



Deep Learning for Vision & Language

Natural Language Processing: Sequence Modeling with RNNs

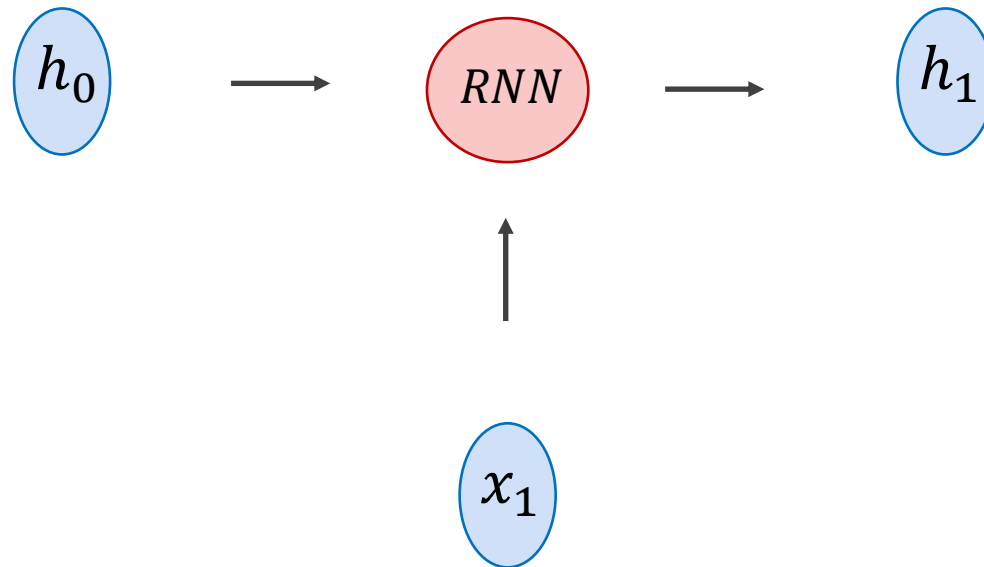


RICE UNIVERSITY

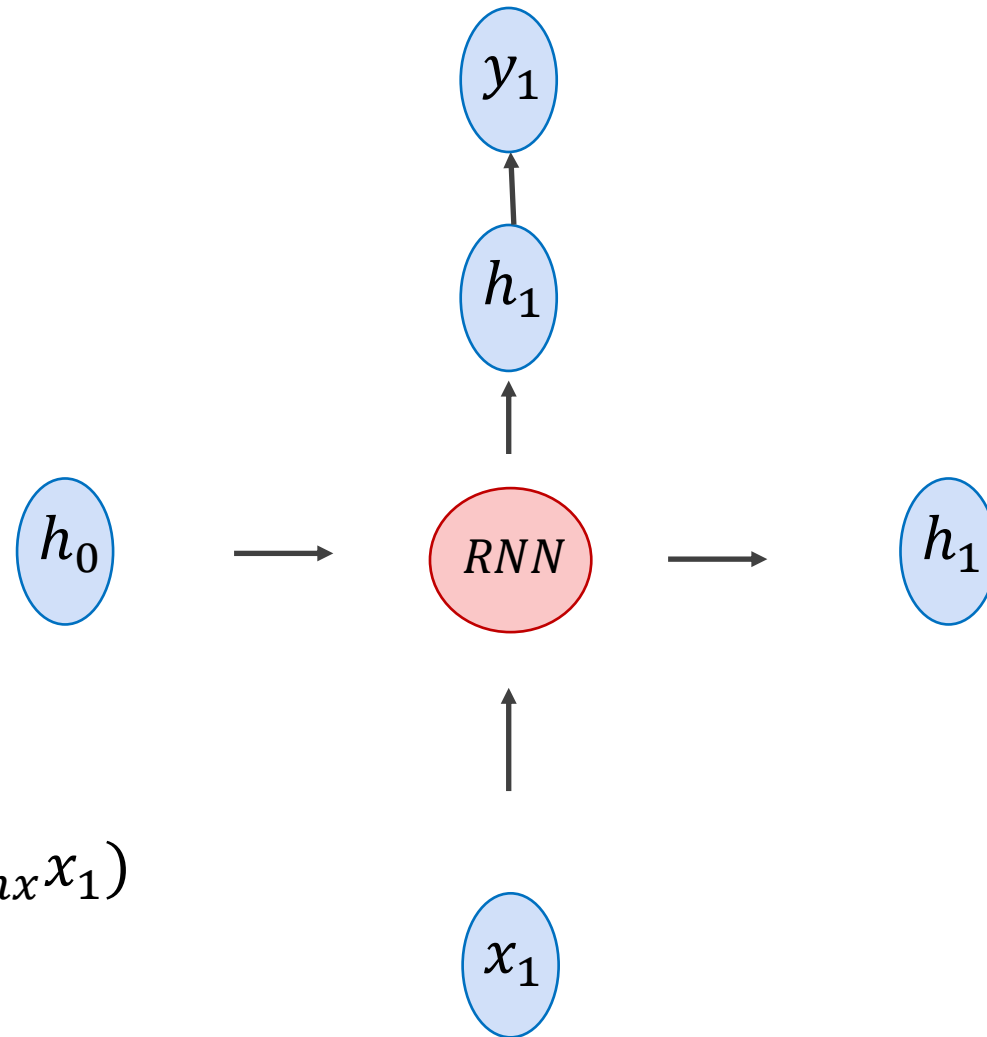


Recurrent Neural Network Cell

$$h_1 = \tanh(W_{hh}h_0 + W_{hx}x_1)$$



Recurrent Neural Network Cell



$$h_1 = \tanh(W_{hh}h_0 + W_{hx}x_1)$$

$$y_1 = \text{softmax}(W_{hy}h_1)$$

Recurrent Neural Network Cell

$$y_1 = [0.1, 0.05, 0.05, 0.1, 0.7]$$



$$h_1 = [0.1 \ 0.2 \ 0 \ -0.3 \ -0.1]$$



$$h_0 = [0 \ 0 \ 0 \ 0 \ 0 \ 0 \ 0]$$



$$h_1 = [0.1 \ 0.2 \ 0 \ -0.3 \ -0.1]$$

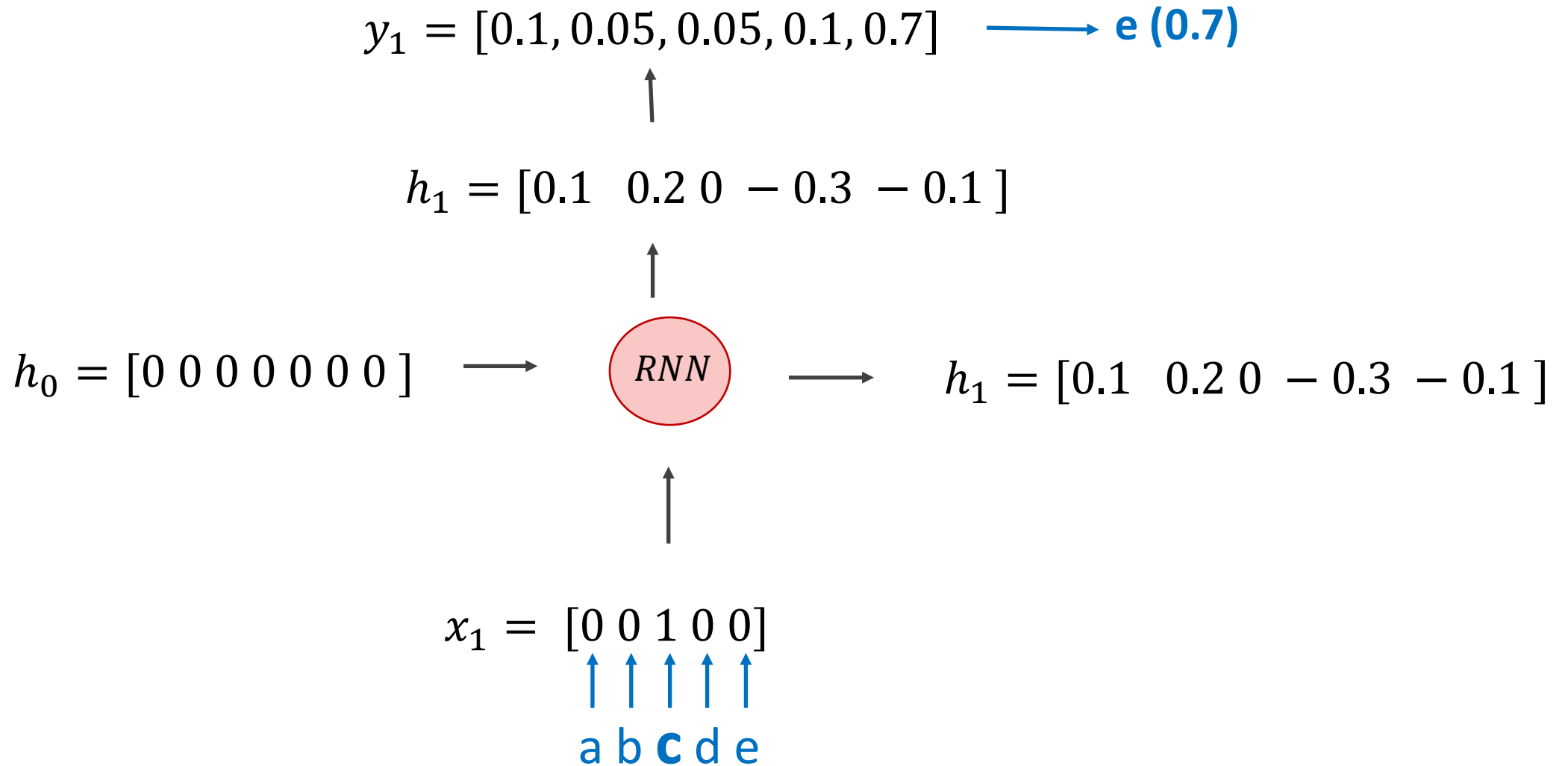


$$h_1 = \tanh(W_{hh}h_0 + W_{hx}x_1)$$

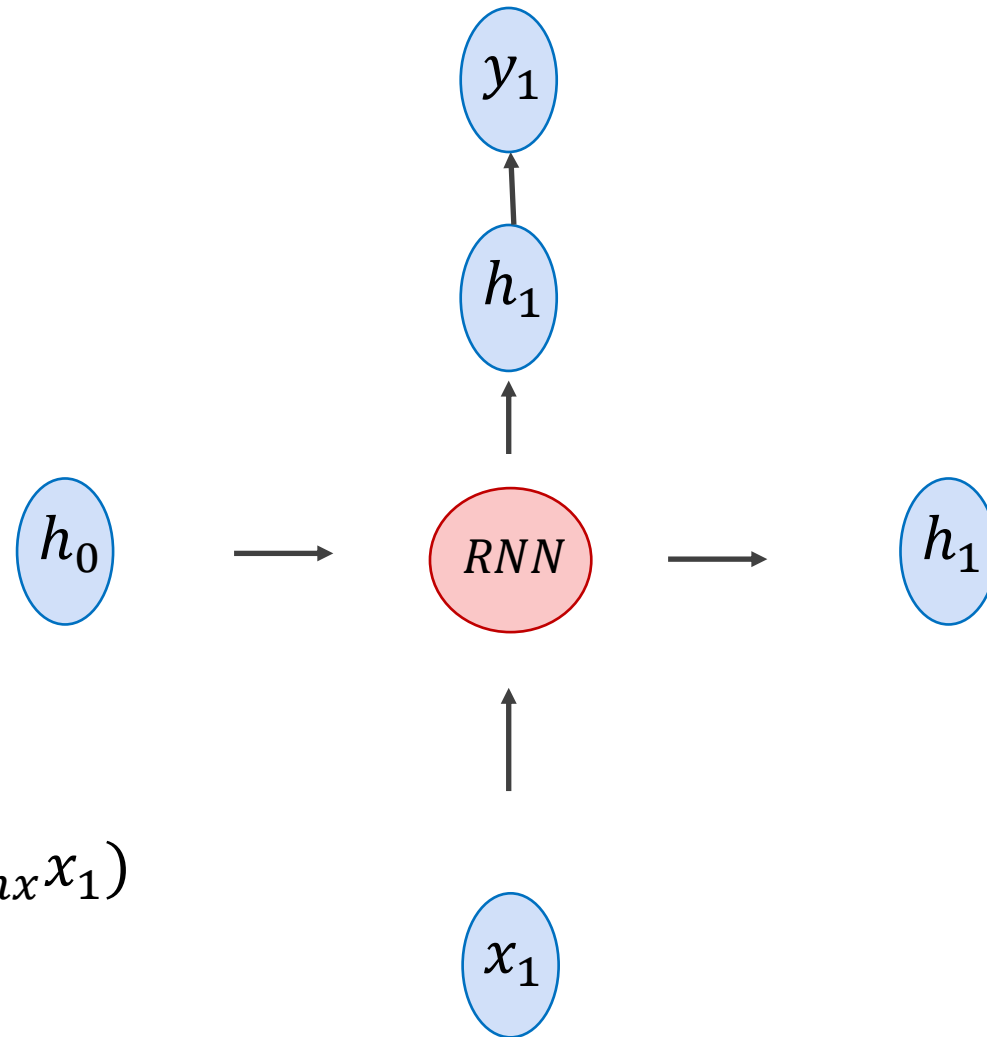
$$x_1 = [0 \ 0 \ 1 \ 0 \ 0]$$

$$y_1 = \text{softmax}(W_{hy}h_1)$$

Recurrent Neural Network Cell



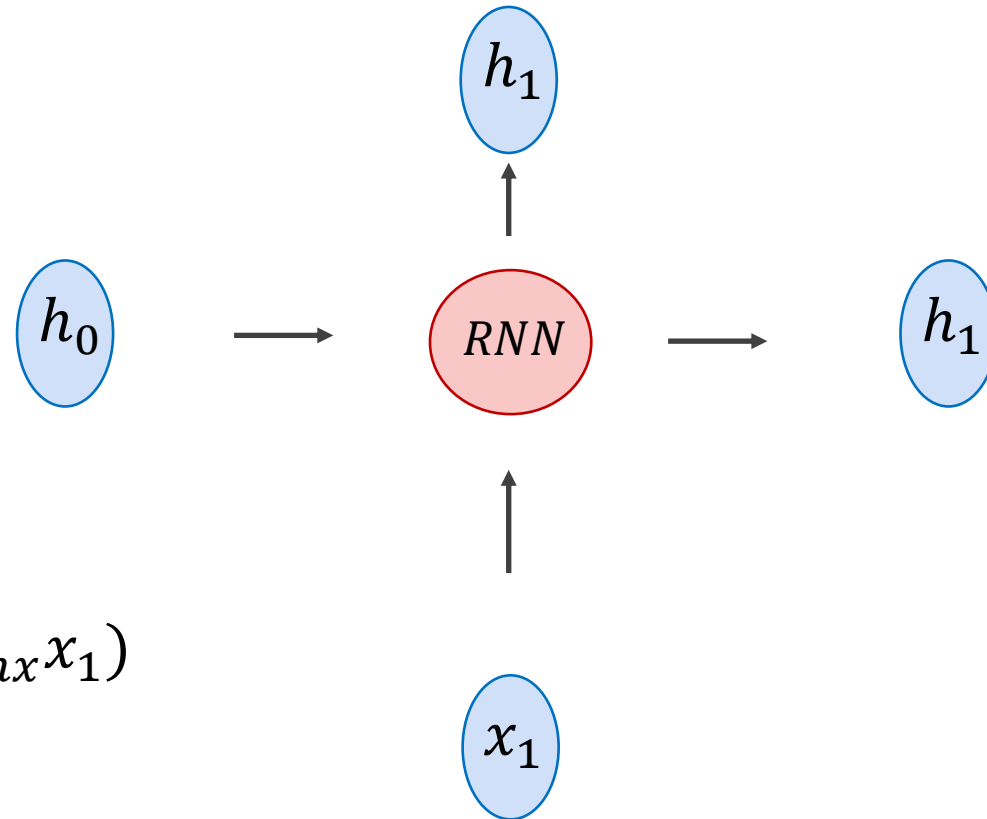
Recurrent Neural Network Cell



$$h_1 = \tanh(W_{hh}h_0 + W_{hx}x_1)$$

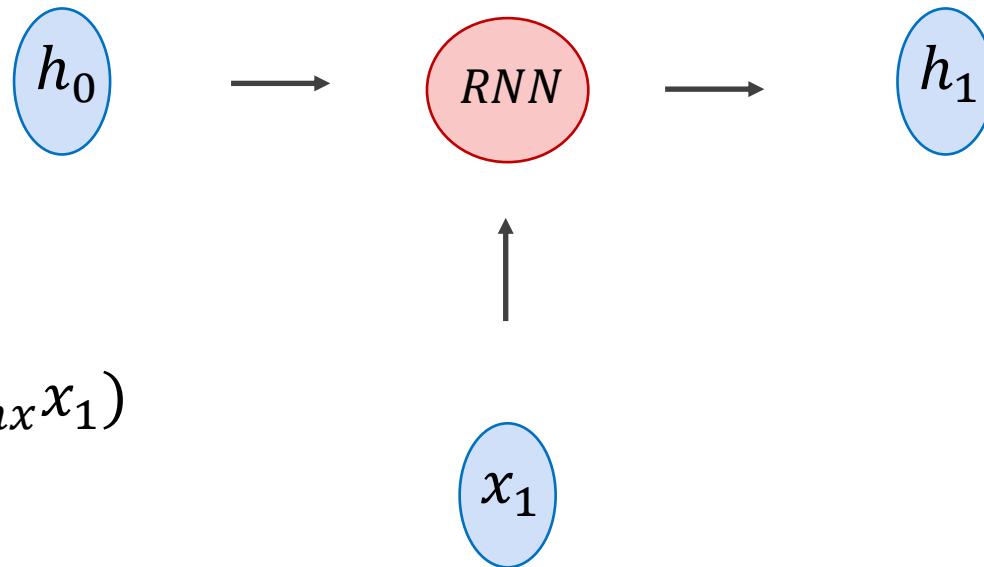
$$y_1 = \text{softmax}(W_{hy}h_1)$$

Recurrent Neural Network Cell



$$h_1 = \tanh(W_{hh}h_0 + W_{hx}x_1)$$

Recurrent Neural Network Cell



$$h_1 = \tanh(W_{hh}h_0 + W_{hx}x_1)$$

RNN

```
CLASS torch.nn.RNN(self, input_size, hidden_size, num_layers=1, nonlinearity='tanh',  
bias=True, batch_first=False, dropout=0.0, bidirectional=False, device=None,  
dtype=None) [SOURCE]
```



Apply a multi-layer Elman RNN with `tanh` or `ReLU` non-linearity to an input sequence. For each element in the input sequence, each layer computes the following function:

$$h_t = \tanh(x_t W_{ih}^T + b_{ih} + h_{t-1} W_{hh}^T + b_{hh})$$

where h_t is the hidden state at time t , x_t is the input at time t , and $h_{(t-1)}$ is the hidden state of the previous layer at time $t-1$ or the initial hidden state at time 0. If `nonlinearity` is `'relu'`, then `ReLU` is used instead of `tanh`.

Parameters

- **input_size** – The number of expected features in the input x
- **hidden_size** – The number of features in the hidden state h
- **num_layers** – Number of recurrent layers. E.g., setting `num_layers=2` would mean stacking two RNNs together to form a *stacked RNN*, with the second RNN taking in outputs of the first RNN and computing the final results. Default: 1
- **nonlinearity** – The non-linearity to use. Can be either `'tanh'` or `'relu'`. Default: `'tanh'`
- **bias** – If `False`, then the layer does not use bias weights b_{ih} and b_{hh} . Default: `True`
- **batch_first** – If `True`, then the input and output tensors are provided as $(batch, seq, feature)$ instead of $(seq, batch, feature)$. Note that this does not apply to hidden or cell states. See the Inputs/Outputs sections below for details. Default: `False`
- **dropout** – If non-zero, introduces a *Dropout* layer on the outputs of each RNN layer except the last layer, with dropout probability equal to `dropout`. Default: 0
- **bidirectional** – If `True`, becomes a bidirectional RNN. Default: `False`

Inputs: input, h_0

- **input:** tensor of shape (L, H_{in}) for unbatched input, (L, N, H_{in}) when `batch_first=False` or (N, L, H_{in}) when `batch_first=True` containing the features of the input sequence. The input can also be a packed variable length sequence. See [torch.nn.utils.rnn.pack_padded_sequence\(\)](#) or [torch.nn.utils.rnn.pack_sequence\(\)](#) for details.
- **h_0:** tensor of shape $(D * \text{num_layers}, H_{out})$ for unbatched input or $(D * \text{num_layers}, N, H_{out})$ containing the initial hidden state for the input sequence batch. Defaults to zeros if not provided.

where:

$N = \text{batch size}$

$L = \text{sequence length}$

$D = 2$ if `bidirectional=True` otherwise 1

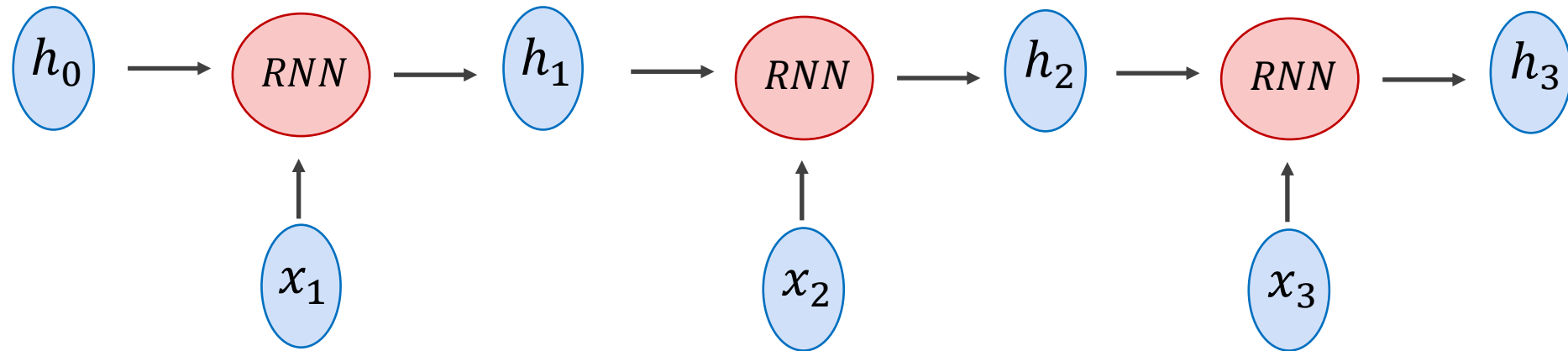
$H_{in} = \text{input_size}$

$H_{out} = \text{hidden_size}$

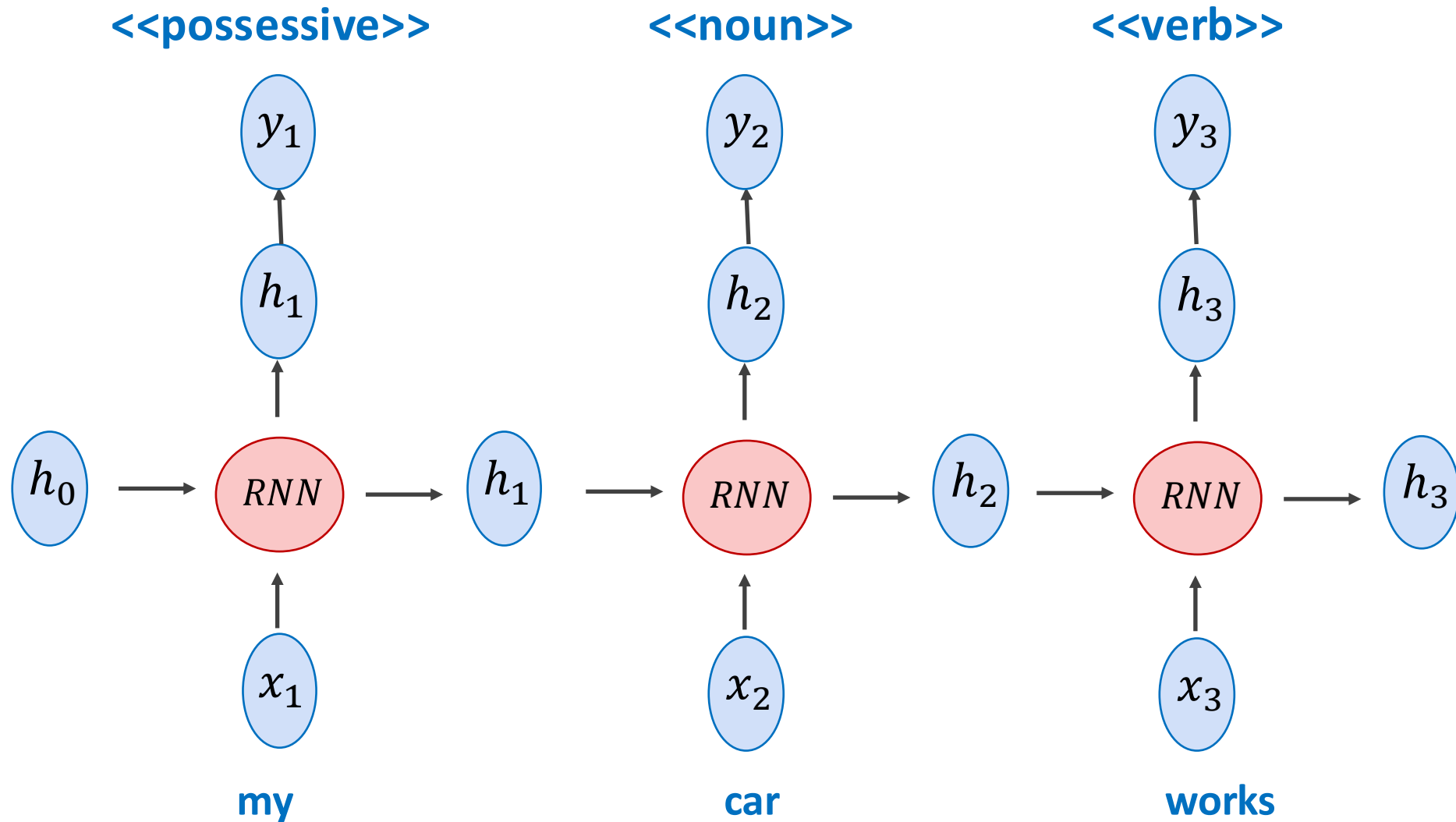
Outputs: output, h_n

- **output:** tensor of shape $(L, D * H_{out})$ for unbatched input, $(L, N, D * H_{out})$ when `batch_first=False` or $(N, L, D * H_{out})$ when `batch_first=True` containing the output features (h_t) from the last layer of the RNN, for each t . If a [torch.nn.utils.rnn.PackedSequence](#) has been given as the input, the output will also be a packed sequence.
- **h_n:** tensor of shape $(D * \text{num_layers}, H_{out})$ for unbatched input or $(D * \text{num_layers}, N, H_{out})$ containing the final hidden state for each element in the batch.

(Unrolled) Recurrent Neural Network



How can it be used? – e.g. Tagging a Text Sequence
One-to-one Sequence Mapping Problems



How can it be used? – e.g. Tagging a Text Sequence
One-to-one Sequence Mapping Problems

Training examples don't need to be the same length!

input

output

my car works

<<possessive>> <<noun>> <<verb>>

my dog ate the assignment

<<possessive>> <<noun>> <<verb>> <<pronoun>> <<noun>>

my mother saved the day

<<possessive>> <<noun>> <<verb>> <<pronoun>> <<noun>>

the smart kid solved the problem

<<pronoun>> <<qualifier>> <<noun>> <<verb>> <<pronoun>> <<noun>>

How can it be used? – e.g. Tagging a Text Sequence
One-to-one Sequence Mapping Problems

Training examples don't need to be the same length!

input

L(my car works) = 3

L(my dog ate the assignment) = 5

L(my mother saved the day) = 5

L(the smart kid solved the problem) = 6

output

L(<<possessive>> <<noun>> <<verb>>) = 3

L(<<possessive>> <<noun>> <<verb>> <<pronoun>> <<noun>>) = 5

L(<<possessive>> <<noun>> <<verb>> <<pronoun>> <<noun>>) = 5

L(<<pronoun>> <<qualifier>> <<noun>> <<verb>> <<pronoun>> <<noun>>) = 6

How can it be used? – e.g. Tagging a Text Sequence
One-to-one Sequence Mapping Problems

Training examples don't need to be the same length!

If we assume a vocabulary of a 1000 possible words and 20 possible output tags

input

T: 1000 x 3

T: 1000 x 5

T: 1000 x 5

T: 1000 x 6

output

T: 20 x 3

T: 20 x 5

T: 20 x 5

T: 20 x 6

How can it be used? – e.g. Tagging a Text Sequence
One-to-one Sequence Mapping Problems

Training examples don't need to be the same length!

If we assume a vocabulary of a 1000 possible words and 20 possible output tags

input

T: 1000 x 3

T: 1000 x 5

T: 1000 x 5

T: 1000 x 6

output

T: 20 x 3

T: 20 x 5

T: 20 x 5

T: 20 x 6

How do we create batches if inputs and outputs have different shapes?

How can it be used? – e.g. Tagging a Text Sequence
One-to-one Sequence Mapping Problems

Training examples don't need to be the same length!

If we assume a vocabulary of a 1000 possible words and 20 possible output tags

input

T: 1000 x 3

T: 1000 x 5

T: 1000 x 5

T: 1000 x 6

output

T: 20 x 3

T: 20 x 5

T: 20 x 5

T: 20 x 6

How do we create batches if inputs and outputs have different shapes?

Solution 1: Forget about batches, just process things one by one.

How can it be used? – e.g. Tagging a Text Sequence
One-to-one Sequence Mapping Problems

Training examples don't need to be the same length!

If we assume a vocabulary of a 1000 possible words and 20 possible output tags

input

T: 1000 x 3

T: 1000 x 5

T: 1000 x 5

T: 1000 x 6

output

T: 20 x 3

T: 20 x 5

T: 20 x 5

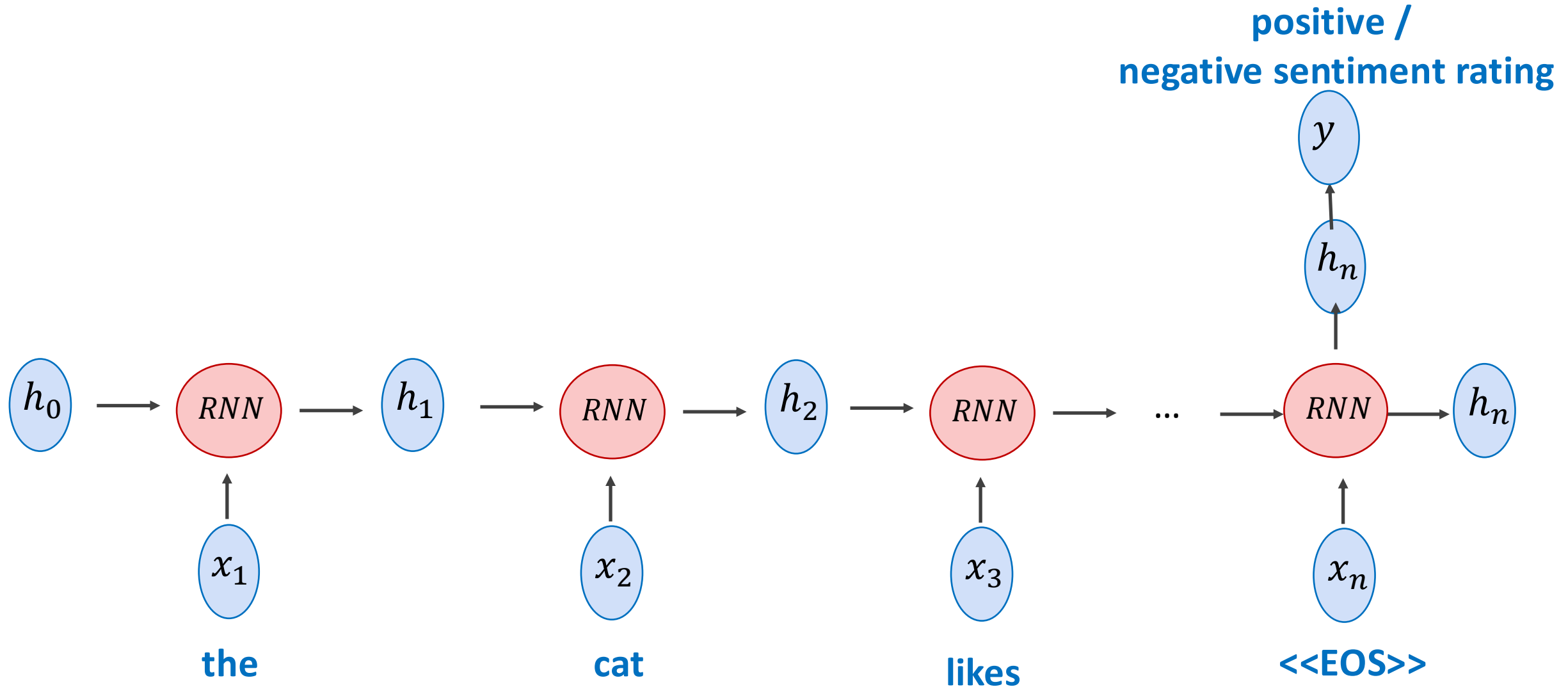
T: 20 x 6

How do we create batches if inputs and outputs have different shapes?

Solution 2: Zero padding.

We can put the above vectors in **T: 4 x 1000 x 6**

How can it be used? – e.g. Scoring the Sentiment of a Text Sequence
Many-to-one Sequence to score problems



How can it be used? – e.g. Sentiment Scoring
Many to one Mapping Problems

Input training examples don't need to be the same length!
In this case outputs can be.

input

output

this restaurant has good food

Positive

this restaurant is bad

Negative

this restaurant is the worst

Negative

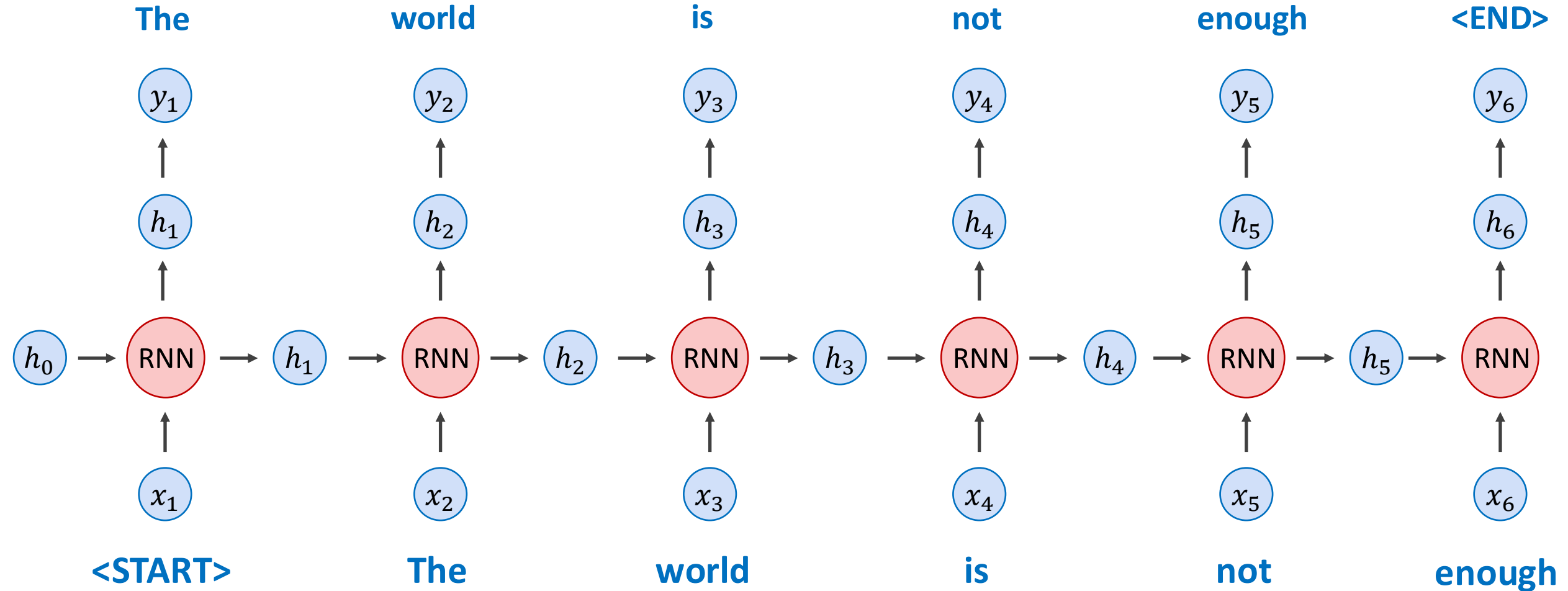
this restaurant is well recommended

Positive

How can it be used? – e.g. Text Generation

Auto-regressive model – Sequence to Sequence during Training, Auto-regressive during test

DURING TRAINING



How can it be used? – e.g. Text Generation
Auto-regressive Models

Input training examples don't need to be the same length!
In this case outputs can be.

input

output

<START> this restaurant has good food

this restaurant has good food <END>

<START> this restaurant is bad

this restaurant is bad <END>

<START> this restaurant is the worst

this restaurant is the worst <END>

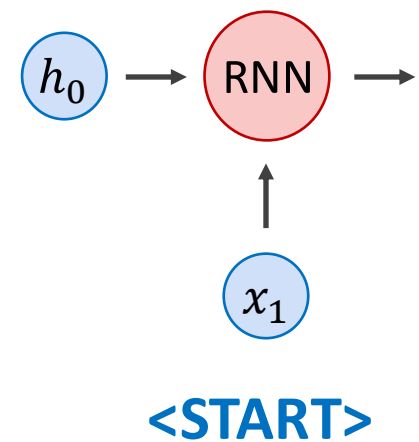
<START> this restaurant is well recommended

this restaurant is well recommended <END>

How can it be used? – e.g. Text Generation

Auto-regressive model – Sequence to Sequence during Training, Auto-regressive during test

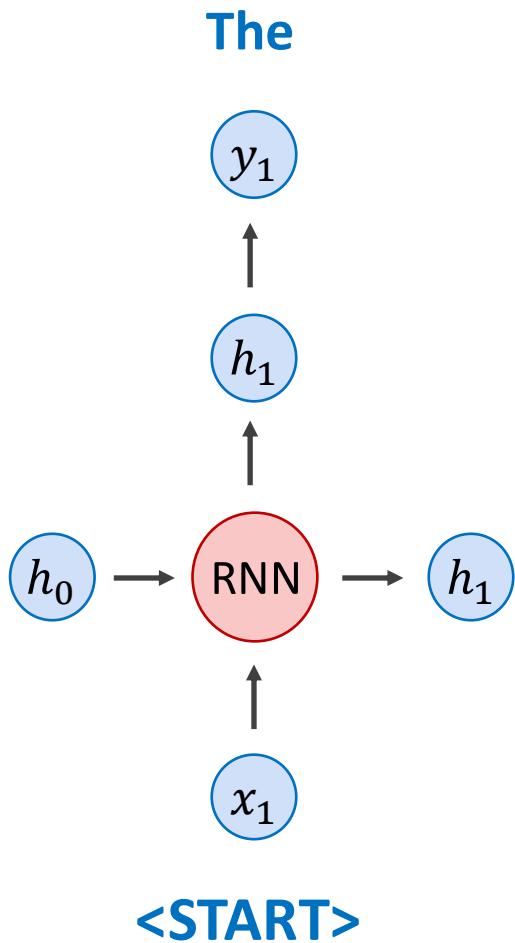
DURING TESTING



How can it be used? – e.g. Text Generation

Auto-regressive model – Sequence to Sequence during Training, Auto-regressive during test

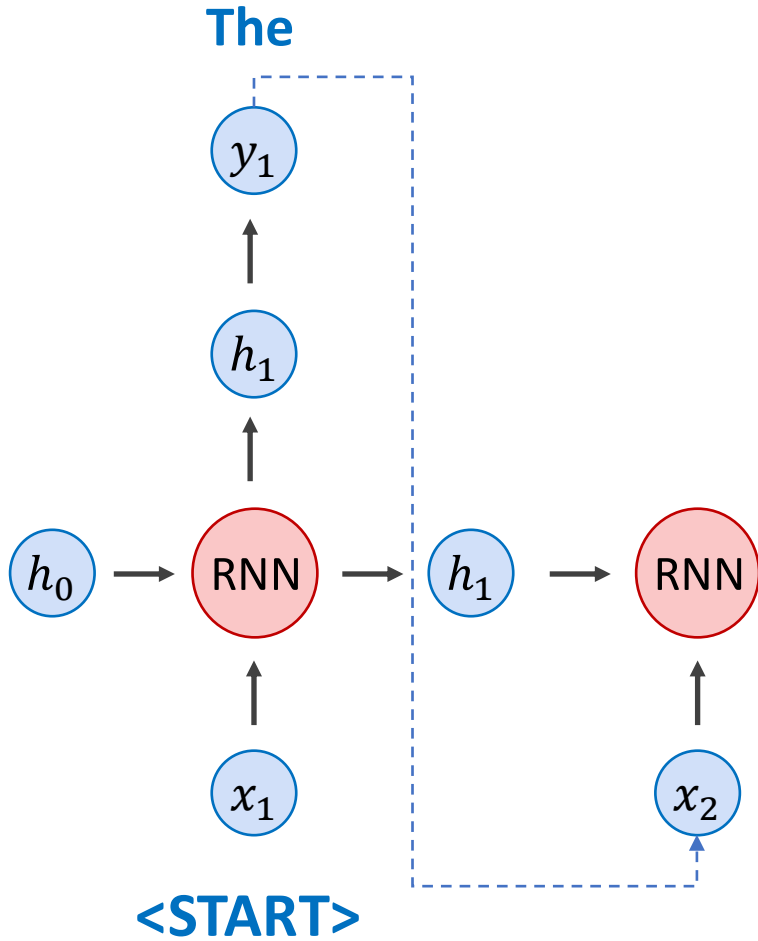
DURING TESTING



How can it be used? – e.g. Text Generation

Auto-regressive model – Sequence to Sequence during Training, Auto-regressive during test

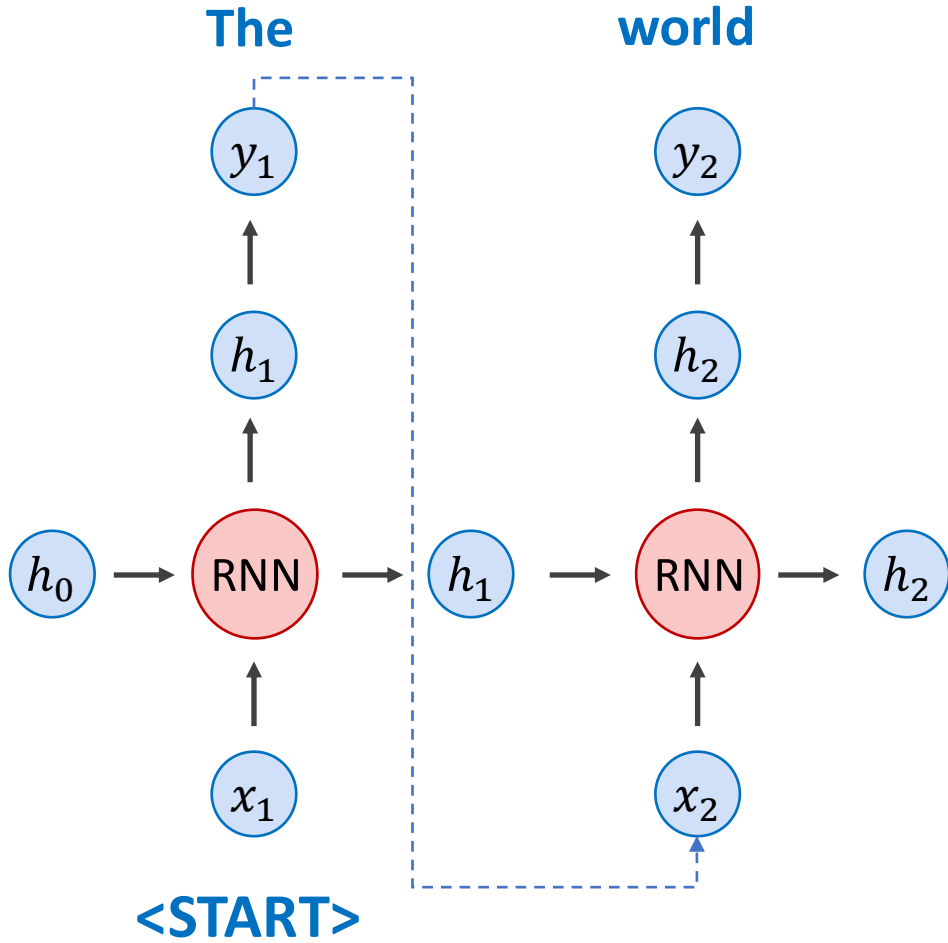
DURING TESTING



How can it be used? – e.g. Text Generation

Auto-regressive model – Sequence to Sequence during Training, Auto-regressive during test

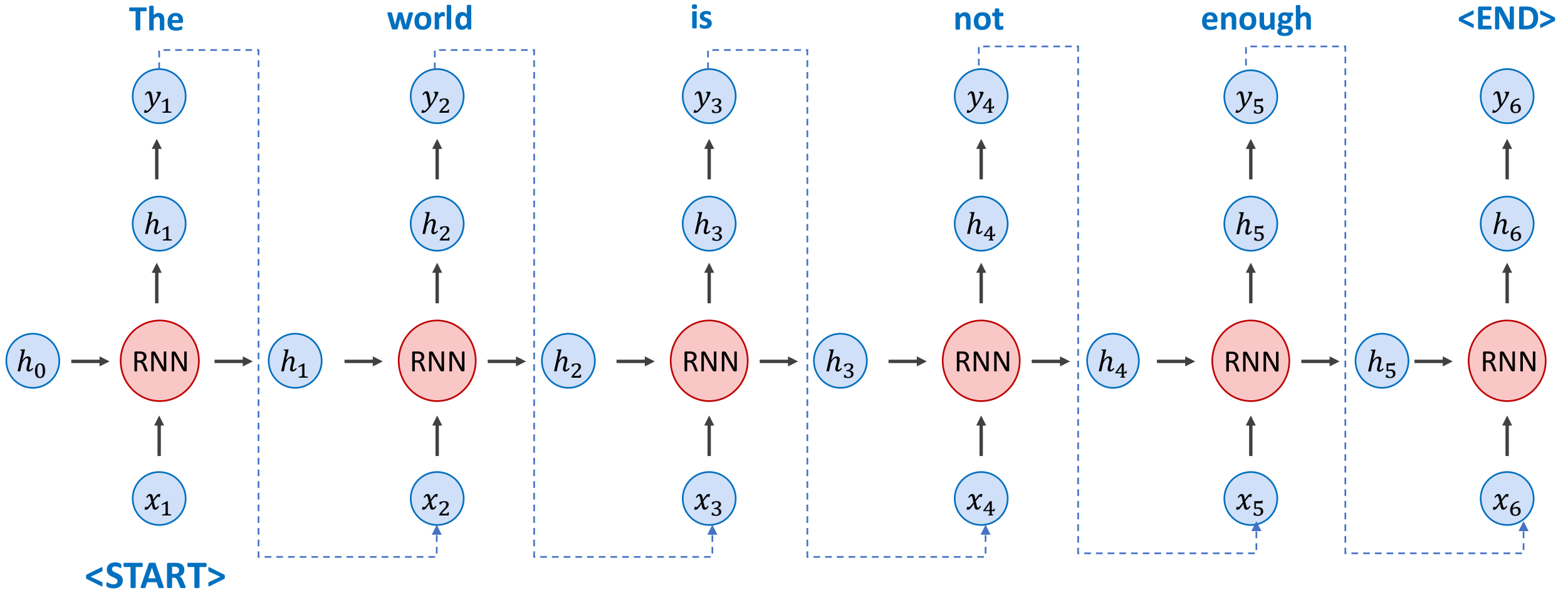
DURING TESTING



How can it be used? – e.g. Text Generation

Auto-regressive model – Sequence to Sequence during Training, Auto-regressive during test

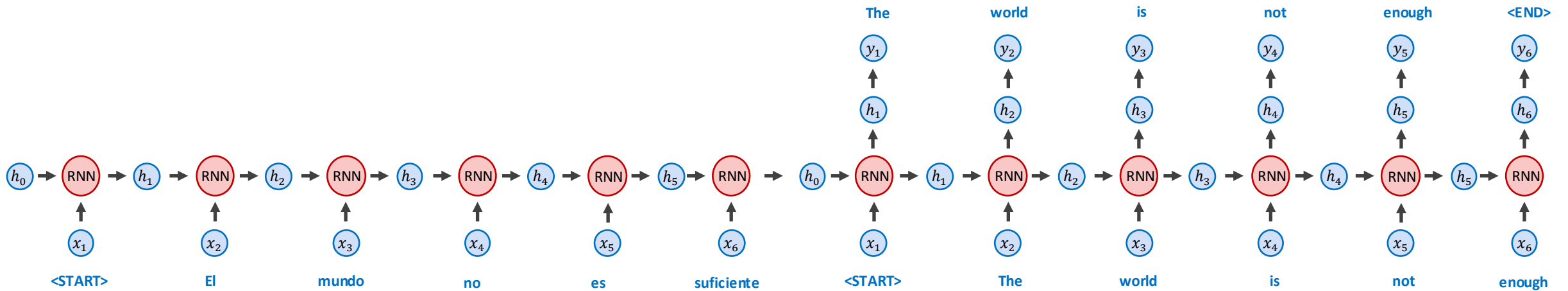
DURING TESTING



How can it be used? – e.g. Machine Translation

Sequence to Sequence – Encoding – Decoding – Many to Many mapping

DURING TRAINING



How can it be used? – e.g. Machine Translation
Sequence to Sequence Models

Input training examples don't need to be the same length!
In this case outputs can be.

input

output

<START> este restaurante tiene buena comida

this restaurant has good food <END>

<START> this restaurant has good food

<START> el mundo no es suficiente

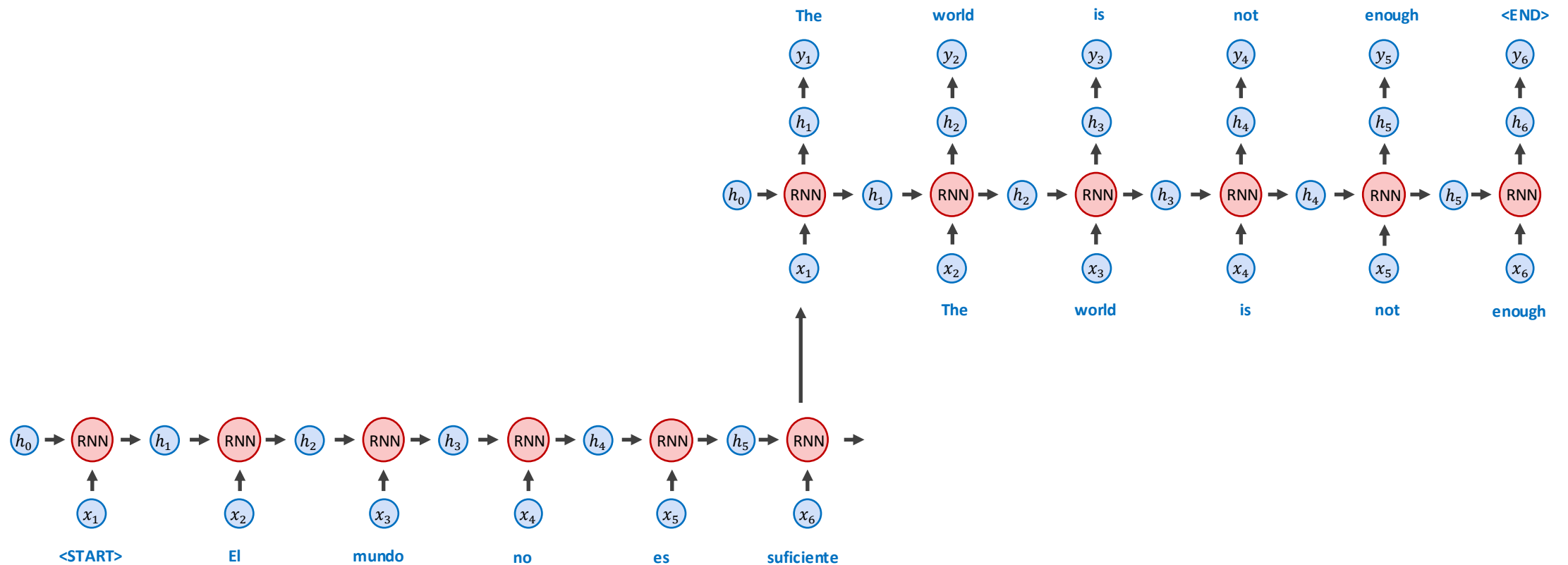
the world is not enough <END>

<START> the world is not enough

How can it be used? – e.g. Machine Translation

Sequence to Sequence – Encoding – Decoding – Many to Many mapping

DURING TRAINING – (Alternative)



Problems

- Long Sequences lead to vanishing
- Hidden states can not carry information in a long sequence (Telephone Game problem)

Solutions Proposed

- Use another hidden state variable and experiment with more complex transition functions than $h = \tanh(W_1 h + W_2 x)$.
 - Read about LSTMs, GRUs, etc

LSTM Cell (Long Short-Term Memory)

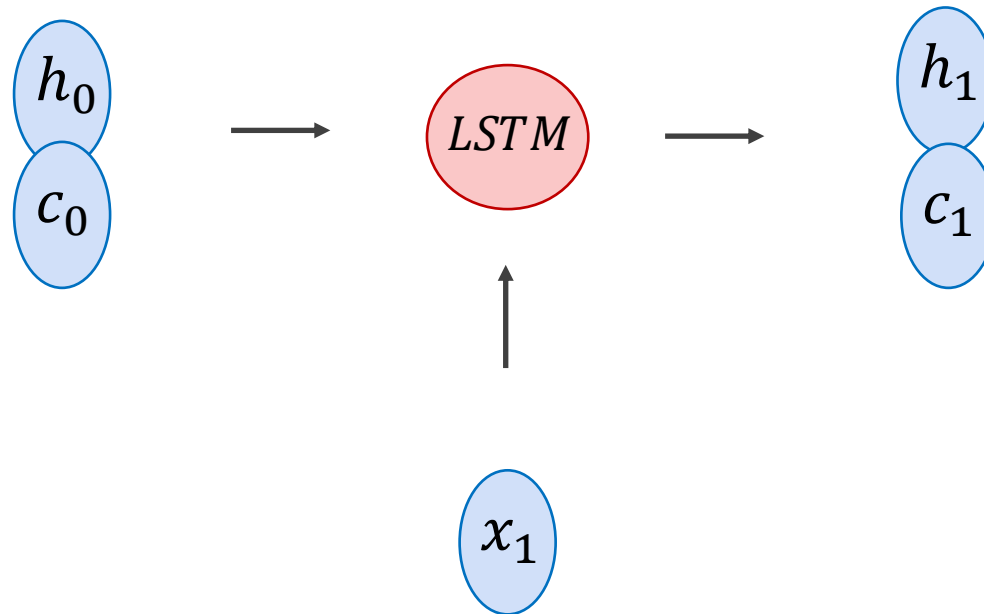
$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i) \quad (7)$$

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f) \quad (8)$$

$$c_t = f_t c_{t-1} + i_t \tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \quad (9)$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o) \quad (10)$$

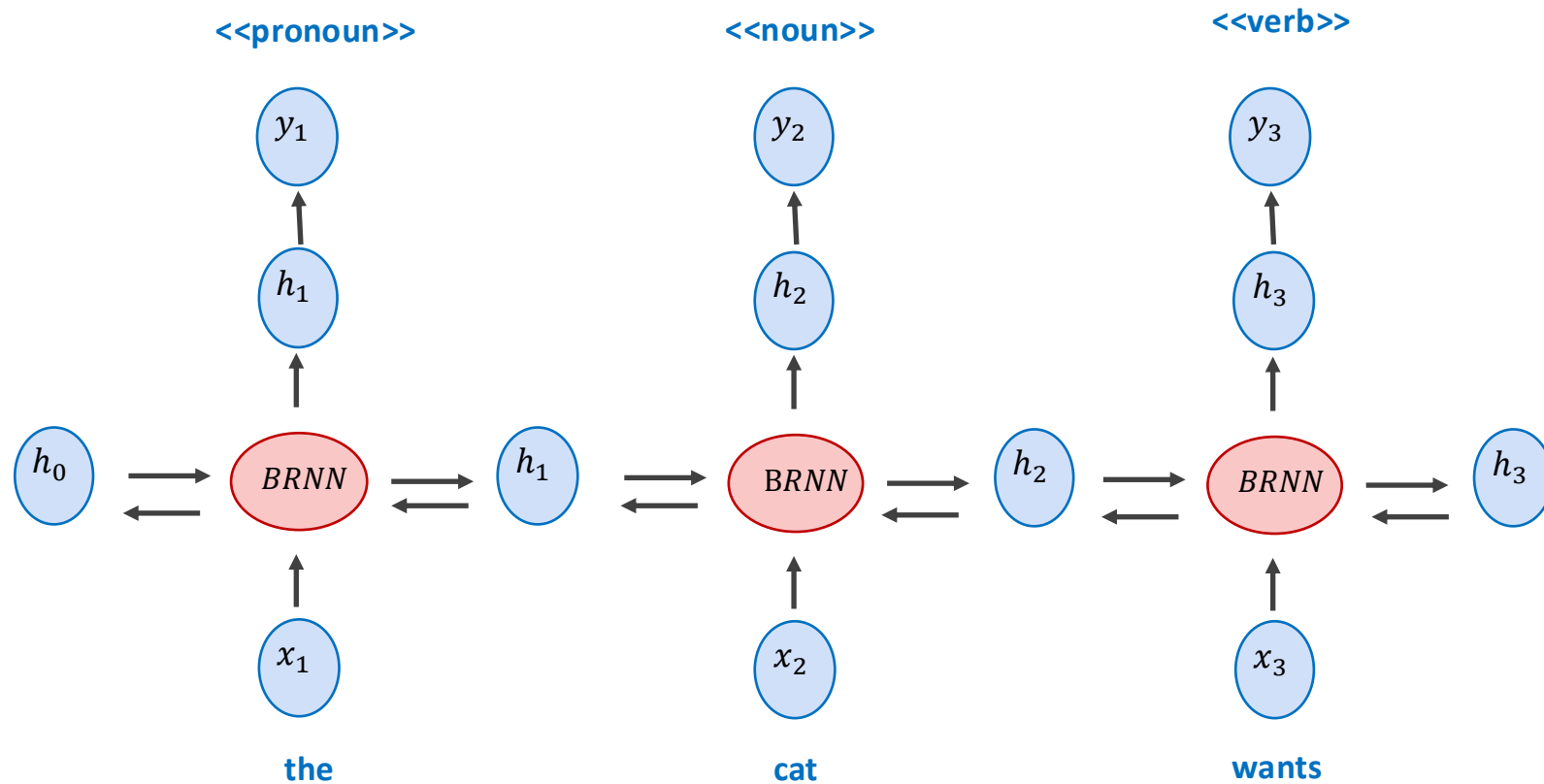
$$h_t = o_t \tanh(c_t) \quad (11)$$



Solutions Proposed

- Use another hidden state variable and experiment with more complex transition functions than $h = \tanh(W_1h + W_2x)$.
 - Read about LSTMs, GRUs, etc
- Encode the sentences both from left-to-right and right-to-left using two RNNs and combine the final hidden states from each direction.
 - Read about Bidirectional RNNs (BiRNNs), BiLSTMs, BiGRUs

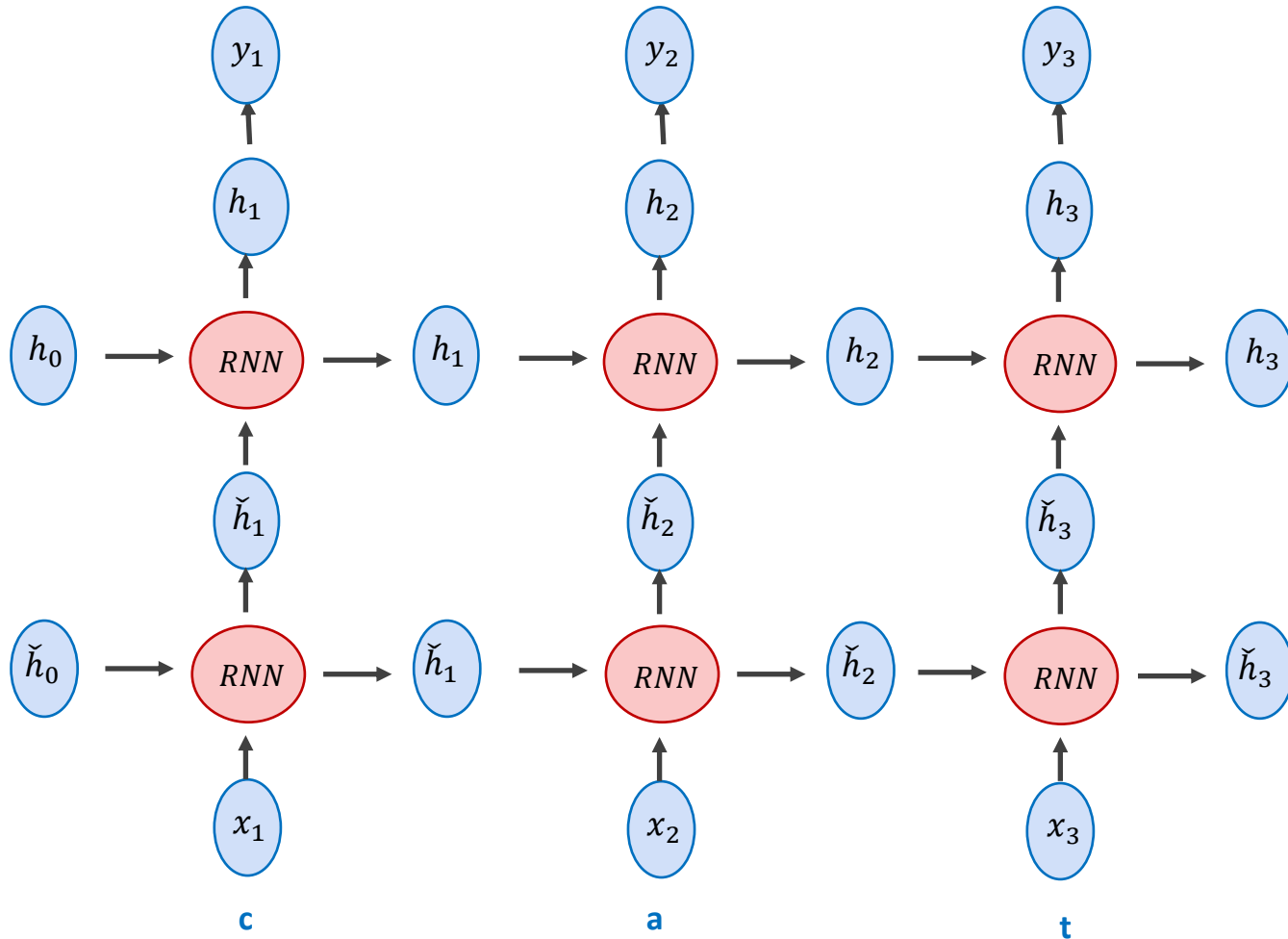
Bidirectional Recurrent Neural Network



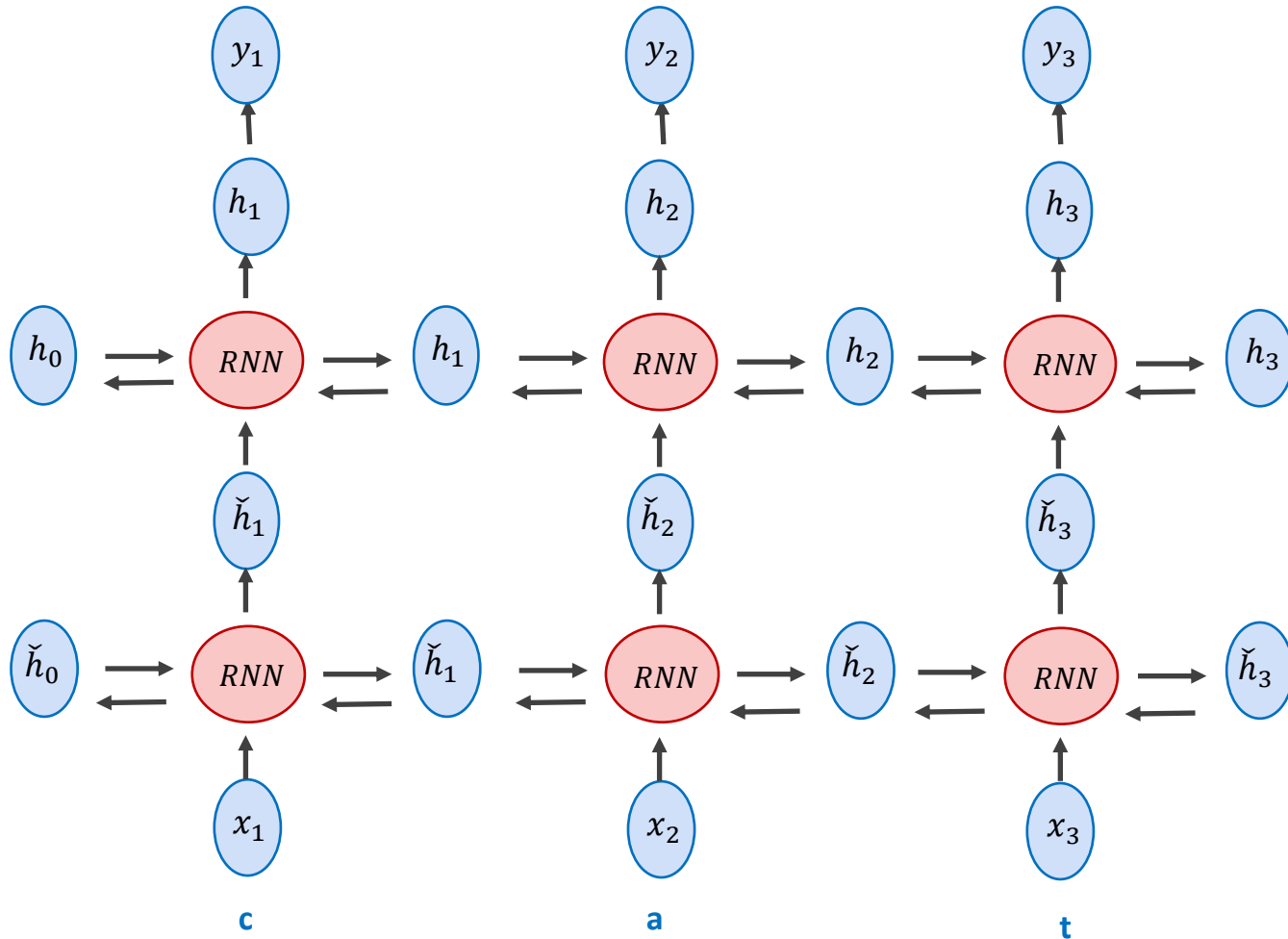
Solutions Proposed

- Use another hidden state variable and experiment with more complex transition functions than $h = \tanh(W_1h + W_2x)$.
 - Read about LSTMs, GRUs, etc
- Encode the sentences both from left-to-right and right-to-left using two RNNs and combine the final hidden states from each direction.
 - Read about Bidirectional RNNs (BiRNNs), BiLSTMs, BiGRUs
- Stack RNNs, use an RNN that feeds its output states to another RNN and this second RNN outputs the final output states.
 - Stacked RNNs, or Deep RNNs.

Stacked Recurrent Neural Network



Stacked Bidirectional Recurrent Neural Network



Best Solution: Learning Attention Weights

Questions?