COS 484

Natural Language Processing

L9: Recurrent neural networks

Spring 2024

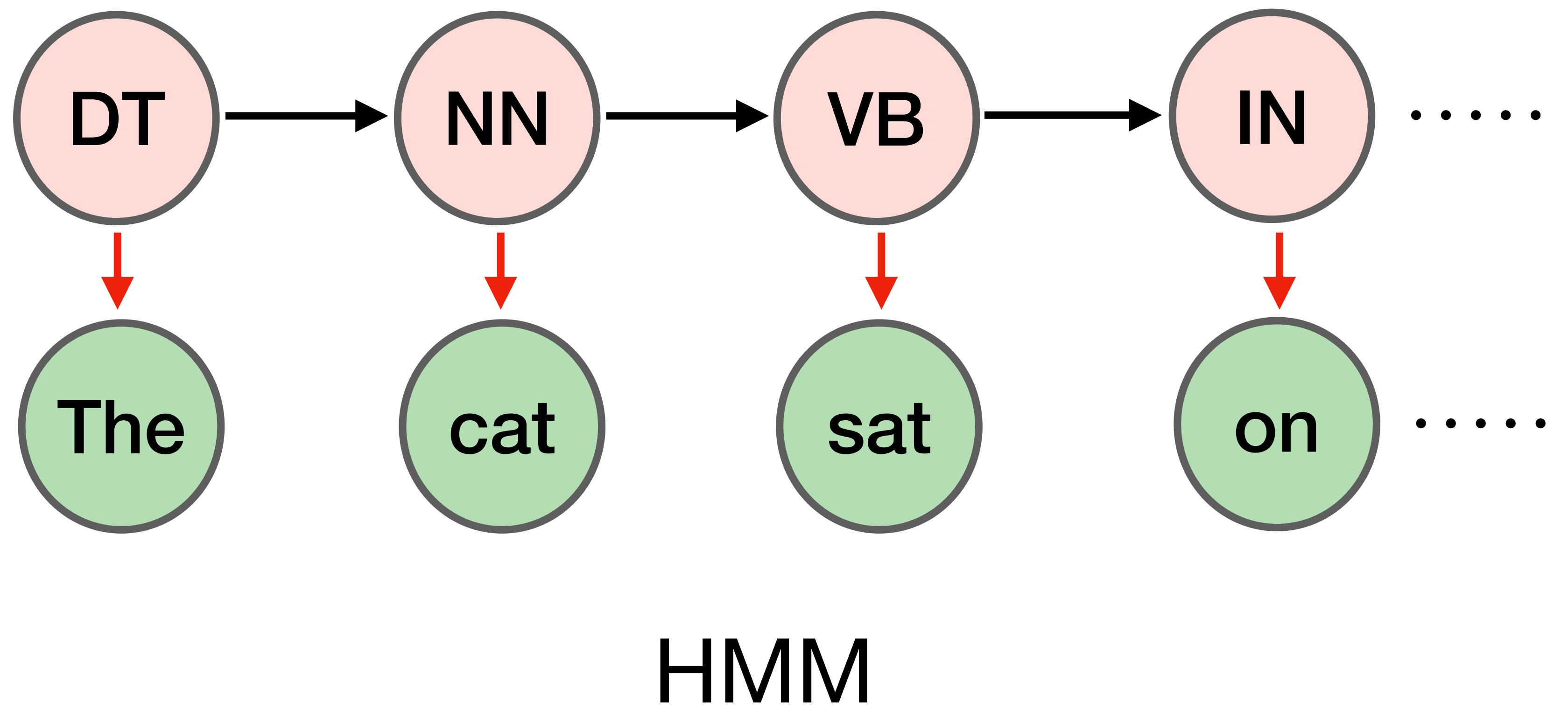
(Some slides adapted from Chris Manning, Abigail See)

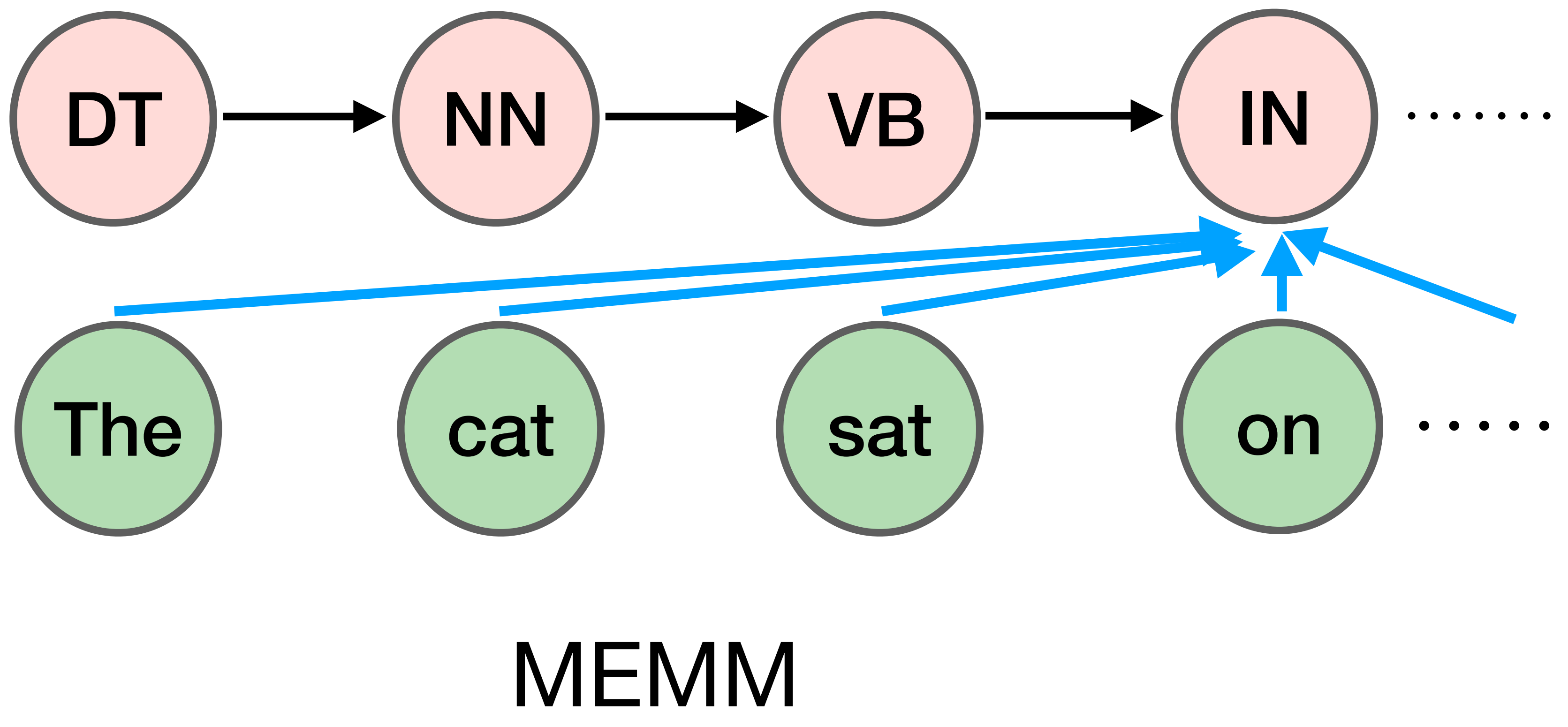
Midterm logistics

- A 3-hour timed exam from **March 7th 1:30pm** (Thu) to **March 8th 4:30pm (Fri)**
  - We will provide email support for the following times:
    Mar 7, 1:30-3:30pm
    Mar 7, 5:30-7:30pm
    Mar 8, 8-10am
    Mar 8, 10am-12pm
    Mar 8, 1pm-3pm
- Open-book (lecture slides & readings), no internet / ChatGPT allowed
- Practice midterms have been released on Ed
- All topics up to today's lecture will be covered in the midterm

Recurrent neural networks (RNNs)

How can we model sequences using **neural networks**?





- Recurrent neural networks = A class of neural networks used to model sequences, allowing to handle **variable length inputs**

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

Recap: n-gram vs neural language models

Language models: Given $x_1, x_2, \ldots, x_n \in V$, the goal is to model:

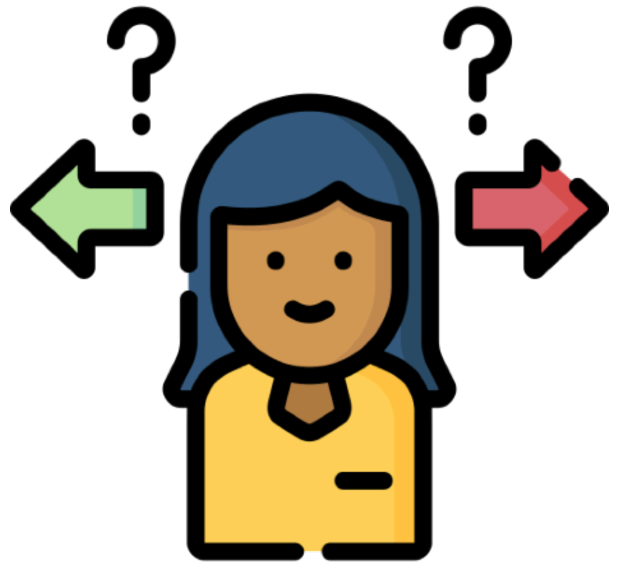
$$P(x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} P(x_i \mid x_1, \ldots, x_{i-1})$$

N-gram models:

$$P(\text{sat}|\text{the cat}) = \frac{\text{count}(\text{the cat sat})}{\text{count}(\text{the cat})}$$

Dilemma:

- We need to model bigger context!
- The # of probabilities that we need to estimate grow exponentially with window size!



~~As the proctor started the clock, the~~ students opened their ______

Recap: Feedforward neural language models

Feedforward neural language models approximate the probability based on the previous *m* (e.g., 5) words - *m* is a hyper-parameter!

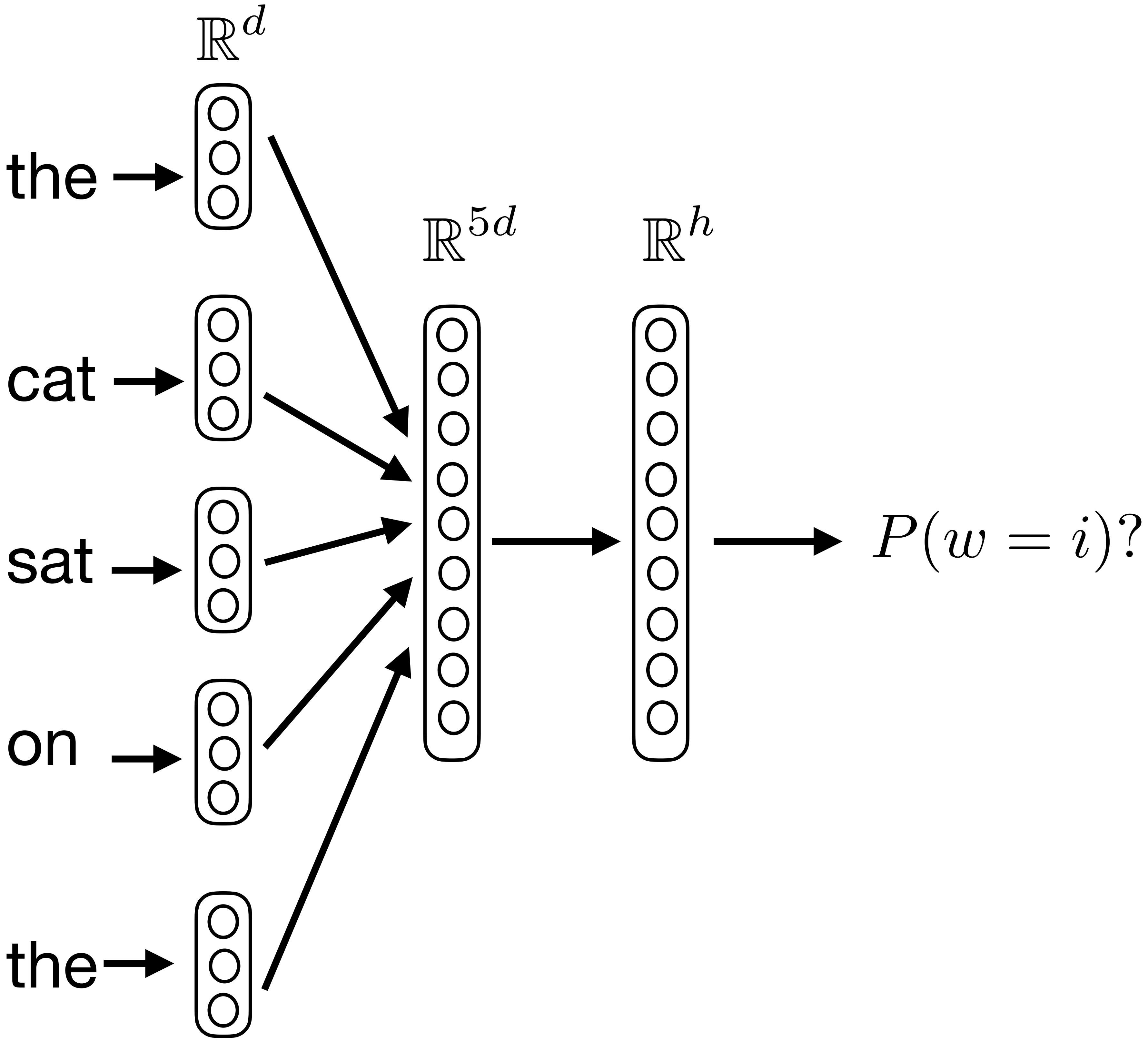
$$P(x_1, x_2, \ldots, x_n) \approx \prod_{i=1}^{n} P(x_i \mid x_{i-m+1}, \ldots, x_{i-1})$$

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

d: word embedding size

h: hidden size

It is a |V|-way classification problem!



Recap: Feedforward neural language models

P(mat | the cat sat on the) = ? d: word embedding size h: hidden size

- Input layer (m= 5):

  $\mathbf{x} = [e(\text{the}); e(\text{cat}); e(\text{sat}); e(\text{on}); e(\text{the})] \in \mathbb{R}^{md}$

- Hidden layer:

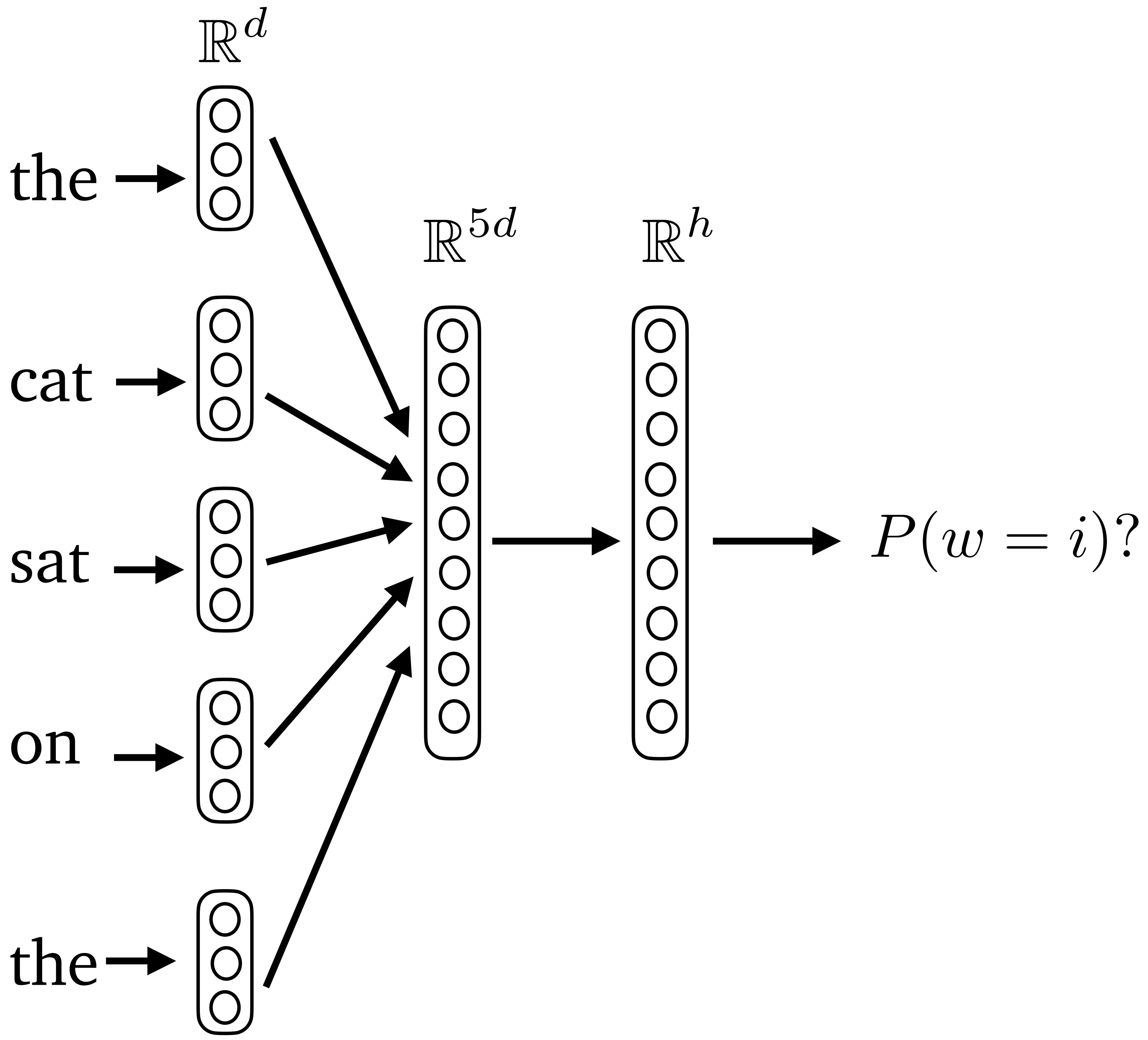
  $\mathbf{h} = \tanh(\mathbf{W}\mathbf{x} + \mathbf{b}) \in \mathbb{R}^{h}$

- Output layer

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

$$P(w = i \mid \text{the cat sat on the})$$

$$= \text{softmax}_i(\mathbf{z}) = \frac{e^{z_i}}{\sum_k e^{z_k}}$$



Recap: Feedforward neural language models

The Fat Cat Sat on the Mat is a 1996 children's book by Nurit Karlin. Published by Harper Collins as part of the reading readiness program, the book stresses the ability to read words of specific structure, such as -at.

→

the fat cat sat on → the

fat cat sat on the → mat

cat sat on the mat → is

sat on the mat is → a

…

Limitations?

- $\mathbf{W}$ **linearly** scales with the context size *m*
- The models learns separate patterns for different positions!

$\mathbf{W}[: ,3d : 5d]$

the fat cat sat on → the

fat cat sat on the → mat

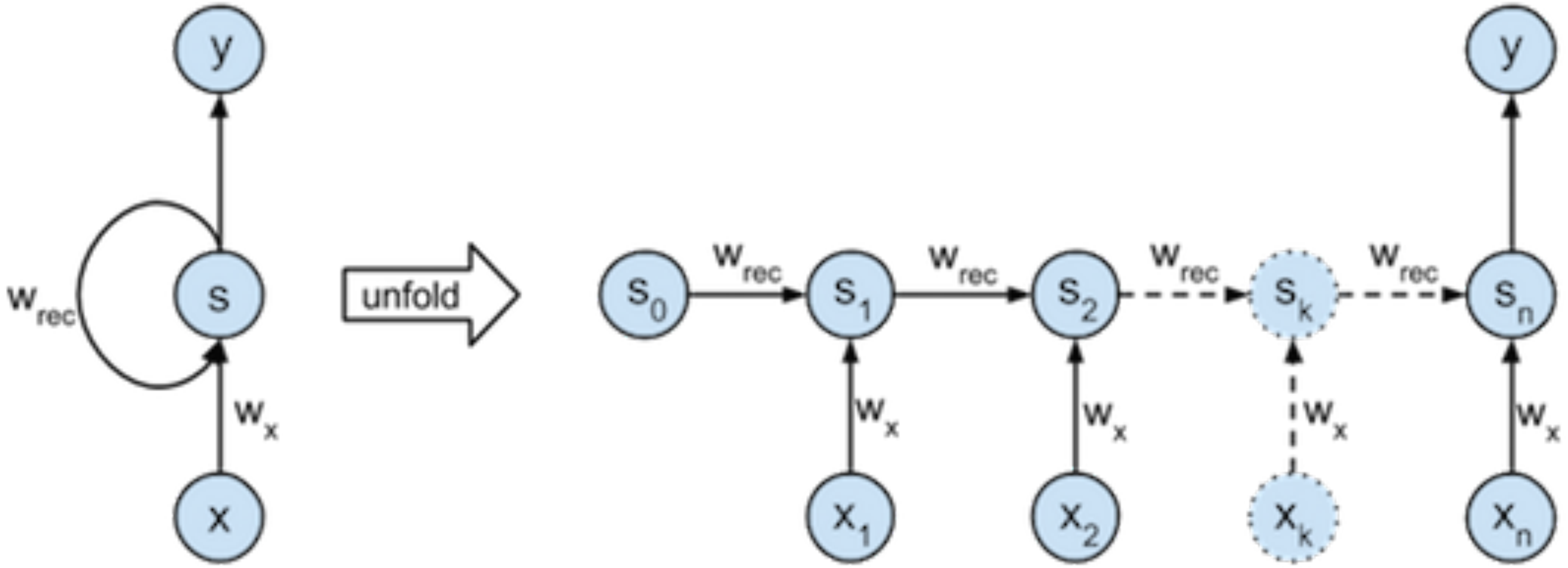
cat sat on the mat → is

$\mathbf{W}[: ,1d : 3d]$

“sat on” corresponds to different parameters in $\mathbf{W}$

Recurrent neural networks (RNNs)

A family of neural networks that can handle **variable length inputs**



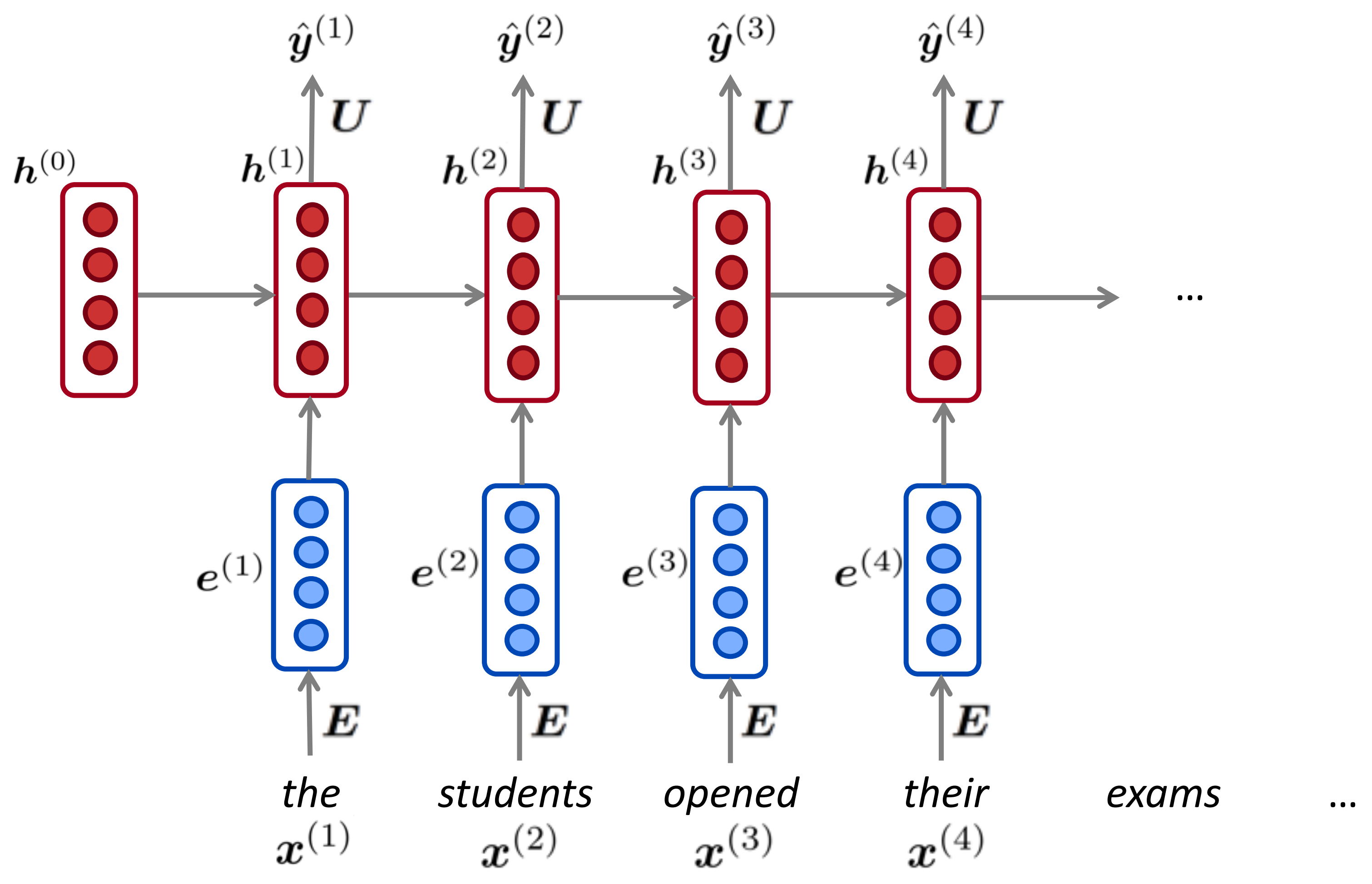
A function: $\mathbf{y} = \text{RNN}(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n) \in \mathbb{R}^h$ where $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$

Core idea: apply the same weights repeatedly at different positions!

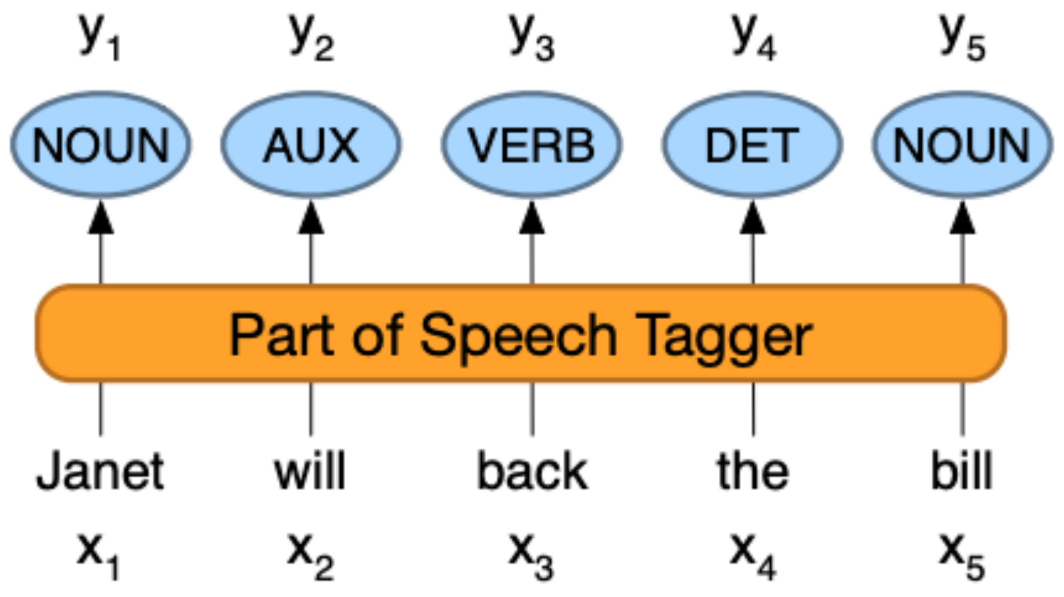
Recurrent neural networks (RNNs)

Highly effective approach for language modeling, sequence tagging, text classification:

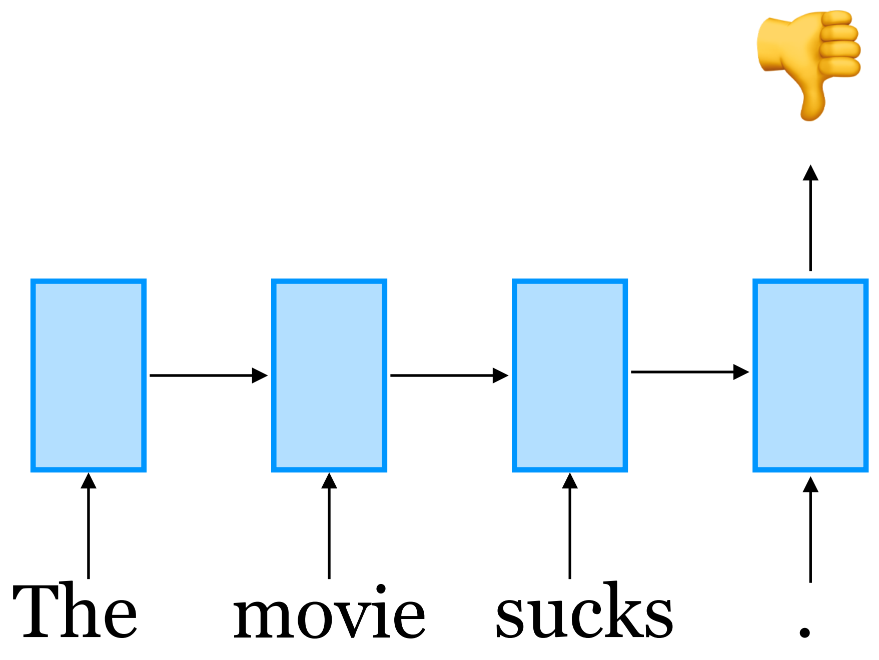
Language modeling



Sequence tagging



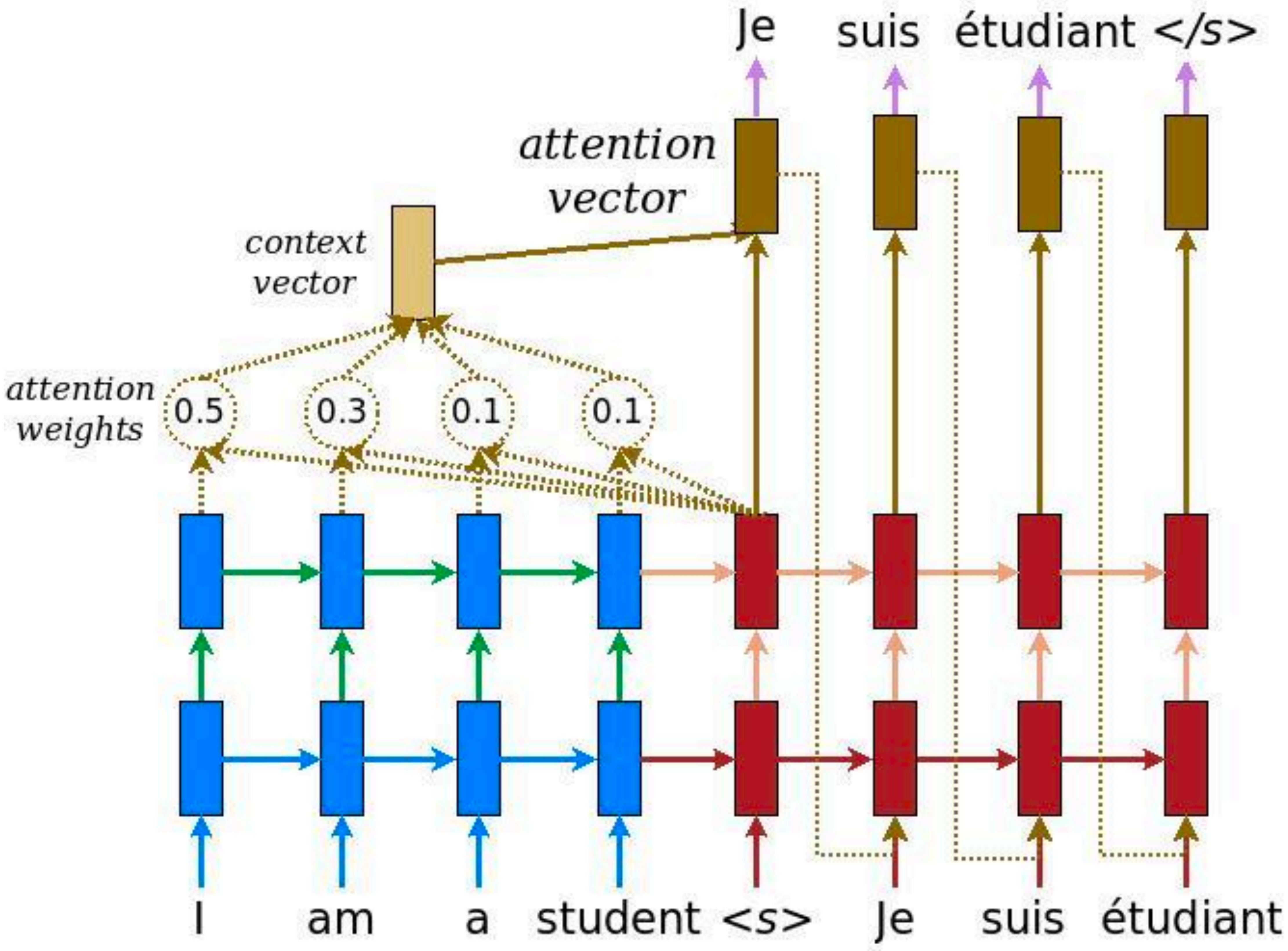
Text classification



Recurrent neural networks (RNNs)

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

sequence-to-sequence models



(Sutskever et al., 2014): Sequence to Sequence Learning with Neural Networks

Simple recurrent neural networks

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

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

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

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

Simple RNNs:

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

g: nonlinearity (e.g. tanh, ReLU),

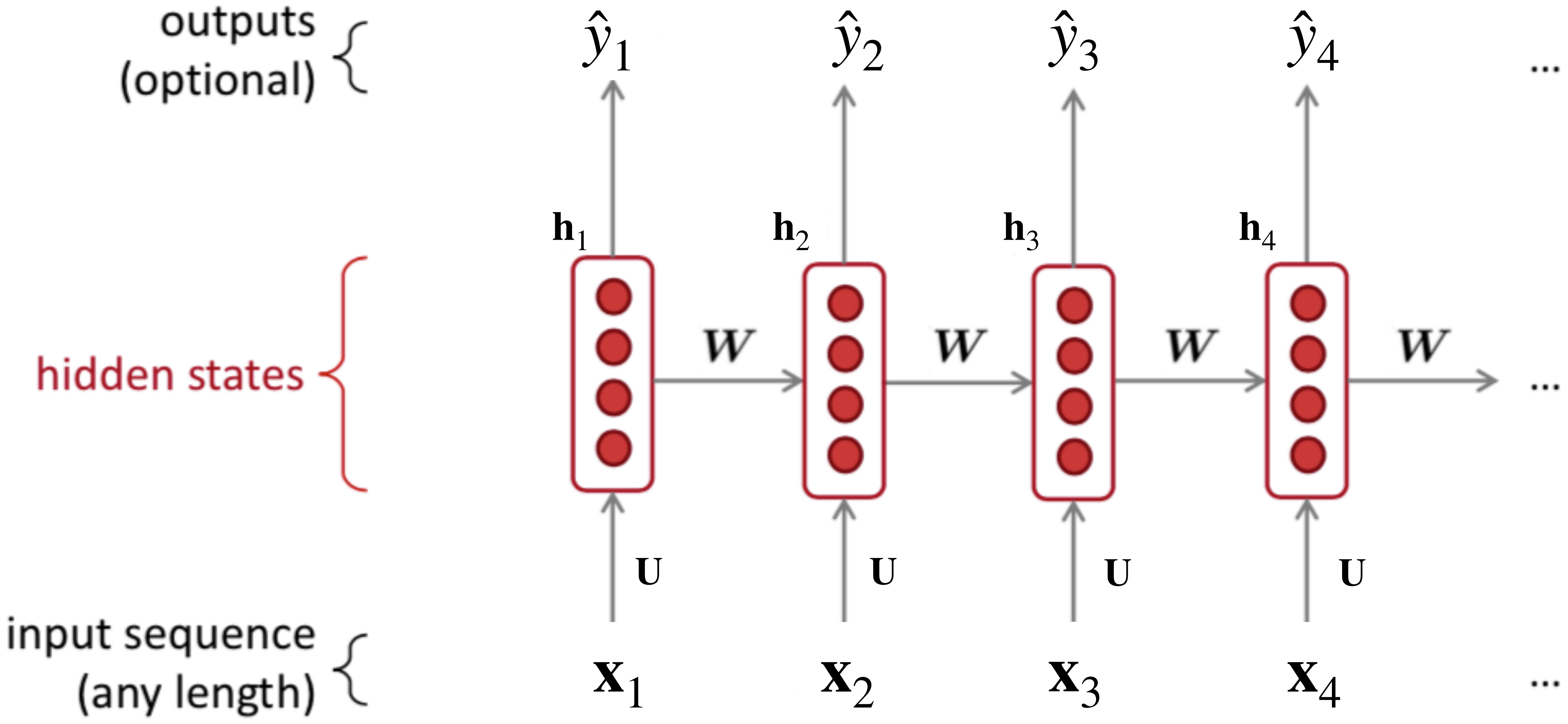
$$\mathbf{W} \in \mathbb{R}^{h \times h}, \mathbf{U} \in \mathbb{R}^{h \times d}, \mathbf{b} \in \mathbb{R}^h$$

This model contains $h \times (h + d + 1)$ parameters, and optionally *h for* $\mathbf{h}_0$ (a common way is just to set $\mathbf{h}_0$ as $\mathbf{0}$)

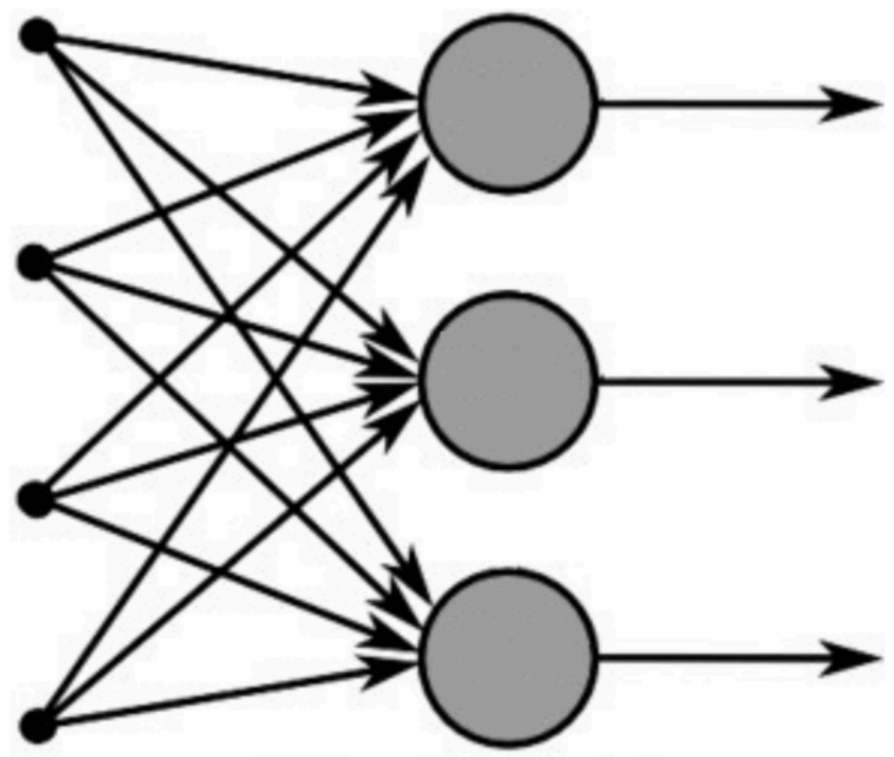
Simple recurrent neural networks

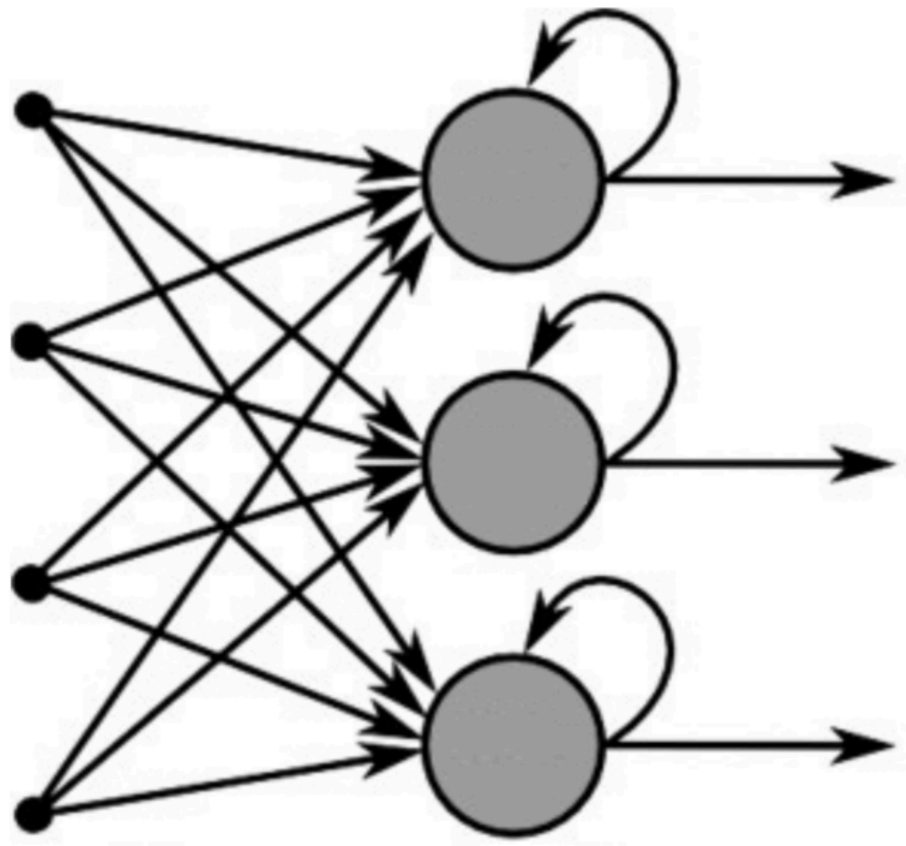
$$\mathbf{h}_t = g(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b}) \in \mathbb{R}^h$$

Key idea: apply the same weights $\mathbf{W}, \mathbf{U}, \mathbf{b}$ repeatedly



RNNs vs Feedforward NNs


Feed-Forward Neural Network


Recurrent Neural Network

$$\mathbf{h}_1 = g(\mathbf{W}^{(1)}\mathbf{x} + \mathbf{b}^{(1)}) \in \mathbb{R}^{h_1}$$

$$\mathbf{h}_2 = g(\mathbf{W}^{(2)}\mathbf{h}_1 + \mathbf{b}^{(2)}) \in \mathbb{R}^{h_2}$$

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

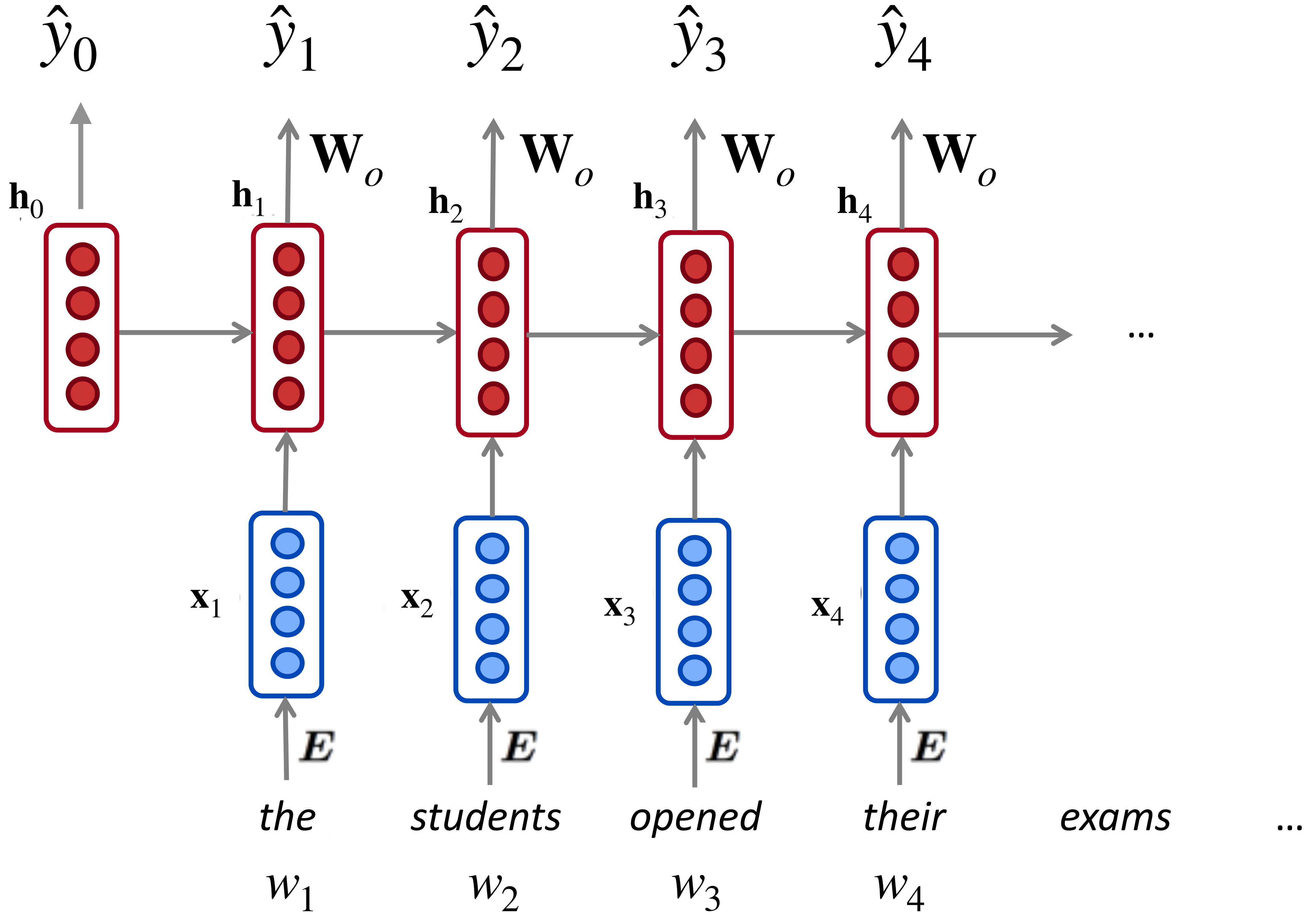
Recurrent neural language models (RNNLMs)

Recurrent neural language models (RNNLMs)

$$P(w_1, w_2, \ldots, w_n) = P(w_1) \times P(w_2 \mid w_1) \times P(w_3 \mid w_1, w_2) \times \ldots \times P(w_n \mid w_1, w_2, \ldots, w_{n-1})$$

$$= P(w_1 \mid \mathbf{h}_0) \times P(w_2 \mid \mathbf{h}_1) \times P(w_3 \mid \mathbf{h}_2) \times \ldots \times P(w_n \mid \mathbf{h}_{n-1})$$

No Markov assumption here!

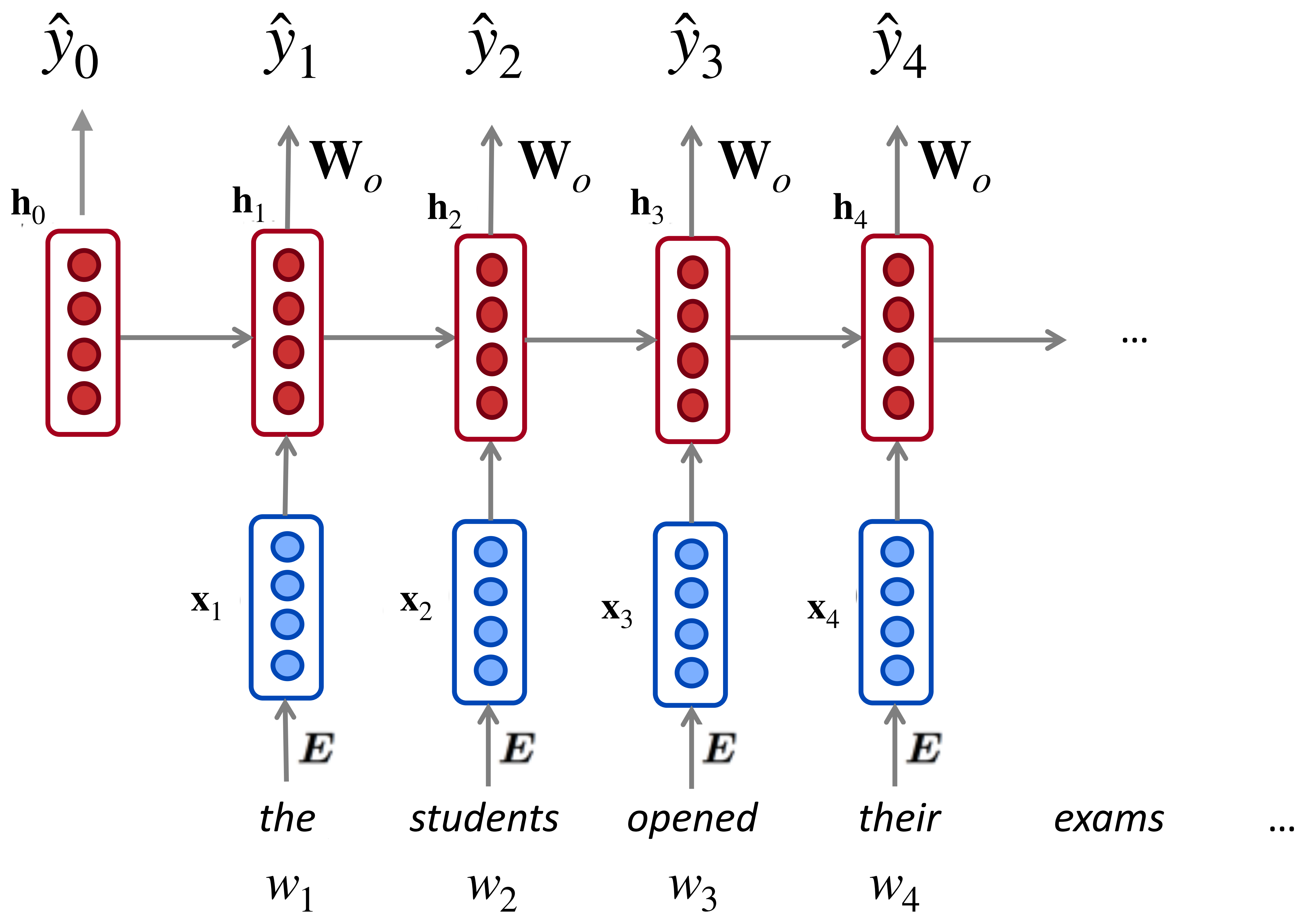


Denote $\hat{\mathbf{y}}_t = softmax(\mathbf{W}_o \mathbf{h}_t)$, $\mathbf{W}_o \in \mathbb{R}^{|V| \times h}$

$$= \hat{y}_0(w_1) \times \hat{y}_1(w_2) \ldots \times \hat{y}_{n-1}(w_n)$$

$\hat{y}_1(w_2)$ = the probability of w_2

Recurrent neural language models (RNNLMs)



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

$$\hat{\mathbf{y}}_t = softmax(\mathbf{W}_o\mathbf{h}_t)$$

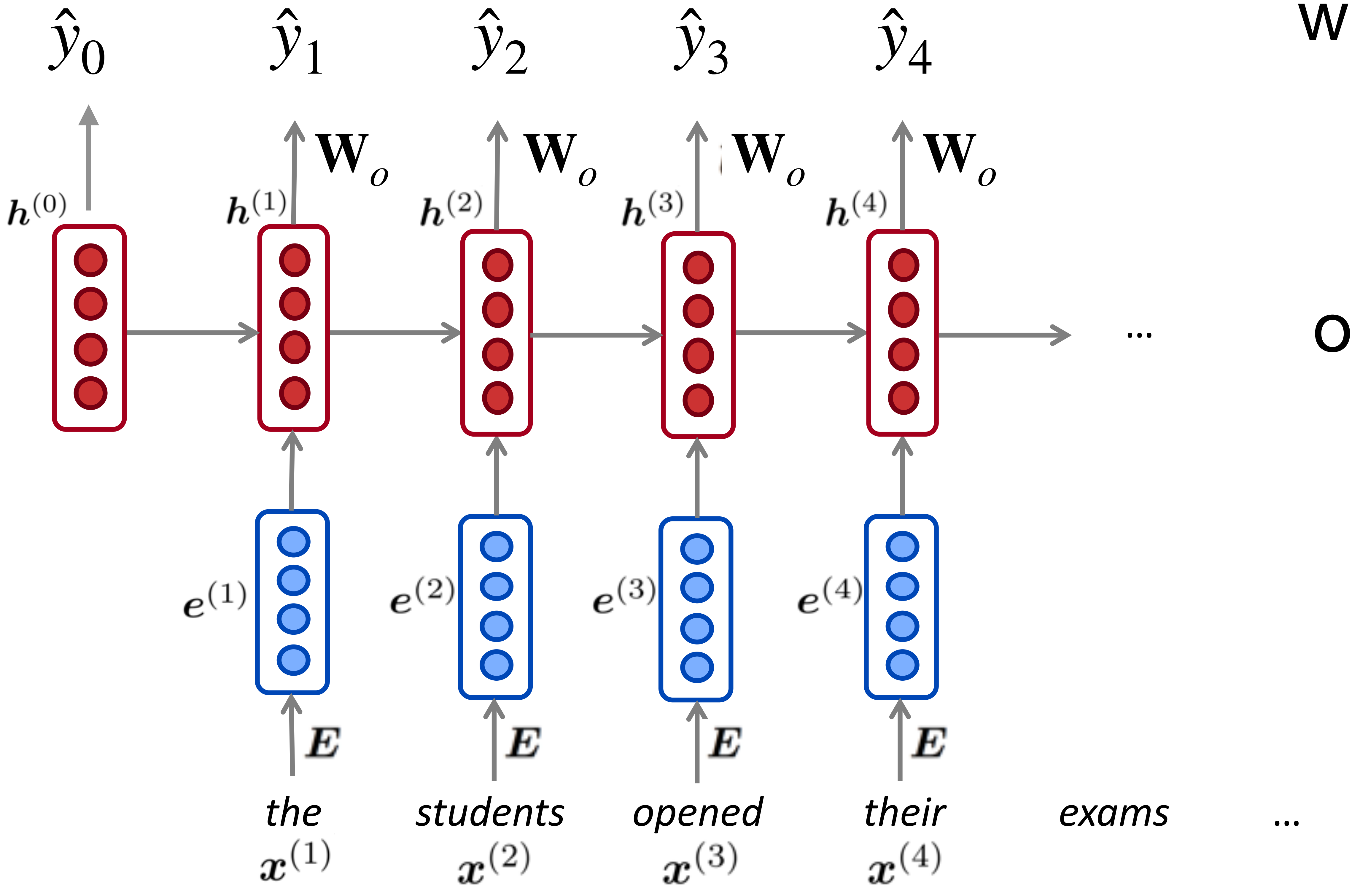
Training loss:

$$\boxed{L(\theta) = -\frac{1}{n}\sum_{t=1}^{n} \log \hat{\mathbf{y}}_{t-1}(w_t)}$$

Trainable parameters:

$$\theta = \{\mathbf{W}, \mathbf{U}, \mathbf{b}, \mathbf{W}_o, \mathbf{E}\}$$

RNNLMs: weight tying



word embeddings (= input embeddings):

$$\mathbf{E} \in \mathbb{R}^{|V| \times d}$$

output embeddings:

$$\mathbf{W}_o \in \mathbb{R}^{|V| \times h}$$

If $d = h$, we can just merge $\mathbf{E}$ and $\mathbf{W}_o$!

$$\theta = \{\mathbf{W}, \mathbf{U}, \mathbf{b}, \mathbf{E}\}$$

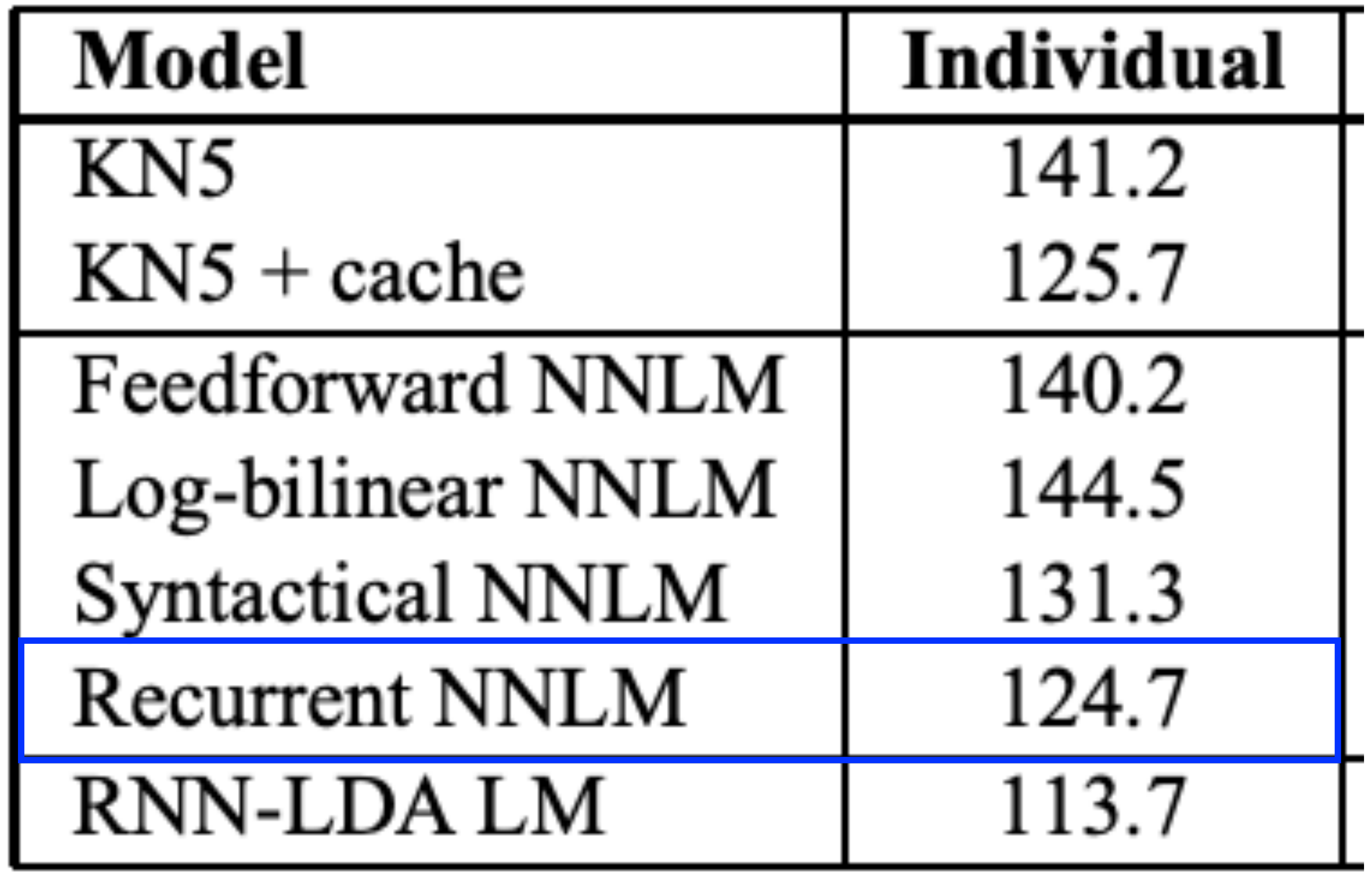
It works better empirically and becomes a common practice

Progress on language models

On the Penn Treebank (PTB) dataset

Metric: perplexity

KN5: Kneser-Ney 5-gram



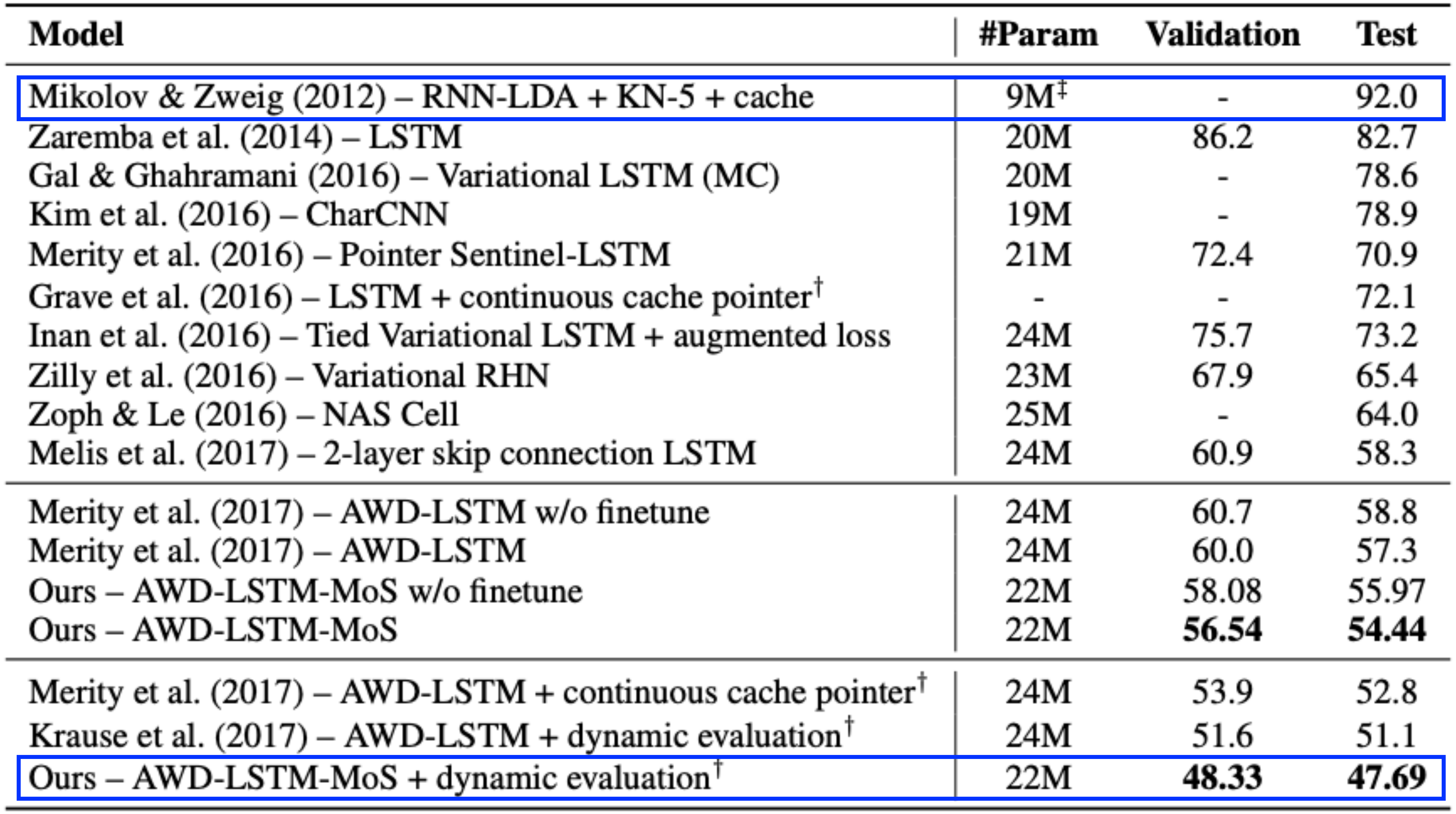
Model	Individual
KN5	141.2
KN5 + cache	125.7
Feedforward NNLM	140.2
Log-bilinear NNLM	144.5
Syntactical NNLM	131.3
Recurrent NNLM	124.7
RNN-LDA LM	113.7

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

Progress on language models

On the Penn Treebank (PTB) dataset

Metric: perplexity



Model	#Param	Validation	Test
Mikolov & Zweig (2012) – RNN-LDA + KN-5 + cache	9M‡	-	92.0
Zaremba et al. (2014) – LSTM	20M	86.2	82.7
Gal & Ghahramani (2016) – Variational LSTM (MC)	20M	-	78.6
Kim et al. (2016) – CharCNN	19M	-	78.9
Merity et al. (2016) – Pointer Sentinel-LSTM	21M	72.4	70.9
Grave et al. (2016) – LSTM + continuous cache pointer†	-	-	72.1
Inan et al. (2016) – Tied Variational LSTM + augmented loss	24M	75.7	73.2
Zilly et al. (2016) – Variational RHN	23M	67.9	65.4
Zoph & Le (2016) – NAS Cell	25M	-	64.0
Melis et al. (2017) – 2-layer skip connection LSTM	24M	60.9	58.3
Merity et al. (2017) – AWD-LSTM w/o finetune	24M	60.7	58.8
Merity et al. (2017) – AWD-LSTM	24M	60.0	57.3
Ours – AWD-LSTM-MoS w/o finetune	22M	58.08	55.97
Ours – AWD-LSTM-MoS	22M	**56.54**	**54.44**
Merity et al. (2017) – AWD-LSTM + continuous cache pointer†	24M	53.9	52.8
Krause et al. (2017) – AWD-LSTM + dynamic evaluation†	24M	51.6	51.1
Ours – AWD-LSTM-MoS + dynamic evaluation†	22M	**48.33**	**47.69**

(Yang et al, 2018): Breaking the Softmax Bottleneck: A High-Rank RNN Language Model

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

Disadvantages:

- Recurrent computation is slow (can't parallelize) ← Transformers can!
- In practice, difficult to access information from many steps back (optimization issue) ← We will see some advanced RNNs (e.g., LSTMs, GRUs)

Training RNNLMs

- Forward pass + backward pass (compute gradients)
- Forward pass:

```
$L = 0 \quad \mathbf{h}_0 = \mathbf{0}$
For $t$ =1, 2, …, $n$
    $y = -\log \mathrm{softmax}(\mathbf{W}_o \mathbf{h}_{t-1})(w_t)$
    $\mathbf{x}_t = e(w_t)$
    $\mathbf{h}_t = g(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b})$
    $L = L + \frac{1}{n} y$
```

accumulate loss

What is the running time of a forward pass?

What is the running time of a forward pass?

(a) $O(h \times (d + h + |V|))$

(b) $O(n \times h \times (d + h + |V|))$

(c) $O(n \times (d + h + |V|))$

(d) $O(n \times h \times (d + h))$

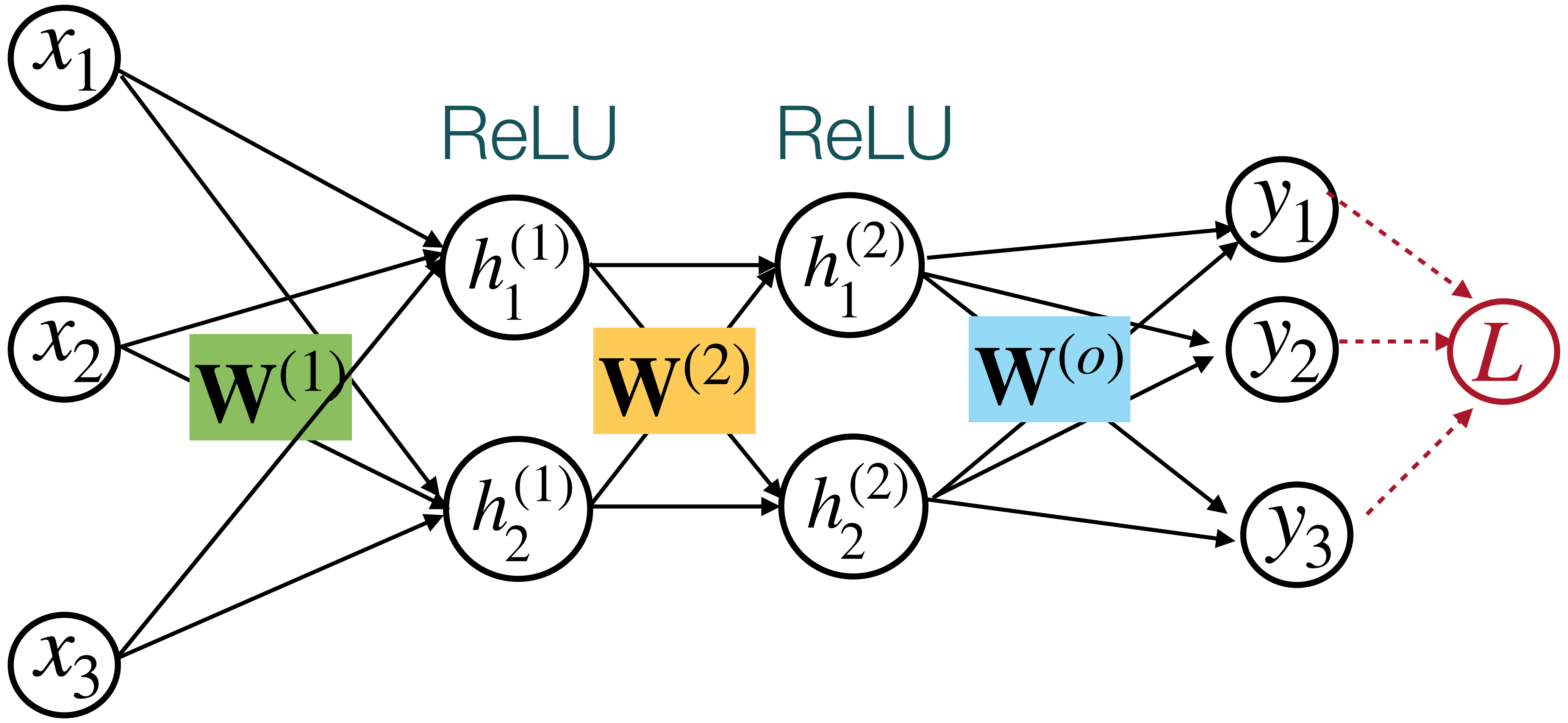
The answer is (b).

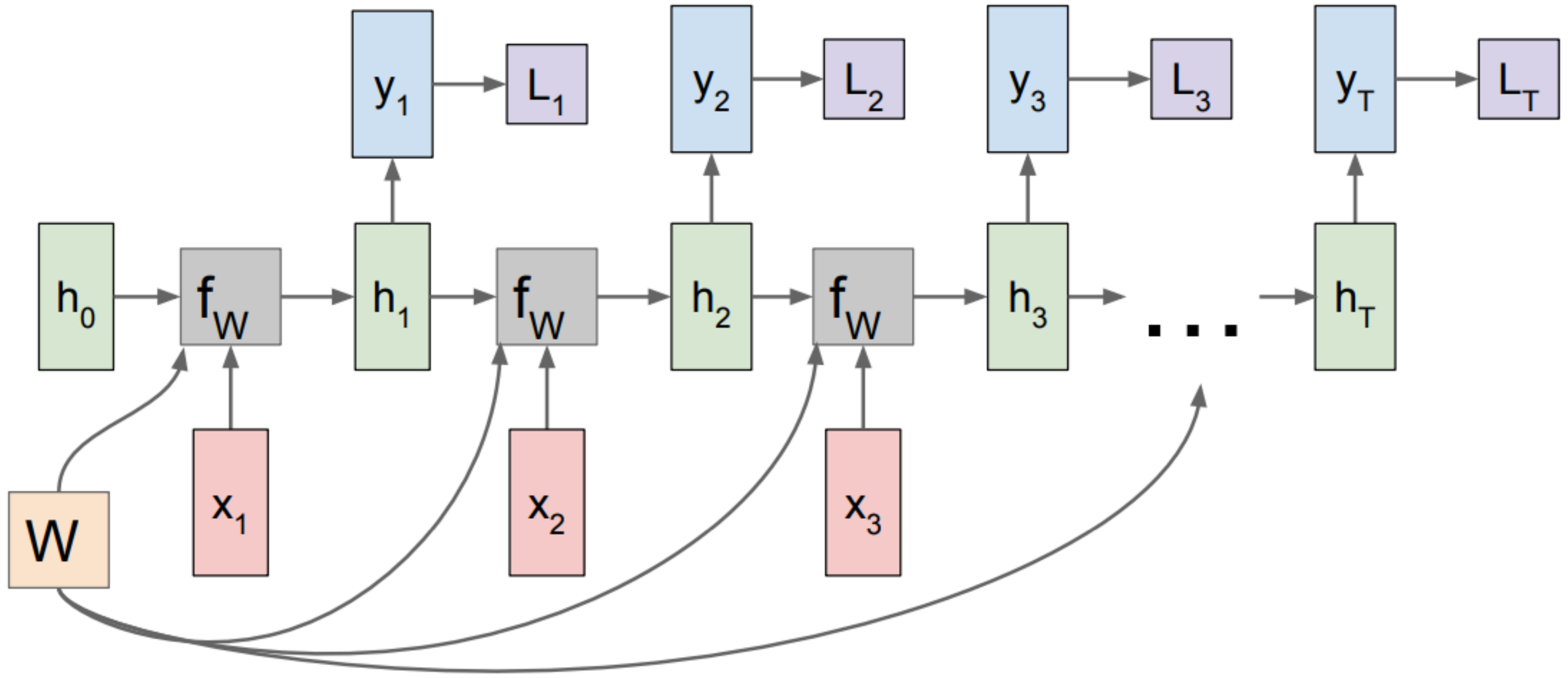
```
$L = 0 \quad \mathbf{h}_0 = \mathbf{0}$
For $t = 1, 2, \ldots, n$
    $y = -\log \text{softmax}(\mathbf{W}_o \mathbf{h}_{t-1})(w_t)$
    $\mathbf{x}_t = e(w_t)$
    $\mathbf{h}_t = g(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b})$
    $L = L + \frac{1}{n} y$
```

n = number of time steps
h = hidden dimension
d = word vector dimension
V = output vocabulary

Training RNNLMs

- Backward pass:
  - Backpropagation? Yes, but not that simple!





- The algorithm is called Backpropagation Through Time (BPTT).

Backpropagation through time

$$\mathbf{h}_1 = g(\boxed{\mathbf{W}}\mathbf{h}_0 + \mathbf{U}\mathbf{x}_1 + \mathbf{b})$$

$$\mathbf{h}_2 = g(\boxed{\mathbf{W}}\mathbf{h}_1 + \mathbf{U}\mathbf{x}_2 + \mathbf{b})$$

$$\mathbf{h}_3 = g(\boxed{\mathbf{W}}\mathbf{h}_2 + \mathbf{U}\mathbf{x}_3 + \mathbf{b}) \qquad \hat{\mathbf{y}}_3 = \mathrm{softmax}(\mathbf{W}_o\mathbf{h}_3)$$

$$L_3 = -\log \hat{\mathbf{y}}_3(w_4)$$

First, compute gradient with respect to hidden vector of last time step: $\dfrac{\partial L_3}{\partial \mathbf{h}_3}$

$$\frac{\partial L_3}{\partial \mathbf{W}} = \frac{\partial L_3}{\partial \mathbf{h}_3}\frac{\partial \mathbf{h}_3}{\partial \mathbf{W}} + \frac{\partial L_3}{\partial \mathbf{h}_3}\frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_2}\frac{\partial \mathbf{h}_2}{\partial \mathbf{W}} + \frac{\partial L_3}{\partial \mathbf{h}_3}\frac{\partial \mathbf{h}_3}{\partial \mathbf{h}_2}\frac{\partial \mathbf{h}_2}{\partial \mathbf{h}_1}\frac{\partial \mathbf{h}_1}{\partial \mathbf{W}}$$

More generally, $\boxed{\dfrac{\partial L}{\partial \mathbf{W}} = -\dfrac{1}{n}\sum_{t=1}^{n}\sum_{k=1}^{t}\dfrac{\partial L_t}{\partial \mathbf{h}_t}\left(\prod_{j=k+1}^{t}\dfrac{\partial \mathbf{h}_j}{\partial \mathbf{h}_{j-1}}\right)\dfrac{\partial \mathbf{h}_k}{\partial \mathbf{W}}}$

If *k* and *t* are far away, the gradients can grow/shrink exponentially

(called the gradient exploding or gradient vanishing problem)

What if gradients become too large or small?

What will happen if the gradients become too large or too small?

(a) If too large, the model will become difficult to converge

(b) If too small, the model can’t capture long-term dependencies

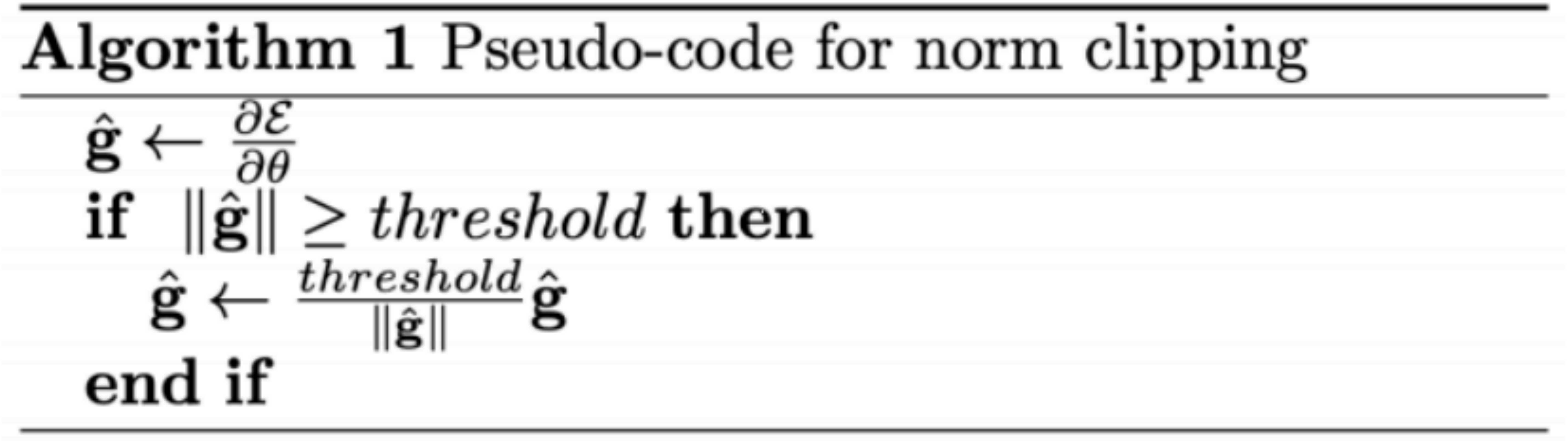
(c) If too small, the model may capture a wrong recent dependency

(d) All of the above

All of these are correct, so (d) ☺

Backpropagation through time

One solution for **gradient exploding** is called **gradient clipping** — if the norm of the gradient is greater than some threshold, scale it down before applying SGD update.



Algorithm 1 Pseudo-code for norm clipping

```
$\hat{\mathbf{g}} \leftarrow \frac{\partial \mathcal{E}}{\partial \theta}$
if $\|\hat{\mathbf{g}}\| \geq threshold$ then
    $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$
end if
```

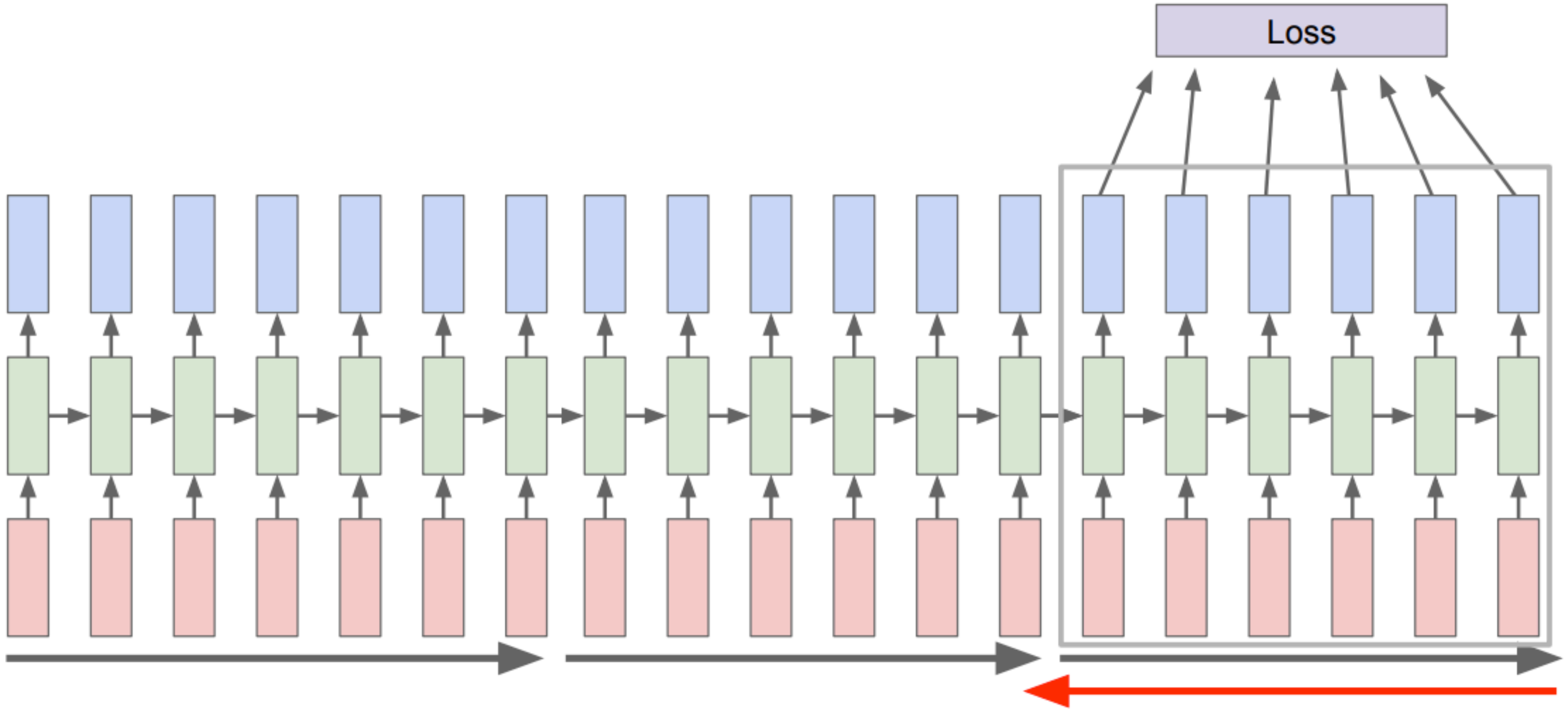
Intuition: take a step in the same direction but a smaller step!

Gradient vanishing is a harder problem to solve:

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

Truncated backpropagation through time

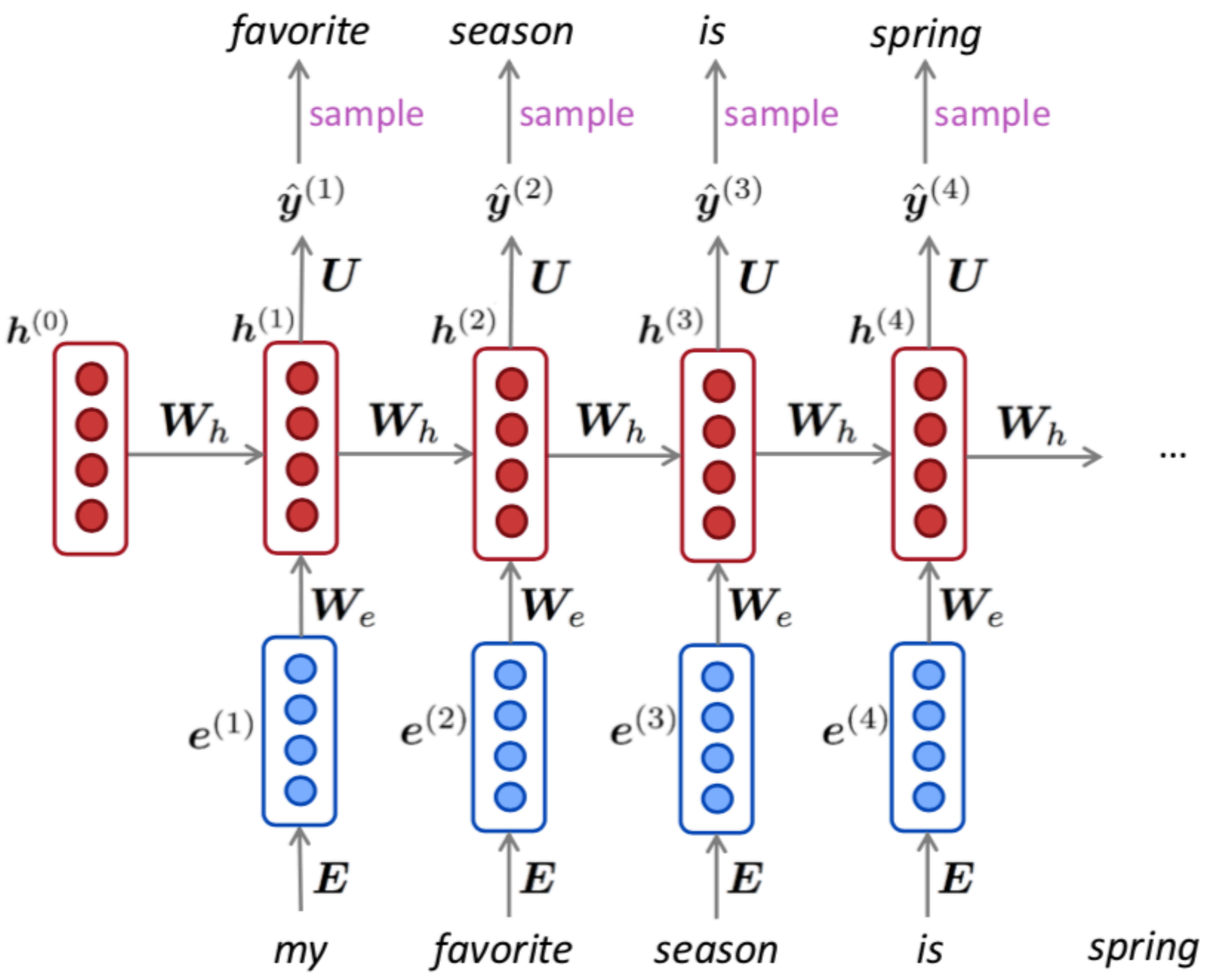
- Backpropagation is very expensive if you handle long sequences



- Run forward and backward through chunks of the sequence instead of whole sequence
- Carry hidden states forward in time forever, but only back-propagate for some smaller number of steps

Applications and variants

Application: Text generation



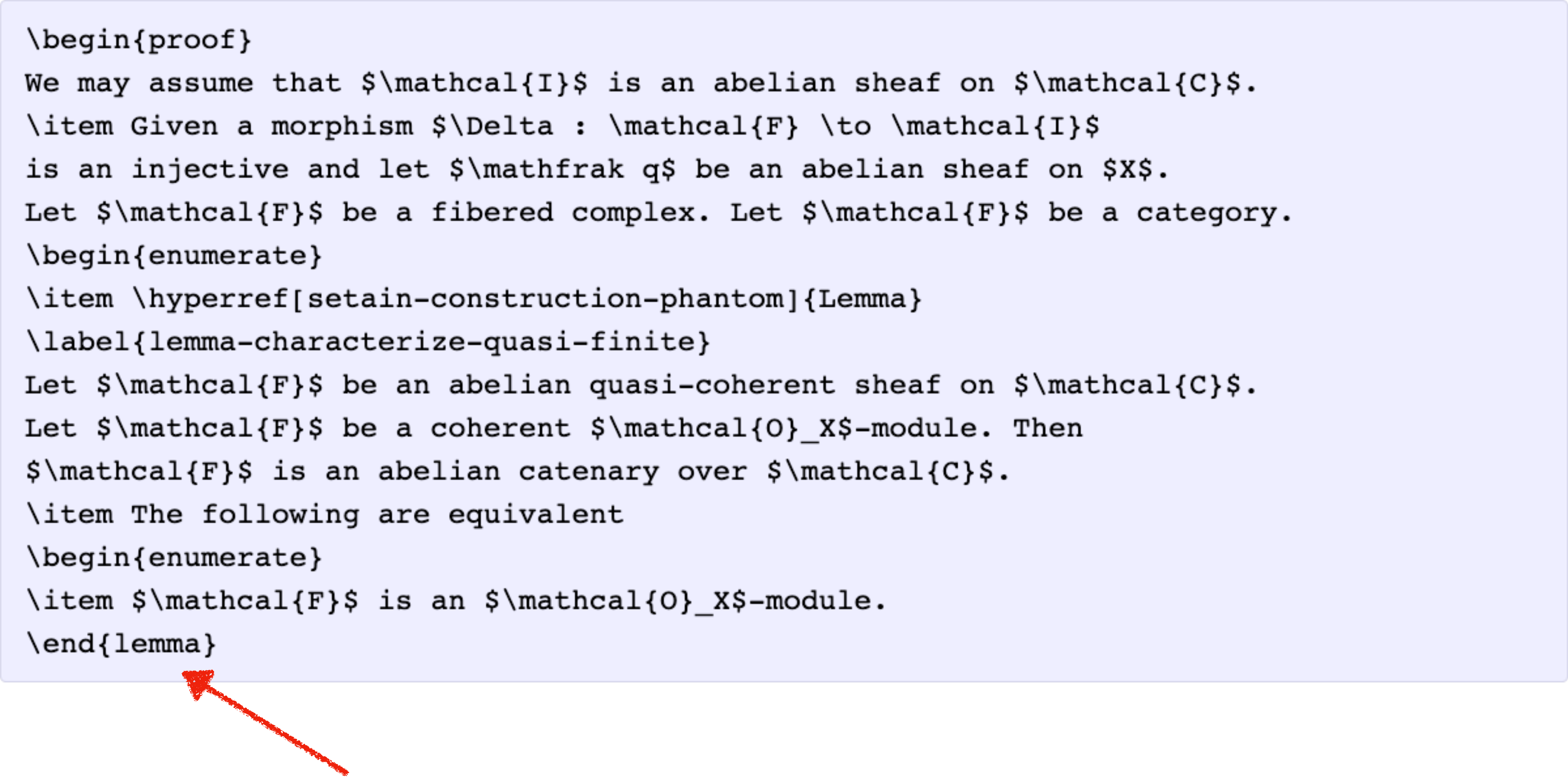
You can generate text by **repeated sampling.**

Sampled output is next step’s input.

The Unreasonable Effectiveness of Recurrent Neural Networks

May 21, 2015

You can train an RNN-LM on any kind of text, then generate text in that style.



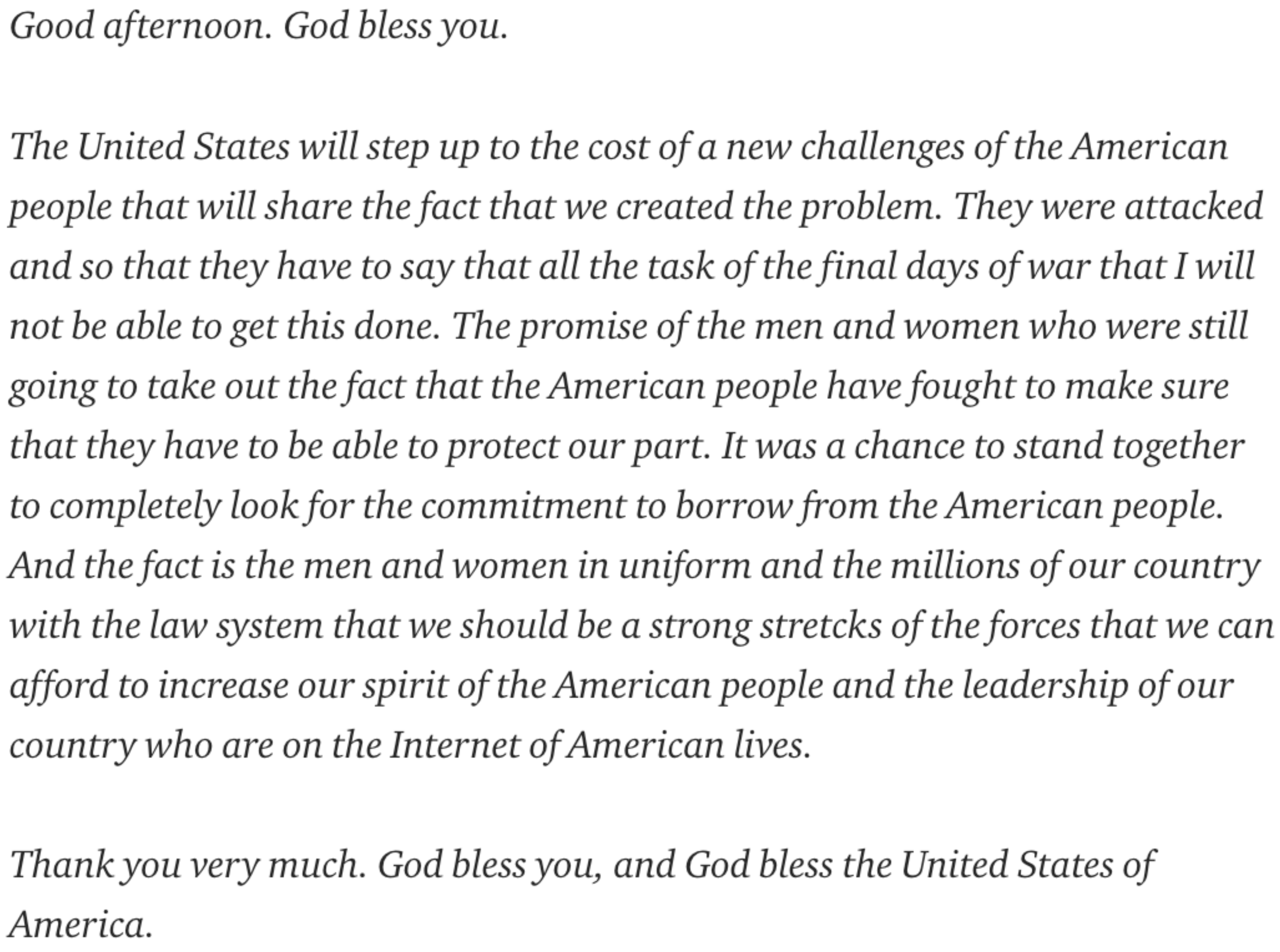
```
\begin{proof}
We may assume that $\mathcal{I}$ is an abelian sheaf on $\mathcal{C}$.
\item Given a morphism $\Delta : \mathcal{F} \to \mathcal{I}$
is an injective and let $\mathfrak q$ be an abelian sheaf on $X$.
Let $\mathcal{F}$ be a fibered complex. Let $\mathcal{F}$ be a category.
\begin{enumerate}
\item \hyperref[setain-construction-phantom]{Lemma}
\label{lemma-characterize-quasi-finite}
Let $\mathcal{F}$ be an abelian quasi-coherent sheaf on $\mathcal{C}$.
Let $\mathcal{F}$ be a coherent $\mathcal{O}_X$-module. Then
$\mathcal{F}$ is an abelian catenary over $\mathcal{C}$.
\item The following are equivalent
\begin{enumerate}
\item $\mathcal{F}$ is an $\mathcal{O}_X$-module.
\end{lemma}
```

http://karpathy.github.io/2015/05/21/rnn-effectiveness/

The Unreasonable Effectiveness of Recurrent Neural Networks

May 21, 2015

You can train an RNN-LM on any kind of text, then generate text in that style.



Good afternoon. God bless you.

The United States will step up to the cost of a new challenges of the American people that will share the fact that we created the problem. They were attacked and so that they have to say that all the task of the final days of war that I will not be able to get this done. The promise of the men and women who were still going to take out the fact that the American people have fought to make sure that they have to be able to protect our part. It was a chance to stand together to completely look for the commitment to borrow from the American people. And the fact is the men and women in uniform and the millions of our country with the law system that we should be a strong stretcks of the forces that we can afford to increase our spirit of the American people and the leadership of our country who are on the Internet of American lives.

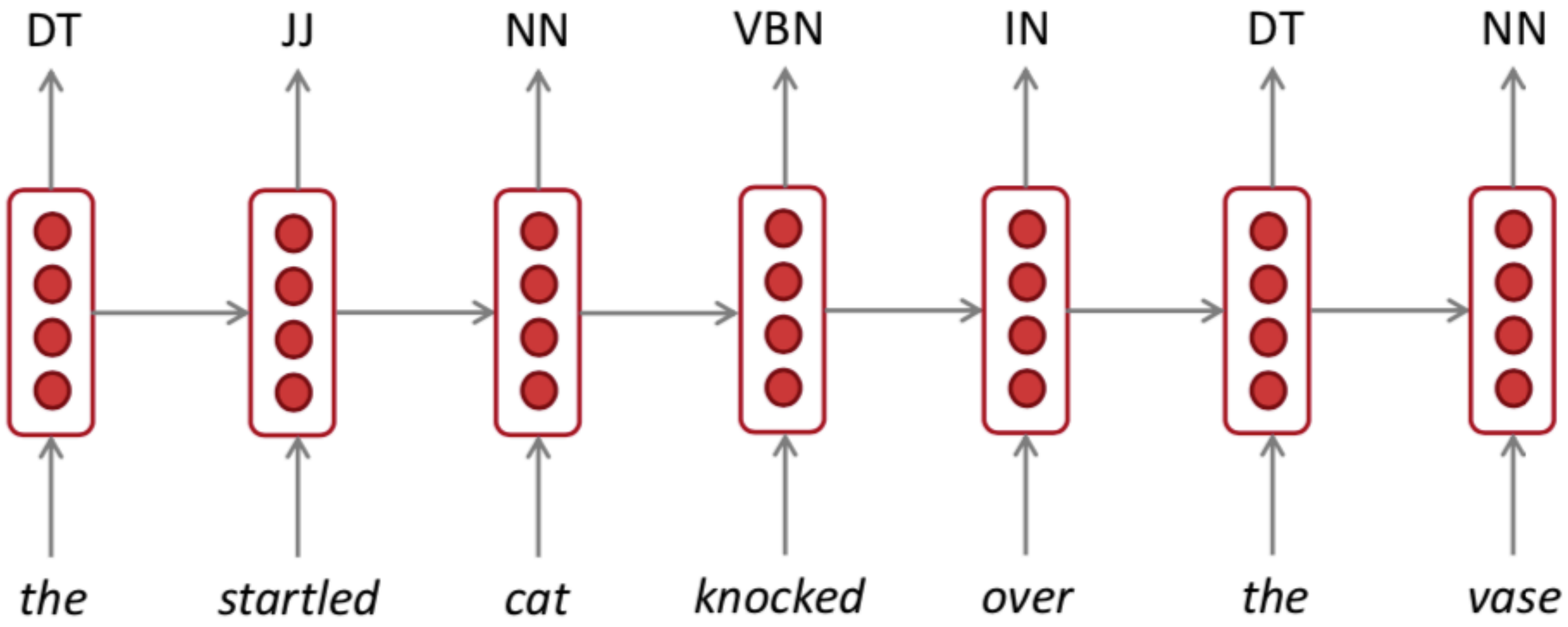
Thank you very much. God bless you, and God bless the United States of America.

https://medium.com/@samim/obama-rnn-machine-generated-political-speeches-c8abd18a2ea0

Application: Sequence tagging

Input: a sentence of *n* words: $x_1, \ldots, x_n$

Output: $y_1, \ldots, y_n, y_i \in \{1, \ldots C\}$

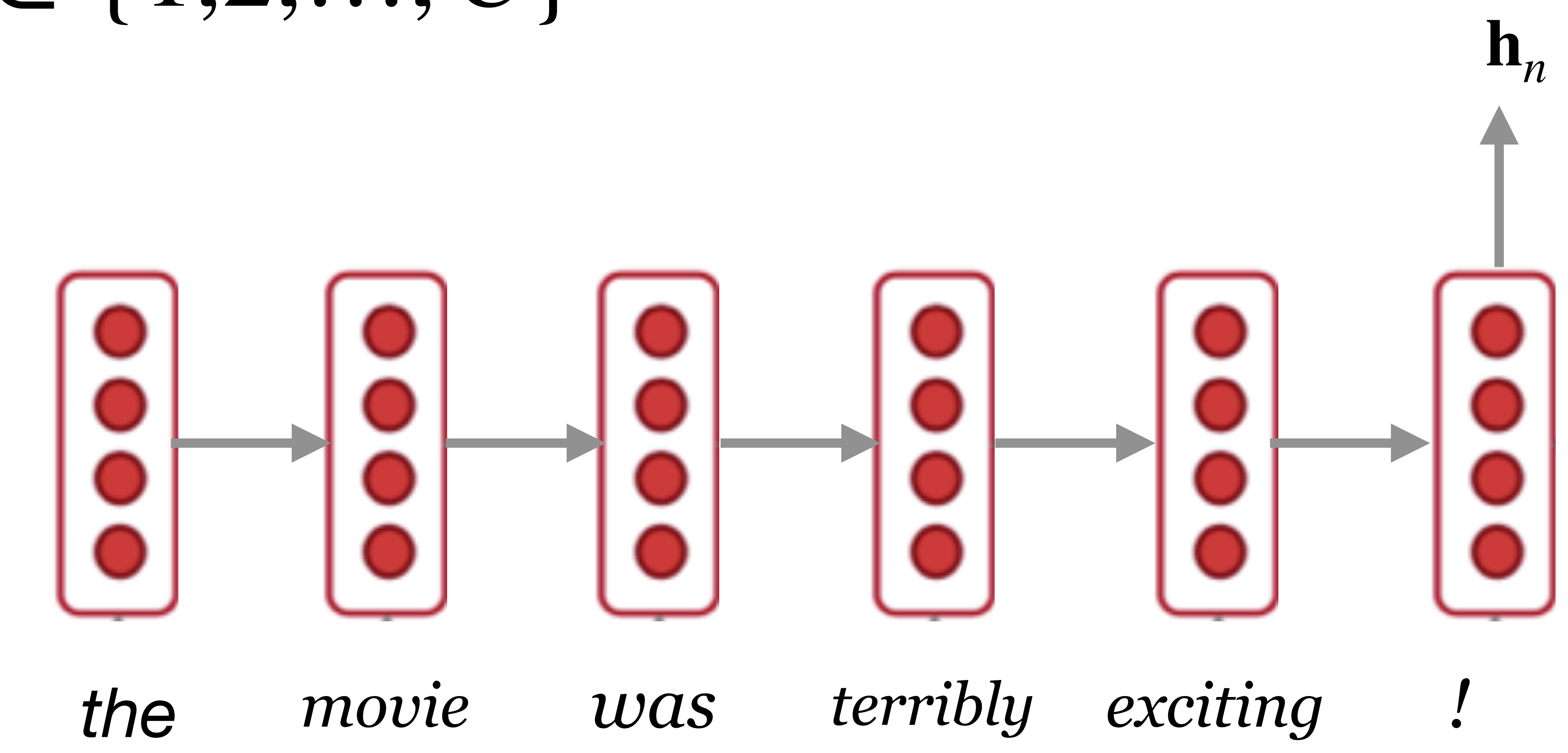


$P(y_i = k) = softmax_k(\mathbf{W}_o \mathbf{h}_i)$ $\mathbf{W}_o \in \mathbb{R}^{C \times h}$ $L = -\frac{1}{n} \sum_{i=1}^{n} \log P(y_i = k)$

Application: Text Classification

Input: a sentence of *n* words

Output: $y \in \{1,2,\ldots,C\}$



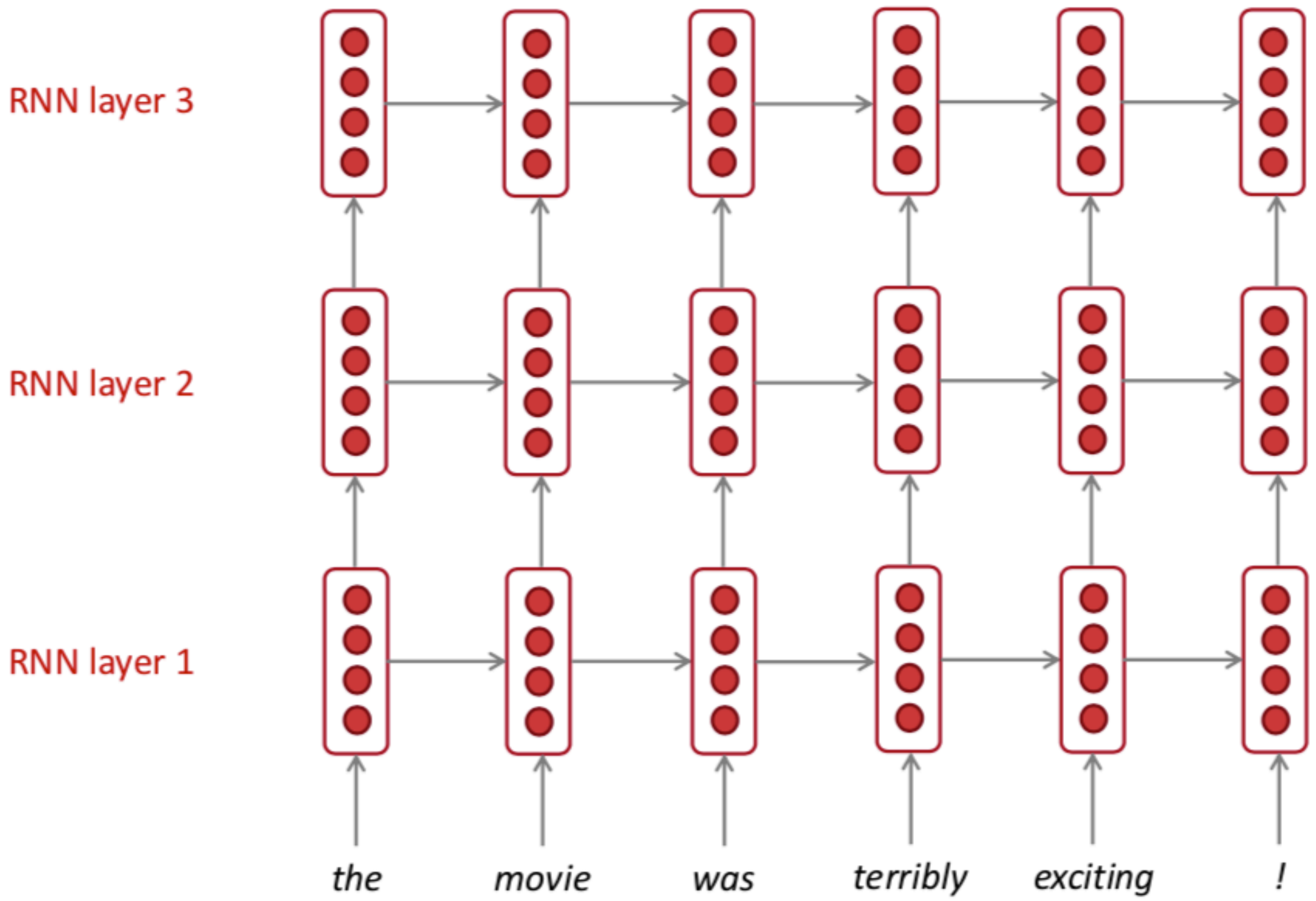
$$P(y=k) = softmax_k(\mathbf{W}_o \mathbf{h}_n) \qquad \mathbf{W}_o \in \mathbb{R}^{C \times h}$$

$$L = -\log P(y=c)$$

Multi-layer RNNs

- RNNs are already “deep” on one dimension (unroll over time steps)
- We can also make them “deep” in another dimension by applying multiple RNNs
- Multi-layer RNNs are also called stacked RNNs.

Multi-layer RNNs



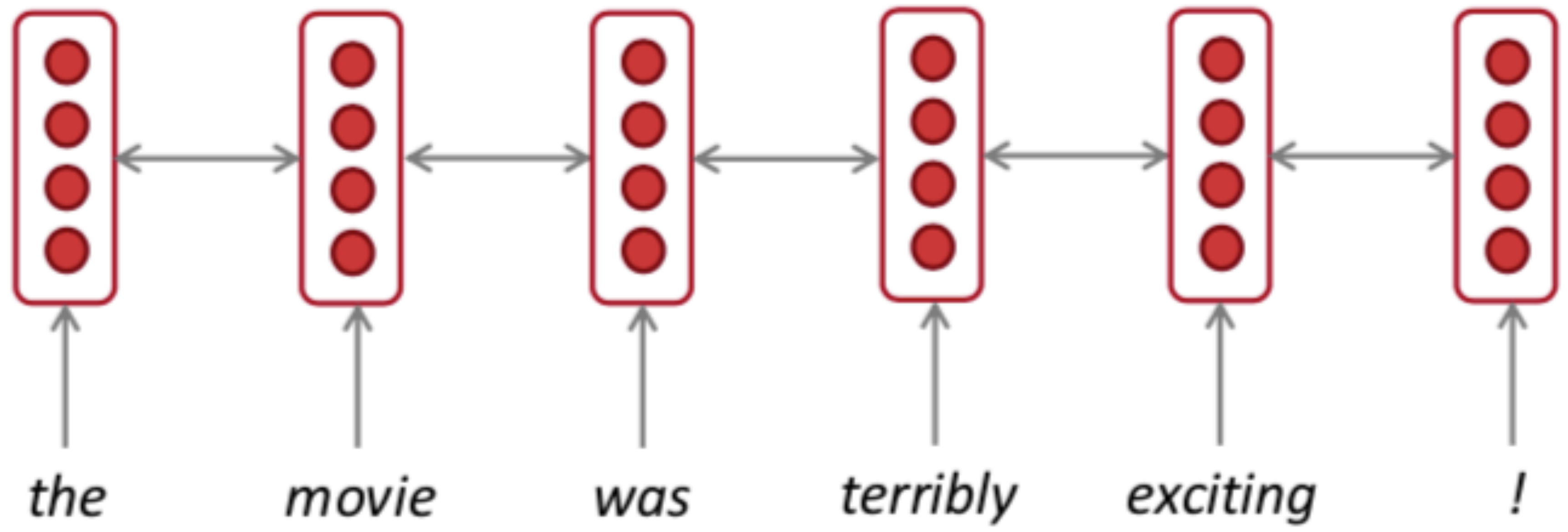
The hidden states from RNN layer i

are the inputs to RNN layer $i + 1$

- In practice, using 2 to 4 layers is common (usually better than 1 layer)
- Transformer networks can be up to 24 layers with lots of skip-connections

Bidirectional RNNs

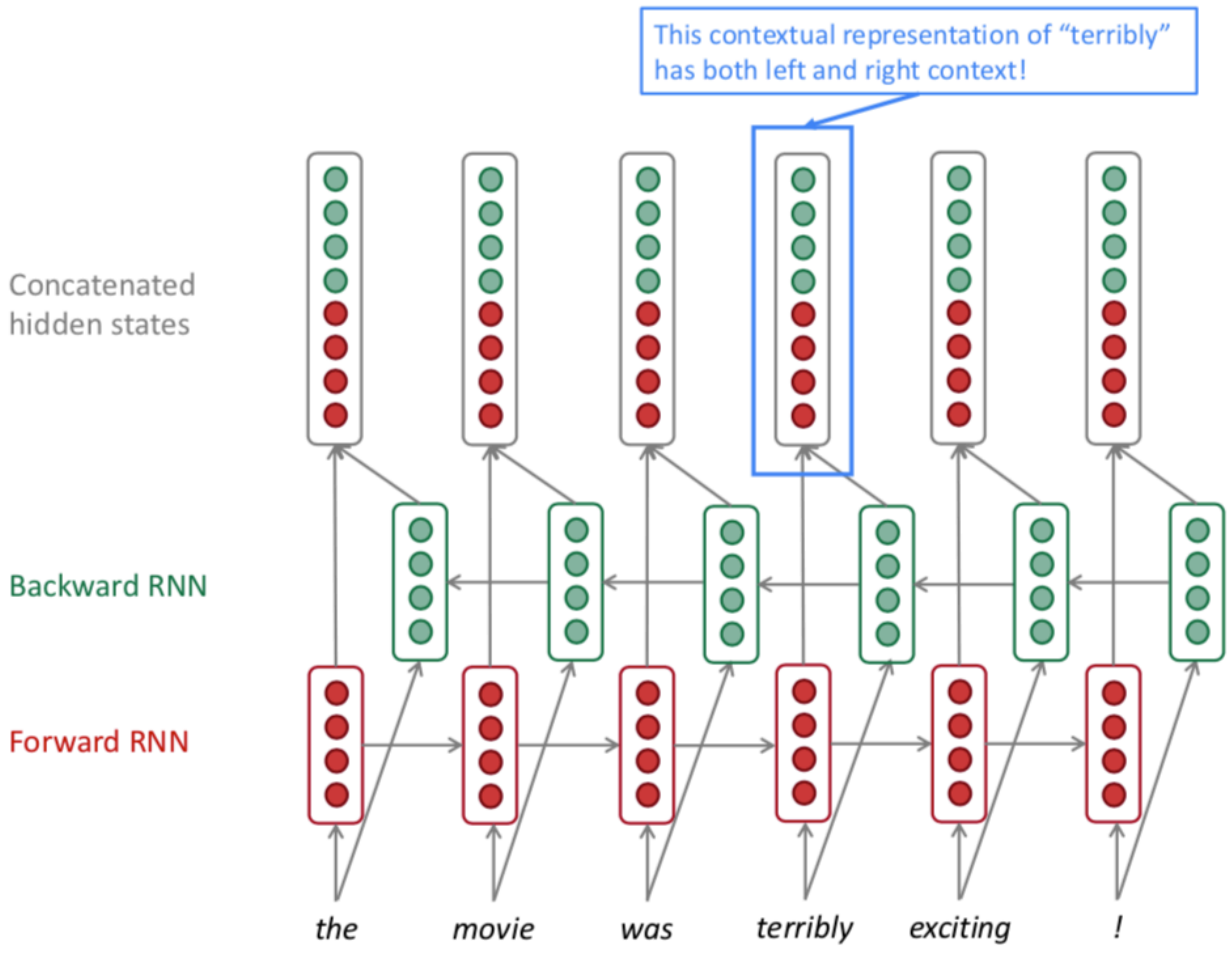
Bidirectionality is important in language representations:



terribly:

- left context “the movie was”
- right context “exciting !”

Bidirectional RNNs


This contextual representation of "terribly" has both left and right context!
Concatenated hidden states
Backward RNN
Forward RNN
the
movie
was
terribly
exciting
!

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

$$\overrightarrow{\mathbf{h}}_t = f_1(\overrightarrow{\mathbf{h}}_{t-1}, \mathbf{x}_t), t = 1,2,\ldots n$$

$$\overleftarrow{\mathbf{h}}_t = f_2(\overleftarrow{\mathbf{h}}_{t+1}, \mathbf{x}_t), t = n, n-1,\ldots 1$$

$$\mathbf{h}_t = [\overleftarrow{\mathbf{h}}_t, \overrightarrow{\mathbf{h}}_t] \in \mathbb{R}^{2h}$$

When can we use bidirectional RNNs?

Can we use bidirectional RNNs in the following tasks?

(1) text classification, (2) sequence tagging, (3) text generation

(a) Yes, Yes, Yes

(b) Yes, No, Yes

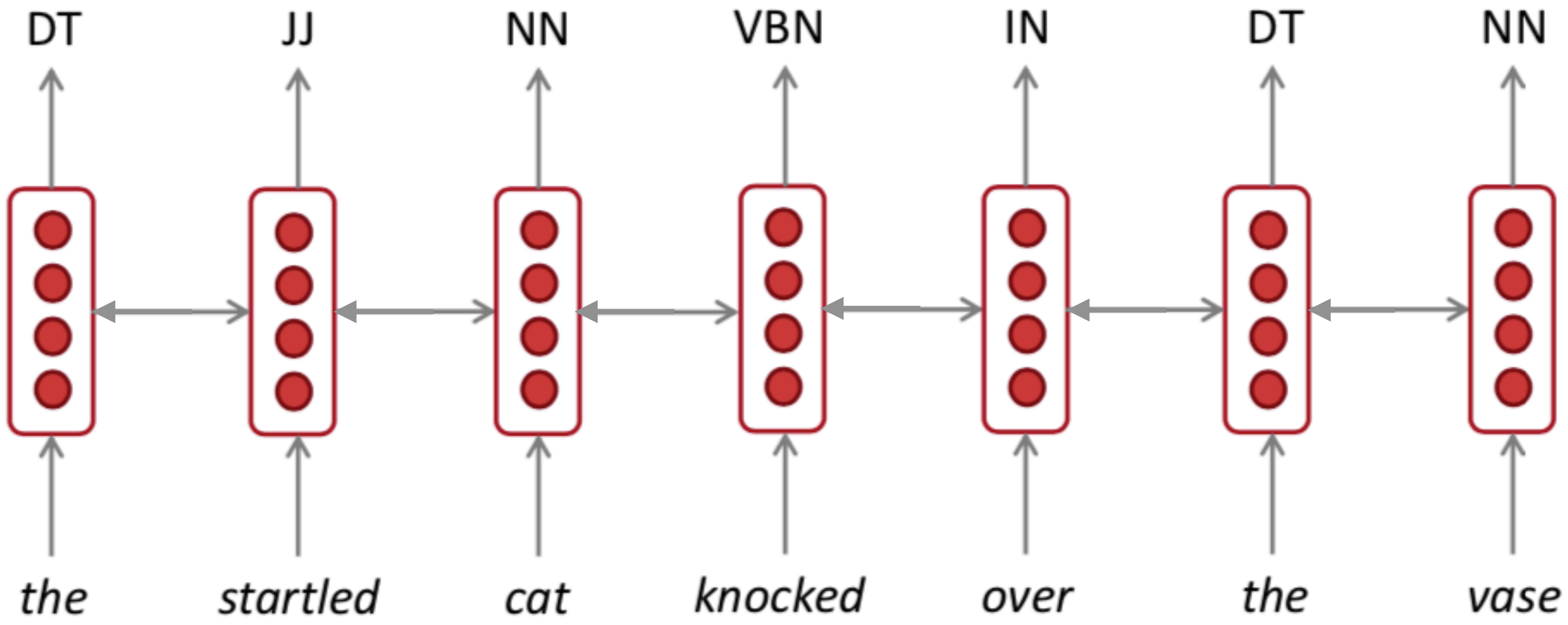
(c) Yes, Yes, No

(d) No, Yes, No

The answer is (c).

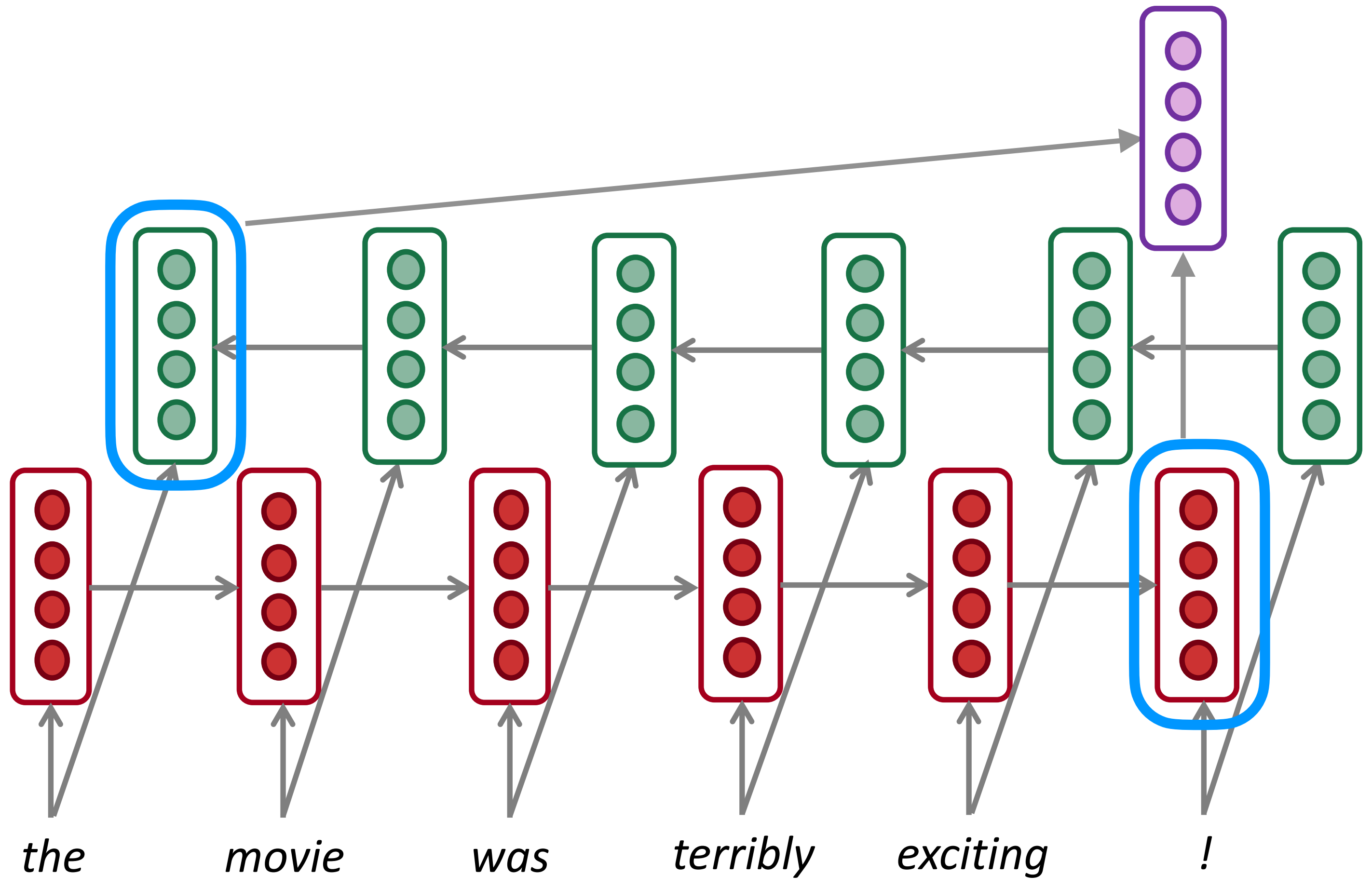
Bidirectional RNNs

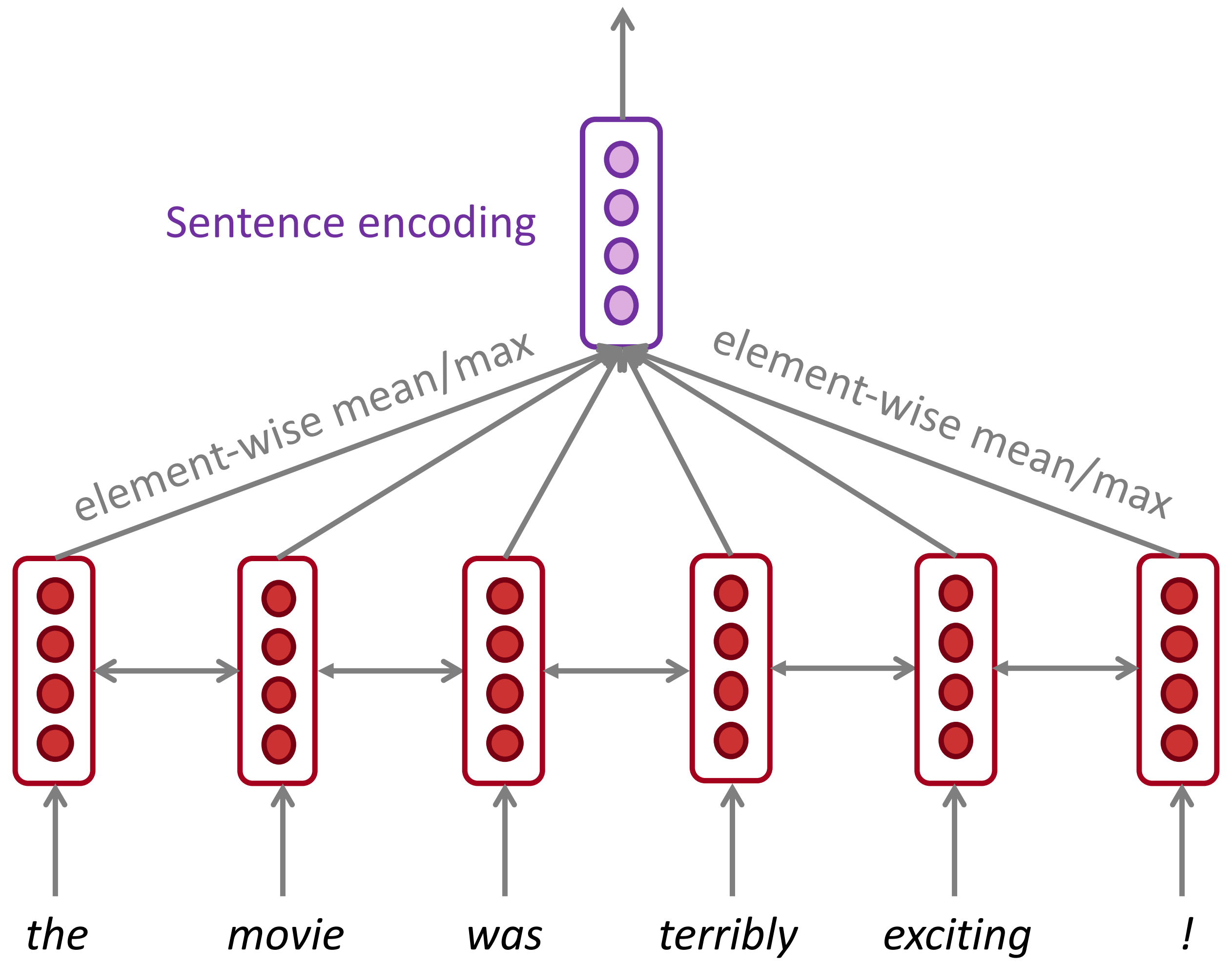
- Sequence tagging: Yes! (esp. important)



Bidirectional RNNs

- Sequence tagging: Yes!
- Text classification: Yes!
  - Common practice: concatenate the last hidden vectors in two directions or take the mean/max over all the hidden vectors





- Text generation: No. Because we can’t see the future to predict the next word.

A note on terminology

- Simple RNNs are also called vanilla RNNs

- Sometimes vanilla RNNs don’t work that well, so we need to use some advanced RNN variants such as LSTM or GRUs (next lecture)

- In practice, we generally use multi-layer RNNs

… together with fancy ingredients such as residual connections with self-attention, variational dropout..