Indian Institute of Technology Kharagpur
Department of Computer Science & Engineering

CS60075
Natural Language Processing
Autumn 2020

Module 6A

Sequence processing with Recurrent Networks

Sequences in NLP

Language is inherently temporal.

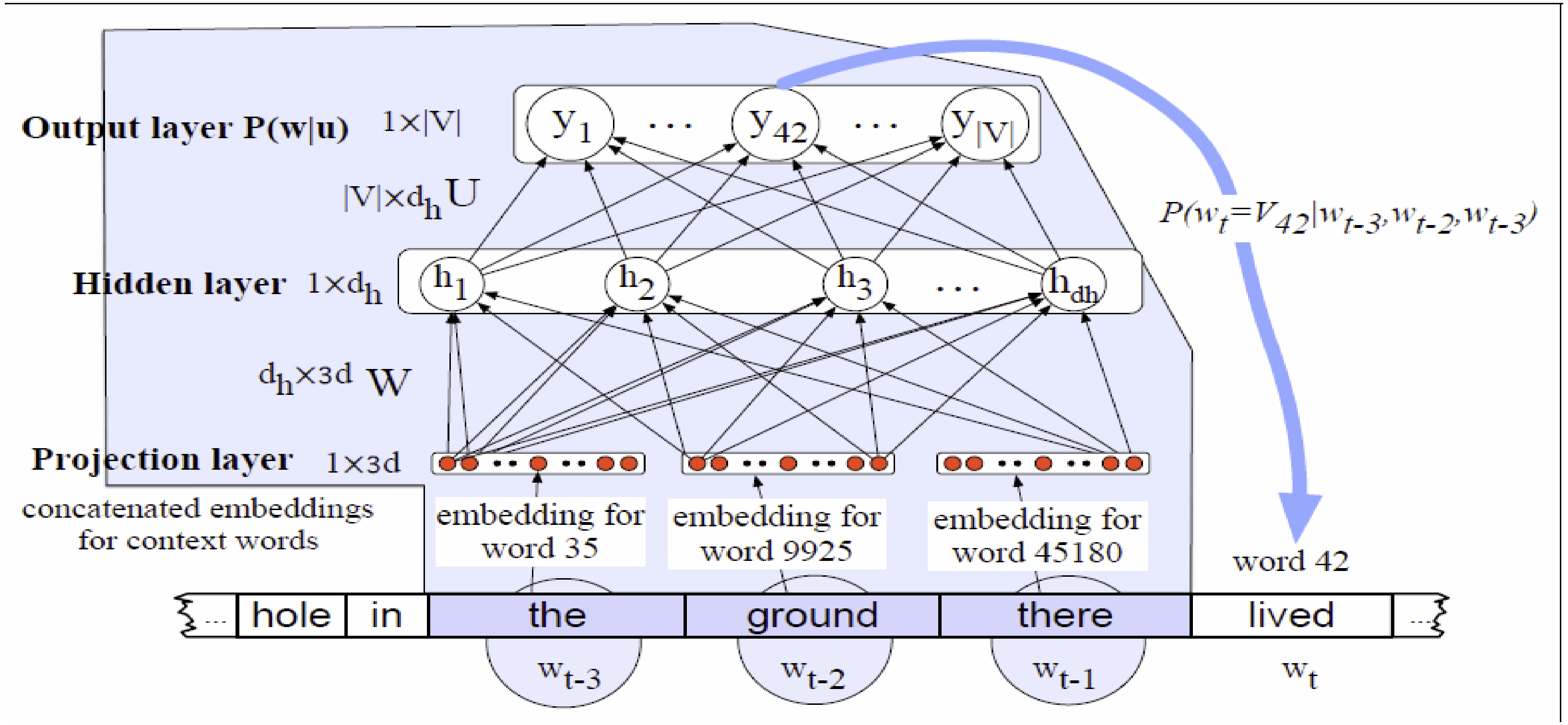
- I hope there are no more classes
- no I hope there are more classes

Applications of sequence processing

- Syntactic parsing
- Part of speech tagging
- Viterbi algorithm

3-gram Neural Language Model

Assume Pre-trained embeddings



$$L = -\log P(w_t \mid w_{t-1})$$

- Limits the context from which
- information can be extracted
- Makes it difficult to learn systematic patterns

Sequence Processing with Recurrent Networks

Address limitation of sliding window approach

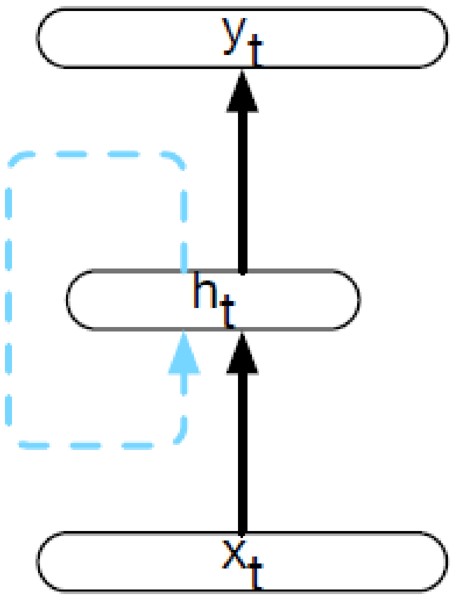
Handle variable length input

Networks that contain cycles within their connections

The value of a unit is dependent on outputs from previous time steps as input

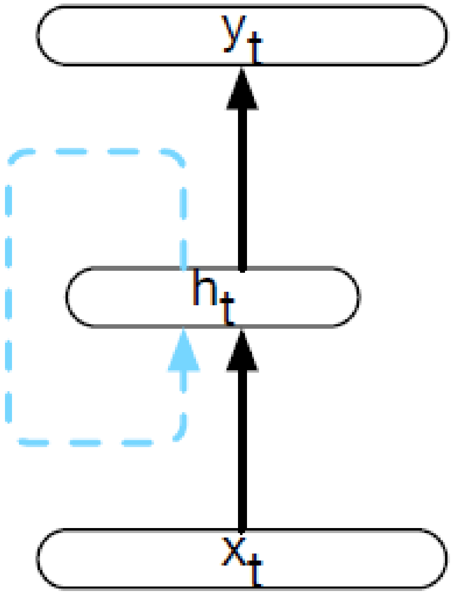
Many different variations among RNNs

Long short-term memory network (LSTM)	Bidirectional LSTM (BiLSTM)	Gated recurrent unit (GRU)



Decoding for sequence labeling

- HMM
- Linear-Chain Conditional Random Fields:
  - Viterbi algorithm for inference
- Neural model (Recurrent NN) (Elman, 1990) –
- What are the challenge?
  - input and output of the network does not have a fixed length
  - **Solution**: activation value of the hidden layer $\mathbf{h_t}$ depends on the current input $\mathbf{x_t}$ as well as the activation value of the hidden layer from the previous time step $\mathbf{h_{t-1}}$.



Simple recurrent neural network illustrated as a feed-forward network

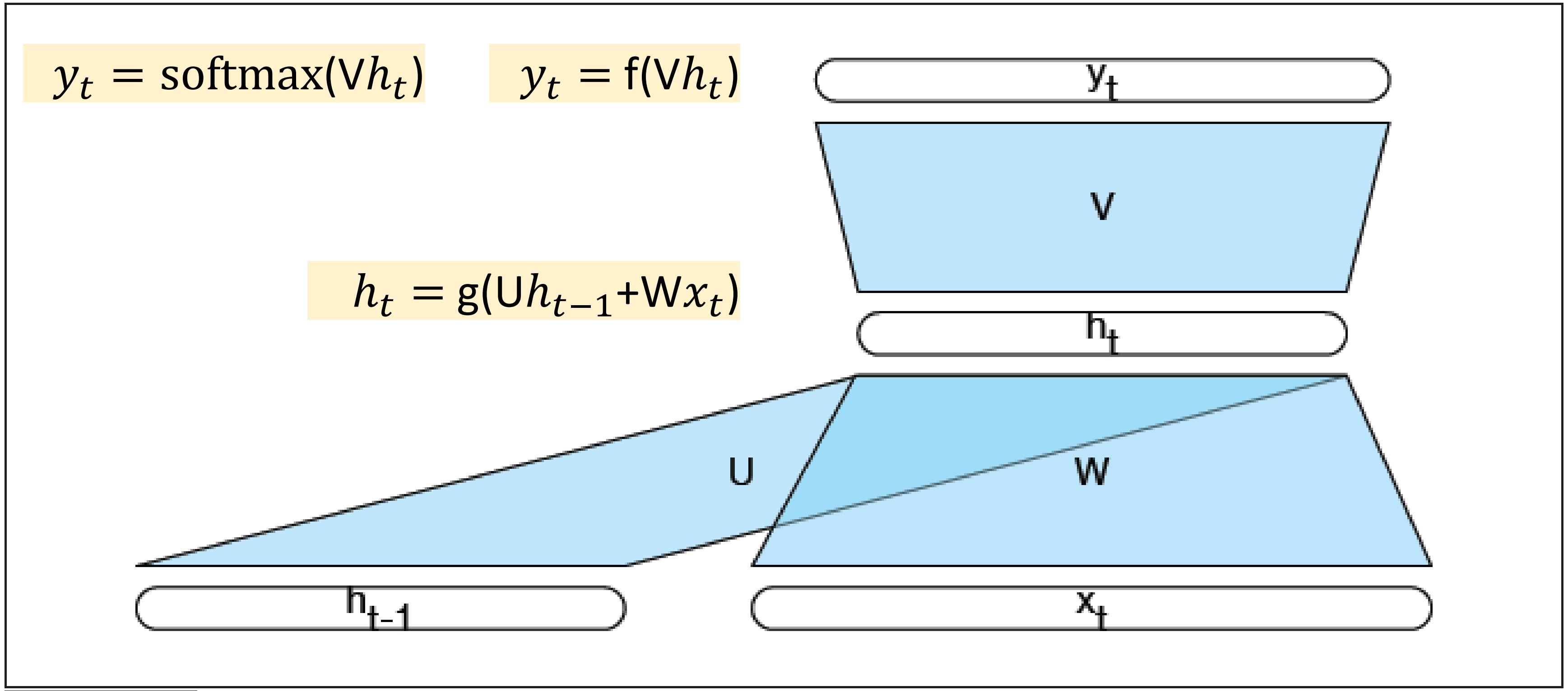


Figure 9.3 Simple recurrent neural network illustrated as a feed-forward network.

RNN unrolled in time + Inference

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

Figure 9.5 Forward inference in a simple recurrent network.

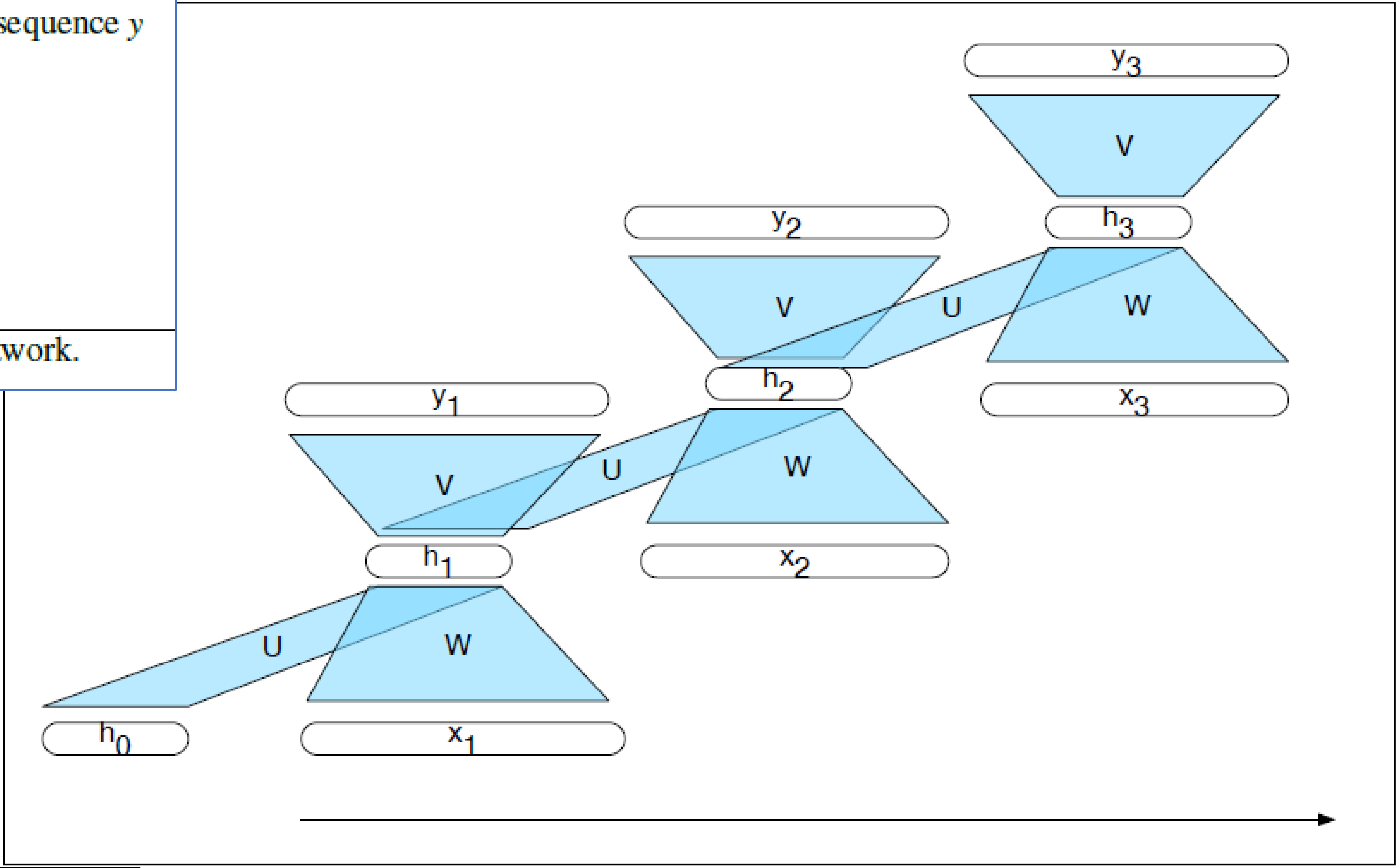


Figure 9.4 A simple recurrent neural network shown unrolled in time. Network layers are copied for each timestep, while the weights U, V and W are shared in common across all timesteps.

Training an RNN

- Loss Function
- Backpropagation *through time*

Two-pass algorithm for training RNNs

First pass: **Perform forward inference**

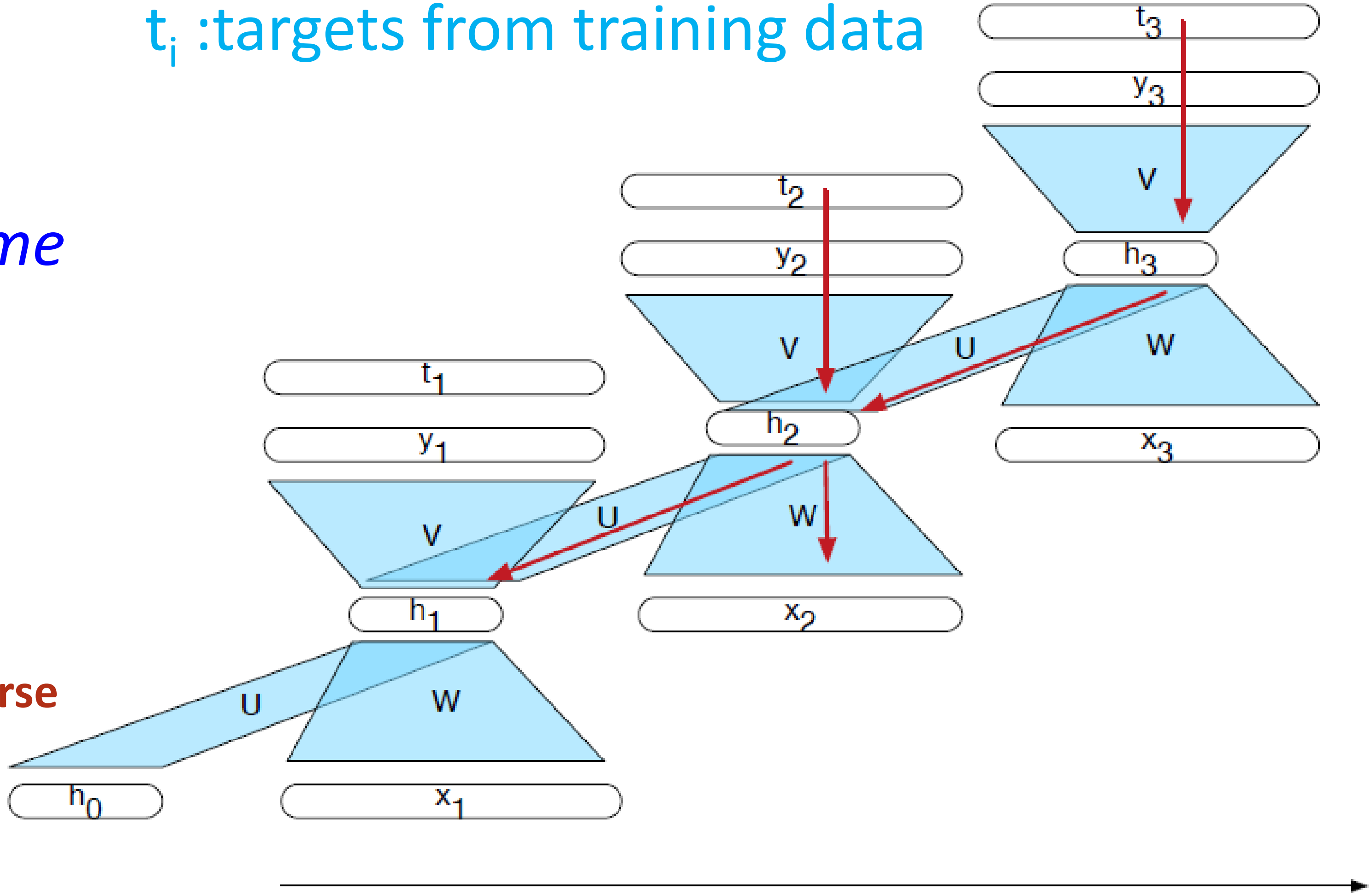
Compute h_t and y_t at each step in time

Compute the loss at each step in time

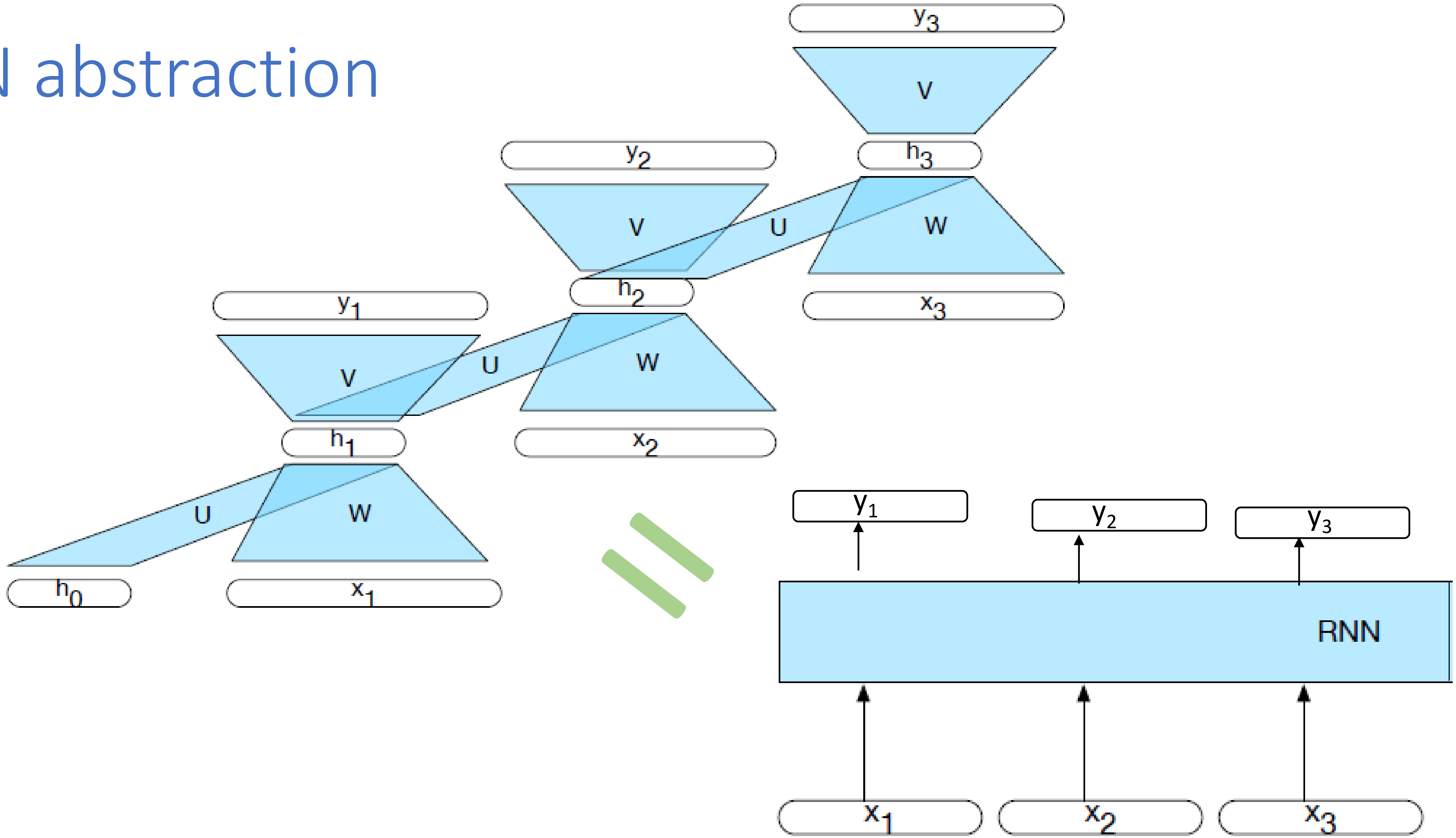
Second pass: **Process the sequence in reverse**

Compute the required error gradients at each step backward in time

t_i :targets from training data



RNN abstraction

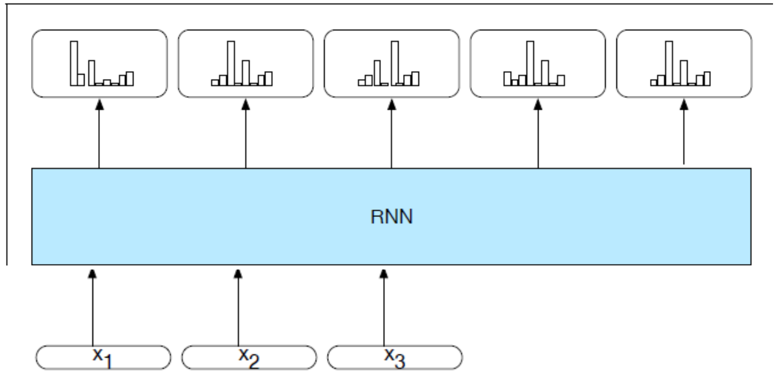


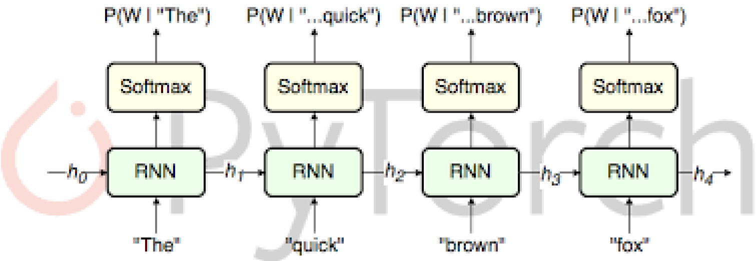
Truncated Backpropagation training through time

- For applications that involve much longer input sequences, such as speech recognition, character-by- character sentence processing, or streaming of continuous inputs, unrolling an entire input sequence may not be feasible.
- In these cases, we can unroll the input into manageable fixed-length segments and treat each segment as a distinct training item. This approach is called Truncated Backpropagation Through Time (TBTT).

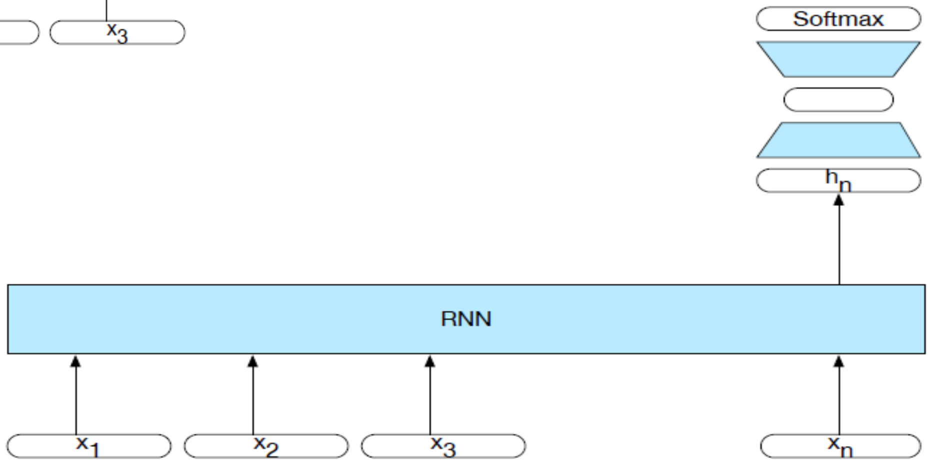
RNN Applications

- Language Modeling

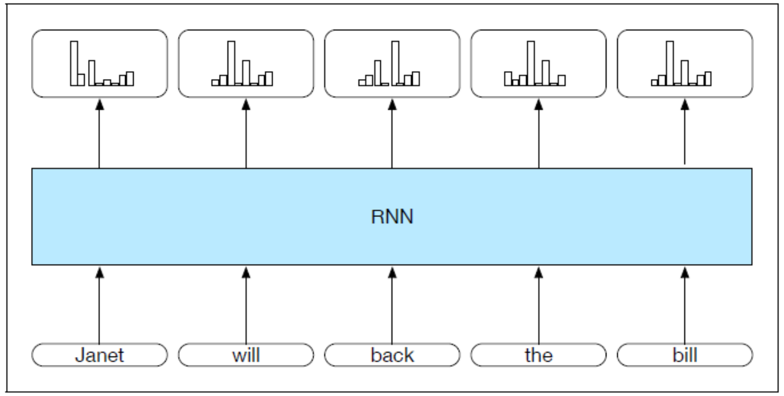




- Sequence Classification (Sentiment, Topic)



- Sequence to Sequence

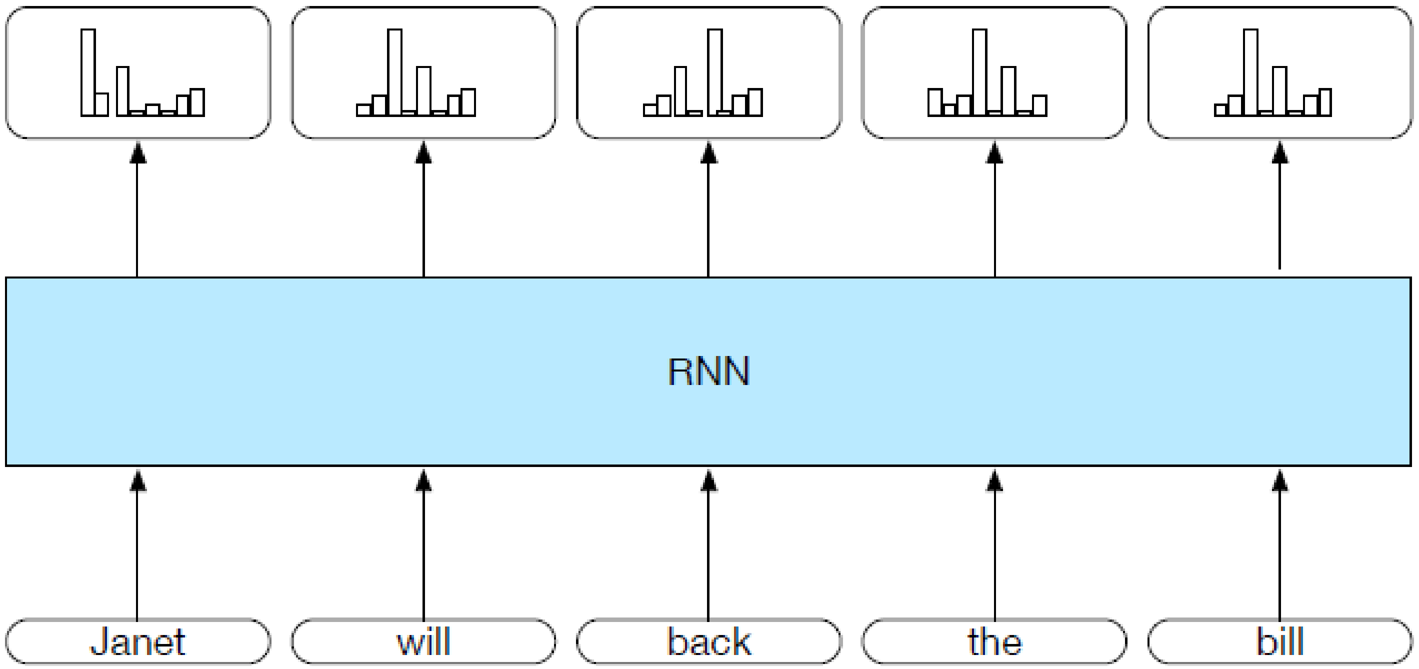


RNN Applications: Sequence Labeling (e.g., POS)

- **Input**: pre-trained embeddings
- **Output**: softmax layer provides a probability distribution over the part-of-speech tags as output at each time step

- Choosing max probability label for each item does not necessarily result in optimal (or even very good) tag sequence

- Combine with Viterbi for most likely sequence



RNN Applications: sequence classification (RNN + FeedForward)

- Hidden layer from final state (compressed representation of entire sequence) ->
- Input to feed-forward trained to selects correct class

- No intermediate outputs for items in the sequence preceding x_n => no intermediate losses

- Only cross-entropy loss on final classification backpropagated all the way…