COS 484/584

(Advanced) Natural Language Processing

L11: Recurrent Neural Networks

Spring 2021

(Some slides adapted from Chris Manning, Abigail See)

Announcements

- Midterm - Wednesday to Thursday (24 hours)
  - You can open the exam after Wed 12pm-Thu 9am (NOT later than that!!!)
  - This lecture will be included in the midterm

- Open book: you can have access to our course materials (lecture slides, readings, videos) and calculators. Not allowed to access the internet otherwise. You can prepare a cheatsheet if you want.

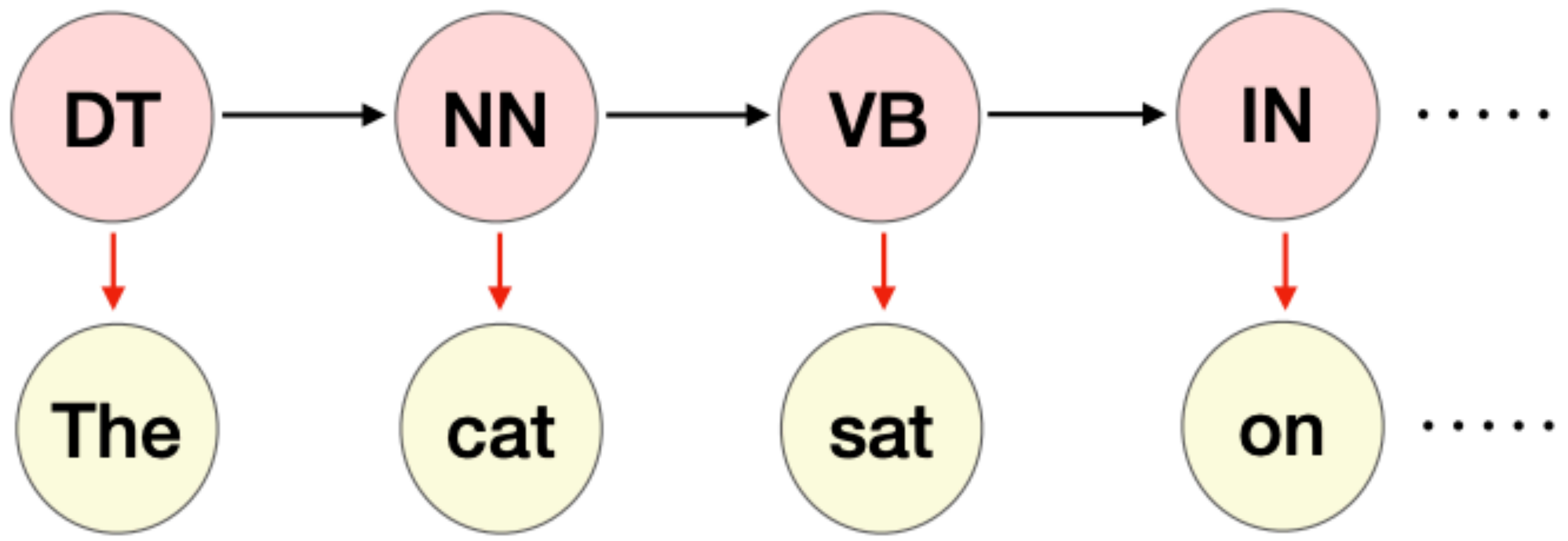
- Please don't use Ed during the exam period. If you have any questions, please write to cos484584.midterm@gmail.com (mention the problem ID in your email title)

- Please fill out your preference of exam time TODAY if you haven't:

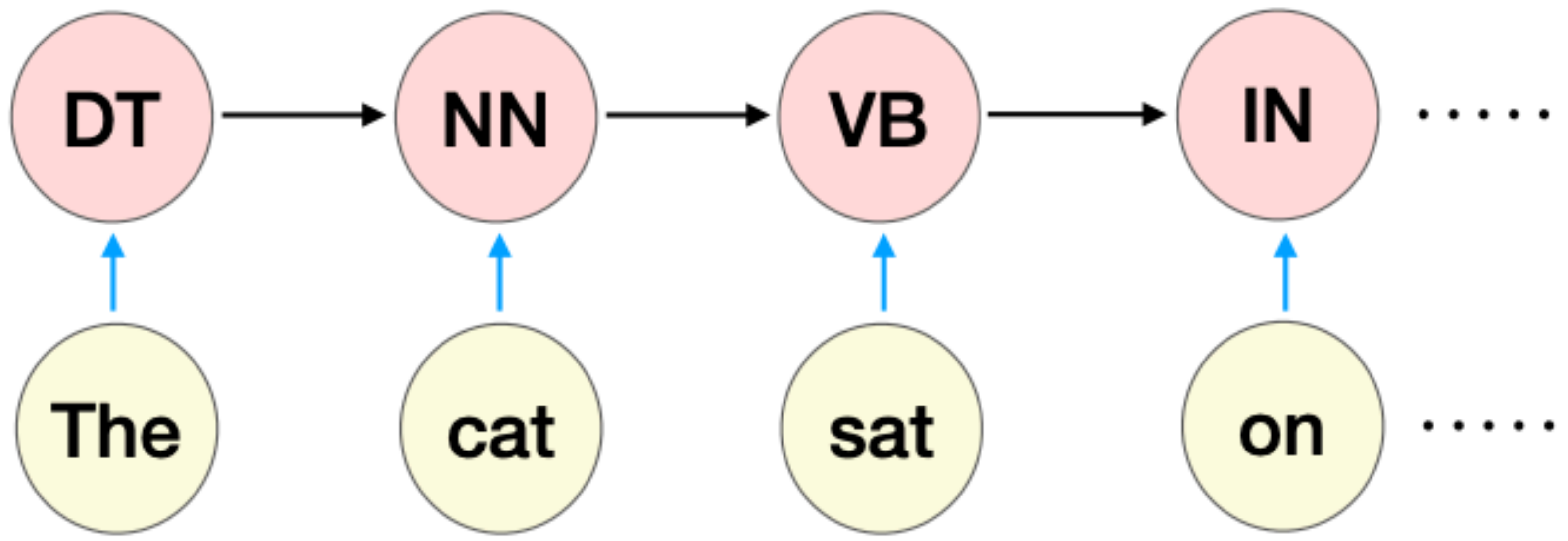
https://forms.gle/E5dpMct7Y5v9AUbN8

Recurrent neural networks (RNNs)

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



HMM



MEMM

- 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

Motivation: language models

n-gram language models

Sentence: "the cat sat on the mat"

$$
\begin{aligned}
P(\text{the cat sat on the mat}) = P(\text{the}) * P(\text{cat}|\text{the}) * P(\text{sat}|\text{the cat}) \\
* P(\text{on}|\text{the cat sat}) * P(\text{the}|\text{the cat sat on}) \\
* P(\text{mat}|\text{the cat sat on the})
\end{aligned}
$$

1st order

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

2nd order

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

k^{th} order Markov

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

Q: How do we know what size of k is needed?

Motivation: language models

n-gram language models

the students opened their ______

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

Q: Why can't we just keep a very large value of k?

Because it is too sparse to estimate the probabilities as k increases:

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

Motivation: language models

n-gram language models

Generate text with a 4-gram LM:

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 .

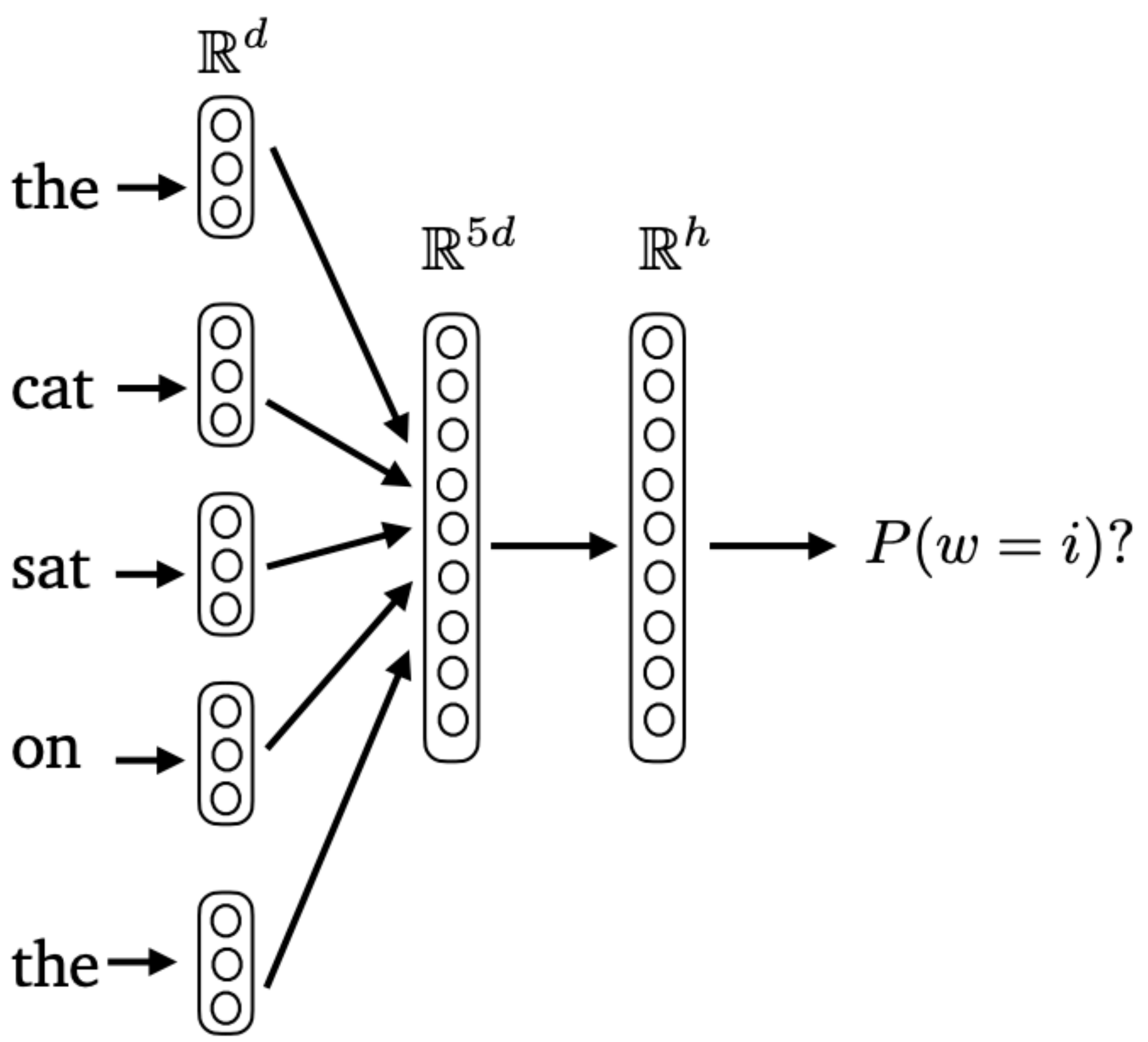
Surprisingly grammatical!

...but **incoherent.** We need to consider more than three words at a time if we want to model language well.

Motivation: language models

Feedforward neural language model

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



Previous n words = k-th order Markov assumption!

- Input layer (n = 5):

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

- Hidden layer

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

- Output layer (softmax)

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

$$P(w = i \mid \text{the cat sat on the}) = \text{softmax}_i(\mathbf{z}) = \frac{e^{z_i}}{\sum_k e^{z_k}}$$

Q: Why is this model still not good enough?

Motivation: language models

Feedforward neural language model

- Input layer (n = 5):

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

- Hidden layer

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

- $W \in \mathbb{R}^{h \times nd}$ scales with n
- The model learns separate patterns for the same item!

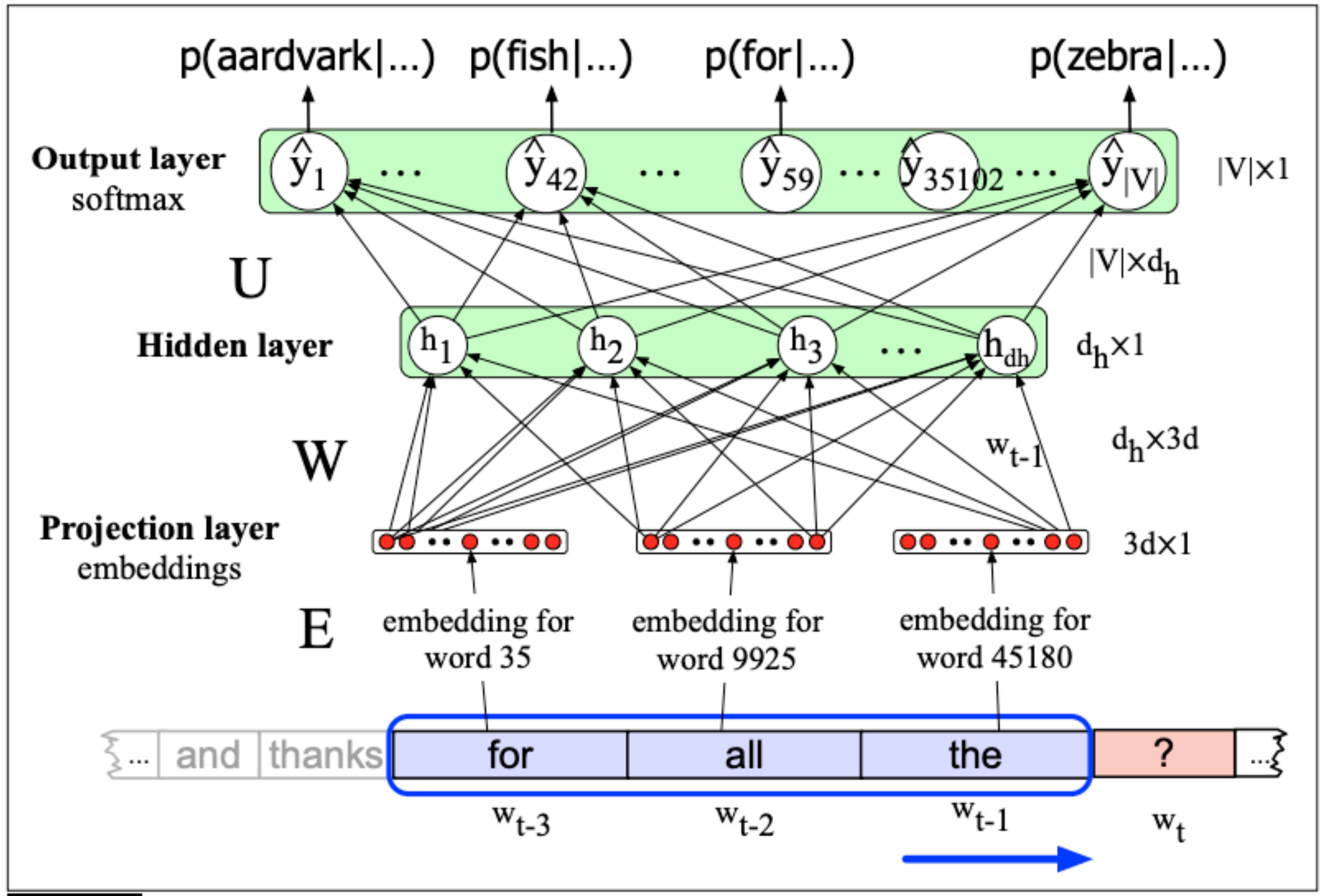


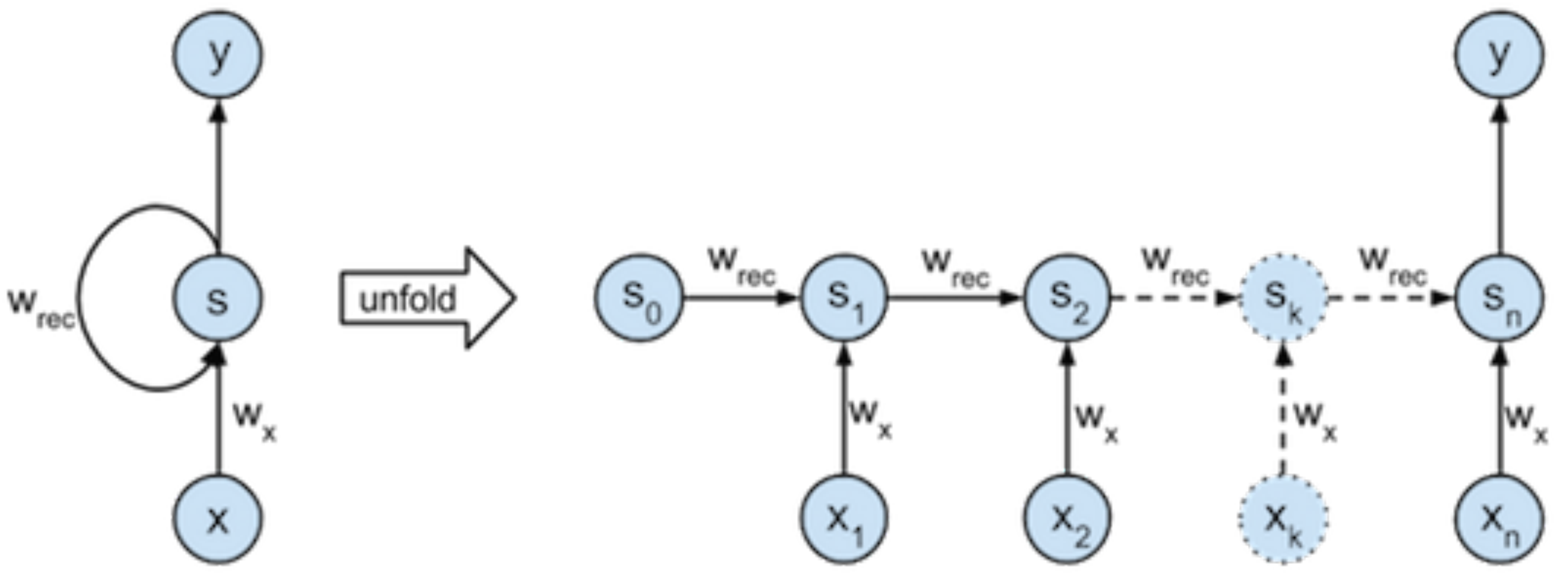
Figure 9.1 A simplified view of a feedforward neural language model moving through a text. At each time step t the network takes the 3 context words, converts each to a d-dimensional embedding, and concatenates the 3 embeddings together to get the $1 \times Nd$ unit input layer x for the network. The 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 are recurrent neural networks?

Recurrent neural networks (RNNs)

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



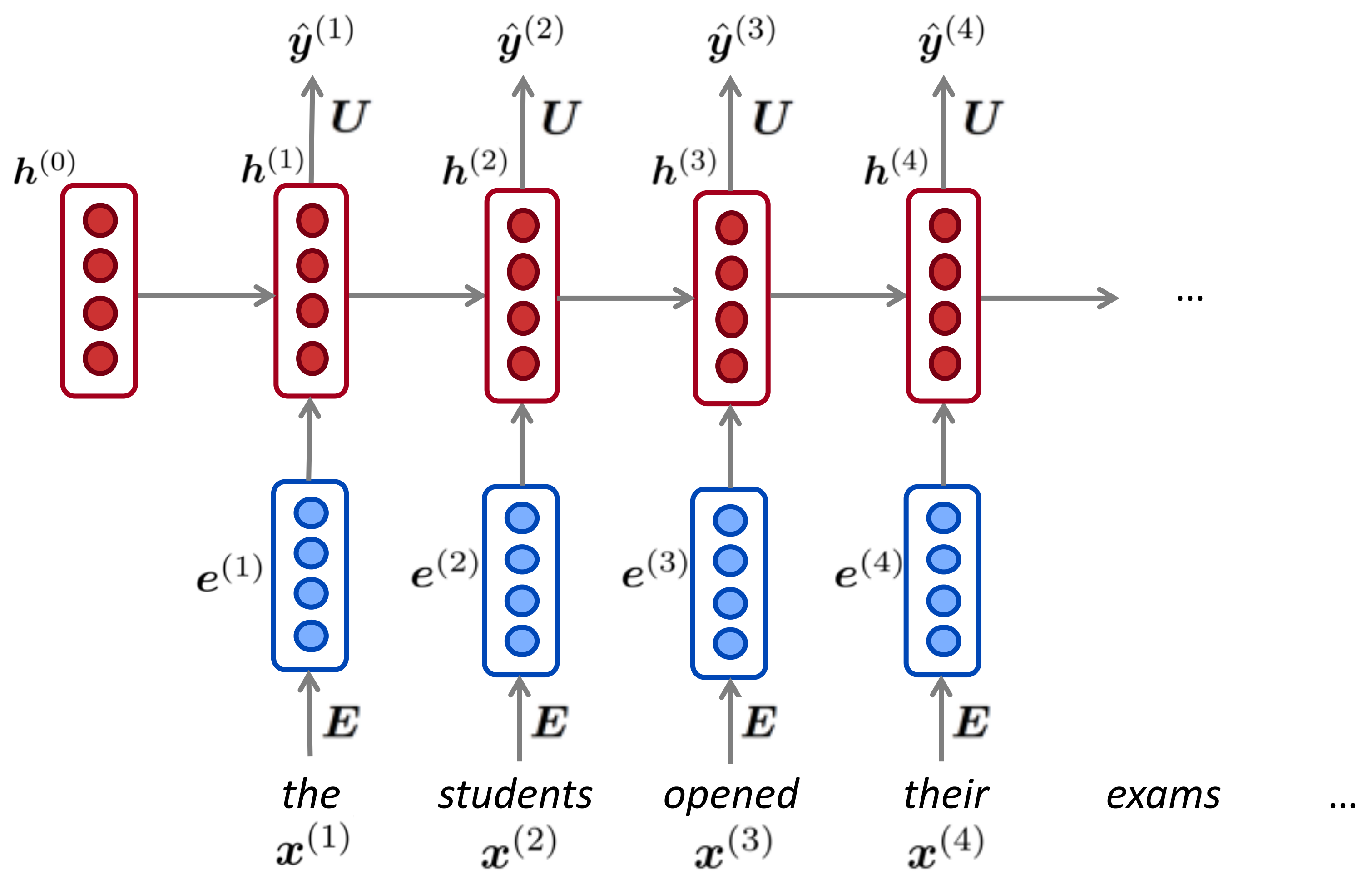
A function: $y = \text{RNN}(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n) \in \mathbb{R}^h$ where $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^d$

Core idea: apply the same weights repeatedly at different positions

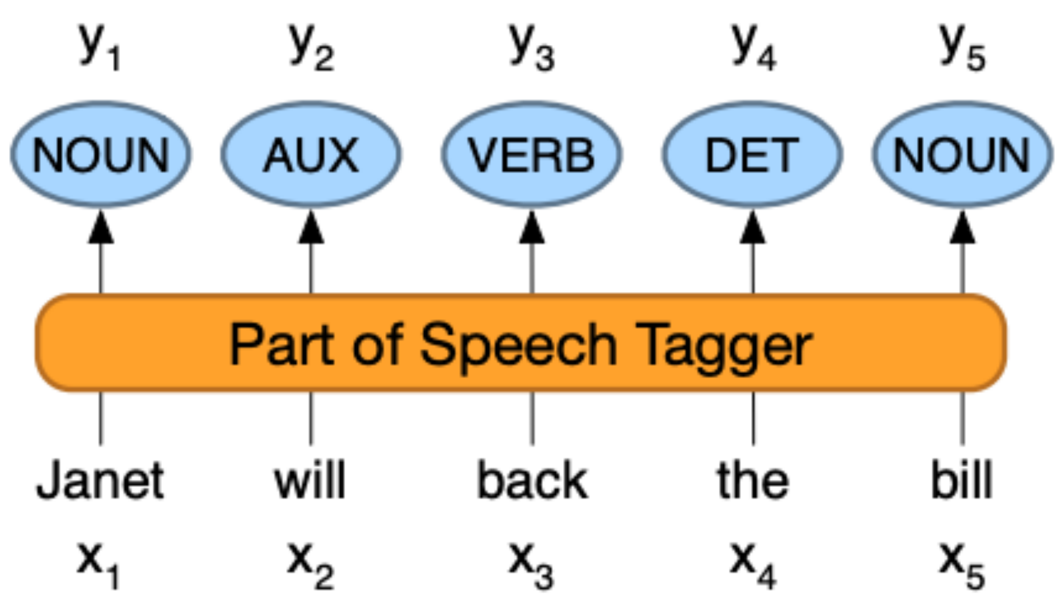
Recurrent neural networks (RNNs)

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

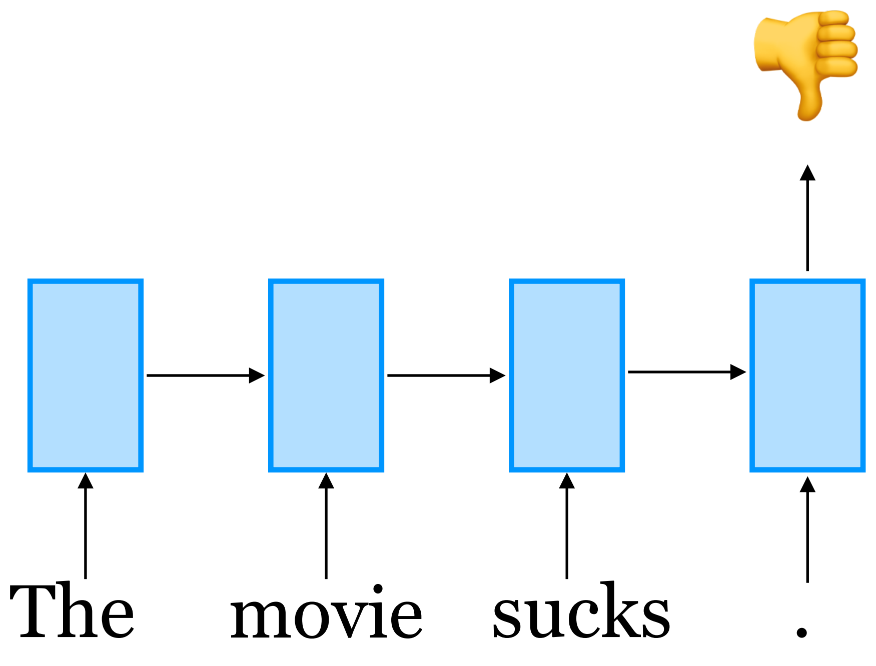
Language modeling



Sequence tagging



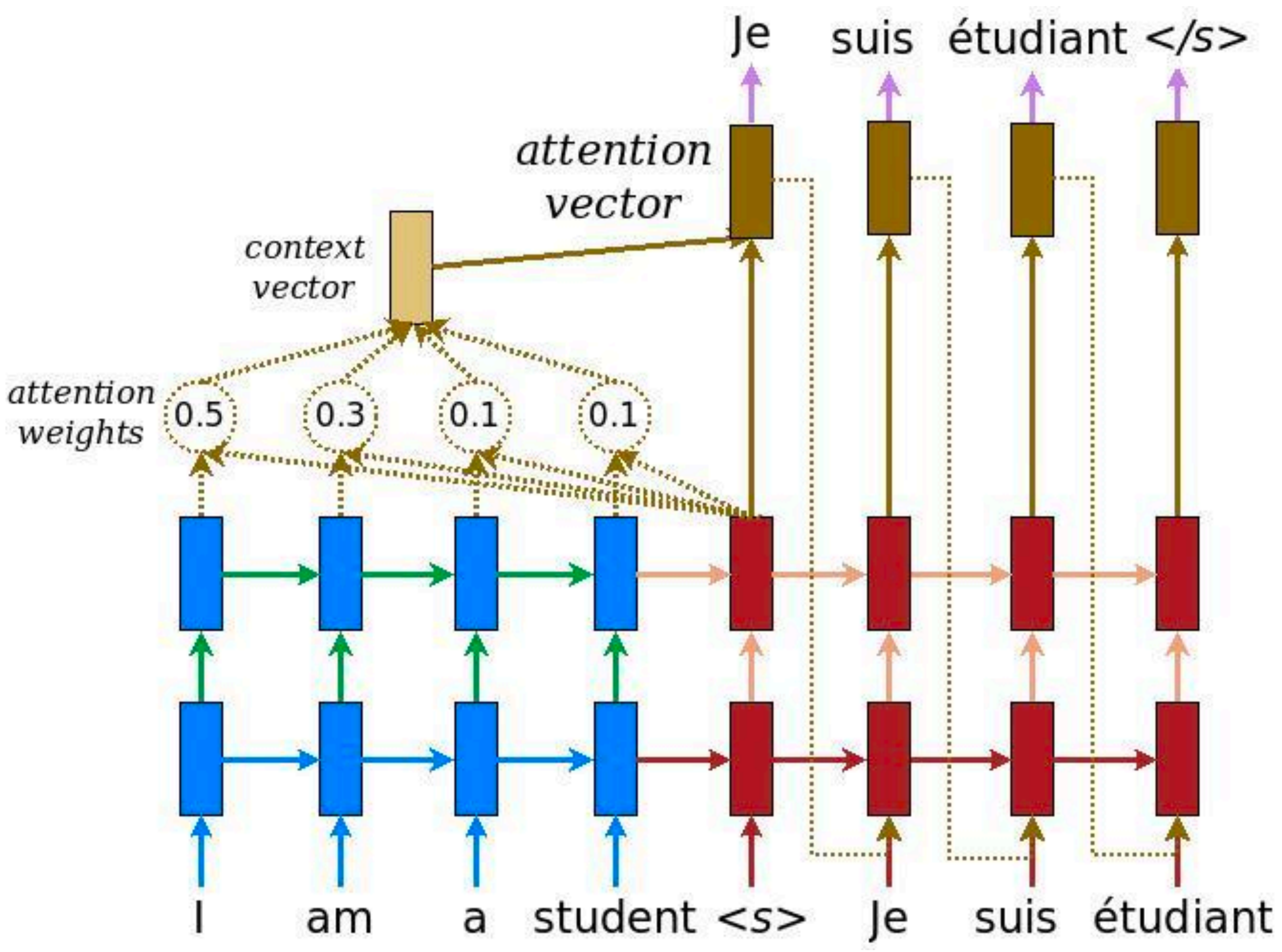
Text classification



Recurrent neural networks (RNNs)

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

sequence-to-sequence models



Simple recurrent neural networks

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

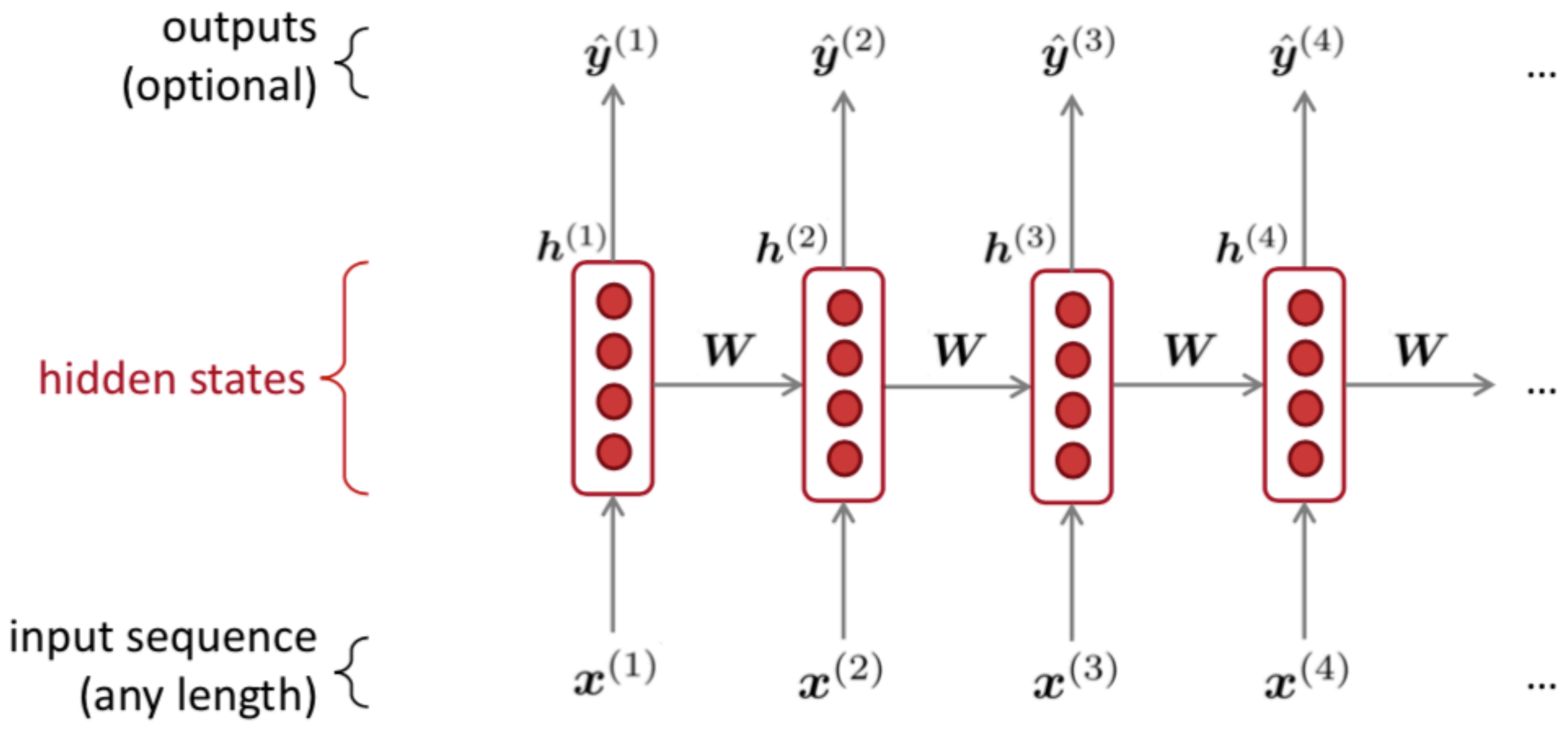
$\mathbf{W} \in \mathbb{R}^{h \times h}, \mathbf{U} \in \mathbb{R}^{h \times d}, \mathbf{b} \in \mathbb{R}^h$

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 $\mathbf{0}$)

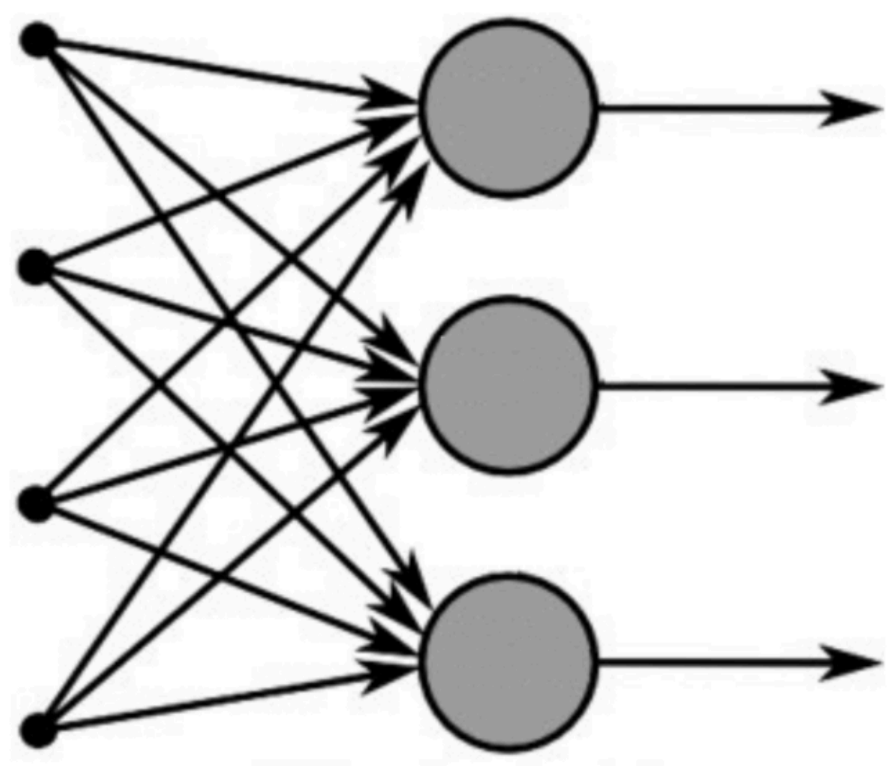
Simple recurrent neural networks

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

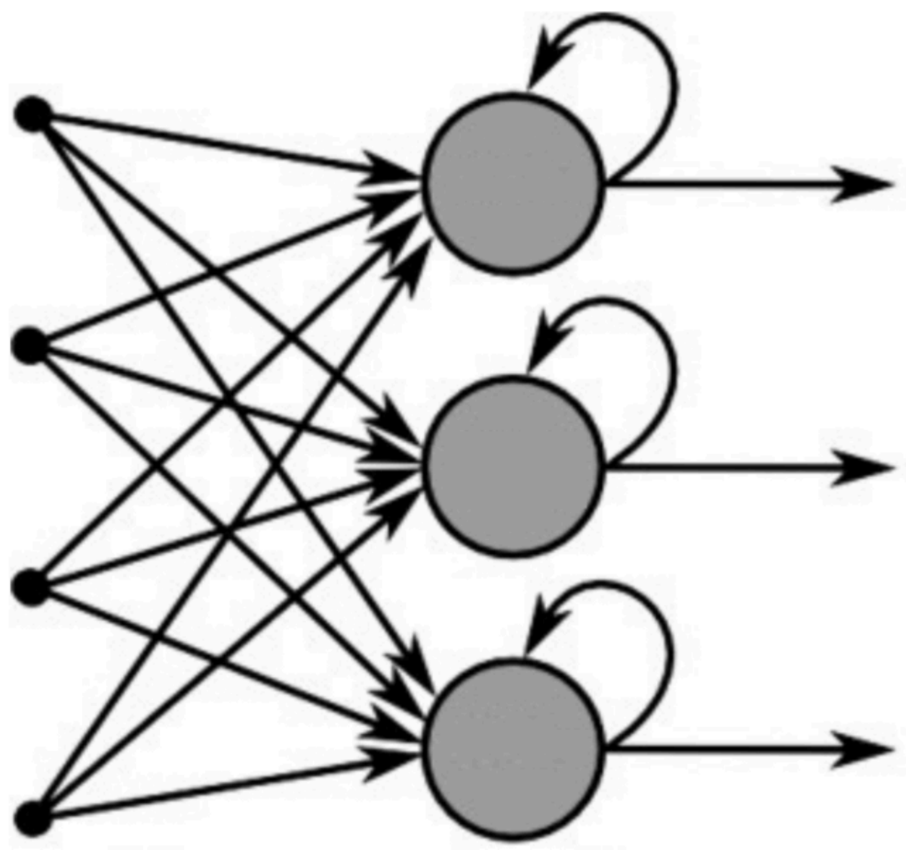
Key idea: apply the same weights $\mathbf{W}, \mathbf{U}, \mathbf{b}$ repeatedly



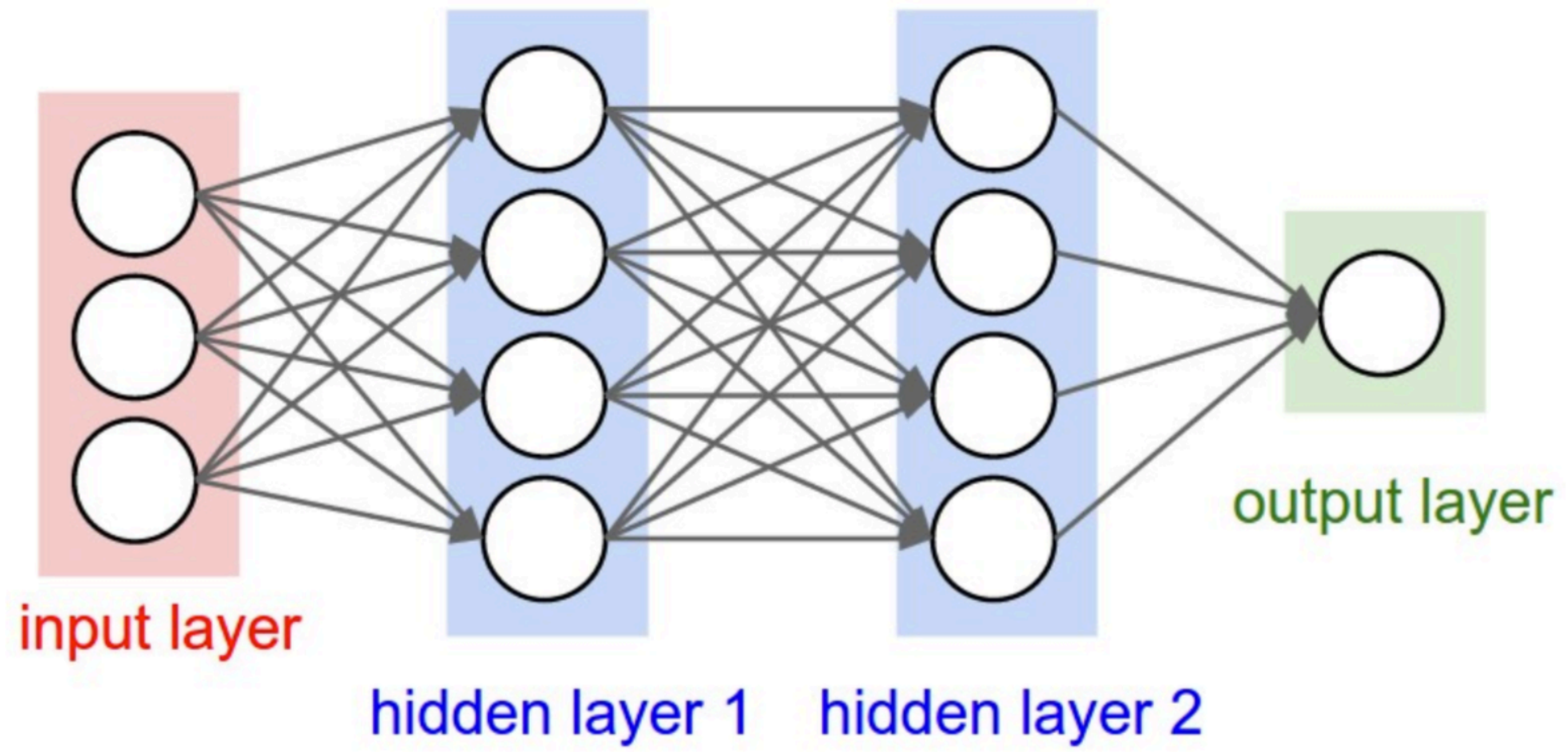
RNNs vs Feedforward NNs



Feed-Forward Neural Network



Recurrent Neural Network



Recurrent Neural Language Models (RNNLMs)

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

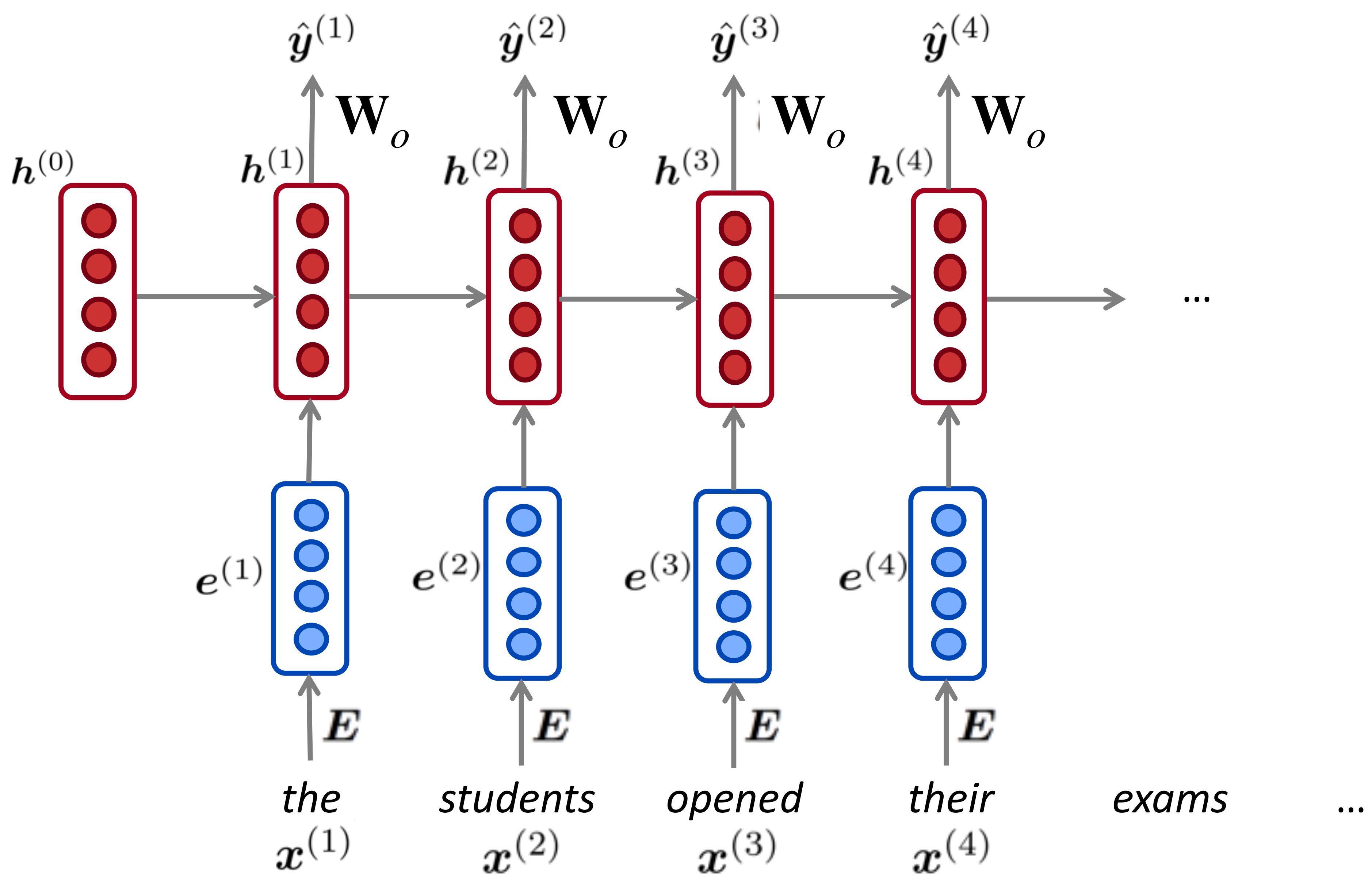
$$= P(w_1 \mid \mathbf{h}_0) \times P(w_2 \mid \mathbf{h}_1) \times P(w_3 \mid \mathbf{h}_2) \times \ldots \times P(w_n \mid \mathbf{h}_{n-1})$$

No Markov assumption here!

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

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

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



Progress on language models

On the Penn Treebank (PTB) dataset

Metric: perplexity

$$\mathrm{ppl}(S) = 2^x \quad \text{where}$$
$$x = -\frac{1}{W}\sum_{i=1}^{n} \log_2 P(S^i)$$

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

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**

We will talk about LSTMs later

(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) ⟵ We will learn Transformer networks!
- 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 $t = 1, 2, \ldots, n$

$$y = -\log \mathrm{softmax}(\mathbf{W}_o \mathbf{h}_{t-1})(w_t)$$

$$\mathbf{x}_t = e(w_t)$$

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

$$L = L + \frac{1}{n} y$$

Zoom poll

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 $t = 1, 2, \ldots, n$

$$y = -\log \mathrm{softmax}(\mathbf{W}_o \mathbf{h}_{t-1})(w_t)$$

$$\mathbf{x}_t = e(w_t)$$

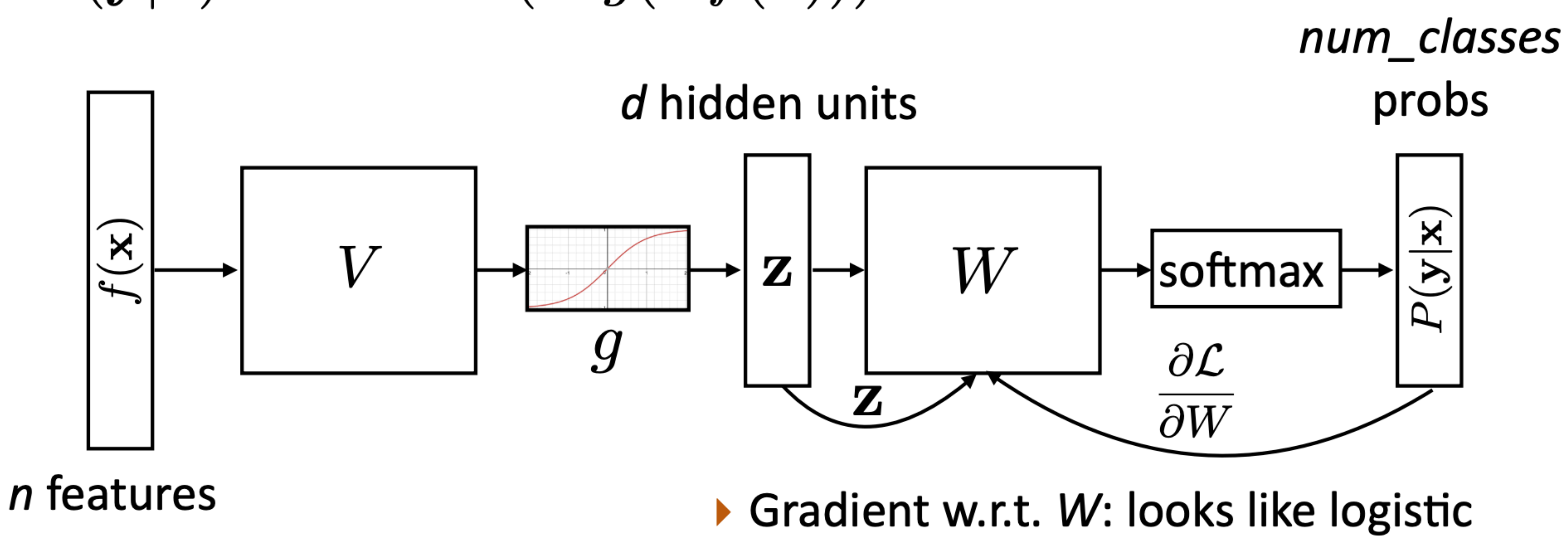
$$\mathbf{h}_t = g(\mathbf{W}\mathbf{h}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b})$$

$$L = L + \frac{1}{n} y$$

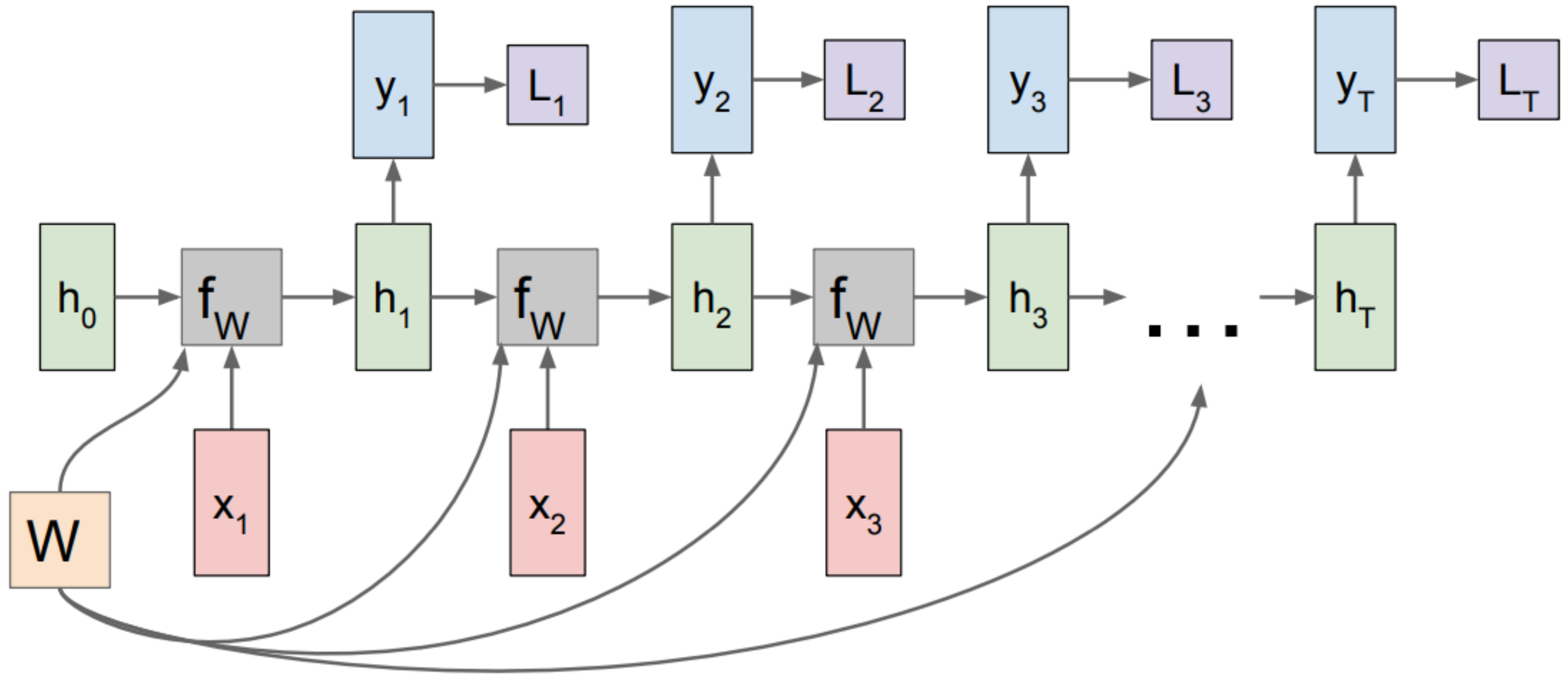
Training RNNLMs

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

$P(\mathbf{y}|\mathbf{x}) = \text{softmax}(W g(V f(\mathbf{x})))$



- Gradient w.r.t. *W*: looks like logistic regression, can be computed treating **z** as the features



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

Backpropagation through time [advanced]

$$\mathbf{h}_1 = g(\mathbf{W}\mathbf{h}_0 + \mathbf{U}\mathbf{x}_1 + \mathbf{b})$$
$$\mathbf{h}_2 = g(\mathbf{W}\mathbf{h}_1 + \mathbf{U}\mathbf{x}_2 + \mathbf{b})$$
$$\mathbf{h}_3 = g(\mathbf{W}\mathbf{h}_2 + \mathbf{U}\mathbf{x}_3 + \mathbf{b})$$

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

You should know how to compute: $\frac{\partial L_3}{\partial \mathbf{h}_3}$

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

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

If k and t are far away, the gradients are very easy to grow/shrink exponentially

(called the gradient exploding or gradient vanishing problem)

Zoom poll

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) None of the above

All of these are correct (a) (b) (c) ☺

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**
$\quad \hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$
end if

Backpropagation through time

Gradient vanishing is a harder problem to solve:

When she tried to print her tickets, she found that the printer was out of toner.
She went to the stationery store to buy more toner. It was very overpriced.
After installing the toner into the printer, she finally printed her ________

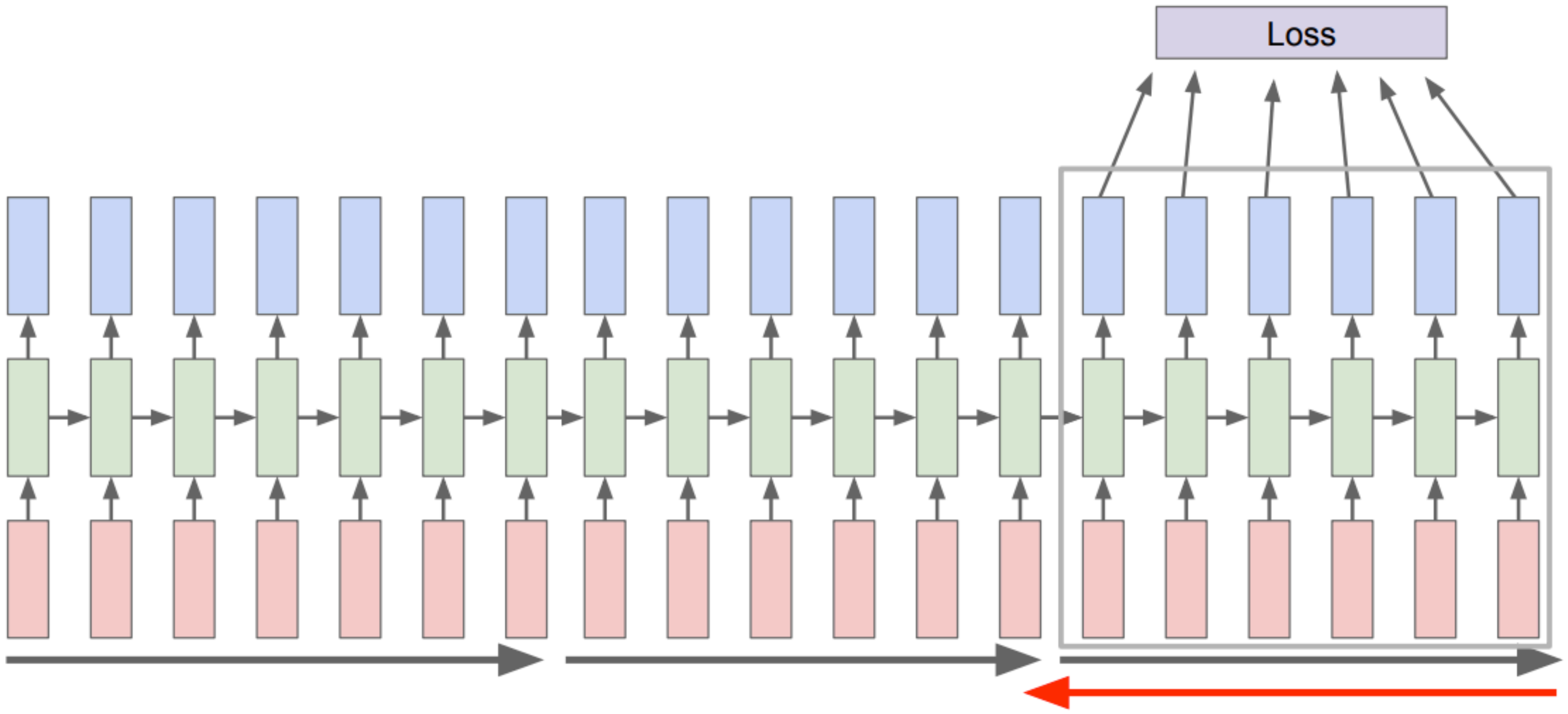
The writer of the books is/are (planning a sequel)

Syntactic recency: *The writer of the books is* (correct)

Sequential recency: *The writer of the books are* (incorrect)

Truncated backpropagation through time

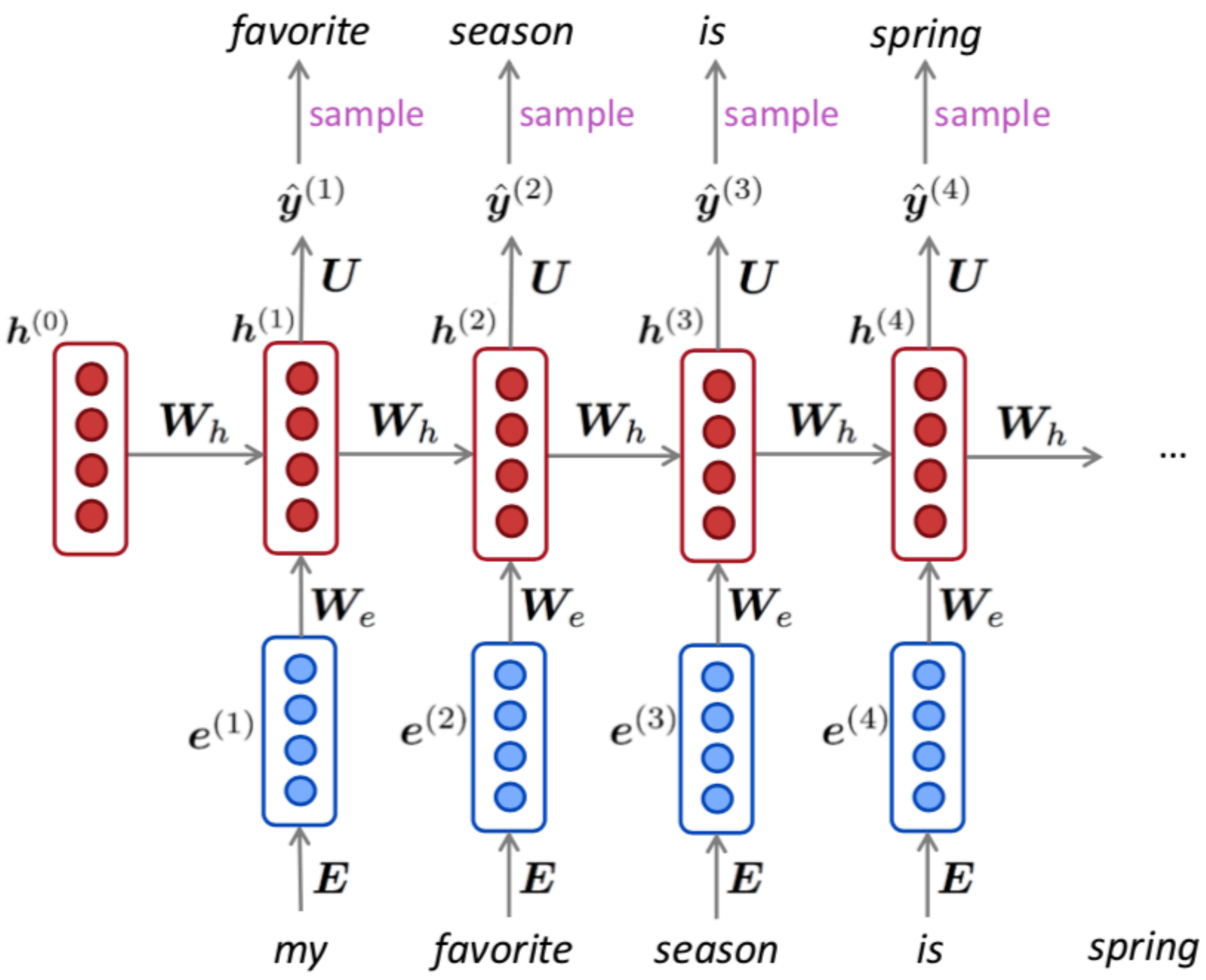
- Backpropagation is very expensive if you handle long sequences



- 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.

Let's have some fun

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

Let’s have some fun

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

“Sorry,” Harry shouted, panicking — “I’ll leave those brooms in London, are they?”

“No idea,” said Nearly Headless Nick, casting low close by Cedric, carrying the last bit of treacle Charms, from Harry’s shoulder, and to answer him the common room perched upon it, four arms held a shining knob from when the spider hadn’t felt it seemed. He reached the teams too.

“You believe if we’ve got friendly to come down and out of the library. I think I’ve found out Potter, I asked you he had . . . me. I think he’s not telling Dobby if yeh get with our Hogwarts …”

https://medium.com/deep-writing/harry-potter-written-by-artificial-intelligence-8a9431803da6

Let's have some fun

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

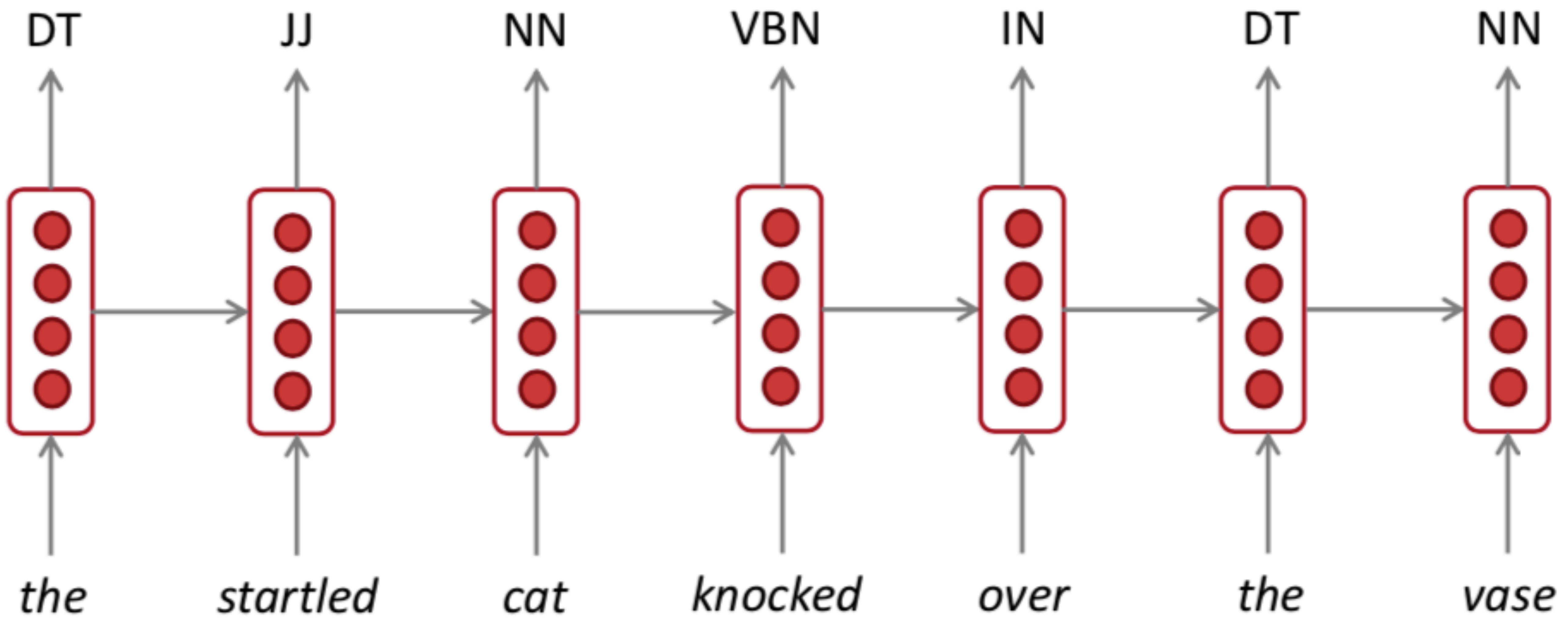
```
\begin{proof}
We may assume that $\mathcal{I}$ is an abelian sheaf on $\mathcal{C}$.
\item Given a morphism $\Delta : \mathcal{F} \to \mathcal{I}$
is an injective and let $\mathfrak q$ be an abelian sheaf on $X$.
Let $\mathcal{F}$ be a fibered complex. Let $\mathcal{F}$ be a category.
\begin{enumerate}
\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/

Application: Sequence Tagging

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

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



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

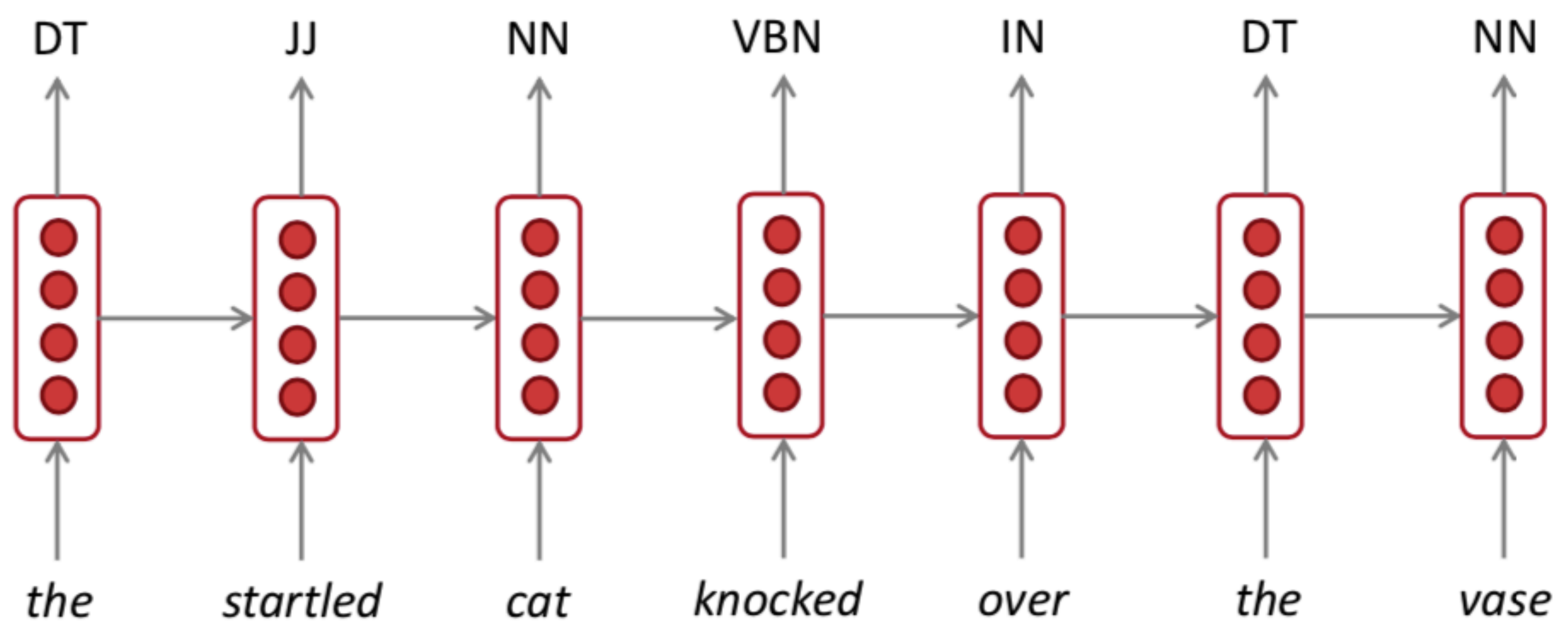
Q: How do we decode y_i at testing time?

Application: Sequence Tagging

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

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

The model doesn't model dependencies between output labels!



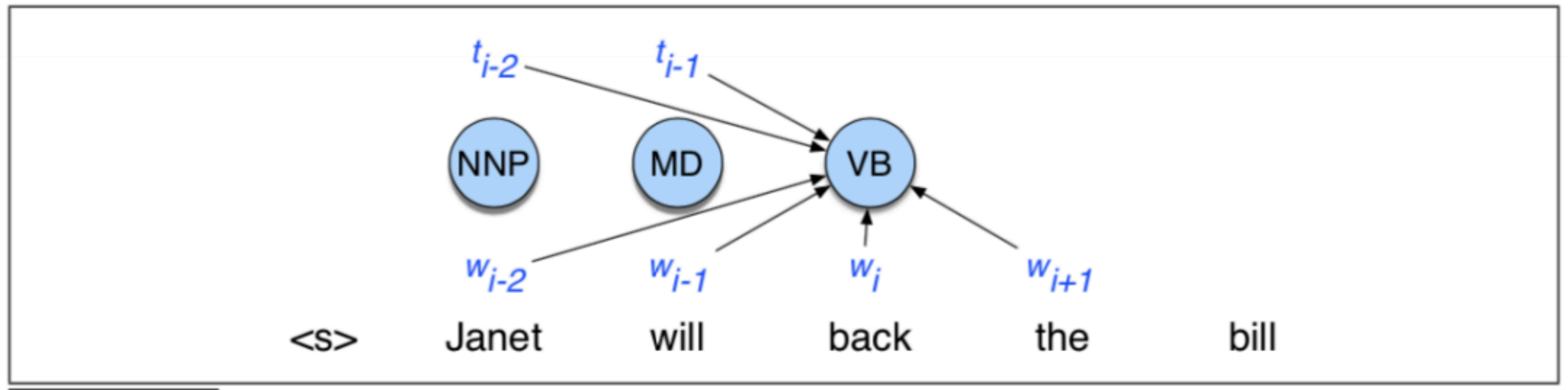


Figure 8.13 An MEMM for part-of-speech tagging showing the ability to condition on more features.

[advanced]

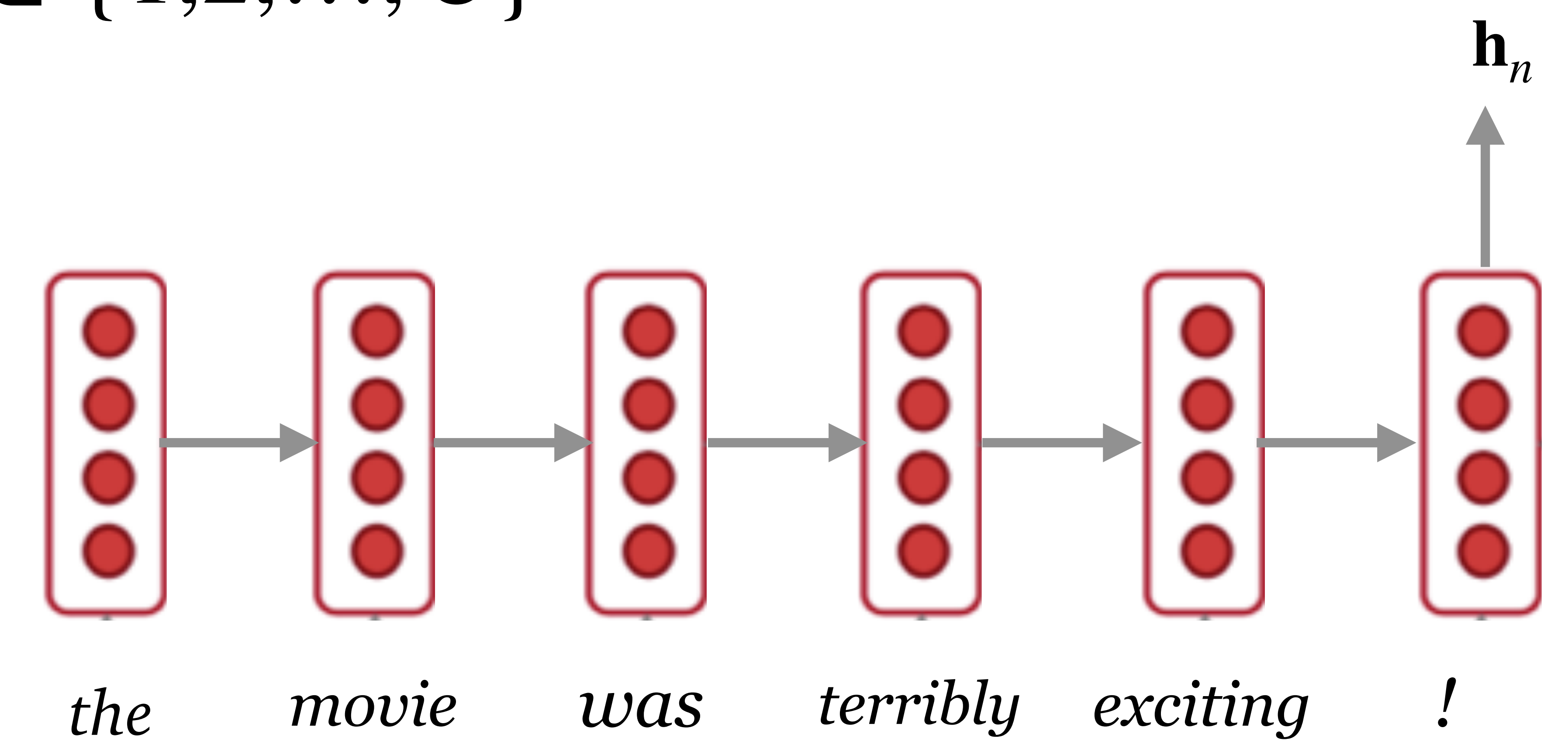
- We can still model the joint probabilities over $\{y_1, y_2, \ldots, y_n\}$ and use beam search at decoding time
- The main difference compared to MEMMs - you don't need to define manual features and the RNNs can derive features automatically!

(Lample et al, 2016): Neural Architectures for Named Entity Recognition

Application: Text Classification

Input: a sentence of n words

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



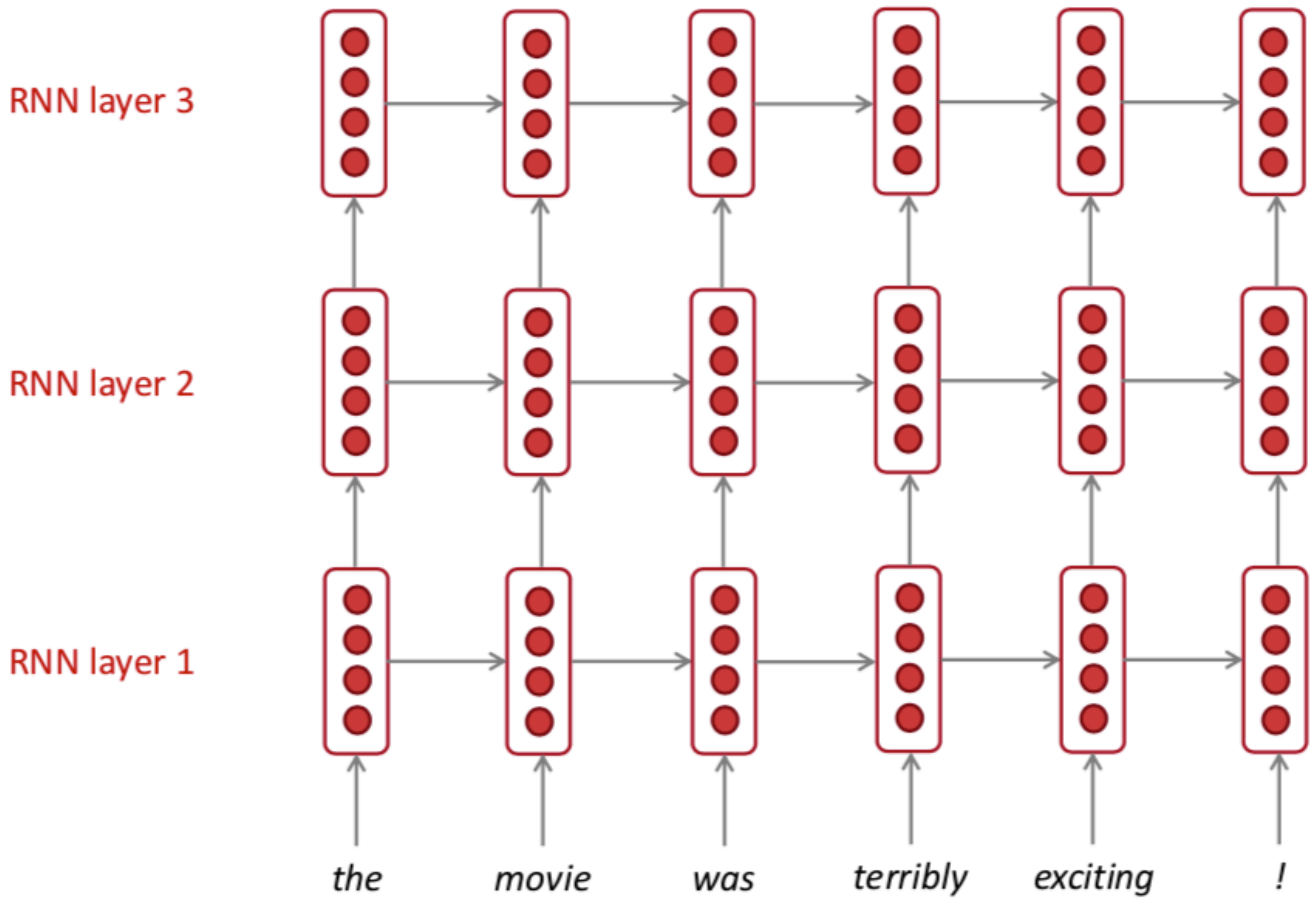
$$P(y = k) = softmax_k(\mathbf{W}_o \mathbf{h}_n) \qquad \mathbf{W}_o \in \mathbb{R}^{C \times h}$$

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

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

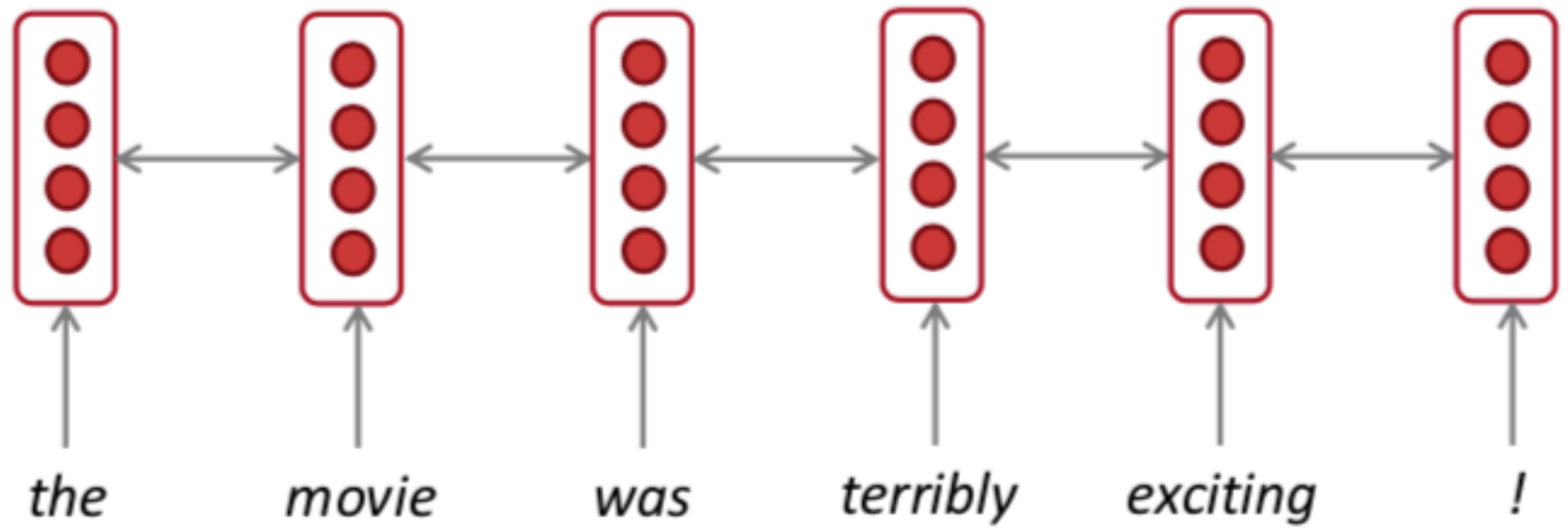


The hidden states from RNN layer i are the inputs to RNN layer $i + 1$

- 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

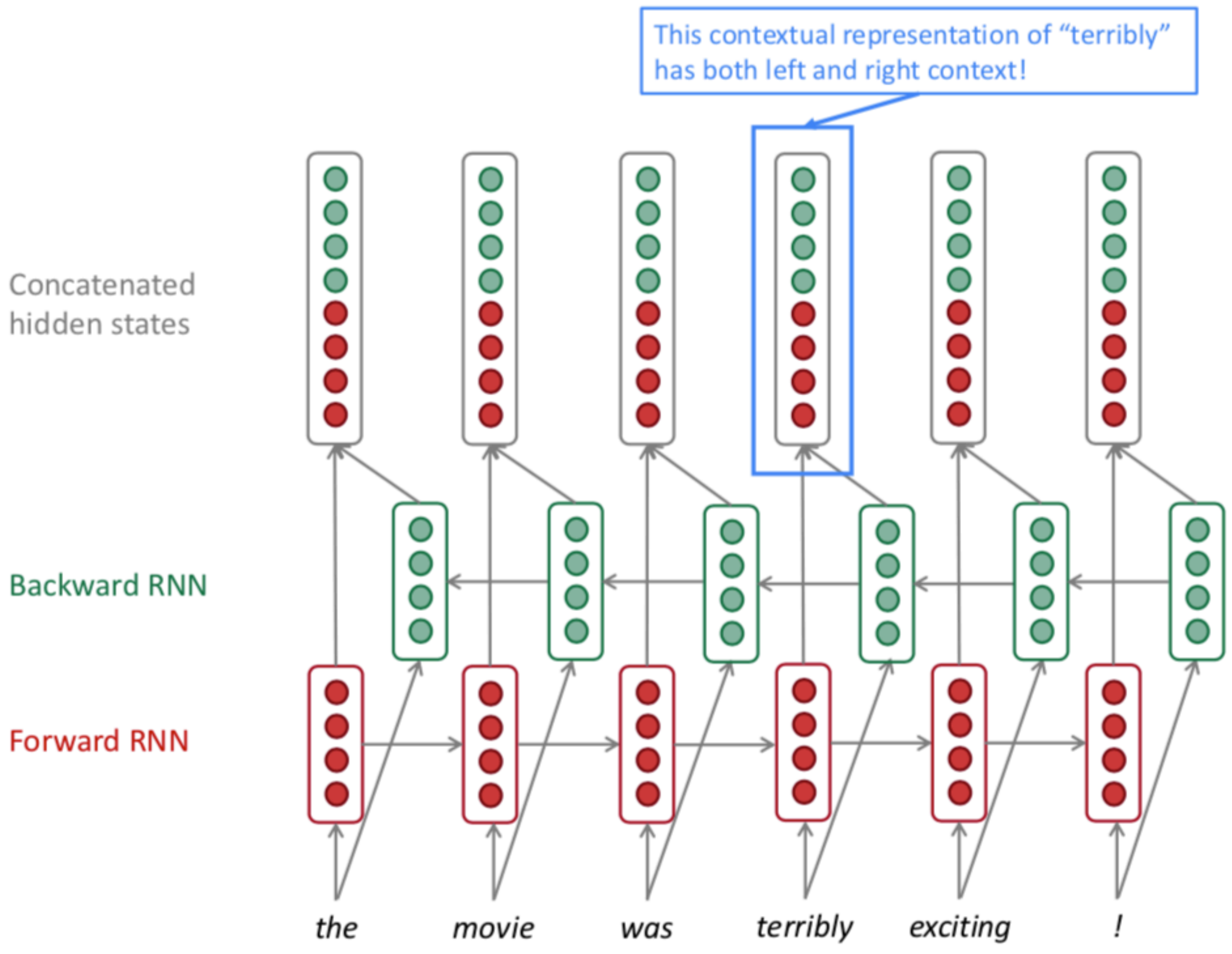
- Bidirectionality is important in language representations:



terribly:

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

Bidirectional RNNs



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

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

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

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

Zoom poll

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

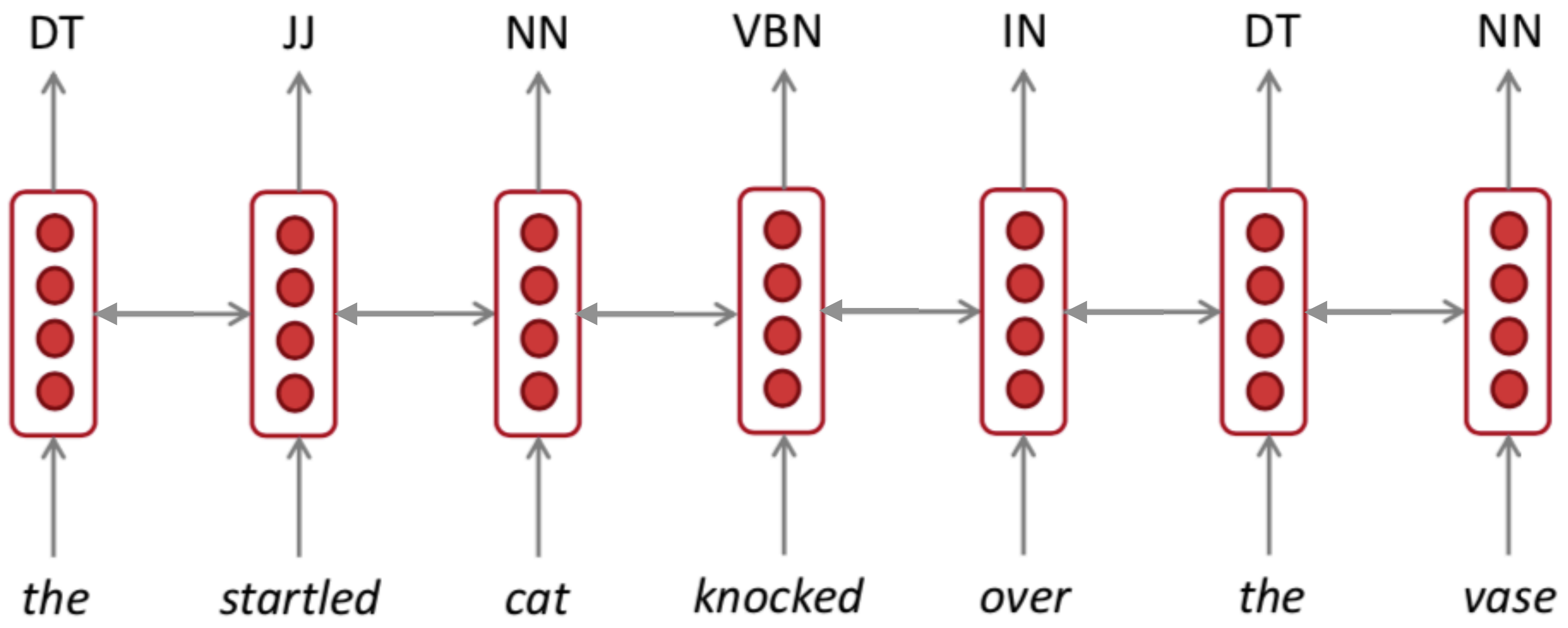
(c) Yes, Yes, No

(d) No, Yes, No

The answer is (c).

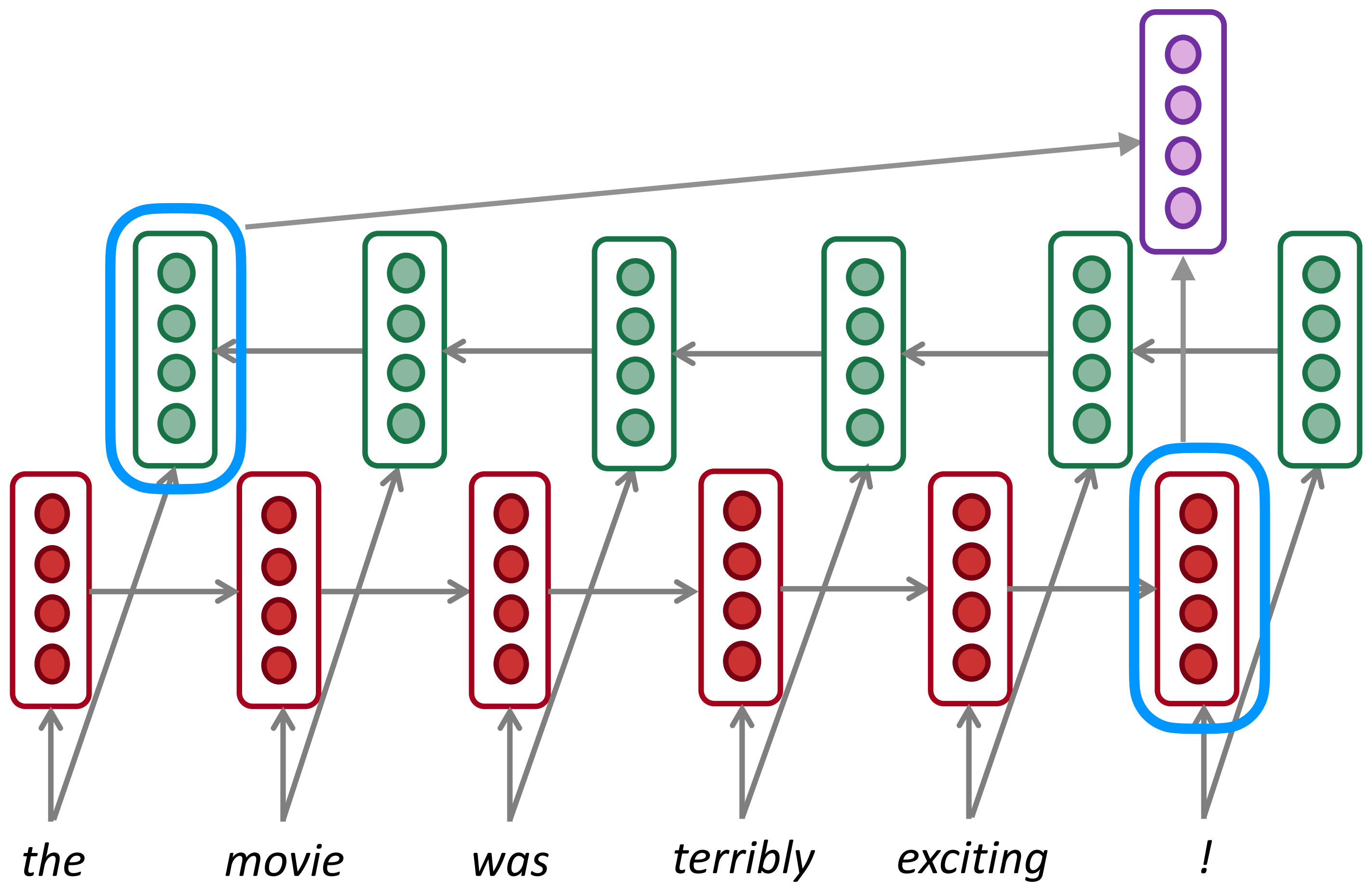
Bidirectional RNNs

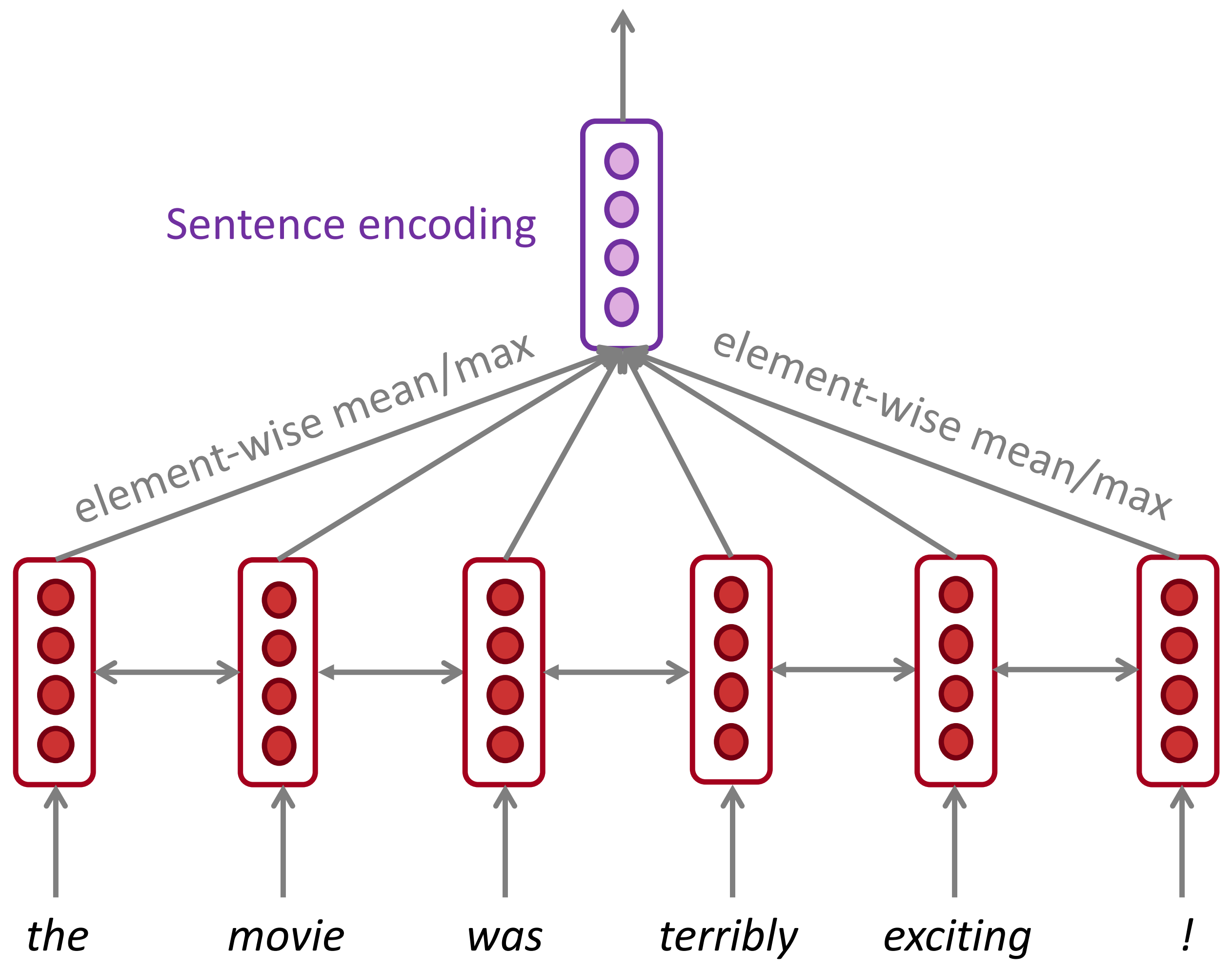
- Sequence tagging: Yes! (esp. important)



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 LSTMs  or GRUs  (next lecture)

- In practice, we always use multi-layer RNNs 

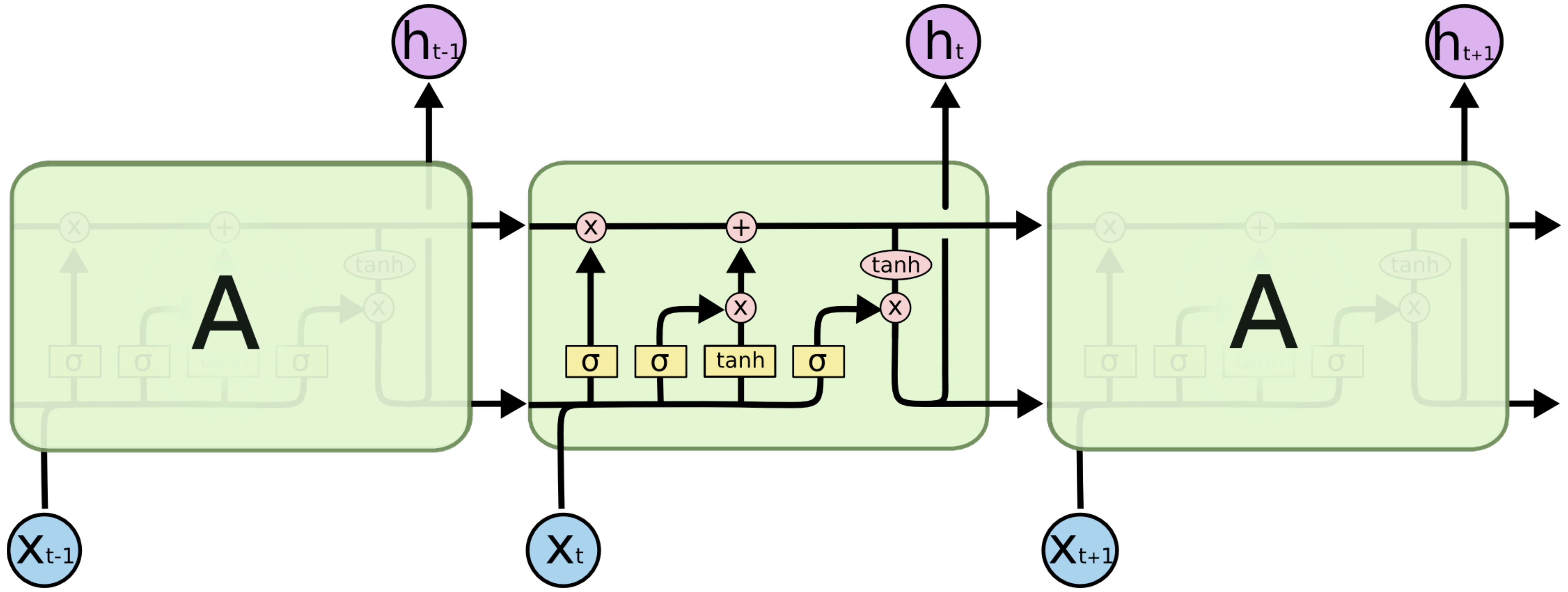
… together with fancy ingredients such as residual connections with self-attention, variational dropout..

Slide credit: Abigail See (with modifications)

Next Lecture

- Advanced RNN variants: LSTMs vs GRUs

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



- PyTorch/final project

Good luck with the midterm!