

# Word Vectors, Language Models and Recurrent Neural Networks

# Recap...Language Modelling

- The task of predicting what word comes next.

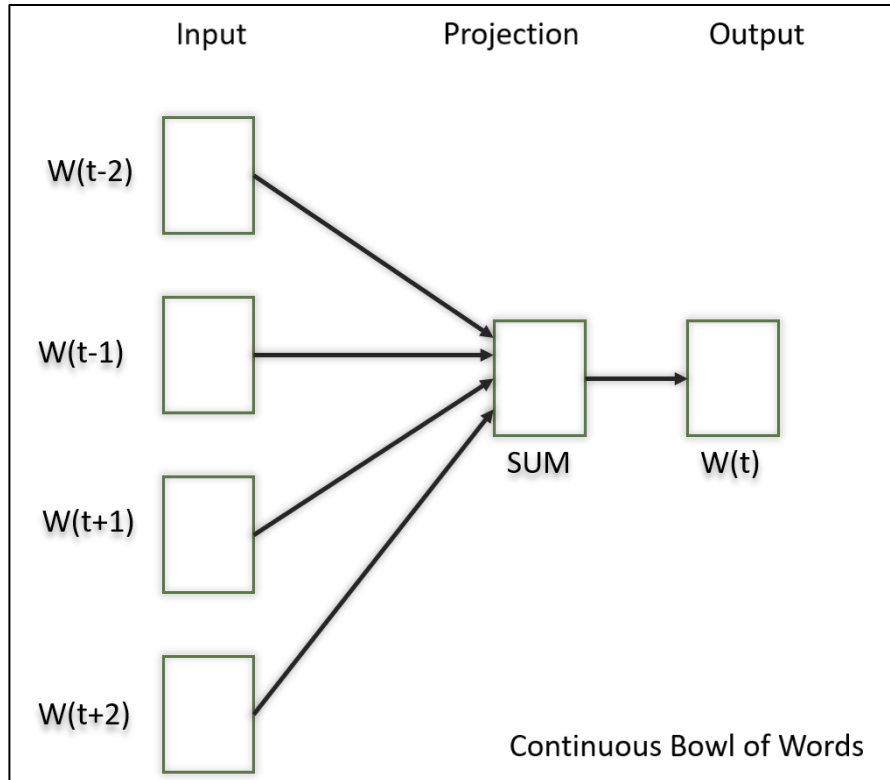
The students opened their Books/ Laptops/ Exams/...

- More formally: Given a sequence of words:  $x_1, x_2, x_3, \dots x_t$   
Compute Probability distribution of the next word  $x_{t+1}$

$$P(x_{t+1} | x_1, x_2, x_3, \dots x_t)$$

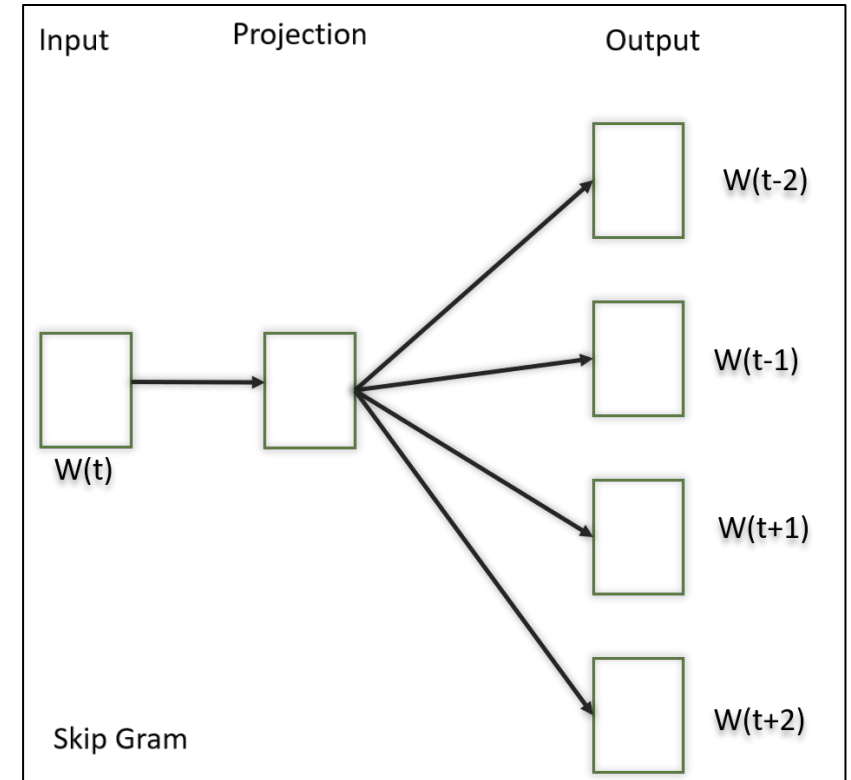
where  $x_{t+1}$  can be any word in the vocabulary  $V = \{w_1, w_2, \dots, w_{|V|}\}$

# Recap... Word Embeddings



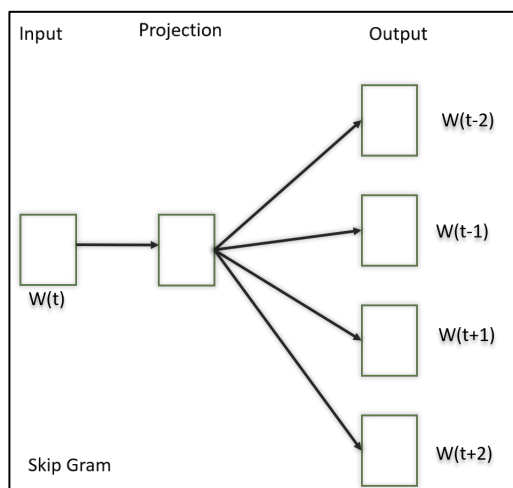
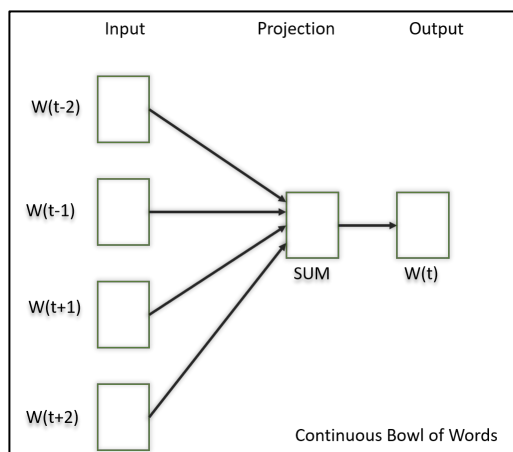
*linguistics* =

$\begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$



# Word Embeddings

## Word2Vec

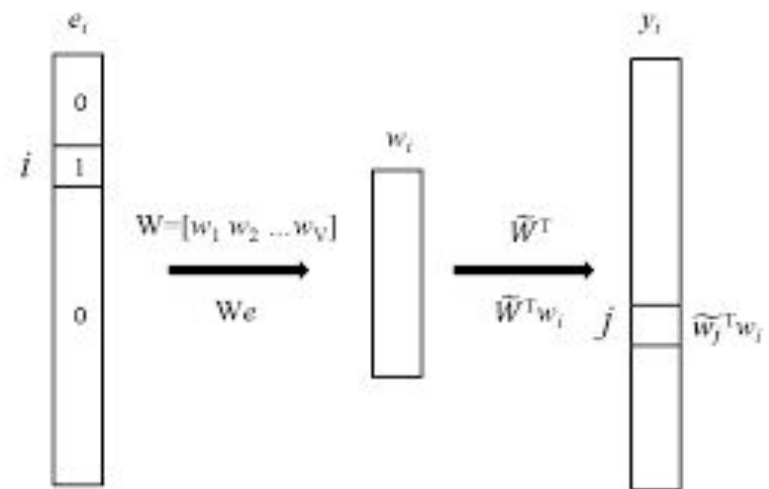


## GLOVE

The fast cat wears no hat

The cat in th4 hat ran fast

	cat	fast	hat	in	no	ran	the	wears
cat	0							
fast	2	0						
hat	2	2	0					
in	1	1	1	0				
no	1	1	1	0	0			
ran	1	1	1	1	0	0		
the	3	3	3	2	1	2	1	
wears	1	1	1	1	1	0	1	0



# Semantics

- How does a computer understand meaning?

What is the meaning of “bardiwac”?

He handed her a glass of bardiwac.

Beef dishes are made to complement the bardiwacs.

Malbec, one of the lesser-known bardiwac grapes, responds well to Australia’s sunshine.

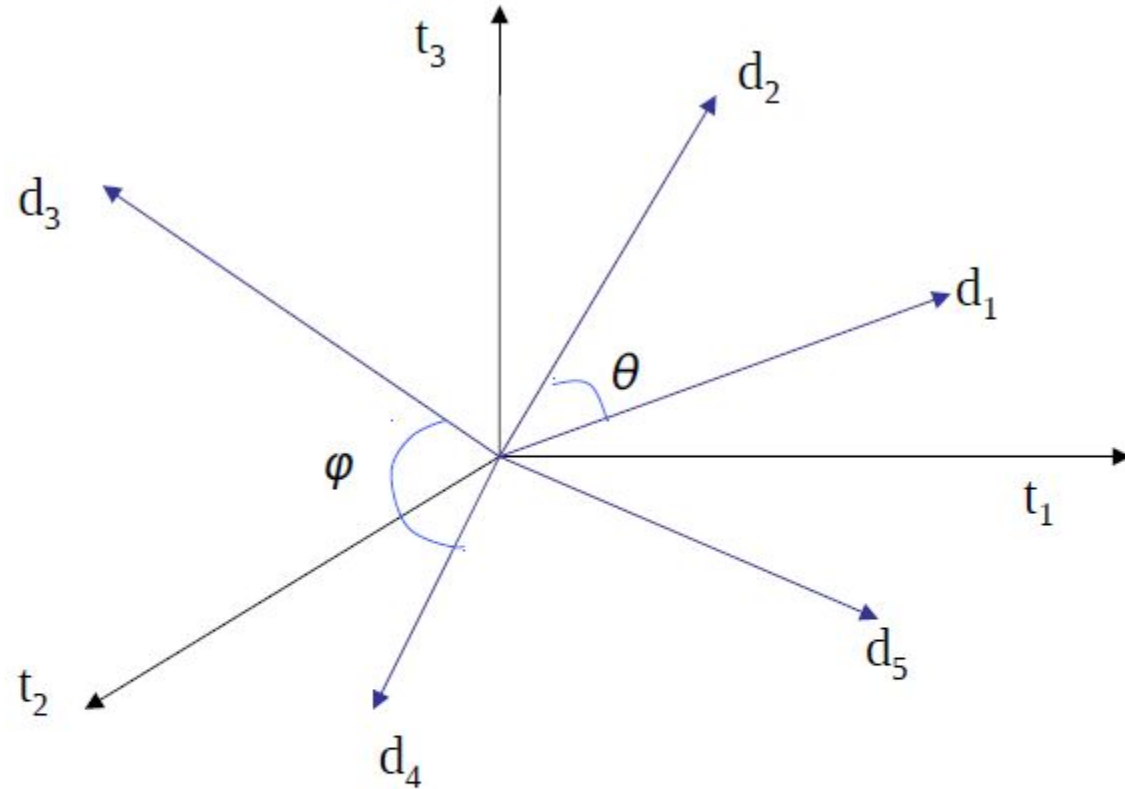
# Meaning and Neighbourhood

You shall know a word by the company it keeps!

***Meaning*** of a word can be derived from  
the meaning of its ***contexts***

*AKA Distributional Semantics*

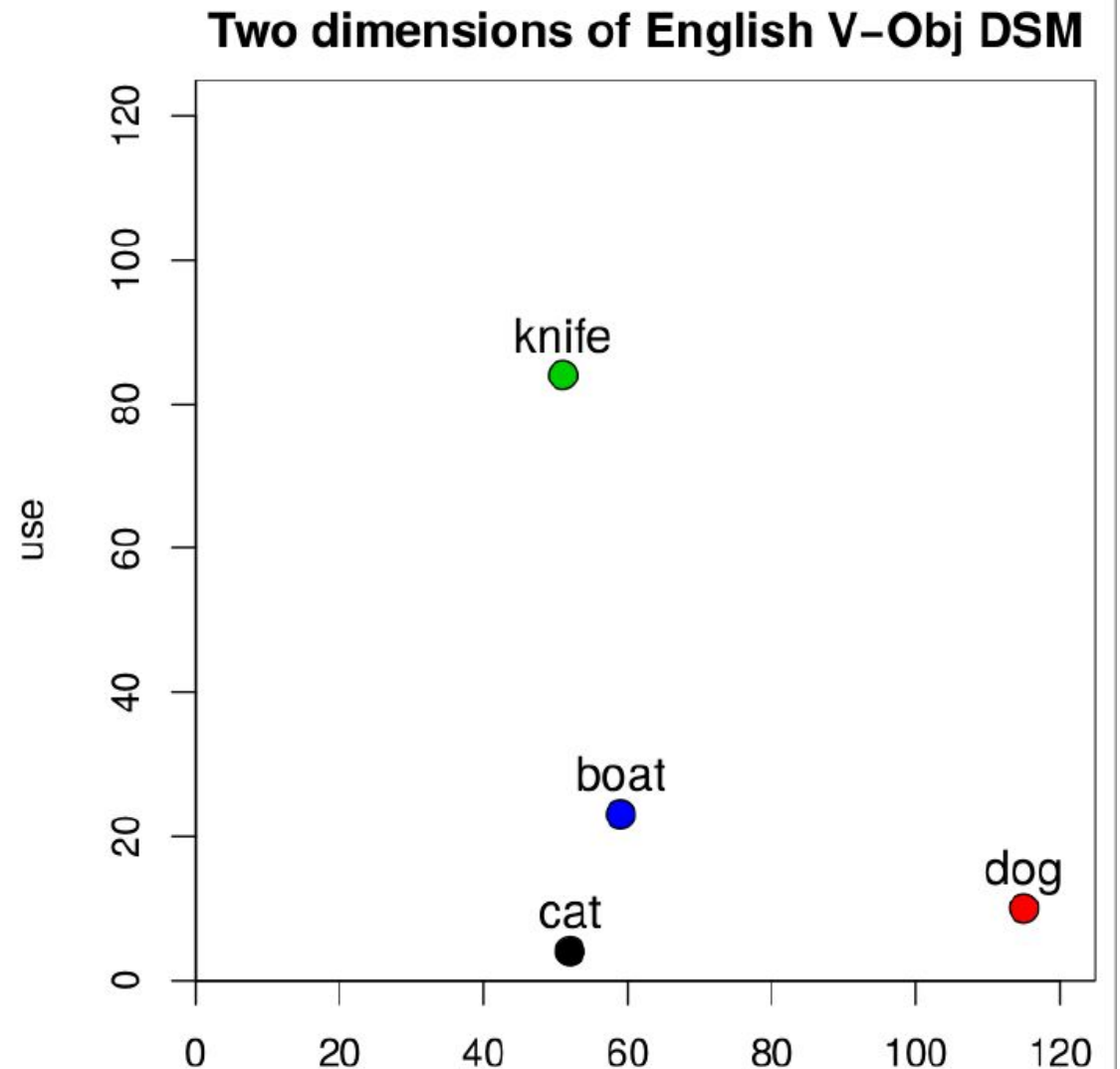
# Geometric Interpretation



*Postulate:* Words that are “close together”  
in the vector space talk about the same things.

# Geometric Interpretation

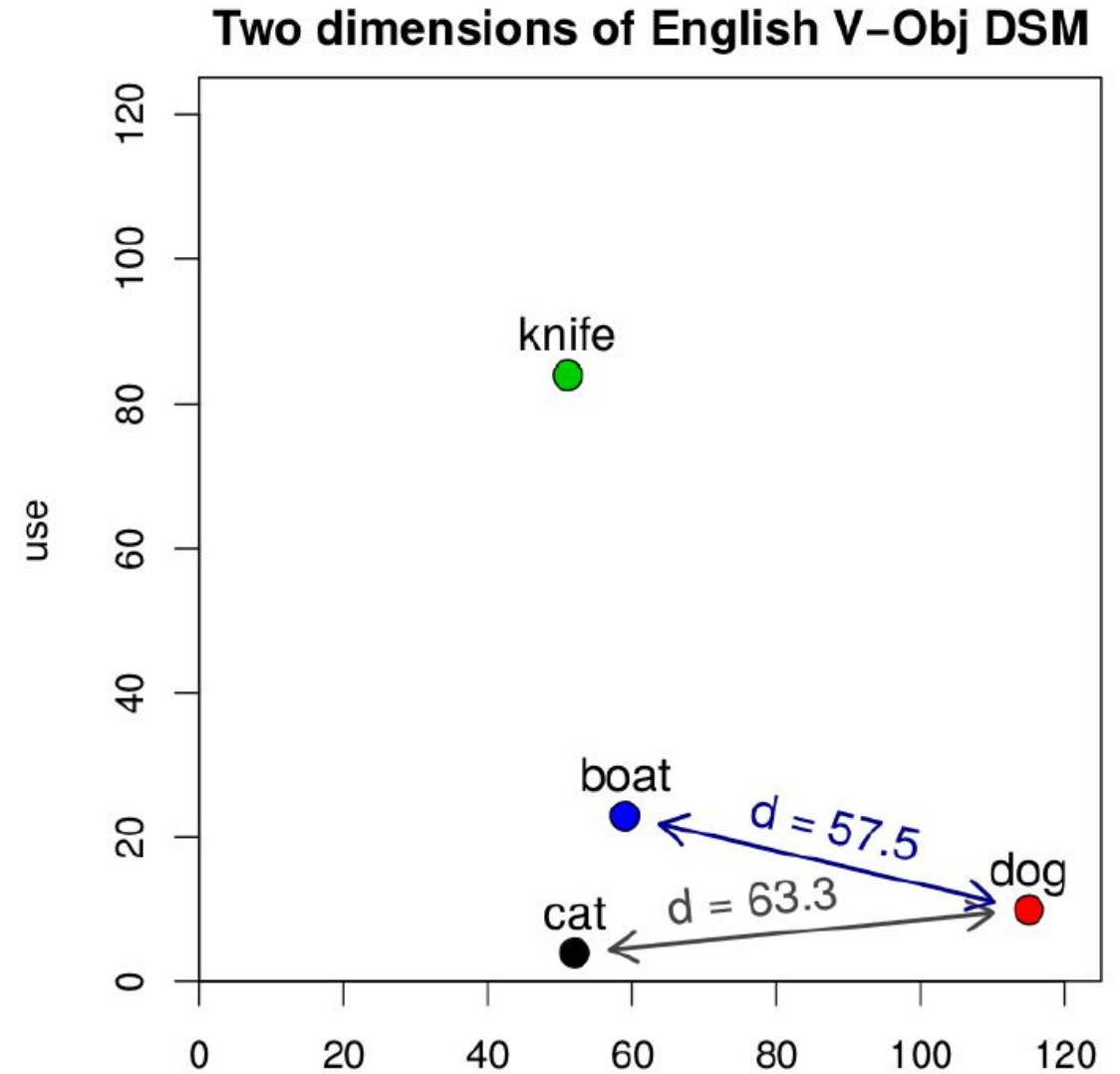
- ▶ row vector  $\mathbf{x}_{\text{dog}}$  describes usage of word *dog* in the corpus
- ▶ can be seen as coordinates of point in  $n$ -dimensional Euclidean space
- ▶ illustrated for two dimensions: *get* and *use*
- ▶  $\mathbf{x}_{\text{dog}} = (115, 10)$





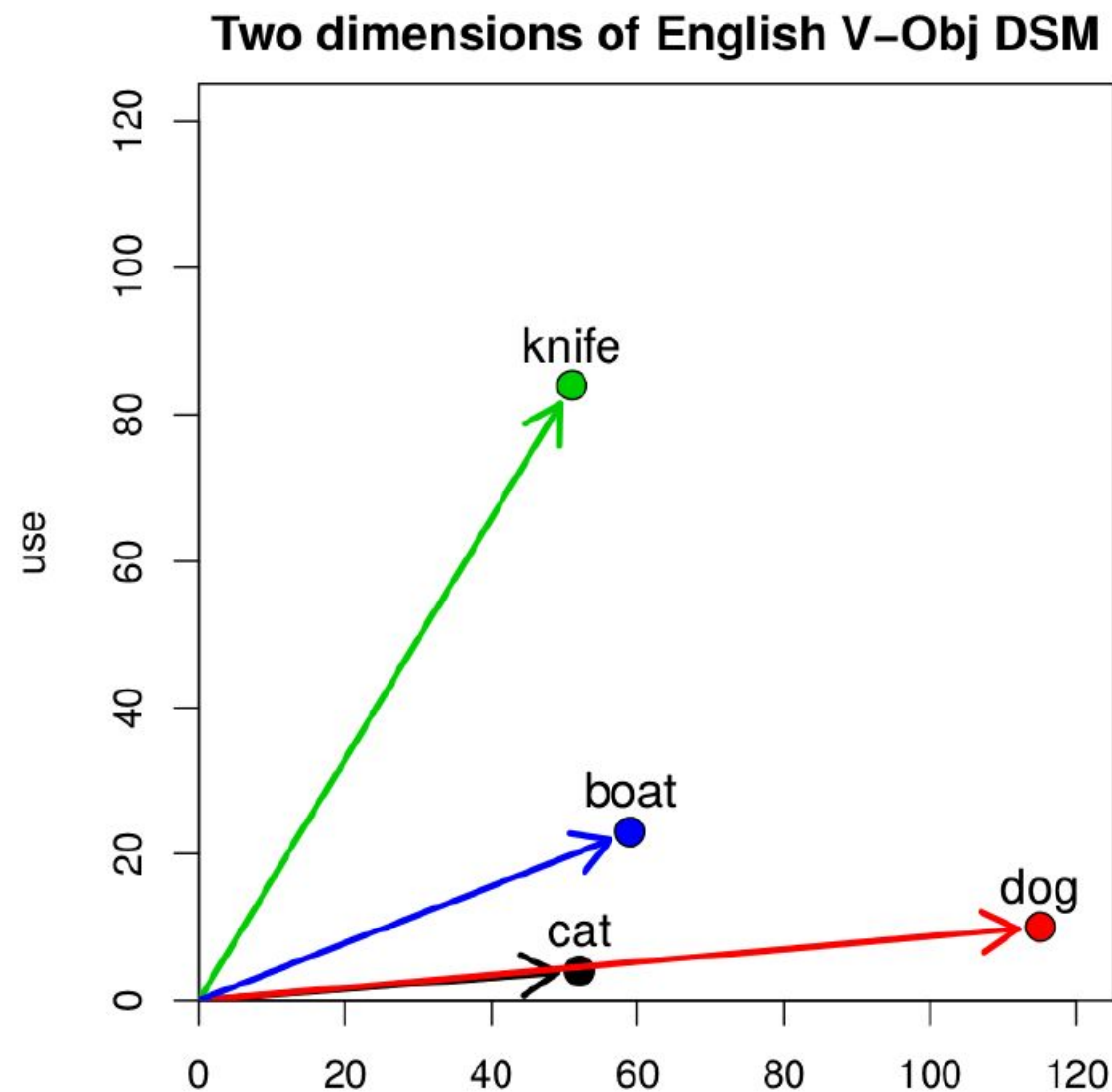
# Geometric Interpretation

- ▶ similarity = spatial proximity (Euclidean dist.)
- ▶ location depends on frequency of noun ( $f_{\text{dog}} \approx 2.7 \cdot f_{\text{cat}}$ )



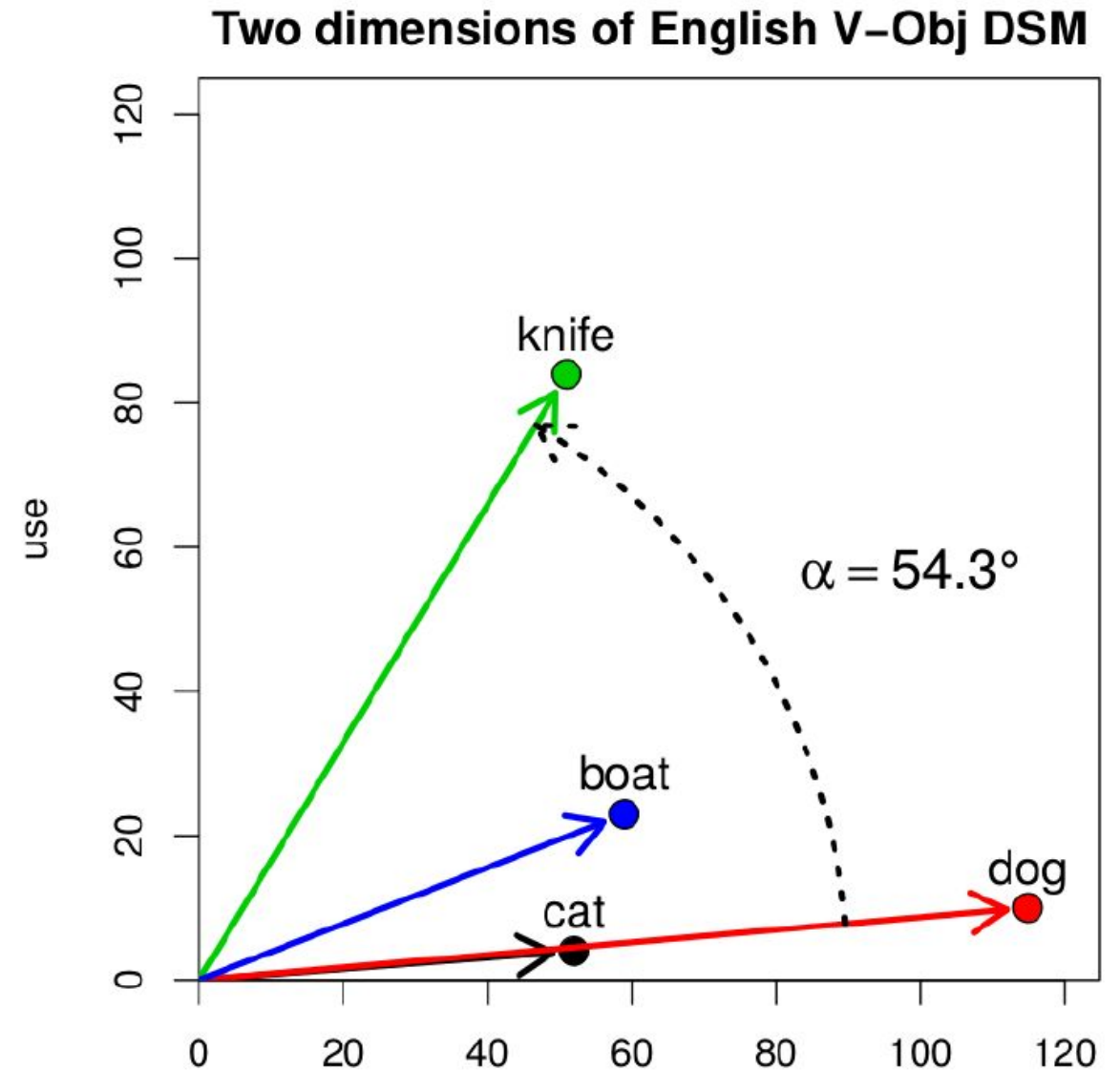
# Geometric Interpretation

- ▶ vector can also be understood as arrow from origin
- ▶ direction more important than location



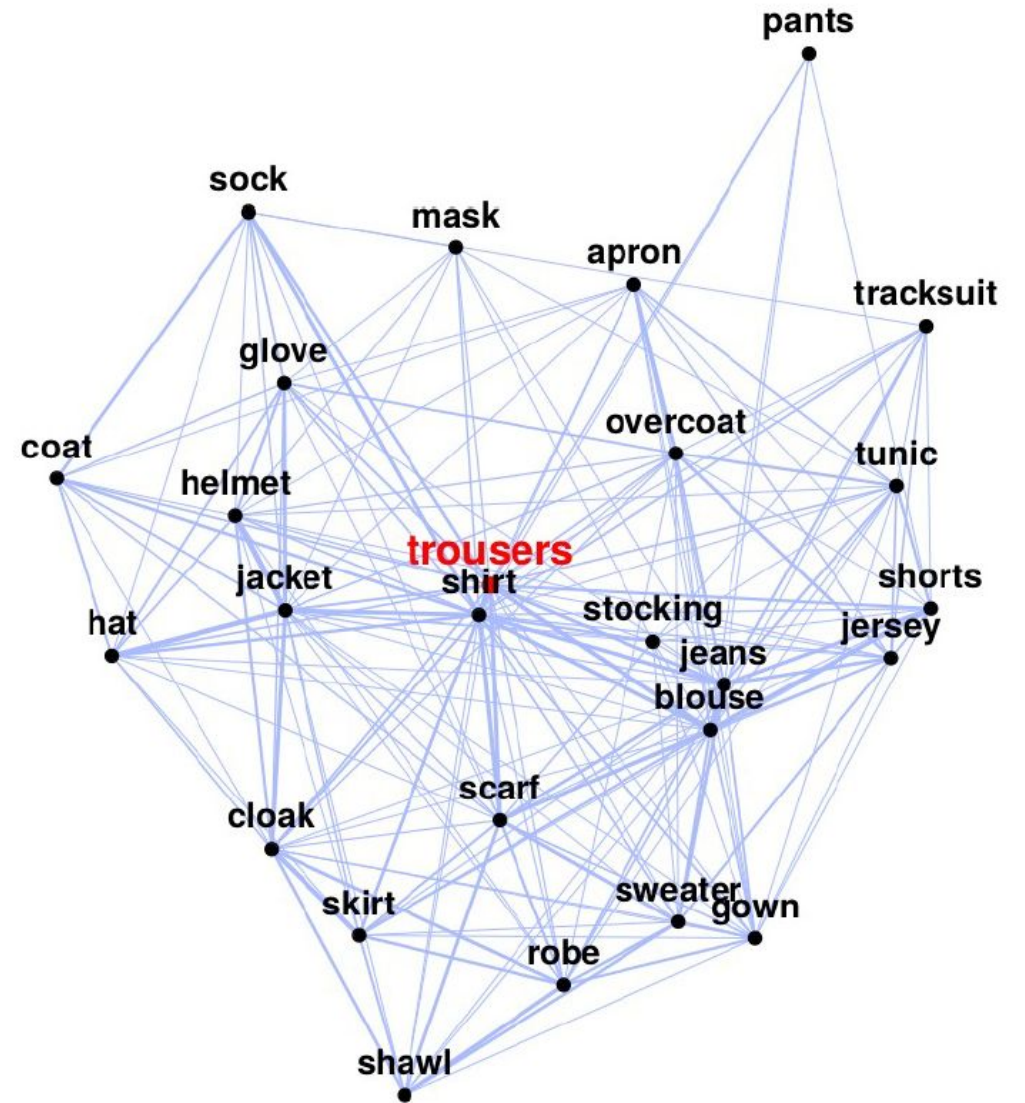
# Geometric Interpretation

- ▶ vector can also be understood as arrow from origin
- ▶ direction more important than location
- ▶ use angle  $\alpha$  as distance measure



## Nearest neighbours with similarity graph

$$\text{Trousers} = \begin{pmatrix} 0.286 \\ 0.792 \\ -0.177 \\ -0.107 \\ 0.109 \\ -0.542 \\ 0.349 \\ 0.271 \end{pmatrix}$$

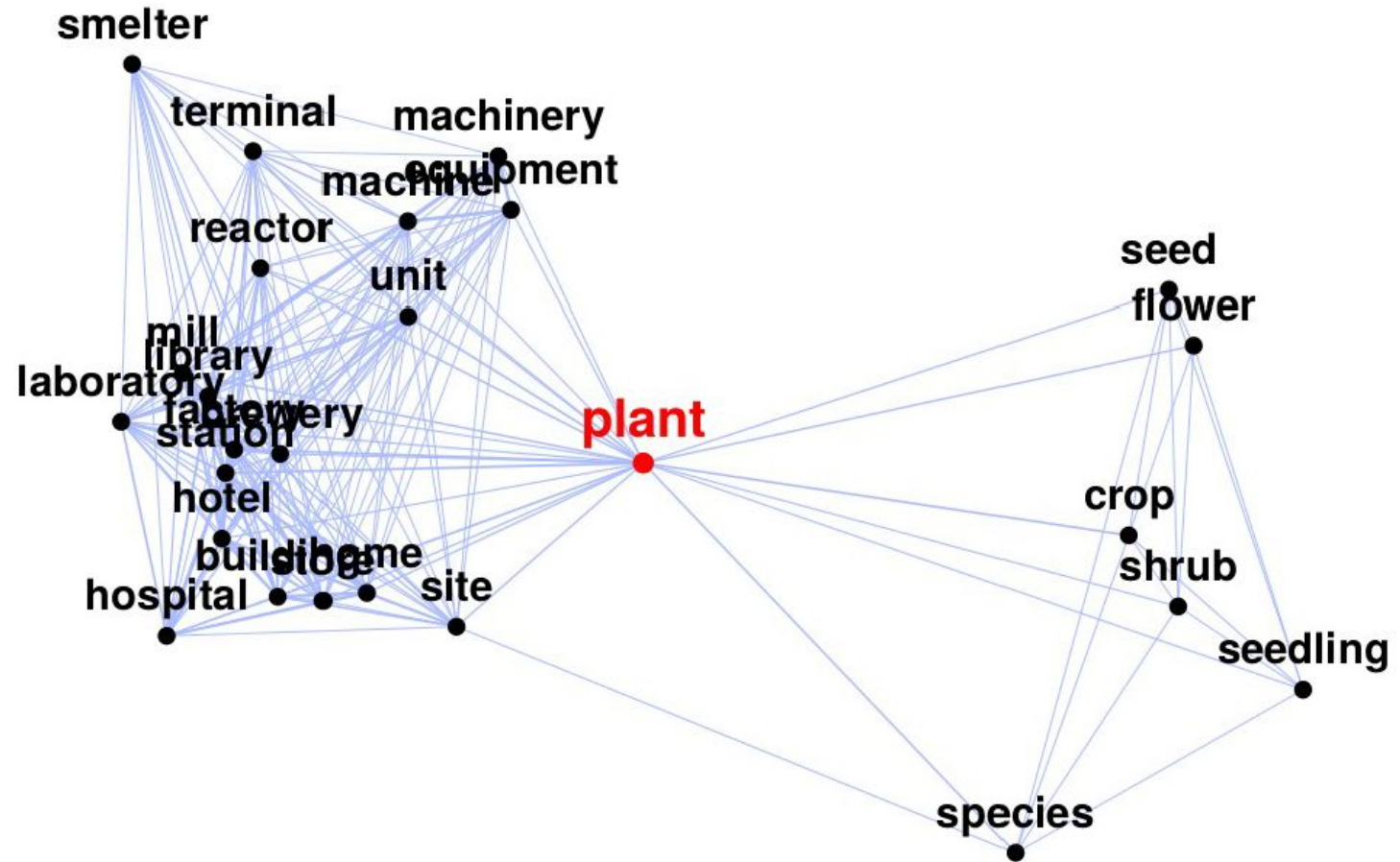


# Word Sense Disambiguation

What is the meaning of “**Bank**”?

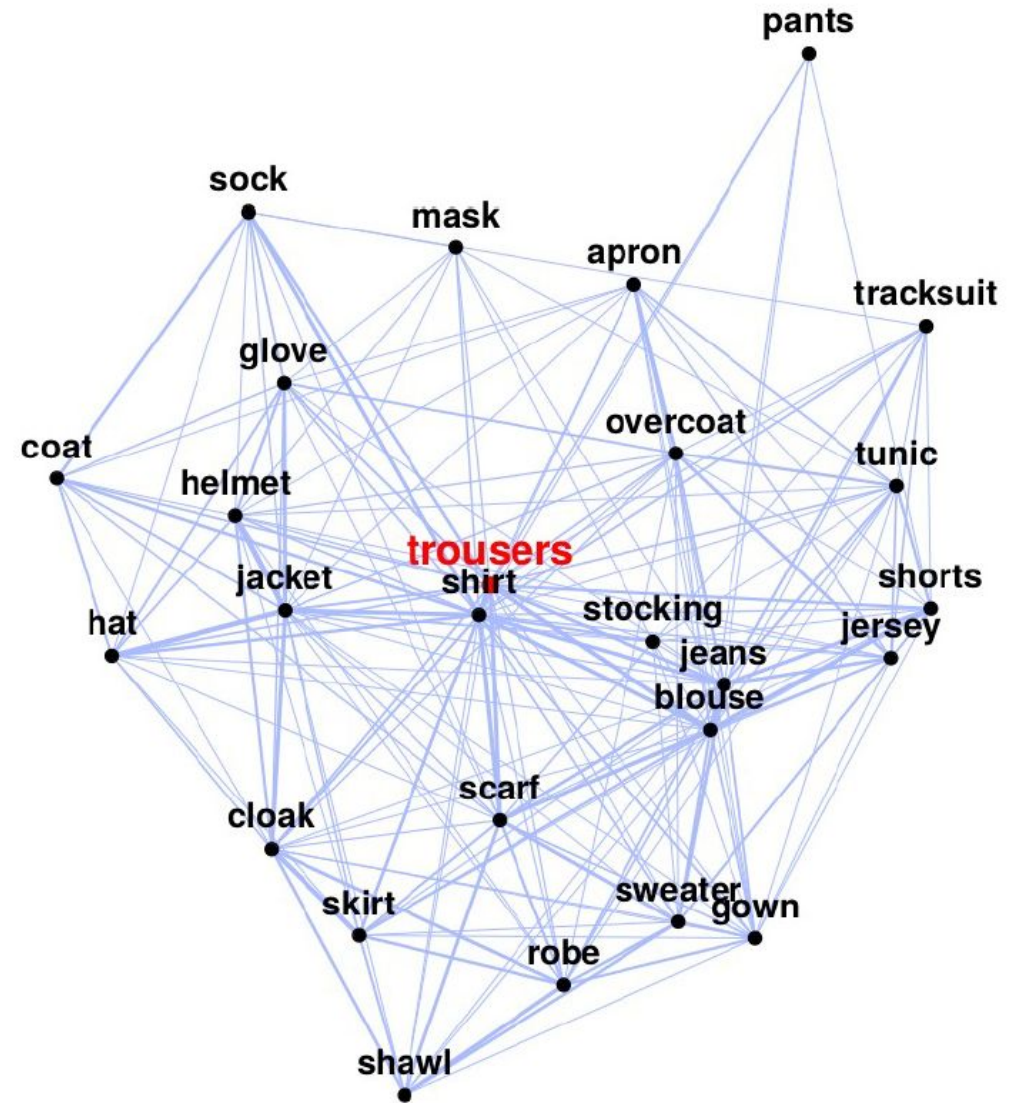
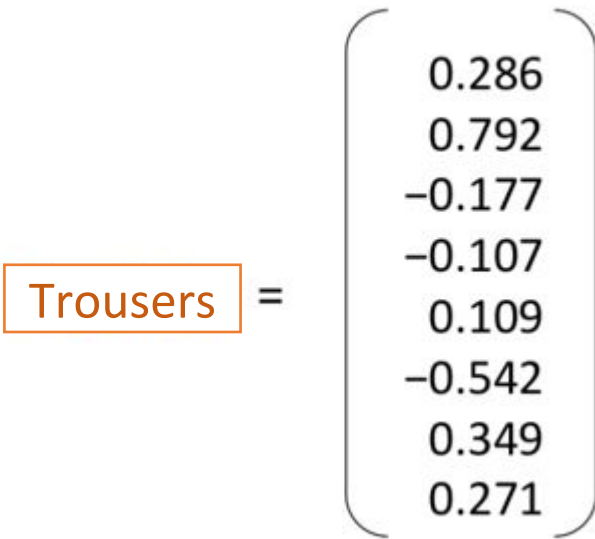
- He sat on the bank of the river and watched the currents --> **sloping land**
- A huge bank of earth --> **a long ridge or pile**
- The State Bank has allowed me the loan --> **Financial Institution**

# Nearest neighbours with similarity graph





## Nearest neighbours with similarity graph



# What are words?



# Morphology

- A field of linguistics that studies the structure of words.
  - It identifies how a word is produced through the use of morphemes.
  - A morpheme is a basic unit of the English language.
  - The morpheme is the smallest element of a word that has grammatical function and meaning.
- **Free morpheme** and **bound morpheme** are the two types of morphemes.
  - A single free morpheme can become a complete word.
  - A bound morpheme, on the other hand, cannot stand alone and must be joined to a free morpheme to produce a word.

# Types of Morphology

- Inflectional and Derivational Morphology
  - boy-boys, play-playing, come-came, go-went
  - govern, government, governance, governor, governable, ungovernable
- Compound Formations: Football, Blackboard, Underworld
- Multiword Expressions: Grand father, Wall clock
- Complex Predicates/phrasal verbs: lift up, bring up
- etc...

# Recap...Language Modelling

- The task of predicting what word comes next.

The students opened their \_\_\_\_\_ Books/ Laptops/ Exams/...

- More formally: Given a sequence of words:  $x_1, x_2, x_3, \dots x_t$

Compute Probability distribution of the next word  $x_{t+1}$

$$P(x_{t+1} | x_1, x_2, x_3, \dots x_t)$$

where  $x_{t+1}$  can be any word in the vocabulary  $V = \{w_1, w_2, \dots, w_{|V|}\}$

## How to learn a Language Model?

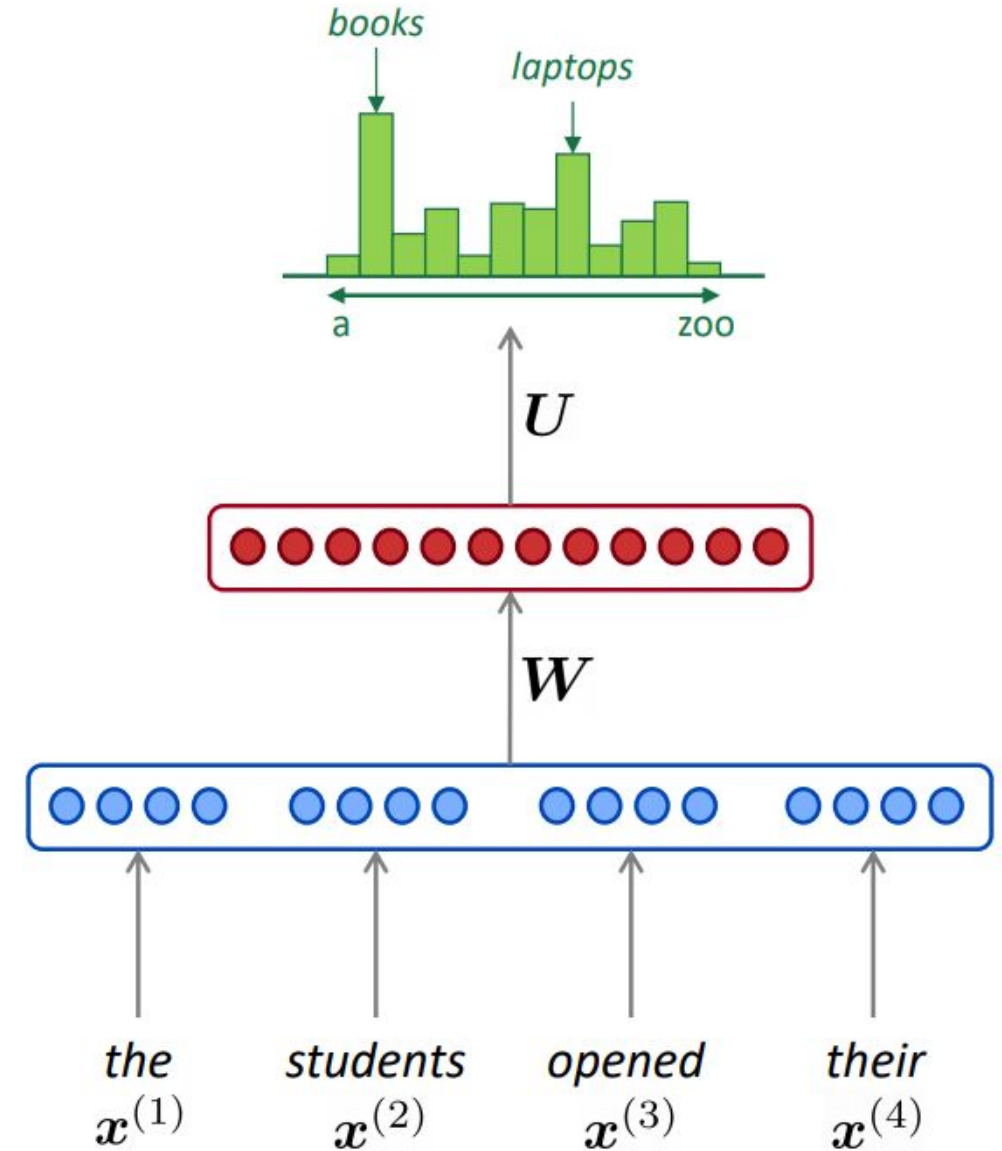
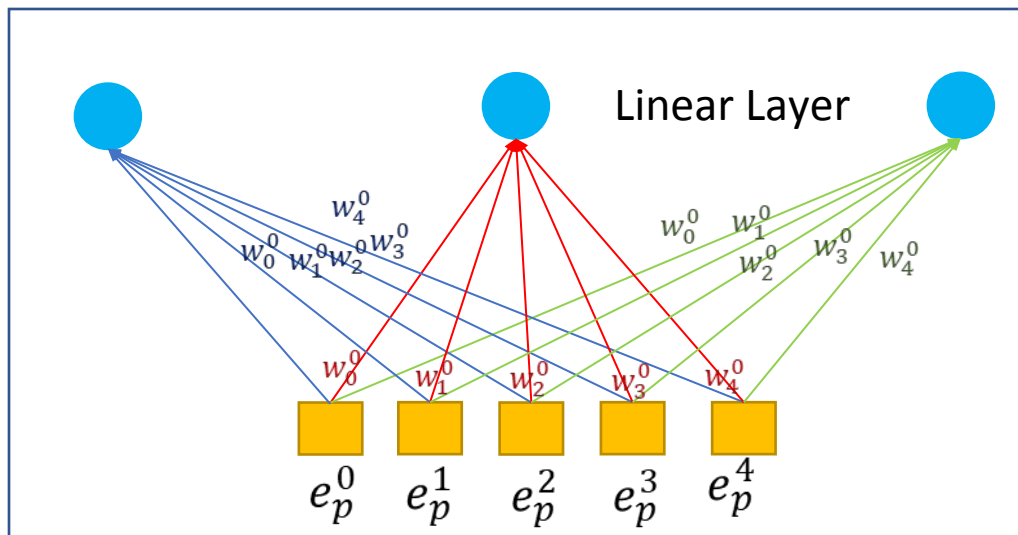
Answer (pre- Deep Learning): learn an n-gram Language Model!

Definition: A n-gram is a chunk of n consecutive words.

- unigrams: “the”, “students”, “opened”, “their”
- bigrams: “the students”, “students opened”, “opened their”
- trigrams: “the students opened”, “students opened their”

# Neural Language Model

- Recall the Language Modeling task:
  - Input: sequence of words  $x^{(1)}, \dots, x^{(T)}$
  - Output: prob dist of the next word  $P(x^{(t+1)} | x^{(t)}, \dots, x^{(1)})$
- Window-based neural model



# A fixed-window neural Language Model

Approximately: Y. Bengio, et al. (2000/2003): A Neural Probabilistic Language Model

**Improvements** over  $n$ -gram LM:

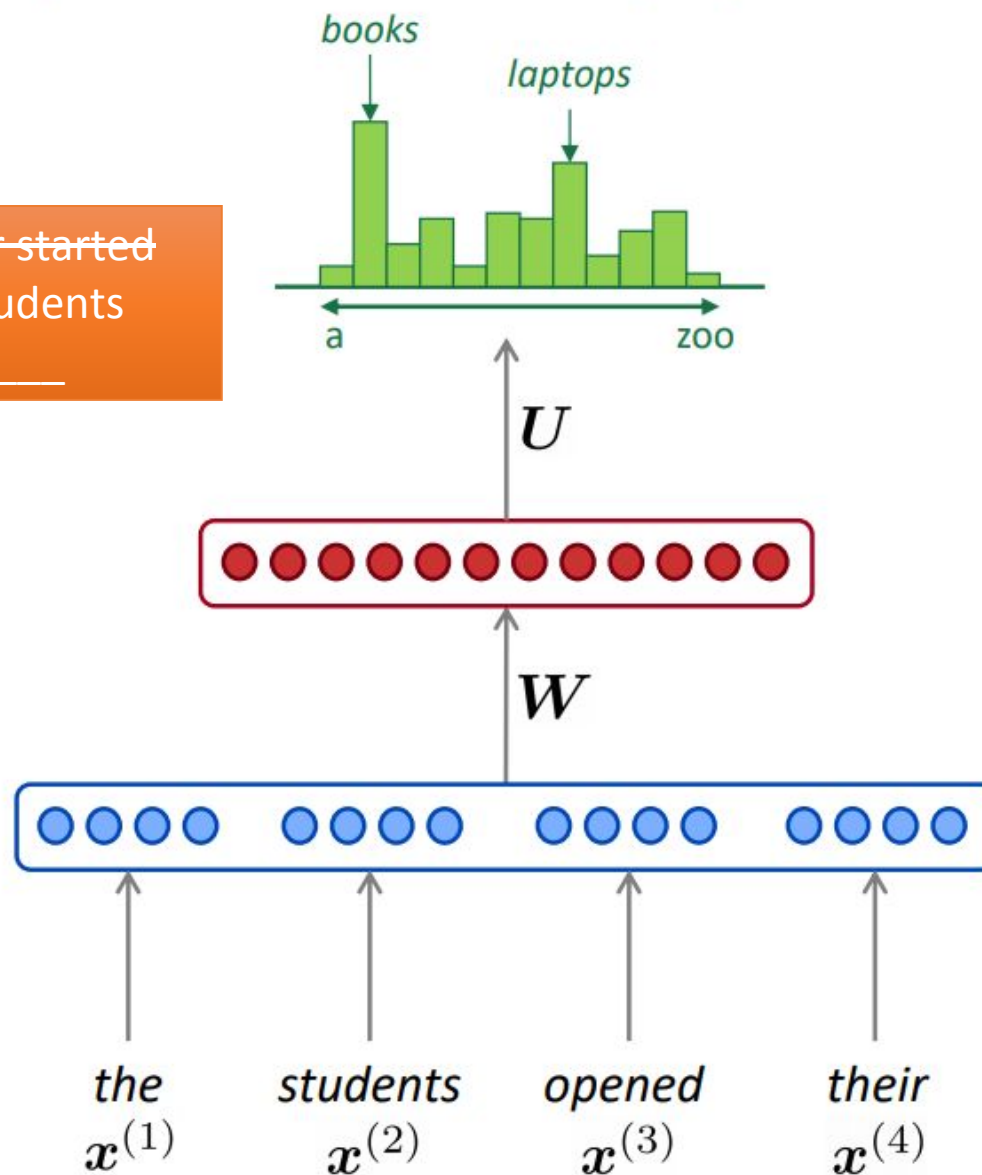
- No sparsity problem
- Don't need to store all observed  $n$ -grams

Remaining **problems**:

- Fixed window is **too small**
  - Enlarging window enlarges  $W$
  - Window can never be large enough!
  - $x^{(1)}$  and  $x^{(2)}$  are multiplied by completely different weights in  $W$ .
- No symmetry** in how the inputs are processed.

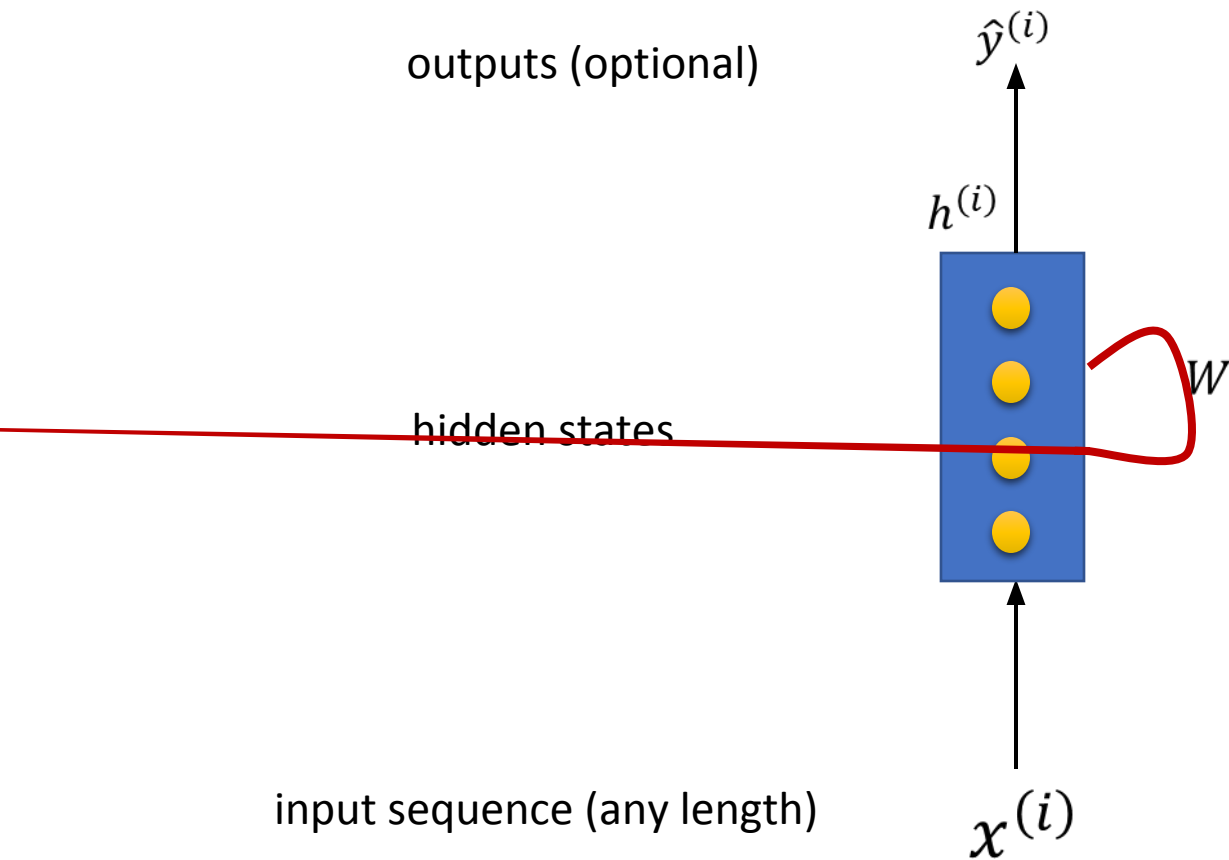
We need a neural architecture that can process *any length input*

as the invigilator started  
the clock, the students  
opened their \_\_\_\_\_



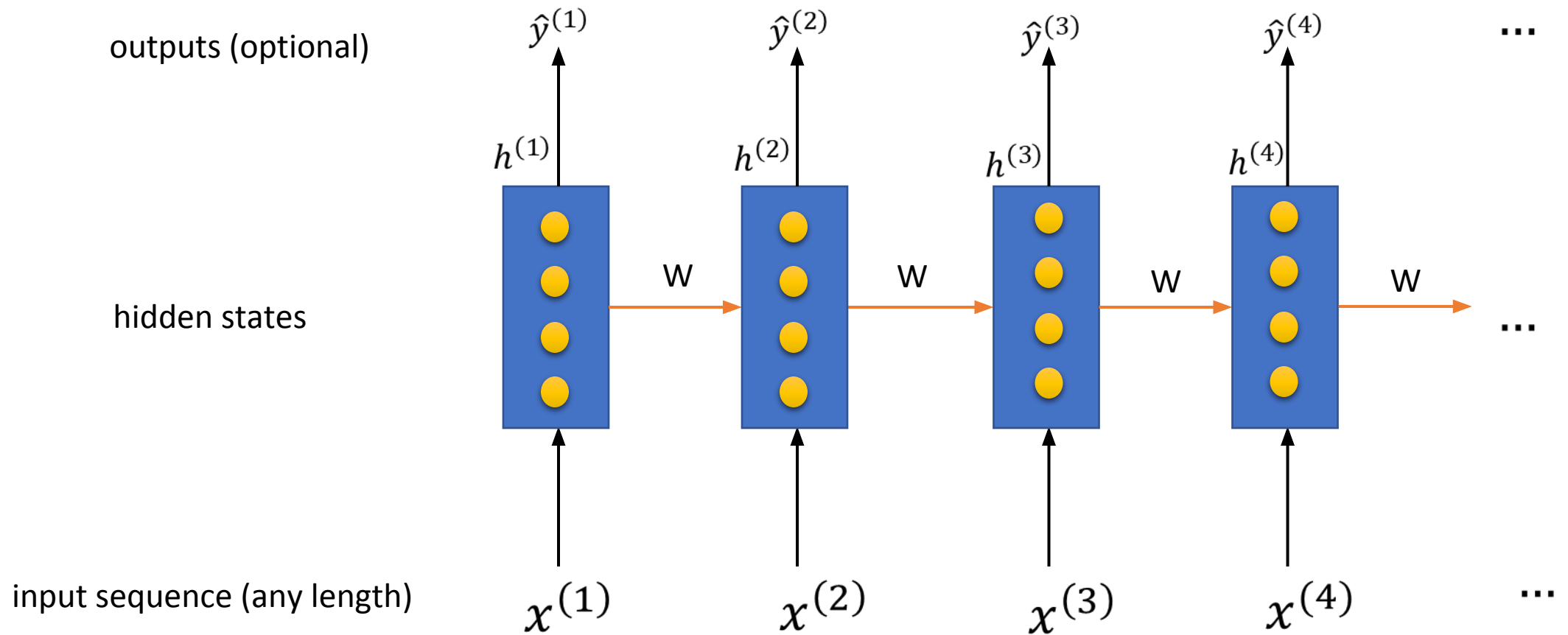
# Recurrent Neural Networks (RNN)

Core idea: Apply the same weights  $W$  repeatedly



# Recurrent Neural Networks (RNN)

Core idea: Apply the same weights  $W$  repeatedly



# A Simple RNN Language Model

output distribution

$$\hat{\mathbf{y}}^{(t)} = \text{softmax} \left( \mathbf{U} \mathbf{h}^{(t)} + \mathbf{b}_2 \right) \in \mathbb{R}^{|V|}$$

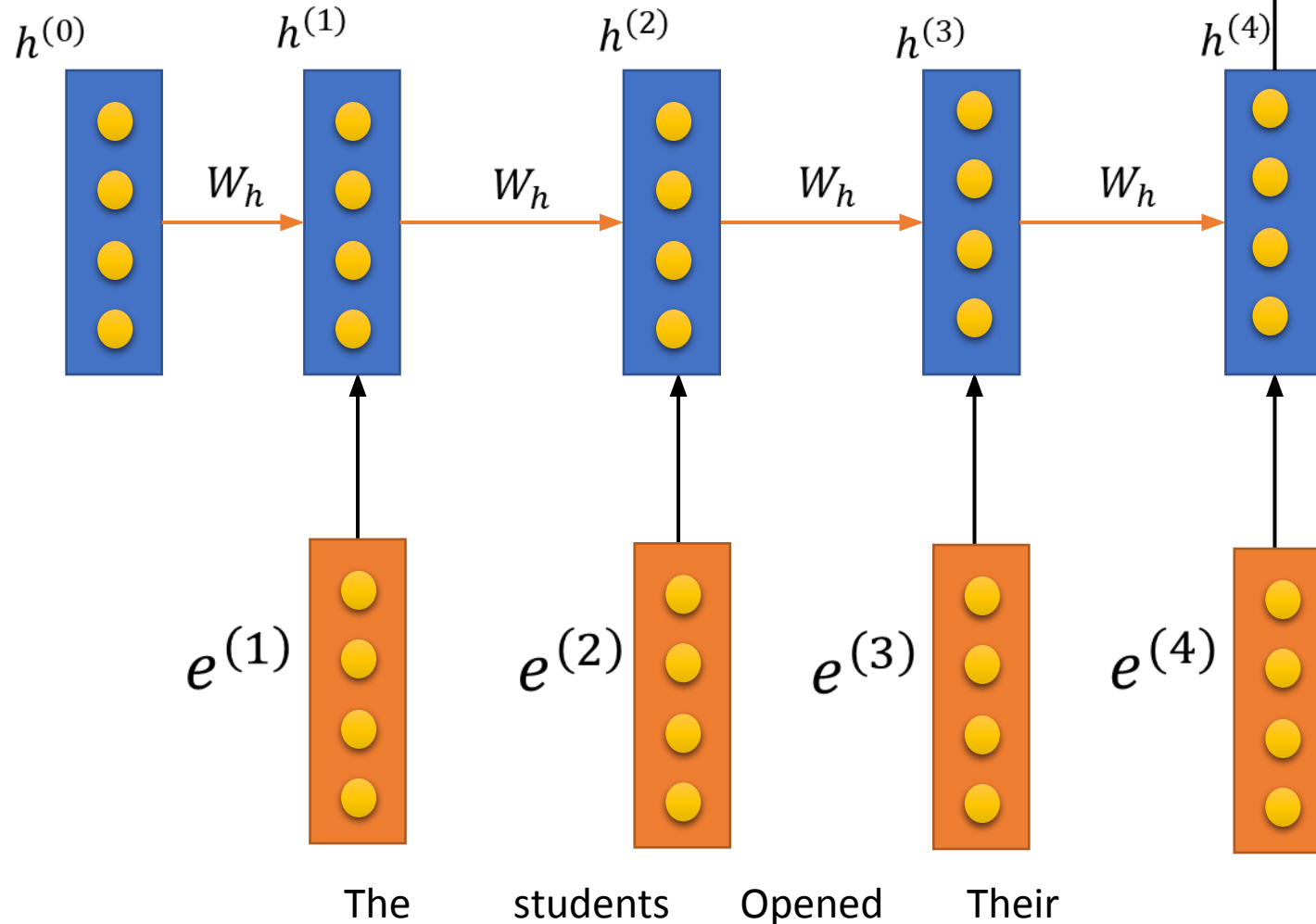
hidden states

$$\mathbf{h}^{(t)} = \sigma \left( \mathbf{W}_h \mathbf{h}^{(t-1)} + \mathbf{W}_e \mathbf{e}^{(t)} + \mathbf{b}_1 \right)$$

$\mathbf{h}^{(0)}$  is the initial hidden state

word embeddings  $\mathbf{e}^{(t)}$

words  $\mathbf{x}^{(t)}$





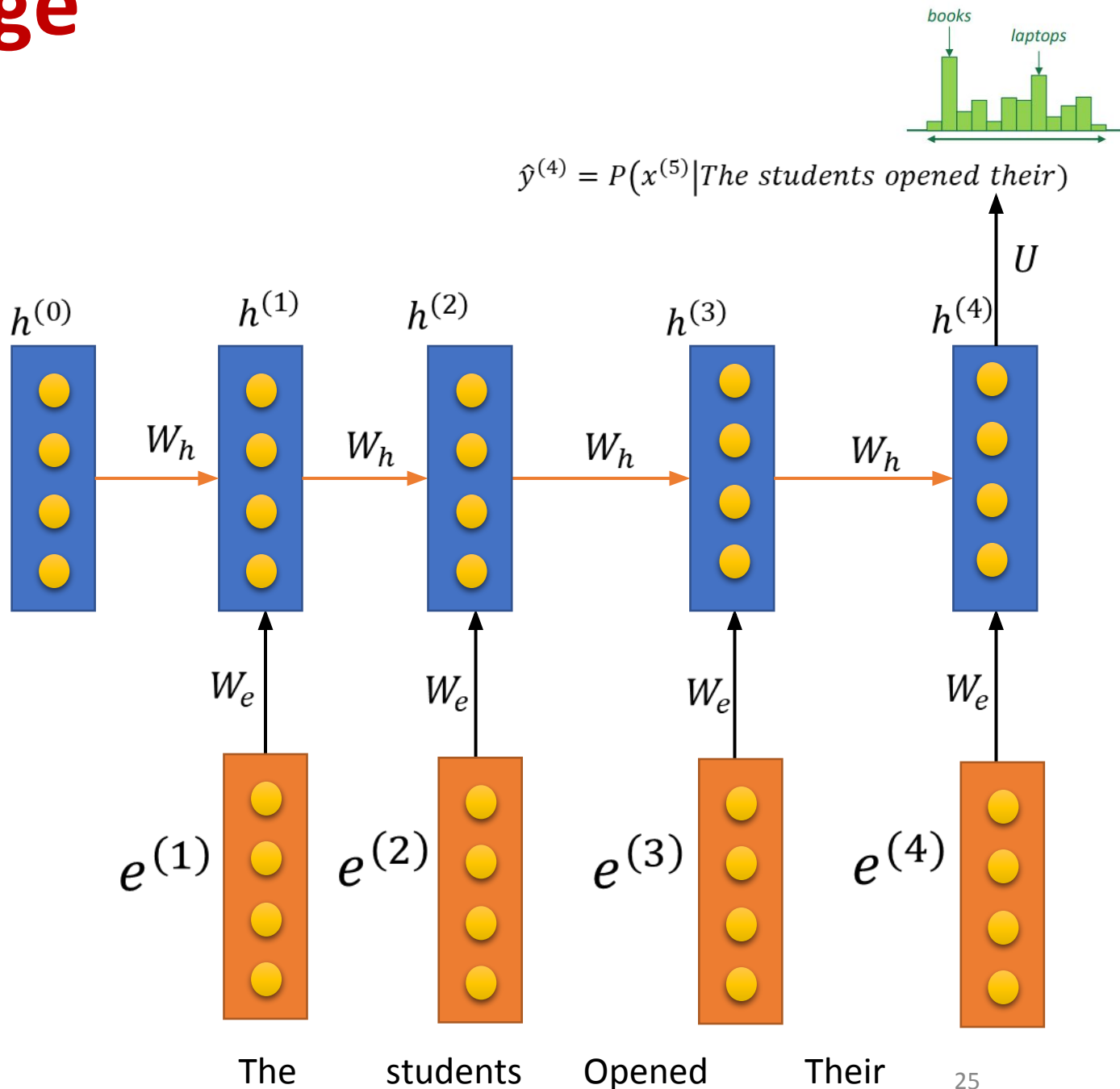
# A Simple RNN Language Model

## 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
- Same weights applied on every timestep, so there is symmetry in how inputs are processed.

## Limitations:

- Recurrent computation is slow
- In practice, difficult to access information from many steps back



# Training an RNN Language Model

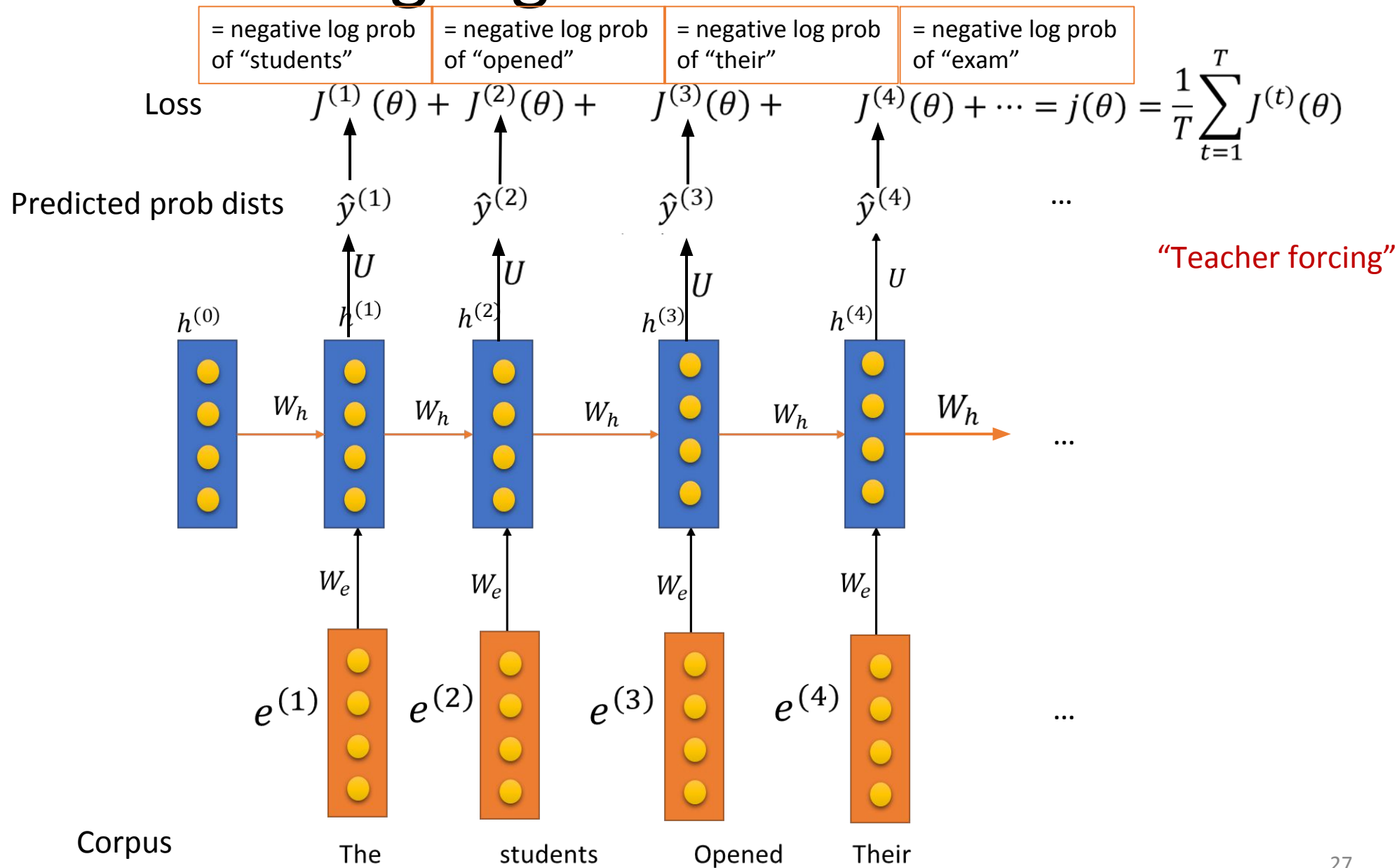
- Get a big corpus of text which is a sequence of words  $x^{(1)}, x^{(2)}, \dots, x^{(T)}$
- Feed into RNN-LM; compute output distribution  $\hat{y}^t$  for every step  $t$ .
  - i.e. predict probability dist of every word, given words so far
- Loss function on step  $t$  is cross-entropy between predicted probability distribution  $\hat{y}^t$ , and the true next word  $y^t$  (one-hot for  $x^{(t+1)}$ ):

$$J^{(t)}(\theta) = CE(\mathbf{y}^{(t)}, \hat{\mathbf{y}}^{(t)}) = - \sum_{w \in V} \mathbf{y}_w^{(t)} \log \hat{\mathbf{y}}_w^{(t)} = - \log \hat{\mathbf{y}}_{\mathbf{x}_{t+1}}^{(t)}$$

- Average this to get overall loss for entire training set:

$$J(\theta) = \frac{1}{T} \sum_{t=1}^T J^{(t)}(\theta) = \frac{1}{T} \sum_{t=1}^T - \log \hat{\mathbf{y}}_{\mathbf{x}_{t+1}}^{(t)}$$

# Training an RNN-Language Model



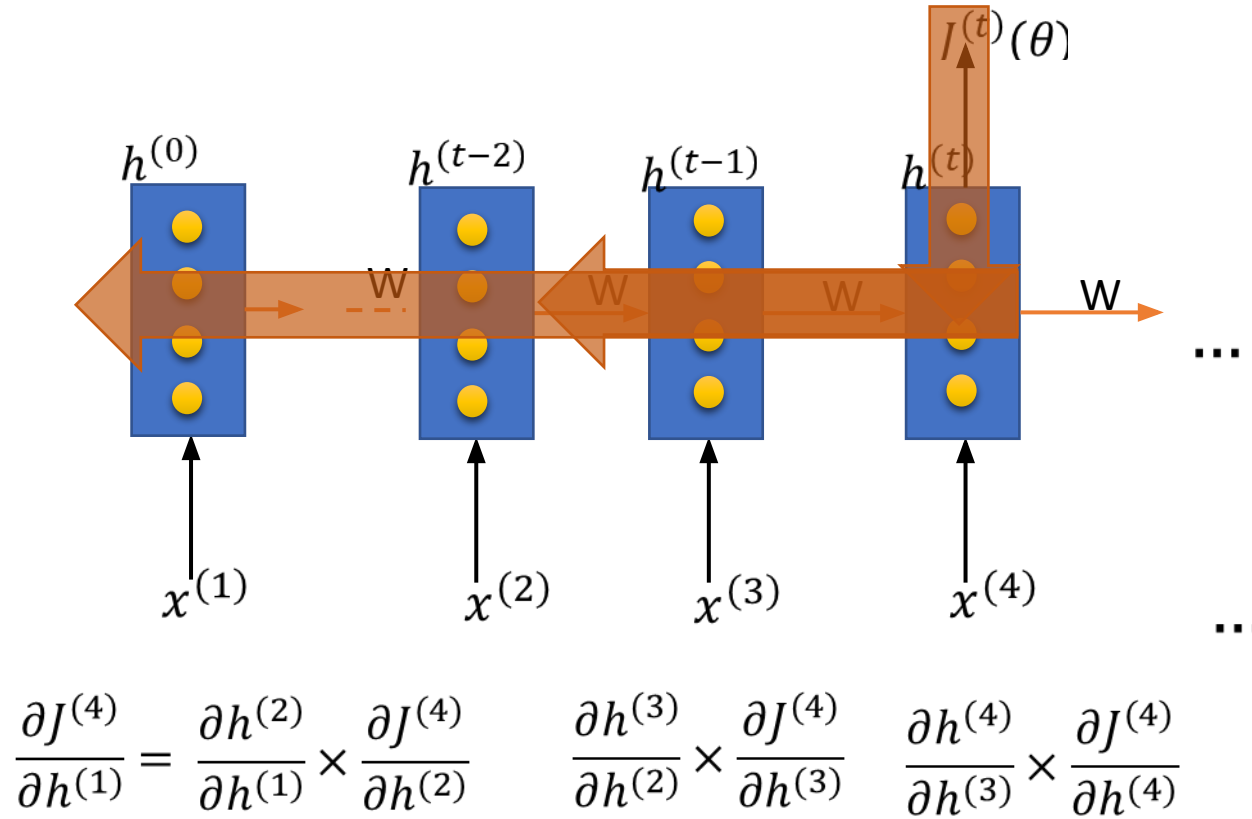
# Training a RNN Language Model

- However: Computing loss and gradients across entire corpus  $x^{(1)}, \dots, x^{(T)}$  is too expensive!

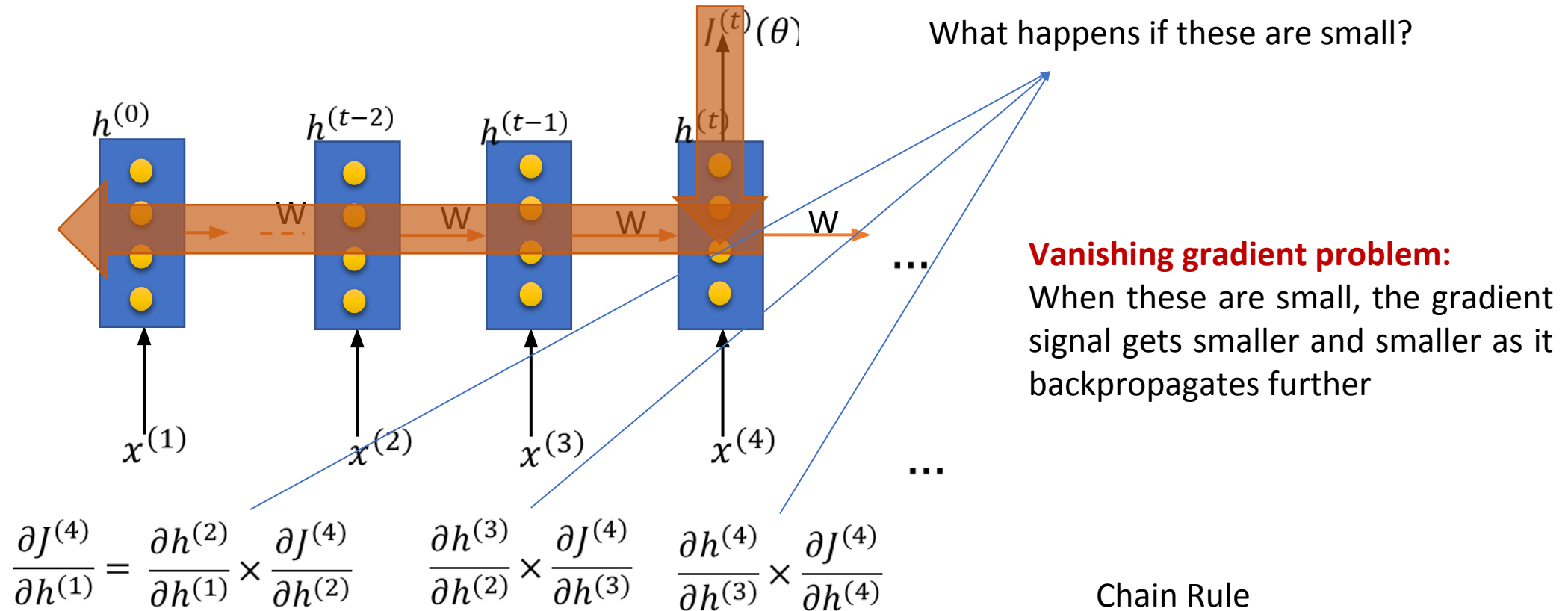
$$j(\theta) = \frac{1}{T} \sum_{t=1}^T J^{(t)}(\theta)$$

- In practice, consider  $x^{(1)}, \dots, x^{(T)}$  as a sentence (or a document)
- Recall: Stochastic Gradient Descent (SGD) allows us to compute loss and gradients for small chunk of data, and update.
- Compute loss  $J(\theta)$  for a sentence (actually a batch of sentences), compute gradients and update weights. Repeat.

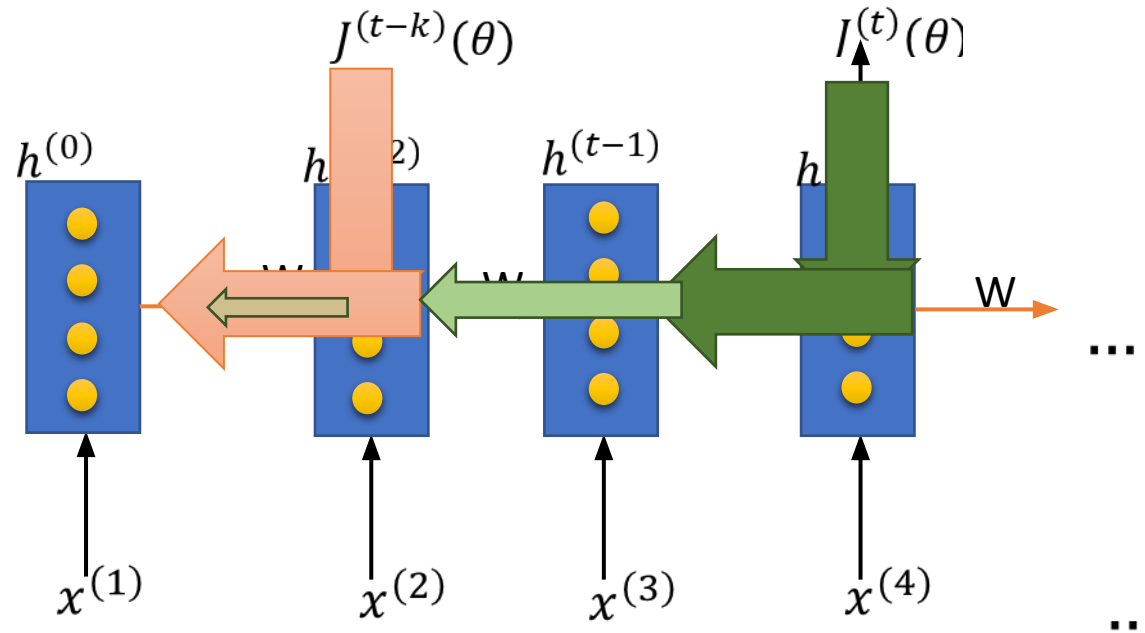
# Vanishing gradient intuition



# Vanishing gradient intuition



# Why is vanishing gradient a problem?



## Vanishing gradient problem:

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

- Gradient signal from faraway is lost because it's much smaller than gradient signal from close-by.
- So model weights are updated only with respect to near effects, not long-term effects.

# Why is vanishing gradient a problem?

- 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  $t+n$



# Effect of vanishing gradient on RNN-LM

- LM task: The writer of the books \_\_\_\_ is or are
- Sequential recency: The writer of the books are
- Syntactic recency: The writer of the books is
- Vanishing gradient problems may bias RNN-LMs towards learning from sequential recency, so they make this type of error more often than we'd like. [Linzen et al 2016]
- **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 \_\_\_\_\_*

# 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)

# Solution: Gradient clipping

- 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

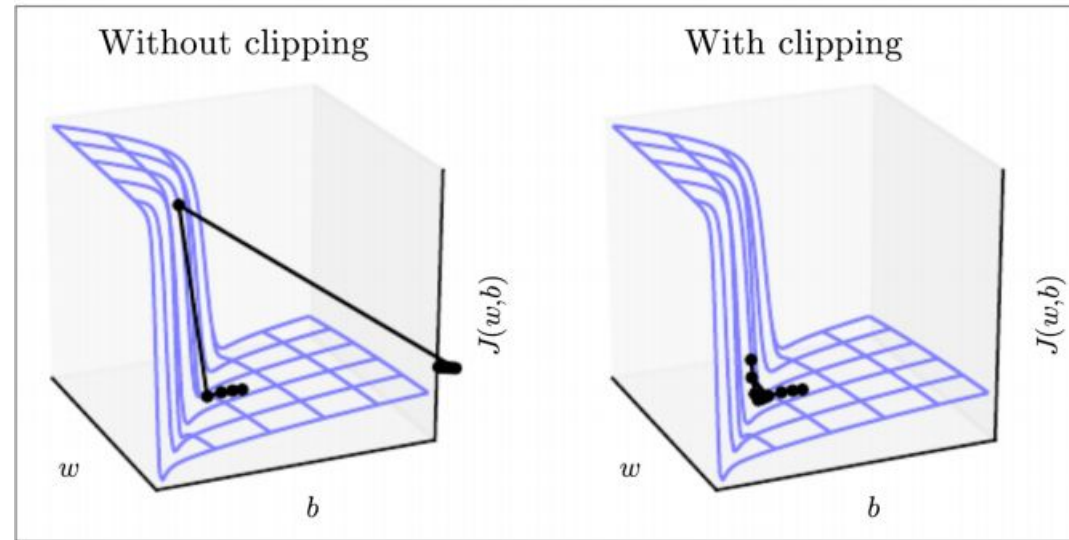
---

$$\begin{aligned} \hat{\mathbf{g}} &\leftarrow \frac{\partial \mathcal{E}}{\partial \theta} \\ \text{if } \|\hat{\mathbf{g}}\| &\geq \textit{threshold} \text{ then} \\ &\quad \hat{\mathbf{g}} \leftarrow \frac{\textit{threshold}}{\|\hat{\mathbf{g}}\|} \hat{\mathbf{g}} \\ \text{end if} \end{aligned}$$

---

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

# Gradient clipping



- 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

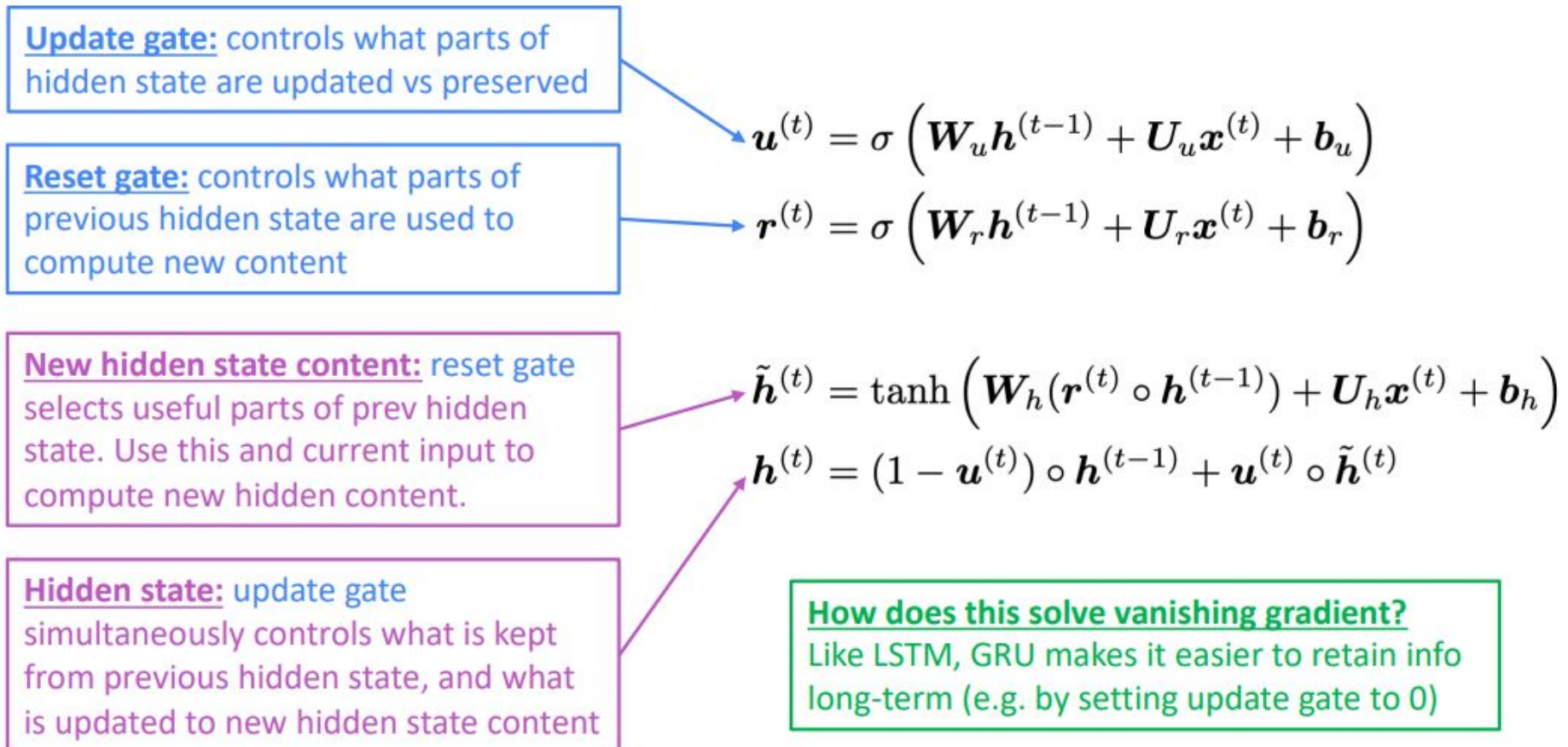
# How to fix vanishing gradient problem?

- 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
- How about a RNN with separate memory?

# Gated Recurrent Unit (GRU)

# Gated Recurrent Units (GRU)

- Proposed by Cho et al. in 2014 as a simpler alternative to the LSTM.
  - On each timestep  $t$  we have input and hidden state (no cell state).

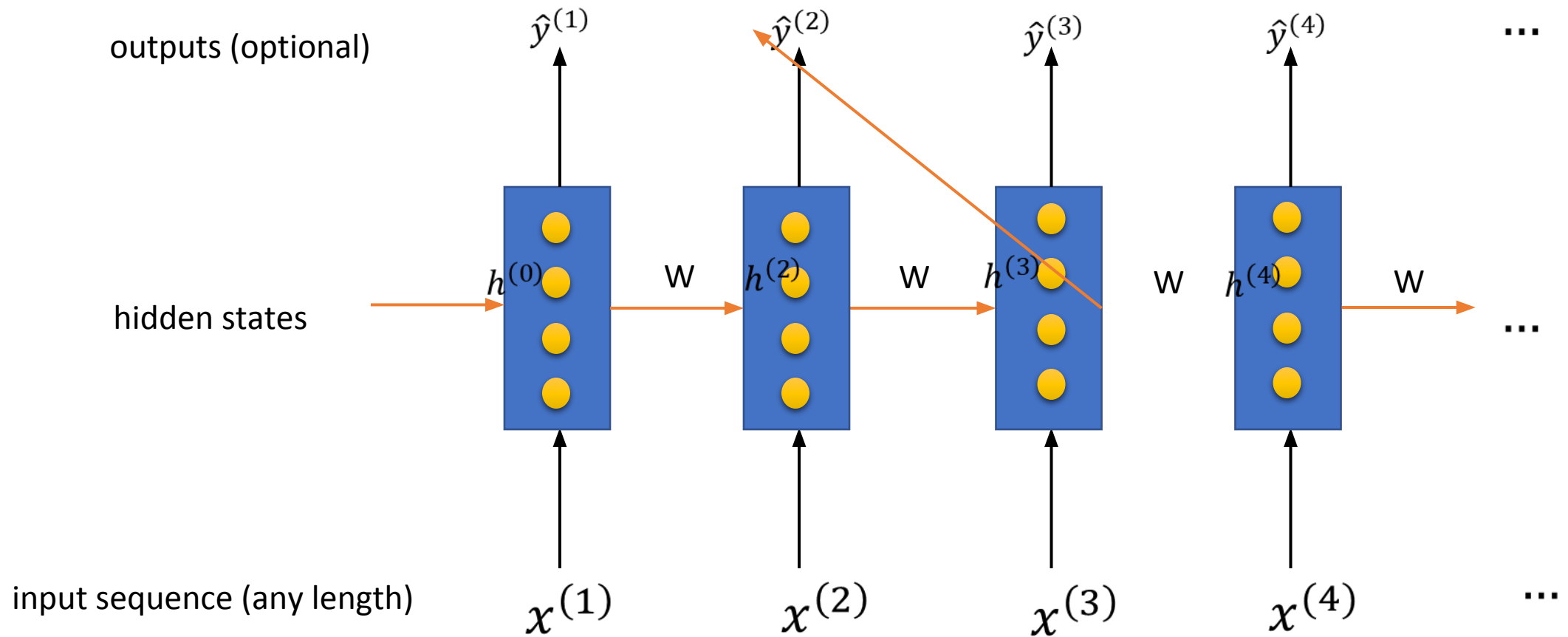


# Long Short-Term Memory (LSTM)

- A type of RNN proposed in 1997 as a solution to the vanishing gradients problem.
- On step  $t$ , there is a hidden state  $h^t$  and a cell state  $c^t$
- Both are vectors length  $n$ 
  - The cell stores long-term information
  - 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
  - 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.
  - The gates are dynamic: their value is computed based on the current context

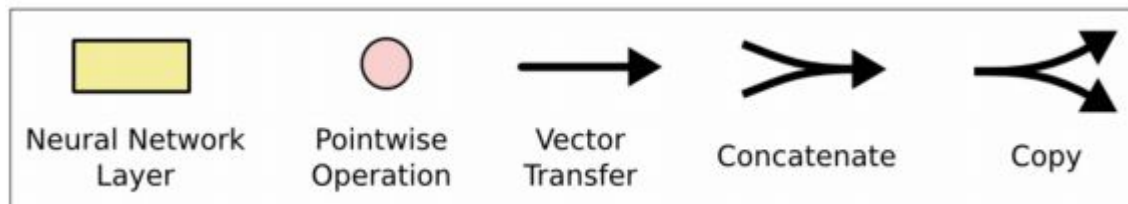
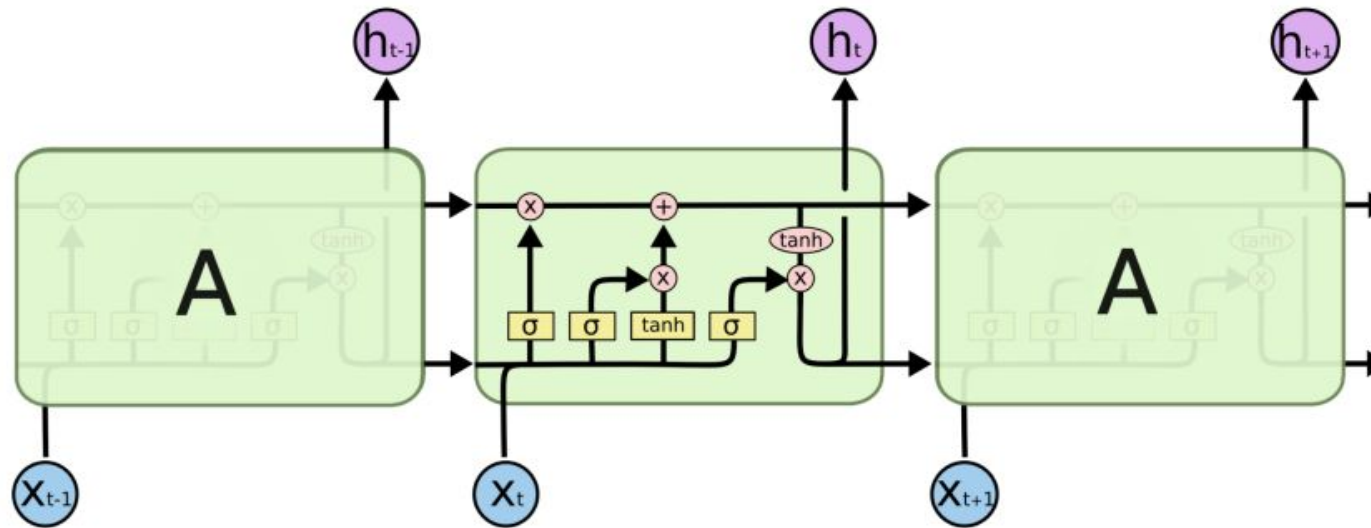


# LSTM Intuition

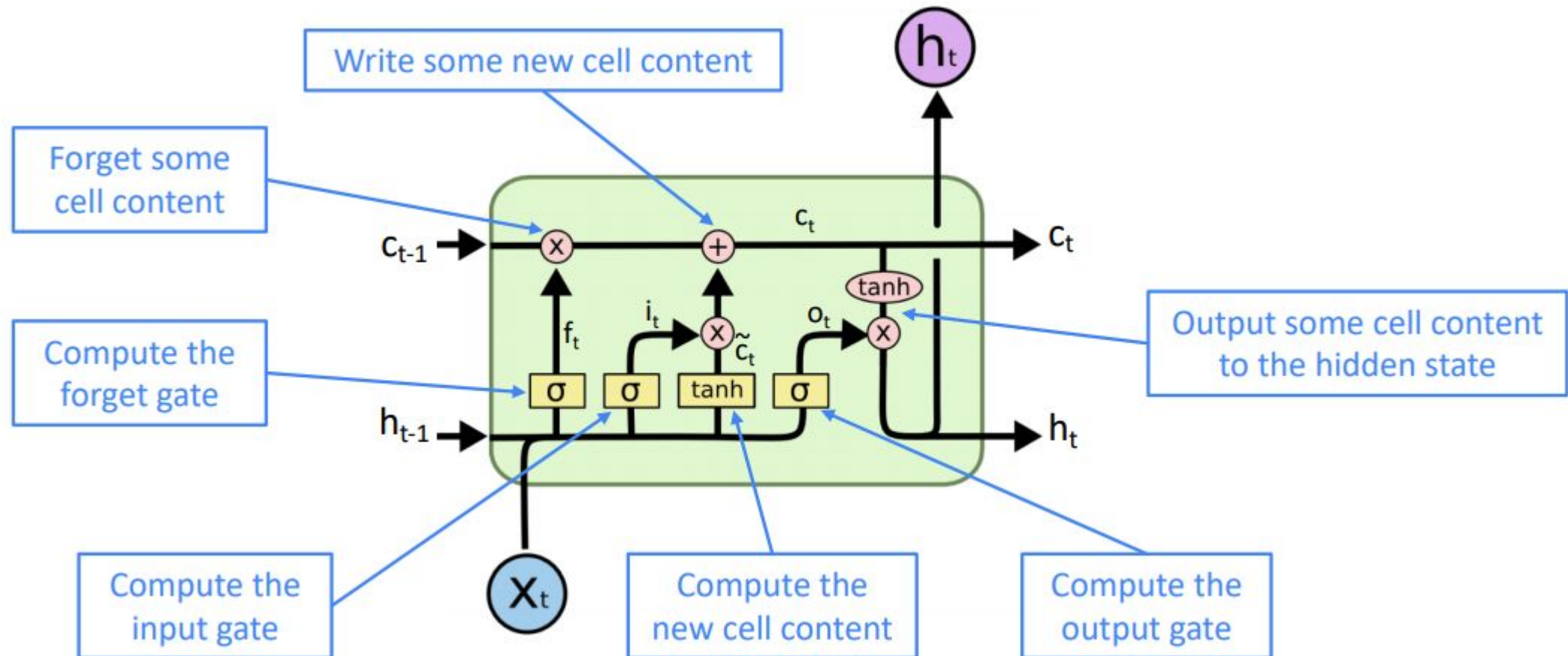


# Long Short-Term Memory (LSTM)

- You can think of the LSTM equations visually like this:



# Long Short-Term Memory (LSTM)



# How does LSTM solve vanishing gradients?

- The LSTM architecture makes it easier for the 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.
  - 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

# LSTM vs GRU

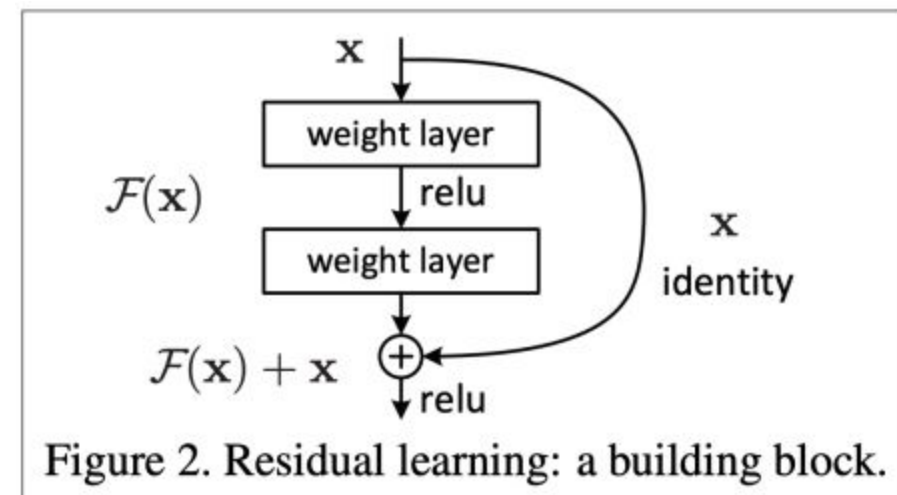
- Researchers have proposed many gated RNN variants, but LSTM and GRU are the most widely-used
- **Rule of thumb:** LSTM is a good default choice (especially if your data has particularly long dependencies, or you have lots of training data); Switch to GRUs for speed and fewer parameters.

# 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)

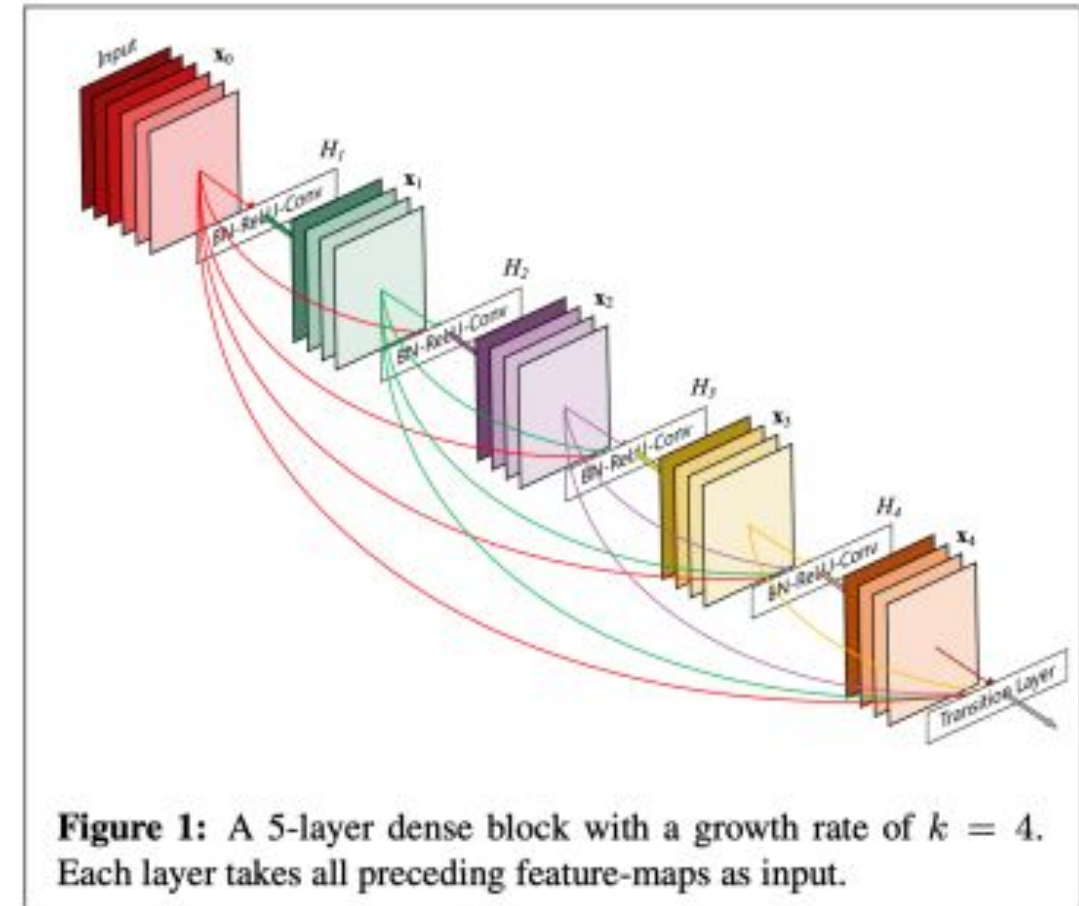
# Is vanishing/exploding gradient just a RNN problem?

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



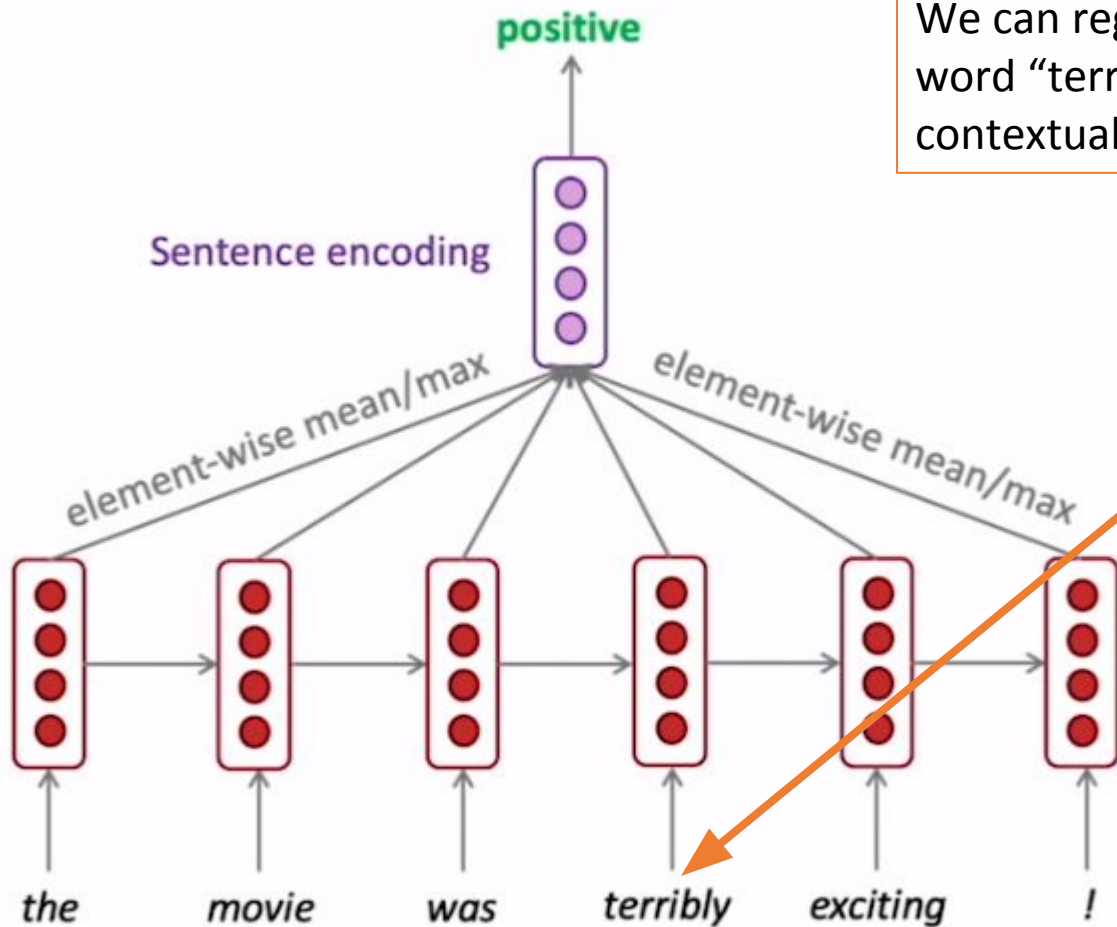
# Is vanishing/exploding gradient just a RNN problem?

- **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





# Bi-Directional RNN



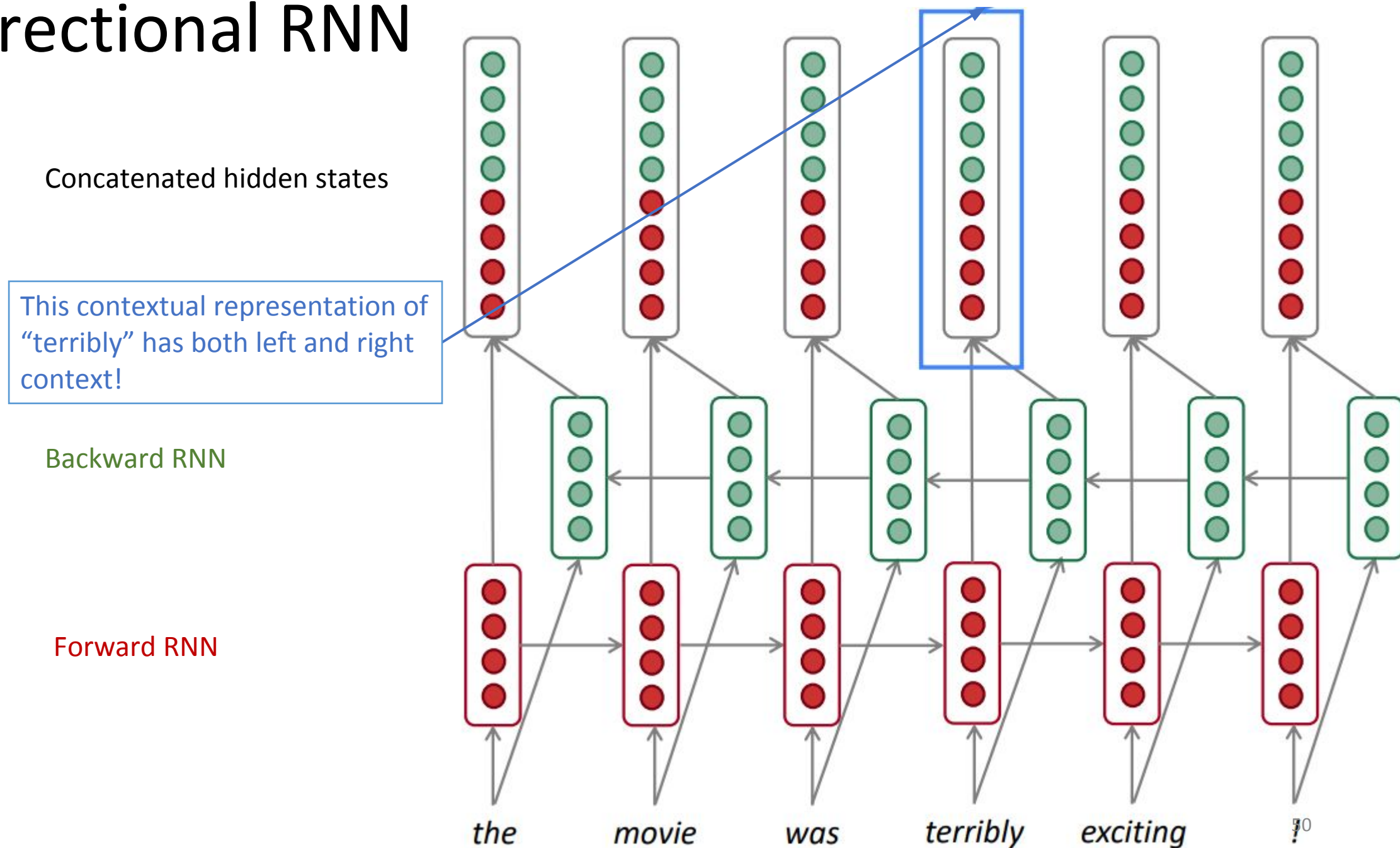
We can regard this hidden state as a representation of the word “terribly” in the context of this sentence. We call this a contextual representation.

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)

# Bi-Directional RNN



# Bidirectional RNNs

On timestep  $t$ :

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

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

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

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

Generally, these two RNNs have separate weights

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

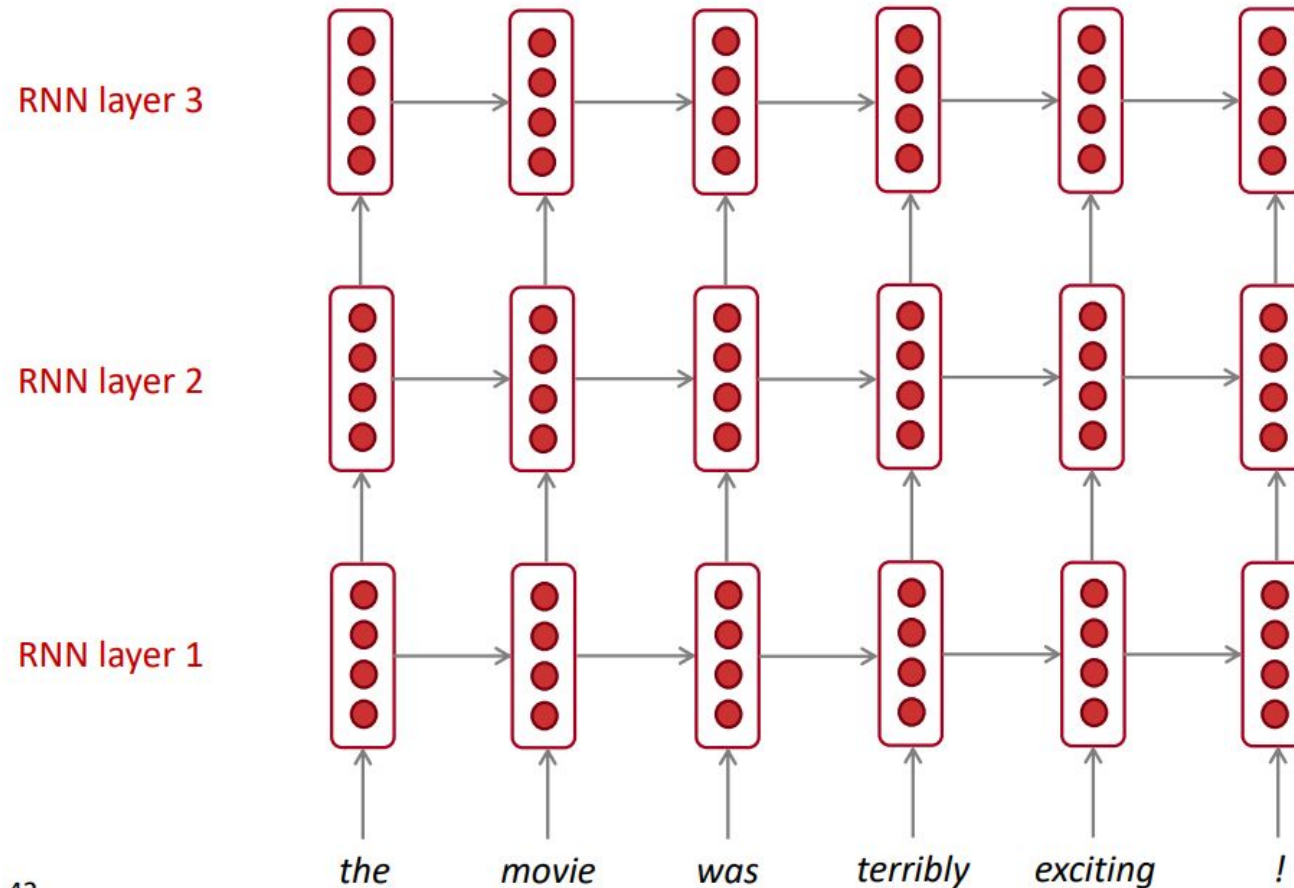
- Only applicable if you have access to the entire input sequence.
- They are not applicable to Language Modeling, because in LM you only have left context available.
- If you do have entire input sequence (e.g. any kind of encoding), bidirectionality is powerful (you should use it by default).
- For example, BERT (Bidirectional Encoder Representations from Transformers) is a powerful pretrained contextual representation system built on bidirectionality. • You will learn more about 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
- 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  $i+1$



# Types of RNN

1-to-1: single input and single output	1-to-N: single input and sequence output
	
A RNN models sequential interactions through a hidden state, or memory. It can take up to N inputs and produce up to N outputs.	An input could be a single image, and the output could be a sequence of words corresponding to the description of the image.
N-to-1: sequence input and single output	N-to-N: sequence input and sequence output
	
An input could be a sentence, and the output is the sentiment classification of the sentence	An input sequence may be a sentence with the outputs being the part-of-speech tag for each word

- Each rectangle represents a feature vector