has unique input but weights are shared. Weights 'learn'. Increase number of neurons to increase learning performance



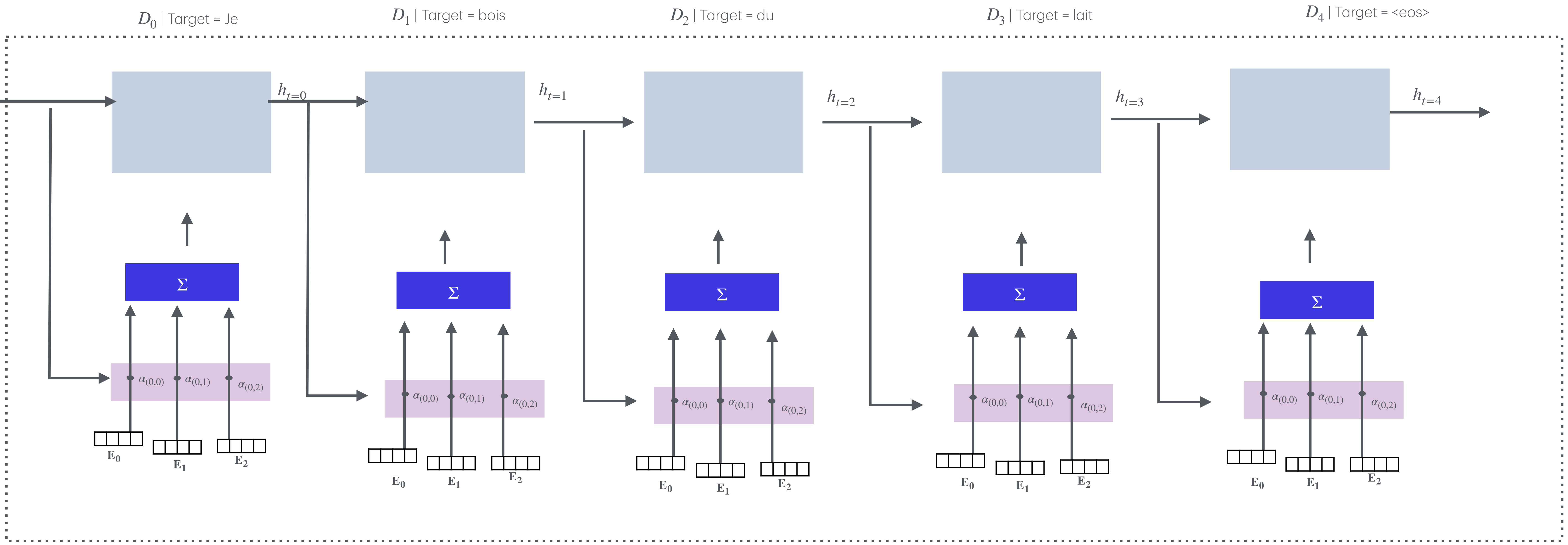
1x3x6

- 5 length vector string per time step
- 3 time steps
- Largest width = 5 (underscore is <pad>)
- Bidirectional
  - Normal Data
  - Reverse Data

Attention Mechanisms (Inference - Alignment Method 1)

RNN(10)

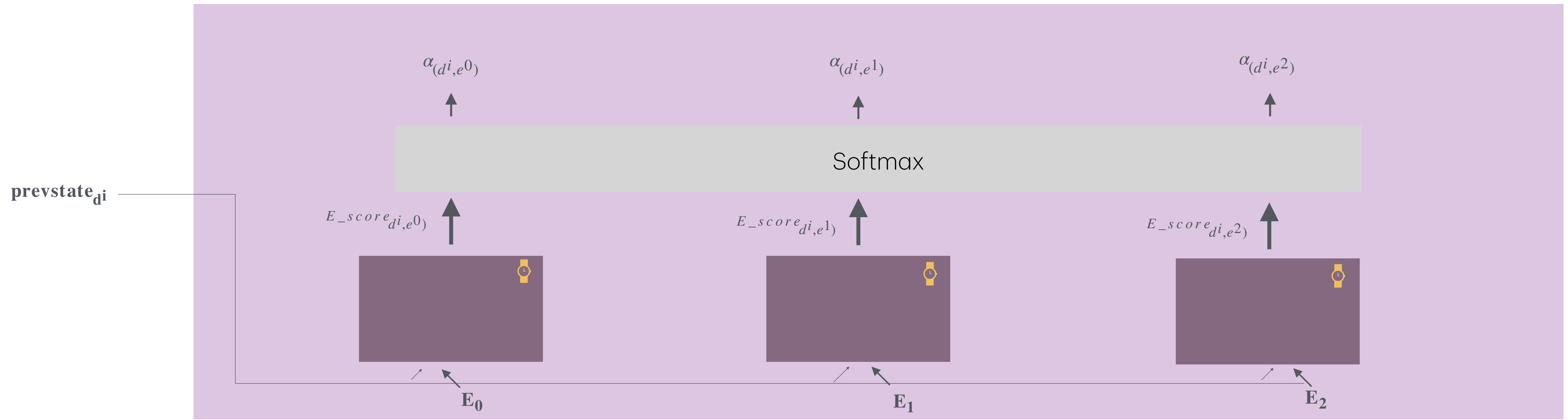
State
(Decoder Last State(



Encoder outputs weighted by attention weights are each decoder time step. At each step the decoder will pay more attention to a certain encoder input if previous state aligns with trained encoder input

Alignment Model
(Bandana Attention)

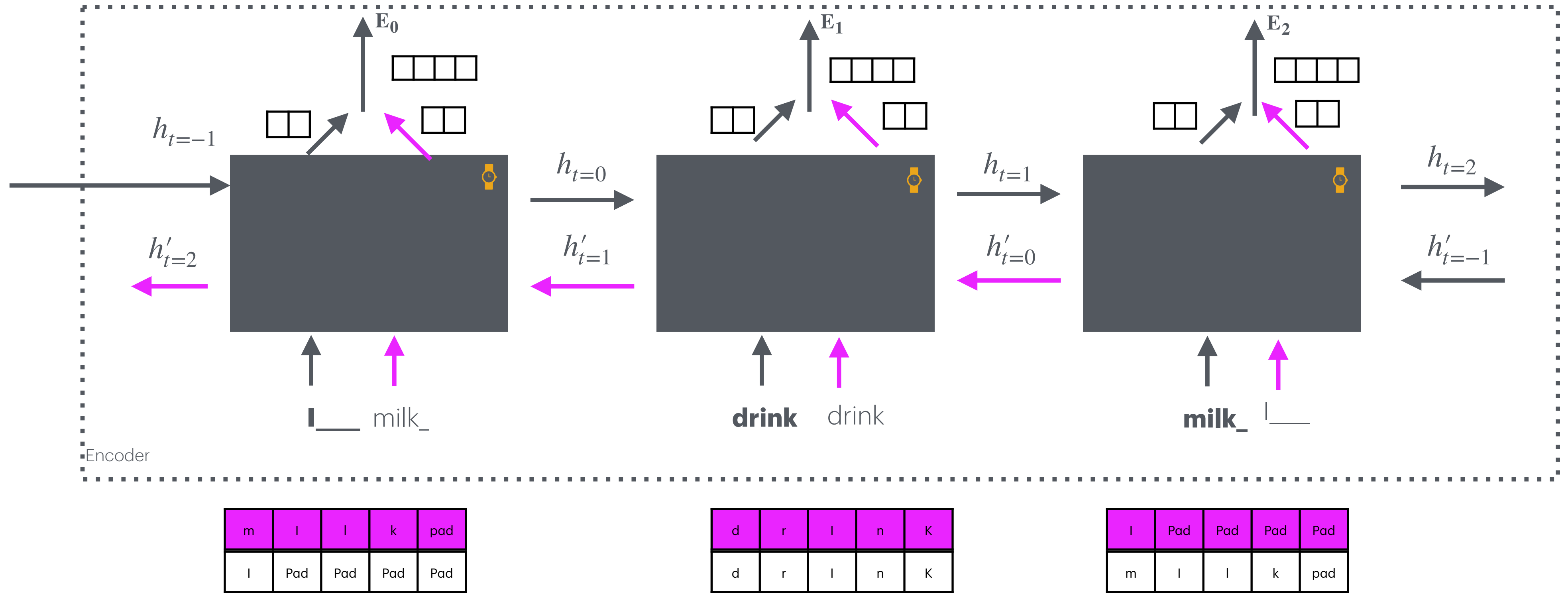
RNN(1)



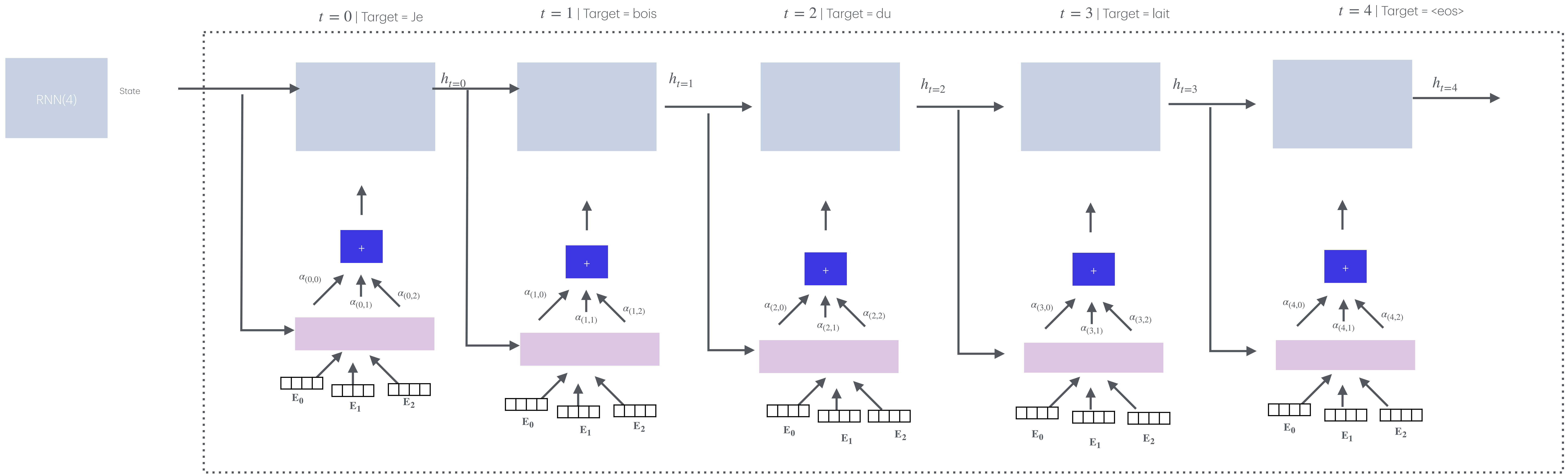
RNN(2)

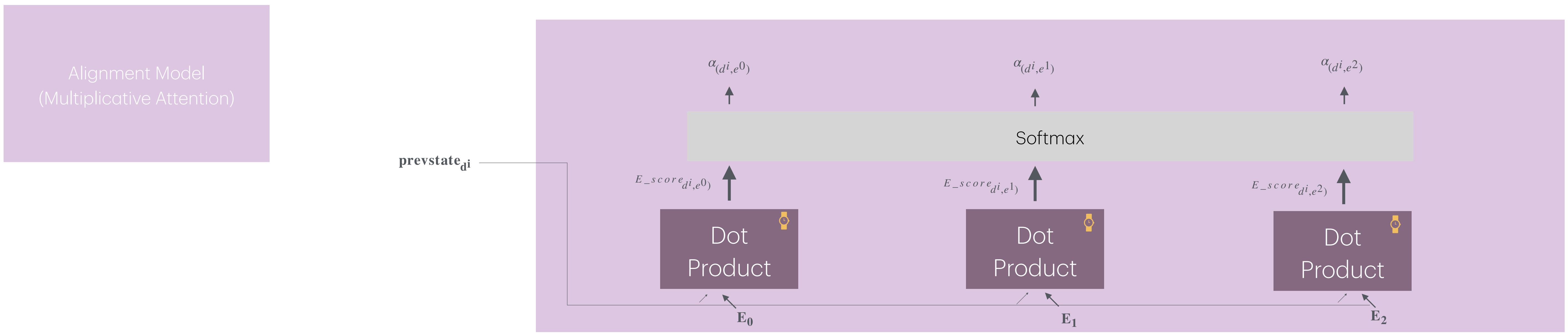
1x3x6

- 5 length vector string per time step
- 3 time steps
- Largest width = 5 (underscore is <pad>)
- Bidirectional
  - Normal Data
  - Reverse Data



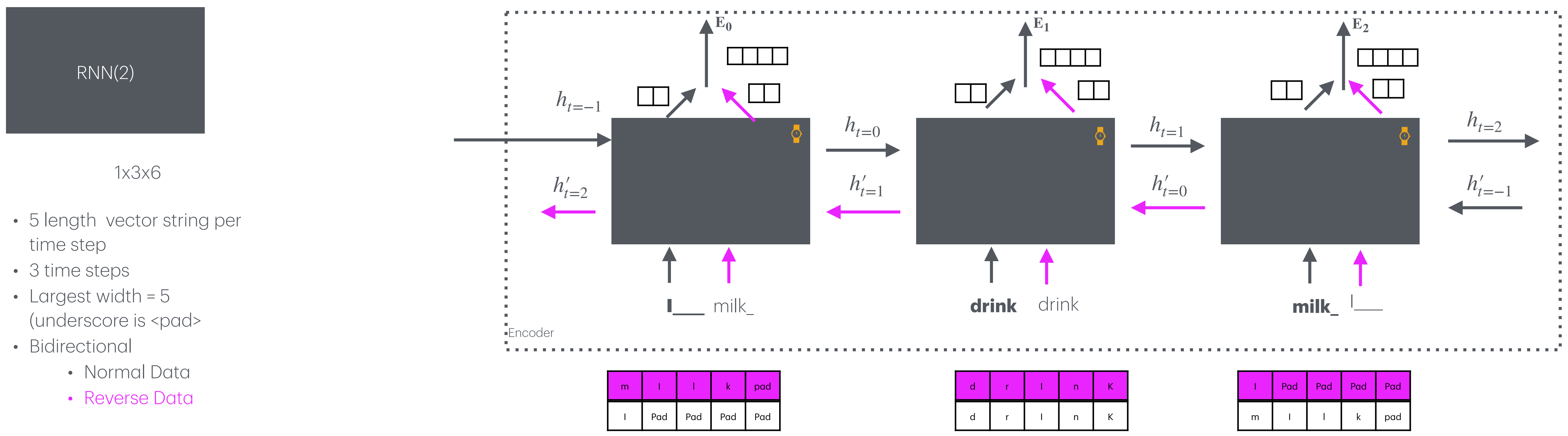
Attention Mechanisms (Train Alignment Method 2)





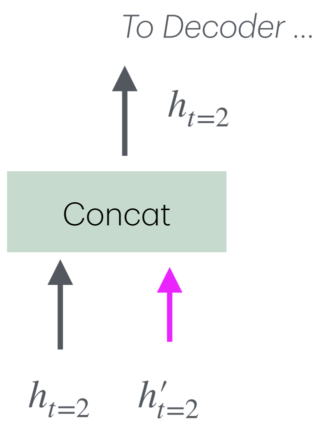
Measures similarity between encoder output 'alignment' and previous hidden state by taking dot product

Note: state and encoder input must have the same dimensions



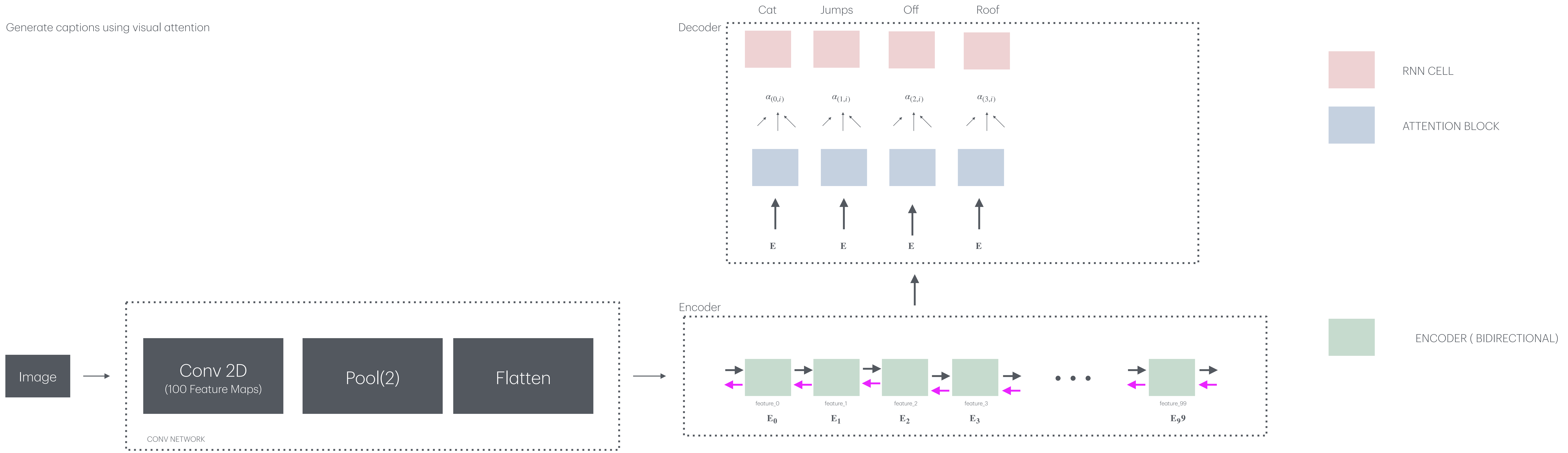
1x3x6

- 5 length vector string per time step
- 3 time steps
- Largest width = 5 (underscore is <pad>)
- Bidirectional
  - Normal Data
  - Reverse Data



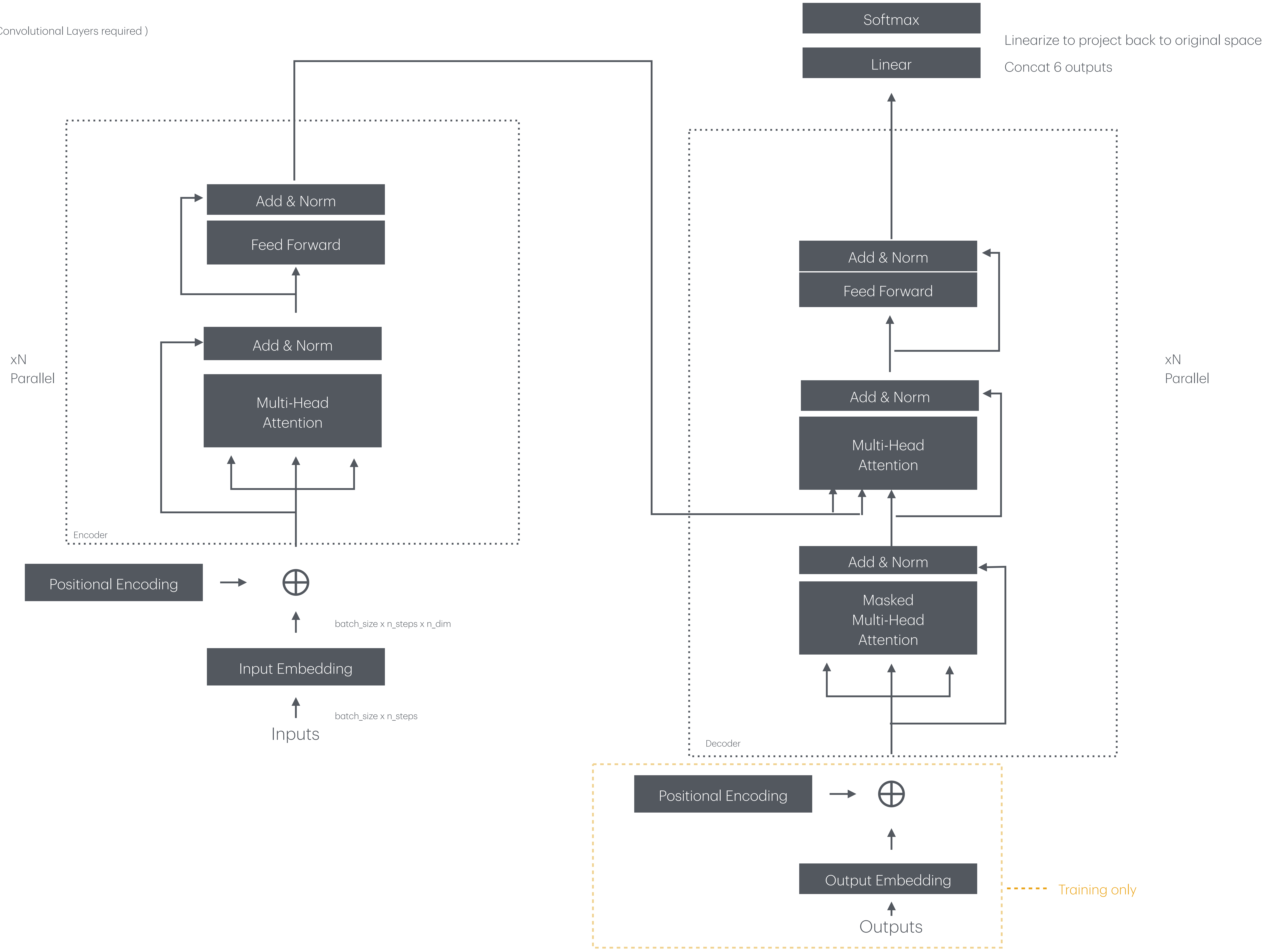
Visual Attention

Generate captions using visual attention

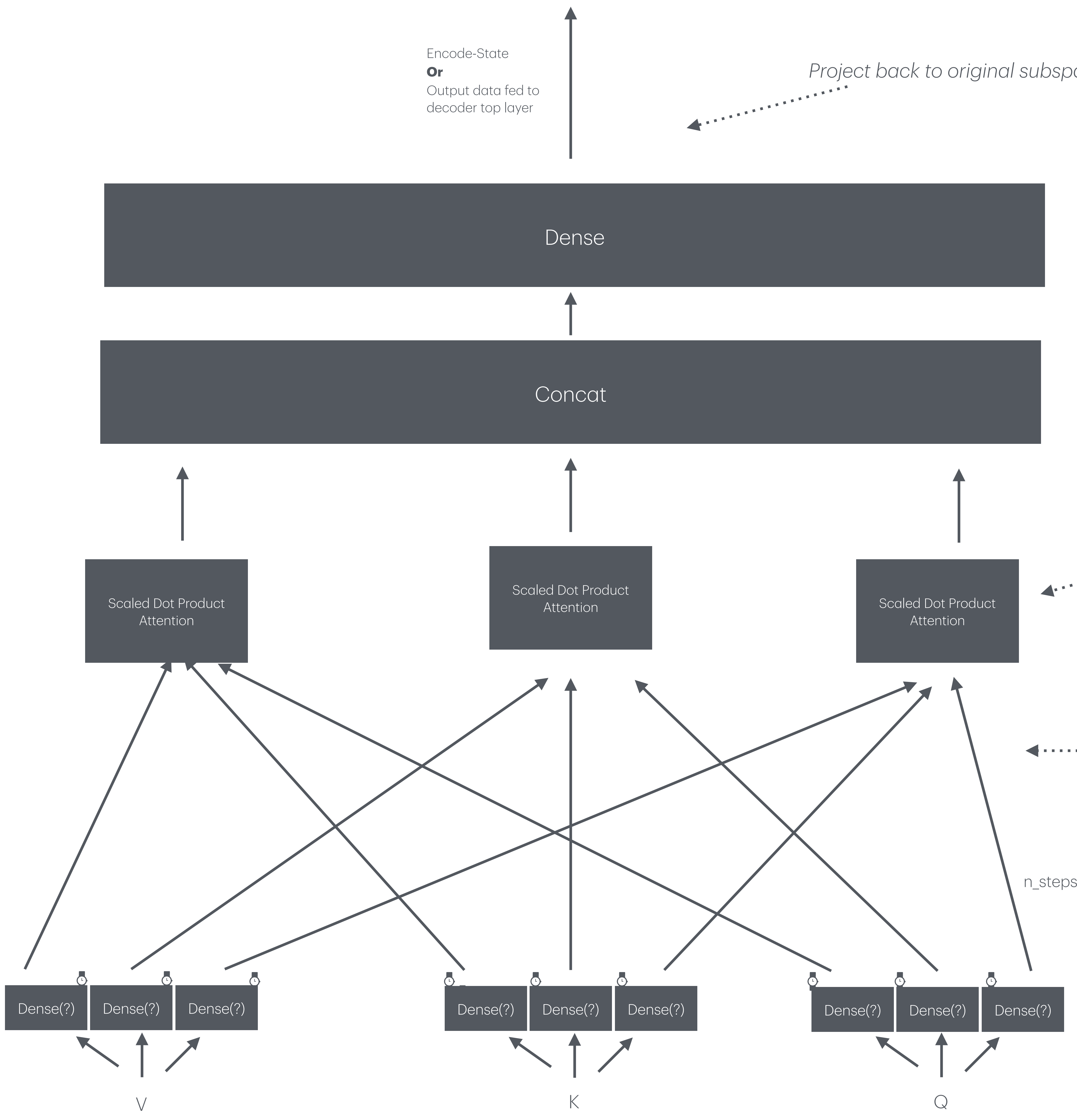


Transformer

Faster and easier to train (not RNN or Convolutional Layers required)



Multi-Head Attention



Project back to original subspace

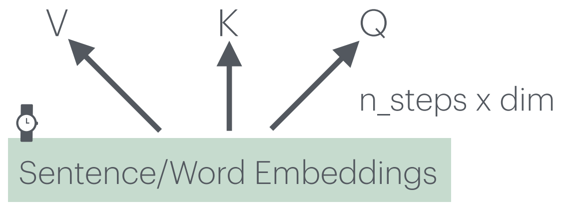
Look up phase

- *Trained lookup table*
- *During decode, a query looks for 'key' which focuses to an encoded subspace. This encoded subspace has information to support the word translation of next output*
- *During encode, model trains to encode characteristics about each input word*

Encoded data, or word representations (I.e. word characteristics) learned

Linear transformation allows model to apply different learned representations which are projections into different subspaces highlighting unique characteristics

- *Verb subspace (i.e. Played)*
- *Past Tense subspace (i.e. They)*



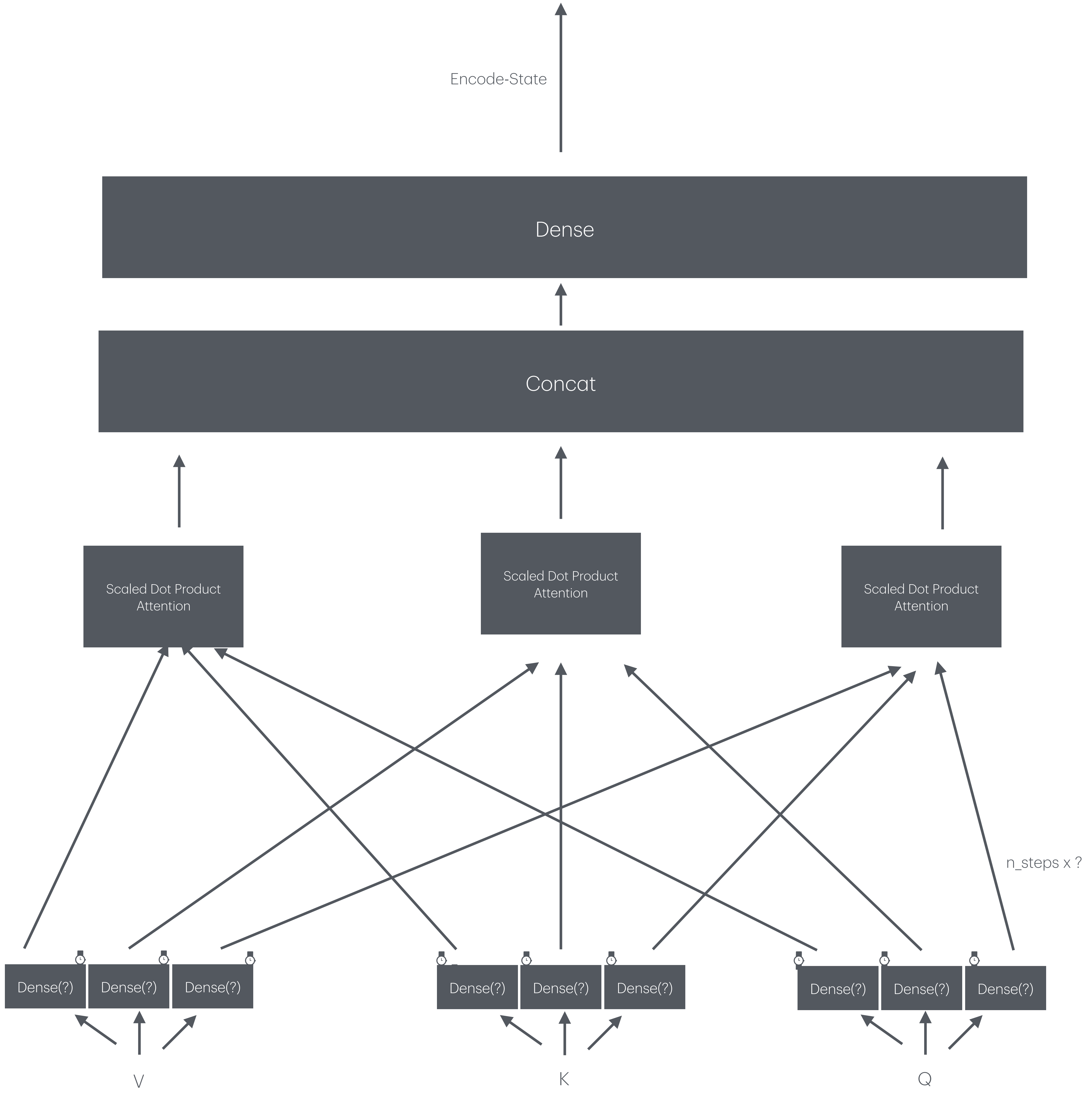
Word Representations

Q ,K, V are trading sentences

- Location MHA used in Transformer:
  - Input embeddings fed into encoder
  - Encoder_state and masked_MHA outputs fed into MHA in decoder

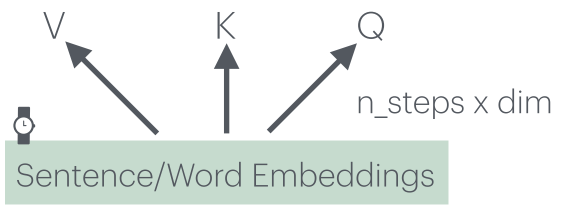
Masked Multi-Head Attention

TIME = T



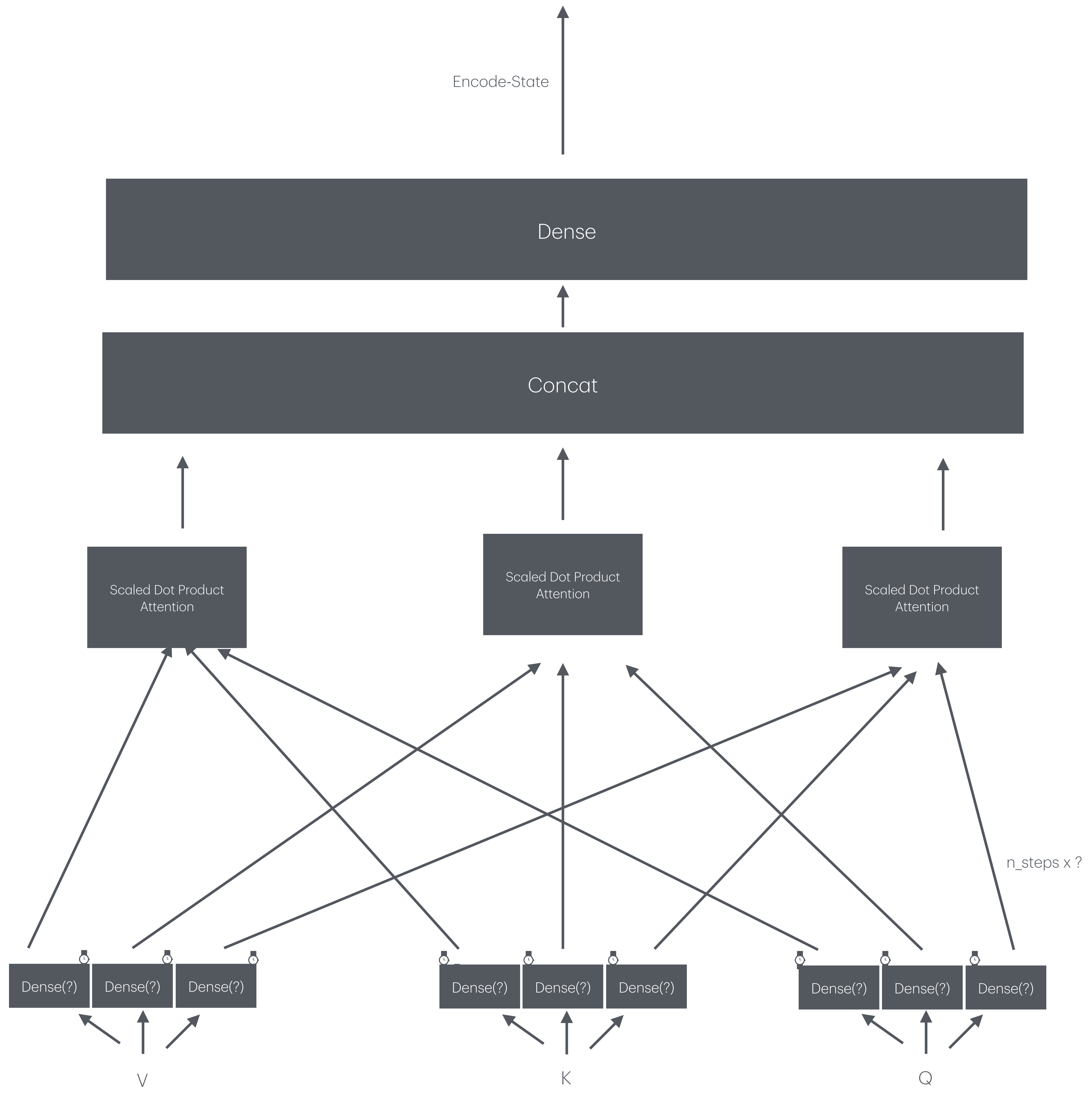
During Training Q, K, V are words in target sentences. The model will feed the decoder's masked multi-head in a casual fashion. At first time step, the previous output would the the <SOS> word

Some Word <SOS>			
Some Word <SOS>			
Some Word <SOS>			
Some Word <SOS>			
Some Word <SOS>			



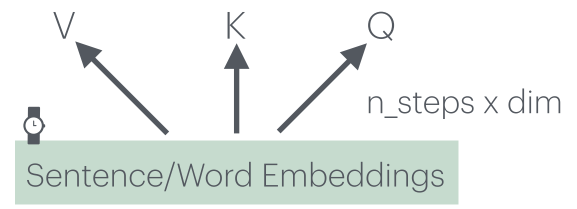
Masked Multi-Head Attention

TIME = T + 1



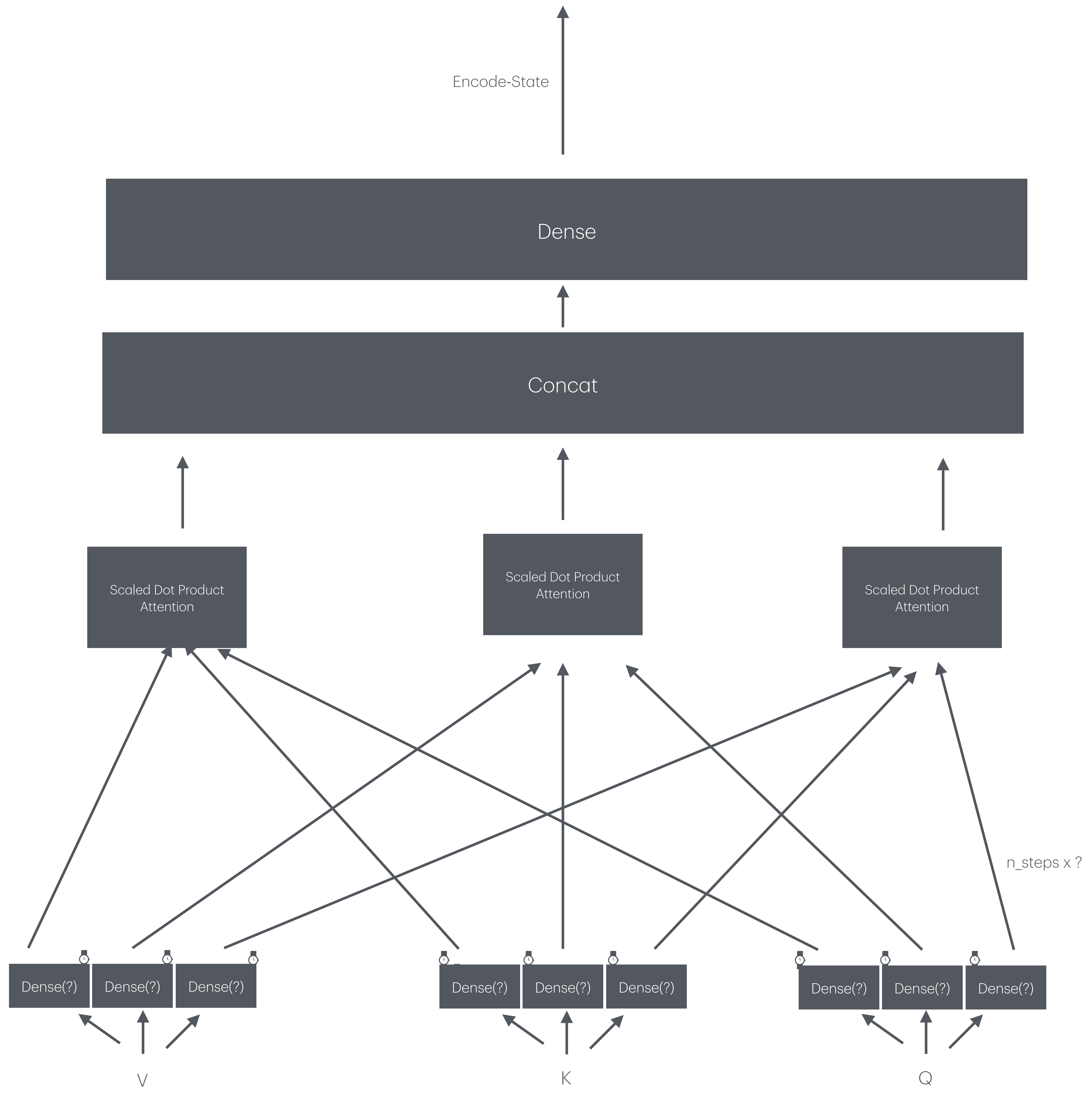
During Training Q, K, V are words in target sentences. The model will feed the decoder's masked multi-head in a casual fashion. At second time step, the previous output(s) would the the <SOS> and They

Some Word <SOS>	They		
Some Word <SOS>	Some Word		
Some Word <SOS>	Some Word		
Some Word <SOS>	Some Word		
Some Word <SOS>	Some Word		

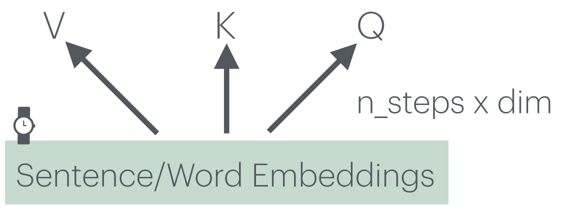


Masked Multi-Head Attention

TIME = T + 2



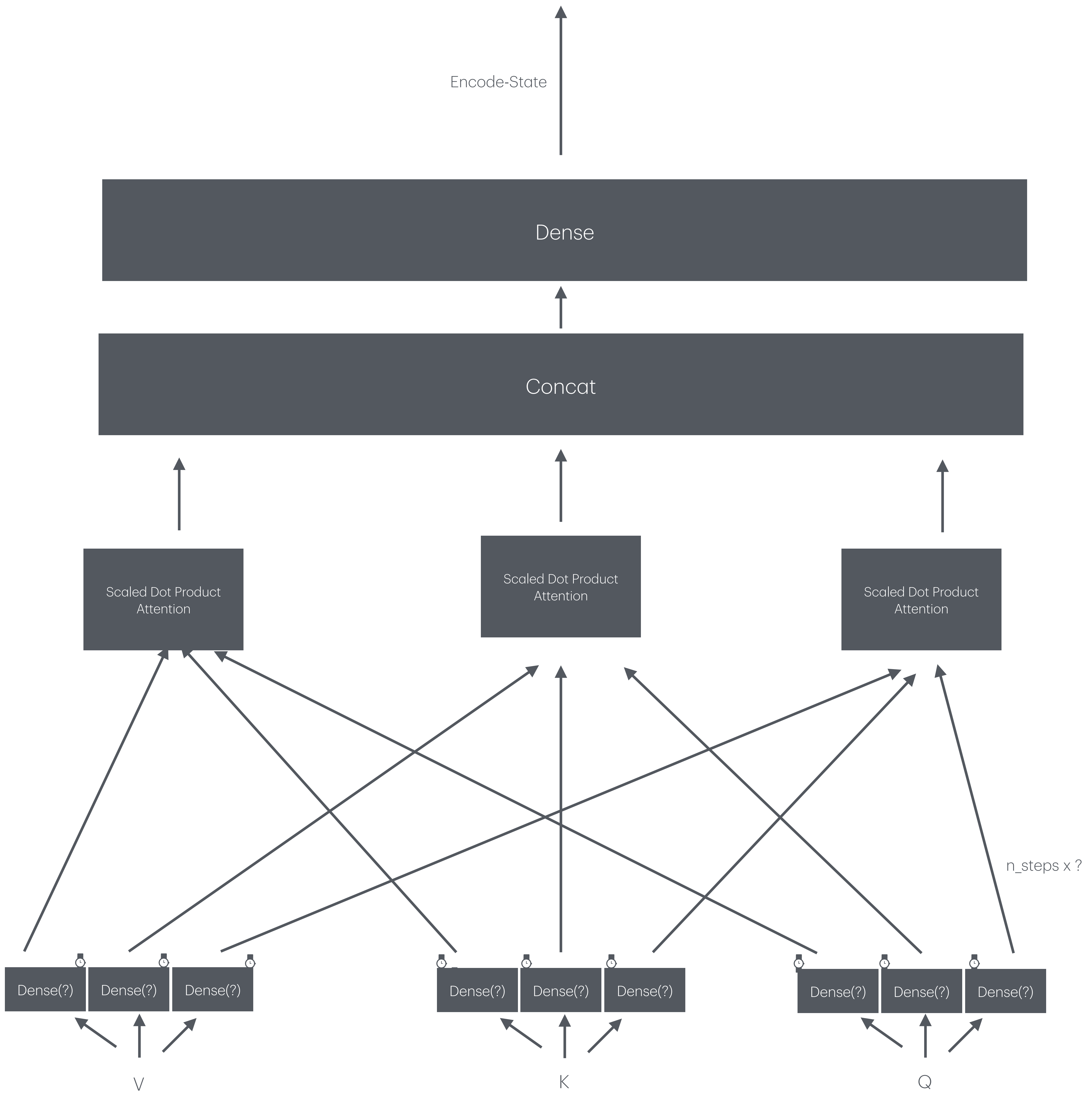
Some Word <SOS>	They	Play	
Some Word <SOS>	Some Word	Some Word	
Some Word <SOS>	Some Word	Some Word	
Some Word <SOS>	Some Word	Some Word	
Some Word <SOS>	Some Word	Some Word	



During Training Q, K, V are words in target sentences. The model will feed the decoder's masked multi-head in a casual fashion. At third time step, the previous output(s) would be the the <SOS> and They and PLAY

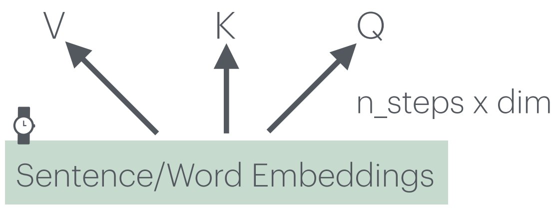
Masked Multi-Head Attention

TIME = T + 3

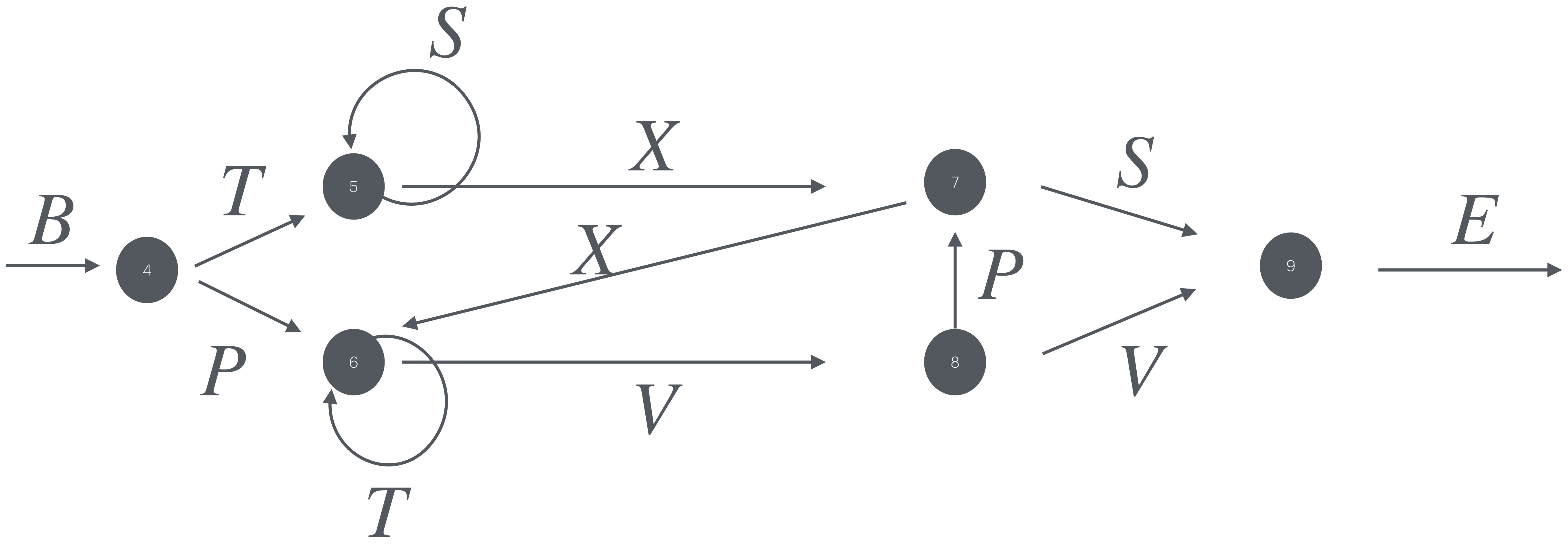


During Training Q, K, V are words in target sentences. The model will feed the decoder's masked multi-head in a casual fashion. At fourth time step, the previous output(s) would be the the <SOS> and They and PLAY and ChESS

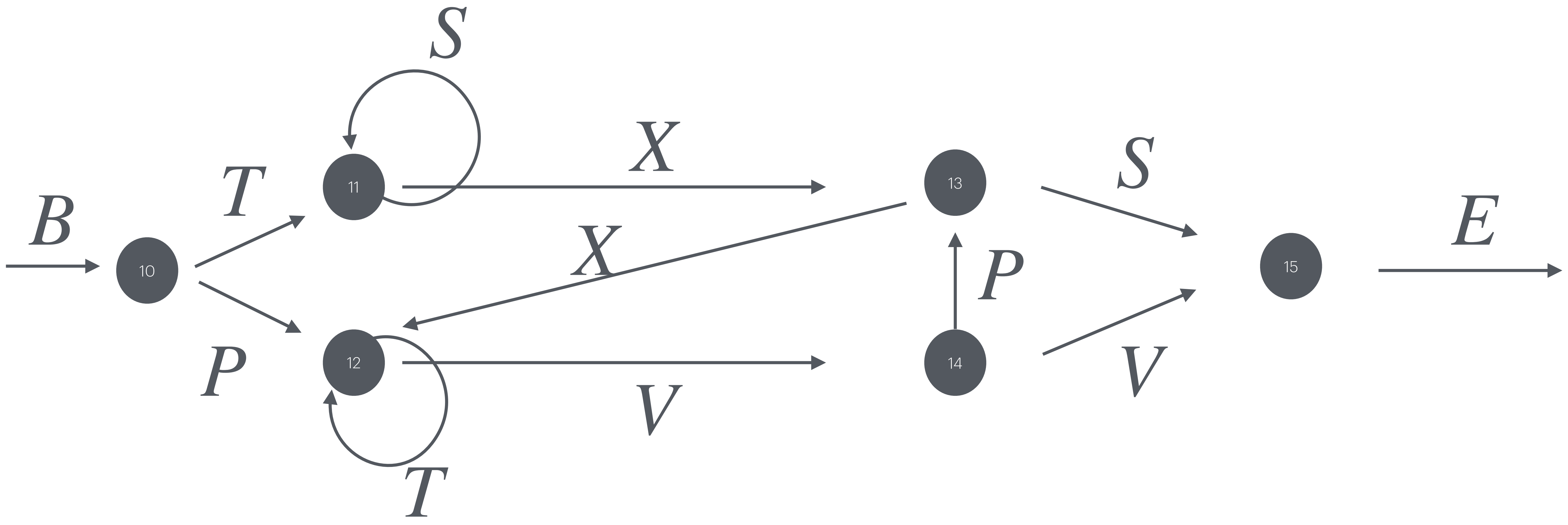
Some Word <SOS>	They	Play	Chess
Some Word <SOS>	Some Word	Some Word	Some Word
Some Word <SOS>	Some Word	Some Word	Some Word
Some Word <SOS>	Some Word	Some Word	Some Word
Some Word <SOS>	Some Word	Some Word	Some Word



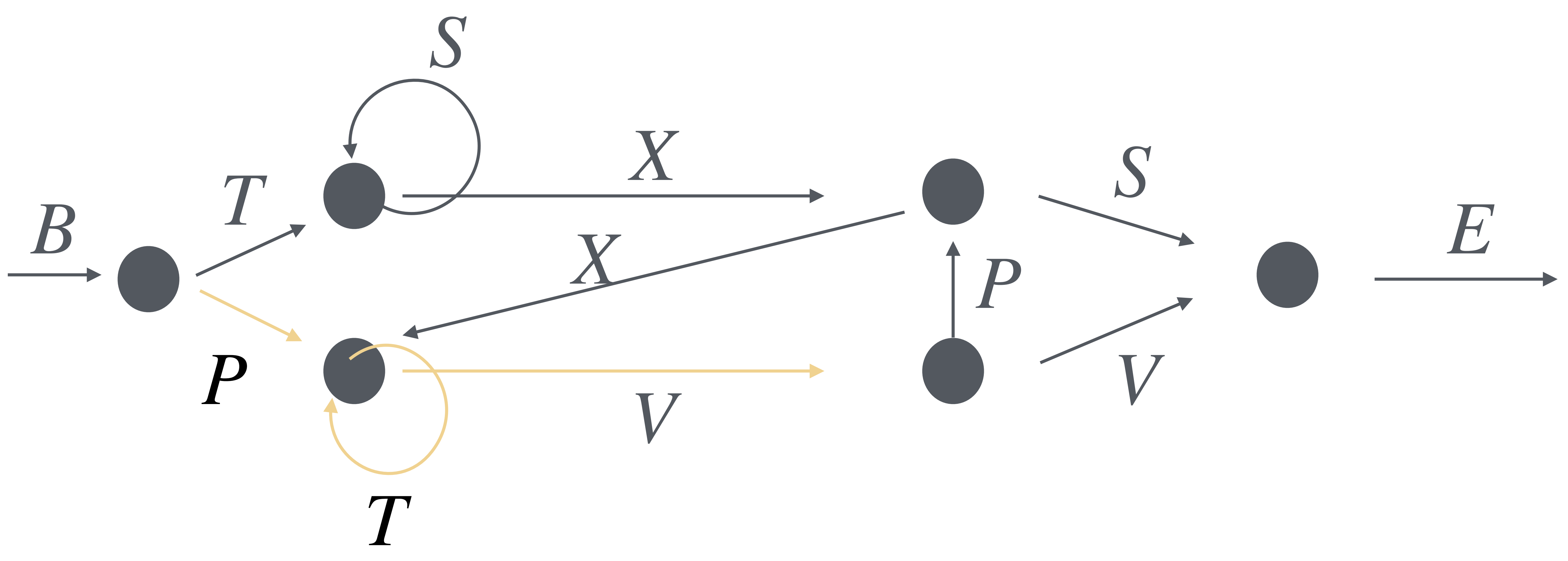
Reber1



Reber2



Embedded Reber



T is followed by T or V if previous state of **T** is T or P

P -**T** - T

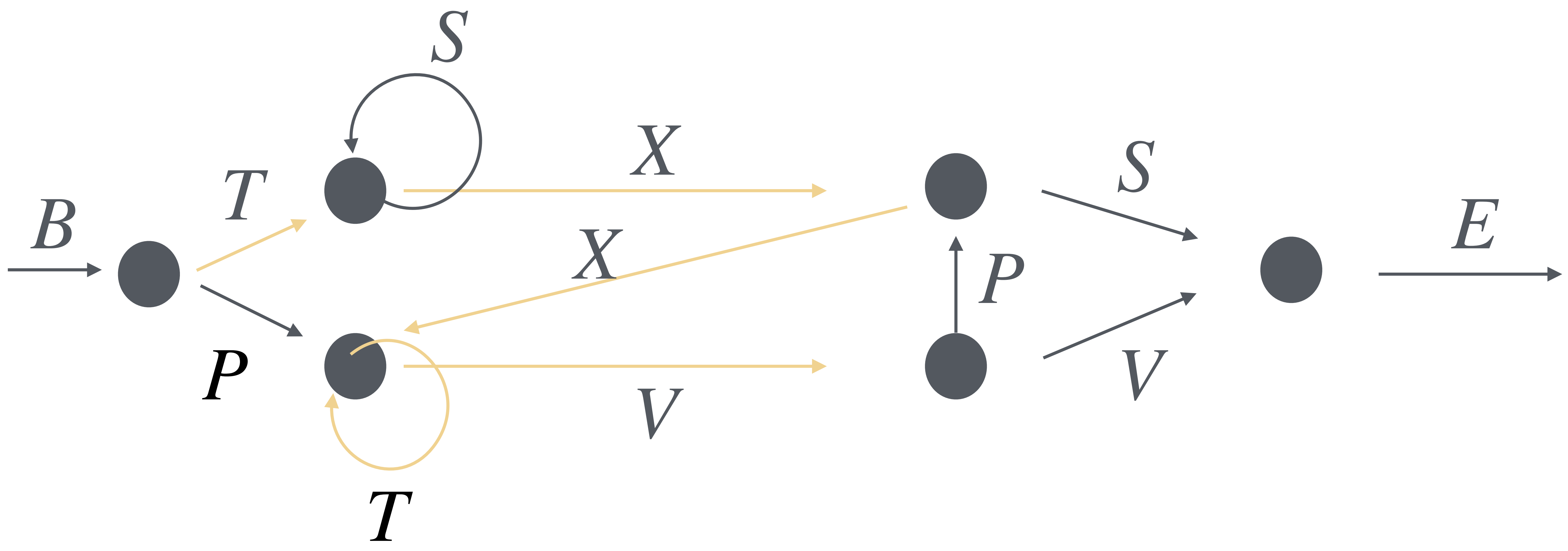
T -**T** - T

P - **T** - V

T - **T** - V

Lower Path

Invalid Path

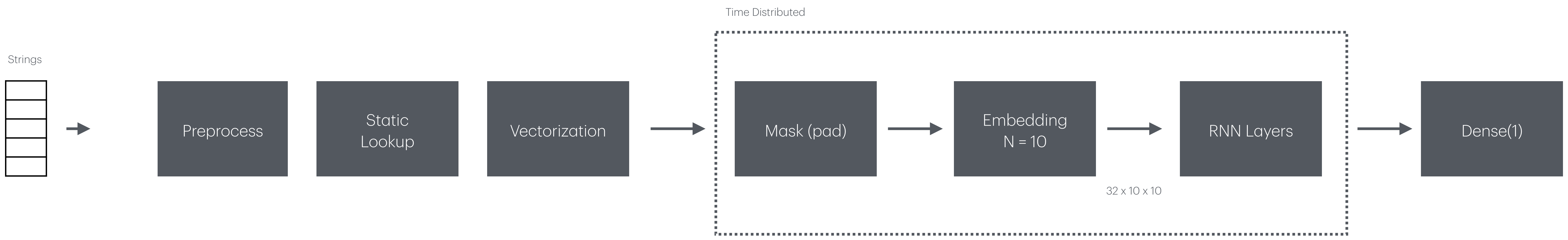


X-T-T (Not Valid)
X-T-V (Not Valid)

Not possible because upper path requires X to be preceded by states.

T-X-X-T-V (VALID)
T-X-X-T-T (VALID)

Rebel Classification Model



[
'BT',
'BBDACBT',
'BTSSXXTVVE',
'BTBTSSXXVVETE',
'BTSSPXSE',
'BTBPTVVETE',
]

Valid Code

Invalid Code

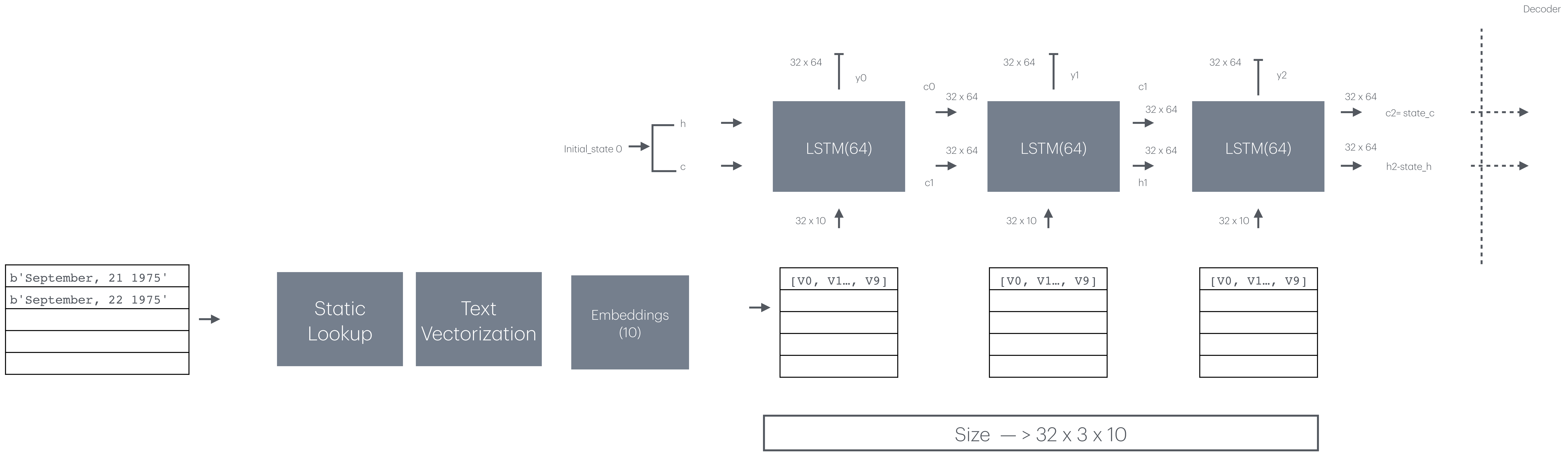
Probabilities

```
[

 [2.1597907e-04]
 [3.6814836e-05]
 [3.3914832e-05]
 [7.4838662e-01]
 [3.4200060e-05]
 [3.1849074e-01]

]
```

Encoder Decoder



Encoder **Decoder**