SFU NatLangLab

CMPT 413/713: Natural Language Processing

Recurrent Neural Networks

How to model sequences using neural networks?

Spring 2024
2024-01-31

Adapted from slides from Anoop Sarkar, Danqi Chen, Karthik Narasimhan, and Justin Johnson

(Some slides adapted from Chris Manning, Abigail See, Andrej Karpathy)

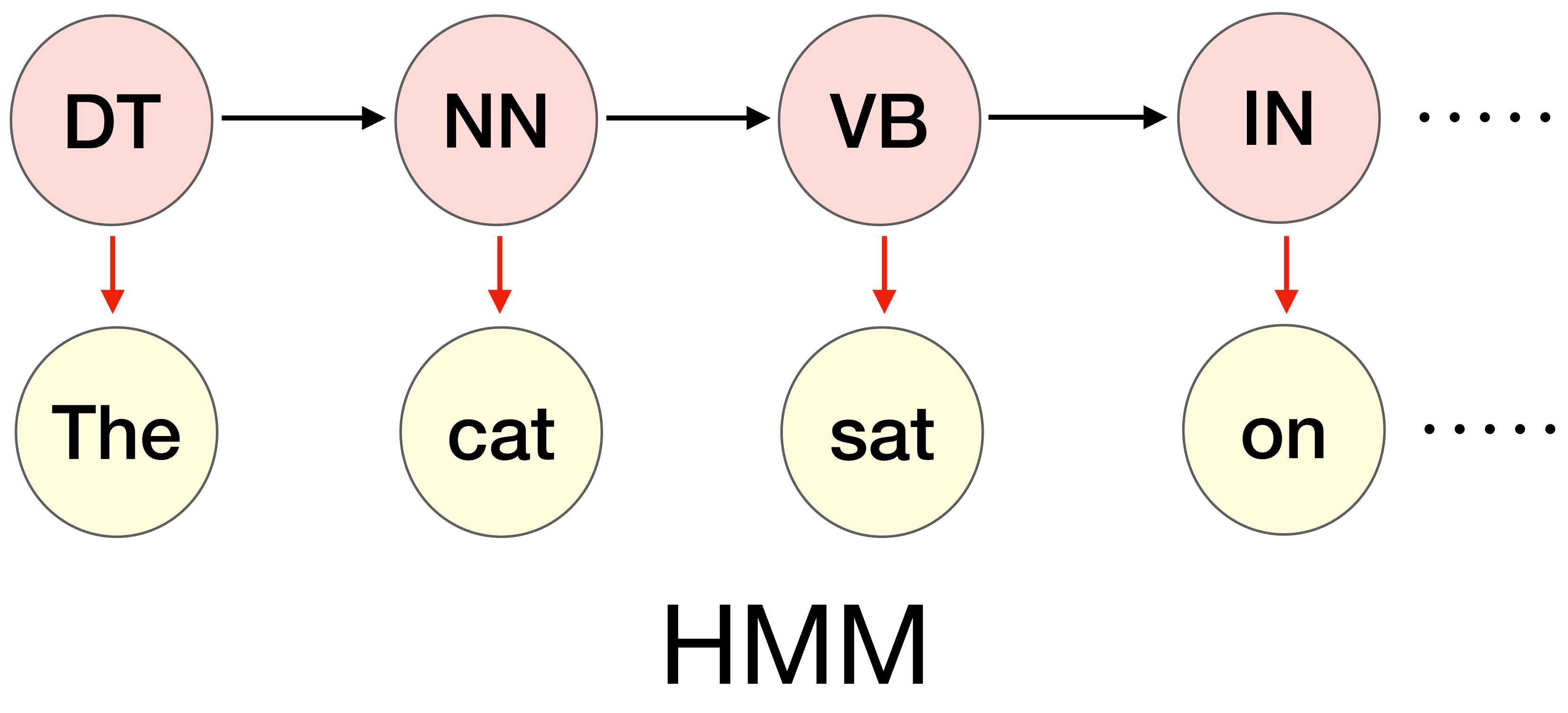
Overview

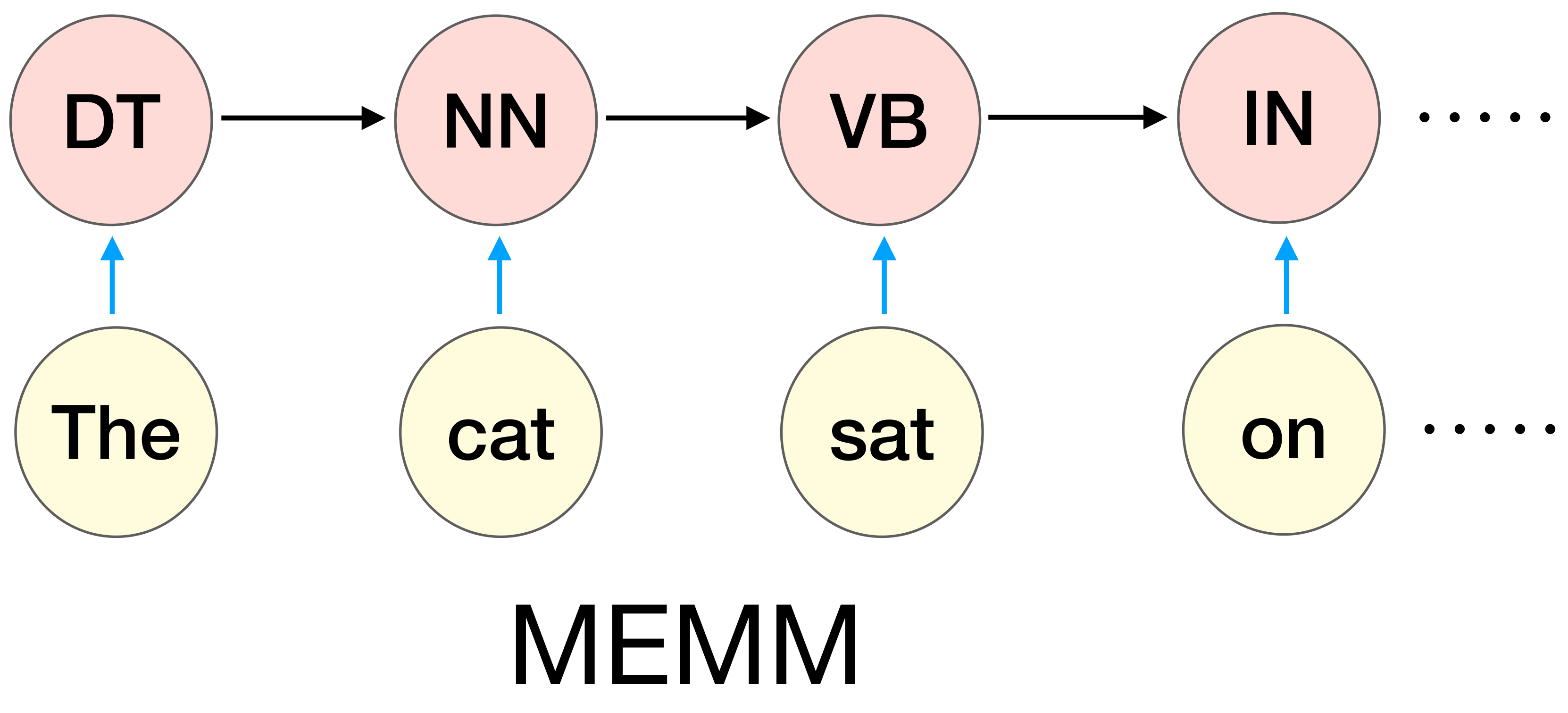
- **What is a recurrent neural network (RNN)?**
- **Simple RNNs**
- Backpropagation through time
- Applications
- Variants: Stacked RNNs, Bidirectional RNNs

What are recurrent neural networks?

Recurrent neural networks (RNNs)

How can we model sequences using neural networks?



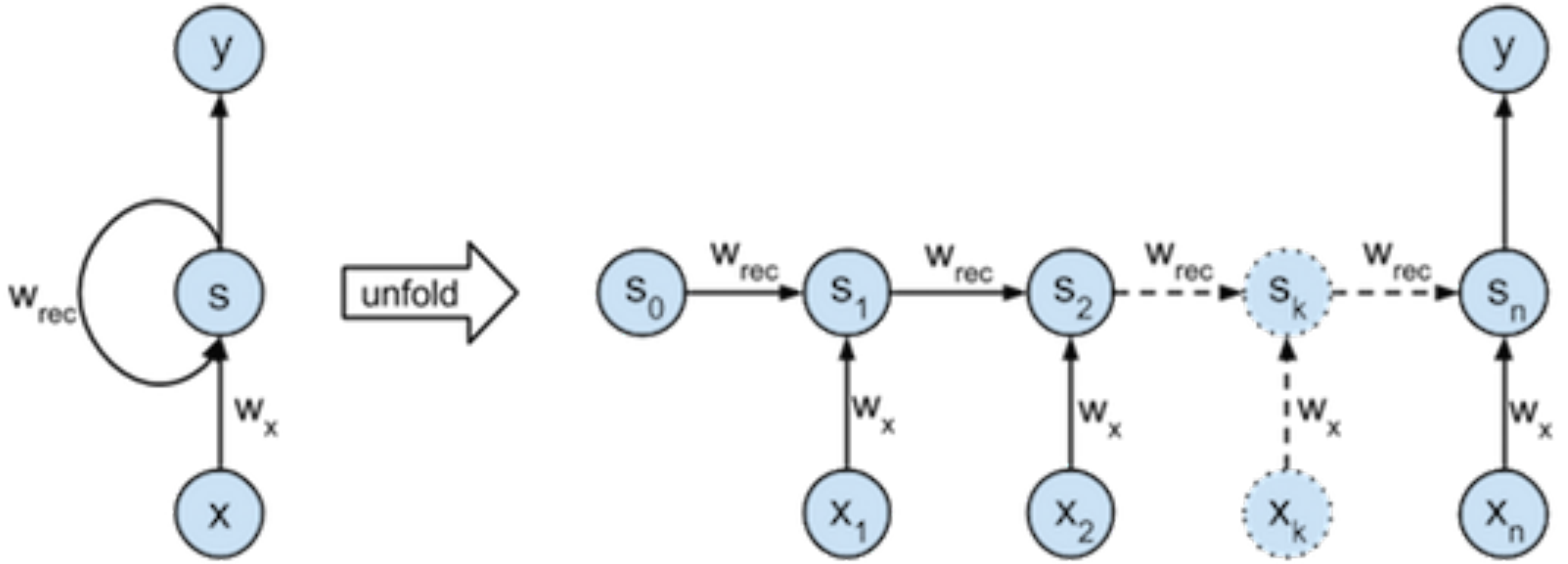


- Recurrent neural networks = A class of neural networks used to model sequences, allows for handling of **variable length inputs**.

- Very crucial in NLP problems (different from images) because sentences/paragraphs are variable-length, sequential inputs.

Recurrent neural networks (RNNs)

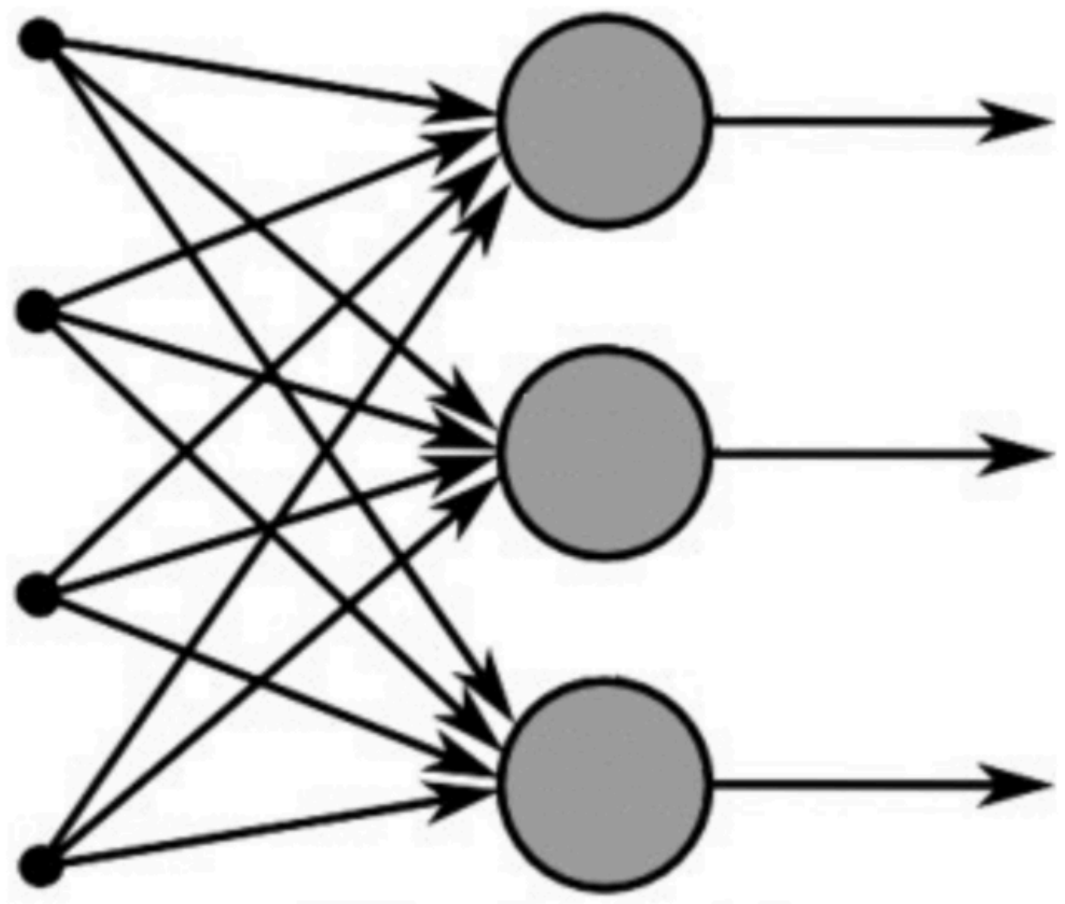
A class of neural networks allowing to handle **variable length inputs**



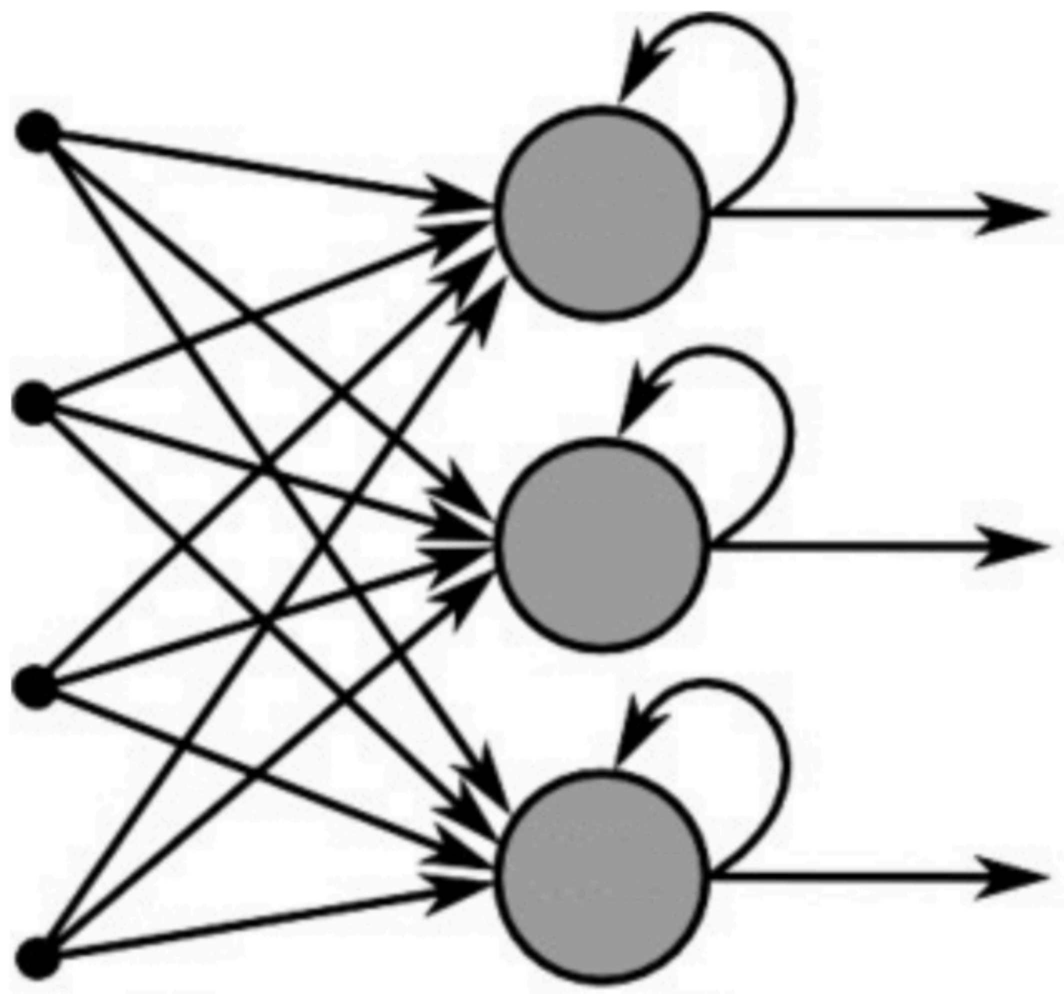
A function: $y = \text{RNN}(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n) \in \mathbb{R}^d$ where $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^{d_{in}}$

Core idea: apply the same weights repeatedly at different positions

RNNs vs Feedforward NNs



Feed-Forward Neural Network

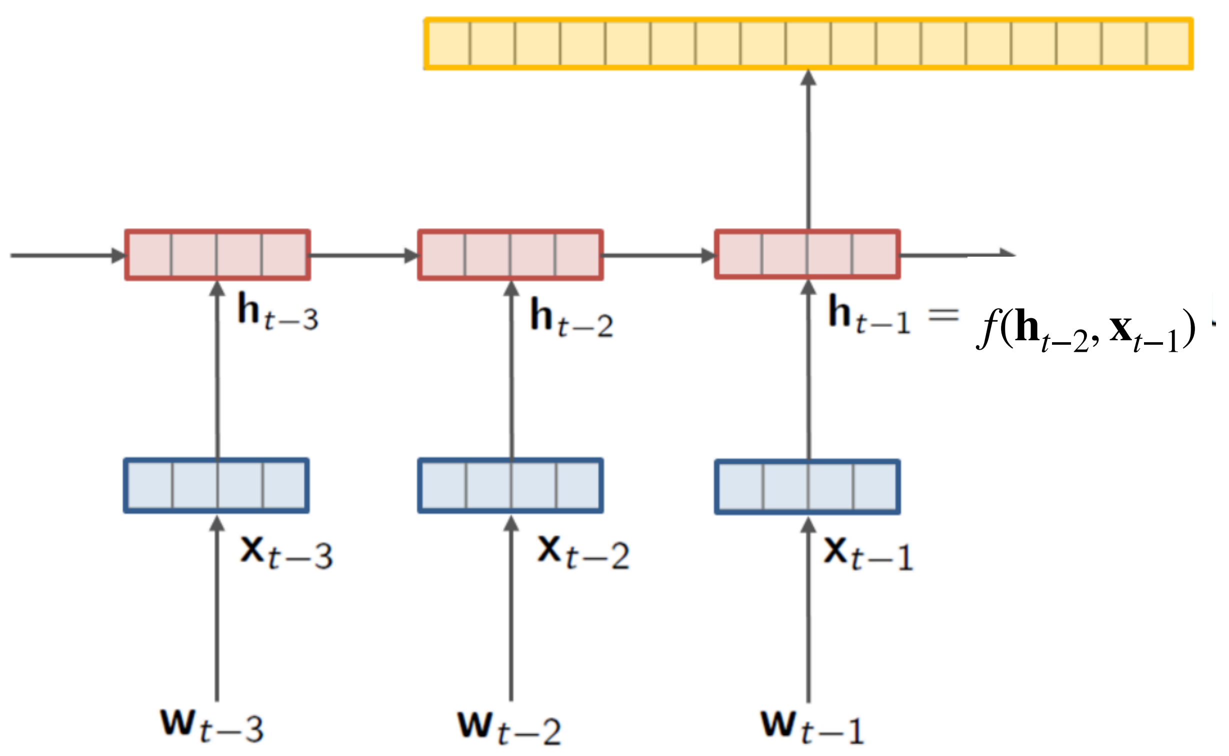


Recurrent Neural Network

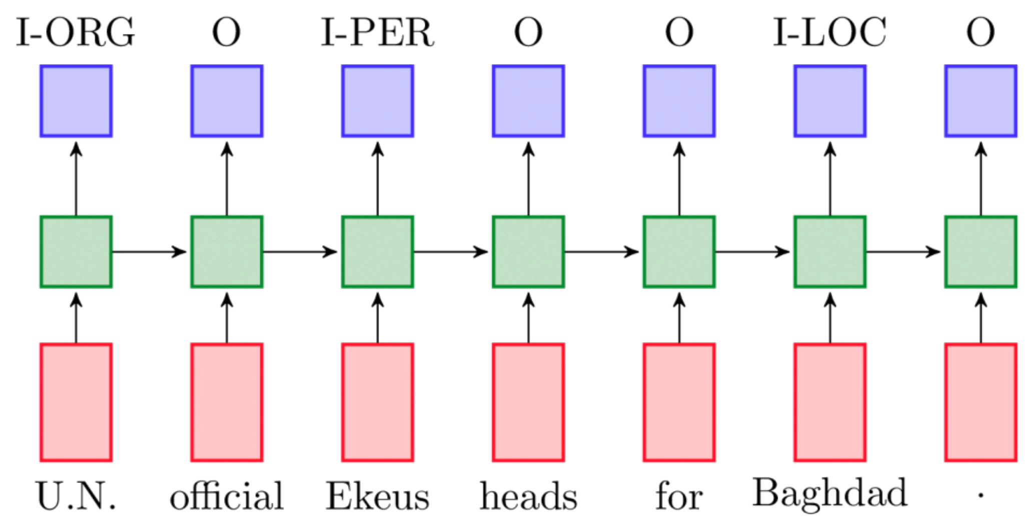
Recurrent neural networks (RNNs)

Proven to be an highly effective approach to **language modeling**, **sequence tagging** as well as **text classification** tasks:

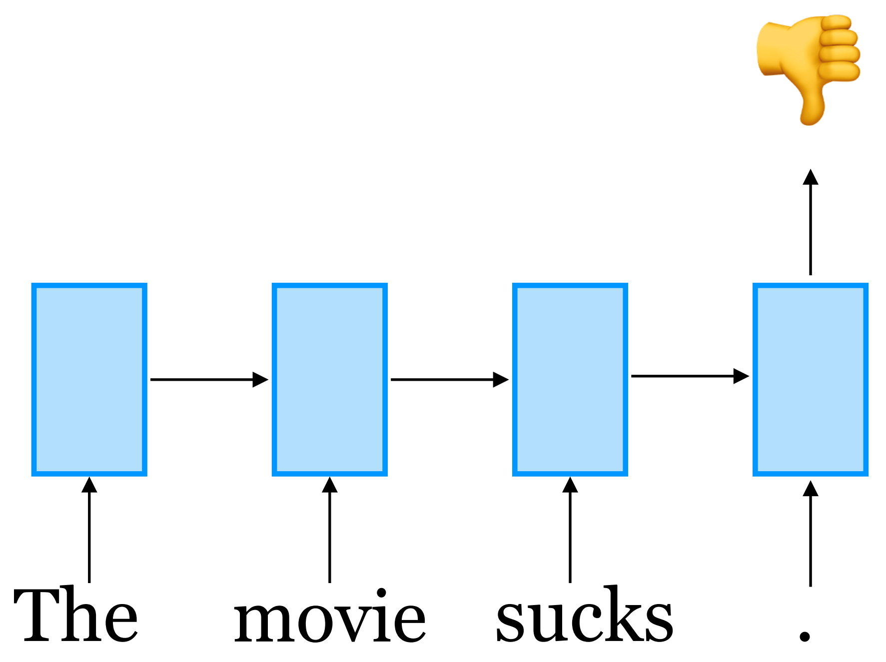
Language modeling



Sequence tagging



Text classification



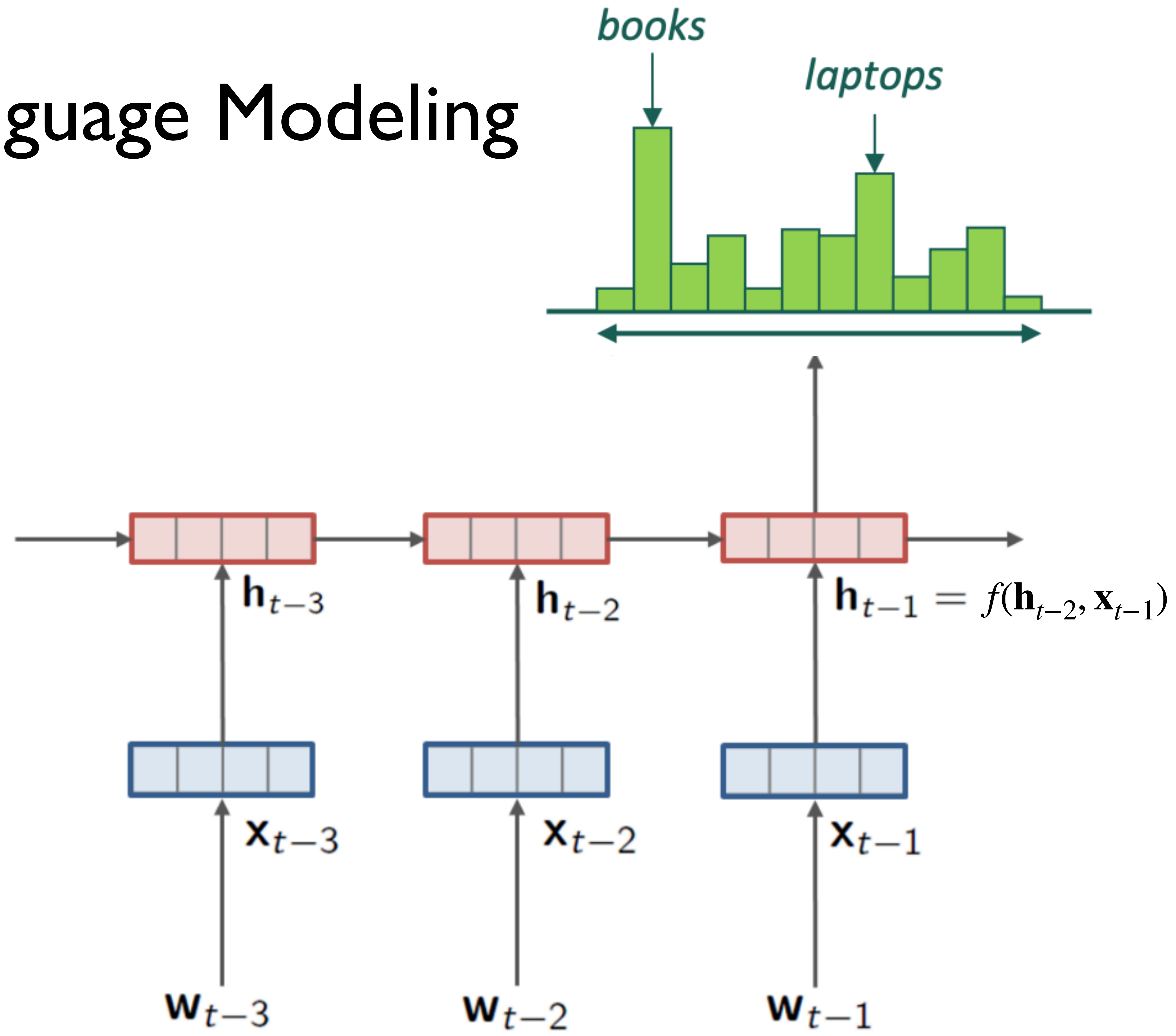
Form the basis for the modern approaches to **machine translation**, **question answering** and **dialogue**:

RNNs for Language Modeling

Use a RNN to

- capture the history of the previous words as a hidden state
- use hidden state to estimate the probability of the next word

$$P(w_t \mid w_1, \ldots, w_{t-1}) = P(w_t \mid h_{t-1})$$



Recall: Language Models

- Model the probability of a sequence of words

$$P(w_1, w_2, \ldots, w_n) = \prod_{i=1}^{n} P(w_i | w_1, \ldots, w_{i-1})$$

- kth order Markov assumption

$$P(w_1 w_2 \ldots w_n) \approx \prod_i P(w_i \mid w_{i-k} \ldots w_{i-1})$$

- N-grams

Bigram (1st order)

$$P(\text{mat}|\text{the cat sat on the}) \approx P(\text{mat}|\text{the})$$

Trigram (2nd order)

$$P(\text{mat}|\text{the cat sat on the}) \approx P(\text{mat}|\text{on the})$$

Issues with traditional n-grams

Consider:

As the proctor started the clock, the students opened their ____________

For a 4-gram, the probability of the next word would be estimated by

~~*As the proctor started the clock, the*~~ *students opened their* ____________

$$P(w \,|\, \text{students opened their } w) = \frac{\text{count(students opened their } w)}{\text{count(students opened their)}}$$

Small n: not enough context for long range dependencies

Issues with traditional n-grams

- Example generation from trigram model

  today the price of gold per ton , while production of shoe lasts and shoe industry , the bank intervened just after it considered and rejected an imf demand to rebuild depleted european stocks , sept 30 end primary 76 cts a share .

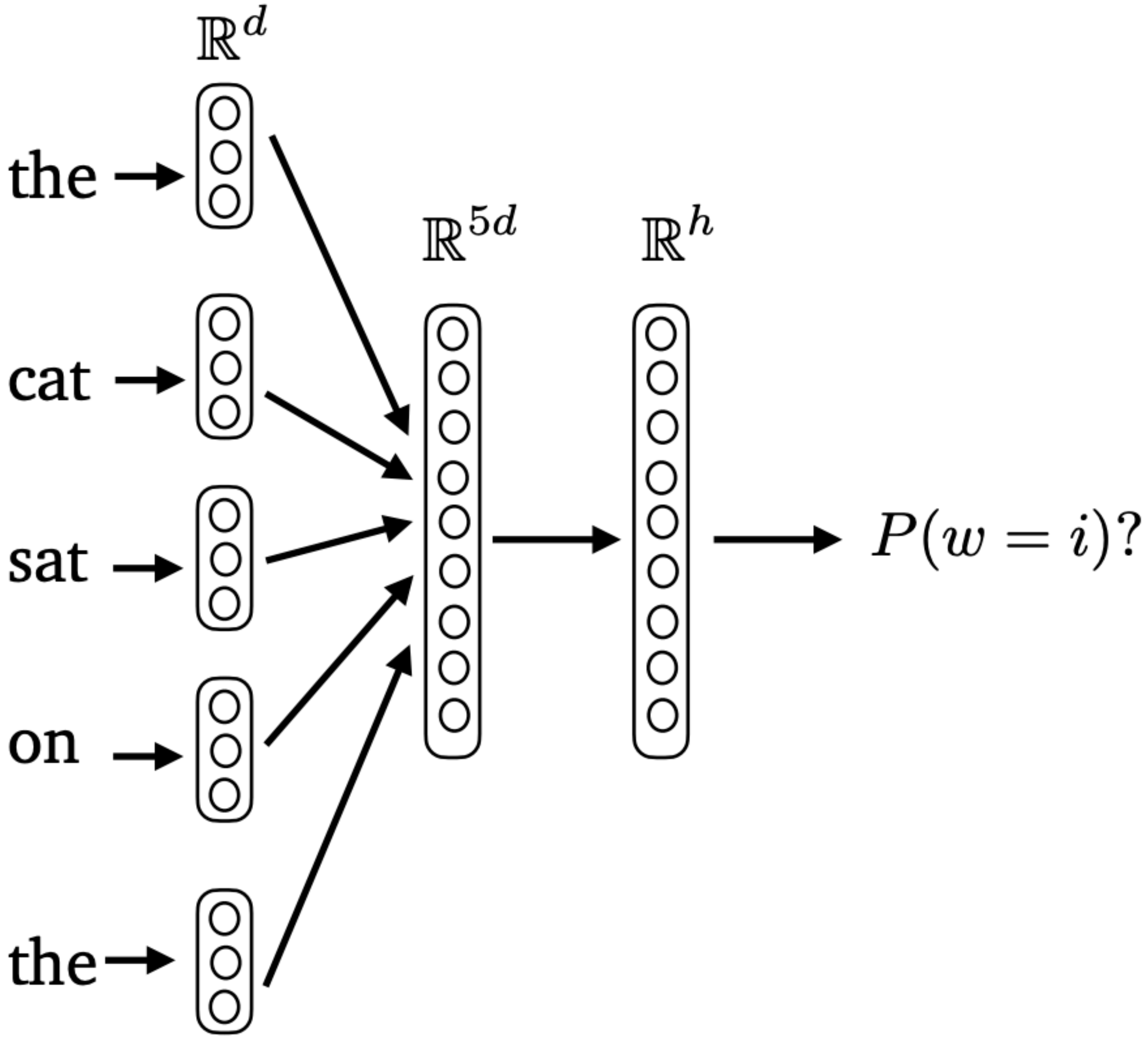
- Surprisingly grammatical!
- But incoherent. Need to consider longer history to model language well.

Why can't we just increase *n*?

- Sparsity issues when n is large
- Model size (number of parameters) increases exponentially with n
  - Takes up a lot of memory to store all those probabilities

Feedforward Neural Language Model

- P(mat | the cat sat on the) = ?



- Input layer (context size n = 5):

$$\mathbf{e} = [\mathbf{e}_{\mathrm{the}}; \mathbf{e}_{\mathrm{cat}}; \mathbf{e}_{\mathrm{sat}}; \mathbf{e}_{\mathrm{on}}; \mathbf{e}_{\mathrm{the}}] \in \mathbb{R}^{dn}$$

concatenate word embeddings

- Hidden layer

$$\mathbf{h} = \tanh(\mathbf{W}\mathbf{e} + \mathbf{b}_1) \in \mathbb{R}^h$$

- Output layer (softmax)

$$\mathbf{z} = \mathbf{U}\mathbf{h} + \mathbf{b}_2 \in \mathbb{R}^{|V|}$$

$$P(w = i \mid \mathrm{context}) = softmax_i(\mathbf{z})$$

(Bengio et 2003): A Neural Probabilistic Language Model

Feedforward Neural Language Model

output distribution

$$\hat{\boldsymbol{y}} = \mathrm{softmax}(\boldsymbol{U}\boldsymbol{h} + \boldsymbol{b}_2) \in \mathbb{R}^{|V|}$$

hidden layer

$$\boldsymbol{h} = f(\boldsymbol{W}\boldsymbol{e} + \boldsymbol{b}_1) \qquad \mathbf{W} \in \mathbb{R}^{h \times nd}$$

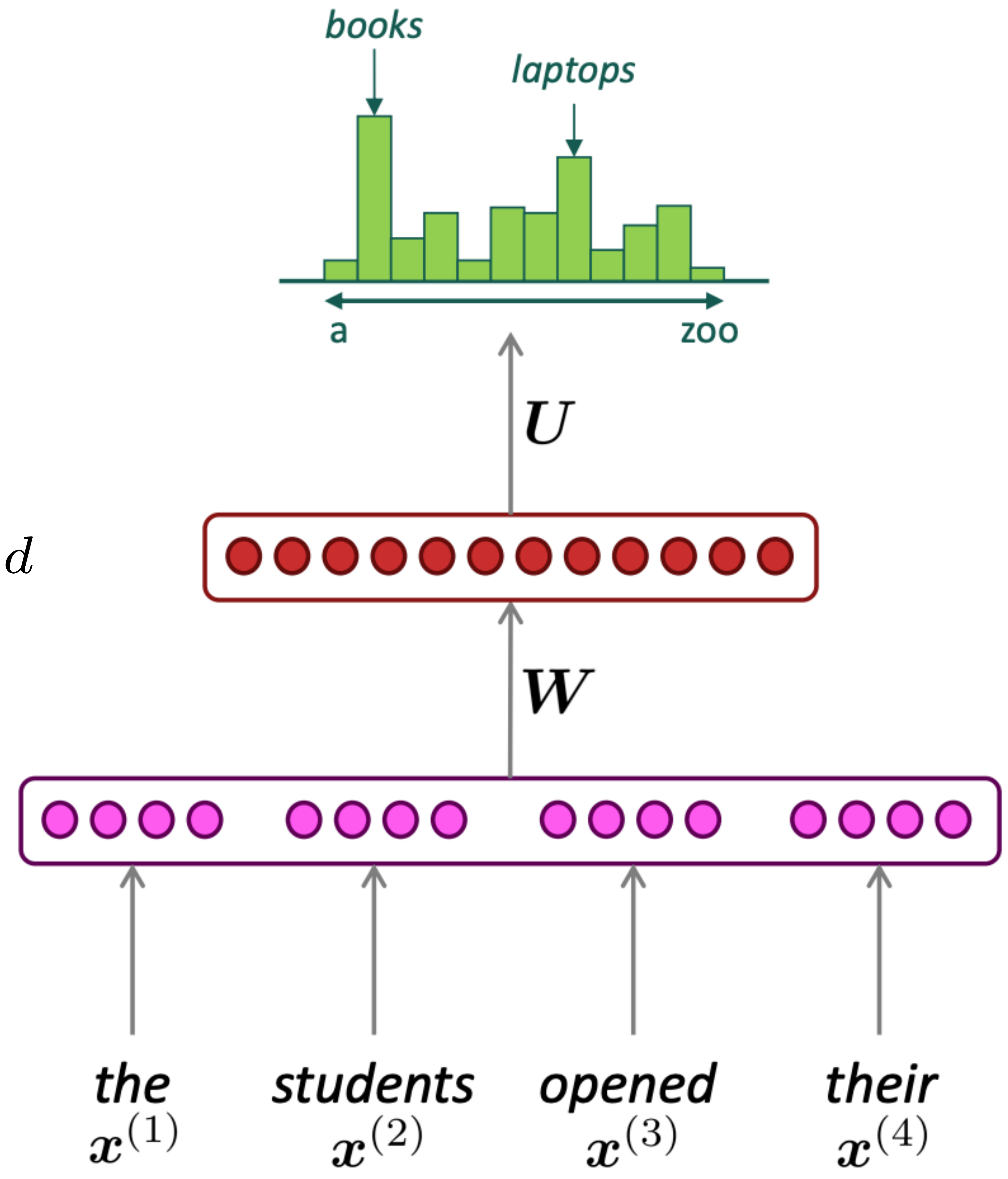
concatenated word embeddings

$$\boldsymbol{e} = [\boldsymbol{e}^{(1)}; \boldsymbol{e}^{(2)}; \boldsymbol{e}^{(3)}; \boldsymbol{e}^{(4)}]$$

Fixed window LM

words / one-hot vectors

$$\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \boldsymbol{x}^{(3)}, \boldsymbol{x}^{(4)}$$



Slide credit: Chris Manning

Issues with fixed-window neural LM

Improvements over n-gram LMs

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

Remaining issues

- Fixed window is still limited in size (too small)
- Enlarging window increases parameters: $\mathbf{W} \in \mathbb{R}^{h \times nd}$
- Each word in the window is multiplied by a different set of weights
  - No symmetry in how the inputs are processed

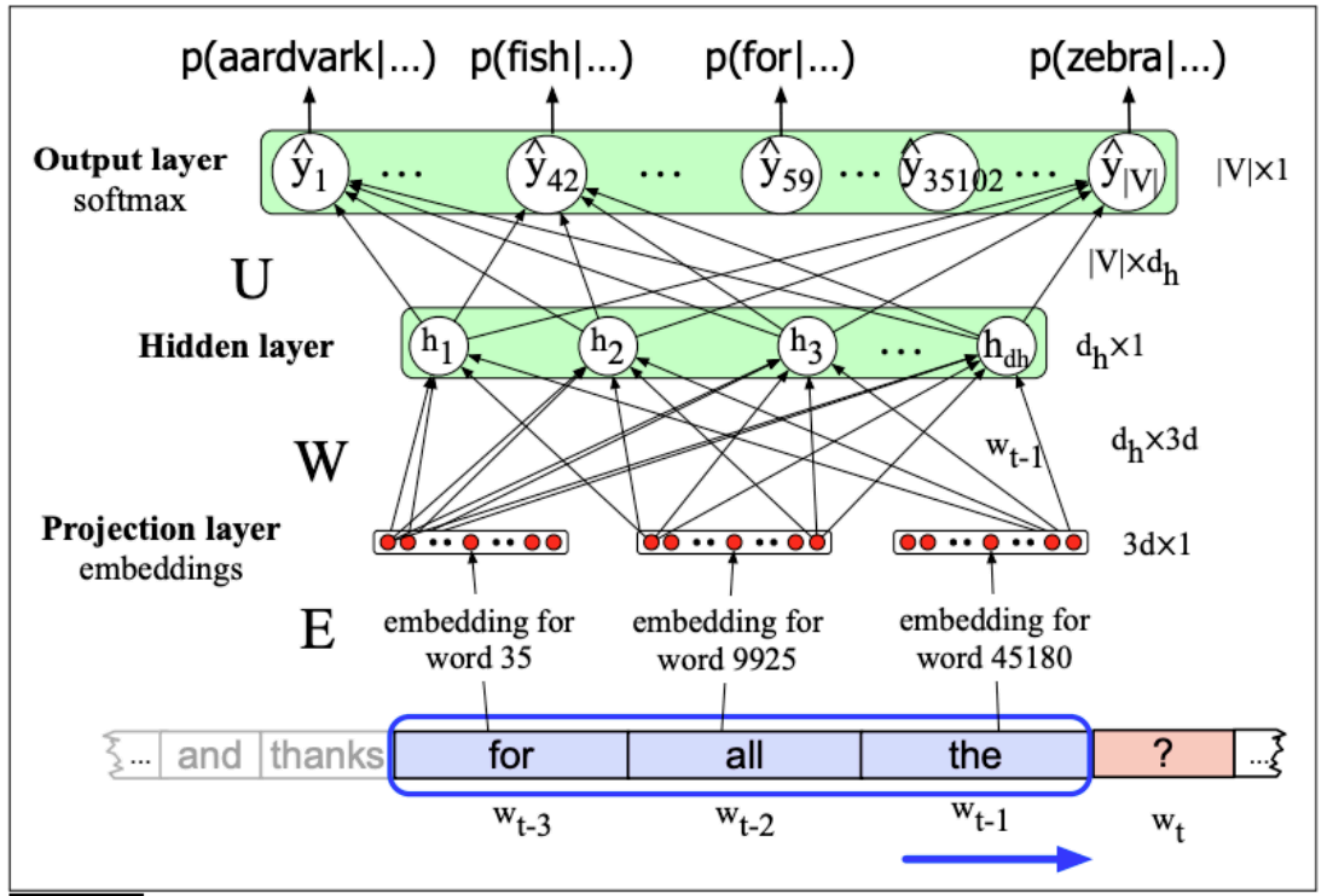


Figure 9.1 A simplified view of a feedforward neural language model moving through a text. At each time step t the network takes the 3 context words, converts each to a d-dimensional embedding, and concatenates the 3 embeddings together to get the $1 \times Nd$ unit input layer x for the network. The output of the network is a probability distribution over the vocabulary representing the models belief with respect to each word being the next possible word.

"all the" appears in different positions of two sliding windows

What we really want:

- Neural network to handle input sequences of arbitrary length!

Language Modeling

Predict probability of sequence of words

$$P(s) = P(w_1, \ldots, w_T) = \prod_{t=1}^{T} P(w_t | w_{<t})$$

with n-grams

$$P(w_t | w_{<t}) \approx P(w_t | w_{t-n+1,t-1})$$

with HMMs

$$P(w_t | w_{<t}) \approx P(w_t | h_t) P(h_t | h_{t-n+1,t-1})$$

with neural networks

$$p(w_t \mid w_{<t}) \approx p(w_t \mid \phi(w_1, \ldots, w_{t-1}))$$

with fixed window

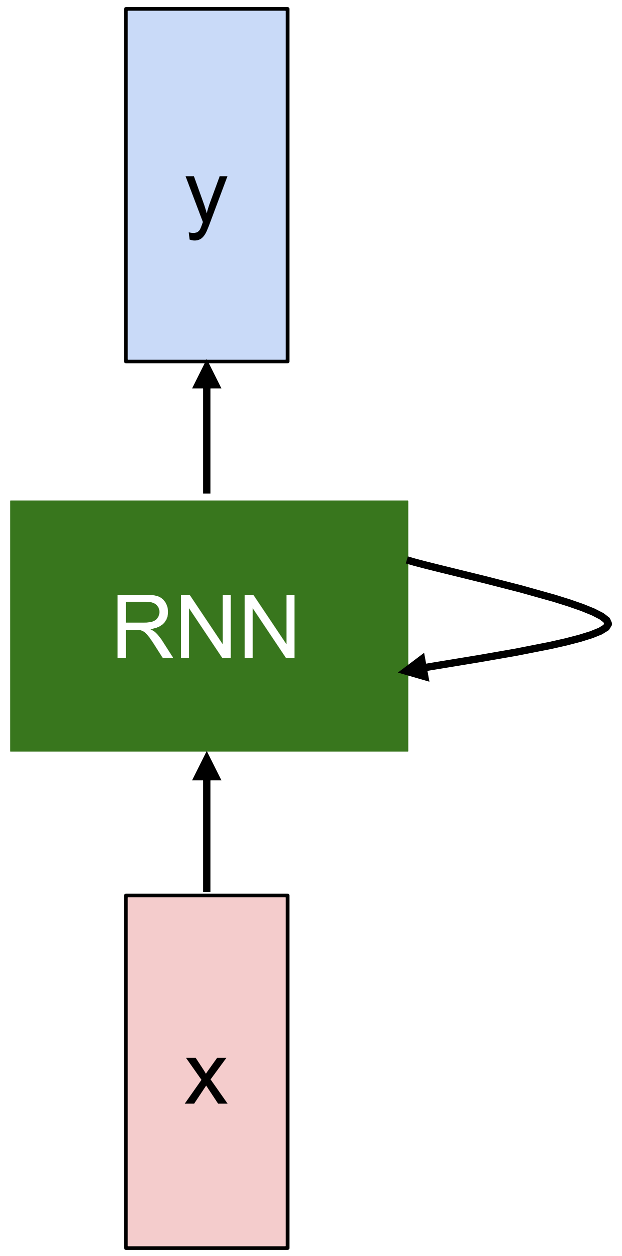
$$P(w_t | w_{<t}) \approx P(w_t | \phi(w_{t-n+1,t-1}))$$

with RNNs

$$\boxed{P(w_t | w_{<t}) \approx P(w_t | \mathbf{h}_t), \mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t)}$$

Simple RNNs

Components of RNN cells



$\mathbf{h}_0 \in \mathbb{R}^d$ is an initial state

function with weights W

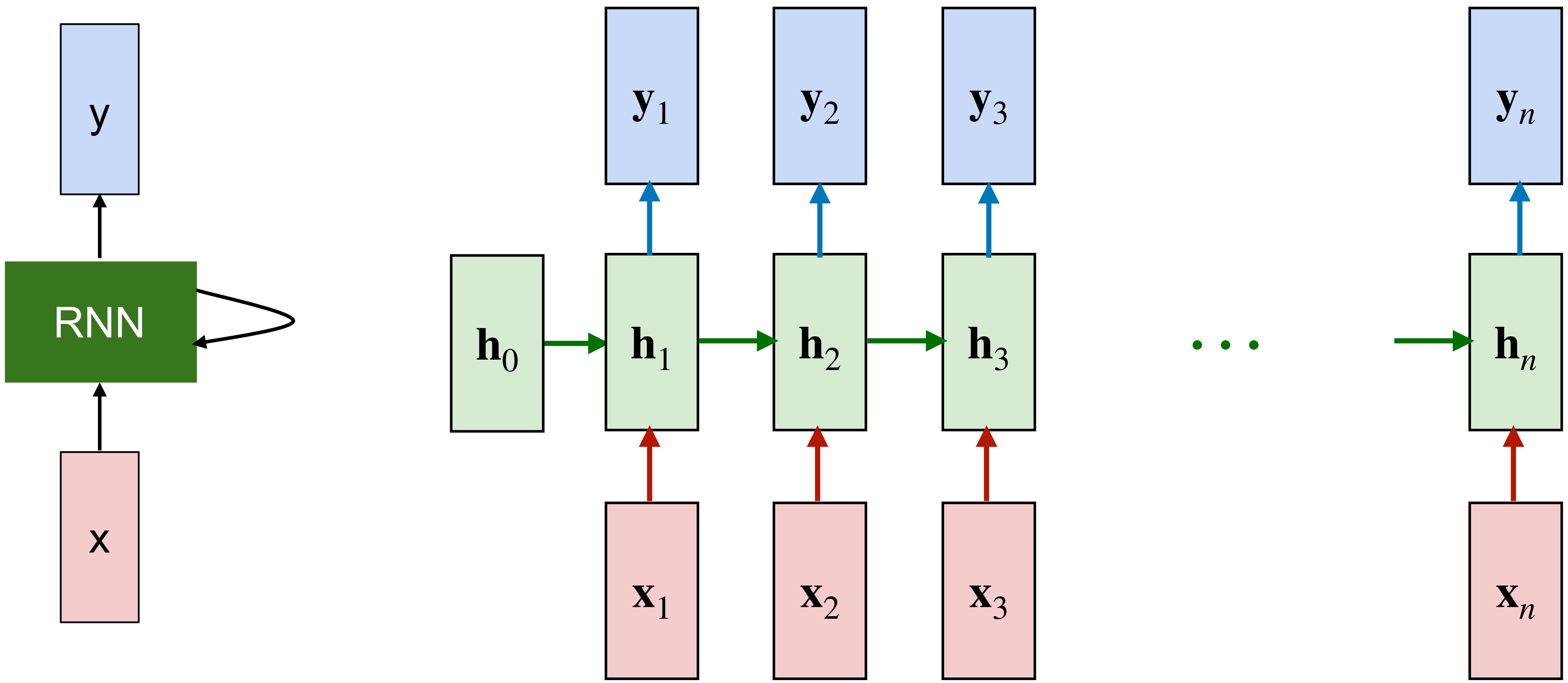
$$\boxed{\mathbf{h}_t = f_{\mathbf{W}}(\mathbf{h}_{t-1}, \mathbf{x}_t) \in \mathbb{R}^d}$$

new state   old state   input at time t

$\mathbf{h}_t$: hidden states which store information from $\mathbf{x}_1$ to $\mathbf{x}_t$

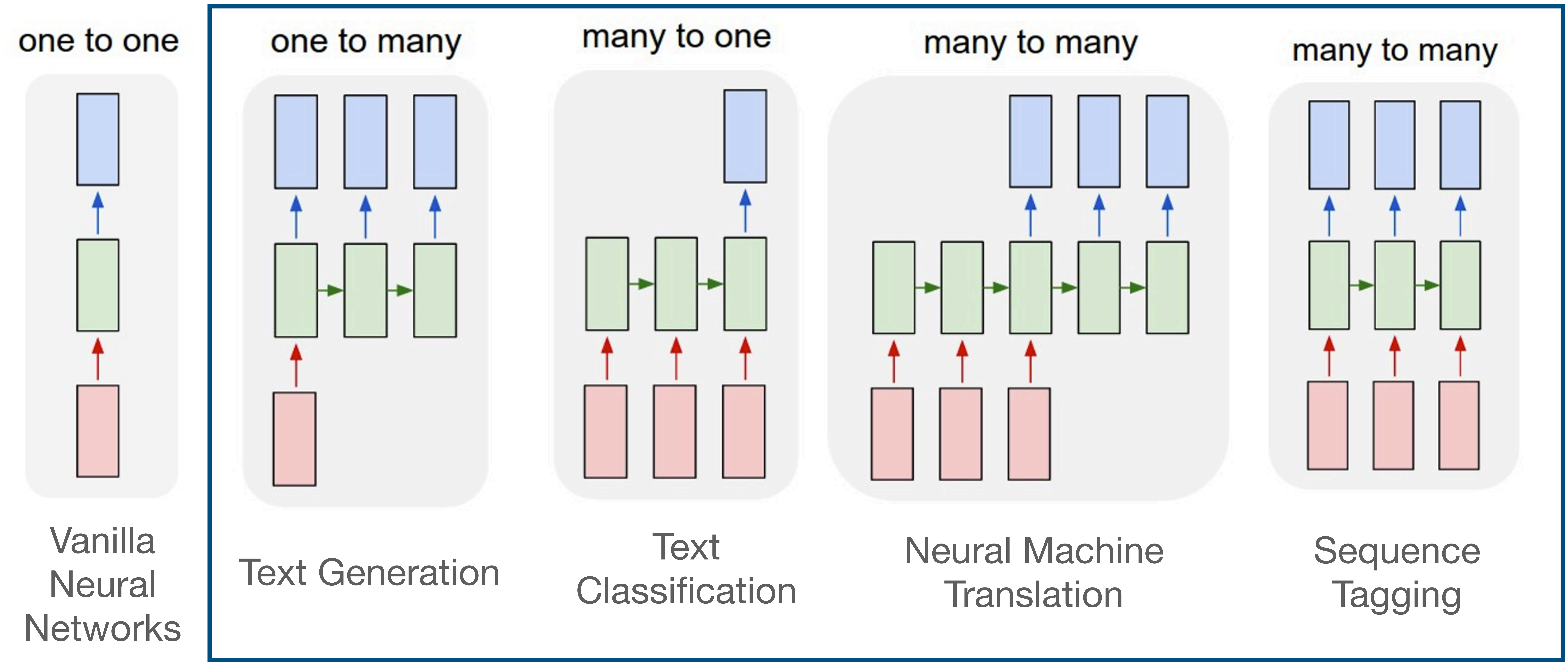
Output label for each time step: Denote $\hat{\mathbf{y}}_t = \text{softmax}(\mathbf{W}_o \mathbf{h}_t)$, $\mathbf{W}_o \in \mathbb{R}^{|L| \times d}$

Unrolling the RNN

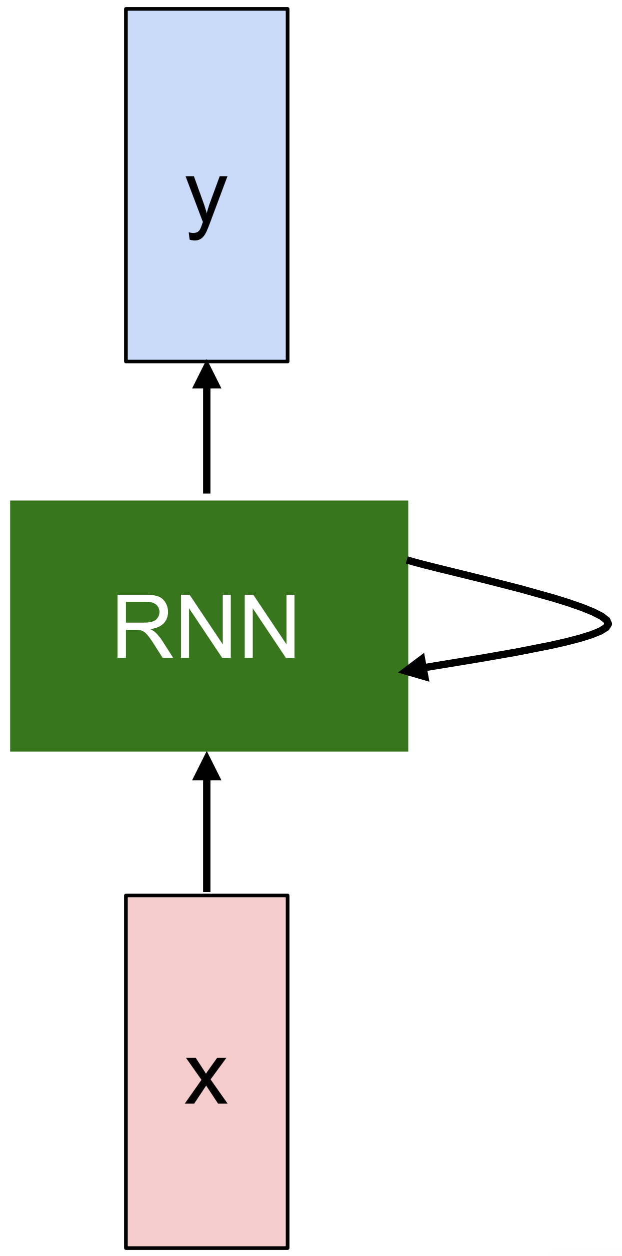


Structure of cell and weights are shared across time steps

Types of sequence processing problems



Simple (vanilla) RNNs



$\mathbf{h}_0 \in \mathbb{R}^d$ is an initial state

function with weights W

$$\boxed{\mathbf{h}_t = f_{\mathbf{W}}(\mathbf{h}_{t-1}, \mathbf{x}_t) \in \mathbb{R}^d}$$

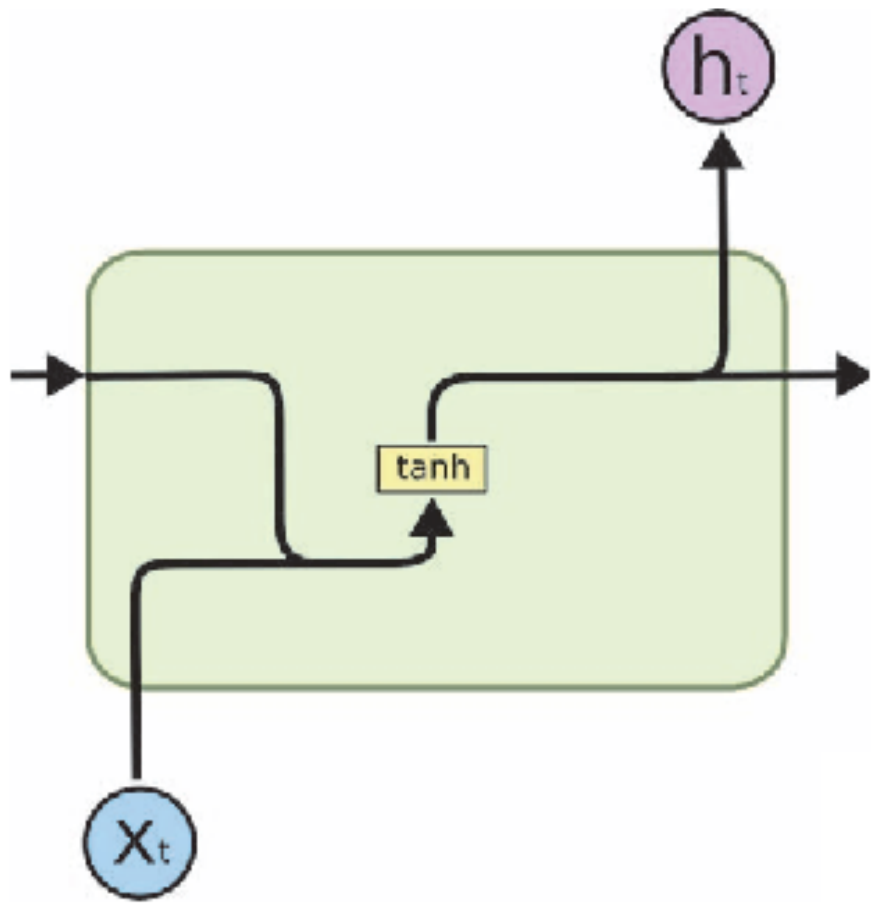
new state old state input at time t

$\mathbf{h}_t$: hidden states which store information from $\mathbf{x}_1$ to $\mathbf{x}_t$

Output label for each time step: Denote $\hat{\mathbf{y}}_t = \text{softmax}(\mathbf{W}_o \mathbf{h}_t)$, $\mathbf{W}_o \in \mathbb{R}^{|L| \times d}$

Simple (vanilla) RNNs:

$$\boxed{\mathbf{h}_t = g(\mathbf{W}_h \mathbf{h}_{t-1} + \mathbf{W}_x \mathbf{x}_t + \mathbf{b}) \in \mathbb{R}^d}$$



g: nonlinearity (e.g. tanh),

$\mathbf{W}_h \in \mathbb{R}^{d \times d}$, $\mathbf{W}_x \in \mathbb{R}^{d \times d_{in}}$, $\mathbf{b} \in \mathbb{R}^d$

RNN Language Model

output distribution

$$\hat{\boldsymbol{y}}^{(t)} = \text{softmax}\left(\boldsymbol{U}\boldsymbol{h}^{(t)} + \boldsymbol{b}_2\right) \in \mathbb{R}^{|V|}$$

Output label size: |V|

hidden states

$$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h\boldsymbol{h}^{(t-1)} + \boldsymbol{W}_e\boldsymbol{e}^{(t)} + \boldsymbol{b}_1\right)$$

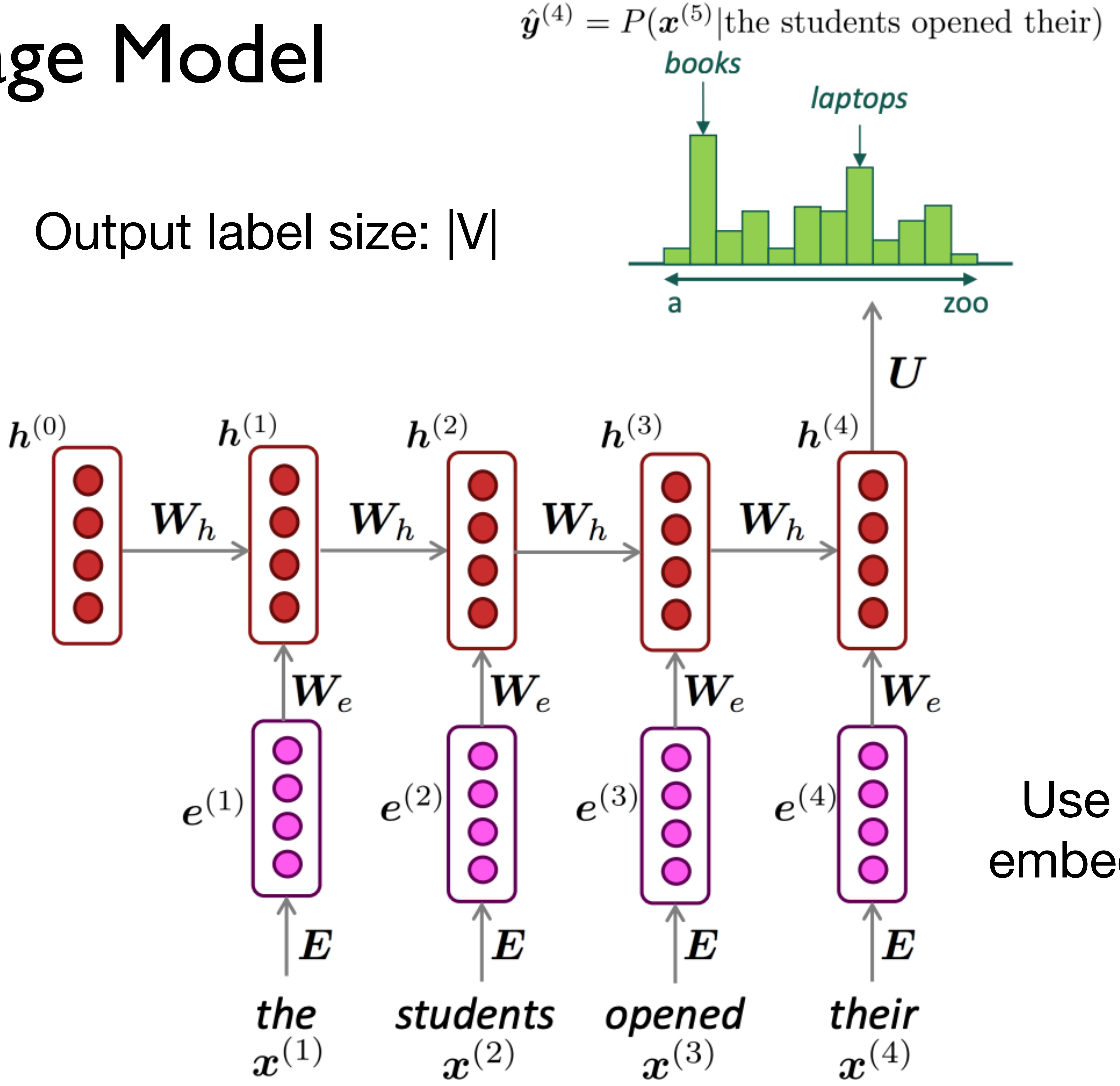
$\boldsymbol{h}^{(0)}$ is the initial hidden state

word embeddings

$$\boldsymbol{e}^{(t)} = \boldsymbol{E}\boldsymbol{x}^{(t)}$$

words / one-hot vectors

$$\boldsymbol{x}^{(t)} \in \mathbb{R}^{|V|}$$



Use word embeddings

Slide credit: Chris Manning

RNNs: pros and cons

- Advantages:
  - Can process any length input
  - Computation for step t can (in theory) use information from many steps back
  - Model size doesn't increase for longer input context
  - Same weights applied on every timestep (symmetry in how inputs are processed)

- Disadvantages:
  - Recurrent computation is slow (can't parallelize)
  - In practice, difficult to access information from many steps back

Can parallelize with transformers!

Progress on language models

On the Penn Treebank (PTB) dataset

Metric: perplexity

$$\text{ppl}(S) = 2^x \quad \text{where}$$

$$x = -\frac{1}{W}\sum_{i=1}^{n} \log_2 P(S^i)$$

KN5: Kneser-Ney 5-gram

Model	Individual
KN5	141.2
KN5 + cache	125.7
Feedforward NNLM	140.2
Log-bilinear NNLM	144.5
Syntactical NNLM	131.3
Recurrent NNLM	124.7
RNN-LDA LM	113.7

(Mikolov and Zweig, 2012): Context dependent recurrent neural network language model