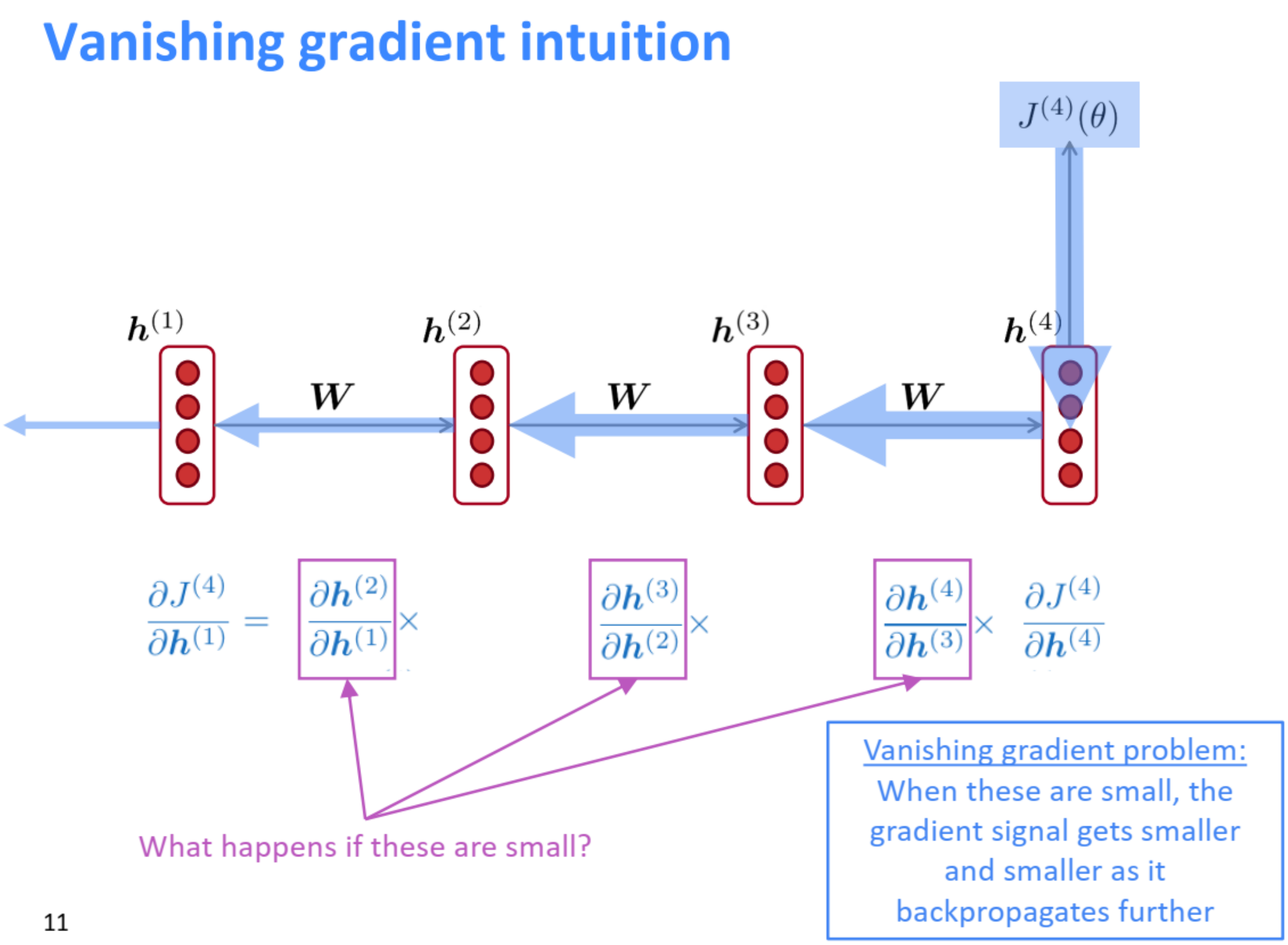
CS224n: NLP with Deep Learning

Lecture 7: Vanishing Gradients and Fancy RNN

Vanishing Gradient Problem

Intuition

- When the gradients $\frac{\partial h^{(i)}}{\partial h^{(i-1)}}$ are small,
  the overall gradient is going to get smaller and smaller, as we go backwards



- If the largest eigenvalue of W_h is < 1:

the gradient will shrink exponentially

→ Vanishing gradient !

Proof Sketch

Vanishing gradient proof sketch

- Recall: $$\boldsymbol{h}^{(t)} = \sigma\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x \boldsymbol{x}^{(t)} + \boldsymbol{b}_1\right)$$

- Therefore: $$\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}} = \operatorname{diag}\left(\sigma'\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x \boldsymbol{x}^{(t)} + \boldsymbol{b}_1\right)\right) \boldsymbol{W}_h$$ (chain rule)

- Consider the gradient of the loss $J^{(i)}(\theta)$ on step *i*, with respect to the hidden state $\boldsymbol{h}^{(j)}$ on some previous step *j*.

$$\frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(j)}} = \frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \prod_{j<t\leq i} \frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}}$$ (chain rule)

$$= \frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(i)}} \boxed{\boldsymbol{W}_h^{(i-j)}} \prod_{j<t\leq i} \operatorname{diag}\left(\sigma'\left(\boldsymbol{W}_h \boldsymbol{h}^{(t-1)} + \boldsymbol{W}_x \boldsymbol{x}^{(t)} + \boldsymbol{b}_1\right)\right)$$ (value of $\frac{\partial \boldsymbol{h}^{(t)}}{\partial \boldsymbol{h}^{(t-1)}}$)

If W_h is small, then this term gets vanishingly small as *i* and *j* get further apart

12 **Source**: "On the difficulty of training recurrent neural networks", Pascanu et al, 2013. http://proceedings.mlr.press/v28/pascanu13.pdf

Vanishing gradient proof sketch

- Consider matrix L2 norms:

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

- Pascanu et al showed that that if the largest eigenvalue of W_h is less than 1, then the gradient $\left\|\frac{\partial J^{(i)}(\theta)}{\partial \boldsymbol{h}^{(j)}}\right\|$ will shrink exponentially
  - Here the bound is 1 because we have sigmoid nonlinearity

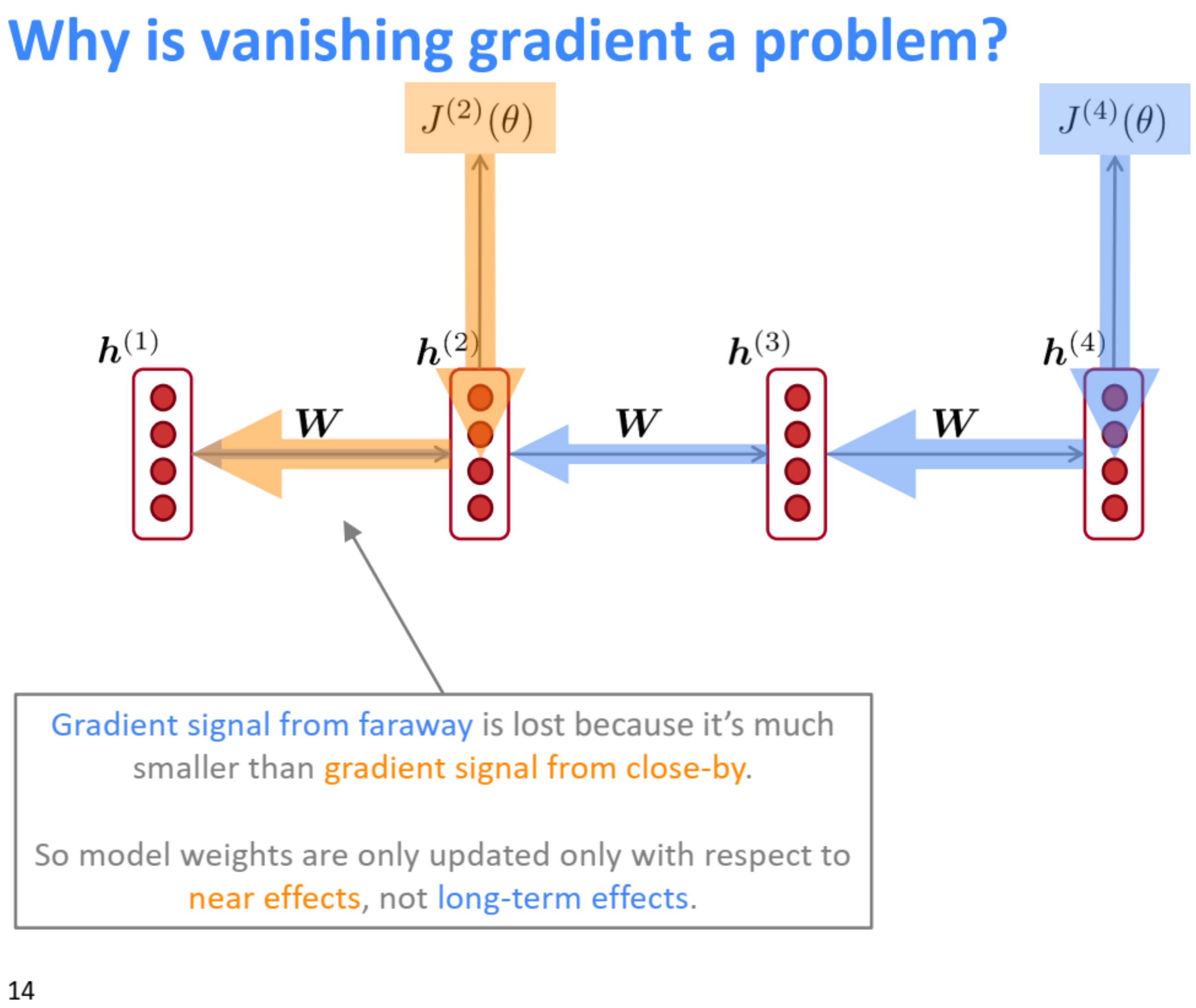
- There's a similar proof relating a largest eigenvalue >1 to exploding gradients

Source: "On the difficulty of training recurrent neural networks", Pascanu et al, 2013. http://proceedings.mlr.press/v28/pascanu13.pdf

But why is this a problem?

- The gradient from steps far away will be much smaller than gradients from close-by

→ The weights will only be updated with respect to close effects, not long-term !



Remark:

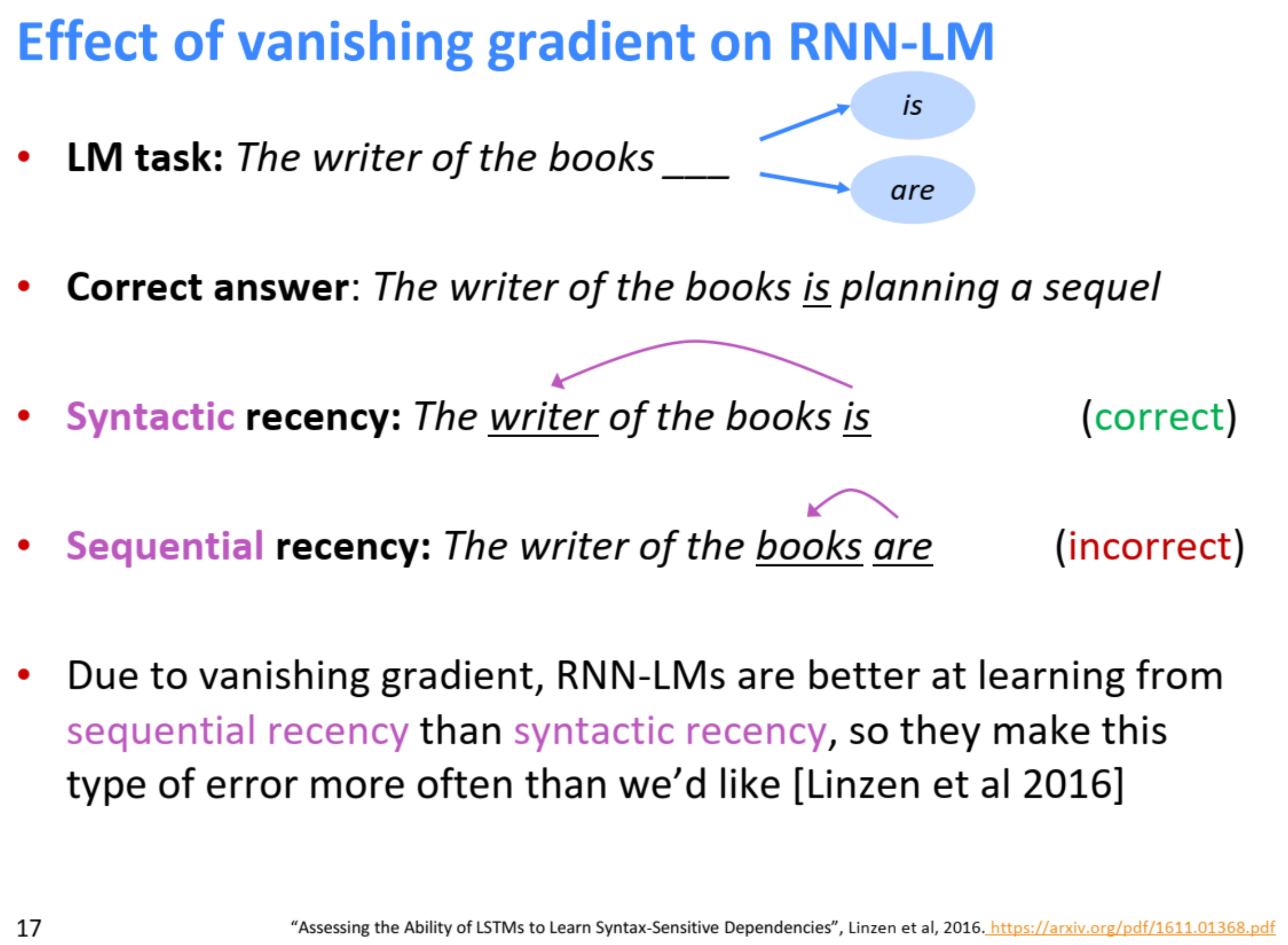
- We are studying $\frac{\partial J}{\partial h^{(i-1)}}$ because we have to calculate it to get $\frac{\partial J}{\partial W}$

Effect on RNN-LM

- Incapability of learning long-distance dependencies

- Syntactic recency (correct):
  Making dependencies to words close **in their syntax**
- Sequential recency (incorrect):
  Making dependencies to words close **spatially**

Unfortunately, RNN-LMs are better at learning sequential recency than syntactic recency:
(see example below)



Exploding Gradient Problem

- If our gradient is too big, the update term for our parameter becomes too big

→ Take update steps too large, and get stuck in a local optimum

→ Get **Inf** or **Nan** values

Solution: Gradient clipping

- Scale down gradients that are higher than some thereshold

LSTM

- Reason to exist = To solve the vanishing gradient problem
- A RNN with ability to learn / preserve information from many timesteps ago

What about a RNN with separate memory?

Structure

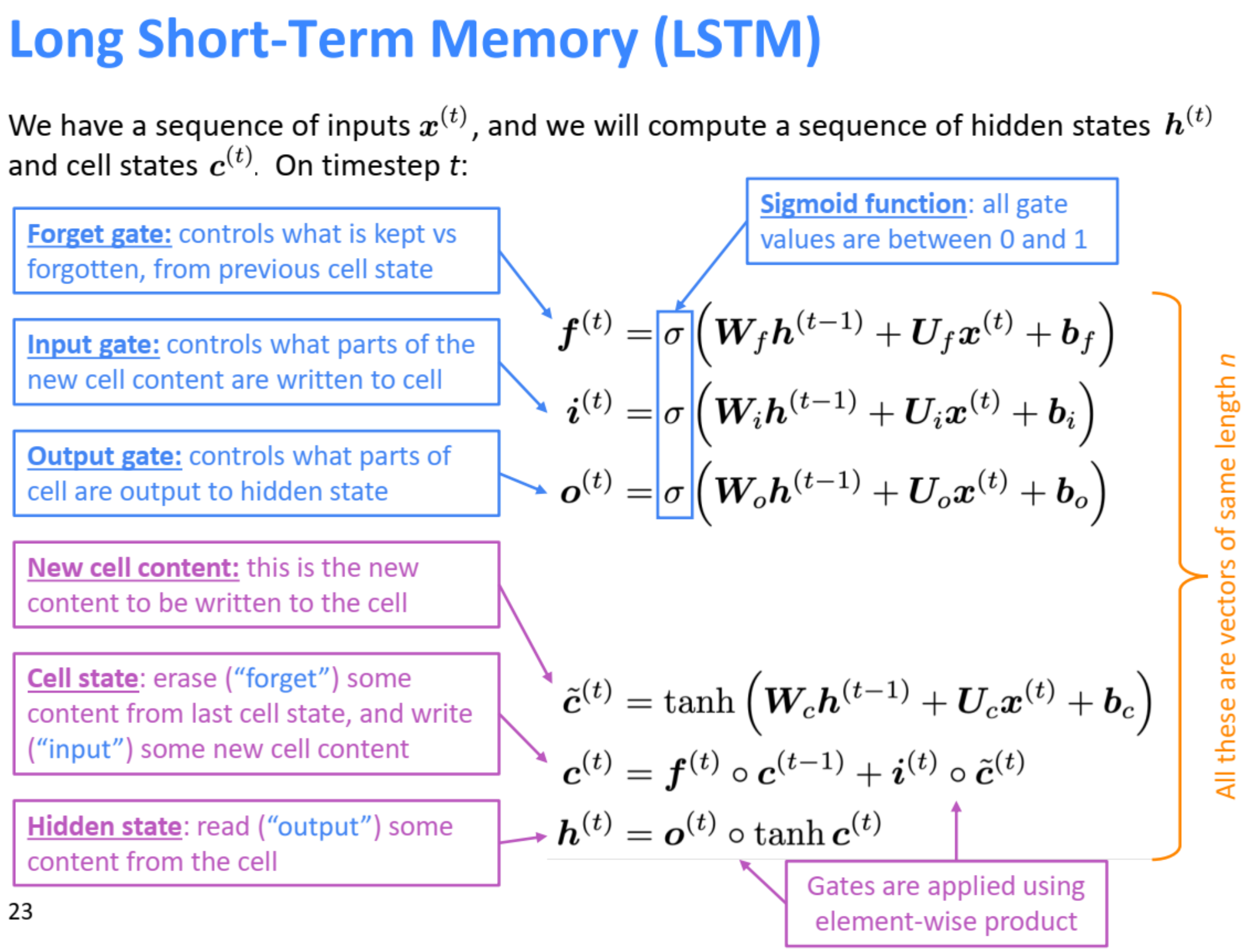
For each step t:

- a hidden state $h^{(t)}$: $(n, 1)$
- a cell state $c^{(t)}$: $(n, 1)$
  Stores long-term information

LSTM can erase, write and read information from cell !

This is done using gates of length n, whose values are in $[0, 1]$, thanks to our **sigmoid** friend:

- 1: open gate
- 0: closed gate



Cell state $c^{(t)}$ is the sum of:

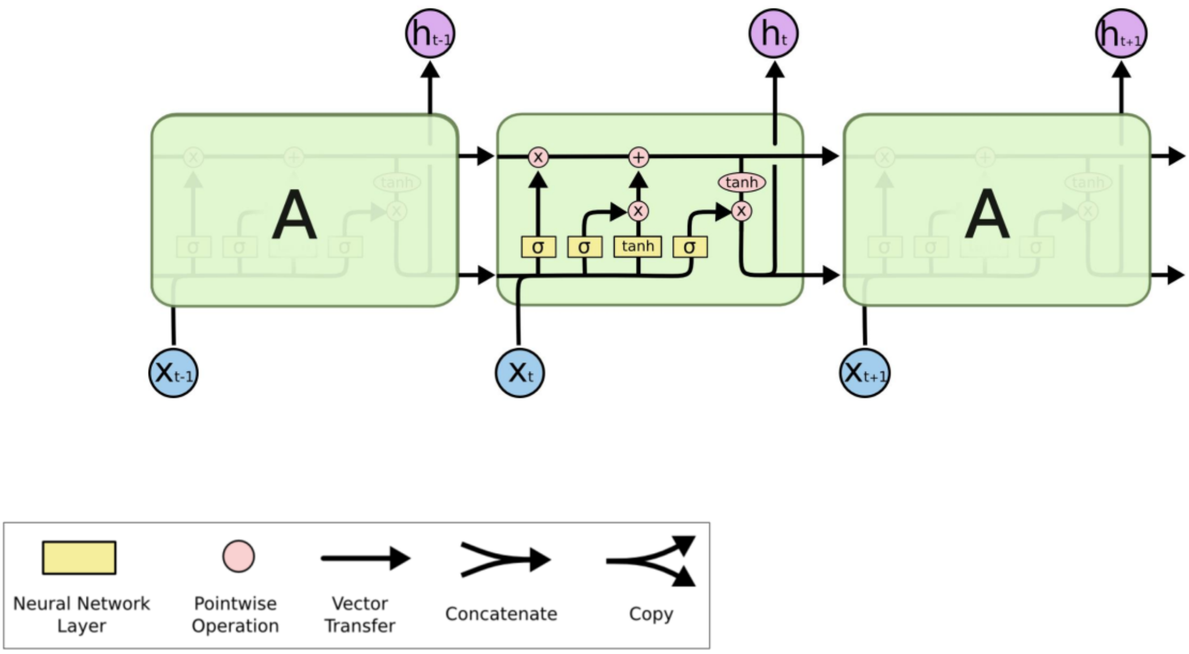
- previous cell state, masked by our forget gate
- our new cell, masked by our input gate

We can think of the hidden states as the ouput of the RNN

- These cell states ~ general memory, generally not accessible from the outside
- The hidden states will be passed on to the model

Long Short-Term Memory (LSTM)

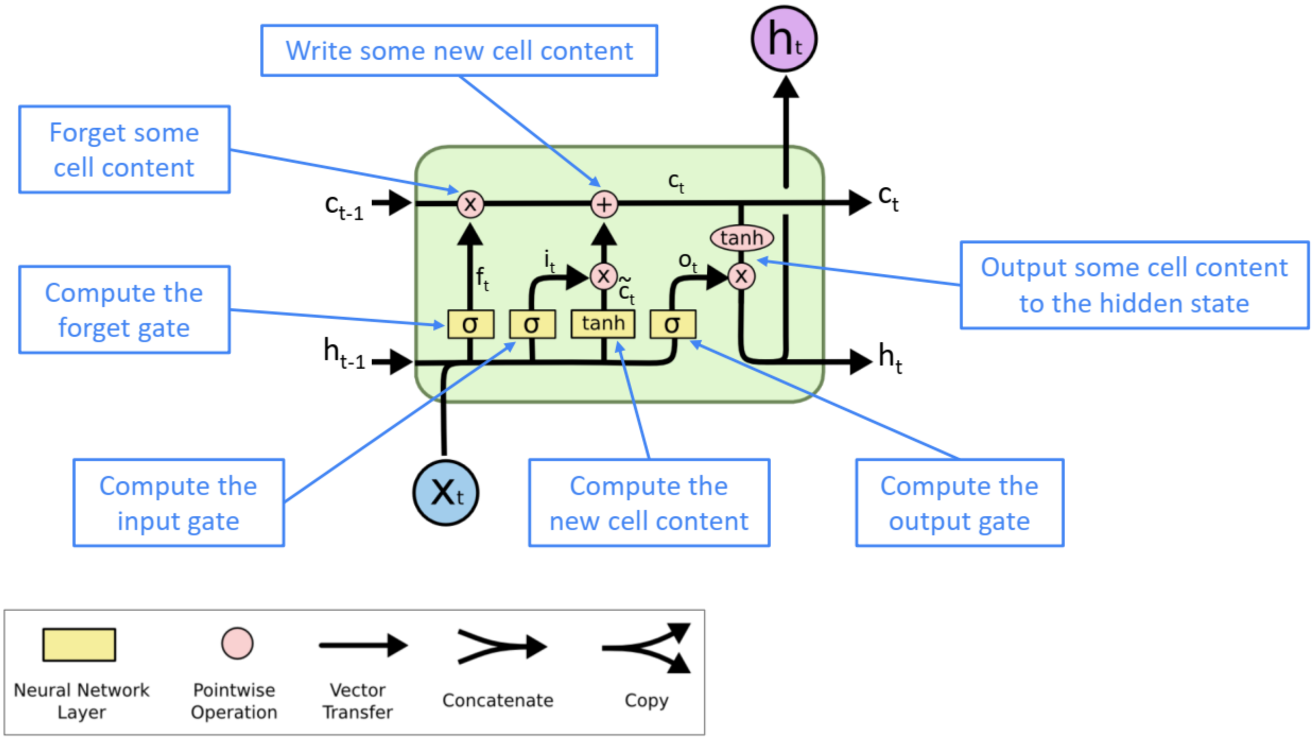
You can think of the LSTM equations visually like this:



Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Long Short-Term Memory (LSTM)

You can think of the LSTM equations visually like this:



25

Source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

It is possible to set the forget gate so that the LSTM remembers everything from each time step

⚠ We could still have vanishing/exploding gradients, even when using LSTMs !

- With RNN, the hidden states were a bottleneck:
  all gradients have to pass through them !!
- With LSTM, we can view the cell as a shortcut connection
  There is a potential route within the cell that doesn't make the gradient vanish !

Applications

- Handwriting recognition
- Speech recognition
- Machine translation
- Parsing
- Image captioning

However, more recently, Transformers seem to have taken over ! !

Gated Recurrent Units (GRU)

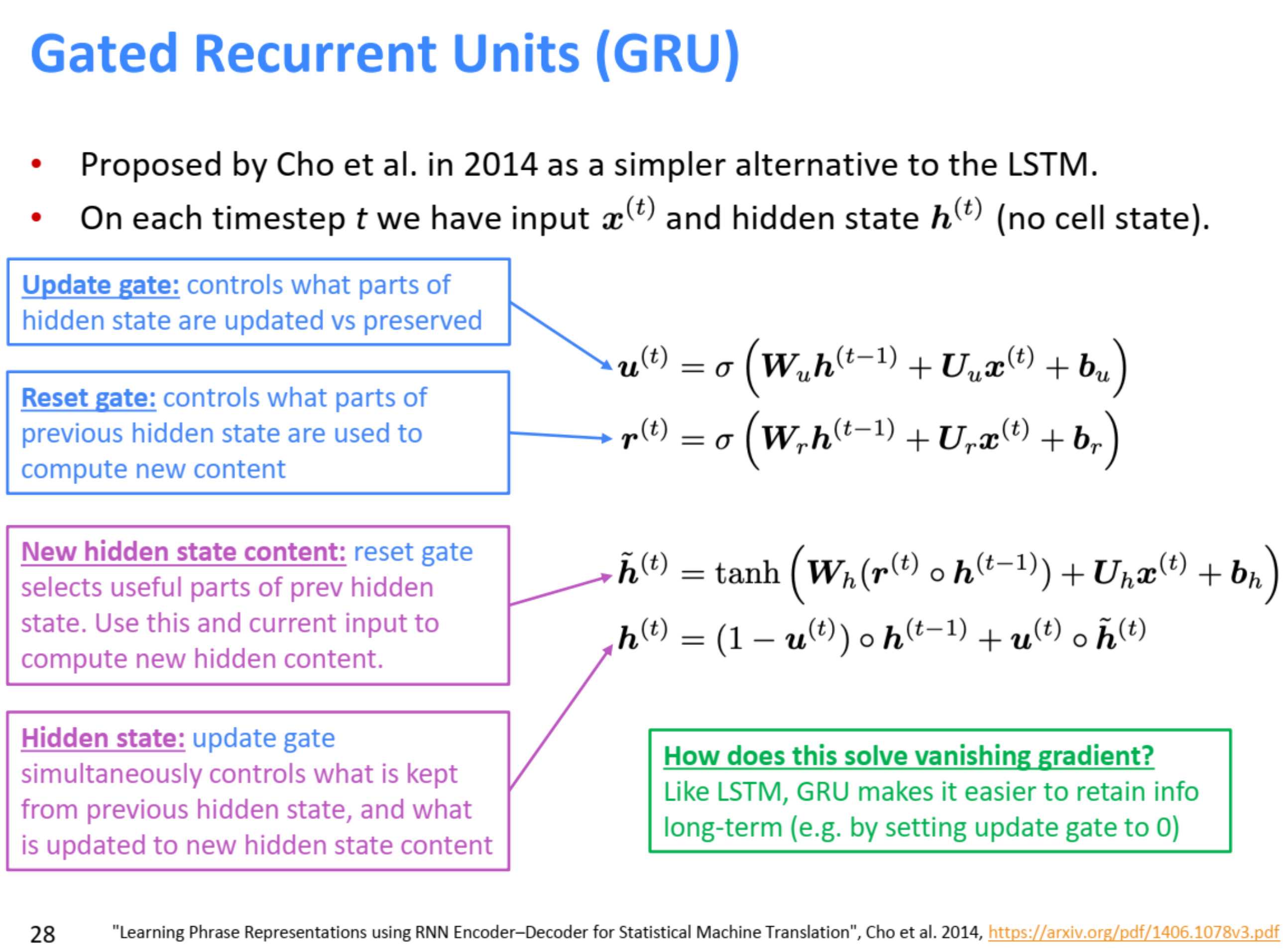
- Simpler alternative to LSTMs
- We have no cell state, only hidden states
- But we still have our cool gates!

Gates

- Update gate:
  controls what is updated or preserved
  ~ input & forget gates for LSTM

- Reset gate:
  selects which parts of the previous state are useful

- News hidden state = combination of previous hidden state, and computed new content



GRU compared to LSTM:

- Quicker
- Less parameters
- Similar performances

Other fixes

Gradient clipping

(See above)

Skip connections

- Create direct connections from lower layers to newer layers
  → Allowing the gradients to flow more easily

Example: ResNet

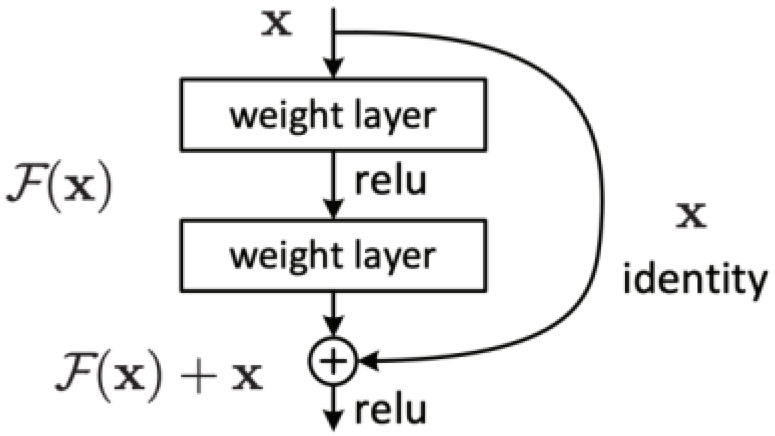


Figure 2. Residual learning: a building block.

- DenseNet:
  Going even further: connect every layer to each other!

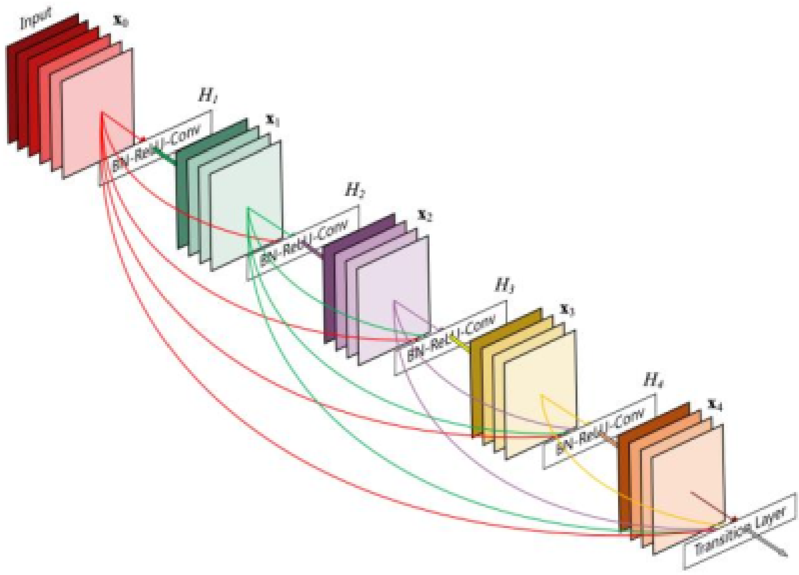


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

Recap 1

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
  - Multi-layer RNNs

Both of these are pretty simple

More fancy RNN variants

Bidirectional RNN

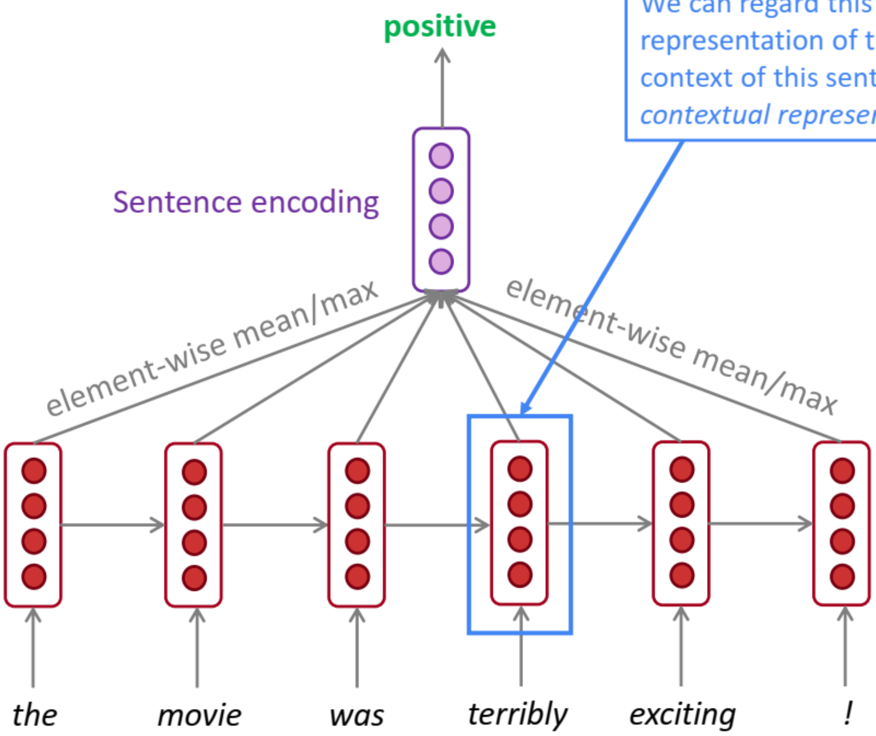
Motivation

- Our contextual representation only gets information from the left context !

We might want to get context coming from the right direction as well !

Bidirectional RNNs: motivation

<u>Task: Sentiment Classification</u>



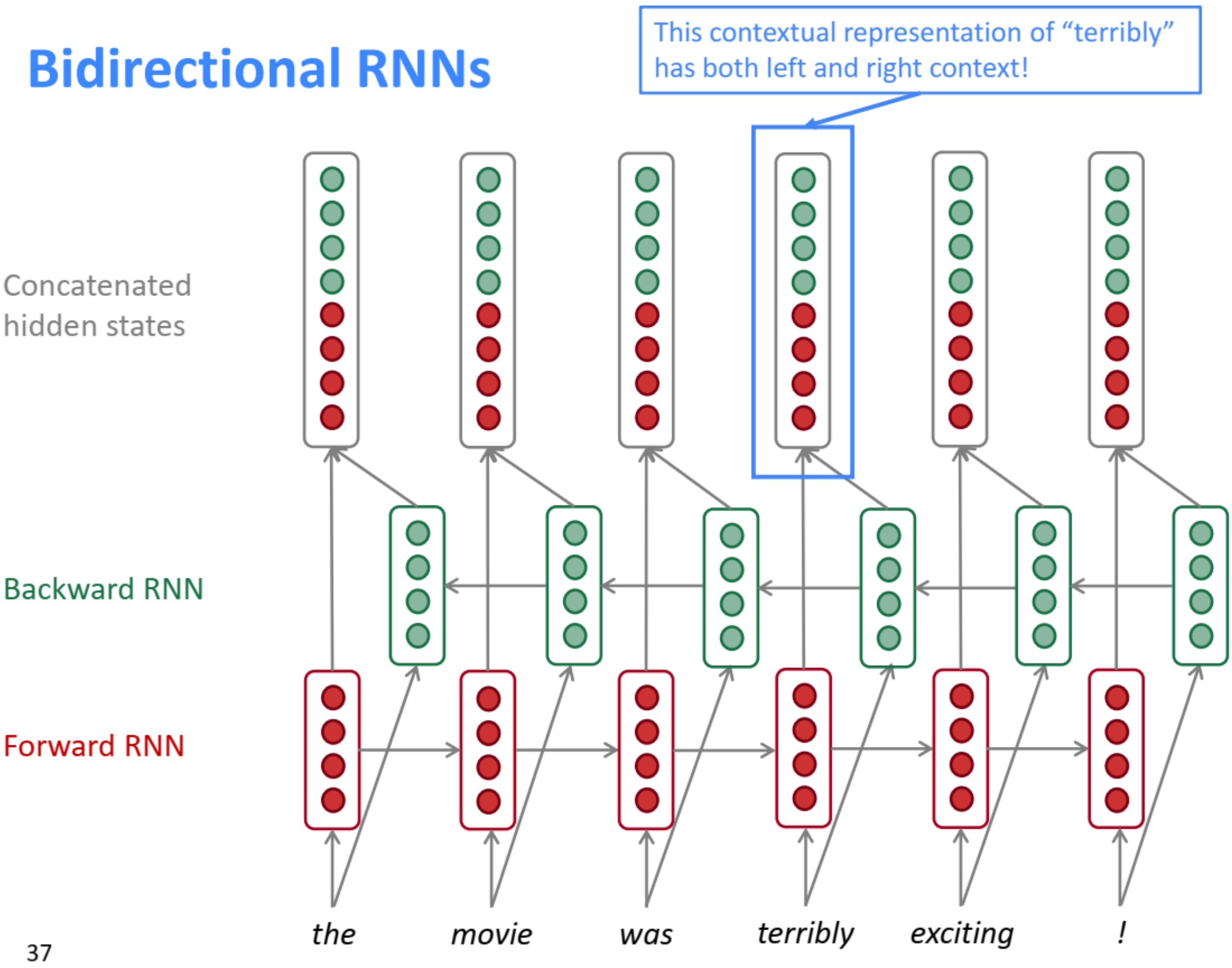
We can regard this hidden state as a representation of the word *"terribly"* in the context of this sentence. We call this a *contextual representation.*

These contextual representations only contain information about the *left* context (e.g. *"the movie was"*).

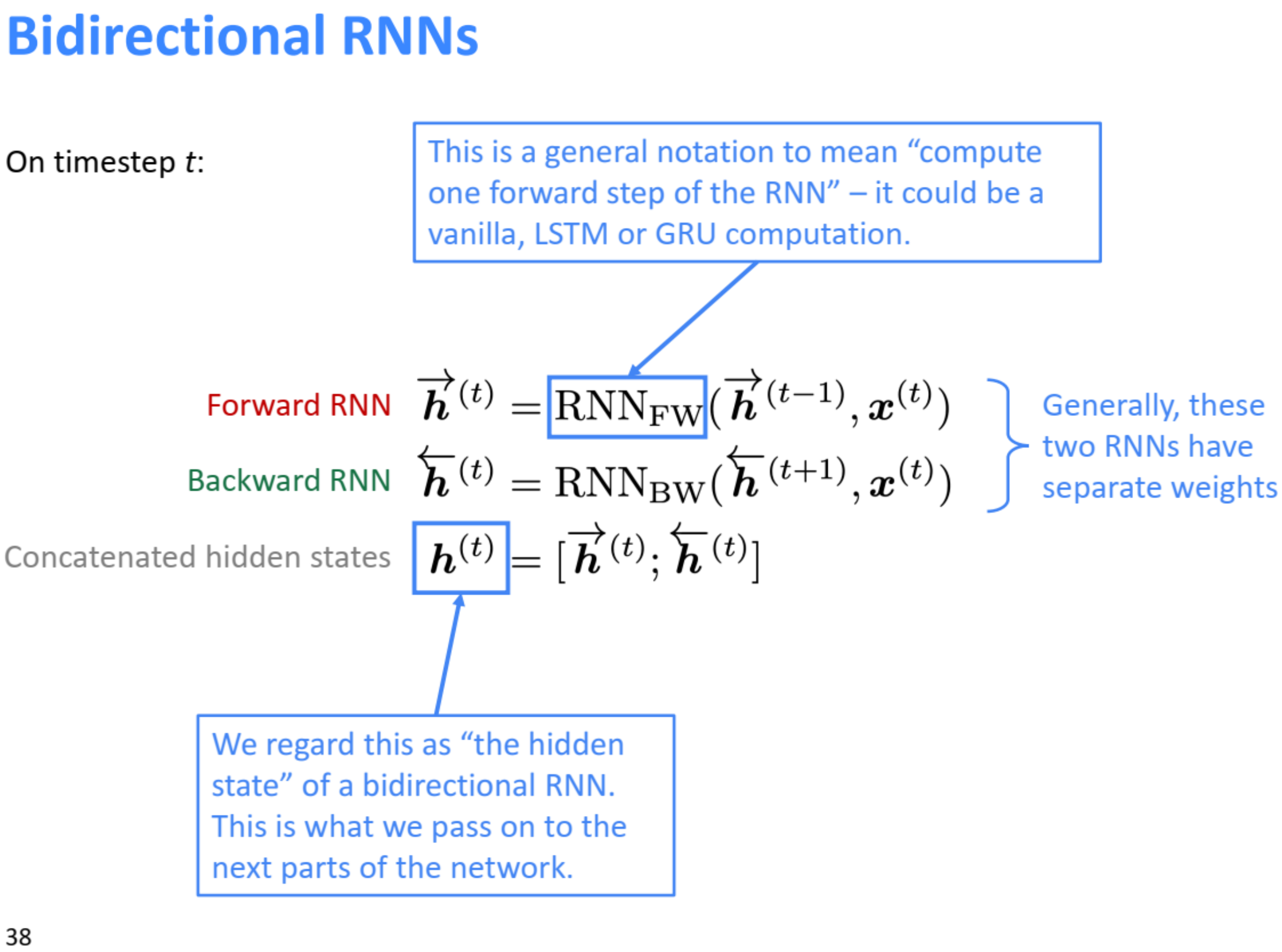
What about *right* context?

In this example, *"exciting"* is in the right context and this modifies the meaning of *"terribly"* (from negative to positive)

- We have 2 parallel RNN: one for each direction
- Hidden states = [Hidden state from left, Hidden state from right]


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
!
37

Notations



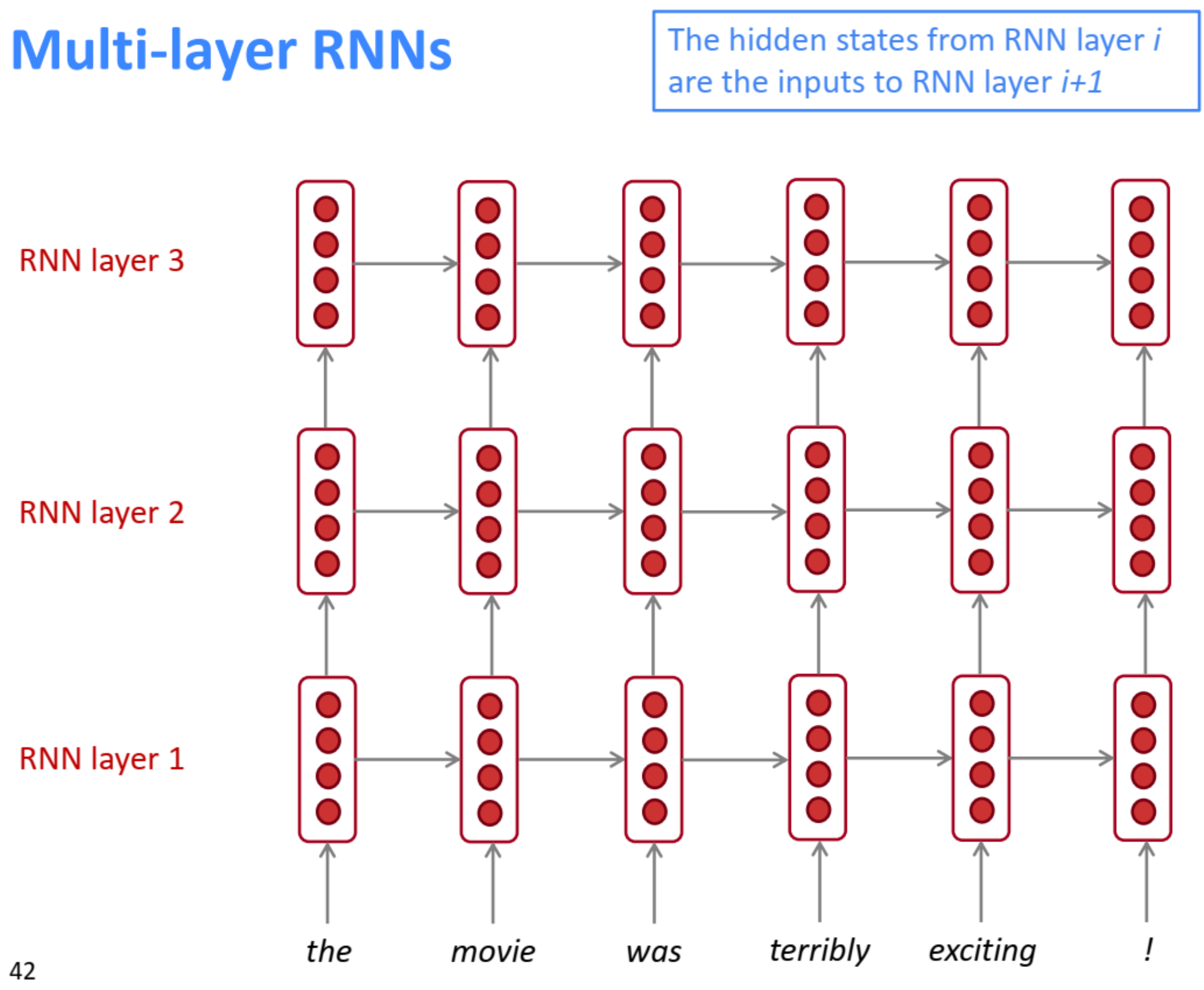
- The 2 RNNs are trained together, not separately

- Not useful for Language Modelling (as the Right context is missing, by definition!)

Multi-layer RNN

- Using multiple RNNs
- Allows to compute more complex representations

- Hidden states of layer i are the inputs to layer $i+1$

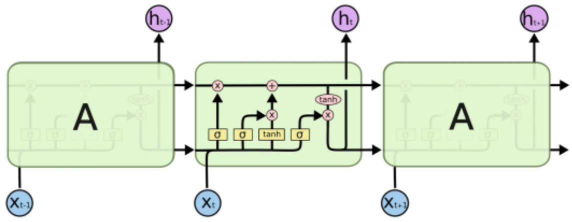


- RNNs have to be computed sequentially
  → Expensive to compute, can't go too deep

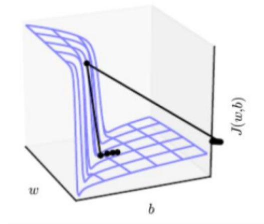
Recap 2: Practical Takeaways

In summary

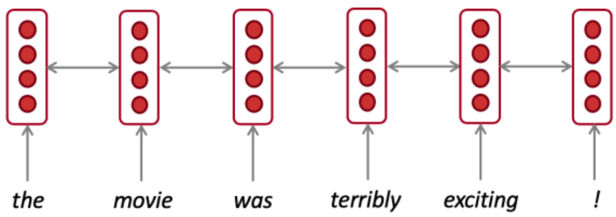
Lots of new information today! What are the practical takeaways?



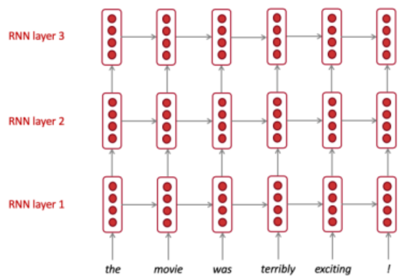
1. LSTMs are powerful but GRUs are faster



2. Clip your gradients



3. Use bidirectionality when possible



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