A fixed-window neural language model: Pros and Cons

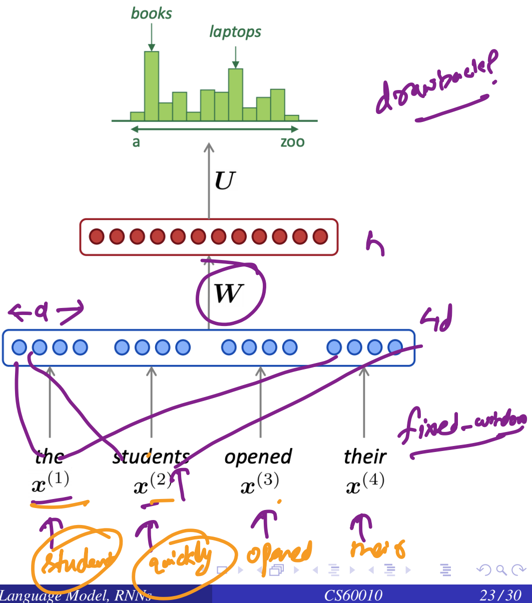
Improvements over *n*-gram LM:

- No sparsity problem
- Don't need to store all observed *n*-grams

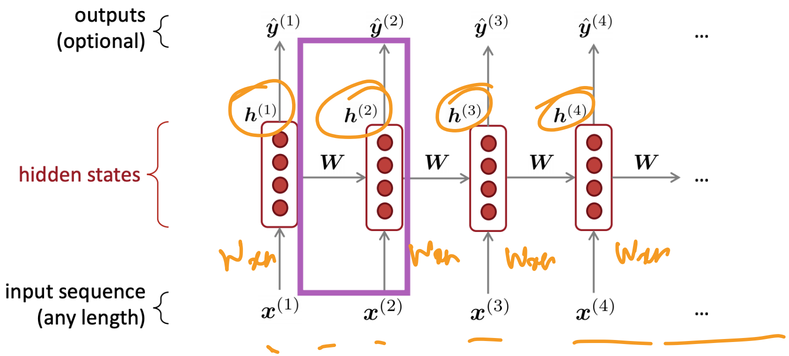
Remaining **problems**:

- Fixed window is too small
- Enlarging window enlarges W
- Window can never be large enough!
- $x^{(1)}$ and $x^{(2)}$ are multiplied by completely different weights in W. No symmetry in how the inputs are processed.

We need a neural architecture that can process *any length input*



Recurrent Neural Networks



Core Idea

Apply the same weights repeatedly!

Recurrent Neural Networks

We can process a sequence of vectors x by applying a recurrence formula at each step:

$$\boxed{h_t} = \boxed{f_W}(\boxed{h_{t-1}}, \boxed{x_t})$$

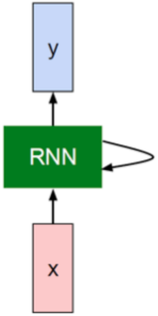
new state

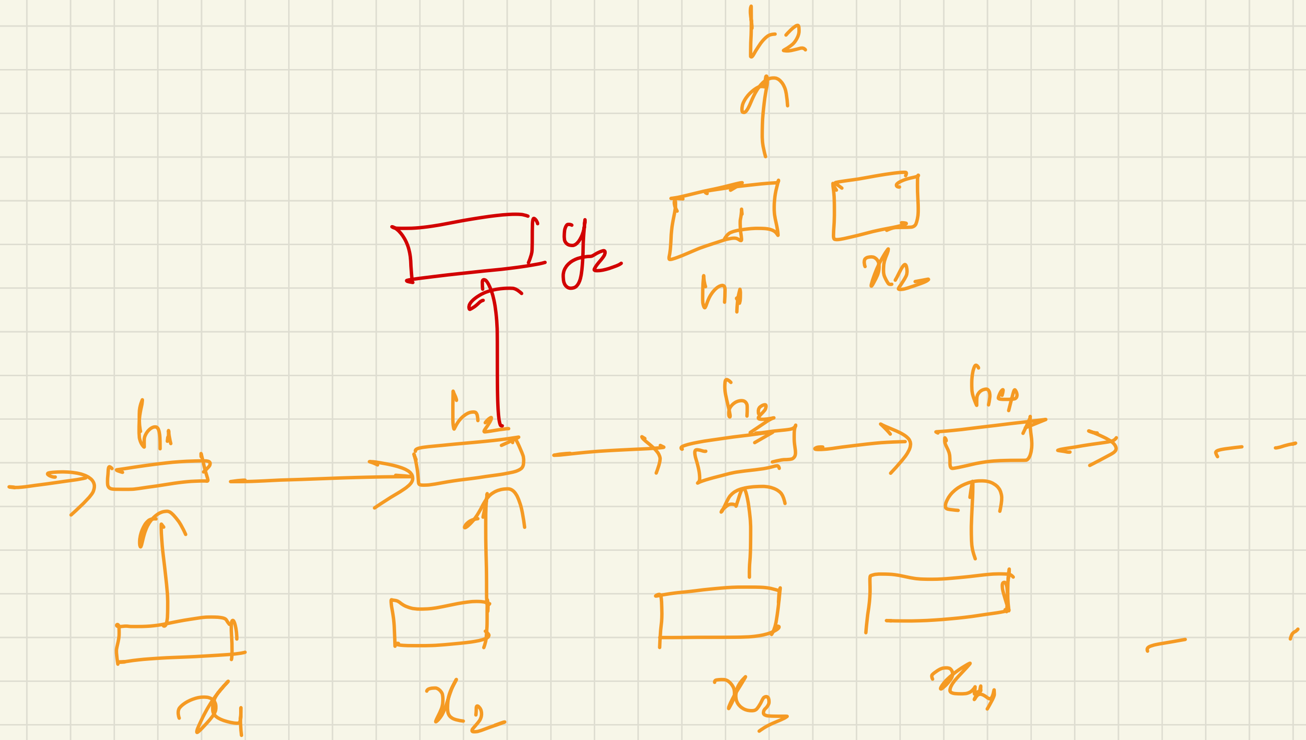
old state

input vector at some time step

some function with parameters W

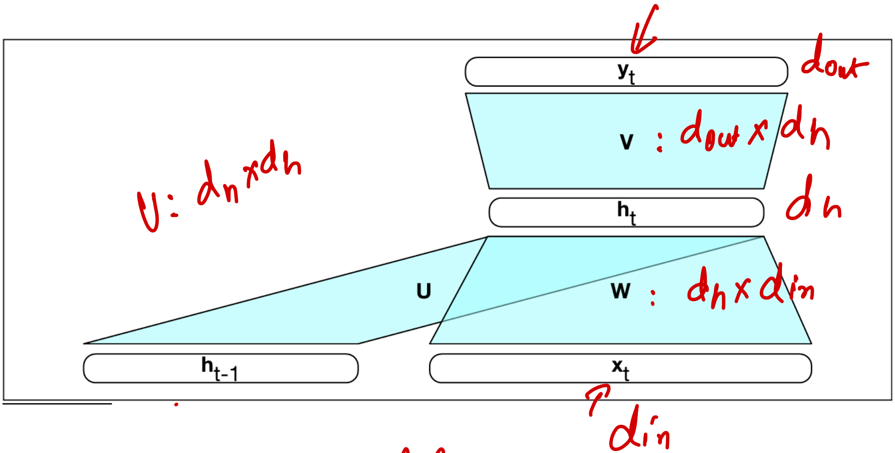
Notice: the same function and the same set of parameters are used at every time step.





h_2
y_2
h_1
x_2
h_1
h_2
h_3
h_4
x_1
x_2
x_3
x_4

RNN as a feed-forward network



$$h_t = f(Uh_{t-1} + Wx_t + b)$$

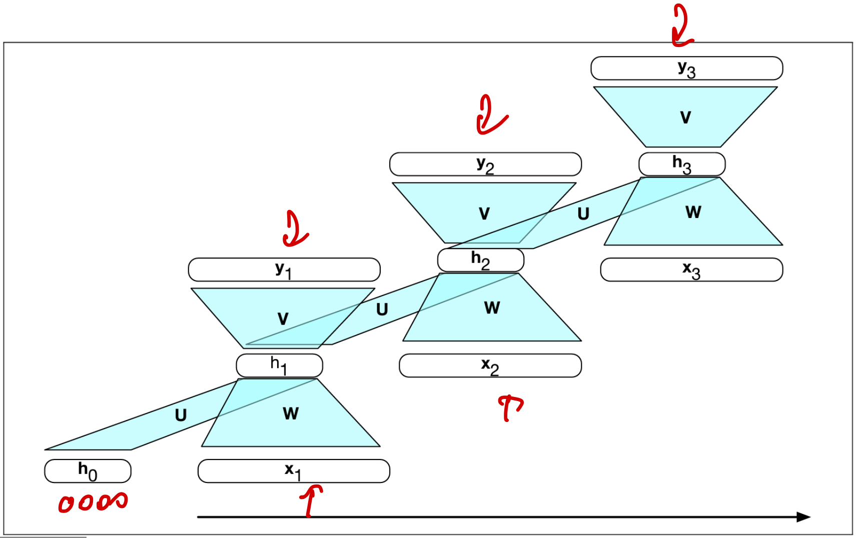
Forward Propagation

$$h_t = g(Uh_{t-1} + Wx_t)$$

$$y_t = softmax(Vh_t)$$

- Let the dimensions of the input, hidden and output be d_{in}, d_h and d_{out}, respectively
- The three parameter matrices: $\underline{W : d_h \times d_{in},\ U : d_h \times d_h,\ V : d_{out} \times d_h}$

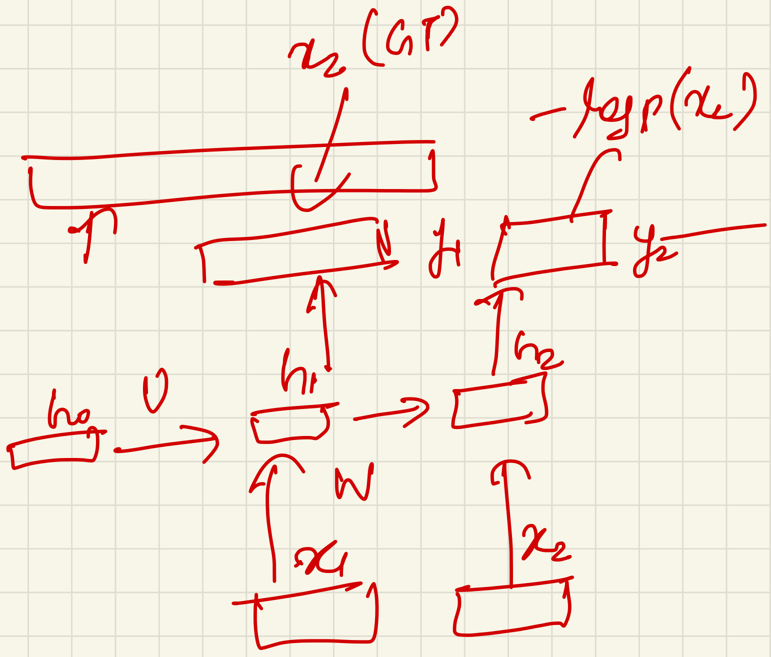
RNN Unrolled in Time



Train RNN LM

Corpus

Self-supervision



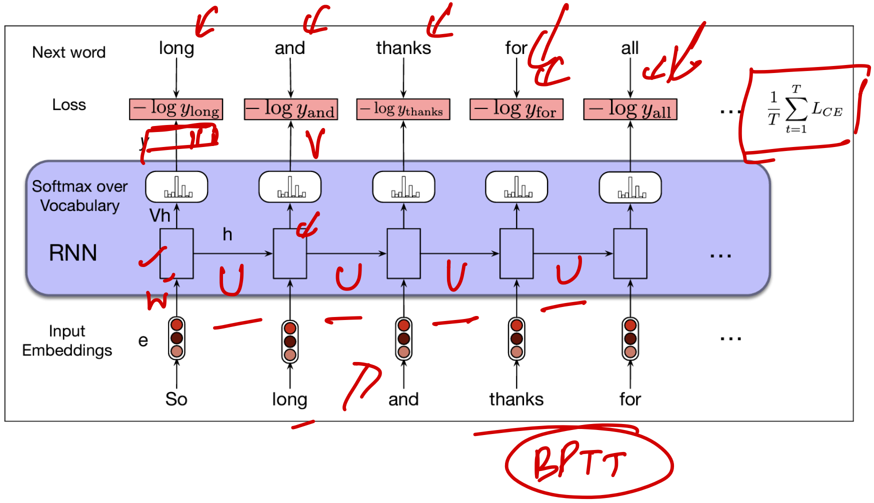
Training an RNN language model

- To train RNN LM, we use self-supervision (or self-training)
- We take a corpus of text as training material
- At each time step t, we ask the model to predict the next word

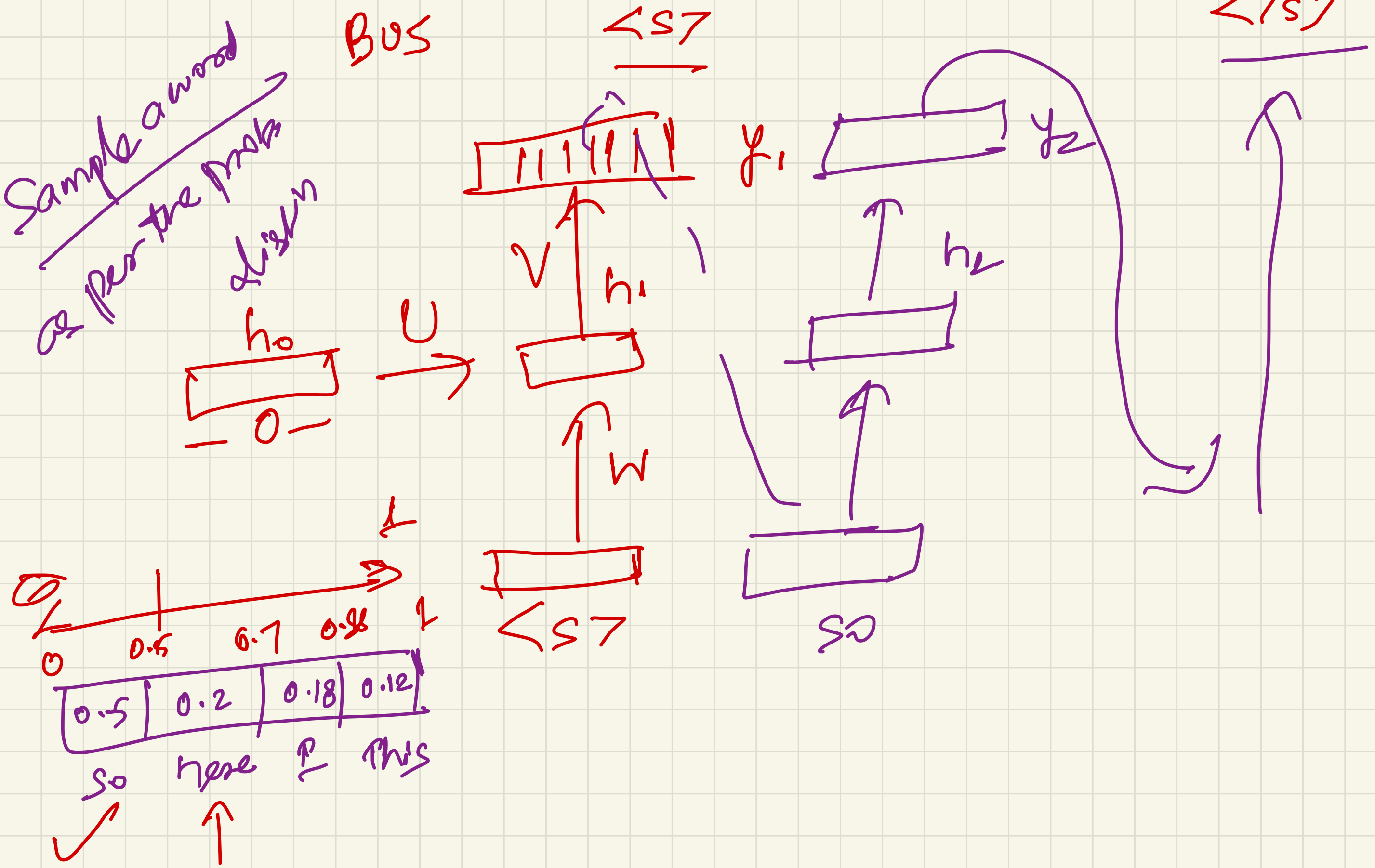
Why is it called self-supervision?

- We do not add any gold data, the natural sequence of words is its own supervision!
- We simply train the model to minimize the error in predicting the true next word in the training sequence

Training an RNN language model



Sample a word
a per the prob
distn



Generating text with an RNN Language Model

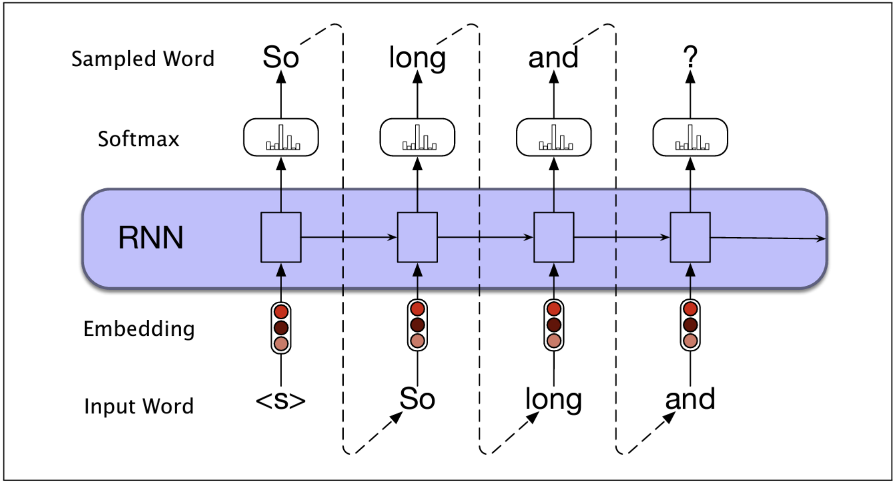
$\underline{U, V, W}$

- RNN-based language models can be used for language generation (and hence, for machine translation, dialog, etc.)
- A language model can incrementally generate words by repeatedly sampling the words conditioned on the previous choices – also known as *autoregressive generation*.

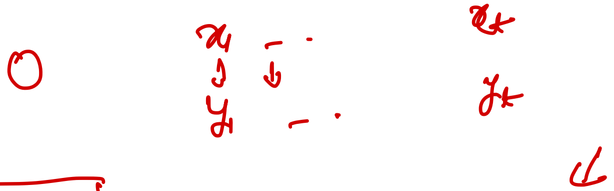
Autoregressive Generation with RNNs

- All your parameters have already been trained.
- Start with a special begin of sentence token <s> as input
- Through forward propagation, obtain the probability distribution at the output, and sample a word
- Feed the word as input at the next time-step (its word vector)
- Continue generating until the end of sentence token is sampled, or a fixed length of the sentence has been reached.

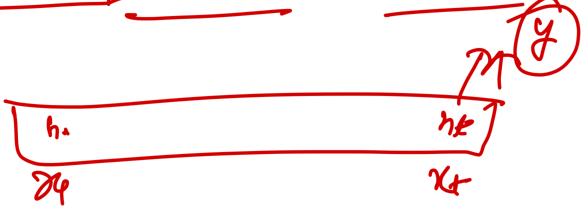
Autoregressive Generation with RNNs



RNNs can be used for various other applications



- Sequence labeling: Named Entity Recognition, Parts-of-Speech Tagging
- Text Classification: Sentiment Analysis, Spam Detection



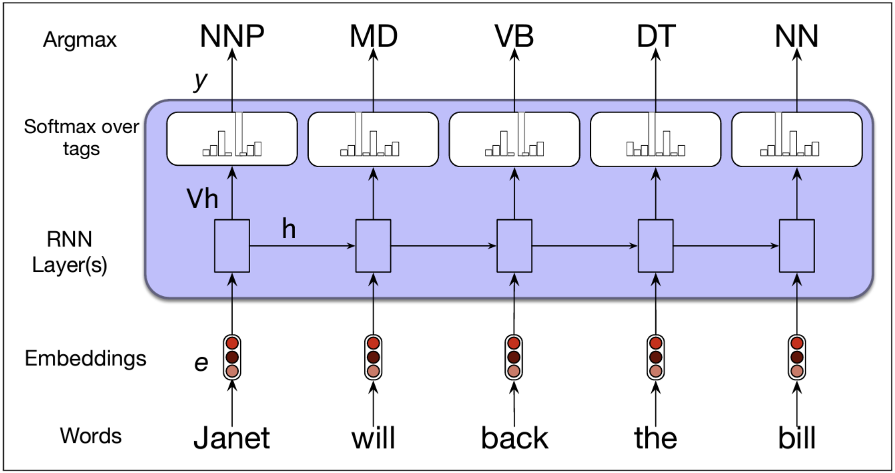
RNNs for Sequence Labeling

Task

Assign a label chosen from a small fixed set of labels to each element of the sequence

- Inputs: Word embeddings
- Outputs: Tag probabilities generated by the softmax layer

RNNs for Sequence Labeling



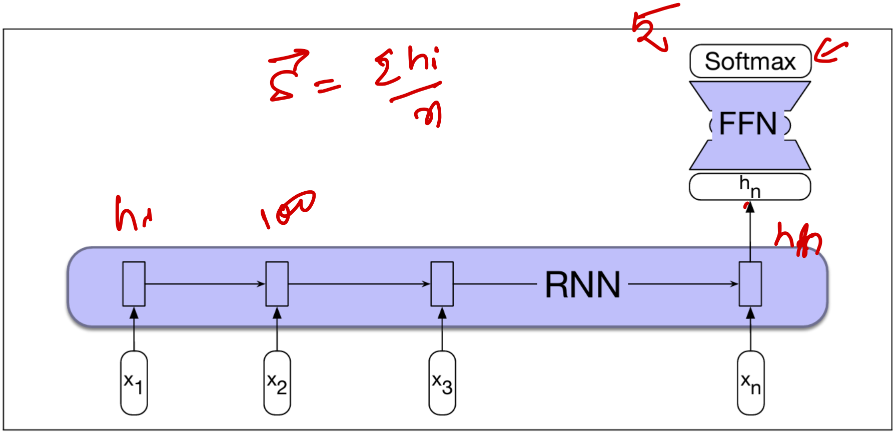
RNNs for Sequence Classification

Task

Classify the entire sequence rather than the token within them

- Pass the text to be classified a word at a time, generating new hidden states at each time step
- The hidden state of the last token can be thought of as a compressed representation of the entire sequence
- This last hidden state is passed through a feed-forward network that chooses a class via softmax
- There are other options of combining information from all the hidden states

RNNs for Sequence Classification



300-d

Other Variations: Stacked RNNs

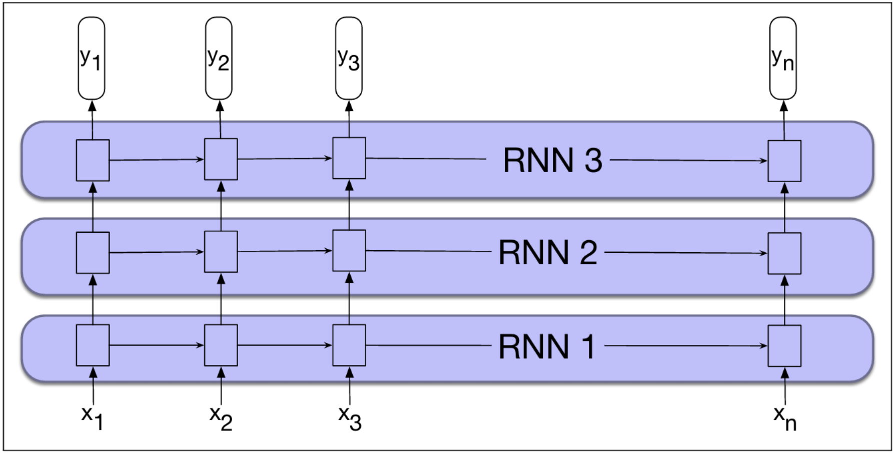


Figure 9.10 Stacked recurrent networks. The output of a lower level serves as the input to higher levels with the output of the last network serving as the final output.

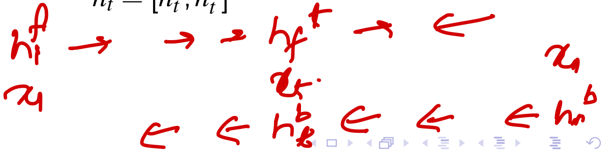
Other Variations: Bidirectional RNNs

- RNN makes use of information from left (prior) context to predict at time t
- In many applications, the entire sequence is available; so it makes sense to also make use of the right context to predict at time t
- Bidirectional RNNs combine two independent RNNs, one where the input is processed from left to right (forward RNN), and another from end to the start (backward RNN).

$$h_t^f = RNN_{forward}(x_1, \ldots, x_t)$$

$$h_t^b = RNN_{backward}(x_n, \ldots, x_t)$$

$$h_t = [h_t^f ; h_t^b]$$



Other Variations: Bidirectional RNNs

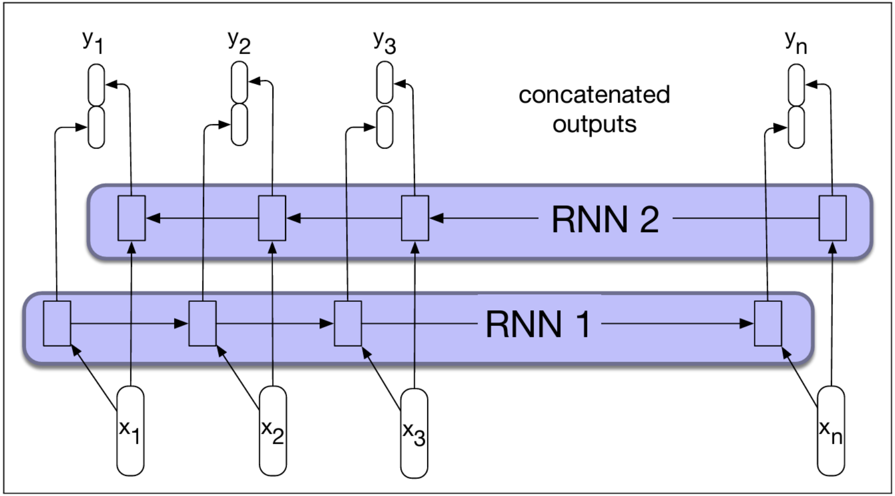


Figure 9.11 A bidirectional RNN. Separate models are trained in the forward and backward directions, with the output of each model at each time point concatenated to represent the bidirectional state at that time point.