

# Natural Language Processing with Deep Learning

## CS224N/Ling284



Lecture 7:  
Vanishing Gradients  
and Fancy RNNs

**Abigail See**

# Announcements

- Assignment 4 released today
  - Due Thursday next week (9 days from now)
  - Based on Neural Machine Translation (NMT)
    - NMT will be covered in Thursday's lecture
  - You'll use Azure to get access to a virtual machine with a GPU
    - Budget extra time if you're not used to working on a remote machine (e.g. ssh, tmux, remote text editing)
  - Get started early
    - The NMT system takes 4 hours to train!
    - **Assignment 4 is quite a lot more complicated than Assignment 3!**
    - Don't be caught by surprise!
    - Thursday's slides + notes are already online

# Announcements

- Projects
  - Next week: lectures are all about choosing projects
  - It's fine to delay thinking about projects until next week
  - But if you're already thinking about projects, you can view some info/inspiration on the website's project page
  - Including: project ideas from potential Stanford AI Lab mentors. For these, best to get in contact and get started early!

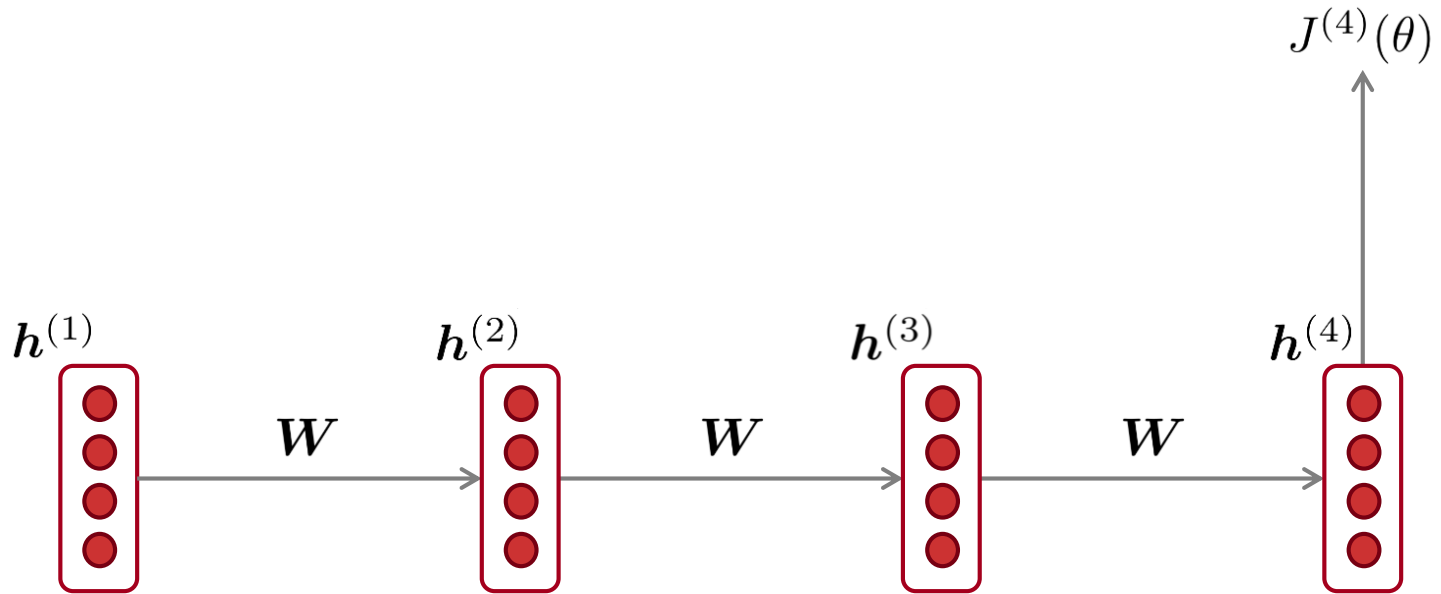
# Overview

- Last lecture we learned:
  - Recurrent Neural Networks (RNNs) and why they're great for Language Modeling (LM).
- Today we'll learn:
  - Problems with RNNs and how to fix them
  - More complex RNN variants
- Next lecture we'll learn:
  - How we can do Neural Machine Translation (NMT) using an RNN-based architecture called sequence-to-sequence with attention

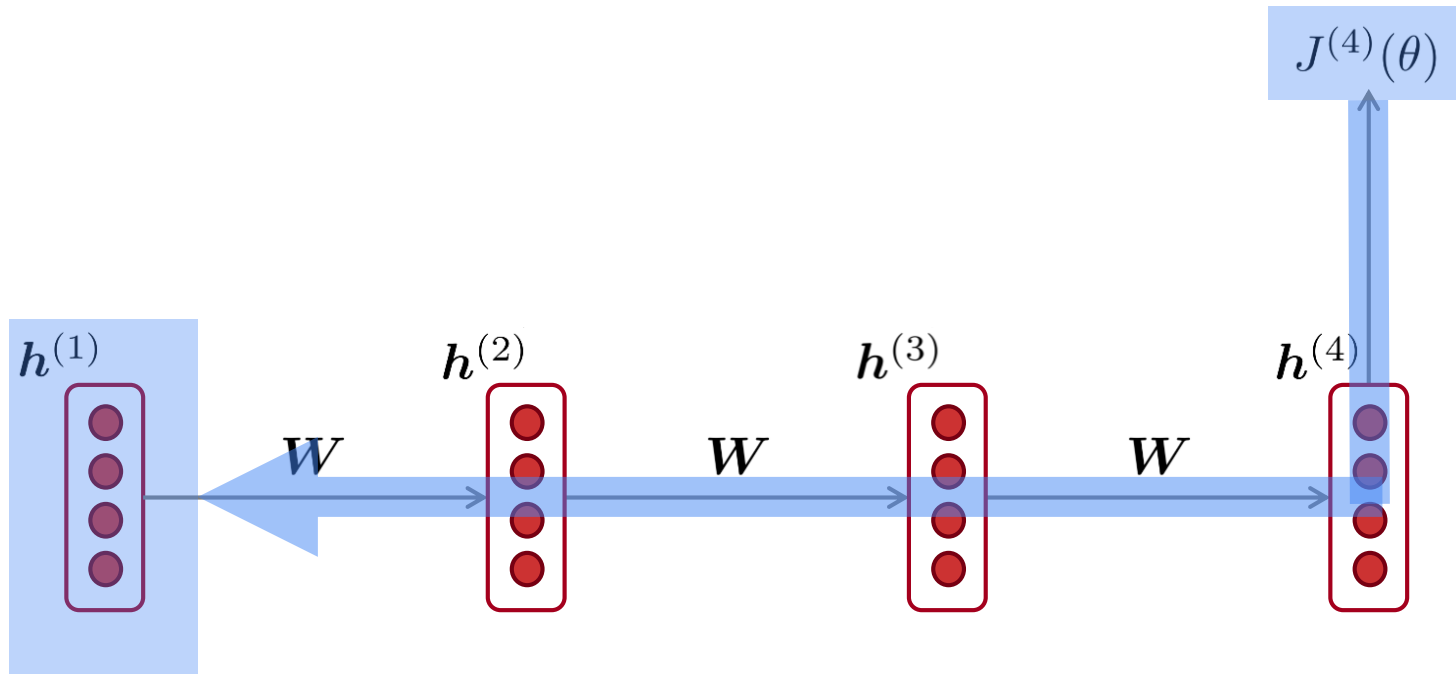
# Today's lecture

- Vanishing gradient problem
  - Two new types of RNN: LSTM and GRU
  - Other fixes for vanishing (or exploding) gradient:
    - Gradient clipping 简单
    - Skip connections 新
  - More fancy RNN variants:
    - Bidirectional RNNs
    - Multi-layer RNNs
- motivates
- Lots of important definitions today!

# Vanishing gradient intuition

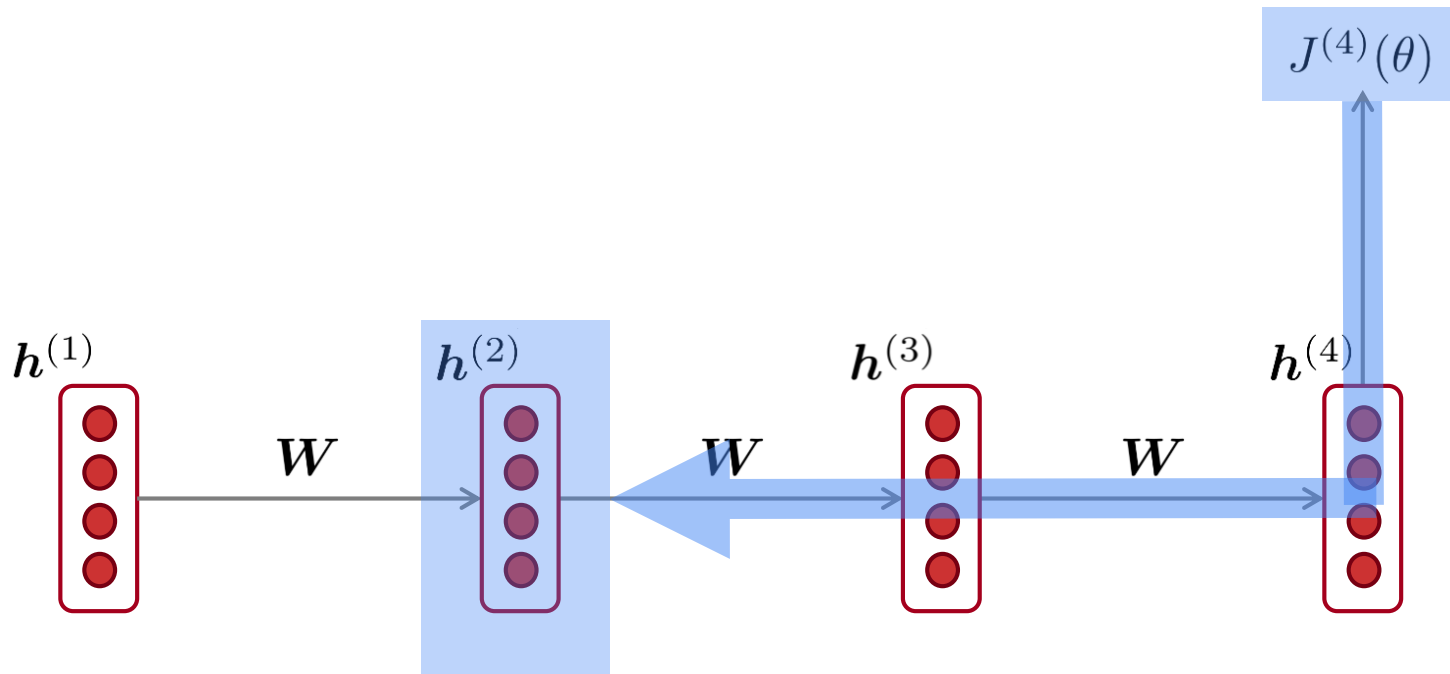


# Vanishing gradient intuition



$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = ?$$

# Vanishing gradient intuition

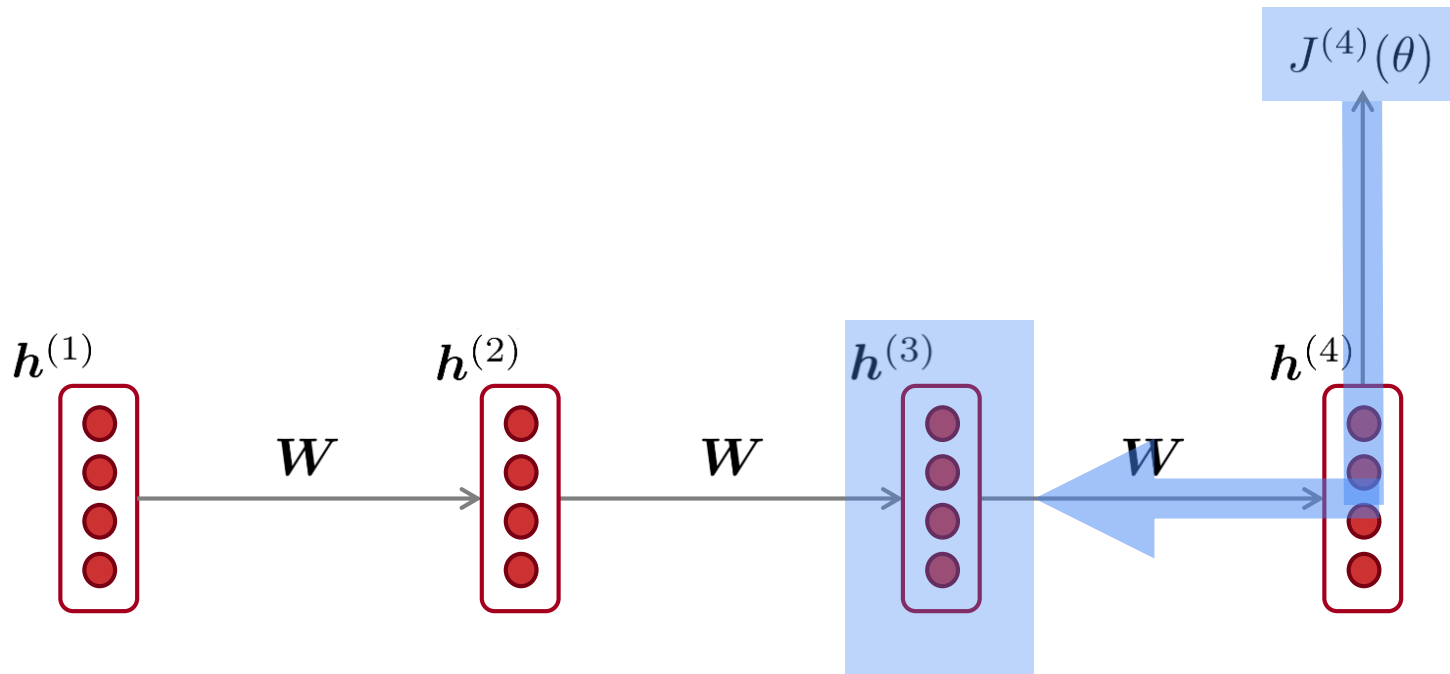


$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times \frac{\partial J^{(4)}}{\partial h^{(2)}}$$

chain rule!



# Vanishing gradient intuition

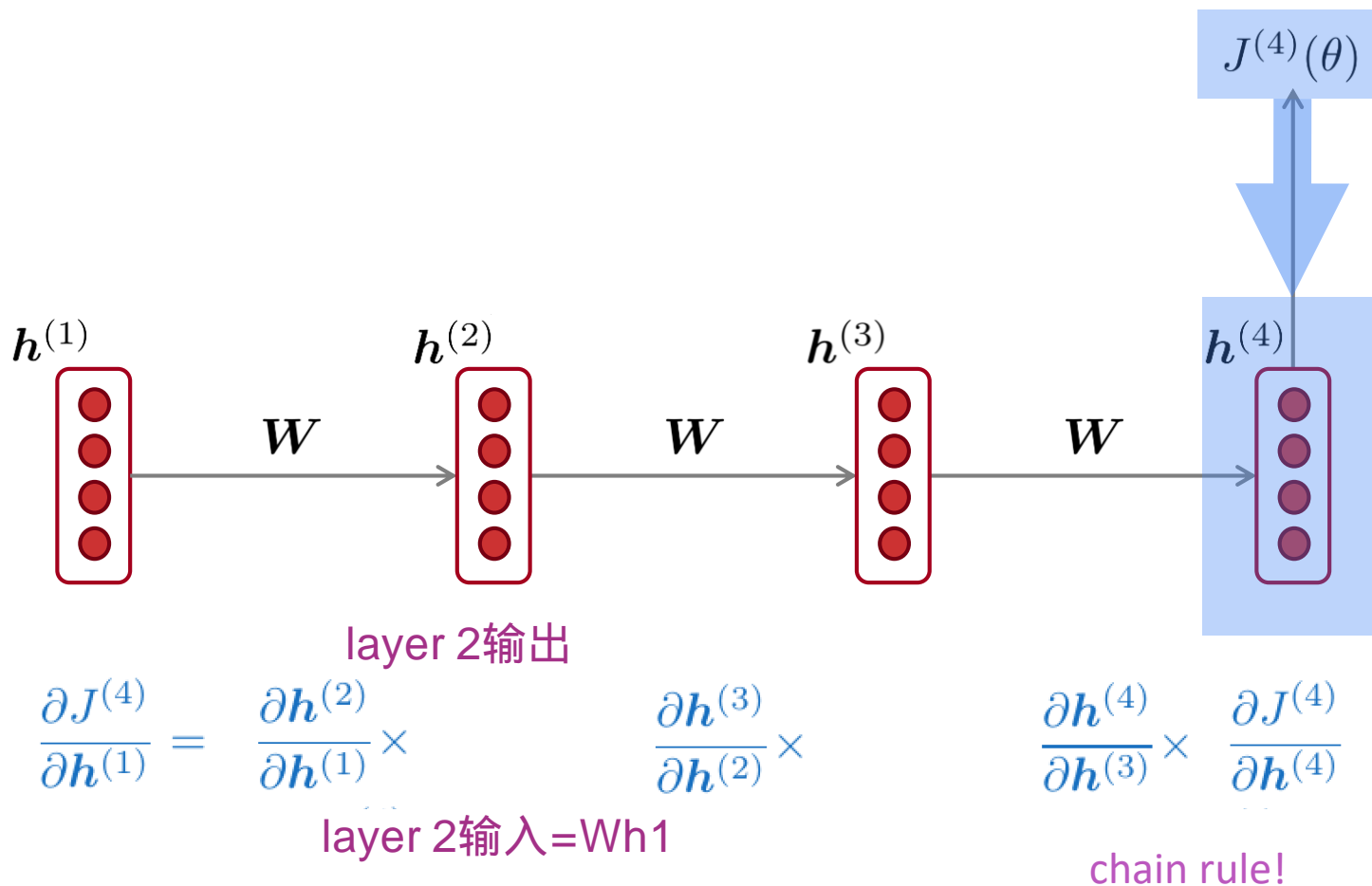


$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times$$

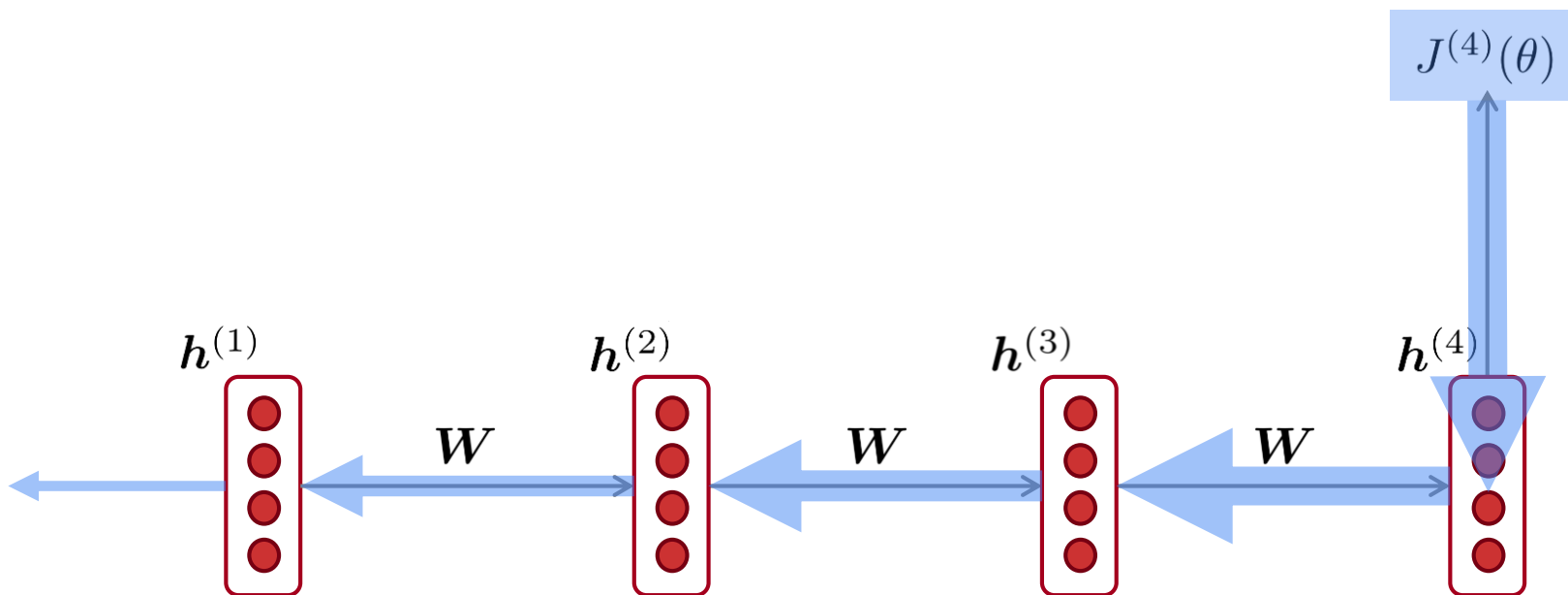
$$\frac{\partial h^{(3)}}{\partial h^{(2)}} \times \frac{\partial J^{(4)}}{\partial h^{(3)}}$$

chain rule!

# Vanishing gradient intuition



# Vanishing gradient intuition



$$\frac{\partial J^{(4)}}{\partial \mathbf{h}^{(1)}} = \boxed{\frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{h}^{(1)}}} \times \boxed{\frac{\partial \mathbf{h}^{(3)}}{\partial \mathbf{h}^{(2)}}} \times \boxed{\frac{\partial \mathbf{h}^{(4)}}{\partial \mathbf{h}^{(3)}}} \times \frac{\partial J^{(4)}}{\partial \mathbf{h}^{(4)}}$$

What happens if these are small?

Vanishing gradient problem:  
When these are small, the  
gradient signal gets smaller  
and smaller as it  
backpropagates further

# Vanishing gradient proof sketch

- Recall:  $\mathbf{h}^{(t)} = \sigma \left( \overset{\text{上个state}}{\mathbf{W}_h \mathbf{h}^{(t-1)}} + \underset{\text{当前输入}}{\mathbf{W}_x \mathbf{x}^{(t)}} + \mathbf{b}_1 \right)$  是element-wise function
- Therefore:  $\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(t-1)}} = \text{diag} \left( \sigma' \left( \mathbf{W}_h \mathbf{h}^{(t-1)} + \mathbf{W}_x \mathbf{x}^{(t)} + \mathbf{b}_1 \right) \right) \mathbf{W}_h$  (chain rule)
- Consider the gradient of the loss  $J^{(i)}(\theta)$  on step  $i$ , with respect to the hidden state  $\mathbf{h}^{(j)}$  on some previous step  $j$ .

$$\begin{aligned} \frac{\partial J^{(i)}(\theta)}{\partial \mathbf{h}^{(j)}} &= \frac{\partial J^{(i)}(\theta)}{\partial \mathbf{h}^{(i)}} \prod_{j < t \leq i} \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(t-1)}} && \text{(chain rule)} \\ &= \frac{\partial J^{(i)}(\theta)}{\partial \mathbf{h}^{(i)}} \boxed{\mathbf{W}_h^{(i-j)}} \prod_{j < t \leq i} \text{diag} \left( \sigma' \left( \mathbf{W}_h \mathbf{h}^{(t-1)} + \mathbf{W}_x \mathbf{x}^{(t)} + \mathbf{b}_1 \right) \right) && \left( \text{value of } \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(t-1)}} \right) \end{aligned}$$

↑ i-j次方【！】

If  $\mathbf{W}_h$  is small, then this term gets vanishingly small as  $i$  and  $j$  get further apart

# Vanishing gradient proof sketch

- Consider matrix L2 norms:

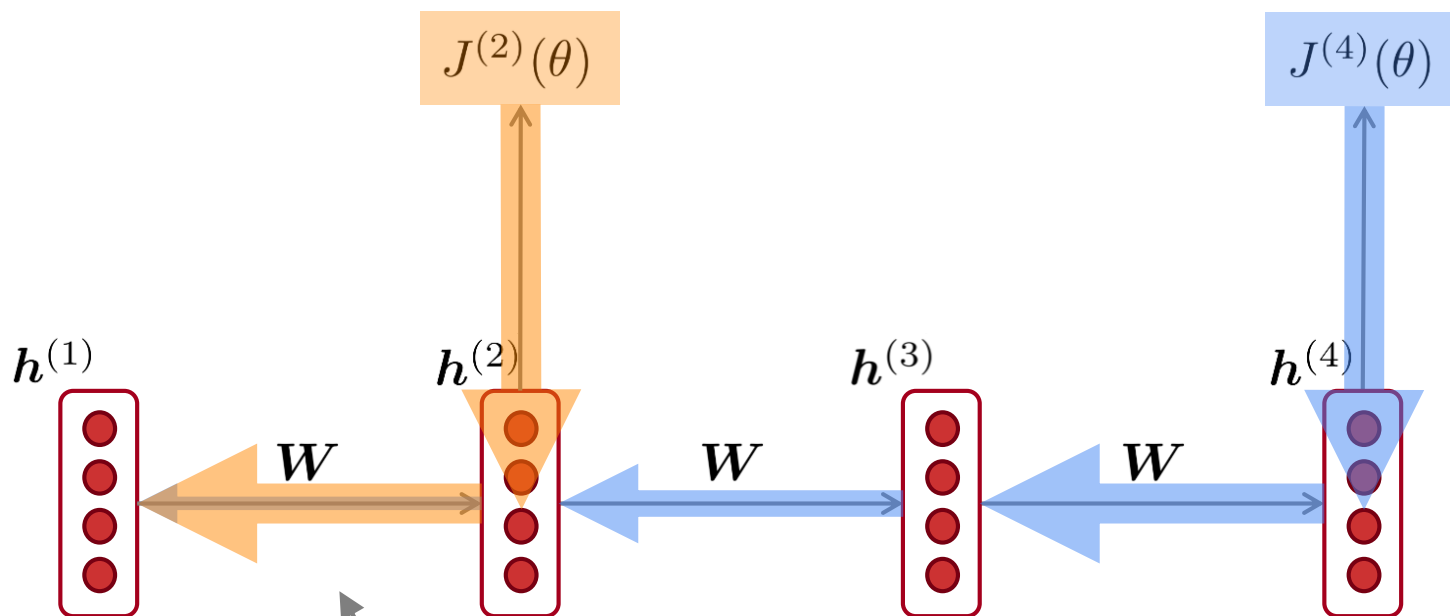
$$\left\| \frac{\partial J^{(i)}(\theta)}{\partial \mathbf{h}^{(j)}} \right\| \leq \left\| \frac{\partial J^{(i)}(\theta)}{\partial \mathbf{h}^{(i)}} \right\| \|\mathbf{W}_h\|^{(i-j)} \prod_{j < t \leq i} \left\| \text{diag} \left( \sigma' \left( \mathbf{W}_h \mathbf{h}^{(t-1)} + \mathbf{W}_x \mathbf{x}^{(t)} + \mathbf{b}_1 \right) \right) \right\|$$

取绝对值

- Pascanu et al showed that that if the largest eigenvalue of  $\mathbf{W}_h$  is less than 1, then the gradient  $\left\| \frac{\partial J^{(i)}(\theta)}{\partial \mathbf{h}^{(j)}} \right\|$  will shrink exponentially
  - Here the bound is 1 because we have sigmoid nonlinearity
- There's a similar proof relating a largest eigenvalue >1 to exploding gradients

【 ? 】 为什么是dJ/dh呢？——对h求微分是为了进一步计算dJ/dW？

# Why is vanishing gradient a problem?



更新自己的，越近的step影响越大 丢失了长期信息

Gradient signal from faraway is lost because it's much smaller than gradient signal from close-by.

So model weights are only updated only with respect to near effects, not long-term effects.

# Why is vanishing gradient a problem?

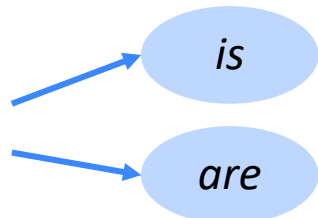


- Another explanation: Gradient can be viewed as a measure of *the effect of the past on the future*
- If the gradient becomes vanishingly small over longer distances (step  $t$  to step  $t+n$ ), then we can't tell whether:
  1. There's **no dependency** between step  $t$  and  $t+n$  in the data
  2. We have **wrong parameters** to capture the true dependency between  $t$  and  $t+n$  但因为梯度过小所以更新不了

# Effect of vanishing gradient on RNN-LM

- **LM task:** *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 \_\_\_\_\_*
- To learn from this training example, the RNN-LM needs to **model the dependency** between “tickets” on the 7<sup>th</sup> step and the target word “tickets” at the end.
- But if gradient is small, the model **can’t learn this dependency**
  - So the model is **unable to predict similar long-distance dependencies** at test time



# Effect of vanishing gradient on RNN-LM

- **LM task:** *The writer of the books \_\_\_\_* 
- **Correct answer:** *The writer of the books is planning a sequel*
- **Syntactic recency:** *The writer of the books is* (correct) 
- **Sequential recency:** *The writer of the books are* (incorrect) 
- Due to vanishing gradient, RNN-LMs are better at learning from **sequential recency** than **syntactic recency**, so they make this type of error more often than we'd like [Linzen et al 2016]

# Why is exploding gradient a problem?

- If the gradient becomes too big, then the SGD update step becomes too big:

$$\theta^{new} = \theta^{old} - \overbrace{\alpha}^{\text{learning rate}} \underbrace{\nabla_{\theta} J(\theta)}_{\text{gradient}}$$

- This can cause **bad updates**: we take too large a step and reach a bad parameter configuration (with large loss)
- In the worst case, this will result in **Inf** or **NaN** in your network (then you have to restart training from an earlier checkpoint)

# Gradient clipping: solution for exploding gradient

- 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  
     $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$  保证量级不超过threshold  
end if
```

---

- Intuition: take a step in the same direction, but a smaller step

# Gradient clipping: solution for exploding gradient

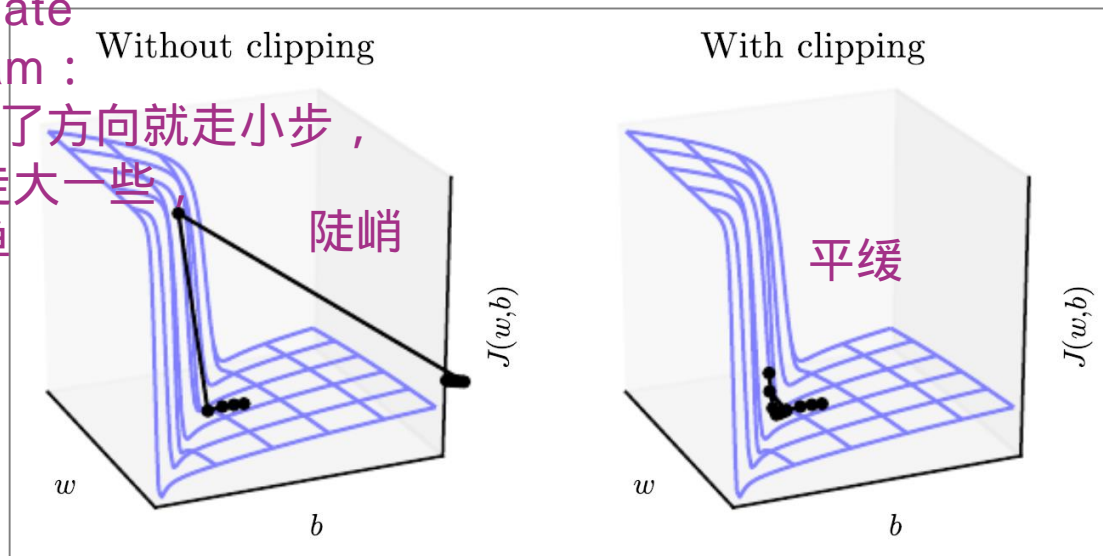
更安全的update

——联系Adam：

Adam是改变了方向就走小步，

没改变可以走大一些

这里的更简单



$w, b$ 都是标量

- This shows the loss surface of a simple RNN (hidden state is a scalar not a vector)
- The “cliff” is dangerous because it has steep gradient
- On the left, gradient descent takes two very big steps due to steep gradient, resulting in climbing the cliff then shooting off to the right (both bad updates)
- On the right, gradient clipping reduces the size of those steps, so effect is less drastic

# How to fix vanishing gradient problem?

(之前讲了exploding clipping, 现在进入vanishing)

- The main problem is that *it's too difficult for the RNN to learn to preserve information over many timesteps.*

- In a vanilla RNN, the hidden state is constantly being rewritten

$$\mathbf{h}^{(t)} = \sigma \left( \mathbf{W}_h \mathbf{h}^{(t-1)} + \mathbf{W}_x \mathbf{x}^{(t)} + \mathbf{b} \right)$$

- How about a RNN with separate memory?

之前都是来自上一个hidden state

(不容易保存上次的信息, 因为经过了非线性的)

# Long Short-Term Memory (LSTM)

- A type of RNN proposed by Hochreiter and Schmidhuber in 1997 as a solution to the vanishing gradients problem.
- On step  $t$ , there is a hidden state  $\mathbf{h}^{(t)}$  and a cell state  $\mathbf{c}^{(t)}$ 
  - Both are vectors length  $n$
  - The cell stores long-term information memory unit
  - The LSTM can erase, write and read information from the cell
- The selection of which information is erased/written/read is controlled by three corresponding gates controller
  - The gates are also vectors length  $n$
  - On each timestep, each element of the gates can be open (1), closed (0), or somewhere in-between. 门开了, information才能pass through
  - The gates are dynamic: their value is computed based on the current context 不是常数, 每个timestep都会不同

# Long Short-Term Memory (LSTM)

所有向量都是  $n \times 1$  的

Q: 为什么 forget gate 不需要看一下 cell state  $c^{t-1}$

We have a sequence of inputs  $\mathbf{x}^{(t)}$ , and we will compute a sequence of hidden states  $\mathbf{h}^{(t)}$  and cell states  $\mathbf{c}^{(t)}$ . On timestep  $t$ :

保存/遗忘

**Forget gate:** controls what is kept vs forgotten, from previous cell state

**Sigmoid function:** all gate values are between 0 and 1

控制 input 进入 cell

**Input gate:** controls what parts of the new cell content are written to cell

控制输出到 hidden state

**Output gate:** controls what parts of cell are output to hidden state

$$\mathbf{f}^{(t)} = \sigma \left( \overset{\text{上一state}}{\mathbf{W}_f \mathbf{h}^{(t-1)}} + \overset{\text{当前input}}{\mathbf{U}_f \mathbf{x}^{(t)}} + \mathbf{b}_f \right)$$

$$\mathbf{i}^{(t)} = \sigma \left( \mathbf{W}_i \mathbf{h}^{(t-1)} + \mathbf{U}_i \mathbf{x}^{(t)} + \mathbf{b}_i \right)$$

$$\mathbf{o}^{(t)} = \sigma \left( \mathbf{W}_o \mathbf{h}^{(t-1)} + \mathbf{U}_o \mathbf{x}^{(t)} + \mathbf{b}_o \right)$$

**New cell content:** this is the new content to be written to the cell 新memory

**Cell state:** erase ("forget") some content from last cell state, and write ("input") some new cell content

**Hidden state:** read ("output") some content from the cell

【QA】

Q: 为什么 new cell 要经过 tanh ?

A: 可能会更有 expressivity ?

$$\tilde{\mathbf{c}}^{(t)} = \tanh \left( \mathbf{W}_c \mathbf{h}^{(t-1)} + \mathbf{U}_c \mathbf{x}^{(t)} + \mathbf{b}_c \right)$$

$$\mathbf{c}^{(t)} = \underset{\text{上一memory}}{\mathbf{f}^{(t)}} \circ \mathbf{c}^{(t-1)} + \underset{\text{新memory}}{\mathbf{i}^{(t)}} \circ \tilde{\mathbf{c}}^{(t)}$$

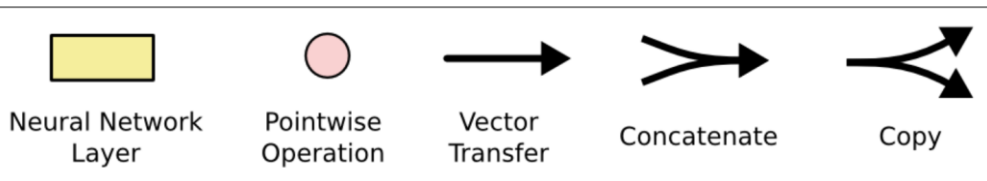
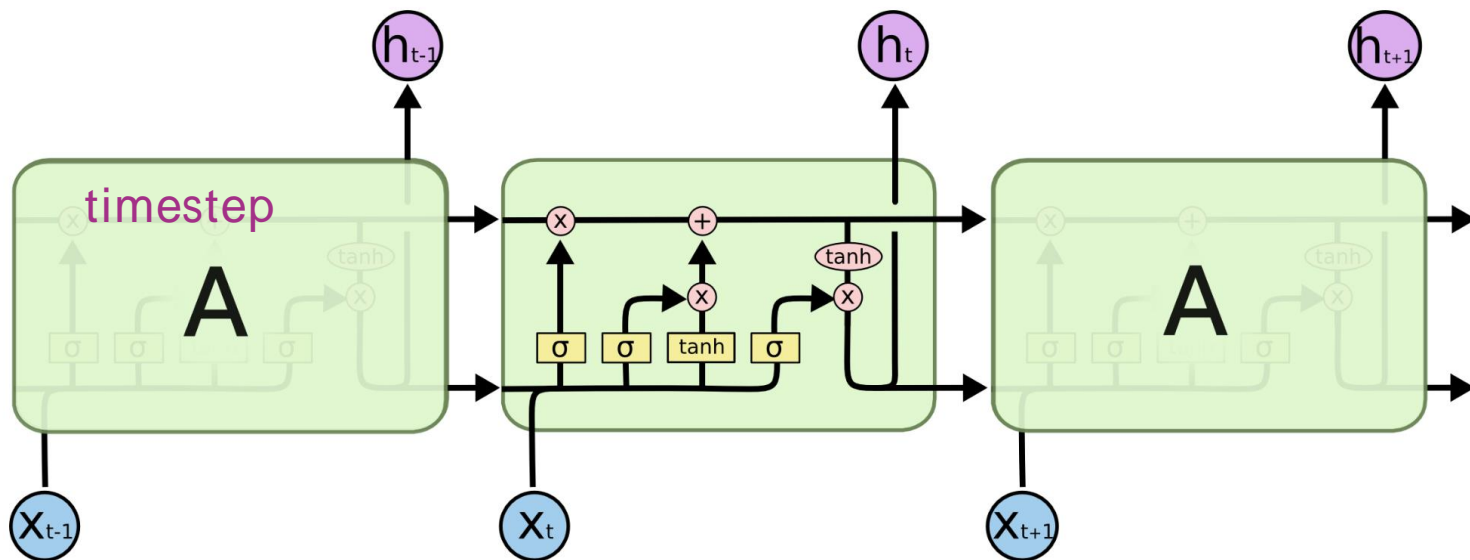
$$\mathbf{h}^{(t)} = \mathbf{o}^{(t)} \circ \tanh \mathbf{c}^{(t)}$$

Gates are applied using element-wise product

All these are vectors of same length  $n$

# Long Short-Term Memory (LSTM)

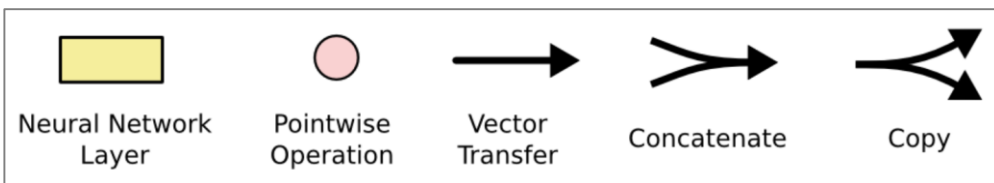
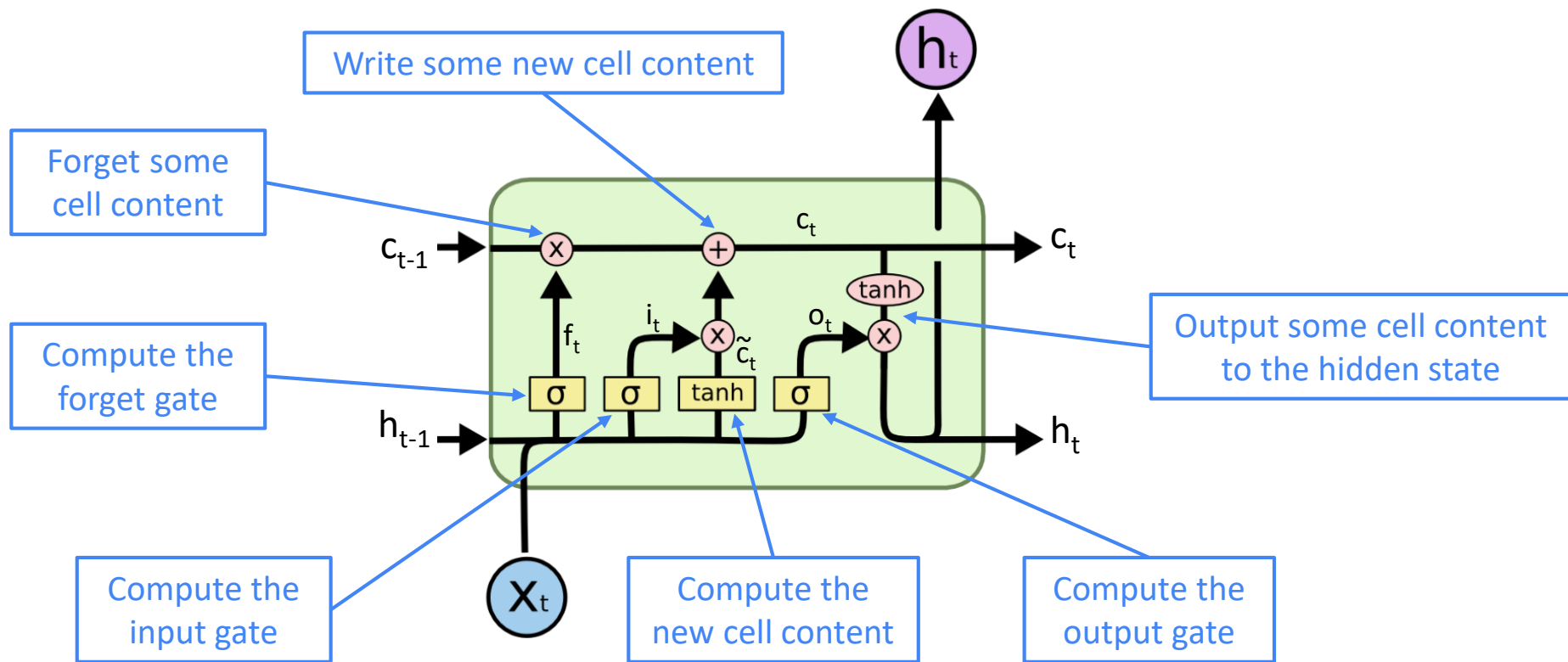
You can think of the LSTM equations visually like this:





# Long Short-Term Memory (LSTM)

You can think of the LSTM equations visually like this:



# How does LSTM solve vanishing gradients?

- The LSTM architecture makes it **easier** for the RNN to **preserve information over many timesteps**  
straightforward way
  - e.g. if the forget gate is set to remember everything on every timestep, then the info in the cell is preserved indefinitely
  - By contrast, it's harder for vanilla RNN to learn a recurrent weight matrix  $W_h$  that preserves info in hidden state
- LSTM doesn't guarantee that there is no vanishing/exploding gradient, but it does provide an easier way for the model to learn long-distance dependencies

存在shortcut connection，梯度不一定要经过所有timestep才能反向传播回来

Q: 这样如何检验梯度呢？

A: 多元链式法则

# LSTMs: real-world success

之前取得了巨大成功

- In 2013-2015, LSTMs started achieving state-of-the-art results
  - Successful tasks include: handwriting recognition, speech recognition, machine translation, parsing, image captioning
  - LSTM became the dominant approach

但现在transformer独领风骚

- Now (2019), other approaches (e.g. Transformers) have become more dominant for certain tasks.
  - For example in **WMT** (a MT conference + competition):
  - In WMT 2016, the summary report contains "RNN" 44 times
  - In WMT 2018, the report contains "RNN" 9 times and "Transformer" 63 times

Source: "Findings of the 2016 Conference on Machine Translation (WMT16)", Bojar et al. 2016, <http://www.statmt.org/wmt16/pdf/W16-2301.pdf>

Source: "Findings of the 2018 Conference on Machine Translation (WMT18)", Bojar et al. 2018, <http://www.statmt.org/wmt18/pdf/WMT028.pdf>

# Gated Recurrent Units (GRU)

- 更简单，没有cell state，但还是用gate控制  
Proposed by Cho et al. in 2014 as a simpler alternative to the LSTM.
- On each timestep  $t$  we have input  $\mathbf{x}^{(t)}$  and hidden state  $\mathbf{h}^{(t)}$  (no cell state).

**Update gate:** controls what parts of hidden state are updated vs preserved

更新比例, balanced

$$\mathbf{u}^{(t)} = \sigma \left( \mathbf{W}_u \mathbf{h}^{(t-1)} + \mathbf{U}_u \mathbf{x}^{(t)} + \mathbf{b}_u \right)$$

**Reset gate:** controls what parts of previous hidden state are used to compute new content

$$\mathbf{r}^{(t)} = \sigma \left( \mathbf{W}_r \mathbf{h}^{(t-1)} + \mathbf{U}_r \mathbf{x}^{(t)} + \mathbf{b}_r \right)$$

新内容中老state的比例

**New hidden state content:** reset gate selects useful parts of prev hidden state. Use this and current input to compute new hidden content.

新内容

$$\tilde{\mathbf{h}}^{(t)} = \tanh \left( \mathbf{W}_h (\mathbf{r}^{(t)} \circ \mathbf{h}^{(t-1)}) + \mathbf{U}_h \mathbf{x}^{(t)} + \mathbf{b}_h \right)$$

old state      new input

$$\mathbf{h}^{(t)} = (1 - \mathbf{u}^{(t)}) \circ \mathbf{h}^{(t-1)} + \mathbf{u}^{(t)} \circ \tilde{\mathbf{h}}^{(t)}$$

**Hidden state:** update gate simultaneously controls what is kept from previous hidden state, and what is updated to new hidden state content

**How does this solve vanishing gradient?**

Like LSTM, GRU makes it easier to retain info long-term (e.g. by setting update gate to 0)

# LSTM vs GRU

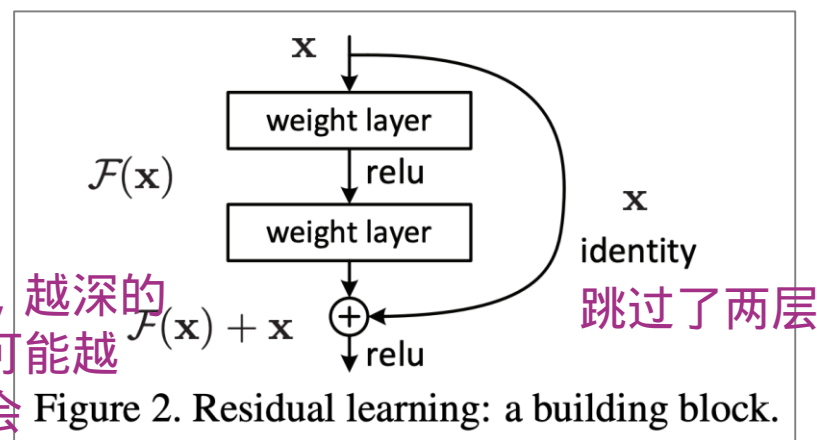
- Researchers have proposed many gated RNN variants, but LSTM and GRU are the most widely-used
- The biggest difference is that GRU is quicker to compute and has fewer parameters  
GRU更简单
- There is no conclusive evidence that one consistently performs better than the other
- LSTM is a good default choice (especially if your data has particularly long dependencies, or you have lots of training data)  
长距离LSTM更好      LSTM有超多参数，GRU则容易过拟合？
- Rule of thumb: start with LSTM, but switch to GRU if you want something more efficient

# Is vanishing/exploding gradient just a RNN problem?

- No! It can be a problem for all neural architectures (including **feed-forward** and **convolutional**), especially **deep** ones.
  - Due to chain rule / choice of nonlinearity function, gradient can become vanishingly small as it backpropagates **比如sigmoid**
  - Thus lower layers are learnt very slowly (hard to train)
  - Solution: lots of new deep feedforward/convolutional architectures that **add more direct connections** (thus allowing the gradient to flow)

For example:

- **Residual connections** aka “ResNet”
- Also known as **skip-connections**
- The **identity connection** 如果没有的话, 越深的 preserves information by default 网络可能越学不会
- This makes **deep** networks much easier to train

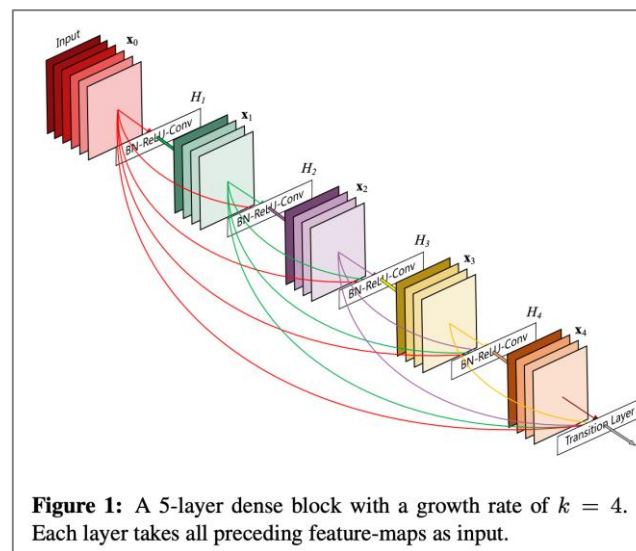


# Is vanishing/exploding gradient just a RNN problem?

- No! It can be a problem for all neural architectures (including feed-forward and convolutional), especially deep ones.
  - Due to chain rule / choice of nonlinearity function, gradient can become vanishingly small as it backpropagates
  - Thus lower layers are learnt very slowly (hard to train)
  - Solution: lots of new deep feedforward/convolutional architectures that add more direct connections (thus allowing the gradient to flow)

For example:

- Dense connections aka “DenseNet”
- Directly connect everything to everything!  
skip connections更多了



# Is vanishing/exploding gradient just a RNN problem?

- No! It can be a problem for all neural architectures (including **feed-forward** and **convolutional**), especially **deep** ones.
  - Due to chain rule / choice of nonlinearity function, gradient can become vanishingly small as it backpropagates
  - Thus lower layers are learnt very slowly (hard to train)
  - Solution: lots of new deep feedforward/convolutional architectures that **add more direct connections** (thus allowing the gradient to flow)

For example:

- **Highway connections** aka “**HighwayNet**”
- Similar to residual connections, but the identity connection vs the transformation layer is controlled by a **dynamic gate**控制balance of identity+上层
- Inspired by LSTMs, but applied to deep feedforward/convolutional networks



# Is vanishing/exploding gradient just a RNN problem?

- No! It can be a problem for all neural architectures (including feed-forward and convolutional), especially deep ones.
  - Due to chain rule / choice of nonlinearity function, gradient can become vanishingly small as it backpropagates
  - Thus lower layers are learnt very slowly (hard to train)
  - Solution: lots of new deep feedforward/convolutional architectures that add more direct connections (thus allowing the gradient to flow)

# Is vanishing/exploding gradient just a RNN problem?

不是，但是RNN问题尤其明显

- No! It can be a problem for all neural architectures (including feed-forward and convolutional), especially deep ones.
  - Due to chain rule / choice of nonlinearity function, gradient can become vanishingly small as it backpropagates
  - Thus lower layers are learnt very slowly (hard to train)
  - Solution: lots of new deep feedforward/convolutional architectures that add more direct connections (thus allowing the gradient to flow)
- Conclusion: Though vanishing/exploding gradients are a general problem, RNNs are particularly unstable due to the repeated multiplication by the same weight matrix [Bengio et al, 1994]

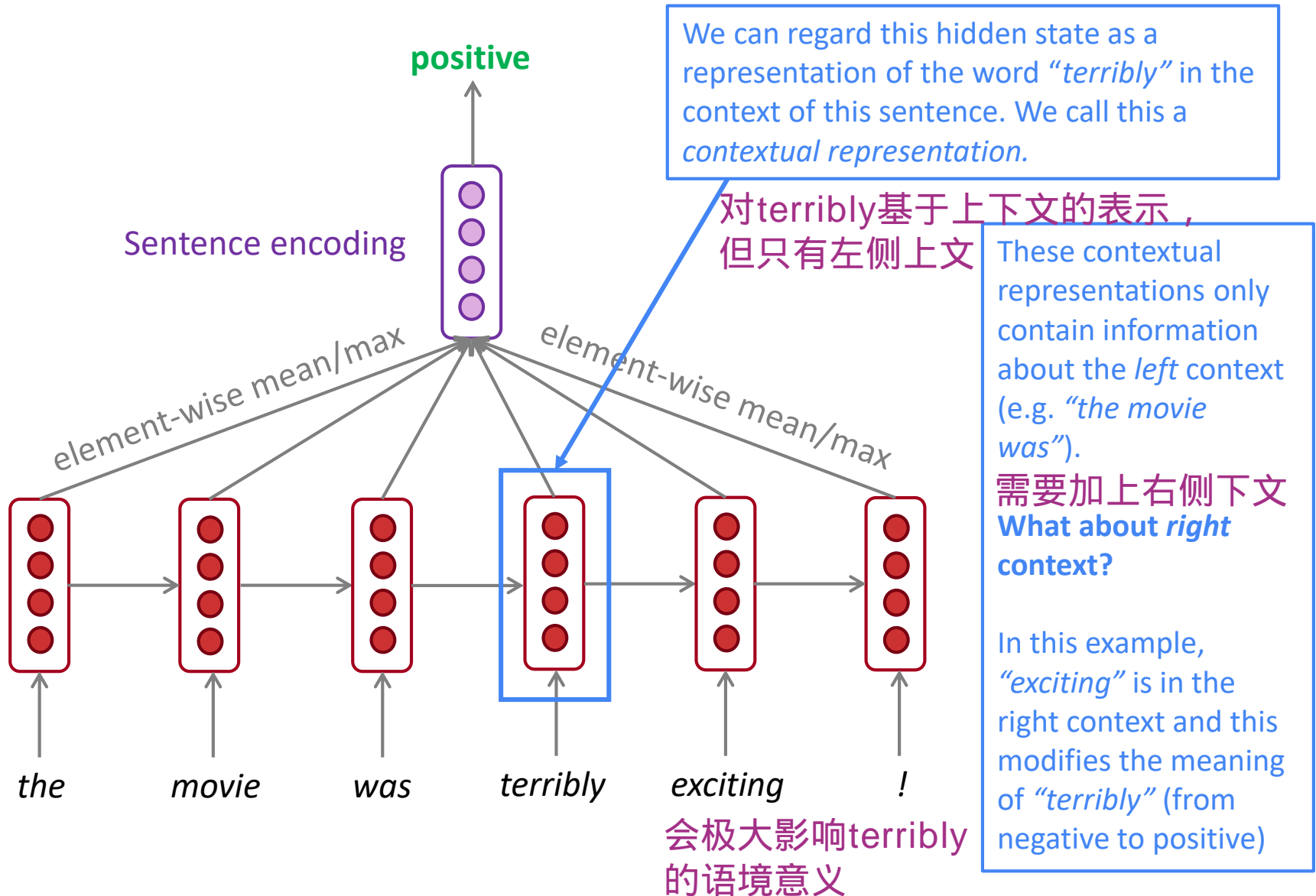
# Recap

- Today we've learnt:
  - **Vanishing gradient problem:** what it is, why it happens, and why it's bad for RNNs
  - **LSTMs and GRUs:** more complicated RNNs that use gates to control information flow; they are more resilient to vanishing gradients
- Remainder of this lecture:
  - **Bidirectional RNNs** info双向流动
  - **Multi-layer RNNs**

} Both of these are pretty simple conceptually

# Bidirectional RNNs: motivation

Task: Sentiment Classification



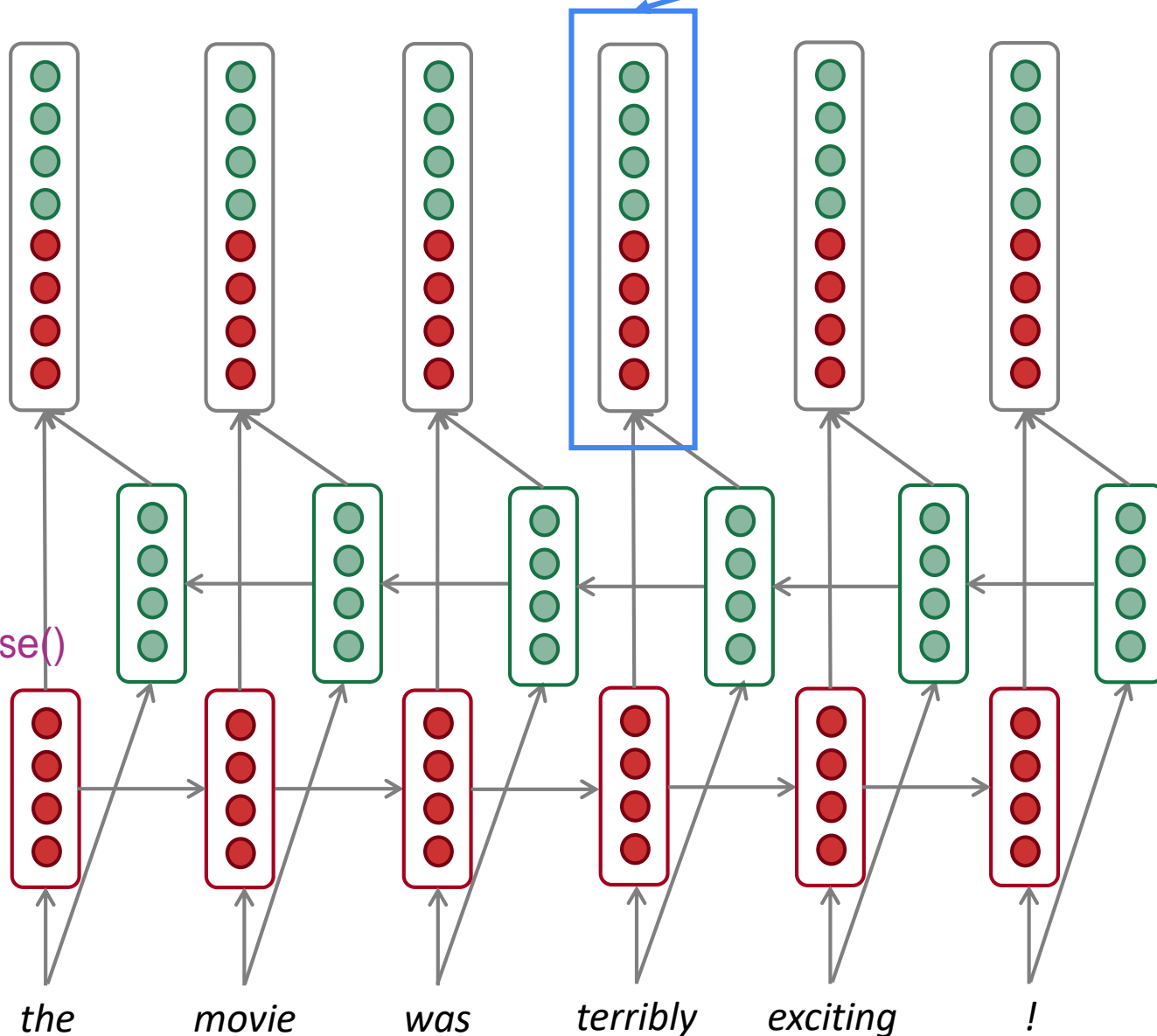
# Bidirectional RNNs

This contextual representation of “terribly” has both left and right context!

Concatenated  
hidden states

和forward完全一样  
Backward RNN  
只有sentence.reverse()

Forward RNN



# Bidirectional RNNs

On timestep  $t$ :

This is a general notation to mean “compute one forward step of the RNN” – it could be a vanilla, LSTM or GRU computation.

Forward RNN  $\vec{h}^{(t)} = \text{RNN}_{\text{FW}}(\vec{h}^{(t-1)}, \mathbf{x}^{(t)})$

Backward RNN  $\overleftarrow{h}^{(t)} = \text{RNN}_{\text{BW}}(\overleftarrow{h}^{(t+1)}, \mathbf{x}^{(t)})$

Generally, these two RNNs have separate weights

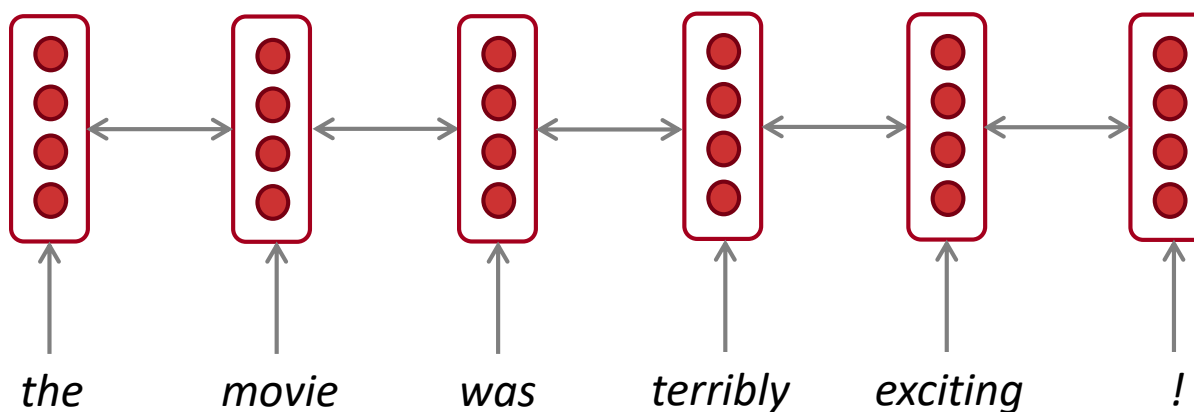
Concatenated hidden states

$$\mathbf{h}^{(t)} = [\vec{h}^{(t)}; \overleftarrow{h}^{(t)}]$$

也有些论文是 shared, 但 data 很多的时候不同更好

We regard this as “the hidden state” of a bidirectional RNN. This is what we pass on to the next parts of the network.

# Bidirectional RNNs: simplified diagram



只是把箭头变成了左右都有

The two-way arrows indicate bidirectionality and the depicted hidden states are assumed to be the concatenated forwards+backwards states.

standard practice是两个方向的RNN trained together

# Bidirectional RNNs

- Note: bidirectional RNNs are only applicable if you have access to the **entire input sequence**. 要有完整的上文 & 下文, LM只有上文
  - They are **not** applicable to Language Modeling, because in LM you *only* have left context available.
- If you do have entire input sequence (e.g. any kind of encoding), **bidirectionality is powerful** (you should use it by default).
- ~2019
  - For example, **BERT** (**Bidirectional** Encoder Representations from Transformers) is a powerful pretrained contextual representation system **built on bidirectionality**. 双向性是BERT很重要的部分
    - You will learn more about BERT later in the course!



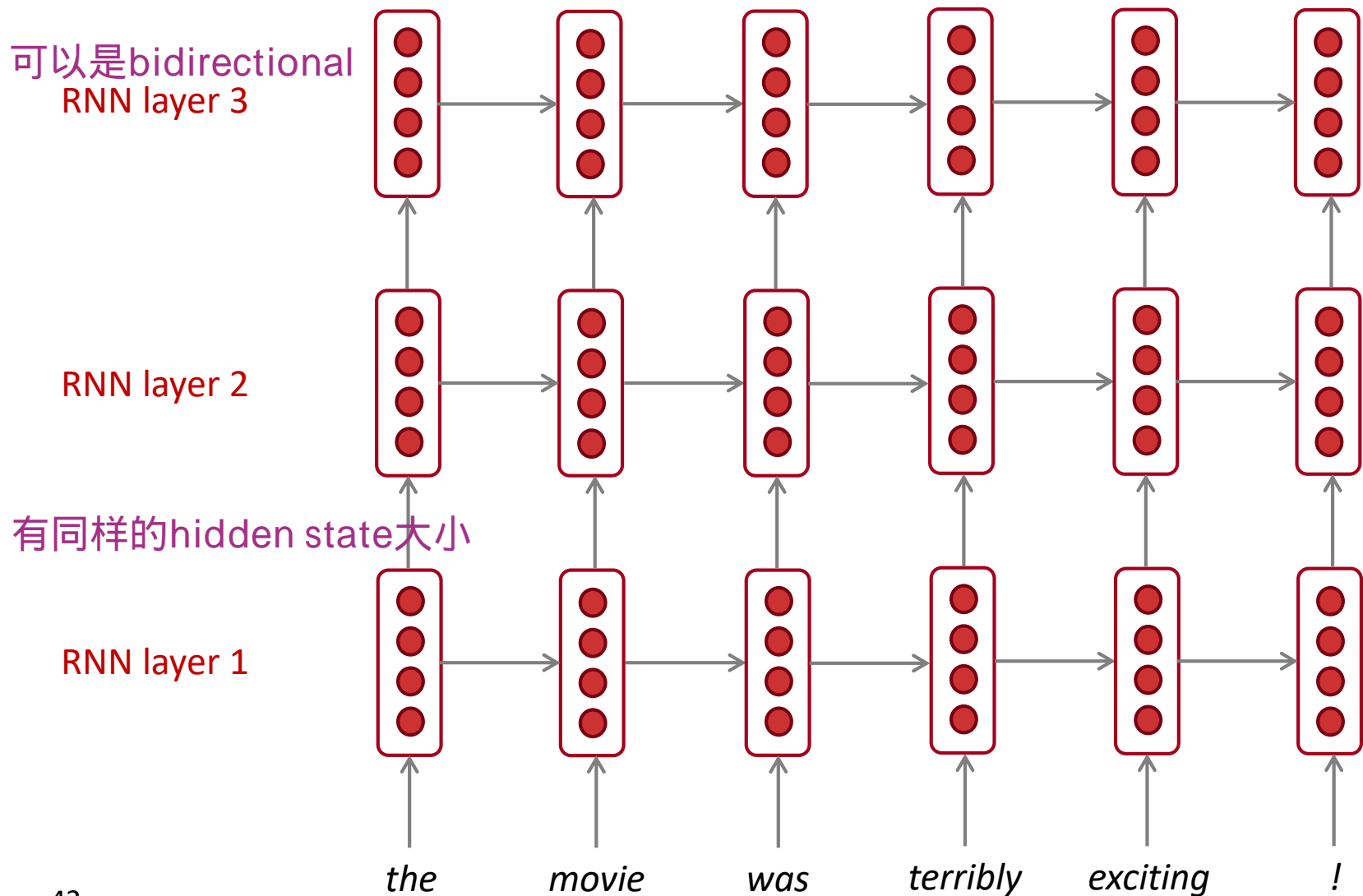
# Multi-layer RNNs

- RNNs are already “deep” on one dimension (they unroll over many timesteps)
- We can also make them “deep” in another dimension by **applying multiple RNNs** – this is a multi-layer RNN.
- This allows the network to compute **more complex representations** 和“deeper is better”的逻辑相同，底层学到syntax，顶层 semantics
  - The **lower RNNs** should compute **lower-level features** and the **higher RNNs** should compute **higher-level features**.
- Multi-layer RNNs are also called ***stacked RNNs***.

# Multi-layer RNNs

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

计算顺序可以横向可以纵向（单向RNN时），PyTorch是layer by layer（bi只能这样）

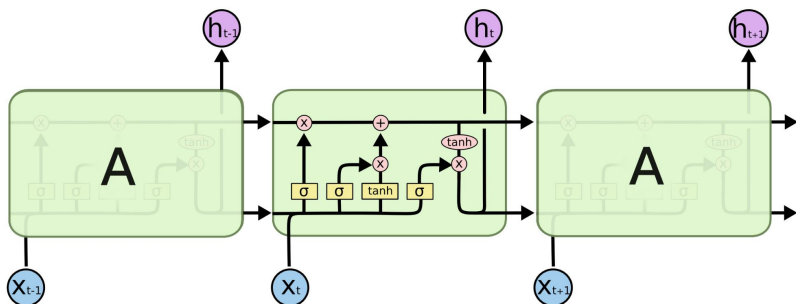


# Multi-layer RNNs in practice

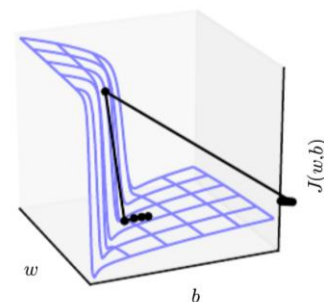
- High-performing RNNs are often multi-layer (but aren't as deep as convolutional or feed-forward networks)
- For example: In a 2017 paper, Britz et al find that for Neural Machine Translation, 2 to 4 layers is best for the encoder RNN, and 4 layers is best for the decoder RNN
  - However, skip-connections/dense-connections are needed to train deeper RNNs (e.g. 8 layers) 可以更深，但是深了计算复杂度也大了
- Transformer-based networks (e.g. BERT) can be up to 24 layers
  - You will learn about Transformers later; they have a lot of 另一version是12层 skipping-like connections

# In summary

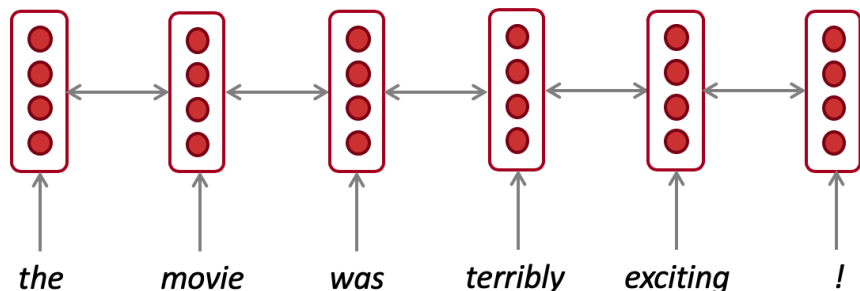
Lots of new information today! What are the **practical takeaways**?



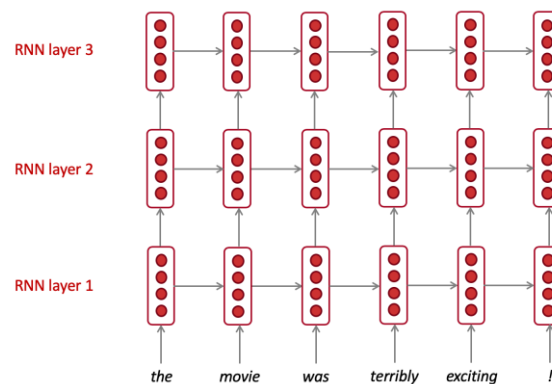
1. LSTMs are powerful but GRUs are faster



2. Clip your gradients  
否则很可能出现NaN



3. Use bidirectionality when possible  
by default



4. Multi-layer RNNs are powerful, but you might need skip/dense-connections if it's deep