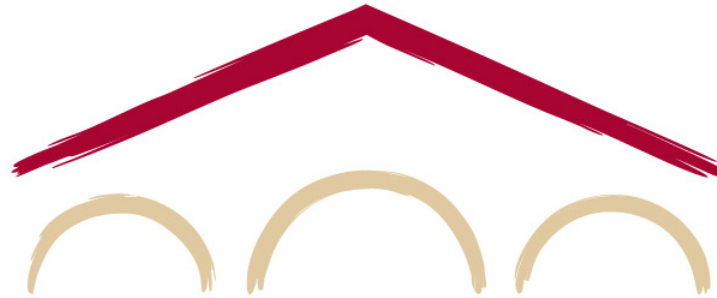


Natural Language Processing with Deep Learning

CS224N/Ling284



Christopher Manning

Lecture 6: LSTM RNNs and Neural Machine Translation

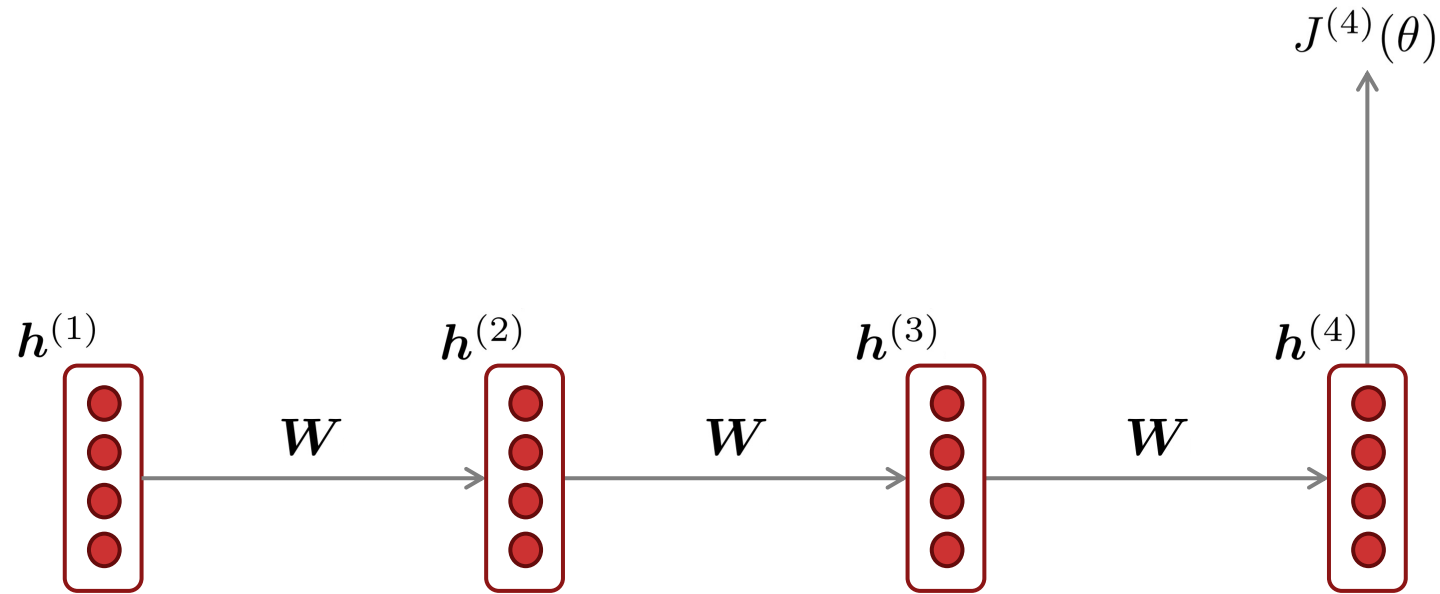
Lecture Plan

1. Exploding and vanishing gradients (15 mins)
 2. Long Short-Term Memory RNNs (LSTMs) (25 mins)
 3. Other uses of RNNs (5 mins)
 4. Bidirectional and multi-layer RNNs (15 mins)
 5. Machine translation (10 mins)
 6. Neural machine translation introduction (10 mins)
- **Final Projects:** Next Tuesday, part of the lecture is about choosing final projects
 - It's fine to just work on Ass3 and 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. It's still last year's information at present! DFP will be different!
 - It's great if you can line up your own mentor; we are also lining up some mentors

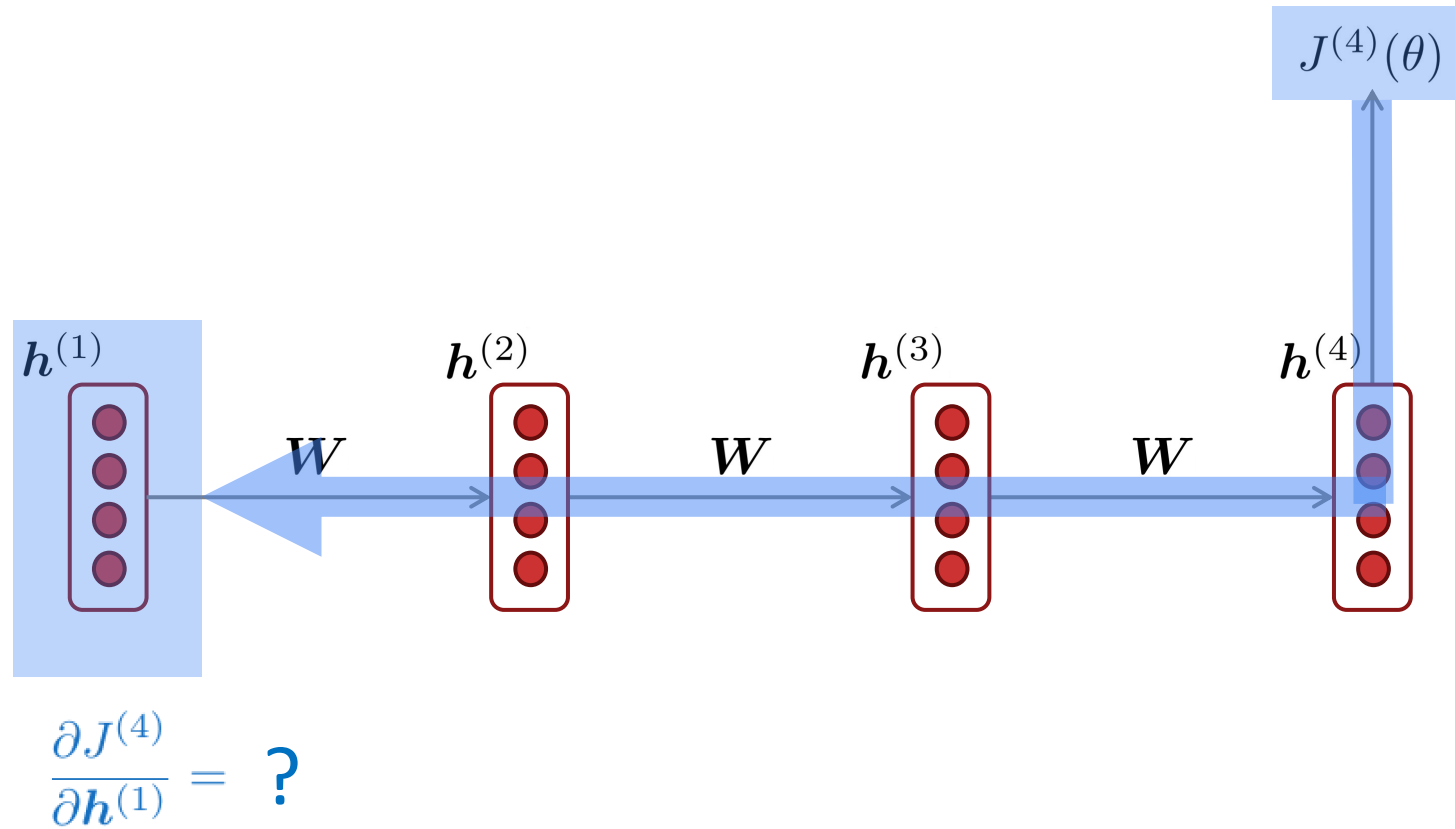
Recap

- **Language Model**: A system that predicts the next word
- **Recurrent Neural Network**: A family of neural networks that:
 - Take sequential input of any length; apply the same weights on each step
 - Can optionally produce output on each step
- **Recurrent Neural Network \neq Language Model**
 - RNNs can be used for many other things (see later)
- **Language Modeling** is a traditional subcomponent of many NLP tasks, all those involving generating text or estimating the probability of text:
 - Now everything in NLP is being rebuilt upon Language Modeling: GPT-3 is an LM!

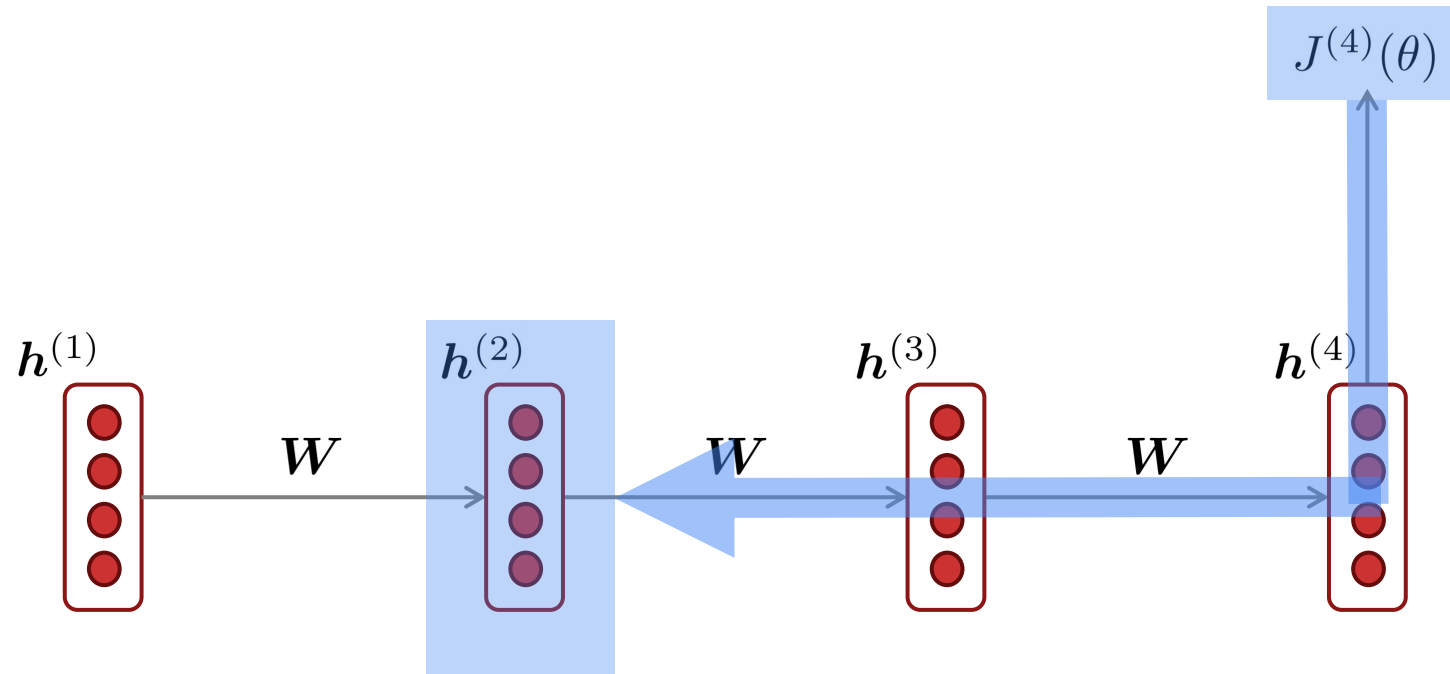
1. Problems with RNNs: Vanishing and Exploding Gradients



Vanishing gradient intuition



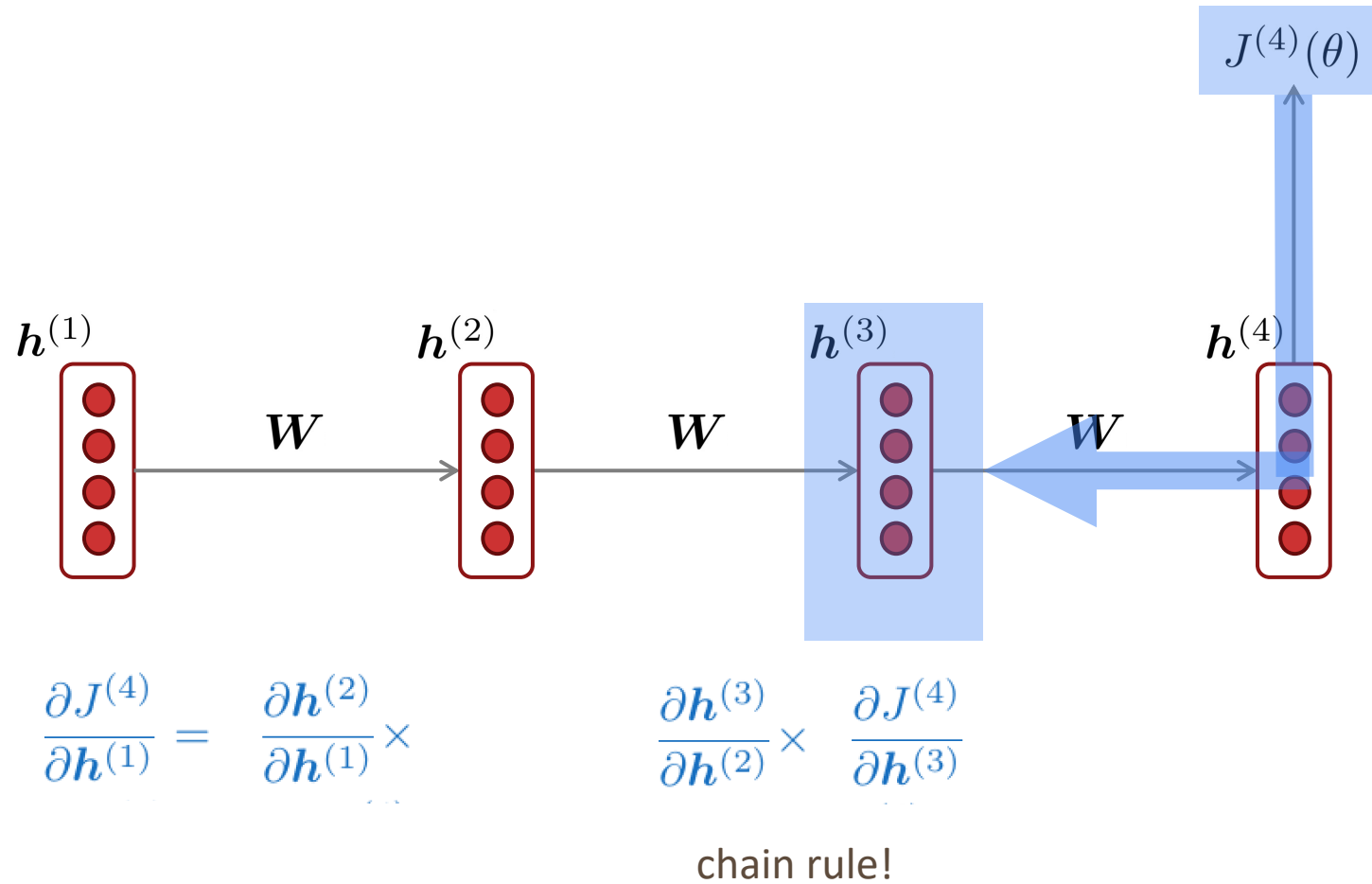
Vanishing gradient intuition



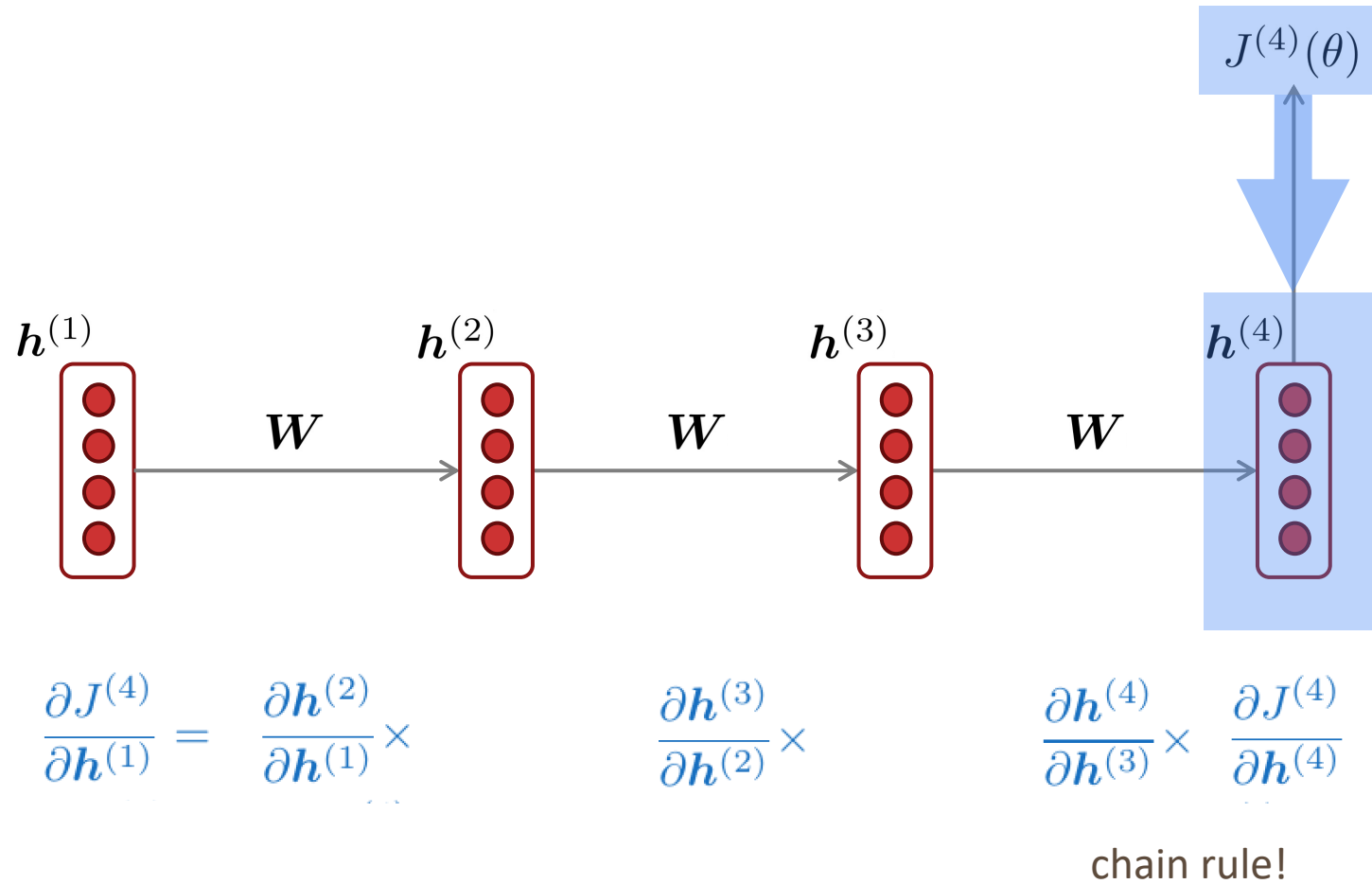
$$\frac{\partial J^{(4)}}{\partial h^{(1)}} = \frac{\partial h^{(2)}}{\partial h^{(1)}} \times \frac{\partial J^{(4)}}{\partial h^{(2)}}$$

chain rule!

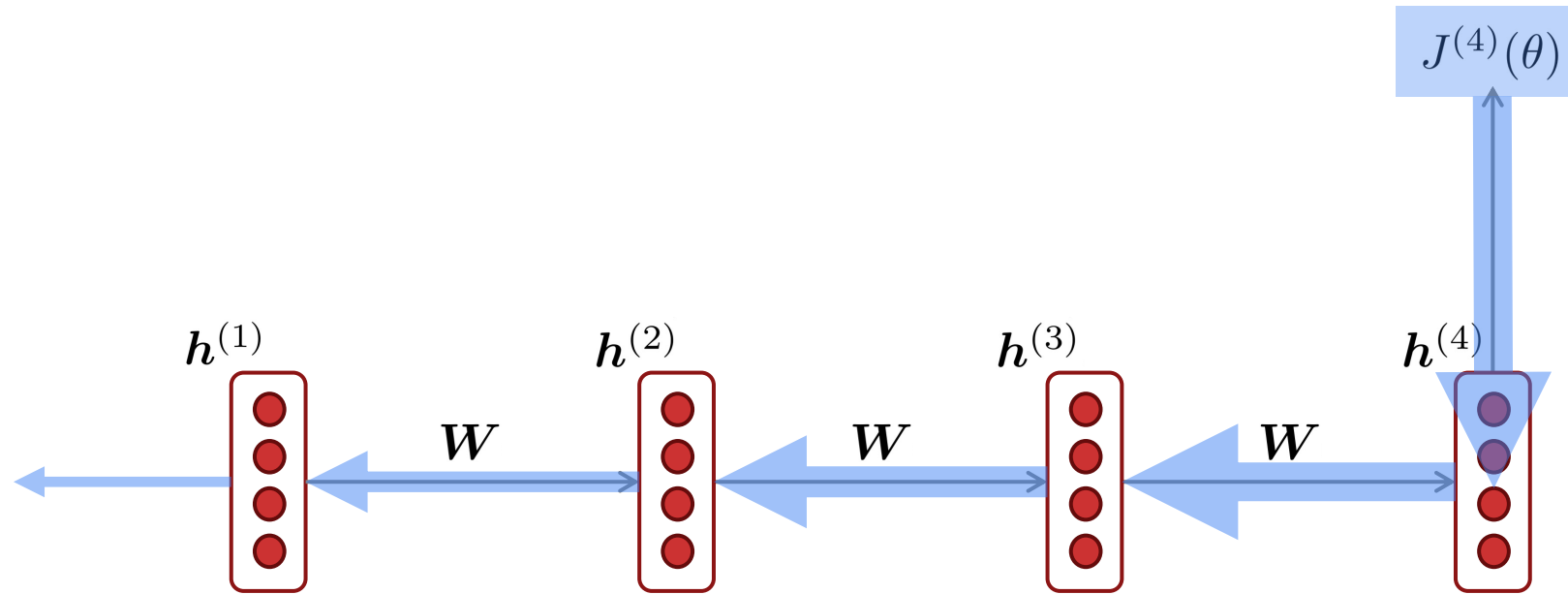
Vanishing gradient intuition



Vanishing gradient intuition



Vanishing gradient intuition



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

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 (linear case)

- Recall:
$$\mathbf{h}^{(t)} = \sigma \left(\mathbf{W}_h \mathbf{h}^{(t-1)} + \mathbf{W}_x \mathbf{x}^{(t)} + \mathbf{b}_1 \right)$$
- What if σ were the identity function, $\sigma(x) = x$?

$$\begin{aligned} \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(t-1)}} &= \text{diag} \left(\sigma' \left(\mathbf{W}_h \mathbf{h}^{(t-1)} + \mathbf{W}_x \mathbf{x}^{(t)} + \mathbf{b}_1 \right) \right) \mathbf{W}_h && \text{(chain rule)} \\ &= \mathbf{I} \mathbf{W}_h = \mathbf{W}_h \end{aligned}$$

- 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 . Let $\ell = i - j$

$$\begin{aligned} \frac{\partial J^{(i)}(\theta)}{\partial \mathbf{h}^{(j)}} &= \frac{\partial J^{(i)}(\theta)}{\partial \mathbf{h}^{(i)}} \prod_{j < t \leq i} \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(t-1)}} && \text{(chain rule)} \\ &= \frac{\partial J^{(i)}(\theta)}{\partial \mathbf{h}^{(i)}} \prod_{j < t \leq i} \mathbf{W}_h = \frac{\partial J^{(i)}(\theta)}{\partial \mathbf{h}^{(i)}} \boxed{\mathbf{W}_h^\ell} && \text{(value of } \frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{h}^{(t-1)}} \text{)} \end{aligned}$$


If \mathbf{W}_h is “small”, then this term gets exponentially problematic as ℓ becomes large

Vanishing gradient proof sketch (linear case)

- What's wrong with W_h^ℓ ?

- Consider if the eigenvalues of W_h are all less than 1:

$$\lambda_1, \lambda_2, \dots, \lambda_n < 1$$

q_1, q_2, \dots, q_n (eigenvectors)

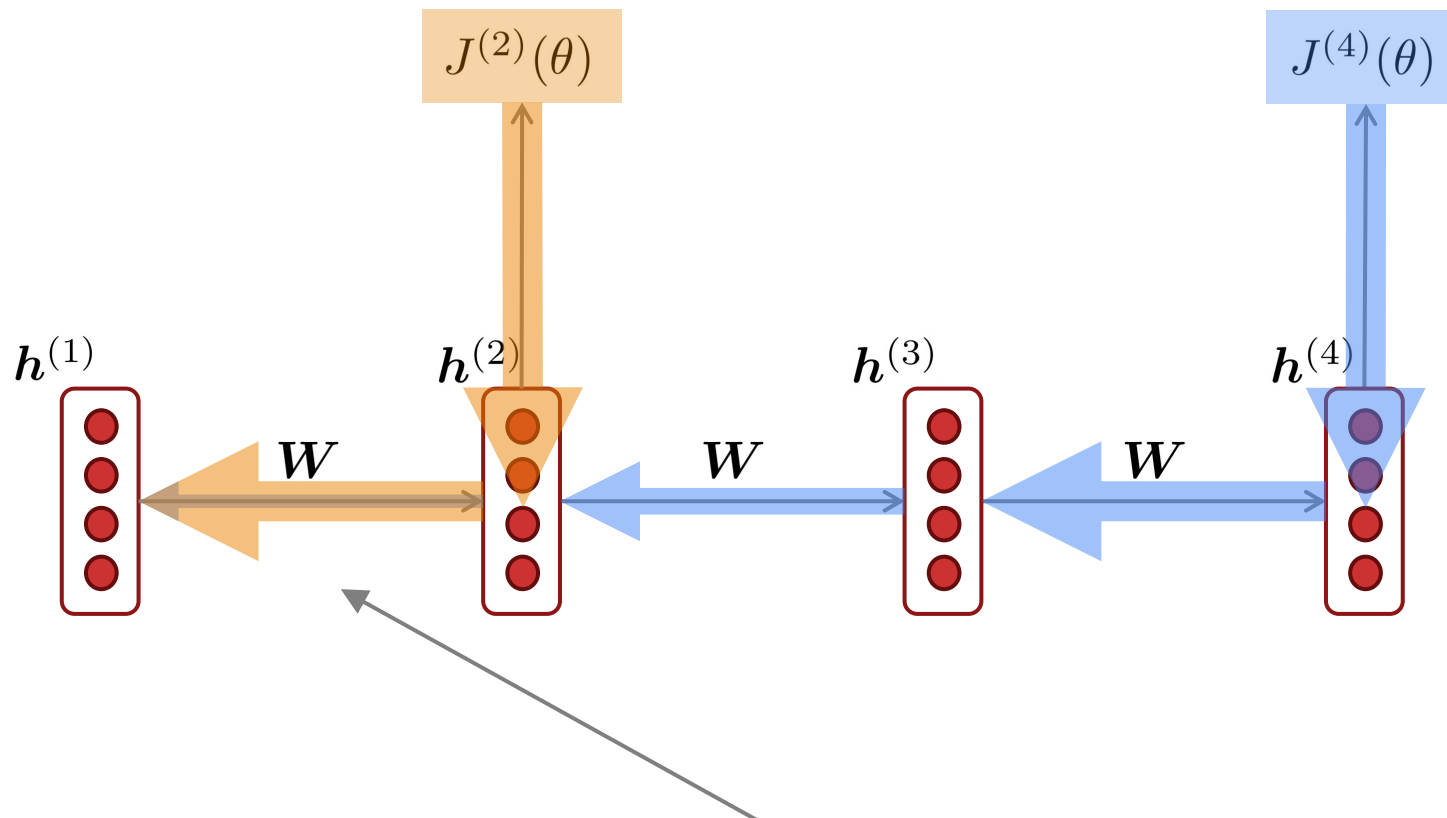
- We can write $\frac{\partial J^{(i)}(\theta)}{\partial h^{(i)}} W_h^\ell$ using the eigenvectors of W_h as a basis:

$$\frac{\partial J^{(i)}(\theta)}{\partial h^{(i)}} W_h^\ell = \sum_{i=1}^n c_i \lambda_i^\ell q_i \approx \mathbf{0} \text{ (for large } \ell \text{)}$$

Approaches 0 as ℓ grows, so gradient vanishes

- What about nonlinear activations σ (i.e., what we use?)
 - Pretty much the same thing, except the proof requires $\lambda_i < \gamma$ for some γ dependent on dimensionality and σ

Why is vanishing gradient a problem?



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

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

Effect of vanishing gradient on RNN-LM

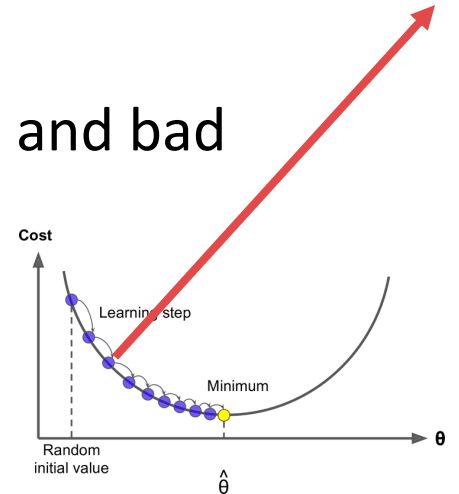
- **LM task:** *When she tried to print her tickets, she found that the printer was out of toner. She went to the stationery store to buy more toner. It was very overpriced. After installing the toner into the printer, she finally printed her _____*
- To learn from this training example, the RNN-LM needs to **model the dependency** between “*tickets*” on the 7th step and the target word “*tickets*” at the end.
- But if the gradient is small, the model **can't learn this dependency**
 - So, the model is **unable to predict similar long-distance dependencies** at test time
- In practice a simple RNN will only condition ~7 tokens back **[vague rule-of-thumb]**

Why is exploding gradient a problem?

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

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

- This can cause **bad updates**: we take too large a step and reach a weird and bad parameter configuration (with large loss)
 - You think you've found a hill to climb, but suddenly you're in lowa
- 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  
     $\hat{\mathbf{g}} \leftarrow \frac{threshold}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}}$   
end if
```

- **Intuition**: take a step in the same direction, but a smaller step
- In practice, **remembering to clip gradients is important**, but exploding gradients are an easy problem to solve

How to fix the vanishing gradient problem?

- 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


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

- Could we design an RNN with separate memory which is added to?

2. LSTMs: Apple WWDC Keynote 2016



Long Short-Term Memory RNNs (LSTMs)

- A type of RNN proposed by Hochreiter and Schmidhuber in 1997 as a solution to the problem of vanishing gradients
 - Everyone cites that paper but really a crucial part of the modern LSTM is from Gers et al. (2000) 
- Only started to be recognized as promising through the work of S's student Alex Graves c. 2006
 - Work in which he also invented CTC (connectionist temporal classification) for speech recognition
- But only really became well-known after Hinton brought it to Google in 2013
 - Following Graves having been a postdoc with Hinton

Hochreiter and Schmidhuber, 1997. Long short-term memory. <https://www.bioinf.jku.at/publications/older/2604.pdf>

Gers, Schmidhuber, and Cummins, 2000. Learning to Forget: Continual Prediction with LSTM. <https://dl.acm.org/doi/10.1162/089976600300015015>

Graves, Fernandez, Gomez, and Schmidhuber, 2006. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural nets. https://www.cs.toronto.edu/~graves/icml_2006.pdf

Long Short-Term Memory RNNs (LSTMs)

- 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**
 - The LSTM can **read**, **erase**, and **write** information from the cell
 - The cell becomes conceptually rather like RAM in a computer
- The selection of which information is erased/written/read is controlled by three corresponding **gates**
 - The gates are also vectors of length n
 - On each timestep, each element of the gates can be **open** (1), **closed** (0), or somewhere in-between
 - The gates are **dynamic**: their value is computed based on the current context

Long Short-Term Memory (LSTM)

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

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

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

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

New cell content: this is the new content to be written to the cell

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

Sigmoid function: all gate values are between 0 and 1

$$f^{(t)} = \sigma \left(W_f h^{(t-1)} + U_f x^{(t)} + b_f \right)$$

$$i^{(t)} = \sigma \left(W_i h^{(t-1)} + U_i x^{(t)} + b_i \right)$$

$$o^{(t)} = \sigma \left(W_o h^{(t-1)} + U_o x^{(t)} + b_o \right)$$

$$\tilde{c}^{(t)} = \tanh \left(W_c h^{(t-1)} + U_c x^{(t)} + b_c \right)$$

$$c^{(t)} = f^{(t)} \circ c^{(t-1)} + i^{(t)} \circ \tilde{c}^{(t)}$$

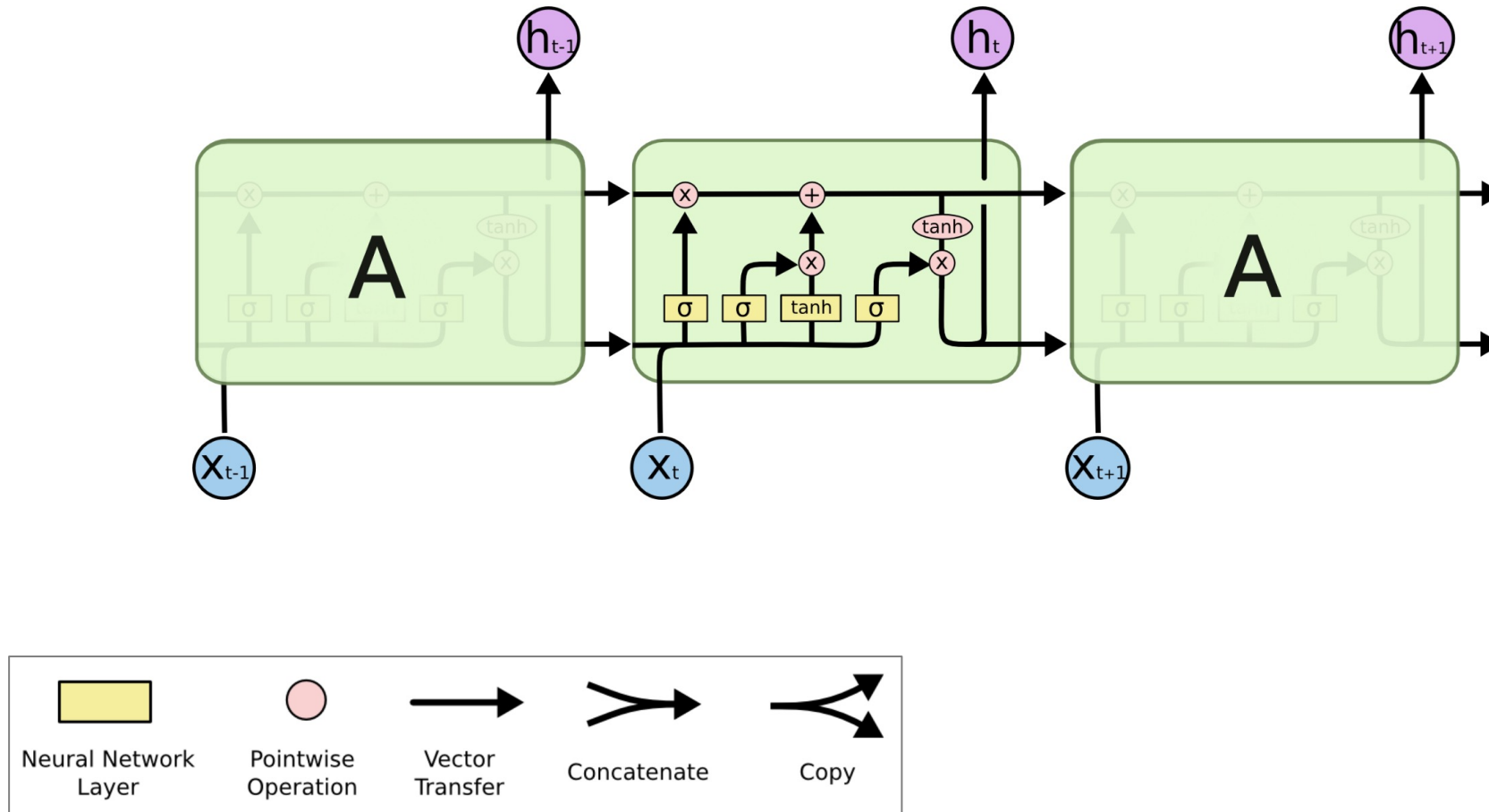
$$h^{(t)} = o^{(t)} \circ \tanh c^{(t)}$$

All these are vectors of same length n

Gates are applied using element-wise (or Hadamard) product: \odot

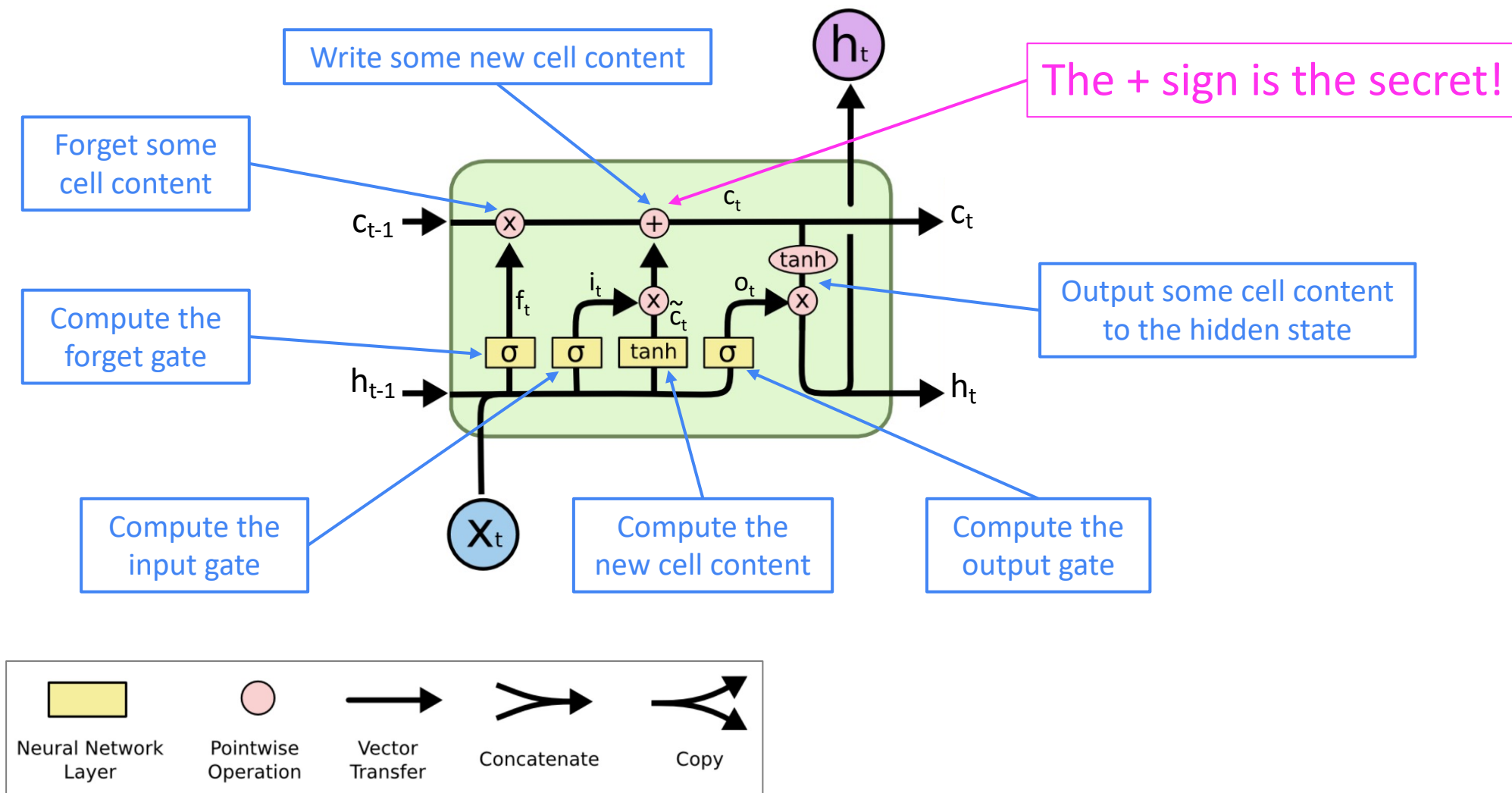
Long Short-Term Memory (LSTM)

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?

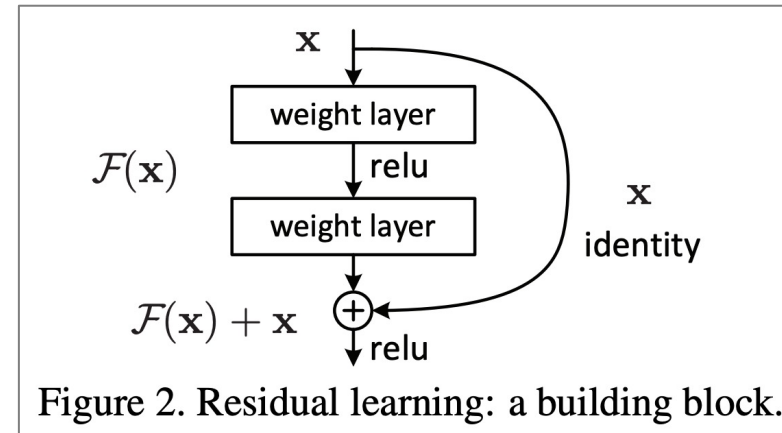
- The LSTM architecture makes it much easier for an RNN to preserve information over many timesteps
 - e.g., if the forget gate is set to 1 for a cell dimension and the input gate set to 0, then the information of that cell is preserved indefinitely.
 - In contrast, it's harder for a vanilla RNN to learn a recurrent weight matrix W_h that preserves info in the hidden state
 - In practice, you get about 100 timesteps rather than about 7
- However, there are alternative ways of creating more direct and linear pass-through connections in models for long distance dependencies

Is vanishing/exploding gradient just an RNN problem?

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

For example:

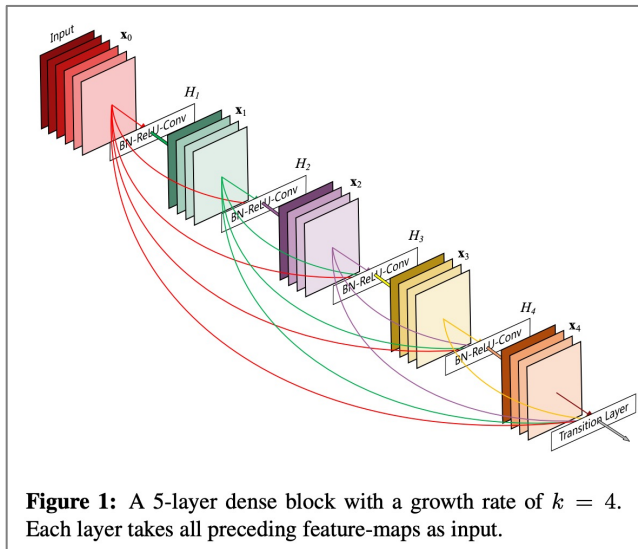
- **Residual connections** aka “ResNet”
- Also known as **skip-connections**
- The **identity connection** **preserves information** by default
- This makes **deep** networks much **easier to train**



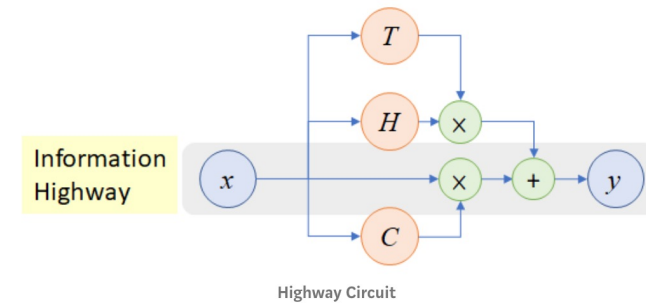
Is vanishing/exploding gradient just a RNN problem?

Other methods:

- **Dense connections** aka “DenseNet”
- Directly connect each layer to all future layers!



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



- **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]

“Densely Connected Convolutional Networks”, Huang et al, 2017. <https://arxiv.org/pdf/1608.06993.pdf>

“Highway Networks”, Srivastava et al, 2015. <https://arxiv.org/pdf/1505.00387.pdf>

“Learning Long-Term Dependencies with Gradient Descent is Difficult”, Bengio et al. 1994, <http://ai.dinfo.unifi.it/paolo//ps/tnn-94-gradient.pdf>

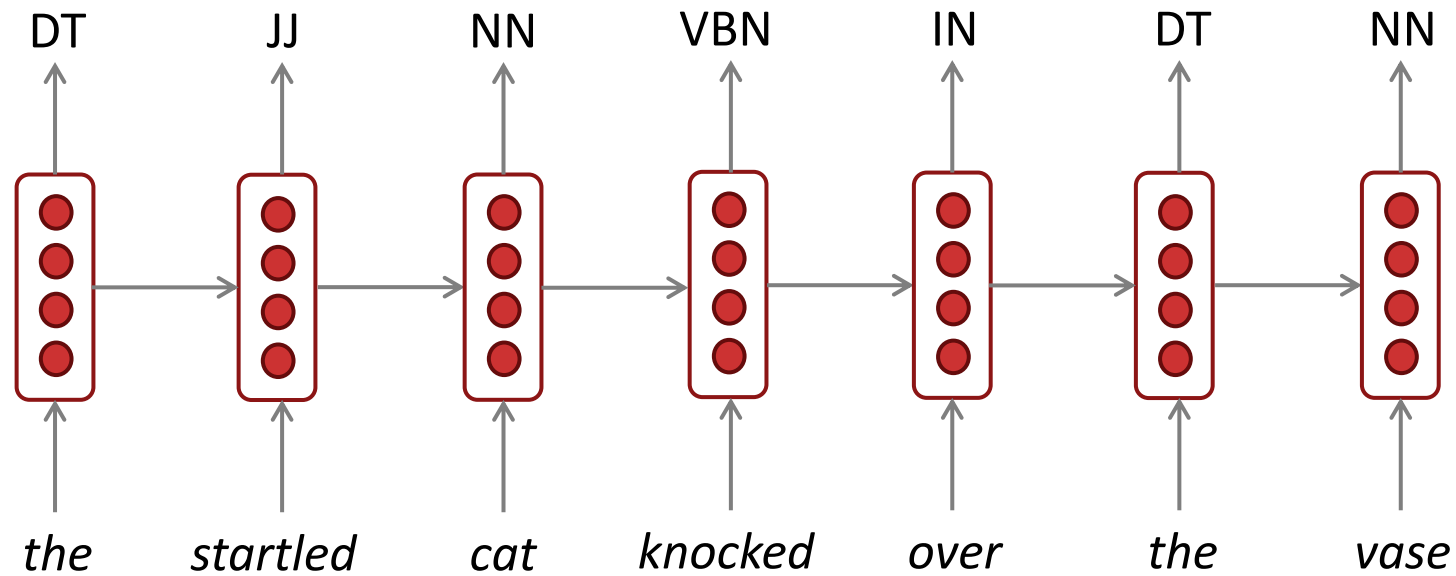
LSTMs: real-world success

- In 2013–2015, LSTMs started achieving state-of-the-art results
 - Successful tasks include handwriting recognition, speech recognition, machine translation, parsing, and image captioning, as well as language models
 - LSTMs became the dominant approach for most NLP tasks
- Now (2019–2023), Transformers have become dominant for all tasks
 - For example, in WMT (a Machine Translation conference + competition):
 - In WMT 2014, there were 0 neural machine translation systems (!)
 - In WMT 2016, the summary report contains “RNN” 44 times (and these systems won)
 - In WMT 2019: “RNN” 7 times, “Transformer” 105 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>
Source: "Findings of the 2019 Conference on Machine Translation (WMT19)", Barrault et al. 2019, <http://www.statmt.org/wmt18/pdf/WMT028.pdf>

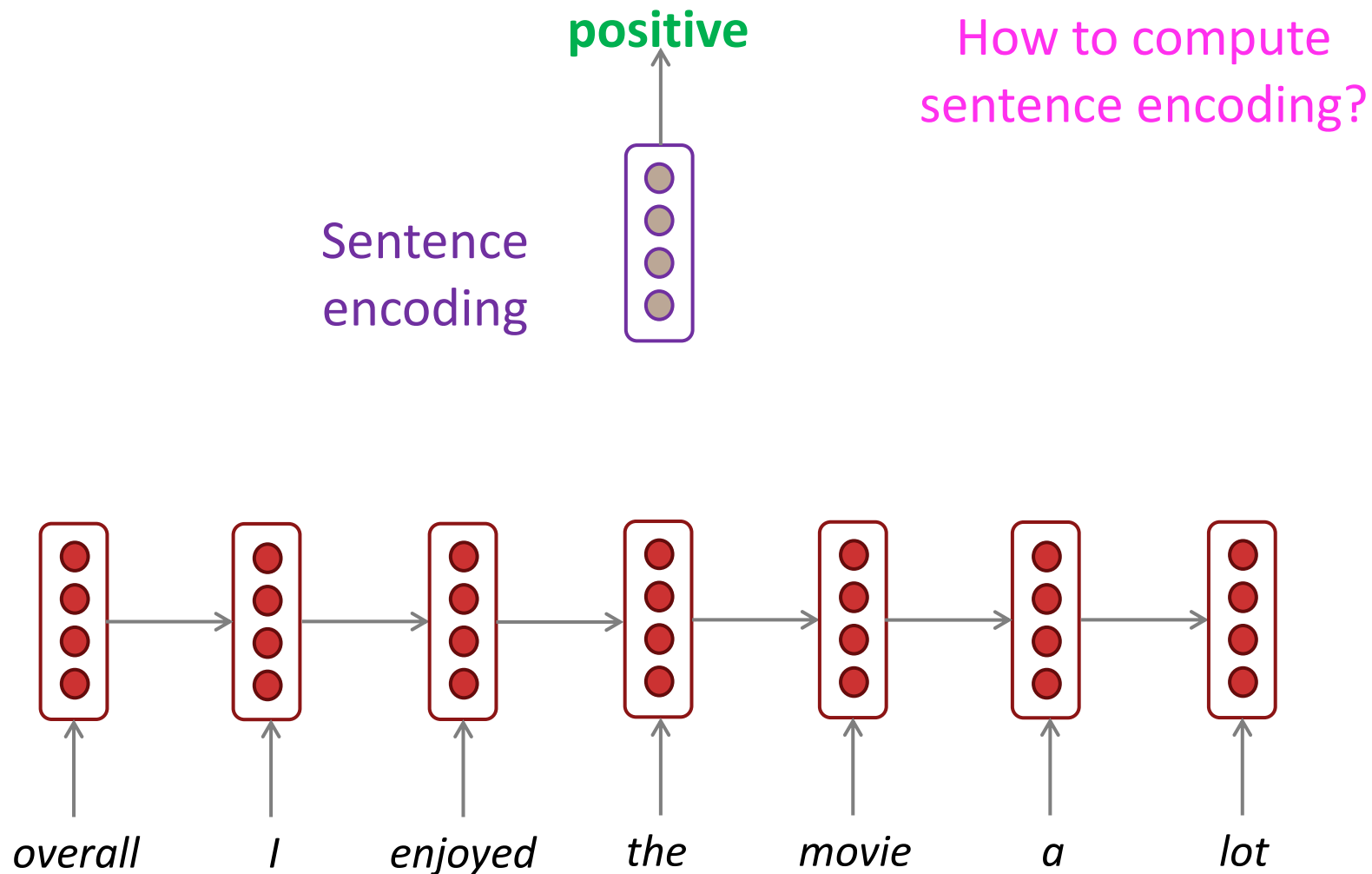
3. Other RNN uses: RNNs can be used for sequence tagging

e.g., part-of-speech tagging, named entity recognition



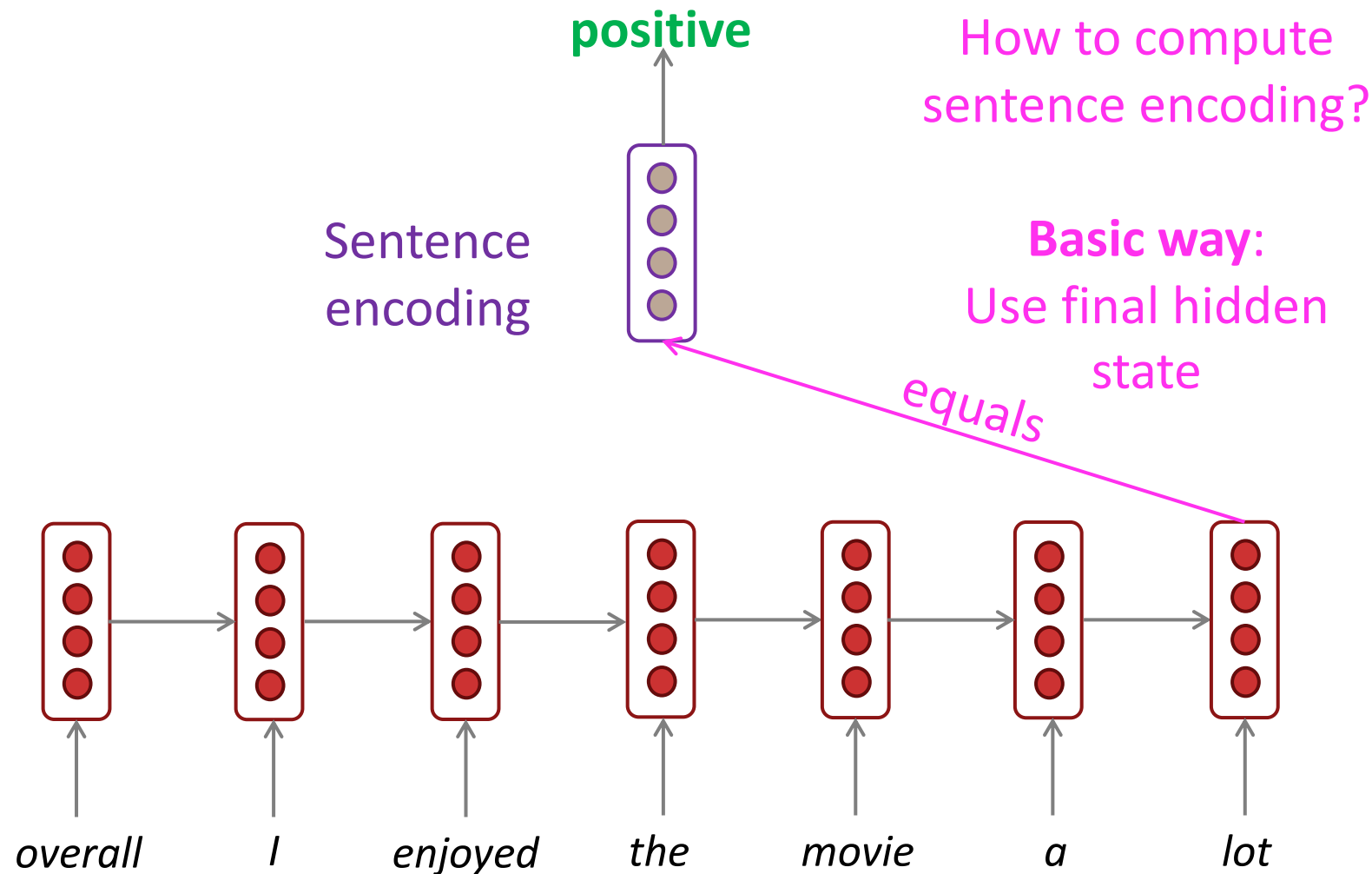
RNNs can be used as a sentence encoder model

e.g., for sentiment classification



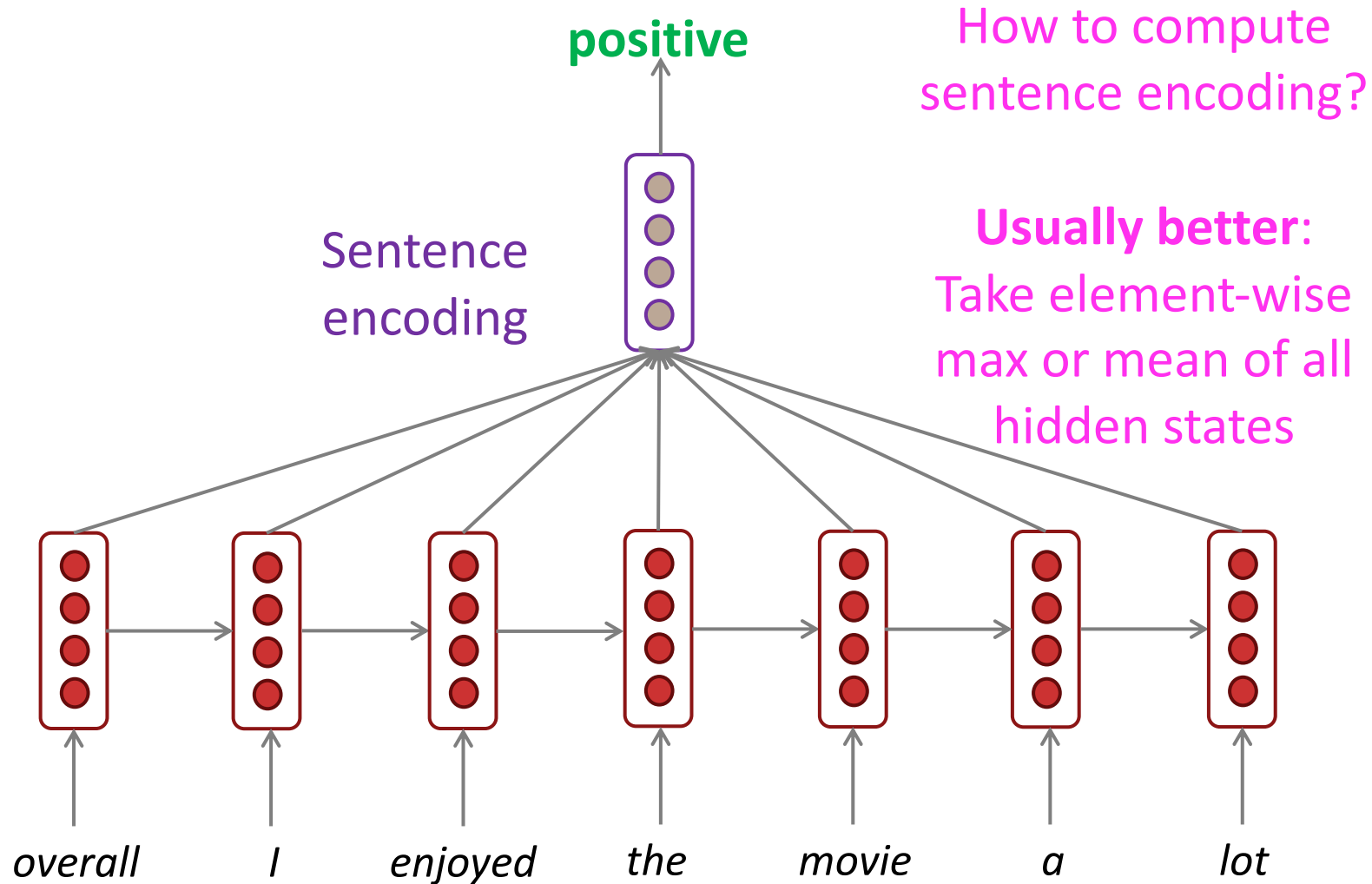
RNNs can be used as a sentence encoder model

e.g., for sentiment classification



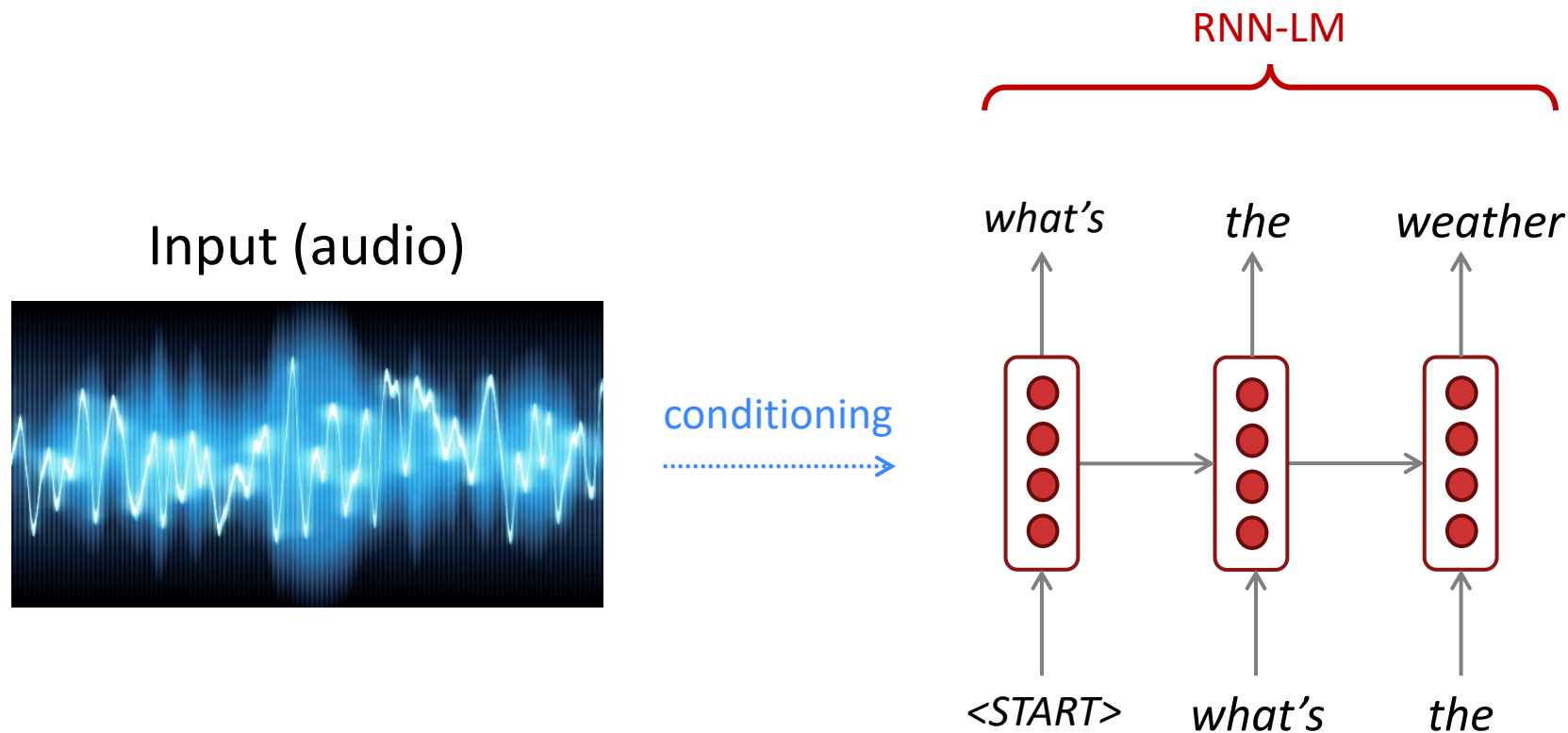
RNNs can be used as a sentence encoder model

e.g., for sentiment classification



RNN-LMs can be used to generate text based on other information

e.g., speech recognition, machine translation, summarization

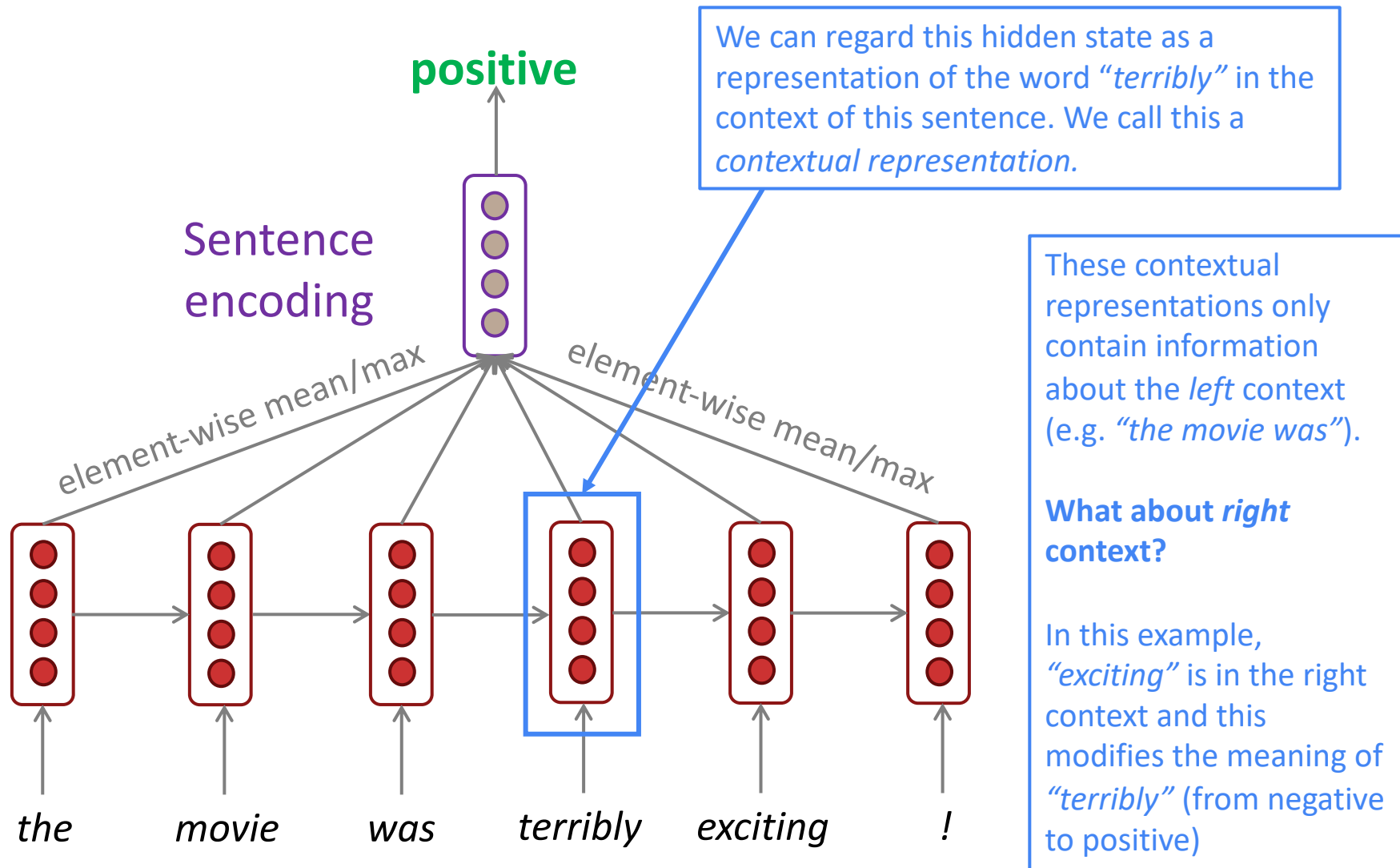


This is an example of a *conditional language model*.

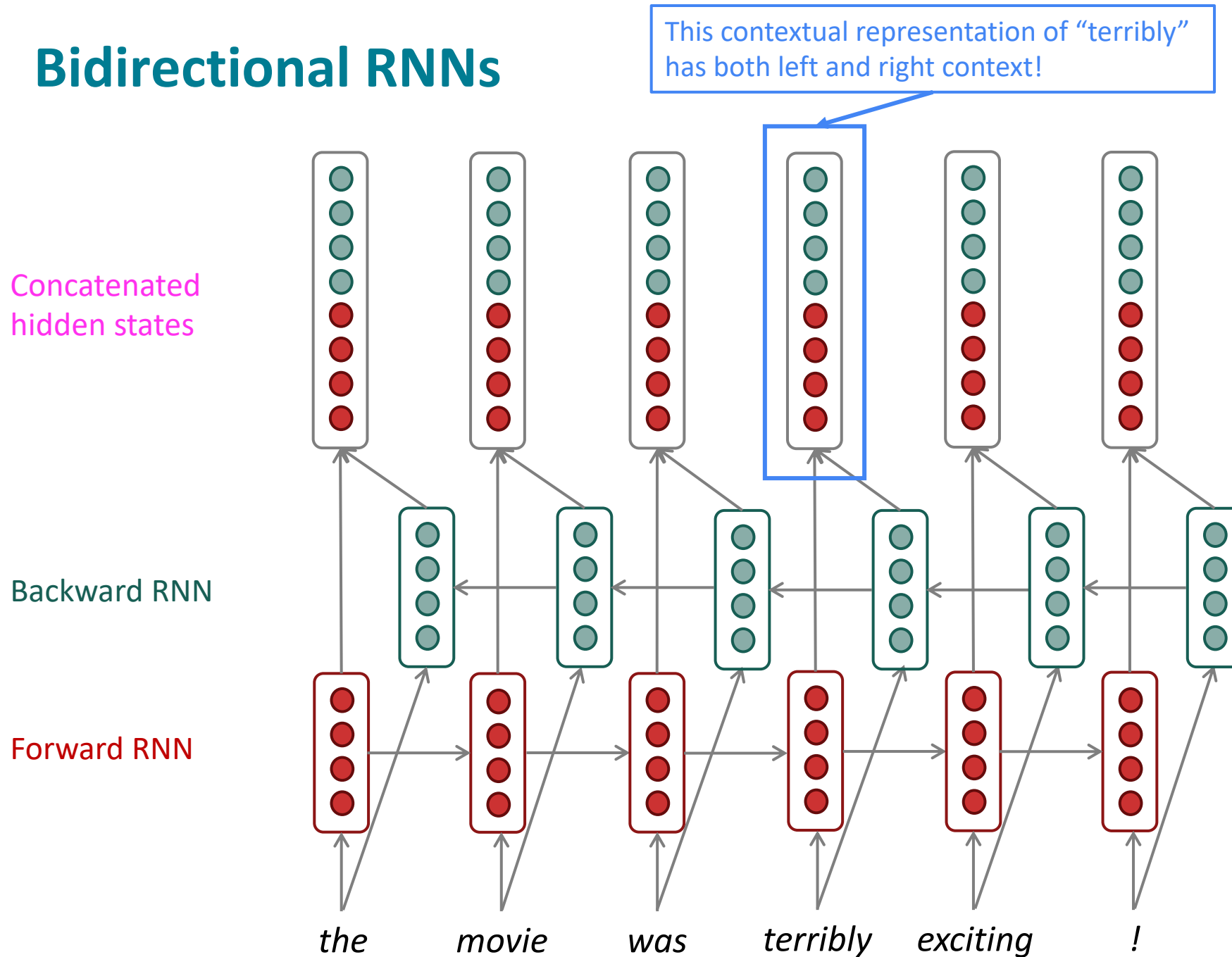
We'll see Machine Translation as an example in much more detail

4. Bidirectional and Multi-layer RNNs: motivation

Task: Sentiment Classification



Bidirectional RNNs



Bidirectional RNNs

On timestep t :

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

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

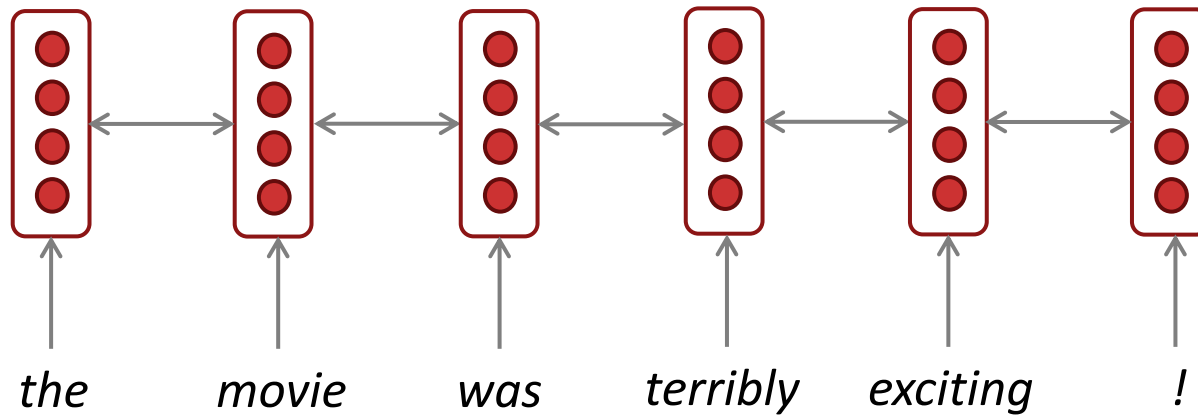
Backward RNN $\overleftarrow{h}^{(t)} = \text{RNN}_{\text{BW}}(\overleftarrow{h}^{(t+1)}, \mathbf{x}^{(t)})$

Generally, these two RNNs have separate weights

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

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



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

Bidirectional RNNs

- Note: bidirectional RNNs are only applicable if you have access to the **entire input sequence**
 - 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).
- For example, **BERT** (**Bidirectional** Encoder Representations from Transformers) is a powerful pretrained contextual representation system **built on bidirectionality**.
 - You will learn more about **transformers**, including BERT, in a couple of weeks!

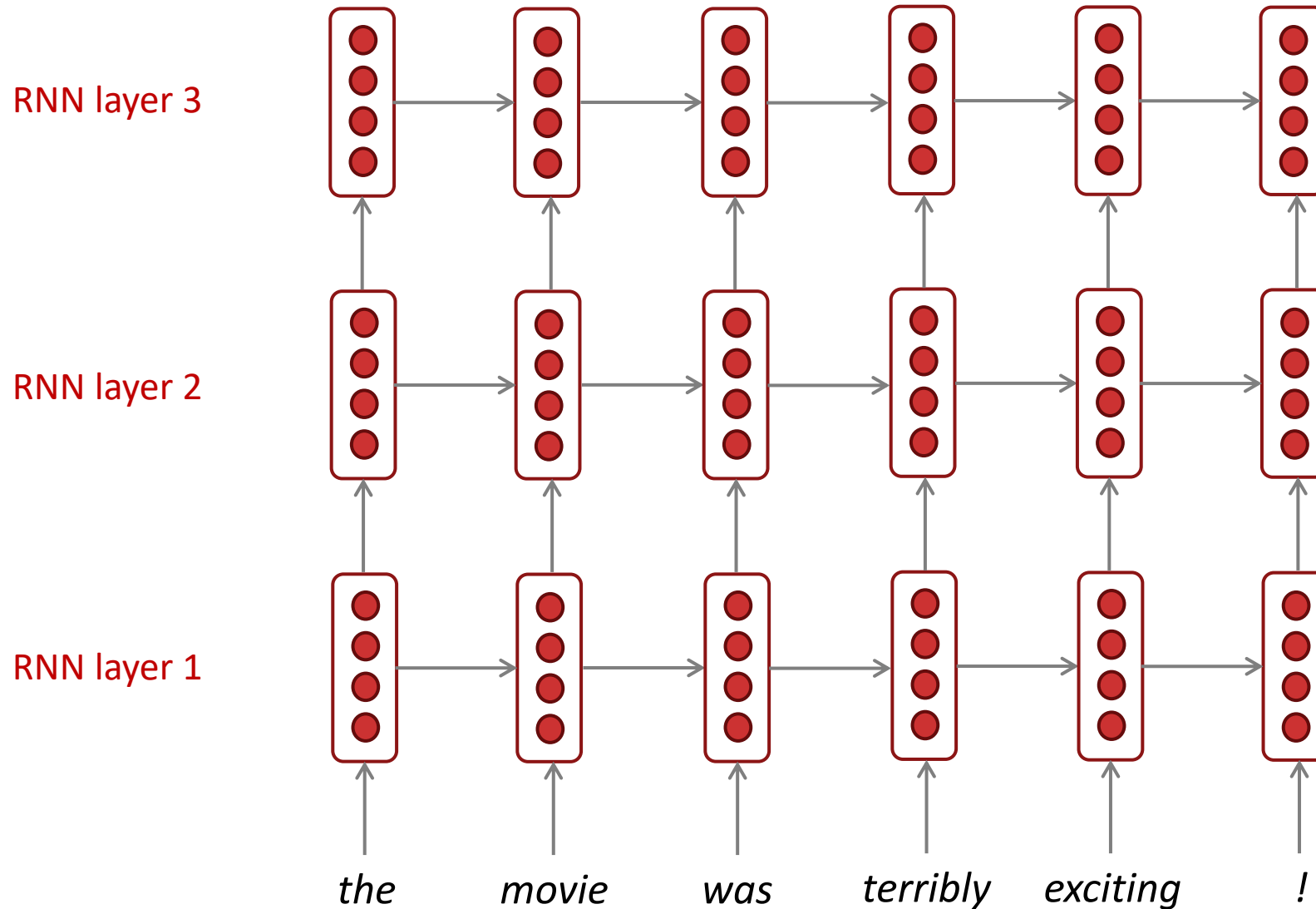
Multi-layer RNNs

- 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**
 - 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 $i+1$



Multi-layer RNNs in practice

- Multi-layer or stacked RNNs allow a network to compute **more complex representations**
 - they work better than just have one layer of high-dimensional encodings!
 - The **lower RNNs** should **compute lower-level features** and the **higher RNNs** should compute **higher-level features**.
- **High-performing RNNs are usually 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
 - Often 2 layers is a lot better than 1, and 3 might be a little better than 2
 - Usually, **skip-connections/dense-connections** are needed to train deeper RNNs (e.g., **8 layers**)
- **Transformer-based networks** (e.g., BERT) are usually deeper, like **12 or 24 layers**.
 - You will learn about Transformers later; they have a lot of skipping-like connections

Machine Translation

Machine Translation (MT) is the task of translating a sentence x from one language (the **source language**) to a sentence y in another language (the **target language**).

$x:$ *L'homme est né libre, et partout il est dans les fers*



$y:$ *Man is born free, but everywhere he is in chains*

– Rousseau

The early history of MT: 1950s

- Machine translation research began in the **early 1950s** on machines less powerful than high school calculators (before term “A.I.” coined!)
- Concurrent with foundational work on automata, formal languages, probabilities, and information theory
- MT heavily funded by military, but basically just simple rule-based systems doing word substitution
- Human language is more complicated than that, and varies more across languages!
- Little understanding of natural language syntax, semantics, pragmatics
- Problem soon appeared intractable

1 minute video showing 1954 MT:

<https://youtu.be/K-HfpsHPmvw>

The early history of MT: 1950s



1990s-2010s: Statistical Machine Translation

- Core idea: Learn a **probabilistic model** from **data**
- Suppose we're translating French \rightarrow English.
- We want to find **best English sentence** y , given **French sentence** x

$$\operatorname{argmax}_y P(y|x)$$

- Use Bayes Rule to break this down into **two components** to be learned separately:

$$= \operatorname{argmax}_y \underbrace{P(x|y)}_{\text{Translation Model}} \underbrace{P(y)}_{\text{Language Model}}$$

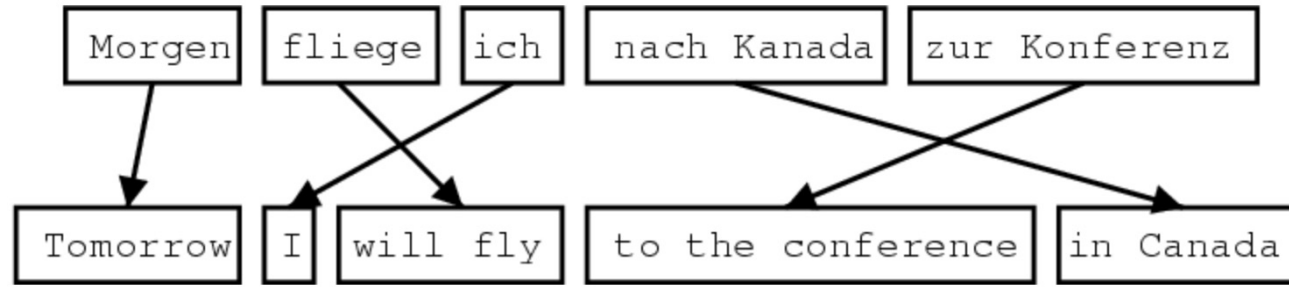
Translation Model

Models how words and phrases
should be translated (*fidelity*).
Learned from parallel data.

Language Model

Models how to write
good English (*fluency*).
Learned from monolingual data.

What happens in translation isn't trivial to model!



1519年600名西班牙人在墨西哥登陆，去征服**几百万人口**
的阿兹特克帝国，初次交锋他们损兵三分之二。

In 1519, six hundred Spaniards landed in Mexico to conquer **the Aztec Empire** **with a population of a few million**. They lost two thirds of their soldiers in the first clash.

translate.google.com (2009): 1519 600 Spaniards landed in Mexico, **millions of people** to **conquer the Aztec empire**, the first two-thirds of soldiers against their loss.

translate.google.com (2013): 1519 600 Spaniards landed in Mexico **to conquer the Aztec empire, hundreds of millions of people**, the initial confrontation loss of soldiers two-thirds.

translate.google.com (2015): 1519 600 Spaniards landed in Mexico, **millions of people** to **conquer the Aztec empire**, the first two-thirds of the loss of soldiers they clash.

1990s–2010s: Statistical Machine Translation

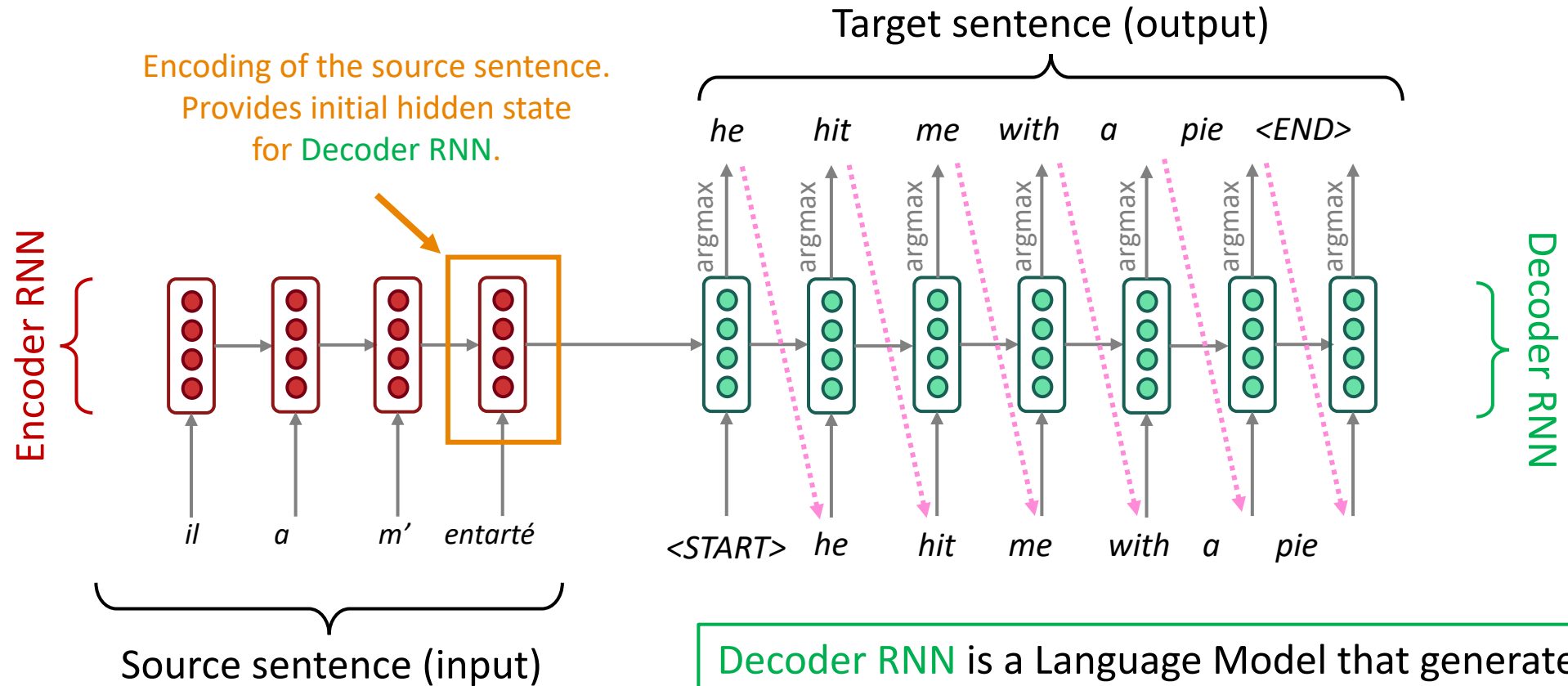
- SMT was a **huge research field**
- The best systems were **extremely complex**
 - Hundreds of important details
- Systems had many **separately-designed subcomponents**
 - Lots of **feature engineering**
 - Need to design features to capture particular language phenomena
 - Required compiling and maintaining **extra resources**
 - Like tables of equivalent phrases
 - Lots of **human effort** to maintain
 - Repeated effort for each language pair!

What is Neural Machine Translation?

- Neural Machine Translation (NMT) is a way to do Machine Translation with a *single end-to-end neural network*
- The neural network architecture is called a *sequence-to-sequence* model (aka *seq2seq*) and it involves *two RNNs*

Neural Machine Translation (NMT)

The sequence-to-sequence model



Encoder RNN produces an **encoding** of the source sentence.

Decoder RNN is a Language Model that generates target sentence, *conditioned on encoding*.

Note: This diagram shows **test time** behavior: decoder output is fed in as next step's input

Sequence-to-sequence is versatile!

- The general notion here is an **encoder-decoder** model
 - One neural network takes input and produces a neural representation
 - Another network produces output based on that neural representation
 - If the input and output are sequences, we call it a seq2seq model
- Sequence-to-sequence is useful for *more than just MT*
- Many NLP tasks can be phrased as sequence-to-sequence:
 - **Summarization** (long text → short text)
 - **Dialogue** (previous utterances → next utterance)
 - **Parsing** (input text → output parse as sequence)
 - **Code generation** (natural language → Python code)

Neural Machine Translation (NMT)

- The **sequence-to-sequence** model is an example of a **Conditional Language Model**
 - **Language Model** because the decoder is predicting the next word of the target sentence y
 - **Conditional** because its predictions are *also* conditioned on the source sentence x

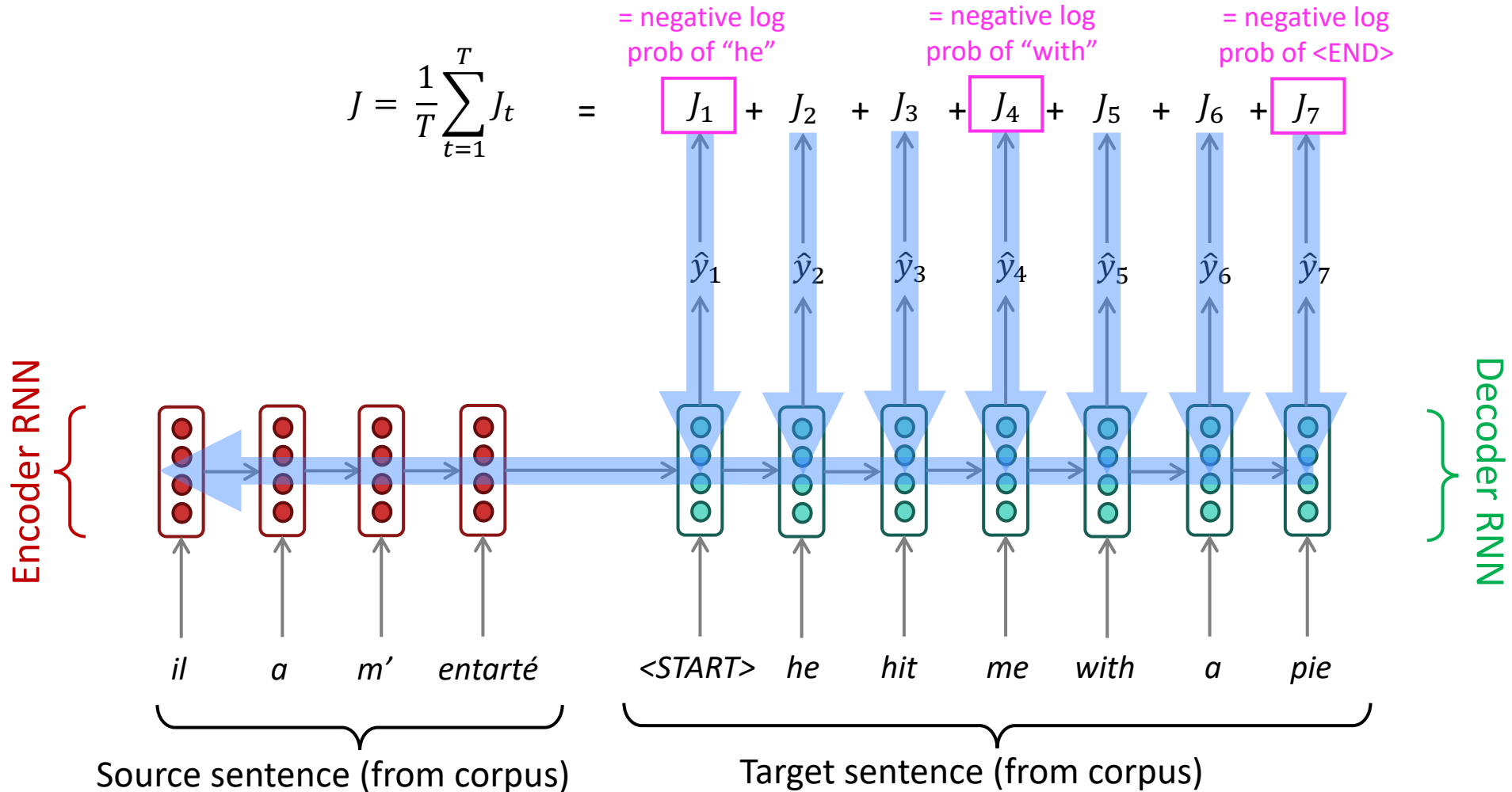
- NMT directly calculates $P(y|x)$:

$$P(y|x) = P(y_1|x) P(y_2|y_1, x) P(y_3|y_1, y_2, x) \dots P(y_T|y_1, \dots, y_{T-1}, x)$$

Probability of next target word, given
target words so far and source sentence x

- **Question:** How to train an NMT system?
- **(Easy) Answer:** Get a big parallel corpus...
 - But there is now exciting work on “unsupervised NMT”, data augmentation, etc.

Training a Neural Machine Translation system

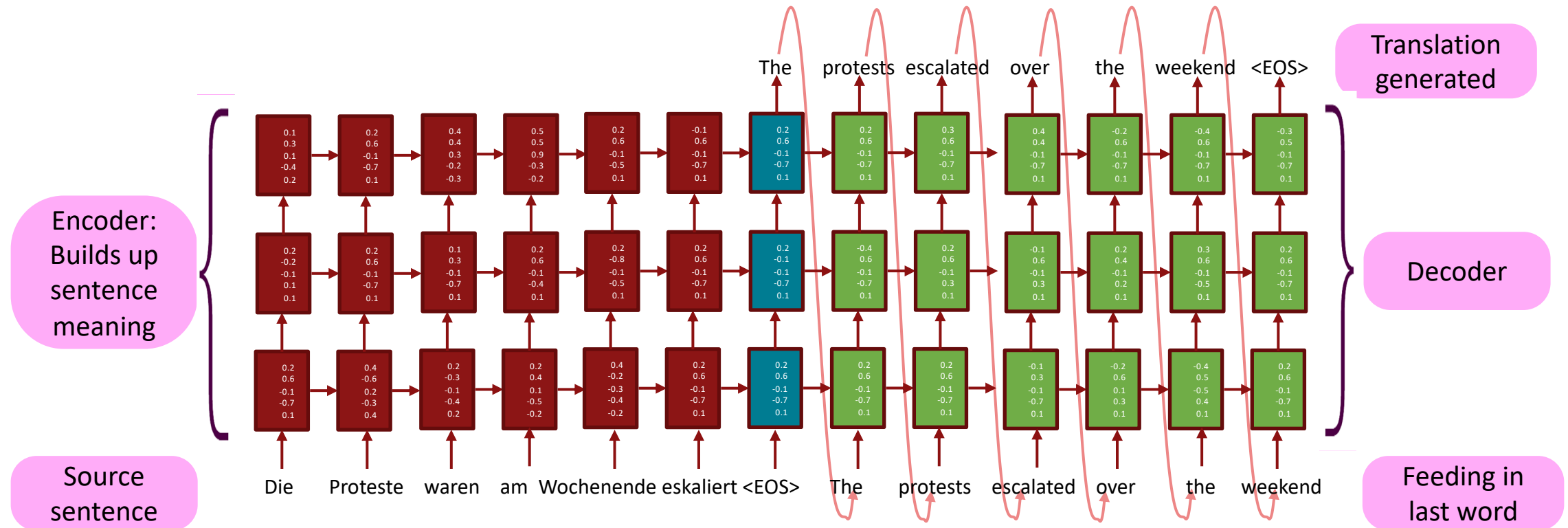


Seq2seq is optimized as a **single system**. Backpropagation operates "*end-to-end*".

Multi-layer deep encoder-decoder machine translation net

[Sutskever et al. 2014; Luong et al. 2015]

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



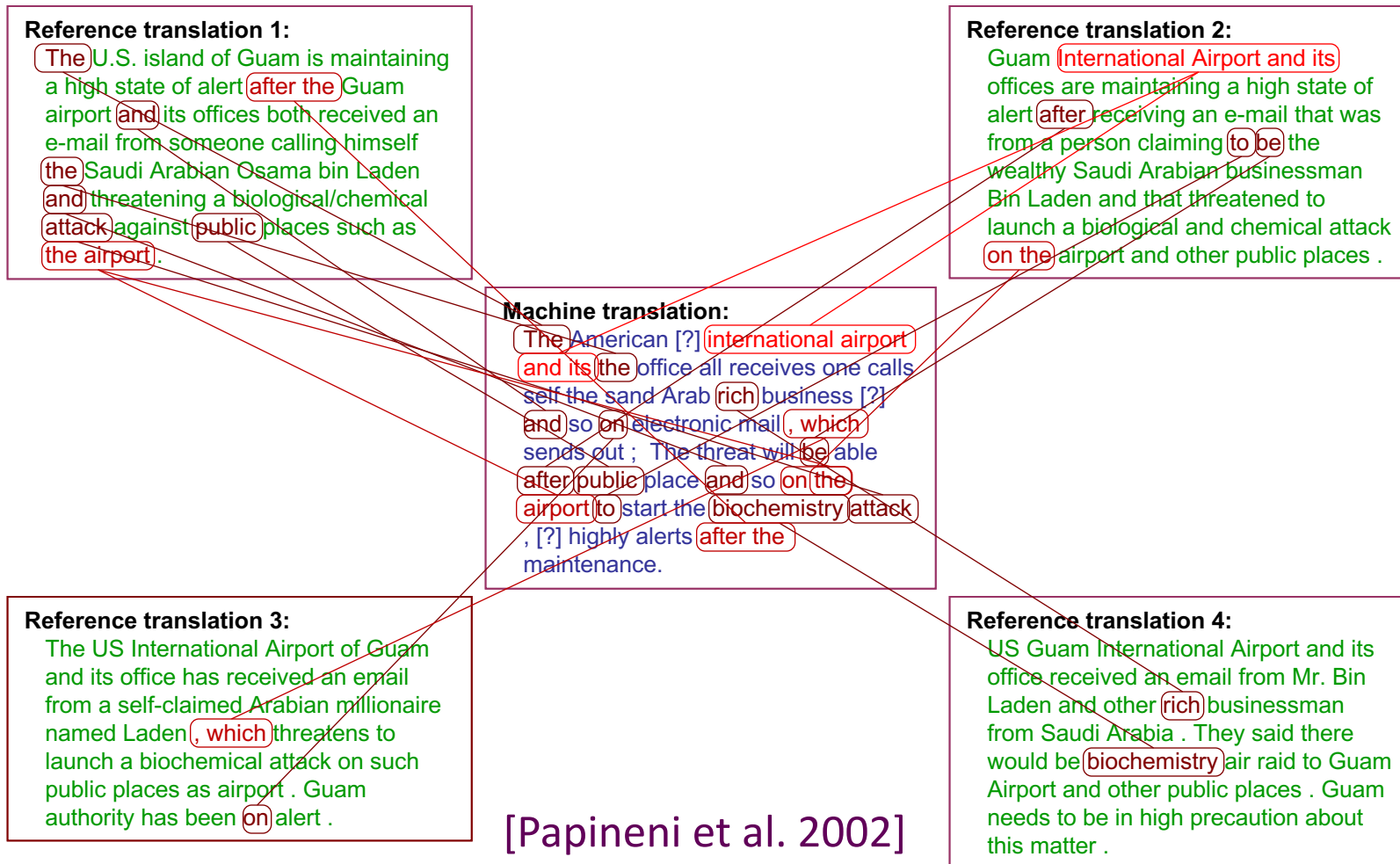
How do we evaluate Machine Translation?

BLEU (Bilingual Evaluation Understudy)

You'll see BLEU in detail
in Assignment 4!

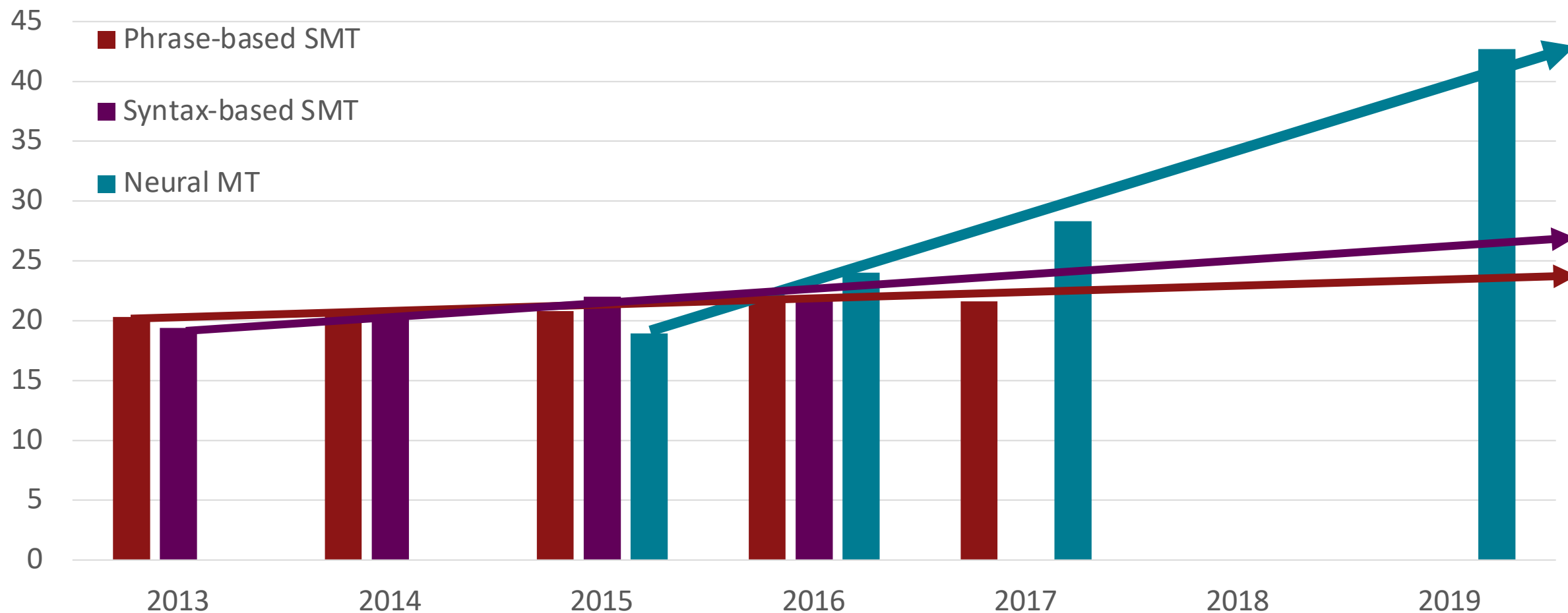
- BLEU compares the machine-written translation to one or several human-written translation(s), and computes a **similarity score** based on:
 - ***n*-gram precision** (usually for 1, 2, 3 and 4-grams)
 - Plus a penalty for too-short system translations
- BLEU is **useful** but **imperfect**
 - There are many valid ways to translate a sentence
 - So a **good** translation can get a **poor** BLEU score because it has low *n*-gram overlap with the human translation 😞

BLEU score against 4 reference translations



MT progress over time

[Edinburgh En-De WMT newstest2013 Cased BLEU; NMT 2015 from U. Montréal; NMT 2019 FAIR on newstest2019]



Sources: http://www.meta-net.eu/events/meta-forum-2016/slides/09_sennrich.pdf & <http://matrix.statmt.org/>

Advantages of NMT

Compared to SMT, NMT has many advantages:

- Better performance
 - More fluent
 - Better use of context
 - Better use of phrase similarities
- A single neural network to be optimized end-to-end
 - No subcomponents to be individually optimized
- Requires much less human engineering effort
 - No feature engineering
 - Same method for all language pairs

Disadvantages of NMT?

Compared to SMT:

- NMT is **less interpretable**
 - Hard to debug
- NMT is **difficult to control**
 - For example, can't easily specify rules or guidelines for translation
 - Safety concerns!

NMT: the first big success story of NLP Deep Learning

Neural Machine Translation went from a fringe research attempt in **2014** to the leading standard method in **2016**

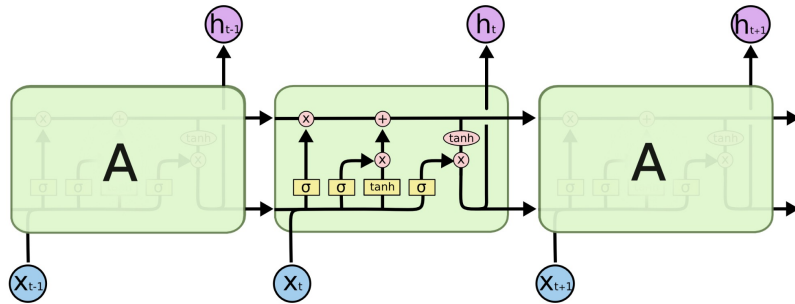
- **2014**: First seq2seq paper published
- **2016**: Google Translate switches from SMT to NMT – and by 2018 everyone has



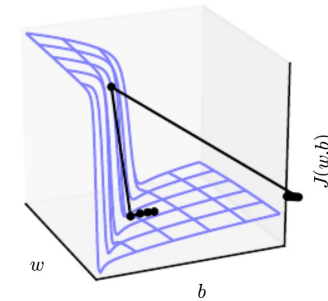
- This is amazing!
 - **SMT** systems, built by hundreds of engineers over many years, outperformed by NMT systems trained by a small group of engineers in a few months

In summary

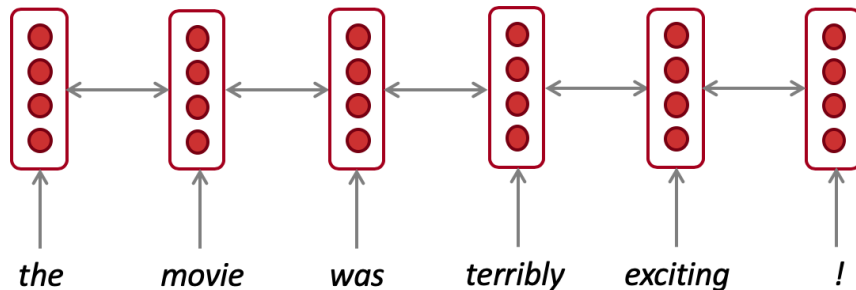
Lots of new information today! What are some of the **practical takeaways**?



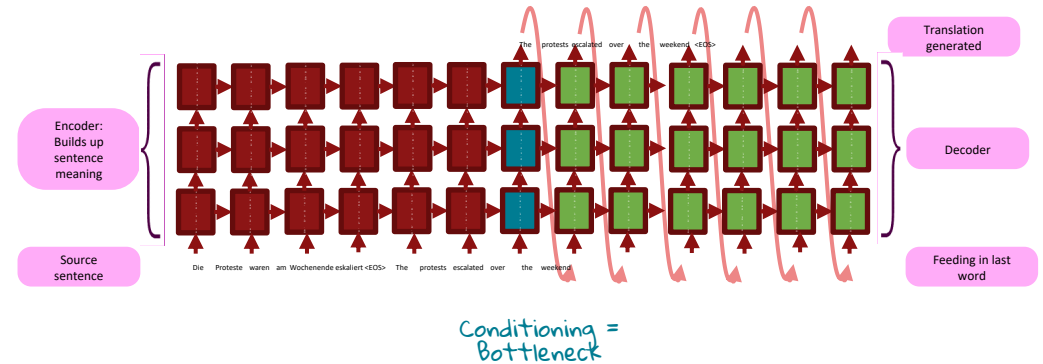
1. LSTMs are powerful



2. Clip your gradients



3. Use bidirectionality when possible



4. Encoder-Decoder Neural Machine Translation Systems work very well