Amirkabir University of Technology
(Tehran Polytechnic)

Natural Language Processing

Lecture 11: Sequence Modeling with RNNs

Amirkabir University of Technology
Dr Momtazi

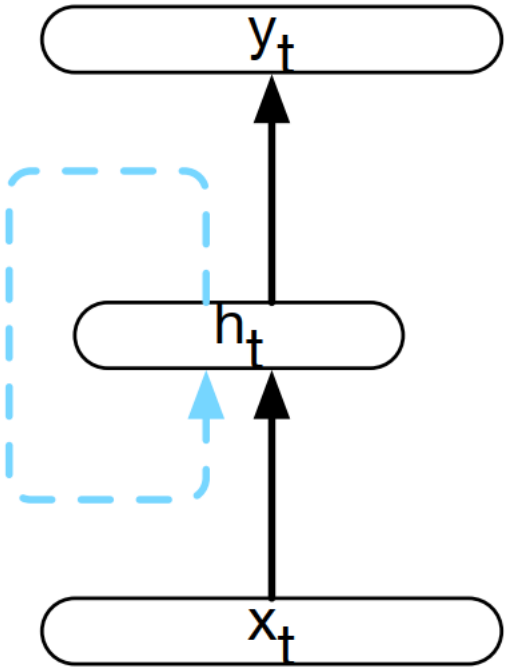
Outline

- **Recurrent Neural Networks**
- RNN Applications
- Stacked and Bidirectional RNNs
- LSTM and GRU

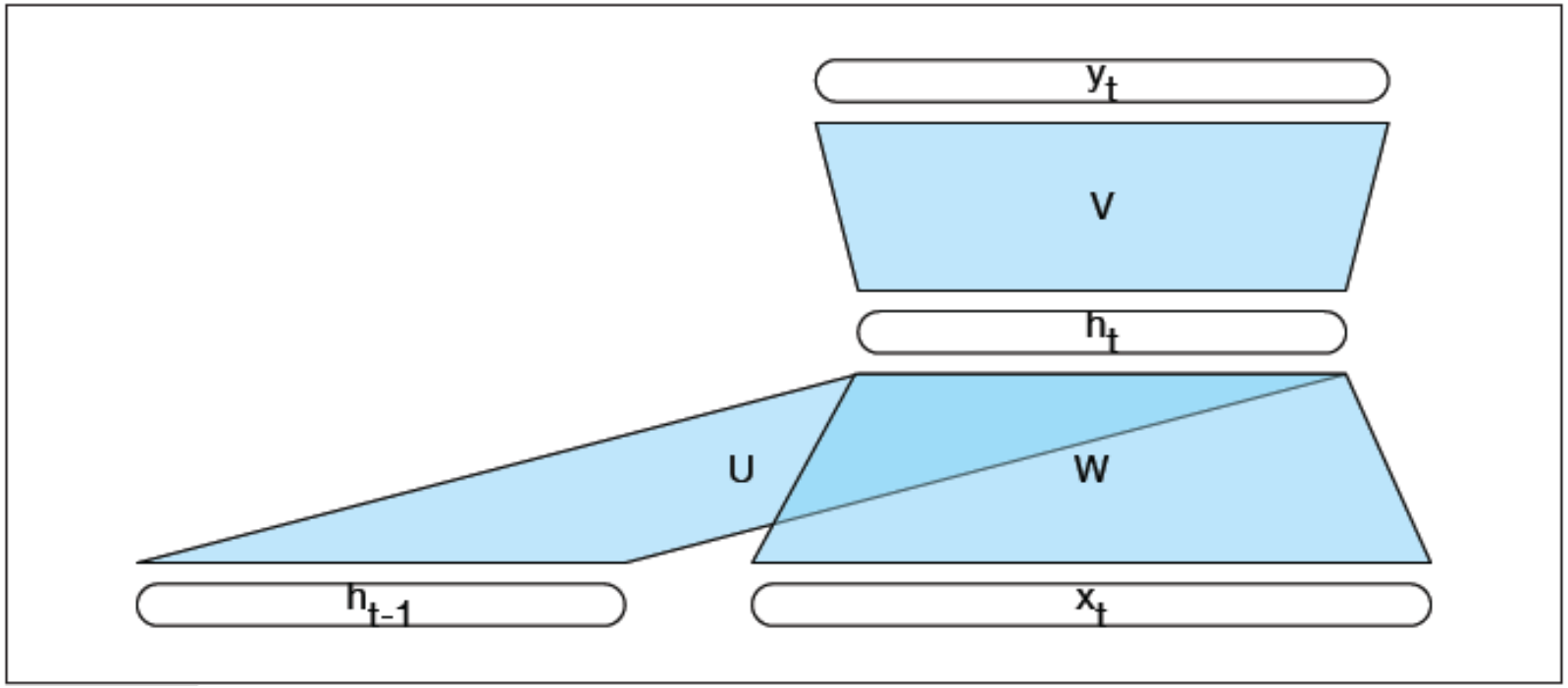
Recurrent Neural Networks (RNNs)

- To be used in scenarios where a sequence of inputs and/or outputs is being modelled
- RNNs have memory that captures information about what has been computed so far
- Performing the same task for every element of sequence, with output dependent on previous computations
- In theory, RNNs can make use of information in arbitrarily long sequences - in practice, however, they are limited to looking back only few steps

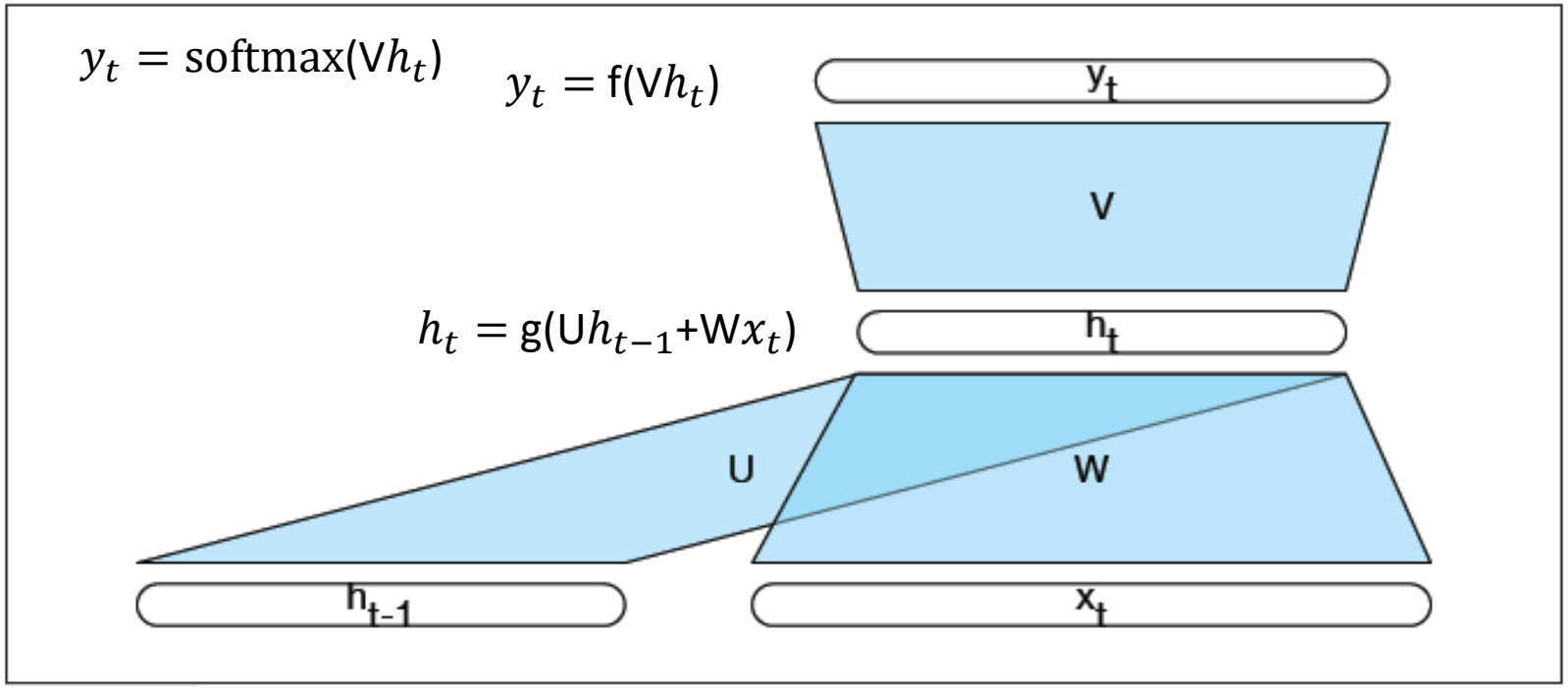
RNN Structure



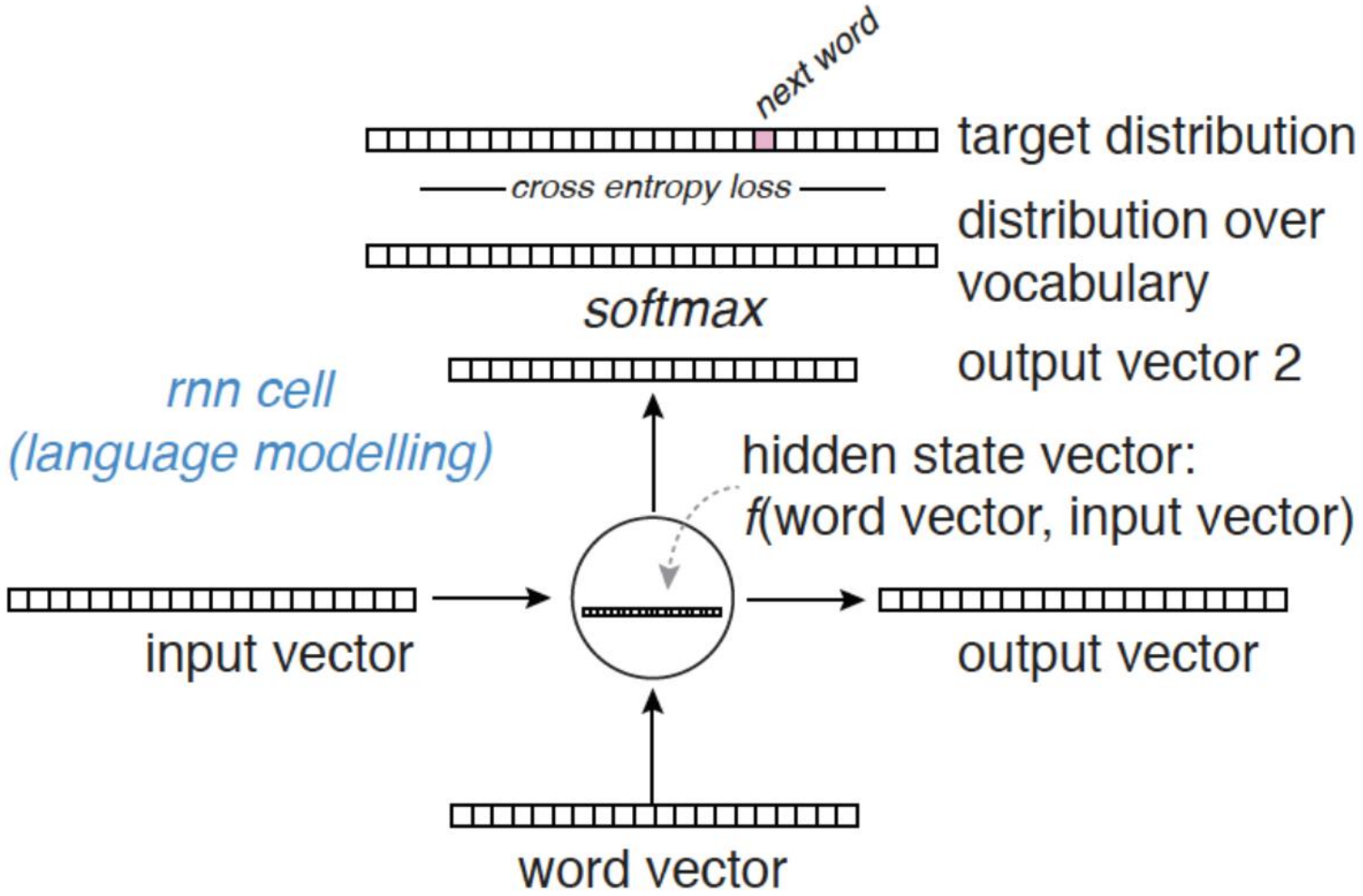
RNN Structure



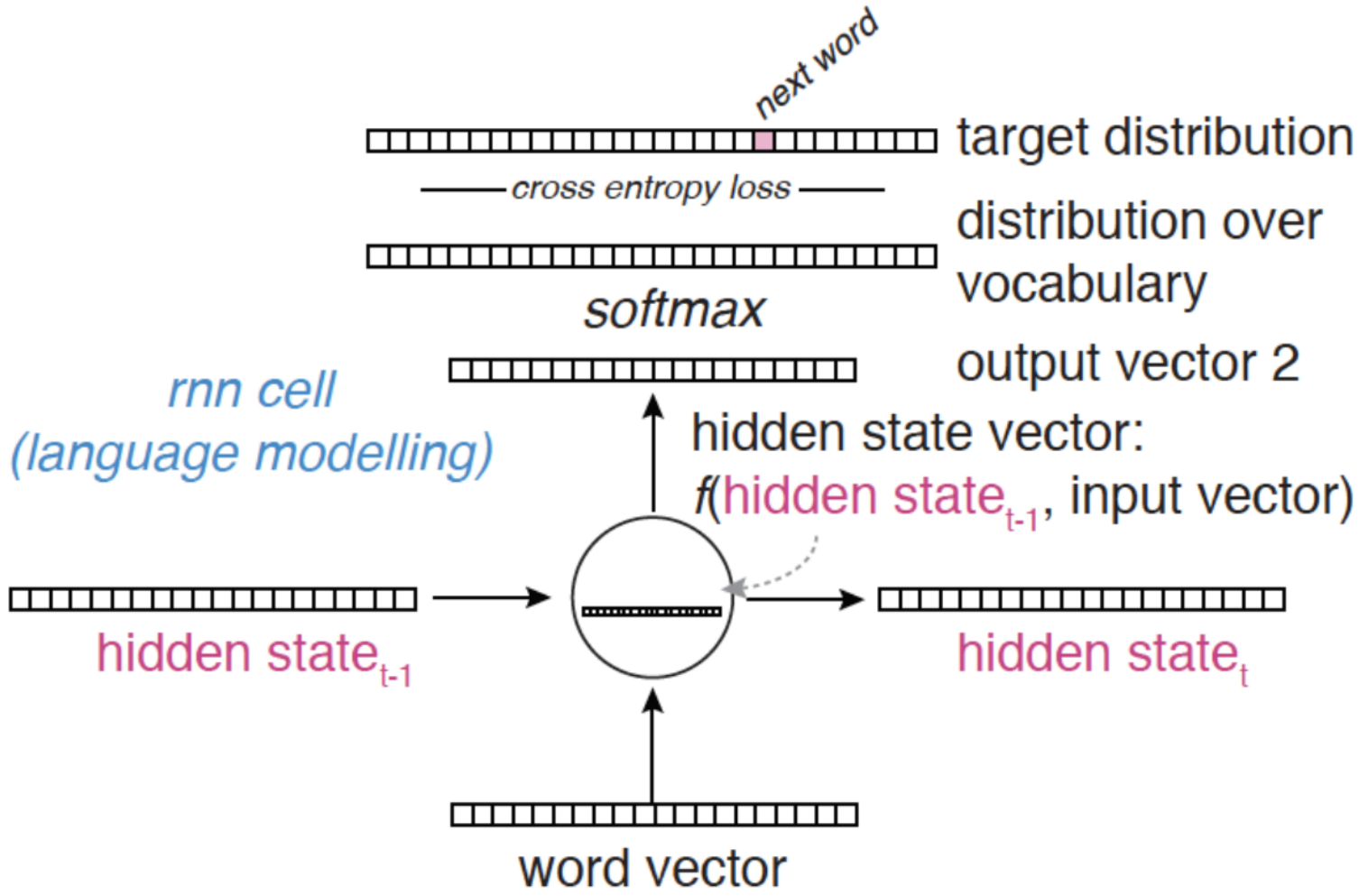
RNN Structure



Recurrent Cell

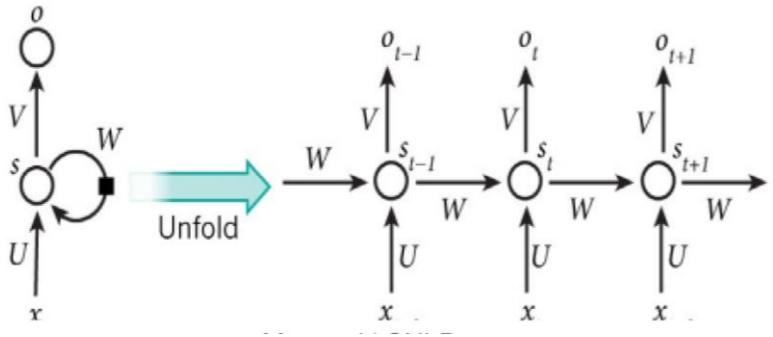


Recurrent Cell



Recurrent Neural Networks

- RNN being unrolled (or unfolded) into full network
- Unrolling: write out network for complete sequence
- Formulas governing computation:
  - x_t input at time step t
  - s_t hidden state at time step t - memory of the network, calculated based on:
    - input at the current step $S_t = f(U_{x_i} + W_{S_{t-1}})$
    - previous hidden state
  - f usually nonlinearity, e.g., tanh or ReLU (it can also be LSTM or GRU.)

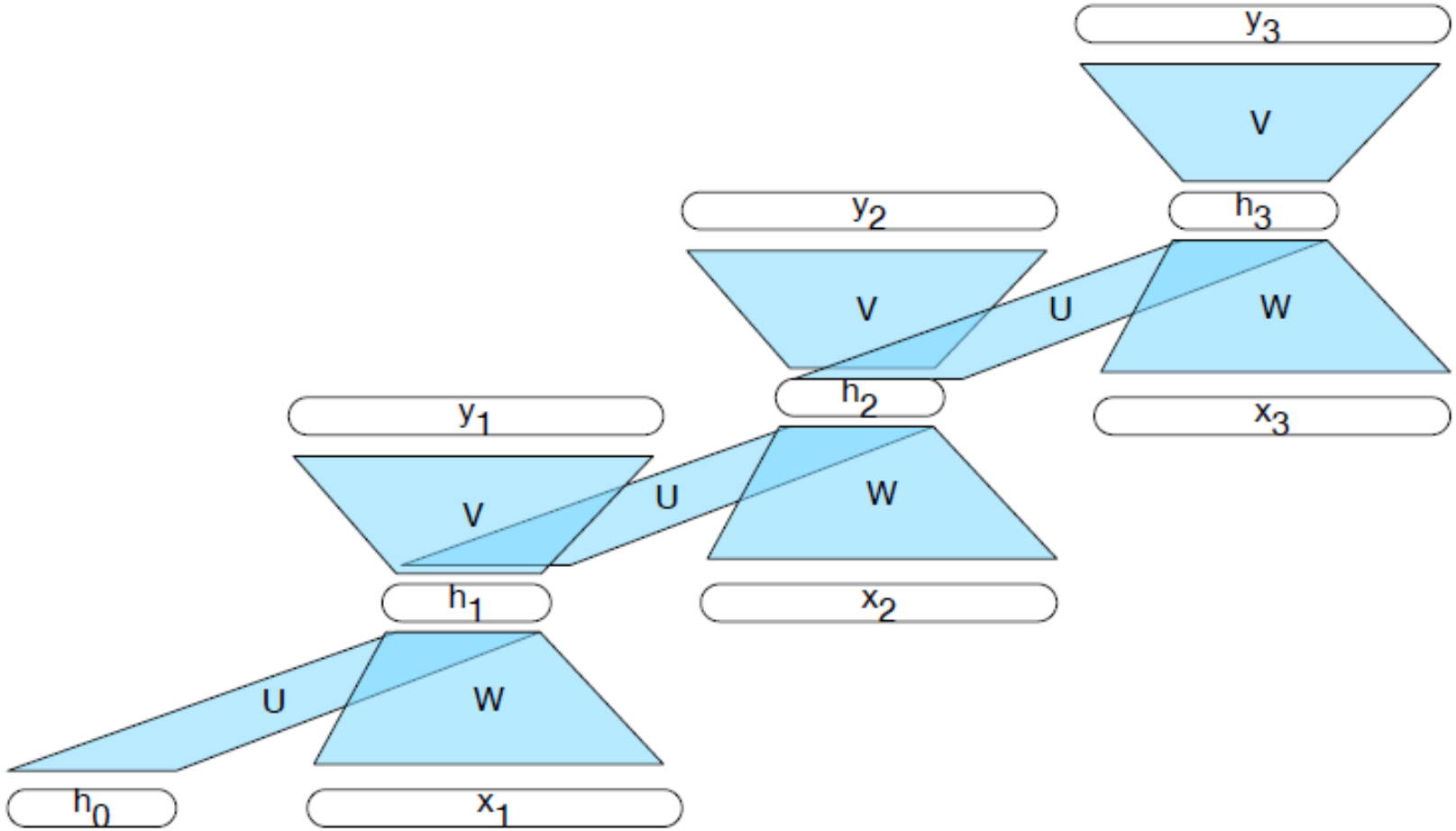


Inference in RNN

function FORWARDRNN(x, *network*) **returns** output sequence y

$h_0 \leftarrow 0$
for $i \leftarrow 1$ **to** LENGTH(x) **do**
    $h_i \leftarrow g(U\ h_{i-1} + W\ x_i)$
    $y_i \leftarrow f(V\ h_i)$
return y

Unrolled RNN



Outline

- Recurrent Neural Networks
- **RNN Applications**
- Stacked and Bidirectional RNNs
- LSTM and GRU

RNN Applications

- Language Modelling
- Sequence Labeling
- Text Generation
- Text Classification

RNN Applications

- Fully-connected networks use fixed- size inputs, along with associated weights, to capture all the relevant aspects of an example at once.
- This makes it difficult to deal with sequences of varying length and fails to capture important temporal aspects of language
- Anything outside the context window has no impact on the decision being made.

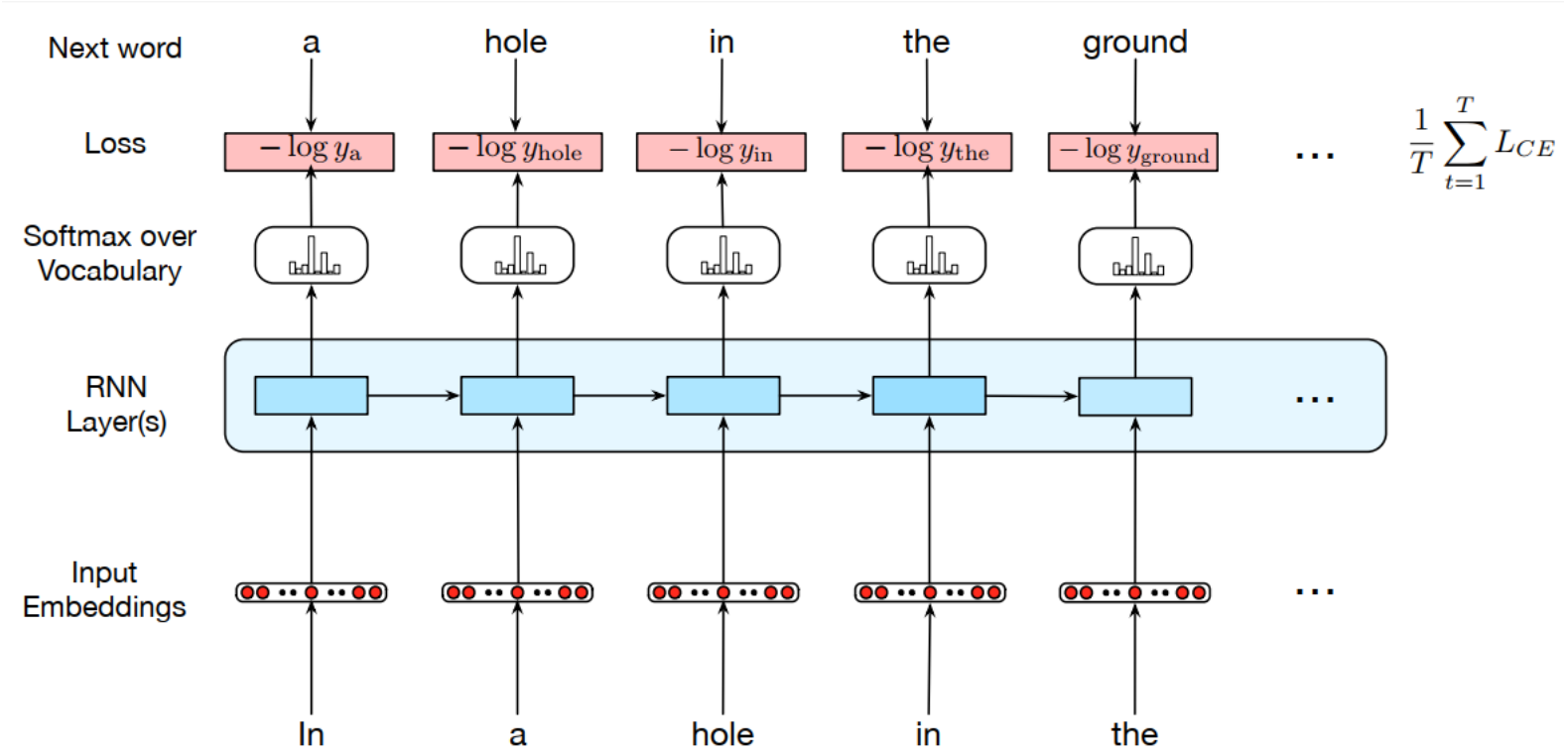
Language modeling using RNNs

- Language model allows us to predict probability of observing sentence (in a given dataset) as:

$$P(w_1, w_2, \ldots, w_n)$$

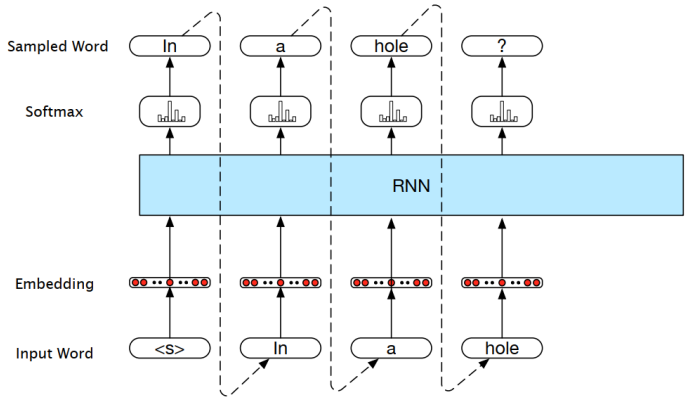
- In RNN, set: we want output at step t to be actual next word
- Cross-entropy loss as loss function
- Training RNN similar to training a traditional NN backpropagation algorithm, but with small twist
  - Parameters shared by all time steps, so gradient at each output depends on calculations of previous time steps
  - Backpropagation Through Time

Training RNNs as Language models

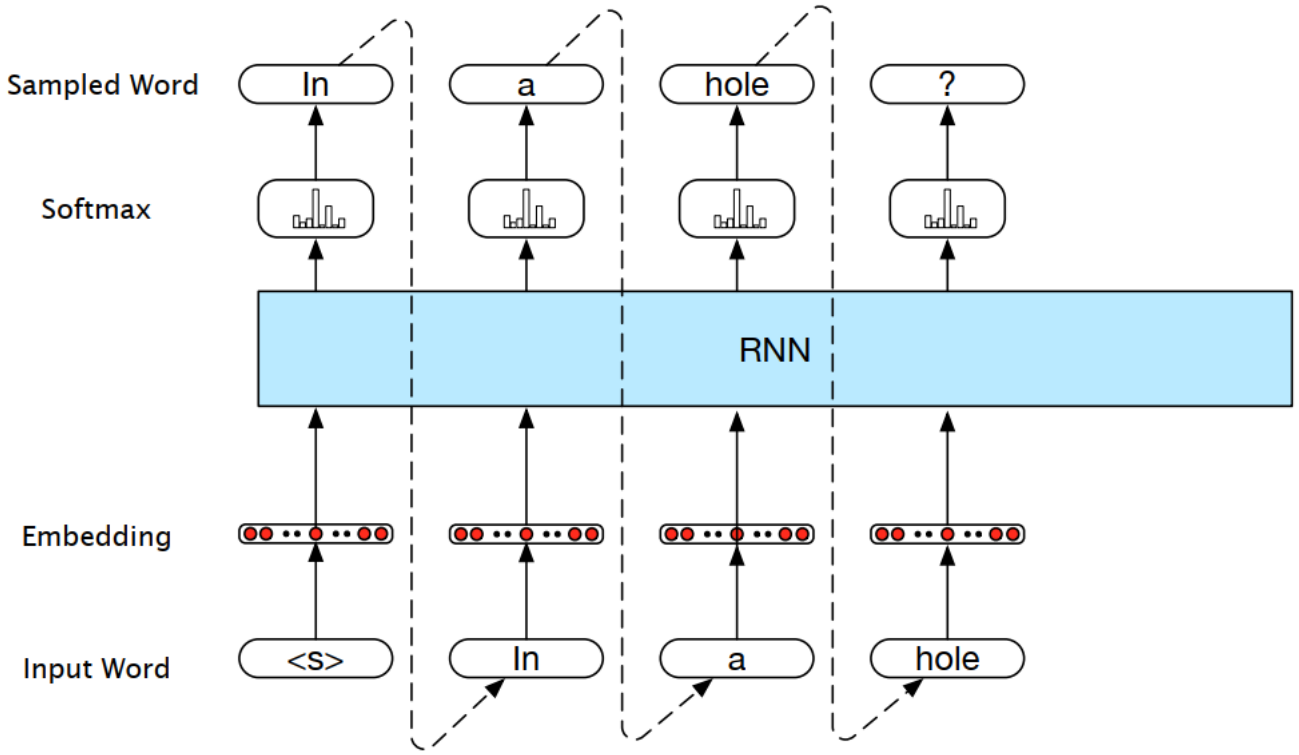


Text Generation with RNNs

- To begin, sample a word in the output from the softmax distribution that results from using the beginning of sentence marker, <s>, as the first input.
- Use the word embedding for that first word as the input to the network at the next time step, and then sample the next word in the same fashion.
- Continue generating until the end of sentence marker, </s>, is sampled or a fixed length limit is reached.
- This technique is called **autoregressive** generation since the word generated at each time step is conditioned on the word selected by the network from the previous step.

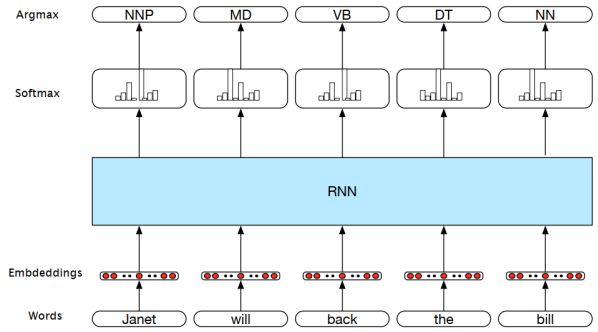


Text Generation with RNNs

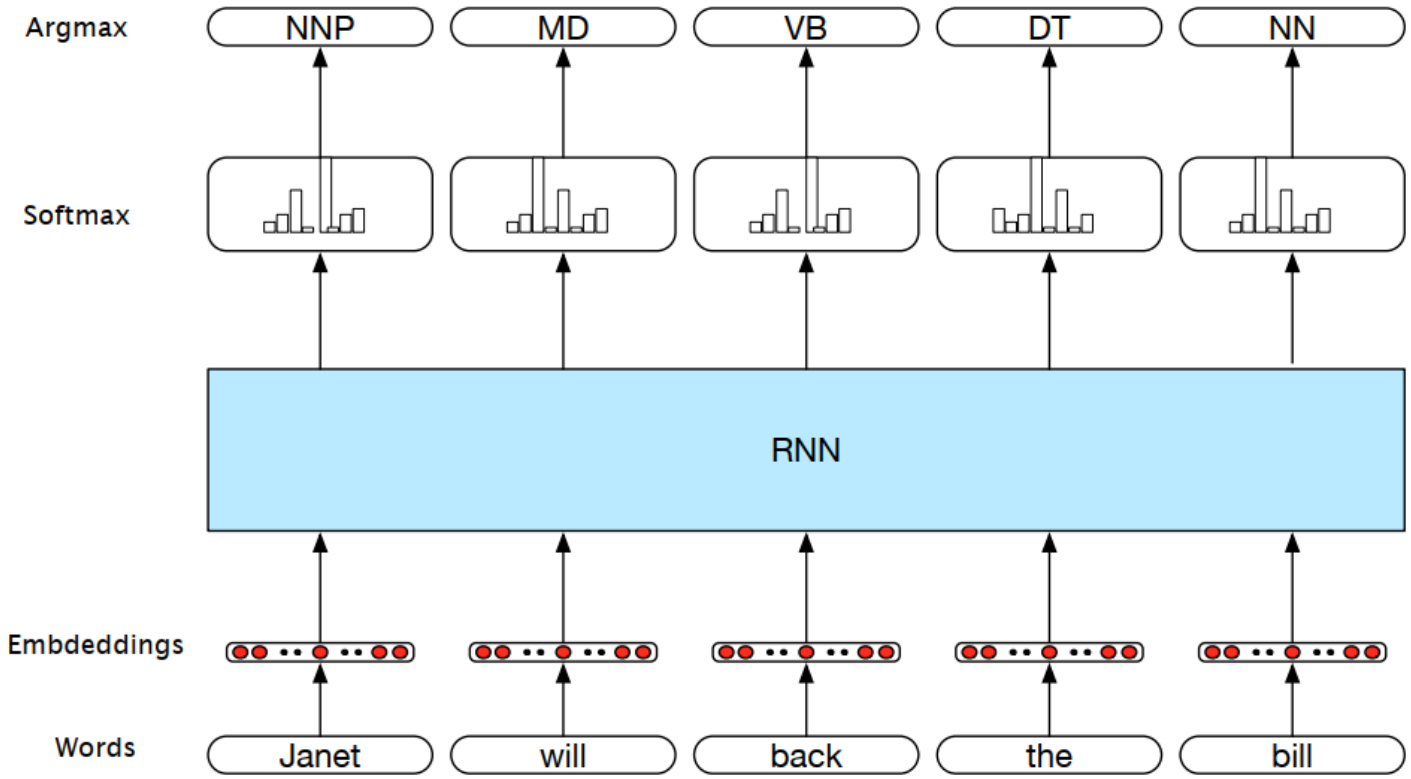


Sequence Labeling with RNNs

- In sequence labeling, the network's task is to assign a label chosen from a small fixed set of labels to each element of a sequence.
- Canonical examples of sequence labeling include
  - Part-of-speech tagging
  - Named entity recognition
- To generate a sequence of tags for a given input, we run forward inference over the input sequence and select the most likely tag from the softmax at each step.
- Since we're using a softmax layer to generate the probability distribution over the output tagset at each time step, we will employ the cross-entropy loss during training.

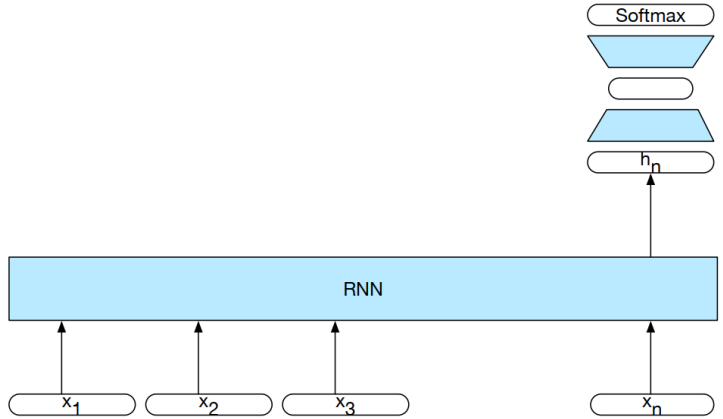


Sequence Labeling with RNNs



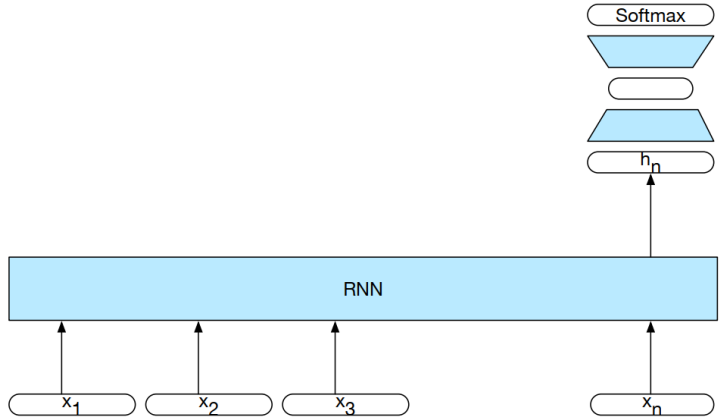
Text Classification with RNNs

- Another use of RNNs is to classify entire sequences rather than the tokens within Them.
- In all of these applications, sequences of text are classified as belonging to one of a small number of categories.
- To apply RNNs in this setting, the text to be classified is passed through the RNN a word at a time generating a new hidden layer at each time step.
- The hidden layer for the final element of the text, h_n, is taken to constitute a compressed representation of the entire sequence. In the simplest approach to classification, h_n, serves as the input to a subsequent feedforward network that chooses a class via a softmax over the possible classes

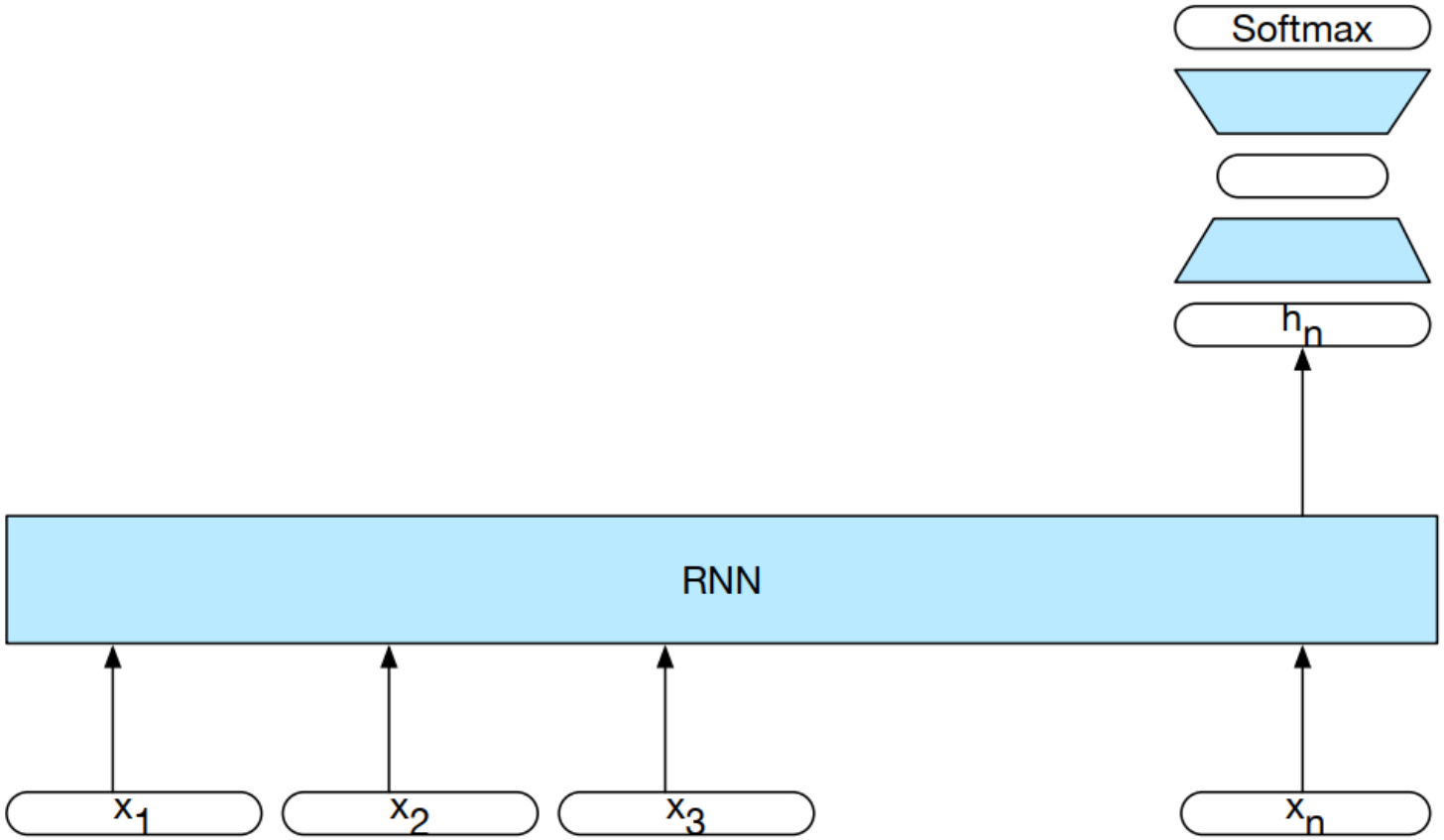


Text Classification with RNNs

- Note that in this approach there are no intermediate outputs for the words in the sequence preceding the last element.
- Therefore, there are no loss terms associated with those elements.
- Instead, the loss function used to train the weights in the network is based entirely on the final text classification task.
- Specifically, the output from the softmax output from the feedforward classifier together with a cross-entropy loss drives the training.
- The error signal from the classification is backpropagated all the way through the weights in the feedforward classifier through, to its input, and then through to the three sets of weights in the RNN



Text Classification with RNNs

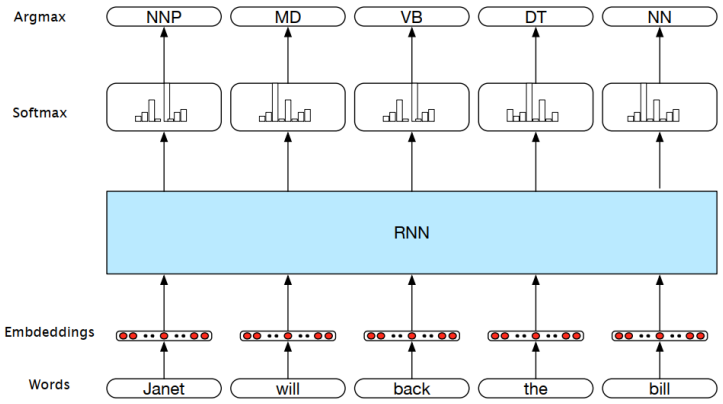


Outline

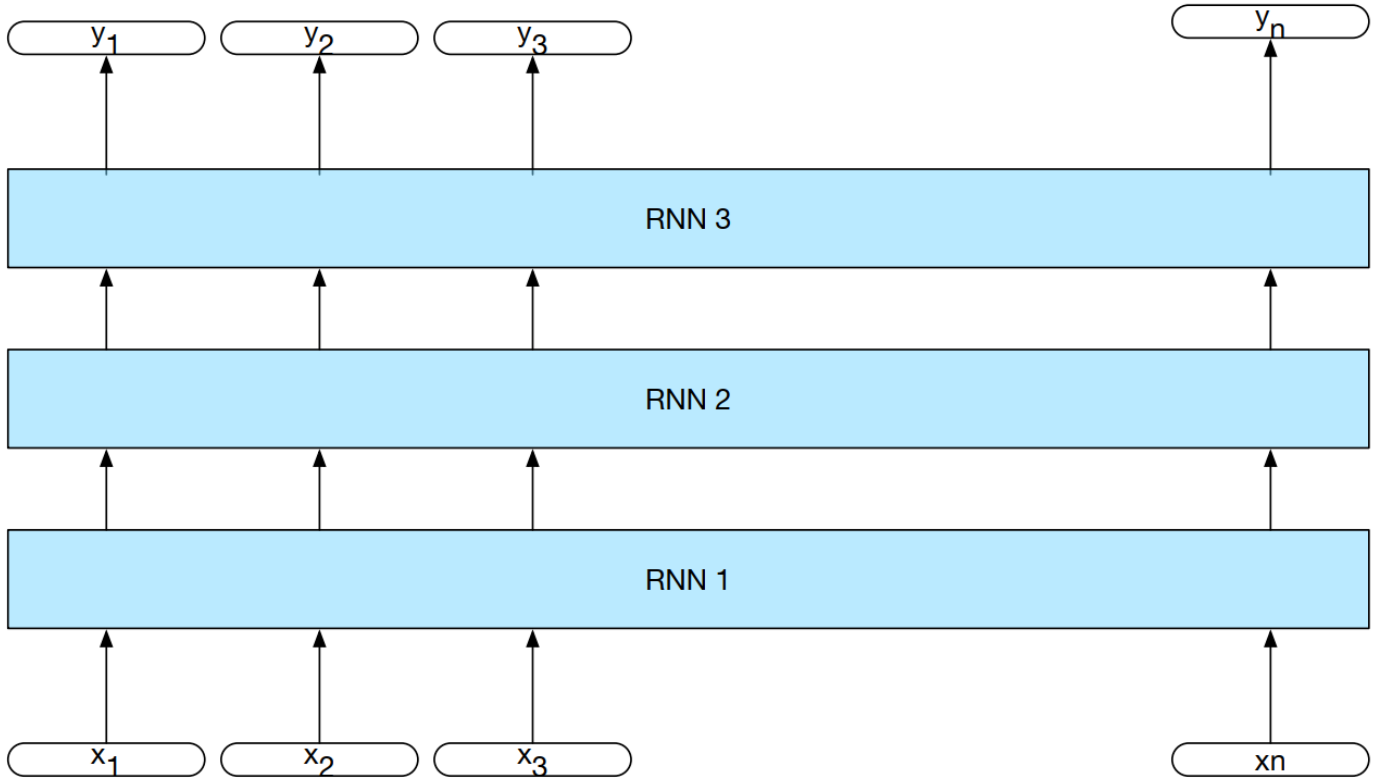
- Recurrent Neural Networks
- RNN Applications
- **Stacked and Bidirectional RNNs**
- LSTM and GRU

Stacked RNNs

- Using the entire sequence of outputs from one RNN as an input sequence to another one.
- Increasing the network's ability to induce representations at different levels of abstraction across layers.
- Just as the early stages of the human visual system detect edges that are then used for finding larger regions and shapes, the initial layers of stacked networks can induce representations that serve as useful abstractions for further layers
  - Representations that might prove difficult to induce in a single RNN.

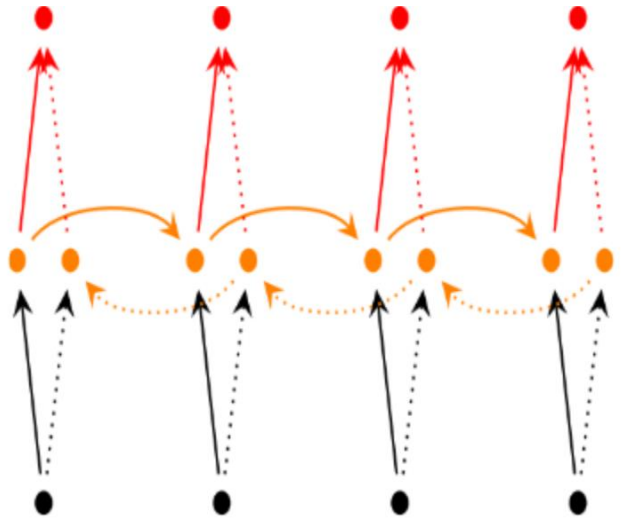


Stacked RNNs

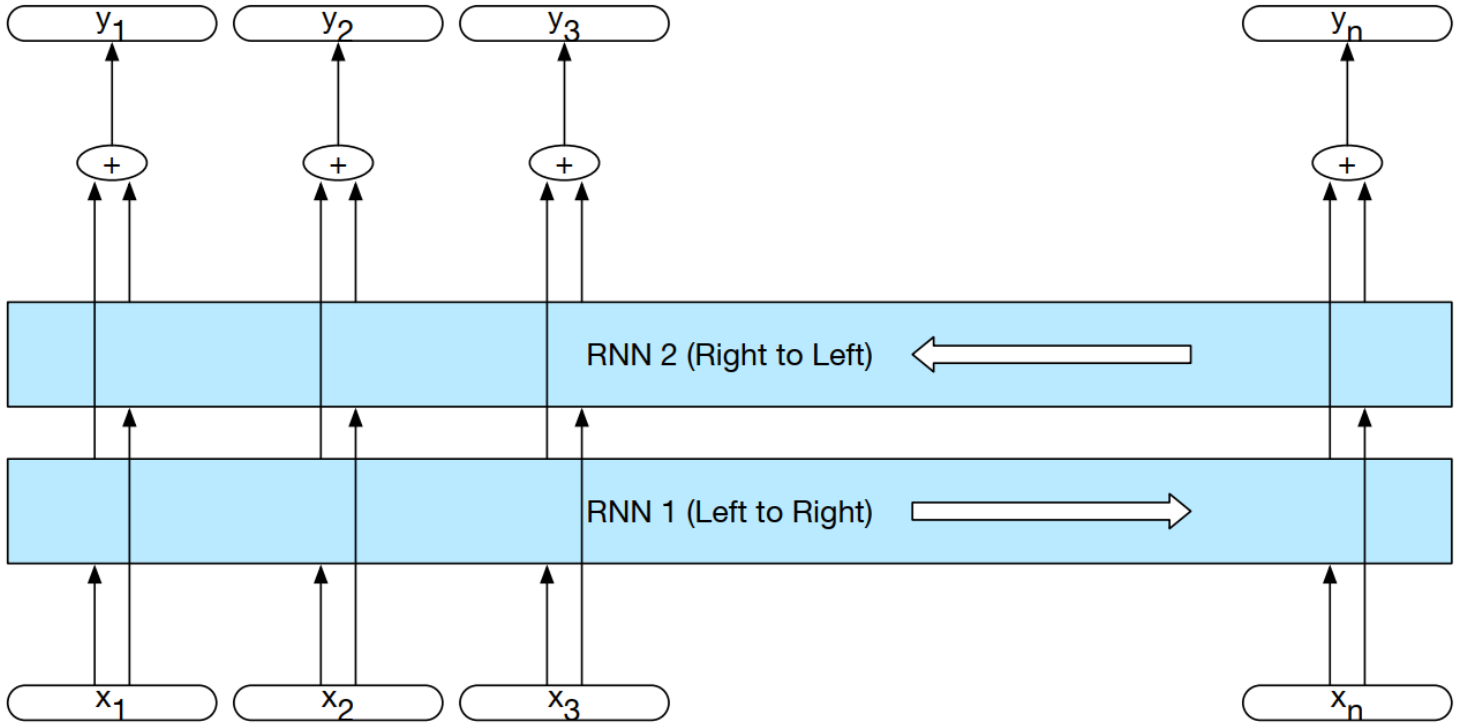


Bidirectional RNNs

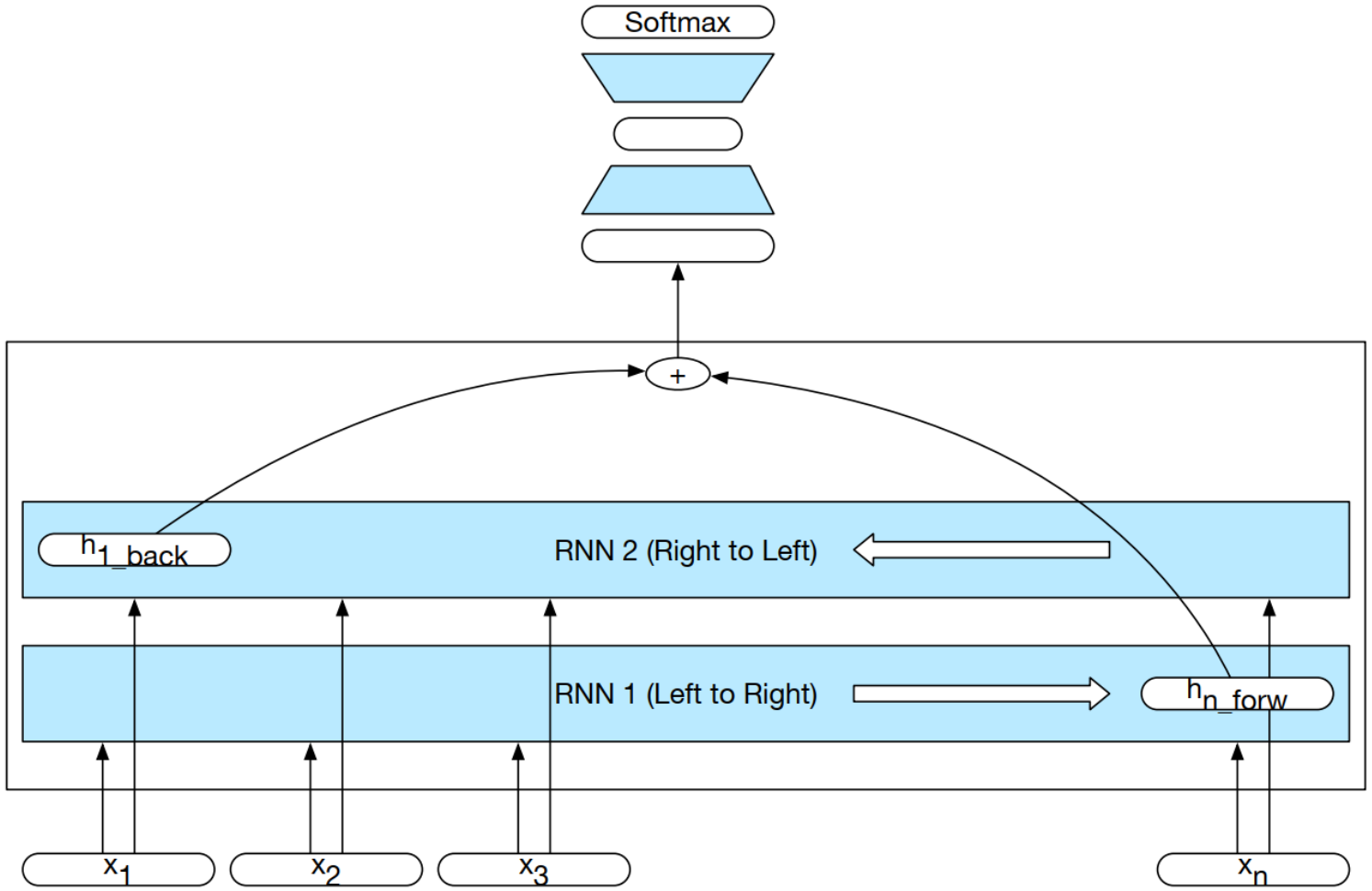
- Bidirectional RNNs based on idea that output at time t may depend on previous and future elements in sequence
  - Example: predict missing word in a sequence
- Bidirectional RNNs are two RNNs stacked on top of each other
- Output is computed based on hidden state of both RNNs



Bidirectional RNNs



Text Classification with Bidirectional RNNs



Outline

- Sequence Modeling Tasks
- Hidden Markov Model
- Recurrent Neural Networks
- RNN Applications
- Stacked and Bidirectional RNNs
- **LSTM and GRU**

Managing Context

- In practice, it is quite difficult to train RNNs for tasks that require a network to make use of information distant from the current point of processing.
- Despite having access to the entire preceding sequence, the information encoded in hidden states tends to be fairly local, more relevant to the most recent parts of the input sequence and recent decisions.
- It is often the case, however, that distant information is critical to many language applications.

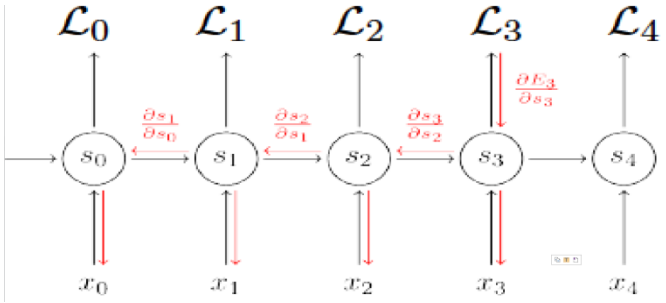
The **flights** the airline was cancelling **were** full.

Managing Context

- One reason for the inability of RNNs to carry forward critical information is that the hidden layers, and, by extension, the weights that determine the values in the hidden layer, are being asked to perform two tasks simultaneously:
  - Provide information useful for the current decision,
  - Updating and carrying forward information required for future decisions.

RNNs

- For training RNNs, calculate gradients for U, V , W



- Gradients for W:

$$\frac{\partial L_3}{\partial w} = \frac{\partial L_3}{\partial O_3}\frac{\partial O_3}{\partial S_3}\frac{\partial S_3}{\partial W} = \Sigma_{k=1}^{3}\frac{\partial L_3}{\partial O_3}\frac{\partial O_3}{\partial S_3}\frac{\partial S_3}{\partial S_k}\frac{\partial S_k}{\partial W}$$

- More generally:

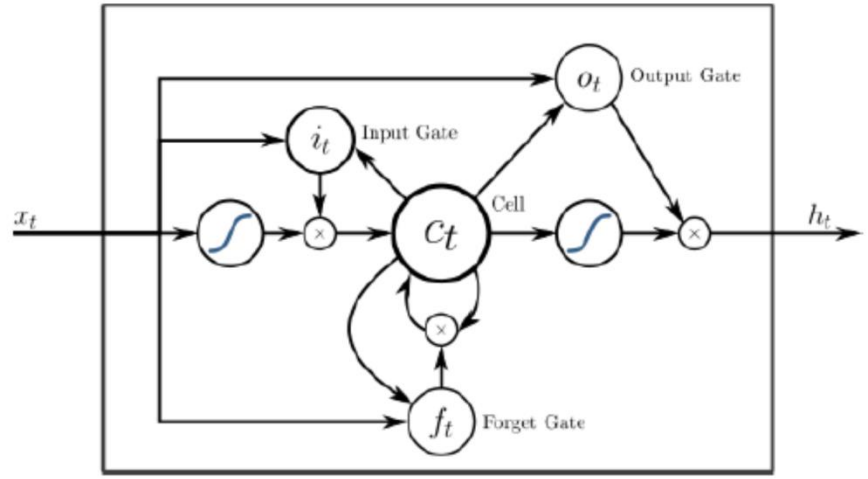
$$\frac{\partial L}{\partial S_t} = \frac{\partial L}{\partial S_m}\frac{\partial S_m}{\partial S_{m-1}}\frac{\partial S_{m-1}}{\partial S_{m-2}}\cdots\frac{\partial S_{t+1}}{\partial S_t} \ll 1$$

Vanishing and Exploding Gradients

- Backpropagation:
  - Moving backward in the network
  - Calculating gradients of loss(Error) with respect to the weights
- The gradients tends to get smaller and smaller as we keep on moving backward in the network
  - The neurons in the earlier layers learn very slowly as compared to the neurons in the later layers in the hierarchy.
  - Gradient contributions from far away steps become zero: state at those steps does not contribute to what you are learning

Long Short Term Memory (LSTM)

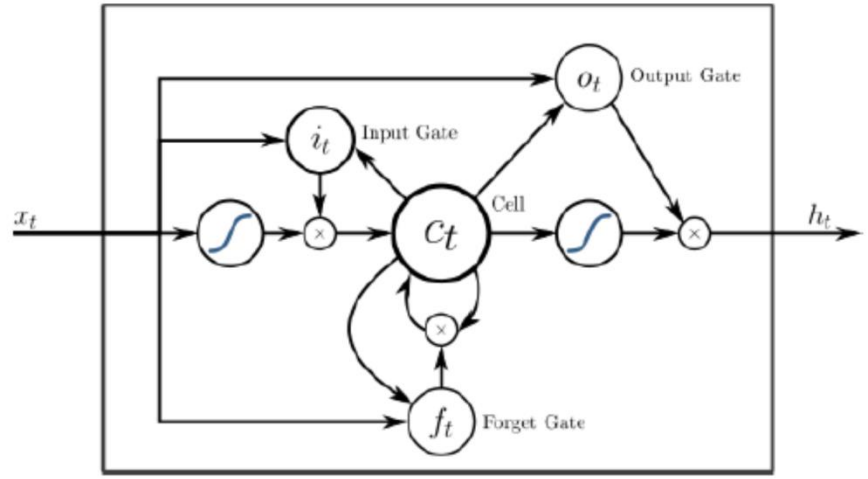
- LSTMs designed to combat vanishing gradients through gating mechanism
- The LSTM works via three gates:
  - Input: regulates how much of the new cell state to keep
  - Forget: regulates how much of the existing memory to forget
  - Output: regulates how much of the cell state should be exposed to the next layers of the network



Long Short Term Memory (LSTM)

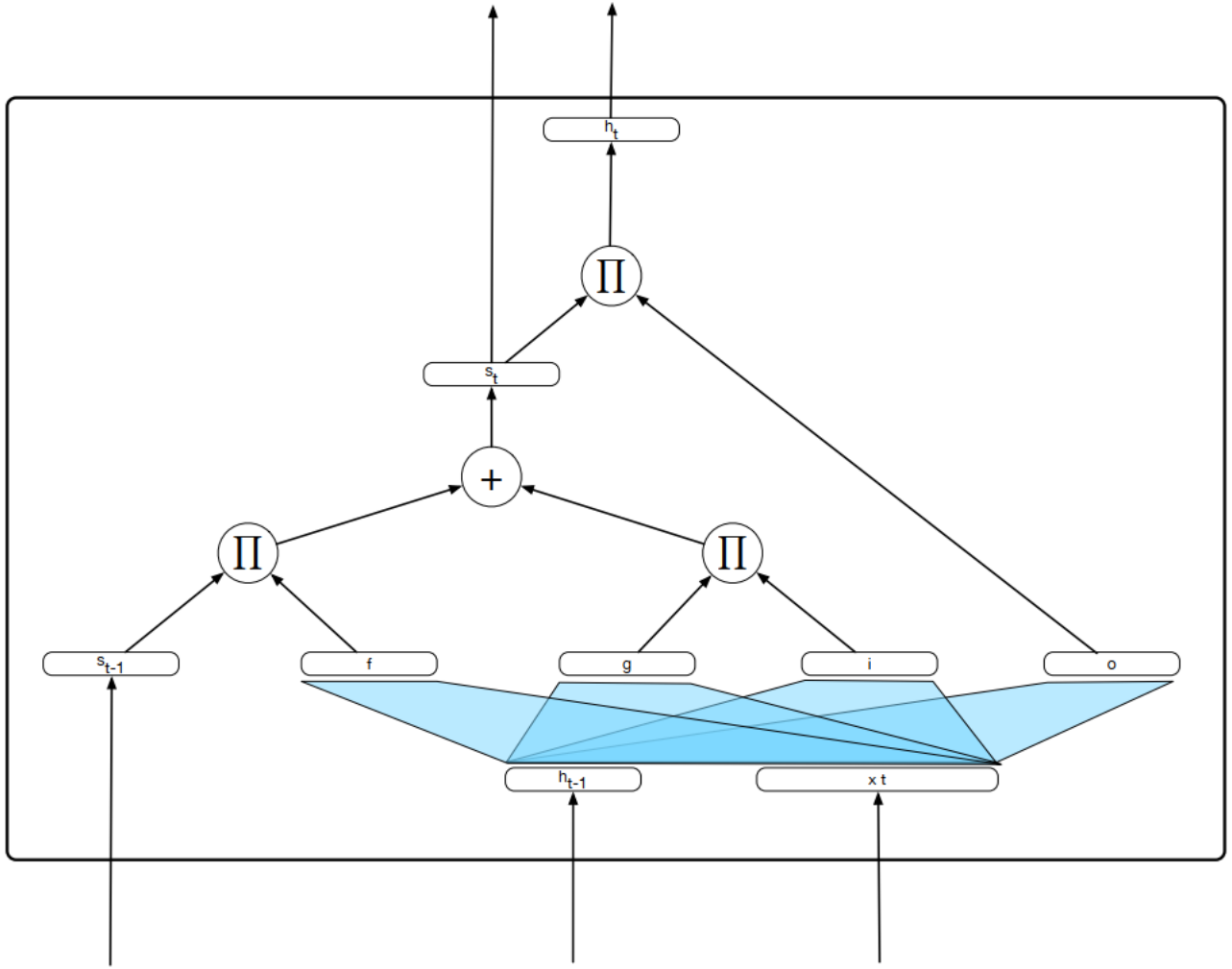
- LSTMs designed to combat vanishing gradients through gating mechanism

$$
\begin{aligned}
i_t &= \sigma(x_t\, U^i + s_{t-1}\, W^i) \\
f_t &= \sigma(x_t\, U^f + s_{t-1}\, W^f) \\
o_t &= \sigma(x_t\, U^o + s_{t-1}\, W^o) \\
g_t &= \tanh(x_t\, U^g + s_{t-1}\, W^g) \\
c_t &= c_{t-1} \circ f + g \circ ig_t \\
s_t &= \tanh(c_t) \circ o
\end{aligned}
$$



o is elementwise multiplication σ is hard sigmoid

Long Short Term Memory (LSTM)



Gated Recurrent Units (GRU)

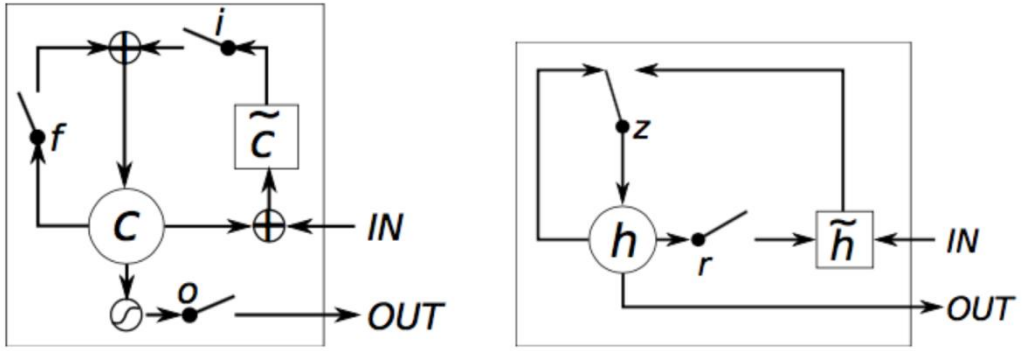
- GRU layer quite similar to that of LSTM layer, as are the equations:

$$z = \sigma(x_t\, U^z + s_{t-1}\, W^z)$$
$$r = \sigma(x_t\, U^r + s_{t-1}\, W^r)$$
$$h = \tanh(x_t\, U^h + (s_{t-1} \circ r)\, W^h)$$
$$s_t = (1 - z) \circ h + z \circ s_{t-1}$$

- GRU has two gates:
  - Reset (r): determines how to combine new input with previous memory
  - Update (z): defines how much of the previous memory to keep around
- Set reset to all 1's and update gate to all 0's to get plain RNN model
- On many tasks, LSTMs and GRUs perform similarty

LSTM vs. GRU

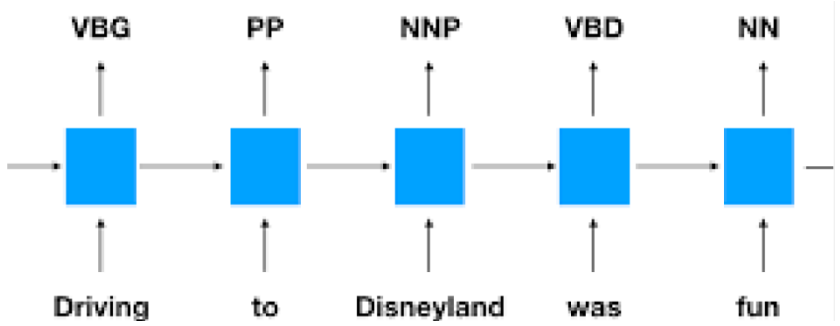
- Both LSTMs and GRUs have the same goal of tracking long-term dependencies effectively while mitigating the vanishing/exploding gradient problems
- Both LSTMs and GRUs have the ability to keep memory/state from previous activations rather than replacing the entire activation like a vanilla RNN
  - Allowing them to remember features for a long time
  - Allowing backpropagation to happen through multiple bounded nonlinearities, which reduces the likelihood of the vanishing gradient.

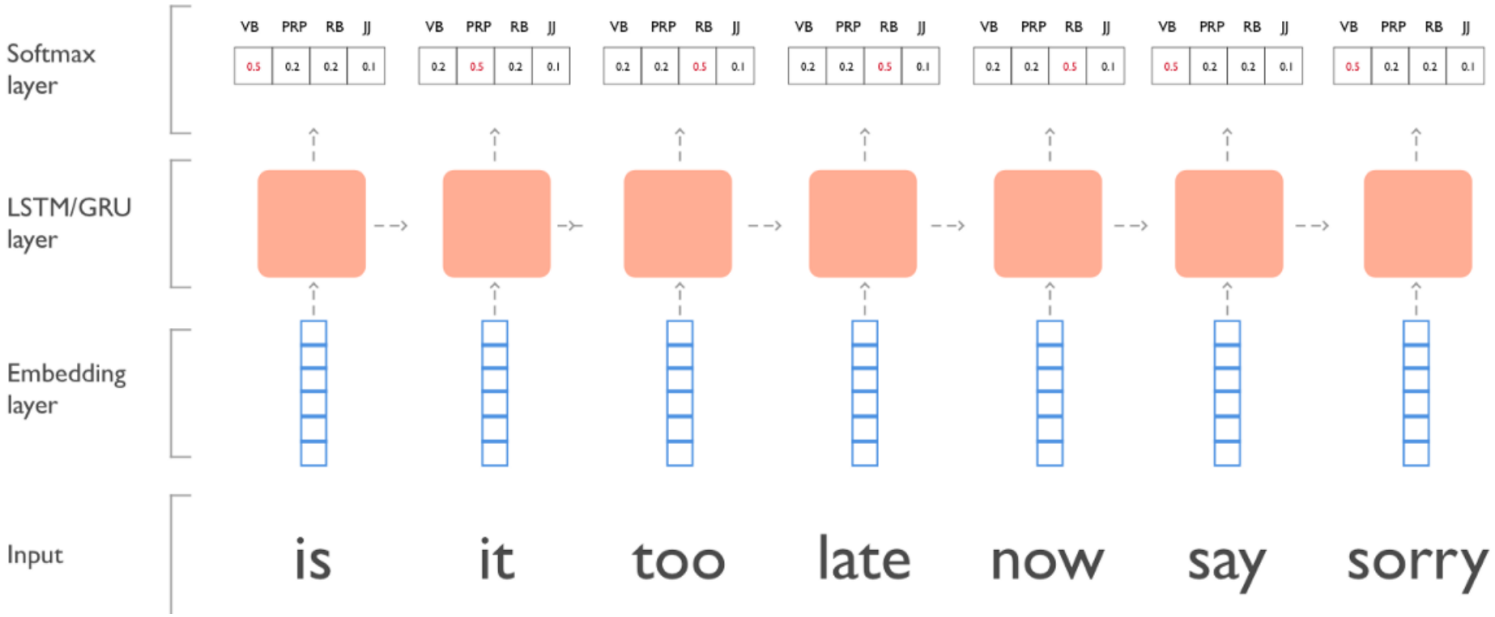


(a) Long Short-Term Memory (b) Gated Recurrent Unit

POS Tagging with RNNs

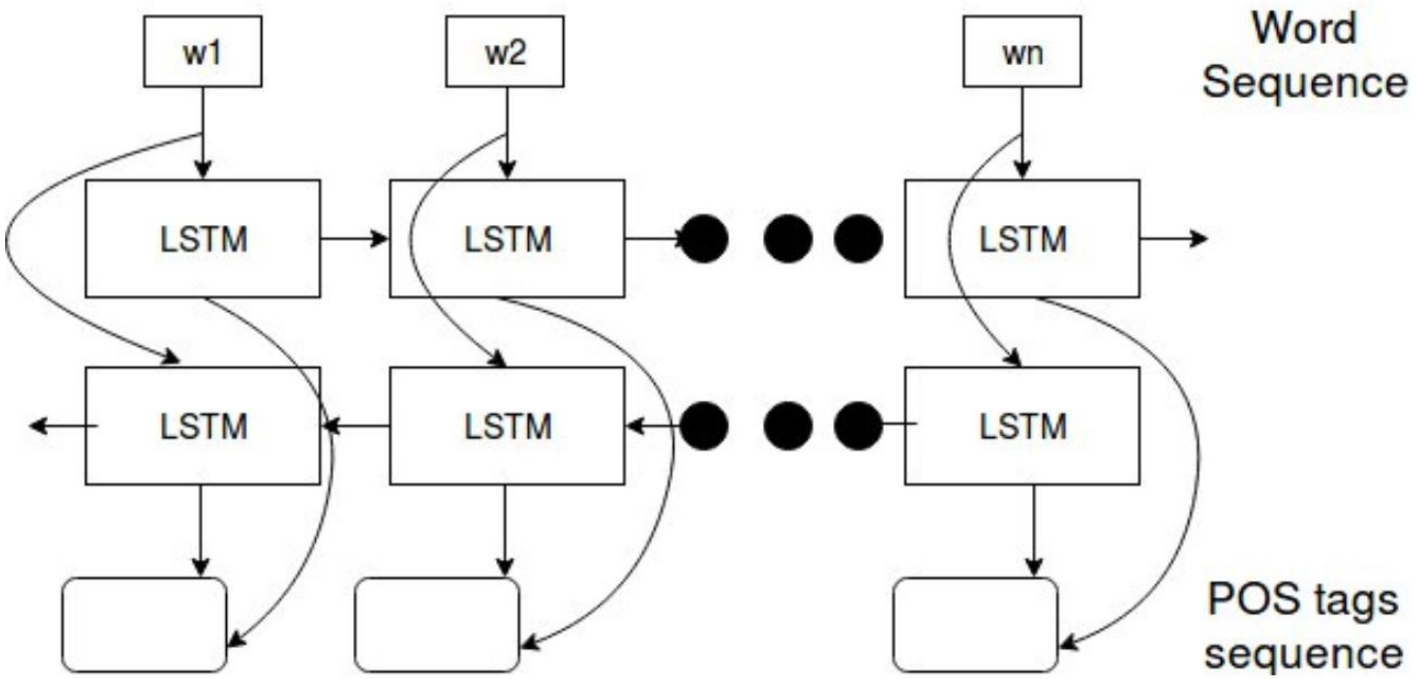
- Words as input of each cell
- POS labels as output of each cell
- Using LSTM or GRU as recurrent cells





POS Tagging with BiLSTM

- BiLSTM



Further Reading

- Speech and Language Processing (3rd ed. draft)
  - Chapter 9